# Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-452-M2024/it202-module-6-milestone-1-2024/grade/vs53

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

Submissions:

Submission Selection

1 Submission [active] 7/10/2024 3:31:08 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 ^

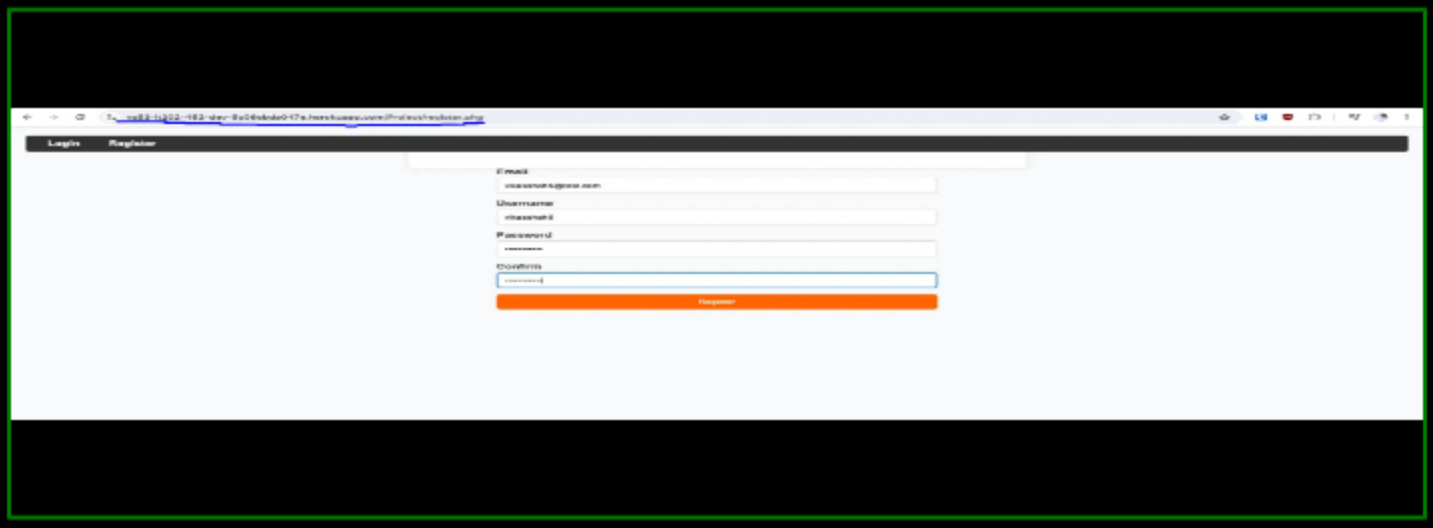
^COLLAPSE ^

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*

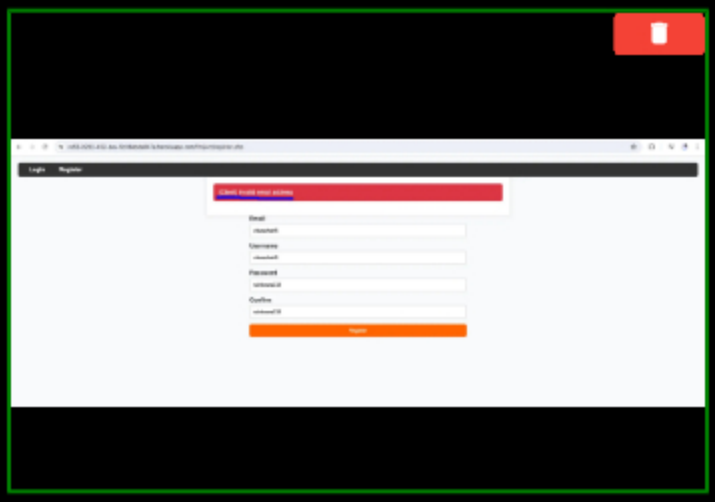
I'm showing my register.php form and the page, showing that I am asking for the Email, Username,and password with valid data.

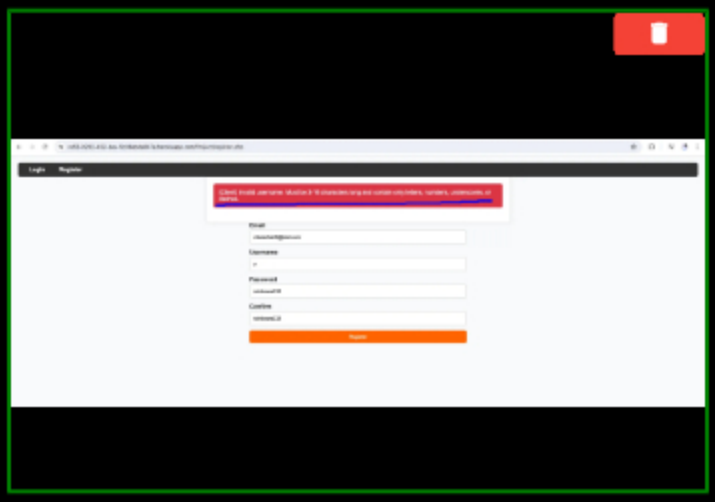
**URL (required)** ✓

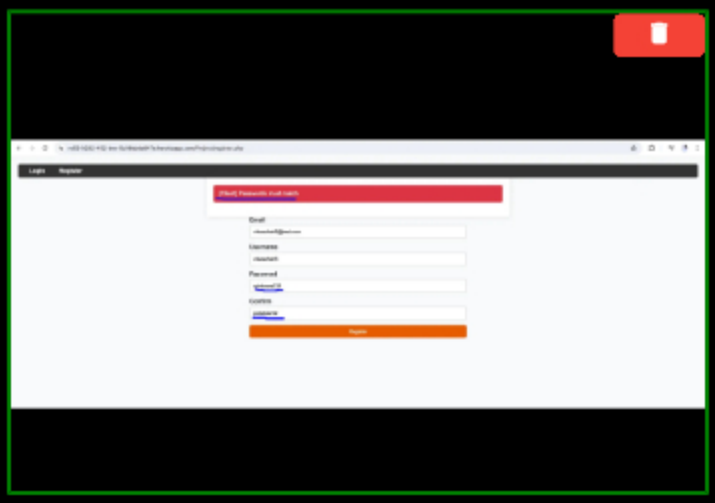
*Prod link to the registration page*

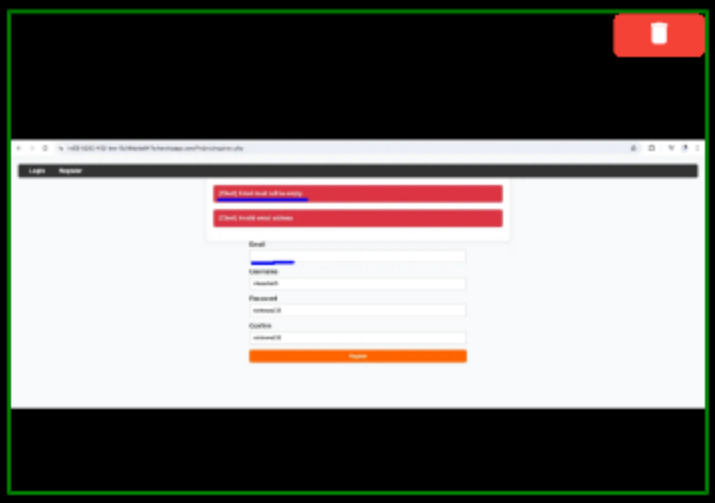
https://vs53-it202-452-prod-1c77a1c304b2.herokuapp.com/Project/register.php

#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







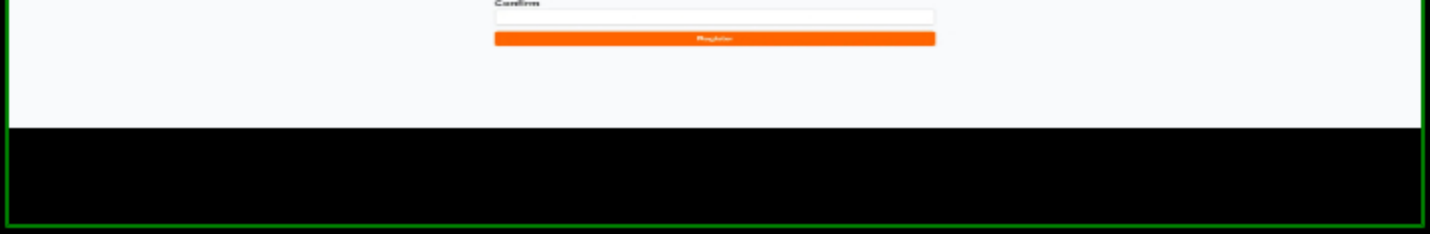


**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing JS validation,I ran the script to disable HTML validation. I showed JS validation for email,username,password,and fields

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

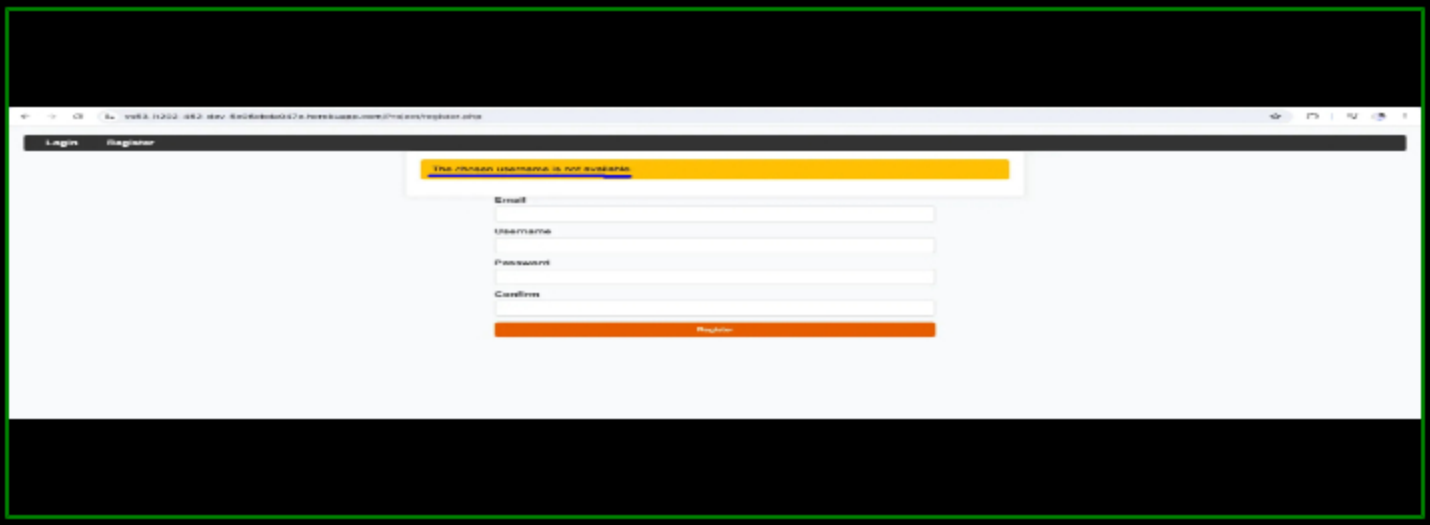


**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing that email is not available because its already in use

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

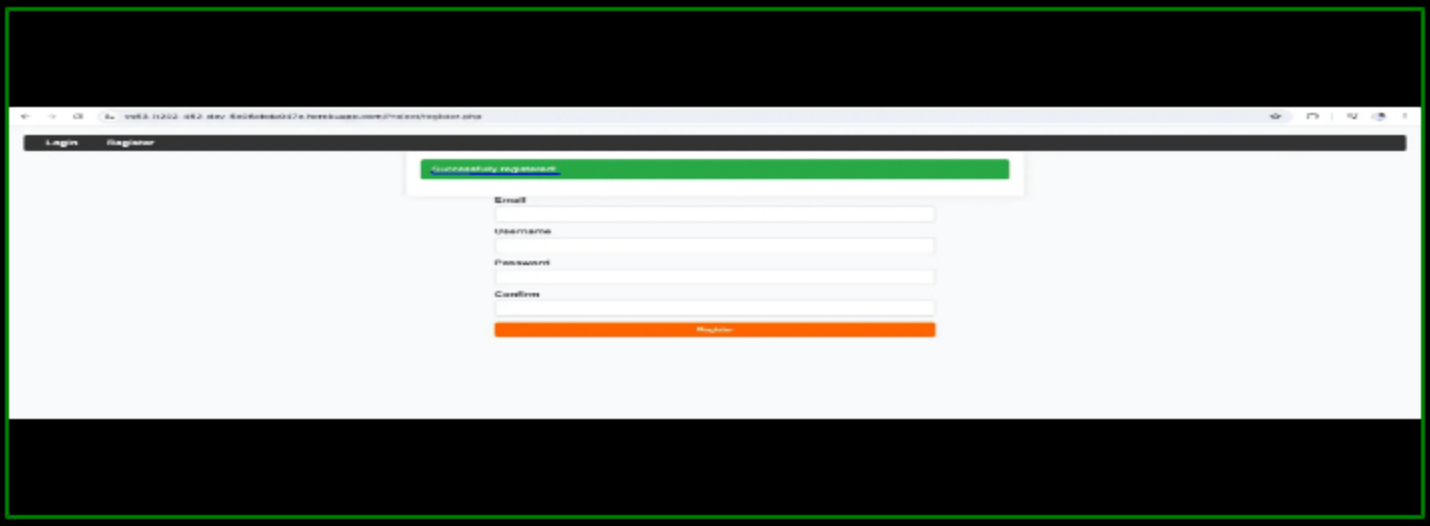



**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing that the username is not available because its already in use

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



**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing a green box showing that a new user has been created

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: vs53, Date: Jul 10th 2024 -->
    You, now • Uncommitted changes
<form onsubmit="return validate(this)" method="POST">
    <div>
        <label for="email">Email</label>
        <input type="email" name="email" required />
    </div>
    <div>
        <label for="username">Username</label>
        <input type="text" name="username" 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>
<script>
```

**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing a screenshot of html code for the form, with my UCID and the date included.

**Explanation (required)** ✓

*Briefly explain the html for each field including the chosen attributes*

PREVIEW RESPONSE

The email field includes a label and an input with type="email", requiring the user to enter a valid email address, and it is marked as required. The username field includes a label and an input with type="text", allowing any text input, with a maximum length of 30 characters, and it is also required. The password field includes a label and an input with type="password", which hides the input text, requires at least 8 characters, and is marked as required. The confirm password field includes a label and an input with type="password", asking the user to re-enter the password for confirmation, with the same requirements as the password field. The form includes a submit button with type="submit" and the text "Register" on it

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*

Providing 2 screenshots showing javascript validations with my UCID and Date in the first screenshot

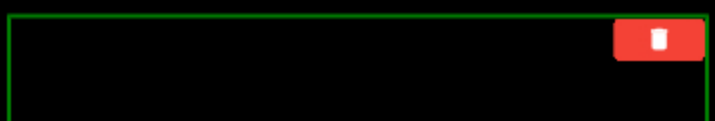
**Explanation (required)** ✓

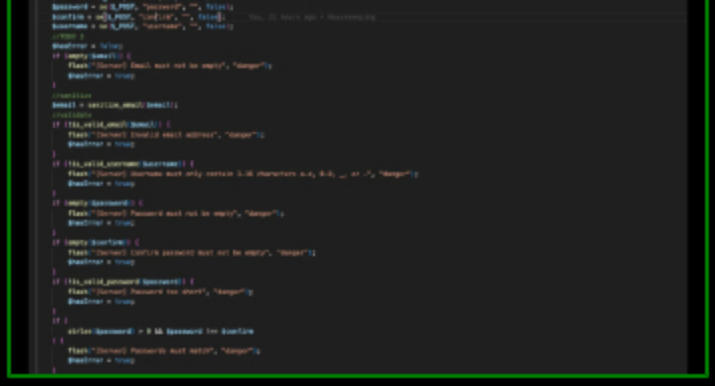
*Explain in concise steps how this logically works*

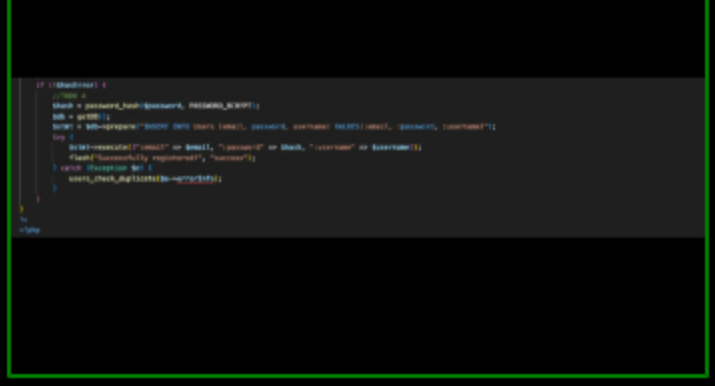
PREVIEW RESPONSE

The JavaScript validation begins when the form is submitted. It retrieves the values of the email, username, password, and confirm password fields. It clears any previous flash messages. It checks if the email is empty and then validates the email format. It checks if the username is empty and validates the username format to be between 3-16 characters with only allowed characters. It checks if the password and confirm password fields are empty. It validates the password length to be at least 8 characters and checks if the password matches the confirm password field. If any validation fails, it displays an appropriate error message and prevents form submission. If all validations pass, the form is allowed to be submitted.

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







**Caption (required)** ✓

*Describe/highlight what's being shown*

Providing 2 screenshots showing PHP validations with my UCID and Date in the first screenshot

**Explanation (required)** ✓

*Explain in concise steps how this logically works*

PREVIEW RESPONSE

The PHP validation starts by checking if all required fields (email, password, confirm, username) are set in the POST request. It sanitizes the email to remove any unwanted characters and then validates it to confirm it is a proper email address. The username is validated to meet the criteria of 3-16 characters and contains only allowed characters. It checks if any fields are empty and adds an error message if they are. The password is validated to confirm it is at least 8 characters long and matches the confirmation password. If there are no errors, the password is hashed for security and the user's data is inserted into the database. If any errors occur during database insertion, they are handled

COLLAPSE

Task #4 - Points: 1

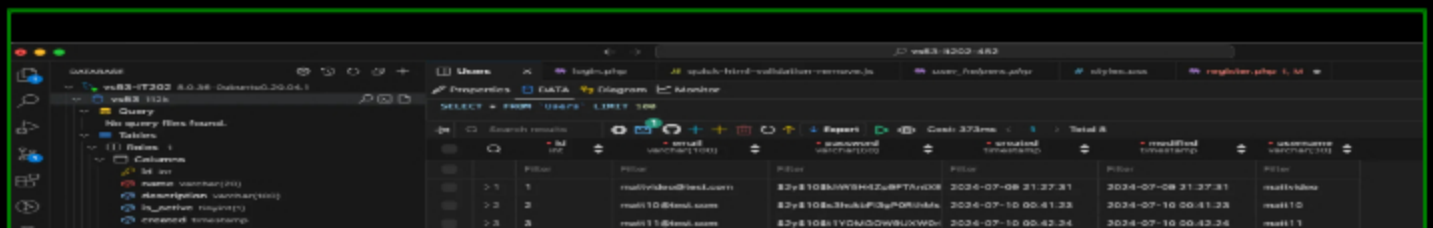
**Text: Screenshot of the Users table with a valid user entry**

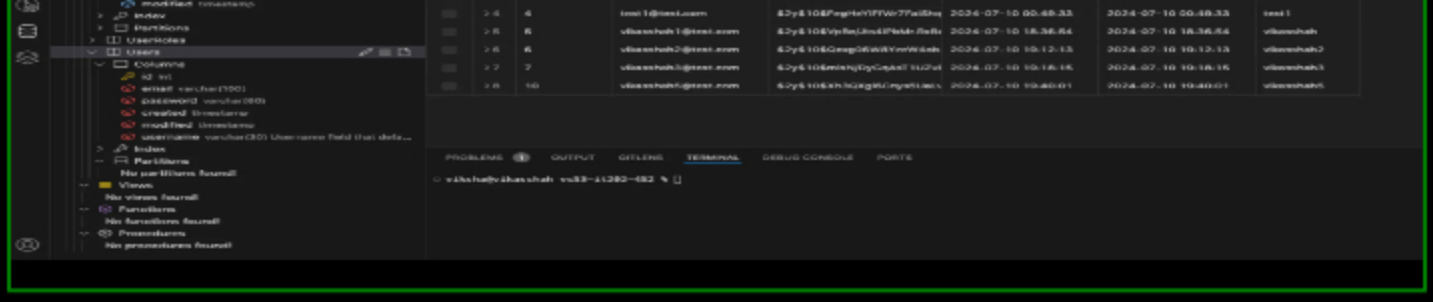
Checklist

*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*

Showing a screenshot of the database, with the password hashed,all the fields, and the left panel showing my UCID (vs53)

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:

When the registration page loads, it first includes necessary PHP files for functions and resets any previous session. The user fills out the registration form with their email, username, password, and confirm password. When the user submits the form, JavaScript validation runs to check for empty fields, valid email and username formats, and matching passwords. If any validation fails, error messages are displayed and the form is not submitted. If the JavaScript validation passes, the form data is sent to the server. PHP then checks if all fields are set and sanitizes the email. It validates the email and username formats and checks for empty fields again. The password is validated for length and compared with the confirm password. If any validation fails, error messages are stored in the session and displayed to the user. If all validations pass, the password is hashed for security. The user's email, hashed password, and username are then inserted into the database. If the insertion is successful, a success message is displayed. If there is a database error, such as a duplicate entry, an appropriate error message is shown to the user.

COLLAPSE

Task #6 - Points: 1

**Text: Include pull request links related to this feature**

Details:

Should end in /pull/#

URL #1

URL

https://github.com/VikShah/vs53-it202-452/pull/16

https://github.com/VikShah/vs53-it202-452/pull/

+ ADD ANOTHER URL

User Login (2 pts.)

^COLLAPSE ^

^COLLAPSE ^

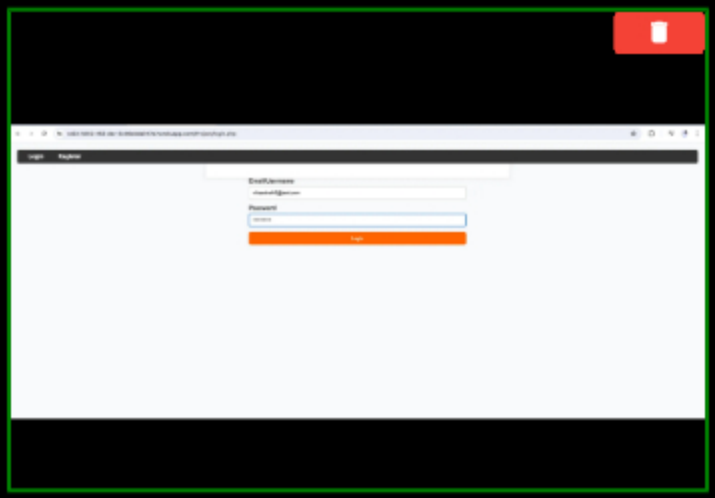
Task #1 - Points: 1

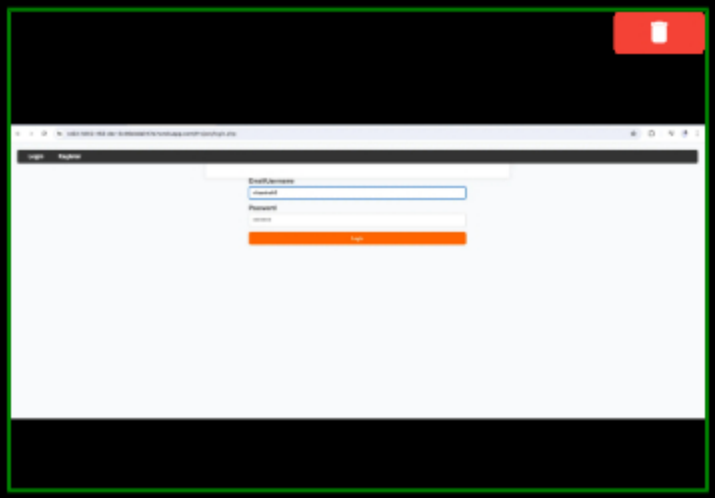
**Text: Screenshot of form on website page**

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) 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*

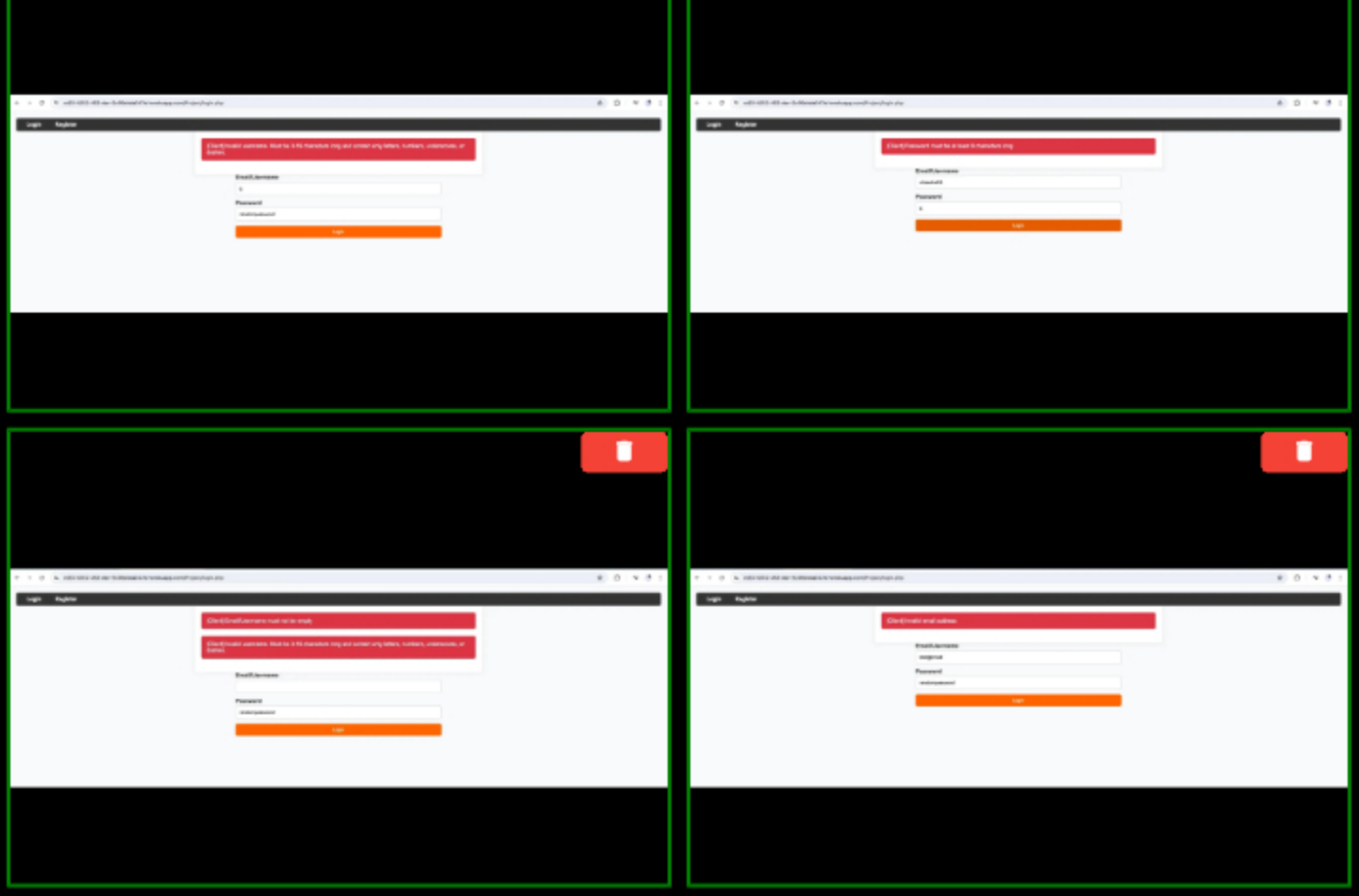
Showing 2 screenshots one with a valid username filled out and one with a valid email filled out.

**URL (required)** ✓

*Prod link to the login page*

https://vs53-it202-452-prod-1c77a1c304b2.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)

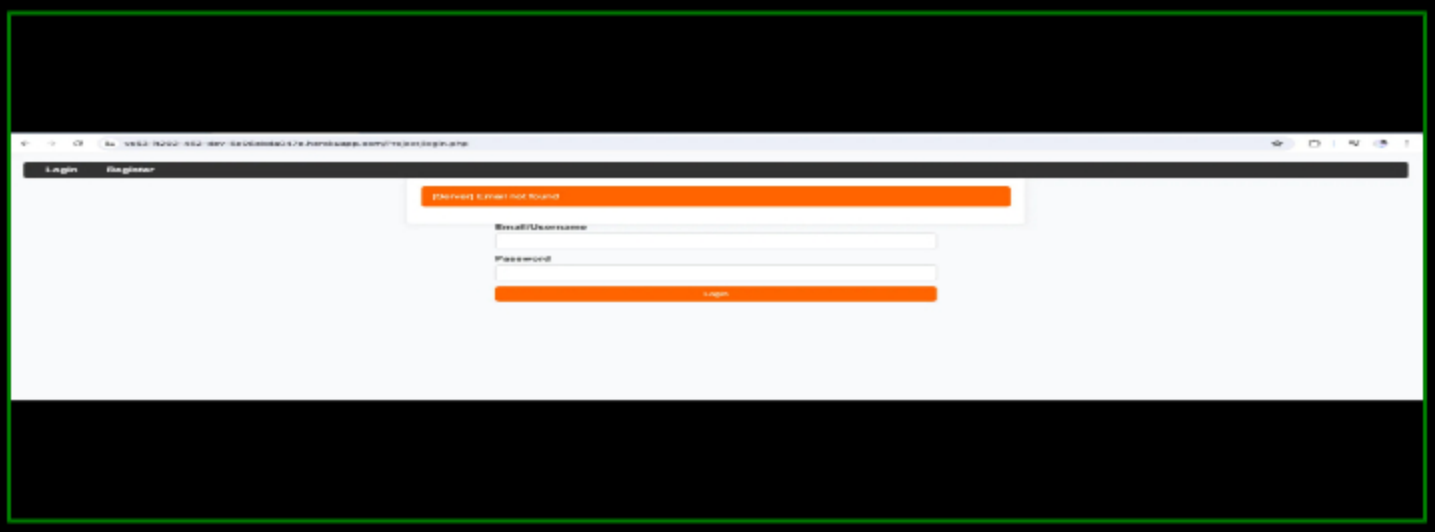


**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing JS validation,I ran the script to disable HTML validation. I showed JS validation for email,username,password,and fields

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

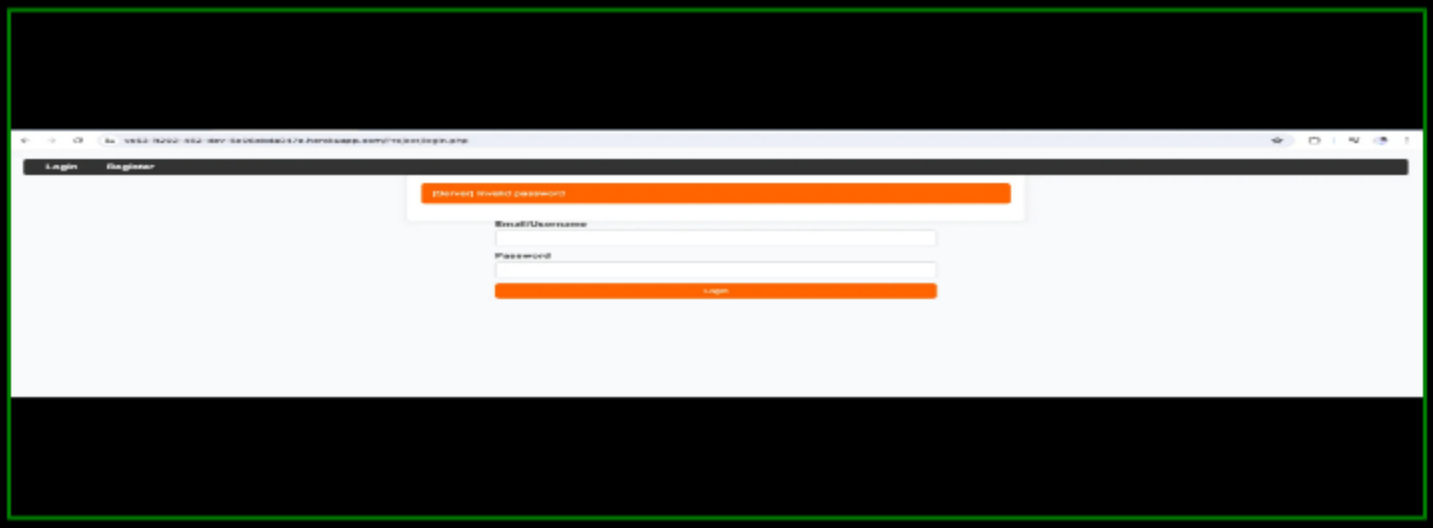



**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing the message when an account doesn't exist

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




**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing the message when the password is incorrect or match whats in the DB

## #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*

Showing what happens when you succesfully login and the destination page

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

**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing the session data being set and captured from the server log in Heroku

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: vs53, Date: Jul 10th 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>        You, yesterday • Added login and nav part 1
    <input type="submit" value="Login" />
</form>
<script>
```

**Caption (required)** ✓

*Describe/highlight what's being shown*

Shownig the html of the form with a comment showing my UCID and the date

**Explanation (required)** ✓

*Briefly explain the html for each field including the chosen attributes*

*Briefly explain the html for each field including the chosen attributes*

PREVIEW RESPONSE

First is the form tag, which includes the onsubmit attribute to call the validate function and the method attribute set to POST. The first field is for email or username, with a label for the email input field, which has the type attribute set to text, allowing any text input, and it is required to be filled. The second field is for the password, with a label for the password input field, which has the type attribute set to password to hide the input characters and a minlength attribute set to 8, making it required to be at least 8 characters long and mandatory to fill. there is a submit button with the type attribute set to submit, allowing the form to be submitted when clicked

## #2) 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*

Showing the javascript validations of the form with a comment showing my UCID and the date

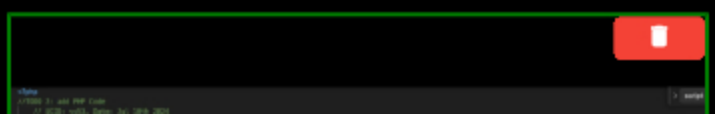
**Explanation (required)** ✓

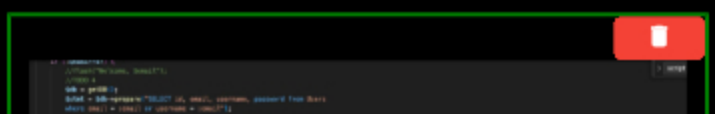
*Explain in concise steps how this logically works*

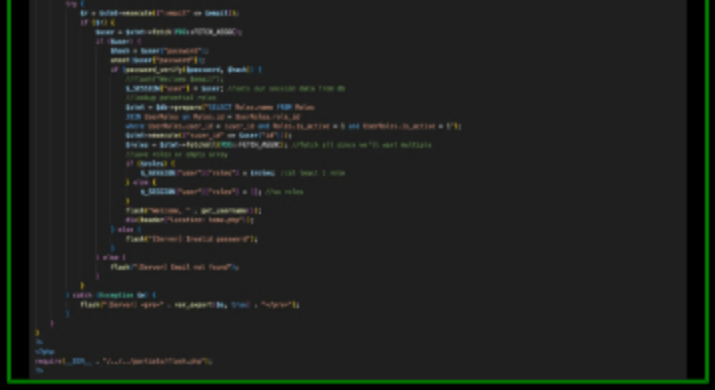
PREVIEW RESPONSE

It starts when the form is submitted. It retrieves the email or username and password values from the form. It clears any previous flash messages. It checks if the email or username field is empty and adds an error message if it is. If the input contains an "@" symbol, it validates it as an email address using a regular expression; otherwise, it validates it as a username with another regular expression. It checks if the password field is empty and if the password is at least 8 characters long, adding error messages if these conditions are not met. If any validation fails, it sets the isValid flag to false and prevents form submission. The addFlashMessage function is used to display the error messages, and helper functions validateEmail and validateUsername are used to verify the input formats are correct.

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







**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing the PHP validations of the form with a comment showing my UCID and the date

**Explanation (required)** ✓

*Explain in concise steps how this logically works*

PREVIEW RESPONSE

First it checks if the email and password fields are submitted. It retrieves the values and sanitizes the email. If the input contains an "@" symbol, it treats it as an email and validates it; otherwise, it validates it as a username. It checks if the email or username and password fields are empty. It also checks if the password is at least 8 characters long. If any validation fails, error messages are displayed and further processing stops. If all checks pass, it queries the database for the user with the given email or username. It then verifies the password with the stored hashed password. If the password is correct, it sets the session data with the user's information and roles and redirects to the home page. If there are database errors, it catches the exceptions and displays the error messages.

COLLAPSE

Task #3 - Points: 1

**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:

When the login page loads, it displays a form for the user to enter their email or username and password. When the form is submitted, JavaScript validation checks for empty fields and validates the email or username format. If validation passes, the form data is sent to the server. The server checks if the email and password fields are set. It sanitizes and validates the email, treating it as an email or username based on its format. It checks if the password is not empty and is at least 8 characters long. If there are no errors, it queries the database for a user with the given email or username. It retrieves the user's data and verifies the entered password with the stored hashed password. If the password matches, it sets session data with the user's ID, email, username, and roles. The session is used to keep

the user logged in across different pages by storing their information on the server and associating it with a unique session ID stored in the user's browser. If the login is successful, the user is redirected to the home page; otherwise, error messages are displayed.

COLLAPSE

Task #4 - Points: 1

**Text: Include pull request links related to this feature**

Details:
Should end in /pull/#

URL #1
https://github.com/VikShah/vs53-it202-452/pull/17

URL
https://github.com/VikShah/vs53-it202-452/pull/

URL #2
https://github.com/VikShah/vs53-it202-452/pull/23

URL
https://github.com/VikShah/vs53-it202-452/pull/

ADD ANOTHER URL

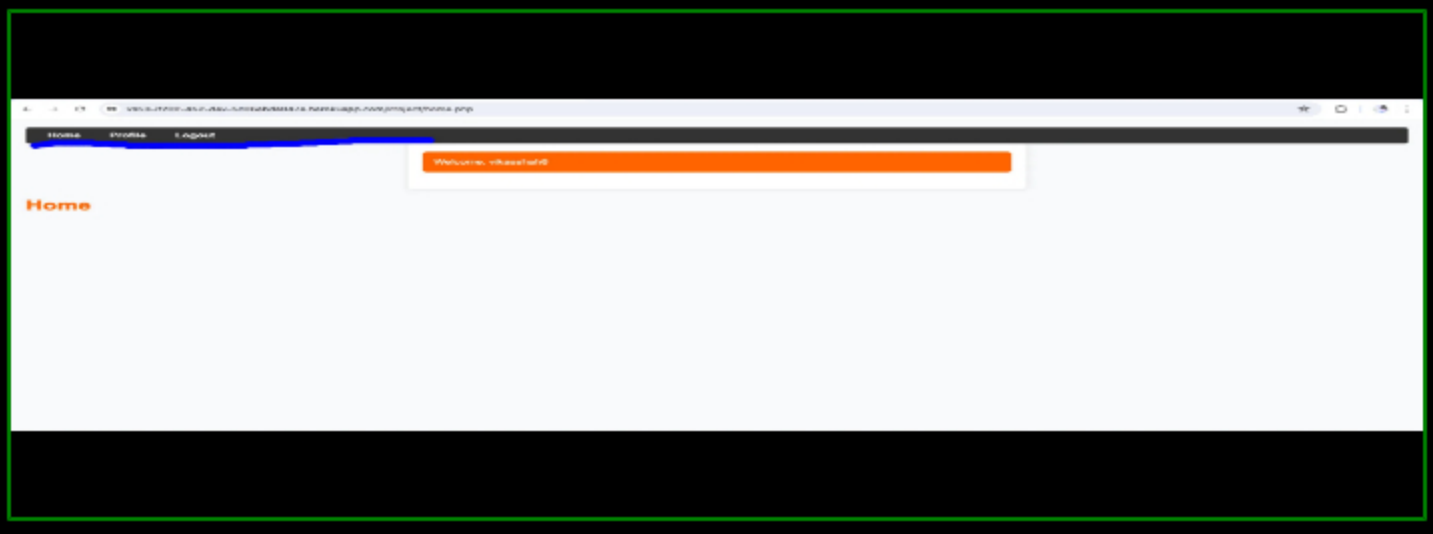
**User Logout** (1 pt.)

COLLAPSE

Task #1 - Points: 1

**Text: Capture the following screenshots**

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

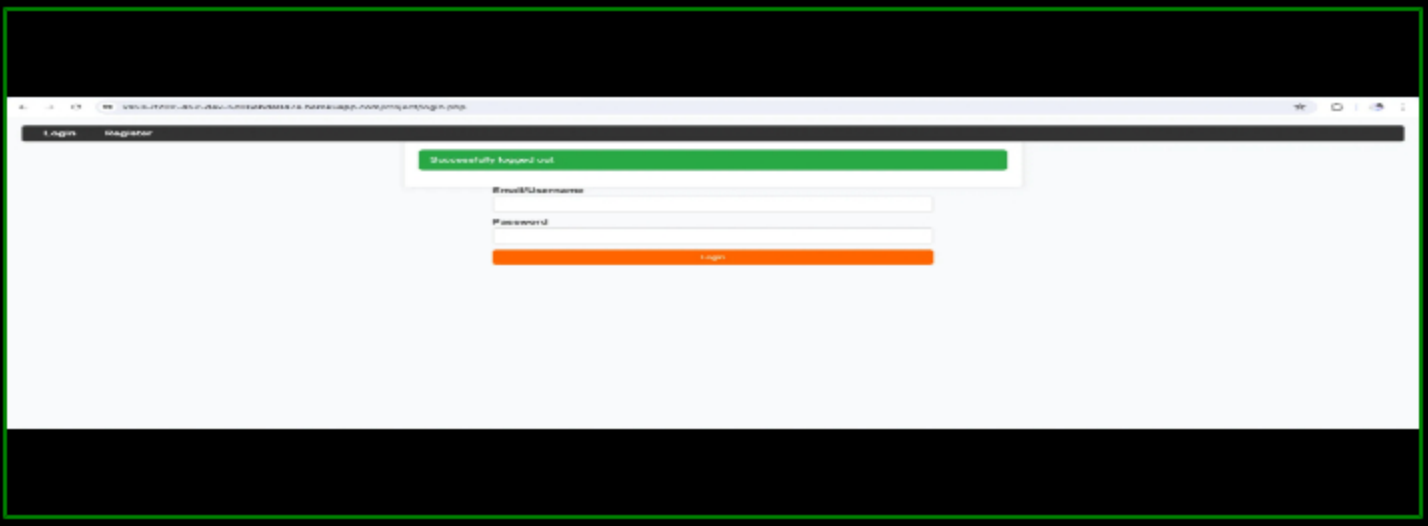


**Caption (required)** ✓

*Describe/highlight what's being shown*

I highighted in the blue the navigation bar when the user is logged in

#2) Screenshot of the redirect to login with the user-friendly logged-out message (site)




**Caption (required)** ✓

*Describe/highlight what's being shown*

Above is the screenshot of what the user sees when a user logs out with a user-friendly logged out message.

#3) Screenshot of the logout-related code showing the session is destroyed (code). Ensure ucid/date comment is present.

```
1  <?php
2  // UCID: vs53, Date: Jul 10th 2024
3  // You, yesterday • Uncommitted changes
4  session_start();
5  require(__DIR__ . "/../../lib/functions.php");
6  reset_session();
7
8  flash("Successfully logged out", "success");
9  header("Location: login.php");
```

**Caption (required)** ✓

*Describe/highlight what's being shown*

A screenshot above showing the logout-related code showing the session is destroyed with my ucid/date included.

**Explanation (required)** ✓

*Explain in concise steps how this logically works*

PREVIEW RESPONSE

It starts with session_start(), allowing access to the current session data. It then includes necessary functions from functions.php. The reset_session() function is called to destroy all session data, logging the user out. A flash message "Successfully logged out" is set to notify the user. And then finally, the script redirects the user to the login page using header("Location: login.php"), completing the logout process. This sequence clears the user's session and navigates them away from restricted pages.

COLLAPSE

Task #2 - Points: 1

**Text: Include pull request links related to this feature**

Details:
Should end in /pull/#

URL #1
https://github.com/VikShah/vs53-it202-452/pull/17

URL
https://github.com/VikShah/vs53-it202-452/pull/

URL #2
https://github.com/VikShah/vs53-it202-452/pull/18

URL
https://github.com/VikShah/vs53-it202-452/pull/

ADD ANOTHER URL

Basic Security Rules and Roles (2 pts.)

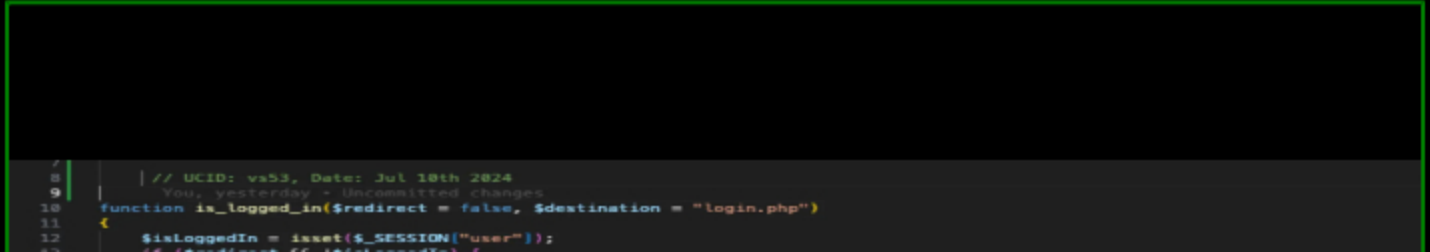
COLLAPSE

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



```php
    if ($redirect && !$isLoggedIn) {
        //if this triggers, the calling script won't receive a reply since die()/exit() terminates it
        flash("You must be logged in to view this page", "warning");
        die(header("Location: $destination"));
    }
    return $isLoggedIn;
}
```

**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing a screenshot of the function that checks if a user is logged in with my UCID and Date as a comment

**Explanation (required)** ✓

*Explain in concise steps how this logically works*

PREVIEW RESPONSE

The is_logged_in function starts by checking if the $ _ SESSION user variable is set, which indicates the user is logged in. If the $redirect parameter is true and the user is not logged in, it sets a flash message saying the user must be logged in to view the page. It then redirects the user to the login page and stops further execution with die(). If the user is logged in, it returns true; otherwise, it returns false.

#2) Screenshot of the login check function being used (i.e., profile likely). Also, caption what pages it's used on

```php
require_once(__DIR__ . "/../../partials/nav.php");
// UCID: ws53, Date: Jul 10th 2024
is_logged_in(true);
?>
<?php
if (isset($_POST["save"])) {
    $email = se($_POST, "email", null, false);
    $username = se($_POST, "username", null, false);
    $hasError = false;
    //sanitize
    $email = sanitize_email($email);
    //validate
    if (!is_valid_email($email)) {
        flash("[Server] Invalid email address", "danger");
        $hasError = true;
    }
    if (!is_valid_username($username)) {
        flash("[Server] Username must only contain 3-16 characters a-z, 0-9, _, or -", "danger");
        $hasError = true;
    }
    if (!$hasError) {
        $params = [":email" => $email, ":username" => $username, ":id" => get_user_id()];
        $db = getDB();
        $stmt = $db->prepare("UPDATE Users set email = :email, username = :username where id = :id");
        try {
            $stmt->execute($params);
            flash("Profile saved", "success");
        } catch (Exception $e) {
            users_check_duplicate($e->errorInfo);
        }
        //select fresh data from table
        $stmt = $db->prepare("SELECT id, email, username from Users where id = :id LIMIT 1");
        try {
            $stmt->execute([":id" => get_user_id()]);
            $user = $stmt->fetch(PDO::FETCH_ASSOC);
            if ($user) {
                //$_SESSION["user"] = $user;
                $_SESSION["user"]["email"] = $user["email"];
                $_SESSION["user"]["username"] = $user["username"];
            } else {
                flash("[Server] User doesn't exist", "danger");
```

**Caption (required)** ✓

*Describe/highlight what's being shown*

Screenshot of the function being used in profile.php. This function is also used on pages like home.php

**Explanation (required)** ✓

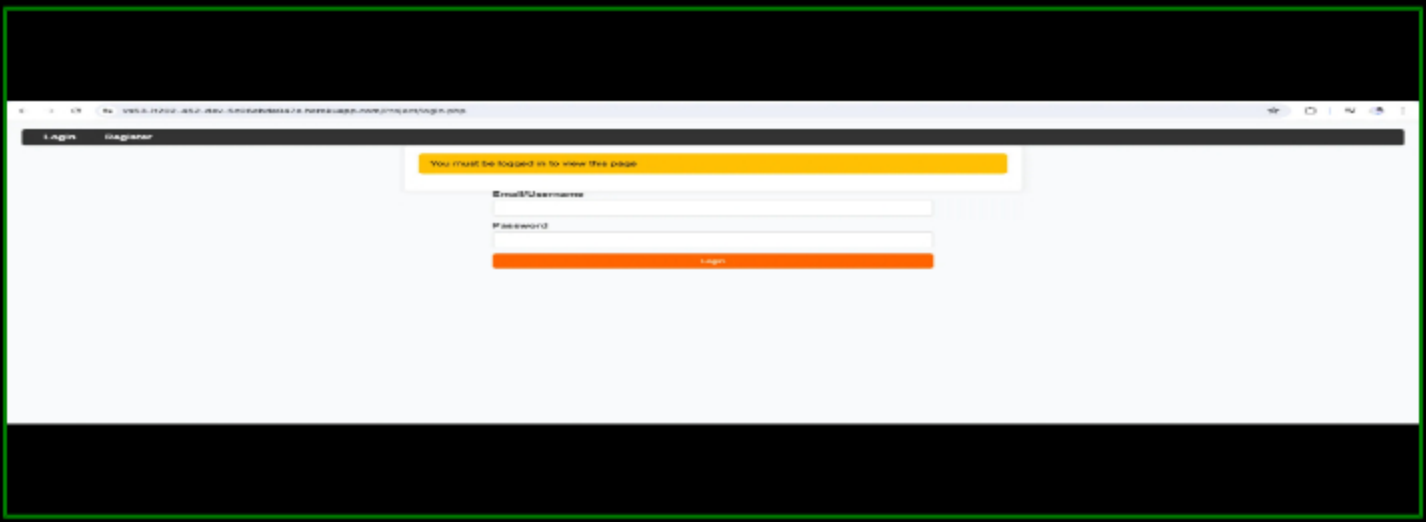
*Explain in concise steps how this logically works*

PREVIEW RESPONSE

The profile.php file starts by including the navigation bar using require_once. It then calls the is_logged_in function with true as the parameter. This checks if the user is logged in by seeing if the $ _ SESSION "user" variable is set. If the user is not logged in, it sets a flash message saying the user must be logged in to view the page, redirects to the login page, and stops further execution. If the user is logged in, the function allows the rest of the code to execute

login page, and stops further execution. If the user is logged in, the function allows the rest of the code to execute. This helps protect the profile page by requiring the user to be logged in to access it.

#3) Demonstrate the user-friendly message of trying to manually access a login-protected page while being logged out (must show the appropriate message)




**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing what happens if you try to access a login-pted page while being logged out, you will see a appropriate message

^COLLAPSE ^

## Task #2 - Points: 1

**Text: Authorization Screenshots**

ℹ Details:

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

#1) Screenshot of the function that checks for a specific role

```
}
// UCID: vs53, Date: Jul 10th 2024
You, yesterday • Uncommitted changes
function has_role($role)
{
    if (is_logged_in() && isset($_SESSION["user"]["roles"])) {
        foreach ($_SESSION["user"]["roles"] as $r) {
            if ($r["name"] === $role) {
                return true;
            }
        }
    }
    return false;
}
```

**Caption (required)** ✓

*Describe/highlight what's being shown*

Screenshot above showing the function that checks for a specific role with my UCID and Date

**Explanation (required)** ✓

*Explain in concise steps how this logically works*

PREVIEW RESPONSE

The has_role function checks if the logged-in user has a specific role. It first calls the is_logged_in function to see if the user is logged in. If the user is logged in and the roles are set in the session, it loops through the roles. For each role, it checks if the role name matches the one provided as the argument. If a match is found, it returns true. If no match is found after checking all roles, it returns false.

#2) Screenshot of the role check function being used. Also, caption what pages it's used on

```
    // UCID: vs53, Date: Jul 10th 2024

if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: " . get_url("home.php")));
}
```

```
_html > Project > admin > create_role.php > ...

...
<?php
//note we need to go up 1 more directory
require(__DIR__ . "/../../../partials/nav.php");


    // UCID: vs53, Date: Jul 10th 2024

if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: " . get_url("home.php")));
}
```

```
_html > Project > admin > assign_roles.php > ...

You, yesterday | 1 author (You)
<?php
//note we need to go up 1 more directory
require(__DIR__ . "/../../../partials/nav.php");


    // UCID: vs53, Date: Jul 10th 2024
    You, yesterday • Uncommitted changes
if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: $BASE_PATH" . "/home.php"));
}
```

**Caption (required)** ✓

*Describe/highlight what's being shown*

Showing 3 screenshots where the role check function is being used in assign_roles.php, create_role.php and list_roles.php

**Explanation (required)** ✓

*Explain in concise steps how this logically works*

PREVIEW RESPONSE

The has_role("Admin") function checks if the logged-in user has the "Admin" role by looking through the roles associated with the user in the session data. If the user does not have the "Admin" role, it sets a flash message saying the user does not have permission to view the page. It then redirects the user to the home page and stops further execution with die(). If the user has the "Admin" role, the script continues to execute. This function is used to restrict access to administrative pages like assign_role.php, create_role.php, and list_roles.php, allowing only users with the "Admin" role to manage roles.

#3) Demonstrate the user-friendly message of trying to manually access a role-protected page while being logged out (must show the appropriate message)


**Caption (required)** ✓

*Describe/highlight what's being shown*

2 Screenshots one showing what would happen if you try to acess a page that role-protected and one if your logged out.

COLLAPSE

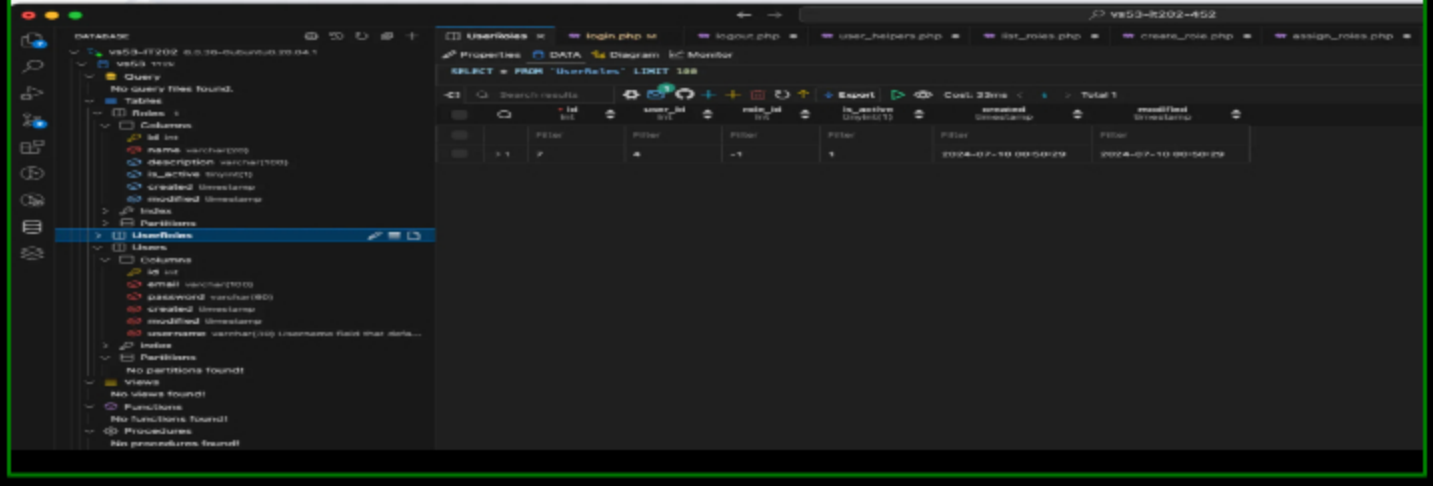
## Task #3 - Points: 1

**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)

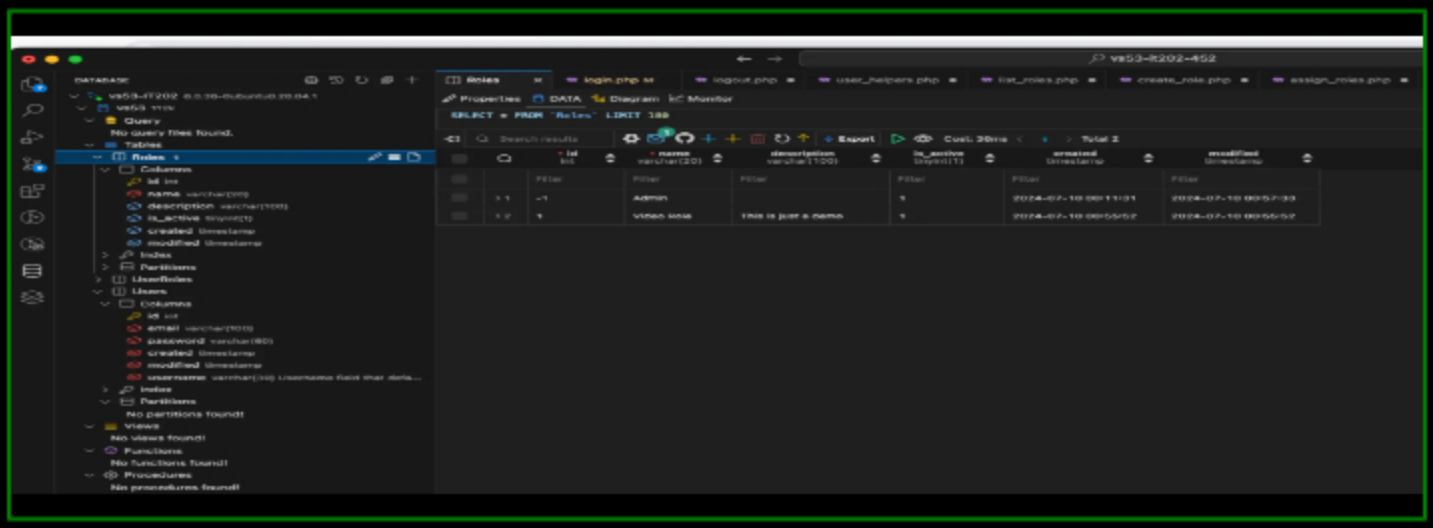


**Caption (required)** ✓

*Describe/highlight what's being shown*

Screenshot of my UserRoles table showing least one valid entry

#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*

Screenshot of my Roles Table with at least one valid entry

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

The purpose of the UserRoles table is to map users to their respective roles within the system. It acts as a bridge between the Users table and the Roles table. Each entry in the UserRoles table associates a user with a specific role by referencing the user_id from the Users table and the role_id from the Roles table. This setup allows for role assignments and management.

#2) Roles

**Explanation (required)** ✓

*How does Roles.is_active differ from UserRoles.is_active?*

PREVIEW RESPONSE

The Roles.is_active field shows whether a specific role is currently active and can be assigned to users. If a role is not active, it cannot be assigned or used in the system. On the other hand, the UserRoles.is_active field shows whether the specific role assigned to a user is active. This means a user can have an inactive role assignment if their UserRoles.is_active is set to false, even if the role itself is active. this allows for temporary deactivation of user roles without removing the role entirely

COLLAPSE

Task #5 - Points: 1

**Text: Include pull request links related to this feature**

Details:
Should end in /pull/#

URL #1
https://github.com/VikShah/vs53-it202-452/pull/18

URL
https://github.com/VikShah/vs53-it202-452/pull/

URL #2
https://github.com/VikShah/vs53-it202-452/pull/19

URL
https://github.com/VikShah/vs53-it202-452/pull/

URL #3
https://github.com/VikShah/vs53-it202-452/pull/22

URL
https://github.com/VikShah/vs53-it202-452/pull/

ADD ANOTHER URL

COLLAPSE

**User Profile** (2 pts.)

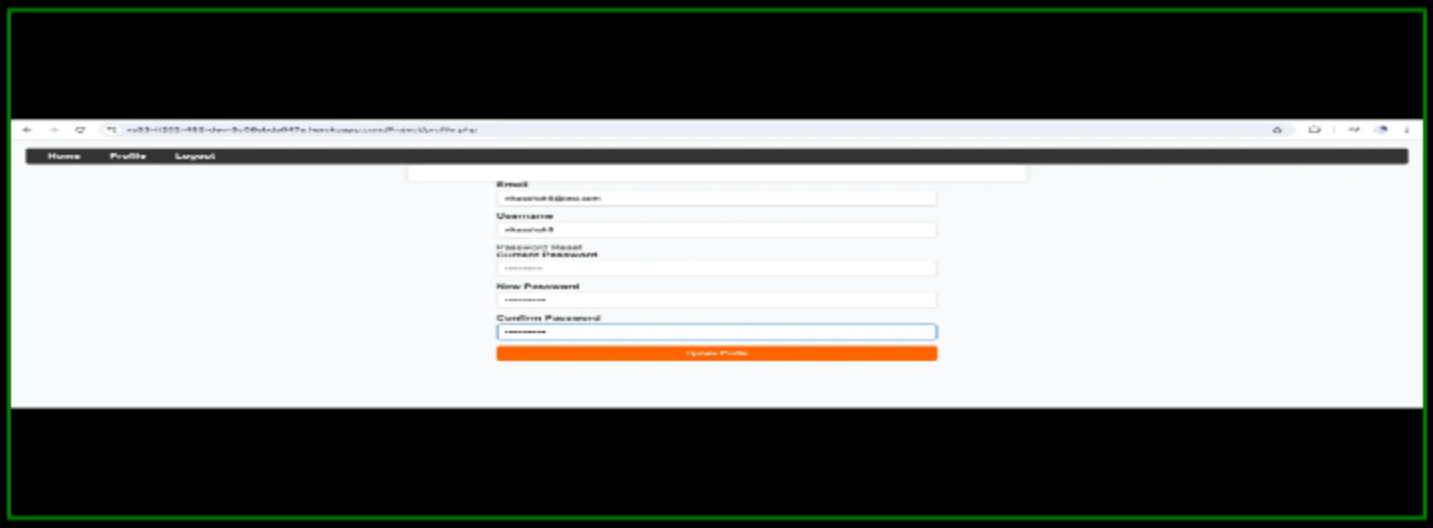
COLLAPSE

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: username, email, current password, new password, confirm password (or similar)




**Caption (required)** ✓
*Describe/highlight what's being shown*
Screenshot above showing profile form with all the fields

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:

When the profile page is visited, it first checks if the user is logged in. If not, the user is redirected to the login page. Then, it retrieves the user's current email and username from the session data. The PHP code queries the database for the latest user information and updates the session data if any changes are found. The email and username fields in the form are populated with this data. When the user submits the form to update their profile, the data is sanitized and validated before saving it to the database. Any changes are reflected back into the session and displayed on the profile page.

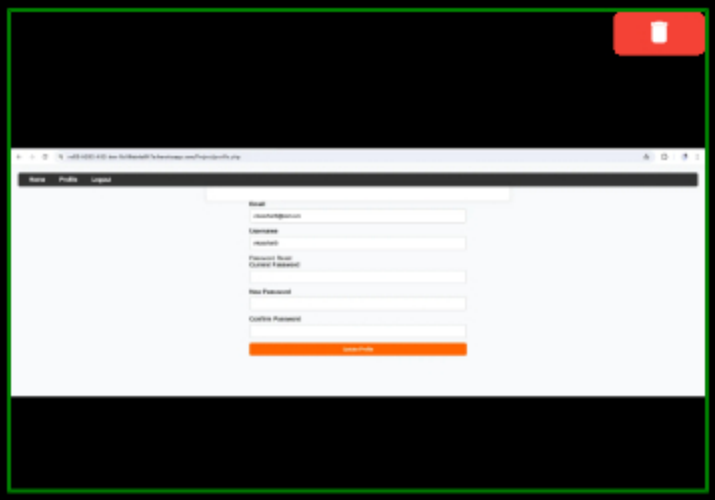
COLLAPSE

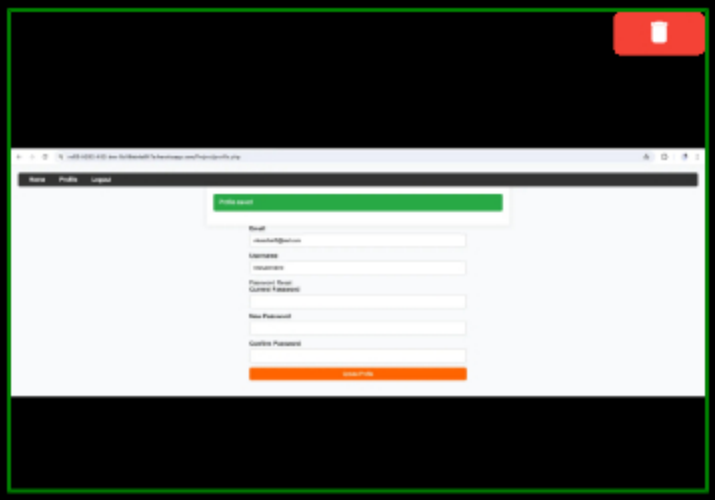
## Task #3 - Points: 1

**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 with before and after of a username change (including success message)



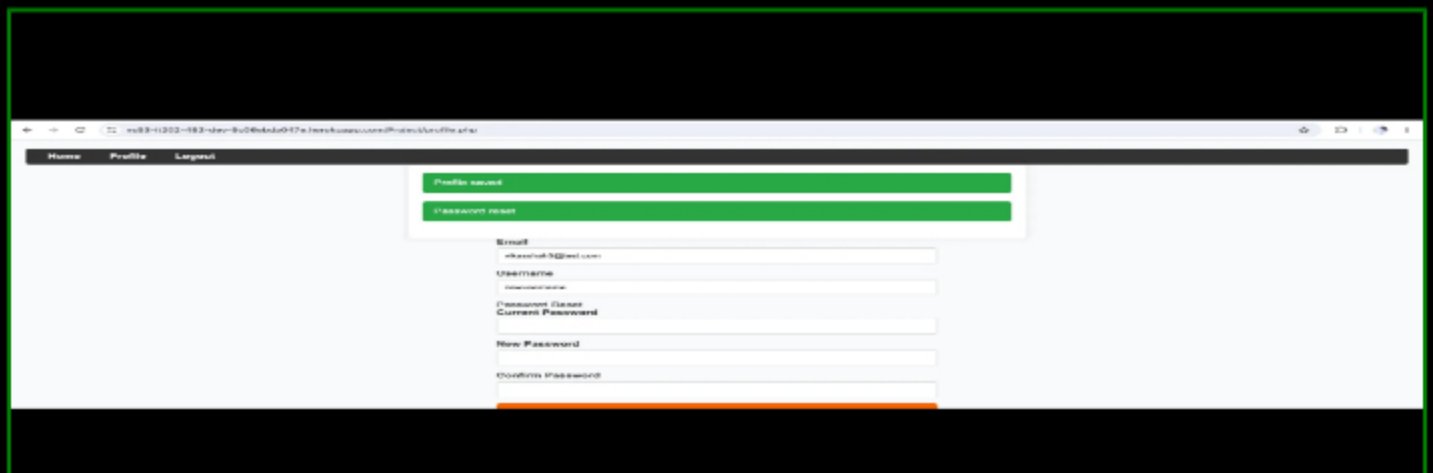


**Caption (required)** ✓

*Describe/highlight what's being shown*

Two screenshots one showing my username before the change and one after the change with the success message

#2) Demonstrate the success message of updating password

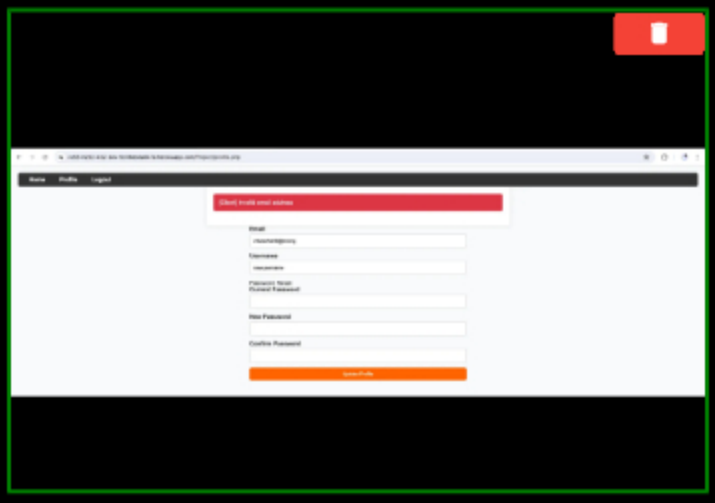


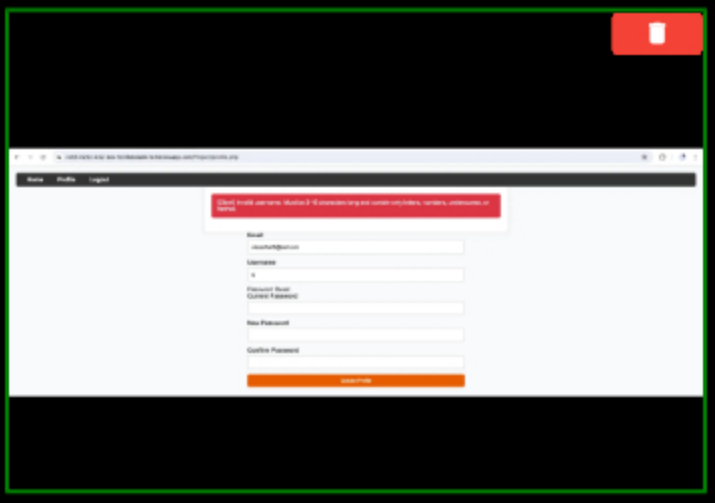
**Caption (required)** ✓

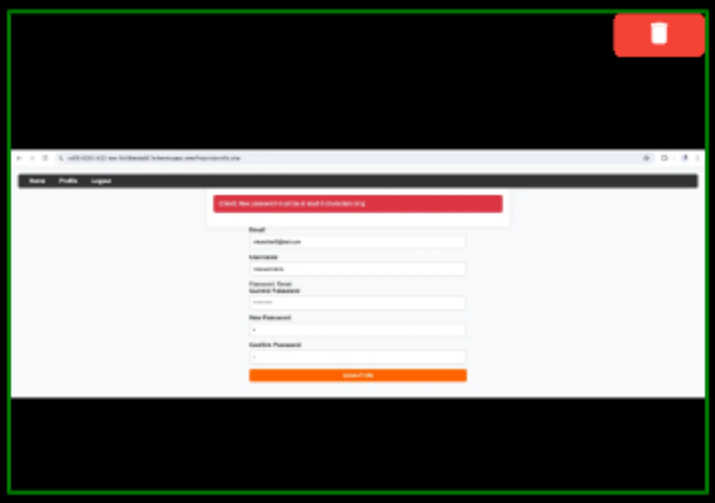
*Describe/highlight what's being shown*

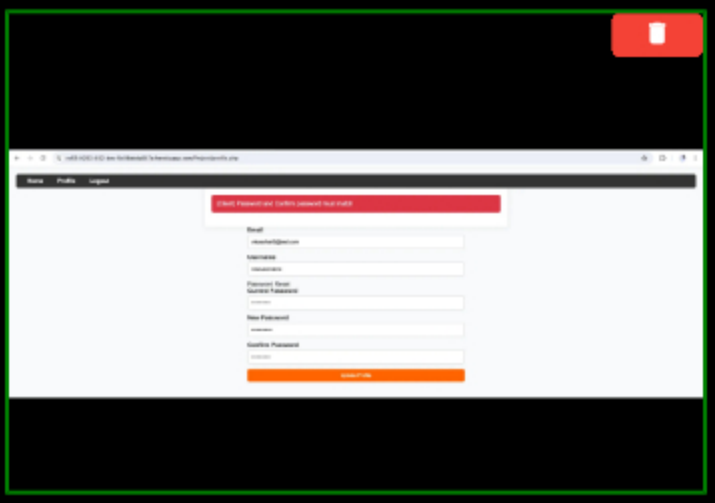
Screenshot showing the success message of updating the password

#3) Demonstrate all JavaScript user-friendly validation messages [can be combined] (email format, username format, password format, new password matching confirm password)









**Caption (required)** ✓

*Describe/highlight what's being shown*

4 screenshots showing all the valdiation messages including email, username, password. and confirm password

#4) Demonstrate PHP user-friendly validation message (desired email is already in use)

**Caption (required)** ✓

*Describe/highlight what's being shown*

Screenshot showing PHP user-friendly validation message if the desired email is already in use

#5) Demonstrate PHP user-friendly validation message (Desired username is already in use)


**Caption (required)** ✓

*Describe/highlight what's being shown*

Screenshot showing the PHP user-friendly validation message if the desired username is already in use

#6) Demonstrate PHP user-friendly validation message (Current password doesn't match what's in the DB)

**Caption (required)** ✓

*Describe/highlight what's being shown*

Screenshot showing the PHP user-friendly validation message if the current password doesn't match

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

**Explanation (required)** ✓

*Explain in concise steps how this logically works*

PREVIEW RESPONSE

When updating the username or email, the form data is first submitted. The server retrieves the email and username from the POST request. The email is sanitized to remove unwanted characters. The server checks if the email is valid and if the username follows the allowed format. If there are no errors, the server prepares an SQL statement to update the user's email and username in the database using their user ID. The database is queried to update these fields. If the update is successful, the new email and username are saved in the session. Then finally, a success message is displayed to the user. If there are errors, appropriate error messages are shown.

### #2) Updating password

**Explanation (required)** ✓

*Explain in concise steps how this logically works*

PREVIEW RESPONSE

When updating the password, the form data is submitted with the current password, new password, and confirm password fields. The server checks if these fields are not empty. It verifies if the new password meets the length requirement. If the new password and confirm password match, the server queries the database to fetch the current password hash for the user. The server then verifies the current password with the stored hash. If the current password is correct, the new password is hashed and an SQL statement is prepared to update the password in the database using the user's ID. If the update is successful, a success message is displayed. If there are errors, appropriate error messages are shown.

COLLAPSE

## Task #5 - Points: 1

**Text: Include pull request links related to this feature**

Details:
Should end in /pull/#

URL #1
https://github.com/VikShah/vs53-it202-452/pull/20

URL
https://github.com/VikShah/vs53-it202-452/pull/

URL #2
https://github.com/VikShah/vs53-it202-452/pull/21

URL
https://github.com/VikShah/vs53-it202-452/pull/

+ ADD ANOTHER URL

Misc (1 pt.)

^COLLAPSE ^

^COLLAPSE ^

Task #1 - Points: 1

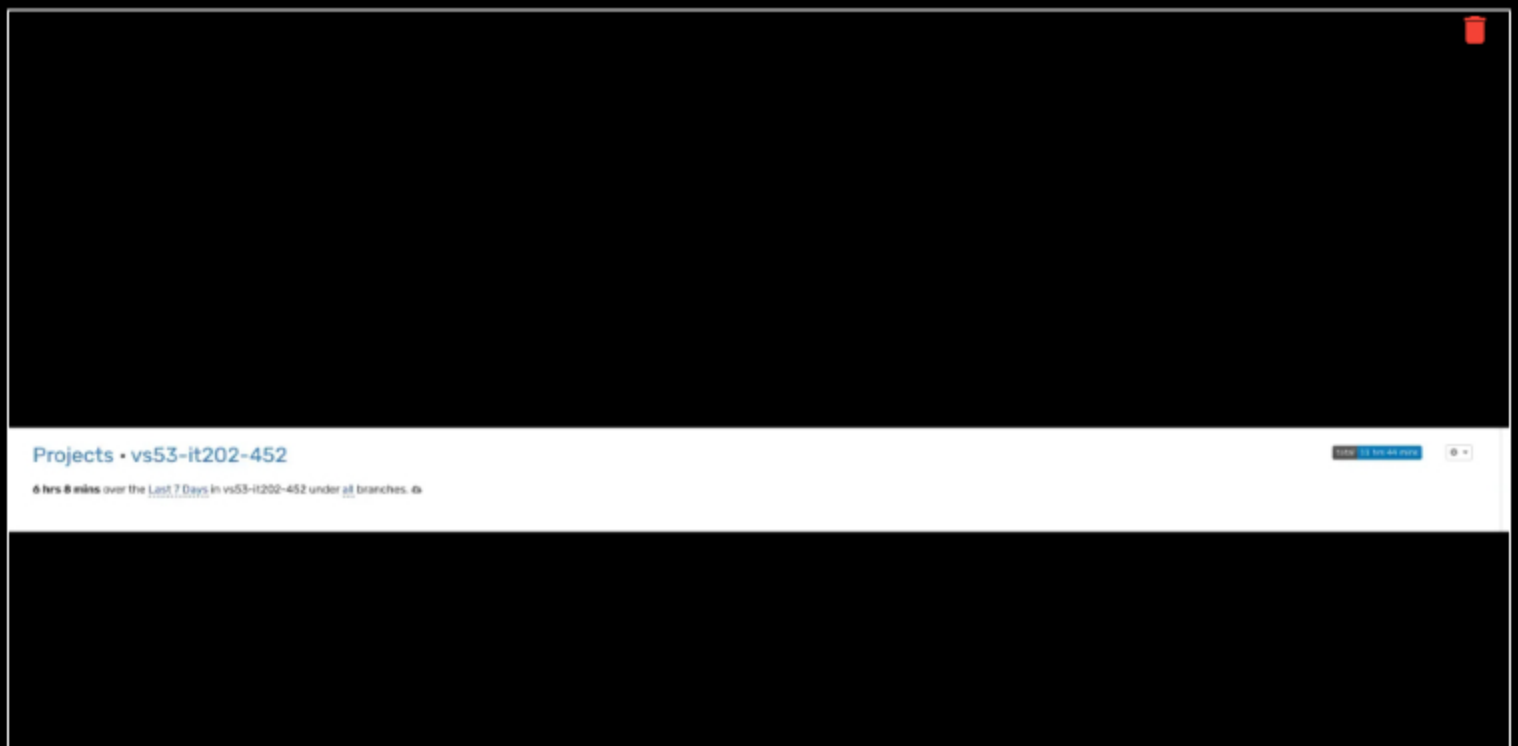
**Text: Screenshot of wakatime**

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

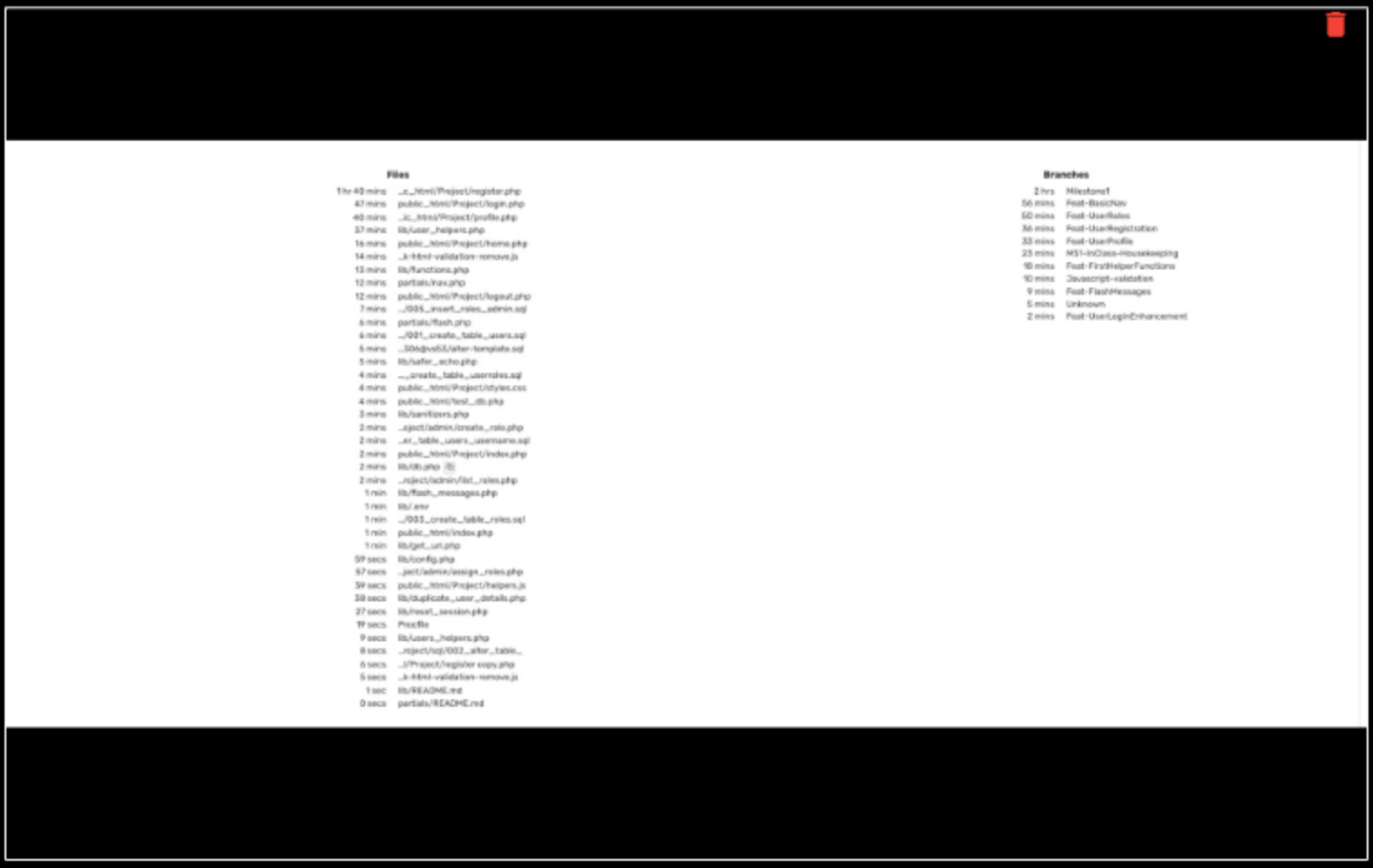
Task Screenshots:

Gallery Style: Large View

Small Medium Large



Wakatime showing total hours spent over the last 7 days in my branch



**Files**

| | |
|---|---|
| 1 hr 40 mins | ...e_/html/Project/register.php |
| 47 mins | public_html/Project/login.php |
| 40 mins | ...ic_html/Project/profile.php |
| 37 mins | lib/user_helpers.php |
| 16 mins | public_html/Project/home.php |
| 14 mins | ...k-html-validation-remove.js |
| 13 mins | lib/functions.php |
| 12 mins | partials/nav.php |
| 12 mins | public_html/Project/logout.php |
| 7 mins | .../005_insert_roles_admin.sql |
| 6 mins | partials/flash.php |
| 6 mins | .../001_create_table_users.sql |
| 5 mins | ...50b@vs53/alter-template.sql |
| 5 mins | lib/safer_echo.php |
| 4 mins | ..._create_table_userroles.sql |
| 4 mins | public_html/Project/styles.css |
| 4 mins | public_html/test_db.php |
| 3 mins | lib/sanitizers.php |
| 2 mins | ...oject/admin/create_role.php |
| 2 mins | ...er_table_users_username.sql |
| 2 mins | public_html/Project/index.php |
| 2 mins | lib/db.php |
| 2 mins | ...roject/admin/list_roles.php |
| 1 min | lib/flash_messages.php |
| 1 min | lib/.env |
| 1 min | .../003_create_table_roles.sql |
| 1 min | public_html/index.php |
| 1 min | lib/get_url.php |
| 59 secs | lib/config.php |
| 57 secs | ...ject/admin/assign_roles.php |
| 39 secs | public_html/Project/helpers.js |
| 38 secs | lib/duplicate_user_details.php |
| 27 secs | lib/reset_session.php |
| 19 secs | Procfile |
| 9 secs | lib/users_helpers.php |
| 8 secs | ...roject/sql/002_alter_table_ |
| 6 secs | .i/Project/register copy.php |
| 5 secs | ...k-html-validation-remove.js |
| 1 sec | lib/README.md |
| 0 secs | partials/README.md |

**Branches**

| | |
|---|---|
| 2 hrs | Milestone1 |
| 56 mins | Feat-BasicNav |
| 50 mins | Feat-UserRoles |
| 36 mins | Feat-UserRegistration |
| 33 mins | Feat-UserProfile |
| 23 mins | MS1-InClass-Housekeeping |
| 18 mins | Feat-FirstHelperFunctions |
| 10 mins | Javascript-validation |
| 9 mins | Feat-FlashMessages |
| 5 mins | Unknown |
| 2 mins | Feat-UserLoginEnhancement |

Showing all the branches I created and how long I spent on them each

COLLAPSE

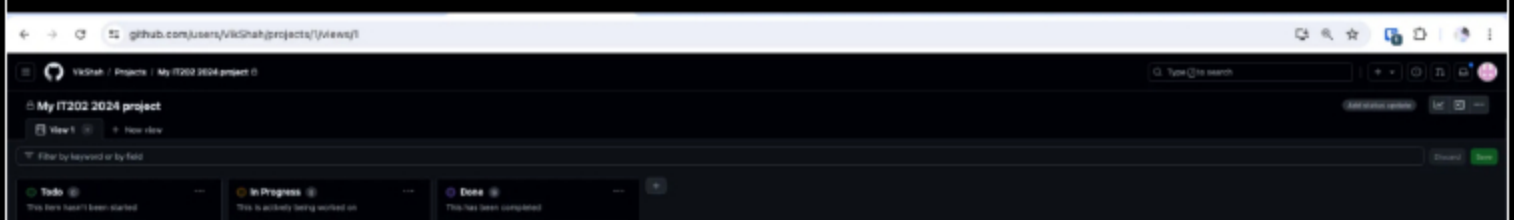
## Task #2 - Points: 1

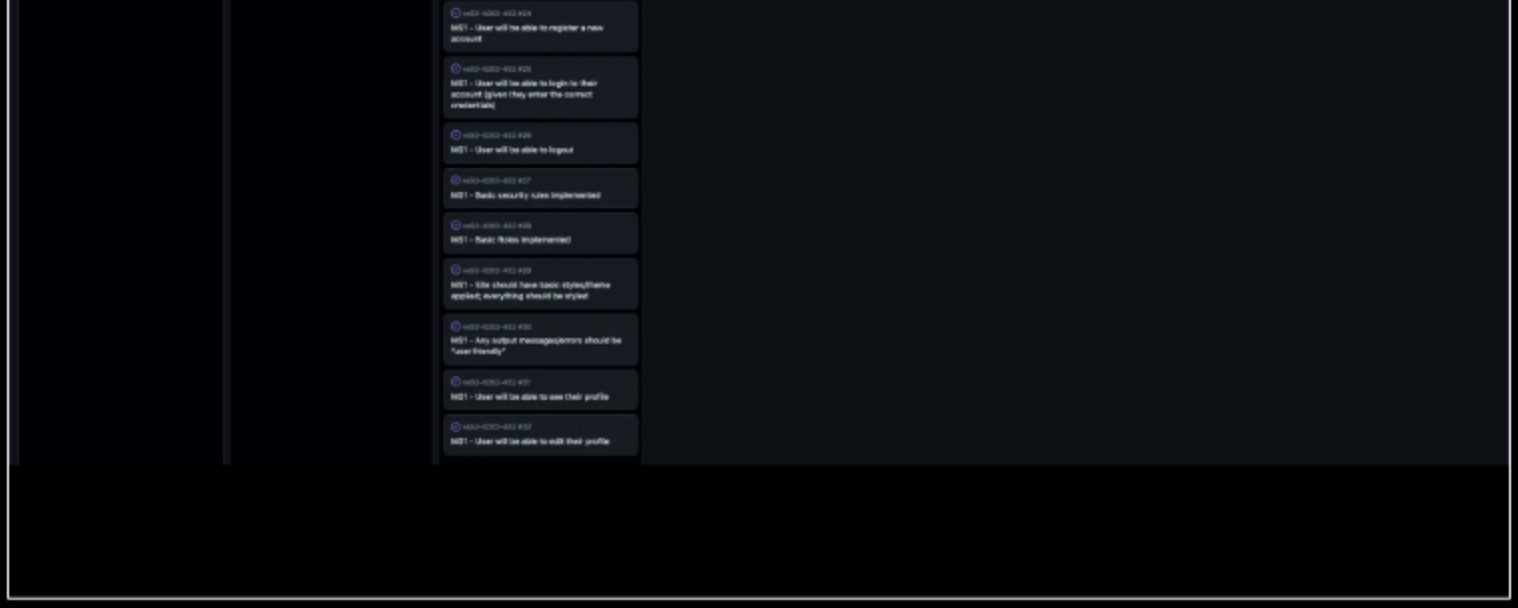
**Text: Screenshot of your project board from GitHub (tasks should be in the proper column)**

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Showing all the issues that are closed, 9 of them to be precise

^COLLAPSE ^

Task #3 - Points: 1

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

URL #1

https://github.com/users/VikShah/projects/1

URL

https://github.com/users/VikShah/projects/1

+ ADD ANOTHER URL

End of Assignment