**W5 - Assignment**

Charlie Farris

The University of Arizona Global Campus

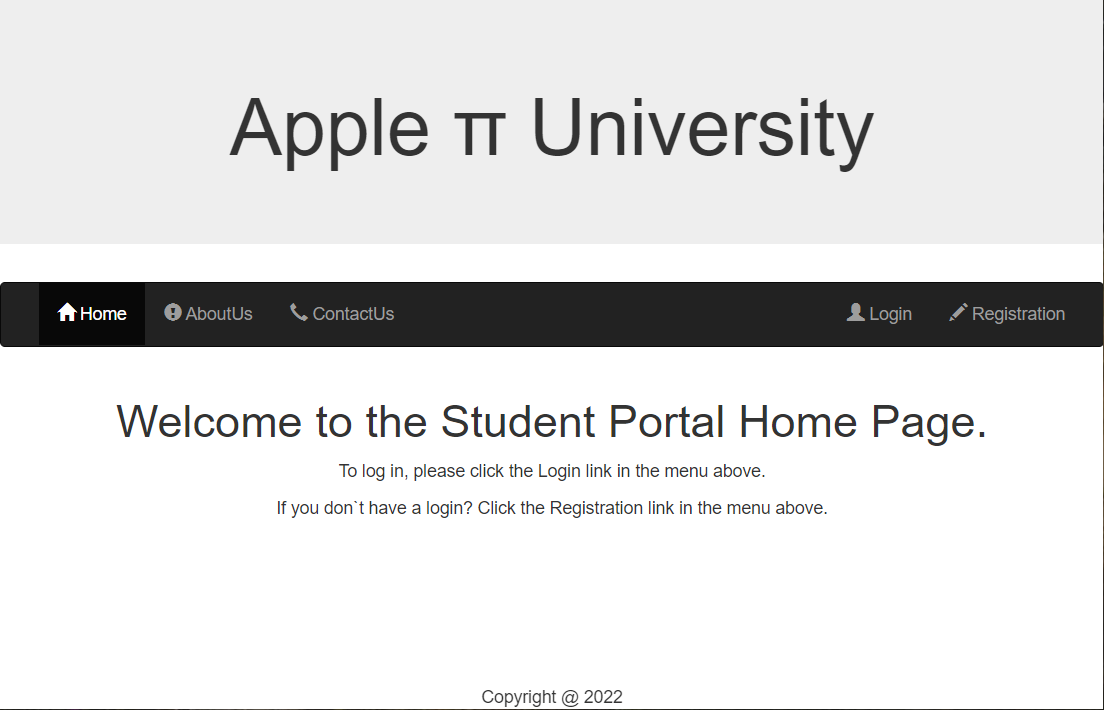
CST 499 Software Development

Dr. Charmelia Butler

11/17/2022

**Landing, Login, and Registration Page Development**

The Apple Pi Student Portal is designed to allow end users to access the home page and login to see their student information if they have already registered. If they have not registered, they also have the opportunity to do such via the registration link on the home page. Once a user is done reviewing their profile, they can log out of the site via a link that appears after they log in. Using this functionality secures their data from being seen by others. To build this site, I have taken advantage of the Employee portal CST310 project I developed (why re-invent the wheel) and the following tools: XAMP, PHP, HTML5, and MySQL.



XAMPP is a simple, lightweight Apache distribution used by developers to create a local web server for testing code. PHP programs run within a web browser. To run a PHP file using XAMP you first need to create a PHP program. Then save the php file to the htdocs folder within the xampp folder. (For this project, I actually saved the php in a subdirectory in the htdocs folder so that I could keep the projects we create straight.) Then execute the XAMPP server and start Apache and MySQL. Next, open your web browser and type [http://localhost/<programName>.php](http://localhost/%3cprogramName%3e.php) into your browser window. At this point you should see the output of your PHP code.

I created the MySQL database studentportal using phpMyAdmin. I used SQL DDL commands like CREATE TABLE and ALTER TABLE to generate the tables in my database. I then used SQL DML commands INSERT and UPDATE to populate those tables with some default information. I then created a database user, "test\_tester", and granted that user privileges on the studentportal database. The "test\_tester" and password are what I use to access the database from the PHP code. I selected the PHP Data Objects (PDO) extension for creating the database connection class Database.php (Figure 21) file. This choice is because PDO offers a consistent interface regardless of which database-specific PDO driver that is used. ( I am using the MySQL driver for PDO.) When the class is instantiated, it checks to ensure that the PDO MySQL driver is loaded before connecting to the database. The class then establishes a connection to the database using the database type, name, user, and password that are stored in the code. (If I have time these passwords will be moved out to a config file for better protection.) Once the connection is established, it is returned back to the calling function for use.

The first page to be developed/updated was the registration page, as seen in Figure 1. This was done so that an end-user could save their information to the database. The Registration page needed to have:

* Labels and fields to capture the user input
* Process that information into an SQL statement
* Using the executeQuery function of Database.php, post the information to the database
* If there is an error in posting, notify the end user, reload the Registration page, and prepopulate the fields with the information already entered (except for the password)

The first step was to create a form in the registration.php with the method of post. (At first, I was going to post to the same page, but to help keep things straight in my mind, I used another page called form\_processor.) Using an unordered list, I created labels and input fields in the registration.php page (figure 2) with the following format:

<li>

<label for="email">E-mail:</label>

<input type="email" id="email" name="user\_email" value="<?php echo($uemail); ?>">

</li>

Creating the label for and setting the input id to the same identifier allowed the two to be connected as seen in Figure 1. Then I added the submit button so that the end user could post this information to the form\_processor.php seen in Figure 3. Here I used the super global $\_POST to create my SQL statement. Using the executeQuery function of Database.php I then submit the transaction to the database. If the database update was successful, I informed the user and then sent the user to the Login page. If it was not, I informed the user and sent them back to the registration page along with the values that were entered so that I could prepopulate the fields with their information. Although I can see the logic should work, I am having problems causing a failure. I think this has to do with XAMPP being in a development mode. This would allow the partial updates of good information and set the bad values to null or 0 in the database. I will need to work more in this area. I should also add logic in the php to validate the fields and handle it the same way that I am trying to handle a bad database update. The update to the data base can be seen in Figure 4 where I give a before Figure 1 is submitted and an after figure 1 is submitted.

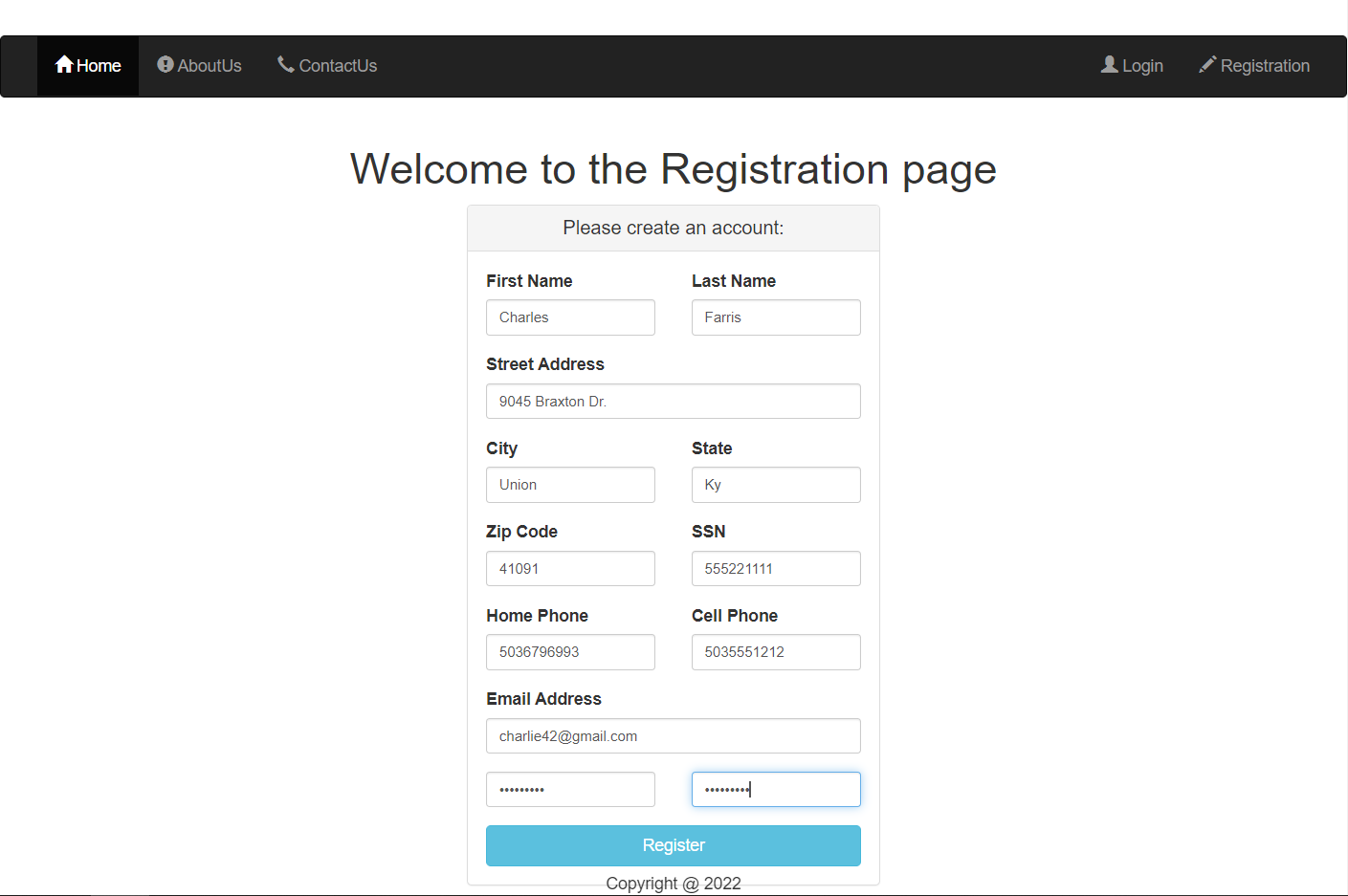


Figure 1: Registration page layout

Figure 2: Registration page PHP source code

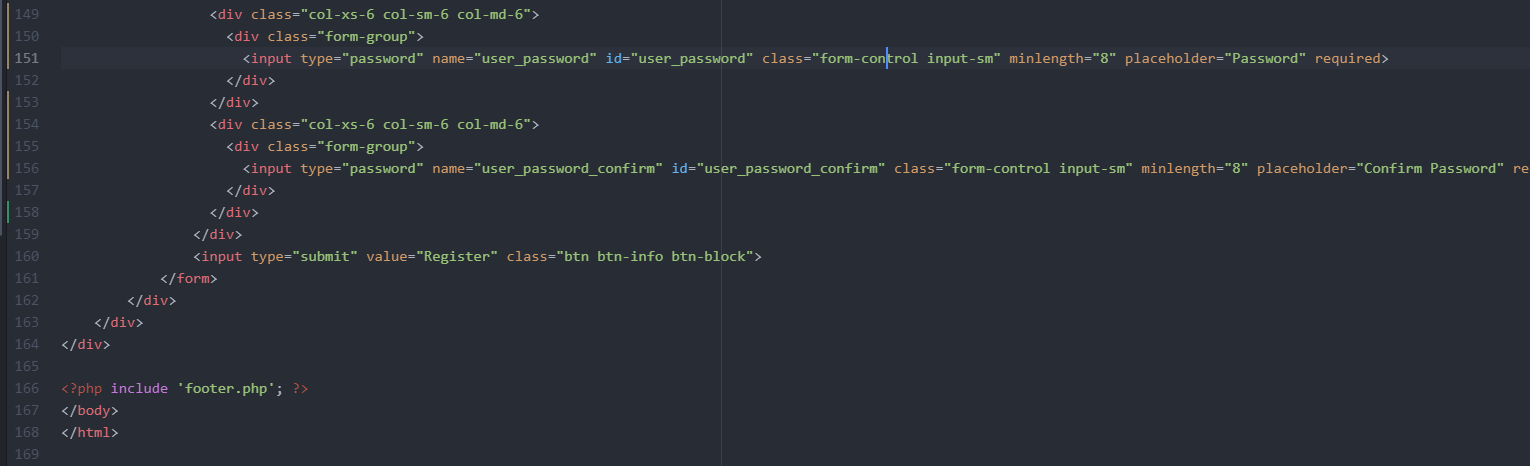
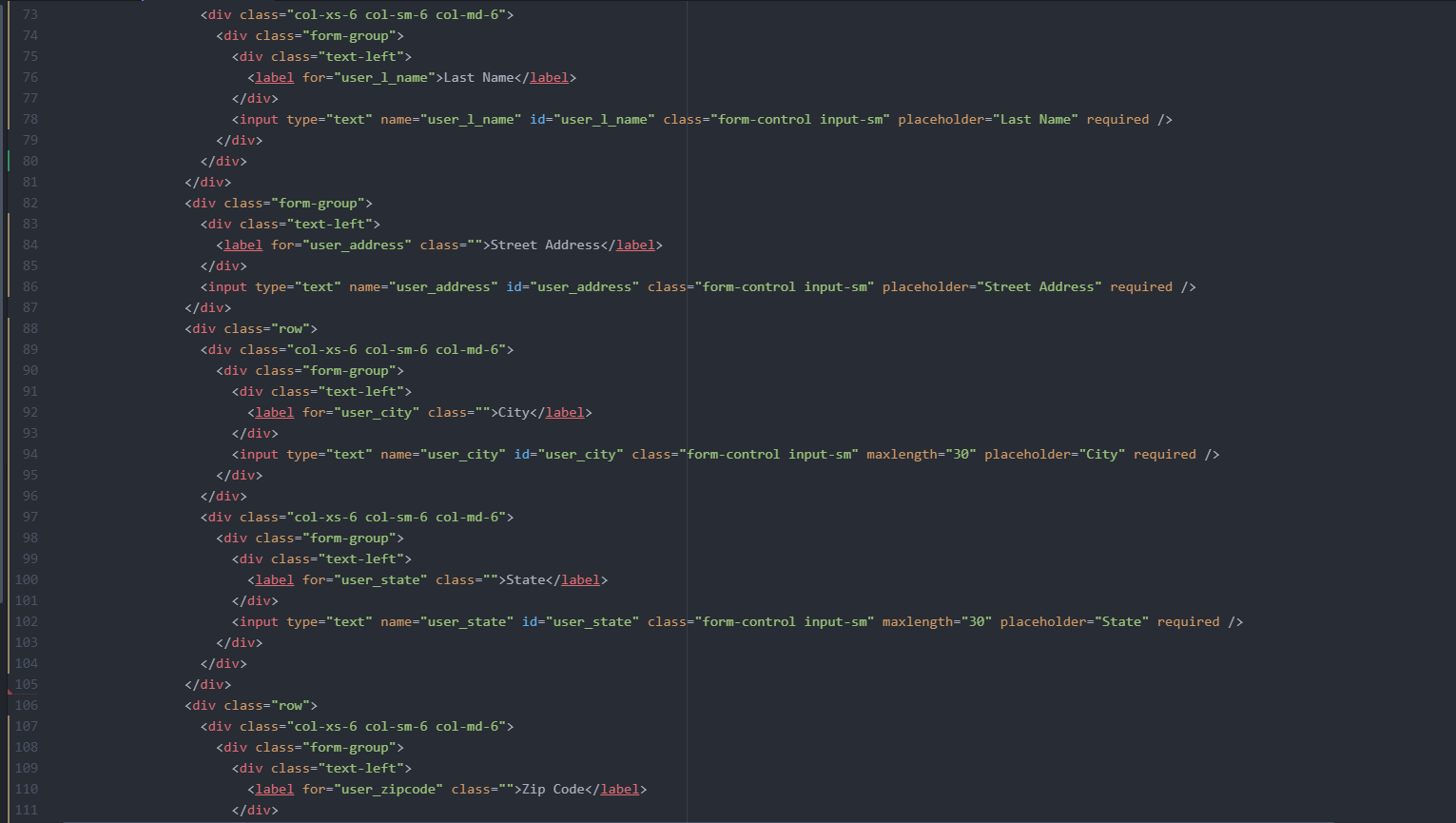
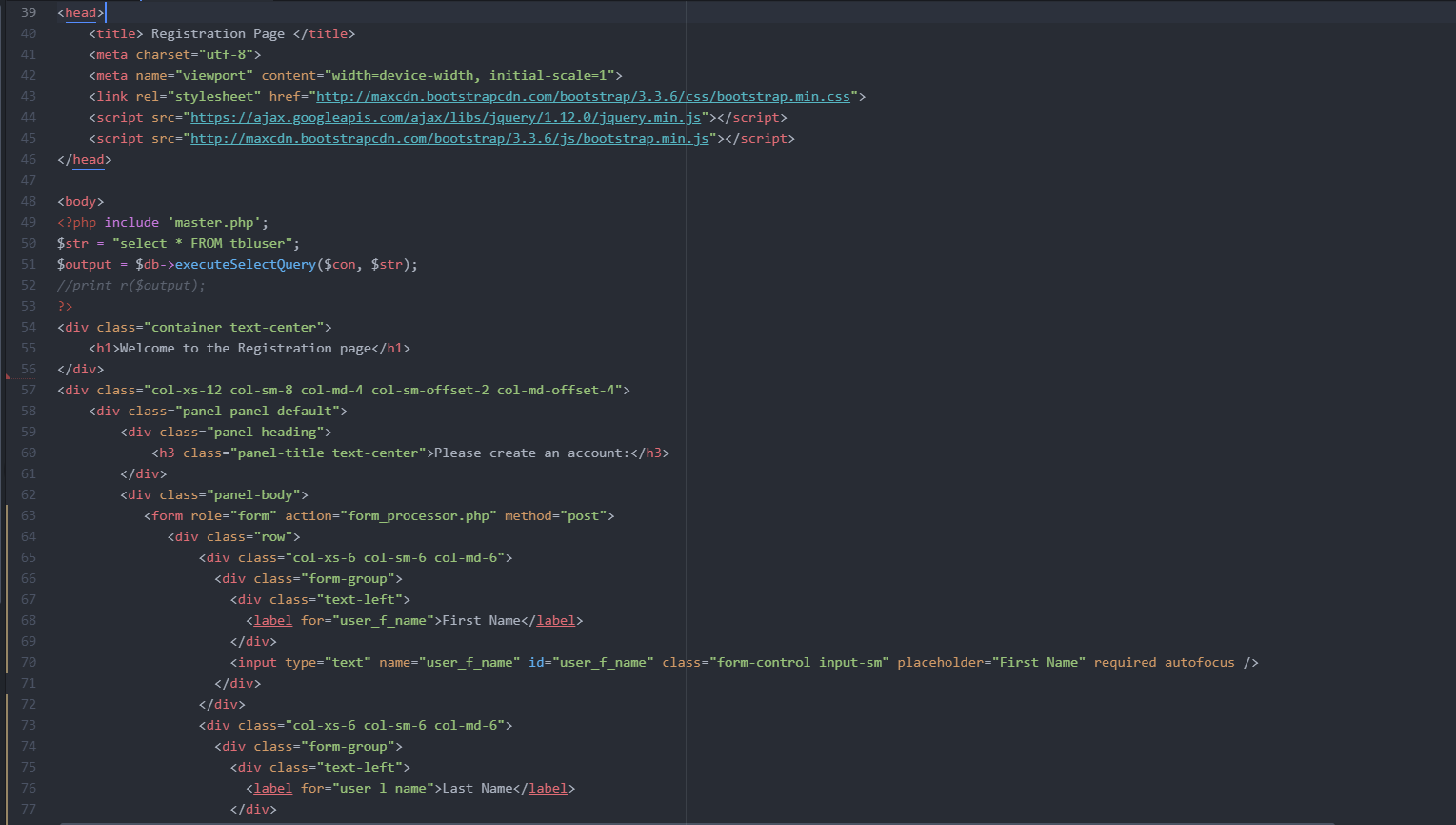
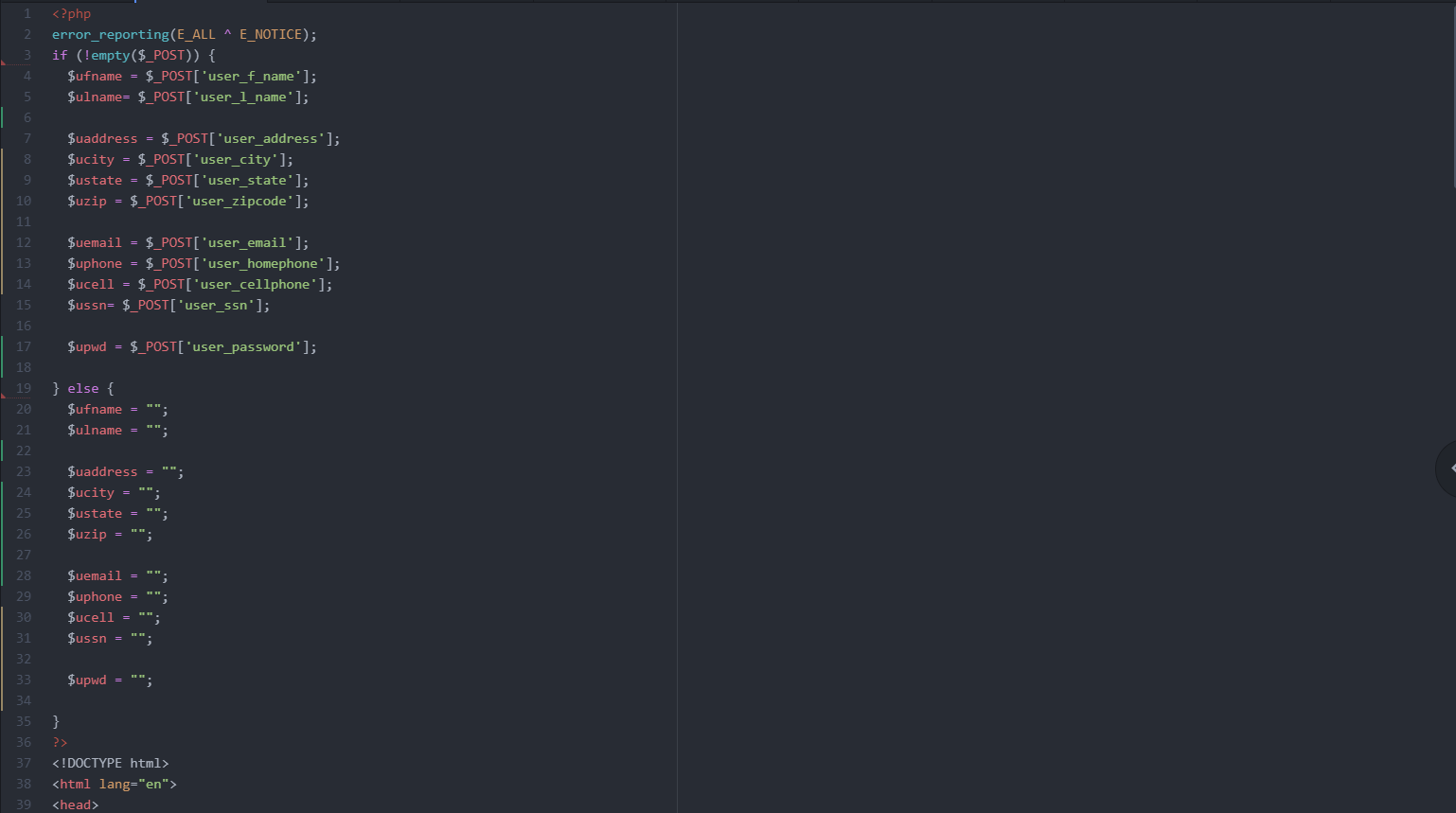


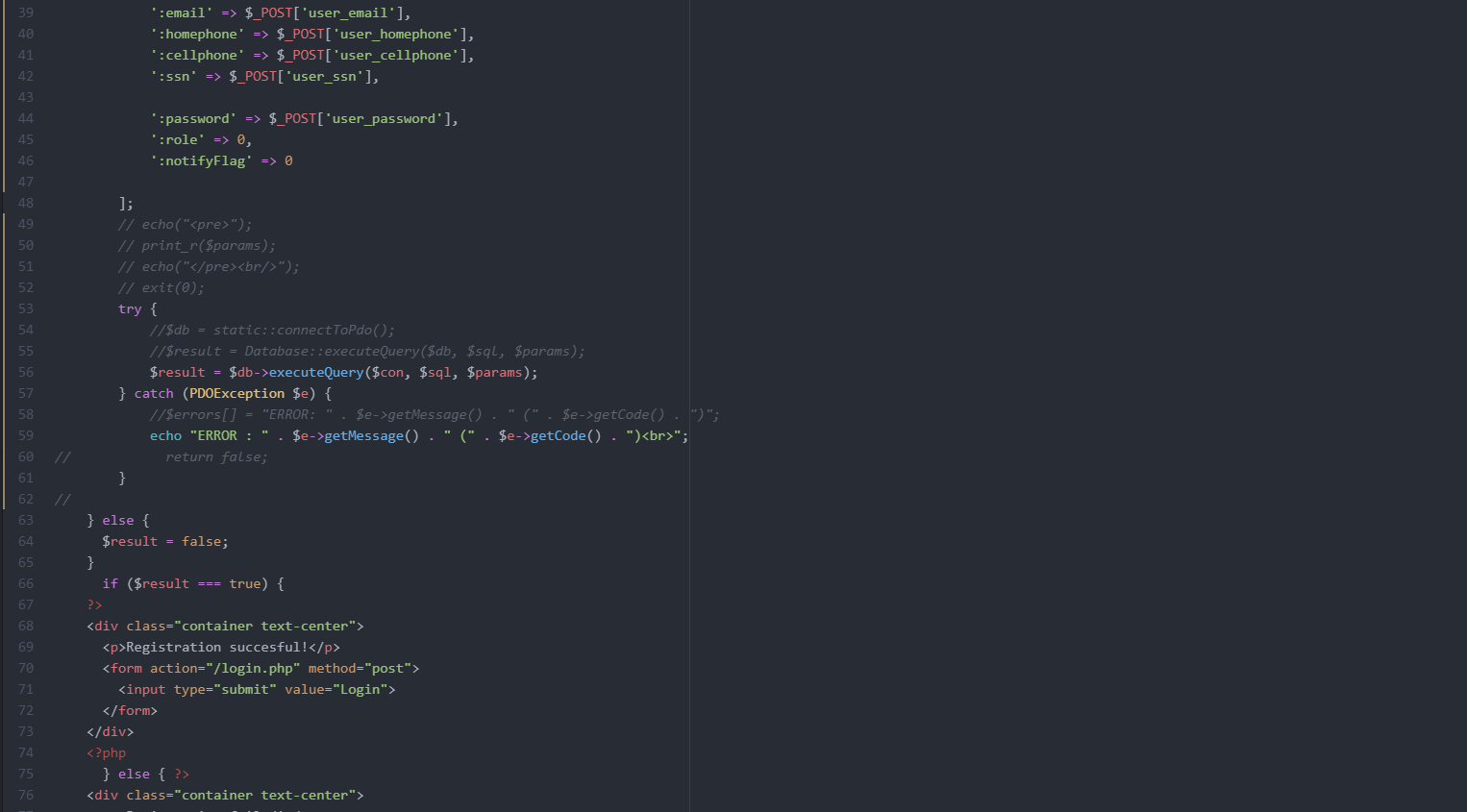
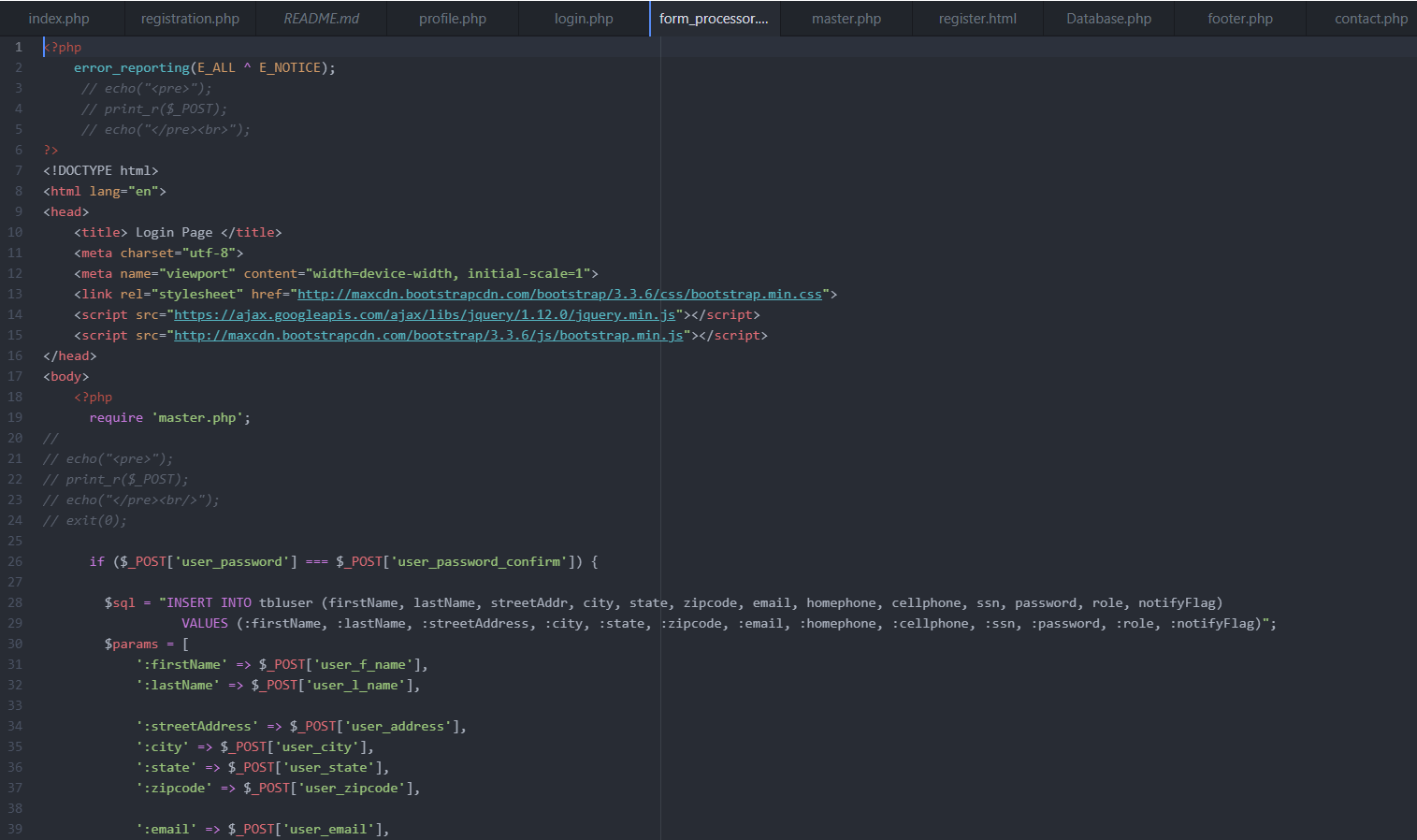
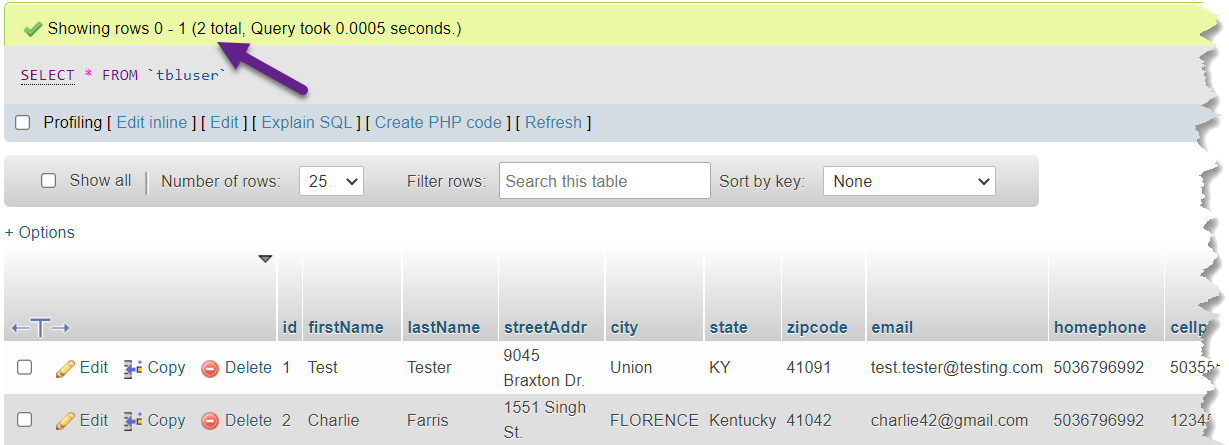
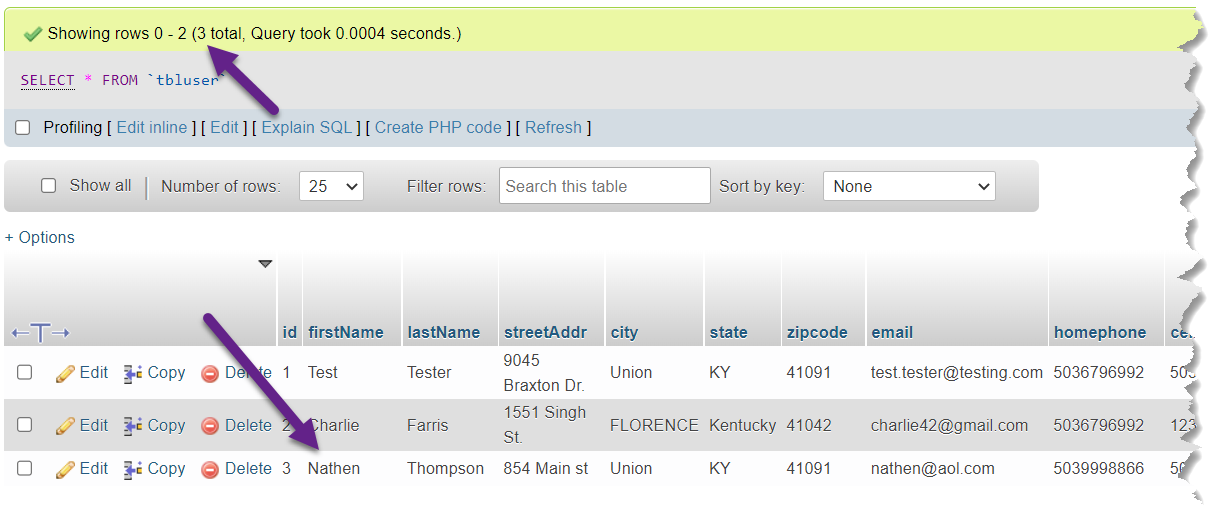
Figure 3: Registration form\_processor PHP source code

Figure 4: MySQL table before & after saving the user information in the database





After creating the Registration page, the Login page layout seen in Figure 5 was a simple matter. The trick was learning about how to POST to the same page and completing the validations check. The first step was to take the user input and use my executeSelectQuery function. I then took the results of that query and confirmed is $\_POST[‘password’] was the same as the password that the user entered in. If it was, I gave the end user the all good message and then set $\_SESSION to $results. Did you notice that I was missing the [0]? I didn’t either for a while, but finally figured it out and updated my code to $\_SESSION = $results[0] as seen in on line 27 on the Login.php. I also set the session username to $results[0]['firstName'] so that when I refreshed the header and went back to index.php and the home page the user name would appear correctly.

Figure 5: Login page layout

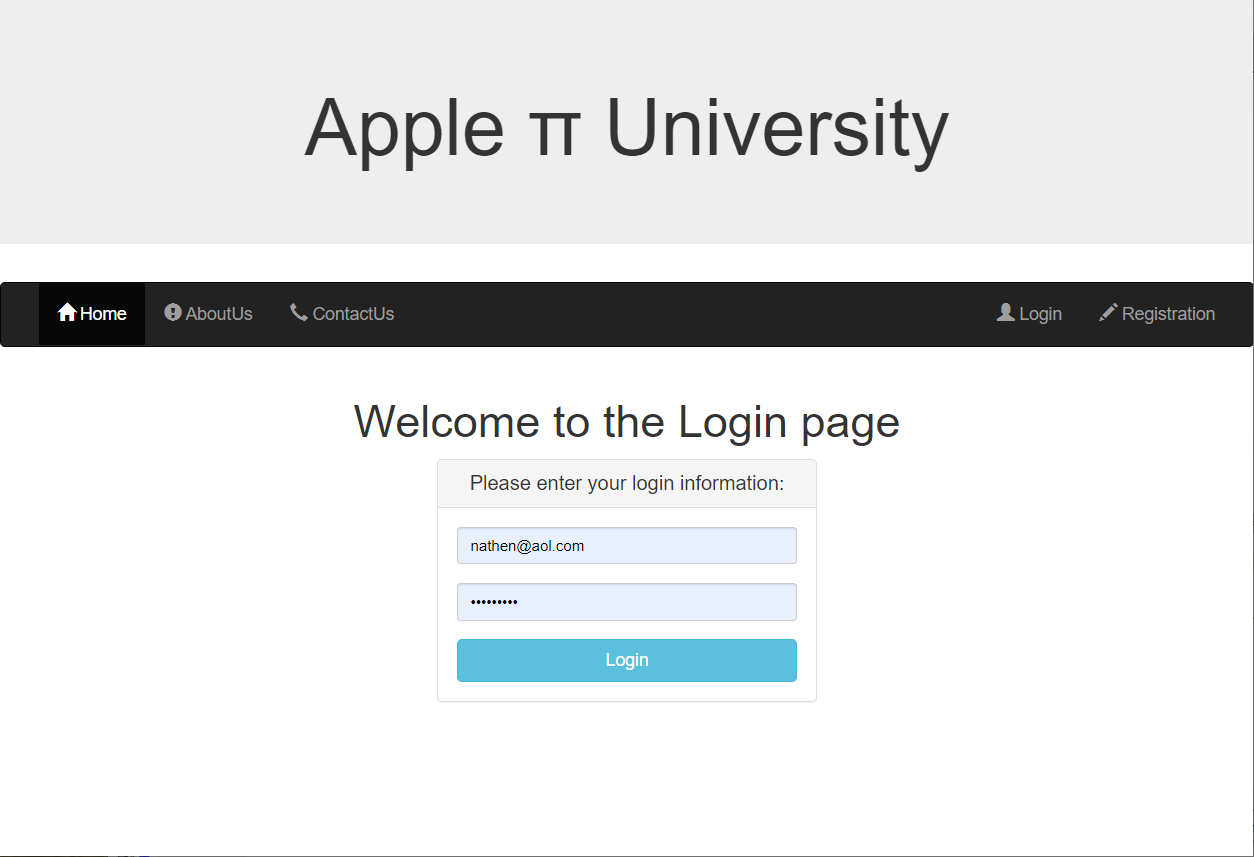
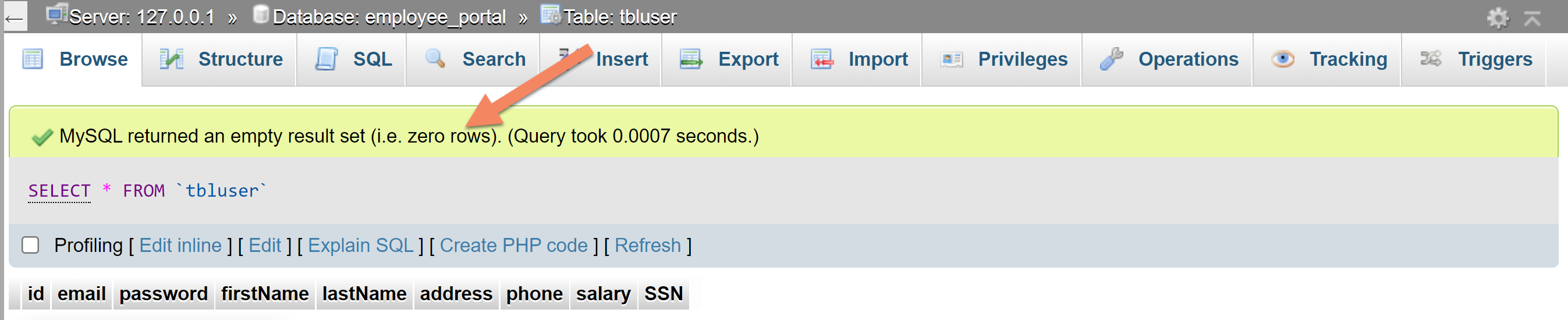
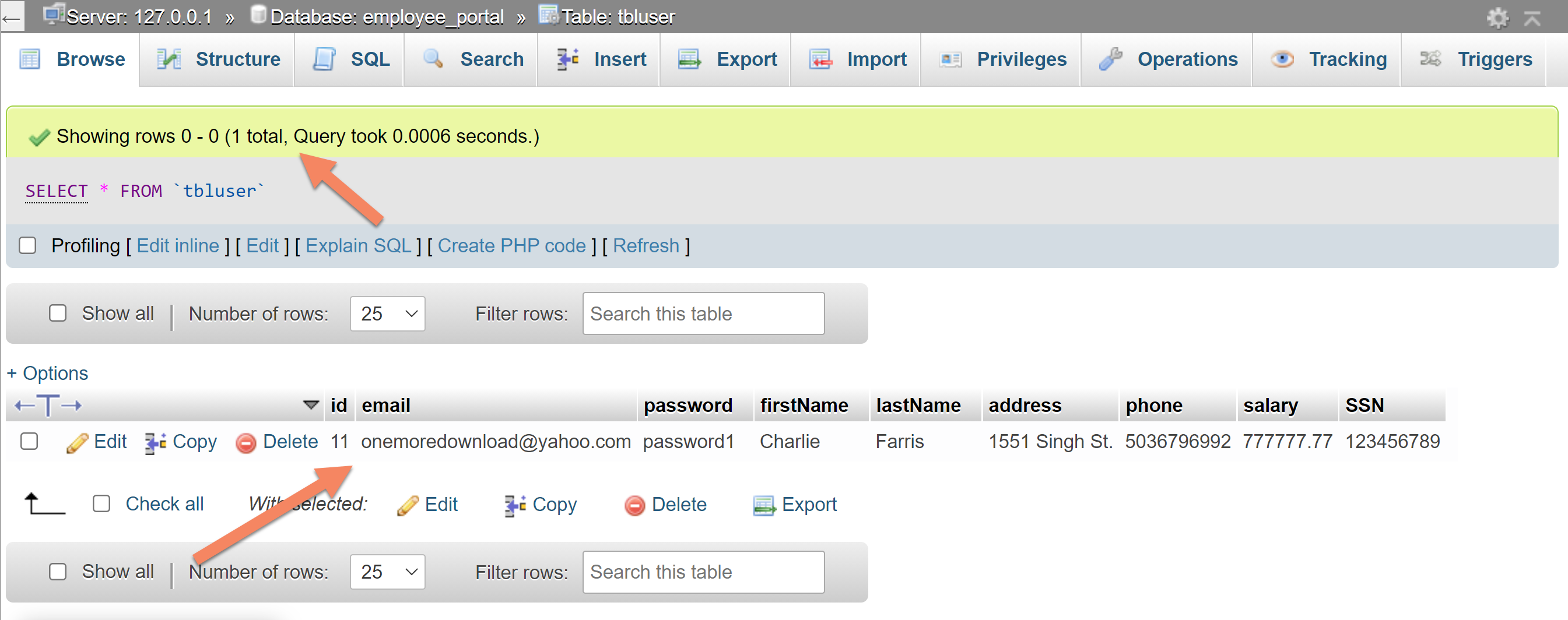


Figure 6: MySQL table before & after saving the user information in the database





The Profile page was a simple matter of reusing and updating the code from the Registration page. My first update, as seen in the Profile.php source code page, was to add the line session\_start(); so that the global $\_SESSION that I populated on the Login page was available to me. I then updated the fields as this was now a display values screen and not a user input screen. The changes came down to the following things:

* Update the fields from Required to Disabled so that it did not appear that a user could modify them.
* Remove the placeholder section of the input line.
* Change the type to text in the input line.
* Add in the value to be populated to the input line.
* Remove unassay lines of code.

I also made two other minor adjustments. The first was to move the password field up next to the SSN field, and the second was to add formatting for the SSN field so that it was recognizable as such. This can be seen in Figure 2:

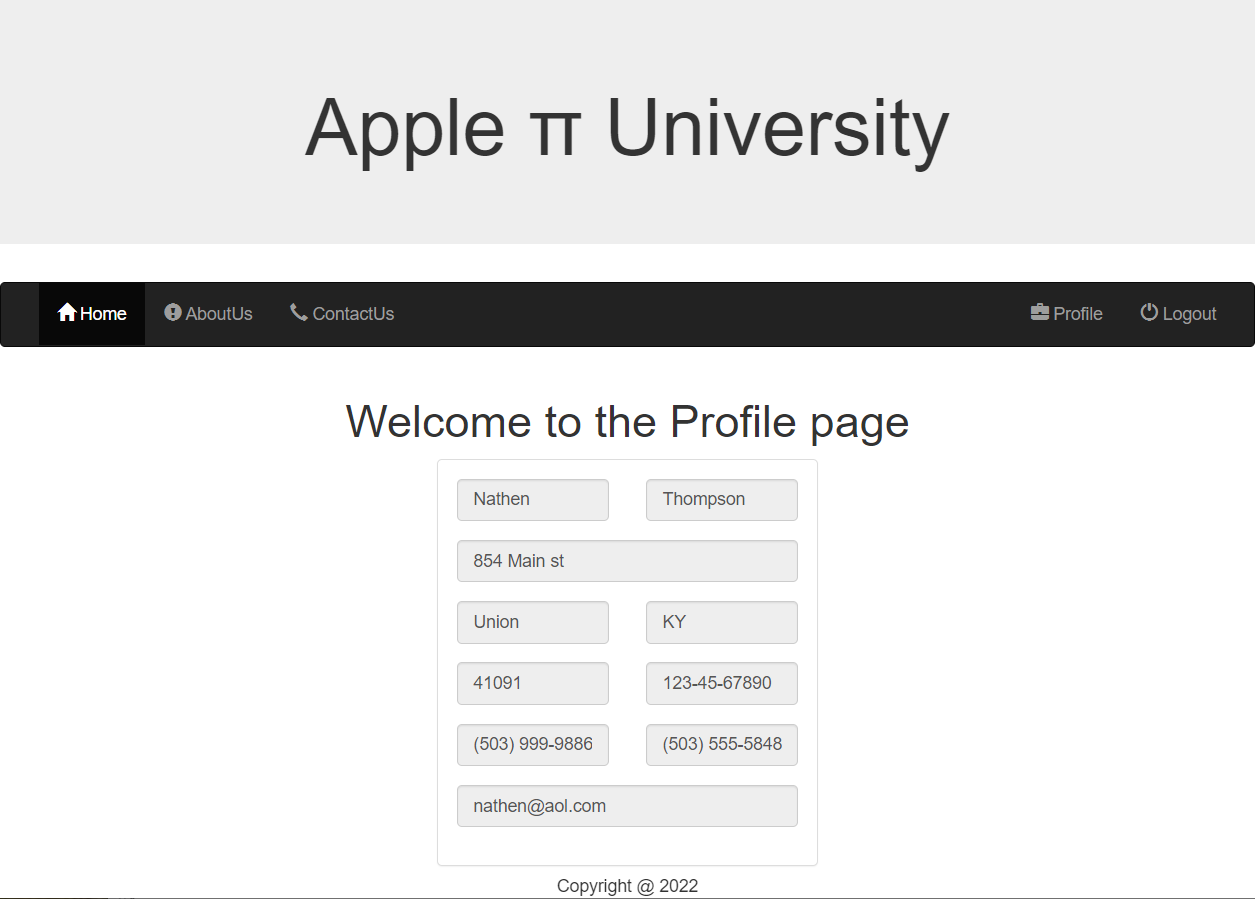
<?php

$ssn = substr($\_SESSION['SSN'], 0, 3).'-'.substr($\_SESSION['SSN'], 3, 2).'-'.substr($\_SESSION['SSN'],5);

?>

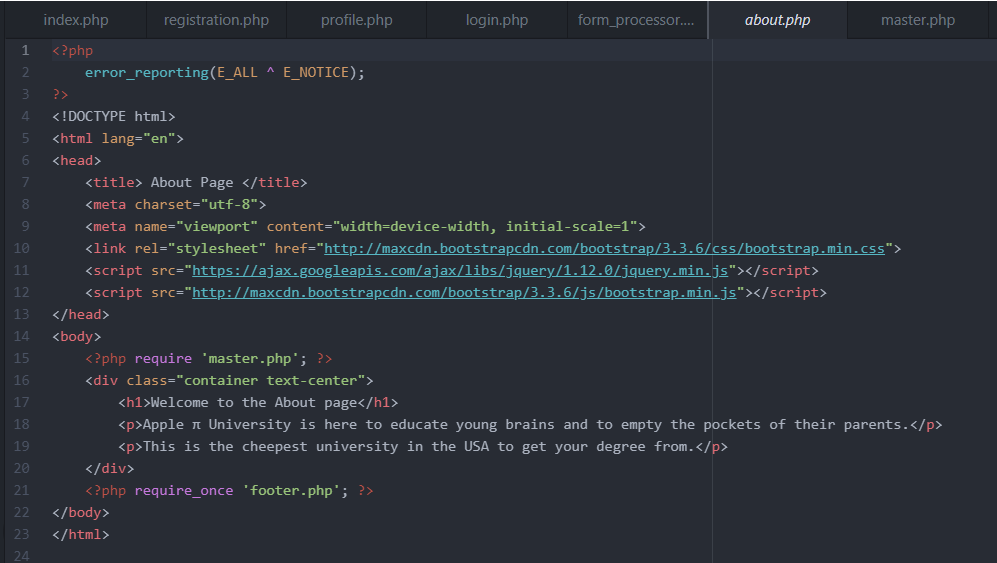
<input type="text" name="ssn" id="ssn" class="form-control" maxlength="11" value="<?php echo $ssn; ?>" disabled />

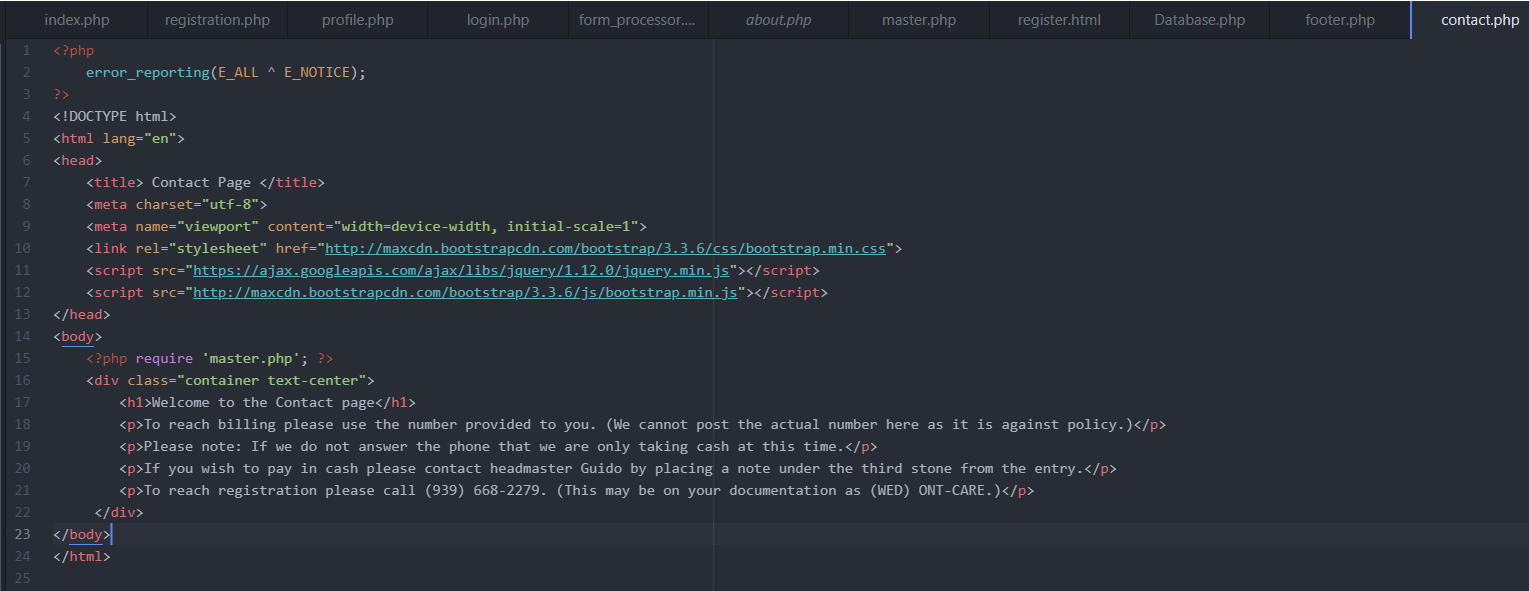
Figure 7: Profile page layout

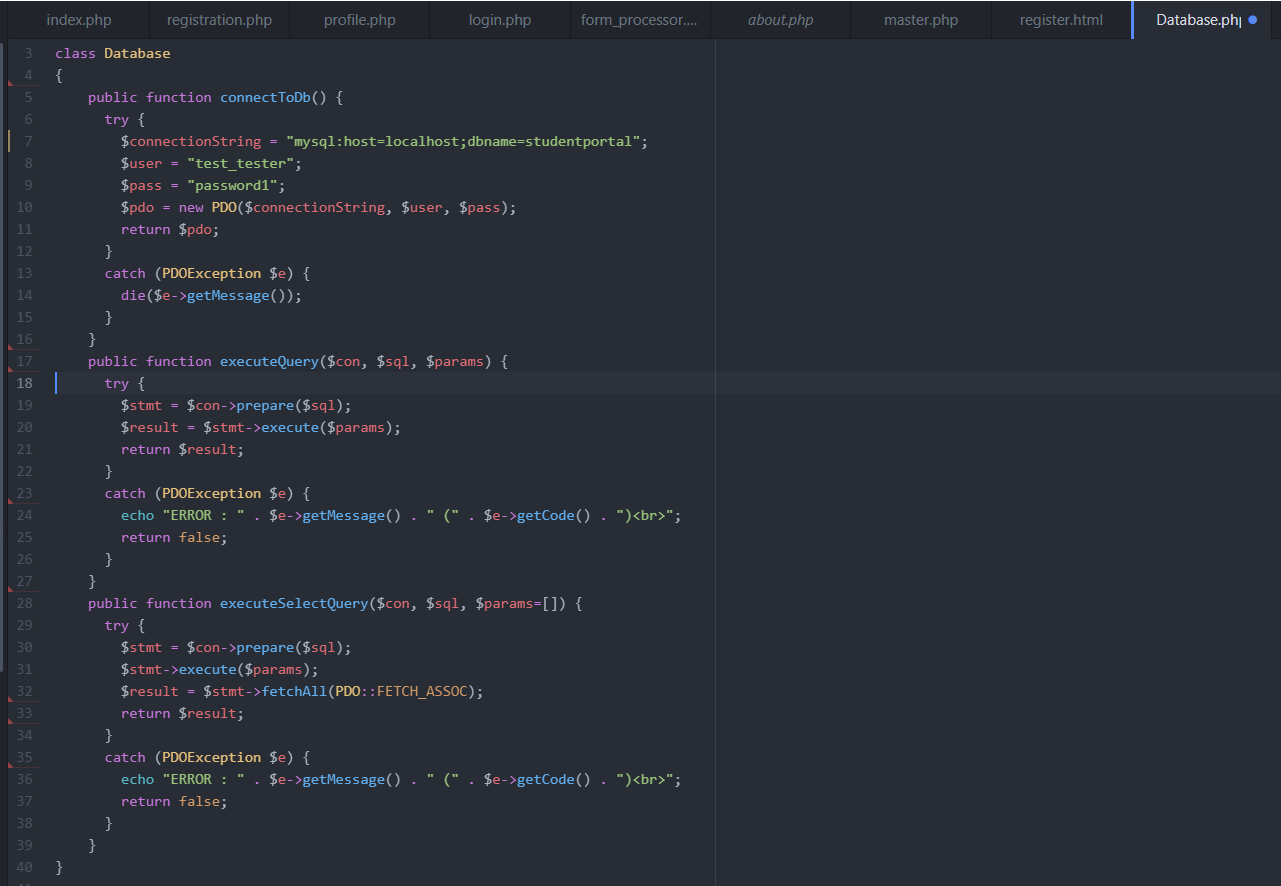


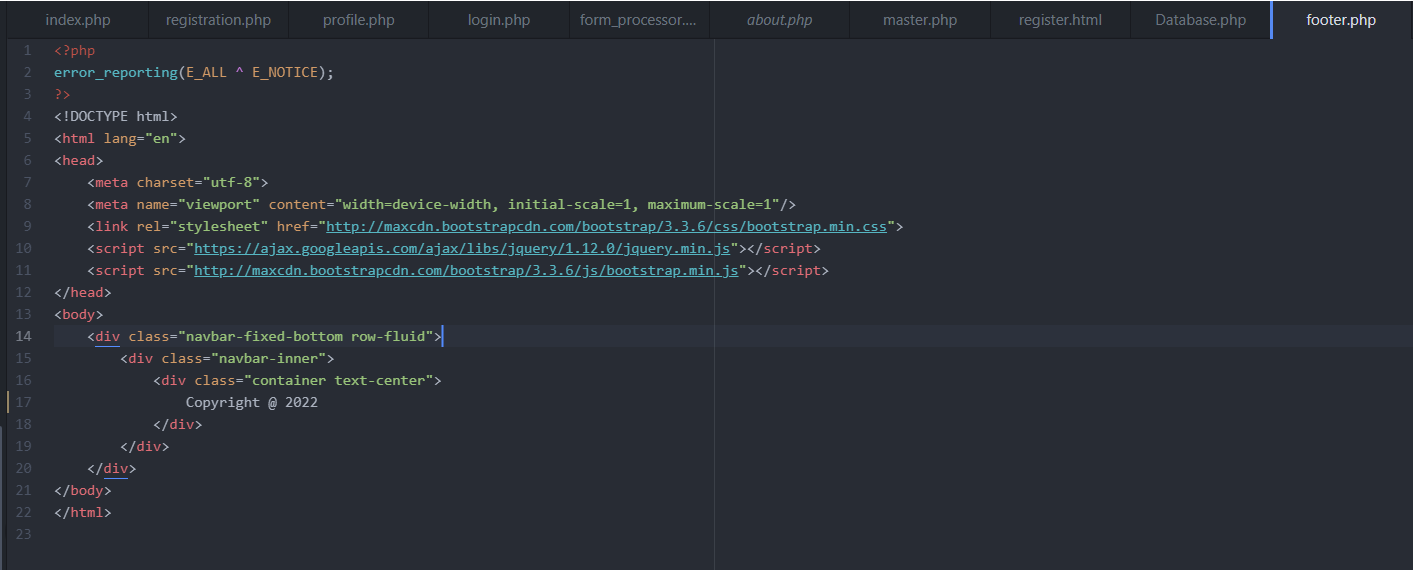
If you review Figure 7 you will find that the right most link changed from Registration to Logout. This happened when the end-user logged in successfully. The Logout feature enacts the session\_destroy() function in the index.php. The session\_destroy() function destroys all of the data associated with the current session. This makes sure that the super global variables contain no data. In our case the $\_POST global super variable is set back to an initial null state. This allows for web page security so that others cannot view this user’s profile page. Once the Logout link is clicked, the user is returned to the Home page.

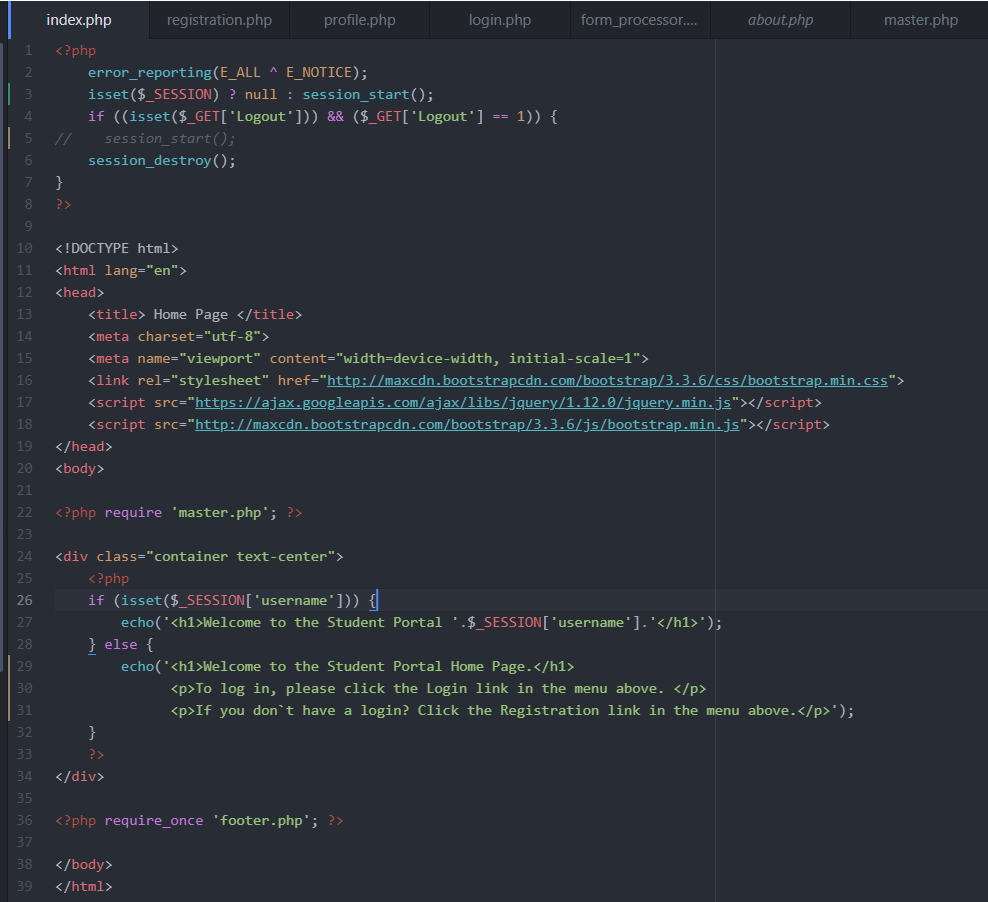
The following is a listing of each additional php page created that allowed this website to work properly:

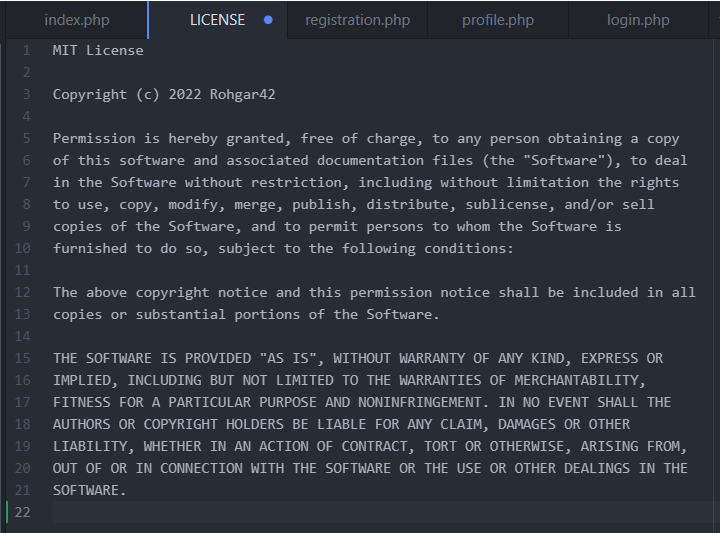


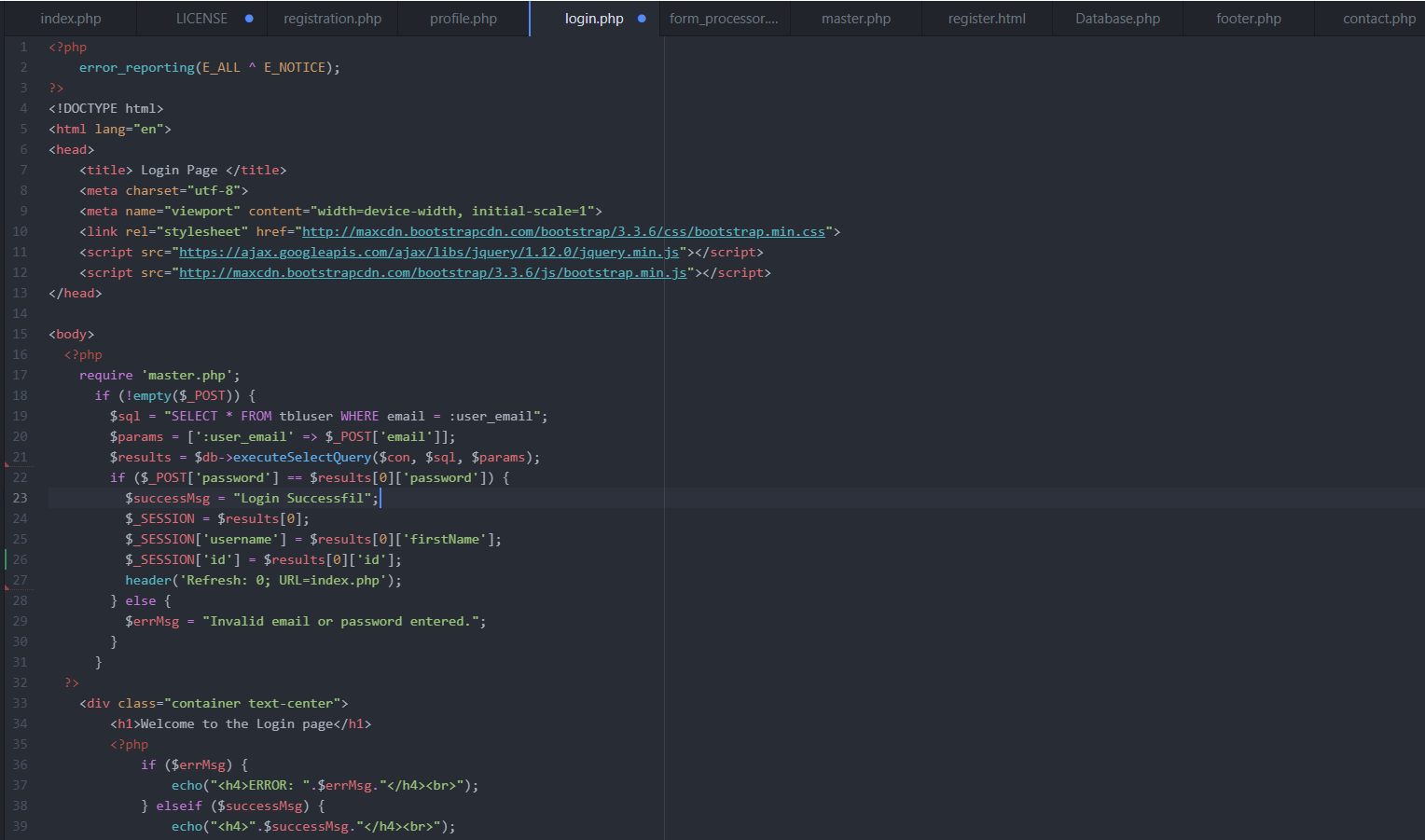


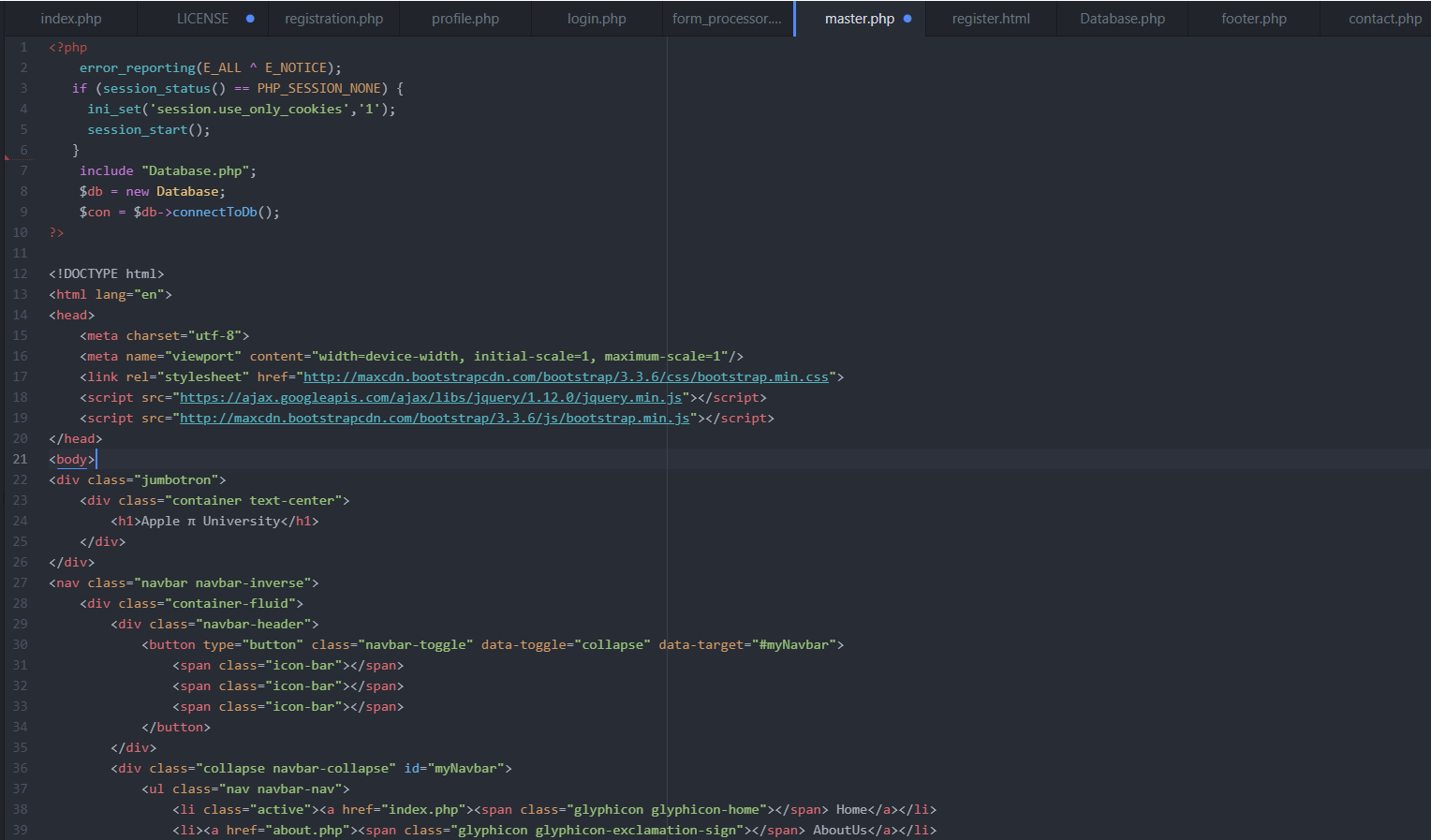


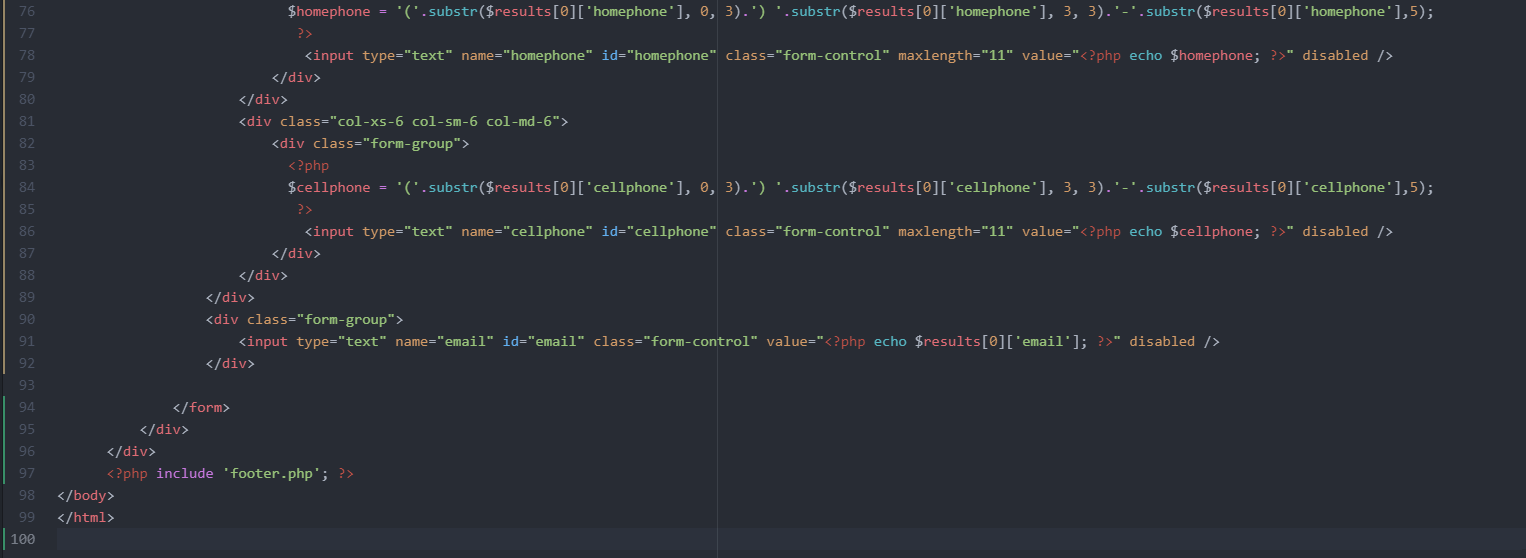
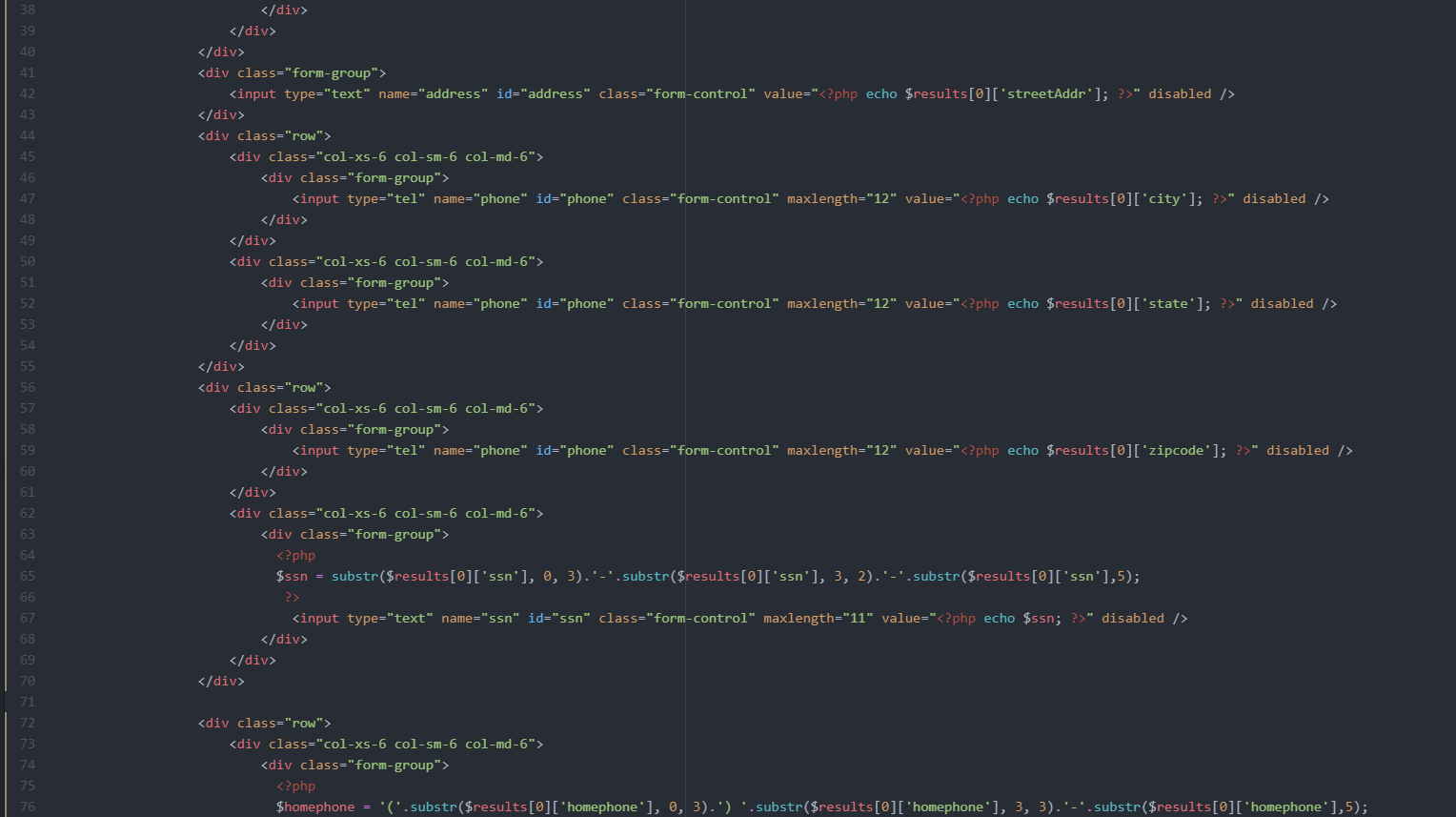
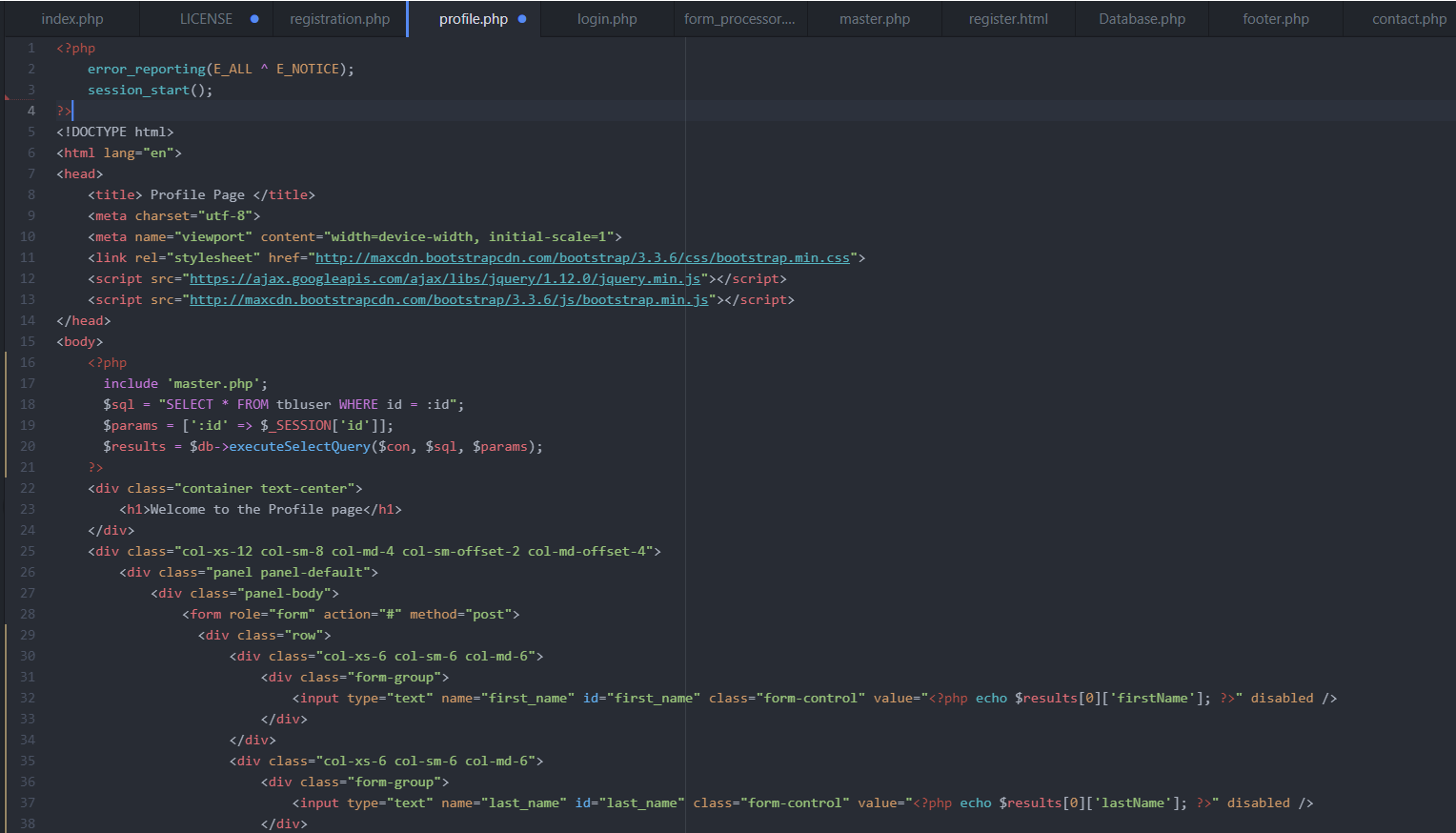


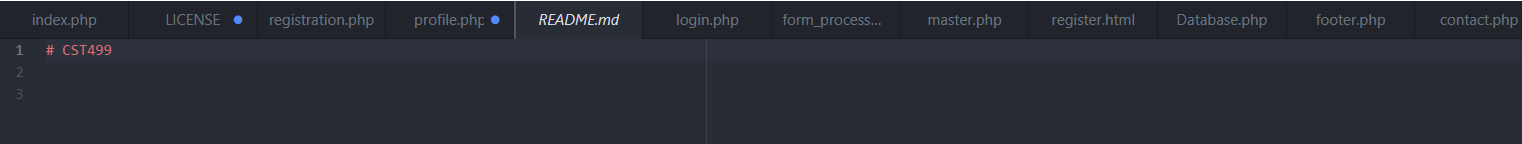












**References**

Mikoluk, K. (2013, September 18). *Xampp tutorial: How to use xampp to run your own web server*. Udemy Blog. <https://blog.udemy.com/xampp-tutorial/>

Connolly, R., & Hoar, R. (2018). *Fundamentals of web development* (Second edition). Pearson. <https://platform.virdocs.com/r/s/0/doc/350543/sp/15713223/mi/52963315>

HTML <head> Tag. (2018, October 5). *GeeksforGeeks*. <https://www.geeksforgeeks.org/html-head-tag/>