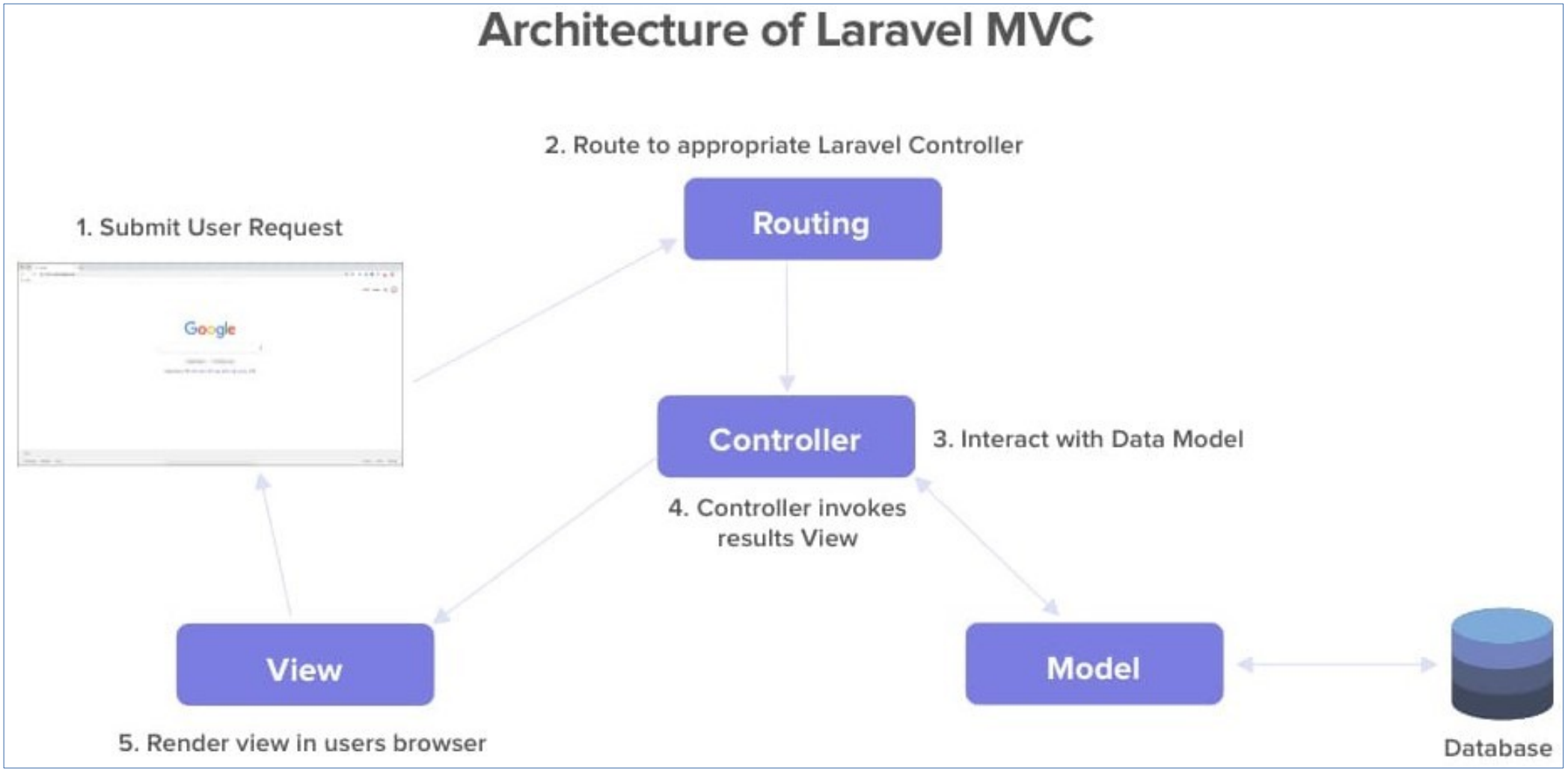
UECS3294 ADVANCED WEB APPLICATION DEVELOPMENT
CHAPTER 4 : FORMS and VALIDATION

LOO YIM LING
ylloo@utar.edu.my

Previously - Laravel Framework Architecture



Information available on https://www.netsolutions.com/insights/laravel-framework-benefits/

Forms and Validation

1) Validation Logics
2) Displaying Errors
3) Middleware

Validation

1) Laravel provides several different approaches to validate web application's incoming data. It is most common to use the `validate` method available on all incoming HTTP requests. However, we will discuss other approaches to validation as well.
2) Laravel includes a wide variety of convenient validation rules to be applied to data, even providing the ability to validate if values are unique in a given database table.

Validation (Writing The Validation Logic)

1) If validation fails during a traditional HTTP request, a redirect response to the previous URL will be generated.
2) Documentation of validation rules available:
3) https://laravel.com/docs/8.x/validation#available-validation-rules

```
public function store(Request $request)
{
    $validated = $request->validate([
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ]);
}
```

Validation (Stopping On First Validation Failure)

1) In case if one wish to stop running validation rules on an attribute after the first validation failure, the `bail` rule need to be assigned to the attribute.

```
$request->validate([
    'title' => 'bail|required|unique:posts|
        max:255',
    'body' => 'required',
]);
```

Validation (A Note On Nested Attributes)

1) If the incoming HTTP request contains "nested" field data, you may specify these fields in your validation rules using "dot" syntax.

```
$request->validate([
    'title' => 'required|unique:posts|max:255',
    'author.name' => 'required',
    'author.description' => 'required',
]);
```

Displaying Errors

1) So, what if the incoming request fields do not pass the given validation rules? Laravel will automatically redirect the user back to their previous location. In addition, all of the validation errors and request input will automatically be flashed to the session.
2) An `$errors` variable is shared with all of your application's views by the `Illuminate\View\Middleware\ShareErrorsFromSession` middleware, which is provided by the `web` middleware group.

Displaying Errors

1) When this middleware is applied an `$errors` variable will always be available in your views, allowing you to conveniently assume the `$errors` variable is always defined and can be safely used. The `$errors` variable will be an instance of `Illuminate\Support\MessageBag`.

```
@if ($errors->any())
   @foreach ($errors->all() as $error)
      <li>{{ $error }}</li>
   @endforeach
@endif
```

Displaying Errors

1) The `@error` Blade directive to quickly determine if validation error messages exist for a given attribute. Within an `@error` directive, one may echo the `$message` variable to display the error message.

```
<input id="title" type="text"
class="@error('title') is-invalid @enderror">

@error('title')
    <div class="alert alert-danger">{{ $message
}}</div>
@enderror
```

Customizing Error Messages

1) Laravel's built-in validation rules each has an error message that is located in web application's `resources/lang/en/validation.php` file. Within this file, one will find a translation entry for each validation rule. One is free to change or modify these messages based on the needs of the web application.
2) In addition, this file can be copied to another translation language directory to translate the messages for web application's language.

Middleware

1) Middleware provide a convenient mechanism for inspecting and filtering HTTP requests entering the web application.
2) For example, Laravel includes a middleware that verifies the user of the web application is authenticated. If the user is not authenticated, the middleware will redirect the user to web application's login screen.
3) However, if the user is authenticated, the middleware will allow the request to proceed further into the application.

Middleware

1) Additional middleware can be written to perform a variety of tasks besides authentication. For example, a logging middleware might log all incoming requests to the web application.
2) There are several middleware included in the Laravel framework, including middleware for authentication and CSRF protection. All of these middleware are located in the `app/Http/Middleware` directory.

Middleware: Creating Middleware

1) To create a new middleware, use the `make:middleware` Artisan command
2) The example command below will place a new `EnsureTokenIsValid` class within `app/Http/Middleware` directory.

```
php artisan make:middleware EnsureTokenIsValid
```

Middleware: Registering Middleware

1) If one wants a middleware to run during every HTTP request to web application, list the middleware class in the `$middleware` property of `app/Http/Kernel.php` class.
2) There are three types of middleware namely Global, Group and Route. Each can me registered in different middleware stacks within HTTP Kernel (according to Laravel comments inside the class).

Middleware: Assigning Middleware

Group Middleware:

```
Route::group(['middleware' =>
['ensureToken']], function()
    {
        Route::view("signUp", "signUp");
        Route::view("contact",'contact');
    }
);
```

Route Middleware:

```
Route::view("login", "login")->
middleware('ensureToken');
```

END OF LECTURE 05