Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-008-S2024/it202-milestone-1-2024/grade/yc73

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

Submissions:

Submission Selection

1 Submission [active] 4/1/2024 7:37:14 AM

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

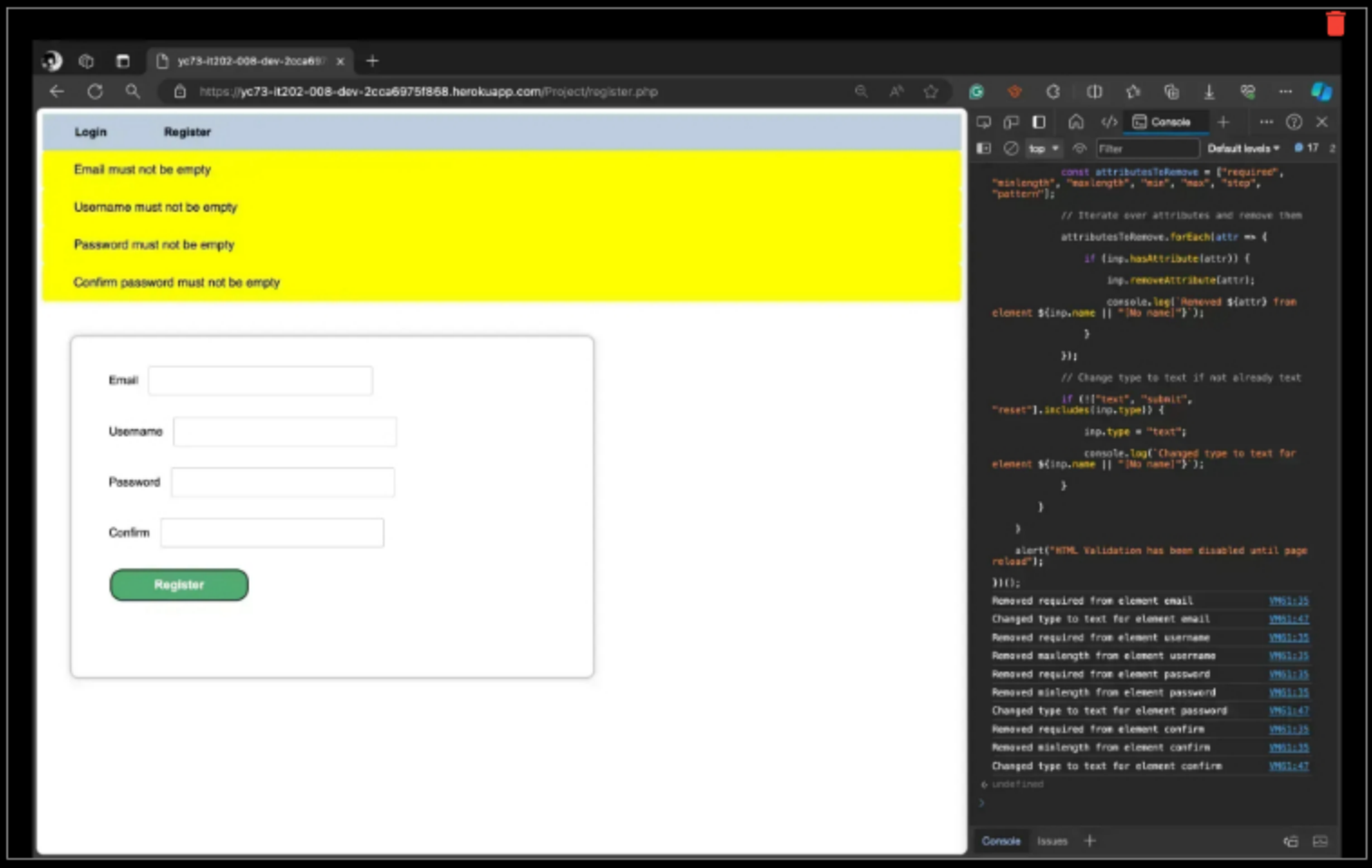
*The checkboxes are for your own tracking

#	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

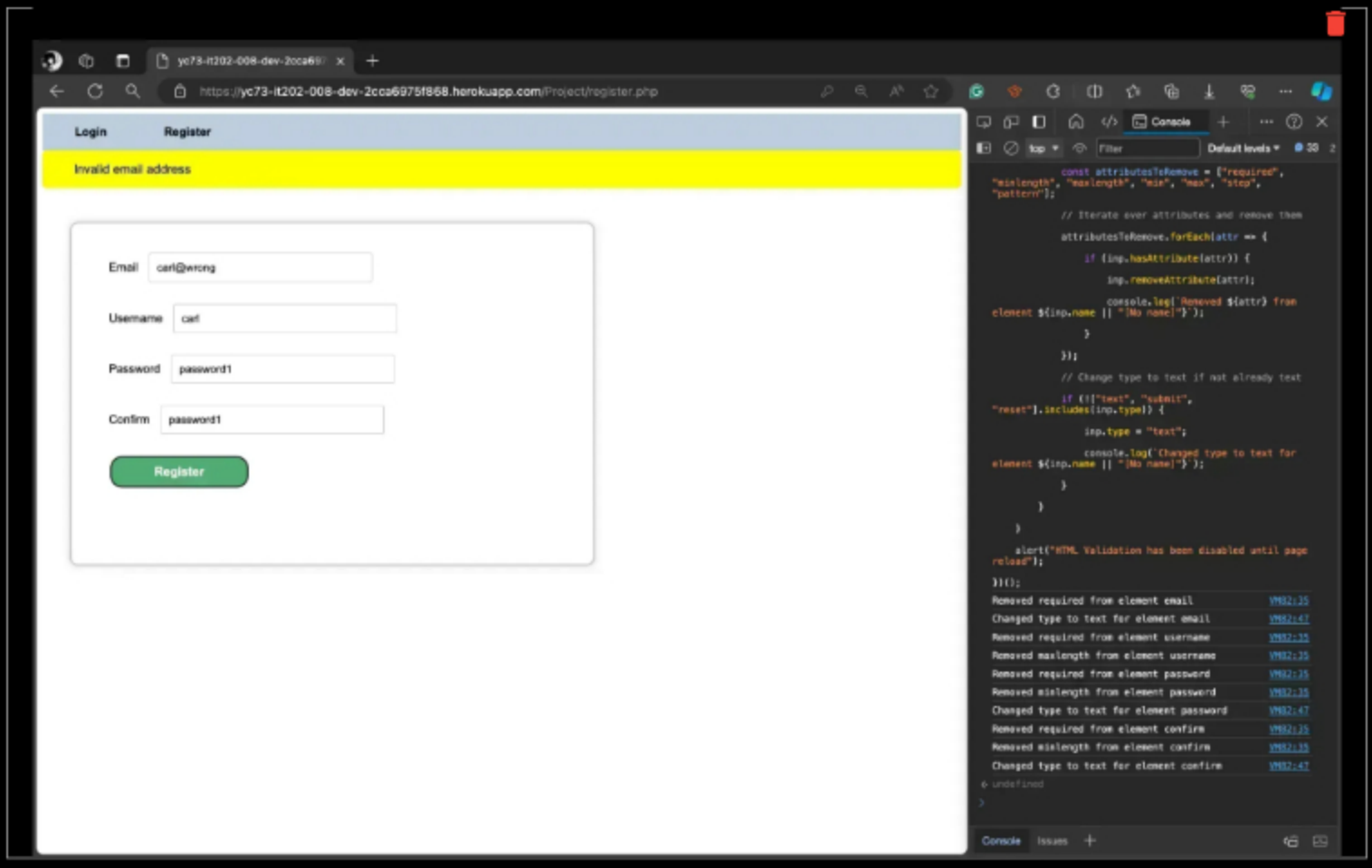
Small Medium Large



JavaScript email, username, and password validation => for empty fields (p.s. I made my JS validations "warnings")

Checklist Items (1)

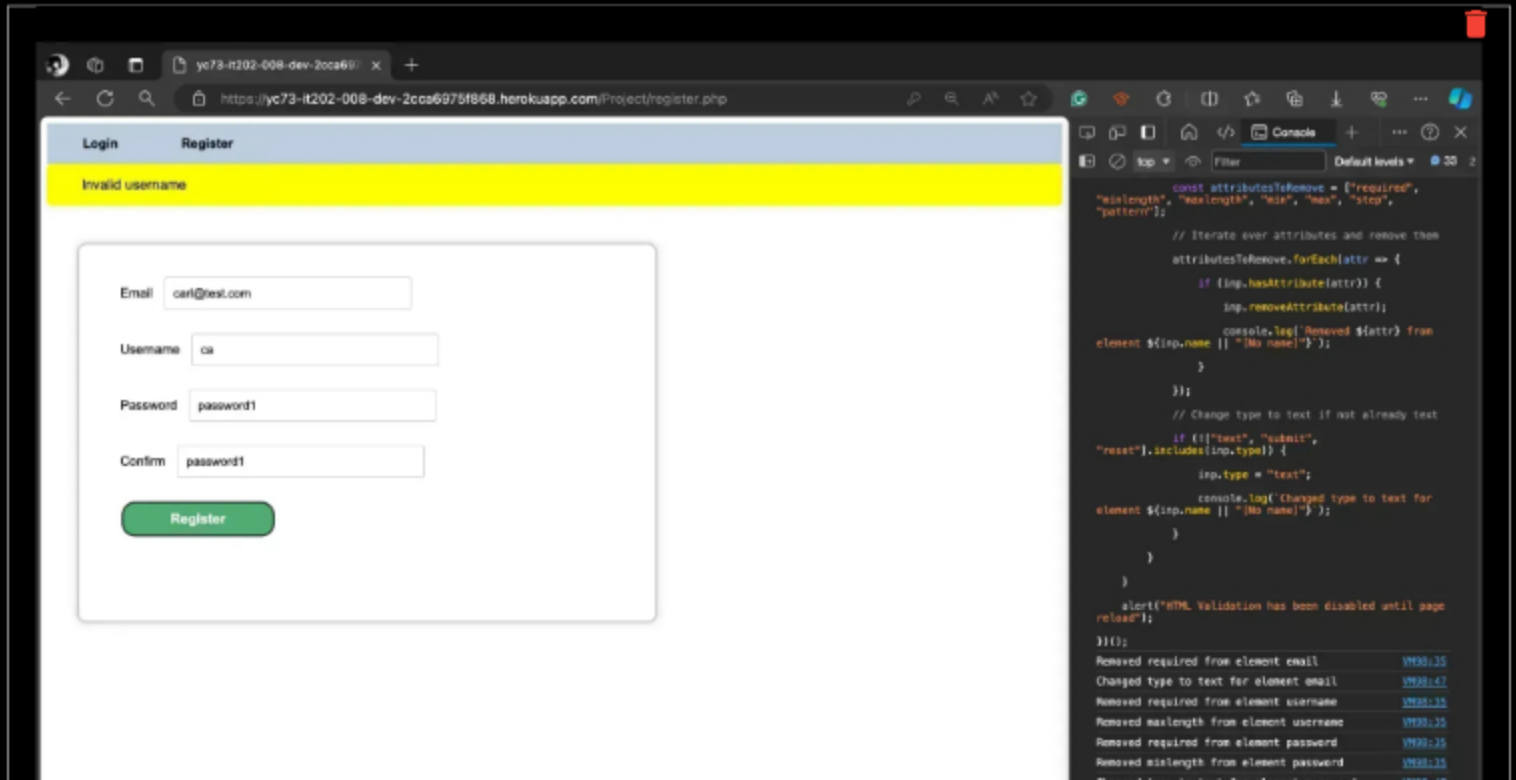
#3 Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required



JavaScript email validation => (checks invalid) email format

Checklist Items (1)

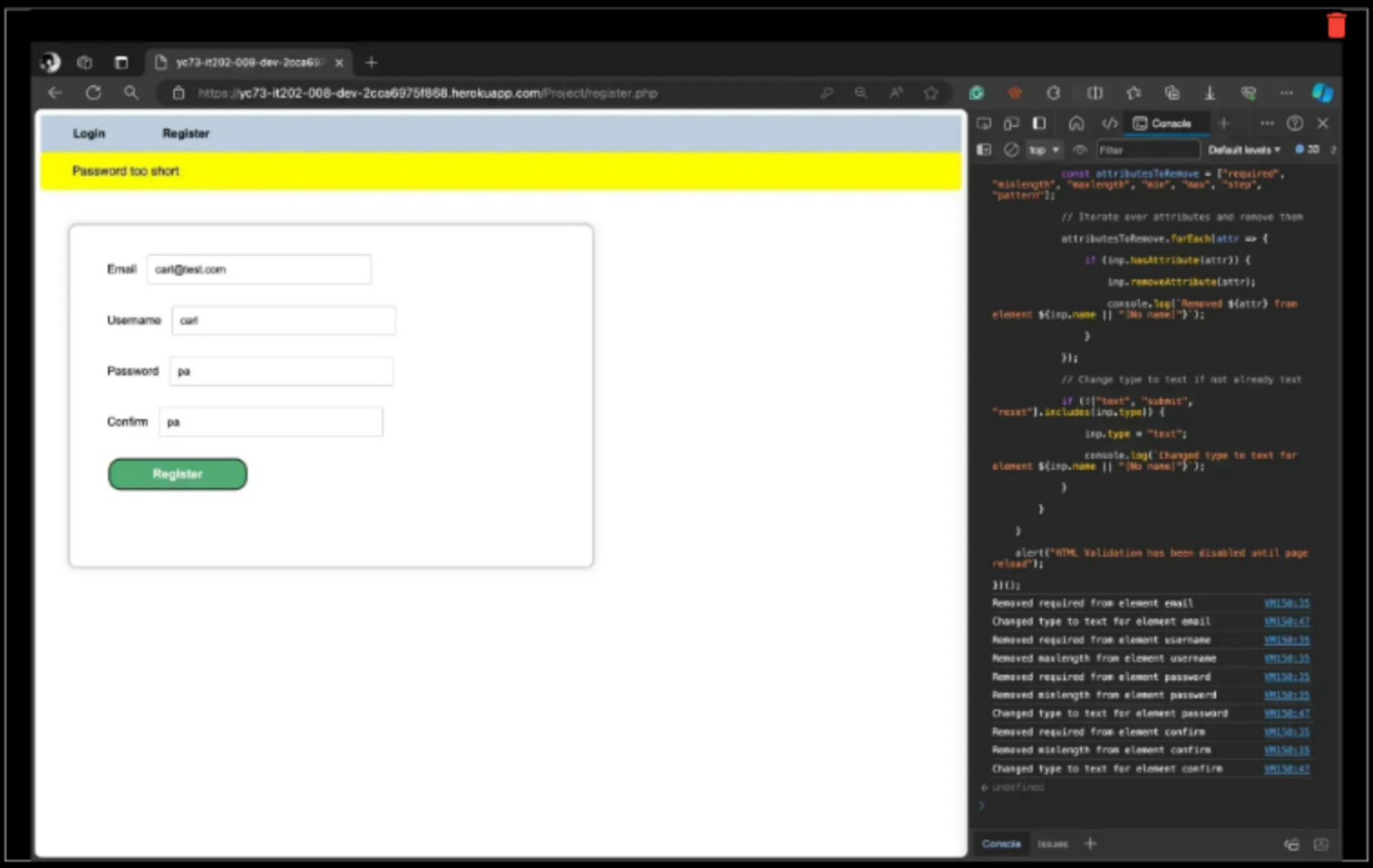
#3 Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required



JavaScript username validation => (checks invalid) username format

Checklist Items (1)

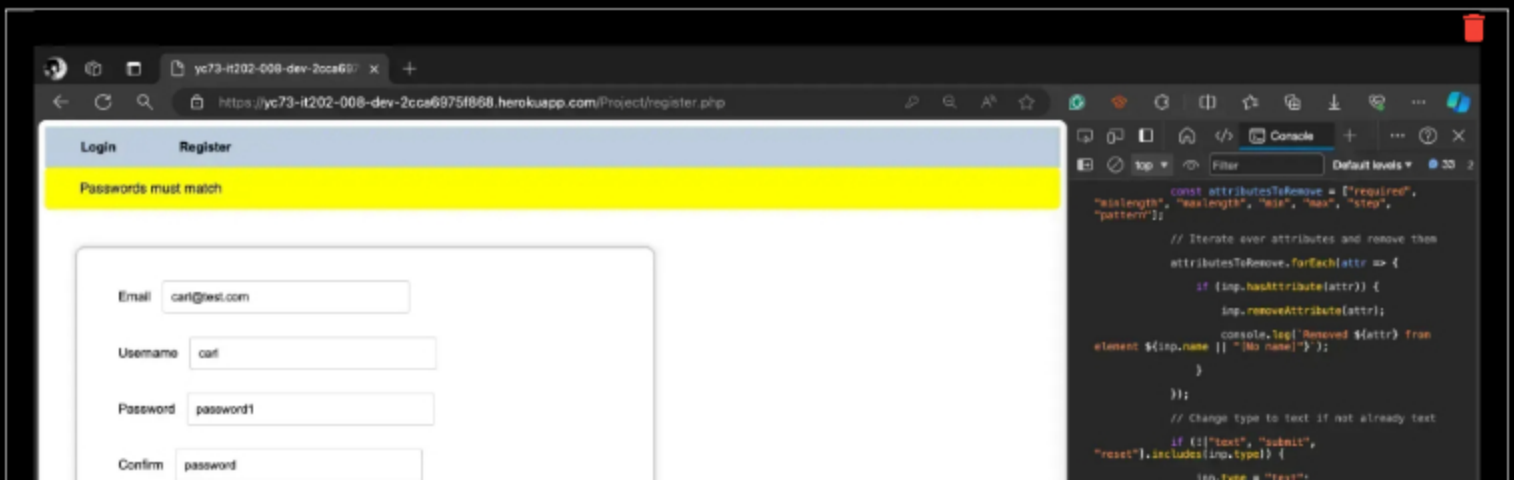
#3 Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required



JavaScript password validation => password length (too short)

Checklist Items (1)

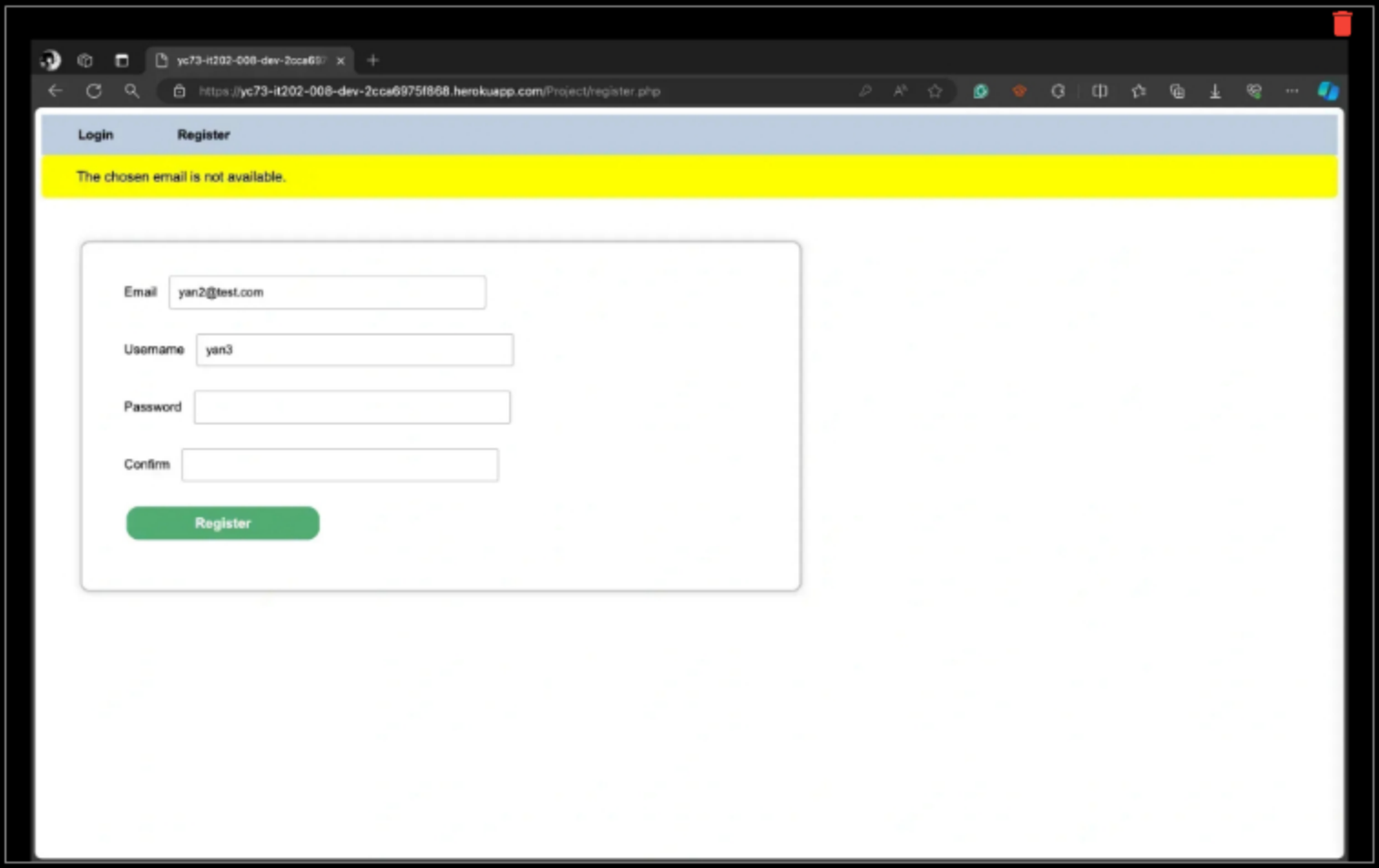
#3 Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required



JavaScript password validation => passwords MUST match

Checklist Items (1)

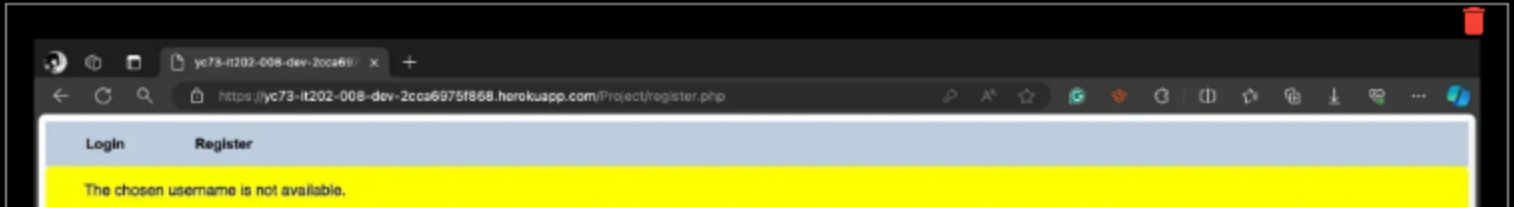
#3 Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required



email already in use message

Checklist Items (1)

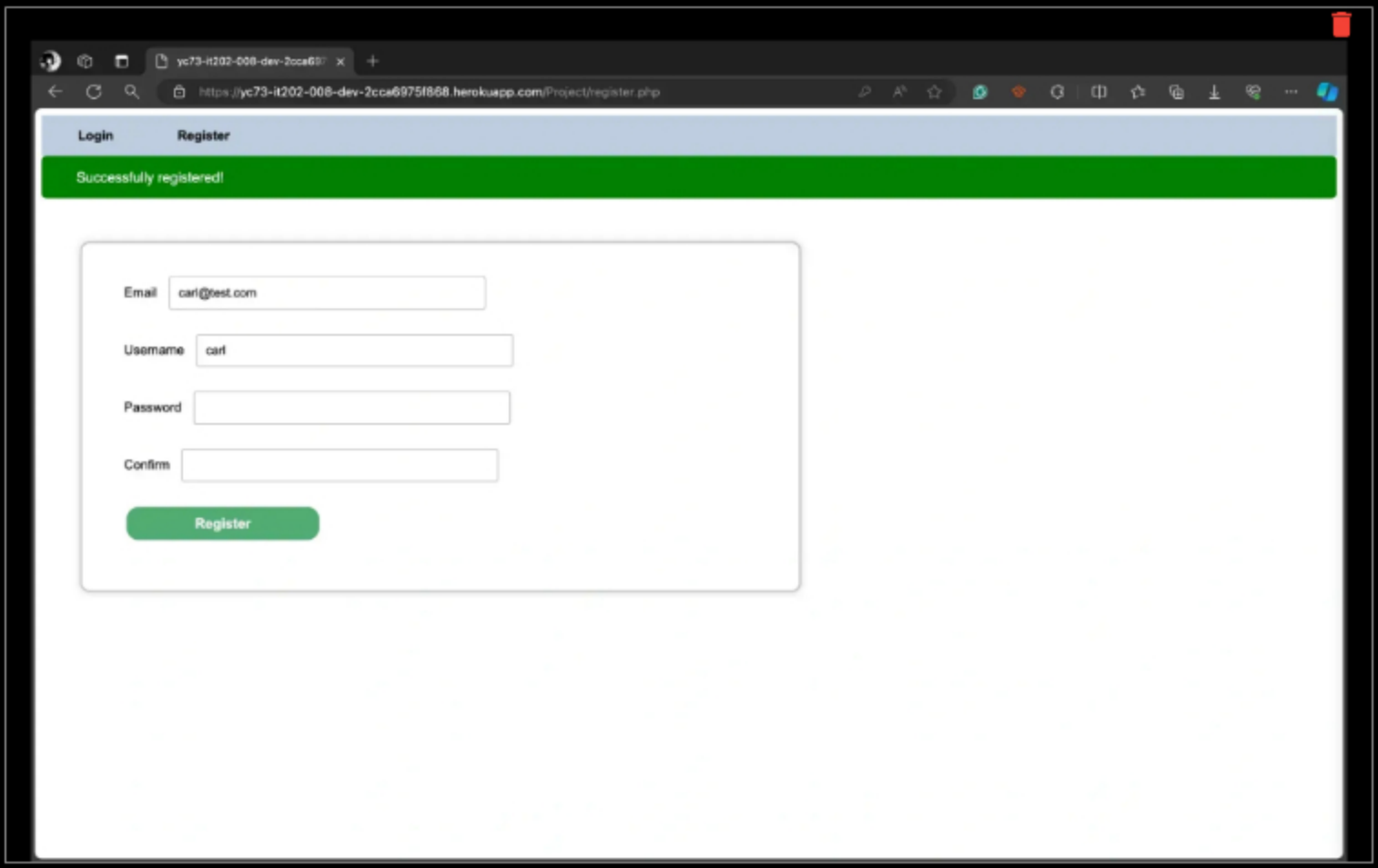
#4 Demonstrate email already in use message (message text doesn't need to be exact, but should be clear)



username already in use message

Checklist Items (1)

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



new account created message

Checklist Items (1)

#6 Demonstrate user-friendly message of new account being created

COLLAPSE

Task #2 - Points: 1

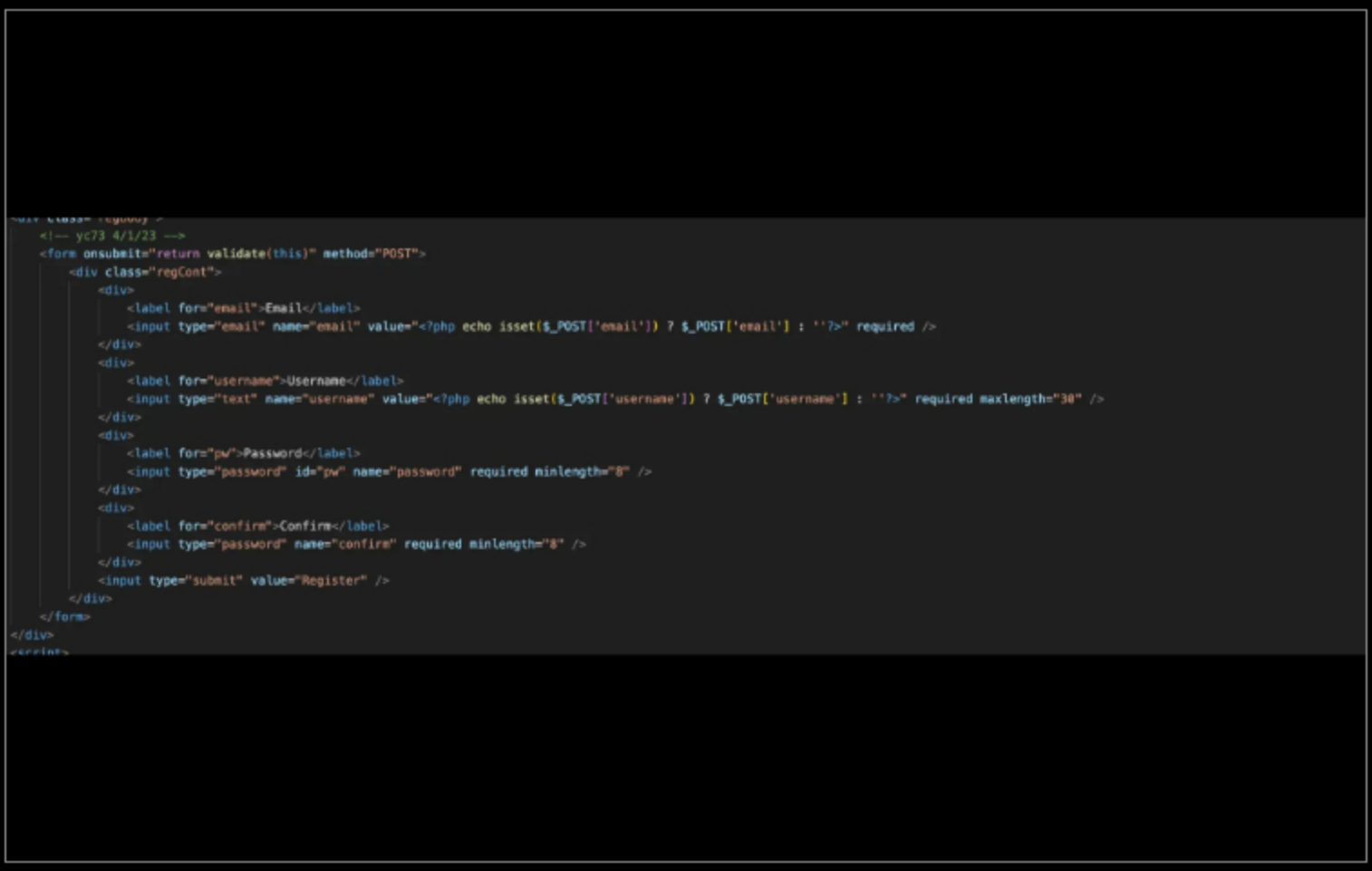
Text: Screenshot of the form code

Details:
Should have appropriate input types for the fiel

Task Screenshots:

Gallery Style: Large View

Small Medium Large



```
    <!-- yc73 4/1/23 -->
    <form onsubmit="return validate(this)" method="POST">
        <div class="regCont">
            <div>
                <label for="email">Email</label>
                <input type="email" name="email" value="<?php echo isset($_POST['email']) ? $_POST['email'] : ''?>" required />
            </div>
            <div>
                <label for="username">Username</label>
                <input type="text" name="username" value="<?php echo isset($_POST['username']) ? $_POST['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" />
        </div>
    </form>
</div>
```

Showing (registration) 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

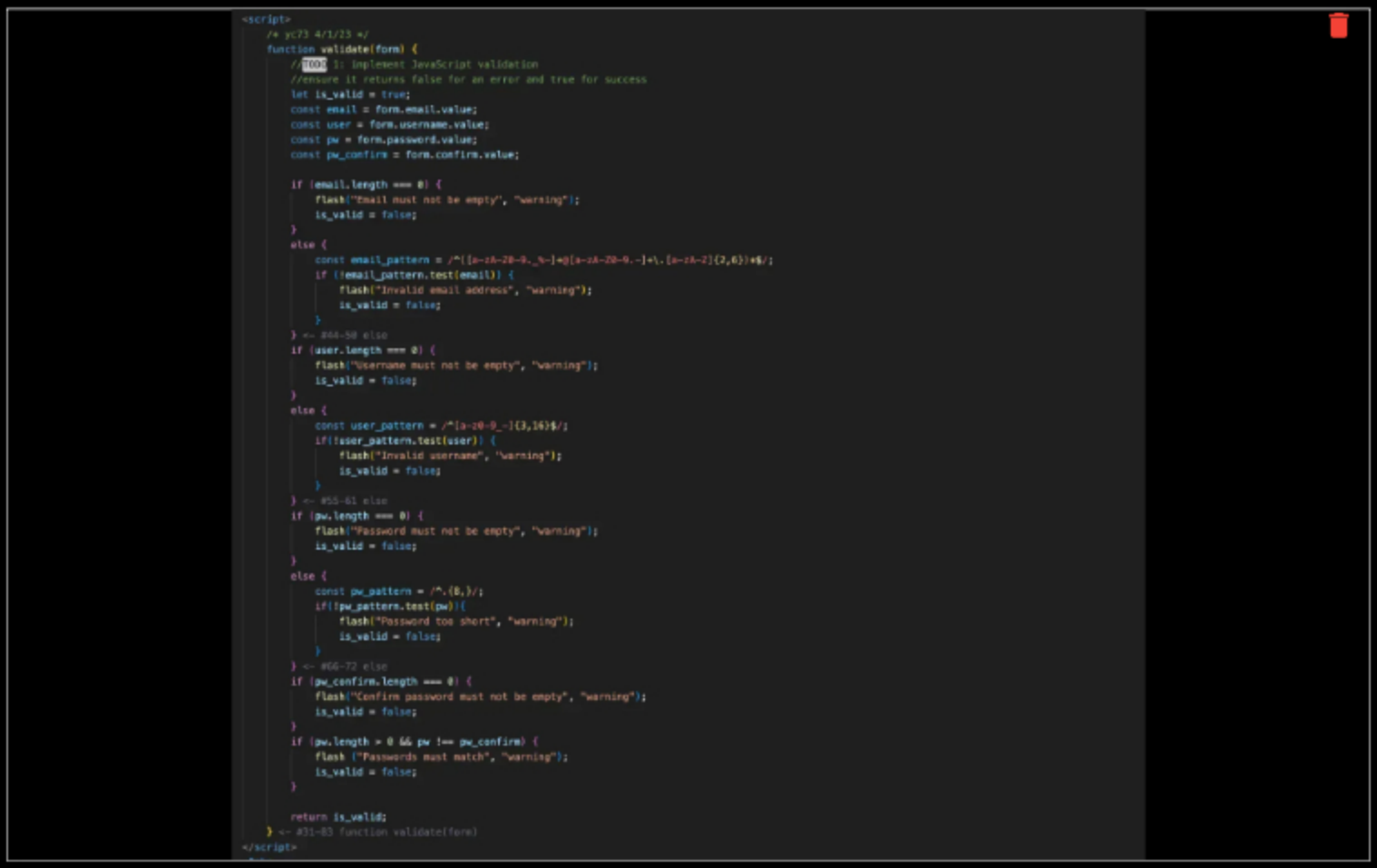
#	Points	Details
#1	1	Show the JavaScript validations (include any extra files related)

#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

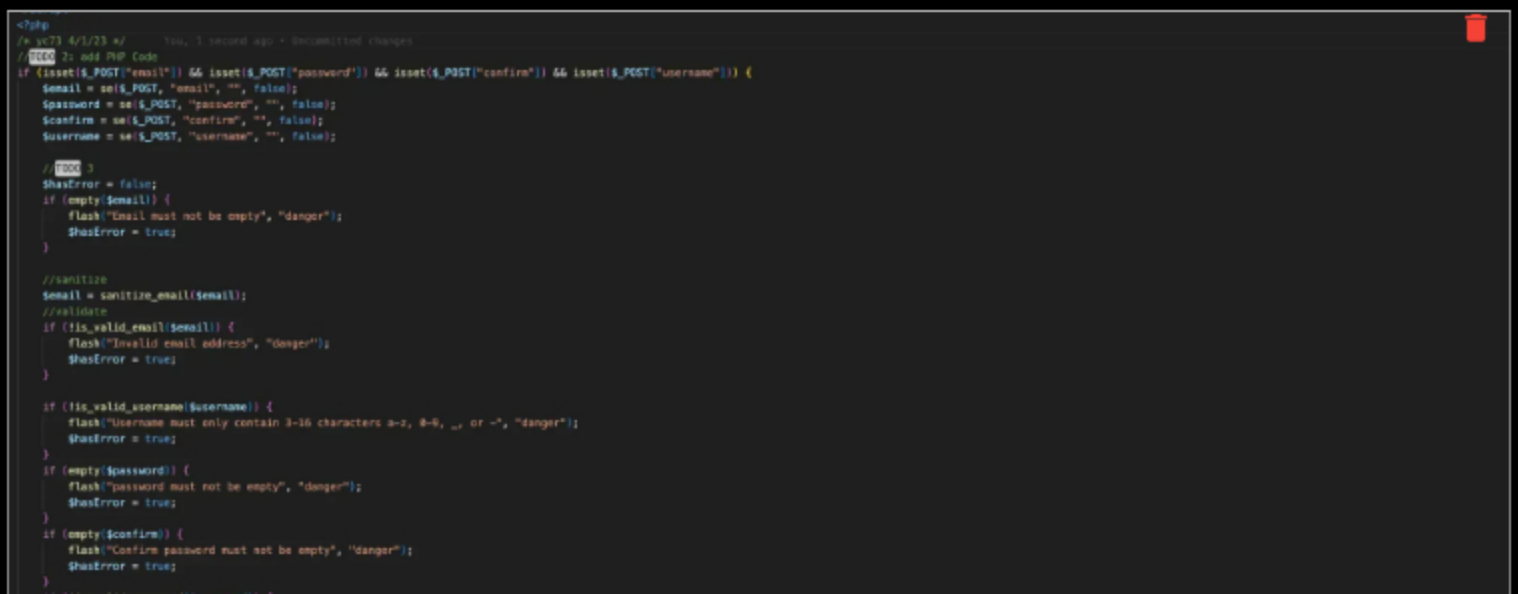


```
<script>
    /* yc73 4/1/23 */
    function validate(form) {
        //TODO 1: implement JavaScript validation
        //ensure it returns false for an error and true for success
        let is_valid = true;
        const email = form.email.value;
        const user = form.username.value;
        const pw = form.password.value;
        const pw_confirm = form.confirm.value;

        if (email.length === 0) {
            flash("Email must not be empty", "warning");
            is_valid = false;
        }
        else {
            const email_pattern = /^([a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6})+$/;
            if (!email_pattern.test(email)) {
                flash("Invalid email address", "warning");
                is_valid = false;
            }
        } <- #44-58 else
        if (user.length === 0) {
            flash("Username must not be empty", "warning");
            is_valid = false;
        }
        else {
            const user_pattern = /^[a-z0-9_-]{3,16}$/;
            if(!user_pattern.test(user)) {
                flash("Invalid username", "warning");
                is_valid = false;
            }
        } <- #55-61 else
        if (pw.length === 0) {
            flash("Password must not be empty", "warning");
            is_valid = false;
        }
        else {
            const pw_pattern = /^.{8,}/;
            if(!pw_pattern.test(pw)){
                flash("Password too short", "warning");
                is_valid = false;
            }
        } <- #66-72 else
        if (pw_confirm.length === 0) {
            flash("Confirm password must not be empty", "warning");
            is_valid = false;
        }
        if (pw.length > 0 && pw !== pw_confirm) {
            flash ("Passwords must match", "warning");
            is_valid = false;
        }

        return is_valid;
    } <- #31-83 function validate(form)
</script>
```

Showing JavaScript validations

Checklist Items (1)

#1 Show the JavaScript validations (include any extra files related)



```
<?php
/* yc73 4/1/23 */      You, 1 second ago • Uncommitted changes
//TODO 2: add PHP Code
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;
    }

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

```php
    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);
        }
    } <- #129-140 if (!$hasError)
} <- #88-141 if (isset($_POST["email"]) && isset($_POST["password"]) && is...
?>
```

Showing PHP validations

Checklist Items (1)

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

COLLAPSE

Task #4 - Points: 1

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

Checklist

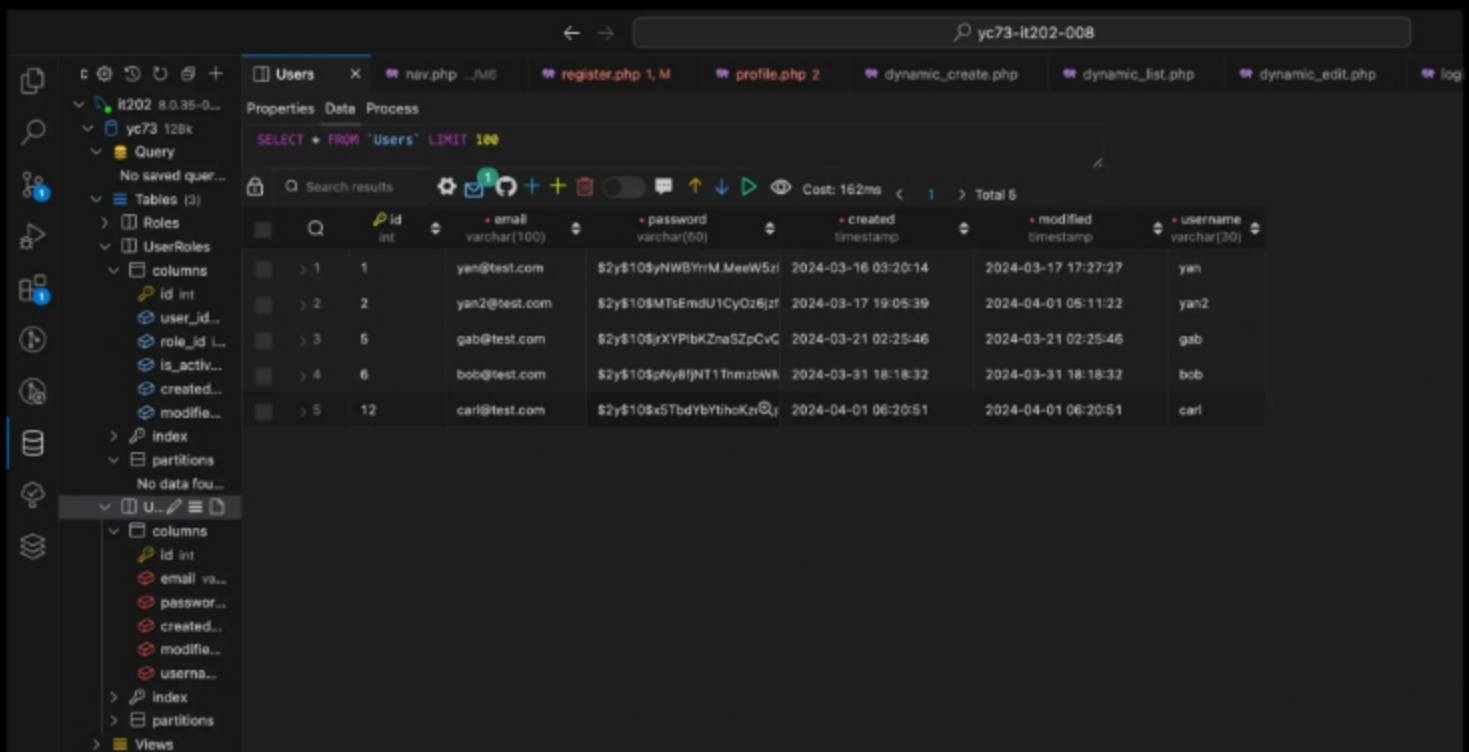
*The checkboxes are for your own tracking

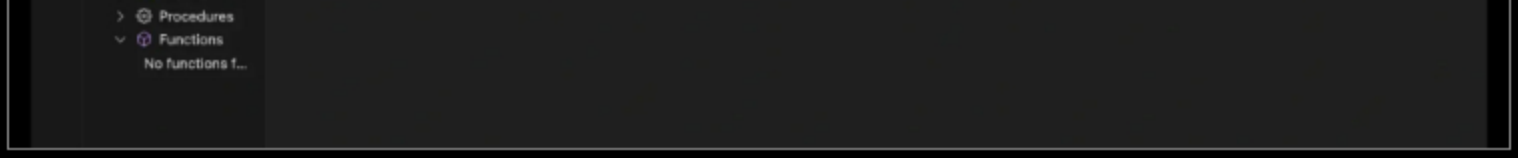
#	Points	Details
#1	1	Password should be hashed
#2	1	Should have email, password, username (unique), created, modified, and id fields
#3	1	Ensure left panel or database name is present (should contain your ucid)

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Showing "Users" table (with id, email, username (unique), password, created, modified)

Checklist Items (1)

#2 Should have email, password, username (unique), created, modified, and id fields

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:

- Upon loading the registration page, an (HTML) form is displayed to allow users to enter their email, username, password, and confirm the password. When the user submits that registration form, it will trigger the "onsubmit" attribute and "validate" function. The validation function is the client-side JavaScript validation which checks the email, username, password, and confirm password fields.
  - First, it grabs the values (of what the user entered for the fields) using the "form" object (ex: "form.email.value", "form.username.value", "form.password.value", "form.confirm.value"). Then, it stores the values into corresponding variables (email, user, pw, pw_confirm).
  - To check if the fields are empty, it looks at the length of the variable with ".length===0".
  - To check the formatting, regex is used. For email, the regex looks for alphanumerics at the start, followed by an @ symbol, another set of alphanumerics, followed by a dot (.), and ends with 2-6 alphabetic characters. For username, the regex looks for alphanumerics, possible _- characters, and a length of 3-16 characters.
  - To check the length of the password, it uses regex to look for a length of 8 characters or more.
  - To check if password and confirm password match, they are compared with strict inequality (!==).
- The "onsubmit" takes a boolean value, so if any of these validations returns a false, then warning messages are displayed to the user, and the form is not submitted. However, if none of the validations fail, it returns true and sends the data as POST.
- So, if the client-side validation is valid (true), the form data is sent to the server-side, and it goes through PHP validations.
  - First, it checks if the POST array contains email, password, confirm password, and username. If it does, saferecho uses $_POST to retrieve the data, then sets it to variables and begins validating the data.
  - To check if the email and passwords are empty it uses an empty() function.
  - To check if the email format is valid, it must first sanitize it with "sanitize_email()" to remove (filter/trim) illegal characters. Then, it can be validated with "is_valid_email()" which uses FILTER_VALIDATE_EMAIL to check for invalid formatting/characters - functions are defined in "sanitizers.php".
  - To check if the username format is valid, it uses "is_valid_username()" (in "sanitizers.php"), which checks with regex if it has alphanumerics, possibly contains _-, and is between 3-16 characters.
  - To check if the password has a valid length, it uses "is_valid_password()" (in "sanitizers.php"), which

- To check if the password has a valid length, it uses "is_valid_password()" (in "sanitizers.php"), which uses "strlen" to see if the length is equal to or greater than 8.
- To check if password and confirm password match, they are compared with strict inequality (!==).

- If any fields fail the validations, it returns true for "hasError" and flashes error messages to the user. However, if all the fields pass the validations, it returns false for "hasError" (meaning there are no errors), the password is hashed using the "password_hash()", and the user's data is inserted into the "Users" table in the database (using a prepared statement). The SQL statement "INSERT INTO Users" is executed and inserts the user's email, password, and username into the corresponding columns. When it is successfully inserted, the user receives a successfully registered message.

COLLAPSE

Task #6 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/yaneliii/yc73-it202-008/pull/17

URL #2
https://github.com/yaneliii/yc73-it202-008/pull/27

URL #3
https://github.com/yaneliii/yc73-it202-008/pull/32

User Login (2 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of form on website page

Checklist

*The checkboxes are for your own tracking

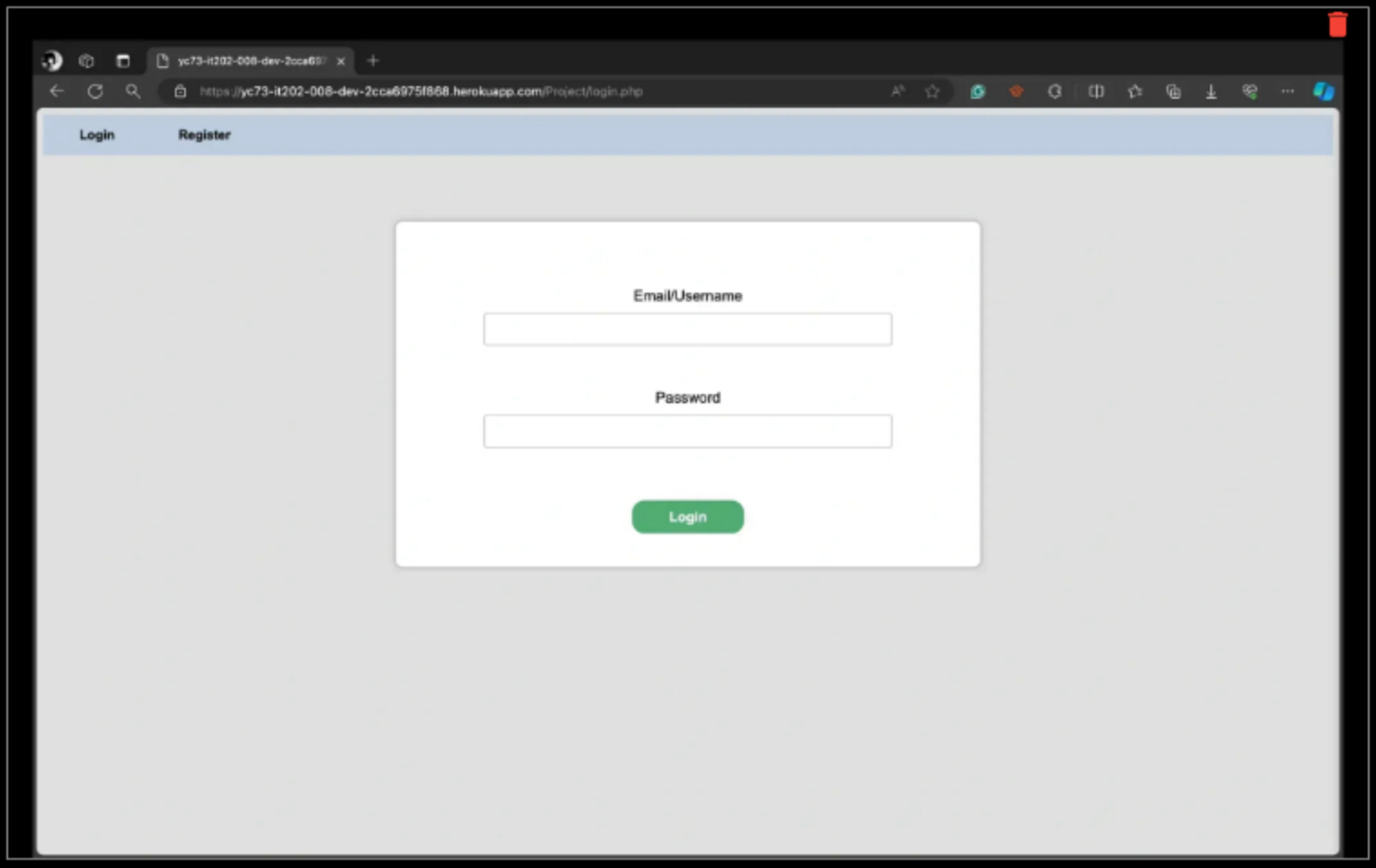
#	Points	Details
#1	1	Heroku dev url should be present in the address bar
#2	1	Should have thoughtful CSS applied
#3	1	Include correct data where username is used to login
#4	1	Include correct data where email is used to login
#5	1	Show success login message
#6	1	Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)
#7	1	Demonstrate user-friendly message of when an account doesn't exist
#8	1	Demonstrate user-friendly message of when password doesn't match what's in the DB

#9	1	Demonstrate successful login message and the destination page (i.e., home or some landing/dashboard page)
#10	1	Demonstrate session data being set (captured from server logs)

Task Screenshots:

Gallery Style: Large View

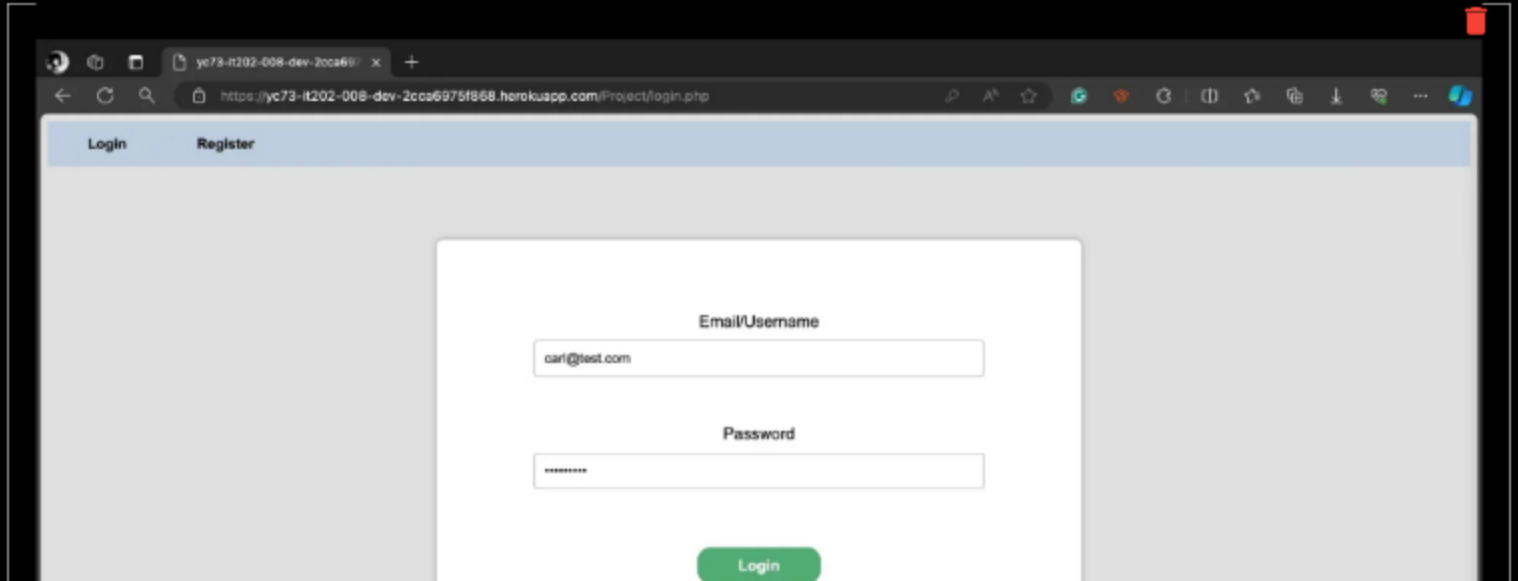
Small Medium Large

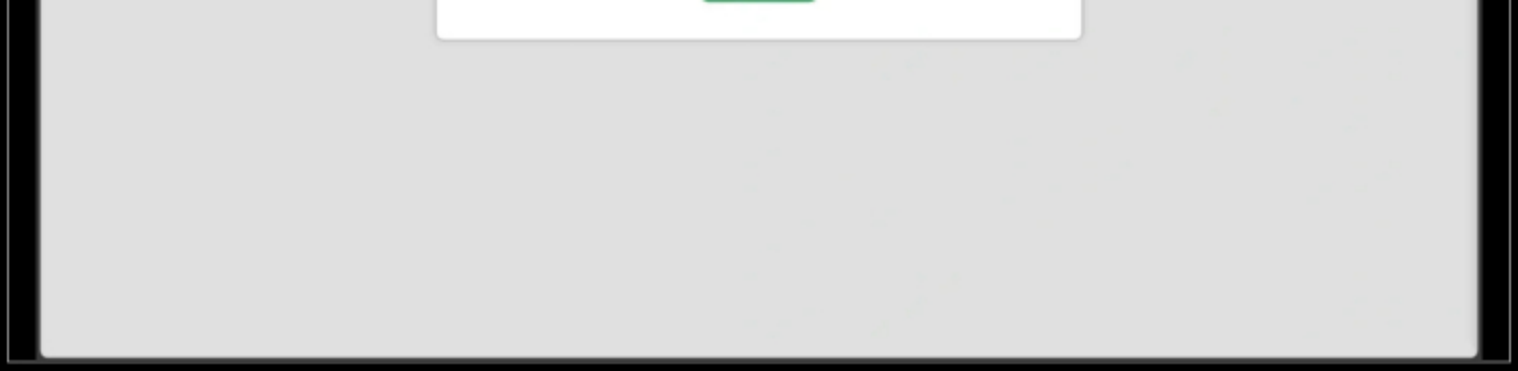


Showing (blank) login page with CSS

Checklist Items (1)

#2 Should have thoughtful CSS applied

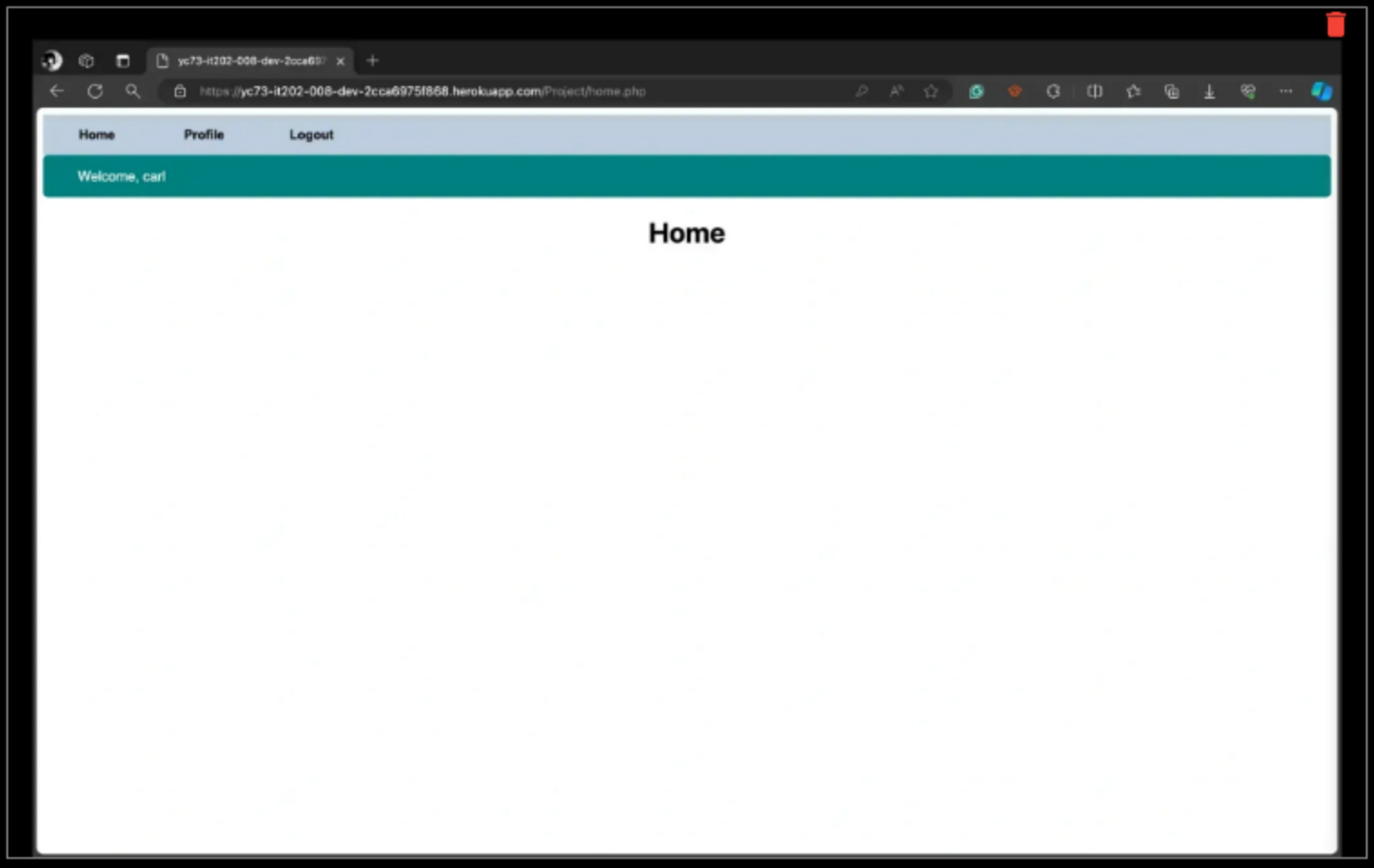




Showing correct EMAIL data to login

Checklist Items (1)

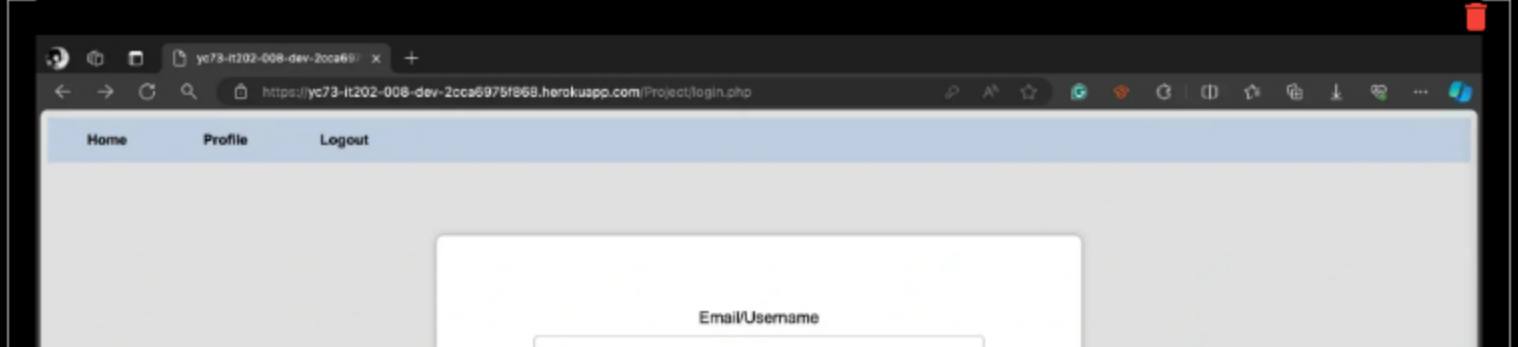
#4 Include correct data where email is used to login

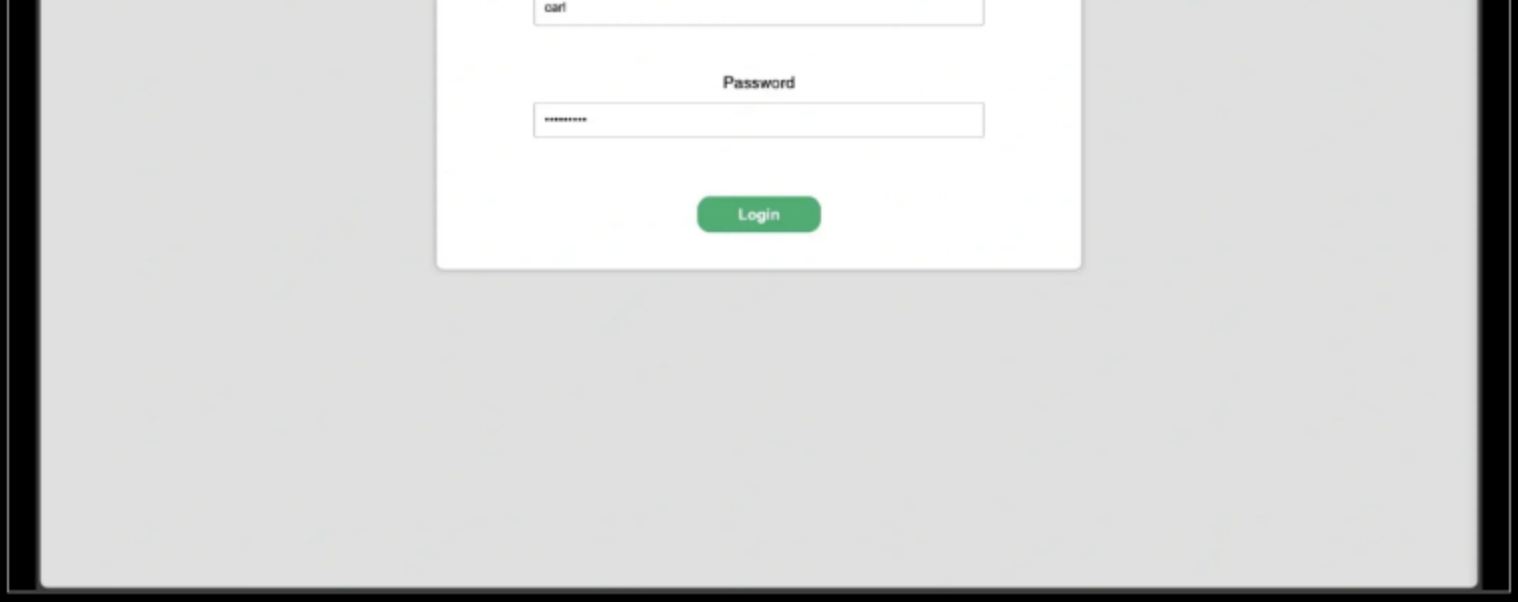


Showing success login message (with correct EMAIL)

Checklist Items (1)

#5 Show success login message

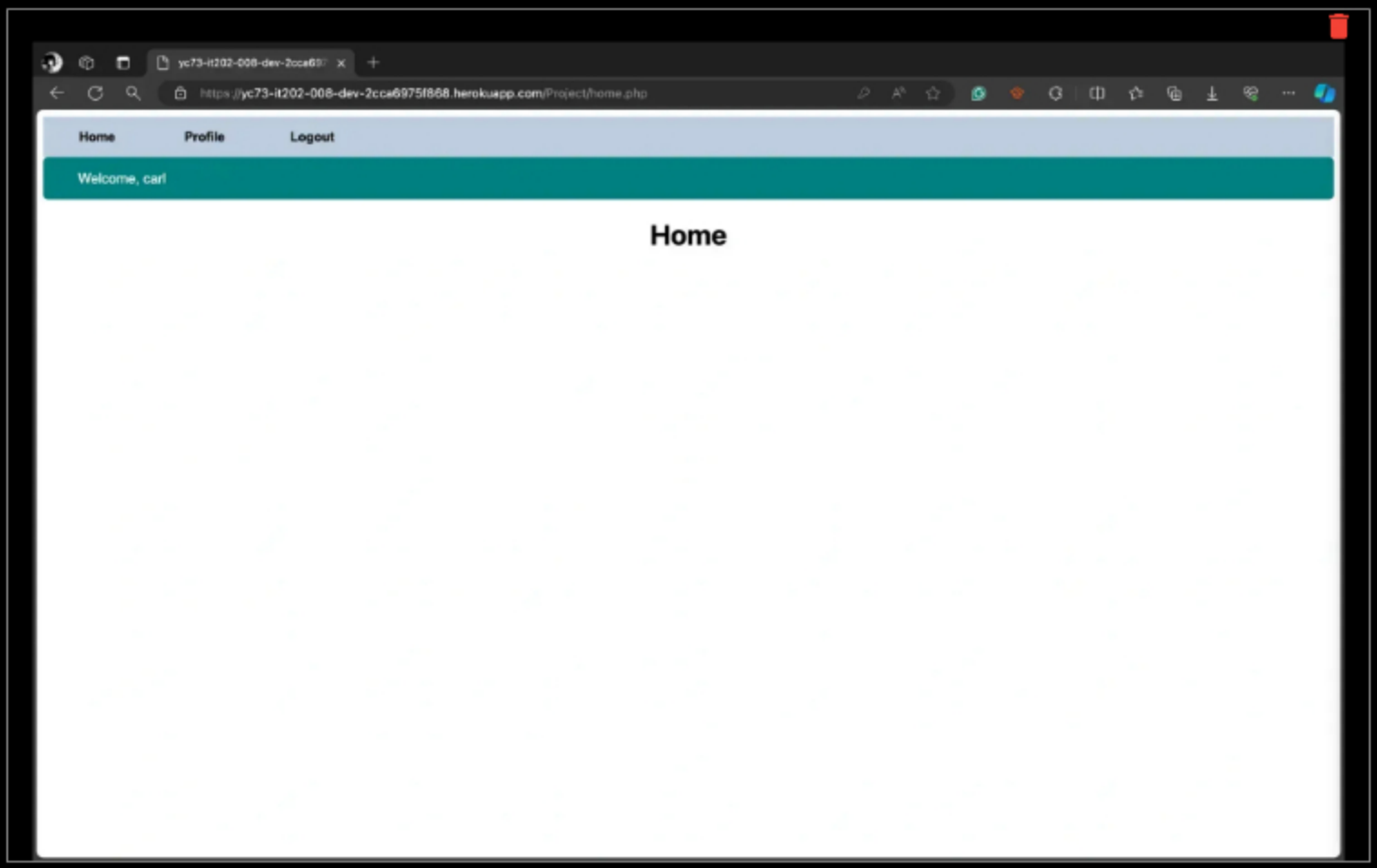




Showing correct USERNAME data to login

Checklist Items (1)

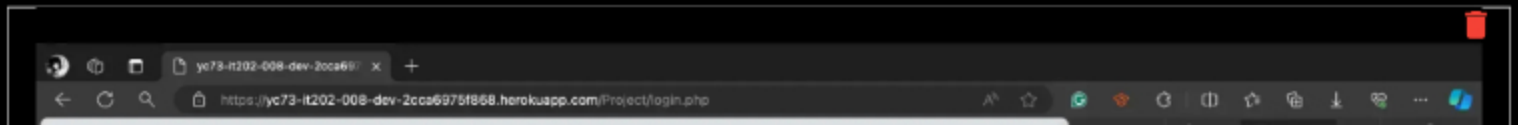
#3 Include correct data where username is used to login

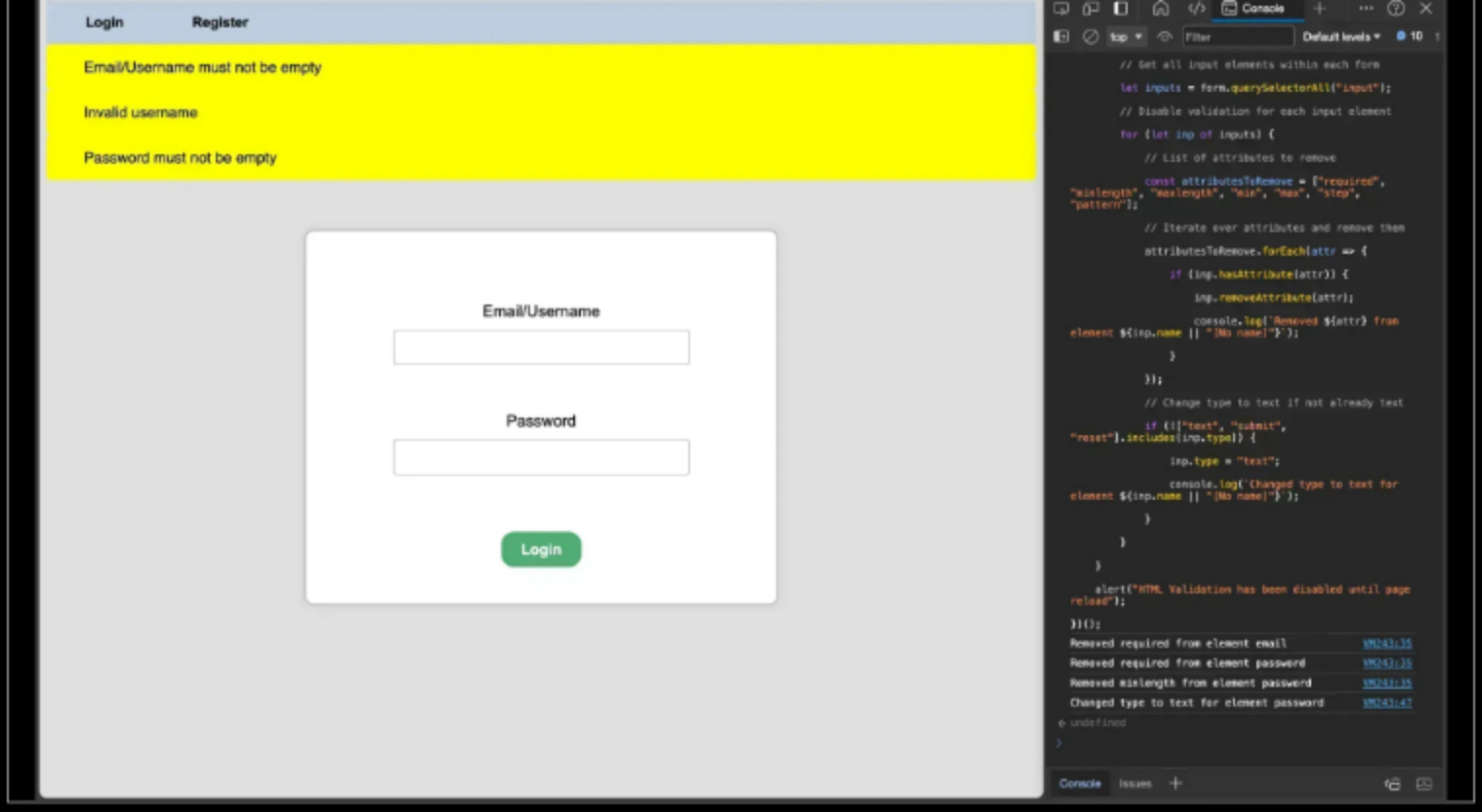


Showing success login message (with correct USERNAME) and the destination page (HOME page)

Checklist Items (1)

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

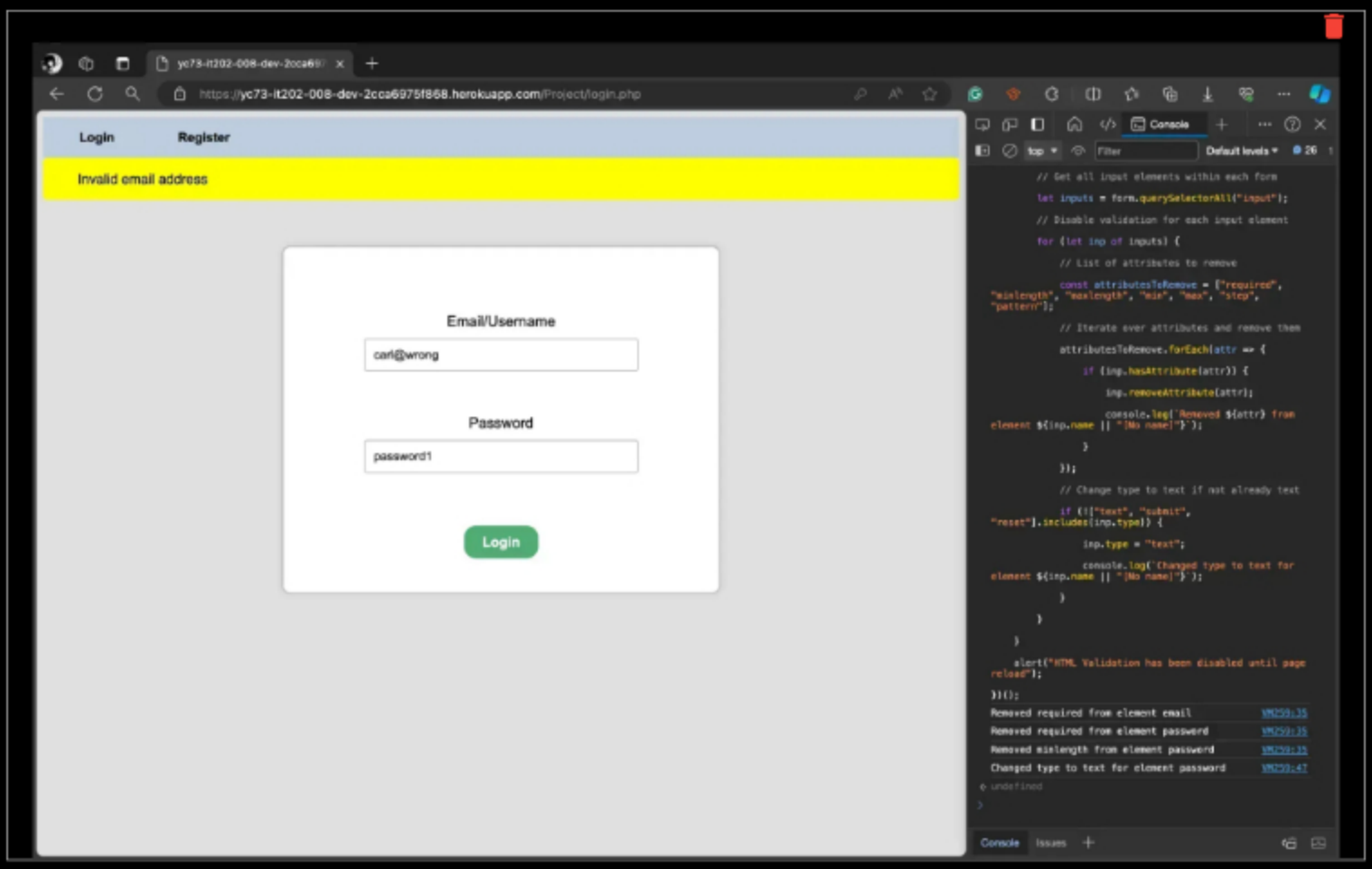




JavaScript email, username, and password validation => for empty fields (p.s. I made my JS validations "warnings")

Checklist Items (1)

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

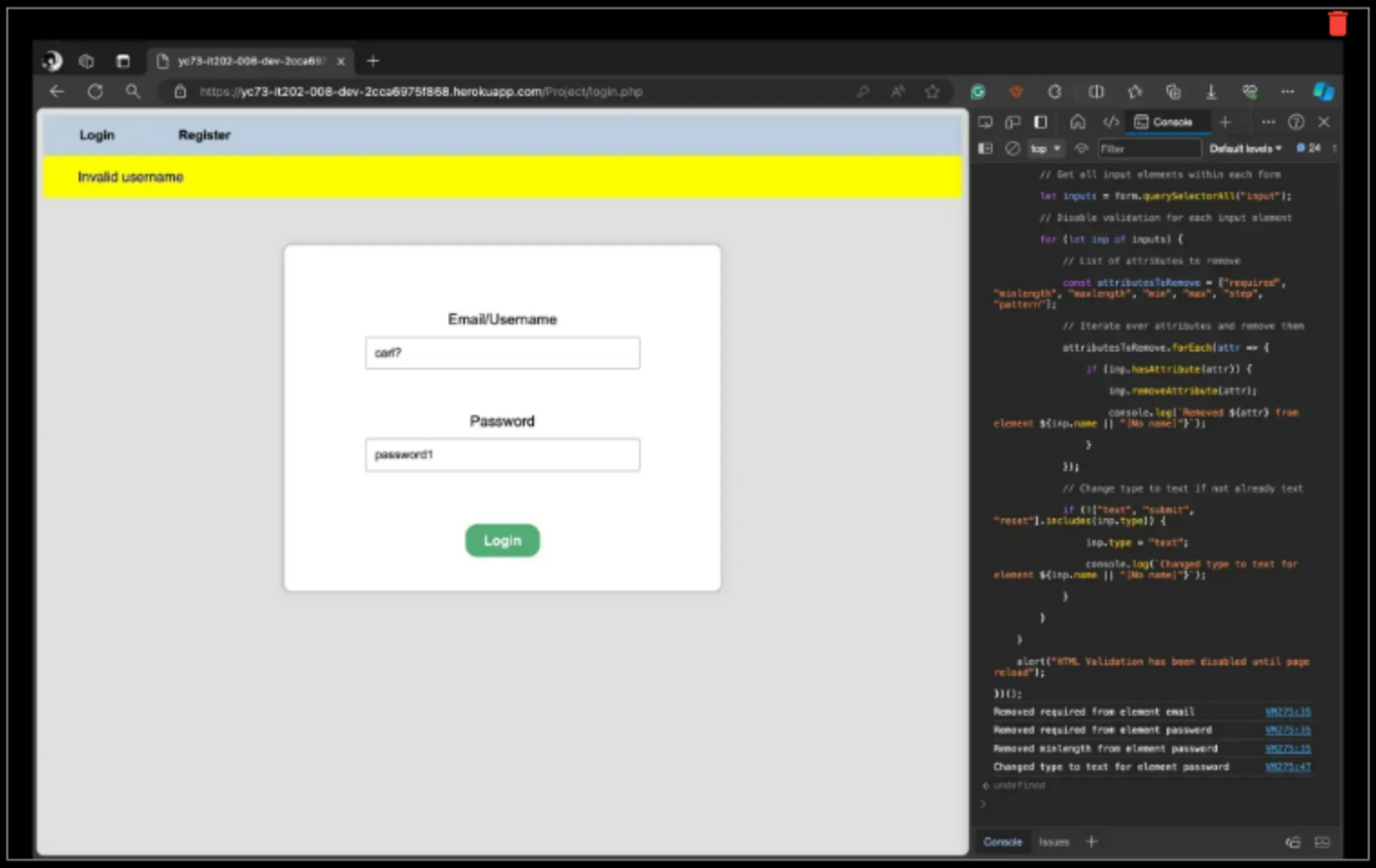


JavaScript email validation => (checks for invalid) email format

Checklist Items (1)

Checklist Items (1)

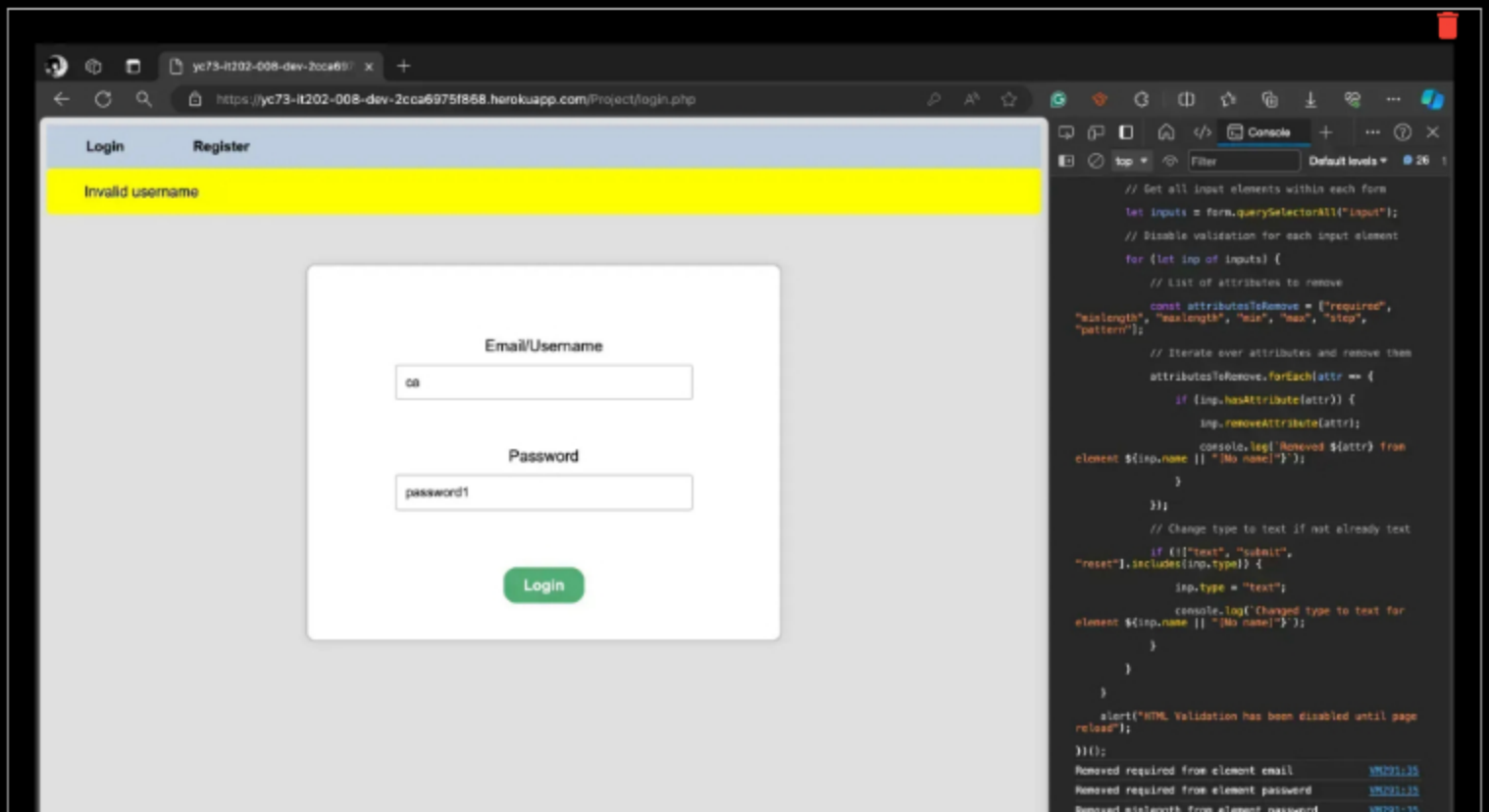
#6 Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)



JavaScript username validation => (checks for invalid) username format (with invalid symbol)

Checklist Items (1)

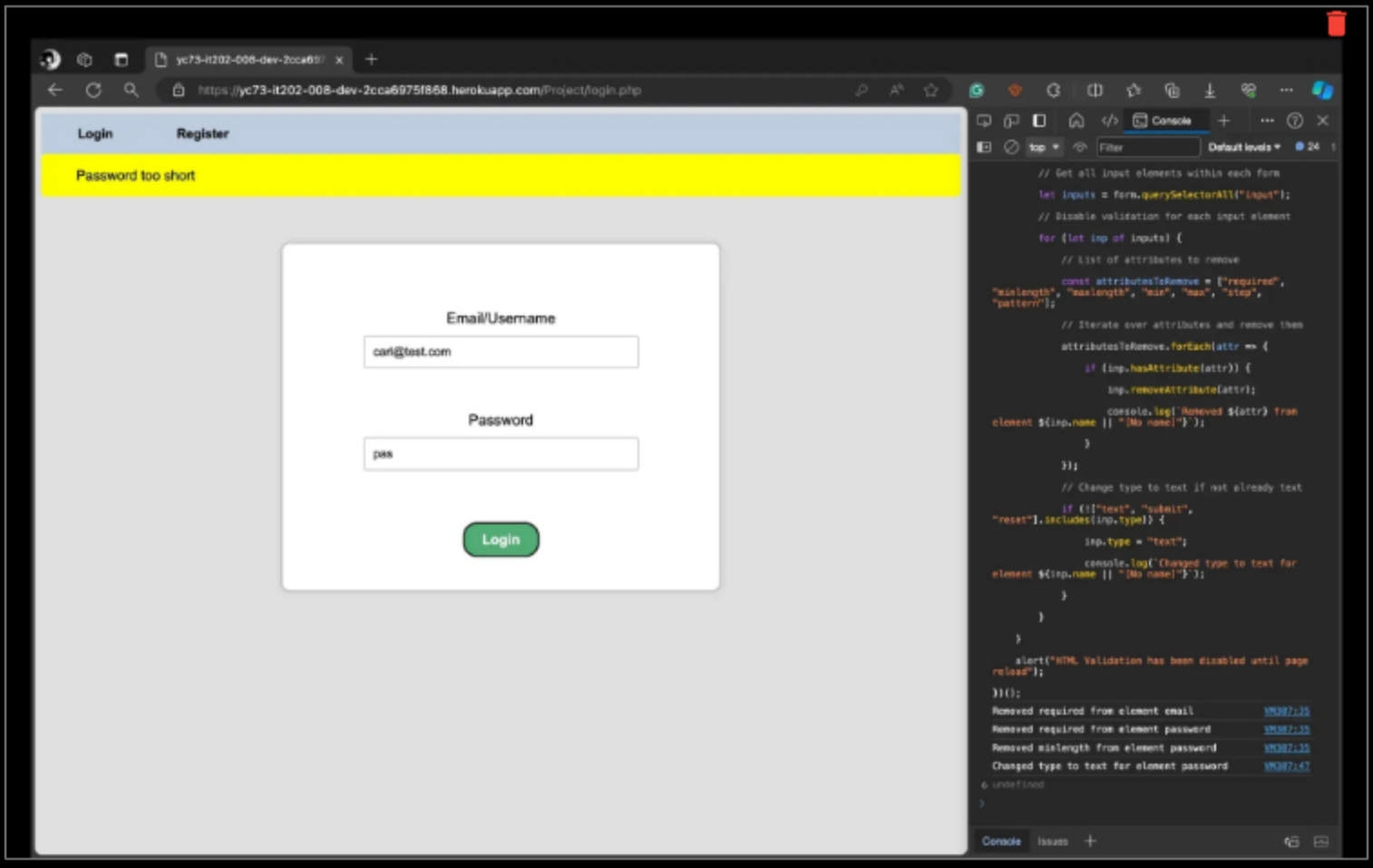
#6 Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)



JavaScript username validation => (checks for invalid) username format (with invalid length)

Checklist Items (1)

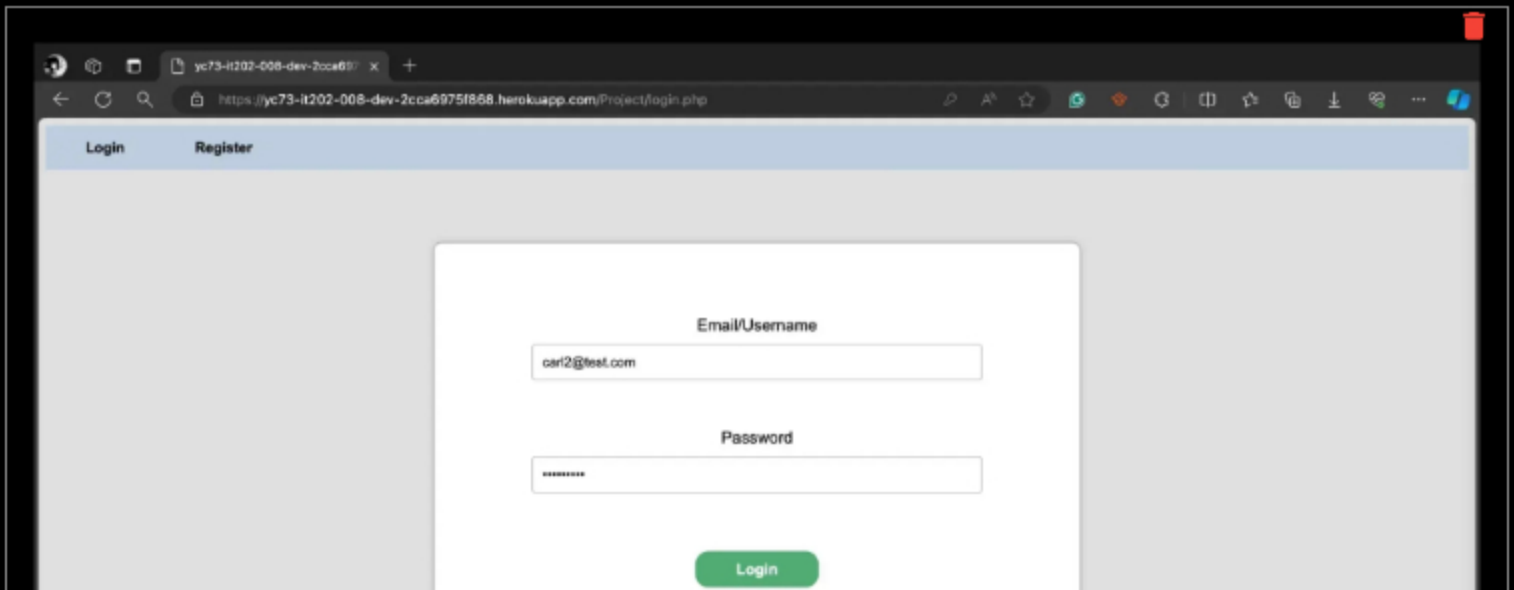
#6 Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)



JavaScript password validation => password length (too short)

Checklist Items (1)

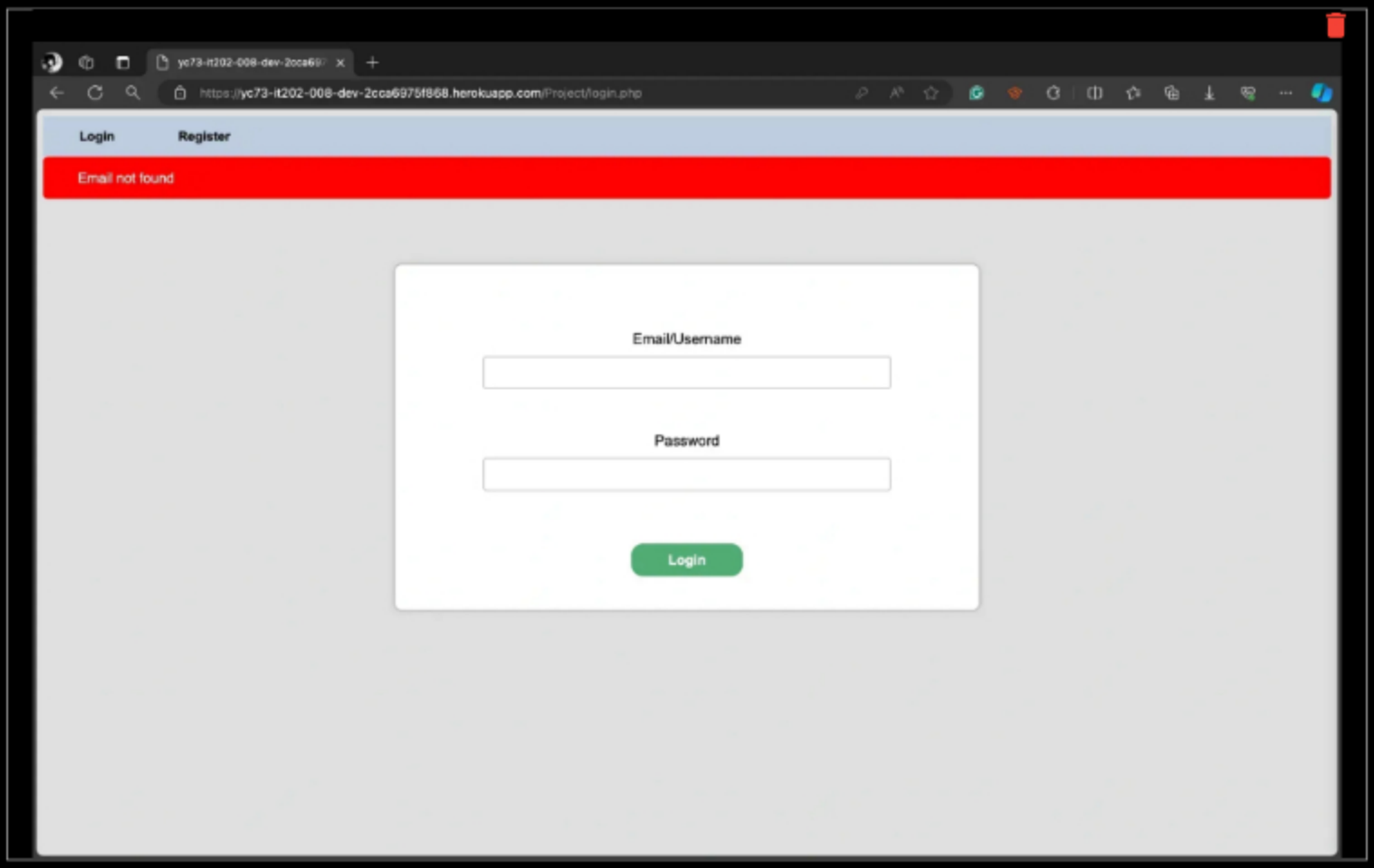
#6 Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)



Showing email that does NOT exist

Checklist Items (1)

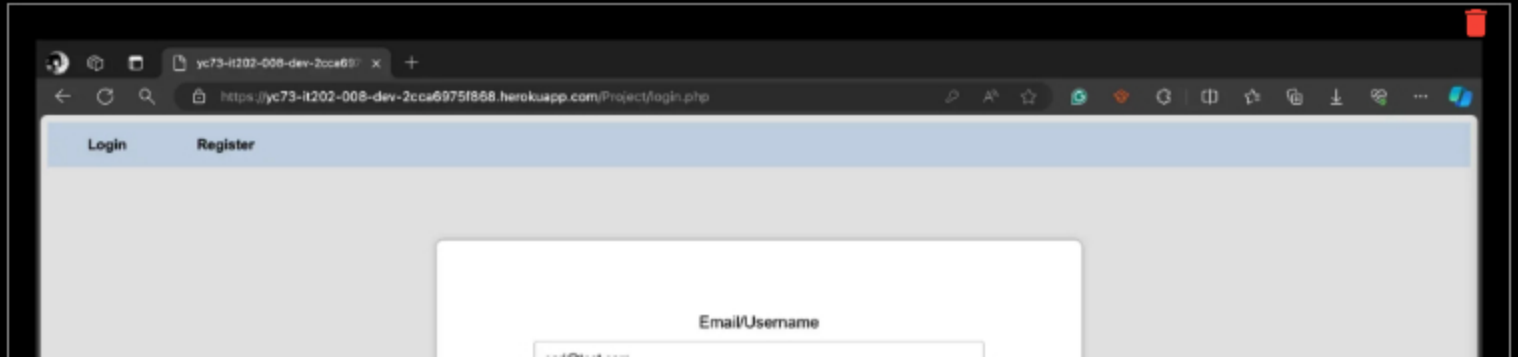
#7 Demonstrate user-friendly message of when an account doesn't exist

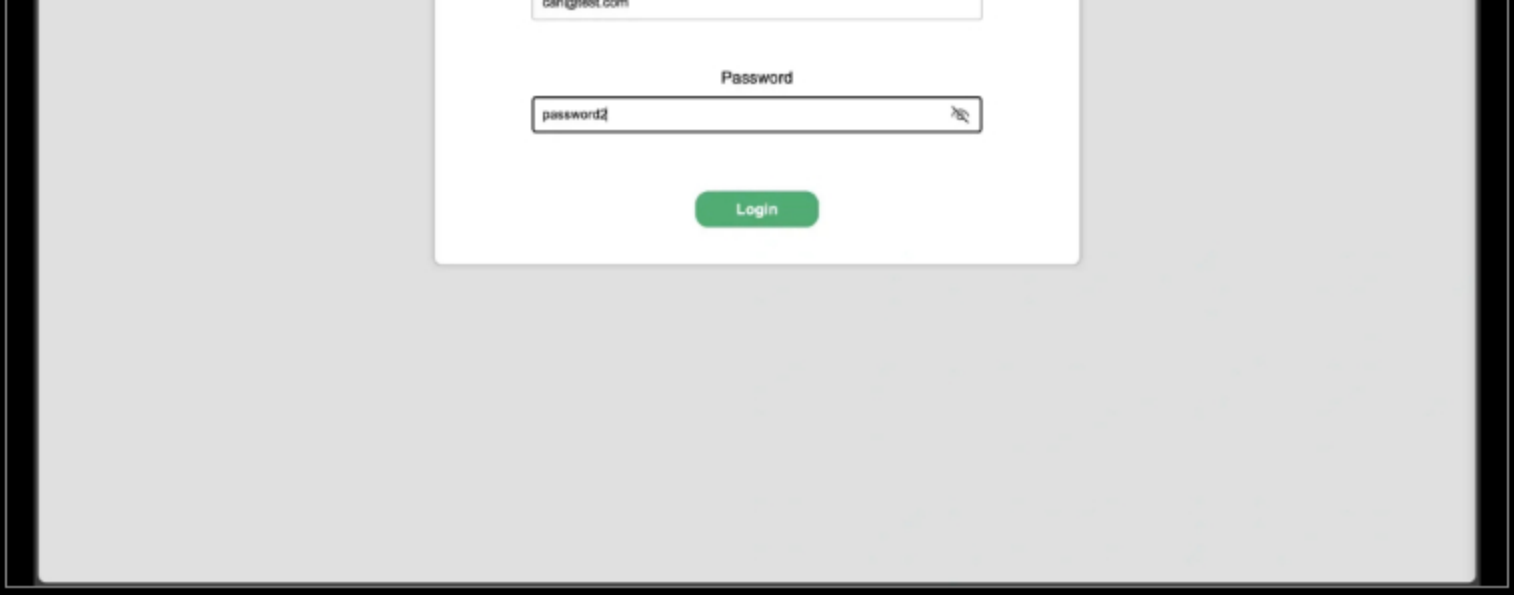


Showing message when an account does NOT exist

Checklist Items (1)

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

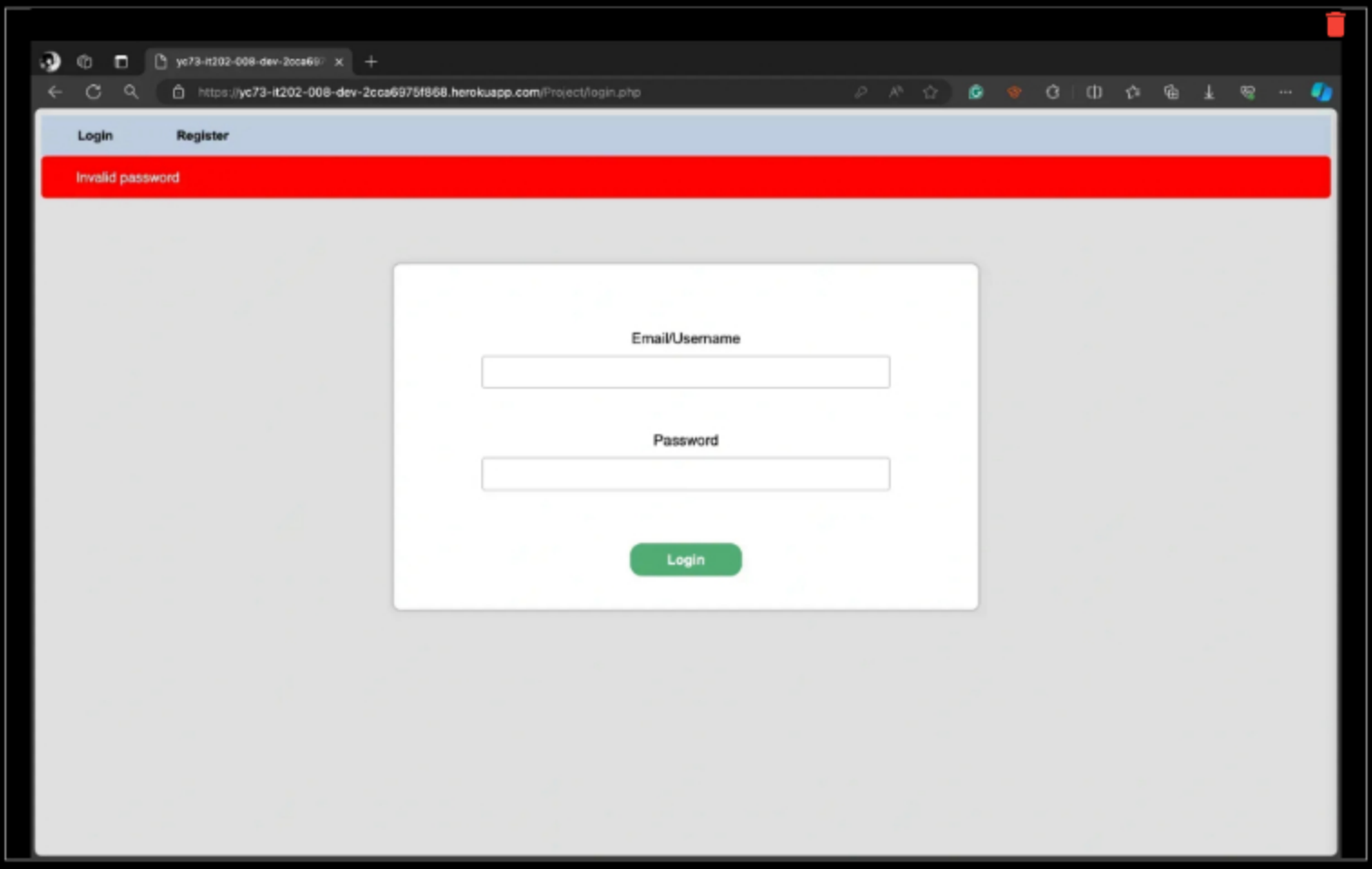




Showing INCORRECT password

Checklist Items (1)

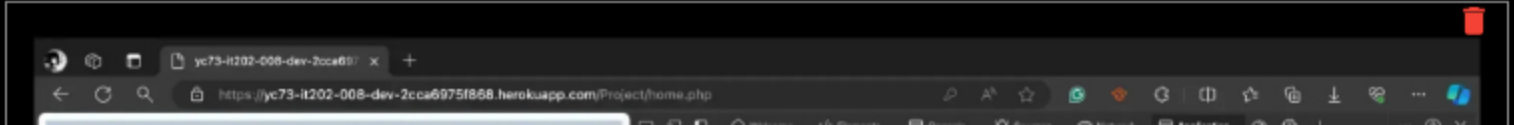
#8 Demonstrate user-friendly message of when password doesn't match what's in the DB

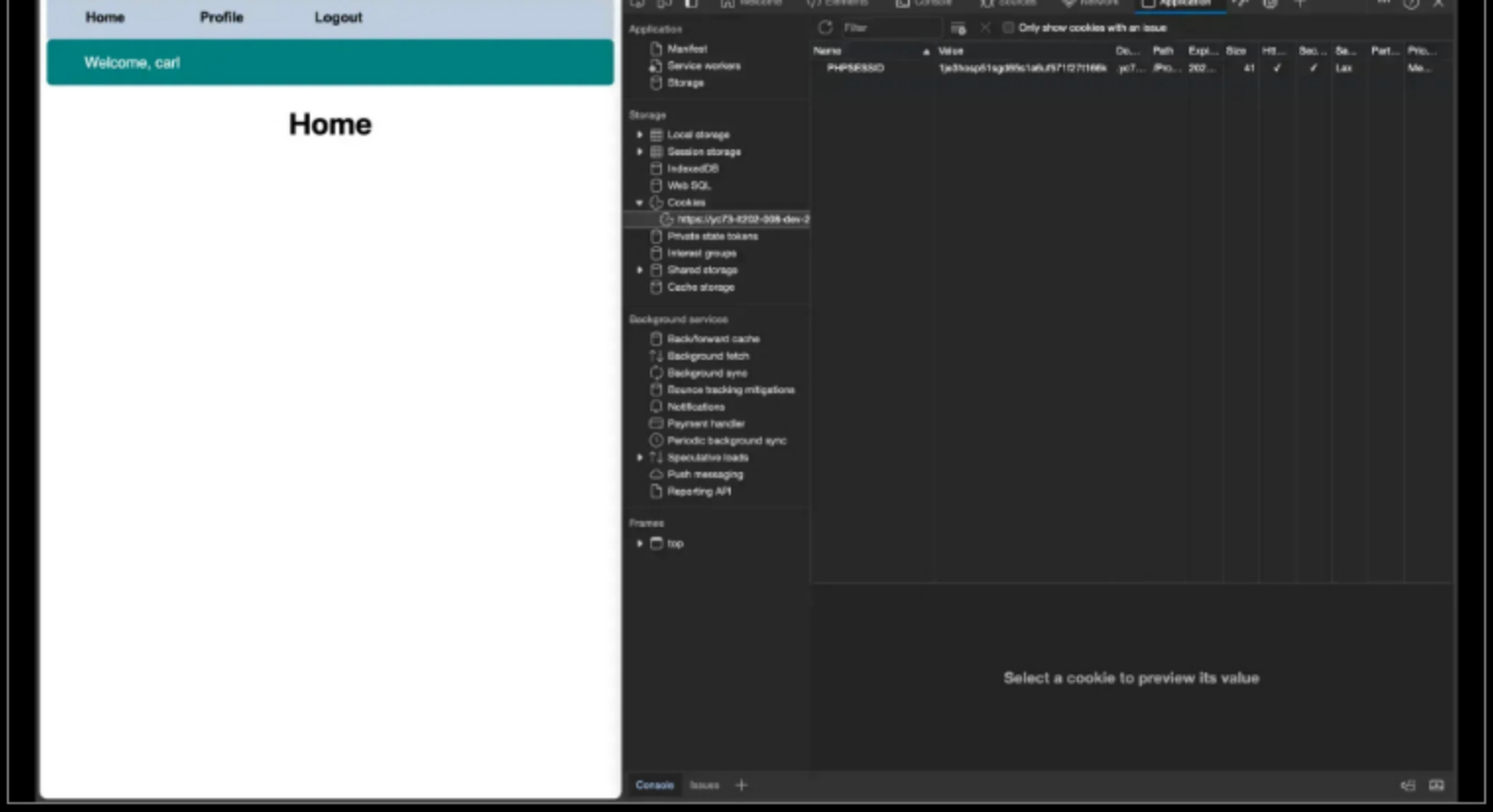


Showing message when password does NOT match what's in the DB

Checklist Items (1)

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

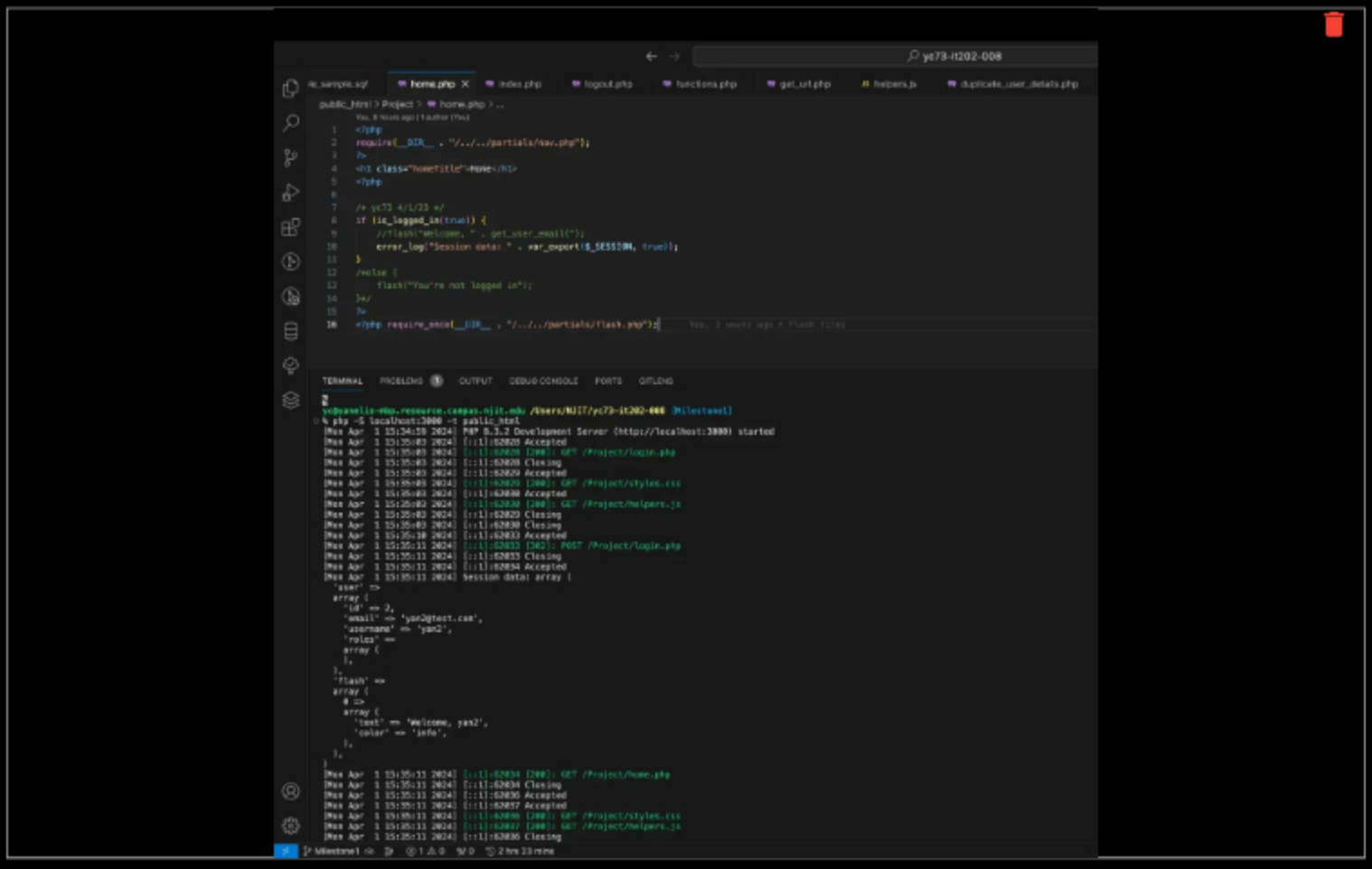




Showing session using developer tools

Checklist Items (1)

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



Showing session data being set (captured from server logs)

Checklist Items (1)

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

COLLAPSE

Task #2 - Points: 1

Text: Screenshot of the form code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Should have proper input types for the fields (note in the caption if you're using two fields for Username/Email or a combined field)
#2	1	Show JavaScript validations (include any extra files related)
#3	1	Show PHP validations (include any lib content)
#4	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
<!-- yc73 4/1/23 -->
<form onsubmit="return validate(this)" method="POST">
    <div class="loginCont">
        <div class="loginCenter">
            <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>
            <div>
                <input type="submit" value="Login" />
            </div>
        </div>
    </div>
</form>
```

Showing (login) form with a COMBINED field (Email/Username)

Checklist Items (1)

#1 Should have proper input types for the fields (note in the caption if you're using two fields for Username/Email or

a combined field)

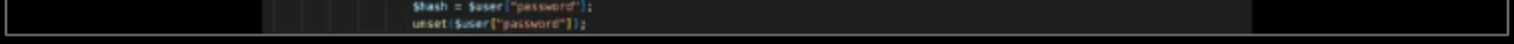
```
<script>
    /* yc73 4/1/23 */
    function validate(form) {
        //TODO 1: implement JavaScript validation
        //ensure it returns false for an error and true for success
        let is_valid = true;
        const email = form.email.value;
        const pw = form.password.value;

        if (email.length === 0) {
            flash("Email/Username must not be empty", "warning");
            is_valid = false;
        }
        if (email.includes("@")) {
            const email_pattern = /^([a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6})+$/;
            if (!email_pattern.test(email)) {
                flash("Invalid email address", "warning");
                is_valid = false;
            }
        } <- #37-43 if (email.includes("@"))
        else {
            const user_pattern = /^[a-z0-9_-]{3,16}$/;
            if(!user_pattern.test(email)) {
                flash("Invalid username", "warning");
                is_valid = false;
            }
        } <- #44-50 else
        if (pw.length === 0) {
            flash("Password must not be empty", "warning");
            is_valid = false;
        }
        else {
            const pw_pattern = /^.{8,}/;
            if(!pw_pattern.test(pw)){
                flash("Password too short", "warning");
                is_valid = false;
            }
        } <- #55-61 else

        //TODO update clientside validation to check if it should
        //valid email or username
        return is_valid;
    } <- #26-66 function validate(form)
</script>
```

Showing JavaScript validations

Checklist Items (1)

#2 Show JavaScript validations (include any extra files related)

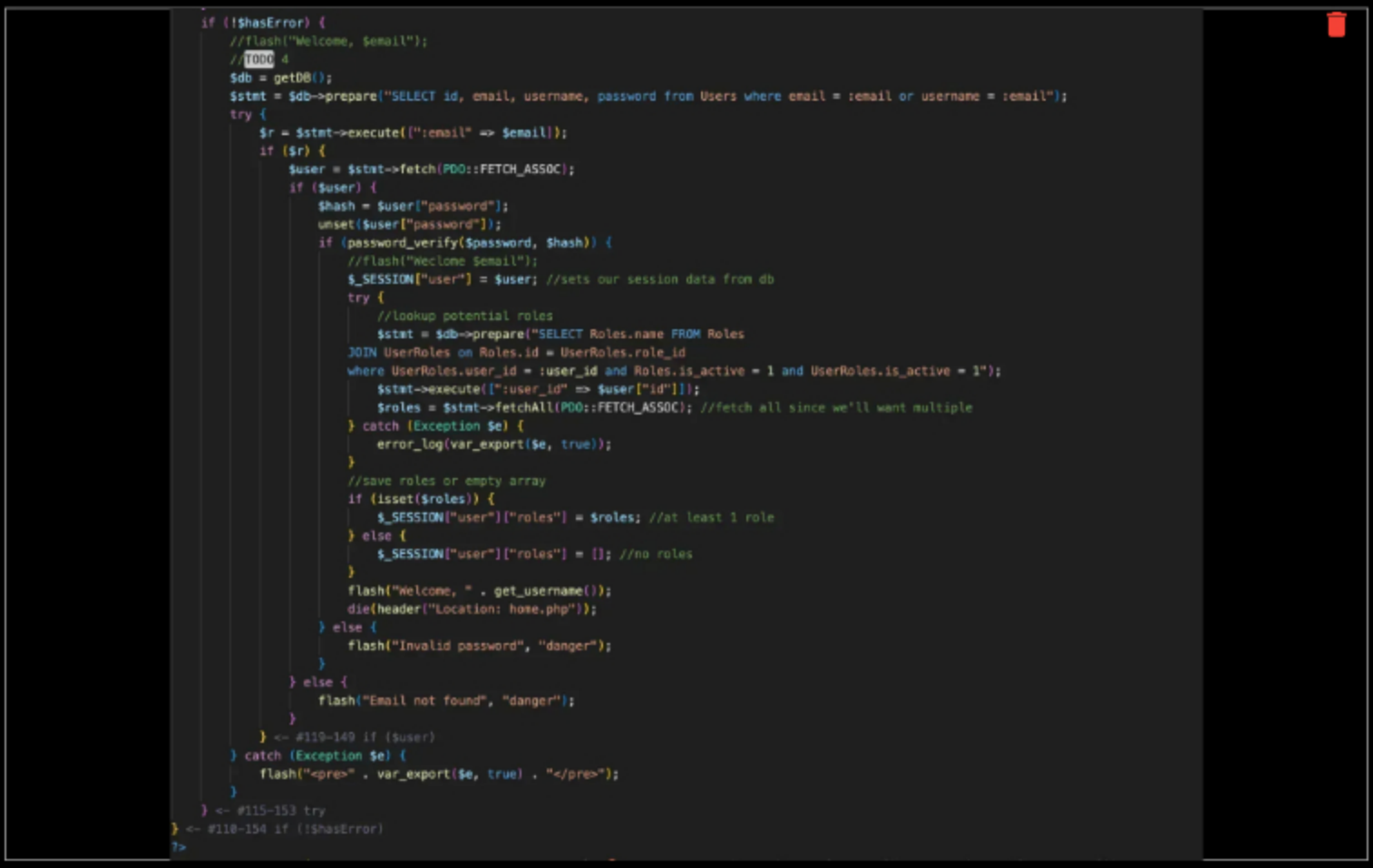
```
<?php
/* yc73 4/1/23 */

//TODO 2: add PHP Code
if (isset($_POST["email"]) && isset($_POST["password"])) {
    $email = se($_POST, "email", "", false);
    $password = se($_POST, "password", "", false);

    //TODO 3
    $hasError = false;
    if (empty($email)) {
        flash("Email must not be empty", "danger");
        $hasError = true;
    }
    if (str_contains($email, "@")) {
        //sanitize
        //$email = filter_var($email, FILTER_SANITIZE_EMAIL);
        $email = sanitize_email($email);
        //validate
        /*if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            flash("Invalid email address");
            $hasError = true;
        }*/
        if (!is_valid_email($email)) {
            flash("Invalid email address", "danger");
            $hasError = true;
        }

    } else {
        if (!is_valid_username($email)) {
            flash("Invalid username", "danger");
            $hasError = true;
        }
    } <- #96-101 else
    if (empty($password)) {
        flash("Password must not be empty", "danger");
        $hasError = true;
    }
    if (!is_valid_password($password)) {
        flash("Password too short", "danger");
        $hasError = true;
    }
    if (!$hasError) {
        //flash("Welcome, $email");
        //TODO 4
        $db = getDB();
        $stmt = $db->prepare("SELECT id, email, username, password from Users where email = :email or username = :email");
        try {
            $r = $stmt->execute([":email" => $email]);
            if ($r) {
                $user = $stmt->fetch(PDO::FETCH_ASSOC);
                if ($user) {
```



Showing PHP validations (part 1)

Checklist Items (1)

#3 Show PHP validations (include any lib content)



Showing PHP validations (part 2)

Checklist Items (1)

#3 Show PHP validations (include any lib content)

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

Checklist

*The checkboxes are for your own tracking

#	Points	Details
☐ #1	1	Don't just show code, translate things to plain English
☐ #2	1	Explain how the session works and why/how it's used

Response:

- When the login page loads, there is a form for users to input their email/username and password. Upon

submitting that login form, it will trigger the "onsubmit" attribute and "validate" function. The validation function is the client-side JavaScript validation that checks the email, username, and password fields.
    - First, it grabs the values (of what the user entered) using the "form" object ("form.email.value", "form.username.value", and "form.password.value") - since the input fields were named.
    - Then, it stores the values into corresponding variables (email and pw).
    - To check if the fields are empty, it looks at the length of the variable with ".length===0".
    - To check the formatting, regex is used. For email, the regex looks for alphanumerics at the start, followed by an @ symbol, another set of alphanumerics, followed by a dot (.), and ends with 2-6 alphabetic characters. For username, the regex looks for alphanumerics, possible _- characters, and a length of 3-16 characters.
    - To check the length of the password, it uses regex to look for a length of 8 characters or more.
- The "onsubmit" takes a boolean value. So, if any of these validations returns a false, warning messages are displayed to the user and the form is not submitted. However, if none of the validations fail, it returns true and sends the data as POST.
- Thus, if the client-side validation is valid (true), the form data is sent to the server-side, and it goes through PHP validations.
    - First, it checks if the POST array contains email and password. If it does, saferecho uses $_POST to retrieve the data, then sets it to variables, and begins validating the data.
    - To check if the email and passwords are empty it uses an empty() function.
    - To check if the email format is valid, it must first sanitize it with "sanitize_email()" to remove (filter/trim) illegal characters. Then, it can be validated with "is_valid_email()" which uses FILTER_VALIDATE_EMAIL to check for invalid formatting/characters.
    - To check if the username format is valid, it uses "is_valid_username()", which checks with regex if it has alphanumerics, possibly contains _-, and is between 3-16 characters.
    - To check if the password has a valid length, it uses "is_valid_password()" (in "sanitizers.php"), which uses "strlen" to see if the length is equal to or greater than 8.
- If any fields fail the validations, it returns true for "hasError" and flashes error messages to the user.
- However, if all the fields pass the validations, it returns false for "hasError" (meaning there are no errors) and proceeds to retrieve data from the "Users" table in the database (using a prepared statement).
    - It first checks if the entered email or username exists in the database. If it does NOT, the user receives a message saying the email/username does not exist.
    - If the email/username DOES exist, then it retrieves the hashed password to compare it to the password that was entered.
    - If the passwords do NOT match, the user receives a message saying the password is invalid.
    - If the passwords DO match, the user (data) is saved to the session.
    - It will also use the user's id to try and find any assigned roles from the "User Roles" table. If any roles are found they are also saved to the session.
    - These sessions are initialized by "session_start()" (in nav) and can save user data (in $_SESSION) (temporarily as they expire after 24 minutes). Sessions can also be terminated with "session_destroy()" or by letting them expire naturally - this clears the session data. So, sessions are used to identify users and persist their access across different pages of the website.

COLLAPSE

Task #4 - Points: 1

Text: Include pull request links related to this feature

Details:

Should end in /pull/#

Should end in /pull/#

URL #1
https://github.com/yaneliii/yc73-it202-008/pull/18

URL #2
https://github.com/yaneliii/yc73-it202-008/pull/21

URL #3
https://github.com/yaneliii/yc73-it202-008/pull/26

User Logout (1 pt.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Capture the following screenshots

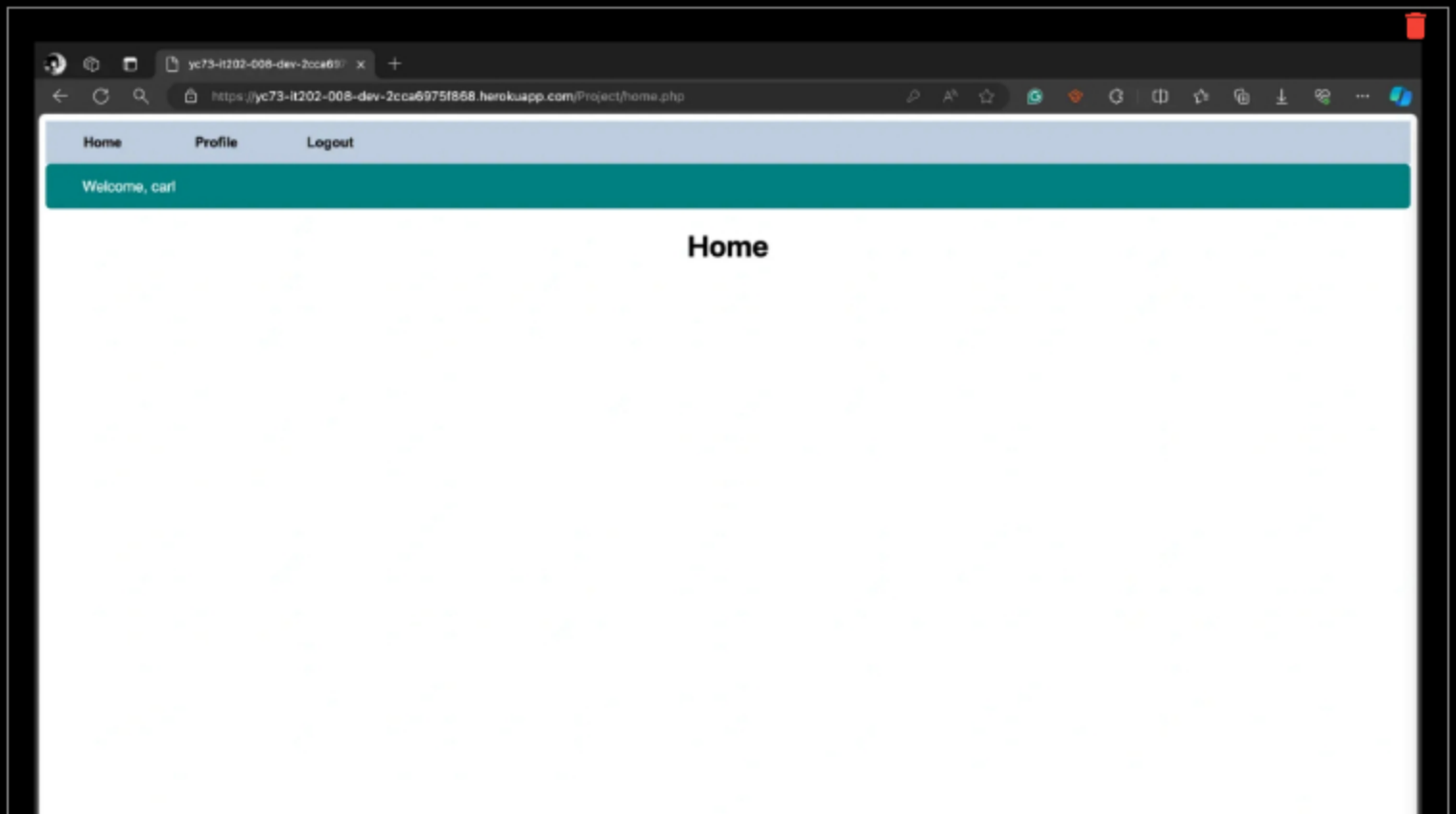
Checklist *The checkboxes are for your own tracking

#	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

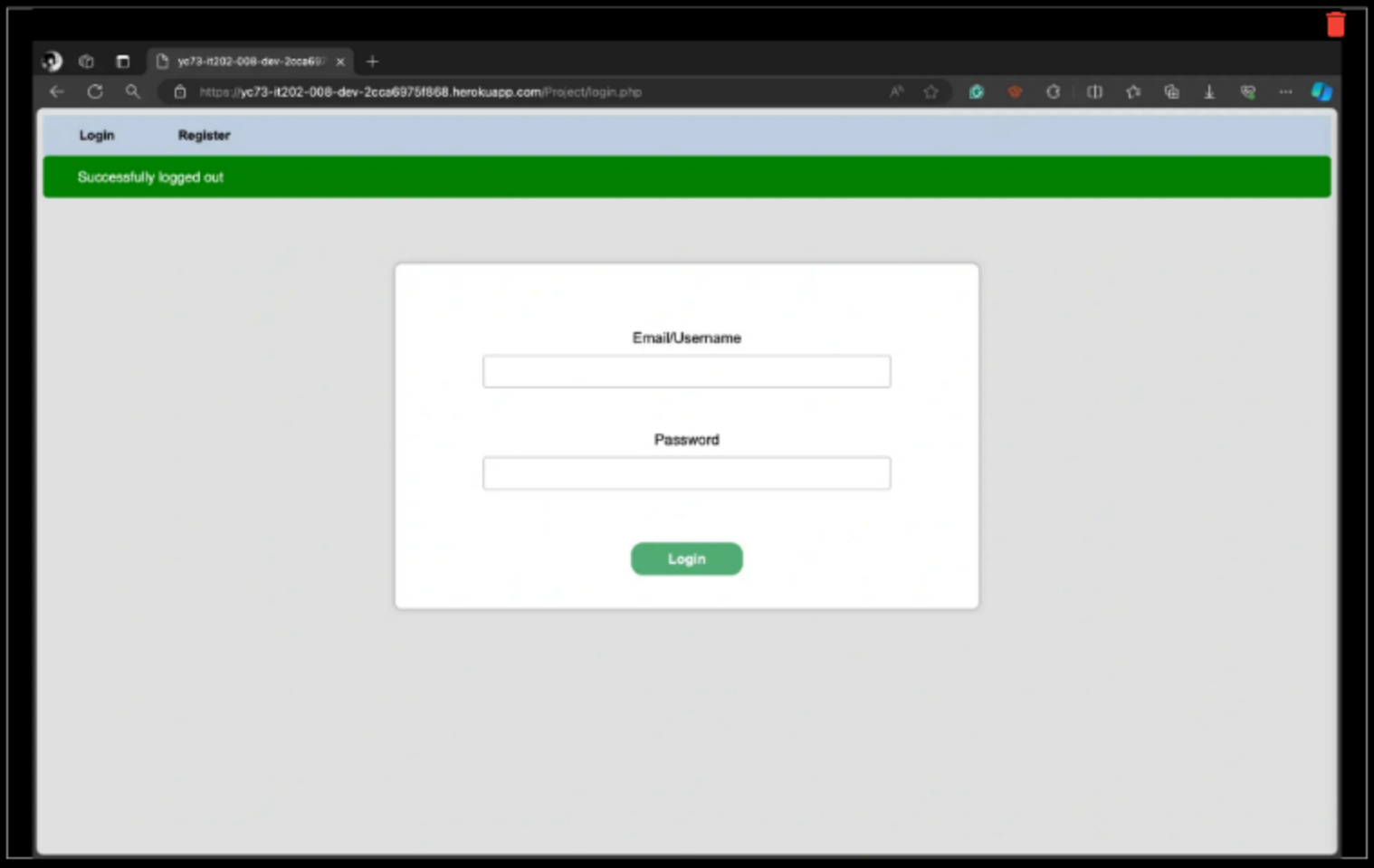
Small Medium Large



Showing navigation when logged in (site)

Checklist Items (1)

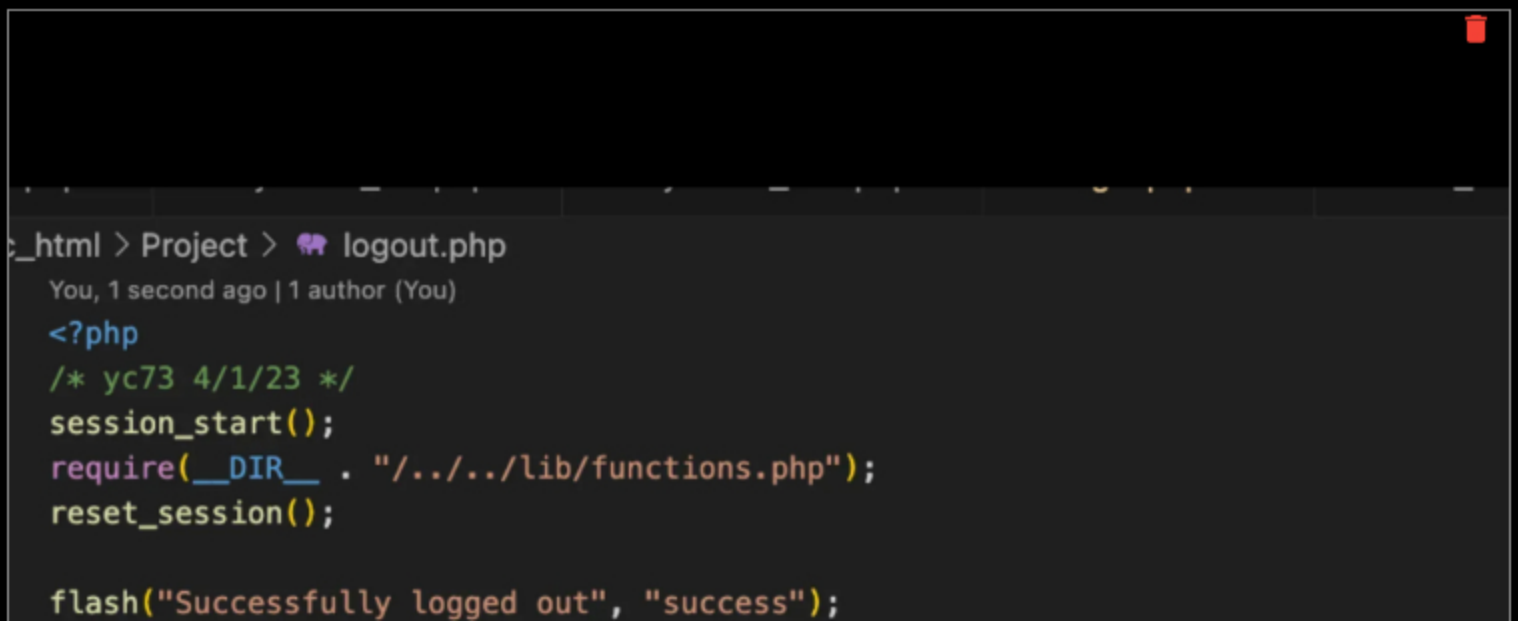
#1 Screenshot of the navigation when logged in (site)



Showing redirect to login with the user-friendly logged-out message (site)

Checklist Items (1)

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



```php
header("Location: login.php");
```

Showing logout code

Checklist Items (0)

reset_session.php > ...

You, 1 second ago | 1 author (You)

```php
<?php
/* yc73 4/1/23 */
function reset_session()
{
    session_unset();
    session_destroy();
    session_start();
} <- #4-8 function reset_session()
```

Screenshot of logout-related code showing the session is destroyed

Checklist Items (1)

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

COLLAPSE

Task #2 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1

URL #1

https://github.com/yaneliii/yc73-it202-008/pull/18

URL #2

https://github.com/yaneliii/yc73-it202-008/pull/22

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
/* yc73 4/1/23 */
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");
        die(header("Location: $destination"));
    } <- #12-16 if ($redirect && !$isLoggedIn)
    return $isLoggedIn;
} <- #10-18 function is_logged_in($redirect = false, $destination = "logi...
function has_role($role)
{
    if (is_logged_in() && isset($_SESSION["user"]["roles"])) {
        foreach ($_SESSION["user"]["roles"] as $r) {
            if ($r["name"] === $role) {
                return true;
            }
        } <- #22-26 foreach ($_SESSION["user"]["roles"] as $r)
    } <- #21-27 if (is_logged_in() && isset($_SESSION["user"]["roles"]))
    return false;
} <- #20-29 function has_role($role)
function get_username()
{
    if (is_logged_in()) { //we need to check for login first because "user" key may not exist
        return se($_SESSION["user"], "username", "", false);
    }
    return "";
} <- #31-36 function get_username()
function get_user_email()
{
    if (is_logged_in()) { //we need to check for login first because "user" key may not exist
        return se($_SESSION["user"], "email", "", false);
    }
    return "";
} <- #38-43 function get_user_email()
function get_user_id()
{
    if (is_logged_in()) { //we need to check for login first because "user" key may not exist
        return se($_SESSION["user"], "id", false, false);
    }
    return false;
} <- #45-50 function get_user_id()
```

Showing the function that checks if a user is logged in

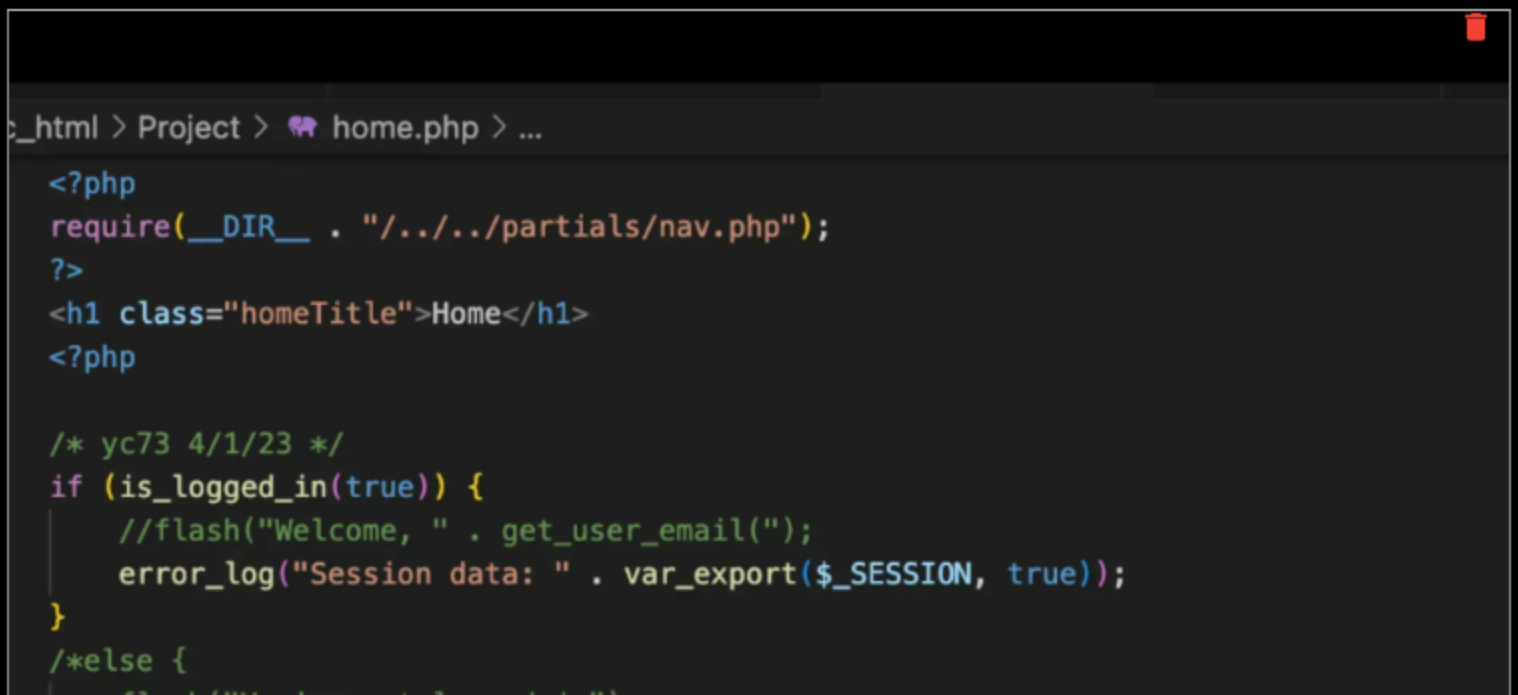
Checklist Items (1)

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

Showing the login check function being used in "Profile" page (also used on "Home" page)

Checklist Items (1)

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

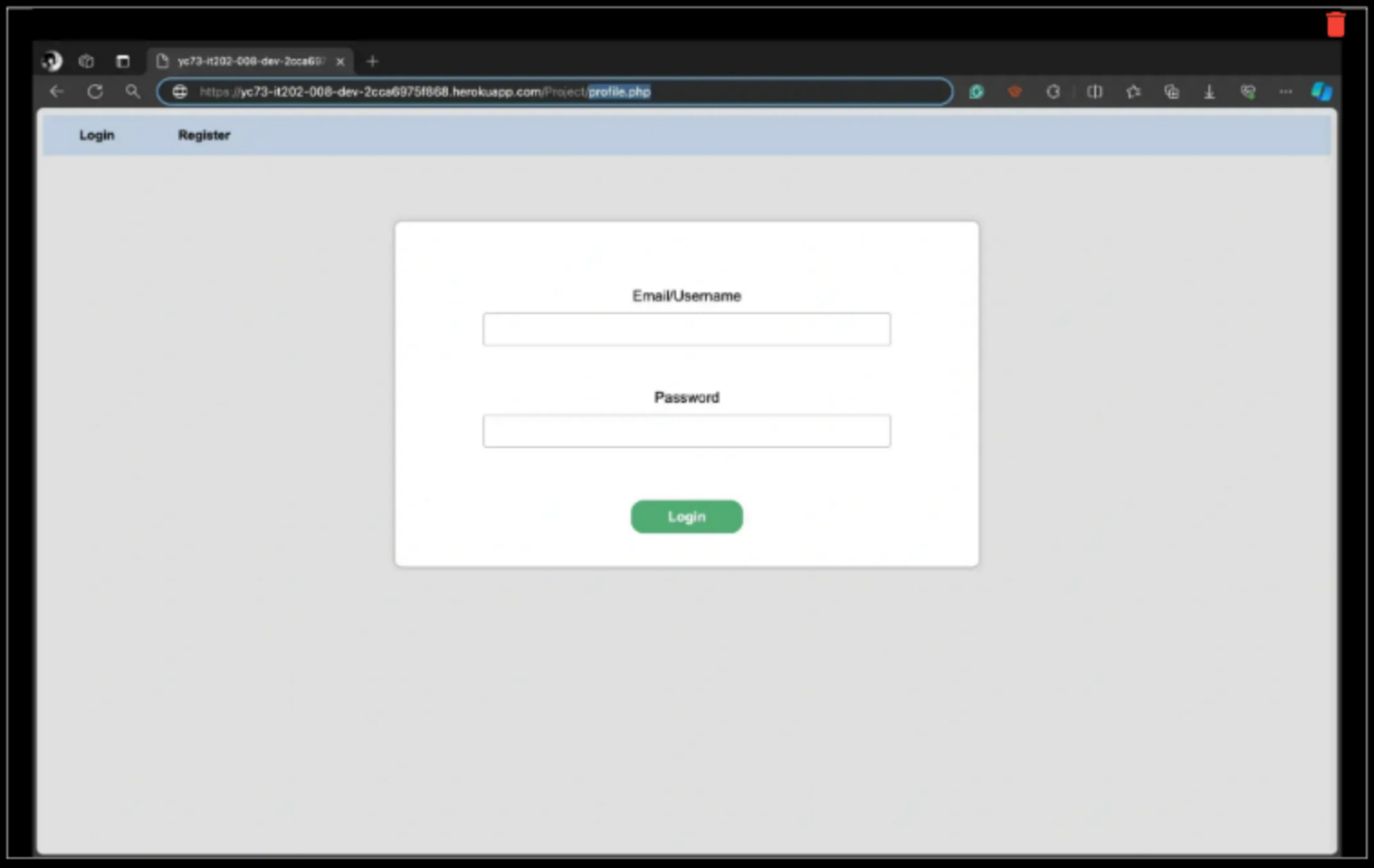


```
    flash("You're not logged in");
}*/
?>
<?php require_once(__DIR__ . "/../../partials/flash.php");
```
You, 2 weeks ago • f

Showing the login check function being used in "Home" page (also used on "Profile" page)

Checklist Items (1)

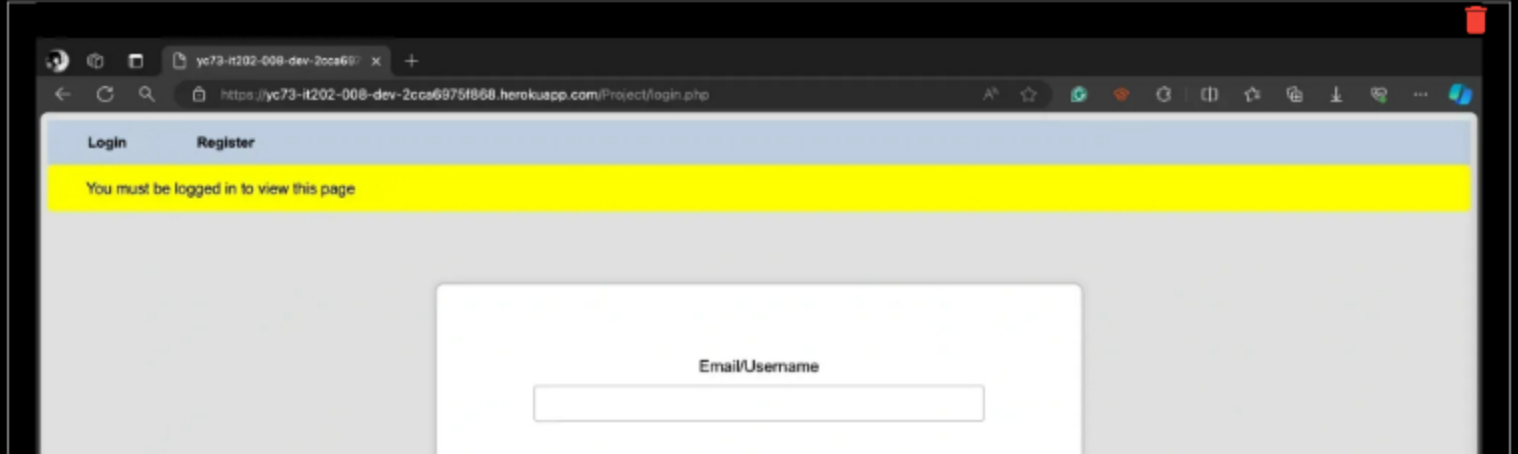
#2 Screenshot of the login check function being used (i.e., profile likely). Also caption what pages it's used on

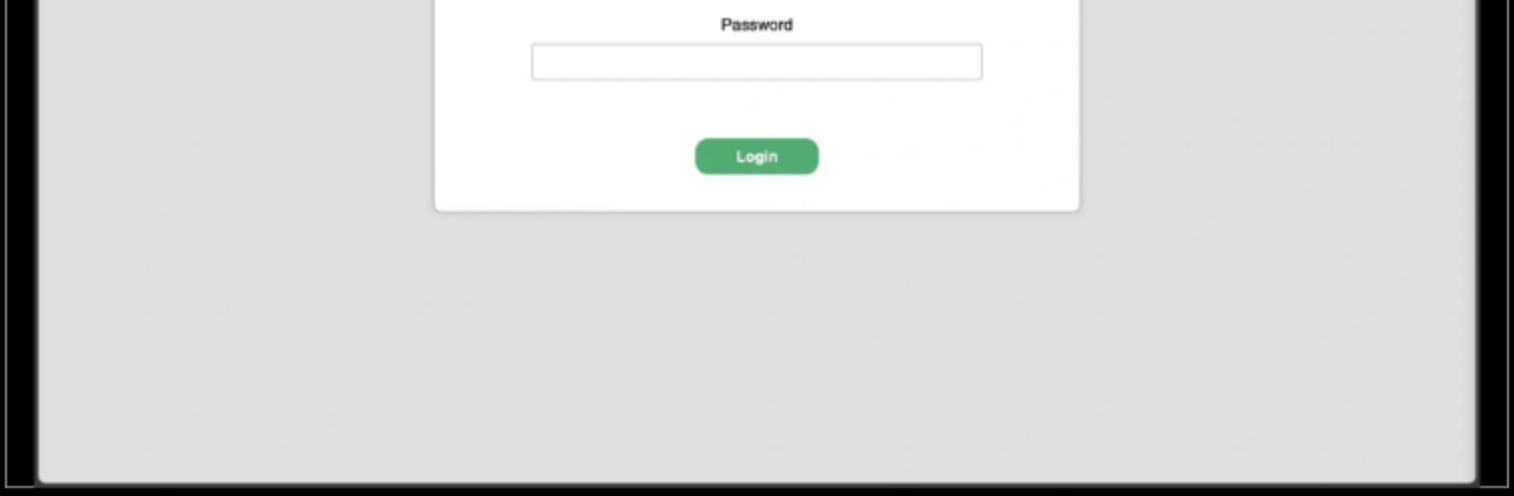


Showing in search bar the login-protected page (Profile) that I am attempting to access (while being logged out)

Checklist Items (1)

#4 Demonstrate the user-friendly message of trying to manually access a login-protected page while being logged out





Showing message when trying to manually access a login-protected page while being logged out

Checklist Items (1)

#4 Demonstrate the user-friendly message of trying to manually access a login-protected page while being logged out

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

user_helpers.php > get_user_id

```
} <- #10-18 function is_logged_in($redirect = false, $destination = "logi...
/* yc73 4/1/23 */
function has_role($role)
{
    if (is_logged_in() && isset($_SESSION["user"]["roles"])) {
        foreach ($_SESSION["user"]["roles"] as $r) {
            if ($r["name"] === $role) {
                return true;
            }
        } <- #23-27 foreach ($_SESSION["user"]["roles"] as $r)
    } <- #22-28 if (is_logged_in() && isset($_SESSION["user"]["roles"]))
    return false;
} <- #21-30 function has_role($role)
function get_username()
{
    if (is_logged_in()) { //we need to check for login first because "user" key may not exist
        return se($_SESSION["user"], "username", "", false);
```

```php
    }
    return "";
} <- #32-37 function get_username()
function get_user_email()
{
    if (is_logged_in()) { //we need to check for login first because "user" key may not exist
        return se($_SESSION["user"], "email", "", false);
    }
    return "";
} <- #39-44 function get_user_email()
function get_user_id()
{
    if (is_logged_in()) { //we need to check for login first because "user" key may not exist
        return se($_SESSION["user"], "id", false, false);
    }
    return false;
```

Showing the function that checks for a specific role

Checklist Items (1)

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

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

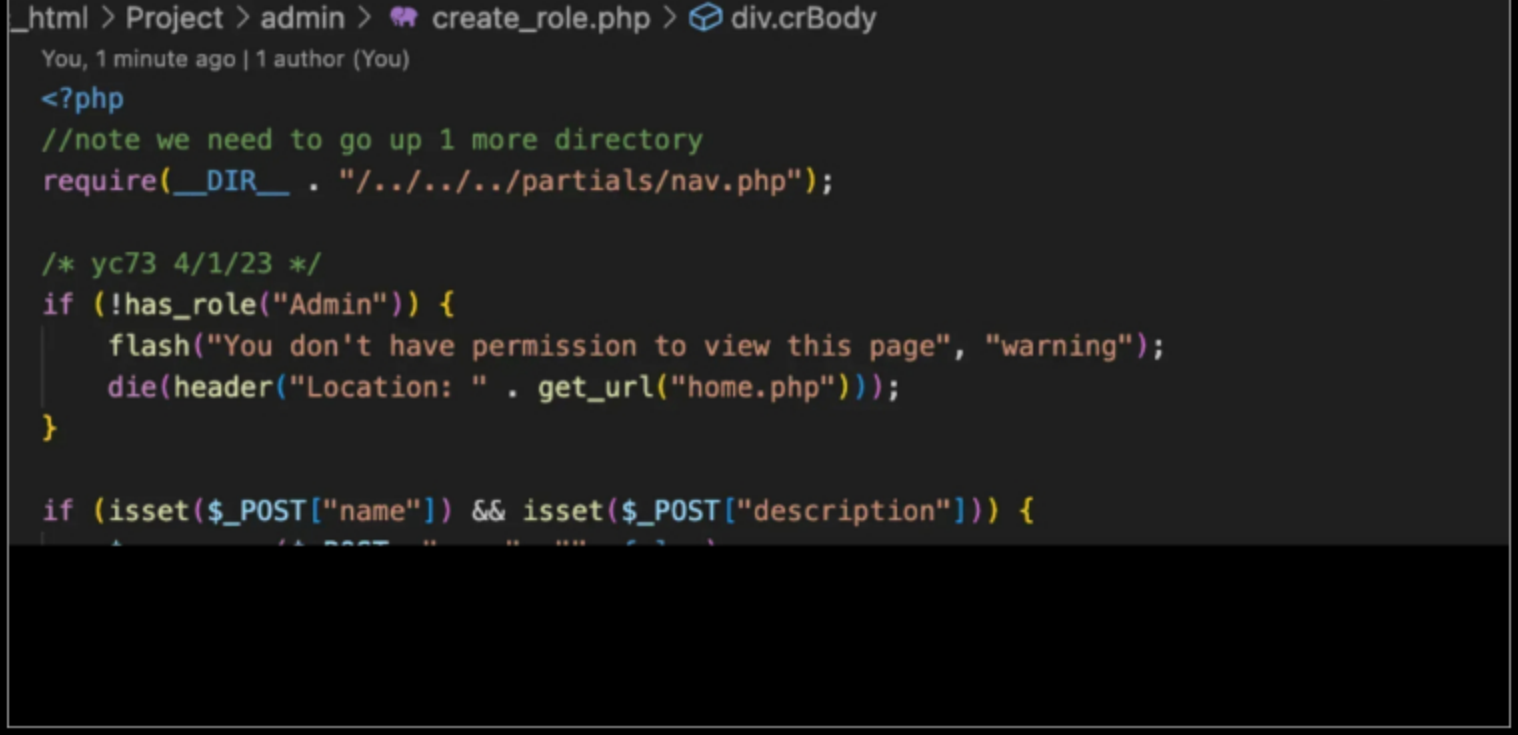
You, 14 seconds ago | 1 author (You)

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

/* yc73 4/1/23 */
if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: $BASE_PATH" . "/home.php"));
}
//attempt to apply
```

Showing the role check function being used in the "Assign Roles" page (also used in "Create Roles and List Roles" pages)

Checklist Items (1)

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



Showing the role check function being used in the "Create Roles" page (also used in "Assign Roles and List Roles" pages)

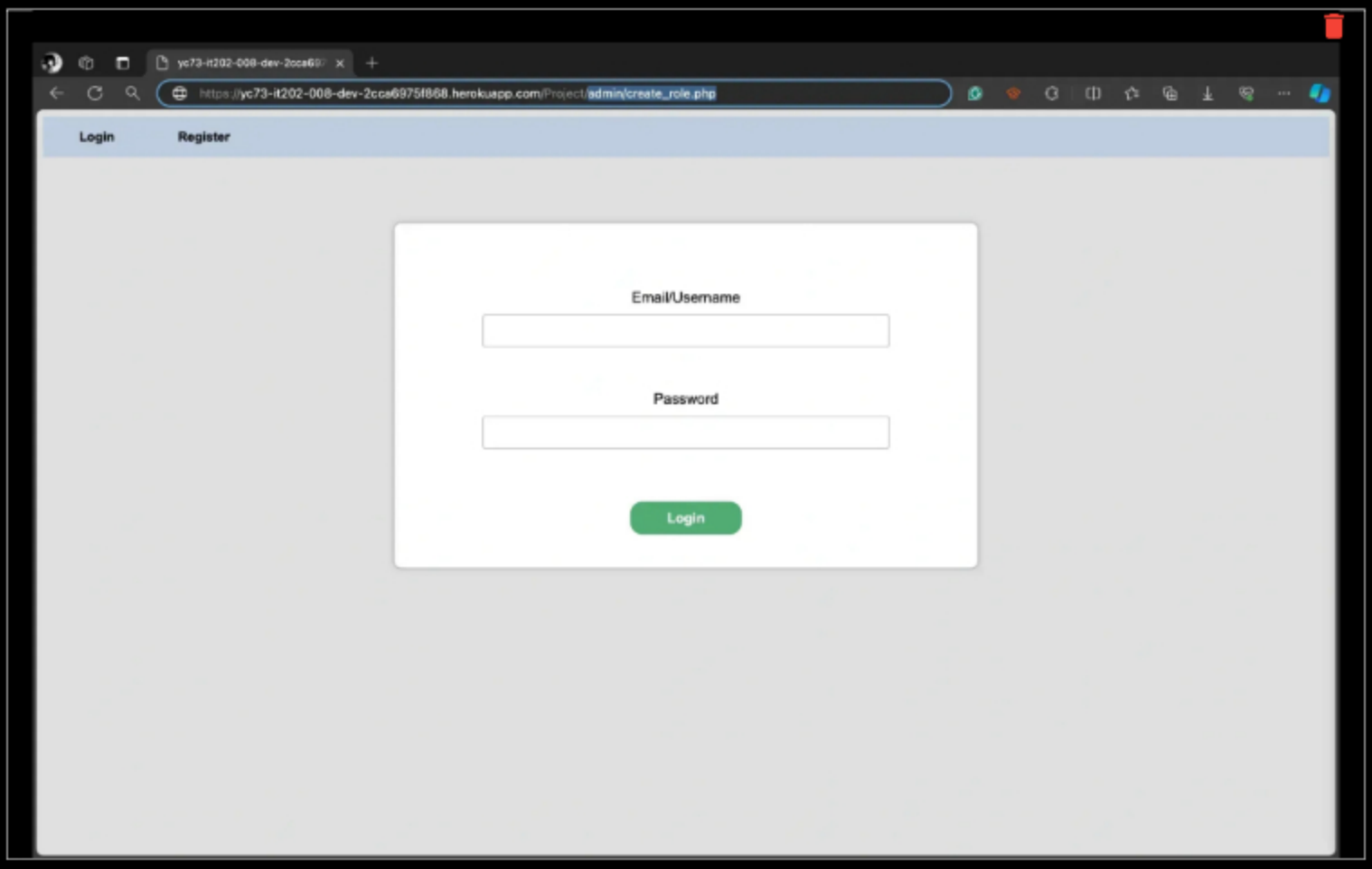
Checklist Items (1)

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

Showing the role check function being used in the "List Roles" page (also used in "Assign Roles and Create Roles" pages)

Checklist Items (1)

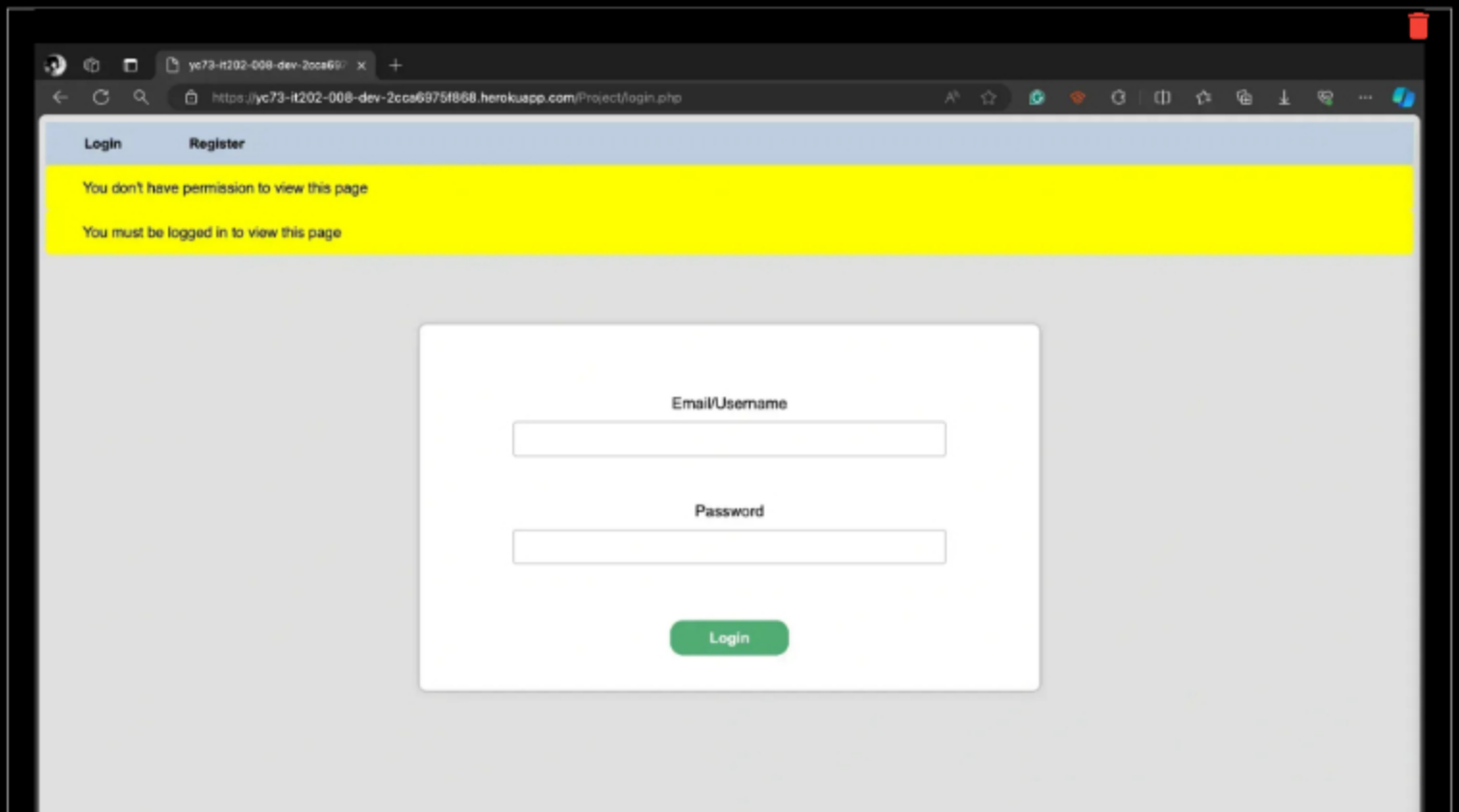
#2 Screenshot of the role check function being used. Also caption what pages it's used on



Showing in search bar the role-protected page (Create Roles) that I am attempting to access (while being logged out)

Checklist Items (1)

#4 Demonstrate the user-friendly message of trying to manually access a role-protected page while being logged out



Showing message when trying to manually access a role-protected page while being logged out

Checklist Items (1)

#4 Demonstrate the user-friendly message of trying to manually access a role-protected page while being logged out

COLLAPSE

Task #3 - Points: 1

Text: Screenshots of UserRoles and Roles Tables

Checklist

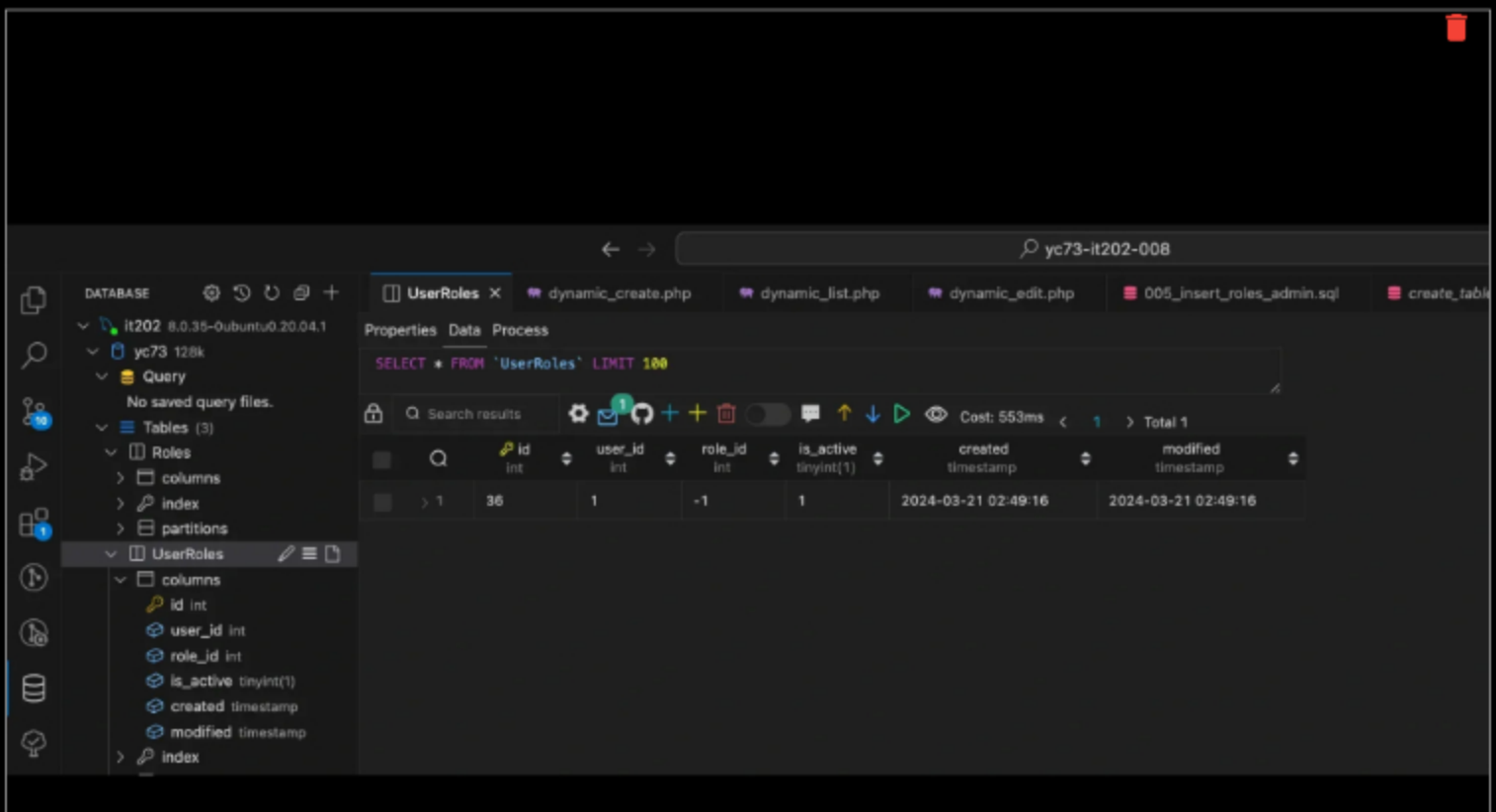
*The checkboxes are for your own tracking

#	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

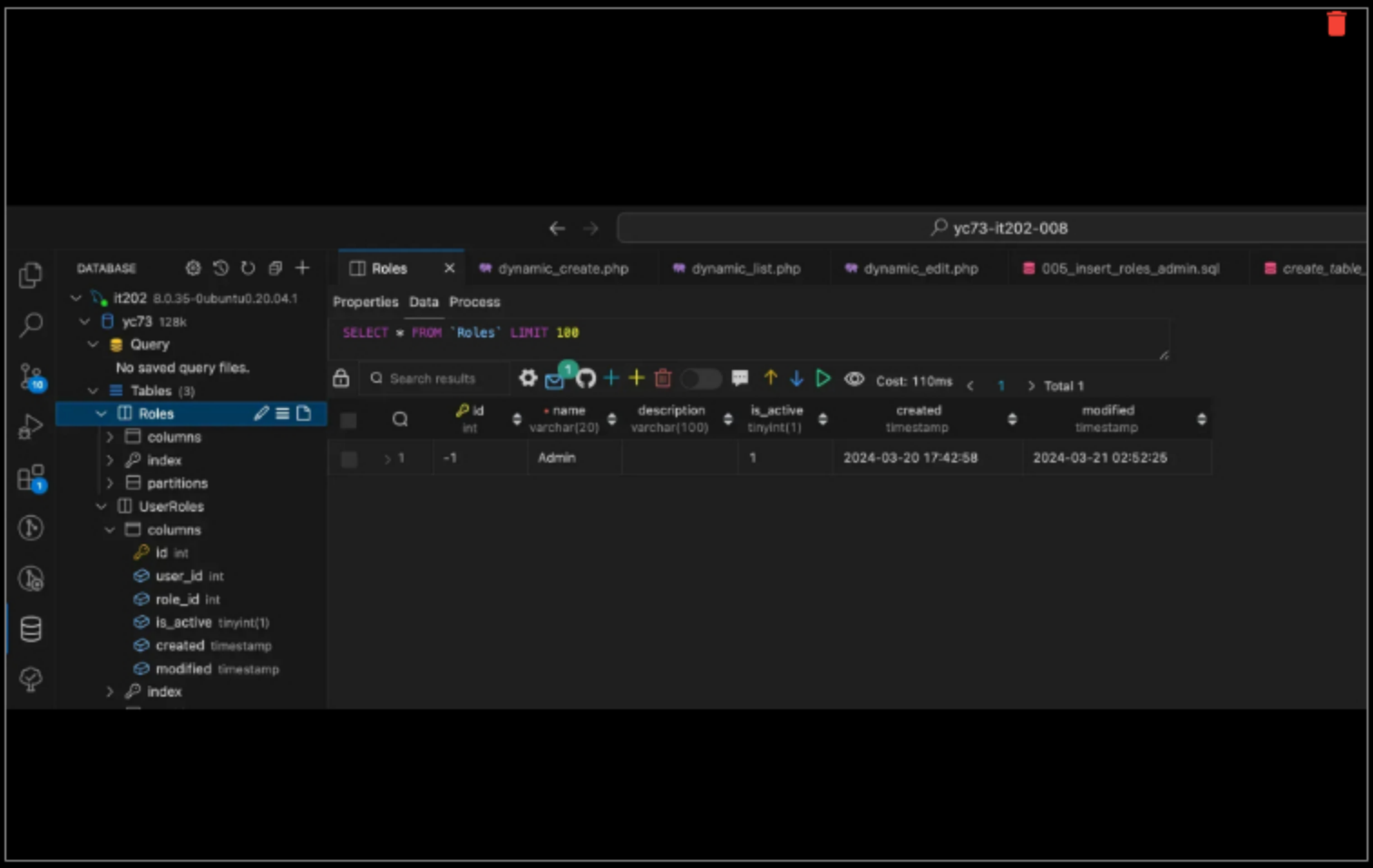
Small Medium Large



Showing valid and enabled User->Role reference (UserRoles table)

Checklist Items (1)

#1 At least one valid and enabled User->Role reference (UserRoles table)



Showing valid and enabled Role (Roles table)

Checklist Items (1)

#4 At least one valid and enabled Role (Roles table)

COLLAPSE

Task #4 - Points: 1

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:

To load the role-protected pages, the Roles and UserRoles tables work in conjunction with the Users table to manage

user access levels. The Users table contains information on the users - such as their email, username, and password - and gives a unique "id" for each user. The "Roles" table holds the defined roles and assigns each a unique "id". The "UserRoles" table has the purpose of acting as a bridge to 'connect' the data between the "Users" and "Roles" tables - basically managing the assignment of roles so we can check which users have a role. It contains foreign keys for "user_id" that reference the user's "id" from the "Users" table and "role_id" that reference the roles "id" from the "Roles" table. So, with the "UserRoles" table we can check if a specific user exists as ever having a role (based on the user id), and if they do exist (as having a role) then we can check what role they have (with role id). The difference between "Roles.is_active" and "UserRoles.is_active" is "Roles.is_active" shows whether the actual role itself is active or not, for instance, a role can be disabled to prevent access to certain pages to everyone with the role. While "UserRoles.is_active" shows whether a specific user's assignment to a role is active or not, without affecting the role itself - meaning the status of which users have access can be controlled individually.

COLLAPSE

Task #5 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/yaneliii/yc73-it202-008/pull/19

URL #2
https://github.com/yaneliii/yc73-it202-008/pull/22

URL #3
https://github.com/yaneliii/yc73-it202-008/pull/23

User Profile (2 pts.)

COLLAPSE

Task #1 - Points: 1

Text: View Profile Website Page

Checklist

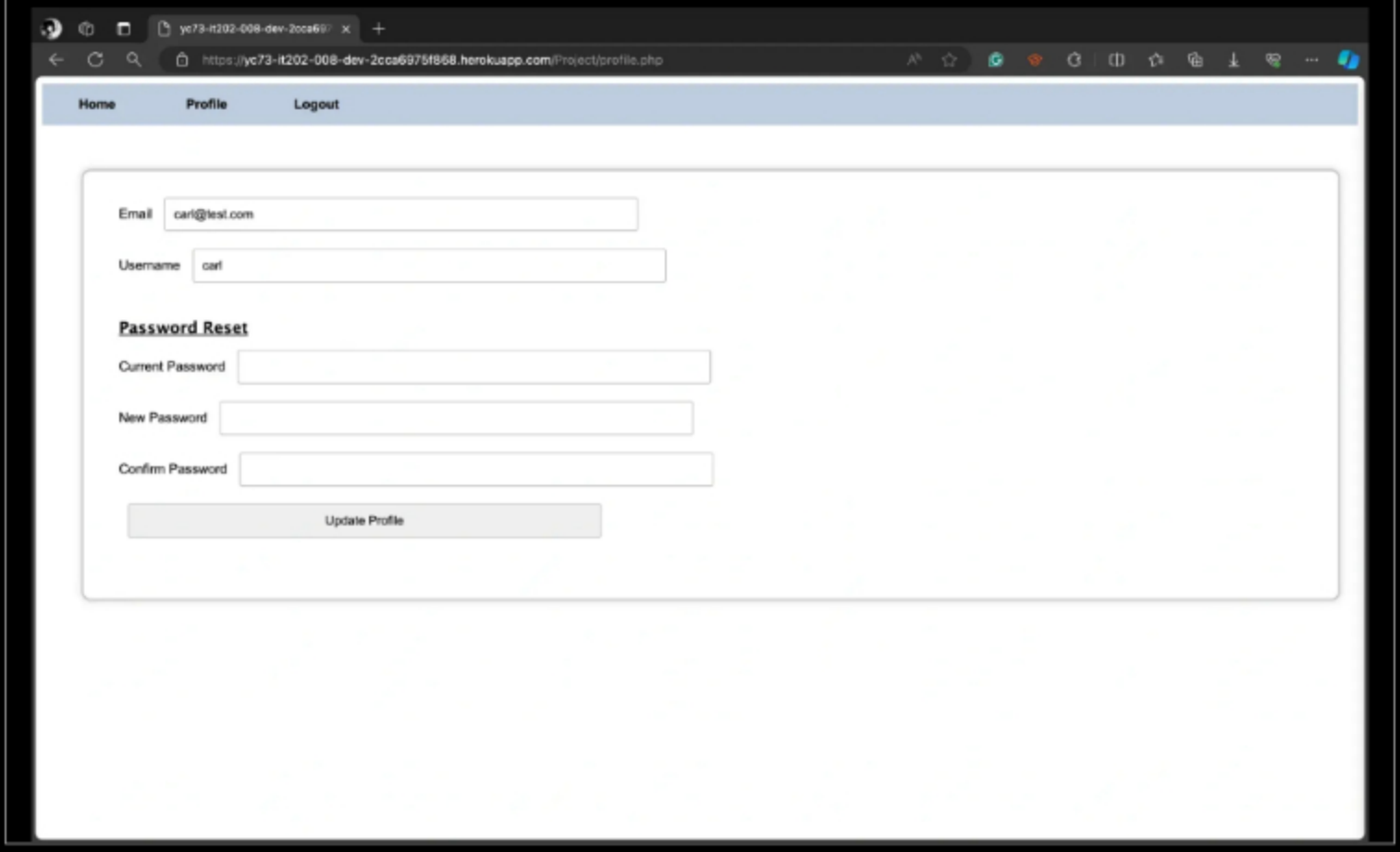
*The checkboxes are for your own tracking

#	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



Showing profile form correctly populated on page load (username, email) (along with the fields: current password, new password, confirm password)

Checklist Items (2)

#3 Show the profile form correctly populated on page load (username, email)

#4 Should have the following fields: username, email, current password, new password, confirm password (or similar)

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:

When the profile page is visited, it must first check if the user is logged in. If the user is logged in and no form has been submitted, it grabs the user's email and username with the helper functions "get_user_email()" and "get_username()". These functions first check if the user is logged in, in case they do not exist, and if they do exist it grabs the email and username from the session data - specifically the "$_SESSION["user"]" array. Then, while generating the profile (HTML)

form, the php safer echo function is used to prefill the retrieved values, it makes sure the data is safe to display and then outputs the value to the email and username fields.

COLLAPSE

Task #3 - Points: 1

Text: Edit Profile Website Page

Checklist

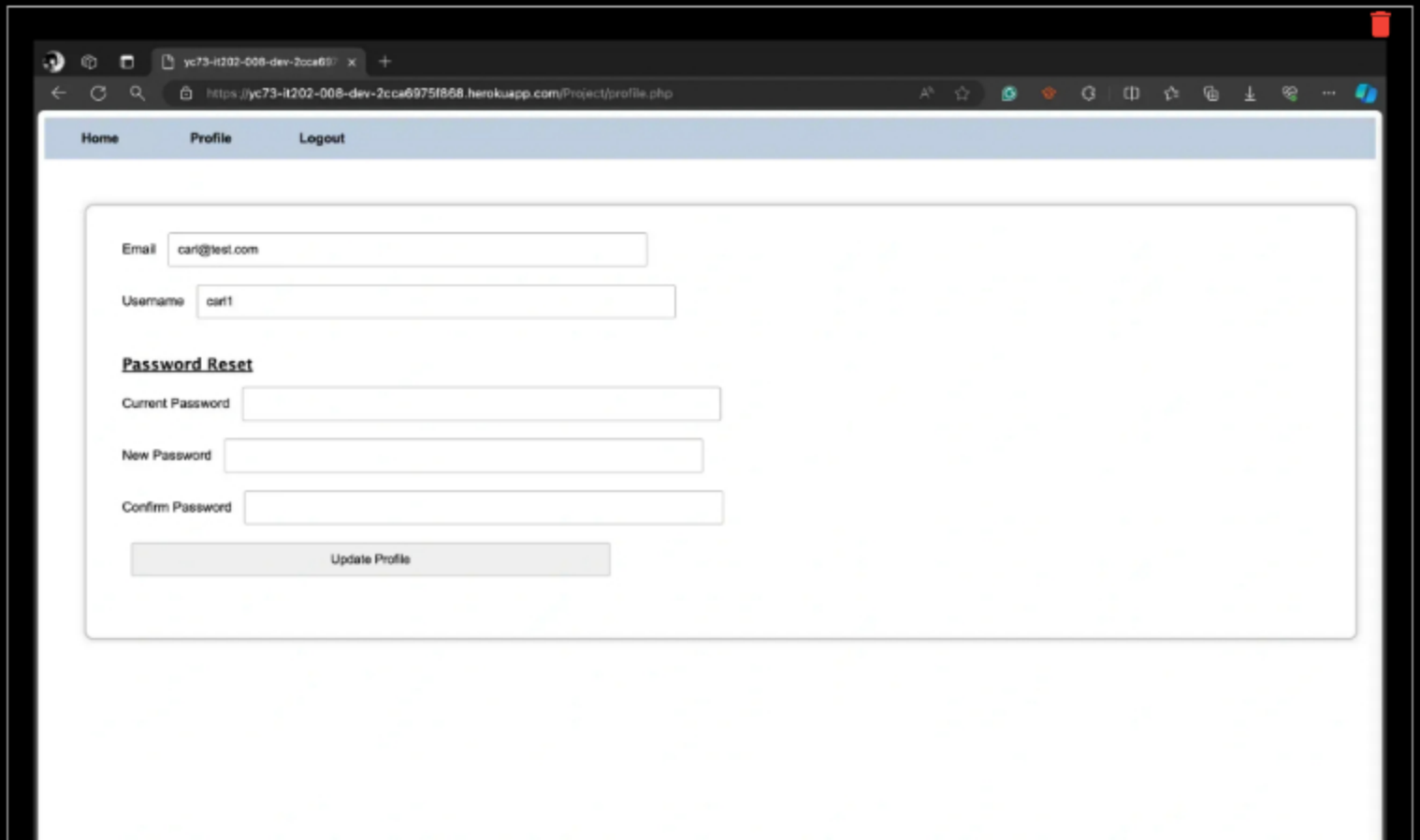
*The checkboxes are for your own tracking

#	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

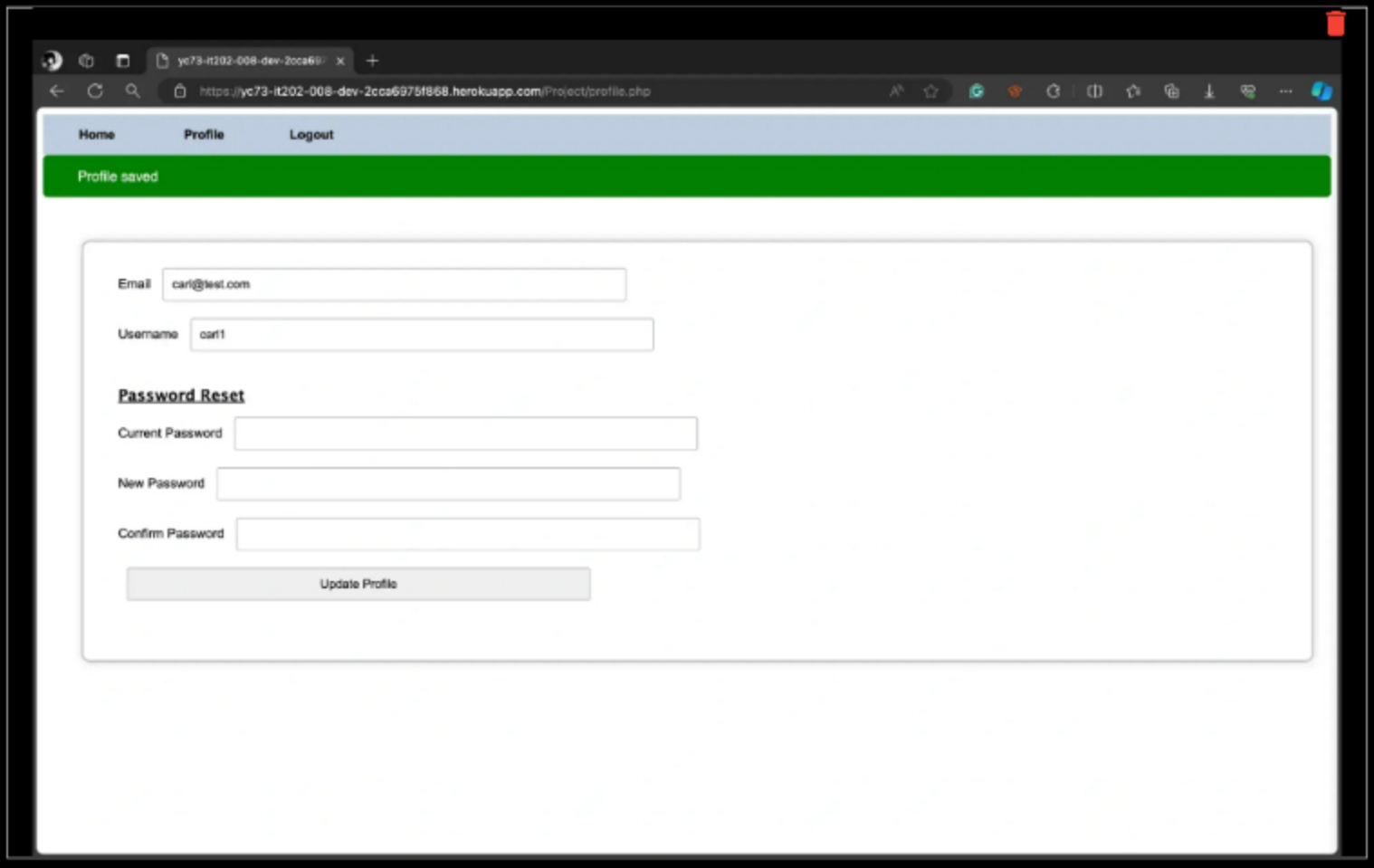
Small Medium Large



Showing BEFORE username change

Checklist Items (1)

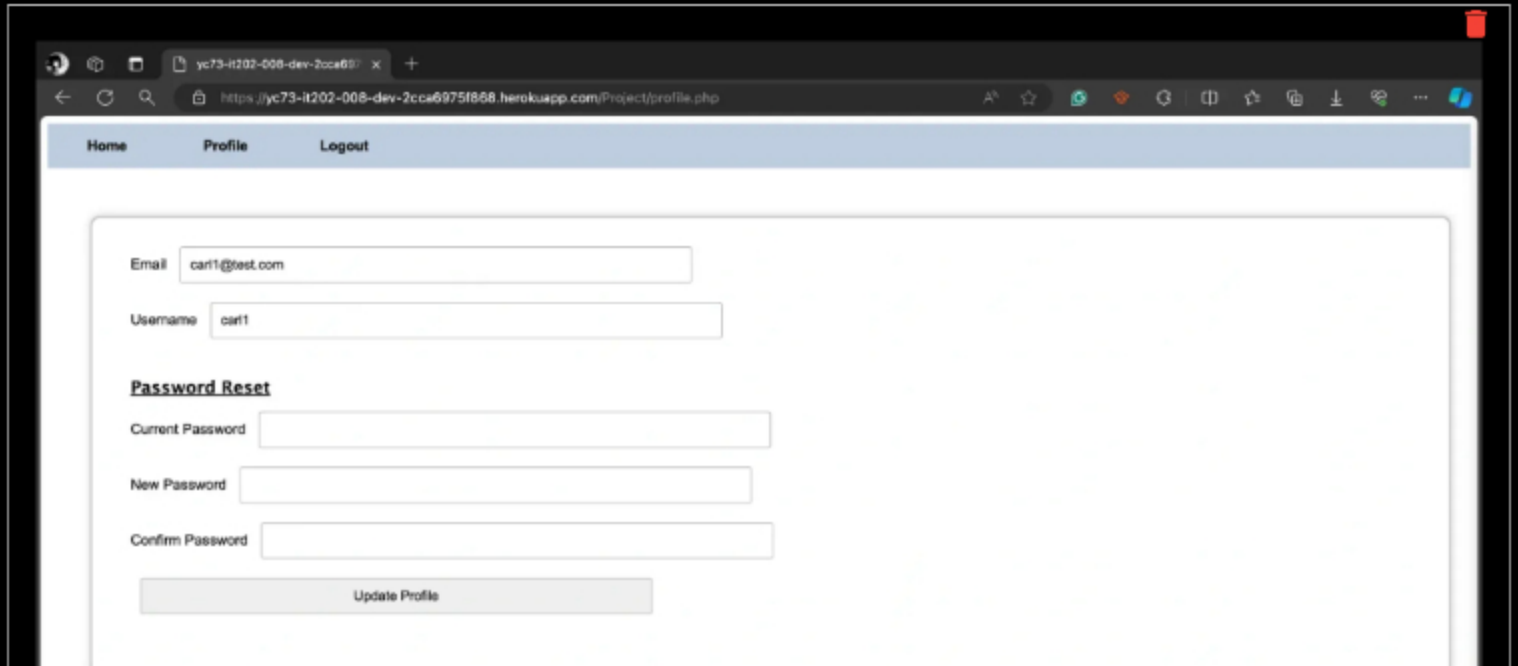
#3 Demonstrate with before and after of a username change (including success message)



Showing AFTER username change with success message

Checklist Items (1)

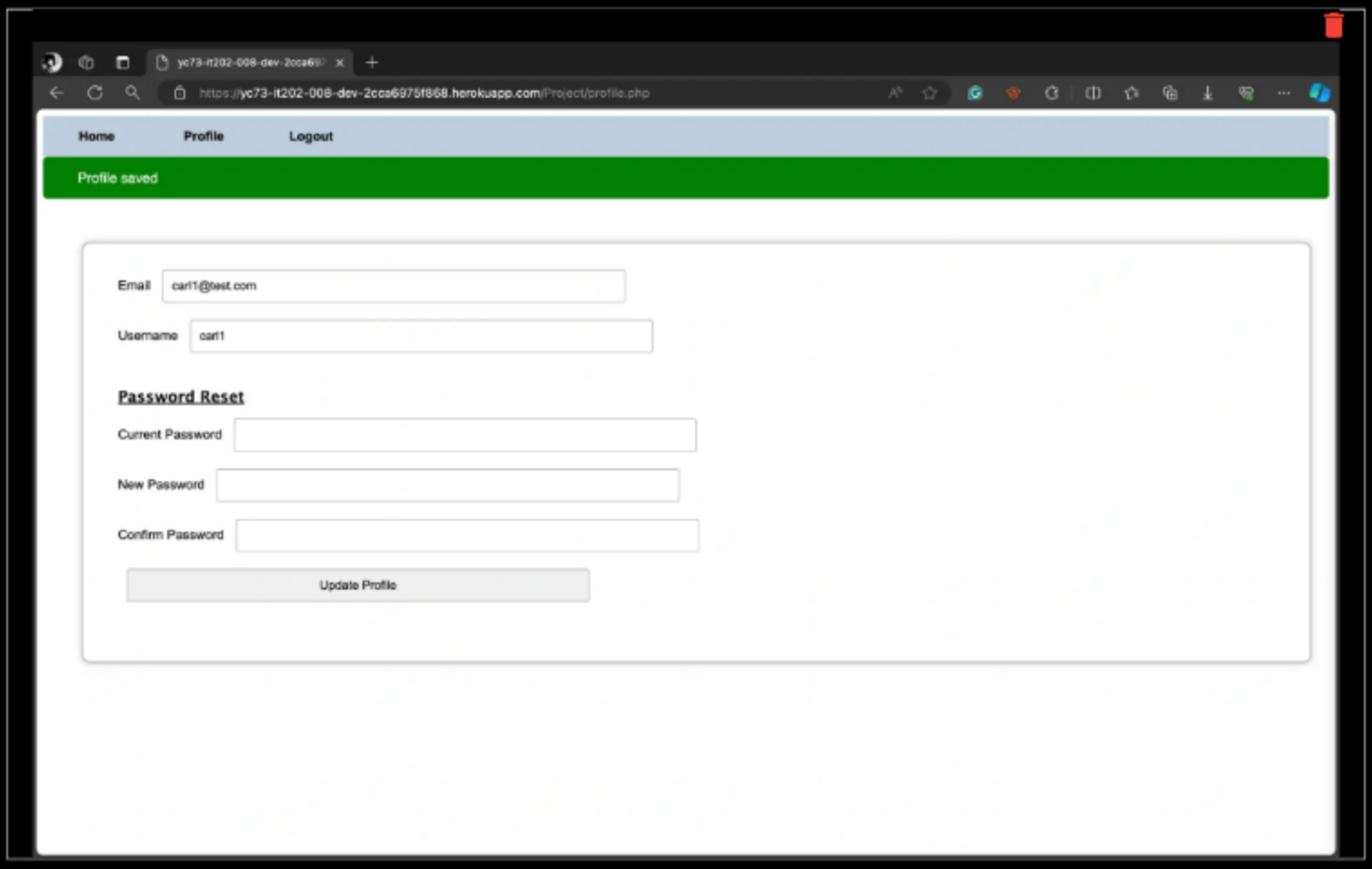
#3 Demonstrate with before and after of a username change (including success message)



Showing BEFORE email change

Checklist Items (1)

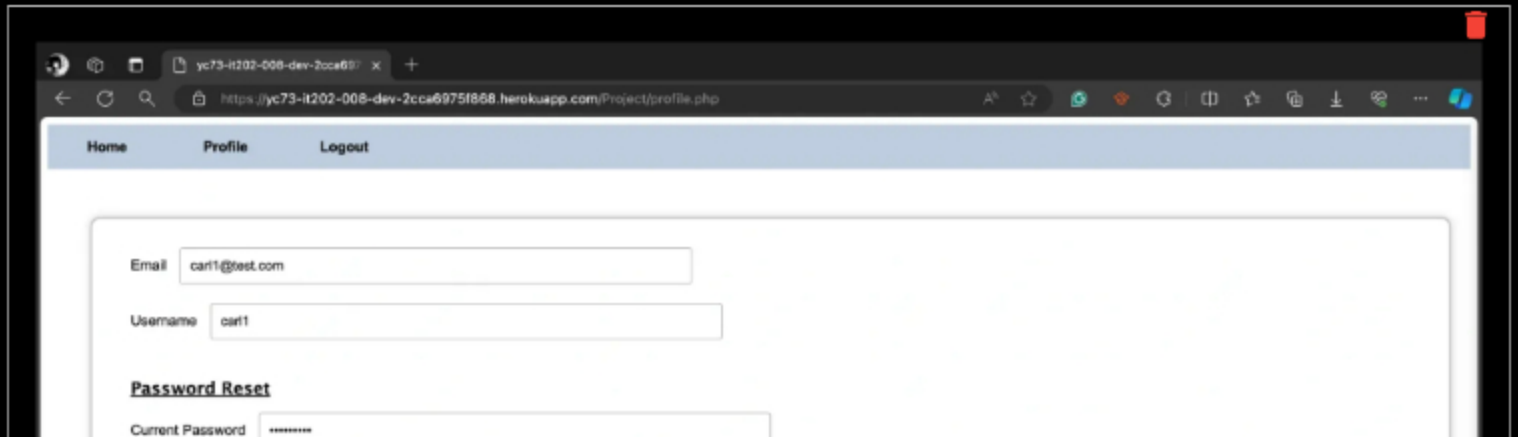
#4 Demonstrate with a before and after of an email change (including success message)

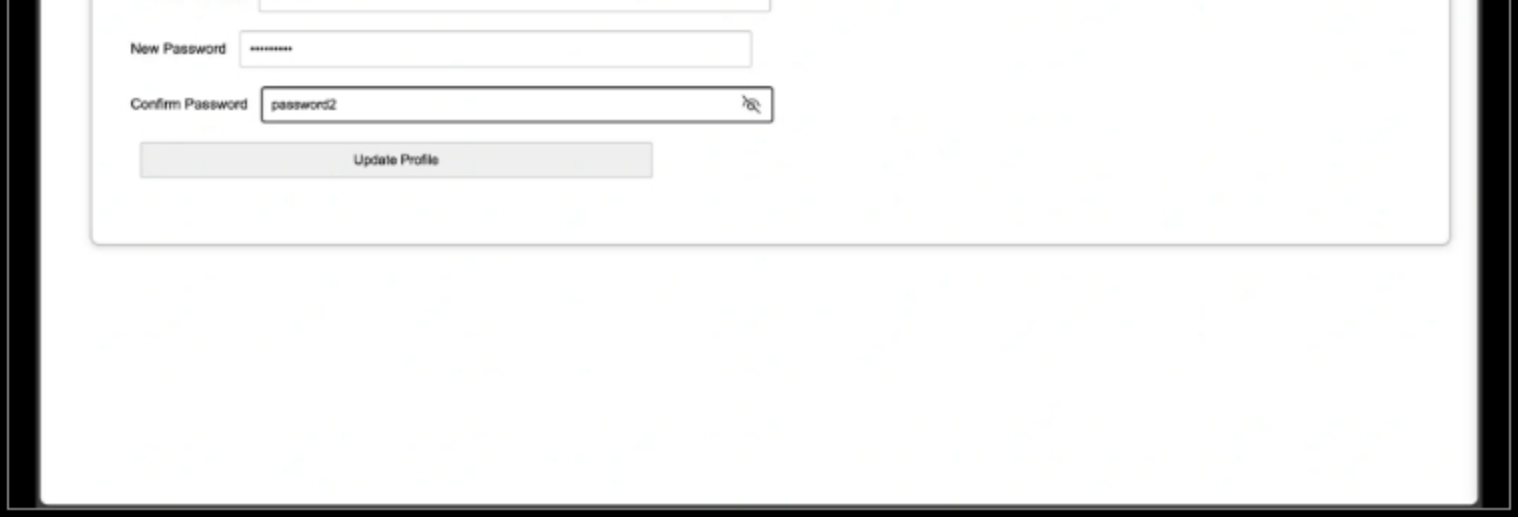


Showing BEFORE email change with success message

Checklist Items (1)

#4 Demonstrate with a before and after of an email change (including success message)

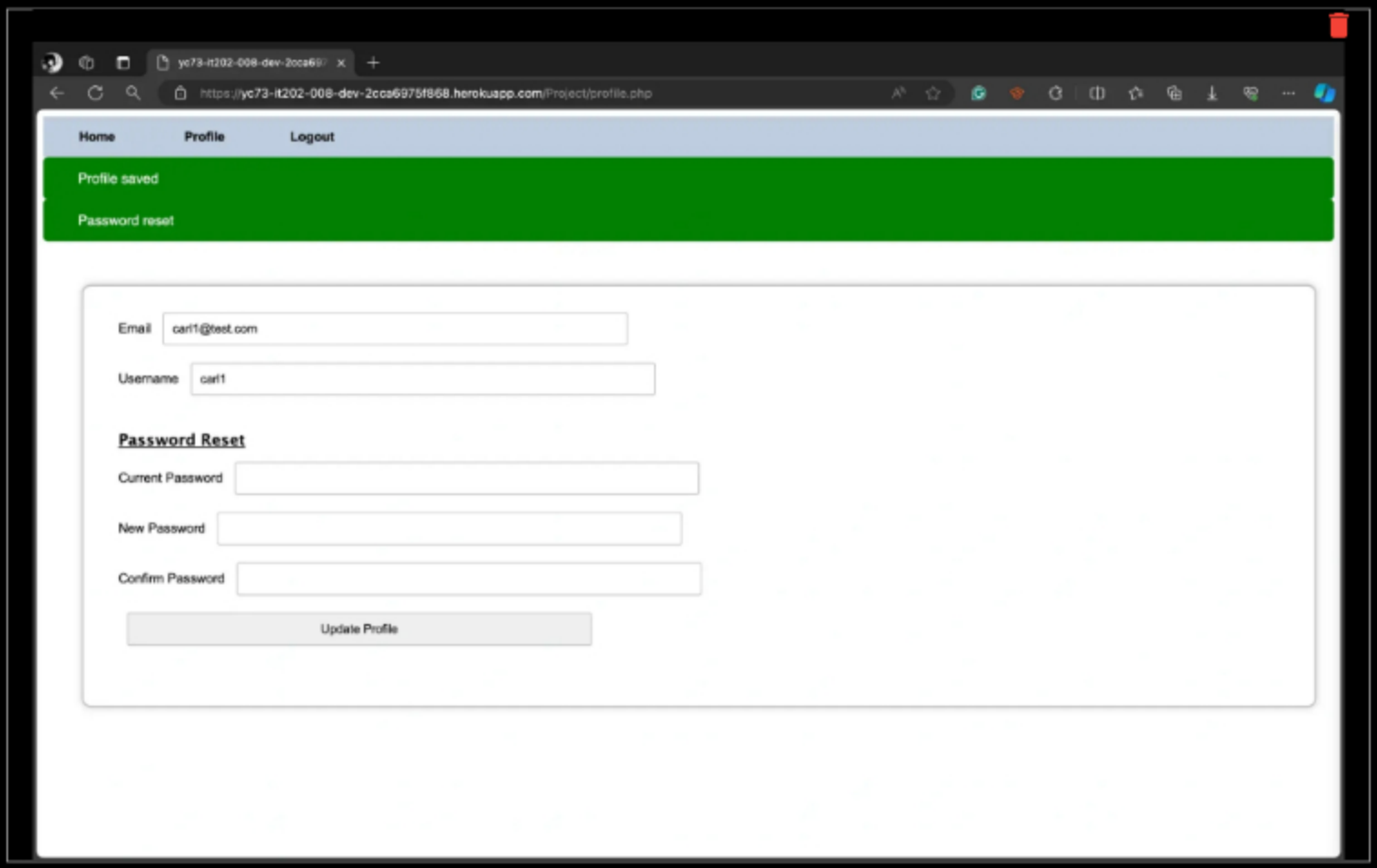




Showing BEFORE updating password

Checklist Items (1)

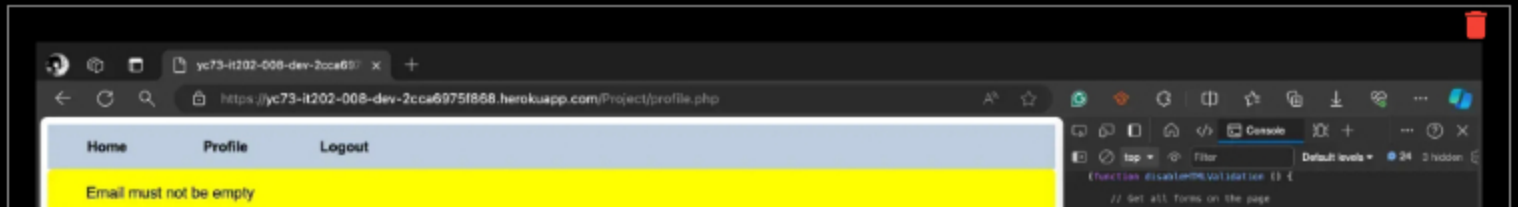
#5 Demonstrate the success message of updating password

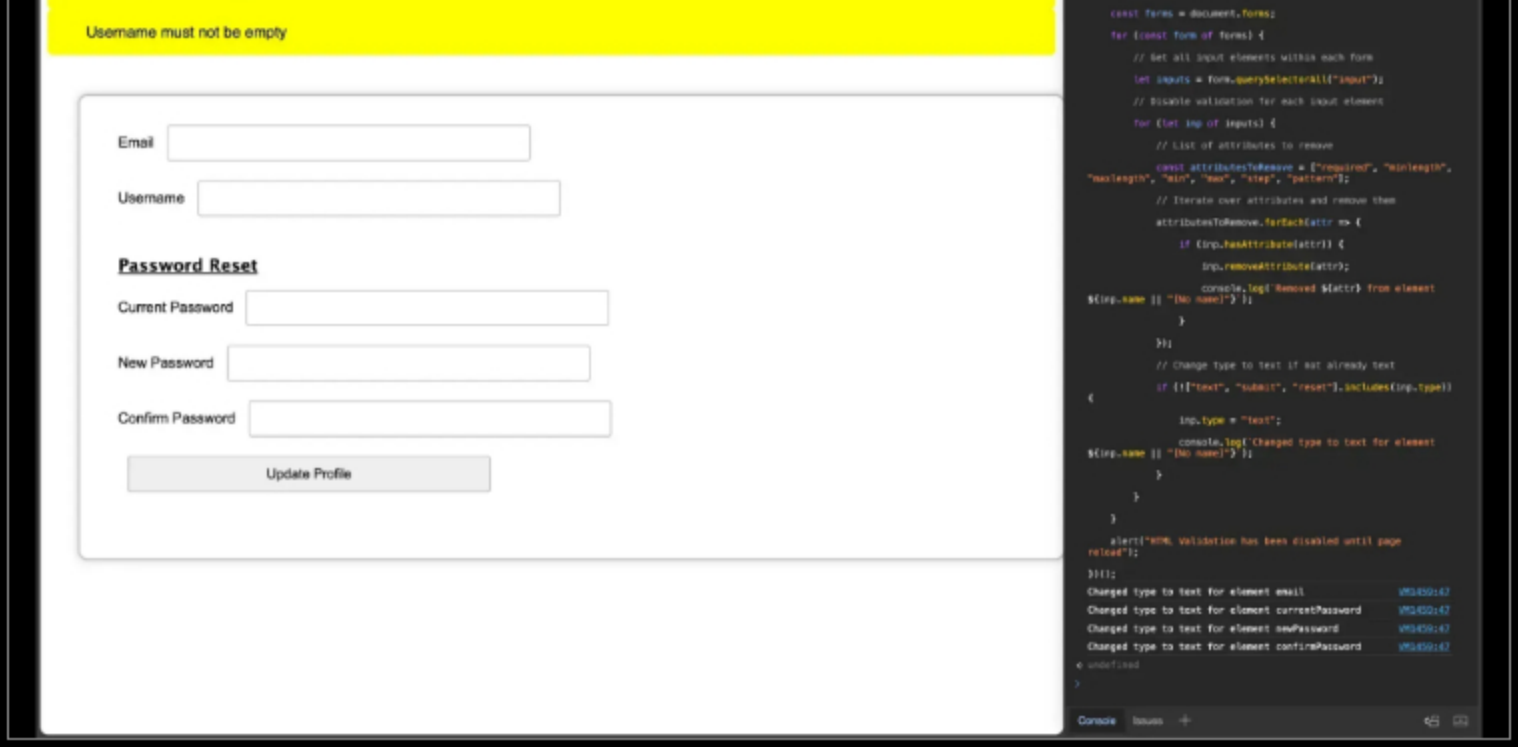


Showing AFTER updating password with success message

Checklist Items (1)

#5 Demonstrate the success message of updating password

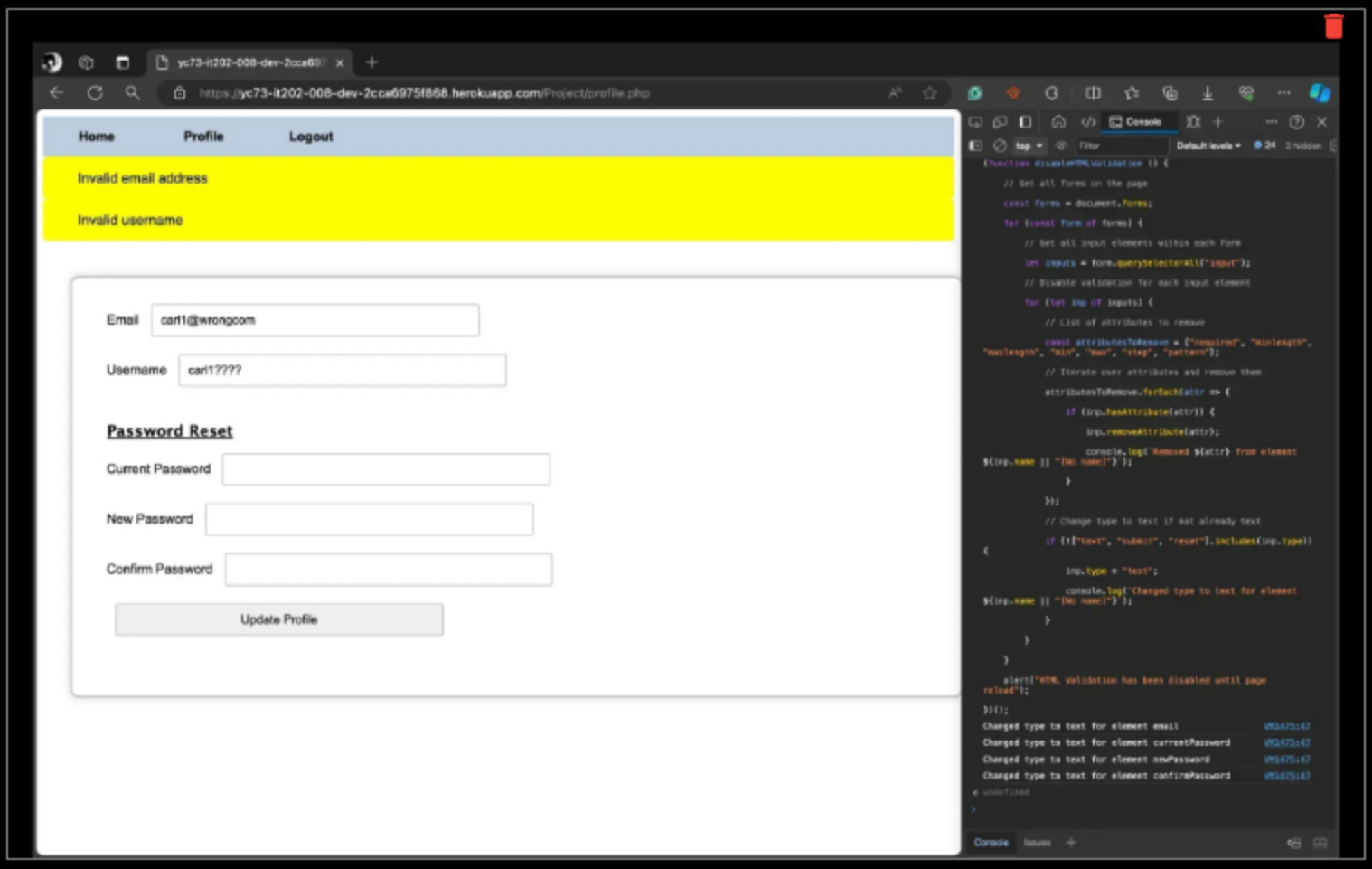




JavaScript email, username => for empty fields

Checklist Items (1)

#6 Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)

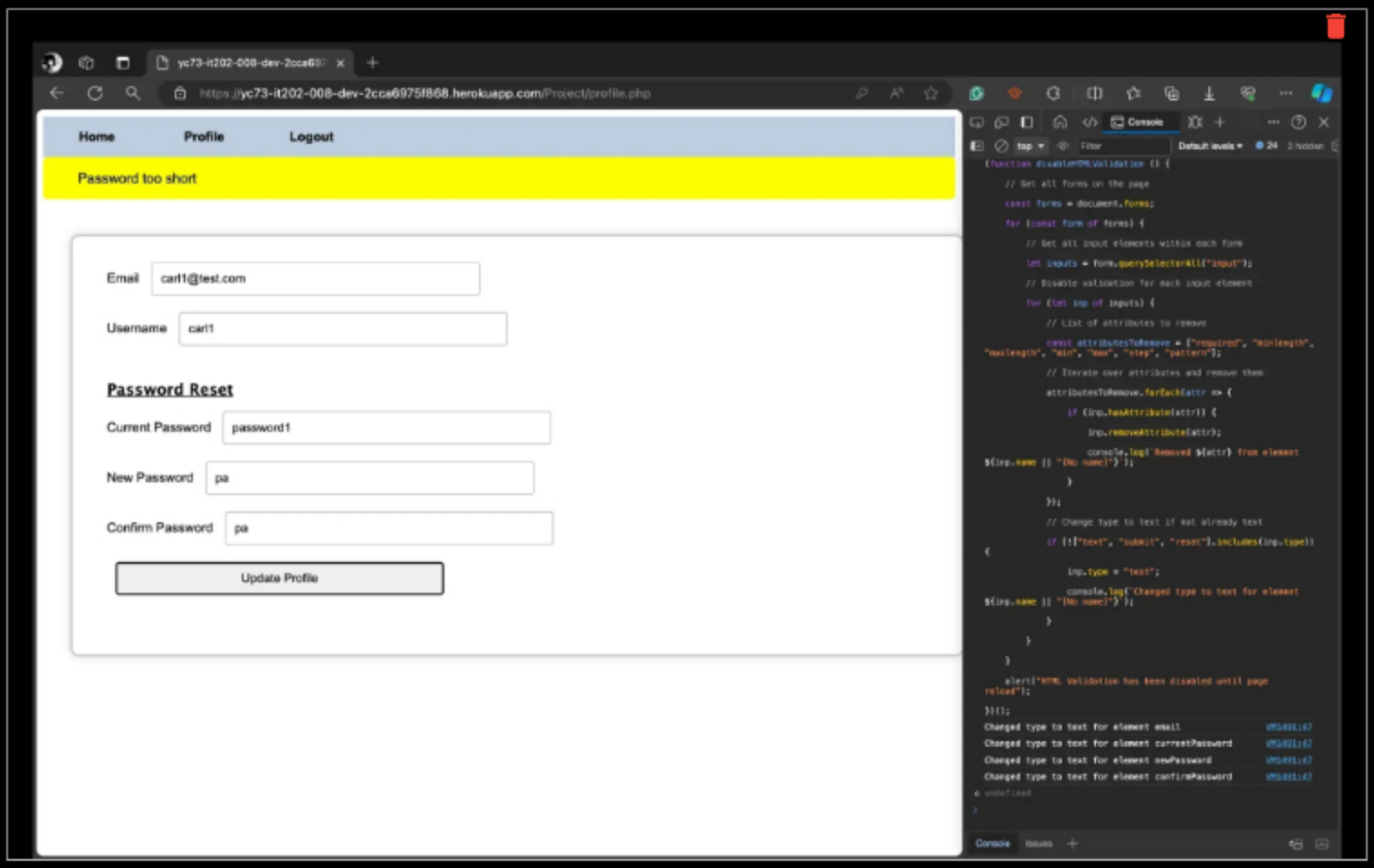


JavaScript email & username validation => (checks invalid) email & username format

Checklist Items (1)

#6 Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)

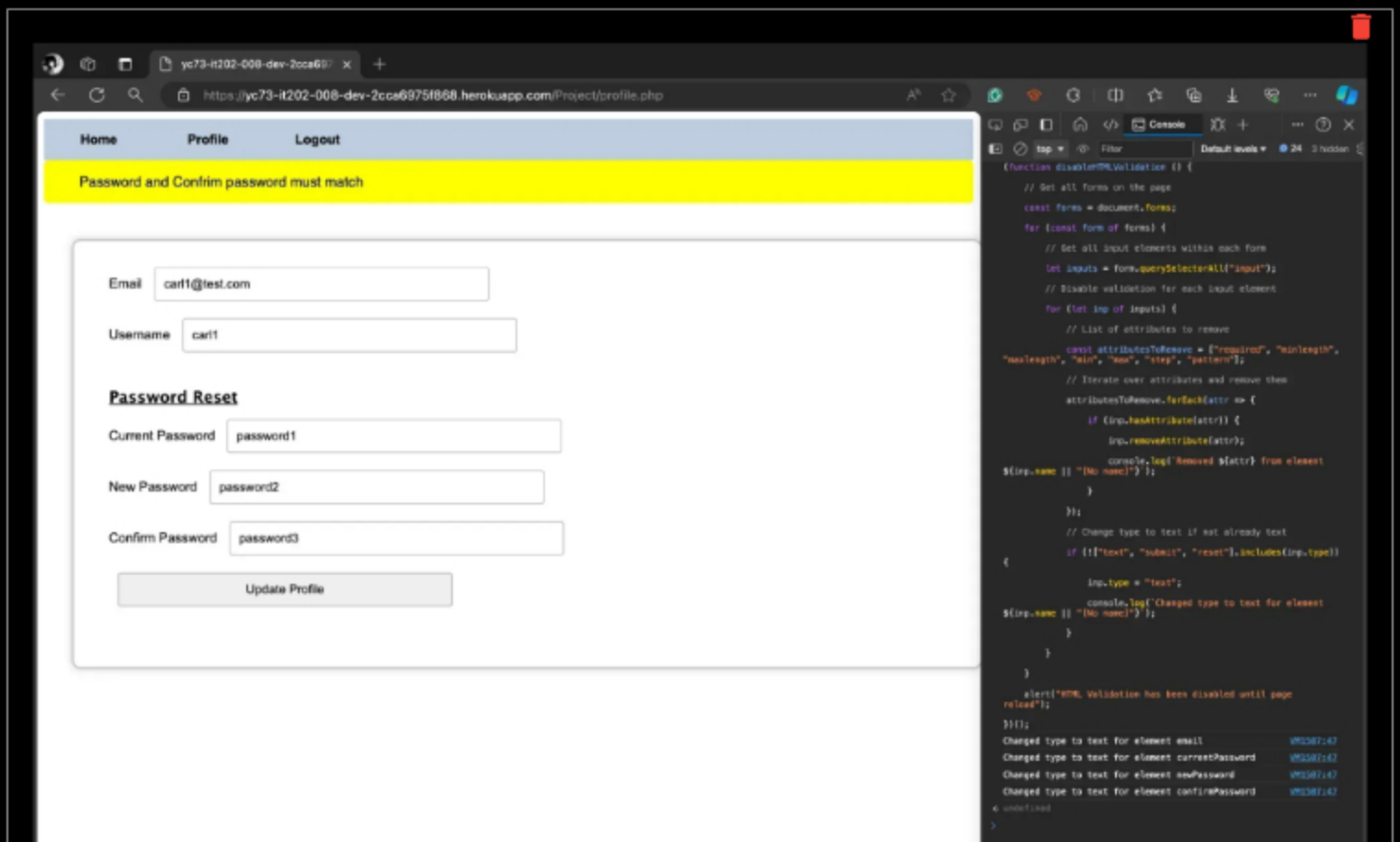
new password matching confirm password)



JavaScript password validation => password length (too short)

Checklist Items (1)

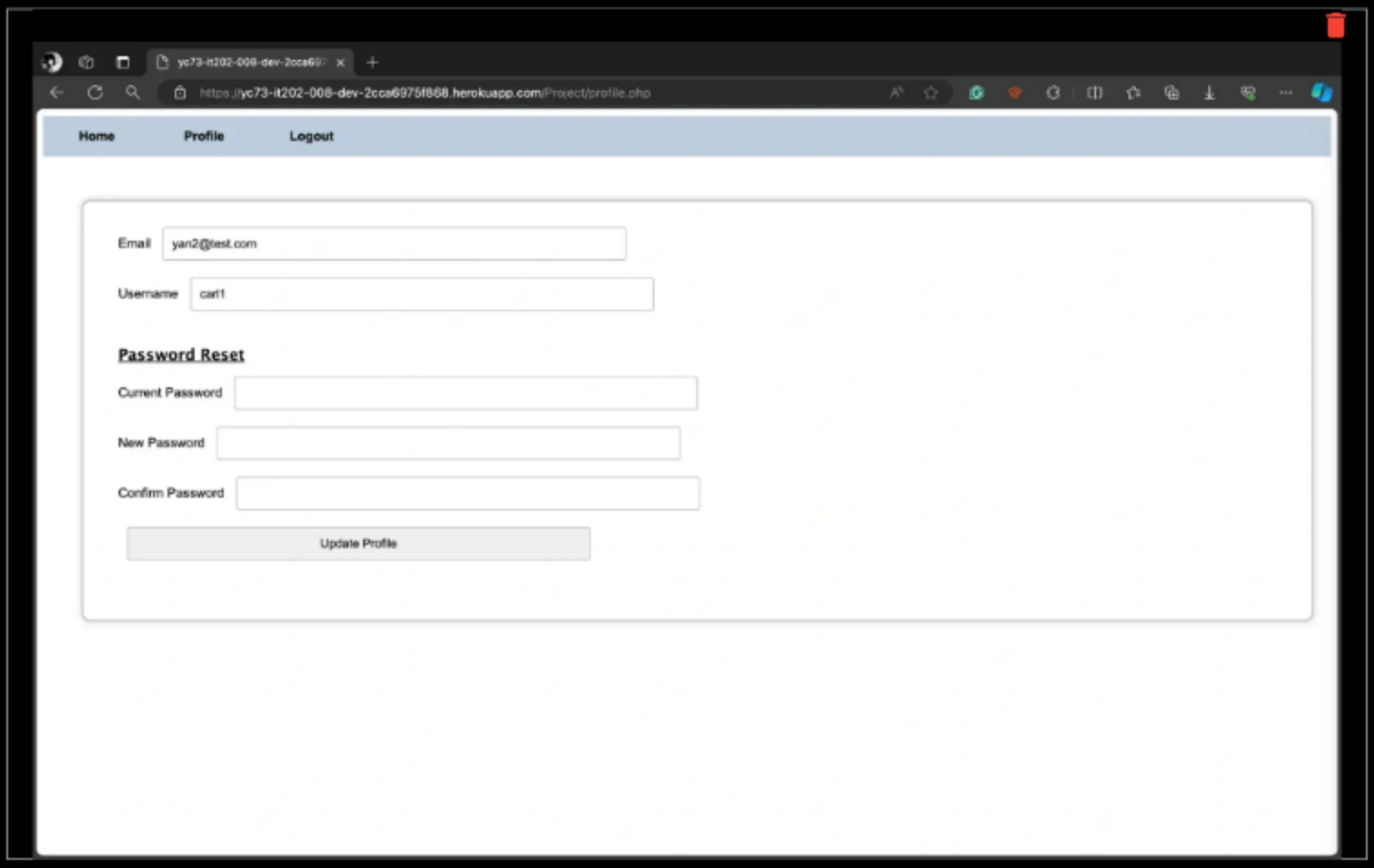
#6 Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)



JavaScript password validation => passwords MUST match

Checklist Items (1)

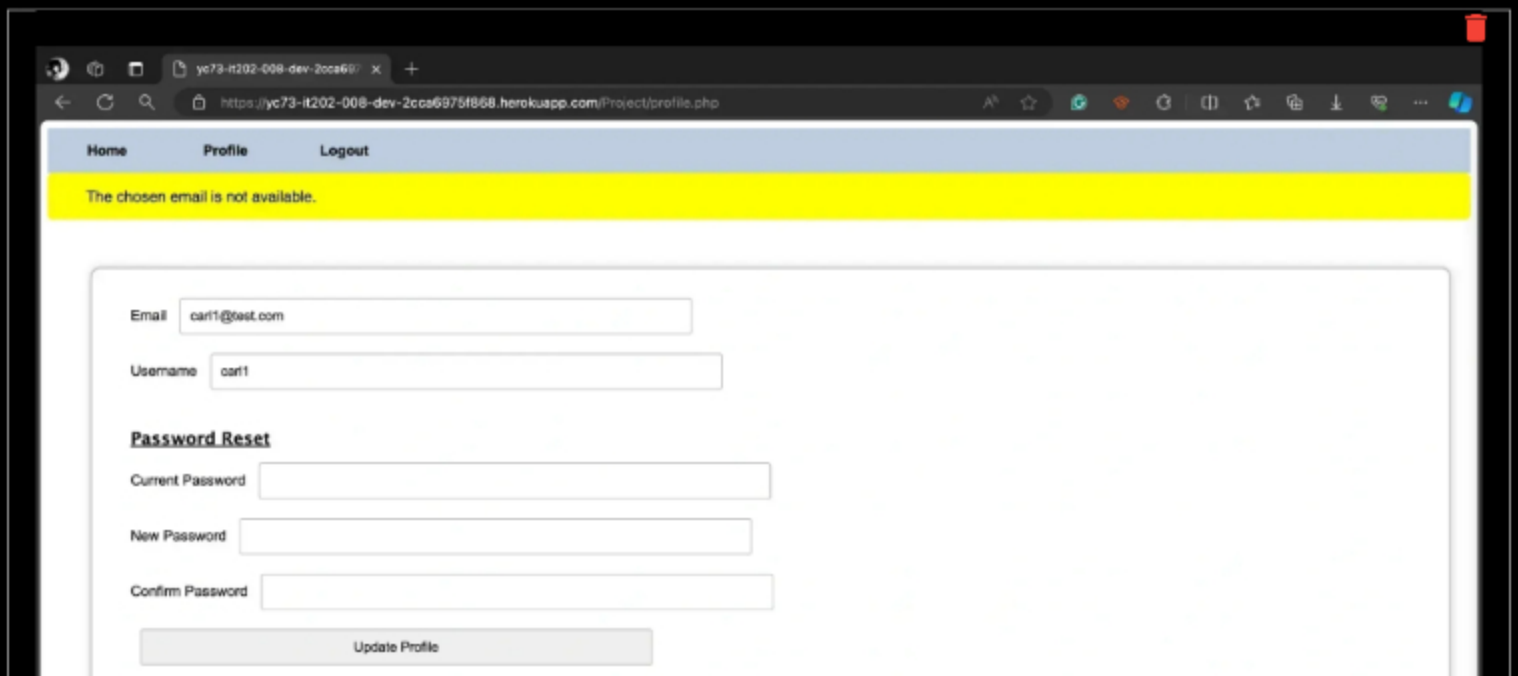
#6 Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)



Showing already taken EMAIL

Checklist Items (1)

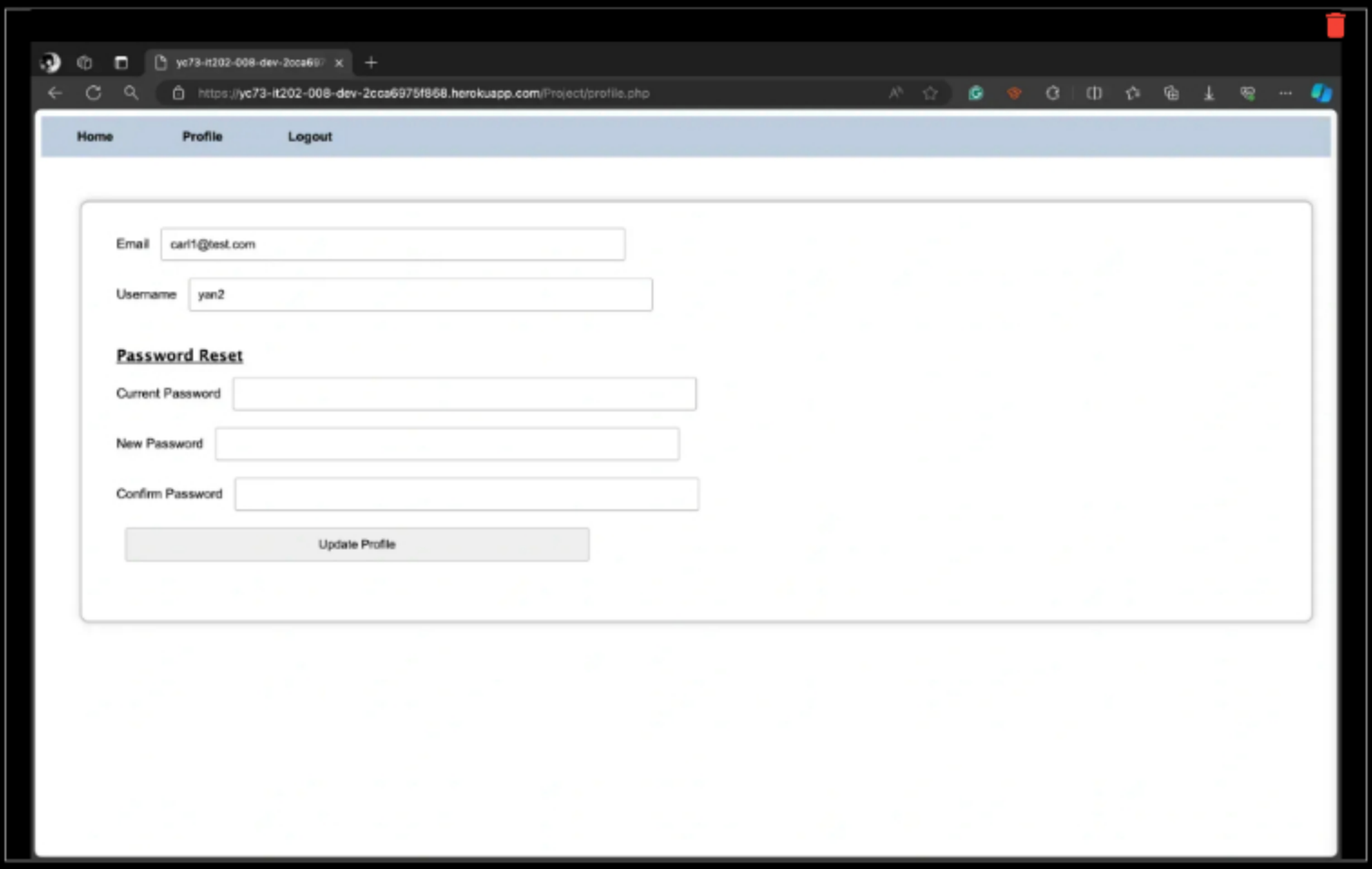
#7 Demonstrate PHP user-friendly validation messages (desired email is already in use, desired username is already in use, current password doesn't match what's in the DB)



Showing PHP validation message when EMAIL is already in use

Checklist Items (1)

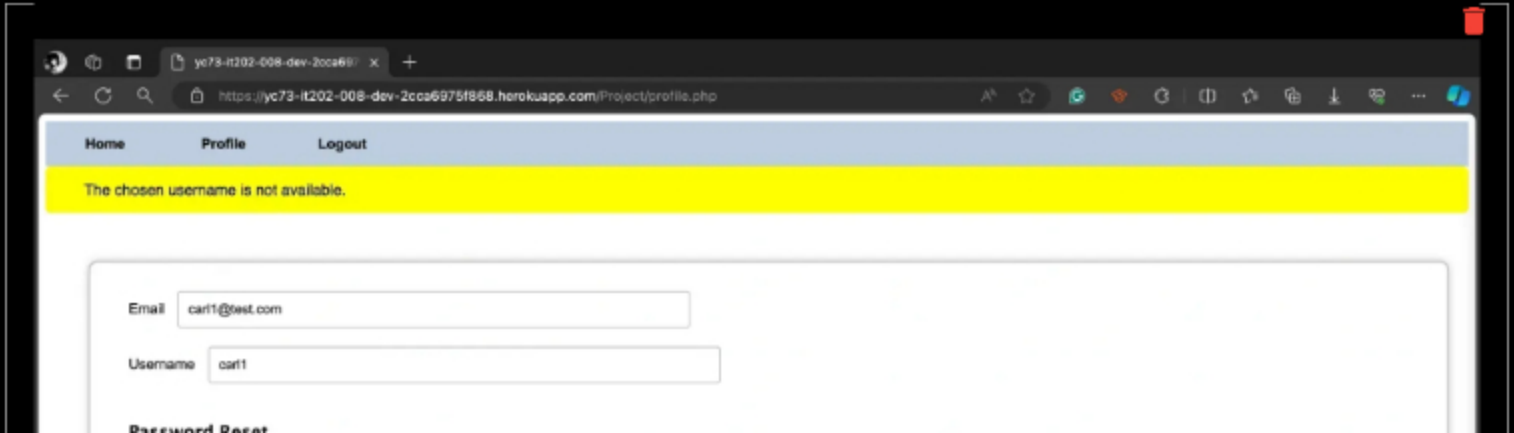
#7 Demonstrate PHP user-friendly validation messages (desired email is already in use, desired username is already in use, current password doesn't match what's in the DB)

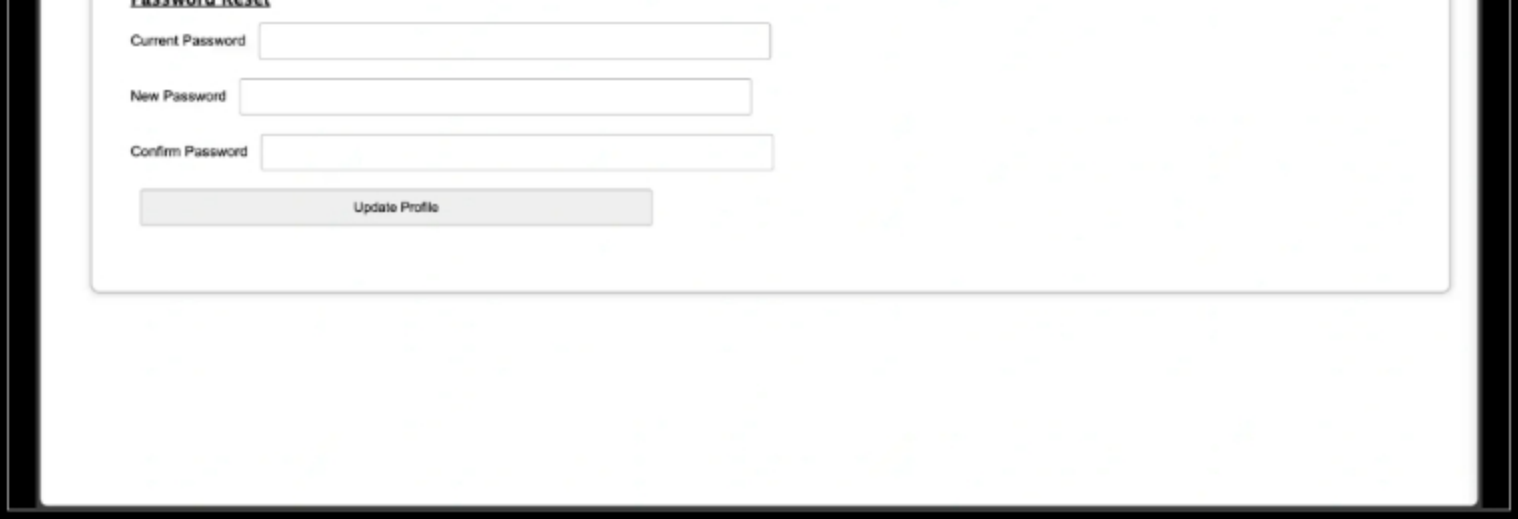


Showing already taken USERNAME

Checklist Items (1)

#7 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)

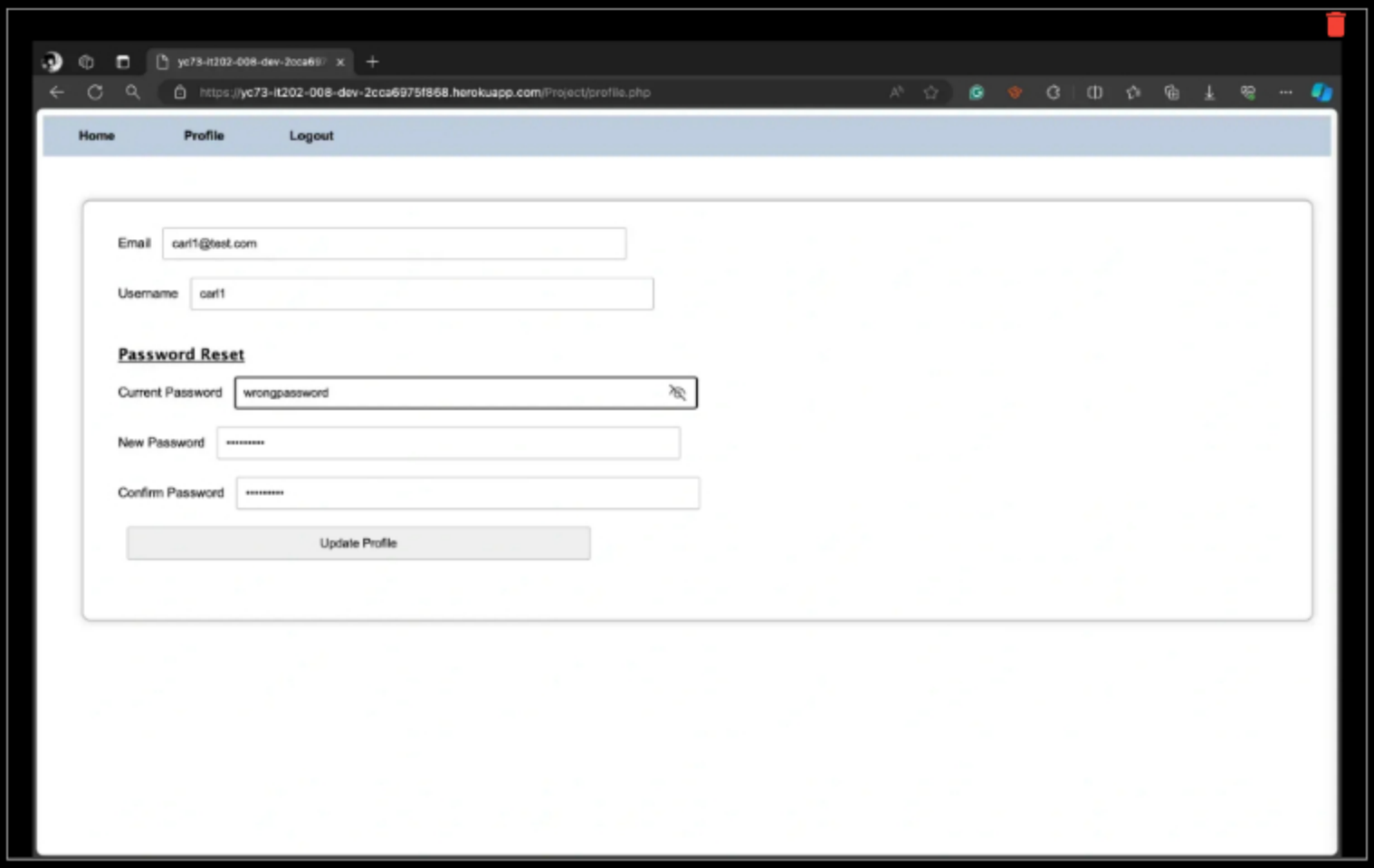




Showing PHP validation message when USERNAME is already in use

Checklist Items (1)

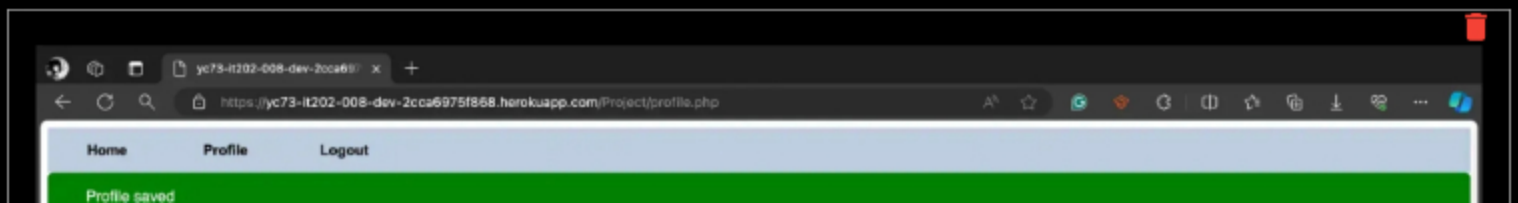
#7 Demonstrate PHP user-friendly validation messages (desired email is already in use, desired username is already in use, current password doesn't match what's in the DB)



Showing incorrect current password

Checklist Items (1)

#7 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)



Showing PHP validation message when PASSWORD does NOT match what's in the DB

Checklist Items (1)

#7 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)

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:

- Upon loading the profile page, a form is displayed to allow users to enter their new email, username, and password (current, new, and confirm). When the user submits that profile form, it will trigger the "onsubmit" attribute and "validate" function. The validation function is the client-side JavaScript validation which checks the email, username, password, and confirm password fields.
  - First, it grabs the values (of what the user entered for the fields) using the "form" object (ex: ("form.email.value", "form.username.value", and "form.pw.value", "form.con.value").
  - Then, it stores the values into corresponding variables (email, user, pw, con).
  - To check if the fields are empty, it looks at the length of the variable with ".length===0".
  - To check the formatting, regex is used. For email, the regex looks for alphanumerics at the start, followed by an @ symbol, another set of alphanumerics, followed by a dot (.), and ends with 2-6 alphabetic characters. For username, the regex looks for alphanumerics, possible _- characters, and a length of 3-16 characters.

    - To check the length of the password, it uses regex to look for a length of 8 characters or more.
    - To check if password and confirm password match, they are compared with strict inequality (!==).
- The "onsubmit" takes a boolean value, so if any of these validations returns a false, then warning messages are displayed to the user, and the form is not submitted. However, if none of the validations fail, it returns true and sends the data as POST.
- When the form is submitted, the server-side php validation first checks if was submitted, and if it has it proceeds to validate the data. The safe echo uses $_POST to retrieve the email and username data, then sets it to variables and begins validating the data.
    - To check if the email format is valid, it must first sanitize it with "sanitize_email()" to remove (filter/trim) illegal characters. Then, it can be validated with "is_valid_email()" which uses FILTER_VALIDATE_EMAIL to check for invalid formatting/characters.
    - To check if the username format is valid, it uses "is_valid_username()", which checks with regex if it has alphanumerics, possibly contains _-, and is between 3-16 characters.
    - If the email or username returns a true for hasError (meaning it failed a validation), the user is flashed an error message.
    - If the email and username return false for hasError (meaning it passed validation), the user's email or username is updated in the "Users" table in the database (using a prepared statement). The SQL statement "UPDATE Users" is executed and sets the email or username for the correct user. When it is successfully updated, the user receives a successful profile saved message.
- Then, the safe echo uses $_POST to retrieve the current, new, and confirm password data. If all are empty (on the form), then the user is not resetting their password. If it is NOT empty (on the form), then the user does want to reset their password.
    - It first checks if the password has a valid length. It uses "is_valid_password()" (in "sanitizers.php"), which uses "strlen" to see if the length is equal to or greater than 8.
    - If it IS a valid length, it proceeds to check if password and confirm password match by comparing them with strict equality (===). If the two do not match, the user receives a message that the new passwords do not match.
    - If they DO match, it retrieves the hashed password from the "Users" table to compare it to the password that was entered. If the entered password does NOT match, the user receives a message saying the current password is invalid.
    - If the entered password DOES match the password stored in the database, the SQL statement "UPDATE Users" is executed and sets the new password for the correct user. Then, it flashes a successful password reset message 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/yaneliii/yc73-it202-008/pull/21

URL #2
https://github.com/yaneliii/yc73-it202-008/pull/22

URL #3

https://github.com/yaneliii/yc73-it202-008/pull/28

Misc (1 pt.)

COLLAPSE

COLLAPSE

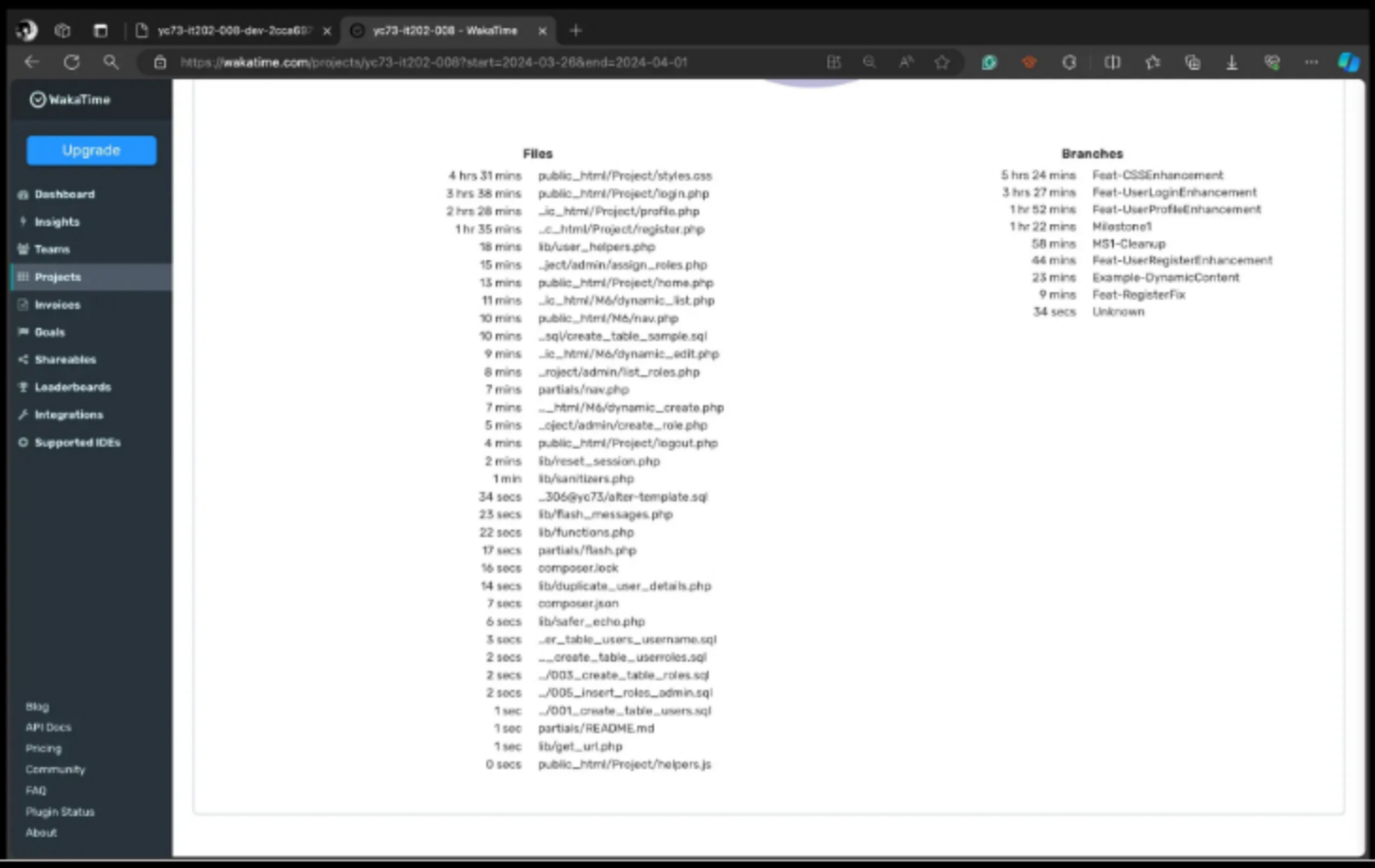
Task #1 - Points: 1

Text: Screenshot of wakatime

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Showing wakatime

COLLAPSE

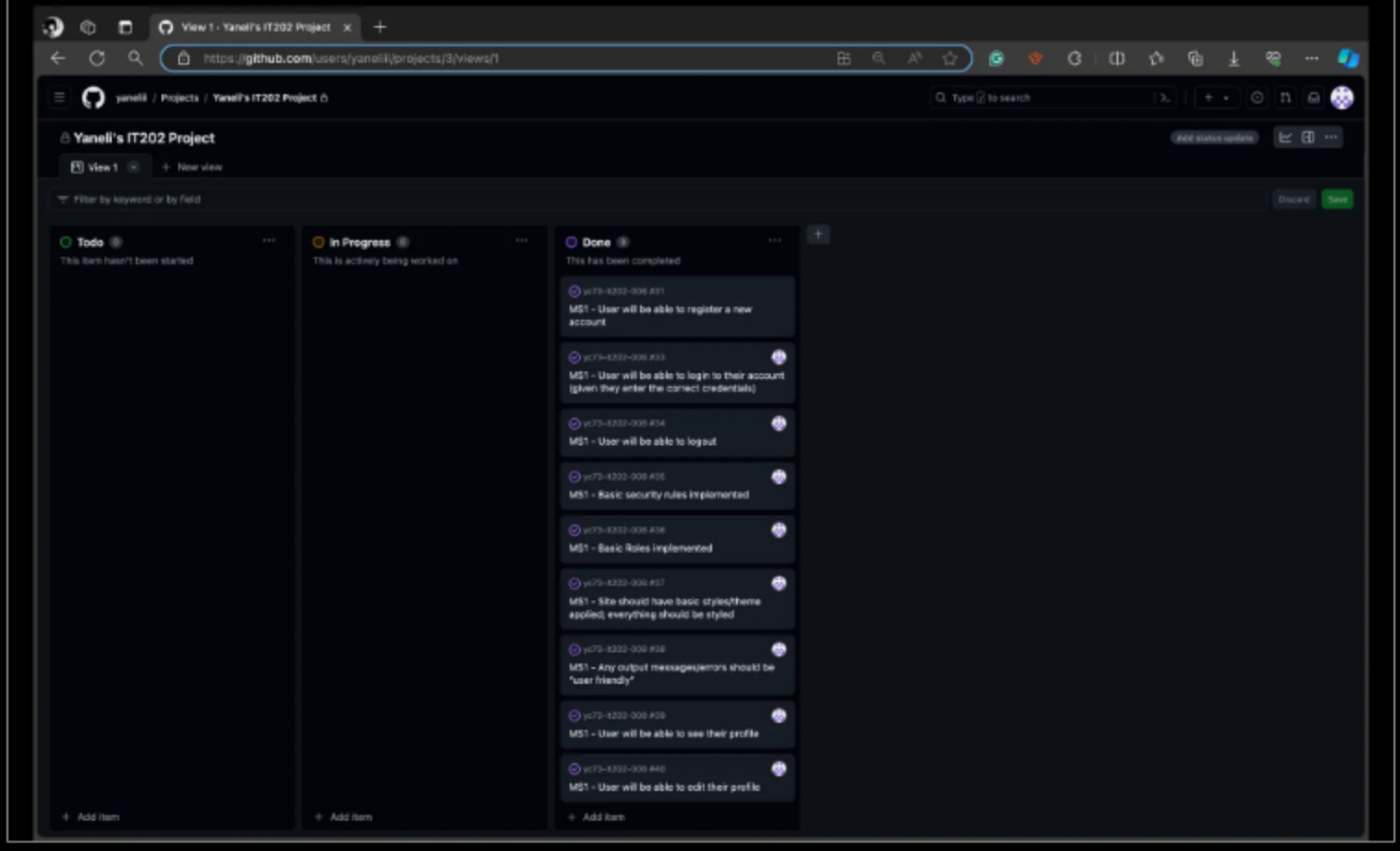
Task #2 - Points: 1

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

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Showing project board (done)

COLLAPSE

Task #3 - Points: 1

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

URL #1

https://github.com/users/yaneliii/projects/3

COLLAPSE

Task #4 - Points: 1

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

URL #1

https://yc73-it202-008-prod-35e9bd30f553.herokuapp.com/Project/login.php

End of Assignment