Connection of Controller with View

Objective: Understand how controllers work and how to route requests to them.

1. Create a Simple Controller

Step 1: Command to create a controller:

```
    php artisan make:controller ExampleController
```

It will create a controller named “**ExampleController**” in “**ProjectName/app/Http/Controller**” directory.

Step 2: Open the **ExampleController** and define a method:

```
ExampleController.php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ExampleController extends Controller
{

    public function showMessage()
{
    $message = "Hello from the Controller!";
    return view('message', ['message' => $message]);
}
}
```

Step 3: Create a message.blade.php file for viewing the message fetched from the ExampleController

```
<html>
<head>
    <title>Message</title>
</head>
<body>
    <h1>{{ $message }}</h1>
</body>
</html>
```

Step 4: Map the controller method to a route in web.php:

```
use App\Http\Controllers\ExampleController;
Route::get('/message', [ExampleController::class, 'showMessage']);
```

Now, run the php artisan serve command in the terminal and go to http://127.0.0.1:8000/message. If the page shows Hello from the Controller!" message then you have **successfully** make a connection between view and controller.

2. Pass Data from Controller to View

Objective: Learn how to pass and display data without a database.

Step 1: Define a method in ExampleController:

```
public function showView()
{
    $data = [
        'title' => 'Welcome to Laravel',
        'description' => 'This is a demo without a database.',
    ];
    return view('example', $data);
}
```

Step 2: Create the view resources/views/example.blade.php

```
<h1>{{ $title }}</h1>
<p>{{ $description }}</p>
```

Step 3: Map a route to the method

```
Route::get('/example', [ExampleController::class, 'showView']);
```

3. Use Static Arrays to Simulate Data

Objective: Simulate working with data as if it came from a database.

Step 1: Define a method to return multiple items:

```
public function listItems()
{
    $items = [
        ['id' => 1, 'name' => 'Item 1', 'price' => 100],
        ['id' => 2, 'name' => 'Item 2', 'price' => 200],
        ['id' => 3, 'name' => 'Item 3', 'price' => 300],
    ];
    return view('items', ['items' => $items]);
}
```

Step 2: Create the view resources/views/items.blade.php:

```
<h1>Items</h1>
<ul>
    @foreach ($items as $item)
        <li>{{ $item['name'] }} - ${{ $item['price'] }}</li>
    @endforeach
</ul>
```

Step 3: Add a route:

```
Route::get('/items', [ExampleController::class, 'listItems']);
```

4. Handle User Input

Objective: Demonstrate how controllers process form submissions.

Step 1: Create a form in resources/views/form.blade.php:

```
<form action="/submit" method="POST">
    @csrf   <!-- CSRF token for security against cross-site request forgery attacks -->
    <label for="name">Name:</label>
    <input type="text" id="name" name="name">
    <button type="submit">Submit</button>
</form>
```

Step 2: Define a method in ExampleController to handle the form submission:

```
public function handleForm(Request $request)
{
    $name = $request->input('name');
    return "Form submitted! Hello, $name!";
}
```

Step 3: Add routes:

```
Route::get('/form', function () {
    return view('form');
});
Route::post('/submit', [ExampleController::class, 'handleForm']);
```