Create your WEB Application with MVC paradigm and OPP – PAP support

Tutorial 6

At this point you should have this two views showing in your web application.

http://localhost/public/login http://localhost/public/register

Time to connect our project to a database.

Next, we will prepare the variables to handle the connection to the database. We will use the MySQL database.

Please create a config.php file inside your core folder and add the php tags like the example:

Copy the code.

Let's debug the code:

DB_NAME -> that's your DB name inside your MYSQL program

DB_USER -> like the name suggests is the user credential

DB_PASSWORD -> access pass to the specific user

DB_TYPE -> Type of database

DB_HOST -> connection host definition

Now we have debug mode statment active to cover some errors when we development the code.

app > core > config.php

```php
<?php
define("WEBSITE_TITLE", 'MY SHOP');

//database name
define('DB_NAME', "lmgdb");
define('DB_USER' , "root");
define('DB_PASS' , "" );
define('DB_TYPE' , "mysql");
define("DB_HOST", "localhost");


define('DEBUG', true);

if(DEBUG){

    ini_set('display_errors', 1);
}else{

    ini_set('display_errors', 0);
}
```

Now we need to refer this new file inside the init.php app folder:

At this point you have the first configurations to database done.

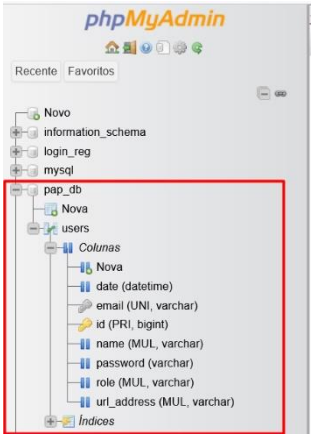
```php
<?php

include "../app/core/config.php";
include "../app/core/controller.php";
include "../app/core/database.php";
include "../app/core/functions.php";
include "../app/core/app.php";
```

Go to your phpMyAdmin site, and create a database with the name that you choose before:

Then, you will create a table users to test the connection and to start to create you login and register system from your application:

Assigment:

- Create table users;
- Create some fields;
- Put ID with PK and AI at this moment, and email with Unique field;
- You can add the fields that you want (be careful with your choices).



Attention, in my example, I have a 'role' field and a 'url_address' field. This 'url_address' is mandatory. You can give it any name you want but stay alert, this field will be essential for handling connection sessions from the users and other functionalities.

Prepare connection to database:

At this point, you should have your first table configured.

Next, we will proceed to establish the connection between the application and the database. Within the 'Database' file in your core folder, you will insert the following code: This code serves as the mechanism to establish the connection between both components.

```php
<?php
class Database
{
    public static $con;


    public function __construct()
    {
        try{

            $string = DB_TYPE . ":host=" . DB_HOST . ";dbname=" . DB_NAME;
            //echo($string);
            self::$con = new PDO($string , DB_USER , DB_PASS);

        }
        catch (PDOException $e){

            die($e->getMessage());

        }

    }
```

It indicates that we will attempt to connect to the specified string using the predefined parameters above (host and dbname). Subsequently, employing PDO with the defined scope, please endeavor to establish the connection.

Next to the construct, you will put another function:

The getInstance() method, when used within the context of managing a database connection (or any other resource), is typically implemented to ensure that only one instance of the connection object exists throughout the lifetime of the application.

```php
public static function getInstance()
{
    if(self::$con)
    {
        return self::$con;
    }

    //self::$con = new self();
    return $instance = new self();
}
```

By restricting the creation of new instances to just one, you avoid the overhead of multiple connections to the database, which can improve performance and resource utilization. Additionally, it helps maintain consistency and avoids potential issues that may arise from having multiple independent instances of the connection object.

Next you will create a function to read data from your DB:

```php
//read data from database
public function read($query, $data = array())
{
    $stm = self::$con->prepare($query);
    $result = $stm->execute($data);

    if($result)
    {
        $data = $stm->fetchAll(PDO::FETCH_OBJ);
        if(is_array($data) && count($data) > 0)
        {
            return $data;
        }
    }

    return false;
}
```

This code defines a method named read() within a class. This method is used for executing SELECT queries on a database and retrieving data.

Overall, this method is used to read data from the database based on the provided SQL query and return the result as an array of objects. If no data is found or if there's an error in executing the query, it returns false.

Finally, you will create a function to write in your DB:

```php
//write to database
public function write($query, $data = array())
{
    $stm = self::$con->prepare($query);
    $result = $stm->execute($data);

    if($result)
    {
        return true;
    }

    return false;
}
}
```

This code defines a method named write() within a class. This method is used for executing INSERT, UPDATE, DELETE, or other SQL queries that modify the database.

Overall, this method is used to write data to the database based on the provided SQL query and return true if the operation was successful, or false otherwise.

This is all that you need for now, to create interaction between your application and database.

Next you will start to construct you script to use this connection and insert your first data to DB.

END Tutorial 6

Reference: **PHP Ecommerce website development | Create the app class | MVC OOP - Quick programming**

Template: NICEADMIN, design by BOOTSTRAP