**Step 5: Database Setup**

* **XAMPP MySQL**: Start the MySQL server through XAMPP and let’s create a new database for our blog.
	+ 1. **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

// 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***](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';

1. **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

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

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

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

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!";