Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-008-S2024/it202-milestone-1-2024/grade/ak2774

IT202-008-S2024 - [IT202] Milestone 1 2024

Submissions:

Submission Selection

1 Submission [active] 4/1/2024 5:44:11 PM

Instructions

COLLAPSE

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
  - 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, Logout, 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: 26 Points: 10.00

User Registration (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of form on website page

Checklist

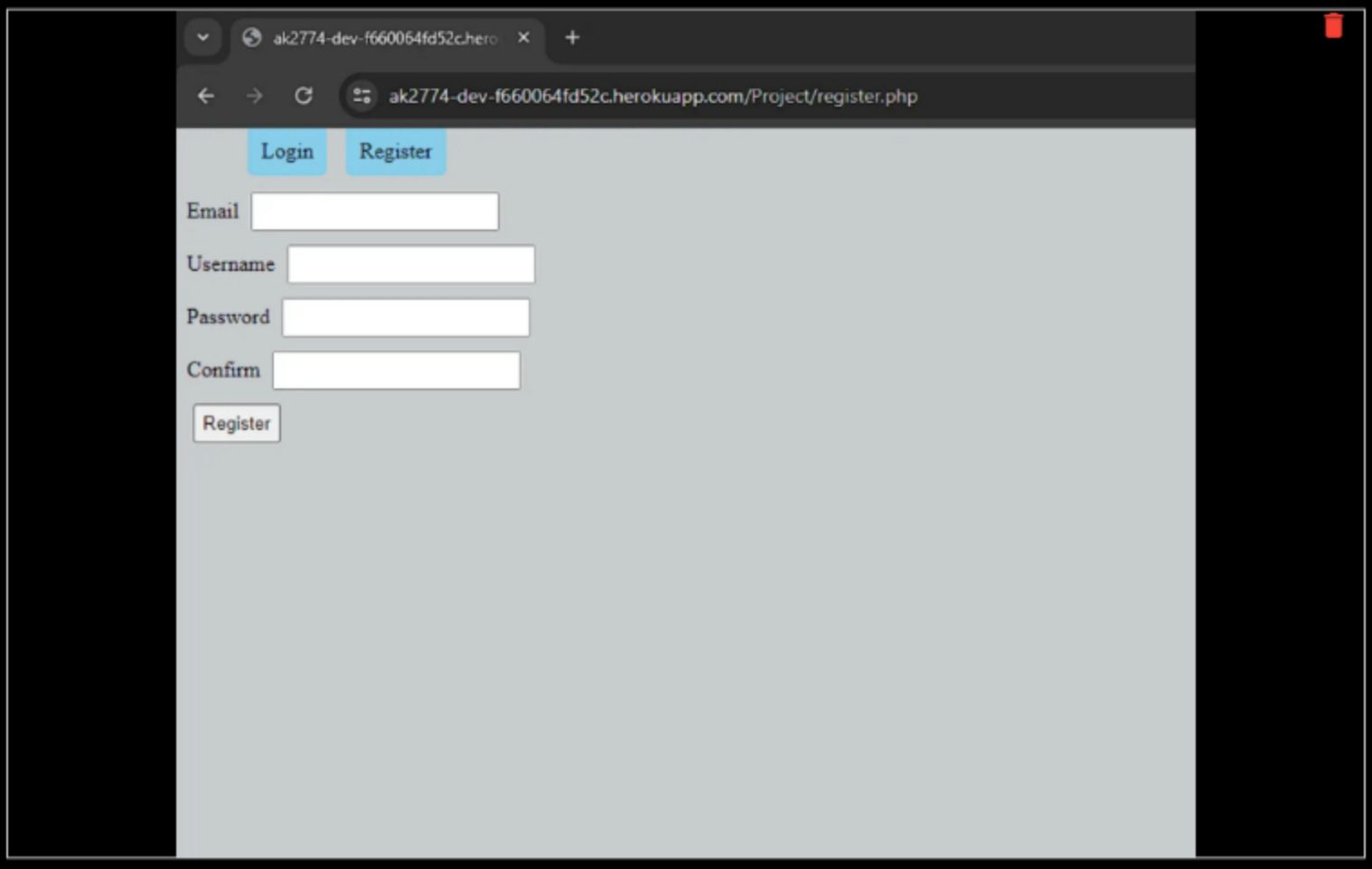
*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Heroku dev url should be present in the address bar
☐ #2	1	Should have thoughtful CSS applied
☐ #3	1	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
☐ #4	1	Demonstrate email already in use message (message text doesn't need to be exact, but should be clear)
☐ #5	1	Demonstrate username already in use message (message text doesn't need to be exact, but should be clear)
☐ #6	1	Demonstrate user-friendly message of new account being created

Task Screenshots:

Gallery Style: Large View

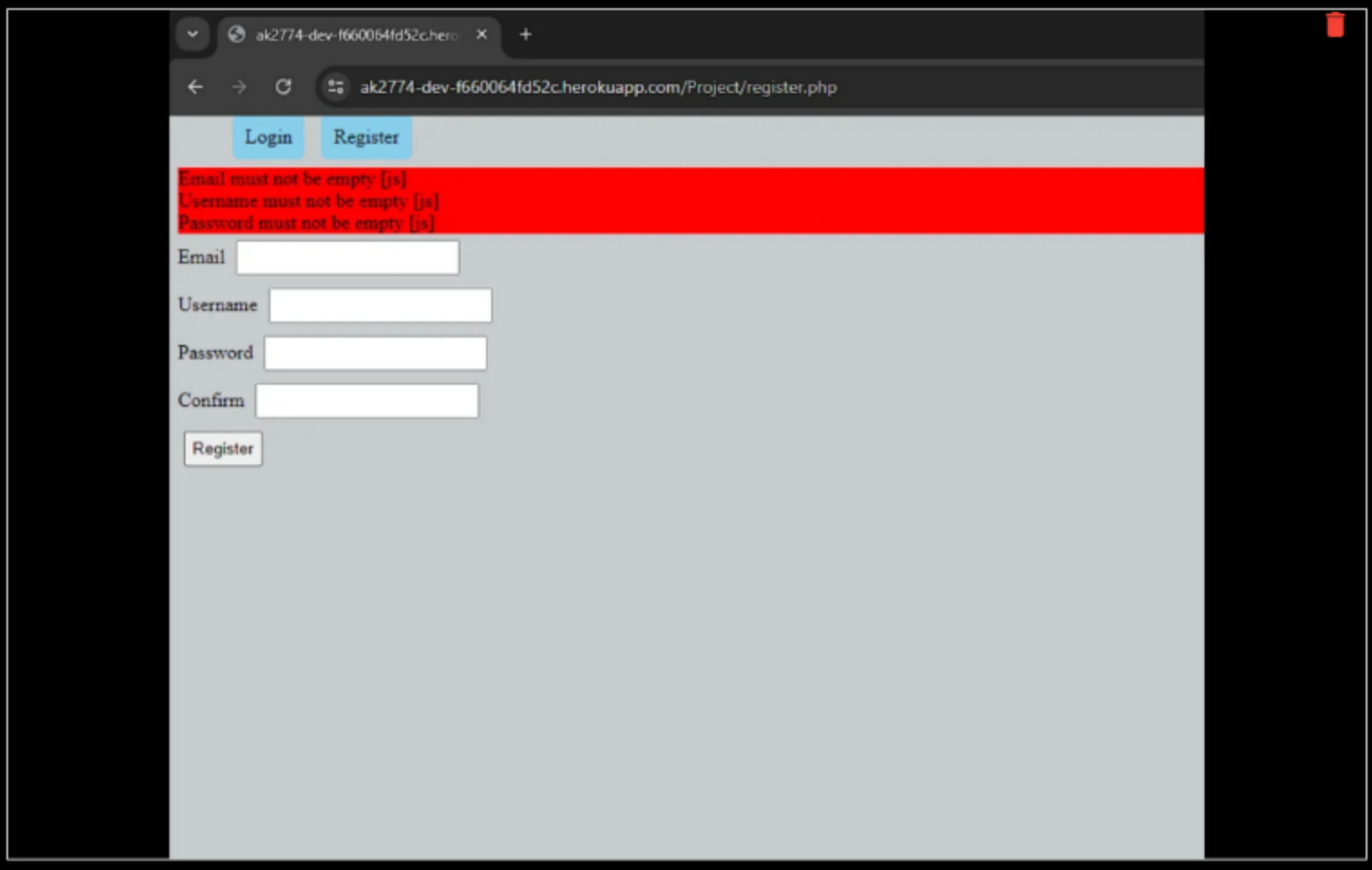
Small Medium Large



Main page when clicked on register

Checklist Items (1)

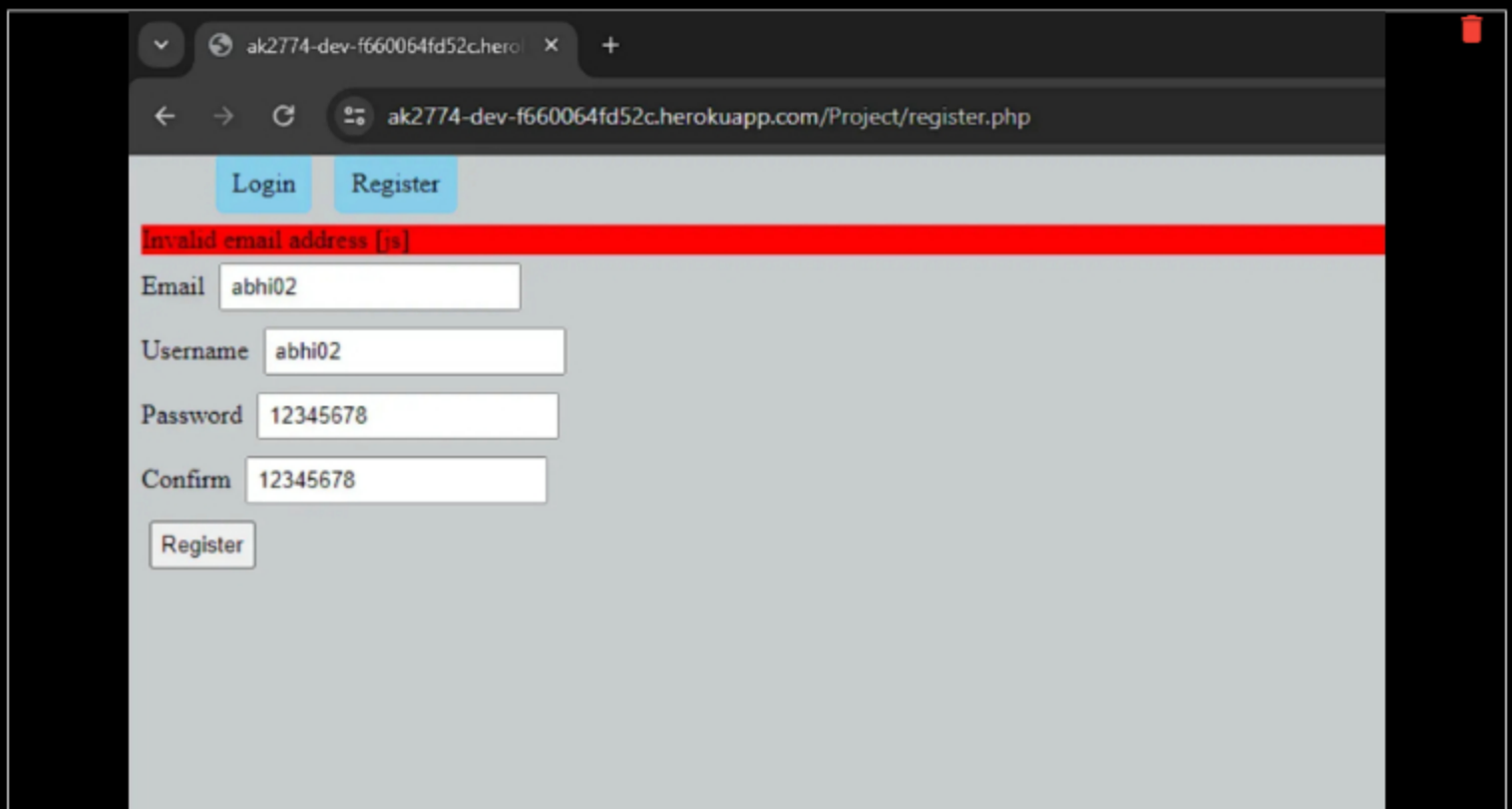
#1 Heroku dev url should be present in the address bar



JS validation showing each field is required

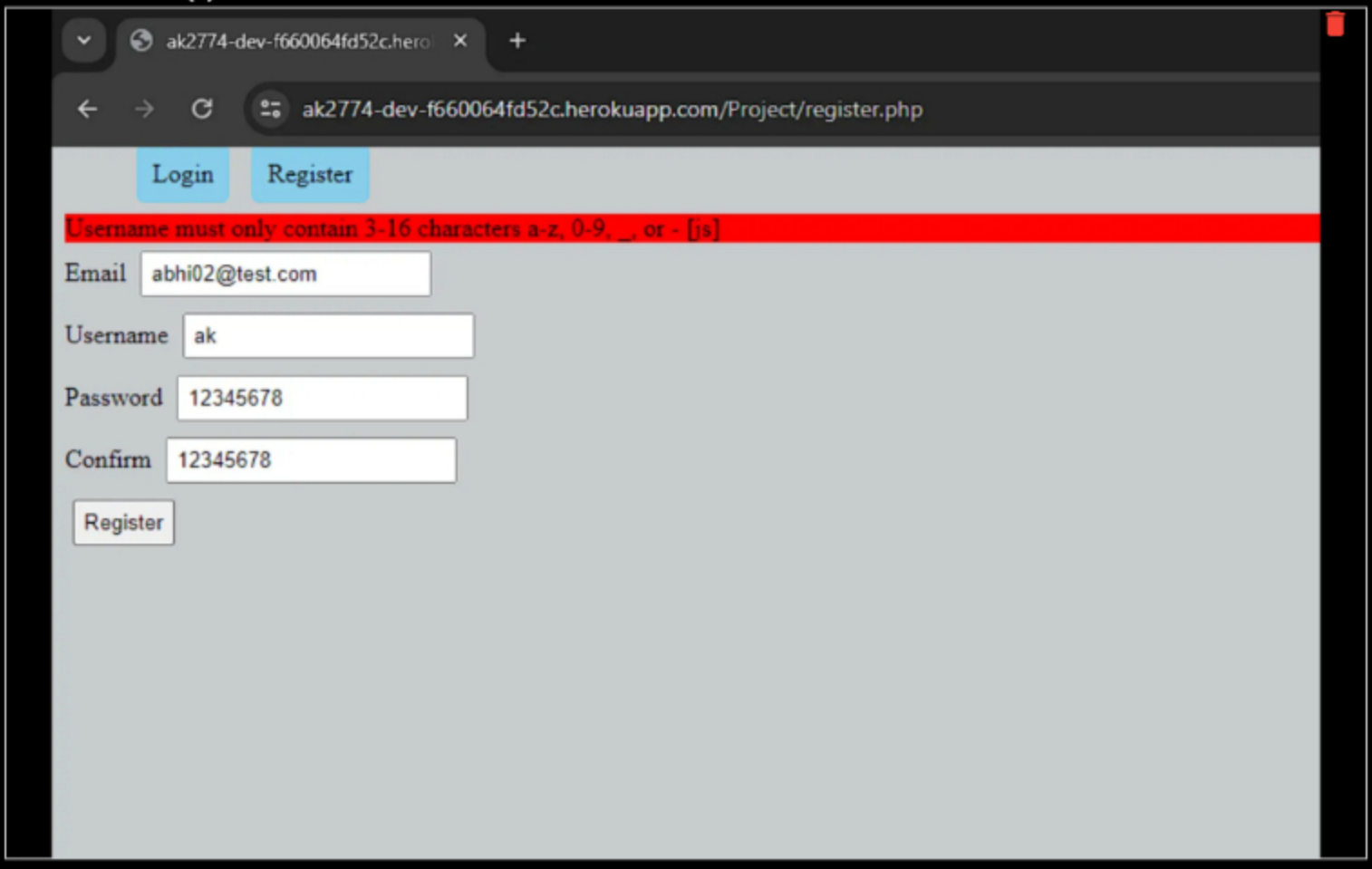
Checklist Items (1)

#3 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



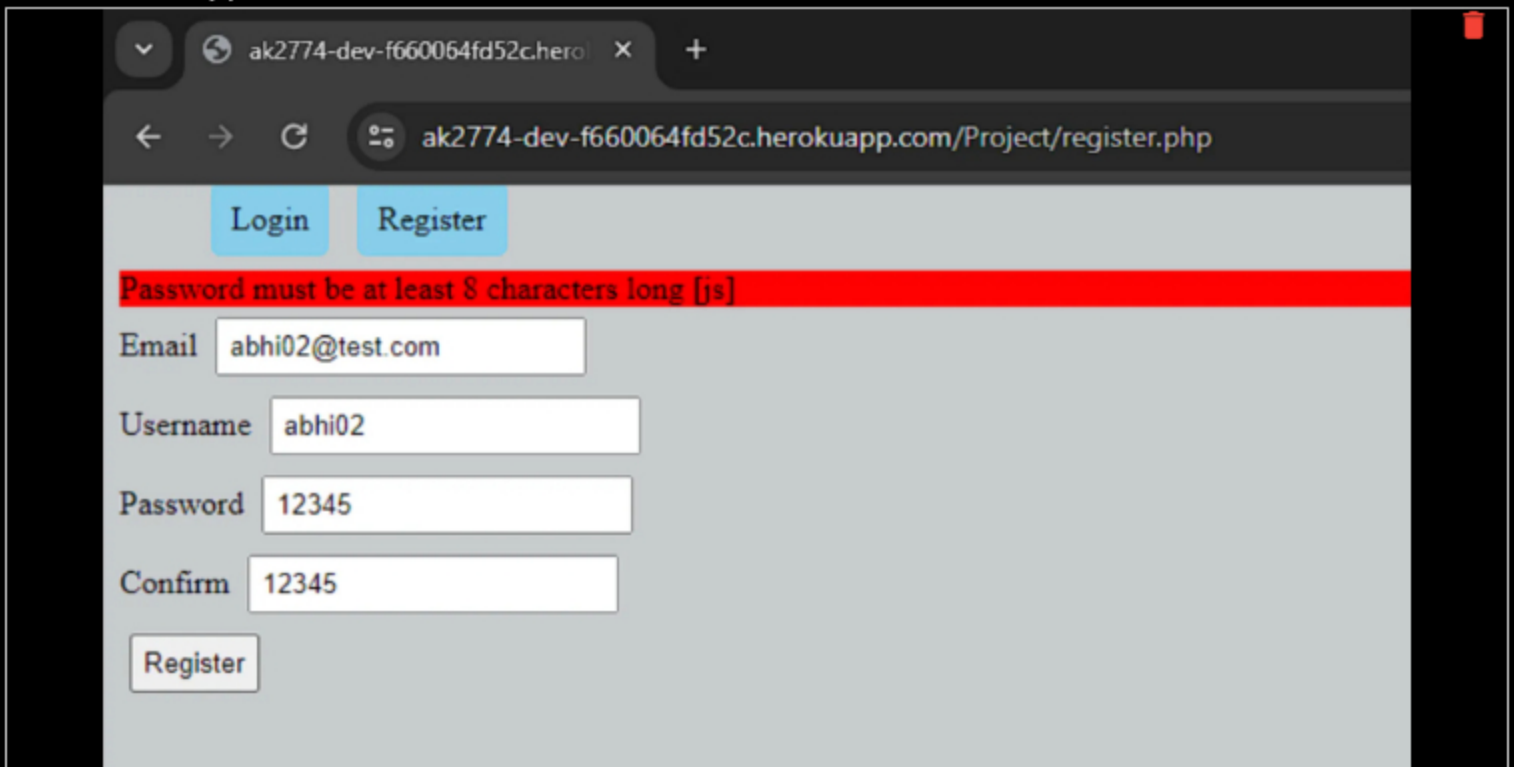
JS validation showing email format

Checklist Items (0)



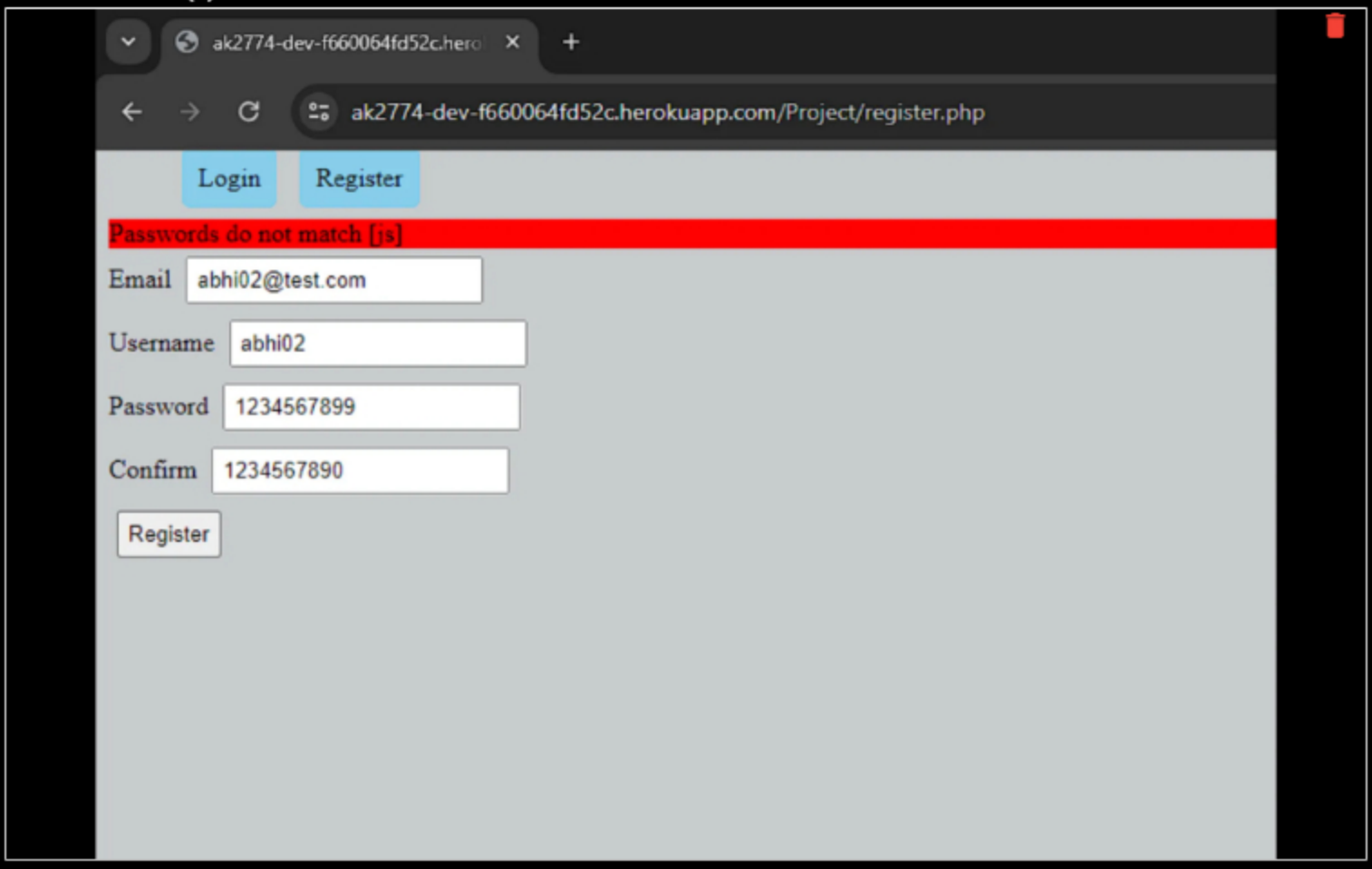
JS validation showing username format

Checklist Items (0)



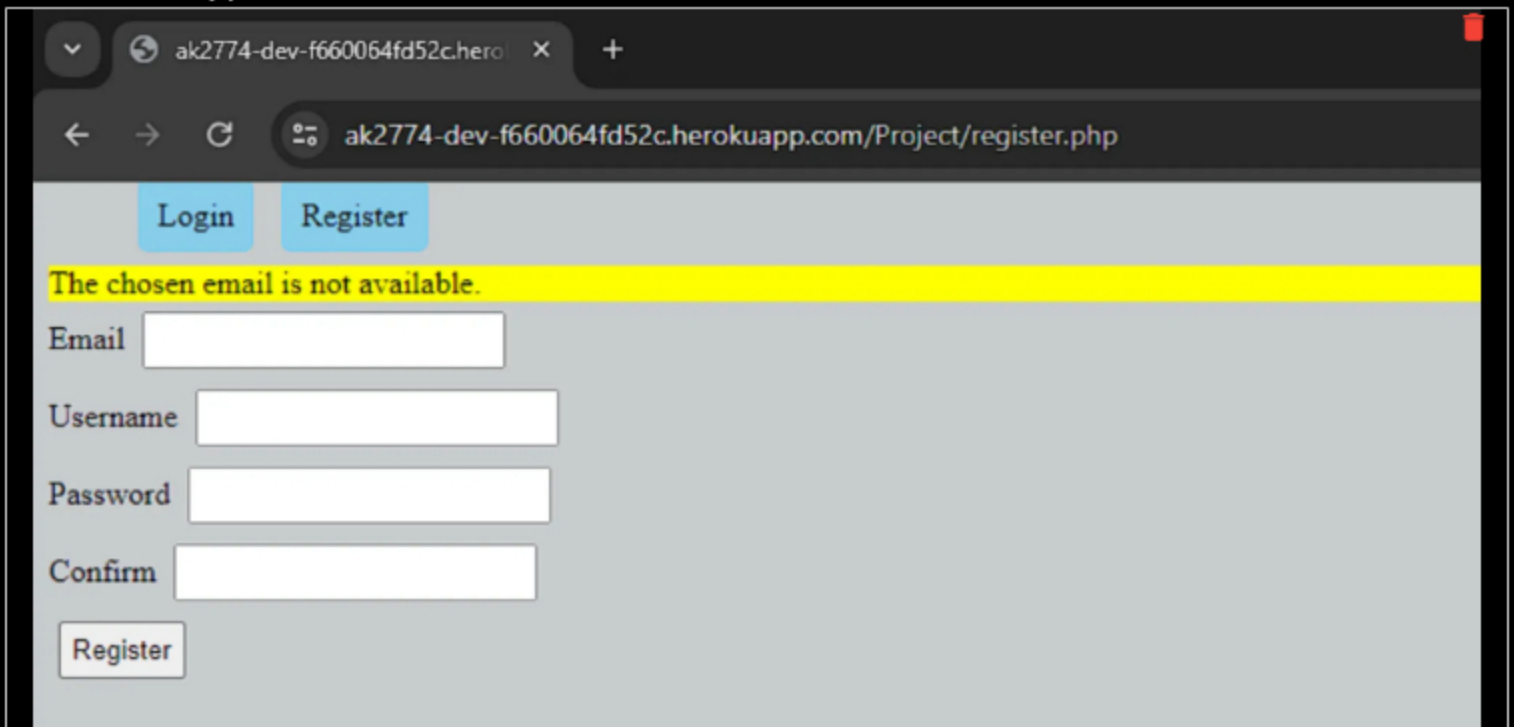
JS validation showing password format

Checklist Items (0)



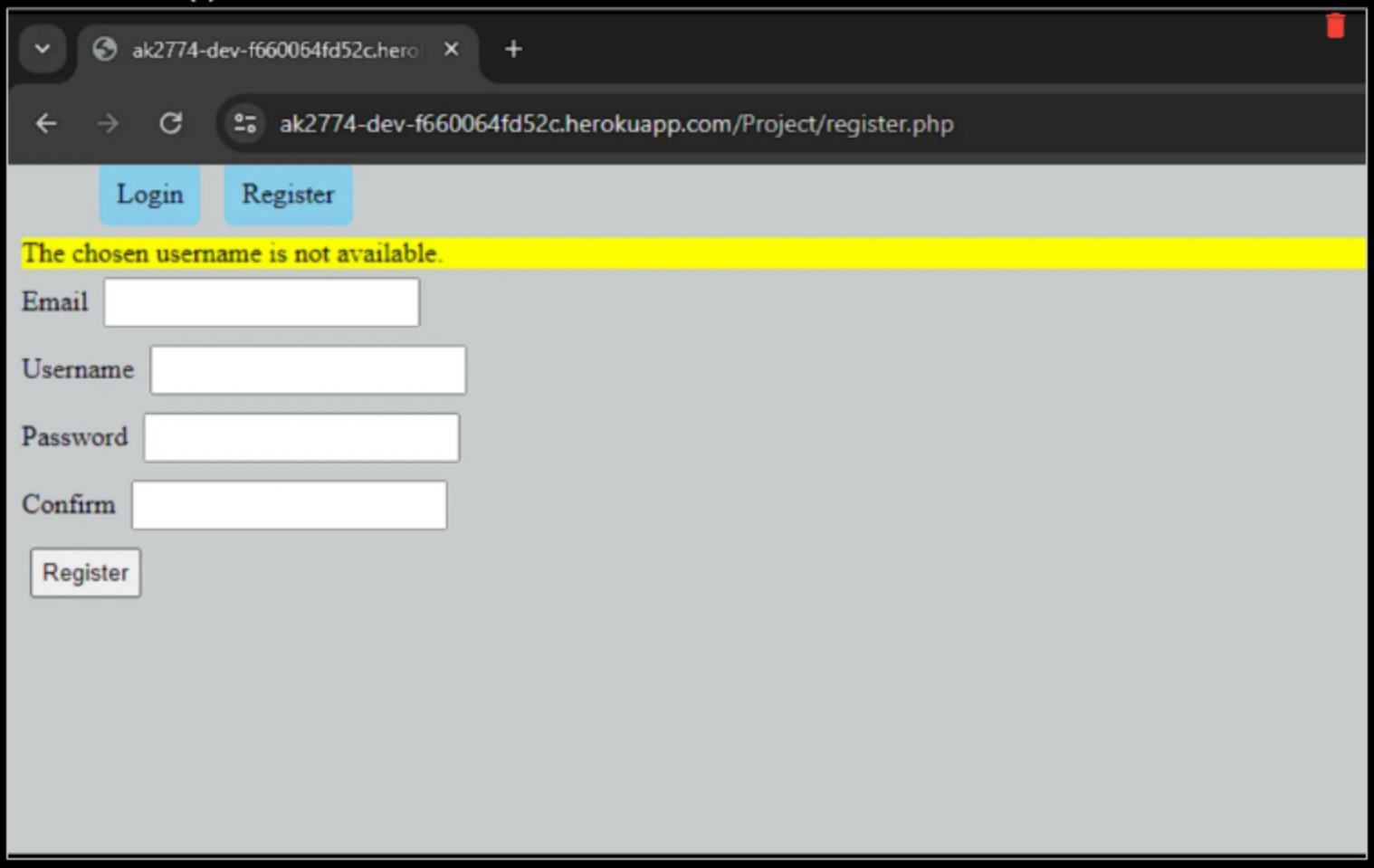
JS validation showing passwords don't match

Checklist Items (0)



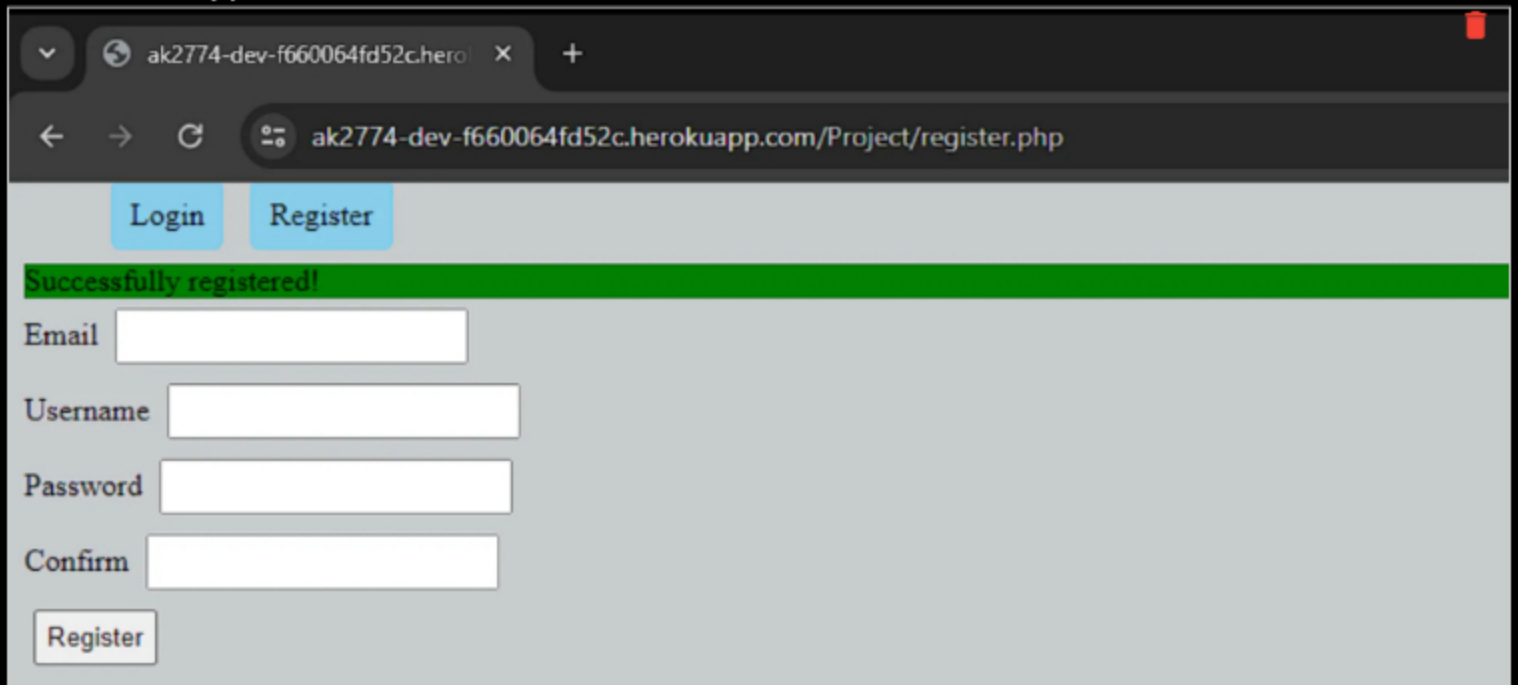
Email already in use message

Checklist Items (0)



Username already in use message

Checklist Items (0)



New account being created message

Checklist Items (0)

^COLLAPSE^

Task #2 - Points: 1

Text: Screenshot of the form code

Details:

Should have appropriate input types for the fiel

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
<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>
```

Screenshot of form code

^COLLAPSE^

Task #3 - Points: 1

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

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Show the JavaScript validations (include any extra files related)
☐ #2	1	Show the PHP validations (include any lib content)
☐ #3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```javascript
function validate(form) {
    //TODO 1: implement JavaScript validation
    //ensure it returns false for an error and true for success
    var email = form.email.value;
    var username = form.username.value;
    var password = form.password.value;
    var confirm = form.confirm.value;
    let isValid=true;
    var emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    var usernamePattern = /^[a-zA-Z0-9_-]{3,16}$/;

    if (email === "") {
        flash("Email must not be empty [js]", "danger");
        isValid=false;
    }
    else if (!emailPattern.test(email)) {
        flash("Invalid email address [js]", "danger");                    //ak2774
        isValid=false;                                                    //4/1/2024
    }
    if (username === "") {
        flash("Username must not be empty [js]", "danger");
        isValid=false;
    }
    else if (!usernamePattern.test(username)) {
        flash("Username must only contain 3-16 characters a-z, 0-9, _, or - [js]", "danger");
        isValid=false;
    }
    if (password === "") {
        flash("Password must not be empty [js]", "danger");
        isValid=false;
    }
    else if (password.length < 8 || confirm.length < 8) {
        flash("Password must be at least 8 characters long [js]", "danger");
        isValid=false;
    }
    if (password != confirm) {
        flash("Passwords do not match [js]", "danger");
        isValid=false;
    }

    return isValid;
} <- #25-82 function validate(form)
```

Client-side validation

Checklist Items (0)

```php
if (isset($_POST["email"]) && isset($_POST["password"]) && isset($_POST["confirm"]) && isset($_POST["username"])) {
    $email = se($_POST, "email", "", false);
    $password = se($_POST, "password", "", false);
    $confirm = se($_POST, "confirm", "", false);
    $username = se($_POST, "username", "", false);
    //TODO 3
    $hasError = false;
    if (empty($email)) {
        flash("Email must not be empty", "danger");
        $hasError = true;
    }
    //sanitize
    $email = sanitize_email($email);
    //validate
    if (!is_valid_email($email)) {
        flash("Invalid email address", "danger");
        $hasError = true;
```

```php
    }
    if (!is_valid_username($username)) {
        flash("Username must only contain 3-16 characters a-z, 0-9, _, or -", "danger");
        $hasError = true;
    }
    if (empty($password) || empty($confirm)) {
        flash("password must not be empty", "danger");            //ak2774
        $hasError = true;                                         //4/1/2024
    }

    if (!is_valid_password($password)) {
        flash("Password too short", "danger");
        $hasError = true;
    }
    if (strlen($password) > 0 && $password !== $confirm) {
        flash("Passwords must match", "danger");
        $hasError = true;
    }
    if (!$hasError) {
        // TODO 4
        $hash = password_hash($password, PASSWORD_BCRYPT);
        $db = getDB();
        $stmt = $db->prepare("INSERT INTO Users (email, password, username) VALUES(:email, :password, :username)");
        try {
            $stmt->execute([":email" => $email, ":password" => $hash, ":username" => $username]);
            flash("Successfully registered!", "success");
        } catch (Exception $e) {
            users_check_duplicate($e->errorInfo);
        }
    } <- #123-134 if (!$hasError)
} <- #86-135 if (isset($_POST["email"]) && isset($_POST["password"]) && is...
```

Server-side validation

Checklist Items (0)

COLLAPSE

Task #4 - Points: 1

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

Checklist

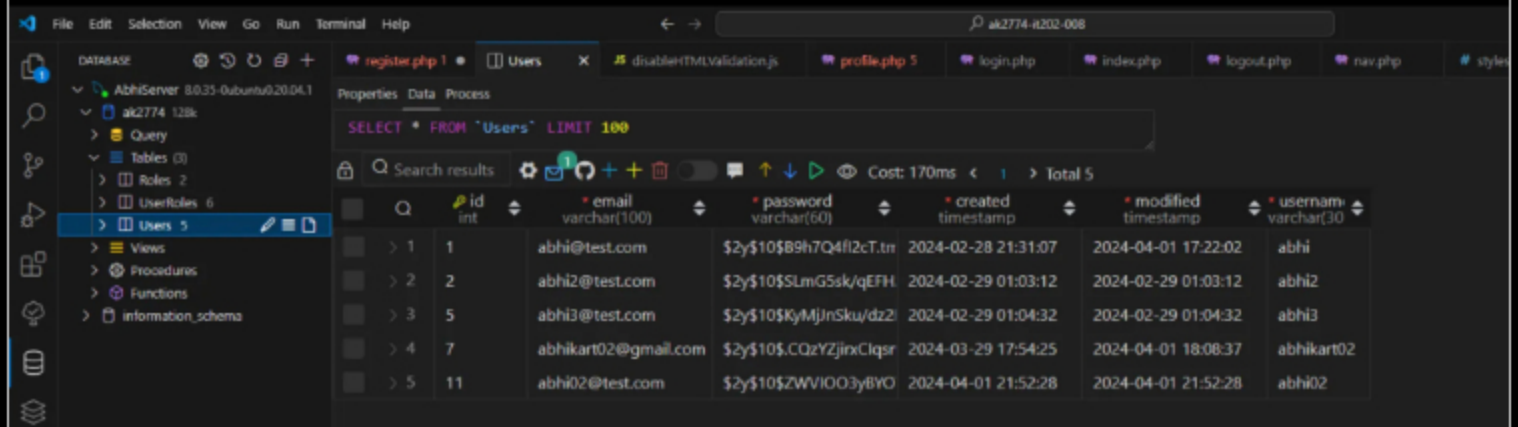
*The checkboxes are for your own tracking

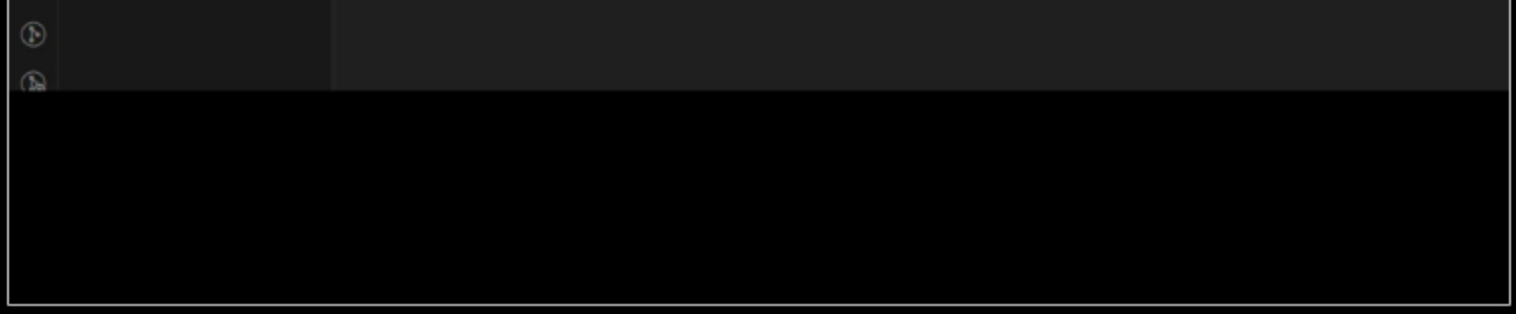
#	Points	Details
☐ #1	1	Password should be hashed
☐ #2	1	Should have email, password, username (unique), created, modified, and id fields
☐ #3	1	Ensure left panel or database name is present (should contain your ucid)

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Screenshot of Users table

Checklist Items (0)

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

Response:

Page Load:

The PHP code at the beginning includes necessary files and resets the session. The HTML form for user registration is displayed, including fields for email, username, password, and password confirmation. JavaScript code for client-side validation is included in the page. This code defines a validate() function that checks if the input values are valid before submitting the form.

Client-Side Validation (JavaScript):

When the form is submitted, the validate() function is called. This function retrieves input values for email, username, password, and confirmation. It validates each field according to defined criteria, such as non-empty, valid email format, valid username format, minimum password length, and password confirmation. If any validation fails, an error message is displayed using the flash() function, and isValid is set to false. If all validations pass, isValid remains true.

Server-Side Validation (PHP):

After the client-side validation, if the form is submitted, the PHP code checks if the required POST variables are set. It retrieves the submitted values for email, username, password, and confirmation. Server-side validation checks are performed to ensure that the submitted data is valid: Email must not be empty, and it must be a valid email format. Username must not be empty, and it must follow the specified pattern. Password and confirmation must not be empty, and password must be at least 8 characters long. Password and confirmation must match. If any validation fails, error messages are flashed using the flash() function, and $hasError is set to true.

Database Insertion:

If all validations pass (both client-side and server-side), $hasError remains false. The password is hashed using password_hash() function for security. A database connection is established, and a prepared statement is used to insert the user's data (email, hashed password, username) into the database table Users. If the insertion is successful,

a success message is flashed to the user. If there's an error during the database operation, such as a duplicate entry, it's handled appropriately, and an error message is flashed.

COLLAPSE

Task #6 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/abhinavkarthikn/ak2774-it202-008/pull/29

User Login (2 pts.)

COLLAPSE

EXPAND

Task #1 - Points: 1

Text: Screenshot of form on website page

EXPAND

Task #2 - Points: 1

Text: Screenshot of the form code

EXPAND

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

EXPAND

Task #4 - Points: 1

Text: Include pull request links related to this feature

User Logout (1 pt.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Capture the following screenshots

Checklist

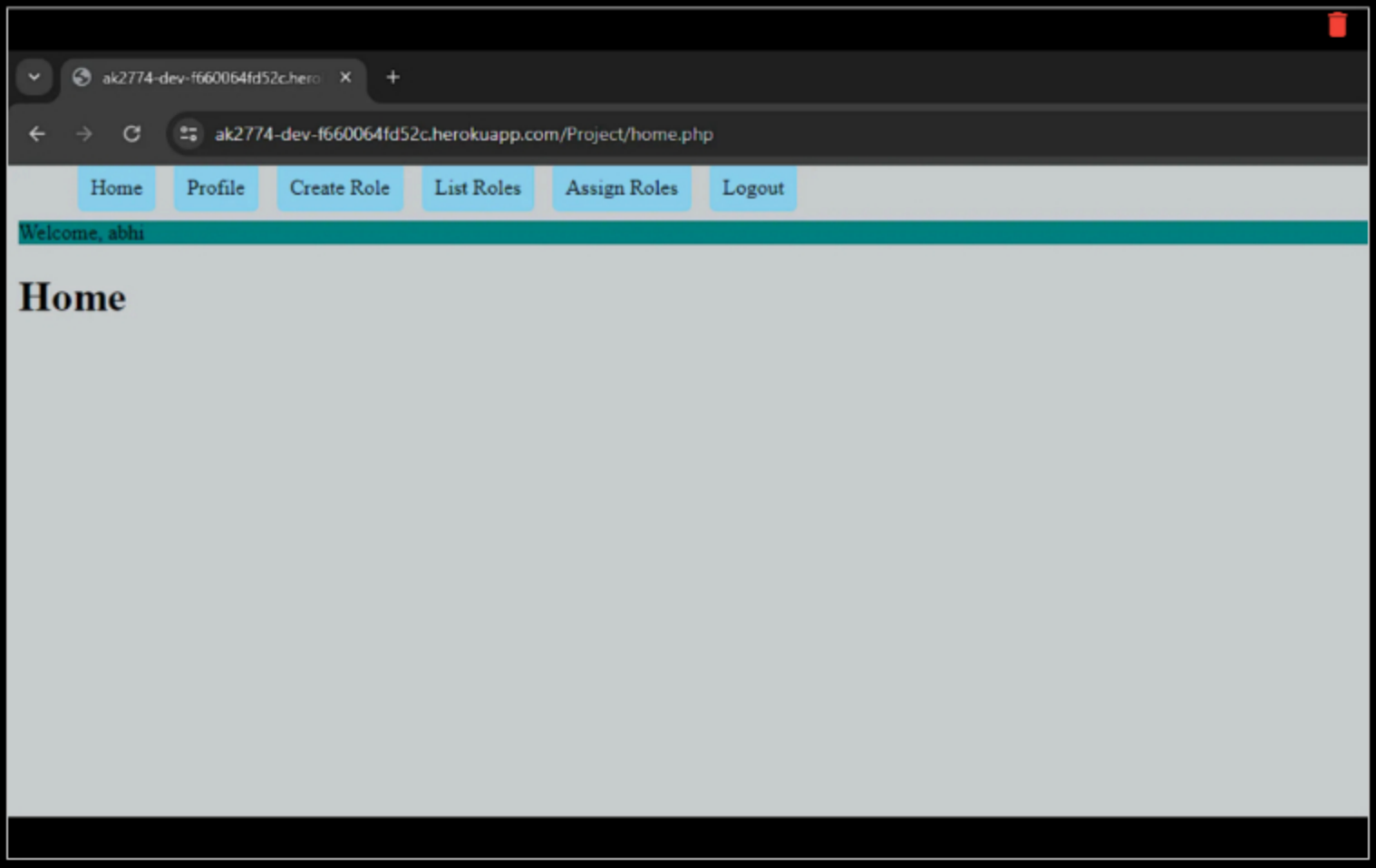
*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Screenshot of the navigation when logged in (site)
☐ #2	1	Screenshot of the redirect to login with the user-friendly logged-out message (site)
☐ #3	1	Screenshot of the logout-related code showing the session is destroyed (code). Ensure ucid/date comment is present.

Task Screenshots:

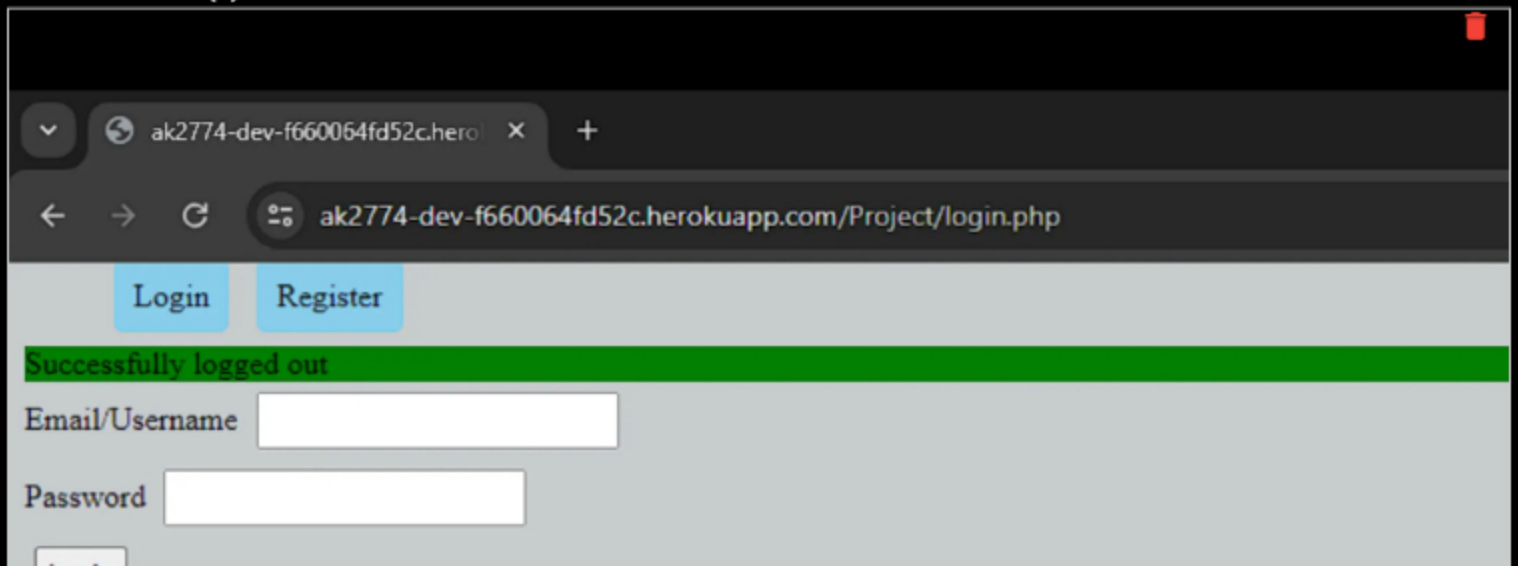
Gallery Style: Large View

Small Medium Large



Screenshot of page when logged in

Checklist Items (0)



Login

Screenshot of login page with log out message

Checklist Items (0)

public_html > Project > logout.php

```php
<?php
session_start();
require(__DIR__ . "/../../lib/functions.php");    //ak2774
reset_session();                                  //4/1/2024

flash("Successfully logged out", "success");
header("Location: login.php");
```

Screenshot of code

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1

https://github.com/abhinavkarthikn/ak2774-it202-008/pull/32

Basic Security Rules and Roles (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Authentication Screenshots

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Screenshot of the function that checks if a user is logged in
☐ #2	1	Screenshot of the login check function being used (i.e., profile likely). Also caption what pages it's used on
☐ #3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)
☐ #4	1	Demonstrate the user-friendly message of trying to manually access a login-protected page while being logged out

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```php
function is_logged_in($redirect = false, $destination = "login.php")
{
    $isLoggedIn = isset($_SESSION["user"]);
    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");     //ak2774
        die(header("Location: $destination"));                           //4/1/2024
    } <- #10-14 if ($redirect && !$isLoggedIn)
    return $isLoggedIn;
} <- #8-16 function is_logged_in($redirect = false, $destination = "logi...
```

Screenshot of is_logged_in function

Checklist Items (0)

```php
<?php
require_once(__DIR__ . "/../../partials/nav.php");  // ak2774, 4/1/2024
is_logged_in(true);
?>
<?php
```

Used on profile.php

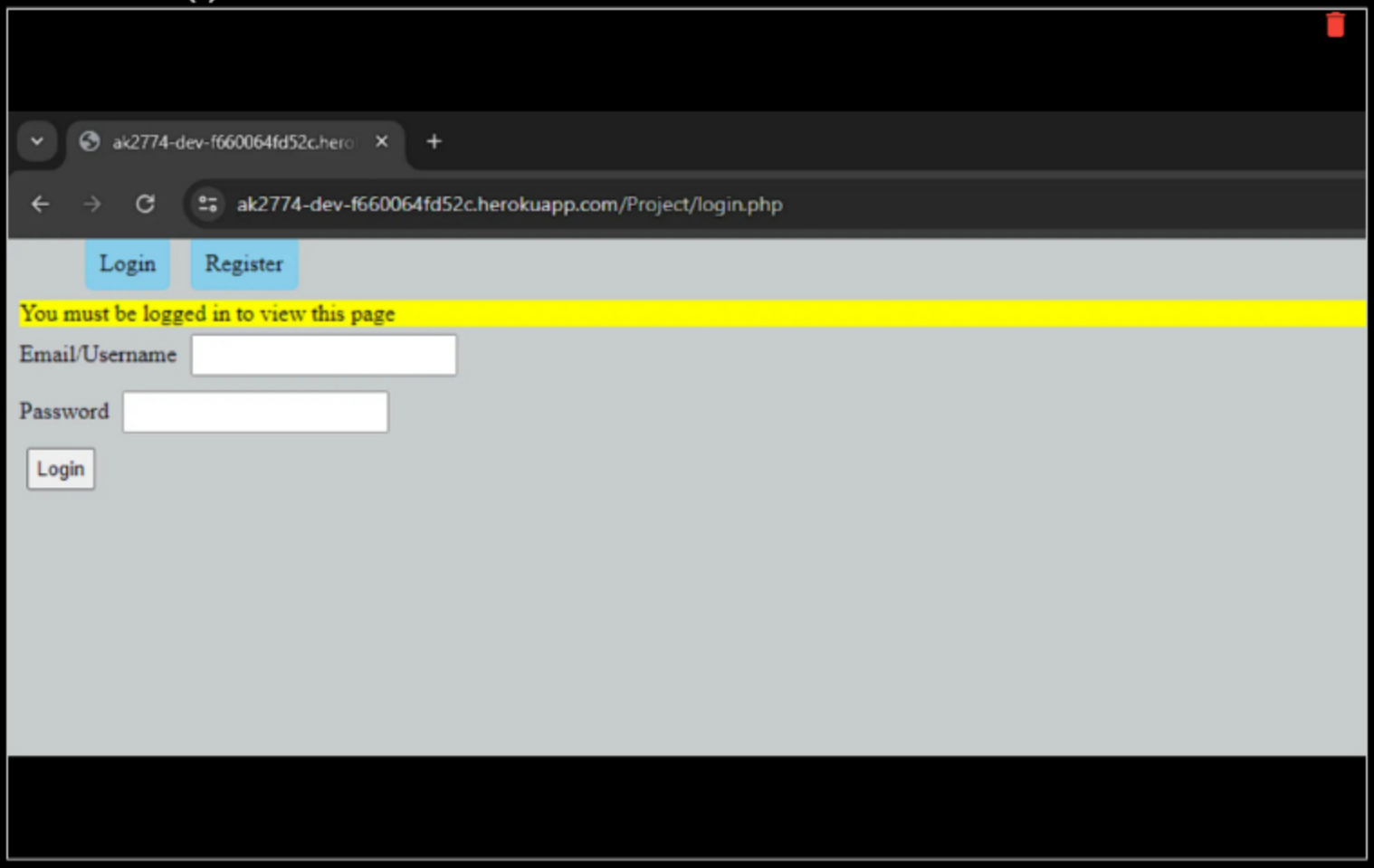
Checklist Items (0)

```php
<?php
require(__DIR__ . "/../../partials/nav.php");
?>
<h1>Home</h1>
<?php

if (is_logged_in(true)) {
    //comment this out if you don't want to see the session variables
    error_log("Session data: " . var_export($_SESSION, true));       //ak2774, 4/1/2024
}
?>
<?php
require(__DIR__ . "/../../partials/flash.php");
?>
```

Used on home.php

Checklist Items (0)



Message of manual access

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Authorization Screenshots

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Screenshot of the function that checks for a specific role
☐ #2	1	Screenshot of the role check function being used. Also caption what pages it's used on
☐ #3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)
☐ #4	1	Demonstrate the user-friendly message of trying to manually access a role-protected page while being logged out

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```php
function has_role($role)
{
    if (is_logged_in() && isset($_SESSION["user"]["roles"])) {
        foreach ($_SESSION["user"]["roles"] as $r) {
            if ($r["name"] === $role) {          //ak2774, 4/1/2024
                return true;
            }
        } <- #20-24 foreach ($_SESSION["user"]["roles"] as $r)
    } <- #19-25 if (is_logged_in() && isset($_SESSION["user"]["roles"]))
    return false;
} <- #18-27 function has_role($role)
```


Screenshot of has_role function

Checklist Items (0)

```php
if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");  //ak2774, 4/1/2024
    die(header("Location: $BASE_PATH" . "/home.php"));
}
```

Used in assign_roles

Checklist Items (0)

```php
if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");   //ak2774, 4/1/2024
    die(header("Location: " . get_url("home.php")));
}
```

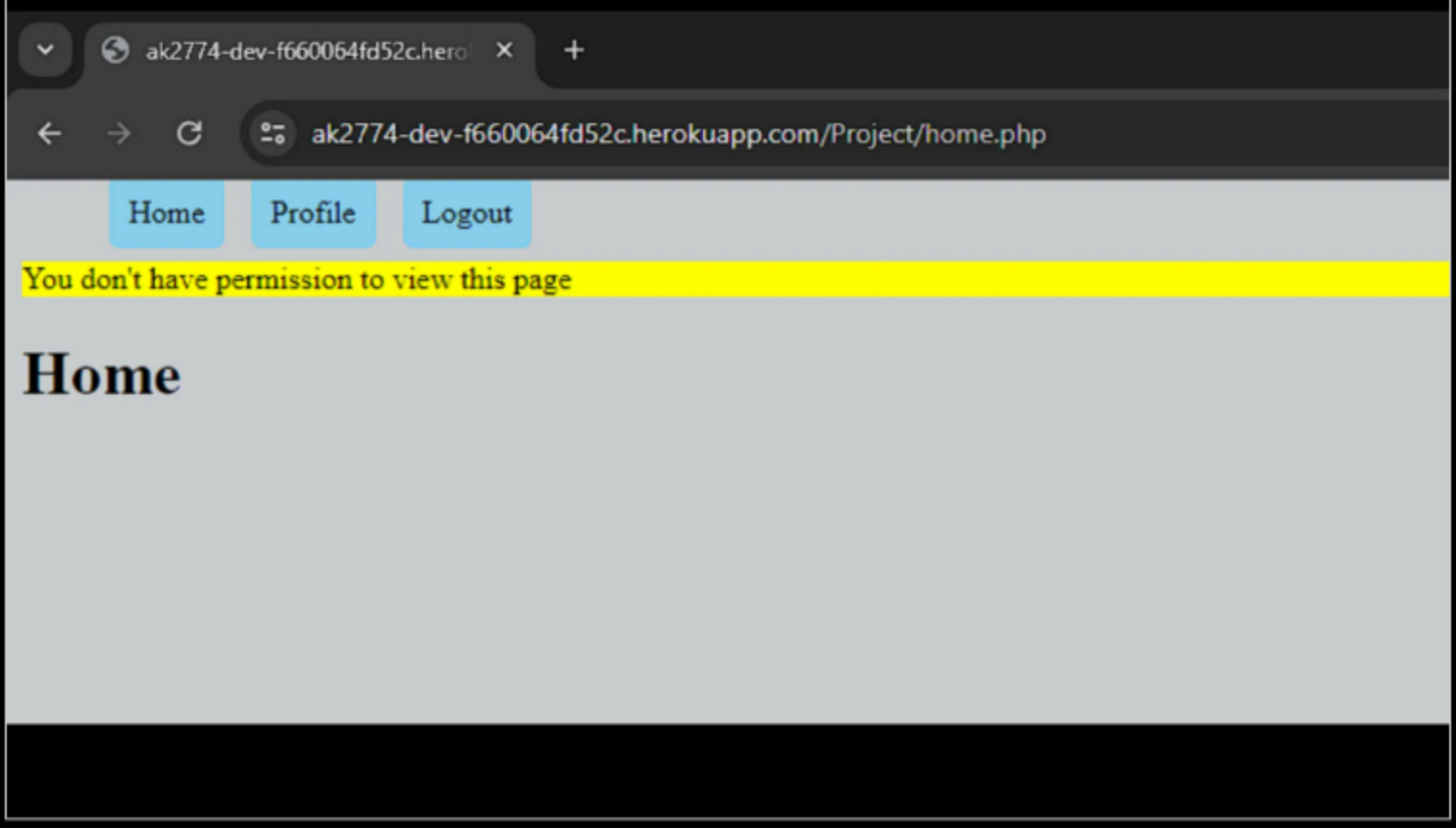
Used in create_role

Checklist Items (0)

```php
if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");   //ak2774, 4/1/2024
    die(header("Location: " . get_url("home.php")));
}
```

Used in list_roles

Checklist Items (0)



Message showing manually accessing a page user doesn't have access to

Checklist Items (0)

COLLAPSE

Task #3 - Points: 1

Text: Screenshots of UserRoles and Roles Tables

Checklist

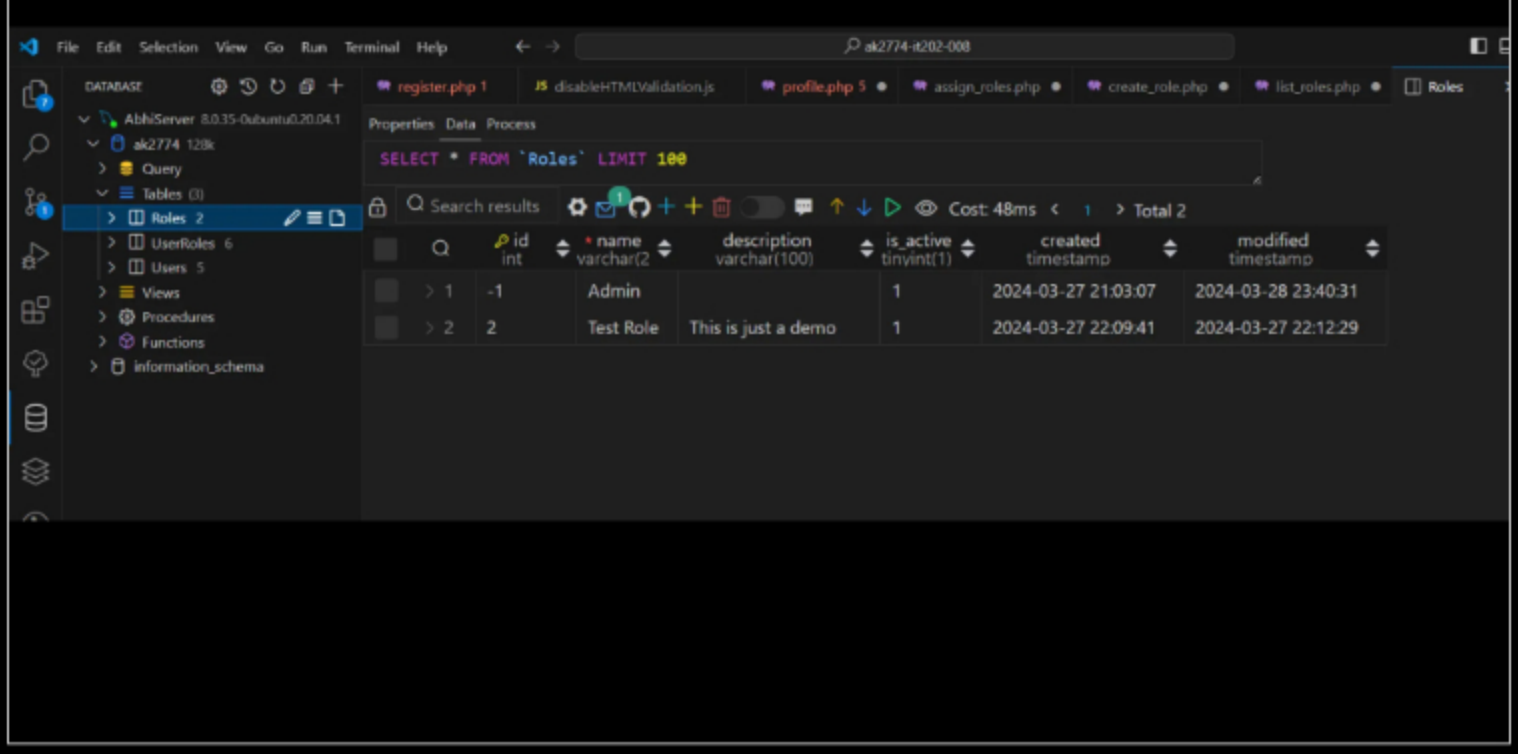
*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	At least one valid and enabled User->Role reference (UserRoles table)
☐ #2	1	UserRoles Table should have id, user_id, role_id, is_active, created, and modified columns
☐ #3	1	Roles Table should have id, name, description, is_active, modified, and created columns
☐ #4	1	At least one valid and enabled Role (Roles table)
☐ #5	1	Ensure left panel or database name is present in each table screenshot (should contain your ucid)

Task Screenshots:

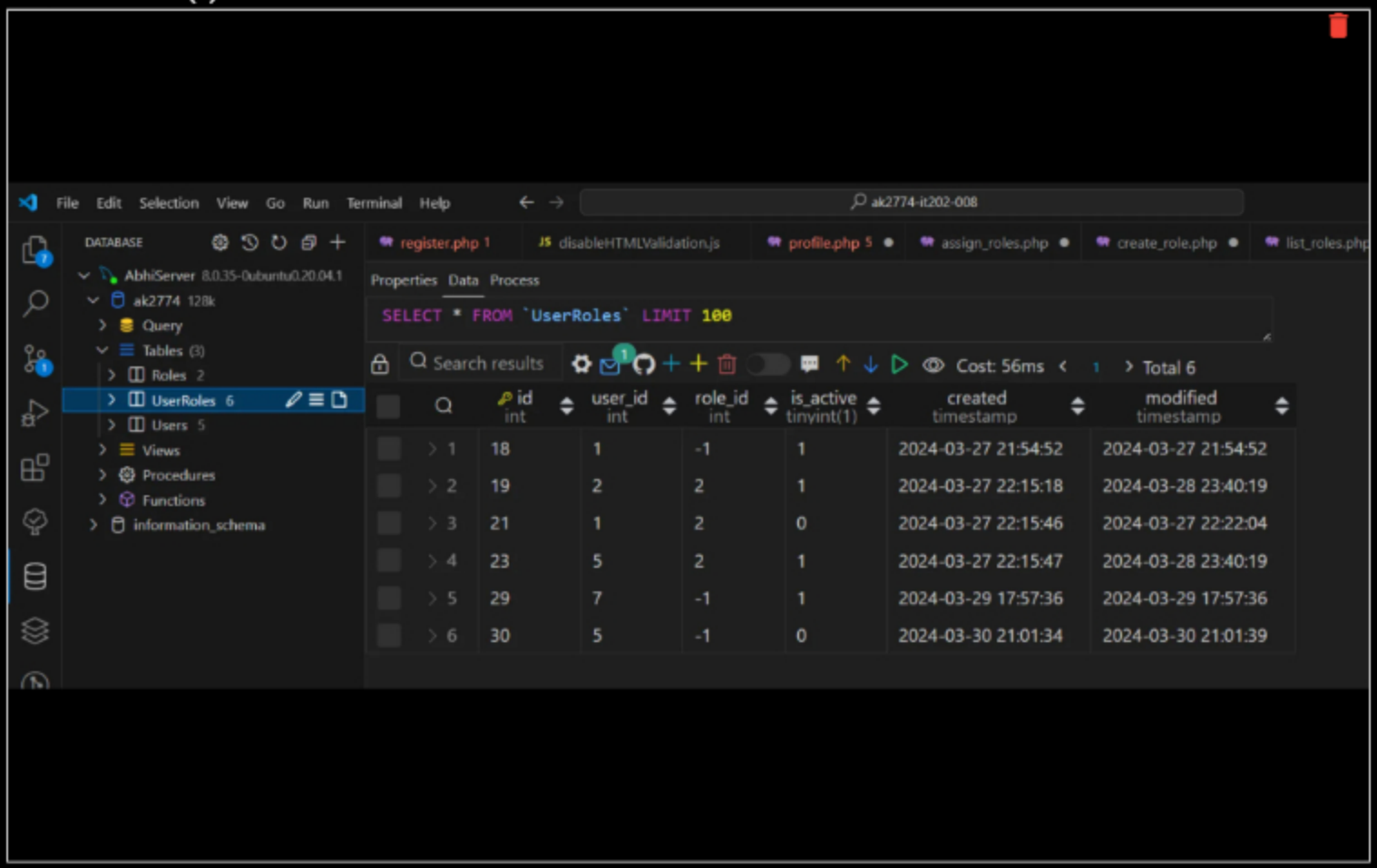
Gallery Style: Large View

Small Medium Large



Screenshot of roles table

Checklist Items (0)



Screenshot of userRoles table

Checklist Items (0)

COLLAPSE

Task #4 - Points: 1

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

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

Checklist *The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	What's the purpose of the UserRoles table?
☐ #2	1	How does Roles.is_active differ from UserRoles.is_active?

Response:

The purpose of the UserRoles table is to show which user has which role or roles active. Roles.is_active shows whether a role is active or not and UserRoles.is_active shows whether a user has a role active or not

COLLAPSE

Task #5 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/abhinavkarthikn/ak2774-it202-008/pull/37

User Profile (2 pts.)

COLLAPSE

Task #1 - Points: 1

Text: View Profile Website Page

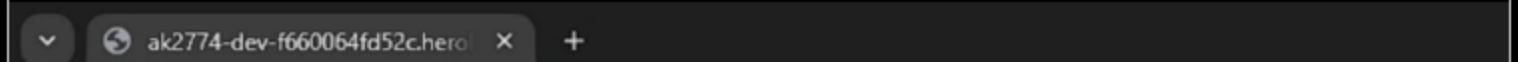
Checklist *The checkboxes are for your own tracking

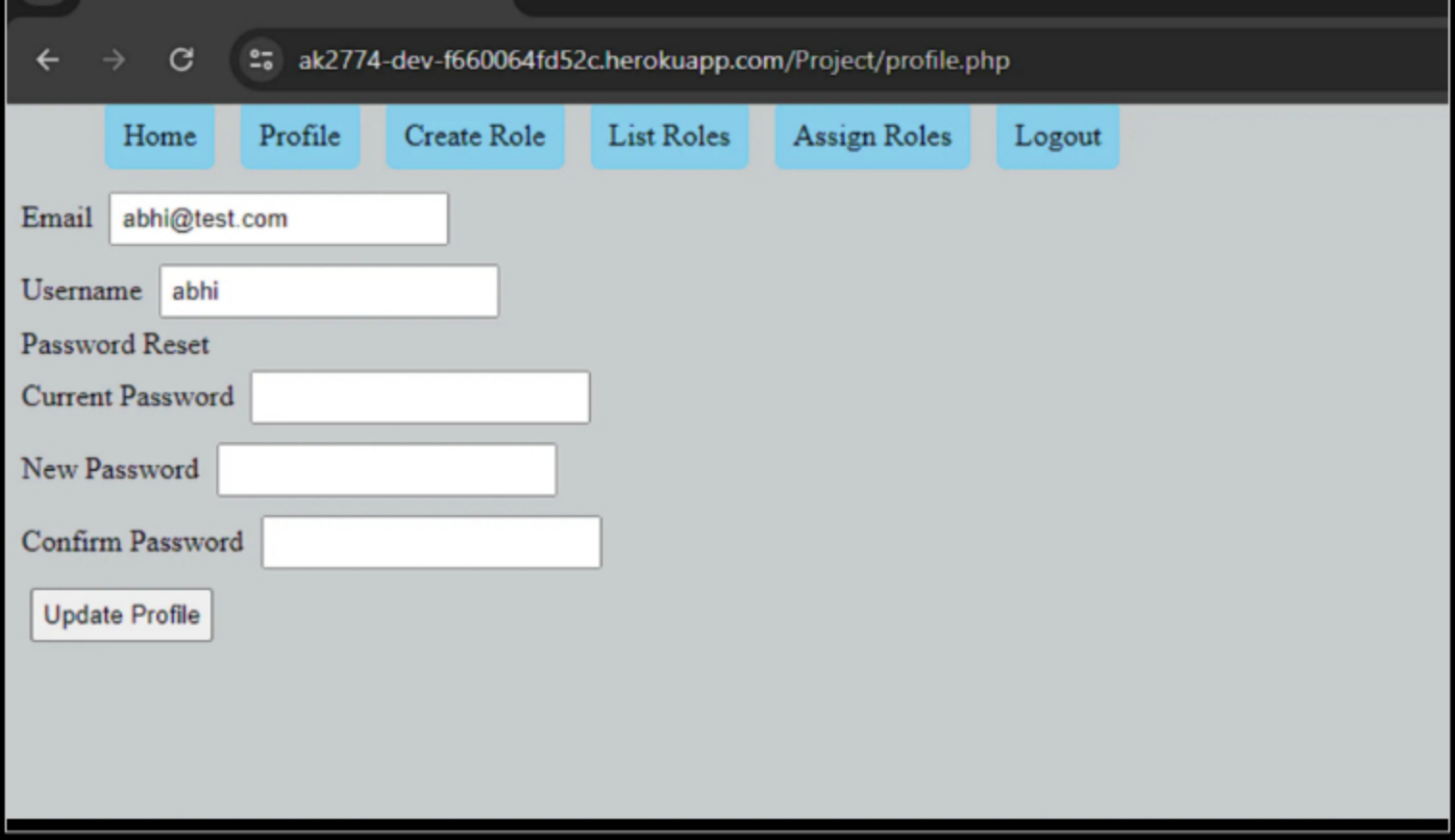
#	Points	Details
☐ #1	1	Heroku dev url should be present in the address bar
☐ #2	1	Should have thoughtful CSS applied
☐ #3	1	Show the profile form correctly populated on page load (username, email)
☐ #4	1	Should have the following fields: username, email, current password, new password, confirm password (or similar)

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Screenshot of profile page

Checklist Items (0)

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

Response:

Navigation and Authentication:

The partials/nav.php file is required to include the navigation bar. The is_logged_in(true) function is called to ensure that the user is logged in. If the user is not logged in, they are redirected to the login page.

Handling Form Submission:

If the form is submitted (isset($_POST["save"])), the PHP code executes. It retrieves the submitted values for email and username using the se() function, which provides default values if the keys are not found in the $_POST array. It prepares a SQL query to update the user's email and username in the database based on their user ID. The query is executed using a prepared statement, and if successful, a success message is flashed to the user.

Error Handling for Database Operations:

If an error occurs during the database operation, it is caught by a try-catch block. If the error code indicates a duplicate entry (1062), it checks the error message for details about which field caused the duplicate entry. Depending on the type of error, appropriate error messages are flashed to the user.

Fetching Updated User Data:

After updating the user's email and username, a new SQL query is prepared to fetch the updated user data from the database based on their user ID. If the user data is successfully fetched, it is stored in the $_SESSION["user"] array, updating the email and username values. If the user doesn't exist in the database, an error message is flashed to the user.

Handling Password Reset:

The current password, new password, and confirm password fields are retrieved from the form submission. If all fields are not empty, the current password is verified against the hashed password stored in the database using password_verify(). If the current password is valid, the new password is hashed using password_hash() and updated in the database. If the current password is invalid or the new passwords don't match, appropriate error messages are flashed to the user.

Form Population:

The user's current email and username are fetched using get_user_email() and get_username() functions, respectively. These values are preloaded into the corresponding input fields in the HTML form.

Client-Side Validation (JavaScript):

The validate() function is defined to perform client-side validation before form submission. It checks if the email and username fields are empty and if they match the required patterns. It also validates the new password fields for length and matching confirmation. If any validation fails, error messages are flashed using the flash() function.

COLLAPSE

Task #3 - Points: 1

Text: Edit Profile Website Page

Checklist

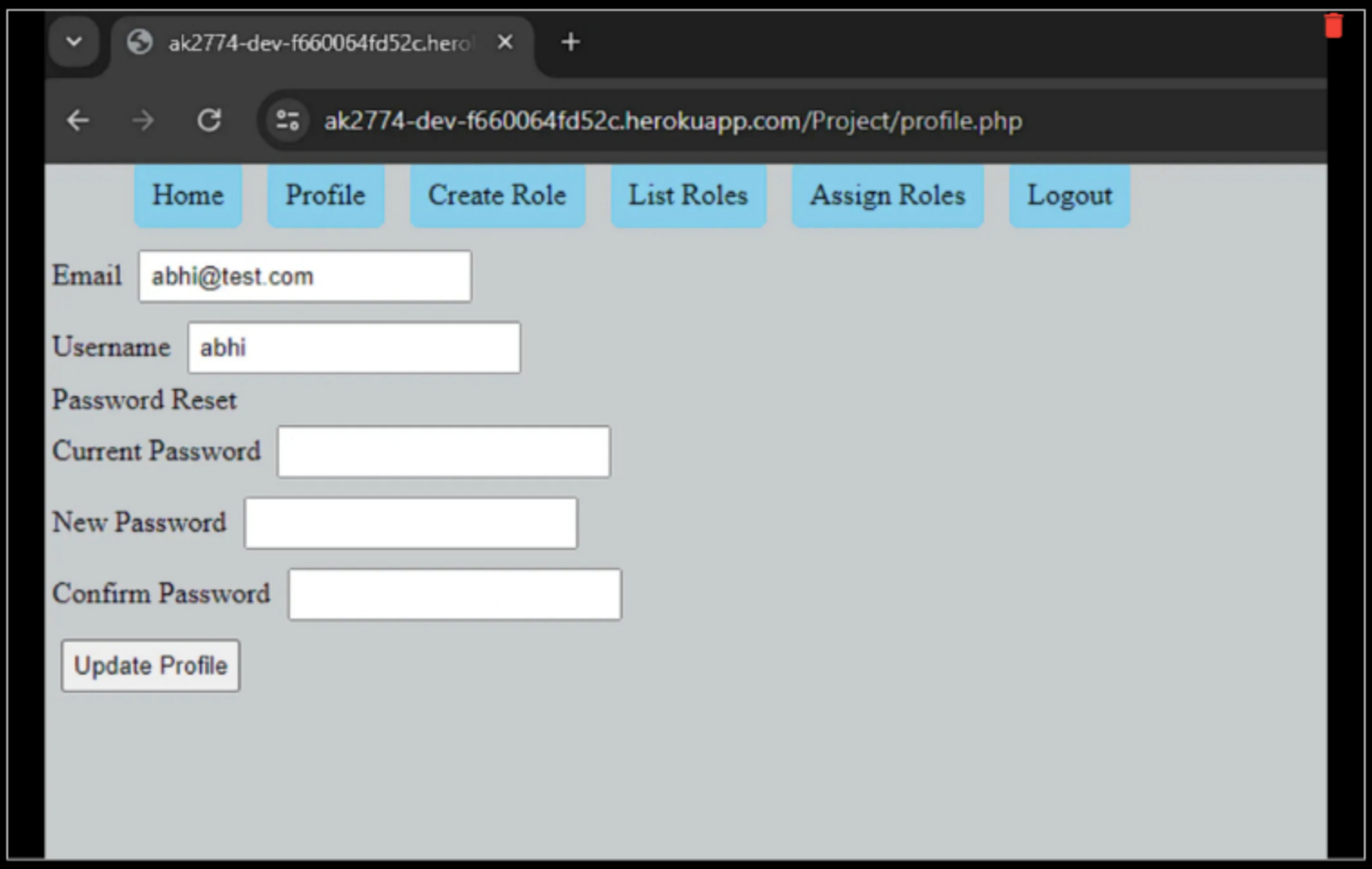
*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Heroku dev url should be present in the address bar
☐ #2	1	Should have thoughtful CSS applied
☐ #3	1	Demonstrate with before and after of a username change (including success message)
☐ #4	1	Demonstrate with a before and after of an email change (including success message)
☐ #5	1	Demonstrate the success message of updating password
☐ #6	1	Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)
☐ #7	1	Demonstrate PHP user-friendly validation messages (desired email is already in use, desired username is already in use, current password doesn't match what's in the DB)

Task Screenshots:

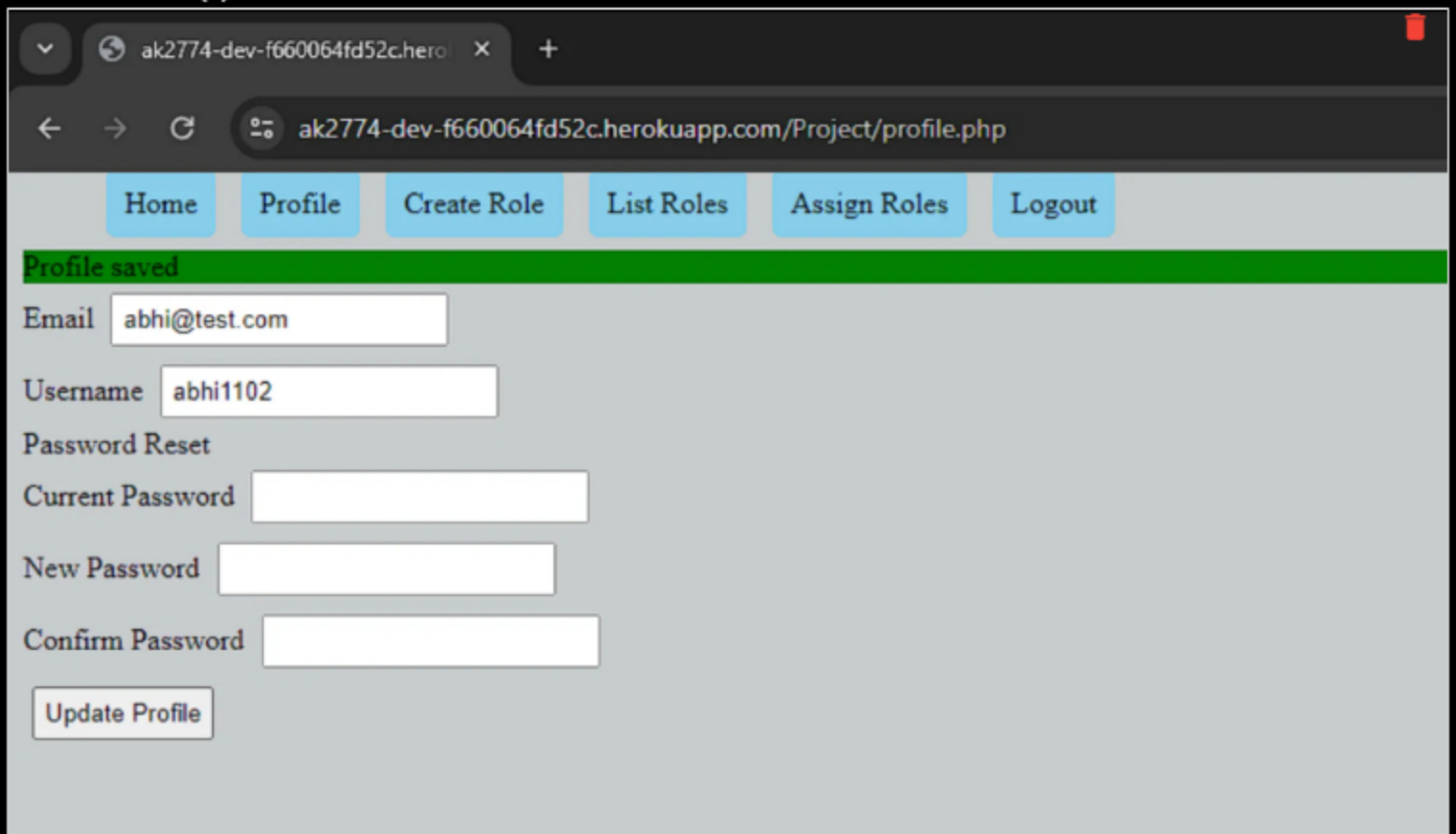
Gallery Style: Large View

Small Medium Large



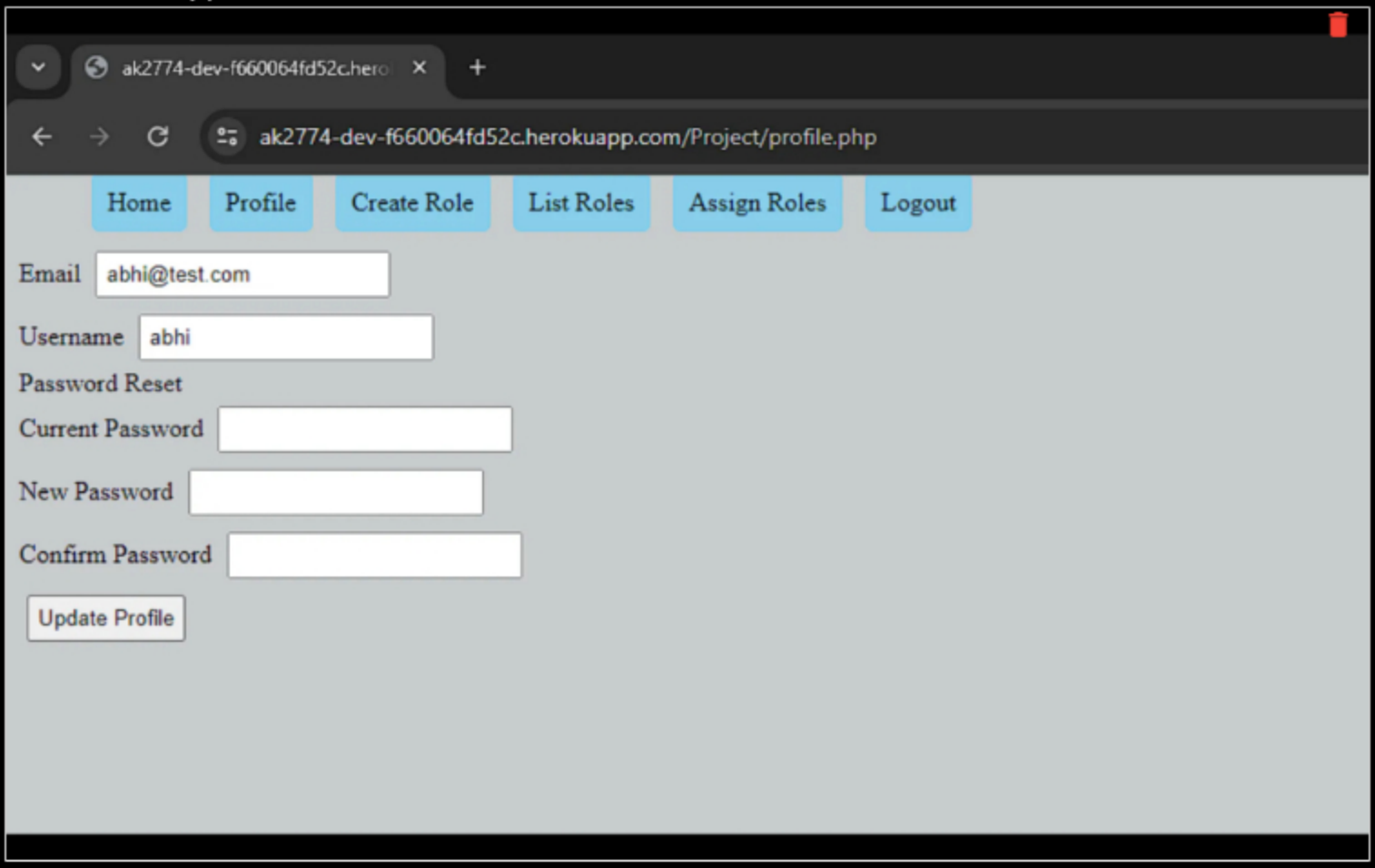
Before username change

Checklist Items (0)



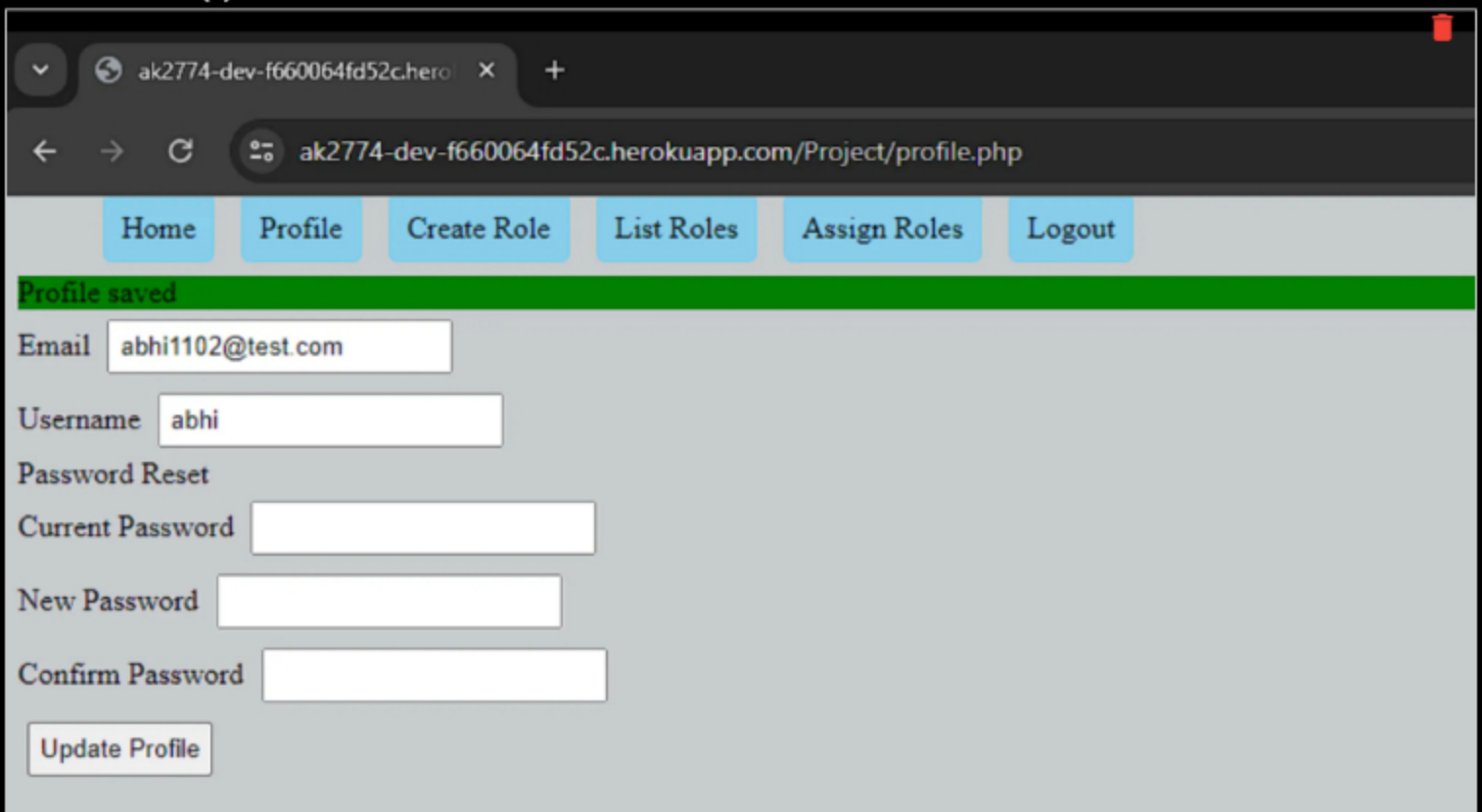
After username change

Checklist Items (0)



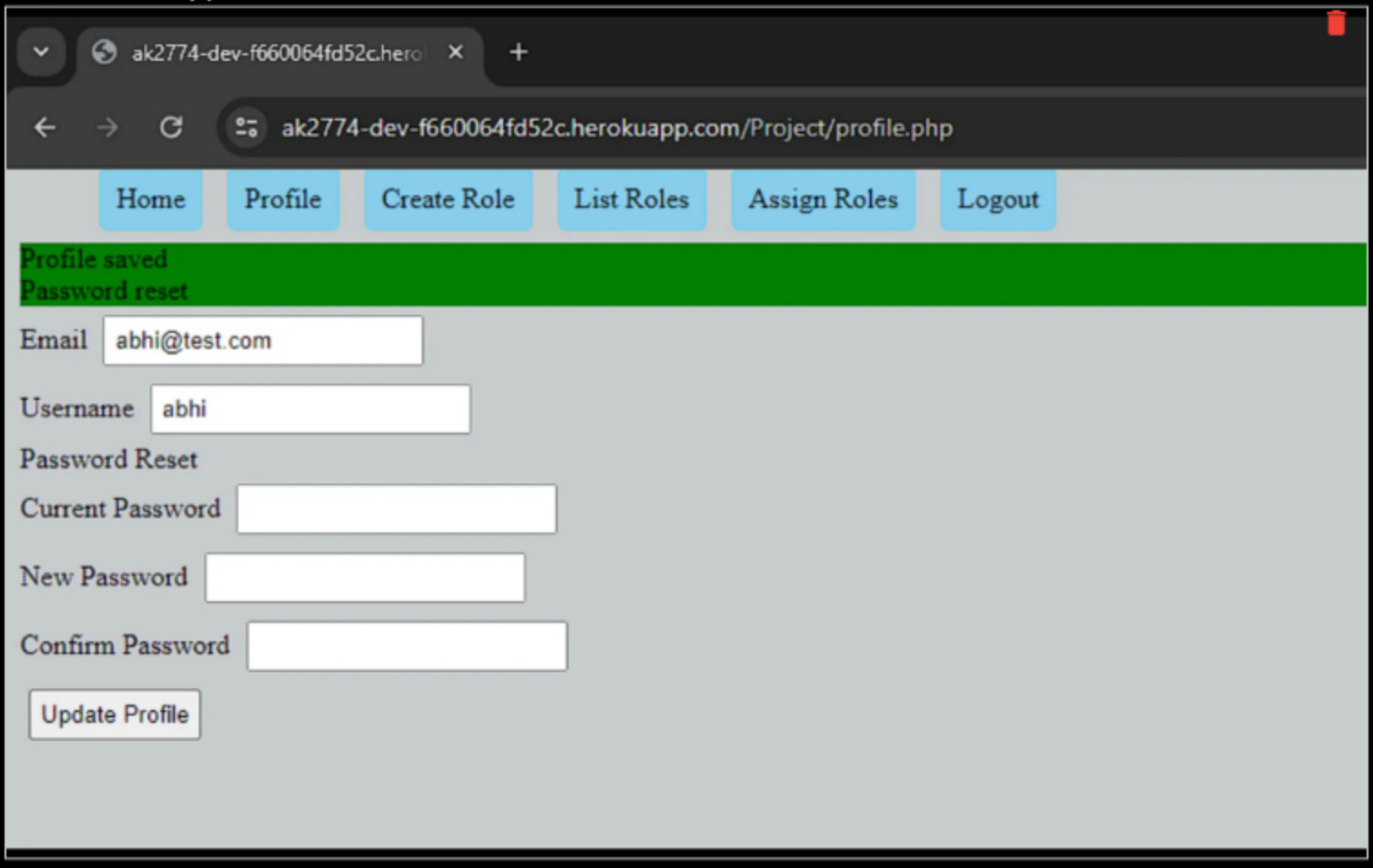
Before email change

Checklist Items (0)



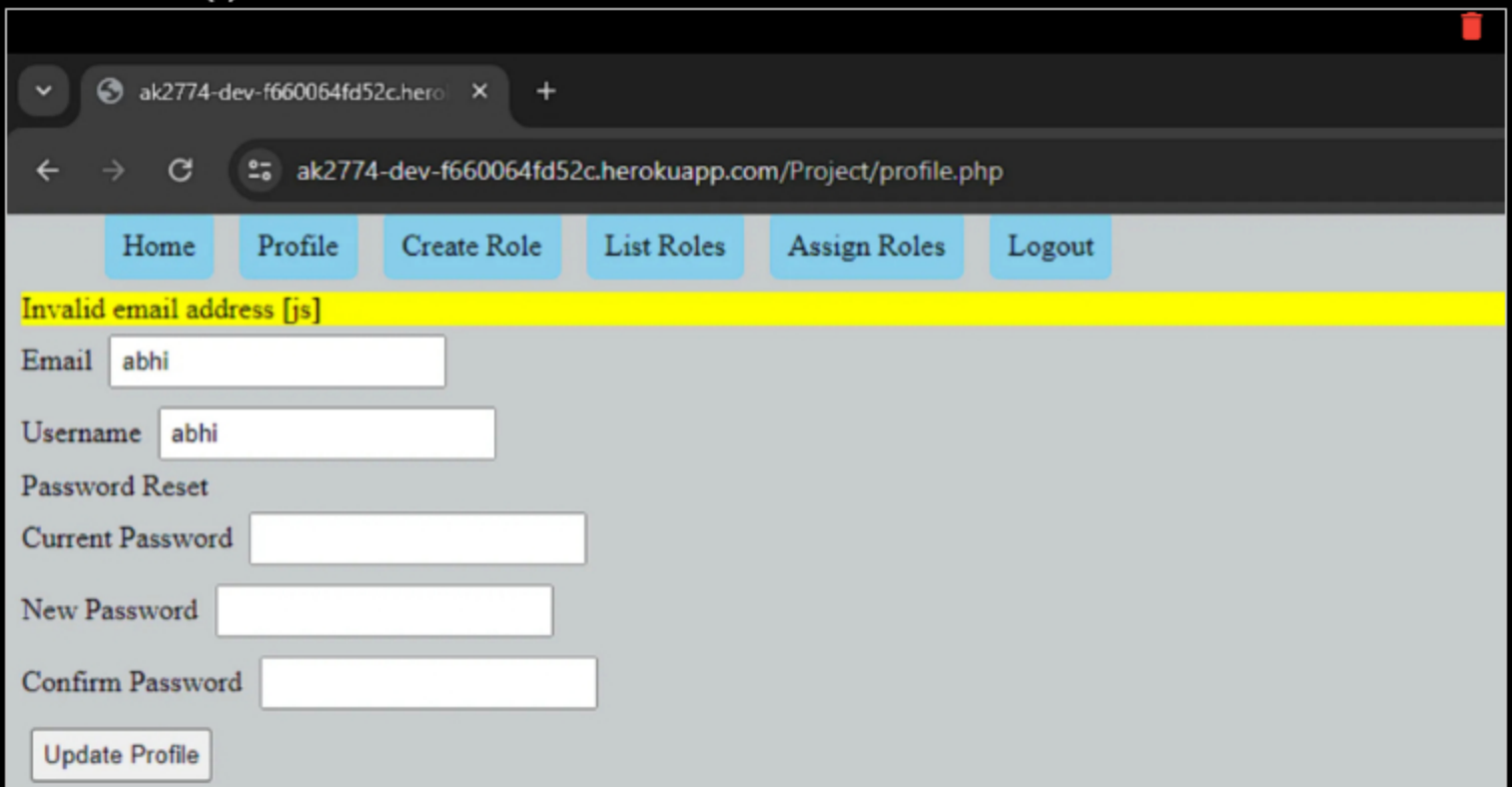
After email change

Checklist Items (0)



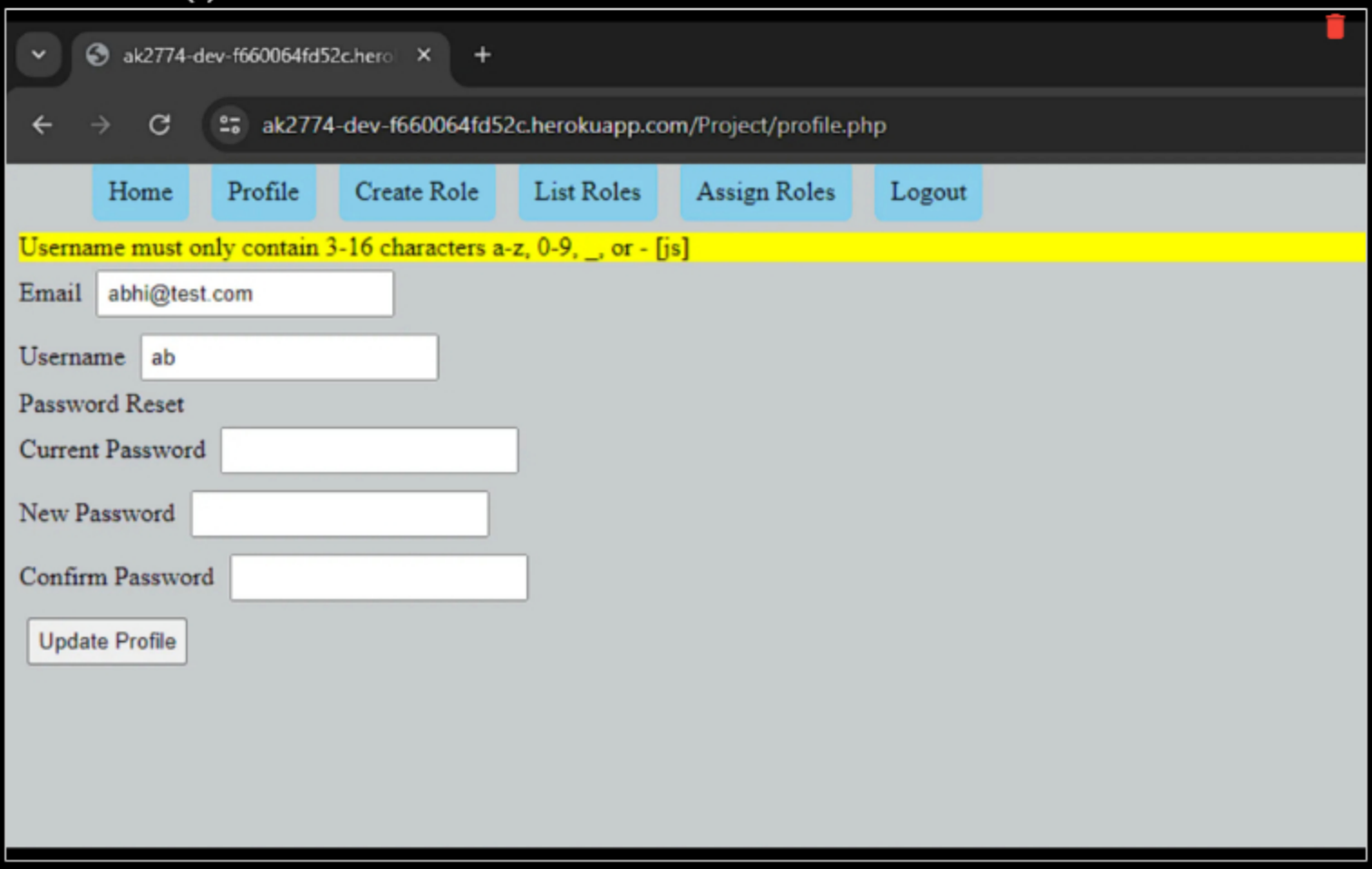
Password reset message

Checklist Items (0)



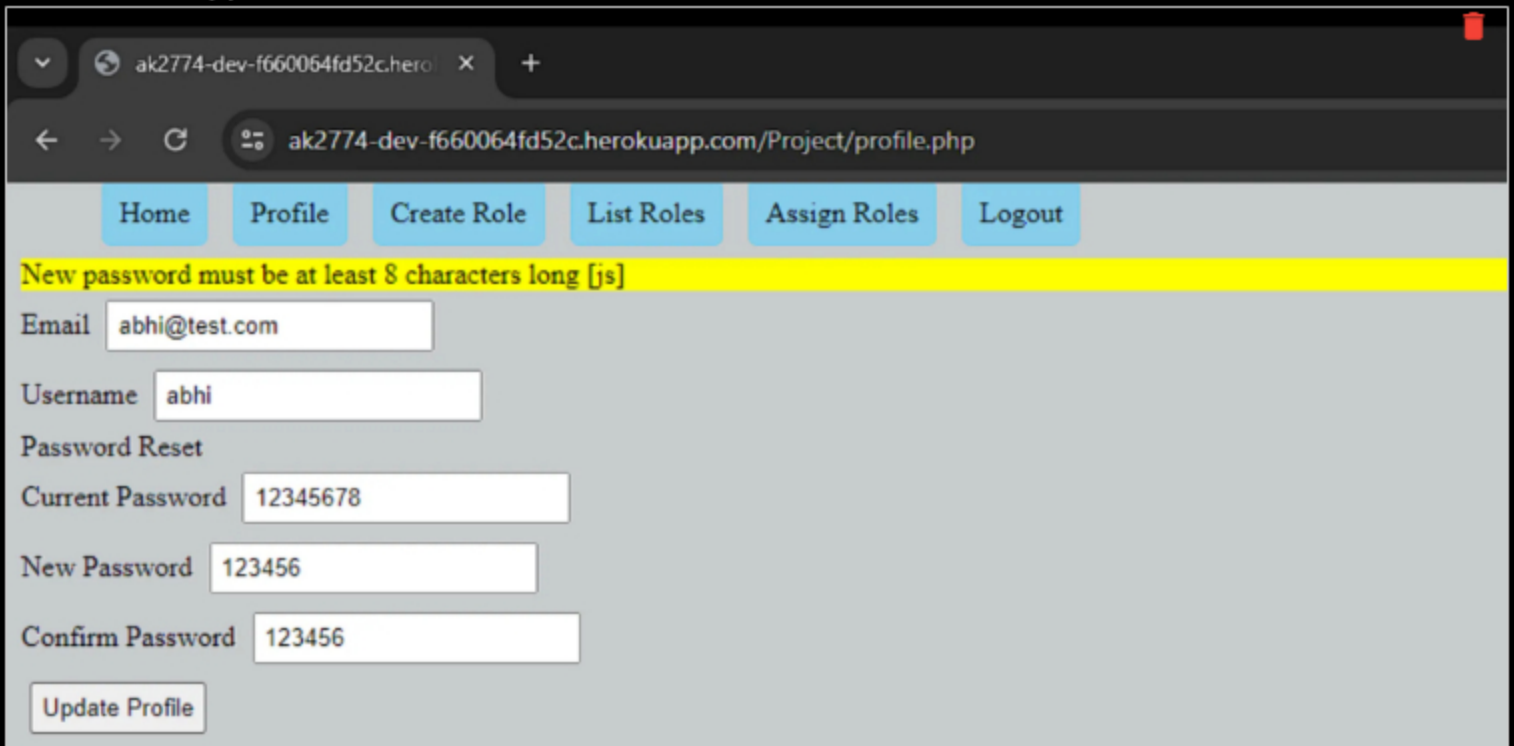
JS validation of email format

Checklist Items (0)



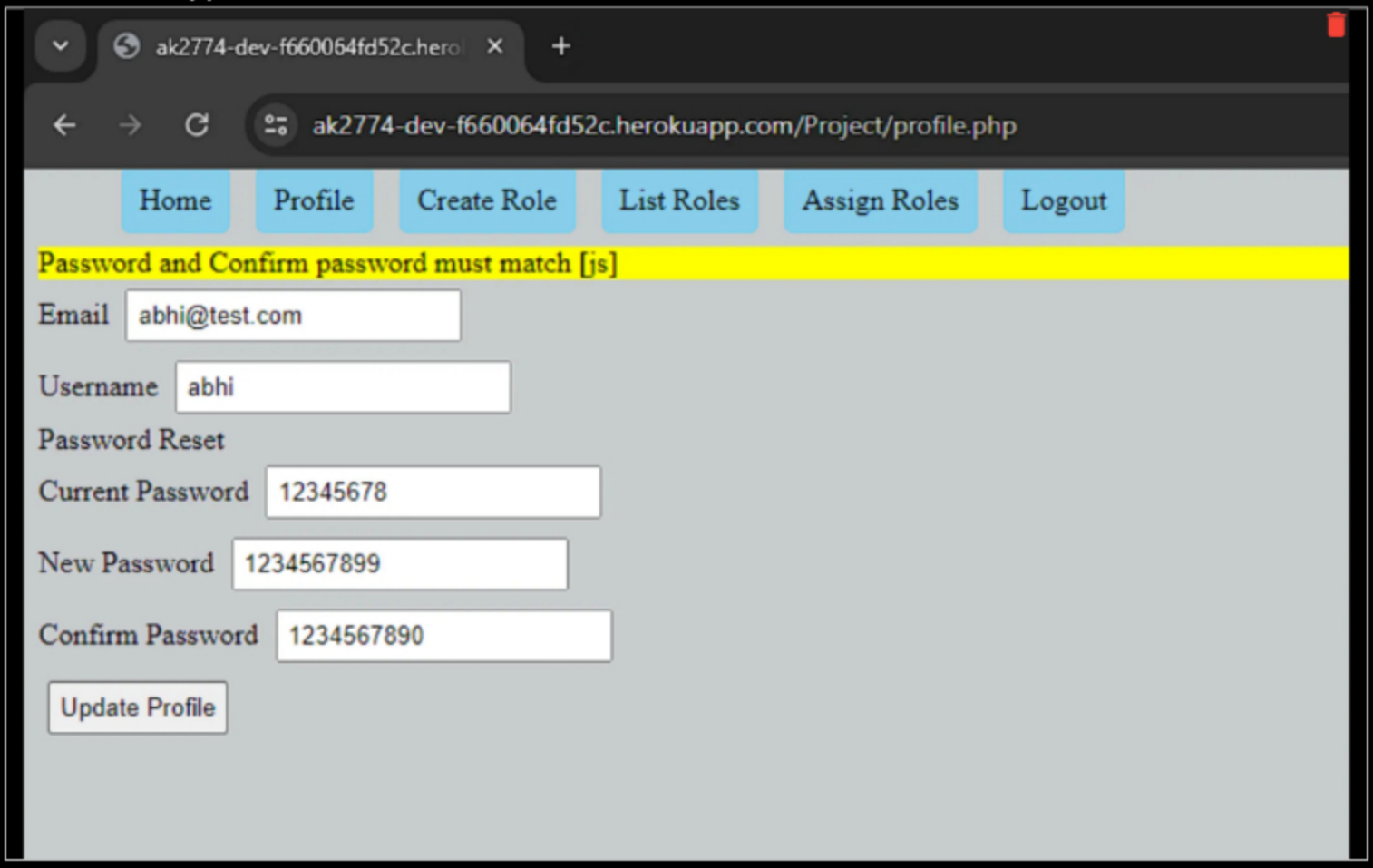
JS validation of username format

Checklist Items (0)



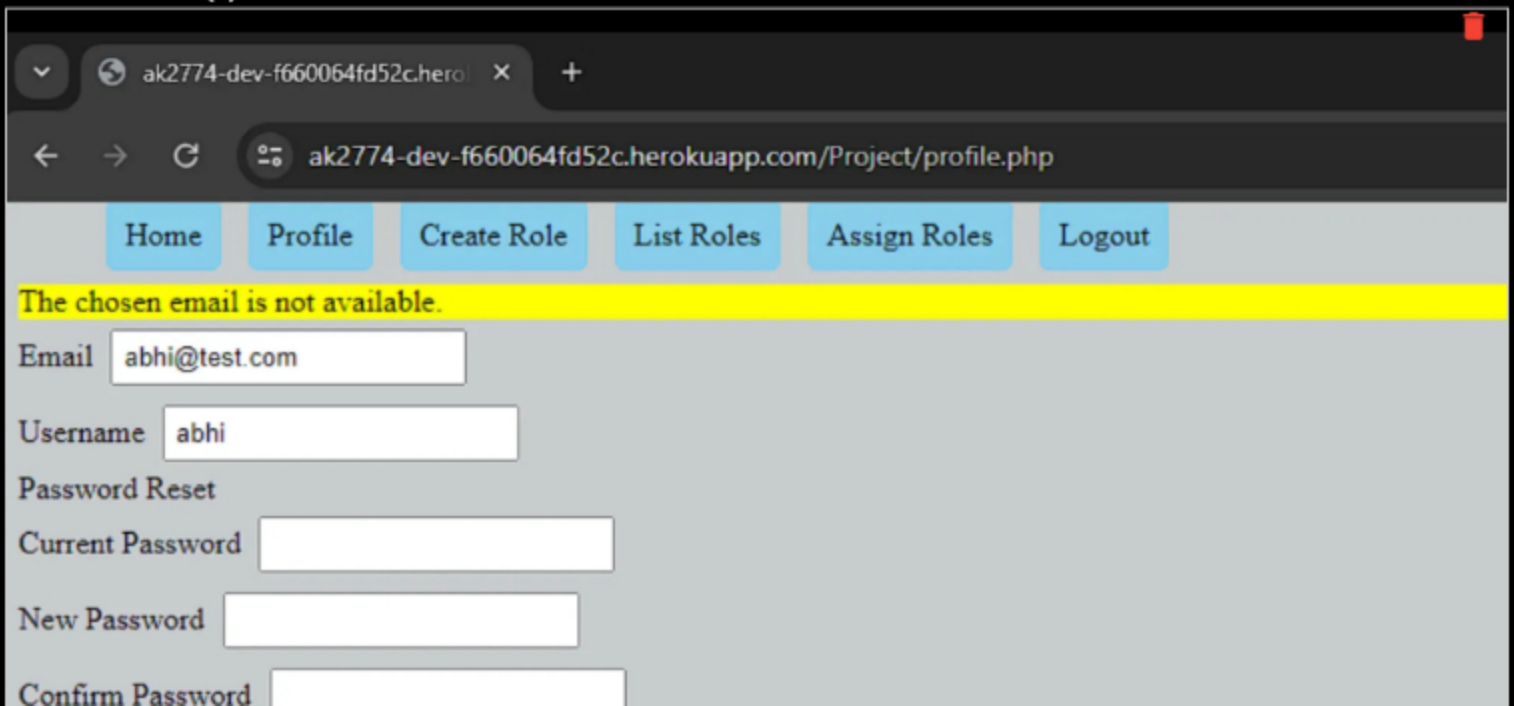
JS validation of new password format

Checklist Items (0)



JS validation of new and confirm password matching

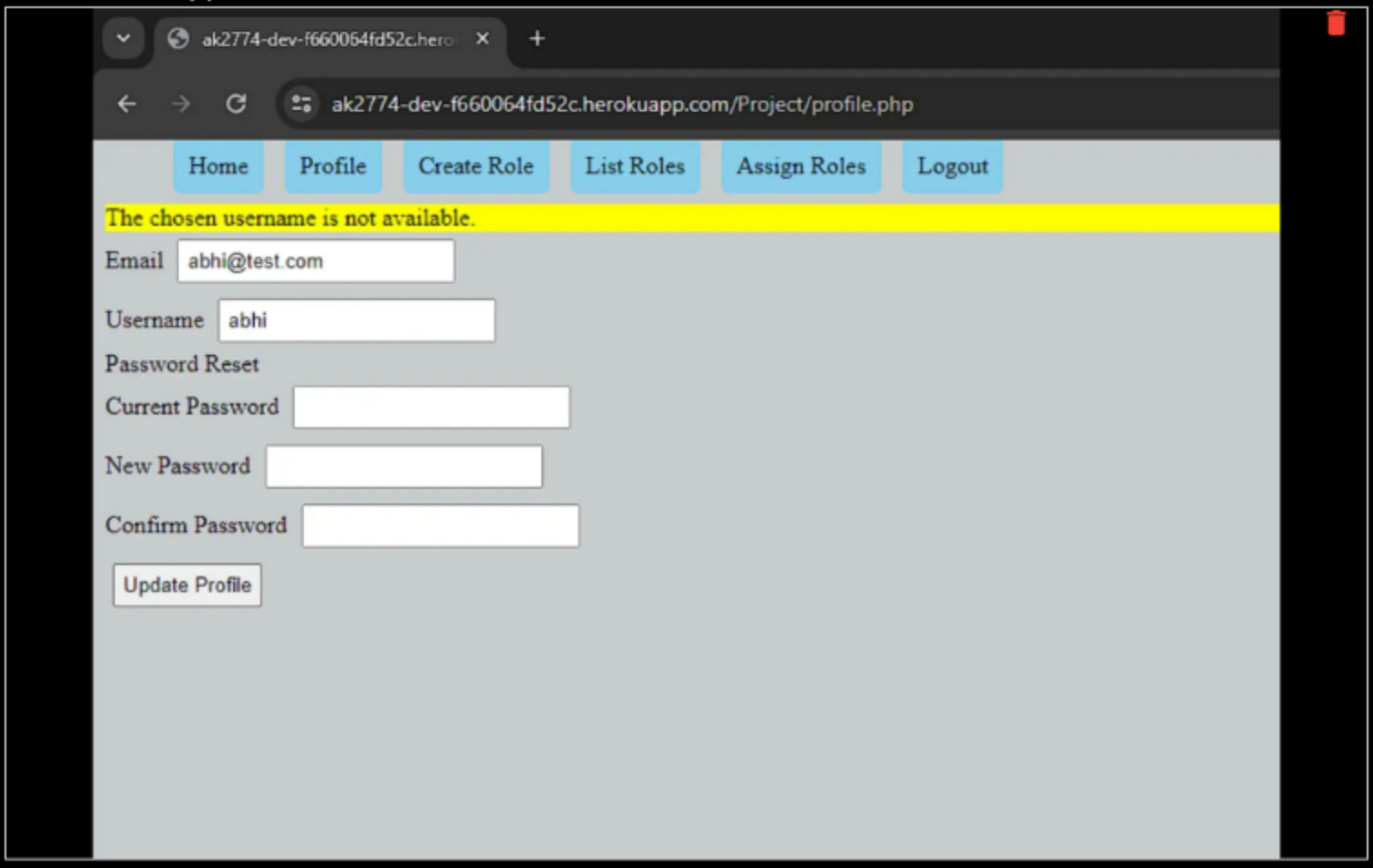
Checklist Items (0)



Update Profile

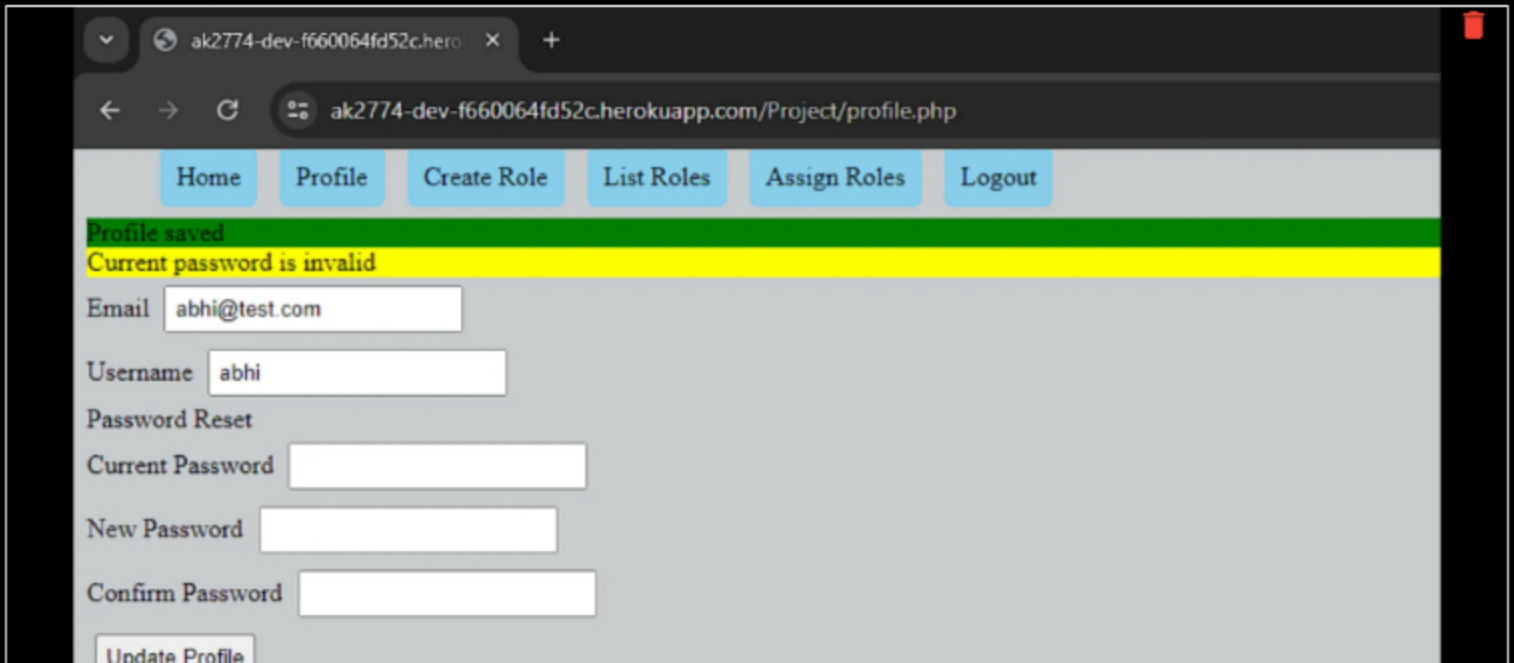
Message of email already in use

Checklist Items (0)



Message of username already in use

Checklist Items (0)



Message that current password is not correct

Checklist Items (0)

COLLAPSE

Task #4 - Points: 1

Text: Explain the logic step-by-step of how the data is checked and saved for the following scenarios

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Updating Username/Email
☐ #2	1	Updating password
☐ #3	1	Don't just show code, translate things to plain English

Response:

Updating Username/Email:

Form Submission Handling:

When the form is submitted, the PHP code checks if the save button is clicked (isset($_POST["save"])). It retrieves the submitted values for email and username from $_POST. It prepares parameters for a SQL query to update the user's email and username in the database.

Database Update Query:

The SQL query is prepared to update the user's email and username in the Users table based on their user ID. The prepared statement is executed with the provided parameters.

Error Handling:

If the query execution is successful, a success message is flashed to the user indicating that the profile has been saved. If there's a duplicate entry error (error code 1062), it checks the error message to determine which field caused the duplicate entry. Depending on the error, appropriate error messages are flashed to the user.

Fetching Updated User Data:

After successfully updating the email and username in the database, a new query is prepared to fetch the updated user data. The user's data is fetched based on their user ID. If the user data is found, it is updated in the session to reflect the changes.

Updating Password:

Form Submission Handling:

The current password, new password, and confirm password fields are retrieved from the form submission. It checks if all fields are not empty.

Current Password Verification:

If all password fields are filled, it prepares a query to fetch the current hashed password from the database based on the user ID. It verifies the current password by comparing the hashed password retrieved from the database with the submitted current password using password_verify().

New Password Hashing and Update:

If the current password is verified successfully, the new password is hashed using password_hash(). A query is prepared to update the user's password in the database with the newly hashed password. The password update query is executed with the user ID and the new hashed password.

Error Handling:

If the current password is invalid, a warning message is flashed to the user. If the new passwords don't match, a warning message indicating that the passwords don't match is flashed.

Success Message:

If the password update is successful, a success message is flashed to the user indicating that the password has been reset.

COLLAPSE

Task #5 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/abhinavkarthikn/ak2774-it202-008/pull/33

Misc (1 pt.)

COLLAPSE

COLLAPSE

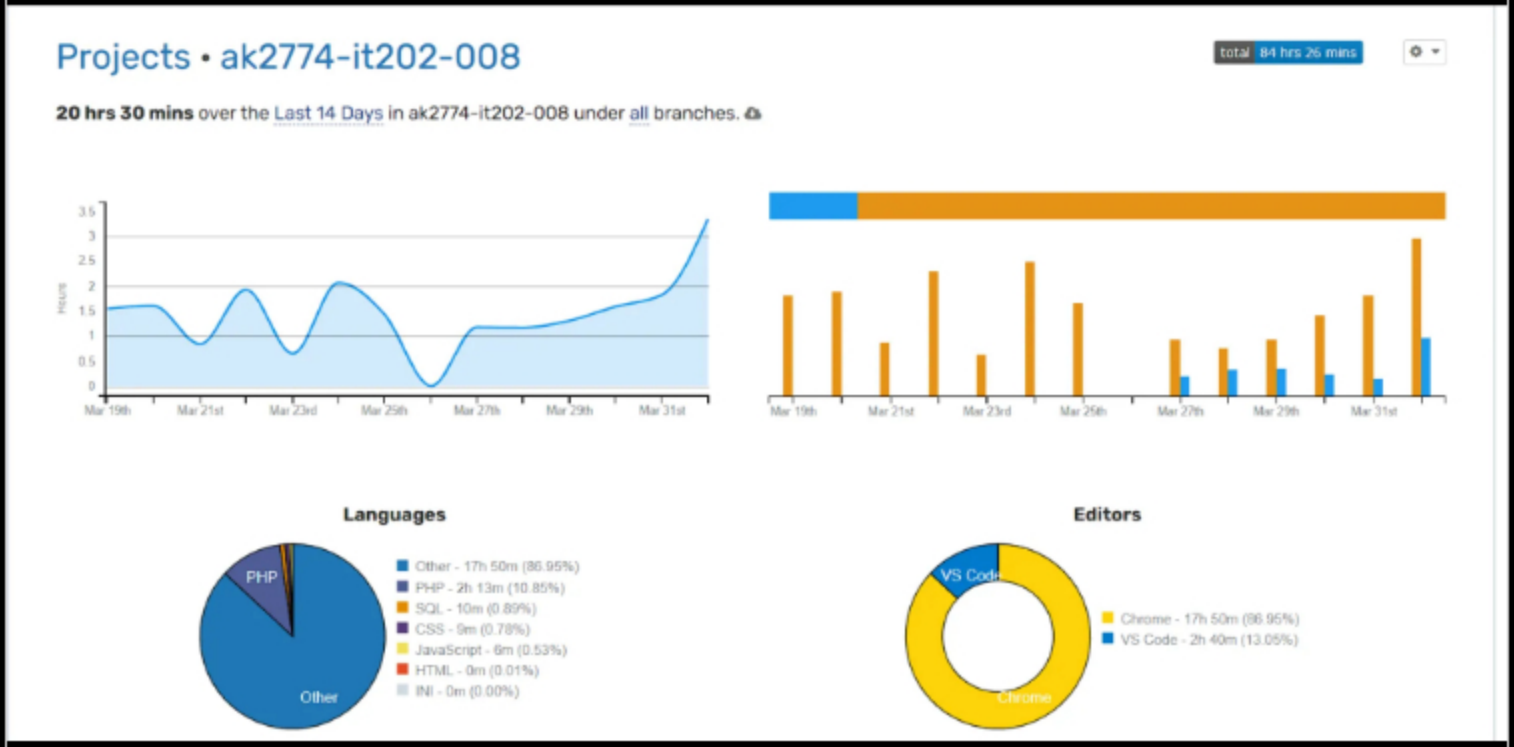
Task #1 - Points: 1

Text: Screenshot of wakatime

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Screenshot of wakatime

COLLAPSE

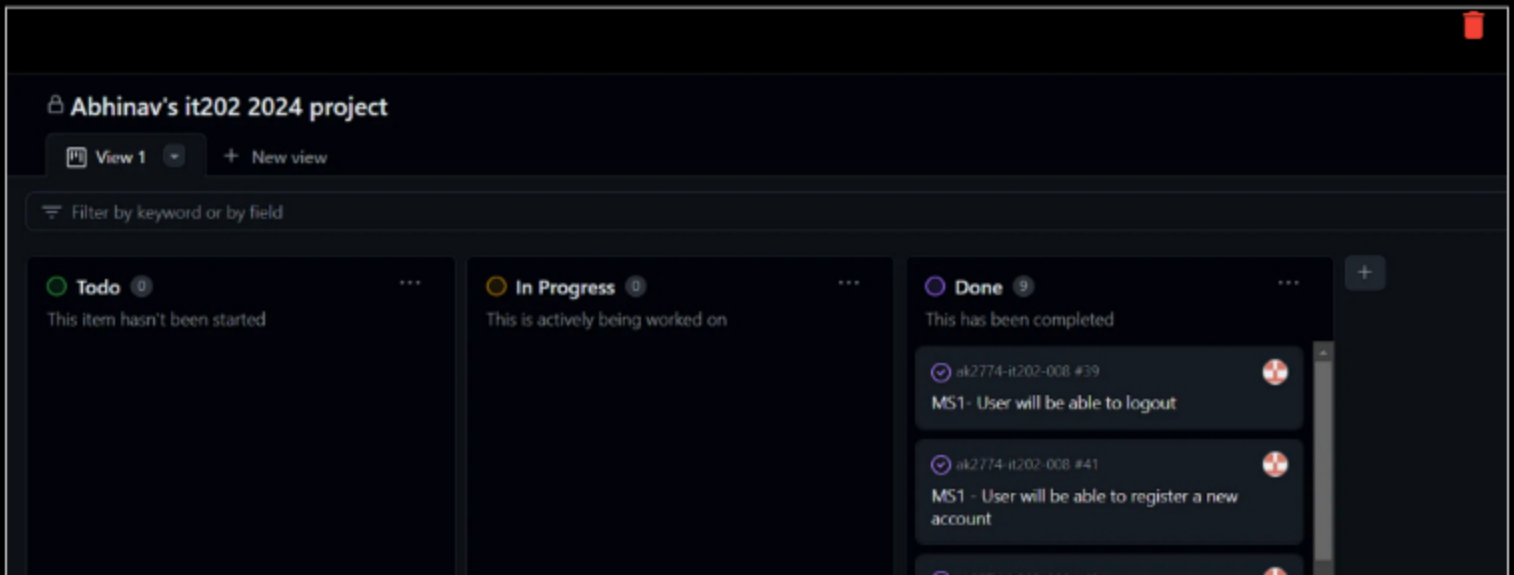
Task #2 - Points: 1

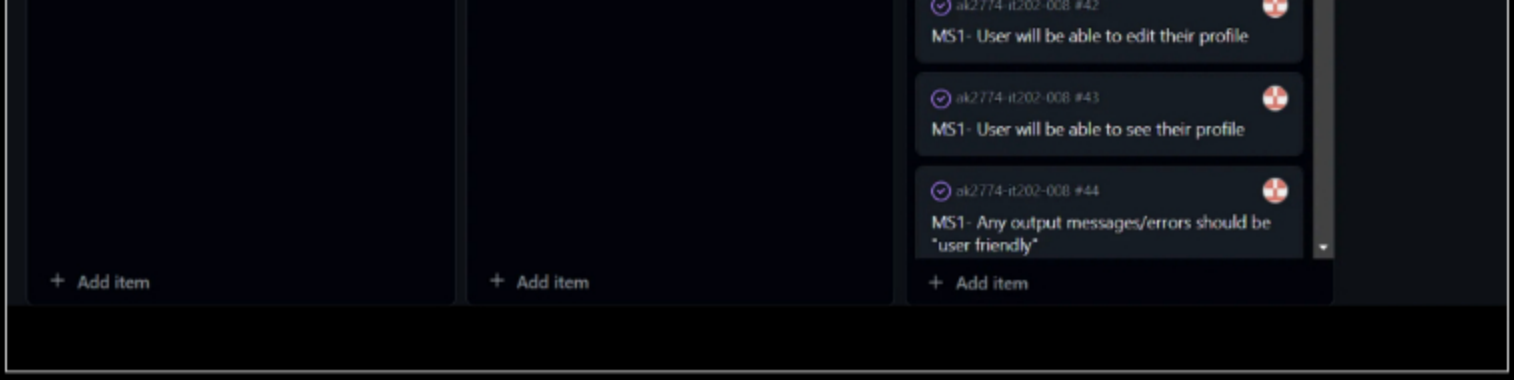
Text: Screenshot of your project board from GitHub (tasks should be in the proper column)

Task Screenshots:

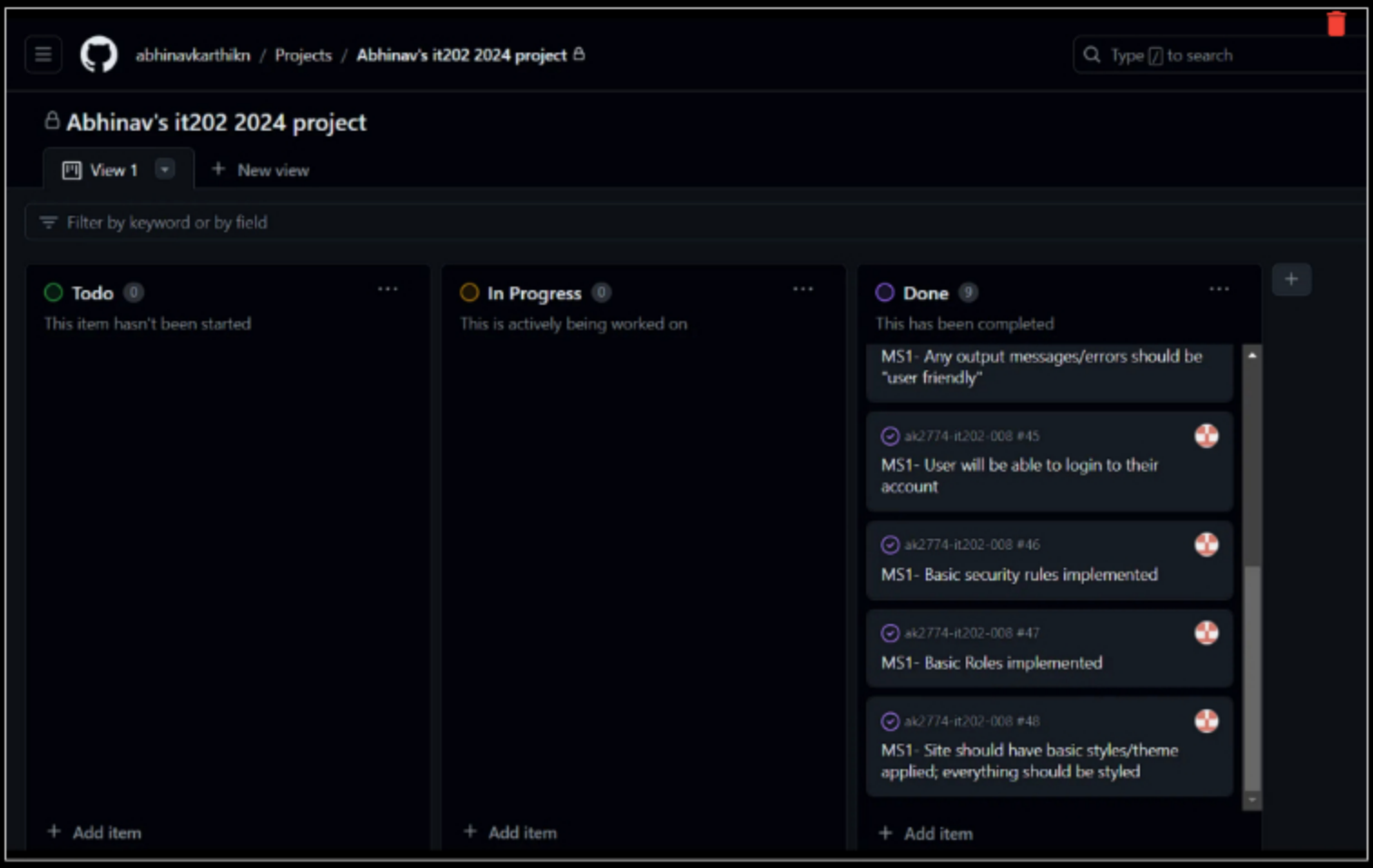
Gallery Style: Large View

Small Medium Large





Screenshot of Project board part 1



Screenshot of Project Board part 2

COLLAPSE

Task #3 - Points: 1

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

URL #1

https://github.com/users/abhinavkarthikn/projects/1/views/1

COLLAPSE

Task #4 - Points: 1

Text: Provide a direct link to the login page from your prod instance

URL #1

URL #1

https://ak2774-prod-76402b453c05.herokuapp.com/Project/login.php

End of Assignment