Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-007-F2024/it202-module-6-milestone-1-2024/grade/mcp62

Course: IT202-007-F2024
Assigment: [IT202] Module 6 Milestone 1 2024
Student: Michael P. (mcp62)

Submissions:

Submission Selection

1 Submission [submitted] 11/14/2024 2:02:50 PM

Instructions

COLLAPSE

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

Prereqs:

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

Instructions:

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

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

Branch name: Milestone1

Group

100%

Group: User Registration
Tasks: 6
Points: 2

COLLAPSE

Task

100%

Group: User Registration
Task #1: Screenshot of form on website page
Weight: ~17%
Points: ~0.33

COLLAPSE

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

Columns: 1

Sub-Task

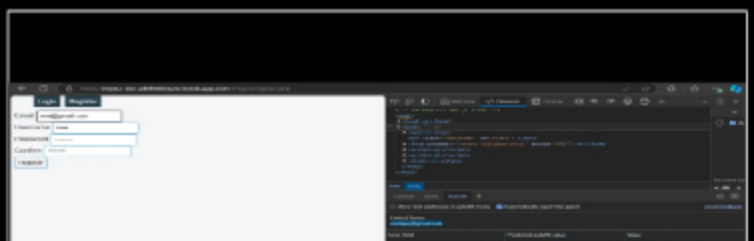
100%

Group: User Registration
Task #1: Screenshot of form on website page
Sub Task #1: Screenshot of the form (ensure valid data is filled in)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Registration form

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task URLs

URL #1

https://mcp62-dev-adbf9d9e4a42.herokuapp.com/Project/register.php

URL

https://mcp62-dev-adbf9d9e4a42.herokuapp.cc

Sub-Task

100%

Group: User Registration

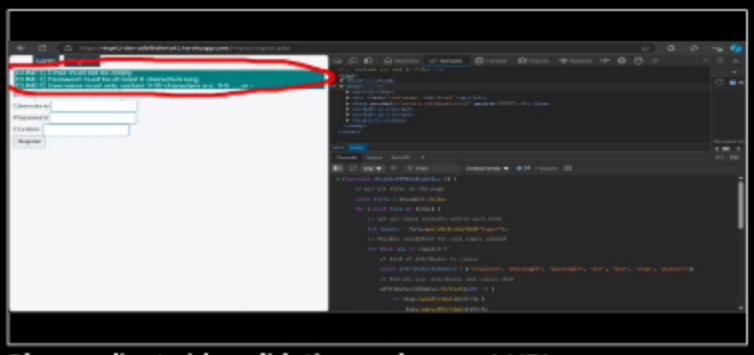
Task #1: Screenshot of form on website page

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Shows client side validation and correct URL

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: User Registration

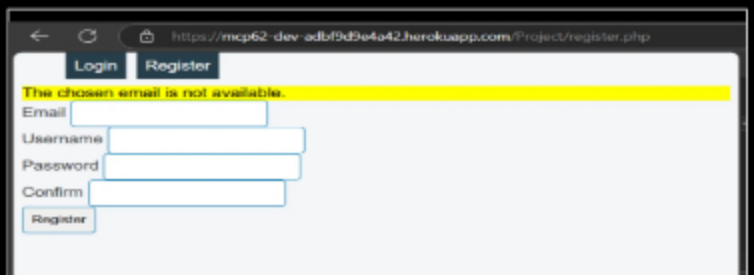
Task #1: Screenshot of form on website page

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Shows chosen email is not available (friendly data, does not indicate email is in use)

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task 100%

Group: User Registration
Task #1: Screenshot of form on website page
Sub Task #4: Demonstrate username already in use message (message text doesn't need to be exact, but should be clear)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Chosen username not available (friendly data so client does not know if username belongs to another person)

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

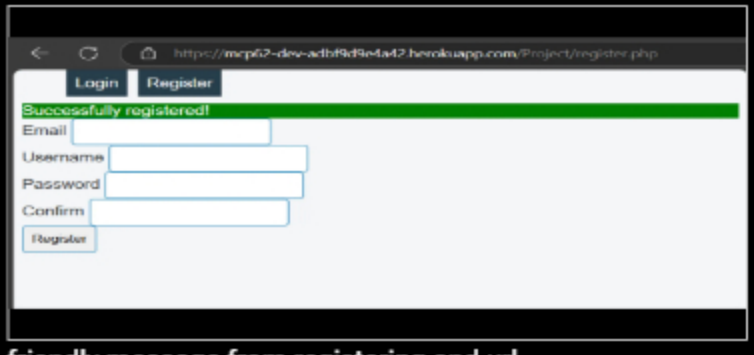
Sub-Task 100%

Group: User Registration
Task #1: Screenshot of form on website page
Sub Task #5: Demonstrate user-friendly message of new account being created

Task Screenshots

Gallery Style: 2 Columns

4 2 1



friendly message from registering and url

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 1

Task

100%

Group: User Registration
Task #2: Screenshot of the form code
Weight: ~17%
Points: ~0.33

COLLAPSE

Details:

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

Sub-Task

100%

Group: User Registration
Task #2: Screenshot of the form code
Sub Task #1: Show the html of the form

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
<?php
require(__DIR__ . "/../../partials/nav.php");
reset_session();
?>
<form onsubmit="return validate(this)" method="POST">
    <div>
        <label for="email">Email</label>
        <input type="email" name="email" required />
    </div>
    <div>
        <label for="username">Username</label>
        <input type="text" name="username" required maxlength="30" />
    </div>
    <div>
        <label for="pw">Password</label>
        <input type="password" id="pw" name="password" required minlength="8" />
    </div>
    <div>
        <label for="confirm">Confirm</label>
        <input type="password" name="confirm" required minlength="8" />
    </div>
    <input type="submit" value="Register" />
</form>
<script>
```

html

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the html for each field including the chosen attributes

Response:

The form element uses onsubmit="return validate(this)", which calls a JavaScript function for custom validation before submission, and method="POST" to send form data securely to the server. The email field is set to type="email", ensuring that the user enters a valid email address, and has the required attribute to make it mandatory. The username field, with type="text", allows the user to input plain text, and it also includes the required attribute to prevent submission without input. Additionally, maxlength="30" limits the username to 30 characters. The password field is of type="password", which hides the input, and is given the id="pw" to enable direct reference. The name="password" allows the server to identify the input, while required and minlength="8" enforce that the

password is at least 8 characters long. The confirm password field mirrors the password field, with type="password", name="confirm", and similar validation, ensuring the user re-enters their password correctly. Lastly, the submit button with type="submit" and value="Register" submits the form, completing the registration process when clicked. All these elements work together to ensure that the form captures and validates the necessary user information before submission.

End of Task 2

Task

100%

Group: User Registration
Task #3: Screenshot of the client-side and server-side validation code
Weight: ~17%
Points: ~0.33

COLLAPSE

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

Columns: 1

Sub-Task

100%

Group: User Registration
Task #3: Screenshot of the client-side and server-side validation code
Sub Task #1: Show the JavaScript validations of the form (include any extra files related if you made separate files)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

User registration validation JS

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

I have set up a series of if statments that checks a conditon for each element, if the condition is not met isValid is

set to false and the form does not get validated, before this a message of what went wrong gets flashed. If all conditions are met the form get submitted.

Sub-Task

100%

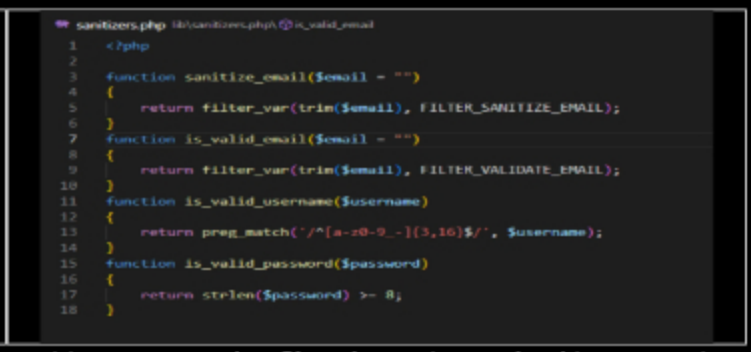
Group: User Registration
Task #3: Screenshot of the client-side and server-side validation code
Sub Task #2: Show the PHP validations (include any lib content)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

PHP validation



sanitizers are used to filter data prior to checking, regex is used to determine a valid format for the email address.

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

For the PHP Validation it happens on the server after the form is submitted. It validates the data after it's sent to the server, which means the user cannot bypass it (unless they manipulate the form submission). After the page reloads the flash messages are rendered via the HTML and are visible when the page reloads. The sanatizer contains functions that filter the raw data and get sent to the validation form where the filtered data is then checked with conditionals.

End of Task 3

Task

100%

Group: User Registration
Task #4: Screenshot of the Users table with a valid user entry
Weight: ~17%
Points: ~0.33

COLLAPSE

Checklist

*The checkboxes are for your own tracking

#	Details

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

Sub-Task

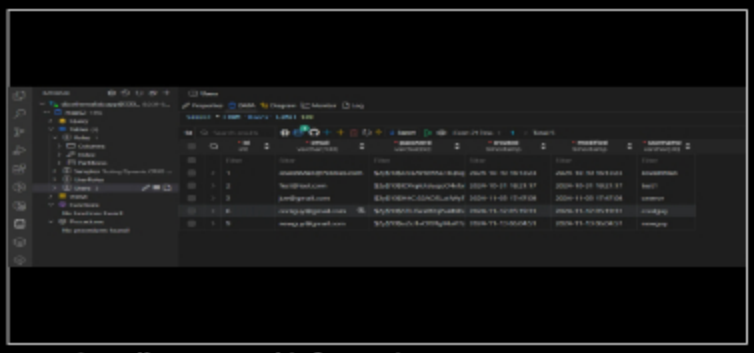
100%

Group: User Registration
Task #4: Screenshot of the Users table with a valid user entry
Sub Task #1: Show valid data per the checklist

Task Screenshots

Gallery Style: 2 Columns

4 2 1



contains all requested information

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

End of Task 4

Task

100%

Group: User Registration
Task #5: Explain the registration logic in a step-by-step manner from when the page loads to when the data is saved to the DB
Weight: ~17%
Points: ~0.33

COLLAPSE

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

May be worthwhile using a list.

Task Response Prompt

Response:

Check if Form Fields Are Set: The code first checks if the email, password, confirm password, and username fields are set in the $_POST array, checking that the form has been submitted.

Sanitize and Validate Email: Sanitize: The sanitize_email() function is used to remove unwanted characters from the email. Validate: The is_valid_email() function checks if the email is in a valid format. If the email is empty or invalid, an

error message is flashed. Validate Username: A regex expression is used to check that the username only contains the needed requirments. If invalid, an error message is displayed. Check Password and Confirmation: The code ensures that both the password and confirm password fields are not empty. It checks if the password is at least 8 characters long. The password and confirm password fields must match, or an error is shown. Hash Password and Save User Data: If there are no validation errors, the password is hashed using password_hash() with the BCRYPT algorithm. The email, hashed password, and username are then inserted into the Users database table. Error Handling for Database Operations: If an error occurs during the database insert, such as a duplicate email or username, a duplicate check function (users_check_duplicate) handles the error by displaying an appropriate message. Display Success Message: If registration is successful, a success message is flashed to the user.

Additionally note the db.php in the libs folder connects the project with the db, contained in this file is code thaat vaidates the connection.

End of Task 5

Task

100%

Group: User Registration
Task #6: Include pull request links related to this feature
Weight: ~17%
Points: ~0.33

COLLAPSE

Details:
Should end in /pull/#

Task URLs

URL #1
https://github.com/Onervv/mcp62-IT202-007/pull/19/files

URL
https://github.com/Onervv/mcp62-IT202-007/pul

End of Task 6

End of Group: User Registration
Task Status: 6/6

Group

100%

Group: User Login
Tasks: 4
Points: 2

COLLAPSE

Task

Group: User Login

100%

Task #1: Screenshot of form on website page
Weight: ~25%
Points: ~0.50

COLLAPSE

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

Columns: 1

Sub-Task

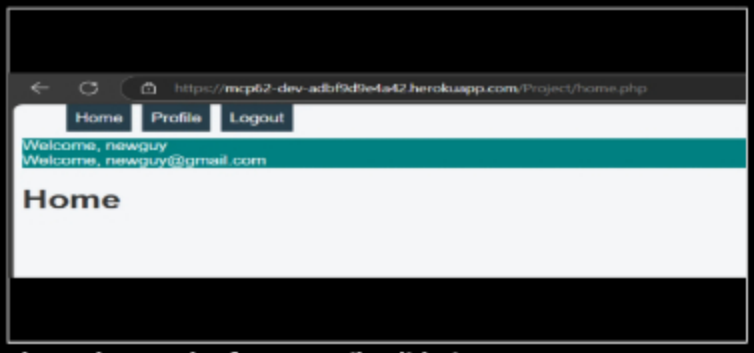
100%

Group: User Login
Task #1: Screenshot of form on website page
Sub Task #1: Two Screenshot of the form (one with valid email data filled and one with valid username data filled)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



shows home.php from Email validation

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task URLs

Sub-Task

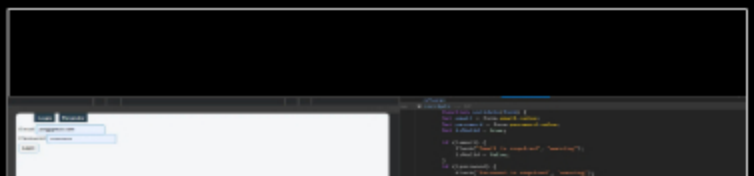
100%

Group: User Login
Task #1: Screenshot of form on website page
Sub Task #2: Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



JS validation for email and password

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

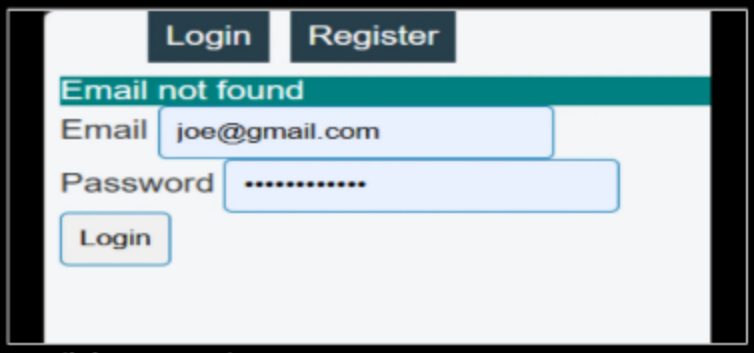
Sub-Task 100%

Group: User Login
Task #1: Screenshot of form on website page
Sub Task #3: Demonstrate user-friendly message of when an account doesn't exist

Task Screenshots

Gallery Style: 2 Columns

4 2 1



email does not exist

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

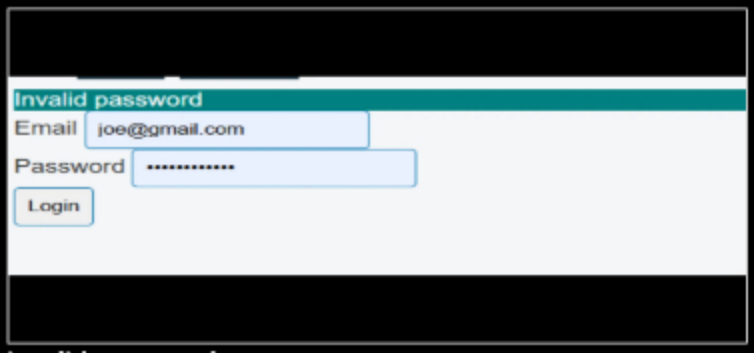
Sub-Task 100%

Group: User Login
Task #1: Screenshot of form on website page
Sub Task #4: Demonstrate user-friendly message of when password doesn't match what's in the DB

Task Screenshots

Gallery Style: 2 Columns

4 2 1



invalid password

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

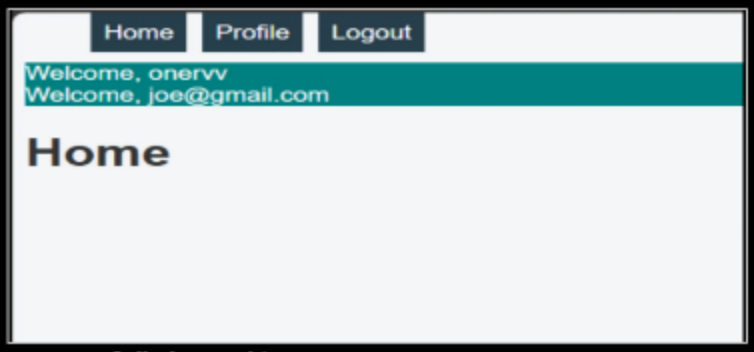
Sub-Task 100%

Group: User Login
Task #1: Screenshot of form on website page
Sub Task #5: Demonstrate successful login message and the destination page (i.e., home or some landing/dashboard page)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



successfully logged in

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task 100%

Group: User Login
Task #1: Screenshot of form on website page
Sub Task #6: Demonstrate session data being set (captured from server logs)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

shows cookies stored

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 1

Task

Group: User Login
Task #2: Screenshot of the form code

100%

Weight: ~25%
Points: ~0.50

COLLAPSE

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

Columns: 1

Sub-Task

100%

Group: User Login
Task #2: Screenshot of the form code
Sub Task #1: Show the html of the form

Task Screenshots

Gallery Style: 2 Columns

4 2 1

user login html

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the html for each field including the chosen attributes

Response:

This form collects an email and password, checks them with JavaScript, and sends the data securely on submission. Form Setup: The form tag uses method="POST" to securely send data to the server and has onsubmit="return validate(this)" to run JavaScript validation before submission. Email Field: label and input type="email" name="email" required /> have an email entry that requires a valid email format. Password Field: label and input type="password" name="password" required minlength="8" /> create a password field that's hidden as you type, requires a minimum of 8 characters, and is mandatory. Submit Button: input type="submit" value="Login" /> is the button to submit the form.

Sub-Task

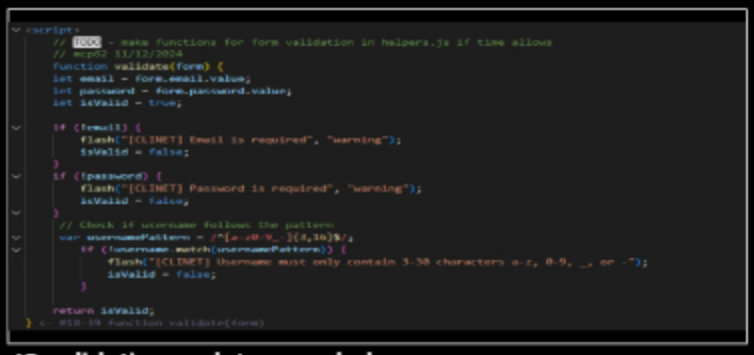
100%

Group: User Login
Task #2: Screenshot of the form code
Sub Task #2: Show the JavaScript validations of the form (include any extra files related if you made separate files)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



JS validation updates needed

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

Main Purpose: The validate function ensures the email and password fields are not empty before form submission. User Feedback: Displays warnings (via flash function) if fields are missing. Control Submission: Stops form submission if either field is empty.

Sub-Task

100%

Group: User Login

Task #2: Screenshot of the form code

Sub Task #3: Show PHP validations (include any lib content)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

php validation, same lib content from before is used here

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

Checks Fields: Ensures email and password fields are submitted. Sanitizes & Validates: Sanitizes the email, then validates its format and password requirements. Handles Errors: Sets $hasError if any validation fails. Proceeds if Valid: Continues with login if no errors are found.

End of Task 2

Task

100%

Group: User Login
Task #3: 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
Weight: ~25%
Points: ~0.50

COLLAPSE

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

May be worthwhile using a list.

Task Response Prompt

Response:

When the login page loads and the user submits their email/username and password, the server retrieves the data via $_POST, validating that both fields are present. It sanitizes the email/username, checks its format, and verifies the password meets requirements. If validation succeeds, the server queries the database for a matching user and, if found, compares the submitted password to the stored hashed password. On success, the server initiates a session for the user, storing relevant user data, and redirects them to their dashboard or a logged-in area.

End of Task 3

Task

100%

Group: User Login
Task #4: Include pull request links related to this feature
Weight: ~25%
Points: ~0.50

COLLAPSE

Details:
Should end in /pull/#

Task URLs

URL #1
https://github.com/Onervv/mcp62-IT202-007/pull/19

https://github.com/Onervv/mcp62-IT202-007/pul

End of Task 4

End of Group: User Login
Task Status: 4/4

Group

100%

Group: User Logout
Tasks: 2
Points: 1

COLLAPSE

Task

100%

Group: User Logout
Task #1: Capture the following screenshots
Weight: ~50%
Points: ~0.50

COLLAPSE

Columns: 1

Sub-Task

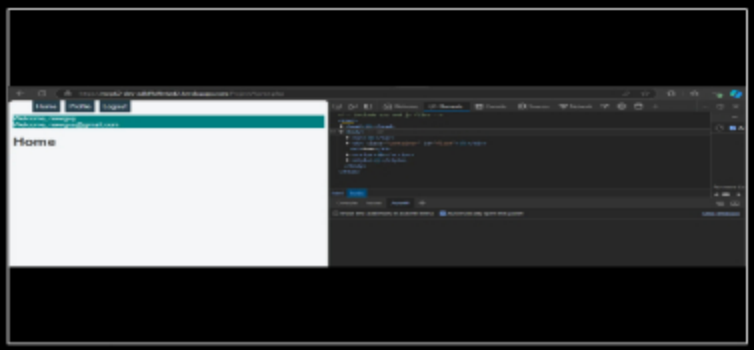
100%

Group: User Logout
Task #1: Capture the following screenshots
Sub Task #1: Screenshot of the navigation when logged in (site)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



logged in site

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Sub-Task

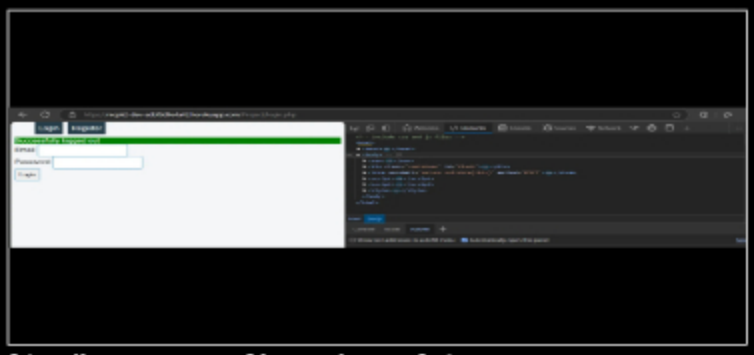
100%

Group: User Logout
Task #1: Capture the following screenshots
Sub Task #2: Screenshot of the redirect to login with the user-friendly logged-out message (site)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



friendly message of logged out of site

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task 100%

Group: User Logout

Task #1: Capture the following screenshots

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

Task Screenshots

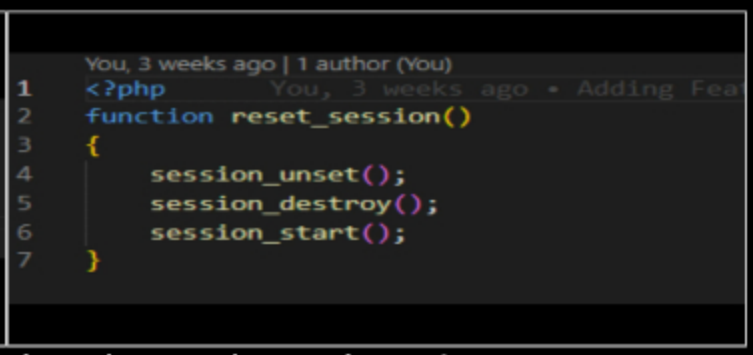
Gallery Style: 2 Columns

4 2 1

```
logout.php public_html/Project/logout.php
You, 4 weeks ago | 1 author (You)
<?php
session_start();
require(__DIR__ . "/../../lib/functions.php");
reset_session();

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

Logout code



```
You, 3 weeks ago | 1 author (You)
<?php
function reset_session()
{
    session_unset();
    session_destroy();
    session_start();
}
```

Shows that reset destroys the session

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

The reset_session() function first clears all session variables with session_unset(), then fully destroys the session using session_destroy(). This removes any data tied to the user's session, effectively logging them out. Afterward, the flash() function displays a "Successfully logged out" message, and the user is redirected to the login page via header("Location: login.php").

End of Task 1

Task

100%

Group: User Logout
Task #2: Include pull request links related to this feature
Weight: ~50%
Points: ~0.50

COLLAPSE

Details:
Should end in /pull/#

Task URLs

URL #1
https://github.com/Onervv/mcp62-IT202-007/pull/22

URL
https://github.com/Onervv/mcp62-IT202-007/pul

End of Task 2

End of Group: User Logout
Task Status: 2/2

Group

100%

Group: Basic Security Rules and Roles
Tasks: 5
Points: 2

COLLAPSE

Task

100%

Group: Basic Security Rules and Roles
Task #1: Authentication Screenshots
Weight: ~20%
Points: ~0.40

COLLAPSE

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

Columns: 1

Sub Task

Sub-Task

100%

Group: Basic Security Rules and Roles
Task #1: Authentication Screenshots
Sub Task #1: Screenshot of the function that checks if a user is logged in

Task Screenshots

Gallery Style: 2 Columns

4 2 1

log in function

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

The get_user_id function checks if a user is logged in. If they are, it retrieves and returns their user ID from the session. If not, it returns false. It uses two helper functions: is_logged_in() to verify login status and se() to safely access the user ID in the session data.

Sub-Task

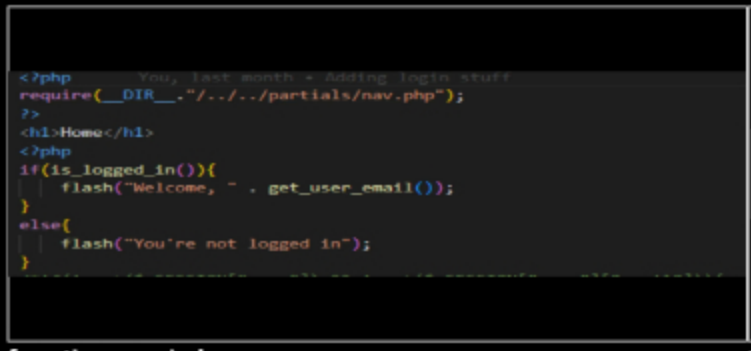
100%

Group: Basic Security Rules and Roles
Task #1: Authentication Screenshots
Sub Task #2: Screenshot of the login check function being used (i.e., profile likely). Also, caption what pages it's used on

Task Screenshots

Gallery Style: 2 Columns

4 2 1



function use in home

Profile

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

Both of these function calls just check to see if user is logged in from refrenced logic if true the rest of the form loads if false minimal data is shown and a message is flashed.

Sub-Task

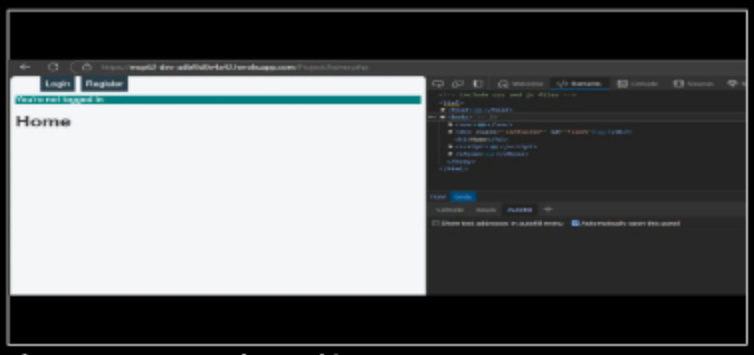
100%

Group: Basic Security Rules and Roles
Task #1: Authentication Screenshots
Sub Task #3: Demonstrate the user-friendly message of trying to manually access a login-protected page while being logged out (must show the appropriate message)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



shows you are not logged in

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 1

Task

100%

Group: Basic Security Rules and Roles
Task #2: Authorization Screenshots
Weight: ~20%
Points: ~0.40

COLLAPSE

Details:

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

Columns: 1

Sub-Task

100%

Group: Basic Security Rules and Roles
Task #2: Authorization Screenshots

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

checks if the user has a role

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

The has_role function checks if a logged-in user has a specific role. It first verifies that the user is logged in and that the session contains a "roles" array under $_SESSION["user"]. Then, it iterates through each role in this array and checks if any role's name matches the specified $role. If a match is found, it returns true; otherwise, it returns false.

Sub-Task

100%

Group: Basic Security Rules and Roles

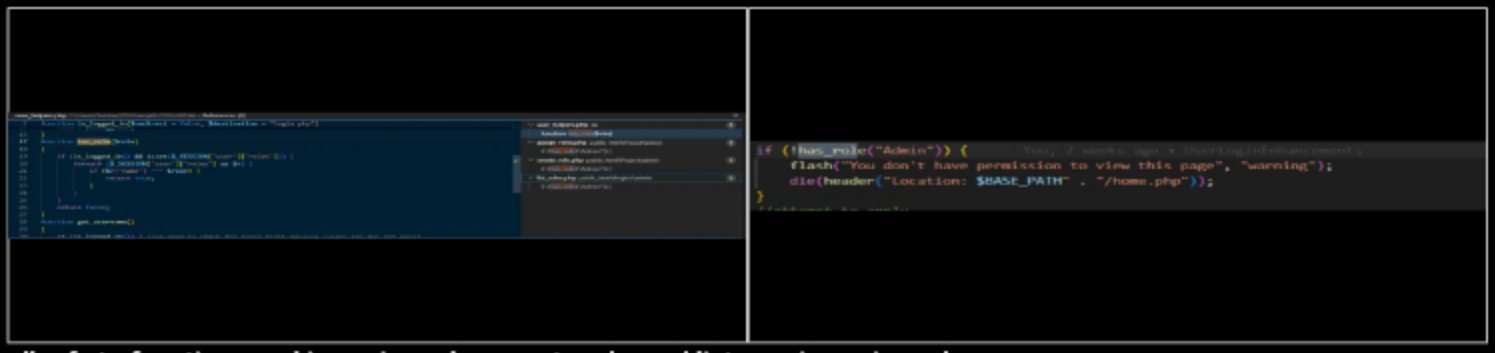
Task #2: Authorization Screenshots

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



all refs to function used in assign roles, create role, and list use in assign roles

roles

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

checks if the user does not have the "Admin" role by calling the has_role function. If the user isn't an admin, it displays a warning message using the flash function, telling them they don't have permission to view the page. Then, it redirects the user to the home page (home.php) using the $BASE_PATH variable. The die function immediately stops further code execution after the redirect is set.

Sub-Task

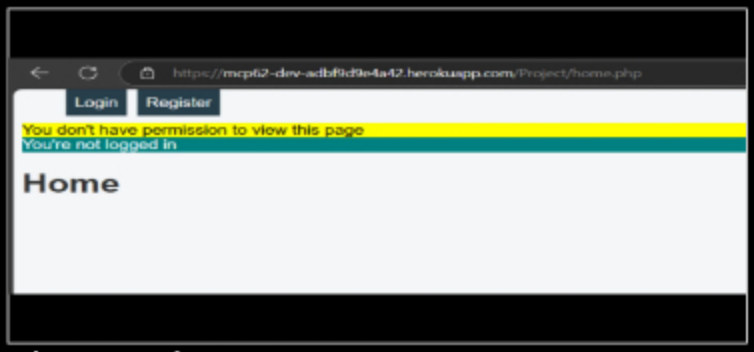
100%

Group: Basic Security Rules and Roles
Task #2: Authorization Screenshots
Sub Task #3: Demonstrate the user-friendly message of trying to manually access a role-protected page while being logged out (must show the appropriate message)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



role protected page message

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 2

Task

100%

Group: Basic Security Rules and Roles
Task #3: Screenshots of UserRoles and Roles Tables
Weight: ~20%
Points: ~0.40

COLLAPSE

Details:

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

Columns: 1

Sub-Task

Group: Basic Security Rules and Roles

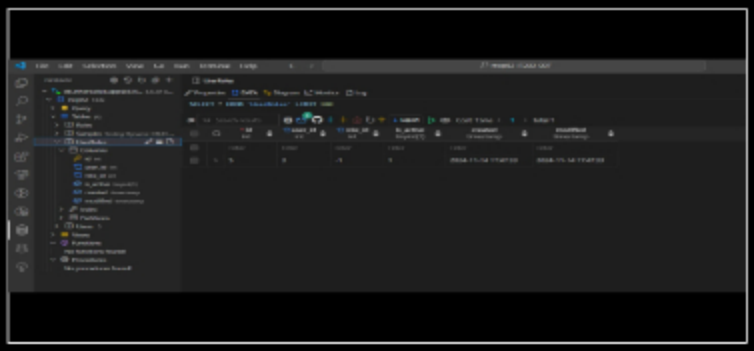
100%

Task #3: Screenshots of UserRoles and Roles Tables
Sub Task #1: UserRoles table with at least one valid entry (Table should have id, user_id, role_id, is_active, created, and modified columns)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Valid user roles entry

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

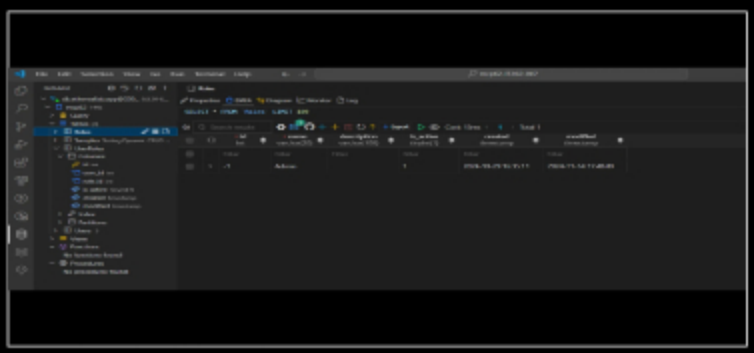
100%

Group: Basic Security Rules and Roles
Task #3: Screenshots of UserRoles and Roles Tables
Sub Task #2: Roles table with at least one valid entry (Table should have id, name, description, is_active, modified, and created columns)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Roles table with valid entry

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 3

Task

100%

Group: Basic Security Rules and Roles
Task #4: Explain how Roles and UserRoles tables work in conjunction with the Users table
Weight: ~20%
Points: ~0.40

COLLAPSE

Columns: 1

Sub-Task

100%

Group: Basic Security Rules and Roles
Task #4: Explain how Roles and UserRoles tables work in conjunction with the Users table
Sub Task #1: UserRoles

Task Response Prompt

What's the purpose of the UserRoles table?

Response:

UserRoles has a self assgined ID, User_ID connects to the USER_ID in the USER table. ROLE_ID linked to the Roles table value which in this case -1 refs to the role Admin. time stamps are also auto assigned.

Sub-Task

100%

Group: Basic Security Rules and Roles
Task #4: Explain how Roles and UserRoles tables work in conjunction with the Users table
Sub Task #2: Roles

Task Response Prompt

How does Roles.is_active differ from UserRoles.is_active?

Response:

Roles connects to the other tables, but primarly the code, within this table a ID needs to be set and a name to connect the level/amount of permissions the USER will have, filter connects back to the code where 1 reepresents the role being active and 0 represents the role being turned off.

End of Task 4

Task

100%

Group: Basic Security Rules and Roles
Task #5: Include pull request links related to this feature
Weight: ~20%
Points: ~0.40

COLLAPSE

Details:
Should end in /pull/#

Task URLs

URL #1
https://github.com/Onervv/mcp62-IT202-007/pull/26

URL
https://github.com/Onervv/mcp62-IT202-007/pul

End of Task 5

End of Group: Basic Security Rules and Roles
Task Status: 5/5

Group

100%

Group: User Profile
Tasks: 5
Points: 2

COLLAPSE

Task

100%

Group: User Profile
Task #1: View Profile Website Page
Weight: ~20%
Points: ~0.40

COLLAPSE

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

Sub-Task

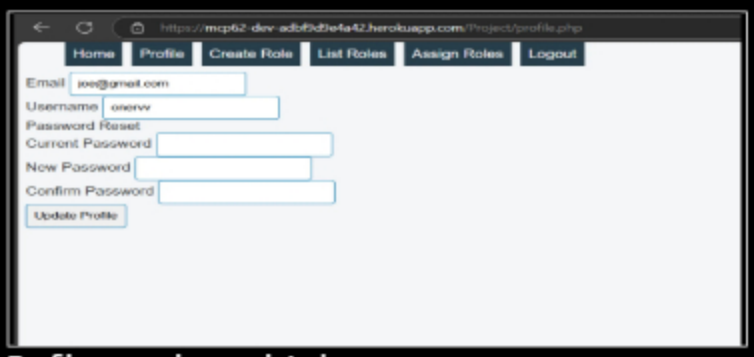
100%

Group: User Profile
Task #1: View Profile Website Page
Sub Task #1: Show the profile form correctly populated on page load (username, email) Form should have the following fields: username, email, current password, new password, confirm password (or similar)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Profile properly populated

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

End of Task 1

Task

100%

Group: User Profile
Task #2: Explain the logic step-by-step of how the data is loaded and populated when the profile page is visited
Weight: ~20%
Points: ~0.40

COLLAPSE

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

May be worthwhile using a list.

Task Response Prompt

Response:

When the form is submitted, the script first sanitizes and validates the email and username, displaying error messages if the input is invalid. If there are no errors, it updates the user's email and username in the database, then refreshes the session data. If the user is resetting their password, the script verifies the current password and checks that the new passwords match and meet the security criteria before updating the password in the database. Client-side validation is also included via JavaScript to check the email, username, and password fields before submission.

End of Task 2

Task

100%

Group: User Profile
Task #3: Edit Profile Website Page
Weight: ~20%
Points: ~0.40

COLLAPSE

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

Columns: 1

Sub-Task

100%

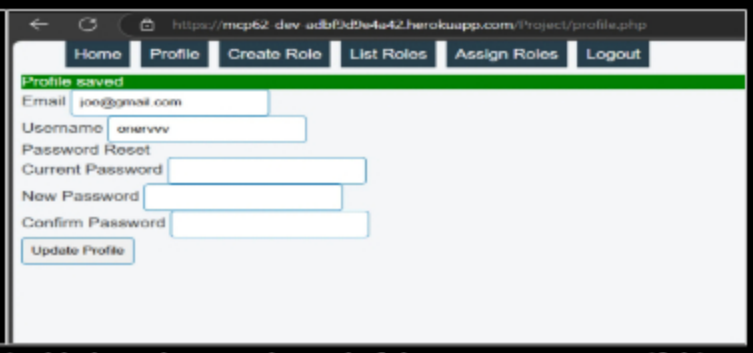
Group: User Profile
Task #3: Edit Profile Website Page
Sub Task #1: (Two Screenshots) Demonstrate with before and after of a username change (including success message)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Before



i added another v to the end of the userName, sorry if this isnt an obvious change

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: User Profile

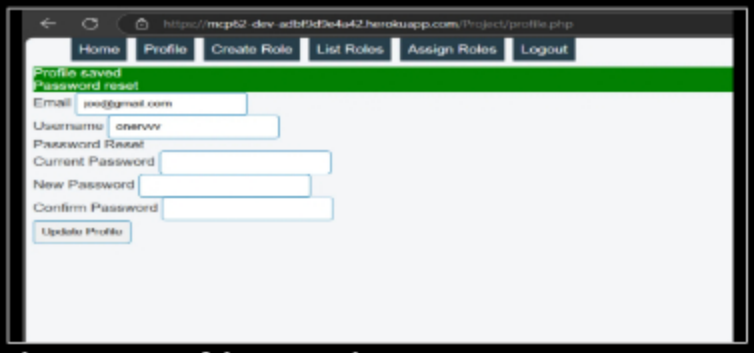
Task #3: Edit Profile Website Page

Sub Task #2: Demonstrate the success message of updating password

Task Screenshots

Gallery Style: 2 Columns

4 2 1



shows successful password reset

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: User Profile

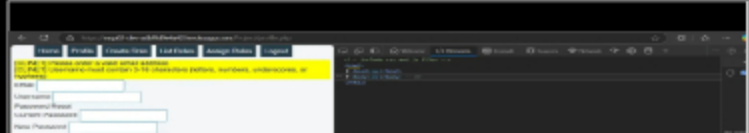
Task #3: Edit Profile Website Page

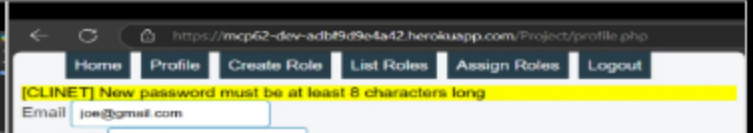
Sub Task #3: Demonstrate all JavaScript user-friendly validation messages [can be combined] (email format, username format, password format, new password matching confirm password)

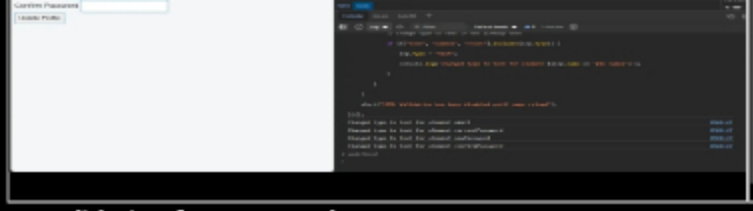
Task Screenshots

Gallery Style: 2 Columns

4 2 1







JS validation for empty values.

Pass format validation

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: User Profile

Task #3: Edit Profile Website Page

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

email not valid due to being taken message

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: User Profile

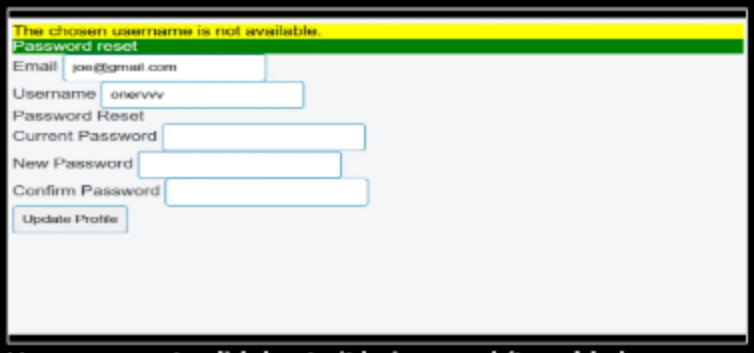
Task #3: Edit Profile Website Page

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Username not valid due to it being used (I could change this message to be more clear but still contains the

message)

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: User Profile

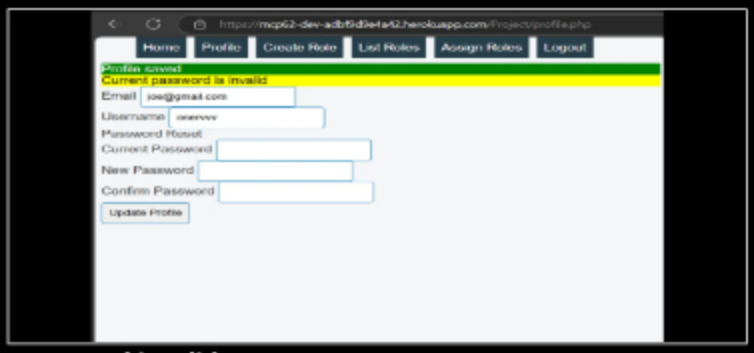
Task #3: Edit Profile Website Page

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Password invalid message

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 3

Task

100%

Group: User Profile

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

Weight: ~20%

Points: ~0.40

COLLAPSE

Details:

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

Columns: 1

Sub-Task

100%

Group: User Profile

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

Sub Task #1: Updating Username/Email

Task Response Prompt

Explain in concise steps how this logically works

Response:

updates a user's password after verifying their current one. It retrieves the current, new, and confirmed new passwords from a form submission and first checks if all required fields are populated. Then, it validates that the new password meets certain criteria (using is_valid_password). If the new passwords match, it fetches the stored password hash for the user and verifies the entered current password. If the current password is valid, it updates the database with the new password (hashed for security).

Sub-Task

100%

Group: User Profile
Task #4: Explain the logic step-by-step of how the data is checked and saved for the following scenarios
Sub Task #2: Updating password

Task Response Prompt

Explain in concise steps how this logically works

Response:

updates a user's password by verifying the current password and checking that the new password meets criteria and matches the confirmation password. If the current password is correct, it securely hashes the new password and updates it in the database, providing success or error messages based on the outcome of each step

End of Task 4

Task

100%

Group: User Profile
Task #5: Include pull request links related to this feature
Weight: ~20%
Points: ~0.40

COLLAPSE

Details:
Should end in /pull/#

Task URLs

URL #1
https://github.com/Onervv/mcp62-IT202-007/pull/22

URL
https://github.com/Onervv/mcp62-IT202-007/pul

End of Task 5

End of Group: User Profile
Task Status: 5/5

Group

Group: Misc
Tasks: 3
Points: 1

100%

COLLAPSE

Task

Group: Misc
Task #1: Screenshot of wakatime
Weight: ~33%
Points: ~0.33

100%

COLLAPSE

Details:

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

waka

End of Task 1

Task

Group: Misc
Task #2: Screenshot of your project board from GitHub (tasks should be in the proper column)
Weight: ~33%
Points: ~0.33

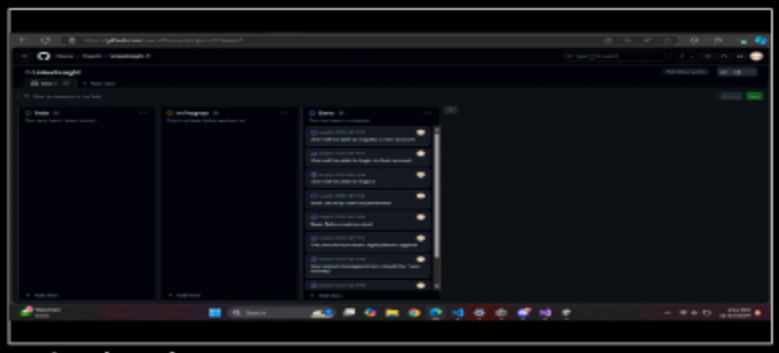
100%

COLLAPSE

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Project board

End of Task 2

Task

100%

Group: Misc
Task #3: Provide a direct link to the project board on GitHub
Weight: ~33%
Points: ~0.33

COLLAPSE

Task URLs

URL #1
https://github.com/users/Onervv/projects/1

URL
https://github.com/users/Onervv/projects/1

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment