Contents

Author: Foster Amponsah Asante

LESSON 1: PHP INSTALLATION

a. Installing multiple php versions

1. Download the php version you prefer
2. Unzip the binary file
3. Copy the unzipped directory/folder to a safe place (preferably Drive C)
4. Open the unzipped php directory and locate to the php executable file
5. Copy the path and add to Environment variables (**System Variables**)
6. When all the above are done, you can easily switch your php versions

LESSON 2: LARAVEL CONTROLLERS

In Laravel, controllers are classes that handle the logic of incoming HTTP requests and generate appropriate responses.

Key Roles of Controllers in Laravel:

1. **Handling HTTP Requests:** Controllers receive incoming HTTP requests (GET, POST, PUT, DELETE, PATCH etc.) and process them according to their defined logic.

2. **Processing Request Data:** They extract and validate data from the request, such as form input, query parameters, or request body.

3. **Interacting with Models:** Controllers often interact with models to retrieve or manipulate data from the database.

4. **Generating Responses:** Based on the processed data and business logic, controllers generate appropriate responses, which can be HTML views, JSON data, redirects, or other formats.

5. **Routing Requests:** Controllers are linked to specific routes defined in the routes directory. When a request matches a route, the corresponding controller method is executed.

a. Types of Controllers in Laravel:

Basic Controllers: These are simple classes with methods that handle specific HTTP requests. They are suitable for handling individual actions or simple use cases.

Artisan Command

```
php artisan make:controller AboutController
```

Controller code

```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class AboutController extends Controller
{
    public function index()
    {
        return view('about');
    }
}
```

Route

```
Route::get('/about', [AboutController::class, 'index']);
```

Resource Controllers: These controllers are designed to handle all CRUD (Create, Read, Update, Delete) operations for a specific resource. They provide a convenient way to define RESTful APIs.

Artisan Command

```
php artisan make:controller PostController –resource
```

Controller code

```
<?php

    namespace App\Http\Controllers;

    use App\Models\Post;
    use Illuminate\Http\Request;

    class PostController extends Controller
    {
        /**
         * Display a listing of the resource.
         */
        public function index()
        {
            $posts = Post::all();
            return view('posts.index', compact('posts'));
        }

        // ... other CRUD methods (create, store, show, edit, update, destroy)
    }
```

Route

```
Route::resource('posts', PostController::class);
```

Form Request Controllers: These controllers are used to validate incoming form requests and provide a more structured approach to input validation.

Artisan Command

```
php artisan make:request StorePostRequest
```

Controller Code

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;
```

```
class StorePostRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     */
    public function authorize(): bool
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     */
    public function rules(): array
    {
        return [
            'title' => 'required|string|max:255',
            'body' => 'required|string',
        ];
    }
}
```

Usage in controller

```
public function store(StorePostRequest $request)
{
    Post::create($request->validated());
    return redirect()->route('posts.index');
}
```

LESSON 3: Route Parameters

- **Purpose:** Identify a specific resource within a collection.
- **Syntax:** Embedded directly into the URL path, enclosed in curly braces.
- **Example:** /users/{userId}
- **Use Cases:**
  - Fetching a specific user by their ID.
  - Retrieving a particular product by its SKU.
  - Accessing a detailed article by its unique slug.

```
Route::get('/users/{userId}', function ($userId) {

  // Access the userId parameter here

  return 'User ID: ' = $userId;

});
```

LESSON 4: Query Strings

- **Purpose:** Filter, sort, or paginate a collection of resources.
- **Syntax:** Appended to the URL after a question mark, using key-value pairs separated by ampersands.
- **Example:** /products?category=electronics&sort=price&page=2
- **Use Cases:**
  - Filtering products by category.
  - Sorting products by price or popularity.
  - Paginating results to improve performance and user experience.

```
Route::get('/products', function (Request $request) {
  $category = $request->query('category');
  $sort = $request->query('sort');
  $page = $request->query('page');

  // Use the query parameters to filter, sort, and paginate products
});
```

OR

```
Route::get('/products', function (Request $request) {
    $category = $request->input('category');
    $sort = $request->input('sort');
    $page = $request->input('page');

    // ...
});
```

OR

```
Route::get('/products', function (Request $request) {
    $category = $request->category;
    $sort = $request->sort;
    $page = $request->page;

    // ...
});
```

LESSON 4: MODEL BINDING

Model binding is a powerful feature in Laravel that allows you to automatically inject (**makes it available**) Eloquent model instances into your routes based on URL parameters. This simplifies your controller logic and makes your code more concise and readable.

b. Types of Model Binding (Binding a model to a route):

1. **Implicit Model Binding:**
    - Laravel automatically binds models to route parameters based on their names.
    - If a route parameter matches the name of an Eloquent model, Laravel will attempt to find a model with that ID.
    - **Example:**

```
Route::get('/users/{user}', function (User $user) {
    // $user is an instance of the User model
});
```

2. **Explicit Model Binding:**

- You can explicitly bind a route parameter to a model using the **Route::model()** method.
- This allows you to customize the binding behavior, such as specifying a custom key or a different model.
- **Example:**

```
Route::model('user', User::class, 'username');

Route::get('/users/{user}', function (User $user) {
    // $user is an instance of the User model, found by username
});
```

Customizing Model Binding:

- **Customizing the Key:** You can specify a custom key to use for the model lookup:

  ```
  Route::model('post', Post::class, 'slug');
  ```

- **Customizing the Model:** You can specify a different model to be bound:

  ```
  Route::model('user', AdminUser::class);
  ```

- **Customizing the 404 Response:** You can customize the 404 response that is generated when a model is not found:

  ```
  Route::missing(function () {
      return response()->view('errors.404', [], 404);
  });
  ```