

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
<input type="checkbox"/> #1	1	Heroku dev url should be present in the address bar
<input type="checkbox"/> #2	1	Should have thoughtful CSS applied
<input type="checkbox"/> #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)
<input type="checkbox"/> #4	1	Demonstrate email already in use message (message text doesn't need to be exact, but should be clear)
<input type="checkbox"/> #5	1	Demonstrate username already in use message (message text doesn't need to be exact, but should be clear)
<input type="checkbox"/> #6	1	Demonstrate user-friendly message of new account being created

Task Screenshots:

Gallery Style: Large View

Small

Medium

Large

The screenshot shows a browser window with a registration form. The URL is <https://yc73-lt202-008-dev-2cca698.herokuapp.com/project/register.php>. The developer console on the right side shows a log of validation changes made to the input fields.

```

const attributesToRemove = ["required", "minlength", "maxlength", "max", "step", "pattern"];
// iterate over attributes and remove them
attributesToRemove.forEach(attr => {
  if (inp.hasAttribute(attr)) {
    inp.removeAttribute(attr);
    console.log(`Removed ${attr} from element ${inp.name} || "[No name]"`);
  }
})
// Change type to text if not already text
if (!["text", "submit", "reset"].includes(inp.type)) {
  inp.type = "text";
  console.log(`Changed type to text for element ${inp.name} || "[No name]"`);
}
}
alert("HTML Validation has been disabled until page reload");
})(());
Removed required from element email
Changed type to text for element email
Removed required from element username
Removed minlength from element username
Removed required from element password
Removed minlength from element password
Changed type to text for element password
Removed required from element confirm
Removed minlength from element confirm
Changed type to text for element confirm
6 undefined
>

```

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)

The screenshot shows a registration form on a web page. The form has four fields: Email, Username, Password, and Confirm. The 'Email' field contains 'carl@wrong'. A yellow header bar displays the error message 'Invalid email address'. The 'Register' button is green.

```
const attributesToRemove = ["required", "maxLength", "min", "max", "step", "pattern"];
```

```
// Iterate over attributes and remove them
attributesToRemove.forEach(attr => {
  if (inp.hasAttribute(attr)) {
    inp.removeAttribute(attr);
    console.log(`Removed ${attr} from element ${inp.name} || "[No name]"`);
  }
});
```

```
// Change type to text if not already text
if (!["text", "submit", "reset"].includes(inp.type)) {
  inp.type = "text";
  console.log(`Changed type to text for element ${inp.name} || "[No name]"`);
}
```

```
alert("HTML Validation has been disabled until page reload");
```

```
)();
```

```
Removed required from element email 1982:35
Changed type to text for element email 1982:47
Removed required from element username 1982:35
Removed maxlen from element username 1982:35
Removed required from element password 1982:35
Removed minlength from element password 1982:35
Changed type to text for element password 1982:47
Removed required from element confirm 1982:35
Removed maxlen from element confirm 1982:35
Changed type to text for element confirm 1982:47
> undefined
```

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)

The screenshot shows a registration form on a web page. The form has four fields: Email, Username, Password, and Confirm. The 'Username' field contains 'ca'. A yellow header bar displays the error message 'Invalid username'. The 'Register' button is green.

```
const attributesToRemove = ["required", "maxLength", "min", "max", "step", "pattern"];
```

```
// Iterate over attributes and remove them
attributesToRemove.forEach(attr => {
  if (inp.hasAttribute(attr)) {
    inp.removeAttribute(attr);
    console.log(`Removed ${attr} from element ${inp.name} || "[No name]"`);
  }
});
```

```
// Change type to text if not already text
if (!["text", "submit", "reset"].includes(inp.type)) {
  inp.type = "text";
  console.log(`Changed type to text for element ${inp.name} || "[No name]"`);
}
```

```
alert("HTML Validation has been disabled until page reload");
```

```
)();
```

```
Removed required from element email 1988:35
Changed type to text for element email 1988:47
Removed required from element username 1988:35
Removed maxlen from element username 1988:35
Removed required from element password 1988:35
Removed minlength from element password 1988:35
Changed type to text for element password 1988:47
Removed required from element confirm 1988:35
Removed maxlen from element confirm 1988:35
Changed type to text for element confirm 1988:47
> undefined
```

```
Changed type to text for element password    MM158:47
Removed required from element confirm        MM158:35
Removed minlength from element confirm       MM158:35
Changed type to text for element confirm     MM158:47
↳ undefined
>
```

Console Issues +

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

The screenshot shows a registration form on a web page. The form has fields for Email, Username, Password, and Confirm. The 'Email' field contains 'carl@test.com', 'Username' contains 'carl', 'Password' contains 'pa', and 'Confirm' also contains 'pa'. A yellow error bar at the top says 'Password too short'. The browser's developer tools are open, showing the JavaScript code responsible for the validation. The code iterates over attributes and removes them if they are not needed. It then changes the type of input elements to 'text' if they were not already. Finally, it reloads the page.

```
const attributesToRemove = ["required", "maxlength", "min", "max", "step", "pattern"];  
  
// Iterate over attributes and remove them  
attributesToRemove.forEach(attr => {  
  if (inp.hasAttribute(attr)) {  
    inp.removeAttribute(attr);  
  
    console.log(`Removed ${attr} from element ${inp.name} || ${[No name]}`);  
  }  
});  
  
// Change type to text if not already text  
if (!["text", "submit", "reset"].includes(inp.type)) {  
  inp.type = "text";  
  
  console.log(`Changed type to text for element ${inp.name} || ${[No name]}`);  
}  
  
alert("HTML Validation has been disabled until page reload");  
31()
```

Removed required from element email MM158:35
Changed type to text for element email MM158:47
Removed required from element username MM158:35
Removed minlength from element username MM158:35
Removed required from element password MM158:35
Removed minlength from element password MM158:35
Changed type to text for element password MM158:47
Removed required from element confirm MM158:35
Removed minlength from element confirm MM158:35
Changed type to text for element confirm MM158:47
↳ undefined
>

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

The screenshot shows a registration form on a web page. The form has fields for Email, Username, Password, and Confirm. The 'Email' field contains 'carl@test.com', 'Username' contains 'carl', 'Password' contains 'password1', and 'Confirm' contains 'password'. A yellow error bar at the top says 'Passwords must match'. The browser's developer tools are open, showing the JavaScript code responsible for the validation. The code iterates over attributes and removes them if they are not needed. It then changes the type of input elements to 'text' if they were not already. Finally, it reloads the page.

```
const attributesToRemove = ["required", "maxlength", "min", "max", "step", "pattern"];  
  
// Iterate over attributes and remove them  
attributesToRemove.forEach(attr => {  
  if (inp.hasAttribute(attr)) {  
    inp.removeAttribute(attr);  
  
    console.log(`Removed ${attr} from element ${inp.name} || ${[No name]}`);  
  }  
});  
  
// Change type to text if not already text  
if (!["text", "submit", "reset"].includes(inp.type)) {  
  inp.type = "text";  
  
  console.log(`Changed type to text for element ${inp.name} || ${[No name]}`);  
}  
  
alert("HTML Validation has been disabled until page reload");  
31()
```

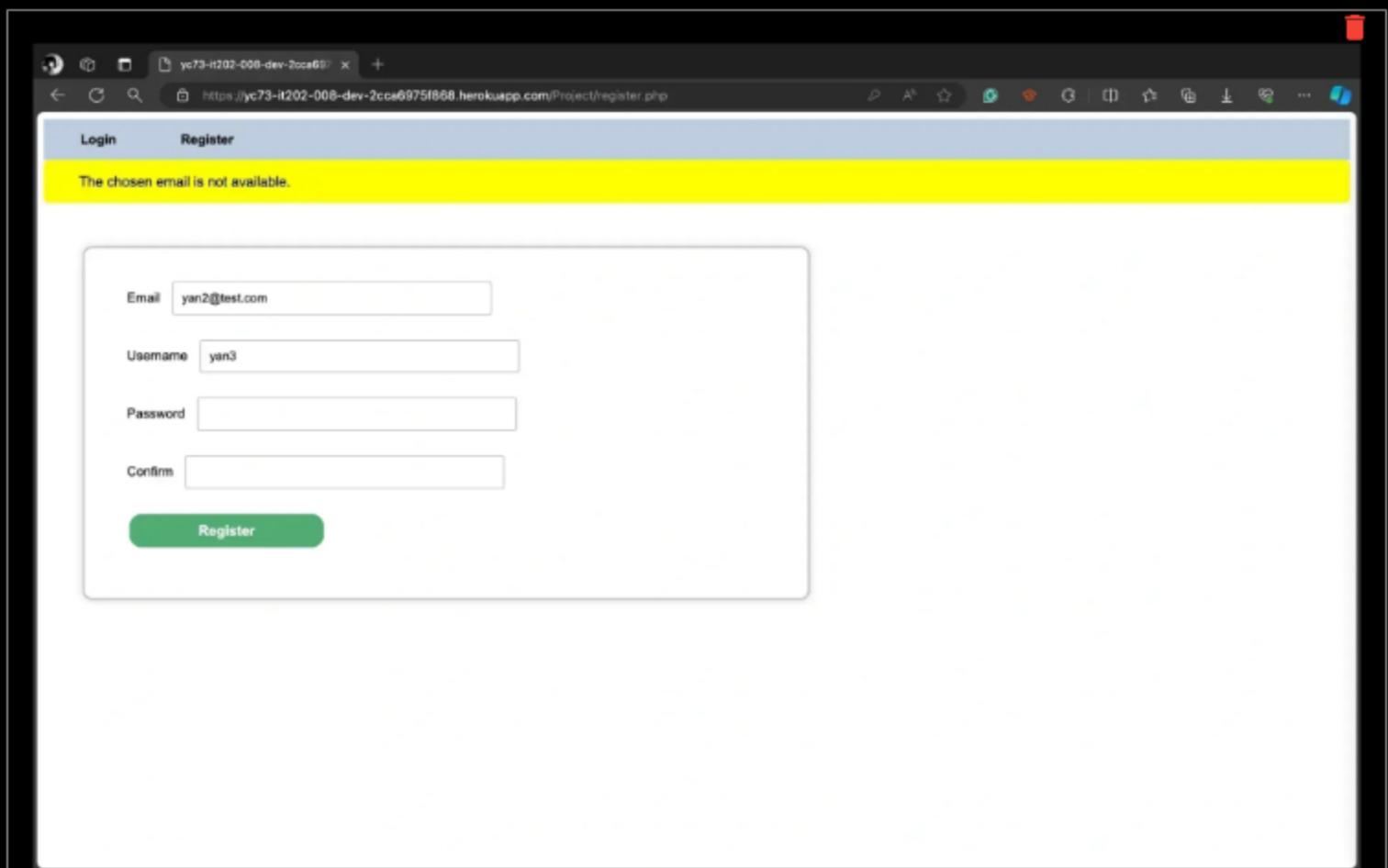
Removed required from element email MM158:35
Changed type to text for element email MM158:47
Removed required from element username MM158:35
Removed minlength from element username MM158:35
Removed required from element password MM158:35
Removed minlength from element password MM158:35
Changed type to text for element password MM158:47
Removed required from element confirm MM158:35
Removed minlength from element confirm MM158:35
Changed type to text for element confirm MM158:47
↳ undefined
>



JavaScript password validation => passwords **MUST** match

Checklist Items (1)

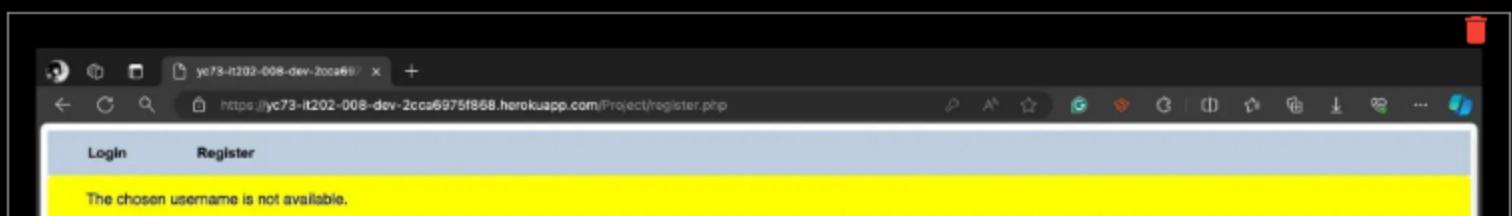
#3 Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required)



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)



Email

Username

Password

Confirm

Register

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)

A screenshot of a web browser window displaying a registration form. The URL in the address bar is `https://yc73-il202-008-dev-2cca6975f868.herokuapp.com/project/register.php`. The browser interface includes standard navigation buttons and a search bar.

The page has a header with "Login" and "Register" buttons. A green horizontal bar displays the message "Successfully registered!". Below this, there is a registration form with fields for Email, Username, Password, and Confirm, each with an input field. A green "Register" button is at the bottom of the form.

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 fields

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>
</main>

```

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)

<input checked="" type="checkbox"/> #2	1	Show the PHP validations (include any lib content)
<input checked="" type="checkbox"/> #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>
/* sc73 4/1/23 */
function validateForm() {
    // TODO: 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_.-){0,(a-zA-Z0-9_.-){0,(a-zA-Z){2,6}}$)/;
        if (!email_pattern.test(email)) {
            flash("Invalid email address", "warning");
            is_valid = false;
        }
    }
    if (user.length === 0) {
        flash("Username must not be empty", "warning");
        is_valid = false;
    }
    else {
        const user_pattern = /^[a-zA-Z0-9_.-]{3,16}$/;
        if (!user_pattern.test(user)) {
            flash("Invalid username", "warning");
            is_valid = false;
        }
    }
    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;
        }
    }
    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 validateForm()
</script>
```

Showing JavaScript validations

Checklist Items (1)

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

```
<?php
/* sc73 4/1/23 */      You, 1 second ago + Uncommitted changes
// 2: add PHP Code
if (isset($_POST["email"]) && isset($_POST["password"]) && isset($_POST["confirm"]) && isset($_POST["username"])) {
    $email = $_POST["email"];
    $password = $_POST["password"];
    $confirm = $_POST["confirm"];
    $username = $_POST["username"];
}

// 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, A-Z, _, 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;
}
```

```

if (!is_valid_password($password)) {
    $hasError = true;
}

if (strlen($password) > 0 && $password !== $confirm) {
    $flash["Passwords must match", "danger"];
    $hasError = true;
}

if ($hasError) {
    // [REDACTED]
    $hash = password_hash($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);
    }
}
} <- #129-108 IF ($hasError)
} <- #88-141 IF (isset($_POST["email"])) && isset($_POST["password"]) && is...
To:

```

Showing PHP validations

Checklist Items (1)

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

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
<input checked="" type="checkbox"/> #1	1	Password should be hashed
<input checked="" type="checkbox"/> #2	1	Should have email, password, username (unique), created, modified, and id fields
<input checked="" type="checkbox"/> #3	1	Ensure left panel or database name is present (should contain your ucid)

Task Screenshots:

Gallery Style: Large View

Small Medium Large

The screenshot shows the MySQL Workbench interface with the 'yc73-it202-008' database selected. The 'Users' table is open, displaying the following data:

	id	email	password	created	modified	username
1	1	yan@test.com	\$2y\$10\$yNWBYrrM.MeeW5zI	2024-03-16 03:20:14	2024-03-17 17:27:27	yan
2	2	yan2@test.com	\$2y\$10\$MTsEmdu1CyOz6jzf	2024-03-17 19:05:39	2024-04-01 05:11:22	yan2
3	5	gabi@test.com	\$2y\$10\$XYPlbKZnaSzpCvC	2024-03-21 02:25:46	2024-03-21 02:25:46	gab
4	6	bob@test.com	\$2y\$10\$pNy8IJNT1TnmzbWk	2024-03-31 18:18:32	2024-03-31 18:18:32	bob
5	12	carl@test.com	\$2y\$10\$e5TbdYbYthoKzrQj	2024-04-01 06:20:51	2024-04-01 06:20:51	carl

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

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.

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.)

Task #1 - Points: 1

Text: Screenshot of form on website page

Checklist

*The checkboxes are for your own tracking

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

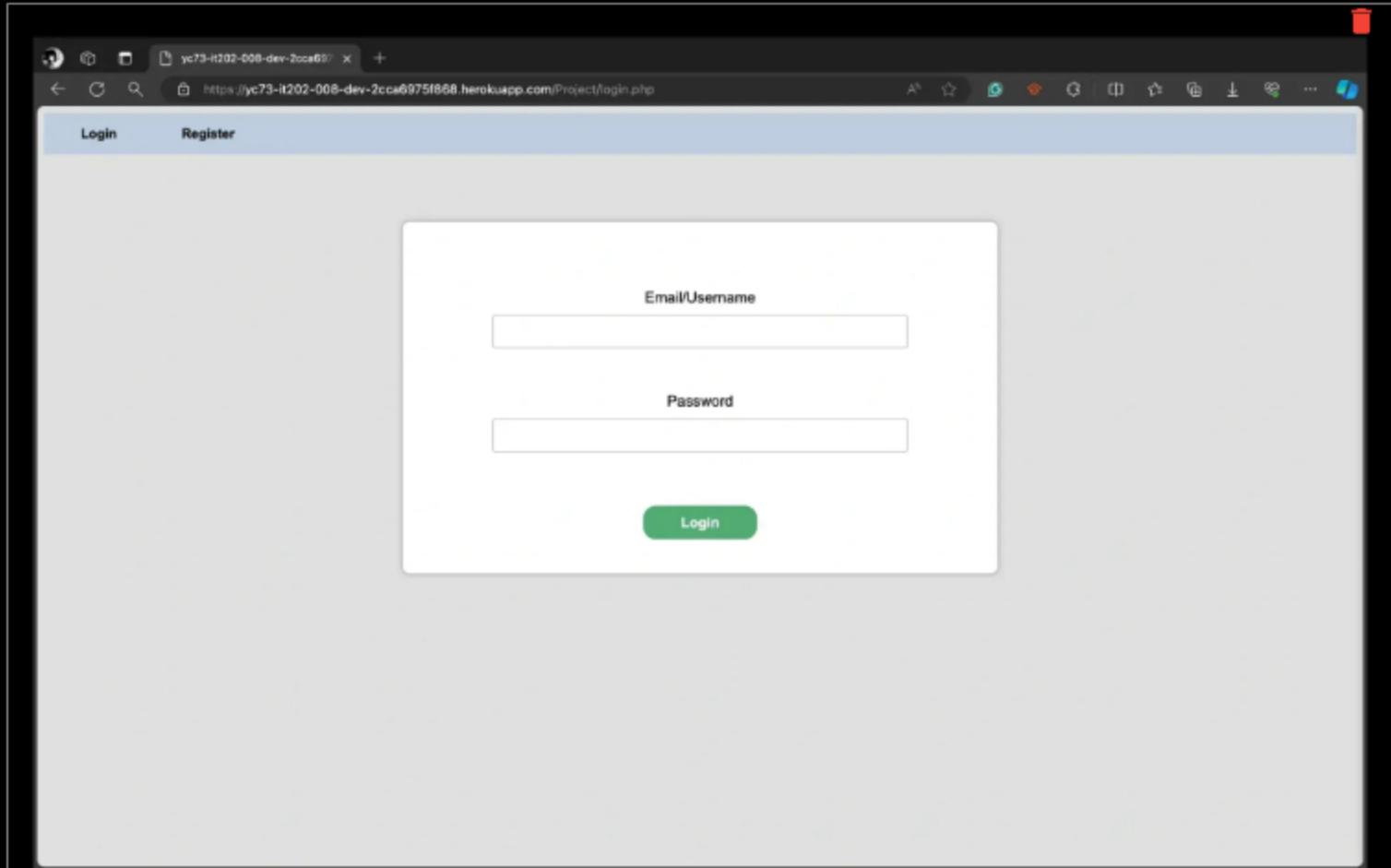
Task Screenshots:

Gallery Style: Large View

Small

Medium

Large

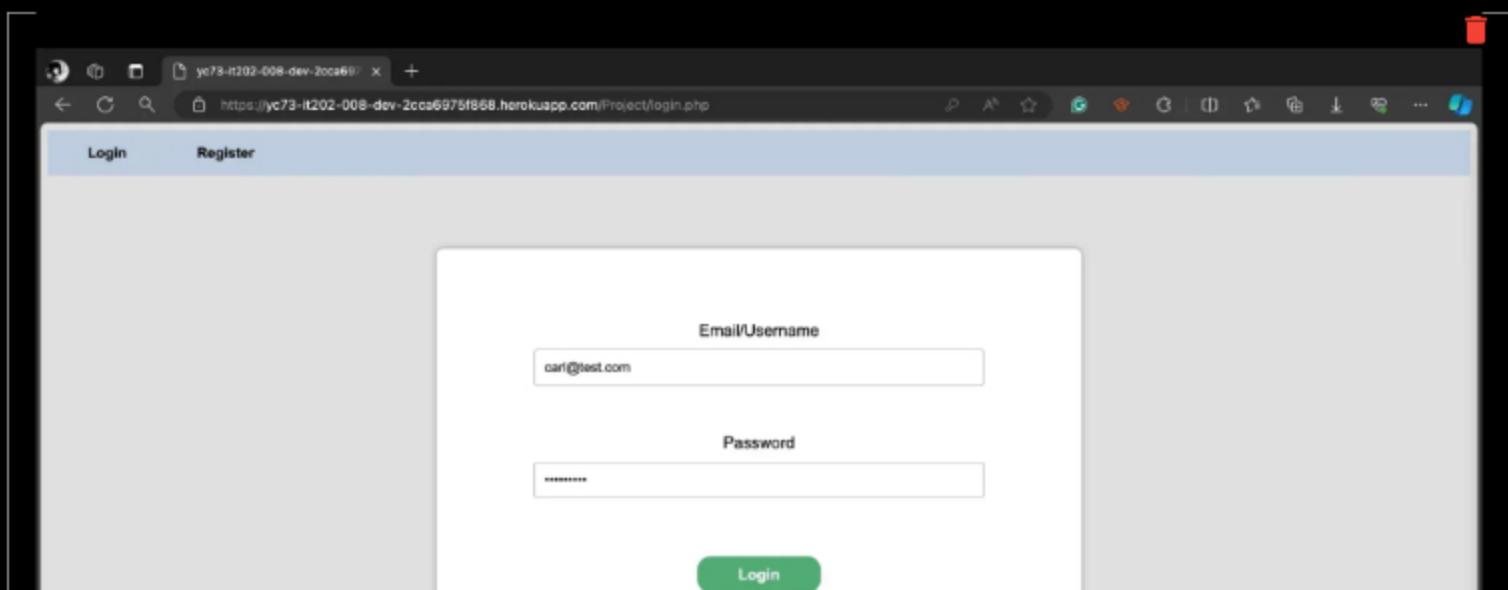


A screenshot of a web browser window displaying a login form. The browser's address bar shows the URL <https://yc73-it202-008-dev-2cca6975f868.herokuapp.com/project/login.php>. The page has a light blue header bar with 'Login' and 'Register' buttons. Below the header is a large white rectangular input field containing two text input boxes labeled 'Email/Username' and 'Password', and a green rounded rectangular 'Login' button at the bottom.

Showing (blank) login page with CSS

Checklist Items (1)

#2 Should have thoughtful CSS applied

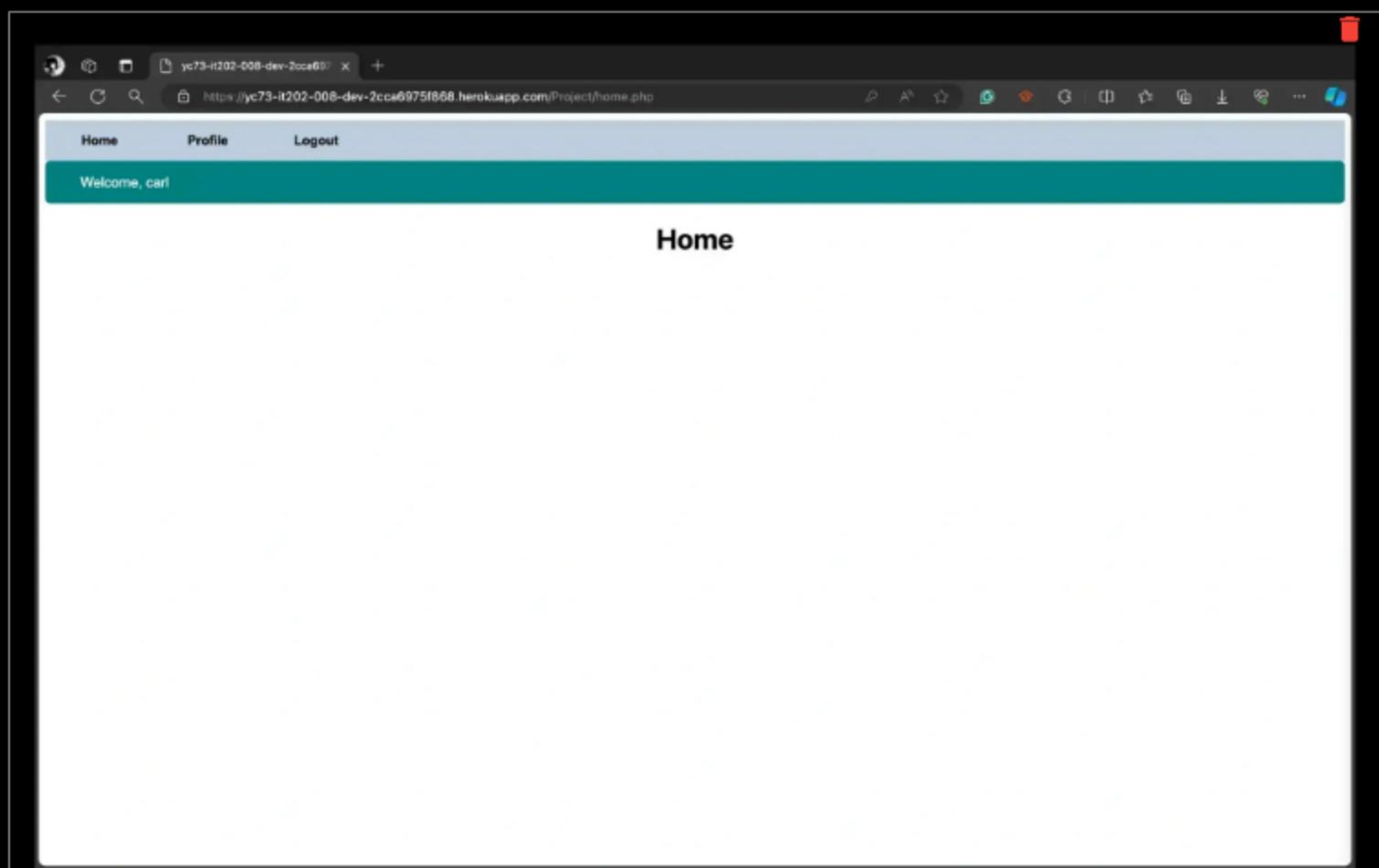


A screenshot of a web browser window displaying a login form. The browser's address bar shows the URL <https://yc73-it202-008-dev-2cca6975f868.herokuapp.com/project/login.php>. The page has a light blue header bar with 'Login' and 'Register' buttons. Below the header is a large white rectangular input field containing two text input boxes. The top input box is labeled 'Email/Username' and contains the placeholder text 'carl@test.com'. The bottom input box is labeled 'Password' and contains the placeholder text '*****'. A green rounded rectangular 'Login' button is at the bottom.

Showing correct EMAIL data to login

Checklist Items (1)

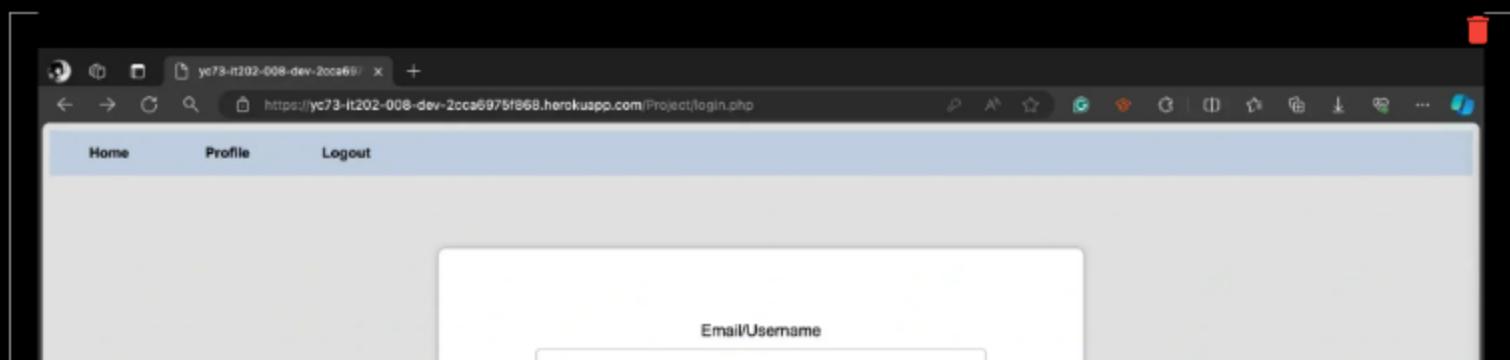
#4 Include correct data where email is used to login

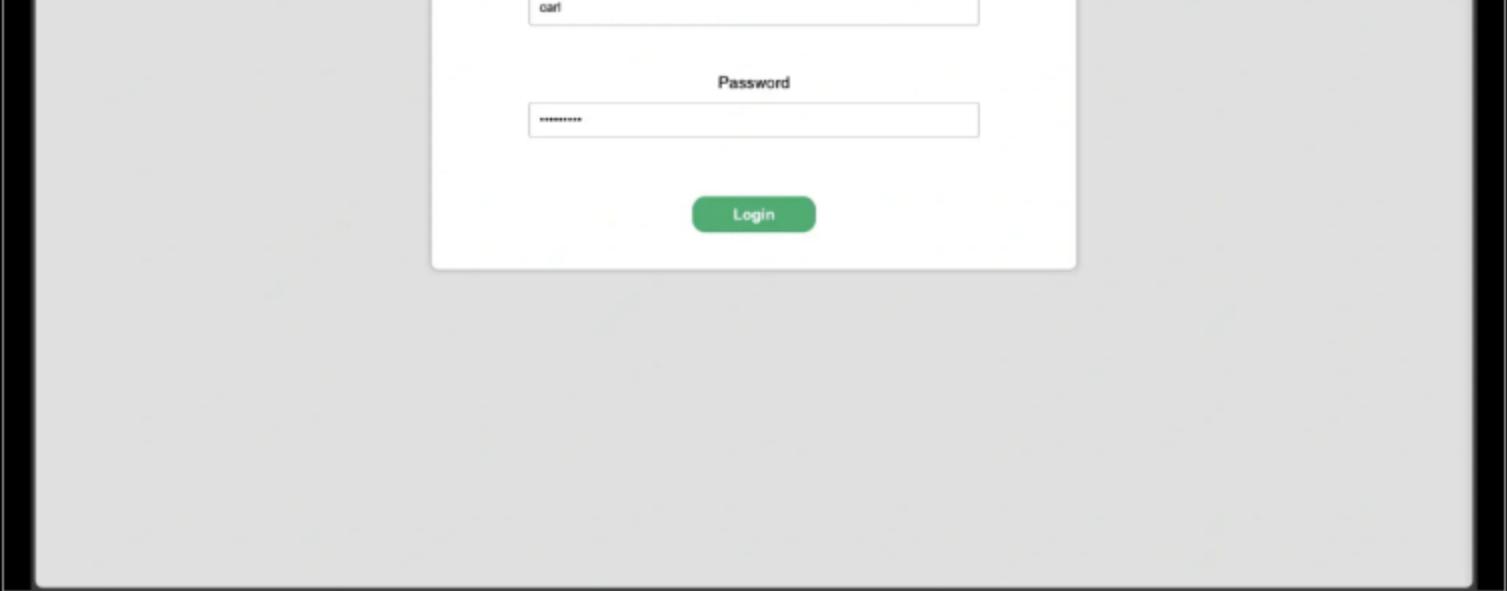


Showing success login message (with correct EMAIL)

Checklist Items (1)

#5 Show success login message

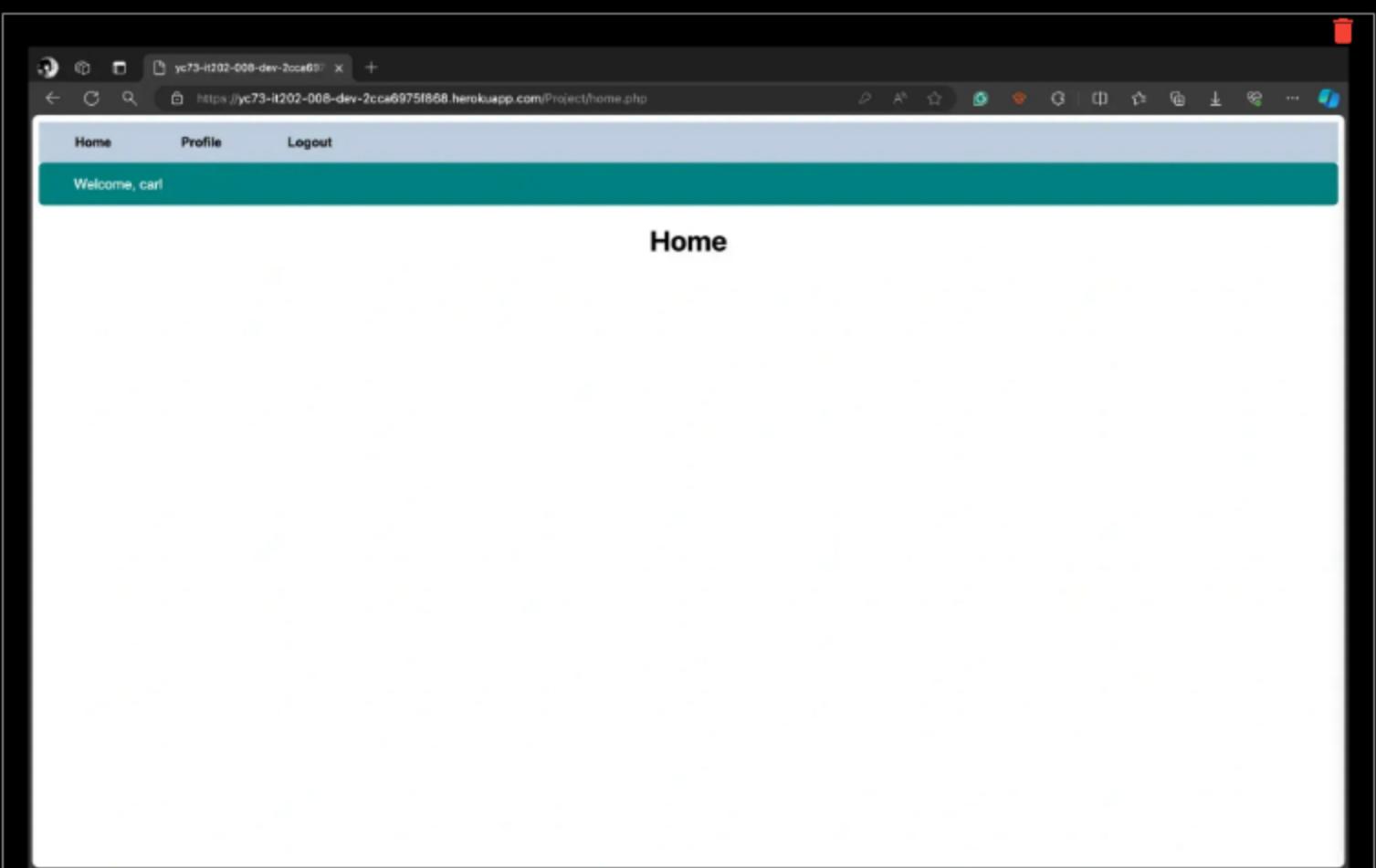




Showing correct USERNAME data to login

Checklist Items (1)

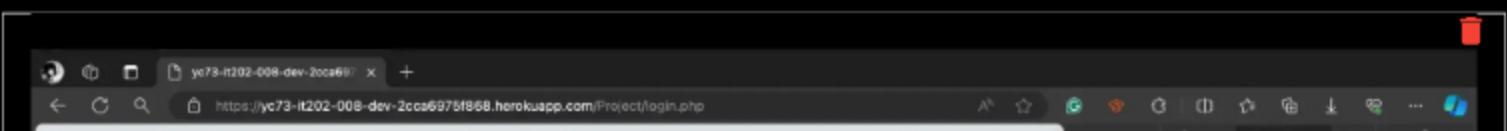
#3 Include correct data where username is used to login



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)



```

Login Register
Email/Username must not be empty
Invalid username
Password must not be empty

Email/Username
Password
Login

```

```

// Get all input elements within each form
let inputs = form.querySelectorAll("input");
// Disable validation for each input element
for (let inp of inputs) {
    // List of attributes to remove
    const attributesToRemove = ["required", "maxlength", "min", "max", "step", "pattern"];
    // Iterate over attributes and remove them
    attributesToRemove.forEach(attr => {
        if (inp.hasAttribute(attr)) {
            inp.removeAttribute(attr);
            console.log(`Removed ${attr} from element ${inp.name} || "[No name]"`);
        }
    });
    // Change type to text if not already text
    if (!["text", "submit", "reset"].includes(inp.type)) {
        inp.type = "text";
        console.log(`Changed type to text for element ${inp.name} || "[No name]"`);
    }
}
alert("HTML Validation has been disabled until page reload!");
})(());
Removed required from element email
Removed required from element password
Removed minlength from element password
Changed type to text for element password
< undefined
>

```

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)

```

Login Register
Invalid email address

Email/Username
carl@wrong
Password
password1
Login

```

```

// Get all input elements within each form
let inputs = form.querySelectorAll("input");
// Disable validation for each input element
for (let inp of inputs) {
    // List of attributes to remove
    const attributesToRemove = ["required", "maxlength", "min", "max", "step", "pattern"];
    // Iterate over attributes and remove them
    attributesToRemove.forEach(attr => {
        if (inp.hasAttribute(attr)) {
            inp.removeAttribute(attr);
            console.log(`Removed ${attr} from element ${inp.name} || "[No name]"`);
        }
    });
    // Change type to text if not already text
    if (!["text", "submit", "reset"].includes(inp.type)) {
        inp.type = "text";
        console.log(`Changed type to text for element ${inp.name} || "[No name]"`);
    }
}
alert("HTML Validation has been disabled until page reload!");
})(());
Removed required from element email
Removed required from element password
Removed minlength from element password
Changed type to text for element password
< undefined
>

```

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)

The screenshot shows a browser window with a login form. The URL is <https://yc73-lt202-008-dev-2cca6975f868.herokuapp.com/Project/login.php>. The page has 'Login' and 'Register' buttons at the top. A yellow banner below the buttons says 'Invalid username'. The login form contains two input fields: 'Email/Username' with value 'car?' and 'Password' with value 'password1'. Below the inputs is a green 'Login' button. On the right side of the browser, the developer tools' console tab is open, showing the following JavaScript code and its execution:

```
// Get all input elements within each form
let inputs = form.querySelectorAll("input");
// Disable validation for each input element
for (let inp of inputs) {
    // List of attributes to remove
    const attributesToRemove = ["required", "maxlength", "minlength", "max", "step", "pattern"];
    // Iterate over attributes and remove them
    attributesToRemove.forEach(attr => {
        if (inp.hasAttribute(attr)) {
            inp.removeAttribute(attr);
            console.log(`Removed ${attr} from element ${inp.name} || ${No name}`);
        }
    });
    // Change type to text if not already text
    if (!["text", "submit", "reset"].includes(inp.type)) {
        inp.type = "text";
        console.log(`Changed type to text for element ${inp.name} || ${No name}`);
    }
}
alert("HTML Validation has been disabled until page reload");
})(());
Removed required from element email
Removed required from element password
Removed minlength from element password
Changed type to text for element password
```

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)

The screenshot shows a browser window with a login form. The URL is <https://yc73-lt202-008-dev-2cca6975f868.herokuapp.com/Project/login.php>. The page has 'Login' and 'Register' buttons at the top. A yellow banner below the buttons says 'Invalid username'. The login form contains two input fields: 'Email/Username' with value 'ca' and 'Password' with value 'password1'. Below the inputs is a green 'Login' button. On the right side of the browser, the developer tools' console tab is open, showing the same JavaScript code as the previous screenshot, indicating that the validation logic remains the same.

Removed required from element email
Removed required from element password
Removed minlength from element password
Changed type to text for element password

undefined

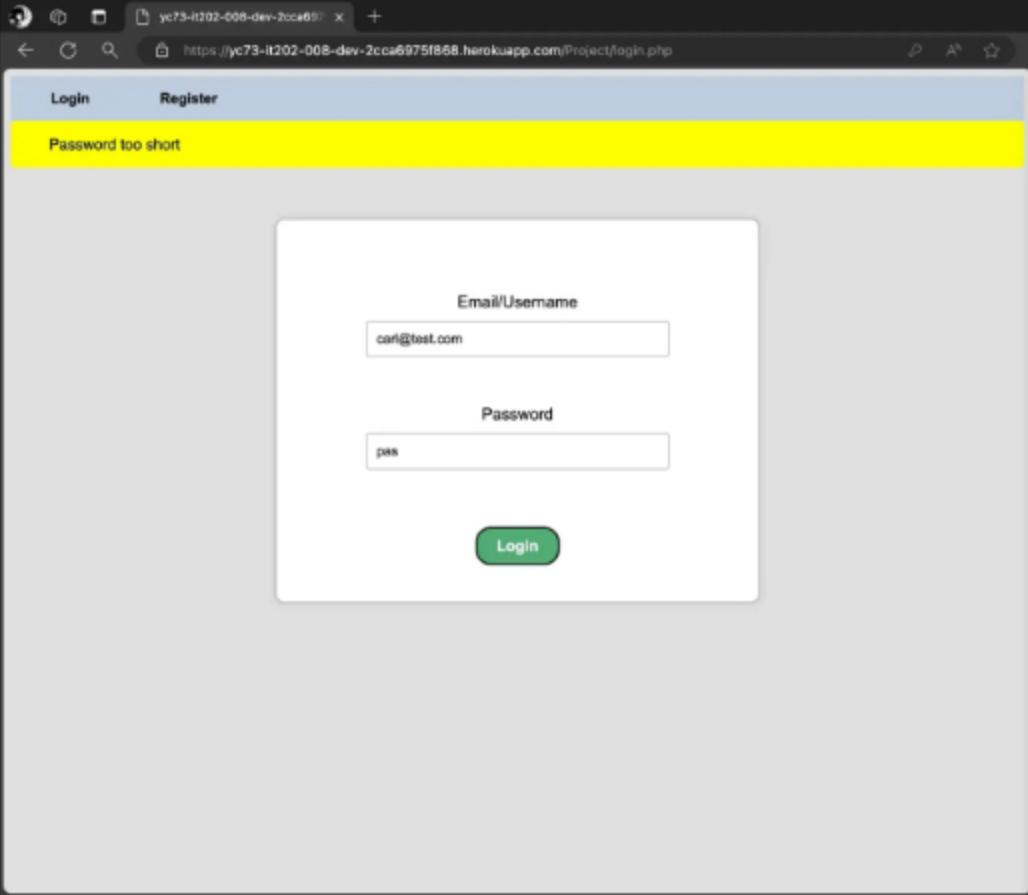
>

Console Issues +

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)



The screenshot shows a browser window with a login form. At the top, there are two tabs: "Login" and "Register". Below the tabs, a yellow bar displays the error message "Password too short". The main form contains two input fields: "Email/Username" with the value "carl@test.com" and "Password" with the value "pas". A green "Login" button is at the bottom. On the right side of the browser, the developer console is open, showing the following JavaScript code and its execution:

```
// Get all input elements within each form
let inputs = form.querySelectorAll("input");
// Disable validation for each input element
for (let inp of inputs) {
    // List of attributes to remove
    const attributesToRemove = ["required", "minlength", "maxlength", "min", "max", "step", "pattern"];
    // Iterate over attributes and remove them
    attributesToRemove.forEach(attr => {
        if (inp.hasAttribute(attr)) {
            inp.removeAttribute(attr);
            console.log(`Removed ${attr} from element ${inp.name} || ${No name}`); // No name
        }
    });
    // Change type to text if not already text
    if (!["text", "submit", "reset"].includes(inp.type)) {
        inp.type = "text";
        console.log(`Changed type to text for element ${inp.name} || ${No name}`); // No name
    }
}
alert("HTML Validation has been disabled until page reload");
```

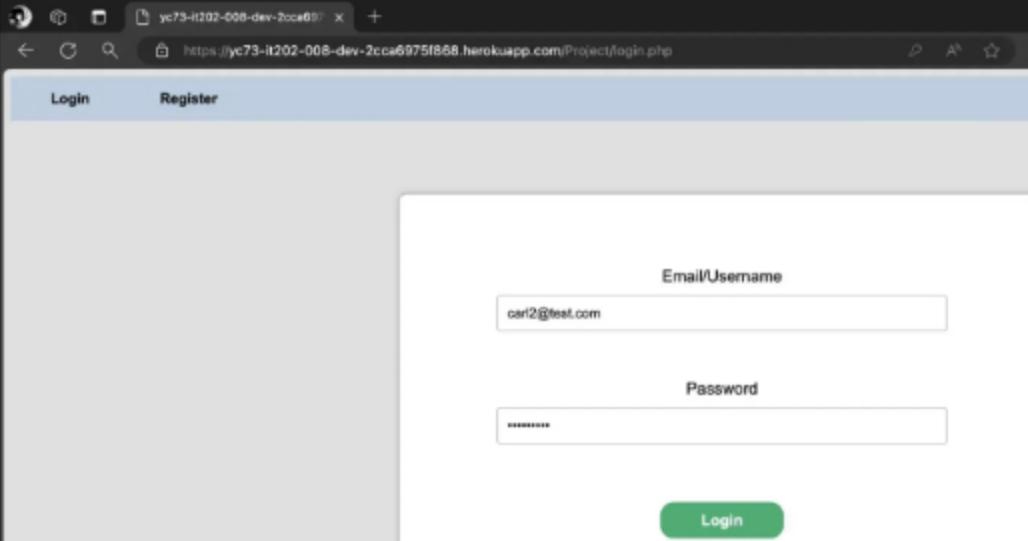
The console also lists several log entries:

- Removed required from element email
- Removed required from element password
- Removed minlength from element password
- Changed type to text for element password

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)



The screenshot shows a browser window with a login form. At the top, there are two tabs: "Login" and "Register". Below the tabs, a yellow bar displays the error message "Password too short". The main form contains two input fields: "Email/Username" with the value "carl2@test.com" and "Password" with the value "*****". A green "Login" button is at the bottom. On the right side of the browser, the developer console is open, showing the following JavaScript code and its execution:

```
// Get all input elements within each form
let inputs = form.querySelectorAll("input");
// Disable validation for each input element
for (let inp of inputs) {
    // List of attributes to remove
    const attributesToRemove = ["required", "minlength", "maxlength", "min", "max", "step", "pattern"];
    // Iterate over attributes and remove them
    attributesToRemove.forEach(attr => {
        if (inp.hasAttribute(attr)) {
            inp.removeAttribute(attr);
            console.log(`Removed ${attr} from element ${inp.name} || ${No name}`); // No name
        }
    });
    // Change type to text if not already text
    if (!["text", "submit", "reset"].includes(inp.type)) {
        inp.type = "text";
        console.log(`Changed type to text for element ${inp.name} || ${No name}`); // No name
    }
}
alert("HTML Validation has been disabled until page reload");
```

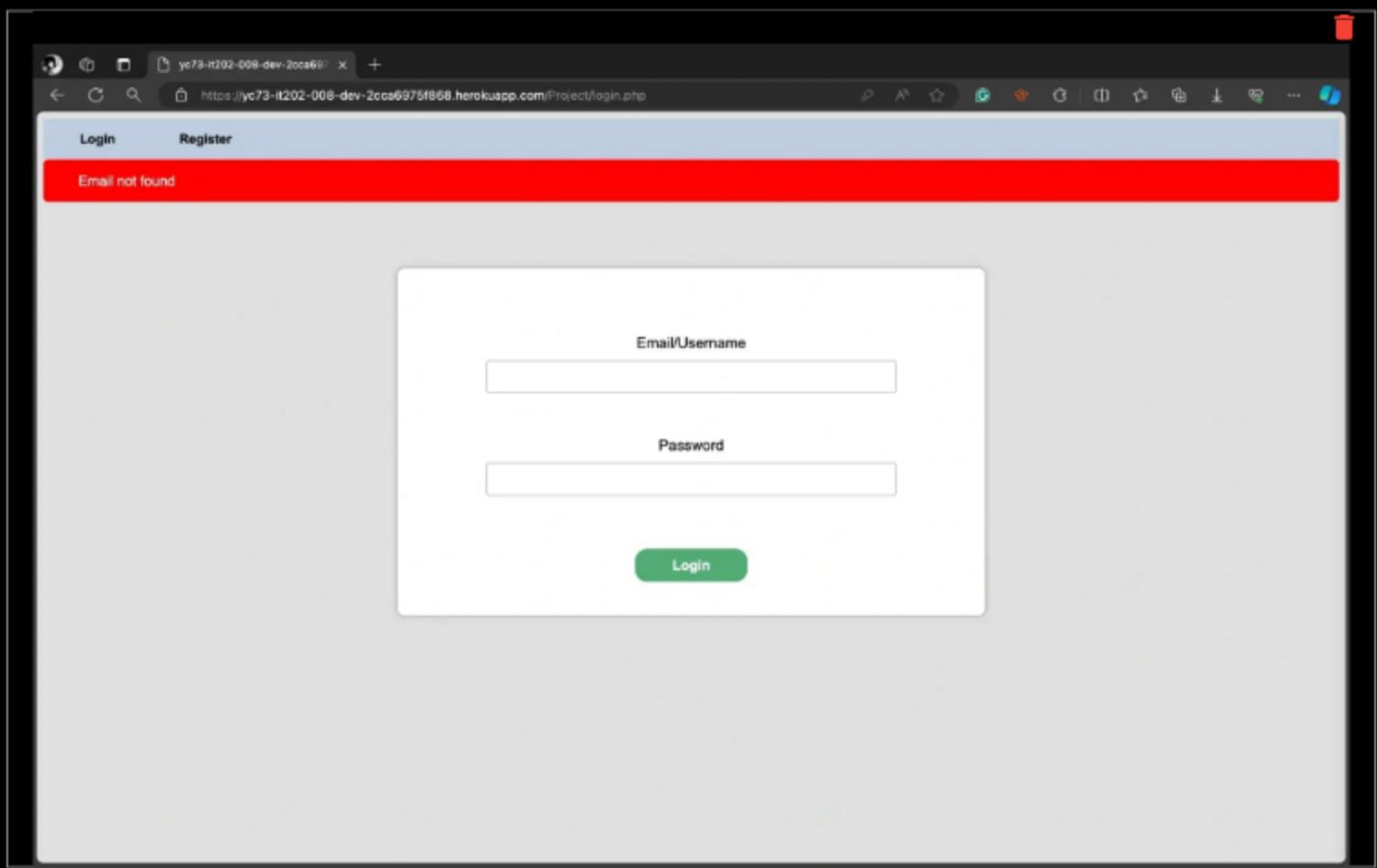
The console also lists several log entries:

- Removed required from element email
- Removed required from element password
- Removed minlength from element password
- Changed type to text for element password

Showing email that does NOT exist

Checklist Items (1)

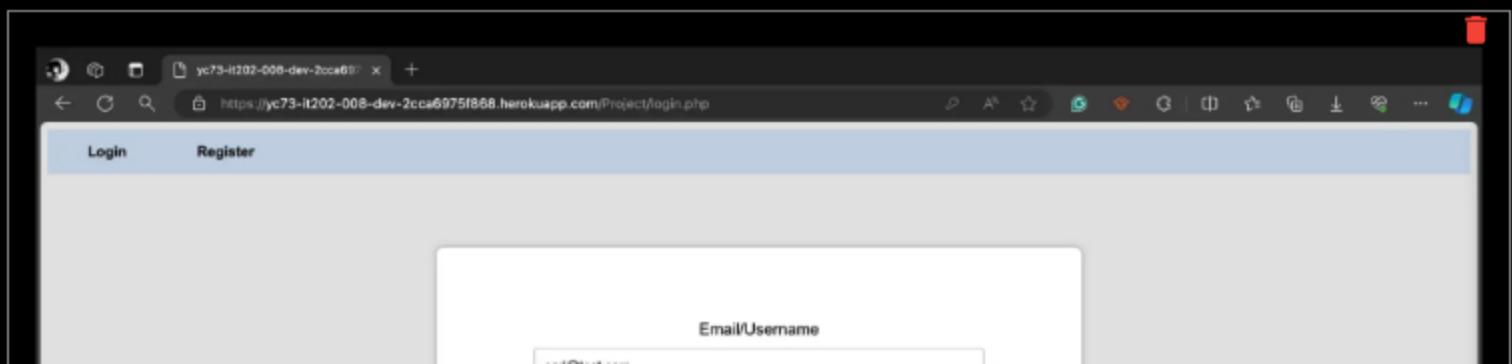
#7 Demonstrate user-friendly message of when an account doesn't exist

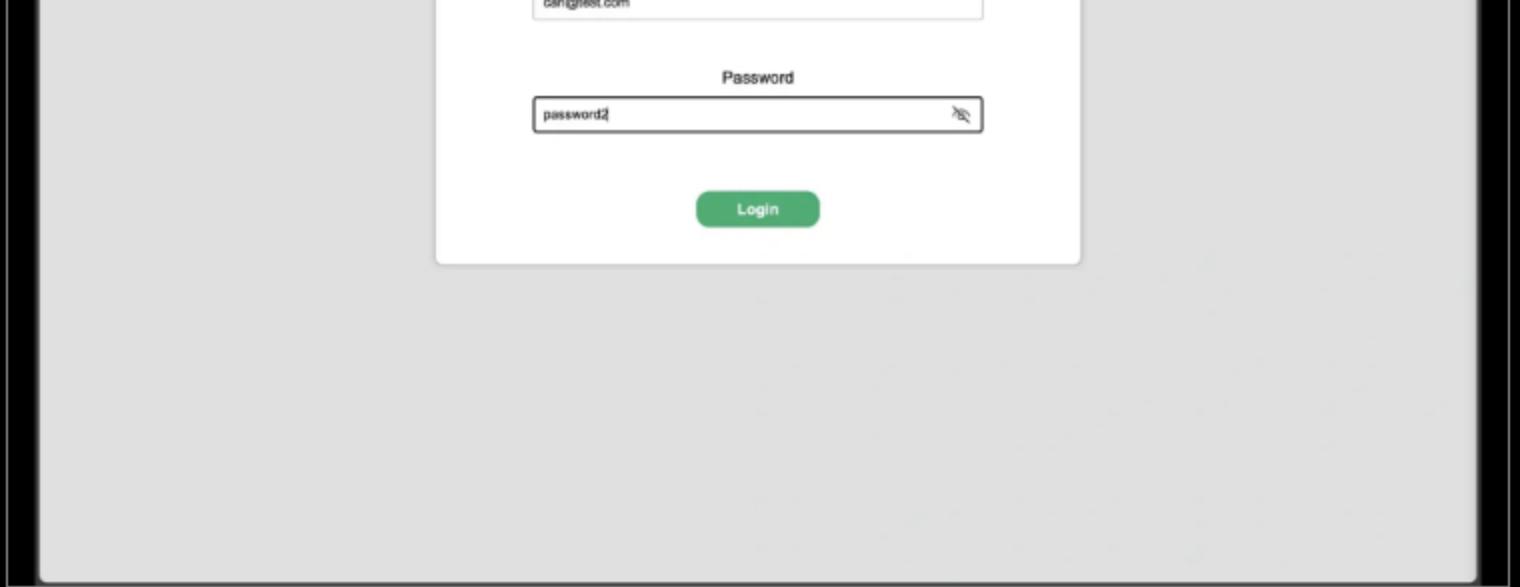


Showing message when an account does NOT exist

Checklist Items (1)

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

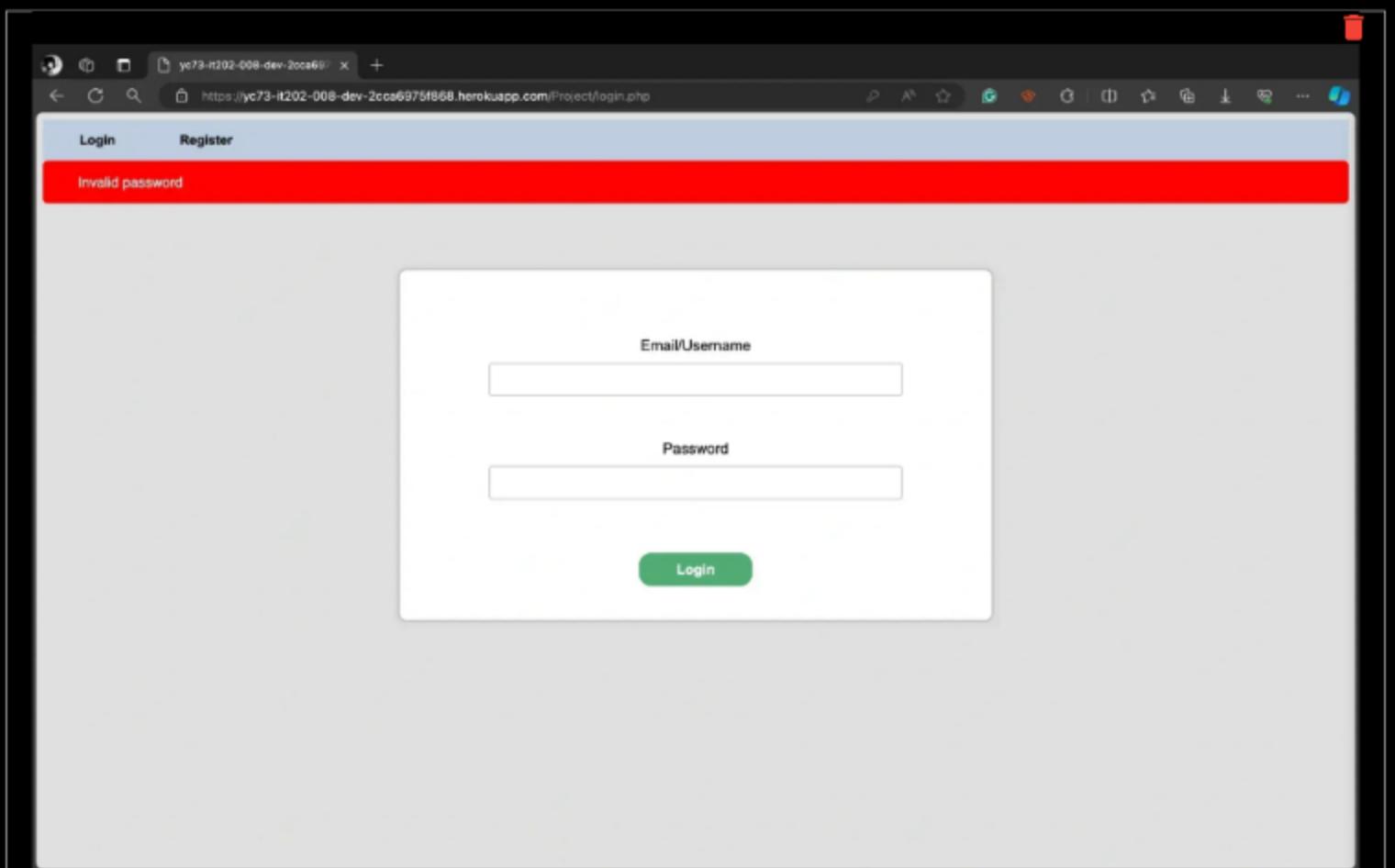




Showing INCORRECT password

Checklist Items (1)

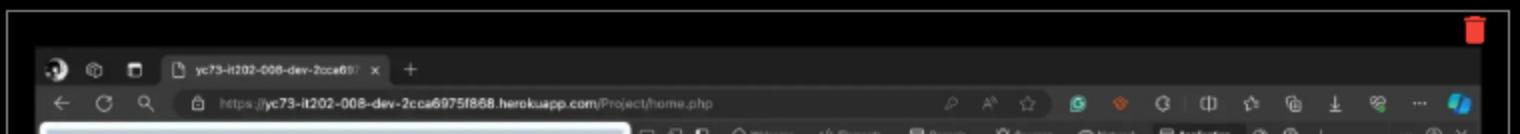
#8 Demonstrate user-friendly message of when password doesn't match what's in the DB

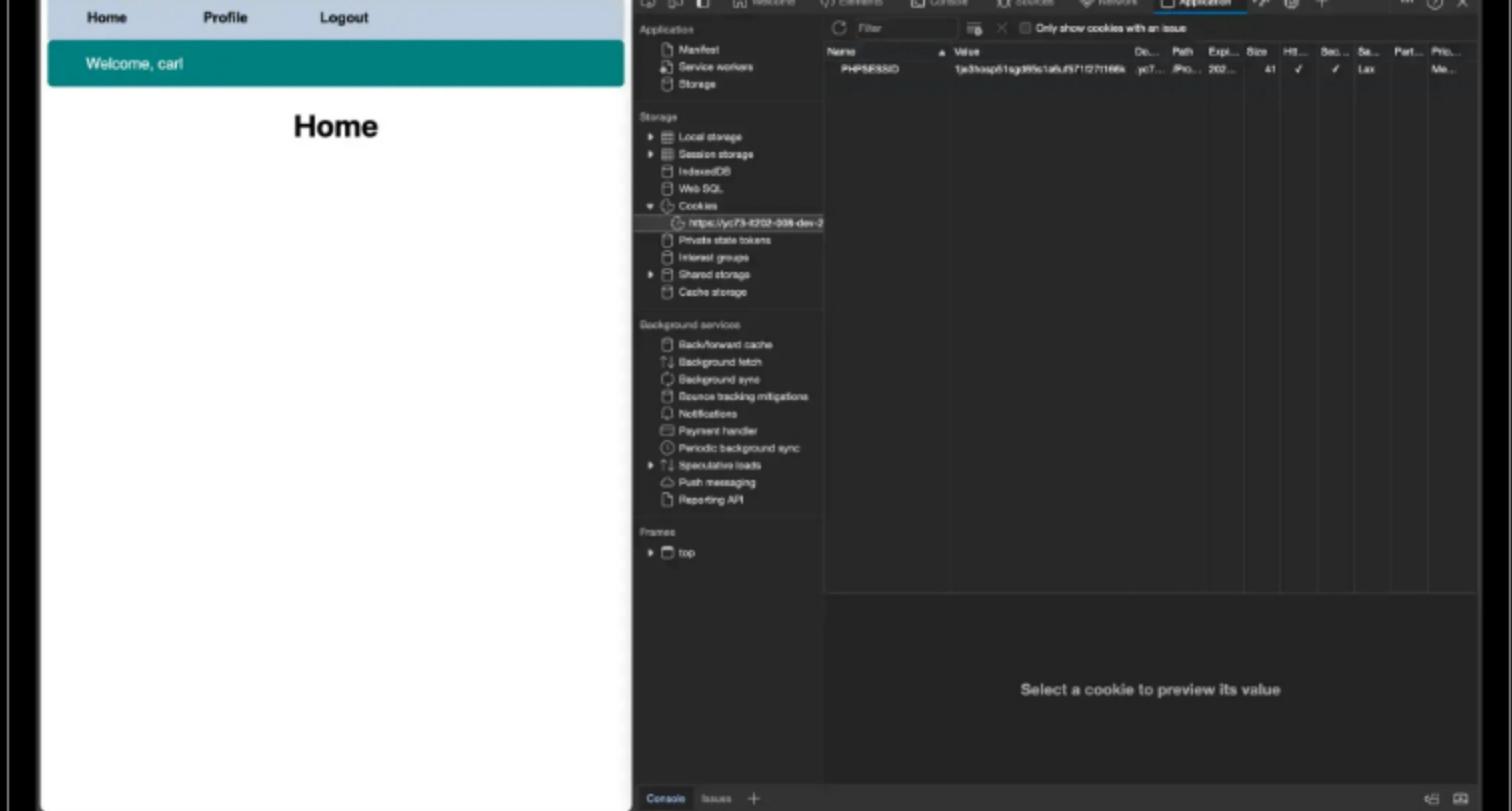


Showing message when password does NOT match what's in the DB

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)

^COLLAPSE ^

Task #2 - Points: 1**Text: Screenshot of the form code****Checklist**

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #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)
<input checked="" type="checkbox"/> #2	1	Show JavaScript validations (include any extra files related)
<input checked="" type="checkbox"/> #3	1	Show PHP validations (include any lib content)
<input checked="" type="checkbox"/> #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})+$/;
        if (!email_pattern.test(email)) {
            flash("Invalid email address", "warning");
            is_valid = false;
        }
    } <- #37-43 if (email.includes("@"))
    else {
        const user_pattern = /^[a-zA-Z0-9._-]{3,16}$/;
        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
    // validate 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 = $_POST["email", "", false];
    $password = $_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)

```
if (!hasError) {
    //flash("Welcome, $email");
    //1000-4
    $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) {
                $hash = $user["password"];
                unset($user["password"]);
                if (password_verify($password, $hash)) {
                    //flash("Reclone $email");
                    $_SESSION["user"] = $user; //sets our session data from db
                    try {
                        //lookup potential roles
                        $stmt = $db->prepare("SELECT Roles.name FROM Roles
JOIN UserRoles on Roles.id = UserRoles.role_id
where UserRoles.user_id = :user_id and Roles.is_active = 1 and UserRoles.is_active = 1");
                        $stmt->execute([":user_id" => $user["id"]]);
                        $roles = $stmt->fetchAll(PDO::FETCH_ASSOC); //fetch all since we'll want multiple
                    } catch (Exception $e) {
                        error_log(var_export($e, true));
                    }
                    //save roles or empty array
                    if (isset($roles)) {
                        $_SESSION["user"]["roles"] = $roles; //at least 1 role
                    } else {
                        $_SESSION["user"]["roles"] = []; //no roles
                    }
                    flash("Welcome, " . get_username());
                    die(header("Location: home.php"));
                } else {
                    flash("Invalid password", "danger");
                }
            } else {
                flash("Email not found", "danger");
            }
        } <- #119-149 if ($user)
    } catch (Exception $e) {
        flash("<pre>" . var_export($e, true) . "</pre>");
    }
} <- #115-153 try
} <- #118-154 if (!hasError)
?>
```

Showing PHP validations (part 2)

Checklist Items (1)

#3 Show PHP validations (include any lib content)

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
<input type="checkbox"/> #1	1	Don't just show code, translate things to plain English
<input type="checkbox"/> #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 submission, the data is sent to the PHP script which then performs validation. This validation consists of:

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.

Task #4 - Points: 1

 COLLAPSE

Text: Include pull request links related to this feature

Details:

Should end in /null/#

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>

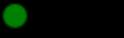
URL #3

<https://github.com/yaneliii/yc73-it202-008/pull/26>



User Logout (1 pt.)

[COLLAPSE](#)



Task #1 - Points: 1

[COLLAPSE](#)

Text: Capture the following screenshots

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input type="checkbox"/> #1	1	Screenshot of the navigation when logged in (site)
<input type="checkbox"/> #2	1	Screenshot of the redirect to login with the user-friendly logged-out message (site)
<input type="checkbox"/> #3	1	Screenshot of the logout-related code showing the session is destroyed (code). Ensure uid/date comment is present.

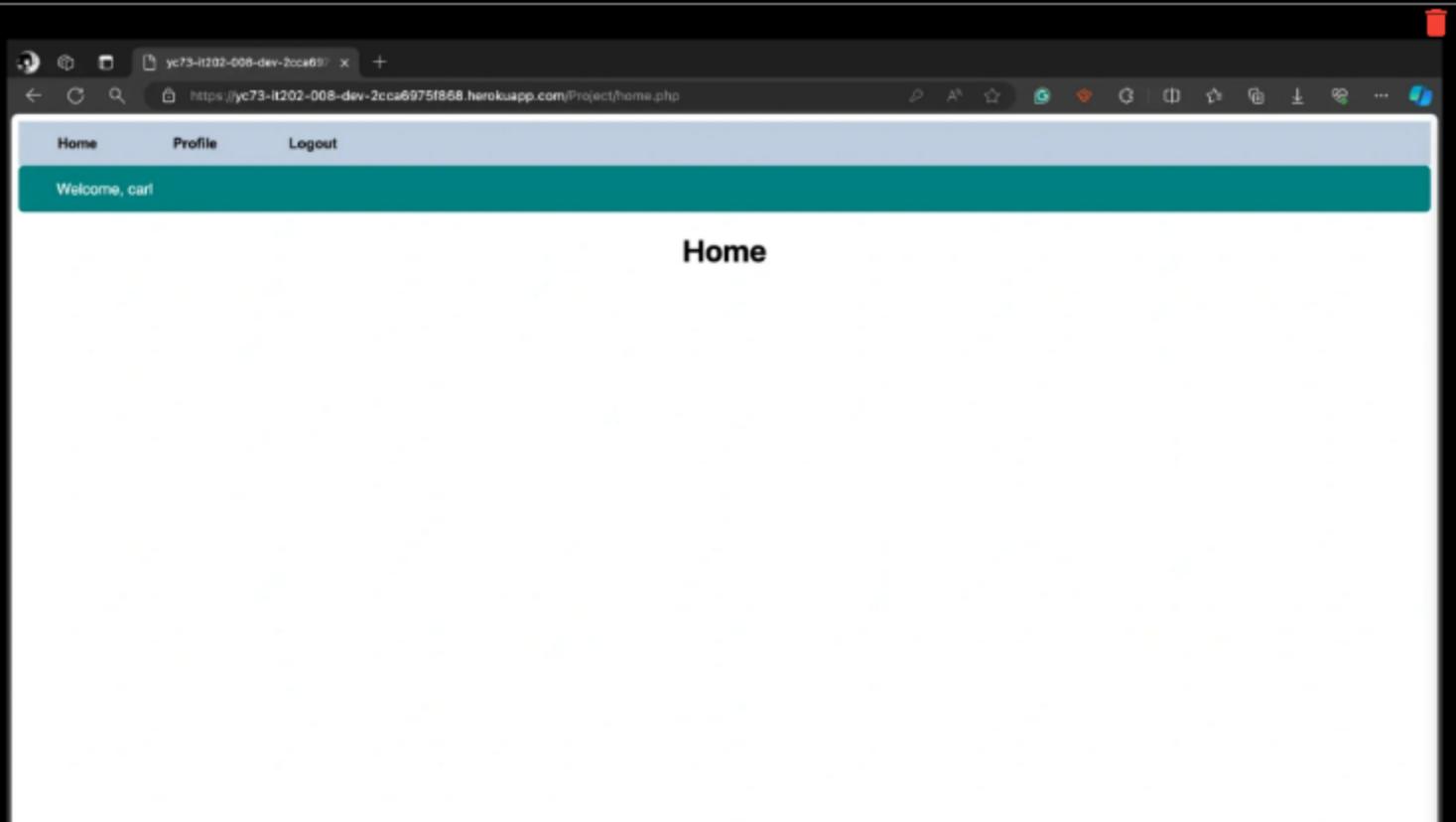
Task Screenshots:

Gallery Style: Large View

Small

Medium

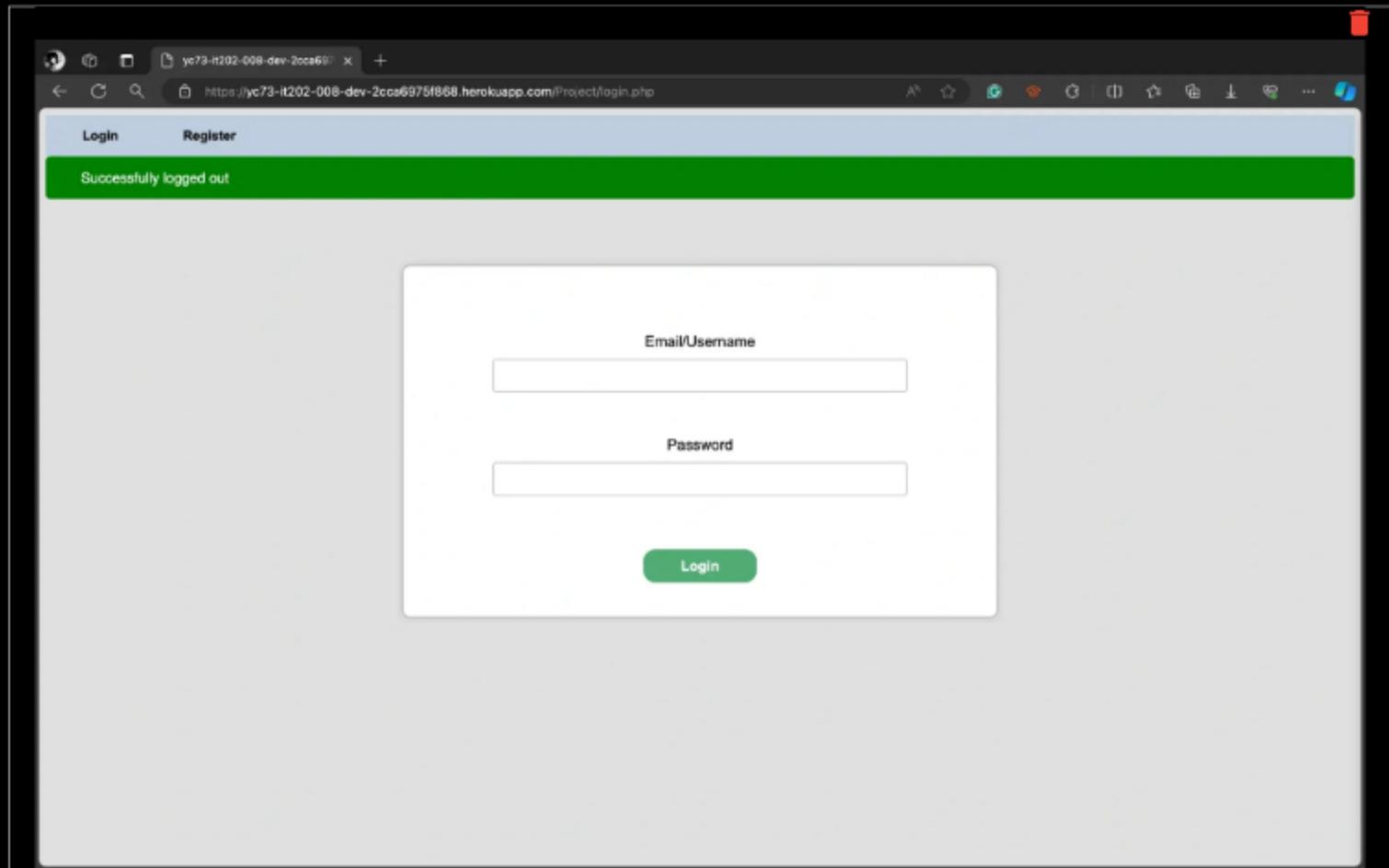
Large



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)

```
c_html > Project > 🐘 logout.php
You, 1 second ago | 1 author (You)
<?php
/* yc73 4/1/23 */
session_start();
require(__DIR__ . "/../../../../lib/functions.php");
reset_session();

flash("Successfully logged out", "success");
```

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

Showing logout code

Checklist Items (0)

reset_session.php > ...

You, 1 second ago | 1 author (You)

```
<?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.

Task #2 - 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/18>

URL #2

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

Basic Security Rules and Roles (2 pts.)

[^COLLAPSE ^](#)

Task #1 - Points: 1

Text: Authentication Screenshots

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Screenshot of the function that checks if a user is logged in
<input checked="" type="checkbox"/> #2	1	Screenshot of the login check function being used (i.e., profile likely). Also caption what pages it's used on
<input checked="" type="checkbox"/> #3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)
<input checked="" type="checkbox"/> #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

```
/* 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 $isLoggedIn;
} <- #18-19 function is_logged_in($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;
} <- #28-29 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

c_html > Project > profile.php > ...

```
<?php
require_once(__DIR__ . "/../../partials/nav.php");
/* yc73 4/1/23 */
is_logged_in(true);
?>
<?php
```

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

c_html > Project > home.php > ...

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

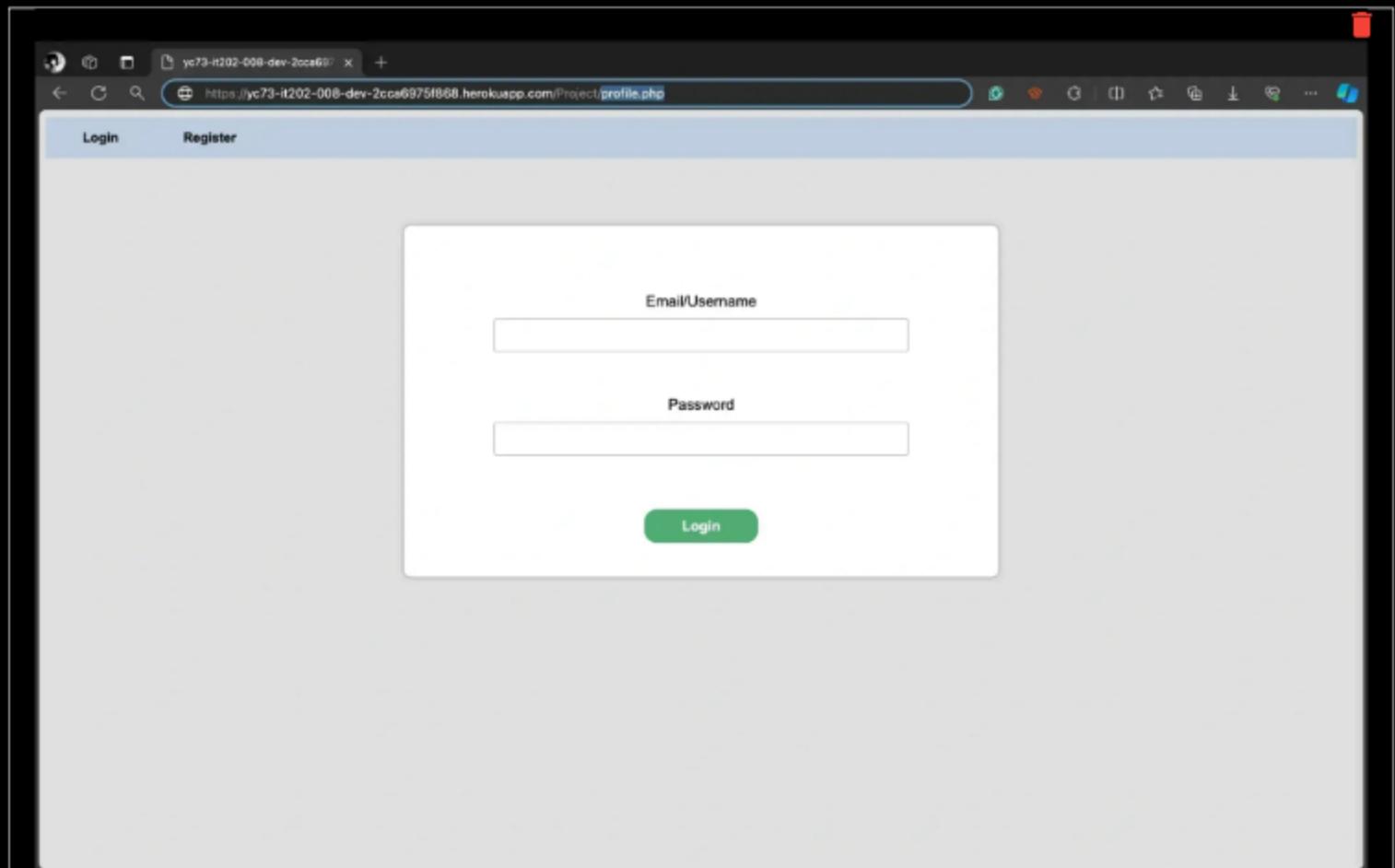
/* yc73 4/1/23 */
if (is_logged_in(true)) {
    //flash("Welcome, " . get_user_email());
    error_log("Session data: " . var_export($_SESSION, true));
}
/*else {
    flash("You must log in to view this page.");
}
```

```
    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)

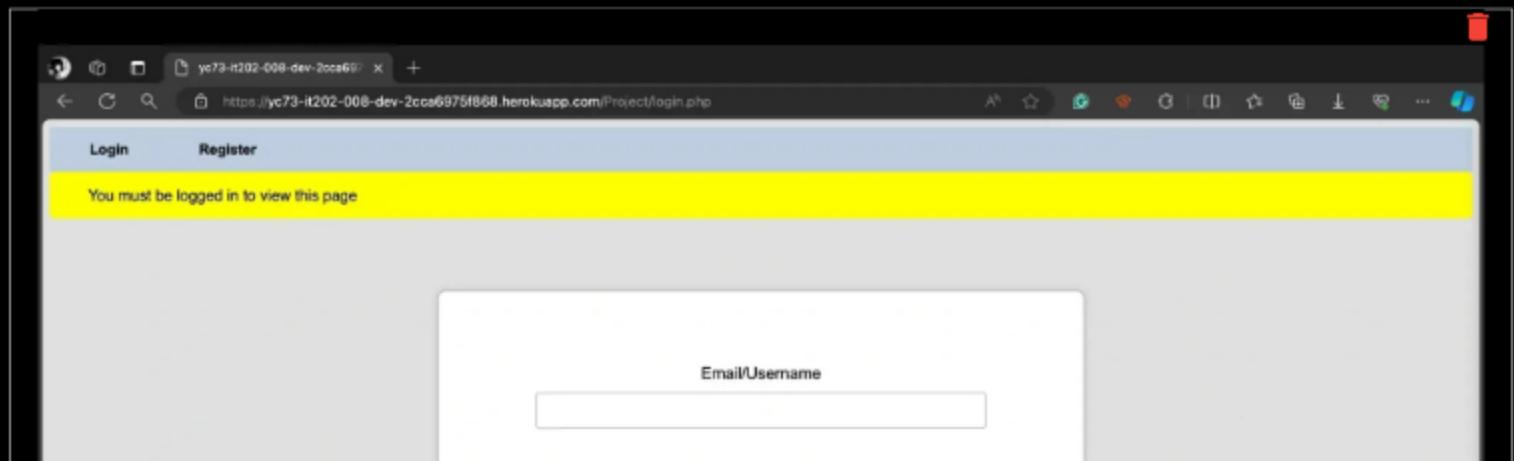
#2 Screenshot of the login check function being used (i.e., profile likely). Also caption what pages it's used on

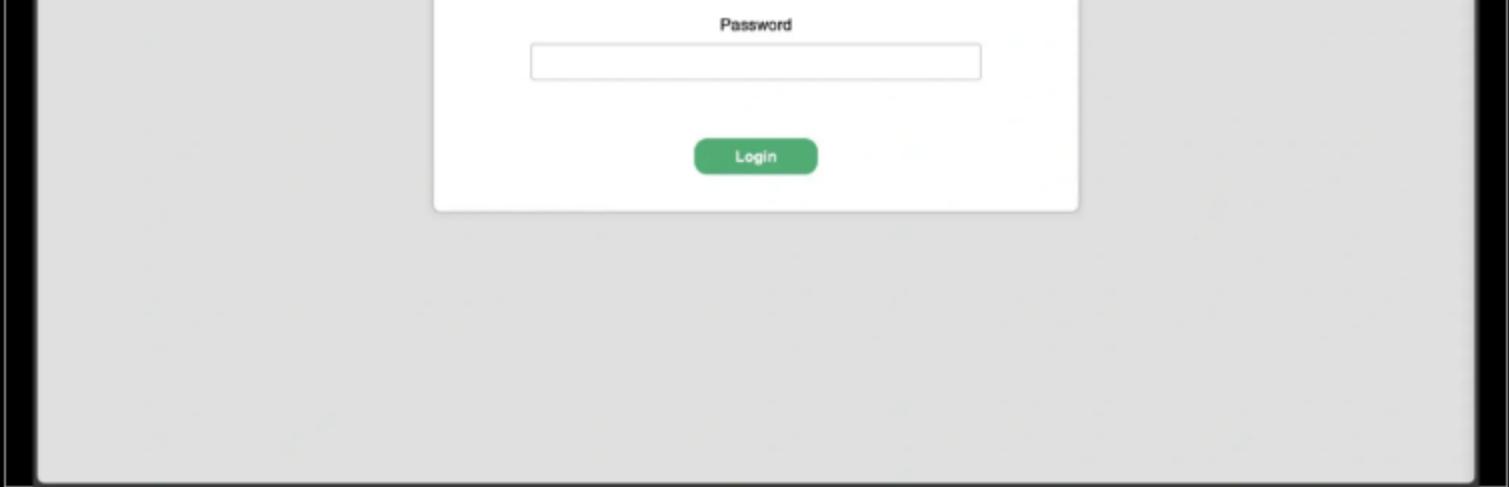


Showing in search bar the login-protected page (Profile) that I am attempting to access (while being logged out)

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

Task #2 - Points: 1

Text: Authorization Screenshots

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Screenshot of the function that checks for a specific role
<input checked="" type="checkbox"/> #2	1	Screenshot of the role check function being used. Also caption what pages it's used on
<input checked="" type="checkbox"/> #3	1	Include uid/date code comments in all screenshots (one per screenshot is sufficient)
<input checked="" type="checkbox"/> #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 $_SESSION["user"]["username"];
    }
}
```

```
        }
        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)

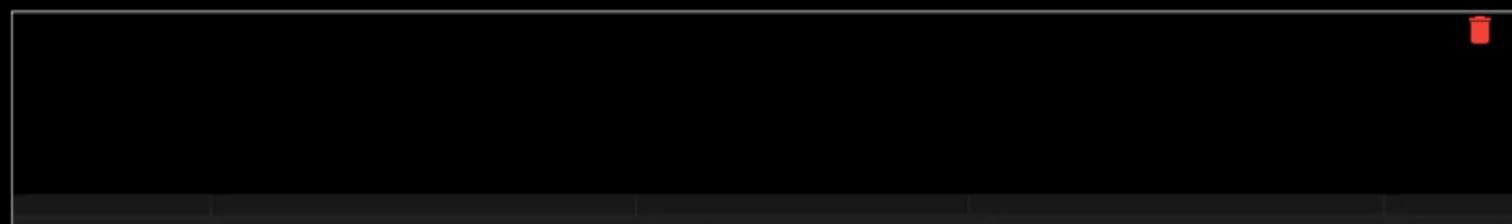
#1 Screenshot of the function that checks for a specific role

The screenshot shows a GitHub commit history. The commit message is: "You, 14 seconds ago | 1 author (You) <?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". The commit has a red trash icon in the top right corner.

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



_html > Project > admin > 📄 create_role.php > div.crBody

You, 1 minute ago | 1 author (You)

```
<?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: " . get_url("home.php")));
}

if (isset($_POST["name"]) && isset($_POST["description"])) {
    // handle POST ...
}
```

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

_html > Project > admin > 📄 list_roles.php > ...

You, 2 minutes ago | 1 author (You)

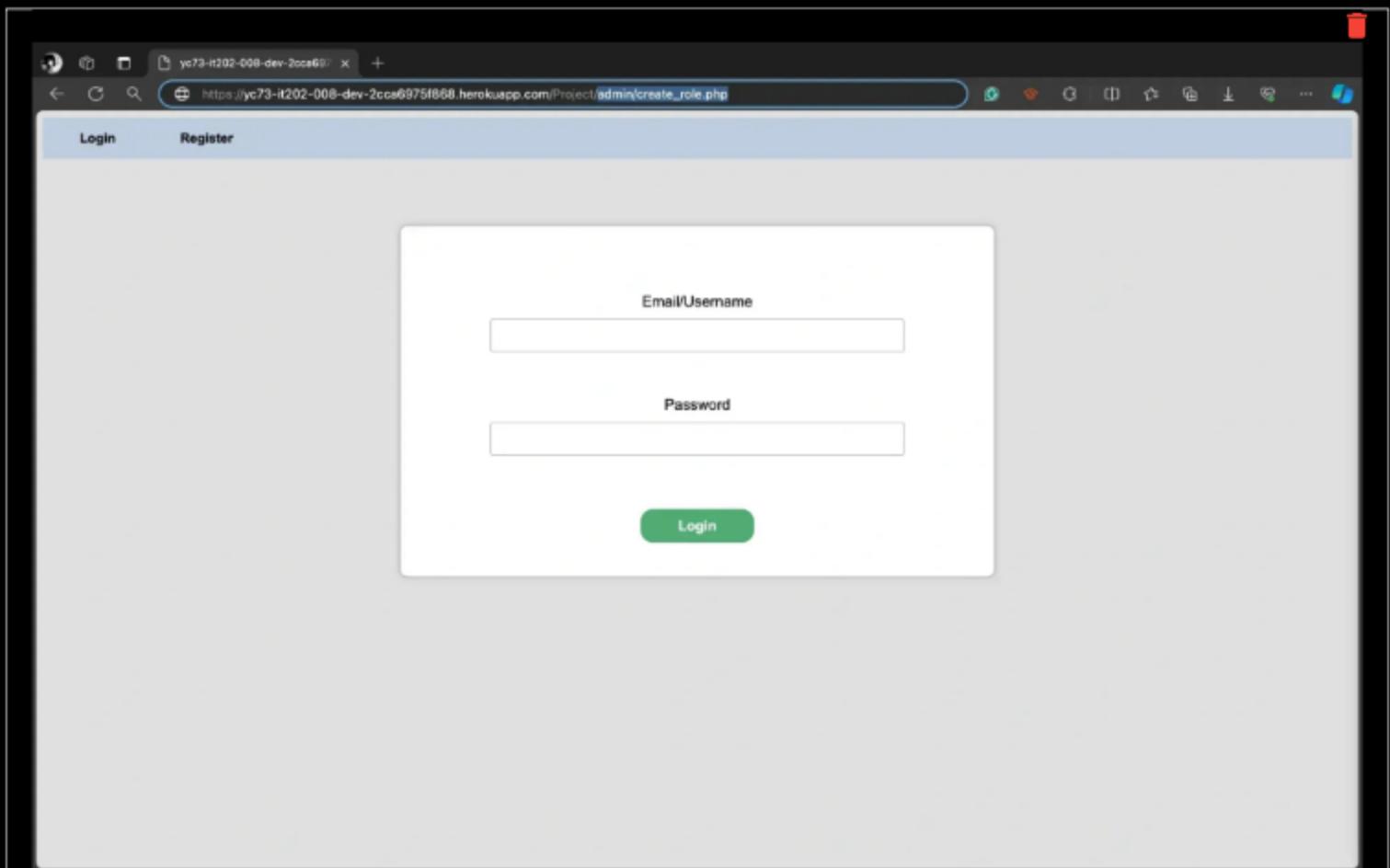
```
<?php
//note we need to go up 1 more directory
require(__DIR__ . "/../../../../partials/nav.php");

/* yc73 4/1/23 */
if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: " . get_url("home.php")));
}
//handle the toggle first so select pulls fresh data
if (isset($_POST["role_id"])) {
```

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

Checklist Items (1)

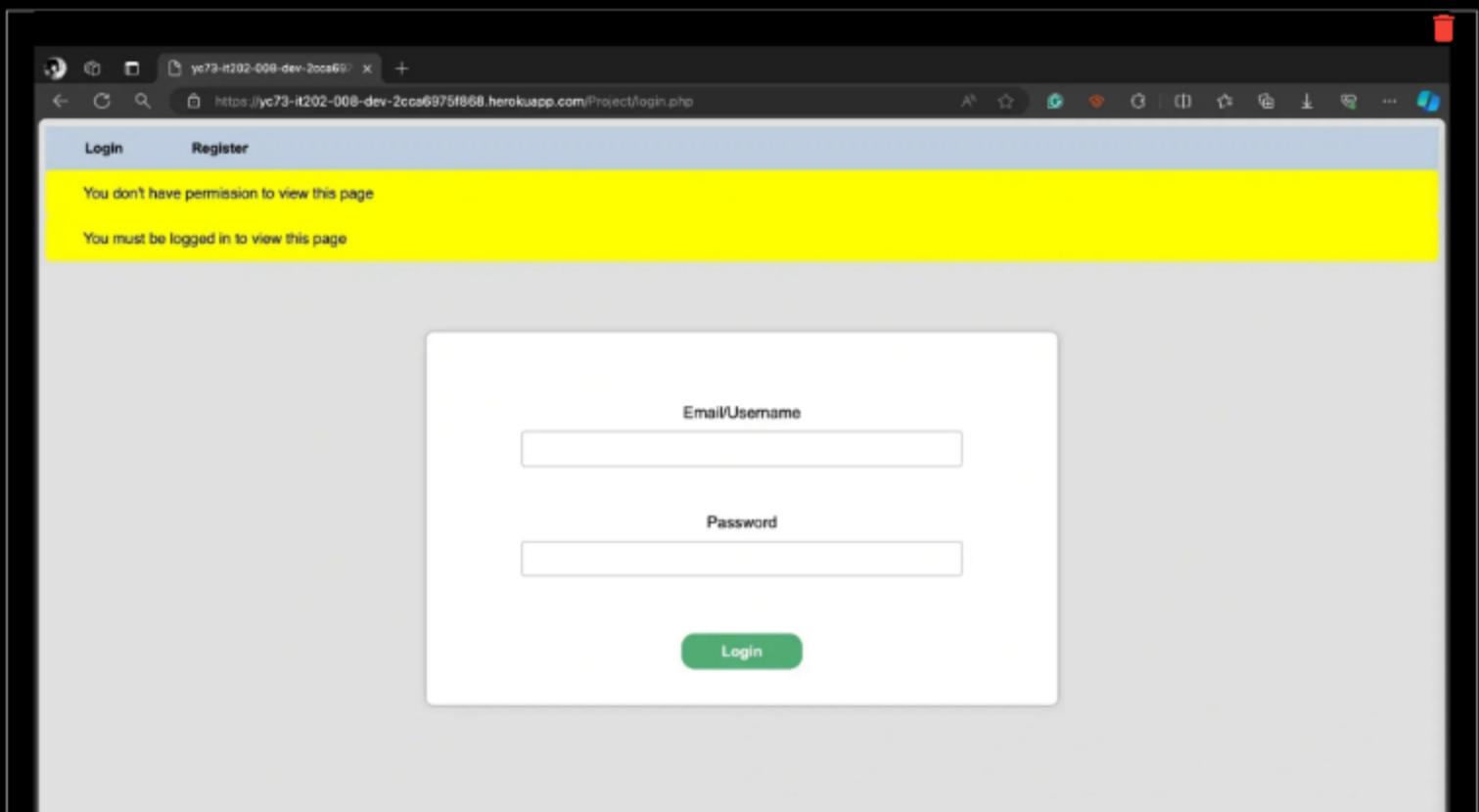
#2 Screenshot of the role check function being used. Also caption what pages it's used on



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

Task #3 - Points: 1

Text: Screenshots of UserRoles and Roles Tables

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input type="checkbox"/> #1	1	At least one valid and enabled User->Role reference (UserRoles table)
<input type="checkbox"/> #2	1	UserRoles Table should have id, user_id, role_id, is_active, created, and modified columns
<input type="checkbox"/> #3	1	Roles Table should have id, name, description, is_active, modified, and created columns
<input type="checkbox"/> #4	1	At least one valid and enabled Role (Roles table)
<input type="checkbox"/> #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

The screenshot shows the MySQL Workbench interface with the 'UserRoles' table selected in the left sidebar under the 'Tables' section. The table has the following structure:

	id	user_id	role_id	is_active	created	modified
>	1	36	-1	1	2024-03-21 02:49:16	2024-03-21 02:49:16

The SQL query in the editor is:

```
SELECT * FROM 'UserRoles' LIMIT 100
```

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

Checklist Items (1)

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

The screenshot shows the MySQL Workbench interface. On the left, the database tree shows 'it202' selected, with 'Tables (3)' expanded to show 'Roles' and 'UserRoles'. The 'UserRoles' table is selected. The main pane displays the 'Roles' table with the following data:

	Id	name	description	is_active	created	modified
	1	Admin		1	2024-03-20 17:42:58	2024-03-21 02:52:25

Showing valid and enabled Role (Roles table)

Checklist Items (1)

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

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
<input type="checkbox"/> #1	1	What's the purpose of the UserRoles table?
<input type="checkbox"/> #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.

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
<input checked="" type="checkbox"/> #1	1	Heroku dev url should be present in the address bar
<input checked="" type="checkbox"/> #2	1	Should have thoughtful CSS applied
<input checked="" type="checkbox"/> #3	1	Show the profile form correctly populated on page load (username, email)
<input checked="" type="checkbox"/> #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

The screenshot shows a web application's profile editing interface. At the top, there's a navigation bar with links for 'Home', 'Profile', and 'Logout'. The main content area is titled 'Password Reset' and contains four input fields: 'Email' (with the value 'carl@test.com'), 'Username' (with the value 'carl'), 'Current Password', 'New Password', and 'Confirm Password'. Below these fields is a large, light-grey 'Update Profile' button.

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)

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)

username from the session data (specifically the `$_SESSION['username']` entry). Then, while generating the profile 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.

Task #3 - Points: 1

Text: Edit Profile Website Page

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input type="checkbox"/> #1	1	Heroku dev url should be present in the address bar
<input type="checkbox"/> #2	1	Should have thoughtful CSS applied
<input type="checkbox"/> #3	1	Demonstrate with before and after of a username change (including success message)
<input type="checkbox"/> #4	1	Demonstrate with a before and after of an email change (including success message)
<input type="checkbox"/> #5	1	Demonstrate the success message of updating password
<input type="checkbox"/> #6	1	Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)
<input type="checkbox"/> #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

yc73-il202-008-dev-2cc6975f868.herokuapp.com/project/profile.php

Home Profile Logout

Email: carl@test.com

Username: carl1

Password Reset

Current Password:

New Password:

Confirm Password:

Update Profile

Showing BEFORE username change

Checklist Items (1)

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

Profile saved

Email carl@test.com

Username carl

Password Reset

Current Password

New Password

Confirm Password

Update Profile

Showing AFTER username change with success message

Checklist Items (1)

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

Profile saved

Email carl1@test.com

Username carl1

Password Reset

Current Password

New Password

Confirm Password

Update Profile

Showing BEFORE email change

Checklist Items (1)

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

A screenshot of a web browser window showing a profile editing form. The URL in the address bar is <https://yc73-it202-008-dev-2cca6975f868.herokuapp.com/project/profile.php>. The page has a header with 'Home', 'Profile', and 'Logout' links. A green success message box at the top says 'Profile saved'. Below it is a form with fields for 'Email' (carl1@test.com) and 'Username' (carl1). Underneath is a 'Password Reset' section with fields for 'Current Password', 'New Password', and 'Confirm Password'. At the bottom is a grey 'Update Profile' button.

Showing BEFORE email change with success message

Checklist Items (1)

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

A screenshot of a web browser window showing a profile editing form. The URL in the address bar is <https://yc73-it202-008-dev-2cca6975f868.herokuapp.com/project/profile.php>. The page has a header with 'Home', 'Profile', and 'Logout' links. A green success message box at the top says 'Profile saved'. Below it is a form with fields for 'Email' (carl1@test.com) and 'Username' (carl1). Underneath is a 'Password Reset' section with fields for 'Current Password' (showing as '*****') and 'New Password'. At the bottom is a grey 'Update Profile' button.

New Password:

Confirm Password: password2 

Showing BEFORE updating password

Checklist Items (1)

#5 Demonstrate the success message of updating password

Profile saved
Password reset

Email:

Username:

Password Reset

Current Password:

New Password:

Confirm Password:

Showing AFTER updating password with success message

Checklist Items (1)

#5 Demonstrate the success message of updating password

Email must not be empty

Username must not be empty

Email

Username

Password Reset

Current Password

New Password

Confirm Password

```
const forms = document.forms;
for (const form of forms) {
    // Get all input elements within each form
    let inputs = form.querySelectorAll("input");
    // Disable validation for each input element
    for (let inp of inputs) {
        // List of attributes to remove
        const attributesToRemove = ["required", "minlength", "maxlength", "min", "max", "step", "pattern"];
        // Iterate over attributes and remove them
        attributesToRemove.forEach(attr => {
            if (inp.hasAttribute(attr)) {
                inp.removeAttribute(attr);
                console.log(`Removed ${attr} from element ${inp.name} || ${[No name]}`);
            }
        });
        // Change type to text if not already text
        if (!["text", "submit", "reset"].includes(inp.type)) {
            inp.type = "text";
            console.log(`Changed type to text for element ${inp.name} || ${[No name]}`);
        }
    }
}
alert("HTML Validation has been disabled until page reload!");
//:
Changed type to text for element email VMS459162
Changed type to text for element currentPassword VMS459162
Changed type to text for element newPassword VMS459167
Changed type to text for element confirmPassword VMS459162
e.undefined
:
```

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)

The screenshot shows a web browser window with the URL <https://yc73-it202-008-dev-2cca6975f868.herokuapp.com/Project/profile.php>. The page displays a profile update form with two error messages: "Invalid email address" and "Invalid username". The developer tools console tab is open, showing the following JavaScript code and logs:

```
function disableHTMLValidation() {
    // Get all forms on the page
    const forms = document.forms;
    for (const form of forms) {
        // Get all input elements within each form
        let inputs = form.querySelectorAll("input");
        // Disable validation for each input element
        for (let inp of inputs) {
            // List of attributes to remove
            const attributesToRemove = ["required", "minlength", "maxlength", "min", "max", "step", "pattern"];
            // Iterate over attributes and remove them
            attributesToRemove.forEach((attr) => {
                if (inp.hasAttribute(attr)) {
                    inp.removeAttribute(attr);
                }
            });
            console.log(`Removed ${attr} from element ${inp.name} || ${!(inp.name)})`);
        }
    }
    // Change type to text if not already text
    if (!["text", "submit", "reset"].includes(inp.type)) {
        inp.type = "text";
        console.log(`Changed type to text for element ${inp.name} || ${!(inp.name)})`);
    }
}
alert("HTML Validation has been disabled until page reload");
})( );

```

The developer tools also show several log entries indicating attribute changes:

- Changed type to text for element email
- Changed type to text for element currentPassword
- Changed type to text for element newPassword
- Changed type to text for element confirmPassword

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

Console output (JavaScript code for validation):

```
// Function to validate forms
function disableHTMLValidation() {
    // Get all forms on the page
    const forms = document.forms;
    for (const form of forms) {
        // Get all input elements within each form
        let inputs = form.querySelectorAll("input");
        // Disable validation for each input element
        for (let inp of inputs) {
            // List of attributes to remove
            const attributesToRemove = ["required", "minlength", "maxlength", "min", "max", "step", "pattern"];
            // Iterate over attributes and remove them
            attributesToRemove.forEach(attr => {
                if (inp.hasAttribute(attr)) {
                    inp.removeAttribute(attr);
                    console.log(`Removed ${attr} from element ${inp.name} || ${inp.type}`);
                }
            });
            // Change type to text if not already text
            if (!["text", "submit", "reset"].includes(inp.type)) {
                inp.type = "text";
                console.log(`Changed type to text for element ${inp.name} || ${inp.type}`);
            }
        }
    }
    alert("HTML Validation has been disabled until page reload");
}

// HTML validation messages
Changed type to text for element email
Changed type to text for element currentPassword
Changed type to text for element newPassword
Changed type to text for element confirmPassword
undefined
```

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)

Console output (JavaScript code for validation):

```
// Function to validate forms
function disableHTMLValidation() {
    // Get all forms on the page
    const forms = document.forms;
    for (const form of forms) {
        // Get all input elements within each form
        let inputs = form.querySelectorAll("input");
        // Disable validation for each input element
        for (let inp of inputs) {
            // List of attributes to remove
            const attributesToRemove = ["required", "minlength", "maxlength", "min", "max", "step", "pattern"];
            // Iterate over attributes and remove them
            attributesToRemove.forEach(attr => {
                if (inp.hasAttribute(attr)) {
                    inp.removeAttribute(attr);
                    console.log(`Removed ${attr} from element ${inp.name} || ${inp.type}`);
                }
            });
            // Change type to text if not already text
            if (!["text", "submit", "reset"].includes(inp.type)) {
                inp.type = "text";
                console.log(`Changed type to text for element ${inp.name} || ${inp.type}`);
            }
        }
    }
    alert("HTML Validation has been disabled until page reload");
}

// HTML validation messages
Changed type to text for element email
Changed type to text for element currentPassword
Changed type to text for element newPassword
Changed type to text for element confirmPassword
undefined
```

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)

The screenshot shows a web browser window with a profile update form. The URL in the address bar is <https://yc73-it202-008-dev-2cc697f868.herokuapp.com/Project/profile.php>. The form has fields for Email (containing "yan2@test.com"), Username (containing "carl1"), and Password Reset (with three input fields for Current Password, New Password, and Confirm Password). Below the password fields is a "Update Profile" button.

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)

The screenshot shows a web browser window with a profile update form. The URL in the address bar is <https://yc73-it202-008-dev-2cc697f868.herokuapp.com/Project/profile.php>. A yellow error message bar at the top of the page says "The chosen email is not available." The form has fields for Email (containing "carl1@test.com"), Username (containing "carl1"), and Password Reset (with three input fields for Current Password, New Password, and Confirm Password). Below the password fields is a "Update Profile" button.

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 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)

The chosen username is not available.

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 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)

The chosen username is not available.

Profile Update

Current Password

New Password

Confirm Password

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)

yc73-lt202-008-dev-2cca69f868.herokuapp.com/Project/profile.php

Home Profile Logout

Email

Username

Password Reset

Current Password

New Password

Confirm Password

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)

yc73-lt202-008-dev-2cca69f868.herokuapp.com/Project/profile.php

Home Profile Logout

Profile saved

Current password is invalid

Email: carl1@test.com

Username: carl1

Password Reset

Current Password:

New Password:

Confirm Password:

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)

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
<input type="checkbox"/> #1	1	Updating Username/Email
<input type="checkbox"/> #2	1	Updating password
<input type="checkbox"/> #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 alphanumeric, 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.

 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

Misc (1 pt.)

[^COLLAPSE ^](#)

Task #1 - Points: 1

Text: Screenshot of wakatime

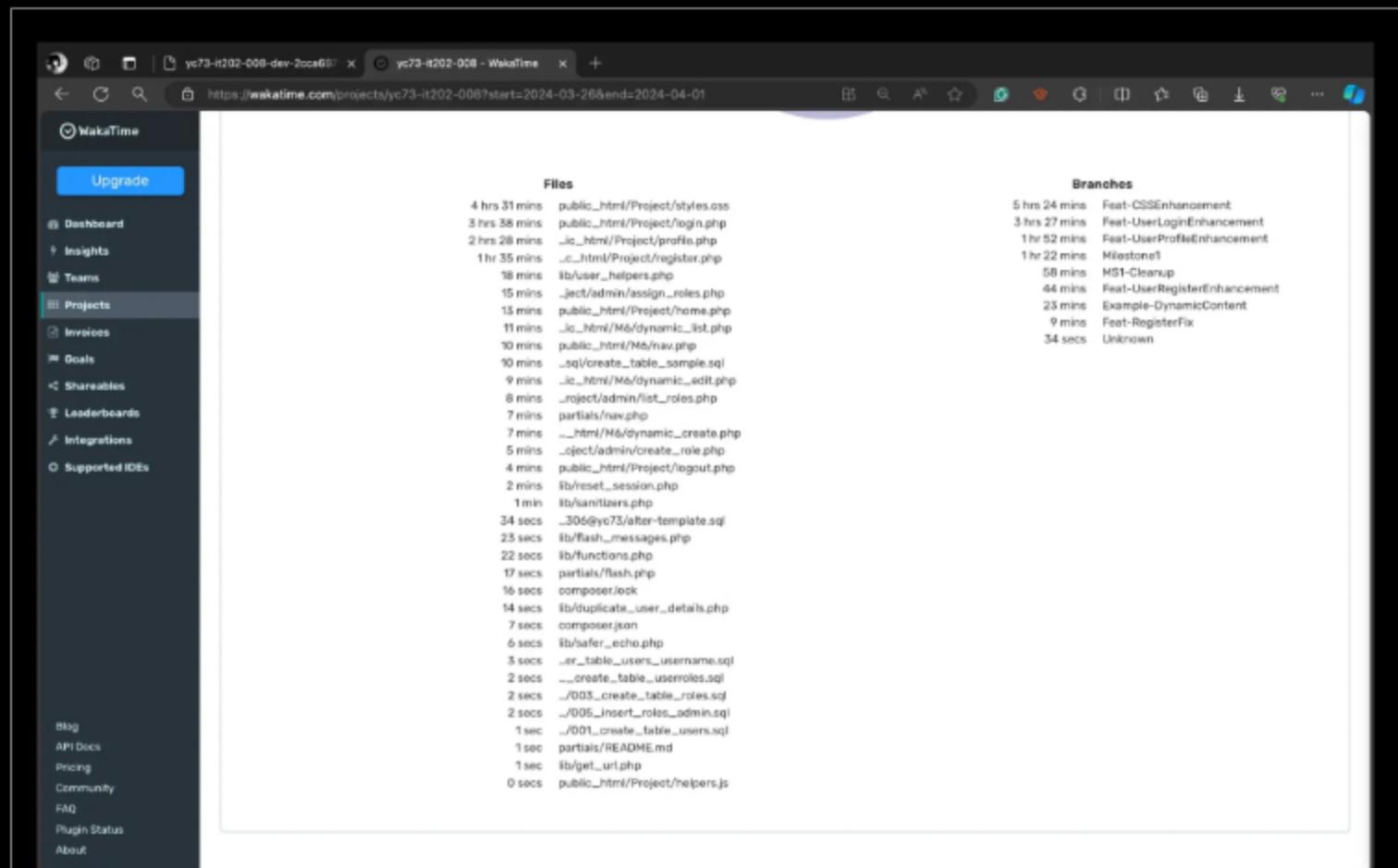
Task Screenshots:

Gallery Style: Large View

Small

Medium

Large



Showing wakatime

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

The screenshot shows a GitHub project board for 'Yanelli's IT202 Project'. The board has three columns: 'Todo' (empty), 'In Progress' (empty), and 'Done'. The 'Done' column contains a list of completed items, each with a checkmark icon and a link to a GitHub issue:

- yc73-it202-008 #31 - MS1 - User will be able to register a new account
- yc73-it202-008 #32 - MS1 - User will be able to log in to their account (given they enter the correct credentials)
- yc73-it202-008 #34 - MS1 - User will be able to logout
- yc73-it202-008 #36 - MS1 - Basic security rules implemented
- yc73-it202-008 #38 - MS1 - Basic Roles implemented
- yc73-it202-008 #40 - MS1 - Site should have basic style/theme applied; everything should be styled
- yc73-it202-008 #48 - MS1 - Any output messages/errors should be "user friendly"
- yc73-it202-008 #50 - MS1 - User will be able to see their profile
- yc73-it202-008 #40 - MS1 - User will be able to edit their profile

Showing project board (done)

Task #3 - Points: 1

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

URL #1

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

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