WEB

MVC (Model-View-Controller) คือรูปแบบในการออกแบบและการจัดระบบของแอปพลิเคชันที่ช่วยแยกส่วนประกอบของแอปพลิเคชันออกเป็นส่วนๆ เพื่อให้การพัฒนาและบำรุงรักษาระบบเป็นไปอย่างมีระเบียบ มีความยืดหยุ่น และสามารถเปลี่ยนแปลงแต่ละส่วนโดยอิสระโดยไม่กระทบกับส่วนอื่น ๆ ของระบบมากนัก

ส่วนประกอบหลักของ MVC ประกอบด้วย:

1. Model (โมเดล): เป็นส่วนที่เกี่ยวข้องกับข้อมูลและฐานข้อมูล มักจะเป็นส่วนที่จัดการกับการเข้าถึงและจัดการข้อมูล ซึ่งอาจเป็นการอ่านและเขียนข้อมูลไปยังฐานข้อมูล หรือจัดการกับโครงสร้างข้อมูลที่จำเป็นสำหรับแสดงผลใน View
2. View (วิว): เป็นส่วนที่เกี่ยวข้องกับการแสดงผลข้อมูล ในส่วนนี้มักจะเป็นโค้ดที่ใช้แสดงผลในหน้าเว็บ ซึ่งมักจะเป็นโค้ดที่ใช้เพื่อแสดงข้อมูลที่ได้รับมาจาก Model และแสดงผลให้กับผู้ใช้
3. Controller (คอนโทรลเลอร์): เป็นส่วนที่เกี่ยวข้องกับการควบคุมการทำงานของแอปพลิเคชัน คอนโทรลเลอร์จะรับข้อมูลจากผู้ใช้และส่งต่อไปยังโมเดลเพื่อดำเนินการกับข้อมูล และส่งผลลัพธ์ที่ได้กลับมายังวิวเพื่อแสดงผลให้กับผู้ใช้

ขั้นตอนการทำงานของ MVC มีดังนี้:

1. ผู้ใช้ทำการกระทำ (เช่น คลิกปุ่มหรือกรอกข้อมูลในฟอร์ม) บนส่วนของ View
2. View แจ้งเตือน Controller ว่าผู้ใช้ได้ทำการกระทำ
3. Controller รับข้อมูลจาก View และดำเนินการตามขั้นตอนที่เหมาะสม เช่น การอ่านข้อมูลจากฐานข้อมูลผ่าน Model
4. Model ดำเนินการตามที่ Controller สั่งให้ทำ เช่น การอ่านหรือเขียนข้อมูลในฐานข้อมูล
5. Model ส่งผลลัพธ์กลับไปยัง Controller
6. Controller จัดรูปแบบข้อมูลและส่งไปยัง View เพื่อแสดงผลให้กับผู้ใช้
7. View แสดงผลข้อมูลให้กับผู้ใช้

การใช้งาน MVC ช่วยให้การพัฒนาแอปพลิเคชันมีความรู้สึกเป็นระเบียบและมีความยืดหยุ่นในการเปลี่ยนแปลงแก้ไขระบบ แต่ยังคงมีความเป็นระเบียบในการแยกส่วนประกอบของระบบอยู่เพื่อความง่ายในการจัดการและการบำรุงรักษาในอนาคต

```
Route::get('/contact',function(){

$price=500000;

return view('contact')->with('price',$price);

});

<!DOCTYPE html>

<html>

<body>

<h1>Route and View</h1>

<h1>Price = {{number_format($price,2)}}</h1>

</body>

</html>
```

ค้ดดังกล่าวเป็นตัวอย่างของการกำหนดเส้นทาง (Route) และการสร้าง View ใน Laravel:

1. Route: ในไฟล์ routes/web.php กำหนดเส้นทาง '/contact' ด้วยเมธอด 'get' ที่จะใช้สำหรับร้องขอหน้า '/contact' โดยจะทำการทำงานในฟังก์ชันแบบไม่มีชื่อ (anonymous function) ซึ่งอยู่ในลักษณะ Closure ซึ่งเป็นฟังก์ชันที่ไม่ได้รับการกำหนดชื่อและไม่ต้องการการนิยามในที่อื่น โดยในฟังก์ชันนี้จะทำการกำหนดตัวแปร $price เป็น 500,000 และคืนค่า View ชื่อ 'contact' พร้อมกับส่งตัวแปร $price ไปให้กับ View ด้วยเมธอด 'with'.
2. View: ในไฟล์ resources/views/contact.blade.php สร้างหน้า View ที่จะแสดงผลเมื่อเรียกหน้า '/contact' ซึ่งมีโค้ดในลักษณะ HTML ซึ่งแสดงข้อความ "Route and View" และแสดงราคาที่รับมาจากตัวแปร $price ที่ถูกส่งมาจาก Controller โดยใช้ตัวแปรคือ "{{number_format($price,2)}}" เพื่อแสดงราคาในรูปแบบทศนิยมสองตำแหน่ง (2 decimal places).

สรุป: การกำหนดเส้นทางใน Laravel โดยใช้ Route::get('/contact', function()) ในที่นี้จะเป็นการกำหนดเส้นทางสำหรับหน้า '/contact' และทำการส่งค่า $price ไปยัง View ชื่อ 'contact' ซึ่งใน View นั้นจะแสดงราคาที่ถูกส่งมาในรูปแบบทศนิยมสองตำแหน่ง

web.php

```
Route::get('/string',function(){
$description="Information Technology Information Technology Information
Technology Information Technology";
return view('string')->with('description',$description);
});
```

sting.blade.php

```
<!DOCTYPE html>
<html>
<body>
<h1>{{Str::limit($description,3)}}</h1>
</body>
</html>
```

ในโค้ดที่คุณแสดงมาคือตัวอย่างของการใช้งานฟังก์ชัน Str::limit() ใน Laravel สำหรับจำกัดความยาวของข้อความ (String) ในหน้า View ที่ชื่อ string.blade.php

1. web.php: ในไฟล์ routes/web.php กำหนดเส้นทาง /string ด้วยเมธอด get ที่จะใช้สำหรับร้องขอหน้า /string และทำการส่งตัวแปร $description ไปยัง View ชื่อ string ผ่านฟังก์ชัน with() ซึ่งเป็นวิธีการส่งข้อมูลไปยัง View ใน Laravel
2. string.blade.php: ในไฟล์ resources/views/string.blade.php สร้างหน้า View ที่จะแสดงผลเมื่อเรียกหน้า /string ในส่วนนี้มีโค้ด HTML ซึ่งใน <h1> ทำการใช้ {{ Str::limit($description, 3) }} เพื่อแสดงข้อความ $description โดยจำกัดให้แสดงเพียง 3 ตัวอักษรเท่านั้น และการใช้ {{ ... }} คือการแสดงผลข้อมูลที่ส่งมาจาก Controller หรือในที่นี้คือข้อความที่ส่งมาจาก Route

ตัวอย่างการใช้งาน:

หากเราเข้าถึง URL /string บนเว็บเบราว์เซอร์ จะทำการเรียกใช้ Route ที่กำหนดใน web.php ซึ่งจะส่งตัวแปร $description ให้กับ View string.blade.php และใน View นั้นก็จะทำการแสดงผลข้อความ $description โดยจำกัดให้แสดงเพียง 3 ตัวอักษรเท่านั้น ดังนั้นผลลัพธ์ที่ได้คือ:ซึ่งจะแสดงข้อความ "Inf" ที่ถูกจำกัดความยาวให้เหลือ

เพียง 3 ตัวอักษรเท่านั้น โดยตัวอักษรที่เหลือหลังจากนั้นถูกตัดทิ้ง อ้างอิงจากตัวอย่างข้อมูลที่กำหนดในตัวแปร $description.

Routing ใน Laravel (และหลายๆ ภาษาคอมพิวเตอร์) นั้นค่อนข้างแตกต่างกัน ซึ่งเกี่ยวข้องกับเว็บเฟรมเวิร์กเท่านั้น ใน Laravel Routing คือกระบวนการกำหนดเส้นทางสำหรับการเข้าถึง URL หรือเรียกใช้งานตัวแอปพลิเคชัน กล่าวคือเมื่อผู้ใช้งานเข้ามาที่ URL หรือเส้นทางที่เรากำหนดไว้ใน Laravel ระบบจะนำมาเรียกใช้งานคอนโทรลเลอร์ (Controller) หรือฟังก์ชันที่เรากำหนดไว้ในเส้นทางนั้นๆ เพื่อทำการประมวลผลและแสดงผลให้กับผู้ใช้งาน ดังนั้นการกำหนด Routing เป็นสิ่งสำคัญในการสร้างแอปพลิเคชันเว็บใน Laravel เพื่อให้ผู้ใช้งานสามารถเข้าถึงหน้าเว็บและฟังก์ชันต่างๆ ได้ด้วย URL ที่เรากำหนดไว้ในเส้นทาง (Route) ใน Laravel ที่กำหนดในไฟล์ routes/web.php

```
Route::get('/about', function(){

 return view('about');

});

Route::view('about','about');

Route::view('contact','contact');
```

โค้ดที่คุณแสดงมาคือตัวอย่างของการกำหนด Routing ใน Laravel โดยใช้ฟังก์ชัน Route::get() และ Route::view()

1. Route::get('/about', function(){ return view('about'); });: กำหนดเส้นทางสำหรับ URL '/about' ด้วยเมธอด 'get' ซึ่งเมื่อมีคนเข้าถึง URL นี้ Laravel จะเรียกใช้งานฟังก์ชันแบบไม่มีชื่อ (anonymous function) และทำการส่งข้อมูล View ชื่อ 'about' กลับไปให้กับผู้ใช้ ซึ่งในที่นี้คือหน้า View ที่มีชื่อว่า 'about.blade.php'
2. Route::view('about','about');: กำหนดเส้นทางสำหรับ URL '/about' ด้วยเมธอด 'view' ซึ่งทำงานเช่นเดียวกับเมธอด 'get' แต่เป็นกรณีที่ไม่ต้องใช้ฟังก์ชันแบบไม่มีชื่อ (anonymous function) และส่งข้อมูล View ชื่อ 'about' กลับไปให้กับผู้ใช้ ซึ่งในที่นี้คือหน้า View ที่มีชื่อว่า 'about.blade.php'
3. Route::view('contact','contact');: กำหนดเส้นทางสำหรับ URL '/contact' ด้วยเมธอด 'view' ทำงานเช่นเดียวกับเมธอด 'get' และ 'view' ข้างต้น แต่ในที่นี้คือหน้า View ที่มีชื่อว่า 'contact.blade.php'

สรุป: โค้ดข้างต้นเป็นตัวอย่างของการกำหนด Routing ใน Laravel โดยใช้ฟังก์ชัน Route::get() และ Route::view() เพื่อกำหนดเส้นทางสำหรับ URL และให้ Laravel แสดงผลหน้า View ที่เกี่ยวข้องกับ URL นั้น ๆ ในที่นี้คือ View ที่ชื่อว่า 'about.blade.php' และ 'contact.blade.php' โดยผู้ใช้งานสามารถเข้าถึงหน้าเว็บดังกล่าวโดยพิมพ์ URL '/about' หรือ '/contact' ในเว็บเบราว์เซอร์

```
Route::get('/{name}', function ($name) {
 return view('welcome',['name'=>$name]);
});
Route::view('abc','contact');
```

ในโค้ดดังกล่าวเป็นการส่งข้อมูลผ่านการกำหนดเส้นทาง (Routing) ใน Laravel โดยใช้ Route::get() และ Route::view():

1. Route::get('/{name}', function ($name) { return view('welcome',['name'=>$name]); });: กำหนดเส้นทางสำหรับ URL ที่มีพารามิเตอร์ชื่อ name ด้วยเมธอด 'get' ซึ่งจะทำการรับค่าพารามิเตอร์ name ที่ถูกส่งมาใน URL และนำค่านี้มาใช้ในการส่งต่อไปยัง View ที่ชื่อ 'welcome' ซึ่งส่งข้อมูลตัวแปร name ไปด้วย
2. Route::view('abc','contact');: กำหนดเส้นทางสำหรับ URL '/abc' ด้วยเมธอด 'view' ซึ่งไม่ต้องรับค่าพารามิเตอร์ และจะทำการส่งข้อมูลไปยัง View ที่ชื่อ 'contact' โดยไม่ต้องมีการรับค่าพารามิเตอร์

เมื่อมีผู้ใช้เข้าถึง URL ที่กำหนดในเส้นทางต่างๆ ใน Laravel ทั้งสองเส้นทางนี้ จะเกิดการส่งข้อมูลไปยัง View และทำการแสดงผลข้อมูลที่ถูกส่งมาในหน้าเว็บ

สรุป: โค้ดข้างต้นเป็นตัวอย่างของการส่งข้อมูลผ่านการกำหนดเส้นทาง (Routing) ใน Laravel โดยใช้ Route::get() และ Route::view() ซึ่งให้สามารถรับค่าพารามิเตอร์จาก URL และส่งข้อมูลไปยัง View เพื่อแสดงผลหน้าเว็บดังกล่าว

```
<h1>Welcome page</h1>
<a href="/about">about us</a><br>
<a href="/contact">contact us</a>
```

โค้ดที่คุณแสดงมาเป็นการสร้าง Anchor Tag (ลิงก์) เพื่อนำทางไปยังเส้นทางที่กำหนดใน Laravel ซึ่งในที่นี้คือ '/about' และ '/contact'

about us: ลิงก์นี้จะสร้าง Anchor Tag ที่เมื่อคลิกที่ลิงก์นี้จะนำทางไปยังเส้นทาง '/about' ซึ่งตามโค้ด Route::get('/about', function(){ ... }); ในไฟล์ web.php จะทำการแสดงหน้า View ที่ชื่อ 'about' และเมื่อเรียก URL '/about' ผู้ใช้งานจะเห็นหน้าเว็บที่เกี่ยวข้องกับเนื้อหาในหน้า 'about'

contact us: ลิงก์นี้จะสร้าง Anchor Tag ที่เมื่อคลิกที่ลิงก์นี้จะนำทางไปยังเส้นทาง '/contact' ซึ่งตามโค้ด Route::view('contact', 'contact'); ในไฟล์ web.php จะทำการแสดงหน้า View ที่ชื่อ 'contact' และเมื่อเรียก URL '/contact' ผู้ใช้งานจะเห็นหน้าเว็บที่เกี่ยวข้องกับเนื้อหาในหน้า 'contact'

การใช้งาน Anchor Tag เป็นวิธีที่นำทางไปยังหน้าเว็บต่างๆ หรือส่วนของเนื้อหาที่เกี่ยวข้องกันในเว็บเพื่อให้ผู้ใช้งานสามารถเลือกเข้าถึงหน้าต่างๆ ได้ง่ายและสะดวก โดยที่สิ่งนี้เป็นที่นิยมในการสร้างเว็บไซต์เพื่อมุ่งเน้นความเข้าใจและความสะดวกสบายในการใช้งานสำหรับผู้ใช้งานของเว็บไซต์

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

 return redirect('contact');

});
```

โค้ดที่คุณแสดงมาเป็นตัวอย่างของการกำหนด Routing ใน Laravel ที่ใช้เพื่อทำการ Redirect หรือนำทางไปยัง URL อื่น ๆ

โค้ดนี้กำหนดเส้นทางสำหรับ URL หลัก (root URL) คือ '/' ด้วยเมธอด 'get' ซึ่งเป็นตัวแสดงหน้าแรกของเว็บไซต์ แต่ในที่นี้จะไม่แสดงหน้าแรกของเว็บไซต์เพราะมีการใช้งานฟังก์ชัน redirect() ซึ่งเป็นฟังก์ชันใน Laravel ที่ใช้ในการนำทางหรือทำการ Redirect ไปยัง URL อื่น

เมื่อมีคนเข้าถึง root URL '/' ก็จะทำการเรียกใช้งานฟังก์ชันแบบไม่มีชื่อ (anonymous function) ซึ่งจะทำงานโดยคืนค่าการ Redirect ไปยัง URL '/contact' โดยการใช้ redirect('contact') นั่นคือให้นำทางผู้ใช้งานไปยัง URL '/contact' แทนที่จะแสดงหน้าแรกของเว็บไซต์ ดังนั้นผู้ใช้งานจะถูกนำไปยังหน้าติดต่อ (Contact) โดยอัตโนมัติ

สรุป: โค้ดข้างต้นเป็นตัวอย่างของการกำหนด Routing ใน Laravel ที่ใช้เพื่อทำการ Redirect หรือนำทางผู้ใช้งานไปยัง URL อื่น ๆ โดยการใช้ redirect() ในการกำหนดเส้นทางใหม่และนำทางผู้ใช้งานไปยังหน้าต่าง ๆ ตามที่กำหนดในฟังก์ชัน redirect() นั้น ๆ

```
php artisan make:Controller Users

public function index()

 {

echo "Hello from controller";

 }
```

การใช้คำสั่ง php artisan make:controller Users จะสร้าง Controller ใหม่ที่มีชื่อว่า UsersController ในโปรเจค Laravel ของคุณ ซึ่งจะถูกสร้างขึ้นในโฟลเดอร์ app/Http/Controllers ตามด้วยชื่อไฟล์ UsersController.php

หลังจากที่ Controller UsersController ถูกสร้างขึ้นแล้ว คุณสามารถเขียนโค้ดที่อยู่ภายใน Controller เพื่อกำหนดหน้าที่และการทำงานของ Controller ตามที่คุณต้องการให้มันทำ

ตัวอย่างการสร้าง Controller UsersController ที่มีฟังก์ชัน index() ที่ส่งข้อความ "Hello from controller" ไปแสดงผล:

เมื่อคุณเรียกใช้งาน Controller UsersController ผ่าน URL ตามเส้นทางที่คุณกำหนด เช่น /users จะเรียกใช้งานฟังก์ชัน index() ใน Controller นี้ ซึ่งจะแสดงข้อความ "Hello from controller" บนหน้าเว็บของคุณ

php artisan make:controller LoginController

```
public function index()
 {
 return view('login');
 }
// public function loginSubmit()
 public function loginSubmit(Request $request)
 {
 // return "Form Submitted";
returen $request->all();
 }
```

โค้ดที่คุณให้มาเป็นตัวอย่างการสร้าง Controller ที่ชื่อว่า LoginController ใน Laravel และมีฟังก์ชันสองฟังก์ชันคือ index() และ loginSubmit() ที่อยู่ภายใน Controller นี้

php artisan make:controller LoginController: เป็นคำสั่งที่ใช้สร้าง Controller ใหม่ที่มีชื่อว่า LoginController ในโปรเจค Laravel ของคุณ โดย Controller นี้จะถูกสร้างขึ้นในโฟลเดอร์ app/Http/Controllers และในตัวอย่างนี้คือ LoginController.php

public function index(): เป็นฟังก์ชันที่ถูกกำหนดไว้ใน LoginController ซึ่งจะคืนค่า View ที่ชื่อว่า 'login' ซึ่งอยู่ในโฟลเดอร์ resources/views กลับไปแสดงผลบนหน้าเว็บ เมื่อคุณเรียกใช้งาน Controller ผ่าน URL ที่ตรงกับเส้นทางที่คุณกำหนด เช่น '/login' แล้ว Controller จะแสดง View 'login' ให้กับผู้ใช้งาน

public function loginSubmit(Request $request): เป็นฟังก์ชันที่ถูกกำหนดไว้ใน LoginController ซึ่งรับค่า Request ที่ส่งมาจากผู้ใช้งานเมื่อทำการส่งฟอร์มในหน้า 'login' และในตัวอย่างนี้คือ $request ซึ่งจะใช้ในการดึงข้อมูลที่ผู้ใช้กรอกในฟอร์ม และฟังก์ชันนี้คืนค่าข้อมูลทั้งหมดที่ถูกส่งมาใน Request กลับไปแสดงผลในหน้าเว็บเพื่อให้คุณตรวจสอบค่าที่ถูกส่งมาจากฟอร์มหรือประมวลผลต่อไป

สรุป: โค้ดข้างต้นเป็นตัวอย่างของการสร้าง Controller LoginController และกำหนดฟังก์ชัน index() และ loginSubmit() ที่ใช้ในการคืนค่า View และการรับข้อมูล Request ใน Controller นี้ ซึ่งสามารถนำไปใช้งานเพื่อจัดการหน้าเว็บที่เกี่ยวข้องกับการเข้าสู่ระบบ (Login) ได้

```
public function loginSubmit(Request $request)

 {

$email = $request->input('email');

 $password = $request->input('password');

 return 'Email :'.$email. 'Password :'.$password;

 }
```

ในโค้ดข้างต้นคือฟังก์ชัน loginSubmit() ที่ถูกกำหนดใน Controller LoginController ใน Laravel ซึ่งมีหน้าที่ใช้ในการรับข้อมูลจาก Request ที่ส่งมาจากผู้ใช้งานเมื่อทำการส่งฟอร์มในหน้า 'login' และนำข้อมูลที่รับมาใน Request ไปใช้งาน

1. $request: เป็นตัวแปรที่รับค่า Request ที่ส่งมาจากผู้ใช้งาน ซึ่งในกรณีนี้คือ Request ที่ส่งมาจากการส่งฟอร์มในหน้า 'login'
2. $request->input('email'): เป็นการใช้ฟังก์ชัน input() บนตัวแปร $request เพื่อดึงค่าที่มีชื่อว่า 'email' ออกมา ซึ่งในที่นี้เป็นการดึงค่าอีเมล (email) ที่ผู้ใช้กรอกในฟอร์ม
3. $request->input('password'): เหมือนกับข้อ 2 แต่คือการดึงค่ารหัสผ่าน (password) ที่ผู้ใช้กรอกในฟอร์ม
4. return 'Email :'.$email. 'Password :'.$password;: เมื่อได้รับข้อมูลอีเมลและรหัสผ่านจากฟอร์มแล้วก็จะนำข้อมูลที่รับมามาเชื่อมต่อกันและคืนค่าออกมาในรูปแบบของข้อความ ซึ่งข้อความที่คืนค่าก็คือ "Email : ค่าอีเมลที่กรอกในฟอร์ม Password : ค่ารหัสผ่านที่กรอกในฟอร์ม"

สรุป: ฟังก์ชัน loginSubmit() ใน Controller LoginController เป็นตัวอย่างการรับข้อมูลอีเมลและรหัสผ่านที่ผู้ใช้กรอกในฟอร์มและนำข้อมูลนี้ไปใช้งานโดยคืนค่าออกมาในรูปแบบข้อความเพื่อให้แสดงผลในหน้าเว็บหรือประมวลผลต่อไปตามที่คุณต้องการในแอปพลิเคชันของคุณ

```
<form method="POST" action="{{route('login.submit')}}">
 @csrf
 <div class="mb-3">
 <label for="Email1" class="form-label">Email address</label>
 <input type="email" name="email" class="form-control" aria-describedby="emailHelp">
 <div id="emailHelp" class="form-text">We'll never share your email with anyone else.</div>
 </div>
 <div class="mb-3">
 <label for="Password" class="form-label">Password</label>
 <input type="password" class="form-control"name="password">
 </div>
 <div class="mb-3 form-check">
 <input type="checkbox" class="form-check-input" id="Check">
 <label class="form-check-label" for="Check">Check meout</label>
 </div>
 <button type="submit" class="btn btn-primary">Submit</button>
 </form>
```

โค้ดที่คุณให้มาเป็นตัวอย่างของฟอร์ม (Form) ที่ใช้ในการเข้าสู่ระบบ (Login Form) ที่มีการส่งข้อมูลไปยังเส้นทางที่ชื่อว่า 'login.submit' ซึ่งคือเส้นทางที่กำหนดให้กับการจัดการการเข้าสู่ระบบ (Login) ใน Controller

`<form method="POST" action="{{route('login.submit')}}">`: ในส่วนนี้เป็นการกำหนดฟอร์มที่มีเส้นทางการส่งข้อมูลไปยัง URL ที่กำหนดใน route('login.submit') ซึ่งจะเป็นเส้นทางที่เรากำหนดให้ในไฟล์ web.php เพื่อทำการจัดการเมื่อมีการส่งฟอร์มนี้ขึ้นมา

@csrf: ในส่วนนี้เป็นการสร้างคีย์ CSRF token ของ Laravel เพื่อป้องกันการโจมตีแบบ Cross-Site Request Forgery (CSRF) ซึ่งเป็นการโจมตีที่นำข้อมูลจากผู้โจมตีมาใช้ใน Request ของผู้ใช้งาน คำสั่ง @csrf นี้จะสร้างคีย์ CSRF token และเพิ่มเข้าไปในฟอร์มที่คุณสร้าง เพื่อให้ความปลอดภัยในการส่งข้อมูล

<input type="email" name="email" class="form-control" aria-describedby="emailHelp">: ฟิลด์ Input ของอีเมล (email) ในฟอร์ม ซึ่งมีชื่อว่า 'email' และเป็นชนิด Input แบบ email ซึ่งจะให้ผู้ใช้กรอกอีเมลเท่านั้น

<input type="password" class="form-control" name="password">: ฟิลด์ Input ของรหัสผ่าน (password) ในฟอร์ม ซึ่งมีชื่อว่า 'password' และเป็นชนิด Input แบบ password ซึ่งจะเป็นตัวอักษรที่เรากำหนดให้แสดงเป็นจุดเมื่อผู้ใช้กรอกรหัสผ่าน

<input type="checkbox" class="form-check-input" id="Check">: ฟิลด์ Checkbox ที่ใช้ในการรับค่าอนุญาตเช็คที่เปิด/ปิด เพื่ออนุญาตให้เก็บข้อมูลเพิ่มเติมหรือเมื่อต้องการบอกว่าต้องการให้ทำอะไรบางอย่าง

<button type="submit" class="btn btn-primary">Submit</button>: ปุ่ม Submit ที่ใช้สำหรับส่งข้อมูลที่อยู่ในฟอร์มไปยังเส้นทางที่กำหนดใน action ของฟอร์ม เมื่อผู้ใช้กดปุ่มนี้ก็จะส่งข้อมูลที่กรอกในฟอร์มไปยังเส้นทางที่กำหนด

```
public function loginSubmit(Request $request)
 {
 $validateData = $request->validate([
 'email'=>'required|email',
 'password'=>'required|min:6|max:12'
 ]);
 $email = $request->input('email');
 $password = $request->input('password');
 return 'Email :'.$email. 'Password :'.$password;
 }
```

ในโค้ดข้างต้นคือฟังก์ชัน loginSubmit() ใน Controller LoginController ซึ่งมีหน้าที่ใช้ในการรับข้อมูลจาก Request ที่ส่งมาจากผู้ใช้งานเมื่อทำการส่งฟอร์มในหน้า 'login' และตรวจสอบความถูกต้องของข้อมูลที่ผู้ใช้กรอกในฟอร์มด้วยคำสั่ง validate() ของ Laravel

1. $request: เป็นตัวแปรที่รับค่า Request ที่ส่งมาจากผู้ใช้งาน ซึ่งในกรณีนี้คือ Request ที่ส่งมาจากการส่งฟอร์มในหน้า 'login'
2. $request->validate([...]): เป็นคำสั่งที่ใช้ในการตรวจสอบความถูกต้องของข้อมูลที่ผู้ใช้กรอกในฟอร์ม ในตัวอย่างนี้คือตรวจสอบว่าอีเมล (email) ที่ผู้ใช้กรอกเป็นค่าที่มีรูปแบบของอีเมลที่ถูกต้องและรหัสผ่าน (password) ที่ผู้ใช้กรอกมีความยาวอยู่ระหว่าง 6 ถึง 12 ตัวอักษร ถ้าข้อมูลที่ผู้ใช้กรอกไม่ถูกต้องตามเงื่อนไขที่กำหนด ก็จะมีการคืนค่ากลับมาแสดงข้อผิดพลาดเพื่อให้ผู้ใช้กรอกข้อมูลใหม่
3. $email = $request->input('email'): เหมือนกับข้อ 2 แต่คือการดึงค่าอีเมล (email) ที่ผู้ใช้กรอกในฟอร์มเพื่อนำไปใช้งานในฟังก์ชัน
4. $password = $request->input('password'): เหมือนกับข้อ 3 แต่คือการดึงค่ารหัสผ่าน (password) ที่ผู้ใช้กรอกในฟอร์มเพื่อนำไปใช้งานในฟังก์ชัน
5. return 'Email :'.$email. 'Password :'.$password;: เมื่อได้รับข้อมูลอีเมลและรหัสผ่านจากฟอร์มแล้ว ก็จะนำข้อมูลที่รับมามาเชื่อมต่อกันและคืนค่าออกมาในรูปแบบข้อความ ซึ่งข้อความที่คืนค่าก็คือ "Email : ค่าอีเมลที่กรอกในฟอร์ม Password : ค่ารหัสผ่านที่กรอกในฟอร์ม"

สรุป: ฟังก์ชัน loginSubmit() ใน Controller LoginController เป็นตัวอย่างการรับข้อมูลอีเมลและรหัสผ่านที่ผู้ใช้กรอกในฟอร์ม และตรวจสอบความถูกต้องของข้อมูลด้วยคำสั่ง validate() ก่อนจะนำข้อมูลนี้ไปใช้งานแสดงผลหรือประมวลผลต่อไปตามที่คุณต้องการในแอปพลิเคชันของคุณ

<input type="email" name="email" class="form-control"

aria-describedby="emailHelp">

@error('email')

{{$message}}

@enderror

โค้ดที่คุณให้มาเป็นตัวอย่างของการสร้างฟิลด์ Input สำหรับอีเมล (email) ในฟอร์ม (Form) ใน Laravel ซึ่งมีการใช้งานคำสั่ง @error('email') เพื่อแสดงข้อความข้อผิดพลาดที่เกิดขึ้นจากการตรวจสอบความถูกต้องของข้อมูลที่ผู้ใช้กรอกในฟิลด์นี้

<input type="email" name="email" class="form-control" aria-describedby="emailHelp">: ฟิลด์ Input ของอีเมล (email) ในฟอร์ม ซึ่งมีชื่อว่า 'email' และเป็นชนิด Input แบบ email ซึ่งจะให้ผู้ใช้กรอกอีเมลเท่านั้น

@error('email'): เป็นคำสั่งของ Blade Template ใน Laravel ที่ใช้ในการตรวจสอบว่ามีข้อผิดพลาดในฟิลด์ที่มีชื่อว่า 'email' หรือไม่ ถ้ามีข้อผิดพลาดในฟิลด์นี้ เช่น ไม่กรอกอีเมลหรือกรอกไม่ถูกต้องตามรูปแบบ ก็จะทำการแสดงข้อความข้อผิดพลาดที่เกิดขึ้น

{{$message}}: เมื่อเกิดข้อผิดพลาดในฟิลด์ 'email' ก็จะนำข้อความข้อผิดพลาดที่ได้มาจากการตรวจสอบมาแสดงผลในส่วนนี้ โดยใช้ตัวแปร $message ซึ่งเป็นตัวแปรที่ Laravel ให้มาใช้ในการเก็บข้อความข้อผิดพลาดในการตรวจสอบ ซึ่งข้อความที่แสดงจะเป็นข้อความที่เรากำหนดในกฎการตรวจสอบความถูกต้องของฟิลด์ใน Controller ของเรา

middle ware คืออะไร

Middleware ใน Laravel เป็นตัวกลางที่ทำหน้าที่ตรวจสอบและประมวลผล Request ก่อนที่ Request จะถึงหรือไปยัง Controller หรือสิ่งอื่นในระบบ เช่นการตรวจสอบการเข้าถึงหน้าต่างๆ การตรวจสอบความถูกต้องของข้อมูลที่ส่งมา การตรวจสอบสิทธิ์การเข้าถึง หรือการทำงานอื่นๆ ที่ต้องการให้เกิดขึ้นก่อนการประมวลผลต่อ

Middleware มักถูกกำหนดในไฟล์ app/Http/Kernel.php ของโปรเจค Laravel โดยใช้ชื่อที่กำหนดเองหรือใช้ Middleware ที่มีอยู่ใน Laravel มาก่อนแล้วกำหนดลำดับการทำงาน (priority) เพื่อให้ Middleware ทำงานตามลำดับที่กำหนด

เมื่อมี Request ถึง Middleware ตัวนึง จะตรวจสอบข้อมูลใน Request และสามารถเปลี่ยนแปลงข้อมูล Request หรือทำการ Redirect ไปหน้าอื่น หากมีเงื่อนไขที่ต้องการเปลี่ยนแปลงหรือตรวจสอบ

เมื่อ Middleware ทำงานเสร็จสิ้น หาก Middleware ไม่ได้ทำการ Redirect หรือสิ้นสุดการประมวลผล Request ใน Middleware นั้นๆ ก็จะส่ง Request ไปยัง Middleware ถัดไปหรือไปยัง Controller หรือสิ่งอื่นที่เป็นต้นทางของ Request ต่อไป

Middleware มีประโยชน์อย่างมากในการจัดการเรื่องต่างๆ เช่น ควบคุมการเข้าถึงหน้าต่างๆ ตรวจสอบความถูกต้องของข้อมูล ทำ Logging การตรวจสอบสิทธิ์ และหน้าที่อื่นๆ ในการควบคุมและประมวลผล Request ตามที่ต้องการในแต่ละแอปพลิเคชัน Laravel

```
public function handle(Request $request, Closure $next)
 {
 if($request->age && $request->age<18){
 return redirect('/noaccess');
 }
```

```
 return $next($request);

 }
```

ฟังก์ชันที่คุณให้มาเป็นตัวอย่างคือ Middleware ใน Laravel ซึ่งมีหน้าที่ในการตรวจสอบอายุของผู้ใช้งานที่ส่งมาใน Request ว่ามีอายุน้อยกว่า 18 ปีหรือไม่ ถ้ามีอายุน้อยกว่า 18 ปีก็จะทำการ redirect ผู้ใช้งานไปยังเส้นทางที่มีชื่อว่า '/noaccess' ซึ่งคือเส้นทางที่เรากำหนดให้ในไฟล์ web.php เพื่อให้ผู้ใช้ไม่สามารถเข้าถึงหน้านั้นได้ และถ้าอายุไม่น้อยกว่า 18 ปีก็จะทำการส่ง Request ต่อไปยัง Middleware หรือ Controller ถัดไปในระบบดั้งเดิม (ถ้ามี)

$request: เป็นตัวแปรที่เก็บข้อมูล Request ที่ส่งมาจากผู้ใช้งาน ในกรณีนี้คือ Request ที่ส่งมาเมื่อผู้ใช้งานเข้าถึง URL ใดๆ ในระบบ

$request->age: เป็นการเข้าถึงข้อมูลอายุที่ผู้ใช้งานส่งมาใน Request ซึ่งเป็นตัวแปรที่มีชื่อว่า 'age' ใน Request นี้

if($request->age && $request->age<18): ในส่วนนี้เป็นการตรวจสอบว่าผู้ใช้งานส่งข้อมูลอายุมาหรือไม่ และถ้ามีการส่งข้อมูลอายุมาใน Request นี้ ก็จะตรวจสอบว่าอายุน้อยกว่า 18 ปีหรือไม่

return redirect('/noaccess');: เมื่อเกิดเงื่อนไขที่อายุน้อยกว่า 18 ปี ก็จะมีการ redirect ผู้ใช้งานไปยัง URL '/noaccess' ซึ่งในตัวอย่างนี้คือเส้นทางที่เรากำหนดให้ในไฟล์ web.php เพื่อให้ผู้ใช้งานไม่สามารถเข้าถึงหน้านั้นได้

return $next($request);: เมื่ออายุของผู้ใช้งานไม่น้อยกว่า 18 ปี ก็จะทำการส่ง Request ต่อไปยัง Middleware หรือ Controller ถัดไปในระบบดั้งเดิม (ถ้ามี) ซึ่งในกรณีนี้ Middleware นี้อาจจะเป็น Middleware ที่ตรวจสอบการเข้าถึงหน้าต่างๆ หรือ Controller ที่จะทำการประมวลผลต่อไปของระบบของเรา

XAMPP คือชุดซอฟต์แวร์ที่มีไว้สำหรับการพัฒนาและทดสอบเว็บแอปพลิเคชัน (Web Application) บนเซิร์ฟเวอร์ (Server) ในระบบปฏิบัติการ Windows, macOS, Linux, และ Solaris ซึ่งส่วนใหญ่ใช้งานบนเซิร์ฟเวอร์ของ Windows และ macOS อย่างแพร่หลายในวงกว้าง ชุดซอฟต์แวร์นี้ถูกพัฒนาขึ้นโดย Apache Friends (ที่มักจะเรียกตัวเองว่า "XAMPP Apache Friends") ซึ่งเป็นชุดซอฟต์แวร์ที่ไม่เสียค่าใช้จ่าย และมีความยืดหยุ่นในการติดตั้งและใช้งาน นักพัฒนาและผู้ที่ต้องการทดสอบและพัฒนาเว็บแอปพลิเคชันสามารถใช้ XAMPP เพื่อสร้างเซิร์ฟเวอร์ส่วนตัว (localhost) บนเครื่องของตนเองและทดสอบแอปพลิเคชันโดยไม่ต้องใช้เซิร์ฟเวอร์จริงๆ ที่เปิดให้บริการในอินเทอร์เน็ต ชุดซอฟต์แวร์ XAMPP ประกอบด้วยส่วนประกอบหลัก ๆ คือ:

1. Apache: เป็นเซิร์ฟเวอร์ HTTP ที่ใช้ในการรับและตอบคำขอเว็บ (HTTP Requests) จากผู้ใช้งาน และทำหน้าที่ส่งข้อมูลตอบกลับไปยังเบราว์เซอร์ของผู้ใช้งาน (HTTP Responses) ซึ่งเป็นส่วนหนึ่งของโปรแกรม XAMPP ที่ใช้ในการเปิดใช้งานเซิร์ฟเวอร์เพื่อทำงานบนเครื่องของคุณ
2. MySQL: เป็นระบบจัดการฐานข้อมูล (Database Management System) ที่ใช้ในการจัดเก็บข้อมูลที่เกี่ยวข้องกับเว็บแอปพลิเคชัน เช่น ข้อมูลผู้ใช้งาน, สินค้า, รายการสั่งซื้อ เป็นต้น ที่สามารถเข้าถึงและดูข้อมูลในฐานข้อมูลได้ผ่านเครื่องมือในรูปแบบของ MySQL Database Management System (DBMS)
3. PHP: เป็นภาษาโปรแกรมที่ใช้ในการเขียนสคริปต์เพื่อดำเนินการต่างๆ ในเว็บแอปพลิเคชัน เช่น การดึงข้อมูลจากฐานข้อมูล, การดำเนินการทางคณิตศาสตร์, การตรวจสอบความถูกต้องของข้อมูล, การสร้างเว็บเพจต่างๆ เป็นต้น คำสั่ง PHP จะถูกทำงานฝั่งเซิร์ฟเวอร์และส่งข้อมูลแสดงผลกลับไปยังเบราว์เซอร์ของผู้ใช้งาน
4. Perl: เป็นภาษาโปรแกรมเพื่อดำเนินการต่างๆ ในเว็บแอปพลิเคชัน แต่ใน XAMPP ในปัจจุบันมักไม่นิยมใช้งาน Perl มากนัก เนื่องจากความนิยมในการใช้งานน้อยลง และมักใช้ PHP เป็นหลัก
5. phpMyAdmin: เป็นเครื่องมือในการจัดการและจัดการฐานข้อมูล MySQL ในรูปแบบกราฟิกเว็บเบส ซึ่งช่วยให้ผู้ใช้งานสามารถดูข้อมูลในฐานข้อมูล, สร้างตาราง, เพิ่ม/ลบ/แก้ไขข้อมูล, และดำเนินการอื่นๆ ในระบบฐานข้อมูล MySQL ได้อย่างสะดวกสบาย

Composer คือเครื่องมือในการจัดการและบริหารจัดการกับแพ็คเกจ (packages) หรือไลบรารี (libraries) ในภาษา PHP ซึ่งเป็นตัวช่วยที่สำคัญในการพัฒนาและจัดการโปรเจค PHP ของคุณ โดยเฉพาะอย่างยิ่งในโปรเจค Laravel ซึ่งมีการใช้งาน Composer เป็นอย่างส่วนใหญ่

Composer ช่วยให้คุณสามารถดาวน์โหลดและติดตั้งแพ็คเกจที่ต้องการจาก Packagist (ที่เป็นคลังข้อมูลแพ็คเกจ PHP ที่มีจำนวนมาก) โดยอัตโนมัติ เพื่อให้คุณสามารถนำแพ็คเกจเหล่านี้มาใช้ในโปรเจค PHP ของคุณได้ง่ายๆ

สิ่งที่คุณสามารถทำได้ด้วย Composer รวมถึง:

1. ติดตั้งแพ็คเกจ PHP: คุณสามารถระบุแพ็คเกจที่คุณต้องการใช้ในโปรเจคของคุณในไฟล์ composer.json และใช้คำสั่ง composer install เพื่อดาวน์โหลดและติดตั้งแพ็คเกจที่ระบุในโปรเจคของคุณ
2. อัปเดตแพ็คเกจ PHP: คุณสามารถอัปเดตแพ็คเกจที่มีอยู่ในโปรเจคของคุณเพื่อให้ได้เวอร์ชันล่าสุดได้ด้วยคำสั่ง composer update
3. แก้ไขความขัดแย้งแพ็คเกจ: เมื่อคุณมีแพ็คเกจหลายๆ ตัวที่มีความขัดแย้งกัน เช่น การใช้เวอร์ชันของแพ็คเกจที่ไม่เข้ากัน คุณสามารถใช้ Composer เพื่อแก้ไขและเรียกใช้เวอร์ชันที่เข้ากันได้
4. โหลดแพ็คเกจที่กำหนดเอง: หากคุณพัฒนาแพ็คเกจของตัวเอง คุณสามารถสร้างและโหลดแพ็คเกจที่คุณกำหนดเองได้ด้วย Composer

5. จัดการ Autoloading: Composer ช่วยให้คุณไม่ต้องกังวลเกี่ยวกับ Autoloading ของไฟล์ PHP ที่คุณเขียน มันจะทำการสร้าง Autoloader ให้คุณโดยอัตโนมัติเมื่อคุณติดตั้งแพ็คเกจ

โดยทั้งนี้ การใช้ Composer ช่วยเพิ่มความสะดวกในการจัดการและพัฒนาโปรเจค PHP โดยให้คุณสามารถใช้งานแพ็คเกจต่างๆ ที่มีอยู่ใน Packagist มาใช้ในโปรเจคของคุณได้ง่ายและรวดเร็วมากยิ่งขึ้น ซึ่งทำให้การพัฒนาโปรแกรม PHP มีความสะดวกสบายและมีประสิทธิภาพมากยิ่งขึ้น

Artisan เป็นเครื่องมือใน Laravel ที่ใช้ในการทำงานคำสั่งต่างๆ ซึ่งช่วยให้นักพัฒนาและผู้ใช้งานสามารถจัดการและทำงานกับโครงสร้างของโปรเจค Laravel ได้อย่างง่ายและสะดวกสบาย นักพัฒนาสามารถเรียกใช้คำสั่ง Artisan ผ่าน Command Line Interface (CLI) หรือ Terminal เพื่อทำการสร้างไฟล์, สร้างฐานข้อมูล, รัน Migrations, สร้าง Model, Controller, Middleware และอื่นๆ ตามคำสั่งที่ต้องการ

คำสั่งที่ใช้ใน Artisan มีหลากหลายและแตกต่างกันขึ้นอยู่กับเวอร์ชัน Laravel และ Package ที่ติดตั้งในโปรเจคนั้นๆ ดังนี้เป็นตัวอย่างของคำสั่ง Artisan ที่ใช้บ่อยๆ:

php artisan make:controller ControllerName: สร้าง Controller ใหม่

php artisan make:model ModelName: สร้าง Model ใหม่

php artisan make:middleware MiddlewareName: สร้าง Middleware ใหม่

php artisan make:migration create_table_name: สร้าง Migration เพื่อสร้างตารางในฐานข้อมูล

php artisan migrate: รัน Migration เพื่อสร้างตารางในฐานข้อมูล

php artisan db:seed: รัน Seeder เพื่อใส่ข้อมูลสำหรับการทดสอบในฐานข้อมูล

php artisan route:list: แสดงรายการเส้นทาง (Routes) ทั้งหมดในโปรเจค Laravel

php artisan serve: เริ่มเซิร์ฟเวอร์ในโหมด Development บน localhost เพื่อรันแอปพลิเคชัน Laravel ที่พัฒนา

php artisan key:generate: สร้าง Application Key สำหรับโปรเจค Laravel

นอกจากนี้ Artisan ยังมีคำสั่งอื่นๆ ที่ช่วยให้นักพัฒนาสามารถจัดการกับแอปพลิเคชัน Laravel ได้อย่างง่ายและมีประสิทธิภาพมากยิ่งขึ้น การใช้ Artisan เป็นสิ่งที่สำคัญในการพัฒนาและทำงานกับโครงสร้างของโปรเจค Laravel และช่วยให้กระบวนการพัฒนาเว็บแอปพลิเคชันเป็นไปอย่างมีระเบียบและเป็นระบบ นอกจากนี้ยังช่วยประหยัดเวลาและความพยุงที่ใช้ในการพัฒนาอีกด้วย