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

Tutorial 7

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

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

Follow the example below:

If you don't create a register page and his controller, please create it!

Do the same to your login.

Inside the controller register, put this code:

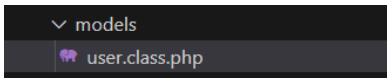
So, this code will listen a POST call and grab all data from the fields that you identify with the tag name and create a json file to send the data to database.

```php
<?php

class Register extends Controller
{
    //Public default metodo index, mesmo que o utilizador coloque ou não qualquer URL, o Index vai sempre correr
    public function index()
    {
        $this->title = 'User - Register';

        if($_SERVER['REQUEST_METHOD'] == "POST")
        {
            //show($_POST);

            $User = $this->load_model("User");
            $User->signup($_POST);
        }


        //Direção de onde esta a view que vai carregar
        $this->view("register");

    }
}
```

If you refer, you have a new function call `load_model()` in the code above, but you haven't created that function yet. Go to your controller file inside the core folder and add this new function next to the `view` function. Here is the code: This code will attempt to retrieve a model file from the model folder to send data to the database.

```php
//In this method is where we call the route and we can also pass some data to the view
public function load_model($model)
{

    if (file_exists("../app/models/" . strtolower($model) . ".class.php"))
    {
        include "../app/models/" . strtolower($model) . ".class.php";
        return $a = new $model();
    }
    return false;
}
```

Let's go to model folder:



Create a file a name it like the example:

Now you will put the follow code example:

```php
Class User
{

    private $error = "";

    function SignUp($POST)
    {

        $data = array();
        $db = Database::getInstance();

        $data['name']       = trim($POST['name']);
        //$data['username']   = trim($POST['username']);
        $data['email']      = trim($POST['email']);
        $data['password']   = trim($POST['password']);
        $password2          = trim($POST['password2']);
```

Now, let's debug this code for parts.

First you have a empty private variable to handler the error messages in the code.

Next you have a signup function that will handle the process to the register from the user.

This declares a function named SignUp which takes one parameter $POST. It likely receives data submitted via POST method from a form.

The empty array named $data will be used to store user data.

The getInstance()is the function that you do in the previous, that will initialized a database connection using a method named from a class called Database.

The $POST extracts user input from the array and assigns it to corresponding keys in the $data array after trimming whitespace. In this example, you seems that you capture the user's name, email, password, and a confirmation password.

Next to code above, inside the same function we will do some performing validation checks on user input data, particularly for email, name, and password fields. Let's break down each part:

This block checks if the email field is empty or if the email format does not match a regular expression pattern for a valid email address. If either condition is true, an error message is appended to the $error property of the current object ($this->error).

```php
//validação de email
if (empty($data['email']) || !preg_match("/^([a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,})$/", $data['email'])) {
    $this->error .= "Please enter a valid email <br>";
}
```

(e.i. you can search for this sequential numbers and letters in web)

This block checks if the name field is empty or if the name contains characters other than letters. If either condition is true, an error message is appended to the $error property.

```php
//validação do nome
if (empty($data['name']) || !preg_match("/^[a-zA-Z]+$/", $data['name'])) {
    $this->error .= "Please enter a valid name <br>";
}
```

This block checks if the password field matches the confirmation password ($password2). If they do not match, an error message is appended to the $error property.

```php
//verificar se a password é igual a password2
if ($data['password'] !== $password2) {
    $this->error .= "password do not match <br>";
}
```

This block checks if the password length is less than 4 characters. If it is, an error message is appended to the $error property.

```php
//verificar se a password tem no minimo 4 caracteres
if (strlen($data['password']) < 4) {
    $this->error .= "Password must be at last 4 characters long! <br>";
}
```

This code segment is used to check if the email provided by the user already exists in the database.

```php
//ver se o email ja existe na base de dados
//$sql = "select * from users where email = : email limite 1";
$sql = "select * from users where email = :email limit 1";
$arr['email'] = $data['email'];

$check= $db->read($sql,$arr);
if(is_array($check)){

    $this->error .= "That email already exits! <br>";

}
```

This SQL query selects all columns from the users table where the email matches the provided email. The LIMIT 1 clause ensures that only one result is returned.

This executes the SQL query using the read() method of the database connection ($db). The $sql variable contains the SQL query, and $arr contains the data to bind to the placeholders.

This checks if the result returned by the database query is an array. If it is, it means that the email already exists in the database, and an error message is appended to the $error property indicating that the email already exists.

Overall, this code segment performs a database query to check if the provided email already exists in the database and sets an error message if it does.

Now you need to create a function to generate a random string to assign a user account. If you remember when you create a database you create a user_address variable.

This function will fill that field when the user made is registration.

Below the function register you will put this new one:

```php
// Private method to generate a random string
private function get_random_string_max($length)
{
    $characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
    $random_string = '';

    for ($i = 0; $i < $length; $i++) {
        $random_string .= $characters[rand(0, strlen($characters) - 1)];
    }

    return $random_string;
}
```

Will be private, because it is only for use inside this class.

Next, in continuation from the signup function, you will put this next code. This line of code assigns a randomly generated string to the $data['url_address'] variable.

```php
//Generate a random string with numbers and letters to assign to user
// This will be use like content access token
$data['url_address'] = $this->get_random_string_max(60);
```

Next, it is not necessary, but the good practices made to it, so we need to check if the url_address its not in use in DB. So next make that validation.

```php
//verificar as URL
$arr = false;
$sql = "select * from users where url_address = :url_address limit 1";
$arr['url_address'] = $data['url_address'];
$check = $db->read($sql, $arr);
if (is_array($check)) {
    $data['url_address'] = $this->get_random_string_max(60);
}
```

This checks if the result returned by the database query is an array. If it is, it means that the URL address already exists in the database. In this case, a new random URL address is generated using the get_random_string_max(60) method and assigned to $data['url_address'].

Finally, we go to serialize all the fields and send it to DB.

```php
    if ($this->error == "") {
        // Para testar com a minha base de dados
        $data['role'] = "costumer";
        $data['date'] = date("Y-m-d H:i:s");

        //Hash a palavra passe
        $data['password'] = hash('sha1', $data['password']);
        //Query to insert values into DB table users
        $query = "insert into users (url_address,name,email,password,date,role) values(:url_address,:name,:email,:password,:date,:role)";
        //Call method insert
        $result = $db->write($query,$data);

       //Check the result
       //If ok, send th user to new location and disconet the db connection
        if ($result) {
            header("location: " . ROOT . "login");
            die;
        }
    }
    // If not give the error
    $_SESSION['error'] = $this->error;
}
```

Overall, this code segment completes the user sign-up process by inserting validated user data into the database and handling success or failure accordingly. Additionally, it manages error messages to provide feedback to the user if needed.

You can also to do the insert like this way.

```php
//Query to insert values into DB table users
//$query = "insert into users (url_address,name,email,password,date,role) values(:url_address,:name,:email,:password,:date,:role)";

// Query to insert values into DB table users
$query = "INSERT INTO users (url_address, name, email, password, date, role)
VALUES ('$data[url_address]', '$data[name]', '$data[email]', '$data[password]', '$data[date]', 'costumer')";
```

Next tutorial we will see the login.

END Tutorial 7

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

Template: NICEADMIN, design by BOOTSTRAP