2024 PDF VERSION

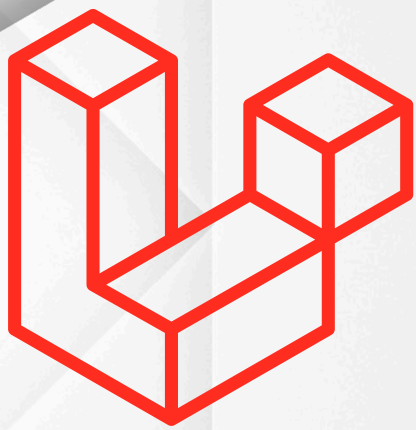
LARAVEL FRAMEWORK 11

ฉบับปรับปรุงใหม่ภาษาไทย

ตุลาคม 2567

ราคา

450 บาท



“พัฒนาเว็บไซต์ พัฒนาระบบ อย่างเข้าใจ ด้วย Framework ยอดนิยม”

นครินทร์ ม่วงอ่อน

ประสบการณ์การเขียน **Laravel Framework** กว่า 10 ปี

สนับสนุนโดย **goodlydev.com**

ปฐมบท

หนังสือเล่มนี้ผมตั้งใจเขียนขึ้น เนื่องจากว่า ผมมีความชื่นชอบรูปแบบการเขียนของ Laravel Framework ซึ่งมีความเรียบง่ายในการใช้งานเป็นอย่างมาก พยายามศึกษาหาข้อมูลจากสื่อต่างๆ ทั้งในรูปแบบ Ebook และ VDO จนตัวเองคิดว่า น่าจะมีหนังสือที่เป็นภาษาไทยสักเล่มหนึ่งที่รวบรวมเรื่องต่างๆ เกี่ยวกับ การเขียน Laravel Framework แบบที่ว่ามีตั้งแต่พื้นฐาน (พื้นฐานของการใช้ Laravel Framework) จนถึงระดับ Advanced ครบวงจรในเล่มเดียว

หนังสือเล่มนี้จะเป็นหนังสือที่รวบรวมสิ่งต่างๆที่ควรรู้เกี่ยวกับการใช้งาน Laravel Framework ก็จะเริ่มตั้งแต่เริ่มต้นทำความรู้จักกับ Laravel Framework ไปจนถึงการสร้าง Workshop สำหรับใช้งานจริง ในแต่ละหัวข้อผมจะพยายามใช้ภาษาที่เขียนออกมาแล้วให้ท่านผู้อ่านเข้าใจง่ายได้มากที่สุด ผมเองก็อยากจะมีหนังสือดีๆ สักเล่มหนึ่งเก็บไว้ให้รุ่นน้อง ๆ ลูก ๆ หลาน ๆ ได้อ่าน ได้ศึกษา เผื่ออนาคต Laravel Framework จะเป็น Framework ที่ใครๆ ก็อยากจะเรียนรู้ เพราะว่าเรียบง่ายตามแบบของมัน

ผมต้องออกตัวก่อนนะครับ ว่าผมไม่ใช่ กูรู ไม่ใช่เทพ จุติ องค์ใด ไม่ใช่ผู้วิเศษใด เพียงแต่ว่าผมมีความตั้งใจมากๆ ที่จะเขียนหนังสือเล่มนี้ออกมา อาจจะไม่ใช่เล่มที่ดีที่สุดสำหรับท่านผู้อ่าน แต่มันจะเป็นหนังสือที่จะแนะนำหรือนำทางให้ท่านที่สนใจในการเขียน Laravel Framework เพื่อที่จะพัฒนาเว็บ Application ในรูปแบบที่ท่านต้องการได้ ไม่ว่าจะเป็น ระบบขนาดเล็กหรือขนาดใหญ่

หนังสือเล่มนี้จะเป็นการใช้งาน Laravel 11 ซึ่งเป็น Version ล่าสุดในตอนที่ผมเขียนหนังสือเล่มนี้ และใช้ PHP >=8.2.0 ถ้า PHP Version ต่ำกว่านี้อาจจะติดตั้ง Laravel 11 ไม่ได้

เนื่องจากว่าหนังสือเล่มนี้เป็นเล่มที่เขียนต่อจาก Laravel 10 เล่มก่อน ฉะนั้นรูปแบบการเขียนจะคล้าย ๆ กันทั้งหมด ผู้อ่านไม่ต้องตกใจนะครับว่าทำไมเนื้อหาบางหน้าเป็น Laravel9 หรือ Laravel 10 อยู่

เริ่มต้นวันนี้ไม่ไหว ก็ไม่ต้องเริ่ม วันพรุ่งนี้ค่อยเริ่มใหม่ ก็คงไม่สาย

เข้าร่วมกลุ่ม Facebook (แนะนำ)

https://www.facebook.com/groups/laravelthailand/

ดาวน์โหลด source code : https://github.com/leksoft/laravel9

เริ่มจำหน่าย 11 มกราคม 2566

เวอร์ชัน 1 จัดส่งวันที่ 1 กุมภาพันธ์ 2566
เวอร์ชัน 2 จัดส่งวันที่ 26 กุมภาพันธ์ 2566
เวอร์ชัน 3 จัดส่งวันที่ 20 มีนาคม 2566 (Laravel 10)
เวอร์ชัน 4 จัดส่งวันที่ 20 ตุลาคม 2567 (Laravel 11)

กลุ่มเป้าหมาย

• โปรแกรมเมอร์และนักพัฒนา Web Application ทั่วไป

• ฝ่ายไอที ผู้ดูแลและพัฒนาระบบขององค์กร

• นักเรียน / นักศึกษา / ผู้สนใจทั่วไป

ความรู้พื้นฐานที่ผู้อ่านควรต้องมี

1. มีความรู้การเขียนเว็บโปรแกรมมิ่งด้วยภาษา PHP MySQL และ HTML CSS มาก่อนเล็กน้อยไม่ต้องเยอะ
2. เข้าใจหลักการของ OOP มาบ้างแล้วไม่ต้องเยอะก็ได้แค่เข้าใจหลักการ
3. ถ้าเคยใช้ Framework ตัวอื่น ๆ มาก่อนจะทำความเข้าใจได้ง่ายยิ่งขึ้น
4. ความรู้พื้นฐานเกี่ยวกับการใช้โปรแกรมแก้ไขโค้ด เช่น Visual Studio Code หรืออื่นๆ ตามถนัด

สิ่งที่คุณจะได้เรียนรู้

1. Laravel 11 พื้นฐานตั้งแต่เริ่มต้น
2. Laravel Single and Multi Auth (with Guard) System
3. Laravel CRUD
4. ระบบ Login
5. ระบบลืมรหัสผ่านและรีเซ็ตรหัสผ่าน
6. Contact form email sending using Ajax
7. Newsletter email sending using Ajax and email verification
8. ตั้งค่าเว็บไซต์ logo, favicon
9. เว็บไซต์หลายภาษา
10. ระบบการยืนยันอีเมล
11. Website Color Settings Option
12. Admin and author profile managements
13. ระบบอัพโหลดรูปภาพ
14. Home page search option using chained select with jquery
15. Security for post section in author section
16. การใช้งานข้อความแจ้งเตือนด้วย Toastr message notification
17. การส่งอีเมลในเว็บเซิร์ฟเวอร์
18. อัพโหลดโปรเจ็คขึ้นเว็บเซิร์ฟเวอร์เพื่อใช้งานจริง
19. มี Workshop แต่ละหัวข้อให้ทำตาม

สารบัญ

เนื้อหา หน้า

Laravel Fundamentals – Laravel 11

XAMPP คืออะไร....1
ติดตั้ง XAMPP....2
Extensions ของ PHP ที่จำเป็นต้องเปิดใช้งานสำหรับใช้งาน Laravel 11....7
Visual Studio Code คืออะไร....9
ติดตั้ง Visual Studio Code....10
ติดตั้ง Extension สำหรับ Visual Studio Code....12
Laravel Framework คืออะไร....14
MVC คืออะไร....17
Composer คืออะไร....20
ติดตั้ง Composer....22
ติดตั้ง Laravel 11....27
แนะนำการ Start Project ผ่าน Visual Studio Code....31
โครงสร้างไฟล์ใน Laravel Framework....34
ตั้งค่าต่าง ๆ ที่สำคัญก่อนเริ่มเขียน Laravel 11....37
Debugging ใน Laravel....41
พื้นฐานการเขียน Routes รูปแบบต่าง ๆ....47
กรอง HTTP request ด้วย Middleware....61
พื้นฐานการเขียน Controllers....65
Resource Controllers (CRUD) 70
พื้นฐานการเขียน Views....75
Blade Template....80
เชื่อมโยง Routes - Controller - View ใช้งานร่วมกัน....82
สร้าง Template หลัก (Master Layouts) 87
การใช้งาน Template หลัก....89
Control Structures....94
Database Migration And Seeding with Artisan....98
เพิ่มข้อมูลตั้งต้นด้วย Seeding....107
จำลองเนื้อหา ด้วย Factory....110
พื้นฐานการเขียน Models....114
เพิ่มข้อมูลลงตาราง....129

สารบัญ

เนื้อหา หน้า

Laravel Fundamentals – Laravel 11

อ่านข้อมูลจากตาราง....134
แก้ไขข้อมูลจากตาราง....136
ลบข้อมูลออกจากตาราง....140
Database – Query Builder (เพิ่มข้อมูล)....142
Database – Query Builder (อ่านข้อมูล)....147
Database – Query Builder (แก้ไขข้อมูล)....149
Database – Query Builder (ลบข้อมูล)....151
Database- ORM....152
Database – Eloquent ORM (เพิ่มข้อมูล)....153
Database – Eloquent ORM (อ่านข้อมูล)....156
Database – Eloquent ORM (แก้ไขข้อมูล)....157
Database – Eloquent ORM (ลบข้อมูล)....158
Eloquent ORM (กรองข้อมูลตามเงื่อนไขที่ต้องการ)....159
Eloquent ORM (แสดงรายละเอียดข้อมูลบทความแต่ละรายการ)....160
Pagination – การแบ่งหน้าเนื้อหา....162
การใช้งาน Query Scopes....166
การใช้งาน Accessors....168
การใช้งาน Mutators....169
กำหนดความสัมพันธ์ของตารางด้วย Eloquent Relationships....170
Database – Eloquent ORM (แบบ One to One Relationship)....170
Database – Eloquent ORM (แบบ One to One Reverse Relationship)....176
Database – Eloquent ORM (แบบ One to Many Relationship)....177
Database – Eloquent ORM (แบบ Many to Many Relationship)....180
สร้าง Web Forms ใน Laravel....189
Form Validation And Error Message....195

สารบัญ

เนื้อหา หน้า

Laravel Fundamentals – Laravel 11

แจ้งเตือนข้อผิดพลาดจากไฟล์ภายนอก....197
การอับโหลดไฟล์และบันทึกข้อมูลลงฐานข้อมูล....198
การแก้ไขข้อมูลด้วย Form Model Binding อย่างง่าย....201
ลบข้อมูลให้ปลอดภัยด้วย Laravel Collective....208
Soft Delete and Data Restore – การลบข้อมูลและการกู้คืน....210
Helper Functions....216
Sending Email....218
การใช้งาน Session....224
Flash Data....225
Toastr library....230
Sweet Alert Library....228
ระบบล็อกอินและระบบลงทะเบียน (Auth Login and Registration)....229
จัดการสิทธิ์การเข้าใช้งานเมนู....241
Role & Permission (spatie Library)....247
สร้าง API และ เรียกใช้งาน API....276
เรียกใช้งาน API ฝั่ง Frontend ด้วย Guzzle HTTP....287
สร้างรายงาน PDF ด้วย TCPDF library....289
เว็บไซต์หลายภาษา (Multi-Language)....294
Laravel Livewire CRUD ด้วย Bootstrap Modal....307
การใช้งาน Ckeditor 5 ร่วมกับ Laravel 11....322
การใช้งาน Ckeditor 5 กับการอับโหลดไฟล์....326
ระบบลืมรหัสผ่านและรีเซ็ตรหัสผ่าน....329
โหลดเนื้อหาเพิ่มเติมด้วยการคลิกปุ่ม....335
Laravel 11 Sanctum REST API Authentication....341

XAMPP คืออะไร

XAMPP คือ โปรแกรมสำหรับจำลองเครื่องคอมพิวเตอร์ส่วนบุคคลของเราให้ทำงานในลักษณะของ Web Server ซึ่งเครื่องคอมพิวเตอร์ของเราจะเป็นทั้งเครื่องแม่และเครื่องลูกในเครื่องเดียวกัน โดยไม่ต้องเชื่อมต่อกับ Internet ก็สามารถทดสอบกับเว็บไซต์ที่เราสร้างขึ้นมาได้ทุกที่ทุกเวลา อีกทั้งยังประหยัดเวลาและไม่มีค่าใช้จ่ายใดๆ ทั้งสิ้น ซึ่งในปัจจุบันนี้ได้รับความนิยมจากผู้ใช้ CMS ในการสร้างเว็บไซต์

XAMPP จะชุดโปรแกรมที่แถมมาให้โดยที่เราไม่ต้องไปหาโปรแกรมอื่นมาติดตั้งทีละโปรแกรม เราใช้ XAMPP ตัวเดียวจบเลย โดยจะประกอบด้วย Apache, PHP, MySQL, PHPMyAdmin, Perl ซึ่งเป็นโปรแกรมพื้นฐาน รองรับระบบปฏิบัติการ เช่น Windows, MacOS, Linux

XAMPP เมื่อแยกตัวอักษรแต่ละตัวจะประกอบด้วย

X = Cross Platform โปรแกรม XAMPP รองรับการรันบนระบบปฏิบัติการ Windows, MacOS, Linux

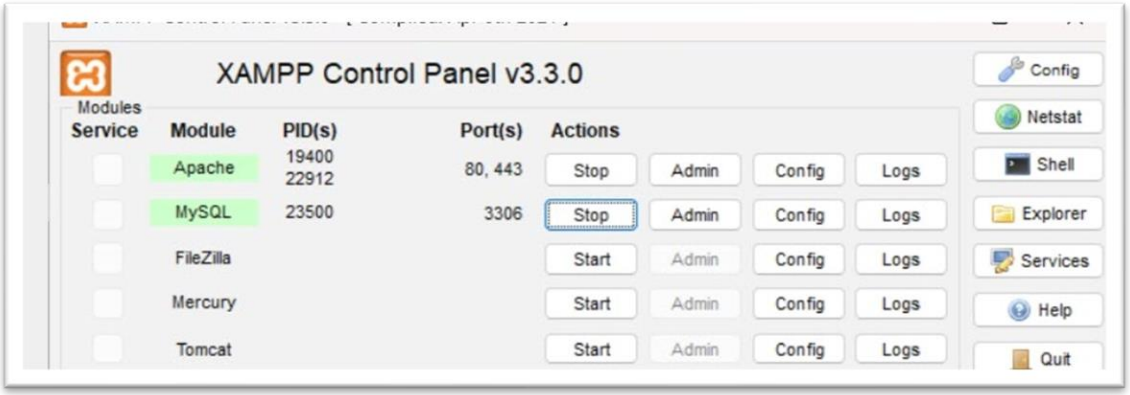
A = Apache Web Server ทำงานบนโปรโตคอล HTTP ทำหน้าที่ในการจัดเก็บ Homepage และส่ง Homepage ไปยัง Browser ซึ่งปัจจุบันจัดได้ว่าเป็น web server ที่มีความน่าเชื่อถือมาก เนื่องจากเป็นที่นิยมใช้กันทั่วโลก

M = MySQL (MariaDB) โปรแกรมฐานข้อมูล ใช้สำหรับบริหารจัดการฐานข้อมูล

P = PHP ภาษาคอมพิวเตอร์สำหรับพัฒนาเว็บไซต์ มีการประมวลผลทางฝั่ง Server

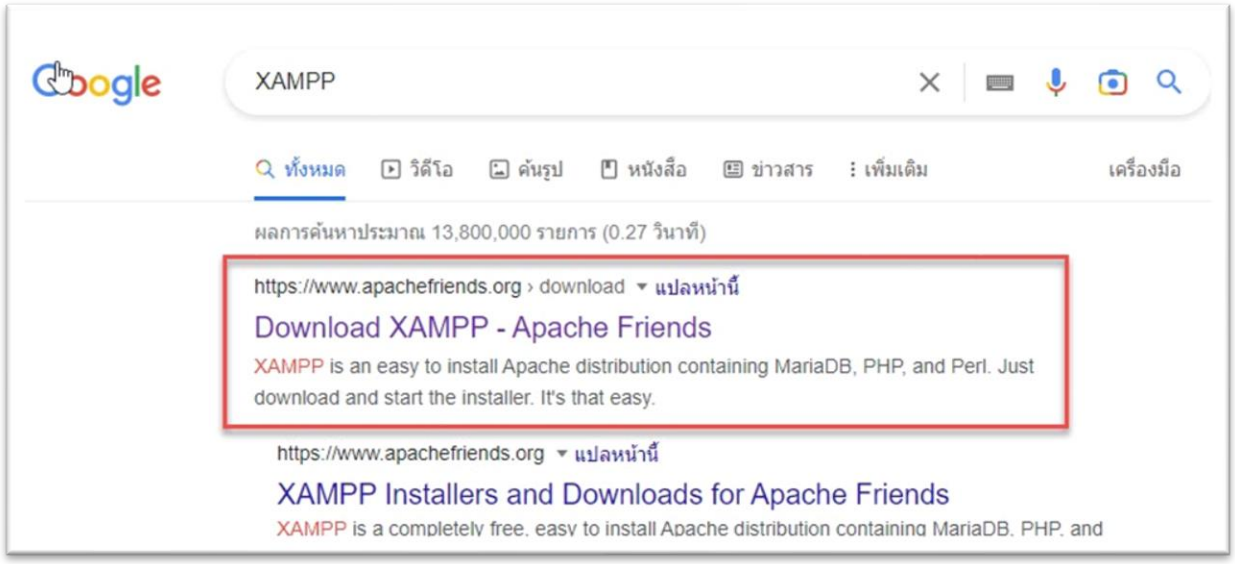
P = Perl เป็นภาษาคอมพิวเตอร์ รองรับการพัฒนาโปรแกรมในรูปแบบต่างๆ

เวอร์ชั่นที่ใช้ในขณะที่เขียนหนังสือเล่มนี้ คือ V.3.3.0 รองรับ PHP 8.2.0



ติดตั้ง XAMPP

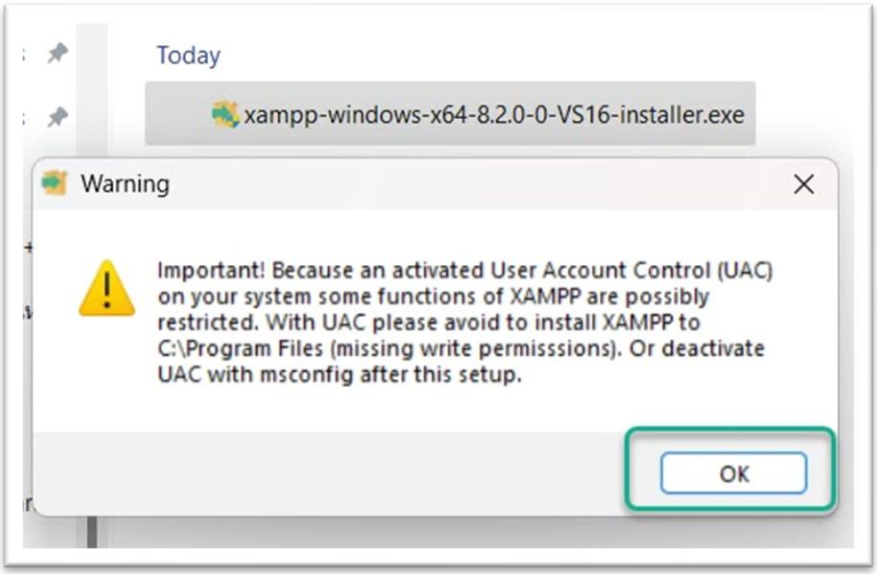
การติดตั้ง XAMPP นั้นง่ายมากครับ โดยสามารถเข้าไปดาวน์โหลดโปรแกรมได้ที่เว็บไซต์หลักของโปรแกรมได้เลยครับหรือจะค้นหา XAMPP จาก Google ได้เลยครับ โดยในที่นี้ผมจะเข้าไปลิงค์นี้ครับ https://www.apachefriends.org



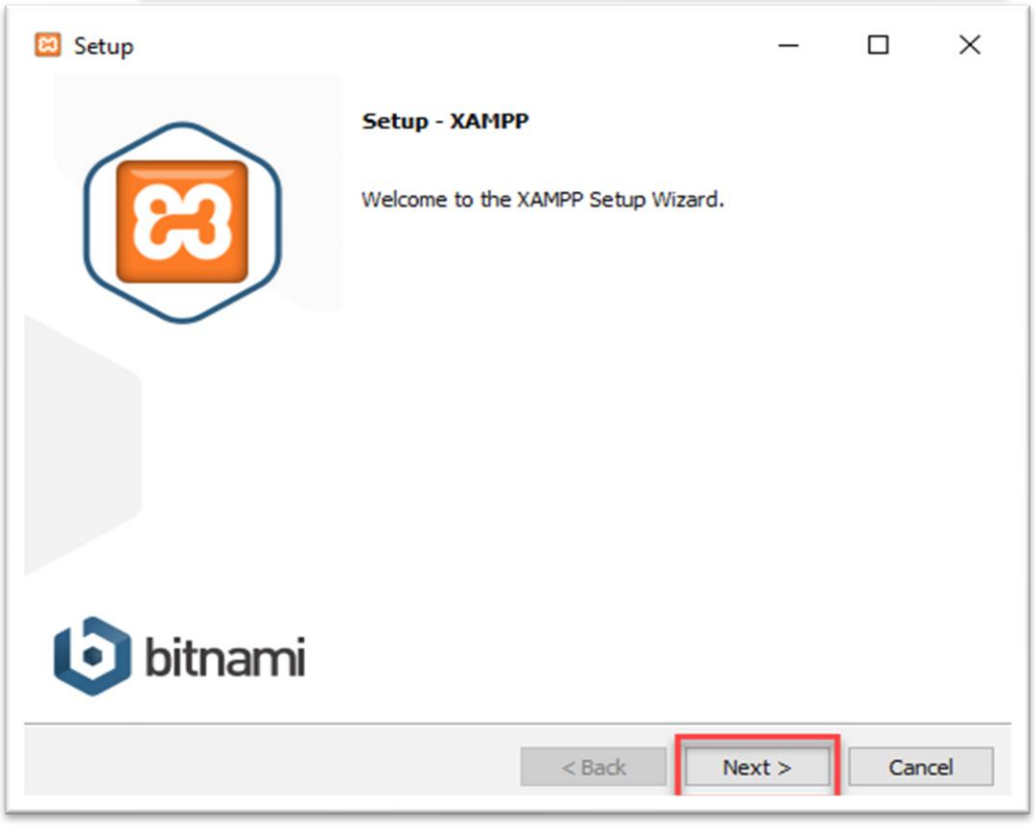
พอเข้ามาในเว็บแล้วก็จะเห็นคำว่า Download แล้วให้คลิกดาวน์โหลดโปรแกรมตาม OS ที่ใช้ได้เลยครับ โดยในหนังสือนี้ผมจะติดตั้งผ่าน Windows และใช้ XAMPP Version ล่าสุด คือ XAMPP for Windows 8.2.0 (PHP 8.2.0)



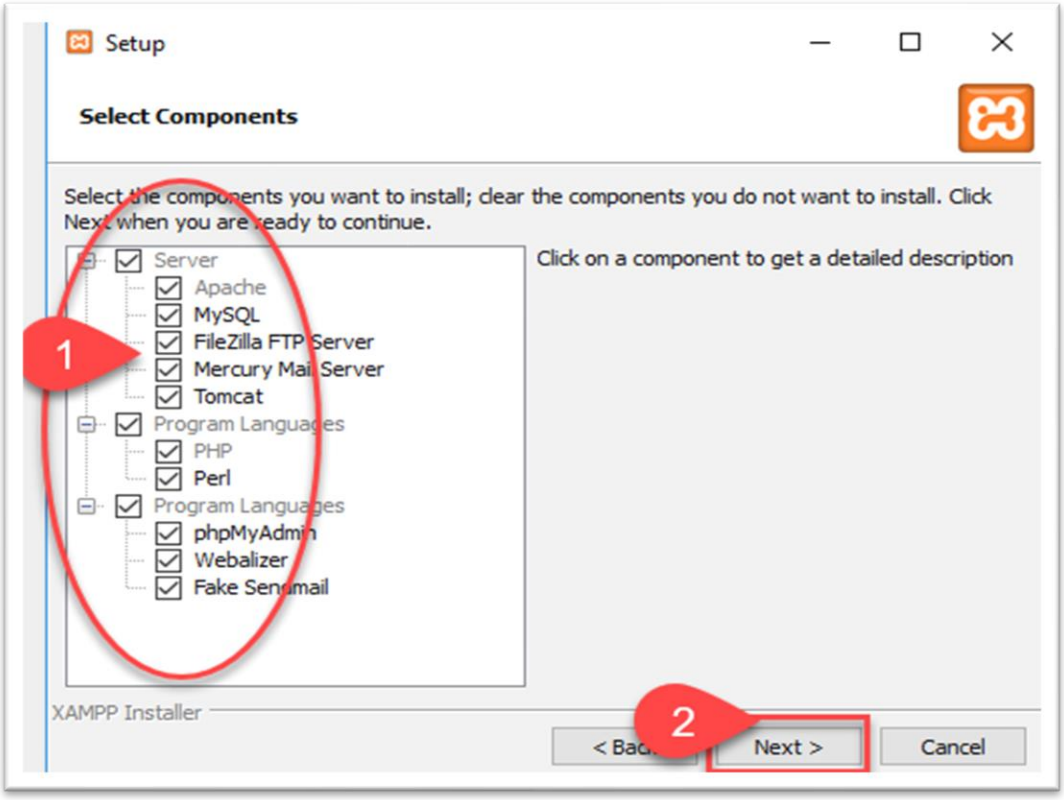
หลังจากที่ดาวน์โหลดไฟล์เสร็จแล้ว ให้ดับเบิลคลิกเพื่อทำการติดตั้งโปรแกรม จะมีกล่องข้อความแจ้งเตือน Use Account Control ขึ้นมาให้คลิกที่ปุ่ม Yes เพื่อยอมรับการติดตั้ง



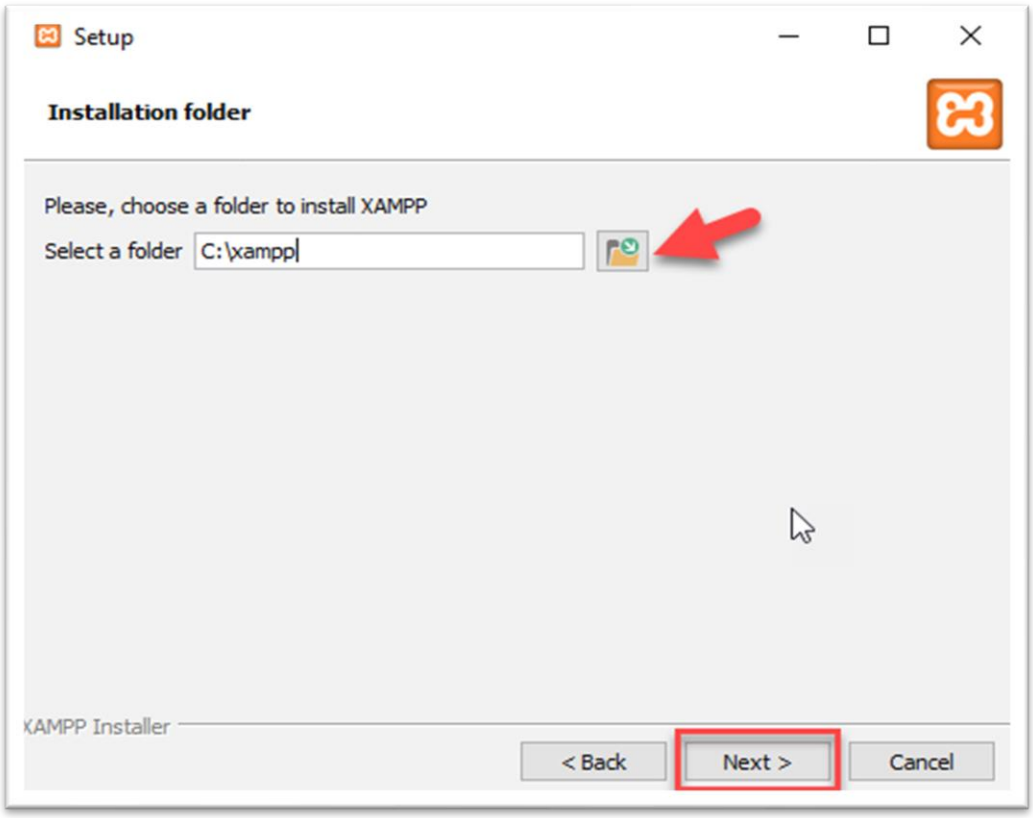
กด OK



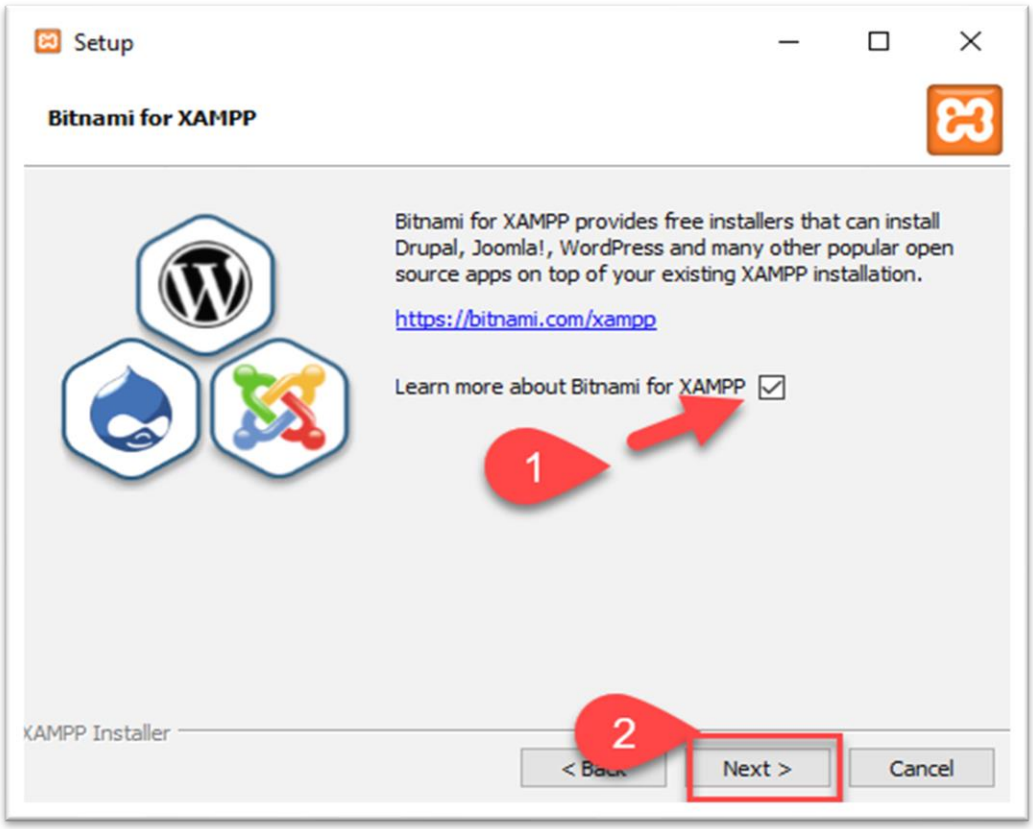
กด Next



ติ๊กเลือกตามภาพ แล้วกด Next ไป

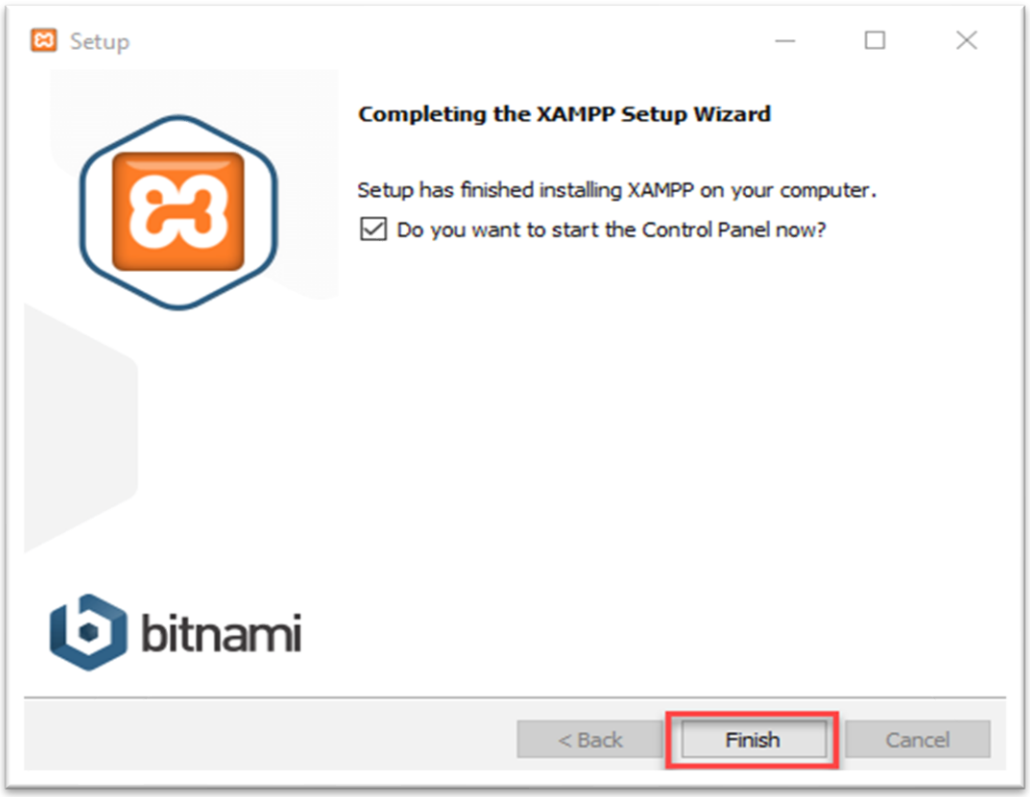


เลือก folder ที่ต้องการติดตั้ง โปรแกรมไว้ในที่นี้เลือก C:\xampp แล้วกด Next



ติ๊กเครื่องหมายออก แล้วกด Next ไป

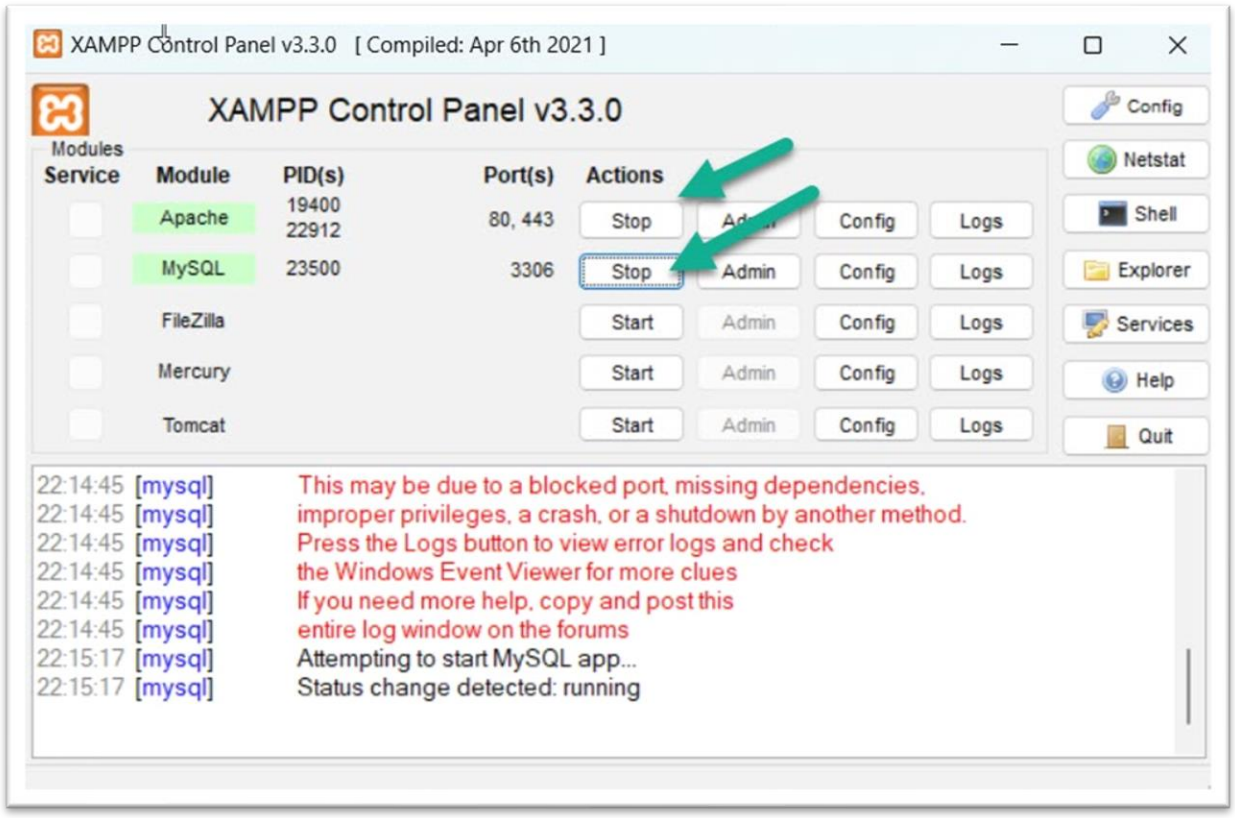




เสร็จสิ้นการติดตั้ง กด finish



เลือกภาษาที่ต้องการ ในที่นี้เลือก ภาษาอังกฤษ กด Save



กด start apache และ MySQL เพื่อให้ Server ทำงาน เป็นอันเสร็จสิ้นการติดตั้ง

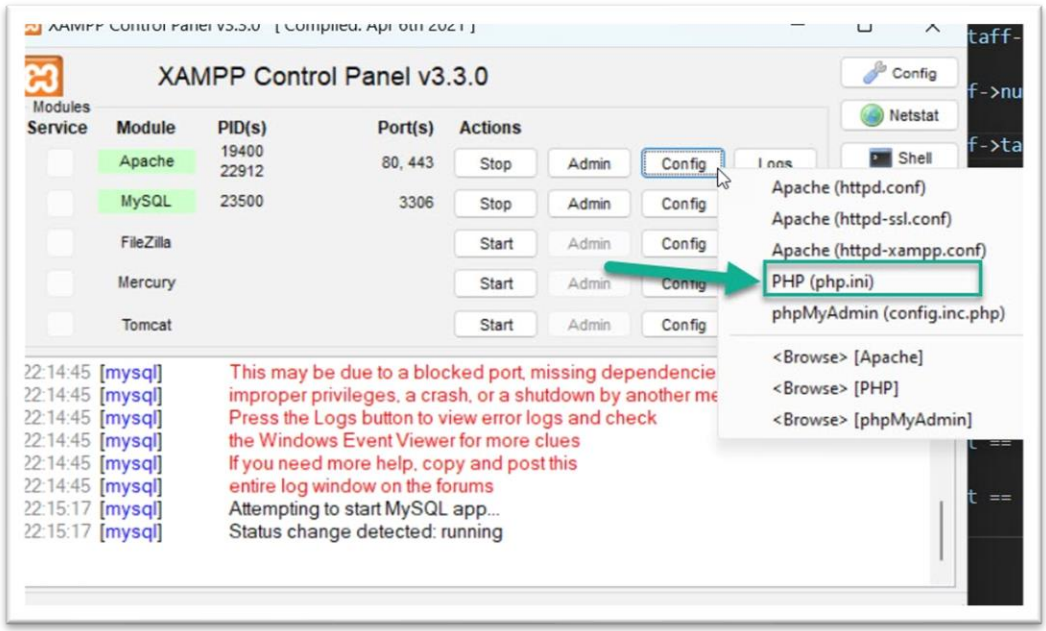
Extensions ของ PHP ที่จำเป็นต้องเปิดใช้งานสำหรับใช้งาน Laravel 11

การที่เราจะเขียน Laravel ให้ไม่ติดขัดปัญหาบางครั้งที่เราต้องใช้คำสั่ง Composer เพื่อสั่งให้ทำอะไรบางอย่าง อาจจะมี errors แจ้งเตือนสำหรับบางคำสั่ง เราควรต้องตรวจสอบหรือเปิด extensions ของ PHP ให้เรียบร้อยก่อน ซึ่ง Laravel มีความต้องการบางอย่างในเวอร์ชัน 11 ดังต่อไปนี้

PHP >= 8.2

BCMath PHP Extension	Ctype PHP Extension
cURL PHP Extension	DOM PHP Extension
Fileinfo PHP Extension	JSON PHP Extension
Mbstring PHP Extension	OpenSSL PHP Extension
PCRE PHP Extension	PDO PHP Extension
Tokenizer PHP Extension	XML PHP Extension

ให้เราตรวจสอบให้แน่ใจก่อนว่า extensions ที่ระบุด้านบนได้เปิดไว้หรือยังแต่โดยปกติถ้าเราใช้ XAMPP มันจะเปิดไว้ให้อัตโนมัตินะ เพื่อความมั่นใจให้เราเข้าไปตรวจสอบ extensions ในไฟล์ php.ini โดยปกติไฟล์ php.ini จะอยู่ใน C:\xampp\php\php.ini หรือเปิดไฟล์ php.ini ในโปรแกรม XAMPP ก็ได้เช่นกัน โดยเปิด XAMPP ขึ้นมาแล้วก็คลิกที่ปุ่ม config ใน Module ของ Apache แล้วเลือก PHP (php.ini) ตามภาพครับ





```
extension=curl
;extension=ffi
;extension=ftp
extension=fileinfo
extension=gd
extension=gettext
;extension=gmp
;extension=intl
;extension=imap
;extension=ldap
extension=mbstring
extension=exif      ; Must be after mbstring as it depends on it
extension=mysqli
;extension=oci8_12c  ; Use with Oracle Database 12c Instant Client
;extension=oci8_19  ; Use with Oracle Database 19 Instant Client
;extension=odbc
;extension=openssl
;extension=pdo_firebird
extension=pdo_mysql
;extension=pdo_oci
;extension=pdo_odbc
extension=pdo_pgsql
extension=pdo_sqlite
;extension=pgsql
;extension=shmop

; The MIBS data available in the PHP distribution must be installed.
; See https://www.php.net/manual/en/snmp.installation.php
;extension=snmp

;extension=soap
;extension=sockets
;extension=sodium
;extension=sqlite3
```

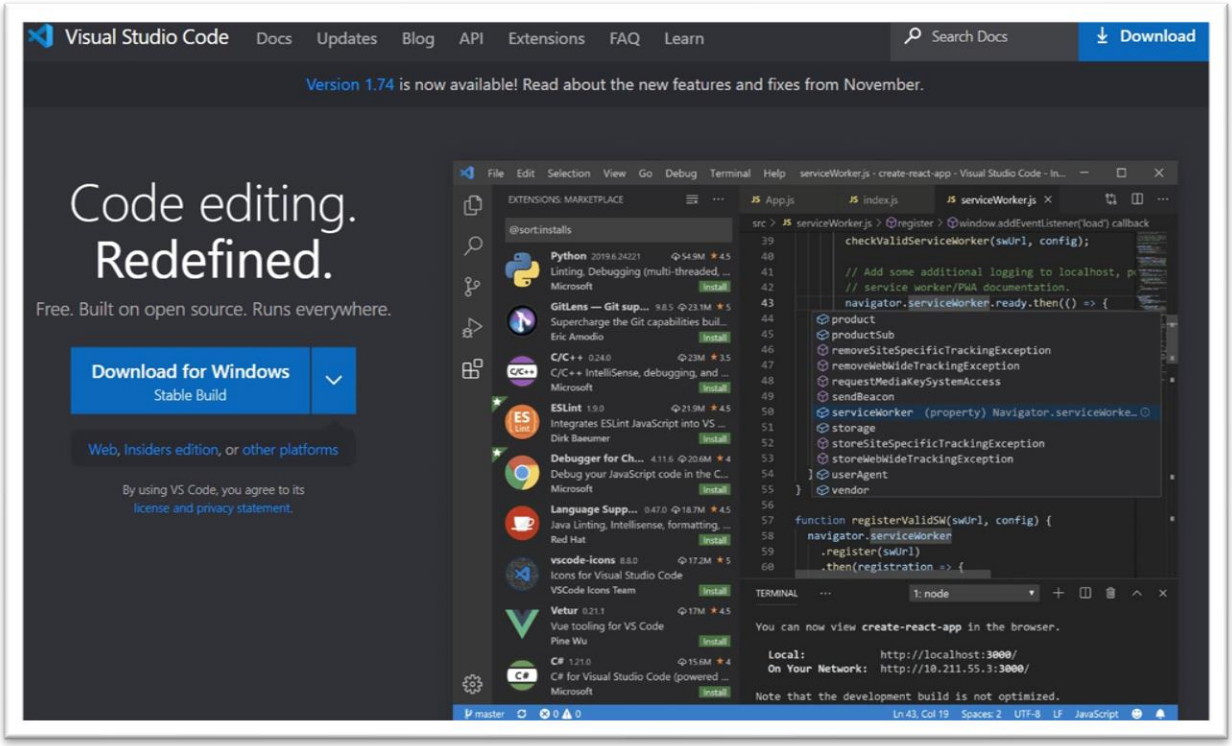
วิธีการเปิด extension ให้เอาเครื่องหมาย ; (เซมิโคล่อน) ด้านหน้า extension ที่เราต้องการจะเปิดออก เสร็จแล้วก็ให้บันทึกไฟล์ แล้วทำการ Restart Apache ใหม่

extension ที่มีเครื่องหมาย ; (เซมิโคล่อน) ด้านหน้า หมายถึง การปิดใช้งาน

extension ที่ไม่มีเครื่องหมาย ; (เซมิโคล่อน) ด้านหน้า หมายถึง การเปิดใช้งาน

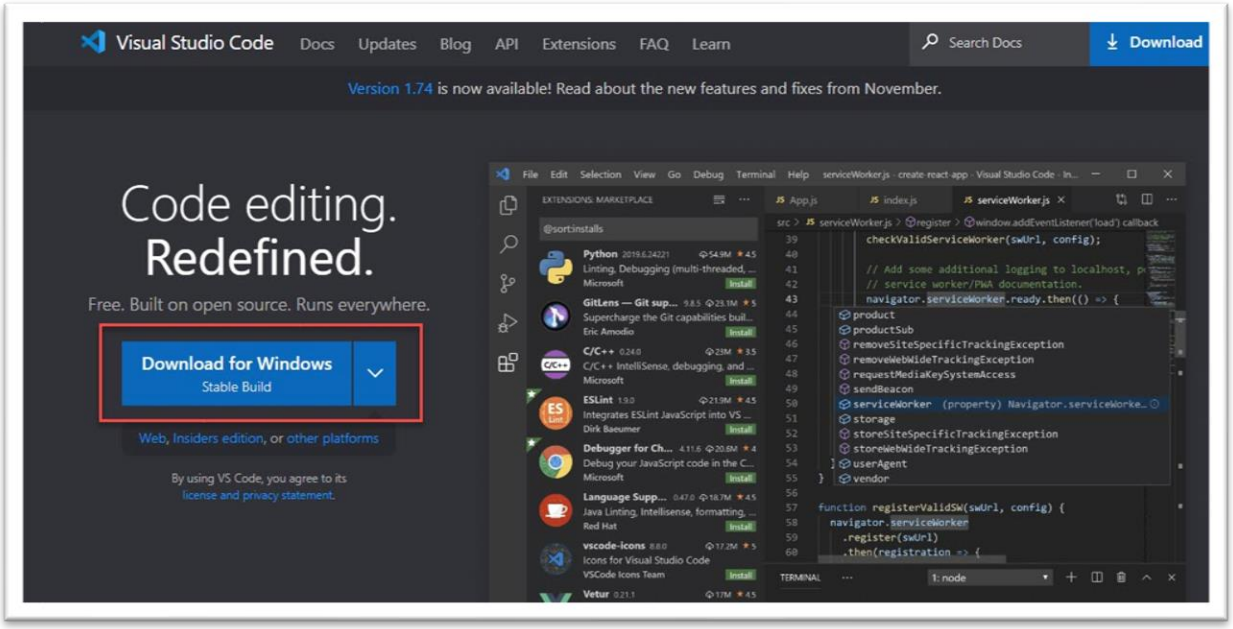
Visual Studio Code คืออะไร

Visual Studio Code หรือเรียกสั้นๆ ว่า VSCode เป็นโปรแกรมสำหรับการเขียนโปรแกรมภาษาต่างๆ ซึ่งเป็น IDE ยอดนิยมที่เหมาะมากสำหรับการใช้เขียนโปรแกรมในปัจจุบัน ด้วยความที่ว่าโปรแกรมมีความเบาและโหลดเร็ว ที่สำคัญมี Extension ให้ติดตั้งใช้งานได้เยอะมากๆ ทำให้ผู้เขียนโปรแกรมสะดวกต่อการเขียนโปรแกรมมากกว่าเดิม VSCode รองรับการเขียนหลายภาษา เช่น C++, C#, Java, Python, PHP, Go, .NET เพราะเหตุนี้แหละครับว่าทำไมถึงเป็นที่นิยมกัน โดยในหนังสือนี้จะใช้ VSCode ในการเขียนโปรแกรม หรือถ้าใครถนัด IDE ตัวอื่นไหนก็สามารถใช้ได้นะครับ เอาตามที่ท่านถนัดเลย 😊

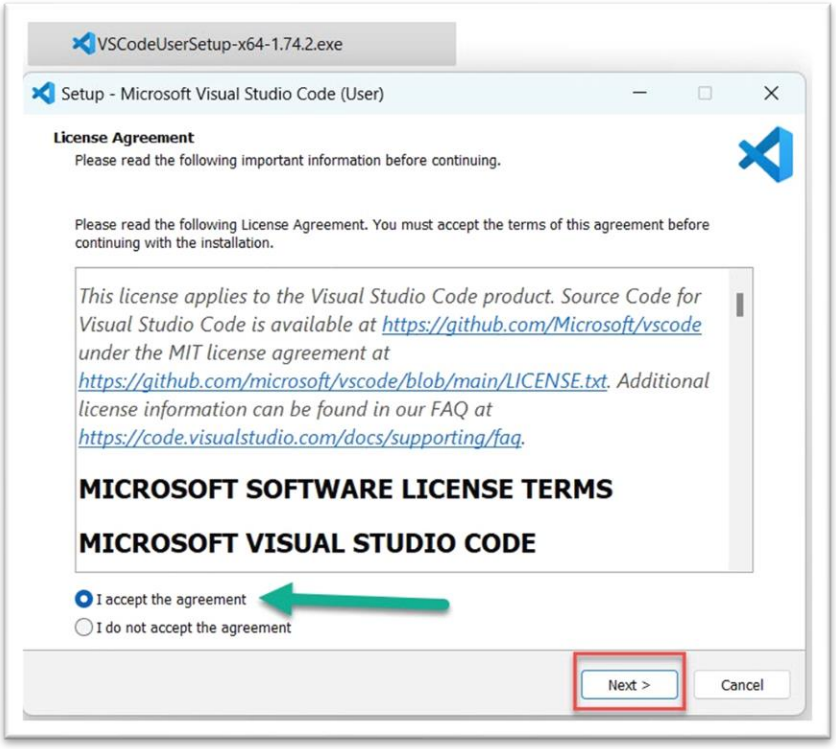


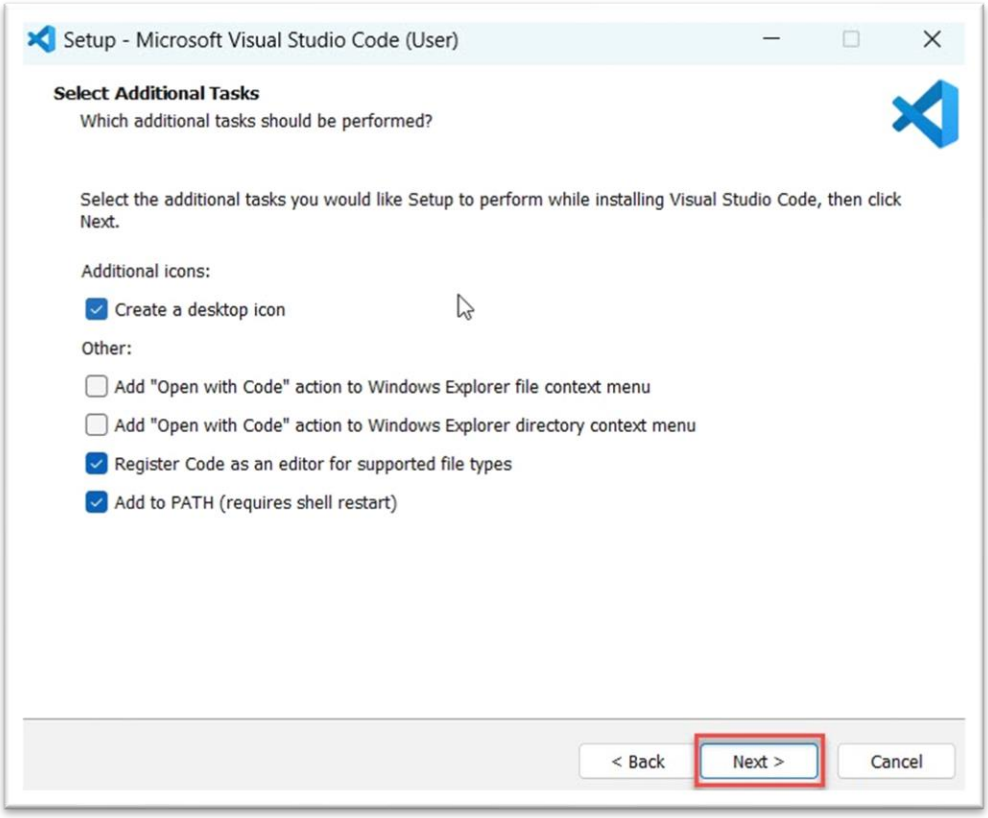
ติดตั้ง Visual Studio Code

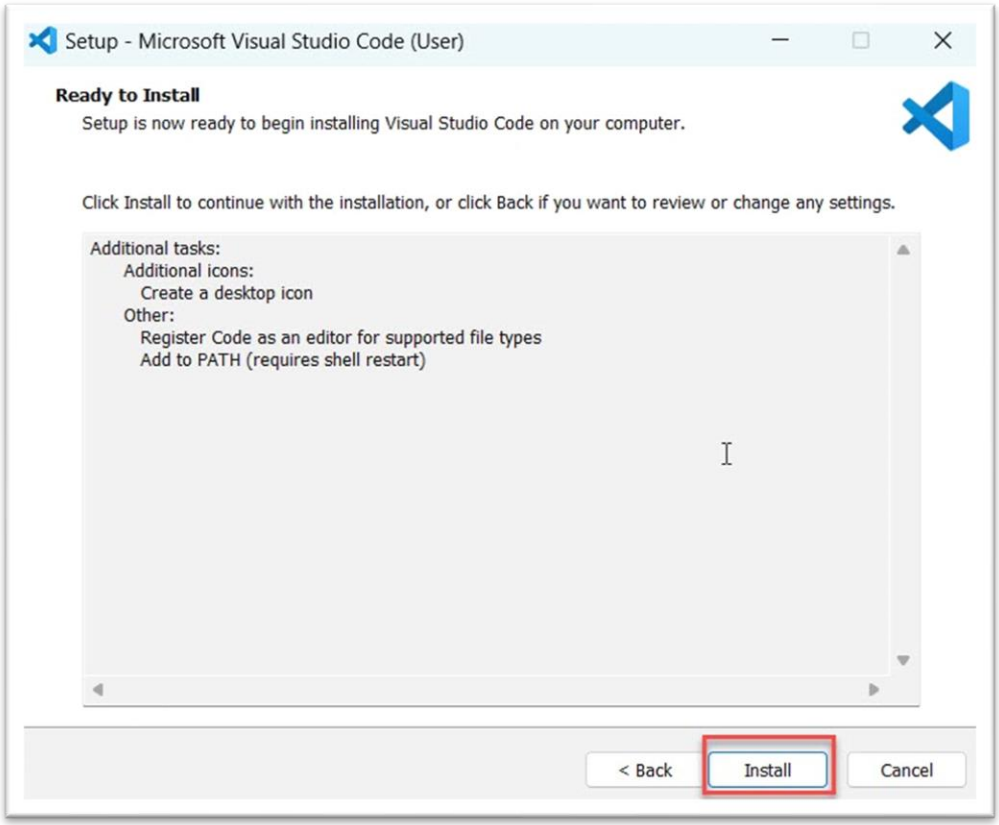
การติดตั้งสามารถทำได้โดยการดาวน์โหลดโปรแกรมจากเว็บไซต์หลักของ Visual Studio Code ที่ลิงค์ https://code.visualstudio.com ดาวน์โหลดติดตั้งตาม OS ที่ท่านใช้งานได้เลย ในหนังสือนี้จะใช้ระบบปฏิบัติการ Windows นะครับ การติดตั้งก็ไม่ยากเหมือนกันครับ



โดยปกติแล้วเราใช้ ระบบปฏิบัติการตัวไหน หน้าเว็บมันจะเลือกให้เราอัตโนมัติ คลิกที่ Download for Windows Stable Build เมื่อดาวน์โหลดมาแล้วให้ ดับเบิลคลิก ที่โปรแกรมจะเจอหน้าต่าง Setup ให้ติ๊กตามลูกศรชี้ แล้วกด Next







คลิก Install เพื่อติดตั้ง จากนั้นก็รอให้ติดตั้งจนเสร็จครับ ใช้เวลาไม่นาน

ติดตั้ง Extension สำหรับ Visual Studio Code

Extension จะเป็นตัวเสริมที่ทำให้เราเขียนโปรแกรมได้สะดวกสบายขึ้น โดยที่ผมแนะนำให้ติดตั้งก็จะมีดังนี้

HTML Snippets เอาไว้สร้าง Tag HTML แบบรวดเร็ว ของ Mohamed Abusaid

Laravel Blade Snippets ทำงานคล้ายกับ “HTML Snippets” แต่จะมีคำสั่งเฉพาะ Blade Template ด้วย เช่น ถ้าพิมพ์ “b:section” จะมี Auto Complete แสดงมาให้เลือกเป็น @section...@endsection ได้เลย ของ Winnie Lin

Laravel 5 snippets ทำงานคล้ายกับ “HTML Snippets” เช่นกัน เช่น พิมพ์ Auth::, Config:: and Session:: มันก็จะมี Auto Complete แสดงขึ้นมาให้เลือก ของ Winnie Lin

Path Intellisense ใช้สำหรับดึงชื่อ directory หรือ path file ครั้งที่พิมพ์ / มันจะมีชื่อ folder และไฟล์ ต่างๆ แสดงขึ้นมา เช่น จะดึงรูปก็ใช้ src ในแท็ก img (<img src=”/”>) เป็นต้น ของ Felix Becker

Indenticator ไว้ดูการย่อหน้า หรือปีกกาเปิดปิด if else , loop, class หรือ function ต่าง ๆ เพื่อให้รู้ว่าเป็นของ Block ไหนกันแน่ ของ SirTori

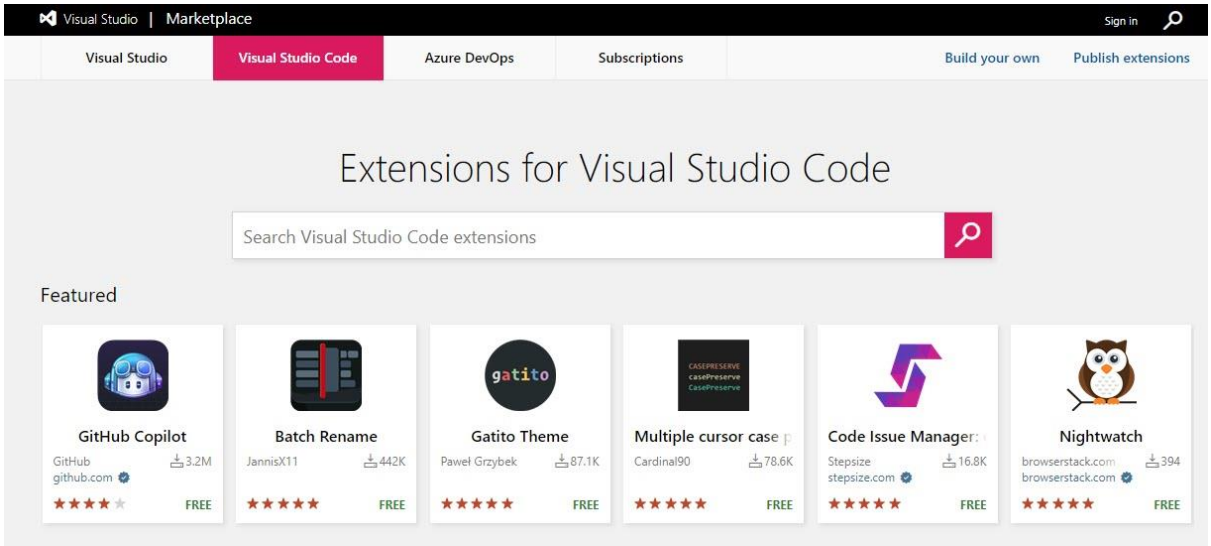
GitLens ความสามารถอย่างเช่น Cursor วางไว้บรรทัดไหนจะมีข้อมูล Git แสดง บอกว่าใคร Commit ไปเวลาเท่าไร ของ Eric Amodio

Prettier — Code formatter ไว้กด Save แล้วจะจัด Format ให้อัตโนมัติ (ต้องเข้าไปใน Setting แล้วเพิ่มค่าใน User Setting “editor.formatOnSave”: true) ของ Esben Petersen

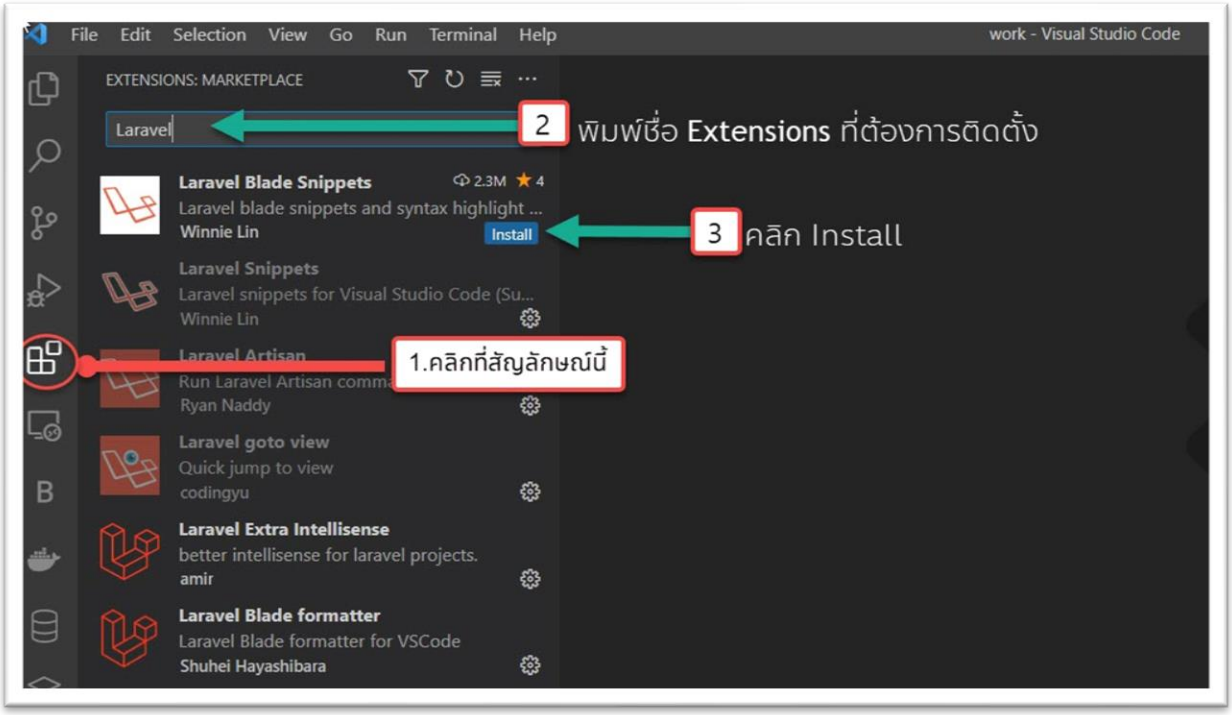
Vscode-icons ไว้สำหรับแสดง Icon ไฟล์ใน Explorer เพื่อให้เราดูง่าย ของ Roberto Huertas

Beautify css/sass/scss/less ช่วยให้เราจัดรูปแบบโค้ดที่เราพิมพ์ให้สวยงามอ่านง่าย ของ michelemelluso

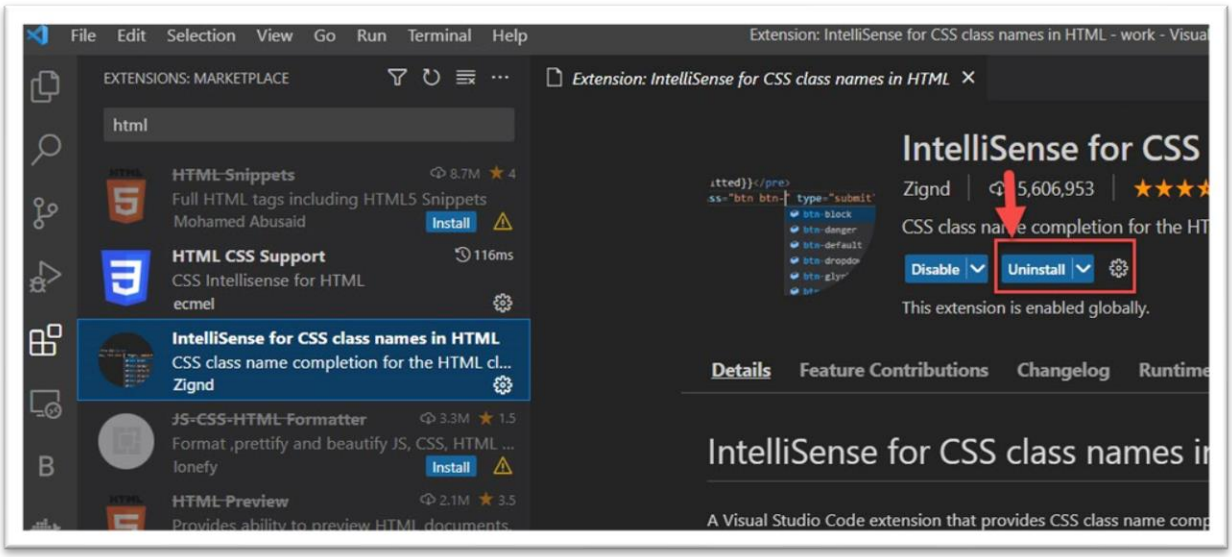
Highlight Matching Tag ช่วย Highlight แท็กเปิด ปิด ของ vincaslt



วิธีการติดตั้งให้เราเปิดโปรแกรม Visual Studio Code ขึ้นมาก่อน แล้วทำตามขั้นตอนตามภาพ



ให้ทำแบบนี้ทุก ๆ Extension ที่เราต้องการติดตั้ง อย่าลืมกด Reload Extension เพื่อให้โปรแกรมทำงานได้สมบูรณ์ขึ้นด้วยนะครับ ส่วนถ้าจะลบ Extension ก็ให้คลิกที่ปุ่ม Uninstall เพื่อลบได้เลยครับ



Laravel Framework คืออะไร



Laravel อ่านว่า ลา-รา-เวล เป็น PHP Framework ที่มีคุณสมบัติที่ช่วยให้เราเขียน Web Application ได้สะดวกสบายและง่ายขึ้น มีคุณสมบัติครบถ้วน การเขียนโค้ดที่สั้นขึ้น การเขียน Code ดูสะอาดตาสามารถอ่านและแก้ไขได้ง่าย เหมาะกับการทำงานร่วมกันเป็นทีม มี Package/Library/Helper ให้เราเรียกใช้งานได้เลย Laravel ถือว่าเป็นโอเพนซอร์สที่ออกแบบมาเพื่อพัฒนาเว็บ Web Application มีการจัดโครงสร้างให้ดูมีระเบียบเรียบร้อย เพิ่มความปลอดภัยของระบบ และอีกทั้งให้ความสะดวกในการพัฒนาเว็บแอปพลิเคชันหรือระบบงานที่มีความซับซ้อน เนื่องจากการพัฒนาเว็บแอพพลิเคชันจะพัฒนาตามรูปแบบ MVC Pattern (Model-View-Controller) คือการแยกการทำงานออกเป็นส่วนๆ เช่น การจัดการฐานข้อมูล,การจัดการ Sessions ,Caching ,การตรวจสอบข้อมูล (Validation) เป็นต้น โดยใน Laravel จะมี Artisan CLI (Artisan Command-Line Interface) หรือชุดคำสั่งที่ใช้เรียกงานผ่านทาง Command Line เพื่อช่วยในการพัฒนาเว็บติดมาด้วย นอกจากนี้ Laravel ยังเป็น Framework ที่มีคนนิยมใช้งานเป็นจำนวนมาก จึงเป็นเรื่องง่ายที่จะค้นหาข้อมูลจาก Google ในเวลาที่ติดปัญหาหรือมีข้อสงสัย Laravel สร้างขึ้นโดย Taylor Otwell ภายใต้ลิขสิทธิ์ของ MIT และ Source Code ได้ถูกเก็บไว้บน Host ของ Github ซึ่งในวันที่เขียนหนังสือนี้ Laravel ได้พัฒนามาถึงเวอร์ชัน 11 แล้ว

Framework คือ โครงสร้างหรือรูปแบบที่ใช้ในการเขียนโปรแกรม มีการวางโค้ดไว้อย่างเป็นระบบและมีแบบแผนการเขียนโค้ดที่เป็นมาตรฐาน นอกจากนี้ยังมีการกำหนดโฟลเดอร์ที่เก็บไฟล์ไว้อย่างเป็นระเบียบ เหมาะกับการพัฒนาแอปพลิเคชันที่มีขนาดใหญ่ มีผู้พัฒนาหลายคน เพราะจะช่วยให้รูปแบบการเขียนโค้ดไม่สะเปะสะปะและเป็นไปในทิศทางเดียวกัน สามารถจัดการไฟล์ได้อย่างง่าย ซึ่งหนึ่งใน Framework ยอดนิยมที่มีผู้ใช้งานกันอย่างแพร่หลายก็คือ Laravel

ข้อดี Laravel เป็น Framework ที่มี Library ให้ติดตั้งใช้งานเสริมมากมายที่มีผู้พัฒนาเผยแพร่ใน packagist สามารถโหลดมาติดตั้งใช้งานกับโปรเจคได้ฟรี ๆ จึงทำให้นักพัฒนาระบบหรือเว็บแอพพลิเคชันไม่ต้องเสียเวลาเขียนโค้ดหรือเขียน Library เองทั้งหมด

ข้อเสีย เมื่อมีการอัพเดต Version บางครั้งอาจทำให้มีบางไฟล์เปลี่ยนไป ทำให้เกิดความวุ่นวายในการอัพเดตพอสมควร หรือ PHP คนละ Version ถ้า Library ไม่อัพเดตตามก็จะทำให้ Library ที่เราใช้อยู่ทำงานไม่ได้ ฉะนั้นแล้วการเลือกใช้ Library ต้องดูด้วยว่า Library ที่ใช้อยู่ปัจจุบันทางผู้พัฒนาได้มีการอัพเดตปรับปรุงอยู่เรื่อย ๆ หรือไม่

ทำไมต้องใช้ Laravel Framework

1. เป็น Framework ที่มีการอับเดตตลอดเวลาในขณะที่เขียนหนังสือนี้ เป็น Laravel 11
2. มีการเขียนโค้ดที่เป็นรูปแบบที่มาตรฐาน ช่วยลดระยะเวลาในการทำงาน
3. มี CLI artisan ช่วยในเรื่อง Generate ไฟล์/คำสั่งต่างๆ
4. มี Migrate ที่ช่วยในเรื่อง Generate table, fields
5. มี Library/Package ให้เลือกใช้มากมาย
6. มีเครื่องมือช่วยเหลือให้เราเขียนโปรแกรมได้แบบง่ายๆ เช่น ORM, Blade Template, Webpack
7. ช่วยให้การทำงานเป็นทีมงานขึ้น โค้ดเป็นมาตรฐานเดียวกัน
8. ช่วยในการบำรุงรักษาโค้ดได้ง่ายขึ้น
9. เป็นที่นิยมอย่างแพร่หลายในกลุ่มนักพัฒนาต่างๆ
10. มีชุมชนมากมายที่สามารถสอบถาม ปัญหา ต่าง ๆ ได้

จุดเด่นของ Laravel Framework

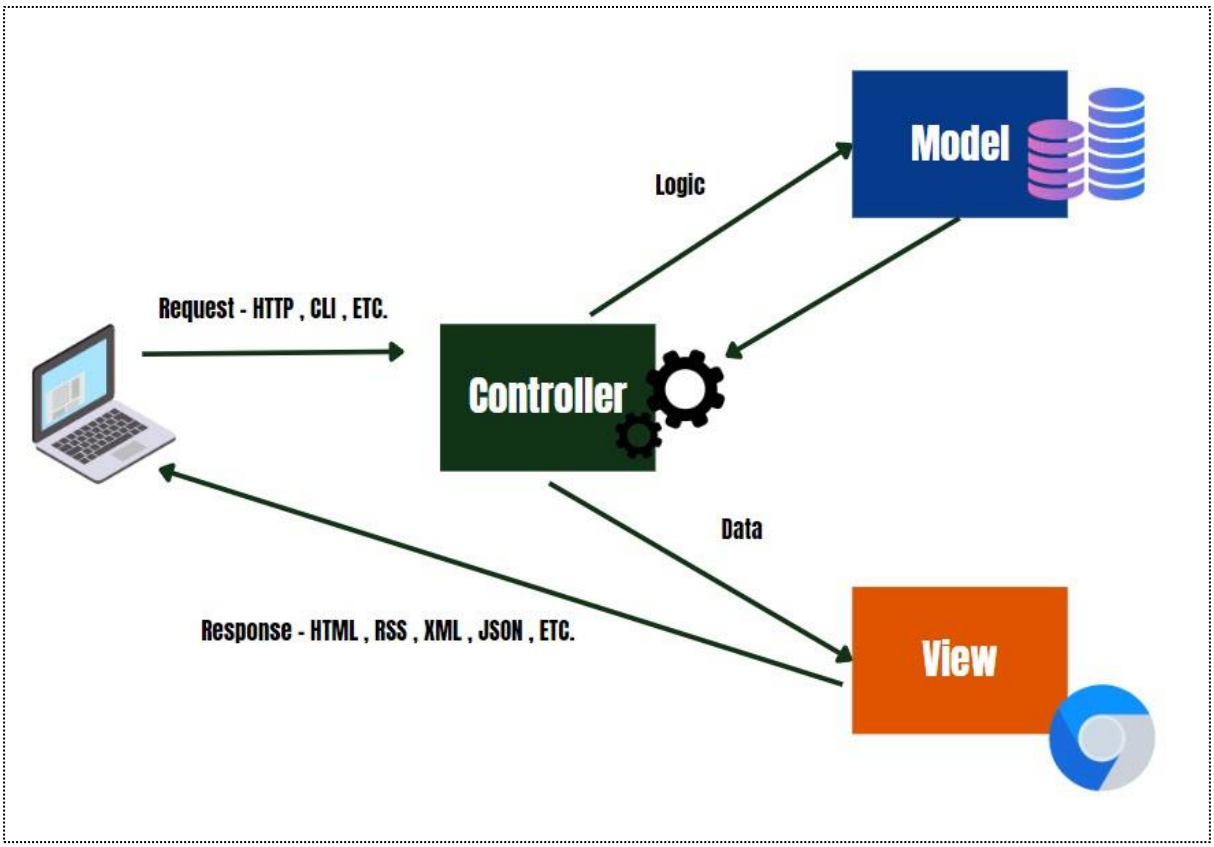
1. **Bundle** คือ สิ่งที่มาพร้อมกับ แพ็คเกจของระบบ ที่ทำให้เราสามารถนำมาใช้กับเว็บแอปพลิเคชันของเราได้เลย จึงทำให้เราประหยัดเวลาในการเขียนโค้ด และลดจำนวนการเขียนโค้ดลงไปอย่างมาก
2. **Class Auto loading** คือ การโหลดคลาสอัตโนมัติ ระบบจะทำการโหลดคลาสของ PHP มาใช้งานอัตโนมัติ โดยไม่ต้องกำหนดค่าลงไปใช้งานเอง ในการโหลด ระบบจะป้องกันการโหลดในส่วนประกอบ (Component) ที่ไม่ใช้งานและจะเลือกโหลดเฉพาะส่วนประกอบที่นำมาใช้งานเท่านั้น
3. **View Composer** คือ ส่วนนี้จะเป็นส่วนของโค้ด HTML ที่นำมาเรียงติดต่อกันและจะทำงานหลังจากที่ประกอบโค้ดกันเสร็จเรียบร้อยแล้ว เช่น การแบ่งส่วน Header, Container, Sidebar และ Footer เป็นต้น
4. **Migration** สำหรับจัดการฐานข้อมูล เช่น การสร้างตารางเก็บข้อมูลต่าง ๆ ดีมาก ๆ
5. **Unit Testing** คือ หน่วยที่ใช้ทดสอบ Laravel จะยินยอมให้ผู้ใช้งานสามารถสร้าง Unit Test ขึ้นมาเพื่อทดสอบงานของตัวเองได้โดยผ่าน Artisan utility
6. **The Eloquent ORM** คือ ชุดคำสั่งใน Query เป็นเครื่องมือที่ช่วยในการคิวรี่ข้อมูลต่างๆในฐานข้อมูล
7. **Reverse Routing** การเร้าท์ติ้ง ในส่วนนี้จะทำให้ผู้ใช้งานสามารถกำหนดชื่อของ URL เพื่อที่จะชี้ไปยังส่วนต่างๆตามต้องการ
8. **Restful Controller** คือ การกรองชนิดตามการส่งคำขอ จะช่วยให้ผู้ใช้งานสามารถกรองชนิดการส่งคำร้องขอจากฟอร์มทั้งแบบ POST และ แบบ GET
9. **The IoC (Inversion of Control) Container** เป็นส่วนในการจัดเก็บ Library ภายนอกที่ผู้ใช้งานจะนำเข้ามาใช้


The PHP Framework
for Web Artisans


laravel-app % artisan test
PASS Tests/Feature/TodoTest
✓ task can be added
✓ task can be completed
✓ task can be deleted
Test 3 passed
Time 0.41s

MVC คืออะไร

MVC คือ สถาปัตยกรรมซอฟท์แวร์ชนิดหนึ่ง ซึ่งในขณะนี้ถือว่าเป็นแบบแผนสถาปัตยกรรม (Architectural Pattern) ที่ใช้ในสาขาวิศวกรรมซอร์ฟแวร์ รูปแบบ MVC ใช้เพื่อแยกส่วนซอฟท์แวร์ในส่วนตรรกะเนื้อหา (Domain Logic) ได้แก่ความเข้าใจในระบบของผู้ใช้ และส่วนการป้อนข้อมูลและแสดงผล (GUI) ซึ่งช่วยให้การพัฒนา การทดสอบ และการดูแลรักษาซอฟท์แวร์ แยกออกจากกัน มีดังนี้



M โมเดล (Model) ส่วนของซอฟท์แวร์ที่ใช้ในการแปลการทำงานของระบบ ไปสู่สิ่งที่ระบบซอฟท์แวร์ได้ถูกออกแบบเอาไว้ ตรรกะเนื้อหาใช้เพื่อให้ความหมายแก่ข้อมูลดิบ (ยกตัวอย่างเช่น การคำนวณว่าวันนี้เป็นวันเกิดของผู้ใช้หรือไม่, หรือจำนวนเงินรวม ภาษี และค่าส่งสินค้า ในตะกร้าสินค้า) เมื่อโมเดลมีการเปลี่ยนแปลง จะมีการส่งคำเตือนให้แก่ วิว ที่เกี่ยวข้องเพื่อปรับค่าระบบซอฟท์แวร์หลายระบบใช้การเก็บข้อมูลถาวร เช่น ฐานข้อมูล เพื่อเก็บข้อมูลเหล่านี้ MVC ไม่ได้กำหนดถึงระดับการเข้าถึงข้อมูล เพราะเป็นที่เข้าใจกันว่าส่วนนี้จะอยู่ภายใต้ หรือถูกครอบคลุมด้วยโมเดล โมเดลไม่ได้เป็นเพียงอ๊อบเจ็คที่ใช้เข้าถึงข้อมูล แต่ในระบบซอฟท์แวร์เล็กๆ ซึ่งมีความซับซ้อนน้อยจะไม่เห็นความแตกต่างมากนัก

V วิว (View) แสดงผลค่าในโมเดลในรูปแบบที่เหมาะสมต่อการปฏิสัมพันธ์กับผู้ใช้ ในแต่ละโมเดลสามารถมีวิวได้หลายแบบ เพื่อใช้ในจุดประสงค์ที่ต่างกัน

C คอนโทรลเลอร์ (Controller) รับค่าเข้ามา และทำการตอบสนองโดยเรียกใช้ออบเจคในโมเดล แอพพลิเคชั่นที่ใช้ MVC อาจจะเป็นกลุ่มของ โมเดล/วิว/คอนโทรลเลอร์ โดยแต่ละกลุ่มใช้ในงานต่างกันไป MVC มักจะพบได้ในเว็บแอพพลิเคชันโดย วิว จะเป็น HTML หรือ XHTML ที่สร้างโดยแอพพลิเคชั่นนั้น ส่วน

คอนโทรลเลอร์รับค่า GET หรือ POST เข้ามา แล้วเลือกติดต่อกับโมเดลในส่วนที่เกี่ยวข้องเพื่อตอบสนองโมเดลซึ่งมี business rules จะทำการจัดการตามคำร้องขอนั้นๆ

ภาพรวมของรูปแบบ MVC

แม้ว่า MVC จะมีรูปแบบต่างกันไป แต่ control flow จะเป็นดังนี้

1) ผู้ใช้ทำการตอบโต้ กับ user interface ในทางใดทางหนึ่ง

2) คอนโทรลเลอร์ จัดการ event ที่เกิดขึ้นจาก user interface มักจะโดย handler ที่ได้สร้างเอาไว้ หรือ callback แล้วแปลง event เป็นการกระทำของผู้ใช้ที่เหมาะสม ซึ่งเข้าใจได้ด้วยโมเดล

3) คอนโทรลเลอร์ แจ้งโมเดลถึงการกระทำของผู้ใช้ เช่นอาจจะทำให้เกิดการเปลี่ยนสถานะในโมเดล (ยกตัวอย่างเช่น คอนโทรลเลอร์ปรับปรุงตะกร้าสินค้าของผู้ใช้)

4) วิว ถามโมเดลเพื่อที่จะสร้างการแสดงผลที่เหมาะสมแก่ผู้ใช้ (ยกตัวอย่างเช่น วิวแสดงรายการสินค้าในตะกร้าสินค้า) โดยวิวได้ข้อมูลมาจากโมเดล ในบางรูปแบบของ MVC คอนโทรลเลอร์อาจจะให้คำสั่งแก่วิวด้วย หรือบางรูปแบบ วิวจะได้รับการบอกจากโมเดลถึงการเปลี่ยนแปลง

5) user interface รอการปฏิสัมพันธ์จากผู้ใช้ ซึ่งจะกลับเข้าสู่วงจรในขั้นแรกในบางรูปแบบของ MVC เช่นใน W3C XForms ใช้หลักการของ dependency graph เพื่อให้เกิดการปรับปรุงวิวโดยอัตโนมัติ เมื่อข้อมูลในโมเดลเปลี่ยนแปลง ด้วยการแยกโมเดลและวิวออกจากกัน MVC ช่วยให้ความซับซ้อนในการออกแบบลดลง และเพิ่มความคล่องตัวและการนำโค้ดกลับมาใช้ใหม่ (Code reuse)

จากประการณ์ของผมที่เคยเขียนโปรแกรมแบบธรรมดา เมื่อเปลี่ยนมาเขียนในรูปแบบ MVC ทำให้ชีวิตการเขียนโปรแกรมง่ายขึ้นมากๆ สามารถทำโปรเจกใหญ่ๆเป็นทีม ร่วมกันได้อย่างสบาย ซึ่งถ้าใครได้เขียน Framework ก็จะรู้ดี ตัวอย่างที่ใช้งาน MVC เช่น Codeigniter Framework , Yii Framework , Laravel Framework

บทสรุป ของ MVC

Model เป็นส่วนที่ติดต่อสื่อสารระหว่าง Object กับ Database ใช้เชื่อมความสัมพันธ์ของข้อมูล และ ตรวจสอบความถูกต้องของข้อมูล

Controller เป็นส่วนที่ใช้สำหรับควบคุมการทำงานของโปรแกรม (Logic) และใช้ติดต่อการทำงานระหว่างผู้ใช้และโปรแกรม รวมถึงการติดต่อกับ Database ผ่าน Model และแสดงผลข้อมูลผ่านทาง View Controller จะเป็นส่วนที่มีการประมวลผลหลักของโปรแกรม

View จะเป็นส่วนที่แสดงผลผ่าน Web Browser ที่จะทำให้ผู้ใช้งานมองเห็น โดยจะใช้ HTML หรือ PHP ในส่วนของ View จะทำงานสัมพันธ์กับ Controller

ข้อดี ของการเขียนโปรแกรมแบบ MVC นั้น ทำให้เราสามารถแก้ไขโค้ดได้ง่าย เช่น ถ้าเราต้องการแก้ไขการแสดงผลก็ให้ไปแก้ไขในโค้ดส่วนของ View โดยไม่ต้องยุ่งเกี่ยวกับโค้ดส่วนของ Model และ Controller และเนื่องจากแบ่งโค้ดออกเป็นส่วน ๆ แบบนี้ ทำให้ข้อเสียที่ตามมาคือจำนวนไฟล์ในระบบก็จะมีมากขึ้น แต่ก็ยังถือว่าโครงสร้างแบบ MVC ช่วยให้พัฒนาระบบได้ง่ายขึ้น

Best Practices การเขียนโปรแกรมตามรูปแบบ MVC ใน Laravel นั้นต้องศึกษาแนวทางที่ดีกันก่อน ว่าจะเขียนให้ดียังไม่ อาจจะไม่ดีมาก แต่มันจะทำให้เราบำรุงรักษาโปรแกรมที่เราเขียนได้ง่ายในภายภาคหน้าและเมื่ออนาคตอาจจะมีผู้พัฒนาโปรแกรมต่อจากเรา ก็สามารถรู้ได้ว่าส่วนต่าง ๆที่เขียนไว้มันอยู่ที่จุดไหน ผมจะสรุปแต่ละส่วนให้สั้นๆ แบบนี้ครับ

1. Model ถ้าจะเขียนให้ดี ควรเขียนโค้ดที่เป็นส่วน Business Data เช่น โค้ดในส่วนของการตรวจสอบความถูกต้องของข้อมูล การประมวลผล Business login ถ้าจะเขียนโปรแกรมดึงข้อมูลจากฐานข้อมูลมาใช้งานก็ให้เขียนเป็น Method ไว้ หลีกเลี่ยงการเขียน HTML ตรง ๆ ใน Model

2. Controller ถ้าจะเขียนให้ดี ควรมีไว้เรียก Method เกี่ยวกับ Models และเรียก Component ถ้าจะมีการเขียนเกี่ยวกับ request ข้อมูลหรือส่งข้อมูลไปให้ Views เพื่อนำไปแสดงผลให้มาเขียนไว้ที่ Controller ไม่ควรมีโค้ดการประมวลผลของ Models ถ้ามีให้ไปเขียนไว้ที่ Models เลย หลีกเลี่ยงการเขียน HTML และโค้ดที่เกี่ยวกับการแสดงผล ให้ไปเขียนไปที่ View จะดีที่สุด

3. View ถ้าจะเขียนให้ดี จะต้องไม่มีโค้ดที่เกี่ยวกับการ Query ข้อมูลจากฐานข้อมูล หลีกเลี่ยงการรับค่า $_GET , $_POST เพราะเป็นหน้าที่ของ Controller ถ้าจะรับค่ามาจาก Model จะต้องไม่ไปแก้ไขค่าที่รับมา ต้องมีโค้ดเฉพาะ HTML และ PHP ที่เกี่ยวข้องกับการแสดงผลเท่านั้น

Composer คืออะไร

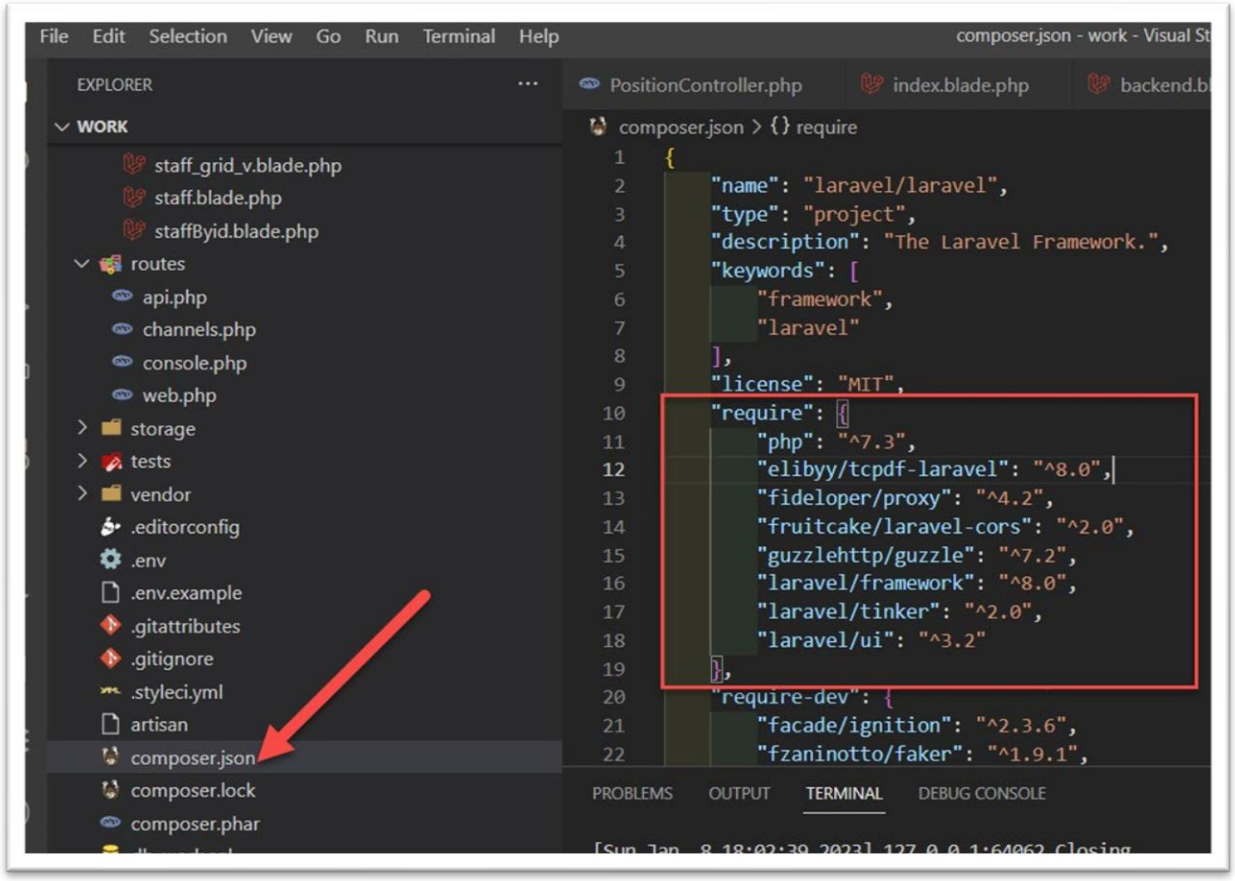


Composer เป็นโปรแกรมที่ใช้สำหรับจัดการ Package หรือ Library ต่างๆ ของ PHP โดยสมัยก่อนการเขียนทำ Web Application ด้วยภาษา PHP นั้น จะมีการทำ Package แต่ละคนไว้คนละที่กัน ใน Project หนึ่งอาจมีหลาย Package ดังนั้นเมื่อแต่ละ Package มีการ Update เราก็ตามไปดาวน์โหลด Package ในแต่ละที่มา Update ใน Project เราใหม่ ทำให้ต้องเสียเวลาในการไปตรวจสอบว่ามี Package ไหน Update บ้าง Composer จึงเกิดขึ้นมาเพื่อจัดการเรื่อง Package ของ PHP โดยจะจัดเก็บไว้ในเว็บไซต์ Packagist.org* Composer นั้นสามารถใช้ร่วมกับ PHP Framework ได้หลากหลาย เช่น Codeigniter Framework , Yii Framework , Django Framework และส่วนใหญ่ก็รองรับ Composer อยู่แล้ว ทำให้มีความปลอดภัยในการดาวน์โหลดมาใช้งาน ความเสี่ยงต่างๆ จากการดาวน์โหลด ไม่ว่าจะเป็นความปลอดภัย ไวรัส มัลแวร์ ก็จะน้อยลง

Composer มีแนวคิดคือจะรวบรวมและจัดระเบียบ package และ library ต่างๆ ให้มาอยู่ในแหล่งเดียวกัน สามารถติดตั้ง Library ได้จาก Command ซึ่งโปรแกรมจะทำการตรวจสอบและดาวน์โหลด Package นั้นๆ มาใช้งาน

ทำไมถึงต้องใช้ Composer

1. ลดการซ้ำซ้อนของการสร้าง Library สามารถค้นหาและตรวจสอบได้ว่า Library ตัวที่เราใช้มีแล้วหรือยัง
2. Library ที่เราสร้างจะสามารถใช้งานได้กับทุก Framework ที่รองรับ Composer
3. ลดเวลาการค้นหา Library ที่เราต้องการ เพราะ Library ที่อยู่บน packagist.org นั้นสามารถใช้ร่วมกับ Composer ได้
4. เมื่อ Library มีการอัพเดต ก็สามารถอัพเดตได้เลยผ่าน Composer โดยที่เราไม่ต้องไปติดตามการอัพเดตที่หน้าเว็บเลย ใน Laravel จะใช้คำสั่ง composer update



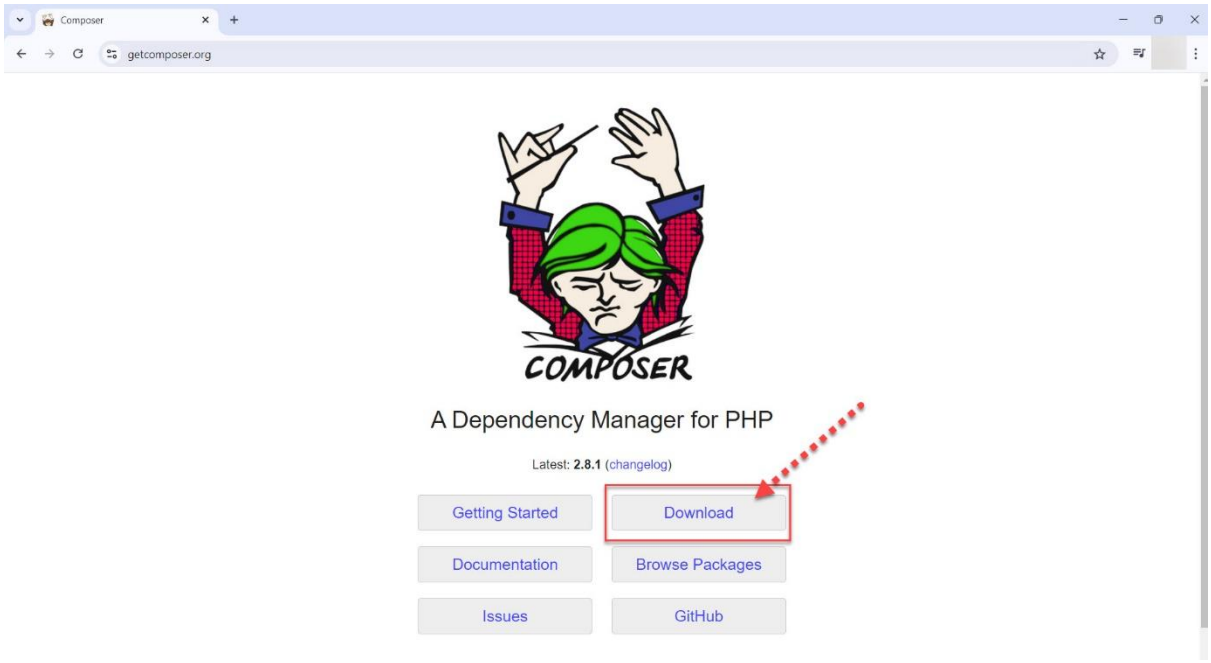
การทำงานของ Composer นั้นจะทำงานผ่านไฟล์ composer.json ซึ่งเป็นไฟล์ที่ใช้เป็นตัวจัดการ Library ต่างๆ ทำให้เราสะดวกต่อการใช้งานและสามารถตรวจสอบได้ง่ายด้วยว่า ใน Project งานที่เราทำนั้นมีการเรียกใช้ Library อะไรบ้าง 😊

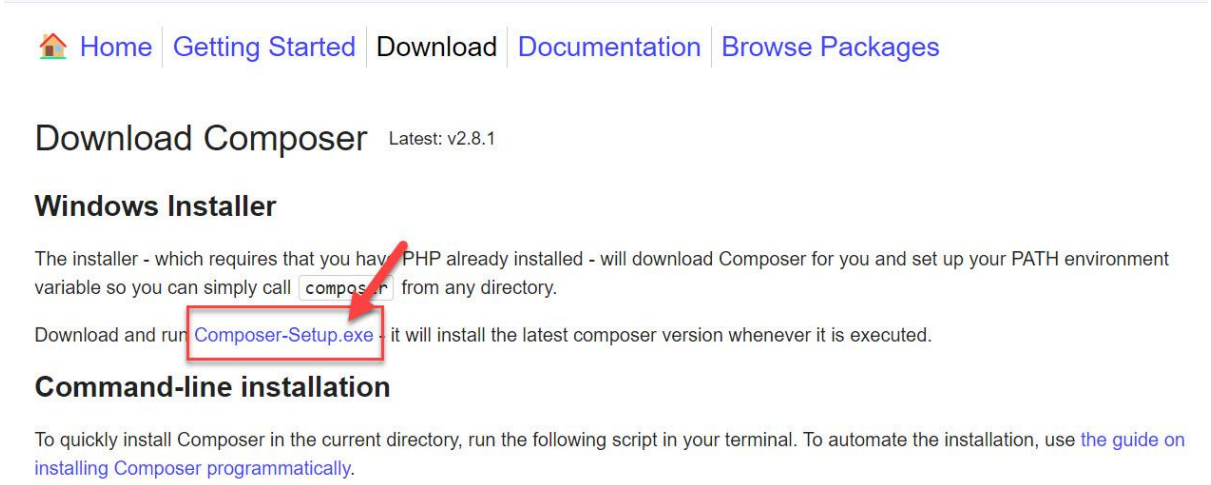
Packaagist.org* เป็นเว็บไซต์ที่รวบรวม Package ต่างๆ ไว้เพื่อให้สามารถติดตั้งผ่านโปรแกรม Composer ได้ โดยเป็นการระบุเพียงว่า Package นั้นๆ มีการ update version ใหม่หรือยัง โดยที่ไม่ได้เก็บ source code ไว้แต่อย่างใด ในการตรวจสอบนั้นจะใช้การ Hook ไปยังแหล่งเก็บ source code ต่างๆ เช่น Github.com หรือ Bitbucket.org หรือแหล่งอื่นๆ



ติดตั้ง Composer

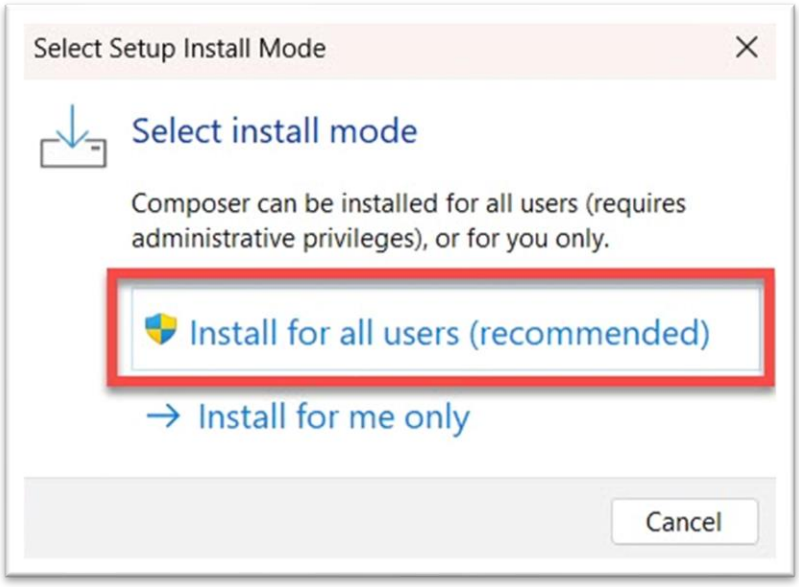
เข้าไปที่เว็บไซต์ https://getcomposer.org แล้วคลิกที่ปุ่ม Download ไฟล์ที่เป็น .exe สำหรับระบบปฏิบัติการ Windows ตามภาพครับ



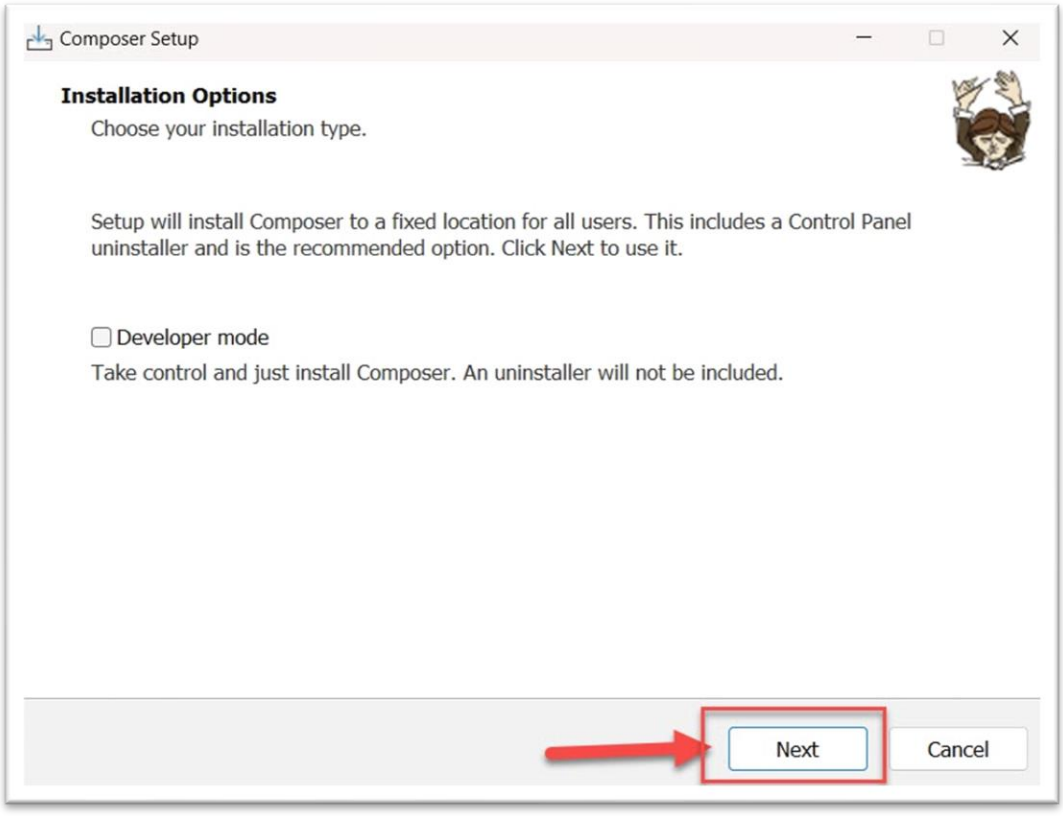


จะได้ไฟล์ชื่อว่า Composer-Setup.exe มา แล้วเริ่มติดตั้งได้เลย โดยการดับเบิลคลิกที่ไฟล์ เพื่อติดตั้งลงเครื่อง การติดตั้งในระบบปฏิบัติการ Windows เป็นเรื่องที่ไม่ยากครับ โดยปกติตัว Composer Version ล่าสุดนี้ จะมีการตั้งค่า PATH ให้อัตโนมัติ เราไม่ต้องไปตั้งค่า PATH ให้วุ่นวาย ไปดูขั้นตอนต่อไปเลยครับ

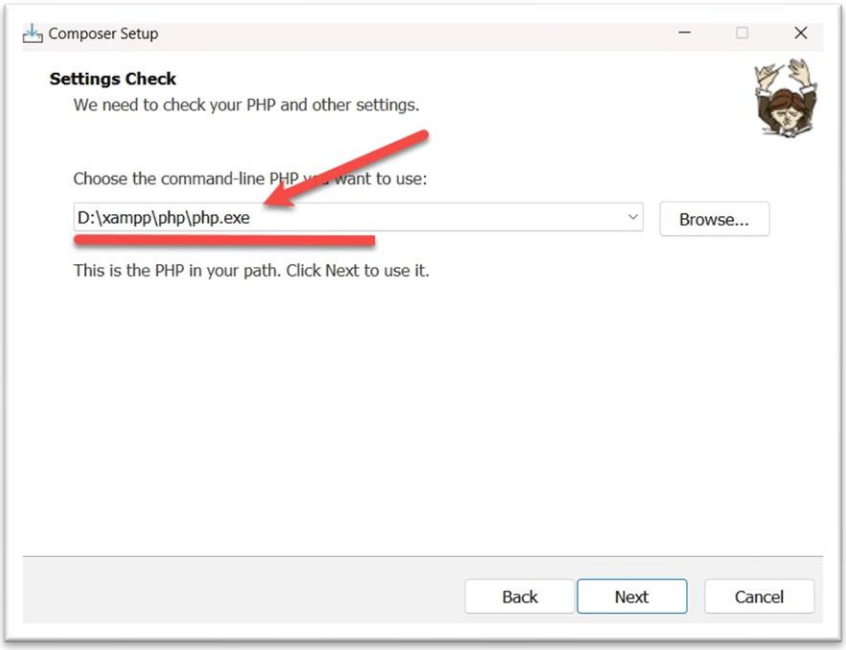
คลิกที่ install for all user ครับ



กด Next ไปเลยครับ

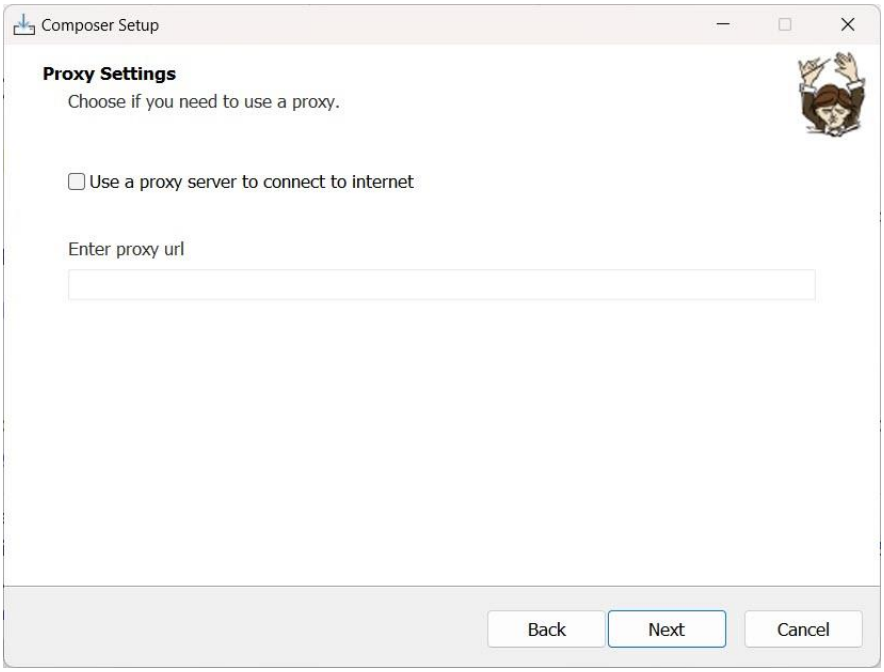


หน้าจอนี้ Composer จะตรวจหา PHP ที่มีอยู่บนเครื่องของเราและแสดงตำแหน่งที่ PHP อยู่ ให้เราเห็น กรณีติดตั้ง XAMPP ไว้แล้ว ปกติโปรแกรมก็จะเลือกให้เราเอง หากไม่ตรง ให้เรากด Browse... เพื่อเลือกตำแหน่งที่อยู่ของ PHP ให้ถูกต้อง ตามภาพปกติ PHP จะอยู่ใน Folder xampp\php\php.exe

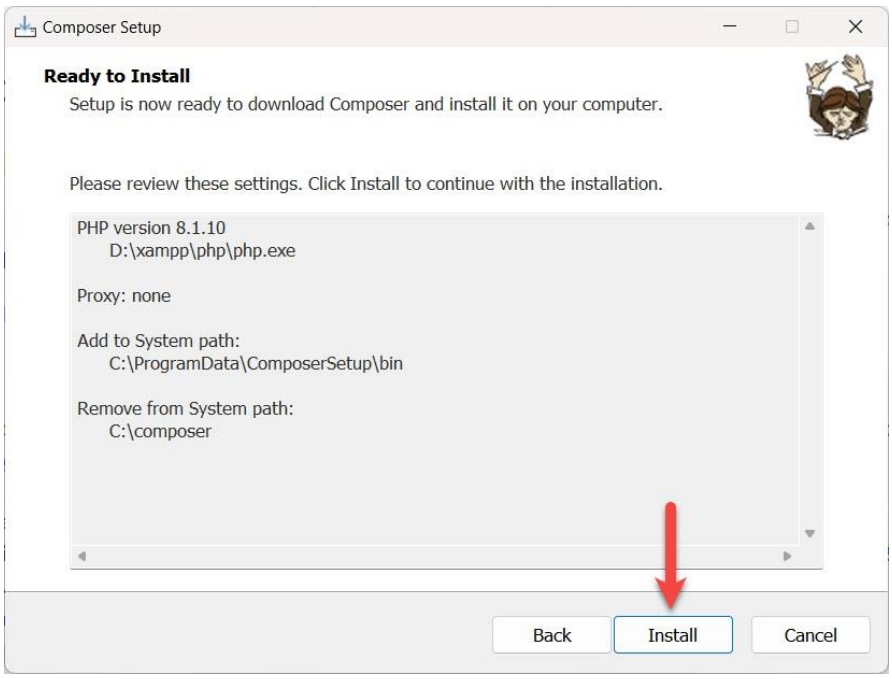


อย่าลืมทำเครื่องหมายถูกที่ช่องตามภาพด้วยนะครับ เพื่อให้ Composer กำหนด PATH ให้ถ้าติดตั้งครั้งต่อไป เมื่อมี Version มาใหม่ เราก็ไม่ต้องมาทำเครื่องหมายแบบนี้ กรณีนี้เครื่องผมได้เคยติดตั้ง Composer แล้วมันเลยไม่ขึ้นให้ผมทำเครื่องหมายถูก ตัวอย่างก็จะตามภาพด้านล่างนี้เลยครับ ในกรณีที่มีให้ทำเครื่องหมายถูก อาจจะสงสัยว่าทำไมต้องทำเครื่องหมายถูกตรงนี้ ก็เพราะว่าเวลาเราจะใช้งาน composer เราจะได้ไม่ต้องพิมพ์ชื่อยาว ๆ เช่น เวลาจะใช้งานปกติจะพิมพ์ php composer.phar ทุกครั้ง แทนที่จะพิมพ์ composer ... เลย และอีกอย่างเราสามารถเรียกใช้งานจากจุดไหนก็ได้ โดยไม่ต้องไปเสียเวลาดาวน์โหลดไฟล์ composer.phar ไปไว้ในโฟลเดอร์นั้น ๆ

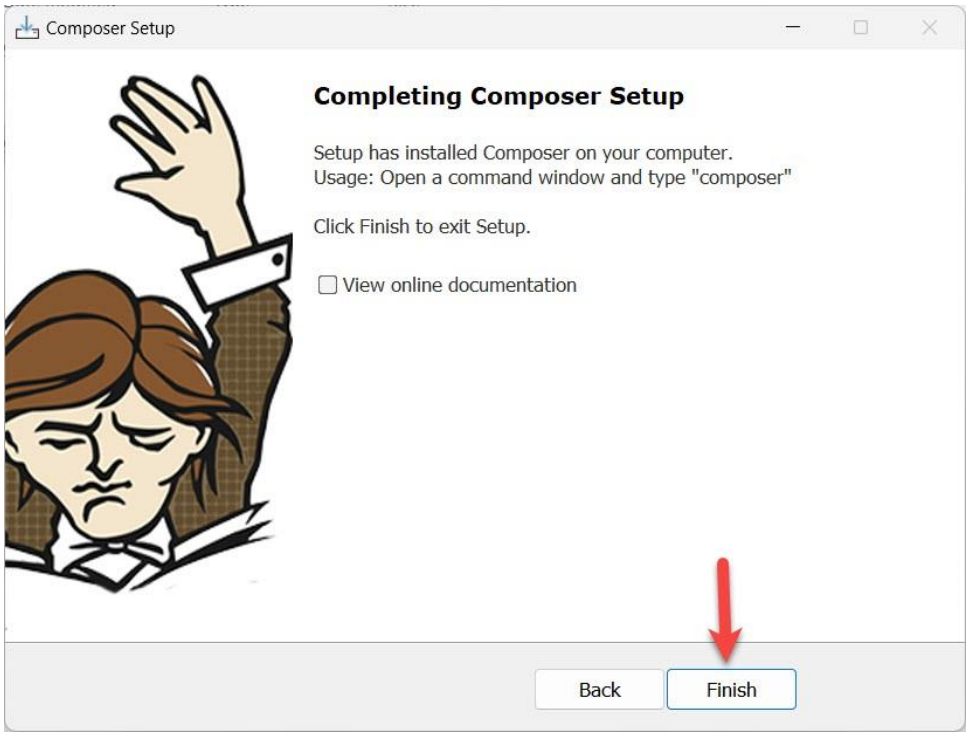




กำหนด Proxy ถ้ามีนะ

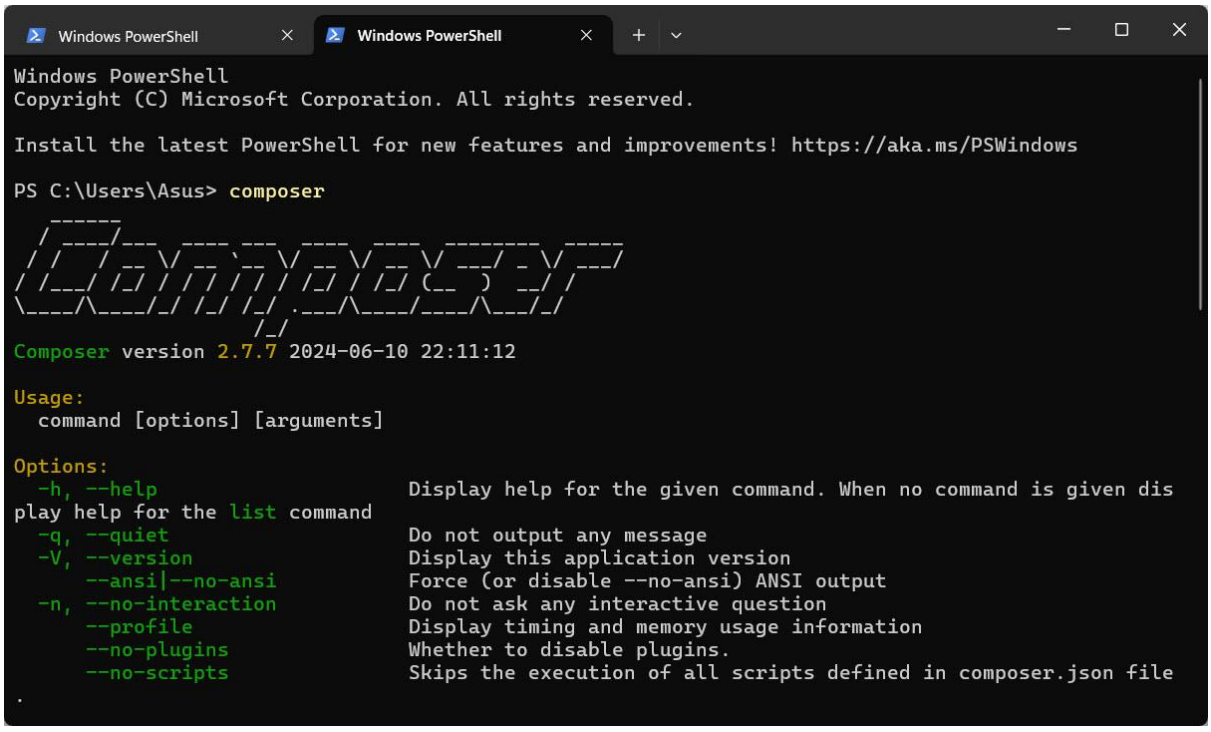


กดปุ่ม Install แล้วรอจนเสร็จ จะมีหน้าจอแสดงข้อมูลเกี่ยวกับความสำคัญของการติดตั้ง Composer ให้คลิกที่ปุ่ม Next แล้วจะมีหน้าต่าง Completing Composer Setup



ทดสอบใช้งานว่าเราได้ติดตั้ง Composer เสร็จสมบูรณ์พร้อมใช้งานหรือยัง โดยใช้ คำสั่ง composer หรือ composer -v บน command Prompt หรือ Windows PowerShell แล้วกดปุ่ม Enter ถ้าขึ้นหน้าจอแสดงโลโก้ composer ขึ้นมาตามภาพแสดงว่าการติดตั้งเสร็จสมบูรณ์พร้อมที่จะติดตั้ง Laravel แล้ว ถือเป็นการเริ่มต้นที่ดีครับ 😊

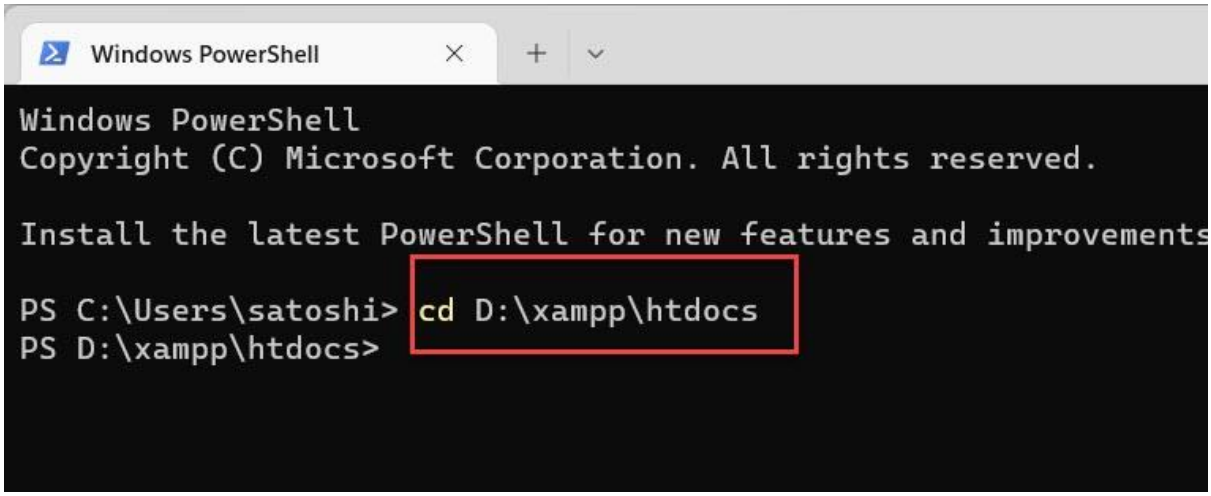
พิมพ์ คำสั่ง composer



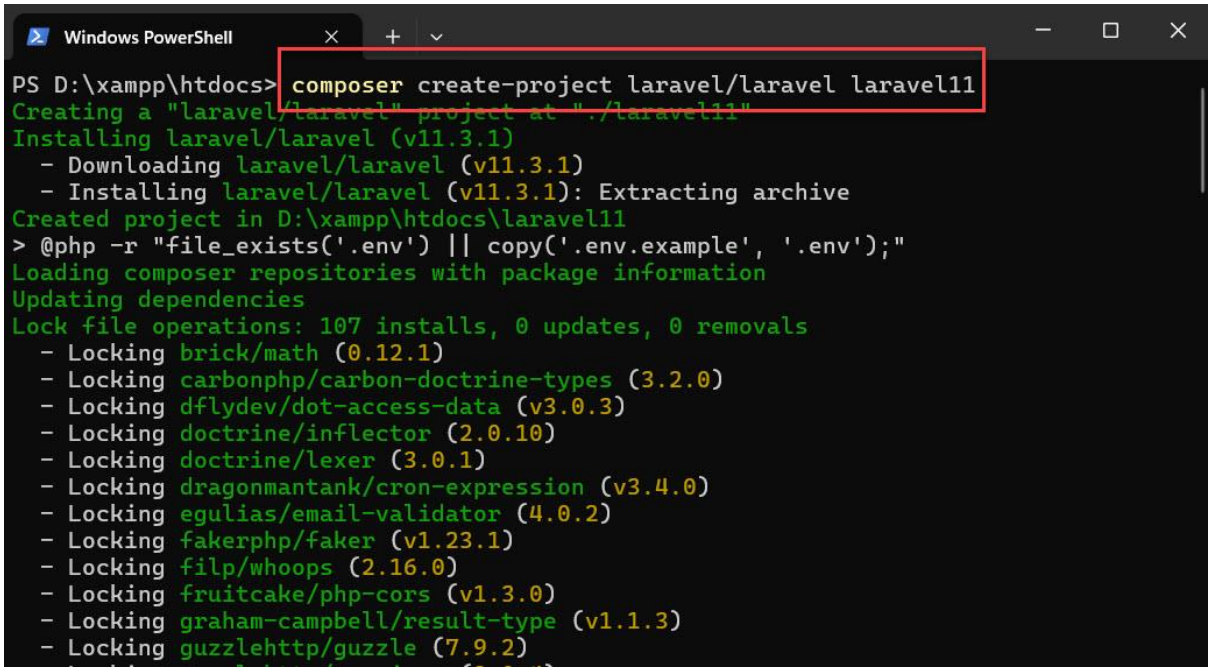
ติดตั้ง Laravel 11

การติดตั้ง Laravel นั้น สามารถทำได้ง่ายๆ ไม่วุ่นวายมากนัก เพียงแค่ใช้คำสั่ง CLI บรรทัดเดียว ก่อนอื่นควรตรวจสอบเครื่องของท่านก่อนว่าเชื่อมสัญญาน Internet หรือยัง อันนี้สำคัญนะ ถ้าไม่เชื่อมสัญญาน Internet ก็จะมีปัญหาตอนติดตั้ง แนะนำให้ติดตั้งผ่านตัว Composer จะสะดวกสำหรับมือใหม่ มาเริ่มกันเลยครับ...

1.ให้เปิด Command Prompt หรือ Windows PowerShell แล้วพิมพ์คำสั่ง cd D:\xampp\htdocs เพื่อเข้าไปในโฟลเดอร์ htdocs ซึ่งเป็นตำแหน่ง root directory เราจะเก็บ Project งานไว้ในนี้นะครับ ตอนติดตั้ง xampp เราติดตั้งไว้ที่ไหนก็ให้เข้าไปในโฟลเดอร์ที่ติดตั้งไว้นะครับ อันนี้ไม่ “งง” นะครับ



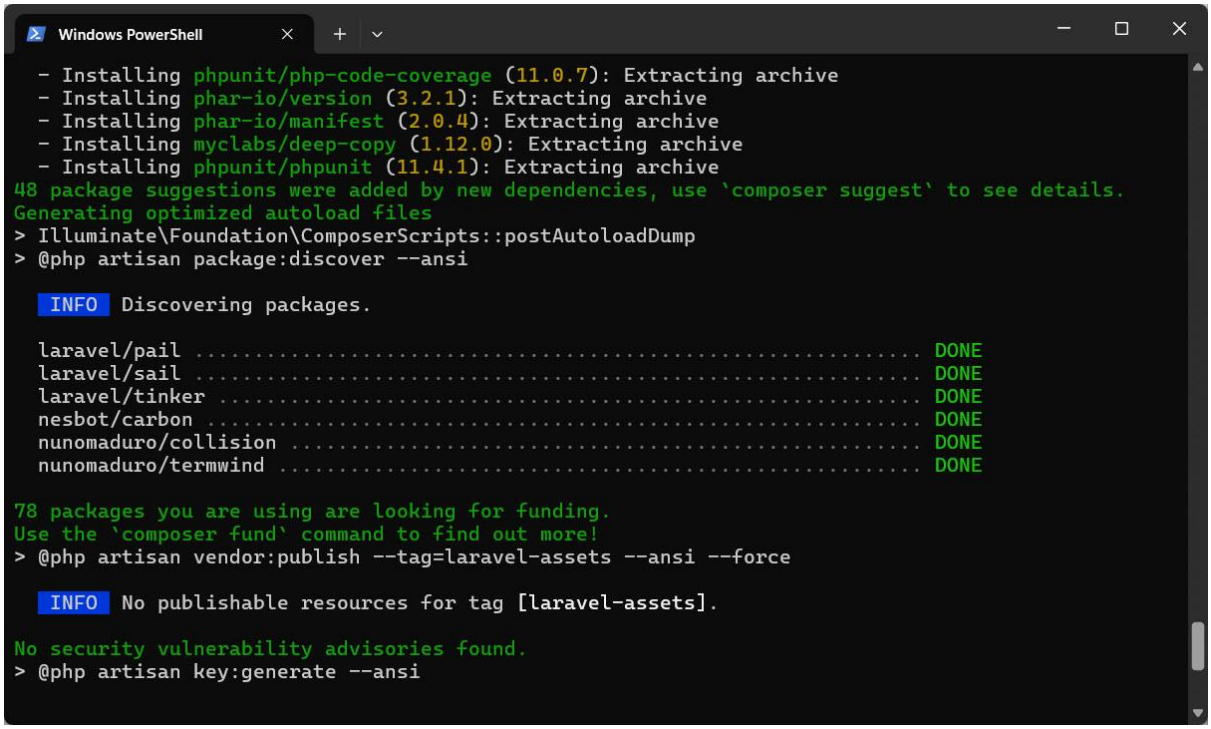
2.พิมพ์คำสั่ง composer create-project laravel/laravel “ชื่อโปรเจคของเรา” แล้วกด Enter



Composer ก็จะทำการดาวน์โหลดไฟล์ package ต่างๆมาติดตั้งให้ รอจนกว่าจะติดตั้งเสร็จ จะใช้เวลาเร็วหรือช้าก็ขึ้นอยู่กับสัญญาน Internet ของเราด้วยนะครับ ว่าสัญญานแรงแค่ไหน ...

จะเห็นว่าผมสร้างโปรเจคชื่อว่า laravel11 ซึ่งเราสามารถตั้งเป็นชื่ออื่นได้ เมื่อเรารอจนติดตั้งเสร็จแล้ว หากไม่เจอปัญหาอะไร ก็จะแสดงเหมือนภาพด้านล่างนี้ ก็ถือว่าเป็นการติดตั้งเสร็จสมบูรณ์

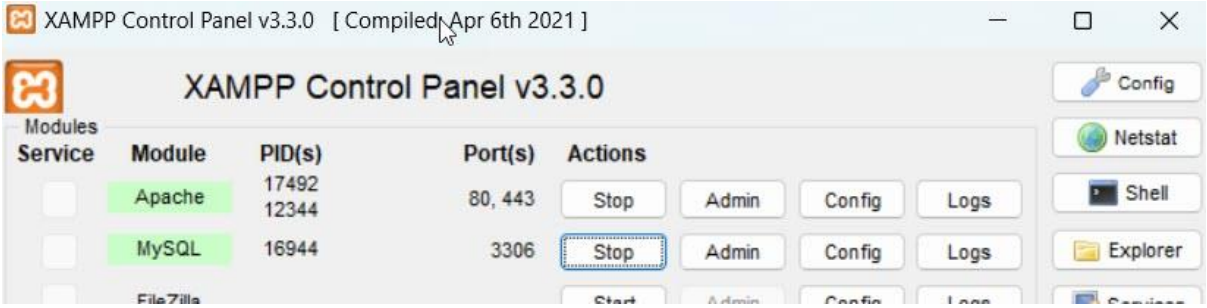
พอเราเข้าไปดูที่ตำแหน่งเก็บโปรเจคเราใน D:\xampp\htdocs ก็จะเห็นโฟลเดอร์ชื่อ laravel11 เก็บอยู่ในนั้น



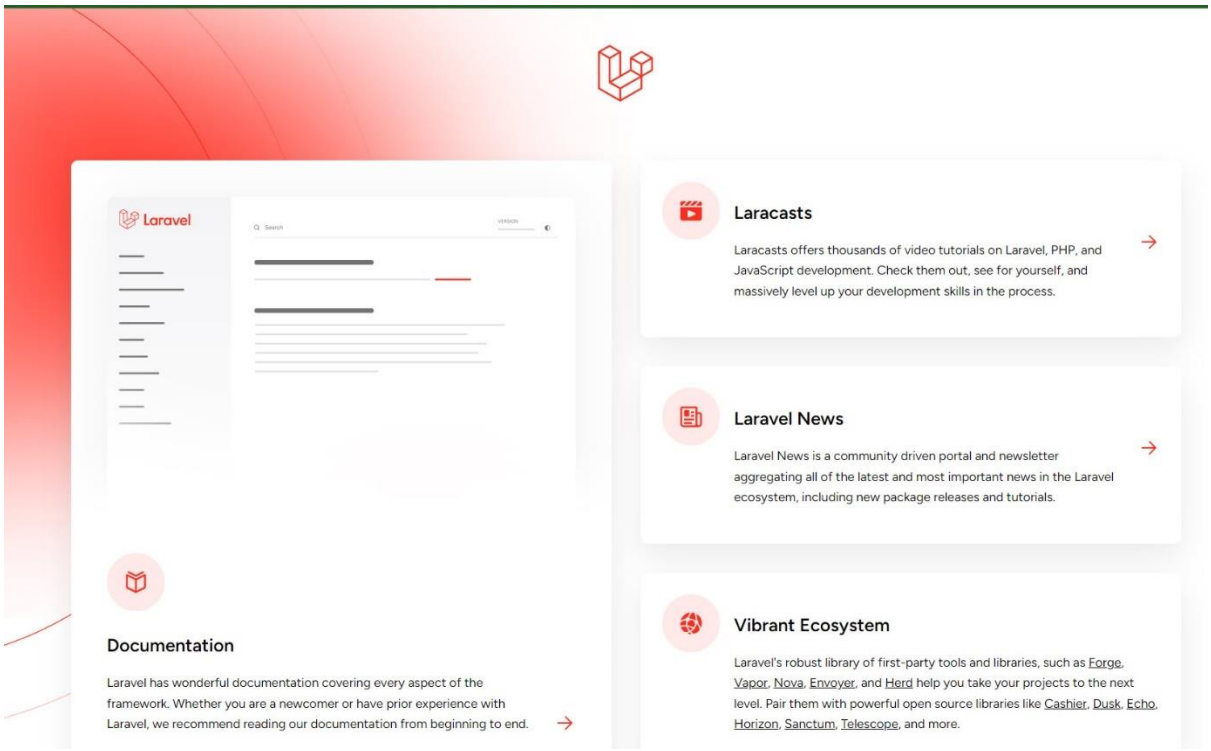
คราวนี้ก็ลองทดสอบดูว่าติดตั้งเรียบร้อยไหม ผ่าน Browser โดยพิมพ์ URL ดังนี้

http://localhost/laravel11/public

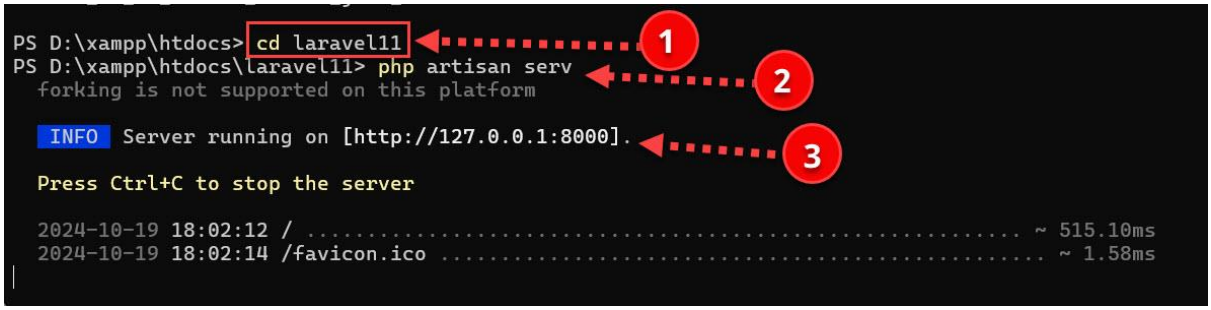
((((อ่อ !!! อย่าลืม Start Apache ด้วยนะครับ !!!))))



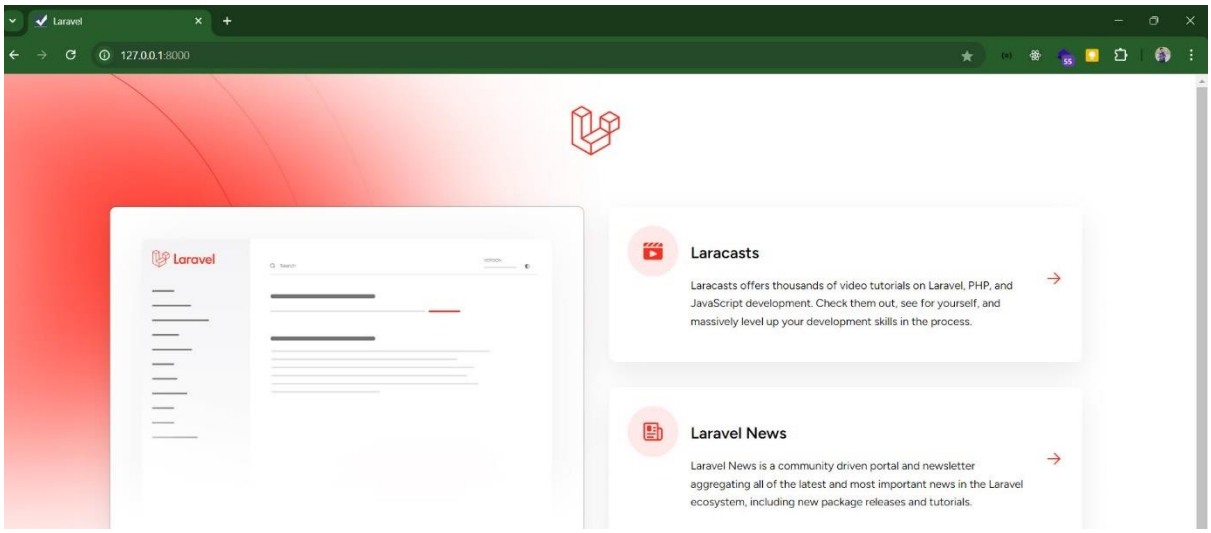
ถ้าไม่ติดปัญหาอะไรก็จะขึ้นหน้าจอต้อนรับตามภาพ เย้ๆ คราวนี้ก็พร้อมจะลุยกันต่อแล้ว 😊



เราสามารถรันโปรเจค Laravel อีกวิธีหนึ่งคือเราสามารถสั่ง ให้ Laravel ทำงานได้ผ่าน Command Prompt หรือ Windows PowerShell ด้วยคำสั่ง php artisan serv ได้ดังนี้ครับ

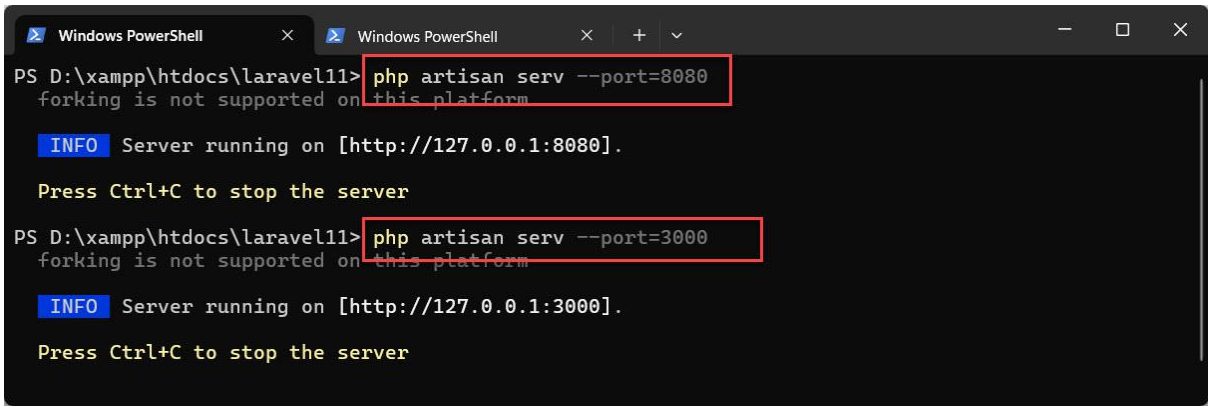


1. พิมพ์คำสั่ง cd laravel11 เพื่อเข้าไปในโปรเจค (ต้อง cd เข้าไปในตำแหน่ง Folder xampp\htdocs ก่อนนะครับ)
2. พิมพ์คำสั่ง php artisan serv เพื่อสั่งให้โปรเจคทำงาน
3. จะขึ้นข้อความ Server running on .....
4. กดปุ่ม ctrl ที่คีย์บอร์ดค้างไว้แล้วใช้เมาส์คลิกที่ [http://127.0.0.1:8000] เพื่อเป็นการเปิด Browser หรือจะพิมพ์ http://127.0.0.1:8000 หรือ http://localhost:8000 ลงใน Browser เลยก็ได้ครับ



เราสามารถกำหนด port การใช้งานเองได้ด้วยการใช้คำสั่ง php artisan serv –port=8080

–port=8080 เป็นการระบุพอร์ตการใช้งานกรณีที่ พอร์ต 8000 ถูกใช้งานไปแล้ว



จะเลือกใช้วิธีไหนก็ได้ตามถนัดหรือชอบเลยครับ ... ในหนังสือนี้ผมจะใช้คำสั่ง php artisan serv เพื่อสั่งรันโปรเจค ตลอดทั้งเล่มนะครับ

ข้อแตกต่างระหว่าง Run โปรเจคผ่าน XAMPP และ คำสั่ง Artisan

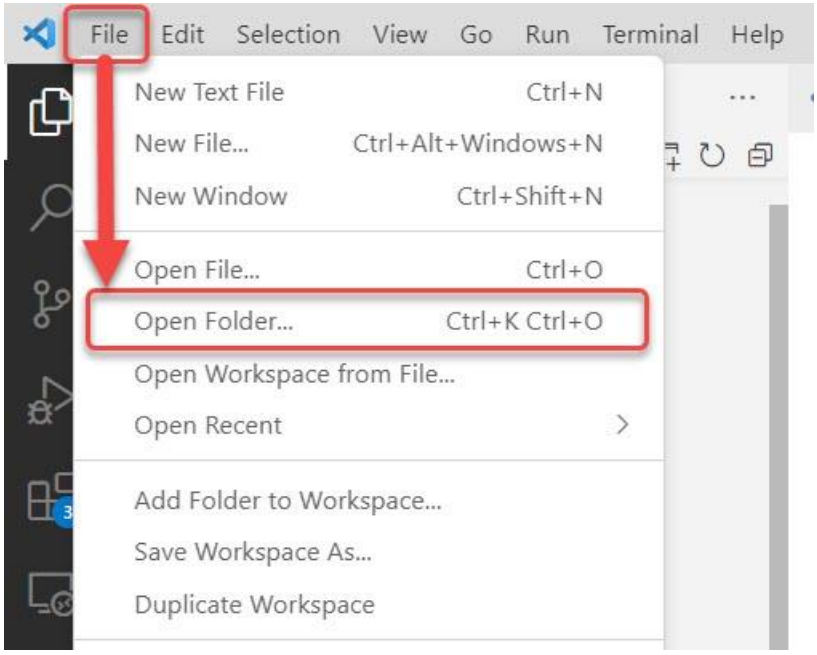
Run ผ่าน XAMPP จะต้องพิมพ์ http://localhost/laravel10/public ซึ่งถ้าสังเกตดี ๆ เราจะพิมพ์ public ต่อท้ายชื่อโปรเจคทุกครั้ง ถามว่าผิดไหม ก็ไม่ผิดครับ ก็ยังใช้ได้แต่เพียงมันทำให้เสียเวลา ถ้าจะเปลี่ยน Port ก็จะวุ่นวายหน่อย หรือบางครั้งถ้าลืม Start Apache โปรเจคเราก็รันต่อไม่ได้

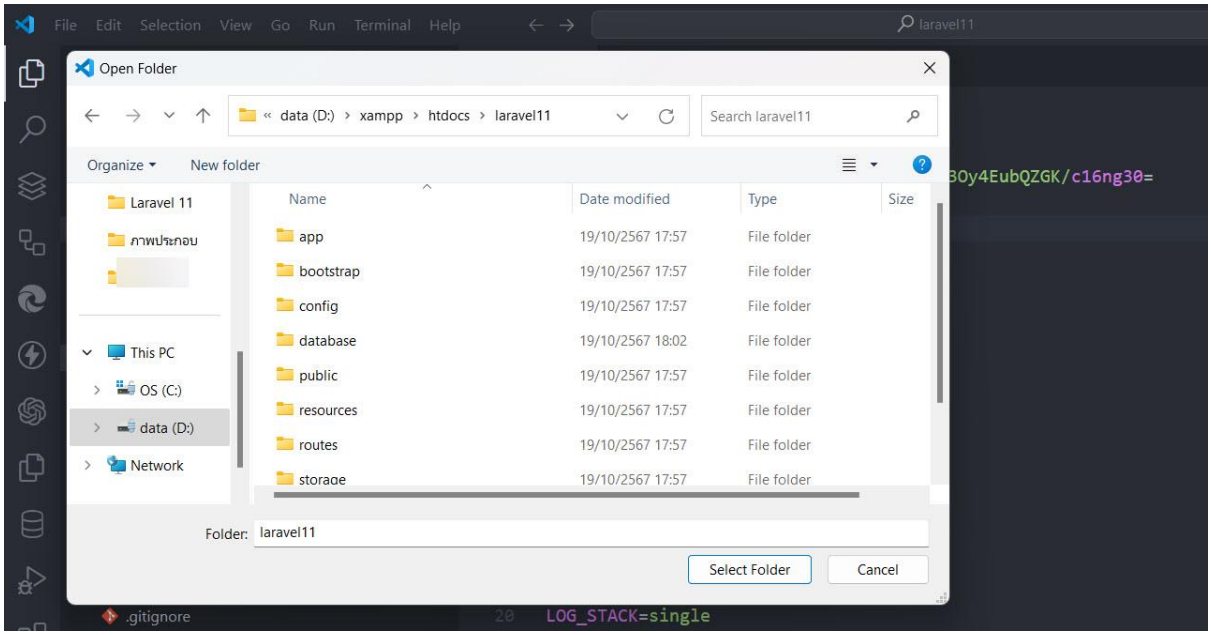
คำสั่ง Artisan ไม่ต้องพิมพ์ URL และไม่ต้องมี public ต่อท้าย พิมพ์แค่ 127.0.0.1:8000 ก็ Run ได้ เปลี่ยน port ได้ง่ายกว่า ไม่ต้อง Start Apache

แนะนำการ Start Project ผ่าน Visual Studio Code

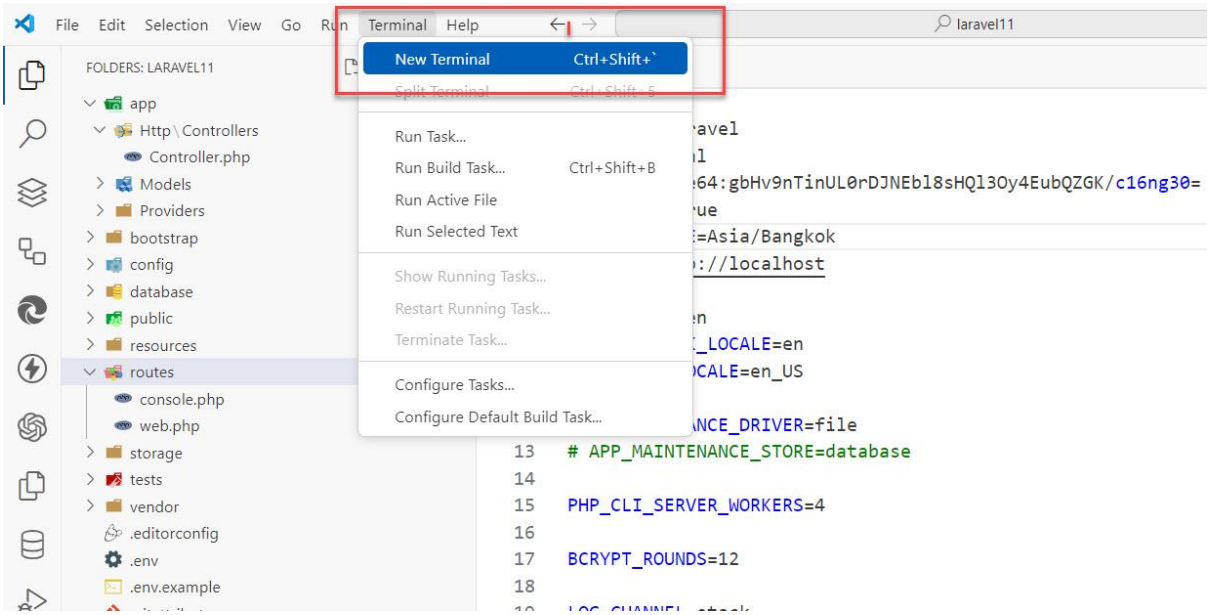
จะเป็นวิธีที่ผมแนะนำให้ทุกท่านใช้ครับ เพราะจะทำให้เราสะดวก ไม่ต้องสลับหน้าจอ กลับไปกลับมาระหว่าง Command Prompt หรือ Windows PowerShell

เปิดโปรแกรม VSCode ขึ้นมาก่อนแล้วคลิกที่เมนู File -> Open Folder... จากนั้นก็เลือกโฟลเดอร์โปรเจคที่เราได้ติดตั้งไว้แล้ว ในที่นี้คือ Laravel11 แล้วกด Select Folder เพื่อเปิดโค้ด

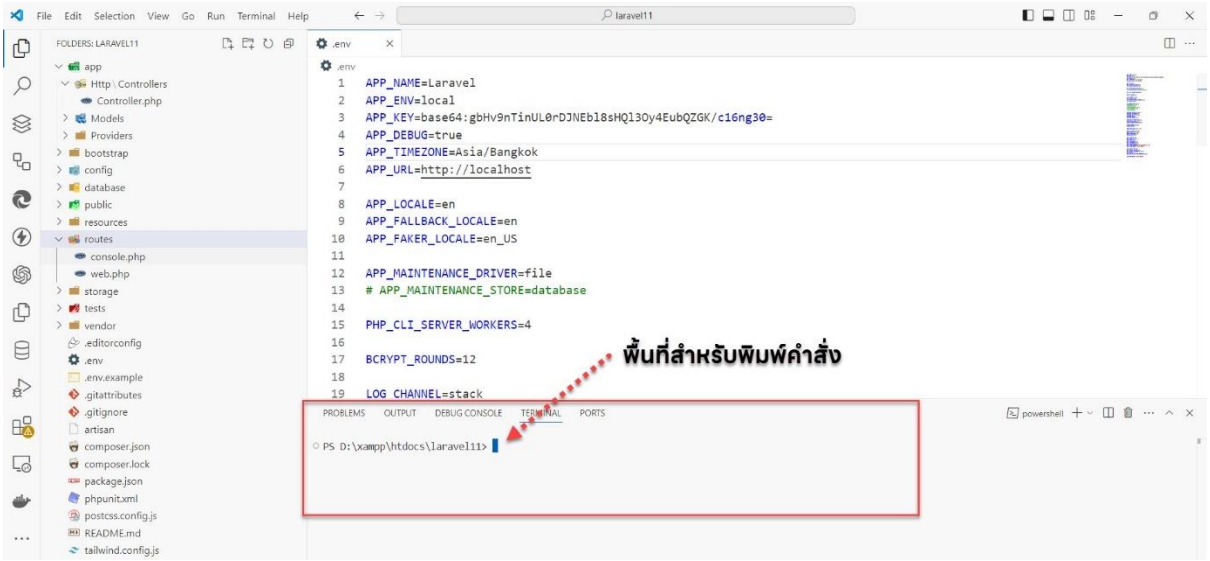




จากนั้นไปที่ เมนู Terminal > New Terminal

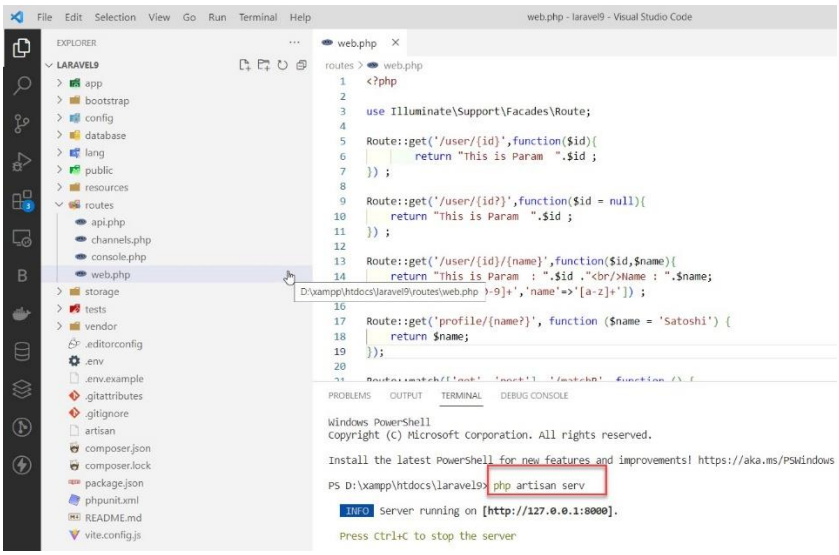


จะได้หน้าต่างสำหรับพิมพ์คำสั่งแสดงออกมาด้านล่าง



พิมพ์คำสั่งสำหรับ Start Server

```
php artisan serv
```

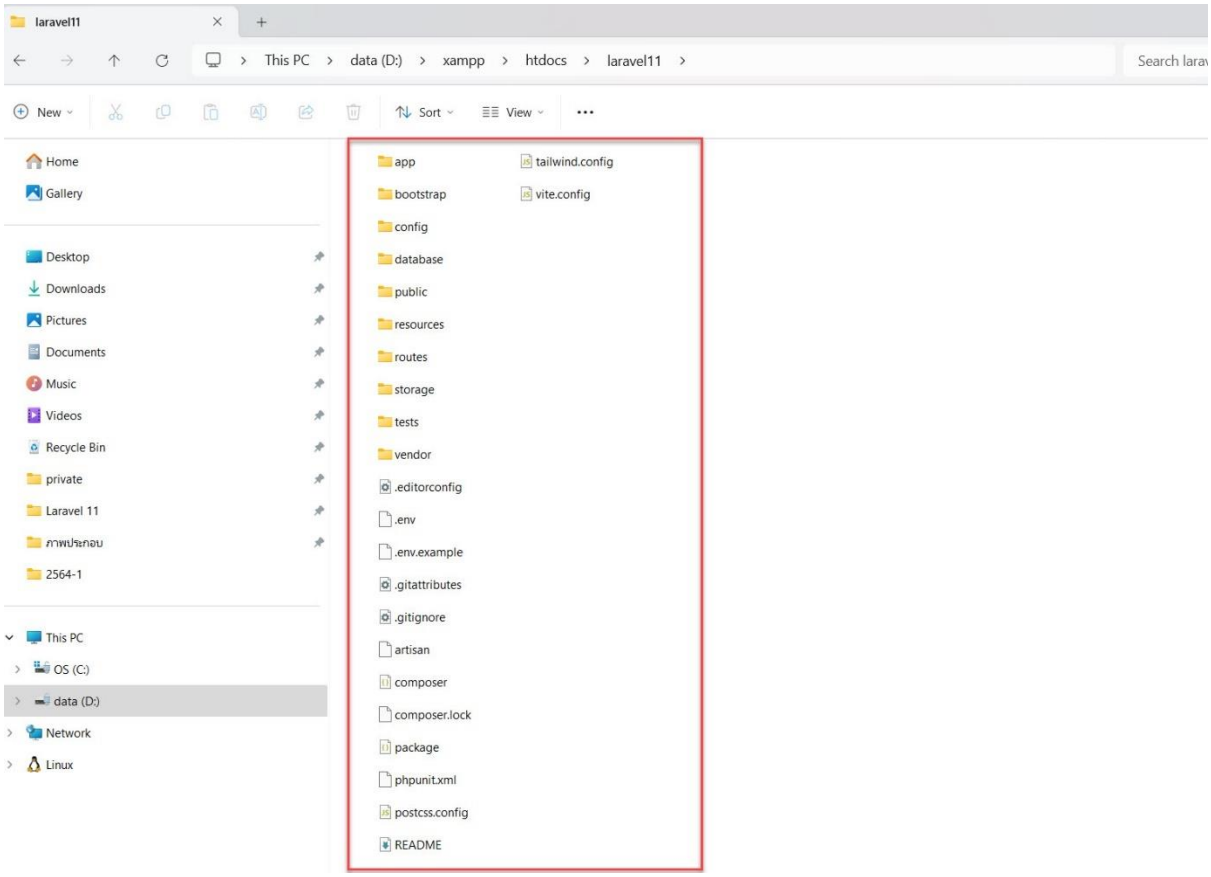


ถ้าต้องการ Stop Server ให้กด Ctrl + C

ให้เป็นที่เข้าใจกันนะครับว่าทุกครั้งที่มีการพิมพ์คำสั่งต่าง ๆ ผมจะพิมพ์ลงใน Terminal ที่มากับ VSCode

โครงสร้างไฟล์ใน Laravel Framework

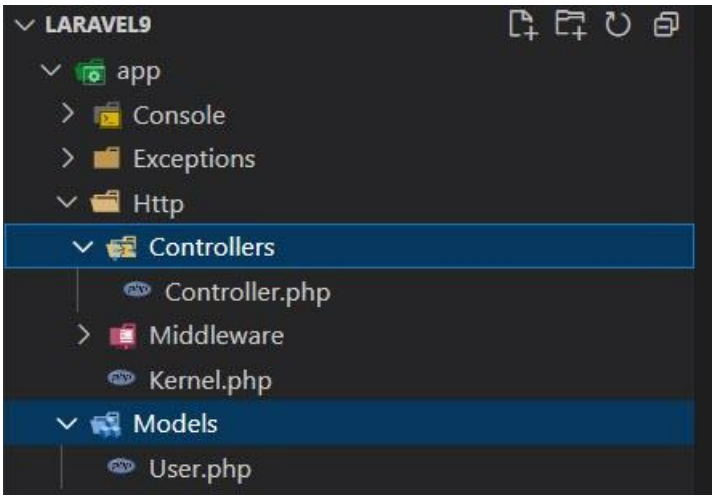
เอาหล่ะบัดนี้ก็มาดูโครงสร้างและทำความรู้จักกันครับว่าใน Laravel 11 มีโครงสร้างแบบไหน แต่ละโฟลเดอร์ แต่ละไฟล์หลักๆคืออะไร มีหน้าที่ทำอะไร ก็จะเห็นโครงสร้างตามภาพ



Directory Structure

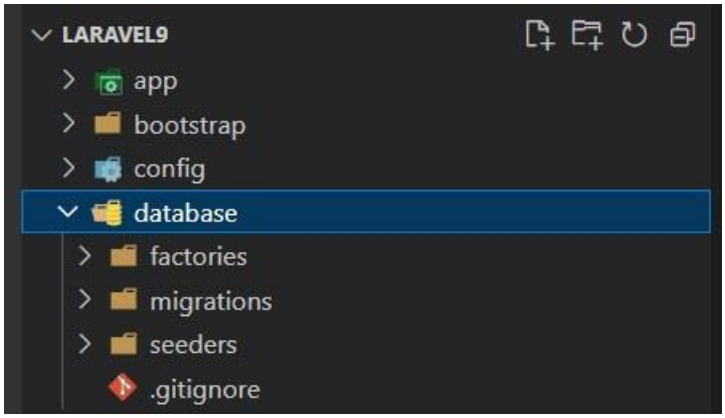
app	- เก็บส่วนหลัก ๆ ของ Project ประกอบด้วย Controller และ Models
bootstrap	- ส่วนของการเก็บค่าของ bootstrap ไม่ใช่ CSS นะ และค่า Config ต่าง ๆ
config	- เก็บส่วนการตั้งค่าต่าง ๆ ของโปรเจค
database	- เก็บส่วนที่จัดการเกี่ยวกับฐานข้อมูล เช่น การทำ Migration และการ Seed Data
public	- เป็นส่วนที่เก็บไฟล์เริ่มต้นของ Project คือ index.php และจะเก็บไฟล์พวก JavaScript , CSS , Image
resources	- เก็บส่วนของ View ทั้งหมด
routes	- เก็บส่วนของไฟล์ที่ใช้สำหรับกำหนดเส้นทางของเว็บ หรือ เส้นทางของ URL
storage	- เก็บส่วนของไฟล์ session , cache ต่าง
test	- ส่วนของ automated test
vendor	- อันนี้ไม่มีไม่ได้ จะใช้เก็บ Library หรือ Packege ต่าง ที่ติดตั้งผ่าน composer
.env	- ไฟล์ที่จำเป็นต้องมี ใช้ในการตั้งค่าส่วนต่าง ๆ ของ Project

Directory และ Files ที่ใช้บ่อย



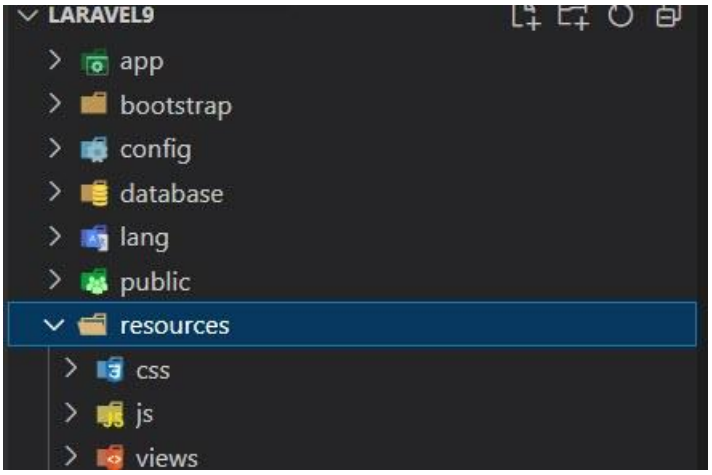
app ใน Folder นี้ประกอบด้วย

- app / Http / Controllers สำหรับเก็บไฟล์ controller
- app / Http / Middleware สำหรับเก็บไฟล์คัดกรอง request ที่เข้ามาภายในแอปของเรา เช่น การกำหนดการเข้าถึง URL ต่างๆ
- app / Http / Models สำหรับเก็บไฟล์ model



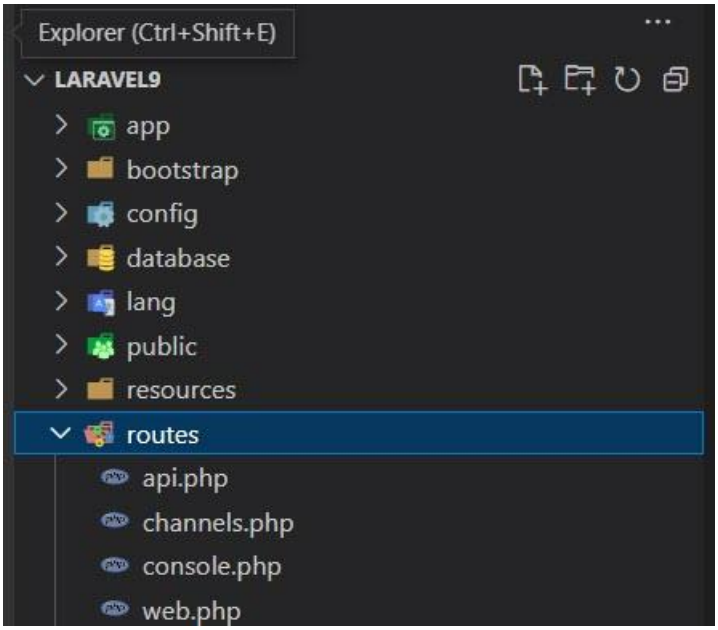
database ใน folder นี้ประกอบด้วย

- database / factories ช่วยสร้าง Model Object สำหรับการปั้นข้อมูลให้สามารถนำมาใช้งานได้ง่ายขึ้นเพียงเรียก method ที่ชื่อ create
- database / migrations ช่วยสร้าง table ของข้อมูลที่ต้องการ
- database seeders เป็นอีกคลาสหนึ่งที่ใช้งานร่วมกับ Factory ซึ่งเอาไว้สำหรับสร้างข้อมูลทดสอบ ลงในฐานข้อมูลตามที่ ซึ่งตามที่เราต้องการ



resources ใน folder นี้ประกอบด้วย

- views เก็บส่วนของไฟล์ template , layouts , html , php ทั้งหมด

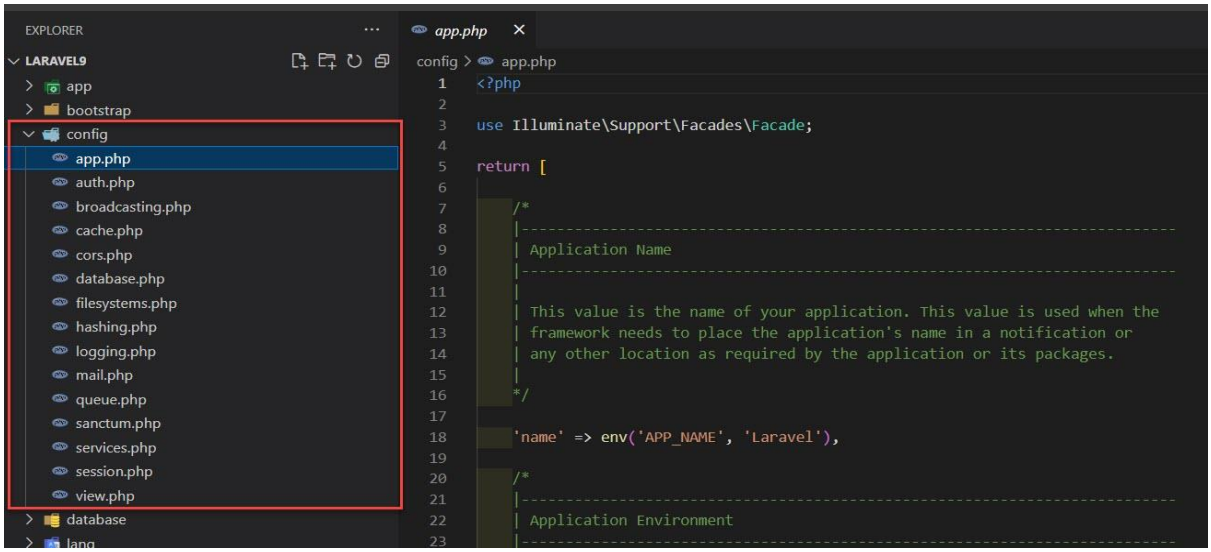


routes ใน folder นี้ประกอบด้วย

- api.php สำหรับใช้ทำ api
- web.php สำหรับสร้างเส้นทางการเข้าถึงภายใน application ของเรา หรือ การกำหนด URL ต่างๆ

การตั้งค่าระบบของ Laravel

หลังจากที่เราทำการติดตั้ง Laravel เรียบร้อยแล้ว ก็ควรจะตั้งค่าต่างๆ ให้เรียบร้อยก่อน โดยให้เราเปิดโฟลเดอร์ config ขึ้นมา



ไฟล์ตั้งค่าที่สำคัญๆ ได้แก่

- app.php เป็นไฟล์สำหรับการตั้งค่าภาพรวมของระบบทั้งหมด เช่น กำหนดการเปิดหรือปิด ของ Debug Mode , การกำหนด timezone ให้กับ Web Application เช่น

- auth.php เป็นไฟล์สำหรับการตั้งค่าเกี่ยวกับการ Login การรับรองหรือตรวจสอบผู้ใช้ เช่น การกำหนดตารางผู้ใช้ในฐานข้อมูล , การตั้งค่าเกี่ยวกับการ reset รหัสผ่าน

- broadcasting.php เป็นไฟล์สำหรับการตั้งค่าเกี่ยวกับการเชื่อมต่อระบบ Broadcast

- cache.php เป็นไฟล์สำหรับการตั้งค่าของ cache รองรับ cache ได้หลายตัว ได้แก่ filesystem , database , mem-cached , redis ค่า default ที่ Laravel กำหนดมาให้จะเป็น filesystem

- database.php เป็นไฟล์สำหรับการตั้งค่าเกี่ยวกับการเชื่อมต่อฐานข้อมูล MySQL/MariaDB , sqlite , pgsql , sql server แนะนำให้กำหนดค่าต่างๆ ที่เกี่ยวกับการเชื่อมต่อฐานข้อมูลที่ไฟล์ .env

- filesystems.php เป็นไฟล์สำหรับการตั้งค่าและกำหนดปลายทางของระบบไฟล์ในโปรเจคของเรา

- hashing.php เป็นไฟล์สำหรับการตั้งค่าเกี่ยวกับการเข้ารหัสสำหรับการกำหนดรหัสผ่าน ซึ่งจะรองรับการเข้ารหัสแบบ bcrypt , argon , argon2id โดยค่าปกติที่กำหนดมาให้คือ การเข้ารหัสที่ใช้อัลกอลิทึม bcrypt

- mail.php เป็นไฟล์สำหรับการตั้งค่าเกี่ยวกับอีเมลว่าจะใช้รูปแบบไหนในการส่งอีเมล รองรับ smtp , mail , sendmail , mailgun , sparkpost , ses , log

- queue.php เป็นไฟล์สำหรับการเชื่อมต่อกับ API ค่าปกติที่กำหนดไว้ คือ sync รองรับ "database", "beanstalkd", "sqs", "redis", "null"

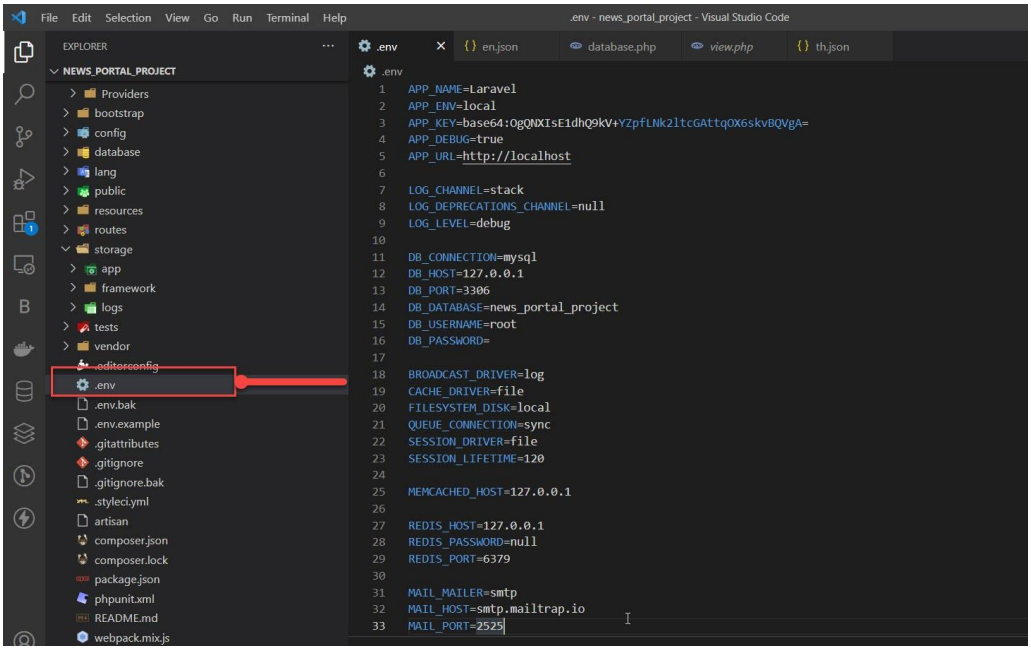
- services.php เป็นไฟล์สำหรับการตั้งค่าและกำหนดบริหารของ third-party ต่าง เช่น Stripe ใช้เป็น geteway สำหรับการชำระเงิน

- session.php เป็นไฟล์สำหรับการตั้งค่าระบบ Session ของ PHP สามารถกำหนดได้หลายแบบว่า

จะใช้แบบไหน ได้แก่ cookie , database , file , apc , redis และ array

- view.php เป็นไฟล์สำหรับการตั้งค่าที่อยู่หรือ path สำหรับ view

การตั้งค่าตามสภาพแวดล้อม (.env)



.env ไฟล์ จะใช้สำหรับการตั้งค่าตามสภาพแวดล้อม เช่น การตั้งชื่อแอบพลิเคชันของเรา การตั้งค่าการเชื่อมต่อฐานข้อมูล การส่งอีเมล ซึ่งจะมีโค้ดที่ต้องแก้ไขตามนี้ครับ

APP_NAME=Laravel (ชื่อเว็บหรือระบบงานของเรา)

APP_ENV=local (โหมดการพัฒนาถ้าเอาขึ้น Server จริงให้เปลี่ยนเป็น production)

APP_DEBUG=true (เปิดโหมด debug หากเอาขึ้น Production ให้ false)

APP_TIMEZONE=Asia/Bangkok (สำหรับกำหนดโซนเวลา)

APP_URL=http://localhost (URL ระบบ ของเรา)

DB_CONNECTION=mysql (ชื่อยี่ห้อของฐานข้อมูลที่เราใช้)

DB_HOST=127.0.0.1 (HOST)

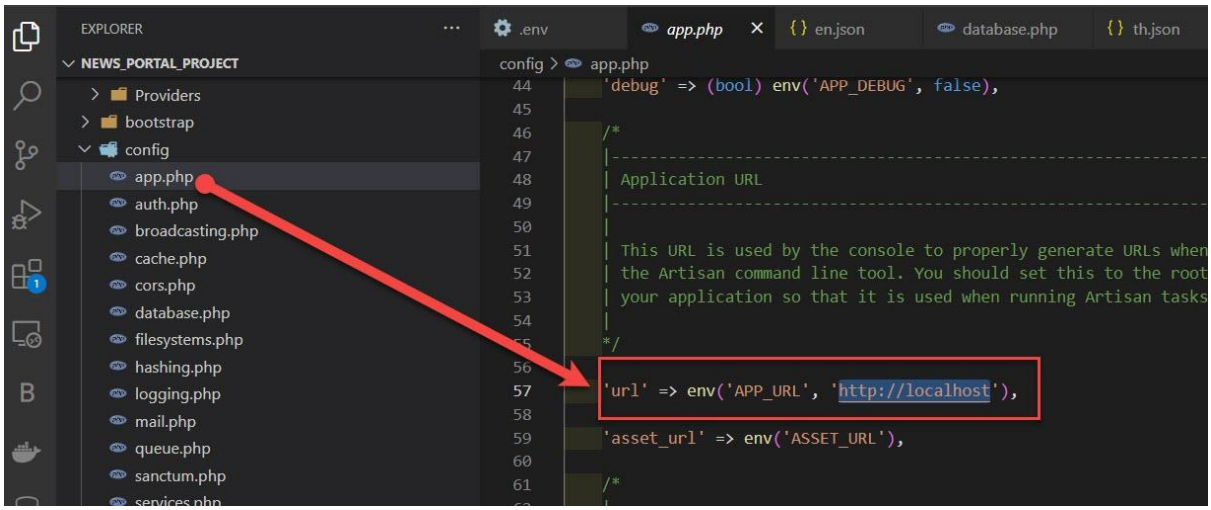
DB_PORT=3306 (พอร์ต ฐานข้อมูล)

DB_DATABASE=homestead (ชื่อฐานข้อมูล)

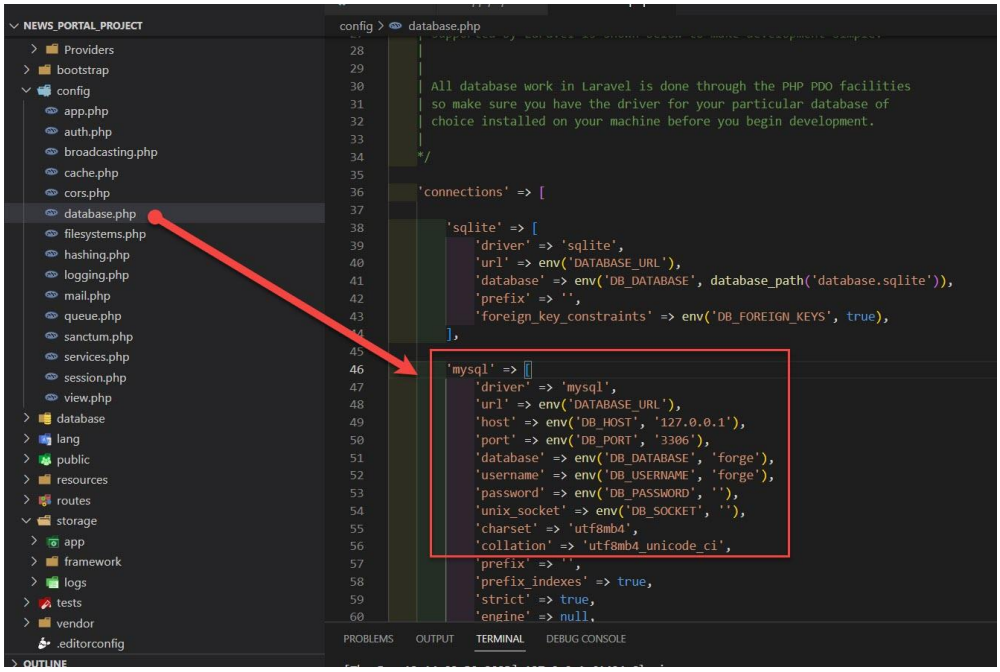
DB_USERNAME=homestead (user ฐานข้อมูล)

DB_PASSWORD=secret (รหัสผ่านฐานข้อมูล)

ตอนที่เราเอาโปรเจคขึ้น Server (Production) แล้ว เราไม่ควรนำไฟล์ .env ขึ้นไปด้วย ให้เราไปตั้งค่าที่ไฟล์ อยู่ในโฟลเดอร์ config แทน เช่น เราต้องการแก้ไข path URL ให้เข้าไปแก้ที่ config/app.php



หากต้องการแก้ไขการเชื่อมต่อฐานข้อมูล ให้ไปแก้ไขไฟล์ database.php ในโฟลเดอร์ config



หลังจากที่เราตั้งค่าต่างๆ เสร็จแล้ว ไม่ว่าจะเป็นตอนเราพัฒนาหรือตอนเราเอาขึ้น Production ให้เปิด Command line แล้วพิมพ์คำสั่งด้านล่าง เพื่อเคลียร์ config เก่า แล้วใช้ config ที่เราเพิ่งตั้งค่าใหม่แทน

```
php artisan config:clear
```

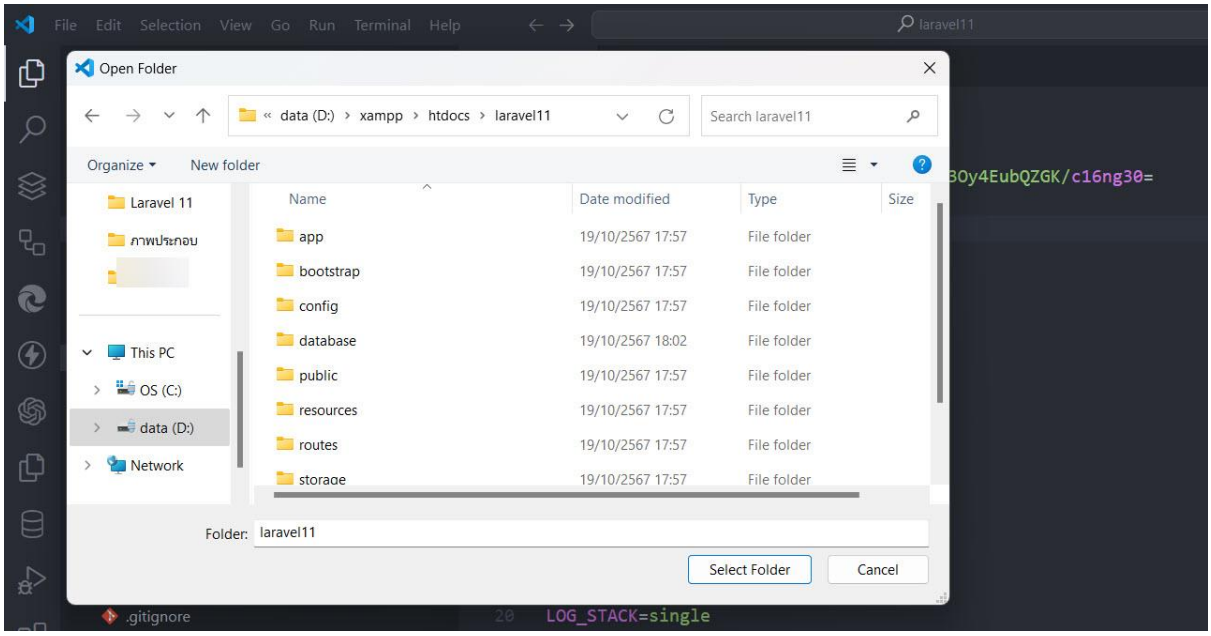
และพิมพ์คำสั่งด้านล่างเพื่อเก็บ cache config

```
php artisan config:cache
```

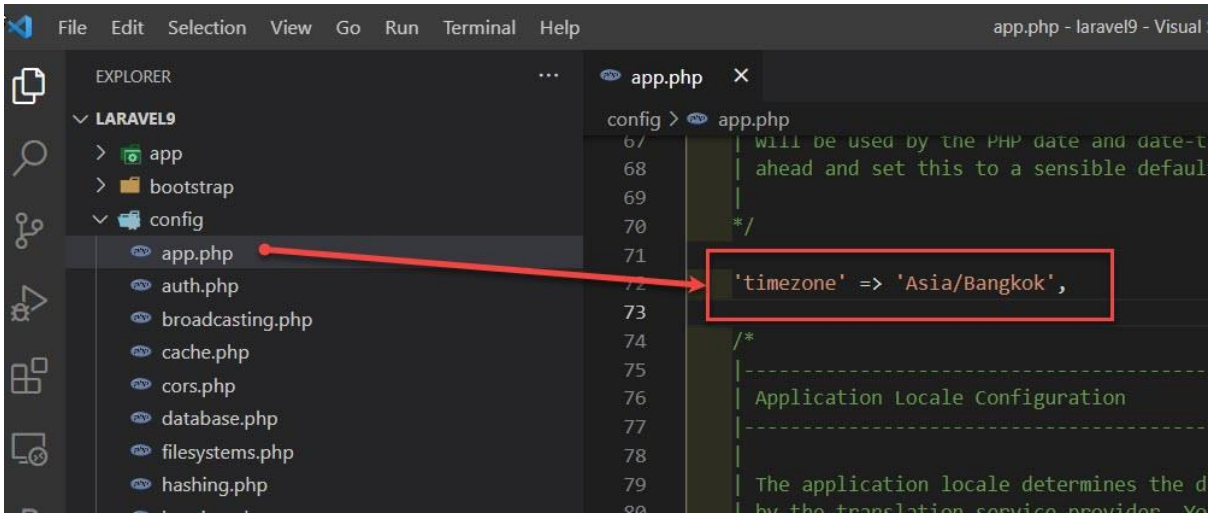
โปรดจำไว้ว่า หากเรากำลังพัฒนาอยู่ให้ตั้งค่าที่ไฟล์ .env แต่หากเราเอาขึ้น Production แล้ว ต้องไปตั้งค่าที่ โฟล์เดอร์ config แทน ...

การตั้งค่า timezone

เพื่อให้ง่ายต่อการแก้ไขโค้ดให้เราเปิดโปรแกรม VSCode ขึ้นมาก่อนแล้วคลิกที่เมนู File -> Open Folder... จากนั้นก็เลือกโฟลเดอร์โปรเจคที่เราได้ติดตั้งไว้แล้ว ในที่นี้คือ Laravel11 แล้วกด Select Folder



เปิดไฟล์ app.php ในโฟลเดอร์ config/app.php แล้วแก้ไขค่า timezone จาก UTC เป็น Asia/Bangkok เพื่อให้แสดงวันที่และเวลาปัจจุบันได้ถูกต้องตามโซนของประเทศไทย

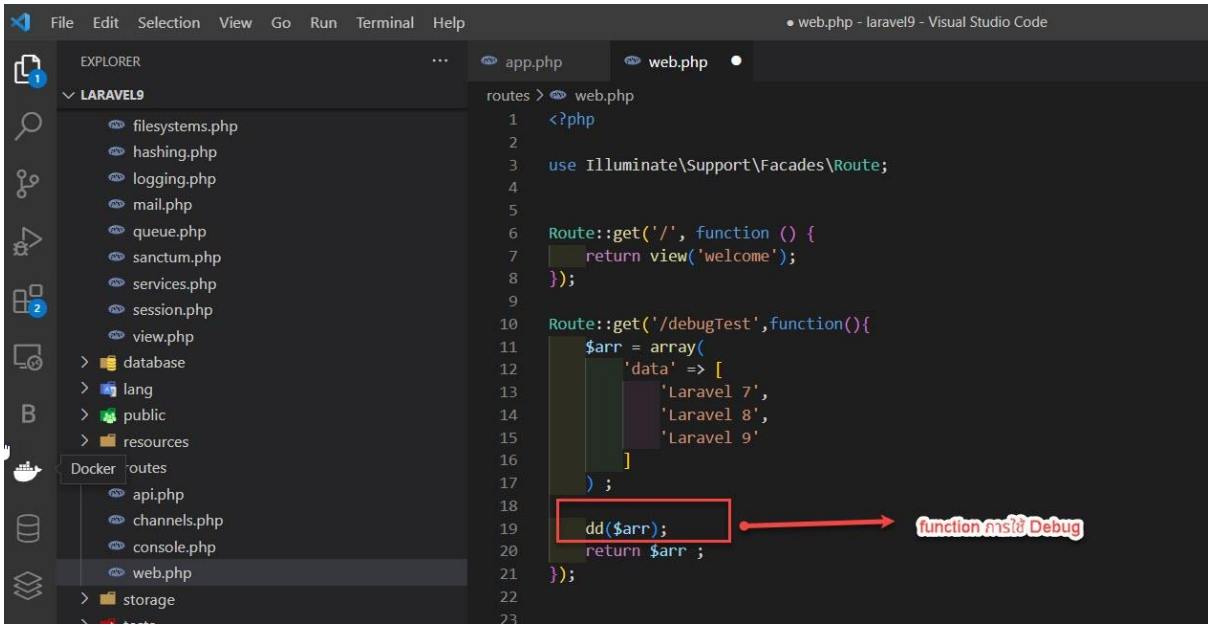


สามารถกำหนดค่า timezone ผ่านไฟล์ .env ก็ได้ โดยแก้ไขจากตัวแปร APP_TIMEZONE=Asia/Bangkok

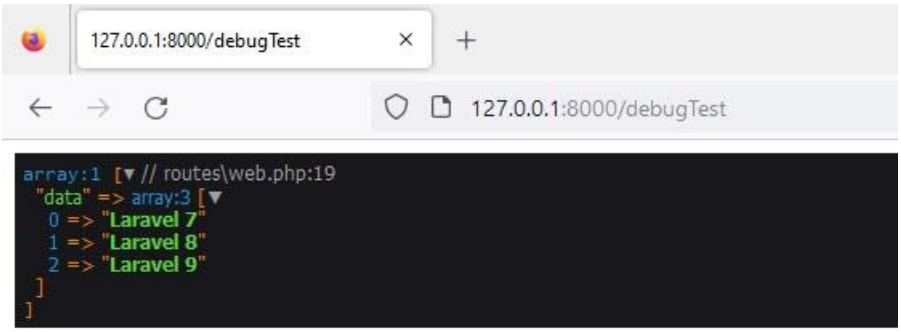
Debugging ใน Laravel

การ Debug โค้ดใน Laravel นั้นเป็นสิ่งที่สำคัญมาก เราจะต้องรู้จักวิธีการใช้งานและเรียนรู้การ Debug ให้เข้าใจ เพราะเวลาที่เราเขียนโค้ดนั้น การจะดูข้อผิดพลาดที่เกิดขึ้นกับโปรแกรมนั้น ไม่ว่าจะเกิดจากเขียนโค้ดผิดรูปแบบ ตรรกะ หรืออะไรก็ตาม ปกติแล้วเราอาจจะใช้คำสั่ง Console.log() ในการดู แต่ใน Laravel จะมีวิธีการ Dubug โค้ดง่ายมาก ซึ่งสามารถทำได้หลายวิธี อาจจะใช้ function ที่มีมากับ Laravel หรือการติดตั้ง Library เพื่ออำนวยความสะดวกในการใช้งาน function ที่ผมชอบใช้เป็นประจำก็จะมี function ที่ชื่อว่า dd() ครับ

เราอยากจะ Debug ดูตัวแปรไหน ก็ใช้ ฟังก์ชัน dd() ได้เลยครับ ลองดูโค้ดด้านล่างนี้ดูครับ



จะเห็นได้ว่า เมื่อเราใช้ ฟังก์ชัน dd() ในการ Debug โค้ด ข้อดีของ dd() คือ มันจะหยุดหรือจบการทำงานที่ฟังก์ชัน dd() ทันที จะไม่ทำส่วนอื่นต่อ ผลลัพธ์ที่ได้ก็จะแสดงให้เราเห็นผ่านหน้า Browser แบบนี้ครับ

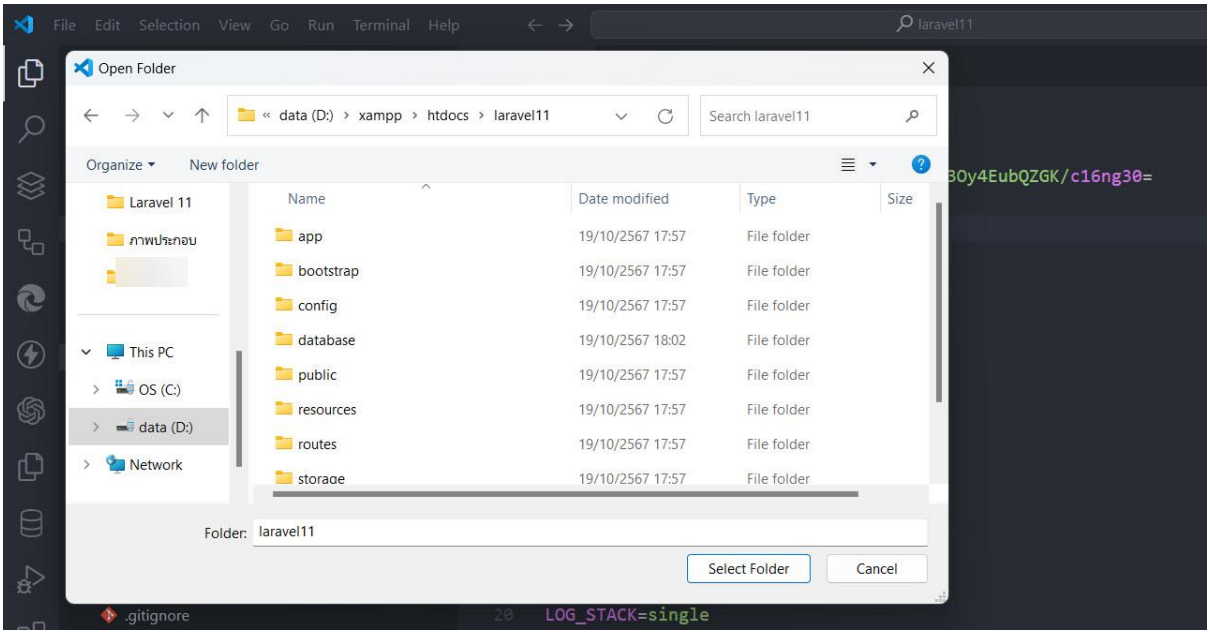


หรือสามารถใช้ print_r() หรือ dump() แทนก็ได้ครับ

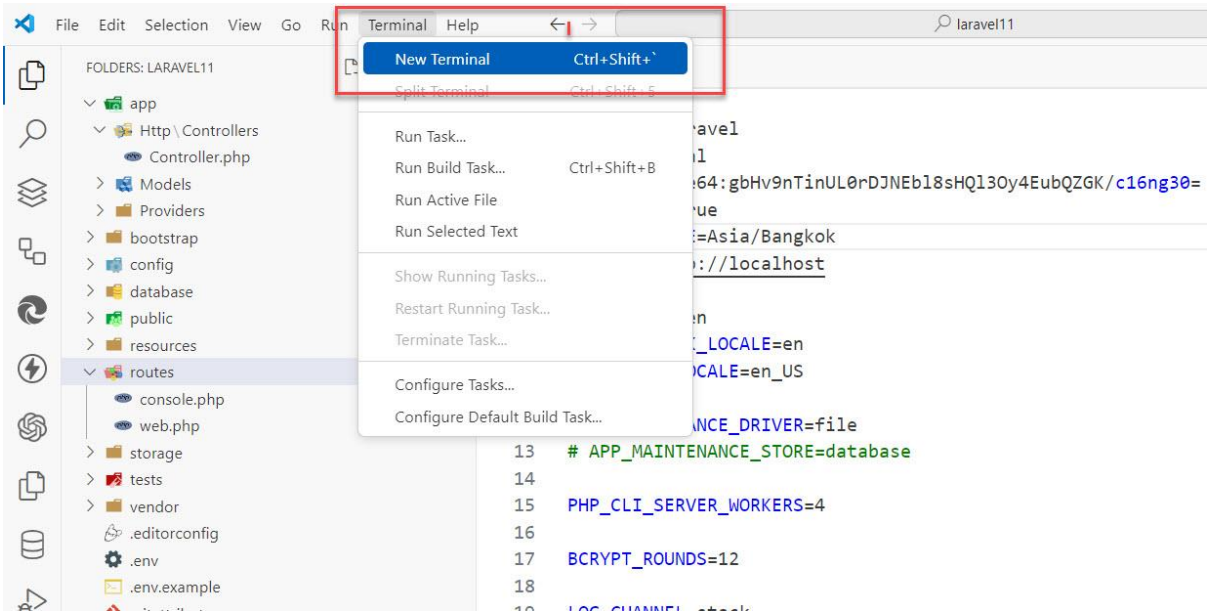
อีกวิธีหนึ่งคือการติดตั้ง Library แทนก็ได้ครับ ซึ่ง Library ที่ผมใช้เป็นประจำก็คือ Laravel Debugbar ครับ ข้อดีของการใช้ Laravel Debugbar คือ สามารถดูรายละเอียดข้อผิดพลาด ต่างๆได้ง่ายมากๆ โดยสามารถดูผ่าน Browser ได้เลย แนะนำตัวนี้เลยครับ

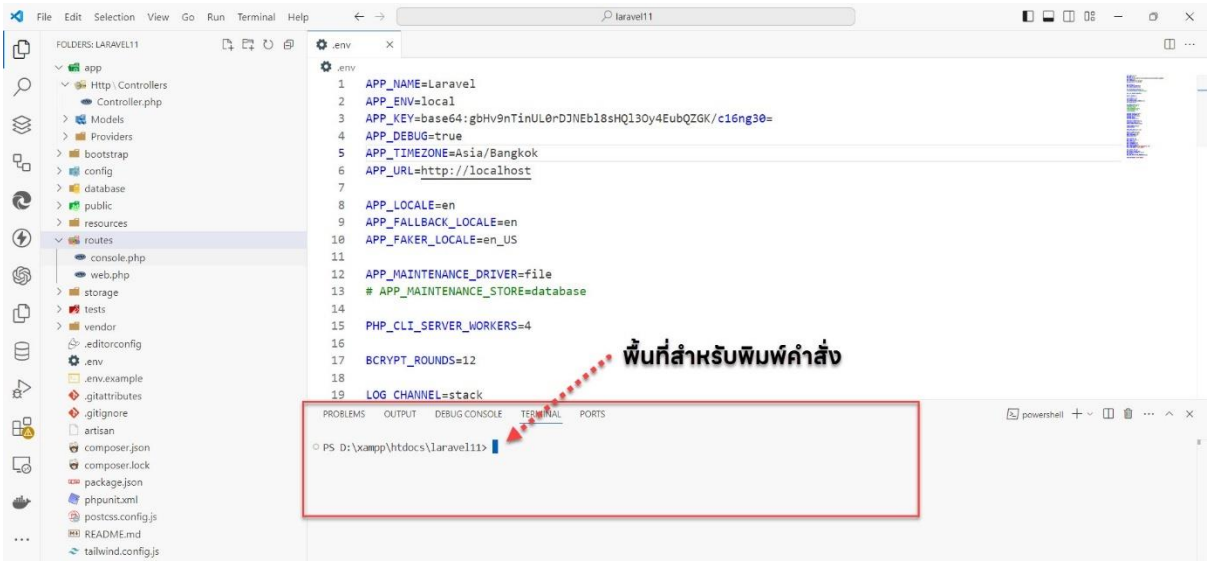
การติดตั้งและใช้งาน Laravel Debugbar ให้ทำตามขั้นตอนดังนี้ครับ

เปิดโปรแกรม VSCode ขึ้นมาก่อนแล้วคลิกที่เมนู File -> Open Folder... จากนั้นก็เลือกโฟลเดอร์โปรเจคที่เราได้ติดตั้งไว้แล้ว ในที่นี้คือ Laravel9 แล้วกด Select Folder



จากนั้น ให้ไปที่เมนู Terminal -> New Terminal ตามภาพครับ เราจะใช้งาน Terminal ที่อยู่ใน VSCode ทุกครั้งนะครับ ทุกครั้งที่ผมบอกว่าให้พิมพ์คำสั่งต่างๆ ก็ให้ท่านพิมพ์คำสั่ง ใน Terminal ที่อยู่ใน VSCode ได้เลยนะครับ จะได้ไม่ต้องเปิดหน้าจอหลายๆ หน้าจอ

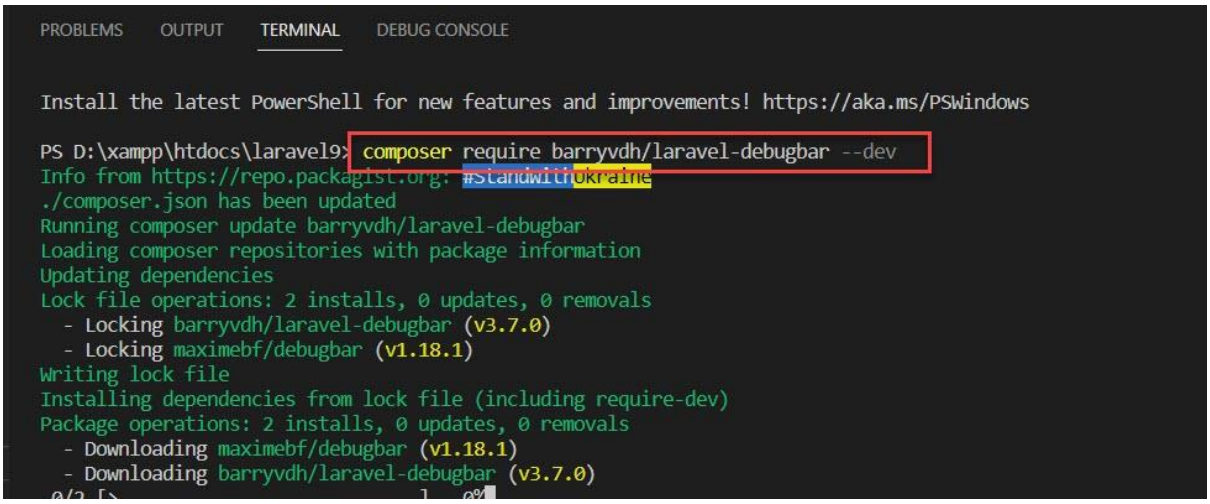




พิมพ์คำสั่งนี้แล้วกด Enter แล้วรอจนติดตั้งเสร็จ

```
composer require barryvdh/laravel-debugbar -dev
```

สามารถเข้าไปดูวิธีการใช้งานต่างๆได้จากลิงค์นี้ได้เลยครับ https://laravel-news.com/laravel-debugbar



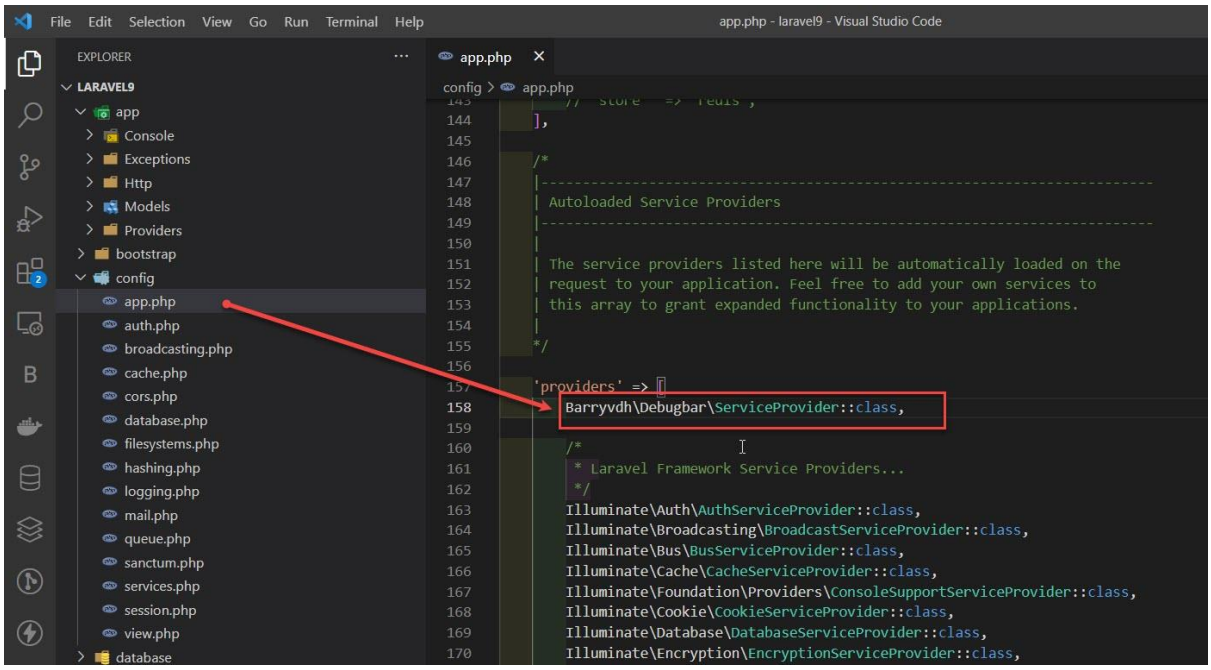
เมื่อติดตั้งเสร็จแล้ว ให้ไปเพิ่ม ServiceProvider ใน config/app.php ในส่วน Array ของ providers

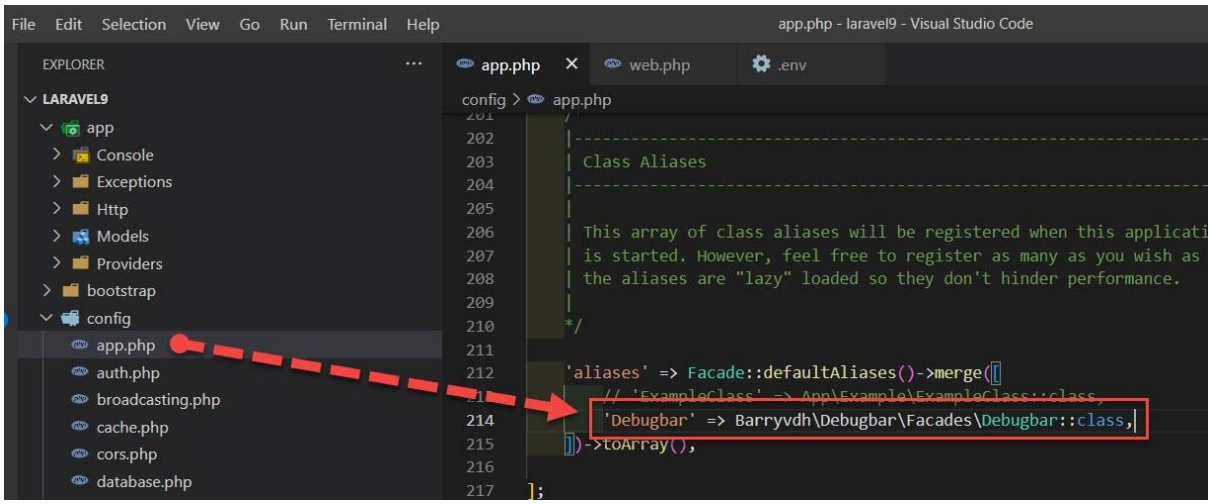
เพิ่มคำสั่งนี้เข้าไป ตามนี้ครับ

```
Barryvdh\Debugbar\ServiceProvider::class,
```

และเพิ่มคำสั่งนี้เข้าไปอีก จะเป็นชื่อสำหรับเรียกใช้งาน เพิ่มเข้าไปในส่วน Array ของ aliases

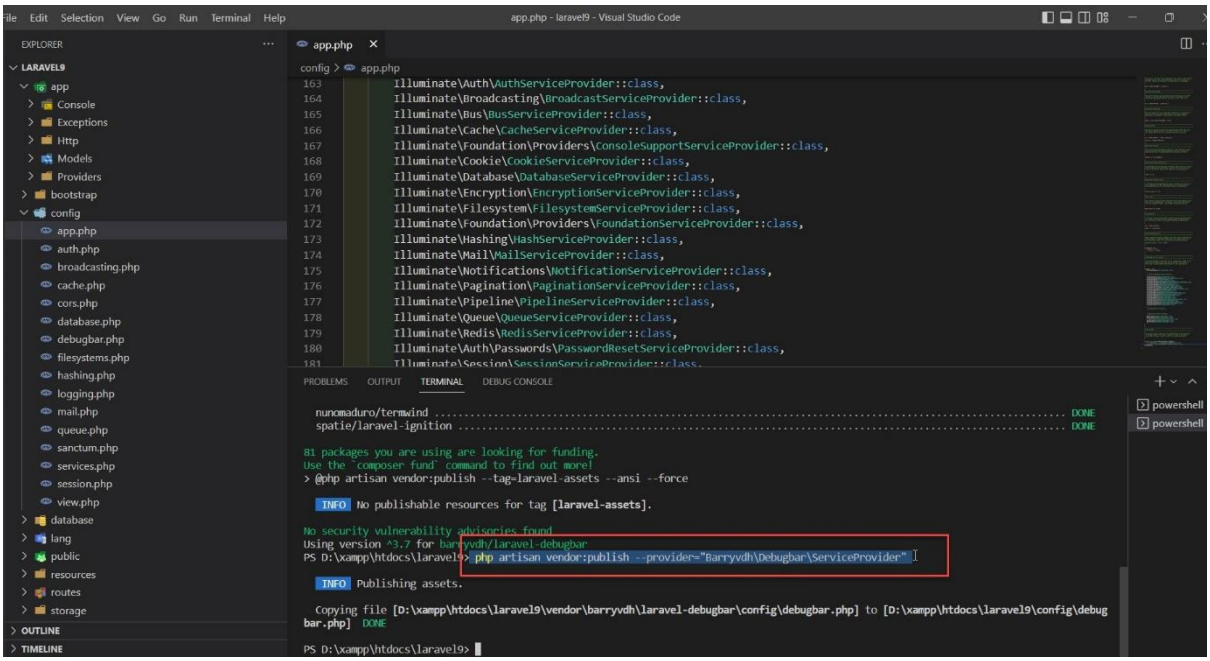
```
'Debugbar' => Barryvdh\Debugbar\Facade::class,
```





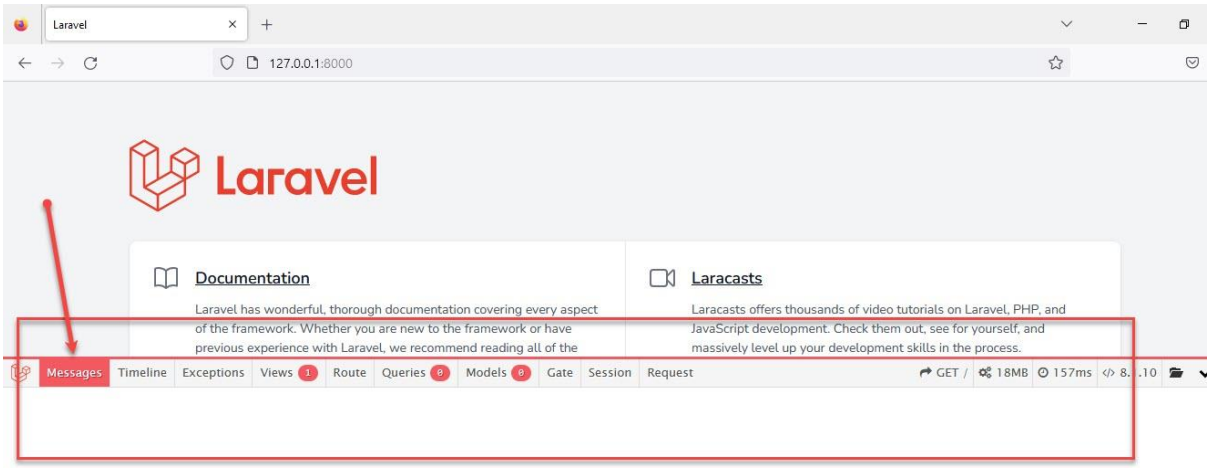
ทุกครั้งที่แก้ไข อย่าลืมกด Save ด้วยนะครับ จากนั้นก็พิมพ์คำสั่งด้านล่างนี้ ใน Terminal

```
php artisan vendor:publish --provider="Barryvdh\Debugbar\ServiceProvider"
```



ก็จะได้ไฟล์เพิ่มเข้ามาใหม่ใน config.php ชื่อว่า debugbar.php เป็นอันเสร็จขั้นตอนการติดตั้งแล้ว

วิธีการดูก็ให้เราเปิดดูผ่าน Browser ได้เลยครับ โดยจะมีแถบเพิ่มเข้ามาให้เราดูตามภาพนี้เลยครับ ซึ่งตัว Library Laravel Debugbar จะช่วยอำนวยความสะดวกในการดู Error ต่าง ๆ ให้เราได้อย่างมาก



รายละเอียดต่างๆ ที่เราเห็น จะมีดังนี้ครับ

Message : ช่วยให้เราดู erroes ต่างๆ ได้

Timeline : เอาไว้ให้เราสามารถดูเวลารวมในการโหลด page แต่ละ page

Exceptions : ดูข้อผิดพลาด exceptions เมื่อเราโยน thrown ออกมาจากระบบ

Views : ใช้ดูรายละเอียดในการ render รายการต่างๆทั้งหมด ที่อยู่ใน view

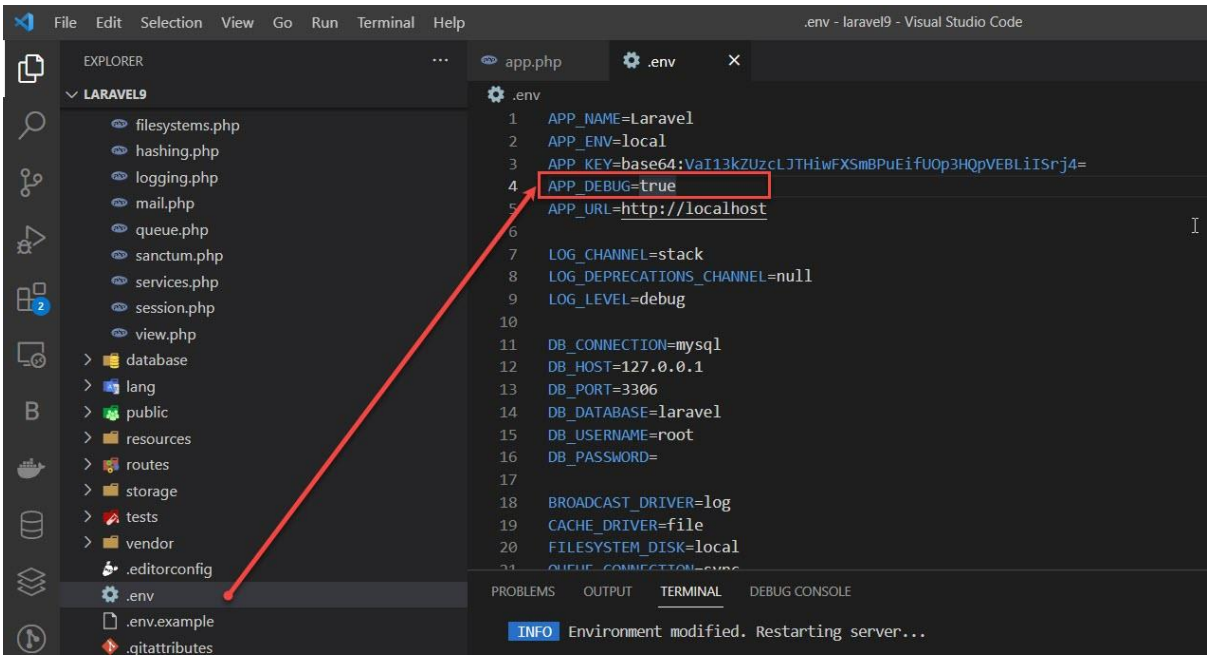
Route : ใช้ดู request ต่างๆ ที่เกี่ยวกับเส้นทางการทำงานของ Route

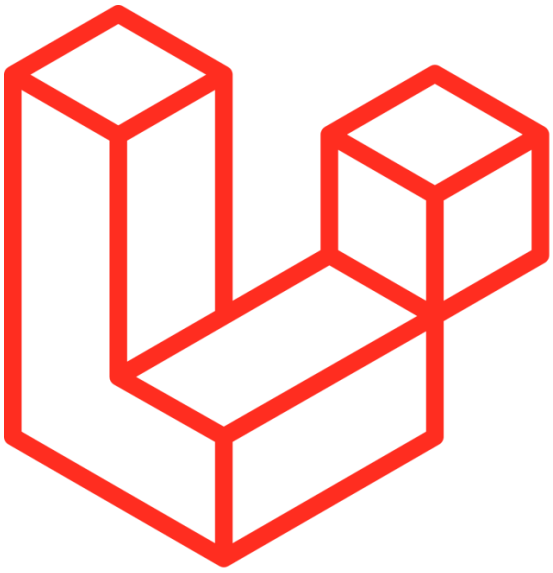
Queries : ใช้ดู SQL Queries ที่กำลังทำงานอยู่

Mails : ดูรายละเอียดเกี่ยวกับการรับส่งอีเมล

Request : ดู Status Code , request headers

อย่าลืมตั้งค่า APP_DEBUG=true ในไฟล์ .env ด้วยนะครับ

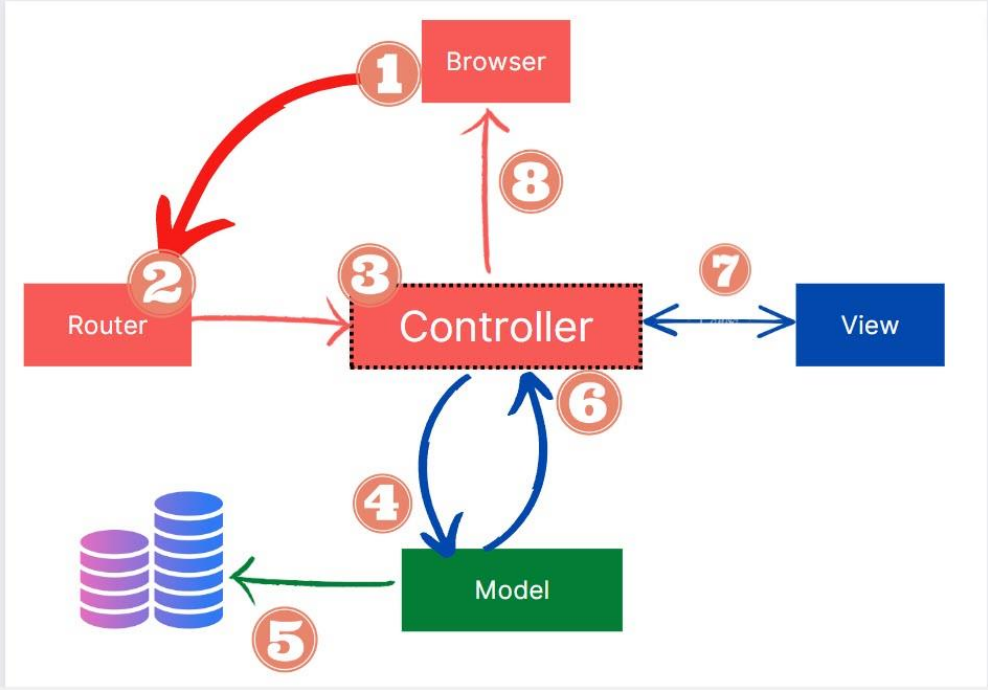




พื้นฐานการเขียน Routes ในรูปแบบต่างๆ

Routing ใน Laravel เป็นส่วนที่สำคัญมาก เพราะว่าจะเป็นตัวจัดการเส้นทางว่าในระบบงานของเราจะให้สามารถเรียกใช้งาน Controllers ไหนบ้าง จะเรียกว่า URL หรือหน้าเว็บแต่ละหน้าของเว็บเรานั่นเอง ทุก ๆ Request จะต้องควบคุมผ่าน Routing เนื่องจากว่า Route ใน Laravel ยังไม่มีคำสั่งในการสร้างอัตโนมัติ ฉะนั้นจึงต้องเขียนเองทั้งหมด และกำหนดทุกอย่างเอง อาจจะดูยุ่งยากบ้างในช่วงแรก แต่ถ้าเขียนบ่อย ๆ ทุกอย่างจะง่ายและสะดวก

Request Lifecycle



1. เมื่อมีการส่ง request ต่างๆ เช่นเราทำการพิมพ์ URL ผ่าน Browser
2. Router จะทำการตรวจสอบ request ที่ถูกส่งเข้ามาแล้วจะถูกส่งไปให้ Controller ทำงานตาม request ที่ถูกเรียกอาจจะอยู่ในรูปแบบของฟังก์ชัน
3. เมื่อ Controller ได้รับคำสั่งจาก Router ก็จะทำการตรวจสอบข้อมูลหรือ Request ที่ถูกส่งเข้ามา
4. หากพบว่า request ที่ส่งเข้ามีการเรียกใช้งาน Model
5. Model ก็จะถูกเรียกให้ทำงานพร้อมกับไปดึงข้อมูลจากฐานข้อมูล
6. เมื่อได้รับข้อมูลจาก Model แล้ว ส่งกลับไปให้ Controller ประมวลผล
7. หากมีการเรียกใช้ View ด้วย Controller ก็จะทำการอะไรบางอย่างเชื่อมกับ View แล้วส่งข้อมูลที่ได้ประมวลแล้วไปให้ View แสดงผล แล้วก็ส่งกลับไปที่ Browser ซึ่งเราจะเห็นข้อมูล UI ต่างๆ ได้ผ่านทาง Browser นั่นเอง

HTTP Verb

เพื่อให้เราเข้าใจยิ่งขึ้นเกี่ยวกับ route เรามาดูก่อนว่ามีรูปแบบไหนที่ต้องใช้บ่อยและต้องสร้างให้เป็น ผมจะแนะนำส่วนที่ใช้งานบ่อย ๆ เท่านั้นนะครับ route นั้นมีหลายรูปแบบด้วยกัน ซึ่งการกำหนด Routing ใน Laravel นั้น เราจะต้องกำหนดตาม HTTP Verb ของ HTTP Request ประกอบด้วยดังนี้

Route::get($uri,$callback)

ใช้ทำงานแบบเรียกใช้ปกติหรือถ้ามีการส่งค่าจะเป็นการส่งค่าผ่าน URL อาจจะมีการส่งค่า Parameter ด้วย ผู้ใช้งานสามารถมองเห็นข้อมูลที่ส่งไปได้

Route::post($uri,$callback)

ใช้เมื่อมีการรับค่าจาก From ใช้บ่อยมาก ผู้ใช้งานจะไม่สามารถองเห็นข้อมูลที่ส่งไป

Route::put($uri,$callback)

ใช้เมื่อเราจะส่งข้อมูลไป update หรือ แก้ไขข้อมูล

Route::patch(($uri,$callback)

ใช้เมื่อเราจะส่งข้อมูลไป update หรือ แก้ไขข้อมูล คล้ายกับ put เลยครับ

Route::delete($uri,$callback)

จะใช้เมื่อเราต้องการลบข้อมูล

Route::resource($uri,$callback)

ใช้สำหรับการสร้าง CRUD หรือ RESTful API

Route::view($uri,$view)

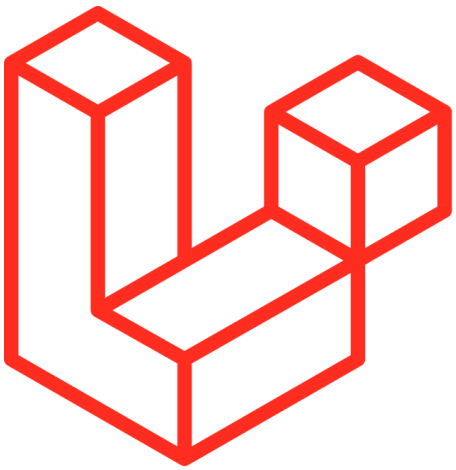
ใช้สำหรับเรียก view ตรงๆ โดยไม่ต้องผูกกับ Controller จะใช้ได้แค่บางงานเท่านั้น

Route::prefix()

ใช้สำหรับจัดกลุ่ม routing โดยอ้างอิงชื่อ prefix ทำให้เราไม่ต้องสร้างชื่อซ้ำ

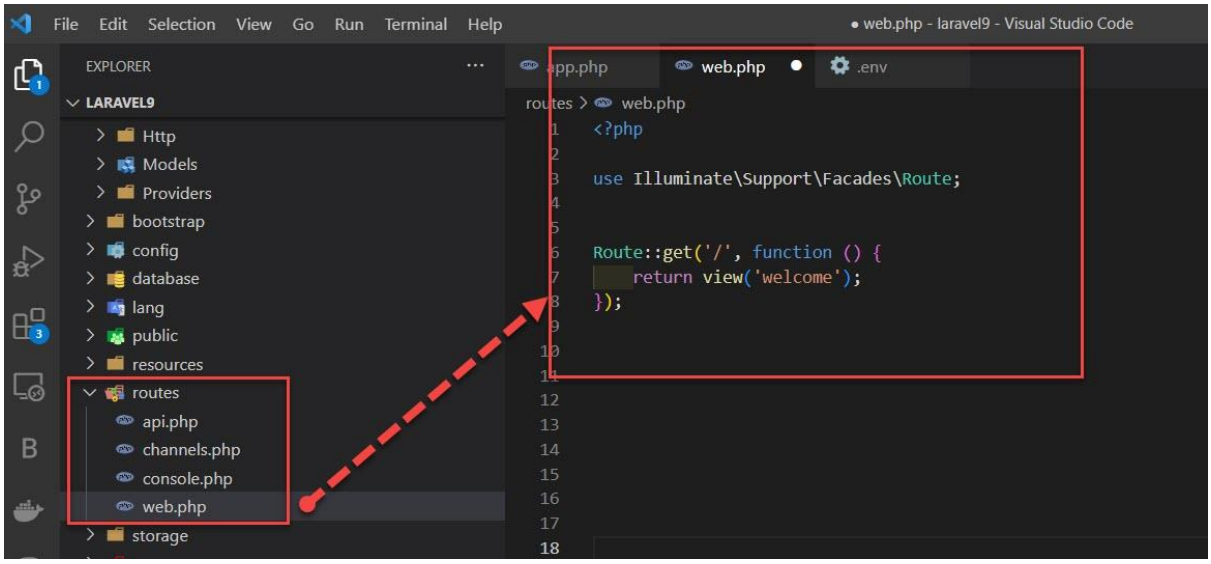
Route::middleware()

ใช้สำหรับการกรองคำขอ HTTP หรือการตรวจสอบสิทธิ์การเข้าถึงแอปพลิเคชัน



Routes (GET)

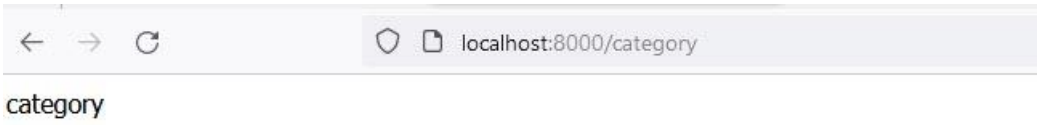
เป็นการตรวจสอบ URL Request เพื่อระบุว่าจะให้ทำงานอะไร หรือ จะทำงานที่ Controller ใด Action ใด หรือสั่งให้แสดง View ไหน สามารถระบุการส่ง Parameter ได้หลายรูปแบบ การสร้าง Route เราจะสร้างไว้ที่ไฟล์ web.php และ api.php จะอยู่ในโฟลเดอร์ routes เปิด VSCode ขึ้นมาแล้วเปิดไฟล์ routes/web.php ตามภาพครับ



จะเห็นว่ามีการสร้าง route ที่เป็น default ไว้แล้ว 1 url คือ ' / ' คือ index.php นั่นเอง และมีการคืนค่าไปที่หน้า view ชื่อว่า welcome (เก็บไว้ที่โฟลเดอร์ resources/views/welcome.blade.php) ยังไม่ผูกกับ Controller นะครับ (หากต้องการสร้าง route เพิ่มก็สามารถทำได้ ด้วยการพิมพ์ชื่อ route แล้วกำหนดชื่อ URL ตามที่เราต้องการได้ เช่น หากต้องการจะตั้งชื่อ URL ก็ให้สร้างแบบนี้ครับ

```
//Inline function

Route::get('/category',function(){

        return "category";
});

Route::get('/posts',function(){
        return "posts";
});

Route::get('/news-detail',function(){
        return "news-detail";
});
```

แล้วทดสอบพิมพ์ http://localhost:8000/category , http://localhost:8000/posts , http://localhost:8000/news-detail' ดูผลลัพธ์ครับ



category , post , news-detail ก็คือชื่อ URL ของเรา สามารถสร้างได้ตามต้องการเลยครับ ในตัวอย่างเป็นการสร้าง URL ชื่อต่างๆ แล้วคืนค่าข้อความออกมา อันนี้เข้าใจนะครับ

Routing with Parameters

Ex1.กำหนด parameters แบบบังคับต้องใส่ค่า

```
<?php
use Illuminate\Support\Facades\Route;
Route::get('/user/{id}',function($id){
    return "This is Param  ".$id ;
}) ;
```

สร้าง URL ชื่อว่า user และส่ง id ไปด้วย ใน function เราก็ทำการรับค่ามาด้วย อย่าลืม !!! ระบุตัวแปรที่ต้องการจะรับค่าในฟังก์ชัน จากนั้นก็คืนค่า id กลับไป ทำการป้อน URL ผ่าน Browser พิมพ์ localhost:8000/user/ แล้วก็ลองระบุค่าที่ต้องการส่ง

ผลลัพธ์ที่ได้



หมายเหตุ การที่เราจะส่งค่า Parameters ไปกับ Route นั้นเราจะต้องระบุเครื่องหมาย {...} ใน Route ด้วยทุกครั้ง ถ้าไม่ระบุมันก็จะมองเห็นเป็นชื่อ URL ธรรมดานั่นเอง และห้ามใช้ – แต่ให้ใช้ _ แทน

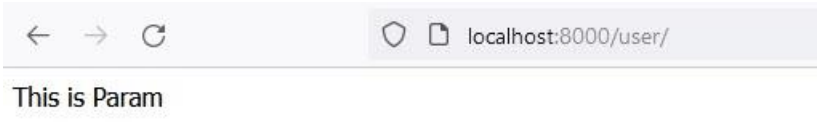
Ex2.กำหนด parameters แบบไม่บังคับต้องใส่ค่า (Optional Route)

แบบต่อมา กรณีที่เราไม่อยากบังคับว่าจะให้ส่งค่าไปด้วย หมายความว่าเราจะใส่ค่าก็ได้หรือไม่ใส่ก็ได้ ก็สามารถทำได้ ด้วยการใส่เครื่องหมาย ? ต่อท้ายตัวแปร หรือจะกำหนดค่าเริ่มต้นให้กับ parameter ก็ได้เช่นกัน

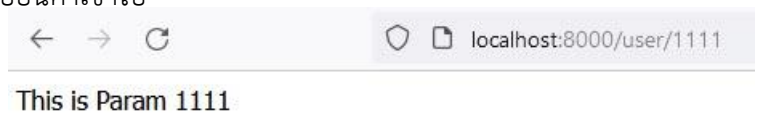
```
Route::get('/user/{id?}',function($id = null){
    return "This is Param  ".$id ;
}) ;
//กำหนดค่าเริ่มต้น
Route::get('profile/{name?}', function ($name = 'Satoshi') {
    return $name;
});
```

แล้วทดสอบดูอีกรอบ ทีนี้ผมจะลบค่า parameter ออกโดยไม่ต้องใส่ค่าอะไร

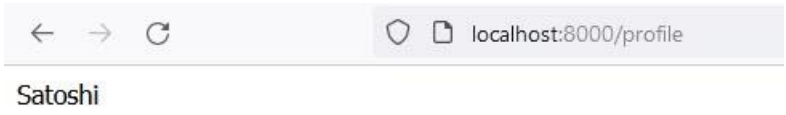
ผลลัพธ์ที่ได้ พิมพ์ localhost:8000/user แบบไม่ต้องใส่ค่า



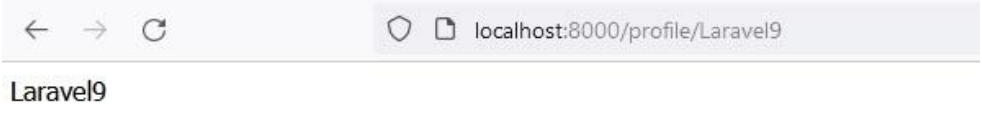
▶ ป้อนค่าเข้าไป



พิมพ์ localhost:8000/profile แบบไม่ต้องใส่ค่า



▶ ป้อนค่าเข้าไป

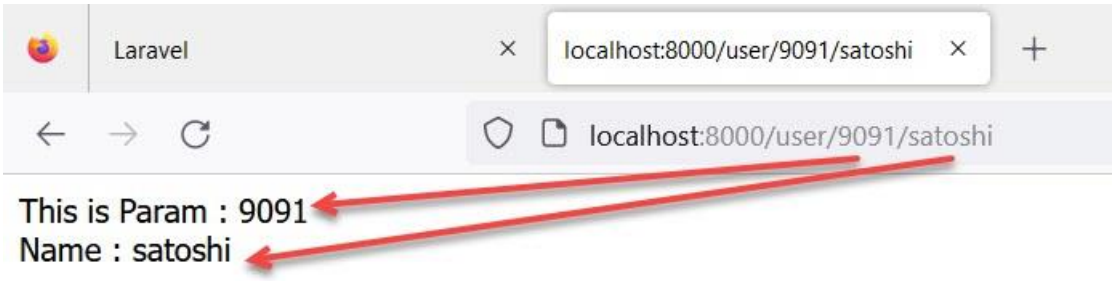


Ex3.กำหนด parameters หลาย ๆ ค่า

เราสามารถส่งค่าได้หลาย ๆ ค่า ก็ได้เช่นกัน อยากส่งค่าตัวแปรกี่ตัวก็กำหนดตามตัวอย่างด้านล่างครับ

```
Route::get('/user/{id}/{name}',function($id,$name){
    return "This is Param  : ".$id ."<br/>Name : ".$name;
});
```

พิมพ์ localhost:8000/user/9091/satoshi แล้วใส่ค่าเข้าไป

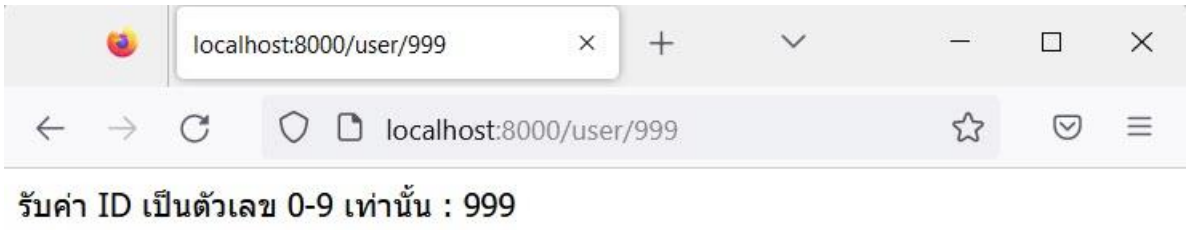


Regular Expression Route

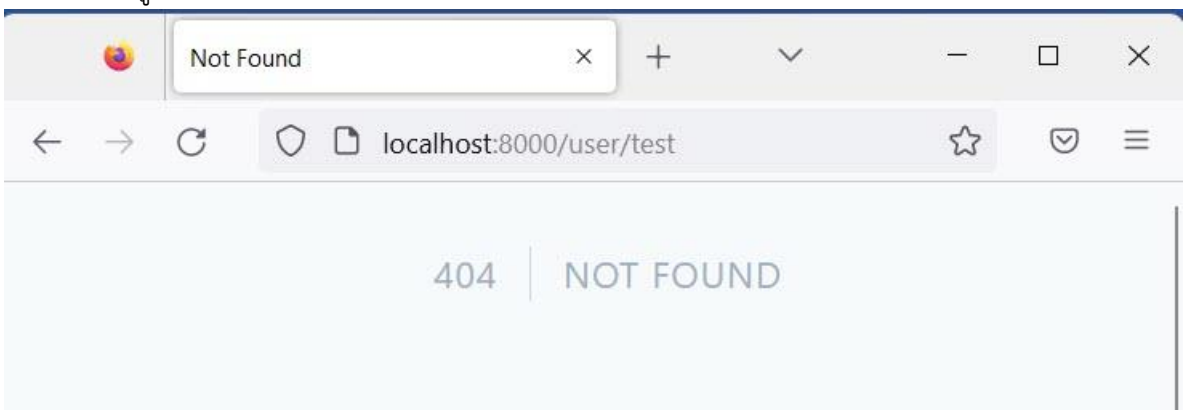
การกำหนดเงื่อนไขข้อมูลเพื่อควบคุมการทำงานตามที่ต้องการ จะใช้เมื่อเราต้องการกรองข้อมูลตามที่เราต้องการเท่านั้น เช่น จะต้องป้อนค่าแค่ตัวเลข หรือ ให้ป้อนค่าเป็นตัวอักษร เท่านั้น ตัวอย่างการเขียน ให้เราใส่ where() ต่อท้าย ดังนี้

```
Route::get('/user/{id}',function($id){
    return "รับค่า ID เป็นตัวเลข 0-9 เท่านั้น :  ".$id ;
})->where('id','[0-9]+') ;
```

ทดสอบการทำงาน พิมพ์ localhost:8000/user/999



ลองป้อนข้อมูลที่เป็นข้อความหรือตัวอักษรเข้าไป แล้วทดสอบการทำงาน พิมพ์ localhost:8000/user/test

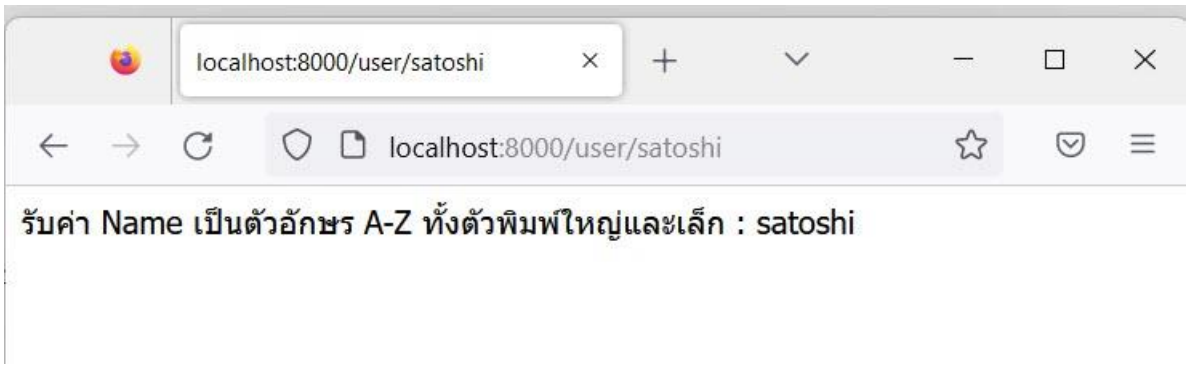


เมื่อป้อนค่าอื่น ๆ ที่ไม่เป็นไปตามเงื่อนไขก็จะขึ้นหน้า error แสดงข้อผิดพลาด 404 NOT FOUND

รับค่าตัวแปรที่เป็นตัวอักษรเท่านั้น ตัวอย่างการเขียน

```
Route::get('/user/{name}',function($name){
    return "รับค่า Name เป็นตัวอักษร A-Z ทั้งตัวพิมพ์ใหญ่และเล็ก :  ".$name ;
})->where('name','[A-Za-z]+') ;
```

ทดสอบการทำงาน พิมพ์ localhost:8000/user/satoshi



จะกำหนดรูปแบบข้อมูลทั้งสองแบบก็สามารถทำได้เช่นกัน

```
Route::get('/user/{id}/{name}',function($id, $name){
    return "This Id :  ".$id ."<br/>This Name :".$name;
})->where(['id'=>'[0-9]+','name'=>'[A-Za-z]+']) ;
```

Named Routes With Parameters

การกำหนดชื่อให้กับ route ช่วยให้เราสามารถเรียกใช้งานได้สะดวก ตัวอย่างการเขียน ให้เราใส่ name() ต่อท้าย ดังนี้

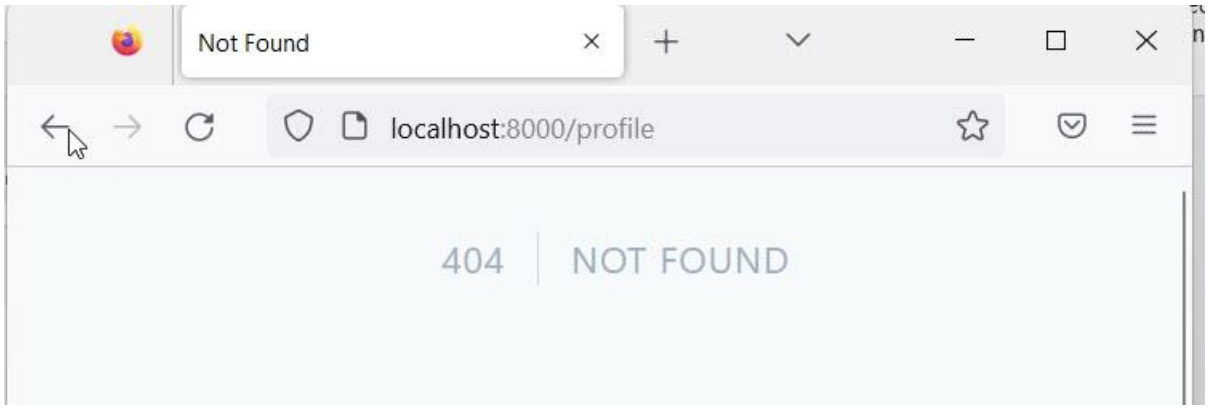
web.php

routes > web.php

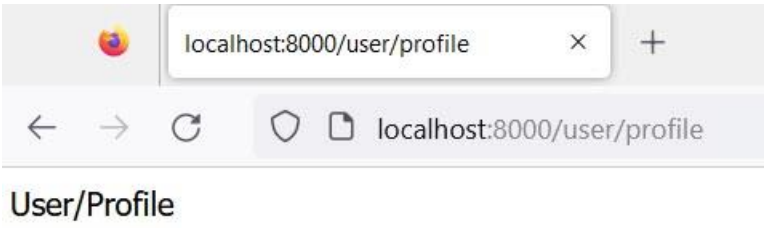
```php
<?php

use Illuminate\Support\Facades\Route;

Route::get('/user/profile', function () {
    return "User/Profile";
})->name('profile');

```

การใช้งาน เราจะป้อน URL ตรง ๆ ผ่าน Browser ไม่ได้นะ ไม่เชื่อลองรันดูครับ

พิมพ์ localhost:8000/profile ที่ Browser



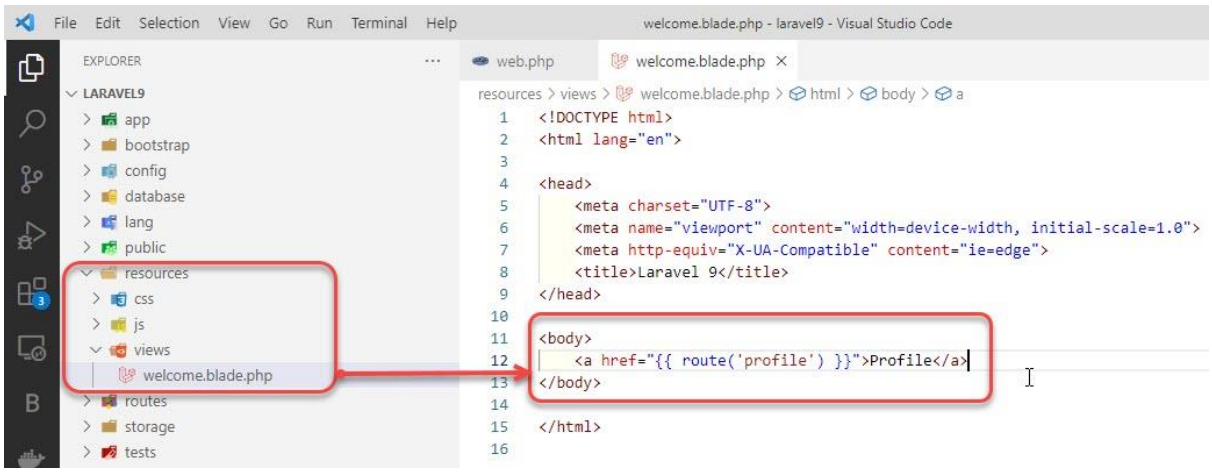
แต่ถ้าเราพิมพ์ URL แบบนี้ localhost:8000/user/profile อ้าวได้ซะงั้น



ที่เป็นแบบนี้ก็เพราะว่าเมื่อใดก็ตามที่เรากำหนดชื่อให้ route เราจะต้องใช้ร่วมกับ helper function นั่นเองครับ เช่น route() , redirect() หมายความว่าจะต้องมีอะไรบางมาสั่งให้ทำงาน ถ้านึกภาพไม่ออก ก็ลองมาดูตัวอย่างการใช้งานกันดูครับ

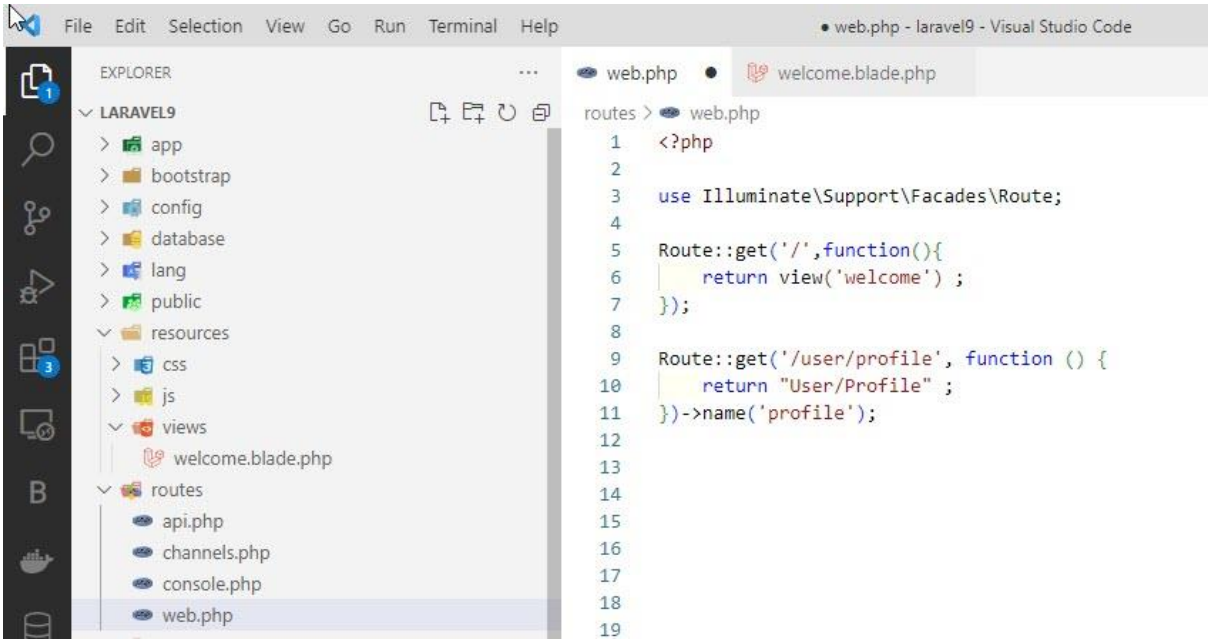
เปิดโปรแกรม VSCode ขึ้นมาก่อนครับแล้วไปเขียนโค้ด html ที่ views ไฟล์ welcome.blade.php ดังนี้

resources/views/welcome.blade.php

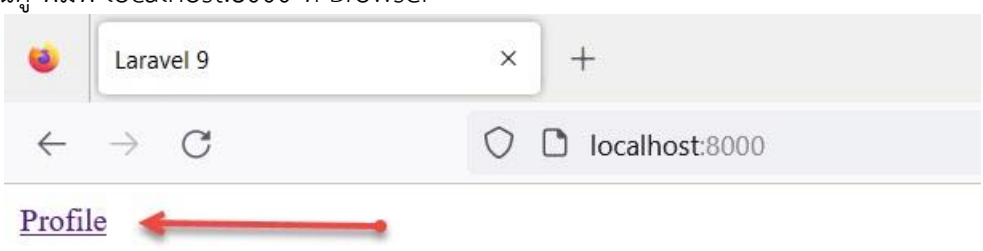


ไฟล์ web.php

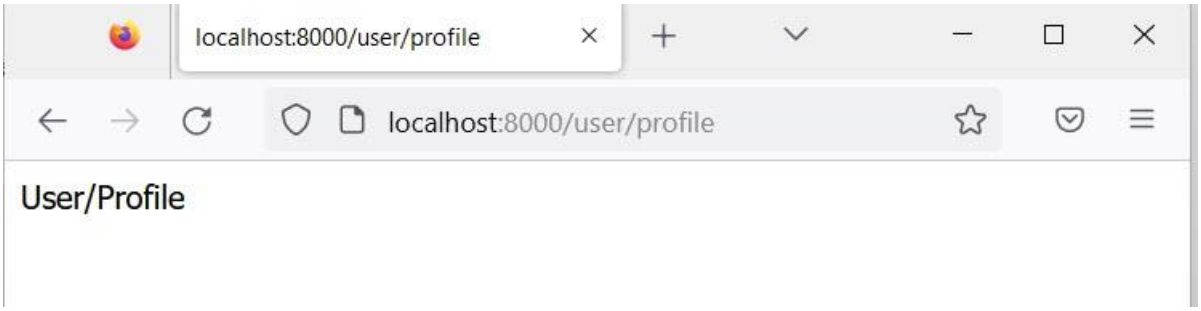
routes/web.php



ทดสอบรันดู พิมพ์ localhost:8000 ที่ Browser



จะมีลิงค์ข้อความคำว่า Profile ให้เรากด และเมื่อเรากดที่ลิงค์ มันก็จะ redirect ไปที่หน้า user/profile ตามภาพครับ



การส่งค่าตัวแปรไปกับ Route ให้แก้ไขโค้ดที่ไฟล์ web.php ตามนี้

web.php | welcome.blade.php

routes > web.php

```php
<?php

use Illuminate\Support\Facades\Route;

Route::get('/',function(){
    return view('welcome') ;
});

Route::get('/user/profile/{id}/{name}', function ($id = null , $name = null) {
    return "User ID : ".$id ."<br/>User Name :".$name ;
})->name('profile');

```

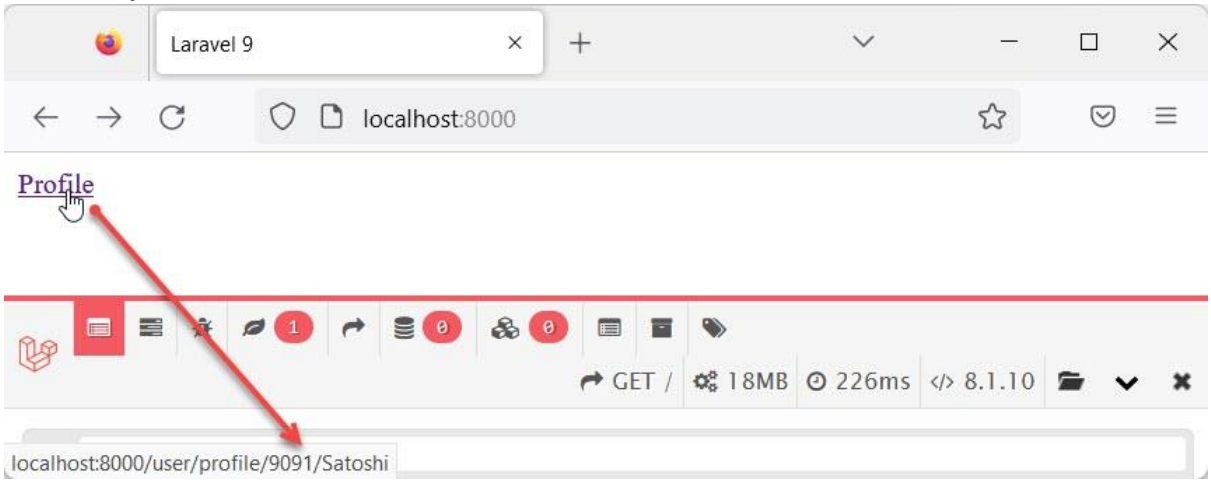
เพิ่มโค้ดที่ไฟล์ welcome.blade.php ตามนี้

web.php | welcome.blade.php

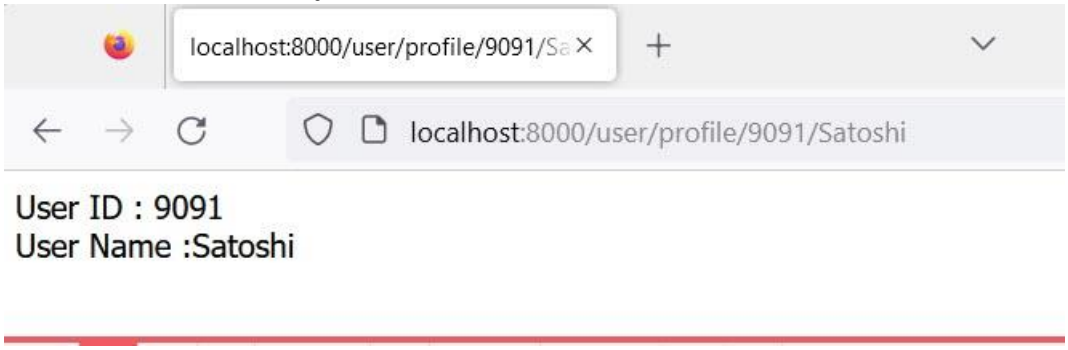
resources > views > welcome.blade.php > html

```php
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Laravel 9</title>
</head>

<body>
    <a href="{{ route('profile', ['id' => 9091, 'name' => 'Satoshi']) }}">Profile</a>
</body>

</html>
```

ทดสอบรันดูผลลัพธ์ พิมพ์ localhost:8000 ที่ Browser



จะเห็นว่าเมื่อเรานำเมาส์มาชี้ที่ลิงค์ Profile ก็จะมีค่าตัวแปรที่เราส่งมา และเมื่อคลิกก็จะแสดงหน้า user/profile พร้อมแสดงค่าตัวแปรที่ถูกส่งมา ตามภาพ

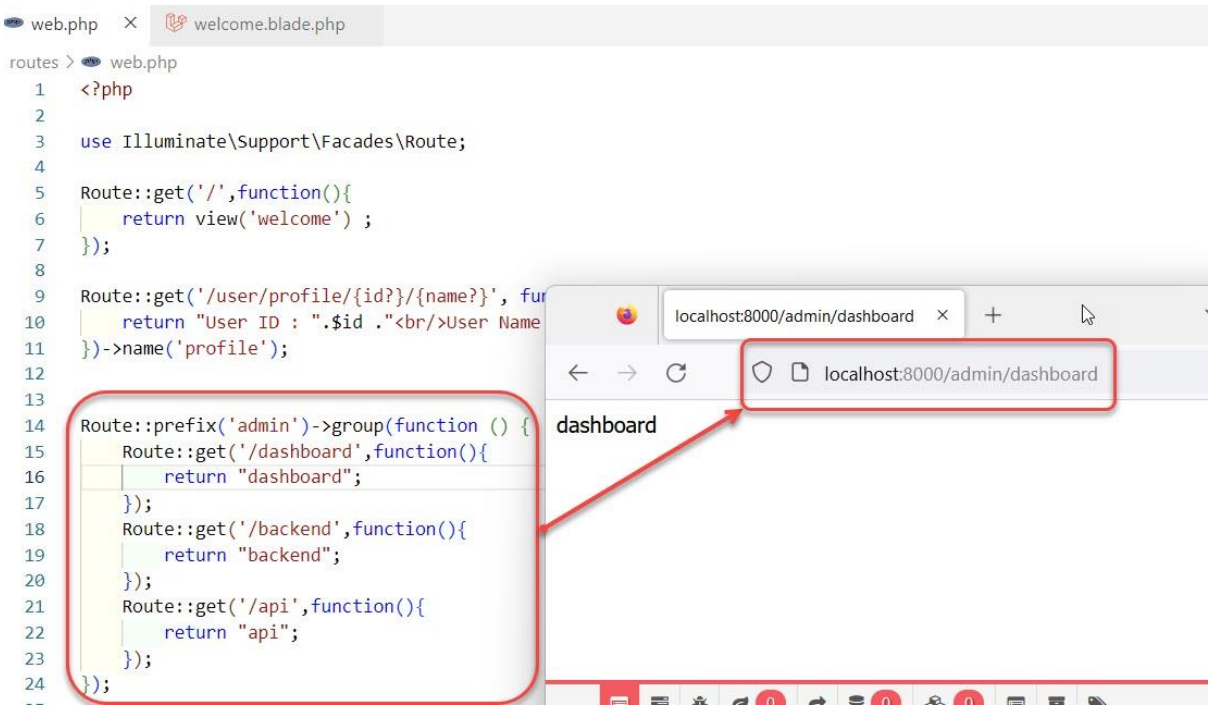


Routes Groups prefix

เป็นการจัดกลุ่มให้กับ routes นั้นจะใช้คลาส prefix() ในการตั้งชื่อกลุ่ม และ group() ในการจัดกลุ่ม ทั้งสองคลาสจะใช้เป็นตัวจัดการ ทำให้เราไม่ต้องสร้างชื่อซ้ำๆ และทำให้ routes เราดูเป็นระเบียบมากขึ้นในกรณีที่อยากให้ routes อยู่ในกลุ่มเดียวกัน ตัวอย่างการเขียน

```
Route::prefix('admin')->group(function () {

        Route::get('/dashborad',function(){
        });

        Route::get('/backend',function(){
        });

        Route::get('/api',function(){
        });

});
```

admin/dashboard , admin/backend , admin/api ตัวอย่างตามภาพ



จะเห็นว่า prefix คือ admin ซึ่งภายในจะเก็บ routes ต่าง ๆ ไว้สามารถสร้างได้แบบไม่จำกัดเลยครับ อยากได้กี่ routes ก็ยัดเข้าไปไว้ในนี้ได้เลย

Routes (POST)

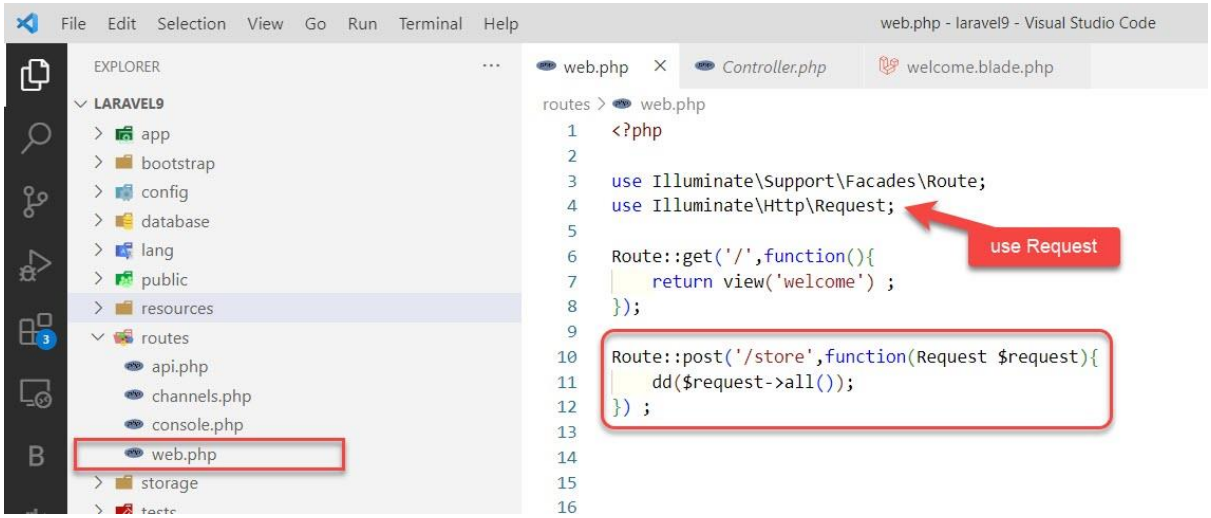
ทีนี้มาดูในส่วนของ routes ที่เป็นแบบ post , put , patch และ delete กันต่อเลยครับ ลักษณะการเขียนก็จะเหมือนกันกับ get เลยครับ การป้อนพารามิเตอร์ หรือการตั้งชื่อก็จะใช้เหมือนกันหมดทุกอย่าง จะต่างกันก็แค่ method ที่จะใช้โดยจะเปลี่ยนจาก Route::get() เป็น Route::post() เท่านั้นเอง

Route::post() ส่วนมากแล้วจะใช้เป็นตัวจัดการเกี่ยวกับการรับข้อมูลที่ส่งมาจากฟอร์มที่ผู้ใช้งานกรอกข้อมูลแล้วมีการกดปุ่ม submit เข้ามาเป็นส่วนใหญ่ ไปดูวิธีการเขียนเลยครับ

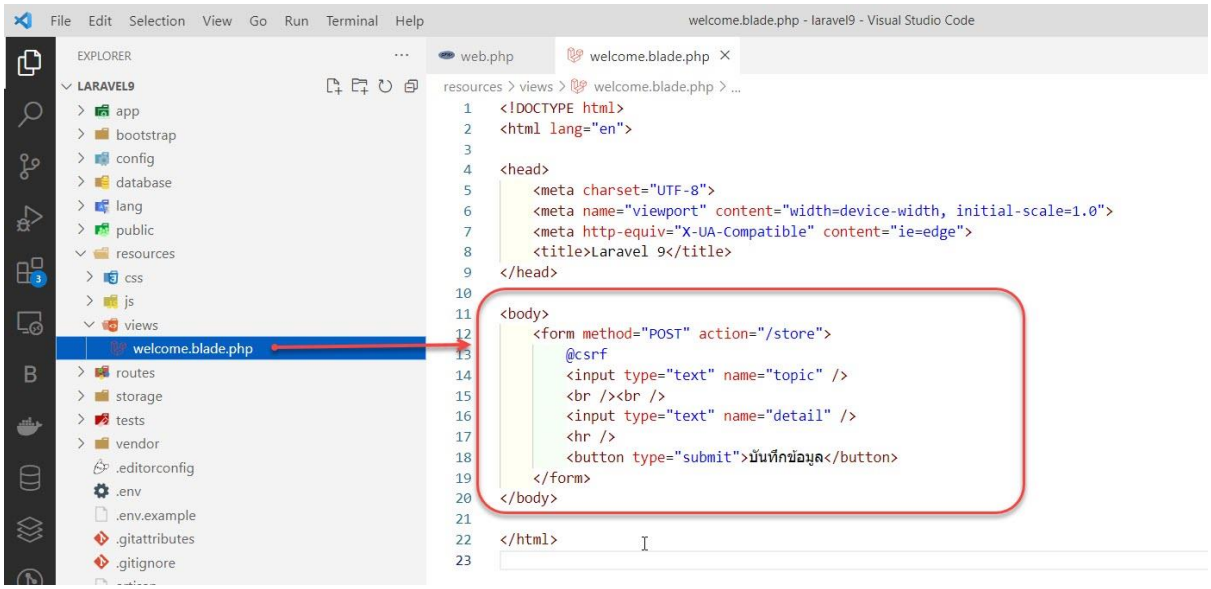
ข้อแตกต่างระหว่าง GET และ POST

GET ส่งค่าผ่าน URL ผู้ใช้จะมองเห็นข้อมูลที่ส่งไป
POST ข้อมูลที่ถูกส่งไปจะไม่สามารถมองเห็นข้อมูลนั้นได้

เปิดไฟล์ routes/web.php แล้วเขียนโค้ดลงไปตามนี้



เพิ่มโค้ดที่ไฟล์ welcome.blade.php

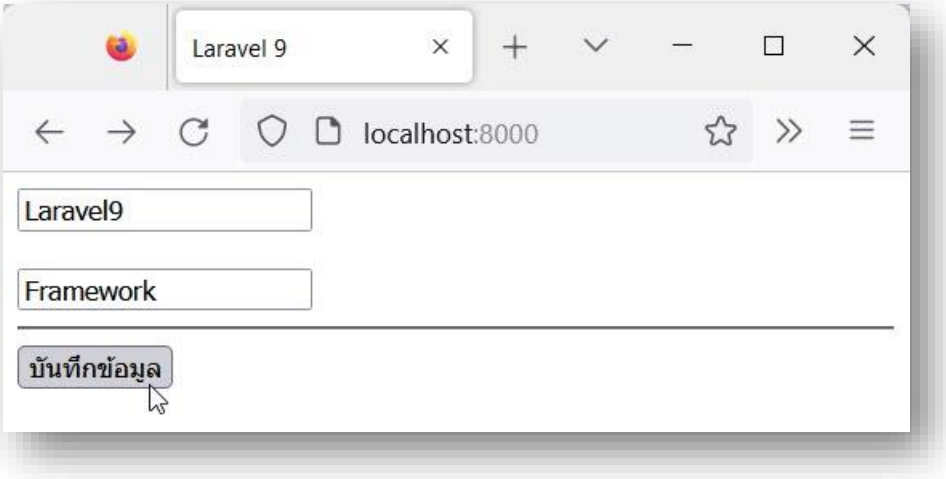


อธิบายโค้ด

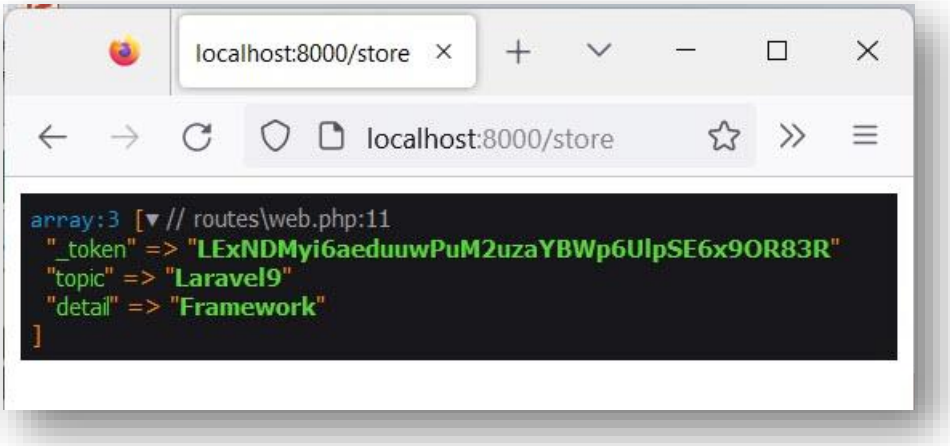
ที่ไฟล์ web.php เป็นการสร้าง routes โดยใช้ method post สำหรับการรับข้อมูลจากฟอร์ม ตั้งชื่อ routes ว่า store และมีการกำหนด Parameter 2 ตัวคือ Request และ $request เพื่อเป็นตัวแปรสำหรับรับค่าจากฟอร์ม จากนั้นก็ทำการดีบั๊ก (Debug) ดูข้อมูลที่รับมา ทุกครั้งที่มีการใช้ Request ให้ทำการ Use Request ด้วยทุกครั้ง (บรรทัดที่ 4 ในไฟล์ web.php) ไม่งั้นจะเกิดข้อผิดพลาดขึ้นทันทีเวลาที่กดบันทึกข้อมูล

ที่ไฟล์ welcome.blade.php เป็นการเขียนโค้ดสำหรับให้ผู้ใช้กรอกข้อมูล โดยกำหนด method ให้กับ form เป็น post และกำหนด action ที่จะส่งไปคือ store และเพิ่ม helper @csrf ภายใต้ form

ทดสอบรันดู พิมพ์ localhost:8000 ที่ Browser แล้วกรอกข้อมูลลงในฟอร์ม แล้วกดปุ่ม บันทึกข้อมูล



ผลลัพธ์ที่ได้



CSRF Protection (@csrf)

โปรดจดจำไว้ว่าทุกครั้งที่มีการสร้างฟอร์มให้ผู้ใช้กรอกข้อมูล ท่านจะต้องกำหนด @csrf ภายใต้ฟอร์มนั้น ๆ ไม่งั้นฟอร์มที่ท่านสร้างจะใช้ไม่ได้ จะเกิดข้อผิดพลาดเกิดขึ้นทันที

CSRF (Cross-site Request Forgery) ใน Laravel ใช้สำหรับป้องกันการโจมตีเว็บไซต์ที่ Hacker นิยมใช้โจมตีเว็บไซต์ของเรา เพราะฉะนั้นแล้ว อย่าลืมเพิ่มเข้าไปทุกครั้งนะครับ 😊 จะเขียนแบบนี้ก็ได้นะ

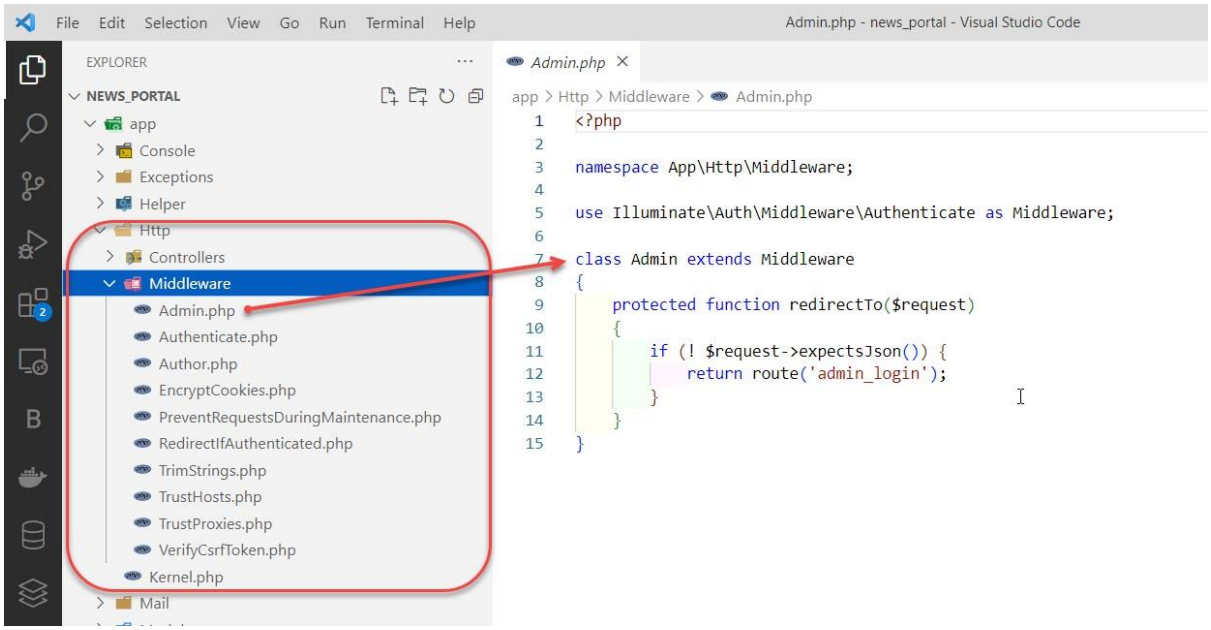
```
<input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">
```

หรือ @csrf ก็ใช้ได้เหมือนกัน .. แต่แนะนำ ใช้ @csrf ดีกว่า มันเขียนสั้น ...

กรอง HTTP request ด้วย Middleware

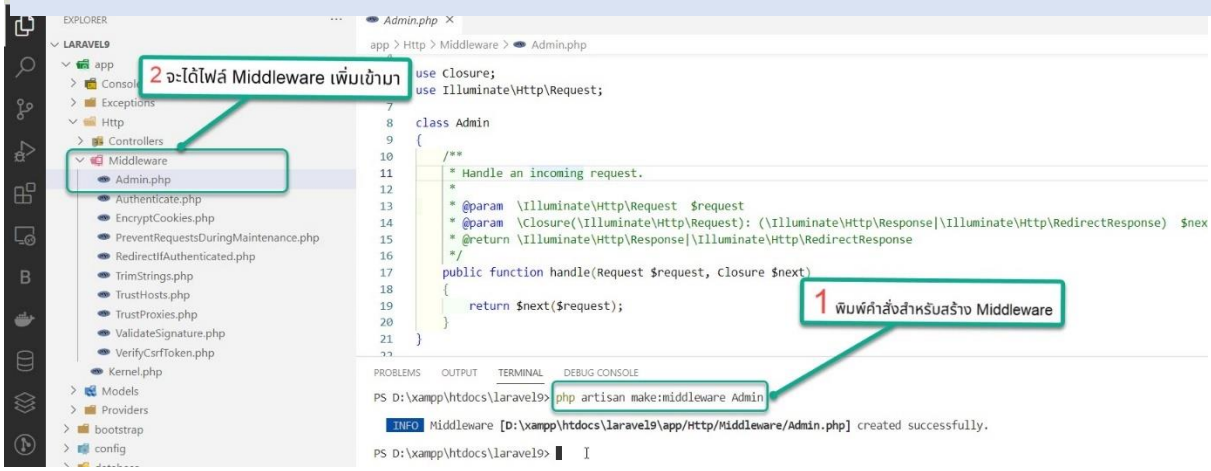
Middleware ใน Laravel เป็นส่วนที่ใช้สำหรับเขียน Logic หรือคัดกรอง request เพื่อตรวจสอบเงื่อนไขต่าง ๆ ว่าในเว็บไซต์ของเราส่วนไหนบ้างที่จะให้ผู้ใช้งานเข้าถึงได้ สามารถมี Middleware ได้หลายอัน ซึ่งแต่ละอันนั้นก็ทำหน้าที่แตกต่างกัน เช่น หน้าเว็บส่วนไหนบ้างที่จะอนุญาตให้เข้าใช้งานได้ โดยไม่ต้องทำการล็อกอินเข้าระบบก่อน ส่วนไหนที่เป็นหน้าสำหรับผู้ดูแลระบบ ผู้ใช้อื่นที่ไม่ใช่สิทธิ์ผู้ดูแลระบบก็จะไม่สามารถเข้าไปใช้งานได้

Middleware เมื่อทำการสร้างจะเขียนไว้ที่โฟลเดอร์ app/Http/Middleware



การสร้าง Middleware นั้นสามารถสร้างได้ด้วยการใช้คำสั่งผ่าน Terminal ดังนี้

```
php artisan make:middleware <ระบุชื่อ Middleware>
```



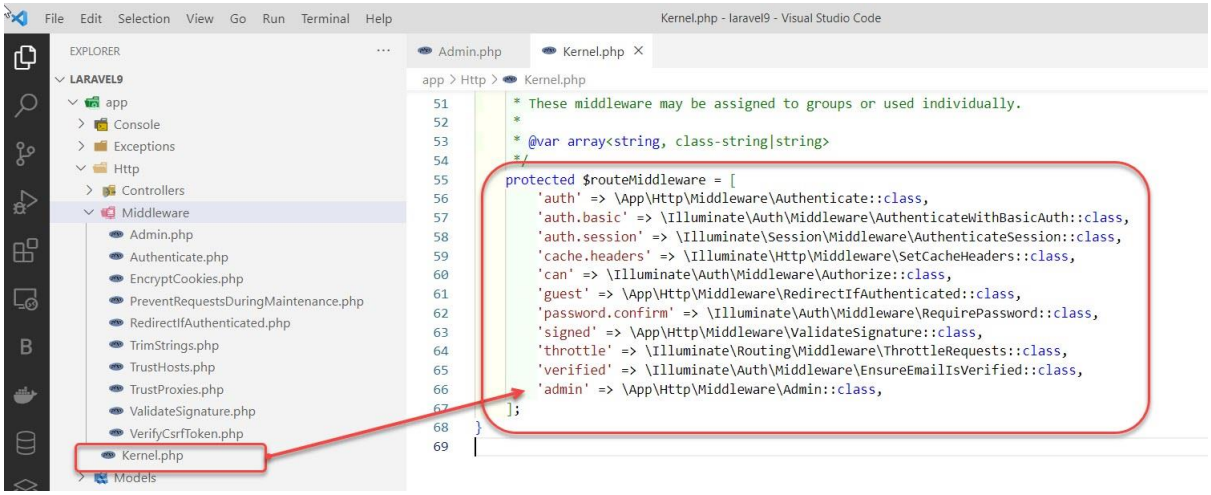
การทำงานของ Middleware นั้นจะทำงานที่ฟังก์ชัน ชื่อว่า handle() หรือ redirectTo() โดยมีการรับ $request ต่าง ๆ เข้ามา อาจมีการเขียนเงื่อนไขต่างๆ ตามที่ต้องการเพิ่มเข้าไป ตัวอย่างการเขียน Middleware พื้นฐานง่าย ๆ ในการตรวจสอบผู้ใช้งานว่ามีการล็อกอินหรือไม่โดยการใช้ฟังก์ชัน redirectTo()

```
<?php
namespace App\Http\Middleware;
use Illuminate\Auth\Middleware\Authenticate as Middleware;
class Authenticate extends Middleware {
    protected function redirectTo($request) {
        if (! $request->expectsJson()) {
            return route('login');    //ถ้าไม่ได้ Login ก่อน ก็ให้ไปที่หน้า Login
        }
    }
}
```

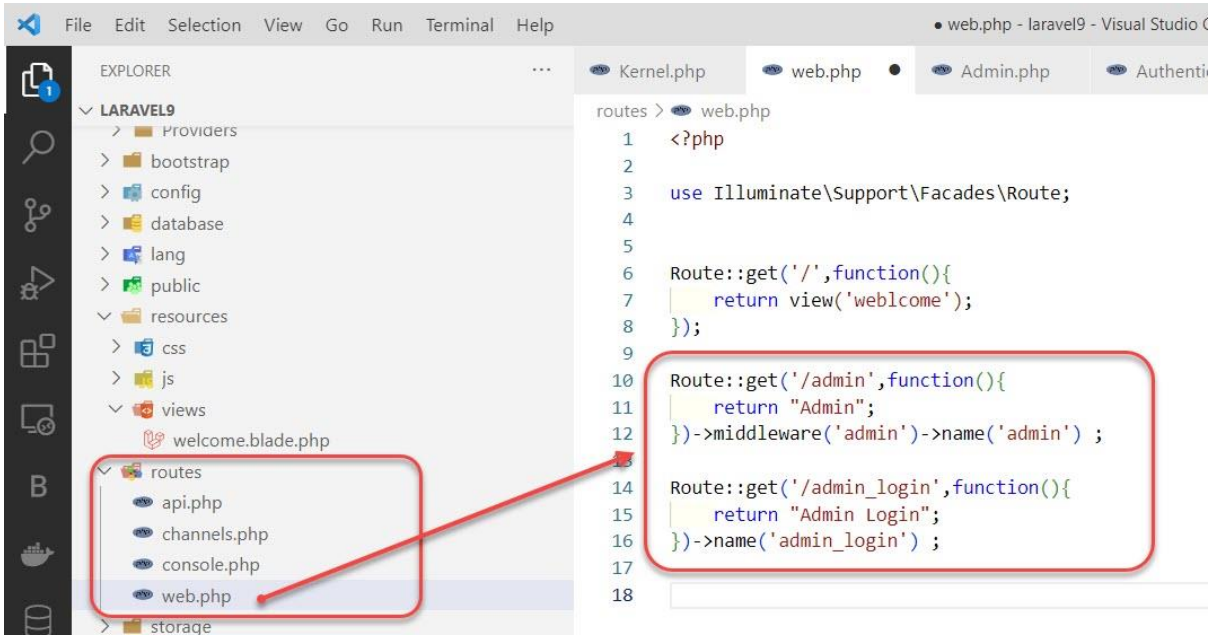
ตัวอย่างการเขียน Middleware ในการตรวจสอบผู้ใช้งานว่าเป็น Admin หรือไม่ โดยการใช้ฟังก์ชัน redirectTo()

```
<?php
namespace App\Http\Middleware;
use Illuminate\Auth\Middleware\Authenticate as Middleware;
class Admin extends Middleware {
    protected function redirectTo($request) {
        if (! $request->expectsJson()) {
            return route('admin_login');
          }
       }
}
```

Middleware และ Path ของ Middleware ที่เราจะใช้งานเข้าไป ตามตัวอย่างด้านล่างนี้ครับ

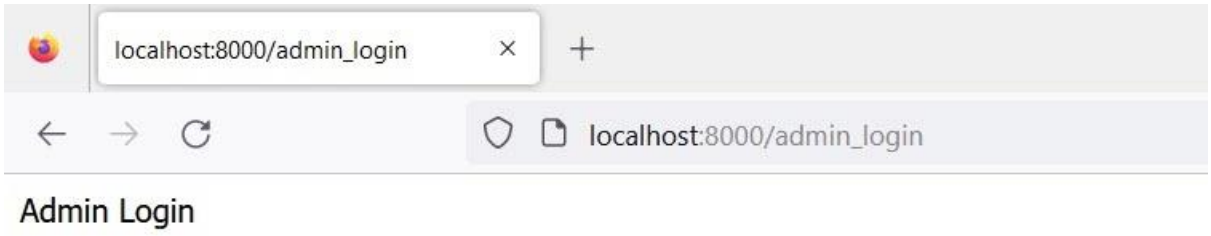


การเรียกใช้งาน Middleware สามารถกำหนดได้ที่ ไฟล์ routes/web.php โดยเพิ่ม middleware ต่อเข้าไปแล้วใส่ชื่อ Middleware ที่ต้องการใช้ได้เลย ตัวอย่างการเรียกใช้ ตามนี้ครับ



จากตัวอย่างด้านบน มีการสร้าง routes ชื่อว่า admin และมีการใช้ middleware admin ในการตรวจสอบผู้ใช้งานว่ามีการล็อกอินหรือไม่ ถ้าไม่ได้ล็อกอินก็จะทำการ redirect ไปที่ routes admin_login

ทดสอบรันดู พิมพ์ localhost:8000/admin ที่ Browser



เห็นไหมครับ เราพิมพ์ localhost:8000/admin แทนที่มันจะแสดงหน้า admin แต่กลับไปแสดงผลที่หน้า admin_login ที่เป็นแบบนี้ก็เพราะเราใช้ middleware ใน routes ไหนที่เราอยากให้ล็อกอินก่อนก็ให้

ใส่ middleware ในหน้า ๆ นั้น หรือถ้าเราไม่อยากใช้ middleware ในทุก ๆ routes ในการตรวจสอบการล็อกอิน ก็สามารถใช้คำสั่ง Auth::routes() ก็ได้นะ

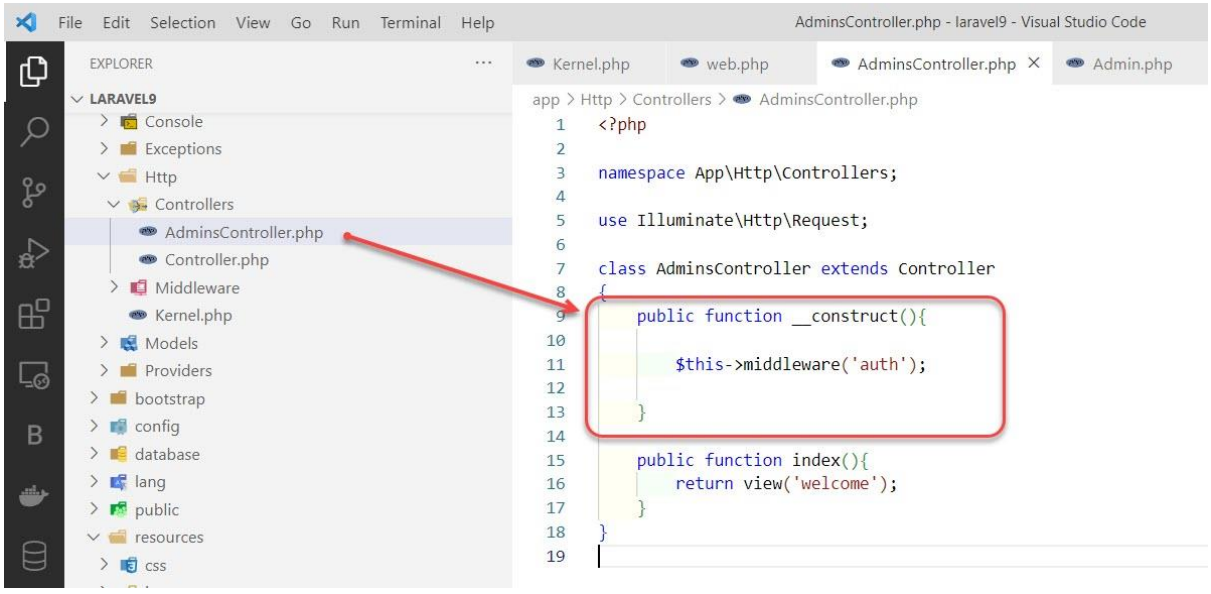
web.php

D: > xampp > htdocs > work > routes > web.php

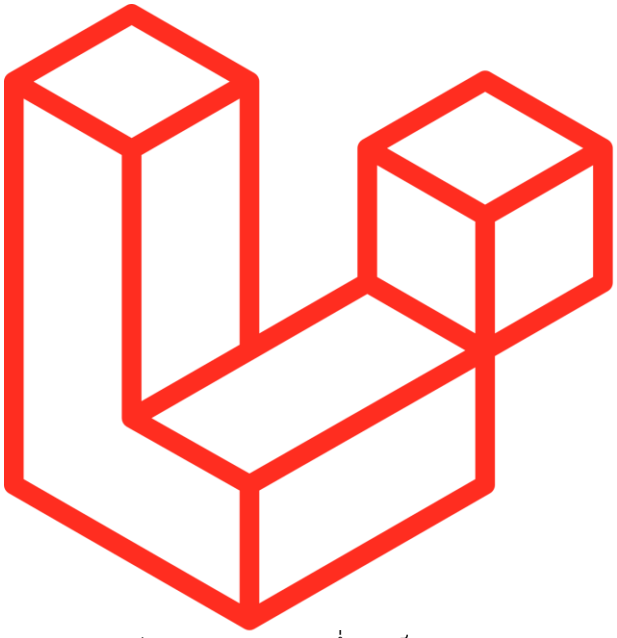
```php
Route::post('/upload/attachfile3',[FrontendController::class,'attachfile3'])->name('attachfile3');
Route::post('/upload/attachfile4',[FrontendController::class,'attachfile4'])->name('attachfile4');
Route::post('/upload/attachfile5',[FrontendController::class,'attachfile5'])->name('attachfile5');
Route::post('/upload/attachfile6',[FrontendController::class,'attachfile6'])->name('attachfile6');
Route::post('/upload/attachfile7',[FrontendController::class,'attachfile7'])->name('attachfile7');

//complete
Route::post('/complete',[FrontendController::class,'complete'])->name('complete');

Route::fallback(function(){
    return "<h2>Sorry, the page does not exits</h2>";
});

Auth::routes();
Route::get('/dashboard', [HomeController::class, 'index'])->name('dashboard');
Route::get('/getStaff',[HomeController::class,'getStaff'])->name('getStaff');
Route::get('/getStaffId/{staffid}',[HomeController::class,'getStaffId'])->name('getStaffId');

//รายละเอียดตำแหน่ง
Route::get('/position',[PositionController::class,'index'])->name('position');
Route::get('/position/create',[PositionController::class,'create'])->name('position.create');
Route::post('/position/store',[PositionController::class,'store'])->name('position.store');
Route::get('/position/edit/{id}',[PositionController::class,'edit'])->name('position.edit');
Route::patch('/position/update/{id}',[PositionController::class,'update'])->name('position.update');
Route::get('/staff/destroy/{id}',[PositionController::class,'destroy'])->name('position.destroy');
```

เพิ่ม Auth::routes() เข้าไปในส่วนด้านบน routes ที่ต้องการให้ตรวจสอบการล็อกอิน routes ที่ถูกเขียนอยู่ใต้คำสั่ง Auth::routes() จะถือว่าเป็นหน้าที่ต้องล็อกอินก่อนนั่นเอง ก็จะช่วยให้เราลดระยะเวลาในการกำหนด middleware ทุกๆ หน้าด้วยเช่นกัน

นอกจากการใช้ Middleware ผ่าน routes แล้วยังสามารถใช้งาน Middleware ใน Controller ได้อีกด้วย โดยการสร้าง Constructor ขึ้นมา ตัวอย่างการเขียน



__constructor คือ ฟังก์ชันที่ถูกเรียกให้ทำงานอัตโนมัติ มีไว้สำหรับกำหนดค่าเริ่มต้น ซึ่ง constructor ก็จะเป็นตัวช่วยบังคับให้ต้องระบุค่าเริ่มต้นตอนสร้าง object ซึ่งจะถูกเรียกใช้งานก่อนทุกครั้งก่อนที่จะไปทำงานส่วนอื่น ๆ

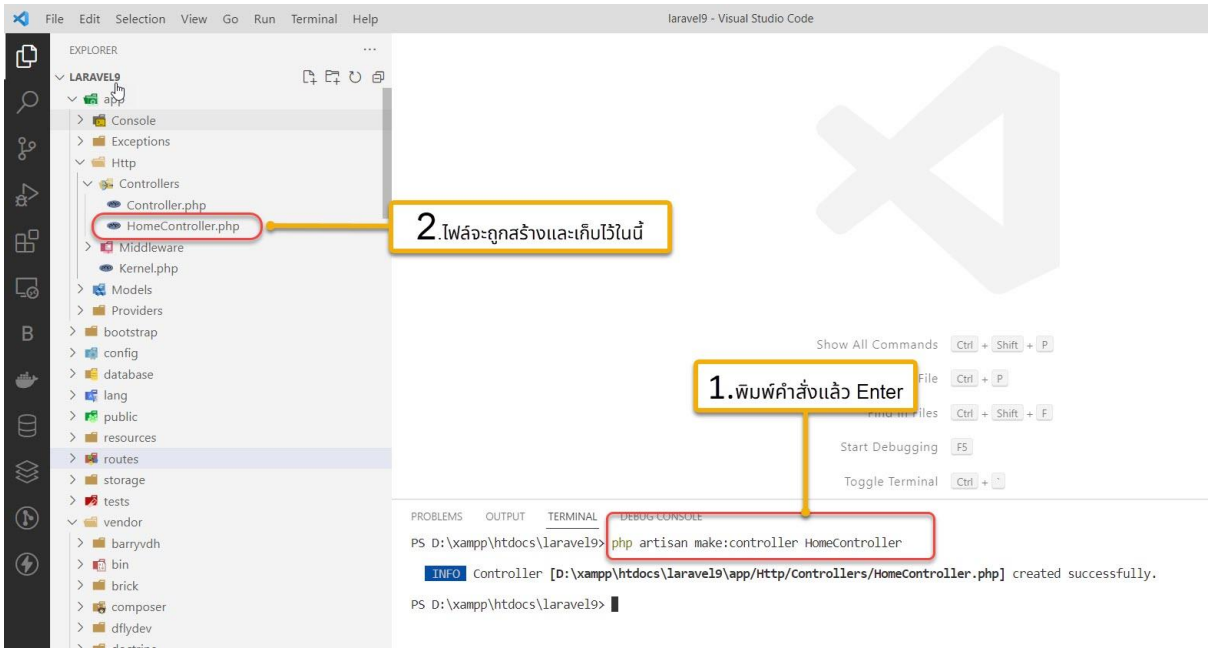


พื้นฐานการเขียน Controllers

คอนโทรลเลอร์ (Controllers) เป็นส่วนของการจัดการ การทำงานต่าง ๆ ของระบบ ซึ่งจะเป็นตัวกลางสำหรับการติดต่อกับ Model และ View การสร้าง controller นั้นทำได้ง่าย ใน Laravel จะมีเครื่องมือในการช่วยให้เราสร้าง controllers ได้อย่างรวดเร็ว เรามาลองสร้าง controllers ขึ้นมาแล้วเรียกใช้กันเลยครับ อ้าวลุย !!!

พิมพ์คำสั่งผ่าน Terminal ดังนี้

```
php artisan make:controller <ชื่อ Controller>
```



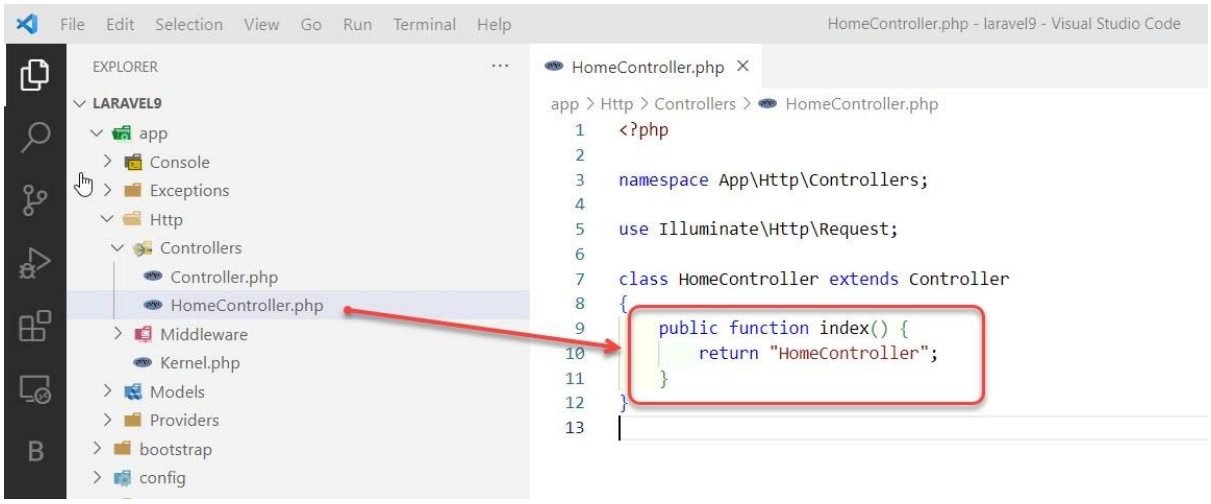
เมื่อพิมพ์คำสั่งสำหรับการสร้างคอนโทรลเลอร์แล้วให้เคาะ Enter 1 ครั้ง จะขึ้นข้อความ created successfully ไฟล์คอนโทรลเลอร์ก็จะถูกสร้างขึ้นมาให้เราและจะเก็บไว้ที่โฟลเดอร์ app/Http/Controllers โดยจากตัวอย่างในภาพ เราได้สร้างคอนโทรลเลอร์มา 1 ไฟล์ชื่อ HomeController.php

การสร้างคอนโทรลเลอร์ใหม่เราจะตั้งชื่อขึ้นต้นด้วยตัวพิมพ์ใหญ่และตามด้วยความว่า Controller เสมอ จะได้เราง่ายต่อการบำรุงรักษาโค้ดในอนาคตจะได้แยกออกว่าอันไฟล์ไหนเป็นไฟล์ controllers นั่นเอง ตำแหน่งโฟลเดอร์สำหรับเก็บไฟล์ controller ทั้งหมด

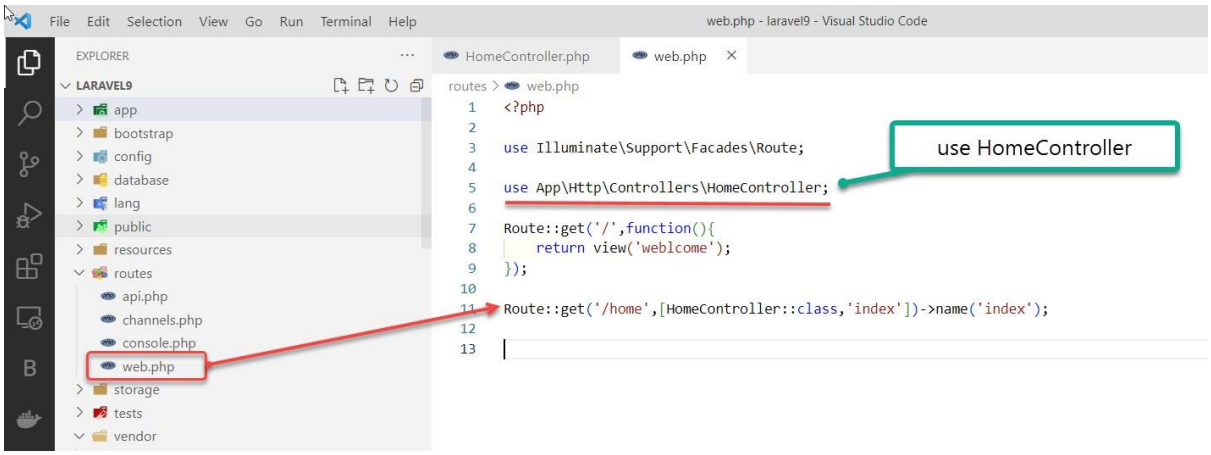
App/Http/Controllers/HomeController.php

เมื่อเราสร้าง controllers ขึ้นมาแล้ว ก็ต้องมีการเรียกใช้งาน การใช้งาน controllers ที่เราสร้างขึ้นมานั้น ปกติแล้วเราจะพิมพ์ชื่อ controllers โดยตรงไม่ได้นะ จะต้องมีตัวช่วยในการเรียกใช้งาน ตัวช่วยนั้นก็คือเราจะเรียกใช้งาน controllers ที่เราสร้าง ผ่าน routes นั่นเองครับ ปกติที่ผ่านมานั้น เราได้เขียน routes โดยไม่ได้เชื่อมต่อกับ controllers สักอันเลย ทีนี้มาลองเขียน routes เพื่อใช้งาน controllers กันต่อเลยครับ

ที่ไฟล์ App/Http/Controllers/HomeController.php ให้เพิ่มเมธอด index() เข้าไปตามนี้ครับ



จากนั้นก็เขียนโค้ดเพิ่มที่ไฟล์ web.php ครับ ดังนี้ครับ



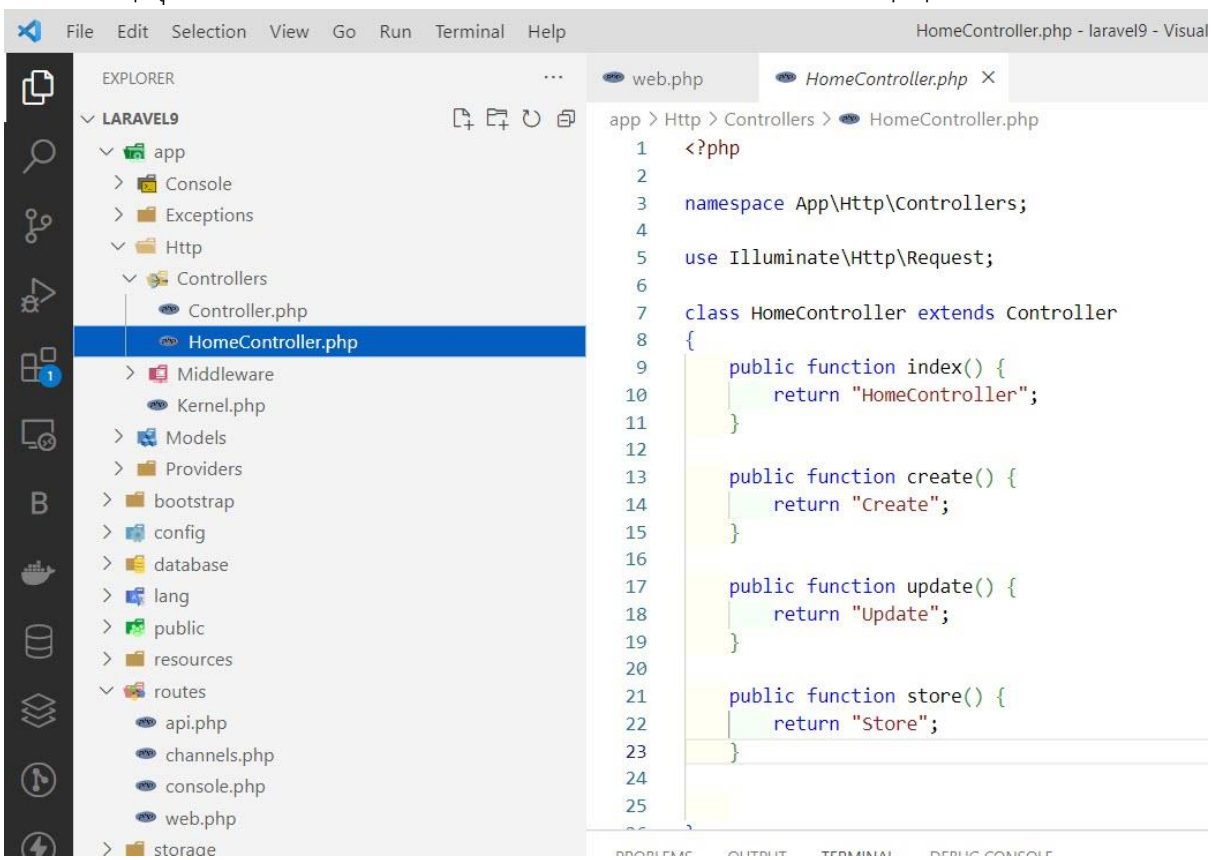
ใน Laravel 11 นั้น ทุกครั้งที่มีการเขียน routes แล้วมีการเชื่อมต่อกับ controllers การเรียกใช้ controllers จะต้อง use controllers นั้น ๆ ด้วย มิเช่นนั้นจะเรียกใช้งานไม่ได้ ก็คำสั่งนี้ครับ

use App\Http\Controllers\HomeController;

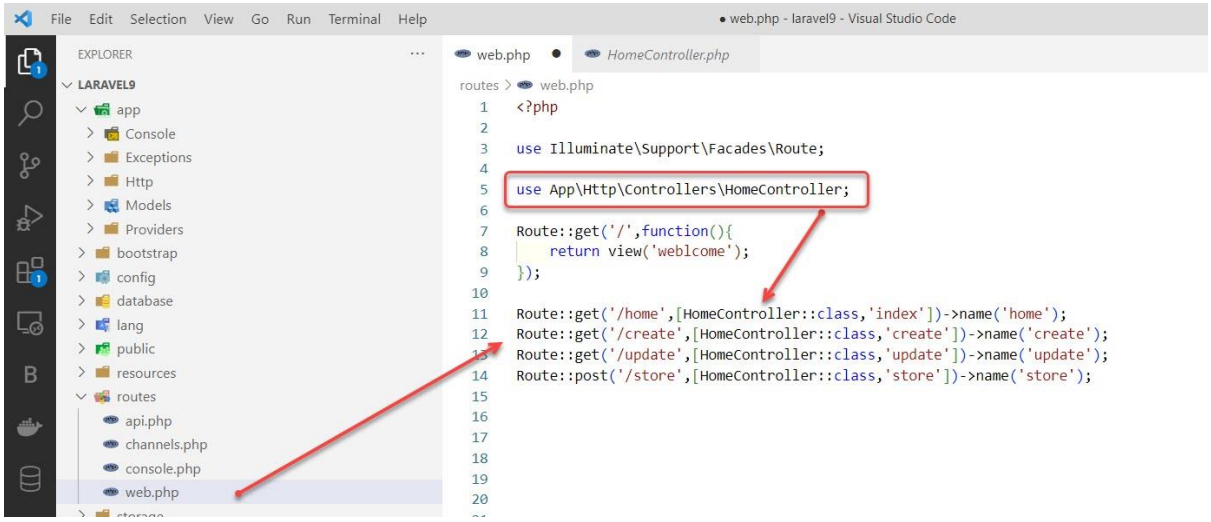
จากภาพด้านบนได้มีการสร้าง controllers ขึ้นมาคือ HomeControllers.php และมีการสร้างเมธอด index() เพื่อแสดงข้อมูลตามที่ต้องการ แล้วในไฟล์ web.php มีการเขียนคำสั่งเพื่อที่จะเรียกใช้ โดยตั้งชื่อ URL ชื่อว่า home แล้วเชื่อมต่อกับ controllers

```
Route::get('/home',[HomeController::class,'index'])->name('index');
```

HomeController::class คือ คอนโทรลเลอร์ที่เราจะเรียกใช้ **index** คือ เมธอดที่สร้างไว้ในคอนโทรลเลอร์ หากมีการสร้างเมธอดเพิ่มใน controllers เราก็ต้องมาเขียนโค้ดเพิ่มใน web.php เพื่อเรียกใช้เมธอดนั้น ๆ ทุกครั้งนะครับ ตัวอย่างนี้ ให้เพิ่มเมธอดลงไปใน HomeController.php



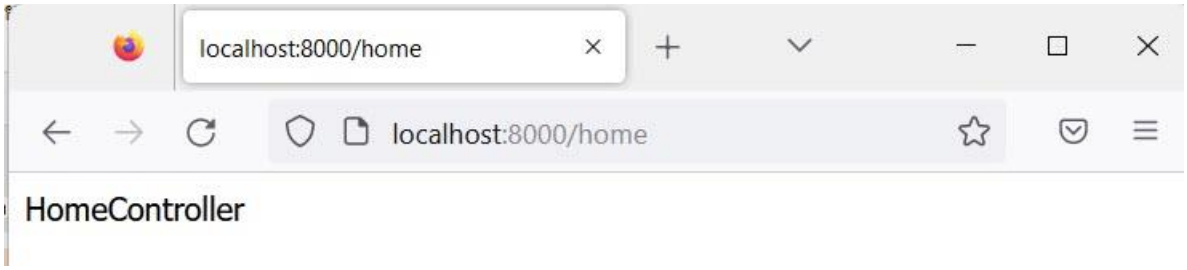
เวลาเรียกใช้ก็เขียนระบุลงไปในไฟล์ web.php ดังนี้



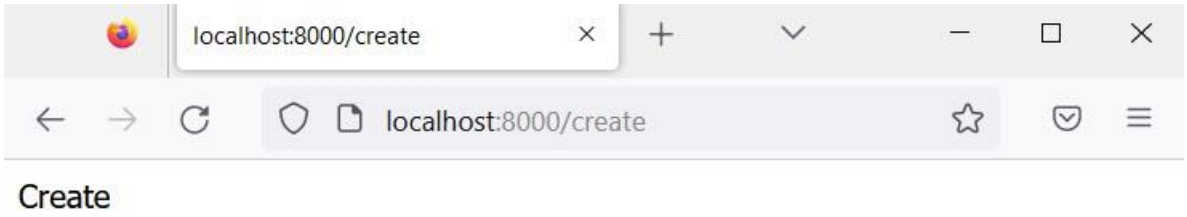
การเรียกใช้งานก็เรียกผ่านชื่อ routes ที่เราตั้งไว้เหมือนเดิม

เปิด Browser ขึ้นมาแล้วพิมพ์ localhost:8000/home , localhost:8000/create , localhost:8000/update

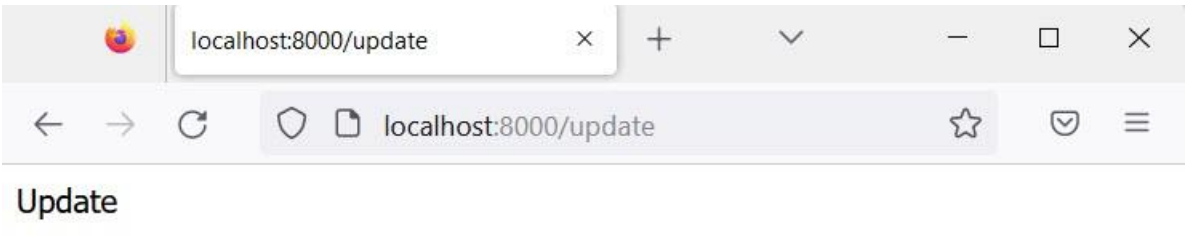
home จะเรียกหน้า index มาแสดงผล



create จะเรียกหน้า create มาแสดงผล

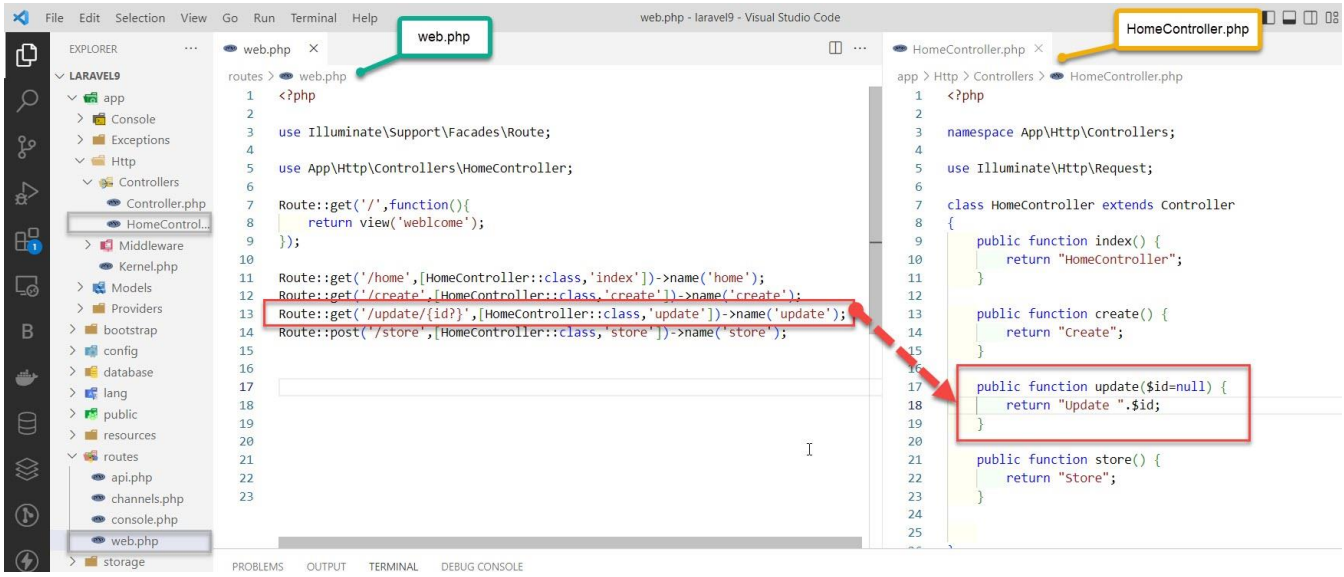


update จะเรียกหน้า update มาแสดงผล



การส่งค่า Parameters ไปให้ Controllers

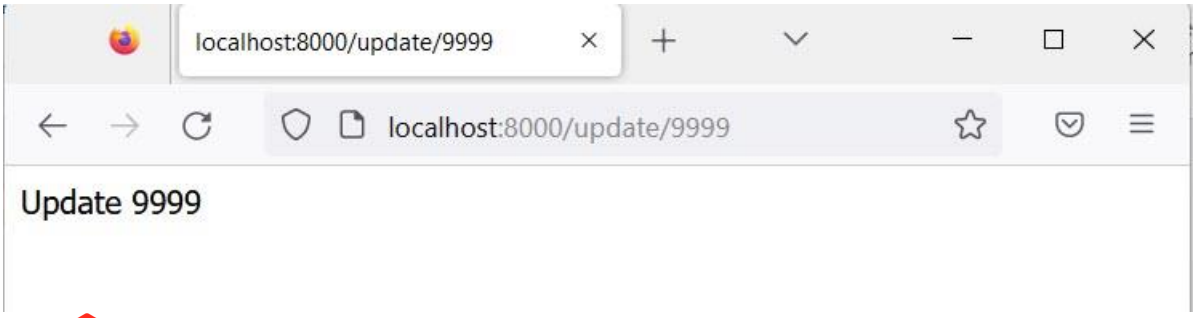
หากต้องการส่งค่าตัวแปรเข้าไปให้ controllers ใช้งาน สามารถทำได้ง่าย ๆ เราได้เรียนรู้เรื่องการส่งตัวแปรไปบ้างแล้วนั้น หลักการก็จะคล้าย ๆ กันจากที่เราได้เรียนมาเลยครับ เพียงแต่เปลี่ยนรูปแบบการเขียนนิดหน่อย ไปดูตัวอย่างการเขียนเลยครับ



1.เขียนโค้ดเพิ่มเข้าไปที่ไฟล์ web.php โดยใส่ {id?} ในส่วนที่ต้องการส่งตัวแปรไป ในตัวอย่างเราจะส่งไปที่เมธอด update ใน HomeController

2.เขียนโค้ดสำหรับการรับค่าตัวแปรในไฟล์ HomeController.php เมธอด update แล้วรับตัวแปรชื่อ $id จากนั้นก็คืนค่าตัวแปรออกมาแสดงผลที่หน้าจอ

เปิด Browser ขึ้นมาแล้วพิมพ์ localhost:8000/update/9999 ที่ Browser



Resource Controllers (CRUD)

เป็นอีกเครื่องมือหนึ่งที่ช่วยอำนวยความสะดวกให้กับผู้พัฒนา เนื่องจากว่า Resource Controllers จะเตรียมให้หมด สิ่งที่เตรียมให้คือเมธอดต่าง ๆ ที่มี CRUD (Create คือ การเพิ่มข้อมูล , Read คือ การอ่านหรือการดึงข้อมูลมาแสดงผล , Update คือ การอับเดตหรือแก้ไขข้อมูล , Delete คือ การลบข้อมูล) มาให้

Resource Controllers จะช่วยลดการเขียน routes หลาย ๆ บรรทัด ทำให้เราออกแรงเขียนโค้ดน้อยลง ตัวอย่างการเขียนแบบธรรมดา

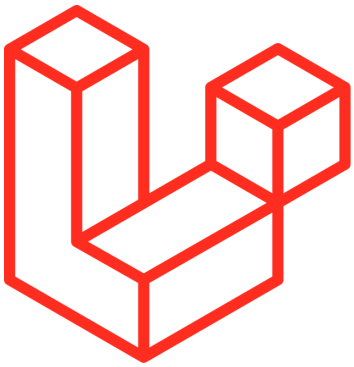
```
Route::get('/home',[HomeController::class,'index'])->name('home');
Route::get('/create',[HomeController::class,'create'])->name('create');
Route::get('/edit/{id?}',[HomeController::class,'edit'])->name('edit');
Route::put('/store',[HomeController::class,'store'])->name('store');
Route::post('/show/{id?},[HomeController::class,show])->name('show');
Route::delete('/destroy/{id?}',[HomeController::class,'destroy'])->name('destroy');
```

แต่เมื่อเราสร้าง controlers แบบ Resource Controllers ทุกอย่างจะเปลี่ยนไปทันที คือจะใช้คำสั่งในการเขียน routes จากเดิมที่เราเขียนหลาย ๆ บรรทัดเพื่อจัดการ routes พอเราสร้าง controllers แบบ resource เราจะเหลือคำสั่งในการสร้าง routes เพียงบรรทัดเดียว

คำสั่งที่ใช้สำหรับการสร้าง controllers resource ดังนี้

```
php artisan make:controller <ชื่อคอนโทรลเลอร์> --resource
```

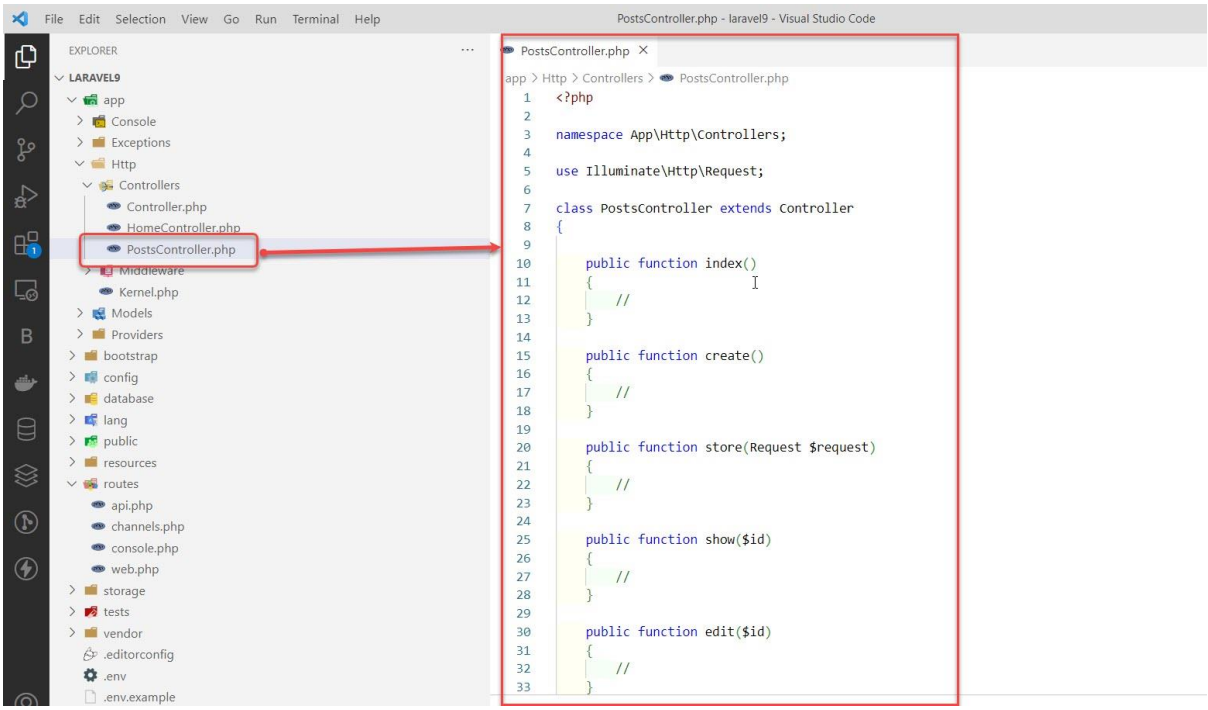
ทุกครั้งที่เราต้องการสร้าง controllers แบบ resource ให้เราระบุคำสั่ง --resource ต่อท้ายชื่อคอนโทรลเลอร์ ด้วยนะครับ ไปดูตัวอย่างการสร้างในหน้าต่อไปเลยครับ 😊



พิมพ์คำสั่งสำหรับการสร้าง ตามนี้ครับ (พิมพ์เสร็จแล้ว ก็กดปุ่ม Enter ด้วยนะครับ 😊)



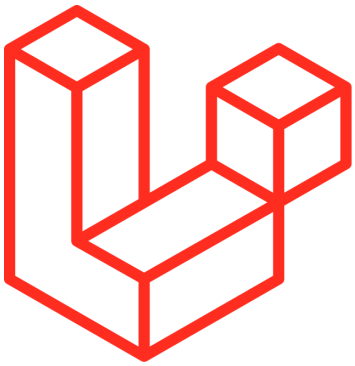
ก็จะมีการสร้างไฟล์ controllers ใหม่ขึ้นมาให้เราครับ เราได้ใช้คำสั่งด้านบนแล้วตั้งชื่อ controllers เป็น PostsController แล้วระบุคำสั่ง –resource ต่อท้าย แล้ว Artisan command ก็จะสร้างเมธอดให้เรา ตามนี้ครับ index(), create(), show($id), edit($id), store(Request $request), update(Request $request, $id), destroy($id)

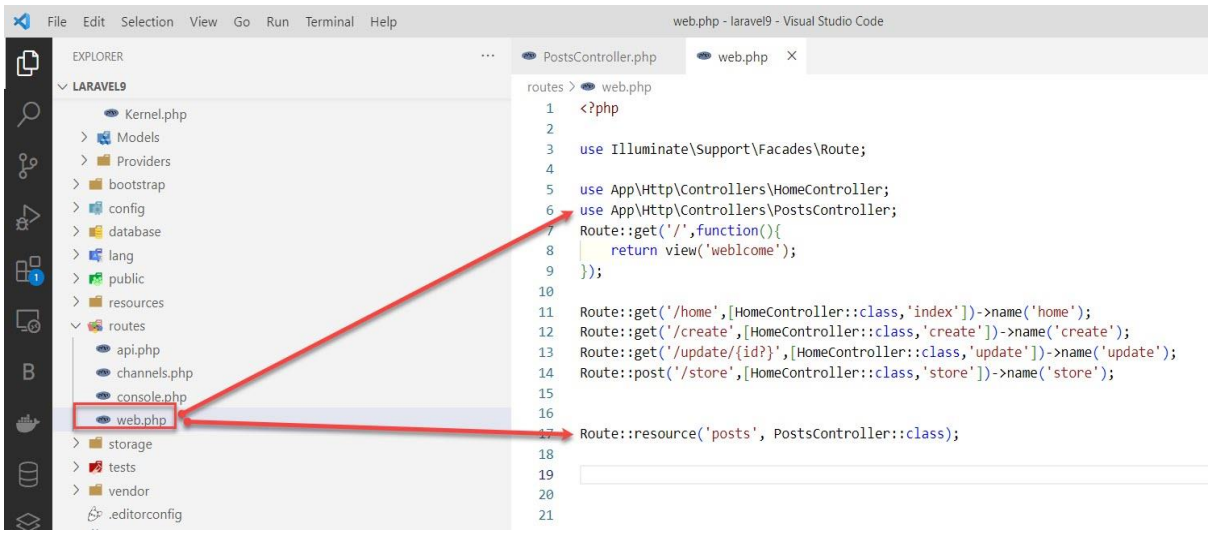


ในไฟล์ web.php ให้เราเพิ่ม routes เข้าไปด้วยคำสั่ง

Route::resource('posts', PostsController::class);

เข้าไปใหม่ ตามนี้





จะได้ routes ที่ถูกสร้างให้อัตโนมัติตามนี้

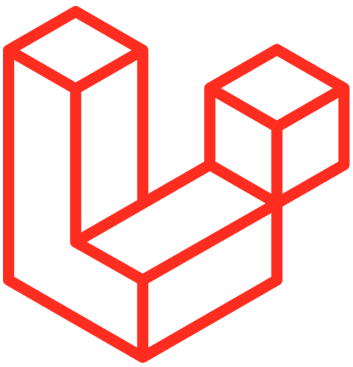
```
Route::get('/posts,[PostsController::class,'index']);
Route::get('/posts/create',[PostsController::class,'create']);
Route::get('/ posts/{id}',[PostsController::class,'show']);
Route::get('/posts/{id}/edit,[PostsController::class,edit]);
Route::post('/posts/{id?},[PostsController::class,store]);
Route::put('/posts/{id?},[PostsController::class,update]);
Route::delete('/posts/{id?}',[PostsController::class,'destroy']);
```

ใช่ไหม โม้ไปเองหรือเปล่า อ้าว ที่นี้ลองมาดูครับว่ามันสร้างให้จริงๆ ไหม เดียวว่าผมโม้ไปเอง ให้พิมพ์คำสั่งนี้ลงไปใน Terminal ตามนี้ครับ

พิมพ์คำสั่งนี้แล้วกดปุ่ม Enter

```
php artisan route:list
```

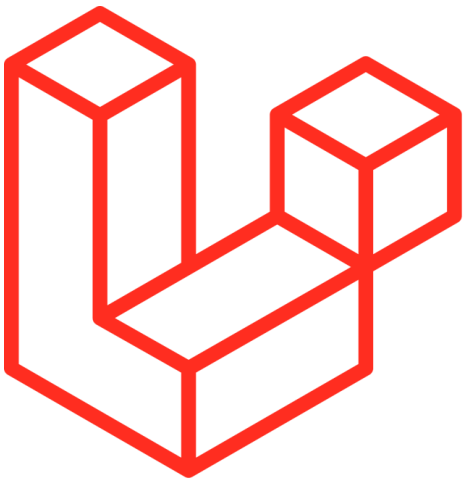
* route:list ใช้ดูโครงสร้าง uri ทั้งหมดในระบบเรา



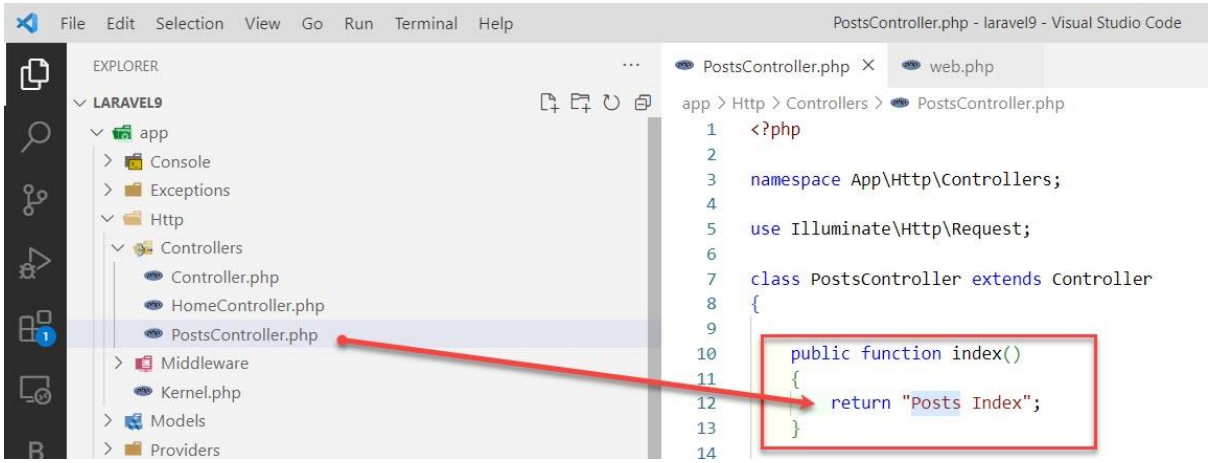
```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

PS D:\xampp\htdocs\laravel9> php artisan route:list

  GET|HEAD        / ..... 
  GET|HEAD        _debugbar/assets/javascript ..... debugbar.assets.js › Barryvdh\Debugbar › AssetController@js
  GET|HEAD        _debugbar/assets/stylesheets ..... debugbar.assets.css › Barryvdh\Debugbar › AssetController@css
  DELETE          _debugbar/cache/{key}/{tags?} ..... debugbar.cache.delete › Barryvdh\Debugbar › CacheController@delete
  GET|HEAD        _debugbar/clockwork/{id} ..... debugbar.clockwork › Barryvdh\Debugbar › OpenHandlerController@clockwork
  GET|HEAD        _debugbar/open ..... debugbar.openhandler › Barryvdh\Debugbar › OpenHandlerController@handle
  POST            _ignition/execute-solution ignition.executeSolution › Spatie\LaravelIgnition › ExecuteSolutionControll…
  GET|HEAD        _ignition/health-check ..... ignition.healthCheck › Spatie\LaravelIgnition › HealthCheckController
  POST            _ignition/update-config ..... ignition.updateConfig › Spatie\LaravelIgnition › UpdateConfigController
  GET|HEAD        api/user ..... 
  GET|HEAD        create ..... create › HomeController@create
  GET|HEAD        home ..... home › HomeController@index
  GET|HEAD        posts ..... posts.index › PostsController@index
  POST            posts ..... posts.store › PostsController@store
  GET|HEAD        posts/create ..... posts.create › PostsController@create
  GET|HEAD        posts/{post} ..... posts.show › PostsController@show
  PUT|PATCH       posts/{post} ..... posts.update › PostsController@update
  DELETE          posts/{post} ..... posts.destroy › PostsController@destroy
  GET|HEAD        posts/{post}/edit ..... posts.edit › PostsController@edit
  GET|HEAD        sanctum/csrf-cookie ..... sanctum.csrf-cookie › Laravel\Sanctum › CsrfCookieController@show
  POST            store ..... store › HomeController@store
  GET|HEAD        update/{id?} ..... update › HomeController@update

                                                                                Showing [22] routes
```

ความหมายแต่ละเมธอดที่ถูกสร้างขึ้นให้

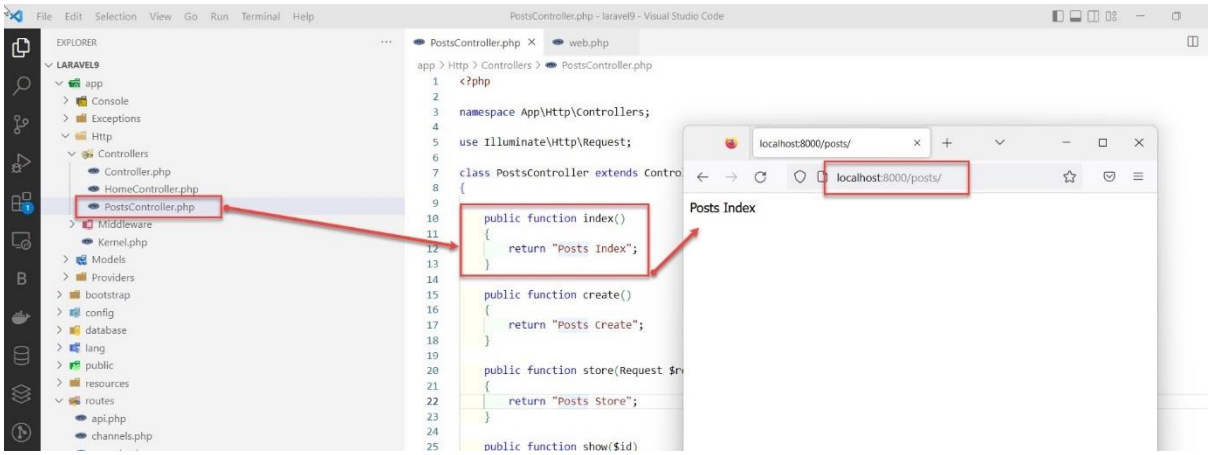
HTTP Verb	URI	เมธอด	ชื่อ routes	คำอธิบาย
GET	/posts	index	posts.index	แสดงหน้าแรก
GET	/posts/create	create	posts.create	แสดงหน้าฟอร์มกรอกข้อมูล
POST	/posts	store	posts.store	สำหรับบันทึกข้อมูล
GET	/posts/{post}	show	posts.show	แสดงข้อมูลตาม id
GET	/posts/{post}/edit	edit	posts.edit	แสดงหน้าแก้ไขข้อมูล
PUT/PATCH	/posts/{post}	update	postspdate	สำหรับบันทึกข้อมูลการแก้ไข
DELETE	/posts/{post}	Destroy	posts.destroy	สำหรับลบข้อมูล



ที่นี้ลองไปเพิ่มข้อความใน PostsController ในเมธอด index() ตามนี้ครับ

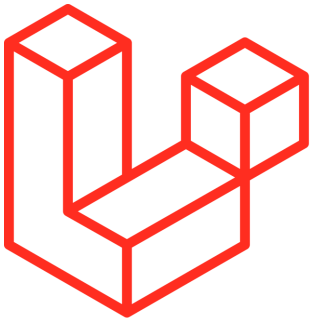


การเรียกใช้งาน ก็จะเรียกใช้งานผ่าน URL เช่น หากต้องการเรียกใช้งานหน้าแรกของ posts ก็ให้พิมพ์ URL localhost:8000/posts ถ้าต้องการเรียกหน้า ฟอร์มสำหรับกรอกข้อมูล ก็ให้พิมพ์ URL localhost:8000/posts/create



//ทุกอย่างต้องผ่านการฝึกฝน ลองผิด ลองถูก แล้วคุณจะเก่งเอง

“อย่าเพิ่งท้อนะ”

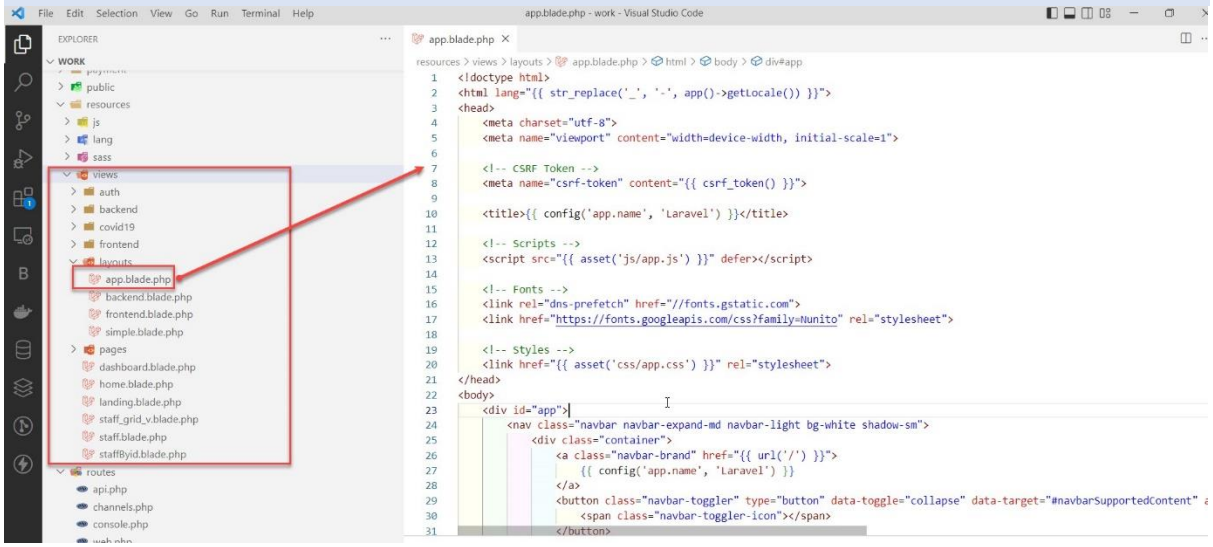


พื้นฐานการเขียน Views

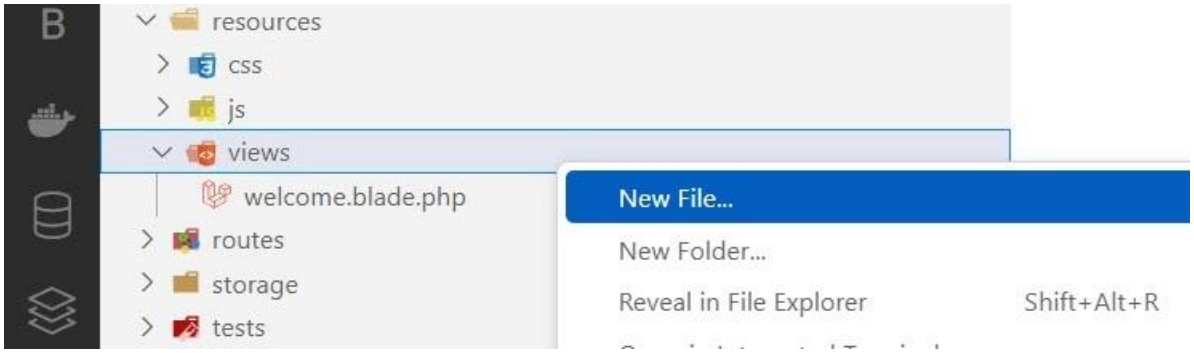
views เป็นส่วนที่ใช้สำหรับการเขียนโค้ดที่เป็นภาษา HTMP / PHP / CSS / JAVASCRIPT เพื่อใช้สำหรับการแสดงผลออกทางหน้าเจอ ใช้ออกแบบและควบคุม UI ต่าง ๆ ซึ่งเราจะเขียน Layout หรือหน้าเว็บแต่ละหน้าไว้ใน views จะช่วยให้เราบริหารจัดการ template ของเว็บได้ง่าย และที่สำคัญจะมี Template engine คือ blade template ให้ใช้อีกด้วย ยิ่งทำให้ง่ายต่อการบริหารจัดการ views อย่างมาก

views จะเขียนไว้ที่โฟลเดอร์ resources/views

resources/views/...



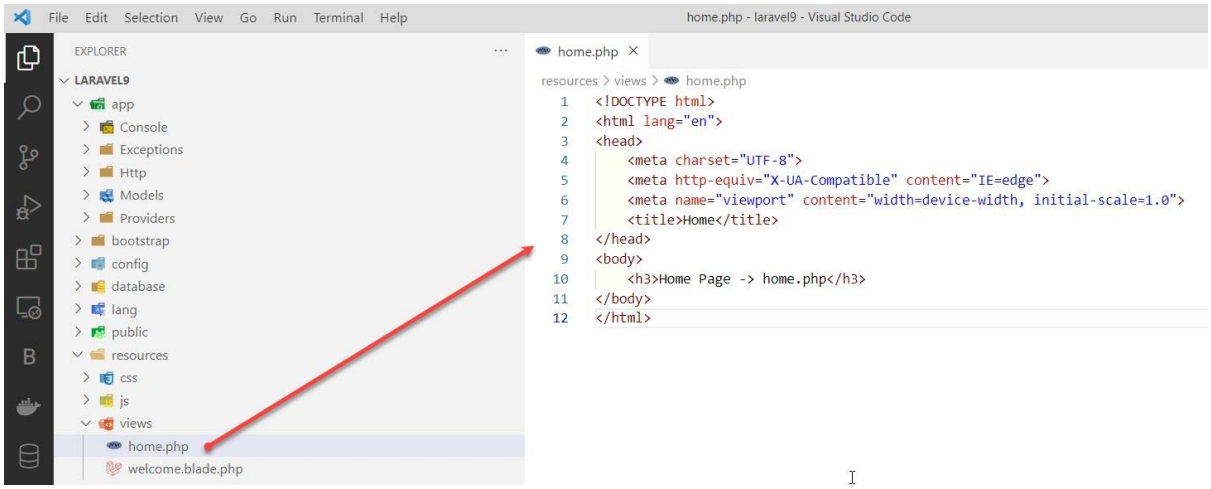
สามารถสร้างโฟลเดอร์ย่อยเพื่อเก็บ views แต่ละส่วนได้อย่างอิสระ ลองมาสร้างและใช้งานดูเลยครับ คลิกขวาที่โฟลเดอร์ views แล้วเลือก New File...



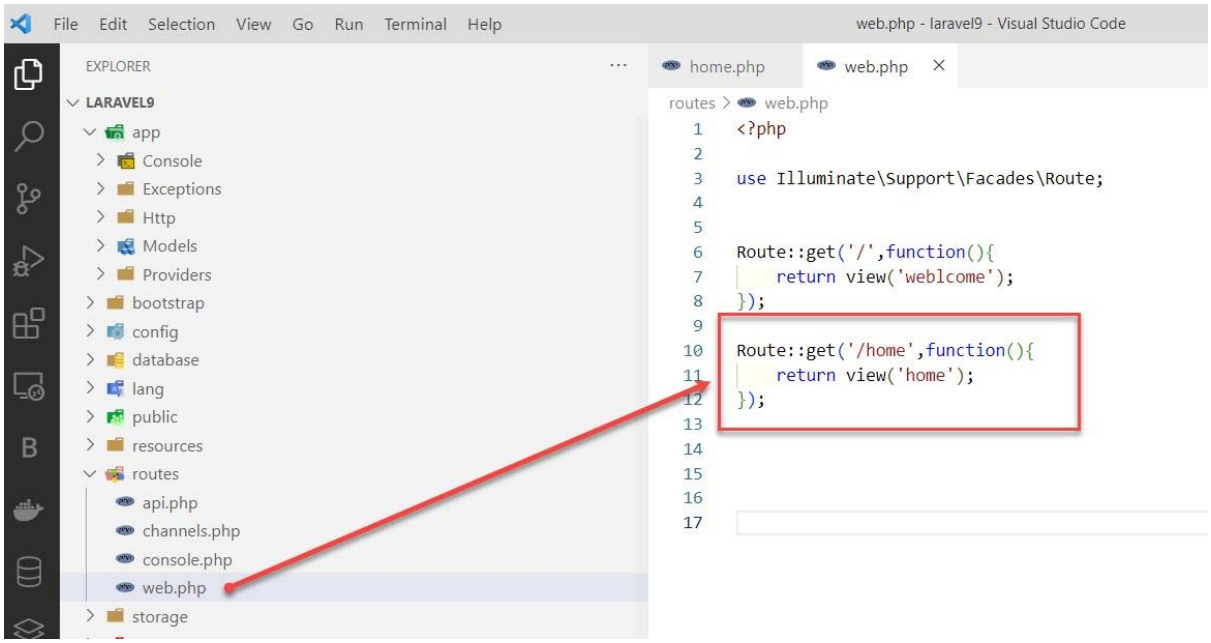
ตั้งชื่อที่ต้องการ



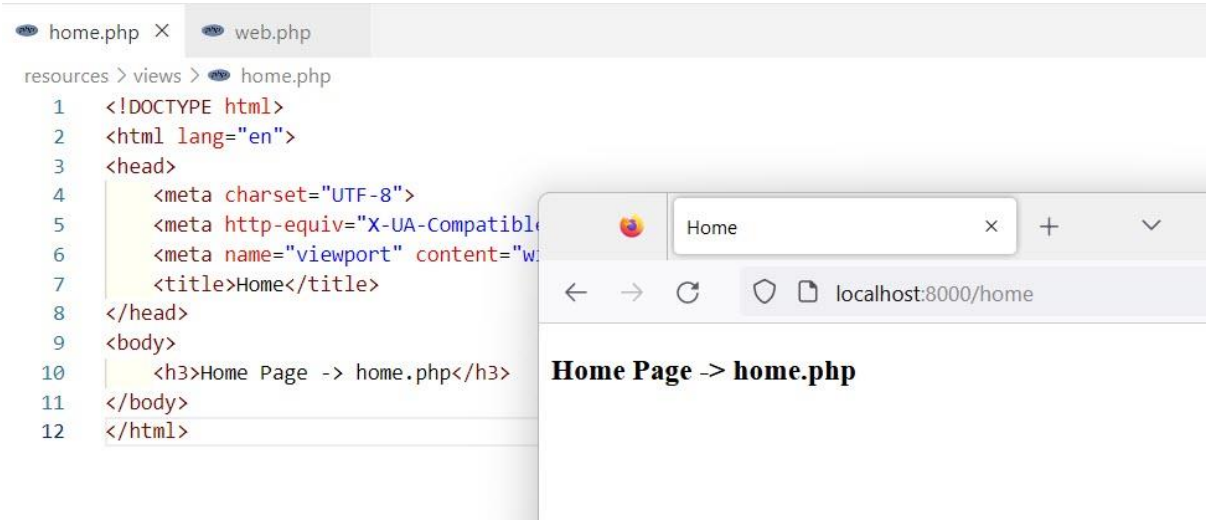
เขียนโค้ดเพิ่มเข้าไปในไฟล์ home.php



เขียนคำสั่งเพิ่มที่ routes/web.php เพื่อเรียกใช้งาน เขียนเสร็จก็อย่าลืม Save ด้วยนะ



เปิด Browser ขึ้นมาแล้วพิมพ์ localhost:8000/home



จะเห็นว่าจากโค้ดที่เราเขียนด้านบน เริ่มแรกเราได้สร้างไฟล์ home.php ไว้ที่โฟลเดอร์ views จากนั้นก็มีการเขียน routes เพื่อเรียกใช้งาน views ใน routes ที่เราเขียนก็จะทำการ render ไฟล์ home.php ด้วยคำสั่ง return view(...) เพื่อนำข้อมูลจาก views มาแสดงให้เราเห็นผ่านหน้าจอ

ทุกครั้งมีการเขียน views เวลาเรียกใช้การก็ต้องมากำหนดเส้นทางใน routes ด้วยนะครับ

ส่งตัวแปร ไปให้ views

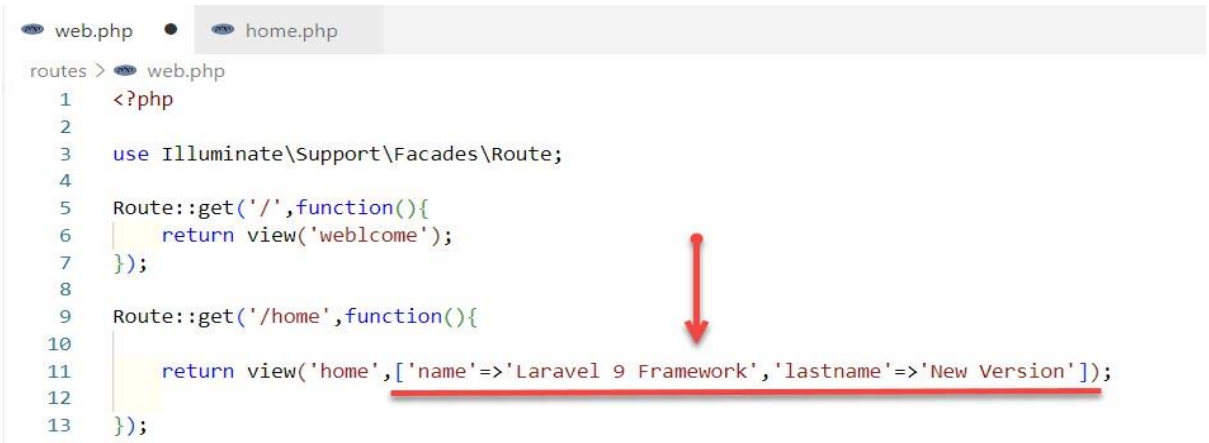
บางครั้งการเขียนเว็บไซต์หรือพัฒนาระบบเราคงหนีไม่พ้นการส่งตัวแปรต่าง ๆ ไปใช้งานในแต่ละหน้าเว็บ ใน Laravel นี้ก็เช่นกัน จะมีวิธีการส่งตัวแปรไปใช้งานได้หลายวิธี

วิธีที่ 1 ส่งค่าตัวแปรในรูปแบบ Associate Array

วิธีนี้จะเป็นการสร้าง array ขึ้นมาแล้วกำหนดชื่อตัวแปร และ ค่าลงไปพร้อมกันในตอนใส่ค่า

```
['name'=>'Laravel 11 Framework','lastname'=>'New Version']
```

เขียนโค้ดเพิ่มลงไปที่ไฟล์ routes/web.php โดยเพิ่มพารามิเตอร์ตัวที่ 2 เข้าไปอีกตัวตามภาพครับ



ที่ไฟล์ resource/views/home.php ให้เพิ่มโค้ดเข้าไป เพื่อให้แสดงผล ตามนี้ครับ

web.php home.php

resources > views > home.php

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Home</title>
8  </head>
9  <body>
10     <h3>Home Page - <?php echo $name ;?><br/><?php echo $lastname ; ?></h3>
11 </body>
12 </html>
```

เปิด Browser ขึ้นมาแล้วพิมพ์ localhost:8000/home ที่ Browser



วิธีที่ 2 ส่งค่าตัวแปรในรูปแบบฟังก์ชัน with()

```
->with('name','Laravel 11 Framework - With')->with('lastname','New Version - With');
```

เขียนโค้ดเพิ่มลงไปที่ไฟล์ rotest/web.php

web.php home.php

routes > web.php

```
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  Route::get('/',function(){
6      return view('weblcome');
7  });
8
9  Route::get('/home',function(){
10
11     //return view('home',['name'=>'Larave 9 Framework','lastname'=> 'New Version']);  //วิธีที่ 1
12
13     return view('home')->with('name','Laravel 9 Framework - With')->with('lastname','New Version - With');
14
15 });
16
```

วิธีที่ 3 ส่งค่าตัวแปรในรูปแบบฟังก์ชัน compact()

เขียนโค้ดเพิ่มลงไปที่ไฟล์ routes/web.php

web.php

routes > web.php

```php
<?php

use Illuminate\Support\Facades\Route;

Route::get('/',function(){
    return view('weblcome');
});

Route::get('/home',function(){

    //return view('home',['name'=>'Larave 9 Framework','lastname'=> 'New Version']);  //วิธีที่ 1

    //return view('home')->with('name','Laravel 9 Framework - With')->with('lastname','New Version - With'); //วิธีที่ 2

    $data = ['name'=>'Laravel 9 Framework - Compact','lastname' => 'New Version - Compact'] ;
    return view('home',compact('data'));
});
```

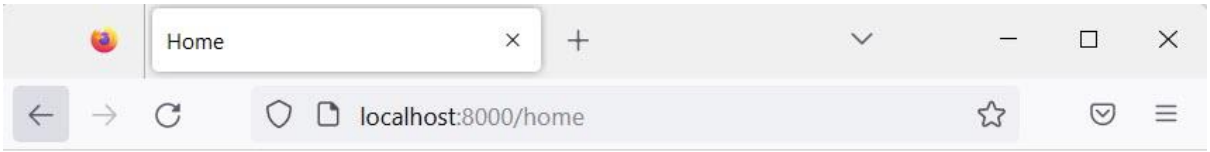
ที่ไฟล์ resources/views/home.php ให้เพิ่มโค้ดเข้าไป เพื่อให้แสดงผล ตามนี้ครับ

web.php | home.php

resources > views > home.php

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home</title>
</head>
<body>
    <h3>Home Page - <?php echo $data['name'] ;?><br/><?php echo $data['lastname'] ; ?></h3>
</body>
</html>
```

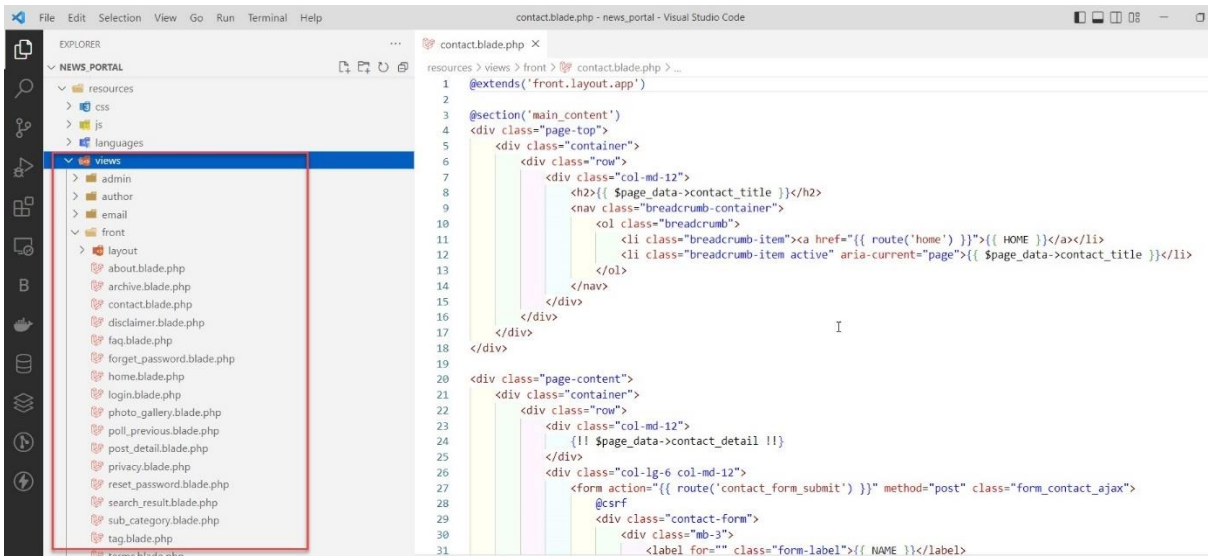
เปิด Browser ขึ้นมาแล้วพิมพ์ localhost:8000/home ที่ Browser



Home Page - Laravel 9 Framework - Compact
New Version - Compact

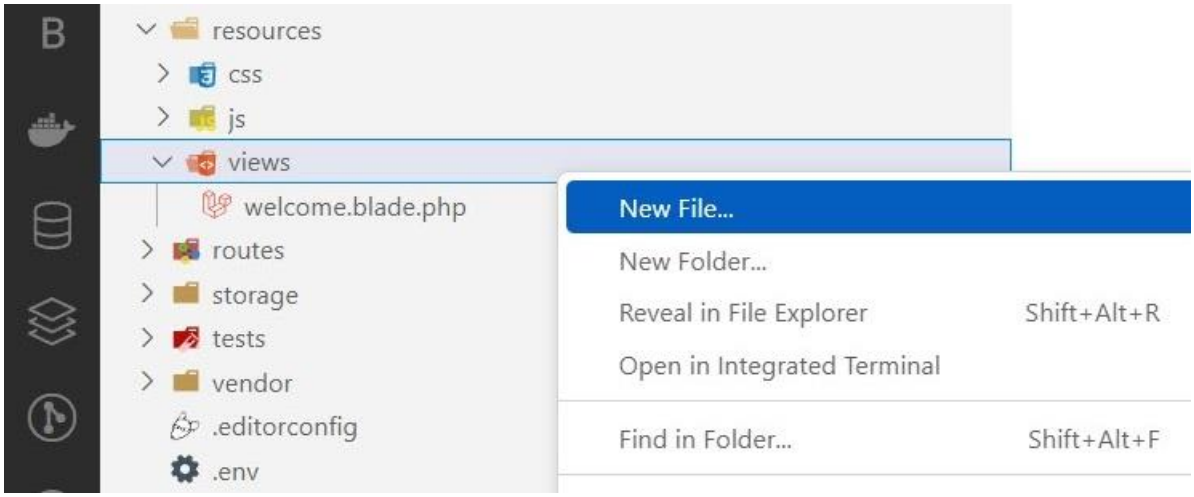
Blade Template

ปกติแล้วเราจะสร้างเป็นไฟล์สำหรับเขียน views เป็นนามสกุล .php ธรรมดาก็ได้ แต่ใน Laravel จะมี template ให้เรียกใช้ เรียกว่า blade คือชุดคำสั่งมาตรฐานในการแสดงผลข้อมูลใน Laravel Framework สามารถเขียนชุดคำสั่งของภาษา PHP ได้ ซึ่งจะช่วยให้เราเขียนไฟล์เพื่อแสดงผลได้ง่ายและมีความสามารถเยอะกว่าแบบเดิม การใช้ blade ก็เพียงแค่กำหนดชื่อไฟล์ลงท้ายด้วย .blade.php ในตอนที่เราสร้างนั่นเอง ให้มองว่า blade มันคือ views นั่นแหละ จะเขียนไว้ที่โฟลเดอร์ resources/views เหมือนเดิม

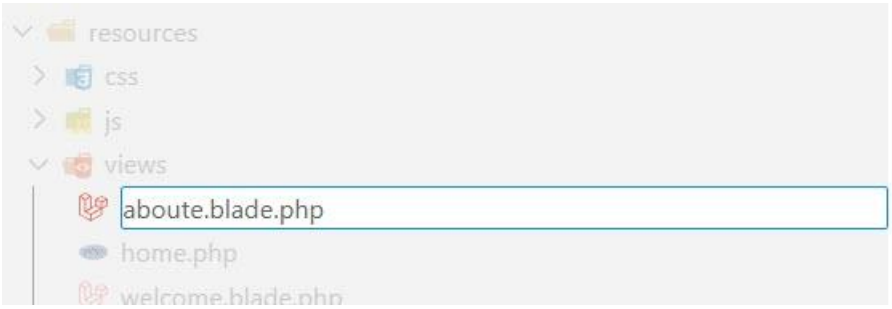


ลองสร้างไฟล์แล้วเรียกใช้งาน ตามขั้นตอนนี้ได้เลย

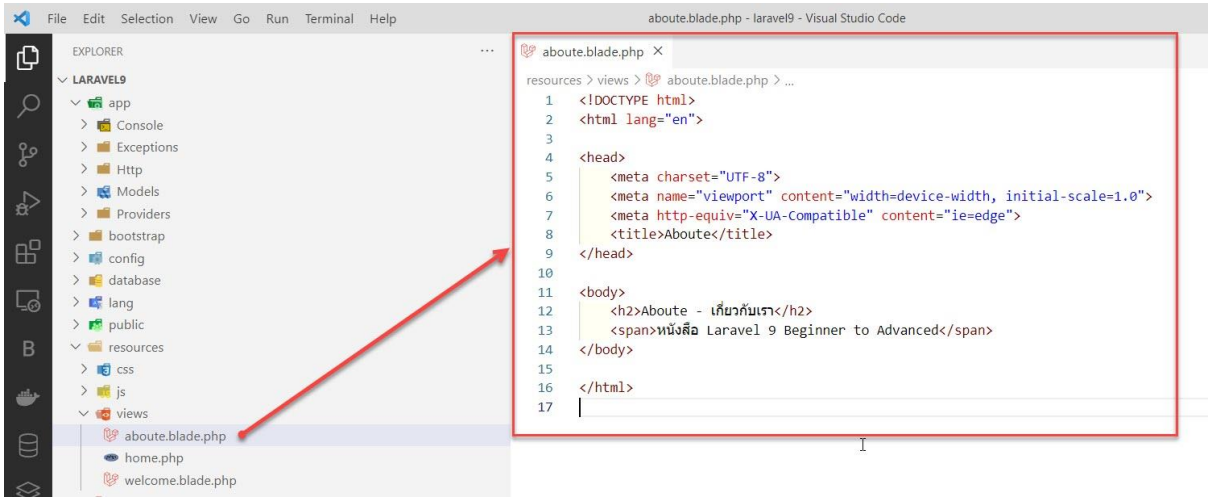
1.คลิกขวาที่โฟลเดอร์ views ตามภาพ แล้วเลือก New File...



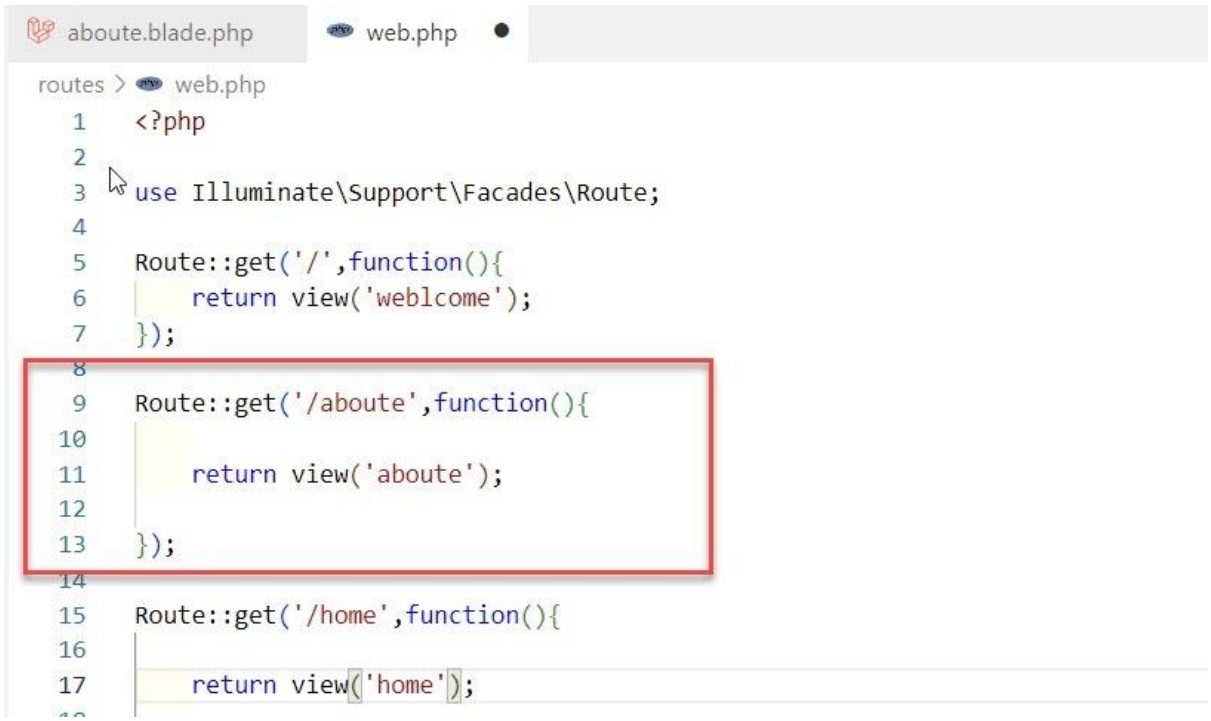
2.ตั้งชื่อไฟล์ aboute.blade.php อย่าลืมนะการตั้งชื่อ views ให้พิมพ์ชื่อและตามด้วย .blade.php ทุกครั้ง



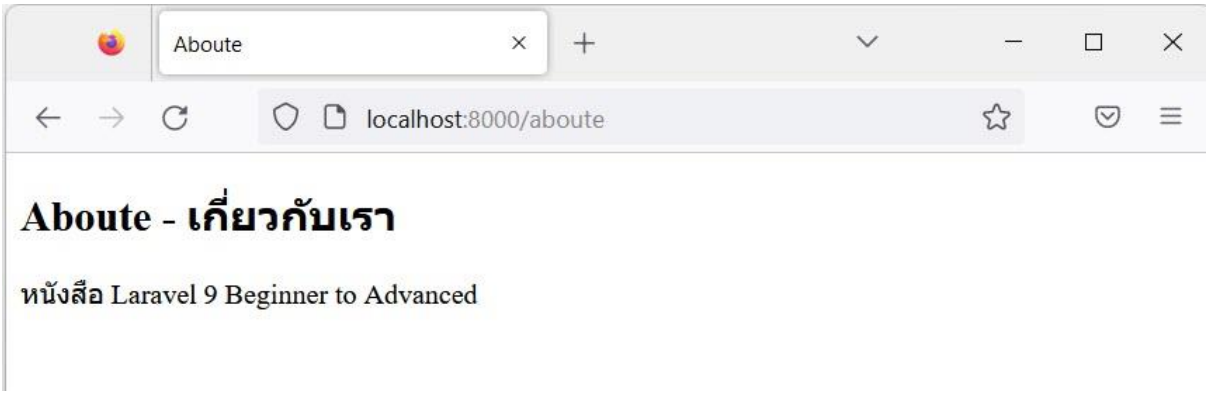
3.เปิดไฟล์ aboute.blade.php แล้วพิมพ์โค้ดเข้าไป



4.เปิดไฟล์ routes/web.php เพื่อสร้างเส้นทาง สำหรับการเข้าถึง เขียนโค้ดดังนี้



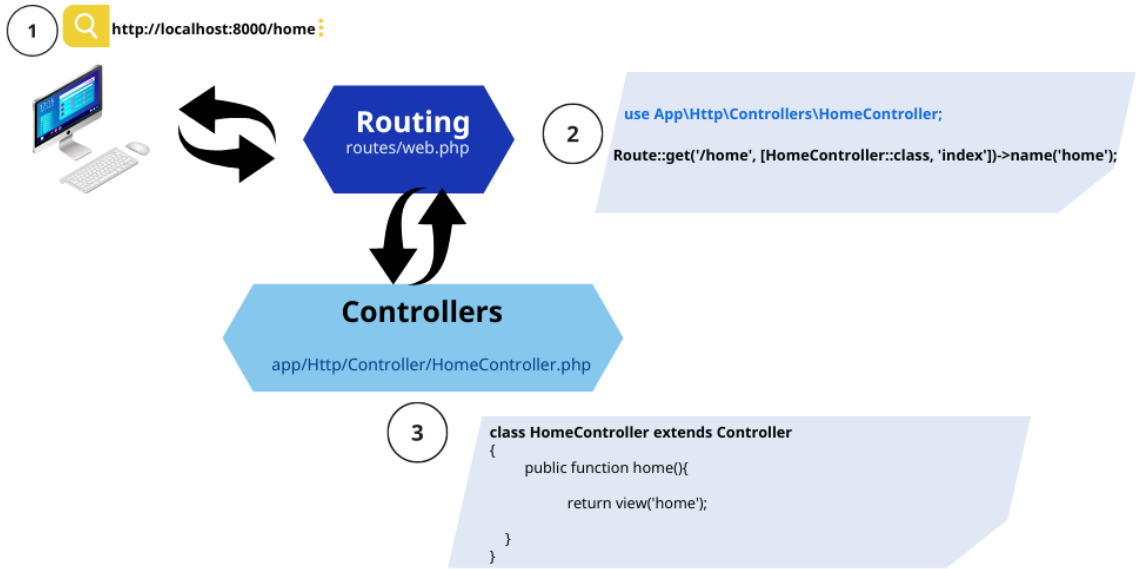
เปิด Browser ขึ้นมาแล้วพิมพ์ localhost:8000/aboute ที่ Browser



เพียงเท่านี้เราก็สามารถสร้างหน้าเว็บเรียบร้อยแล้ว ลองฝึกเขียนบ่อย ๆ นะครับ

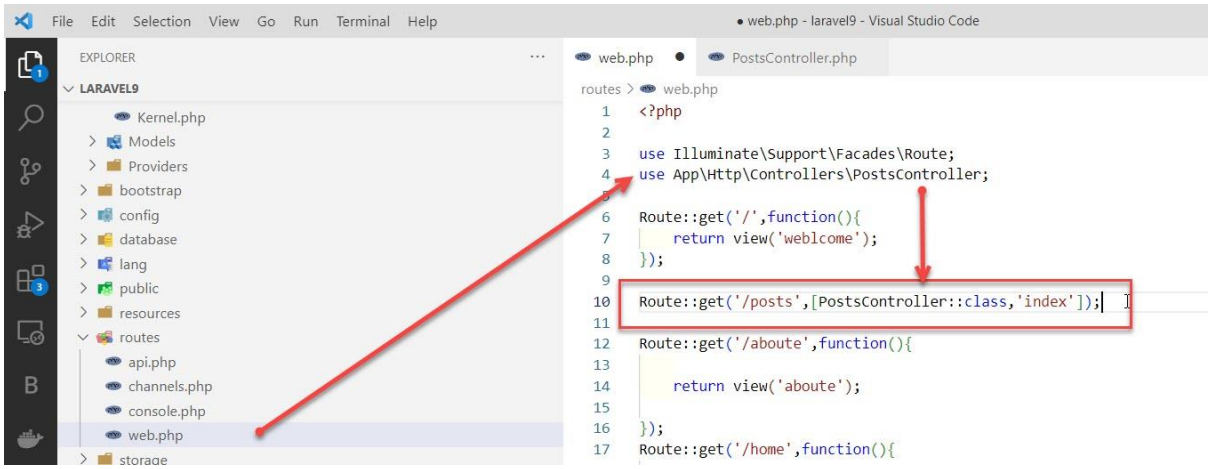
เชื่อมโยง Routes – Controllers – Views ใช้งานร่วมกัน

จากบทเรียนที่ผ่านมาเราได้เรียนรู้เกี่ยวกับการสร้างและใช้งานส่วนต่าง ๆ แล้ว คิดว่าก็คงจะเข้าใจนะครับ ว่าหลักการทำงานและการเขียนส่วนต่าง ๆ เป็นอย่างไร ในการเขียนโค้ดโดยความเป็นจริงแล้ว เมื่อเว็บเรามี URI หรือ Routes เพิ่มขึ้นเรื่อย ๆ เราไม่ควรเขียนทุกอย่างที่ routes/web.php ที่จุดเดียว ซึ่งจะทำให้จัดการลำบาก ให้เราเขียนแยกไว้ทีละส่วน โดยให้ใช้ controllers เป็นส่วนที่จะทำหน้าที่เชื่อมโยงกับส่วนอื่น ๆ แทน เช่น Views หรือ Model แล้วค่อยส่งผลที่ได้กลับมายังส่วนการทำ Routing เพื่อแสดงผลลัพธ์กลับไปยังหน้าเว็บอีกที



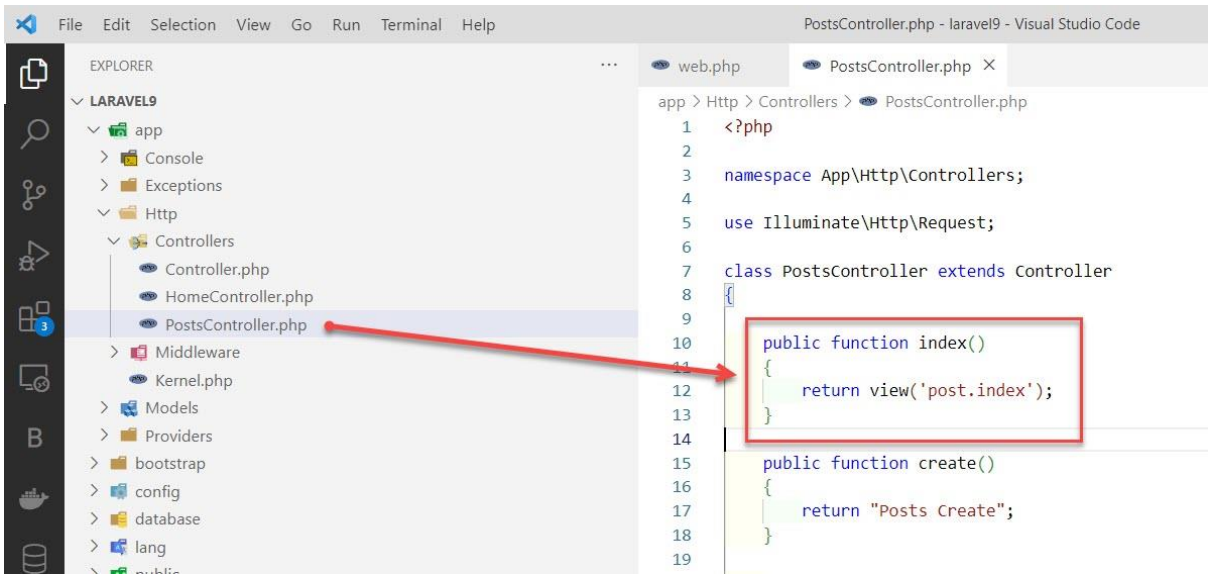
เพื่อให้เราทำความเข้าใจกระบวนการทำงานมากขึ้น ลองมาสร้างหน้าเว็บใหม่เพิ่มอีก 1 หน้า แล้วเขียนโค้ดให้เชื่อมต่อระหว่าง controllers และ views

1.เปิดไฟล์ routes/web.php เพื่อสร้างเส้นทาง สำหรับการเข้าถึง เขียนโค้ดดังนี้



ก่อนหน้านั้นเราได้สร้างคอนโทรลเลอร์ PostsController.php ไว้แล้ว ก็สามารถเรียกใช้งานได้เลย โดยการ use PostsController ที่ด้านบนก่อนจะเริ่มสร้าง Route จากนั้นในส่วนของการเขียน routes เราจะให้ชี้ไปยังคอนโทรลเลอร์ชื่อ PostsController และให้ไปทำงานที่เมธอด ชื่อว่า index()

2.เปิดไฟล์ PostsController.php แล้วแก้ไขโค้ดตามนี้



ในเมธอด index() ให้เพิ่ม คำสั่งสำหรับเชื่อมต่อกับ views โดยการใช้คำสั่ง return view() แล้วระบุชื่อโฟลเดอร์หรือชื่อไฟล์ที่เราต้องการจะเชื่อมต่อ

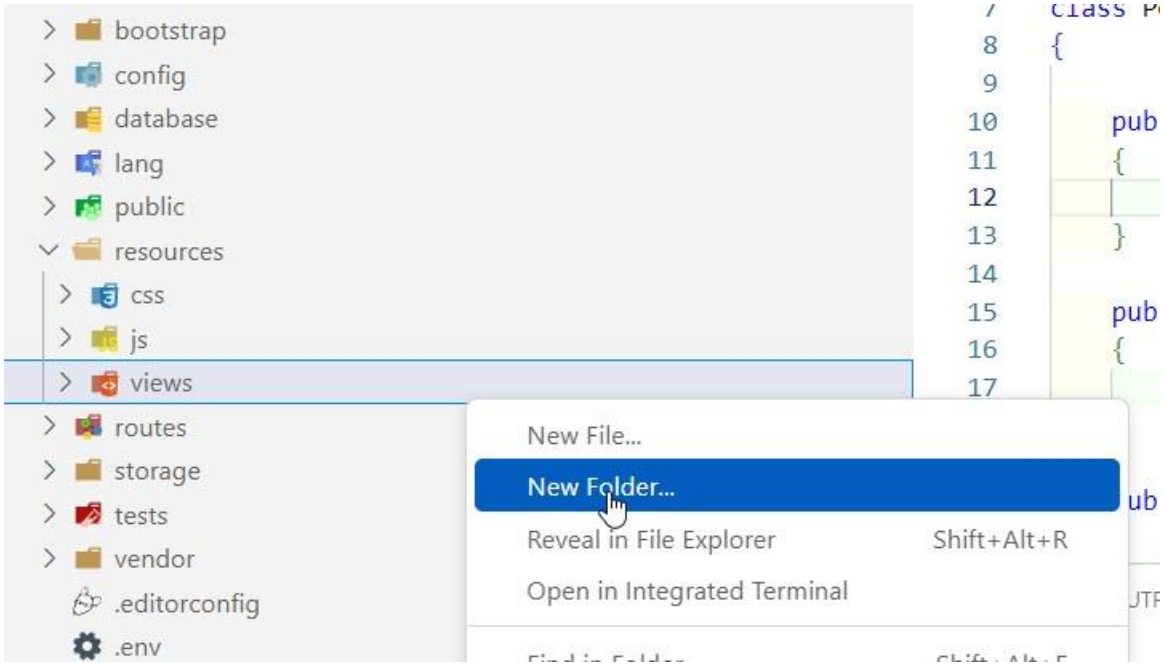
ตอนนี้เราได้ทำการสร้าง routes กับ controllers เสร็จแล้ว ถ้าเราสังเกตในเมธอด index() จะเห็นว่าเราได้เขียนโค้ดเพื่อสั่งให้ render ไปที่ views ชื่อว่า post.index (ไปสร้าง views เพิ่มกันต่อเลยครับ)

ที่นี้เรายังไม่มีการสร้าง views ชื่อว่า post.index เลย การสร้างก็เหมือนเดิมครับ ให้ไปสร้างไฟล์ views ไว้ที่โฟลเดอร์ resources/views

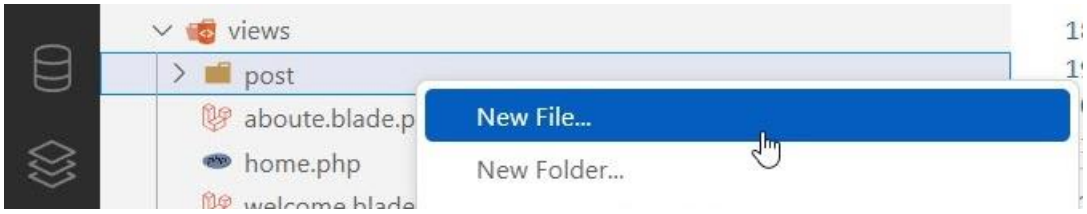
ในโฟลเดอร์ views เราสามารถสร้างโฟลเดอร์ซ้อนกันได้ ถ้ามีโฟลเดอร์ซ้อนกันก็สามารถสร้างได้ เช่น return view('post.index') จะหมายความว่า post เป็นโฟลเดอร์ที่สร้างขึ้นใหม่ และ index ก็คือไฟล์

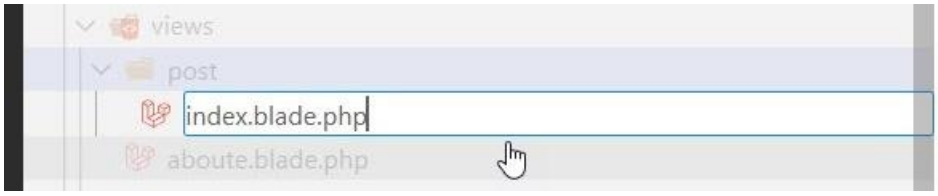
index.blade.php นั่นเอง เพียงแต่ตอนเราเขียนเรียกใช้ใน controllers เราไม่ต้องระบุนามสกุล .blade.php

3.คลิกขวาที่โฟลเดอร์ views แล้วเลือก New Folder...แล้วตั้งชื่อโฟลเดอร์ ชื่อว่า post

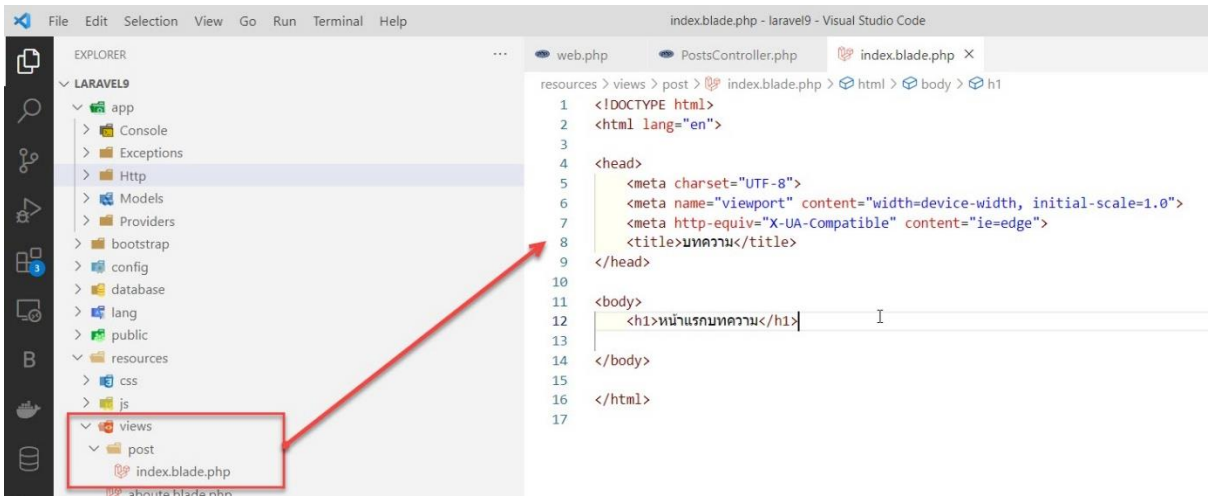


โฟลเดอร์ post จะถูกสร้างเก็บไว้ภายใต้โฟลเดอร์ views จากนั้นคลิกขาวที่โฟลเดอร์ post แล้วเลือก New File... เพื่อสร้างไฟล์ชื่อ index.blade.php

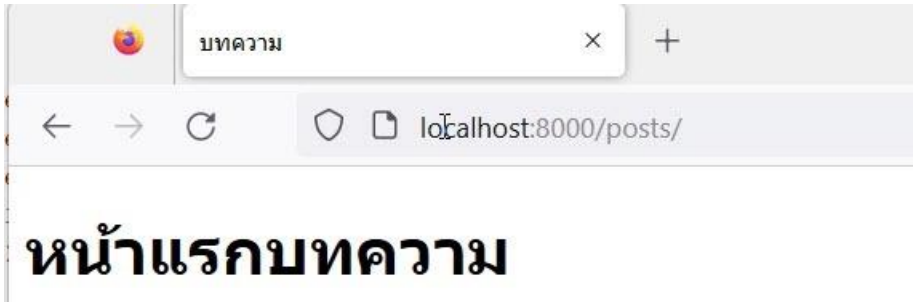




4.เปิดไฟล์ index.blade.php แล้วเพิ่มโค้ดเข้าไป ดังนี้



5.บันทึกไฟล์ แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/posts ที่ Browser



6.ส่งข้อมูลผ่านตัวแปร ไปแสดงผลที่ views สามารถเขียนเพิ่มได้ โดยเพิ่มคำสั่งที่เมธอด index() ดังนี้

app > Http > Controllers > PostsController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PostsController extends Controller
{

    public function index()
    {
        $PostName = "หน้าแรกบทความ";
        $CountPost = "จำนวนบทความ 100 หน้า";
        return view('post.index',compact('PostName','CountPost'));
    }
}
```

แปรชือ $PostName และ $CountPost เพื่อไปแสดงผลที่ views

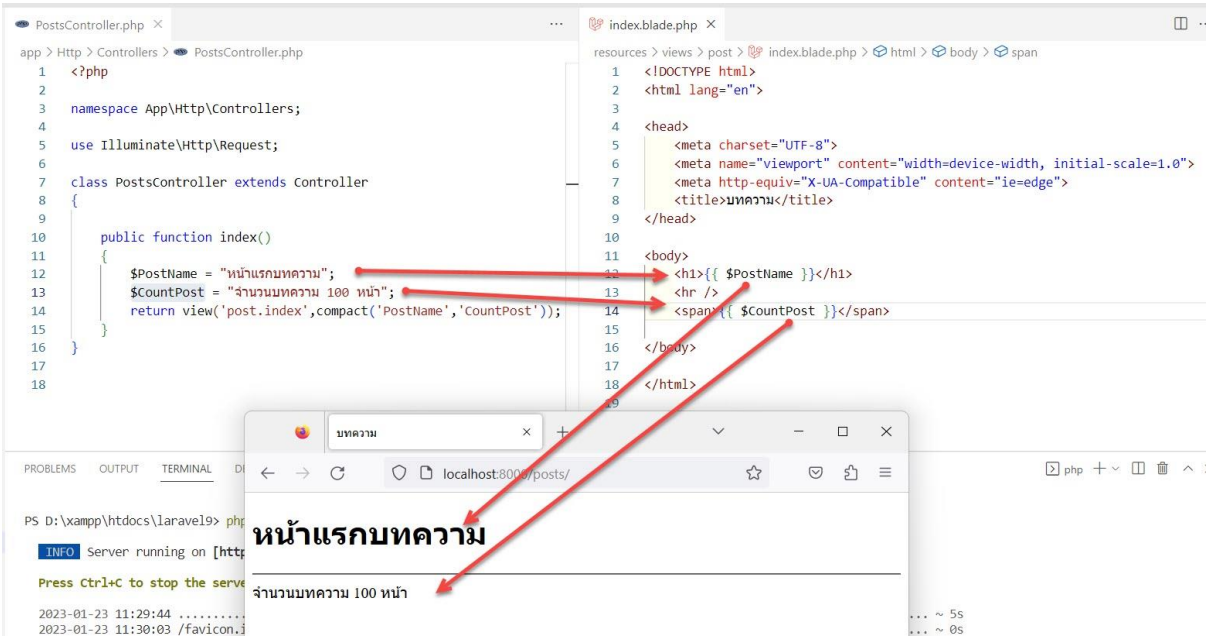
7.เขียนโค้ดเพื่อแสดงค่าข้อมูลของตัวแปรที่ส่งมา ได้แก่ $PostName และ $CountPost ดังนี้

PostsController.php | index.blade.php

resources > views > post > index.blade.php > html > body > span

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <meta http-equiv="X-UA-Compatible" content="ie=edge">
8      <title>บทความ</title>
9  </head>
10
11 <body>
12     <h1>{{ $PostName }}</h1>
13     <hr />
14     <span>{{ $CountPost }}</span>
15
16 </body>
17
18 </html>
```

8.บันทึกไฟล์แล้วทดสอบรันดูอีกรอบ พิมพ์ localhost:8000/posts ที่ Browser



สร้าง Template หลัก (Master Layouts)

Template คือ รูปแบบโครงสร้างหน้าตาของเว็บไซต์ที่เป็นต้นแบบ สามารถออกแบบขึ้นมาเองได้ หรือสามารถดาวน์โหลดจากเว็บที่บริการ template ได้ซึ่งมีให้เลือกใช้มากมายทั้งที่มีแจกฟรีและจัดจำหน่าย อยากได้หน้าเว็บแบบไหน ก็เลือกดาวน์โหลดมาใช้ตามต้องการ สามารถดาวน์โหลดมาใช้ใน Laravel ได้ เพียงแค่เรามาปรับแต่งเพิ่มเติมนิดหน่อย

ใน Laravel เราจะใช้ Blade Template ในการจัดการ Layout หรือ Section แต่ละส่วน ช่วยให้เราสร้างหน้าเว็บได้รวดเร็วและสะดวกมากขึ้น โดยที่เราไม่ต้องเขียน HTML ซ้ำ ๆ กัน Blade Template นั้นจะทำให้เราบริหารจัดการข้อมูลแต่ละหน้าสามารถเปลี่ยนแปลงได้ตามที่เราต้องการ โดยจะใช้คำสั่ง @yield(...) เป็นส่วนแยกการแสดงผลแต่ละหน้าเว็บ ถ้าใน PHP ก็จะใช้คำสั่ง include หรือ require

การใช้งาน คำสั่ง @yield('ชื่อที่ต้องการ') เช่น

```
@yield('title')
@yield('css_before')

@yield('main_content')

@yield('js_before')
```

โครงสร้างไฟล์ HTML Master Layoute

```
<!-- resources/views/layouts/app.blade.php -->

 <html>

   <head>

      <title>App Name - @yield('title')</title>
      <link rel="stylesheet" href="{{ asset('dist-front/css/bootstrap.min.css') }}">
      @yield('css_before')

   </head>

   <body>
            <div class="container">

                @yield('content')

              </div>

   </body>
      <script src="{{ asset('dist-front/js/jquery-3.6.0.min.js') }}"></script>
      @yield('js_before')

</html>
```

จากโครงสร้างไฟล์ HTML จะเห็นว่ามีการเขียนคำสั่งแยกส่วนของโค้ดไว้เป็นส่วน ๆ ดังนี้

@yield('title') จะเป็นส่วนแยกข้อความที่แถบไตเติ้ลบาร์ของเว็บ browser หากมีหน้าเว็บใดเรียกใช้งานก็จะแสดงข้อความตามหน้าเว็บนั้น ๆ กำหนดไว้

@yield('css_before') จะเป็นส่วนของการนำไฟล์ CSS จากภายนอกในกรณีที่แต่ละหน้าเว็บต้องการแสดงรูปแบบการแสดงผลเพิ่มเติม

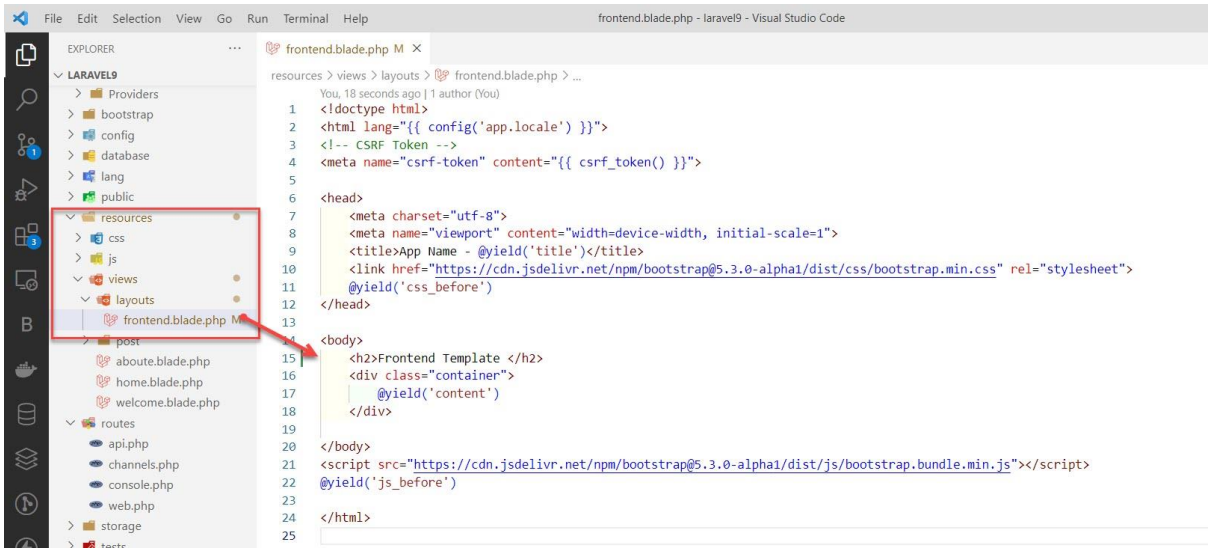
@yield('content') จะเป็นส่วนของการแสดงเนื้อหาที่แตกต่างกันของแต่ละหน้าเว็บ

@yield('js_before') จะเป็นส่วนของการนำไฟล์ JavaScript เข้ามาใช้

เราสามารถสร้าง Master Layouts ได้หลายไฟล์ ซึ่งแต่ละหน้าเว็บเราอาจจะใช้การแสดงผลที่ไม่เหมือนกัน เช่น ส่วนของหน้าผู้ใช้งานทั่วไป ส่วนของหน้าสำหรับสมาชิก ส่วนของหน้าสำหรับผู้ดูแลระบบ

การสร้าง template หลักนั้น เราจะสร้างไฟล์เก็บไว้ที่โฟลเดอร์ resources/views ไปสร้างกันเลยครับ เพื่อให้เป็นหมวดหมู่เดียวกัน ในที่นี้จะสร้างโฟลเดอร์ ชื่อว่า layouts ขึ้นมาก่อน เพื่อเก็บไฟล์ที่เป็น template ต่าง ๆ จากนั้นสร้างชื่อไฟล์ในโฟลเดอร์ layouts ชื่อว่า frontend.blade.php แล้วเขียนโค้ด HTML ลงไปดังนี้

- โฟลเดอร์เก็บ resources/views/layouts/frontend.blade.php



สามารถคัดลอก Source Code ได้ที่ลิงค์ http://bit.ly/3ZYMMXp

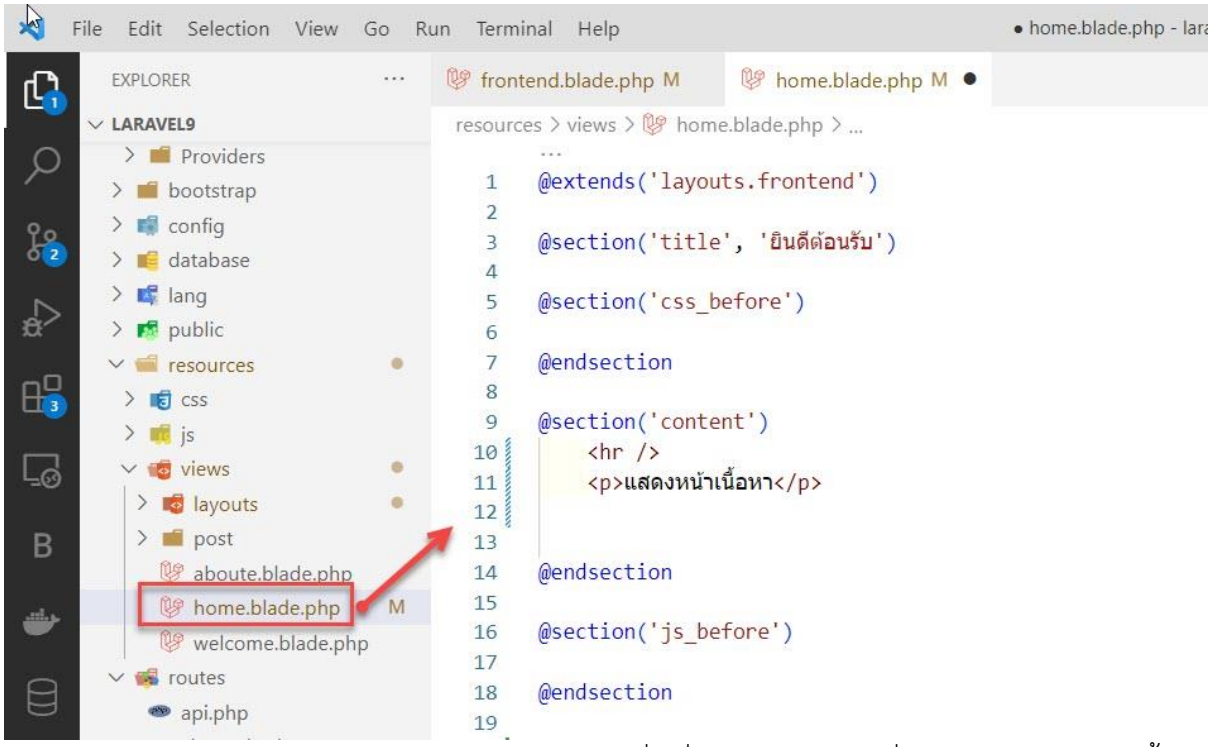
ตัวอย่างด้านบนเป็นการสร้าง template โดยใช้ css bootstrap ในการจัดการ UI ต่าง ๆ โดยเบื้องต้นนั้นจะใช้รูปแบบการเรียกใช้แบบ CDN (Content Delivery Network) ดังนั้นต้องเชื่อมต่ออินเตอร์เน็ตด้วยนะครับ มิเช่นนั้นจะใช้ CSS ไม่ได้

การใช้งาน css bootstrap เดียวจะเสริมให้ในหัวข้อถัดไป นะครับ ...

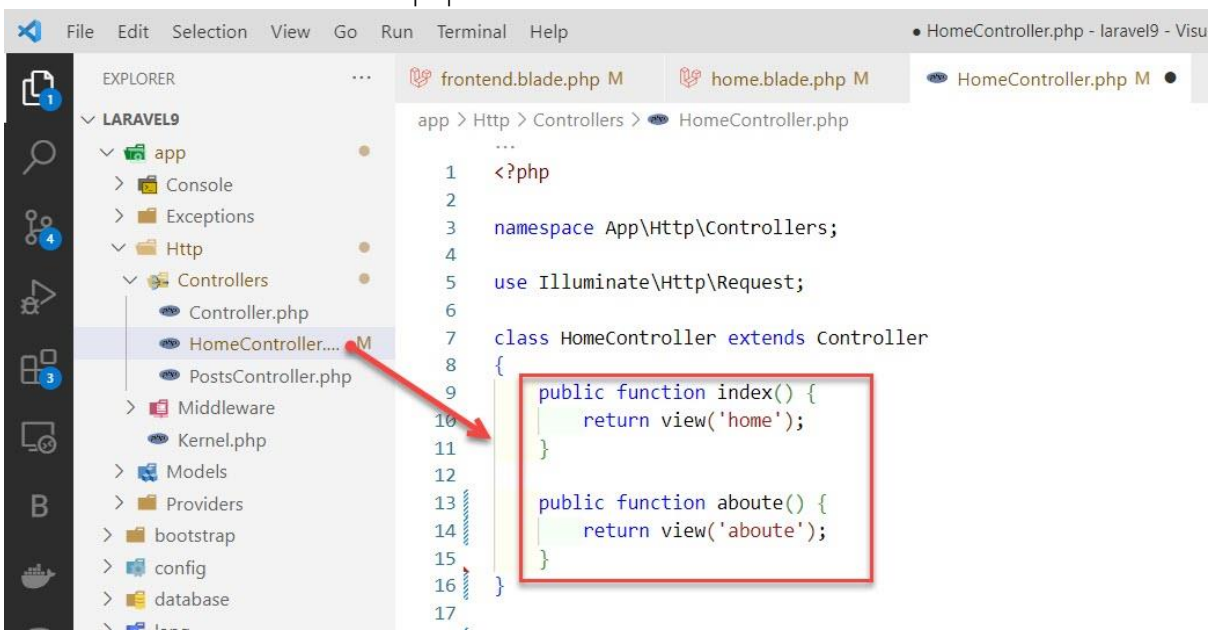
การใช้งาน Template หลัก

การใช้งาน template หลัก ที่เราได้ออกแบบและสร้างขึ้นนั้น เราจะต้องสร้างไฟล์ขึ้นมาอีกไฟล์เพื่อเรียกใช้งานตัว template หลัก โดยจะเรียกใช้งานด้วยคำสั่ง @extends('ชื่อ template หลักที่ต้องการใช้') จะเขียนไว้ที่ตำแหน่งด้านบนสุดของไฟล์นั้น ๆ ส่วนของการเรียกใช้เนื้อหาที่จะแสดงจะใช้คำสั่ง @section('ชื่อที่ตั้งขึ้นจากไฟล์ template ที่เรากำหนด @yield ไว้') และปิดท้ายด้วยคำสั่ง @endsection

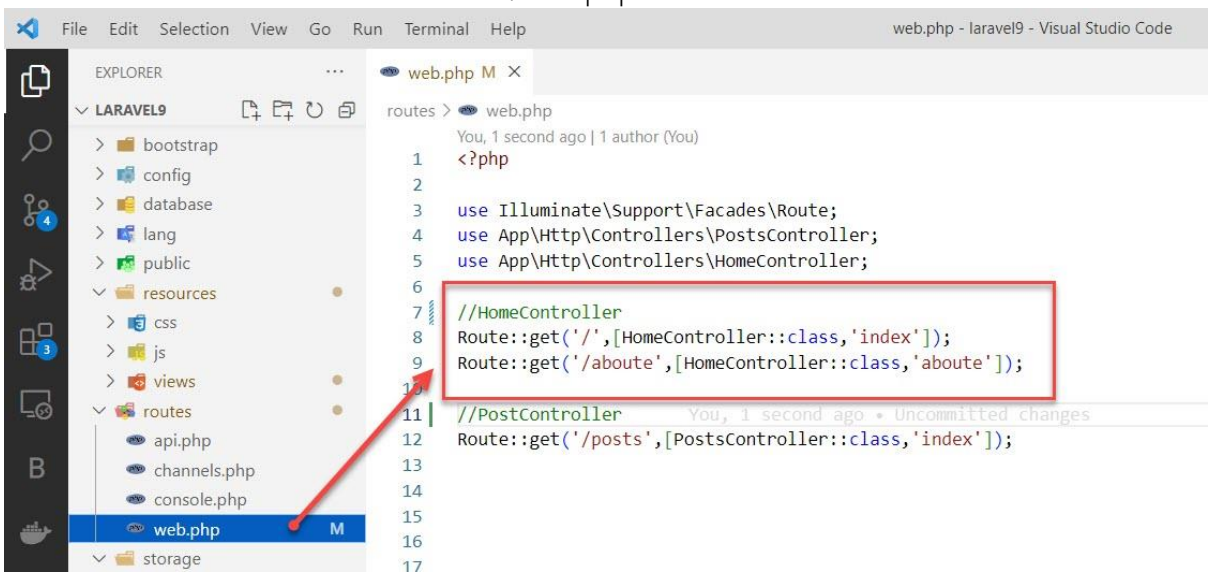
ตัวอย่างการสร้าง ให้สร้างไฟล์ขึ้นมาอีกไฟล์ ชื่อ home.blade.php ในโฟลเดอร์ resource/views แล้วเขียนโค้ด ดังนี้



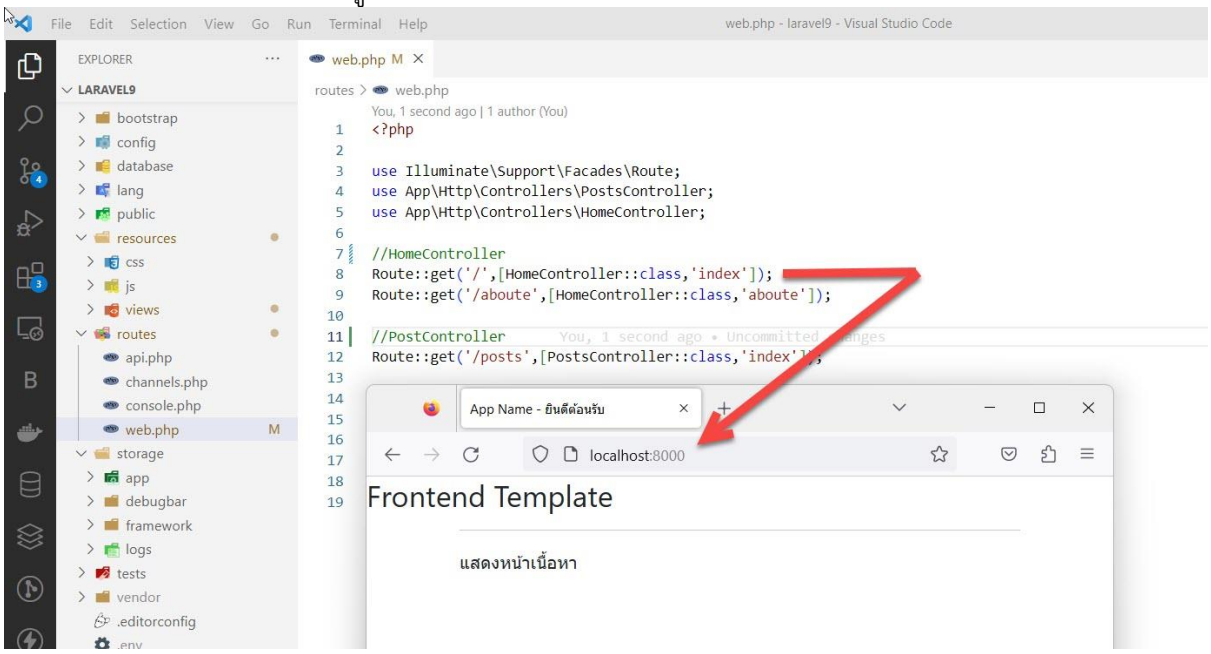
เปิดไฟล์ HomeController.php แล้วแก้ไขโค้ดเพื่อเชื่อมต่อกับ views ที่ต้องการแสดงผล ตามนี้



ซึ่งจากหน้า 65 เราได้สร้างไฟล์ controllers ชื่อ HomeController.php แล้วก่อนหน้านั้น ต่อจากนั้นให้แก้ไขและปรับแต่งโค้ดใน routes/web.php ตามนี้



บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/ ที่ Browser



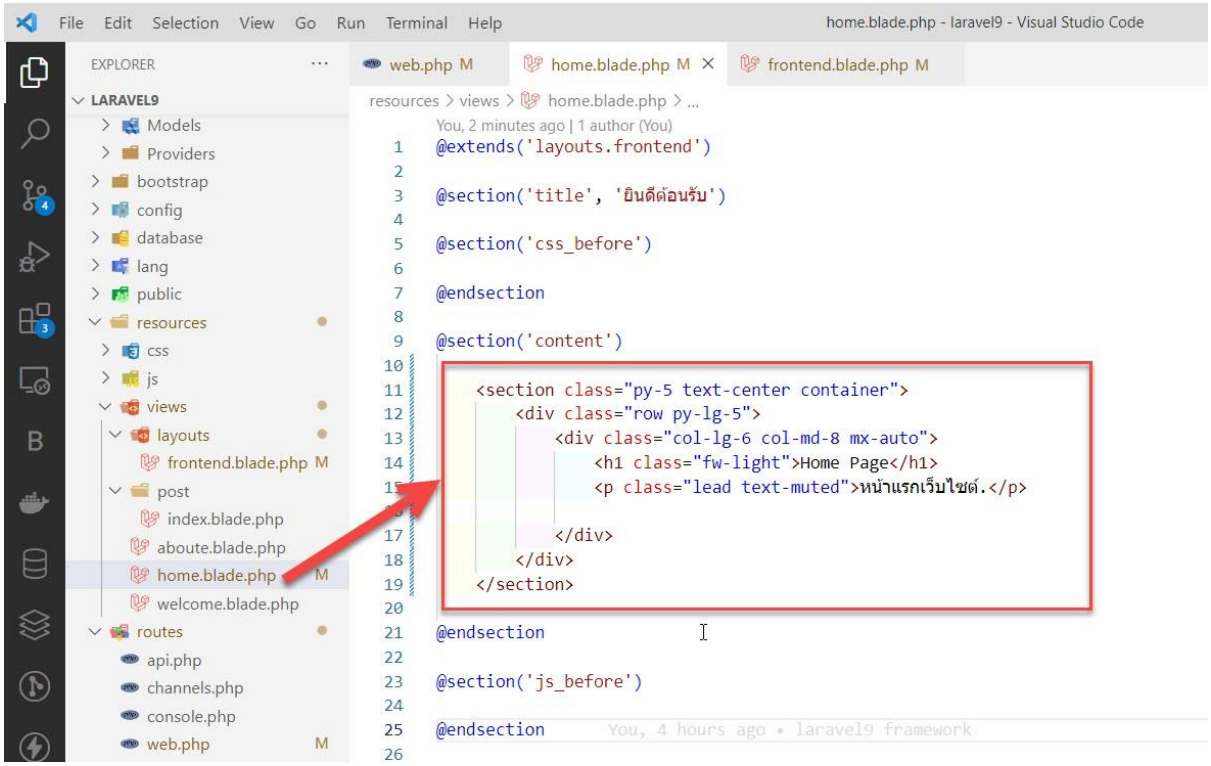
อธิบายเพิ่มเติม จากหน้าที่ 90 ได้ทำการสร้างไฟล์ template หลักชื่อว่า frontend.blade.php เก็บไว้ที่โฟลเดอร์ views/layouts/... แล้วสร้างไฟล์ home.blade.php ในไฟล์ home.blade.php ได้มีการเรียกใช้ template หลัก โดยใช้คำสั่ง `@extends('layouts.frontend')` เพื่อกำหนดว่าจะใช้ template ไหน แล้วใช้คำสั่ง `@section('content')` เพื่อแสดงเนื้อหาแต่ละหน้าตามที่เราต้องการ จากนั้นก็ได้ทำการแก้ไขโค้ดในไฟล์ HomeController.php เพื่อให้ชี้ไปที่ views สำหรับหน้าที่ต้องการ สุดท้ายก็ไปแก้ไขโค้ดในส่วนของ routes/web.php เพื่อกำหนดเส้นทางการเข้าถึงแต่ละหน้า

จะเห็นว่าหน้า template ของเรายังดูไม่สวยเท่าไหร่นัก ทีนี้เรามาปรับ template ให้ดูดีมีสีสันเพิ่มกันดูต่อเลยครับ หนังสือนี้จะใช้ CSS Bootstrap ในการช่วยปรับหน้าตาเว็บของเราให้ดูดีขึ้นนะครับ

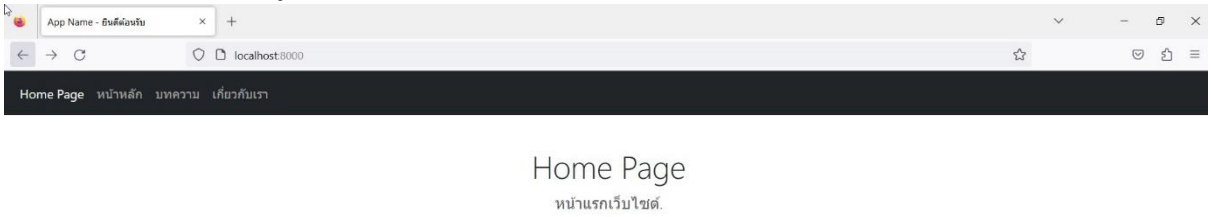
ให้เพิ่มโค้ดที่ไฟล์ frontend.blade.php ตามนี้

```
<!-- resources/views/layouts/frontend.blade.php -->
<!doctype html>
<html lang="{{ config('app.locale') }}">
<!-- CSRF Token -->
<meta name="csrf-token" content="{{ csrf_token() }}">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>App Name - @yield('title')</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
rel="stylesheet">
    @yield('css_before')
</head>
<body>
    <nav class="navbar navbar-expand-lg fixed-top navbar-dark bg-dark" aria-label="Main navigation">
        <div class="container-fluid">
            <div class="navbar-collapse offcanvas-collapse" id="navbarsExampleDefault">
                <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                    <li class="nav-item">
                        <a class="nav-link active" aria-current="page" href="{{ url('/') }}Home Page</a>

                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url('/') }}">หน้าหลัก</a>

                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url('/posts') }}">บทความ</a>

                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url('/aboute') }}">เกี่ยวกับเรา</a>

                    </li>
                </ul>
            </div>
        </div>
    </nav>
    <div class="container">

        @yield('content')
    </div>
</body>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"></script>
@yield('js_before')
</html>
```

ปรับแต่งหน้า home.blade.php ใหม่ โดยเพิ่มโค้ด html และข้อความ เข้าไปตามนี้



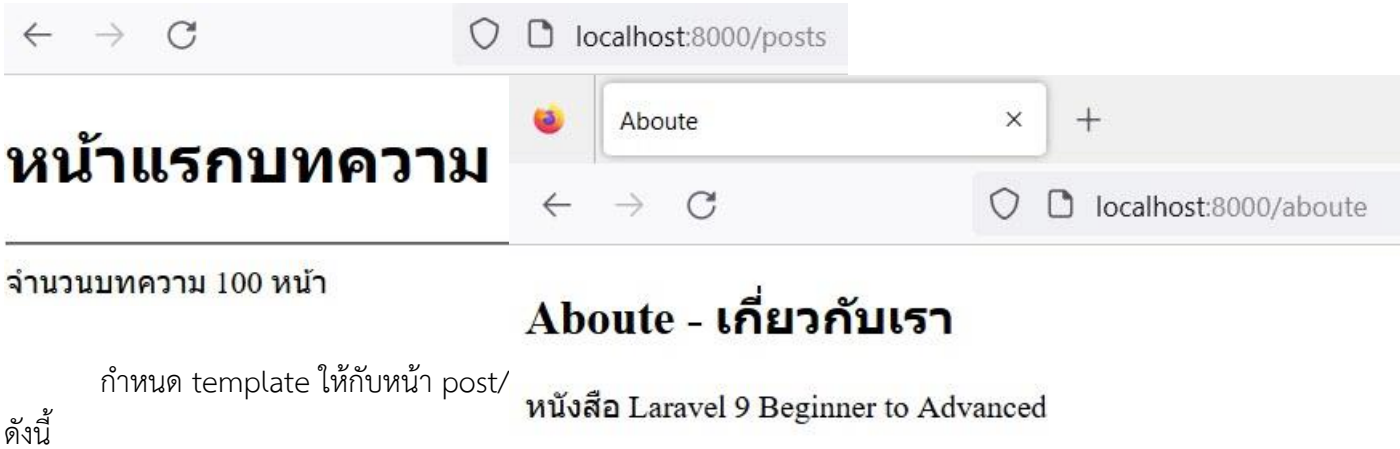
บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/ ที่ Browser



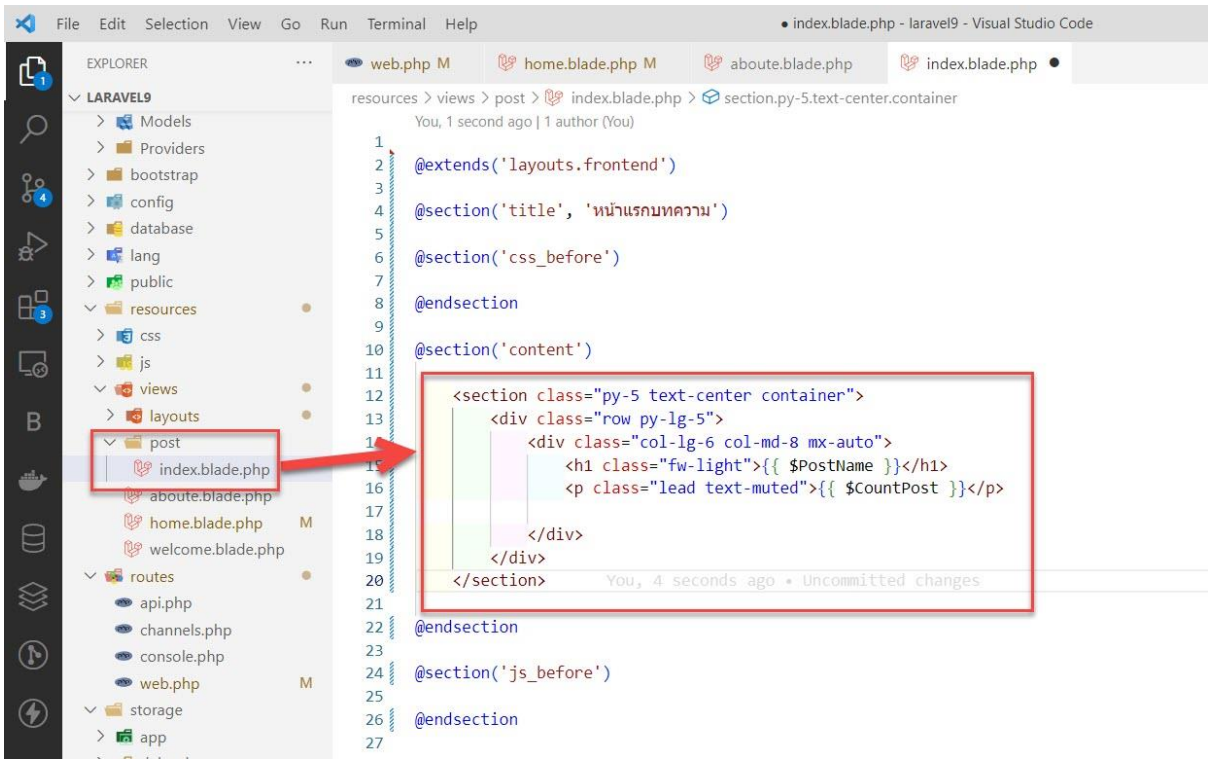
Home Page

หน้าแรกเว็บไซต์.

เพียงเท่านี้เราก็จะได้ template เว็บไซต์ที่เราสร้างขึ้นเพื่อใช้สำหรับงานของเราเรียบร้อยแล้ว ส่วนที่เป็นหน้าเว็บอื่น ๆ ที่เราได้สร้างไว้ก่อนหน้านั้น เช่น post/index.blade.php และ aboute.blade.php ถ้าเราอยากกำหนดให้ template ที่เราเพิ่งสร้างขึ้นก็สามารถทำได้ เมื่อเราคลิกที่ เมนูบทความ และ เมนูเกี่ยวกับเรา ก็จะเขียนว่ายังไม่มีการเรียกใช้งาน template ใด ๆ เลย



กำหนด template ให้กับหน้า post/ ดังนี้



บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/ ที่ Browser แล้วคลิกที่ **เมนูบทความ**



เพียงเท่านี้ก็จะได้หน้าเว็บไซต์ตาม template ที่เรากำหนดไว้ได้แล้ว ส่วน เมนูเกี่ยวกับเรา ก็ทำตามแบบเดียวกันเลย ลองฝึกเขียนดูนะครับ

จะเห็นได้ว่าเมื่อเรานำ blade template มาใช้งาน เราจะได้ไฟล์ต่าง ๆ ซึ่งแยกส่วนกัน ระหว่างหน้าที่ใช้สำหรับการจัดการ template ซึ่งสามารถสร้าง Master Layouts ที่เป็นส่วนหลักของระบบ แบบแยกแต่ละหน้าจอได้อย่างอิสระมากขึ้น ทำให้ง่ายและสะดวกต่อการพัฒนาเว็บไซต์ ไม่ต้องเขียนโค้ดคำสั่งที่ ซ้ำๆ กัน

Control Structures

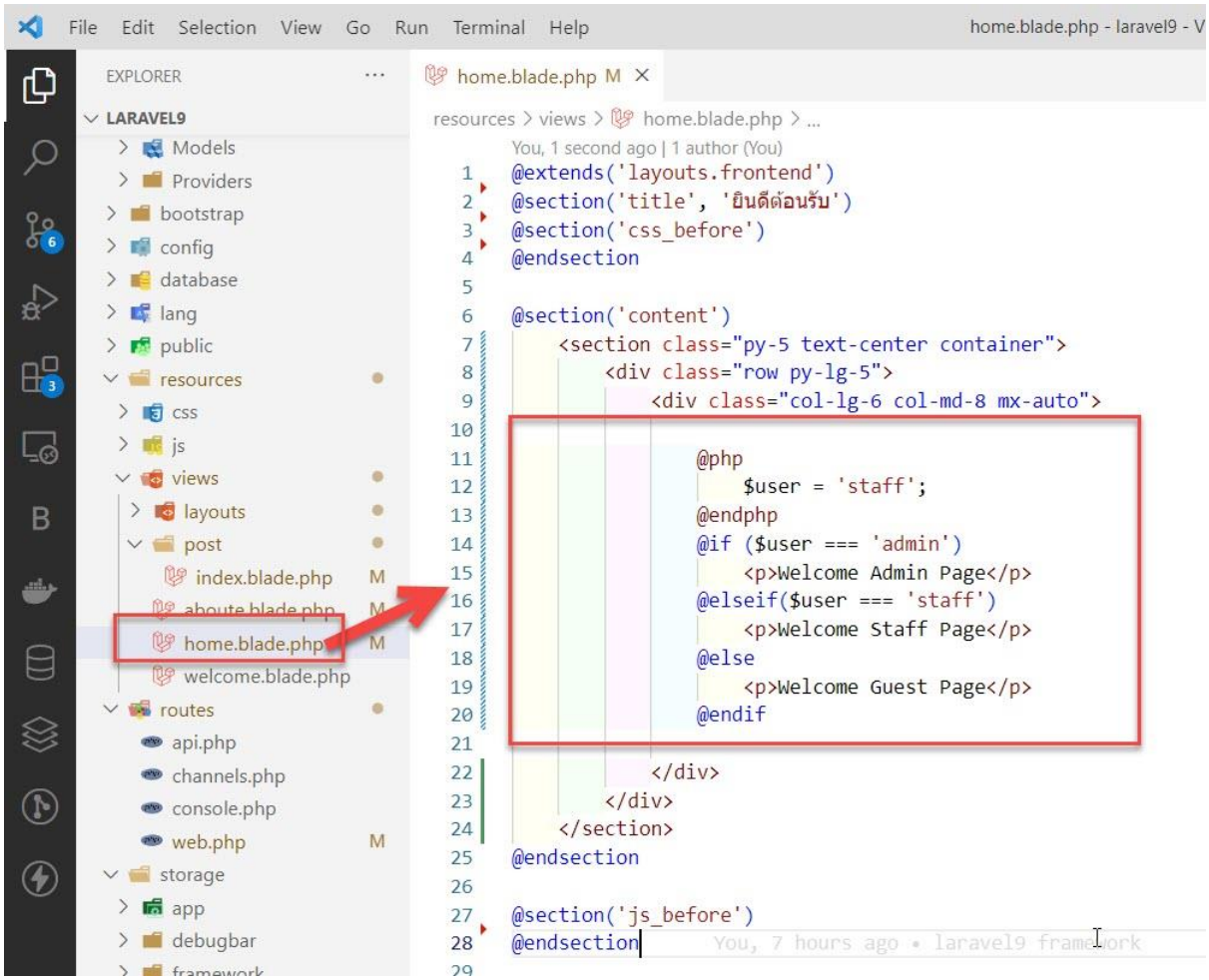
ในการใช้งาน blade template หากเราต้องการที่จะใช้งาน control structures เช่น คำสั่ง if , else , switch , loop รูปแบบต่าง ๆ เช่น for , foreach , while ต่าง ๆ เหล่านี้เป็นคำสั่งสำหรับการตรวจสอบเงื่อนไขการแสดงผลของข้อมูล หรือ แสดงข้อมูลต่าง ๆ ที่อยู่ในรูปแบบออก Array โดย blade จะมีรูปแบบคำสั่งมาให้ เช่น ถ้าต้องการตรวจสอบเงื่อนไขข้อมูลโดยใช้ if ปกติถ้าเราเขียน PHP เราจะเขียนแบบนี้

```php
<?php
   $number = 18;
   if ($number > 10) {
     echo 'Yes';
   }
```

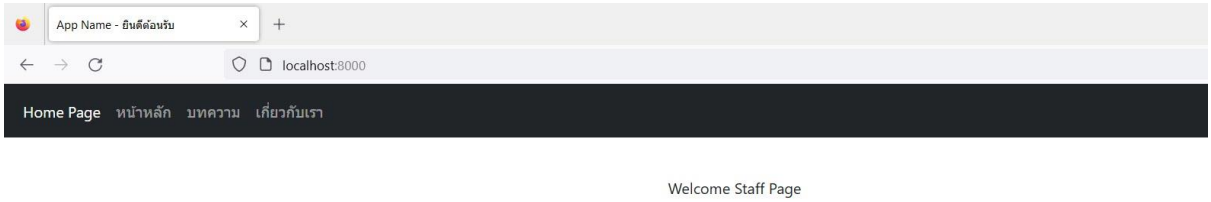
, @while การที่จะใช้คำสั่งต่าง ๆ นี้ได้ ก็ต้องสร้างไฟล์ views ให้เป็น นามสกุล .blade.php ด้วยนะ

EX1.การเขียนคำสั่ง @if สำหรับตรวจสอบเงื่อนไข

เปิดไฟล์ resources/views/home.blade.php แล้วพิมพ์โค้ด ดังนี้

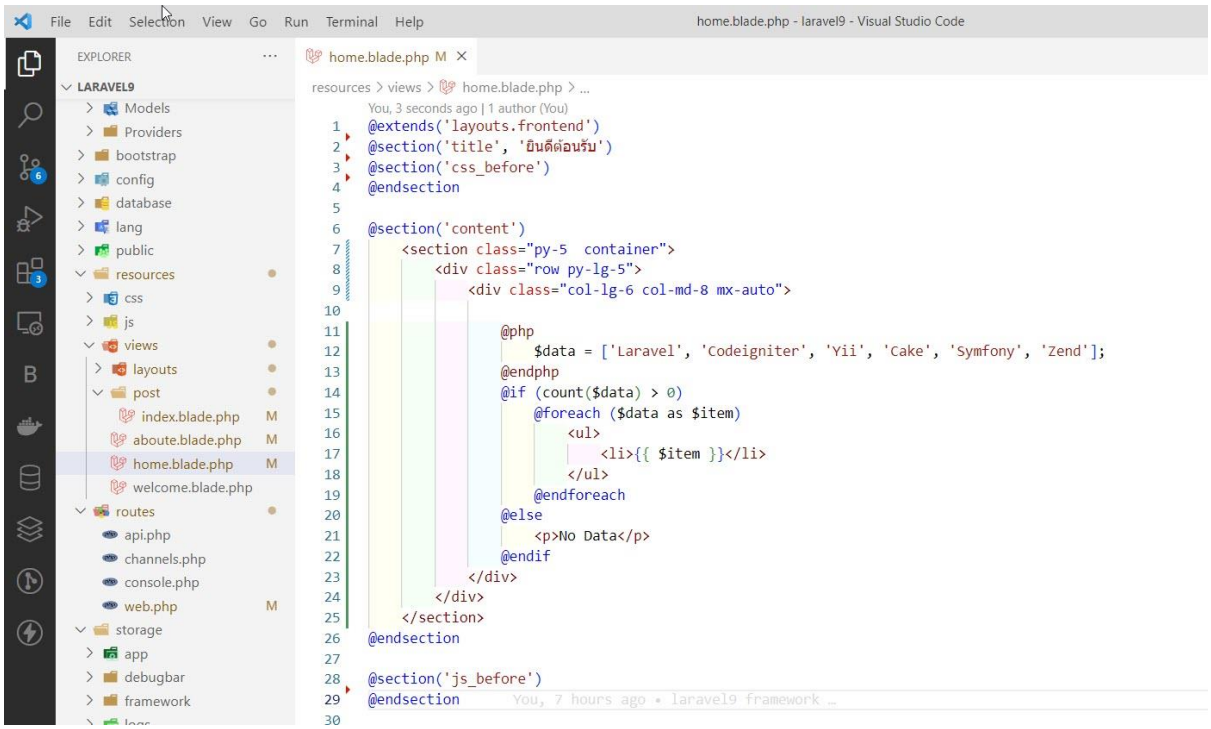


บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/ ที่ Browser

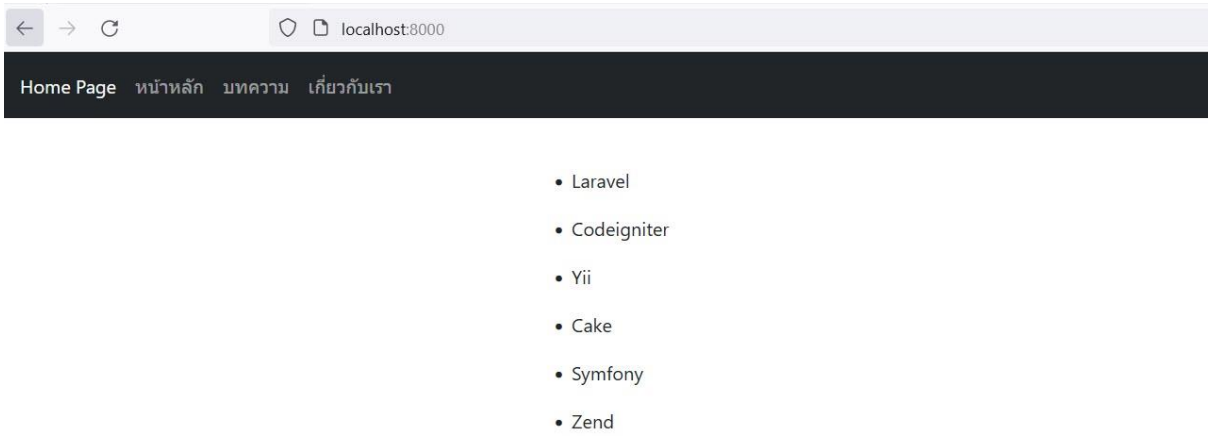


จากตัวอย่างโค้ดที่เขียนเป็นการแสดงให้เห็นถึงการใช้คำสั่ง if โดยเริ่มต้นตั้งแต่บรรทัดที่ 12 ทำการสร้างตัวแปร $user แล้วกำหนดค่าเป็น staff จากนั้นก็ใช้คำสั่ง @if เพื่อตรวจสอบเงื่อนไขการทำงาน ซึ่งจะทำการตรวจสอบผู้ใช้งานในแต่ละสิทธิ์

EX2.การเขียนคำสั่ง @foreach สำหรับแสดงชุดข้อมูล Array

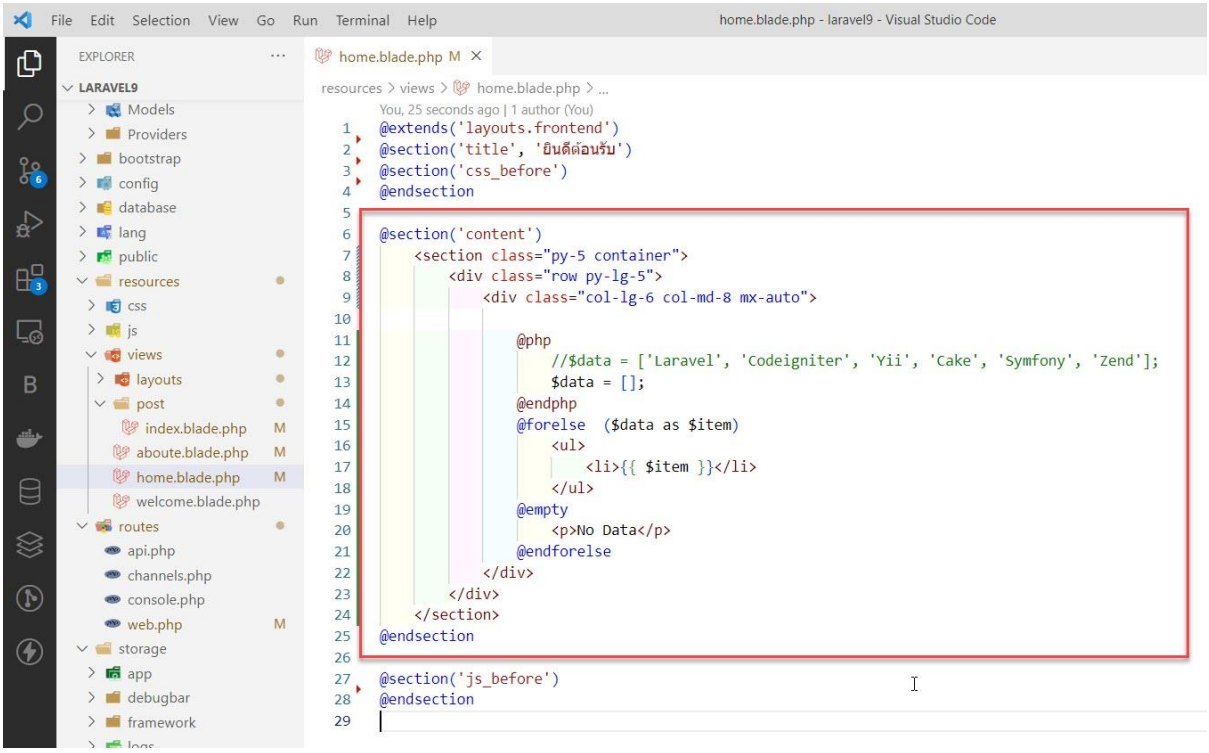


บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/ ที่ Browser



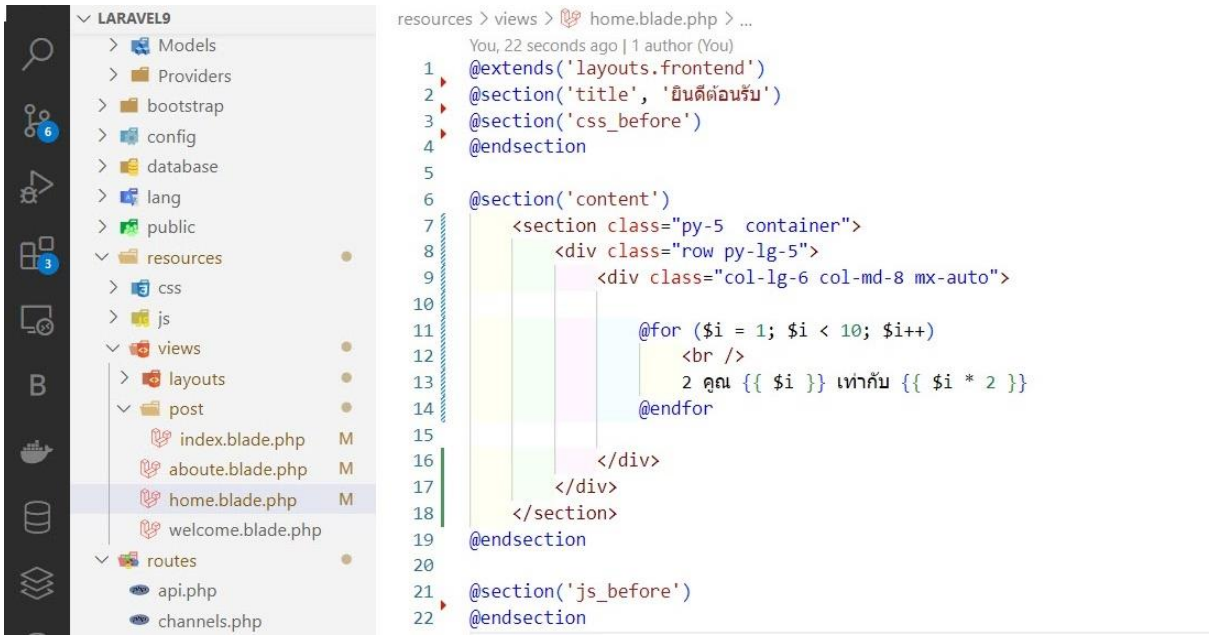
จากโค้ดที่เขียนหน้าที่ผ่านนั้น เป็นตัวอย่างเบื้องต้นเพื่อแสดงให้เห็นวิธีการใช้งานคำสั่ง foreach เพื่อแสดงข้อมูลที่อยู่ในรูปแบบของ Array

EX3.การเขียนคำสั่ง @forelse สำหรับแสดงชุดข้อมูล Array

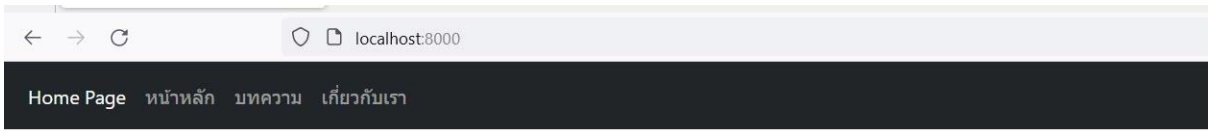


คำสั่ง @forelse จะเป็นคำสั่งสำหรับ loop ข้อมูลที่เป็นชุด Array มาแสดงเหมือนกันกับคำสั่ง @foreach เลยครับ จะเลือกใช้แบบไหน ก็ได้ทั้งสองแบบเลยครับ

EX4.การเขียนคำสั่ง @for

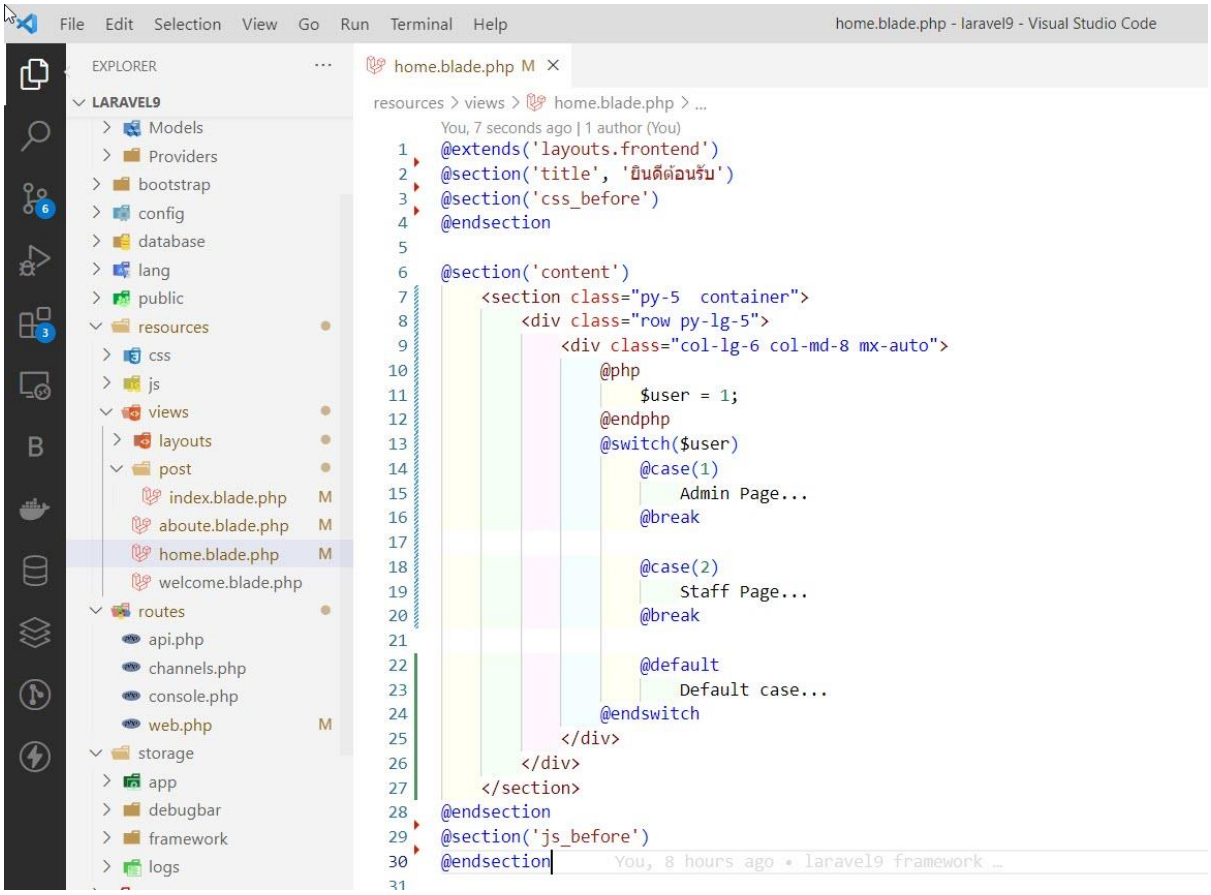


บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/ ที่ Browser



2 คูณ 1 เท่ากับ 2
2 คูณ 2 เท่ากับ 4
2 คูณ 3 เท่ากับ 6
2 คูณ 4 เท่ากับ 8
2 คูณ 5 เท่ากับ 10
2 คูณ 6 เท่ากับ 12
2 คูณ 7 เท่ากับ 14
2 คูณ 8 เท่ากับ 16
2 คูณ 9 เท่ากับ 18

EX5.การเขียนคำสั่ง @switch



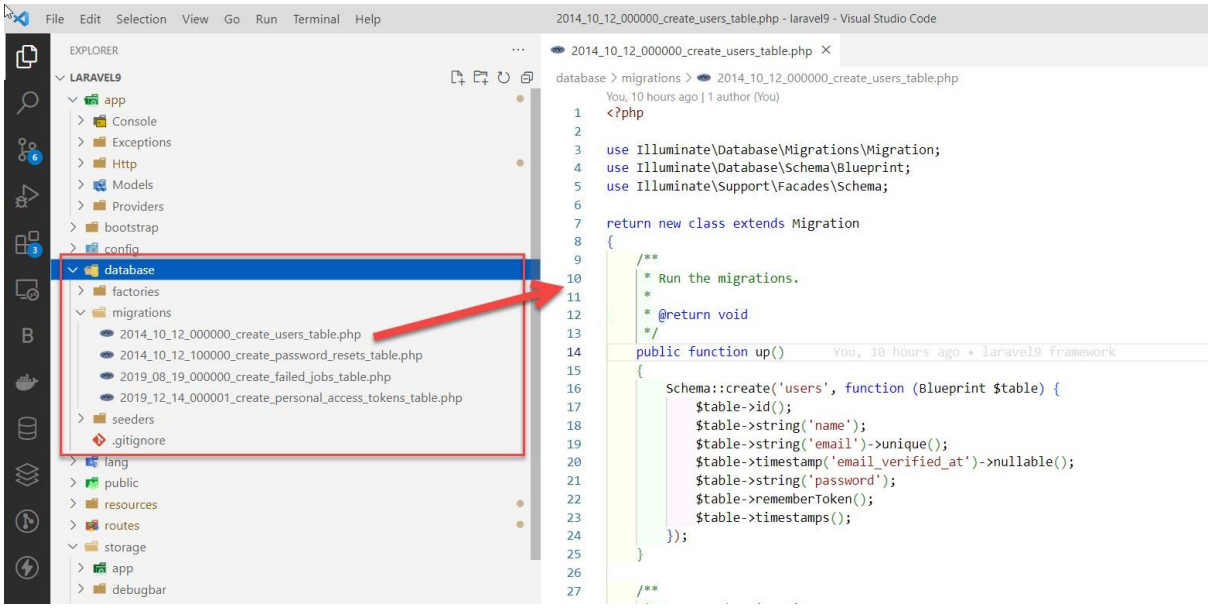
Database Migration And Seeding with Artisan

Laravel Framework จะมีตัวช่วยที่ใช้สำหรับการสร้างตาราง (tables) และการกำหนดคุณสมบัติหรือโครงสร้างให้กับตารางที่เราจะสร้างได้ง่าย ๆ นั่นคือใช้คำสั่ง artisan command ที่มีมากับ Laravel ช่วยในการสร้างตาราง นอกจากจะใช้ artisan command ในการสร้างตารางแล้วยังสามารถสร้างข้อมูลเริ่มต้น (seeding) ของตารางต่าง ๆ ได้ ตัวอย่างคำสั่งพื้นฐานสำหรับการทำ Database Migration

php artisan make:migration [ชื่อคลาสหรือชื่อตารางที่จะสร้าง]

php artisan make:seeder [ชื่อคลาสหรือชื่อตารางที่ต้องการจะทำ **seeding**]

โดยเมื่อรันคำสั่งด้านบนนี้แล้วไฟล์จะถูกสร้างขึ้นและเก็บไว้ที่ โฟลเดอร์ database/migration หรือ database/seeders



จากภาพตัวอย่างจะเห็นว่า Laravel ได้เตรียมตารางบางส่วน ไว้ให้เราแล้ว เก็บไว้ในโฟลเดอร์ database/migrations ซึ่งประกอบได้ด้วย ตาราง users , ตาราง password_resets , ตาราง failed_jobs และ personal_access_tokens

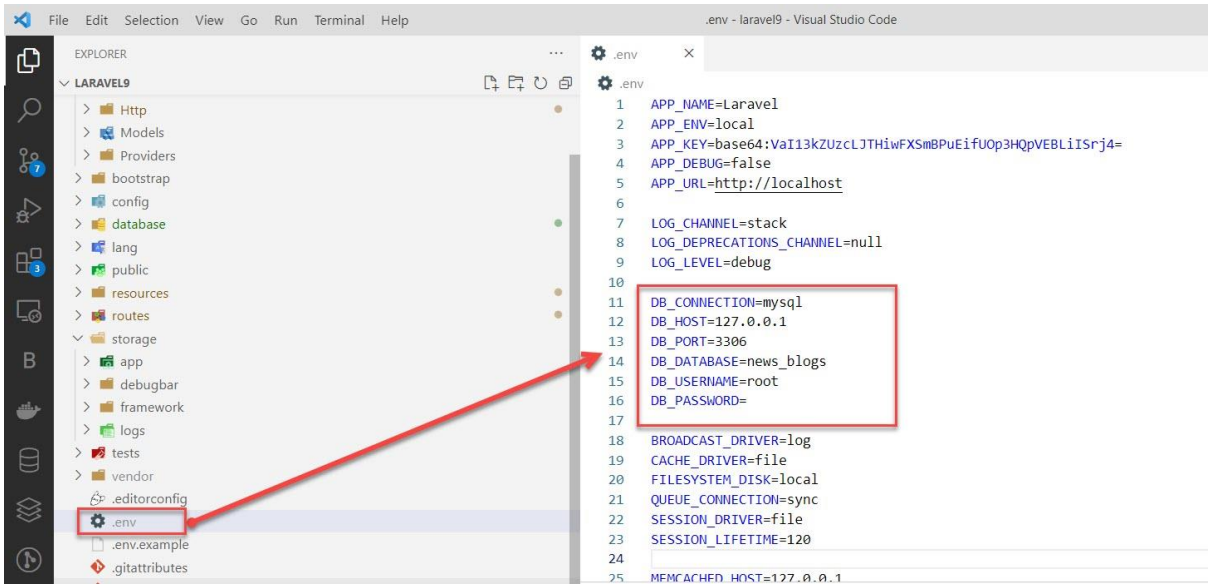
php artisan migrate:install	ใช้สำหรับสร้างตาราง migration ในฐานข้อมูล
php artisan migrate	ใช้สำหรับรัน migration
php artisan migrate:refresh	ใช้สำหรับ rollback ตารางทั้งหมดและสั่งให้รัน migration ใหม่
php artisan migrate:rollback	ใช้สำหรับ Undo การทำงานก่อนหน้า
php artisan migrate:fresh	ใช้สำหรับลบตารางทั้งหมดและสั่ง migrate ใหม่อีกครั้ง
php artisan migrate:status	ใช้สำหรับแสดงสถานะของ Migrations

สร้างฐานข้อมูลและตั้งค่าการเชื่อมต่อฐานข้อมูล

ก่อนที่เราจะใช้คำสั่ง php artisan migrate:install ซึ่งเป็นคำสั่งที่ใช้สำหรับการสร้างตารางลงในฐานข้อมูล นั้น เราต้องตั้งค่าฐานข้อมูลให้เรียบร้อยก่อน มิเช่นนั้น Laravel จะไม่สามารถเชื่อมต่อกับฐานข้อมูลได้ ในหนังสือเล่มนี้จะเป็นการใช้ฐานข้อมูล MySQL/MariaDB โดยใช้ XAMPP เป็น Web Server

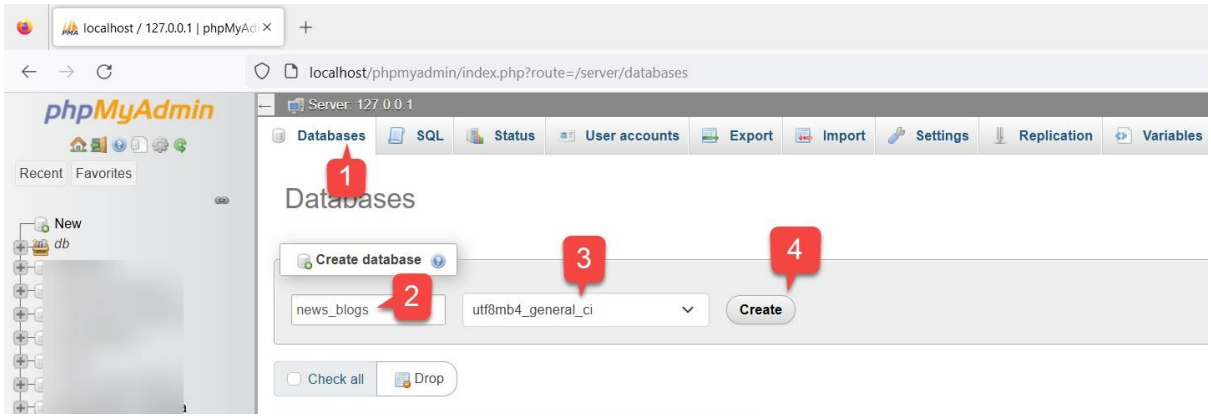
สามารถกำหนดค่าได้ที่ไฟล์ .env

1.เปิดไฟล์ .env แล้วกรอกข้อมูลสำหรับการเชื่อมต่อกับฐานข้อมูล MySQL ดังนี้



ให้กรอกรายละเอียด DB_DATABASE , DB_USERNAME , DB_PASSWORD ให้ถูกต้อง ถ้าใช้ XAMPP อยู่แล้วในส่วนของ DB_PASSWORD จะไม่ได้กำหนดรหัสผ่านมาให้จึงว่างไว้ แล้วบันทึกไฟล์

2.เปิด Browser ขึ้นมาแล้วพิมพ์ URL http://localhost/phpmyadmin เพื่อจัดการฐานข้อมูล แล้วสร้างฐานข้อมูลชื่อ news_blogs หรือตามที่เรากำหนดในไฟล์ .env ในส่วนของ DB_DATABASE กำหนดรายละเอียดตามภาพนี้ แล้วกดปุ่ม Create เพื่อสร้าง



ทีนี้ลองมาสร้างตารางเพิ่มสัก 2 ตาราง ดูครับ โดยจะสร้างตาราง posts (สำหรับเก็บข้อมูลบทความ) และตาราง categories (สำหรับเก็บข้อมูลประเภทข่าว) เราจะได้ตารางเพิ่มเข้ามาอีก 2 ตาราง ซึ่งรวมกับ

ตารางเก่าที่ Laravel สร้างมาให้เราเรียบร้อยแล้ว อีก 4 ตาราง คือ ตาราง users , ตาราง password_resets , ตาราง failed_jobs และ personal_access_tokens จะได้เป็น 5 ตาราง

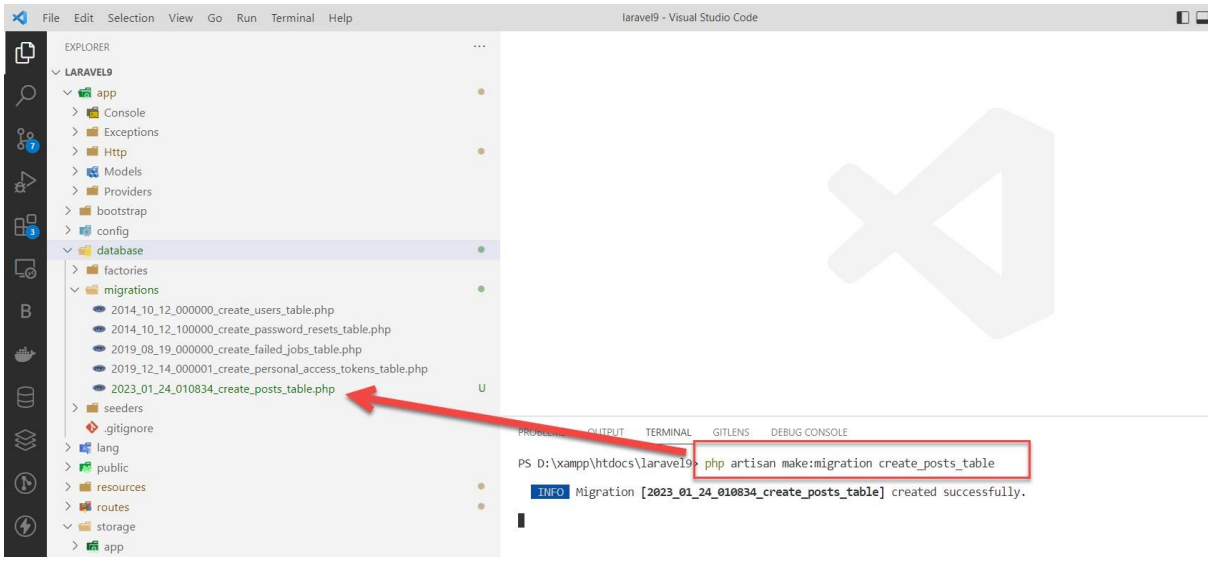
เราสามารถสร้างเพิ่มได้ โดยจะสร้างตารางใหม่ชื่อ posts และ categories โดยพิมพ์คำสั่งลงใน Terminal ที่ VSCode ทีละคำสั่ง ดังนี้

php artisan make:migration create_posts_table	คำสั่งสำหรับสร้างตาราง **posts**
php artisan make:migration create_categories_table	คำสั่งสำหรับสร้างตาราง **categories**

* ถ้ามีการรันคำสั่งค้างอยู่ให้กดปุ่ม ctrl + c เพื่อยกเลิกการรันก่อน

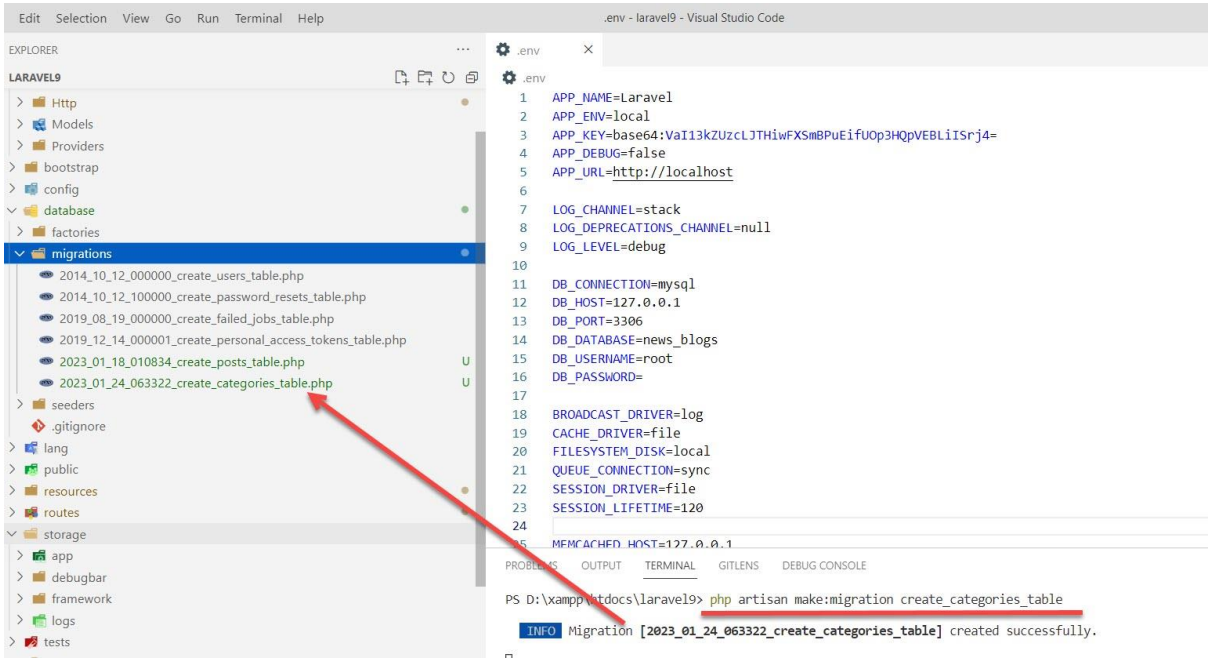


พิมพ์คำสั่งแล้วกด Enter ไฟล์จะถูกสร้างขึ้นมาให้เราใหม่ ดังนี้



Migrations ของสำหรับ posts จะถูกสร้างขึ้นมา

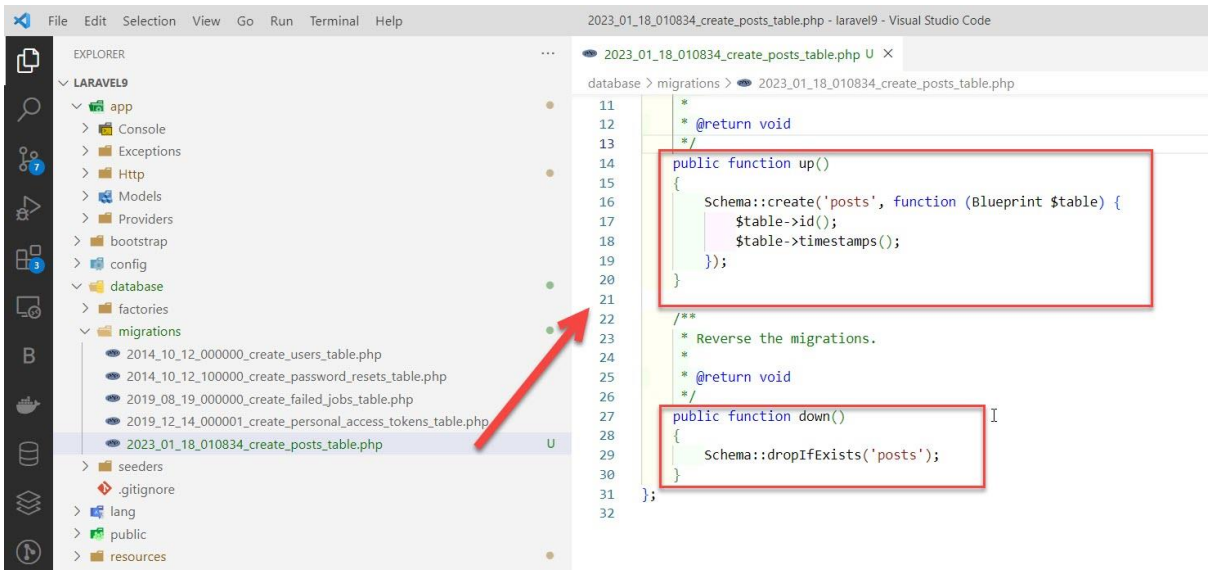
จากนั้นสร้างตาราง categories ต่อเลย



Migrations ของสำหรับ categories จะถูกสร้างขึ้นมา

ตัวเลขด้านหน้าจะเป็นวันที่และเวลาที่สร้างในขณะนั้น ไม่มีผลกระทบอะไร ซึ่งแต่ละครั้งที่สร้างจะไม่เหมือนกัน

จากนั้นเปิดไฟล์ create_posts_table.php ขึ้นมาเพื่อกำหนดฟิลด์



ก็จะมีเมธอดที่ถูกสร้างขึ้นมาเป็นค่าเริ่มต้นให้ จะประกอบด้วย 2 เมธอด คือ up() และ down()

เมธอด up() ใช้สำหรับการสร้างตารางพร้อมกับระบุฟิลด์ที่ต้องการสร้างซึ่งสามารถระบุรูปแบบการจัดเก็บข้อมูลได้ตามรูปแบบที่ Laravel มีมาให้ เมธอด down() ใช้สำหรับลบตาราง ในการสร้างตาราง Laravel จะใช้คลาส Schema ในการจัดการตาราง

ให้เพิ่มโค้ดสำหรับการสร้างในตาราง posts ดังนี้

database > migrations > 2023_01_18_010834_create_posts_table.php

```
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::create('posts', function (Blueprint $table) {
17             $table->id();
18             $table->string('post_title');
19             $table->text('post_detail');
20             $table->string('post_photo')->nullable();
21             $table->integer('visitors')->nullable();
22             $table->timestamps();
23         });
24     }
25
```

จากคำสั่งที่เขียนด้านบนเป็นการกำหนดคุณสมบัติให้กับฟิลด์ต่าง ๆ ซึ่งเราสามารถกำหนดค่าได้ตามรูปแบบที่ Laravel มีมาให้

คำอธิบายโครงสร้างของฟิลด์ ตาราง posts

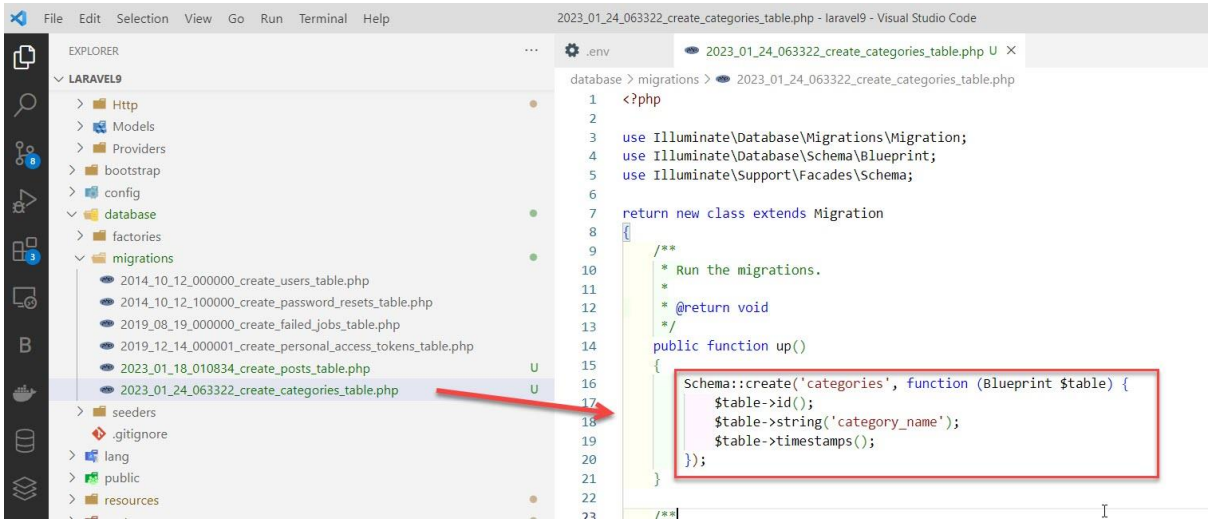
Schema	Description	MySQL Data Type
$table->id()	สร้างฟิลด์ชื่อ id (Primary key And AUTO_INCREMENT)	bigint(20)
$table->string('post_title')	สร้างฟิลด์ชื่อ post_title เก็บหัวข้อบทความ	varchar(255)
$table->text('post_detail')	สร้างฟิลด์ชื่อ post_detail เก็บรายละเอียดบทความ	text
$table->string('post_photo')	สร้างฟิลด์ชื่อ post_photo เก็บภาพประจำบทความ	varchar(255)
$table->integer('visitors')	สร้างฟิลด์ชื่อ visitors เก็บจำนวนคนเปิดผ่านบทความ	int(11)
$table->timestamps()	สร้างฟิลด์ชื่อ created_at และ updated_at เก็บวันที่สร้างและแก้ไข	timestamp

$table->timestamps() เป็นการกำหนด column สำหรับเก็บวันเวลาที่สร้างและวันเวลาปรับปรุงข้อมูล ในที่นี้ Laravel จะกำหนด 2 column ให้เราเลย คือ created_at และ updated_at รูปแบบ timestamp อัตโนมัติ

ดูรายละเอียดเพิ่มเติม https://laravel.com/docs/9.x/migrations หัวข้อ Available Column Types

การใช้งาน Migration นั้น เราควรออกแบบฐานข้อมูลให้เรียบร้อยก่อน เราจะได้ไม่เสียเวลาในการสร้าง หรือถ้ามีฐานข้อมูลอยู่แล้ว เราก็ไม่ต้องสร้าง Migration ก็ได้ สามารถสร้างฟิลด์ต่าง ๆ ผ่าน phpMyAdmin ได้เหมือนกัน แต่แนะนำให้ทำ Migration ไว้จะดีกว่า

เปิดไฟล์ create_categories_table.php ขึ้นมาแล้วเพิ่มฟิลด์ ดังนี้



หลังจากที่เราได้สร้างไฟล์ Migrations และทำการกำหนดรูปแบบของฟิลด์ที่ใช้ในการจัดเก็บข้อมูลในแต่ละตารางตามที่ได้ออกแบบไว้ ในตัวอย่างเป็นการสร้างตาราง categories ประกอบด้วยฟิลด์ id,name,created_at,updated_at

$table->id() คำสั่งสำหรับสร้างฟิลด์ id โดยมีชนิดเป็น BIGINT และและกำหนดเพิ่มอัตโนมัติ (AutoIncrements) และกำหนดเป็นคีย์หลัก (Primary Key) ด้วย

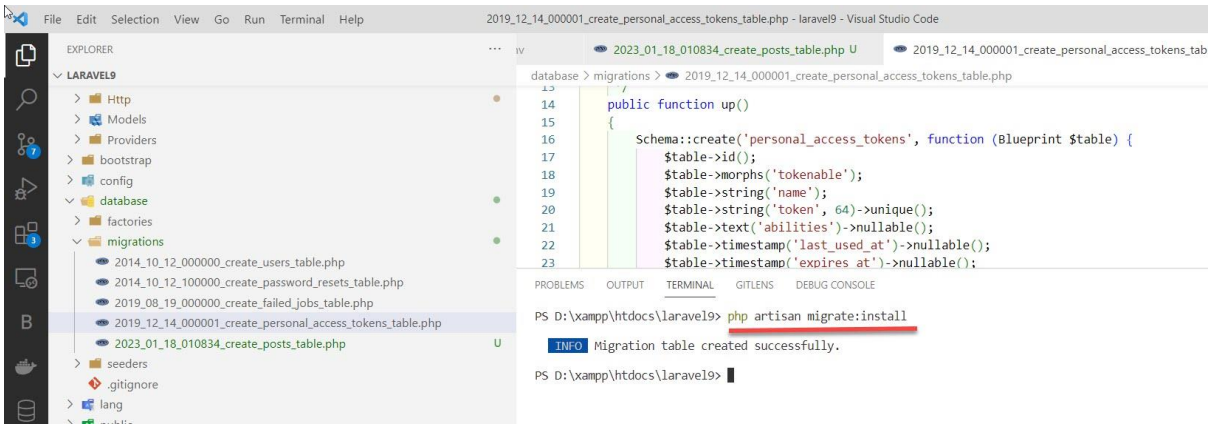
$table->string('name') คำสั่งสำหรับสร้างฟิลด์ name โดยมีชนิดเป็น varchar(255)

$table->timestamps() คำสั่งสำหรับสร้างฟิลด์ created_at และ updated_at สำหรับเก็บวันเวลาที่สร้างและวันเวลาปรับปรุงข้อมูล หากไม่ต้องการใช้ก็สามารถเปิดการใช้งานในโมเดลได้ โดยกำหนด public $timestamps = false ; ในโมเดล

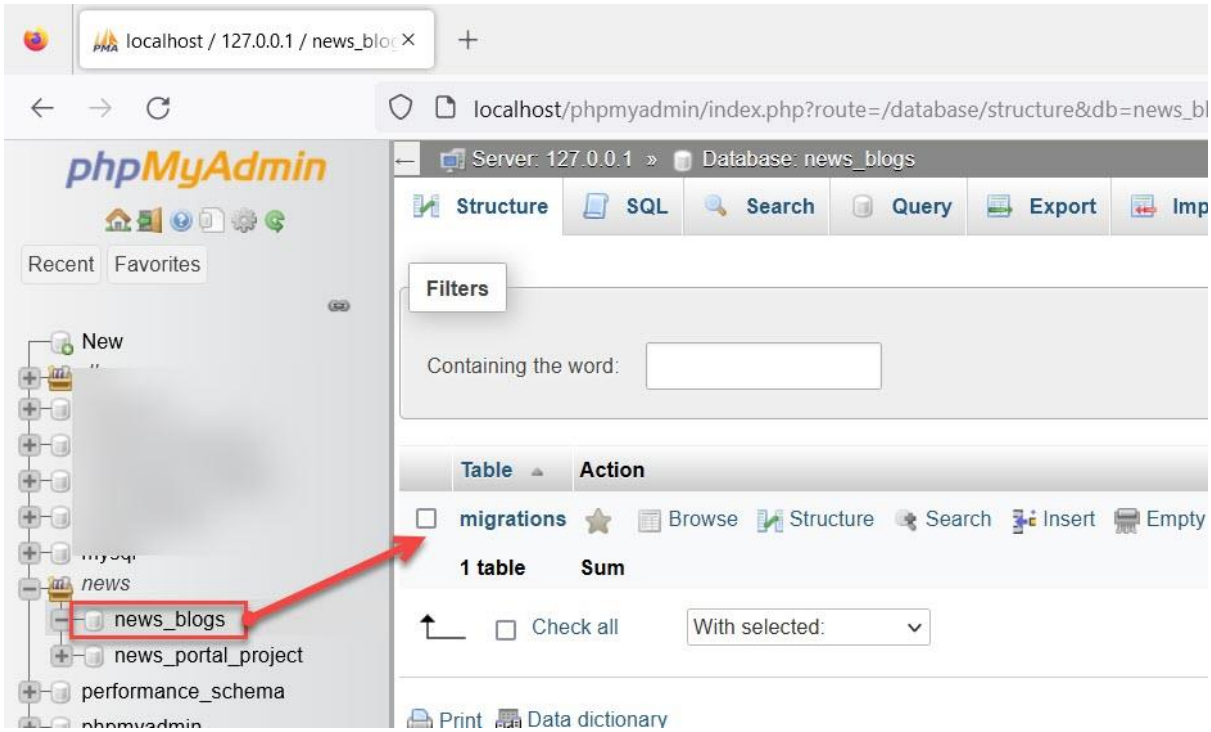
จากนั้นก็ให้ใช้คำสั่งสำหรับสร้างตาราง migration ลงในฐานข้อมูล โดยพิมพ์คำสั่งนี้ใน Terminal

```
php artisan migrate:install
```

เปิด Terminal ใน VSCode ขึ้นมาแล้วพิมพ์คำสั่งและกด Enter ถ้าไม่มีข้อผิดพลาดอะไร จะมีข้อความแสดงขึ้น ว่า Migration table created successfully หมายความว่าสร้างตาราง migration สำเร็จ



เมื่อเปิดดูใน phpMyAdmin จะมีฐานข้อมูลชื่อ news_blogs ถูกสร้างขึ้นมา และมีการสร้างตาราง migration ที่ได้จากการใช้คำสั่ง **php artisan migrate:install** เพิ่มมาให้เรา ตามภาพ

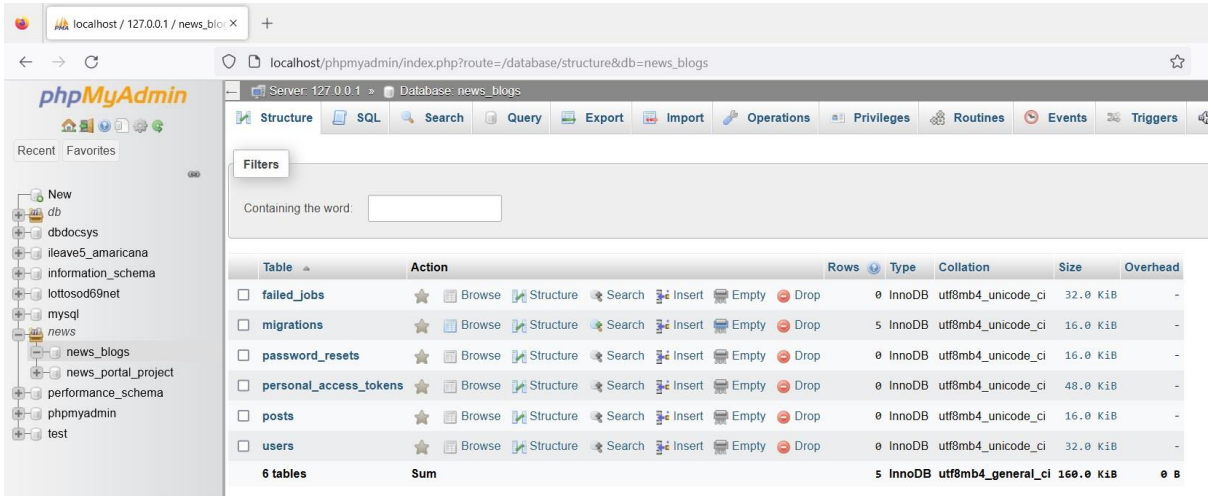


สั่งรันไฟล์ migration ทั้งหมดเพื่อสร้างตารางลงในฐานข้อมูล โดยพิมพ์คำสั่ง

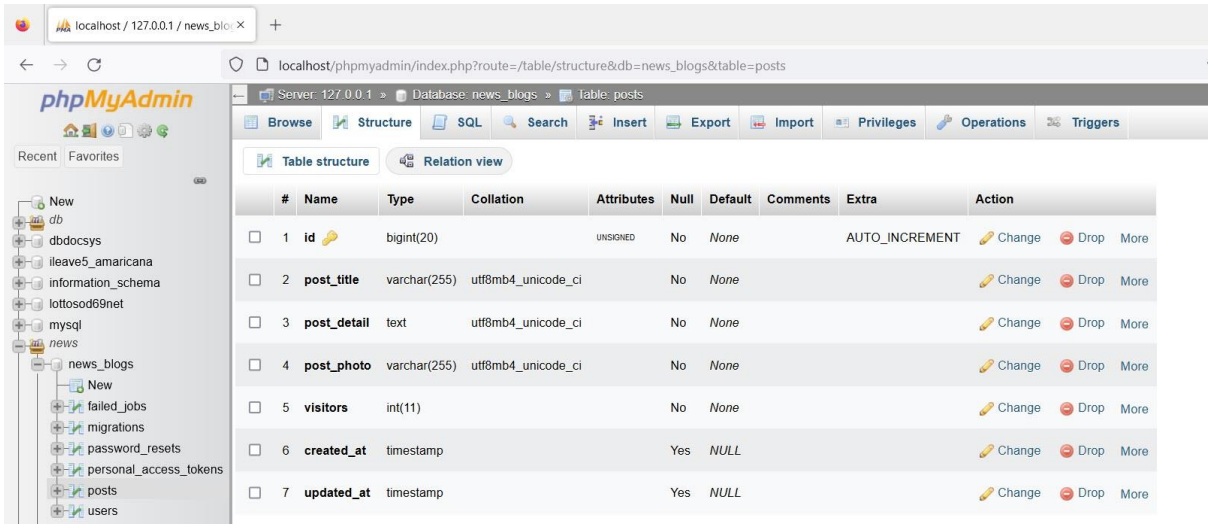
```
php artisan migrate
```



เปิดดูใน phpMyAdmin อีกครั้งเพื่อดูว่าตารางถูกสร้างขึ้นมาจริงไหม



ก็จะเห็นโครงสร้างของตารางที่เราได้ออกแบบไว้



เมื่อเรารันคำสั่ง php artisan migrate แล้ว พบว่าชื่อฟิลด์หรือรูปแบบตารางไม่ถูกต้อง สามารถทำการแก้ไขได้ ด้วยการใช้คำสั่ง php artisan migrate:rollback เพื่อย้อนกลับไปก่อนหน้าที่จะรันคำสั่งนั้นได้ แล้วทำการแก้ไขโครงสร้างของตารางใหม่ให้ถูกต้อง เสร็จแล้วก็ค่อยรันคำสั่ง php artisan migrate อีกครั้ง

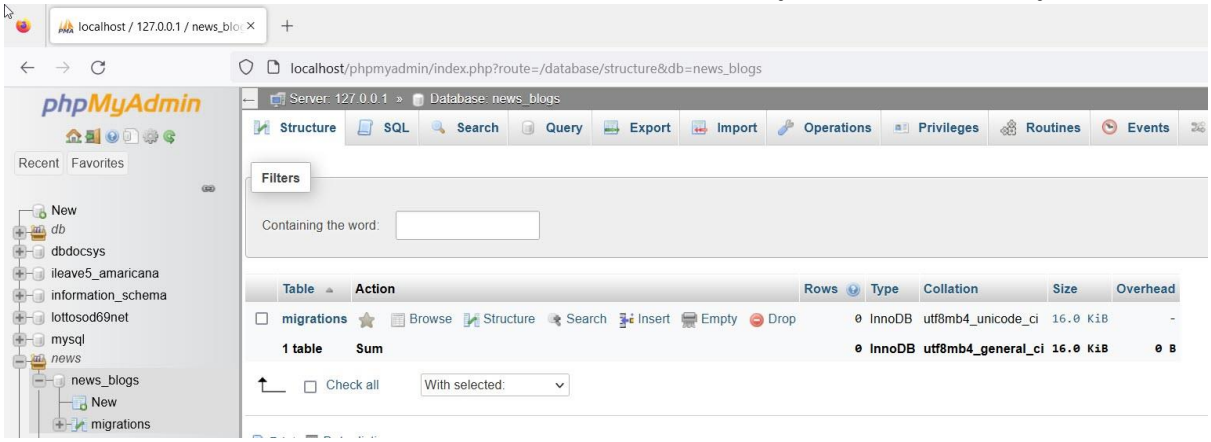
เช่นจากตาราง posts ถ้าต้องการเพิ่มฟิลด์เข้าไปใหม่อีก 1 ฟิลด์ เช่น ต้องการเพิ่ม user_id เพื่อใช้เก็บข้อมูลเจ้าของบทความ ให้เราสร้างไฟล์ Migrations ใหม่ ดังนี้

```
php artisan make:migration add_user_id_column_to_posts_table
```

ใช้คำสั่ง rollback เพื่อลบตารางเก่าออกจากฐานข้อมูลก่อน พิมพ์คำสั่งแล้วกด Enter

```
PS D:\xampp\htdocs\laravel9> php artisan migrate:rollback

  INFO  Rolling back migrations.

  2023_01_18_010834_create_posts_table ..... 14ms DONE
  2019_12_14_000001_create_personal_access_tokens_table ..... 4ms DONE
  2019_08_19_000000_create_failed_jobs_table ..... 5ms DONE
  2014_10_12_100000_create_password_resets_table ..... 5ms DONE
  2014_10_12_000000_create_users_table ..... 4ms DONE
```

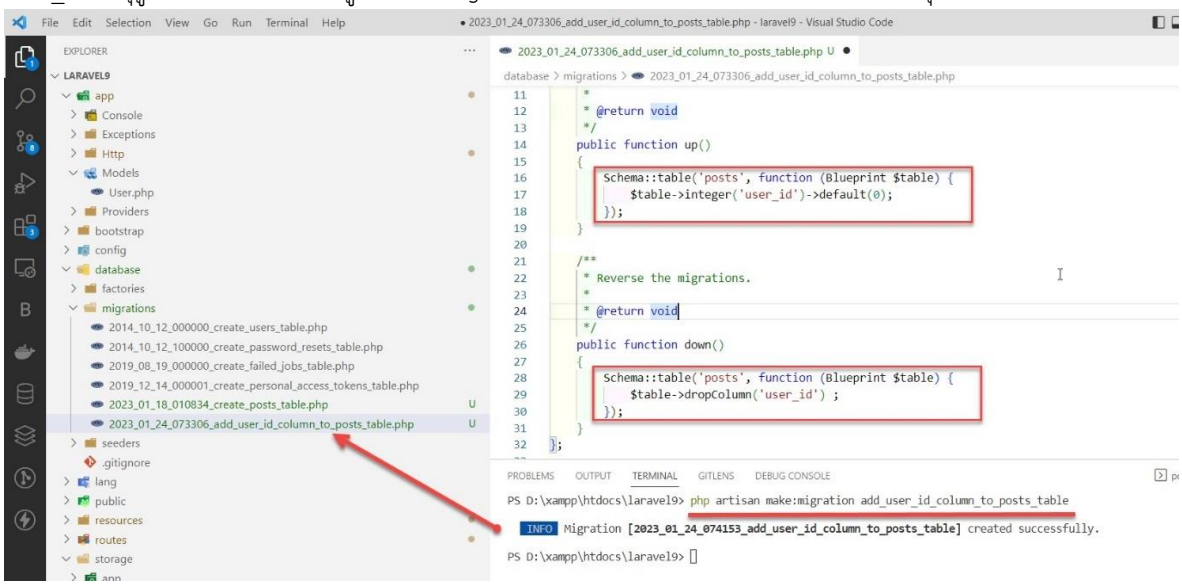
ใน phpMyAdmin ให้กด refresh อีกครั้งก็จะเห็นว่าตารางที่ถูกสร้างขึ้นก่อนหน้านั้นถูกลบไปแล้ว



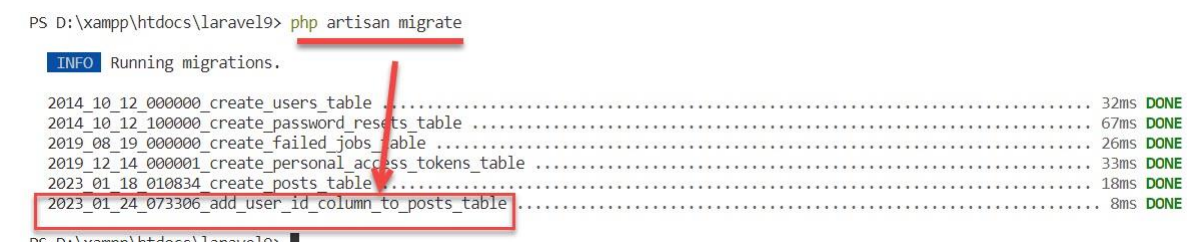
กลับมาที่ Terminal แล้วพิมพ์คำสั่งสำหรับเพิ่มฟิลด์ user_id เข้าไปในตาราง posts ดังนี้



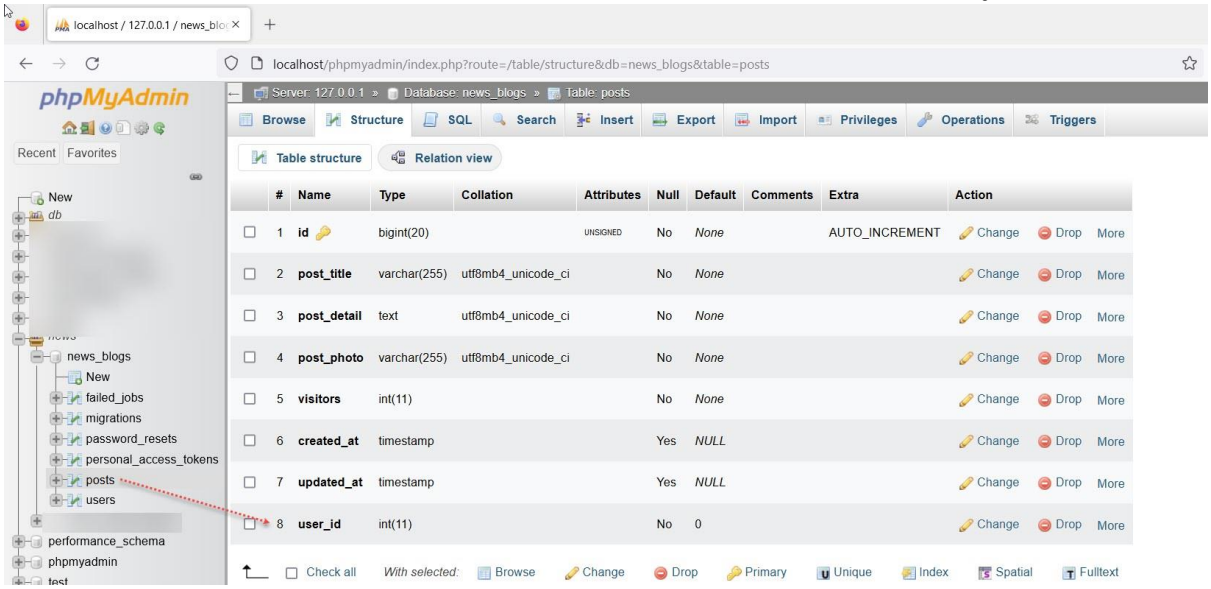
พิมพ์คำสั่งแล้วกด Enter ก็จะได้ไฟล์ migration เพิ่มเข้ามาใหม่ ตัวอย่างตามภาพ จากนั้นเปิดไฟล์ xxx_add_user_id_column_to_posts_table.php แล้วเขียนโค้ดเข้าไป ในส่วนของ เมธอด up() ก็เพิ่มฟิลด์ใหม่ชื่อว่า user_id ระบุรูปแบบการจัดเก็บข้อมูลแบบ integer และกำหนดเริ่มต้นเป็น 0 และ down() ระบุให้ลบฟิลด์ที่สร้างขึ้น



บันทึกไฟล์ แล้วรันคำสั่ง php artisan migrate ใหม่อีกครั้ง



ใน phpMyAdmin ให้กด refresh อีกครั้งก็จะเห็นว่าตารางและฟิลด์ที่แก้ไขใหม่ถูกสร้างขึ้นมาแล้ว



เพิ่มข้อมูลตั้งต้นด้วย Seeding

ในบางครั้งเราต้องการข้อมูลตั้งต้นในการใช้งานบนเว็บไซต์หรือโปรแกรมของเรา เช่น การตั้งค่าระบบพื้นฐาน กำหนดประเภทของบทความ หรือการกำหนดชื่อผู้ใช้และรหัสผ่าน ปกติเรามักจะใช้วิธีการเพิ่มข้อมูลเข้าไปในตารางโดยตรงผ่านโปรแกรมจัดการด้วย phpMyAdmin ใน Laravel Framework จะมีเครื่องมือที่ช่วยในการเตรียมข้อมูลตั้งต้น นั่นคือ **Seeding**

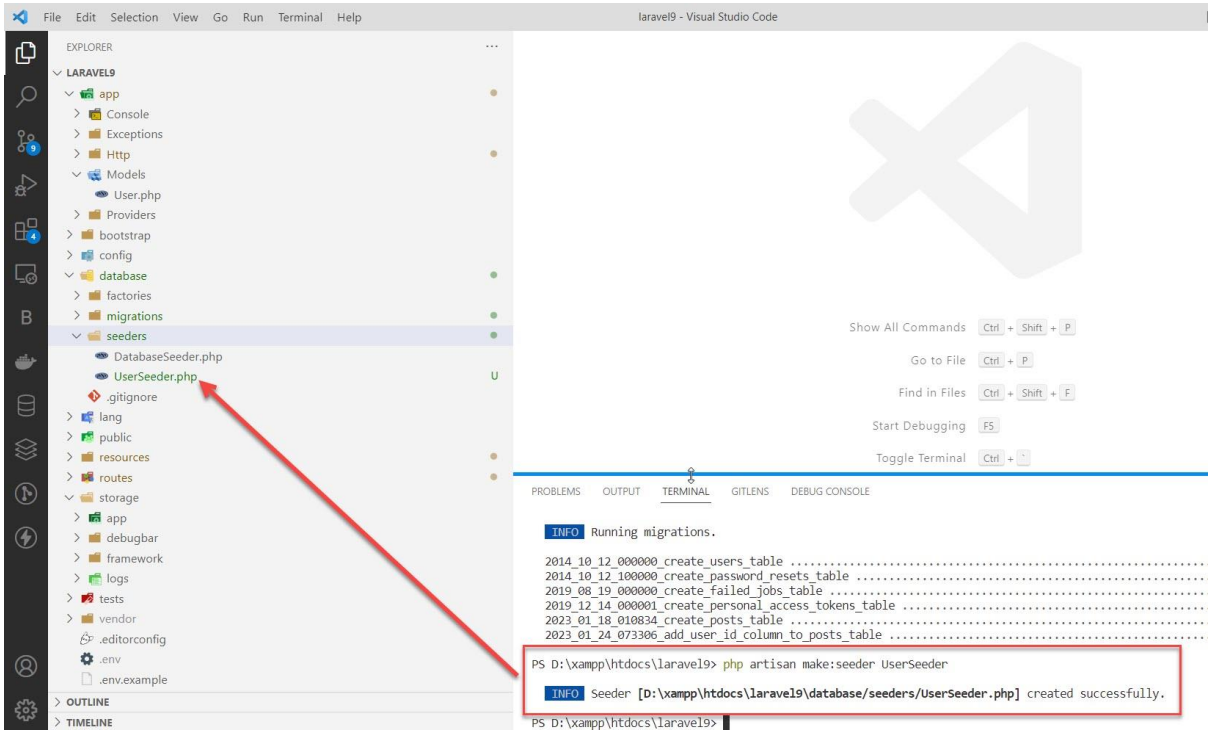
คำสั่งที่ใช้สร้าง

```
php artisan make:seeder UserSeeder
```
คำสั่งสำหรับสร้าง UserSeeder

เมื่อรันคำสั่งด้านบนนี้แล้วไฟล์จะถูกสร้างขึ้นและเก็บไว้ที่ โฟลเดอร์ database/seeders

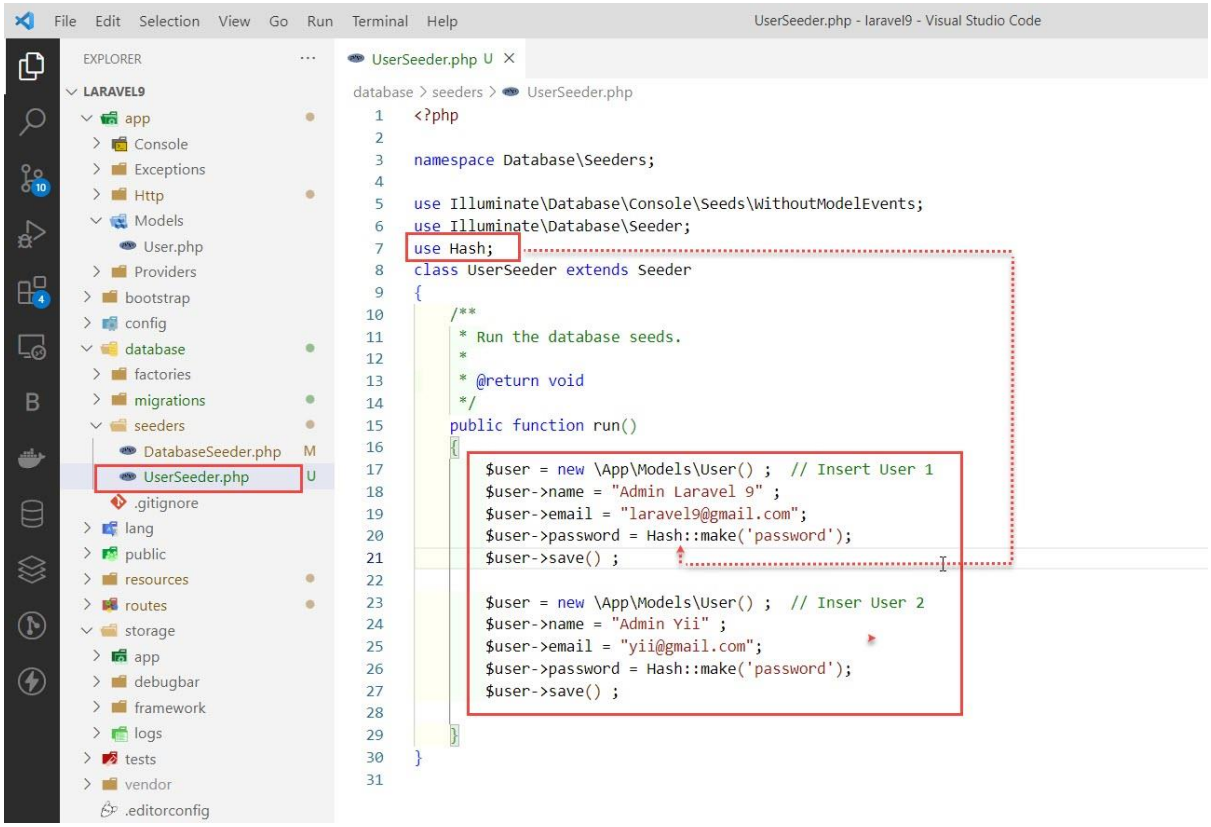
เพื่อให้มองเห็นภาพแล้วเข้าใจยิ่งขึ้น เรามาเขียนกันเลย

เปิด Terminal ใน VSCode ขึ้นมาแล้วพิมพ์คำสั่งและกด Enter ดังนี้



ไฟล์ UserSeeder.php จะถูกสร้างขึ้นมาให้ จะเห็นว่าเราได้ตั้งชื่อ ว่า UserSeeders เมื่อเราเปิดดูที่โฟลเดอร์ Models จะมี Models อยู่ 1 ไฟล์ที่ Laravel สร้างมาให้เราตอนติดตั้ง นั่นคือ User.php ซึ่งเป็น Models สำหรับตาราง users เพื่อให้เข้าใจตรงกัน ในการตั้งชื่อ Seeders ควรตั้งชื่อให้สอดคล้องกับ Model ในที่นี้ เราจะกำหนดข้อมูลตั้งต้นให้กับตาราง users เราจึงสร้างไฟล์ seeders ชื่อว่า UserSeeder นั่นเอง

เปิดไฟล์ database/seeders/UserSeeder.php ขึ้นมา แล้วเขียนโค้ดสำหรับเพิ่มข้อมูลในตาราง ดังนี้



ให้เขียนโค้ดลงภายในฟังก์ชัน run() จากตัวอย่างเป็นการกำหนดข้อมูลตั้งต้นโดยอ้างอิงตารางจาก Models ชื่อว่า User และเพิ่มข้อมูล เข้าไป 2 รายการ โดยกำหนดชื่อฟิลด์และระบุข้อมูลให้กับฟิลด์ที่ต้องการเพิ่ม คือ name ,email , password ในส่วนของ password มีการใช้ Hash::make() สำหรับเข้ารหัสผ่าน

เปิดไฟล์ database/seeders/DatabaseSeeder.php แล้วกำหนดไฟล์ที่ต้องการ seed ในเมธอด run() โดยพิมพ์คำสั่ง `$this->call(`

```
[
    UserSeeder::class,      //ระบุ  class ที่ต้องการ คั่นด้วยเครื่องหมาย ,
];
```

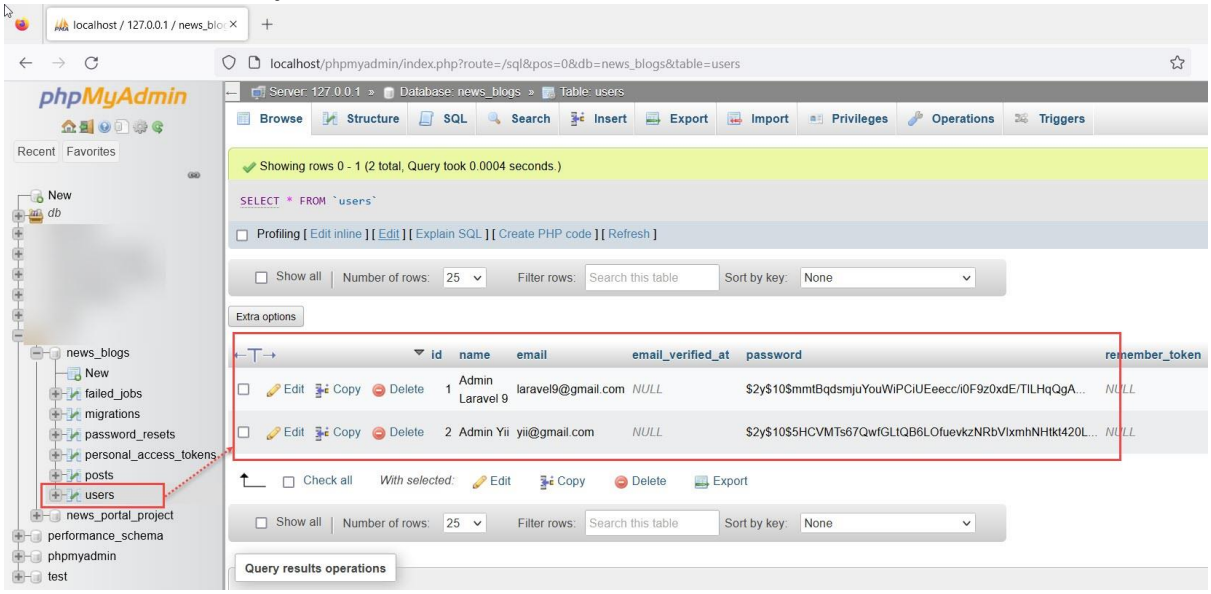
รันคำสั่ง บน Terminal

```
php artisan db:seed
```



ข้อมูลที่เราได้กำหนดเป็นค่าตั้งต้นก็จะถูกนำเข้าในตาราง users ตามที่เราได้กำหนดไว้

เปิดดูใน phpMyAdmin อีกครั้งเพื่อดูว่าข้อมูลในตาราง users ถูกเพิ่มเข้าไปในตารางจริงไหม คลิกดูที่ตาราง users ก็จะเห็นมีว่ามีข้อมูลเพิ่มเข้าไปในตาราง 2 รายการตามที่เราได้ระบุไว้ในไฟล์ UserSeeder.php



จำลองเนื้อหา ด้วย Factory

คิดเนื้อหาไม่ออกให้นึกถึง Factory เคยเจอไหมเวลาที่เราออกแบบอะไรสักอย่างแล้วไม่มีข้อมูล ที่จะเติมลงไป จะใส่ประโยคหรือข้อความเองก็ไม่รู้จะใส่อะไร ยิ่งเวลาทำเว็บไซต์หรือออกแบบงานที่ต้องมีข้อมูลเยอะๆ แต่กลับไม่มีข้อมูลมาให้เลย

Factory จะเปรียบเหมือน Lorem Ipsum (โลเล็ม อิปซัม) นั่นคือ ก้อนประโยคหรือเนื้อหาจำลองแบบเรียบ ๆ ที่จัดเรียงมาแบบโคตรมั่วนิ่ม แต่ก็กลับดูเป็นประโยคที่จัดเรียงได้อย่างสวยงาม โดยสามารถนำมา Copy วางในงานของเราได้เลย ตัวอย่างเช่นเว็บ https://lorem.in.th และ https://www.lipsum.com



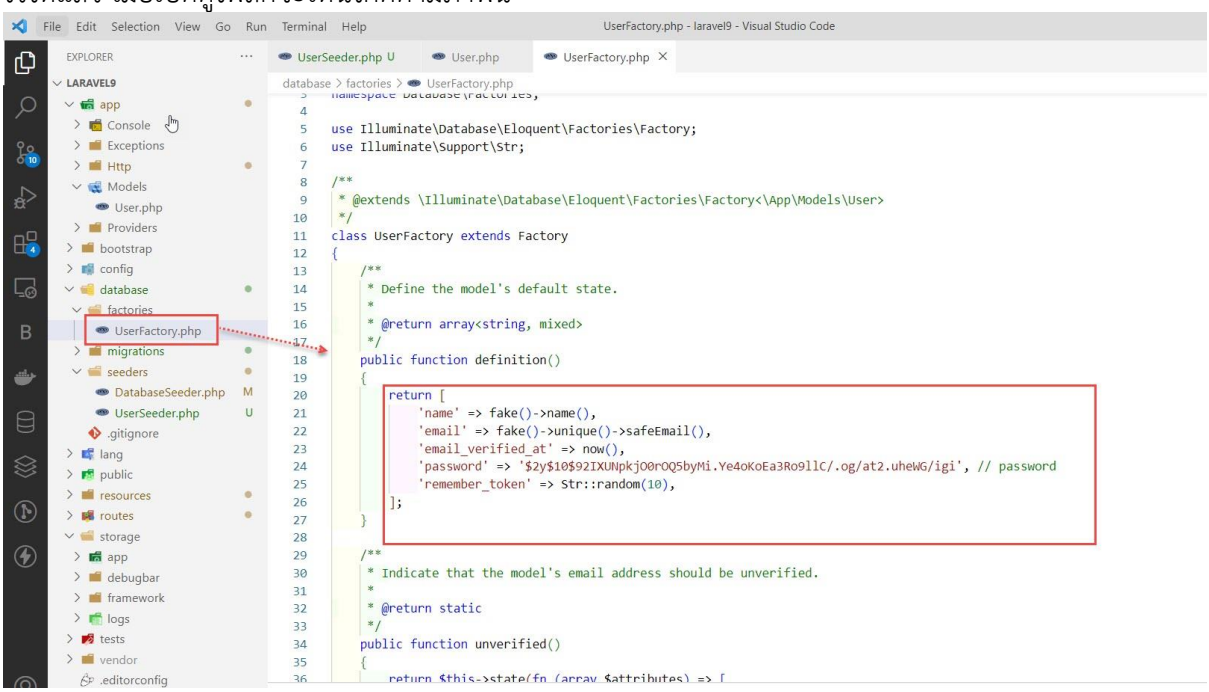


ใน Laravel Framework จะมีเครื่องมือสำหรับการสร้างเนื้อหาจำลอง เรียกว่า Factory

คำสั่งที่ใช้สร้าง

```
php artisan make:factory [ระบุชื่อ Factories]
```

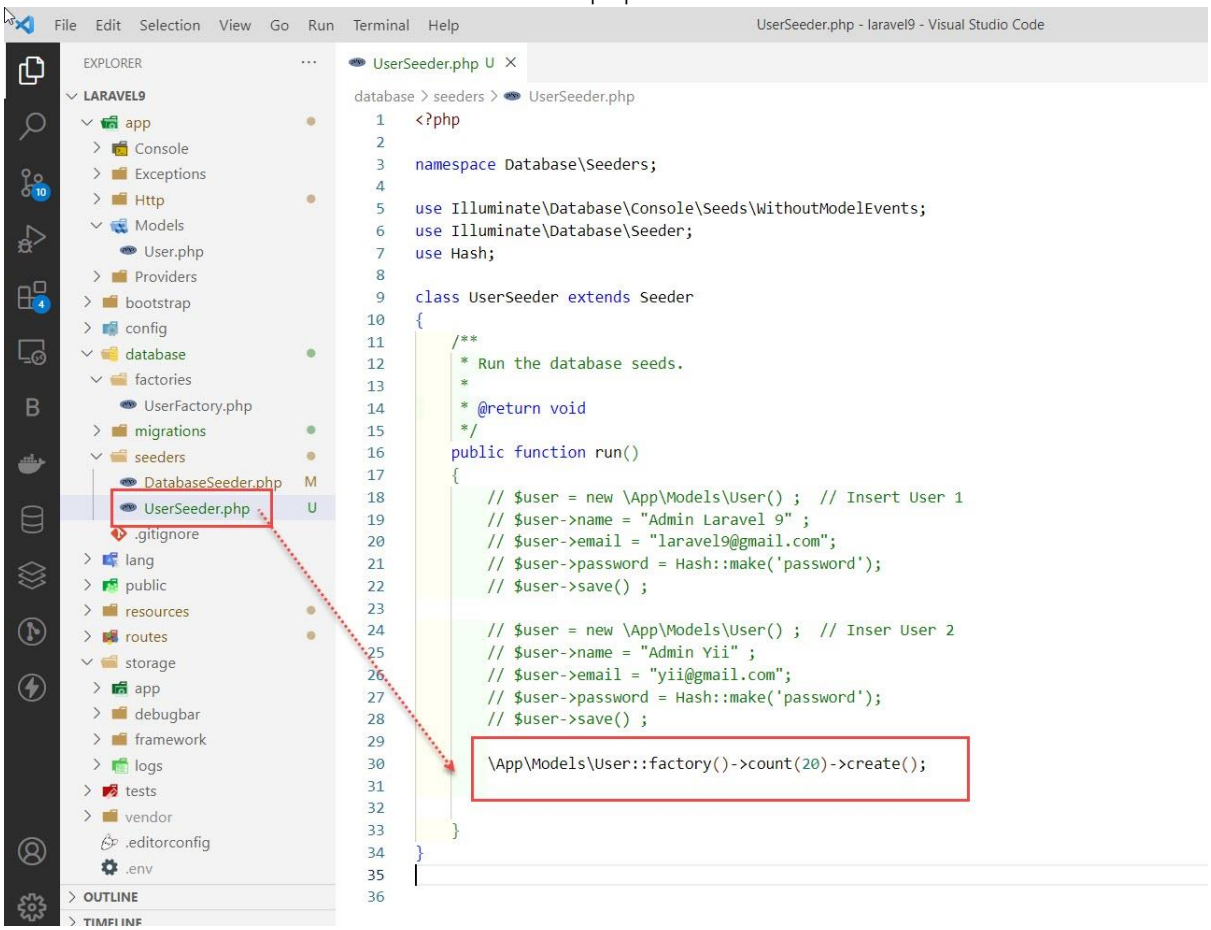
เมื่อรันคำสั่งด้านบนนี้แล้วไฟล์จะถูกสร้างขึ้นและเก็บไว้ที่ โฟลเดอร์ database/factories เนื่องจากว่า Laravel ได้สร้าง Factory มาให้เราแล้ว นั่นคือ UserFactory.php แล้วมีการเขียนโค้ดไว้ให้แล้ว เมื่อเปิดดูไฟล์ก็จะเห็นโค้ดตามภาพนี้



ให้เขียนโค้ดลงภายในฟังก์ชัน definition() จากตัวอย่างเป็นการเขียนกำหนดข้อมูลที่เราต้องการใส่ โดยกำหนดอยู่รูปแบบของ array โดยการระบุชื่อฟิลด์ในตาราง และใช้ fake()->name() เพื่อ generate ข้อความ

ตัวอย่างวิธีการใช้งานเพิ่มเติมได้ที่ https://github.com/FakerPHP/Faker

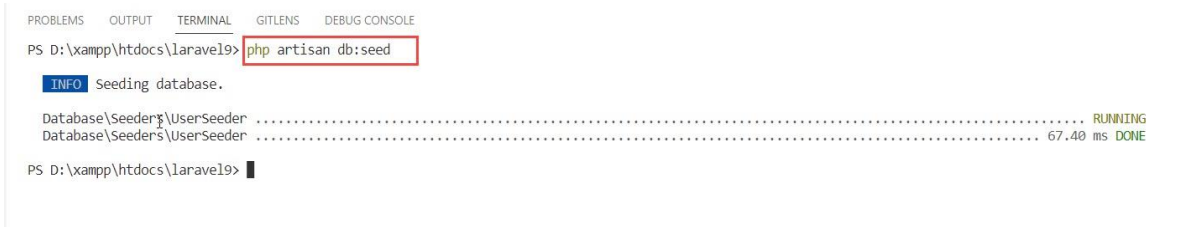
เปิดไฟล์ database/seeders/UserSeeder.php ขึ้นมา แล้วแก้ไขโค้ด ดังนี้



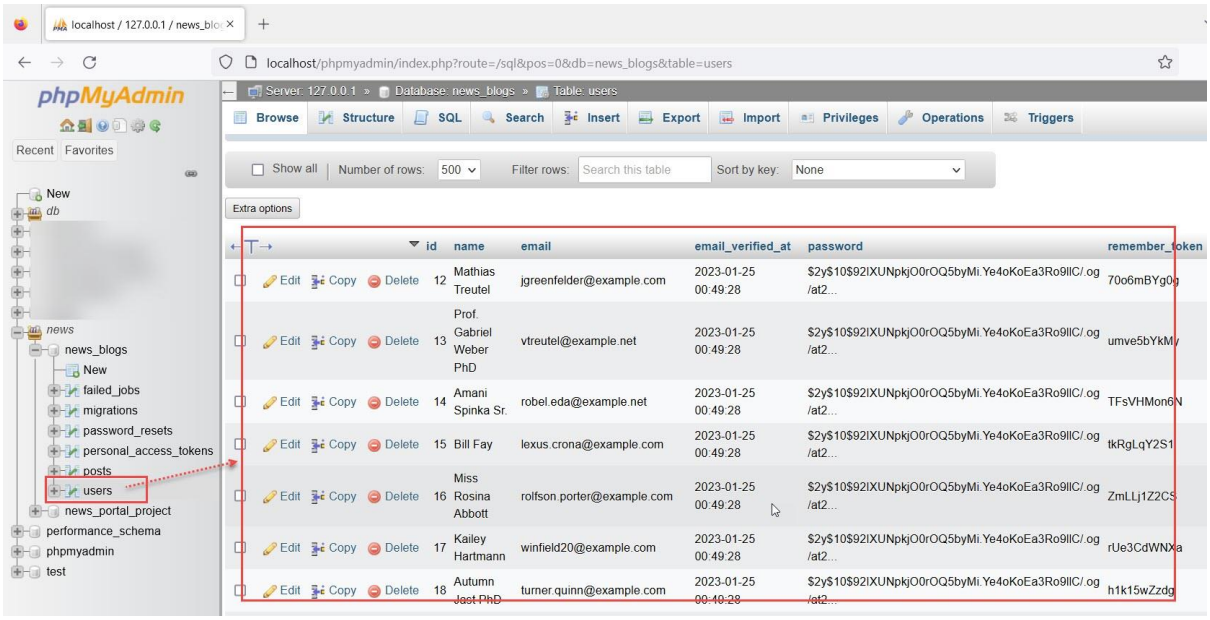
จากภาพนี้ ได้มีการ comment โค้ดเก่าไว้ แล้วเพิ่มคำสั่ง สำหรับการสร้างเนื้อหาจำลองแทน โดยเรียกใช้คำสั่ง factory แล้วระบุว่าต้องการจะเพิ่มข้อมูลเข้าไปในตาราง users กี่รายการ ในตัวอย่างเป็นการเพิ่มข้อมูลจำนวน 20 รายการ

บันทึกไฟล์แล้วรันคำสั่งนี้ ใน Terminal

```
php artisan db:seed
```



เปิดดูใน phpMyAdmin อีกครั้งเพื่อดูว่าข้อมูลในตาราง users ถูกเพิ่มเข้าไปในตารางจริงไหม คลิกดูที่ตาราง users ก็จะเห็นมีว่ามีข้อมูลเพิ่มเข้ามาตามจำนวนรายการที่ระบุ

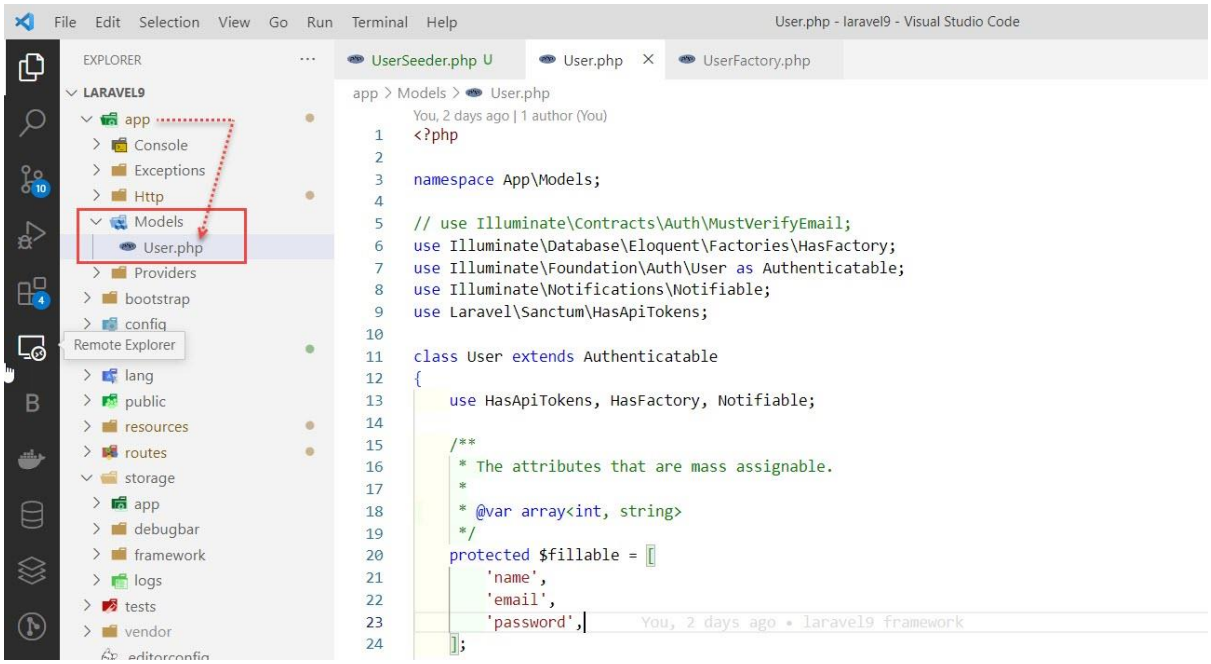


ทีนี้เรามีตารางไหนที่อยากจะเพิ่มข้อมูล เราสามารถเพิ่มลงในตารางต่าง ๆ ที่เราต้องการได้โดยใช้วิธีการตามนี้ได้เลย เช่น ถ้าเราอยากจะเพิ่มเนื้อหาจำลองเข้าไปในตาราง posts แต่จะเห็นว่ายังไม่มีโมเดล posts เราจะต้องสร้างโมเดล posts เพื่อเป็นตัวอ้างอิงตารางในฐานข้อมูลก่อน จากนั้นก็สร้าง factory มาใช้งาน

พื้นฐานการเขียน Models

เป็นส่วนที่สำคัญในการจัดการเกี่ยวกับฐานข้อมูล เช่น การเพิ่มข้อมูล การอ่านข้อมูล การแก้ไขข้อมูล การลบข้อมูล การพัฒนาโปรแกรมในรูปแบบ MVC จะถูกกำหนดเป็นส่วนของ Model เพื่อใช้เชื่อมโยงกับฐานข้อมูล Laravel รองรับฐานข้อมูลทั้ง MySQL, PostgreSQL, SQLite, SQL Server

Models จะเก็บไว้ที่โฟลเดอร์ app\Models

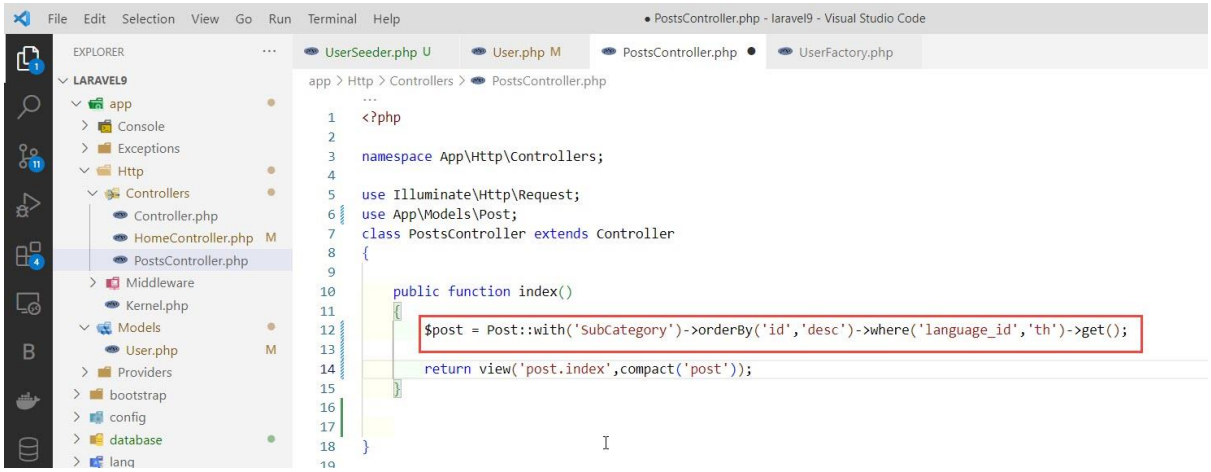


ปกติทุก ๆ ครั้งเราเขียนโปรแกรมจัดการกับฐานข้อมูล ส่วนใหญ่แล้วเราจะเขียนเป็นภาษา SQL ธรรมดาในการดึงข้อมูล การเพิ่มข้อมูล หรือแม้กระทั่งการ Join ข้อมูลหลาย ๆ ตาราง จะต้องเขียนคำสั่ง SQL ยาวหน่อยกว่าจะได้แต่ละส่วน เช่น คำสั่ง SQL ใช้ดึงข้อมูลจากฐานข้อมูลมาแสดง ปกติก็จะเขียนแบบนี้

SQL Output Statistics

```
SELECT U.USERID , U.USERCODE , U.USERFULLNAME ,
 U.POSITIONNAMUE , U.DEPARTMENT ,
 U.DEPARTMENTNAME , S.DEGREELEVELNAMEABB,
  U.DEGREEFULLNAME ,S.MAJORNAME ,
  U.UNIVERSITYNAME,U.COUNTRYNAME,
  U.USERTYPENAME, U.USERSTATUS ,
   U.ADMITDATE , U.LEAVEDATE,S.WORKAGETXT
FROM PIMS.V_API_USER_INFO  S
WHERE S.USERSTATUS < 70
AND   S.POSITIONNAME not in 'admin'
ORDER BY S.DEPARTMENTCODE, S.USERFULLNAME;
```

พอมาเปลี่ยนวิธีการดึงข้อมูลด้วย Models คำสั่งก็จะสั้นลง เช่น จะดึงข้อมูลจากฐานข้อมูลมาแสดง เมื่อใช้ Models ก็จะเขียน ดังนี้



จะเห็นว่าการเขียนจะสั้นลง เนื่องจาก Models จะเก็บข้อมูลเป็น Object (ORM) เพื่อให้ง่ายต่อการเรียกใช้งาน และช่วยในการ Join ตารางหลาย ๆ ตารางได้ง่าย โดยไม่จำเป็นต้องเขียนภาษา SQL เลย โดยการใช้งานนั้นจะต้องมีการสร้างไฟล์โมเดลขึ้นมาก่อน เพื่ออ้างอิงตารางในฐานข้อมูล ปกติการเรียกใช้งานโมเดลจะเขียนโค้ดส่วนของ controllers แล้วส่งไปให้ views แสดงผล

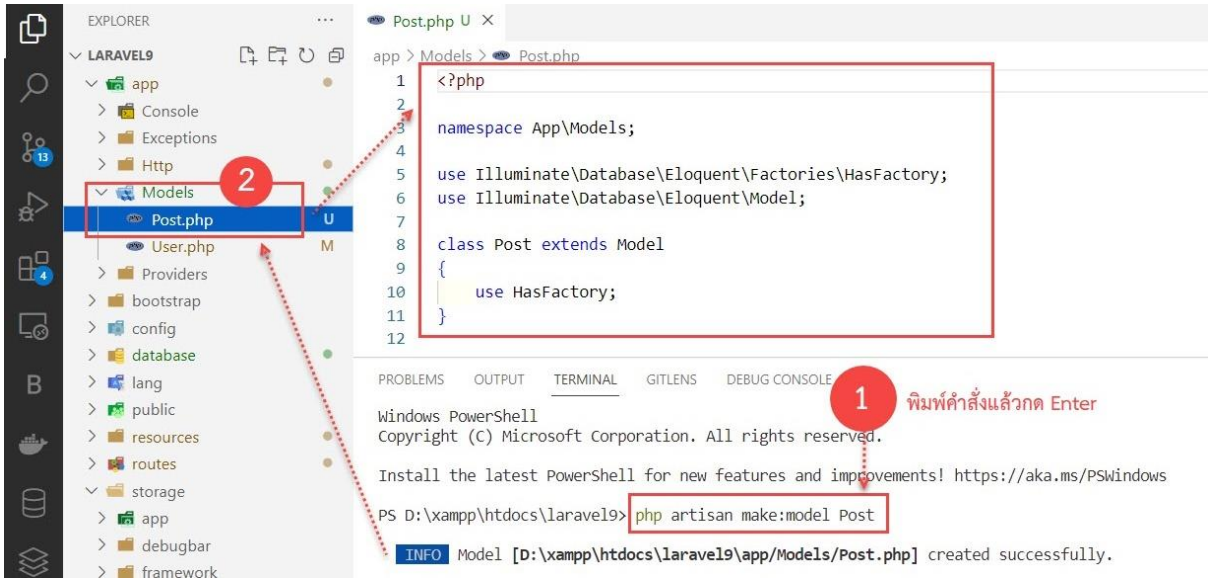
การสร้าง Models สามารถใช้คำสั่ง Artisan ดังนี้

```
php artisan make:model [ระบุชื่อ Models] -m
```

ถ้ากรณี ใช้ keyword -m ต่อท้ายคำสั่ง จะเป็นการสร้าง migrations มาให้ด้วย

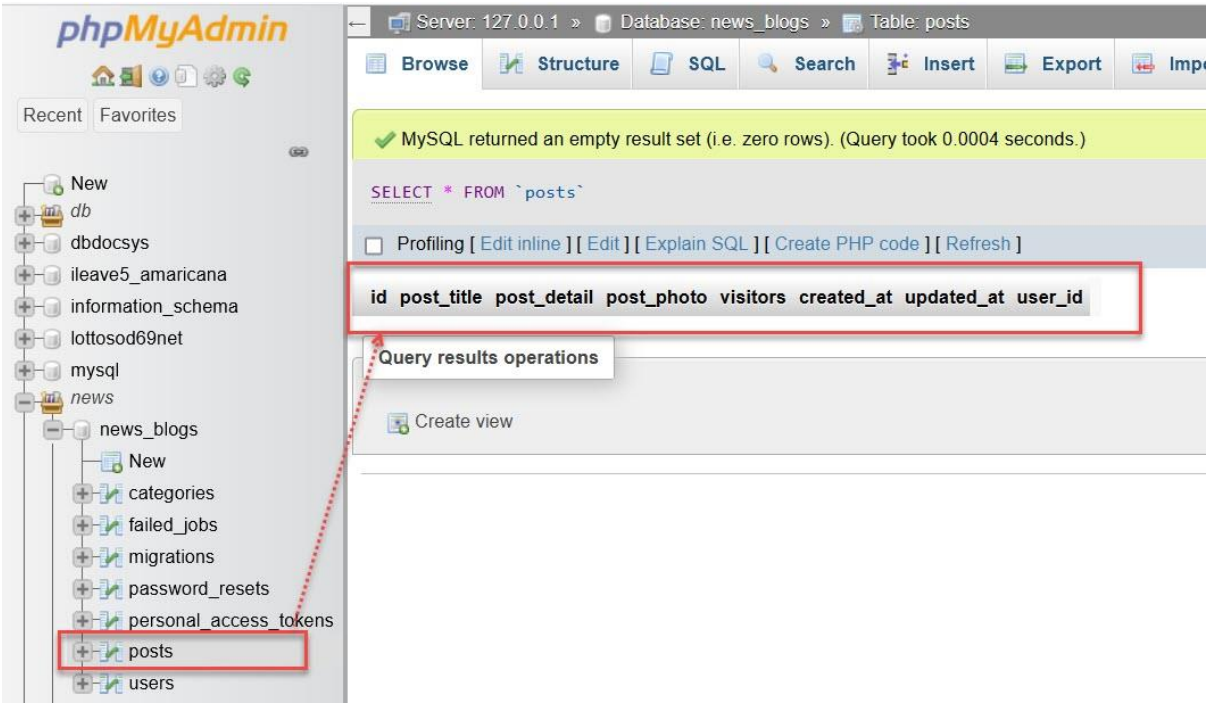
1.สร้าง Model ให้กับตาราง posts เปิด Terminal ขึ้นมา แล้วพิมพ์คำสั่ง ดังนี้

```
php artisan make:model Post
```



เมื่อรันคำสั่งเสร็จระบบก็จะสร้างไฟล์ Post.php เก็บไว้ที่โฟลเดอร์ app\Model เพียงเท่านี้เราก็สามารถใช้เรียกใช้งานโมเดลได้แล้ว

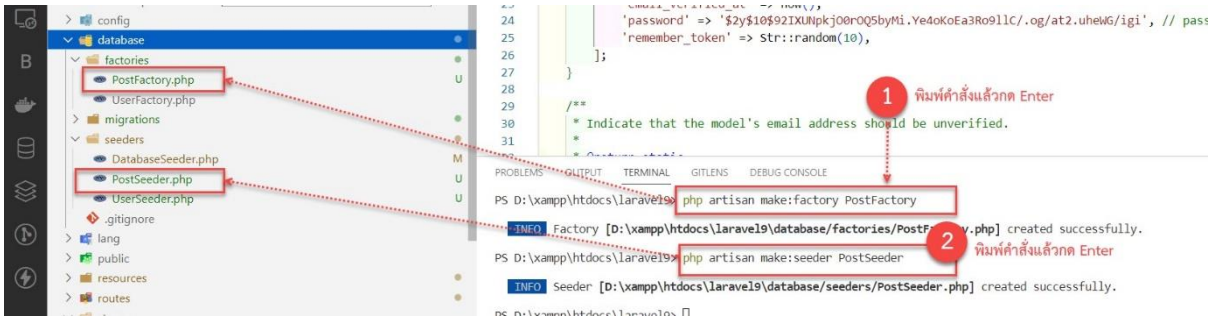
2.ลองตรวจสอบข้อมูลตาราง posts ใน โปรแกรม phpMyAdmin ดู



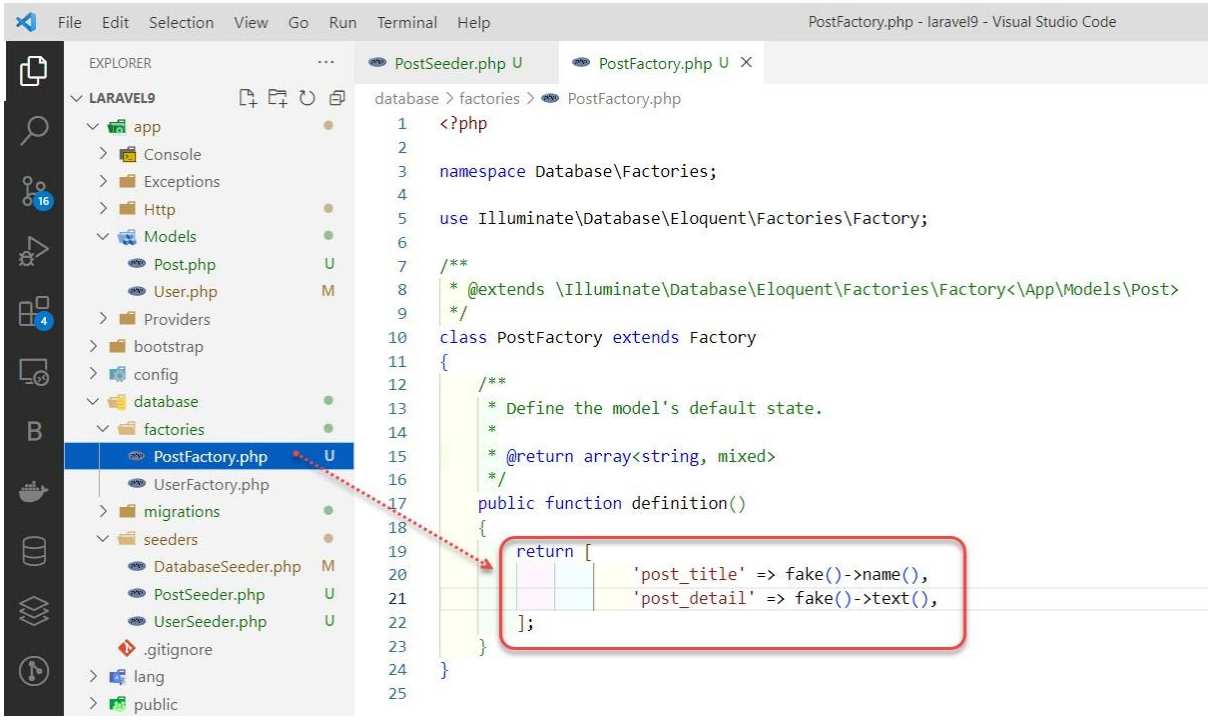
แต่เอ๊ะ !! ในตาราง posts ยังไม่มีข้อมูลนิ ก็ไปเพิ่มใส่สิ การเพิ่มข้อมูลลงตารางจะมีหลายวิธีครับ อาจจะใช้วิธีเพิ่มลงไปตรง ๆ ใน โปรแกรม phpMyAdmin หรือใช้วิธีสร้างฟอร์มสำหรับการกรอกข้อมูลแล้วบันทึกลง ก็ได้ครับ เบื้องต้นเราต้องการข้อมูลหรือเนื้อหาตัวอย่างสำหรับการแสดงผลข้อมูล เราจะใช้วิธีการโดยใช้ Factory เพื่อ generate ข้อความแล้วเพิ่มลงตาราง posts ครับ ไปสร้าง Factory กันเลย

2.1 เริ่มสร้าง PostFactory และ PostSeeder ด้วยคำสั่ง

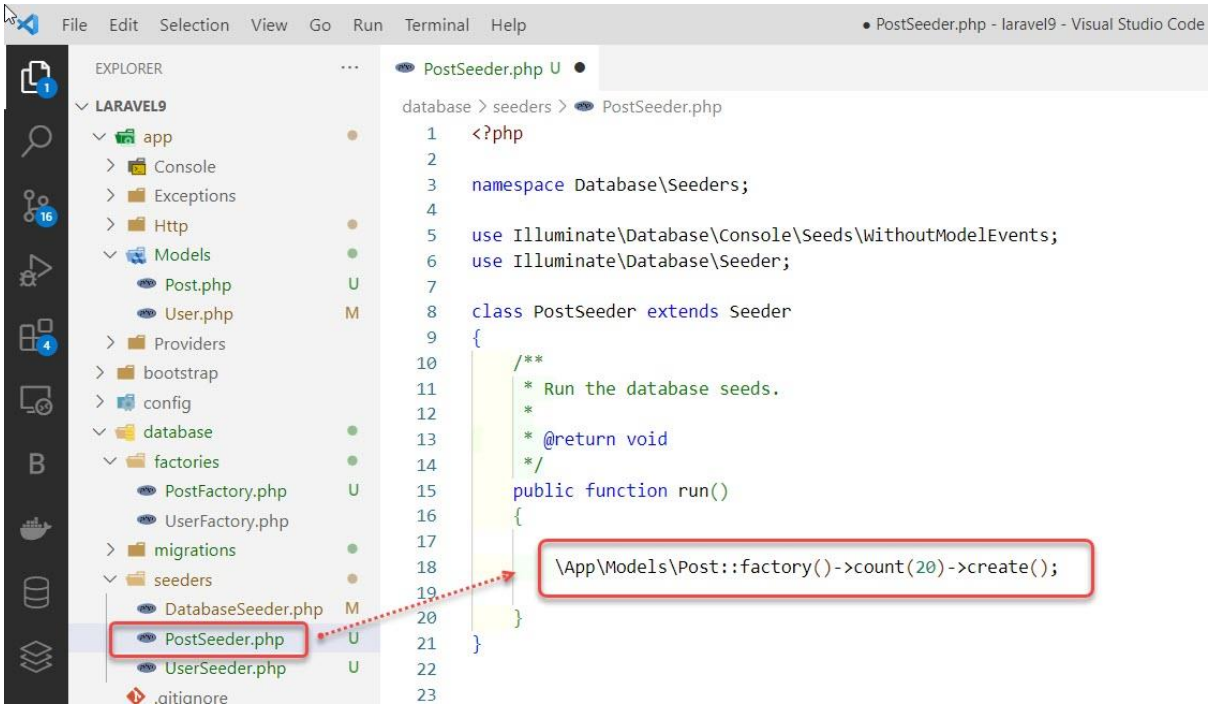
```
php artisan make:factory PostFacetory
และ
php artisan make:seeder PostSeeder
```



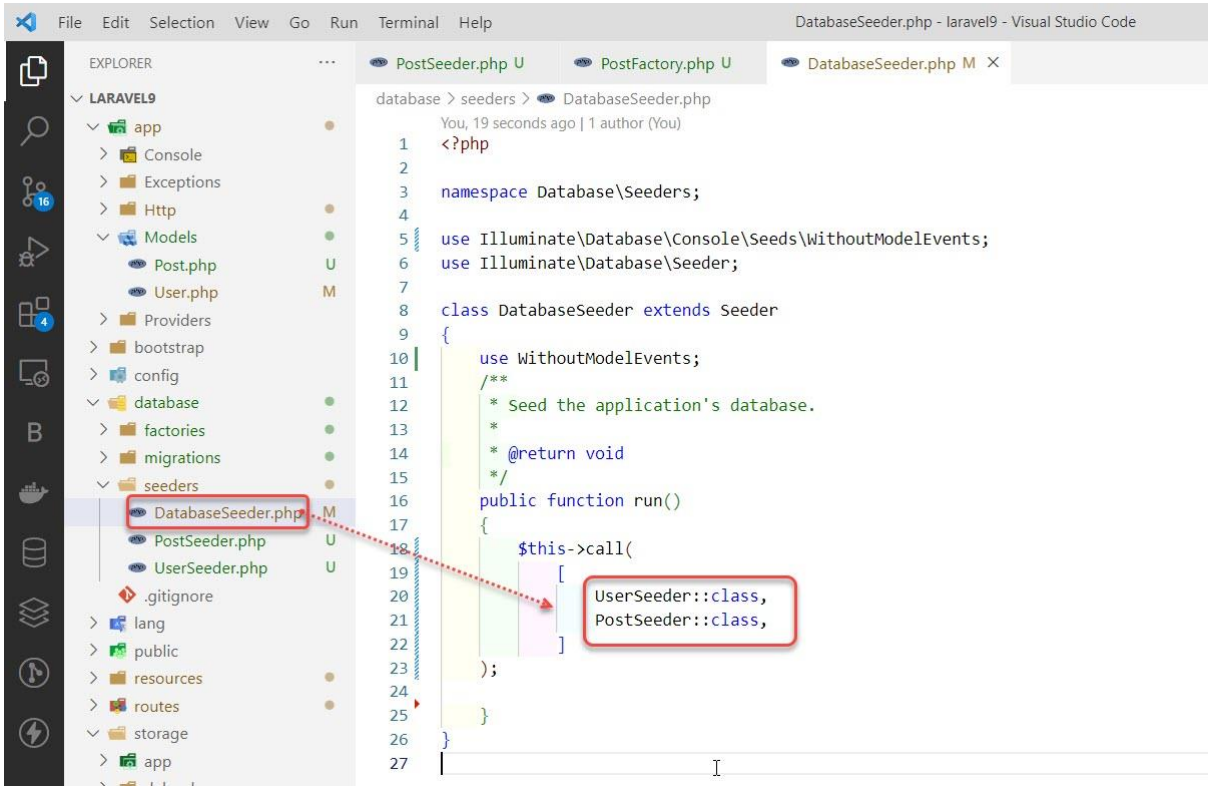
2.2 เปิดไฟล์ PostFactory.php ในโฟลเดอร์ database/factories/... แล้วพิมพ์คำสั่งและบันทึกไฟล์ดังนี้



2.3 เปิดไฟล์ PostSeeder.php ในโฟลเดอร์ database/seeders/... แล้วพิมพ์คำสั่งและบันทึกไฟล์ดังนี้



2.4 เปิดไฟล์ DatabaseSeeder.php ในโฟลเดอร์ database/seeders/... แล้วเพิ่มโค้ดเข้าไป ตามตัวอย่างในภาพ เพื่อเรียกใช้งานไฟล์ seeder ทั้งสองไฟล์ ดังนี้



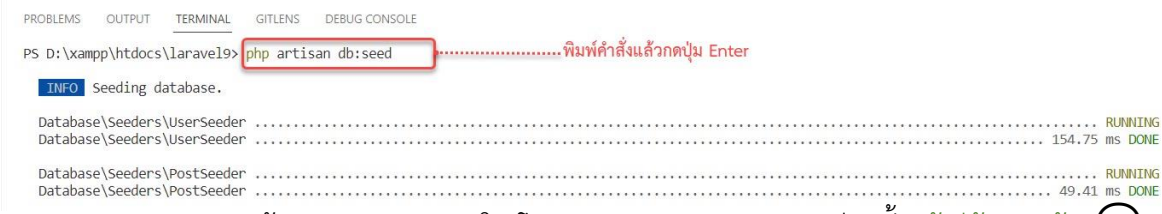
2.5 พิมพ์คำสั่ง ที่ Terminal เพื่อทำการ migrate ใหม่

php artisan migrate:refresh	ใช้สำหรับ rollback ตารางทั้งหมดและสั่งให้รัน migration ใหม่

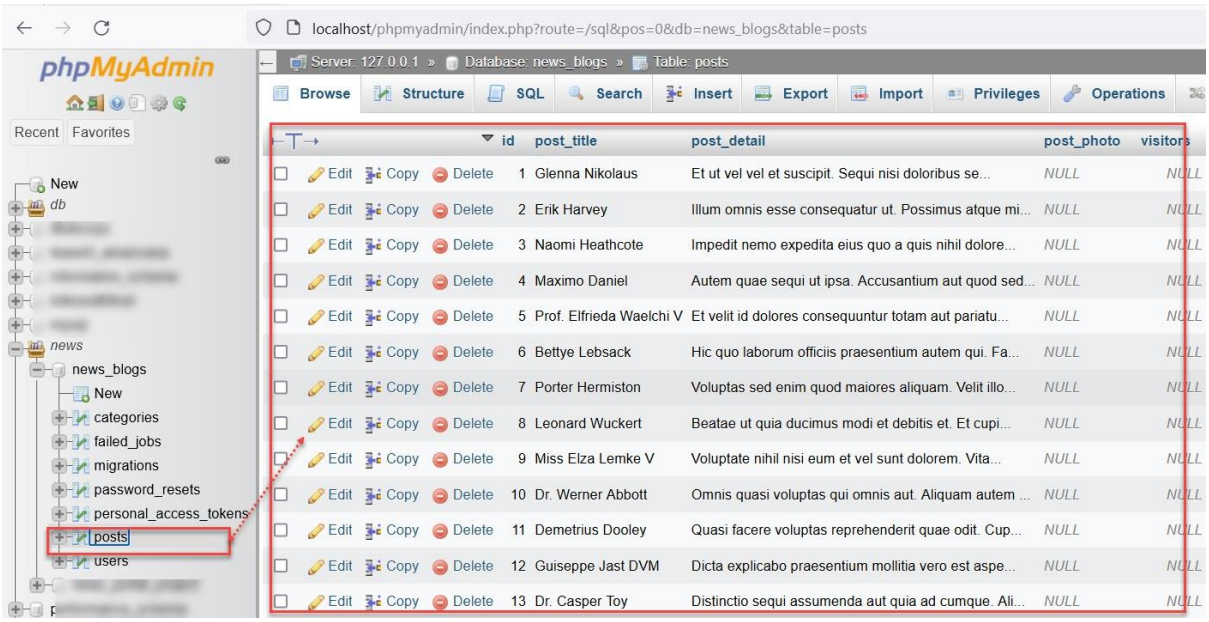


2.6 พิมพ์คำสั่ง ที่ Terminal เพื่อทำการ seeding ข้อมูลอีกครั้ง

```
php artisan db:seed
```



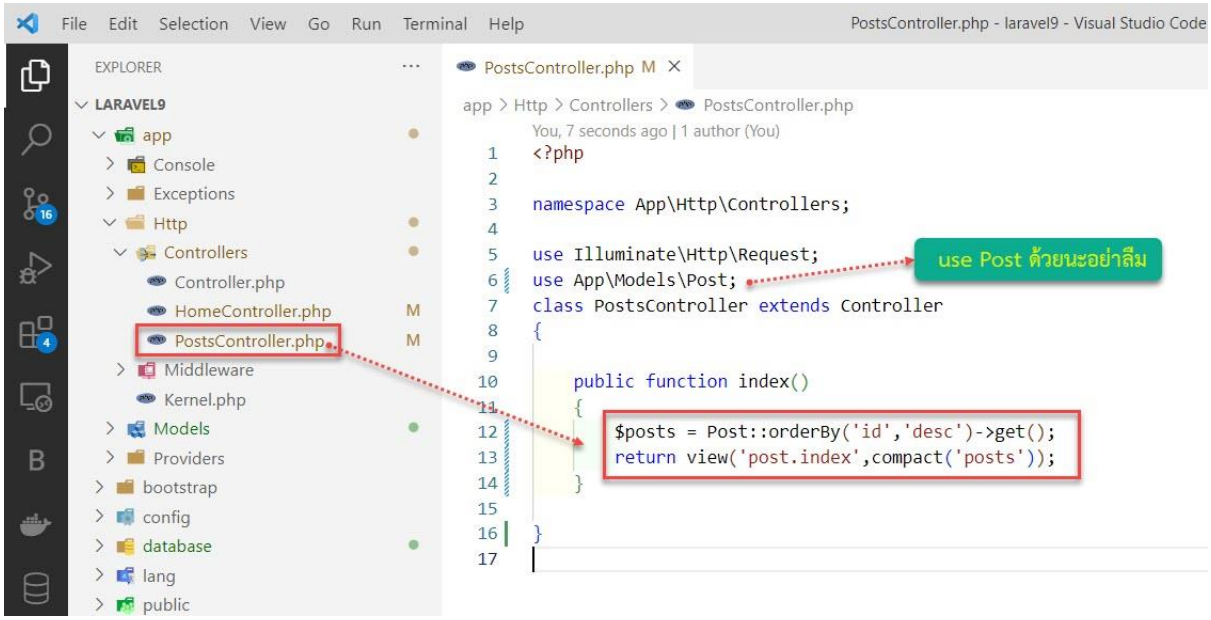
2.7 ตรวจสอบข้อมูลตาราง posts ใน โปรแกรม phpMyAdmin ดูอีกครั้ง เย้ มีข้อมูลแล้ว 😊



ข้อดีของการใช้วิธีนี้ คือ บางครั้งที่เราเขียนเว็บไปแล้วเราอาจจะต้องมีการแก้ไขตาราง เราก็แก้ไขไฟล์ migrate และ seeder แล้วก็ใช้คำสั่งรันเพื่อเริ่มต้นใหม่ เราก็จะได้ตารางและข้อมูลเริ่มต้น ซึ่งดีกว่าเราจะไปแก้ที่ phpMyAdmin โดยตรง

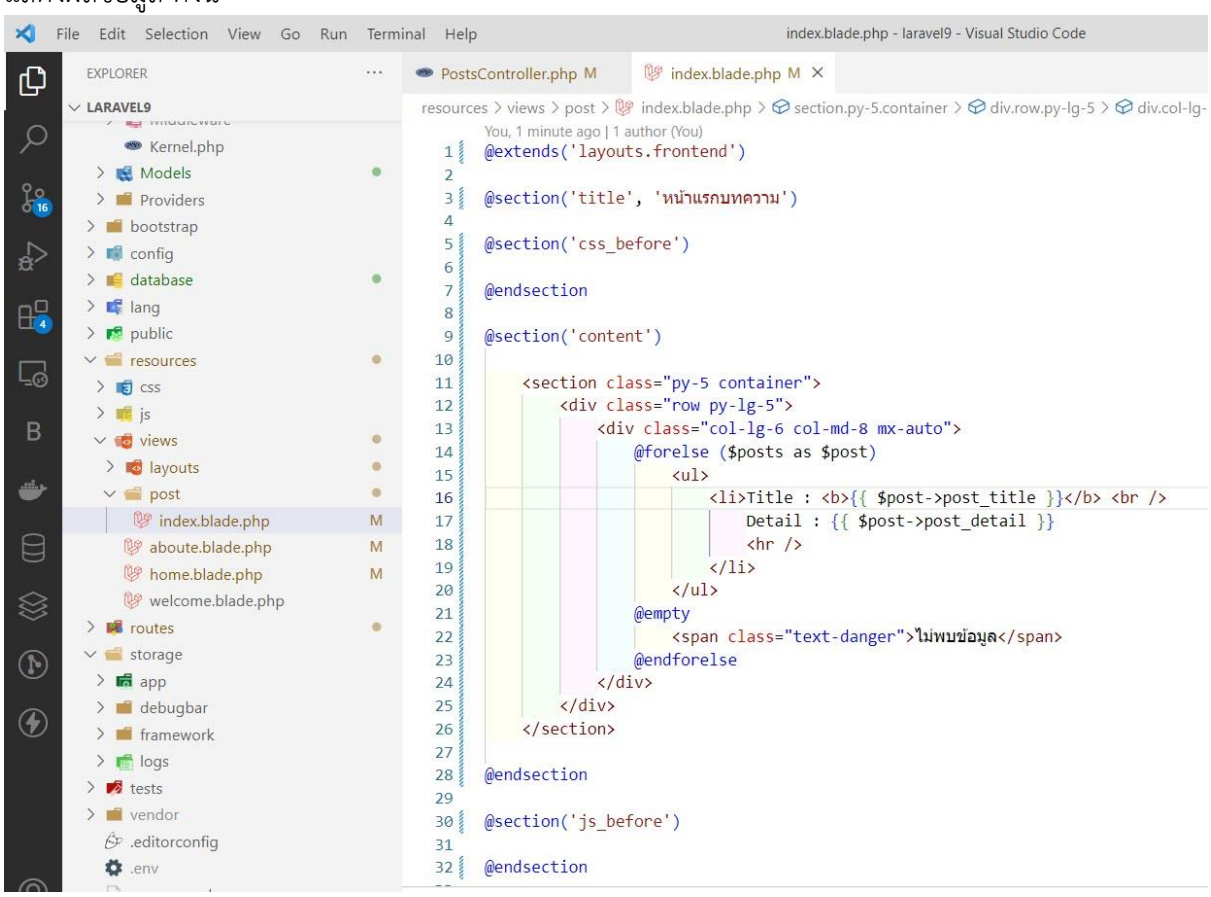
3.ต่อไปลองดึงข้อมูลจากตารางไปแสดงผลที่หน้าเว็บดู กันต่อเลยครับ

3.1 เปิดไฟล์ PostsController.php ในโฟลเดอร์ app/Http/Controllers/... แล้วแก้ไขโค้ดในเมธอด index() และบันทึกไฟล์ ดังนี้

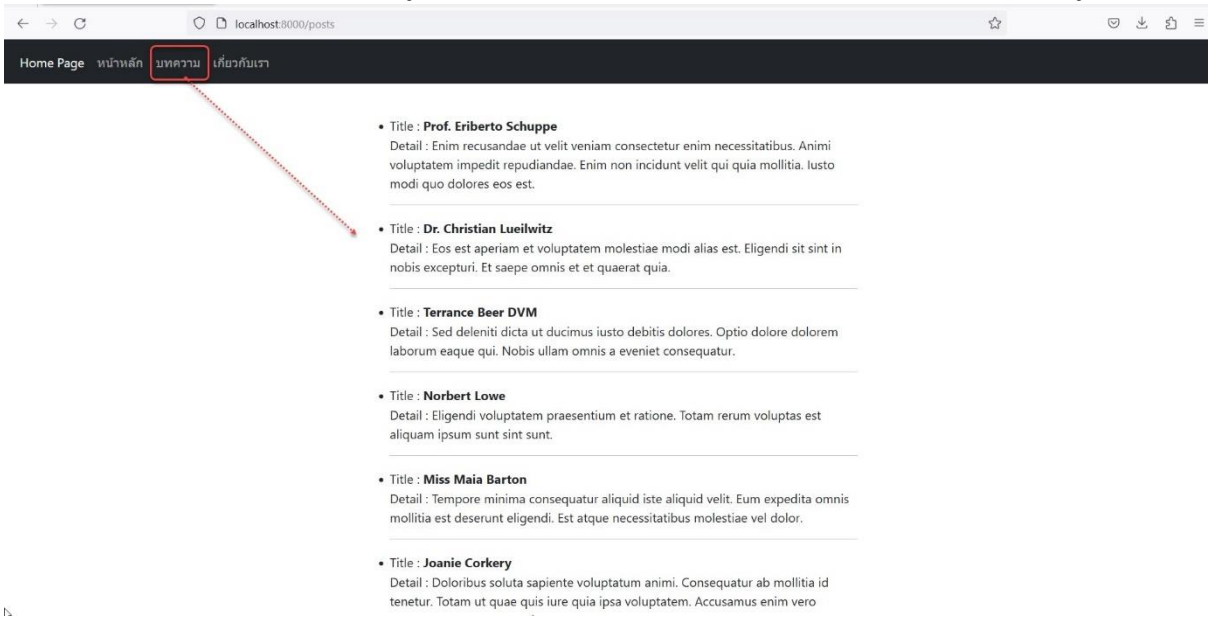


อธิบาย ในไฟล์ PostsControllers.php บรรทัดที่ 6 เป็นคำสั่งสำหรับอ้างอิงโมเดลที่อยู่ในโฟล์เดอร์ App\Models ในตัวอย่างนี้เป็นการเรียกใช้งานโมเดล Post (ตาราง posts) บรรทัดที่ 7 เป็นการสร้างคลาสชื่อ PostsController สืบทอด (extends) จาก controller ในบรรทัดที่ 10 เป็นการสร้างเมธอดชื่อ index() แล้วมีการเขียนโค้ดสำหรับดึงข้อมูลจากฐานข้อมูลโดยเรียกตาราง posts (Eloquent ORM) โดยมีการกำหนดเงื่อนไขด้วยการใช้ orderBy() เพื่อแสดงข้อมูลที่ถูกเพิ่มล่าสุด เรียงข้อมูลจากใหม่ไปเก่า จากนั้นก็สั่งให้ดึงข้อมูลจากฐานข้อมูลมาทั้งหมด ด้วย get() บรรทัดที่ 13 เป็นคำสั่งสำหรับการ render ไฟล์ที่อยู่ใน views คือ post.index แล้วมีการส่งค่าตัวแปรชื่อ posts เพื่อให้ views แสดงผลข้อมูล คำสั่งพวกนี้เดียวเราจะได้เรียนรู้ในหัวข้อถัด ๆ ไปนะครับ

3.2 เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources/views/post เพื่อแก้ไขในส่วนของการแสดงผลข้อมูล ดังนี้



3.3 บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/ ที่ Browser คลิกที่เมนู **บทความ**



เพียงเท่านี้เราก็สามารถดึงข้อมูลที่อยู่ในฐานข้อมูลมาแสดงผลได้แล้ว

ไม่สร้าง Migration ได้ไหมขี้เกียจอ่ะ

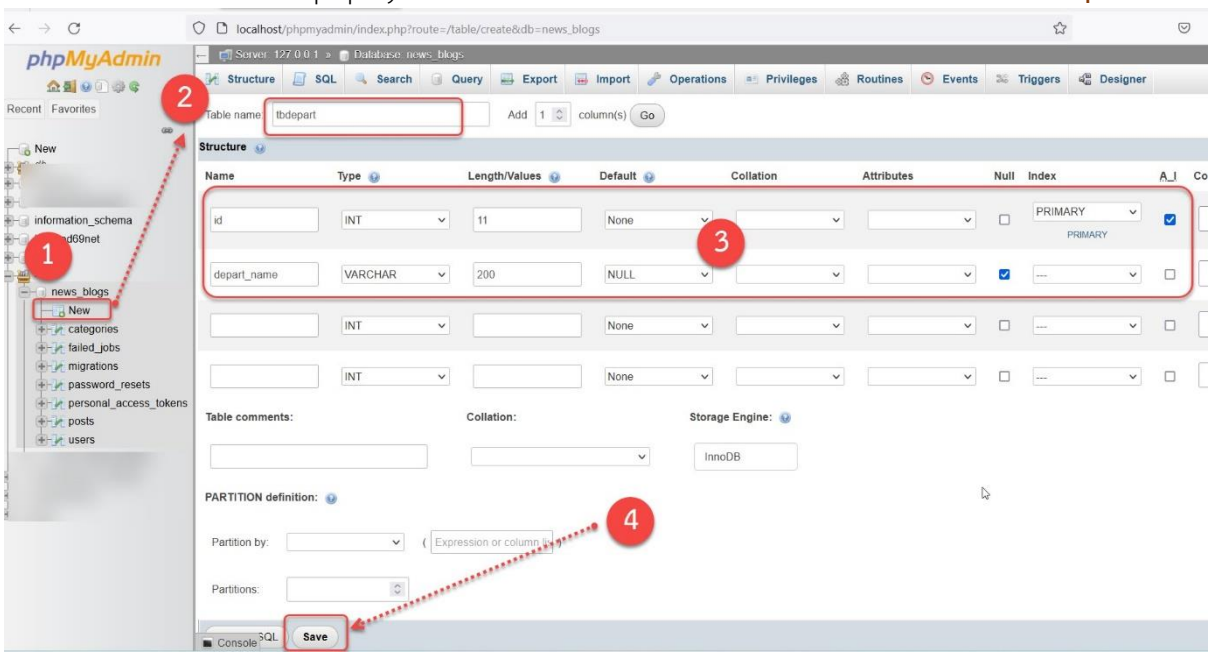
จากบทเรียนที่ผ่านมาเราได้เรียนรู้วิธีการการสร้าง Model และ Migration ไปแล้วนั้นก็คงจะเข้าใจกันบ้างแล้วนะครับ ซึ่งที่ผ่านมานั้นเราจะสร้าง Migration ก่อน ต่อจากนั้นก็ทำการสร้าง Models หรือจะสร้าง Models ก่อนแล้วไปสร้าง Migration ถามว่าผิดไหม ก็ไม่ผิดนะ สามารถทำได้เหมือนกัน ขึ้นอยู่กับว่าผู้พัฒนาแต่ละคนจะสร้างด้วยวิธีไหน ให้เข้าใจตรงกันนะครับว่า Migration นั้นมันเป็นการสร้างตารางในกรณีที่ในฐานข้อมูลยังไม่มีตาราง ๆ นั้น แต่ถ้าในฐานข้อมูลนั้นมีตารางแล้ว ก็ไม่ต้องสร้าง Migration ก็ได้ ยกตัวอย่าง เรามีก้อนฐานข้อมูลก้อนหนึ่งอยู่ในรูปแบบนามสกุล SQL เช่น databases.sql ซึ่งนำมาจากโปรเจคอื่นที่ไม่ได้เขียนด้วย Laravel แล้วเรา import ฐานข้อมูลนั้นเข้าโดยใช้ phpMyAdmin ก็จะได้ตารางในฐานข้อมูลเราเพื่อที่จะใช้ Laravel เขียนแทน เราก็เพียงแค่สร้าง Models และกำหนดค่าบางอย่าง เช่น

```
protected $table = 'atricle';
```

เพื่ออ้างอิงตารางเพื่อเรียกใช้เท่านั้นเอง คิดว่าทุกท่านคงจะ งง ใช่ไหม ต้อง งง แน่ ๆ งั้นไปเขียนโค้ดกันเลย

อ้างถึง... ข้อความที่โม้ มาข้างบนนั้น

1.ให้เปิดโปรแกรม phpMyAdmin ขึ้นมาแล้วสร้างตารางขึ้นมาใหม่ ตั้งชื่อตารางว่า **tbdepart**



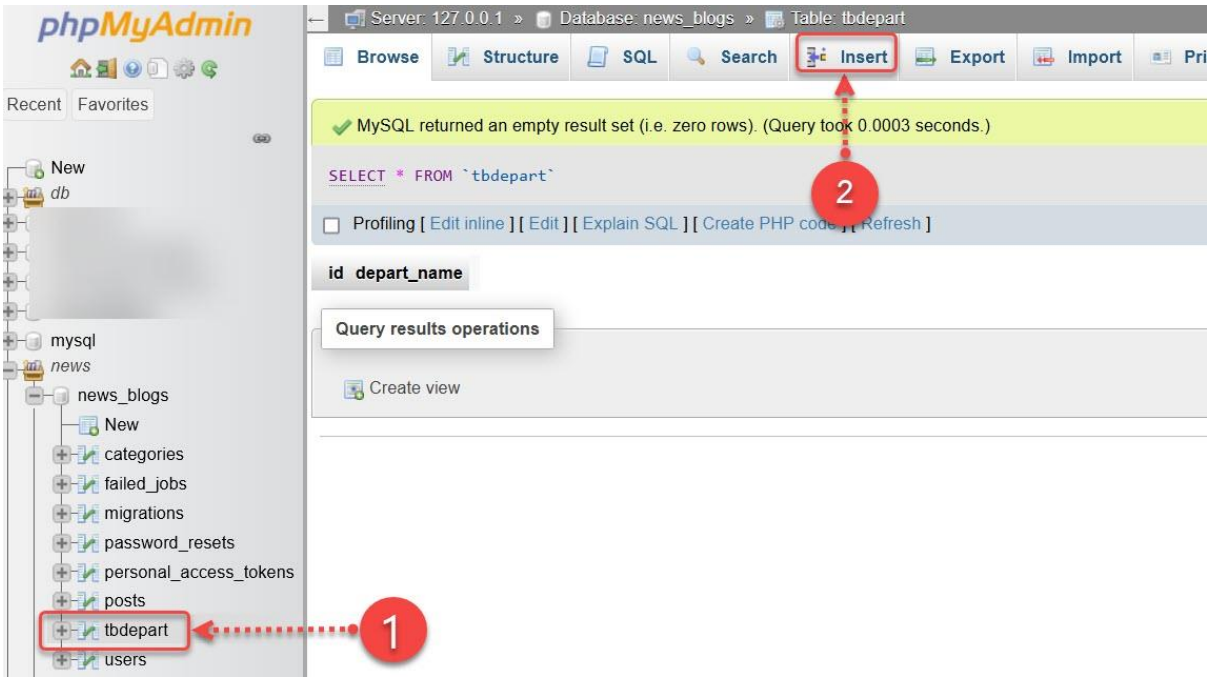
หมายเลข 1 คลิกที่ New เพื่อสร้างตารางใหม่

หมายเลข 2 ที่ช่อง Table name ให้ตั้งชื่อตารางใหม่ ชื่อว่าตาราง tbdepart

หมายเลข 3 สร้างฟิลด์ชื่อ id int(11) ระบุ index เป็นคีย์หลัก primary key และ AutoIncrements

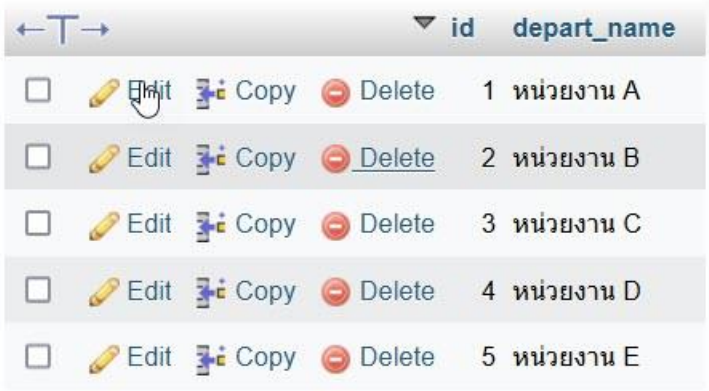
หมายเลข 4 กดปุ่ม save เพื่อสร้างตาราง tbdepart

ตาราง tbdepart จะถูกสร้างขึ้นมาให้เรา จากนั้นทำการเพิ่มข้อมูลลงไป โดยคลิกที่ตาราง tbdepart ก่อน จากนั้นคลิกที่เมนู Insert เพื่อเพิ่มข้อมูลลงในตาราง



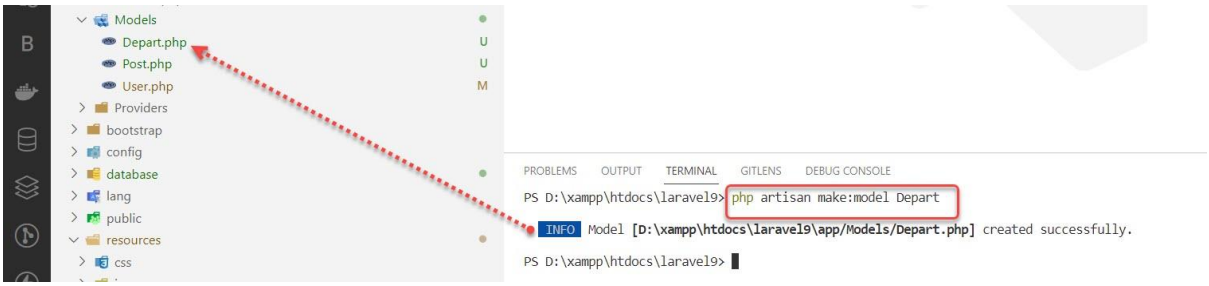
หมายเลข 1 คลิกที่ตาราง tbdepart

หมายเลข 2 คลิกที่เมนู Insert แล้วกรอกข้อมูลเข้าไป



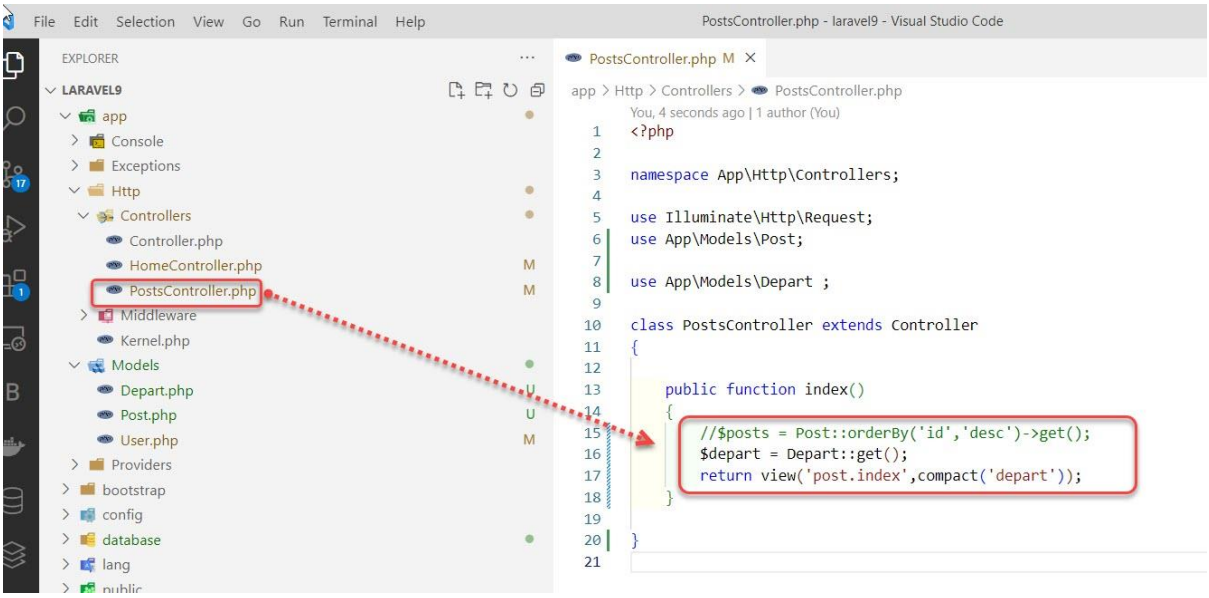
ที่นี้ตาราง tbdepart ก็มีข้อมูลแล้ว

2. เปิด Terminal แล้ว พิมพ์คำสั่ง php artisan make:model Depart และกด Enter

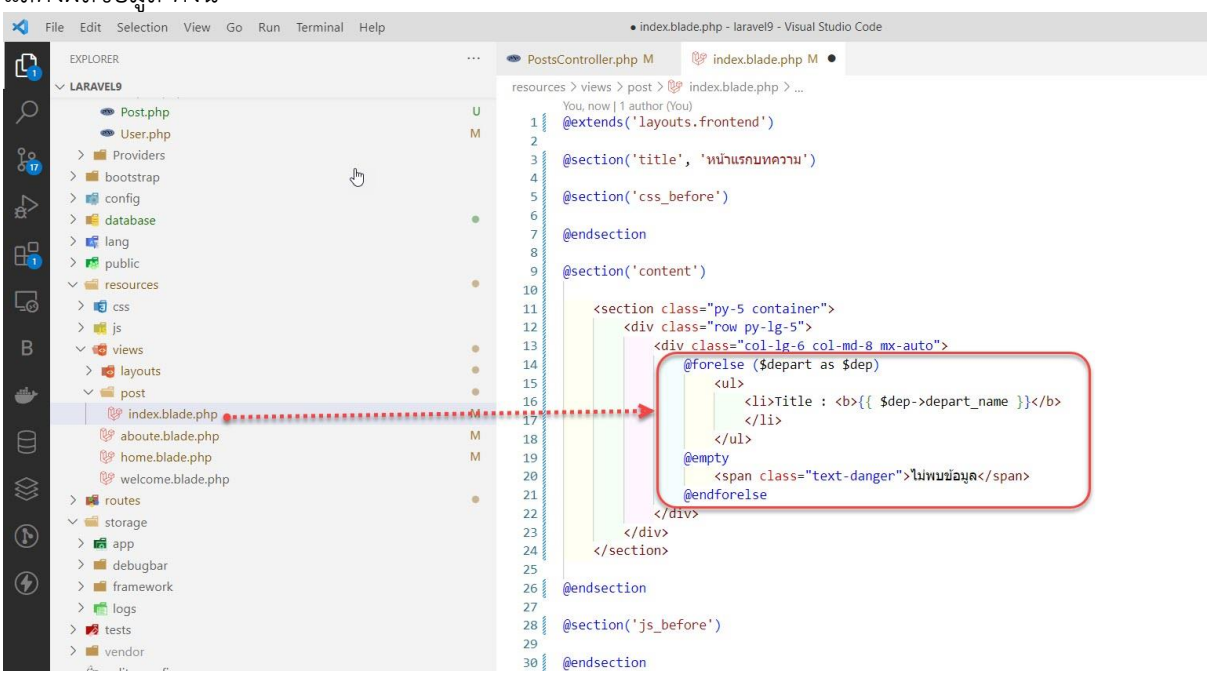


โมเดล Depart จะถูกสร้างขึ้นมาให้

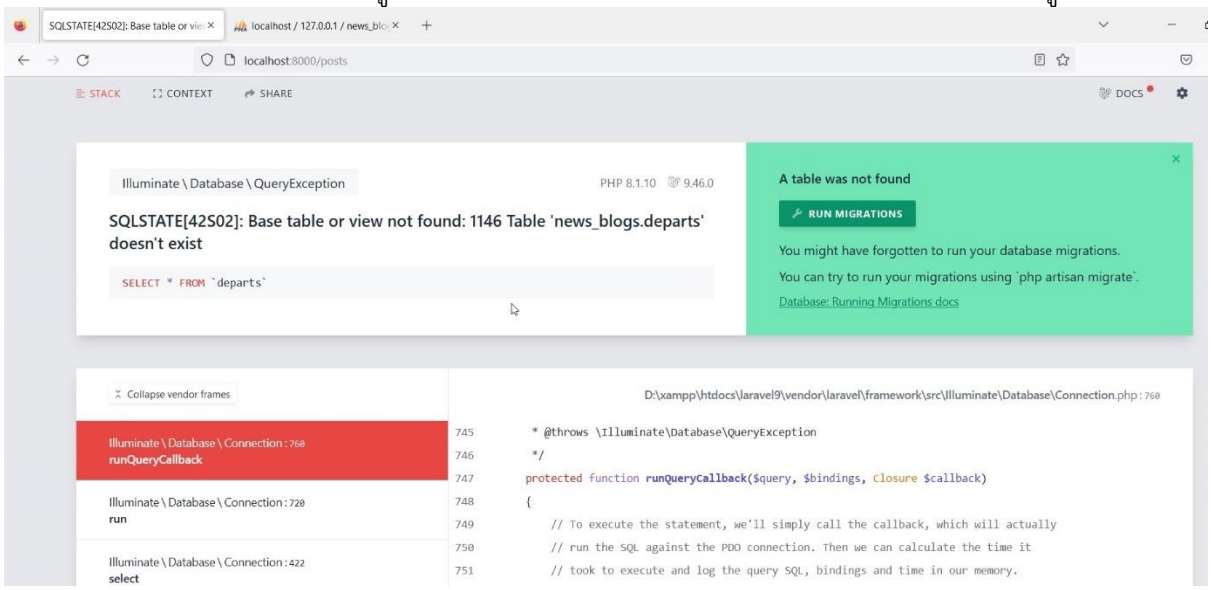
3.เปิดไฟล์ PostsController.php ในโฟลเดอร์ app/Http/Controllers/... แล้วแก้ไขโค้ดในเมธอด index() และบันทึกไฟล์ ดังนี้



4.เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources/views/post เพื่อแก้ไขในส่วนของการแสดงผลข้อมูล ดังนี้



5.บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/ ที่ Browser คลิกที่เมนู **บทความ**



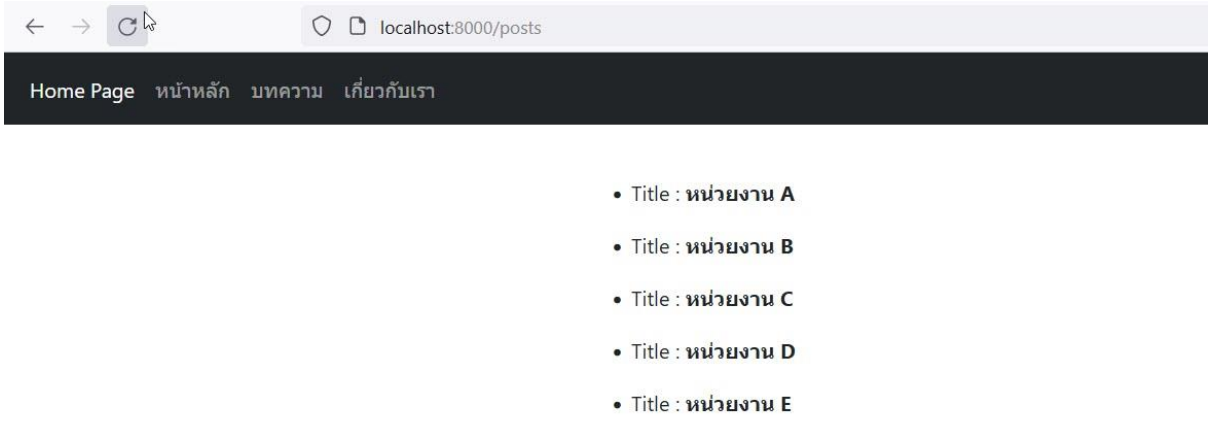
ผลลัพธ์ที่ได้ก็คือหน้าแสดงข้อผิดพลาด ซึ่งข้อผิดพลาดนี้บอกว่า Base table or view not found: 1146 Table 'news_blogs.departs' doesn't exist หมายความว่า ไม่พบตาราง departs ในฐานข้อมูล news_blogs จะเห็นว่าใน Laravel เราจะต้องตั้งชื่อตารางในรูปแบบของพหุพจน์ (plurals) ชื่อตารางที่สร้างต้องเติม s ต่อท้าย หากชื่อตารางลงท้ายด้วย y และด้านหน้า y เป็นพยัญชนะ สามารถเปลี่ยนให้เป็นพหูพจน์ได้ด้วยการตัด y ทิ้งและเติม -ies แทน เช่น city -> cities , daisy -> daisies , baby -> babies , box -> boxes , bus -> buses

แก้ไข ง่าย ๆ โดยการกำหนดตารางใน Models Depart ดังนี้

```
protected $table = 'tbdepart';
```



บันทึกไฟล์ แล้ว refresh หน้าอีกครั้ง



เอ่อ ก็เขียนแบบนี้ก็ได้นี่ ใช่ครับมันเป็นอย่างนั้นแหละ แต่แนะนำให้สร้างไว้เถอะครับ มีประโยชน์มากเวลาการบริหารจัดการต่าง ๆ เช่น การย้ายข้อมูลจากเครื่องหนึ่งไปอีกเครื่องหนึ่ง อยากเขียนโค้ดเครื่องอื่นก็จะบริหารได้สะดวก

ทุก ๆ ครั้งที่เราพัฒนาโปรแกรมหรือเว็บไซต์ ใน Laravel Framework เราจะต้องสร้างคอนโทรลเลอร์ สร้างโมเดล สร้าง migration เป็นประจำอยู่แล้ว ปกติจะใช้ คำสั่ง

```
php artisan make:model TestName
php artisan make:controller TestName
php artisan make:migration create_TestName_table
```

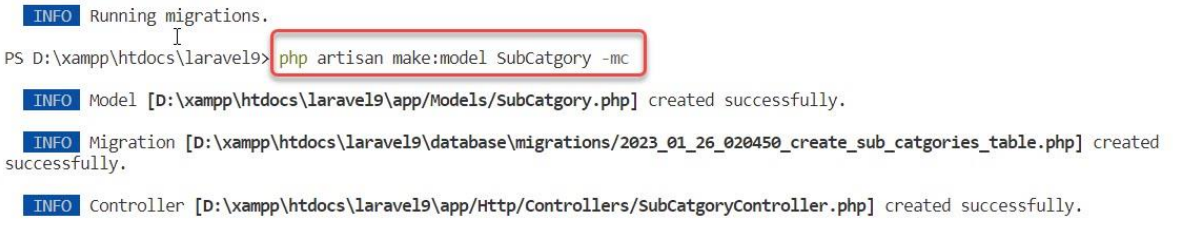
เราสามารถเขียนคำสั่งให้สั้นลงให้เหลือเพียงบรรทัดเดียวได้ โดยใช้คำสั่งนี้

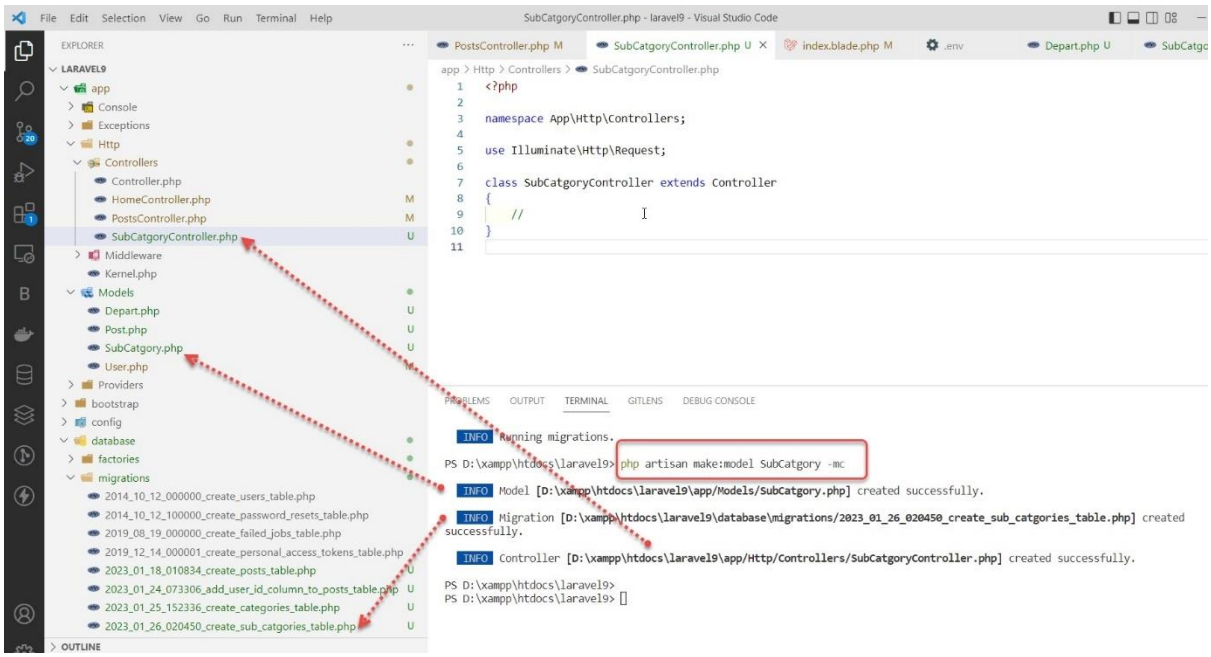
```
php artisan make:model [ชื่อโมเดล] -mc
```

เมื่อรันคำสั่งนี้แล้ว เราจะได้ไฟล์ Model , Migration และ Controller สร้างอยู่ในโฟลเดอร์นั้น ๆ

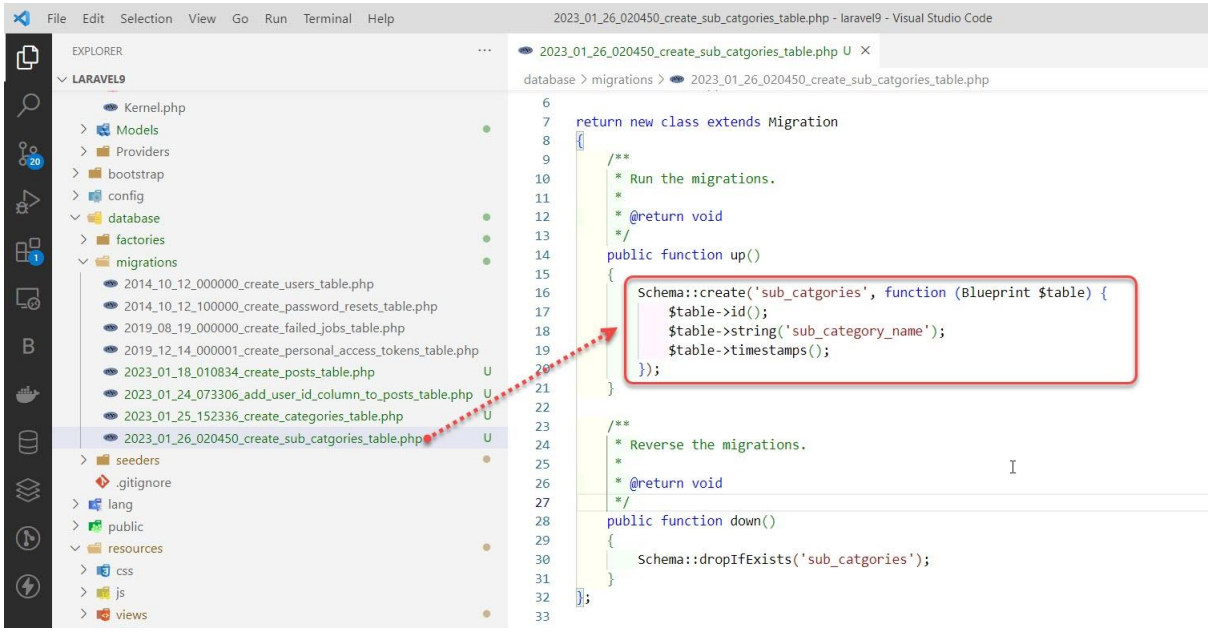
เพื่อให้เราเข้าใจยิ่งขึ้น ให้เปิด Terminal แล้วพิมพ์คำสั่ง เพื่อสร้าง Model , Migration และ Controller พิมพ์คำสั่งแล้วกด Enter

```
php artisan make:model SubCategory -mc
```





เปิดไฟล์ xxx_create_sub_categories_table.php ในโฟลเดอร์ database/migrations/... แล้วเพิ่มโค้ดคำสั่งในเมธอด up() ดังนี้

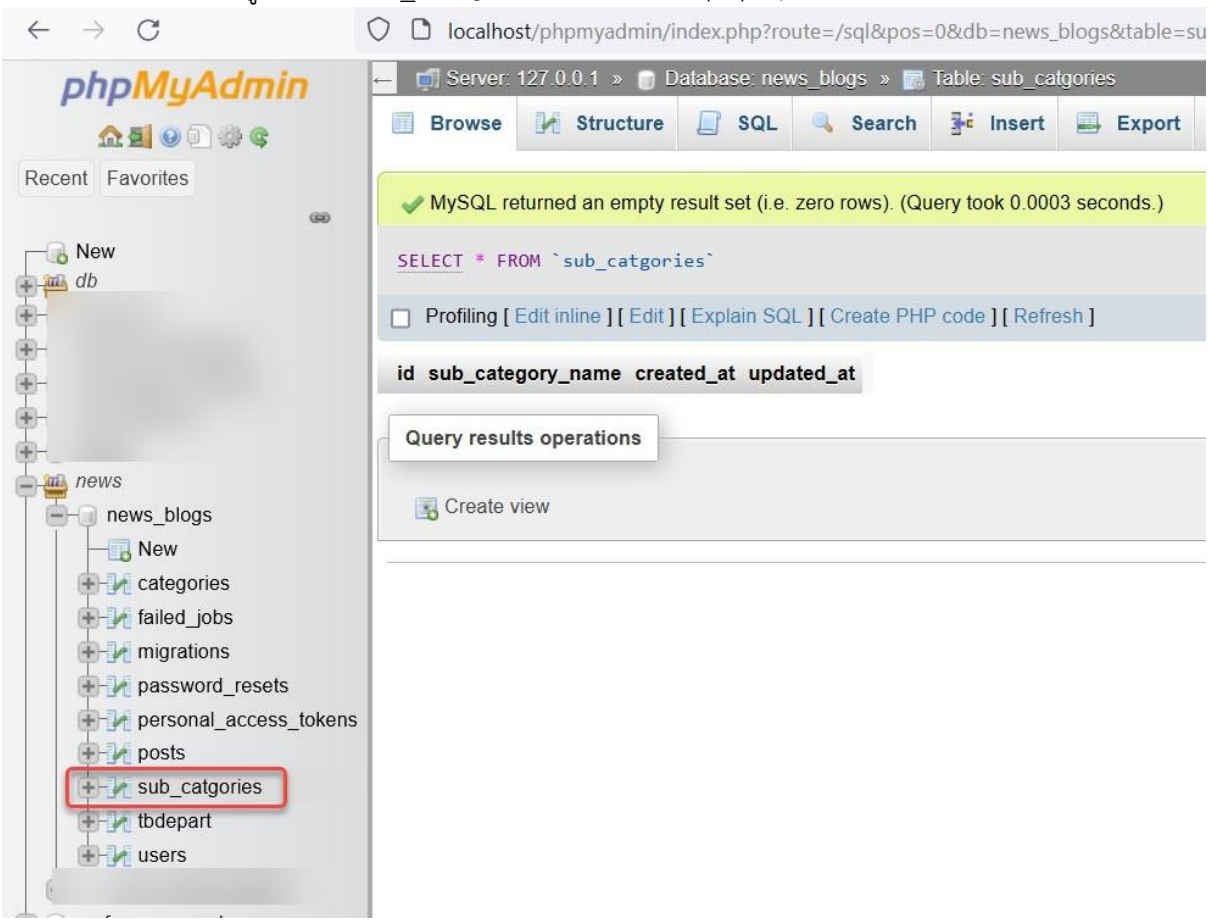


จากตัวอย่างเป็นการสร้างตารางใหม่ขึ้นมาอีก 1 ตารางชื่อ sub_categories ประกอบด้วยฟิลด์ id , sub_category_name , created_at , updated_at

บันทึกไฟล์แล้วพิมพ์คำสั่ง php artisan migrate:refresh เพื่อลบตารางและสั่งให้รัน migration ใหม่ และกด Enter



ตรวจสอบข้อมูลตาราง sub_categories ใน โปรแกรม phpMyAdmin



จะมีตาราง sub_catgories ที่ถูกสร้างขึ้นตามโครงสร้างที่ได้กำหนดค่าไว้

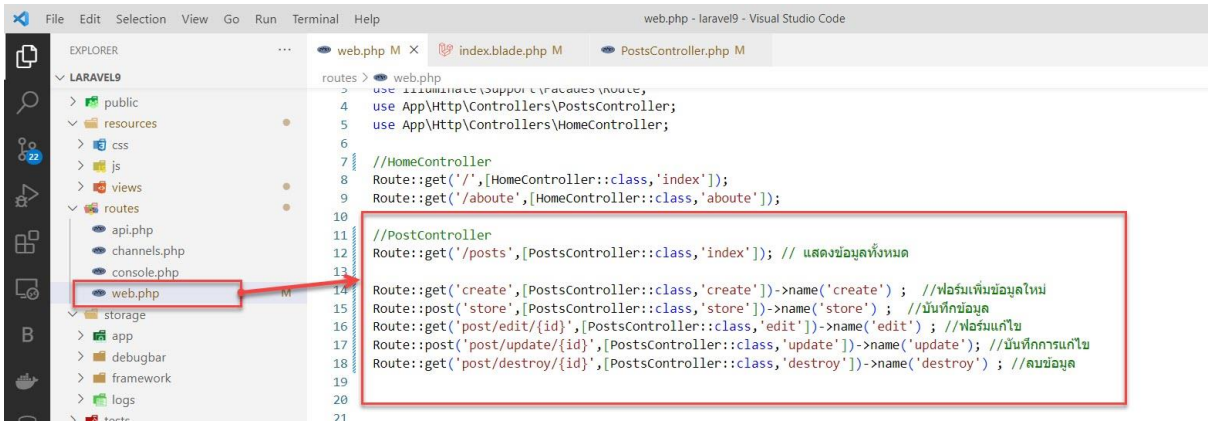
DATABASE RAW SQL

เป็นการเขียนคำสั่ง SQL ยัดเข้าไปในฟังก์ชันต่าง ๆ เพื่อจัดการข้อมูลในตาราง เช่น การเพิ่มข้อมูลลงตาราง การอ่านข้อมูลจากตาราง การแก้ไขข้อมูลในตาราง และ การลบข้อมูลออกจากตาราง

เพิ่มข้อมูลลงตาราง (DB::insert)

หัวข้อนี้จะเป็นการแนะนำวิธีการเพิ่มข้อมูลลงในตารางแบบพื้นฐานโดยการใช้คำสั่ง แบบ RAW SQL เพื่อให้เราทำความเข้าใจการทำงานของแต่ละส่วนและยังเป็นทบทวนบทเรียนที่ผ่าน ๆ มา ด้วย เพื่อให้ผู้อ่านเข้าใจกระบวนการของ Laravel ลองมาสร้างเว็บฟอร์ม เพื่อใช้กรอกข้อมูลผ่านหน้าเว็บแล้ว เพื่อบันทึกลงบนฐานข้อมูล เริ่มเลยครับ

1.เปิดไฟล์ routes/web.php เพื่อเพิ่มเส้นทางในการเข้าใช้งาน (สร้าง URL) เพิ่มโค้ด ดังนี้



ให้สร้างไว้ตามภาพก่อนนะครับ ในการสร้าง route เราจะให้ชี้ไปยัง Controller ชื่อ PostsController ซึ่งเราได้สร้างไว้ก่อนหน้านั้นแล้ว ถ้ายังไม่มีการให้สร้างเพิ่ม และให้ไปทำงานที่เมธอดนั้น ๆ

อธิบาย การสร้าง route

บรรทัดที่ 14 เป็นการสร้างเส้นทาง URL ชื่อว่า create เพื่อแสดงหน้าจอแบบฟอร์มให้กรอกข้อมูล โดยใช้ HTTP Verb แบบ GET และตั้งชื่อ route ว่า create

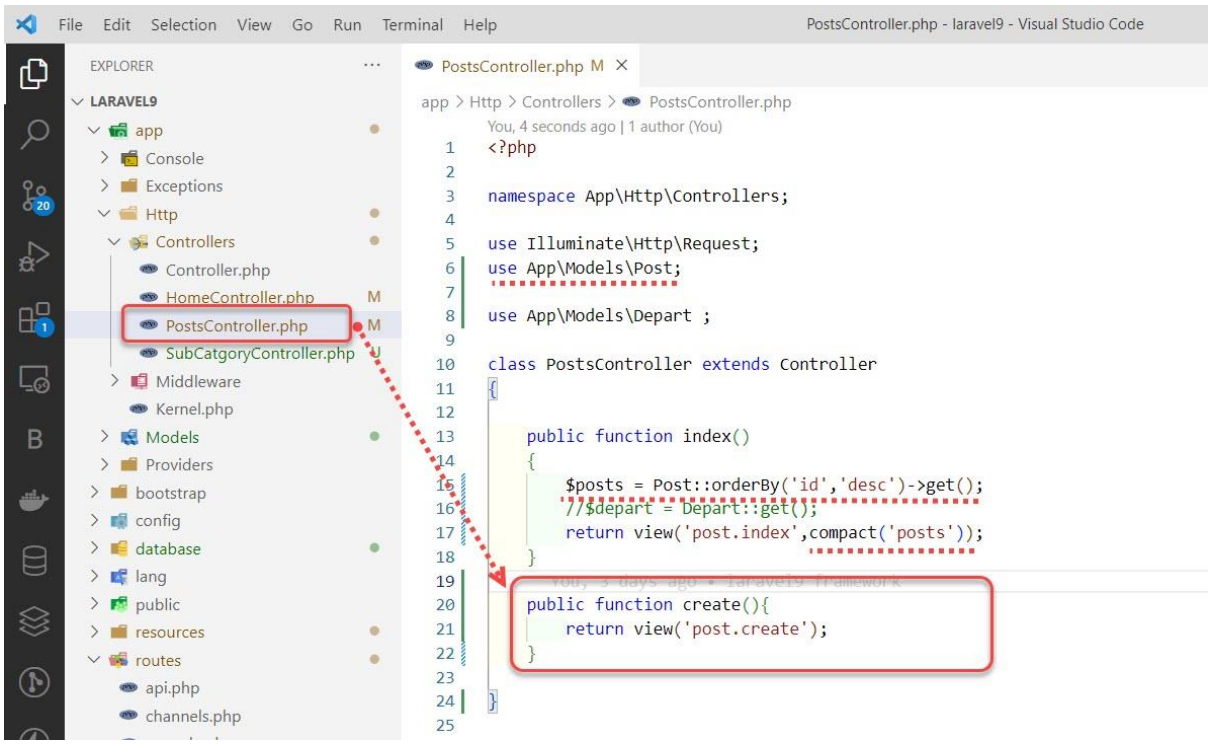
บรรทัดที่ 15 เป็นการสร้างเส้นทาง URL ชื่อว่า store สำหรับรับ Request จากฟอร์มแล้วบันทึกลงตาราง โดยใช้ HTTP Verb แบบ POST และตั้งชื่อ route ว่า store

บรรทัดที่ 16 เป็นการสร้างเส้นทาง URL ชื่อว่า post/edit/ และมีการป้อนค่า parameter เข้ามาด้วย คือ {id} โดยจะดึงข้อมูลการแก้ไข 1 รายการ โดยใช้ HTTP Verb แบบ GET และตั้งชื่อ route ว่า edit

บรรทัดที่ 17 เป็นการสร้างเส้นทาง URL ชื่อว่า post/update/ และมีการป้อนค่า parameter เข้ามาด้วย คือ {id} สำหรับรับ Request จากฟอร์มแล้วแก้ไขข้อมูลบันทึกลงตาราง โดยใช้ HTTP Verb แบบ PUT หรือจะใช้ POST ก็ได้ และตั้งชื่อ route ว่า update

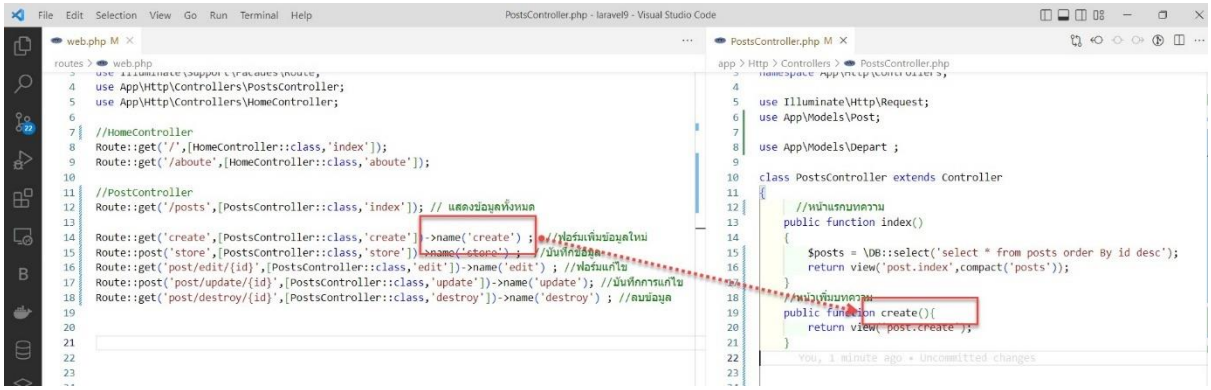
บรรทัดที่ 18 เป็นการสร้างเส้นทาง URL ชื่อว่า post/destroy/ และมีการป้อนค่า parameter เข้ามาด้วย คือ {id} สำหรับลบข้อมูลออกจากตาราง โดยใช้เมธอด get เป็น HTTP Verb แบบ GET หรือ POST ก็ได้ และตั้งชื่อ route ว่า destroy

2.เปิดไฟล์ PostsController.php ในโฟลเดอร์ app\Http\Controllers แล้วเขียน เมธอด ชื่อว่า create ดังนี้



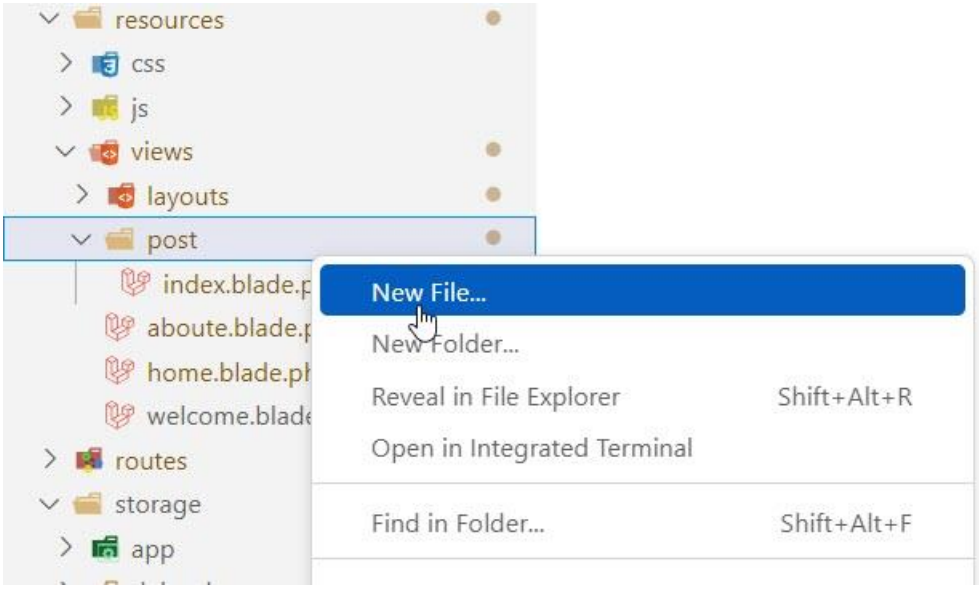
สร้างเมธอด create() โดยจะไปเรียกส่วน views ชื่อ post.create มาแสดงผล คือหน้าฟอร์มสำหรับการกรอกข้อมูลนั่นเอง

จุดสังเกต เมธอดที่ตั้งขึ้นมานั้นจะต้องสอดคล้องกับการเขียน route แนะนำว่าควรตั้งชื่อให้เหมือนกันกับ route

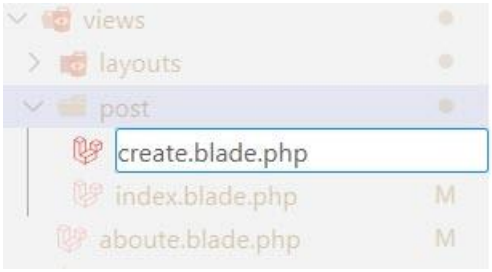


3.สร้างไฟล์ resources/views/post/create.blade.php เพื่อแสดงฟอร์มเพิ่มข้อมูล ดังนี้

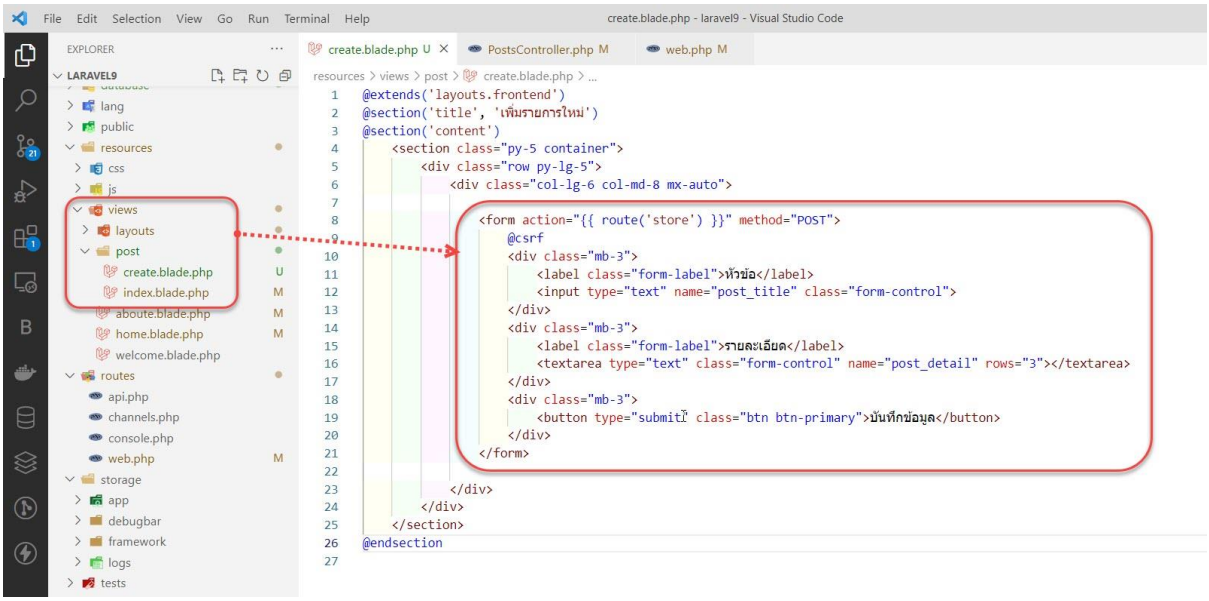
คลิกขวาที่โฟลเดอร์ post แล้วเลือก New File...



4. ตั้งชื่อ views พิมพ์ create.blade.php แล้วกด Enter (การสร้าง views ใช้นามสกุลไฟล์ .blade.php เสมอ)



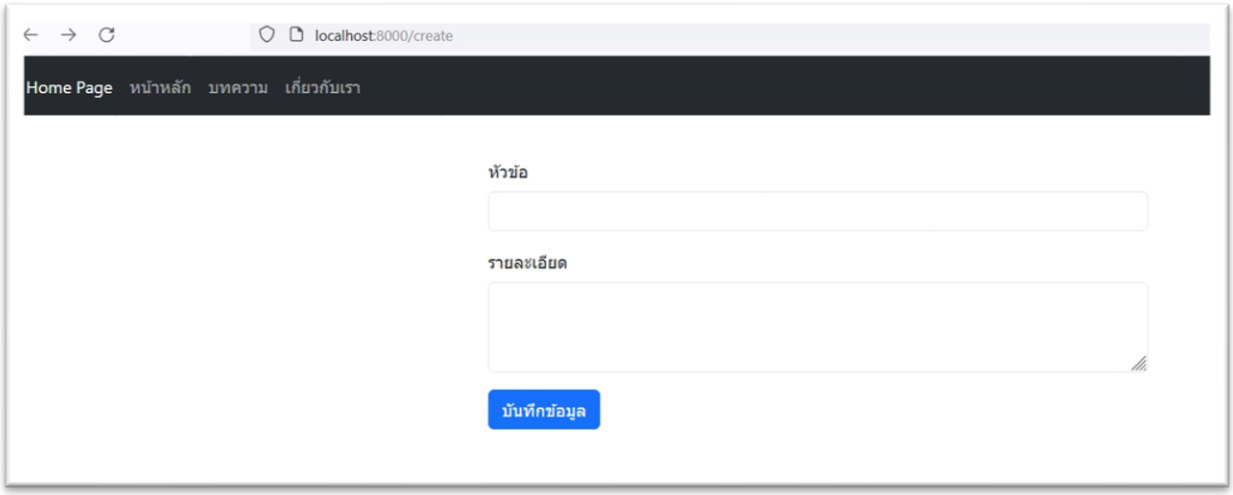
5.เปิดไฟล์ create.blade.php แล้วพิมพ์โค้ด HTML สำหรับสร้างฟอร์มกรอกข้อมูล ดังนี้



การสร้างฟอร์มกรอกข้อมูลเราจะต้องเปิด tag <form> และปิด tag </form> ด้วยเสมอ ใน Laravel จะมีใช้ CSRF หลังเปิดฟอร์มทุกครั้ง ช่วยเพิ่มความ secure โดยการเพิ่ม @csrf ใน form

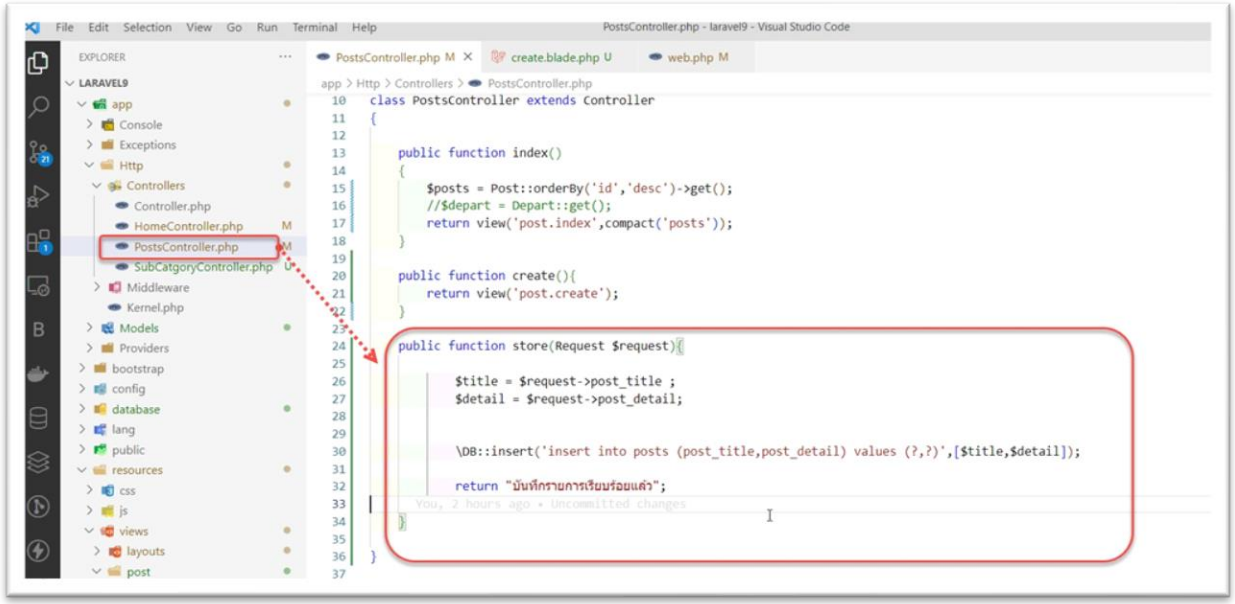
```
<form>
@csrf   // <input type="hidden" name="_token" value="{{ csrf_token()">
....
</form
```

6.บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/create ที่ Browser



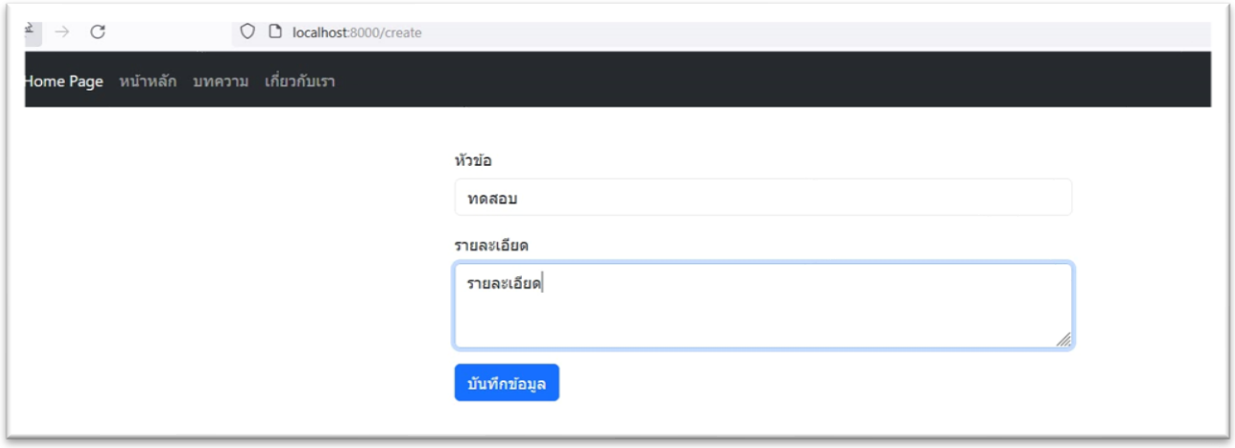
จะได้แบบฟอร์มสำหรับกรอกข้อมูล ตอนนี้เราสามารถกรอกข้อมูลลงในช่องแต่ละรายการได้แล้ว เมื่อกดปุ่มบันทึกข้อมูลก็จะยังไม่มีอะไรเกิดขึ้น ข้อมูลที่เรากรอก จะยังไม่สามารถเพิ่มลงไปในตาราง posts ได้

7.เปิดไฟล์ PostsController.php ในโฟลเดอร์ app\Http\Controllers แล้วเพิ่ม เมธอด ชื่อว่า store ต่อจากเมธอด create และเขียนโค้ด ดังนี้

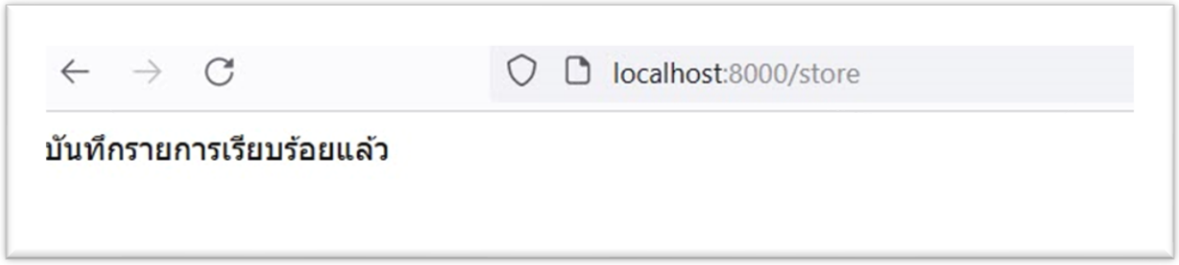


เมธอด store() จะทำหน้าที่รับค่าที่ถูกส่งมาจากฟอร์ม โดยใช้ dependency injection ที่ชื่อว่า Request แล้วใช้ตัวแปร $request->input('....') เพื่อระบุ input ข้อมูลที่ต้องการจะรับเข้ามา ในตัวอย่างเราจะรับค่า input จากฟอร์ม 2 ค่า คือ post_title และ post_detail สามารถรับแบบ $request->post_title ก็ได้ จากนั้นก็เขียนคำสั่งสำหรับให้บันทึกข้อมูลลงในตาราง เมื่อบันทึกข้อมูลแล้วจะมีข้อความแจ้งเตือนว่าบันทึกรายการเรียบร้อยแล้ว

7.บันทึกไฟล์แล้วพิมพ์ localhost:8000/create ที่ Browser จะแสดงหน้าจอให้กรอกข้อมูลใหม่ ให้กรอกข้อมูลอีกครั้ง

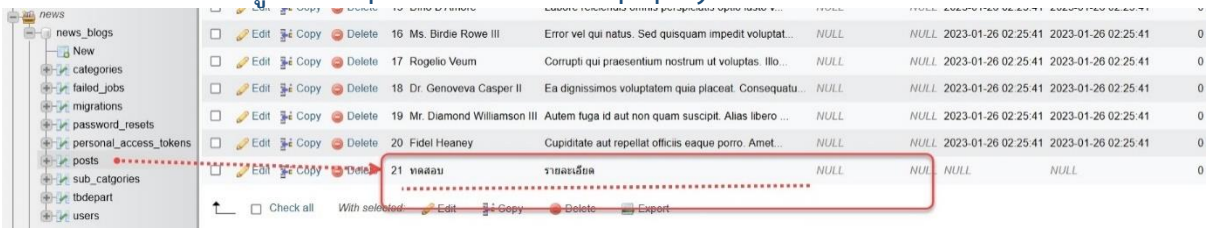


กดปุ่มบันทึกข้อมูล



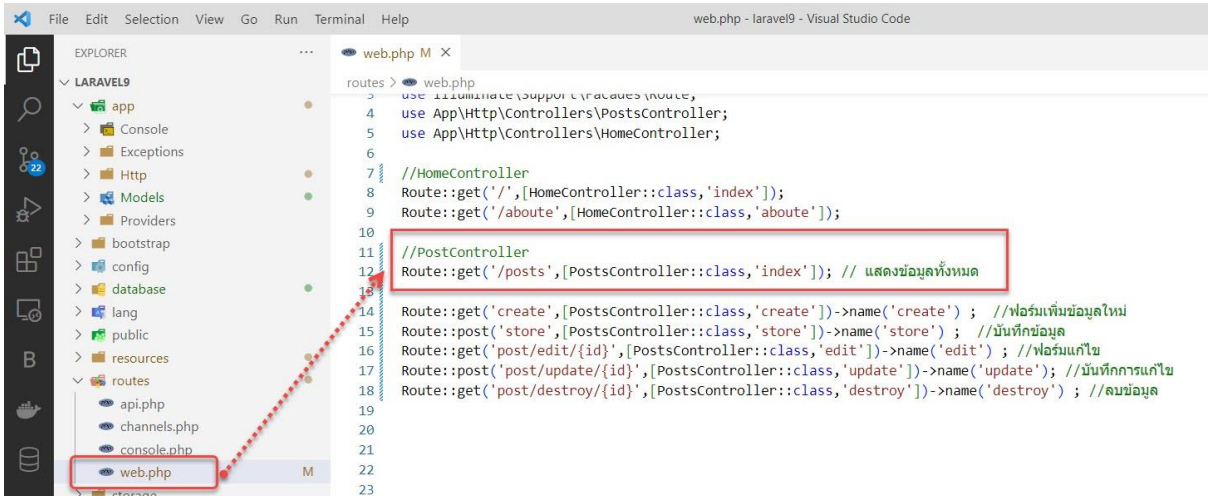
เมื่อไม่มีข้อผิดพลาดใด ข้อมูลที่กรอกผ่านฟอร์มก็จะถูกเพิ่มเข้าไปในตาราง posts แล้ว

ตรวจสอบข้อมูลตาราง posts ใน โปรแกรม phpMyAdmin

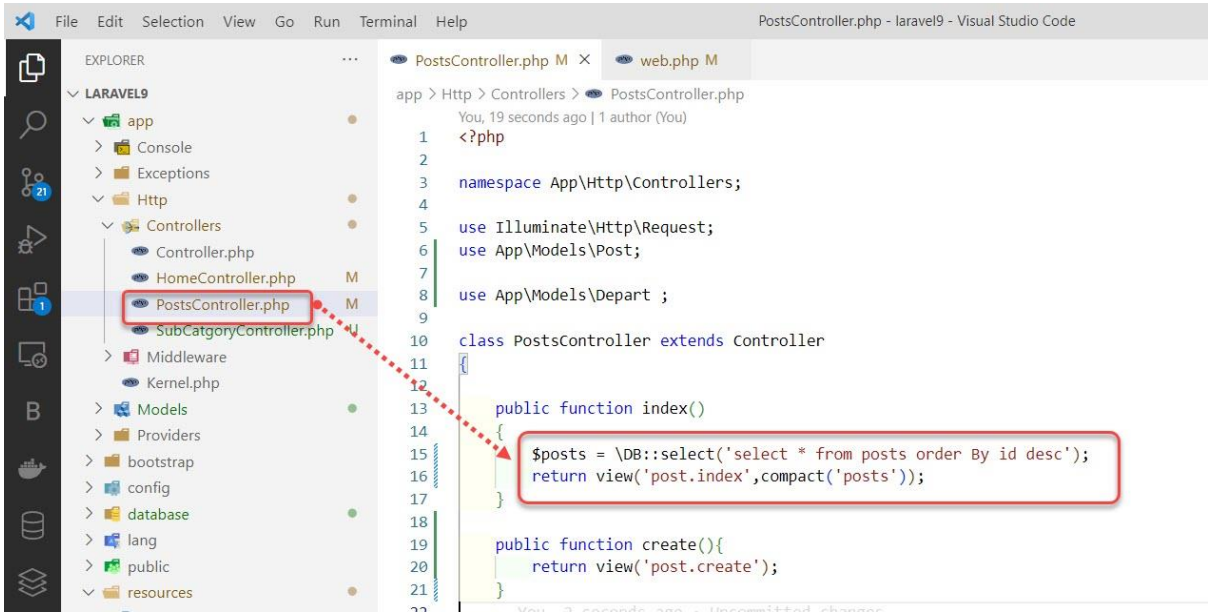


อ่านข้อมูลจากฐานข้อมูล (DB::select)

เมื่อมีการเพิ่มข้อมูลลงในตารางแล้ว เราอยากจะดึงข้อมูลนั้นมาแสดงผลบนหน้าเว็บ ตามเงื่อนไขที่เราอยากแสดงนั้น เป็นสิ่งที่จำเป็น การแสดงผลจะเรียกใช้ URL posts ซึ่งเราได้สร้างไว้ก่อนหน้านั้นแล้ว



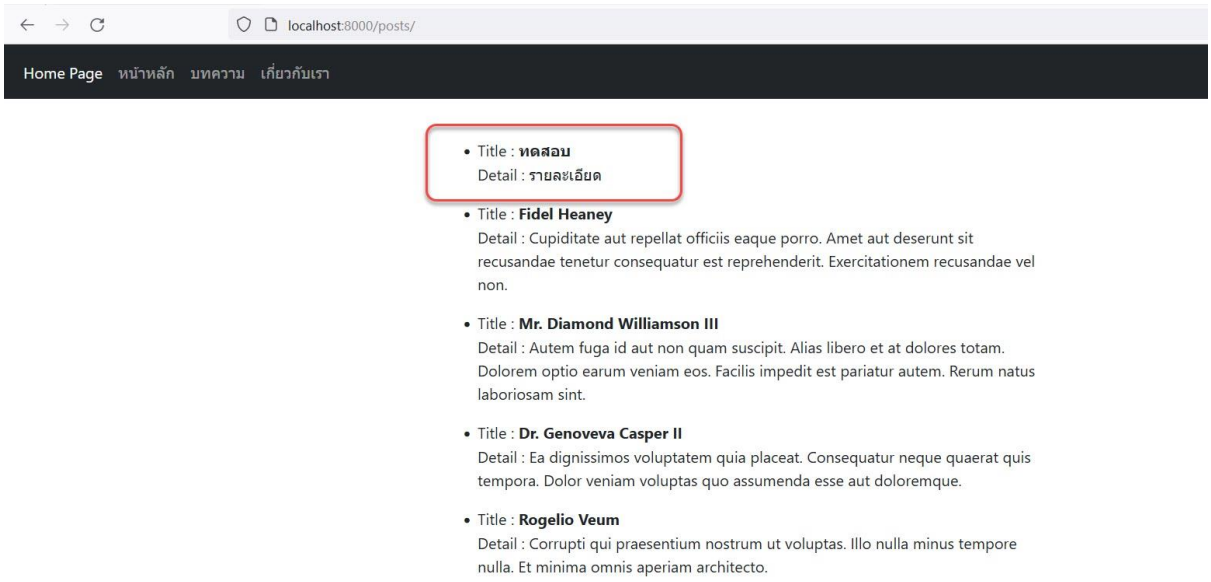
1.เปิดไฟล์ PostsController.php ในโฟลเดอร์ app\Http\Controllers แล้วแก้ไขโค้ดในเมธอด index() ดังนี้



2.เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources/views/post/... แล้วแก้ไขโค้ด ดังนี้



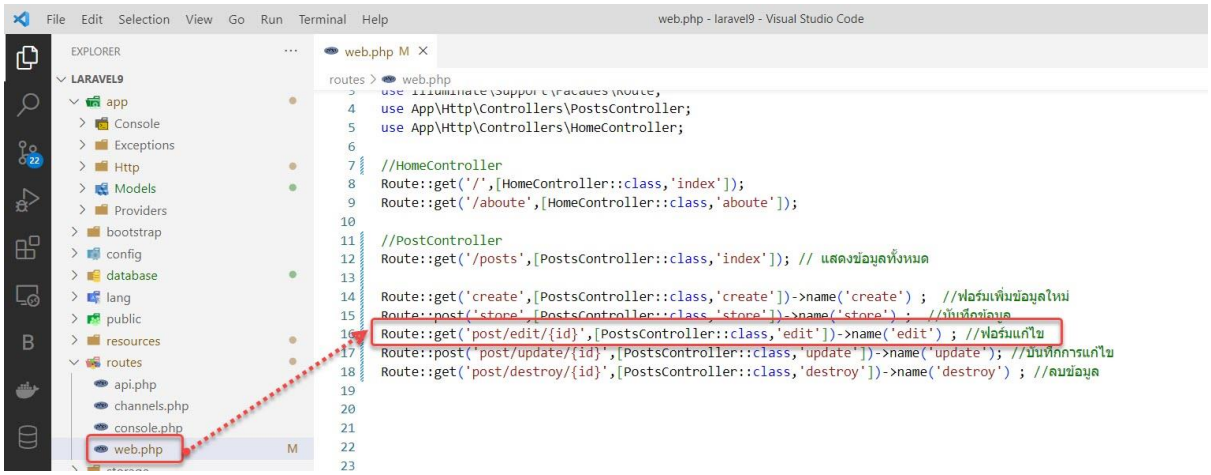
3.บันทึกไฟล์แล้วพิมพ์ localhost:8000/posts ที่ Browser



แก้ไขข้อมูล (DB::select)

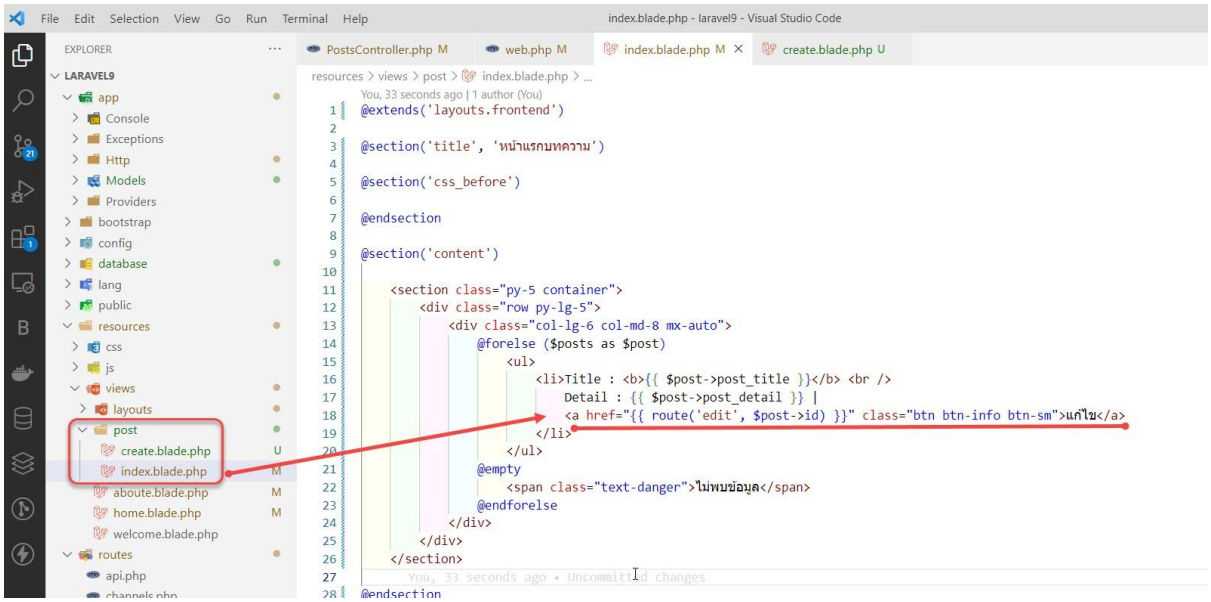
การแก้ไขข้อมูลในตาราง เราจะสร้างฟอร์มขึ้นมาเหมือนกันกับฟอร์มเพิ่มข้อมูลเลย แต่การแก้ไขต้องมีการส่งค่ารายการที่ต้องการที่จะแก้ไขไปด้วย ดังนั้นการสร้างฟอร์มแก้ไขเราก็ต้องดึงข้อมูลเก่าขึ้นมาแสดงในฟอร์มด้วย เพื่อให้เป็นค่าเริ่มต้นให้ผู้ใช้งานแก้ไข เริ่มต้นด้วยการเขียน routes เพื่อกำหนดเส้นทางว่าจะให้ไปทำงานที่ URL ไหนหรือหน้าเว็บไหนเพื่อที่เรียกหน้าฟอร์มสำหรับการแก้ไขขึ้นมา

ในตัวอย่างนี้ เราได้มีการเขียน routes ไว้แล้ว ในไฟล์ routes/web.php ดังนี้

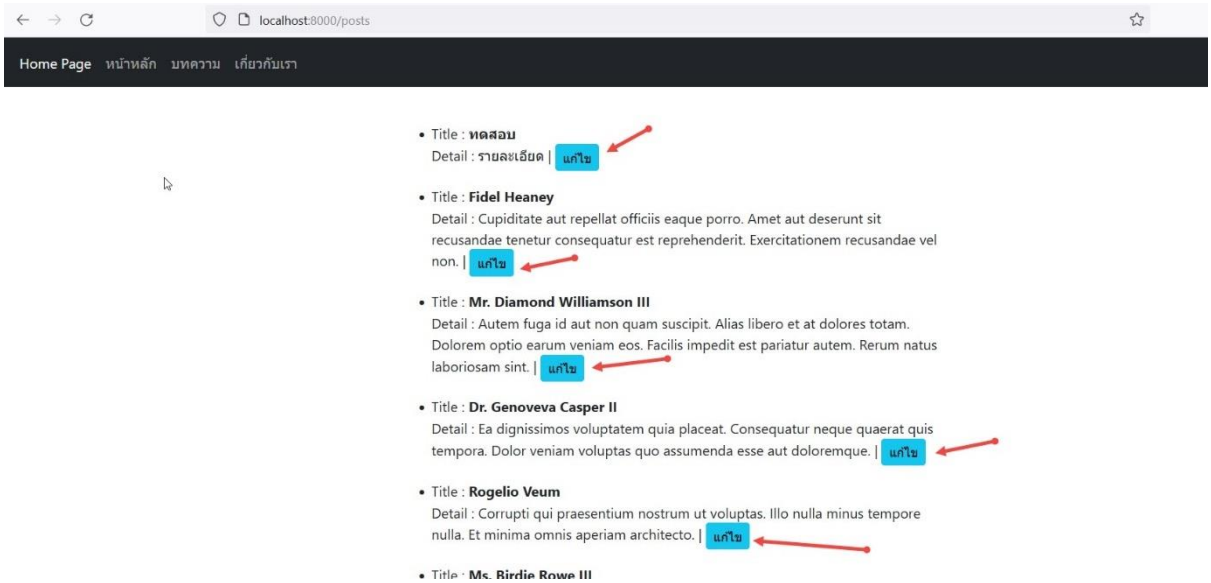


ในบรรทัดที่ 16 เราได้กำหนด URL ชื่อ post/edit/{id} เชื่อมต่อกับ PostsController แล้วไปทำงานที่ เมธอด edit()

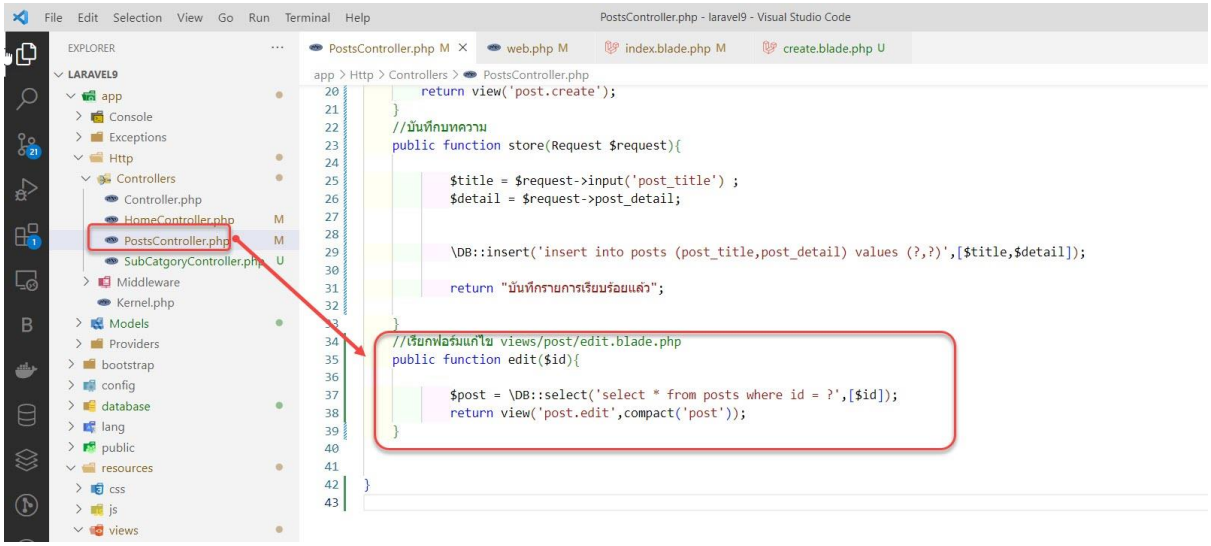
1.ให้เพิ่มลิงค์สำหรับแก้ไขที่ไฟล์ index.blade.php ในโฟลเดอร์ views/post/...แล้วเพิ่มโค้ดเข้าไปดังนี้



บันทึกไฟล์แล้วพิมพ์ localhost:8000/posts ที่ Browser ก็จะแสดงปุ่มสำหรับการแก้ไขของแต่ละรายการ ตามภาพ



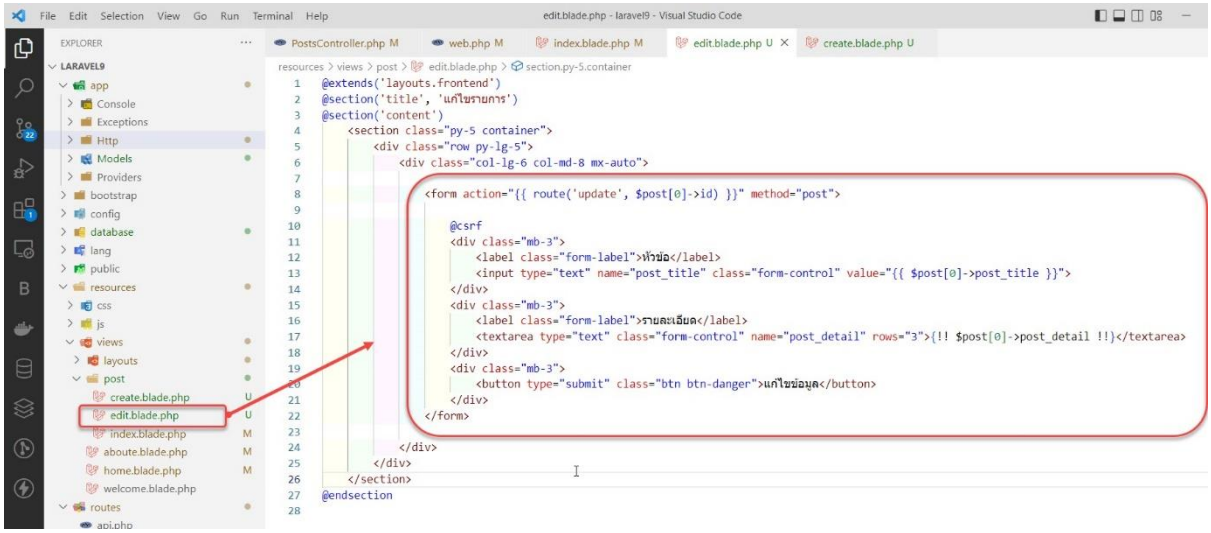
2.เปิดไฟล์ PostsController.php ในโฟลเดอร์ app\Http\Controllers แล้วสร้าง เมธอด ชื่อว่า edit() ต่อจากเมธอด store() เลยครับ แล้วเพิ่มโค้ด ดังนี้



บรรทัดที่ 35 สร้างเมธอด edit() ขึ้นมาและมีการรับค่า $id ของรายการข้อมูลที่ต้องการแก้ไขเข้ามาด้วย แล้วเขียนโค้ดกำหนดเงื่อนไขการดึงข้อมูลจากตารางเก็บไว้ที่ตัวแปร $post จากนั้นก็ส่งข้อมูลไปให้ view แสดงผลตามเงื่อนไข

3.สร้างไฟล์ edit.blade.php ที่โฟลเดอร์ post สำหรับแสดงฟอร์มแก้ไขข้อมูล (คลิกขวาที่โฟลเดอร์ post แล้วเลือก New File...) ฟอร์มแก้ไขจะเหมือนกันกับฟอร์มเพิ่มข้อมูลเลยครับ ดังนั้นให้เราไป Copy โค้ดทั้งหมดที่อยู่ในไฟล์ create.balde.php มาวางในไฟล์ edit.blade.php เลยนะครับ จะใช้โค้ดเหมือนกันเลย

แล้วค่อยมาแก้ไขโค้ดเพิ่มเติม ดังนี้

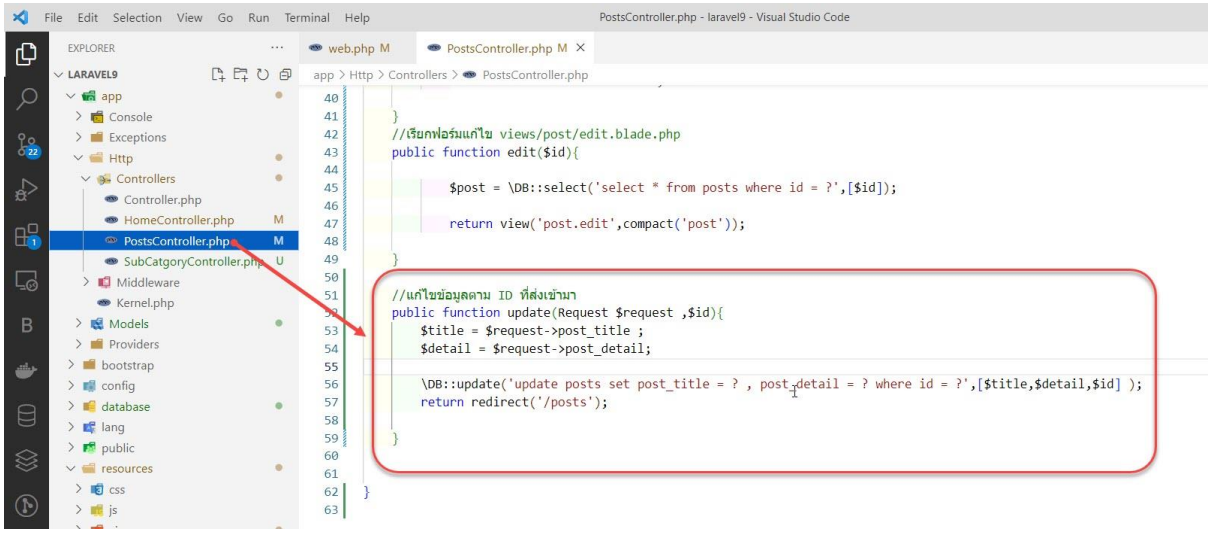


บันทึกไฟล์แล้วพิมพ์ localhost:8000/posts ที่ Browser แล้วคลิกที่ลิงค์แก้ไขข้อมูล



จากตัวอย่างจะเห็นว่ามีการรับค่าข้อมูลจากฐานข้อมูลมาแสดงผลในฟอร์มเพื่อให้ผู้ใช้แก้ไขข้อมูลตามที่ส่งมา โดยในฟอร์มได้มีการรับตัวแปร id เพื่อใช้เป็นรหัสอ้างอิงในการแก้ไขข้อมูลตามรายการที่เราเลือกแก้ไข ภายในฟอร์มได้กำหนด value ให้กับแต่ละ input พร้อมดึงข้อมูลจากฐานข้อมูลมาใส่ในช่อง input และเมื่อมีการกดปุ่มแก้ไขข้อมูล ก็จะถูกส่งไปทำงานที่ routes update พร้อมแนบ id ส่งตามไปด้วยโดยใช้วิธีการส่งแบบ POST

4.เปิดไฟล์ PostsController.php ในโฟลเดอร์ app\Http\Controllers แล้วสร้าง เมธอด ชื่อว่า update() แล้วเพิ่มโค้ด ดังนี้



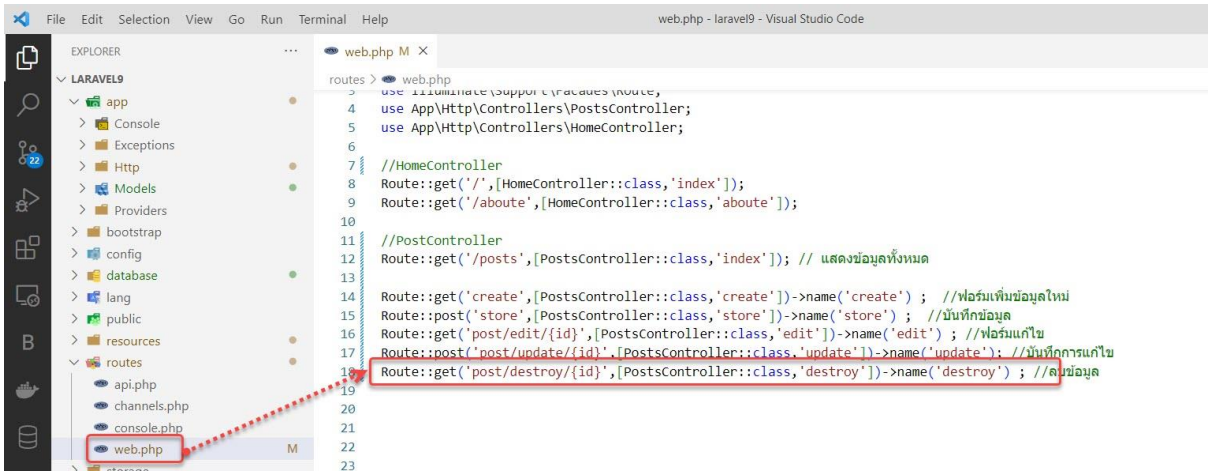
จากตัวอย่างการสร้าง เมธอด update() จะมีการรับค่า $request จากฟอร์ม และรับค่า id และแก้ไขข้อมูลตามรายการ id ที่ส่งเข้ามา เมื่อแก้ไขเสร็จแล้วก็จะทำการ redirect กลับไปที่หน้า /posts ลองทำการแก้ไขแล้วดูผลลัพธ์ดูนะครับ ...

จุดสังเกต การใช้ Request เพื่อรับค่าข้อมูลจากฟอร์ม สามารถใช้ `$request->input('name');` หรือ `$request->name;` ก็ได้

```
$title = $request->input('post_title') ;
$detail = $request->input('post_detail');
-------------------------------------------------
$title = $request->post_title ;
$detail = $request->post_detail;
```

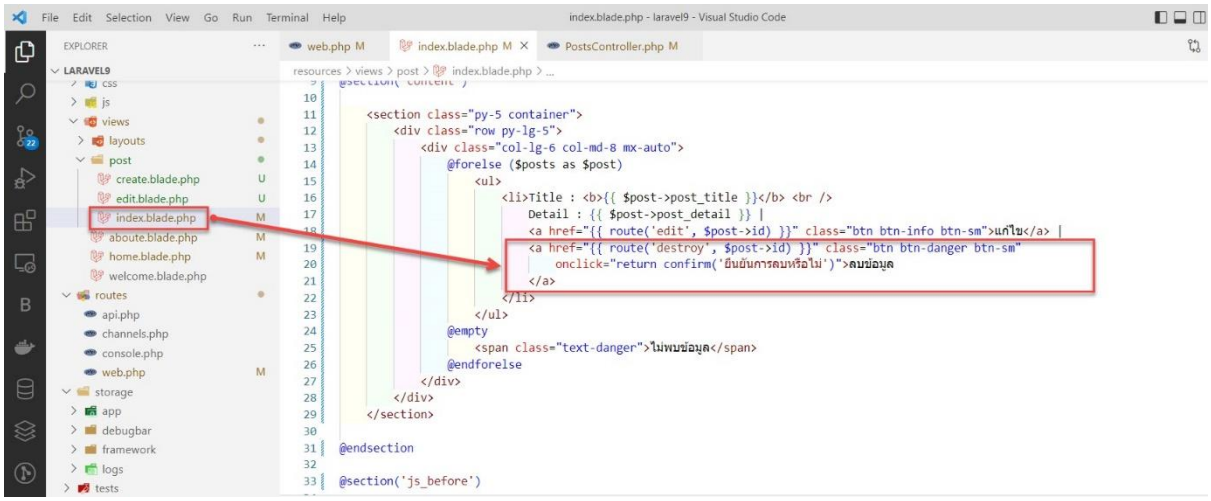
ลบข้อมูล (DB::delete)

1. เราได้มีการเขียน routes ไว้แล้ว ในไฟล์ routes/web.php ดังนี้



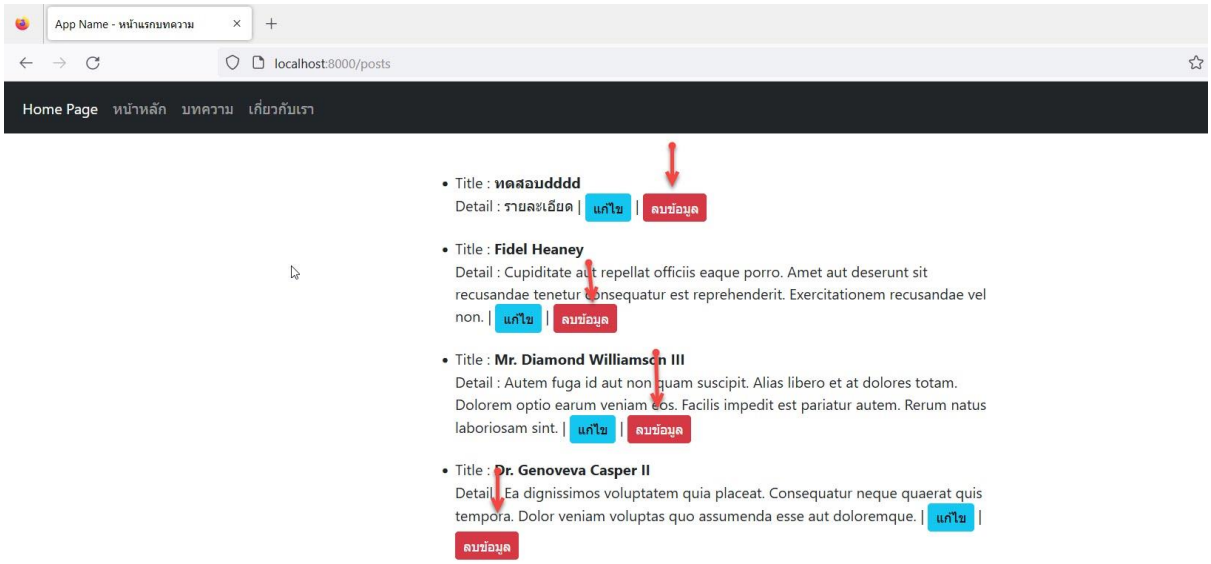
ในบรรทัดที่ 18 เราได้กำหนด Routes แบบ GET URL ชื่อ post/destroy/{id} เชื่อมต่อกับ PostsController แล้วไปทำงานที่ เมธอด destroy()

2. ให้เพิ่มลิงค์สำหรับลบข้อมูลที่ไฟล์ index.blade.php ในโฟลเดอร์ views/post/...แล้วเพิ่มโค้ดเข้าไป ดังนี้

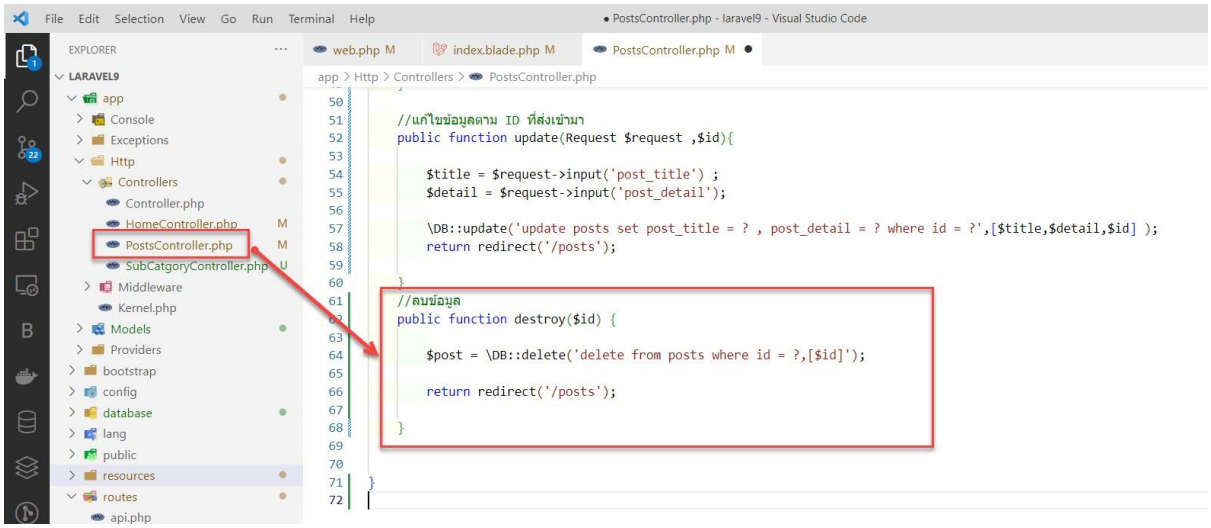


จากตัวอย่างการสร้าง เป็นการเพิ่มข้อความ ลบข้อมูล และเมื่อคลิกที่ลิงค์ ลบข้อมูล ก็จะไปทำงานที่ routes destroy พร้อมส่งค่า id นั้น ๆ ไปด้วย ก่อนการลบจะมีกล่องข้อความแจ้งเตือนว่า ยืนยันการลบหรือไม่ เพื่อเป็นการยืนยันลบข้อมูล

บันทึกไฟล์แล้วพิมพ์ localhost:8000/posts ที่ Browser



3. เปิดไฟล์ PostsController.php ในโฟลเดอร์ app\Http\Controllers แล้วสร้าง เมธอด ชื่อว่า destroy() แล้วเพิ่มโค้ด ดังนี้



จากตัวอย่างการสร้าง เมธอด destroy() จะมีการรับค่า $id จาก URL ที่มีการคลิกในแต่ละรายการที่เราต้องการจะลบ และรับค่า id เข้ามาด้วยเพื่อเป็นเงื่อนไขในการลบ เมื่อลบเสร็จแล้วก็จะทำการ redirect กลับไปที่หน้า /posts ลองทำการลบแล้วดูผลลัพธ์ดูนะครับ ...

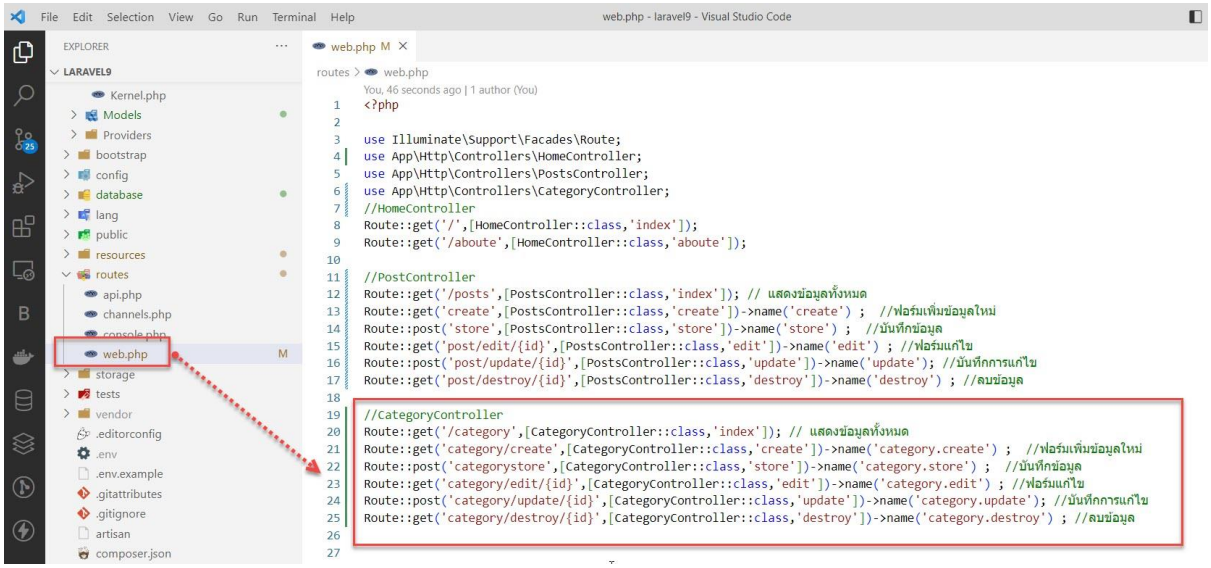
DATABASE Query Builder

Query Builder เป็นชุดคำสั่งที่ Laravel มีมาให้ ในกรณีที่เราไม่อยากเขียน SQL เอง ช่วยให้ลดข้อผิดพลาดในการเขียน SQL ของเรา โดยจะประกอบไปด้วยชุดคำสั่งที่ใช้ในการเพิ่มข้อมูล อ่านข้อมูล แก้ไขข้อมูล ลบข้อมูล ซึ่งจะเป็นชุดคำสั่งสั้น ๆ กระชับ สามารถเขียนคำสั่งได้เพียงไม่กี่บรรทัด

Query Builder (เพิ่มข้อมูล)

ตัวอย่างนี้เป็นการแสดงวิธีการใช้งานคำสั่งเพิ่มข้อมูลอย่างง่ายลงในตาราง categories (ตารางเก็บประเภทบทความ) ซึ่งเราได้มีการสร้างไว้แล้วนั้นในหน้า 103 และมีการกำหนดโครงสร้างตารางในหน้า 105 เราจะทำการเรียนรู้การเพิ่มข้อมูลลงด้วยการใช้คำสั่งแบบ Query Buider การเพิ่มข้อมูลนั้นเราจะมีการสร้างฟอร์มสำหรับให้กรอกข้อมูลเช่นเดียวกันกับการเพิ่มข้อมูลลงตาราง posts โดยให้เราเพิ่มเส้นทางใหม่สำหรับการเข้าใช้งาน ดังนี้

1.เปิดไฟล์ routes/web.php เพื่อเพิ่มเส้นทางใหม่ในการเข้าใช้งาน (สร้าง URL) เพิ่มโค้ด ดังนี้



ให้สร้าง route ตามภาพเลยนะครับ ในการสร้าง route เราจะให้ชี้ไปยัง Controller ชื่อ CategoryController และให้ไปทำงานที่เมธอดนั้น ๆ ซึ่งเราจะต้องไปสร้าง Controller ชื่อ CategoryController และ เมธอด ต่าง ๆ เพิ่มเองวิธีสร้างเดี๋ยวไปดูในหัวข้อที่ 2 นะครับ

อธิบาย การสร้าง route

บรรทัดที่ 20 เป็นการสร้างเส้นทาง URL หลักของหน้าประเภทบทความ เพื่อแสดงหน้าประเภทข้อมูล โดยใช้ HTTP Verb แบบ GET และตั้งชื่อ URL ว่า /category ซึ่งจะไปทำงานที่คอนโทรลเลอร์ชื่อ Category เมธอด index()

บรรทัดที่ 21 เป็นการสร้างเส้นทาง URL ชื่อว่า category/create เพื่อแสดงหน้าจอแบบฟอร์มให้กรอกข้อมูล โดยใช้ HTTP Verb แบบ GET และตั้งชื่อ route ว่า category.create

บรรทัดที่ 22 เป็นการสร้างเส้นทาง URL ชื่อว่า category/store สำหรับรับ Request จากฟอร์มแล้วบันทึกลงตาราง โดยใช้ HTTP Verb แบบ POST และตั้งชื่อ route ว่า category.store

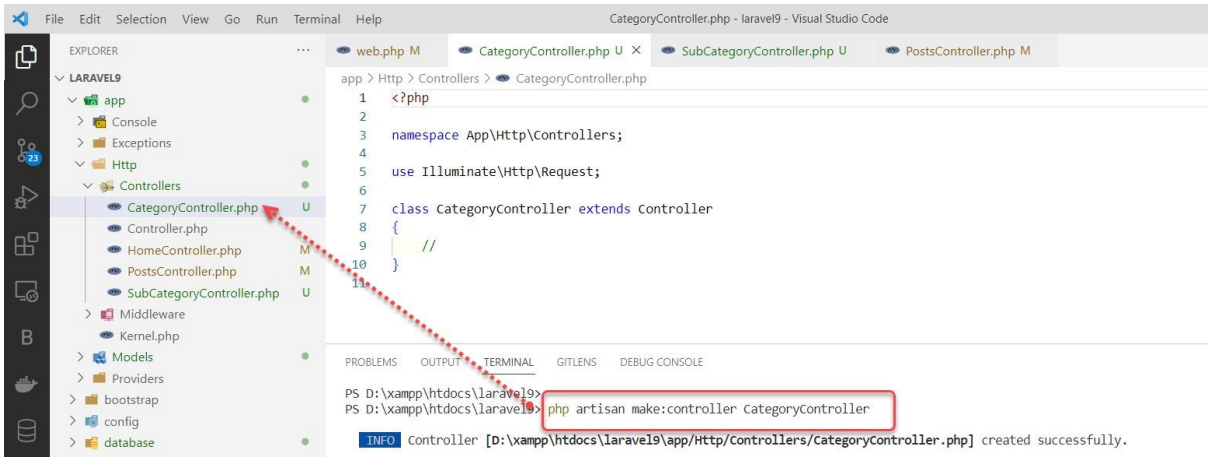
บรรทัดที่ 23 เป็นการสร้างเส้นทาง URL ชื่อว่า category/edit/ และมีการป้อนค่า parameter เข้า

มาด้วย คือ {id} โดยจะดึงข้อมูลการแก้ไข 1 รายการ โดยใช้ HTTP Verb แบบ GET และตั้งชื่อ route ว่า category.edit

บรรทัดที่ 24 เป็นการสร้างเส้นทาง URL ชื่อว่า category/update/ และมีการป้อนค่า parameter เข้ามาด้วย คือ {id} สำหรับรับ Request จากฟอร์มแล้วแก้ไขข้อมูลบันทึกลงตาราง โดยใช้ HTTP Verb แบบ PUT หรือจะใช้ POST ก็ได้ และตั้งชื่อ route ว่า category.update

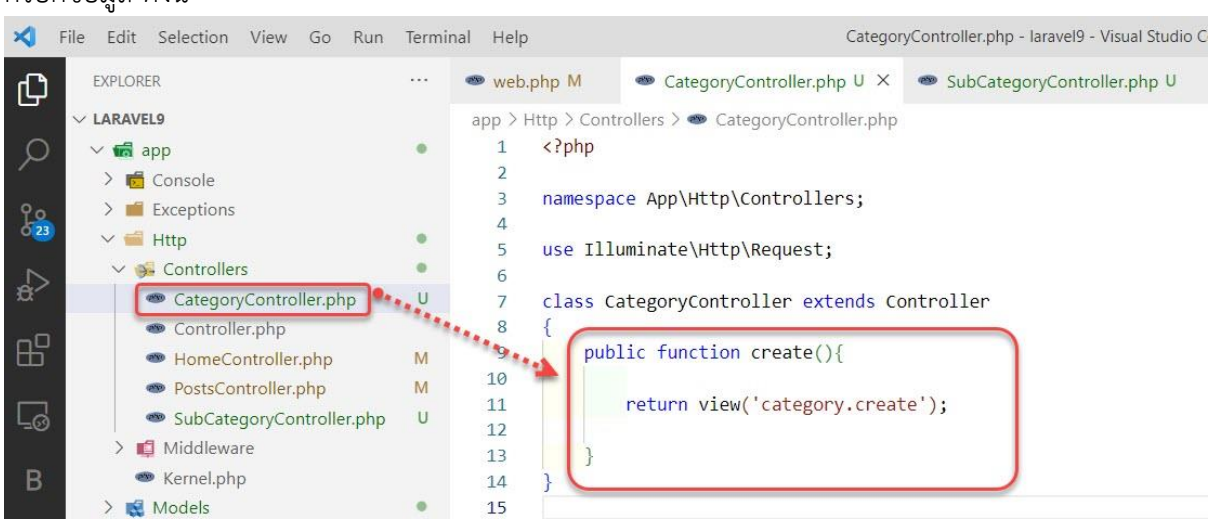
บรรทัดที่ 25 เป็นการสร้างเส้นทาง URL ชื่อว่า category/destroy/ และมีการป้อนค่า parameter เข้ามาด้วย คือ {id} สำหรับลบข้อมูลออกจากตาราง โดยใช้เมธอด get เป็น HTTP Verb แบบ GET หรือ POST ก็ได้ และตั้งชื่อ route ว่า category.destroy

2.สร้าง CategoryController โดยพิมพ์คำสั่งใน Terminal ดังนี้

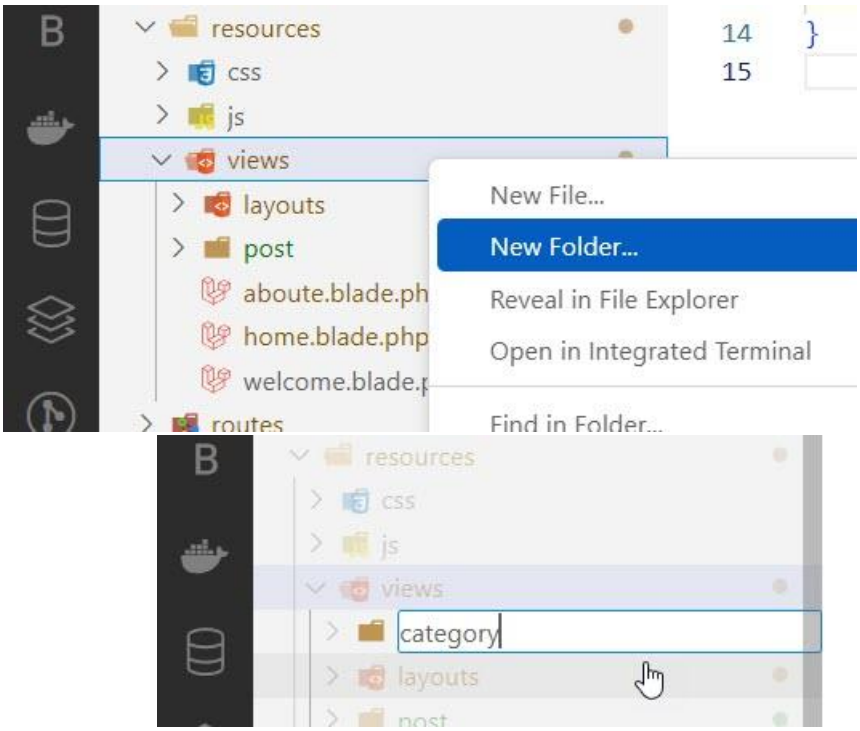


จะได้ไฟล์ CategoryController.php ถูกสร้างขึ้นมาให้แล้ว

3.เปิดไฟล์ CategoryController.php ในโฟลเดอร์ app\Http\Controllers แล้วเพิ่ม เมธอด ชื่อว่า create() โดยจะไปเรียกส่วน views ชื่อ category.create.blade.php มาแสดงผลสำหรับแสดงหน้าฟอร์มให้กรอกข้อมูล ดังนี้

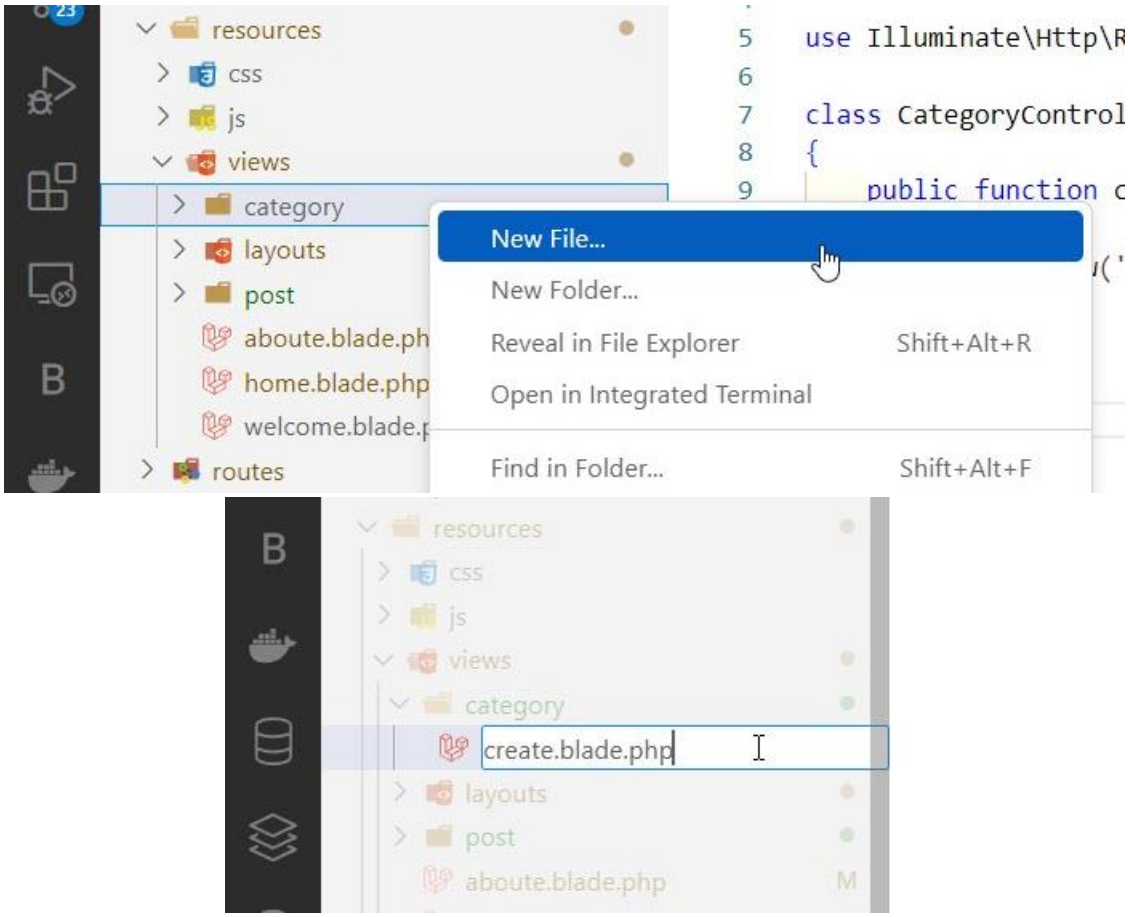


4.สร้างโฟลเดอร์ category เก็บไว้ที่ resources/views/... เพื่อเก็บไฟล์ต่าง ๆ ดังนี้ คลิกขวาที่โฟลเดอร์ views แล้วเลือก New Folder...จากนั้นตั้งชื่อโฟลเดอร์ว่า category

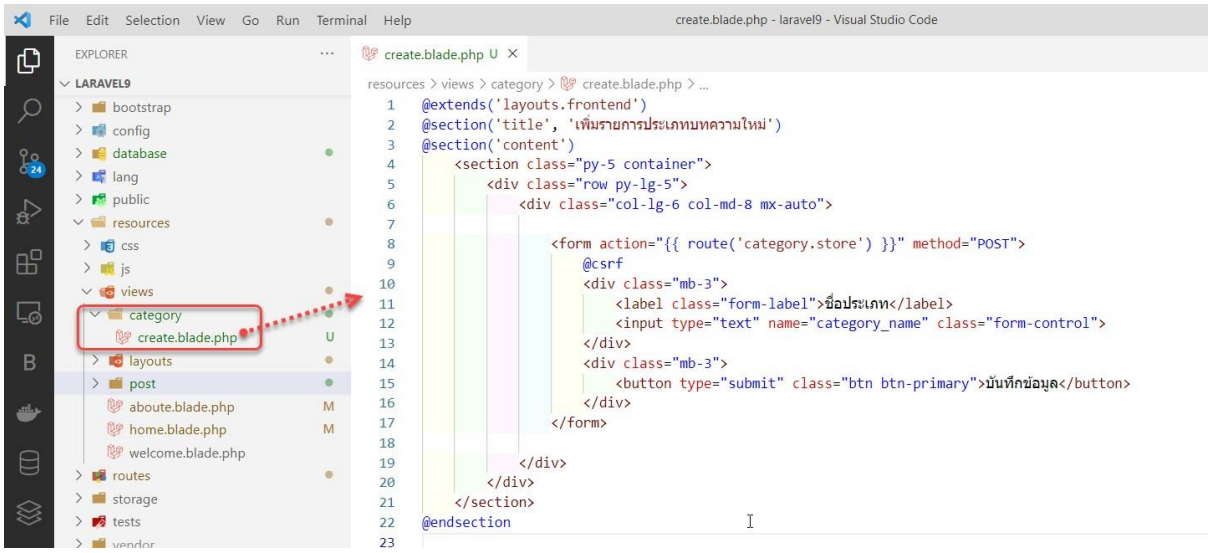


สร้างไฟล์ create.blade.php เก็บไว้ที่ resources/views/category/... ดังนี้

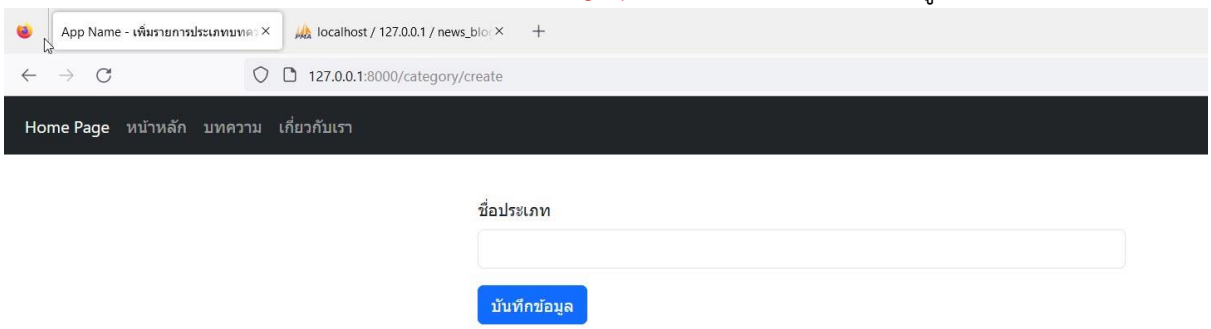
คลิกขวาที่โฟลเดอร์ category แล้วเลือก New File...จากนั้นตั้งชื่อไฟล์ว่า create.blade.php



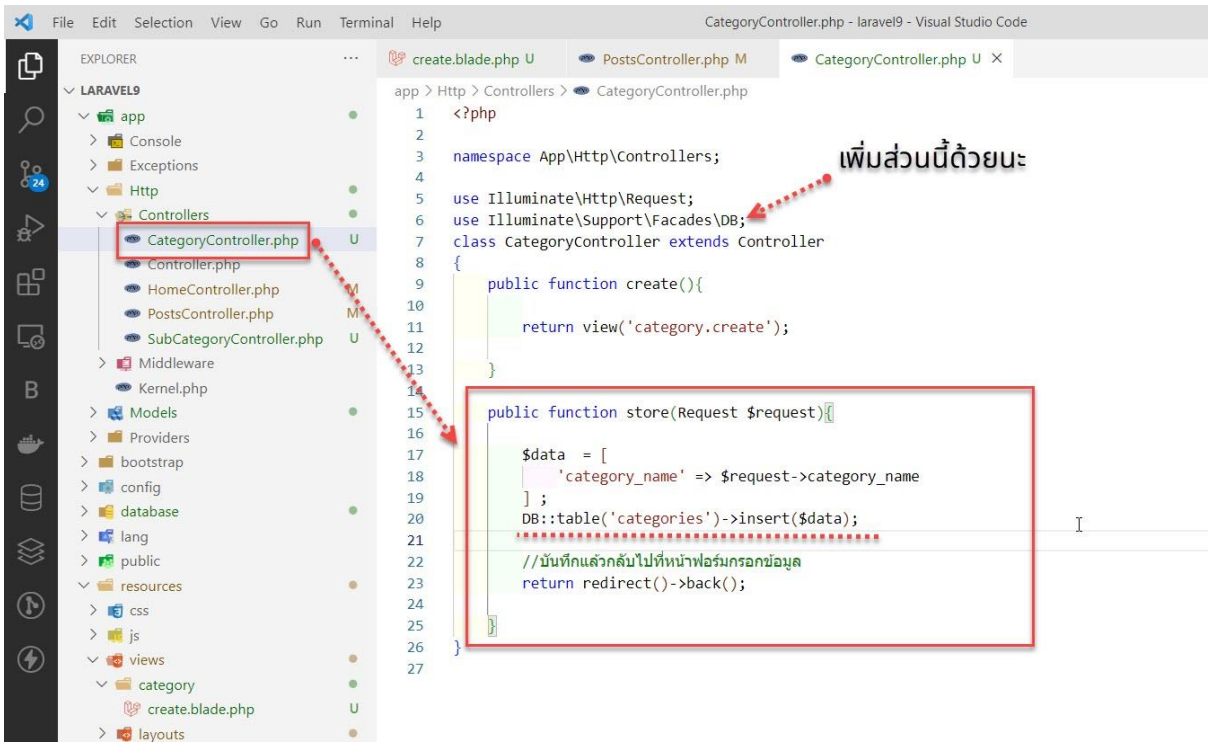
5.เปิดไฟล์ create.blade.php แล้วเพิ่มโค้ด HTML สำหรับสร้างฟอร์มให้กรอกข้อมูล ดังนี้



บันทึกไฟล์แล้วพิมพ์ localhost:8000/category/create ที่ Browser เพื่อดูผลลัพธ์ที่สร้าง

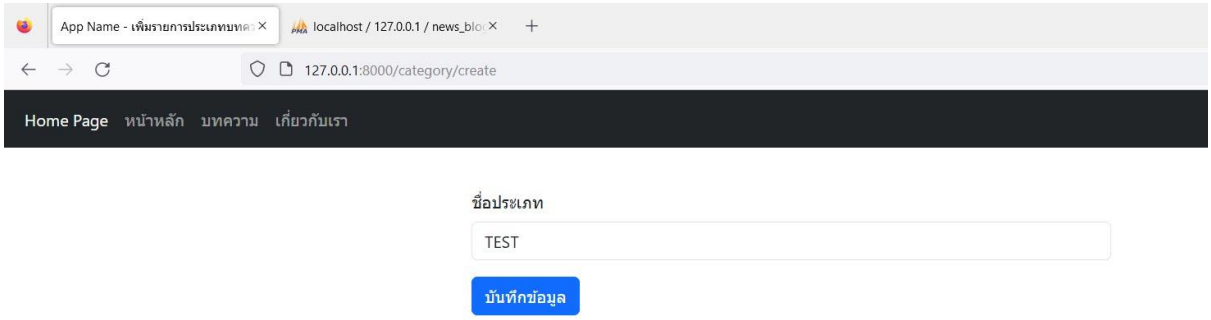


6.เปิดไฟล์ CategoryController แล้วเพิ่มเมธอด store() แล้วเขียนโค้ดเข้าไป ดังนี้

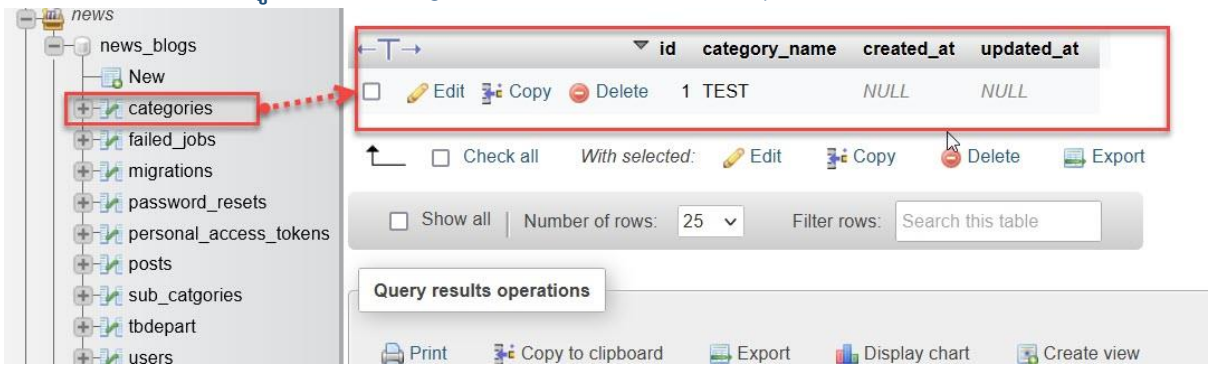


เมธอด store() จะทำหน้าที่รับค่าที่ถูกส่งมาจากฟอร์ม โดยใช้ dependency injection ที่ชื่อว่า Request แล้วใช้ตัวแปร $request->category_name เพื่อระบุ input ข้อมูลที่ต้องการจะรับเข้ามา ในตัวอย่างเราจะรับค่า input จากฟอร์ม 1 ค่า คือ category เก็บไว้ในตัวแปร Array $data จากนั้นก็เขียนคำสั่งสำหรับให้บันทึกข้อมูลลงในตาราง โดยใช้ Query Builder แทน ดูได้จากโค้ดบรรทัดที่ 20 เมื่อบันทึกข้อมูลแล้วจะ redirect กลับไปที่หน้าฟอร์มกรอกข้อมูล

7. บันทึกไฟล์แล้วพิมพ์ localhost:8000/category/create ที่ Browser จะแสดงหน้าจอให้กรอกข้อมูลใหม่ ให้กรอกข้อมูลอีกครั้งแล้วบันทึกดูผลลัพธ์ที่เกิดขึ้น



ตรวจสอบข้อมูลตาราง categories ใน โปรแกรม phpMyAdmin



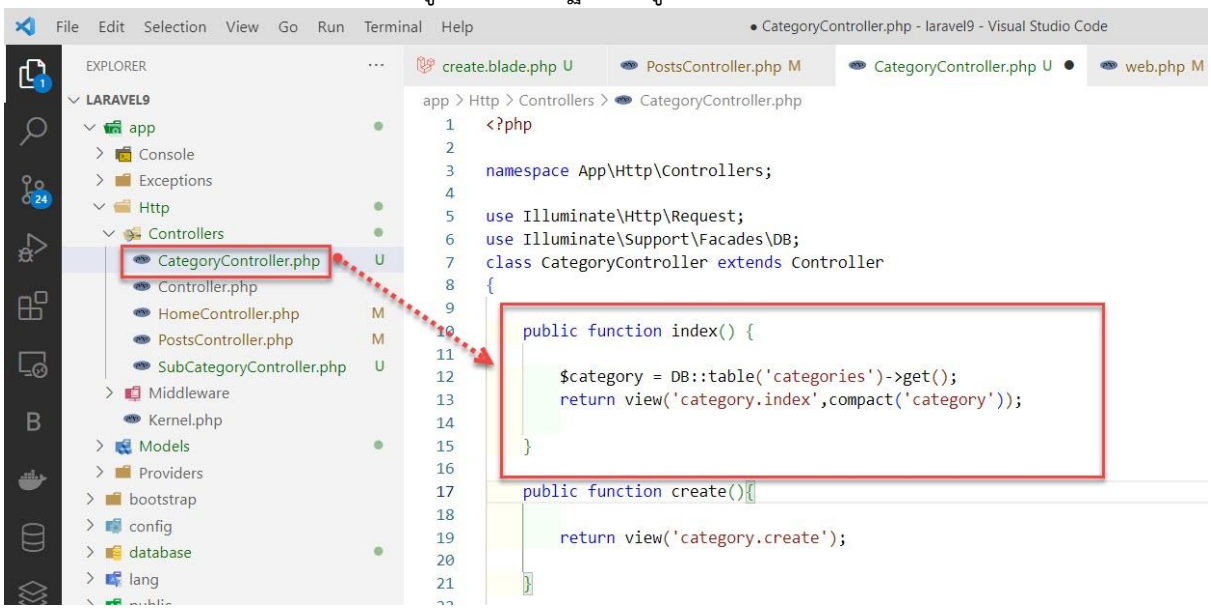
ก็จะเห็นข้อมูลที่กรอกเข้ามาผ่านฟอร์มถูกเพิ่มเข้ามาในตาราง categories เรียบร้อยแล้ว

Query Builder (อ่านข้อมูล)

การอ่านข้อมูลขึ้นมาแสดง จะทำงานที่ routes category ซึ่งได้เขียนโค้ดให้เชื่อมต่อกับคอนโทรลเลอร์ชื่อ CategoryController แล้วไปทำงานที่เมธอด index()



1.เปิดไฟล์ CategoryController.php ในโฟลเดอร์ app\Http\Controllers แล้วเพิ่มเมธอด index() และพิมพ์โค้ดสำหรับแสดงข้อมูลจากตารางฐานข้อมูล ดังนี้



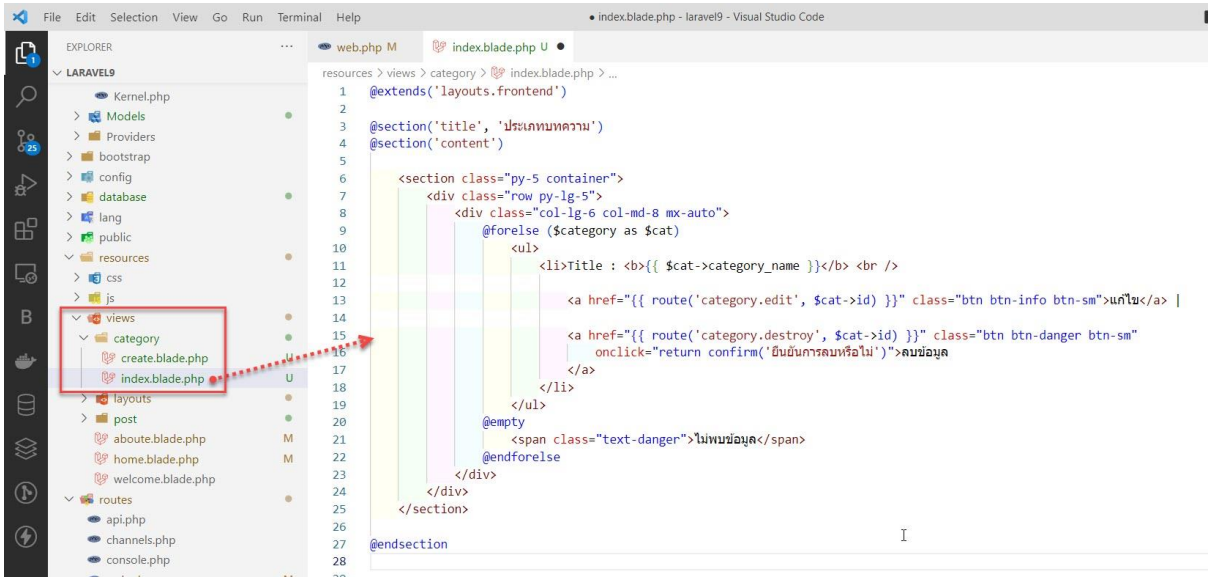
จากตัวอย่าง ได้เพิ่มเมธอดชื่อว่า index() แล้วเขียนคำสั่งโดยใช้รูปแบบ Query Builder ในการดึงข้อมูลจากตาราง categories จากนั้นก็ใช้ฟังก์ชัน get() เพื่อดึงข้อมูลทั้งหมดที่อยู่ในตาราง เก็บไว้ในตัวแปร $category จากนั้นก็ส่งตัวแปรไปให้ views ชื่อ category.index แสดงผล ตอนนี้ยังไม่มีไฟล์ view ชื่อ index ใน โฟลเดอร์ category นะ ต้องไปสร้างก่อน

2. สร้างไฟล์ index.blade.php เก็บไว้ที่ resources/views/category/...

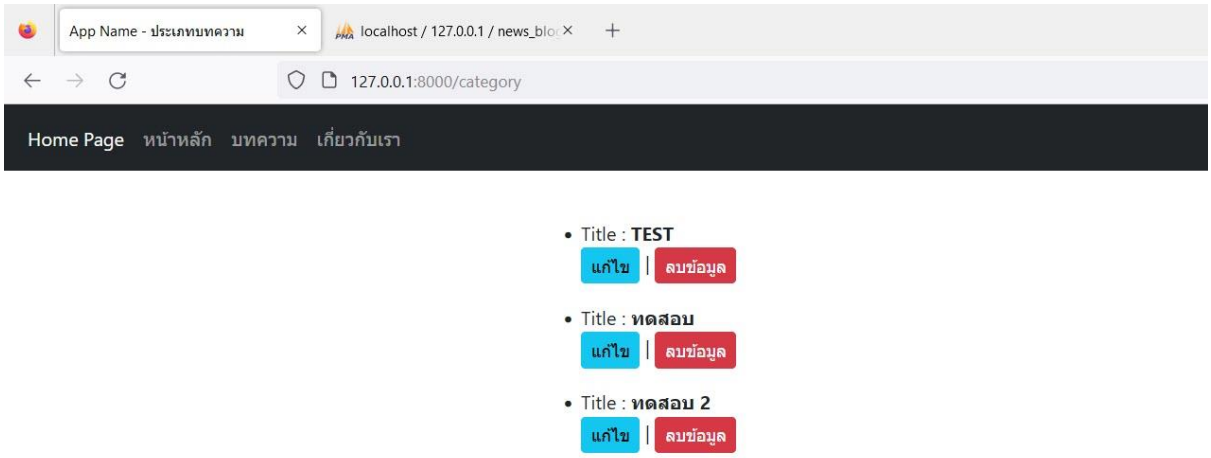
คลิกขวาที่โฟลเดอร์ category แล้วเลือก New File...จากนั้นตั้งชื่อไฟล์ว่า index.blade.php

เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources/views/category/... แล้วเพิ่มโค้ดสำหรับแสดง

ข้อมูลจากฐานข้อมูล ดังนี้

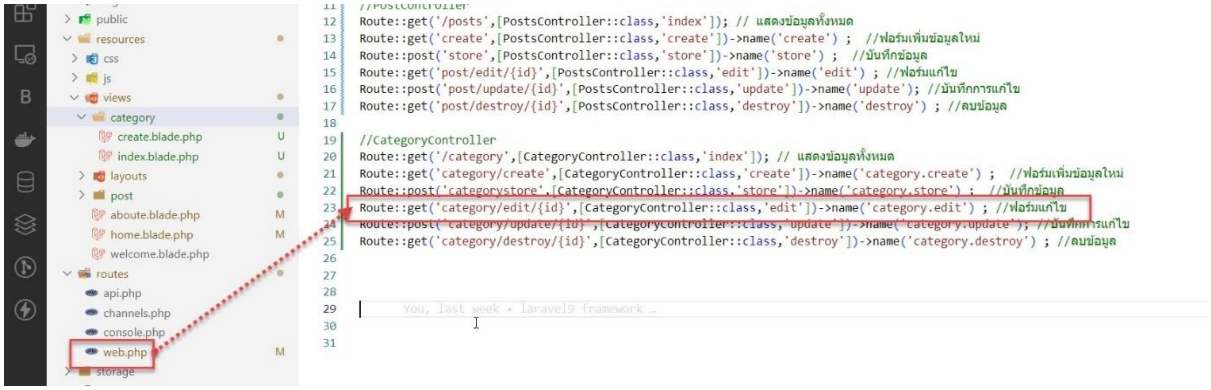


3.บันทึกไฟล์แล้วพิมพ์ localhost:8000/category ที่ Browser



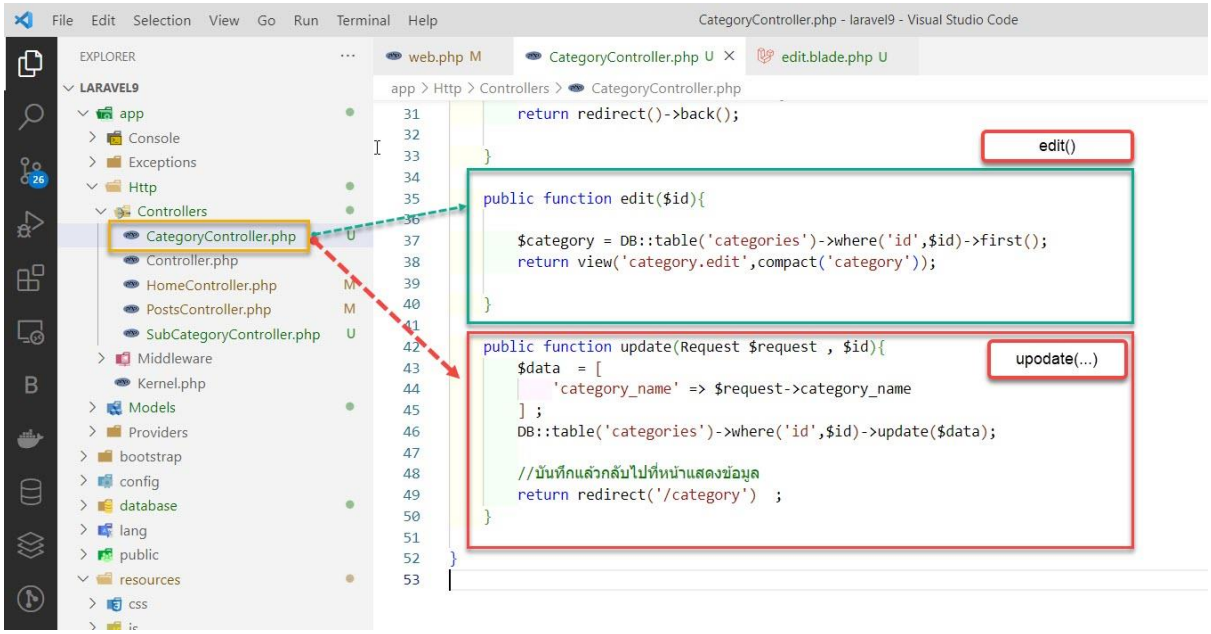
Query Builder (แก้ไขข้อมูล)

การแก้ไขข้อมูลในตารางนั้น เราได้มีการเขียน routes ไว้แล้ว ในไฟล์ routes/web.php และมีการสร้างลิงค์สำหรับให้ผู้ใช้กดแล้วแก้ไขข้อมูลแล้วนั้น เมื่อผู้ใช้งานกดที่ลิงค์แก้ไขข้อมูล ก็จะไปทำงานที่ routes category/edit/{id} ซึ่งมีการส่ง parameter เข้ามาด้วย คือ id รายการที่เราต้องการแก้ไขแต่ละรายการนั่นเอง



ในบรรทัดที่ 23 เราได้กำหนด URL ชื่อ category/edit/{id} เชื่อมต่อกับ CategoryController แล้วไปทำงานที่ เมธอด edit() เพื่อแสดงหน้าฟอร์มสำหรับการแก้ไขข้อมูล

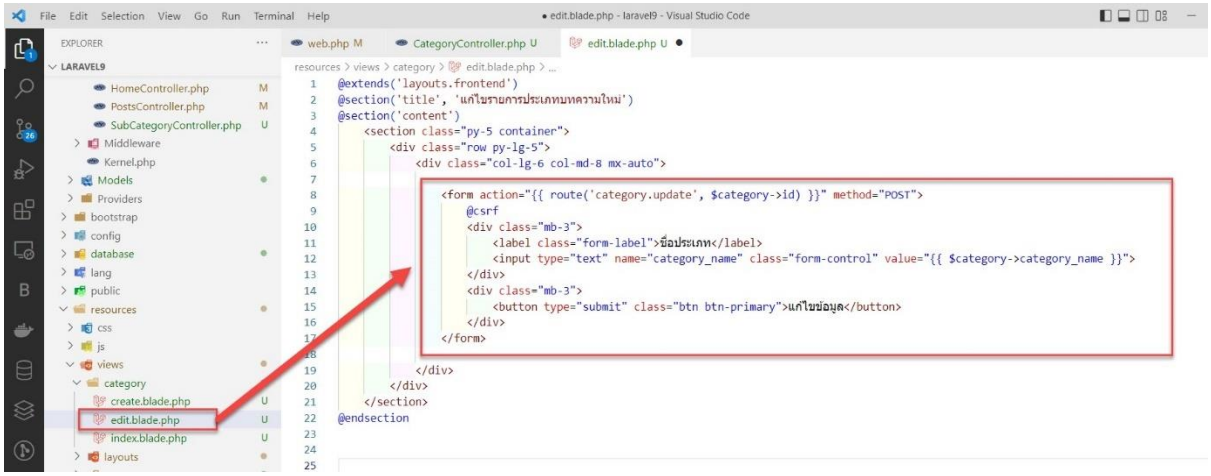
1.เปิดไฟล์ CategoryController.php ในโฟลเดอร์ app\Http\Controllers แล้วสร้าง เมธอด ชื่อว่า edit() และ update() แล้วเพิ่มโค้ด ดังนี้



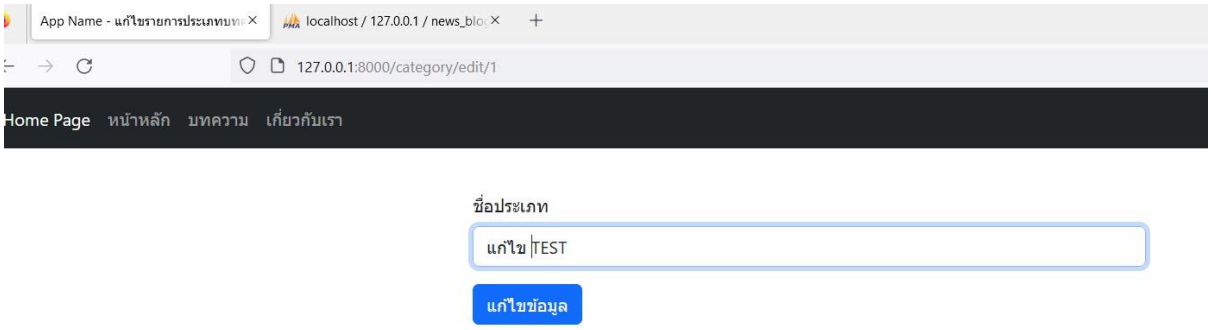
บรรทัดที่ 35 สร้างเมธอด edit() ขึ้นมาและมีการรับค่า $id ของรายการข้อมูลที่ต้องการแก้ไขเข้ามาด้วย แล้วเขียนโค้ดกำหนดเงื่อนไขการดึงข้อมูลจากตารางเก็บไว้ที่ตัวแปร $category จากนั้นก็ส่งข้อมูลไปให้ view แสดงผลตามเงื่อนไข

บรรทัดที่ 42 เมธอด update() จะมีการรับค่า $request จากฟอร์ม และรับค่า id และแก้ไขข้อมูลตามรายการ id ที่ส่งเข้ามาโดยใช้คำสั่งแบบ Query Builder เมื่อแก้ไขเสร็จแล้วก็จะทำการ redirect กลับไปที่หน้า /category

2.สร้างไฟล์ edit.blade.php ที่โฟลเดอร์ category สำหรับแสดงฟอร์มแก้ไขข้อมูล (คลิกขวาที่โฟลเดอร์ category แล้วเลือก New File...) ฟอร์มแก้ไขจะเหมือนกันกับฟอร์มเพิ่มข้อมูลเลยครับ ดังนั้นให้เราไป Copy โค้ดทั้งหมดที่อยู่ในไฟล์ create.balde.php มาวางในไฟล์ edit.blade.php เลยนะครับ จะใช้โค้ดเหมือนกันเลย แล้วค่อยมาแก้ไขโค้ดเพิ่มเติม ดังนี้



บันทึกไฟล์แล้วพิมพ์ localhost:8000/category ที่ Browser แล้วคลิกที่ลิงค์แก้ไขข้อมูล



จากตัวอย่างจะเห็นว่ามีการรับค่าข้อมูลจากฐานข้อมูลมาแสดงผลในฟอร์มเพื่อให้ผู้ใช้แก้ไขข้อมูลตามที่ส่งมา โดยในฟอร์มได้มีการรับตัวแปร id เพื่อใช้เป็นรหัสอ้างอิงในการแก้ไขข้อมูลตามรายการที่เราเลือกแก้ไข ภายในฟอร์มได้กำหนด value ให้กับ input พร้อมดึงข้อมูลจากฐานข้อมูลมาใส่ในช่อง input และเมื่อมีการกดปุ่มแก้ไขข้อมูล ก็จะถูกส่งไปทำงานที่ routes update พร้อมแนบ id ส่งตามไปด้วยโดยใช้วิธีการส่งแบบ POST เมื่อทำการแก้ไขข้อมูลเสร็จแล้ว จะ redirect ไปที่ หน้า /category

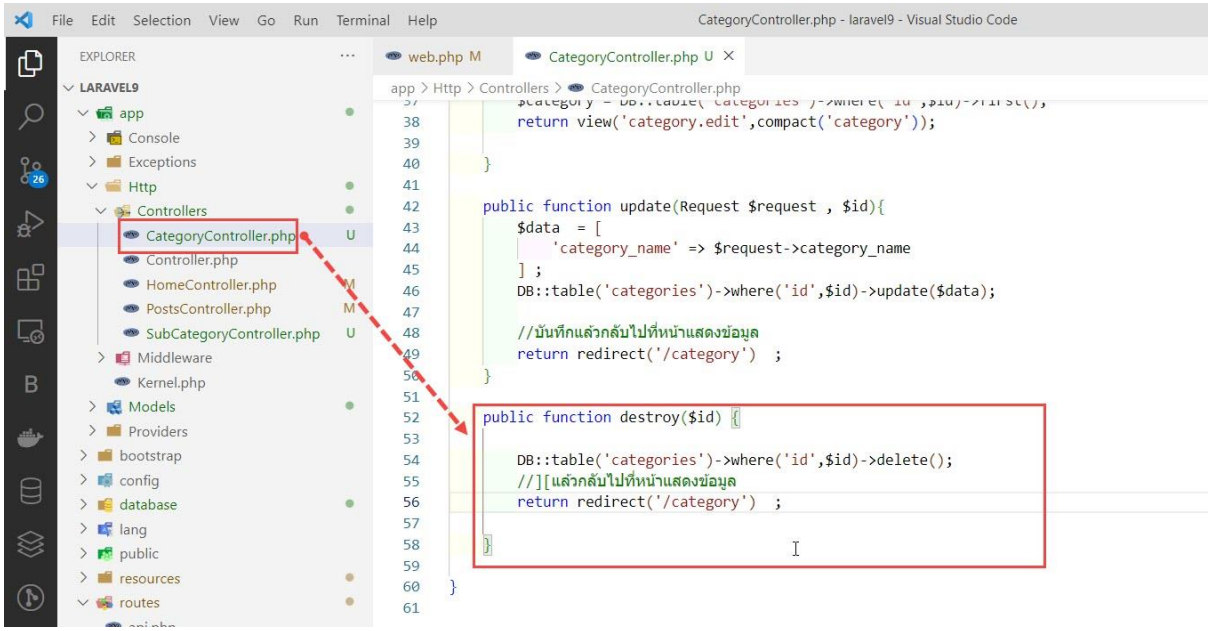
Query Builder (ลบข้อมูล)

1. เราได้มีการเขียน routes ไว้แล้ว ในไฟล์ routes/web.php ดังนี้



ในบรรทัดที่ 26 เราได้กำหนด Routes แบบ GET URL ชื่อ category/destroy/{id} เชื่อมต่อกับ CategoryController แล้วไปทำงานที่ เมธอด destroy()

2. เปิดไฟล์ CategoryController.php ในโฟลเดอร์ app\Http\Controllers แล้วสร้าง เมธอด ชื่อว่า destroy() เพิ่มโค้ด ดังนี้



3.บันทึกไฟล์แล้วพิมพ์ localhost:8000/category ที่ Browser แล้วคลิกที่ลิงค์ลบข้อมูล



“ลองทดสอบดูนะครับ”

จัดการฐานข้อมูลด้วย Eloquent ORM

Eloquent เป็นตัวช่วยให้เราสามารถเขียนโค้ดเพื่อจัดการส่วนต่าง ๆ เกี่ยวกับฐานข้อมูลได้ง่ายขึ้น โดยเขียนคำสั่งไม่กี่บรรทัด เช่น การเพิ่มข้อมูล การอ่านข้อมูล การแก้ไขข้อมูล การลบข้อมูล

ORM หรือ Object Relational Mapping คือ เป็นวิธีการการ map ข้อมูลในรูปแบบตารางฐานข้อมูล ให้มาอยู่ในรูปแบบ object-oriented programming (OOP) เมื่อจะจัดการกับข้อมูลเราก็ทำผ่าน class ที่ map กับตารางในฐานข้อมูล โดยที่เราไม่ต้องเขียน SQL statement ใด ๆ เลย โดยใน Laravel จะมีคำสั่งต่าง ๆ ให้เราได้เลือกใช้งานมากมาย โดยจะสืบทอดการกำหนดค่าคุณสมบัติของตารางมาจาก Model ดังนั้น การที่เราจะสามารถใช้งาน Eloquent ORM ได้ จะต้องมีการสร้าง Model เพื่อใช้อ้างอิงตารางในฐานข้อมูลก่อน

ตัวอย่างคำสั่งที่มักจะใช้บ่อย ๆ

ตัวอย่างคำสั่ง	ความหมาย
$category = Category::all()	ดึงข้อมูลจากตารางทั้งหมดเพื่อแสดงผล
$category = Category::findOrFail(9) หรือใช้ find(9)	เรียกดูข้อมูล 1 แถว
$category = Category::where('cat_id', '=', '9')->first()	เรียกดูข้อมูลตามเงื่อนไข
$category = Category::where('cat_id', '=', '9')->get()	ดึงข้อมูลตามเงื่อนไขทั้งหมดในกรณีอื่น ๆ
$category = Category::count()	นับจำนวนแถวทั้งหมด
$orders = Order::max('total')	หาจำนวนมากที่สุด
$orders = Order::min('total')	หาจำนวนน้อยที่สุด
$orders = Order::avg('total')	หาค่าเฉลี่ย
$orders = Order::sum('total')	หาผลรวมทั้งหมด

ตัวอย่างคำสั่งสำหรับการกรองข้อมูล

ตัวอย่างคำสั่ง	ความหมาย
$posts = Post::where('post_title', 'like', '%A%')->get()	สำหรับค้นหาข้อมูลในฟิวด์ post_title
โดยค้นหา keyword ที่มีตัวอักษร A	
$posts = Post::take(9)->get() หรือ $posts = Post::orderBy('id', 'asc')->limit(9)->get()	สำหรับกำหนดแถวข้อมูลตามเงื่อนไขที่ต้องการ

ตัวอย่างคำสั่งสำหรับเพิ่ม แก้ไข และลบข้อมูล

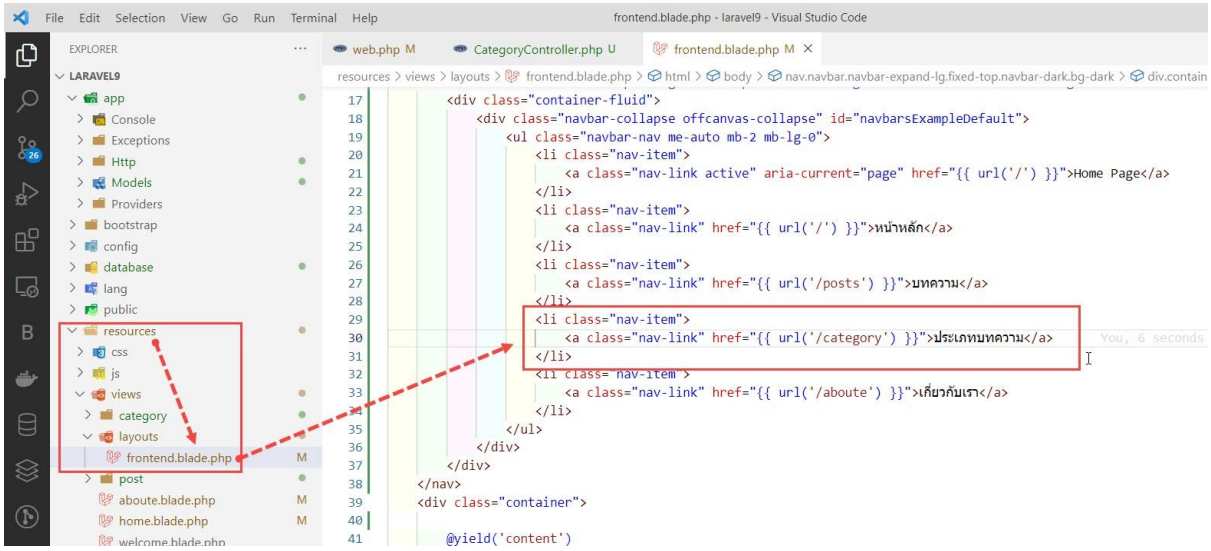
ตัวอย่างคำสั่ง	ความหมาย
save()	ใช้สำหรับเพิ่มข้อมูล หรือ แก้ไขข้อมูล
create()	ใช้สำหรับเพิ่มข้อมูลแบบ Mass Assignment หมายถึงการเพิ่มข้อมูลในบรรทัดเดียว ต้องกำหนดตัวแปร $fillable ใน Model เพื่อเปิดใช้งาน
update()	ใช้สำหรับแก้ไขข้อมูล
delete()	ใช้สำหรับลบข้อมูลตามรายการที่ต้องการลบ ต้องมีการส่ง id ที่ต้องการลบ
$cat = Category::find(9)->delete()	

destroy()	ใช้สำหรับลบข้อมูลตาม id ที่ต้องการลบ สามารถลบได้ทีละหลายรายการ
$cat = Category::destroy(9)	
$cat = Category::destroy(1,2,3,4)	

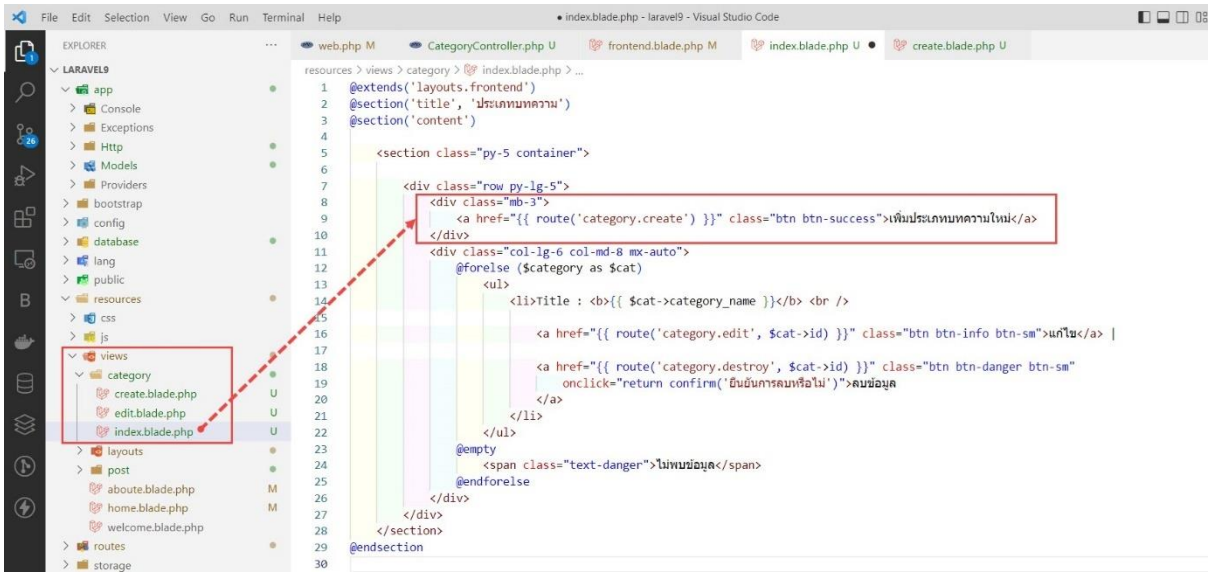
เราได้เรียนรู้คำสั่งต่าง ๆ ของ Eloquent ORM ไปแล้วทีนี้เรามาลองสร้างหน้าเว็บสำหรับแสดงข้อมูลที่ใช้คำสั่งแบบ Eloquent ORM กันต่อเลยครับ จากบทเรียนที่ผ่านมาเราได้เรียนรู้ในเรื่องของการสร้างประเภทบทความ แบบ Query Builder ไปแล้วนั้น เนื่องจากว่ารูปแบบการสร้างจะเหมือนกันต่างกันที่คำสั่งในการเพิ่มข้อมูล การเรียกดูข้อมูล การแก้ไขข้อมูลและการลบข้อมูล ฉะนั้นในบทเรียนนี้เราจะใช้ไฟล์ CategoryController.php ที่ได้สร้างไว้แล้วในการเขียนและทดสอบ...

Eloquent ORM (เพิ่มข้อมูล)

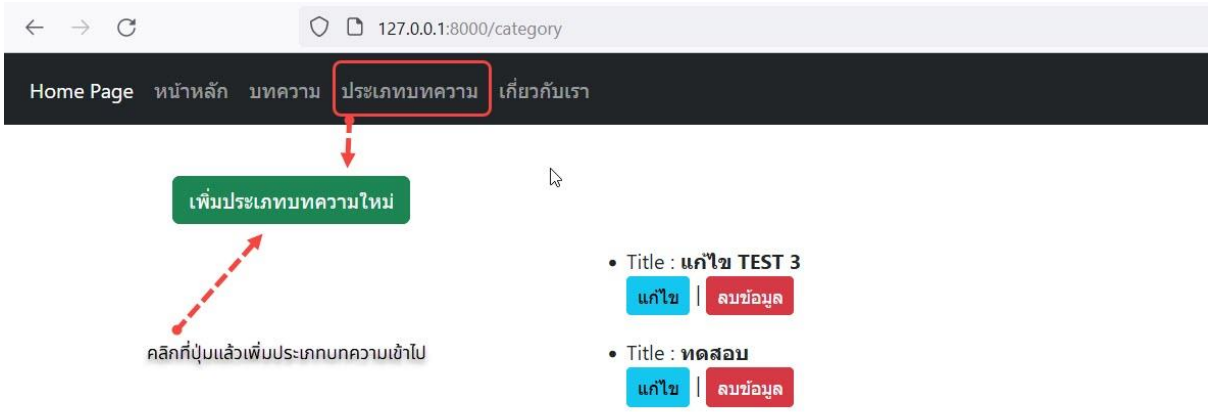
1.1 เปิดไฟล์ frontend.blade.php ในโฟลเดอร์ resources/views/layouts/... แล้วเพิ่มเมนูชื่อว่า ประเภทบทความ ดังนี้



1.2 เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources/views/category/... แล้วเพิ่มปุ่มข้อความว่า เพิ่มประเภทบทความใหม่ ดังนี้



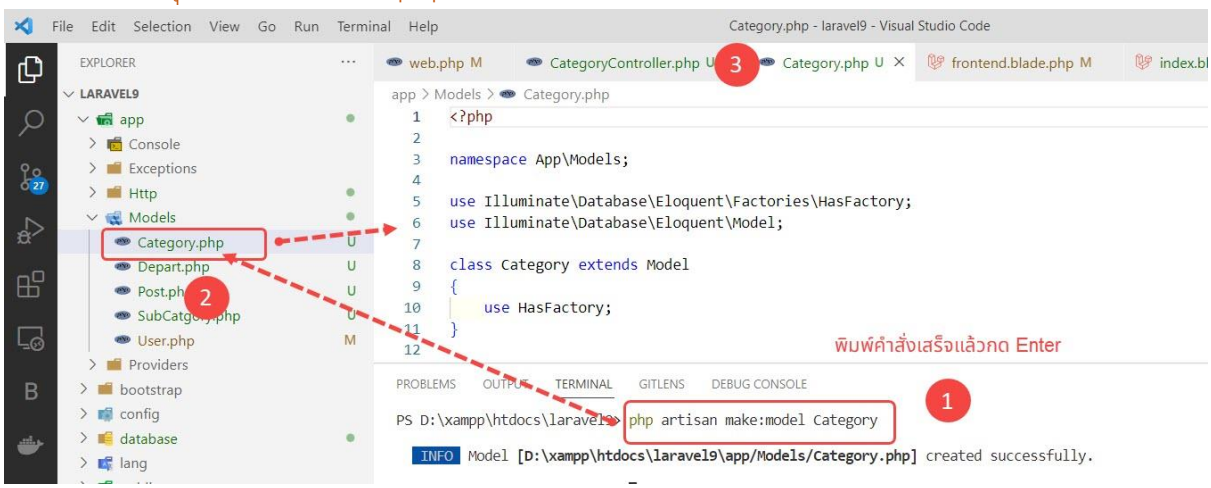
บันทึกไฟล์แล้วพิมพ์ localhost:8000/ ที่ Browser แล้วคลิกที่เมนู ประเภทบทความ จะแสดงหน้าบทความและมีปุ่มสำหรับเพิ่มประเภทบทความใหม่



2.สร้าง Models ใหม่ชื่อว่า Category ดังนี้ (จำได้ไหมครับคำสั่งในการสร้าง Model)

```
php artisan make:model Category
```

เปิด Terminal แล้วพิมพ์คำสั่งแล้วกดปุ่ม Enter ดังนี้ (ถ้า server รันอยู่ก็ให้ กดปุ่ม ctrl+c ออกก่อนนะครับ พอทำทุกอย่างเสร็จค่อยสั่ง php artisan serv ใหม่)

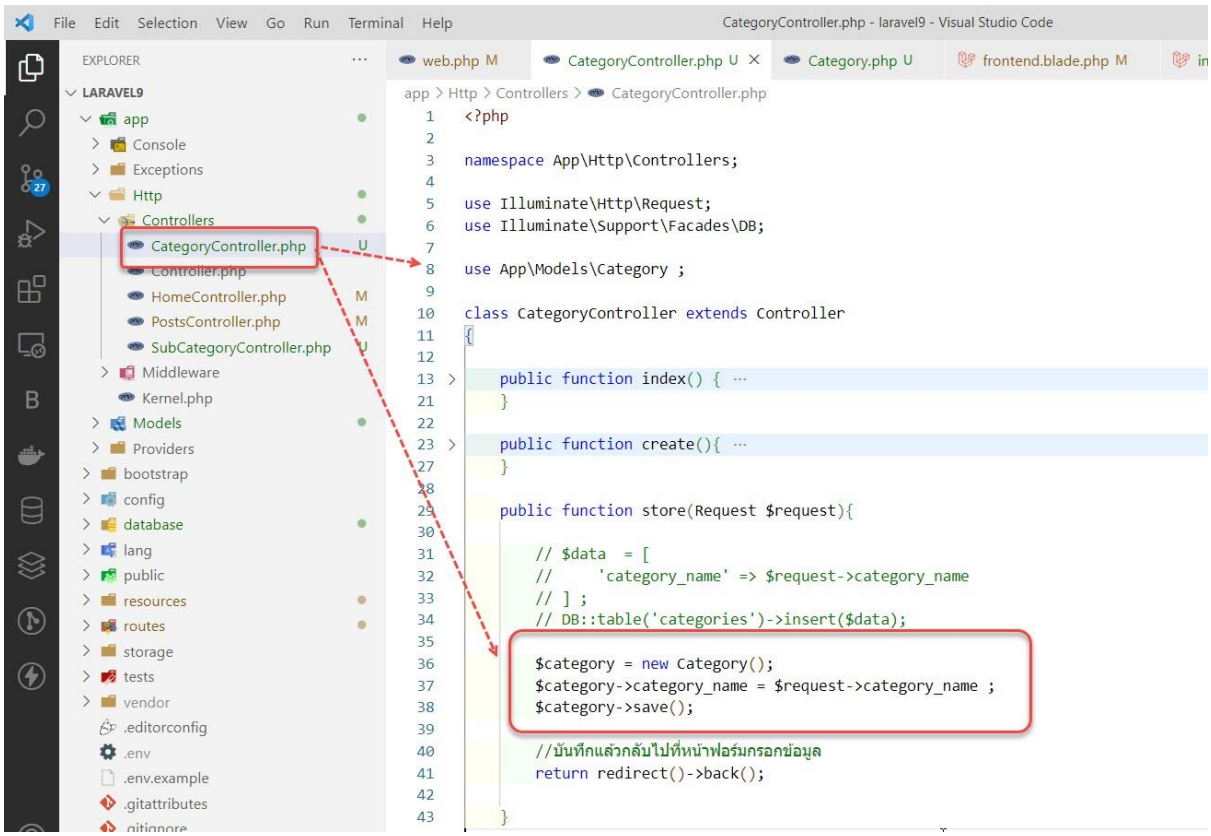


หมายเลข 1 พิมพ์คำสั่ง php artisan make:model Category แล้วกด Enter

หมายเลข 2 ไฟล์ Category.php ถูกสร้างขึ้นมาในโฟลเดอร์ app\Model\...

หมายเลข 3 โค้ดคำสั่งเบื้องต้นที่ถูกสร้างขึ้นอัตโนมัติ

3.เปิดไฟล์ CategoryController.php ใน app\Http\Controller\...แล้วแก้ไขโค้ดใหม่ในเมธอด store() เพื่อใช้งาน Eloquent ORM ดังนี้



บรรทัดที่ 8 เป็นการอ้างอิงโมเดลที่เราต้องการจะเรียกใช้งาน

บรรทัดที่ 36 เป็นการสร้าง Object ใหม่เพื่ออ้างอิงโมเดล Category เก็บไว้ที่ตัวแปร $category

บรรทัดที่ 37 เป็นการรับค่าจาก $requets->category_name แล้วอ้างอิงฟิวด์ในตารางเพื่อบันทึกตาม $request ที่รับมาแล้วยัดเข้าไปในตารางตามฟิวด์ที่ระบุ

บรรทัดที่ 38 คำสั่งในการบันทึกข้อมูล save()

บรรทัดที่ 41 เมื่อบันทึกข้อมูลแล้วให้ redirect กลับไปที่หน้าฟอร์มกรอกข้อมูล

4.บันทึกไฟล์แล้วพิมพ์ localhost:8000/category ที่ Browser คลิกที่ปุ่ม เพิ่มประเภทบทความใหม่ แล้วทดสอบเพิ่มข้อมูลดูครับ ถ้าไม่ติดปัญหาอะไร ก็สามารถเพิ่มข้อมูลได้ปกติ ให้เพิ่มข้อมูลลงไปตามต้องการได้เลยครับ เดี๋ยวจะไปจะเป็นการสร้างหน้าเพจเพื่อแสดงข้อมูลในหน้าเว็บต่อครับ

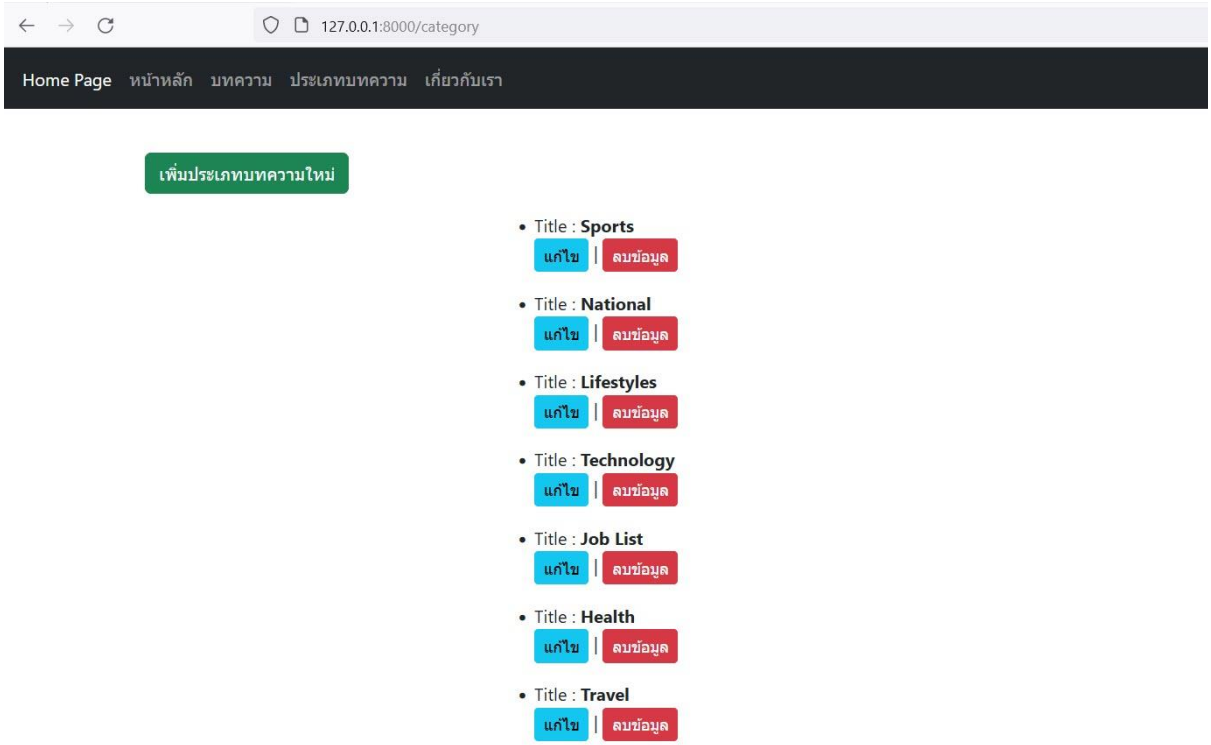
Eloquent ORM (แสดงข้อมูล)

1.เปิดไฟล์ CategoryController.php ใน app\Http\Controller\...แล้วแก้ไขโค้ดใหม่ในเมธอด index() เพื่อใช้งาน Eloquent ORM ดังนี้



จากไฟล์ CategoryController.php **บรรทัดที่ 8** เป็นคำสั่งสำหรับการอ้างอิงโมเดล Category **บรรทัดที่ 17** เป็นการใช้คำสั่ง Eloquent ORM สำหรับดูหรือดึงข้อมูลจากตาราง categories และใช้ฟังก์ชัน all() เพื่อเรียกดูข้อมูลทั้งหมดที่อยู่ในตาราง หากต้องการแสดงข้อมูลแบบกำหนดเงื่อนไข

เมื่อบันทึกไฟล์แล้วพิมพ์ localhost:8000/category ที่ Browser จะเห็นว่าข้อมูลก็แสดงเหมือนเดิม



หากต้องการใส่เงื่อนไขการแสดงข้อมูล สามารถเขียนคำสั่ง ดังนี้

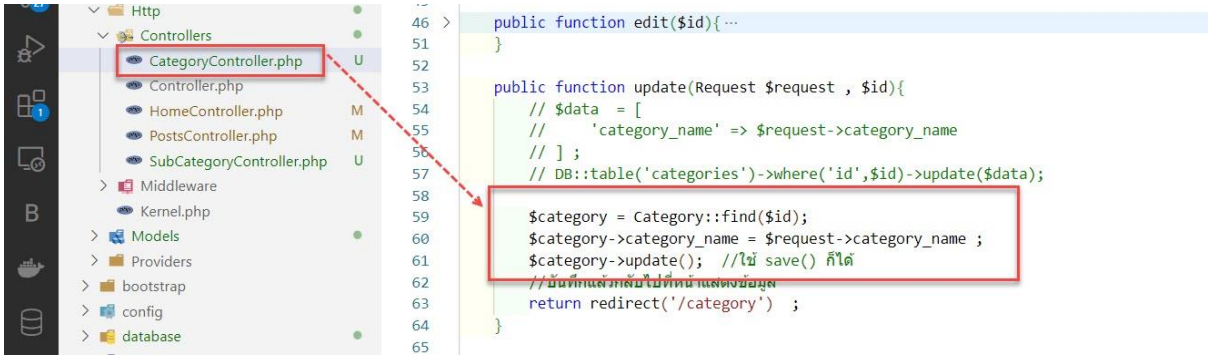
```
12
13      public function index() {
14
15          //$category = DB::table('categories')->get();
16
17          //$category = Category::all() ;
18          $category = Category::orderBy('id','asc')->limit(2)->get() ;
19
20          return view('category.index',compact('category'));
21
22      }
23
```

```
13      public function index() {
14
15          //$category = DB::table('categories')->get();
16
17          //$category = Category::all() ;
18          $category = Category::take(4)->get() ;
19
20          return view('category.index',compact('category'));
21
22      }
23
```

บันทึกและทดสอบการแสดงผลก็จะได้ข้อมูลตามเงื่อนไขที่กำหนด

Eloquent ORM (แก้ไขข้อมูล)

1.เปิดไฟล์ CategoryController.php ใน app\Http\Controller\...แล้วแก้ไขโค้ดใหม่ในเมธอด update() เพื่อใช้งาน Eloquent ORM ดังนี้



บรรทัดที่ 59 เป็นการใช้คำสั่ง Eloquent ORM สำหรับแสดงข้อมูลตามเงื่อนไขที่ต้องการแก้ไข โดยกำหนดเป็นตัวแปร $id คือการค่าที่ถูกส่งเข้ามาเพื่อแก้ไข จะใช้ find() เป็นเงื่อนไขของการแก้ไขข้อมูลแต่ละรายการ สามารถใช้แบบนี้ก็ได้

```
$category = Category::findOrFail($id);
$category = Category::where('id',$id)->first();
```

บรรทัดที่ 60 เป็นการรับค่าจาก $requets->category_name แล้วอ้างอิงฟิวด์ในตารางเพื่อบันทึก

ตาม $request ที่รับมาแล้วยัดเข้าไปในตารางตามฟิวด์ที่ระบุ

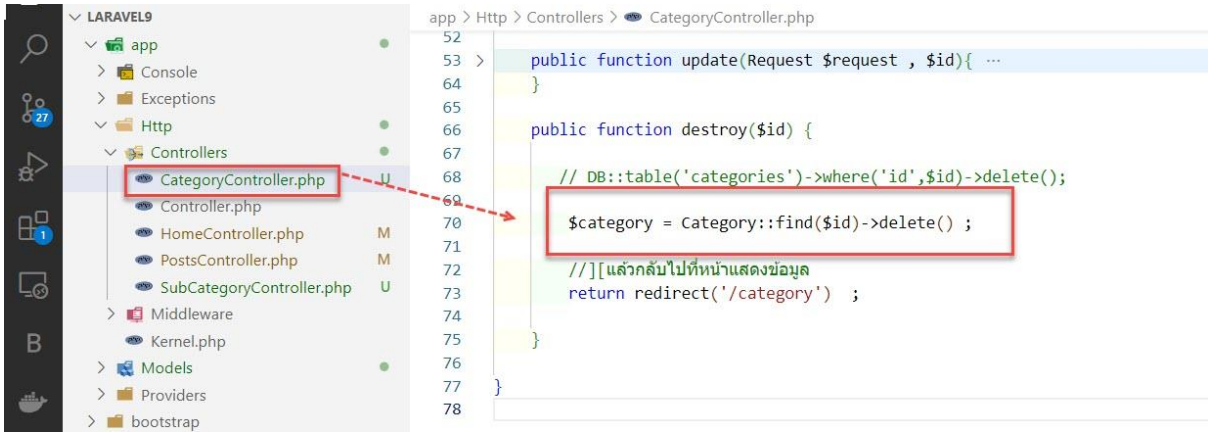
บรรทัดที่ 38 คำสั่งในการแก้ไขข้อมูล update()

บรรทัดที่ 41 เมื่อบันทึกข้อมูลแล้วให้ redirect กลับไปที่หน้าประเภทบทความ

2.บันทึกไฟล์แล้วพิมพ์ localhost:8000/category ที่ Browser คลิกที่ปุ่ม แก้ไข แล้วทดสอบแก้ไขข้อมูลดูครับ

Eloquent ORM (ลบข้อมูล)

1.เปิดไฟล์ CategoryController.php ใน app\Http\Controller\...แล้วแก้ไขโค้ดใหม่ในเมธอด destroy() เพื่อใช้งาน Eloquent ORM ดังนี้



หรือจะใช้คำสั่งนี้ ก็ได้เหมือนกัน

```
$category = Category::destroy($id) ;
```

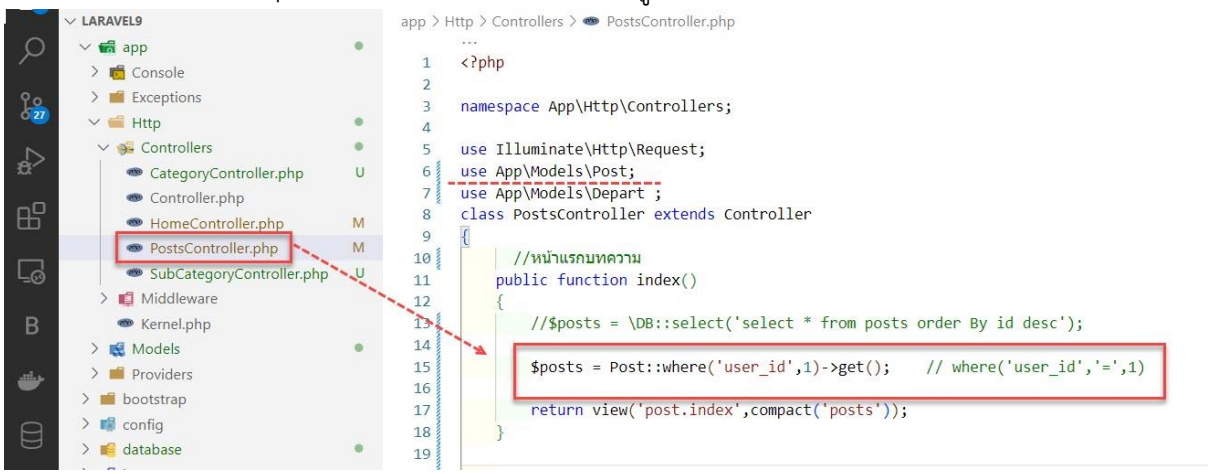
2.บันทึกไฟล์แล้วพิมพ์ localhost:8000/category ที่ Browser คลิกที่ปุ่ม ลบข้อมูล

----------ฝากไปทดสอบและลองทำดูนะครับ----------

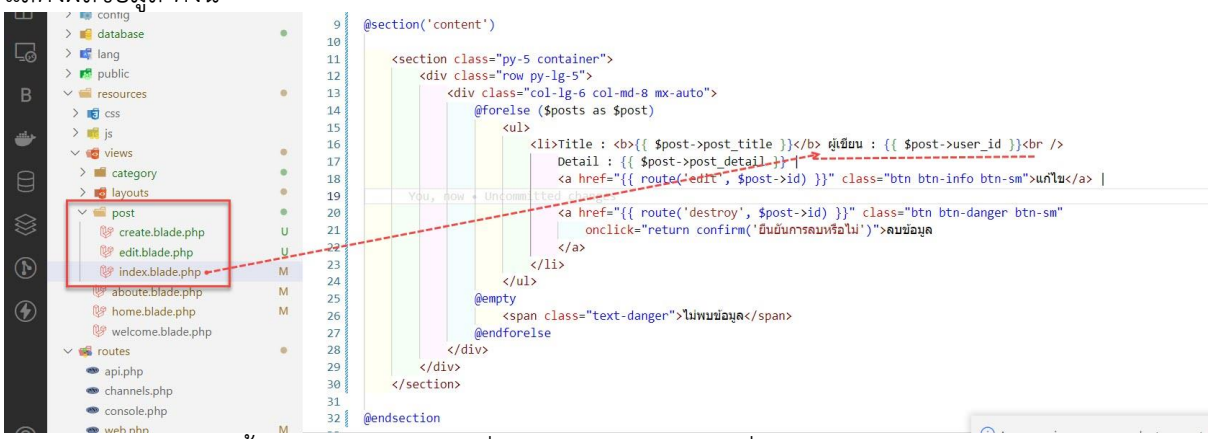
Eloquent ORM (กรองข้อมูลตามเงื่อนไขที่ต้องการ)

การกรองข้อมูล คือ การระบุเงื่อนไขเพิ่มเติมในการดึงข้อมูลจากฐานข้อมูลมาแสดงในส่วนแสดงผล

1.เปิดไฟล์ PostsController.php ใน app\Http\Controller\...แล้วแก้ไขโค้ดใหม่ในเมธอด index() เพื่อใช้งาน Eloquent ORM สำหรับการกรองข้อมูล ดังนี้

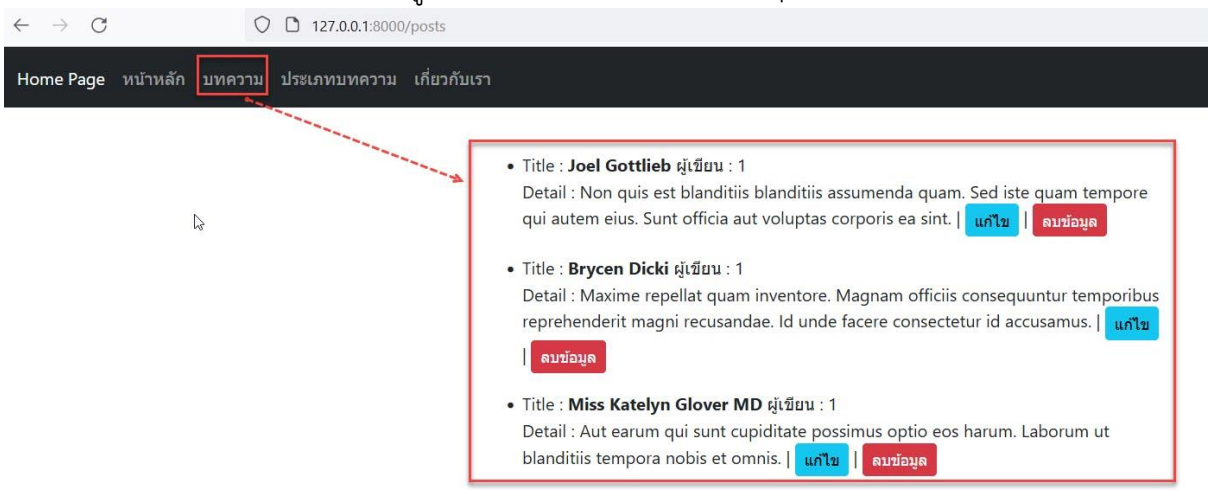


2.เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources/views/post เพื่อแก้ไขในส่วนของการแสดงผลข้อมูล ดังนี้



จากตัวอย่างนี้จะเห็นว่า ในบรรทัดที่ 15 มีการสร้างตัวแปรชื่อ $posts และกำหนดค่าให้กับตัวแปร โดยอ้างอิง Models ชื่อว่า Post เพื่อเรียกตาราง posts จากฐานข้อมูลและมีการใส่เงื่อนไข `where('user_id',1)` โดยการกรองข้อมูล user_id = 1 เท่านั้น จากนั้นก็ส่งตัวแปร $posts ไปให้ส่วนของการแสดงผล คือ post.index ในส่วนของการแสดงผล ก็จะพ่นตัวแปร user_id ออกมาตามเงื่อนไขที่กำหนด

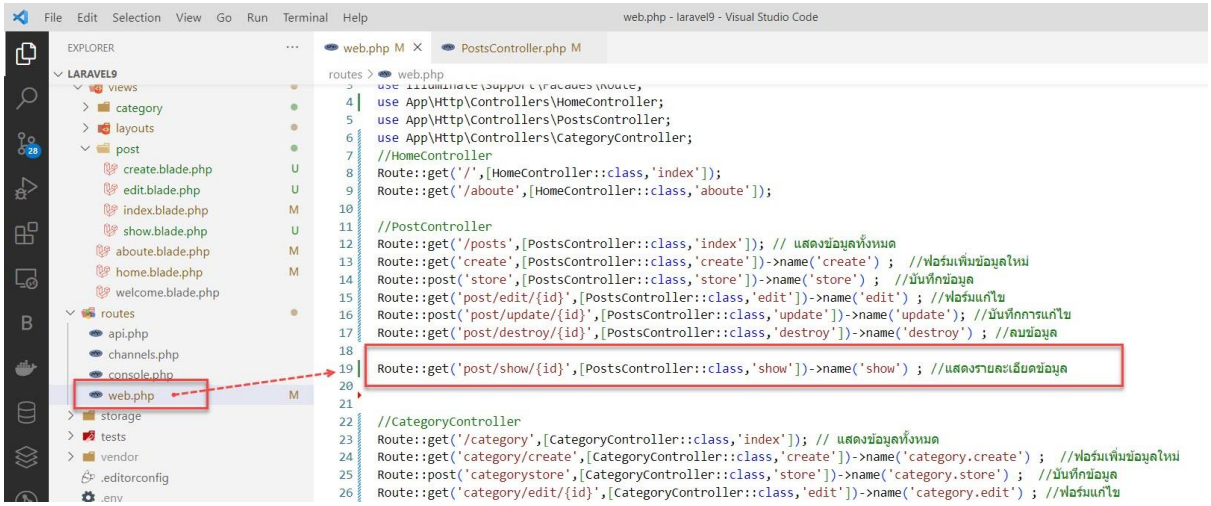
3.บันทึกไฟล์แล้วทดสอบรันดูอีกรอบ พิมพ์ localhost:8000/posts ที่ Browser



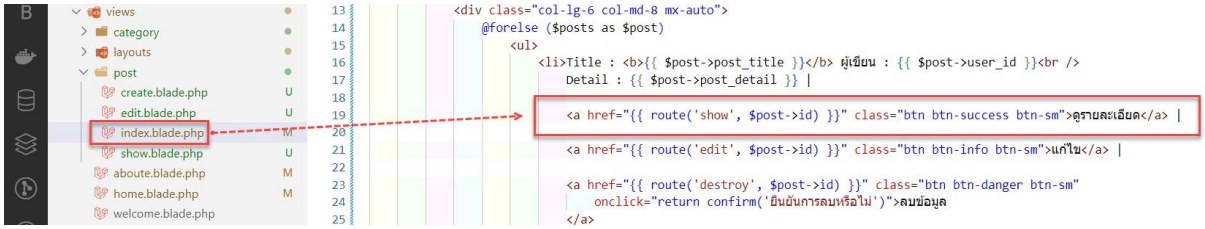
Eloquent ORM (แสดงรายละเอียดข้อมูลบทความแต่ละรายการ)

หากต้องการแสดงรายละเอียดของแต่ละบทความ สามารถทำได้ด้วยการใช้ฟังก์ชัน find(1) , findOrFail(2) หรือใช้ where('id',1)->first() ก็ได้ครับ

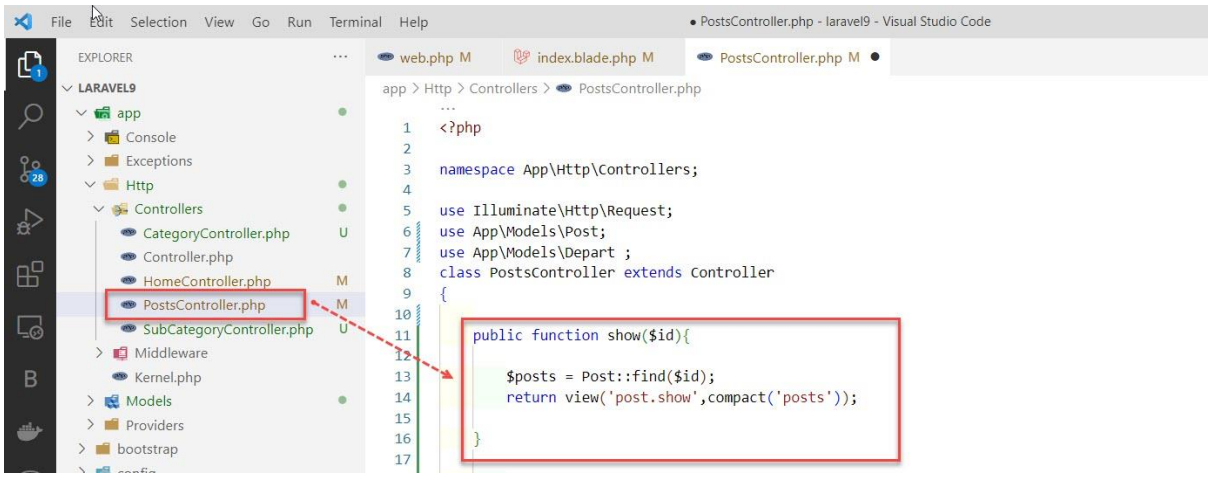
1.เปิดไฟล์ routes/web.php เพื่อสร้างเส้นทาง สำหรับการเข้าถึง เขียนโค้ดดังนี้



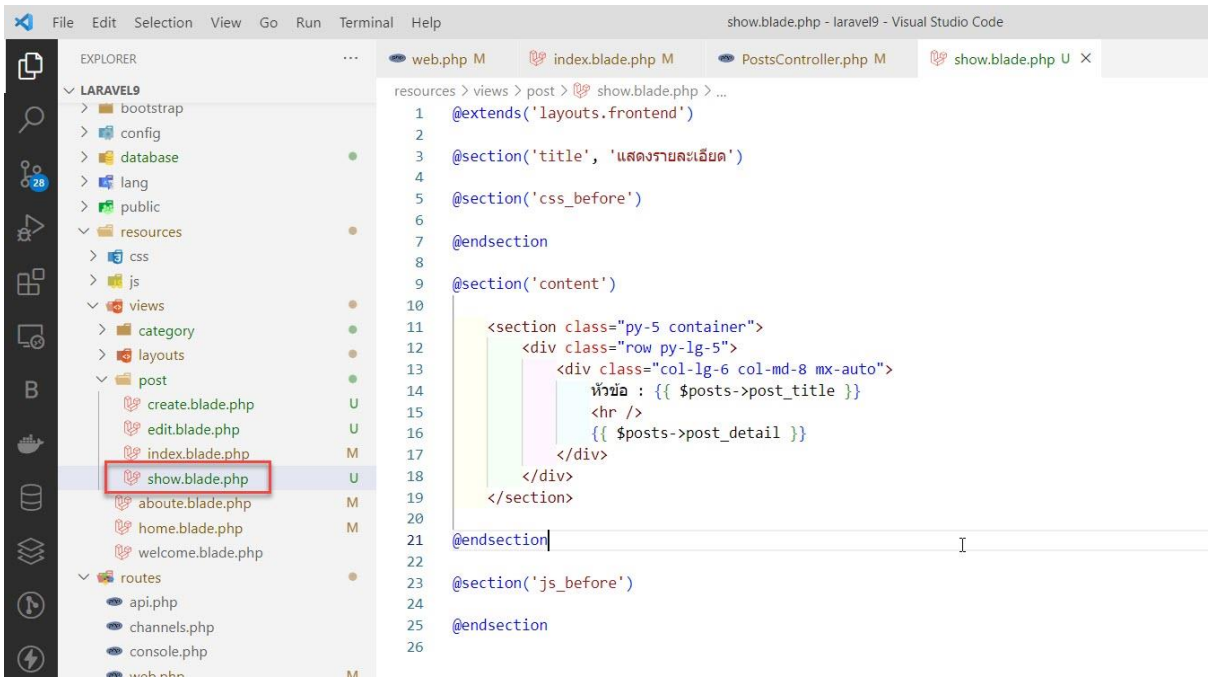
2.เปิดไฟล์ index.blade.php ใน resources\views\post\...แล้วเพิ่มลิงค์สำหรับเข้าถึงรายละเอียดบทความ ดังนี้



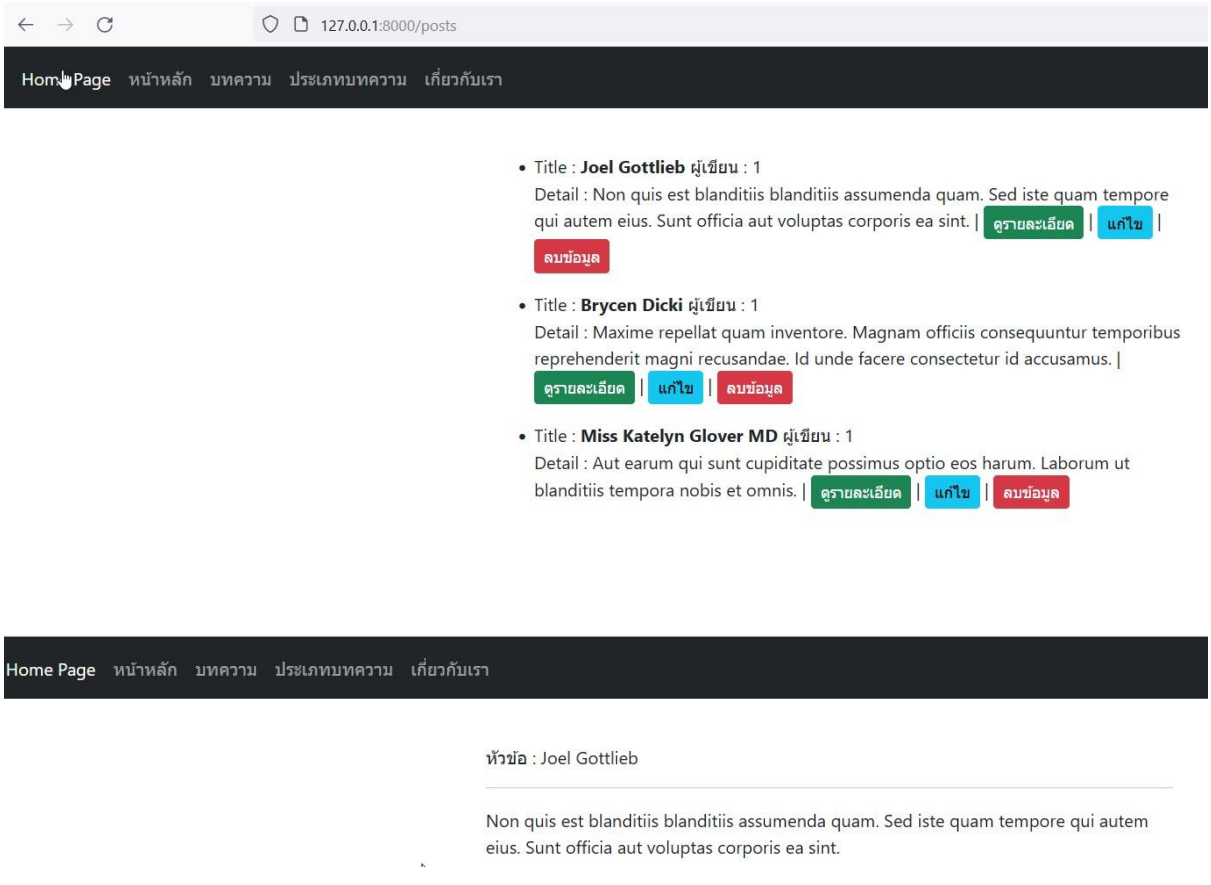
3.เปิดไฟล์ PostsController.php ใน app\Http\Controller\...แล้วเพิ่มเมธอด ชื่อ show() แล้วเขียนโค้ด ดังนี้



4.สร้างไฟล์ใหม่สำหรับการแสดงผล ชื่อ show.blade.php ใน resources\views\post\...แล้วเพิ่มโค้ด ดังนี้



5.บันทึกไฟล์แล้วพิมพ์ localhost:8000/posts ที่ Browser คลิกที่ปุ่ม ดูรายละเอียด



Paginate (การแบ่งหน้าเนื้อหา)

ใน Laravel Framework จะมีเครื่องมือไว้สำหรับการแบ่งหน้าการแสดงผลข้อมูลมาให้เราใช้งานอย่างง่าย สามารถกำหนดจำนวนข้อมูลหรือจำนวนหน้าตามต้องการ โดยจะใช้คำสั่ง

```
Paginate() or simplePaginate()
```

การเขียนคำสั่ง

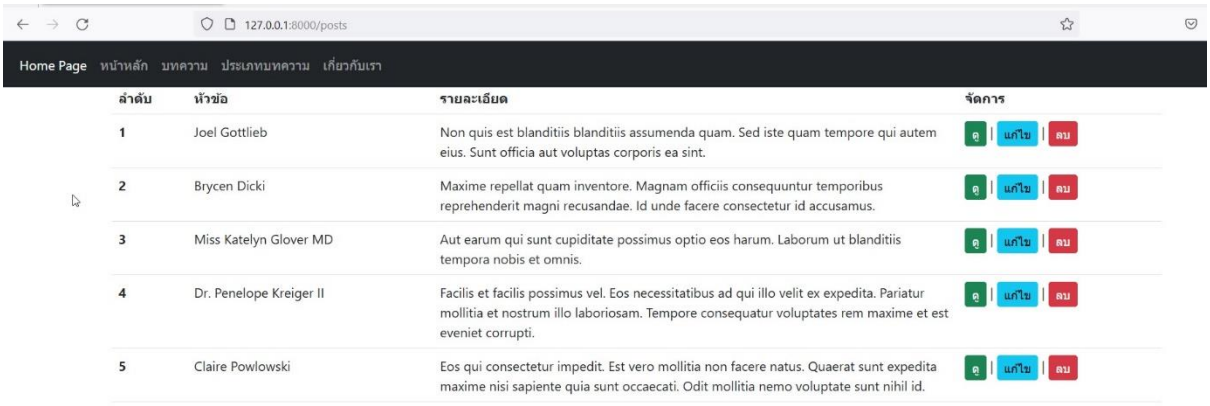
```
$posts = Post::paginate(10) ;      //แสดงข้อมูล 10 รายการต่อหน้าแบบปกติ

$posts = Post::simplePaginate(10) ;      //แบ่งหน้าอย่างง่าย
```

ก่อนอื่น ให้ไปปรับรูปแบบการแสดงผลในหน้าบทความ (posts) ใหม่ ดังนี้

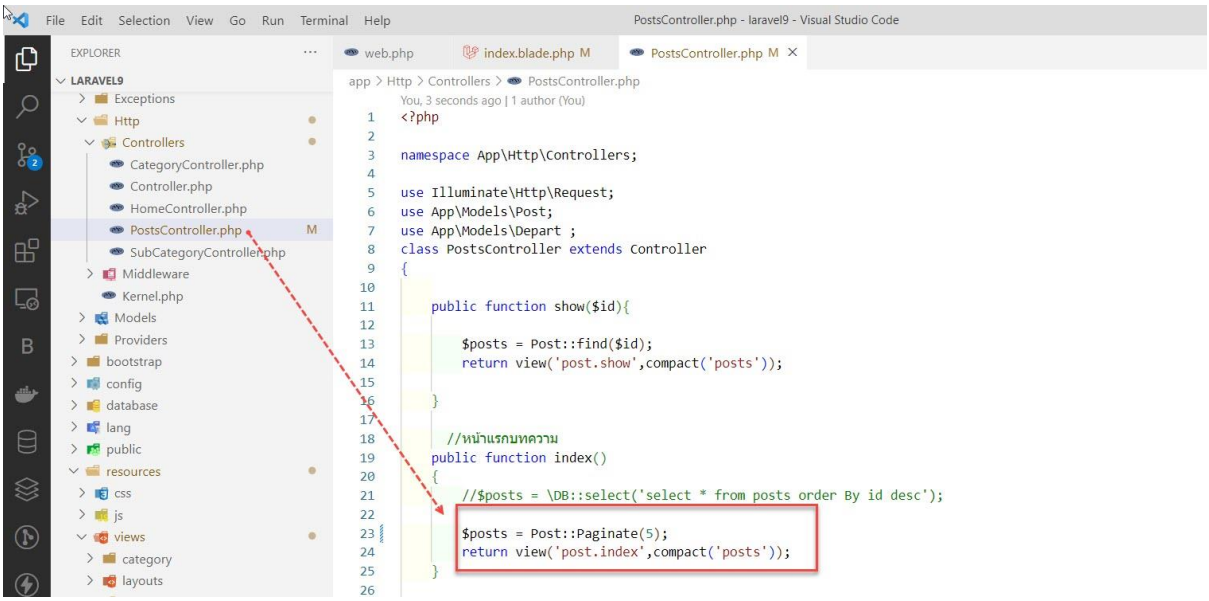
1. เปิดไฟล์ index.blade.php ใน resources\views\post\...แล้วแก้ไขรูปแบบการแสดงผล ดังนี้

```
@extends('layouts.frontend')
@section('title', 'หน้าแรกบทความ')
@section('content')
   <section class="py-5 container">
      <div class="row">
         <div class="col-lg-12  mx-auto">
            <table class="table table-hover">
               <thead><tr>
                     <th scope="col">ลำดับ</th>
                     <th scope="col">หัวข้อ</th>
                     <th scope="col">รายละเอียด</th>
                     <th scope="col">จัดการ</th>
                  </tr>
               </thead>
               @foreach ($posts as $post)
                  <tbody><tr>
                        <th scope="col">{{ $post->id }}</th>
                        <td scope="col">{{ $post->post_title }}</td>
                        <td scope="col" style="width: 50%">{{ $post->post_detail }}</td>
                        <td scope="col">
                           <a href="{{ route('show', $post->id) }}" class="btn btn-success btn-sm">ดู</a>|
                           <a href="{{ route('edit', $post->id) }}" class="btn btn-info btn-sm">แก้ไข</a>|
                           <a href="{{ route('destroy', $post->id) }}" class="btn btn-danger btn-sm"
                              onclick="return confirm('ยืนยันการลบหรือไม่')">ลบ
                           </a>
                        </td></tr></tbody>
               @endforeach
            </table>
         </div></div></section>
@endsection
```

บันทึกไฟล์แล้วพิมพ์ localhost:8000/posts ที่ Browser



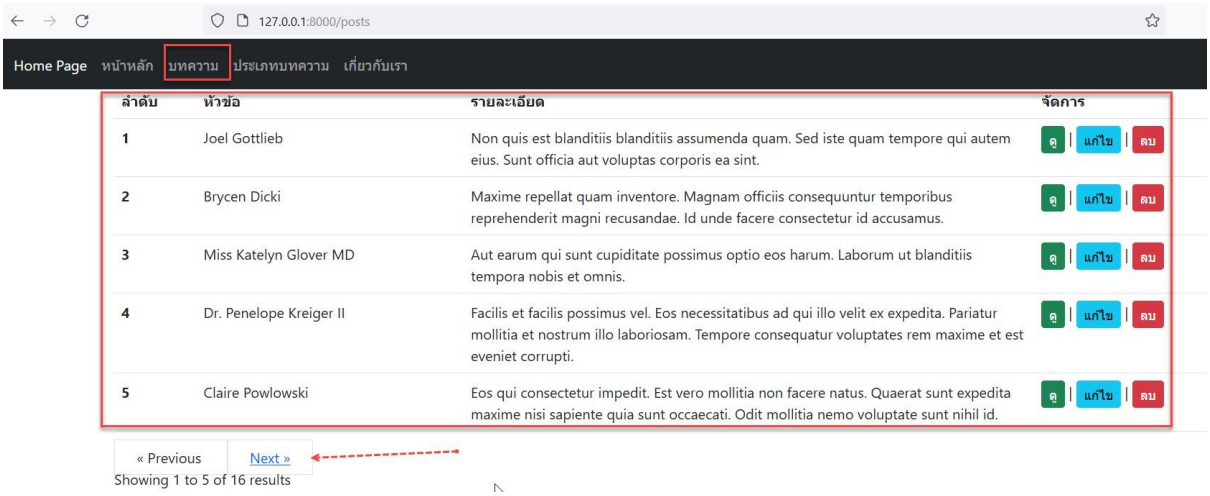
2.เปิดไฟล์ PostsController.php ใน app\Http\Controller\...แล้วแก้ไขโค้ดในส่วนของ เมธอด index() แล้วแก้ไขโค้ด ดังนี้



จากตัวอย่างเป็นการแบ่งหน้าให้แสดงผลข้อมูลจำนวน 5 รายการต่อหน้าของข้อมูลที่มีทั้งหมด บันทึกไฟล์แล้วไปเพิ่มคำสั่งสำหรับแสดงผลที่มีการแบ่งหน้าที่ไฟล์ index.blade.php ในโฟลเดอร์ resources\views\post\...โดยเพิ่มคำสั่ง {{ $posts->links() }} ในตำแหน่งที่ต้องการให้สร้างเลขหน้า

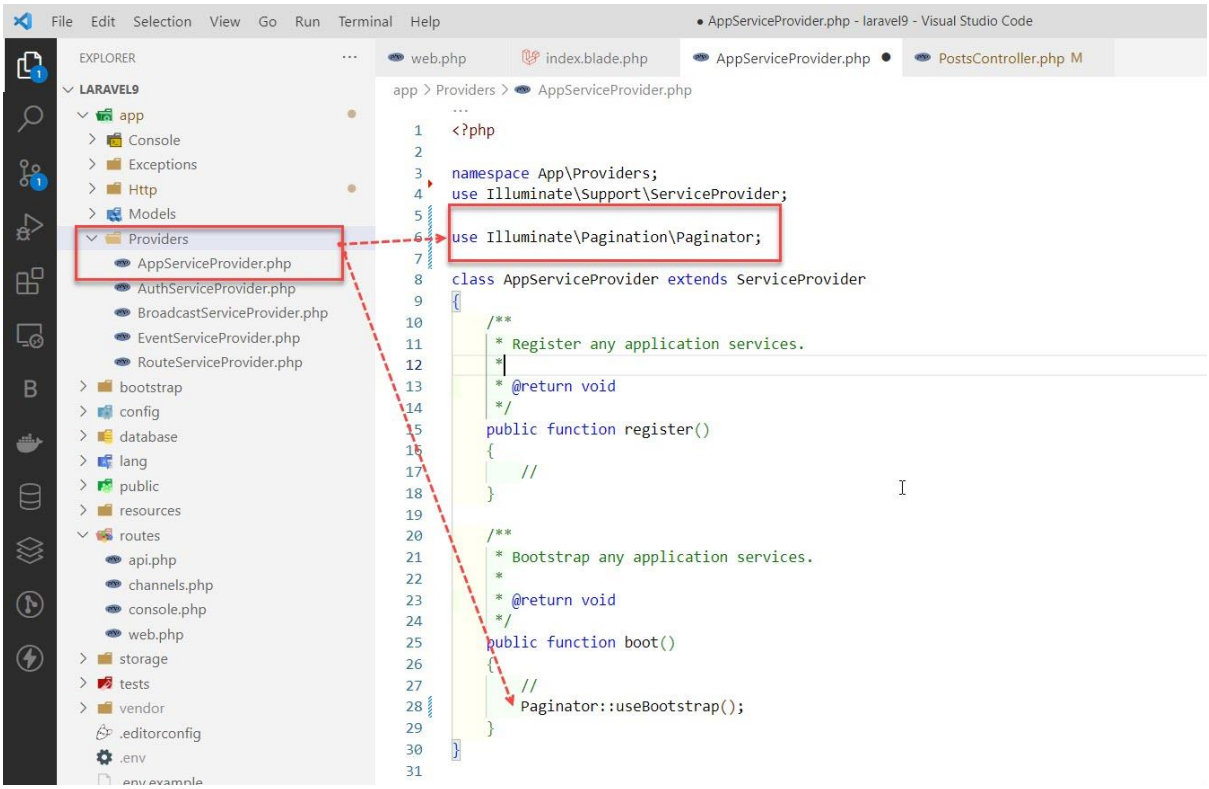


บันทึกไฟล์แล้วพิมพ์ localhost:8000/posts ที่ Browser อีกรอบ

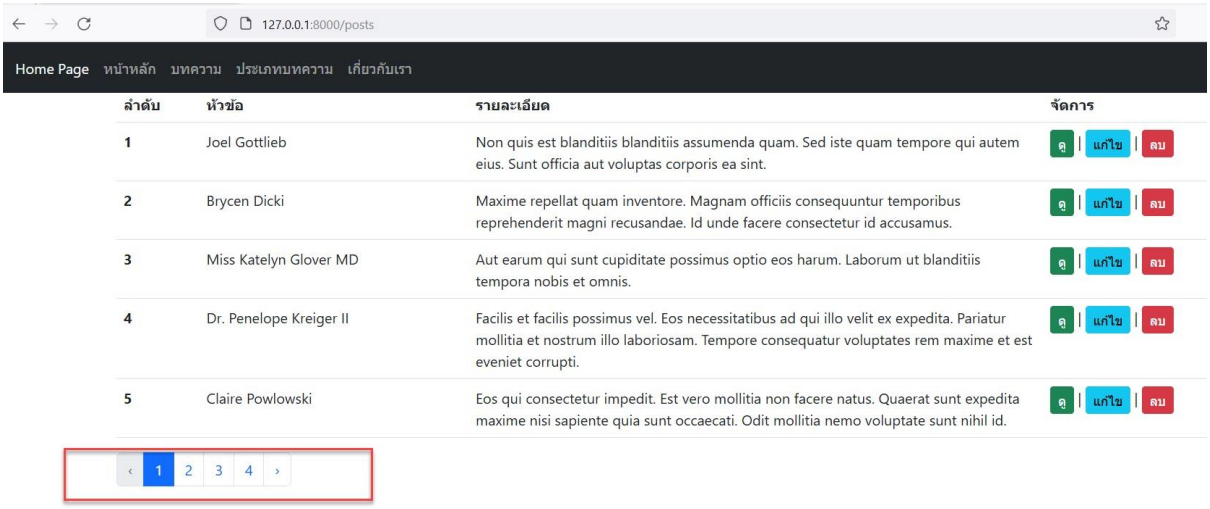


สามารถปรับแต่งการแบ่งหน้าให้แสดงตัวเลขหน้า ได้ดังนี้

เปิดไฟล์ AppServiceProvider.php ใน app\providers\...แล้วแก้ไขโค้ดในส่วนของเมธอด boot() แล้วเพิ่มโค้ด ดังนี้



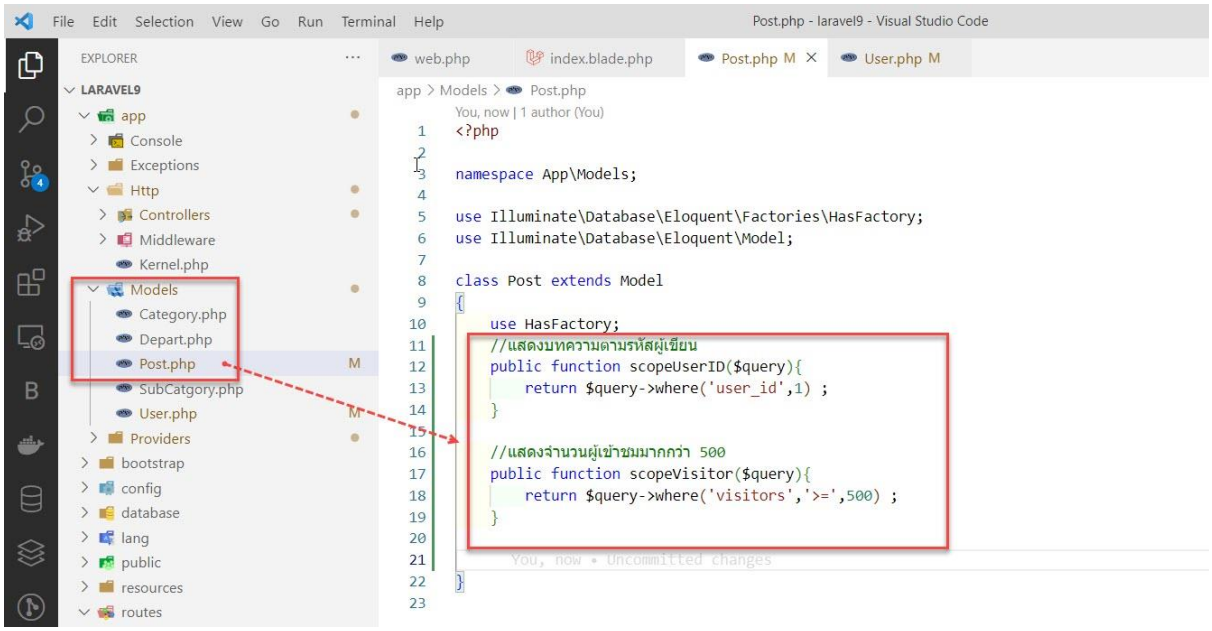
บันทึกไฟล์แล้วดูการแสดงผลอีกครั้ง



การใช้งาน Query Scopes

เป็นการรวมเอา Query ที่มีความซับซ้อนมากขึ้น เพื่อให้ได้ข้อมูลที่ตรงตามความต้องการ โดยจะเขียนไว้ในส่วนของ Models ที่ต้องการ การตั้งชื่อ Query Scopes จะต้องเขียนขึ้นต้นด้วยคำว่า scope ทุกครั้ง จากนั้นค่อยระบุชื่อเมธอดตามต้องการ การเขียนสามารถเขียนได้ตามตัวอย่าง ดังนี้

เปิดไฟล์ Post.php ใน app\Http\Models\...แล้วเพิ่มเมธอด (userid , visitor) ดังนี้

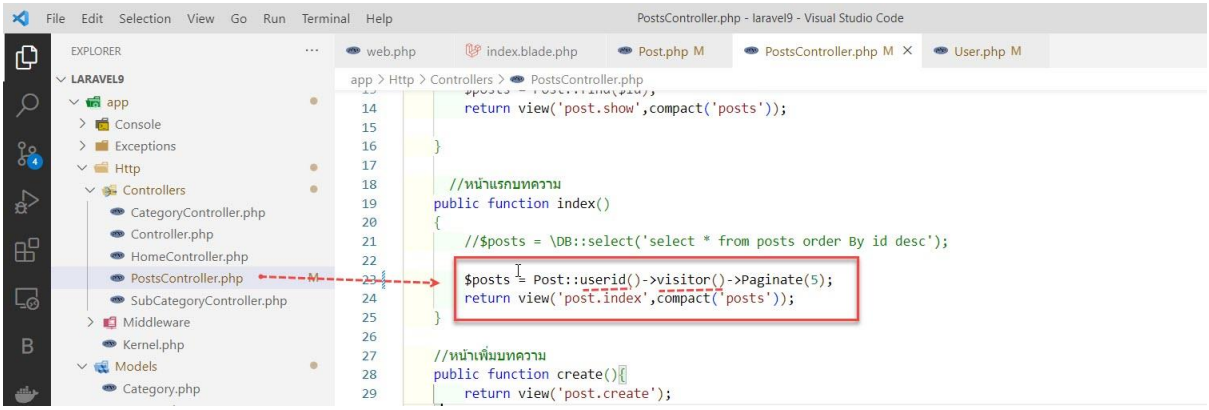


เวลาเรียกใช้งาน ให้เรียกที่คอนโทรลเลอร์ โดยให้ตัดคำว่า scope ออกไป เช่น

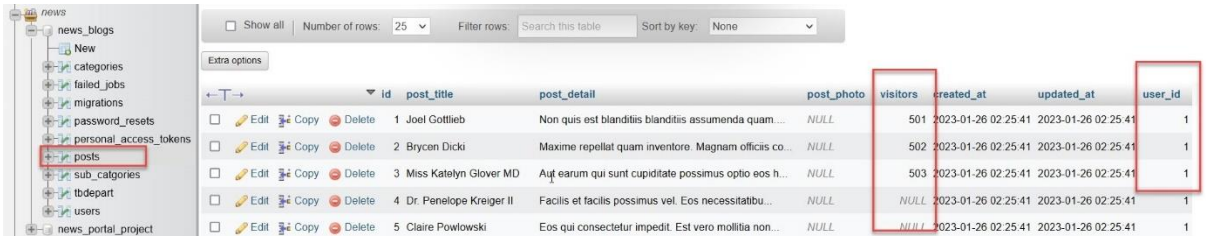
```
$posts = Post::userid()->visitor()->all()
```

ลองเขียนคำสั่งเรียกใช้ในคอนโทรลเลอร์ดูกันเลยครับ

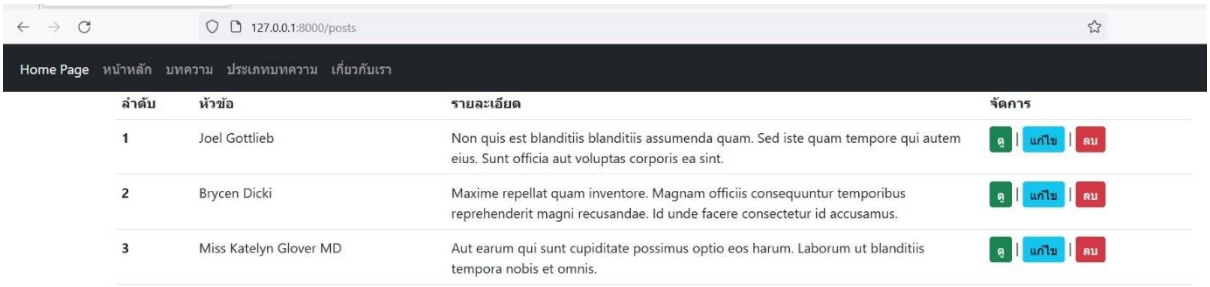
เนื่องจากเราเขียน Query Scope ไว้ที่ Models Post ฉะนั้นเวลาเรียกใช้งานก็ให้เปิดไฟล์ PostsController.php ใน app\Http\Controller\...แล้วแก้ไขโค้ดในส่วนของ เมธอด index() แล้วแก้ไขโค้ด ดังนี้



บันทึกไฟล์แล้วไปแก้ค่า ใน phpMyAdmin ในส่วนของตาราง posts ฟิวด์ visitors และ user_id ตามนี้ครับ



แล้วพิมพ์ localhost:8000/posts ที่ Browser อีกรอบ

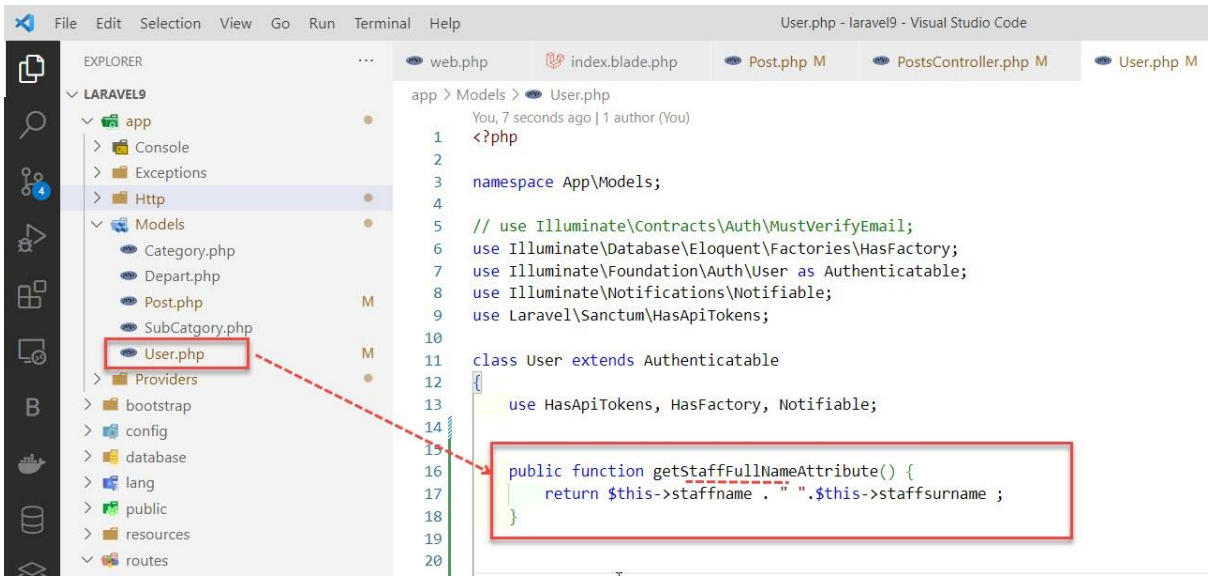


จะแสดงผลข้อมูลตาม Query Scopes ที่เราเขียนไว้ คือ บทความที่มี user_id = 1 และมียอดคนดูมากกว่าหรือเท่ากับ 500

การใช้งาน Accessors

Accessors หรือ getter เป็นการกำหนดชื่อให้กับฟิลด์ของข้อมูลที่ไม่มีในตารางของข้อมูล สามารถสร้าง Attribute ขึ้นมาใช้งานใหม่ เพื่อเตรียมข้อมูลสำหรับการเรียกใช้งาน โดยการเรียกใช้งานจะเรียกใช้งานผ่าน Attribute ใหม่ เช่น หากต้องการนำ คำนำหน้าชื่อและชื่อหรือนามสกุล ที่อยู่คนละฟิวด์มาเชื่อมต่อกันให้ได้ชื่อเต็ม การคำนวณผลรวมต่าง ๆ โดยจะสร้างเมธอดไว้ในไฟล์ Models ที่ต้องการ และชื่อเมธอดต้องขึ้นต้นด้วย get เสมอแล้วค่อยตามด้วยชื่อ Accessors ที่ต้องการและต่อท้ายด้วย Attribute สามารถเขียนได้ตามตัวอย่าง ดังนี้

เปิดไฟล์ User.php ใน app\Http\Models\...แล้วเพิ่มเมธอด ดังนี้



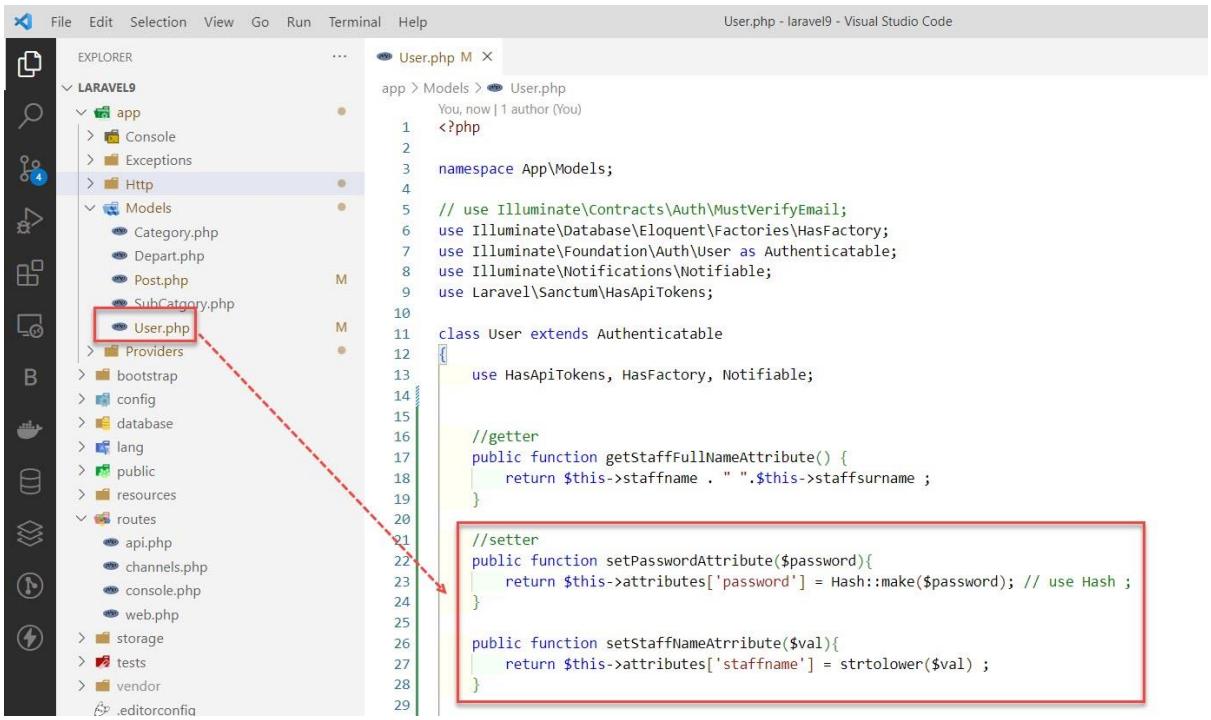
เวลาเรียกใช้งาน ให้เรียกที่คอนโทรลเลอร์ โดยให้ตัดคำว่า get และ Attribute ออกไป เช่น

```
$user = User::find(9);
$name = $user->fullname ;
```

การใช้งาน Mutators

Mutators หรือ setter การทำงานจะเหมือนกันกับ Accessors คือ เป็นการสร้างฟิวด์หรือสร้าง Attribute ขึ้นมาใหม่ที่ไม่มีในตาราง เพื่อจัดเก็บรูปแบบของข้อมูลตามที่ต้องการ โดยจะสร้างเมธอดขึ้นแล้วรับค่าพารามิเตอร์เข้ามาแล้ว set ค่าให้กับ Attribute เช่น การเข้ารหัสผ่าน การกำหนดข้อความตัวอักษรให้เป็นตัวพิมพ์เล็ก หรือ ตัวพิมพ์ใหญ่ โดยจะสร้างเมธอดไว้ในไฟล์ Models ที่ต้องการ และชื่อเมธอดต้องขึ้นต้นด้วย set เสมอแล้วค่อยตามด้วยชื่อ Mutators ที่ต้องการและต่อท้ายด้วย Attribute สามารถเขียนได้ตามตัวอย่าง ดังนี้

เปิดไฟล์ User.php ใน app\Http\Models\...แล้วเพิ่มเมธอด ดังนี้



เวลาเรียกใช้งาน ให้เรียกที่คอนโทรลเลอร์ โดยให้ตัดคำว่า set และ Attribute ออกไป เช่น

```
$user = User::find(9);
$user->staffname = 'SATOSHI'      //Strtolower
$user->password = 'password' //Hash Password
```

กำหนดความสัมพันธ์ของตารางด้วย Eloquent Relationships

การจัดการเกี่ยวกับความสัมพันธ์ของตารางนั้นมีความสำคัญอย่างมาก การกำหนดความของความสัมพันธ์ของ Eloquent นั้นเป็นการกำหนดว่ามีตารางใดบ้างในฐานข้อมูลที่มีความสัมพันธ์กัน ซึ่งความสัมพันธ์ของข้อมูลมีหลายรูปแบบด้วยกัน ขึ้นอยู่กับแต่ละตารางและการออกแบบระบบฐานข้อมูล การสร้างความสัมพันธ์ให้กับแต่ละตารางนั้นจะสร้าง เมธอดขึ้นที่ Model ที่ต้องการสร้างความสัมพันธ์กัน ต้องมีการระบุคีย์หลัก (Primary Key) เพื่อใช้ในการอ้างอิงความสัมพันธ์ของตาราง รูปแบบความสัมพันธ์ มีดังนี้

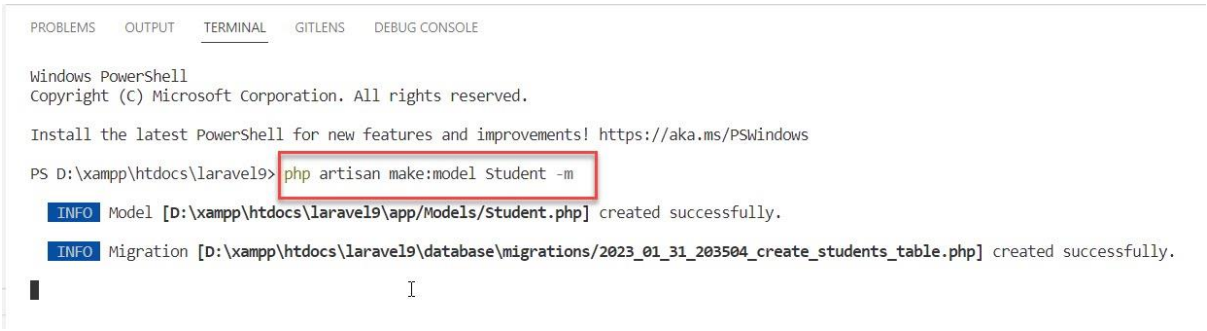
One to One Relationship คือ ความสัมพันธ์แบบ 1 ต่อ 1

ความสัมพันธ์แบบหนึ่งต่อหนึ่ง เป็นการกำหนดความสัมพันธ์ของตารางแบบหนึ่งเรคคอร์ดสัมพันธ์กับอีกหนึ่งเรคคอร์ดของอีกตาราง การกำหนดความสัมพันธ์จะใช้คำสั่ง hasOne() และระบุคีย์นอกสำหรับอ้างอิงในอีกตาราง ลองมาดูตัวอย่างการสร้างและใช้งานกันเลย จะหลายขั้นตอนหน่อยนะครับ โดยตัวอย่างนี้จะเป็นสร้างเริ่มต้นสร้างตั้งแต่แรกเลย จะทำส่วนไหนก่อนก็ได้ เพื่อความเข้าใจให้ทำตามตัวอย่างก่อนครับ

1.ให้สร้างตาราง students และสร้าง Model Student เพิ่มโดยใช้คำสั่ง migration ดังนี้

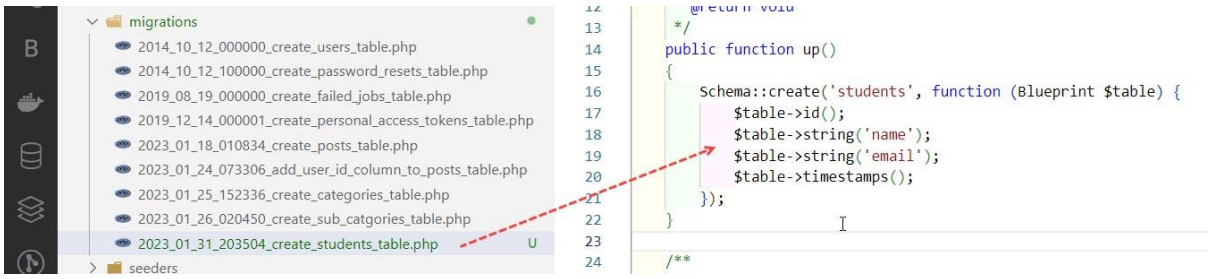
```
php artisan make:model Student -m    //สร้างโมเดลและสร้าง migration
```

กดปุ่ม ctrl+c เพื่อสั่งหยุด server ก่อน แล้วพิมพ์คำสั่งที่ Terminal ตามภาพ



จะได้โมเดล ชื่อ Student.php ถูกสร้างขึ้นที่โฟลเดอร์ app\Models\...และไฟล์ migration สำหรับสร้างตารางและกำหนดโครงสร้างฟิวด์ ในโฟลเดอร์ database\migrations\xxxx_create_students_table..

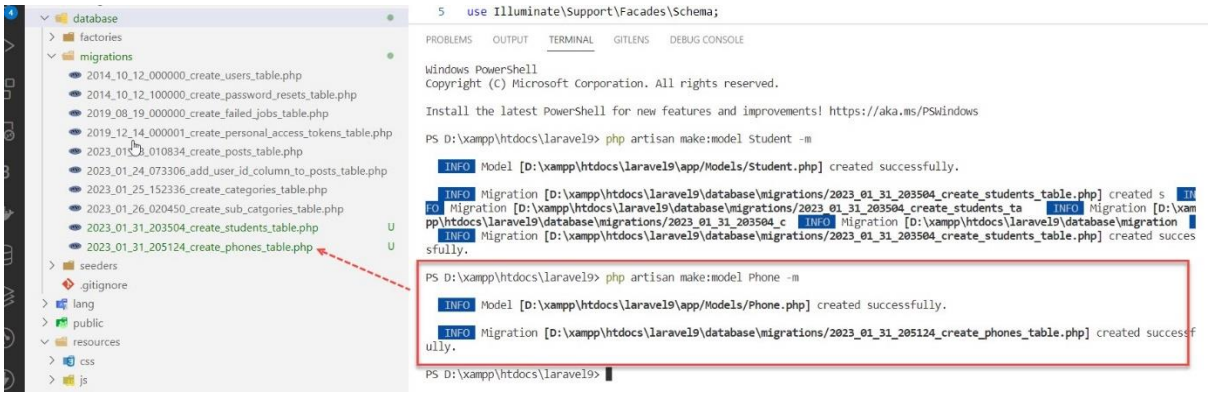
2.เปิดไฟล์ xxxx_create_student_table.php แล้วเพิ่มโครงสร้างฟิวด์ ดังนี้ เสร็จแล้วบันทึกไฟล์



3.ให้สร้างตาราง phones และสร้าง Model Phone เพิ่มโดยใช้คำสั่ง migration ดังนี้

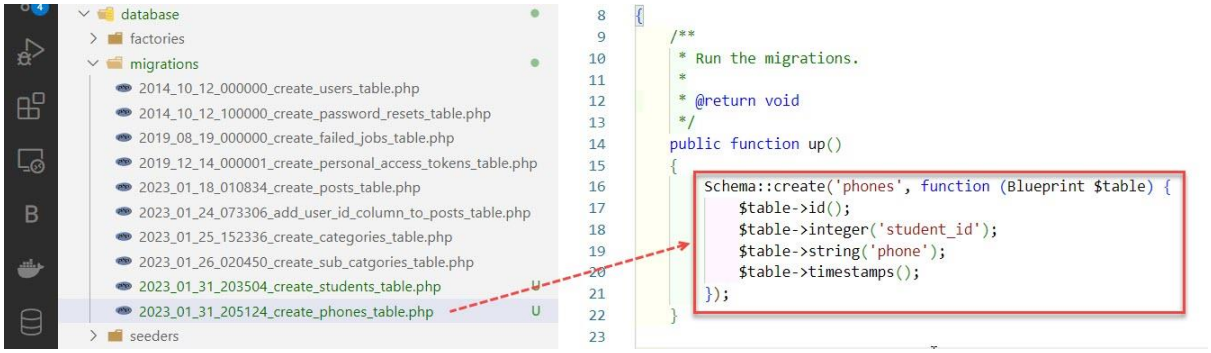
```
php artisan make:model Phone -m
```

พิมพ์คำสั่งที่ Terminal ตามภาพ



จะได้โมเดล ชื่อ Phone.php ถูกสร้างขึ้นที่โฟลเดอร์ app\Models\...และไฟล์ migration สำหรับสร้างตารางและกำหนดโครงสร้างฟิวด์ ในโฟลเดอร์ database\migrations\xxxx_create_ phones_table..

4.เปิดไฟล์ xxxx_create_phones_table.php แล้วเพิ่มโครงสร้างฟิวด์ ดังนี้ เสร็จแล้วบันทึกไฟล์



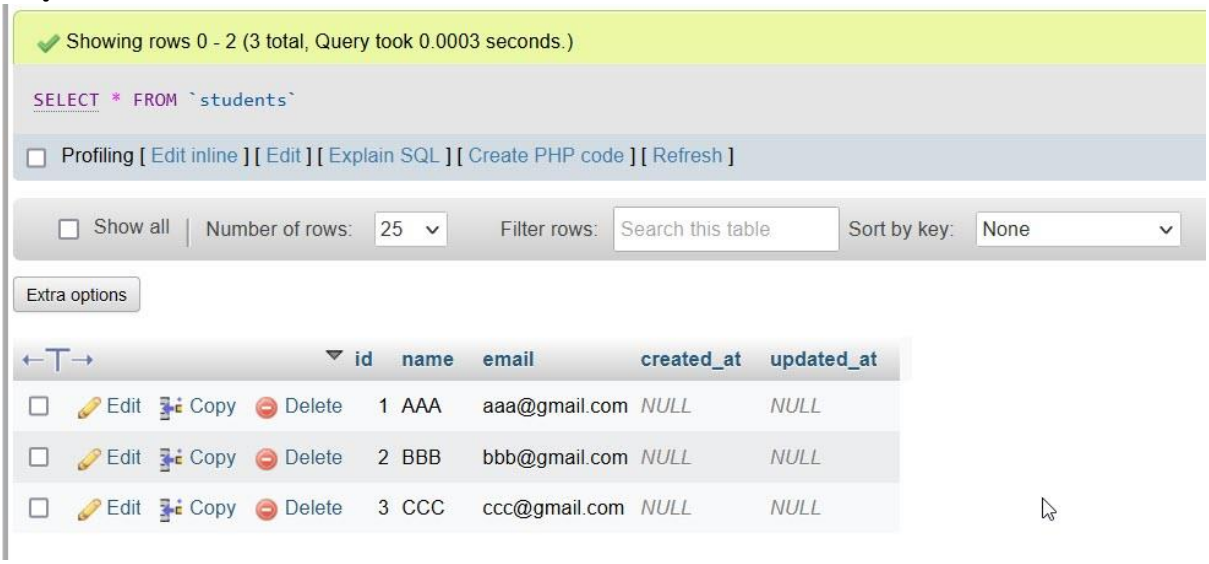
5. สั่งรันไฟล์ migration เพื่อสร้างตารางลงในฐานข้อมูล โดยพิมพ์คำสั่ง ดังนี้



6.ตรวจสอบข้อมูลตาราง students และ phones ใน โปรแกรม phpMyAdmin

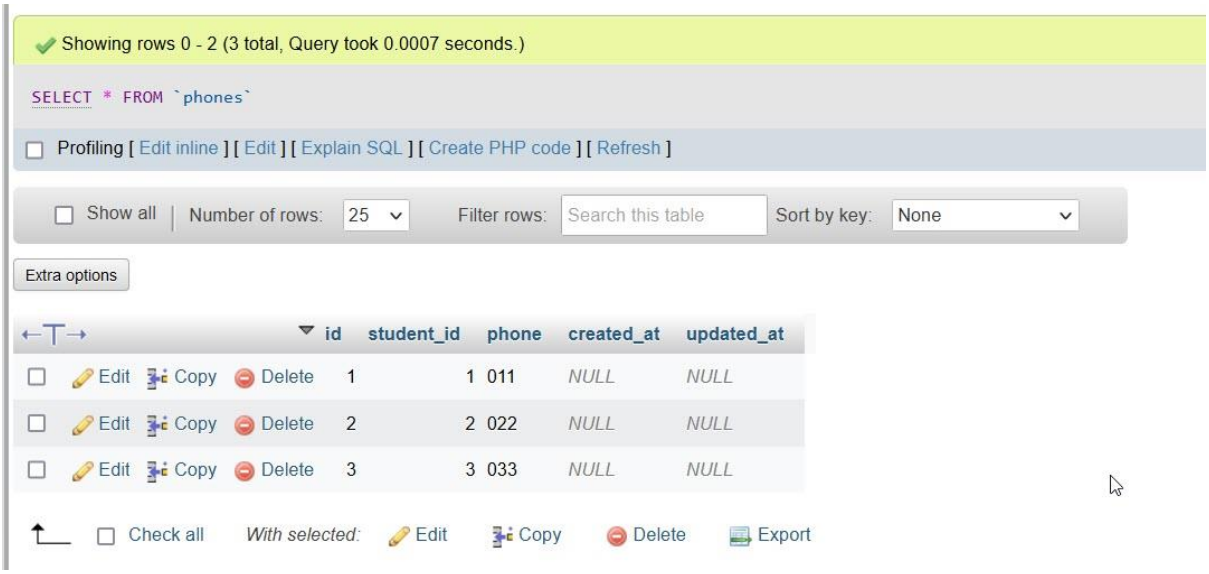


7. เพิ่มข้อมูลเข้าไปในตาราง students (คลิกที่ตาราง students แล้วเลือกที่เมนู insert แล้วเพิ่มข้อมูลลงไปสัก 3 รายการ)



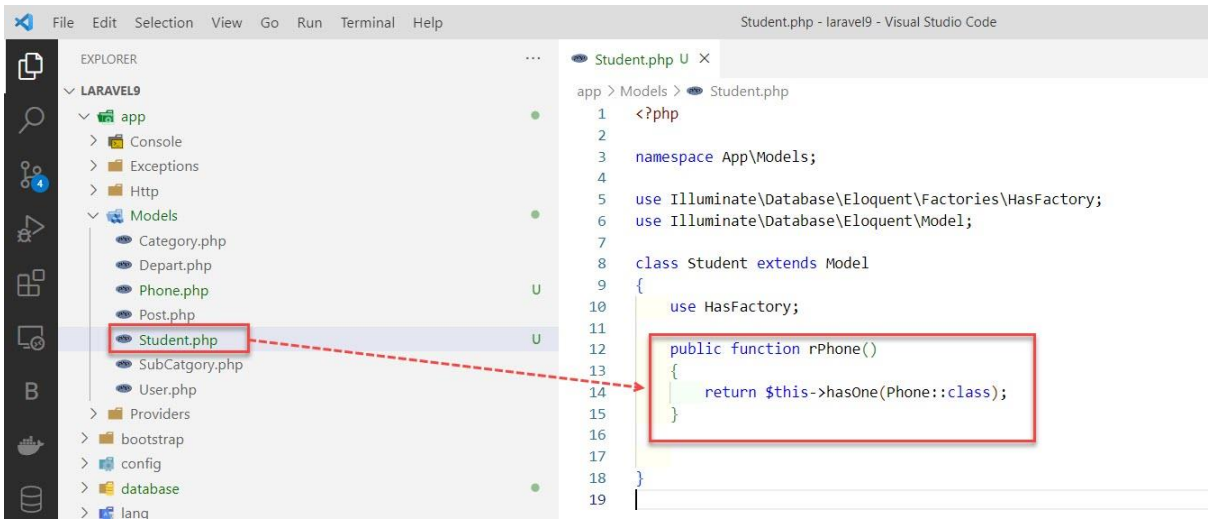
ให้ลองเพิ่มตามนี้ดูก่อนนะครับ

8. เพิ่มข้อมูลเข้าไปในตาราง phones (คลิกที่ตาราง phones แล้วเลือกที่เมนู insert แล้วเพิ่มข้อมูลลงไปสัก 3 รายการ)



9.กำหนดความสัมพันธ์แบบหนึ่งต่อหนึ่งในโมเดล Student โดยใช้ hasOne()

เปิดไฟล์ Student.php ในโฟลเดอร์ app\Models\... เขียนโค้ดดังนี้



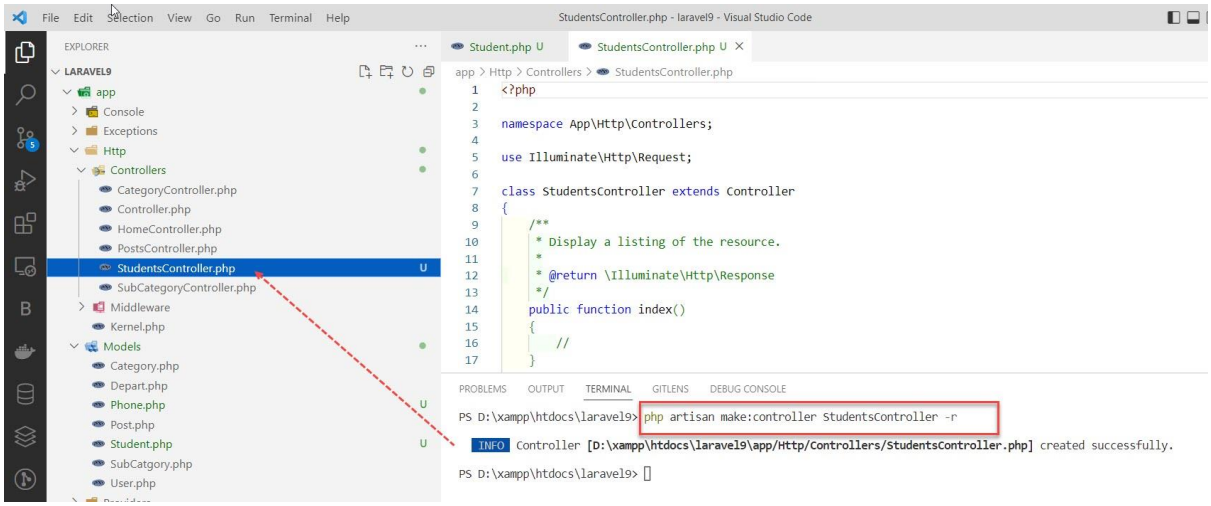
จากตัวอย่างโค้ด จะเป็นการเชื่อมความสัมพันธ์แบบ หนึ่งต่อหนึ่ง ไปยังตาราง phones ซึ่งตาราง phones ต้องเก็บ id ของตาราง students ด้วยเสมอ นั่นคือ student_id

10. สร้าง StudentsController.php เพิ่ม โดยใช้คำสั่ง

```
php artisan make:controller StudentsController -r    //RESTful Controller
```

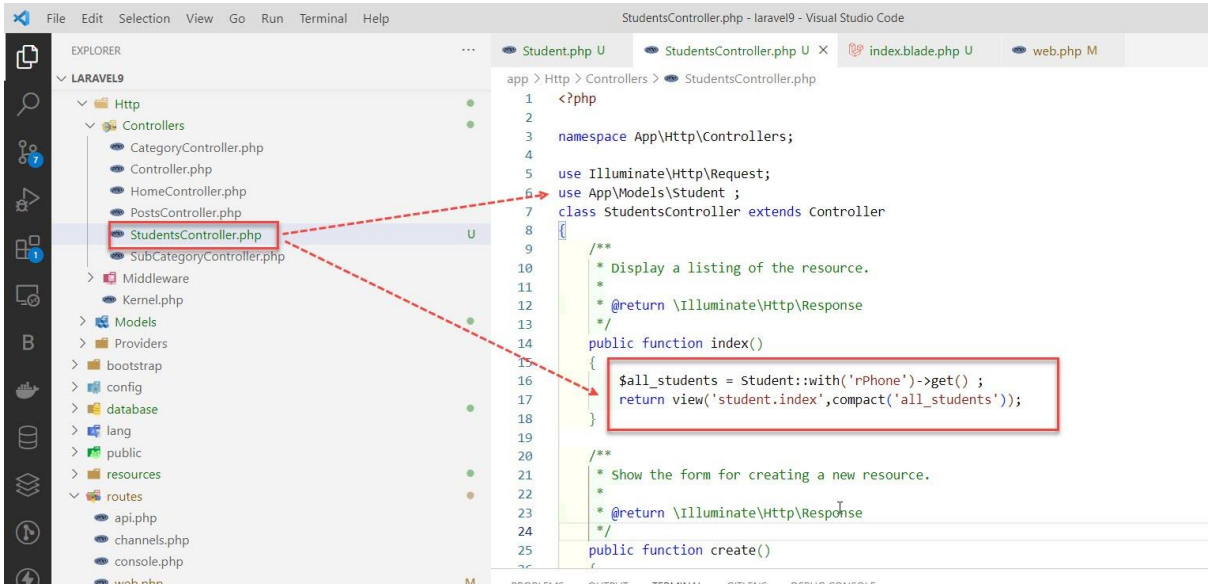
-r หรือ -resource เป็นการสร้างคอนโทรลเลอร์ในรูปแบบของ RESTful

พิมพ์คำสั่งที่ Terminal ตามภาพ



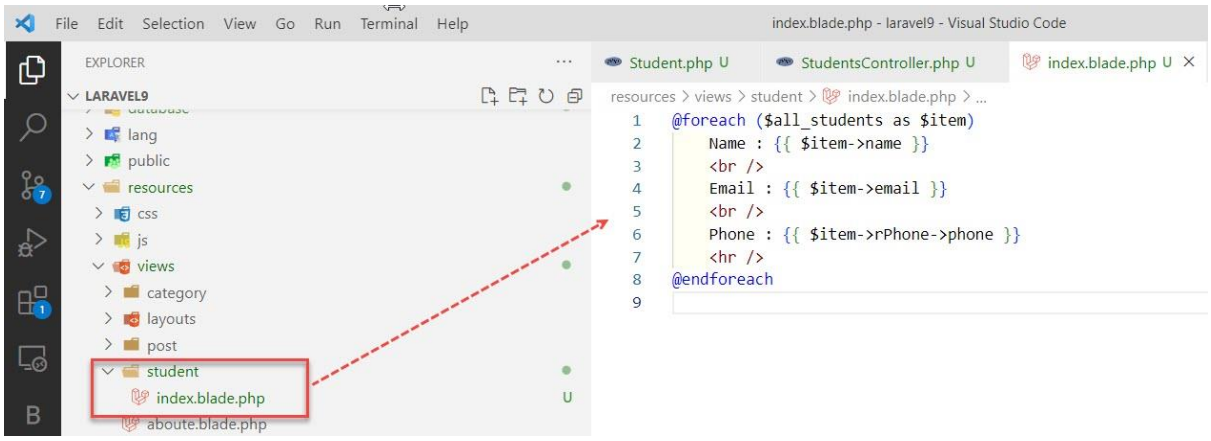
11.เปิดไฟล์ StudentsController.php แล้วเขียนโค้ด ในเมธอด index() ดังนี้

อย่าลืม use App\Models\Student ; มาใช้งานด้วยนะ

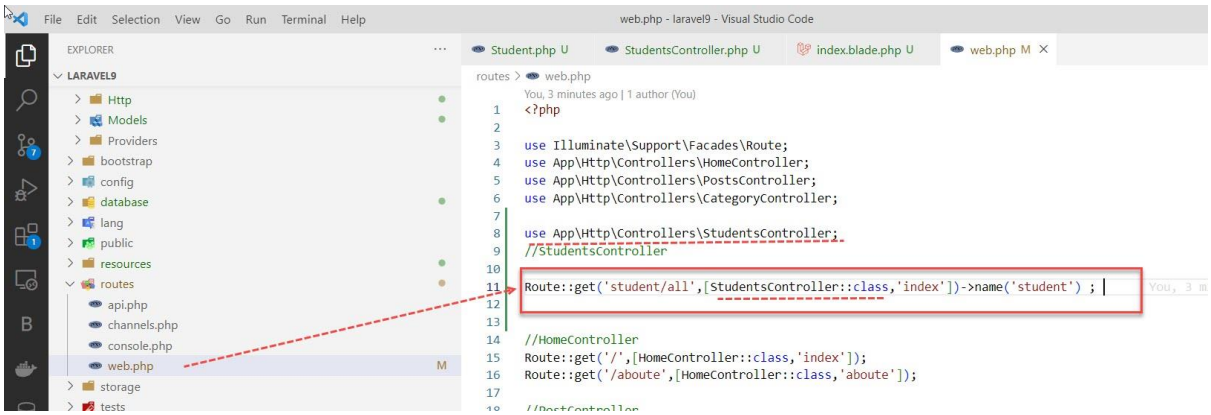


จะใช้ with() เป็นตัวกำหนดความสัมพันธ์ โดยเวลาเรียกใช้ก็เรียกผ่าน rPhone แล้วระบุฟิวด์ของตารางที่ต้องการ

12.สร้างโฟลเดอร์ student ไว้ที่ views แล้วเพิ่มไฟล์ index.blade.php (resources\views\student\...) เขียนโค้ดสำหรับแสดงข้อมูล (ค่อยไปปรับแต่งความสวยงามทีหลังได้ เขียนแบบลูกทุ่ง ๆ ไปก่อน ครับ) ดังนี้



13.สร้าง routes สำหรับเป็นเส้นทางเข้าถึงหน้าเว็บ ที่ไฟล์ routes\web.php ดังนี้

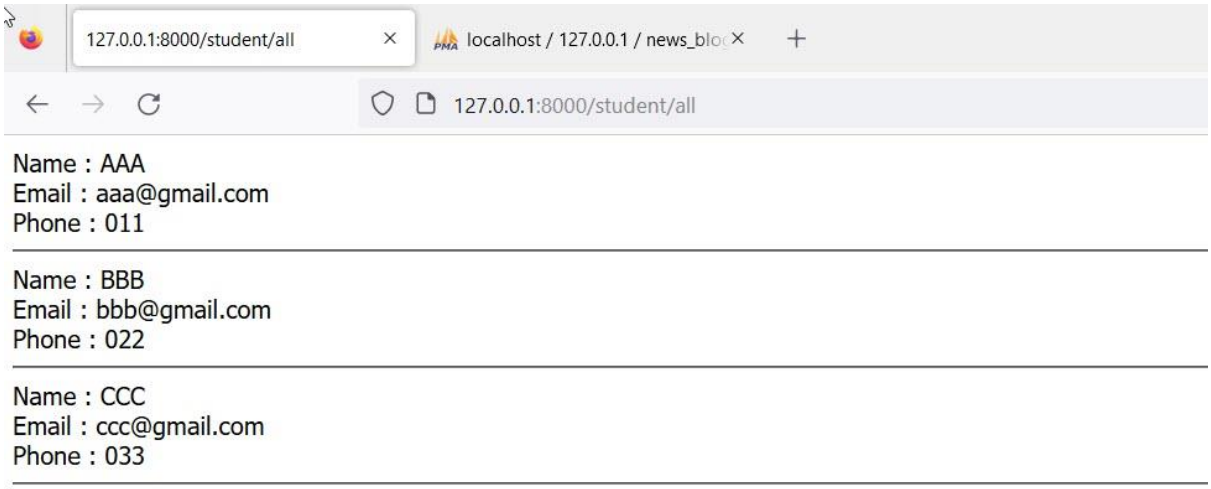


มาถึงจุดนี้แล้วถือว่าเสร็จกระบวนการสร้างทั้งหมดแล้วครับ

ให้ใช้คำสั่ง php artisan serv เพื่อสั่งให้ Server ทำงาน



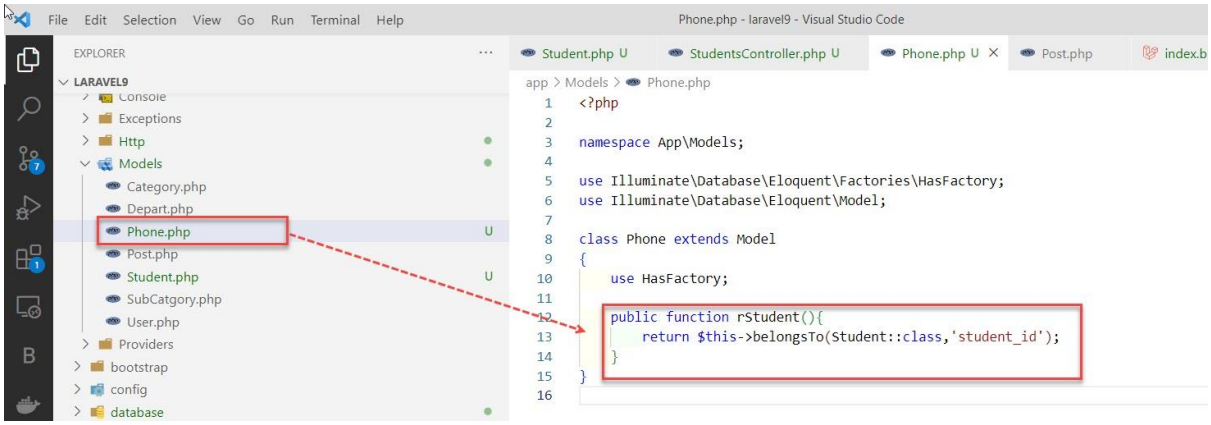
14.ทดสอบดูการแสดงผล พิมพ์ localhost:8000/student/all ที่ Browser



One to One Reverse Relationship

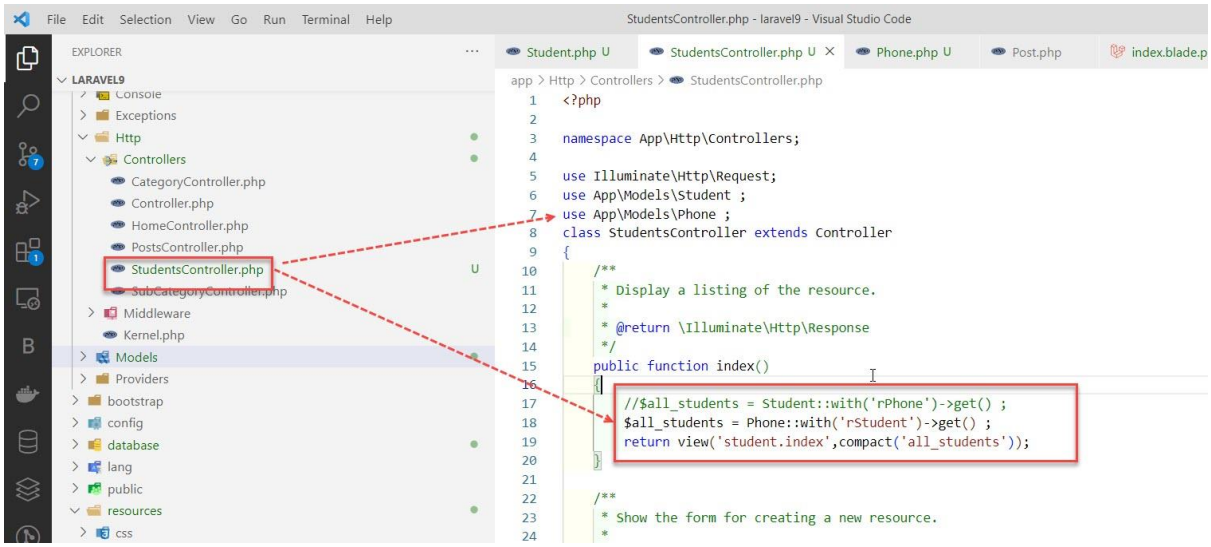
Reverse เป็นความสัมพันธ์แบบย้อนกลับ จะทำตรงกันข้าม คือ ไปกำหนดความสัมพันธ์ที่โมเดล Phone โดยใช้ belongsTo() พร้อมระบุ FK

1.เปิดไฟล์ Phone.php ในโฟลเดอร์ app\Models\... เขียนโค้ดดังนี้

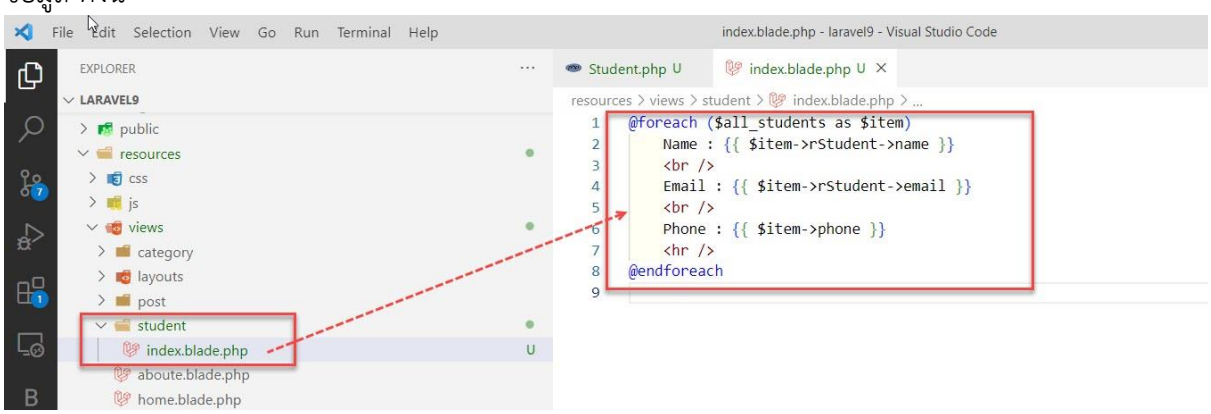


จากตัวอย่างโค้ด จะเป็นการเชื่อมความสัมพันธ์แบบ หนึ่งต่อหนึ่ง ไปยังตาราง students ซึ่งตาราง phones ต้องเก็บ id ของตาราง students ด้วยเสมอ นั่นคือ student_id

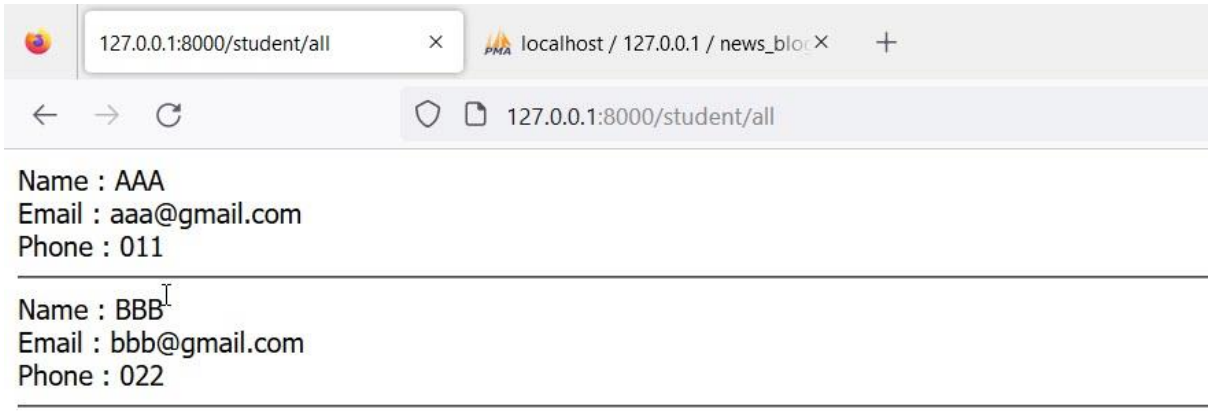
2.เปิดไฟล์ StudentsController.php แล้วแก้ไขโค้ด ในเมธอด index() ดังนี้



3.เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources\views\student\... แก้ไขโค้ดสำหรับแสดงข้อมูล ดังนี้

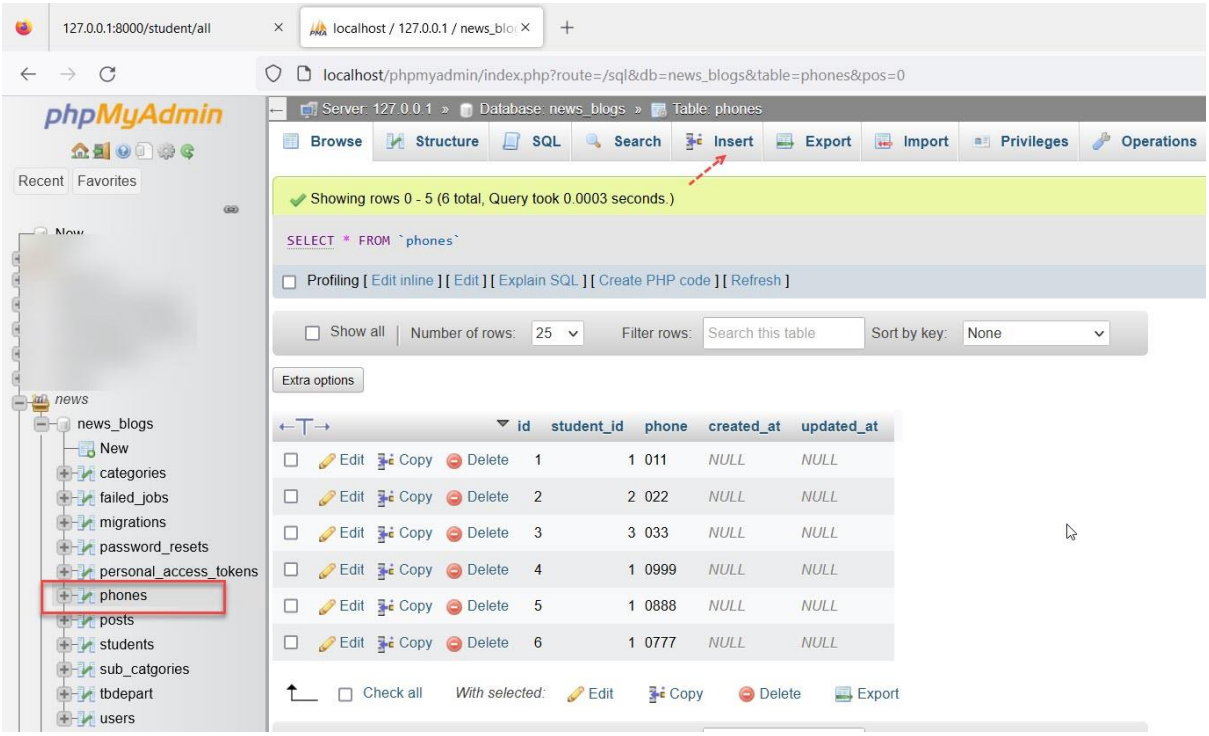


4.ทดสอบดูการแสดงผล พิมพ์ localhost:8000/student/all ที่ Browser อีกครั้ง



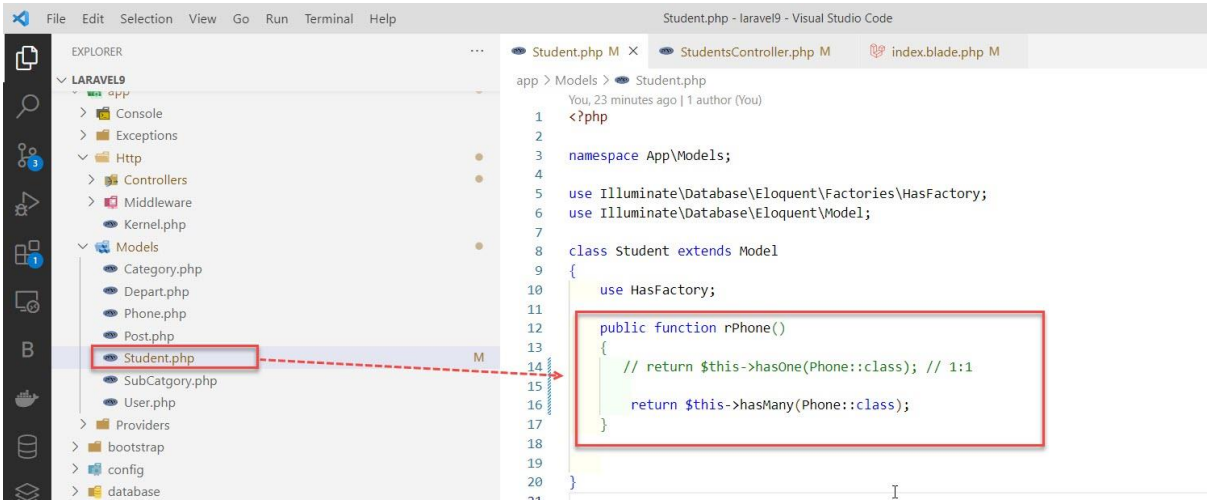
One to Many Relationship คือ ความสัมพันธ์แบบ 1 ต่อ กลุ่ม

เป็นความสัมพันธ์ที่ใช้บ่อย One to Many คือความสัมพันธ์ของตารางข้อมูลแบบ 1 ต่อ กลุ่ม หมายความว่า ข้อมูล 1 เรคคอร์ด ของตารางหนึ่ง มีความสัมพันธ์กับข้อมูลหลาย ๆ เรคคอร์ดของอีกตาราง เช่น ผู้เขียนบทความ 1 คนหรือ 1 user สามารถเขียนบทความได้มากกว่า 1 บทความ หรือ คนหนึ่งคนสามารถมีหมายเลขโทรศัพท์ได้หลายเบอร์ ก็เป็นได้ ตัวอย่างการเขียน จะใช้ตาราง students และตาราง phones เดิมที่มีอยู่แล้วนะครับ ก่อนอื่นให้เข้าไปที่โปรแกรม phpMyAdmin แล้วเพิ่มข้อมูลลงในตาราง phones ดังนี้ (คลิกที่ตาราง phones แล้วกดที่เมนู Insert แล้วเพิ่มข้อมูล)

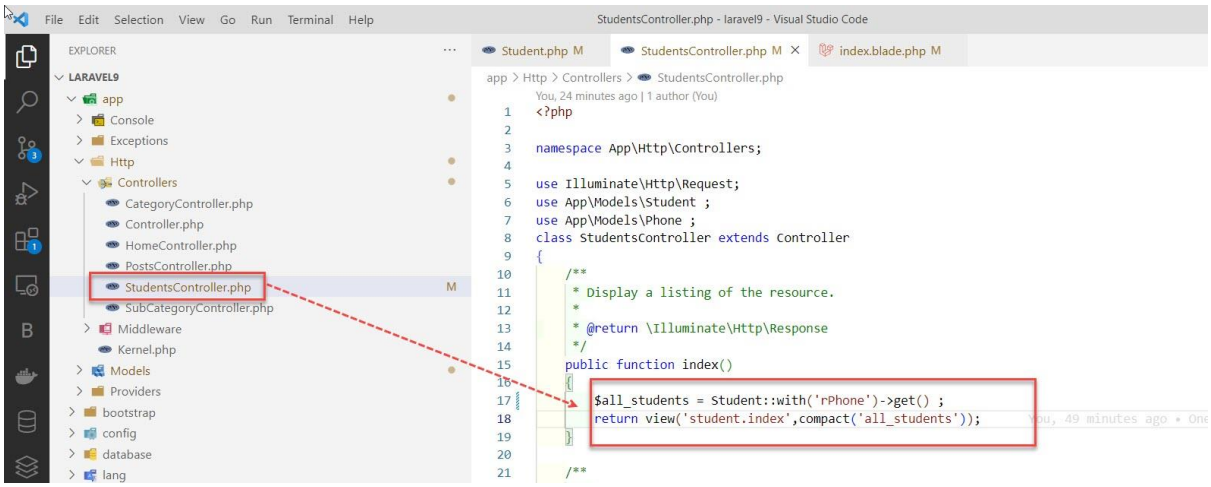


ก็กลายเป็นว่า คนหนึ่งคนสามารถมีเบอร์โทรศัพท์หลายเบอร์แล้ว

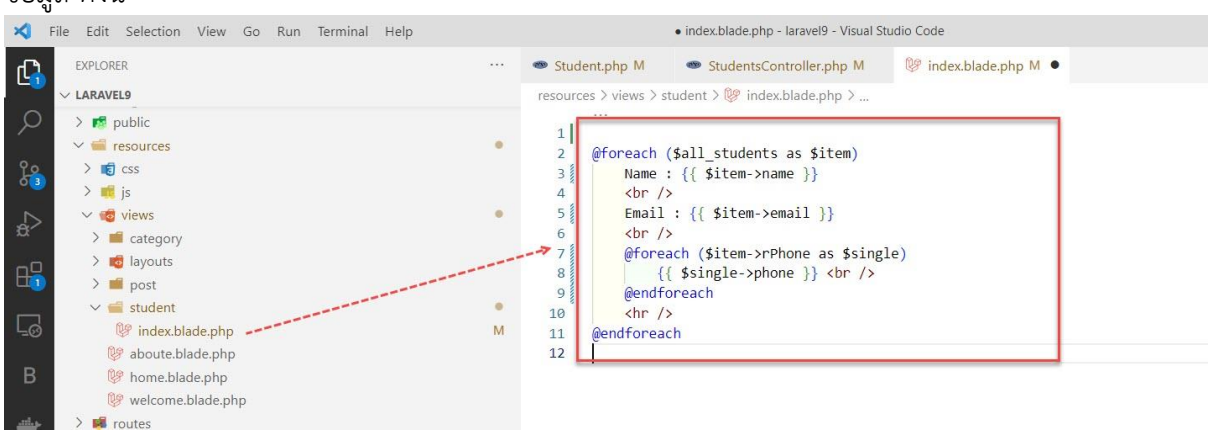
1.เปิดไฟล์ Student.php ในโฟลเดอร์ app\Models\... เขียนโค้ดดังนี้



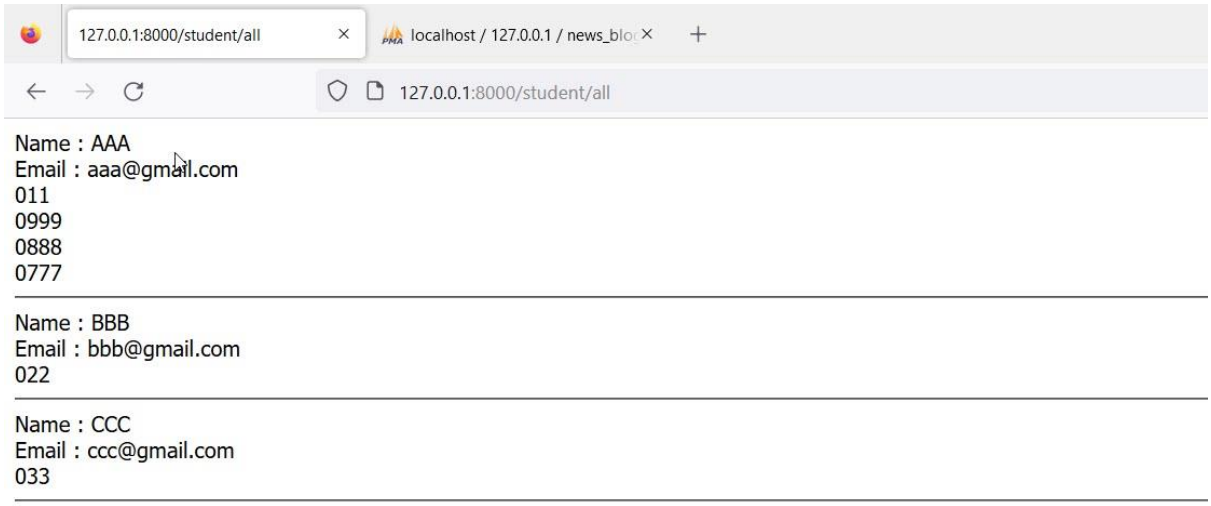
2.เปิดไฟล์ StudentsController.php แล้วแก้ไขโค้ด ในเมธอด index() ดังนี้



3.เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources\views\student\... แก้ไขโค้ดสำหรับแสดงข้อมูล ดังนี้



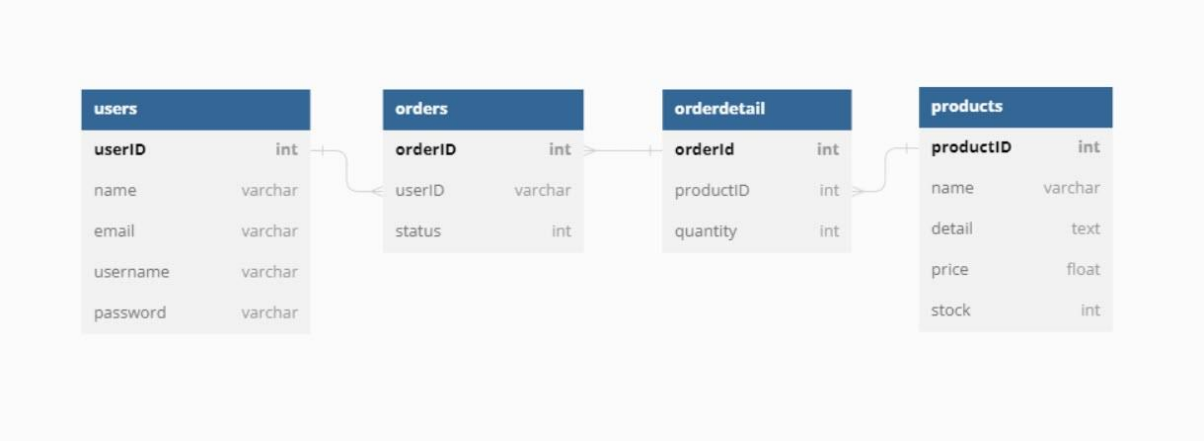
4.ทดสอบดูการแสดงผล พิมพ์ localhost:8000/student/all ที่ Browser อีกครั้ง



Many to Many Relationship คือ ความสัมพันธ์แบบ กลุ่ม ต่อ กลุ่ม

เป็นความสัมพันธ์ของตารางข้อมูลที่มีหลายเรคคอร์ดสัมพันธ์กับข้อมูลหลายเรคคอร์ดในอีกตาราง เช่น สินค้ามีผู้ซื้อได้หลายคน และแต่ละคนก็สามารถซื้อสินค้าได้หลายอย่าง (M:M)

หากตารางมีความสัมพันธ์แบบกลุ่มต่อกลุ่ม จะต้องมีตารางอีกตารางหนึ่งแยกออกมาเพื่อเก็บค่าของคีย์หลักทั้งสองตารางที่มีความสัมพันธ์กัน เช่น ตารางสมาชิกกับตารางสินค้า ที่สมาชิกหนึ่งคนสามารถซื้อสินค้าได้หลายชิ้น หรือตารางสินค้ากับตารางคำสั่งซื้อ ที่คำสั่งซื้อหนึ่งสามารถมีสินค้าได้หลายรายการ และสินค้าแต่ละชนิดสามารถอยู่ได้ในหลายคำสั่งซื้อ ซึ่งจะต้องมีตารางรายละเอียดคำสั่งซื้อขึ้นมาอีกหนึ่งตารางเพื่อเพื่อใช้เชื่อมความสัมพันธ์ของทั้งสองตาราง .



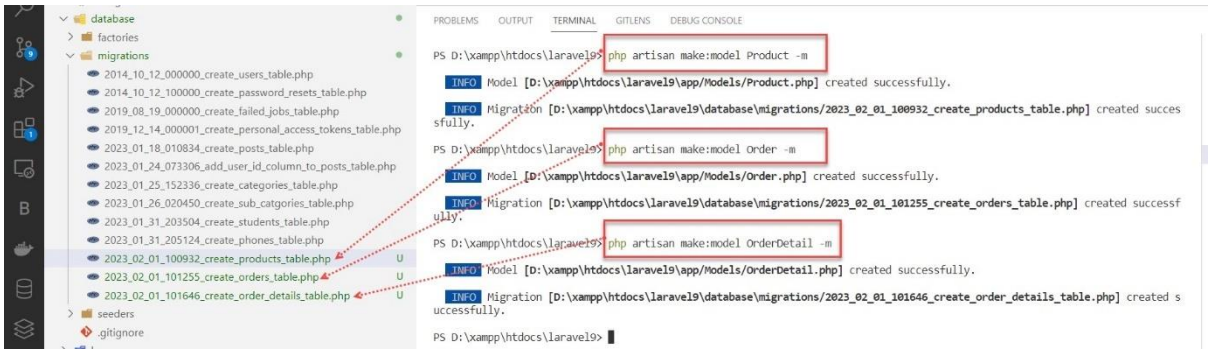
จากตารางมีตารางใหม่เกิดขึ้น 3 ตาราง ได้แก่ตาราง orders ไว้สำหรับเก็บข้อมูลคำสั่งซื้อ ตาราง orderdetail ไว้สำหรับเก็บรายละเอียดคำสั่งซื้อ และตาราง products ไว้สำหรับเก็บข้อมูลสินค้า

มาดูตัวอย่างและวิธีการสร้างความสัมพันธ์และกำหนดค่าคุณสมบัติความสัมพันธ์ของข้อมูลแบบกลุ่มต่อกลุ่ม กันเลยครับ

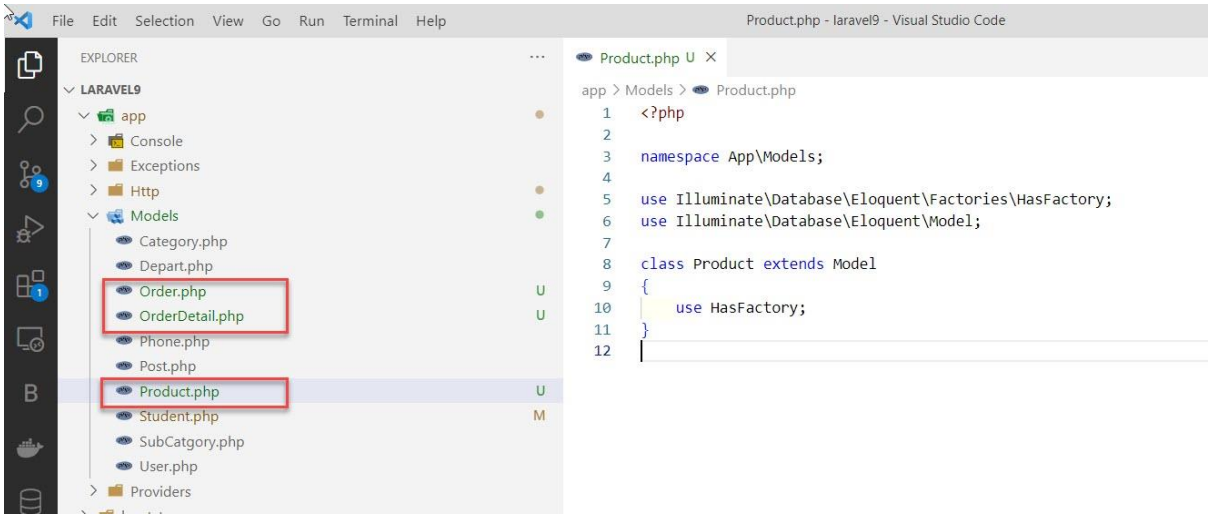
1.เริ่มต้นให้สร้างตาราง ทั้ง 3 ตารางเพิ่ม โดยวิธีการรันคำสั่งผ่าน Terminal ดังนี้

```
php artisan make:model Product -m

php artisan make:model Order -m

php artisan make:model OrderDetail -m
```

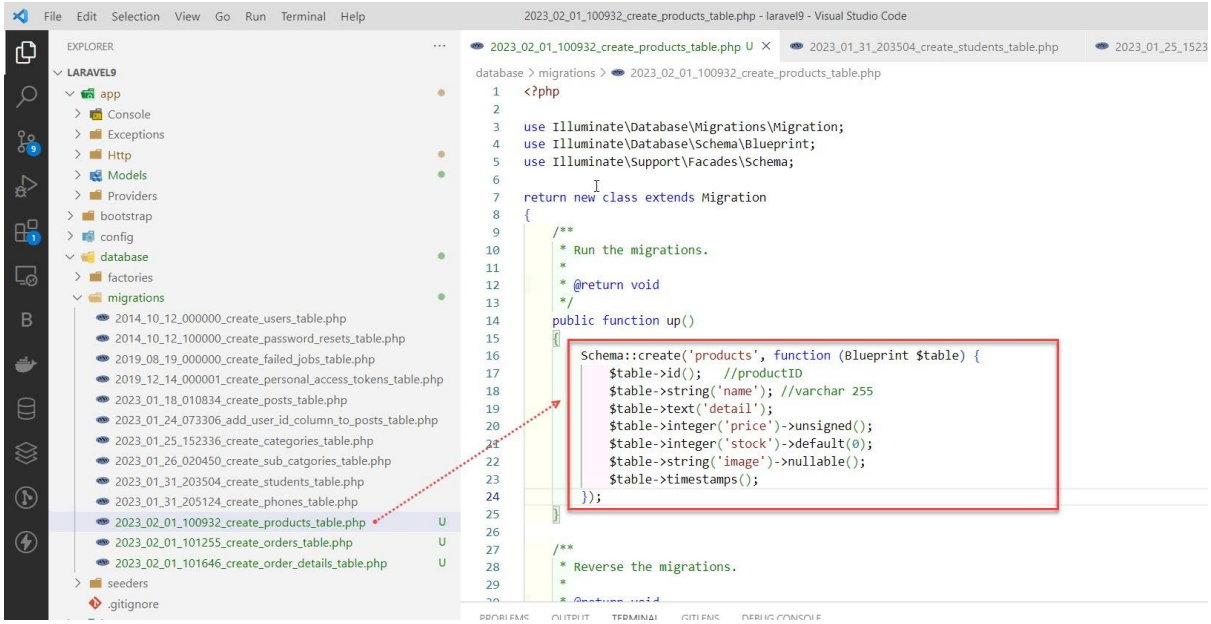
ซึ่ง 3 คำสั่งนี้เป็นคำสั่งสำหรับสร้างไฟล์ Model และไฟล์ Migrations เพื่อสร้างตารางและกำหนดโครงสร้างให้ตาราง



เมื่อรันคำสั่งทั้ง 3 คำสั่งผ่าน Terminal แล้ว ระบบจะสร้างไฟล์ Migrations ทั้ง 3 ไฟล์ไว้ในโฟลเดอร์ database/migrations/... และจะสร้างไฟล์ Models ทั้ง 3 ไฟล์ ไว้ในโฟลเดอร์ app/models/...

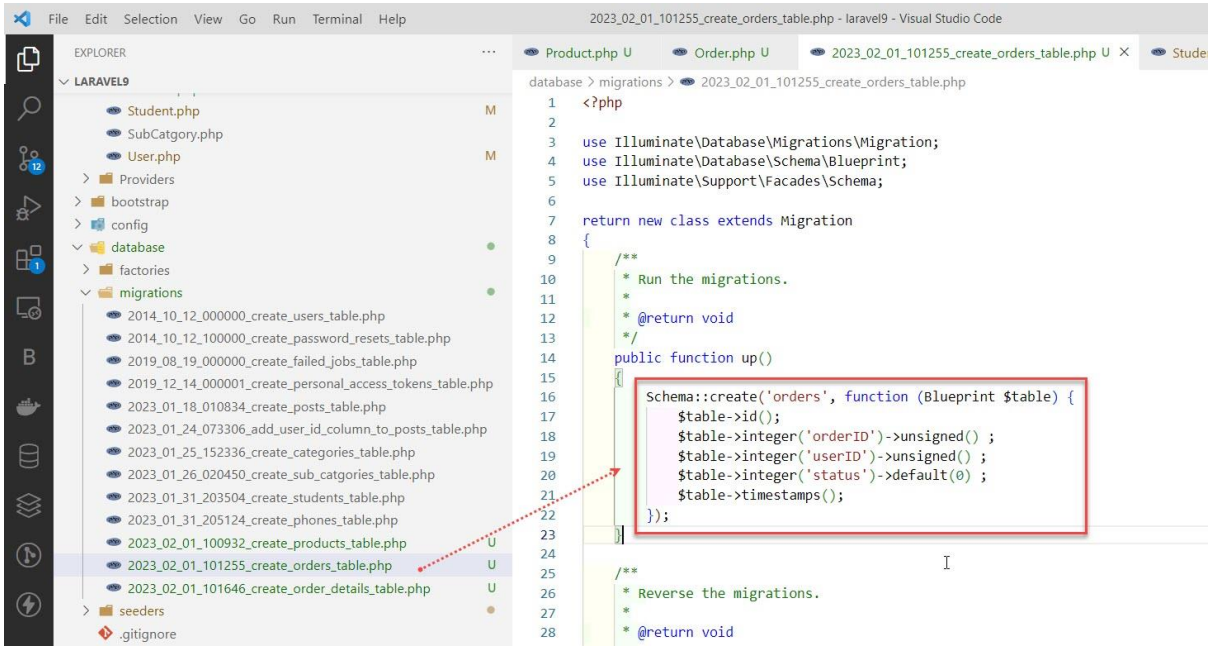


2.เปิดไฟล์ ...create_products_table.php ในโฟลเดอร์ database/migrations/ แล้วกำหนดรูปแบบของตารางข้อมูลสินค้า ซึ่งใช้จัดเก็บรหัสสินค้า ชื่อสินค้า รายละเอียดสินค้า ราคาสินค้า จำนวนสินค้าที่มีในสต๊อก ภาพสินค้า ดังนี้

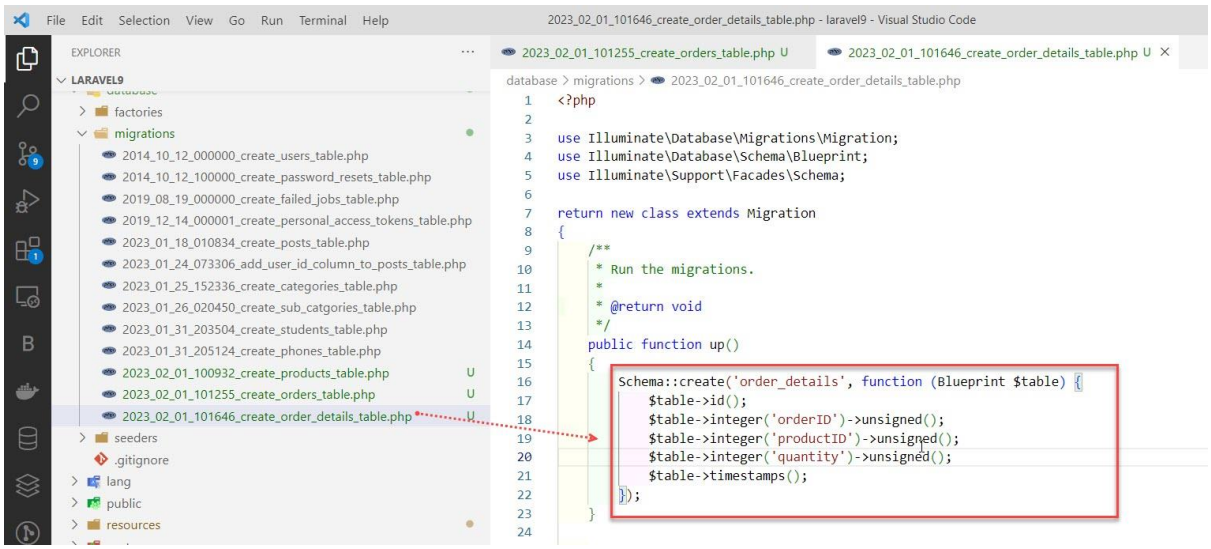


เปิดไฟล์ ...create_orders_table.php ในโฟลเดอร์ database/migrations/ แล้วกำหนดรูปแบบ

ของตารางข้อมูลคำสั่งซื้อ จัดเก็บรหัสการสั่งซื้อ รหัสผู้ซื้อ และสถานการสั่งซื้อ ดังนี้

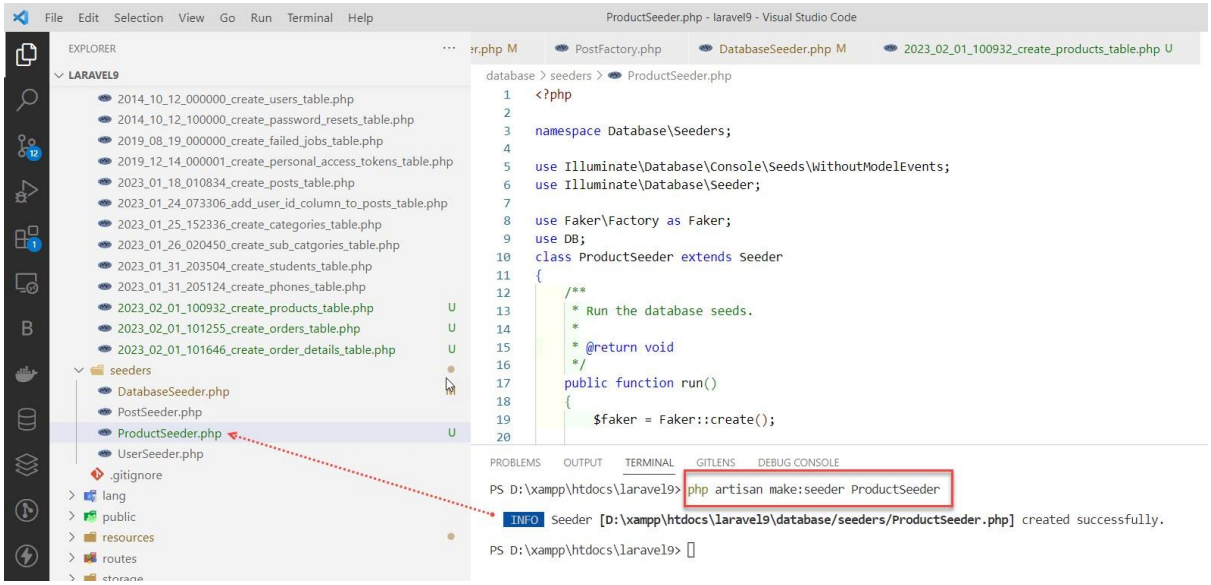


เปิดไฟล์ ...create_order_details_table.php ในโฟลเดอร์ database/migrations/ แล้วกำหนดรูปแบบของตารางข้อมูลรายละเอียดคำสั่งซื้อ จัดเก็บรหัสคำสั่งซื้อ รหัสสินค้าและจำนวนสินค้า ดังนี้

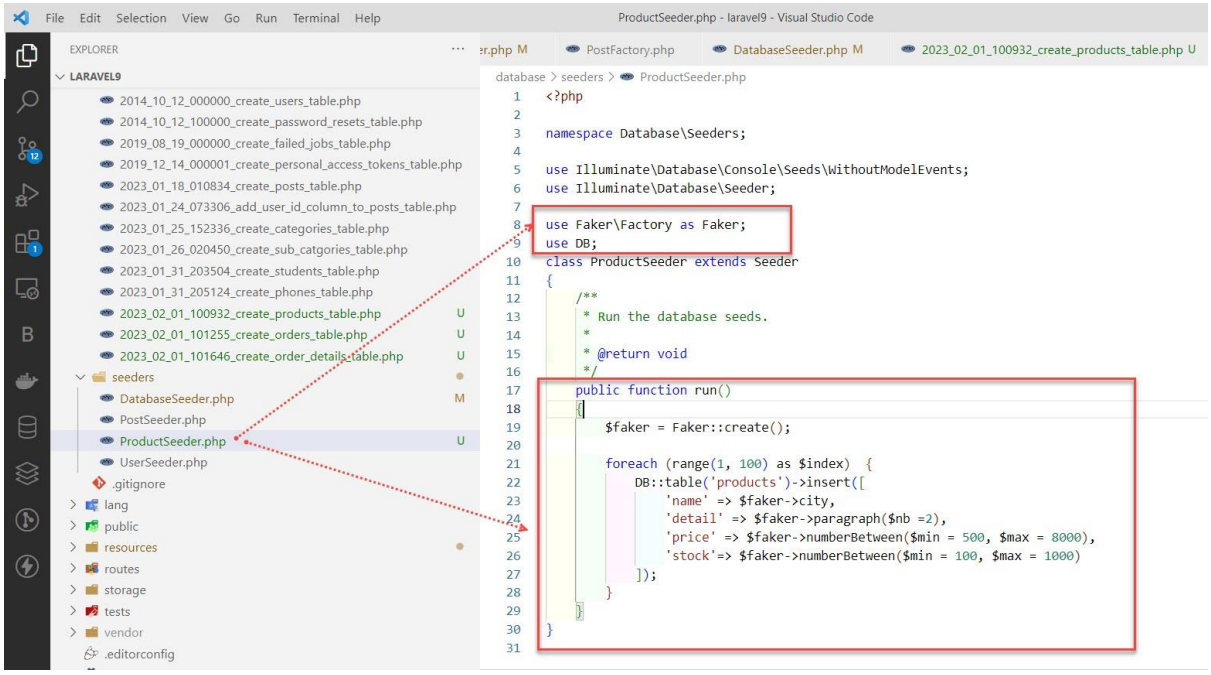


3.จากนั้นให้เตรียมข้อมูลตั้งต้นให้กับตารางสินค้า โดยรันคำสั่งผ่าน Terminal ดังนี้

```
php artisan make:seeder ProductSeeder  //พิมพ์คำสั่งแล้วกด Enter
```



เปิดไฟล์ ProductSeeder.php ในโฟลเดอร์ database/seed/ แล้วพิมพ์คำสั่งเพื่อเตรียมข้อมูลตั้งต้นให้กับตารางสินค้า ดังนี้



บันทึกไฟล์และรันคำสั่งผ่าน Terminal **migrate:refresh** เพื่อให้ลบตารางทั้งหมดและสร้าง migration ใหม่ ดังนี้

```
php artisan migrate:refresh
```

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE

PS D:\xampp\htdocs\laravel9> php artisan migrate:refresh

  INFO  Rolling back migrations.

  2023_02_01_101646_create_order_details_table ..... 21ms DONE
  2023_02_01_101255_create_orders_table ..... 6ms DONE
  2023_02_01_100932_create_products_table ..... 6ms DONE
  2023_01_31_205124_create_phones_table ..... 7ms DONE
  2023_01_31_203504_create_students_table ..... 7ms DONE
  2023_01_26_020450_create_sub_catgories_table ..... 6ms DONE
  2023_01_25_152336_create_categories_table ..... 7ms DONE
  2023_01_24_073306_add_user_id_column_to_posts_table ..... 14ms DONE
  2023_01_18_010834_create_posts_table ..... 7ms DONE
  2019_12_14_000001_create_personal_access_tokens_table ..... 8ms DONE
  2019_08_19_000000_create_failed_jobs_table ..... 8ms DONE
  2014_10_12_100000_create_password_resets_table ..... 7ms DONE
  2014_10_12_000000_create_users_table ..... 8ms DONE

  INFO  Running migrations.
```

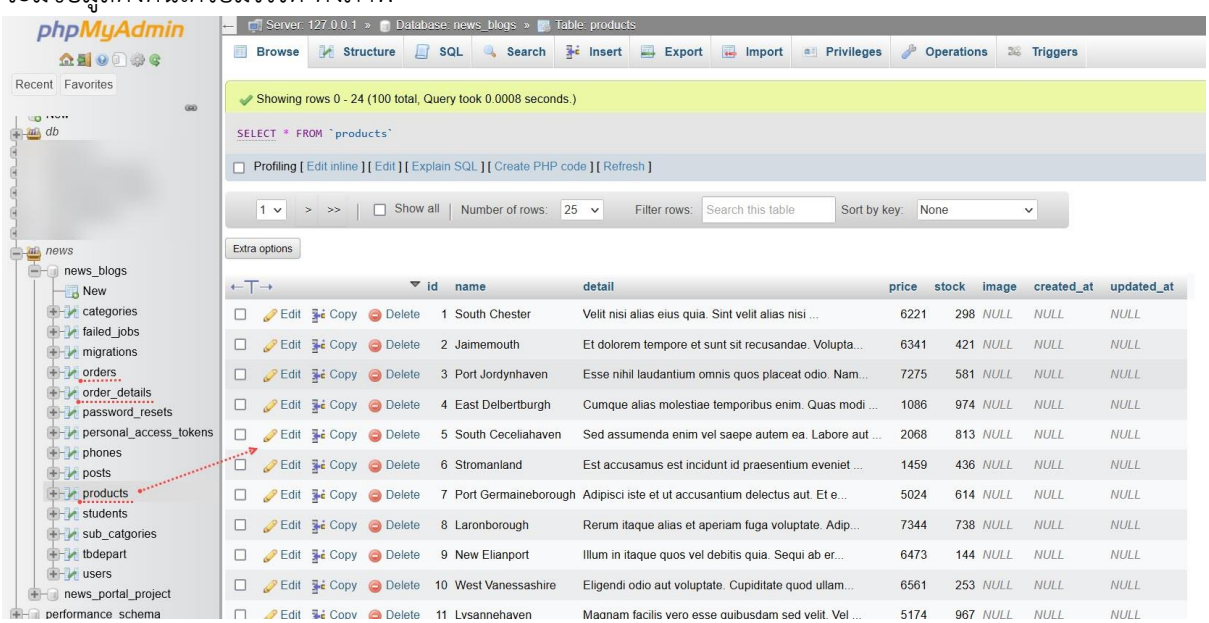
จากนั้นรันคำสั่ง db:seed เพื่อให้ระบบเตรียมข้อมูลให้กับตารางใหม่

```
php artisan db:seed
```

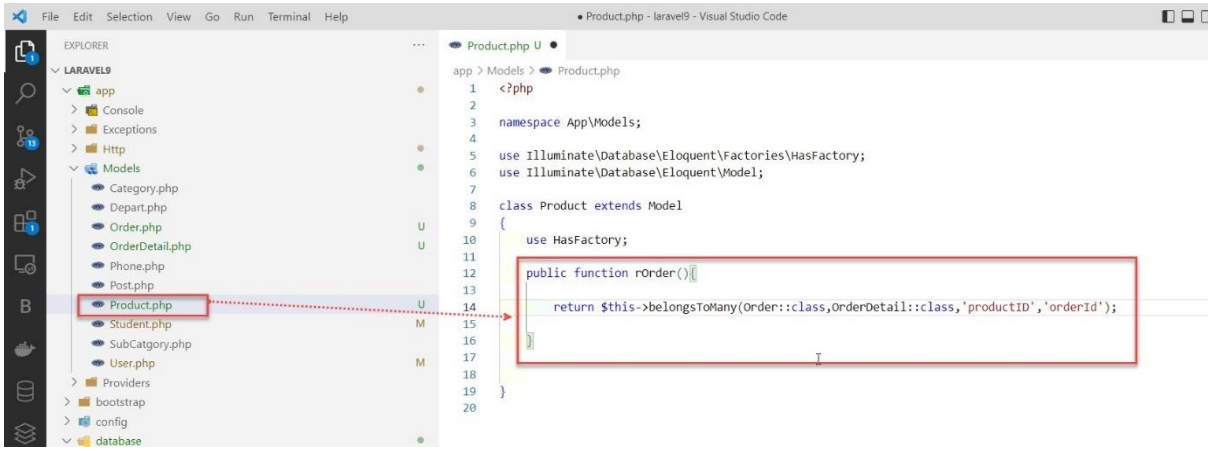
```
PS D:\xampp\htdocs\laravel9> php artisan db:seed

  INFO  Seeding database.

  Database\Seeders\UserSeeder ..... RUNNING
  Database\Seeders\UserSeeder ..... 386.26 ms DONE

  Database\Seeders\PostSeeder ..... RUNNING
  Database\Seeders\PostSeeder ..... 477.91 ms DONE

  Database\Seeders\ProductSeeder ..... RUNNING
  Database\Seeders\ProductSeeder ..... 220.80 ms DONE

PS D:\xampp\htdocs\laravel9>
```

เปิดโปรแกรม phpMyAdmin เพื่อตรวจสอบตารางว่าถูกสร้างให้ครบหรือไม่

เมื่อทำตามขั้นตอนที่ผ่านมาเรียบร้อยแล้ว จะมีตารางเพิ่มขึ้นมาให้เรา และคลิกดูที่ตาราง productsก็จะมีข้อมูลตั้งต้นเตรียมไว้ให้ ดังภาพ



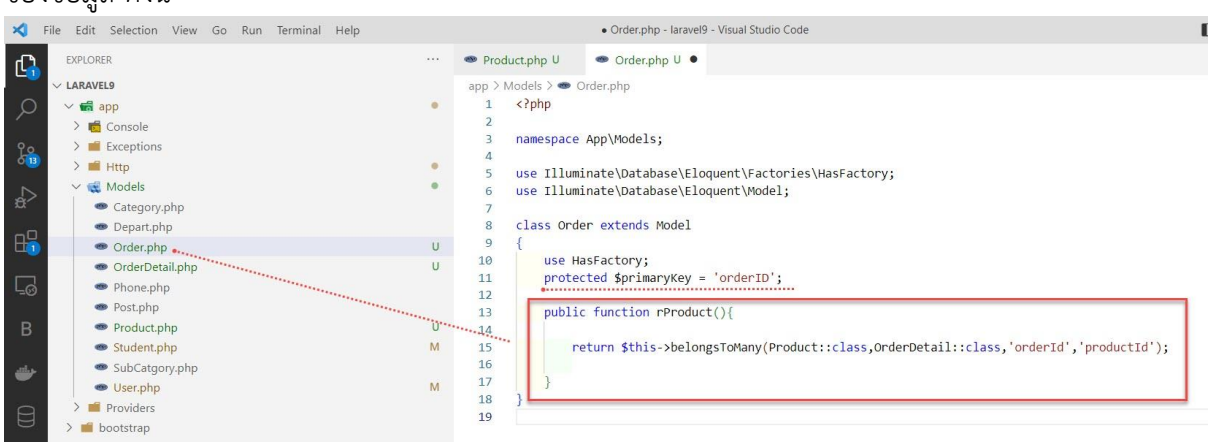
4. เปิดไฟล์ Product.php ในโฟลเดอร์ app/models/.. เพื่อกำหนดคุณสมบัติและเชื่อมโยงความสัมพันธ์ของข้อมูล ไฟล์ Product.php เป็น Model สำหรับจัดการเกี่ยวกับข้อมูลสินค้า โดยจะเชื่อมความสัมพันธ์กับไฟล์ Order.php ซึ่งเป็น Model สำหรับการจัดการข้อมูลสั่งซื้อ โดยเพิ่มเมธอดสำหรับจัดการความสัมพันธ์ ดังนี้



การเชื่อมความสัมพันธ์ระหว่างตารางแบบกลุ่มต่อกลุ่ม จะใช้คำสั่ง belongsToMany() แล้วระบุพารามิเตอร์เข้าไป คือ Order::class , OrderDetail::class ซึ่งเป็นโมเดลอ้างอิงการเชื่อมโยงความสัมพันธ์ทั้งสองตาราง คือตาราง orders และ ตาราง orderdetails จากนั้นก็ระบุคีย์หลักของสินค้าและคีย์หลักของคำสั่งซื้อ คือ productID และ orderID

ความสัมพันธ์แบบกลุ่มต่อกลุ่มจะต้องกำหนดความสัมพันธ์ทั้งฝั่ง คือ สินค้าสามารถตรวจสอบได้ว่าอยู่ในคำสั่งซื้อใด หรือ คำสั่งซื้อสามารถตรวจสอบได้ว่ามีสินค้าใด ซึ่งจะต้องกำหนดความสัมพันธ์กับอีกฝั่งของนั่นคือ Order

เปิดไฟล์ Order.php ในโฟลเดอร์ app/models/.. เพื่อกำหนดคุณสมบัติและเชื่อมโยงความสัมพันธ์ของข้อมูล ดังนี้



จากภาพเป็นการสร้างเมธอด rProduct สำหรับการเชื่อมความสัมพันธ์โดยใช้คำสั่ง belongsToMany() โดยกำหนดให้มีความสัมพันธ์กับโมเดล Product โดยมี orderdetails เป็นตารางเก็บข้อมูลความสัมพันธ์ และกำหนด orderID เป็นคีย์หลักของตาราง และใช้ productID เป็นคีย์อ้างอิงสินค้า

5.ให้ทดลองเพิ่มข้อมูลลงในตาราง orders และ orderdetails และอ้างอิงรหัสของแต่ละตารางเพื่อใช้ทดสอบในการเขียนการแสดงผล โดยให้เพิ่มผ่านโปรแกรม phpMyAdmin ดังนี้

ตาราง users (สมาชิก)

	id	name	email	email_verified_at
Edit Copy Delete	1	Admin Laravel 9	laravel9@gmail.com	NULL
Edit Copy Delete	2	Admin Yii	yii@gmail.com	NULL

ตาราง products (สินค้า)

	id	name	detail	price	stock	image	created_at
Edit Copy Delete	1	South Chester	Velit nisi alias eius quia. Sint velit alias nisi ...	6221	298	NULL	NULL
Edit Copy Delete	2	Jaimemouth	Et dolorem tempore et sunt sit recusandae. Volupta...	6341	421	NULL	NULL
Edit Copy Delete	3	Port Jordynhaven	Esse nihil laudantium omnis quos placeat odio. Nam...	7275	581	NULL	NULL
Edit Copy Delete	4	East Delbertburgh	Cumque alias molestiae temporibus enim. Quas modi ...	1086	974	NULL	NULL

ตาราง orders (คำสั่งซื้อ)

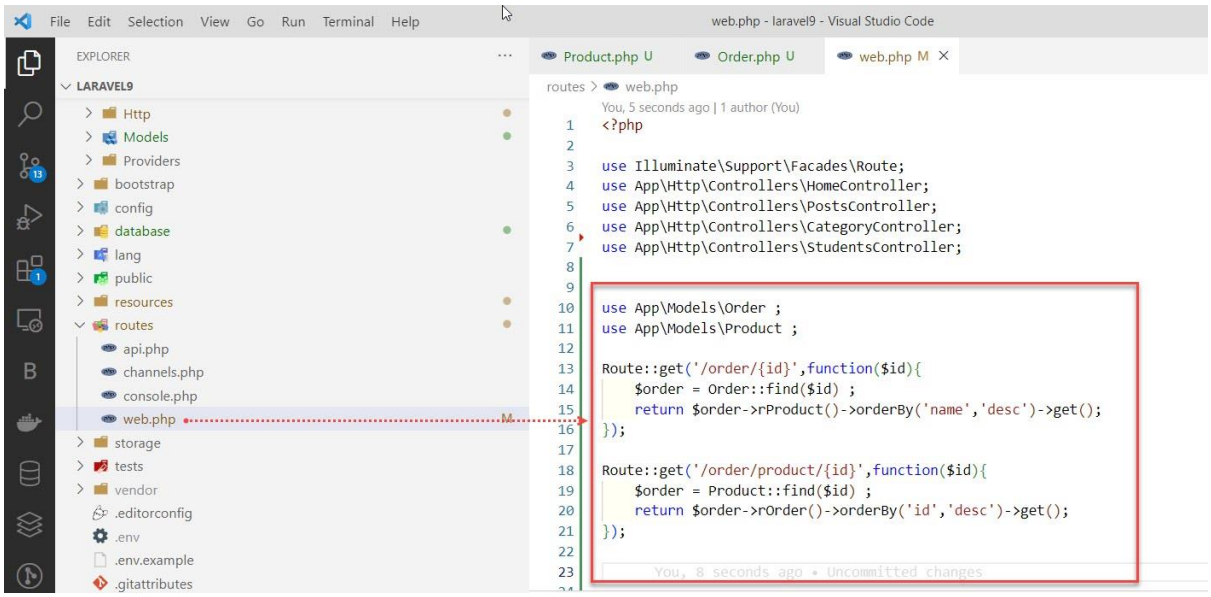
	id	orderID	userID	status	created_at	updated_at
Edit Copy Delete	1	1	1	1	2022-11-02 11:33:30	2022-11-02 11:33:30
Edit Copy Delete	2	2	1	1	2022-11-02 11:33:30	2022-11-02 11:33:30

ตาราง orderdetails (รายละเอียดคำสั่งซื้อ)

	id	orderID	productID	quantity	created_at	updated_at
Edit Copy Delete	1	1	1	4	2022-11-02 11:33:30	2022-11-02 11:33:
Edit Copy Delete	2	1	2	2	2022-11-02 11:33:30	2022-11-02 11:33:
Edit Copy Delete	3	1	3	1	2022-11-02 11:33:30	2022-11-02 11:33:

หลักจากนั้นให้สร้างช่องทางการเข้าใช้ผ่าน routes แล้วส่งไปให้ views แสดงผลเพื่อทดสอบการแสดงผล โดยการเรียกใช้งานนั้นจะเรียกใช้งานผ่านโมเดลที่ทำการเชื่อมความสัมพันธ์ไว้

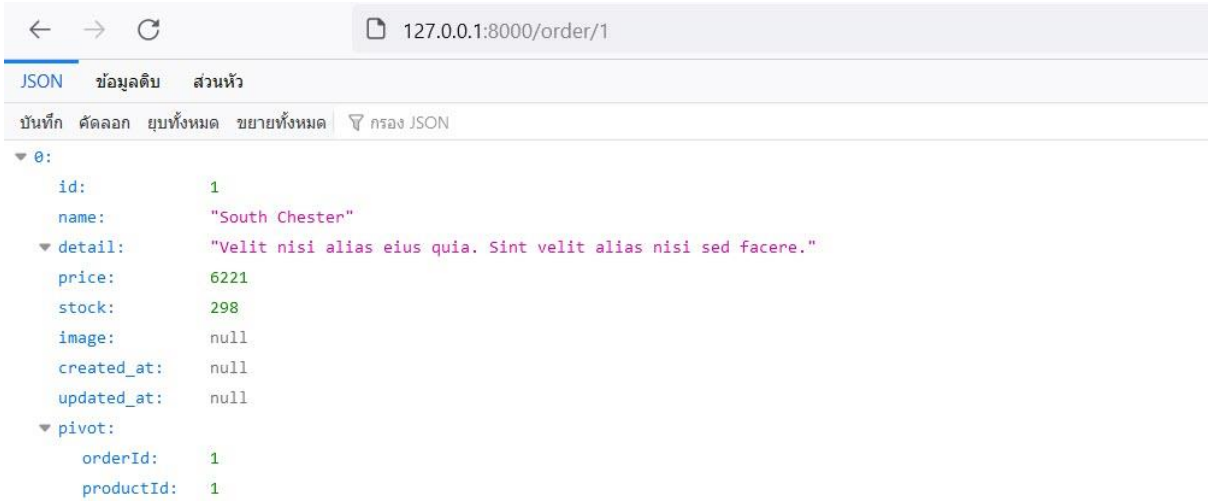
6.สร้าง routes สำหรับเพิ่มเส้นทางเข้าถึงหน้าเว็บ โดยเพิ่ม URL order และ order/product เปิดไฟล์ web.php ในโฟลเดอร์ routes/web.php ดังนี้



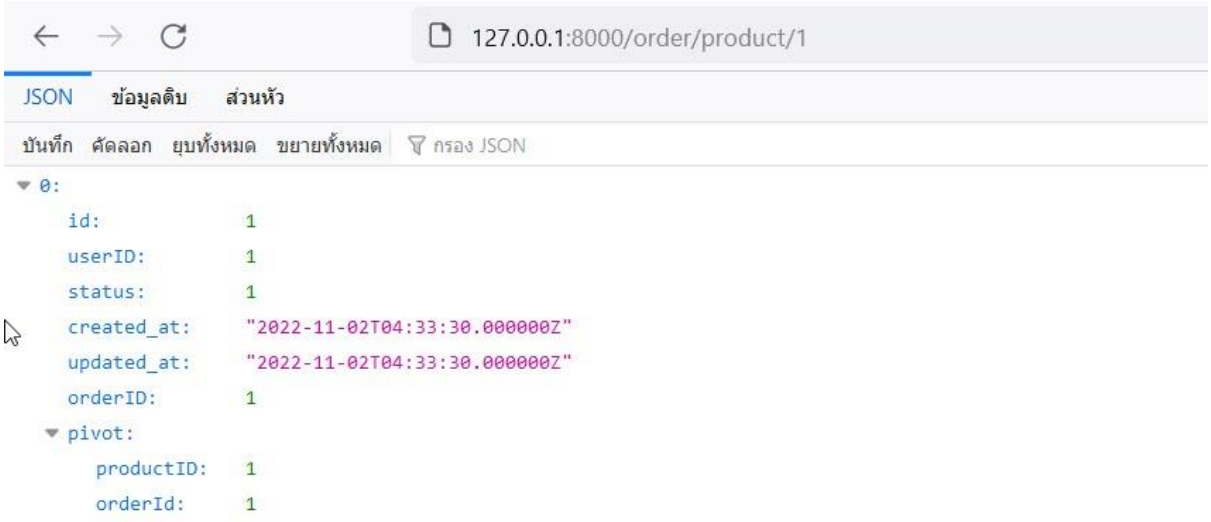
จากตัวอย่างมีการกำหนดช่องทางการเข้าหน้าเว็บผ่าน URL /order/{id} (http://localhost:8000/order/1) และมีการแนบ id (รหัสคำสั่งซื้อ) เข้ามาด้วย จะเห็นได้ว่ามีการเรียกใช้โมเดลชื่อว่า Order และใช้ find($id) สำหรับเป็นเงื่อนไขการดึงข้อมูลของคำสั่งซื้อนั้น ๆ และเรียกใช้ตารางสินค้าที่เชื่อมความสัมพันธ์ไว้คือ rProduct() แล้วคืนค่ากลับไปแสดงผลผ่าน browser

และมีการกำหนดช่องทางการเข้าหน้าเว็บอีกอันคือ URL /order/product/{id} (http://localhost:8000/order/product/1) มีการแนบ id (รหัสคำสั่งซื้อ) มาด้วย มีการเรียกใช้โมเดลชื่อว่า Product และใช้ find($id) สำหรับเป็นเงื่อนไขว่าสินค้าที่สั่งซื้ออยู่ในคำสั่งซื้อใดบ้าง และเรียกใช้ตารางคำสั่งซื้อที่เชื่อมความสัมพันธ์ไว้คือ rOrder() แล้วคืนค่ากลับไปแสดงผลผ่าน browser

4.ทดสอบดูการแสดงผลดูคำสั่งซื้อ พิมพ์ localhost:8000/order/1 ที่ Browser จะเห็นว่ามีการแสดงข้อมูลสินค้าที่เชื่อมกับคำสั่งซื้อไว้แล้ว ดูตรง pivot นะครับ



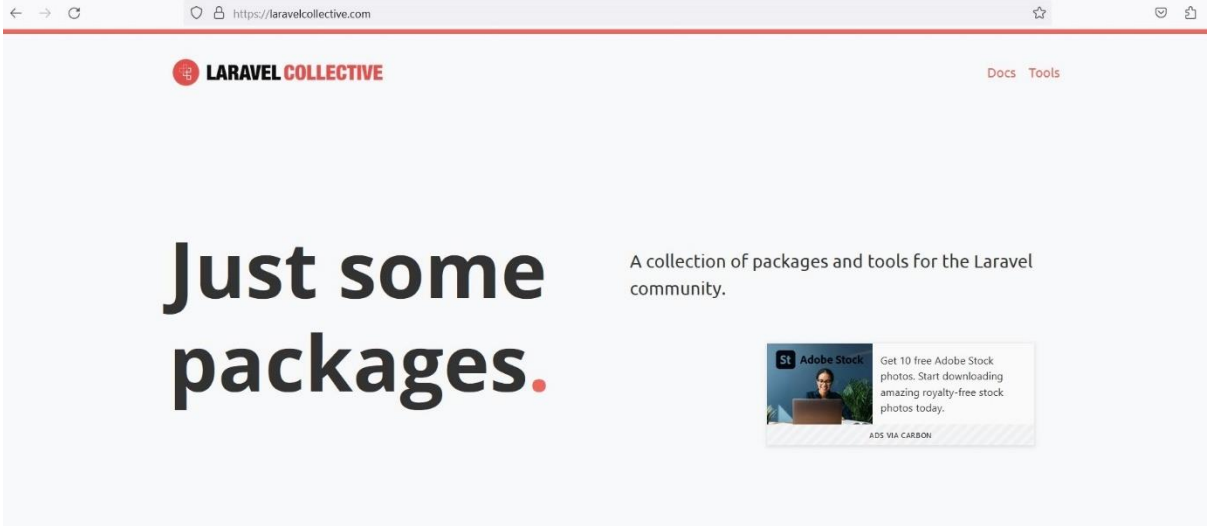
ทดสอบดูการแสดงผล สินค้าว่าอยู่ในคำสั่งซื้อไหนบ้าง พิมพ์ localhost:8000/order/product/1 ที่ Browser จะเห็นว่ามีการแสดงข้อมูลสินค้าที่อยู่ในคำสั่งซื้อแต่ละรายการ ดูตรง pivot นะครับ



จากที่ผ่านเราได้เรียนรู้การสร้างความสัมพันธ์ในรูปแบบต่าง ๆ ไปแล้วนั้นก็คงจะ งง !! ไม่ใช่น้อย ที่ท่าน งง !! ไม่ใช่ปัญหาเลยครับ สำหรับคนที่ไม่ งง เลย ท่านคือเทวดา ครับ ลองทดสอบสร้างความสัมพันธ์และทดลองใช้คำสั่งต่าง ๆ ดูด้วยตัวเองแล้วทดสอบดูบ่อย ๆ นะครับ

สร้าง Web Forms ใน Laravel

จากบทเรียนที่ผ่านเราได้สร้างฟอร์มไปในบางส่วนแล้วนั้น ซึ่งเป็นการเขียนโค้ด HTML เองทั้งหมด ก็จะใช้เวลาในการพิมพ์คำสั่ง HTML เยอะพอสมควร ผลลัพธ์ที่ได้ก็จะได้เหมือนกัน ก็สามารถเขียนฟอร์มโดยการเขียน HTML ได้ปกติเลยครับ แต่ถ้าต้องการลดเวลาในการสร้างฟอร์ม Laravel Framework จะมีเครื่องมือช่วยในการสร้าง นั่นคือ Laravel Collective ซึ่งเป็นคลาสที่ช่วยให้เราสามารถบริหารจัดการฟอร์ม กรอกข้อมูลได้รวดเร็วมากยิ่งขึ้น ช่วยให้เขียนคำสั่งสั้นลง หน้าเว็บไซต์หลักของ Laravel Collective



ติดตั้งและใช้งาน Laravel Collective

การติดตั้งเป็นอะไรที่ง่ายมากเพียงพิมพ์คำสั่งผ่าน Terminal ก็สามารถใช้งานคลาสในการสร้างฟอร์มได้เลย โดยไม่ต้องตั้งค่าอะไรเพิ่มเติม

เปิด Terminal แล้วพิมพ์คำสั่งและกด Enter รอจนกว่าจะโหลดไฟล์ต่าง ๆ มาติดตั้งจนเสร็จ (เชื่อมต่ออินเทอร์เน็ตด้วยนะ)

```
composer require laravelcollective/html
```

```
PS D:\xampp\htdocs\laravel9> composer require laravelcollective/html
./composer.json has been updated
Running composer update laravelcollective/html
Loading composer repositories with package information
Updating dependencies
Nothing to modify in lock file
Writing lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

  INFO  Discovering packages.

  barryvdh/laravel-debugbar ..... DONE
  laravel/sail ..... DONE
  laravel/sanctum ..... DONE
  laravel/tinker ..... DONE
  laravelcollective/html ..... DONE
  nesbot/carbon ..... DONE
  nunomaduro/collision ..... DONE
  nunomaduro/termwind ..... DONE
  spatie/laravel-ignition ..... DONE

81 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

  INFO  No publishable resources for tag [laravel-assets].

No security vulnerability advisories found
Using version ^6.3 for laravelcollective/html
```

โดยหนังสือนี้จะใช้ laravelcollective/html version 6.3 ซึ่งเป็นเวอร์ชันล่าสุด ณ ตอนที่เขียนหนังสือนี้

หลังจากติดตั้ง Laravel Collective เรียบร้อยแล้ว เราสามารถสร้างฟอร์ม สร้างปุ่ม หรือสร้างลิงค์ต่าง ๆ ได้ การสร้างฟอร์มนั้นขึ้นอยู่กับว่าเราต้องการสร้าง input สำหรับกรอกข้อมูลแบบไหน เช่น กล่องข้อความสำหรับกรอกข้อมูลแบบธรรมดา ช่องสำหรับอัปโหลดไฟล์ หรือการกำหนดช่องการกรอกข้อมูลที่เป็นตัวเลขหรือข้อความตัวอักษร ลิสต์รายการสำหรับเลือกข้อมูลที่ต้องการ เราจะต้องมีการออกแบบฐานข้อมูลเพื่อให้รองรับข้อมูลที่ได้จากฟอร์มด้วย เพื่อเป็นการทดสอบว่าเราติดตั้ง Laravel Collective สำเร็จหรือไม่

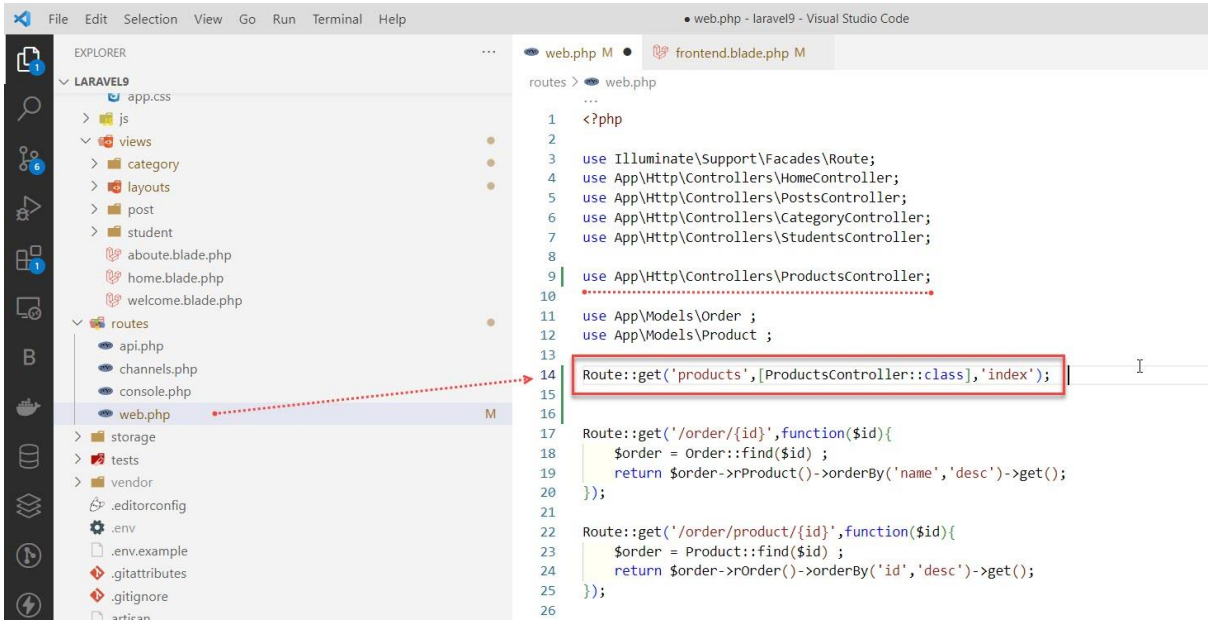
ลองสร้างเมนูเพิ่มสำหรับการจัดการหน้าสินค้า ดังนี้

1.เปิดไฟล์ frontend.blade.php ในโฟลเดอร์ resources/views/layouts/... แล้วเพิ่มเมนู สินค้า ดังนี้



2.สร้าง routes สำหรับเพิ่มเส้นทางเข้าถึงหน้าเว็บ โดยเพิ่ม URL /products แล้วเขียนคำสั่งเชื่อมต่อกับ Controller และ Views ตามขั้นตอนดังนี้

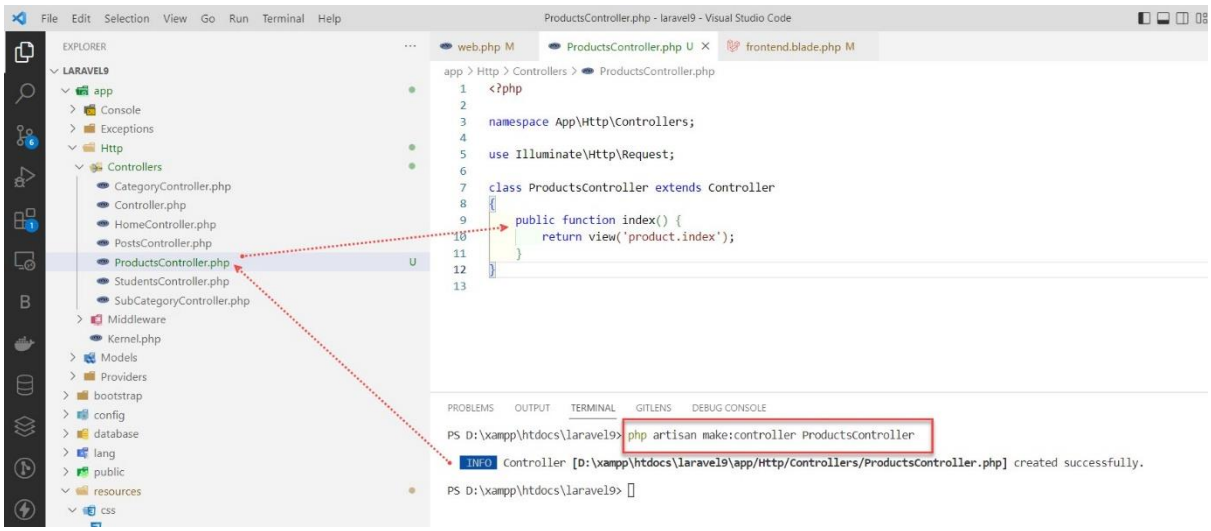
เปิดไฟล์ web.php ในโฟลเดอร์ routes/web.php และเพิ่ม routes ดังนี้



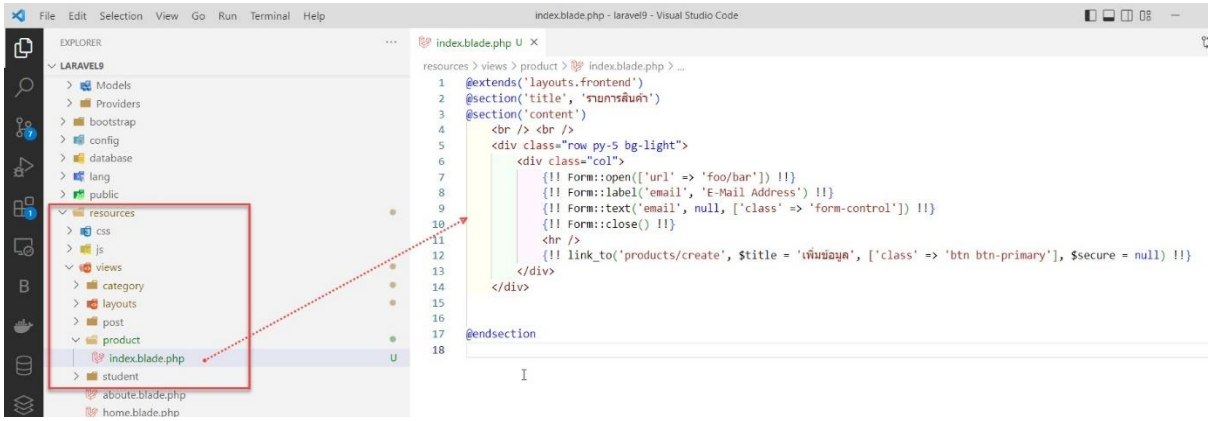
3.สร้างคอนโทรลเลอร์ ProductsController.php เพิ่ม และเพิ่มเมธอด index() แล้วเขียนโค้ดสำหรับเชื่อมต่อ Views ในเมธอด index()

พิมพ์คำสั่งผ่าน Terminal แล้วกด Enter ดังนี้

```
php artisan make:controller ProductsController
```



4.สร้างโฟลเดอร์ product ไว้ที่ views แล้วเพิ่มไฟล์ index.blade.php (resources/views/product/...) เขียนโค้ดสำหรับแสดงผล ดังนี้




```
{!! Form::open(['url' => 'foo/bar']) !!}
{!! Form::label('email', 'E-Mail Address') !!}
{!! Form::text('email', null, ['class' => 'form-control']) !!}
{!! Form::close() !!}
<hr />
{!! link_to('products.create', $title = 'เพิ่มสินค้า', ['class' => 'btn btn-primary']) !!}
```

5.บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000 ที่ Browser แล้วคลิกที่เมนู สินค้า

อย่าลืมพิมพ์คำสั่ง php artisan serv เพื่อสั่งให้ server ทำงานด้วยนะครับ



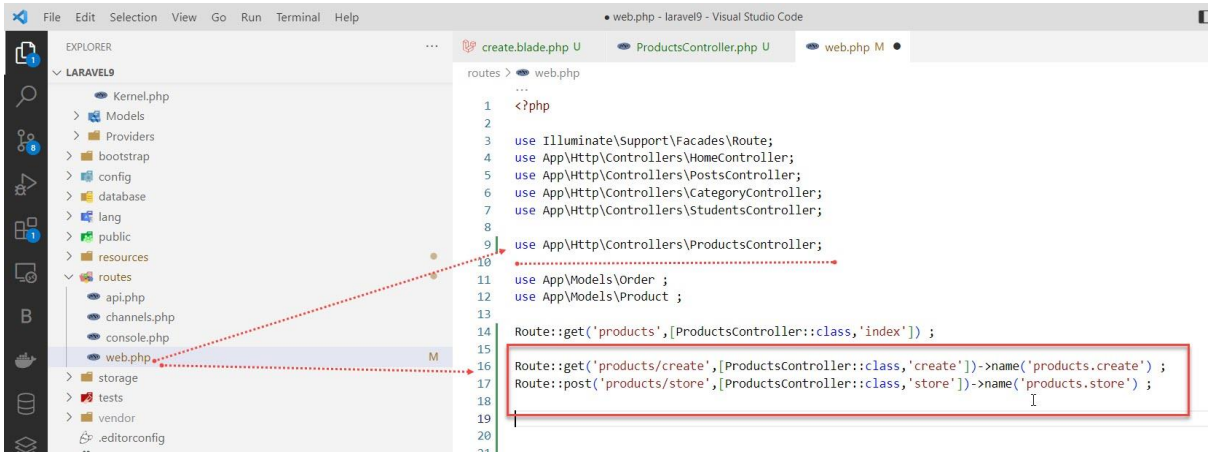
หากมีช่องให้กรอกข้อความและมีปุ่มเพิ่มข้อมูลเข้ามาแสดงว่าการติดตั้งเสร็จสมบูรณ์ สามารถเรียกใช้งานคลาสต่าง ๆ ได้ ตามต้องการ

สร้างฟอร์มเพิ่มข้อมูลด้วย Laravel Collective

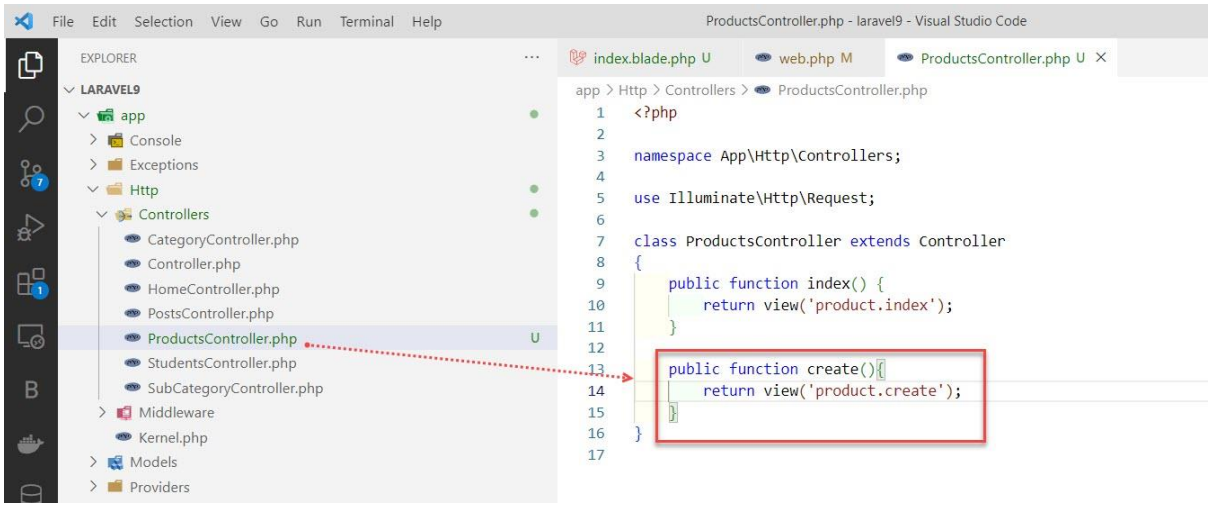
บทเรียนนี้จะได้ศึกษาเกี่ยวกับวิธีการสร้างฟอร์มสำหรับบันทึกข้อมูลสินค้า ซึ่งเราได้สร้างตารางข้อมูลสินค้าไว้แล้วนั้น นั่นก็คือตาราง products โดยในตารางจะประกอบด้วย ชื่อสินค้า รายละเอียดสินค้า ราคาสินค้า จำนวนสต๊อกสินค้า ราคาสินค้า และรูปสินค้า เราจะทำการสร้างฟอร์มเพื่อจัดการสินค้าได้ตามขั้นตอนดังนี้

เมื่อคลิกที่ปุ่ม เพิ่มสินค้า จะไปทำงานที่หน้าฟอร์มสำหรับเพิ่มข้อมูลสินค้า จากข้อ 4 ที่ผ่านมาจะเห็นว่าได้มีการสร้างลิงค์ เพิ่มสินค้า โดยอ้างอิงชื่อ /products/create ไว้แล้วแต่ยังไม่ได้มีการสร้าง routes ให้เราไปสร้างเพิ่ม ดังนี้

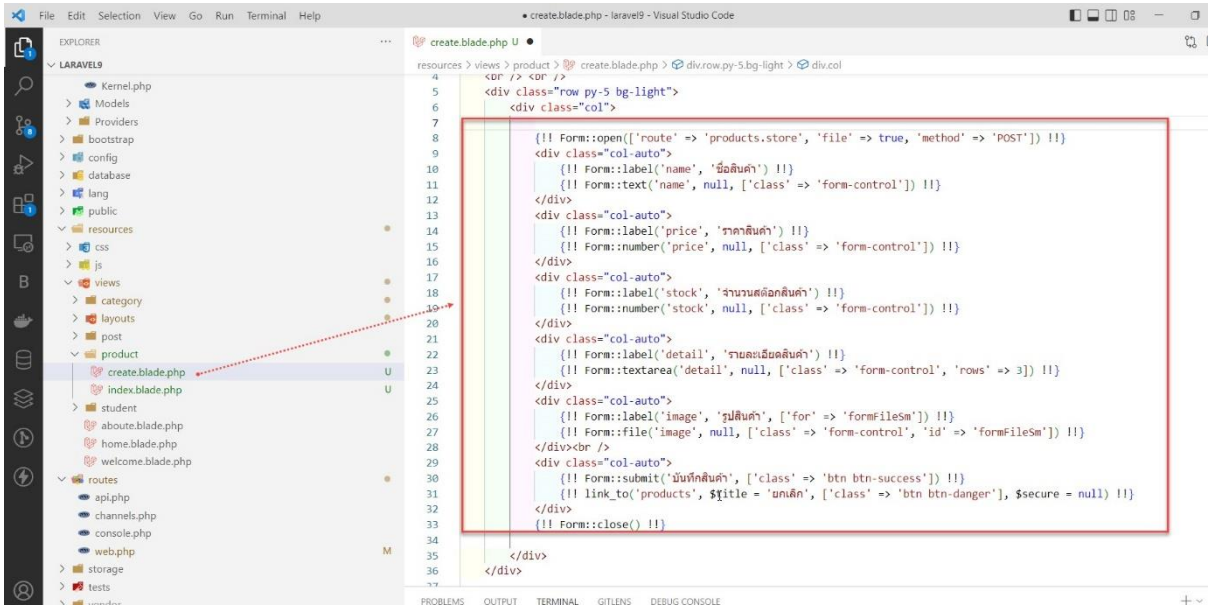
1.เปิดไฟล์ web.php ในโฟลเดอร์ routes/web.php แล้วเพิ่ม routes ดังนี้



2.เปิดไฟล์ ProductsController.php และเพิ่มเมธอด create() แล้วเขียนโค้ดสำหรับเชื่อมต่อ Views ในเมธอด create() ดังนี้

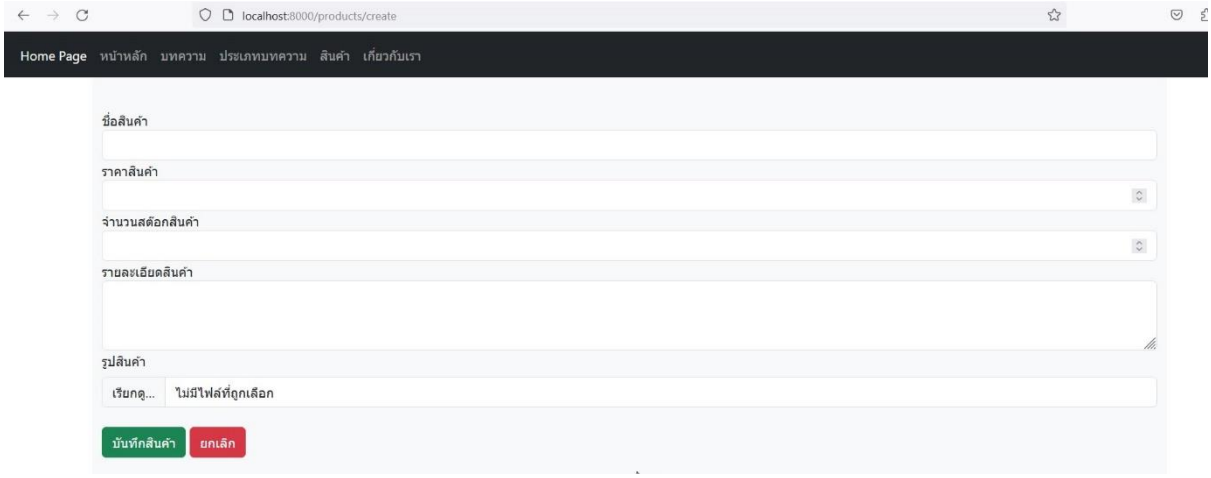


3.สร้างไฟล์ create.blade.php ไว้ในโฟลเดอร์ resources/views/product/... แล้วเขียนโค้ดเข้าไป ดังนี้



จากตัวอย่างโค้ดที่เขียน เห็นว่าจะมีการใช้ Laravel Collection ช่วยในการสร้างฟอร์มซึ่งจะเริ่มต้นด้วยการเปิดฟอร์ม โดยใช้คำสั่ง {!! Form::open() !!} มีการกำหนดคุณสมบัติให้กับฟอร์มซึ่งจะเก็บเป็นอาเรย์ 3 ตัว คือ action ที่จะส่งไป คือ products.store ซึ่งจะถูกส่งไปทำงานที่เมธอด store อยู่ในคอนโทรลเลอร์ชื่อ ProductsController โดยระบุวิธีการส่งข้อมูลแบบ Post และมีการระบุให้ฟอร์มสามารถอับโหลดไฟล์ได้ด้วย และปิดท้ายฟอร์มด้วย {!! Form::close() !!}

5.บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/products/cretate ที่ Browser



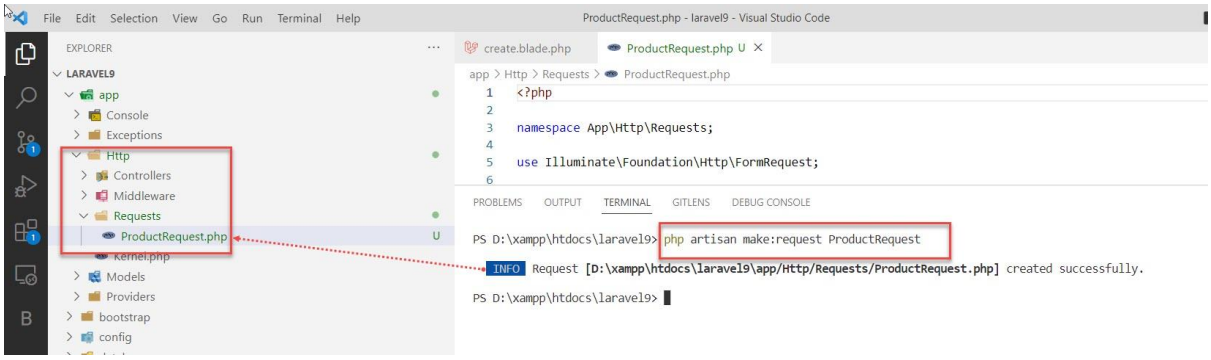
Form Validation

เป็นการตรวจสอบความถูกต้องของฟอร์มก่อนการบันทึกข้อมูล เช่น รูปแบบของการกรอกข้อมูลจากผู้ใช้ ตรวจเช็คไม่ให้ฟอร์มมีค่าว่าง ตรวจสอบรูปแบบของอีเมล ถ้าตรวจสอบแล้วข้อมูลไม่เป็นไปตามรูปแบบหรือกฎที่ตั้งไว้ก็จะมีข้อความแจ้งเตือนให้ทราบ Laravel จะมีกฎในการตรวจสอบความถูกต้องของข้อมูลมาให้แล้ว สามารถสร้าง Request เพื่อตรวจสอบความถูกต้องของฟอร์มและระบุชื่อของ Request ที่ต้องการได้ การสร้างจะใช้คำสั่งผ่าน Terminal ดังนี้

```
php artisan make:request ProductRequest
```

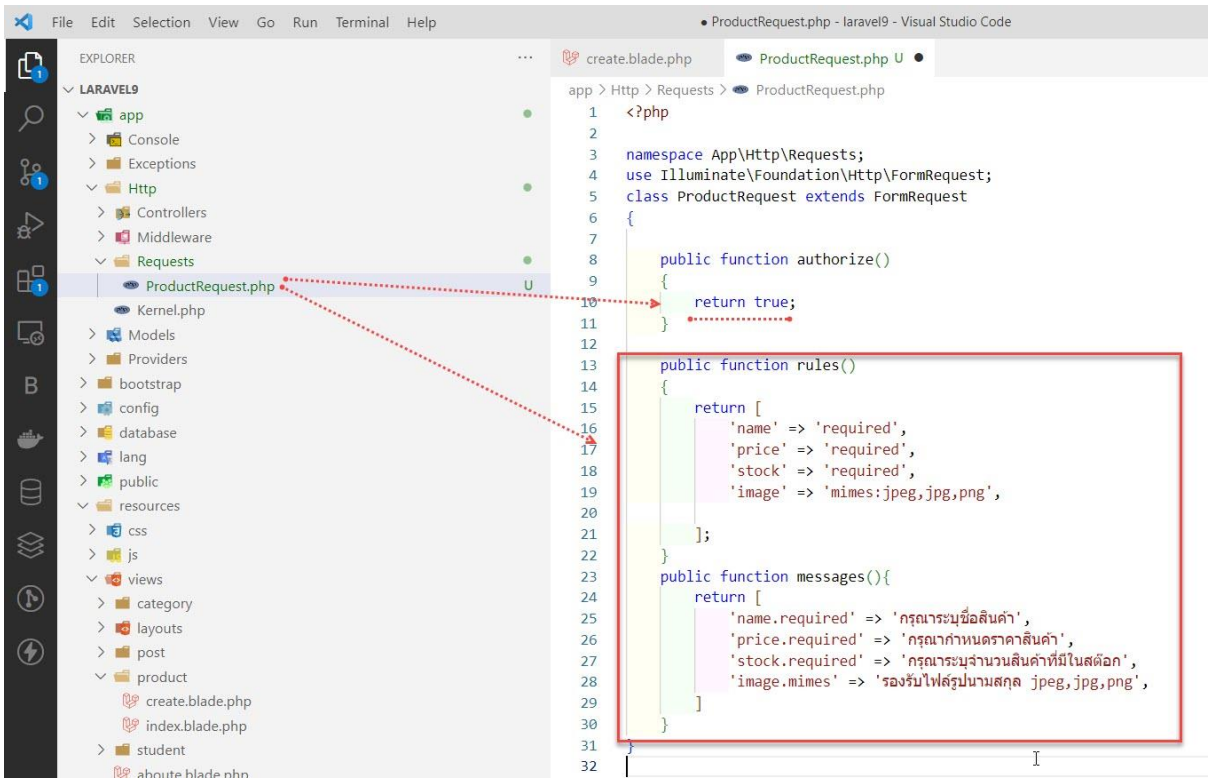
การสร้างและกำหนดกฎหรือรูปแบบของข้อมูล สามารถทำตามขั้นตอนดังนี้

1.เปิด Terminal แล้วพิมพ์คำสั่งและกด Enter

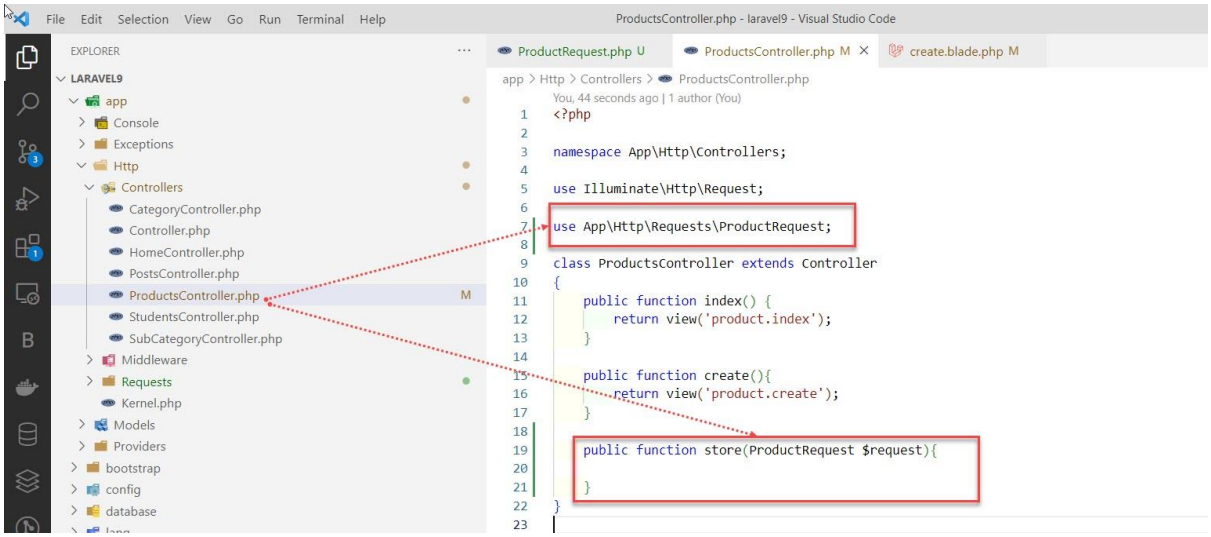


เมื่อรันคำสั่งเสร็จ ไฟล์ ProductRequest.php จะถูกสร้างขึ้นไว้ที่โฟลเดอร์ app\Http\Rquests\...

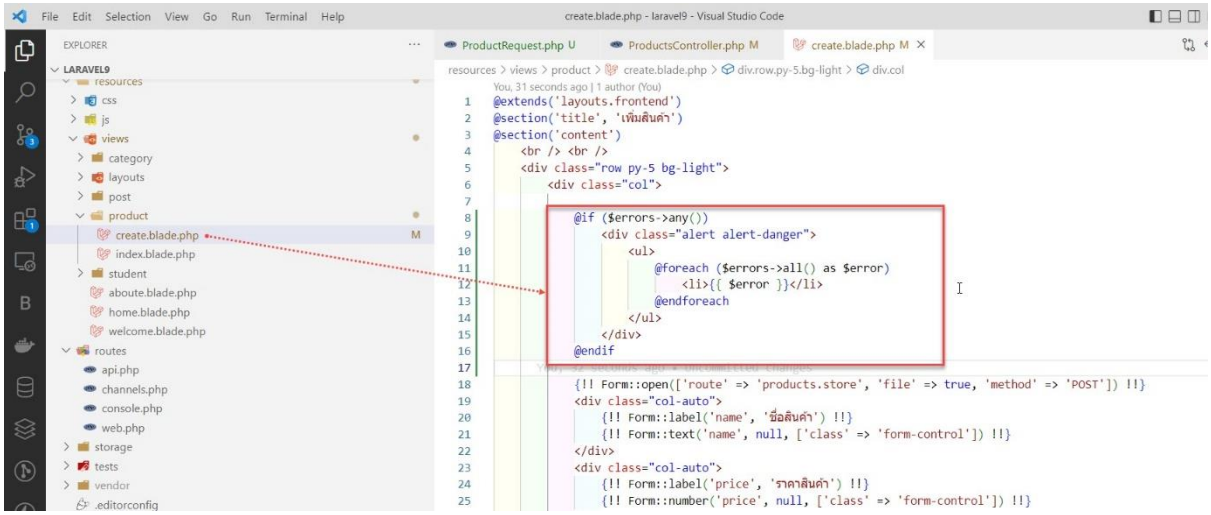
2.เปิดไฟล์ ProductRequest.php แล้วกำหนดค่าในส่วนของ เมธอด authorize() จาก false เป็น true จากนั้นให้เพิ่มกฎที่ต้องการตรวจสอบด้วยการระบุฟิวด์ที่ต้องการตรวจสอบใน เมธอด rules() และสร้างเมธอด messages() เพิ่มเพื่อกำหนดข้อความแจ้งเตือนที่ต้องการ ดังนี้



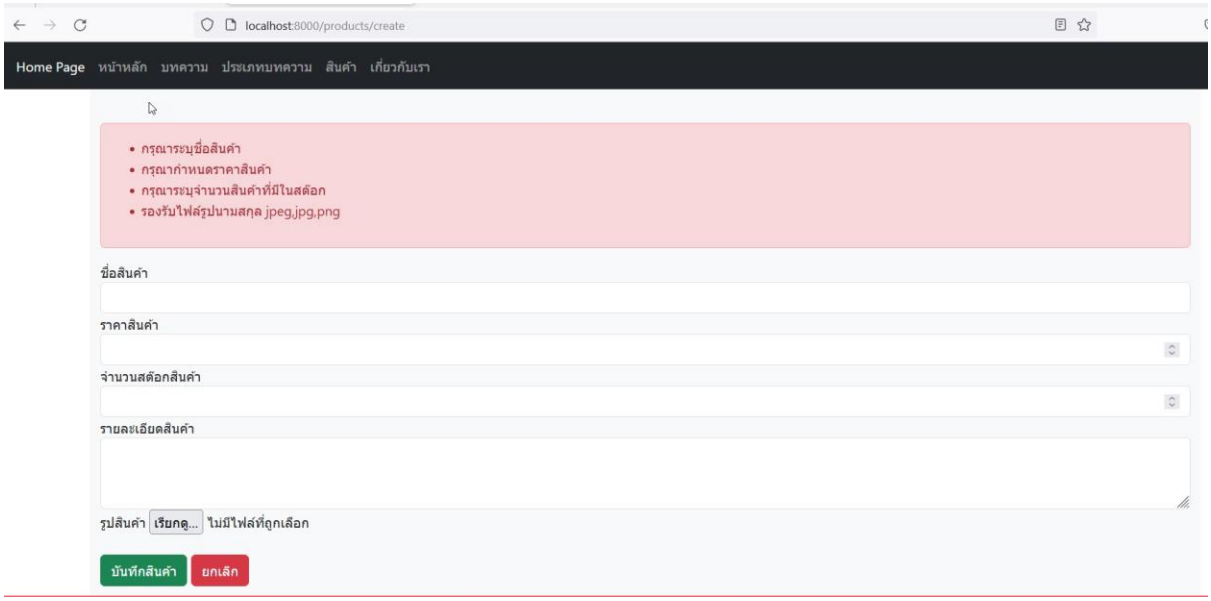
3.เปิดไฟล์ ProductsController.php แล้วสร้างเมธอด store() เพิ่มเข้าไป แล้วเรียกใช้งาน ProductRequest ก่อนเรียกใช้ให้ use app\Http\Requests\ProductRequest เข้ามาด้วย เพื่อเรียกใช้งาน



4. เปิดไฟล์ create.blade.php ในโฟลเดอร์ resources\views\product\create.blade.php เพื่อแสดงข้อความ Errors แจ้งเตือนความผิดพลาดของข้อมูล สามารถกำหนดตำแหน่งที่ต้องการได้ โดยการแทรกโค้ด ในตัวอย่างนี้จะแทรกโค้ดก่อนคำสั่งเปิดฟอร์ม {!! Form::open() !!} ดังนี้



บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/products/cretate ที่ Browser แล้วลองกดบันทึกข้อมูลดูอีกรอบ



ก็จะเห็นข้อความแจ้งเตือนข้อผิดพลาดให้เราทราบ

แจ้งเตือนข้อผิดพลาดจากไฟล์ภายนอก

บางครั้งการทำงานเกี่ยวกับฟอร์มหลายๆฟอร์ม จะต้องพิมพ์คำสั่งแจ้งเตือนข้อผิดพลาด ซ้ำๆ บ่อย ๆ ในทุกๆ ฟอร์มที่ต้องการแจ้งเตือน เราสามารถสร้างไฟล์แยกออกเป็นอีกไฟล์ได้ แล้วค่อยเรียกใช้งานในส่วนของฟอร์มหรือหน้าที่ต้องการแสดงข้อความแจ้งเตือน โดยเริ่มต้นให้สร้างโฟลเดอร์ขึ้นมาใหม่ชื่อว่า includes เก็บไว้ในโฟลเดอร์ resources/views/... จากนั้นให้สร้างไฟล์ใหม่ ชื่อว่า errors.blade.php แล้วเขียนคำสั่งเข้าไปในไฟล์ ดังนี้



ที่ส่วนของฟอร์มเราต้องการให้แสดงข้อความแจ้งเตือนส่วนไหน ก็สามารถไปกำหนดที่ตรงนั้น โดยใช้คำสั่ง @include() แล้วระบุไฟล์ที่ต้องการ แทรกเข้าไปในส่วนนั้น ๆ

```
@include('includes.errors')
```

เปิดไฟล์ create.blade.php ในโฟลเดอร์ resources\views\product\create.blade.php แล้วแก้ไขคำสั่งใหม่ดังนี้

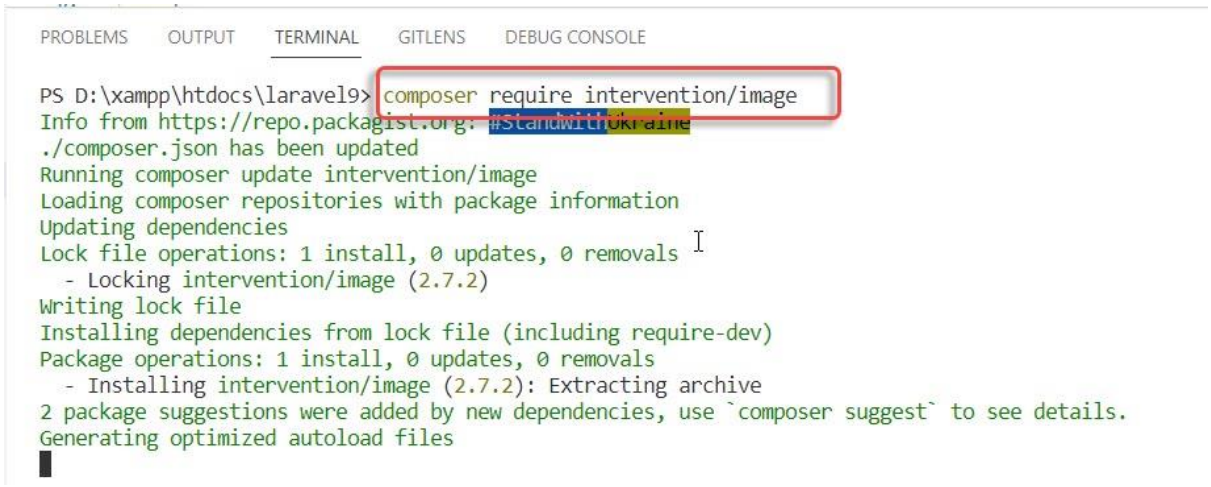


การอับโหลดไฟล์และบันทึกข้อมูลลงฐานข้อมูล

จากบทเรียนที่ผ่านมานั้น เราได้สร้างฟอร์มเก็บข้อมูลของสินค้า และมีการอับโหลดไฟล์ภาพของสินค้าด้วย ในบทเรียนนี้เราจะได้เรียนรู้วิธีการจัดการกับไฟล์ต่าง ๆ การกำหนดคุณสมบัติให้กับไฟล์ ก่อนการอับโหลด เช่น กำหนดขนาดของไฟล์ภาพในกรณีภาพมีความจุมากเกินไป กำหนดชื่อของภาพไม่ให้ซ้ำกัน กำหนดรูปแบบนามสกุลไฟล์ที่ต้องการ (JPEG , PNG , GIF ฯลฯ) การย่อขนาดของภาพตามต้องการ โดยจะใช้ Image Library ช่วยจัดการ ชื่อว่า Intervention Image ก่อนอื่นต้องติดตั้งและกำหนดค่าพื้นฐานก่อนการใช้กงานก่อนนะครับ ให้ทำตามขั้นตอน ดังนี้

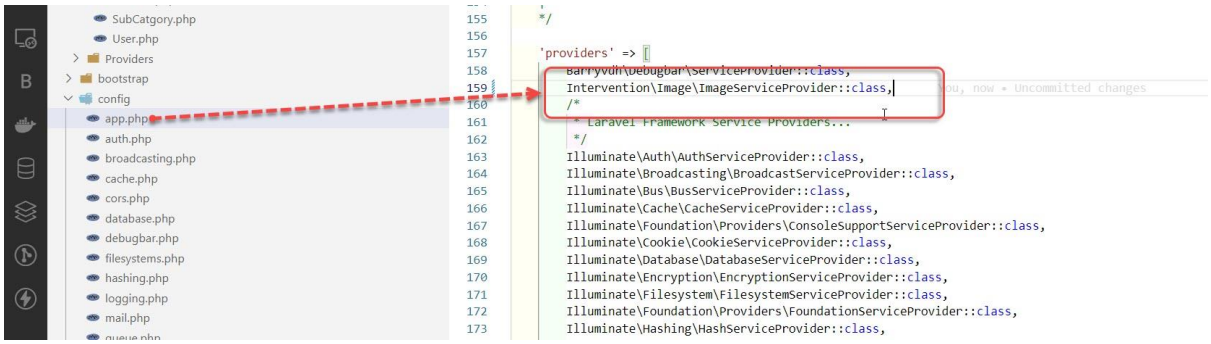
เปิด Terminal แล้วพิมพ์คำสั่งและกด Enter แล้วรอจนกว่าจะติดตั้งเสร็จ

```
composer require intervention/image
```



ติดตั้งเสร็จแล้วให้ตั้งค่าเพิ่มเติมที่ไฟล์ config\app.php แล้วเพิ่มโค้ดในส่วนของ providers Array ดังนี้

```
Intervention\Image\ImageServiceProvider::class
```

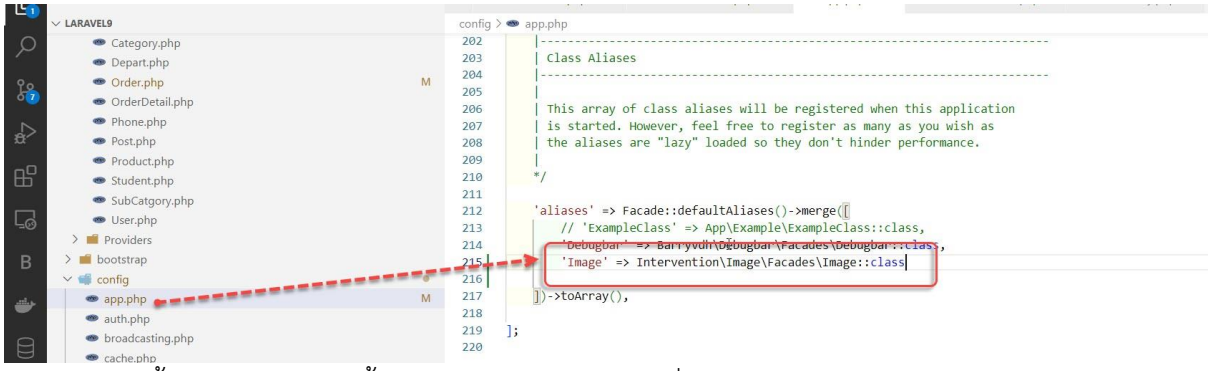


ดูคู่มือการติดตั้งเพิ่มเติม

https://image.intervention.io/v2/introduction/installation

และเพิ่มโค้ดในส่วนของ aliases ด้วย (จะอยู่ประมาณบรรทัดที่ 212 ในไฟล์ config\app.php)

```
'Image' => Intervention\Image\Facades\Image::class
```



จากนั้นก็บันทึกไฟล์ แค่นี้เราก็สามารถใช้ package ชื่อ **Image** ได้แล้ว

ทีนี้ก็ลองเขียนโค้ดเพิ่มข้อมูลและจัดการไฟล์อัปโหลดต่อเลยครับ จากขั้นตอนที่ผ่านมานั้นเราได้มีการสร้างฟอร์มสำหรับเพิ่มข้อมูลสินค้าไว้เรียบร้อยแล้ว และมีการสร้าง ProductsController ไว้แล้วนั้น ให้ไปเขียนโค้ดต่อได้เลย

โดยให้เปิดไฟล์ ProductsController.php ในโฟลเดอร์ app\Http\Controllers\... แล้วเขียนโค้ดในเมธอด store() ดังนี้

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests\ProductRequest; //เรียกใช้ ProductRequest
use App\Models\Product ;  //เรียกใช้ Models Product
use Image;        // เรียกใช้ Image
class ProductsController extends Controller
{
    public function index() {
        return view('product.index');
    }

    public function create(){
        return view('product.create');
    }
```

ดูโค้ดต่อหน้าต่อไปครับ...

```
public function store(ProductRequest $request){

1         $product = new Product();
2         if ($request->hasFile('image')) {
3
4             $rename = time() . '.' .$request->file('image')
5                       ->getClientOriginalExtension();
6             $request->file('image')->move('uploads/product/', $rename);
7            //resize image
8            Image::make(('uploads/product/' . $rename))->resize(450, 450)
9                        ->save(('uploads/resize/' . $rename));
10             $product->image = $rename;
11
12          }else{
13              $product->image = "https://via.placeholder.com/450x450.png";
14          }
15              $product->name = $request->name ;
16              $product->detail = $request->detail ;
17              $product->price = $request->price ;
18              $product->stock = $request->stock ;
19              $product->save();
20              return redirect('/products');
21
22     }
}
```

ตรวจสอบความถูกต้องของข้อมูลที่กรอกผ่านฟอร์ม ถ้าการตรวจสอบถูกต้อง ตัว $request จะรับค่าจากฟอร์มที่ถูกส่งเข้ามา ในบรรทัดที่ 1 เป็นการสร้างตัวแปร $product และกำหนดค่าเป็น Object โมเดล Product เพื่อใช้อ้างอิงตารางสำหรับบันทึกข้อมูลต่าง ๆ ตาม request ที่ถูกส่งมา บรรทัดที่ 2 มีการตรวจสอบเงื่อนไขว่าได้มีการเลือกไฟล์อับโหลดมาหรือไม่ โดยใช้ hasFile() ถ้ามีการอับโหลดไฟล์เข้ามาจะมีการตั้งชื่อใหม่ให้กับไฟล์นั้น ๆ พร้อมอับโหลดไฟล์เก็บไว้ที่โฟลเดอร์ uploads/product/... จากนั้นก็ใช้ `Image::make()` ในการกำหนดขนาดรูปภาพตามที่ต้องการ จากตัวอย่างได้ทำการย่อขนาดให้เหลือ 450x450 พร้อมอับโหลดไฟล์เก็บไว้ที่โฟลเดอร์ uploads/resize/... แต่หากไม่ได้เลือกไฟล์อับโหลดเข้ามาก็จะเป็นการใช้รูปที่เราได้กำหนดไว้เอง บรรทัดที่ 15 – 18 เป็นการกำหนดฟิวด์ในตารางฐานข้อมูลที่เราจะบันทึกโดยจะบันทึกตาม $request ที่รับเข้ามา จากนั้นก็สั่งให้บันทึกข้อมูลลงตาราง โดยใช้ฟังก์ชัน save() บันทึกเสร็จแล้วก็ redirect ไปที่หน้า products (localhost:8000/product)



*** ให้สร้าง โฟลเดอร์สำหรับเก็บไฟล์ที่อับโหลด ไว้ใน public ด้วยนะครับ โดยโฟลเดอร์ product และ resize ให้สร้างไว้ภายใต้โฟลเดอร์ uploads

บันทึกไฟล์แล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/products/cretate ที่ Browser แล้วลองกดบันทึกข้อมูลดูอีกรอบ

การอับโหลดไฟล์อย่าลืมใส่ attributes เข้าไปใน Form::open([...]) ด้วยนะครับ ไม่งั้นไฟล์ที่อับโหลดจะไม่สามารถอับโหลดได้ โดยระบุ 'file' => true

```
 //resources/views/product/create.blade.php
{!! Form::open([
                'route' => 'products.store',
                'method' => 'POST',
                'file' => true,
            ]) !!}

…
{!! Form::close() !!}
```

การแก้ไขข้อมูลด้วย Form Model Binding อย่างง่าย

การแก้ไขข้อมูลที่ต้องการบ่อยครั้งเราจะต้องเขียนโค้ดบางอย่างเพื่อดึงข้อมูลจากฐานข้อมูลมาใส่ไว้ที่ช่อง input แต่ละอัน เพื่อเป็นข้อมูลเริ่มต้นให้เราได้แก้ไขจำได้ไหมครับจากที่ผ่านมาเราเคยเขียนโค้ดแบบนี้

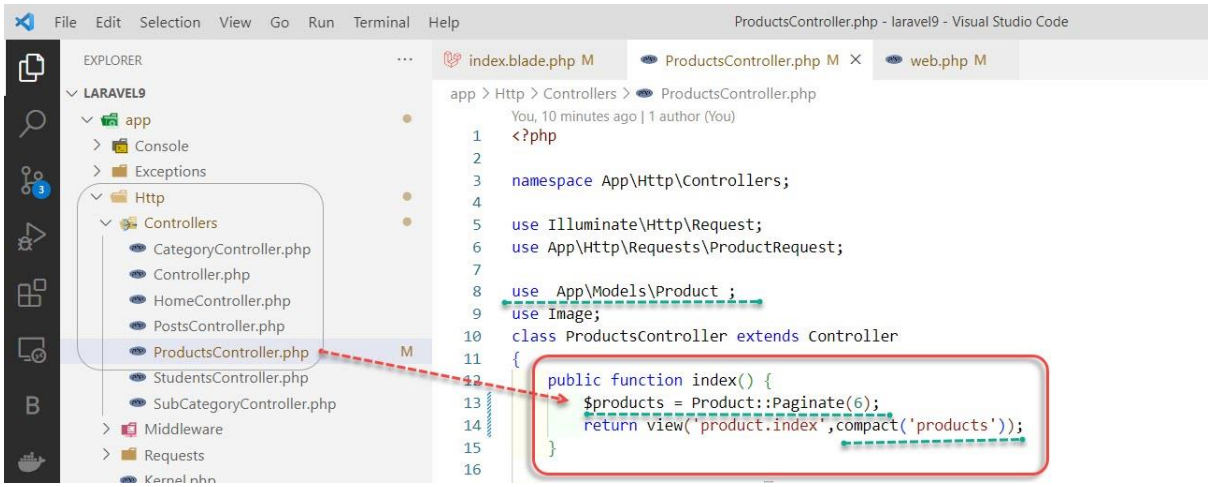
```
<input type="text" name="post_title" value="{{ $post[0]->post_title }}">
```

ตรง value นั้นแหละครับที่เราต้องเพิ่ม ซึ่งถ้ามีช่อง input หลาย ๆ ช่องหละ เราก็ต้องมาใส่ให้ครบทุกช่องใช่ไหมครับ คำตอบคือใช่ ก็ทำได้นะ แต่มันมีวิธีที่ง่ายและรวดเร็วกว่านี้ คือ ใช้ Model Binding สิหมดปัญหาการมานั่งเติม value ในทุกๆ ช่อง ในตัวอย่างต่อไปนี้ จะยกตัวอย่างการสร้างฟอร์มสำหรับการแก้ไขข้อมูลสินค้าอย่างง่าย โดยเราจะทำการคัดลอกโค้ดทั้งหมดที่อยู่ในไฟล์ create.blade.php มาใช้แล้วเปลี่ยนค่าบางอย่างเพียงเล็กน้อยก็สามารถสร้างฟอร์มสำหรับแก้ไขได้แล้ว

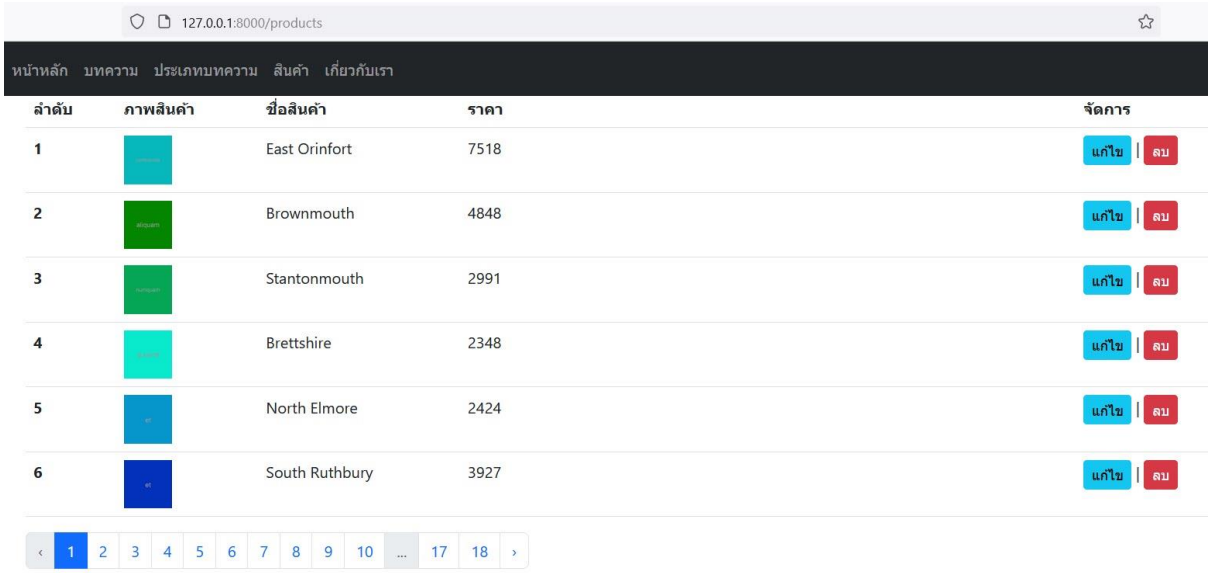
เพื่อให้ง่ายต่อการจัดการส่วนต่าง ๆ ให้สร้างหน้าสำหรับแสดงข้อมูลสินค้าขึ้นมาก่อนในไฟล์ index.blade.php ที่อยู่ในโฟลเดอร์ resources/views/product/... จากนั้นก็ให้ดึงข้อมูลสินค้าทั้งหมดที่อยู่ในฐานข้อมูลมาแสดงและมีการแบ่งหน้าแล้วสร้างปุ่มสำหรับการแก้ไขข้อมูลพร้อมปุ่มสำหรับลบข้อมูล โดยทำตามขั้นตอน ดังนี้ครับ

1.เปิดไฟล์ resources/views/product/index.php แล้วเขียนโค้ดสำหรับการแสดงข้อมูล และกำหนดให้มีการแบ่งหน้าให้ด้วย ดังนี้

```
@extends('layouts.frontend')
@section('title', 'รายการสินค้าทั้งหมด')
@section('content')
<section class="py-5 container">
   <div class="row">
     <div class="col-lg-12  mx-auto">
       <table class="table table-hover">
       <thead>
             <tr>
               <th scope="col">ลำดับ</th>
               <th scope="col">ภาพสินค้า</th>
               <th scope="col">ชื่อสินค้า</th>
               <th scope="col">ราคา</th>
               <th scope="col">จัดการ</th>
             </tr>
       </thead>
               @foreach ($products as $product)
       <tbody>
       <tr>
       <td scope="col">{{ $product->id }}</td>
       <td scope="col"><img src="{{ asset($product->image) }}" width="50px" /></td>
       <td scope="col">{{ $product->name }}</td>
       <td scope="col" style="width: 50%">{{ $product->price }}</td>
       <td scope="col">
           <a href="{{ route('products.edit', $product->id) }}"
           class="btn btn-info btn-sm">แก้ไข</a>
           <a href="{{ route('products.destroy', $product->id) }}"
class="btn btn-danger btn-sm" onclick="return confirm('ยืนยันการลบหรือไม่')">ลบ
           </a>
       </td>
       </tr>
     </tbody>
             @endforeach
       </table>
         {{ $products->links() }}    //สำหรับแสดงปุ่มแบ่งหน้า
           </div>
        </div>
</section>
@endsection
```

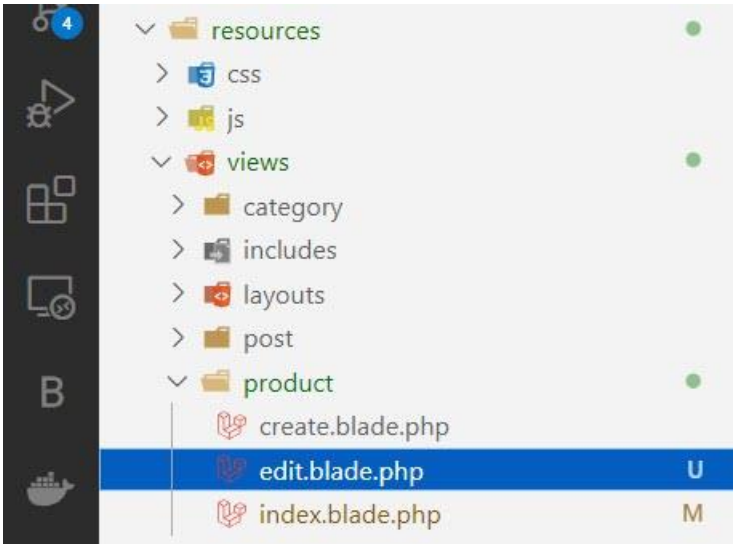


บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/product ที่ Browser



จะได้ข้อมูลสินค้าทั้งหมดแสดงที่หน้าจอพร้อมมีการแบ่งหน้าการแสดงผล โดยเราได้กำหนดการแสดงข้อมูลต่อหน้าไว้หน้าละ 6 รายการ และมีการสร้างปุ่มสำหรับแก้ไขข้อมูลและลบข้อมูล

3. สร้างไฟล์ edit.blade.php ไว้ในโฟลเดอร์ resources/views/product/...



4. ให้ทำการคัดลอกโค้ดทั้งหมดที่อยู่ในไฟล์ resources/views/product/create.blade.php มาวางไว้ใน edit.blade.php แล้วแก้ไขโค้ดในส่วนของ Form::open(...) เป็น Form::model(...) และกำหนดค่าดังนี้

```
// resources/views/product/edit.blade.php
@extends('layouts.frontend')
@section('title', 'เพิ่มสินค้า')
@section('content')
    <br/> <br />
    <div class="row py-5 bg-light">
        <div class="col">
            @include('includes.errors')
               {!! Form::model($product, [
                    'route' => [products.update', $product->id],
                     'method' => 'POST',
                     'file' => true,
                    'enctype' => 'multipart/form-data'
                     ]) !!}
             <div class="col-auto">
                 {!! Form::label('name', 'ชื่อสินค้า') !!}
                 {!! Form::text('name', null, ['class' => 'form-control']) !!}
             </div>
             <div class="col-auto">
                 {!! Form::label('price', 'ราคาสินค้า') !!}
                 {!! Form::number('price', null, ['class' => 'form-control']) !!}
             </div>
             <div class="col-auto">
                 {!! Form::label('stock', 'จำนวนสต๊อกสินค้า') !!}
                 {!! Form::number('stock', null, ['class' => 'form-control']) !!}
             </div>
             <div class="col-auto">
                 {!! Form::label('detail', 'รายละเอียดสินค้า') !!}
                 {!! Form::textarea('detail', null, ['class' => 'form-control',
                  'rows' =>   3]) !!}
             </div>
             <div class="col-auto">
                 {!! Form::label('image', 'รูปสินค้า', ['for' => 'formFileSm']) !!}
                 {!! Form::file('image', null, ['class' => 'form-control',
                 'id' => 'formFileSm']) !!}
             </div><br/>
             <div class="col-auto">
                 {!! Form::submit('บันทึกสินค้า', ['class' => 'btn btn-success']) !!}
                 {!! link_to('products', $title = 'ยกเลิก', ['class' => 'btn btn-danger'],
                      $secure = null) !!}
             </div>
             {!! Form::close() !!}
        </div></div>
@endsection
```

จากตัวอย่างโค้ดที่เขียนนั้นจะเห็นว่าฟอร์มแก้ไขจะเหมือนกันกับฟอร์มเพิ่มข้อมูลเลย จะต่างกันแค่ตอนใช้คำสั่งเปิดฟอร์มคือโค้ดส่วนนี้ ให้เราแก้ไขตามนี้เลย

```
{!! Form::model($product, [
          'route' => [products.update', $product->id],
          'method' => 'POST',
          'file' => true,
         'enctype' => 'multipart/form-data'

    ]) !!}
     …
{!! Form::close !!}
```

Form::model(...) เป็นอีกคลาสที่ใช้สำหรับกำหนดค่าข้อมูลให้กับฟอร์ม หรือที่เรียกว่า Session Flash Data (Old Input) โดยจะทำการผูกเข้ากับ Model ให้อัตโนมัติ ในตัวอย่างคือตัวแปร $product ซึ่งเป็นตัวแปรที่รับค่ามาจาก Controller และได้กำหนด Attributes เข้าไป 4 ค่า คือ

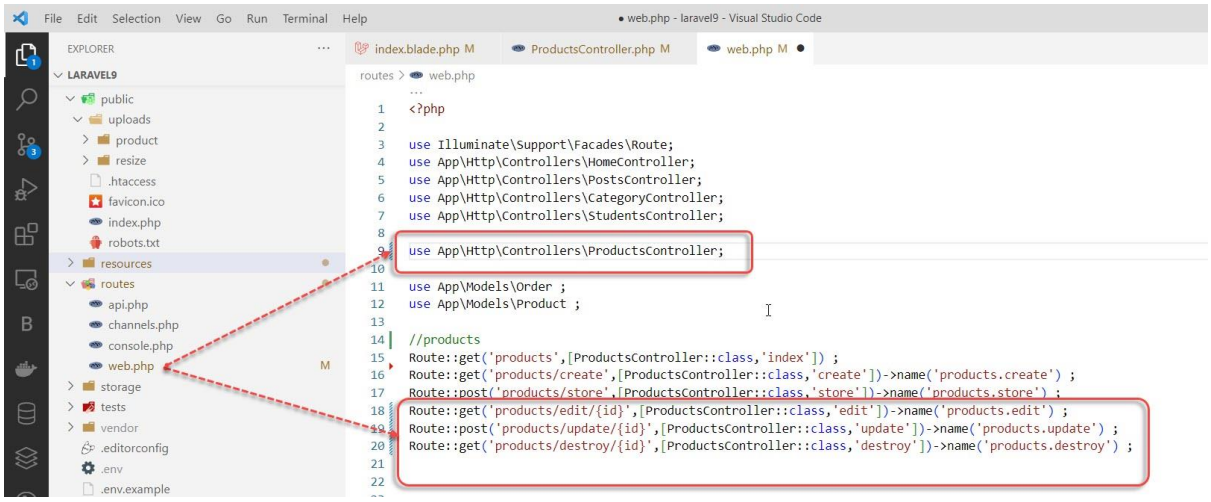
route เป็นส่วนที่ระบุว่าจะให้ไปทำงานที่ไหนเมื่อมีการกดปุ่มบันทึกในตัวอย่างคือ product.update และมีการแนบ `$product->id` ไปด้วยเพื่อเป็นเงื่อนไขการแก้ไขข้อมูลตามรายการที่เลือก ฉะนั้นเราต้องไปสร้าง routes เพิ่มใน routes/web.php (เดียวพาสร้างในขั้นตอนต่อไป)

method เป็นการกำหนดรูปแบบการส่งในตัวอย่างคือส่งข้อมูลไปแบบ POST

file การเปิดใช้คุณสมบัติการอัปโหลด กำหนดเป็น true

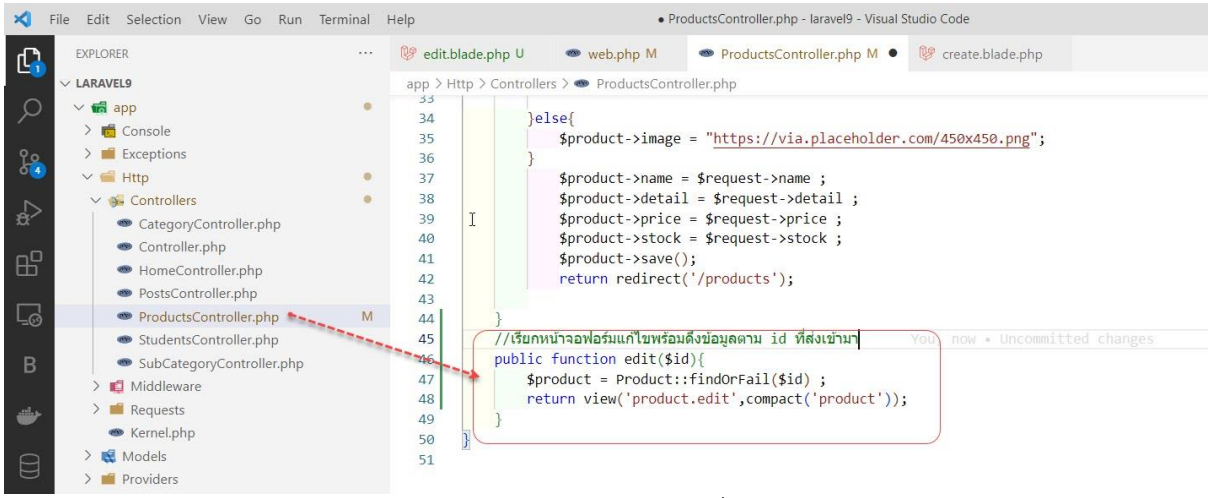
enctype = "multipart/form-data" กำหนดให้สามารถอัปโหลดไฟล์ได้

5.เปิดไฟล์ web.php ในโฟลเดอร์ routes/web.php แล้วเพิ่ม routes ดังนี้



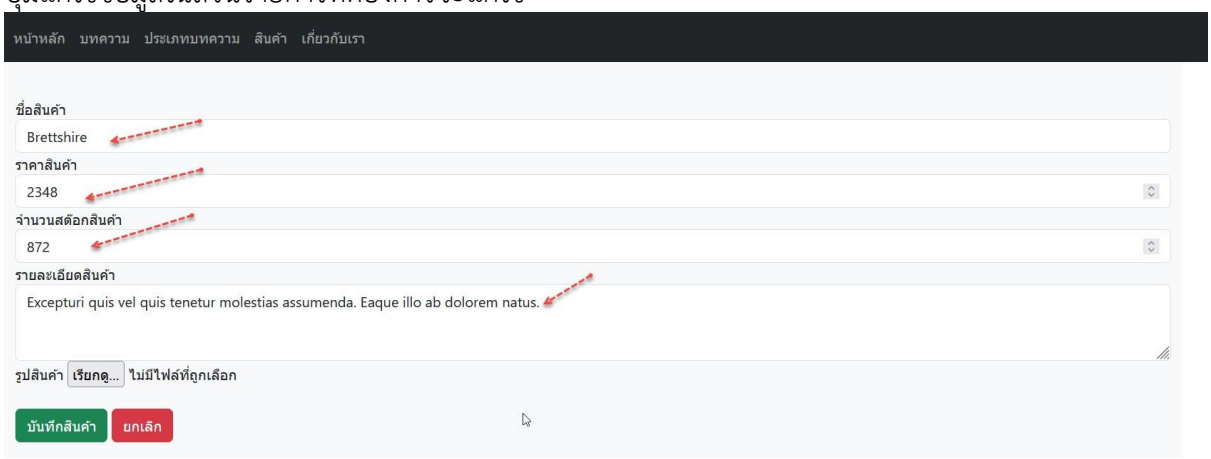
จากโค้ดที่เขียนเพิ่มเป็นการกำหนดเส้นทางเข้าถึงหน้าเว็บหรือเพิ่ม URL และเชื่อมโยงกับ ProductsController เมื่อมีการกดแก้ไขข้อมูลก็จะมาทำงานที่ URL products/edit/{id} จากนั้นก็จะส่งต่อไปให้คอนโทรลเลอร์ทำงานในเมธอด edit() เพื่อเรียกหน้าแก้ไขขึ้นมาแสดงทางหน้าจอ

5.เปิดไฟล์ ProductsController.php ในโฟลเดอร์ app\Http\Controllers\... แล้วเพิ่มเมธอด edit() แล้วเขียนโค้ดเข้าไป ดังนี้



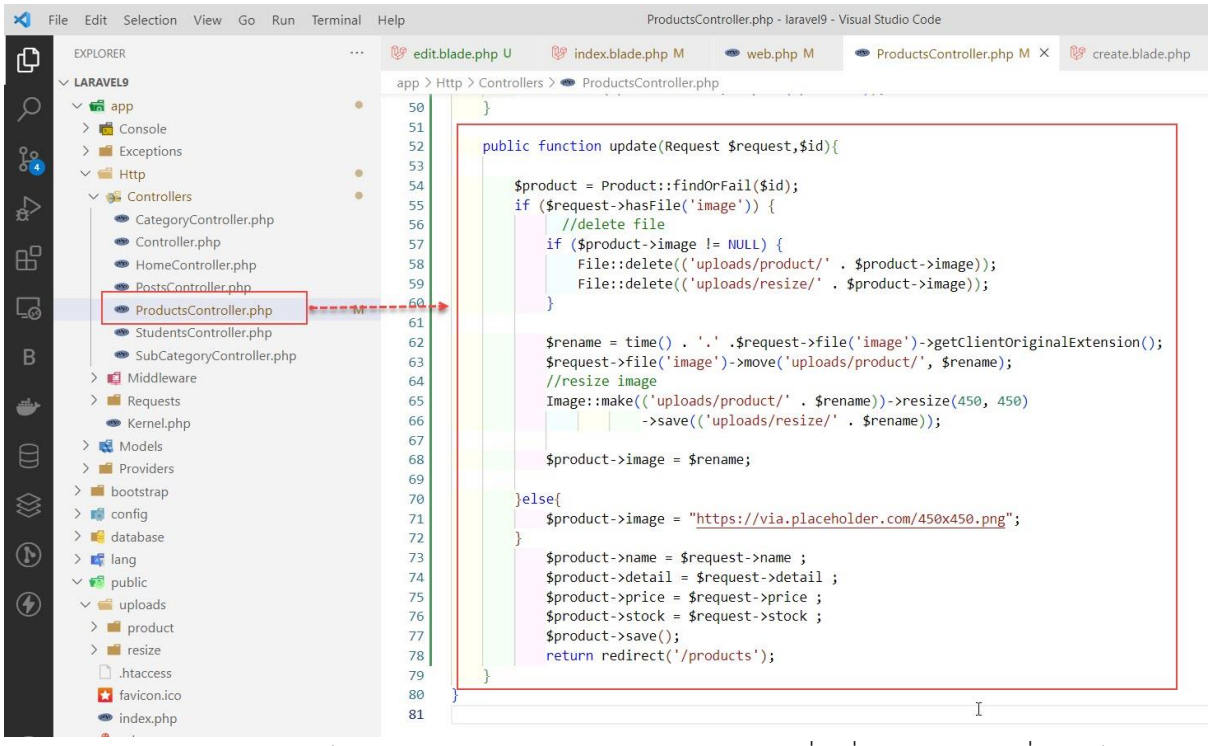
เมธอด edit() เป็นการเรียกดูข้อมูลตามรายการหรือ id ที่รับเข้ามา โดยจะใช้ Product::findOrFail($id) เก็บไว้ที่ตัวแปร $product จากนั้นก็ส่งตัวแปร $product ไปให้ส่วนของการแสดงผล

บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/product ที่ Browser แล้วคลิกที่ปุ่มแก้ไขข้อมูลในส่วนรายการที่ต้องการจะแก้ไข

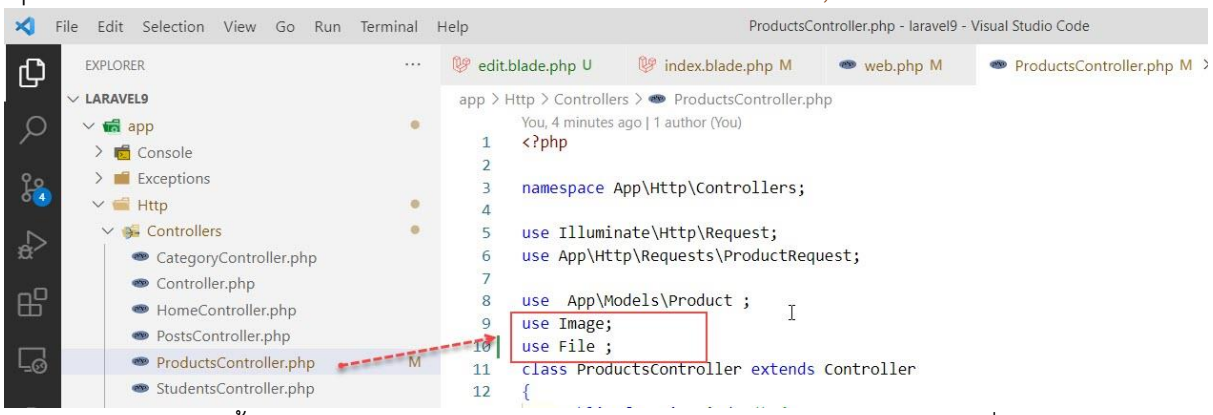


ข้อมูลก็จะถูกดึงขึ้นมาแสดงในช่องกรอกข้อมูลแต่ละช่องเรียบร้อยแล้ว

5.เปิดไฟล์ ProductsController.php ในโฟลเดอร์ app\Http\Controllers\... แล้วเพิ่มเมธอด update() เพื่อต่อจากเมธอด edit() แล้วเขียนโค้ดเข้าไป ดังนี้



จากตัวอย่างโค้ดจะเห็นว่าจะเหมือนกับตอนบันทึกข้อมูลเลย จุดที่เปลี่ยนคือ บรรทัดที่ 53 เป็นการกำหนดเงื่อนไขการแก้ไขข้อมูลตามค่าที่ส่งมา และบรรทัดที่ 56 – 59 เป็นเงื่อนไขการตรวจสอบไฟล์ที่อัปโหลดเข้ามา เมื่อมีไฟล์ใหม่อัปโหลดมาแล้วให้ทำการลบไฟล์เก่าที่อยู่ในโฟลเดอร์ upload/product และ upload/resize ออกก่อนแล้วค่อยบันทึกไฟล์ใหม่เข้าไปแทน ให้ use File ; เข้ามาใช้งานด้วยนะ



บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/product ที่ Browser แล้วคลิกที่ปุ่มแก้ไขข้อมูลในส่วนรายการที่ต้องการจะแก้ไข และบันทึกข้อมูลดูอีกครั้งก็เป็นอันว่าเราสามารถแก้ไขรายการข้อมูลได้แล้ว

หากพบว่าไฟล์ภาพที่อัปโหลดเข้าไปไม่แสดงผลภาพทางหน้าจอให้ไปแก้ไข path ภาพให้ถูกต้อง

```
// resources\views\product\index.blade.php
<img src="{{ asset('uploads/product/' . $product->image) }}
<img src="{{ asset('uploads/resize/' . $product->image) }}
```

ลบข้อมูลให้ปลอดภัยด้วย Laravel Collective

การลบข้อมูลนั้นสามารถทำได้หลายวิธี จากที่ผ่านมาเราได้เรียนรู้ไปบางส่วนแล้วนั้น ซึ่งเป็นวิธีที่ไม่อยากจะแนะนำให้ทำ เนื่องจากไม่มีความปลอดภัยอะไรเลย ไม่ปลอดภัยแบบไหนหรอ ให้ผู้อ่านนึกภาพแบบนี้นะ ผมจะยกตัวอย่างโค้ดที่เขียนจากที่ผ่านมา ดังนี้

ในส่วนโค้ดที่เขียนใน routes/web.php

```
Route::get('post/destroy/{id}',[PostsController::class,'destroy']);
```

ในส่วนโค้ดที่เขียนใน views

```
<a href="{{ route('destroy', $post->id) }}" class="btn btn-danger btn-sm"
onclick="return confirm('ยืนยันการลบหรือไม่')">ลบ</a>
```



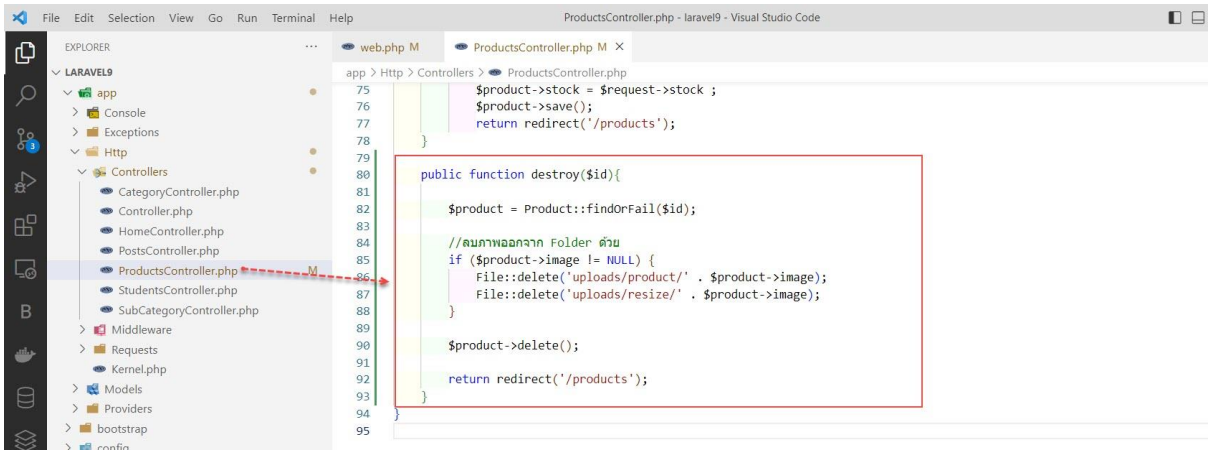
เห็นอะไรไหมครับ จะเห็นว่าเมื่อเราลองนำเม้าส์ไปชี้ที่ปุ่มลบข้อมูลแล้วสังเกตที่ด้านล่างตามที่ลูกศรชี้ไป ก็จะมี URL พร้อมแสดงค่าพารามิเตอร์ที่ถูกส่งไปปรากฏให้เห็น แบบนี้ไม่ค่อยจะปลอดภัยเท่าไหร่นัก ซึ่งถ้าเป็นแบบนี้ผู้ไม่หวังดีก็สามารถพิมพ์ URL ที่เห็นแล้ว ซึ่งถ้าไม่ได้ตรวจสอบให้ดีจะทำให้ข้อมูลถูกลบออกจากฐานข้อมูลเราได้ เพื่อป้องกันการโจมตีจากผู้ไม่หวังดี ให้ใช้อีกวิธี ซึ่งจะช่วยป้องกันการโจมตีจากผู้ไม่หวังดี และระบบของเราก็จะมีความปลอดภัยยิ่งขึ้น ให้ทำตามขั้นตอน ดังนี้

1.เปิดไฟล์ web.php ในโฟลเดอร์ routes/web.php แล้วแก้ไขโค้ดดังนี้



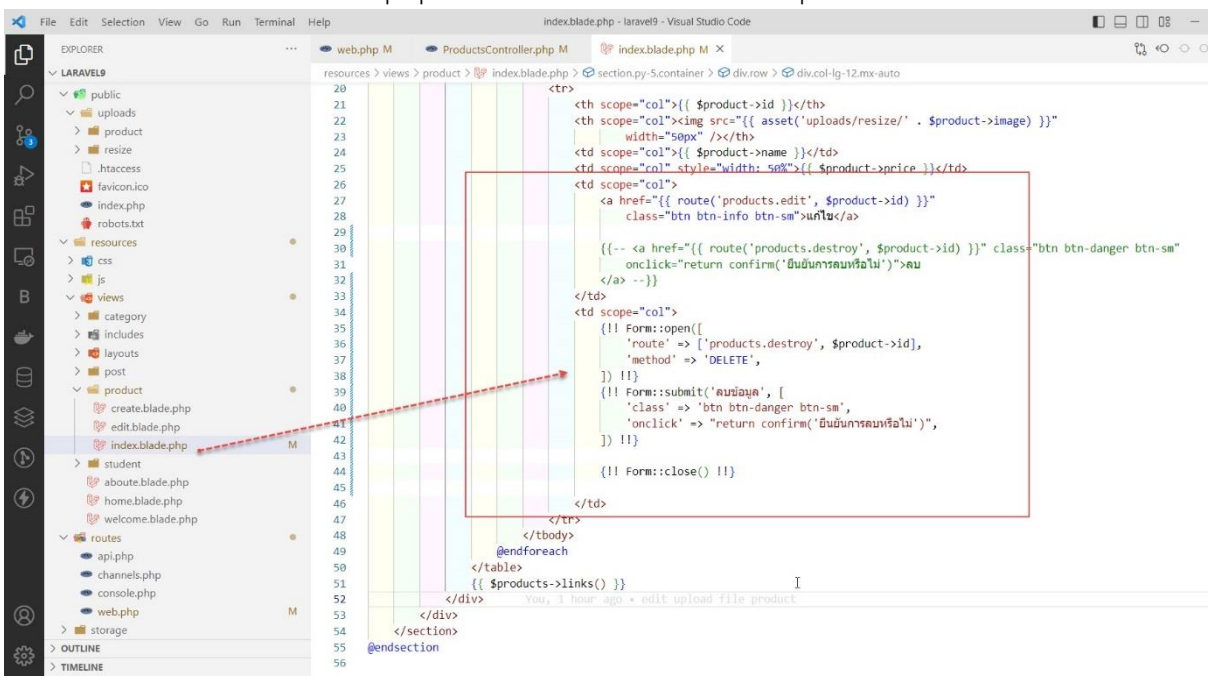
โดยเปลี่ยนรูปแบบจาก Route::get() เป็น Route::delete() พร้อมเชื่อมต่อกับคอนโทรลเลอร์และเมธอดที่ต้องการ จากตัวอย่างคือให้ชี้ไปที่ ProductsController และทำงานในส่วนของเมธอด destroy()

2.เปิดไฟล์ ProductsController.php ในโฟลเดอร์ app\Http\Controller\... แล้วเพิ่มเมธอด destroy() เขียนโค้ดเข้าไปดังนี้



จากตัวอย่างโค้ด เป็นการเขียนคำสั่งสำหรับลบข้อมูลออกจากฐานข้อมูลและลบภาพออกจากโฟลเดอร์ โดยจะลบข้อมูลตาม id ที่รับเข้ามา จากนั้นใช้ delete() เพื่อสั่งให้ลบ เมื่อลบข้อมูลเสร็จแล้วจะย้อนกลับไปหน้าหลักของข้อมูลสินค้า

3.เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources/views/product/... แล้วแก้ไขโค้ด ดังนี้

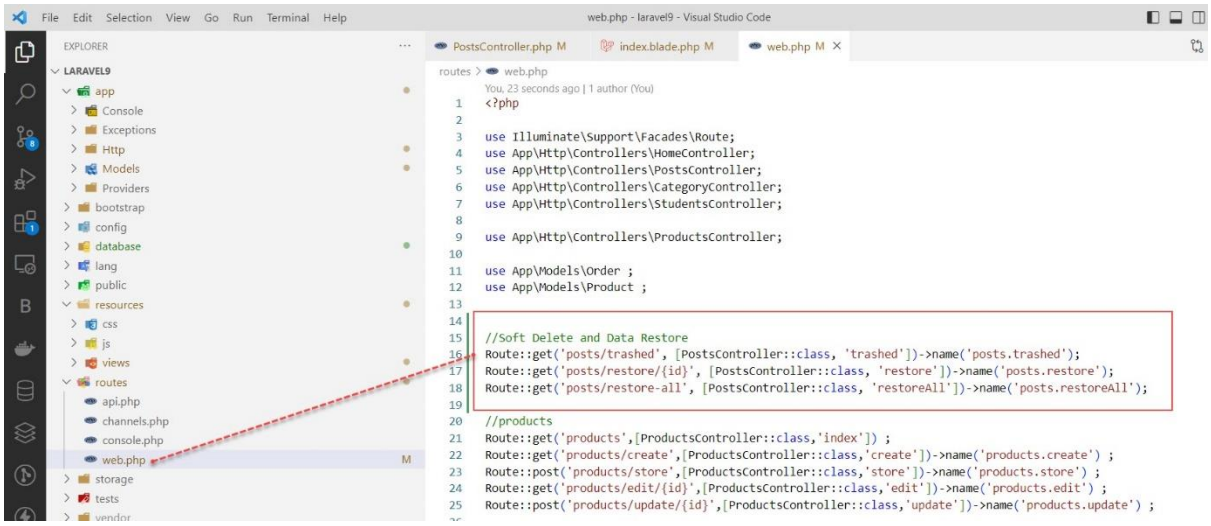


4.บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/product ที่ Browser แล้วคลิกที่ปุ่มลบข้อมูลในส่วนรายการที่ต้องการจลบ จะเห็นว่าเมื่อเราลองนำเม้าส์ไปชี้ที่ปุ่มลบข้อมูลอีกครั้งก็จะไม่เห็น URL แสดงให้เห็นแล้ว ...

Soft Delete and Data Restore – การลบข้อมูลและการกู้คืน

การลบข้อมูลนั้นจากที่ผ่านมาเมื่อเราลบข้อมูลไปแล้วข้อมูลก็จะถูกลบออกจากตารางทันที ซึ่งไม่สามารถกู้คืนได้ แต่หากว่าในระบบของเราต้องการจะกู้คืนข้อมูลที่ถูกลบไปนั้น เราสามารถทำได้ อารมณ์จะเหมือนกับเราลบอีเมลแล้วจะไปตกอยู่ที่ถังขยะ ประมาณนั้นครับ อยากจะกู้คืนตอนไหนก็สามารถทำได้ ใน Laravel จะมีคุณสมบัตินี้ให้เราใช้งานนั่นคือ SoftDeletes มาดูวิธีการใช้งานได้เลย โดยในตัวอย่างนี้จะยกตัวอย่างการลบบทความแล้วทำการกู้คืนบทความที่ถูกลบ ดังนี้

1.เปิดไฟล์ routes/web.php แล้วเพิ่มโค้ด เพื่อกำหนดเส้นทางการทำงาน ดังนี้



ในโค้ดเป็นการเพิ่มเส้นทางสำหรับการเข้าถึงคือ posts/trashed สำหรับดูข้อมูลที่ถูกลบทั้งหมด posts/restore/{id} สำหรับกู้คืนบทความที่ถูกลบไปแล้ว posts/restore-all สำหรับกู้คืนบทความทั้งหมด

2.เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources/views/post/... แล้วเพิ่มลิงค์สำหรับการกู้คืนข้อมูลที่ถูกลบดังนี้

ส่วนที่ 1

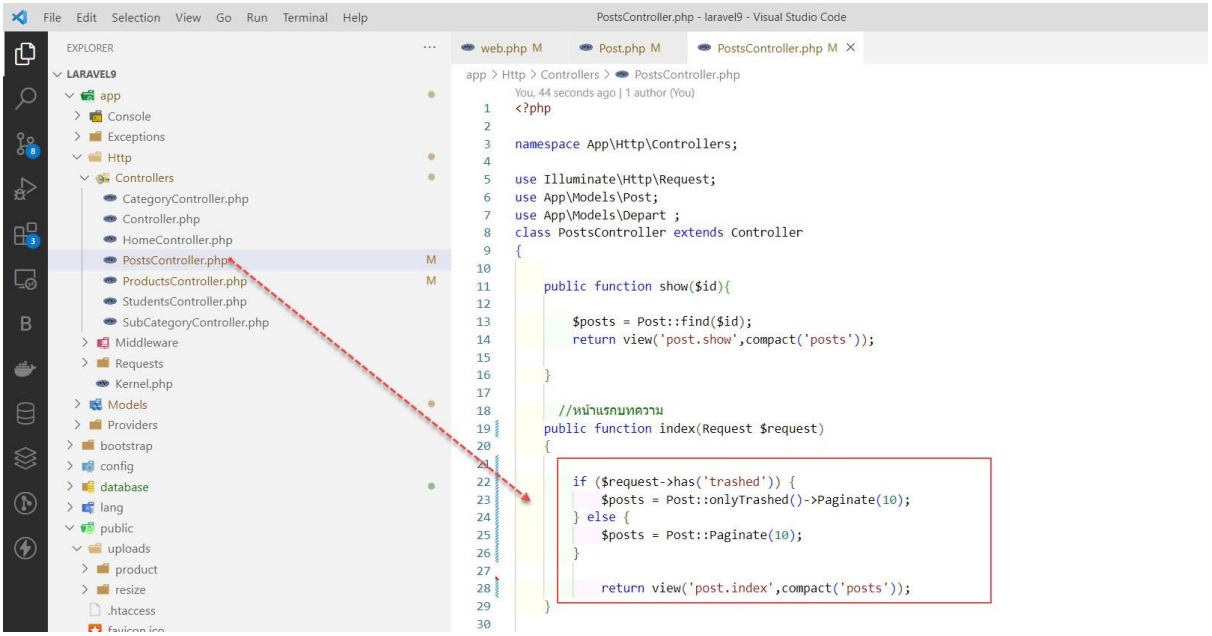
```
@if (request()->has('trashed'))
    <a href="{{ route('posts.index') }}" class="btn btn-info">ดูบทความทั้งหมด</a>
    <a href="{{ route(' posts.restoreAll')  }}" class="btn btn-success">กู้คืนทั้งหมด</a>
@else
    <a href="{{ route('posts.index', ['trashed' => 'post']) }}" class="btn btn-
primary">บทความที่ถูกลบ</a>
@endif
```

```
 @if (request()->has('trashed'))
    <a href="{{ route('posts.restore', $post->id) }}"class="btn btn-success btn-
sm">
กู้คืน</a>
@else
    <a href="{{ route('destroy', $post->id) }}" class="btn btn-danger btn-sm"
    onclick="return confirm('ยืนยันการลบหรือไม่')">ลบ</a>
@endif
```

โค้ดทั้งหมด

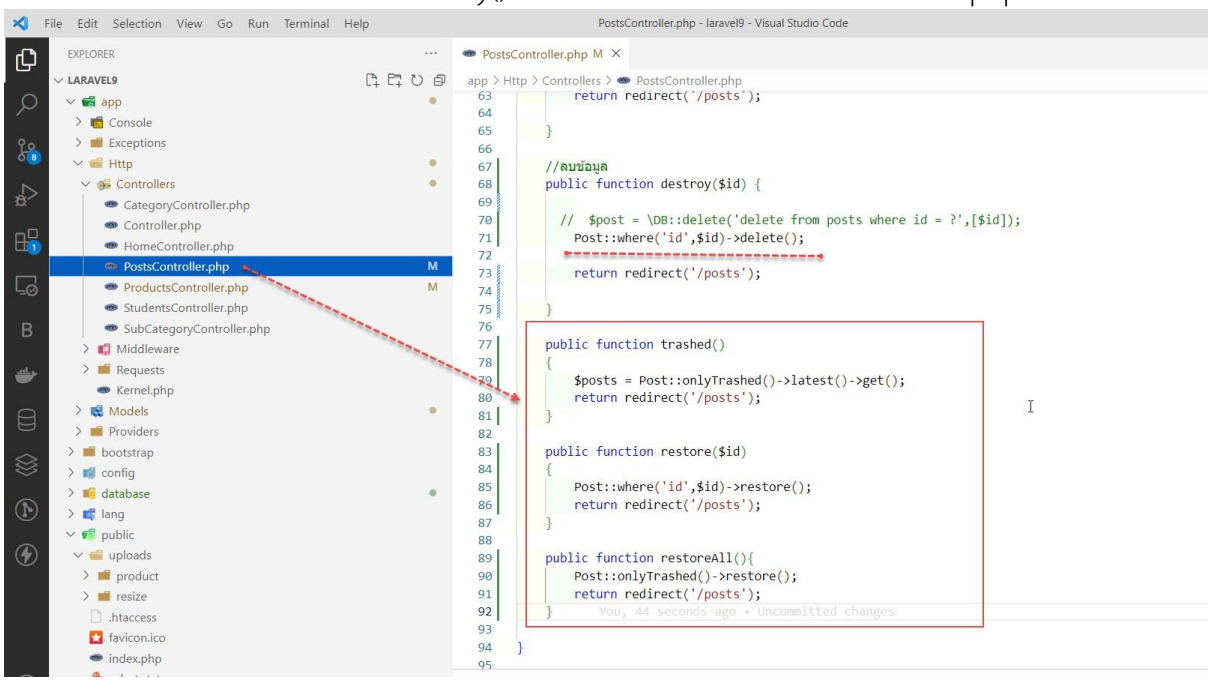
```
//resources/views/post/index.blade.php
@extends('layouts.frontend')
@section('title', 'หน้าแรกบทความ')
@section('content')
<section class="py-5 container">
   <div class="row">
     <div class="col-lg-12  mx-auto">
       <br />
       <div class="float-end">
           @if (request()->has('trashed'))
 <a href="{{ route('posts.index') }}" class="btn btn-info">ดูบทความทั้งหมด</a>
 <a href="{{ route(' posts.restoreAll')  }}" class="btn btn-success">กู้คืนทั้งหมด</a>
           @else
 <a href="{{ route('posts.index', ['trashed' => 'post']) }}" class="btn btn-primary">บทความที่ถูกลบ</a>
           @endif
      </div>
           <table class="table table-hover">
               <thead><tr>
             <th scope="col">ลำดับ</th>
             <th scope="col">หัวข้อ</th>
             <th scope="col">รายละเอียด</th>
             <th scope="col">จัดการ</th></tr>
               </thead>
             @foreach ($posts as $post)
               <tbody><tr>
               <th scope="col">{{ $post->id }}</th>
               <td scope="col">{{ $post->post_title }}</td>
               <td scope="col" style="width: 50%">{{ $post->post_detail }}</td>
               <td scope="col">
<a href="{{ route('show', $post->id) }}" class="btn btn-success btn-sm">ดู</a>|
<a href="{{ route('edit', $post->id) }}" class="btn btn-info btn-sm">แก้ไข</a>|
               @if (request()->has('trashed'))
<a href="{{ route('posts.restore', $post->id) }}" class="btn btn-success btn-sm">กู้คืน</a>
               @else
<a href="{{ route('destroy', $post->id) }}" class="btn btn-danger btn-sm"
onclick="return confirm('ยืนยันการลบหรือไม่')">ลบ</a>
               @endif
               </td></tr></tbody>
            @endforeach
                </table>
                {{ $posts->links() }}
            </div>
        </div>
    </section>
@endsection
```

3.เปิดไฟล์ PostsController.php ในโฟลเดอร์ app\Http\Controller\... แล้วแก้ไขโค้ดในเมธอด index() ดังนี้



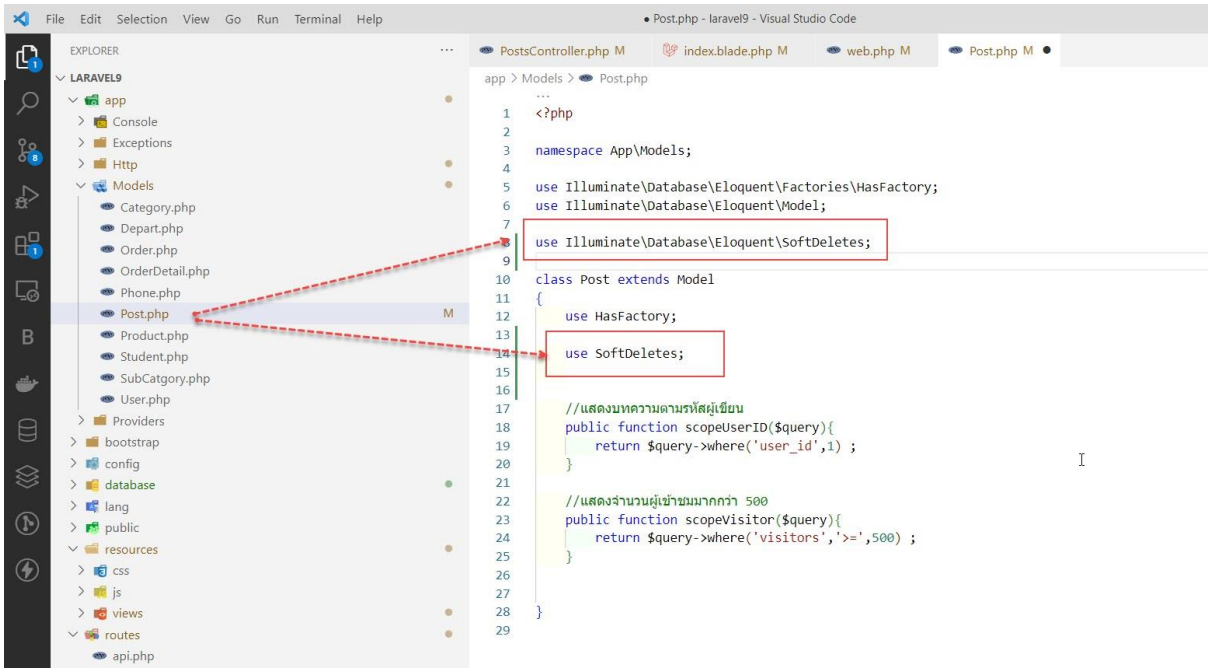
ในเมธอด index() ได้มีการใส่พารามิเตอร์ Request $request เพื่อรับค่าเมื่อมีการคลิกลิงค์เข้ามา แล้วตรวจสอบ 'trashed' หากมีการคลิกลิงค์บทความที่ถูกลบ รายการที่ถูกลบก็จะแสดงให้เห็น

จากนั้นแก้ไขโค้ดในเมธอด destroy() และเพิ่มเมธอดใหม่ใน PostsController.php ดังนี้

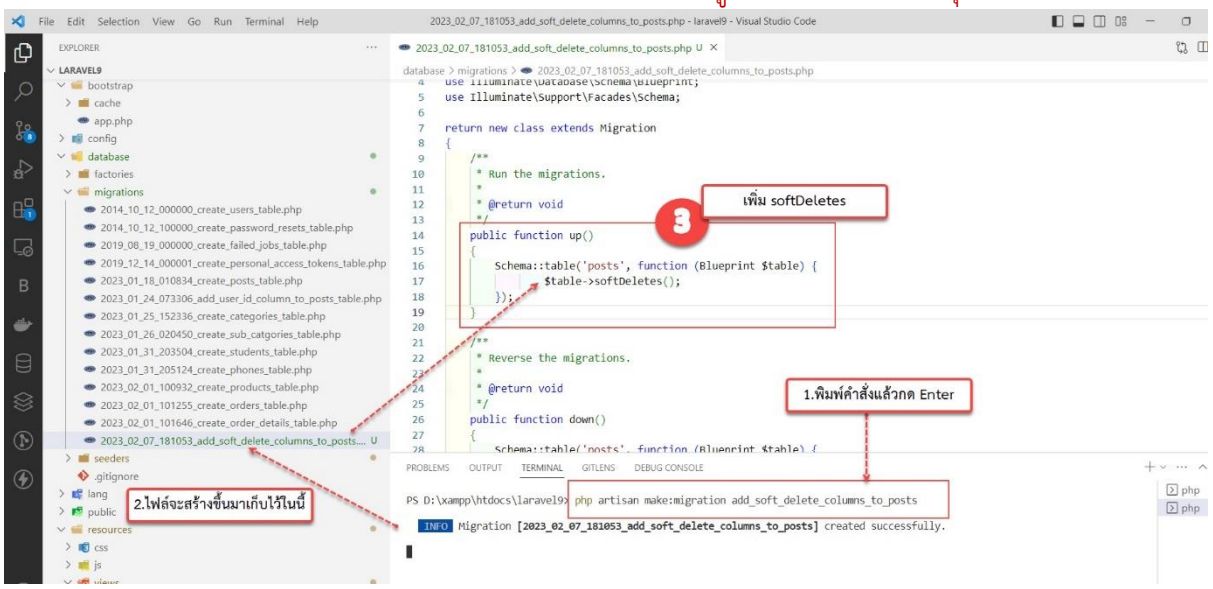


จากตัวอย่างโค้ดได้ทำการเพิ่มเมธอด trashed() ใช้สำหรับแสดงรายการข้อมูลที่ถูกลบ โดยเรียงลำดับจากรายการที่ลบล่าสุด เมธอด restore() ใช้สำหรับกู้คืนข้อมูลตามรายการที่ต้องการ restoreAll() ใช้สำหรับกู้คืนข้อมูลทั้งหมด

4.เปิดไฟล์ Post.php ในโฟลเดอร์ app\Models\... แล้วเพิ่มโค้ดเข้าไป ดังนี้



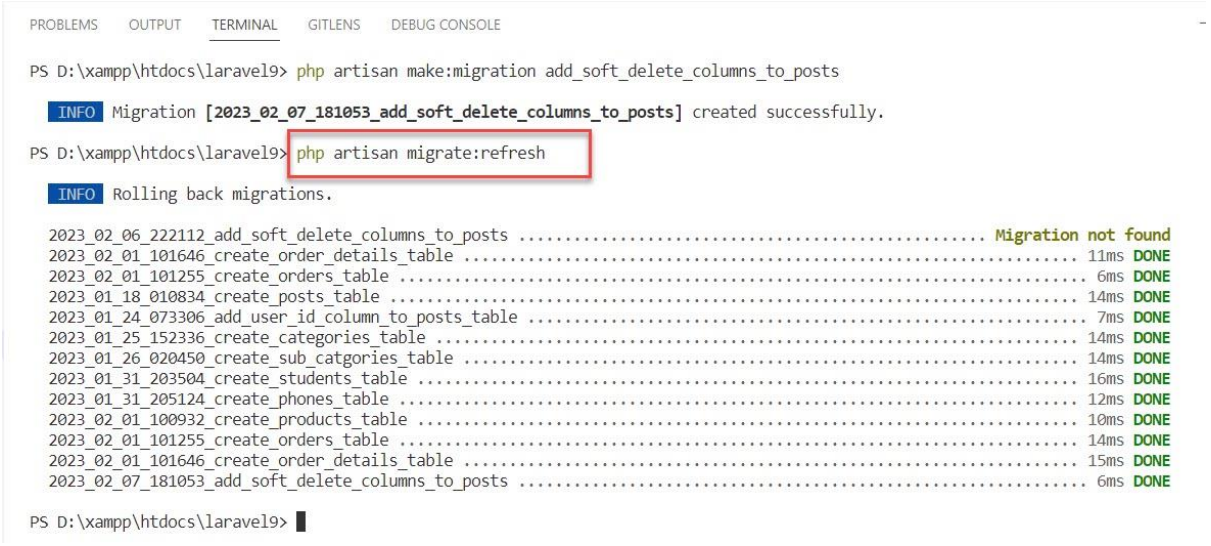
5.เปิด Terminal แล้วพิมพ์คำสั่ง ดังนี้ (ถ้า server รันอยู่ให้กด ctrl+c เพื่อหยุดการทำงาน server ก่อน)



6.รันคำสั่ง rollback , migrate และ seed เพื่อลบตารางทั้งหมดแล้วสร้างตารางใหม่ และใส่ข้อมูลตั้งต้นใหม่ โดยพิมพ์คำสั่งแล้วกด Enter ตามลำดับดังนี้

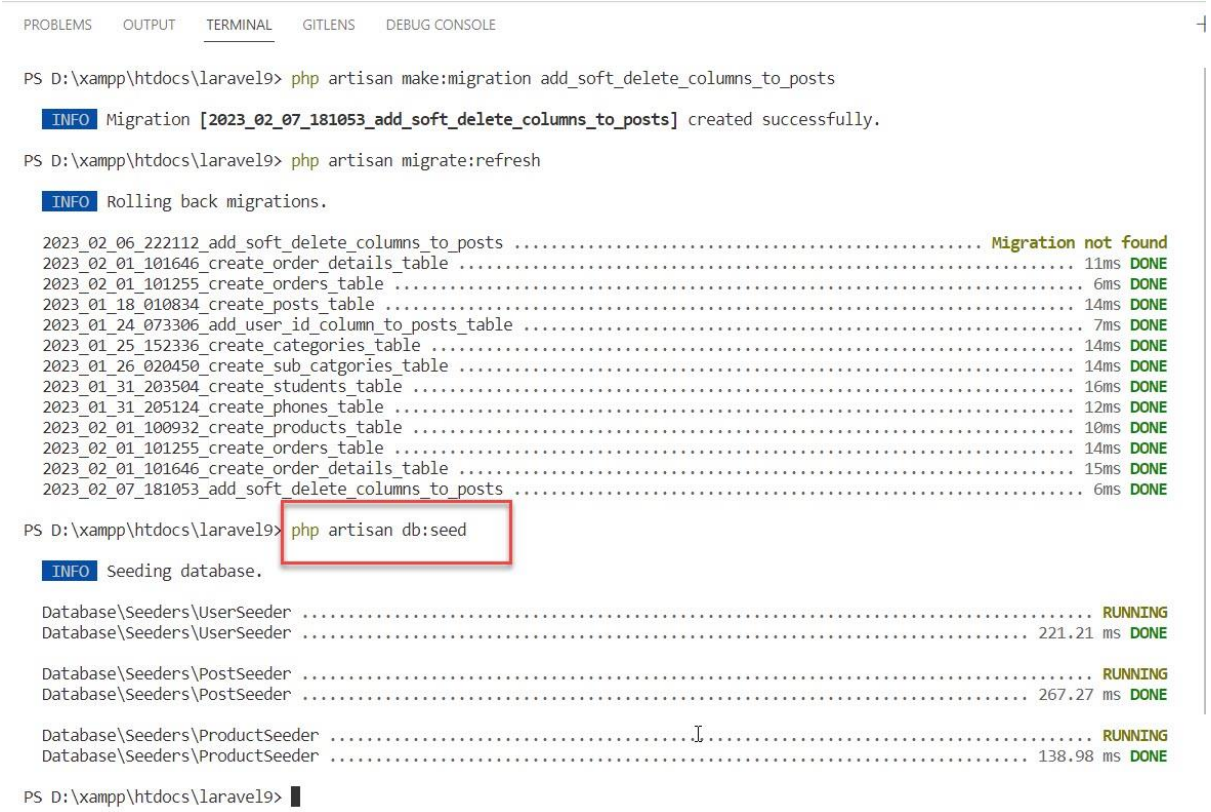
```
php artisan migrate:refresh
php artisan db:seed
```

พิมพ์คำสั่ง php artisan migrate:refresh ผ่าน Terminal แล้วกด Enter



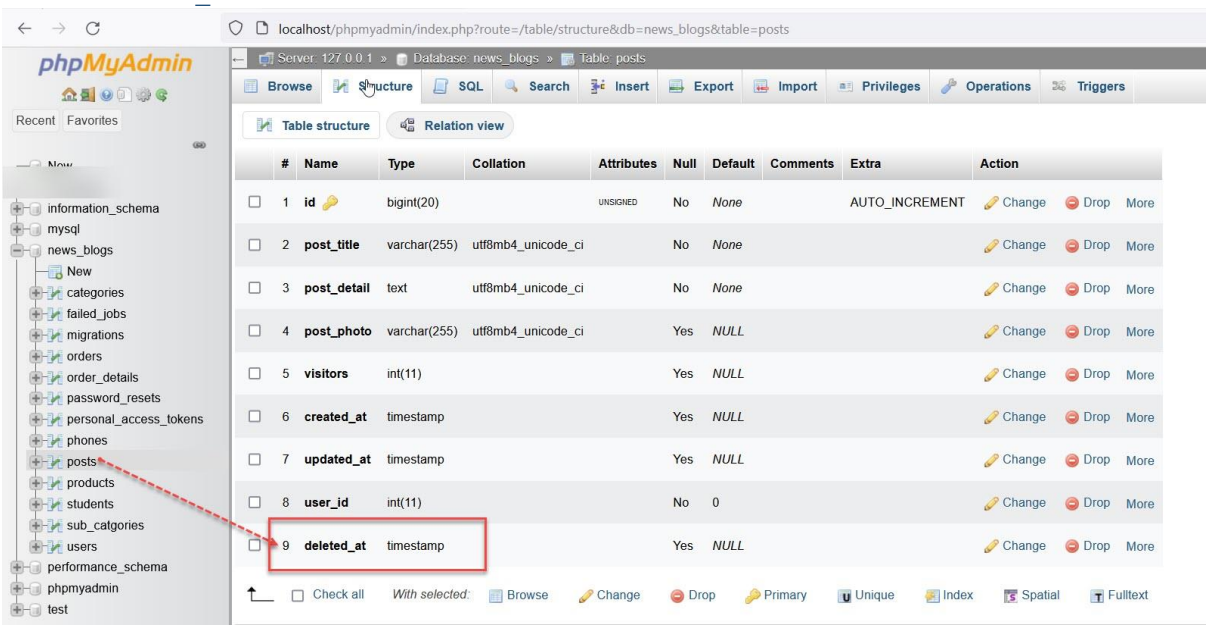
```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE

PS D:\xampp\htdocs\laravel9> php artisan make:migration add_soft_delete_columns_to_posts

  INFO  Migration [2023_02_07_181053_add_soft_delete_columns_to_posts] created successfully.

PS D:\xampp\htdocs\laravel9> php artisan migrate:refresh

  INFO  Rolling back migrations.

  2023_02_06_222112_add_soft_delete_columns_to_posts ..... Migration not found
  2023_02_01_101646_create_order_details_table ..... 11ms DONE
  2023_02_01_101255_create_orders_table ..... 6ms DONE
  2023_01_18_010834_create_posts_table ..... 14ms DONE
  2023_01_24_073306_add_user_id_column_to_posts_table ..... 7ms DONE
  2023_01_25_152336_create_categories_table ..... 14ms DONE
  2023_01_26_020450_create_sub_catgories_table ..... 14ms DONE
  2023_01_31_203504_create_students_table ..... 16ms DONE
  2023_01_31_205124_create_phones_table ..... 12ms DONE
  2023_02_01_100932_create_products_table ..... 10ms DONE
  2023_02_01_101255_create_orders_table ..... 14ms DONE
  2023_02_01_101646_create_order_details_table ..... 15ms DONE
  2023_02_07_181053_add_soft_delete_columns_to_posts ..... 6ms DONE

PS D:\xampp\htdocs\laravel9>
```

พิมพ์คำสั่ง php artisan db:seed แล้วกด Enter



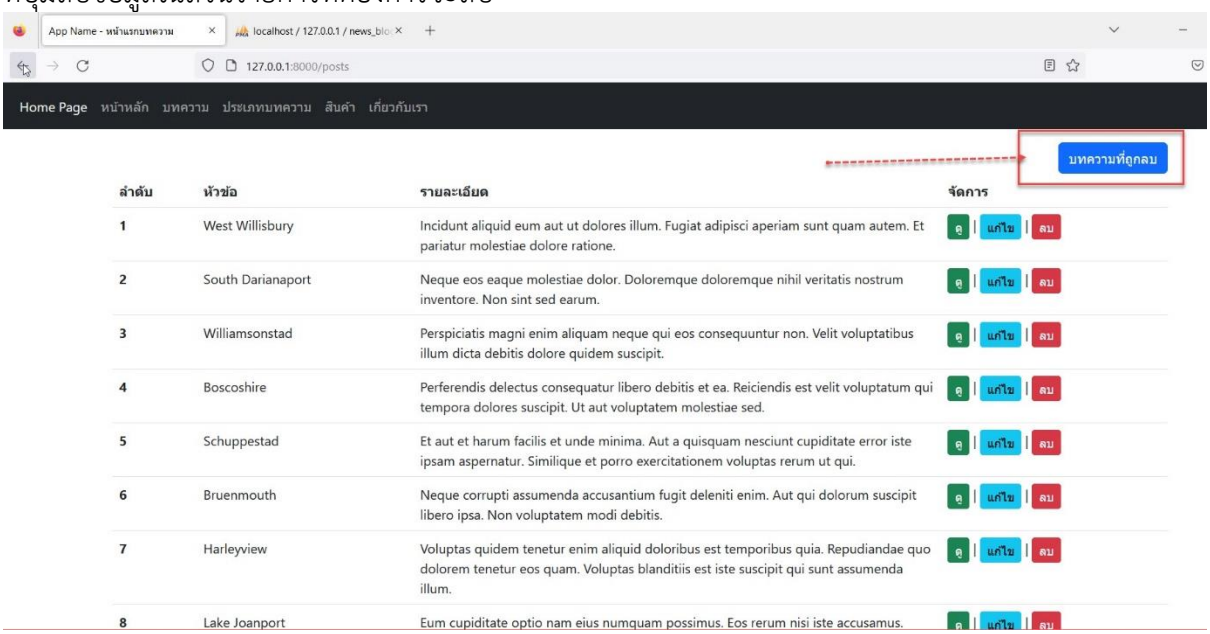
```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE

PS D:\xampp\htdocs\laravel9> php artisan make:migration add_soft_delete_columns_to_posts

  INFO  Migration [2023_02_07_181053_add_soft_delete_columns_to_posts] created successfully.

PS D:\xampp\htdocs\laravel9> php artisan migrate:refresh

  INFO  Rolling back migrations.

  2023_02_06_222112_add_soft_delete_columns_to_posts ..... Migration not found
  2023_02_01_101646_create_order_details_table ..... 11ms DONE
  2023_02_01_101255_create_orders_table ..... 6ms DONE
  2023_01_18_010834_create_posts_table ..... 14ms DONE
  2023_01_24_073306_add_user_id_column_to_posts_table ..... 7ms DONE
  2023_01_25_152336_create_categories_table ..... 14ms DONE
  2023_01_26_020450_create_sub_catgories_table ..... 14ms DONE
  2023_01_31_203504_create_students_table ..... 16ms DONE
  2023_01_31_205124_create_phones_table ..... 12ms DONE
  2023_02_01_100932_create_products_table ..... 10ms DONE
  2023_02_01_101255_create_orders_table ..... 14ms DONE
  2023_02_01_101646_create_order_details_table ..... 15ms DONE
  2023_02_07_181053_add_soft_delete_columns_to_posts ..... 6ms DONE

PS D:\xampp\htdocs\laravel9> php artisan db:seed

  INFO  Seeding database.

  Database\Seeders\UserSeeder ..... RUNNING
  Database\Seeders\UserSeeder ..... 221.21 ms DONE

  Database\Seeders\PostSeeder ..... RUNNING
  Database\Seeders\PostSeeder ..... 267.27 ms DONE

  Database\Seeders\ProductSeeder ..... RUNNING
  Database\Seeders\ProductSeeder ..... 138.98 ms DONE

PS D:\xampp\htdocs\laravel9>
```

6.เปิดโปรแกรม phpMyAdmin ขึ้นมาแล้วตรวจสอบดูตาราง posts ก็จะมีฟิวด์เพิ่มขึ้นมาให้ใหม่อีก 1 ฟิวด์คือ deleted_at

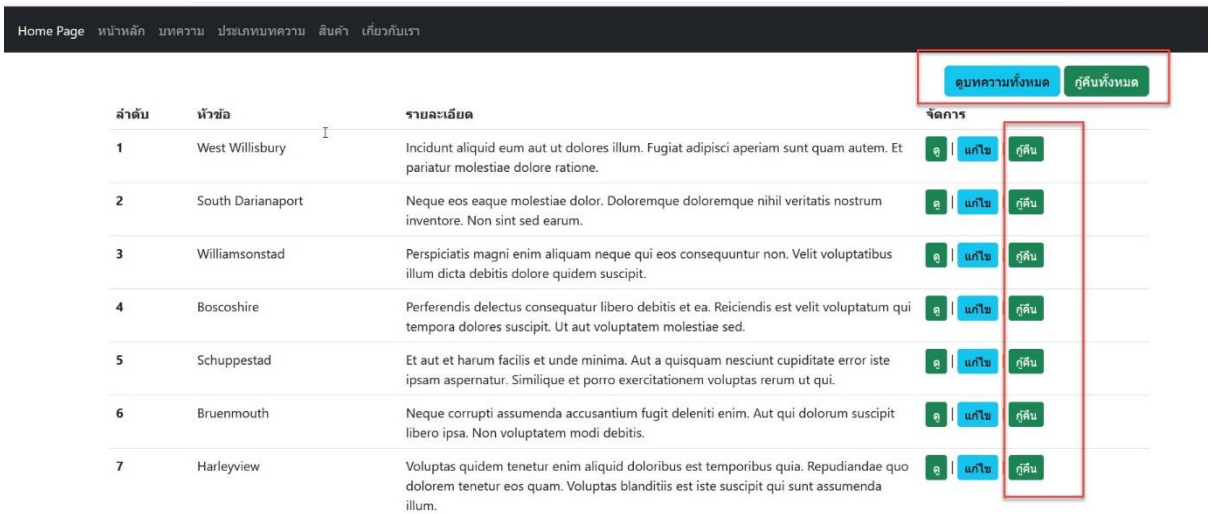


ก็เป็นอันว่าเสร็จขั้นตอนการสร้างแล้ว ต่อไปก็ให้ทดสอบดูการแสดงผล

6.บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/product ที่ Browser แล้วคลิกที่ปุ่มลบข้อมูลในส่วนรายการที่ต้องการจะลบ



เมื่อคลิกที่ปุ่ม บทความที่ถูกลบ ก็จะเห็นรายการที่ถูกลบทั้งหมด

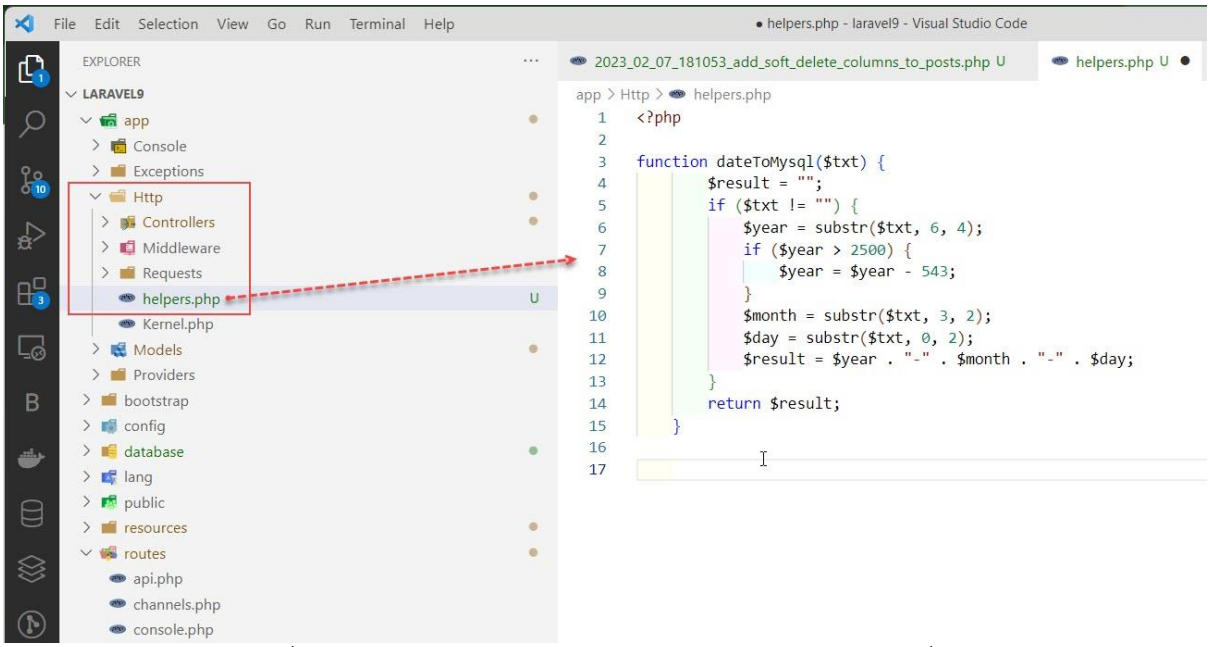


เมื่อกดที่ปุ่ม กู้คืน แต่ละรายการที่ต้องการ รายการนั้นก็จะปรากฏให้เราเห็นเหมือนเดิม

Helper Functions

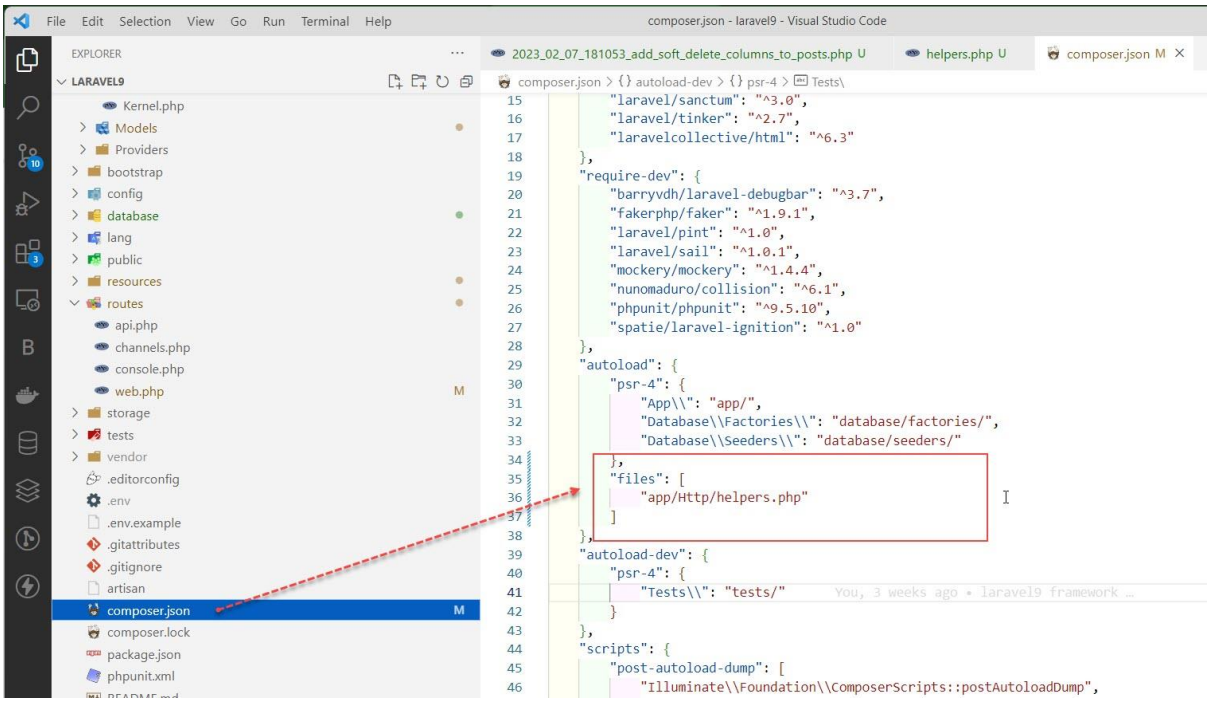
ทุก ๆ ครั้งที่เราอยากมีฟังก์ชันเป็นของตัวเอง แล้วอยากให้สามารถเรียกใช้งานจากที่ไหนก็ได้ ไม่ว่าจะเรียกใช้ที่ Model เรียกใช้ที่ Controller หรือจะเรียกใช้ที่ views ก็สามารถทำได้เช่นกัน สิ่งนั้นเรียกว่า Helper นั่นเอง การสร้าง Helper นั้นทำได้ไม่ยาก โดยมีวิธีการสร้างและเรียกใช้ ตามขั้นตอนนี้

1.สร้างไฟล์ชื่อว่า helpers.php ไว้ในโฟลเดอร์ app\Http\... แล้วเขียนโค้ดดังนี้



จากตัวอย่างโค้ดที่เขียนด้านบน เป็นการสร้างฟังก์ชันสำหรับแปลงข้อมูลวันที่ จาก Text ลงฐานข้อมูล โดยรับค่าเข้ามาเป็นในรูปแบบ วว/ดด/ปปปป แล้วแปลงค่ากลับไปเป็น yyyy/mm/dd

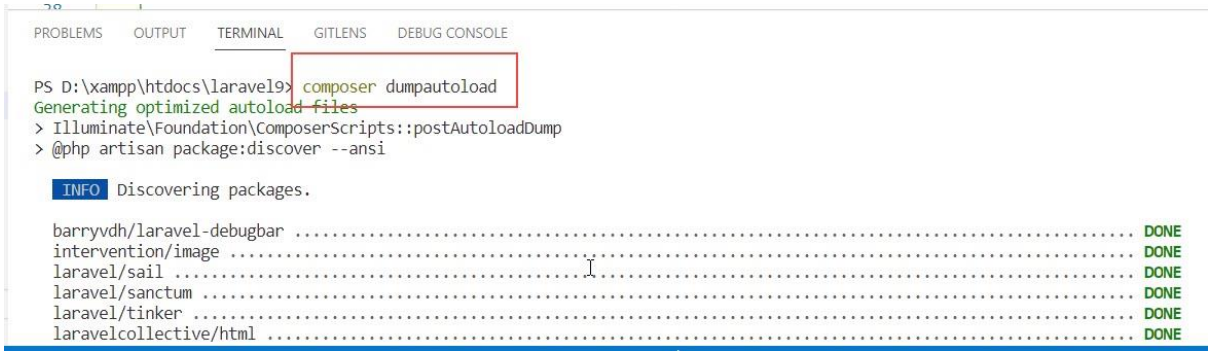
2.เปิดไฟล์ composer.json แล้วตั้งค่าเพิ่มเติมโดยเพิ่มโค้ดลงไปในส่วนของ `autoload` ดังนี้



เป็นการกำหนด path ของ Helper ที่เราสร้างขึ้นเอง ในตัวอย่างได้สร้างไฟล์ Helper เก็บไว้ที่โฟลเดอร์ app\Http\...

จากนั้นให้รันคำสั่งผ่าน Terminal ดังนี้

```
composer dumpautoload
```



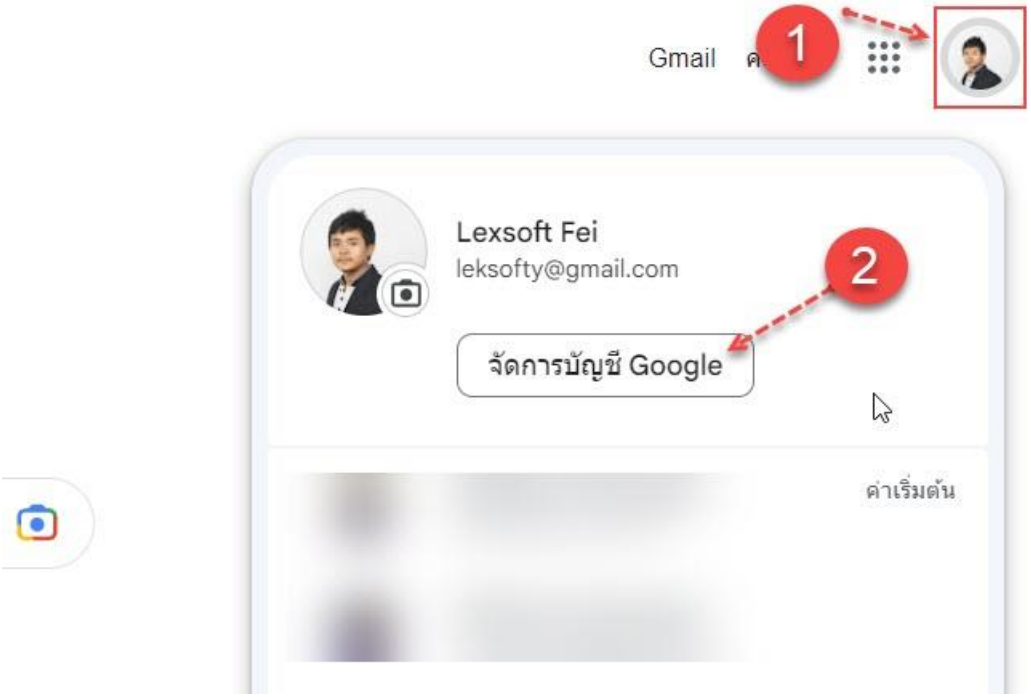
เวลาจะเรียกใช้งานก็สามารถเรียกใช้งานได้เลย โดยที่ไม่ต้อง use ... ที่ด้านบน

```
$input->datesave = dateToMySql('06/05/2528') ;
```

Sending Email

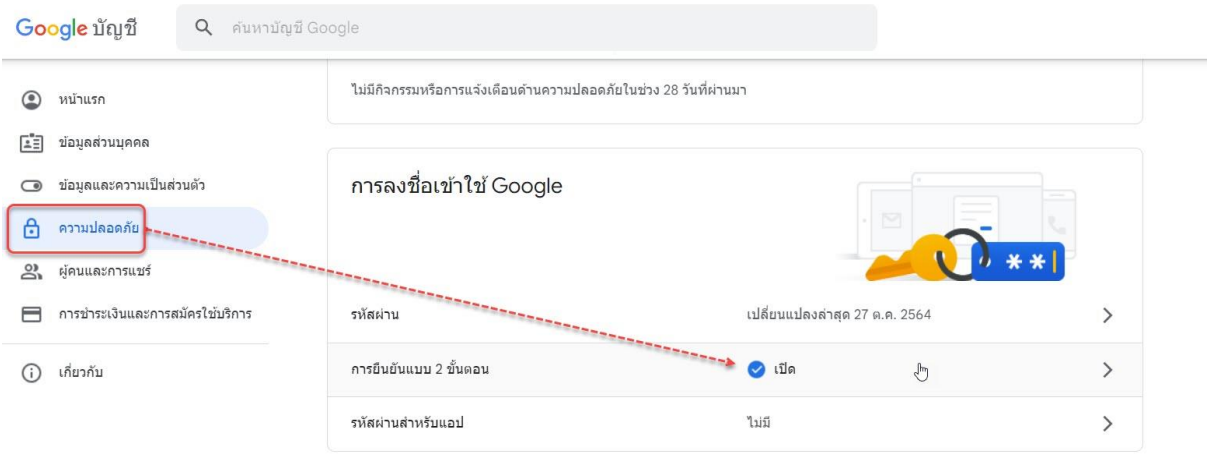
การส่งอีเมลใน Laravel นั้นเป็นเรื่องที่ง่ายมาก ๆ สามารถบริหารจัดการไฟล์ได้สะดวก เช่น การจัดการรูปแบบหน้าตาของอีเมล ในลักษณะต่าง ๆ นอกจากนั้นยังรองรับการส่งอีเมล หลาย ๆ ค่าย มี driver ให้เลือกใช้ เช่น smtp, sendmail, mailgun , mailtrap โดยในตัวอย่างนี้จะยกตัวอย่างการส่งอีเมลแบบ smtp ขั้นตอนการสร้างและส่งอีเมล ดังนี้

1.ตั้งค่าให้รองรับการการส่งอีเมลใน Gmail

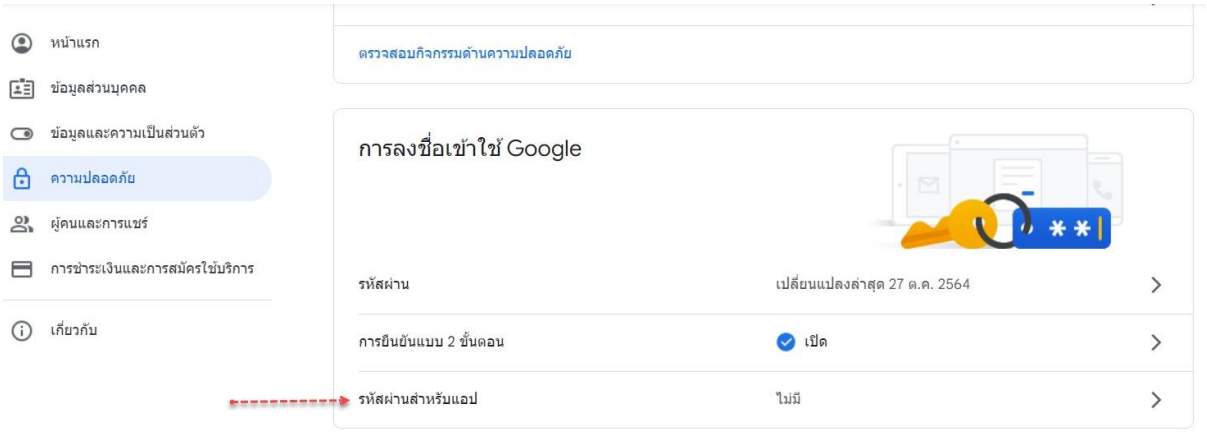


หมายเลข 1 คลิกที่ไอคอน รูป

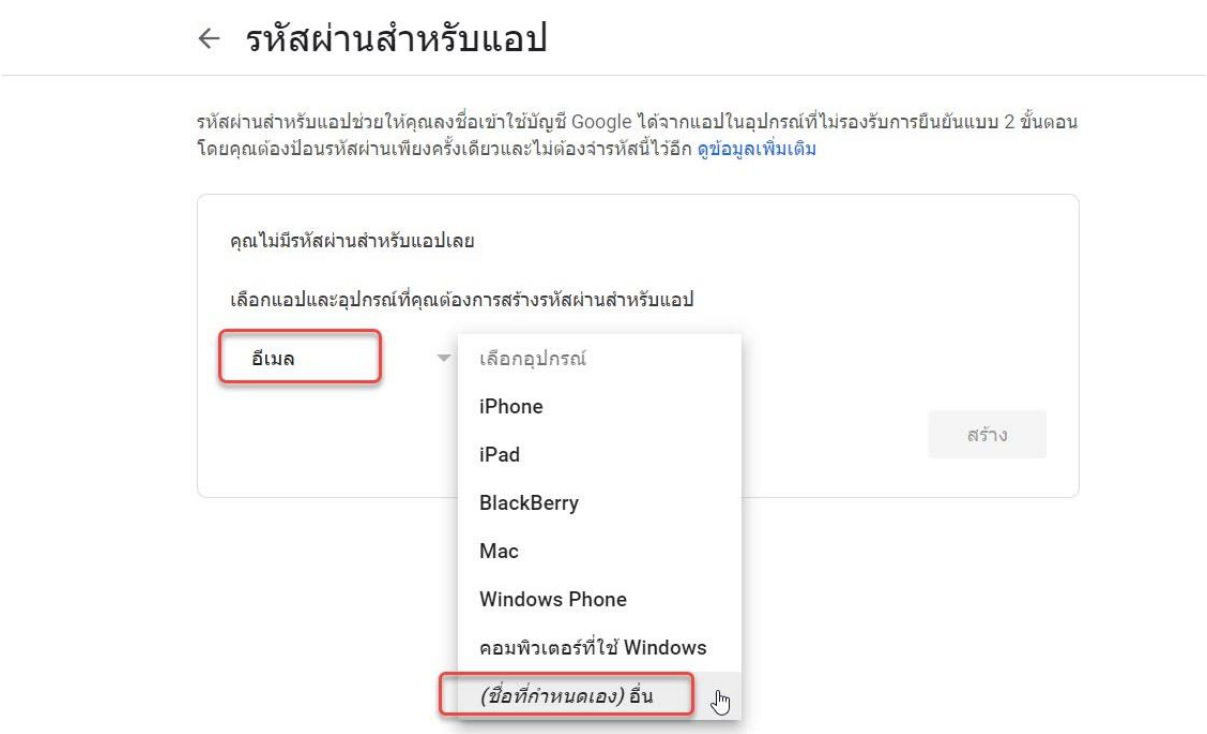
หมายเลข 2 คลิกเลือกจัดการบัญชี Google



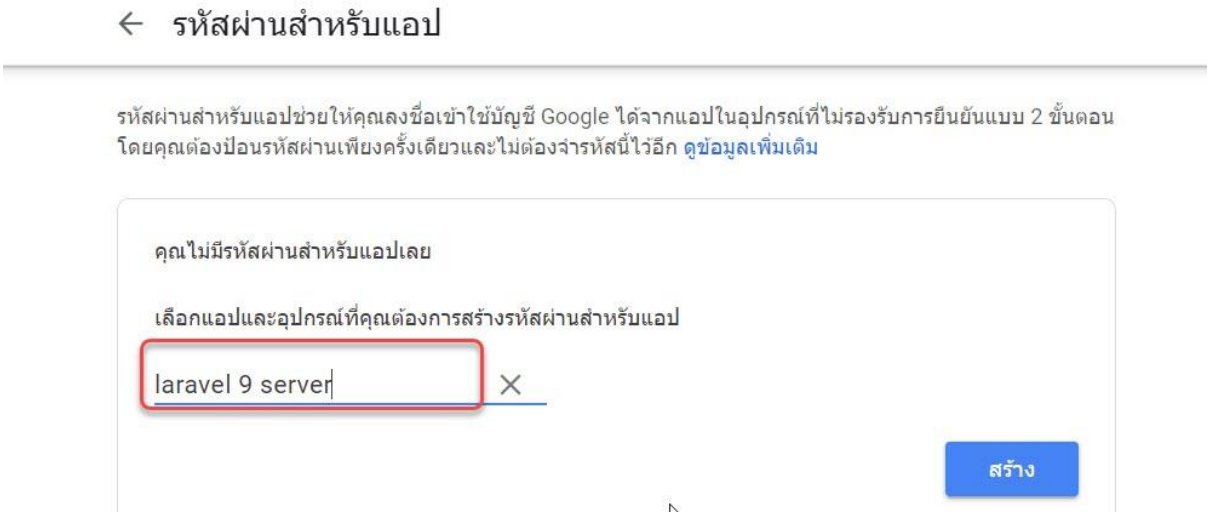
คลิกที่เมนู **ความปลอดภัย** จากนั้นคลิกเปิด การยืนยันแบบ 2 ขั้นตอน



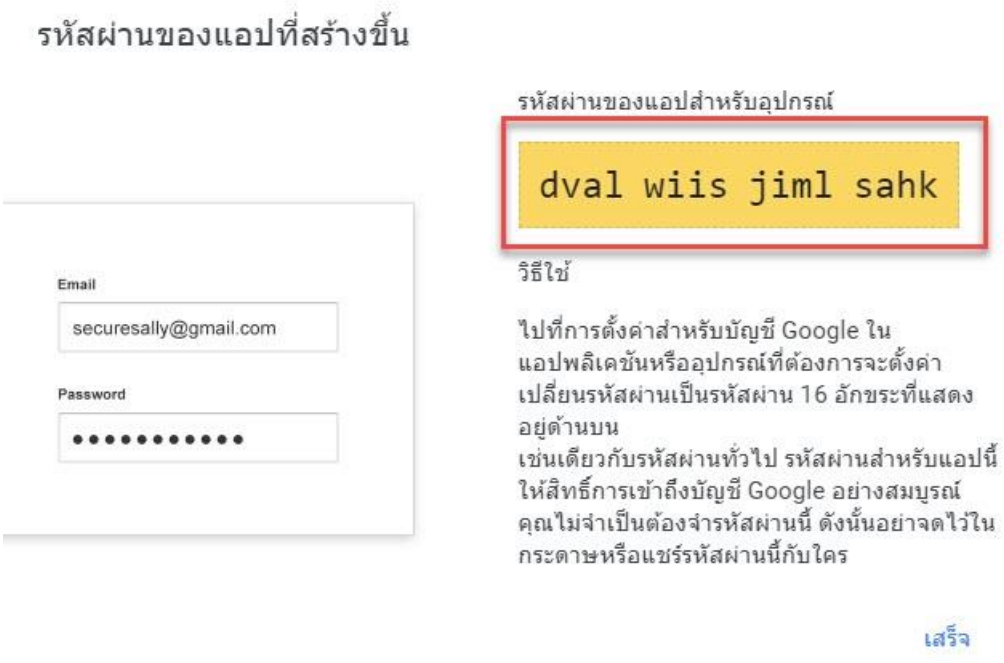
คลิกที่ **รหัสผ่านสำหรับแอป**



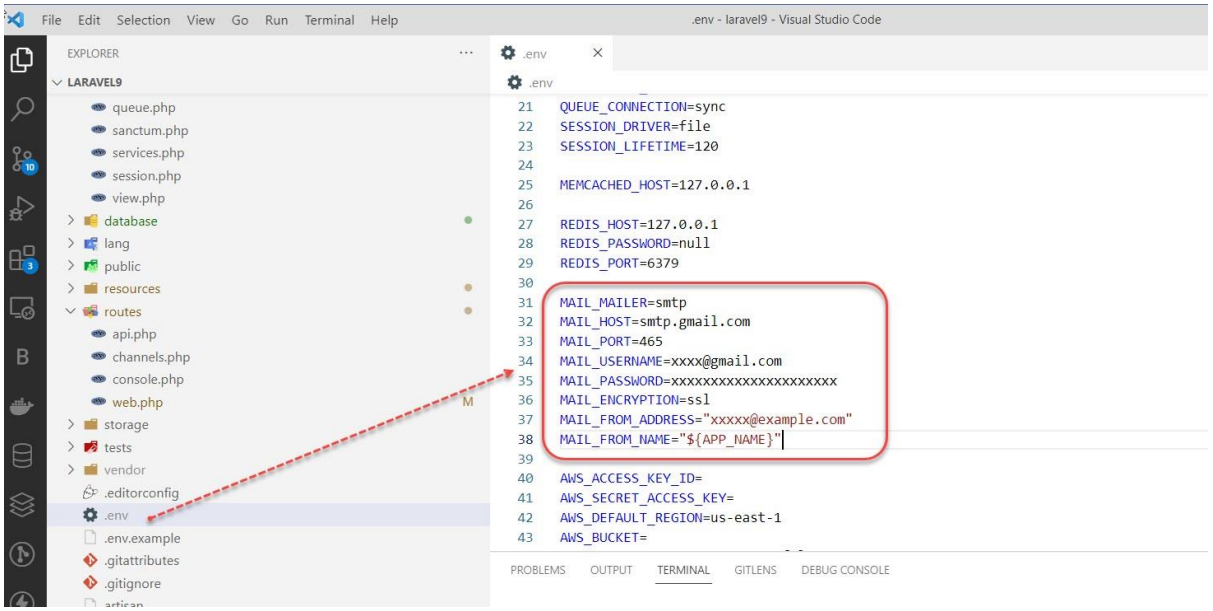
ช่องที่ 1 เลือกรูปแบบเป็นอีเมลเท่านั้น และช่องที่สอง เลือก (ชื่อที่กำหนดเอง) อื่น ๆ แล้วตั้งชื่อ server



ให้กรอกชื่อ server และกดปุ่มสร้าง ก็จะได้รับรหัส 16 หลัก โดยจะใช้รหัส 16 หลักนี้แทน password ให้คัดลอกเก็บไว้ด้วยนะครับ

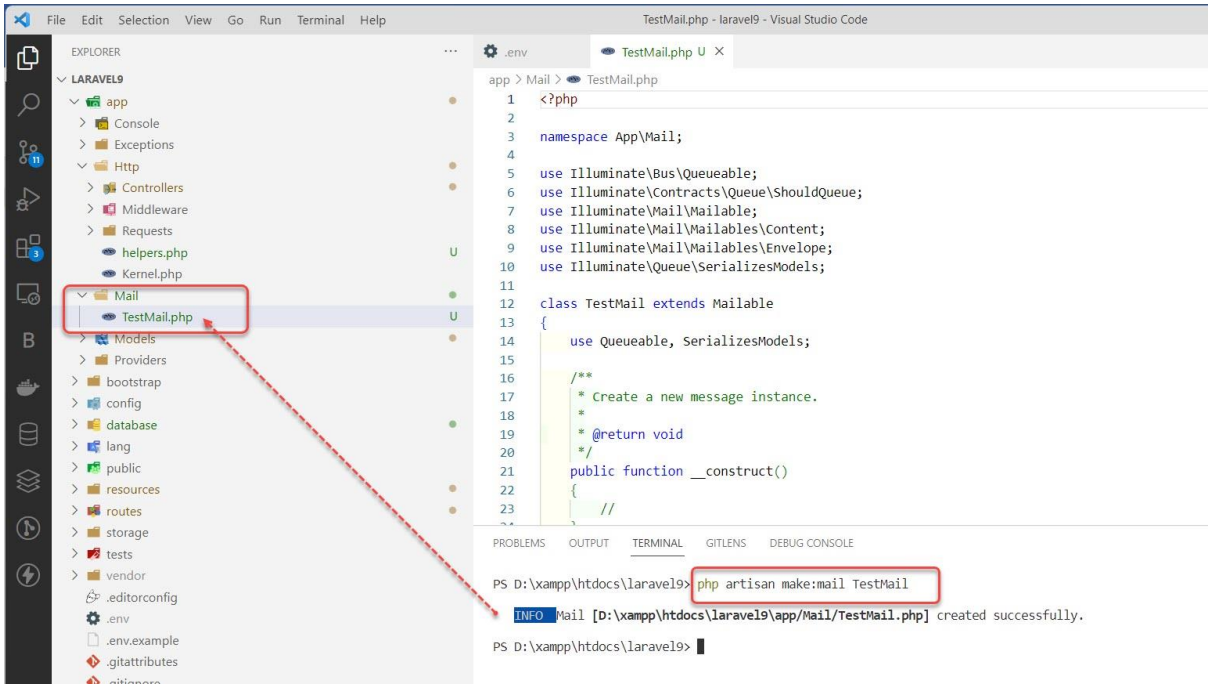


2.ตั้งค่า config ใน Laravel (.env) เปิดไฟล์ .env แล้วแก้ไข ดังนี้



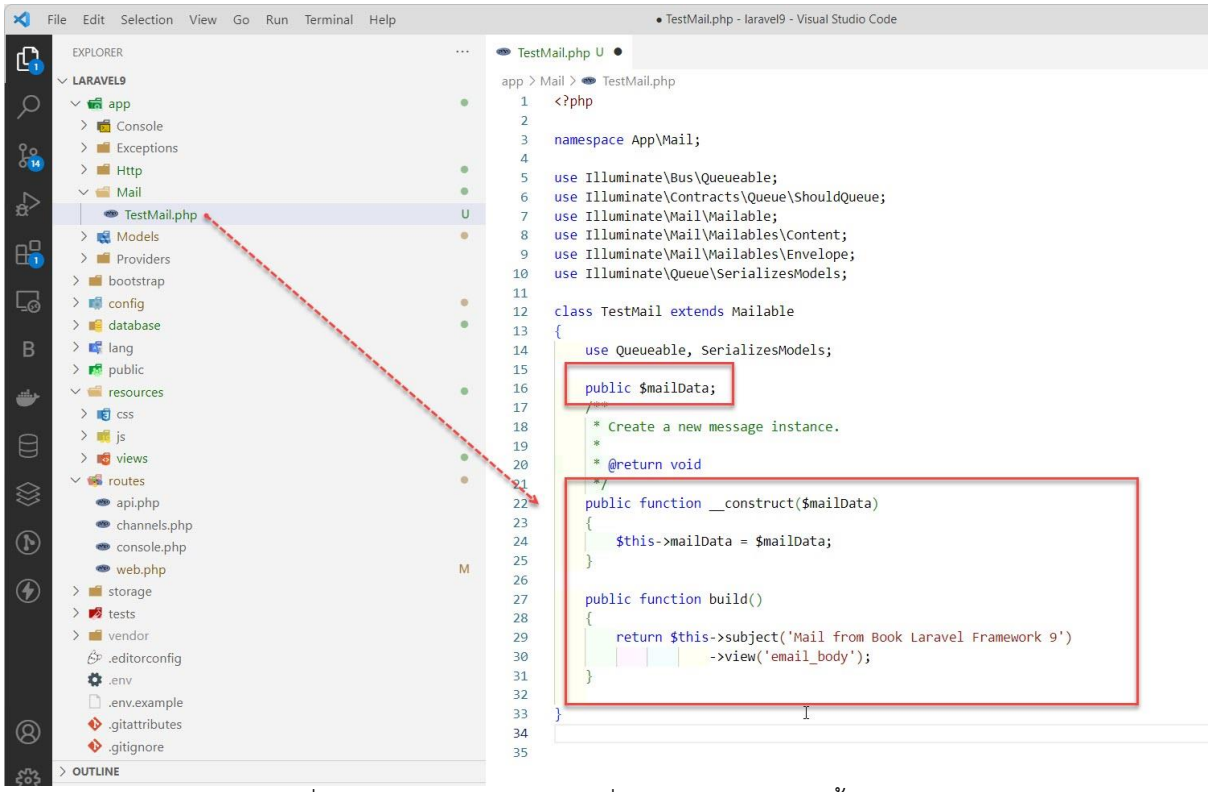
3.สร้างไฟล์สำหรับกำหนดการจัดส่งอีเมล โดยพิมพ์คำสั่งผ่าน Terminal ดังนี้

```
php artisan make:mail TestMail
```



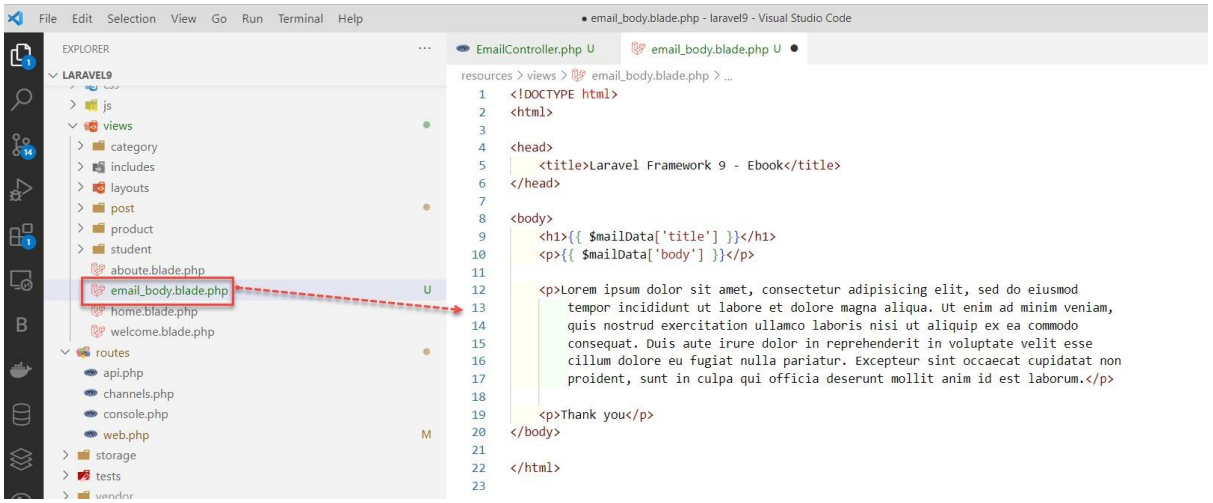
ไฟล์ TestMail.php จะถูกสร้างและเก็บไว้ใน app\Mail\...

4.เปิดไฟล์ TestMail.php แล้วเขียนโค้ดดังนี้



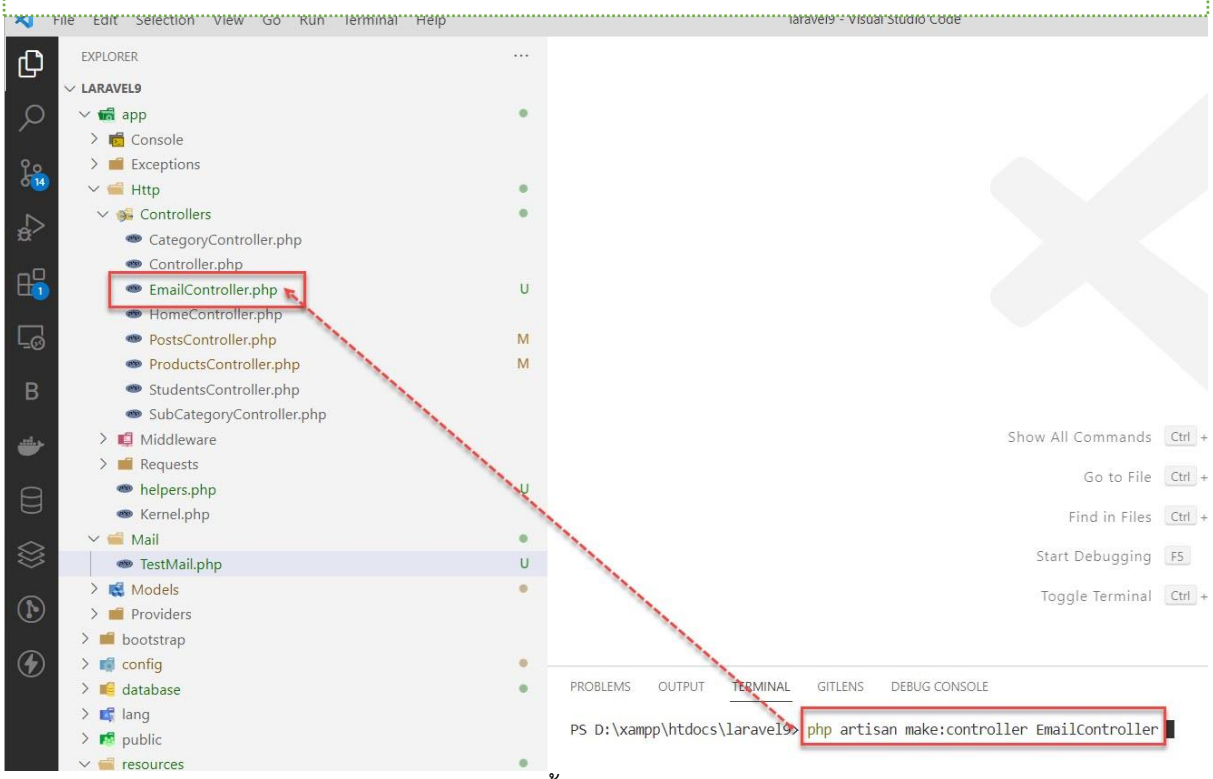
ในตัวอย่างบรรทัดที่ 16 มีการประกาศตัวแปรชื่อ $mailData จากนั้น เมธอด __contruct() จะมีการรับค่า Parameter 1 ค่า คือ $mailData และ ในเมธอด build() เป็นการกำหนดเนื้อหาที่จะส่งไป ในตัวอย่างจะเป็นการแสดงเนื้อหาจาก view ชื่อ email_body

5.สร้างไฟล์ email_body.blade.php ไว้ในโฟลเดอร์ resources\views\... แล้วเขียนโค้ดเข้าไปดังนี้



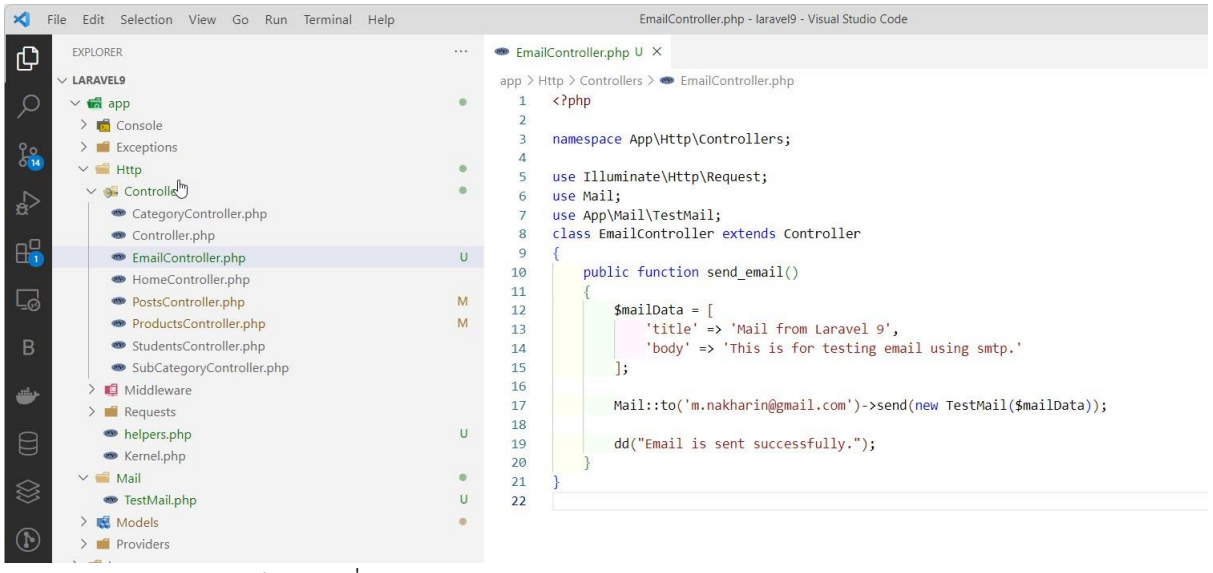
6.สร้างไฟล์ EmailController.php โดยพิมพ์คำสั่งผ่าน Terminal ดังนี้

```
php artisan make:controller EmailController
```



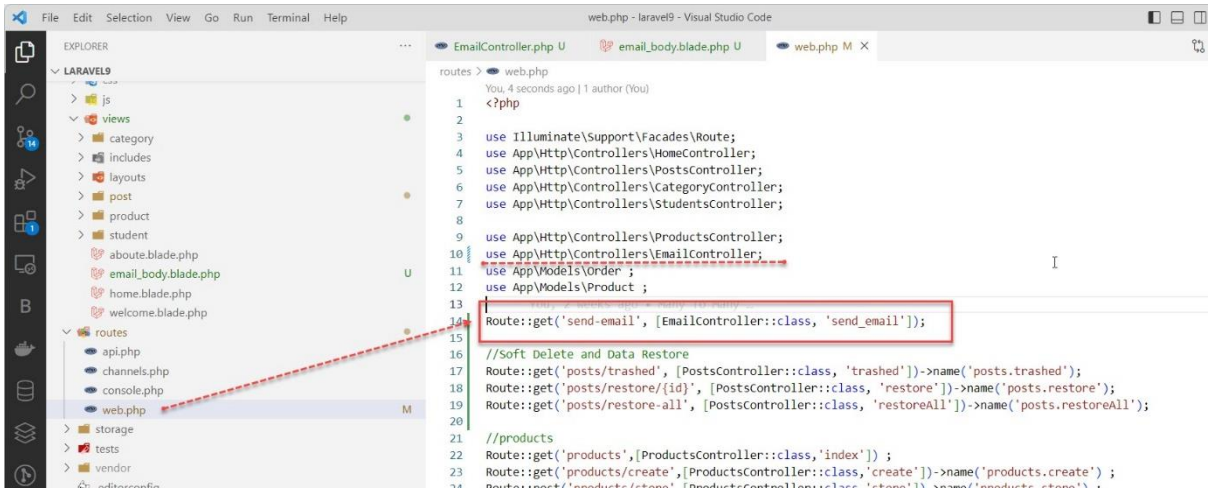
ไฟล์ EmailController.php จะถูกสร้างขึ้นและเก็บไว้ใน app\Http\Controllers\...

7.เปิดไฟล์ EmailController.php แล้วเขียนโค้ดดังนี้

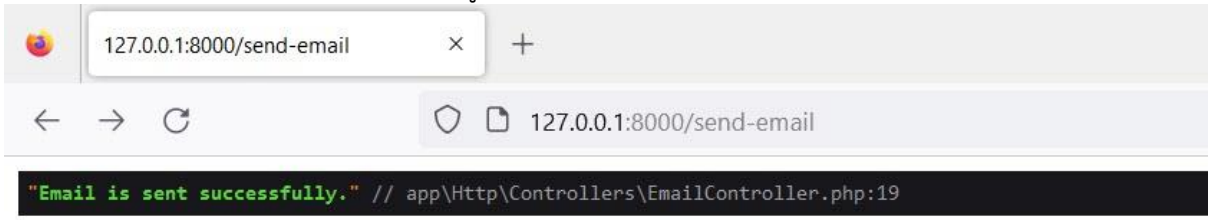


จากตัวอย่างเป็นการเพิ่มเมธอด send_email() แล้วกำหนดรายละเอียดการส่งอีเมล โดยจะส่งตัวแปรไปให้ส่วนแสดงผลทำงาน 2 ค่าคือ title และ body จากนั้นบรรทัดที่ 17 จะใช้คลาส Mail::to(ใส่อีเมลที่ต้องการส่ง) และใช้ send(....) สำหรับส่งอีเมลออกไป

8.เปิดไฟล์ web.php แล้วเพิ่มเส้นทางเข้าถึงหน้าเว็บ เขียนโค้ดเพิ่มดังนี้

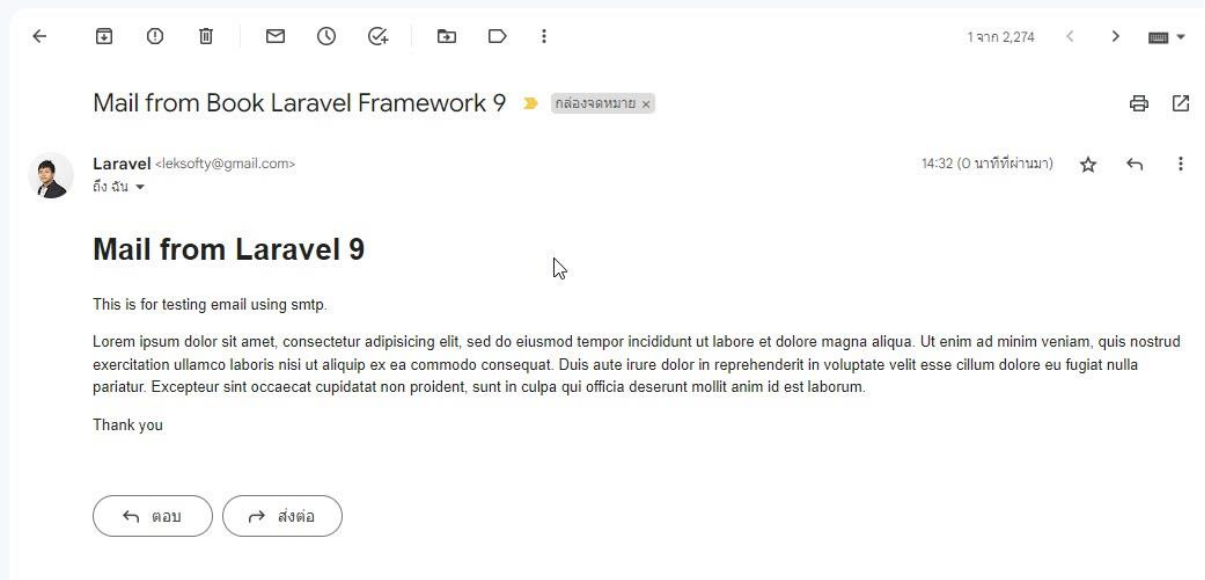


9.บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/send-email ที่ Browser



ถ้าไม่มีปัญหาใด ๆ เมื่อส่งอีเมลสำเร็จก็จะขึ้นข้อความปรากฏตามภาพด้านบนครับ

และเมื่อลองตรวจสอบที่กล่องข้อความก็จะมีอีเมล ถูกส่งมาเรียบร้อยแล้ว



การใช้งาน Session

Session เป็นตัวแปรที่มีไว้สำหรับเก็บข้อมูลอีกรูปแบบมีคุณสมบัติทุกอย่างเหมือนกับตัวแปรปกติ สามารถเก็บค่าข้อความหรือตัวเลขและตัวแปรที่เป็น Object ต่างๆได้ แต่จะมีความพิเศษคือค่าตัวแปรจะยังคงอยู่ไม่ว่าจะเปลี่ยนหน้าไปหน้าใดก็ตาม นิยมนำมาใช้ในการรับส่งค่าข้อมูลที่เก็บเป็นความลับที่จะต้องใช้ในการระบุตัวตนโดยใช้สิทธิ การยืนยันตัวตนด้วย Username และ Password เพื่อให้ได้ค่าตัวแปรนี้มาระบุตัวตน ทำให้มีความปลอดภัยในการเก็บรักษาข้อมูลได้อย่างมาก เราสามารถใช้งาน Session ส่งข้อมูลข้ามเพจแต่ละเพจได้ การใช้งาน Session สามารถใช้งานตัวแปร Session ได้หลายลักษณะ เช่น ใช้เก็บสถานะการเข้าระบบ เก็บข้อมูลตะกร้าสินค้า Session จะอยู่ในรูปแบบ Key / Value เวลาใช้งานจะเรียกใช้ผ่าน key ของข้อมูลในตำแหน่งที่ต้องการ รูปแบบคำสั่ง มีดังนี้

```
$request->session()->put('key','value')
```

เป็นคำสั่งสำหรับการใส่ค่าเข้าไปใน session โดยใช้เมธอด put()

```
$request->session()->get('key')
```

เป็นคำสั่งในการเข้าถึง key ในหน้าต่าง ๆ จะใช้เมธอด get()

```
if($request->session()->has('key')) {
….
}
```

เป็นคำสั่งสำหรับตรวจสอบว่ามี key session หรือไม่ จะใช้ร่วมกับ if ในการตรวจสอบเงื่อนไข

```
$request->session()->all()
```

เป็นคำสั่งสำหรับอ่านค่าตัวแปร session ทั้งหมด จะใช้ เมธอด all()

```
$request->session()->put('key','value')
```

เป็นคำสั่งสำหรับการแทนที่หรือใส่ค่าข้อมูลในตัวแปรแบบ session จะใช้เมธอด put()

```
$request->session()->push('key','value')
```

เป็นคำสั่งสำหรับเก็บตัวแปรแบบ array จะใช้เมธอด push()

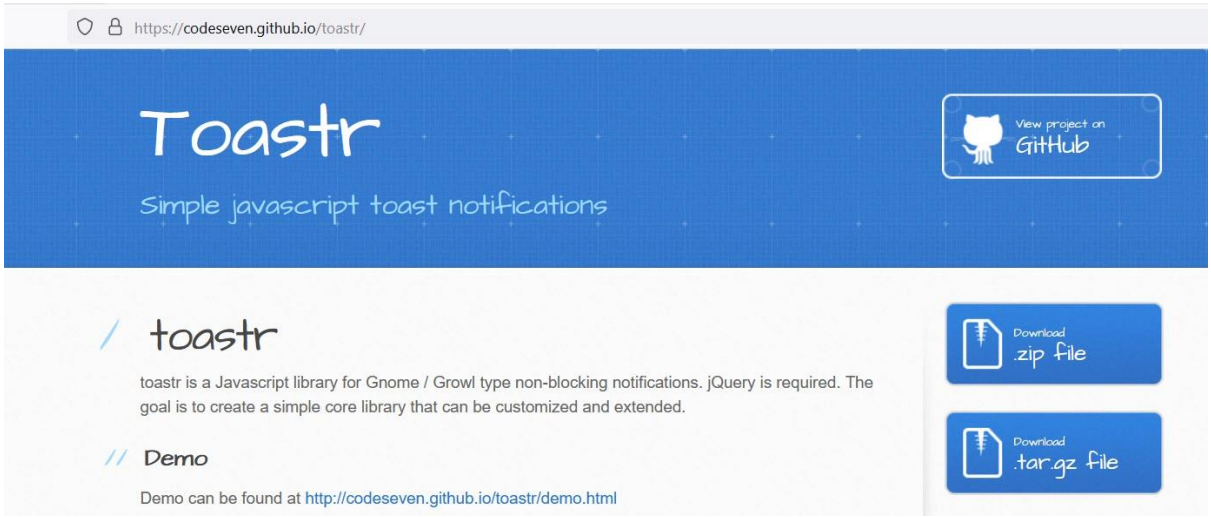
```
$request->session()->forget('key')
$request->session()->flush()
```

เป็นคำสั่งสำหรับลบ session จะใช้เมธอด forget() และ flush() ใช้ร่วมกัน

Flash Data

flash Data เป็นตัวแปร session อีกประเภทหนึ่งที่มีอายุการใช้งานชั่วคราว เมื่อมี request ใหม่ขึ้น request เก่าก็จะหายไป จะใช้สำหรับการโต้ตอบกับผู้ใช้ เช่น แจ้งเตือนการทำรายการต่าง ๆ ไม่ว่าจะเป็นการเพิ่มข้อมูล การแก้ไขข้อมูล การลบข้อมูล หรือแจ้งเตือนอื่น ๆ ที่ต้องการแจ้งเตือนให้ผู้ใช้ทราบ จากที่ผ่านมานั้นเราได้ทำการบันทึกข้อมูลไปแล้วจะเห็นได้ว่าพอบันทึกรายการไปแล้วจะไม่มีกล่องแจ้งเตือนโต้ตอบให้เราทราบเลย เพื่อให้เห็นภาพจะยกตัวอย่างการใช้งาน flash Data ร่วมกับ Toastr library และ Sweet Alert Library โดยจะแสดงตัวอย่างเมื่อมีการบันทึกข้อมูลเรียบร้อยแล้ว ให้มีข้อความแจ้งเตือนว่า **บันทึกข้อมูลเรียบร้อยแล้ว**

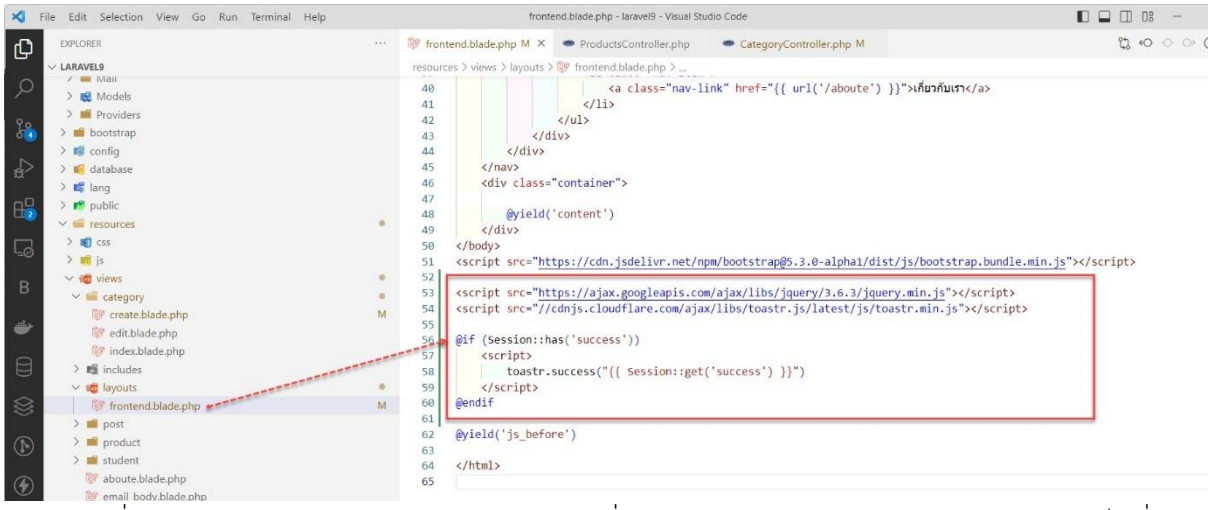
Toastr library เว็บไซต์หลัก https://codeseven.github.io/toastr/



1.เปิดไฟล์ frontend.blade.php ในโฟลเดอร์ resources\views\layouts\... แล้วเพิ่ม script css ในส่วนของ header และเพิ่ม script javascript ถัดจาก tag </body> ดังนี้

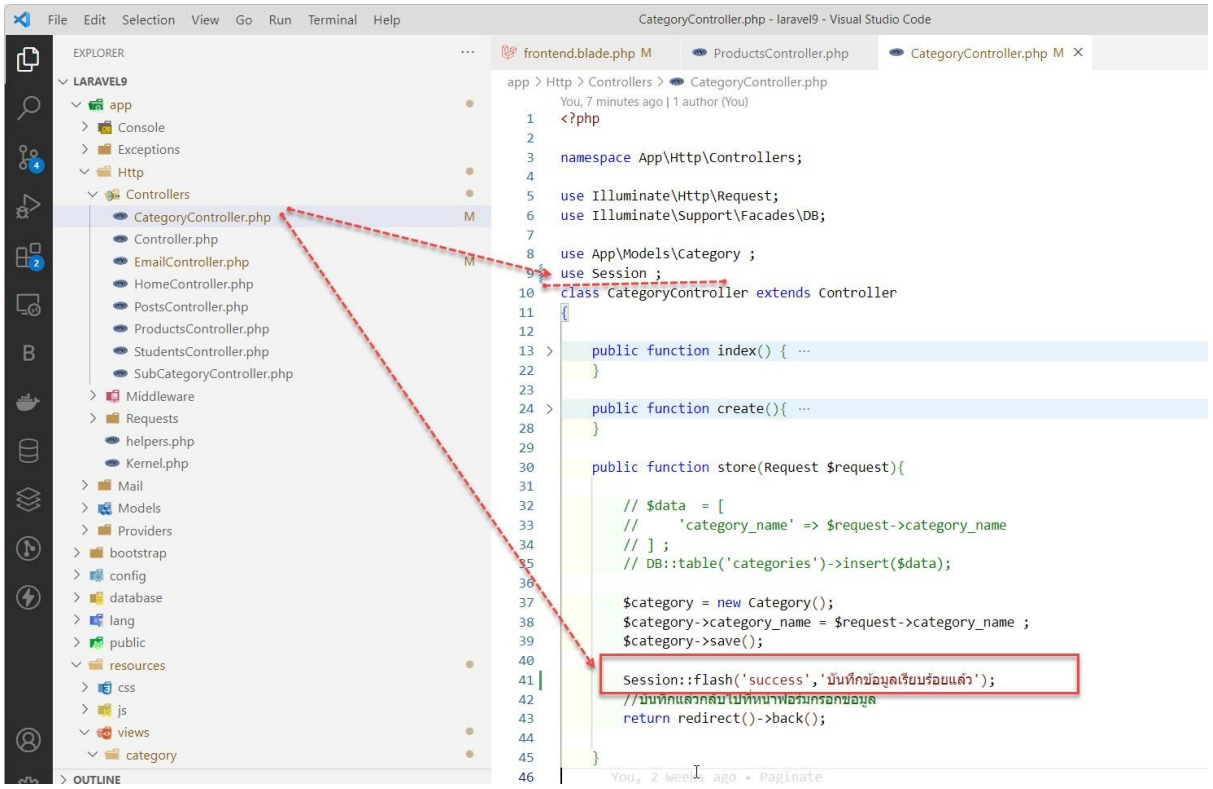


เพิ่ม toastr.min.css



เพิ่ม jquery.min.js และ toastr.min.js แล้วเพิ่มโค้ดตรวจสอบ session โดยกำหนด key เป็นชื่อว่า success และใช้ session::get(...) สำหรับเข้าถึง key

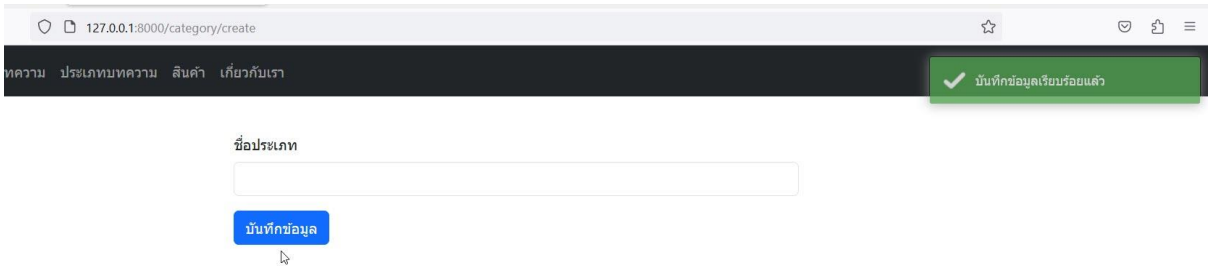
2.เปิดไฟล์ CategoryController.php ในโฟลเดอร์ app\Http\Controllers\... แล้วเพิ่มโค้ดเข้าไป ดังนี้



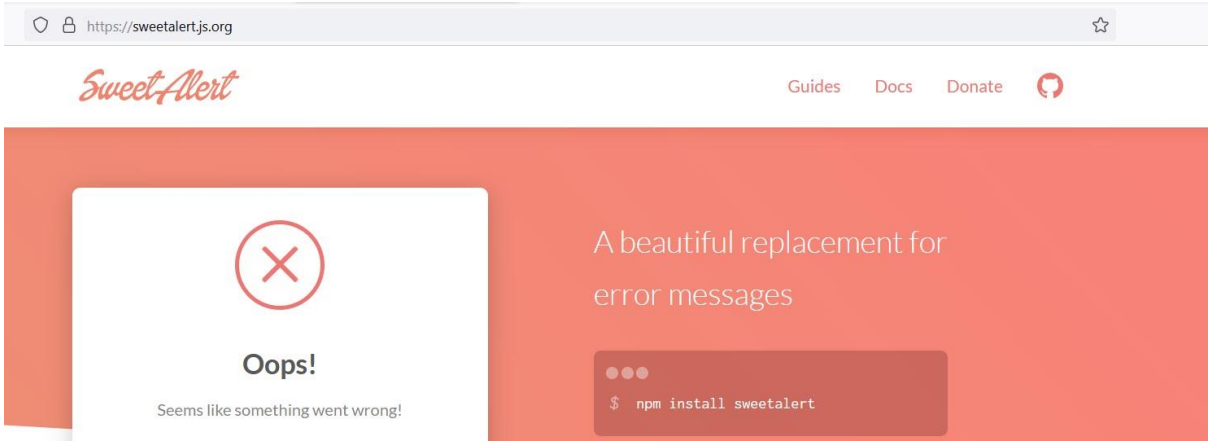
จากตัวอย่างโค้ดเป็นการเขียนคำสั่งสร้าง session สำหรับแจ้งเตือนการทำรายการ จะใช้ Session::flash(...) โดยตั้งชื่อ Key ว่า success และกำหนดค่าข้อความ Value ว่า บันทึกข้อมูลเรียบร้อยแล้ว

จุดสังเกต ก่อนที่เราจะใช้งาน Session::flash(...) นั้นให้ use Session ; เข้ามาด้วยนะครับ จากตัวอย่างจะกำหนดไว้ที่บรรทัดที่ 9

3.บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/category ที่ Browser แล้วทดลองเพิ่มประเภทบทความเข้าไป เมื่อกดปุ่มบันทึกข้อมูลแล้ว เมื่อบันทึกข้อมูลสำเร็จก็จะมีข้อความแจ้งเตือน ว่าบันทึกข้อมูลเรียบร้อยแล้ว ดังภาพ



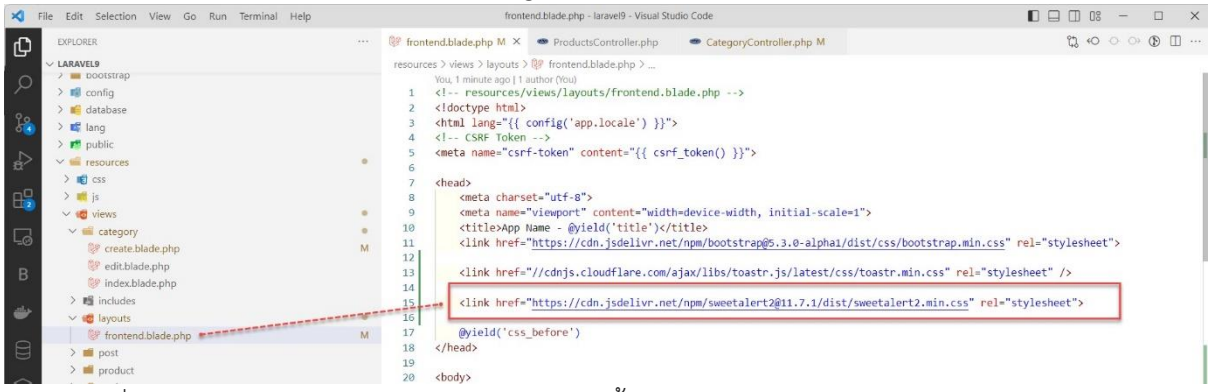
Sweet Alert Library เว็บไซต์หลัก https://sweetalert.js.org/



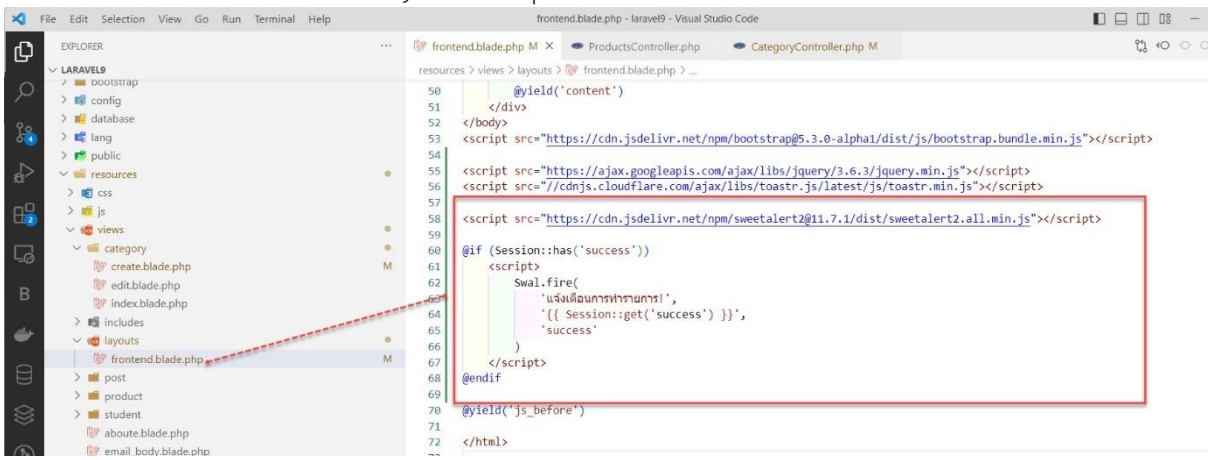
Sweet Alert Library เป็นอีก library ที่นิยมใช้กันมาก เนื่องจากว่ากล่องข้อความแจ้งเตือนมีรูปแบบที่สวยงาม มี option ให้ใช้งานเยอะ การใช้งานให้เราไปแก้ไขเพิ่มเติมตามนี้

เปิดไฟล์ frontend.blade.php ในโฟลเดอร์ resources\views\layouts\... แล้วเพิ่ม script css ในส่วนของ header และเพิ่ม script javascript ถัดจาก tag </body> ดังนี้

เพิ่ม sweetalert2.min.css ภายใต้ tag <head></head>



เพิ่ม sweetalert2.all.min.js และ script ดังนี้

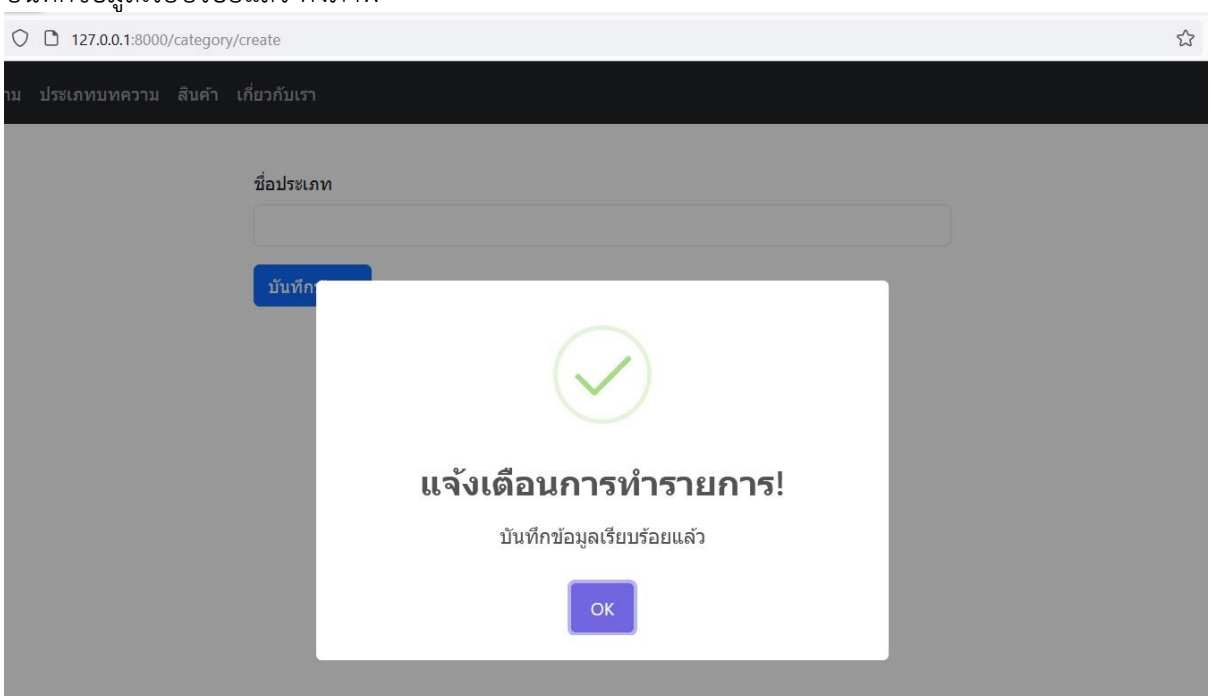


หากขี้เกียจพิมพ์ copy & paste

https://github.com/leksoft/laravel9/blob/main/resources/views/layouts/frontend.blade.php

บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/category ที่ Browser แล้วทดลองเพิ่มประเภทบทความเข้าไป เมื่อกดปุ่มบันทึกข้อมูลแล้ว เมื่อบันทึกข้อมูลสำเร็จก็จะมีข้อความแจ้งเตือน ว่า

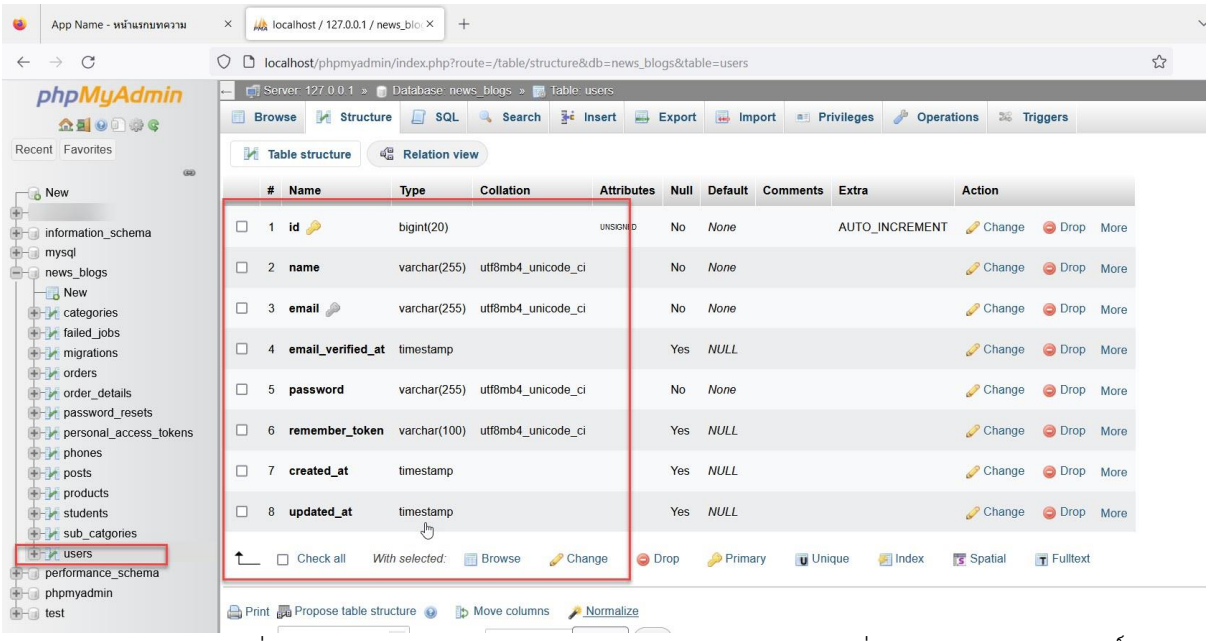
บันทึกข้อมูลเรียบร้อยแล้ว ดังภาพ



ระบบล็อกอินและระบบลงทะเบียน (Auth Login and Registration)

บทเรียนที่ผ่านมานั้น จะเห็นได้ว่าเวลาที่เราทำการเพิ่มข้อมูล แก้ไขข้อมูล หรือลบข้อมูล จะสามารถทำได้ทันทีโดยไม่มีการล็อกอินและไม่มีการตรวจสอบสิทธิ์การเข้าจัดการข้อมูล ปกติแล้วจะไม่ให้ผู้ใช้ทั่วไป เข้าไปเพิ่มข้อมูล หรือ แก้ไขข้อมูลได้ นอกจากว่าจะมีสิทธิ์ในการเข้าไปจัดการเท่านั้น บทเรียนนี้เราจะได้เรียนรู้การสร้างระบบล็อกอินและการจัดการสิทธิ์ (Middleware) ในการจัดการข้อมูลต่าง ๆ การสร้างระบบล็อกอินใน Laravel จะมีเครื่องมือมาให้เราเพียบพร้อมทุกอย่าง ทำให้เราสามารถสร้างระบบล็อกอินได้อย่างง่ายมาก โดยไม่ต้องเขียนโค้ดเองเลย แต่หากว่าอยาก Custom เองก็สามารถทำได้เช่นกัน ในตัวอย่างนี้จะยกตัวอย่างการสร้างระบบล็อกอินพร้อมกับระบบลงทะเบียนอย่างง่าย และจัดการสิทธิ์การเข้าถึงเมนูต่าง ๆ เช่น บทความ ประเภทบทความ สินค้า หากผู้ใช้งานไม่ได้ล็อกอินเข้าระบบ (guest) ก็จะสามารถดูข้อมูลได้อย่างเดียวจะไม่สามารถเพิ่มแก้ไขลบได้

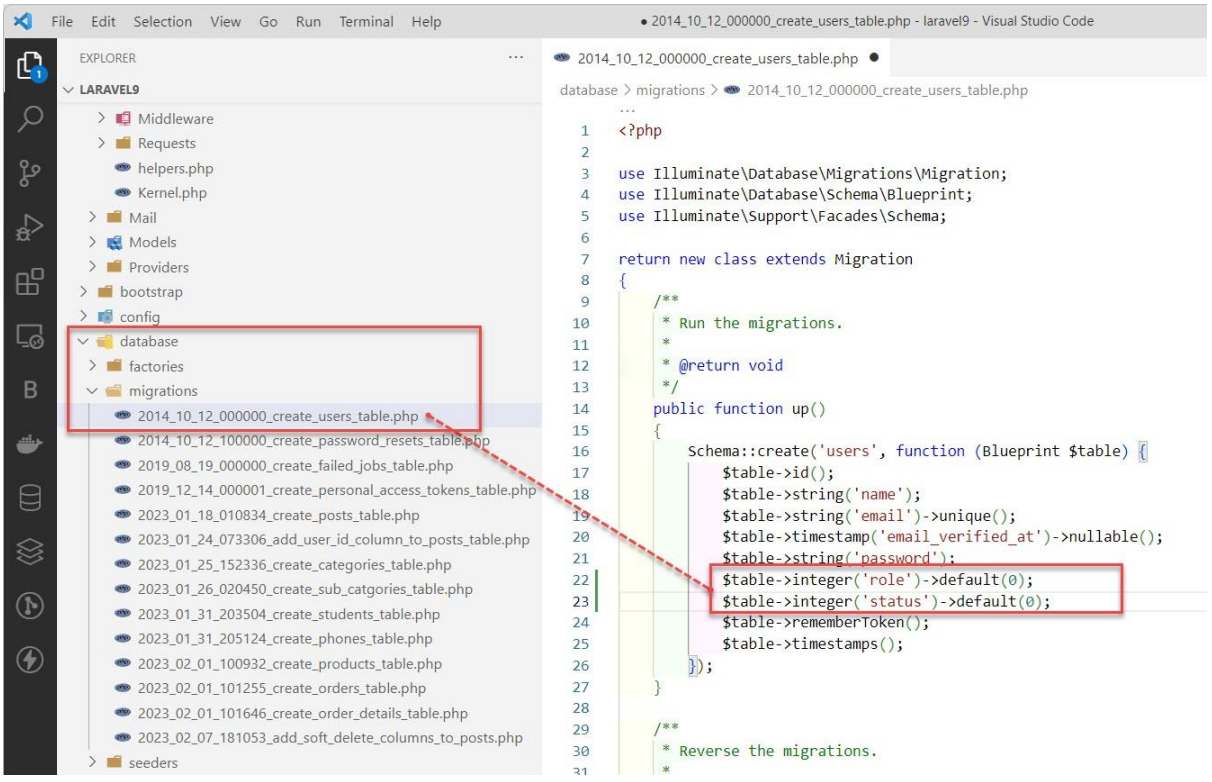
การสร้างระบบล็อกอินนั้น ในหนังสือนี้จะใช้ตาราง users ซึ่งจากที่ผ่านมาถ้าสังเกตในฐานข้อมูลเมื่อมีการ migrate จะมีตาราง users มาให้เราแล้ว ให้เปิดโปรแกรม phpMyAdmin ขึ้นมาดูตามภาพ จะเห็นว่ามีตาราง users ถูกสร้างให้เราแล้ว



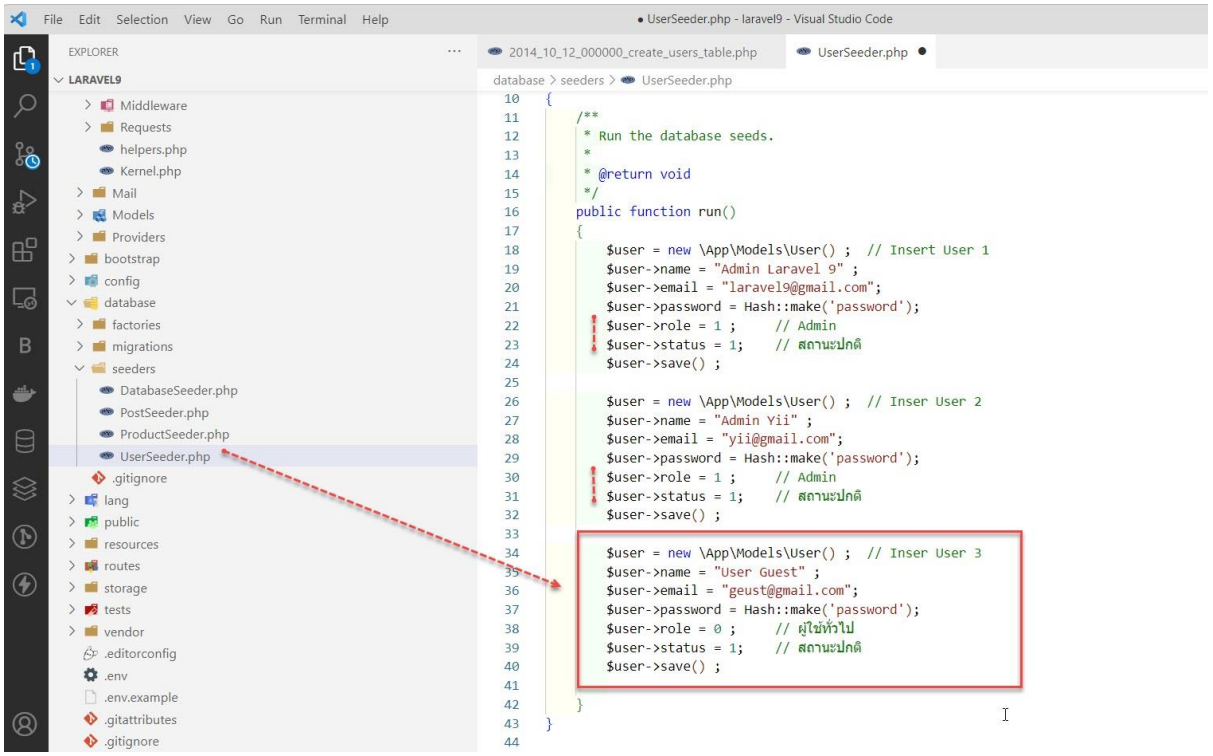
ให้เราทำการเพิ่มโครงสร้างตาราง users เข้าไปอีก 2 ฟิวด์ คือ role เพื่อใช้สำหรับกำหนดสิทธิ์การจัดการข้อมูล และ status เพื่อใช้สำหรับบอกสถานะการทำงาน ตามขั้นตอน ดังนี้

*** การสร้างระบบล็อกอินนั้นมีการเชื่อมต่อกับฐานข้อมูล เพราะฉะนั้นควรตรวจสอบการเชื่อมต่อกับฐานข้อมูลที่ไฟล์ .env ของท่านเองให้ถูกต้อง ดูการตั้งค่าจากหน้า 39

1.เปิดไฟล์ xxx_ create_users_table.php ในโฟลเดอร์ database\migrations\... แล้วเพิ่มโครงสร้างตารางใหม่ ดังนี้



2.เปิดไฟล์ UserSeeder.php ในโฟลเดอร์ database\seeders\... แล้วเพิ่มข้อมูลตั้งต้นเพิ่มเติมดังนี้



จากตัวอย่างเป็นการเพิ่มผู้ใช้งานอีก 1 user คือ บรรทัดที่ 34 – 40 โดยกำหนด role เท่ากับ 0 และกำหนด status เท่ากับ 1 และบรรทัดที่ 22 – 23 และบรรทัด 30 -31 กำหนด role เท่ากับ 0 และกำหนด status เท่ากับ 1 (role = 0 คือผู้ใช้งานทั่วไป , 1 คือ admin ,status = 1 คือ สถานะปกติ , 0 คือ ถูกแบน)

เมื่อเพิ่มเสร็จแล้วบันทึกไฟล์และให้รันคำสั่ง migrate:refresh และ db:seed ใหม่ ผ่าน Terminal

```
php artisan migrate:refresh
php artisan db:seed
```

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE

PS D:\xampp\htdocs\laravel9> php artisan migrate:refresh

  INFO  Rolling back migrations.

  2023_02_07_181053_add_soft_delete_columns_to_posts ..... 93ms DONE
  2023_02_01_101646_create_order_details_table ..... 33ms DONE
  2023_02_01_101255_create_orders_table ..... 11ms DONE
  2023_02_01_100932_create_products_table ..... 9ms DONE
  2014_10_12_000000_create_users_table ..... 52ms DONE
  2014_10_12_100000_create_password_resets_table ..... 59ms DONE
  2019_08_19_000000_create_failed_jobs_table ..... 36ms DONE
  2019_12_14_000001_create_personal_access_tokens_table ..... 48ms DONE
  2023_01_18_010834_create_posts_table ..... 26ms DONE
  2023_01_24_073306_add_user_id_column_to_posts_table ..... 11ms DONE
  2023_01_25_152336_create_categories_table ..... 24ms DONE
  2023_01_26_020450_create_sub_catgories_table ..... 36ms DONE
  2023_01_31_203504_create_students_table ..... 27ms DONE
  2023_01_31_205124_create_phones_table ..... 24ms DONE
  2023_02_01_100932_create_products_table ..... 22ms DONE
  2023_02_01_101255_create_orders_table ..... 20ms DONE
  2023_02_01_101646_create_order_details_table ..... 39ms DONE
  2023_02_07_181053_add_soft_delete_columns_to_posts ..... 12ms DONE

PS D:\xampp\htdocs\laravel9> php artisan db:seed

  INFO  Seeding database.

  Database\Seeders\UserSeeder ..... RUNNING
  Database\Seeders\UserSeeder ..... 656.18 ms DONE

  Database\Seeders\PostSeeder ..... RUNNING
  Database\Seeders\PostSeeder ..... 1,219.72 ms DONE

  Database\Seeders\ProductSeeder ..... RUNNING
  Database\Seeders\ProductSeeder ..... 235.27 ms DONE
```

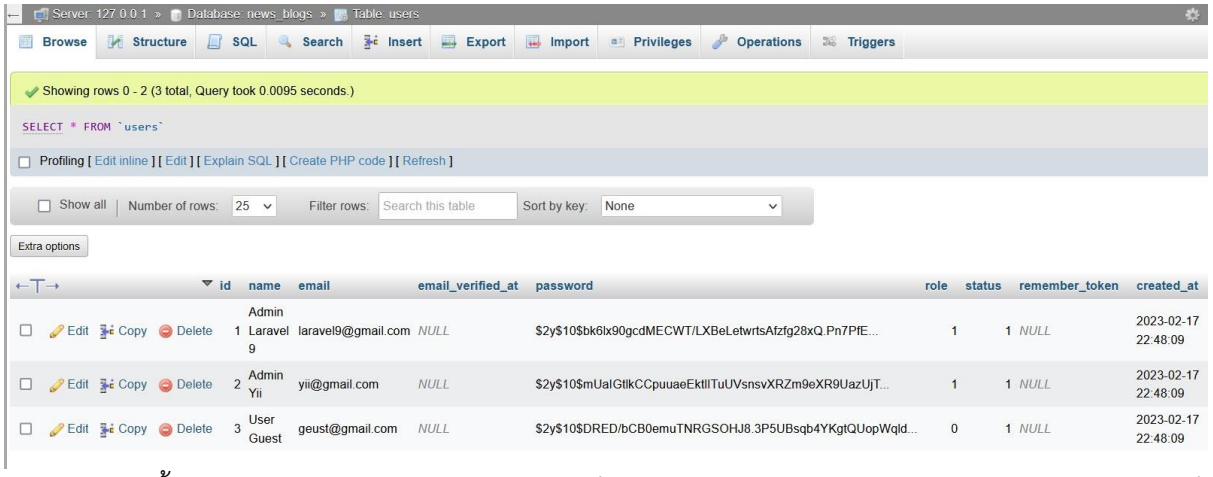
เปิดโปรแกรม phpMyAdmin ดูอีกครั้ง จะเห็นว่ามีฟิวด์ role และ status ถูกสร้างให้เราแล้ว

Server: 127.0.0.1 » Database: news_blogs » Table: users

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Triggers

Table structure | Relation view

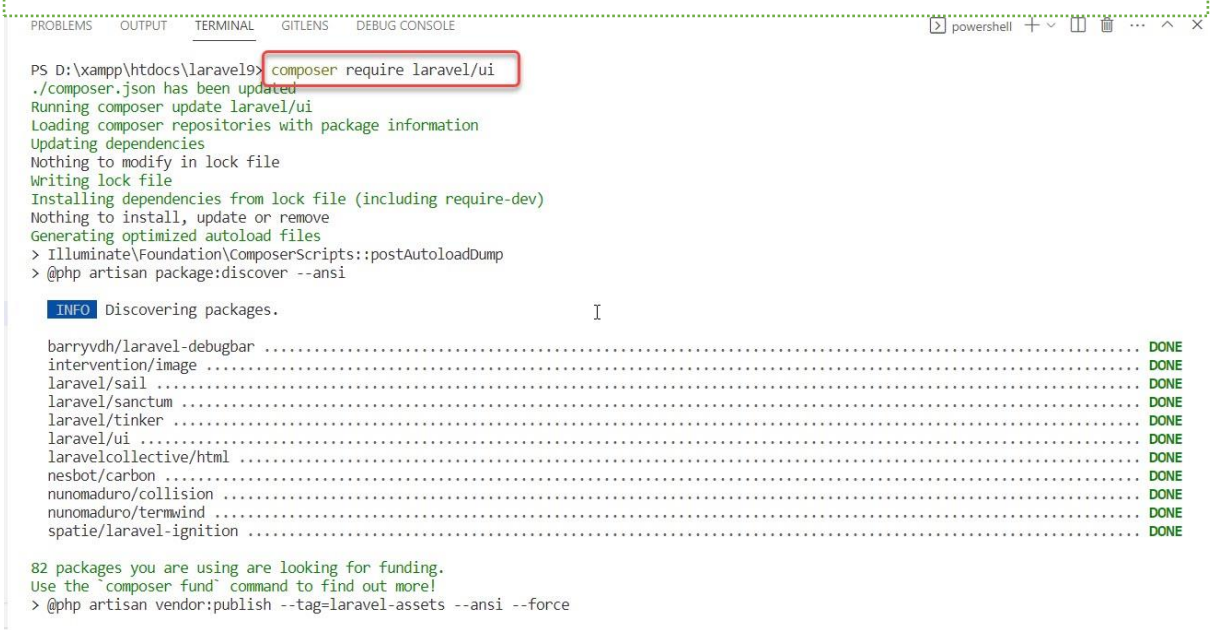
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	name	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
3	email	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
4	email_verified_at	timestamp			Yes	NULL			Change Drop More
5	password	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
6	role	int(11)			No	0			Change Drop More
7	status	int(11)			No	0			Change Drop More
8	remember_token	varchar(100)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
9	created_at	timestamp			Yes	NULL			Change Drop More
10	updated_at	timestamp			Yes	NULL			Change Drop More



ตอนนี้เรามี email และ password สำหรับล็อกอินเข้าระบบเรียบร้อยแล้ว โดยในภาพตัวอย่างจะเห็นว่ามีการกำหนดสิทธิ์ นั่นคือฟิวด์ role = 0 คือผู้ใช้งานทั่วไป , 1 คือ admin , status = 1 คือ สถานะปกติ 0 คือถูกแบน

3. ติดตั้ง package laravel/ui พิมพ์คำสั่งผ่าน Terminal ดังนี้

```
composer require laravel/ui
```



เมื่อใช้คำสั่งด้านบน Laravel จะทำการโหลด package ต่าง ๆ ที่เกี่ยวกับการสร้างระบบล็อกอิน และระบบลงทะเบียน รวมถึงระบบลืมรหัสผ่านมาให้เรา เมื่อติดตั้ง laravel/ui เรียบร้อยแล้วเราสามารถเลือกได้ว่าจะสร้างระบบล็อกอินด้วย UI อะไร ซึ่งใน Laravel จะมีให้เราเลือกใช้งานอยู่ 3 รูปแบบ คือ bootstrap , vue , react

ซึ่งจะมีคำสั่งให้เราเลือกติดตั้งตามนี้

```
// Generate basic scaffolding...
php artisan ui bootstrap
php artisan ui vue
php artisan ui react
// Generate login / registration scaffolding...
php artisan ui bootstrap --auth
php artisan ui vue --auth
php artisan ui react --auth
```

trap

4.สร้างหน้าเข้าระบบและหน้าลงทะเบียน พิมพ์คำสั่งผ่าน Terminal ดังนี้

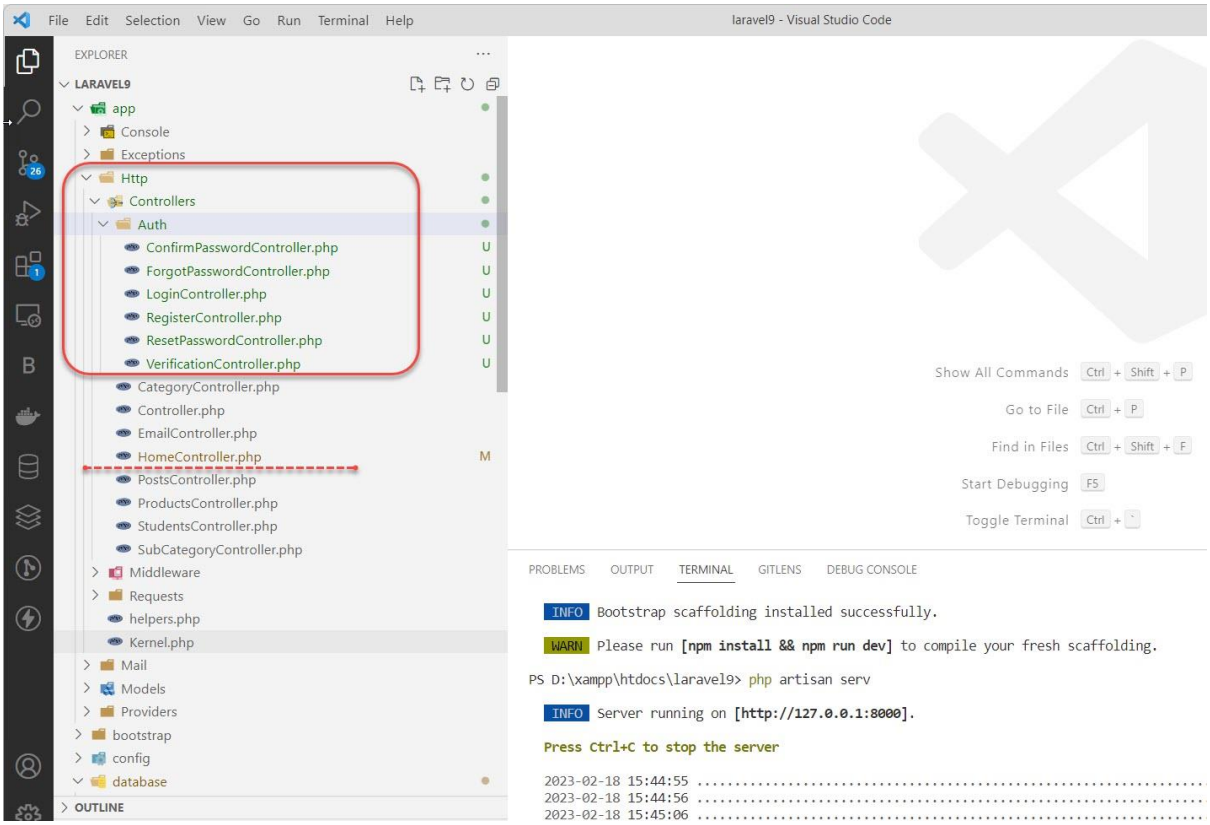
```
php artisan ui bootstrap --auth
```



--auth กำหนดให้สร้างระบบล็อกอิน ระบบลงทะเบียนและระบบลืมรหัสผ่าน อัตโนมัติ
เมื่อพิมพ์คำสั่งด้านบนแล้วกด Enter Laravel จะสร้างไฟล์ที่สำคัญ ๆ มาให้เรา ดังนี้

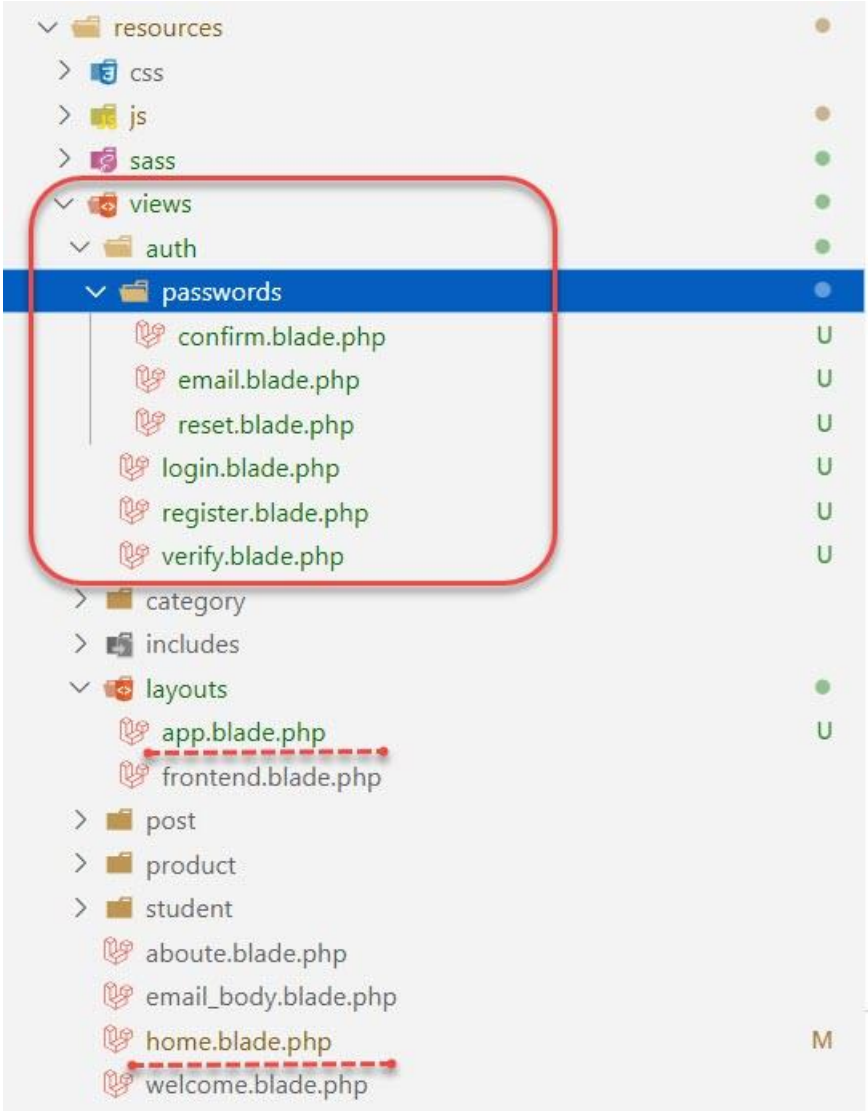
ส่วนของ Controllers ที่ถูกสร้างเพิ่มมาให้

1.ไฟล์ app\Http\Auth\ConfirmPasswordController.php
2.ไฟล์ app\Http\Auth\ForgotPasswordController.php
3.ไฟล์ app\Http\Auth\LoginController.php
4.ไฟล์ app\Http\Auth\RegisterController.php
5.ไฟล์ app\Http\Auth\VerificationController.php
6. ไฟล์ HomeController.php



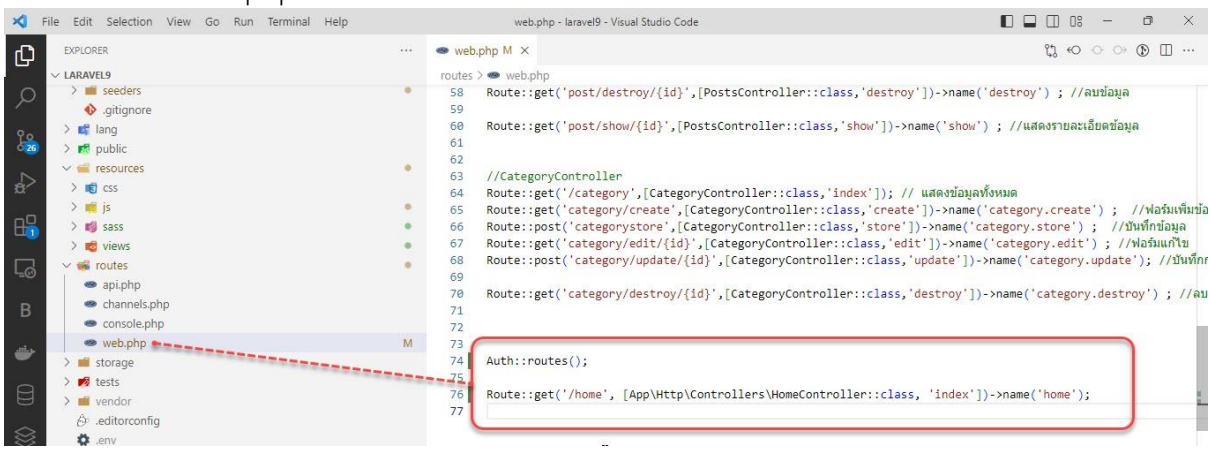
ส่วนของ Views ที่ถูกสร้างเพิ่มมาให้

1.ไฟล์ resources\views\auth\login.blade.php
2.ไฟล์ resources\views\auth\register.blade.php
3.ไฟล์ resources\views\auth\verify.blade.php
4.ไฟล์ resources\views\auth\password\confirm.blade.php
5.ไฟล์ resources\views\auth\password\email.blade.php
6.ไฟล์ resources\views\auth\password\reset.blade.php
7.ไฟล์ resources\views\layouts\app.blade.php
8.ไฟล์ resources\views\ layouts\home.blade.php



ส่วนของ Routes ที่ถูกสร้างเพิ่มมาให้

ในไฟล์ web.php

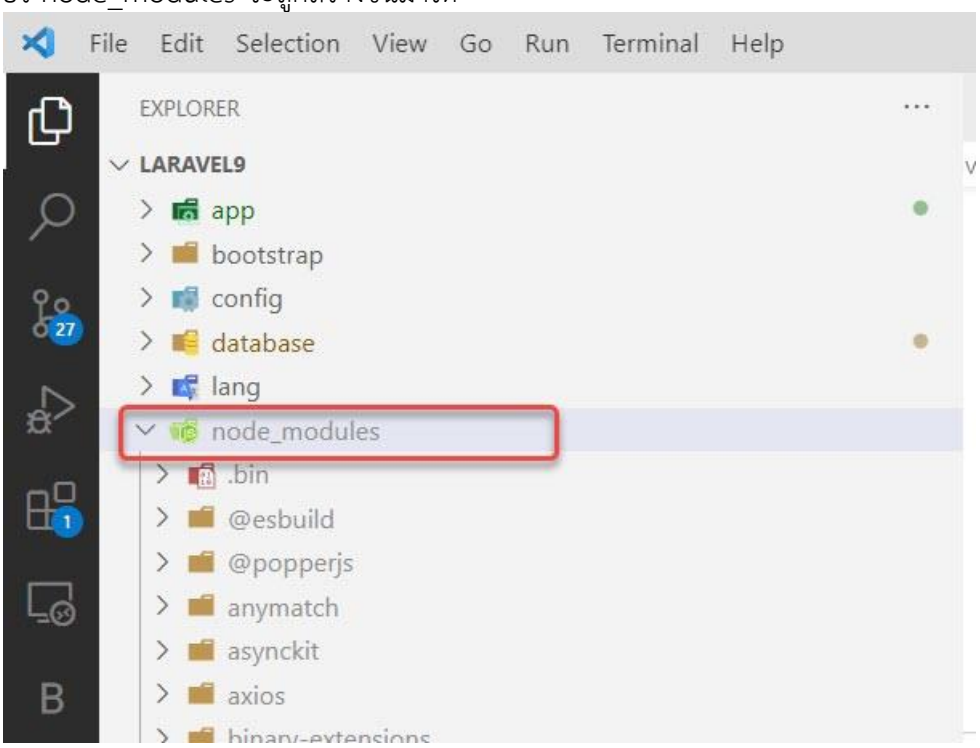


5. ติดตั้ง install npm dependencies โดย run คำสั่ง npm install ผ่าน Terminal ดังนี้

```
npm install หรือ  npm i
```

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE                powershell

PS D:\xampp\htdocs\laravel9> npm i

up to date, audited 45 packages in 887ms

9 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\xampp\htdocs\laravel9>
```

โฟลเดอร์ node_modules จะถูกสร้างขึ้นมาให้



6. รันคำสั่ง npm run build ผ่าน Terminal เพื่อ compile CSS ดังนี้

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE

  2023-02-18 22:54:12 ..... ~ 1s
  2023-02-18 22:54:13 ..... ~ 0s
  2023-02-18 22:54:13 ..... ~ 0s
  2023-02-18 22:54:13 ..... ~ 5s
PS D:\xampp\htdocs\laravel9> npm run build

> build
> vite build

vite v4.1.2 building for production...
✓ 106 modules transformed.
public/build/manifest.json              0.26 kB
public/build/assets/app-67dcdfd2.css  192.28 kB | gzip: 27.26 kB
public/build/assets/app-20b9e4fb.js   110.64 kB | gzip: 35.84 kB
PS D:\xampp\htdocs\laravel9>
```

7.แก้ไขส่วนแสดงผลและเพิ่มลิงค์สำหรับเข้าระบบและลงทะเบียน

เปิดไฟล์ frontend.blade.php ในโฟลเดอร์ resources\views\layouts\... แล้วแก้ไขโค้ดใหม่ ดังนี้

```
<!-- resources/views/layouts/frontend.blade.php -->
<!doctype html>
<html lang="{{ config('app.locale') }}">
<!-- CSRF Token -->
<meta name="csrf-token" content="{{ csrf_token() }}">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>App Name - @yield('title')</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
  <link href="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css" rel="stylesheet" />
  <link href="https://cdn.jsdelivr.net/npm/sweetalert2@11.7.1/dist/sweetalert2.min.css" rel="stylesheet">
  @yield('css_before')
</head>
<body>
  <div class="b-example-divider"></div>
  <nav class="py-2 bg-light border-bottom">
    <div class="container d-flex flex-wrap">
      <ul class="nav me-auto">
        <li class="nav-item"><a href="{{ url('/') }}" class="nav-link link-dark px-2 active"
            aria-current="page">หน้าหลัก</a></li>
        <li class="nav-item"><a href="{{ url('/posts') }}" class="nav-link link-dark px-2">บทความ</a></li>
        <li class="nav-item"><a href="{{ url('/category') }}" class="nav-link link-dark px-2">ประเภทบทความ</a></li>
        <li class="nav-item"><a href="{{ url('/products') }}" class="nav-link link-dark px-2">สินค้า</a></li>
        <li class="nav-item"><a href="{{ url('/aboute') }}" class="nav-link link-dark px-2">เกี่ยวกับเรา</a></li>
      </ul>
      <ul class="nav">
        @guest
          @if (Route::has('login'))
            <li class="nav-item"><a href="{{ route('login') }}" class="nav-link link-dark px-2">เข้าระบบ</a></li>
          @endif
          @if (Route::has('register'))
            <li class="nav-item"><a href="{{ route('register') }}" class="nav-link link-dark px-2">ลงทะเบียน</a></li>
          @endif
        @else
          <li class="nav-item dropdown">
            <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#" role="button"
              data-bs-toggle="dropdown" aria-haspopup="true" aria-expanded="false" v-pre>
              {{ Auth::user()->name }}
            </a>
```

```
                <div class="dropdown-menu dropdown-menu-end" aria-labelledby="navbarDropdown">
                  <a class="dropdown-item" href="{{ route('logout') }}"
                    onclick="event.preventDefault(); document.getElementById('logout-form').submit();">
                      ออกจากระบบ
                  </a>
                  <form id="logout-form" action="{{ route('logout') }}" method="POST" class="d-none">
                    @csrf
                  </form>
                </div>
              </li>
            @endguest
          </ul>
        </div>
      </nav>
      <div class="container">
        @yield('content')
      </div>
    </body>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0alpha1/dist/js/bootstrap.bundle.min.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>
    <script src="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11.7.1/dist/sweetalert2.all.min.js"></script>
            @if (Session::has('success'))
              <script>
                Swal.fire(
                  'แจ้งเตือนการทำรายการ!',
                  '{{ Session::get('success') }}',
                  'success'
                )
              </script>
            @endif
    @yield('js_before')
    </html>
```

8.เปิดไฟล์ login.blade.php ในโฟลเดอร์ resources\views\auth\... แล้วแก้ไขโค้ดใหม่ ดังนี้

```
@extends('layouts.app)
```

แก้ไขเป็น

```
@extends('layouts.frontend')
```

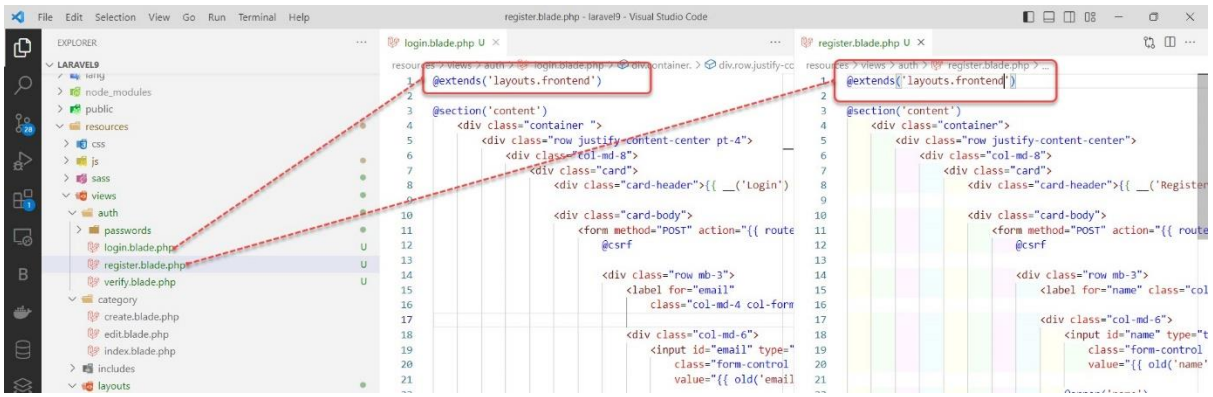
เนื่องจากว่าเรามี template ที่เราได้สร้างขึ้นสำหรับการใช้งานแล้ว ฉะนั้นให้เราแก้ไขในส่วนของ @extends() ให้เป็น template ที่เราสร้าง โดยแก้จาก layouts.app เป็น layouts.frontend

9.เปิดไฟล์ register.blade.php ในโฟลเดอร์ resources\views\auth\... แล้วแก้ไขโค้ดใหม่ดังนี้

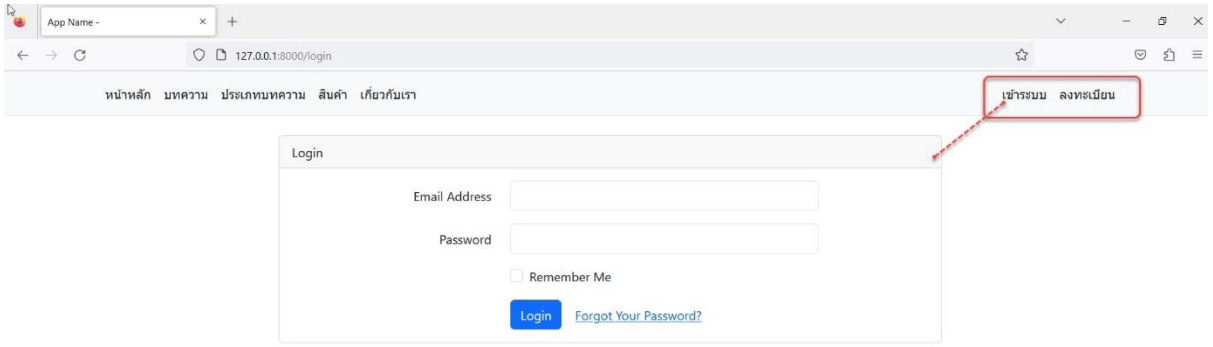
```
@extends('layouts.app)
```

แก้ไขเป็น

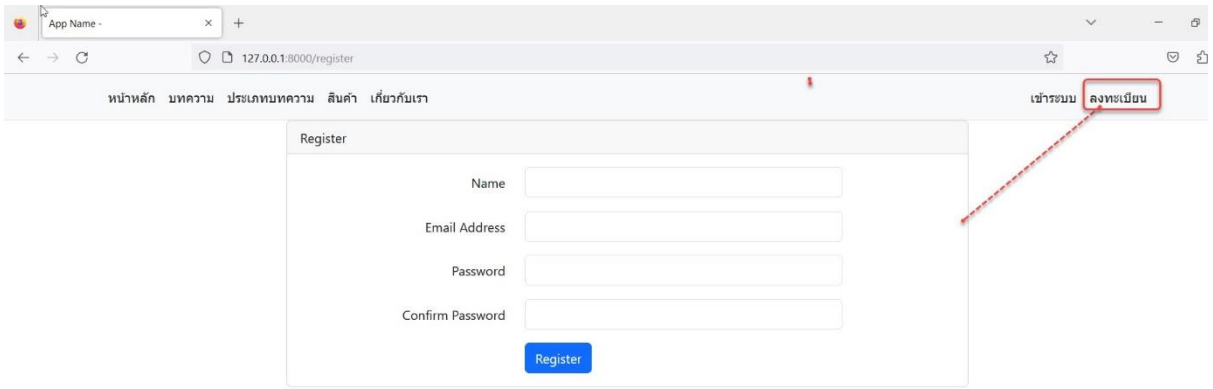
```
@extends('layouts.frontend')
```



บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000 ที่ Browser

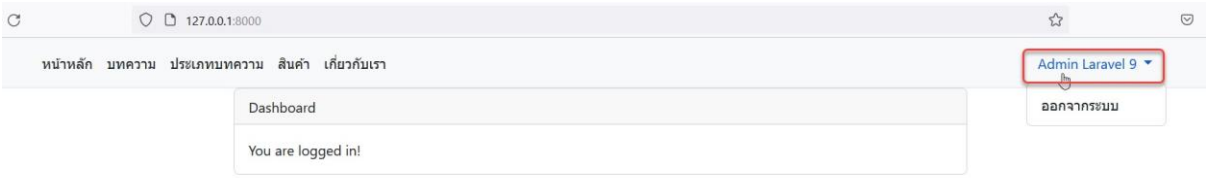


จะเห็นว่ามีเมนู เข้าระบบ และ ลงทะเบียน เพิ่มมาให้เราแล้ว เมื่อเราคลิกที่เมนู เข้าระบบ ก็จะแสดงหน้าสำหรับให้เรา Login เข้าระบบ และเมื่อคลิกที่เมนู ลงทะเบียน ก็จะแสดงหน้าสำหรับให้ลงทะเบียนใช้งานใหม่



ให้เราทดสอบคลิกที่เมนูเข้าระบบ แล้วกรอก Email และ Password แล้วคลิก เมนู Login ดู Email และ Password ที่เราได้กำหนดไว้ในหน้า 236 หรือไม่ก็สามารถคลิกที่เมนูลงทะเบียนใหม่เลยก็ได้

ครับ เมื่อ Login สำเร็จก็จะขึ้นหน้าสำหรับจัดการข้อมูล พร้อมแสดงชื่อผู้เข้าใช้ระบบ ตามภาพ



เพียงเท่านี้เราก็สามารถสร้างระบบล็อกอินและระบบลงเบียนสำหรับไว้ใช้งานในระบบของเราเรียบร้อยแล้ว โดยที่เราไม่ต้องเขียนโค้ดให้เสียเวลาเลย แต่ในบางงานเราอาจจะมีระบบที่ซับซ้อนเราสามารถ Custom ระบบล็อกอินเองก็ได้เช่นกัน ...

จัดการสิทธิ์การเข้าใช้งาน

ในเว็บไซต์หรือระบบงานของเราหากเราต้องการกำหนดสิทธิ์การเข้าใช้งานในแต่ละเมนู เช่น เมนูไหนบ้างที่จะให้ผู้ใช้ทั่วไปเข้าใช้งานได้ เมนูไหนบ้างที่จะให้ผู้ดูแลระบบใช้งานได้ เมนูไหนบ้างที่จะให้สมาชิกใช้งานได้ ซึ่งจะมีสิทธิ์ในการเข้าถึงข้อมูลที่แตกต่างกัน ในหนังสือนี้จะยกตัวอย่างการจัดการสิทธิ์ โดยจะมีสิทธิ์การเข้าใช้งานอยู่ 3 ระดับ ดังนี้

1.ผู้ดูแลระบบ สามารถ ทำรายการต่าง ๆ ได้ดังนี้

- เพิ่ม แก้ไข ลบ ข้อมูลบทความได้ทั้งหมด
- เพิ่ม แก้ไข ลบ ข้อมูลประเภทบทความได้ทั้งหมด
- เพิ่ม แก้ไข ลบ ข้อมูลสินค้าได้ทั้งหมด
- เพิ่ม แก้ไข ลบ แบน สมาชิกได้ทั้งหมด

**จะต้อง Login เข้าระบบก่อน (role = 1 หมายถึง ผู้ดูแลระบบ)

2.สมาชิก สามารถทำรายการต่าง ๆ ได้ดังนี้

- เพิ่ม แก้ไข ลบ ข้อมูลบทความได้เฉพาะบทความของตัวเองเท่านั้น
- เพิ่ม แก้ไข ลบ ข้อมูลสินค้าของตัวเองเท่านั้น

**จะต้อง Login เข้าระบบก่อน (role = 2 หมายถึง สมาชิก)

3.ผู้ใช้ทั่วไป สามารถ

- ดูข้อมูลบทความได้ทั้งหมด
- ดูข้อมูลประเภทบทความได้ทั้งหมด
- ดูข้อมูลสินค้าได้ทั้งหมด

**ไม่ต้อง Login เข้าระบบ (role = 0 หมายถึง ผู้ใช้ทั่วไป)

ตัวอย่างการกำหนดสิทธิ์ ผู้ดูแลระบบ สามารถ ทำรายการต่าง ๆ ได้ดังนี้

- เพิ่ม แก้ไข ลบ ข้อมูลบทความได้ทั้งหมด

เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources\views\post\... แล้วกำหนดสิทธิ์การเข้าใช้งานเมนู ดังนี้

```
//resources\views\post\index.blade.php
@extends('layouts.frontend')
@section('title', 'หน้าแรกบทความ')
@section('content')
  <section class="py-5 container">
    <div class="row">
      <div class="col-lg-12  mx-auto">
        <br />
        <div class="float-end">
          @auth
            @if (Auth::user()->role === 1)
              @if (request()->has('trashed'))
                <a href="{{ route('posts.index') }}" class="btn btn-info">ดูบทความทั้งหมด</a>
                <a href="{{ route('posts.restoreAll') }}" class="btn btn-success">กู้คืนทั้งหมด</a>
              @else
                <a href="{{ route('create', ['trashed' => 'post']) }}" class="btn btn-danger">เพิ่มบทความใหม่
                </a>
                <a href="{{ route('posts.index', ['trashed' => 'post']) }}"
                  class="btn btn-primary">บทความที่ถูกลบ
                </a>
              @endif
            @elseif(Auth::user()->role === 2)
              @if (request()->has('trashed'))
                <a href="{{ route('posts.index') }}" class="btn btn-info">ดูบทความทั้งหมด</a>
              @else
                <a href="{{ route('create', ['trashed' => 'post']) }}" class="btn btn-danger">เพิ่มบทความใหม่
                </a>
                <a href="{{ route('posts.index', ['trashed' => 'post']) }}"
                  class="btn btn-primary">บทความที่ถูกลบ
                </a>
              @endif
            @endif
          @endauth
        </div>
        <table class="table table-hover">
          <thead>
            <tr>
              <th scope="col">ลำดับ</th>
              <th scope="col">หัวข้อ</th>
              <th scope="col">รายละเอียด</th>
              <th scope="col">จัดการ</th>
            </tr>
          </thead>
```

```
//resources\views\post\index.blade.php

            @foreach ($posts as $post)
              <tbody>
                <tr>
                  <th scope="col">{{ $post->id }}</th>
                  <td scope="col">{{ $post->post_title }}</td>
                 <td scope="col" style="width: 50%">{{ $post->post_detail }}</td>
                  <td scope="col">
                    <a href="{{ route('show', $post->id) }}" class="btn btn-success btn-sm">ดู</a>
                    @auth
                      @if (Auth::user()->role === 1 )
                        <a href="{{ route('edit', $post->id) }}" class="btn btn-info btn-sm">แก้ไข</a>|
                        @if (request()->has('trashed'))
                          <a href="{{ route('posts.restore', $post->id) }}"
                            class="btn btn-success btn-sm">กู้คืน</a>
                        @else
                          <a href="{{ route('destroy', $post->id) }}" class="btn btn-danger btn-sm"
                            onclick="return confirm('ยืนยันการลบหรือไม่')">ลบ
                          </a>
                        @endif
                      @endif
                      @elseif(Auth::user()->role === 2)
                        <a href="{{ route('edit', $post->id) }}" class="btn btn-info btn-sm">แก้ไข</a>|
                          <a href="{{ route('destroy', $post->id) }}" class="btn btn-danger btn-sm"
                            onclick="return confirm('ยืนยันการลบหรือไม่')">ลบ
                          </a>
                    @endif
                   @endauth
                  </td>
                </tr>
              </tbody>
            @endforeach
          </table>
          {{ $posts->links() }}
        </div>
      </div>
    </section>
@endsection
```

ในตัวอย่างส่วนที่ทำไฮไลท์สีเขียวไว้นั้นเป็นการเขียนคำสั่งในการตรวจสอบสถานะการล็อกอินว่ามีการล็อกอินเข้ามาแล้วหรือยัง โดยจะใช้คำสั่ง @auth และปิดด้วย @endauth หากมีการล็อกอินก็จะมีการตรวจสอบสิทธิ์การเข้าถึงเมนู โดยใช้คำสั่ง @if(Auth::user()->role === 1) และ Auth::user()->role === 2 เพื่อตรวจสอบสิทธิ์ Auth::user()->role === 1 คือ ผู้ดูแลระบบมีสิทธิ์ในการแก้ไขข้อมูล ลบข้อมูล และกู้คืนข้อมูล Auth::user()->role === 2 คือ สมาชิกมีสิทธิ์ในการแก้ไขข้อมูล ลบข้อมูล

จุดสังเกต 1 การตรวจสอบว่าผู้ใช้งานได้ล็อกอินเข้าระบบหรือไม่ ให้ใช้คำสั่งนี้

```
@auth
   ....เข้าระบบแล้ว

@else

   ... ไม่ได้เข้าระบบ
@endauth
```

หรือ

```
@guest
   ....เข้าระบบแล้ว

@else

   ... ไม่ได้เข้าระบบ
@endguest
```

จุดสังเกต 2 การตรวจสอบสิทธิ์การใช้งานแต่ละส่วน ให้ใช้คำสั่งนี้

```
@if (Auth::user()->role === 1 )
            ...สำหรับผู้ดูแลระบบเท่านั้นจะเห็นในส่วนนี้

@endif
```

หากคำสั่งนั้นมีหลายเงื่อนไขสามารถใช้เครื่องหมาย && (and) หรือ || (or) เพื่อตรวจสอบเงื่อนไขหลาย ๆ รายการได้

```
@if (Auth::user()->role === 1  && Auth::user()->status === 1
            ...สำหรับผู้ดูแลระบบจะเห็นในส่วนนี้

@endif
```

```
@if (Auth::user()->role === 1  || Auth::user()->role  === 2
            ...สำหรับผู้ดูแลระบบจะเห็นในส่วนนี้

@endif
```

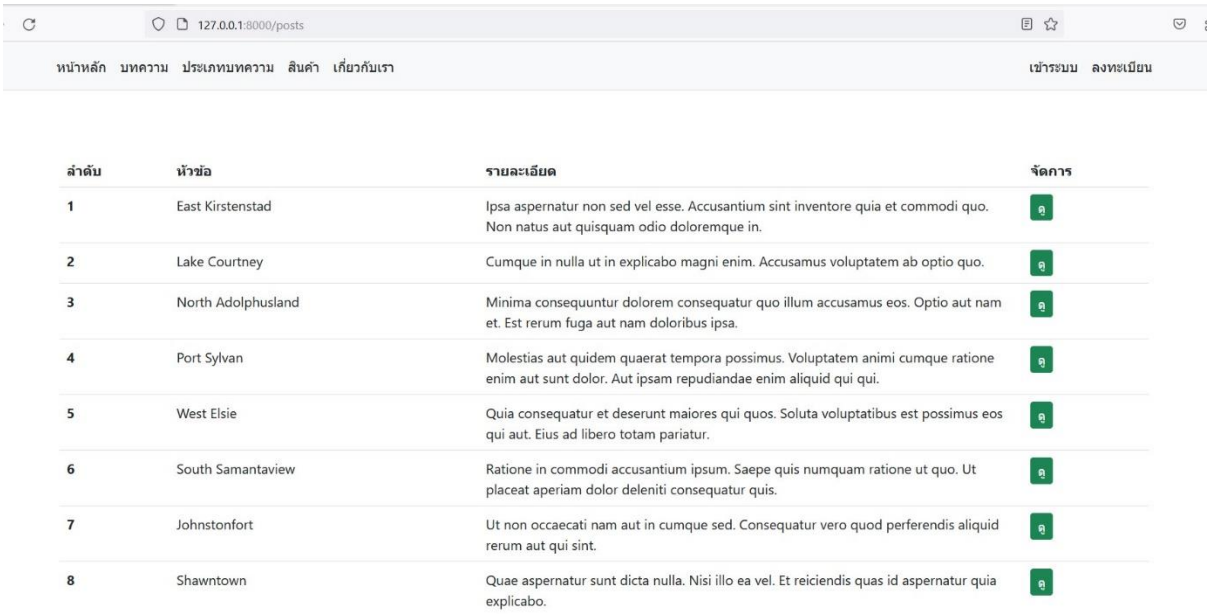
ตัวอย่างการกำหนดสิทธิ์ - เพิ่ม แก้ไข ลบ ข้อมูลประเภทบทความได้ทั้งหมด

เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources\views\category\... แล้วกำหนดสิทธิ์การเข้าใช้งานเมนู ดังนี้

```
//resources\views\category\index.blade.php
@extends('layouts.frontend')
@section('title', 'ประเภทบทความ')
@section('content')
  <section class="py-5 container">
    <div class="row py-lg-5">
      <div class="mb-3">
        @auth
          @if (Auth::user()->role === 1)
            <a href="{{ route('category.create') }}" class="btn btn-success">เพิ่มประเภทบทความใหม่</a>
          @endif
        @endauth
      </div>
      <div class="col-lg-6 col-md-8 mx-auto">
        @forelse ($category as $cat)
          <ul>
            <li>Title : <b>{{ $cat->category_name }}</b> <br />
              @auth
                @if (Auth::user()->role === 1)
                 <a href="{{ route('category.edit', $cat->id) }}" class="btn btn-info btn-sm">แก้ไข</a> |
                  <a href="{{ route('category.destroy', $cat->id) }}" class="btn btn-danger btn-sm"
                      onclick="return confirm('ยืนยันการลบหรือไม่')">ลบข้อมูล</a>
                @elseif(Auth::user()->role === 2)
                   <a href="{{ route('category.edit', $cat->id) }}" class="btn btn-info btn-sm">แก้ไข</a>
                @endif
              @endauth
            </li>
          </ul>
        @empty
          <span class="text-danger">ไม่พบข้อมูล</span>
        @endforelse
      </div>
    </div>
  </section>
@endsection
```

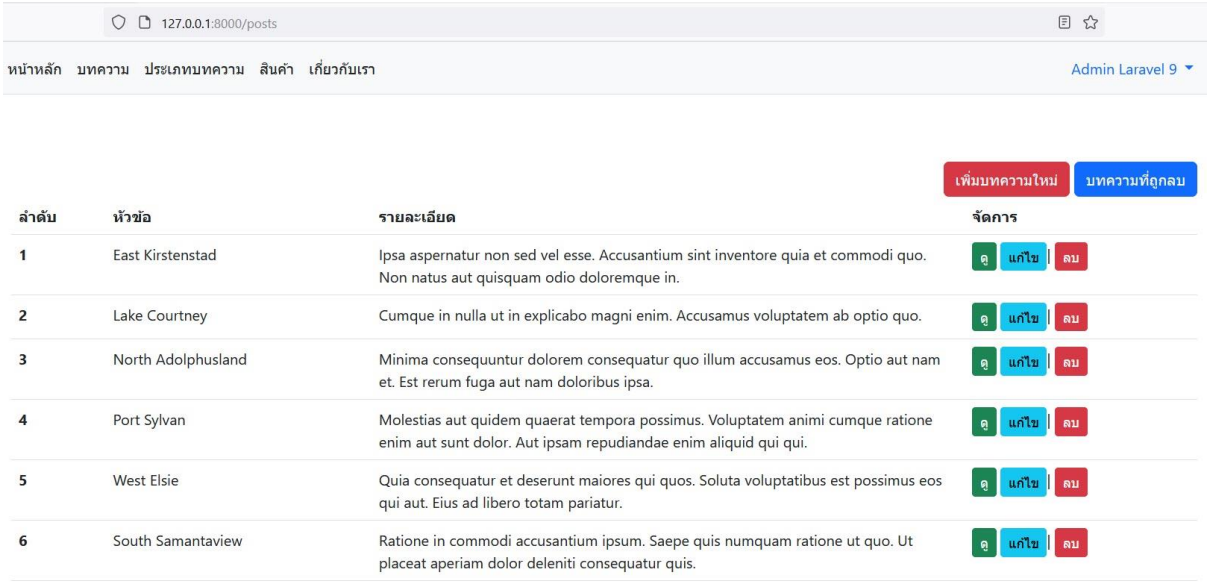
บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/post และ localhost:8000/category ที่ Browser

หน้าบทความ localhost:8000/post เมื่อไม่มีการล็อคอินเข้าระบบ



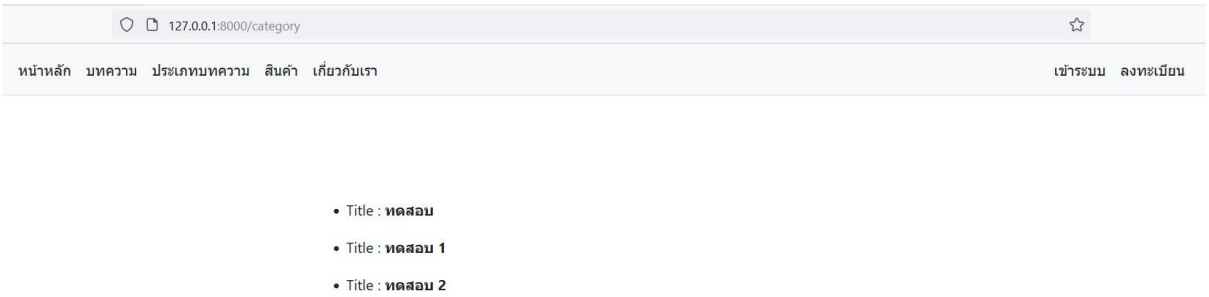
หากไม่มีการล็อกอินเข้าระบบก่อน ปุ่มการเพิ่มบทความ ปุ่มบทความที่ถูกลบ ปุ่มแก้ไข ปุ่มลบ ก็จะถูกซ่อนไว้

หน้าบทความ localhost:8000/post เมื่อมีการล็อคอินเข้าระบบเรียบร้อยแล้ว

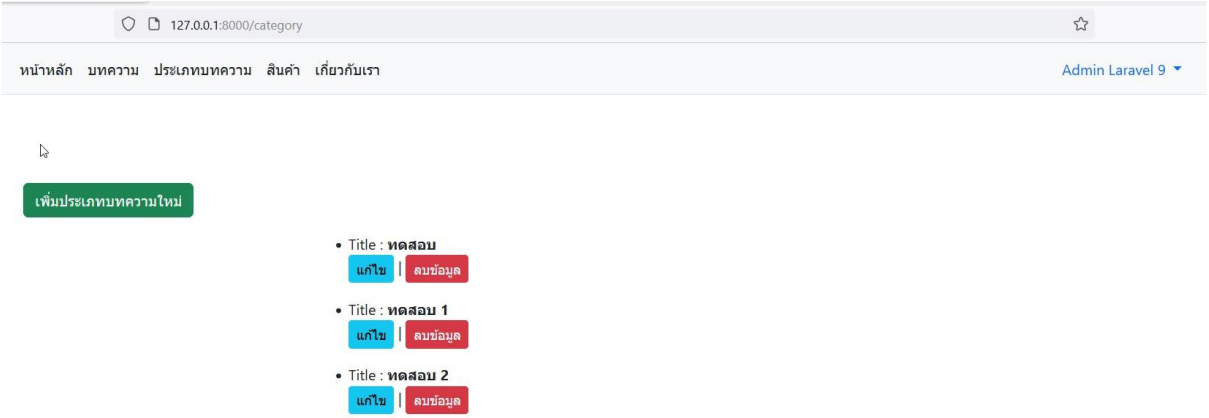


ถ้ามีการล็อกอินเข้าระบบ ปุ่มการเพิ่มบทความ ปุ่มบทความที่ถูกลบ ปุ่มแก้ไข ปุ่มลบ ก็จะถูกแสดงตามสิทธิ์ที่เราได้กำหนดไว้

หน้าประเภทบทความ localhost:8000/categoy เมื่อไม่มีการล็อคอินเข้าระบบ



หน้าประเภทบทความ localhost:8000/category เมื่อมีการล็อคอินเข้าระบบเรียบร้อยแล้ว



จากตัวอย่างที่ผ่านมานั้นเป็นการกำหนดสิทธิ์แบบพื้นฐานง่าย ๆ ไม่ซับซ้อนอะไรมากนัก ซึ่งเราสามารถนำไปประยุกต์ใช้ในงานได้ตามต้องการ นอกจากนั้นแล้วเรายังสามารถกำหนดบทบาทหน้าที่หรือสิทธิ์การเข้าถึงข้อมูลที่ซับซ้อนขึ้นได้ โดยใช้ Library มาช่วยจัดการ เรียนรู้ได้จากบทเรียนถัดไปได้เลยครับ ...

Role & Permission

Role คือ ชื่อของบทบาทหน้าที่ เช่น ผู้ดูแลระบบ (Admin) , ผู้เขียนบทความ (Writer), สมาชิก (User)

Permission คือ ชื่อของสิทธิ์ที่มีในแต่ละบทบาท เช่น ผู้ดูแลระบบมีสิทธิ์ในการจัดการอะไรได้ สมาชิกมีสิทธิ์จัดการอะไรได้บ้าง

ตารางแสดงตัวอย่างการกำหนดบทบาทแต่ละสิทธิ์

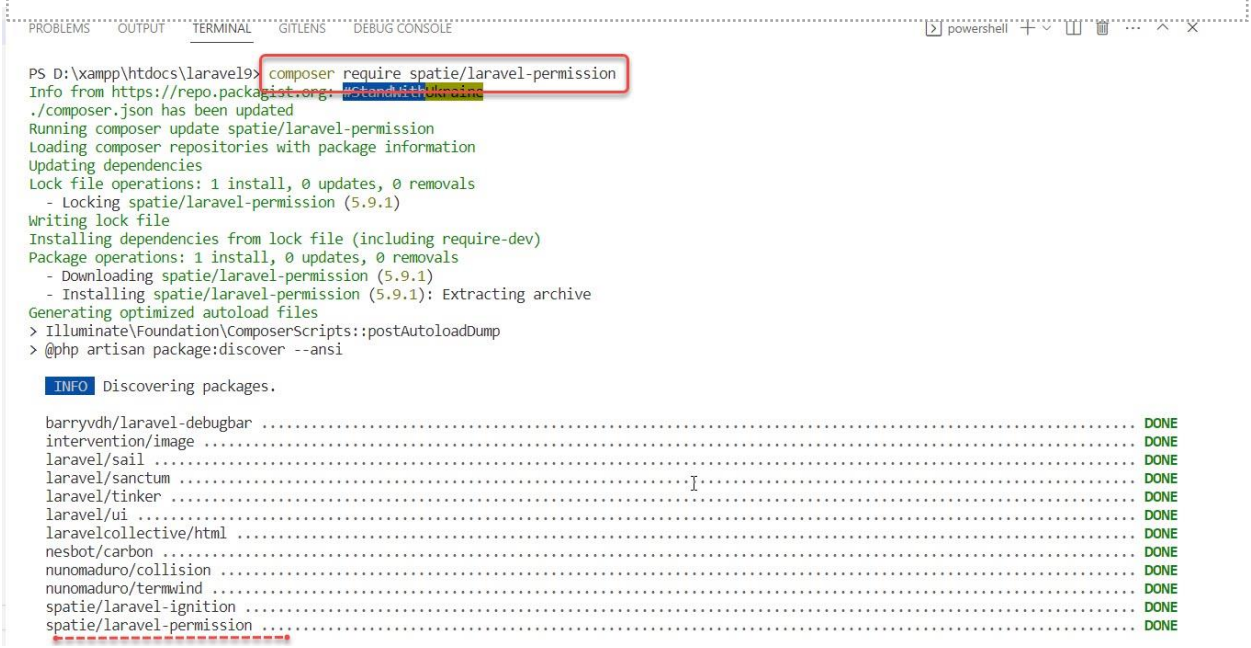
ผู้ดูแลระบบ (Admin)	ผู้เขียนบทความ (Writer)	สมาชิก (User)
จัดการข้อมูลสมาชิก	เพิ่ม แก้ไข ลบบทความของตนเองได้	อ่านบทความได้ เขียนคอมเมนท์ได้
กำหนดสิทธิ์ให้กับสมาชิก		
จัดการข้อมูลบทความ		

ในหนังสือนี้จะยกตัวอย่างการใช้งาน Spatie Permission ซึ่งเป็นอีก Library ที่ใช้งานง่าย เราจะใช้ Library นี้สร้าง ACL (Access Control List) เพื่อจัดการ Role & Permission โดยจะยกตัวอย่างการเพิ่มสมาชิกและกำหนดสิทธิ์การเข้าถึงข้อมูลและสิทธิ์การจัดการข้อมูลสินค้า เพื่อให้ผู้อ่านเข้าใจยิ่งขึ้น เนื้อหานี้จะเป็นเนื้อหาที่ต่อเนื่องจากหน้า 234 เรื่องระบบล็อกอินและระบบลงทะเบียน ฉะนั้นหากผู้อ่านยังไม่ได้สร้างระบบล็อกอินและระบบลงทะเบียน ผู้อ่านต้องกลับไปสร้างก่อนนะครับ จากนั้นค่อยมาติดตั้ง Library Spatie Permission ดูขั้นตอนได้ดังนี้

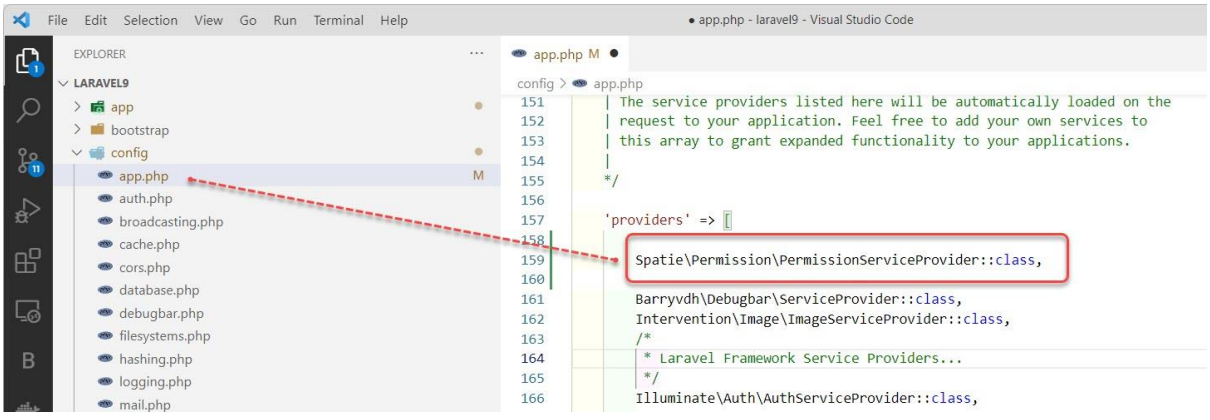
(สามารถดูขั้นตอนเพิ่มเติมได้ที่ https://spatie.be/docs/laravel-permission/v5/installation-laravel)

1.ติดตั้ง spatie/laravel-permission Packages ผ่าน Terminal ด้วยคำสั่ง

```
composer require spatie/laravel-permission
```

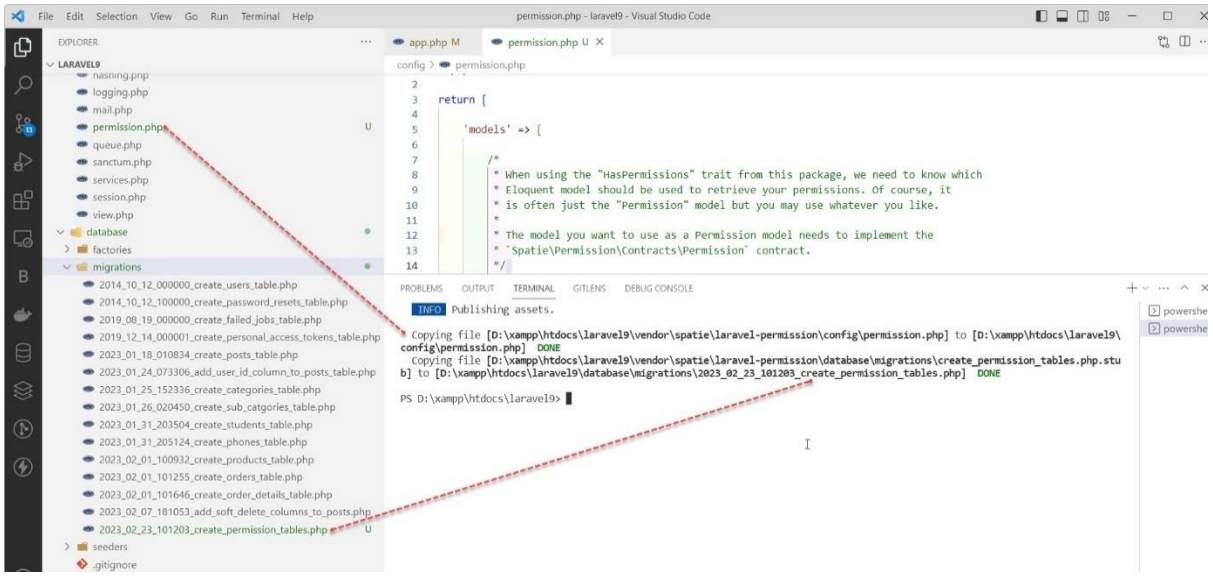


2.เปิดไฟล์ app.php ในโฟลเดอร์ config/app.php แล้วเพิ่มคำสั่ง Spatie... ในส่วนของ provider ดังนี้



3.พิมพ์คำสั่งนี้ ผ่าน Terminal เพื่อให้ระบบสร้างไฟล์ migration สำหรับกำหนดโครงสร้างตาราง permission

```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
```



ไฟล์ permission.php จะถูกสร้างขึ้นมาใหม่อยู่ในโฟลเดอร์ app/... และไฟล์ xxx_create_permission_tables.php จะถูกสร้างขึ้นมาใหม่อยู่ในโฟลเดอร์ database/migrations/... เมื่อเปิดดูที่ไฟล์ทั้งสองไฟล์ก็จะเห็นว่า ระบบจะกำหนดโครงสร้างตารางให้เราแล้ว

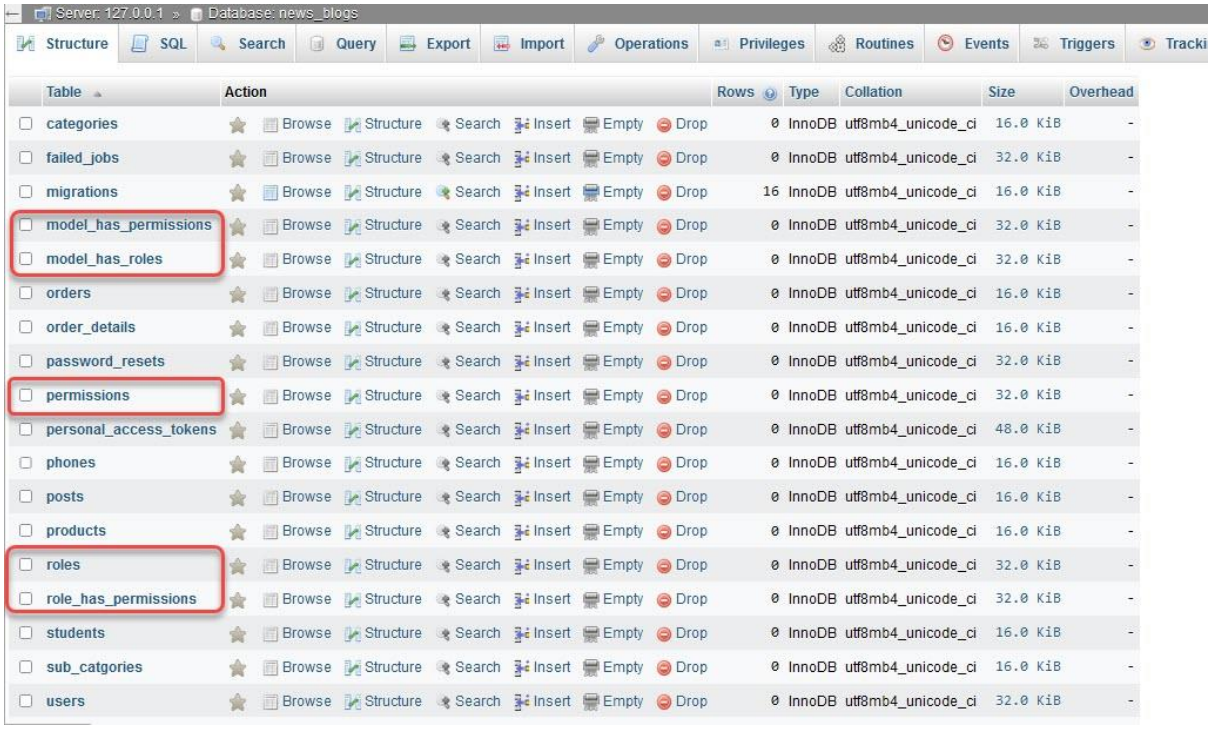
4.รันคำสั่ง migrate ผ่าน Terminal

```
php artisan migrate:refresh
```

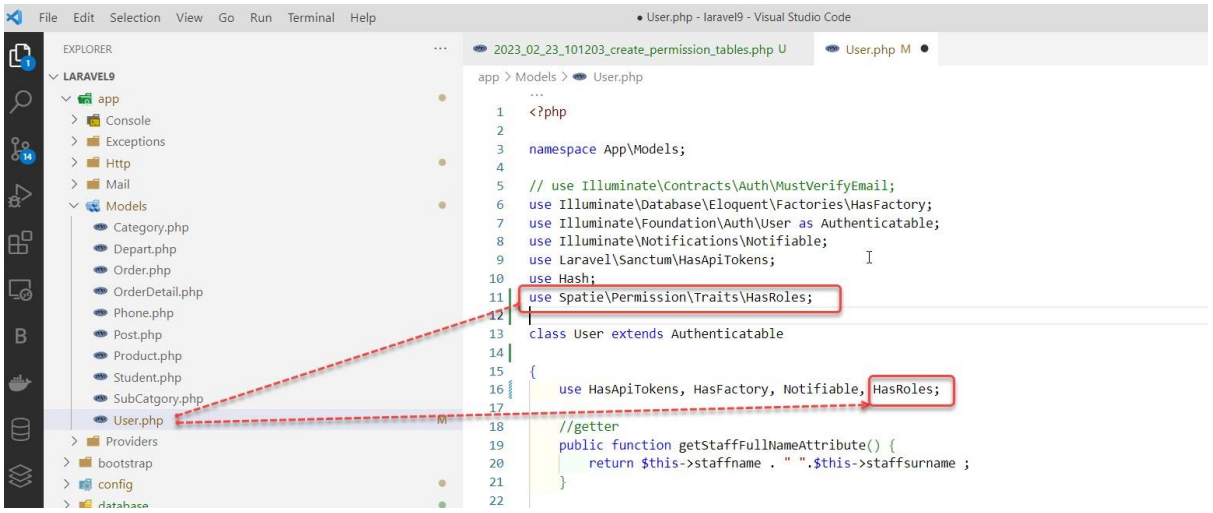


ระบบจะทำการลบตารางเก่าแล้วสร้างตารางให้ใหม่ เมื่อเปิดดูในโปรแกรม phpMyAdmin จะมีตารางสร้างขึ้นเพิ่มให้เราใหม่ 5 ตาราง คือ

1. model_has_permissions สำหรับเก็บโมเดล permissions
2. model_has_roles สำหรับเก็บโมเดล roles
3. rermissions สำหรับเก็บรายการ permission ต่าง ๆ
4. roles สำหรับเก็บรายการบทบาทหน้าที่
5. role_has_permissions สำหรับเก็บสิทธิ์ของแต่ละ user

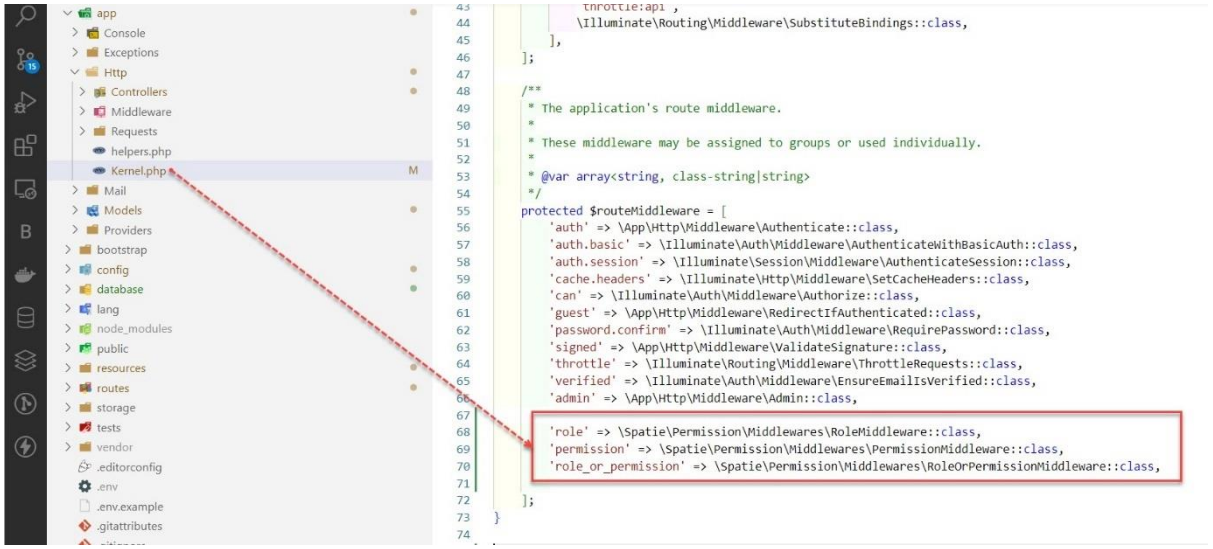


5.เปิดไฟล์โมเดล User.php ในโฟลเดอร์ app/Models/... เพิ่มคำสั่งเข้าไป ดังนี้



6.เปิดไฟล์ kernel.php ในโฟลเดอร์ app/Http/... แล้วเพิ่ม middleware เข้าไปในส่วนของ $routeMiddleware = [...] ดังนี้

```
protected $routeMiddleware = [
    ....
    'role' => \Spatie\Permission\Middlewares\RoleMiddleware::class,
    'permission' => \Spatie\Permission\Middlewares\PermissionMiddleware::class,
    'role_or_permission' => \Spatie\Permission\Middlewares\RoleOrPermissionMiddleware::class,
]
```



7.เปิดไฟล์ web.php ในโฟลเดอร์ routes/... แล้วเพิ่ม routes ใหม่ ดังนี้

```
//routes/web.php
use App\Http\Controllers\UserController;
use App\Http\Controllers\RoleController;

Route::resource('users', UserController::class);
Route::resource('roles', RoleController::class);
```

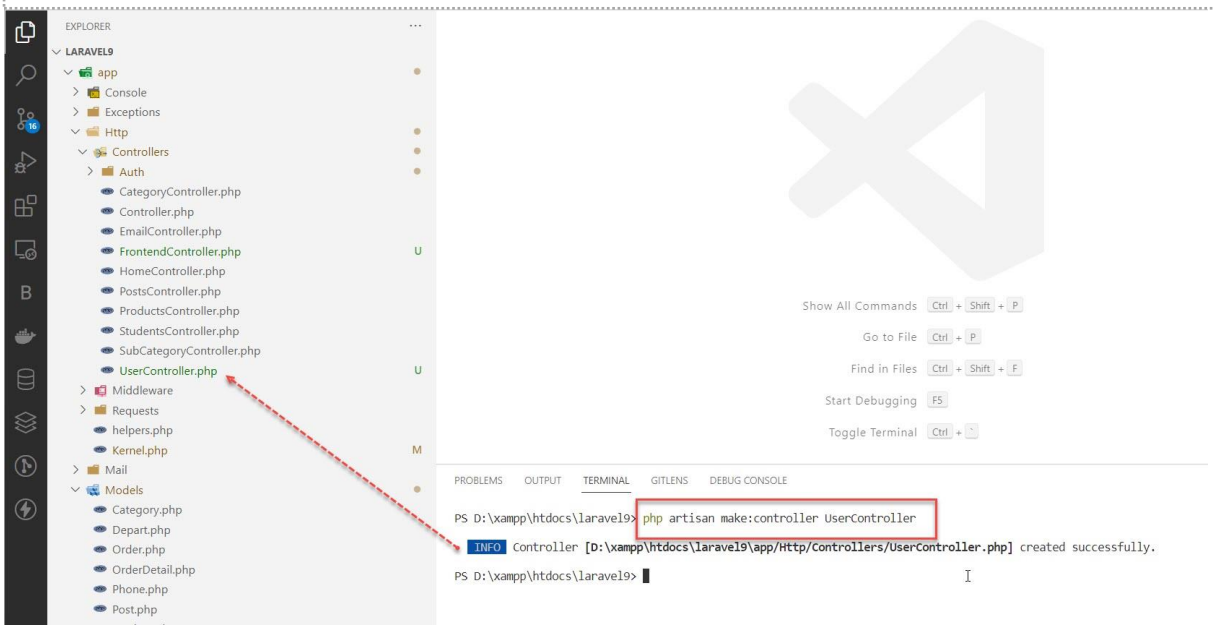
โค้ดเต็มหน้าถัดไป...

```
//routes/web.php
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\PostsController;
use App\Http\Controllers\CategoryController;
use App\Http\Controllers\StudentsController;
use App\Http\Controllers\FrontendController ;
use App\Http\Controllers\ProductsController;
use App\Http\Controllers\EmailController;
use App\Models\Order ;
use App\Models\Product ;
use App\Http\Controllers\UserController;
use App\Http\Controllers\RoleController;
//Frontend
Route::get('/',[FrontendController::class,'index']);
Route::get('/aboute',[FrontendController::class,'aboute']);
Route::get('send-email', [EmailController::class, 'send_email']);
Auth::routes();
Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');
//Soft Delete and Data Restore
Route::get('posts/trashed', [PostsController::class, 'trashed'])->name('posts.trashed');
Route::get('posts/restore/{id}', [PostsController::class, 'restore'])->name('posts.restore');
Route::get('posts/restore-all', [PostsController::class, 'restoreAll'])->name('posts.restoreAll');
//products
Route::get('products',[ProductsController::class,'index']) ;
Route::get('products/create',[ProductsController::class,'create'])->name('products.create') ;
Route::post('products/store',[ProductsController::class,'store'])->name('products.store') ;
Route::get('products/edit/{id}',[ProductsController::class,'edit'])->name('products.edit') ;
Route::post('products/update/{id}',[ProductsController::class,'update'])->name('products.update') ;
Route::delete('products/destroy/{id}',[ProductsController::class,'destroy'])->name('products.destroy') ;
Route::get('/order/{id}',function($id){
   $order = Order::find($id) ;
   return $order->rProduct()->orderBy('name','desc')->get();
});
Route::get('/order/product/{id}',function($id){
   $order = Product::find($id) ;
   return $order->rOrder()->orderBy('id','desc')->get();
});
//StudentsController
Route::get('student/all',[StudentsController::class,'index'])->name('student') ;

//PostController
Route::get('/posts',[PostsController::class,'index'])->name('posts.index'); // แสดงข้อมูลทั้งหมด
Route::get('/posts/create',[PostsController::class,'create'])->name('create') ;  //ฟอร์มเพิ่มข้อมูลใหม่
Route::post('store',[PostsController::class,'store'])->name('store') ;  //บันทึกข้อมูล
Route::get('post/edit/{id}',[PostsController::class,'edit'])->name('edit') ; //ฟอร์มแก้ไข
Route::post('post/update/{id}',[PostsController::class,'update'])->name('update'); //บันทึกการแก้ไข
Route::get('post/destroy/{id}',[PostsController::class,'destroy'])->name('destroy') ; //ลบข้อมูล
```

```
//routes/web.php
Route::get('post/show/{id}',[PostsController::class,'show'])->name('show') ; //แสดงรายละเอียดข้อมูล
//CategoryController
Route::get('/category',[CategoryController::class,'index']); // แสดงข้อมูลทั้งหมด
Route::get('category/create',[CategoryController::class,'create'])->name('category.create') ;  //ฟอร์มเพิ่มข้อมูลใหม่
Route::post('categorystore',[CategoryController::class,'store'])->name('category.store') ;  //บันทึกข้อมูล
Route::get('category/edit/{id}',[CategoryController::class,'edit'])->name('category.edit') ; //ฟอร์มแก้ไข
Route::post('category/update/{id}',[CategoryController::class,'update'])->name('category.update'); //บันทึกการแก้ไข
Route::get('category/destroy/{id}',[CategoryController::class,'destroy'])->name('category.destroy') ; //ลบข้อมูล

Route::resource('users', UserController::class);
Route::resource('roles', RoleController::class);
```

8.สร้าง UserController.php โดยพิมพ์คำสั่งผ่าน Terminal ดังนี้

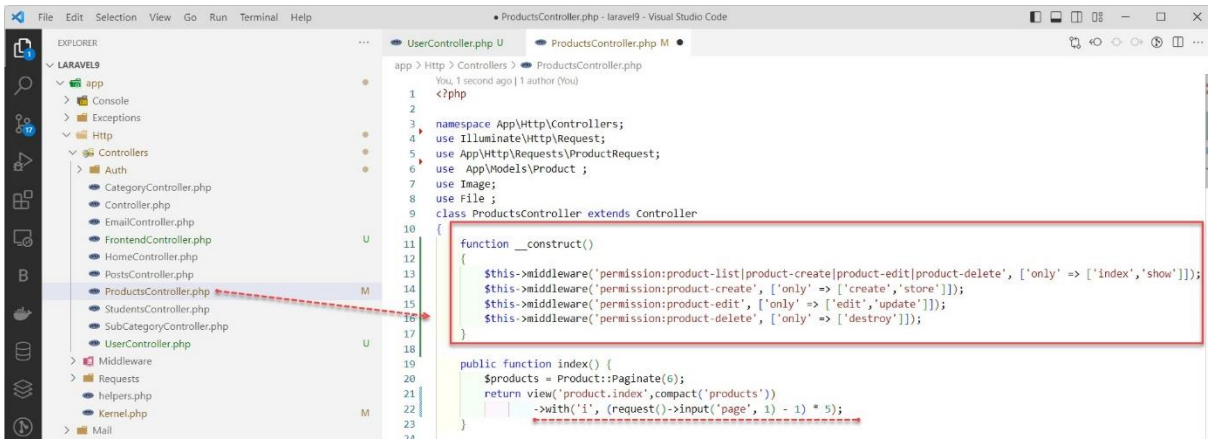
```
php artisan make:controller UserController
```



เปิดไฟล์ UserController.php แล้วเพิ่มโค้ดเข้าไป ดังนี้

```php
//app/Http/Controllers/UserController.php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\Models\User;
use Spatie\Permission\Models\Role;
use DB;
use Hash;
use Illuminate\Support\Arr;

class UserController extends Controller
{
   public function index(Request $request)
   {
      $data = User::orderBy('id','DESC')->paginate(5);
      return view('users.index',compact('data'))
         ->with('i', ($request->input('page', 1) - 1) * 5);
   }

   public function create()
   {
      $roles = Role::pluck('name','name')->all();
      return view('users.create',compact('roles'));
   }

   public function store(Request $request)
   {
      $this->validate($request, [
         'name' => 'required',
         'email' => 'required|email|unique:users,email',
         'password' => 'required|same:confirm-password',
         'roles' => 'required'
      ]);

      $input = $request->all();
      $input['password'] = Hash::make($input['password']);

      $user = User::create($input);
      $user->assignRole($request->input('roles'));

      return redirect()->route('users.index')
                  ->with('success','User created successfully');
   }
```

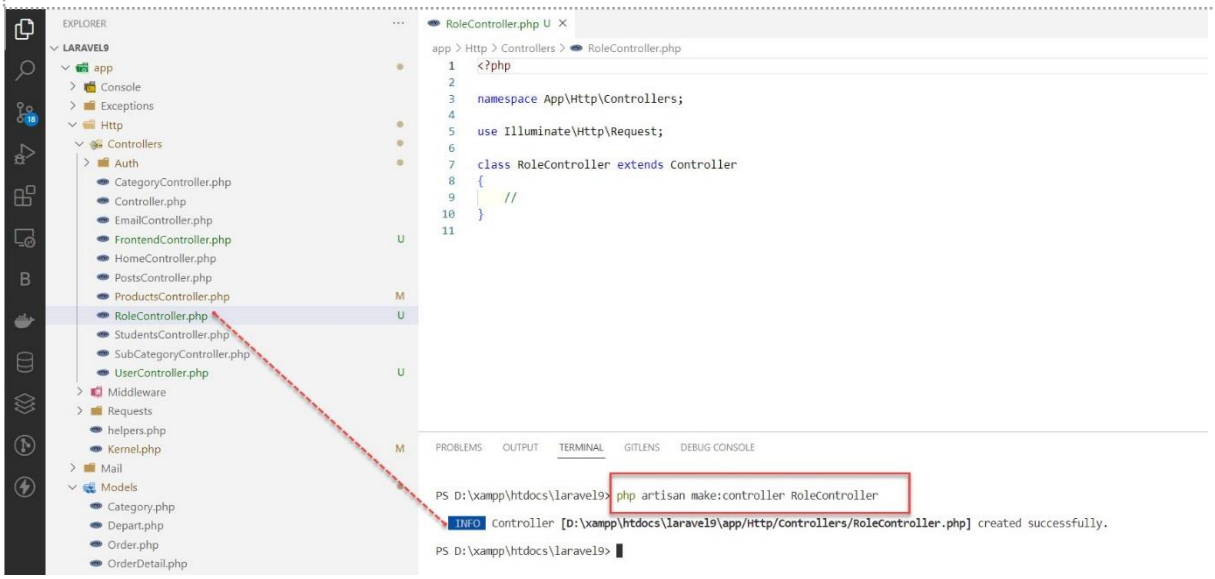
8.เปิดไฟล์ ProductsController.php แล้วแก้ไขโค้ดใหม่ โดยเพิ่มเมธอด __construct() และเพิ่มโค้ดในส่วนของเมธอด index() ดังนี้



ใน __contruct() นั้นเป็นการกำหนดสิทธิ์การเข้าถึงเมนูการจัดการสินค้าแต่ละสิทธิ์ ซึ่งประกอบด้วย product-list, product-create , product-edit , product-delete โดยจะใช้ middleware กำหนดสิทธิ์การเข้าถึงแต่ละสิทธิ์ และในเมธอด index() บรรทัด 22 มีการเพิ่มโค้ดสำหรับจัดเลขหน้าให้ถูกต้อง จากนั้นก็ส่งตัวแปรไปให้ส่วนแสดงผล คือ product.index

9.สร้างไฟล์ RoleController.php โดยพิมพ์คำสั่งผ่าน Terminal ดังนี้

```
php artisan make:controller RoleController
```



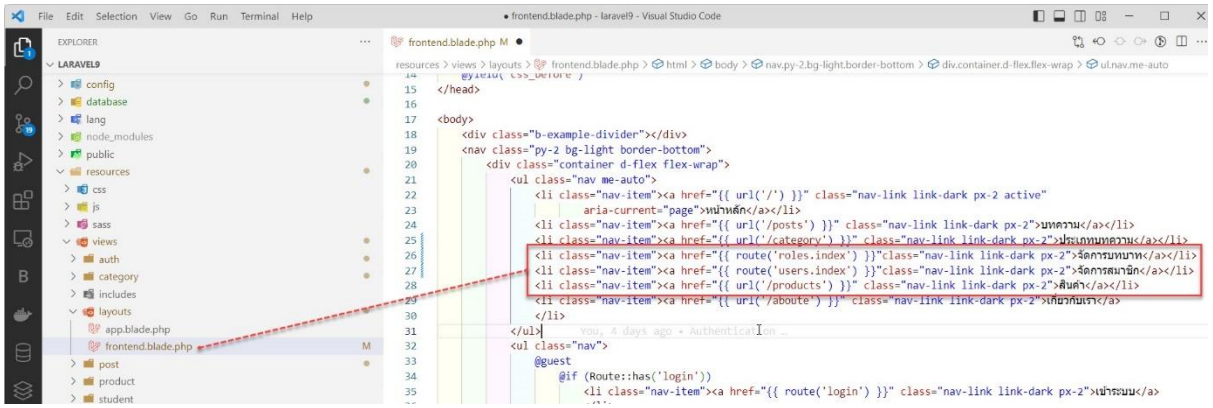
เปิดไฟล์ RoleController.php แล้วเพิ่มโค้ดเข้าไป ดังนี้

```
//app/Http/Controllers/RoleController.php
 <?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;
use DB;

class RoleController extends Controller
{

   function __construct()
   {
      $this->middleware('permission:role-list|role-create|role-edit|role-delete', ['only' => ['index','store']]);
      $this->middleware('permission:role-create', ['only' => ['create','store']]);
      $this->middleware('permission:role-edit', ['only' => ['edit','update']]);
      $this->middleware('permission:role-delete', ['only' => ['destroy']]);
   }

   public function index(Request $request)
   {
      $roles = Role::orderBy('id','DESC')->paginate(5);
      return view('roles.index',compact('roles'))->with('i', ($request->input('page', 1) - 1) * 5);
   }

   public function create()
   {
      $permission = Permission::get();
      return view('roles.create',compact('permission'));
   }

   public function store(Request $request)
   {
      $this->validate($request, [
         'name' => 'required|unique:roles,name',
         'permission' => 'required',
      ]);

      $role = Role::create(['name' => $request->input('name')]);
      $role->syncPermissions($request->input('permission'));

      return redirect()->route('roles.index')->with('success','Role created successfully');
   }
```

มีต่อ ...

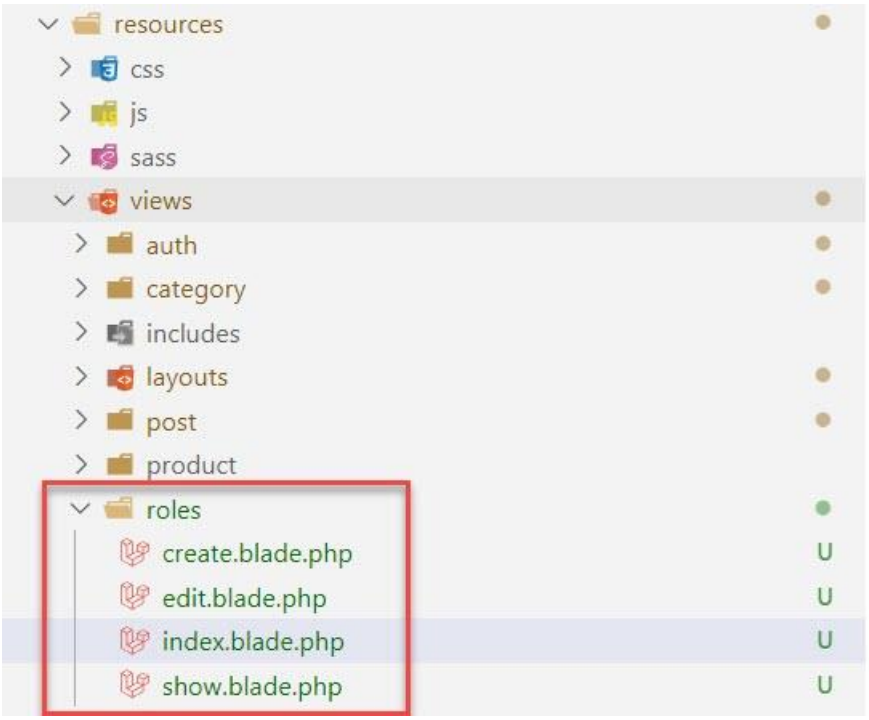
```
//app/Http/Controllers/RoleController.php
 public function show($id)

   {
      $role = Role::find($id);
      $rolePermissions =
Permission::join("role_has_permissions","role_has_permissions.permission_id","=","permissions.id")
         ->where("role_has_permissions.role_id",$id)
         ->get();

      return view('roles.show',compact('role','rolePermissions'));
   }

   public function edit($id)
   {
      $role = Role::find($id);
      $permission = Permission::get();
      $rolePermissions = DB::table("role_has_permissions")->where("role_has_permissions.role_id",$id)
                  ->pluck('role_has_permissions.permission_id','role_has_permissions.permission_id')
                  ->all();

      return view('roles.edit',compact('role','permission','rolePermissions'));
   }

   public function update(Request $request, $id)
   {
      $this->validate($request, [
         'name' => 'required',
         'permission' => 'required',
      ]);

      $role = Role::find($id);
      $role->name = $request->input('name');
      $role->save();

      $role->syncPermissions($request->input('permission'));

      return redirect()->route('roles.index')->with('success','Role updated successfully');
   }

   public function destroy($id)
   {
      DB::table("roles")->where('id',$id)->delete();
      return redirect()->route('roles.index')->with('success','Role deleted successfully');
   }
}
```

8.เปิดไฟล์ frontend.blade.php ในโฟลเดอร์ resources/views/layouts/... แล้วเพิ่มลิงค์สำหรับจัดการ ดังนี้



ให้เพิ่มลิงค์สำหรับจัดการเข้าไป คือ ลิงค์จัดการบทบาท ลิงค์จัดการสมาชิก และ สินค้า

9.สร้างไฟล์สำหรับแสดงผลในส่วนของ roles ให้สร้างโฟลเดอร์ใหม่ชื่อ roles ใน resources/views/roles/... และสร้างไฟล์เพิ่มดังนี้ index.blade.php, create.blade.php, edit.blade.php, และ show.blade.php จะได้โครงสร้างไฟล์สำหรับโฟลเดอร์ roles ดังนี้



จากนั้นให้เปิดไฟล์แต่ละไฟล์แล้วเพิ่มโค้ดเข้าไป ตามลำดับ ดังนี้

สำหรับไฟล์ index.blade.php (resources/views/roles/index.blade.php)

```
@extends('layouts.frontend)
@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2>จัดการบทบาทหน้าที่</h2>
            </div>
            <div class="pull-right">
                @can('role-create')
                    <a class="btn btn-success" href="{{ route('roles.create') }}"> สร้างบทบาทใหม่ </a>
                @endcan
            </div>
        </div>
    </div>
    @if ($message = Session::get('success'))
        <div class="alert alert-success">
            <p>{{ $message }}</p>
        </div>
    @endif
    <table class="table table-bordered">
        <tr>
            <th>ลำดับ</th>
            <th>รายการ</th>
            <th width="280px">จัดการ</th>
        </tr>
@foreach ($roles as $key => $role)
            <tr>
                <td>{{ ++$i }}</td>
                <td>{{ $role->name }}</td>
                <td>
                    <a class="btn btn-info" href="{{ route('roles.show', $role->id) }}">ดู</a>
                    @can('role-edit')
                        <a class="btn btn-primary" href="{{ route('roles.edit', $role->id) }}">แก้ไข</a>
                    @endcan
                    @can('role-delete')
                        {!! Form::open(['method' => 'DELETE', 'route' => ['roles.destroy', $role->id], 'style' => 'display:inline']) !!}
                        {!! Form::submit('ลบ', ['class' => 'btn btn-danger']) !!}
                        {!! Form::close() !!}
                    @endcan
                </td>
            </tr>
        @endforeach
    </table>
    {!! $roles->links() !!}
@endsection
```

สำหรับไฟล์ create.blade.php (resources/views/roles/create.blade.php)

```
@extends('layouts.frontend')
@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left"><h2>สร้างบทบาทใหม่</h2></div>
            <div class="pull-right"><a class="btn btn-primary" href="{{ route('roles.index') }}"> กลับ </a></div>
        </div>
    </div>
    @if (count($errors) > 0)
        <div class="alert alert-danger">
            <strong>Whoops!</strong> Something went wrong.<br><br>
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif
    {!! Form::open(['route' => 'roles.store', 'method' => 'POST']) !!}
    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>ชื่อบทบาท:</strong>
                {!! Form::text('name', null, ['placeholder' => 'Name', 'class' => 'form-control']) !!}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Permission:</strong>
                <br />
                @foreach ($permission as $value)
                    <label>{{ Form::checkbox('permission[]', $value->id, false, ['class' => 'name']) }}
                        {{ $value->name }}</label>
                    <br />
                @endforeach
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12 text-center">
            <button type="submit" class="btn btn-primary">บันทึก</button>
        </div>
    </div>
    {!! Form::close() !!}
@endsection
```

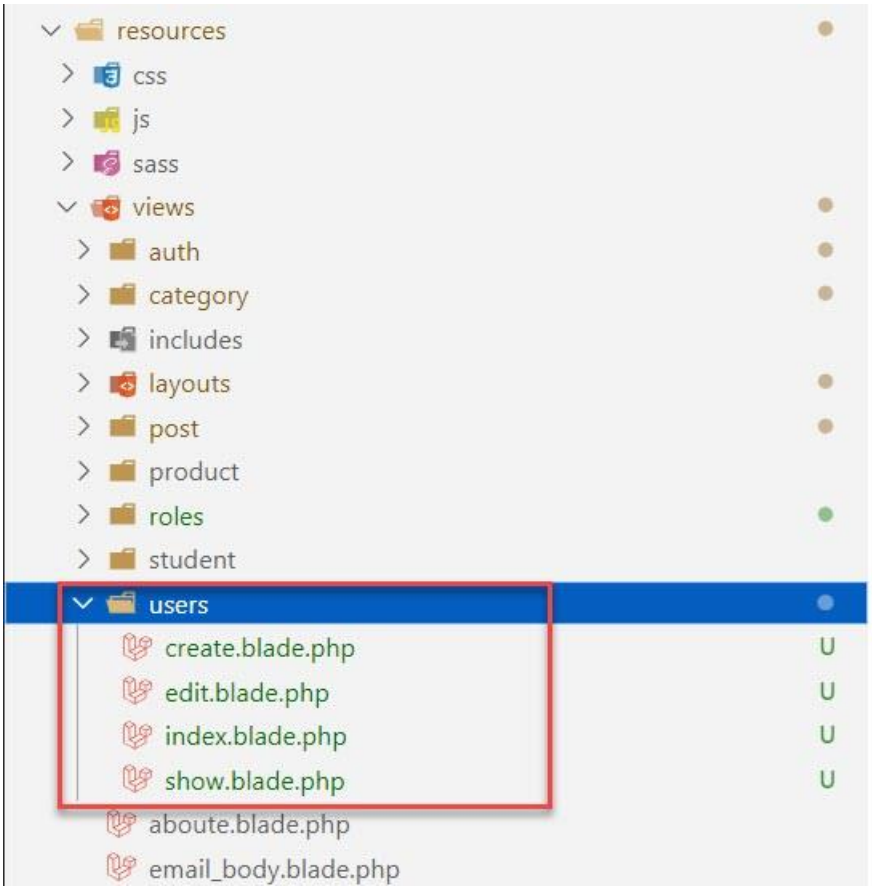
สำหรับไฟล์ edit.blade.php (resources/views/roles/edit.blade.php)

```
@extends('layouts.frontend')
@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left"><h2>แก้ไขบทบาท</h2>
            </div>
        <div class="pull-right"><a class="btn btn-primary" href="{{ route('roles.index') }}"> กลับ </a></div>
        </div>
    </div>
    @if (count($errors) > 0)
        <div class="alert alert-danger">
            <strong>Whoops!</strong> something went wrong.<br><br>
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif
    {!! Form::model($role, ['method' => 'PATCH', 'route' => ['roles.update', $role->id]]) !!}
    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>รายการ:</strong>
                {!! Form::text('name', null, ['placeholder' => 'Name', 'class' => 'form-control']) !!}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>สิทธิ์การเข้าใช้งาน:</strong>
                <br />
                @foreach ($permission as $value)
                    <label>{{ Form::checkbox('permission[]', $value->id, in_array($value->id, $rolePermissions) ? true : false, ['class' => 'name']) }}
                        {{ $value->name }}</label>
                    <br />
                @endforeach
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12 text-center">
            <button type="submit" class="btn btn-primary">บันทึก</button>
        </div>
    </div>
    {!! Form::close() !!}
@endsection
```

สำหรับไฟล์ show.blade.php (resources/views/roles/show.blade.php)

```
@extends('layouts.frontend')
@section('content')
<div class="row">
    <div class="col-lg-12 margin-tb">
        <div class="pull-left">
            <h2>แสดง</h2>
        </div>
        <div class="pull-right">
            <a class="btn btn-primary" href="{{ route('roles.index') }}"> กลับ </a>
        </div>
    </div>
</div>
<div class="row">
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <strong>ชื่อบทบาท:</strong>
            {{ $role->name }}
        </div>
    </div>
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <strong>สิทธิ์การเข้าใช้งาน:</strong>
            @if(!empty($rolePermissions))
                @foreach($rolePermissions as $v)
                    <label class="label label-success">{{ $v->name }},</label>
                @endforeach
            @endif
        </div>
    </div>
</div>
@endsection
```

10.สร้างไฟล์สำหรับแสดงผลในส่วนของ users ให้สร้างโฟลเดอร์ใหม่ชื่อ users ใน resources/views/users/... และสร้างไฟล์เพิ่มดังนี้ index.blade.php, create.blade.php, edit.blade.php, และ show.blade.php จะได้โครงสร้างไฟล์สำหรับโฟลเดอร์ users ดังนี้



จากนั้นให้เปิดไฟล์แต่ละไฟล์แล้วเพิ่มโค้ดเข้าไป ตามลำดับ ดังนี้

สำหรับไฟล์ index.blade.php (resources/views/users/index.blade.php)

```
@extends('layouts.frontend')
@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left"><h2>จัดการสมาชิก</h2>
            </div>
            <div class="pull-right">a class="btn btn-success" href="{{ route('users.create') }}"> สร้างสมาชิกใหม่ </a>
            </div>
        </div>
    </div>
    @if ($message = Session::get('success'))
        <div class="alert alert-success">
            <p>{{ $message }}</p>
        </div>
    @endif
    <table class="table table-bordered">
        <tr>
            <th>ลำดับ</th>
            <th>ชื่อ</th>
            <th>อีเมล</th>
            <th>บทบาทหน้าที่</th>
            <th width="280px">จัดการ</th>
        </tr>
        @foreach ($data as $key => $user)
            <tr>
                <td>{{ ++$i }}</td>
                <td>{{ $user->name }}</td>
                <td>{{ $user->email }}</td>
                <td>
                    @if (!empty($user->getRoleNames()))
                        @foreach ($user->getRoleNames() as $v)
                            <label class="badge badge-success">{{ $v }}</label>
                        @endforeach
                    @endif
                </td>
                <td>
                    <a class="btn btn-info" href="{{ route('users.show', $user->id) }}">ดู</a>
                    <a class="btn btn-primary" href="{{ route('users.edit', $user->id) }}">แก้ไข</a>
                    {!! Form::open(['method' => 'DELETE', 'route' => ['users.destroy', $user->id], 'style' => 'display:inline']) !!}
                    {!! Form::submit('ลบ', ['class' => 'btn btn-danger']) !!}
                    {!! Form::close() !!}
                </td>
            </tr>
        @endforeach
    </table>
    {!! $data->links() !!}
@endsection
```

สำหรับไฟล์ create.blade.php (resources/views/users/create.blade.php)

```
@extends('layouts.frontend')
@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb"><div class="pull-left"><h2>เพิ่มสมาชิกใหม่</h2>div>
            <div class="pull-right"><a class="btn btn-primary" href="{{ route('users.index') }}"> กลับ </a>div>
        </div>
    </div>
    @if (count($errors) > 0)
        <div class="alert alert-danger">
            <strong>Whoops!</strong>Something went wrong.<br><br>
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif
    {!! Form::open(['route' => 'users.store', 'method' => 'POST']) !!}
    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group"><strong>ชื่อ:</strong>
                {!! Form::text('name', null, ['placeholder' => 'Name', 'class' => 'form-control']) !!}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group"><strong>อีเมล:</strong>
                {!! Form::text('email', null, ['placeholder' => 'Email', 'class' => 'form-control']) !!}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group"><strong>รหัสผ่าน:</strong>
                {!! Form::password('password', ['placeholder' => 'Password', 'class' => 'form-control']) !!}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group"><strong>ยืนยันรหัสผ่าน:</strong>
                {!! Form::password('confirm-password', ['placeholder' => 'Confirm Password', 'class' => 'form-control']) !!}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group"><strong>เลือกบทบาท:</strong>
                {!! Form::select('roles[]', $roles, [], ['class' => 'form-control', 'multiple']) !!}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12 text-center">
            <button type="submit" class="btn btn-primary">บันทึก</button>
        </div>
    </div>
    {!! Form::close() !!}
@endsection
```

สำหรับไฟล์ edit.blade.php (resources/views/users/edit.blade.php)

```
@extends('layouts.frontend')
@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb"><div class="pull-left"><h2>แก้ไขสมาชิก</h2></div>
            <div class="pull-right"> <a class="btn btn-primary" href="{{ route('users.index') }}"> กลับ </a></div>
        </div>
    </div>
    @if (count($errors) > 0)
        <div class="alert alert-danger">
            <strong>Whoops!</strong> Something went wrong.<br><br>
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif
    {!! Form::model($user, ['method' => 'PATCH', 'route' => ['users.update', $user->id]]) !!}
    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">strong>ชื่อ:</strong>
                {!! Form::text('name', null, ['placeholder' => 'Name', 'class' => 'form-control']) !!}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group"><strong>อีเมล:</strong>
                {!! Form::text('email', null, ['placeholder' => 'Email', 'class' => 'form-control']) !!}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group"><strong>รหัสผ่าน:</strong>
                {!! Form::password('password', ['placeholder' => 'Password', 'class' => 'form-control']) !!}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">strong>ยืนยันรหัสผ่าน:</strong>
                {!! Form::password('confirm-password', ['placeholder' => 'Confirm Password', 'class' => 'form-control']) !!}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group"><strong>เลือกบทบาท:</strong>
                {!! Form::select('roles[]', $roles, $userRole, ['class' => 'form-control', 'multiple']) !!}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12 text-center">
            <button type="submit" class="btn btn-primary">บันทึก</button>
        </div>
    </div>
    {!! Form::close() !!}
@endsection
```

สำหรับไฟล์ show.blade.php (resources/views/users/show.blade.php)

```
@extends('layouts.frontend')
@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2>รายละเอียด</h2>
            </div>
            <div class="pull-right">
                <a class="btn btn-primary" href="{{ route('users.index') }}"> กลับ </a>
            </div>
        </div>
    </div>
    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>ชื่อ:</strong>
                {{ $user->name }}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>อีเมล:</strong>
                {{ $user->email }}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>บทบาท:</strong>
                @if (!empty($user->getRoleNames()))
                    @foreach ($user->getRoleNames() as $v)
                        <label class="badge badge-success">{{ $v }}</label>
                    @endforeach
                @endif
            </div>
        </div>
    </div>
@endsection
```

11.เปิดไฟล์ index.blade.php ใน resources/views/product/... และแก้ไขโค้ด ดังนี้

```
@extends('layouts.frontend')
@section('title', 'รายการสินค้าทั้งหมด')
@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">h2>สินค้า</h2></div>
            <div class="pull-right">
                @can('product-create')
                    <a class="btn btn-success" href="{{ route('products.create') }}"> เพิ่มสินค้าใหม่ </a>
                @endcan
            </div>
        </div>
    </div>
    <section class="py-5 container">
        <div class="row">
            <div class="col-lg-12  mx-auto">
                <table class="table table-hover">
                    <thead>
                        <tr>
                            <th scope="col">ลำดับ</th>
                            <th scope="col">ภาพสินค้า</th>
                            <th scope="col">ชื่อสินค้า</th>
                            <th scope="col">ราคา</th>
                            <th scope="col" colspan="2">จัดการ</th>
                        </tr>
                    </thead>
                    @foreach ($products as $product)
                        <tbody>
                            <tr>
                                <th scope="col">{{ $product->id }}</th>
                                <th scope="col"><img src="{{ asset('uploads/resize/' . $product->image) }}"
                                        width="50px" /></th>
                                <td scope="col">{{ $product->name }}</td>
                                <td scope="col" style="width: 50%">{{ $product->price }}</td>
                                <td scope="col">
                                    @can('product-edit')
                                        <a class="btn btn-primary" href="{{ route('products.edit', $product->id) }}">แก้ไข</a>
                                    @endcan
                                </td>
                                <td scope="col">
                                    @can('product-delete')
                                        {!! Form::open(['route' => ['products.destroy', $product->id],'method' => 'DELETE',]) !!}
                                        {!! Form::submit('ลบข้อมูล', ['class' => 'btn btn-danger btn-sm','onclick' => "return confirm('ยืนยันการลบหรือไม่')",
                                        ]) !!}
                                        {!! Form::close() !!}
                                    @endcan
```

```
// resources/views/product/index.blade.php
                        </td>
                    </tr>
                </tbody>
            @endforeach
        </table>
        {{ $products->links() }}
    </div>
</div>
</section>
@endsection
```

จากที่ผ่านนั้นเราได้ทำการสร้างไฟล์ต่าง ๆ เรียบร้อยทุกอย่างแล้ว เราจะเห็นว่าในไฟล์ index.blade.php แต่ละไฟล์ที่ถูกสร้างขึ้นไว้ใน โฟลเดอร์ roles , users , product มีการกำหนดสิทธิ์การเข้าถึงข้อมูลแต่ละส่วนไว้แล้ว ซึ่งการกำหนดสิทธิ์นั้นจะใช้คำสั่ง ดังนี้

```
// resources/views/roles/index.blade.php
@can('ระบุชื่อ permission ที่ต้องการ')
 ...
@endcan
```

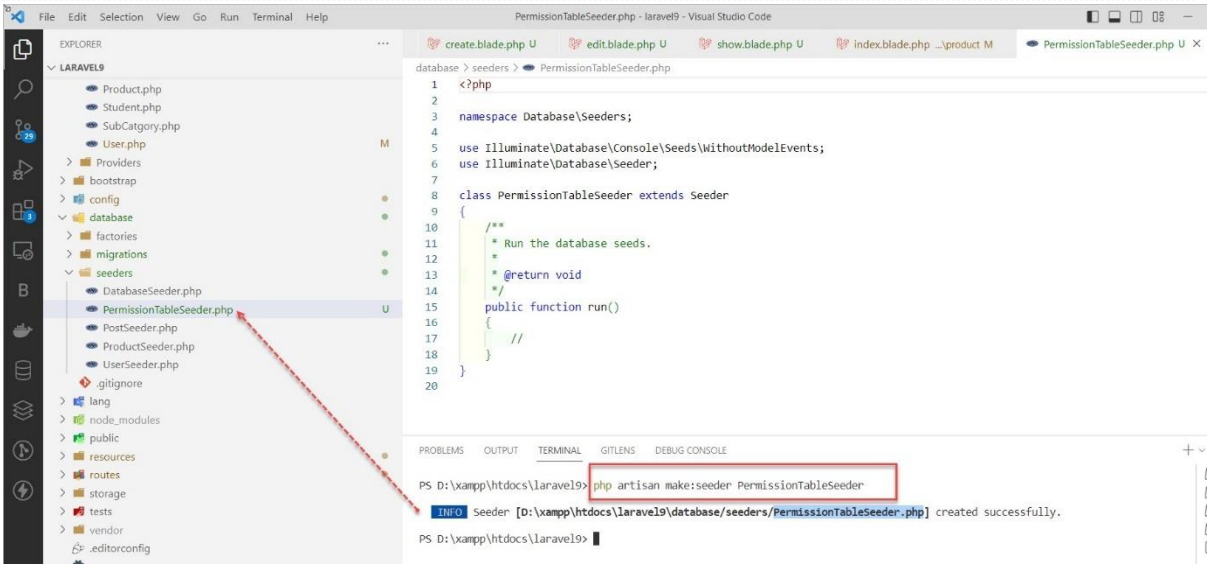
@can จะกรองข้อมูลให้เราตามสิทธิ์นั้น ๆ ให้เรา เช่น

สิทธิ์การเพิ่มข้อมูลบทบาทใหม่ จะใช้คำสั่ง @can('role-create')
สิทธิ์การเพิ่มข้อมูลสมาชิกใหม่ จะใชคำสั่ง @can('users-create')
สิทธิ์การเพิ่มข้อมูลสินค้าใหม่ จะใช้คำสั่ง @can('product-create')
สิทธิ์การแก้ไขข้อมูลบทบาท จะใช้คำสั่ง @can('role-edit')
สิทธิ์การแก้ไขข้อมูลสมาชิก จะใช้คำสั่ง @can('user-edit')
สิทธิ์การแก้ไขข้อมูลสินค้า จะใช้คำสั่ง @can('product-edit')
สิทธิ์การลบข้อมูลบทบาท จะใช้คำสั่ง @can('role-delete')
สิทธิ์การลบข้อมูลสมาชิก จะใช้คำสั่ง @can('user-delete')
สิทธิ์การลบข้อมูลสินค้า จะใช้คำสั่ง @can('product-delete')

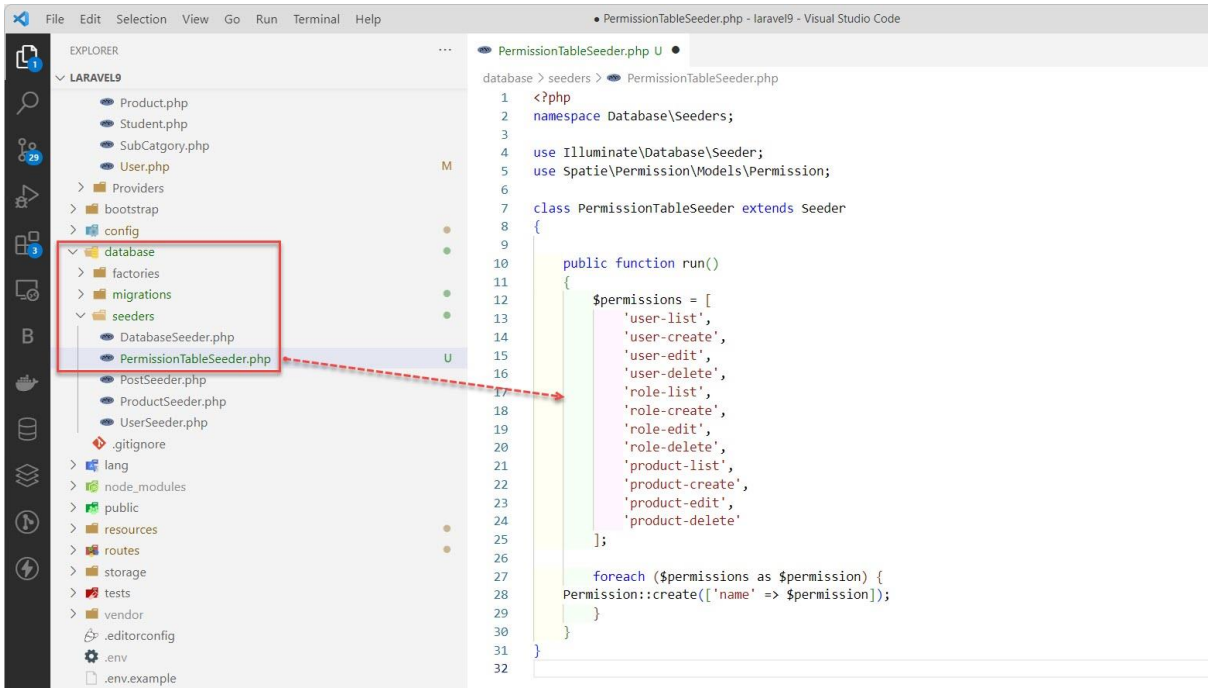
โดยสิทธิ์แต่ละสิทธิ์นั้น จะถูกจัดการโดยผู้ดูแลระบบเป็นคนมอบหมายว่าจะให้แต่ละ user จัดการอะไรได้บ้าง

12.สร้าง seeder เพื่อเพิ่มสิทธิ์ permission เก็บไว้ในฐานข้อมูล โดยรันคำสั่งผ่าน Terminal ดังนี้

```
php artisan make:seeder PermissionTableSeeder
```



เปิดไฟล์ PermissionTableSeeder.php ในโฟลเดอร์ database/seeds/... แล้วเพิ่มโค้ด ดังนี้

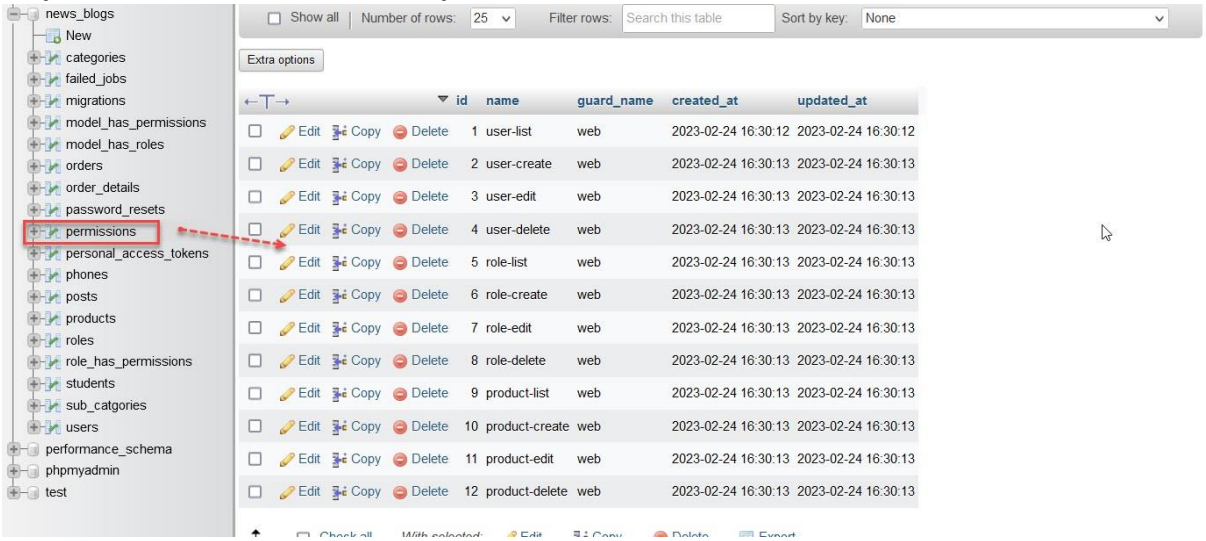


รันคำสั่ง seed ผ่าน Terminal เพื่อให้ระบบเพิ่มข้อมูลตั้งต้นเก็บไว้ในฐานข้อมูล

```
php artisan db:seed --class=PermissionTableSeeder
```

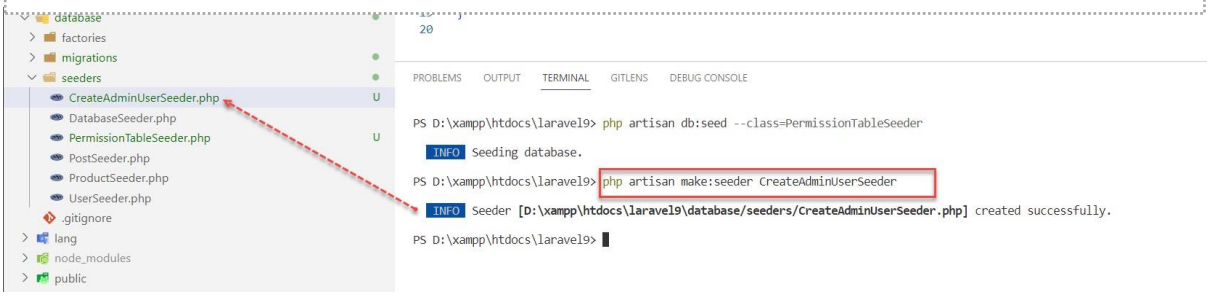


เปิดโปรแกรม phpMyAdmin แล้วตรวจสอบตาราง permission ว่าข้อมูลถูกเพิ่มหรือไม่ หากมีข้อมูลในตารางตามภาพแสดงว่าท่านมาถูกทางแล้ว ไปลุยขั้นตอนต่อไปเลยครับ

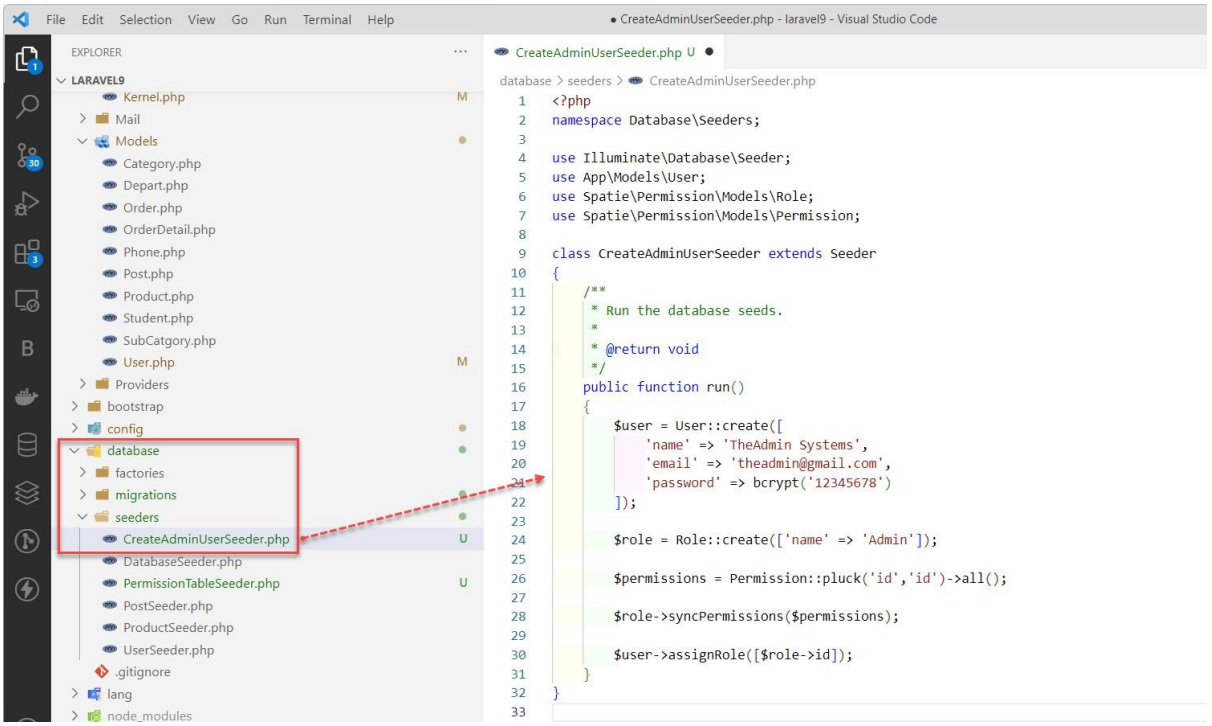


13.สร้าง seeder สำหรับ Admin เก็บไว้ในฐานข้อมูล โดยรันคำสั่งผ่าน Terminal ดังนี้

```
php artisan make:seeder CreateAdminUserSeeder
```



เปิดไฟล์ CreateAdminUserSeeder.php ในโฟลเดอร์ database/seeds/... แล้วเพิ่มโค้ด ดังนี้



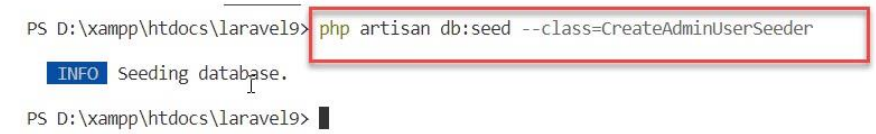
บรรทัด 18-21 เป็นการกำหนดผู้ใช้งานตั้งต้นให้กับระบบเพิ่มอีก 1 user

บรรทัด 24 เป็นการสร้างข้อมูลตั้งต้นให้กับตาราง role โดยเพิ่มบทบาทเป็น Admin

บรรทัด 28 – 30 เป็นการมอบสิทธิ์ตั้งต้นให้กับ user โดยกำหนดสิทธิ์ให้สามารถจัดการข้อมูลทุกอย่างได้ จะใช้คำสั่ง syncPermissions() และ assignRole()

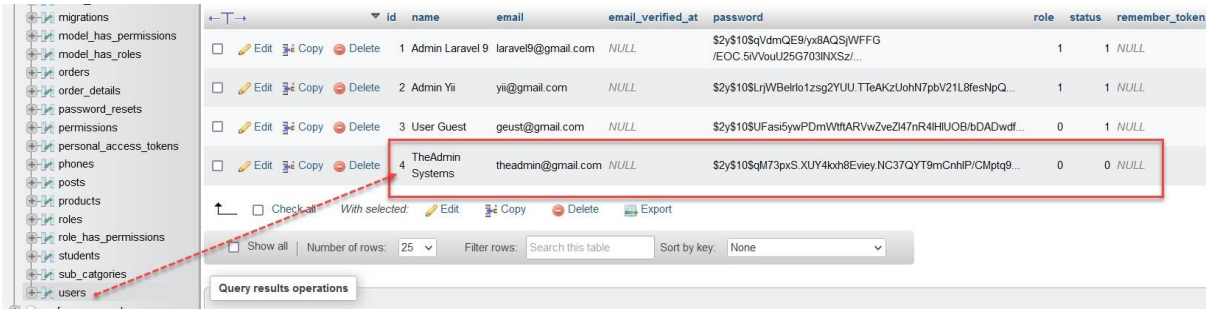
รันคำสั่ง seed ผ่าน Terminal เพื่อให้ระบบเพิ่มข้อมูลตั้งต้นเก็บไว้ในฐานข้อมูล

```
php artisan db:seed --class=CreateAdminUserSeeder
```

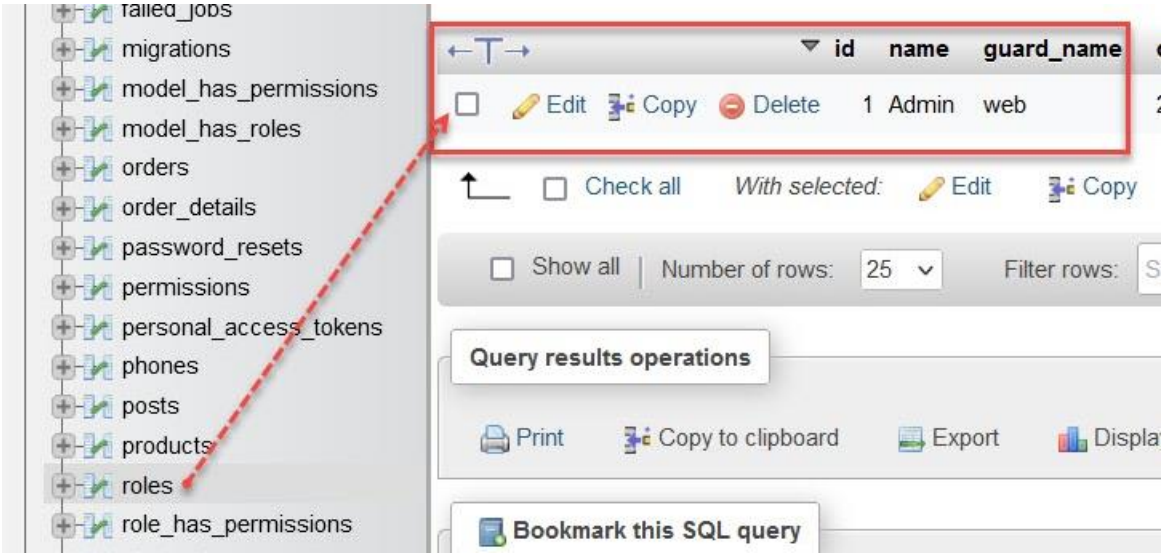


เปิดโปรแกรม phpMyAdmin แล้วตรวจสอบตาราง users , roles , role_has_permissions ว่าข้อมูลถูกเพิ่มหรือไม่

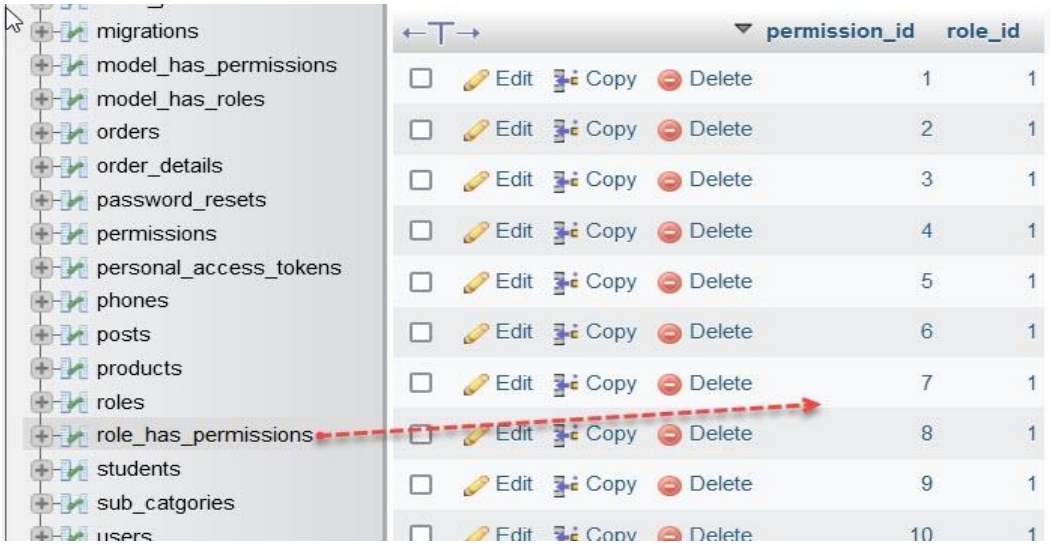
ตาราง users



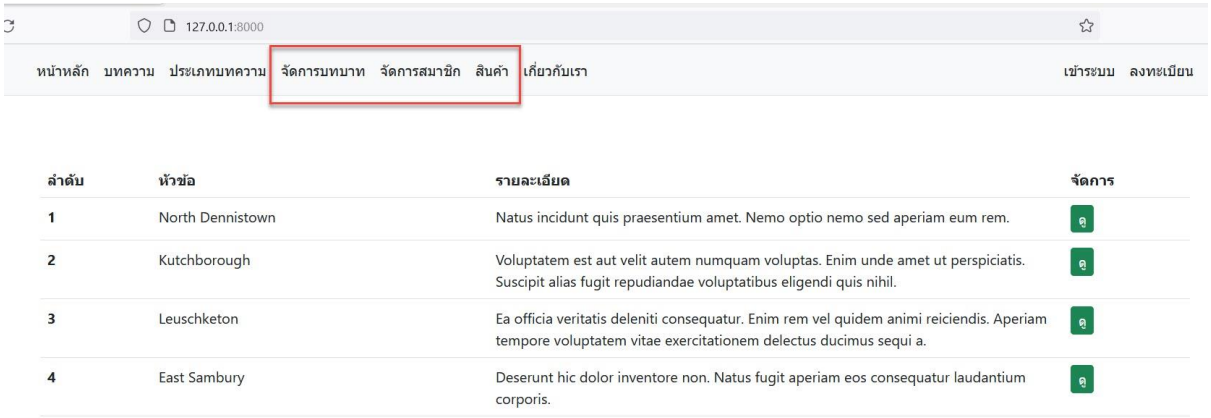
ตาราง roles



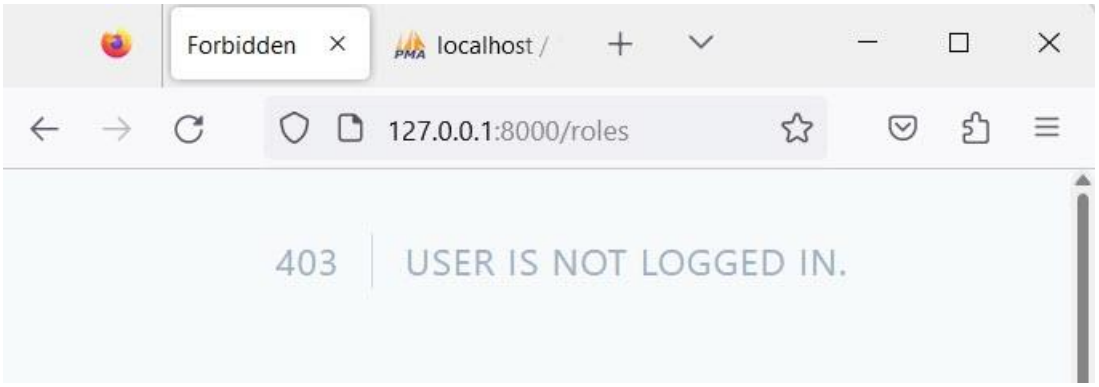
ตาราง role_has_permissions



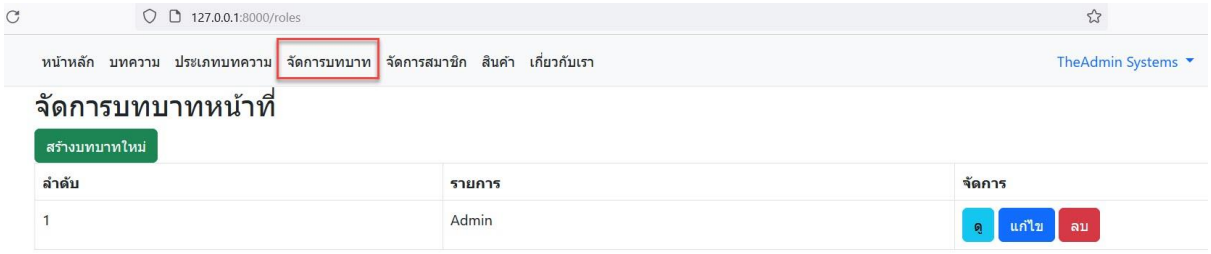
บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/ ที่ Browser (อย่าลืม start server ด้วยนะ php artisan serv)



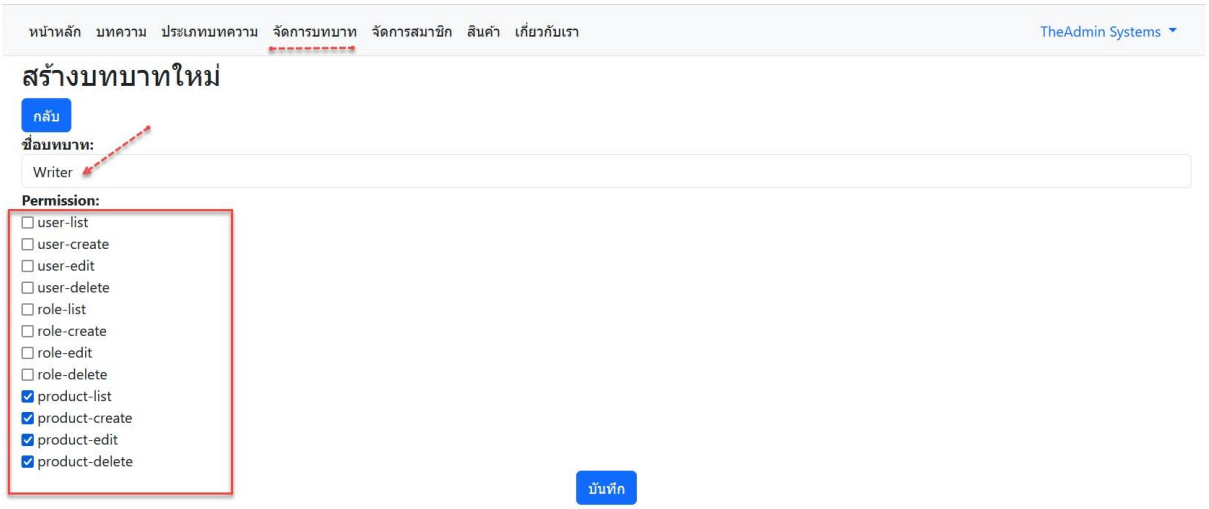
ให้เราโฟกัสที่เมนูจัดการบทบาท จัดการสมาชิก และ สินค้า เมื่อคลิกที่ลิงค์ จัดการบทบาทหากไม่ได้ล็อกอินเข้าระบบเราก็จะไม่สามารถทำรายการใด ๆ ได้เลย จะขึ้นหน้า Error 403 ดังภาพ



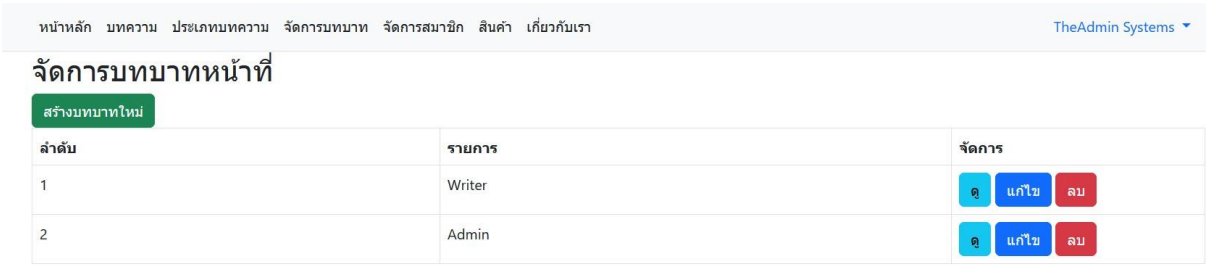
ฉะนั้นต้องล็อกอินเข้าระบบก่อน โดยก่อนหน้าเราได้สร้างผู้ใช้งานขึ้นมาใหม่นั่นคือ theadmin@gmail.com รหัสผ่าน 12345678 ให้คลิกที่เมนูเข้าระบบแล้วกรอก email และ password ดังกล่าว เมื่อเข้าระบบแล้วให้คลิกที่เมนูจัดการบทบาท เราจะพบว่ามีรายการอยู่ 1 รายการ คือ Admin



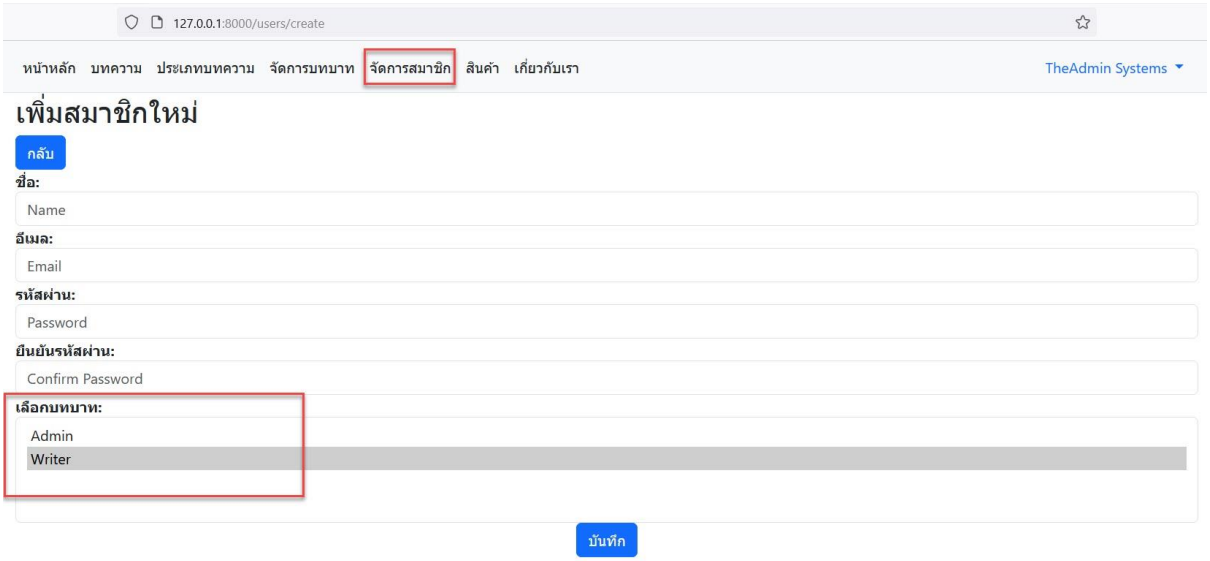
ให้ทดสอบเพิ่มบทบาทใหม่โดยคลิกที่ปุ่มสร้างบทบาทใหม่ โดยในตัวอย่างจะทดลองเพิ่มบทบาทเพิ่มเข้ามาอีก 1 รายการ โดยตั้งชื่อว่า writer และกำหนดสิทธิ์การเข้าใช้งานโดยจะกำหนดให้สามารถจัดการข้อมูลสินค้าได้อย่างเดียว



จากภาพเป็นการเพิ่มบทบาทใหม่ชื่อ Writer และมีการกำหนดสิทธิ์การเข้าถึงข้อมูลให้สามารถดูข้อมูลสินค้าได้ เพิ่มสินค้าได้ แก้ไขสินค้าได้ และลบสินค้าได้ เท่านั้น



จากนั้นให้คลิกที่เมนูจัดการสมาชิก แล้วคลิกที่ปุ่มเพิ่มสมาชิกใหม่



จะเห็นว่ามีบทบาทใหม่ให้เลือกเพิ่ม ให้ผู้อ่านทดลองเพิ่มแล้วล็อกอินเข้าระบบดูในแต่ละส่วนนะครับ

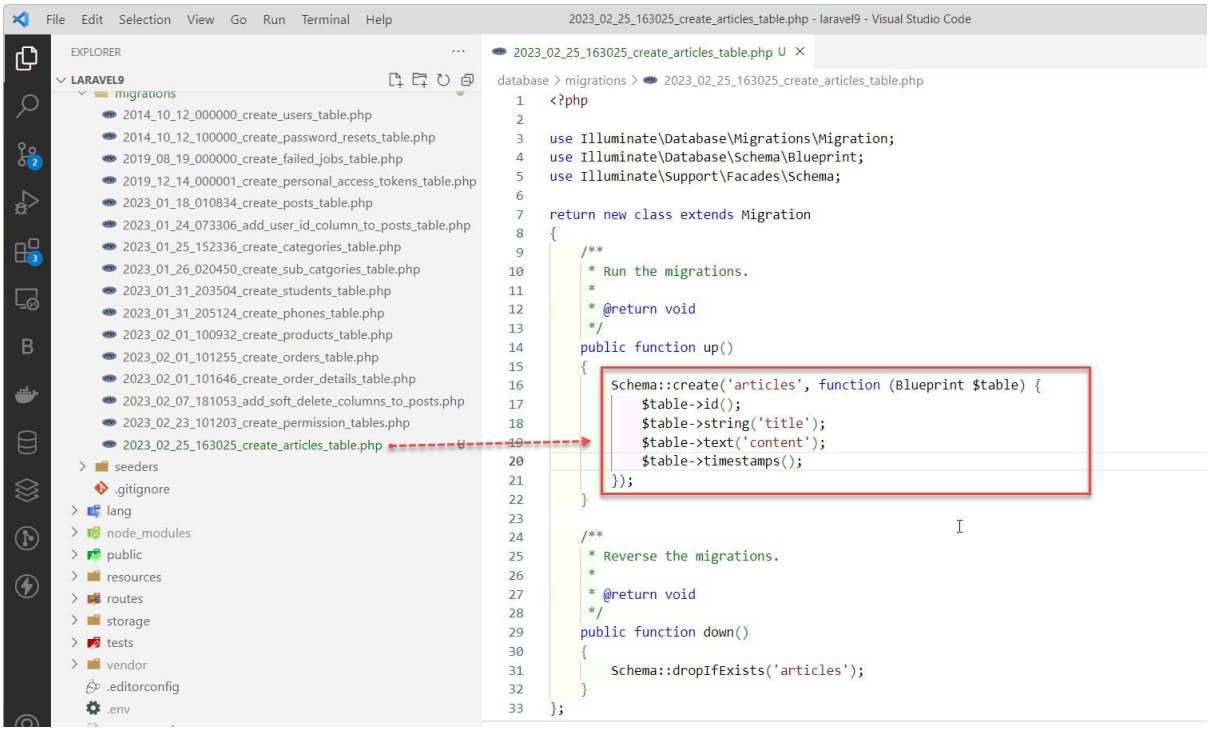
สร้าง API (Backend) และ เรียกใช้งาน API (Frontend)

API ย่อมาจาก Application Programming Interface เป็นการเชื่อมต่อจากระบบหนึ่งไปสู่อีกระบบหนึ่ง เพื่อให้ซอฟต์แวร์ภายนอกเข้าถึงและอัพเดทข้อมูลนั้นๆได้ API เป็นตัวกลางที่จะทำให้คอยรับคำสั่งต่าง ๆ ประมวลผลและกระทำข้อมูลส่งกลับคืนไปยังคนสั่งโดยอัตโนมัติ โดยจะมีภาษากลางในการสื่อสารระหว่างสองฝั่งคือ **JSON** ใน Laravel จะมีวิธีการสร้าง API ที่ง่ายมาก ในหนังสือเล่มนี้จะยกตัวอย่างการสร้าง API สำหรับการอัปโหลดรูปภาพหลาย ๆ ภาพและเนื้อหา โดยใช้ Spatie Media Library เพื่อให้ผู้อ่านเข้าใจหลักการวิธีการสร้างและการนำไปใช้งานในฝั่ง Frontend มาดูวิธีการสร้างตามขั้นตอนดังนี้

1.เปิด Terminal แล้วพิมพ์คำสั่งสำหรับสร้าง migration และ โมเดลใหม่ชื่อว่า Article ด้วยคำสั่ง นี้

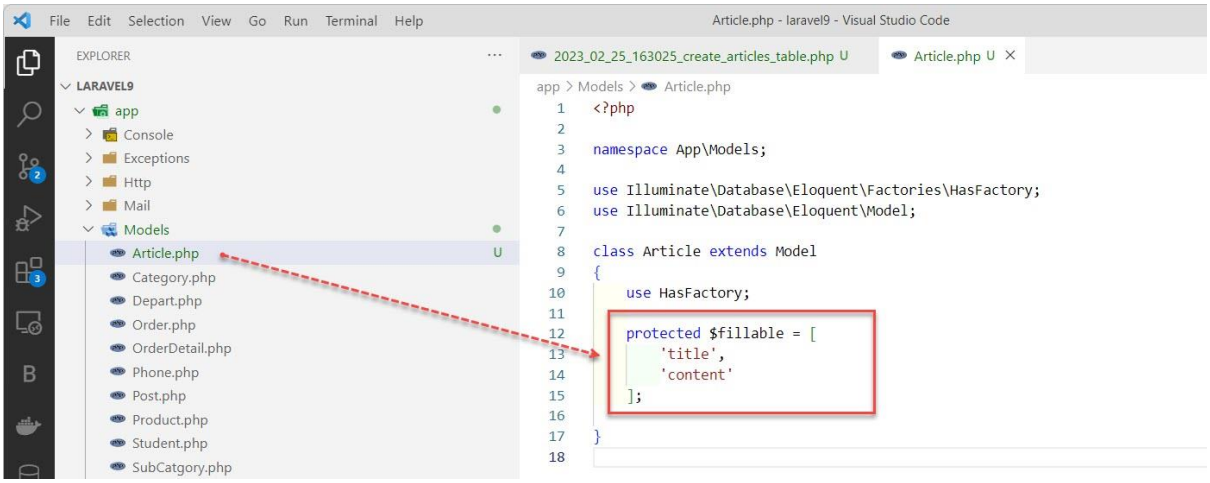
```
php artisan make:model Article -m
```

xxx_create_articles_table.php สำหรับกำหนดโครงสร้างตาราง articles ให้เปิดไฟล์ create_articles_table.php แล้วกำหนดโครงสร้างตาราง ดังนี้



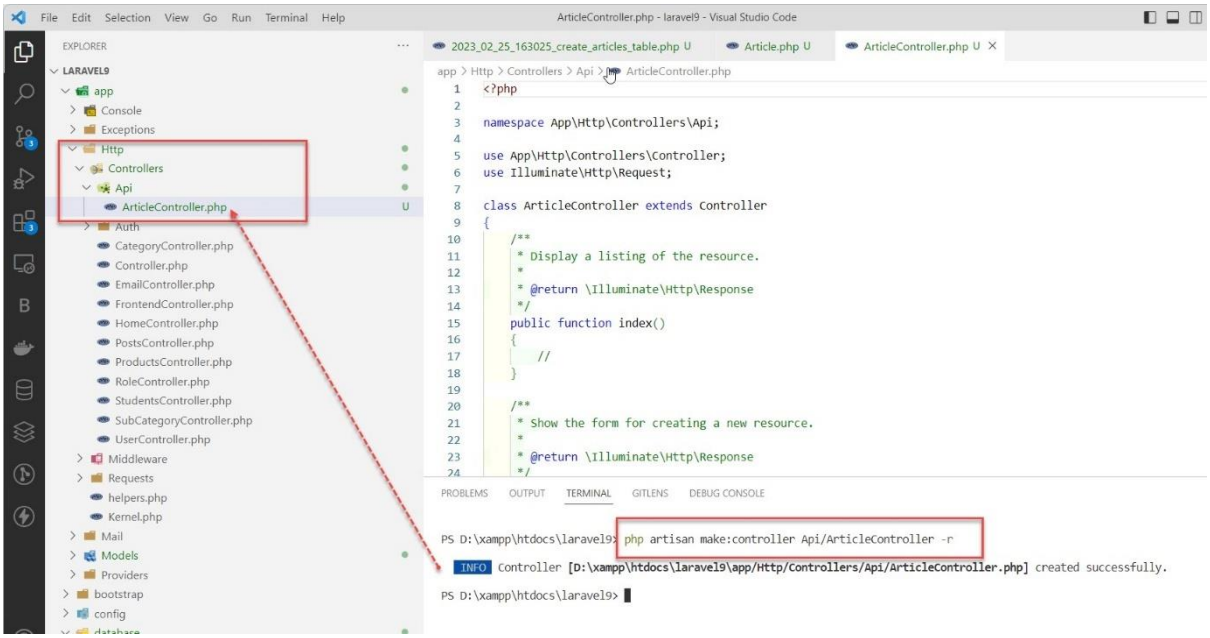
ในตัวอย่างโค้ดที่เขียนในไฟล์โมเดล Article.php เป็นการกำหนดโครงสร้างตารางพื้นฐานสำหรับจัดเก็บข้อมูลประกอบด้วยฟิวด์ title และ content

2.เปิดไฟล์โมเดล Article.php ในโฟลเดอร์ app/Models/... แล้วกำหนด Mass Assignment เพิ่มโค้ดเข้าไปดังนี้



3.สร้าง API Controllers แบบ RESTful API ชื่อว่า ArticleController.php

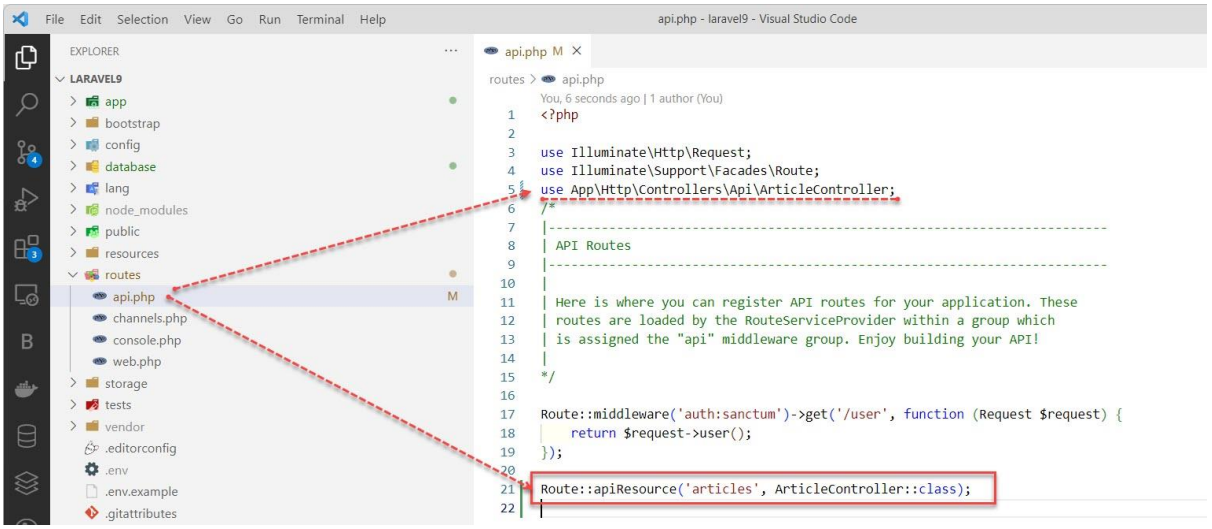
```
php artisan make:controller Api/ArticleController -r
```



เมื่อรันคำสั่งเรียบร้อยจะมีไฟล์ ArticleController.php ถูกสร้างขึ้นเก็บไว้ในโฟลเดอร์ app/Http/Controllers/Api/...

4.กำหนด routes api สำหรับการเข้าถึง โดยปกติเราสามารถกำหนดเส้นทางการเข้าถึงได้ในไฟล์ web.php แต่เพื่อเป็นการแยกส่วนการทำงานและทำให้จัดการได้สะดวกใน Laravel จะมีไฟล์สำหรับเขียน Api โดยเฉพาะเก็บไว้ในไฟล์ api.php ที่อยู่ในโฟลเดอร์ routes/api.php
ให้เปิดไฟล์ api.php ขึ้นมาแล้วเพิ่ม routes เข้าไปดังนี้

ใน Laravel 11 จะยังไม่มีไฟล์ api.php มาให้ ให้ท่านติดตั้งเพิ่ม ผ่าน Terminal ด้วยคำสั่งนี้
php artisan install:api



5.ติดตั้ง media Library ผ่าน Terminal ด้วยคำสั่งนี้

```
composer require "spatie/laravel-medialibrary:^10.0.0"
```

```
INFO  Controller [D:\xampp\htdocs\laravel9\app/Http/Controllers/Api/ArticleController.php] created successfully.

PS D:\xampp\htdocs\laravel9> composer require "spatie/laravel-medialibrary:^10.0.0"
Info from https://repo.packagist.org: #StandWithUkraine
./composer.json has been updated
Running composer update spatie/laravel-medialibrary
Loading composer repositories with package information
Updating dependencies
Lock file operations: 38 installs, 0 updates, 0 removals
  - Locking composer/ca-bundle (1.3.5)
  - Locking composer/class-map-generator (1.0.0)
  - Locking composer/composer (2.5.4)
  - Locking composer/metadata-minifier (1.0.0)
  - Locking composer/pcre (3.1.0)
  - Locking composer/semver (3.3.2)
  - Locking composer/spdx-licenses (1.5.7)
  - Locking composer/xdebug-handler (3.0.3)
  - Locking justinrainbow/json-schema (5.2.12)
  - Locking league/glide (2.2.3)
  - Locking maennchen/zipstream-php (v2.4.0)
  - Locking myclabs/php-enum (1.8.4)
  - Locking nunomaduro/larastan (1.0.4)
  - Locking orchestra/testbench (v7.20.0)
  - Locking orchestra/testbench-core (v7.22.1)
  - Locking phpstan/extension-installer (1.2.0)
  - Locking phpstan/phpstan (1.8.11)
  - Locking pimple/pimple (v3.5.0)
  - Locking react/promise (v2.9.0)
  - Locking seld/jsonlint (1.9.0)
```

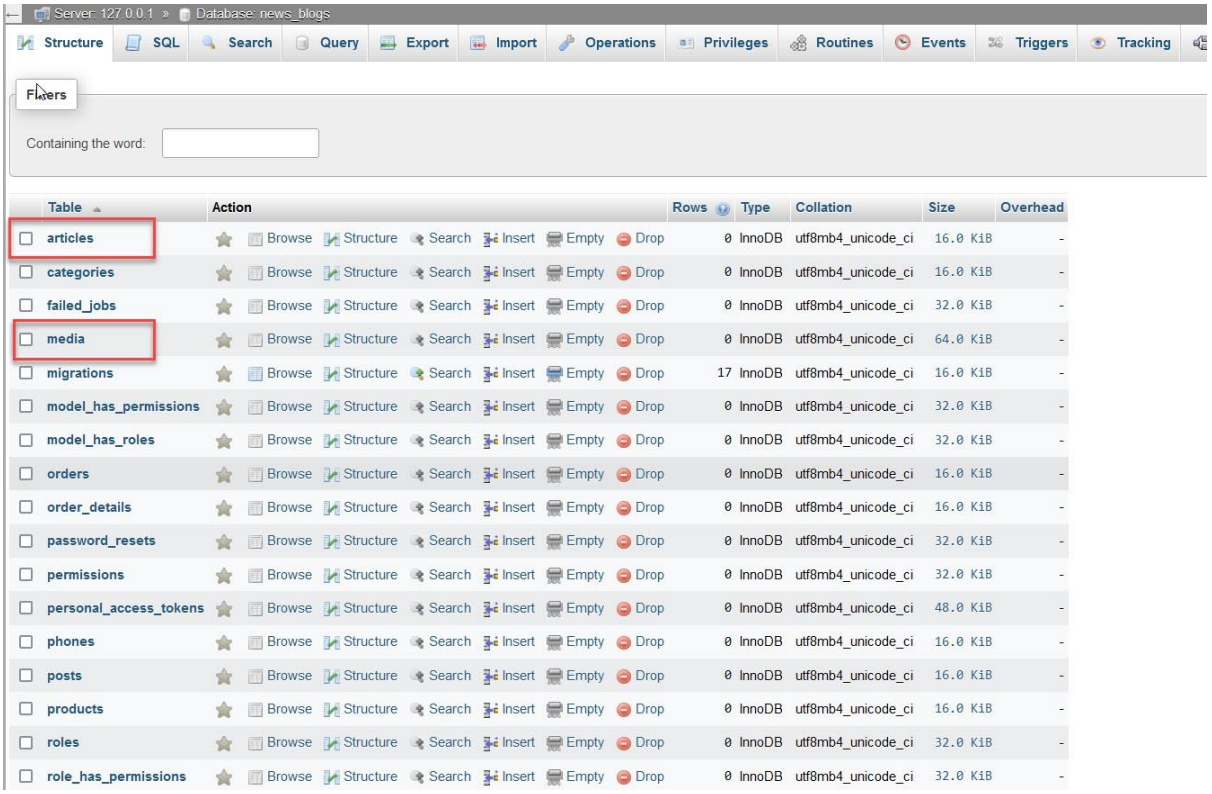
จากนั้นให้สร้าง Migrate เพื่อกำหนดตาราง media ด้วยคำสั่ง

```
php artisan vendor:publish --provider="Spatie\MediaLibrary\MediaLibraryServiceProvider" --tag="migrations"
```

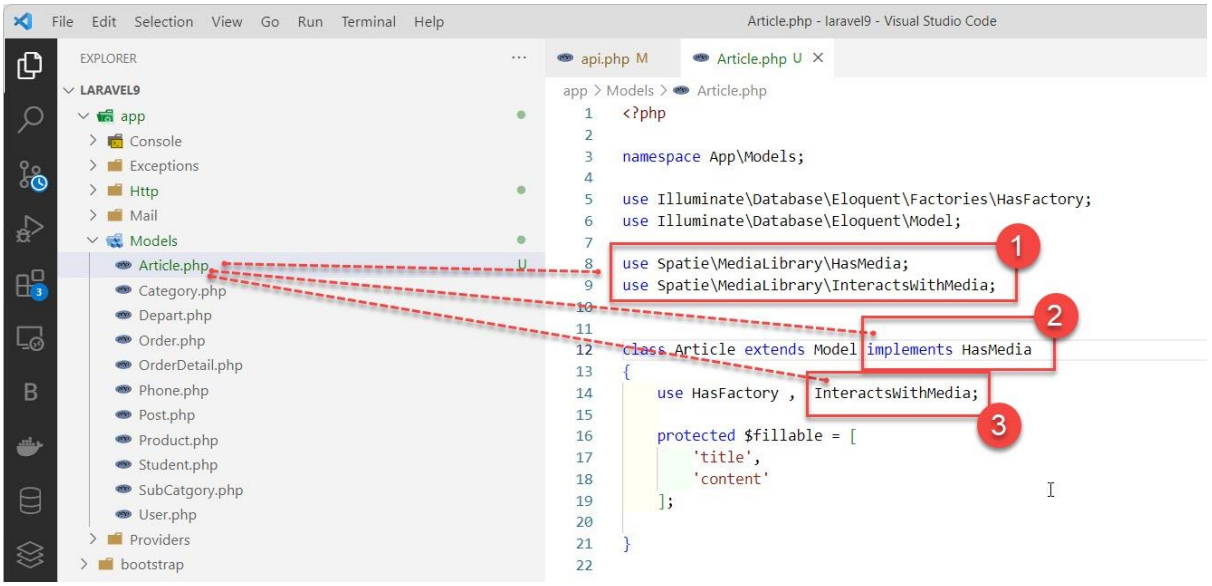
สั่ง migrate:refresh และ seed ข้อมูลใหม่ผ่าน Terminal

```
php artisan migrate:fresh
php artisan db:seed
```

ตรวจสอบโครงสร้างตารางใน phpMyAdmin จะเห็นว่ามีตาราง articles และ media ถูกสร้างขึ้นมาให้แล้ว



6.เปิดไฟล์โมเดล Article.php ในโฟลเดอร์ app/Models/... แล้วเพิ่มโค้ดเข้าไปดังนี้



เพิ่ม class Laravel-medialibrary (HasMedia, InteractsWithMedia)

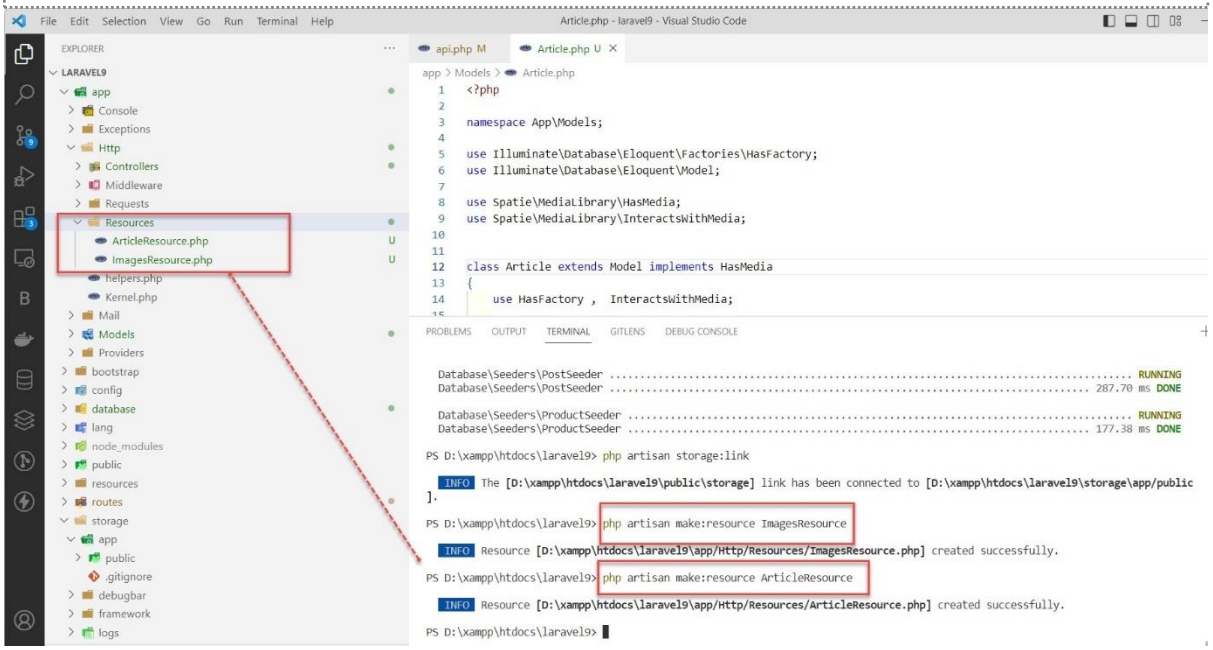
6.รันคำสั่งผ่าน Terminal เพื่อสร้างโฟลเดอร์กำหนดการเชื่อมโยงกับที่เก็บภาพ

```
php artisan storage:link
```

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE

  Database\Seeders\PostSeeder ..... RUNNING
  Database\Seeders\PostSeeder ..... 287.70 ms DONE

  Database\Seeders\ProductSeeder ..... RUNNING
  Database\Seeders\ProductSeeder ..... 177.38 ms DONE

PS D:\xampp\htdocs\laravel9> php artisan storage:link

  INFO  The [D:\xampp\htdocs\laravel9\public\storage] link has been connected to [D:\xampp\htdocs\laravel9\storage\app/public
].

PS D:\xampp\htdocs\laravel9>
```

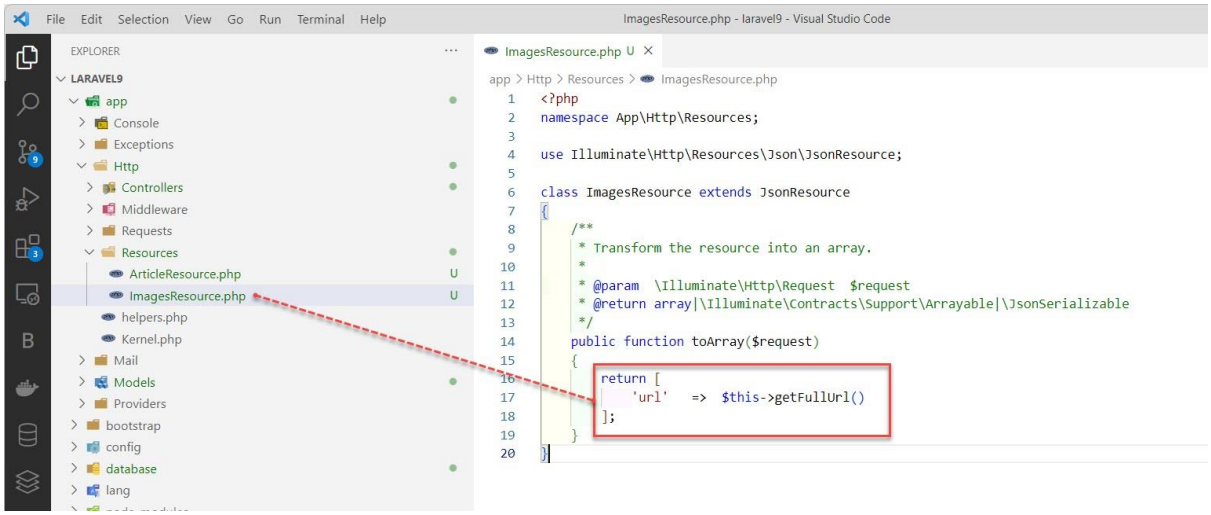
7.สร้างไฟล์ Article Resource และ Article Resource สำหรับการอัปโหลดไฟล์หลาย ๆ ภาพ โดยรันคำสั่งผ่าน Terminal ดังนี้ พิมพ์ทีละคำสั่งแล้วกด Enter

```
php artisan make:resource ImagesResource
php artisan make:resource ArticleResource
```



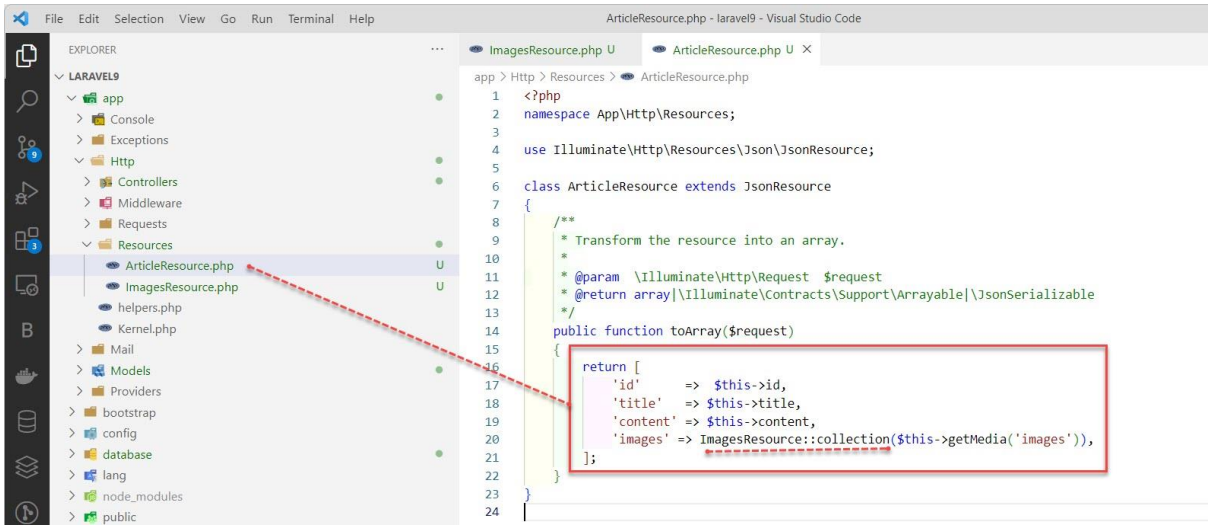
จะมีโฟลเดอร์ resources ถูกสร้างขึ้นภายใต้โฟลเดอร์ app/Http/... รวมถึงไฟล์ ArticleResource.php และ ImagesResource.php ก็จะถูกสร้างขึ้นให้พร้อมกัน

8.เปิดไฟล์ ImagesResource.php ในโฟลเดอร์ app/Http/Resources/... แล้วเพิ่มโค้ดเข้าไปครับ



จากตัวอย่างโค้ดเป็นการกำหนด path ของที่อยู่ภาพ โดยกำหนดให้เป็น path เต็ม ๆ ของภาพ แล้วคืนค่ากลับไปเป็น JSON Array

9.เปิดไฟล์ ArticleResource.php ในโฟลเดอร์ app/Http/Resources/... แล้วเพิ่มโค้ดเข้าไปครับ



จากตัวอย่างโค้ดเป็นการระบุฟิวด์ที่อยู่ในฐานข้อมูลจะสอดคล้องกับฟิวด์ที่กำหนดไว้ในโมเดล Article และมีการกำหนดฟิวด์เพิ่มเติมสำหรับเก็บรูปภาพและเรียกใช้รูปภาพ คือ images จะเห็นว่ามีการเรียกใช้ ImagesResource ซึ่งได้มาจากไฟล์ ImagesResource.php ที่เราเพิ่งเพิ่มเข้าไปก่อนหน้านั้น แล้วคืนค่ากลับไปเป็น JSON Array

10.ทำการเขียนคำสั่งสำหรับสร้าง API ก่อนหน้านั้นในข้อที่ 3 จากหน้าที่ 282 เราได้มีการสร้างไฟล์ API ที่เป็นแบบ RESTful API (CRUD) นั้น เราจะได้ไฟล์ ArticleController.php ในโฟลเดอร์ app/Http/Controllers/Api/...ซึ่งเมื่อเปิดไฟล์ดูก็จะเห็นว่ามีการสร้างเมธอดต่าง ๆ ไว้ให้เรียบร้อยแล้ว ประกอบด้วย

เมธอด index() สำหรับเรียกดูข้อมูลทั้งหมด
เมธอด create() สำหรับเรียกฟอร์มสำหรับการสร้างข้อมูล
เมธอด store() สำหรับบันทึกรายการต่าง ๆ ลงฐานข้อมูล
เมธอด show() สำหรับแสดงผลตามรหัส ID

เมธอด edit() สำหรับเรียกฟอร์มแก้ไขข้อมูล

เมธอด update() สำหรับบันทึกรายการแก้ไขข้อมูล

เมธอด destroy() สำหรับลบข้อมูล

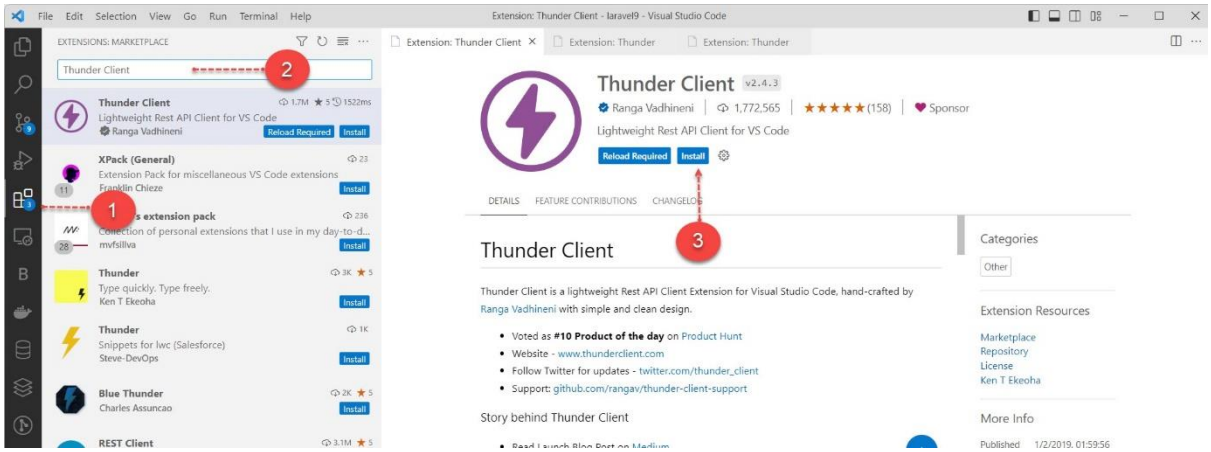
เปิดไฟล์ ArticleController.php แล้วเพิ่มโค้ดเข้าไป ดังนี้

```
//app/Http/Api/ArticleController.php
public function store(Request $request) {
        $this->validate($request, [
            'title' => 'required|string|max:255',
            'content' => 'required',
            'images' => 'nullable|image|mimes:jpg,png,jpeg,gif,svg|max:2048'
            ]);
    $article = Article::create([ 'title' => $request->title,'content' => $request->content]);

    if ($images = $request->images) {
        foreach ($images as $image) {
            $article->addMedia($image)->toMediaCollection('images');
        }
    }
    return ['message' => 'Article created successfully','data' => new ArticleResource($article)
    ];
}
public function show($id) {
    $article = Article::find($id);
    return new ArticleResource($article);
}
public function edit($id) {
}
public function update(Request $request, $id) {
    $this->validate($request, [
        'title' => 'required|string|max:255',
        'content' => 'required',
        'images' => 'nullable|image|mimes:jpg,png,jpeg,gif,svg|max:2048'
    ]);
    $article = Article::find($id);
    $article->title = $request->title;
    $article->content = $request->content;
    $article->save();
    if ($images = $request->images) {
        $article->clearMediaCollection('images');
        foreach ($images as $image) {
            $article->addMedia($image)->toMediaCollection('images');
        }
    }
    return [
        'message' => 'Article updated successfully ',
        'data' => new ArticleResource($article)
    ];
}
```

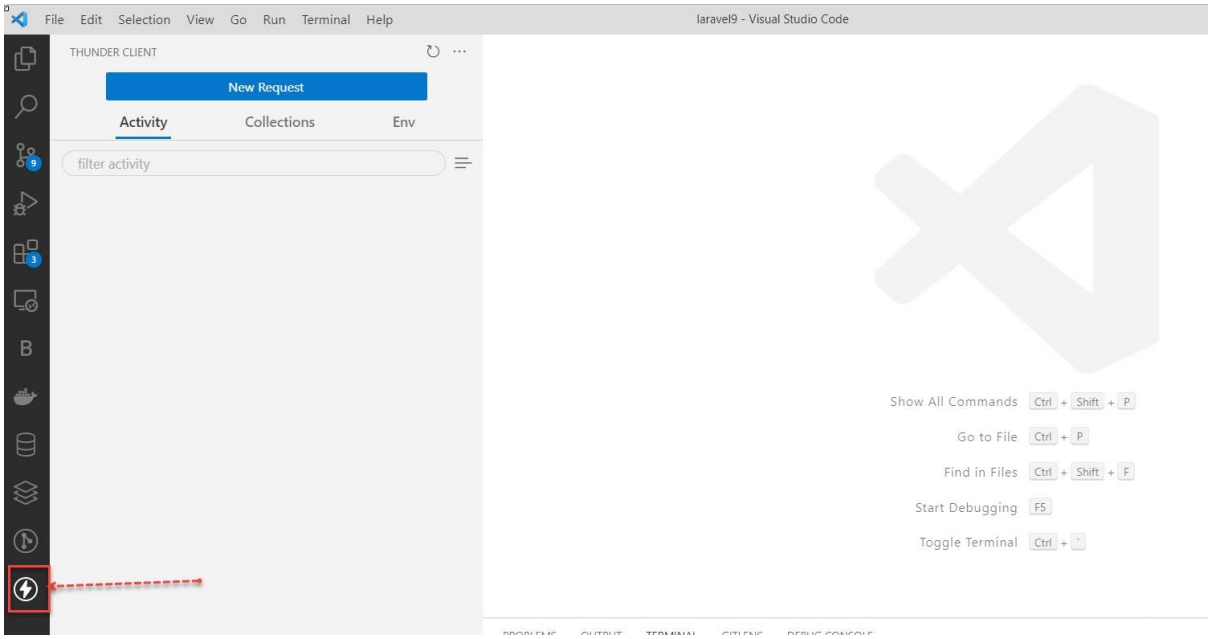
มีโค้ดต่อหน้าถัดไป ...

```
//app/Http/Api/ArticleController.php
  public function destroy($id)

   {
       $article = Article::find($id);
       $article->delete();
       return response(null, 204);
   }
}
```

11.บันทึกไฟล์ทั้งหมดแล้วทดสอบเรียกใช้งาน API โดยการทดสอบนั้น จะมีการติดตั้ง Extensions ของ Ranga Vadhineni แล้วติดตั้งให้เรียบร้อย



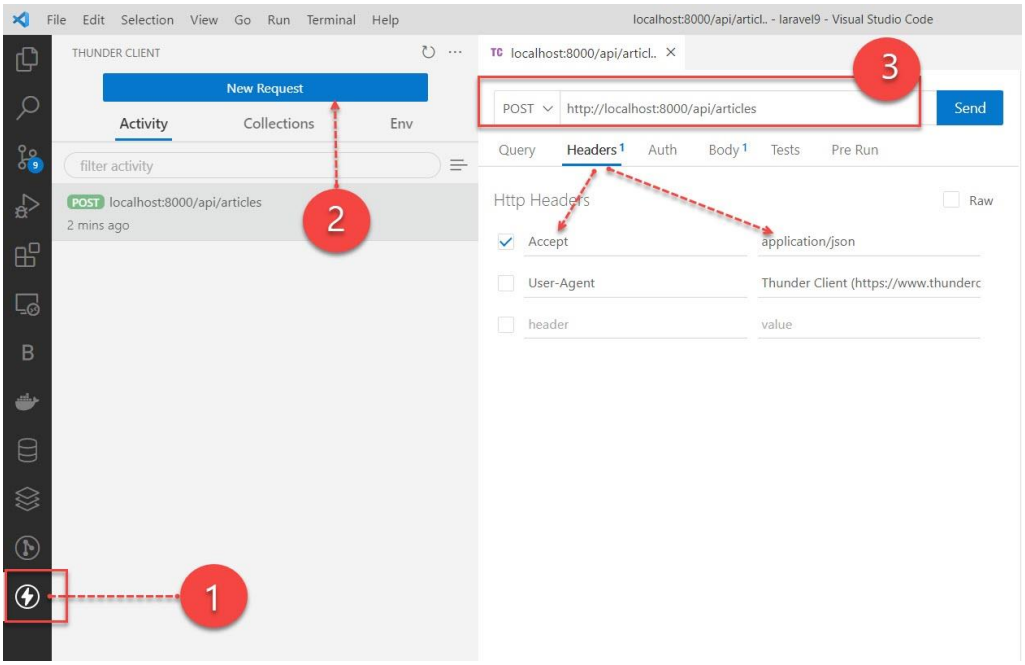
กดปุ่ม install เพื่อติดตั้ง



สัญลักษณ์ Thunder Client จะแสดงตามภาพเมื่อติดตั้งเสร็จเรียบร้อยแล้ว

ให้สั่ง start server ด้วยคำสั่ง php artisan serv ก่อน จากนั้นให้คลิกที่ Thunder Client แล้วสร้าง request ใหม่สำหรับการทดสอบ API ดังนี้

เรียกใช้งาน API ผ่าน URL http://localhost:8000/api/articles

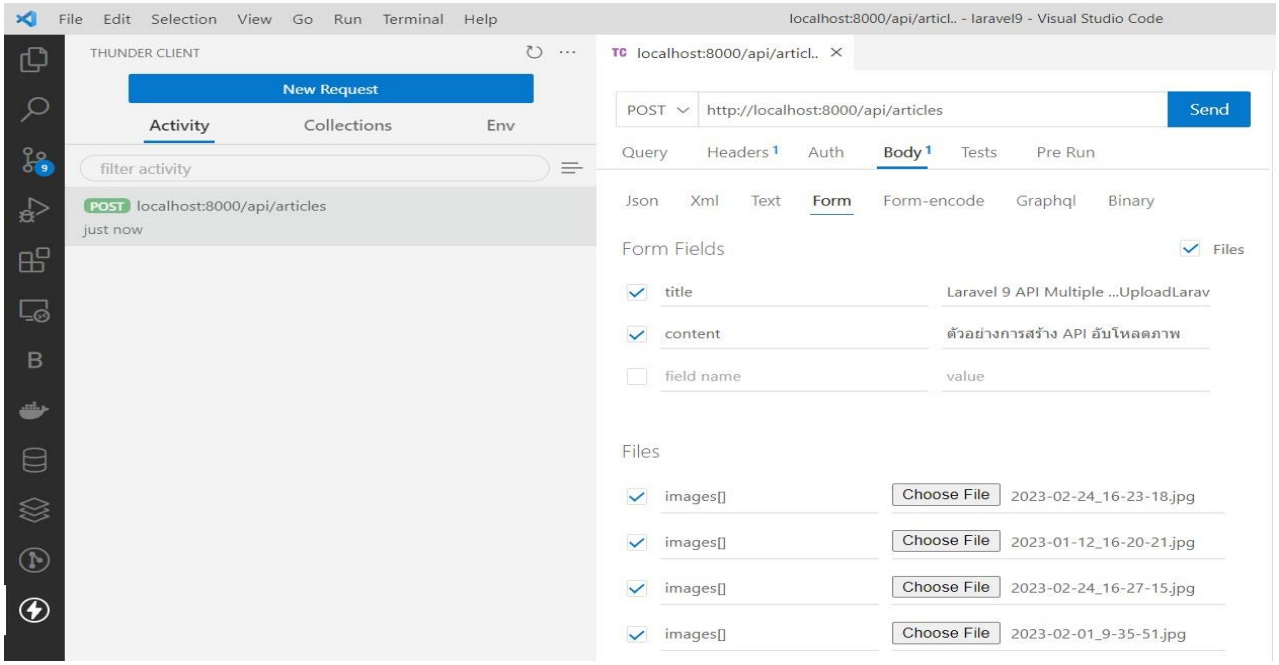


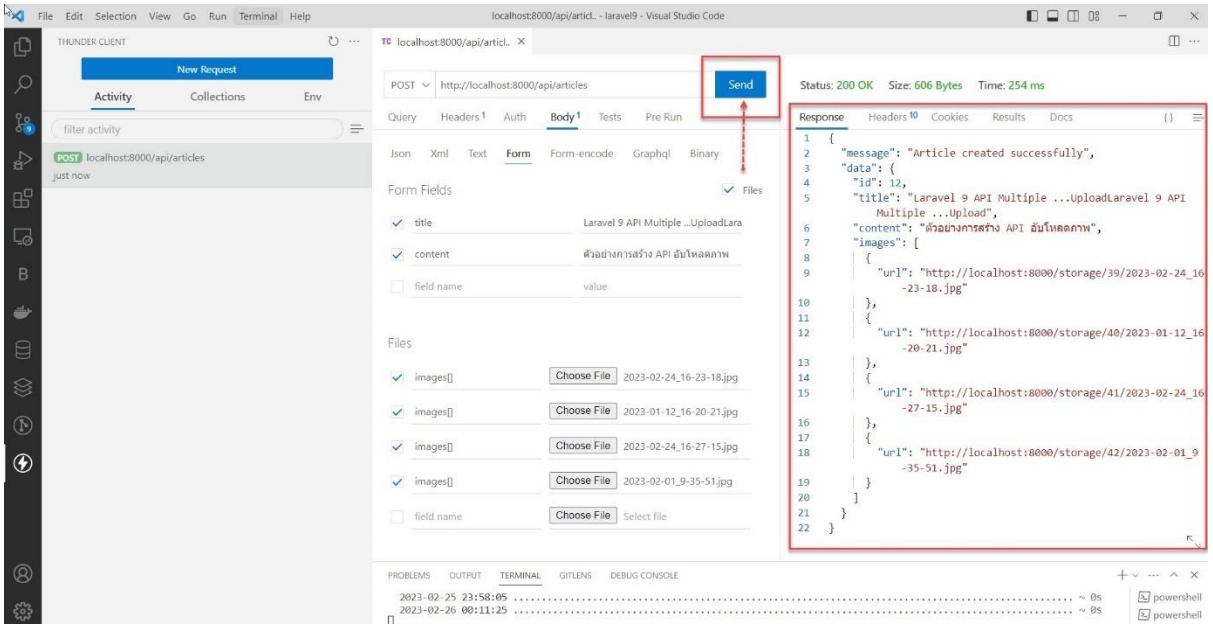
หมายเลข 1 ให้คลิกที่สัญลักษณ์ดังภาพ

หมายเลข 2 คลิกที่ New Request เพื่อสร้าง Request ใหม่

หมายเลข 3 ระบุ URL ของ API แล้วเลือก Method เป็น POST จากนั้นให้ตั้งค่าในส่วนของ Headers ตั้งชื่อว่า Accept แล้วระบุค่าเป็น application/json

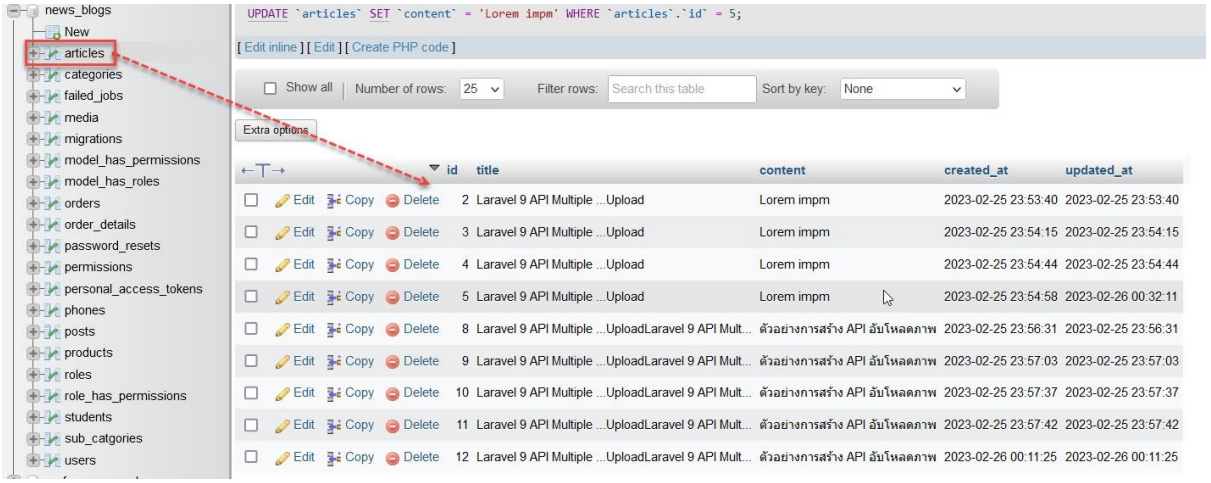
จากนั้นคลิกที่ Body ให้เลือกรูปแบบเป็น From แล้วระบุฟิวด์และใส่ข้อมูลแต่ละฟิวด์ จากนั้นติ๊กทำเครื่องหมายถูกในช่อง Files จะมีช่องให้เลือกอับโหลดไฟล์แสดงขึ้นมาให้ระบุชื่อฟิวด์เป็น images[] แล้วเลือกไฟล์ที่ต้องการจะอับโหลด เสร็จแล้วให้กดที่ปุ่ม Send เพื่อบันทึกข้อมูล



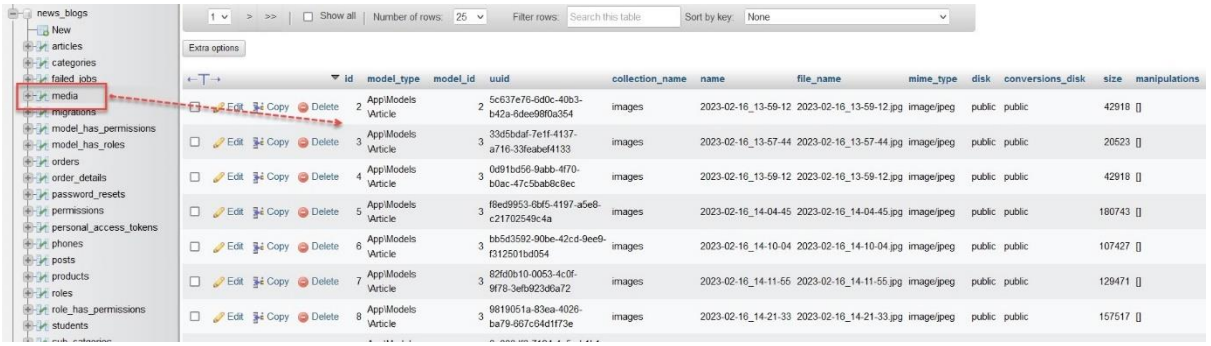


ข้อมูลก็จะถูกบันทึกไว้ในฐานข้อมูลเรียบร้อยแล้ว ตรวจสอบตาราง articles และ media โดยเปิดโปรแกรม phpMyAdmin ขึ้นมาก็จะเห็นว่ามีข้อมูลถูกเพิ่มเข้ามาแล้ว

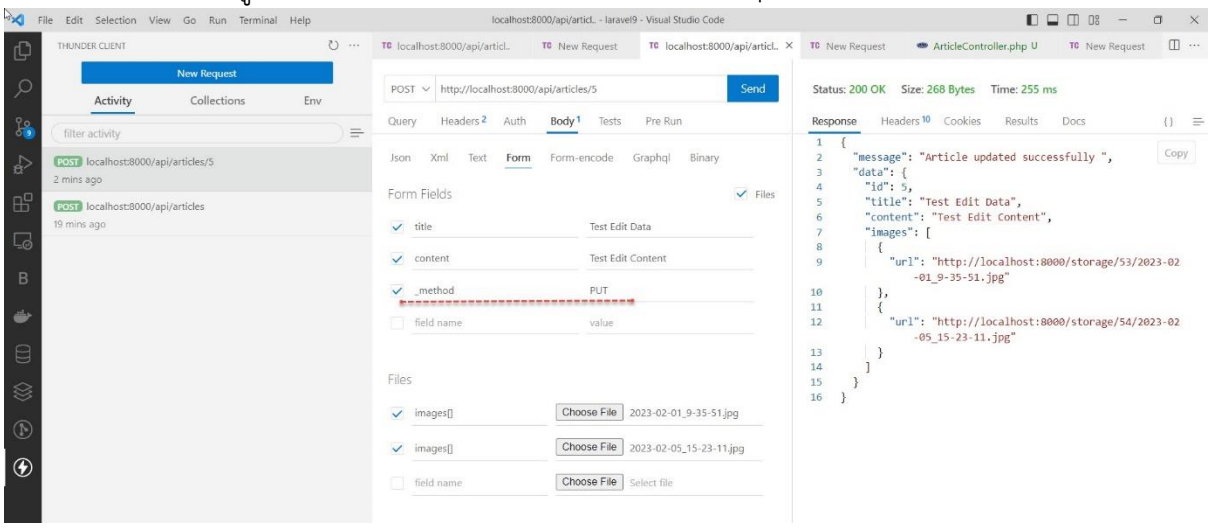
ตาราง articles



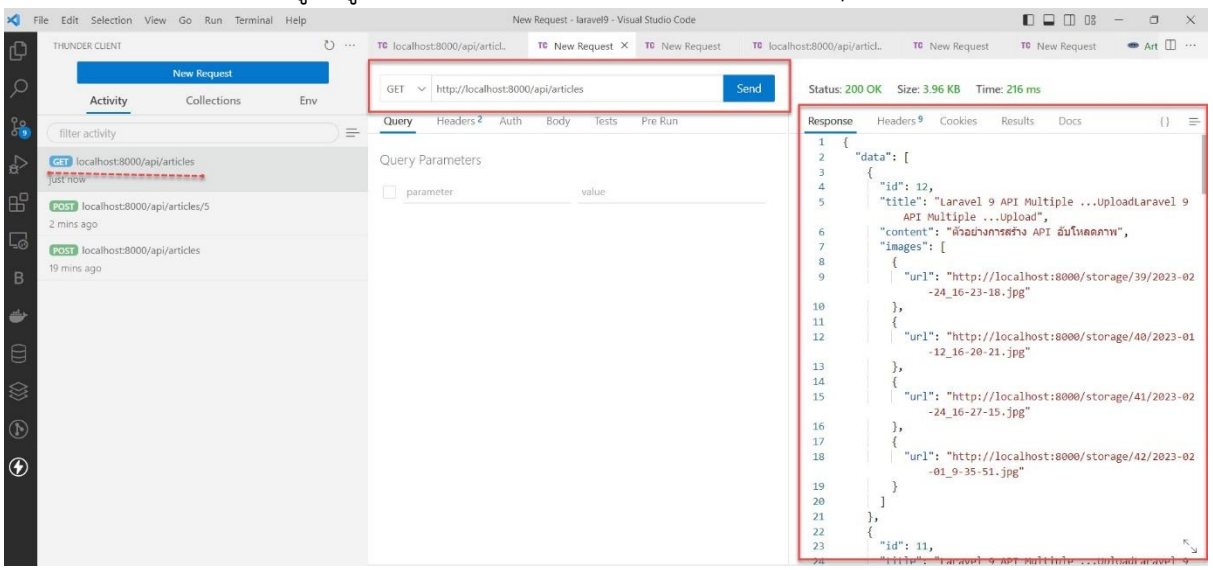
ตาราง media



การแก้ไขข้อมูลสามารถทำได้เหมือนกันโดยให้เรา New Request ขึ้นมาใหม่ แล้วกำหนดค่าตามภาพ



หากต้องการเรียกดูข้อมูลทั้งหมด จะใช้ Method GET ให้สร้าง Request ใหม่ ดังนี้



มาถึงจุดนี้แล้วก็ถือว่าการสร้าง API สำเร็จ การสร้าง API จากตัวอย่างที่ผ่านมา ผู้อ่านสามารถประยุกต์ใช้ตามวัตถุประสงค์ที่ต้องการได้ หากต้องการดูข้อมูลเพิ่มเติมสามารถดูได้จากลิงค์นี้

https://laravel.com/docs/10.x/eloquent-resources

เรียกใช้งาน API ฝั่ง Frontend ด้วย Guzzle HTTP

ในการเชื่อมต่อ API ที่เราจะเข้าถึงนั้น ในตัวอย่างหนังสือนี้จะใช้ Guzzle เป็นตัวเชื่อมต่อไปยัง URL ของ API

Guzzle เป็น PHP HTTP client ที่ใช้สำหรับส่ง HTTP requests และเชื่อมต่อกับ web services สามารถดึงข้อมูล เพิ่ม แก้ไข ลบ (GET, POST, PUT, DELETE) ได้ง่ายเพียงเขียนคำสั่งไม่กี่บรรทัด รองรับ HTTP Transport หลายรูปแบบตั้งแต่ cURL, Socket, PHP Streams ฉะนั้นก่อนการใช้งานให้ติดตั้ง Guzzle ผ่าน Terminal ด้วยคำสั่งนี้

```
composer require guzzlehttp/guzzle
```

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE

PS D:\xampp\htdocs\laravel9> composer require guzzlehttp/guzzle
Info from https://repo.packagist.org: #StandWithUkraine
./composer.json has been updated
Running composer update guzzlehttp/guzzle
Loading composer repositories with package information
Updating dependencies
Nothing to modify in lock file
Writing lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

  INFO  Discovering packages.

  barryvdh/laravel-debugbar ..... DONE
  intervention/image ..... DONE
  laravel/sail ..... DONE
  laravel/sanctum ..... DONE
  laravel/tinker ..... DONE
  laravel/ui ..... DONE
  laravelcollective/html ..... DONE
  nesbot/carbon ..... DONE
  nunomaduro/collision ..... DONE
  nunomaduro/termwind ..... DONE
  spatie/laravel-ignition ..... DONE
  spatie/laravel-medialibrary ..... DONE
  spatie/laravel-permission ..... DONE

88 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

  INFO  No publishable resources for tag [laravel-assets].

Found 1 security vulnerability advisory affecting 1 package.
Run composer audit for a full list of advisories.
Using version ^7.5 for guzzlehttp/guzzle
PS D:\xampp\htdocs\laravel9>
```

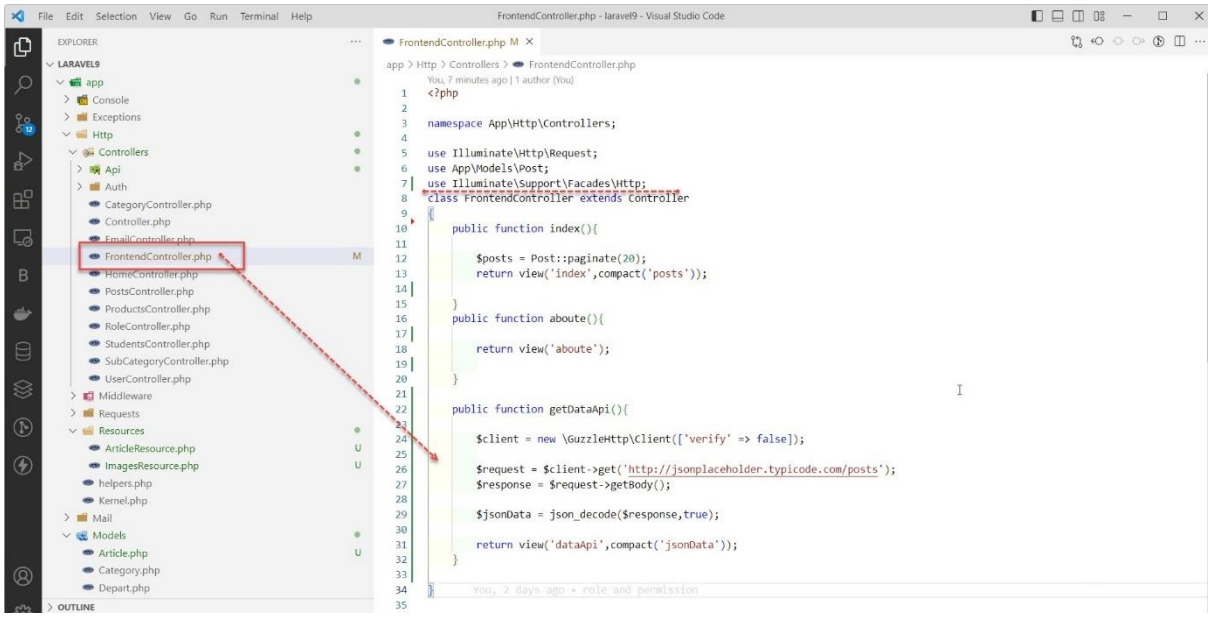
ทดสอบ API ด้วยการใช้ Guzzle HTTP client

ในตัวอย่างนี้จะเรียกใช้งาน API ผ่าน URL https://jsonplaceholder.typicode.com/posts ซึ่งเป็น API สำหรับทดสอบดึงรายการโพสต์ เพื่อให้เข้าใจยิ่งขึ้นไปดูวิธีการดึงข้อมูลจาก API ตามขั้นตอนดังนี้

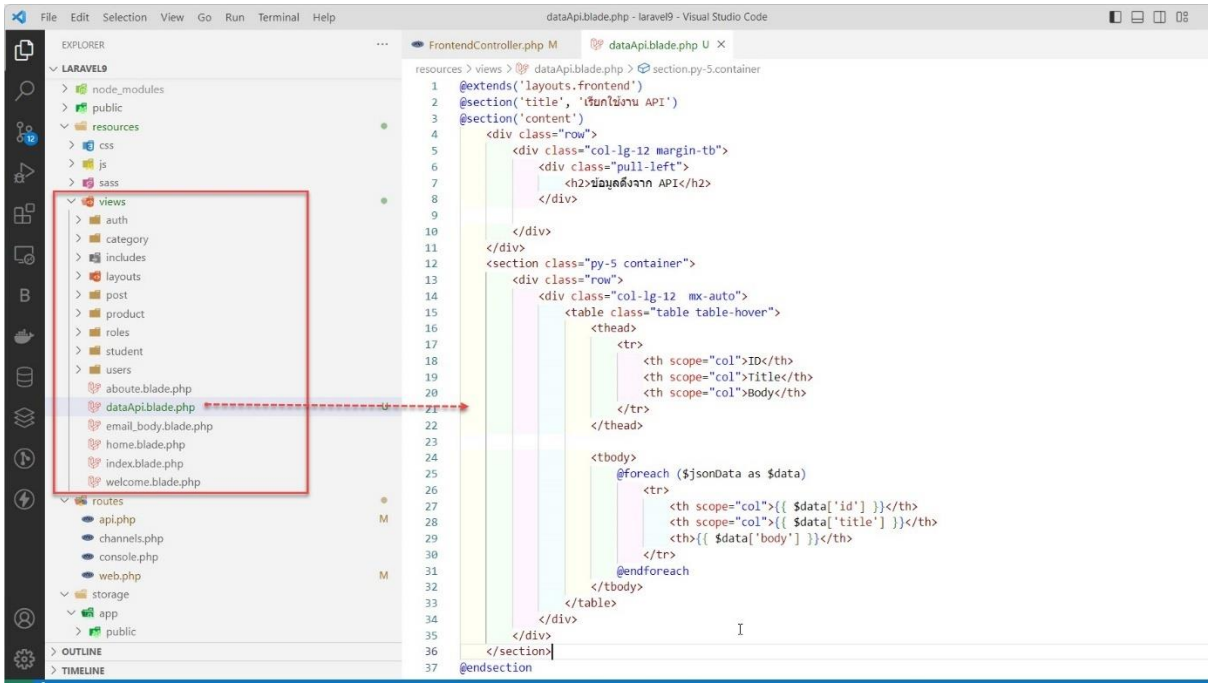
1.สร้าง FrontendController.php ผ่าน Terminal ด้วยคำสั่ง

```
php artisan make:controller FrontendController
```

2.เปิดไฟล์ FrontendController.php แล้วเพิ่มโค้ดเข้าไปดังนี้



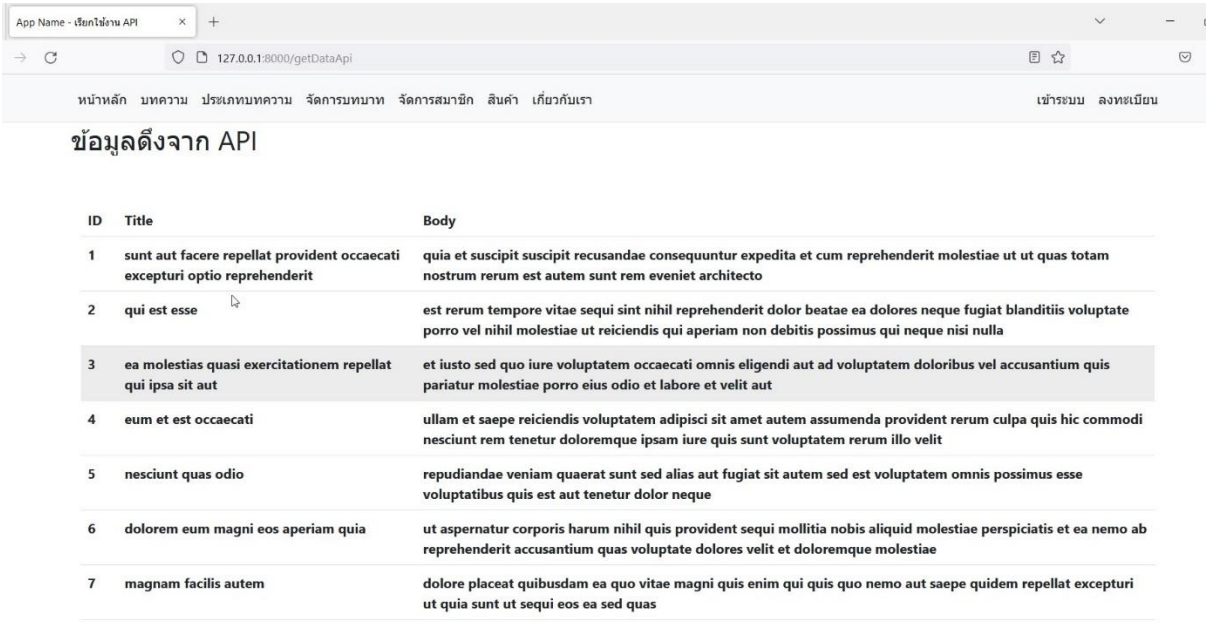
3.สร้าง views ใหม่ ชื่อ dataApi.blade.php (resources/views/dataApi.blade.php) แล้วเพิ่มโค้ดเข้าไป ดังนี้



4.กำหนดเส้นทาง URL ใน routes/web.php

```
use App\Http\Controllers\FrontendController ;
…
Route::get('getDataApi', [FrontendController::class, 'getDataApi']);
```

5.Start Server (php artisan serv) ผ่าน Terminal แล้วทดสอบการแสดงผลผ่าน Browser พิมพ์ URL http://127.0.0.1:8000/getDataApi



ID	Title	Body
1	sunt aut facere repellat provident occaecati excepturi optio reprehenderit	quia et suscipit suscipit recusandae consequuntur expedita et cum reprehenderit molestiae ut ut quas totam nostrum rerum est autem sunt rem eveniet architecto
2	qui est esse	est rerum tempore vitae sequi sint nihil reprehenderit dolor beatae ea dolores neque fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis qui aperiam non debitis possimus qui neque nisi nulla
3	ea molestias quasi exercitationem repellat qui ipsa sit aut	et iusto sed quo iure voluptatem occaecati omnis eligendi aut ad voluptatem doloribus vel accusantium quis pariatur molestiae porro eius odio et labore et velit aut
4	eum et est occaecati	ullam et saepe reiciendis voluptatem adipisci sit amet autem assumenda provident rerum culpa quis hic commodi nesciunt rem tenetur doloremque ipsam iure quis sunt voluptatem rerum illo velit
5	nesciunt quas odio	repudiandae veniam quaerat sunt sed alias aut fugiat sit autem sed est voluptatem omnis possimus esse voluptatibus quis est aut tenetur dolor neque
6	dolorem eum magni eos aperiam quia	ut aspernatur corporis harum nihil quis provident sequi mollitia nobis aliquid molestiae perspiciatis et ea nemo ab reprehenderit accusantium quas voluptate dolores velit et doloremque molestiae
7	magnam facilis autem	dolore placeat quibusdam ea quo vitae magni quis enim qui quis quo nemo aut saepe quidem repellat excepturi ut quia sunt ut sequi eos ea sed quas

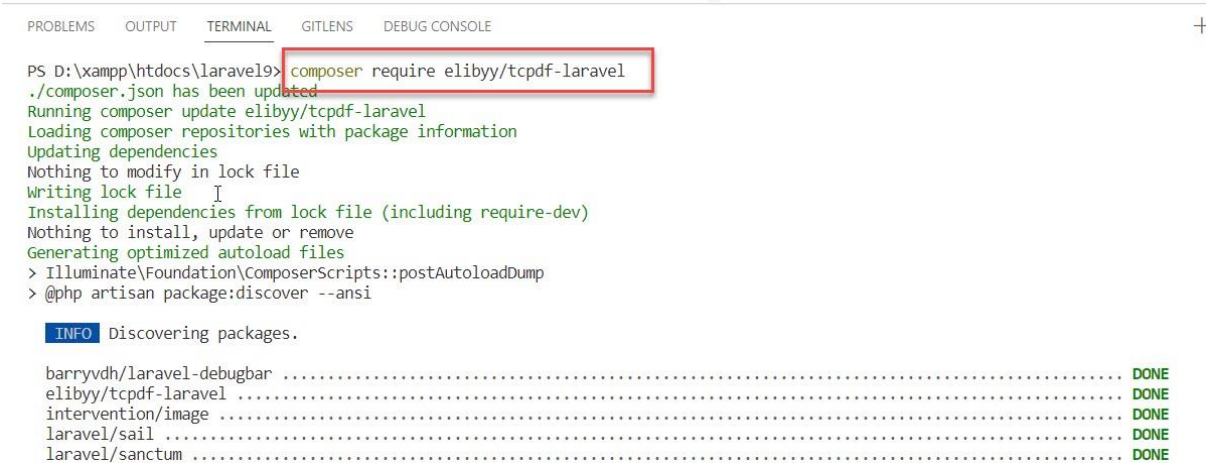
สร้างรายงานด้วย PDF ด้วย TCPDF library

การสร้างรายงานหรือต้องการ Export ไฟล์ออกเป็น PDF นั้นเราจะมี Library อยู่หลายอันที่สามารถใช้งานได้ เช่น mPDF , TCPDF ในหนังสือเล่มนี้จะใช้ Library ชื่อว่า **TCPDF** ในการทำรายงาน โดยมีขั้นตอนการติดตั้งและใช้งาน ดังนี้

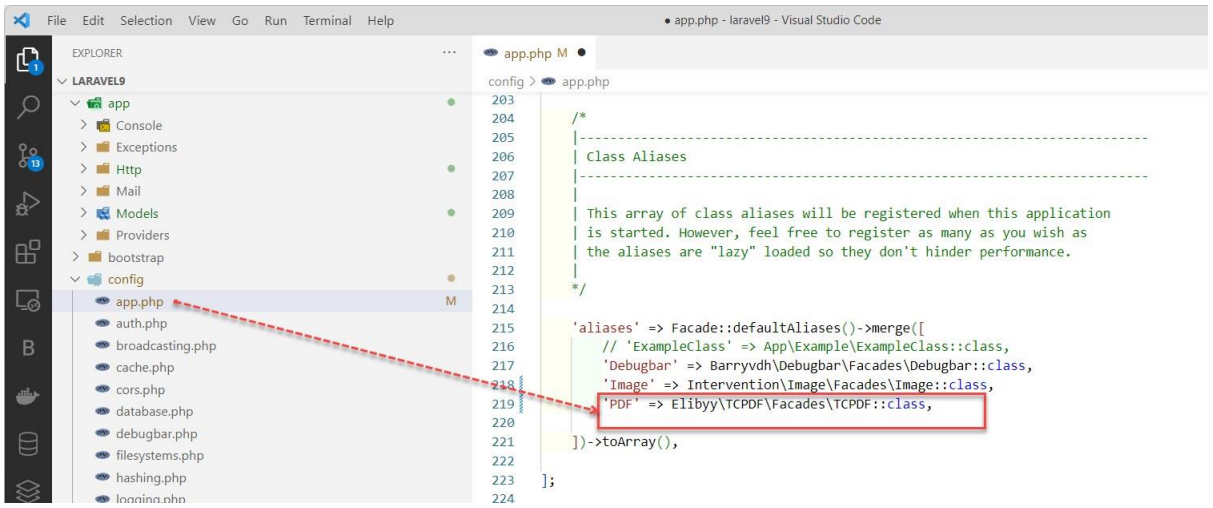
1.ติดตั้ง Library TCPDF ดูคู่มือติดตั้งเพิ่มเติมได้ที่ https://github.com/elibyy/tcpdf-laravel

```
composer require elibyy/tcpdf-laravel
```

เปิด Terminal แล้วพิมพ์คำสั่ง

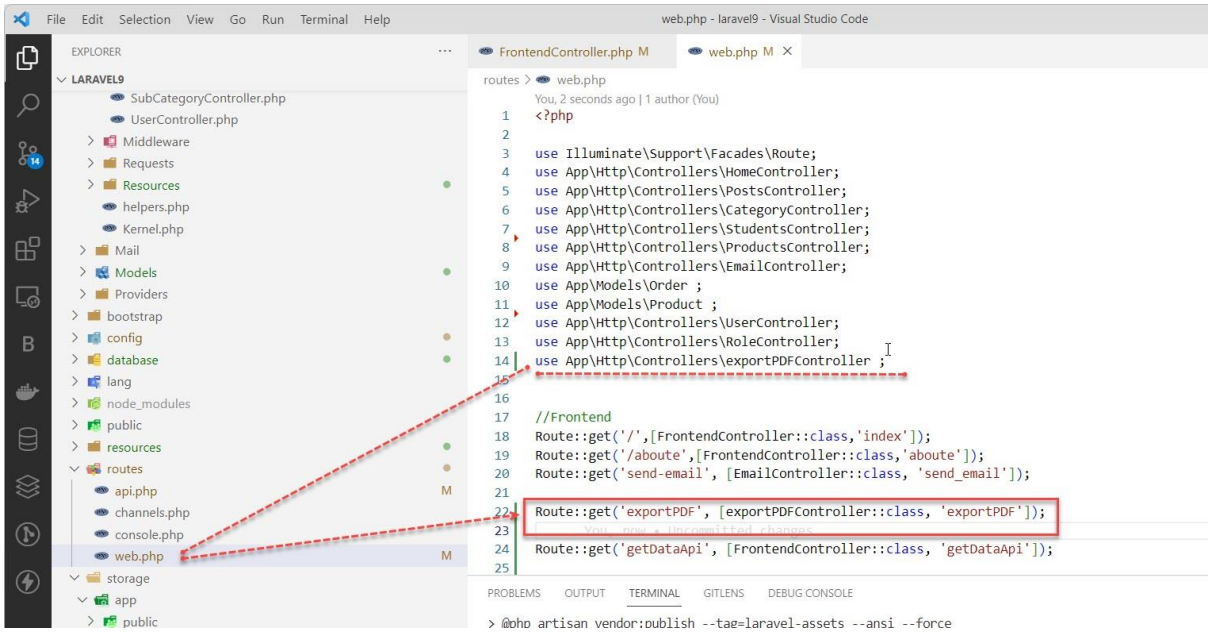


2. Config ค่าเพิ่มเติมเพื่อเปิดใช้งานตัว TCPDF ให้เปิดไฟล์ config/app.php ขึ้นมาแล้วเพิ่มคำสั่งเข้าไปใน aliases



เป็นอันเสร็จการตั้งค่าครับ ทีนี้ก็ลองเรียกใช้งานดูครับ แต่ก่อนอื่นก็ให้เราสร้าง Route สำหรับเรียกใช้งานก่อนนะครับผม โดยไปที่โฟลเดอร์ routes/web.php แล้วเพิ่มคำสั่งเข้าไป

```
use App\Http\Controllers\exportPDFController ;

Route::get('exportPDF', [exportPDFController::class, 'exportPDF']);
```



3.สร้างคอนโทรลเลอร์ exportPDFController.php ผ่าน Terminal ด้วยคำสั่ง

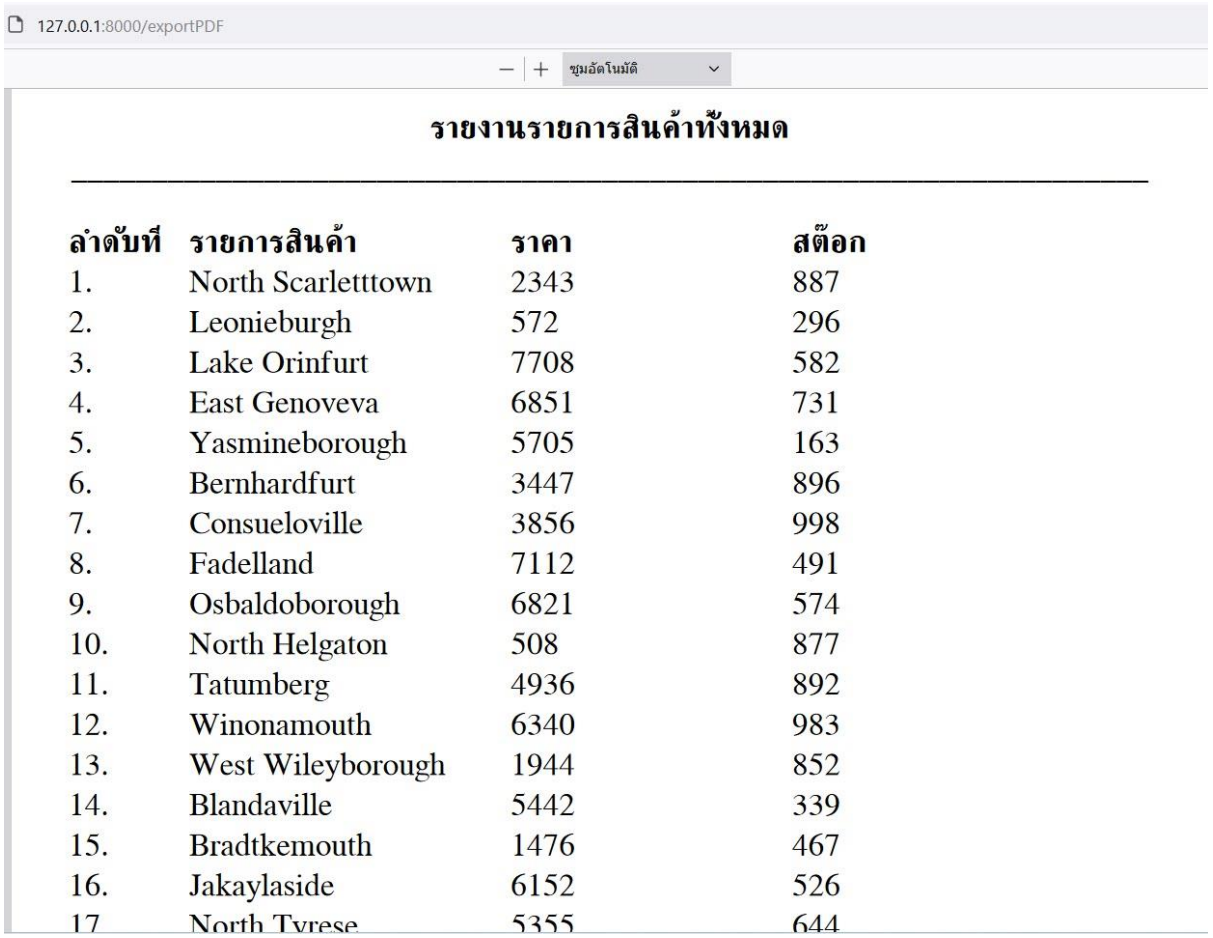
```
php artisan make:controller exportPDFController
```

4.เปิดไฟล์ exportPDFController.php แล้วสร้างเมธอดชื่อ **exportPDF** และเพิ่มคำสั่งสำหรับสร้าง PDF สามารถดูคำสั่งการใช้งานต่างๆ ในเว็บเลยนะครับ https://tcpdf.org/ อย่าลืม use PDF; มาใช้งานด้วยนะครับผม

```
//app/Http/Controllers/exportPDFController.php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use PDF ;
use DB ;
use  App\Models\Product ;
class exportPDFController extends Controller
{
    public function exportPDF(){
        $products = DB::table('products')->limit(20)->get();
        $topic = "รายงานรายการสินค้าทั้งหมด";
        $line = "_____________________________________________________________";

        PDF::SetMargins(10, 40,0);
        PDF::setHeaderCallback(function($pdf) use ($topic,$line) {

          PDF::SetFont('freeserif', 'B', 16, '', 'false');
          PDF::SetY(20); //ระยะห่างจากด้านบนมาล่าง
          PDF::SetX(0); //ระยะห่างจากซ้ายไปขวา
          PDF::Cell(0, 0, $topic, 0, false, 'C', false , 1);

          PDF::SetFont('freeserif', 'B', 16, '', 'false');
          PDF::SetY(27); //ระยะห่างจากด้านบนมาล่าง
          PDF::SetX(0); //ระยะห่างจากซ้ายไปขวา
          PDF::Cell(0, 0, $line, 0, false, 'C', false , 1);
        });
              // Custom Footer
        PDF::setFooterCallback(function($pdf) {
              // Position at 15 mm from bottom
              $pdf->SetY(-15);
              // Set font
              $pdf->SetFont('freeserif', '', 16, '', 'false');
              // Page number
              $pdf->Cell(0, 15, 'หน้า '.$pdf->getAliasNumPage().' จาก '.$pdf->getAliasNbPages(), 0, false, 'C', 0, '', 0,false,'T', 'M');
        });
```

มีโค้ดต่อหน้าถัดไป ...

```
//app/Http/Controllers/exportPDFController.php
        PDF::AddPage('A4');
        $html = '<table border="0" width ="400px">
                <thead>
                <tr>
                <th width ="15%"><b>ลำดับที่</b></th>
                <th width ="40%"><b>รายการสินค้า</b></th>
                <th width = "35%"><b>ราคา</b></th>
                <th width = "35%"><b>สต๊อก</b></th>
                </tr>
                </thead>
        <tbody>';
                $i = 1;
                foreach ($products as $item) {
        $html .='
                <tr>
                <td width="15%">'.$i++.'.</td>
                <td width="40%">'.$item->name.'</td>
                <td width="35%">'.$item->price.'</td>
                <td width="35%">'.$item->stock.'</td>
                </tr>';
                }
        $html .='</tbody></table><br/>'.$line;

        PDF::SetFont('freeserif ', '', 16, '', 'false');
        PDF::writeHTML($html, true, false, true, false, '');

        PDF::lastPage();
        PDF::setPrintFooter(true);
        PDF::Output('product.pdf','I');
        PDF::reset();
        PDF::endPage();
    }
}
```



ลำดับที่	รายการสินค้า	ราคา	สต๊อก
1.	North Scarletttown	2343	887
2.	Leonieburgh	572	296
3.	Lake Orinfurt	7708	582
4.	East Genoveva	6851	731
5.	Yasmineborough	5705	163
6.	Bernhardfurt	3447	896
7.	Consueloville	3856	998
8.	Fadelland	7112	491
9.	Osbaldoborough	6821	574
10.	North Helgaton	508	877
11.	Tatumberg	4936	892
12.	Winonamouth	6340	983
13.	West Wileyborough	1944	852
14.	Blandaville	5442	339
15.	Bradtkemouth	1476	467
16.	Jakaylaside	6152	526
17.	North Tyrese	5355	644

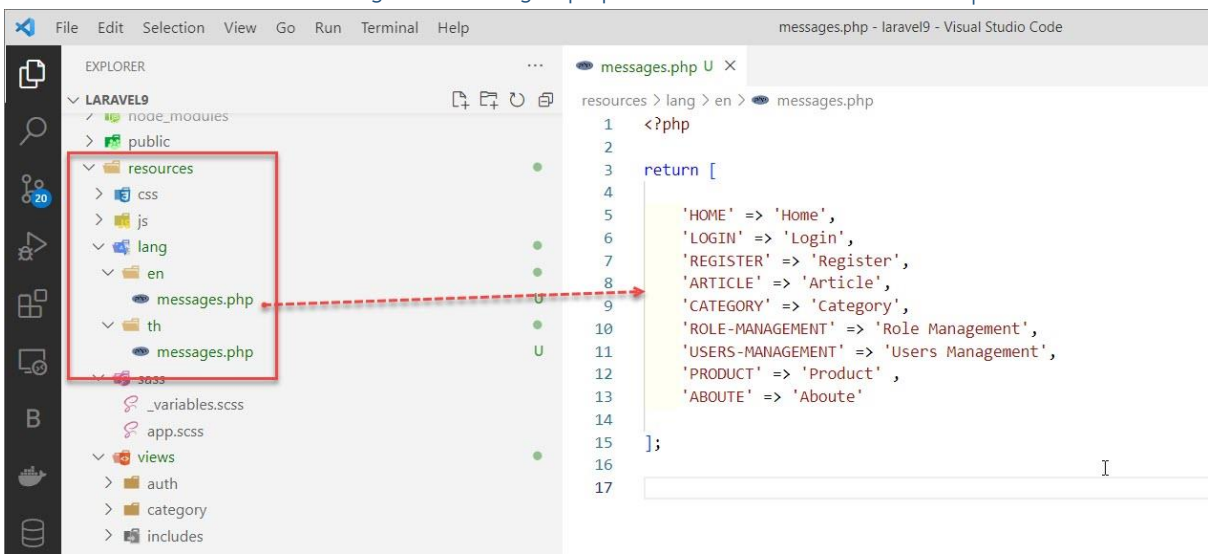
จะได้ไฟล์รายงานที่เป็นแบบ PDF แล้ว จากตัวอย่างจะเห็นว่าเราได้สร้างรายงานของสินค้าโดยดึงข้อมูลจากฐานข้อมูลในตาราง products มาแสดง

เว็บไซต์หลายภาษา (Multi-Language)

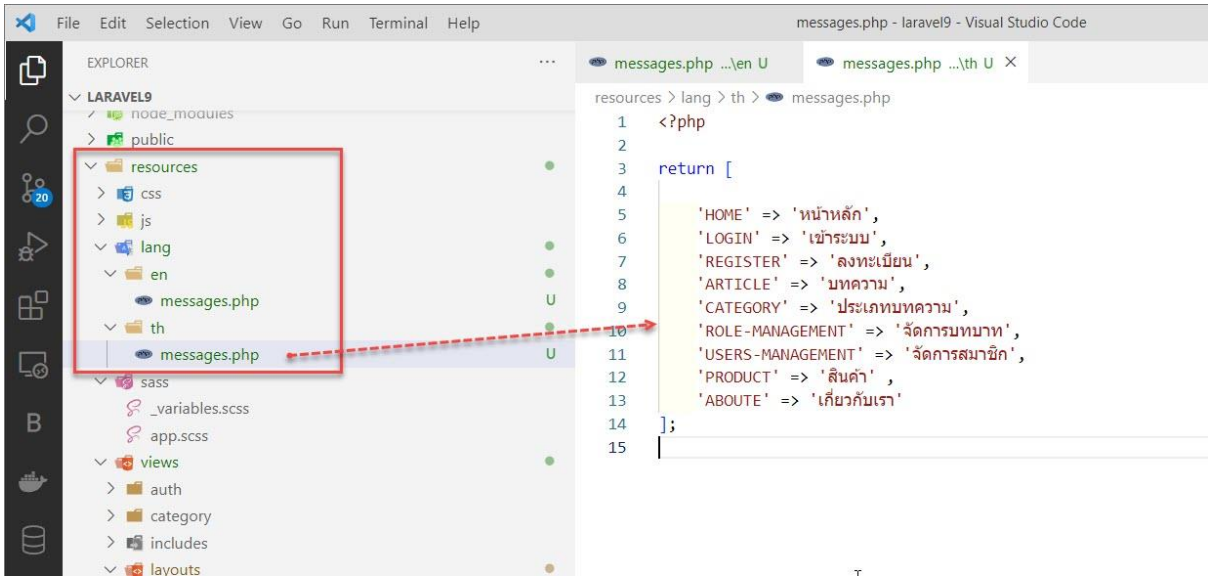
บทเรียนนี้จะเป็นวิธีการสร้างเว็บไซต์ให้รองรับหลาย ๆ ภาษา จะยกตัวอย่างการทำแบบง่าย ๆ โดยจะมีการสร้างตัวเลือกภาษาสำหรับการเปลี่ยนภาษาตามต้องการผ่านหน้าเว็บ ประกอบด้วยภาษาอังกฤษและภาษาไทย มีขั้นตอนดังนี้

1.สร้างไฟล์สำหรับเก็บภาษาแต่ละภาษา โดยสร้างโฟลเดอร์ lang เก็บไว้ใน resources/... ดังนี้

resources/lang/en/messages.php สำหรับเก็บข้อความภาษาอังกฤษ

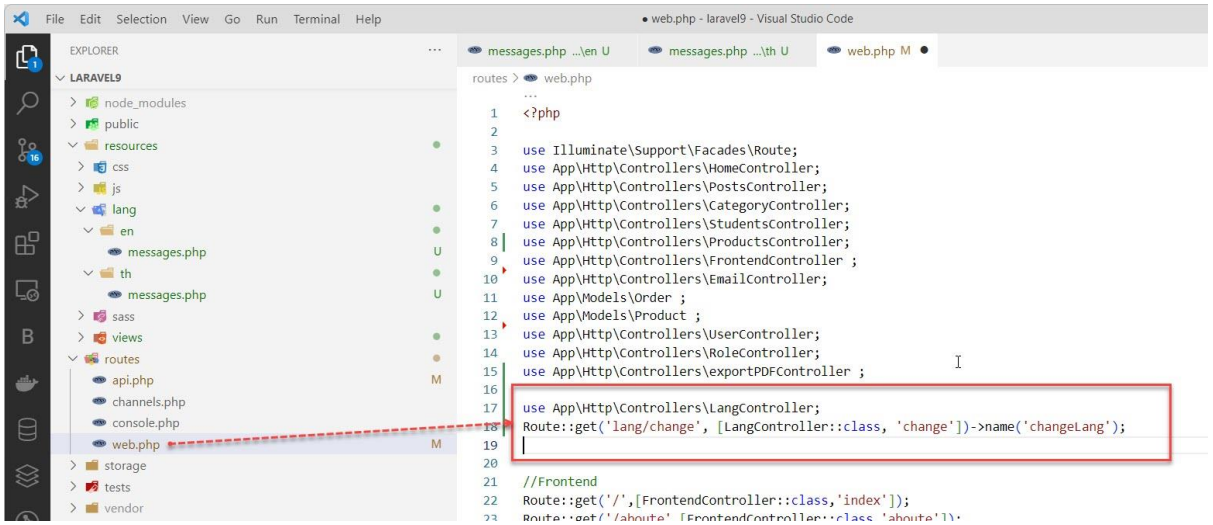


resources/lang/th/messages.php สำหรับเก็บข้อความภาษาไทย



ให้กำหนดคำศัพท์ลงไปในไฟล์ทั้ง 2 ไฟล์ โดยจะเก็บข้อมูลในรูปแบบ Array และระบุ key แล้วก็ค่าของคำศัพท์แต่ละข้อความ โดยสังเกตว่า Key จะใช้ชื่อเดียวกันทั้งไฟล์ en / th

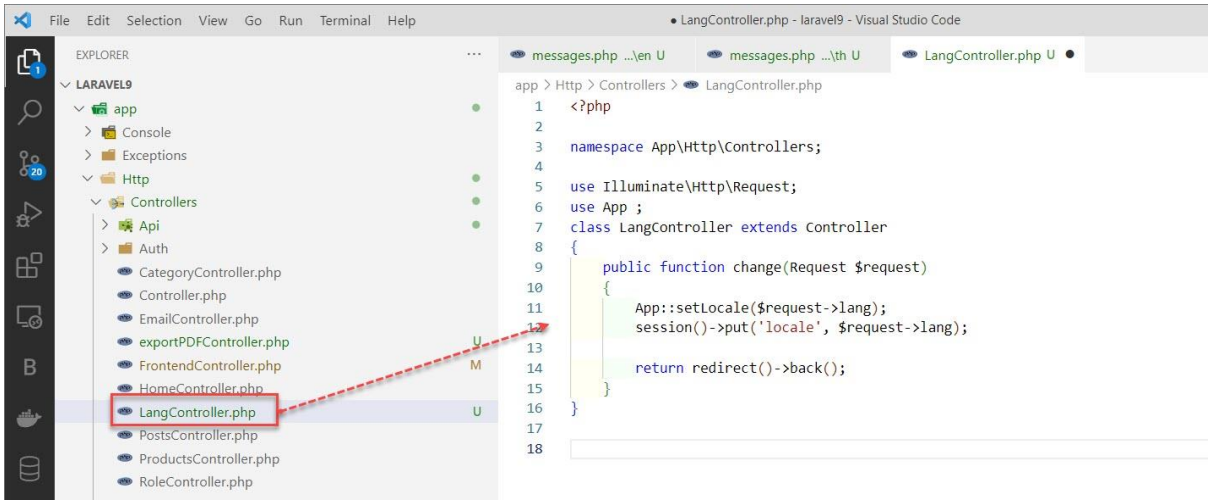
2.สร้าง routes สำหรับเปลี่ยนภาษา โดยเปิดไฟล์ routes/web.php แล้วเพิ่ม routes ใหม่ ดังนี้



3.สร้างไฟล์ LangController.php ผ่าน Terminal ด้วยคำสั่ง

```
php artisan make:controller LangController
```

เปิดไฟล์ LangController.php แล้วสร้างเมธอด change() แล้วเพิ่มโค้ดสำหรับเปลี่ยนภาษา ดังนี้



4.เพิ่มลิงค์สำหรับให้เลือกเปลี่ยนภาษาจากหน้าเว็บ โดยเปิดไฟล์ frontend.blade.php ในโฟลเดอร์ resources/views/layouts/... ให้เพิ่มตามตำแหน่งที่เราต้องการให้แสดงได้เลย ในตัวอย่างจะเพิ่มให้แสดงถัดจากลิงค์ ลงทะเบียน (ดูโค้ดเต็ม ๆ ได้ที่หน้า 303)

```
...
<li>
        <select class="form-control changeLang">
            <option value="en" {{ session()->get('locale') == 'en' ? 'selected' : '' }}>English</option>
            <option value="th" {{ session()->get('locale') == 'th' ? 'selected' : '' }}>ภาษาไทย</option>
        </select>
</li>
....
```

จากนั้นกำหนด Script เพิ่มเติมก่อน `@yield('js_before')` ดังนี้

```
...
<script type="text/javascript">
    var url = "{{ route('changeLang') }}";
    $(".changeLang").change(function() {
        window.location.href = url + "?lang=" + $(this).val();
    });
</script>
@yield('js_before')
</html>
```

แก้ไขเมนูเพื่อให้ข้อความเปลี่ยนแปลงภาษาตามที่เลือกโดยแก้ในส่วนของเมนูทั้งหมด ดังนี้

```
//resources/views/layouts/frontend.blade.php
 <ul class="nav me-auto">

          <li class="nav-item">
            <a href="{{ url('/') }}" class="nav-link link-dark px-2 active"aria-current="page">{{ __('messages.HOME') }}</a>
          </li>
          <li class="nav-item"><a href="{{ url('/posts') }}"
                    class="nav-link link-dark px-2">{{ __('messages.ARTICLE') }}</a>
          </li>
          <li class="nav-item">
                      <a href="{{ url('/category') }}"class="nav-link link-dark px-2">{{ __('messages.CATEGORY') }}</a>
          </li>
          <li class="nav-item">
                      <a  href="{{ route('roles.index') }}"class="nav-link link-dark px-2">{{ __('messages.ROLE-MANAGEMENT') }}</a>
          </li>
            <li class="nav-item">
            <a href="{{ route('users.index') }}"class="nav-link link-dark px-2">
            {{ __('messages.USERS-MANAGEMENT') }}</a>
            </li>
          <li class="nav-item"><a href="{{ url('/products') }}" class="nav-link link-dark px-2">{{ __('messages.PRODUCT') }}</a></li>
          <li class="nav-item"><a href="{{ url('/aboute') }}"class="nav-link link-dark px-2">{{ __('messages.ABOUTE') }}</a></li>
 </ul>
```

{{ trans('ระบุคีย์ของภาษา') }} ส่วนไหนที่ต้องการให้เปลี่ยนภาษาก็ให้ใส่ในส่วนนั้น ๆ

```
//resources/views/layouts/frontend.blade.php
<!doctype html>
<html lang="{{ config('app.locale') }}">
<!-- CSRF Token -->
<meta name="csrf-token" content="{{ csrf_token() }}">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>App Name - @yield('title')</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
    <link href="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css" rel="stylesheet" />
    <link href="https://cdn.jsdelivr.net/npm/sweetalert2@11.7.1/dist/sweetalert2.min.css" rel="stylesheet">
    @yield('css_before')
</head>
<body>
    <div class="b-example-divider"></div>
    <nav class="py-2 bg-light border-bottom">
        <div class="container d-flex flex-wrap">
            <ul class="nav me-auto">
                <li class="nav-item"><a href="{{ url('/') }}" class="nav-link link-dark px-2 active" aria-current="page">
                        {{ __('messages.HOME') }}</a>
                </li>
                <li class="nav-item"><a href="{{ url('/posts') }}" class="nav-link link-dark px-2">
                        {{ __('messages.ARTICLE') }}</a>
                </li>
                <li class="nav-item"><a href="{{ url('/category') }}" class="nav-link link-dark px-2">
                        {{ __('messages.CATEGORY') }}</a>
                </li>
                <li class="nav-item"><a href="{{ route('roles.index') }}"class="nav-link link-dark px-2">
                        {{ __('messages.ROLE-MANAGEMENT') }}</a>
                </li>
                <li class="nav-item"><a href="{{ route('users.index') }}"class="nav-link link-dark px-2">
                        {{ __('messages.USERS-MANAGEMENT') }}</a>
                </li>
                <li class="nav-item"><a href="{{ url('/products') }}" class="nav-link link-dark px-2">
                        {{ __('messages.PRODUCT') }}</a>
                </li>
                <li class="nav-item"><a href="{{ url('/aboute') }}" class="nav-link link-dark px-2">
                        {{ __('messages.ABOUTE') }}</a>
                </li>
            </ul>
            <ul class="nav">
                @guest
                    @if (Route::has('login'))
                        <li class="nav-item"><a href="{{ route('login') }}"class="nav-link link-dark px-2">{{ __('messages.LOGIN') }}</a>
                        </li>
                    @endif
```

มีโค้ดต่อหน้าถัดไป ...

มีโค้ดต่อหน้าถัดไป...

//resources/views/layouts/frontend.blade.php

```
                @if (Route::has('register'))
                    <li class="nav-item"><a href="{{ route('register') }}" class="nav-link link-dark px-2">
                        {{ __('messages.REGISTER') }}</a>
                    </li>
                @endif
            @else
                <li class="nav-item dropdown">
                    <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#" role="button"
                        data-bs-toggle="dropdown" aria-haspopup="true" aria-expanded="false" v-pre> {{ Auth::user()->name }}
                    </a>
                    <div class="dropdown-menu dropdown-menu-end" aria-labelledby="navbarDropdown">
                        <a class="dropdown-item" href="{{ route('logout') }}"
                            onclick="event.preventDefault(); document.getElementById('logout-form').submit();">
                            ออกจากระบบ
                        </a>
                        <form id="logout-form" action="{{ route('logout') }}" method="POST" class="d-none">
                            @csrf
                        </form>
                    </div>
                </li>
            @endguest
            <li>
                <select class="form-control changeLang">
                    <option value="en" {{ session()->get('locale') == 'en' ? 'selected' : '' }}>English
                    </option>
                    <option value="th" {{ session()->get('locale') == 'th' ? 'selected' : '' }}>ภาษาไทย
                    </option>
                </select>
            </li>
        </ul>
    </div>
</nav>
<div class="container">
    @yield('content')
</div>
</body>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>
<script src="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/toastr.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/sweetalert2@11.7.1/dist/sweetalert2.all.min.js"></script>
```

มีโค้ดต่อหน้าถัดไป ...

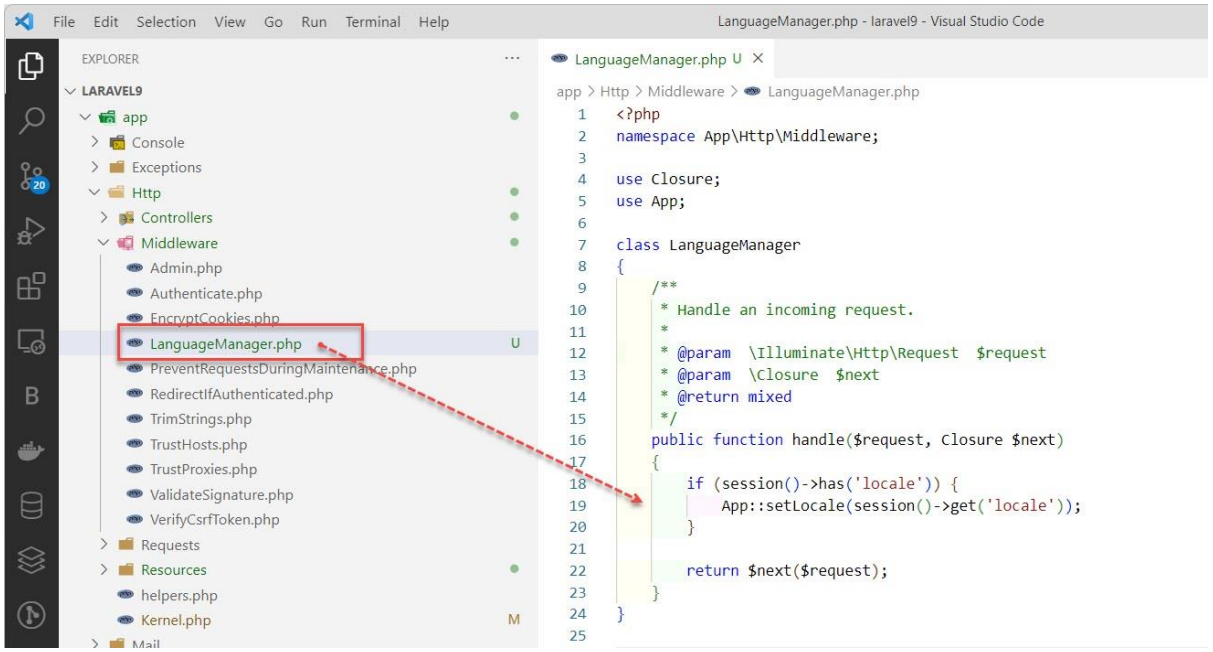
```
//resources/views/layouts/frontend.blade.php
@if (Session::has('success'))
    <script>
        Swal.fire(
            'แจ้งเตือนการทำรายการ!',
            '{{ Session::get('success') }}',
            'success'
        )
    </script>
@endif
<script type="text/javascript">
    var url = "{{ route('changeLang') }}";
    $(".changeLang").change(function() {
        window.location.href = url + "?lang=" + $(this).val();
    });
</script>
@yield('js_before')
</html>
```

4.สร้างไฟล์ Middleware ชื่อ LanguageManager ผ่าน Terminal ด้วยคำสั่ง

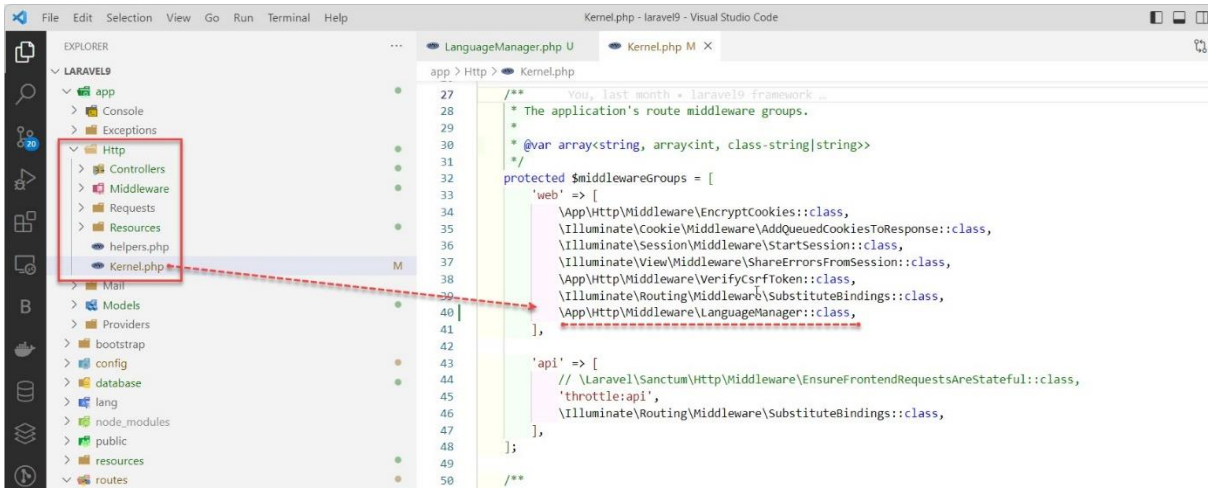
```
php artisan make:middleware LanguageManager
```

ไฟล์ LanguageManager.php จะถูกสร้างไว้ในโฟลเดอร์ app/Http/Middleware/...

เปิดไฟล์ LanguageManager.php แล้วเพิ่มโค้ดสำหรับให้เว็บกำหนดภาษาตาม Session ที่เลือก



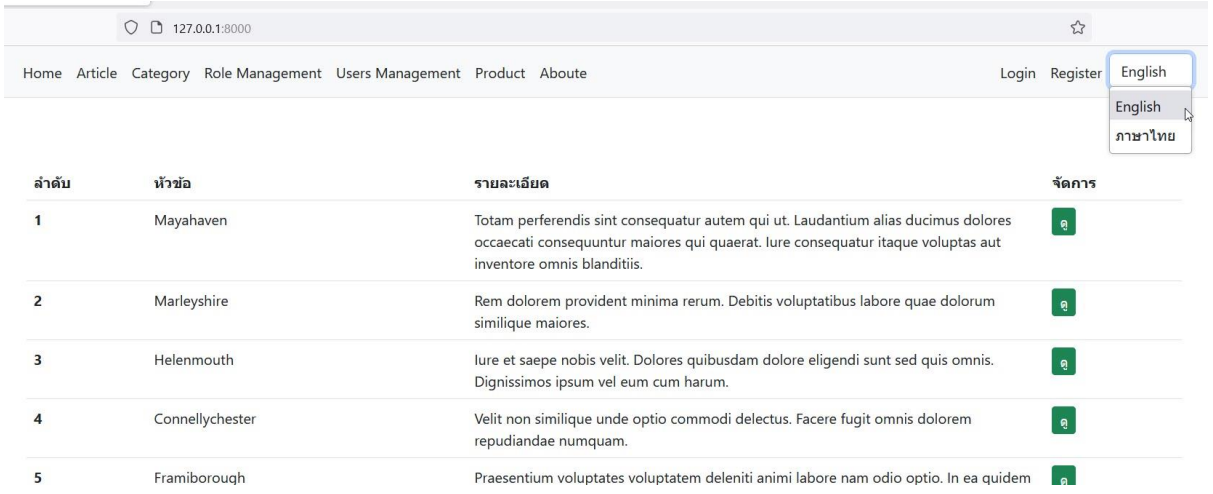
5.ลงทะเบียน Middleware ที่ไฟล์ app/Http/Kelnel.php โดยเพิ่ม Middleware เข้าไปในส่วนของ Array $middlewareGroups = [..] ดังนี้



เมื่อทำตามขั้นตอนที่ผ่านมาหมดทุกขั้นตอนแล้ว ให้ทดสอบการทำงาน โดยสั่ง start server อีกครั้ง ด้วยคำสั่ง php artisan serve จากนั้นเปิด Browser ขึ้นมาแล้วพิมพ์ URL http://localhost:8000



จะปรากฏช่องให้เลือกภาษาตามที่กำหนดไว้ เมื่อคลิกเลือกภาษาที่ต้องการแสดงข้อความเมนูก็จะถูกเปลี่ยนตามภาษาที่กำหนดไว้เรียบร้อยแล้ว



Laravel Livewire CRUD ด้วย Bootstrap Modal

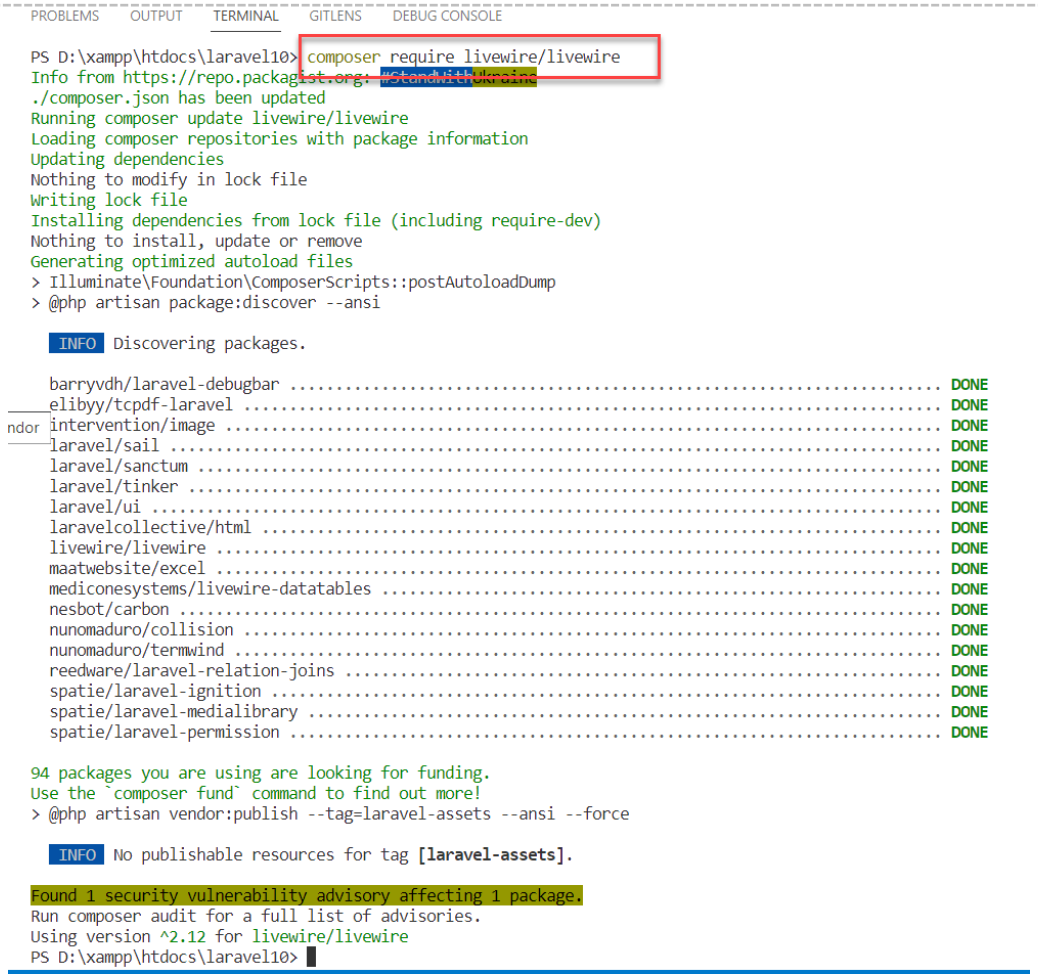
Livewire (ไลวไวร์) คือ ส่วนเสริมที่ทำให้ Laravel มีความสามารถในการสร้าง Component และเรียกใช้ได้เหมือน Component ใน React, Vue, Angular ทำให้เราสามารถสร้างแอปพลิเคชันได้อย่างง่ายดายและรวดเร็ว โดยที่เราไม่จำเป็นต้องเขียน JavaScript เลย

Workshop นี้จะเป็นการเรียนรู้การข้อมูลลูกค้า (Customer) เช่น การเพิ่มข้อมูล การดึงข้อมูลมาแสดง การแก้ไขข้อมูล และการลบข้อมูล โดยเราจะใช้ Bootstrap Modal รวมถึงการทำ Validation ข้อมูลก่อนการบันทึก จากนั้นก็จะไปเรียนรู้การทำตัวแบ่งหน้า (Pagination) และ การค้นหาข้อมูล ซึ่งทุกอย่างนี้เราจะใช้ Livewire จัดการลองมาดูตัวอย่างการสร้างและใช้งานกันเลยครับ

***สามารถดูวิธีการใช้งานเพิ่มเติมได้ที่เว็บไซต์หลัก https://laravel-livewire.com

1.ติดตั้ง Library Livewire โดยพิมพ์คำสั่งผ่าน Terminal แล้วกด Enter ดังนี้

```
composer require livewire/livewire
```



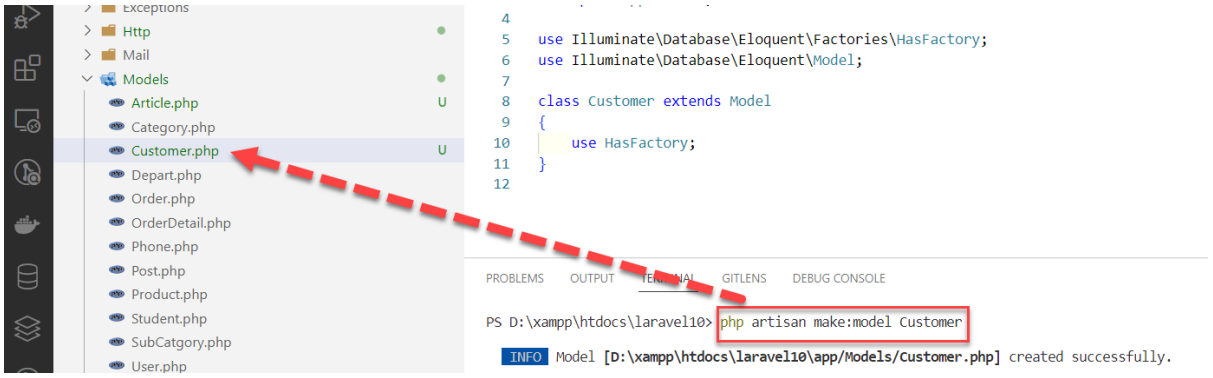
2.ตั้งค่าเพื่อเปิดใช้งาน Livewire

เมื่อทำการติดตั้ง Library Livewire เสร็จเรียบร้อยแล้วหากไม่พบปัญหาอะไร ให้ทำการตั้งค่าเพื่อเปิดใช้งาน Livewire โดยการเพิ่มคำสั่ง @livewireStyles และ @livewireScripts ใน template หลักที่เราใช้งาน โดยเพิ่ม @livewireStyles ไว้ภายใน tag head และ @livewireScripts ถัดจาก tag ปิด </body>

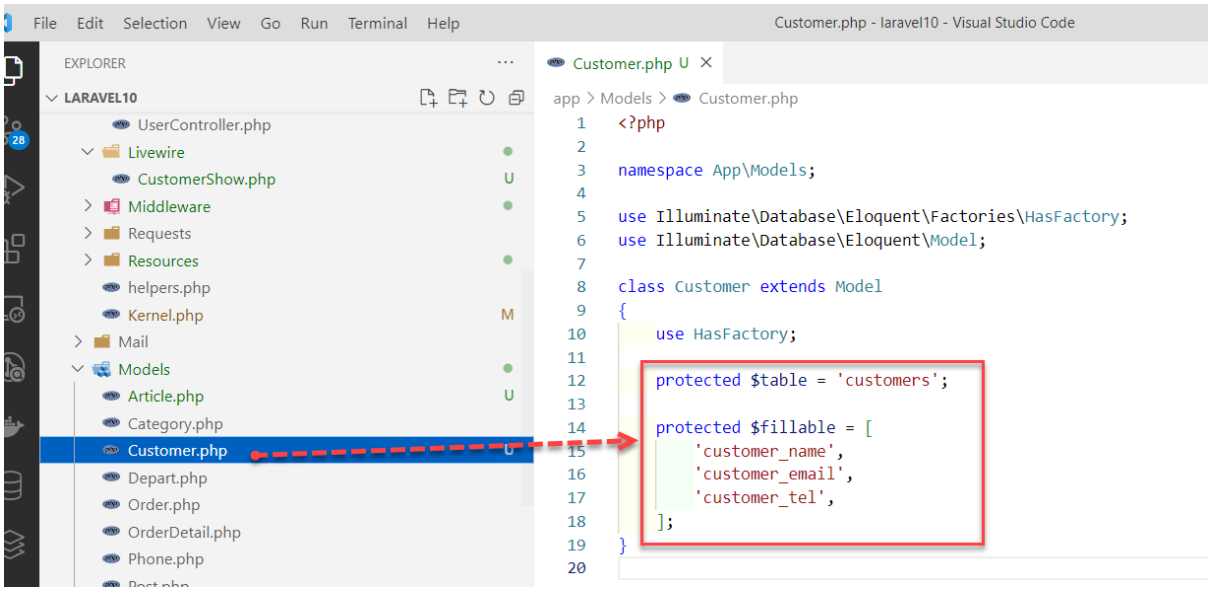
เปิดไฟล์ frontend.blade.php ในโฟลเดอร์ views/layouts/...แล้วเพิ่มคำสั่ง ดังนี้

```
<!-- resources/views/layouts/frontend.blade.php -->
<!doctype html>
<html lang="{{ config('app.locale') }}">
<!-- CSRF Token -->
<meta name="csrf-token" content="{{ csrf_token() }}">
<head>
…
    @livewireStyles

    @yield('css_before')
</head>

<body>
…
</body>
…
   @livewireScripts
…
@yield('js_before')

</html>
```

```
php  artisan make:model Customer
```



ไฟล์ Customer.php จะถูกสร้างและเก็บไว้ในโฟลเดอร์ app/Model/... ให้เปิดไฟล์ Customer.php แล้วเพิ่มโค้ดเข้าไป ดังนี้

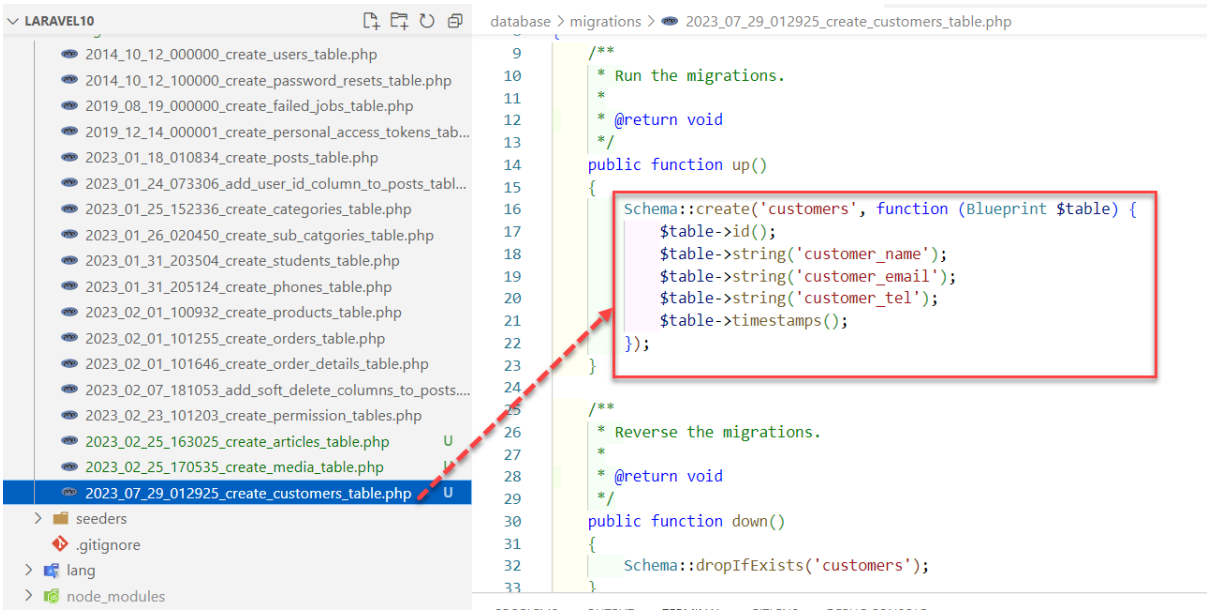


4.สร้างไฟล์ Migration สำหรับการสร้างตาราง Customers โดยพิมพ์คำสั่งผ่าน Terminal แล้วกด Enter ดังนี้

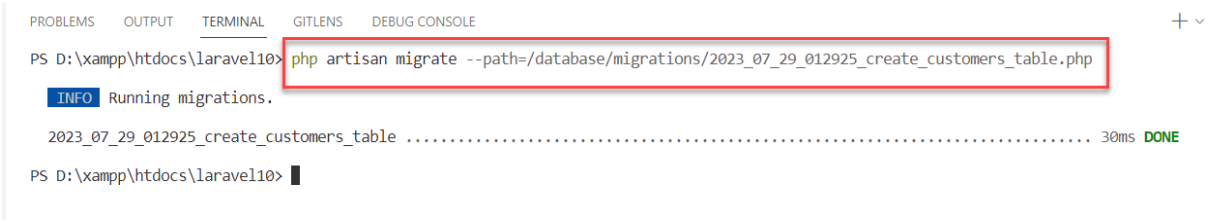
```
php  artisan make:migration create_customers_table
```



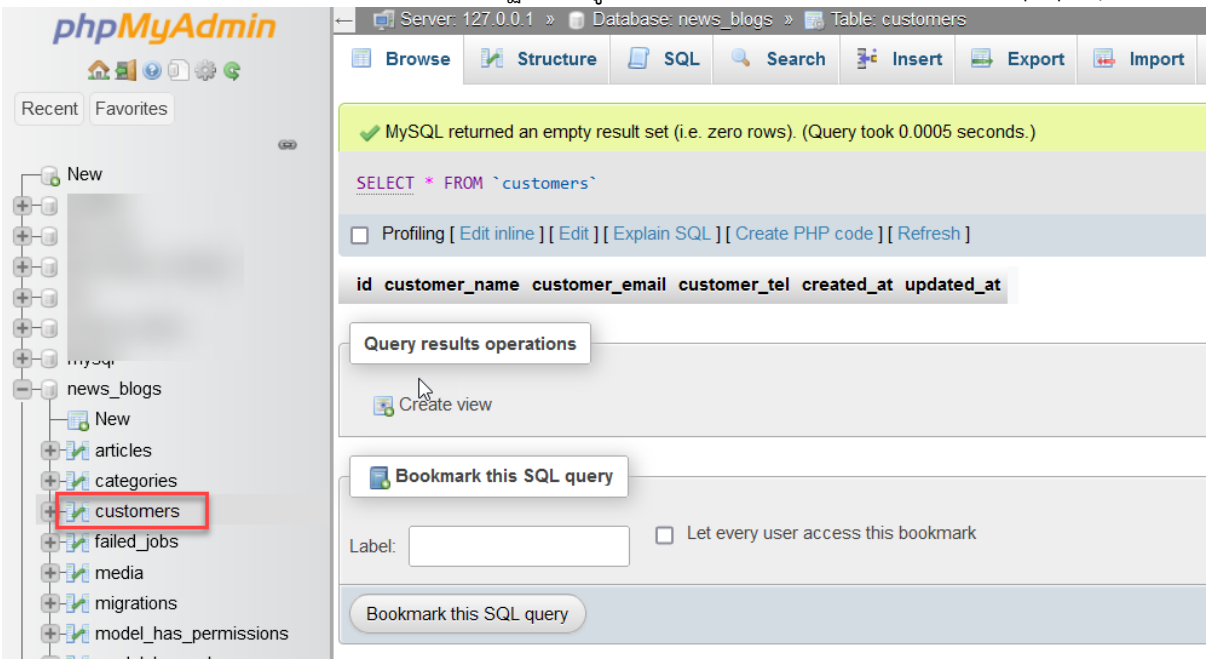
ไฟล์ xxx_create_customer_table.php จะถูกสร้างและเก็บไว้ในโฟลเดอร์ database/migrations/... ให้เปิดไฟล์ create_customer_table.php แล้วเพิ่มโค้ดเข้าไป ดังนี้



จากนั้นทำการรันคำสั่งผ่าน Terminal เพื่อสั่งให้สร้างตารางใหม่ โดยพิมพ์คำสั่ง ดังนี้

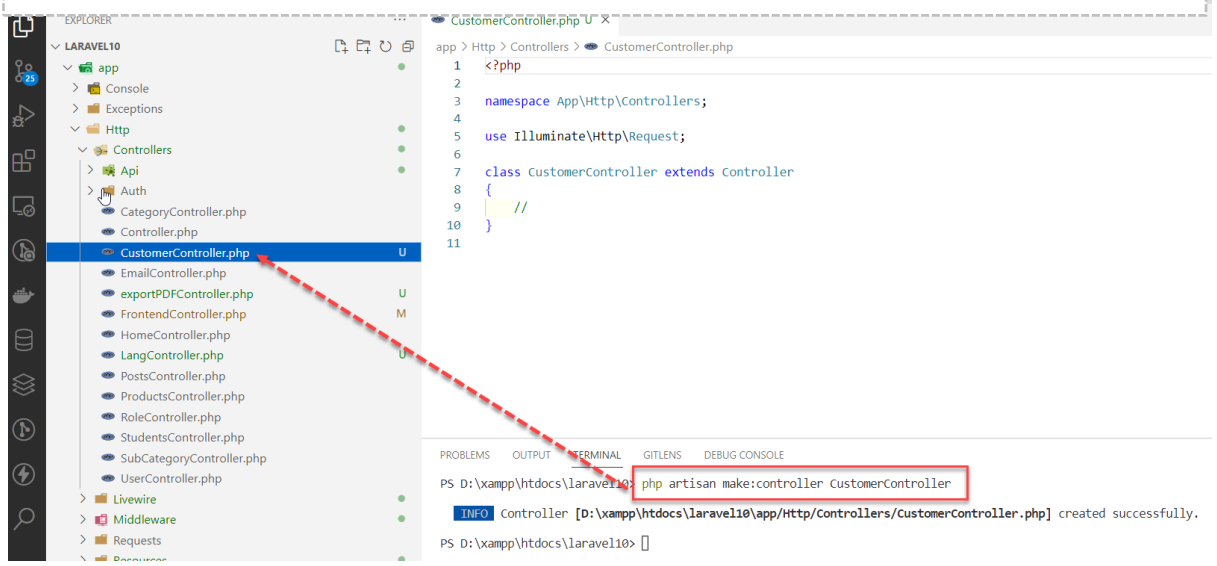


ตาราง customers ก็จะเพิ่มเข้ามาในฐานข้อมูล ตรวจสอบโดยการพิมพ์ localhost/phpmyadmin

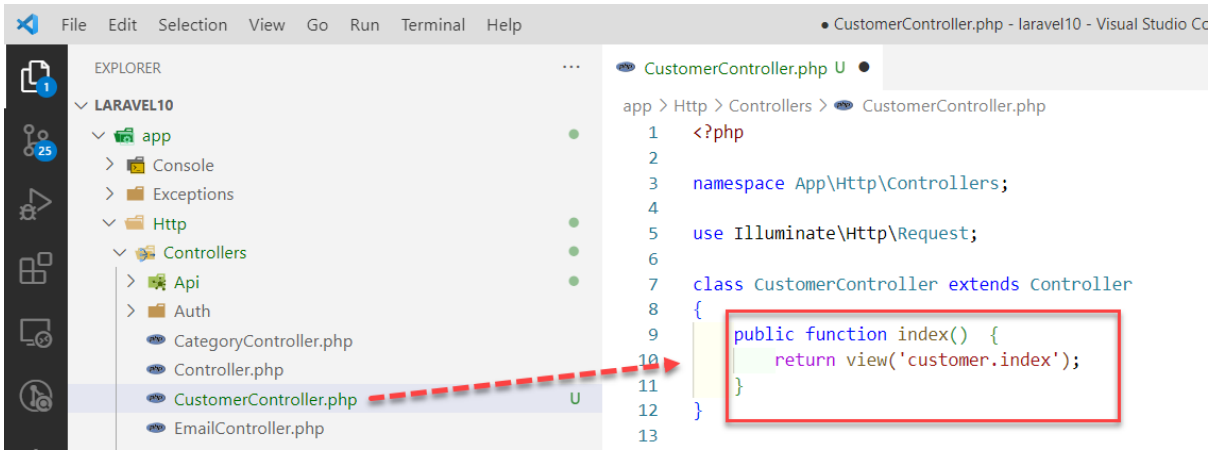


5.สร้างไฟล์ Controller สำหรับ Customer โดยพิมพ์คำสั่งผ่าน Terminal แล้วกด Enter ดังนี้

```
php  artisan make:controller CustomerController
```



ไฟล์ CustomerController.php จะถูกสร้างและเก็บไว้ในโฟลเดอร์ app/Http/Controllers/... ให้เปิดไฟล์ CustomerController.php แล้วเพิ่มโค้ดเข้าไป ดังนี้



6.สร้างไฟล์ component สำหรับ Customer โดยพิมพ์คำสั่งผ่าน Terminal แล้วกด Enter ดังนี้

```
php  artisan make:livewire CustomerShow
```

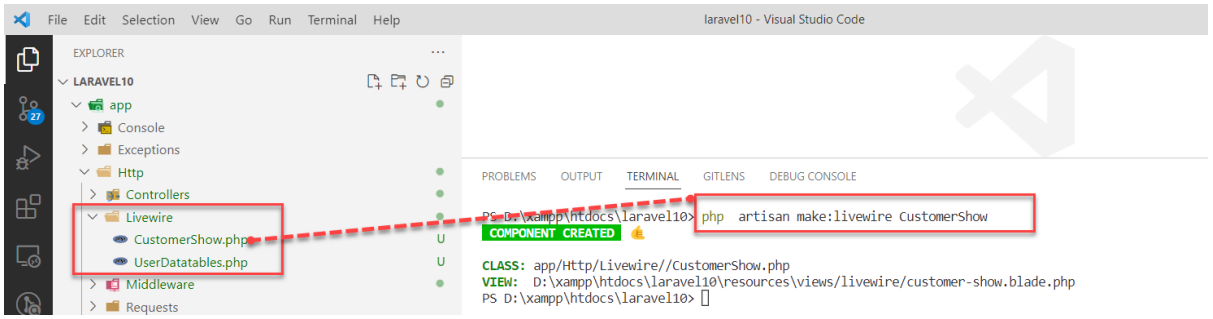
หลังจากที่กด Enter แล้วหากมีข้อความ COMPONENT CREATED Laravel จะมีการสร้าง Component Class และ View มาให้ โดยจะเก็บไฟล์ที่เป็น Class Livewire ไว้ในโฟลเดอร์ app/Http/Livewire/...

CLASS : app/Http/Livewire/CustomerShow.php

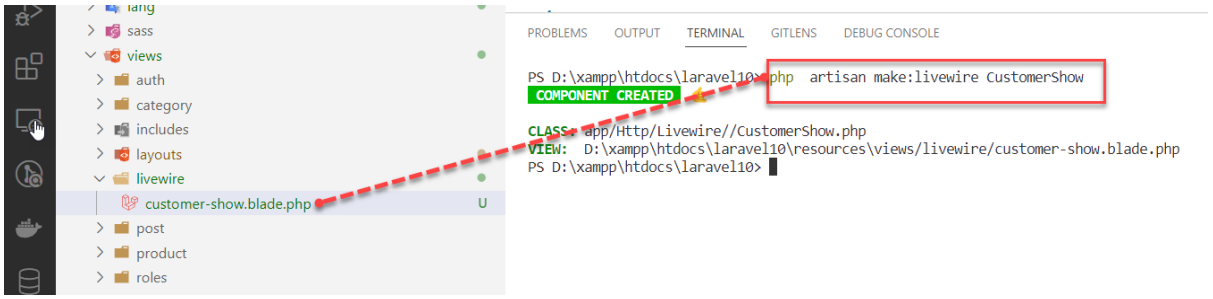
และจะเก็บไฟล์ที่เป็น view Livewire ไว้ในโฟลเดอร์ resources/views/livewire/...

VIEW : resources/views/livewire/customer-show.blade.php

Component CLASS



VIEW



7.เปิดไฟล์ CustomerShow.php ในโฟลเดอร์ app/Http/Livewire/... แล้วเพิ่มโค้ด ดังนี้

```
//app/Http/Livewire/CustomerShow.php

<?php
namespace App\Http\Livewire;
use Livewire\WithPagination;
use App\Models\Customer;
use Livewire\Component;

class CustomerShow extends Component
{
    use WithPagination;

    protected $paginationTheme = 'bootstrap';

    public $customer_name, $customer_email, $customer_tel, $customer_id;
    public $search = '';

    protected function rules()
    {
        return [
            'customer_name' => 'required|string',
            'customer_email' => ['required','email'],
            'customer_tel' => 'required|string',
        ];
    }
```

มีต่อ...

```
    protected function messages(){
        return [
            'customer_name.required' => 'กรุณาระบุชื่อลูกค้า',
            'customer_email.required' => 'กรุณาระบุอีเมล',
            'customer_tel.required' => 'กรุณาระบุเบอร์โทรศัพท์',
        ];
    }

    public function updated($fields)
    {
        $this->validateOnly($fields);
    }

    public function saveCustomer()
    {
        $validatedData = $this->validate();

        Customer::create($validatedData);
        session()->flash('message','เพิ่มลูกค้าเรียบร้อยแล้ว');
        $this->resetInput();
        $this->dispatchBrowserEvent('close-modal');
    }
    //แก้ไขข้อมูลลูกค้า
    public function editCustomer(int $customer_id)
    {
        $customer = Customer::find($customer_id);
        if($customer){

            $this->customer_id = $customer->id;
            $this->customer_name = $customer->customer_name;
            $this->customer_email = $customer->customer_email;
            $this->customer_tel = $customer->customer_tel;
        }else{
            return redirect()->to('/customers');
        }
    }
    //แก้ไขข้อมูลลูกค้า
    public function updateCustomer()
    {
        $validatedData = $this->validate();

        Customer::where('id',$this->customer_id)->update([
            'customer_name' => $validatedData['customer_name'],
            'customer_email' => $validatedData['customer_email'],
            'customer_tel' => $validatedData['customer_tel']
        ]);
        session()->flash('message','แก้ไขข้อมูลลูกค้าเรียบร้อยแล้ว');
        $this->resetInput();
        $this->dispatchBrowserEvent('close-modal');
    }
```

มีต่อ...

```
public function deleteCustomer(int $customer_id)
    {
        $this->customer_id = $customer_id;
    }

    public function destroyCustomer()
    {
        Customer::find($this->customer_id)->delete();
        session()->flash('message','ลูกข้อมูลลูกค้าเรียบร้อยแล้ว');
        $this->dispatchBrowserEvent('close-modal');
    }

    public function closeModal()
    {
        $this->resetInput();
    }

    public function resetInput()
    {
        $this->customer_name = '';
        $this->customer_email = '';
        $this->customer_tel = '';
    }

    public function render()
    {
        $customers = Customer::where('customer_name', 'like','%'.$this->search.'%')
                    ->orderBy('id','DESC')->paginate(5);
        return view('livewire.customer-show', ['customers' => $customers]);
    }
}
```

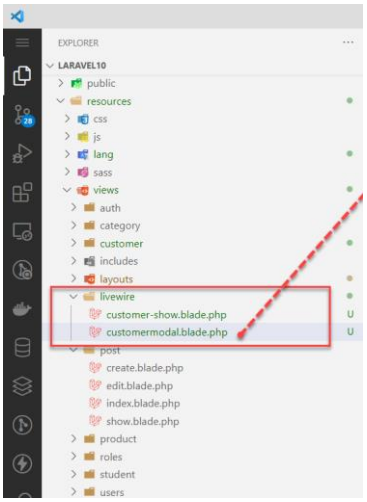
8.เปิดไฟล์ customer-show.blade.php ในโฟลเดอร์ resources/views/livewire/... แล้วเพิ่มโค้ดดังนี้

```
// resources/views/livewire/customer-show.blade.php
<div>
    @include('livewire.customermodal')
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                @if (session()->has('message'))
                    <h5 class="alert alert-success">{{ session('message') }}</h5>
                @endif
                <div class="card">
                    <div class="card-header">
                        <h4>Laravel Livewire CRUD ด้วย Bootstrap Modal
                            <input type="search" wire:model="search" class="form-control
float-end mx-2"placeholder="ค้นหา..." style="width: 250px" />
                            <button type="button" class="btn btn-primary float-end" data-bs-
toggle="modal" data-bs-target="#customerModal">เพิ่มลูกค้าใหม่</button>
                        </h4>
                    </div>
                    <div class="card-body">
                        <table class="table table-borderd table-striped">
                            <thead>
                                <tr>
                                    <th>ID</th>
                                    <th>ชื่อลูกค้า</th>
                                    <th>อีเมล</th>
                                    <th>เบอร์โทร</th>
                                    <th>จัดการ</th>
                                </tr>
                            </thead>
                            <tbody>
                                @forelse ($customers as $customer)
                                    <tr>
                                        <td>{{ $customer->id }}</td>
                                        <td>{{ $customer->customer_name }}</td>
                                        <td>{{ $customer->customer_email }}</td>
                                        <td>{{ $customer->customer_tel }}</td>
                                        <td>
   <button type="button" data-bs-toggle="modal" data-bs-target="#updateCustomerModal"
wire:click="editCustomer({{ $customer->id }})" class="btn btn-primary">แก้ไข</button>
  <button type="button" data-bs-toggle="modal" data-bs-target="#deleteCustomerModal"
wire:click="deleteCustomer({{ $customer->id }})"class="btn btn-danger">ลบ</button>
                                        </td>
                                    </tr>
                                @empty
                                    <tr>
                                        <td colspan="5">ไม่พบข้อมูล</td>
                                    </tr>
                                @endforelse
```

```
                        </tbody>
                        </table>
                        <div>
                            {{ $customers->links() }}
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

ดาวน์โหลดโค้ดตัวอย่าง https://github.com/leksoft/laravel9/blob/main/resources/views/livewire/customer-show.blade.php

9.สร้างไฟล์ใหม่ไว้ที่โฟลเดอร์ resources/views/livewire/...โดยตั้งชื่อไฟล์ customermodal.blade.php สำหรับเรียกใช้ modal ฟอร์มสำหรับเพิ่มข้อมูล ลบข้อมูล และแก้ไขข้อมูล



10.เปิดไฟล์ customermodal.blade.php ในโฟลเดอร์ resources/views/livewire/... แล้วเพิ่มโค้ด ดังนี้

```
// resources/views/livewire/customermodal.blade.php
<!-- Insert Customer Modal -->
<div wire:ignore.self class="modal fade" id="customerModal" tabindex="-1" aria-
labelledby="customerModalLabel"
    aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="customerModalLabel">เพิ่มลูกค้าใหม่</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"
                    wire:click="closeModal"></button>
            </div>
            <form wire:submit.prevent="saveCustomer">
                <div class="modal-body">
                    <div class="mb-3">
                        <label>ชื่อลูกค้า</label>
                        <input type="text" wire:model="customer_name" class="form-
control">
                        @error('customer_name')
                            <span class="text-danger">{{ $message }}</span>
                        @enderror
                    </div>
                    <div class="mb-3">
                        <label>อีเมล</label>
                        <input type="text" wire:model="customer_email" class="form-
control">
                        @error('customer_email')
                            <span class="text-danger">{{ $message }}</span>
                        @enderror
                    </div>
                    <div class="mb-3">
                        <label>เบอร์โทร</label>
                        <input type="text" wire:model="customer_tel" class="form-
control">
                        @error('customer_tel')
                            <span class="text-danger">{{ $message }}</span>
                        @enderror
                    </div>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary"
wire:click="closeModal"
                        data-bs-dismiss="modal">ปิด</button>
                    <button type="submit" class="btn btn-primary">บันทึก</button>
                </div>
            </form>
        </div>
    </div>
</div>
```

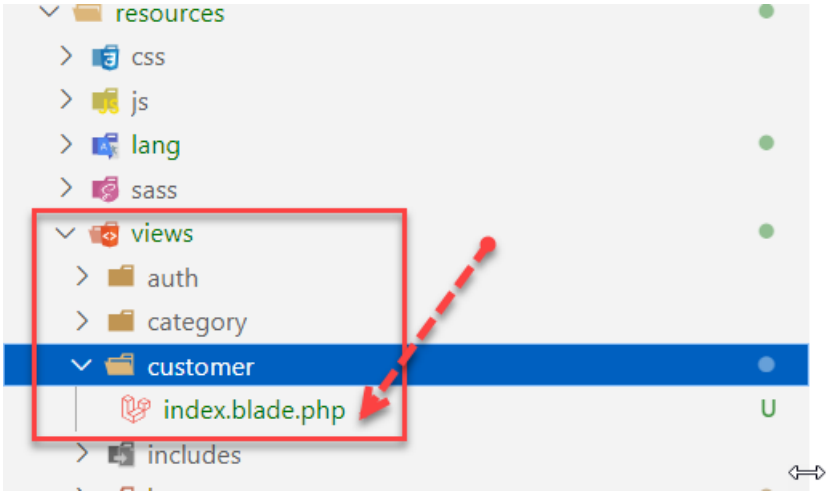
```
// resources/views/livewire/customermodal.blade.php
<!-- Update Customer Modal -->
<div wire:ignore.self class="modal fade" id="updateCustomerModal" tabindex="-1"
    aria-labelledby="updateCustomerModalLabel" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
            <h5 class="modal-title" id="updateCustomerModalLabel">แก้ไขข้อมูลลูกค้า</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal"
wire:click="closeModal"
                    aria-label="Close"></button>
            </div>
            <form wire:submit.prevent="updateCustomer">
                <div class="modal-body">
                    <div class="mb-3">
                        <label>ชื่อลูกค้า</label>
                        <input type="text" wire:model="customer_name"
class="form-control">
                        @error('customer_name')
                            <span class="text-danger">{{ $message }}</span>
                        @enderror
                    </div>
                    <div class="mb-3">
                        <label>อีเมล</label>
                        <input type="text" wire:model="customer_email"
class="form-control">
                        @error('customer_email')
                            <span class="text-danger">{{ $message }}</span>
                        @enderror
                    </div>
                    <div class="mb-3">
                        <label>เบอร์โทร</label>
                        <input type="text" wire:model="customer_tel"
class="form-control">
                        @error('customer_tel')
                            <span class="text-danger">{{ $message }}</span>
                        @enderror
                    </div>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary"
wire:click="closeModal"
                        data-bs-dismiss="modal">ปิด</button>
                    <button type="submit" class="btn btn-primary">แก้ไข</button>
                </div>
            </form>
        </div>
    </div>
</div>
```

```
// resources/views/livewire/customermodal.blade.php
<!-- Delete Customer Modal -->
<div wire:ignore.self class="modal fade" id="deleteCustomerModal" tabindex="-1"
    aria-labelledby="deleteCustomerModalLabel" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
             <h5 class="modal-title" id="deleteCustomerModalLabel">ลบข้อมูลลูกค้า</h5>
                <button type="button" class="btn-close" data-bs-dismiss="modal"
wire:click="closeModal"
                    aria-label="Close"></button>
            </div>
            <form wire:submit.prevent="destroyCustomer">
                <div class="modal-body">
                    <h4>ยืนยันการลบข้อมูลหรือไม่ ?</h4>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary"
wire:click="closeModal"
                        data-bs-dismiss="modal">ปิด</button>
                    <button type="submit" class="btn btn-primary">ยืนยัน</button>
                </div>
            </form>
        </div>
    </div>
</div>
```

ดาวน์โหลดโค้ดตัวอย่าง

https://github.com/leksoft/laravel9/blob/main/resources/views/livewire/customermodal.blade.php

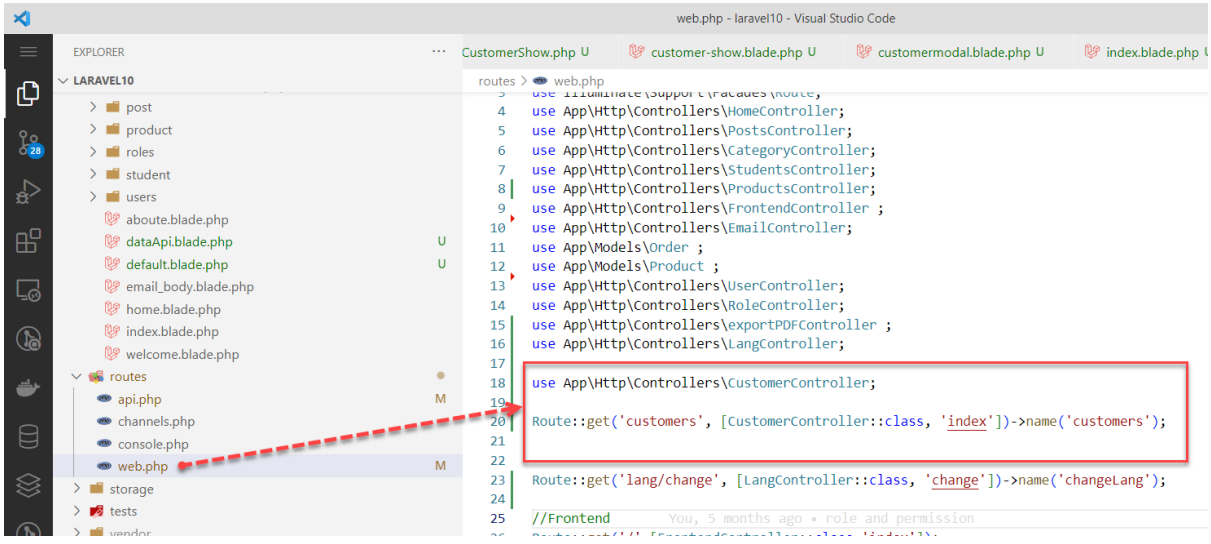
11.สร้างโฟลเดอร์ customer และเพิ่มไฟล์ชื่อ index.blade.php เก็บไว้ที่ resources/views/... ดังนี้



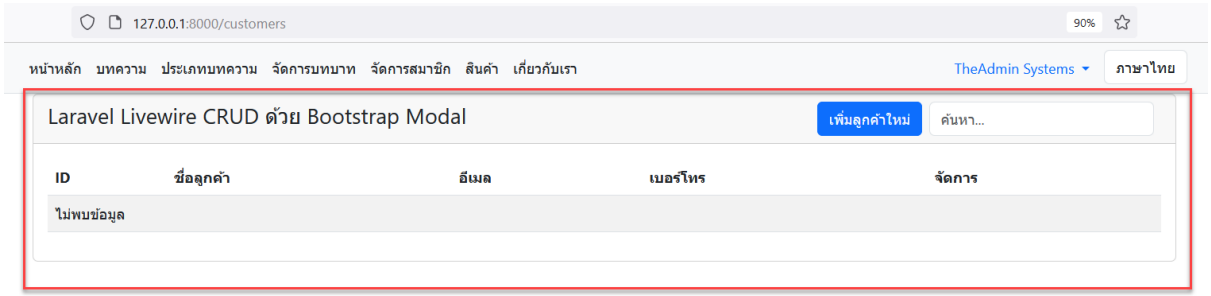
เปิดไฟล์ index.blade.php แล้วเพิ่มโค้ด ดังนี้

```
// resources/views/customer/index.blade.php
@extends('layouts.frontend')

@section('content')
    <div>
        <livewire:customer-show>
    </div>
@endsection

@section('js_before')
    <script>
        window.addEventListener('close-modal', event => {

            $('#customerModal').modal('hide');
            $('#updateCustomerModal').modal('hide');
            $('#deleteCustomerModal').modal('hide');
        })
    </script>
@endsection
```

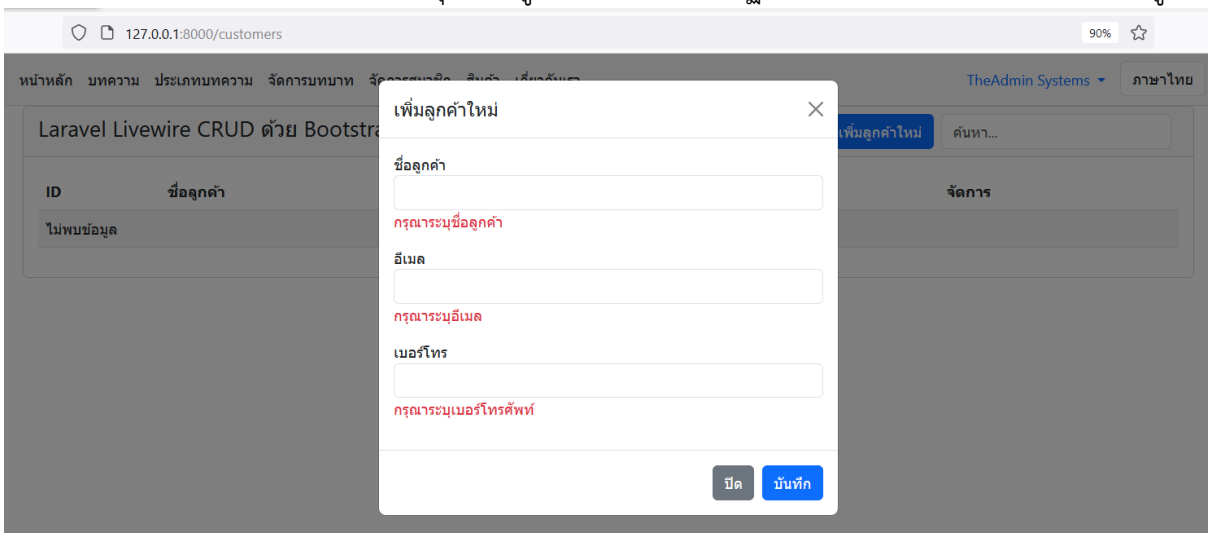
12.เปิดไฟล์ routes/web.php แล้วเพิ่มโค้ด เพื่อกำหนดเส้นทางการทำงาน ดังนี้

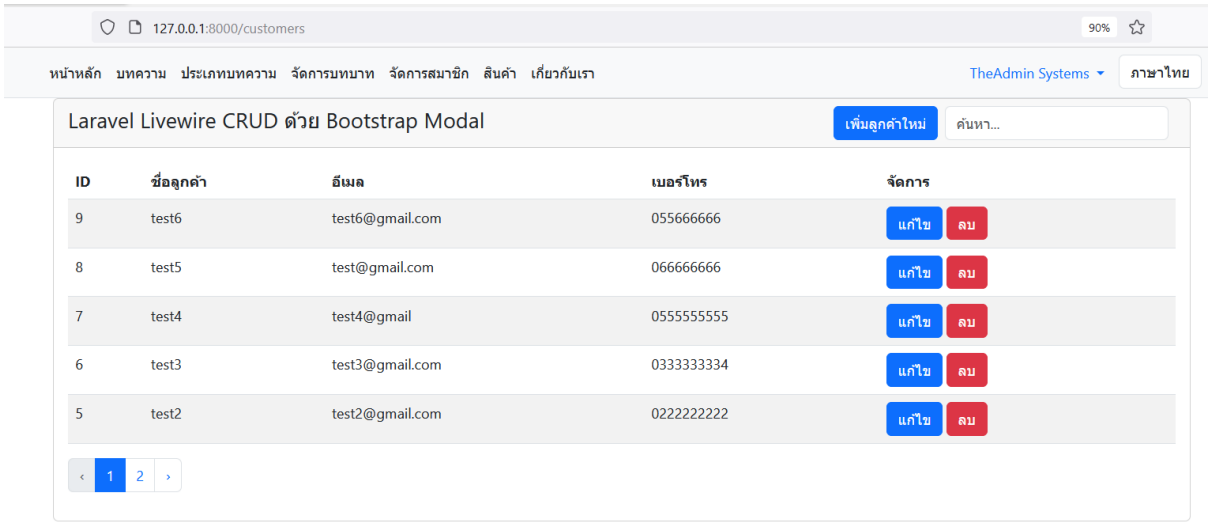


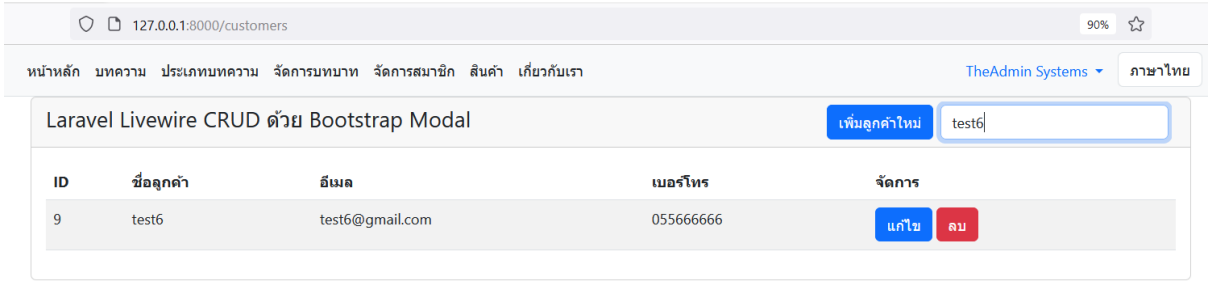
บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/customers ที่ Browser



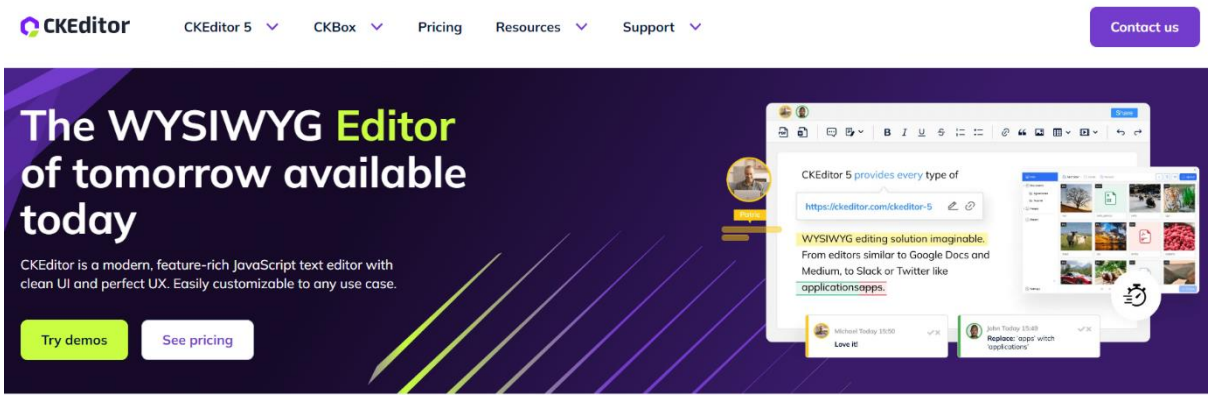
เมื่อลองทดสอบโดยการคลิกที่ปุ่ม เพิ่มลูกค้าใหม่ ก็จะปรากฏ Modal แสดงขึ้นมาให้เรากรอกข้อมูล



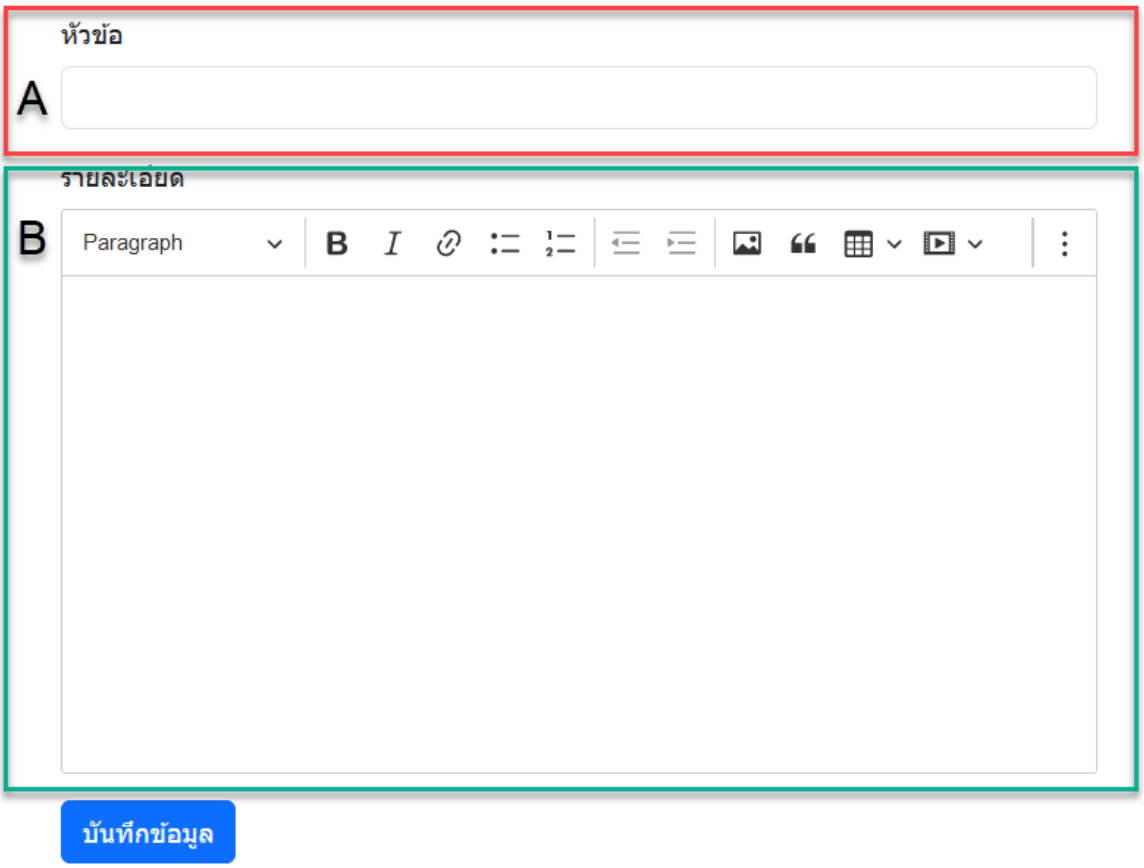




การใช้งาน Ckeditor 5 (Text Editor WYSIWYG) ร่วมกับ Laravel 10



CKEditor คือ เครื่องมือที่ช่วยให้ผู้ใช้งานสามารถเขียนและจัดการข้อความลงบนเว็บไซต์ได้อย่างสะดวกสบายโดยที่ผู้ใช้ไม่ต้องจำโค้ดหรือเขียนโค้ดที่เป็น tag HTML โดยปกติจะใช้ร่วมกับ <input> หรือ <textarea> ซึ่งจะสามารถให้ผู้ใช้กำหนดสีข้อความ ใส่รูปภาพ ปรับขนาดตัวอักษร ใส่ลิงค์ ทำนองเดียวกัน Word เลยครับ ปัจจุบัน ckeditor ถึงเวอร์ชัน 5 ฉะนั้นเราจะเลือกใช้เวอร์ชัน 5 เลยนะครับ

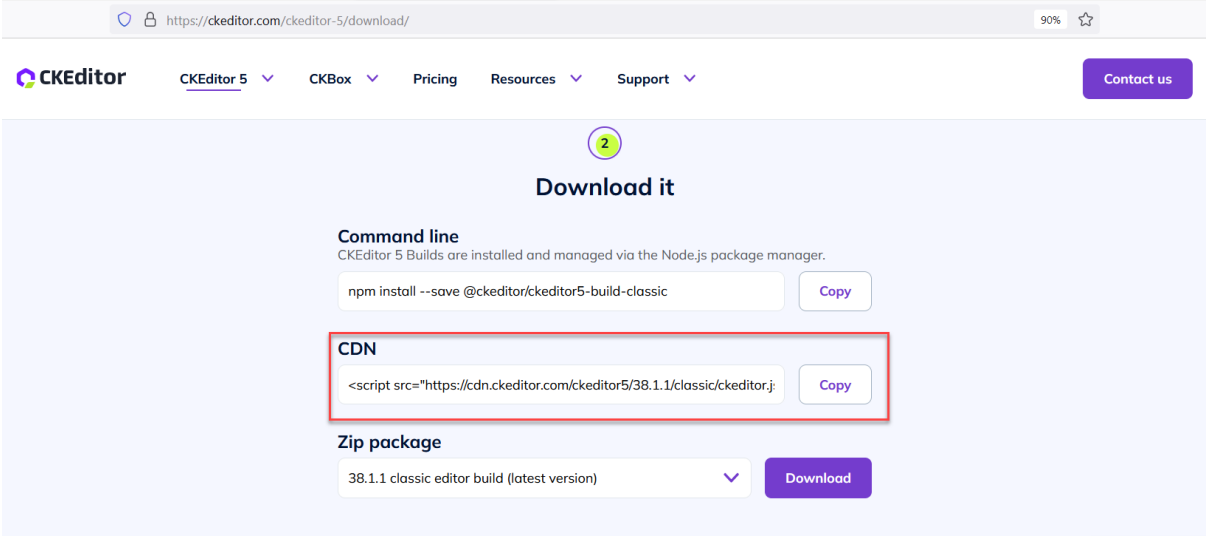


A ช่องกรอกข้อมูลที่ไม่ได้ใช้ ckeditor

B ช่องกรอกข้อมูลที่มีการนำ ckeditor มาใช้

วิธีการใช้งานมีขั้นตอนดังนี้ครับ

1.เราจะใช้วิธีการเรียกใช้งานผ่าน CDN (Content Delivery Network) ไปที่เว็บไซต์ https://ckeditor.com/ckeditor-5/download แล้วคลิกที่ปุ่ม copy เพื่อทำการคัดลอก script ไปใช้



2.ไปที่หน้าที่ต้องการจะเรียกใช้งาน ในที่นี้เราจะใช้กับ post หรือ บทความ ให้เปิดไฟล์ create.blade.php ในโฟลเดอร์ resources/views/post/... แล้วเพิ่มโค้ดเข้าไปในส่วนของ js_before ดังนี้

```
// resources/views/post/create.blade.php

@section('js_before')
<script src="https://cdn.ckeditor.com/ckeditor5/38.1.1/classic/ckeditor.js"></script>

@endsection
```

และเพิ่ม script สำหรับเปิดใช้งานกล่องข้อความของ ckeditor ดังนี้

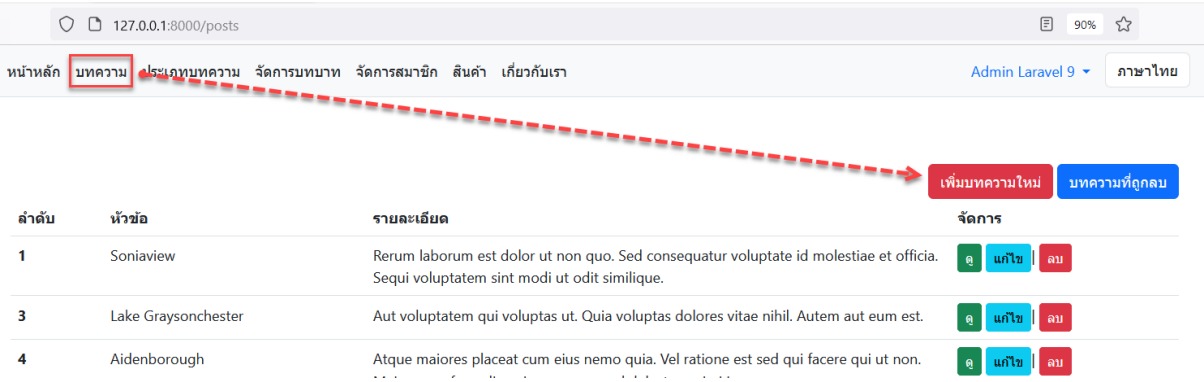
```
// resources/views/post/create.blade.php

@section('js_before')
<script src="https://cdn.ckeditor.com/ckeditor5/38.1.1/classic/ckeditor.js"></script>
 <script>

        ClassicEditor
            .create(document.querySelector('#post_detail'))
            .catch(error => {
                console.error(error);
            });
    </script>


@endsection
```

จากนั้นไปกำหนด id เพิ่มในส่วนของ <textarea> ดังนี้

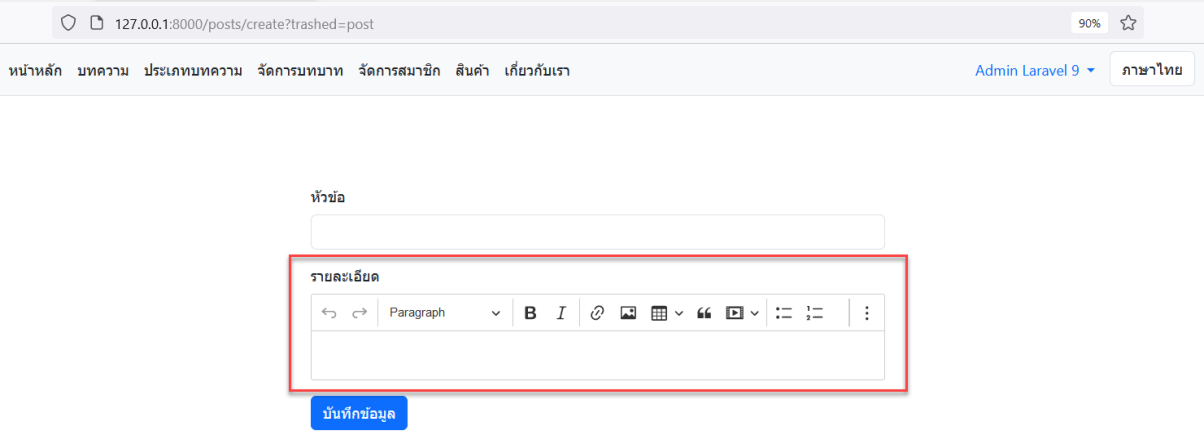
```
// resources/views/post/create.blade.php

<textarea type="text" class="form-control" name="post_detail" id="post_detail" rows="3"></textarea>
```

จะได้โค้ดทั้งหมด ดังนี้

```
// resources/views/post/create.blade.php

@extends('layouts.frontend')
@section('title', 'เพิ่มรายการใหม่')
@section('css_before')
@endsection
@section('content')
    <section class="py-5 container">
        <div class="row py-lg-5">
            <div class="col-lg-6 col-md-8 mx-auto">
                <form action="{{ route('store') }}" method="POST">
                    @csrf
                    <div class="mb-3">
                        <label class="form-label">หัวข้อ</label>
                        <input type="text" name="post_title" class="form-control">
                    </div>
                    <div class="mb-3">
                        <label class="form-label">รายละเอียด</label>
                        <textarea type="text" class="form-control" name="post_detail"
id="post_detail" rows="3"></textarea>
                    </div>
                    <div class="mb-3">
                        <button type="submit" class="btn btn-primary">บันทึกข้อมูล</button>
                    </div>
                </form>

            </div>
        </div>
    </section>
@endsection
@section('js_before')
    <script src="https://cdn.ckeditor.com/ckeditor5/38.1.1/classic/ckeditor.js"></script>
    <script>
        ClassicEditor
            .create(document.querySelector('#post_detail'))
            .catch(error => {
                console.error(error);
            });
    </script>
@endsection
```



เพียงเท่านี้เราก็จะได้ช่องสำหรับกรอกรายละเอียดพร้อมกับ Tool ต่าง ๆ ให้เราได้เลือกใช้แล้ว



สามารถปรับช่องกรอกข้อความให้กว้างขึ้นโดยเพิ่มโค้ดเข้าไปในส่วน css_before ดังนี้

การใช้งาน Ckeditor 5 กับการอับโหลดไฟล์

กรณีที่เราอยากจะอับโหลดไฟล์ เช่น รูปภาพประกอบบทความ ก็สามารถทำได้ง่าย โดยเขียนโค้ดเพิ่มดังนี้

1.ให้เปิดไฟล์ create.blade.php ในโฟลเดอร์ resources/views/post/... แล้วแก้ไขโค้ดเข้าไปในส่วนของ js_before ดังนี้

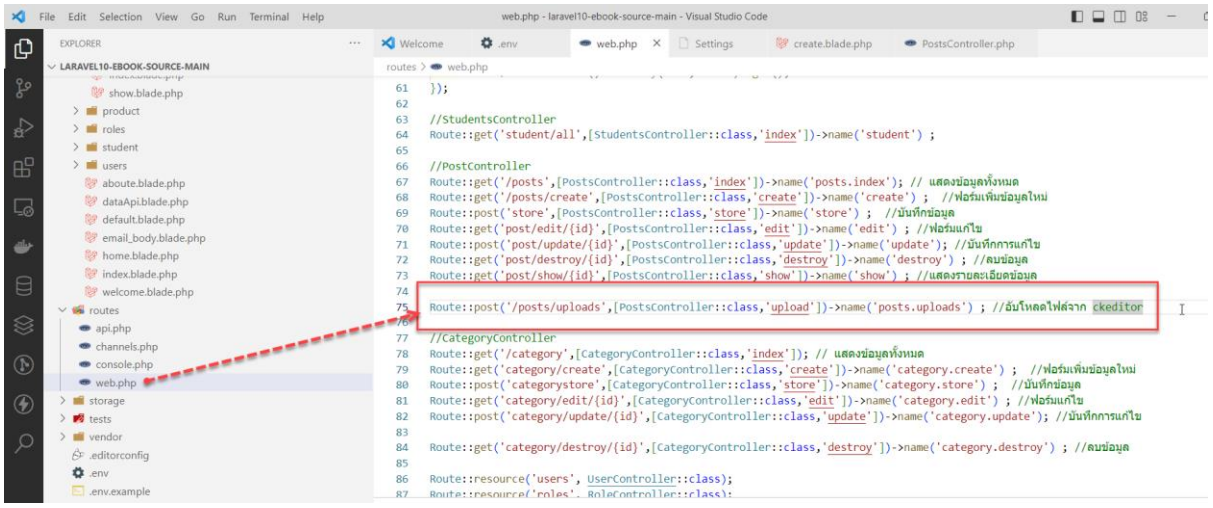
จากโค้ดเดิม

```
// resources/views/post/create.blade.php

@section('js_before')
<script src="https://cdn.ckeditor.com/ckeditor5/38.1.1/classic/ckeditor.js"></script>
 <script>

        ClassicEditor
            .create(document.querySelector('#post_detail'))
            .catch(error => {
                console.error(error);
            });
    </script>


@endsection
```

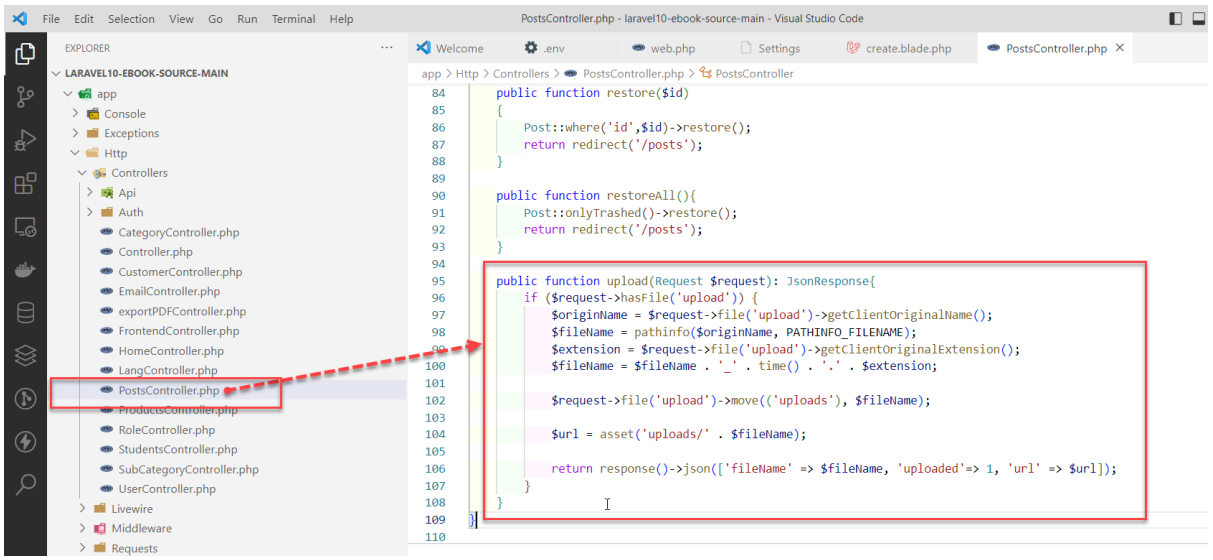
แก้ไขใหม่

```
// resources/views/post/create.blade.php

@section('js_before')
<script src="https://cdn.ckeditor.com/ckeditor5/38.1.1/classic/ckeditor.js"></script>
 <script>

        ClassicEditor
            .create(document.querySelector('#post_detail'), {
                ckfinder: {
                    uploadUrl: '{{ route('posts.uploads') . '?_token=' .
csrf_token() }}',
                }
            })
            .catch(error => {

            });    </script>


@endsection
```

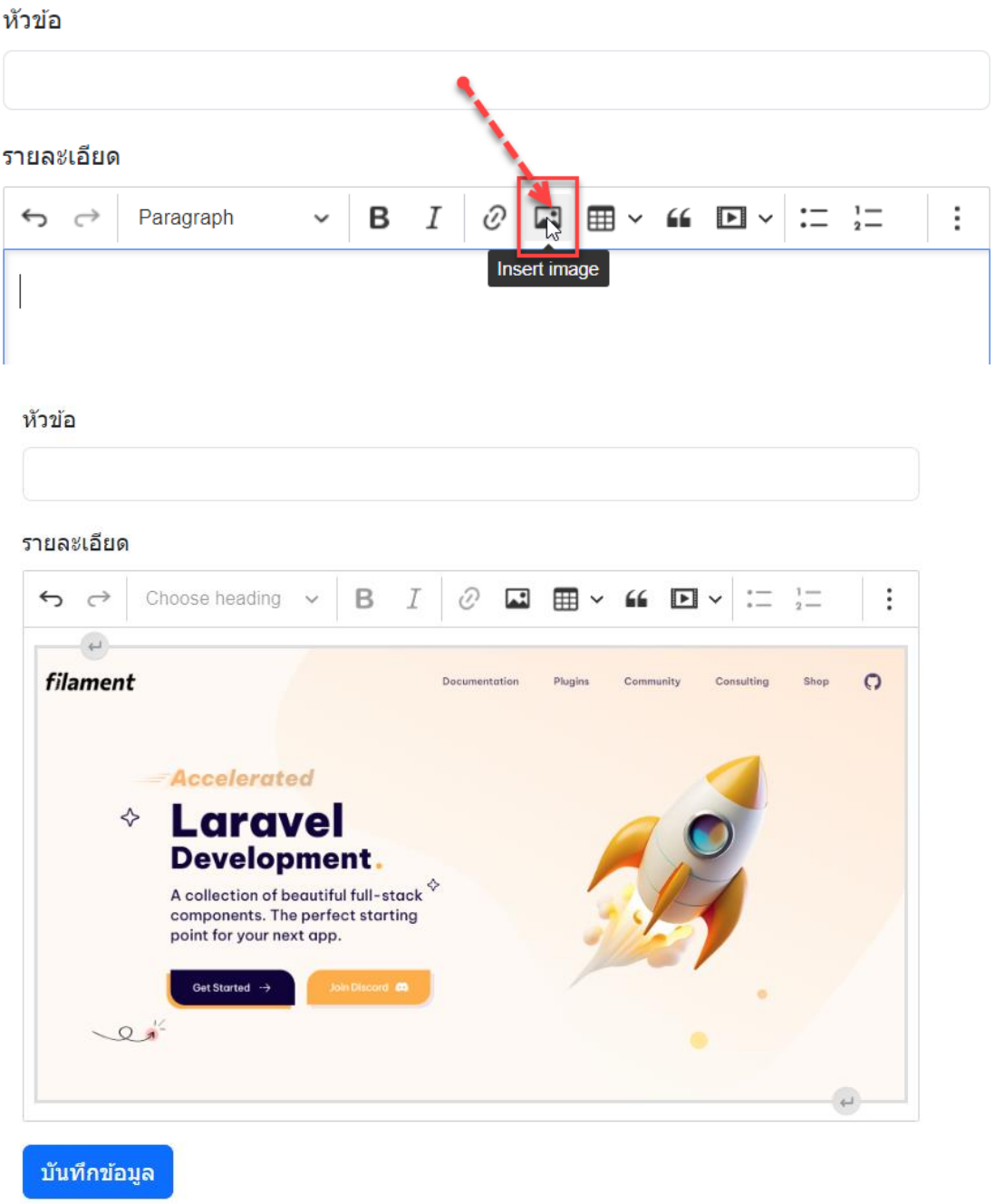
2.กำหนดเส้นทาง URL สำหรับการอัปโหลดไฟล์ โดยตั้งชื่อ routes ว่า posts.upload และกำหนดเมธอดเป็น post ในไฟล์ routes.php ดังนี้



3. เปิดไฟล์ PostsController.php ในโฟลเดอร์ app\Http\Controllers\... แล้วเพิ่มเมธอดใหม่ต่อท้ายเมธอด restoreAll ดังนี้



4.บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/ ที่ Browser คลิกที่เมนู **บทความ** แล้วคลิกปุ่มเพิ่มบทความ แล้วกดที่รูปภาพสำหรับอับโหลดไฟล์



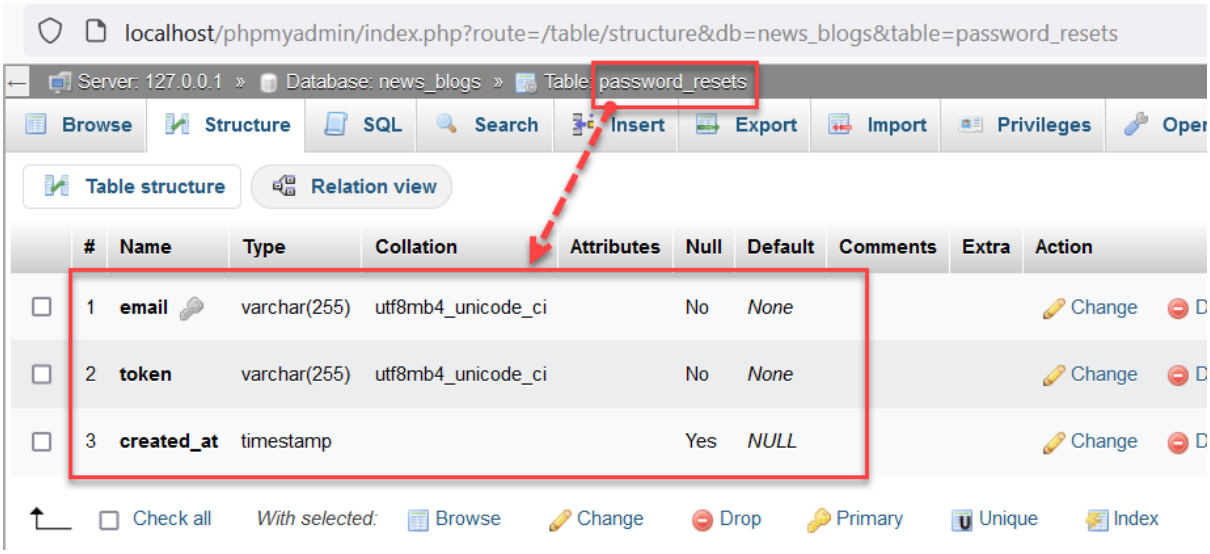
เพียงเท่านี้ก็จะสามารถอับโหลดไฟล์ร่วมกับ ckeditor ได้แล้วครับ

โค้ดตัวอย่าง https://github.com/leksoft/laravel10-ebook-source/tree/uploadfile

ระบบลืมรหัสผ่านและรีเซ็ตรหัสผ่าน

ในบทเรียนนี้จะได้เรียนรู้เกี่ยวกับขั้นตอนและวิธีการรีเซตรหัสผ่านในกรณีที่ผู้ใช้งานเกิดจำรหัสเข้าระบบไม่ได้ก็สามารถขอรีเซตรหัสผ่านเพื่อกำหนดรหัสผ่านใหม่ได้ ปกติแล้วเวลาที่เราทำระบบ Login จากบทเรียนที่ผ่านมาเรื่อง ระบบล็อกอินและระบบลงทะเบียน (Auth Login and Registration) ในหน้า 234 นั้น จะมีระบบลืมรหัสผ่านมาให้เราแล้วซึ่งสามารถใช้งานได้เลย แต่ถ้าเราอยาก Custom ระบบรีเซตรหัสผ่านด้วยตัวเองเราก็สามารถทำได้เหมือนกัน

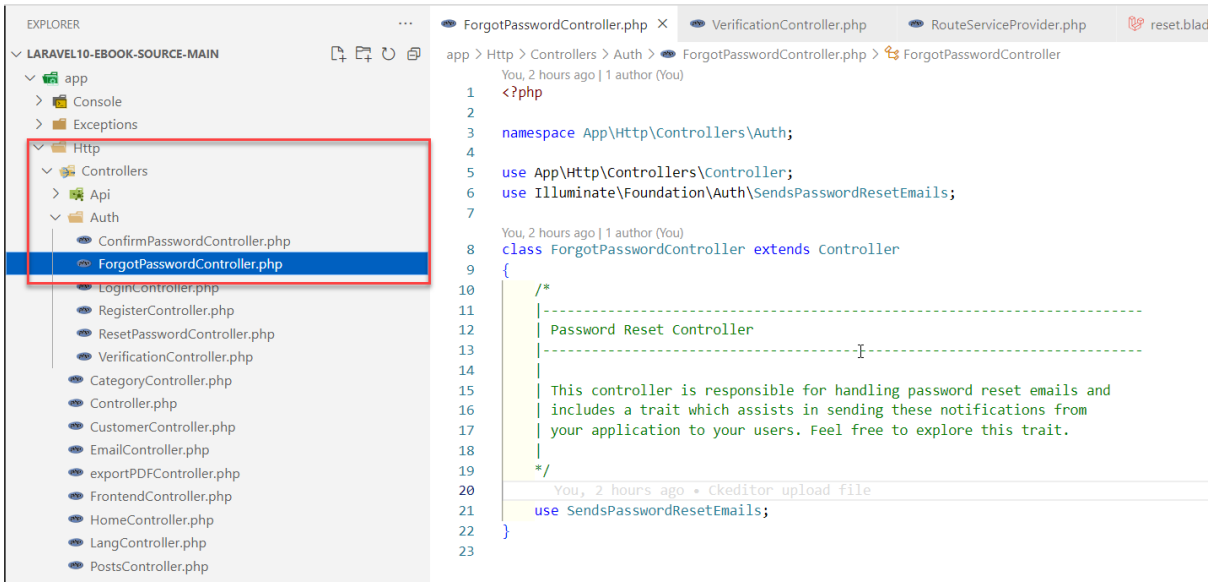
1.ก่อนอื่นเลยให้ผู้อ่านตรวจสอบตารางในฐานข้อมูลก่อนครับ ว่ามีตาราง password_resets ในฐานข้อมูลหรือยัง ปกติผู้อ่านน่าจะมีแล้ว ตอนที่สร้างระบบล็อกอิน ถ้าปรากฏไม่มีให้ผู้อ่านสร้างตารางและระบุโครงสร้างข้อมูลตามนี้เลยครับ



2.สร้าง route สำหรับกำหนดเส้นทาง URL โดยเปิดไฟล์ routes.php แล้วเพิ่มโค้ด ดังนี้

```
18
19  use App\Http\Controllers\Auth\ForgotPasswordController;
20
21  Route::get('forget-password', [ForgotPasswordController::class, 'showForgetPasswordForm'])->name('forget.password.get');
22  Route::post('forget-password', [ForgotPasswordController::class, 'submitForgetPasswordForm'])->name('forget.password.post');
23  Route::get('reset-password/{token}', [ForgotPasswordController::class, 'showResetPasswordForm'])->name('reset.password.get');
24  Route::post('reset-password', [ForgotPasswordController::class, 'submitResetPasswordForm'])->name('reset.password.post');
25
```

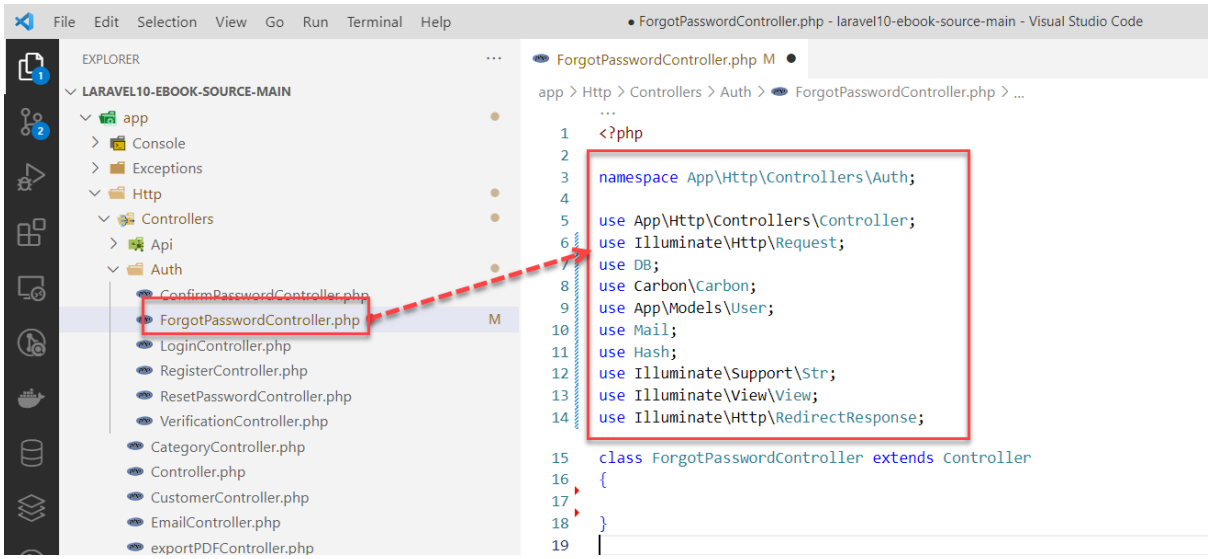
ในไฟล์ routes.php ที่เพิ่มเข้าไปใหม่นั้น จะมีการเรียก namespace จาก App\Http\Controllers\Auth\ForgotPasswordController จะใช้ไฟล์ ForgotPasswordController.php เป็นไฟล์หลักในการ custom โค้ดใหม่



3.เปิดไฟล์ ForgotPasswordController.php แล้วเพิ่ม namespace และเมธอดเข้าไป โดยใน routes.php ที่เราได้เพิ่มจะมี 4 เมธอด นั่นคือ

1. showForgetPasswordForm สำหรับแสดงฟอร์มให้กรอกอีเมล
2. submitForgetPasswordForm สำหรับบันทึกการรีเซตรหัสผ่านส่งเข้าเมลเพื่อให้ผู้ใช้กดยืนยัน
3. showResetPasswordForm สำหรับแสดงฟอร์มให้กรอกรหัสผ่านใหม่
4. submitResetPasswordForm สำหรับบันทึกการรีเซตรหัสผ่านใหม่

เพิ่ม namespace

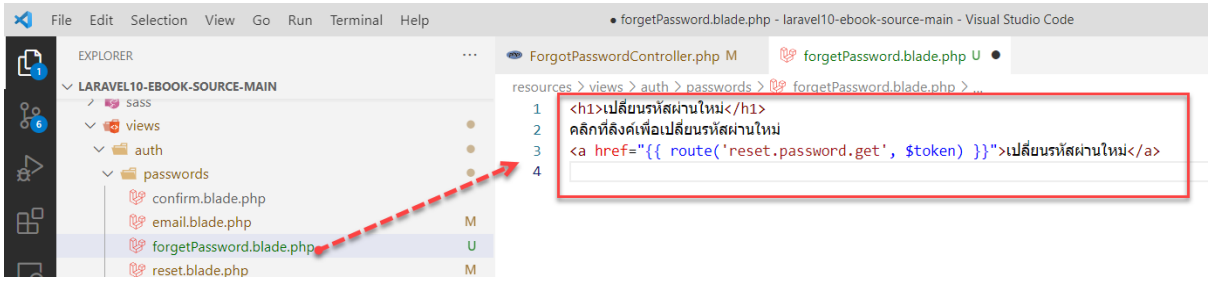


เพิ่มเมธอด

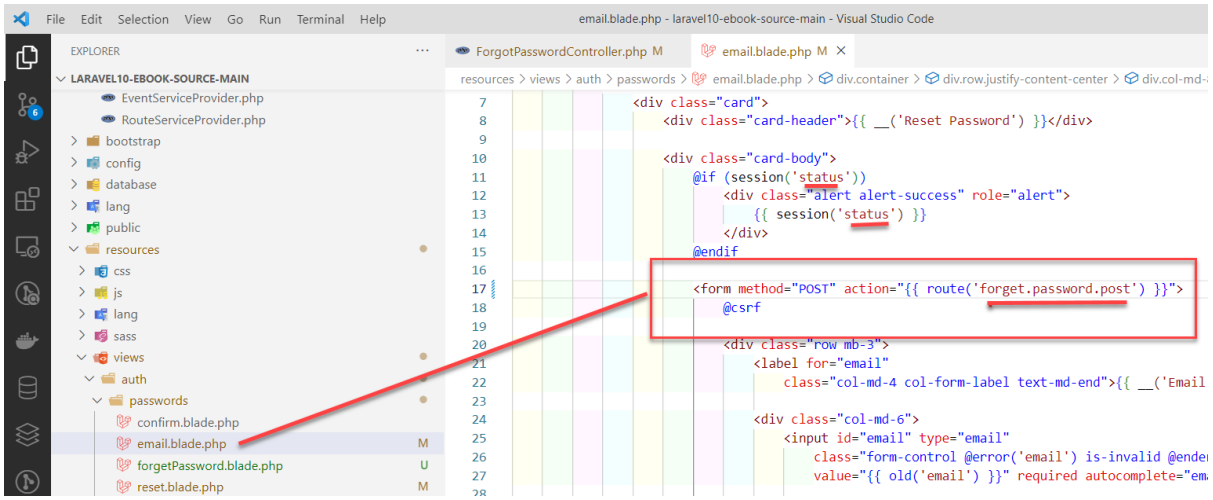
```
// App\Http\Controllers\Auth\ForgotPasswordController.php

class ForgotPasswordController extends Controller
{
    //สำหรับแสดงฟอร์มให้กรอกอีเมล
    public function showForgetPasswordForm(): View
    {
        return view('auth.passwords.email');
    }
    //submit email
    public function submitForgetPasswordForm(Request $request):
RedirectResponse
    {
        $request->validate([
            'email' => 'required|email|exists:users',
        ]);

        $token = Str::random(64);

        DB::table('password_resets')->insert([
            'email' => $request->email,
            'token' => $token,
            'created_at' => Carbon::now()
        ]);
            //ส่งลิงค์เข้าเมล
        Mail::send('auth.passwords.forgetPassword', ['token' => $token],
function($message) use($request){
            $message->to($request->email);
            $message->subject('Reset Password');
        });

        return back()->with('status', 'ระบบได้ส่งลิงค์เข้าอีเมล์ของท่านแล้ว โปรดตรวจสอบอีเมลของท่าน!');

    }

    public function showResetPasswordForm($token): View
    {
        return view('auth.passwords.reset', ['token' => $token]);
    }
```

```
// App\Http\Controllers\Auth\ForgotPasswordController.php

public function submitResetPasswordForm(Request $request): RedirectResponse
    {
            $request->validate([
            'email' => 'required|email|exists:users',
            'password' => 'required|string|min:6|confirmed',
            'password_confirmation' => 'required'
            ]);

        $updatePassword = DB::table('password_resets')
                            ->where([
                                'email' => $request->email,
                                'token' => $request->token
                            ])->first();

        if(!$updatePassword){
            return back()->withInput()->with('error', 'Invalid token!');
        }
            $user = User::where('email', $request->email)
                    ->update(['password' => Hash::make($request->password)]);

            DB::table('password_resets')->where(['email'=> $request->email])-
>delete();

        return redirect('/login')->with('message', 'เปลี่ยนรหัสผ่านของท่านเรียบร้อยแล้ว!');
    }
}
```

views/auth/password/... เพิ่มโค้ดเข้าไป ดังนี้



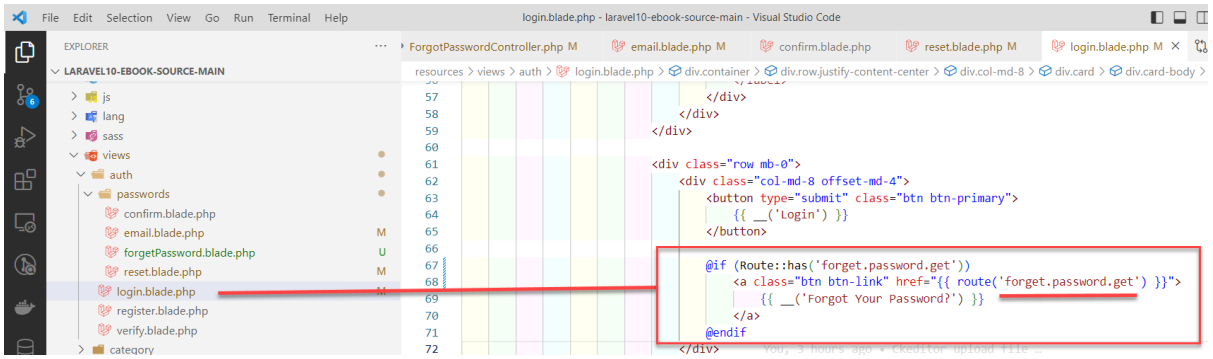
5.เปิดไฟล์ email.blade.php ในโฟลเดอร์ views/auth/passwords/... แล้วแก้ไขโค้ด ดังนี้



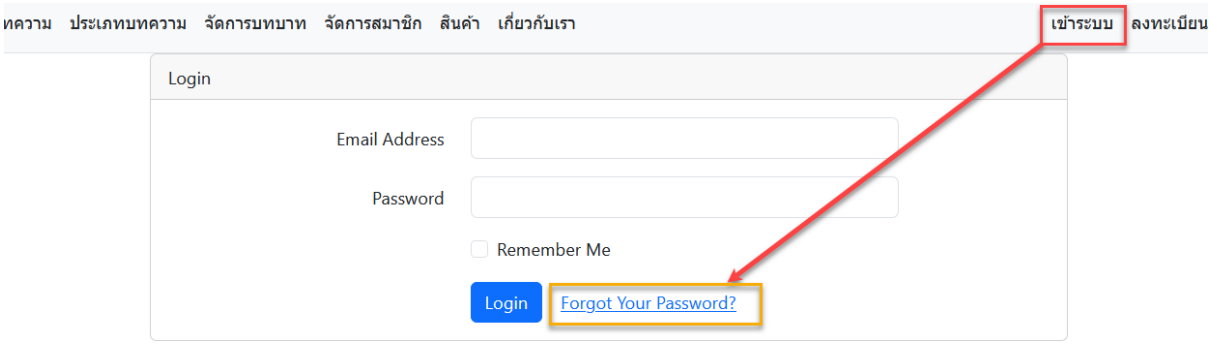
6.เปิดไฟล์ reset.blade.php ในโฟลเดอร์ views/auth/passwords/... แล้วแก้ไขโค้ด ดังนี้



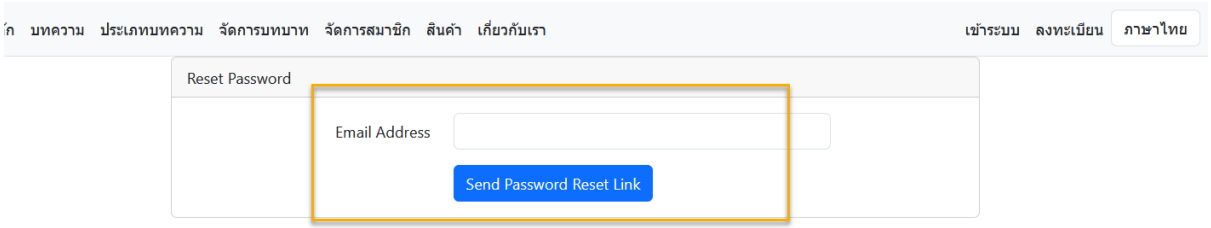
7.เปิดไฟล์ login.blade.php ในโฟลเดอร์ views/auth/... แล้วแก้ไขโค้ด ดังนี้



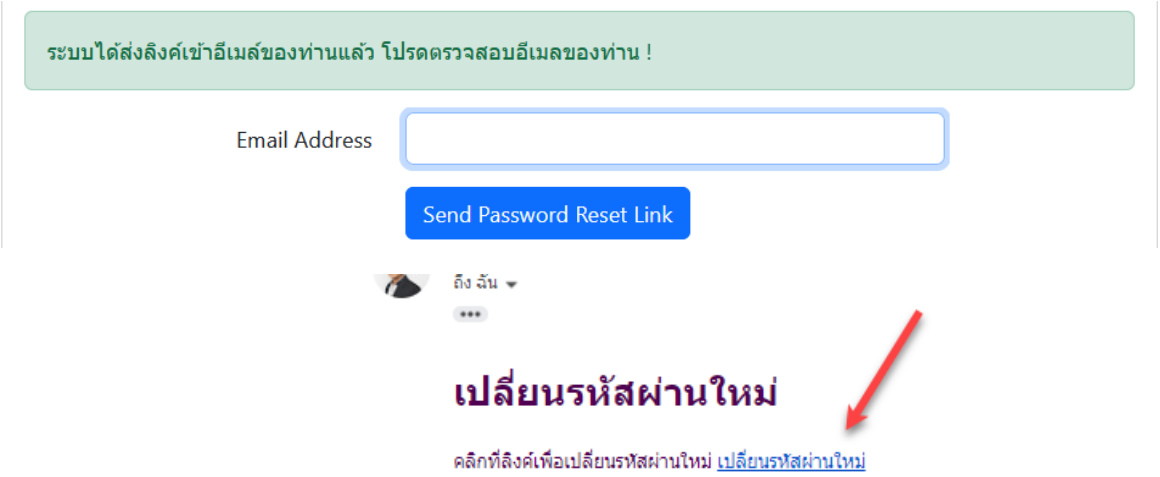
8.บันทึกไฟล์ทั้งหมดแล้วทดสอบดูการแสดงผล พิมพ์ localhost:8000/ ที่ Browser คลิกที่เมนู **เข้าระบบ** แล้วคลิกปุ่ม Forgot Your Password?



ระบบจะแสดงหน้าจอสำหรับให้ผู้ใช้งานกรอกอีเมล



เมื่อกรอกอีเมลแล้วให้กดปุ่ม Send Password Reset Link ระบบจะมีการส่งอีเมลแจ้งเตือนตามอีเมลที่กรอกเข้าไป จากนั้นให้ไปตรวจสอบอีเมลเพื่อรับลิงค์ยืนยันสำหรับรีเซตรหัสผ่านใหม่



โค้ดตัวอย่าง

https://github.com/leksoft/laravel10-ebook-source/tree/ResetPassword

หากพบปัญหาเรื่องการส่งเมลไม่ได้ ให้ตรวจสอบที่ไฟล์ .env นะครับ ว่าการตั้งค่าเกี่ยวกับอีเมลถูกต้องหรือเปล่า

.env

```
// .env

MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=465
MAIL_USERNAME=example@gmail.com
MAIL_PASSWORD=password
MAIL_ENCRYPTION=ssl
MAIL_FROM_ADDRESS=example@gmail.com
MAIL_FROM_NAME="${APP_NAME}"
```

โหลดเนื้อหาเพิ่มเติมด้วยการคลิกปุ่ม

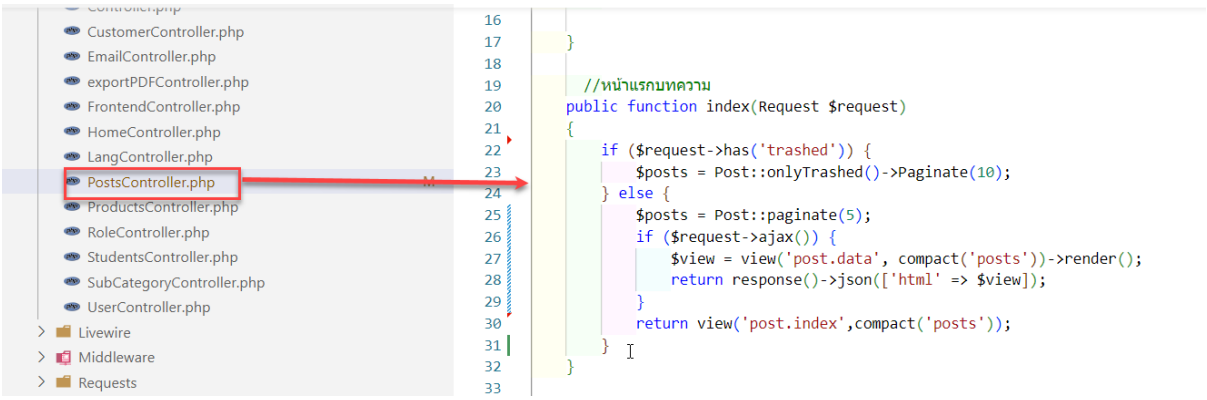
เนื้อหานี้จะเป็นการพูดถึงเทคนิคเล็กๆ ในการทำปุ่มโหลดเนื้อหาหรือข้อมูลเพิ่มเติม ในกรณีที่ว่าเนื้อหามีจำนวนมาก ปกติเราจะแสดงเนื้อหาในรูปแบบการแบ่งหน้า แต่ถ้าเราอยากจะทำปุ่มแล้วให้ผู้ใช้สามารถคลิกแสดงข้อมูลเพิ่มเติมก็สามารถทำได้ โดยจะยกตัวอย่างการใช้ Jquery ในการจัดการปุ่ม ถ้านึกภาพไม่ออกลองดูภาพนี้ตามนะครับ

ลำดับ	หัวข้อ	รายละเอียด	จัดการ
1	Soniaview	Rerum laborum est dolor ut non quo. Sed consequatur voluptate id molestiae et officia. Sequi voluptatem sint modi ut odit similique.	ดู
3	Lake Graysonchester	Aut voluptatem qui voluptas ut. Quia voluptas dolores vitae nihil. Autem aut eum est.	ดู
4	Aidenborough	Atque maiores placeat cum eius nemo quia. Vel ratione est sed qui facere qui ut non. Maiores perferendis quisquam vero vel delectus animi in.	ดู
5	Bradtkeberg	Unde quis distinctio voluptatem. Cumque vero sed atque. Doloremque dolorem et nihil unde.	ดู
6	Port Karinachester	Minus qui dicta dicta beatae nihil nihil eum. Vero impedit non quia tempore expedita. Et ut voluptatem laborum omnis nulla placeat consequatur.	ดู

Load More Data...

โดยเมื่อต้องการโหลดเนื้อหาเพิ่มเติมก็คลิกที่ปุ่ม Load More Data เนื้อหาก็จะแสดงตามครั้งที่กดปุ่มไปเรื่อย ๆ ในตัวอย่างนี้จะใช้ตัวอย่างจากหน้าบทความ ซึ่งเราได้สร้างไว้ก่อนหน้านั้นแล้ว

1.ให้เปิดไฟล์ PostController.php ในโฟลเดอร์ app/Http/Controllers/... แล้วแก้ไขโค้ดในเมธอด index ดังนี้



2.สร้างไฟล์ data.blade.php ไว้ในโฟลเดอร์ resources/views/post/... แล้วเพิ่มโค้ด ดังนี้

```
// resources/views/post/data.blade.php

@foreach ($posts as $post)
    <tr>
        <th scope="col">{{ $post->id }}</th>
        <td scope="col">{{ $post->post_title }}</td>
        <td scope="col" style="width: 50%">{{ $post->post_detail }}</td>
        <td scope="col">
            <a href="{{ route('show', $post->id) }}" class="btn btn-success
btn-sm">ดู</a>
            @auth
                @if (Auth::user()->role === 1)
                    <a href="{{ route('edit', $post->id) }}" class="btn btn-
info btn-sm">แก้ไข</a>|
                    @if (request()->has('trashed'))
                        <a href="{{ route('posts.restore', $post->id) }}"
class="btn btn-success btn-sm">กู้คืน</a>
                    @else
                        <a href="{{ route('destroy', $post->id) }}" class="btn
btn-danger btn-sm" onclick="return confirm('ยืนยันการลบหรือไม่')">ลบ
                        </a>
                    @endif
                @elseif(Auth::user()->role === 2)
                    <a href="{{ route('edit', $post->id) }}" class="btn btn-
info btn-sm">แก้ไข</a>|
                    <a href="{{ route('destroy', $post->id) }}" class="btn btn-
danger btn-sm" onclick="return confirm('ยืนยันการลบหรือไม่')">ลบ
                    </a>
                @endif
            @endauth

        </td>
    </tr>
@endforeach
```

3.เปิดไฟล์ index.blade.php ในโฟลเดอร์ resources/views/post/... แล้วแก้ไขใหม่โค้ด ดังนี้

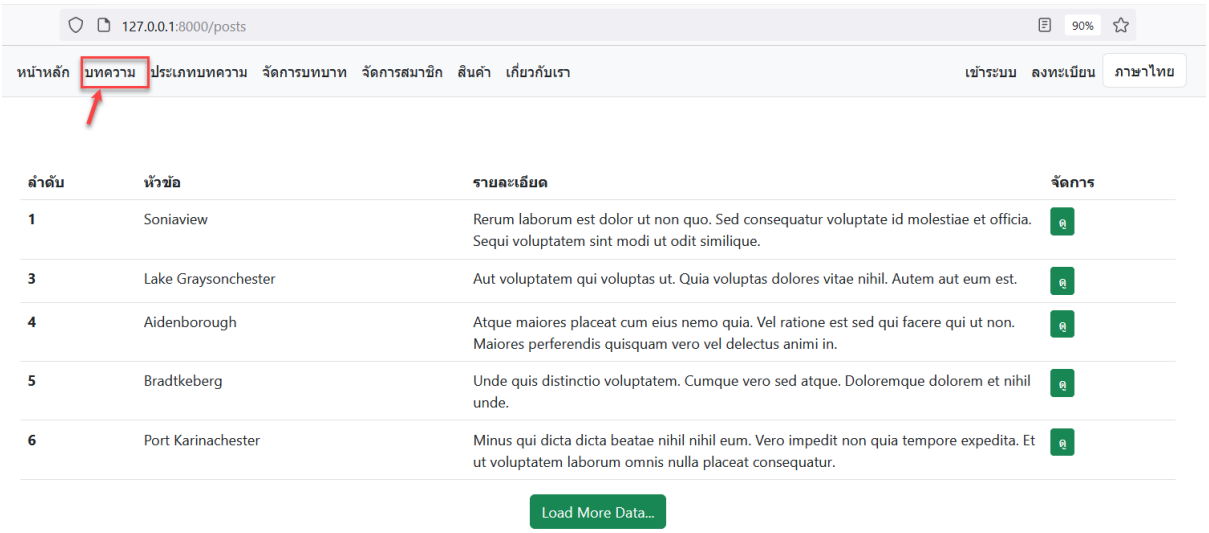
```
// resources/views/post/index.blade.php
@extends('layouts.frontend')

@section('title', 'หน้าแรกบทความ')
@section('content')
    <section class="py-5 container">
        <div class="row">
            <div class="col-lg-12  mx-auto">
                <br />
                <div class="float-end">
                    @auth
                        @if (Auth::user()->role === 1)
                            @if (request()->has('trashed'))
   <a href="{{ route('posts.index') }}" class="btn btn-info">ดูบทความทั้งหมด</a>
   <a href="{{ route('posts.restoreAll') }}" class="btn btn-success">กู้คืนทั้งหมด</a>
                            @else
   <a href="{{ route('create', ['trashed' => 'post']) }}" class="btn btn-danger">
เพิ่มบทความใหม่</a>
   <a href="{{ route('posts.index', ['trashed' => 'post']) }}"
     class="btn btn-primary">บทความที่ถูกลบ</a>
                            @endif
                        @elseif(Auth::user()->role === 2)
                            @if (request()->has('trashed'))
     <a href="{{ route('posts.index') }}" class="btn btn-info">ดูบทความทั้งหมด</a>
                            @else
     <a href="{{ route('create', ['trashed' => 'post']) }}" class="btn btn-danger">
เพิ่มบทความใหม่
     </a>
     <a href="{{ route('posts.index', ['trashed' => 'post']) }}" class="btn btn-
primary">บทความที่ถูกลบ
      </a>
                            @endif
                        @endif
                    @endauth
                </div>
```

มีต่อ...

```
// resources/views/post/index.blade.php

                    <div id="data-wrapper">
                     <table class="table table-hover">
                         <thead>
                             <tr>
                                 <th scope="col">ลำดับ</th>
                                 <th scope="col">หัวข้อ</th>
                                 <th scope="col">รายละเอียด</th>
                                 <th scope="col">จัดการ</th>
                             </tr>
                         </thead>

                         <tbody>
                             @include('post.data')
                         </tbody>
                     </table>
                 </div>
                 <div class="text-center">
<button class="btn btn-success load-more-data">
    <i class="fa fa-refresh"></i> Load More Data...
</button>
                 </div>

                 <!-- แสดง icon Loader -->
                 <div class="auto-load text-center" style="display: none;">
                     <svg version="1.1" id="L9"
xmlns="http://www.w3.org/2000/svg"
                         xmlns:xlink="http://www.w3.org/1999/xlink" x="0px"
y="0px" height="60"
                         viewBox="0 0 100 100" enable-background="new 0 0 0 0"
xml:space="preserve">
                         <path fill="#000"
                             d="M73,50c0-12.7-10.3-23-23-23S27,37.3,27,50
M30.9,50c0-10.5,8.5-19.1,19.1-19.1S69.1,39.5,69.1,50">
                             <animateTransform attributeName="transform"
attributeType="XML" type="rotate" dur="1s"
                                 from="0 50 50" to="360 50 50"
repeatCount="indefinite" />
                         </path>
                     </svg>
                 </div>
             </div>
         </div>
     </section>
@endsection
```

มีต่อ...

```
// resources/views/post/index.blade.php

@section('js_before')
    <script>
        var ENDPOINT = "{{ route('posts.index') }}";
        var page = 1;

        $(".load-more-data").click(function() {
            page++;
            infinteLoadMore(page);
        });

        function infinteLoadMore(page) {
            $.ajax({
                    url: ENDPOINT + "?page=" + page,
                    datatype: "html",
                    type: "get",
                    beforeSend: function() {
                        $('.auto-load').show();
                    }
                })
                .done(function(response) {
                    if (response.html == '') {
                        $('.auto-load').html("ไม่สามารถแสดงเนื้อหาได้ :(");
                        return;
                    }
                    $('.auto-load').hide();
                    $("#data-wrapper").append(response.html);
                })
                .fail(function(jqXHR, ajaxOptions, thrownError) {
                    console.log('Server error occured');
                });
        }
    </script>
@endsection
```



โค้ดตัวอย่าง

https://github.com/leksoft/laravel10-ebook-source/tree/LoadMoreData

Laravel 11 Sanctum REST API Authentication



Sanctum เป็นแพคเกจใหม่ของ Laravel ที่ช่วยให้การตรวจสอบและการรับรองสิทธิ์ในแอปพลิเคชันให้ง่ายขึ้น โดยเฉพาะสำหรับการสร้างแอปพลิเคชันที่เป็นแบบ SPA เช่น ReactJs, VueJs ช่วยให้การเชื่อมต่อกับ API ของ Laravel ส่งข้อมูลปลอดภัยและมีความน่าเชื่อถือ เช่น การล็อกอิน การลงทะเบียนผู้ใช้ หรือการร้องขอข้อมูลอื่น ๆ จากแอปพลิเคชัน เวลาที่เราสร้าง API มาแต่ละส่วนนั้น บางครั้งเราไม่อยากให้ API ของเราเข้าถึงแบบง่าย ๆ หรือไม่มีการปกป้องการเข้าถึงข้อมูลที่เราสร้างจาก API ก็กลายเป็นว่าใคร ๆ ก็สามารถเข้าใช้งานได้ แบบนี้ก็ไม่น่าจะดีเท่าไหร่นัก จึงมีกระบวนการบางอย่างมาช่วยเสริมทำให้ API ของเรามีความปลอดภัยยิ่งขึ้น

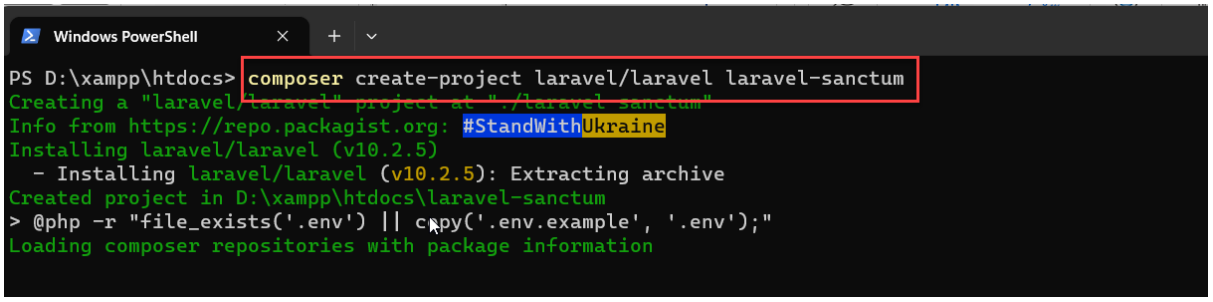
Sanctum มีมาตั้งแต่ Laravel 7 แต่ตอนนั้นยังไม่ได้บรรจุเป็น packages หลักของ Laravel เวลาจะใช้งานจะต้องติดตั้งก่อน จนปัจจุบัน Laravel 11 ก็ได้บรรจุ Sanctum เป็น packages หลัก จะมีมาพร้อมใช้เมื่อติดตั้ง Laravel 11 เราเพียงไปตั้งค่าเพื่อเปิดใช้แล้วก็เขียนโค้ดเข้าไปนิดหน่อยก็สามารถทำงานได้แล้ว Sanctum เป็น Authentication Service ที่สามารถรองรับ Authentication ในรูปแบบของ Session base และ Token base ครบถ้วน จึงถือได้ว่า Laravel Sanctum มีคุณสมบัติรักษาความปลอดภัยและความน่าเชื่อถือในการรับรองสิทธิ์ของ API ได้มาก โดยให้โทเคนทำหน้าที่เป็นตัวบอกว่าผู้ใช้ได้รับสิทธิ์ในการเข้าถึงแอปพลิเคชันของคุณ ทำให้คุณสามารถส่งคำขอ API จากส่วนหน้าโดยไม่ต้องให้ข้อมูลรับรองสิทธิ์ (credentials) ทุกครั้งที่ขอรับข้อมูล ทำให้กระบวนการขอรับข้อมูลมีความสะดวกและรวดเร็วมากยิ่งขึ้น

ในเนื้อหานี้จะแสดงตัวอย่างและวิธีการใช้งาน Laravel Sanctum ในการทำ REST API Authentication อย่างง่าย โดยจะยกตัวอย่างตั้งแต่การติดตั้งจนถึงการตั้งค่าและการใช้งาน มีขั้นตอนดังนี้

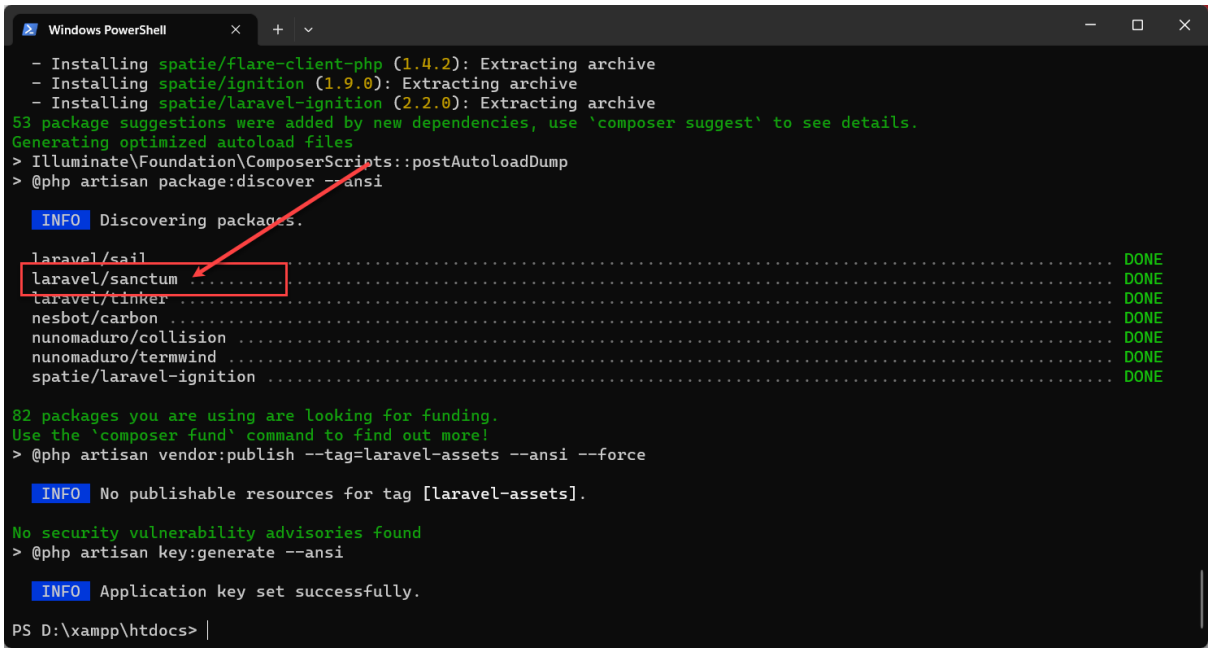
เราจะขึ้นโปรเจคด้วยการ ติดตั้ง Laravel 11 ใหม่เลยนะครับ

1.ให้เปิด Command Prompt หรือ Windows PowerShell แล้วพิมพ์คำสั่ง cd เข้าไปยังตำแหน่งที่เก็บไฟล์ ในที่นี้เก็บไว้ใน D:\xampp\htdocs แล้วรันคำสั่งนี้เพื่อติดตั้ง Laravel 11

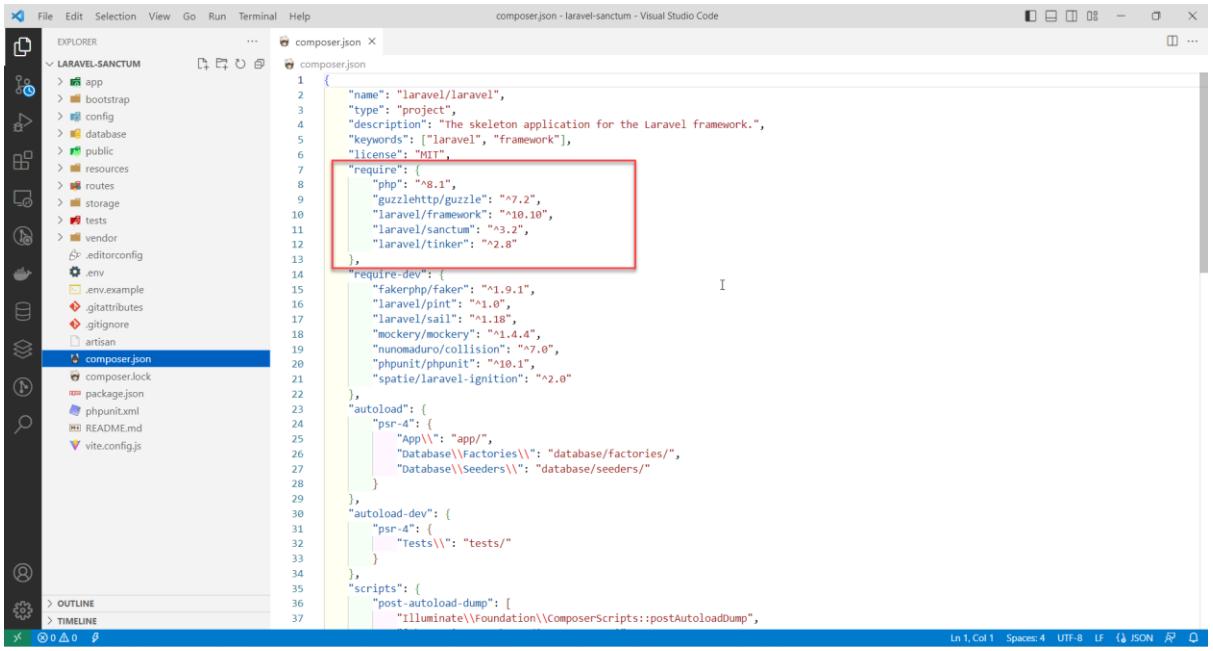
```
composer create-project laravel/laravel laravel-sanctum
```



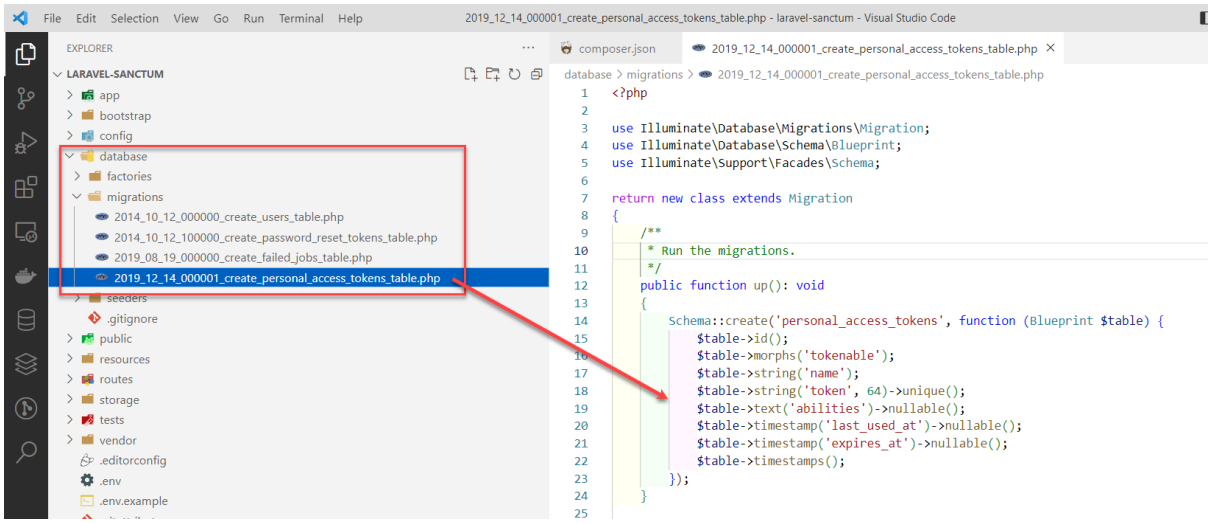
รอจนกว่าจะเสร็จกระบวนการติดตั้งและเมื่อติดตั้งแล้วจะแสดงหน้าแสดงรายการดังนี้



เนื่องจากว่า Laravel 11 ได้บรรจุ Sanctum เป็น packages หลักเราจะเห็นว่ามี packages sanctum เตรียมไว้ให้เราเรียบร้อยแล้ว และเมื่อเข้าไปเปิดดูที่ไฟล์ composer.json ก็จะเห็น laravel/sanctum ถูก require ให้พร้อมใช้



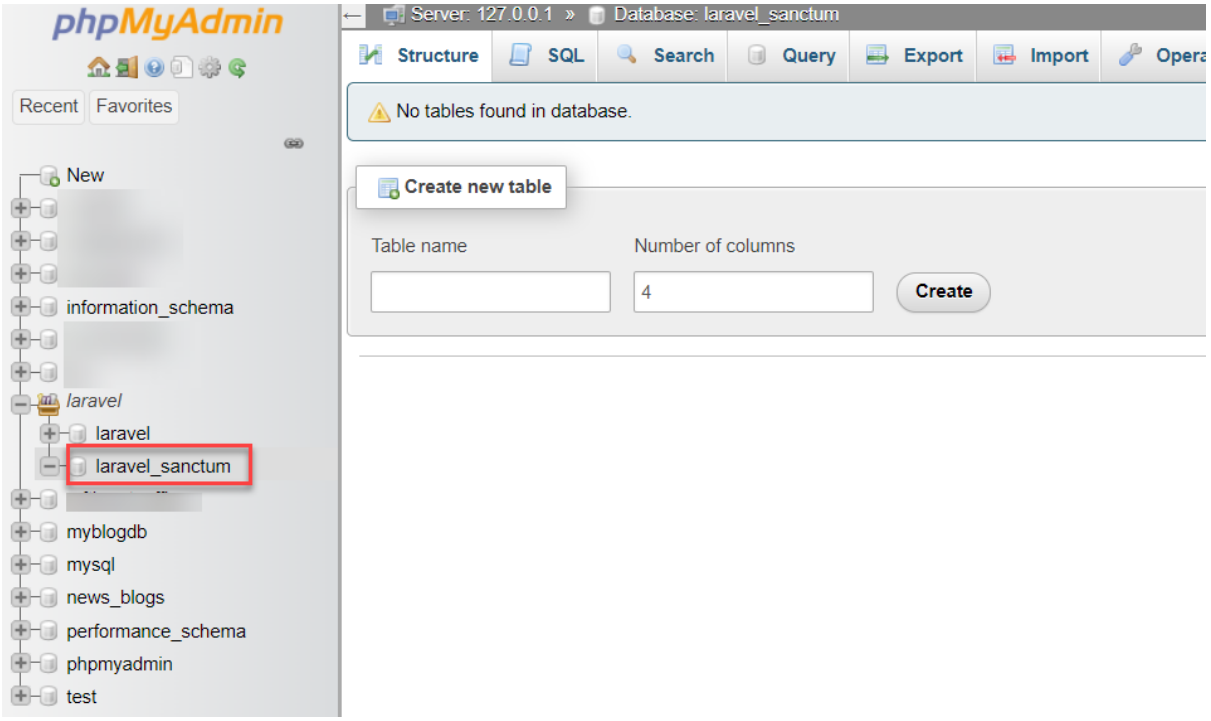
ก่อนที่เราจะใช้ Santum นั้น ให้เราไปตั้งค่าการเชื่อมต่อกับฐานข้อมูลด้วยนะครับ เนื่องจากว่า Sanctum นั้นจะมีการเก็บ token ไว้ในฐานข้อมูล เราจะต้องทำการ migrate เพื่อสร้างตาราง personal_access_tokens โดย Laravel 11 จะเตรียมมาให้แล้วเหมือนกัน ตรวจสอบได้ที่โฟลเดอร์ databases/migrations/...



ตั้งค่า .env เพื่อกำหนดการเชื่อมต่อกับฐานข้อมูล

```
10
11    DB_CONNECTION=mysql
12    DB_HOST=127.0.0.1
13    DB_PORT=3306
14    DB_DATABASE=laravel_sanctum
15    DB_USERNAME=root
16    DB_PASSWORD=
17
```

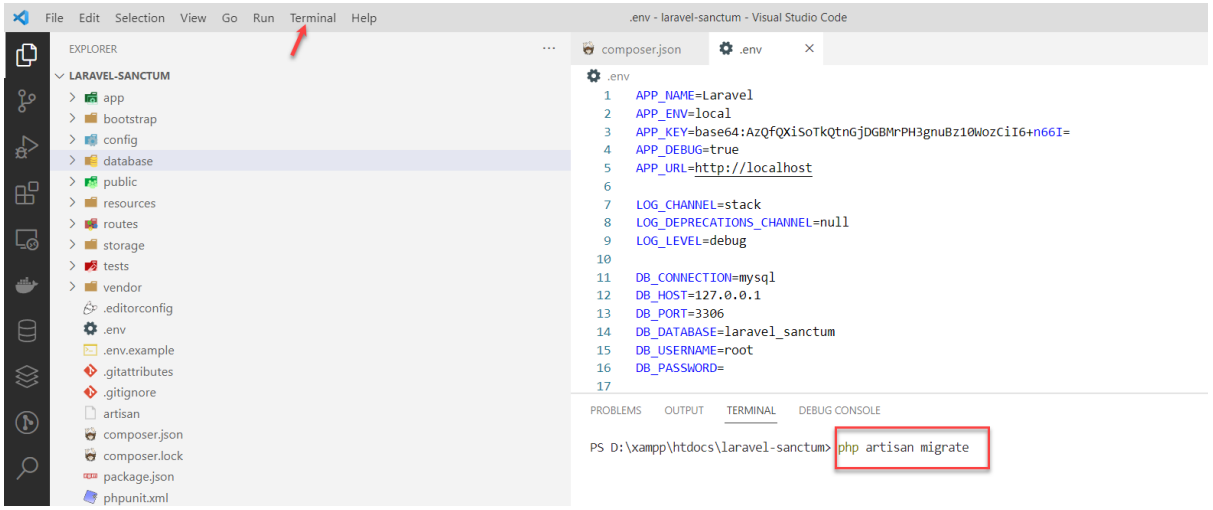
2.สร้างฐานข้อมูลใหม่ชื่อ Laravel_sanctum หรือชื่ออื่นตามที่ต้องการ



เมื่อตั้งค่าทุกอย่างพร้อมแล้วให้รันคำสั่ง migrate

```
php artisan migrate
```

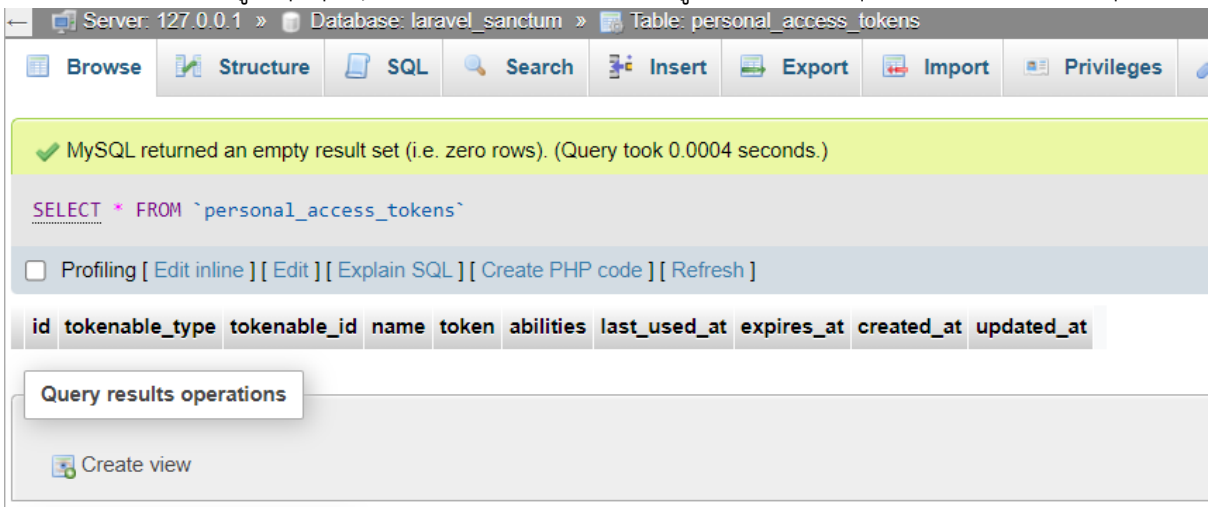
เปิดโปรแกรม Visual Studio Code ขึ้นมาแล้วนำเข้าโปรเจค laravel_sanctum จากนั้นคลิกที่เมนู Terminal เลือก New Terminal เพื่อแสดงหน้า Terminal ผ่าน Visual Studio Code แล้วพิมพ์คำสั่ง migrate ดังนี้ (พิมพ์เสร็จแล้วเคาะ Enter)



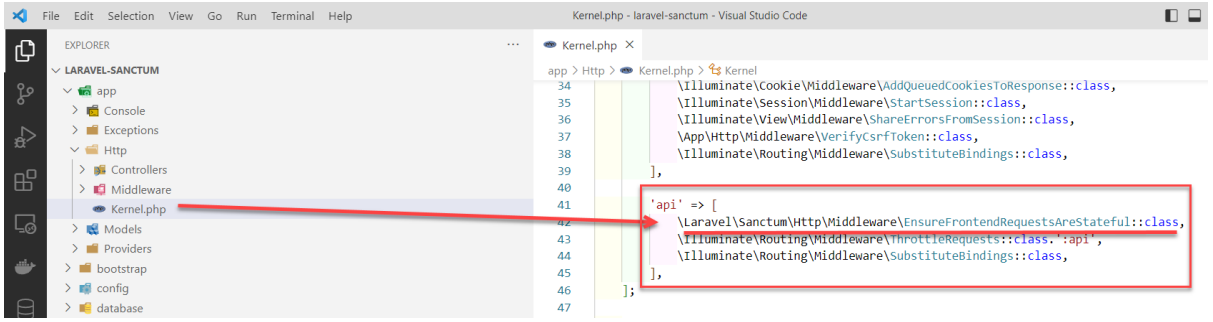
```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                    powershell

PS D:\xampp\htdocs\laravel-sanctum> php artisan migrate

  INFO  Preparing database.

  Creating migration table ..................................................... 66ms DONE

  INFO  Running migrations.

  2014_10_12_000000_create_users_table ......................................... 33ms DONE
  2014_10_12_100000_create_password_reset_tokens_table ......................... 48ms DONE
  2019_08_19_000000_create_failed_jobs_table ................................... 41ms DONE
  2019_12_14_000001_create_personal_access_tokens_table ........................ 43ms DONE

PS D:\xampp\htdocs\laravel-sanctum>
```

เมื่อใช้คำสั่ง php artisan migrate แล้วสิ่งที่เกิดขึ้นเราจะได้ตารางพื้นฐานถูกสร้างขึ้นมา และจะมีตาราง personal_access_tokens ซึ่งเป็นตารางที่สำคัญสำหรับเก็บรหัส Token ที่มีการขอใช้ API

และเมื่อเปิดดูใน phpMyAdmin ก็จะเห็นว่ามีตารางถูกสร้างขึ้นจริง ๆ พร้อมโครงสร้างต่าง ๆ



3.เปิดใช้งาน Middleware ของ Sanctum โดยไปเปิดไฟล์ Kelnel.php ในโฟลเดอร์ app/Http/... ส่วนของ array api ใน middlewareGroups



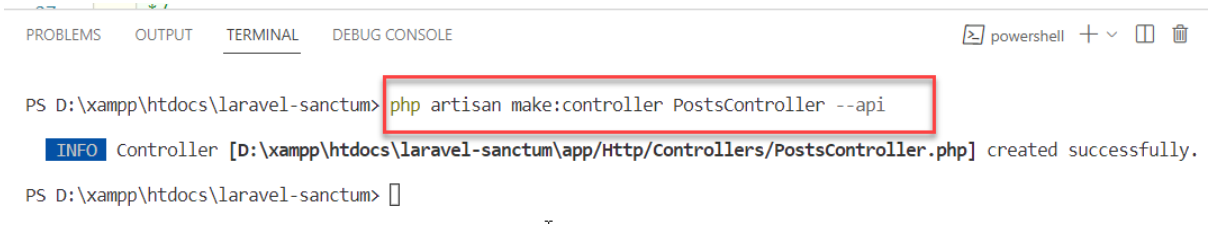

```
//routes/api.php
Route::group(['middleware' => ['auth:sanctum']], function () {

    // protected routes go here

});
```

นั่นหมายความว่าหาก routes ไหนที่จะป้องกันไม่ให้คนอื่นเข้าถึงได้ ก็ให้เขียนไว้ภายใต้ middleware ของ sanctum

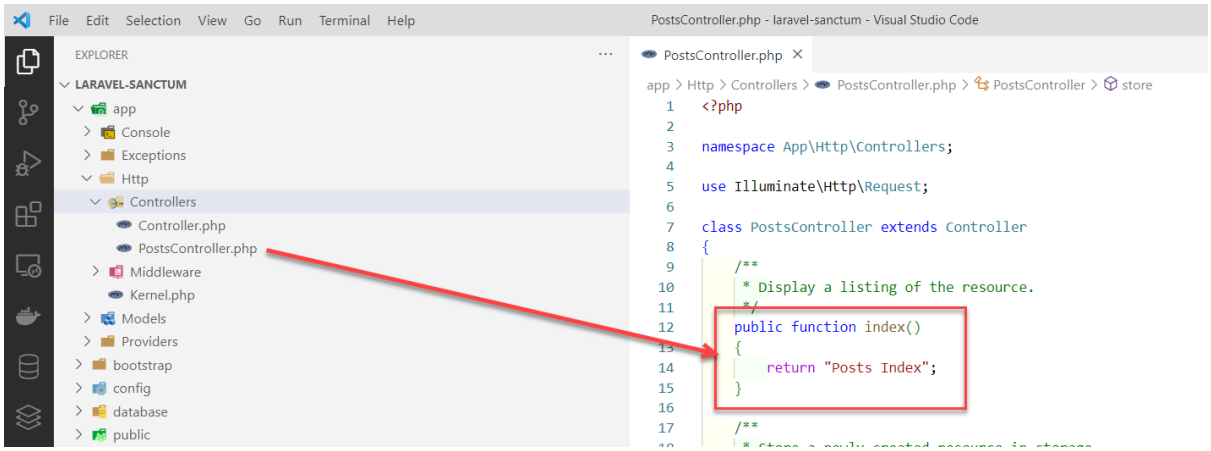
```
Route::group(['middleware' => ['auth:sanctum']], function () {
        Route::resource('posts', PostController::class);
        Route::get('/posts/search/{title}', [PostController::class, 'search']);
        Route::get('/post/author/{id}', [PostController::class, 'get_author']);
});
```

ทีนี้มาลองทดสอบดูครับว่าจะใช้งานได้จริงไหม

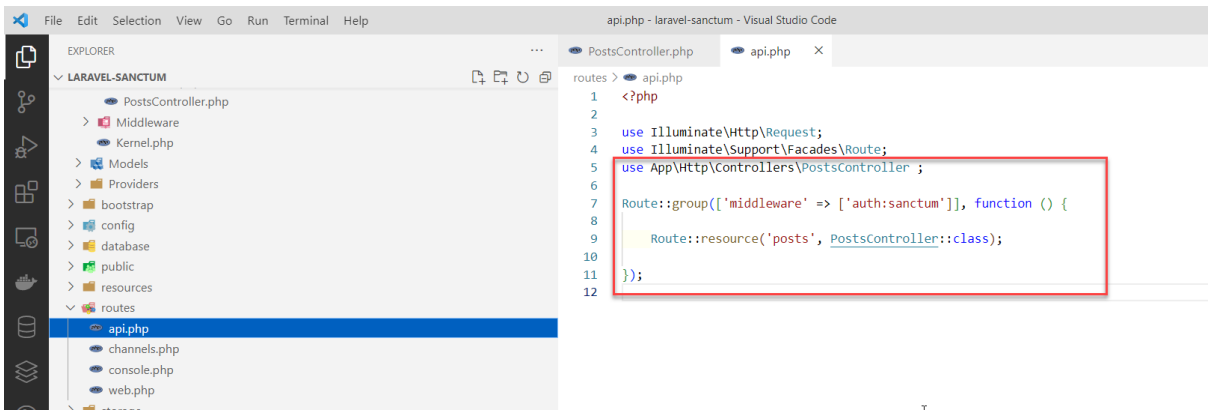
5.ให้สร้าง Controllers ชื่อ postsController พิมพ์คำสั่งดังนี้



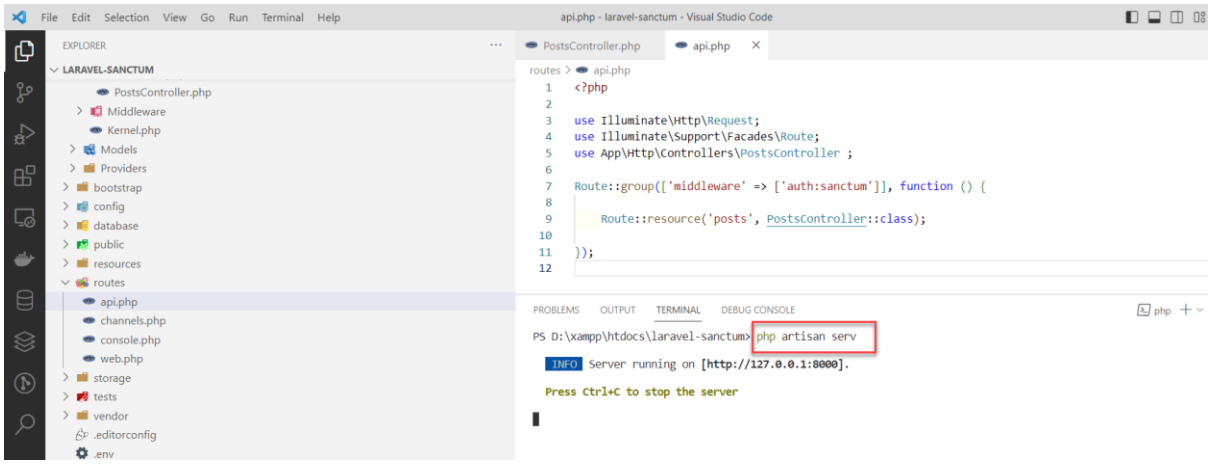
เปิดไฟล์ postsController.php แล้วพิมพ์โค้ดเข้าไปในเมธอด index()



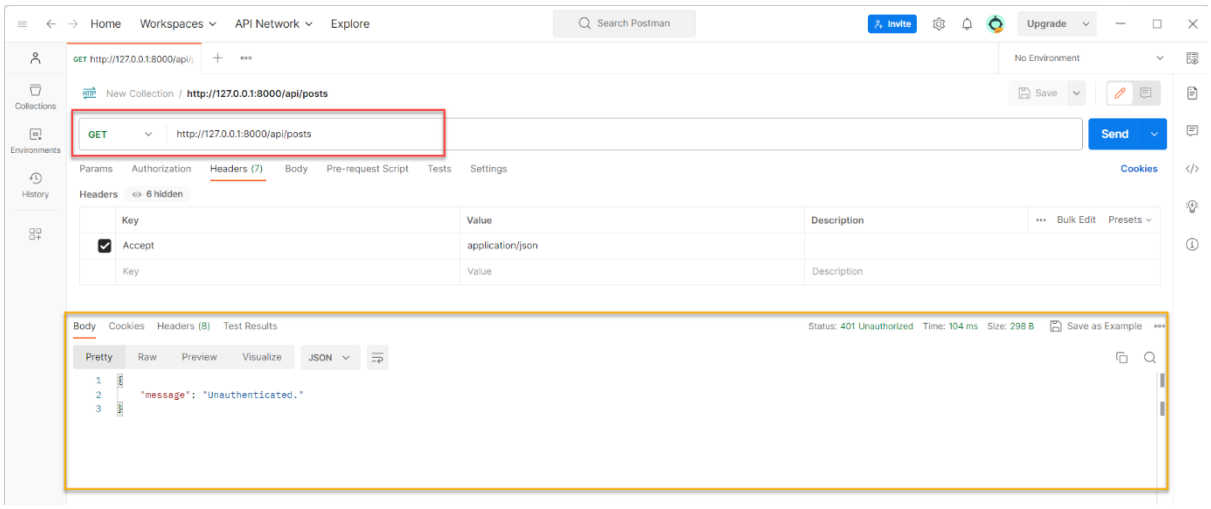
6.เปิดไฟล์ api.php ในโฟลเดอร์ routes/... แล้วสร้าง route ใหม่ดังนี้



บันทึกไฟล์ทั้งหมด แล้ว run project ด้วยคำสั่ง php artisan serv



7.ทดลองเรียกใช้ API ด้วยโปรแกรม Postman หากยังไม่มี Postman ให้ติดตั้งก่อนนะครับ โดยจะเรียกใช้ผ่าน URL localhost:8000/api/posts และระบุเมธอดในการส่งเป็นแบบ GET แล้วดูผลลัพธ์

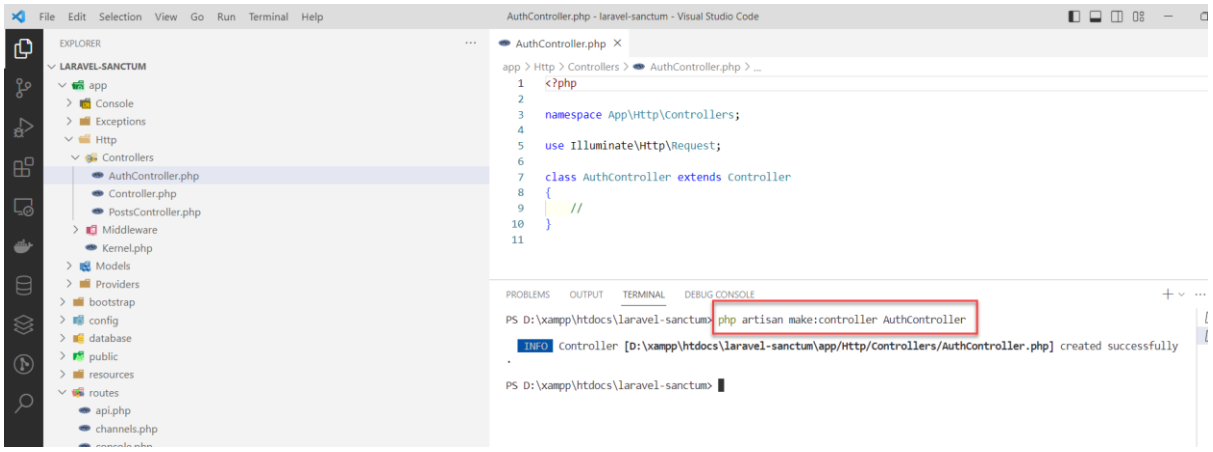


เราคาดหวังว่าเมื่อเราเรียก URL localhost:8000/api/posts แล้ว ในกรองสีส้มจะมีข้อความแจ้งเตือนว่า "Unauthenticated." แสดงว่า API ชื่อ posts ถูกป้องกันไม่ให้เข้าถึงเรียบร้อยแล้ว หากจะเข้าถึง URL นี้ก็จะต้องมี Token แนบไปด้วยถึงจะสามารถเข้าใช้งาน URL ได้ ซึ่ง Token นั้นก็จะได้จากการที่เราจะต้องทำการ Register ก่อน เพื่อขอรับ Token

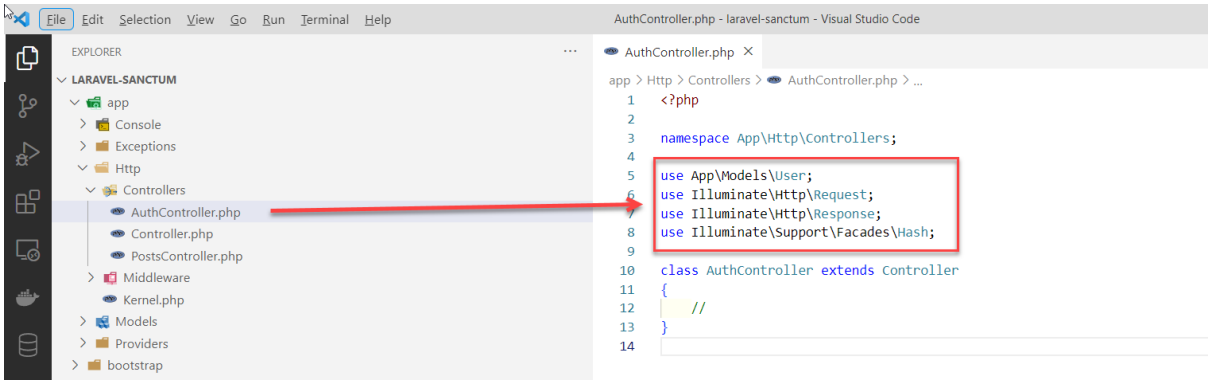
8.ลงทะเบียนขอรับ Token เพื่อขอใช้ API

ลำดับแรกให้สร้าง Controller ใหม่ด้วยคำสั่ง Artisan ตั้งชื่อว่า AuthController

```
php artisan make:controller AuthController
```



จากนั้นให้เพิ่ม namespace ไว้ใต้ namespace App\Http\Controllers ดังนี้



use App\Models\User เป็นโมเดลผู้ใช้งานซึ่งจะถูกสร้างขึ้นให้อัตโนมัติเมื่อติดตั้ง Laravel

use Illuminate\Http\Request เป็น object ที่เก็บข้อมูลที่ผู้ใช้ส่งไปยังเซิร์ฟเวอร์

use Illuminate\Http\Response เป็น object ที่เก็บข้อมูลที่เซิร์ฟเวอร์ส่งกลับมาให้ผู้ใช้

use Illuminate\Support\Facades\Hash เป็น object ที่ใช้ในการเข้ารหัสผ่าน

สร้างเมธอด signUp สำหรับการลงทะเบียน ใน AuthController.php

```
//app/Http/Controller/AuthController.php
public function signUp(Request $request){

}
```

จากนั้นให้ทำการเพิ่มโค้ดสำหรับการตรวจสอบข้อมูลเข้าไปในเมธอด signUp

```
//app/Http/Controller/AuthController.php
 $data = $request->validate([

            'name' => 'required|string',
            'email' => 'required|string|unique:users,email',
            'password' => 'required|string|confirmed'
 ]);
```

โดยมีใช้ฟังก์ชัน Validate ในการตรวจสอบ ดังนี้

name จำเป็นต้องระบุค่า

email จำเป็นต้องระบุค่าและอีเมลต้องไม่ซ้ำกันในตาราง users

password จำเป็นต้องระบุค่าและต้องยืนยันอีกครั้งดังนั้นผู้ใช้ต้องป้อนรหัสผ่านอีกครั้งหนึ่ง

เพิ่มโค้ดต่อจากการ validate เพื่อบันทึกข้อมูลการลงทะเบียนลงฐานข้อมูล บันทึกลงตาราง users

```
//app/Http/Controller/AuthController.php
 $user = User::create([

            'name' => $data['name'],
            'email' => $data['email'],
            'password' => bcrypt($data['password'])
 ]);
```

ใช้ class create สำหรับการบันทึกข้อมูลผู้ใช้ใหม่ลงในตารางพร้อมส่งข้อมูลในรูปแบบอาเรย์ซึ่งเป็นข้อมูลที่รับเข้ามาผ่านฟอร์มลงทะเบียน เมื่อบันทึกข้อมูลแล้ว ให้ทำการสร้าง Token สำหรับการ authentication โดยในการสร้าง Token ใหม่นั้นจะใช้ class `createToken('ระบุชื่ออุปกรณ์')` แล้วแปลง Token ที่ได้ให้เป็น text ธรรมดา โดยใช้ `plainTextToken`

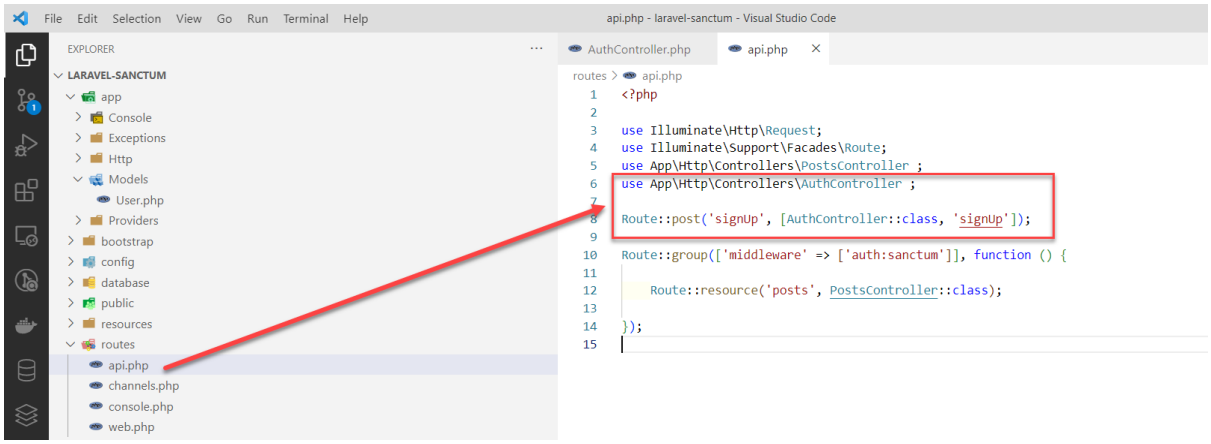
```
//app/Http/Controller/AuthController.php
 $token = $user->createToken('apiToken')->plainTextToken;

        $res = [
            'user' => $user,
            'token' => $token
        ];
        return response($res, 201);
```

จากนั้นก็ทำการ สร้างตัวแปล $res โดยเก็บข้อมูลที่เป็นอาเรย์ซึ่งจะมีข้อมูลผู้ใช้ และ Token ที่ถูกสร้างขึ้น แล้วคืนค่าผลลัพธ์นี้โดยการ return response() พร้อมส่งรหัสสถานะการทำงาน 201 ซึ่งหมายความว่าข้อมูลผู้ใช้ถูกสร้างขึ้นและมีการสร้าง Token เรียบร้อยแล้ว

จะได้โค้ดทั้งหมดในเมธอด signUp ดังนี้

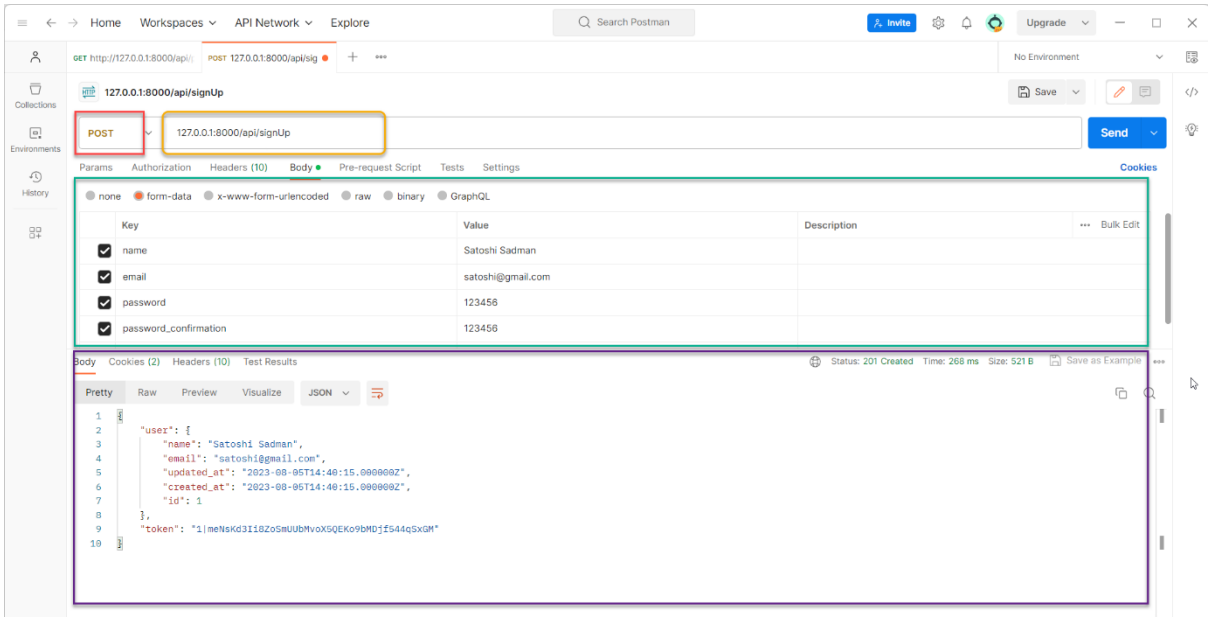
```php
    public function signUp(Request $request){
        $data = $request->validate([
            'name' => 'required|string',
            'email' => 'required|string|unique:users,email',
            'password' => 'required|string|confirmed'
        ]);

        $user = User::create([
            'name' => $data['name'],
            'email' => $data['email'],
            'password' => bcrypt($data['password'])
        ]);

        $token = $user->createToken('apiToken')->plainTextToken;
        $res = [
            'user' => $user,
            'token' => $token
        ];
        return response($res, 201);

    }
```

จากนี้ให้เพิ่ม signUp ใน routes/api.php ดังนี้

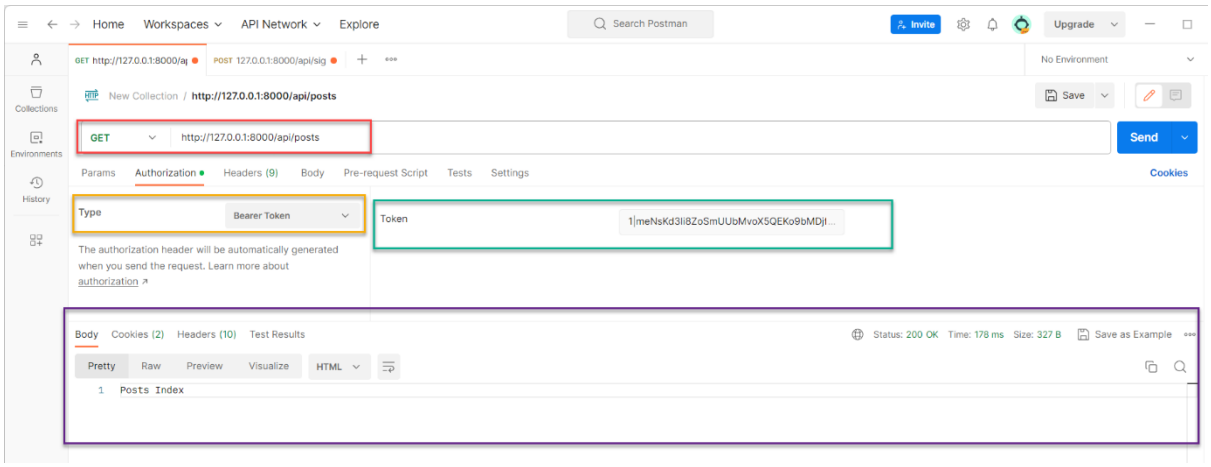


ทดลองเพิ่มผู้ใช้ใหม่โดยเรียกใช้ API ด้วยโปรแกรม Postman

จะเรียกใช้ผ่าน URL localhost:8000/api/signUp และระบุเมธอดในการส่งเป็นแบบ POST



ให้เลือกในแทบของ Body คลิก form-data แล้วระบุ key และ value ตามภาพด้านบน แล้วกด Send เพื่อส่ง Request เมื่อมีการเพิ่มข้อมูลผู้ใช้ใหม่สำเร็จก็จะคืนค่ากลับพร้อมแสดงข้อมูล ผู้ใช้และรหัส Token ที่นี้เราก็ลองเรา Token ที่ได้ไปทดลองใช้งาน ก่อนหน้านั้นเราได้ป้องกันสิทธิ์ในการเข้าถึง URL posts ให้เราลองยิง Request ใหม่ โดยให้แนบ Token ที่ได้ไปด้วย ให้เราคัดลอก Token ที่ได้ไว้ก่อนแล้วเปิด Postman ขึ้นมาใหม่ แล้วกำหนดค่าตามนี้



ให้กำหนดค่าในแทบ Authorization เลือก Type ประเภท Bearer Token จากนั้นก็วางรหัส Token ในช่อง Token แล้วกด Send เพื่อส่ง Request ก็จะเห็นว่าสามารถเข้าถึง URL ที่ถูกป้องกันไว้ได้แล้ว

ฉะนั้นแล้วหากผู้ใช้คนไหนไม่มีรหัส Token ก็จะไม่สามารถเข้าใช้งานในส่วนนั้น ๆ ที่เราใช้ sanctum ป้องกันไว้ได้ ซึ่งรหัส Token นี้จะสามารถใช้งานได้ตลอดจนกว่าผู้ใช้จะทำการลบ Token นั้นออกไป หรือ มีการ Logout ออกจากระบบ Token นั้นก็จะไม่สามารถใช้งานได้ทันที หากจะใช้งานก็ต้องทำการ Login เพื่อรับ Token ใหม่

9.สร้างระบบ Login

ให้สร้างเมธอดสำหรับการทำ Login ใหม่ โดยให้เพิ่มเมธอด login ถัดจากเมธอด signUp ใน AuthController.php

เปิดไฟล์ AuthController.php ในโฟลเดอร์ app/Http/Controllers/... แล้วเพิ่มโค้ดใหม่ ดังนี้

```
//app/Http/Controller/AuthController.php
public function login(Request $request){

}
```

จากนั้นให้ทำการเพิ่มโค้ดสำหรับการตรวจสอบข้อมูลเข้าไปในเมธอด login

```
//app/Http/Controller/AuthController.php
 $data = $request->validate([

            'email' => 'required|string',
            'password' => 'required|string'
        ]);
```

เพิ่มโค้ดสำหรับการตรวจสอบการเข้าระบบ

```
//app/Http/Controller/AuthController.php
$user = User::where('email', $data['email'])->first();

        if (!$user || !Hash::check($data['password'], $user->password)) {
                    return response([
                        'msg' => 'incorrect username or password'
                    ], 401);
        }
```

โค้ดด้านบน จะมีการสร้างตัวแปรขึ้นมา 1 ตัวแปร คือ $user แล้วเก็บคำสั่งในการตรวจสอบอีเมล โดยจะเรียกใช้โมเดล User แล้วใช้ where ในการตรวจสอบข้อมูลอีเมล หลังจากนั้นใช้ if() ตรวจสอบเงื่อนไขตัวแปร $user พร้อมกับตรวจสอบรหัสผ่านที่กรอกเข้ามา โดยใช้ Hash::check() หากไม่พบข้อมูลผู้ใช้งานหรือรหัสผ่านไม่ถูกต้องก็จะแสดงข้อความ 'msg' => 'incorrect username or password' และแสดง Status Code 401 ออกมา

หากตรวจแล้วเงื่อนไขถูกต้อง ให้ทำการสร้าง Token สำหรับการ authentication โดยในการสร้าง Token ใหม่นั้นจะใช้ class createToken('ระบุชื่ออุปกรณ์') แล้วแปลง Token ที่ได้ให้เป็น text ธรรมดา โดยใช้ plainTextToken จากนั้นก็ทำการ สร้างตัวแปร $res โดยเก็บข้อมูลที่เป็นอาเรย์ซึ่งจะมีข้อมูลผู้ใช้ และ Token ที่ถูกสร้างขึ้น แล้วคืนค่าผลลัพธ์นี้โดยการ return response() พร้อมส่งรหัสสถานะการทำงาน 201 ซึ่งหมายความว่ามีการ login เข้าระบบและมีการสร้าง Token เรียบร้อยแล้ว

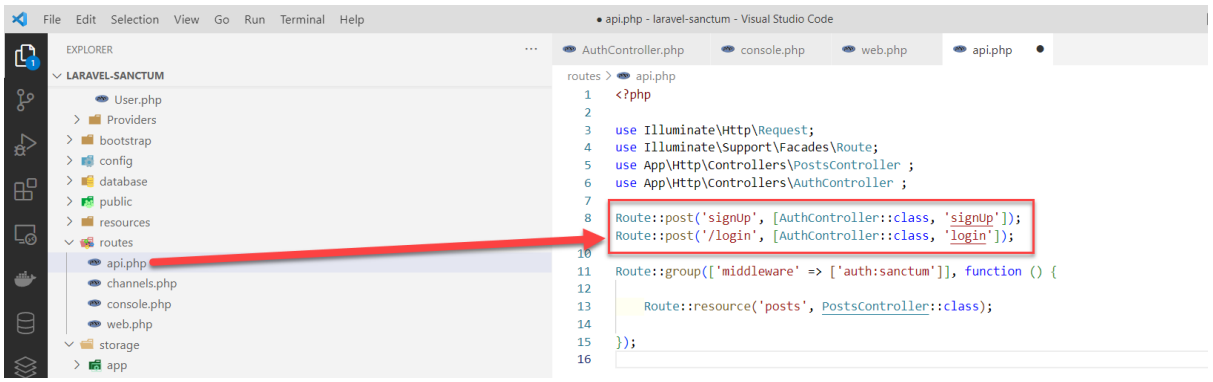
```
//app/Http/Controller/AuthController.php
$token = $user->createToken('apiToken')->plainTextToken;

        $res = [
            'user' => $user,
            'token' => $token
        ];

        return response($res, 201);
```

จะได้โค้ดทั้งหมดในเมธอด login ดังนี้

```
//app/Http/Controller/AuthController.php
    public function login(Request $request){

        $data = $request->validate([
            'email' => 'required|string',
            'password' => 'required|string'
        ]);

        $user = User::where('email', $data['email'])->first();

        if (!$user || !Hash::check($data['password'], $user->password)) {
                    return response([
                        'msg' => 'incorrect username or password'
                    ], 401);
        }

        $token = $user->createToken('apiToken')->plainTextToken;
        $res = [
            'user' => $user,
            'token' => $token
        ];

        return response($res, 201);

    }
```

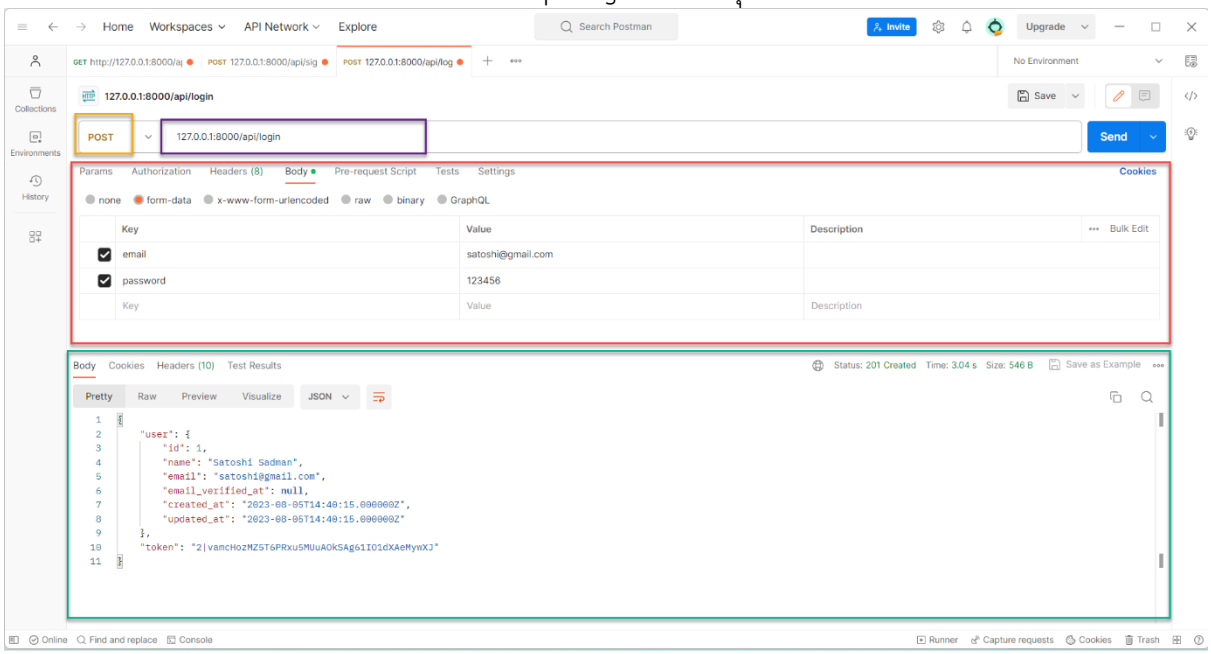
จากนั้นให้เพิ่ม login ใน routes/api.php ดังนี้

```
//routes/api.php
Route::post('/login', [AuthController::class, 'login']);
```



ทดลองเข้าระบบโดยเรียกใช้ API ด้วยโปรแกรม Postman

จะเรียกใช้ผ่าน URL localhost:8000/api/login และระบุเมธอดในการส่งเป็นแบบ POST



ให้เลือกในแทบของ Body คลิก form-data แล้วระบุ key และ value ตามภาพด้านบน แล้วกด Send เพื่อส่ง Request เมื่อมีการเข้าระบบสำเร็จก็จะคืนค่ากลับพร้อมแสดงข้อมูล ผู้ใช้และรหัส Token แสดงให้เห็นตามกรอบสีเขียวด้านบน เพียงเท่านี้เราก็สามารถนำ Token ไปใช้ได้แล้ว

เราสามารถตั้งเวลาของ Token ได้ว่าจะให้ Token หมดอายุเมื่อไหร่ โดยสามารถไปตั่งค่าเพิ่มเติมในส่วนของการตั้งค่าของ sanctum ซึ่งจะอยู่ในไฟล์ app/config/sanctum.php

โดยค่าเริ่มต้นนั้น ระบบจะตั้งค่าเริ่มต้นไว้แบบไม่มีวันหมดอายุคือ `'expiration' => null`

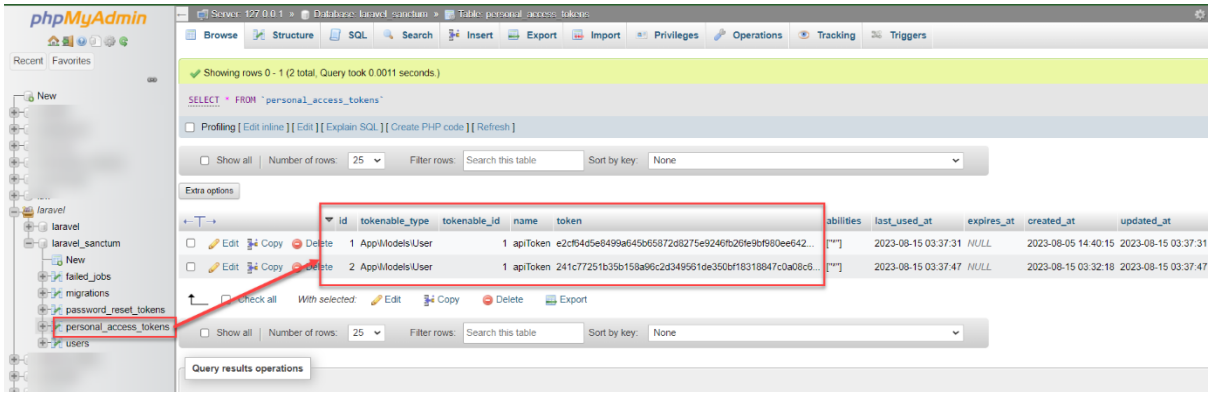
หากต้องการกำหนดให้ Token หมดอายุสามารถตั้งค่าด้วยตัวเองได้ โดยจะกำหนดเป็นตัวเลขเวลาหน่วยเป็น นาที เช่น

```
//app/config/sanctum.php
'expiration' => 60 , // 60 นาที หรือ 1 ชั่วโมง token จะหมดอายุ
```

9. การลบ Token

การลบ Token นั้นสามารถทำได้ 2 แบบคือ ตั้งค่าวันหมดอายุของ Token ที่ไฟล์ sanctum.php และ สร้างระบบให้ผู้ใช้ออกจากระบบด้วยตนเอง จะใช้วิธีไหนก็ได้ครับขึ้นอยู่กับผู้พัฒนาระบบว่าจะให้ Token ลบออกแบบไหน ในที่นี้จะสาธิตการลบ Token ด้วยการออกจากระบบแล้วให้ลบ Token ทันที

ให้ผู้อ่านเปิด โปรแกรม phpMyAdmin ขึ้นมาแล้วตรวจสอบตาราง personal_access_tokens ตอนนี้จะเห็นว่ามีข้อมูลผู้ใช้ Token



เราจะมาดูการเปลี่ยนแปลงหลังจากที่ผู้ใช้ออกจากระบบว่า เมื่อออกจากระบบแล้ว Token จะหายไปจริงหรือไม่ ก่อนอื่น ให้สร้างเมธอดสำหรับการออกจากระบบ โดยให้เพิ่มเมธอด logout ถัดจากเมธอด login ใน AuthController.php

เปิดไฟล์ AuthController.php ในโฟลเดอร์ app/Http/Controllers/... แล้วเพิ่มโค้ดใหม่ ดังนี้

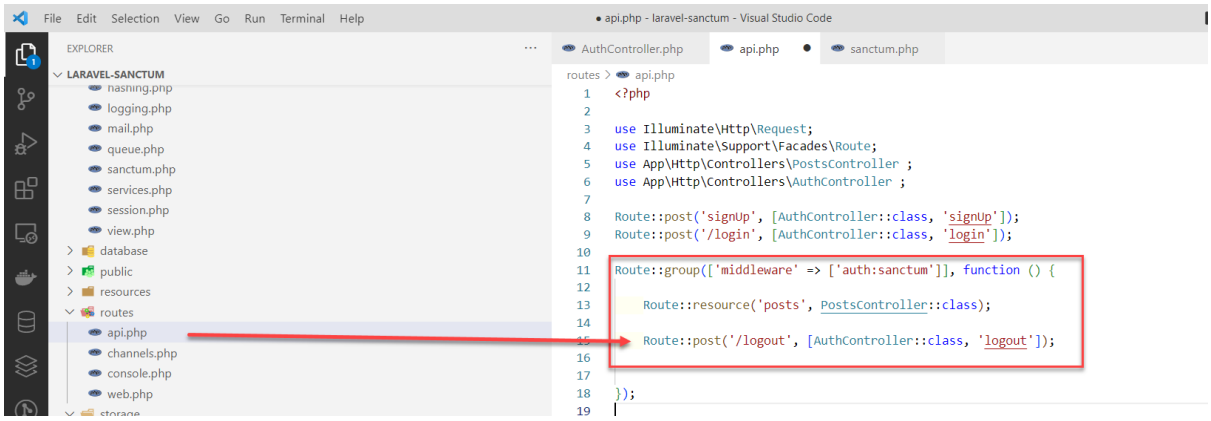
```
//app/Http/Controller/AuthController.php
        auth()->user()->tokens()->delete(); // ออกจากระบบและลบ Token ออก

         return [
             'message' => 'user logged out'
         ];
```

จากนั้นให้ทำการเพิ่มโค้ดสำหรับออกจากระบบและลบ Token ในเมธอด logout

```
//app/Http/Controller/AuthController.php
public function logout(Request $request)

    {
        auth()->user()->tokens()->delete(); // ออกจากระบบและลบ Token ออก
        return [
            'message' => 'user logged out'
        ];

    }
```

จากนั้นให้เพิ่ม logout ใน routes/api.php ดังนี้

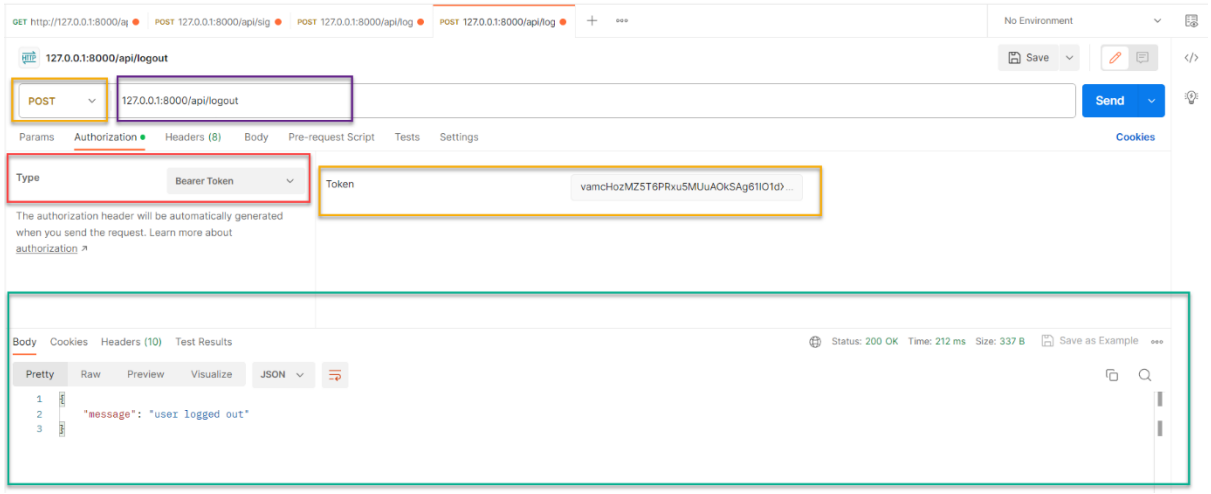
```
//routes/api.php
Route::post('/logout, [AuthController::class, logout]);
```



ให้เพิ่ม route ชื่อ logout ไว้ภายใน middleware ของ sanctum เนื่องจากว่าการที่จะออกจากระบบได้นั้นจะต้องผ่านการ login ก่อน จึงจะสามารถออกจากระบบได้

ทดลองเข้าระบบโดยเรียกใช้ API ด้วยโปรแกรม Postman

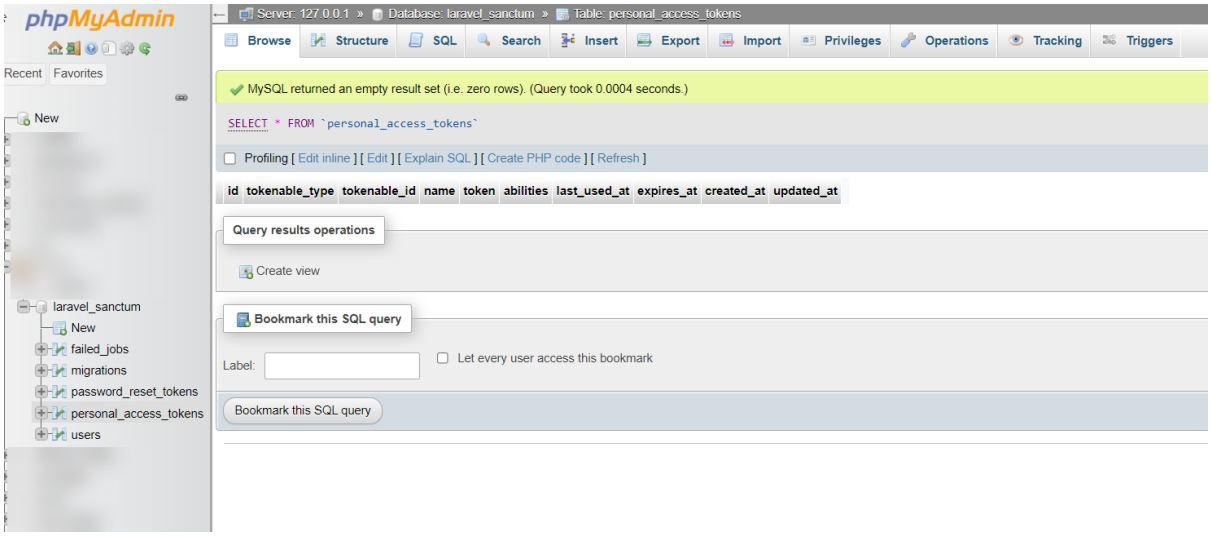
จะเรียกใช้ผ่าน URL localhost:8000/api/logout และระบุเมธอดในการส่งเป็นแบบ POST



ให้กำหนดค่าในแทบ Authorization เลือก Type ประเภท Bearer Token จากนั้นก็วางรหัส Token ที่เข้าระบบไว้ในช่อง Token แล้วกด Send เพื่อส่ง Request

หากออกจากระบบสำเร็จจะขึ้นข้อความ `'message' => 'user logged out'` เพียงเท่านี้ Token ก็จะถูกลบออกเรียบร้อยแล้ว

และเมื่อเปิด โปรแกรม phpMyAdmin ขึ้นมาแล้วตรวจสอบตาราง personal_access_tokens ตอนนี้จะเห็นว่าข้อมูลผู้ใช้ Token ถูกลบออกเรียบร้อยแล้ว



โค้ดตัวอย่าง

https://github.com/leksoft/laravel-sanctum