COSC 360
Lab 9 – Database and Data Storage with PHP

This is an individual assignment. This assignment is marked out of 35 points.

Due Date: April 11, 2021.

Information:

You will need to use your local web development stack to complete this lab.

You can find further practice and info for PHP at https://www.w3schools.com/php/.

Instructions:

In this lab, you are provided starter code for the user input pages as well as a basic database structure. Your goal is to build PHP pages to handle the operations for adding a new user to the database (`newuser.php`), allowing a user to login (essentially checking that the password and username exist (`login.php`), allowing a user to change their password (`changepassword.php`) and looking up a user in the database (`finduser.php`). The user input forms have been provided for you, along with basic validation scripts. Additionally, example code to test your database connectivity has been provided (`db_example.php`) that can be loaded into your webserver as well as showing examples of connections.

Information will be passed from the client side using a `POST`. Your PHP scripts will need to check and parse the appropriate data. You will need to examine the forms in the client side pages to determine the appropriate field names.

The database that you will use has one table (`user`) with the fields: `username`, `firstName`, `lastName`, `email` and `password`. The fields are varchar. The password field is intended to store the hash of the user password using `md5` hash. In this lab, the `password` can be sent in clear text via the `POST` but needs to be stored hashed in the database.

Part 1: Build and Test the Database

1. To ease development, you may want to check out the lab into your `htdocs` folder so that XAMPP will be able to host and render the files. If your files are not in the `htdocs` folder, XAMPP will not be able to process the php files. Make note of the folder name as you will use this in your URL when accessing the files at `localhost`.
2. Open `phpMyAdmin` (`localhost/phpMyAdmin`) from your lamp default homepage (`localhost`). Create a new database called `lab9` using the default settings, as shown in Figure 1.

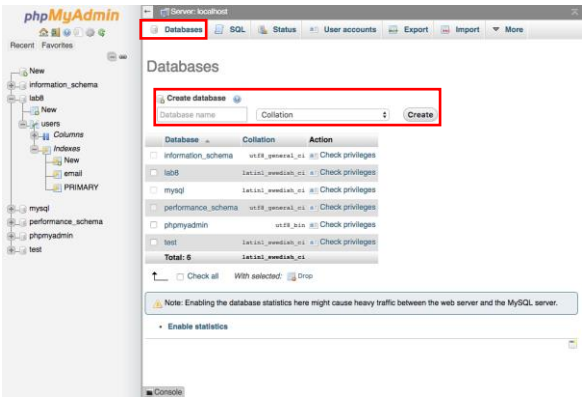


Figure 1 Create a New Database

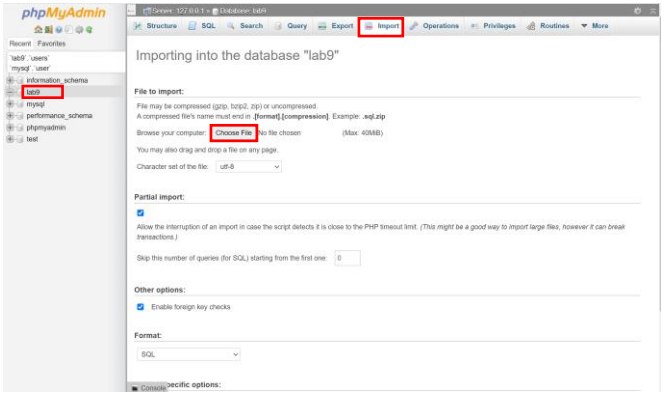
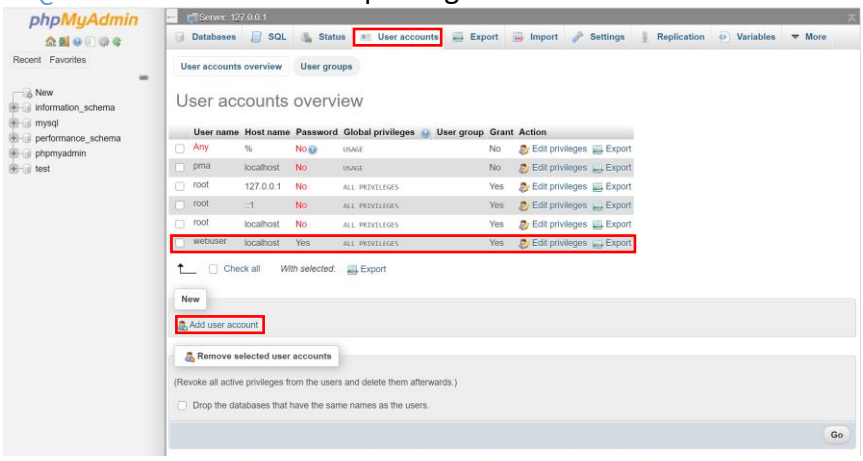


Figure 2 Import Data

3. Select the database you just created from the left-hand side explorer. This will allow you to interact with the database. You can now import the structure of the database to use for this lab. Select the Import operation (from the top menu bar) (Figure 2) and then select the file `lab9.sql` from the lab starter files. Select 'Go' (located at bottom of page). This will load data into the database.
4. Open your web browser, navigate to `localhost/<your lab9 folder>/db_example.php`. This example should connect to the database and display the contents on the user table using the `mysqli` API (http://php.net/manual/en/book.mysqli.php). From the example code you will be able to see the credentials for connecting to the database as well as establishing a connection. The parameters required for the connect are:

```
$host = "localhost";
$database = "lab9";
$user = "webuser";
$password = "P@ssw0rd";
```

   Do not proceed until the example code is operational. The database contains one user for `dVader` with the password of `password`.
5. If you cannot connect to the database, please check if the database user account "`webuser`" is added with the password "`P@ssw0rd`". Select all the privileges.



Part 2: Adding a New User

1. Create a PHP file called `newuser.php`. Examine the page `db_example.php` as a template to start this page as it contains examples of making database connections and running queries. Chapter 14 contains examples that are of use as well. This page will receive the request from `lab9-1.html`

(starter code has been provided for you). The script associated with this file will do partial validation based on if the fields are empty or not. Inspect the code to understand how it is operating but do not change the contents of the `vadiate.js` script.

2. In the head after the point where `validate.js` is included in `lab9-1.html`, create an embedded script block. Within the block create a function called `checkPasswordMatch(e)`. In this function, use JavaScript to check to see if the two password fields are the same. If the two fields for the passwords do not match, use the `makeRed()` function (provided in `validate.js`) to highlight the field and use an alert to let the user know that the passwords do not match. Additionally, prevent the submission of the page until the passwords match. Once the passwords match, allow the submission of the form.
3. Add the functionality to `newuser.php` to allow the script to process the user data sent from `lab9-1.html` via the `POST`. This page will need to connect to the database. Ensure that the page validates the type of request method as well as checking to ensure that all parameters are set. The keys that you will need to use to access the `POST` data are: `firstname`, `lastname`, `username`, `email` and `password`.
4. Check to see if the user already exists in the database based on username or email address. If the user already exists, have the page display the message `User already exists with this name and/or email` as well as displaying a return link back to the referring page (Figure 3).

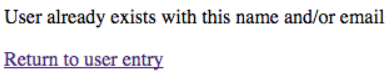


Figure 3 - User Exists Message

5. If the user does not exist, insert their information into the database and display a message that the user account has been created (Figure 4). The `password` (which was send as clear text) will need to be hashed before being inserted into the database which can be done with the `md5()` function,

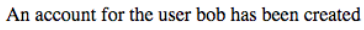


Figure 4 User Created

6. Ensure that your page handles the condition for bad data being injected via a `GET` request as well as properly closing the database connection when complete.
7. Test your page to ensure that you can create multiple users as well as it is preventing duplicates with the same username or email.

Part 3: Logging in

1. Create a PHP file called `login.php`. This page will receive the request from `lab9-2.html` (starter code has been provided for you). Inspect the code to understand how it is operating but you do not need to change the HTML.
2. Complete `login.php`. This page will need to connect to the database and check to see if the username and password entered are correct sent from `lab9-2.html` via the `POST`. Ensure that the

page validates the type of request method as well as checking to ensure that all parameters are set. The keys that you will need to use to access the POST data are username and password.

3. Determine if the username and password combination are valid in the database. Recall that the password is sent in clear text but stored after being hashed with MD5. If the username and password are correct, display the message that the **user has a valid** account otherwise display that the **username and/or password are invalid**.
4. Complete the code to close the database connection as well as handling the condition for bad data.
5. Test your page with the accounts created to ensure valid users are detected as well as bad accounts.

Part 4: Changing a Password

1. Create a PHP file called changepassword.php. This page will receive the request from lab9-3.html (starter code has been provided for you). Inspect the code to understand how it is operating. Add the code to validate that the new passwords are the same using your previous script (see part 2).
2. Complete changepassword.php. This page will need to connect to the database and check to see if the username and password entered are correct sent from lab9-3.html via the POST. Ensure that the page validates the type of request method as well as checking to ensure that all parameters are set. The keys that you will need to use to access the POST data are username, oldpassword and newpassword.
3. Determine if the username and oldpassword combination are valid in the database. Recall that the password is sent in clear text but stored after being hashed with MD5. If the username and oldpassword are correct, update the user's password in the database and display the message that that the **user's password has been updated.** If the username and/or password are incorrect display that the **username and/or password are invalid**.
4. Complete the code to close the database connection as well as handling the condition for bad data.
5. Test your page with the accounts created to ensure valid users are able to change their passwords as well as noting bad accounts.

Part 5: Finding a User

1. Create a PHP file called finduser.php. This page will receive the request from lab9-4.html (starter code has been provided for you). Inspect the code to understand how it is operating.
2. Complete finduser.php. This page will need to connect to the database and check to see if the username exists based on data from the form in lab9-4.html via the POST. Ensure that the page validates the type of request method as well as checking to ensure that all parameters are set. The keys that you will need to use to access the POST data is username.
3. If the user exists, display their first name, last name and email address in a table contained in a fieldset as shown in Figure 5. Do not display the password hash.

Figure 5 User Information

4. Complete the code to close the database connection as well as handling the condition for bad data.
5. Test your page with the accounts created to ensure that data is presented for valid users.

Part 6: Streamlining Your PHP

While working on the various pages, you may have noticed that there is a lot of duplicate code. Can you think of a way to streamline your scripts to reduce the amount of duplicate code you have and increase the supportability? Experiment to address this issue.

Evaluation Criteria:

	Part 2
2 points	Lab9-1.html password validation script
4 points	Validation/ collection of POST data
3 points	Database connection setup/teardown
2 points	Username/password pre-check
1 point	Referring link
1 point	Password storage
2 points	User creation in DB
2 points	Bad data handling
	Part 3
1 point	Validation/collection of POST data
2 points	User validation in DB
1 point	Bad data handling/closing
	Part 4
1 point	Password validation on client side
1 point	Validation/ collection of POST data
2 points	User validation in DB
2 points	Update of password (with hash)
1 point	Bad data handling/closing
	Part 5
1 point	Validation/ collection of POST data
2 points	User validation in DB
3 points	Formatting of data in fieldset/table
1 point	Bad data handling/closing