Build A Full PHP MVC Blog From Scratch

Step 1: Environment Setup

- Install XAMPP: Begin by downloading and installing XAMPP from the official website. This package contains the necessary tools such as Apache server, MySQL, and PHP to run the blog locally.

Here is the link to the xampp software or copy the link ***https://www.apachefriends.org/download.html*** and paste it on a browser URL bar.

- Visual Studio Code: Make sure to have Visual Studio Code installed. Enhance your coding experience with PHP extensions like PHP Intelephense, which provides improved code navigation and IntelliSense.

Get it here https://code.visualstudio.com/

Step 2: Project Initialization

- Create a New Project Directory: Establish the workspace by creating a new directory dedicated to this project.

```
cd xampp/htdocs
mkdir xampp/htdocs/PHPMVCBlog
```

and create your project there.

Mine I will name it **PHPMVCBlog.**

- Initialize Git: Implement version control by initializing a new Git repository within the project directory. This practice is essential for tracking changes and managing code versions throughout the development process.

```
git init
git add .
git commit -m "Initial commit"
git branch -M main
git remote add origin https://github.com/yourusername/PHPMVCBlog.git
git push -u origin master  # or 'main'
```

For incremental changes, the workflow is essentially:

1. Stage changes: **git add .** or **git add <specific-file-or-folder>**
2. Commit changes: **git commit -m "Your commit message"**
3. Push changes: **git push** (if the branch is already set up with upstream) or **git push -u origin main** (the first time you push your main branch, which you've already done).

Step 3: Composer and Autoloading

i) **Install Composer**: If not already installed, get Composer for managing dependencies.
How to check if installed.

```
Composer -v
```

If not installed go here *https://getcomposer.org/* follow the instructions and you're good to go.

- **Autoloading**: Set up autoloading using Composer by creating a **composer.json** file with the appropriate autoloading configuration.

ii) Set Up Project Autoloading with Composer

Once Composer is installed, you'll set up autoloading for your PHP MVC project. Here's how to do it:

1. Create the composer.json file:

- In your project root directory, create a file named **composer.json**.
- Add the following content to it:

```
{
    "name": "brian/my-mvc-blog",
    "description": "A Light Weight PHP Application",
    "type": "library",
    "license": "MIT",
    "autoload": {
        "psr-4": {
            "app\\": "app/"
        }
    },
    "authors": [
        {
            "name": "Brian"
        }
    ],
    "minimum-stability": "beta",
    "require": {}
}
```

The **composer.json** file is a configuration file for Composer, which manages dependencies for PHP projects. Here's a simple explanation of what this file does:

- **name**: This specifies a unique identifier for your project, typically using the format **vendor/package**.
- **description**: A brief description of what your project is.
- **type**: It indicates the type of project. Common types include **project** for applications and **library** for reusable packages.
- **license**: This defines the software license under which your project is released. MIT is one of the most open and permissive licenses.
- **autoload**: This section is important for defining how PHP class files are automatically loaded without the need for manual **require** statements. The **psr-4** entry tells Composer to use the PSR-4 standard for autoloading. It maps a namespace prefix to a directory.
    - In this case, any PHP class that starts with the namespace **app** will be autoloaded from the **app/** directory.
- **authors**: Lists the authors of the project.
- **minimum-stability**: This sets the minimum stability level for your dependencies. By setting it to **beta**, Composer will allow you to install dependencies that are still in beta.

- **require**: Lists the packages that your project depends on. Since it's empty, your project currently does not require any external packages.

The **autoload** section is crucial because it allows you to organize your project's PHP classes into namespaces, which helps prevent class name conflicts and makes your code easier to maintain. With autoloading, you don't need to manually include each class with a **require** or **include** statement. Instead, when you instantiate a new object, Composer's autoloader will automatically include the necessary file based on the namespace and class name.

iii. Generate the autoloader:

- Open the command prompt or terminal.
- Navigate to your project directory.
- Run **composer dump-autoload**. Composer will create the **vendor** directory and the autoloading files within it.

iv.Update .gitignore:

- Create a **.gitignore** file if it doesn't exist and add the following lines to avoid tracking vendor and other non-essential files:

```
/vendor/
/.vscode/
/node_modules/
```

Step 4: MVC Structure

i) PHP MVC Blog Structure

Let’s go ahead and create the project files and folders for our entire MVC Blog. Here is how it will look like.

```
PHPMVCBLOG/
|
├── app/
|   ├── controllers/     # Where you'll place your controller classes
|   ├── models/          # Where you'll place your model classes
|   └── views/           # Where you'll place your view files (HTML/PHP templates)
|
├── config/              # Configuration files (database connection, app settings,
etc.)
|
├── public/              # Publicly accessible files (index.php, assets like
CSS/JS/images)
|   ├── assets/
|   |   ├── css/
|   |   ├── js/
|   |   └── images/
|   └── index.php        # The front controller which bootstraps the application
|
├── src/                 # Additional PHP classes or libraries you might include
```

```
│
├── vendor/              # Composer dependencies and autoloader
│
├── composer.json        # Composer configuration file
├── composer.lock        # Lock file to ensure consistent installs across
environments
└── .gitignore           # Specifies intentionally untracked files to ignore
```

Note: *To easily create the folders and files use the **mkdir** for folders and **touch** command for files.*

You can run **composer dump-autoload** to autoload the files again.

Step 5: Database Setup

- **XAMPP MySQL**: Start the MySQL server through XAMPP and let's create a new database for our blog.

a) Create the Database Configuration File (db.php):

- Here we are going to create the ***config/db.php*** file which is going to define the database connectivity details and connect to our server. Here's how the file looks like.

```php
<?php

// Define database connection details
class Database {
    private $host = 'localhost'; // Database server address
    private $db_name = 'phpmvcblog'; // Database name
    private $username = 'root'; // Database username
    private $password = ''; // Database password (leave empty for security)
    public $conn; // PDO connection object

    // Establish database connection upon object creation
    public function __construct() {
        try {
            // Create a PDO connection
            $this->conn = new PDO(
                "mysql:host=$this->host;dbname=$this->db_name",
                $this->username,
                $this->password
            );

            // Set character encoding to UTF-8
            $this->conn->exec("set names utf8");
            echo "Connection Successful!";
        } catch (PDOException $exception) {
            // Handle connection error
            echo "Connection error: " . $exception->getMessage();
        }
    }
}
```

What's happening here?

1. **Private Variables**:
    - The **host**, **db_name**, **username**, and **password** variables are declared as **private** to enforce encapsulation, which is a key principle of object-oriented programming. ***Encapsulation means that the internal state of an object is hidden from the outside***, only allowing access through methods the class provides.
    - By declaring these variables as **private**, they cannot be accessed directly from outside the **Database** class. ***This helps to protect sensitive details like the username and password from being exposed or altered by other parts of the application or by code outside the class***.
2. **Public $conn Variable**:
    - The **$conn** variable is public because it needs to be accessible from outside the class to perform database operations. When a **Database** object is created, **$conn** holds the active PDO connection that can then be used for querying the database.
3. **Constructor and PDO Connection**:
    - The **__construct()** method is a special method that automatically gets called when a new instance of the **Database** class is created. ***($db_instance = new Database***)
    - Inside the constructor, a PDO connection is established, and the character encoding is set to UTF-8 to ensure proper handling of characters. The **try-catch** block is used to catch any exceptions that may occur during the connection process (like incorrect credentials or server issues).
4. **Error Handling**:
    - The **catch** block captures any **PDOException** that might occur if the connection fails and echoes a message. This is useful for debugging but ***should not be used in a production environment*** as it can reveal sensitive information.

b) How To Run Your PHP Project

When setting up a new PHP project with its own virtual host, especially when you want to run multiple projects simultaneously on your local development machine, you need to create a separate virtual host configuration for each project. Here's what you should do to set up a virtual host for your new PHP MVC blog project:

1. **Define a Unique ServerName**:
    - Each virtual host should have a unique **ServerName** directive. If our new project is called, for instance, **phpmvcblog.local**, we should use that as the **ServerName**.
2. **Set the DocumentRoot**:
    - The **DocumentRoot** directive should point to the **public** directory of your new project. If your project directory is **S:/xampp/htdocs/phpmvcblog/public**, then that should be your **DocumentRoot**.
3. **Configure the Directory Settings**:

- Similar to the **DocumentRoot**, the **<Directory>** directive should point to the **public** directory of your new project and include the proper settings to allow for URL rewriting and access permissions.

4. **Set Up Logs**:
    - It's a good practice to have separate error and access logs for each project. Adjust the **ErrorLog** and **CustomLog** directives to point to new log files specific to your new project.
5. **Update Your Hosts File**:
    - On Windows, you will need to update the **hosts** file to point the new **ServerName** to your local machine. Add a line like **127.0.0.1 phpmvcblog.local** to the **hosts** file located at **C:\Windows\System32\drivers\etc\hosts**.
6. **Restart Apache**:
    - After making changes to the virtual host configuration file and **hosts** file, you will need to restart the Apache server for the changes to take effect.

Here is an example configuration for our new project:

```
<VirtualHost *:80>
    DocumentRoot "S:/xampp/htdocs/phpmvcblog/public"
    ServerName phpmvcblog.local

    <Directory "S:/xampp/htdocs/phpmvcblog/public">
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog "logs/phpmvcblog-error.log"
    CustomLog "logs/phpmvcblog-access.log" common
</VirtualHost>
```

c) Test Your Database Connectivity

To test our database connection to the server. Let's create a ***test_db.php*** within the ***public folder***. Here's how you need to do it.

Run ***touch public/test_db.php.*** then update the file with this code.

```
<?php
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

// Define BASE_DIR as the absolute path to your project's root
define('BASE_DIR', dirname(__DIR__));

// Include the database configuration and connection code
```

```
require BASE_DIR . '/config/db.php';

// Create a new instance of the Database class to initiate a connection
$db_instance = new Database();

// You can now use $db_instance->conn to perform database operations
```

The variable **$db_instance** holds the instance of the **Database** class. During its construction, the **__construct()** method is invoked, and it's within this method that the **$conn** property of the **Database** class is assigned a value — specifically, an instance of the PDO class, representing the connection to the database

So finally run the url ***http://phpmvcblog.local/test_db.php***. This is just a test file you can go ahead and delete it since the database connectivity is working.

e) Let's Bootstrap Our Project

In this part we are going to structure our project. This basically means how we are going to call different files within our app. Let's go ahead and do that.

public/index.php

```
 <?php


require __DIR__ . '/../bootstrap.php';
```

PHPMVCBlog/bootstrap.php

```
<?php
// Start the session
session_start();

// Set error reporting
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

// Define BASE_DIR as the absolute path to your project's root
define('BASE_DIR', __DIR__);


// Include Composer's autoloader
require_once BASE_DIR . '/vendor/autoload.php';

// Include utility functions
require_once BASE_DIR . '/utils.php';

// Include the database configuration and connection setup
require_once BASE_DIR . '/config/db.php';
```

f) Database Migration Scripts:

- Let's use the **create_*.php** files to create the tables in our database.We will start by creating posts table. Since our DocumentRoot is set to the **public** directory, the **create_*.php** files should be in the **public** directory of our PHP project. If it's elsewhere, the server won't be able to find it.

public/create_categories_table.php

```php
<?php

require __DIR__ . '/../bootstrap.php';

$db = new Database;

// Create categories table
$query = "CREATE TABLE IF NOT EXISTS categories (
    id INT(11) AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)";
$db->conn->exec($query);

echo 'Categories table created successfully';
?>
```

Public/create_posts_table.php

```php
<?php

require __DIR__ . '/../bootstrap.php';

$db = new Database;

// Create Posts table
$query = "CREATE TABLE IF NOT EXISTS posts (
    id INT(11) AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    category_id INT(11) DEFAULT NULL,
    image VARCHAR(255) DEFAULT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (category_id) REFERENCES categories(id) ON DELETE SET NULL
)";
$db->conn->exec($query);
echo "Posts Table created successfully!";
```

- **category_id INT(11) DEFAULT NULL**: This line adds the **category_id** column to the table, which can hold integers. The **DEFAULT NULL** allows posts to be created without a category if necessary.

- **FOREIGN KEY (category_id) REFERENCES categories(id) ON DELETE SET NULL**: This line establishes a foreign key relationship with the **categories** table. If a category is deleted, the **category_id** in the **posts** table will be set to **NULL**.

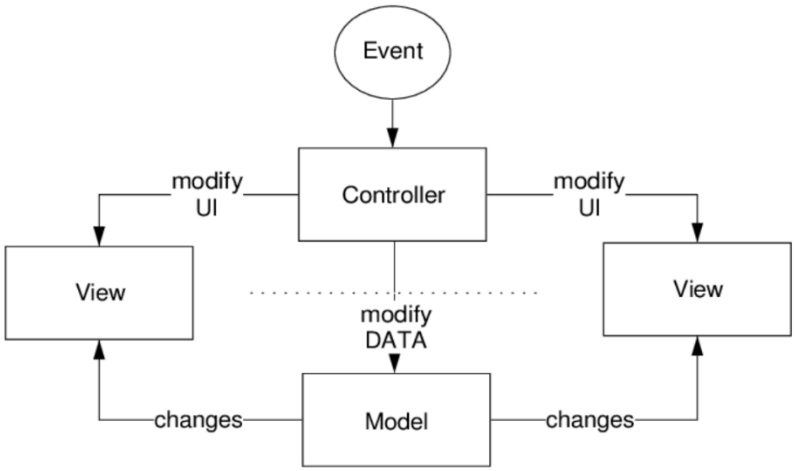
public/create_comments_table.php

```php
<?php

require __DIR__ . '/../bootstrap.php';

$db = new Database;

// Create comments table
$query = "CREATE TABLE IF NOT EXISTS comments (
    id INT AUTO_INCREMENT PRIMARY KEY,
    post_id INT NOT NULL,
    content TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (post_id) REFERENCES posts(id) ON DELETE CASCADE
)";
$db->conn->exec($query);
echo "Comments Table Created Successifully!";
```

public/create_users_table.php

```php
<?php

require __DIR__ . '/../bootstrap.php';

$db = new Database;

// Create users table
$query = "CREATE TABLE IF NOT EXISTS users (
    id INT(11) AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    is_admin TINYINT(1) NOT NULL DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)";

// Insert sample users
// $query = "INSERT INTO users (username, password, is_admin) VALUES
//     ('user1', '" . password_hash('password1', PASSWORD_DEFAULT) . "', 1),
//     ('user2', '" . password_hash('password2', PASSWORD_DEFAULT) . "', 0),
//     ('user3', '" . password_hash('password3', PASSWORD_DEFAULT) . "', 0)
// ";
$db->conn->exec($query);
echo "Users Table created successifully!";
```

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
```

```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
<?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…

a. **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.

b. **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.

c. **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.

d. **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.

e. **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.

f. **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)**.

g. **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.

Step 7: Implementing the Post Model

- It represents the data and the business logic related to blog posts in your application. It's responsible for fetching, inserting, updating, and deleting post data from the database.

Basic Structure:

- Create the **Post** model class in the **app/models** directory
- This class will contain methods that correspond to the various operations you can perform on a blog post.

app/models/Post.php

```
<?php

namespace app\models;

// Include the bootstrap file for global setup
require_once BASE_DIR . '/bootstrap.php';

// Define class for managing posts
class Post {
    private $db; // Database connection object

    // Initialize connection in constructor
    public function __construct() {
        $this->db = new \Database; // Create a Database object for connection
    }

   /**
     * Create a new post with title and content.
     *
     * @param string $title The title of the post.
     * @param string $content The content of the post.
     */
    public function createPost($title, $content) {
        // Prepare an SQL statement to insert a new post into the 'posts' table
        $stmt = $this->db->conn->prepare('INSERT INTO posts (title, content)
VALUES (?, ?)');

        // Execute the prepared statement with the provided title and content
        $stmt->execute([$title, $content]);
```

```
    }

    /**
     * Retrieve a single post by its ID.
     *
     * @param int $id The ID of the post to retrieve.
     * @return object The fetched post as an object.
     */
    public function getPost($id) {
        // Prepare an SQL statement to select a post from the 'posts' table by its
ID
        $stmt = $this->db->conn->prepare('SELECT posts.* FROM posts WHERE posts.id
= :id');

        // Execute the prepared statement with the provided ID
        $stmt->execute(['id' => $id]);

        // Return the fetched post as an object
        return $stmt->fetch(\PDO::FETCH_OBJ);
    }

    /**
     * Retrieve all posts, ordered by ID in descending order.
     *
     * @return array An array of post objects.
     */
    public function getPosts() {
        // Prepare an SQL statement to select all posts from the 'posts' table,
ordered by ID
        $stmt = $this->db->conn->prepare('SELECT posts.* FROM posts ORDER BY
posts.id DESC');

        // Execute the SQL statement
        $stmt->execute();

        // Return all fetched posts as an array of objects
        return $stmt->fetchAll(\PDO::FETCH_OBJ);
    }

    // Update an existing post with new title and content
    public function updatePost($id, $title, $content) {
        $sql = 'UPDATE posts SET title = :title, content = :content WHERE id =
:id';
        $stmt = $this->db->conn->prepare($sql);
        try {
            $stmt->execute(['title' => $title, 'content' => $content, 'id' =>
$id]);
        } catch (PDOException $e) {
            // Log the error
```

```
            error_log("Database error: " . $e->getMessage() . " in " . $e-
>getFile() . ":" . $e->getLine());

            // Optionally, display a user-friendly error message
            echo "<p>An error occurred while updating the post. Please try again
later.</p>";
        }
    }

    public function deletePost($id) {
        $stmt = $this->db->conn->prepare('DELETE FROM posts WHERE id = ?');
        $stmt->execute([$id]);
    }
}
```

Difference Between $db and $conn

In the context of your **Post** class, **$db** and **$conn** refer to different aspects of handling a database connection:

1. **$db**:
    - In your **Post** class, **$db** is a private property that holds an instance of the **Database** class.
    - When you create a new **Post** object, the constructor (**__construct()**) instantiates the **Database** class and assigns the resulting object to **$db**.
    - This property (**$db**) is used within the **Post** class to access the functionality provided by the **Database** class, such as establishing a connection to the database and executing SQL queries.
2. **$conn**
    - **$conn** typically refers to the actual database connection object within the **Database** class.
    - In a common implementation of a **Database** class, **$conn** would be a property that holds the PDO (PHP Data Objects) connection or another type of database connection resource.
    - This **$conn** property is usually used internally within the **Database** class to perform database operations like querying, fetching data, etc.

We will then create an ***AdminController.php*** which will contain methods for the crud functionality for our app.

app/controllers/AdminController.php

```
<?php

namespace app\controllers;

require_once BASE_DIR . 'app/models/Post.php';

class AdminController {
    private $model;

    public function __construct() {
        $this->model = new \app\models\Post;
```

```php
    }

    // Read: Display a list of posts
    public function index() {
        // Fetch all posts without pagination
        $posts = $this->model->getPosts();

        // Load the view for displaying posts
        require BASE_DIR . '/app/views/admin/posts/index.php';
    }

    public function create() {
        // Load the view for creating a new post
        require BASE_DIR . '/app/views/admin/posts/create.php';
    }


    // Read: Display a single post
    public function show($id) {
        // Fetch the post with the given ID
        $post = $this->model->getPost($id);

        if ($post === null) {
            // Post not found, display 404 page
            require BASE_DIR . '/app/views/404.php';
            return;
        }
        // Load the view for displaying the single post
        require BASE_DIR . '/app/views/admin/posts/show.php';
    }

    // Create: Store a new post
      public function store($postData) {

        // Retrieve form data
        $title = $postData['title'];
        $content = $postData['content'];
        $category_id = $postData['category_id'];
        try {
            $this->model->createPost($title, $content);
            // Redirect to the posts page after successful creation
            header('Location: /admin/posts');
            $_SESSION['message'] = 'Post created successfully!.';
            exit;
        } catch (Exception $e) {
            // If there's a database error, display it
            echo 'Error creating post: ' . $e->getMessage();
            exit;
        }
    }
```

```
    public function edit($id) {
        $post = $this->model->getPost($id);
        // Pass post and categories to the view
        require BASE_DIR . '/app/views/admin/posts/edit.php';
    }

    // Update: Update an existing post
    public function update($postData, $id) {
        // Retrieve form data
        $title = $postData['title'];
        $content = $postData['content'];
        try {
            // Call the updatePost method with the image filename
            $this->model->updatePost($id, $title, $content);

            // Redirect to the posts page after successful update
              $_SESSION['message'] = 'Post updated successfully!.';
              header('Location: /admin/posts');
              exit;

            exit;
        } catch (Exception $e) {
            // If there's a database error, display it
            echo 'Error updating post: ' . $e->getMessage();
            exit;
        }
    }
    // Delete: Remove a post
    public function delete($id) {
        $this->model->deletePost($id);
        // Redirect or display success message
        header('Location: /admin/posts');
    }
}
```

Create the View Files

We will now create the views files for our application based on what we have in the ***AdminController.php.***

'/app/views/admin/posts/index.php';

```
<div>
  <!-- Display message -->
    <?php if (isset($_SESSION['message'])): ?>
        <div id="message" class="<?php echo strpos($_SESSION['message'], 'error') !== 
false ? 'message-error' : 'message-success' ?> text-center">
            <?php
                echo $_SESSION['message'];
                unset($_SESSION['message']);
            ?>
```

```
        </div>
    <?php endif; ?>
      <div style="text-align: right;">
        <!-- Create Post Button -->
        <a href="/admin/posts/create">Create New Post</a>
          <a href="/admin/categories/create">Create Categories</a>

    </div>

    <?php foreach ($posts as $post): ?>
        <div style="margin-bottom: 20px; margin-top: 20px;">
            <div>
                <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
                <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
                <a href="/admin/posts/show/<?php echo $post->id; ?>">View</a>
                <a href="/admin/posts/edit/<?php echo $post->id; ?>">Edit</a>
                <!-- Delete Post -->
                <form action="/admin/posts/delete/<?php echo $post->id; ?>"
method="POST" style="display: inline;">
                    <input type="hidden" name="_method" value="DELETE">
                    <button type="submit">Delete</button>
                </form>
            </div>
        </div>
    <?php endforeach; ?>
</div>

<style>
.message-success {
color: green;
/* other styling */
}

.message-error {
    color: red;
    background-color: #ffd6d6;
    padding: 10px;
    border: 1px solid red;
    margin-bottom: 20px;
}
</style>
```

'/app/views/admin/posts/create.php';

```
<!-- admin/posts/create.php -->
<div>
```

```
    <!-- View all Post Button -->
    <div style="text-align: right; margin-bottom: 20px;">
        <a href="/admin/posts/">View Posts</a>
    </div>

    <h1>Create Post</h1>
    <form method="POST" action="/admin/posts/store">
        <div>
            <label for="title">Title:</label>
            <input type="text" id="title" name="title">
        </div>

        <div>
            <label for="content">Content:</label>
            <textarea id="content" name="content"></textarea>
        </div>

        <button type="submit">Create Post</button>
    </form>
</div>
```

'/app/views/admin/posts/show.php';

```
<div>
    <!-- View all Post Button -->
    <div style="text-align: right; margin-bottom: 20px;">
        <a href="/admin/posts/">View Posts</a>
    </div>
    <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
    <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
</div>
```

'/app/views/admin/posts/edit.php';

```
<div>
    <?php if ($post === null): ?>
        <div>Post not found</div>
        <?php return; ?>
        <?php endif; ?>
    <form method="POST" action="/admin/posts/update/<?php echo $post->id ?>"
style="margin-bottom: 20px;">
        <input type="hidden" name="id" value="<?php echo $post->id ?>">
        <div style="margin-bottom: 20px;">
            <label for="title">Title</label>
            <input type="text" id="title" name="title" value="<?php echo
htmlspecialchars($post->title, ENT_QUOTES); ?>">
        </div>
        <div style="margin-bottom: 20px;">
            <label for="content">Content</label>
```

```
            <textarea id="content" name="content"><?php echo htmlspecialchars($post-
>content, ENT_QUOTES); ?></textarea>
        </div>
        <button type="submit">Update Post</button>
    </form>
</div>
```

UpdateThe Routes

```
$routes = [
    'GET' => [
         // Frontend Routes
        '' => ['controller' => '\app\controllers\PostController', 'method' =>
'index'],
        'posts/show/([0-9]+)' => ['controller' => 'app\controllers\PostController',
'method' => 'show'],

        // Admin Posts Routes
        'admin/posts' => ['controller' => '\app\controllers\AdminController',
'method' => 'index'],
        'admin/posts/create' => ['controller' => '\app\controllers\AdminController',
'method' => 'create'],
        'admin/posts/show/([0-9]+)' => ['controller' =>
'\app\controllers\AdminController', 'method' => 'show'],
        'admin/posts/edit/([0-9]+)' => ['controller' =>
'\app\controllers\AdminController', 'method' => 'edit'],

   'POST' => [
 // Admin Post Routes
        'admin/posts/store' => ['controller' => '\app\controllers\AdminController',
'method' => 'store'],
        'admin/posts/update/([0-9]+)' => ['controller' =>
'\app\controllers\AdminController', 'method' => 'update'],
        'admin/posts/delete/([0-9]+)' => ['controller' =>
'\app\controllers\AdminController', 'method' => 'delete'],
    ]
];
```

Step 8: Implementing the Category Model (app/models/Category.php)

- It represents the data and the business logic related to blog categories in your application. It's responsible for fetching, inserting, updating, and deleting categories data from the database.

```
<?php

namespace app\models;

// Include the bootstrap file for global setup
require_once BASE_DIR . '/bootstrap.php';
```

```php
class Category {
    private $db;

    // Initialize connection in constructor
    public function __construct() {
        $this->db = new \Database; // Create a Database object for connection
    }

    public function getCategory($id) {
        // Prepare a SQL statement
        $stmt = $this->db->conn->prepare("SELECT * FROM categories WHERE id = ?");
        // Execute the statement with the ID
        $stmt->execute([$id]);
        // Fetch the category and return it
        return $stmt->fetch(\PDO::FETCH_OBJ);
    }

    public function getCategories() {
        // Fetch all categories from the database
        $stmt = $this->db->conn->query("SELECT * FROM categories");
        return $stmt->fetchAll(\PDO::FETCH_OBJ);
    }

    public function create($data) {
        // Insert a new category into the database
        $stmt = $this->db->conn->prepare("INSERT INTO categories (name) VALUES
(:name)");
        $stmt->execute([':name' => $data['name']]);
    }

    public function updateCategory($data, $id) {
        $stmt = $this->db->conn->prepare("UPDATE categories SET name = :name WHERE
id = :id");
        $stmt->execute([':name' => $data['name'], ':id' => $id]);
    }

    public function getPosts($categoryId) {
        // Get the posts in a specific category
        $stmt = $this->db->conn->prepare("SELECT * FROM posts WHERE category_id =
?");
        $stmt->execute([$categoryId]);
        return $stmt->fetchAll(\PDO::FETCH_OBJ);
    }

    public function deleteCategory($id) {
        // Delete a category from the database
        $stmt = $this->db->conn->prepare("DELETE FROM categories WHERE id = :id");
        $stmt->execute([':id' => $id]);
    }
}
```

app/controllers/CategoryController.php

Next is to update the category controller with different methods such as index, show, update and delete which are going to handle the CRUD functionality for our categories.

```php
<?php

namespace app\controllers;

require_once BASE_DIR . '/app/models/Category.php';

class CategoryController {
    private $model;

    public function __construct() {
        $this->model = new \app\models\Category;
    }

    public function index() {
        // Fetch all categories
        $categories = $this->model->getCategories();
        require BASE_DIR . '/app/views/admin/categories/index.php';
    }

    public function create() {
        // Display the category creation form
        require BASE_DIR . '/app/views/admin/categories/create.php';
    }

    public function showPosts($categoryId) {
        $category = $this->model->getCategory($categoryId);
        if (!$category) {
            die('Category not found');
        }

        $posts = $this->model->getPosts($categoryId);
        // A view that lists posts in this category
        require BASE_DIR . '/app/views/admin/categories/posts.php';
    }

    public function store($data) {
        // Create a new category
        $this->model->create($data);
        header('Location: /admin/categories');
    }

    public function edit($id) {
        $category = $this->model->getCategory($id);
        if (!$category) {
```

```
            die('Category not found');
        }
        require BASE_DIR . '/app/views/admin/categories/edit.php';
    }

    public function update($id, $data) {
        $this->model->updateCategory($id, $data);
        header('Location: /admin/categories');
    }

    public function delete($id) {
      $this->model->deleteCategory($id);
      header('Location: /admin/categories');
    }
}
?>
```

Create The View Files

Views/admin/categories/create.php

```
<div>
    <!-- View Category Button -->
    <div style="text-align: right; margin-bottom: 20px;">
        <a href="/admin/categories/">View Categories</a>
    </div>
    <form method="POST" action="/admin/categories/store" style="margin-bottom:
20px;">
        <div style="margin-bottom: 20px;">
            <label for="name">Category Name</label>
            <input type="text" id="name" name="name">
        </div>
        <button type="submit">Create Category</button>
    </form>
</div>
```

Views/admin/categories/index.php

```
<div>
    <div style="text-align: right; margin-bottom: 20px;">
        <!-- Create Category Button -->
        <a href="/admin/categories/create">Create Category</a>
        <!-- View Posts Button -->
        <a href="/admin/posts/">View Posts</a>
    </div>

    <?php foreach ($categories as $category): ?>
        <div style="margin-bottom: 20px; margin-top: 20px;">
            <div>
                <h2>Categories</h2>
```

```
                <p><?php echo htmlspecialchars($category->name, ENT_QUOTES); ?></p>
                <a href="/admin/categories/edit/<?php echo $category->id;
?>">Edit</a>
                <a href="/admin/categories/posts/<?php echo $category->id; ?>">View
Posts</a>
                <form action="/admin/categories/delete/<?php echo $category->id; ?>"
method="POST" style="display: inline;">
                    <input type="hidden" name="_method" value="DELETE">
                    <button type="submit">Delete</button>
                </form>
            </div>
        </div>
    <?php endforeach; ?>
</div>
```

Views/admin/categories/edit.php

```
<div>
    <form method="POST" action="/admin/categories/update/<?php echo $category->id;
?>" style="margin-bottom: 20px;">
        <input type="hidden" name="id" value="<?php echo $category->id; ?>">
        <div style="margin-bottom: 20px;">
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" value="<?php echo
htmlspecialchars($category->name, ENT_QUOTES); ?>">
        </div>
        <!-- Add more input fields for other category properties... -->
        <button type="submit">Update</button>
    </form>
</div>
```

Views/admin/categories/posts.php

```
<!-- admin/categories/posts.php -->
<div>
    <h2>Posts in Category: <?php echo htmlspecialchars($category->name, ENT_QUOTES);
?></h2>
    <?php if (!empty($posts)): ?>
        <ul>
            <?php foreach ($posts as $post): ?>
                <li>
                    <h3><?php echo htmlspecialchars($post->title, ENT_QUOTES);
?></h3>
                    <p><?php echo htmlspecialchars($post->content, ENT_QUOTES);
?></p>
                    <!-- Add links for edit, delete, view, etc. here -->
                </li>
            <?php endforeach; ?>
        </ul>
    <?php else: ?>
```

```
        <p>No posts found in this category.</p>
    <?php endif; ?>
</div>
```

Add the Routes

public/Index.php

```
$routes = [
    'GET' => [
         // Admin categories Routes

        'admin/categories/create' => ['controller' =>
'\app\controllers\CategoryController', 'method' => 'create'],
        'admin/categories' => ['controller' => '\app\controllers\CategoryController',
'method' => 'index'],
        'admin/categories/edit/([0-9]+)' => ['controller' =>
'\app\controllers\CategoryController', 'method' => 'edit'],
        'admin/categories/posts/([0-9]+)' => ['controller' =>
'\app\controllers\CategoryController', 'method' => 'showPosts']
    ],

'POST' => [
 // Admin categories Routes
        'admin/categories/store' => ['controller' =>
'\app\controllers\CategoryController', 'method' => 'store'],
        'admin/categories/update/([0-9]+)' => ['controller' =>
'\app\controllers\CategoryController', 'method' => 'update'],
        'admin/categories/delete/([0-9]+)' => ['controller' =>
'\app\controllers\CategoryController', 'method' => 'delete'],
    ]
];
```

Step 9: Associate Posts With Categories

In this section we will attach the categories we've created to our posts. Let's go ahead and do that.

Modify Post.php

```
<?php

namespace app\models;

// Include the bootstrap file for global setup
require_once BASE_DIR . '/bootstrap.php';

// Define class for managing posts
class Post {
    private $db; // Database connection object

    // Initialize connection in constructor
```

```php
    public function __construct() {
        $this->db = new \Database; // Create a Database object for connection
    }

    /**
     * Create a new post with title and content.
     *
     * @param string $title The title of the post.
     * @param string $content The content of the post.
     */
    public function createPost($title, $content, $category_id) {
        // Prepare an SQL statement to insert a new post into the 'posts' table
        $stmt = $this->db->conn->prepare('INSERT INTO posts (title, content, category_id) VALUES (?, ?, ?)');

        // Execute the prepared statement with the provided title and content
        $stmt->execute([$title, $content, $category_id]);
    }

    /**
     * Retrieve a single post by its ID.
     *
     * @param int $id The ID of the post to retrieve.
     * @return object The fetched post as an object.
     */
    public function getPost($id) {
        // Prepare an SQL statement to select a post from the 'posts' table by its ID
        $stmt = $this->db->conn->prepare('SELECT posts.*, categories.name as category_name FROM posts LEFT JOIN categories ON posts.category_id = categories.id WHERE posts.id = :id');

        // Execute the SQL statement with the provided post ID
        $stmt->execute(['id' => $id]);

        // Return the fetched post as an object
        return $stmt->fetch(\PDO::FETCH_OBJ);
    }

    /**
     * Retrieve all posts, ordered by ID in descending order.
     *
     * @return array An array of post objects.
     */
    public function getPosts() {
        // Prepare an SQL statement to select all posts from the 'posts' table, ordered by ID
        $stmt = $this->db->conn->prepare('SELECT posts.*, categories.name as category_name FROM posts LEFT JOIN categories ON posts.category_id = categories.id ORDER BY posts.id DESC LIMIT 5');
```

```
        // Execute the SQL statement
        $stmt->execute();

        // Return all fetched posts as an array of objects
        return $stmt->fetchAll(\PDO::FETCH_OBJ);
    }

    // Update an existing post with new title, content and category_id
    public function updatePost($id, $title, $content, $category_id) {
        $sql = 'UPDATE posts SET title = :title, content = :content, category_id =
:category_id WHERE id = :id';
        $stmt = $this->db->conn->prepare($sql);
        try {
            $stmt->execute(['title' => $title, 'content' => $content, 'category_id'
=> $category_id, 'id' => $id]);
        } catch (PDOException $e) {
            // Log the error
            error_log("Database error: " . $e->getMessage() . " in " . $e->getFile()
. ":" . $e->getLine());

            // Optionally, display a user-friendly error message
            echo "<p>An error occurred while updating the post. Please try again
later.</p>";
        }
    }

    public function deletePost($id) {
        $stmt = $this->db->conn->prepare('DELETE FROM posts WHERE id = ?');
        $stmt->execute([$id]);
    }
}
```

Modify the *app/controllers/AdminController.php*

```
<?php

namespace app\controllers;

require_once BASE_DIR . '/app/models/Post.php';

class AdminController {
    private $model;

    public function __construct() {
        $this->model = new \app\models\Post;
    }

    // Read: Display a list of posts
    public function index() {
        // Fetch all posts without pagination
```

```
        $posts = $this->model->getPosts();

        // Load the view for displaying posts
        require BASE_DIR . '/app/views/admin/posts/index.php';
    }

    public function create() {
        // Create an instance of the Category model
        $categoryModel = new \app\models\Category;

        // Fetch all categories
        $categories = $categoryModel->getCategories();
        // Load the view for creating a new post
        require BASE_DIR . '/app/views/admin/posts/create.php';
    }


    // Read: Display a single post
    public function show($id) {
        // Fetch the post with the given ID
        $post = $this->model->getPost($id);

        if ($post === null) {
            // Post not found, display 404 page
            require BASE_DIR . '/app/views/404.php';
            return;
        }
        // Load the view for displaying the single post
        require BASE_DIR . '/app/views/admin/posts/show.php';
    }

    // Create: Store a new post
    public function store($postData) {
        // Retrieve form data
       $title = $postData['title'];
        $content = $postData['content'];
        $category_id = $postData['category_id'];
        try {
            $this->model->createPost($title, $content, $category_id,);
            // Redirect to the posts page after successful creation
            header('Location: /admin/posts');
            $_SESSION['message'] = 'Post created successfully!.';
            exit;
        } catch (Exception $e) {
            // If there's a database error, display it
            echo 'Error creating post: ' . $e->getMessage();
            exit;
        }
    }
```

```
    public function edit($id) {
        $post = $this->model->getPost($id);
        // Fetch categories
        $categoryModel = new \app\models\Category;
        $categories = $categoryModel->getCategories();
        // Pass post and categories to the view
        require BASE_DIR . '/app/views/admin/posts/edit.php';
    }

    // Update: Update an existing post
    public function update($postData, $id) {
        $title = $postData['title'];
        $content = $postData['content'];
        $category_id = $postData['category_id'];
        try {
            // Call the updatePost method
            $this->model->updatePost($id, $title, $content, $category_id);

            // Redirect to the posts page after successful update
              $_SESSION['message'] = 'Post updated successfully!.';
              header('Location: /admin/posts');
              exit;
        } catch (Exception $e) {
            // If there's a database error, display it
            echo 'Error updating post: ' . $e->getMessage();
            exit;
        }
    }

    // Delete: Remove a post
    public function delete($id) {
        $this->model->deletePost($id);
        // Redirect or display success message
        header('Location: /admin/posts');
    }
}
```

Modify The Following View Files

App/views/admin/posts/create

```
<!-- admin/posts/create.php -->
<div>
    <!-- View all Post Button -->
    <div style="text-align: right; margin-bottom: 20px;">
        <a href="/admin/posts/">View Posts</a>
    </div>

    <h1>Create Post</h1>
    <form method="POST" action="/admin/posts/store">
```

```
        <div>
            <label for="title">Title:</label>
            <input type="text" id="title" name="title">
        </div>

        <div>
            <label for="content">Content:</label>
            <textarea id="content" name="content"></textarea>
        </div>

        <!-- Populate Categories in this view -->
        <div>
            <label for="category_id">Category:</label>
            <select id="category_id" name="category_id">
                <?php foreach ($categories as $category): ?>
                    <option value="<?php echo $category->id; ?>"><?php echo
$category->name; ?></option>
                <?php endforeach; ?>
            </select>
        </div>


        <button type="submit">Create Post</button>
    </form>
</div>
```

App/views/admin/posts/edit

```
<div>
    <?php if ($post === null): ?>
        <div>Post not found</div>
        <?php return; ?>
        <?php endif; ?>
    <form method="POST" action="/admin/posts/update/<?php echo $post->id ?>"
style="margin-bottom: 20px;">
        <input type="hidden" name="id" value="<?php echo $post->id ?>">
        <div style="margin-bottom: 20px;">
            <label for="title">Title</label>
            <input type="text" id="title" name="title" value="<?php echo
htmlspecialchars($post->title, ENT_QUOTES); ?>">
        </div>
        <div style="margin-bottom: 20px;">
            <label for="content">Content</label>
            <textarea id="content" name="content"><?php echo htmlspecialchars($post-
>content, ENT_QUOTES); ?></textarea>
        </div>

        <!-- Populate Categories in this view -->
        <div>
            <label for="category_id">Category:</label>
```

```
            <select id="category_id" name="category_id">
            <?php foreach ($categories as $category): ?>
            <option value= "<?php echo $category->id; ?>" <?php echo $category->id ==
$post->category_id ? 'selected' : ''; ?>>
            <?php echo $category->name; ?>
            </option>
            <?php endforeach; ?>
            </select>
        </div>
        <button type="submit">Update Post</button>
    </form>
</div>
```

<option value="<?php echo $category->id; ?>":

- This creates an option in the dropdown list.
- **<?php echo $category->id; ?>** outputs the ID of the category, which is set as the value of the option. When this option is selected and the form is submitted, this ID is sent as part of the form data.

<?php echo $category->id == $post->category_id ? 'selected' : ''; ?>:

- This is a ternary conditional statement in PHP. It checks if the current category's ID (**$category->id**) matches the category ID of the post (**$post->category_id**).
- If the IDs match, it outputs **selected**, making this option the default selected option in the dropdown. This is typically used to show the current category of a post when editing it.
- If the IDs do not match, it outputs an empty string, meaning this option will not be selected by default.

app/views/admin/posts/index

```
<div>
    <!-- Display message -->
    <?php if (isset($_SESSION['message'])): ?>
        <div id="message" class="<?php echo strpos($_SESSION['message'], 'error') !==
false ? 'message-error' : 'message-success' ?> text-center">
            <?php
                echo $_SESSION['message'];
                unset($_SESSION['message']);
            ?>
        </div>
    <?php endif; ?>
      <div style="text-align: right;">
        <!-- Create Post Button -->
        <a href="/admin/posts/create">Create New Post</a>
        <a href="/admin/categories/create">Create Categories</a>
    </div>

    <?php foreach ($posts as $post): ?>
```

```
        <div style="margin-bottom: 20px; margin-top: 20px;">
            <div>
                <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
                <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
                <a href="/admin/posts/show/<?php echo $post->id; ?>">View</a>
                <a href="/admin/posts/edit/<?php echo $post->id; ?>">Edit</a>
                <p><strong>Category:</strong> <?php echo $post->category_name ?
htmlspecialchars($post->category_name, ENT_QUOTES) : 'No category'; ?></p>
                <!-- Delete Post -->
                <form action="/admin/posts/delete/<?php echo $post->id; ?>"
method="POST" style="display: inline;">
                    <input type="hidden" name="_method" value="DELETE">
                    <button type="submit">Delete</button>
                </form>
            </div>
        </div>
    <?php endforeach; ?>
</div>

<style>
.message-success {
color: green;
/* other styling */
}

.message-error {
    color: red;
    background-color: #ffd6d6;
    padding: 10px;
    border: 1px solid red;
    margin-bottom: 20px;
}
</style>
```

Step 10: Implementing the Comment Model (*app/models/Comment.php*)

We will start by adding code to the comment model. Let's go ahead and do that.

Comment.php Model

```
<?php

namespace app\models;

// Include the bootstrap file for global setup
require_once BASE_DIR . '/bootstrap.php';


class Comment {
    private $db;
```

```
    // Initialize connection in constructor
    public function __construct() {
        $this->db = new \Database; // Create a Database object for connection
    }

    public function createComment($postId, $content) {
        // Prepare a SQL statement
        $stmt = $this->db->conn->prepare("INSERT INTO comments (post_id, content)
VALUES (?, ?)");
        // Execute the statement with the post ID and the content
        $stmt->execute([$postId, $content]);
    }

    public function getComments($postId) {
        // Prepare a SQL statement
        $stmt = $this->db->conn->prepare("SELECT * FROM comments WHERE post_id = ?");
        // Execute the statement with the post ID
        $stmt->execute([$postId]);
        // Return the result as an object
        return $stmt->fetchAll(\PDO::FETCH_OBJ);
    }

    public function getCommentById($id) {
        // Prepare a SQL statement
        $stmt = $this->db->conn->prepare("SELECT * FROM comments WHERE id = ?");
        // Execute the statement with the ID
        $stmt->execute([$id]);
        // Return the result as an object
        return $stmt->fetch(\PDO::FETCH_OBJ);
    }

    public function updateComment($id, $data) {
        // Prepare a SQL statement
        $stmt = $this->db->conn->prepare("UPDATE comments SET content = ? WHERE id =
?");
        // Execute the statement with the new content and the ID
        return $stmt->execute([$data['content'], $id]); // Return the result
    }

    public function deleteComment($id) {
        // Get the comment
        $comment = $this->getCommentById($id);
        // Check if the comment exists
        if ($comment) {
            // Prepare a SQL statement
            $stmt = $this->db->conn->prepare("DELETE FROM comments WHERE id = ?");
            // Execute the statement with the ID
            $stmt->execute([$id]);
        }
    }
```

```
}
?>
```

app/controllers/CommentController.php

Let's update the ***app/controllers/CommentController.php***.

```
<?php

namespace app\controllers;

require_once BASE_DIR . '/app/models/Comment.php';

class CommentController {
    private $model;

    public function __construct() {
        $this->model = new \app\models\Comment;
    }

    public function store($data) {
        $postId = $data['post_id'];
        $content = $data['content'];
        // Store the comment in the database
        $this->model->createComment($postId, $content);
        header('Location: /posts/show/' . $postId);
    }

    public function edit($id) {
        // Get the comment from the database
        $comment = $this->model->getCommentById($id);

        // Check if the comment exists
        if (!$comment) {
            die('Comment not found');
        }

        // Create an instance of the Post model
        $postModel = new \app\models\Post;

        // Fetch the post related to the comment
        $post = $postModel->getPost($comment->post_id);

        // Check if the post exists
        if (!$post) {
            die('Post not found');
        }

        // Include the edit form view, pass both comment and post
        require BASE_DIR . '/app/views/admin/comments/edit.php';
```

```
    }

    public function update($data, $id) {
        $data = $_POST;
        $result = $this->model->updateComment($id, $data); // Update the comment
        if (isset($data['post_id']) && !empty($data['post_id'])) {
            $postId = $data['post_id'];
            $_SESSION['message'] = 'Comment updated successfully!.';
            header('Location: /admin/posts/show/' . $postId);
        } else {
            // Handle the error or redirect to a default page
            header('Location: /admin/posts');
        }
    }

 public function delete($id) {
      // Delete the comment
     $this->model->deleteComment($id);
     // Redirect to the related post or a default page if successful
          header('Location: /admin/posts');
       }
    }
}
```

Now were still not able to create a comment for a post. This is because I decided that there is no need for an admin to do that. We will allow the comment to be created on the frontend.

So, what we will do here is that we will update the view file ***admin/posts/show.php***. It will loop over the comments created by a user on the frontend and the admin will be able to either edit or delete the comment. This is what I mean.

admin/posts/show.php

```
<div>
    <!-- View all Post Button -->
    <div style="text-align: right; margin-bottom: 20px;">
        <a href="/admin/posts/">View Posts</a>
    </div>
    <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
    <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
    <?php if (!empty($comments)): ?>
    <h3>Comments</h3>
    <?php foreach ($comments as $comment): ?>
        <div class="mb-2">
            <p><?php echo htmlspecialchars($comment->content, ENT_QUOTES); ?></p>
            <a href="/admin/comments/edit/<?php echo $comment->id; ?>" class="btn
btn-warning btn-sm">Edit</a>
            <form action="/admin/comments/delete/<?php echo $comment->id; ?>"
method="POST" style="display: inline;">
                <input type="hidden" name="_method" value="DELETE">
```

```
                <button type="submit">Delete</button>
                </form>
        </div>
    <?php endforeach; ?>
    <?php else: ?>
        <p>No comments found in this post.</p>
    <?php endif; ?>
</div>
```

AdminController: Show Method

Then Update the ***show method*** within the AdminController like this.

```
require_once BASE_DIR . '/app/models/Comment.php';
// Read: Display a single post
    public function show($id) {
        // Fetch the post with the given ID
        $post = $this->model->getPost($id);

        if ($post === null) {
            // Post not found, display 404 page
            require BASE_DIR . '/app/views/404.php';
            return;
        }
         // Create an instance of the Comment model
         $commentModel = new \app\models\Comment;

         // Fetch the comments for the post
         $comments = $commentModel->getComments($id);
        // Load the view for displaying the single post
        require BASE_DIR . '/app/views/admin/posts/show.php';
    }
```

So, what is happening here is that we are creating an object ***$commentModel.*** This object is used to access the method ***getComments*** where we pass an ***$id*** of the comment related to the post then we define a ***$comments*** variable to store the returned data.

So far no comment will be returned. As we can inspect this by passing the ***<?php if (!empty($comments)): ?>*** code.

Update the Routes File (*public/index.php*)

Since we have the end points ***/admin/comments/edit/*** and ***'admin/comments/update/*** that enables us to edit and update a comment from the admin side. Let's create the routes to the controller methods edit and update in the comment controller.

```
$routes = [
    'GET' => [
 // Admin Comment Routes
```

```
        'admin/comments/edit/([0-9]+)' => ['controller' =>
'\app\controllers\CommentController', 'method' => 'edit'],
],
'POST' => [
// Admin Comment Routes
         'admin/comments/update/([0-9]+)' => ['controller' =>
'\app\controllers\CommentController', 'method' => 'update'],
 'admin/comments/delete/([0-9]+)' => ['controller' =>
'\app\controllers\CommentController', 'method' => 'delete'],
      ]
];
```

Lastly let's create the view file (***admin/comments/edit.php***)

```
<?php
if (!$comment) {
    echo "Comment not found";
    return;
}
?>
<form method="POST" action="/admin/comments/update/<?php echo $comment->id; ?>">
    <input type="hidden" name="id" value="<?php echo $comment->id; ?>">
    <input type="hidden" name="post_id" value="<?php echo $comment->post_id; ?>">

    <label for="content">Comment Content</label>
    <textarea id="content" name="content"><?php echo htmlspecialchars($comment-
>content, ENT_QUOTES); ?></textarea>

    <button type="submit">Update Comment</button>
</form>
```

Updating The PostController

To ensure users can add a comment from the frontend side let's go ahead and update the PostController we created earlier. **Remember its only the admin who will edit and update a comment.**

app/controllers/PostController.php

```
<?php

namespace app\controllers;

require_once BASE_DIR . '/app/models/Post.php';
require_once BASE_DIR . '/app/models/Comment.php';

class PostController {
    private $model;

    public function __construct() {
```

```
        $this->model = new \app\models\Post;
    }

    // Display all posts
    public function index() {
        $posts = $this->model->getPosts();
        require BASE_DIR . '/app/views/posts/index.php';
    }

    // Display a single post
    public function show($id) {
        // Retrieve the specific post by its ID using the Post model
        $post = $this->model->getPost($id);

        // Instantiate the Comment model to interact with the comments data
        $commentModel = new \app\models\Comment;

        // Retrieve all comments associated with the post
        // This assumes that the getComments method fetches comments based on the
post's ID
        $comments = $commentModel->getComments($id);

        // Load the view file that displays the details of the post
        // This view will show the post and its associated comments
        require BASE_DIR . '/app/views/posts/show.php';
    }
}
?>
```

Create the Frontend view Files (Index & Show)

This view files will be involved in displaying the posts in the frontend side. Let's start by updating the index.php view file.

app/views/posts/index.php

```
<?php foreach ($posts as $post): ?>
        <div style="margin-bottom: 20px; margin-top: 20px;">
            <div>
                <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
                <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
                <div>
                  <a href="/posts/show/<?php echo $post->id; ?>">View</a>
                </div>
            </div>
        </div>
    <?php endforeach; ?>
</div>
```

Here we loop over our posts from the database then we print them out using object-property access ***$post->title*** and ***$post->content.***

app/views/posts/show.php

```
<!-- show.php -->
<div>
    <div>
        <div>
            <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
        </div>

        <div>
            <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
        </div>

        <div>
            <h3>Comments</h3>
            <?php foreach ($comments as $comment): ?>
                <div style="margin-bottom: 20px;">
                    <p><?php echo htmlspecialchars($comment->content, ENT_QUOTES);
?></p>
                </div>
            <?php endforeach; ?>

            <form action="/comments/store" method="post" style="margin-top: 20px;">
                <div style="margin-bottom: 20px;">
                    <label for="content">Content:</label>
                    <textarea id="content" name="content"></textarea>
                </div>
                <input type="hidden" name="post_id" value="<?php echo $post->id; ?>">
                <button type="submit">Submit</button>
            </form>
        </div>
    </div>
</div>
```

We loop over the comments in the database and show them in our view file ***show.php***. When adding a new comment. We route to the ***"/comments/store"*.** This endpoint will call the Commentcontroller method store.

```
 'comments/store' => ['controller' => '\app\controllers\CommentController', 'method'
=> 'store'],
```

Which will store our comments in the database table comments. This is the method that does that.

```
    public function store($data) {
        $postId = $data['post_id'];
        $content = $data['content'];
        // Store the comment in the database
        $this->model->createComment($postId, $content);
        header('Location: /posts/show/' . $postId);
```

```
    }
```

It captures the post id of the post we are currently commenting and the content of the comment then routes us back to the show view file.

Editing and Deleting Comments on the Admin Side.

We can now edit and delete comments on the admin side. This is because a user can now comment from the frontend side including the admin. So, the admin can see the comments and do any modification to them.

Step 11: Implement the File Upload

In this section we will handle file upload. This is where we will attach images to our posts in the database and display the posts with images in the view files. Let's get started.

Update the Post.php Model

```
<?php

namespace app\models;

// Include the bootstrap file for global setup
require_once BASE_DIR . '/bootstrap.php';

// Define class for managing posts
class Post {
    private $db; // Database connection object

    // Initialize connection in constructor
    public function __construct() {
        $this->db = new \Database; // Create a Database object for connection
    }

   /**
     * Create a new post with title and content.
     *
     * @param string $title The title of the post.
     * @param string $content The content of the post.
     */
    public function createPost($title, $content, $category_id, $imageFilename = null)
{
      // Prepare an SQL statement to insert a new post into the 'posts' table
      $stmt = $this->db->conn->prepare('INSERT INTO posts (title, content,
category_id, image) VALUES (?, ?, ?, ?)');

       // Bind title, content, category, and image to prepared statement parameters
       $stmt->execute([$title, $content, $category_id, $imageFilename]);
    }

    /**
     * Retrieve a single post by its ID.
     *
```

```
     * @param int $id The ID of the post to retrieve.
     * @return object The fetched post as an object.
     */
    public function getPost($id) {
        // Prepare an SQL statement to select a post from the 'posts' table by its ID
        $stmt = $this->db->conn->prepare('SELECT posts.*, categories.name as
category_name, posts.image as image FROM posts LEFT JOIN categories ON
posts.category_id = categories.id WHERE posts.id = :id');

        // Execute the SQL statement with the provided post ID
        $stmt->execute(['id' => $id]);

        // Return the fetched post as an object
        return $stmt->fetch(\PDO::FETCH_OBJ);
    }

    /**
     * Retrieve all posts, ordered by ID in descending order.
     *
     * @return array An array of post objects.
     */
    public function getPosts() {
        // Prepare an SQL statement to select all posts from the 'posts' table,
ordered by ID
        $stmt = $this->db->conn->prepare('SELECT posts.*, categories.name as
category_name, posts.image as image FROM posts LEFT JOIN categories ON
posts.category_id = categories.id ORDER BY posts.id DESC LIMIT 5');

        // Execute the SQL statement
        $stmt->execute();

        // Return all fetched posts as an array of objects
        return $stmt->fetchAll(\PDO::FETCH_OBJ);
    }

    // Update an existing post with new title, content and category_id
    public function updatePost($id, $title, $content, $category_id, $imageFilename =
null) {
        $sql = 'UPDATE posts SET title = :title, content = :content, category_id =
:category_id, image = :image WHERE id = :id';
        $stmt = $this->db->conn->prepare($sql);
        try {
            $stmt->execute(['title' => $title, 'content' => $content, 'category_id'
=> $category_id,'image' => $imageFilename, 'id' => $id]);
        } catch (PDOException $e) {
            // Log the error
            error_log("Database error: " . $e->getMessage() . " in " . $e->getFile()
. ":" . $e->getLine());

            // Optionally, display a user-friendly error message
```

```
            echo "<p>An error occurred while updating the post. Please try again
later.</p>";
        }
    }

    public function deletePost($id) {
        $stmt = $this->db->conn->prepare('DELETE FROM posts WHERE id = ?');
        $stmt->execute([$id]);
    }
}
```

We set the image filename to null when no image has been uploaded.

Update the AdminController.php

```
<?php

namespace app\controllers;

require_once BASE_DIR . '/app/models/Post.php';
require_once BASE_DIR . '/app/models/Comment.php';

class AdminController {
    private $model;

    public function __construct() {
        $this->model = new \app\models\Post;
    }

    // Read: Display a list of posts
    public function index() {
        // Fetch all posts without pagination
        $posts = $this->model->getPosts();

        // Load the view for displaying posts
        require BASE_DIR . '/app/views/admin/posts/index.php';
    }

    public function create() {
        // Create an instance of the Category model
        $categoryModel = new \app\models\Category;

        // Fetch all categories
        $categories = $categoryModel->getCategories();
        // Load the view for creating a new post
        require BASE_DIR . '/app/views/admin/posts/create.php';
    }


    // Read: Display a single post
```

```php
    public function show($id) {
        // Fetch the post with the given ID
        $post = $this->model->getPost($id);

        if ($post === null) {
            // Post not found, display 404 page
            require BASE_DIR . '/app/views/404.php';
            return;
        }
         // Create an instance of the Comment model
         $commentModel = new \app\models\Comment;

         // Fetch the comments for the post
         $comments = $commentModel->getComments($id);
        // Load the view for displaying the single post
        require BASE_DIR . '/app/views/admin/posts/show.php';
    }

    // Create: Store a new post
    public function store($postData) {
        // Retrieve form data
        $title = $postData['title'];
        $content = $postData['content'];
        $category_id = $postData['category_id'];
        $imageFilename = null;  // Default to null if no image is uploaded

        // Check if an image file is uploaded and there are no upload errors
        if (isset($_FILES['image']) && $_FILES['image']['error'] === UPLOAD_ERR_OK) {
            // Define allowed file types and size
            $allowedTypes = ['image/jpeg', 'image/png', 'image/gif'];
            $maxSize = 5 * 1024 * 1024; // 5 MB

            // Extract file information from the uploaded image
            $fileTmpPath = $_FILES['image']['tmp_name']; // Temporary path where the
uploaded file is stored
            $fileName = $_FILES['image']['name'];       // Original name of the
uploaded file
            $fileType = $_FILES['image']['type'];       // MIME type of the uploaded
file
            $fileSize = $_FILES['image']['size'];       // Size of the uploaded file
in bytes


            // Check file type and size
            if (in_array($fileType, $allowedTypes) && $fileSize <= $maxSize) {
                // Create a unique file name to avoid overwriting existing files
                $newFileName = uniqid() . '-' . $fileName;

                // Set the upload path for the image
```

```
                $uploadPath = 'S:/xampp/htdocs/php8blogapp/public/images/'; // Define
the directory where uploaded images will be stored
                $destination = $uploadPath . $newFileName; // Concatenate the upload
path with the new file name to create the full destination path

                // Ensure the upload directory exists
                if (!file_exists($uploadPath)) {
                    // Create the directory if it does not exist, with full
read/write/execute permissions (0777) and enable recursion (true)
                    mkdir($uploadPath, 0777, true);
                }

                // Move the uploaded file
                if (move_uploaded_file($fileTmpPath, $destination)) {
                    // File uploaded successfully
                    $imageFilename = $newFileName; // Set this for database storage
                } else {
                    // Handle error (file move failed)
                    $errors[] = 'Failed to upload image.';
                }
            } else {
                // Handle invalid file type or size
                $errors[] = 'Invalid file type or size.';
            }
        } else {
            // Handle no file uploaded or upload error
            if ($_FILES['image']['error'] != UPLOAD_ERR_NO_FILE) {
                $errors[] = 'Error in file upload.';
            }
        }

        try {
            $this->model->createPost($title, $content, $category_id, $imageFilename);
            // Redirect to the posts page after successful creation
            header('Location: /admin/posts');
            $_SESSION['message'] = 'Post created successfully!.';
            exit;
        } catch (Exception $e) {
            // If there's a database error, display it
            echo 'Error creating post: ' . $e->getMessage();
            exit;
        }
    }

    public function edit($id) {
        $post = $this->model->getPost($id);
        // Fetch categories
        $categoryModel = new \app\models\Category;
        $categories = $categoryModel->getCategories();
        // Pass post and categories to the view
```

```
        require BASE_DIR . '/app/views/admin/posts/edit.php';
    }

    // Update: Update an existing post
    public function update($postData, $id) {
        $title = $postData['title'];
        $content = $postData['content'];
        $category_id = $postData['category_id'];
        // Initialize the image filename from existing data
        $imageFilename = $postData['existing_image_filename'] ?? null;

        // Check if an image file is uploaded and there are no upload errors
        if (isset($_FILES['image']) && $_FILES['image']['error'] === UPLOAD_ERR_OK) {
            // Define allowed file types and size
            $allowedTypes = ['image/jpeg', 'image/png', 'image/gif'];
            $maxSize = 5 * 1024 * 1024; // 5 MB

            // Extract file information from the uploaded image
            $fileTmpPath = $_FILES['image']['tmp_name']; // Temporary path where the
uploaded file is stored
            $fileName = $_FILES['image']['name'];       // Original name of the
uploaded file
            $fileType = $_FILES['image']['type'];       // MIME type of the uploaded
file
            $fileSize = $_FILES['image']['size'];       // Size of the uploaded file
in bytes

            // Check file type and size
            if (in_array($fileType, $allowedTypes) && $fileSize <= $maxSize) {
                // Create a unique file name to avoid overwriting existing files
                $newFileName = uniqid() . '-' . $fileName;

                // Set the upload path
                $uploadPath = 'S:/xampp/htdocs/php8blogapp/public/images/';
                $destination = $uploadPath . $newFileName;
                // Ensure the upload directory exists
                if (!file_exists($uploadPath)) {
                    mkdir($uploadPath, 0777, true);
                }

                // Move the uploaded file
                if (move_uploaded_file($fileTmpPath, $destination)) {
                    // File uploaded successfully
                    $imageFilename = $newFileName; // Update $imageFilename with new
filename for database storage
                } else {
                    // Handle error (file move failed)
                    $errors[] = 'Failed to upload image.';
                }
            } else {
```

```
                // Handle invalid file type or size
                $errors[] = 'Invalid file type or size.';
            }
        } elseif ($_FILES['image']['error'] != UPLOAD_ERR_NO_FILE) {
            $errors[] = 'Error in file upload.';
        }

        try {
            // Call the updatePost method
            $this->model->updatePost($id, $title, $content, $category_id,
$imageFilename);

            // Redirect to the posts page after successful update
             $_SESSION['message'] = 'Post updated successfully!.';
              header('Location: /admin/posts');
              exit;
        } catch (Exception $e) {
            // If there's a database error, display it
            echo 'Error updating post: ' . $e->getMessage();
            exit;
        }
    }

    // Delete: Remove a post
    public function delete($id) {
        $this->model->deletePost($id);
        // Redirect or display success message
        header('Location: /admin/posts');
    }
}
```

Update The View Files

admin/posts/create.php

```
<!-- admin/posts/create.php -->
<div>
    <!-- View all Post Button -->
    <div style="text-align: right; margin-bottom: 20px;">
        <a href="/admin/posts/">View Posts</a>
    </div>

    <h1>Create Post</h1>
    <form method="POST" action="/admin/posts/store" enctype="multipart/form-data">
        <div>
            <label for="title">Title:</label>
            <input type="text" id="title" name="title">
        </div>

        <div>
```

```
            <label for="content">Content:</label>
            <textarea id="content" name="content"></textarea>
        </div>

        <div>
            <label for="image">Image:</label>
            <input type="file" id="image" name="image">
        </div>

        <!-- Populate Categories in this view -->
        <div>
            <label for="category_id">Category:</label>
            <select id="category_id" name="category_id">
                <?php foreach ($categories as $category): ?>
                    <option value="<?php echo $category->id; ?>"><?php echo
$category->name; ?></option>
                <?php endforeach; ?>
            </select>
        </div>


        <button type="submit">Create Post</button>
    </form>
</div>
```

admin/posts/edit.php

```
<div>
    <?php if ($post === null): ?>
        <div>Post not found</div>
        <?php return; ?>
        <?php endif; ?>
      <form method="POST" action="/admin/posts/update/<?php echo $post->id ?>"
enctype="multipart/form-data" style="margin-bottom: 20px;">
        <input type="hidden" name="id" value="<?php echo $post->id ?>">
        <div style="margin-bottom: 20px;">
            <label for="title">Title</label>
            <input type="text" id="title" name="title" value="<?php echo
htmlspecialchars($post->title, ENT_QUOTES); ?>">
        </div>
        <div style="margin-bottom: 20px;">
            <label for="content">Content</label>
            <textarea id="content" name="content"><?php echo htmlspecialchars($post-
>content, ENT_QUOTES); ?></textarea>
        </div>

        <div>
            <label for="image">Image</label>
            <input type="file" id="image" name="image">
            <?php if (!empty($post->image)): ?>
```

```
            <img src="/images/<?php echo htmlspecialchars($post->image, ENT_QUOTES);
?>" alt="Post Image" style="max-width: 200px;">
            <?php endif; ?>
        </div>

        <!-- Populate Categories in this view -->
        <div>
            <label for="category_id">Category:</label>
            <select id="category_id" name="category_id">
            <?php foreach ($categories as $category): ?>
            <option value="<?php echo $category->id; ?>" <?php echo $category->id ==
$post->category_id ? 'selected' : ''; ?>>
            <?php echo $category->name; ?>
            </option>
            <?php endforeach; ?>
            </select>
        </div>
        <button type="submit">Update Post</button>
    </form>
</div>
```

Display Posts with images in the View Files

Admin Side: (admin/posts/show.php)

```
<div>
    <!-- View all Post Button -->
    <div style="text-align: right; margin-bottom: 20px;">
        <a href="/admin/posts/">View Posts</a>
    </div>
    <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
    <!-- Check if the post has an image and display it -->
    <?php if ($post->image): ?>
        <img src="/images/<?php echo htmlspecialchars($post->image, ENT_QUOTES); ?>"
alt="Post Image" style="max-width: 200px;"> <!-- Small preview -->
    <?php endif; ?>
    <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
    <?php if (!empty($comments)): ?>
    <h3>Comments</h3>
    <?php foreach ($comments as $comment): ?>
        <div class="mb-2">
            <p><?php echo htmlspecialchars($comment->content, ENT_QUOTES); ?></p>
            <a href="/admin/comments/edit/<?php echo $comment->id; ?>" class="btn
btn-warning btn-sm">Edit</a>
            <form action="/admin/comments/delete/<?php echo $comment->id; ?>"
method="POST" style="display: inline;">
                <input type="hidden" name="_method" value="DELETE">
                <button type="submit">Delete</button>
                </form>
        </div>
```

```
    <?php endforeach; ?>
    <?php else: ?>
        <p>No comments found in this post.</p>
    <?php endif; ?>
</div>
```

Frontend Side: views/posts/index.php

```
<?php foreach ($posts as $post): ?>
        <div style="margin-bottom: 20px; margin-top: 20px;">
            <div>
                <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
                 <!-- Check if the post has an image and display it -->
                 <?php if ($post->image): ?>
                        <img src="/images/<?php echo htmlspecialchars($post->image,
ENT_QUOTES); ?>" alt="Post Image">
                    <?php endif; ?>
                <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
                <div>
                  <a href="/posts/show/<?php echo $post->id; ?>">View</a>
                </div>
            </div>
        </div>
    <?php endforeach; ?>
</div>
```

Frontend Side: views/posts/show.php

```
<!-- show.php -->
<div>
    <div>
        <div>
            <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
        </div>

        <div>
            <?php if ($post->image): ?>
                <img src="/images/<?php echo htmlspecialchars($post->image,
ENT_QUOTES); ?>" alt="Post Image" class="img-fluid mb-3">
            <?php endif; ?>
        </div>

        <div>
            <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
        </div>
```

```
        <div>
            <h3>Comments</h3>
            <?php foreach ($comments as $comment): ?>
                <div style="margin-bottom: 20px;">
                    <p><?php echo htmlspecialchars($comment->content, ENT_QUOTES);
?></p>
                </div>
            <?php endforeach; ?>

            <form action="/comments/store" method="post" style="margin-top: 20px;">
                <div style="margin-bottom: 20px;">
                    <label for="content">Content:</label>
                    <textarea id="content" name="content"></textarea>
                </div>
                <input type="hidden" name="post_id" value="<?php echo $post->id; ?>">
                <button type="submit">Submit</button>
            </form>
        </div>
    </div>
</div>
```

Step 12: Implement Validation in Our Blog

In this section we're going to add validation error messages when user fails to fill the creation and edit form correctly. We can do this using frontend languages like Javascript but lets use PHP for our case. Alright let's start.

Update the AdminController.php

In the store method of our AdminController add if condtion to check whether the form fields ***($title***, ***$content*** and ***$category_id*** of the create form are not empty. Here is the method.

```
// Create: Store a new post
    public function store($postData) {
        // Retrieve form data
        $title = $postData['title'];
        $content = $postData['content'];
        $category_id = $postData['category_id'];
        $imageFilename = null;  // Default to null if no image is uploaded

        // Validate title, content, and category
        $errors = [];
        if (empty($title)) {
            $errors[] = 'Title is required.';
        }
        if (empty($content)) {
            $errors[] = 'Content is required.';
        }
        if (empty($category_id)) {
            $errors[] = 'Category is required.';
        }
```

```php
        // Only proceed if there are no errors
        if (empty($errors)) {
        // Check if an image file is uploaded and there are no upload errors
        if (isset($_FILES['image']) && $_FILES['image']['error'] === UPLOAD_ERR_OK) {
            // Define allowed file types and size
            $allowedTypes = ['image/jpeg', 'image/png', 'image/gif'];
            $maxSize = 5 * 1024 * 1024; // 5 MB

            // Extract file information from the uploaded image
            $fileTmpPath = $_FILES['image']['tmp_name']; // Temporary path where the uploaded file is stored
            $fileName = $_FILES['image']['name'];       // Original name of the uploaded file
            $fileType = $_FILES['image']['type'];       // MIME type of the uploaded file
            $fileSize = $_FILES['image']['size'];       // Size of the uploaded file in bytes


            // Check file type and size
            if (in_array($fileType, $allowedTypes) && $fileSize <= $maxSize) {
                // Create a unique file name to avoid overwriting existing files
                $newFileName = uniqid() . '-' . $fileName;

                // Set the upload path for the image
                $uploadPath = 'S:/xampp/htdocs/php8blogapp/public/images/'; // Define the directory where uploaded images will be stored
                $destination = $uploadPath . $newFileName; // Concatenate the upload path with the new file name to create the full destination path

                // Ensure the upload directory exists
                if (!file_exists($uploadPath)) {
                    // Create the directory if it does not exist, with full read/write/execute permissions (0777) and enable recursion (true)
                    mkdir($uploadPath, 0777, true);
                }

                // Move the uploaded file
                if (move_uploaded_file($fileTmpPath, $destination)) {
                    // File uploaded successfully
                    $imageFilename = $newFileName; // Set this for database storage
                } else {
                    // Handle error (file move failed)
                    $errors[] = 'Failed to upload image.';
                }
            } else {
                // Handle invalid file type or size
                $errors[] = 'Invalid file type or size.';
            }
```

```
        } else {
            // Handle no file uploaded or upload error
            if ($_FILES['image']['error'] != UPLOAD_ERR_NO_FILE) {
                $errors[] = 'Error in file upload.';
            }
        }
    }

    // Check again for any errors including file upload errors
    if (empty($errors)) {
        // Create the post in the database
        try {
            $this->model->createPost($title, $content, $category_id, $imageFilename);
            // Redirect to the posts page after successful creation
            header('Location: /admin/posts');
            $_SESSION['message'] = 'Post created successfully!.';
            exit;
        } catch (Exception $e) {
            // If there's a database error, display it
            echo 'Error creating post: ' . $e->getMessage();
            exit;
        }
        } else {
            // Handle validation errors
            $_SESSION['form_errors'] = $errors;
            header('Location: /admin/posts/create');
            exit;
        }
    }
```

Update The View File: ***admin/posts/create.php***

What we are adding here is the code for rendering validation error messages.

```
<!-- admin/posts/create.php -->
<div>
    <!-- View all Post Button -->
    <div style="text-align: right; margin-bottom: 20px;">
        <a href="/admin/posts/">View Posts</a>
    </div>

    <h1>Create Post</h1>
        <!-- Validation error messages -->
        <?php if (isset($_SESSION['form_errors'])): ?>
            <div class="alert alert-danger">
                <?php foreach ($_SESSION['form_errors'] as $error): ?>
                    <p><?php echo $error; ?></p>
                <?php endforeach; ?>
                <?php unset($_SESSION['form_errors']); ?>
            </div>
```

```
        <?php endif; ?>
    <form method="POST" action="/admin/posts/store" enctype="multipart/form-data">
        <div>
            <label for="title">Title:</label>
            <input type="text" id="title" name="title">
        </div>

        <div>
            <label for="content">Content:</label>
            <textarea id="content" name="content"></textarea>
        </div>

        <div>
            <label for="image">Image:</label>
            <input type="file" id="image" name="image">
        </div>

        <!-- Populate Categories in this view -->
        <div>
            <label for="category_id">Category:</label>
            <select id="category_id" name="category_id">
                <?php foreach ($categories as $category): ?>
                    <option value="<?php echo $category->id; ?>"><?php echo
$category->name; ?></option>
                <?php endforeach; ?>
            </select>
        </div>


        <button type="submit">Create Post</button>
    </form>
</div>
```

Add Validation to Update Method

Modify the update method in the admincontroller

We can also add validation to our update method. This is when a user is editing a post. In case the form fields are empty we display error messages like what we did above. Let's go ahead and do this.

```
// Update: Update an existing post
    public function update($postData, $id) {
        $title = $postData['title'];
        $content = $postData['content'];
        $category_id = $postData['category_id'];
        // Initialize the image filename from existing data
        $imageFilename = $postData['existing_image_filename'] ?? null;

         // Validation
         $errors = [];
         if (empty($title)) {
```

```
            $errors[] = 'Title is required.';
        }
        if (empty($content)) {
            $errors[] = 'Content is required.';
        }
        if (empty($category_id)) {
            $errors[] = 'Category is required.';
        }

        // Only proceed if there are no errors
        if (empty($errors)) {
           // Check if an image file is uploaded and there are no upload errors
           if (isset($_FILES['image']) && $_FILES['image']['error'] ===
UPLOAD_ERR_OK) {
                // Define allowed file types and size
                $allowedTypes = ['image/jpeg', 'image/png', 'image/gif'];
                $maxSize = 5 * 1024 * 1024; // 5 MB

                // Extract file information from the uploaded image
                $fileTmpPath = $_FILES['image']['tmp_name']; // Temporary path where
the uploaded file is stored
                $fileName = $_FILES['image']['name'];       // Original name of the
uploaded file
                $fileType = $_FILES['image']['type'];       // MIME type of the
uploaded file
                $fileSize = $_FILES['image']['size'];       // Size of the uploaded
file in bytes

                // Check file type and size
                if (in_array($fileType, $allowedTypes) && $fileSize <= $maxSize) {
                    // Create a unique file name to avoid overwriting existing files
                    $newFileName = uniqid() . '-' . $fileName;

                    // Set the upload path
                    $uploadPath = 'S:/xampp/htdocs/php8blogapp/public/images/';
                    $destination = $uploadPath . $newFileName;
                    // Ensure the upload directory exists
                    if (!file_exists($uploadPath)) {
                        mkdir($uploadPath, 0777, true);
                    }

                    // Move the uploaded file
                    if (move_uploaded_file($fileTmpPath, $destination)) {
                        // File uploaded successfully
                        $imageFilename = $newFileName; // Update $imageFilename with
new filename for database storage
                    } else {
                        // Handle error (file move failed)
                        $errors[] = 'Failed to upload image.';
                    }
```

```
                } else {
                    // Handle invalid file type or size
                    $errors[] = 'Invalid file type or size.';
                }
            } elseif ($_FILES['image']['error'] != UPLOAD_ERR_NO_FILE) {
                $errors[] = 'Error in file upload.';
            }
         }
        // Check again for any errors including file upload errors
        if (empty($errors)) {
            try {
                // Call the updatePost method
                $this->model->updatePost($id, $title, $content, $category_id,
$imageFilename);

              // Redirect to the posts page after successful update
              $_SESSION['message'] = 'Post updated successfully!.';
              header('Location: /admin/posts');
              exit;
            } catch (Exception $e) {
                // If there's a database error, display it
                echo 'Error updating post: ' . $e->getMessage();
                exit;
            }
            } else {
                // Handle validation errors
                $_SESSION['form_errors'] = $errors;
                header('Location: /admin/posts/edit/' . $id);
                exit;
            }
        }
```

Update The View File: (admin/posts/edit)

```
<div>
    <?php if ($post === null): ?>
        <div>Post not found</div>
        <?php return; ?>
        <?php endif; ?>
        <!-- Validation error messages -->
        <?php if (isset($_SESSION['form_errors'])): ?>
        <div class="alert alert-danger">
            <?php foreach ($_SESSION['form_errors'] as $error): ?>
                <p><?php echo $error; ?></p>
            <?php endforeach; ?>
            <?php unset($_SESSION['form_errors']); ?>
        </div>
       <?php endif; ?>
      <form method="POST" action="/admin/posts/update/<?php echo $post->id ?>"
enctype="multipart/form-data" style="margin-bottom: 20px;">
```

```
        <input type="hidden" name="id" value="<?php echo $post->id ?>">
        <div style="margin-bottom: 20px;">
            <label for="title">Title</label>
            <input type="text" id="title" name="title" value="<?php echo
htmlspecialchars($post->title, ENT_QUOTES); ?>">
        </div>
        <div style="margin-bottom: 20px;">
            <label for="content">Content</label>
            <textarea id="content" name="content"><?php echo htmlspecialchars($post-
>content, ENT_QUOTES); ?></textarea>
        </div>

        <div>
            <label for="image">Image</label>
            <input type="file" id="image" name="image">
            <?php if (!empty($post->image)): ?>
            <img src="/images/<?php echo htmlspecialchars($post->image, ENT_QUOTES);
?>" alt="Post Image" style="max-width: 200px;">
            <?php endif; ?>
        </div>

        <!-- Populate Categories in this view -->
        <div>
            <label for="category_id">Category:</label>
            <select id="category_id" name="category_id">
            <?php foreach ($categories as $category): ?>
            <option value="<?php echo $category->id; ?>" <?php echo $category->id ==
$post->category_id ? 'selected' : ''; ?>>
            <?php echo $category->name; ?>
            </option>
            <?php endforeach; ?>
            </select>
        </div>
        <button type="submit">Update Post</button>
    </form>
</div>
```

Step 13: Implement Pagination To The Posts Data

To add pagination which means to display certain number of posts to our pages. We can do it this way.

Update the Post.php Model

```
 public function getPosts($pageNum = 1, $postsPerPage = 5) {
        // Calculate the offset for the SQL query based on the current page number
and posts per page
        $offset = ($pageNum - 1) * $postsPerPage;

        // Prepare an SQL statement to fetch posts with pagination
        // Posts are ordered by their ID in descending order
        // :limit and :offset are placeholders for pagination values
```

```
        $stmt = $this->db->conn->prepare('SELECT posts.*, categories.name as
category_name, posts.image as image FROM posts LEFT JOIN categories ON
posts.category_id = categories.id ORDER BY posts.id DESC LIMIT :limit OFFSET
:offset');

        // Bind the limit and offset values to the prepared statement
        // PDO::PARAM_INT ensures these values are treated as integers
        $stmt->bindValue(':limit', $postsPerPage, \PDO::PARAM_INT);
        $stmt->bindValue(':offset', $offset, \PDO::PARAM_INT);

        // Execute the SQL statement
        $stmt->execute();

        // Return all fetched posts as an array of objects
        return $stmt->fetchAll(\PDO::FETCH_OBJ);
    }
```

Limit and Offset Explained:

- In **SQL**, `**LIMIT**` and `**OFFSET**` are clauses used to constrain the number of rows returned by a query.

- `**LIMIT**` specifies the maximum number of rows to return.

- `**OFFSET**` specifies the number of rows to skip before starting to return rows from the query.

Why Subtract 1 in Offset Calculation:

- The calculation `**$offset = ($pageNum - 1) * $postsPerPage**;` is used to determine where to start fetching rows for a given page.

- Pagination typically starts from page 1, but the offset in SQL starts from 0.

- Subtracting 1 adjusts for this difference.

Example:

- Imagine you have 5 posts per page (`$postsPerPage = 5`).

- For page 1, you want to start from the very first post (offset 0). The calculation will be `(1 - 1) * 5 = 0`.

- For page 2, you want to skip the first 5 posts and start from the 6th post. The calculation will be `(2 - 1) * 5 = 5`.

- Thus, for page 1, the offset is 0 (starting from the first row), and for page 2, the offset is 5 (starting from the 6th row).

Add this method to **Post.php**

```
public function getTotalPostsCount() {
        // Prepare an SQL query to count the total number of posts in the database
        $stmt = $this->db->conn->query('SELECT COUNT(*) FROM posts');
        // Execute the query and return the count result
        // fetchColumn() retrieves the value of the first column in the result set
```

```
        return $stmt->fetchColumn();
    }
```

The rest of the methods remain the same. Let's ago ahead and update the **AdminController.php and PostController.php** indexes with this code.

AdminController.php:

```
// Read: Display a list of posts
    public function index() {
        // Determine the current page number
        $currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
        // Define how many posts you want per page
        $postsPerPage = 5;

        // Fetch posts for the current page
        $posts = $this->model->getPosts($currentPage, $postsPerPage);

        // Calculate the total number of pages
        $totalPosts = $this->model->getTotalPostsCount();
        $totalPages = ceil($totalPosts / $postsPerPage);

         // Load the view for displaying posts
         require BASE_DIR . '/app/views/admin/posts/index.php';
    }
```

PostController.php

```
// Display all posts
    public function index() {
         // Determine the current page number
         $currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;
         $postsPerPage = 5; // Define how many posts you want per page

         // Fetch posts for the current page
         $posts = $this->model->getPosts($currentPage, $postsPerPage);

         // Calculate the total number of pages
         $totalPosts = $this->model->getTotalPostsCount();
         $totalPages = ceil($totalPosts / $postsPerPage);

        require BASE_DIR . '/app/views/posts/index.php';
    }
```

- **isset($_GET['page'])**: This checks if the **page** parameter exists in the URL query string. For example, in the URL **http://example.com/posts?page=2**, it checks if **page=2** is present.
- **(int)$_GET['page']**: This is a type cast to integer. If **page** is present in the URL, its value is converted to an integer. This is important for security and validation purposes to ensure that the value used as the

page number is indeed an integer. Type casting helps prevent potential security issues, such as SQL injection when the value is used in a database query.

- **: 1**: This part of the code provides a default value. If **page** is not present in the URL, the current page number is assumed to be 1. This means that by default, or if the **page** parameter is missing, the application will show the first page of the content.

Putting it all together, **$currentPage = isset($_GET['page']) ? (int)$_GET['page'] : 1;** means "If **page** is specified in the URL and is a valid number, use it as the current page number; otherwise, default to page 1." This is a succinct way to handle pagination in PHP.

Update the View Files:

admin/posts/index.php

Place it below the php end foreach

```
<div>
  <!-- Display message -->
    <?php if (isset($_SESSION['message'])): ?>
        <div id="message" class="<?php echo strpos($_SESSION['message'], 'error') !==
false ? 'message-error' : 'message-success' ?> text-center">
            <?php
                echo $_SESSION['message'];
                unset($_SESSION['message']);
            ?>
        </div>
    <?php endif; ?>
      <div style="text-align: right;">
        <!-- Create Post Button -->
        <a href="/admin/posts/create">Create New Post</a>
        <a href="/admin/categories/create">Create Categories</a>
    </div>

    <?php foreach ($posts as $post): ?>
        <div style="margin-bottom: 20px; margin-top: 20px;">
            <div>
                <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
                <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
                <a href="/admin/posts/show/<?php echo $post->id; ?>">View</a>
                <a href="/admin/posts/edit/<?php echo $post->id; ?>">Edit</a>
                <p><strong>Category:</strong> <?php echo $post->category_name ?
htmlspecialchars($post->category_name, ENT_QUOTES) : 'No category'; ?></p>
                <!-- Delete Post -->
                <form action="/admin/posts/delete/<?php echo $post->id; ?>"
method="POST" style="display: inline;">
                    <input type="hidden" name="_method" value="DELETE">
                    <button type="submit">Delete</button>
                </form>
            </div>
        </div>
    <?php endforeach; ?>
```

```
    <!-- Pagination Controls -->
    <nav aria-label="Page navigation">
        <ul style="list-style: none; padding: 0;">
            <!--we loop through each page number-->
            <?php for ($i = 1; $i <= $totalPages; $i++): ?>
                <!--Check if the loop's current page number ($i, e.g., 3) is the same
as the current page ($currentPage, 3)-->
                <!--If yes, then this is the current page, so we could style it
differently if needed-->
                <li style="<?php echo $i == $currentPage ? 'font-weight: bold;' : '';
?>">
                    <!-- returns the page full URL e.g., http://admin/posts?page=2 --
>
                    <a href="/admin/posts?page=<?php echo $i; ?>"><?php echo $i;
?></a>
                </li>
            <?php endfor; ?>
        </ul>
    </nav>
</div>

<style>
.message-success {
color: green;
/* other styling */
}

.message-error {
    color: red;
    background-color: #ffd6d6;
    padding: 10px;
    border: 1px solid red;
    margin-bottom: 20px;
}
</style>
```

views/posts/index.php

```
<?php foreach ($posts as $post): ?>
        <div style="margin-bottom: 20px; margin-top: 20px;">
            <div>
                <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
                 <!-- Check if the post has an image and display it -->
                 <?php if ($post->image): ?>
                        <img src="/images/<?php echo htmlspecialchars($post->image,
ENT_QUOTES); ?>" alt="Post Image">
                    <?php endif; ?>
                <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
                <div>
                  <a href="/posts/show/<?php echo $post->id; ?>">View</a>
```

```
                </div>
            </div>
        </div>
    <?php endforeach; ?>
    <!-- Pagination Controls -->
    <nav aria-label="Page navigation">
        <ul style="list-style: none; padding: 0;">
            <!--we loop through each page number-->
            <?php for ($i = 1; $i <= $totalPages; $i++): ?>
                <!--Check if the loop's current page number ($i, e.g., 3) is the same
as the current page ($currentPage, 3)-->
                <!--If yes, then this is the current page, so we could style it
differently if needed-->
                <li style="<?php echo $i == $currentPage ? 'font-weight: bold;' : '';
?>">
                    <!-- returns the page full URL e.g., http://admin/posts?page=2 --
>
                    <a href="/?page=<?php echo $i; ?>"><?php echo $i; ?></a>
                </li>
            <?php endfor; ?>
        </ul>
    </nav>
</div>
```

In conclusion to determine the pagination. Here is how we calculate it mathematical wise.

```
# Given values for the example
currentPage = 1  # Assuming currentPage is 1 for the example
postsPerPage = 5
totalPosts = 22  # Let's assume there are 22 posts in total

# Calculate the total number of pages
totalPages = ceil(totalPosts / postsPerPage)

totalPages

Result 5.
```

Step 14: Implement User Authentication

In this step we will implement a user auth for both users and admins. Let’s go ahead and do that.

Update The User.php Model

app/models/User.php

```
<?php

namespace app\models;

// Include the bootstrap file for global setup
```

```php
require_once BASE_DIR . '/bootstrap.php';

class User {
    private $db;

    // Constructor: Initializes database connection
    public function __construct() {
        $this->db = new \Database; // Establishes database connection
    }

    // Fetches all users from the database
    public function getUsers() {
        $result = $this->db->conn->query('SELECT * FROM users');
        return $result->fetchAll(\PDO::FETCH_OBJ); // Returns users as objects
    }

    // Retrieves a single user by username
    public function getUserByUsername($username) {
        $stmt = $this->db->conn->prepare('SELECT * FROM users WHERE username = ?');
        $stmt->execute([$username]); // Executes with provided username
        return $stmt->fetch(\PDO::FETCH_OBJ); // Returns the user as an object
    }

    // Creates a new user record
    public function createUser($username, $password, $is_admin = 0) {
        $stmt = $this->db->conn->prepare('INSERT INTO users (username, password,
is_admin) VALUES (?, ?, ?)');
        $stmt->execute([$username, $password, $is_admin]); // Inserts new user
    }

    // Updates an existing user's username and password
    public function updateUser($id, $username, $password) {
        $stmt = $this->db->conn->prepare('UPDATE users SET username = ?, password = ?
WHERE id = ?');
        $stmt->execute([$username, $password, $id]); // Updates user by ID
    }

    // Deletes a user by ID
    public function deleteUser($id) {
        $stmt = $this->db->conn->prepare('DELETE FROM users WHERE id = ?');
        $stmt->execute([$id]); // Deletes the specified user
    }

     // Check if username exists
     public function doesUsernameExist($username) {
        $stmt = $this->db->conn->prepare("SELECT COUNT(*) FROM users WHERE username =
?");
        $stmt->execute([$username]);
        $count = $stmt->fetchColumn();
        return $count > 0;
```

```
    }

}
```

app/controllers/UserController.php

Let's go ahead and Update the ***UserController.php*** we created earlier.

```
<?php

namespace app\controllers;

require_once BASE_DIR . '/app/models/User.php';

class UserController {
    private $model;

    // Constructor to initialize the model
    public function __construct() {
        $this->model = new \app\models\User;
    }

    // Display the registration form
    public function register() {
        require __DIR__ . '/app/views/users/register.php';
    }

    // Handle the user registration request
    public function store($data) {
        // Check if username and password are provided
        if (empty($data['username']) || empty($data['password'])) {
            // Set an error message and redirect to the registration page
            $_SESSION['message'] = 'Username and password are required';
            header('Location: /users/register');
            exit();
        }

         // Check if the username already exists
        if ($this->model->doesUsernameExist($data['username'])) {
            // Handle the error, e.g., set an error message in the session
            $_SESSION['message'] = "Username already exists. Please choose another
username.";
            header('Location: /users/register');
            exit;
        }


        // Hash the password for secure storage
        $hashedPassword = password_hash($data['password'], PASSWORD_DEFAULT);
        // Create a new user record in the database
```

```
        $this->model->createUser($data['username'], $hashedPassword);

        // Set a success message and redirect to the login page
        $_SESSION['message'] = 'Registration successful. Please log in.';
        header('Location: /users/login');
        exit();
    }

    // Display the login form
    public function showLoginForm() {
        require BASE_DIR . '/app/views/users/login.php';
    }

    // Authenticate the user
    public function authenticate($data) {
        // Validate the provided username and password
        if (empty($data['username']) || empty($data['password'])) {
            // Set an error message and redirect to the login page
            $_SESSION['message'] = 'Username and password are required';
            header('Location: /users/login');
            exit();
        }

        // Retrieve the user from the database
        $user = $this->model->getUserByUsername($data['username']);

        // Check if the user exists and the password is correct
        // Access properties with -> since $user is now an object
        if ($user && password_verify($data['password'], $user->password)) {
            // Initialize session with user info
            // Make sure to access properties as object properties, not array keys
            $this->initializeUserSession($user);

            // Redirect user to the appropriate page based on their role
            // Again, accessing is_admin as an object property
            $this->redirectBasedOnRole($user->is_admin);
        } else {
            // If authentication fails, set an error message and redirect
            $_SESSION['message'] = 'Invalid username or password';
            header('Location: /users/login');
            exit();
        }
    }

    // Initialize the session with user data
    private function initializeUserSession($user) {
        // Set user ID and admin status in the session
        $_SESSION['user_id'] = $user->id;
        $_SESSION['is_admin'] = $user->is_admin;
    }
```

```
    // Redirect user based on their role
    private function redirectBasedOnRole($isAdmin) {
        // If user is an admin, redirect to the admin posts page
        if ($isAdmin) {
            header('Location: /admin/posts');
        } else {
            // If user is not an admin, redirect to a general user page
            header('Location: /'); // Replace with the appropriate user page
            }
            exit();
            }

        // Handle the user logout process
        public function logout() {
            // Clear all session variables, and destroy the session
            session_unset();
            session_destroy();
            // Redirect to the login page after logout
            header('Location: /users/login');
            exit();
        }
    }
```

This controller will have different methods for storing user credentials upon registration. Another method for authenticating the user and a logout method.

Create The View Files login and Register.php

views/users/register.php

```
<div>
    <!--Display Message-->
    <?php if (isset($_SESSION['message'])): ?>
        <div id="message" class="alert <?php echo strpos($_SESSION['message'],
'error') !== false ? 'alert-danger' : 'alert-success' ?> text-center">
            <?php
                echo $_SESSION['message'];
                unset($_SESSION['message']);
                ?>
                </div>
    <?php endif; ?>

    <h4>Register</h4>
    <!-- Registration form -->
    <form method="POST" action="/users/store">
        <div>
            <label for="username">Username</label>
            <input type="text" id="username" name="username">
        </div>
```

```
        <div>
            <label for="password">Password</label>
            <input type="password" id="password" name="password">
        </div>
        <div>
            <button type="submit">Register</button>
        </div>
    </form>
    <div>
        <a href="/users/login">Already have an account? Login</a>
    </div>
</div>
```

views/users/login.php

```
<div>
    <!-- Display message -->
    <?php if (isset($_SESSION['message'])): ?>
        <div id="message" class="alert <?php echo strpos($_SESSION['message'],
'error') !== false ? 'alert-danger' : 'alert-success' ?> text-center">
            <?php
                echo $_SESSION['message'];
                unset($_SESSION['message']);
                ?>
                </div>
    <?php endif; ?>

    <h4>Login</h4>
    <!-- Login form -->
    <form method="POST" action="/users/authenticate">
        <div>
            <label for="username">Username</label>
            <input type="text" id="username" name="username">
        </div>
        <div>
            <label for="password">Password</label>
            <input type="password" id="password" name="password">
        </div>
        <div>
            <button type="submit">Login</button>
        </div>
    </form>
    <div>
        <a href="/users/register">Not a member? Create Account</a>
    </div>
</div>
```

public/index.php

Add the following routes to the ***public/index.php.***

```
$routes = [
    'GET' => [
// User Auth Routes
        'users/register' => ['controller' => '\app\controllers\UserController',
'method' => 'register'],
        'users/login' => ['controller' => '\app\controllers\UserController', 'method'
=> 'showLoginForm'],
   ],

 'POST' => [
  // User Auth Routes
         'users/store' => ['controller' => '\app\controllers\UserController',
'method' => 'store'],
         'users/authenticate' => ['controller' => '\app\controllers\UserController',
'method' => 'authenticate'],
    ]
];
```

Add A Logout Link

Here we are going to add a logout link into the ***admin/posts/index.php file.***

```
<div>
  <!-- Display message -->
    <?php if (isset($_SESSION['message'])): ?>
        <div id="message" class="<?php echo strpos($_SESSION['message'], 'error') !==
false ? 'message-error' : 'message-success' ?> text-center">
            <?php
                echo $_SESSION['message'];
                unset($_SESSION['message']);
            ?>
        </div>
    <?php endif; ?>
    <div style="text-align: right;">
        <!-- Create Post Button -->
        <a href="/admin/posts/create">Create New Post</a>
        | <!-- Add separator for clarity -->
        <a href="/admin/categories/create">Create Categories</a>
        | <!-- Add separator for clarity -->
        <!-- User Login/Logout Links -->
        <?php if (isLoggedIn()): ?>
            <a href="/users/logout">Logout</a>
        <?php else: ?>
            <a href="/users/login">Login</a>
        <?php endif; ?>
    </div>

    <?php foreach ($posts as $post): ?>
        <div style="margin-bottom: 20px; margin-top: 20px;">
            <div>
```

```
                <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
                <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
                <a href="/admin/posts/show/<?php echo $post->id; ?>">View</a>
                <a href="/admin/posts/edit/<?php echo $post->id; ?>">Edit</a>
                <p><strong>Category:</strong> <?php echo $post->category_name ?
htmlspecialchars($post->category_name, ENT_QUOTES) : 'No category'; ?></p>
                <!-- Delete Post -->
                <form action="/admin/posts/delete/<?php echo $post->id; ?>"
method="POST" style="display: inline;">
                    <input type="hidden" name="_method" value="DELETE">
                    <button type="submit">Delete</button>
                </form>
            </div>
        </div>
    <?php endforeach; ?>
    <!-- Pagination Controls -->
    <nav aria-label="Page navigation">
        <ul style="list-style: none; padding: 0;">
            <!--we loop through each page number-->
            <?php for ($i = 1; $i <= $totalPages; $i++): ?>
                <!--Check if the loop's current page number ($i, e.g., 3) is the same
as the current page ($currentPage, 3)-->
                <!--If yes, then this is the current page, so we could style it
differently if needed-->
                <li style="<?php echo $i == $currentPage ? 'font-weight: bold;' : '';
?>">
                    <!-- returns the page full URL e.g., http://admin/posts?page=2 --
>
                    <a href="/admin/posts?page=<?php echo $i; ?>"><?php echo $i;
?></a>
                </li>
            <?php endfor; ?>
        </ul>
    </nav>
</div>

<style>
.message-success {
color: green;
/* other styling */
}

.message-error {
    color: red;
    background-color: #ffd6d6;
    padding: 10px;
    border: 1px solid red;
    margin-bottom: 20px;
}
</style>
```

Update the Routes File

```
 $routes = [

    'GET' => [
   'users/logout' => ['controller' => '\app\controllers\UserController', 'method' =>
'logout'],
]
```

Add Admin Checks

Now we are going to add admin checks. These will ensure its only the admin who will access the backend part.

Create a Base Controller (*app/controllers/AuthController.php)*

```
<?php

namespace app\controllers;

class AuthController {
    protected function ensureAdmin() {
        if (!isLoggedIn() || !isAdmin()) {
            header('Location: /users/login');
            exit;
        }
    }
}
```

This method ensure it's the admin who can access the administrative part. If the person is not an admin he or she will be redirected to the login page.

Extend the Base Controller

Now here we will extend the base controller we've just created to all our controllers that require admin checks.

These controllers are ***AdminController.php, CategoryController.php*** and ***CommentController.php***

```
class AdminController extends AuthController {
```

Call the Protected Method "ensureAdmin"

In the controller methods call the ***ensureAdmin*** function on all the methods you want to protect.

```
 // Read: Display a list of posts
    public function index() {
        $this->ensureAdmin();
```

Note: Do the same for all the other controllers.

Create a Utility File (utils.php)

Lastly create file called ***utils.php.***

```
<?php

function isLoggedIn() {
    return isset($_SESSION['user_id']);
}

function isAdmin() {
    return isset($_SESSION['is_admin']) && $_SESSION['is_admin'] == 1;
}

?>
```

The **isLoggedIn()** and **isAdmin()** functions defined in **utils.php** are global utility functions. They are designed to be used anywhere in your application, regardless of the context. This means you can call these functions in any script where **utils.php** is included, whether it's within controllers, views, or other PHP files.

Call The Utils File

Call the utils file from the ***public/index.php.*** Remember this is the entry file for PHP MVC app.

```
require BASE_DIR . '/utils.php';
```

Step 15: Working on the Frontend

Let's go ahead and implement the frontend. Here we are going to create Navbar, add css and add Javascript to different pages. Let's started.

Create a Base Layout File

This layout file is going to be extended by other files such **navbar.php** and **footer.php.** Also, its going to define the main content for the view files we've created so far.

public/Layouts/layout.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My MVC Blog</title>
    <link rel="stylesheet" href="/css/styles.css">
    <!--Bootstrap Styles-->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
crossorigin="anonymous">
</head>
<body class="d-flex flex-column min-vh-100">
    <?php include BASE_DIR . '/public/Layouts/navbar.php'; ?>
```

```
    <main class="flex-grow-1">
        <?php echo $content; ?>
    </main>

    <?php include BASE_DIR . '/public/Layouts/footer.php'; ?>
</body>
</html>
<!--Bootstrap Js-->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL"
crossorigin="anonymous"></script>
<script src="/js/script.js"></script>
```

Go ahead and create the two files i.e ***public/Layouts/navbar.php*** and ***public/Layouts/footer.php***.

Test it out *app/views/posts/index.php.*

Now to test this. You can Update one of the view files like the ***app/views/posts/index.php.***

```
<?php
// Generate the specific content for this page
ob_start(); // Start output buffering
foreach ($posts as $post): ?>
        <div style="margin-bottom: 20px; margin-top: 20px;">
            <div>
                <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
                 <!-- Check if the post has an image and display it -->
                 <?php if ($post->image): ?>
                        <img src="/images/<?php echo htmlspecialchars($post->image,
ENT_QUOTES); ?>" alt="Post Image">
                    <?php endif; ?>
                <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>
                <div>
                  <a href="/posts/show/<?php echo $post->id; ?>">View</a>
                </div>
            </div>
        </div>
    <?php endforeach; ?>
    <nav aria-label="Page navigation">
        <ul style="list-style: none; padding: 0;">
            <!--we loop through each page number-->
            <?php for ($i = 1; $i <= $totalPages; $i++): ?>
                <!--Check if the loop's current page number ($i, e.g., 3) is the same
as the current page ($currentPage, 3)-->
                <!--If yes, then this is the current page, so we could style it
differently if needed-->
                <li style="<?php echo $i == $currentPage ? 'font-weight: bold;' : '';
?>">
```

```
                    <!-- returns the page full URL e.g., http://admin/posts?page=2 --
>
                    <a href="/admin/posts?page=<?php echo $i; ?>"><?php echo $i;
?></a>
                </li>
            <?php endfor; ?>
        </ul>
    </nav>
</div>
<?php
$content = ob_get_clean(); // Store buffered content in $content

// Include the layout
include BASE_DIR . '/public/Layouts/layout.php'; ?>
```

1. **ob_start():** Begins buffering. After this call, any output (e.g., from **echo**, **print**, or direct HTML outside PHP tags) is stored in the buffer instead of being sent to the browser.
2. **Content Generation:** The **foreach** loop generates content based on **$posts**. This content is captured by the output buffer instead of being immediately displayed.
3. **$content = ob_get_clean();:** The generated content is retrieved from the buffer and assigned to the **$content** variable. The output buffer is then cleared and closed, preventing the generated content from being automatically sent to the browser.

Update Navbar and Footer Files

Navbar.php

Here we will update the navbar file with the code related to it. The code will have bootstrap styles for styling the navbar.

```
<header>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
      <!-- Logo on the left with some margin to push it to the right -->
        <a class="navbar-brand ms-3" href="/"> <!-- ms-3 for margin start -->
            <img src="/images/logo.jpeg" alt="Logo" style="height: 40px;"> <!--
Adjust logo size as needed -->
        </a>
        <!-- Toggler for mobile view -->
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-
bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-
label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>

        <!-- Menu items -->
        <div class="collapse navbar-collapse justify-content-end" id="navbarNav">
            <ul class="navbar-nav">
                <li class="nav-item">
                    <a class="nav-link" href="/">Home</a>
                </li>
```

```
                <?php if (isLoggedIn()): ?>
                    <li class="nav-item">
                        <a class="nav-link" href="/users/logout">Logout</a>
                    </li>
                <?php else: ?>
                    <li class="nav-item">
                        <a class="nav-link" href="/users/login">Login</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="/users/register">Register</a>
                    </li>
                <?php endif; ?>
            </ul>
        </div>
         <!-- ms-auto class pushes the form to the right -->
        <div class="d-flex ms-auto">
            <form action="/search" method="GET" class="d-flex" required>
                <input type="text" name="query" class="form-control me-2"
placeholder="Search...">
                <button type="submit" class="btn btn-outline-success me-
2">Search</button>
            </form>
        </div>
        </nav>
    </header>
```

Footer.php

```
<footer class="bg-dark text-white text-center py-3 mt-auto">
        <p class="mb-0">&copy; <?php echo date('Y'); ?> My MVC BLOG</p>
</footer>
```

Add Bootstrap Styles To The Frontend View Files.

Let's add bootstrap styles to the rest of the view files we've created so far.

app/views/posts/index.php

```
<?php
// Generate the specific content for this page
ob_start(); // Start output buffering
foreach ($posts as $post):
    ?>
   <div class="container mt-4">
    <div class="row">
        <div class="col-md-8 offset-md-2">
            <div class="card">
                <h2 class="card-header"><?php echo htmlspecialchars($post->title,
ENT_QUOTES); ?></h2>
                <div class="card-body">
```

```
                <!-- Check if the post has an image and display it -->
                    <?php if ($post->image): ?>
                        <img src="/images/<?php echo htmlspecialchars($post->image,
ENT_QUOTES); ?>" alt="Post Image" class="img-fluid mb-3">
                    <?php endif; ?>
                    <p class="card-text"><?php echo htmlspecialchars($post->content,
ENT_QUOTES); ?></p>
                    <!-- Check if category data is available -->
                    <p class="card-text"><strong>Category:</strong> <?php echo $post-
>category_name ? htmlspecialchars($post->category_name, ENT_QUOTES) : 'No category';
?></p>
                    </div>
                    <div class="card-footer text-muted">
                        <a href="/posts/show/<?php echo $post->id; ?>" class="btn
btn-primary">View</a>
                    </div>
                </div>
            </div>
        </div>
    <hr>
<?php endforeach ?>

<!-- Pagination Controls -->
<nav aria-label="Page navigation">
      <ul class="pagination">
          <?php for ($i = 1; $i <= $totalPages; $i++): ?>
              <li class="page-item <?php echo $i == $currentPage ? 'active' : '';
?>">
                  <a class="page-link" href="/?page=<?php echo $i; ?>"><?php echo $i;
?></a>
              </li>
          <?php endfor; ?>
      </ul>
   </nav>
<?php
$content = ob_get_clean(); // Store buffered content in $content

// Include the layout
include BASE_DIR . '/public/Layouts/layout.php'; ?>
```

app/views/posts/show.php

```
<?php
// Generate the specific content for this page
ob_start(); // Start output buffering
    ?>
<!-- show.php -->
<div class="container mt-4 mb-5">
    <div class="card">
        <div class="card-header">
```

```
            <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
        </div>

        <div class="card-body">
            <?php if ($post->image): ?>
                <img src="/images/<?php echo htmlspecialchars($post->image,
ENT_QUOTES); ?>" alt="Post Image" class="img-fluid mb-3">
            <?php endif; ?>
            <p class="card-text"><?php echo htmlspecialchars($post->content,
ENT_QUOTES); ?></p>
        </div>

        <div class="card-footer ">
            <h3>Comments</h3>
            <?php foreach ($comments as $comment): ?>
                <div class="mb-2">
                    <p class="card-text"><?php echo htmlspecialchars($comment-
>content, ENT_QUOTES); ?></p>
                </div>
            <?php endforeach; ?>

            <form action="/comments/store" method="post" class="mt-4">
                <div class="mb-3">
                    <label for="content" class="form-label">Content:</label>
                    <textarea id="content" name="content" class="form-
control"></textarea>
                </div>
                <input type="hidden" name="post_id" value="<?php echo $post->id; ?>">
                <button type="submit" class="btn btn-primary">Submit</button>
            </form>
        </div>
    </div>
</div>
<?php
$content = ob_get_clean(); // Store buffered content in $content

// Include the layout
include BASE_DIR . '/public/Layouts/layout.php'; ?>
```

Add Bootstrap Styles To The Backend View Files.

app/views/admin/posts/index.php

```
<!--views/admin/posts/index.php-->
<?php
```

```
// Start output buffering
ob_start();
?>
<!-- Display message -->
<?php if (isset($_SESSION['message'])): ?>
        <div id="message" class="<?php echo strpos($_SESSION['message'], 'error') !==
false ? 'message-error' : 'message-success' ?> text-center">
            <?php
                echo $_SESSION['message'];
                unset($_SESSION['message']);
            ?>
        </div>
    <?php endif; ?>

<div class="container mt-4">
    <div class="row">
        <div class="col text-end">
            <!-- Create Post Button -->
            <a href="/admin/posts/create" class="btn btn-success me-2">Create New
Post</a>
            <!-- Create Category Button -->
            <a href="/admin/categories/create" class="btn btn-success">Create
Categories</a>
        </div>
    </div>

    <?php foreach ($posts as $post): ?>
        <div class="mb-3 mt-2 card">
            <div class="card-body">
                <h2 class="card-title"><?php echo htmlspecialchars($post->title,
ENT_QUOTES); ?></h2>
                <p class="card-text"><?php echo htmlspecialchars($post->content,
ENT_QUOTES); ?></p>
                <a href="/admin/posts/show/<?php echo $post->id; ?>" class="btn btn-
info btn-sm">View</a>
                <a href="/admin/posts/edit/<?php echo $post->id; ?>" class="btn btn-
warning btn-sm">Edit</a>
                <!—Implement ajax here-->
                <button type="button" onclick="deletePost(<?php echo $post->id; ?>)"
class="btn btn-danger btn-sm">Delete</button>
                </div>
                </div>
                <?php endforeach; ?>
                <!-- Pagination Controls -->
                <nav aria-label="Page navigation">
                    <ul class="pagination">
                    <!--we loop through each page number-->
                        <?php for ($i = 1; $i <= $totalPages; $i++): ?>
                            <!--Check if the loop's current page number ($i e.g 3) is
the same as the current page ($currentPage 3)-->
```

```
                            <!--If yes, then this is the current page, so mark it as
'active'-->
                            <!--If no, then it's just a regular page-->
                            <li class="page-item <?php echo $i == $currentPage ?
'active' : ''; ?>">
                            <!-- returns the page full url e.g
http://admin/posts/posts?page=2 -->
                                <a class="page-link" href="/admin/posts?page=<?php
echo $i; ?>"><?php echo $i; ?></a>
                            </li>
                        <?php endfor; ?>
                    </ul>
                </nav>
            </div>
<?php
$content = ob_get_clean(); // Store buffered content in $content
// Include the layout
include BASE_DIR . '/public/Layouts/layout.php'; ?>

<style>
.message-success {
color: green;
/* other styling */
}

.message-error {
    color: red;
    background-color: #ffd6d6;
    padding: 10px;
    border: 1px solid red;
    margin-bottom: 20px;
}
</style>
```

Implement Ajax To admin/posts View File

The **deletePost** is a button that triggers an ajax request to confirm if one wants to really delete a post.

So, on the ***public/js/script.js*** update it with this code.

public/js/script.js

```
 // JavaScript delete functionality for admin posts
function deletePost(postId) {
    if (confirm("Are you sure you want to delete this post?")) {
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "/admin/posts/delete/" + postId, true);
        xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        xhr.onload = function() {
            if (this.status === 200) {
```

```
                // Handle success
                location.reload(); // Reloads the current page
            } else {
                // Handle error
                alert("Error deleting post.");
            }
        };
        xhr.send("id=" + postId);
    }
}
```

Finally update the ***AdminController.php*** with this method.

```
// Delete: Remove a post
    public function delete($id) {
        $this->ensureAdmin();
        $postDeleted = $this->model->deletePost($id);
            if ($postDeleted) {
                header('Location: /admin/posts');
            } else {
                die('Comment not found');
            }
        }
```

app/views/admin/posts/create.php

Let's add the styling for the admin create file. Update the file as follows.

```
<!-- views/admin/posts/create.php -->
<?php
// Generate the specific content for this page
ob_start(); // Start output buffering
?>
<!-- admin/posts/create.php -->
<div class="container mt-4 mb-5">
    <!-- View all Post Button -->
    <div class="mb-4 text-end">
        <a href="/admin/posts/" class="btn btn-success">View Posts</a>
    </div>

    <h1 class="mb-4">Create Post</h1>

    <?php if (isset($_SESSION['form_errors'])): ?>
        <div class="alert alert-danger">
            <?php foreach ($_SESSION['form_errors'] as $error): ?>
                <p><?php echo $error; ?></p>
            <?php endforeach; ?>
            <?php unset($_SESSION['form_errors']); ?>
        </div>
    <?php endif; ?>
```

```
    <form method="POST" action="/admin/posts/store" enctype="multipart/form-data">
        <div class="mb-3">
            <label for="title" class="form-label">Title:</label>
            <input type="text" id="title" name="title" class="form-control">
        </div>

        <div class="mb-3">
            <label for="content" class="form-label">Content:</label>
            <textarea id="content" name="content" class="form-control"></textarea>
        </div>

        <div class="mb-3">
            <label for="category_id" class="form-label">Category:</label>
            <select id="category_id" name="category_id" class="form-select">
                <?php foreach ($categories as $category): ?>
                    <option value="<?php echo $category->id; ?>"><?php echo
$category->name; ?></option>
                <?php endforeach; ?>
            </select>
        </div>

        <div class="mb-3">
            <label for="image" class="form-label">Image:</label>
            <input type="file" id="image" name="image" class="form-control">
        </div>

        <button type="submit" class="btn btn-primary">Create Post</button>
    </form>
</div>
<?php
$content = ob_get_clean(); // Store buffered content in $content

// Include the layout
include BASE_DIR . '/public/Layouts/layout.php'; ?>
```

app/views/admin/posts/show.php

```
<?php
// Start output buffering
ob_start();
?>
  <!-- Display message -->
<?php if (isset($_SESSION['message'])): ?>
        <div id="message" class="<?php echo strpos($_SESSION['message'], 'error') !==
false ? 'message-error' : 'message-success' ?> text-center">
            <?php
                echo $_SESSION['message'];
                unset($_SESSION['message']);
            ?>
```

```
        </div>
    <?php endif; ?>

<div class="container mt-4">
    <!-- View all Post Button -->
    <div class="mb-4 text-end">
        <a href="/admin/posts/" class="btn btn-success">View Posts</a>
    </div>
    <h2><?php echo htmlspecialchars($post->title, ENT_QUOTES); ?></h2>
    <!-- Check if the post has an image and display it -->
    <?php if ($post->image): ?>
        <img src="/images/<?php echo htmlspecialchars($post->image, ENT_QUOTES); ?>"
alt="Post Image" class="img-fluid mb-3" style="max-width: 200px;"> <!-- Small preview
-->
    <?php endif; ?>
    <p><?php echo htmlspecialchars($post->content, ENT_QUOTES); ?></p>

    <h3>Comments</h3>
    <?php foreach ($comments as $comment): ?>
        <div class="mb-2">
            <p><?php echo htmlspecialchars($comment->content, ENT_QUOTES); ?></p>
            <a href="/admin/comments/edit/<?php echo $comment->id; ?>" class="btn
btn-warning btn-sm">Edit</a>
            <button onclick="deleteComment(<?php echo $comment->id; ?>)" class="btn
btn-danger btn-sm">Delete</button>
        </div>
    <?php endforeach; ?>
        </div>
<?php
$content = ob_get_clean(); // Store buffered content in $content

// Include the layout
include BASE_DIR . '/public/Layouts/layout.php'; ?>

<style>
.message-success {
color: green;
/* other styling */
}

.message-error {
    color: red;
    background-color: #ffd6d6;
    padding: 10px;
    border: 1px solid red;
    margin-bottom: 20px;
}
</style>
```

The **deleteComment** button onClick is a ajax request. Let's modify the following files.

public/js/script.js

```
// JavaScript delete functionality for comments
function deleteComment(commentId) {
    if (confirm("Are you sure you want to delete this comment?")) {
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "/admin/comments/delete/" + commentId, true);
        xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        xhr.onload = function() {
            if (this.status === 200) {
                // Handle success - remove the comment from the page or reload
                location.reload();
            } else {
                alert("Error deleting comment.");
            }
        };
        xhr.send("id=" + commentId);
    }
}
```

app/views/admin/posts/edit.php

Let's style edit view in the admin/posts directory. Apply the following bootstrap

```
<!--views/admin/posts/edit.php-->

<?php
// Generate the specific content for this page
ob_start(); // Start output buffering
?>
<div class="container mt-4">
    <?php if ($post === null): ?>
        <div class="alert alert-warning">Post not found</div>
        <?php return; ?>
    <?php endif; ?>

    <?php if (isset($_SESSION['form_errors'])): ?>
        <div class="alert alert-danger">
            <?php foreach ($_SESSION['form_errors'] as $error): ?>
                <p><?php echo $error; ?></p>
            <?php endforeach; ?>
            <?php unset($_SESSION['form_errors']); ?>
        </div>
    <?php endif; ?>

    <form method="POST" action="/admin/posts/update/<?php echo $post->id ?>"
enctype="multipart/form-data" class="mb-3">
        <input type="hidden" name="id" value="<?php echo $post->id ?>">
        <div class="mb-3">
```

```
            <label for="title" class="form-label">Title</label>
            <input type="text" id="title" name="title" value="<?php echo
htmlspecialchars($post->title, ENT_QUOTES); ?>" class="form-control">
        </div>
        <div class="mb-3">
            <label for="content" class="form-label">Content</label>
            <textarea id="content" name="content" class="form-control"><?php echo
htmlspecialchars($post->content, ENT_QUOTES); ?></textarea>
        </div>
        <div class="mb-3">
            <label for="image" class="form-label">Image</label>
            <input type="file" id="image" name="image" class="form-control">
            <?php if (!empty($post->image)): ?>
            <img src="/images/<?php echo htmlspecialchars($post->image, ENT_QUOTES);
?>" alt="Post Image" class="img-fluid mt-2" style="max-width: 200px;">
            <?php endif; ?>
        </div>
        <div class="mb-3">
            <label for="category_id" class="form-label">Category:</label>
            <select id="category_id" name="category_id" class="form-select">
            <?php foreach ($categories as $category): ?>
            <option value="<?php echo $category->id; ?>" <?php echo $category->id ==
$post->category_id ? 'selected' : ''; ?>>
            <?php echo $category->name; ?>
            </option>
            <?php endforeach; ?>
            </select>
        </div>
        <button type="submit" class="btn btn-primary">Update Post</button>
        </form>
    </div>
<?php
$content = ob_get_clean(); // Store buffered content in $content
// Include the layout
include BASE_DIR . '/public/Layouts/layout.php'; ?>
```

Add Login and Register Bootstrap Styles

Let's add the CSS styles for the login and registration files.

app/views/users/login.php

```
<?php
// Generate the specific content for this page
ob_start(); // Start output buffering
?>
<!-- Display message -->
<?php if (isset($_SESSION['message'])): ?>
        <div id="message" class="<?php echo strpos($_SESSION['message'], 'error') !==
false ? 'message-error' : 'message-success' ?> text-center">
            <?php
```

```
                echo $_SESSION['message'];
                unset($_SESSION['message']);
            ?>
        </div>
    <?php endif; ?>

    <div class="container mt-5">
    <div class="row">
        <div class="col-md-6 offset-md-3">
            <div class="card">
                <div class="card-body">
                    <h4 class="card-title text-center">Login</h4>
                    <!-- Login form -->
                    <form method="POST" action="/users/authenticate">
                        <div class="mb-3">
                            <label for="username" class="form-label">Username</label>
                            <input type="text" id="username" name="username"
class="form-control">
                        </div>
                        <div class="mb-3">
                            <label for="password" class="form-label">Password</label>
                            <input type="password" id="password" name="password"
class="form-control">
                        </div>
                        <div class="d-grid gap-2">
                            <button type="submit" class="btn btn-primary btn-
block">Login</button>
                        </div>
                            </form>
                                <div class="text-center mt-3">
                                <a href="/users/register">Not a member? Create
Account</a>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
<?php
$content = ob_get_clean(); // Store buffered content in $content

// Include the layout
include BASE_DIR . '/public/Layouts/layout.php'; ?>

<style>
.message-success {
color: green;
/* other styling */
}
```

```
.message-error {
    color: red;
    background-color: #ffd6d6;
    padding: 10px;
    border: 1px solid red;
    margin-bottom: 20px;
}
</style>
```

app/views/users/register.php

```
<?php
// Generate the specific content for this page
ob_start(); // Start output buffering
?>

<?php if (isset($_SESSION['message'])): ?>
        <div id="message" class="<?php echo strpos($_SESSION['message'], 'error') !==
false ? 'message-error' : 'message-success' ?> text-center">
            <?php
                echo $_SESSION['message'];
                unset($_SESSION['message']);
            ?>
        </div>
    <?php endif; ?>


<div class="container mt-5">
    <div class="row">
        <div class="col-md-6 offset-md-3">
            <div class="card">
                <div class="card-body">
                    <h4 class="card-title text-center">Register</h4>
                    <!-- Registration form -->
                    <form method="POST" action="/users/store">
                        <div class="mb-3">
                            <label for="username" class="form-label">Username</label>
                            <input type="text" id="username" name="username"
class="form-control">
                        </div>
                        <div class="mb-3">
                            <label for="password" class="form-label">Password</label>
                            <input type="password" id="password" name="password"
class="form-control">
                        </div>
                        <div class="d-grid gap-2">
                            <button type="submit" class="btn btn-
primary">Register</button>
```

```
                    </div>
                </form>
                <div class="text-center mt-3">
                    <a href="/users/login">Already have an account? Login</a>
                </div>
            </div>
        </div>
    </div>
</div>
<?php
$content = ob_get_clean(); // Store buffered content in $content

// Include the layout
include BASE_DIR . '/public/Layouts/layout.php'; ?>

<style>
.message-success {
color: green;
/* other styling */
}

.message-error {
    color: red;
    background-color: #ffd6d6;
    padding: 10px;
    border: 1px solid red;
    margin-bottom: 20px;
}
</style>
```

Hide PHP Alert Messages

Let's add a Javascript code to hide the alert messages that pops up when someone registers or trys to register by filling partly the form or in full. Here is the Javascript code.

Add it to the ***public/js/script.js***

```
// JavaScript Timeout functionality for admin posts
document.addEventListener('DOMContentLoaded', function() {
    // console.log("DOM fully loaded and parsed");
    setTimeout(function() {
        var messageElement = document.getElementById('message');
        console.log("Looking for message element", messageElement);
        if (messageElement) {
            messageElement.style.display = 'none';
        }
    }, 5000); // Hide the message after 5 seconds
});
```

Add Styles To Category View Files

app/views/admin/categories/create.php

```
<?php
// Generate the specific content for this page
ob_start(); // Start output buffering
    ?>
<div class="container mt-4">
    <!-- View Category Button -->
    <div class="mb-4 text-end">
        <a href="/admin/categories/" class="btn btn-success">View Categories</a>
    </div>
    <form method="POST" action="/admin/categories/store" class="mb-3">
        <div class="mb-3">
            <label for="name" class="form-label">Category Name</label>
            <input type="text" id="name" name="name" class="form-control" required>
        </div>
        <button type="submit" class="btn btn-primary">Create Category</button>
    </form>
</div>
<?php
$content = ob_get_clean(); // Store buffered content in $content

// Include the layout
include BASE_DIR . '/public/Layouts/layout.php'; ?>
```

app/views/admin/categories/edit.php

```
<?php
// Generate the specific content for this page
ob_start(); // Start output buffering
    ?>
<div class="container mt-4">
    <form method="POST" action="/admin/categories/update/<?php echo $category->id;
?>" class="mb-3">
        <input type="hidden" name="id" value="<?php echo $category->id; ?>">
        <div class="mb-3">
            <label for="name" class="form-label">Name:</label>
            <input type="text" id="name" name="name" value="<?php echo
htmlspecialchars($category->name, ENT_QUOTES); ?>" class="form-control" required>
        </div>
        <!-- Add more input fields for other category properties... -->
        <button type="submit" class="btn btn-primary">Update</button>
    </form>
</div>

<?php
$content = ob_get_clean(); // Store buffered content in $content
```

```
// Include the layout
include BASE_DIR . '/public/Layouts/layout.php'; ?>
```

app/views/admin/categories/index.php

```
<?php
// Generate the specific content for this page
ob_start(); // Start output buffering
    ?>
    <div class="container mt-4">
      <div class="row">
        <div class="col text-end">
            <!-- Create Post Button -->
            <a href="/admin/categories/create" class="btn btn-success me-2">Create
Category</a>
            <!-- Create Category Button -->
            <a href="/admin/posts/" class="btn btn-success">View Posts</a>
        </div>
    </div>

    <h2 class="card-title">Categories</h2>
    <?php foreach ($categories as $category): ?>
        <div class="mb-3 mt-2 card">
          <div class="card-body">
            <p class="card-text"><?php echo htmlspecialchars($category->name,
ENT_QUOTES); ?></p>
                <a href="/admin/categories/edit/<?php echo $category->id; ?>"
class="btn btn-warning btn-sm">Edit</a>
                <a href="/admin/categories/posts/<?php echo $category->id; ?>"
class="btn btn-warning btn-sm">View Post</a>
                <form method="POST" action="/admin/categories/delete/<?php echo
$category->id ?>" class="d-inline">
                    <button type="button" onclick="deleteCategory(<?php echo
$category->id ?>)" class="btn btn-danger btn-sm">Delete</button>
                </form>
            </div>
        </div>
        <?php endforeach; ?>
    </div>
<?php
$content = ob_get_clean(); // Store buffered content in $content

// Include the layout
include BASE_DIR . '/public/Layouts/layout.php';
?>
```

Add Ajax Delete Confirmation For Category

Just as we did for admin/posts and comments. I mean adding ajax for delete confirmation. I have added a similar functionality in the above file. Add this Javascript code in script.js file

public/js/script.js

```
// JavaScript delete functionality for categories
function deleteCategory(categoryId) {
    if (confirm("Are you sure you want to delete this category?")) {
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "/admin/categories/delete/" + categoryId, true);
        xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        xhr.onload = function() {
            if (this.status === 200) {
                // Handle success - remove the category from the page or reload
                location.reload();
            } else {
                alert("Error deleting category.");
            }
        };
        xhr.send("id=" + categoryId);
    }
}
```

app/views/admin/categories/posts.php

```
<!-- admin/categories/posts.php -->
<?php
// Generate the specific content for this page
ob_start(); // Start output buffering
?>
<div class="container mt-4">
    <h2 class="mb-3">Posts in Category: <?php echo htmlspecialchars($category->name,
ENT_QUOTES); ?></h2>
    <?php if (!empty($posts)): ?>
        <div class="list-group">
            <?php foreach ($posts as $post): ?>
                <div class="list-group-item list-group-item-action">
                    <h3 class="h5"><?php echo htmlspecialchars($post->title,
ENT_QUOTES); ?></h3>
                    <p><?php echo htmlspecialchars($post->content, ENT_QUOTES);
?></p>
                    <!-- Add links for edit, delete, view, etc. here -->
                    <a href="/admin/posts/edit/<?php echo $post->id; ?>" class="btn
btn-primary btn-sm">Edit</a>
                    <button type="button" onclick="deletePost(<?php echo $post->id;
?>)" class="btn btn-danger btn-sm">Delete</button>
                    <a href="/admin/posts/show/<?php echo $post->id; ?>" class="btn
btn-secondary btn-sm">View</a>
                </div>
            <?php endforeach; ?>
        </div>
    <?php else: ?>
        <p class="alert alert-info">No posts found in this category.</p>
```

```
    <?php endif; ?>
</div>
<?php
$content = ob_get_clean(); // Store buffered content in $content
// Include the layout
include BASE_DIR . '/public/Layouts/layout.php'; ?>
```

Add Bootstrap Styles For Comments

app/views/admin/comments/edit.php

```
<?php
if (!$comment) {
    echo "<div class='alert alert-warning' role='alert'>Comment not found</div>";
    return;
}
?>
<?php
// Generate the specific content for this page
ob_start(); // Start output buffering
    ?>
<div class="container mt-3">
    <form method="POST" action="/admin/comments/update/<?php echo $comment->id; ?>"
class="needs-validation" novalidate>
        <input type="hidden" name="id" value="<?php echo $comment->id; ?>">
        <input type="hidden" name="post_id" value="<?php echo $comment->post_id; ?>">

        <div class="mb-3">
            <label for="content" class="form-label">Comment Content</label>
            <textarea id="content" name="content" class="form-control" required><?php
echo htmlspecialchars($comment->content, ENT_QUOTES); ?></textarea>
            <div class="invalid-feedback">
                Please enter a comment content.
            </div>
        </div>

        <button type="submit" class="btn btn-primary">Update Comment</button>
    </form>
</div>
<?php
$content = ob_get_clean(); // Store buffered content in $content

// Include the layout
include BASE_DIR . '/public/Layouts/layout.php';
?>
```

Add A Styles File For Basic styling

Let's create a ***public/css/styles.css*** for adding basic styles to our MVC blog. Here is how to go about it.

public/css/styles.css

```
body, html {
    height: 100%;
}

main {
    flex: 1;
}

header nav {
    display: flex;
    justify-content: space-between;
    align-items: center;
}
header nav ul {
    list-style: none;
    display: flex;
    margin: 0;
    padding: 0;
}
header nav ul li {
    margin-left: 20px; /* Spacing between menu items */
}

.message-success {
    color: green;
    /* other styling */
}

.message-error {
    color: red;
    background-color: #ffd6d6;
    padding: 10px;
    border: 1px solid red;
    margin-bottom: 20px;
}

.form-errors {
    color: red;
    background-color: #ffd6d6;
    padding: 10px;
    border: 1px solid red;
    margin-bottom: 20px;
}
```

Update Your *public/Layouts/layout.php* File

```
<link rel="stylesheet" href="/css/styles.css">
```

Add Code To The 404.php File

app/views/404.php

```
<?php
// Start output buffering to capture the 404 message
ob_start();
?>
<div class="container mt-4">
    <div class="text-center">
        <h1 class="display-1">404</h1>
        <p class="lead">Oops! The page you are looking for cannot be found.</p>
        <p>It might have been removed, had its name changed, or is temporarily
unavailable.</p>
        <a href="/" class="btn btn-primary">Go to Homepage</a>
    </div>
</div>
<?php
// End buffering and store the content in $content
$content = ob_get_clean();

// Include the layout, which uses $content
include BASE_DIR . '/public/Layouts/layout.php';
?>
```

Step 17: Add Search For Posts

Let's go ahead and add search functionality for our posts.

app/models/Posts.php

```
   public function searchPosts($query) {
        // Prepare the query for a LIKE statement
        $query = "%" . $query . "%";
        $stmt = $this->db->conn->prepare('SELECT posts.*, categories.name as
category_name FROM posts LEFT JOIN categories ON posts.category_id = categories.id
WHERE posts.title LIKE :query OR posts.content LIKE :query ORDER BY posts.id DESC');
        $stmt->bindValue(':query', $query, \PDO::PARAM_STR);
        $stmt->execute();
        // Fetch and return the results
        return $stmt->fetchAll(\PDO::FETCH_OBJ);
    }
```

Update the ***app/controllers/PostController.php***

```
 public function search() {
        // Get the search query from the URL
        $query = $_GET['query'] ?? '';
        // Initialize an empty array for posts
        $posts = [];

        // Only fetch posts if the query is not empty
        if (!empty($query)) {
            $posts = $this->model->searchPosts($query);
        }

        // Load the view with search results
        // The view will receive either the search results or an empty array if the
query was empty
        require BASE_DIR . '/app/views/posts/search-results.php';
    }
```

Update The Routes

public/index.php (GET Method)

```
 // Search For Posts
        'search' => ['controller' => '\app\controllers\PostController', 'method' =>
'search'],
```

Create a View File (*app/views/posts/search-results.php*)

This view will display the posts that have been fetched from the database. Based on the search query.

```
<?php
// Start output buffering to capture content for this page
ob_start();
?>
<!-- search-results.php -->
<div class="container mt-4 mb-5">
    <div class="row justify-content-center">
        <div class="col-md-10">
```

```
            <h2 class="text-center mb-4">Search Results</h2>
            <?php if (empty($posts)): ?>
                <div class="alert alert-info text-center">No results found for your
search.</div>
            <?php else: ?>
                <?php foreach ($posts as $post): ?>
                    <div class="card mb-4">
                        <div class="card-header">
                            <h3 class="card-title"><?php echo htmlspecialchars($post-
>title, ENT_QUOTES); ?></h3>
                        </div>
                        <div class="card-body">
                            <!-- Display post image if available -->
                            <?php if ($post->image): ?>
                                <img src="/images/<?php echo htmlspecialchars($post-
>image, ENT_QUOTES); ?>" alt="Post Image" class="img-fluid mb-3">
                            <?php endif; ?>
                            <p class="card-text"><?php echo htmlspecialchars($post-
>content, ENT_QUOTES); ?></p>
                            <!-- Display category if available -->
                            <p class="card-text">
                                <strong>Category:</strong> <?php echo $post-
>category_name ? htmlspecialchars($post->category_name, ENT_QUOTES) : 'No category';
?>
                            </p>
                        </div>
                        <div class="card-footer text-muted">
                            <a href="/posts/show/<?php echo $post->id; ?>" class="btn
btn-primary">View</a>
                        </div>
                    </div>
                <?php endforeach; ?>
            <?php endif; ?>
        </div>
    </div>
</div>
<?php
// End buffering and store the content in $content
$content = ob_get_clean();

// Include the layout template, assuming it uses $content for page content
include BASE_DIR . '/public/Layouts/layout.php';
?>
```