Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-007-F2024/it202-module-6-milestone-1-2024/grade/ob75

Course: IT202-007-F2024
Assigment: [IT202] Module 6 Milestone 1 2024
Student: Omar B. (ob75)

Submissions:

Submission Selection

1 Submission [submitted] 11/14/2024 8:23:43 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

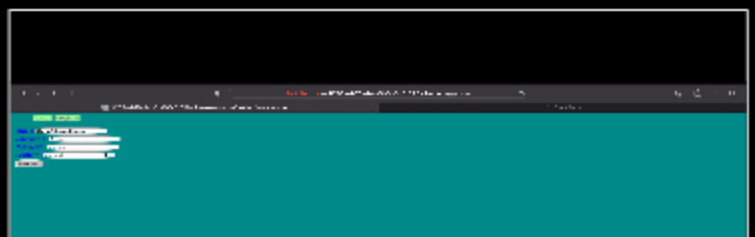
100%

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



Showing screenshot of User registration form page with data.

Caption(s) (required) ✓

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

Task URLs

URL #1 https://it202-ob75-prod-62037cdcca6c.herokuapp.com/project/register.php

https://it202-ob75-prod-62037cdcca6c.herokuap

Sub-Task

100%

Group: User Registration

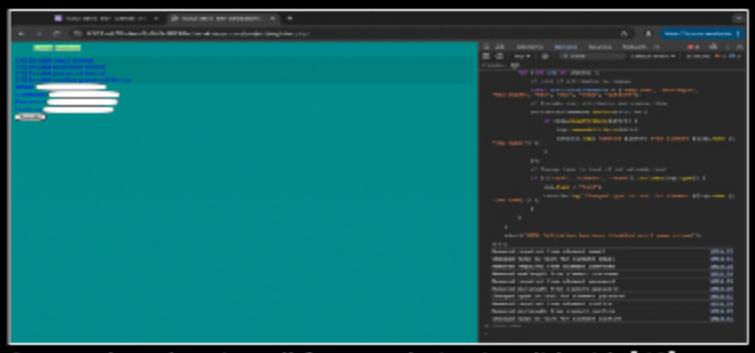
Task #1: Screenshot of form on website page

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



Screenshot showing all formats being invalid with [JS] showing the Javascript validation.

Caption(s) (required) ✓

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

Sub-Task

100%

Group: User Registration

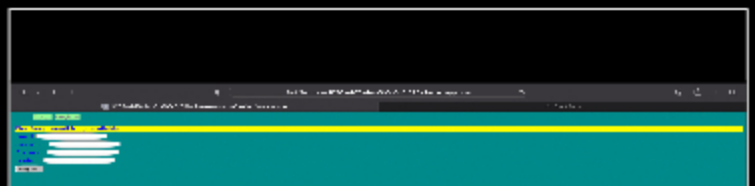
Task #1: Screenshot of form on website page

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



Showing email is no longer available for use for another account.

Caption(s) (required) ✓

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

Sub-Task

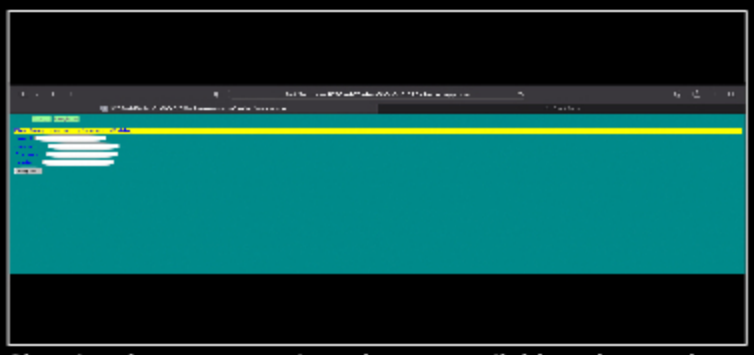
100%

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



Showing the username is no longer available to be used to create an account with same username.

Caption(s) (required) ✓

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

Sub-Task

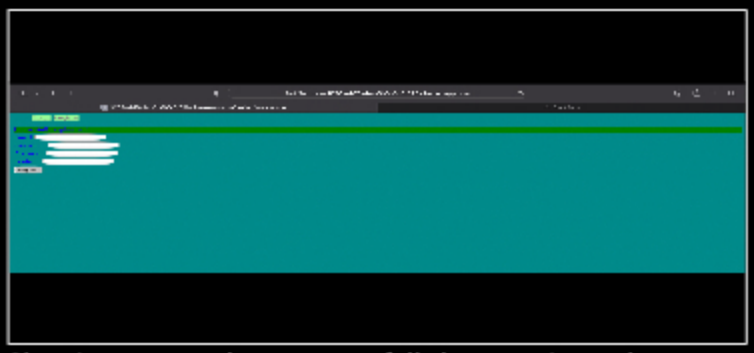
100%

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



Showing account has successfully been registered.

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

HTML form code for registration for the User.

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:

There are HTML labels created in the form that will send all the necessary information to the database with POST and it would be when the user fills out the data and submits the form. There is a lable created for Email, Username, Password, and Confirm Password that takes it the appropriate data in to fill out the information for each user registering on the site. We make each of these required so that the user has to fill out these forms to create an account and some of the requirements have criteria that must be reached or cannot exceed such as for username,

the username has a maxlength of 30 characters. Similarly, password and confirm password must have at least 8 characters to be entered to be considered to submit. Lastly, the submit button is named register for when all the data is filled out, the user can send that data over and be registered into the site with the new account.

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

Screenshot showing Javascript validations for register.php.
(UCID included, relevant code from line 31 - 65)

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

Task Response Prompt

Explain in concise steps how this logically works
Response:

First in validating in javascript for User Registration, there are a couple of variables created and first was isValid set to true so that we can use this when we want to check any of the registration fields and if they pass the checks and if

to true so that we can use this when we want to check any of the registration fields and if they pass the checks and if they did not it would be set to false and not let the form get submitted. The other varaibles such as emailRegex, userNameRegex, and passwdRegex are created to hold in the regex values respected for each varaible to check against. Then the if statements are created to use the regex to check for each of the parameters of the form such as form.email.value, form.username.value, etc. and if the entries that are placed doesn't match any of the regex then the statement shows a javascript validation message with '[JS]' to show that JS validation occured. OR conditions are also placed in the if statements in case the user did not entry any content into the field and will cause the same '[JS]' to pop up and if either or the conditions aren't met, then we set the isValid boolean to false to not pass data along and let the user know. This is done for all of the form values and then lastly, we use another if statement to check if the password value matches the confirm password value and if it doesn't then we flash out the same message again with the [JS] message to show that the passwords did not match in the javascript validation(password regex shows password must also be 8 characters long at minimum to pass validation as well). We at the end return isValid if all the condition for each respective parameter passes and then the form can be sent through.

Sub-Task

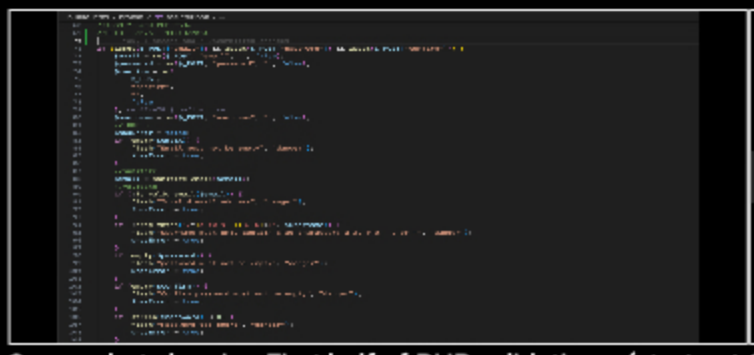
100%

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



Screenshot showing First half of PHP validations. (starts at line 82)

Screenshot showing second half of PHP validations for what couldn't fit in the first screenshot.

Screenshot of lib content for function being used in register.php.

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Data is set and sent over to the server side from the client side here at first when the user enters valid data and registers on the page. First, the hasError is set to false since this will be use in the same way as isValid to check for error when user types in data in the fields. There is a check then after to see if email has empty content and if it does, a flash message is shown to the user that it cannot be empty and sets $hasError to true showing that there is an error in the field. Then the rest of the validation occurs just like the javascript validation previously mentioned, using functions such as is_valid_email that is taken from the lib/sanitizers.php side that returns the email validated. $email is passed into the first function in the if statement checking when its not a valid email, it will flash a message giving the user an friendly error and setting the $hasError to true. The next if statement checks for when username doesnt match any regular expression and will give the user a error message and set $hasError to true. if Empty is done again against password and confirm password in their respective if statements to check if their is content and if not an error will pop up and set $hasError again to true. There is then the check against password for when it does not meet the 8 character requirement and will display the error again and change $hasError. Lastly, the final check is to match password against confirm password and to make sure that password has content as well inside of it and if both conditions are not met, the another error message will be given and set $hasError to true.

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

Showing SQL data being stored and password is hashed in the User table.

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. Firstly for registering a user, we created a form for the page that shows the fields to be for email, username, password, and confirm password.
2. The check for whether or not the data that the user inserts into these fields is correct is done to ensure that the correct data is inserted into the database without having no content in their username, email or password or even using characters that are not available to use to make a profile.
3. If there are no errors that come from the if statements checks, then the passwords the user inserted is hashed so no one else can see it and the rest of the data is stored into the database creating a new profile for that user in the database.
4. Once that data is inserted and stored into the database, the database associates the entries into their respective tables and will then flash a message back to the client that they have successfully registered and now have a profile they may login to.
5. Extra Note(The beginning of this page or when its reloaded resets the sessions everytime.)

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/OmarBarrera1/ob75-IT202-007/pull/13

https://github.com/OmarBarrera1/ob75-IT202-007

End of Task 6

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

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

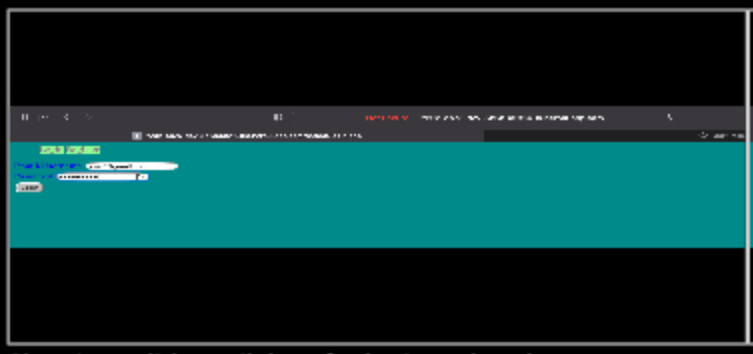
100%

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)

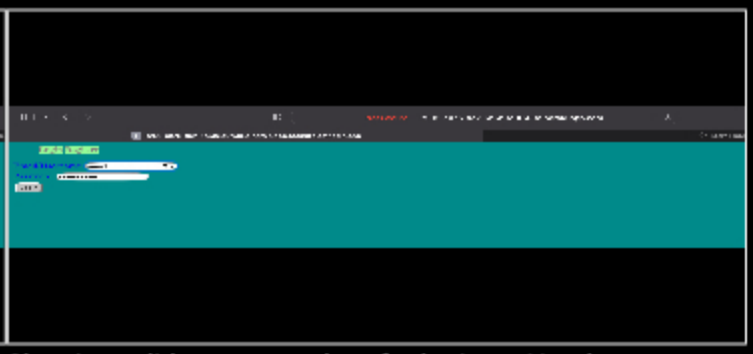
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Showing valid email data for login on heroku.



Showing valid username data for login on Heroku.

Caption(s) (required) ✓

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

Task URLs

Sub-Task

100%

Group: User Login

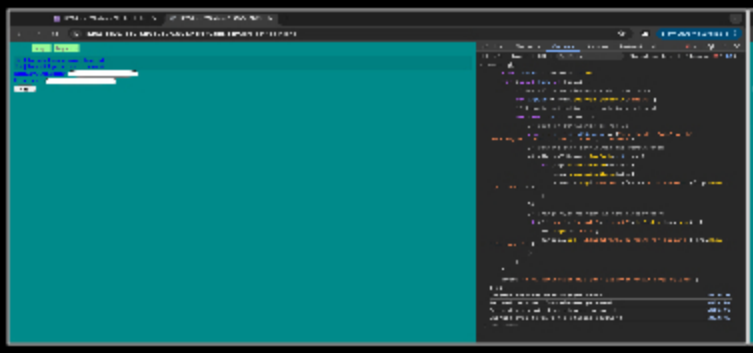
Task #1: Screenshot of form on website page

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

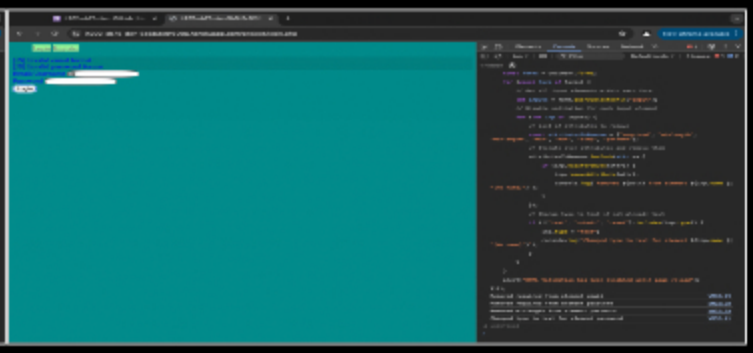
Task Screenshots

Gallery Style: 2 Columns

4 2 1



First screenshot showing invalid username and password format with [JS] as javascript validation.



Second screenshot showing invalid email and password format with [JS] as javascript validation.

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

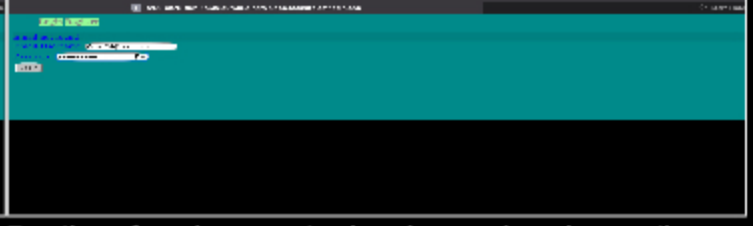
Sub Task #3: Demonstrate user-friendly message of when an account doesn't exist

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Invalid username prompt is shown when there is no account with that user name stored.



Email not found prompt is also shown when the email doesn't exist either.

Caption(s) (required) ✓

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

Sub-Task

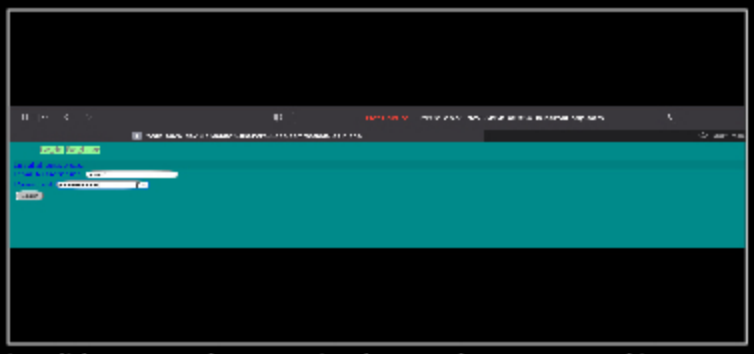
100%

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 prompt is shown when password is not matched with what is in the DB.

Caption(s) (required) ✓

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

Sub-Task

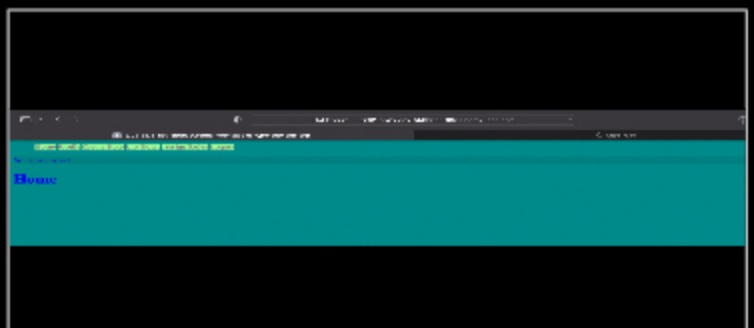
100%

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



Showing home page of user with welcome user, demonstrating successful login

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

Application logs showing the session data being set.

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

HTML code of form for login from line 5 - 16

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:

A label and input field is created for email in the form that will let the user use either email or their username to login. The 'text' attribute signifies that the user will be inserting text into this field. A password label is then created to take in the password that the user enters into the field and its attribute is 'password' to signify that it will be a password for that form. The password also has a criteria to be met which is that it must be a minimum length of 8. Then the submit is at the bottom of the form that will be named login for when the user finally inputs their information and wants to login and send that form over.

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

Screenshot showing javascript validation code for login.php. (UCID included, relevant code starting from 23 - 46)

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Firstly regexs are set to const varaibles for each check that needs to be done such as email, username, and password. isValid boolean is also set here to true to use in the if statements to change if the conditions care met. In the first if condition, we check for the value of email to see if the input value inserted by the user contains the '@', which all emails have, will be email or if they will be regular text to be interpreted that they will use their username to login. If it is an email, we run the same email validation if statement that test the email against the email regex and if it doesn't have those regex characters or if the content is empty, we will flash the message that the email format is invalid with [JS] message to show JS validations and will set isValid to false. If the user logs in with username instead we will test the input value against the username regex instead and if that doesn't match or has empty content, we will again flash the message that this time the username format is invalid with [JS] message to show JS validations and will set isValid to false. Then the last test is to check the password field and use and if statement against that to check if the password is at least a length of 8 and has content and if it doesn't do either of those then it will flash the message that password format is invalid with [JS] message to show JS validations and will set isValid to false. isValid is returned at the end if all conditions are met and if it should move forward with submission or not.

Sub-Task

100%

Group: User Login

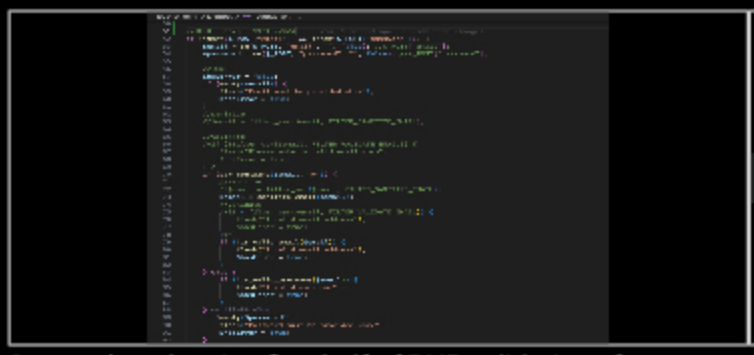
Task #2: Screenshot of the form code

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

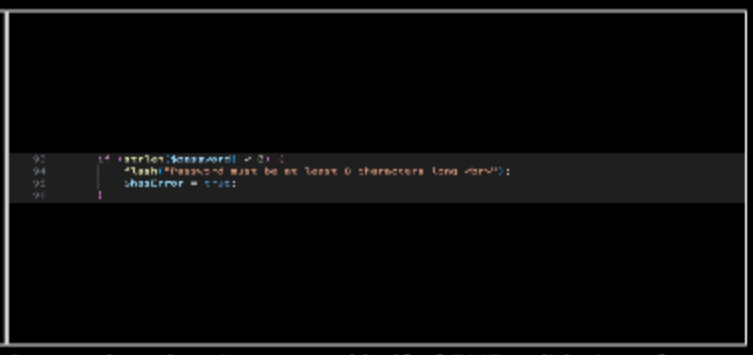
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Screenshot showing first half of PHP validations for user login



Screenshot showing second half of PHP validations for login.php

Screenshot showing lib content for functions used in

login.php for user login php validations.

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

For php validation in the login.php page, we create a hasError variable just like previously that is set to false as their is no error. Then a check is done against the email to check if it has content and if it doesnt, we send a error message back to the user and set hasError to true. This check is also done for password as well to check if password does not have content and gives the appropriate message along with changing hasError to true. We then validate our email using the functions from the lib folder in sanitizers.php and check for when it is not a valid email to send the message and change the hasError variable to true. The same check if statement is done for username to check for when username is not valid and we still pass in the email variable inside as they use the same data if it does not contain the '@'. Then the check for if the length of the password is less than 8 is done to see whenever the password's length is less than 8 to send a message to the user that it needs to be minimum of 8 characters and set hasError to true.

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:

1. Firstly, we create our form for the user to enter their information to be able to login to their profile.
2. We then validate and check for our user entering in the correct information into the fields and we let the user know if there is incorrect information into the fields.
3. If the checks are passed and we gather the correct information, then we go into our database and try grab the user's information and if it is matched with what they entered in.
4. If the information matches with the information in the database then we hash their password and we start to grab any extra information about them such as their roles and see what roles they have.
5. We then show the user that login was successful and welcome them and then redirect the user to their home page after gathering all their information.
6. (If the user inputs the correct criteria into the login fields but does not match credentials in the database, we let

the user know whether their email is not found or their password is invalid.)

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/OmarBarrera1/ob75-IT202-007
https://github.com/OmarBarrera1/ob75-IT202-007/pull/14

URL #2 https://github.com/OmarBarrera1/ob75-IT202-007
https://github.com/OmarBarrera1/ob75-IT202-007/pull/23

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 Group: User Logout

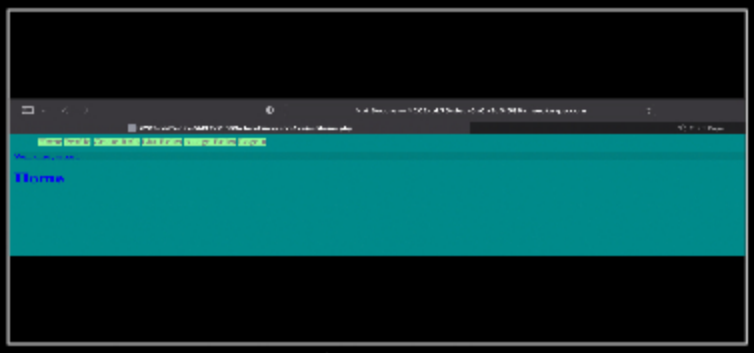
100%

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



Showing the navigation of site when logged in.

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

Sub-Task

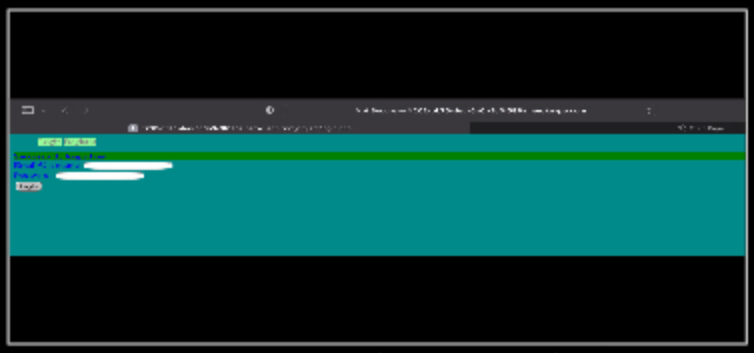
100%

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



User logged out with successful message of being logged out and returns to login page.

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

Sub-Task

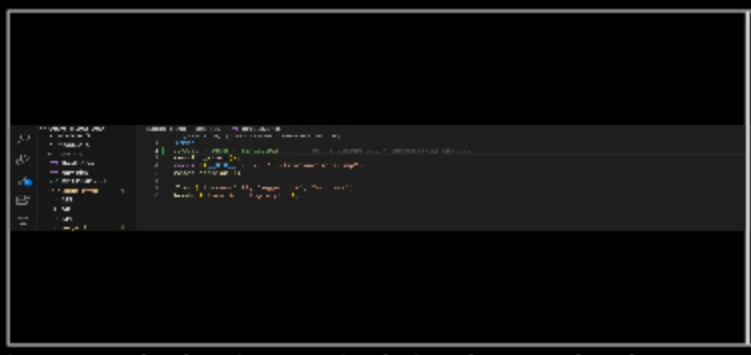
100%

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 code showing session being destroyed and reset.

Showing reset_session function that is done in logout that destroys session.

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

In logout.php we see that our session is started and when we logout, our session is reset with the function reset_session() from reset_session.php that will prevent the session being accessed outside of just logging in. The reset_session function in reset_session.php unsets the session from being active and destroys it so that it stops the session to not allow anyone else in if not logged in and starts the session over again. A flash message will show that it was successful in logging the user out and the header points the user back to the login page. The user will need to login again in order to access their page again.

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/OmarBarrera1/ob75-IT202-007/pull/14

https://github.com/OmarBarrera1/ob75-IT202-007

URL #2
https://github.com/OmarBarrera1/ob75-IT202-007/pull/16

https://github.com/OmarBarrera1/ob75-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

Screenshot of the function is_logged_in being used in user_helpers.php to check for if user is logged in. (at line 0-18)

9-18).

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The is_logged_in function here takes in $redirect to be false since if set to true, it will redirect a user that is logged out to the login page again, and takes in destination which is set to login.php. Our isLoggedIn variable is set to a boolean of whether the session of the user is set. If our redirect and isLoggedIn variables are false, then we go ahead and tell the user that they must be logged in to the view the page they are trying to access. We then bring them back to the login page so that they may enter the necessary data to login. We then return isLoggedIn to show whether or not the user is at the end logged in or not.

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

Login check function being used in profile.php

Screenshot of login check function being used in home.php. (on the site, its the home page for the user)

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Our is_logged_in function is used here in profile.php and it takes in the true boolean value because in order for the user to access the profile page on the site they must be logged in so we call that function in the beginning of profile.php before moving forward with anything else. The is_logged_in function is also used in the home.php page as well because the home page is where the user is first redirected to when the user logs in correctly so the function is placed at the top at the start for the same reason as profile.php as you need to be logged in to access the page.

Sub-Task

Group: Basic Security Rules and Roles

Task #1: Authentication Screenshots

100%

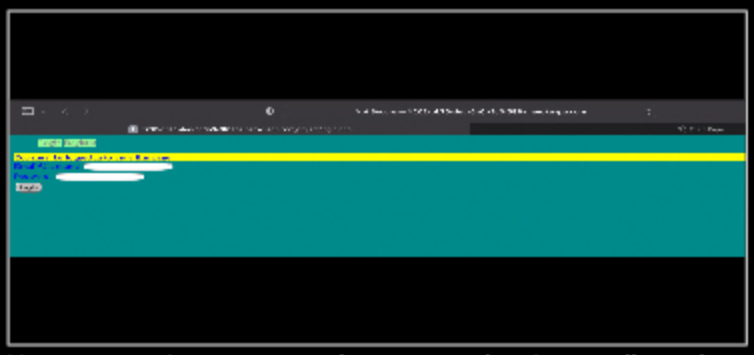
Task #1: Authentication Screenshots

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



User attempting to access home page but is not allowed and met with a message that shows they must be 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

Screenshot of the function that checks for role in a user in user_helpers.php.

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Our functon has_role takes in the role variable to check for whatever role may be checked in this parameter such as admin or any other role. We then create an if statement inside the function to first check if the user is logged in and then also if there is session data set for the user with the role. Then we loop through the roles that the user has and if the name of that role mathes with the role passed in to the function, then we return true showing that the user has that role. if not we return false showing that they do not have that role.

Sub-Task

100%

Group: Basic Security Rules and Roles

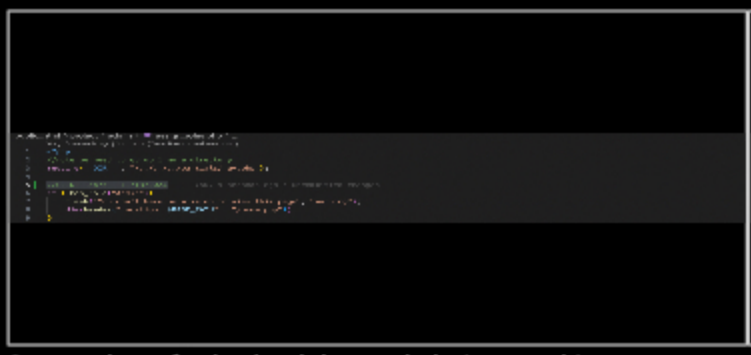
Task #2: Authorization Screenshots

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

Task Screenshots

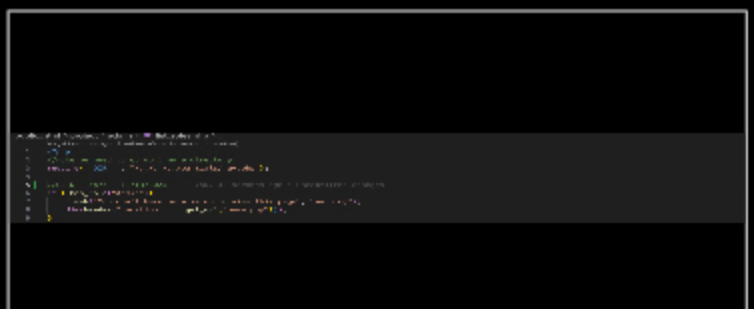
Gallery Style: 2 Columns

4 2 1



Screenshot of role check has_role being used in assign_role.php. (Would be on page assign_role on the site)

Screenshot of role check has_role being used in create_role.php. (Would be on page create_role on the site)



Screenshot of role check has_role being used in list_roles.php. (Would be on page list_roles on the site)

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 functions is used in all 3 of the pages, assign_role.php, create_role.php, and list_roles.php all the same way to check for if the user has the admin role. We create an if statement that checks if the user does not have the admin role so that we can provide a flash message to the user saying they dont have the correct permissions to access the page and we send them back to the home page. These 3 pages are privileges for only the admin to use and do and not allowed for regular users that don't have the admin role.

Sub-Task

100%

Group: Basic Security Rules and Roles

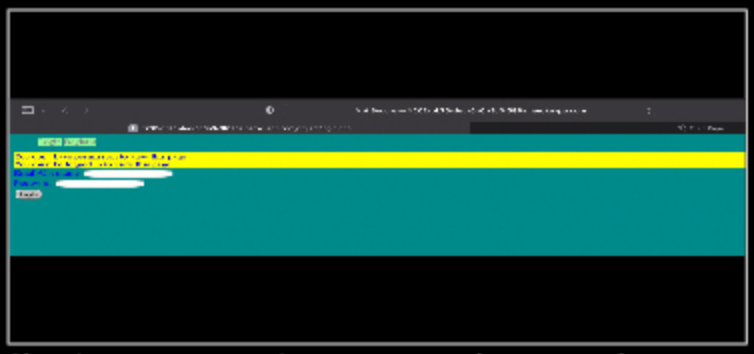
Task #2: Authorization Screenshots

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



Showing user attempting to access role-protected page they don't have access to being logged out and met with 2 messages.

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

Demonstration of User Roles table with valid data.

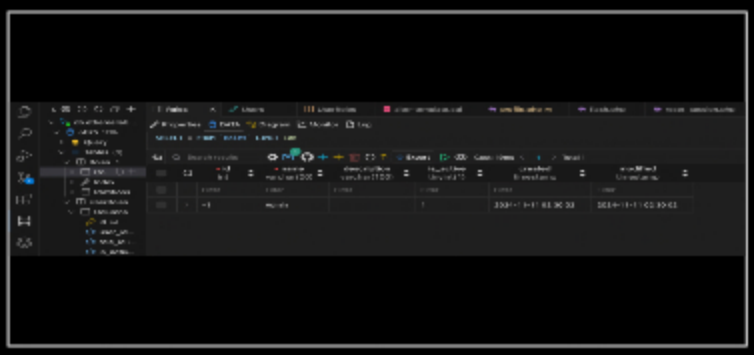
Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

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 demonstrated with at least one 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:

The purpose of UserRoles table is to assign different roles to every user whether it be an Admin role by setting their id to -1 which is the id role for Admin or setting it to a different role that may be created for example allowing a user special priveleges to access some other features that other user cannot access but still does not allow them to change the roles of others like Admin does thus giving them a different role. This can also be just normal roles given to users to just login and access their own information.

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 differs from UserRoles.is_active because Roles.is_active is set to 1 at all times for the admin permissions and gives that active role to someone who gets the same id as the admin field giving them the active role. For UserRoles.is_active, this is deciding whether or not the user has an active role to any role or not role at all in the table. If the UserRoles.is_active is set to 1 then the user has their role active and if it is set to 0 instead or null, then it means their role is set to false or set to not active therefore deactivating the role given to the user.

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/OmarBarrera1/ob75-IT202-007/pull/21

URL
https://github.com/OmarBarrera1/ob75-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

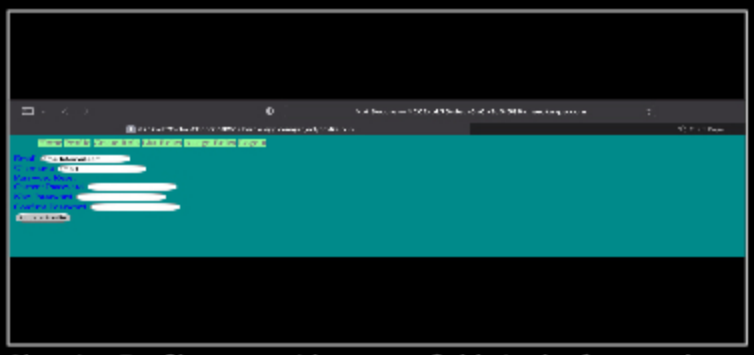
100%

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



Showing Profile page with correct fields in the form and allowing for profile to be updated.

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. We use require once for the nav page to relocate us to the profile page where our profile will be loaded.
2. We first pass in the boolean true value into the is_logged_in function because our user needs to be logged in to access the profile page so we make that function pass our user through once they are logged in.
3. Our forms are created to show for email, username, current password, new password and confirm password in which can all be changed to different values.
4. We then show the page with the user's email data filled in along with the username and the rest of the fields empty.(email and username is fetched from the database and filled in according to the user.)

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.

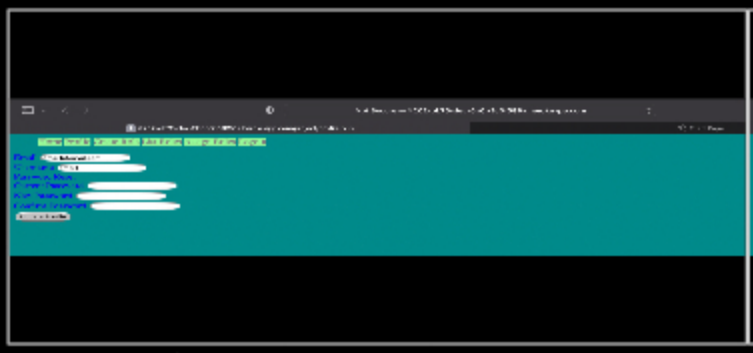
Columns: 1

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)

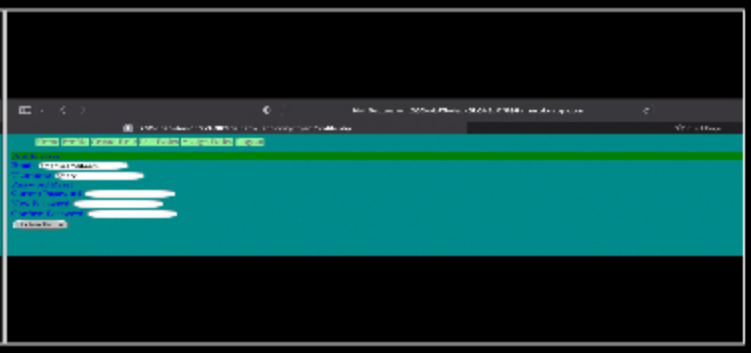
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Username before starts with omar1.



omar2 is the new username change and shows success message at the top with "Profile saved" being shown once done.

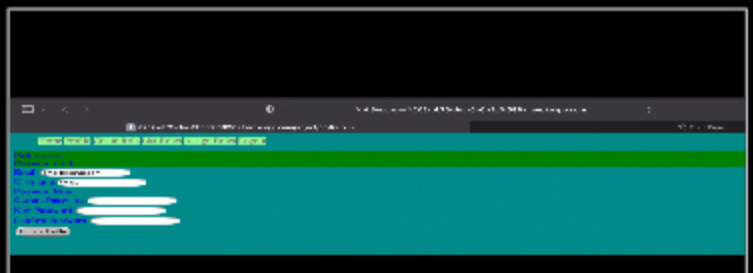
Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

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



Showing profile was saved with password changed and message of "Password reset" showing new a new password was made.

Caption(s) (required) ✓

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

Sub-Task

100%

Group: User Profile

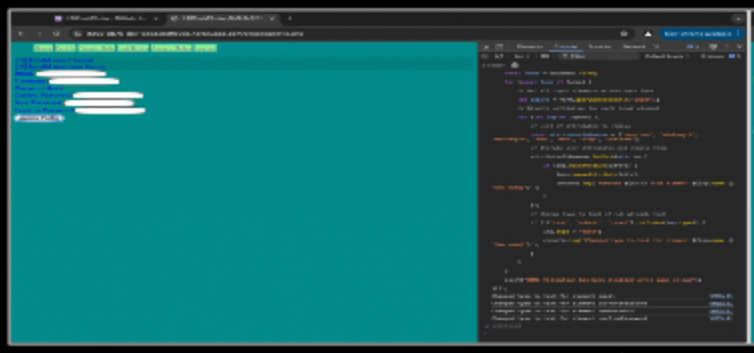
Task #3: Edit Profile Website Page

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

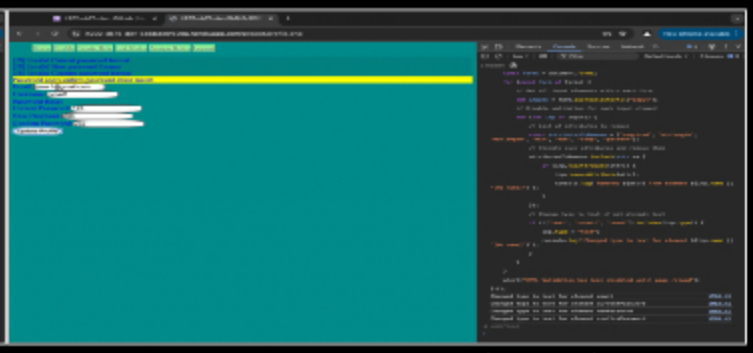
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Screenshot showing Invalid username and email format with [JS] message for javascript validation



Screenshot showing Invalid password formats for current, new, and confirm password with [JS] message for javascript 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

Screenshot showing email is not available to use in the

message and doesn't allow for user to use the same email.

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

Screenshot showing user is not able to user same username as another user as It is taken. Shows message that username is taken.

Caption(s) (required) ✓

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

Sub-Task

100%

Group: User Profile

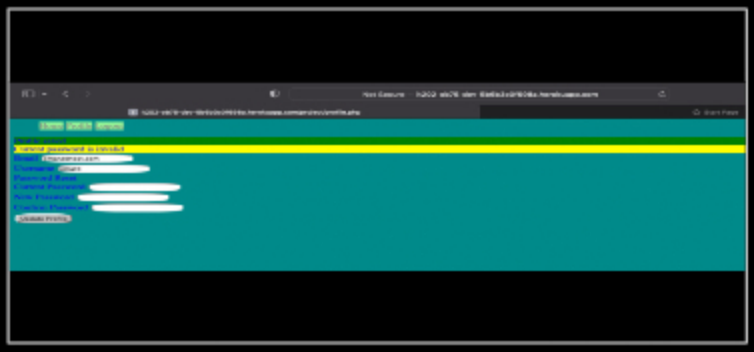
Task #3: Edit Profile Website Page

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



Screenshot showing that current password is invalid in message and will not allow to change password until matched.

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:

Updating email/username is first checked with our validations to ensure that the user is inserting the correct information required into the username or email fields to be able to change them. If not met, it will give the user an error and not allow them to change their information. When the user insert the correct information required and changes their username/email, we will check if that username or email already exist in the database and if it does, it will show the user a message saying what they chose is already taken. If what they chose is not taken by anyone, we will go into the database and update our tables with the user's email or username or both to be according to what the user inputted. The user then gets a 'profile saved' message on their page to properly show their data was updated.

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:

In updating password, We go through our validation check to make sure that what the user is entering for the current password and their new and confirm password is correct to be able to move forward. Once the user enters in the correct necessary password information, we will then check if the current password mathces with the password stored in the database and if not an error message is shown to the user. The new password is also compared to the confirm password and return another error message if not mathced. Once all criteria is met and matched with each

confirm password and return another error message if not matched. Once all criteria is met and matched with each other, then we grab their current password from the database according to their id and we will verify the current password to make sure its correct. Then we hash the new password just like when the user first created the password. Then the new password is stored in place of the previous password in the database and we send a message to the user that their password was now reset.

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/OmarBarrera1/ob75-IT202-007/pull/17

https://github.com/OmarBarrera1/ob75-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

100%

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

COLLAPSE

Details:

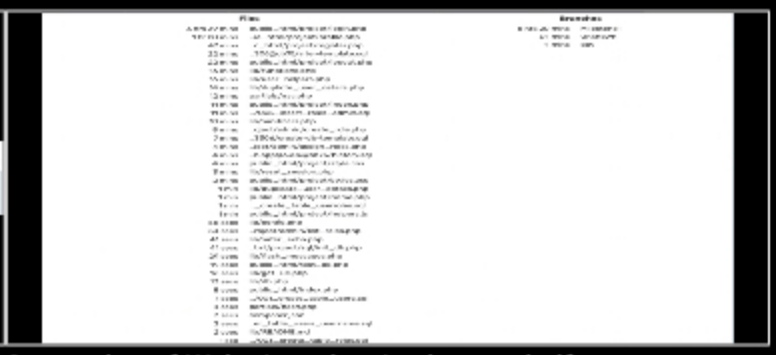
Note: The duration of time isn't directly related to the grade, the goal is to just make sure time is being tracked

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Screenshot of Wakatime top half.



Screenshot of Wakatime showing bottom half.

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%

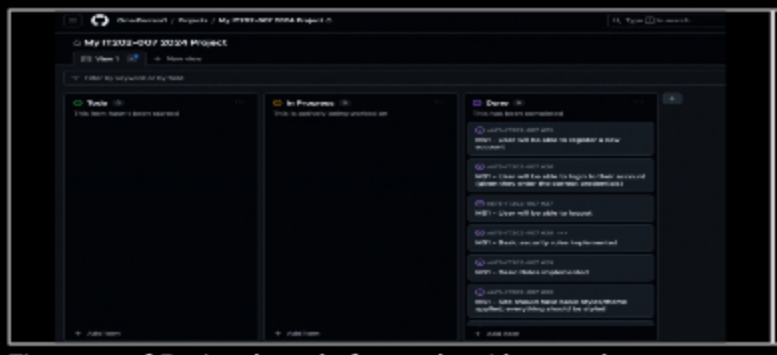
Points: ~0.33

COLLAPSE

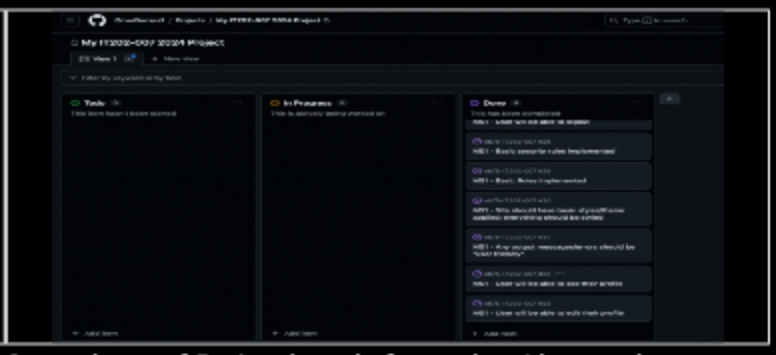
Task Screenshots

Gallery Style: 2 Columns

4 2 1



First part of Project board of complete/done task.



Second part of Project board of complete/done task.

End of Task 2

Task

Group: Misc

100%

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/OmarBarrera1/projects/

https://github.com/users/OmarBarrera1/projects/1/views/1?layout=board

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment