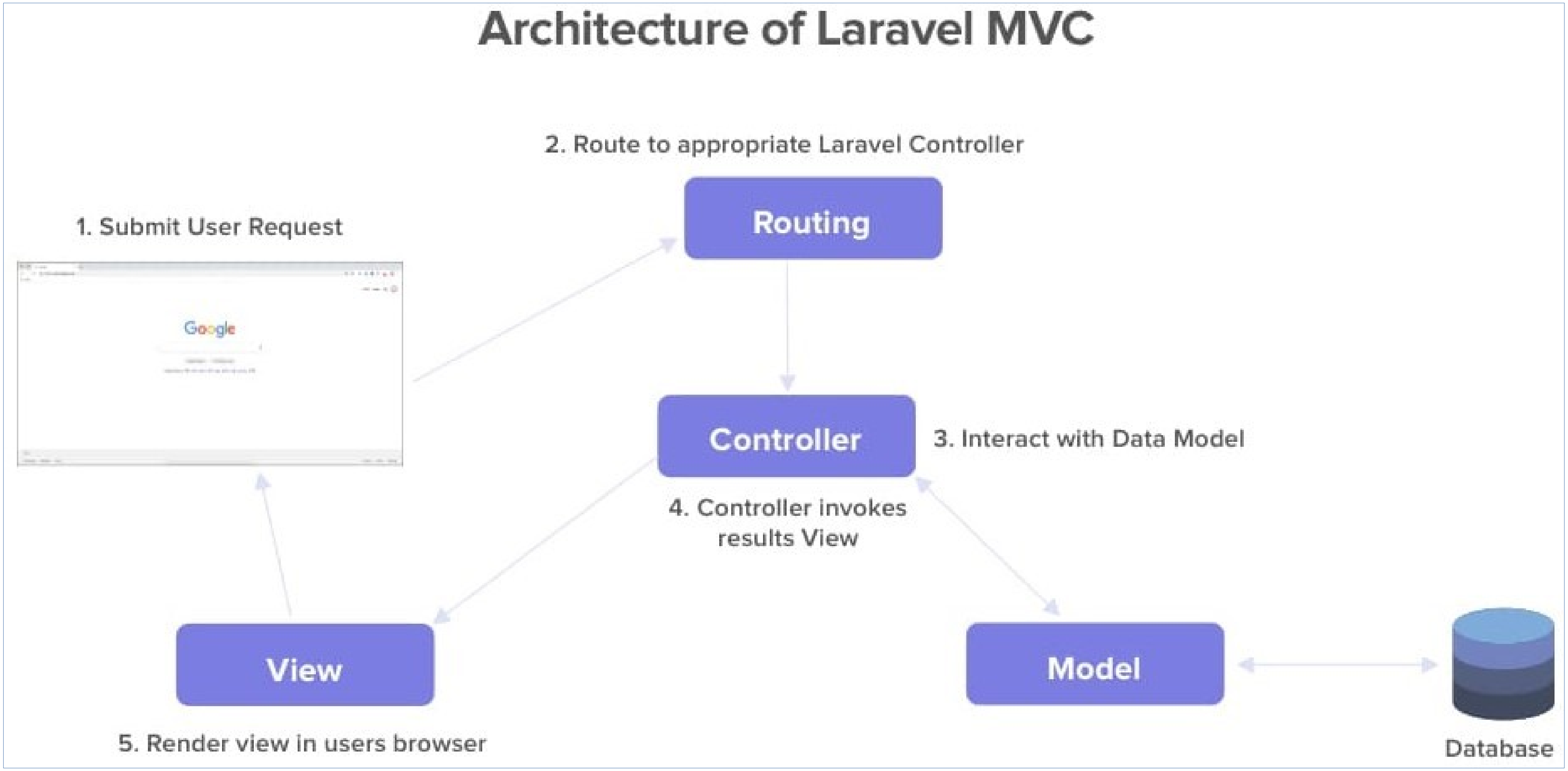
UECS3294 ADVANCED WEB APPLICATION DEVELOPMENT
CHAPTER 5 : MAINTAINING STATE

LOO YIM LING
ylloo@utar.edu.my

Previously - Laravel Framework Architecture


Architecture of Laravel MVC
1. Submit User Request
Google
2. Route to appropriate Laravel Controller
Routing
Controller
3. Interact with Data Model
4. Controller invokes results View
View
Model
Database
5. Render view in users browser

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

Maintaining State

1) **Cookies**
2) **Session**

Cookies

1) All cookies created by the Laravel framework are encrypted and signed with an authentication code, meaning they will be considered invalid if they have been changed by the client.
2) Use the cookie method on a `Illuminate\Http\Request` instance to retrieve a cookie value from the request

```
$value = $request->cookie('name');

$value = Cookie::get('name');
```

Cookies

1) The `cookie` method on response instances enable to easily attach cookies to the response

```
return response($content) ->header('Content-Type',
$type) ->cookie('name', 'value',
$expire_in_minutes, $path, $domain, $secure,
$httpOnly);
```

Cookies

1) Alternatively, use the `Cookie` facade to queue cookies for attachment to the outgoing response from your application.
2) The queue method accepts a `Cookie` instance or the arguments needed to create a `Cookie` instance.
3) These cookies will be attached to the outgoing response before it is sent to the browser

```
Cookie::queue('name', 'value', $minutes);
```

Session

1) Since HTTP driven applications are stateless, sessions provide a way to store information about the user across multiple requests.
2) Laravel ships with a variety of session backends that are accessed through an expressive, unified API.
3) Support for popular backends such as Memcached, Redis, and databases is included out of the box

Session: Configuration

1) The session configuraation file is stored at `config/session.php`
2) By default, Laravel is configured to use the `file` session driver, which will work well for many applications.
3) If the web application will be load balanced across multiple web servers, centralized store that all servers can access should be chosen, such as Redis or a database

Session: Configuration

1) The session `driver` configuration option defines where session data will be stored for each request. Laravel ships with several great drivers out of the box:

- `file` - sessions are stored in `storage/framework/sessions`.
- `cookie` - sessions are stored in secure, encrypted cookies.
- `database` - sessions are stored in a relational database.
- `memcached` / `redis` - sessions are stored in one of these fast, cache based stores.
- `dynamodb` - sessions are stored in AWS DynamoDB.
- `array` - sessions are stored in a PHP array and will not be persisted.

Session: Retrieving Data

1) There are two primary ways of working with session data in Laravel
   - The global session helper
   - via a Request instance by type-hinting in a controller method.
2) A Request instance is shown below:

```
public function show(Request $request, $id)
{
   $value = $request->session()->get('key');
}
```

Session: Retrieving Data

1) Global `session` PHP function can be used to retrieve and store data in the session.
2) When the `session` helper is called with a single, string argument, it will return the value of that session key.
3) When the helper is called with an array of key / value pairs, those values will be stored in the session.

Session: Retrieving Data

```
Route::get('/home', function () {
    // Retrieve a piece of data from the
       session...
    $value = session('key');

    // Specifying a default value...
    $value = session('key', 'default');

    // Store a piece of data in the session...
    session(['key' => 'value']);
});
```

Session: Retrieving Data

1) To retrieve all the data in the session, the `all` method can be used:

```
$data = $request->session()->all();
```

Session: Retrieving Data

1) To determine if an item is present in the session, the `has` method may be used. The `has` method returns `true` if the item is present and is not `null`.
2) To determine if an item is present in the session, even if its value is null, use the `exists` method

```
if ($request->session()->has('users')) {
}

if ($request->session()->exists('users')) {
}
```

Session: Storing Data

1) To store data in the session, use the request instance's `put` method or the `session` helper

```
// Via a request instance...
$request->session()->put('key', 'value');

// Via the global "session" helper...
session(['key' => 'value']);
```

Session: Storing Data

1) The `push` method may be used to push a new value onto a session value that is an array.
2) For example, if the `user.teams` key contains an array of team names, push a new value onto the array as such:

```
$request->session()->push('user.teams',
'developers');
```

Session: Retrieve and Delete

1) The `pull` method will retrieve and delete an item from the session in a single statement

```
$value = $request->session()->pull('key',
'default');
```

Session: Flash Data

1) Sometimes one may wish to store items in the session for the next request. Do so using the `flash` method.
2) Data stored in the session using this method will be available immediately and during the subsequent HTTP request.
3) After the subsequent HTTP request, the flashed data will be deleted.
4) Flash data is primarily useful for short-lived status messages

```
$request->session()->flash('status', 'Task
was successful!');
```

Session: Flash Data

1) If need to persist flash data for several requests, use the `reflash` method, which will keep all of the flash data for an additional request.
2) If only need to keep specific flash data, use the `keep` method

```
$request->session()->reflash();

$request->session()->keep(['username',
'email']);
```

Session: Deleting Data

1) The `forget` method will remove a piece of data from the session.
2) To remove all data from the session, use the `flush` method

```
// Forget a single key...
$request->session()->forget('name');

// Forget multiple keys...
$request->session()->forget(['name',
'status']);

$request->session()->flush();
```

Session: Regenerating Session ID

1) Regenerating the session ID is often done in order to prevent malicious users from exploiting a session fixation attack on a web application.
2) Laravel automatically regenerates the session ID during authentication if one is using one of the Laravel application starter kits or Laravel Fortify

Session: Regenerating Session ID

1) If need to manually regenerate the session ID, use the `regenerate` method
2) If need to regenerate the session ID and remove all data from the session in a single statement, use the `invalidate` method

```
$request->session()->regenerate();

$request->session()->invalidate();
```

END OF LECTURE 06