CMPE2550 – Lab02 – Authentication/Authorization

In this lab, you will reinforce and expand your skills with PHP and mySQL database operations. Additionally, you will explore the handling of credentials for authentication of users and authorization of authenticated users for specific resources based on system roles.

Base Functionality (80%)

Database Setup

For starters, you will need to create tables in your database.

One of the tables will be for storing user information. You will need to consider what information you would like to store, but at minimum you will store a user ID, username, and password.

You will then need to consider how you would like to handle user roles for your system. Can a user have more than one role in your system? The answer to this question will determine whether you need one or more additional tables to store role related information.

If only one role per user is permitted, then a single roles table will suffice, but you will need to add an extra column to your users table to link a user to a role.

If you prefer the flexibility of assigning more than one role per user, either now or possibly in the future, then it is suggested that a role table be created that stores only role information, and then a linking table for connecting users to their role(s).

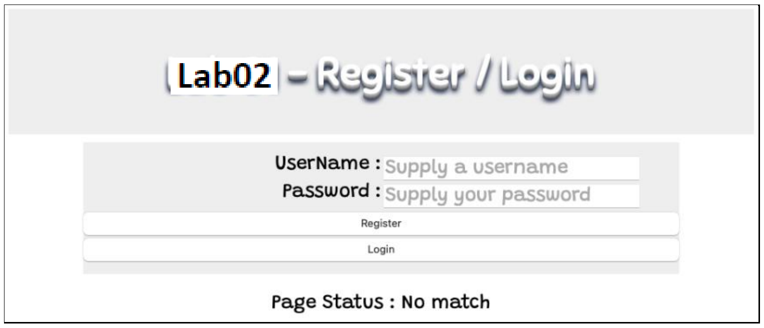
Suggested role information is a role ID, role name, description, and a numerical role value. This last will help with ranking different roles and will make adding or removing roles more seamless than trying to use the role ID also as the role rank. Ask your instructor about this as needed.

Finally, regardless of the rest of the roles that you eventually create, you will want a Root role, and exactly one user that is assigned as the sole Root user in your system. This Root user will be able to affect all other user information, such as assigning role(s) to users, and other site features. You will also want a default role which will have the lowest authority in your system for all newly registered users. The rights of this role should be very minimal, or maybe none.

User Registration / Creation

You will include a page in your website for registering new users. Your page must include the opportunity for new users to provide all the information you decide to store in your user table, minus the user ID which shall be auto populated, and omitting a role that shall automatically be whatever you name your default role with minimal access.

Optionally, you may combine your registration and login pages as shown in the following image.



Of course, as soon as you wish to store more than the minimal user information shown above, you should have a separate page for registration.

When a new user clicks the Register button, the username and password fields will be checked for validity. Feel free to impose whatever rules you wish for valid passwords, but usernames should be unique at minimum. Field validation shall occur on both the client and the server.

A best practice when dealing with user passwords is to ***never*** store plain text passwords. Your site shall adhere to this practice and instead store a password hash. Your instructor will explain this further. Thankfully, PHP provides a function for creating password hashes, appropriately named *password_hash.* You will read about this function and use it when registering users.

The bottom label is only shown as an example of where needed user messaging should appear.

Once a user is registered successfully, they shall be returned to the login page if a separate page has been used for registration.

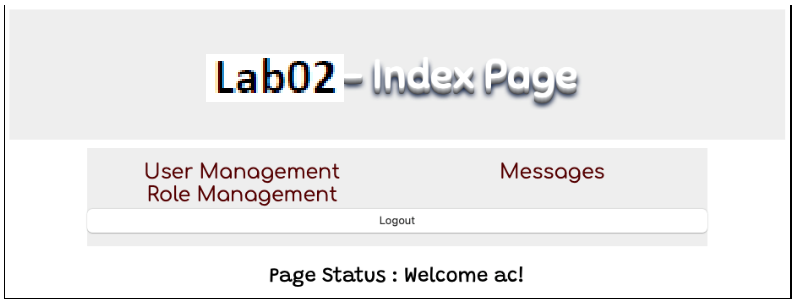
User Authentication / Login

When an existing user clicks the Login button, the username and password fields will be checked for existence. Field validation shall occur on both the client and the server.

For a user to successfully authenticate, you will need to compare the user's entered password to the password hash stored in the user table. To do this, you will need to hash the entered password using the same algorithm that was used when the user was created. Again, PHP provides a function, *password_verify*, that will aid you in the task. You will read about this function and use it when verifying users. It is suggested that you set up a separate Validation function to encapsulate this task.

Again, proper messaging shall be used to communicate to the user any errors that occur while attempting to log in to the site.

Upon successful login, users shall be redirected to an index page, similar to the following image. Your instructor will explain how redirection may be accomplished.



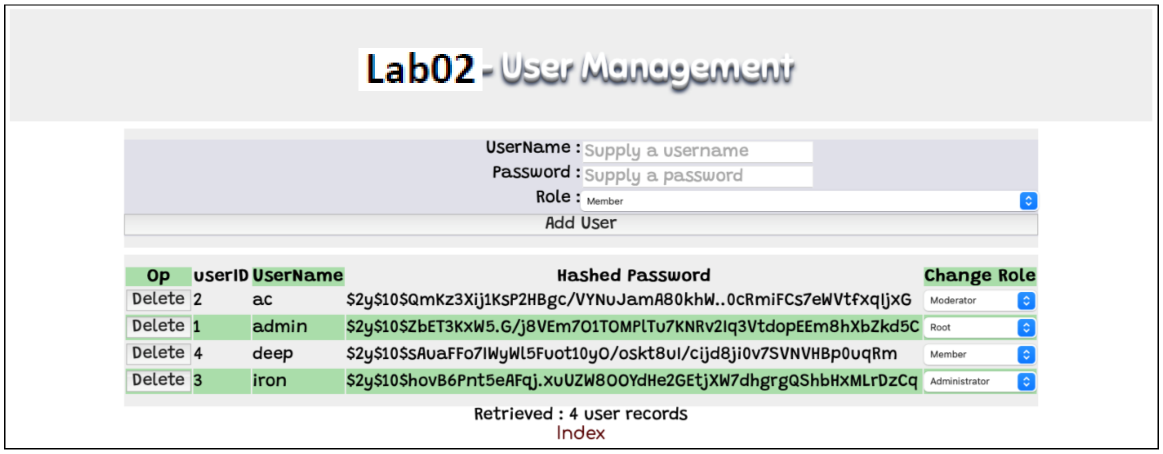
Page Status : Welcome ac!

What the logged in user sees in the above image should differ by the user's role. For instance, the Root user should see all of the options whereas a newly registered user might only see the welcome message at the bottom of the image. As you add roles and functionality into your system you may determine what each role is permitted to see. However, you should think about the ramifications. For instance, if any role other than the Root user is permitted to see and access the Role Management page, they should have limited ability to affect existing roles. Their own role and any role with a higher rank than their own role should be inviolate.

Note: The Messages link may navigate to a page with nothing more than a generic access message for the purpose of this lab. It is not intended to be implemented until Lab 02.

User Management

Exactly what you make your user management page look like is left to your discretion, but a possible example is shown below.



UserName : Supply a username
Password : Supply a password
Role : Member

Op	userID	UserName	Hashed Password	Change Role
Delete	2	ac	$2y$10$QmKz3Xij1KsP2HBgc/VYNuJamA80khW..0cRmiFCs7eWVtfxqljxG	Moderator
Delete	1	admin	$2y$10$ZbET3KxW5.G/j8VEm7O1TOMPlTu7KNRv2Iq3VtdopEEm8hXbZkd5C	Root
Delete	4	deep	$2y$10$sAuaFFo7IWyWl5Fuot10yO/oskt8uI/cijd8ji0v7SVNVHBp0uqRm	Member
Delete	3	iron	$2y$10$hovB6Pnt5eAFqj.xuUZW8OOYdHe2GEtjXW7dhgrgQShbHxMLrDzCq	Administrator

Retrieved : 4 user records

At minimum, the page must provide the ability to submit new users, delete an existing user, and change the role(s) of an existing user. User IDs may not be adjusted. As a best practice, hashed passwords should not be allowed to be edited.

Roles are very important on this page. No user may affect another user with a role whose rank is greater than or equal to their own. For example, in the above image this means that an Administrator may not create, delete, or promote another user to Administrator or Root.

Role Management

The purpose of this page is for users of authorized role(s) level privilege to access role management. If an unauthorized user attempts to access this page (manually as they should not be shown a link on the index page), you will redirect them to the index page with an appropriate status message.

A screenshot is not being provided as the roles page design is being left to your discretion; however, you might consider that maintaining a consistent feel to your site overall would be beneficial for saving time and ease of work. Note again that access to the role management page must not be determined by username, but rather by role in some way.

Upon first load of the page, all available role information must be displayed. Your page must also include functionality for adding new roles, updating all current role information except role ID, and deleting current roles. Also, users of a given role should not be able to affect any roles with a rank greater than or equal to their own.

You must spend some time thinking about cases that would cause your site trouble after you have nailed down the basic functionality. You are likely to encounter some strange behaviour with some actions. Be sure to use the troubleshooting techniques you have learned to this point to help you solve them. It is also highly recommended that you build one piece of functionality at a time and test it thoroughly before moving on to the next. If you end up wrecking or polluting a piece of your database, remember that you may always reload your tables.

For file organization, it is highly suggested that you separate functionality for different pages into separate PHP files. You may even go so far as to include subfolders for separating files by type or functionality, For instance, all Javascript files in one folder or alternately all user management related files in a single folder. Just make sure to be consistent in your site.

Enhancement Functionality (20%)

This section will include a bit of research dealing with transactions for all multiple query data change in your site. This may be accomplished in both PHP and with prestored transactions embedded in stored procedures in your database. You will find the following resources useful as a start:

https://www.php.net/manual/en/mysqli.begin-transaction.php

https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

https://dev.mysql.com/doc/refman/8.0/en/commit.html

https://www.php.net/manual/en/mysqli.quickstart.stored-procedures.php

Note that procedures do not need to be created each time you wish to use them as one might surmise from the presented examples. The examples are just providing everything needed for you to try them out for yourself.

10% of the enhancement mark will be allocated to each of the two transaction strategies.