Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-007-F2024/it202-module-6-milestone-1-2024/grade/jd755

Course: IT202-007-F2024
Assigment: [IT202] Module 6 Milestone 1 2024
Student: Jeremy R. (jd755)

Submissions:

Submission Selection

1 Submission [submitted] 11/9/2024 6:59:45 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

98%

Group: User Registration
Tasks: 6
Points: 2

COLLAPSE

Task

90%

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

50%

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 page

Caption(s) (required) ✓

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

Task URLs

URL #1

Missing URL

URL

Sub-Task

100%

Group: User Registration

Task #1: Screenshot of form on website page

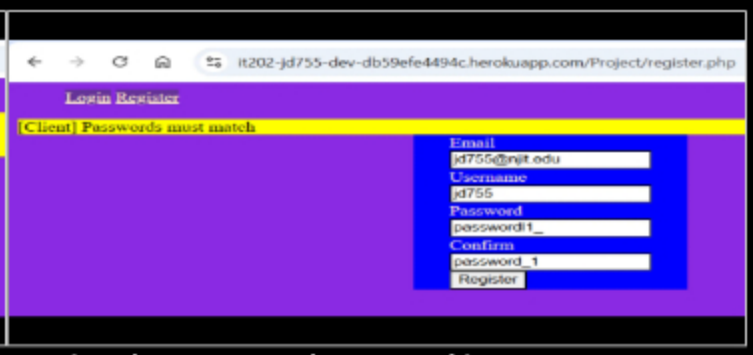
Sub Task #2: Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Client side warnings for invalid email, username, and password



Warning about passwords not matching

Empty fields

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

Warning when email is already used

Caption(s) (required) ✓

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

Sub-Task

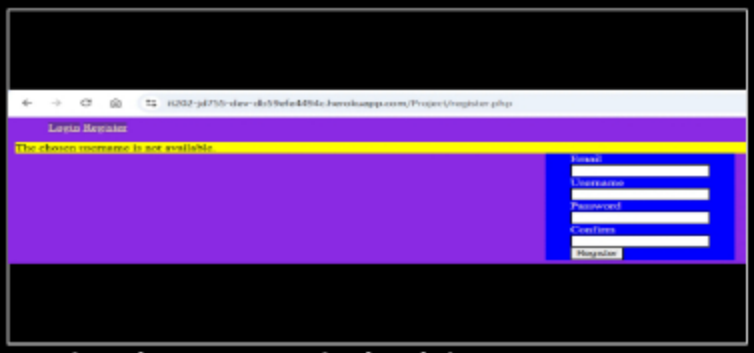
100%

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



Warning when username is already in use

Caption(s) (required) ✓

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

Sub-Task

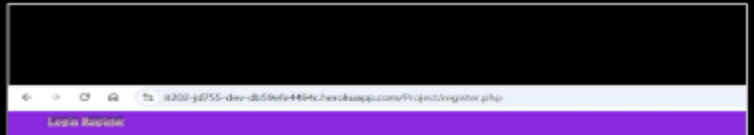
100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Message when regisration is successful

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

```
<!-- jd55 11/09/24 -->
<?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>
```

HTML screenshot 1

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 HTML consists of a form element that contains 4 div elements. The form element is of method post as we don't want the data of the form to easily accessable. The onsubmit attribute is set to call the javascript function 'validate'. The first div element holds a label for email and has an input of type email. The other 3 div elements follow a similar pattern but for username, password, and confirm password respectively.

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

Client side validation 1

Client side validation 2

```
//jd755 11/05/24 <- #1-16 function flash(message = "", color = "info")
function validate_email(email)
{
    console.log("Inside validate email");
    emailRegEx = /^([a-z0-9_\.\+-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/;
    return (emailRegEx.test(email));
} <- #18-22 function validate_email(email)
function validate_username(username)
{
    usernameRegEx = /^[a-z0-9_-]{3,16}$/;
    return (usernameRegEx.test(username));
}
```

Two new functions I added in helpers.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 started by getting all of the values necessary from the HTML form element by using the .value property. Email I first check if the length of the email is 0. If it is, I send a warning saying that the mail field cannot be blank. If it's not blank, I pass the email to the validate email function I created in helpers.js. If it returns false, I send a warning that the email is invalid. Username I used the same method as the email above but I used the validate username function I created in helpers.js and my warning message explains what a valid username must consist of. Password & Confirm Password I check if the password and confirm password is equal to 0. If it is, I sent out appropriate warnings that they cannot be empty. After, I check if password is less than 8. If it is, I send a warning saying the password can be less than 8 characters. Finally, I check if password and confirm password are equal. If they aren't, I send a warning.

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 2

PHP validation 1

```php
<?php

function sanitize_email($email = "")
{
    return filter_var(trim($email), FILTER_SANITIZE_EMAIL);
}
function is_valid_email($email = "")
{
    return filter_var(trim($email), FILTER_VALIDATE_EMAIL);
}
function is_valid_username($username)
{
    return preg_match('/^[a-z0-9_-]{3,16}$/', $username);
}
function is_valid_password($password)
{
    return strlen($password) >= 8;
}
```

sanitizers.php in the lib folder

duplicate_user_details.php inside the lib folder

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The first part of the validation works in a similar way as the JavaScript validation from before. However, this time after the validation is complete and everything is fine, an sql statement is created and sent to the database. If this brings no errors, then a success statement is sent to the user. If not, the function duplicate_user_details.php is called which sends to the user the error that the chosen email/username was already taken.

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

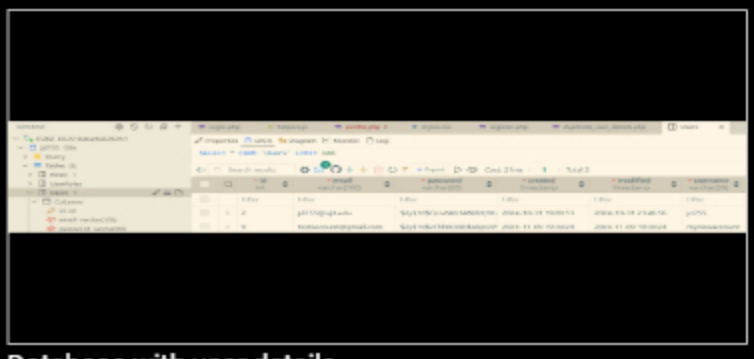
100%

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



Database with user details

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:

1. User is prompted to enter an email, username, and password
2. Clientside and server side validation is used to check if the data entered into these fields are valid
3. Data is sent to database if it is valid. If the database detects a duplicate value, send user a warning message
4. If everything is clear, tell the user the regisration is complete. Now the new data is inside the database.

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/legitdaylight/jd755-IT202-007/pull/19

URL
https://github.com/legitdaylight/jd755-IT202-007

URL #2
https://github.com/legitdaylight/jd755-IT202-007/pull/21

URL
https://github.com/legitdaylight/jd755-IT202-007

URL #3

https://github.com/legitdaylight/jd755-IT202-007/pull/22

https://github.com/legitdaylight/jd755-IT202-007

End of Task 6

End of Group: User Registration
Task Status: 5/6

Group

100%

Group: User Login
Tasks: 4
Points: 2

COLLAPSE

Task

100%

Group: User Login
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

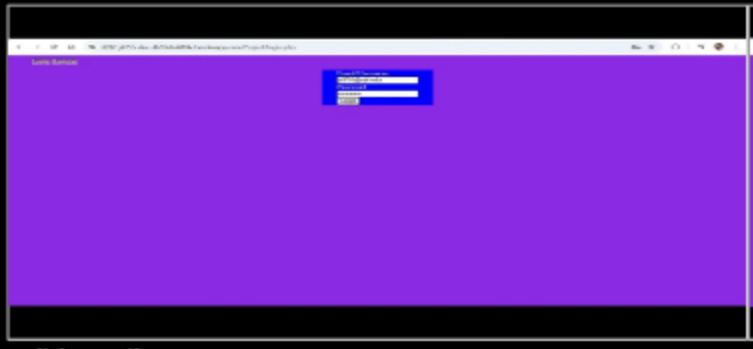
100%

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)

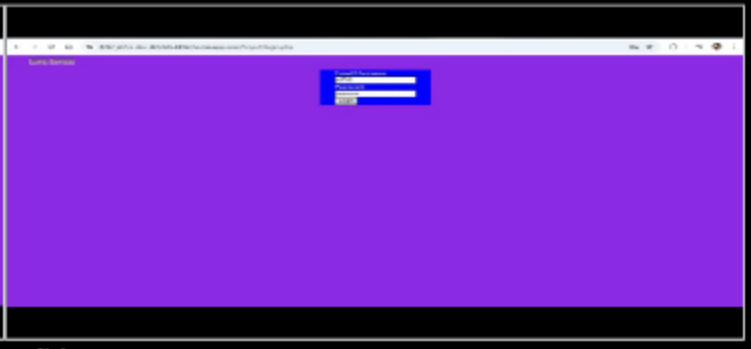
Task Screenshots

Gallery Style: 2 Columns

4 2 1



valid email



Valid username

Caption(s) (required) ✓

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

Task URLs

Sub-Task

100%

Group: User Login

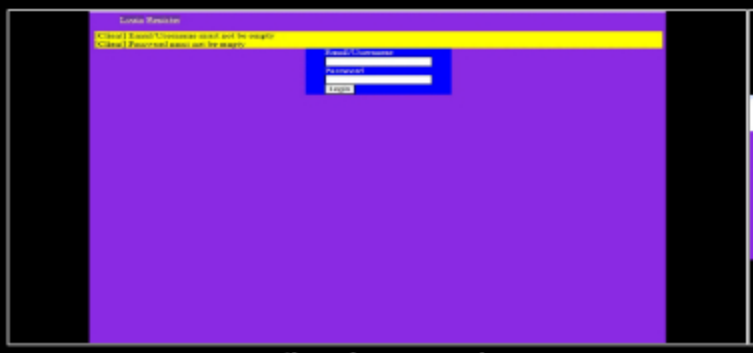
Task #1: Screenshot of form on website page

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

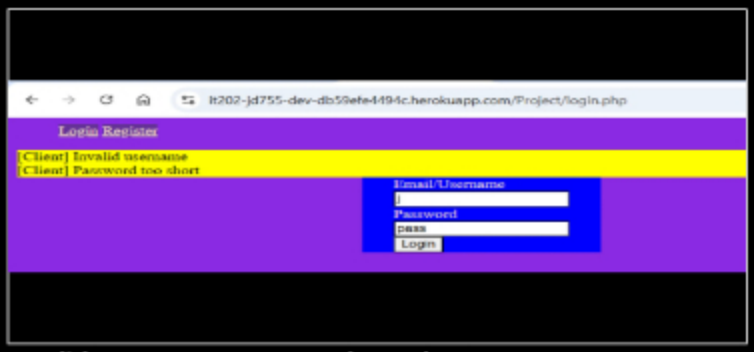
Gallery Style: 2 Columns

4 2 1



Empty username, email, and password

Invalid email



Invalid username, password too short

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/username not found

Caption(s) (required) ✓

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

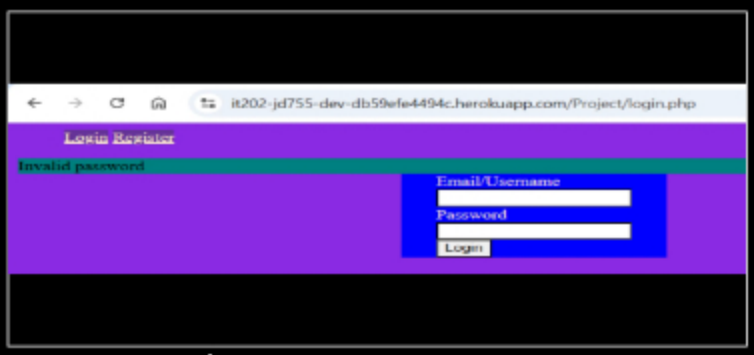
Sub-Task 100%

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



wrong password

Caption(s) (required) ✓

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

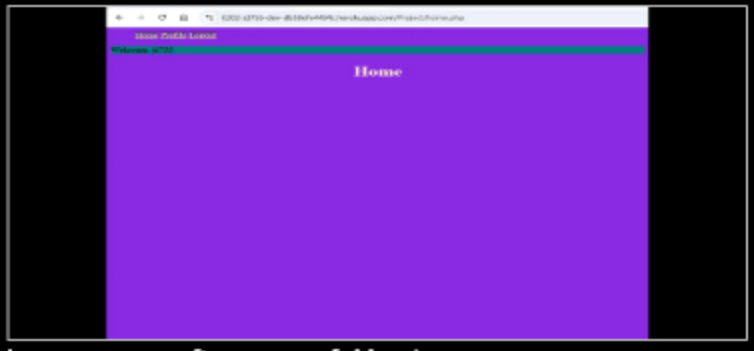
Sub-Task 100%

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



home screen after sucessful log 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

session data in the logs

Caption(s) (required) ✓

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

End of Task 1

Task

100%

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

```
//jd755 11//09/24
<form onsubmit="return validate(this)" method="POST">
    <div>
        <label for="email">Email/Username</label>
        <input type="text" name="email" required />
    </div>
    <div>
        <label for="pw">Password</label>
        <input type="password" id="pw" name="password" required minlength="8" />
    </div>
    <input type="submit" value="Login" />
</form>
```

HTML of the form

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 HTML consists of a form element of method post and an onsubmit attribute that calls the JavaScript validate function. The form contains two divs: one for the email/password input and the other for the password input.

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 1

```
47        //jd755 11/09/24
48        if(password.length == 0)
49        {
50            flash("[Client] Password must not be empty", "warning");
51            notError = false;
52        }
53        else
54        {
55            if(password.length < 8)
56            {
57                flash("[Client] Password too short", "warning");
58                notError = false;
59            }
60        } <- #54-60 else
61        //TODO update clientside validation to check if it should
62        //valid email or username
63        return notError;
64    } <- #16-64 function validate(form)
65 </script>
```

JS Validation 2

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Email/Username I first check if the email/password field is empty. If it is, I send a warning to the user that the username/password can't be empty. If it is not empty, I check if the email contains the character '@'. If it does, I call the validate email function I created in helpers.js and check if the email is valid. If the email does not contain that character, I treat it as a username and I called the validate username function in helpers.js. Password I check if the password field is 0. If it is, I send a warning to the user that the password fifeld cannot be empty. If it is not empty, I

check if the length is lesss than 8. If it is, I send a warning to the user that the password is too short.

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 1

PHP Validation 2

PHP Validation 3

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The basic validation works the same as above in the JavaScript. When this validation is complete with 0 errors, an SQL statement is created with all the data inputted and is sent to the database. If the SQL statement doesn't return anything, then this means that username/email doesn't exist. This sends a warning to the user. IF the username/email does exist but if the password is wrong, then another warning is sent. If everything goes well, the user is redirected to the home page with a welcome message containing their username.

End of Task 2

Task

Group: User Login
Task #3: Explain the login logic step-by-step from when the page loads to when the data is

100%

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:

1. The user enters in their email/username or password. Basic validation is done to check if these inputs are valid.
2. If the inputs are valid, this information is sent to the database. If the email/username doesn't exist or if the password doesn't match, a warning is sent to the user.
3. If the database check was sucessful, the user is redirected to the home screen.

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/legitdaylight/jd755-IT202-007/pull/22

URL
https://github.com/legitdaylight/jd755-IT202-007

URL #2
https://github.com/legitdaylight/jd755-IT202-007/pull/20

URL
https://github.com/legitdaylight/jd755-IT202-007

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

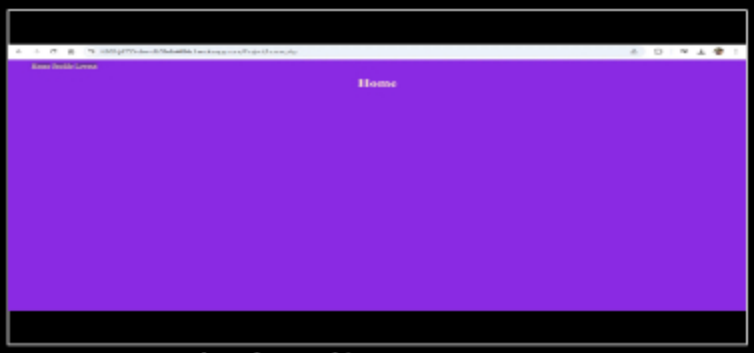
100%

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



Home screen when logged in

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

Logging out

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

```php
//jd755 11/09/24
<?php
session_start();
require(__DIR__ . "/../../lib/functions.php");
reset_session();

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

Log Out code

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The details of the session are first deleted. Then the user is redirected to the log in page and a flash message appears stating they were logged out.

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/legitdaylight/jd755-IT202-007/pull/20

URL
https://github.com/legitdaylight/jd755-IT202-007

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

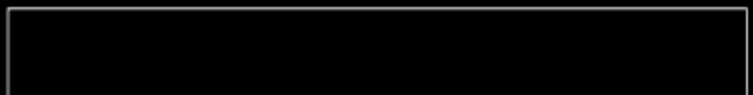
100%

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



Function to check if user is logged in

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The function accepts two parameters: redirect which is defaulted to false, and destination which is defaulted to login.php. The function starts by checking if the session with the current user exists. If it doesn't exist and if redirect is true, a warning is diplayed to the user that they must be logged in to view that page and they are automatically redirected to login.php. If redirect is false, then the user won't be redirected and won't see the message.

Sub-Task

100%

Group: Basic Security Rules and Roles

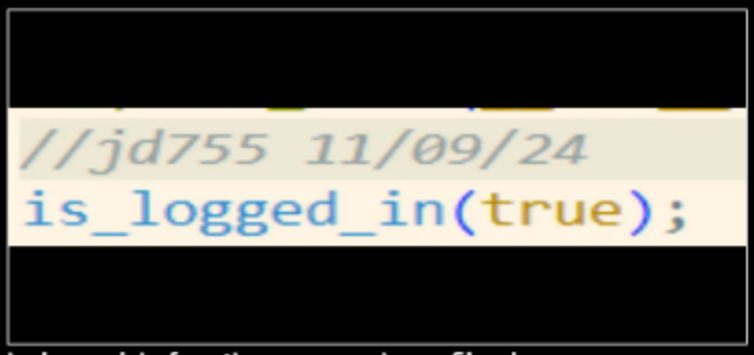
Task #1: Authentication Screenshots

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



is_logged_in function as seen in profile.php

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

This function is called at the beginning of profile.php. True is passed in meaning the user will be redirected if they are not logged in.

Sub-Task

Group: Basic Security Rules and Roles

100%

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

Trying to access a login-protected page while logged out

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

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

Function that 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 function accepts one parameter which is the name of the role. It checks if the user is logged in(without passing in true) and it checks if the user has any roles associated with them. If these are both true, it loops through each role the user has and checks if it matches the passed in role. If a match is found, return true.

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

```
//jd755 11/09/24
if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: $BASE_PATH" . "/home.php"));
}
```

has_role function as seen in assign_roles.php

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The if statement checks if the has_roles function with the role "Admin" returns false. If so, a warning is displayed to the user and they are redirected to the home page.

Sub-Task 100%

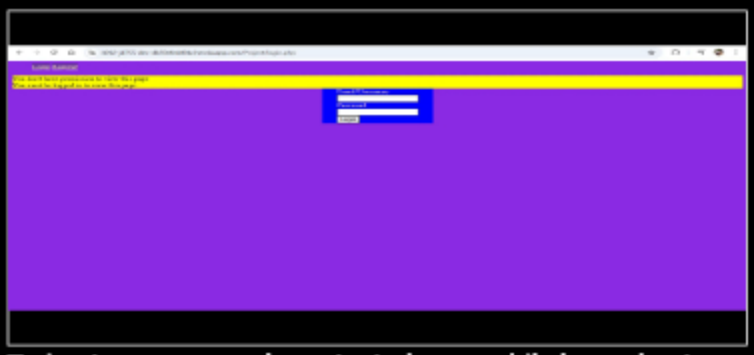
Group: Basic Security Rules and Roles
Task #2: Authorization Screenshots
Sub Task #3: Demonstrate the user-friendly message of trying to manually access a role-protected

page while being logged out (must show the appropriate message)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Trying to access a role protected page while logged out

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

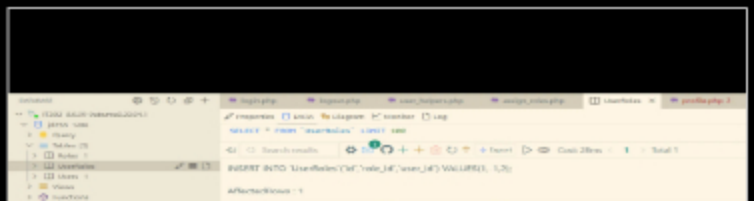
100%

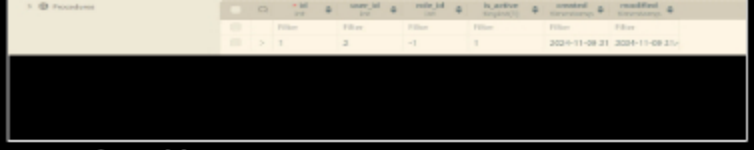
Group: Basic Security Rules and Roles
Task #3: Screenshots of UserRoles and Roles Tables
Sub Task #1: UserRoles table with at least one valid entry (Table should have id, user_id, role_id, is_active, created, and modified columns)

Task Screenshots

Gallery Style: 2 Columns

4 2 1





UserRoles table

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

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:

The UserRoles table combines a role id in the Roles table with a user id in the User table. This essentially "assigns" a

role to a user .

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.is_active determines if a role itself is turned on or off. On the otherhand, UserRoles.is_active determines if a certain user can keep using the permissions of a certain role.

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/legitdaylight/jd755-IT202-007/pull/29

URL
https://github.com/legitdaylight/jd755-IT202-007

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

The profile page

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:

1. When the page is first loaded, the email and username is taken from the session data and assigned to their respective variables.
2. THe HTML uses these variables and sets it as the default value of the email and username inputs of the form.

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

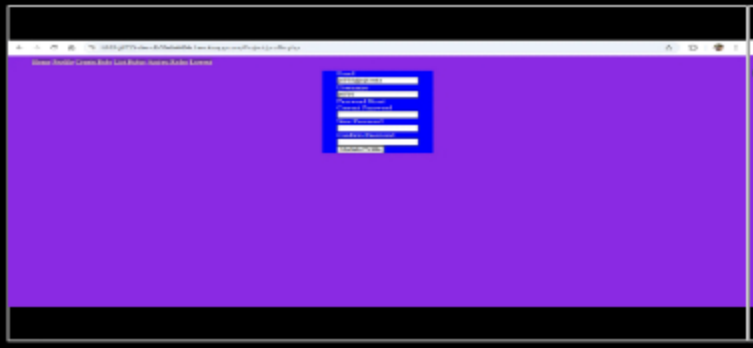
100%

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)

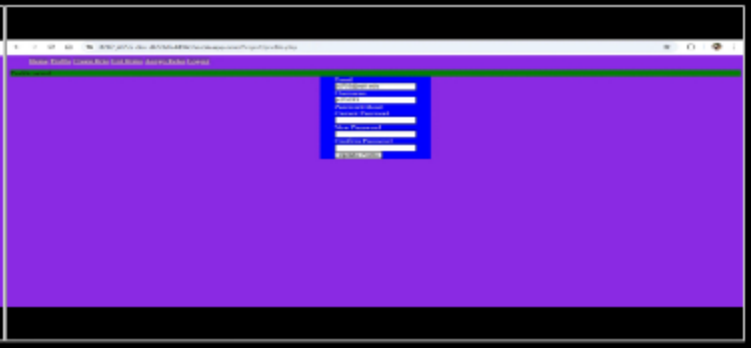
Task Screenshots

Gallery Style: 2 Columns

4 2 1



before username change



after username change

Caption(s) (required) ✓

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

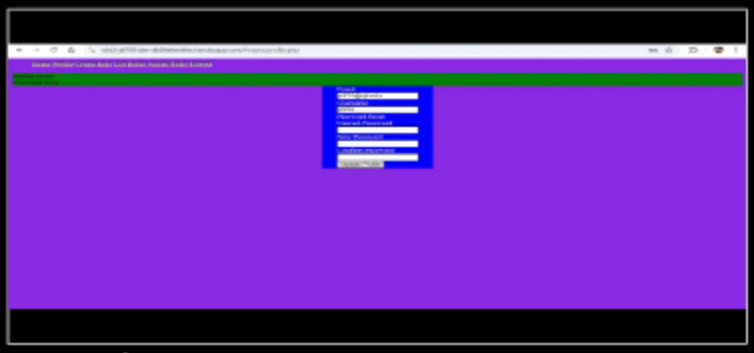
Sub-Task Group: User Profile

Group: User Profile
100%
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



password reset

Caption(s) (required) ✓

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

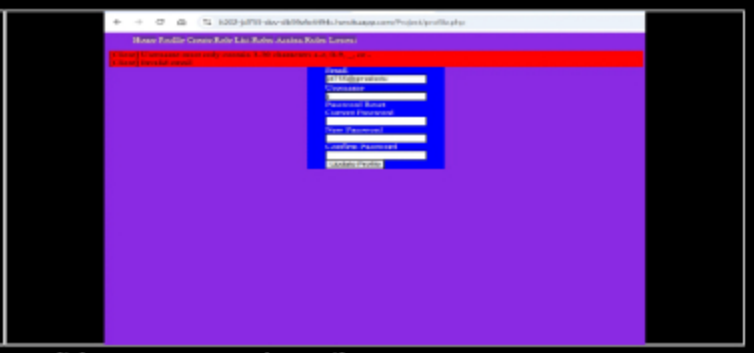
Sub-Task
100%
Group: User Profile
Task #3: Edit Profile Website Page
Sub Task #3: Demonstrate all JavaScript user-friendly validation messages [can be combined] (email format, username format, password format, new password matching confirm password)

Task Screenshots

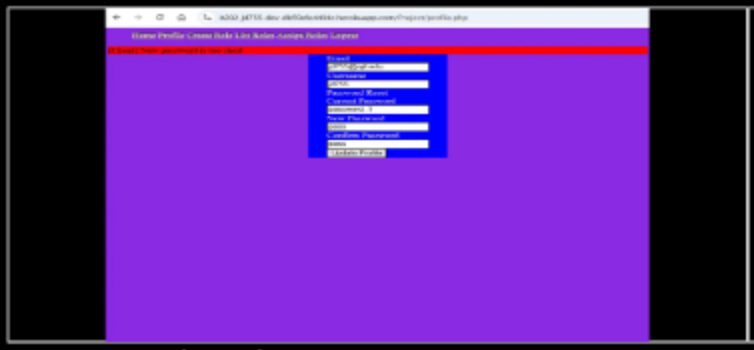
Gallery Style: 2 Columns

4 2 1

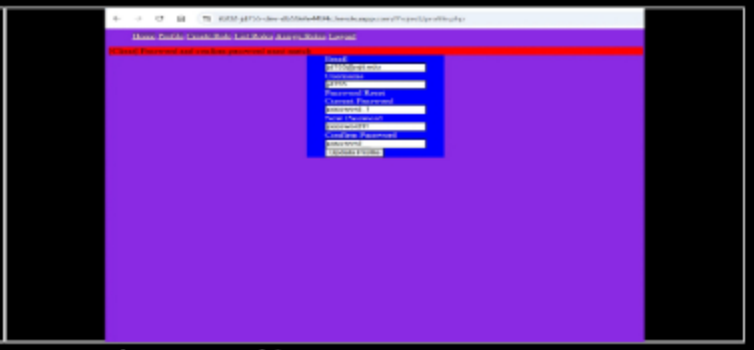
empty email and username



Invalid username and email



new password too short



passwords not matching

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 available

Caption(s) (required) ✓

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

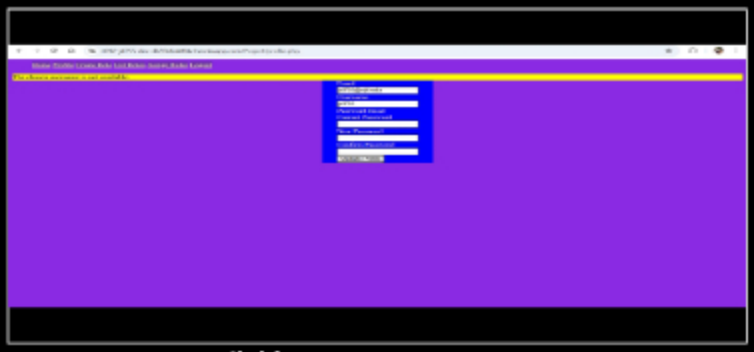
Sub-Task 100%

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 available

Caption(s) (required) ✓

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

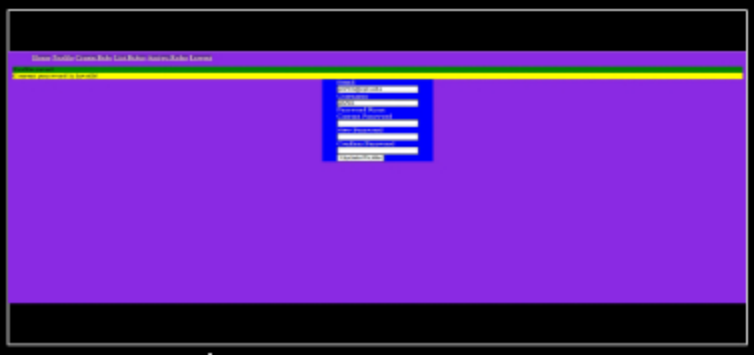
Sub-Task 100%

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



wrong password

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:

Username and Email change After the email and username is valid, this data is sent to the database via a SQL statement. If the query returns an error, the function users_check_duplicate is called to send a message to the user that there was an issue with the process.

Sub-Task

Group: User Profile

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

100%
scenarios
Sub Task #2: Updating password

Task Response Prompt

Explain in concise steps how this logically works

Response:

It is determined if the user wants to change the password if the current, new, and confirm password fields are all filled out. If they are and if they pass validation, an SQL statement is created and sent to the database. This statement takes the ID of the user and gets the password. If the passwords do not match, a warning is sent to the user that the password is invalid. If they do match, the new password is hashed then the database is updated with the new password.

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/legitdaylight/jd755-IT202-007/pull/25

URL
https://github.com/legitdaylight/jd755-IT202-007

End of Task 5

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

Group

100%

Group: Misc
Tasks: 3
Points: 1

COLLAPSE

Task

Group: Misc

100%

Task #1: Screenshot of wakatime

Weight: ~33%

Points: ~0.33

COLLAPSE

Details:

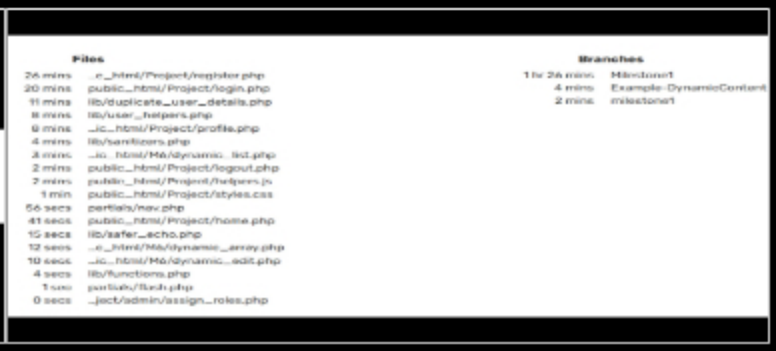
Note: The duration of time isn't directly related to the grade, the goal is to just make sure time is being tracked

Task Screenshots

Gallery Style: 2 Columns

4 2 1

waka time screenshot 1



wakatime screenshot 2

End of Task 1

Task

100%

Group: Misc

Task #2: Screenshot of your project board from GitHub (tasks should be in the proper column)

Weight: ~33%

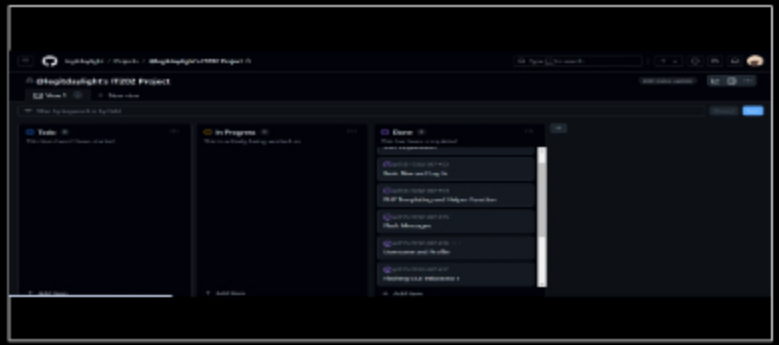
Points: ~0.33

COLLAPSE

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Github 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/legitdaylight/projects/2/views/1

URL
https://github.com/users/legitdaylight/projects/2

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment