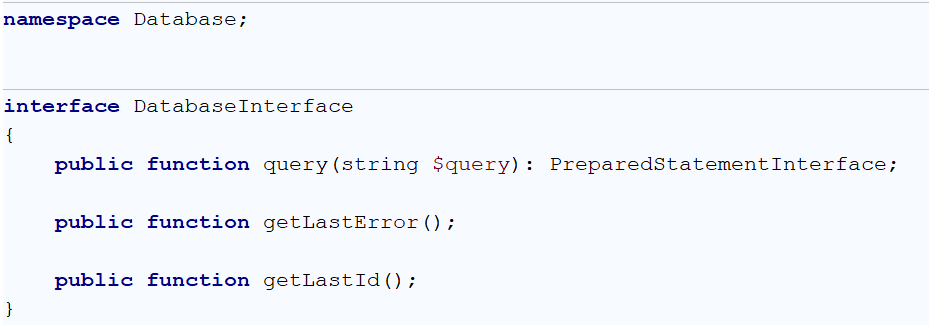
## Session Handling and Authentication

## Abstracting Database

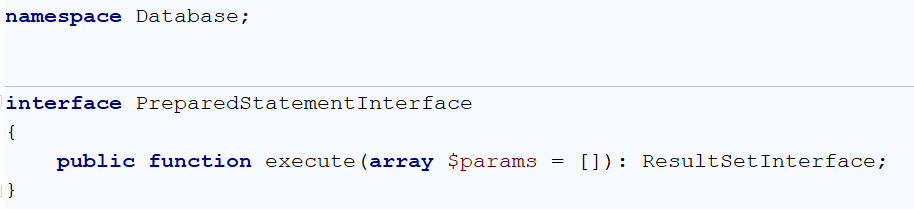
We will **not directly use the PDO object** in our object graph, because **of lack of abstraction** (PDO is concrete object).

As we know from PDO we need first to prepare the query, execute it and then fetch. These three steps will be abstracted in three different objects. Let’s create interfaces for all of them:

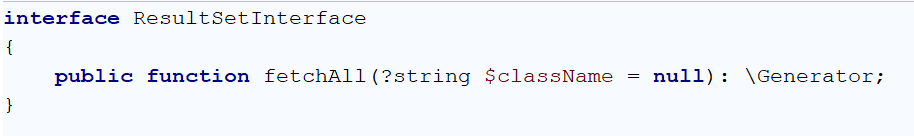
### Query Abstraction



### Prepared Statement Abstraction



### Fetching Abstraction

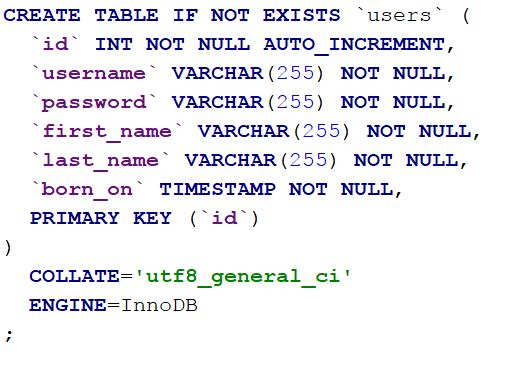


You need to create implementations that:

1. **Starts a query** and **returns a Statement** with the PDOStatement information
2. **Executes the statement** and returns **ResultSet** (the executed statement which can be **later fetched**)

### Create Schema

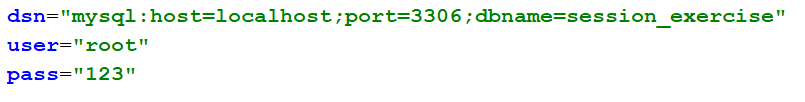
We will need database schema to create our application. Start with table users, that have username, password, first and last name, also a birthday. Add the CREATE TABLE script in a file named export.sql:



### Configure Database

We need our app to know the connection information. Let’s store it in an .ini configuration file (remember how php’s configuration file was named php.ini?). Create a folder “Config” and put a file “db.ini”:





# Lab Exercise: Creating a Blog with PHP and MySQL

This document holds a step-by-step tutorial on **how to create a Blog system** based on **PHP**, **Apache** and **MySQL** with a simple custom **MVC framework**. This lab exercise is part of the ["PHP Web Development Basics" Course @ Software University](https://softuni.bg/trainings/2163/php-web-development-basics-september2018).

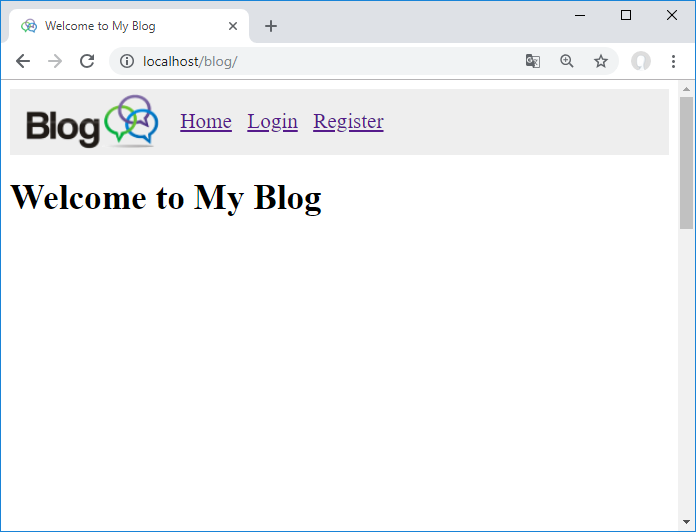
## Project Specification

Design and implement a **“Blog” Web application** in PHP + MySQL. Implement the following functionality:

* **Home**
  + Show **[Login]** and **[Register]** buttons (when no user is logged in).
* **Login**
  + Login in the blog with existing account (username + password).
  + Show a success message after login or error message in case of problem.
* **Register**
  + Register a new user in the MySQL database (by username + password + full name).
  + Show a success message after registration or error message in case of problem.
* **Logout**
  + Logout the current user.
  + This [Logout] button is available after successful login only.
* **View All Users**
  + Logged in users should be able to **view** all users (username + full name) in a table.

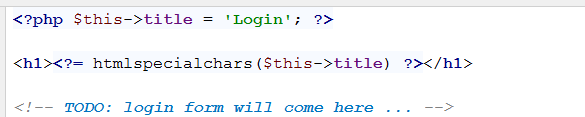
## Screenshots

This is how your HTML pages may look like.

The **home page** looks like this: 

### User Login

Now we need to create the login page. We start from the login form. Go to the **views/users/login.php**. It should contain the following:



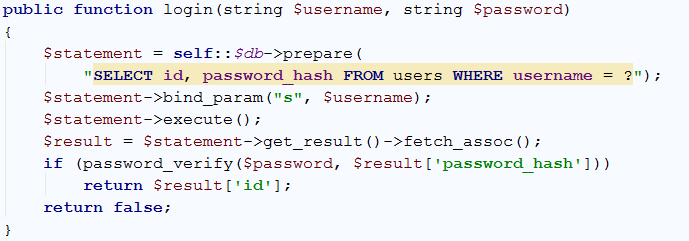
Write a simple form, that has 2 textboxes and a button:



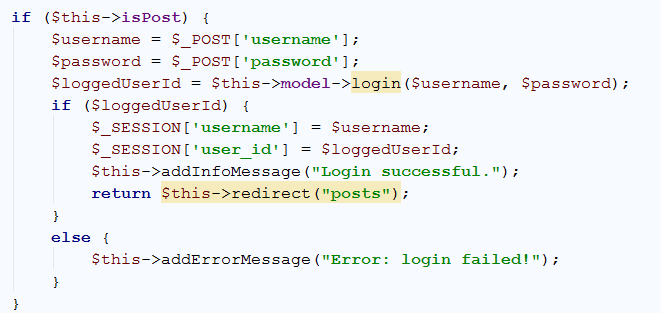
Write a simple form, that has 2 textboxes and a button:



Now we need to create the logic behind the login. Let’s start by going to the UsersModel. Right now we have this:



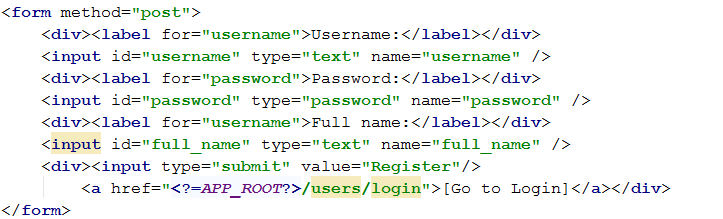
Finally, we need to call that function from the UsersController. In the empty login function we need to write the following code:



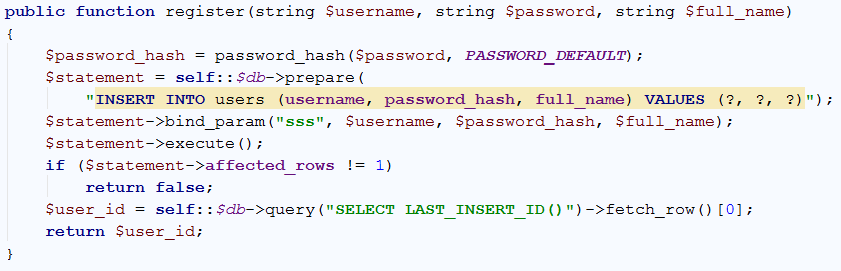
The login form should work now. It will redirect you to an error message when you successfully login, but if go back to the home page, you will see the success message.

### User Registration

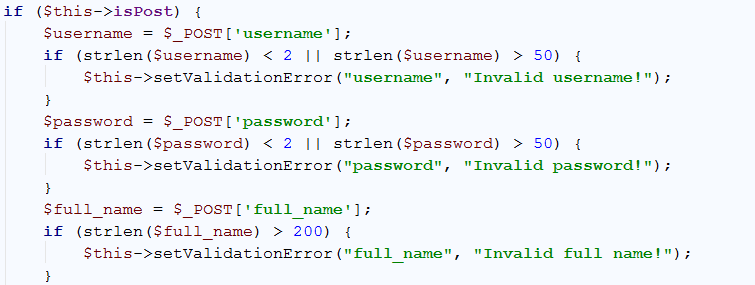
Right now we can login, but we cannot register. We should fix this issue. Let’s start by creating the form in **users/register.php**. Creating a simple form like this will work like a charm:



We should go to the UsersModel and create another function, that will create a query that inserts new row to our users table:



Now that we have the model and the view, the only thing left is to tell the controller what to do. Go to the UsersController and add that code to the empty **register** **function**:



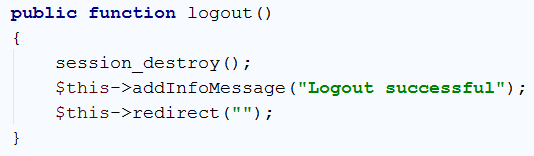
This code validates our input, and if the input is invalid, throws an error message. After the validations, we should write the following code:



Our registration is ready now. After successful registration this error might come up, but don’t worry, we will fix it soon.

### User Logout

Aside from Registration and Login logic, we should provide the user with the ability to log out and use a different username. Go to logout() function in the UsersController, and write the following code.

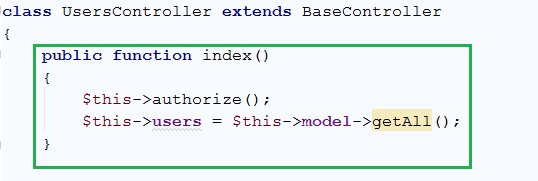


Essentially, the “**login**” logic is done by saving data about the user, which is being logged in, in the **current session**. That data is the only thing that indicates that there is a currently logged in user. If we are to delete that data, there will practically be no logged in user. For that reason, we just clear the session, with the session\_destroy() function, which works as the main “**logout**” logic.

After that, we just print an information message, and we redirect the page to the home.

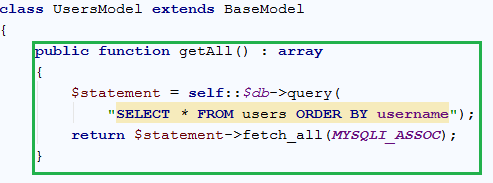
### List Users

The last thing we need to do is list all the currently registered users.  
Let’s go to the UsersController class and add an index() function, which will implement the logic for the home users page.



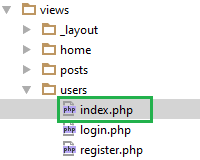
Similar to the posts, it just gets all the users from the Database, using the User Model, and stores them. It also authorizes the client, before all else.

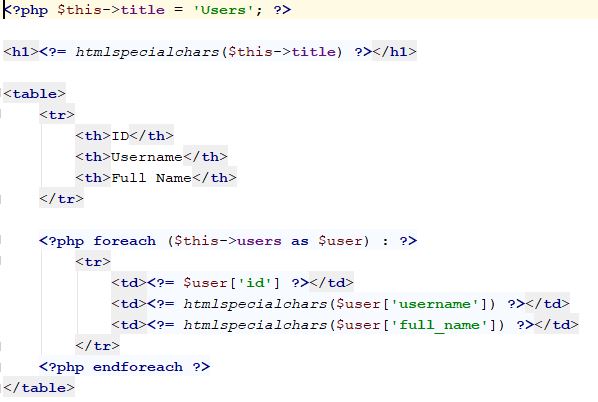
Next, we go to the UsersModel class, because we need a function called getAll().



The function fetches all the users from the Database and orders them in alphabetical order.

The last thing we need to do is to present the users data to the user. Go to the users views, and create a file called index.php. We create a simple table, in which we will present data about our users. After that we just make a foreach loop, to loop over all the users we extracted with the controller.





This was the last functionality we needed to implement. If you have done everything by the steps, you should see the following results: