

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

### Tutorial 8

At this point, you already have some data in your database. Now you will make some updates to your register view.

Imagine the user enters something incorrectly, and the form is unable to submit due to errors, the user needs to know where the errors are.

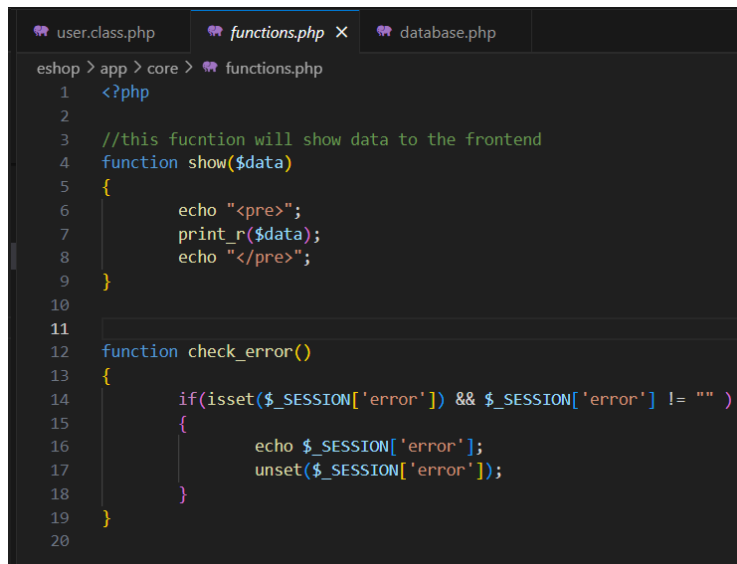
If you remember, the validation checks are already configured in your user.class.php file, in your register function. So, in that case, you will put the following code in your register view:

So, in that case, you will put the follow code in your register view:

```
<!-- Email span validation -->
<span style="color:red"><?php check_error() ?></span>
```

You put this code, where you want to show to the user, the inserted form data error. After that you can make some upgrades with CSS, to change the color of the span. See some examples on the web, or check if your template is already with this option.

Next you need to construct the error function, and you will put it inside file functions, following the show function.



```
eshop > app > core > functions.php
1  <?php
2
3  //this function will show data to the frontend
4  function show($data)
5  {
6      echo "<pre>";
7      print_r($data);
8      echo "</pre>";
9  }
10
11
12  function check_error()
13  {
14      if(isset($_SESSION['error']) && $_SESSION['error'] != "" )
15      {
16          echo $_SESSION['error'];
17          unset($_SESSION['error']);
18      }
19  }
20
```

Now test it!! Try inserting some error in the form and click submit to see the error span in action.

Now, imagine, every time the user inputs their data in form and click submit the view reloads, the fields become empty, because a reload are made it. To avoid this action you need to add this code, where the fields are.

```
<div class="col-12">
  <label for="yourEmail" class="form-label">Your Email</label>
  <input type="email" name="email" value="<?= isset($_POST['email']) ? $_POST['email'] : ''; ?>"
  <div class="invalid-feedback">Please enter a valid Email address!</div>
</div>
```

You only put this in a common field, like name field, email field, etc... Don't put this in password field, it is not a good practice. You need to force the user write the password every time, to maintain security specifications.

If all works well so far, let's begin configure the user login.

The login function is very similar to the register function, but instead of writing to the database you will read.

Follow the Signup() function you will put this new function:

```
function Login($POST)
{
    $data = array();
    $db = Database::getInstance();

    $data['email'] = trim($POST['email']);
    $data['password'] = trim($POST['password']);

    //validation for you write an email corretly
    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>";
    }

    //validation for minimal password lenght
    if(strlen($data['password']) < 4 )
    {
        $this->error .= "Password must be at last 4 characters long! <br>";
    }
}
```

In this first part, you get the instance connection database and get the post from the email, and password.

Note, if you define the email as your principal key to authenticate the user you will use that field, but if you use a username or other, you need to change the code to that field.

Now let's debug the code:

Like the signup we have a validation email insert, and a minimal password characters to fill.

After this we will make the call to our database:

```
if($this->error == "")
{
    //confirm
    $data['password'] = hash('sha1', $data['password']);

    $sql = "select * from users where email = :email && password = :password limit 1";
    $arr['email'] = $data['email'];
    $result= $db->read($sql,$data);

    if(is_array($result))
    {
        $_SESSION['user_url'] = $result[0]->url_address;
        header("Location: " . ROOT . "home");
        die;
    }
    $this->error .= "Wrong email or password! <br>";
}

$_SESSION['error'] = $this->error;
}
```

Ok, this condition give us, check if there are no errors before proceeding with the login attempt. If there are no errors, the login process continues.

The user's password provided in `$data['password']` is securely hashed using the system cryptography.

Next, the SQL query selects user data from the database table users where the email matches the one provided by the user. The LIMIT 1 clause ensures that only one row is returned. The email value is passed as a parameter 'email' to prevent SQL injection. The read method is used to execute the query with the provided parameters.

If the user authentication is successful, the user's URL address (or any relevant session data) is stored in the `$_SESSION` array for future reference, and the user is redirected to the home page (home) using the header function. The die statement ensures that no further code is executed after the redirection.

If the authentication fails (either the email doesn't exist or the password is incorrect), an error message indicating "Wrong email or password!" is appended to the `$this->error` variable.

ASSIGNMENT: Look at your register controller and try to complete the login. When everything is done, go to your login view and put the validation check for errors, like before, and test if the login works well. If everything is okay, you should be directed to your homepage.

<http://localhost/public/login>

## END Tutorial 8

Reference: **PHP Ecommerce website development | Create user login | MVC OOP - Quick programming**

Template: **NICEADMIN**, design by **BOOTSTRAP**