**Step 6: Routing and Controllers**

****

In this step, you'll define how your application responds to client requests to different URIs (Uniform Resource Identifiers). You'll map URIs to specific controller methods.

**Routing**:

1. The **index.php** file in the **public** directory acts as the front controller. It's the entry point for all requests.
2. Inside **index.php**, you have a **$routes** array that defines your application's routes. This array maps URIs to controller actions.

**Creating Controllers**:

1. Create controllers for handling specific parts of your application. We'll use **PostController** to handle the front-end blog posts.
2. In the **app/controllers** directory, define the **PostController**, **AdminController**, **CategoryController**, **CommentController**, and **UserController**.

**Defining Methods**:

1. In **PostController**, define methods like **index()** for displaying a list of posts and **show($id)** for displaying a single post.

**Mapping Routes**:

1. Define routes in the **$routes** array corresponding to the methods in your controllers.
2. For example, route the home URI **'/'** to **PostController->index** and **'posts/show/([0-9]+)'** to **PostController->show**.

**Testing Routes**:

1. Access the routes in your web browser to make sure they trigger the correct controller methods.
2. Initially, you can add simple **echo** statements in the controller methods to check if they're being invoked.

Here's an example of what your controller and routing might look like:

**PostController.php**:

<?php

namespace app\controllers;

class PostController {

    public function index() {

        // Code to fetch and display posts

        echo "All Posts displayed here";

    }

    public function show($id) {

        // Code to display a single post based on the $id

        echo "A Single post displayed";

    }

}

And the index.php file will have the following code

**public/index.php:**

<?php

require \_\_DIR\_\_ . '/../bootstrap.php';

// Extract the path component from the full URL of the current request

$request = trim(parse\_url($\_SERVER['REQUEST\_URI'], PHP\_URL\_PATH), '/');

e.g ***http://phpmvcblog.local/posts/show/1, extract*** *‘****posts/show/1’1***)This is what I mean

// Further trim any leading or trailing slashes from the path

$request = trim($request, '/');

$routes = [

    'GET' => [

        '' => ['controller' => '\app\controllers\PostController', 'method' => 'index'],

        'posts/show/([0-9]+)' => ['controller' => 'app\controllers\PostController', 'method' => 'show'],

        // Add more routes for other controllers and actions

    ],

    // POST routes...

];

// Retrieve the URL path from the current request

$path = $request; (e.g ***http://phpmvcblog.local/posts/show/1, extract*** *‘****posts/show/1’1***)

// Obtain the HTTP method (e.g., GET, POST) of the current request

$method = $\_SERVER['REQUEST\_METHOD'];

// Iterate over each route defined for the current HTTP method

foreach ($routes[$method] as $route => $info) {

    // Modify the route string to a regular expression for matching URL patterns

    $pattern = preg\_replace('#/([0-9]+)#', '/([0-9]+)', $route);

    // Check if the current URL path matches the route pattern

    if (preg\_match("#^$pattern$#", $path, $matches)) { e.g ***posts/show/1 is equal to***

***'posts/show/([0-9]+)***

        // Create an instance of the controller specified in the route information

        $controller = new $info['controller']; e.g ***PostController***

        // Extract any numeric ID present in the URL, default to null if not found

        $id = $matches[1] ?? null;

        // Execute the controller method with parameters based on the request method

        // For POST requests (excluding delete operations), pass both POST data and ID

        if ($method === 'POST' && $info['method'] !== 'delete') {

            $controller->{$info['method']}($\_POST, $id);

        } else {

            // For other request methods, pass only the ID

            $controller->{$info['method']}($id);

        }

        // Exit the loop after finding and handling the matching route

        break;

    }

}

// If no route was matched, return a 404 response

if (!isset($controller)) {

    http\_response\_code(404);

    require BASE\_DIR . '/app/views/404.php';

}

**Here’s what is happening…**

* 1. **Error Reporting Configuration**: The code starts by configuring PHP to display all errors, warnings, and notices. For example, if there's a syntax error in your code, this configuration ensures that it is displayed in the browser, aiding in debugging during development.
	2. **Autoloader Inclusion**: The Composer autoloader is included, allowing classes to be loaded on-demand without manual **require** statements. For instance, when you instantiate a new **PostController**, Composer will automatically include the **PostController.php** file based on the namespace and class name.
	3. **Request URI Extraction**: The script extracts the current request's URI path. For example, if a user accesses **http://php8blogapp.local/posts/show/1**, the script extracts **posts/show/1** as the path, which is used to find the corresponding controller and method.
	4. **Routes Definition**: An associative array **$routes** defines the available routes, linking URI patterns to controllers and methods. For instance, the pattern **'posts/show/([0-9]+)'** is linked to **PostController**'s **show** method, expecting a numeric post ID as a parameter.
	5. **Route Matching Logic**: The code iterates over the routes, using regular expressions to match the extracted path against the route patterns. For example, if the extracted path is **posts/show/1**, the regex will match this against **'posts/show/([0-9]+)'** in the **$routes** array.
	6. **Controller Instantiation and Method Invocation**: When a route match is found, a new controller object is instantiated, and the appropriate method is called with any parameters from the URL. For instance, matching **posts/show/1** instantiates **PostController** and calls **show(1)**.
	7. **404 Error Handling**: If no route matches the request, an HTTP 404 status code is sent, and a 404 error page is displayed. So if a user tries to access **http://php8blogapp.local/nonexistent/path**, they will be presented with the 404 error view.

Through this setup, **index.php** dynamically handles routing, creating a flexible and maintainable entry point for your web application.

Finally Update the ***public/.htaccess* file** with this code

RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteRule ^(.+)$ index.php?url=$1 [QSA,L]

The **.htaccess** file used with Apache web server sets up URL rewriting rules. Here's an explanation of each line:

1. **RewriteEngine On**:
	* This line enables the runtime rewriting engine provided by mod\_rewrite, an Apache module. It allows the server to manipulate URLs before determining the appropriate way to handle a request.
2. **RewriteCond %{REQUEST\_FILENAME} !-f**:
	* This line is a condition for the following **RewriteRule**. It checks if the requested filename does not correspond to an existing file (**!-f**). In other words, if the URL points to a file that actually exists on the server (like an image, a CSS file, etc.), the rule will not be applied.
3. **RewriteCond %{REQUEST\_FILENAME} !-d**:
	* Similar to the previous line, this condition checks if the requested filename does not correspond to an existing directory (**!-d**). So, if the URL points to an existing directory, the rule will not be applied.
4. **RewriteRule ^(.+)$ index.php?url=$1 [QSA,L]**:
	* This line is where the actual URL rewriting occurs. The **RewriteRule** directive defines a rule for rewriting the URL.
	* **^(.+)$** is a regular expression that matches any request (excluding the root **/**). The **^** symbol indicates the start of the request, **(.+)** captures one or more of any character, and **$** indicates the end of the request.
	* **index.php?url=$1** is the target of the rewrite. This means that any request (except for real files or directories) is redirected internally to **index.php**. The original request is passed as a parameter **url**.
	* **[QSA,L]** are flags for the rewrite rule:
		+ **QSA** (Query String Append) means that if there's a query string present on the original URL, it will be appended to the rewrite target.
		+ **L** (Last) indicates that this should be the last rule applied; if this rule is matched, no subsequent rules will be processed.

In summary, this **.htaccess** ***setup directs all requests (except for existing files and directories) to*** **index.php**, allowing a PHP application to handle routing. It's a common setup for PHP frameworks that implement a front controller pattern, where **index.php** acts as a single entry point for all requests.