

Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT202-452-M2024/it202-module-6-milestone-1-2024/grade/dr475>

IT202-452-M2024 - [IT202] Module 6 Milestone 1 2024

Submissions:

Submission Selection

1 Submission [active] 7/8/2024 12:33:04 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

Tasks: 25 Points: 10.00

● User Registration (2 pts.)

▲ COLLAPSE ▲

● Task #1 - Points: 1

Text: Screenshot of form on website page

● 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.

#1) Screenshot of the form (ensure valid data is filled in)



Caption (required) ✓

Describe/highlight what's being shown

All form data filled out on registration page

URL (required) ✓

Prod link to the registration page

<https://it202-dr475-prod-7559be2e2ad4.herokuapp.com/Project/register.php>



#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)

The image consists of twelve screenshots arranged in a 3x4 grid. Each screenshot shows a registration form with different fields highlighted in red to indicate validation errors. The fields include Email, Username, Password, and Confirm. The errors shown are:

- Row 1: Email (Email is required), Username (Username is required), Password (Password is required), Confirm (Confirm is required).
- Row 2: Email (Email is required), Username (Username is required), Password (Password is required), Confirm (Confirm is required).
- Row 3: Email (Email is required), Username (Username is required), Password (Password is required), Confirm (Confirm is required).

Caption (required) ✓

Describe/highlight what's being shown

Javascript validation for each field, and their respective alerts upon errors.



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

The image shows a registration form on a web browser. The URL in the address bar is <https://it202-dr475-dev-cf3330aa7e7a.herokuapp.com/Project/register.php>. The form has four input fields: Email, Username, Password, and Confirm. The Email field contains "redziniakdavid@gmail.com". A yellow horizontal bar above the form displays the error message "The chosen email is not available.". Below the form, there is a "Register" button.

Caption (required) ✓

Describe/highlight what's being shown

Error message shown when email is taken

#4) Demonstrate username already in use message (message text doesn't need to be exact, but should be clear)

The chosen username is not available.

Email redziniak@gmail.com

Username david

Password

Confirm

Register

Caption (required) ✓

Describe/highlight what's being shown

Username is taken error message

#5) Demonstrate user-friendly message of new account being created

Successfully registered!

Email redziniak@gmail.com

Username davidee

Password

Confirm

Register

Caption (required) ✓

Describe/highlight what's being shown

Successful registration message

[COLLAPSE](#)

Task #2 - Points: 1

Text: Screenshot of the form code

ⓘ Details:

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

#1) Show the html of the form



```
<!-- UCTD: dn475 -->
<!-- Date: 07/08/2024 -->
<form onsubmit="return validate(this)" method="POST">
  <div>
    <label for="email">Email</label>
    <input type="email" name="email" value=<?php echo se($_POST, "email", "", false); ?>" required />
  </div>
  <div>
    <label for="username">Username</label>
    <input type="text" name="username" value=<?php echo se($_POST, "username", "", false); ?>" required maxlength="30" />
  </div>
  <div>
    <label for="pwd">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>
```

You, 3 weeks ago via PR #20 • Add registration functionality

Caption (required) ✓

Describe/highlight what's being shown

HTML code of the registration form

Explanation (required) ✓

Briefly explain the html for each field including the chosen attributes

PREVIEW RESPONSE

Email - Input box for the email of type "email"; it is required; the name="email" so it can be targeted using a query selector, and the value is set to the \$POST["email"] variable so upon page reload it saves the previous value.

Username - Input box for the username of type "text"; it is required with a max input length of 30; the name="username" so it can be targeted using a query selector, and the value is set to the \$POST["username"] variable so upon page reload it saves the previous value.

Password - Input box for the password of type "password" so it does not show the currently inputted text; it is required with a minimum length of 8; the name="password" so it can be targeted using a query selector

Confirm - Input box similar to the password input, it is required to have the same value as password as it is used to validate that the password inputted is accurate.

[COLLAPSE](#)

Task #3 - Points: 1

Text: Screenshot of the client-side and server-side validation code

1 Details:

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

#1) Show the JavaScript validations of the form (include any extra files related if you made separate files)



```

// UCID: dr475
// Date: 07/08/24
// Logic Test Practice
// Takes in a value and parses it through validateInput()
// Returns a string
// Takes in an array, a key, and default value and will return the value from the array if the key exists or the default value
// Can pass in flag to determine if the value will immediately return or just return so it can be set to a variable
// Returns object, If no value, default is "", else value + true
// If the value is in the array, then it will return the value
// Otherwise, it will return the default value
// Else if the object has a key, it will return the value
// Otherwise, it will return the default value
// Returns value + true
// Message 05-05-2024 is the same where $s is for $session
// $s is to keep track of previous message
// If $s_error[$key] is in $s_error[$key]
// $previous = $s_error[$key]

```

```

var confirmPassword = document.querySelector("[name='confirm']").value;
// UCID: dr475
// Date: 07/08/24
// Check if both fields are empty
if (pass === "") {
    alert("Client: Password Field cannot be empty.");
    return false;
}
if (confirmPass === "") {
    alert("Client: Confirm Password field cannot be empty.");
    return false;
}

// Check password lengths
if (pass.length < 8 || confirmPass.length < 8) {
    alert("Client: Password length cannot be less than 8 characters.");
    return false;
}

// Check if passwords match
if (pass !== confirmPass) {
    alert("Client: Password and Confirm Password do not match.");
    return false;
}
return true;

```

Caption (required) ✓

Describe/highlight what's being shown

Code for JS validation

Explanation (required) ✓

Explain in concise steps how this logically works

[PREVIEW RESPONSE](#)

The HTML form calls the validate function which will return false (and won't send a request to the server) if any of the input fields are not valid. The function goes through the email and username and uses Regex to validate the length and characters in the input values, and check if they're empty. It also goes through both passwords and check if they're empty, of required length, and if they match. It ultimately returns true when there are no errors found.

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



```

// PHP
// UCID: dr475
// Date: 07/08/24
// Logic Test Practice
// Takes in a value and parses it through validateInput()
// Returns a string
// Takes in an array, a key, and default value and will return the value from the array if the key exists or the default value
// Can pass in flag to determine if the value will immediately return or just return so it can be set to a variable
// Returns object, If no value, default is "", else value + true
// If the value is in the array, then it will return the value
// Otherwise, it will return the default value
// Else if the object has a key, it will return the value
// Otherwise, it will return the default value
// Returns value + true
// Message 05-05-2024 is the same where $s is for $session
// $s is to keep track of previous message
// If $s_error[$key] is in $s_error[$key]
// $previous = $s_error[$key]

```

```

...
1 // PHP
2 // UCID: dr475
3 // Date: 07/08/24
4 Function Flash($msg = "", $color = "info")
5 {
6     $message = ["text" -> $msg, "color" -> $color];
7     If ($s_error[$key] == $msg) {
8         array_push($SESSION['flash'], $message);
9     } Else {
10        $SESSION['flash'] = array();
11        array_push($SESSION['flash'], $message);
12    }
13 }
14
15 Function getMessages()

```

```

<?php
// UCID: dr475
// Date: 07/06/24
// For this we'll turn on error output so we can try to see any problems on the screen
// errors will be active for any code that shouldn't require this one
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

function getDB($host, $username, $password) {
    global $db;
    // This function returns an existing connection or creates a new one if needed
    // and assigns it to the $db variable
    if(!isset($db)) {
        try {
            $db = __DIR__ . '/config.php'; // This gets the absolute path regardless of where the file is being called from
            // This gets the absolute path to this file, then we append the relative path up a directory and inside lib
            require_once($db, 'config.php'); // $db is our credentials
            // We're going to populate our connection
            $connection_string = "mysql:host=$host;dbname=$database;charset=utf8mb4";
            // This connects to the database
            $db = new PDO($connection_string, $username, $password);
            // The default fetch mode is PSQL_FETCH_ASSOC which returns the data as both an indexed array and associative array
            // We'll overwrite the defaults here so it's always returned as an associative array
            $db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
        } catch(PDOException $e) {
            // We log the error
            error_log("getDB() error: " . var_export($e, true));
            $db = null;
            throw new Exception("Error connecting to database, see logs for further information");
        }
    }
    return $db;
}

```

```

<?php
// UCID: dr475
// Date: 07/06/24
function users_check_duplicate($errorInfo) {
    if ($errorInfo[1] === 1062) {
        // NOTE: this assumes your table name is 'Users', edit it accordingly
        preg_match('/Users\.(.*\w)/', $errorInfo[2], $matches);
        if (isset($matches[1])) {
            flash("The chosen " . $matches[1] . " is not available.", "warning");
        } else {
            // Come up with a nice error message
            flash("An unhandled error occurred", "danger");
            // This will log the output to the terminal/console that's running the php server
            error_log(var_export($errorInfo, true));
        }
    } else {
        // Come up with a nice error message
        flash("An unhandled error occurred", "danger");
        // This will log the output to the terminal/console that's running the php server
        error_log(var_export($errorInfo, true));
    }
}

```

```

<?php
// UCID: dr475
// Date: 07/06/24
if (!empty($_POST['email'])) {
    $email = $_POST['email'];
    $email = trim($email);
    if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $email = $email;
    } else {
        $email = null;
    }
}

$username = $_POST['username'];
$username = trim($username);
$username = htmlspecialchars($username);
if (empty($username)) {
    flash("Email must not be empty", "danger");
    $hasError = true;
} else {
    $email = $email;
}

$password = $_POST['password'];
$password = trim($password);
if (empty($password)) {
    flash("Email must not be empty", "danger");
    $hasError = true;
} else {
    $password = $password;
}

$confirm_password = $_POST['confirm_password'];
$confirm_password = trim($confirm_password);
if (empty($confirm_password)) {
    flash("Email must not be empty", "danger");
    $hasError = true;
} else {
    $confirm_password = $confirm_password;
}

if ($password !== $confirm_password) {
    flash("Passwords must match", "danger");
    $hasError = true;
}

if ($hasError) {
    echo "There were errors in the form";
} else {
    $password = password_hash($password, PASSWORD_BCRYPT);
    $stmt = $db->prepare("INSERT INTO Users (email, password, username) VALUES(:email, :password, :username)");
    try {
        $stmt->execute([':email' => $email, ':password' => $password, ':username' => $username]);
        flash("Successfully registered!", "success");
    } catch (PDOException $e) {
        users_check_duplicate($errorInfo);
    }
}

```

```

if (empty($password)) {
    flash("Password must be empty", "danger");
    $hasError = true;
}
if (empty($confirm_password)) {
    flash("Confirm password must be empty", "danger");
    $hasError = true;
}
if (strlen($password) < 8) {
    flash("Password too short", "danger");
    $hasError = true;
}
if (strlen($password) > 50) {
    flash("Passwords must match", "danger");
    $hasError = true;
}
if ($hasError) {
    // ...
    $hash = password_hash($password, PASSWORD_BCRYPT);
    $stmt = $db->prepare("INSERT INTO Users (email, password, username) VALUES(:email, :password, :username)");
    try {
        $stmt->execute([':email' => $email, ':password' => $hash, ':username' => $username]);
        flash("Successfully registered!", "success");
    } catch (PDOException $e) {
        users_check_duplicate($errorInfo);
    }
}

```

Caption (required) ✓

Describe/highlight what's being shown

PHP server side validation code, including lib functions

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

First, the server checks if the POST request received has the required values to register (email, username, password, confirm password). Then, it validates the data (length, valid characters, password and confirm password being the same). If any error is found, it is returned to the client and displayed on the screen. Ultimately, if the data is valid, it hashes the password and sends a query to the database inserting a new row in the Users table with the requested values for email, username, and password. If the query is successful, it returns a success message on the user's screen "Successfully registered!", but if an error occurs, it is shown.

SE/safer_echo = Takes in an array, key, default value and returns the value from the array if it exists or the default provided value.

flash() = Shows the user a provided message with defined styling

users_check_duplicates() = Shows the user an error message if the provided error contains an error pertaining to a duplicate user in the database

getDB() = Provides a connection to the DB given a connection string in config.php

Task #4 - Points: 1**Text:** Screenshot of the Users table with a valid user entry**Checklist**

*The checkboxes are for your own tracking

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

#1) Show valid data per the checklist

	email	password	username	created	modified	lastLogin
1	test@gmail.com	1234567890	test	2024-07-07 16:09:08	2024-07-07 16:09:08	2024-07-07 16:09:08
2	test2@gmail.com	1234567890	test2	2024-07-07 16:09:08	2024-07-07 16:09:08	2024-07-07 16:09:08
3	test3@gmail.com	1234567890	test3	2024-07-07 16:09:08	2024-07-07 16:09:08	2024-07-07 16:09:08
4	test4@gmail.com	1234567890	test4	2024-07-07 16:09:08	2024-07-07 16:09:08	2024-07-07 16:09:08
5	test5@gmail.com	1234567890	test5	2024-07-07 16:09:08	2024-07-07 16:09:08	2024-07-07 16:09:08

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

The user table

Task #5 - Points: 1**Text:** Explain the registration logic in a step-by-step manner from when the page loads to when the data is saved to the DB**i Details:**

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

May be worthwhile using a list.

Response:

1. The user retrieves the register.php page from the server which first resets all the session values with reset_session() (flash, email, username, etc)
2. The user is then presented with the form that is required to fill out in order to create a new user
3. The user then fills out all the input fields (email, username, password, confirm password) and presses the submit button.
4. The JS function validate() is then called to provide client side validation of the fields before it is officially sent to the PHP server.
5. If any of the fields are not valid, it tells the user. If it is all valid, it is then sent to via a POST request to the PHP server for processing.
6. The PHP code in register.php then processes the data and provides extra server-side input validation (lengths of all input text, checks if email is taken, password and confirm password is the same, username is taken).
7. If everything passes validation, the fields are then submitted to the DB with an INSERT SQL query with the values and returns a successful message to the client.
8. Furthermore, if there are any errors when processing the query, it is then returned to the client.

 Task #6 - Points: 1

 Text: Include pull request links related to this feature

 Details:

Should end in /pull/#

URL #1

<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/20>

URL #2

<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/25>

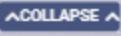
URL #3

<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/45>

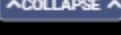
URL #4

<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/47>

 User Login (2 pts.)



 Task #1 - Points: 1

 Text: Screenshot of form on website page

 Details:

Thoughtful LCC should be applied to all parts (must differ from the bulk LCC given via the

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.

#1) Two Screenshot of the form (one with valid email data filled and one with valid username data filled)

A screenshot of a web browser displaying a login page. The URL is <https://it202-dr475-dev-cf3330aa7e7a.herokuapp.com/Project/login.php>. The page has a blue header with 'Join Us' and 'hashgraph/hedera...' links. Below the header are two buttons: 'Login' and 'Register'. There are two input fields: 'Email/username:' containing 'davidee' and 'Password:' containing '*****'. A 'Login' button is at the bottom.

A screenshot of a web browser displaying a login page. The URL is <https://it202-dr475-dev-cf3330aa7e7a.herokuapp.com/Project/login.php>. The page has a blue header with 'Join Us' and 'hashgraph/hedera...' links. Below the header are two buttons: 'Login' and 'Register'. There are two input fields: 'Email/username:' containing 'redziniak@gmail.com' and 'Password:' containing '*****'. A 'Login' button is at the bottom.

Caption (required) ✓

Describe/highlight what's being shown

Valid form data(user/email)

URL (required) ✓

Prod link to the login page

<https://it202-dr475-prod-7559be2e2ad4.herokuapp.com/Project/login.php>

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

A screenshot of a web browser displaying a login page. The URL is <https://it202-dr475-dev-cf3330aa7e7a.herokuapp.com/Project/login.php>. The page has a blue header with 'Join Us' and 'hashgraph/hedera...' links. Below the header are two buttons: 'Login' and 'Register'. There are two input fields: 'Email/username:' containing '' and 'Password:' containing '*****'. A modal dialog box appears with the message 'it202-dr475-dev-cf3330aa7e7a.herokuapp.com says: (Client): invalid.username field cannot be empty.' A 'OK' button is at the bottom right of the dialog.

A screenshot of a web browser displaying a login page. The URL is <https://it202-dr475-dev-cf3330aa7e7a.herokuapp.com/Project/login.php>. The page has a blue header with 'Join Us' and 'hashgraph/hedera...' links. Below the header are two buttons: 'Login' and 'Register'. There are two input fields: 'Email/username:' containing 'redziniak@gmail.com' and 'Password:' containing ''. A modal dialog box appears with the message 'it202-dr475-dev-cf3330aa7e7a.herokuapp.com says: (Client): Password field cannot be empty.' A 'OK' button is at the bottom right of the dialog.

A screenshot of a web browser displaying a login page. The URL is <https://it202-dr475-dev-cf3330aa7e7a.herokuapp.com/Project/login.php>. The page has a blue header with 'Join Us' and 'hashgraph/hedera...' links. Below the header are two buttons: 'Login' and 'Register'. There are two input fields: 'Email/username:' containing '' and 'Password:' containing''. A modal dialog box appears with the message 'it202-dr475-dev-cf3330aa7e7a.herokuapp.com says: (Client): invalid.username field cannot be empty; (Client): Password field cannot be empty.' A 'OK' button is at the bottom right of the dialog.

A screenshot of a web browser displaying a login page. The URL is <https://it202-dr475-dev-cf3330aa7e7a.herokuapp.com/Project/login.php>. The page has a blue header with 'Join Us' and 'hashgraph/hedera...' links. Below the header are two buttons: 'Login' and 'Register'. There are two input fields: 'Email/username:' containing '' and 'Password:' containing''. A modal dialog box appears with the message 'it202-dr475-dev-cf3330aa7e7a.herokuapp.com says: (Client): invalid.username field cannot be empty; (Client): Password field cannot be empty.' A 'OK' button is at the bottom right of the dialog.

Login | Register

Email/username: Password:

it202-dr475-dev-cf3330aa7e7a.herokuapp.com says
[Error] Username must be 3-10 characters and contain only
characters (a-z, A-Z, _ , -)

OK

Login | Register

Email/username: Password:

it202-dr475-dev-cf3330aa7e7a.herokuapp.com says
[Error] Password length cannot be less than 6 characters.

OK

Logout

Login | Register

Email/username: Password:

it202-dr475-dev-cf3330aa7e7a.herokuapp.com says
[Error] Neither seems valid.

OK

Caption (required) ✓

Describe/highlight what's being shown

Client side validation

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



← → G it202-dr475-dev-cf3330aa7e7a.herokuapp.com/Project/login.php

Join Us hashgraph/hedera-...

Login Register

Email not found

Email/username:

Password:

Login

Caption (required) ✓

Describe/highlight what's being shown

Account not found

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



[Join Us](#) [hashgraph/hedera-...](#)

[Login](#) [Register](#)

Invalid password

Email/username:

Password:

[Login](#)

Caption (required) ✓

Describe/highlight what's being shown

Password incorrect

#5) Demonstrate successful login message and the destination page (i.e., home or some landing/dashboard page)



[Join Us](#) [hashgraph/hedera-...](#)

[Home](#) [Profile](#) [Logout](#)

Welcome, test

Home

Caption (required) ✓

Describe/highlight what's being shown

Successful login

#6) Demonstrate session data being set (captured from server logs)



```
2023-07-08T17:08:07.303Z -07:00 [2023-07-08T17:08:07.303Z] [error] [req=1] [method=POST] [path=/Project/login.php] [user=1] [2023-07-08T17:08:07.303Z] [dev=1] [ip=127.0.0.1] [useragent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/120.0.0] [language=en-US] [accept=application/json] [content-type=application/x-www-form-urlencoded] [x-forwarded-for=127.0.0.1] [x-forwarded-port=443] [x-forwarded-proto=https] [x-request-id=1] [x-session-id=1] [x-user-ip=127.0.0.1] [x-user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/120.0.0] [x-user-language=en-US] [x-user-accept=application/json] [x-user-content-type=application/x-www-form-urlencoded] [x-user-forwarded-for=127.0.0.1] [x-user-forwarded-port=443] [x-user-forwarded-proto=https] [x-user-request-id=1] [x-user-session-id=1] [x-user-user-id=1] [x-user-user-name=test] [x-user-user-email=test@example.com] [x-user-user-status=200] [bytes=1200] [x-user-user-https=true]
```

Caption (required) ✓

Describe/highlight what's being shown

Heroku logs

Task #2 - Points: 1

Text: Screenshot of the form code

i Details:

Should have the appropriate type attributes for the fields.

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

#1) Show the html of the form



```
?>
<!-- UCID: dr475 -->
<!-- Date: 07/08/2024 -->
<form onsubmit="return validate(this)" method="POST">
  <div>
    <label for="email">Email/username:</label>
    <input type="text" name="email" required />
  </div>
  <div>
    <label for="pw">Password:</label>
    <input type="password" id="pw" name="password" required minlength="8" />
  </div>
  <input type="submit" value="Login" />
</form>
```

Caption (required) ✓

Describe/highlight what's being shown

HTML code for form

Explanation (required) ✓

Briefly explain the html for each field including the chosen attributes

PREVIEW RESPONSE

Email/Username - Input box for the email or username of type "text"; it is required; the name="email" so it can be targeted using a query selector.

Password - Input box for the password of type "password" so input value is not visible; it is required with a minimum

#2) Show the JavaScript validations of the form (include any extra files related if you made separate files)



```

<script>
// UCID: 00479
// Date: 07/09/24
function validateForm() {
    var emailOrUser = document.querySelector("[name='emailOrUser']").value;
    // Check if email or username is empty
    if (emailOrUser === "") {
        alert("Client: Email/username field cannot be empty.");
        return false;
    }

    // If email, check if email is valid using regex
    if (emailOrUser.includes("@")) {
        if (!/^(?=[a-zA-Z0-9.]{3,64}@([a-zA-Z.]{1,64})\.(?=[a-zA-Z]{2,4}))\.\w+$/i.test(emailOrUser)) {
            alert("Client: " + emailOrUser + " is invalid.");
            return false;
        }
    } else {
        // Username validation
        if (!/^(?=[a-zA-Z0-9_]{3,30})(?=[a-zA-Z_]{1,30})\.\w+$/i.test(emailOrUser)){
            alert("Client: Username must be 3-30 characters and contain valid characters (a-z, 0-9, _, or -)");
            return false;
        }
    }

    var pass = document.querySelector("[name='password']").value;
    // Check if password is empty
    if (pass === "") {
        alert("Client: Password field cannot be empty.");
        return false;
    }

    // Check password length
    if (pass.length < 8) {
        alert("Client: Password length cannot be less than 8 characters.");
        return false;
    }

    return true;
}
</script>

```

Caption (required) ✓

Describe/highlight what's being shown

JS client side validation

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

The HTML form calls the validate function which will return false (and won't send a request to the server) if any of the input fields are not valid. The function goes through the email/username and uses Regex to validate the length and characters in the input values, and check if they're empty. It also goes through the password value and checks if they're the required length. It ultimately returns true when there are no errors found.

#3) Show PHP validations (include any lib content)



```

if($this->isPost())
{
    $this->addInput('emailOrUser');
    if($this->isPost('password')) {
        $password = $this->getInput('password');
        $password = md5($password);
        $password = $this->getInput('password');
        $password = md5($password);
    }
}

if(empty($emailOrUser)) {
    $this->addError('Email/username field cannot be empty.');
}

if(empty($password)) {
    $this->addError('Password field cannot be empty.');
}

if(empty($emailOrUser) || empty($password)) {
    $this->addError('Both fields are required.');
}

if(strlen($emailOrUser) > 30) {
    $this->addError('Email/username must be less than 30 characters long.');
}

if(strlen($password) > 30) {
    $this->addError('Password must be less than 30 characters long.');
}

if(strlen($emailOrUser) < 3) {
    $this->addError('Email/username must be at least 3 characters long.');
}

if(strlen($password) < 8) {
    $this->addError('Password must be at least 8 characters long.');
}

if($this->hasErrors()) {
    $this->show();
}

```

```

if($this->isPost())
{
    $this->addInput('emailOrUser');
    $this->addInput('password');
    $password = md5($password);
    $password = $this->getInput('password');
    $password = md5($password);
    $password = $this->getInput('password');
    $password = md5($password);

    if(empty($emailOrUser)) {
        $this->addError('Email/username field cannot be empty.');
    }

    if(empty($password)) {
        $this->addError('Password field cannot be empty.');
    }

    if(empty($emailOrUser) || empty($password)) {
        $this->addError('Both fields are required.');
    }

    if(strlen($emailOrUser) > 30) {
        $this->addError('Email/username must be less than 30 characters long.');
    }

    if(strlen($password) > 30) {
        $this->addError('Password must be less than 30 characters long.');
    }

    if(strlen($emailOrUser) < 3) {
        $this->addError('Email/username must be at least 3 characters long.');
    }

    if(strlen($password) < 8) {
        $this->addError('Password must be at least 8 characters long.');
    }

    if($this->hasErrors()) {
        $this->show();
    }
}

```

```

<?php
// UCID: dn475
// Date: 07/08/24
** Safe echo function
* Takes an array, a key, and default value and will return the value from the array if the key exists or the default value.
* Can pass a flag to determine if the value will immediately echo or just return so it can be set to a variable
function safe_echo($arr, $key, $default = "", $echoNow = true) {
    if (!is_array($arr) || !is_string($key) || !is_string($default)) {
        return $default;
    }
    if (array_key_exists($key, $arr)) {
        $value = $arr[$key];
    } else {
        $value = $default;
    }
    if ($echoNow == true) {
        // Used in this case where $k is for the $arr key
        echo $value;
    }
    if ($value != $default) {
        return $value;
    }
}
if (isset($_GET['action'])) {
    if ($_GET['action'] == "register") {
        header("Location: http://127.0.0.1:8080/home.php");
    } else if ($_GET['action'] == "login") {
        header("Location: http://127.0.0.1:8080/login.php");
    }
}

function safer_echo($arr, $k = null, $default = "", $echoNow = true){
    return safe_echo($arr, $k, $default, $echoNow);
}

```

```

<?php
// UCID: dn475
// Date: 07/08/24
function flash($msg = "", $color = "Info")
{
    $message = ["text" => $msg, "color" => $color];
    if (!isset($_SESSION['flash'])) {
        array_push($_SESSION['flash'], $message);
    } else {
        $_SESSION['flash'] = array();
        array_push($_SESSION['flash'], $message);
    }
}

function getMessages()
{
    if (isset($_SESSION['flash'])) {
        $flashes = $_SESSION['flash'];
        $_SESSION['flash'] = array();
        return $flashes;
    }
    return array();
}

```

```

<?php
// UCID: dn475
// Date: 07/08/24

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($username)
{
    return preg_match('/^@[a-zA-Z0-9_-]{3,16}$/', $username);
}
function is_valid_password($password)
{
    return strlen($password) >= 8;
}

```

```

<?php
// UCID: dn475
// Date: 07/08/24
// For this we'll turn on error output so we can try to see any problems on the screen
// This will be active for any script that includes/requires this one
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);
set_time_limit(0);
global $db;
// This function returns an existing connection or creates a new one if needed
// And assigns to the $db variable
try {
    // $DB - lets get the current path regardless of where the file is being called from
    // $DB gets the absolute path to this file, then we append the relative one (up a directory and inside lib)
    require_once __DIR__ . "/config.php";
    $db = new PDO("mysql:host=$host;dbname=$database;charset=utf8mb4");
    // Using the PDO connector create a new connect to the DB
    // $db = new PDO($connection_string, $username, $password);
    // If no error occurs we're connected
    // $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
    // $db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
} catch (Exception $e) {
    error_log("getDB() error: " . var_export($e, true));
    $db = null;
    throw new Exception("Error connecting to database, see logs for further information");
}
return $db;
}

```

Caption (required) ✓

Describe/highlight what's being shown

PHP Code and its required lib functions

Explanation (required) ✓

Explain in concise steps how this logically works

PREVIEW RESPONSE

First, login.php imports required functions from lib files. Then, it checks if the client sent a POST request with the required fields (email, password). It then validates and sanitizes the email/username by calling helper functions to verify the validity of the input values. It also then validates the password length and values. Ultimately, if there are not errors, it submits a SELECT SQL query locating the user row with the given parameters (email/username). It then checks the password sent by the user and compares it to the unhashed password stored in the database if it matches. Upon successful authentication, tries to check the user's role in the DB and then sets the session user role variable so it can be used in future requests, then redirects the client to the home.php page. If any errors occur it displays the error on the client's page (invalid password, email not found, etc).

SE/safer_echo = Takes in an array, key, default value and returns the value from the array if it exists or the default provided value.

flash() = Shows the user a provided message with defined styling

sanitization = Methods for username, email and password to trim and validate a string to specifications

getDB() = Provides a connection to the DB given a connection string in .env

^COLLAPSE ^

TASK #3 - POINTS: 1

Text: Explain the login logic step-by-step from when the page loads to when the data is fetched from the DB and stored in the session

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.

Response:

1. The user retrieves the login.php page from the server.
2. The user is then presented with the form that is required to fill out in order to login
3. The user then fills out all the input fields (email/username, password) and presses the submit button.
4. The JS function validate() is then called to provide client side validation of the fields before it is officially sent to the PHP server.
5. If any of the fields are not valid, it tells the user. If it is all valid, it is then sent to via a POST request to the PHP server for processing.
6. The PHP code in login.php then processes the data and provides extra server-side input validation (lengths of all input text, checks if email/username is valid, password is valid).
7. If everything passes validation, the user is successfully authorized and the server gets the user role given to the user (admin) and saves it to the user's session variable for future page visits. The user is then redirected to the home page.
8. Furthermore, if there are any errors when processing the query, it is then returned to the client.



^COLLAPSE ^

Task #4 - Points: 1

Text: Include pull request links related to this feature

Details:

Should end in /pull/#

URL #1

<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/33>

URL #2

<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/46>

URL #3

<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/48>

^COLLAPSE ^

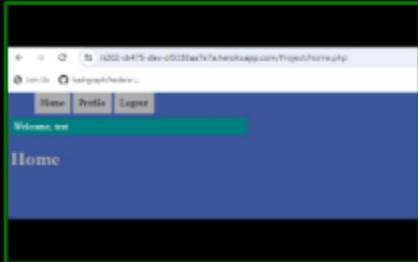
User Logout (1 pt.)

[^COLLAPSE ^](#)

Task #1 - Points: 1

Text: Capture the following screenshots

#1) Screenshot of the navigation when logged in (site)

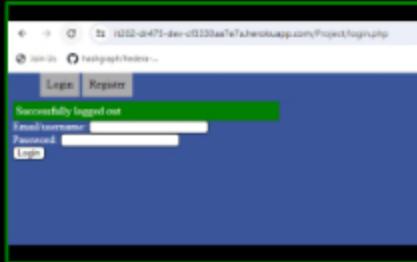


Caption (required) ✓

Describe/highlight what's being shown

User logged in

#2) Screenshot of the redirect to login with the user-friendly

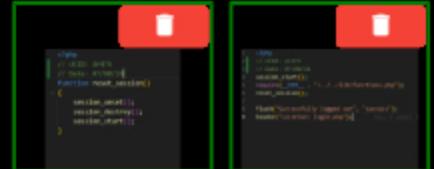


Caption (required) ✓

Describe/highlight what's being shown

User logged out

#3) Screenshot of the logout-related code showing the session is



Caption (required) ✓

Describe/highlight what's being shown

PHP code for logout

Explanation (required) ✓

Explain in concise steps how this logically works

[PREVIEW RESPONSE](#)

When the logout.php is visited, The server destroys the current session of the user (session unset, session destroy), and starts a new one (session start). There are no more session variables associated with the respective user. Then, it redirects to the login.php page (header("Location: login.php"))

[^COLLAPSE ^](#)

Task #2 - Points: 1

Text: Include pull request links related to this feature

Details:

Should end in /pull/#

URL #1

<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/21>

URL #2

<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/25>

URL #3

Basic Security Rules and Roles (2 pts.)

[COLLAPSE ^](#)

Task #1 - Points: 1

Text: Authentication Screenshots

i Details:

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

#1) Screenshot of the function that checks if a user is logged in



```
function is_logged_in($session, $destination = "login.php") {
    if ($session['user']) {
        if (is_logged_in($session)) {
            // User is logged in, so we can use a regular session variable
            // instead of a partial session variable
            $session['user'] = $session['partial']['user'];
            $session['partial'] = null;
        }
    } else {
        // User is not logged in, so we need to redirect them to the login page
        header("Location: $destination");
        exit();
    }
}
```

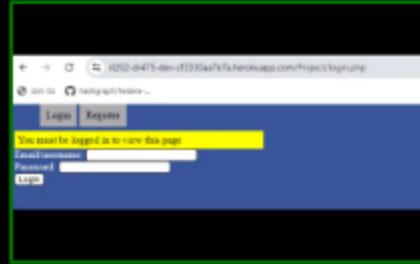
#2) Screenshot of the login check function being used (i.e. profile)



```
if (!is_logged_in($session, "profile.php")) {
    // User is not logged in, so we need to redirect them to the login page
    header("Location: login.php");
    exit();
}

// User is logged in, so we can use a regular session variable
// instead of a partial session variable
$session['user'] = $session['partial']['user'];
$session['partial'] = null;
```

#3) Demonstrate the user-friendly message of trying to manually



Caption (required) ✓

Describe/highlight what's being shown

PHP function is_logged_in

Explanation (required) ✓

Explain in concise steps how this logically works

[PREVIEW RESPONSE](#)

The is_logged_in function checks if the \$SESSION['user'] variable is set (it is set when a user logs in). If it is not, that means the user is not logged in, it redirects the user to the login.php page (or a different URL given in the parameters to the function) and displays a message displaying that they aren't logged in.

Caption (required) ✓

Describe/highlight what's being shown

is_logged_in being used in profile.php

Explanation (required) ✓

Explain in concise steps how this logically works

[PREVIEW RESPONSE](#)

Before executing the rest of the PHP on the page, it first calls the function is_logged_in, which will redirect the user if they aren't logged in. If is_logged_in returns true, the rest of the following PHP code will execute.

Caption (required) ✓

Describe/highlight what's being shown

Visiting profile.php while logged out.

[^COLLAPSE](#)

Task #2 - Points: 1

Text: Authorization Screenshots

Details:

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

#1)
Screenshot
of the



Caption (required) ✓

*Describe/highlight
what's being shown*
PHP code

Explanation (required)

✓
*Explain in concise steps
how this logically works*
[PREVIEW RESPONSE](#)
The has_role function
checks if the user is
logged in and checks if a
variable is set in the
session variables given
a specified role
parameter (e.g. "Admin")

#2)
Screenshot
of the role



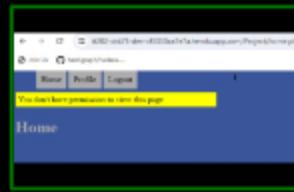
Caption (required) ✓

*Describe/highlight
what's being shown*
PHP code

Explanation (required)

✓
*Explain in concise steps
how this logically works*
[PREVIEW RESPONSE](#)
The PHP logic checks if
the user has the role
"Admin" in the session
variables. If it does have
the variable set, it
renders an additional list
of links to visit in
navigation bar. If it
doesn't, it isn't rendered.

#3)
Demonstrate
the user-



Caption (required) ✓

*Describe/highlight
what's being shown*
Visiting
admin/list_roles.php as
a regular user

[^COLLAPSE](#)

Task #3 - Points: 1

Text: Screenshots of UserRoles and Roles Tables

Details:

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

#1) UserRoles table with at least one valid entry (Table should have id, user_id, role_id, is_active, created, and modified columns)



The screenshot shows the MySQL Workbench interface with the UserRoles table selected. The table has columns: id, user_id, role_id, is_active, created_at, and updated_at. One row is visible with the following values:

	id	user_id	role_id	is_active	created_at	updated_at
	1	17	-1	1	2024-06-30 23:35:14	2024-06-30 23:35:14

Caption (required) ✓

Describe/highlight what's being shown

User with ID "17" has the role Admin (-1)

#2) Roles table with at least one valid entry (Table should have id, name, description, is_active, modified, and created columns)



The screenshot shows the MySQL Workbench interface with the Roles table selected. The table has columns: id, name, description, is_active, created_at, and updated_at. One row is visible with the following values:

	id	name	description	is_active	created_at	updated_at
	-1	Admin		1	2024-06-30 23:35:14	2024-06-30 23:35:14

Caption (required) ✓

Describe/highlight what's being shown

Admin role with id "-1"



[COLLAPSE](#)

Task #4 - Points: 1

Text: Explain how Roles and UserRoles tables work in conjunction with the Users table

#1 UserRoles



Explanation (required) ✓

What's the purpose of the UserRoles table?

PREVIEW RESPONSE

To create a list of users and their respective roles on the website.

#2 Roles



Explanation (required) ✓

How does Roles.is_active differ from UserRoles.is_active?

PREVIEW RESPONSE

Roles.is_active determines whether the role itself is still used in day to day functionality on the website (it could be a defunct role "0"). UserRoles.is_active pertains to the specific user's role on the website (if they are "Admin", it can be toggled off so they are not active as an "Admin")

Task #5 - Points: 1

Text: Include pull request links related to this feature

Details:

Should end in /pull/#

URL #1

<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/22>

URL #2

<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/32>

User Profile (2 pts.)

COLLAPSE

Task #1 - Points: 1

Text: View Profile Website Page

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.

#1) Show the profile form correctly populated on page load (username, email)
Form should have the following fields:



Caption (required) ✓

Describe/highlight what's being shown

User profile with pre-filled email and username

Task #2 - Points: 1

Text: Explain the logic step-by-step of how the data is loaded and populated when the profile page is visited

i Details:

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

May be worthwhile using a list.

Response:

1. The client requests the profile.php page from the server. It is only returned if the user is logged in, else it redirects to the login page.
2. Upon the successful page load, the form is populated with the email and username that are both retrieved using helper functions that access the sessions saved variables (email, username).
3. The profile.php page is then returned to the client with the html and JS included.

Task #3 - Points: 1

Text: Edit Profile Website Page

i Details:

Thoughtful LCC should be applied to all parts (must differ from the initial LCC given via the

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.

#1) (Two Screenshots) Demonstrate



Caption (required) ✓

Describe/highlight what's being shown
Username change

#2) Demonstrate the success



Caption (required) ✓

Describe/highlight what's being shown
Password changed

#3) Demonstrate all



Caption (required) ✓

Describe/highlight what's being shown
JS validation

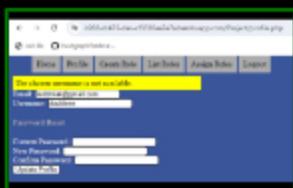
#4) Demonstrate PHP user-



Caption (required) ✓

Describe/highlight what's being shown
Email not available

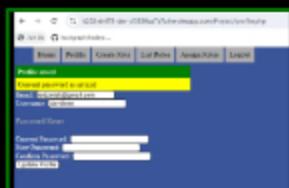
#5) Demonstrate PHP user-



Caption (required) ✓

Describe/highlight what's being shown
Username in use

#6) Demonstrate PHP user-



Caption (required) ✓

Describe/highlight what's being shown
Password is not correct

Task #4 - Points: 1

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

Details:

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

#1) Updating Username/Email



#2) Updating password



Explanation (required) ✓

Explain in concise steps how this logically works

 PREVIEW RESPONSE

Upon the form submit, the client sends a POST request to the server. In profile.php, it then sanitizes the email and username and validates the input. Then, it calls the helper function get_user_id which returns the user ID in the session corresponding to the id field in the database for the specified user. This user ID is then used to submit an UPDATE SQL query to the DB WHERE id = get_user_id(). This functionality allows the user to change both their username and email in one go. Then, it retrieves the updated data to be stored in the SESSION user variable by using a SELECT SQL query WHERE id = user's ID. If any errors occur, it is handled and displayed to the user.

Explanation (required) ✓

Explain in concise steps how this logically works

 PREVIEW RESPONSE

Upon the form submit, the client sends a POST request to the server. It checks if all the password fields are set and are valid (current, new, confirm new). Then it retrieves the user's password from the Users table in the DB WHERE id = user's ID. It then uses the password_verify function to check the hash of the password stored in the DB and compares it to the current_pass provided by the user in the POST request. If successful, it submits an UPDATE SQL query into the DB on the Users table which changes the password field to the hashed new password. If there is any error, it is handled (current password is invalid, new passwords don't match) and displayed to the user.

Task #5 - Points: 1

Text: Include pull request links related to this feature

Details:

Should end in /pull/#

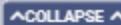
URL #1

<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/49>

URL #2

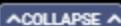
<https://github.com/davidredziniak/dr475-IT202-452-M2024/pull/24>

Misc (1 pt.)



Task #1 - Points: 1

Text: Explain what you did



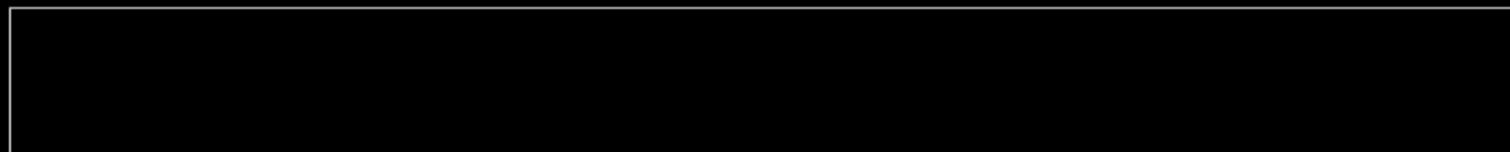
i Details:

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:

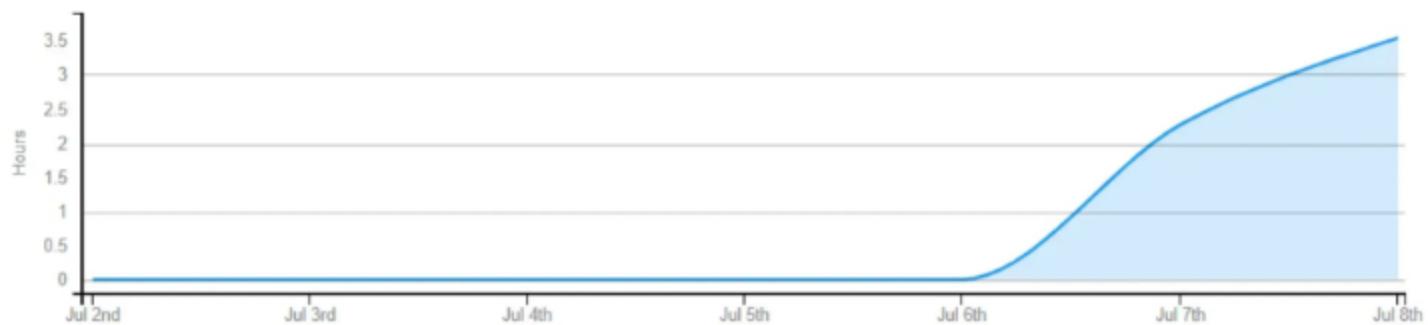
Gallery Style: Large View

Small	Medium	Large
-------	--------	-------



Projects • dr475-IT202-452-M2024

5 hrs 48 mins over the Last 7 Days in dr475-IT202-452-M2024 under all branches. ☁



Wakatime last 7 days working on Milestone 1

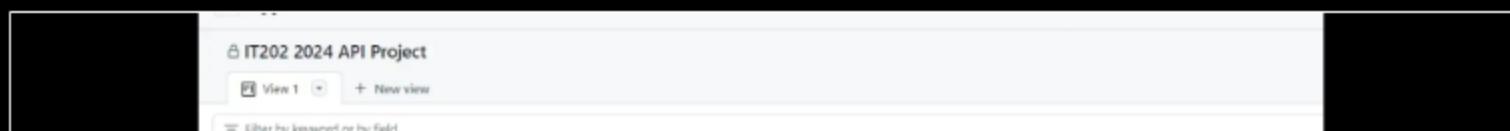
Task #2 - Points: 1

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

Task Screenshots:

Gallery Style: Large View

Small	Medium	Large
-------	--------	-------



A screenshot of a GitHub project board interface. The board is divided into three columns: 'Todo' (0 items), 'In Progress' (0 items), and 'Done' (10 items). The 'Done' column contains ten completed tasks, each with a small circular icon, a unique ID, and a brief description.

Todo	In Progress	Done
0	0	10
This item hasn't been started	This is actively being worked on	This has been completed
		d475-IT202-452-M2024 #36 User will be able to login
		d475-IT202-452-M2024 #35 User will be able to register a new account
		d475-IT202-452-M2024 #37 User will be able to logout
		d475-IT202-452-M2024 #38 Basic security rules implemented
		d475-IT202-452-M2024 #39 Basic Roles Implemented
		d475-IT202-452-M2024 #42 User profile view
		d475-IT202-452-M2024 #41 User friendly output messages/errors
		d475-IT202-452-M2024 #43 User will be able to edit their profile
		d475-IT202-452-M2024 #40 Basic site styling/themes
		d475-IT202-452-M2024 #50 Custom Styling

+ Add item + Add item + Add item

Github project board

Task #3 - Points: 1

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

URL #1

<https://github.com/users/davidredziniak/projects/2>

End of Assignment