Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-452-M2024/it202-module-6-milestone-1-2024/grade/dr475

IT202-452-M2024 - [IT202] Module 6 Milestone 1 2024

Submissions:

Submission Selection

1 Submission [active] 7/8/2024 12:33:04 PM

Instructions

COLLAPSE

Overview Video: https://youtu.be/V7oHa8KKtss

Prereqs:

- Go through each lesson from "Project Setup and SQL" to "User Login Enhancement" and follow the branching names while gathering the code
- Merge each into Milestone1 branch
- Create a Project board on GitHub (if you haven't yet)
    - Add each major item from the proposal doc as an Issue item
    - Invite the grader(s) and myself as collaborators on the board (they're separate from your repository)
        - See Canvas announcements for the Usernames or check your collab list on your repo
- Mark the related GitHub Issues items as "done"
- Implement your own custom CSS (something much different than the default "ugly" CSS given as an example)
- Consider styling all forms/inputs, data output, navigation, etc
- Implement JavaScript validation on Register, Login, and Profile (include "[Client]" in the output messages to differentiate between server-side validations)

Instructions:

1. Make sure you're in Milestone1 with the latest changes pulled
2. Ensure Milestone1 has been deployed to heroku dev
3. Gather the requested evidence and fill in the explanations per each prompt
4. Save the submission and generate the output PDF
5. Put the output PDF into your local repository folder

6. add/commit/push it to GitHub
7. Merge Milestone1 into dev
8. Locally checkout dev and pull the changes
9. Create and merge a pull request from dev to prod to deploy Milestone1 to prod
10. Upload this output PDF to Canvas

Branch name: Milestone1

Tasks: 25 Points: 10.00

User Registration (2 pts.)

^COLLAPSE ^

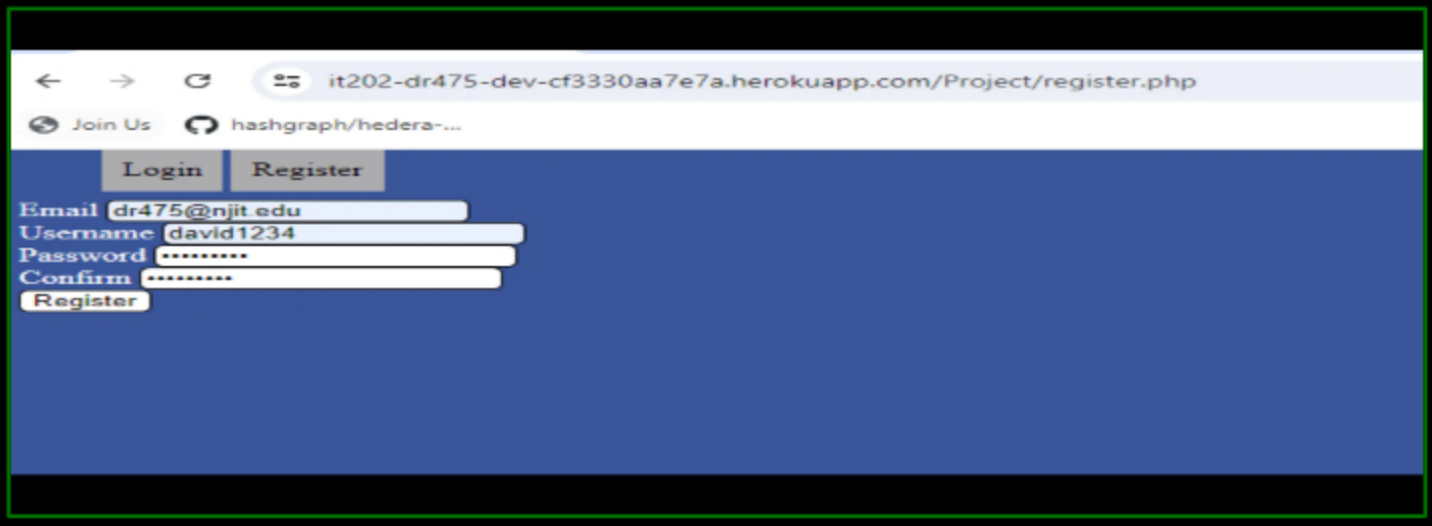
^COLLAPSE ^

Task #1 - Points: 1

Text: Screenshot of form on website page

Details:
Thoughtful CSS should be applied to all parts (must differ from the "ugly" CSS given via the lessons).
The Heroku dev URL must be present in all screenshots of the site.

#1) Screenshot of the form (ensure valid data is filled in)



Caption (required) ✓

Describe/highlight what's being shown

All form data filled out on registration page

URL (required) ✓

Prod link to the registration page

https://it202-dr475-prod-7559be2e2ad4.herokuapp.com/Project/register.php

#2) Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required

Caption (required) ✓

Describe/highlight what's being shown

Javascript validation for each field, and their respective alerts upon errors.

#3) Demonstrate email already in use message (message text doesn't need to be exact, but should be clear)

Caption (required) ✓

Describe/highlight what's being shown

Error message shown when email is taken

#4) Demonstrate username already in use message (message text doesn't need to be exact, but should be clear)

Caption (required) ✓

Describe/highlight what's being shown

Username is taken error message

#5) Demonstrate user-friendly message of new account being created

Caption (required) ✓

Describe/highlight what's being shown

Successful registration message

COLLAPSE

Task #2 - Points: 1

Text: Screenshot of the form code

Details:

Should have the appropriate type attributes for the fields.
Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

#1) Show the html of the form

```
<!-- UCID: dr475 -->
<!-- Date: 07/08/2024 -->
<form onsubmit="return validate(this)" method="POST">
    <div>
        <label for="email">Email</label>
        <input type="email" name="email" value="<?php echo se($_POST, "email", "", false); ?>" required />
    </div>
    <div>
        <label for="username">Username</label>
        <input type="text" name="username" value="<?php echo se($_POST, "username", "", false); ?>" required maxlength="30" />
    </div>
    <div>
        <label for="pw">Password</label>
        <input type="password" id="pw" name="password" required minlength="8" />
    </div>
    <div>
        <label for="confirm">Confirm</label>
        <input type="password" name="confirm" required minlength="8" />
    </div>
    <input type="submit" value="Register" />
</form>      You, 3 weeks ago via PR #20 • Add registration functionality
```

Caption (required) ✓

Describe/highlight what's being shown

HTML code of the registration form

Explanation (required) ✓

Briefly explain the html for each field including the chosen attributes

PREVIEW RESPONSE

Email - Input box for the email of type "email"; it is required; the name="email" so it can be targeted using a query selector, and the value is set to the $POST["email"] variable so upon page reload it saves the previous value.

Username - Input box for the username of type "text"; it is required with a max input length of 30; the name="username" so it can be targeted using a query selector, and the value is set to the $POST["username"] variable so upon page reload it saves the previous value.

Password - Input box for the password of type "password" so it does not show the currently inputted text; it is required with a minimum length of 8; the name="password" so it can be targeted using a query selector

Confirm - Input box similar to the password input, it is required to have the same value as password as it is used to validate that the password inputted is accurate.

^COLLAPSE^

Task #3 - Points: 1

Text: Screenshot of the client-side and server-side validation code

Details:
Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

#1) Show the JavaScript validations of the form (include any extra files related if you made separate files)

Caption (required) ✓

Describe/highlight what's being shown

Code for JS validation

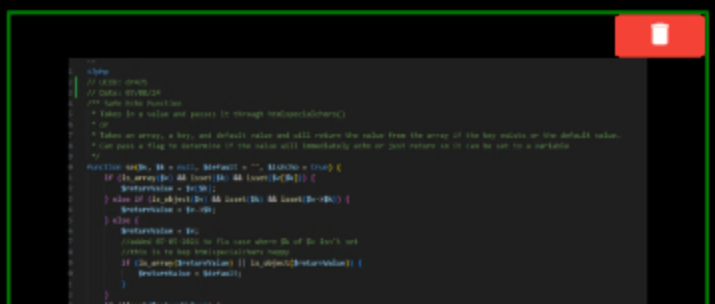
Explanation (required) ✓

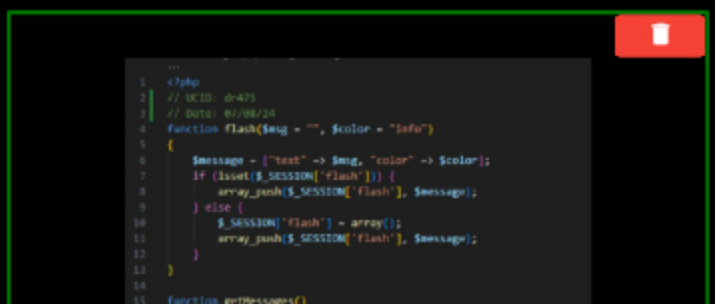
Explain in concise steps how this logically works

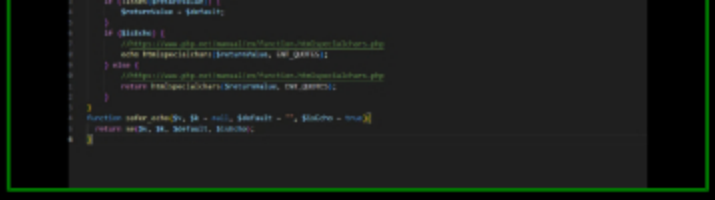
PREVIEW RESPONSE

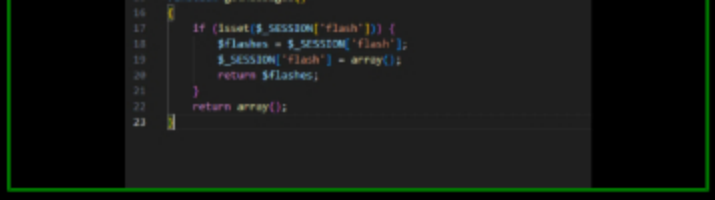
The HTML form calls the validate function which will return false (and wont send a request to the server) if any of the input fields are not valid. The function goes through the email and username and uses Regex to validate the length and characters in the input values, and check if they're empty. It also goes through both passwords and check if they're empty, of required length, and if they match. It ultimately returns true when there are no errors found.

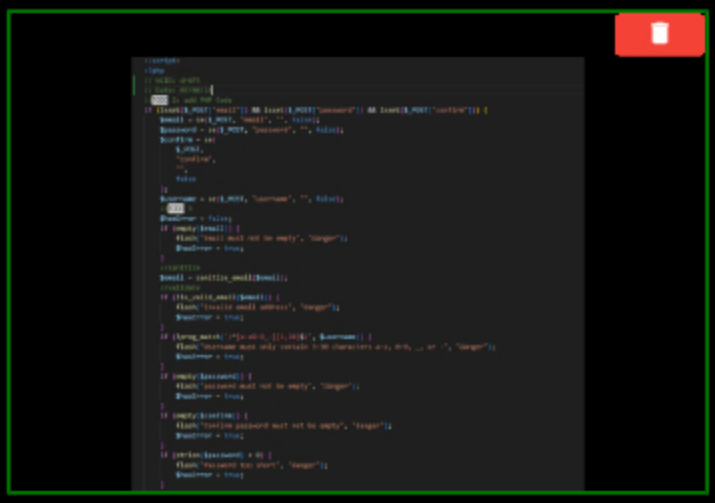
#2) Show the PHP validations (include any lib content)











Caption (required) ✓

Describe/highlight what's being shown

PHP server side validation code, including lib functions

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

First, the server checks if the POST request received has the required values to register (email, username, password, confirm password). Then, it validates the data (length, valid characters, password and confirm password being the same). If any error is found, it is returned to the client and displayed on the screen. Ultimately, if the data is valid, it hashes the password and sends a query to the database inserting a new row in the Users table with the requested values for email, username, and password. If the query is successful, it returns a success message on the user's screen "Successfully registered!", but if an error occurs, it is shown.

SE/safer_echo = Takes in an array, key, default value and returns the value from the array if it exists or the default provided value.
flash() = Shows the user a provided message with defined styling
users_check_duplicates() = Shows the user an error message if the provided error contains an error pertaining to a duplicate user in the database

getDB() = Provides a connection to the DB given a connection string in .env

COLLAPSE

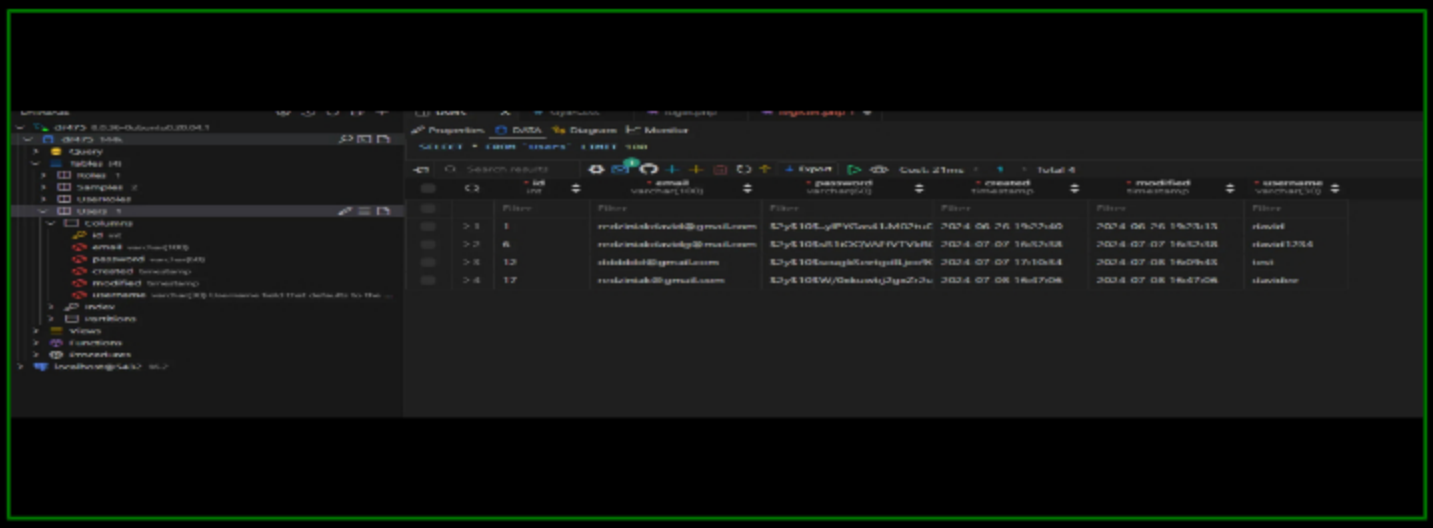
Task #4 - Points: 1

Text: Screenshot of the Users table with a valid user entry

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Password should be hashed
#2	1	Should have email, password, username (unique), created, modified, and id fields
#3	1	Ensure left panel or database name is present (should contain your ucid)

#1) Show valid data per the checklist



Caption (required) ✓

Describe/highlight what's being shown

The user table

COLLAPSE

Task #5 - Points: 1

Text: Explain the registration logic in a step-by-step manner from when the page loads to when the data is saved to the DB

Details:

Don't just show code, translate things to plain English in concise steps.

May be worthwhile using a list.

Response:

1. The user retrieves the register.php page from the server which first resets all the session values with reset_session() (flash, email, username, etc)
2. The user is then presented with the form that is required to fill out in order to create a new user
3. The user then fills out all the input fields (email, username, password, confirm password) and presses the submit button.
4. The JS function validate() is then called to provide client side validation of the fields before it is officially sent to the PHP server.
5. If any of the fields are not valid, it tells the user. If it is all valid, it is then sent to via a POST request to the PHP server for processing.
6. The PHP code in register.php then processes the data and provides extra server-side input validation (lengths of all input text, checks if email is taken, password and confirm password is the same, username is taken).
7. If everything passes validation, the fields are then submitted to the DB with an INSERT SQL query with the values and returns a successful message to the client.
8. Furthermore, if there are any errors when processing the query, it is then returned to the client.

COLLAPSE

Task #6 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/20

URL #2
https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/25

URL #3
https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/45

URL #4
https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/47

User Login (2 pts.)

COLLAPSE

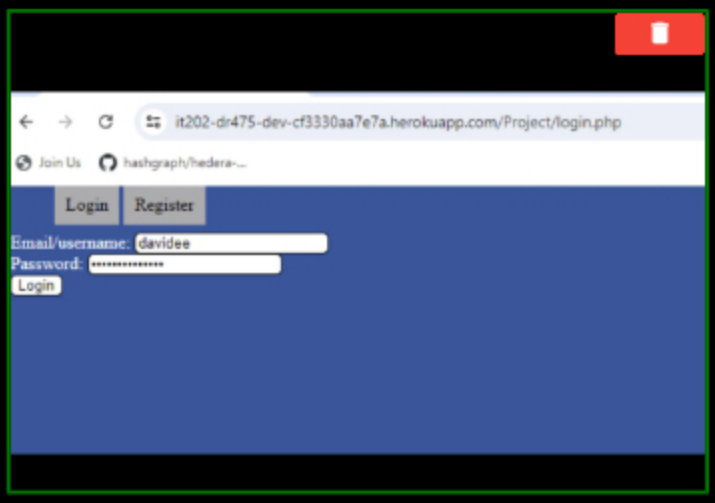
Task #1 - Points: 1

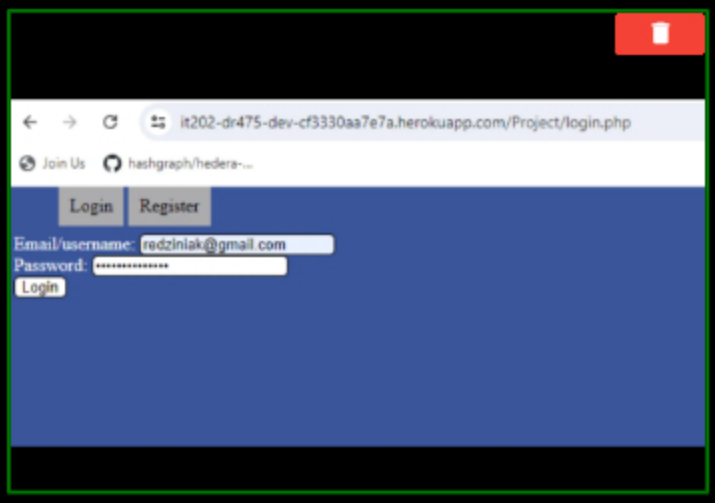
Text: Screenshot of form on website page

Details:

Thoughtful CSS should be applied to all parts (must differ from the "ugly" CSS given via the lessons).
The Heroku dev URL must be present in all screenshots of the site.

#1) Two Screenshot of the form (one with valid email data filled and one with valid username data filled)

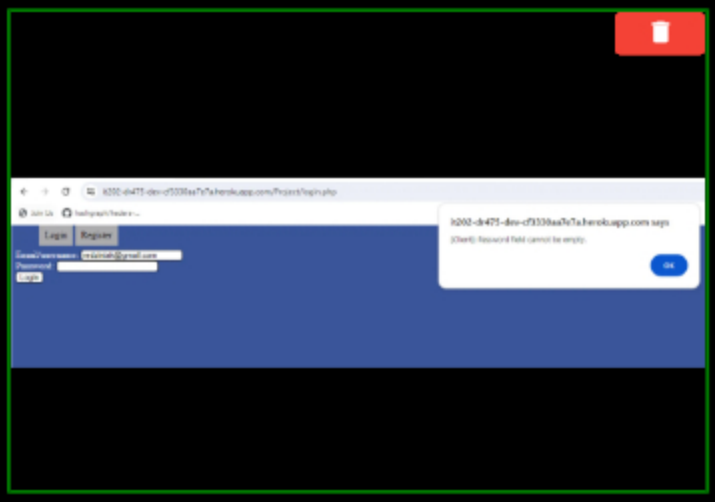


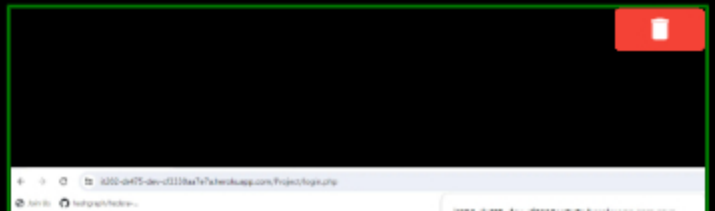


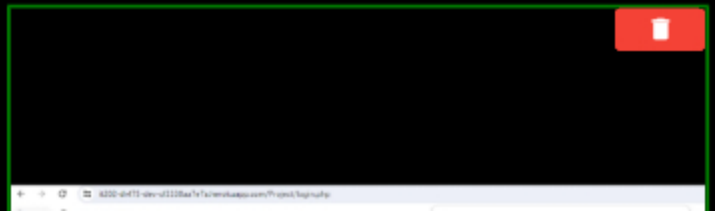
Caption (required) ✓
Describe/highlight what's being shown
Valid form data(user/email)

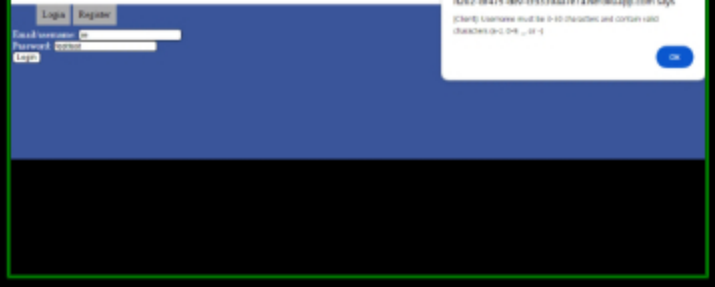
URL (required) ✓
Prod link to the login page
https://it202-dr475-prod-7559be2e2ad4.herokuapp.com/Project/login.php

#2) Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)







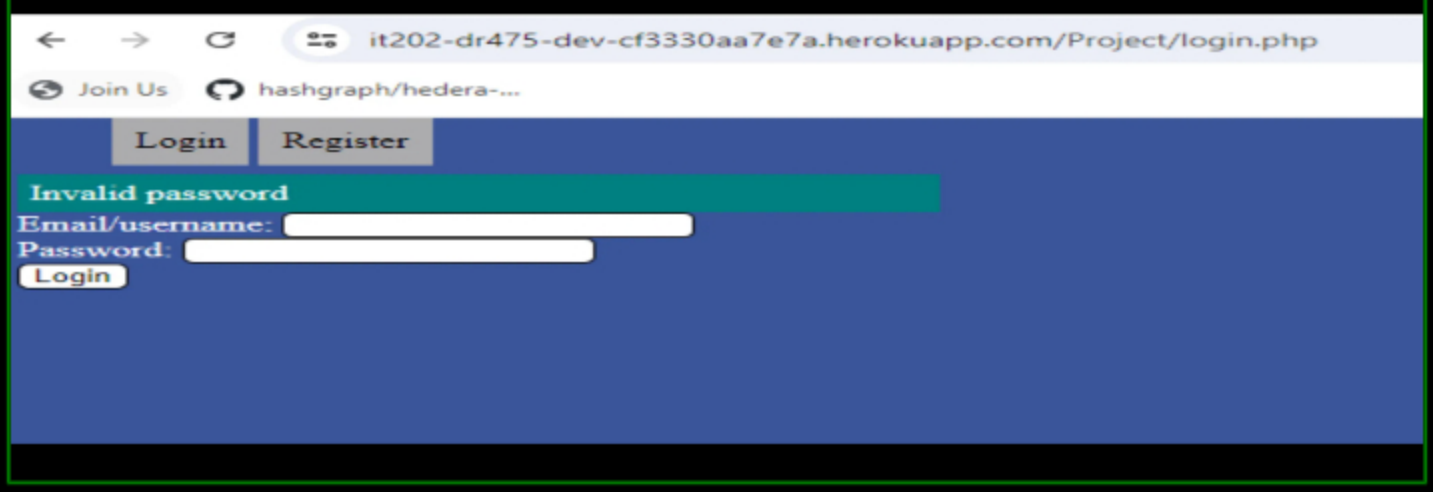


Caption (required) ✓
Describe/highlight what's being shown
Client side validation

#3) Demonstrate user-friendly message of when an account doesn't exist

Caption (required) ✓
Describe/highlight what's being shown
Account not found

#4) Demonstrate user-friendly message of when password doesn't match what's in the DB

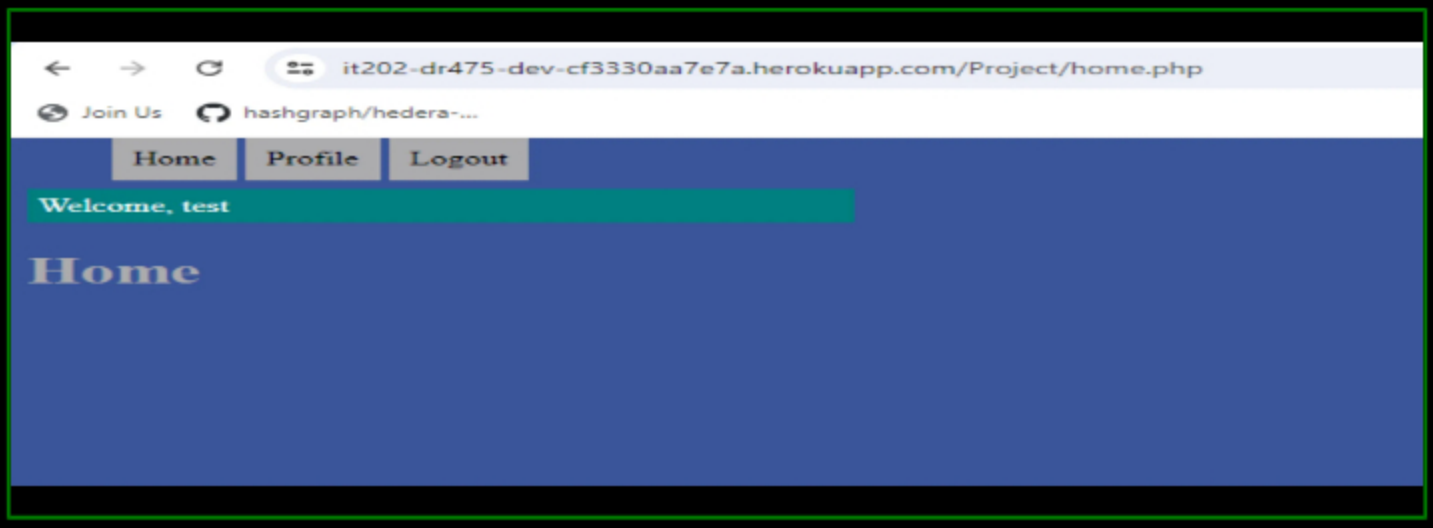


Caption (required) ✓

Describe/highlight what's being shown

Password incorrect

#5) Demonstrate successful login message and the destination page (i.e., home or some landing/dashboard page)



Caption (required) ✓

Describe/highlight what's being shown

Successful login

#6) Demonstrate session data being set (captured from server logs)

Caption (required) ✓
Describe/highlight what's being shown
Heroku logs

COLLAPSE

Task #2 - Points: 1

Text: Screenshot of the form code

Details:
Should have the appropriate type attributes for the fields.
Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

#1) Show the html of the form

```
?>
<!-- UCID: dr475 -->
<!-- Date: 07/08/2024 -->
<form onsubmit="return validate(this)" method="POST">
    <div>
        <label for="email">Email/username:</label>
        <input type="text" name="email" required />
    </div>
    <div>
        <label for="pw">Password:</label>
        <input type="password" id="pw" name="password" required minlength="8" />
    </div>
    <input type="submit" value="Login" />
</form>
```

Caption (required) ✓
Describe/highlight what's being shown
HTML code for form

Explanation (required) ✓
Briefly explain the html for each field including the chosen attributes

PREVIEW RESPONSE

Email/Username - Input box for the email or username of type "text"; it is required; the name="email" so it can be targeted using a query selector.
Password - Input box for the password of type "password" so input value is not visible; it is required with a minimum

value of 8 characters; the name="password" so it can be targeted using a query selector.

#2) Show the JavaScript validations of the form (include any extra files related if you made separate files)

```
<script>
// UCID: dr47s
// Date: 07/09/24
    function validate(form) {
        var emailOrUser = document.querySelector('[name="email"]').value;

        // Check if email or username is empty
        if (emailOrUser === ""){
            alert("[Client]: Email/username field cannot be empty.");
            return false;
        }

        // If email, check if email is valid using regex
        if (emailOrUser.includes("@")){
            if (!/^[a-zA-Z0-9_.]{1,64}@[a-zA-Z0-9_.]{1,64}$/i.test(emailOrUser)){
                alert("[Client]: " + emailOrUser + " is invalid.")
                return false;
            }
        } else {
            // Username validation
            if (!/^[a-zA-Z0-9_-]{3,30}$/.test(emailOrUser)){
                alert("[Client]: Username must be 3-30 characters and contain valid characters (a-z, 0-9, _, or -)");
                return false;
            }
        }

        var pass = document.querySelector('[name="password"]').value;

        // Check if password is empty
        if (pass === ""){
            alert("[Client]: Password field cannot be empty.");
            return false;
        }

        // Check password length
        if (pass.length < 8){
            alert("[Client]: Password length cannot be less than 8 characters.");
            return false;
        }

        return true;
    }
</script>
```

Caption (required) ✓

Describe/highlight what's being shown

JS client side validation

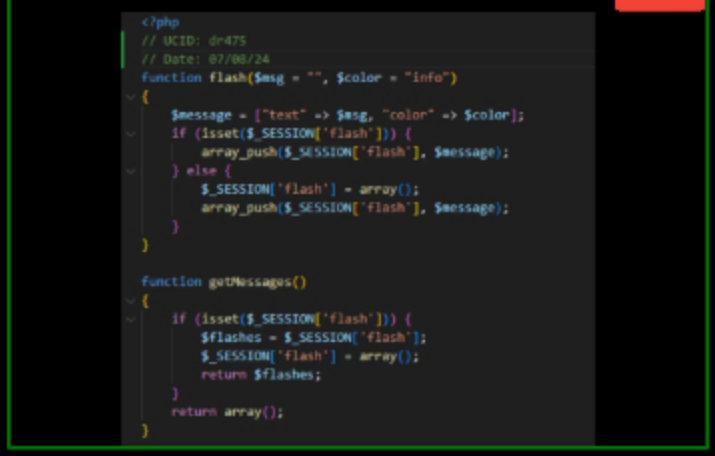
Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

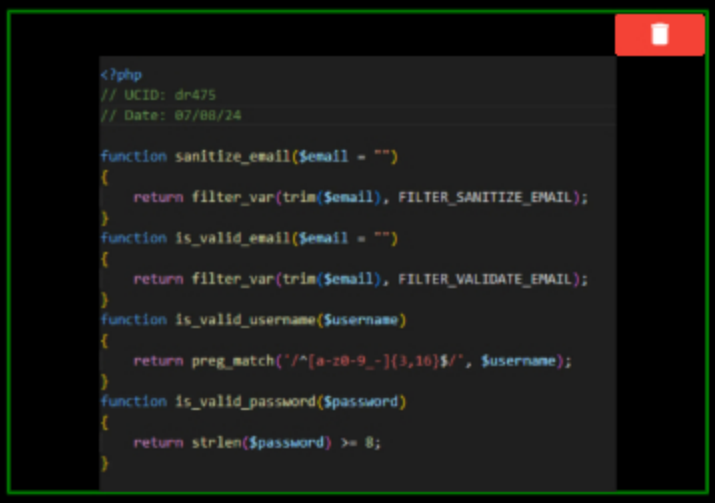
The HTML form calls the validate function which will return false (and wont send a request to the server) if any of the input fields are not valid. The function goes through the email/username and uses Regex to validate the length and characters in the input values, and check if they're empty. It also goes through the password value and checks if they're the required length. It ultimately returns true when there are no errors found.

#3) Show PHP validations (include any lib content)



```php
<?php
// UCID: dr475
// Date: 07/08/24
function flash($msg = "", $color = "info")
{
    $message = ["text" => $msg, "color" => $color];
    if (isset($_SESSION['flash'])) {
        array_push($_SESSION['flash'], $message);
    } else {
        $_SESSION['flash'] = array();
        array_push($_SESSION['flash'], $message);
    }
}

function getMessages()
{
    if (isset($_SESSION['flash'])) {
        $flashes = $_SESSION['flash'];
        $_SESSION['flash'] = array();
        return $flashes;
    }
    return array();
}
```



```php
<?php
// UCID: dr475
// Date: 07/08/24

function sanitize_email($email = "")
{
    return filter_var(trim($email), FILTER_SANITIZE_EMAIL);
}
function is_valid_email($email = "")
{
    return filter_var(trim($email), FILTER_VALIDATE_EMAIL);
}
function is_valid_username($username)
{
    return preg_match('/^[a-z0-9_-]{3,16}$/', $username);
}
function is_valid_password($password)
{
    return strlen($password) >= 8;
}
```

Caption (required) ✓

Describe/highlight what's being shown

PHP Code and it's required lib functions

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

First, login.php imports required functions from lib files. Then, it checks if the client sent a POST request with the required fields (email, password). It then validates and sanitizes the email/username by calling helper functions to verify the validity of the input values. It also then validates the password length and values. Ultimately, if there are not errors, it submits a SELECT SQL query locating the user row with the given parameters (email/username). It then checks the password sent by the user and compares it to the unhashed password stored in the database if it matches. Upon successful authentication, tries to check the user's role in the DB and then sets the session user role variable so it can be used in future requests, then redirects the client to the home.php page. If any errors occur it displays the error on the client's page (invalid password, email not found, etc).

SE/safer_echo = Takes in an array, key, default value and returns the value from the array if it exists or the default provided value.
flash() = Shows the user a provided message with defined styling
sanitization = Methods for username, email and password to trim and validate a string to specifications
getDB() = Provides a connection to the DB given a connection string in .env

Task #3 - Points: 1

COLLAPSE

Text: Explain the login logic step-by-step from when the page loads to when the data is fetched from the DB and stored in the session

Details:
Don't just show code, translate things to plain English in concise steps.
Explain how the session works and why/how it's used

May be worthwhile using a list.

Response:

1. The user retrieves the login.php page from the server.
2. The user is then presented with the form that is required to fill out in order to login
3. The user then fills out all the input fields (email/username, password) and presses the submit button.
4. The JS function validate() is then called to provide client side validation of the fields before it is officially sent to the PHP server.
5. If any of the fields are not valid, it tells the user. If it is all valid, it is then sent to via a POST request to the PHP server for processing.
6. The PHP code in login.php then processes the data and provides extra server-side input validation (lengths of all input text, checks if email/username is valid, password is valid).
7. If everything passes validation, the user is successfully authorized and the server gets the user role given to the user (admin) and saves it to the user's session variable for future page visits. The user is then redirected to the home page.
8. Furthermore, if there are any errors when processing the query, it is then returned to the client.

COLLAPSE

Task #4 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/33

URL #2
https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/46

URL #3
https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/48

User Logout (1 pt.)

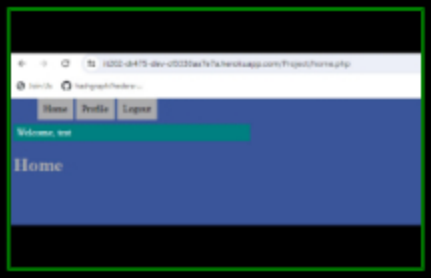
COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Capture the following screenshots

#1) Screenshot of the navigation when logged in (site)

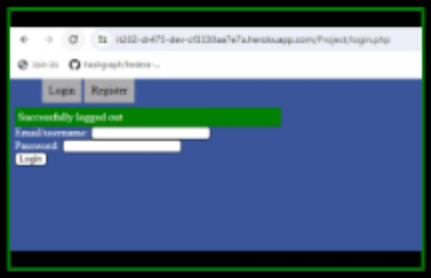


Caption (required) ✓

Describe/highlight what's being shown

User logged in

#2) Screenshot of the redirect to login with the user-friendly



Caption (required) ✓

Describe/highlight what's being shown

User logged out

#3) Screenshot of the logout-related code showing the session is

Caption (required) ✓

Describe/highlight what's being shown

PHP code for logout

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

When the logout.php is visited, The server destroys the current session of the user (session unset, session destroy), and starts a new one (session start). There are no more session variables associated with the respective user. Then, it redirects to the login.php page (header("Location: login.php"))

COLLAPSE

Task #2 - Points: 1

Text: Include pull request links related to this feature

Details:

Should end in /pull/#

URL #1

https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/21

URL #2

https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/25

URL #3

https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/23

Basic Security Rules and Roles (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Authentication Screenshots

Details:
Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

#1) Screenshot of the function that checks if a user is logged in

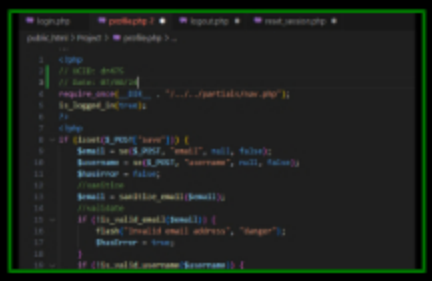
Caption (required) ✓
Describe/highlight what's being shown
PHP function is_logged_in

Explanation (required) ✓
Explain in concise steps how this logically works

PREVIEW RESPONSE

The is_logged_in function checks if the $SESSION['user'] variable is set (it is set when a user logs in). If it is not, that means the user is not logged in, it redirects the user to the login.php page (or a different URL given in the parameters to the function) and displays a message displaying that they aren't logged in.

#2) Screenshot of the login check function being used (i.e., profile



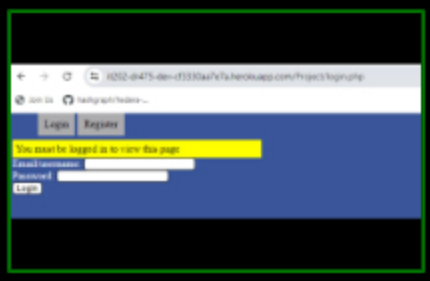
Caption (required) ✓
Describe/highlight what's being shown
is_logged_in being used in profile.php

Explanation (required) ✓
Explain in concise steps how this logically works

PREVIEW RESPONSE

Before executing the rest of the PHP on the page, it first calls the function is_logged_in, which will redirect the user if they aren't logged in. If is_logged_in returns true, the rest of the following PHP code will execute.

#3) Demonstrate the user-friendly message of trying to manually



Caption (required) ✓
Describe/highlight what's being shown
Visiting profile.php while logged out.

COLLAPSE

Task #2 - Points: 1

Text: Authorization Screenshots

Details:
Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

#1) Screenshot of the

Caption (required) ✓
Describe/highlight what's being shown
PHP code

Explanation (required) ✓
Explain in concise steps how this logically works

PREVIEW RESPONSE

The has_role function checks if the user is logged in and checks if a variable is set in the session variables given a specified role parameter (e.g. "Admin")

#2) Screenshot of the role

Caption (required) ✓
Describe/highlight what's being shown
PHP code

Explanation (required) ✓
Explain in concise steps how this logically works

PREVIEW RESPONSE

The PHP logic checks if the user has the role "Admin" in the session variables. If it does have the variable set, it renders an additional list of links to visit in navigation bar. If it doesn't, it isn't rendered.

#3) Demonstrate the user-

Caption (required) ✓
Describe/highlight what's being shown
Visiting admin/list_roles.php as a regular user

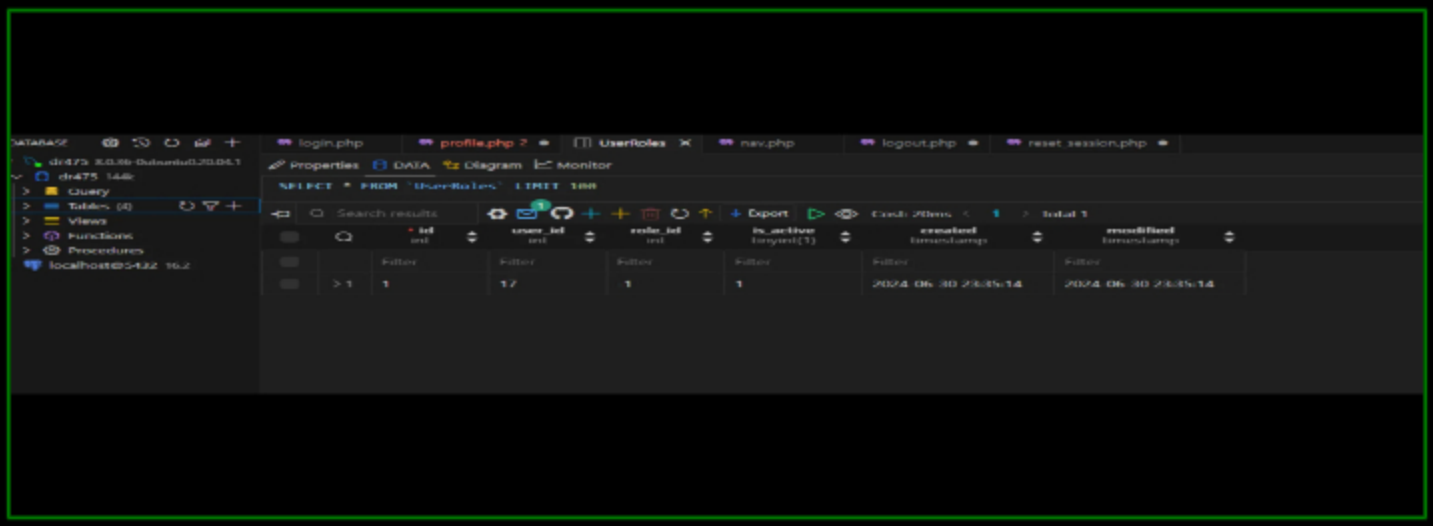
COLLAPSE

Task #3 - Points: 1

Text: Screenshots of UserRoles and Roles Tables

Details:
Ensure left panel or database name is present in each table screenshot (should contain your ucid)

#1) UserRoles table with at least one valid entry (Table should have id, user_id, role_id, is_active, created, and modified columns)

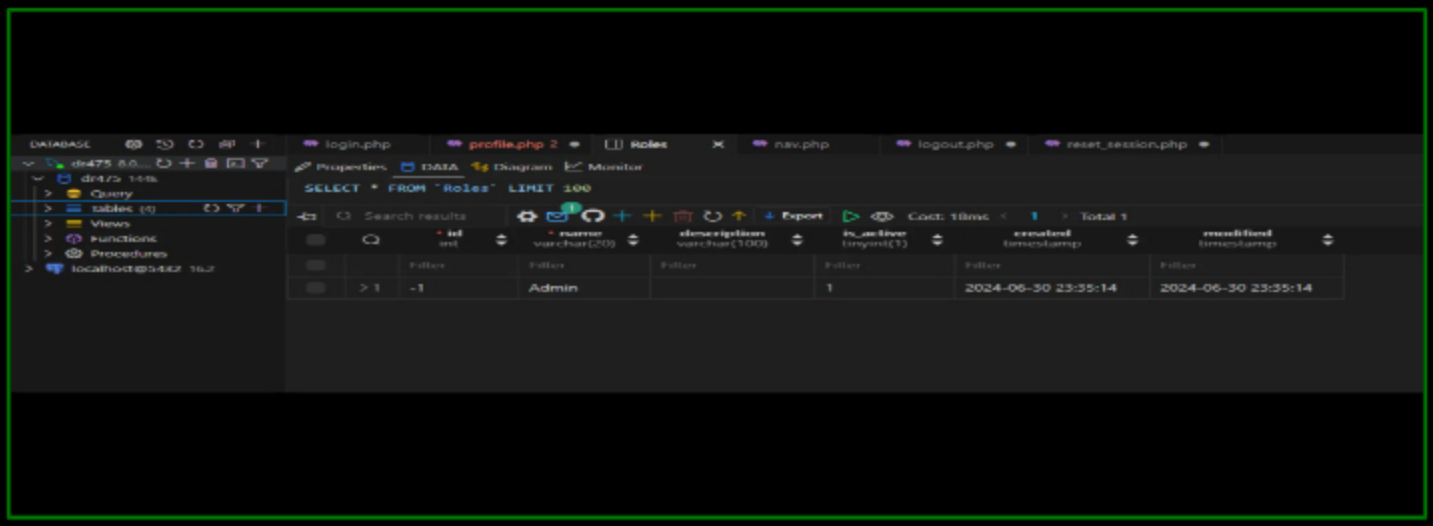


Caption (required) ✓

Describe/highlight what's being shown

User with ID "17" has the role Admin (-1)

#2) Roles table with at least one valid entry (Table should have id, name, description, is_active, modified, and created columns)



Caption (required) ✓

Describe/highlight what's being shown

Admin role with id "-1"

COLLAPSE

Task #4 - Points: 1

Text: Explain how Roles and UserRoles tables work in conjunction with the Users table

#1) UserRoles

Explanation (required) ✓

What's the purpose of the UserRoles table?

PREVIEW RESPONSE

To create a list of users and their respective roles on the website.

#2) Roles

Explanation (required) ✓

How does Roles.is_active differ from UserRoles.is_active?

PREVIEW RESPONSE

Roles.is_active determines whether the role itself is still used in day to day functionality on the website (it could be a defunct role "0"). UserRoles.is_active pertains to the specific user's role on the website (if they are "Admin", it can be toggled off so they are not active as an "Admin")

COLLAPSE

Task #5 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/22

URL #2
https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/32

User Profile (2 pts.)

COLLAPSE

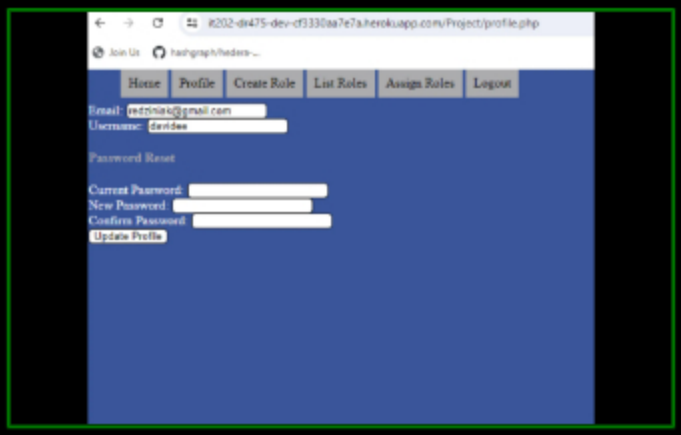
COLLAPSE

Task #1 - Points: 1

Text: View Profile Website Page

Details:
Thoughtful CSS should be applied to all parts (must differ from the "ugly" CSS given via the lessons).
The Heroku dev URL must be present in all screenshots of the site.

#1) Show the profile form correctly populated on page load (username, email) Form should have the following fields:



Caption (required) ✓

Describe/highlight what's being shown

User profile with pre-filled email and username

COLLAPSE

Task #2 - Points: 1

Text: Explain the logic step-by-step of how the data is loaded and populated when the profile page is visited

Details:

Don't just show code, translate things to plain English in concise steps.

May be worthwhile using a list.

Response:

1. The client requests the profile.php page from the server. It is only returned if the user is logged in, else it redirects to the login page.
2. Upon the successful page load, the form is populated with the email and username that are both retrieved using helper functions that access the sessions saved variables (email, username).
3. The profile.php page is then returned to the client with the html and JS included.

COLLAPSE

Task #3 - Points: 1

Text: Edit Profile Website Page

Details:

Thoughtful CSS should be applied to all parts (must differ from the "ugly" CSS given via the lessons).
The Heroku dev URL must be present in all screenshots of the site.

#1) (Two Screenshots) Demonstrate

Caption (required) ✓
Describe/highlight what's being shown
Username change

#2) Demonstrate the success

Caption (required) ✓
Describe/highlight what's being shown
Password changed

#3) Demonstrate all

Caption (required) ✓
Describe/highlight what's being shown
JS validation

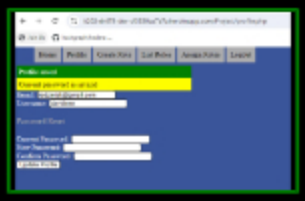
#4) Demonstrate PHP user-

Caption (required) ✓
Describe/highlight what's being shown
Email not available

#5) Demonstrate PHP user-

Caption (required) ✓
Describe/highlight what's being shown
Username in use

#6) Demonstrate PHP user-



Caption (required) ✓
Describe/highlight what's being shown
Password is not correct

^COLLAPSE^

Task #4 - Points: 1
Text: Explain the logic step-by-step of how the data is checked and saved for the following scenarios

Details:
Don't just show code, translate things to plain English

#1) Updating Username/Email

#2) Updating password

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

Upon the form submit, the client sends a POST request to the server. In profile.php, it then sanitizes the email and username and validates the input. Then, it calls the helper function get_user_id which returns the user ID in the session corresponding to the id field in the database for the specified user. This user ID is then used to submit an UPDATE SQL query to the DB WHERE id = get_user_id(). This functionality allows the user to change both their username and email in one go. Then, it retrieves the updated data to be stored in the SESSION user variable by using a SELECT SQL query WHERE id = user's ID. If any errors occur, it is handled and displayed to the user.

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

Upon the form submit, the client sends a POST request to the server. It checks if all the password fields are set and are valid (current, new, confirm new). Then it retrieves the user's password from the Users table in the DB WHERE id = user's ID. It then uses the password_verify function to check the hash of the password stored in the DB and compares it to the current_pass provided by the user in the POST request. If successful, it submits an UPDATE SQL query into the DB on the Users table which changes the password field to the hashed new password. If there is any error, it is handled (current password is invalid, new passwords don't match) and displayed to the user.

COLLAPSE

Task #5 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/49

URL #2
https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/24

COLLAPSE

Misc (1 pt.)

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of wakatime

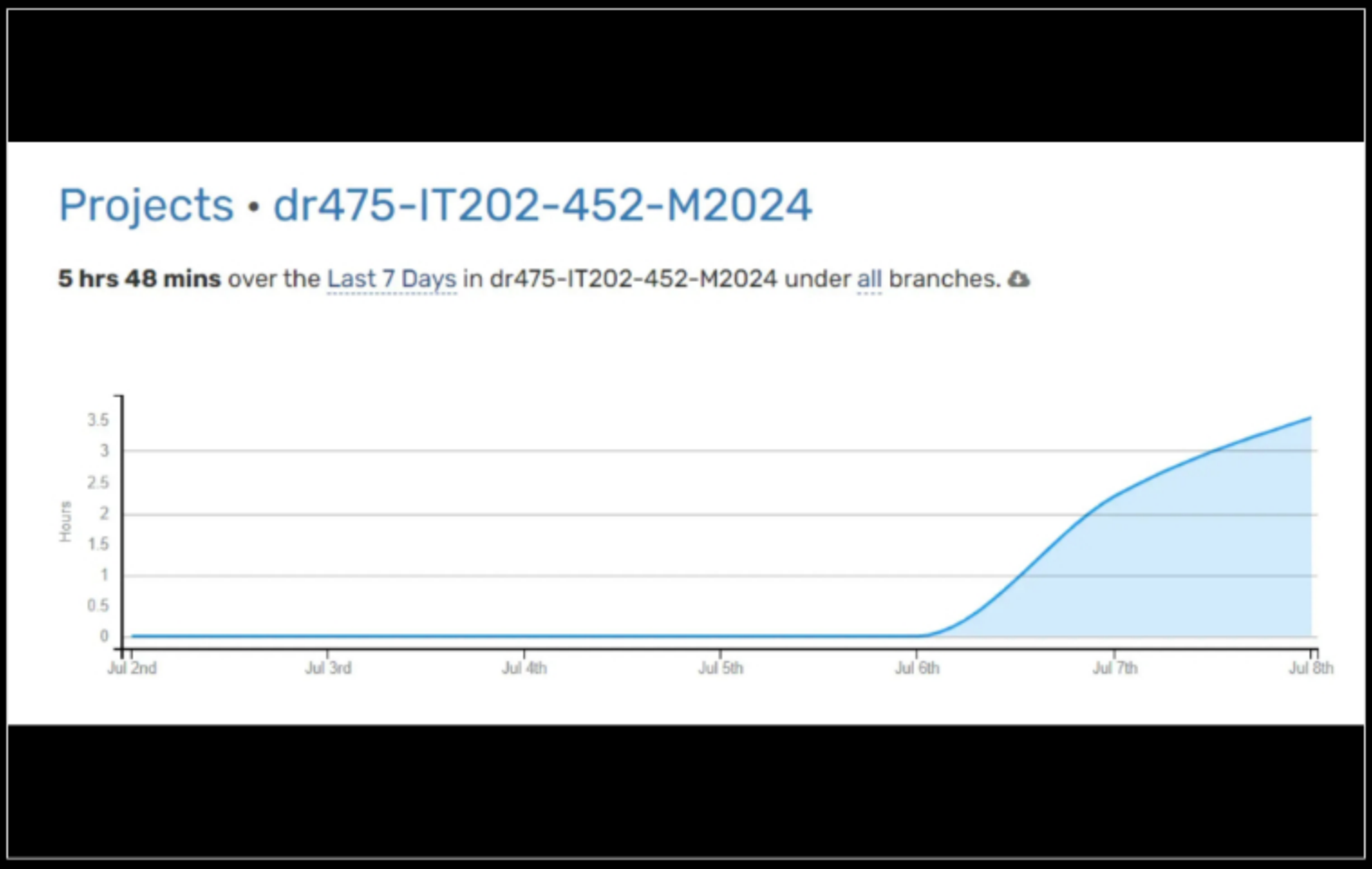
Details:

Note: The duration of time isn't directly related to the grade, the goal is to just make sure time is being tracked

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Wakatime last 7 days working on Milestone 1

COLLAPSE

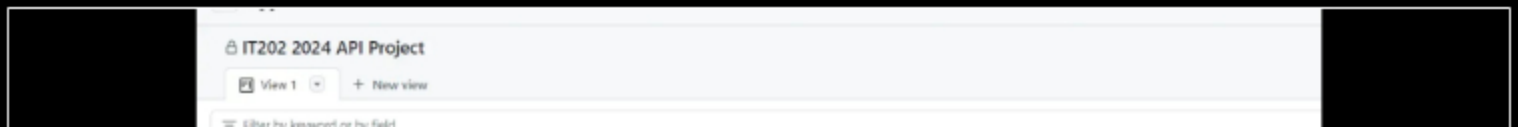
Task #2 - Points: 1

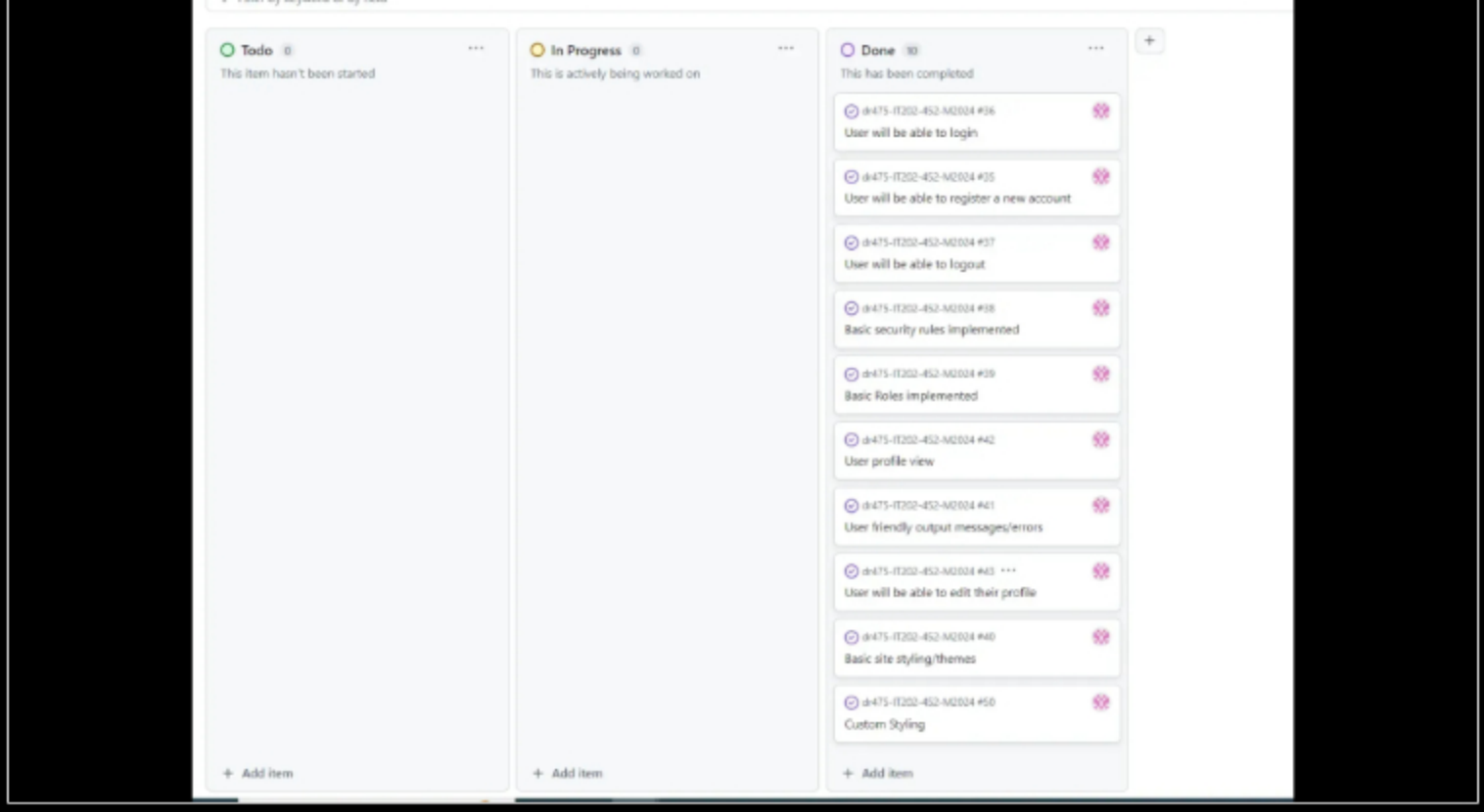
Text: Screenshot of your project board from GitHub (tasks should be in the proper column)

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Github project board

COLLAPSE

Task #3 - Points: 1

Text: Provide a direct link to the project board on GitHub

URL #1

https://github.com/users/davidredziniak/projects/2

End of Assignment