Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-007-F2024/it202-module-6-milestone-1-2024/grade/db624

Course: IT202-007-F2024
Assigment: [IT202] Module 6 Milestone 1 2024
Student: Datha V. (db624)

Submissions:

Submission Selection

1 Submission [submitted] 11/11/2024 11:44:02 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

Screenshot of filled in registration form

Caption(s) (required) ✓

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

Task URLs

URL #1

https://it202-db624-prod-a236ea831ca5.herokuapp.com/Project/register.php

URL

https://it202-db624-prod-a236ea831ca5.heroku

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

Error messages when required fields aren't filled out

Errors when format for input fields aren't followed

Caption(s) (required) ✓

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

Sub-Task

100%

Group: User Registration

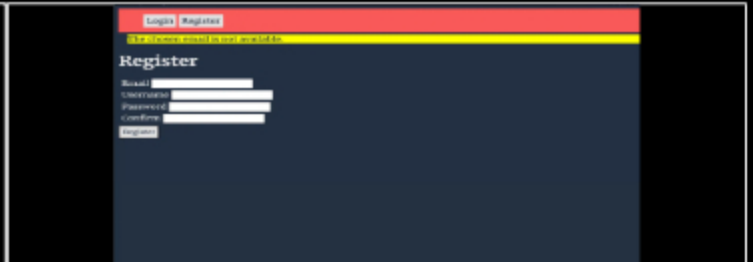
Task #1: Screenshot of form on website page

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



testing re-used email

Error message when using an email that already exists

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

testing username that already exists

Error when trying to use username that already exists

Caption(s) (required) ✓

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

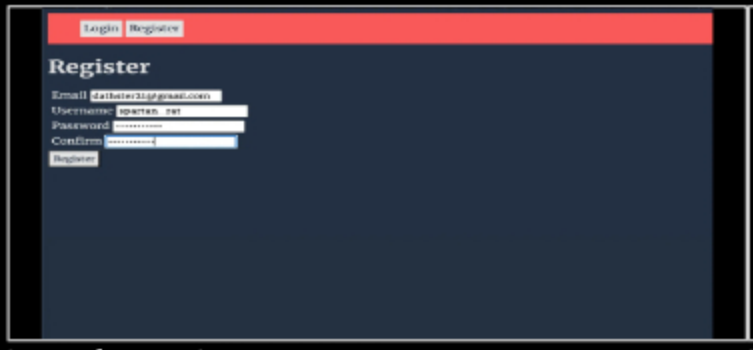
Sub-Task 100%

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



Inputs for creating a new user

Message showing successful user creation

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

```
<!-- db624 it202-007 11/11/24 -->
<h1>Register</h1>
<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 minlength="3" maxlength="16" />
    </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 code for 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:

for the email field, the email type ensures that inputs follow the format of an email. The name field specifies the name the field data will be associated with when sent to the server. The required attribute means that a value must be provided for the form to submit

For the username field, the type is text so it accepts regular text. The name field specifies the name associated with field data when the form is submitted to the server. The required attribute means that a value must be provided for the field to submit. The minlength and maxlength attributes specify the minimum and maximum lengths of any valid username.

For the password and confirm password fields, because their type attribute is password, it means whatever text the user types will be hidden. The required attribute means that the user must provide an input for both fields for the form to be submitted. The name attributes specify the name associated with the inputted data when the form is submitted. The minlength attribute specifies that any inputted passwords must be at least 8 characters long.

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

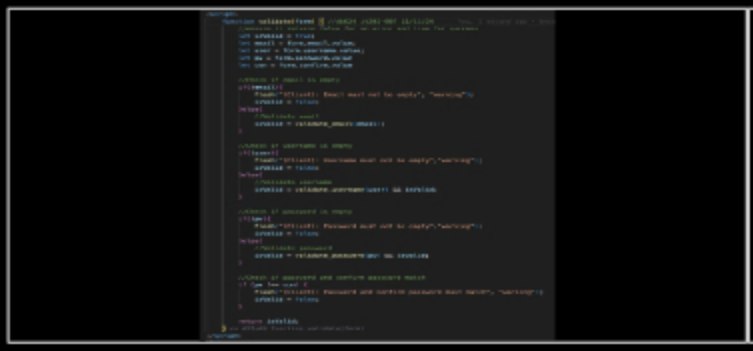
100%

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 of javascript validation in register.php

Screenshot of helper validation functions from 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:

First I store all the data from the form in variables.

Next, I check if the email is empty and print an error message if that's the case. Otherwise, I check if the given email is valid (i.e. it follows user@domain.com format and contains only `a-z,A-Z,0-9,_,-` for the characters

After that, I check if the username field is empty and print an error message if that's the case. Otherwise, I check if the given username is valid (i.e. it is 3-16 characters long and contains only `a-z,0-9,_,-`

Third, I checked if the password field is empty and printed an error message in that case. Otherwise, I checked if the given password was valid (i.e. it must be at least 8 characters long).

Finally, I checked if the password and confirm password fields matched and printed an error if that wasn't the case. If any of the errors above occurred, then the validate function would return false, preventing the form data from being sent to server for validation.

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 screenshot 1

Php validation screenshot 2

```php
<?php
//db624 it202-007 11/11/24
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;
}
```

helper validation code in sanitizers.php

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

First, the script checks if the email, username, password, and confirm password data has been sent through the post request.

Next, I load in the email, username, password, and confirm password data into variables.

After that, I check if the email or username fields are empty, in which case an error will be printed.

If the email isn't empty, then I sanitize it to remove any dangerous characters or scripts from the input.

Next, I check if the given emails and usernames are valid. I use PHP validation function to check if the email is valid, and I used a regular expression to check if the username is between 3 and 16 characters and only contains a-z, 0-9, _, -.

Next, I check if the password and confirm password fields are empty and raised an error if that is the case. If the password isn't empty, then I validated it. I raised an error if the password is shorter than 8 characters.

Next, I checked if the password and confirm password fields matched and raised an error if they are not matching.

If any of the checks below raised an error, then none of this data will be passed to the database.

If no errors were present, the password is hashed and salted and all the form data is stored in the database.

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

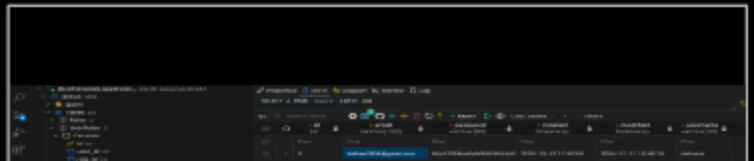
100%

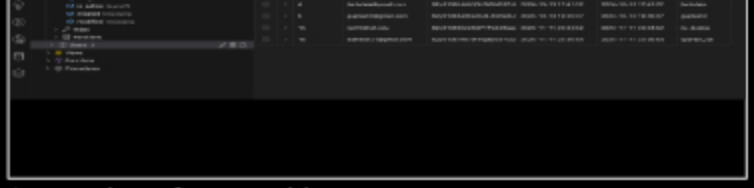
Group: User Registration
Task #4: Screenshot of the Users table with a valid user entry
Sub Task #1: Show valid data per the checklist

Task Screenshots

Gallery Style: 2 Columns

4 2 1





Screenshot of users 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:

First the user inputs all the form data. Once complete they hit submit. HTML, PHP, and Javascript validation check the given inputs and ensure they're valid. This means that no fields can be empty, email must follow user@domain.com format, username must be between 3 and 16 characters and only contain a-z,0-9,_,-, password must be at least 8 characters and password and confirm password must match.

Once HTML and Javascript validations pass, the data is validated by the PHP script and once all checks are passed, the data is prepared to be inserted into the database. First, the password is hashed and salted to ensure that is secure. Then, I access the database and execute an insert query. Through this query, I insert the given username, password, and username into the database. If the query fails, I check if the username or password are duplicates (i.e. there are other ones already in the database). In that case, an error message is printed and no data is inserted into 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/Dathster/db624-it202-007/pull/16

URL
https://github.com/Dathster/db624-it202-007/pul

URL #2
https://github.com/Dathster/db624-it202-007/pull/27

URL
https://github.com/Dathster/db624-it202-007/pul

URL #3
https://github.com/Dathster/db624-it202-007/pull/29

URL
https://github.com/Dathster/db624-it202-007/pul

End of Task 6

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

Group

100%

Group: User Login
Tasks: 4
Points: 2

COLLAPSE

Task

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

Screenshot of valid email data for login

Screenshot showing login using valid username

Screenshot showing successful login

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

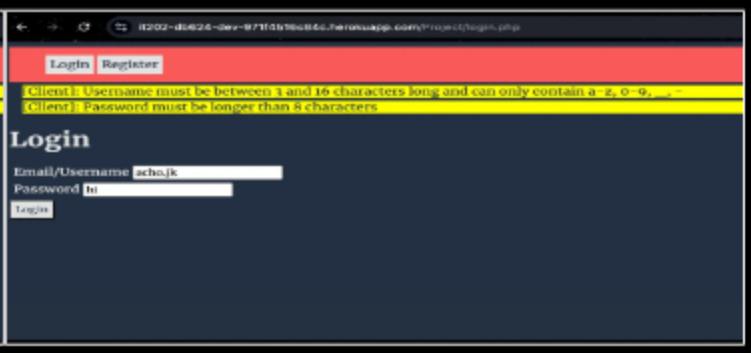
Sub Task #2: Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)

Task Screenshots

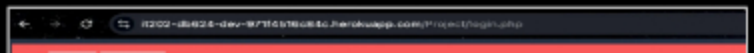
Gallery Style: 2 Columns

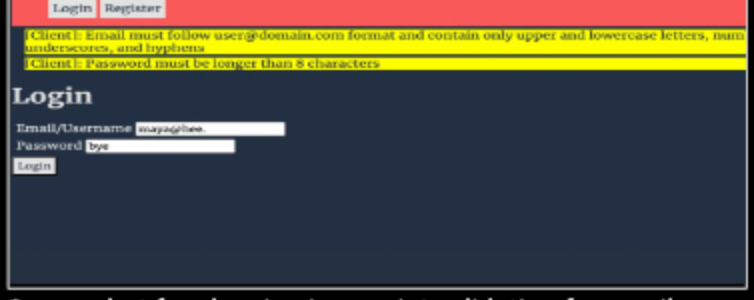
4 2 1

Javascript validation for the case where email/username and password fields are empty



Javascript validation for the case of invalid username and password





Screenshot for showing javascript validation for email

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

Screenshot of error message when an account doesn't exist

Caption(s) (required) ✓

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

Sub-Task 100%

Group: User Login

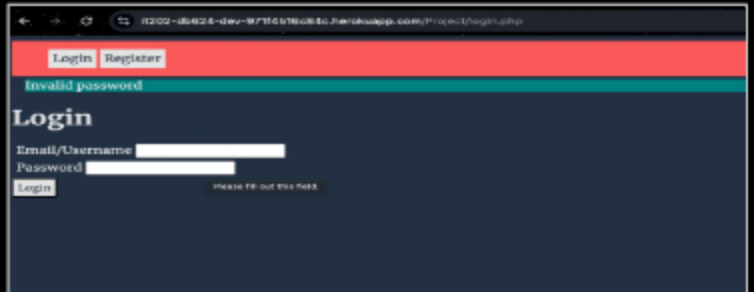
Task #1: Screenshot of form on website page

Sub Task #4: Demonstrate user-friendly message of when password doesn't match what's in the DB

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Screenshot of error message when wrong password is given

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

Screenshot of 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

Screenshot of session data 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

Gallery Style: 2 Columns

4 2 1

```
<h1>Login</h1>
<form method="POST" onsubmit="return validate(this);">
    <div>
        <label for="email">Email/Username</label>
        <input type="text" id="email" 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" name="login" id="login" />
</form>
```

Screenshot of HTML form code

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 email field has a type attribute set to text which means it is treated as regular text. It also has the name email, which is the name associated with the input data when the form is sent to server. The required attribute means that a value must be provided for the form to submit.

The password field has a type attribute set to password which means that whatever text input will be hidden for security. It also has a minlength attribute which ensures that any input is at least 8 characters long. The required attribute means that a value must be provided for the form to submit. The name attribute sets the name associated with the input when it is sent to the server.

Sub-Task

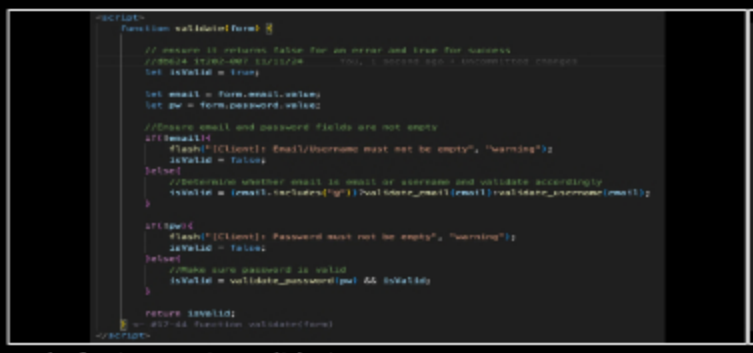
100%

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



code for javascript validation

Code for helper validation functions

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

First, I load the username/email and password into variables. Next, I check if the username/email field is empty. If it is empty, I raise an error. Otherwise, I check if the username/email has an "@". If it does, I validate it as an email. If it doesn't, I validate it as a username. The respective validation functions raise an error if the input doesn't follow the specified format.

Next, I check if the password is empty and raise an error in that case. Otherwise, I run the password validation function, which raises an error if the password is shorter than 8 characters.

If no errors are present, the validation function returns true and returns false otherwise.

Sub-Task

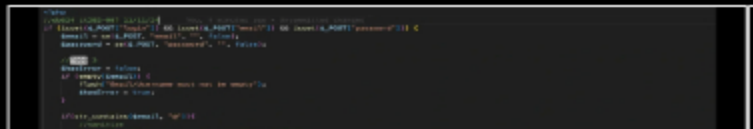
100%

Group: User Login
Task #2: Screenshot of the form code
Sub Task #3: Show PHP validations (include any lib content)

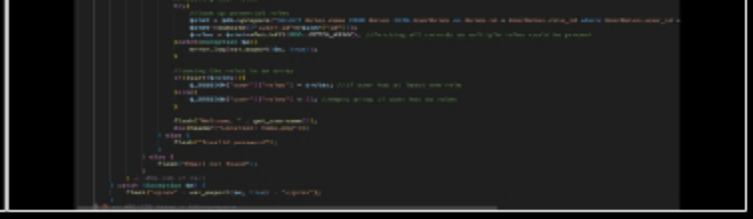
Task Screenshots

Gallery Style: 2 Columns

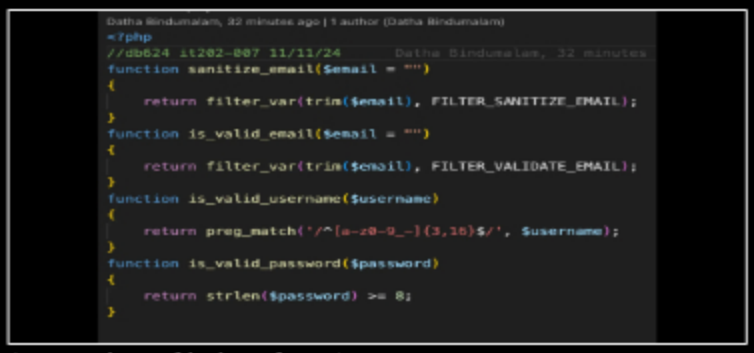
4 2 1



php validation screenshot 1



PHP validation screenshot 2



Screenshot of helper functions

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

First the script checks if the email and password variables have been sent through the post request. Next, it loads the email and password inputs into local variables. It first checks if the email is empty and raises an error if it is. Next, it checks whether the email contains an "@", in which case it validates the input as an email. Otherwise it validates the input as a username. It will print error messages if either of the validations find an error.

Next, it will check if the password is empty and raise an error if so. Otherwise, it will check if the password is shorter than 8 characters and raise an error if so.

If none of the validations above raise errors, the script moves onto checking the input data against database results. First it tries to see if a match exists in the database for the given username/email. If no match is found, an error is printed. If a match is found, then it hashes the given password and uses the password_verify function to check the input hash with the password from database. If the hashes don't match, an error is printed saying invalid password.

If no errors occur and the user had input valid login details, a new session token is created for the user and their roles are also loaded onto the session token.

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:

First, the user inputs data into the email/username and password fields

Next, HTML and Javascript validations validate the user input on client side. If no errors are present, then the data is sent to PHP script via a post request.

Within the PHP script, validation of the username/email and password are done again. If no errors are found, then the PHP script makes a call to database where it tries to find all records where the given username/email matches the username or email column in database. If no matches are found, then an error is raised saying invalid email address. However, if a match is found, then the input password is hashed and then compared to the password hash from database. If the hashes don't match, an error is raised saying invalid password.

If the username/email and password both match values in database, then a new session token is created to store the user's username/email and password and role data.

The session token is a cookie, which is a small file stored on client-side containing information that can be used by the browser to track a user's session. We are using it in this case to store the user's login data so that whenever they navigate to different pages in the website, they won't have to login again. We are using one of the PHP magic variables called $_SESSION to set the session data.

Once the session has been created, the user is redirected to home.php and a successful login message is printed.

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/Dathster/db624-it202-007/pull/17

URL

https://github.com/Dathster/db624-it202-007/pul

End of Task 4

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

Group

100%

Group: User Logout
Tasks: 2
Points: 1

COLLAPSE

Task

100%

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

COLLAPSE

Columns: 1

Sub-Task

100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Screenshot of navigation 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

Screenshot of redirect to login

Caption(s) (required) ✓

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

Sub-Task

100%

Group: User Logout

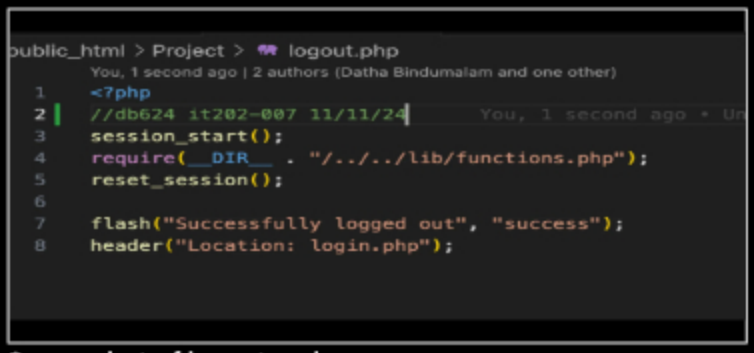
Task #1: Capture the following screenshots

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



Screenshot of logout 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 code erases current session data and resets it. Once reset, it will redirect the user to the login page and print a successful logout message.

End of Task 1

Task

100%

Group: User Logout

Task #2: Include pull request links related to this feature

100%
Weight: ~50%
Points: ~0.50

COLLAPSE

Details:
Should end in /pull/#

Task URLs

URL #1
https://github.com/Dathster/db624-it202-007/pull/17

URL
https://github.com/Dathster/db624-it202-007/pul

End of Task 2

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

Group

100%

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

COLLAPSE

Task

100%

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

COLLAPSE

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

Columns: 1

Sub-Task

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 login check 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 function has the parameters of redirect and destination. Redirect specifies whether the user will be automatically redirected or not. The destionation specifies the location user should be redirected to. They have the default values of false and login.php, so by default it won't redirect and if it were to, it would by default redirect to login.php.

The function checks if the user is logged in by seeing if their session variable is set. If it is not set, and the redirect condition is set to true, then a warning message is displayed and the user is redirected to the login page.

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

Example of login check function being used. This function is used in the following pages: profile home nav

Caption(s) (required) ✓

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

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

The call is_logged_in(true) makes a call to the is_logged_in() function which checks if the user's session variable is set. If it isn't set, this means that the user is not logged in, and since the true argument is being passed, the user will be redirected to the default page of login.php. An error message will also be flashed on the screen stating the user is not logged in.

Sub-Task

100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Given error message when trying to access profile.php without being 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 code checking for specific role

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

First the function checks if the user is logged in and their session token contains their roles. Then, it goes through the array of roles stored in the user's session token and checks if any of the roles match the one provided as a parameter. If a match is found, then true is returned. If no matches are found, then false is returned.

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 being used Used on pages: nav
Create roles List roles Assign roles

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

These pages simply use an if statement where they're checking if the user has the role "admin". If they don't have it, an error is printed stating they don't have permission to view the page and they are redirected back 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 create_roles.php while not being logged in

Screenshot of trying to access create_roles.php when logged in but not having admin role

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

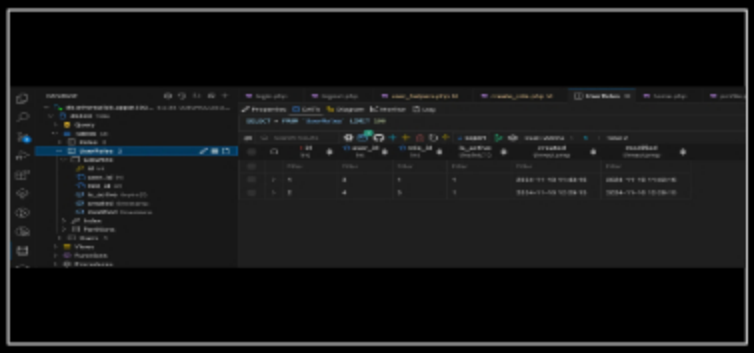
100%

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



Screenshot of user roles table

Caption(s) (required) ✓

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

Sub-Task

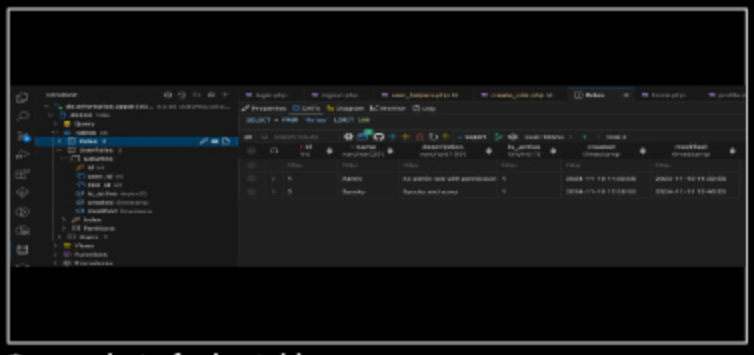
100%

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



Screenshot of roles table

Caption(s) (required) ✓

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

End of Task 3

Task

Group: Basic Security Rules and Roles

100%

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 roles table contains all the roles, their descriptions, and whether they're active or not. The Users table contains data on the user like their email, username, user_id, and password. The UserRoles table brings these two tables together, linking users with their roles. The UserRoles table has the attributes of user_id, role_id, and is_active. This associates a user with a role based on their user_id and role_id. the is_active attribute specifies whether the assigned role is set to active or not for the 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:

the Roles.is_active column is global. When it is disabled, the role will be disabled for all users. However, the UserRoles.is_active column is more specific, when disabled, it only disables the role for the specified user, with the role still functioning as normal for other users.

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/Dathster/db624-it202-007/pull/26

URL
https://github.com/Dathster/db624-it202-007/pul

URL #2
https://github.com/Dathster/db624-it202-007/pull/24

URL
https://github.com/Dathster/db624-it202-007/pul

End of Task 5

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

Group

100%

Group: User Profile
Tasks: 5
Points: 2

COLLAPSE

Task

100%

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

COLLAPSE

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

Sub-Task

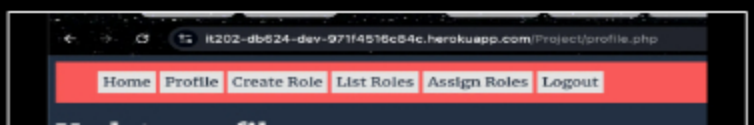
100%

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Screenshot of 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:

First, the get_user_email() and get_username() functions are called. These two functions are from the user helpers file and load the user's email and username onto local variables.

Next, the form fields are created for the email, username, current password, new password, and confirm password. Within the value fields of the email and username input fields, the default value is set to the values for email and username we stored in the local variables earlier. The values are injected into the HTML using a bit of inline PHP and a call to the se() function to format the values to be HTML compatible.

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

After

Caption(s) (required) ✓

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

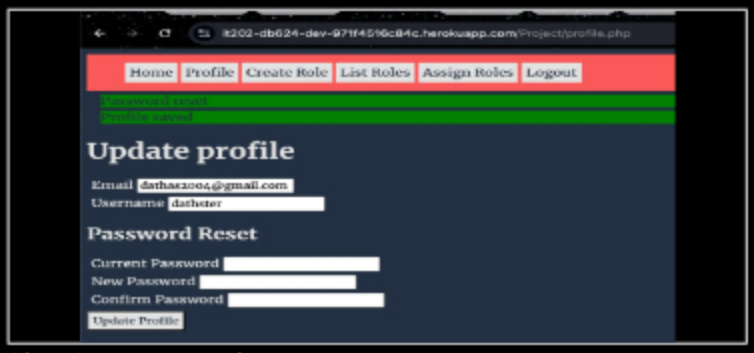
Sub-Task 100%

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 password reset

Caption(s) (required) ✓

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

Sub-Task 100%

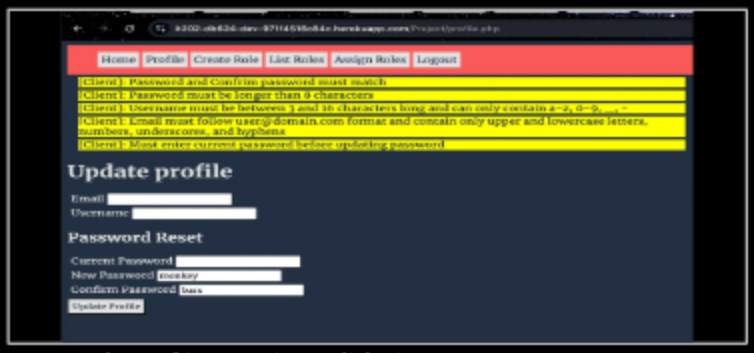
Group: User Profile
Task #3: Edit Profile Website Page

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



screenshot of javascript validation 1

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

chosen email already in use error

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 taken error

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

current password doesn't match what's in db error

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:

The script loads the user inputted email and username onto local variables and uses the get_user_id() function to load in the user's id. The email and username are both validated and errors are raised if these validations fail. Then it tries to execute an update query on the database where it updates the email and username columns for the record where the user_id matches the current user's ID. If an error occurs, we are checking if it has error 1062, and we find a match for the given username or email in the database, an error is raised.

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:

First, the current, new, and confirm passwords are loaded onto local variables. If the passwords don't match or the current password isn't provided, an error is raised. Then, all the passwords are checked to see if they follow correct format of minimum length of 8. After all these checks are passed, the password hash associated with user is loaded from database and current password is also hashed. These two hashes are checked and if they match, the new password is hashed and inserted into the database, replacing old password. If not, errors are raised.

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/Dathster/db624-it202-007/pull/20

URL
https://github.com/Dathster/db624-it202-007/pul

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

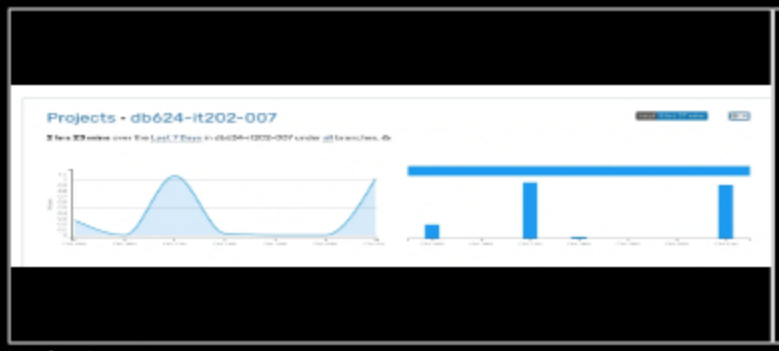
COLLAPSE

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

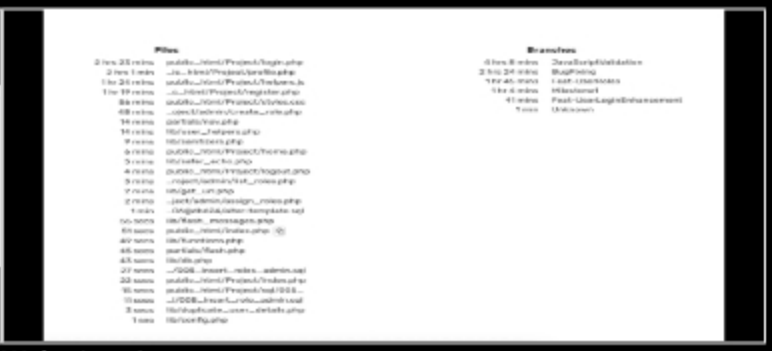
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Wakatime 1



Wakatime 2

End of Task 1

Task

Group: Misc

100%

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

Weight: ~33%

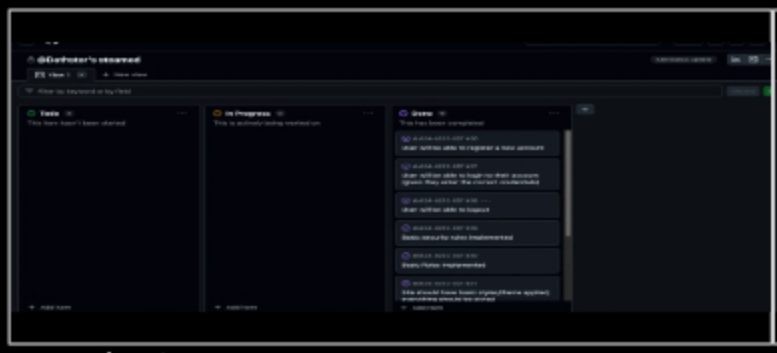
Points: ~0.33

COLLAPSE

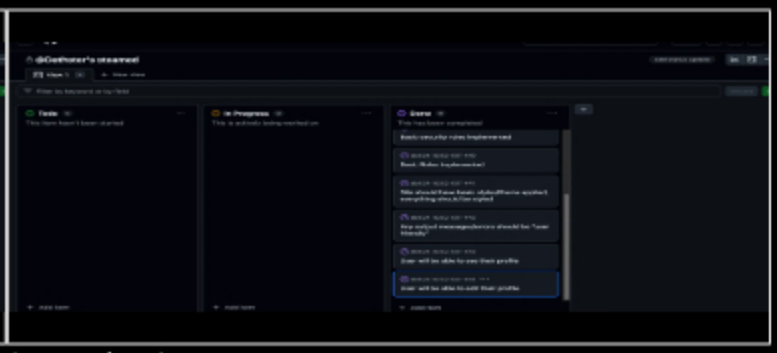
Task Screenshots

Gallery Style: 2 Columns

4 2 1



screenshot 1



Screenshot 2

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/Dathster/projects/4/views/1

URL

https://github.com/users/Dathster/projects/4/vie

End of Task 3

End of Group: Misc

Task Status: 3/3

End of Assignment