Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-008-S2024/it202-milestone-1-2024/grade/bns24

IT202-008-S2024 - [IT202] Milestone 1 2024

Submissions:

Submission Selection

1 Submission [active] 3/31/2024 11:54:20 PM

Instructions

COLLAPSE

Prereqs:

1. 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
    - Mark the related GitHub Issues items as "done"
2. 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
3. Implement JavaScript validation on Register, Logout, and Profile (include "[Client]" in the output messages to differentiate between server-side validations)

Instructions:

Make sure you're in Milestone1 with the latest changes pulled
Ensure Milestone1 has been deployed to heroku dev
Gather the requested evidence and fill in the explanations per each prompt
Save the submission and generate the output PDF
Put the output PDF into your local repository folder
add/commit/push it to GitHub
Merge Milestone1 into dev
Locally checkout dev and pull the changes
Create and merge a pull request from dev to prod to deploy Milestone1 to prod
Upload this output PDF to Canvas

Branch name: Milestone1

Tasks: 26 Points: 10.00

User Registration (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of form on website page

Checklist

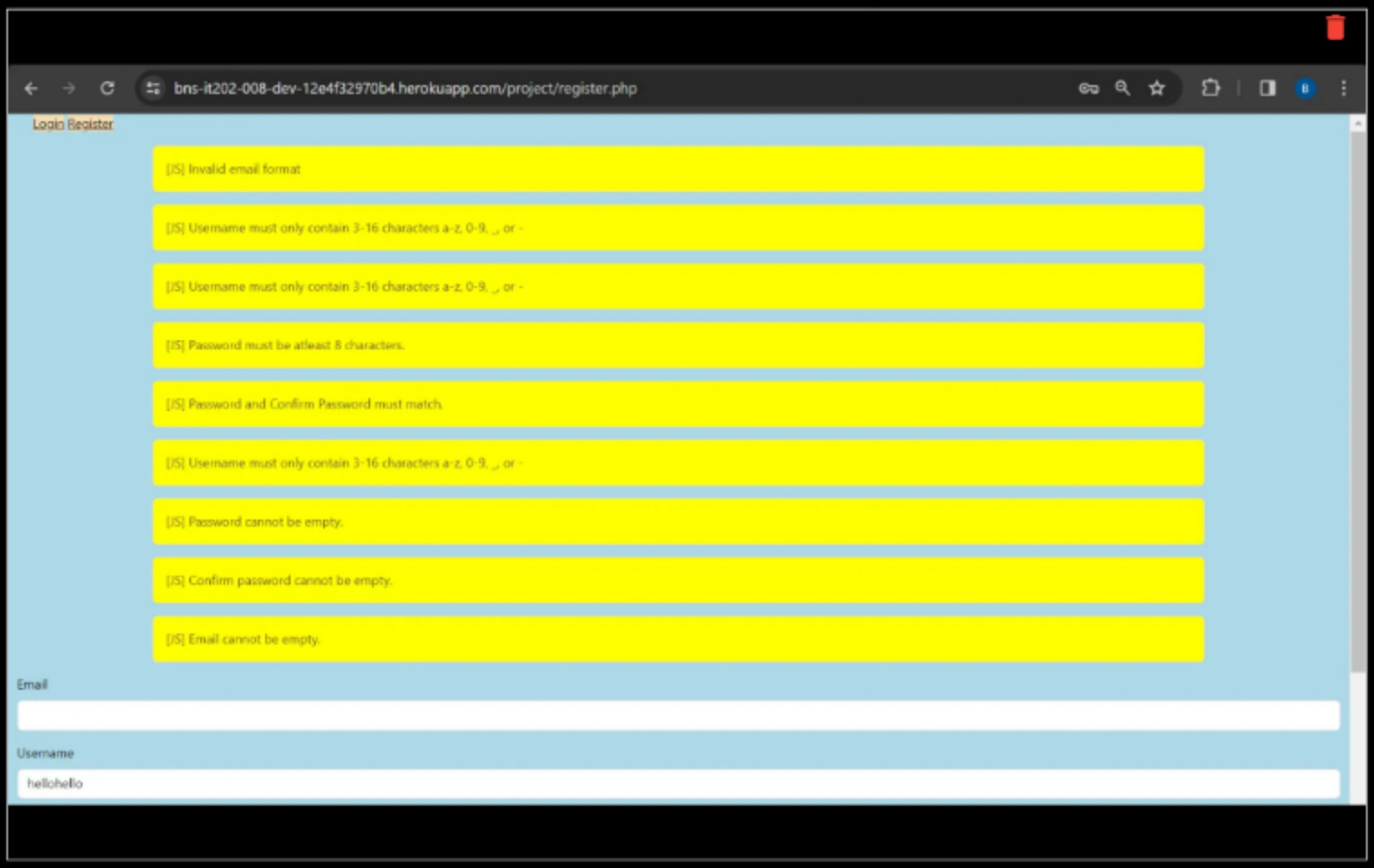
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Heroku dev url should be present in the address bar
#2	1	Should have thoughtful CSS applied
#3	1	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
#4	1	Demonstrate email already in use message (message text doesn't need to be exact, but should be clear)
#5	1	Demonstrate username already in use message (message text doesn't need to be exact, but should be clear)
#6	1	Demonstrate user-friendly message of new account being created

Task Screenshots:

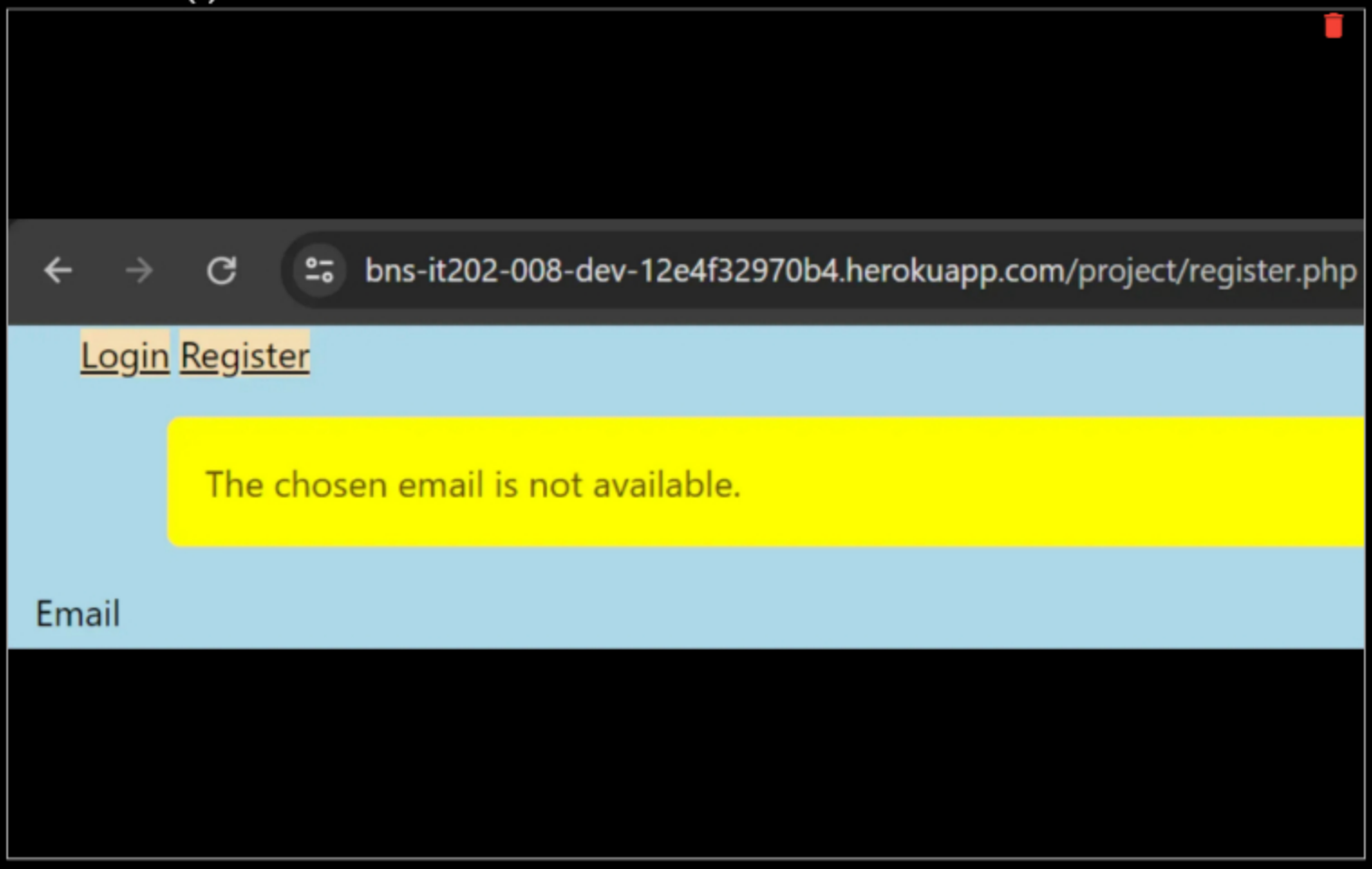
Gallery Style: Large View

Small Medium Large



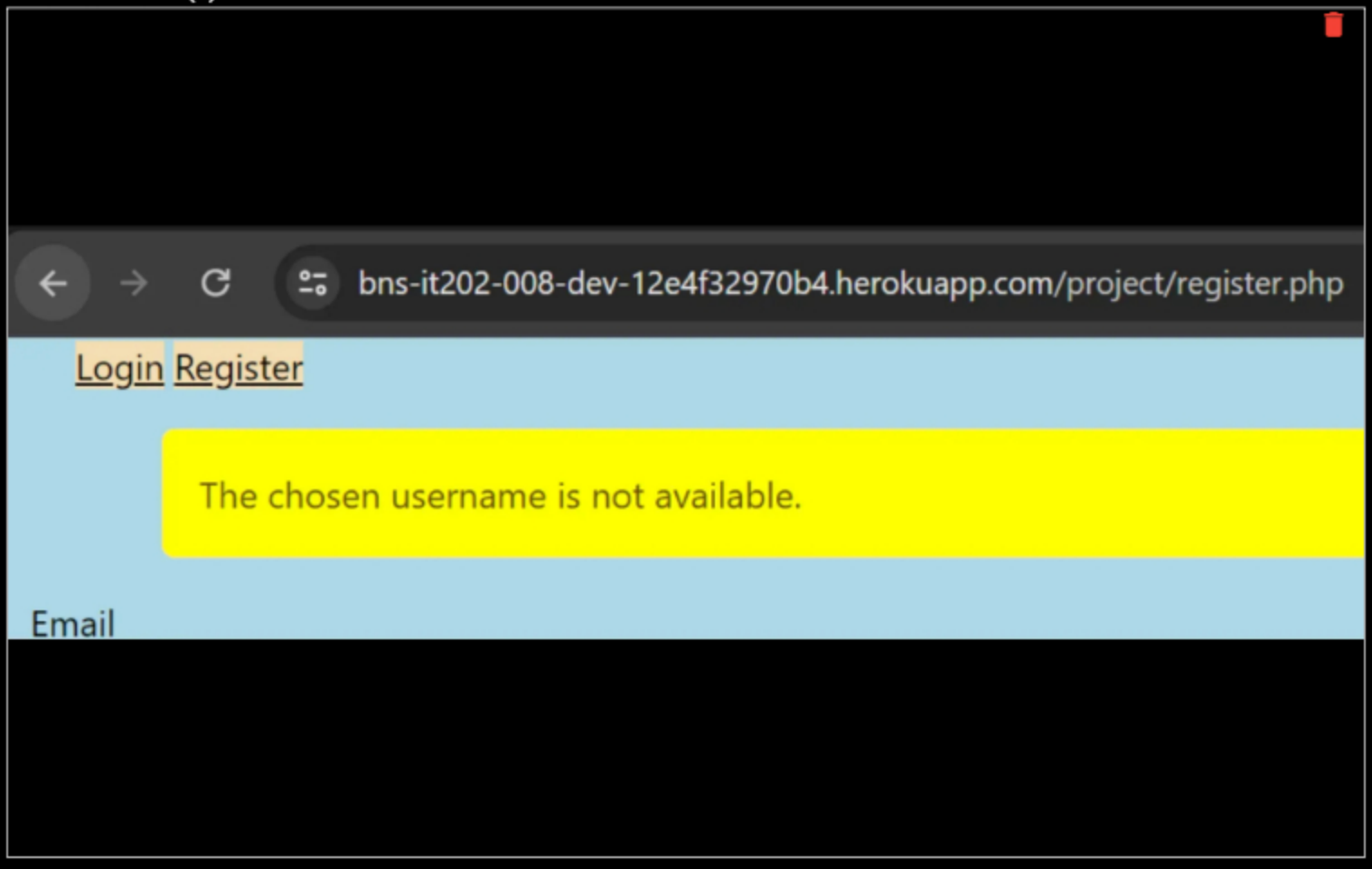
Sorry for having one of the messages repeated. Site is zoomed out to show whole thing, [JS] shows the javascript client side validation.

Checklist Items (0)



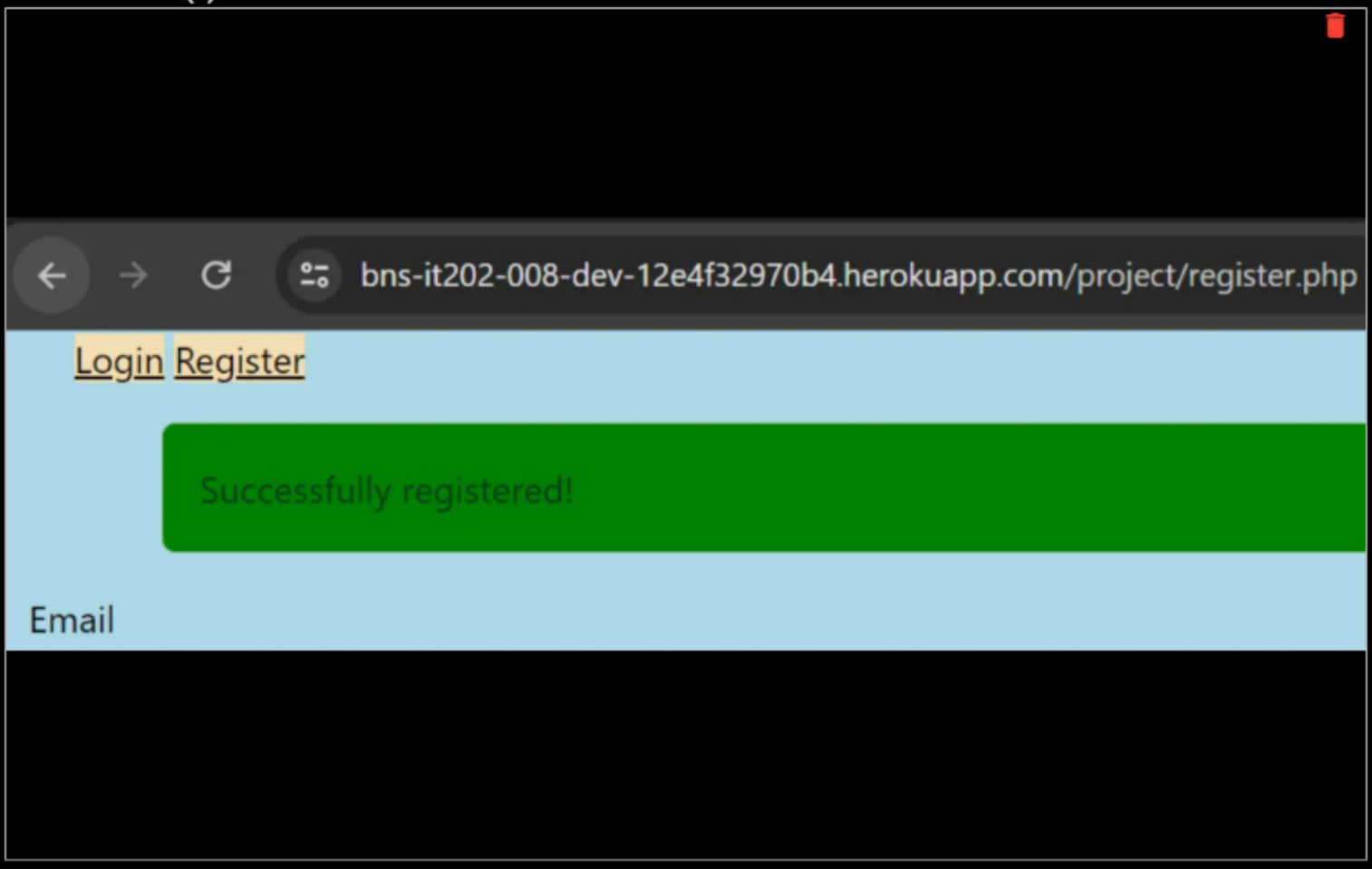
Email is not available flash message

Checklist Items (0)



Username is not available flash message

Checklist Items (0)



Account successfully made flash message

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Screenshot of the form code

Details:

Should have appropriate input types for the fiel

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
<div class="container-fluid">
    <form onsubmit="return validate(this)" method="POST">
        <div class="mb-3">
            <label for="email" class="form-label">Email</label>
            <input type="email" name="email" required class="form-control" />
        </div>
```

```
        <div class="mb-3">
            <label for="username" class="form-label">Username</label>
            <input type="text" name="username" required maxlength="30" class="form-control" />
        </div>
        <div class="mb-3">
            <label for="pw" class="form-label">Password</label>
            <input type="password" id="pw" name="password" required minlength="8" class="form-control" />
        </div>
        <div class="mb-3">
            <label for="confirm" class="form-label">Confirm</label>
            <input type="password" name="confirm" required minlength="8" class="form-control" />
        </div>
        <input type="submit" value="Register" class="btn btn-primary" />
    </form>
</div>
```

Form code with all the appropriate fields in a standard register page.

^COLLAPSE ^

Task #3 - Points: 1

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

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show the JavaScript validations (include any extra files related)
#2	1	Show the PHP validations (include any lib content)
#3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
<script>
    function validate(form) {
        //TODO 1: implement JavaScript validation
        //ensure it returns false for an error and true for success
        //bns24 April 1st, 2024
        var email = document.getElementsByName('email')[0].value;
        var password = document.getElementsByName('password')[0].value;
        var confirm = document.getElementsByName('confirm')[0].value;
        var username = document.getElementsByName('username')[0].value;

        if(email == ""){
            flash("[JS] Email cannot be empty.", "warning");
            return false;
        }

        if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)) {
            flash("[JS] Invalid email format", "warning");
            return false;
        }

        if(!/^[a-z0-9_]{3,16}$/.test(username)){
```

```
            flash("[JS] Username must only contain 3-16 characters a-z, 0-9, _, or -", "warning");
            return false;
        }

        if(password == ""){
            flash("[JS] Password cannot be empty.", "warning");
            return false;
        }
```

(SCREENSHOT 1) Client side validation

Checklist Items (0)

```
    if(confirm == ""){
        flash("[JS] Confirm password cannot be empty.", "warning");
        return false;
        //bns24 April 1st 2024
    }

    if(password.length < 8){
        flash("[JS] Password must be atleast 8 characters.", "warning");
        return false;
    }

    if(confirm !== password){
        flash("[JS] Password and Confirm Password must match.", "warning");
        return false;
    }



    return true;
} <- #27-73 function validate(form)
/script>
```

(SCREENSHOT 2) Client side validation

Checklist Items (0)

```
<?php
//TODO 2: add PHP Code
if (isset($_POST["email"]) && isset($_POST["password"]) && isset($_POST["confirm"]) && isset($_POST["username"])) {
    $email = se($_POST, "email", "", false);
    $password = se($_POST, "password", "", false);
    $confirm = se($_POST, "confirm", "", false);
    $username = se($_POST, "username", "", false);
    //TODO 3 bns24 04/01/24
    $hasError = false;
    if (empty($email)) {
        flash("Email must not be empty", "danger");
        $hasError = true;
    }
    //sanitize
    $email = sanitize_email($email);
    //validate
    if (!is_valid_email($email)) {
        flash("Invalid email address", "danger");
        $hasError = true;
    }
    if (!is_valid_username($username)) {
```

```php
        flash("Username must only contain 3-16 characters a-z, 0-9, _, or -", "danger");
        $hasError = true;
    }
    if (empty($password)) {
        flash("password must not be empty", "danger");
        $hasError = true;
    }
    if (empty($confirm)) {
        flash("Confirm password must not be empty", "danger");
        $hasError = true;
    }
```

(SCREENSHOT 1) PHP Validation

Checklist Items (0)

```php
    if (empty($confirm)) {
        flash("Confirm password must not be empty", "danger");
        $hasError = true;
    }
    if (!is_valid_password($password)) {
        flash("Password too short", "danger");
        $hasError = true;
    }
    if (
        strlen($password) > 0 && $password !== $confirm
    ) {
        flash("Passwords must match", "danger");
        $hasError = true;
    }
    if (!$hasError) {
        //TODO 4 bns24 04/01/24
        $hash = password_hash($password, PASSWORD_BCRYPT);
        $db = getDB();
        $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 $e) {
            users_check_duplicate($e->errorInfo);
        }
    } <- #117-128 if (!$hasError)
} <- #77-129 if (isset($_POST["email"]) && isset($_POST["password"]) && is...
?>
```

(SCREENSHOT 2) PHP Validation

Checklist Items (0)

```php
<?php
function users_check_duplicate($errorInfo)
{
    if ($errorInfo[1] === 1062) {
        //https://www.php.net/manual/en/function.preg-match.php
        //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 " . $matches[1] . " is not available.", "warning");
        } else {
            //TODO come up with a nice error message bns24 04/01/24
            flash("An unhandled error occured", "danger");
            //this will log the output to the terminal/console that's running the php server
            error_log(var_export($errorInfo, true));
```

```php
        } <- #10-15 else
    } else {
        //TODO come up with a nice error message
        flash("An unhandled error occured", "danger");
        //this will log the output to the terminal/console that's running the php server
        error_log(var_export($errorInfo, true));
    } <- #16-21 else
} <- #3-22 function users_check_duplicate($errorInfo)
```

(EXTRA) users_check_duplicate function used in PHP Validation

Checklist Items (0)

COLLAPSE

Task #4 - Points: 1

Text: Screenshot of the Users table with a valid user entry

Checklist

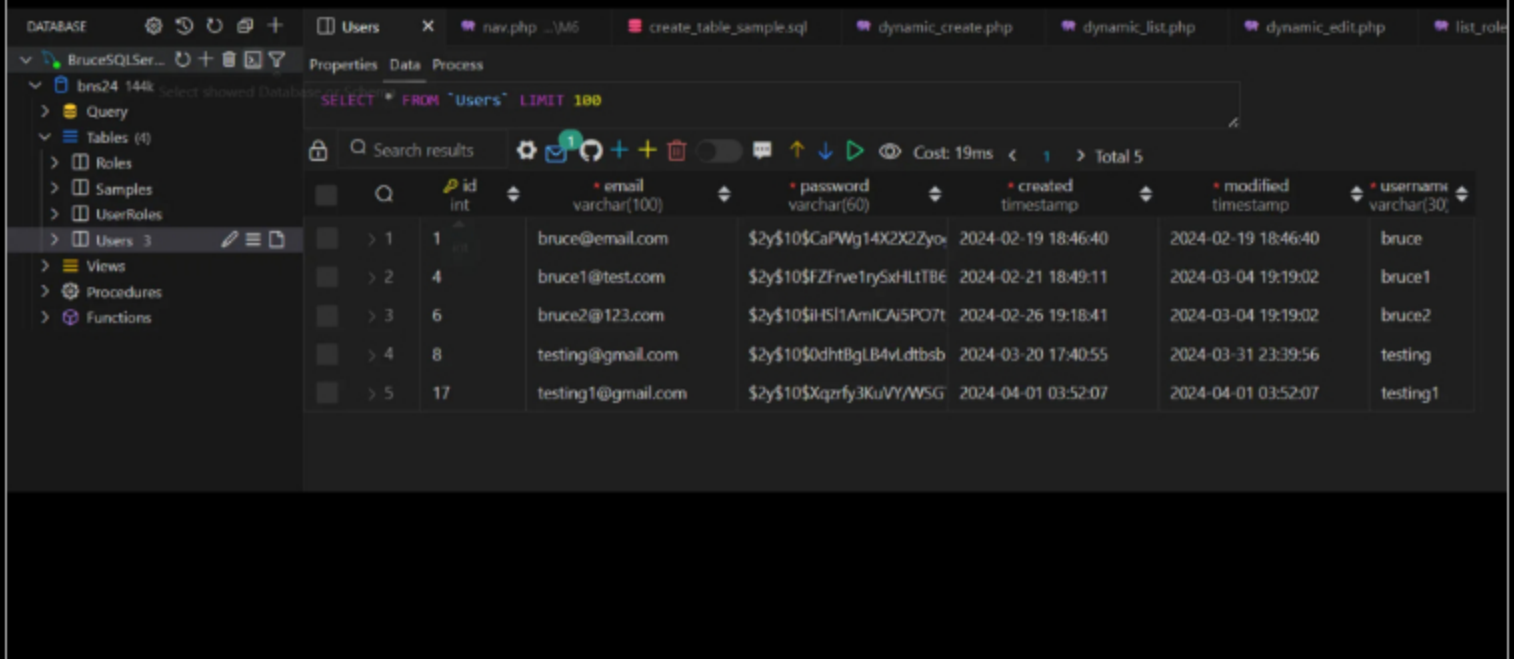
*The checkboxes are for your own tracking

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

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Users Table

Checklist Items (0)

COLLAPSE

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

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

Response:

When the page loads, it first makes sure it has the nav.php file for all the other links, and starts a new session. A form is made with 4 fields, the email, the username (with a required maxlength of 30), the password (with a required minlength of 8), and the confirm password (also with required minlength of 8). Then there is a submit button at the button, which sends data through a POST method.

When the form is submitted, it enters a validate function. First, it does client side validation, checking to see if the email is not empty, if the email is in an invalid format, if the username is in an invalid format, if the password is empty, if the confirm password field is empty, if the password length is less than 8 characters, and if the password and confirm password match. If any error occurs, it returns false and a flash message is displays saying the error, otherwise it returns true.

The code then enters server-side validation, checking at first for the same things. In addition, it prepares the database and inserts the new data into the table. If everything works well, then the user is successfully registered and the flash message displays that. If not, then the program enters the users_check_duplicates function in the lib file, where it checks to see if there is already an account with the existing username or email. If there is, there is a respective flash message displaying that.

COLLAPSE

Task #6 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/BruceSiaNJIT/bns24-it202-008/pull/13

User Login (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshot of form on website page

Checklist

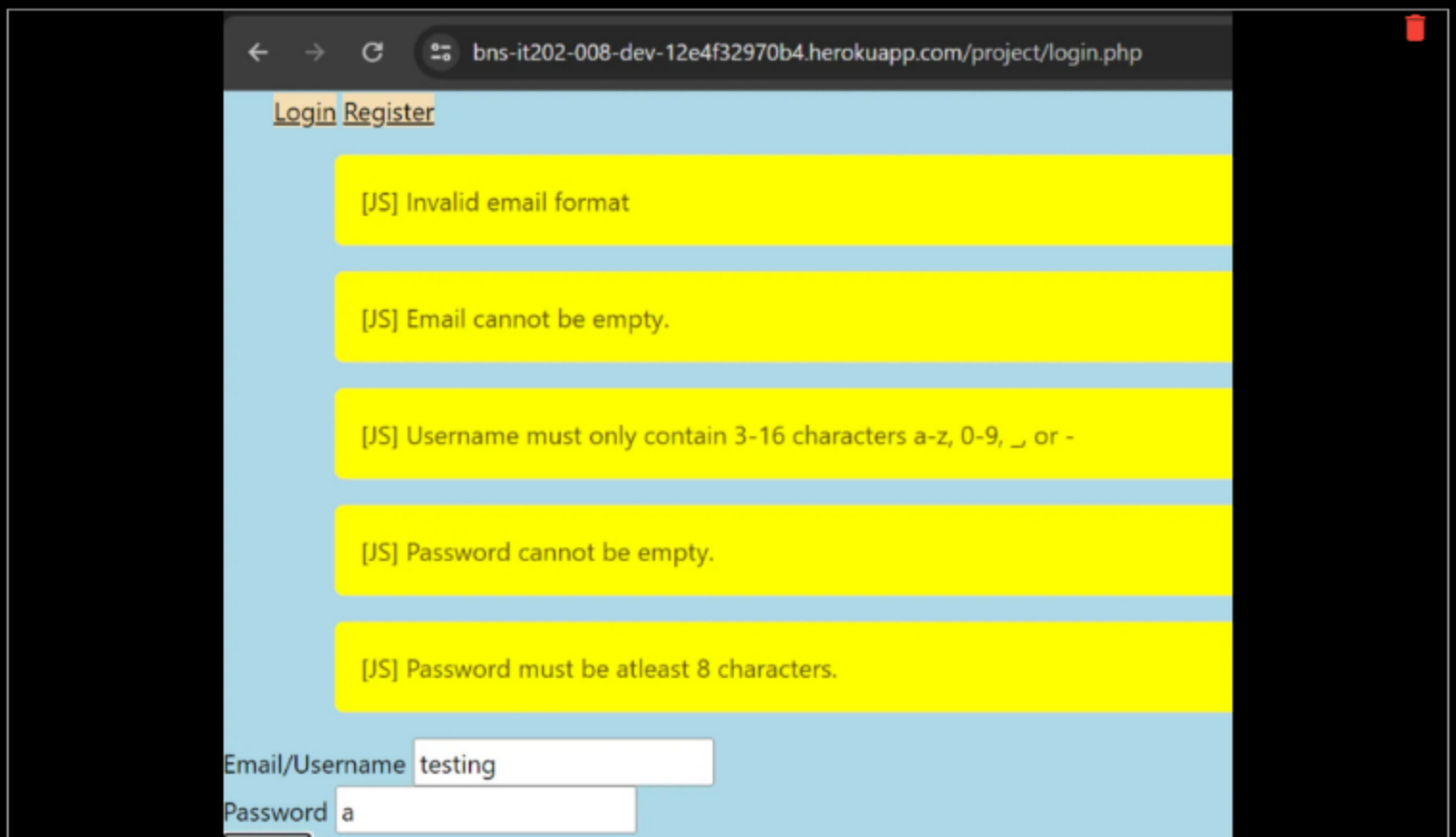
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Heroku dev url should be present in the address bar
#2	1	Should have thoughtful CSS applied
#3	1	Include correct data where username is used to login
#4	1	Include correct data where email is used to login
#5	1	Show success login message
#6	1	Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)
#7	1	Demonstrate user-friendly message of when an account doesn't exist
#8	1	Demonstrate user-friendly message of when password doesn't match what's in the DB
#9	1	Demonstrate successful login message and the destination page (i.e., home or some landing/dashboard page)
#10	1	Demonstrate session data being set (captured from server logs)

Task Screenshots:

Gallery Style: Large View

Small Medium Large



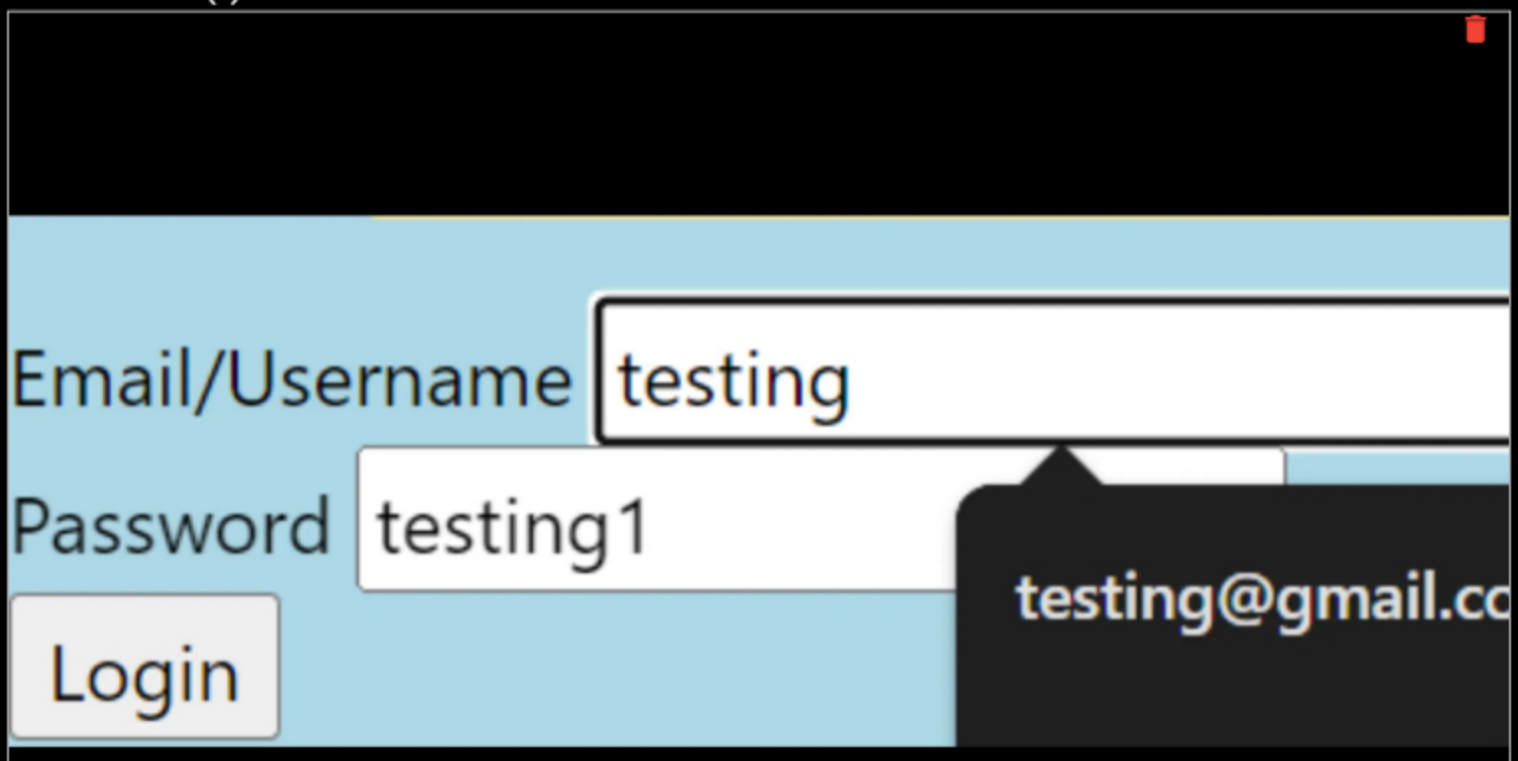
Login

Demonstrating JS Validation

Checklist Items (0)

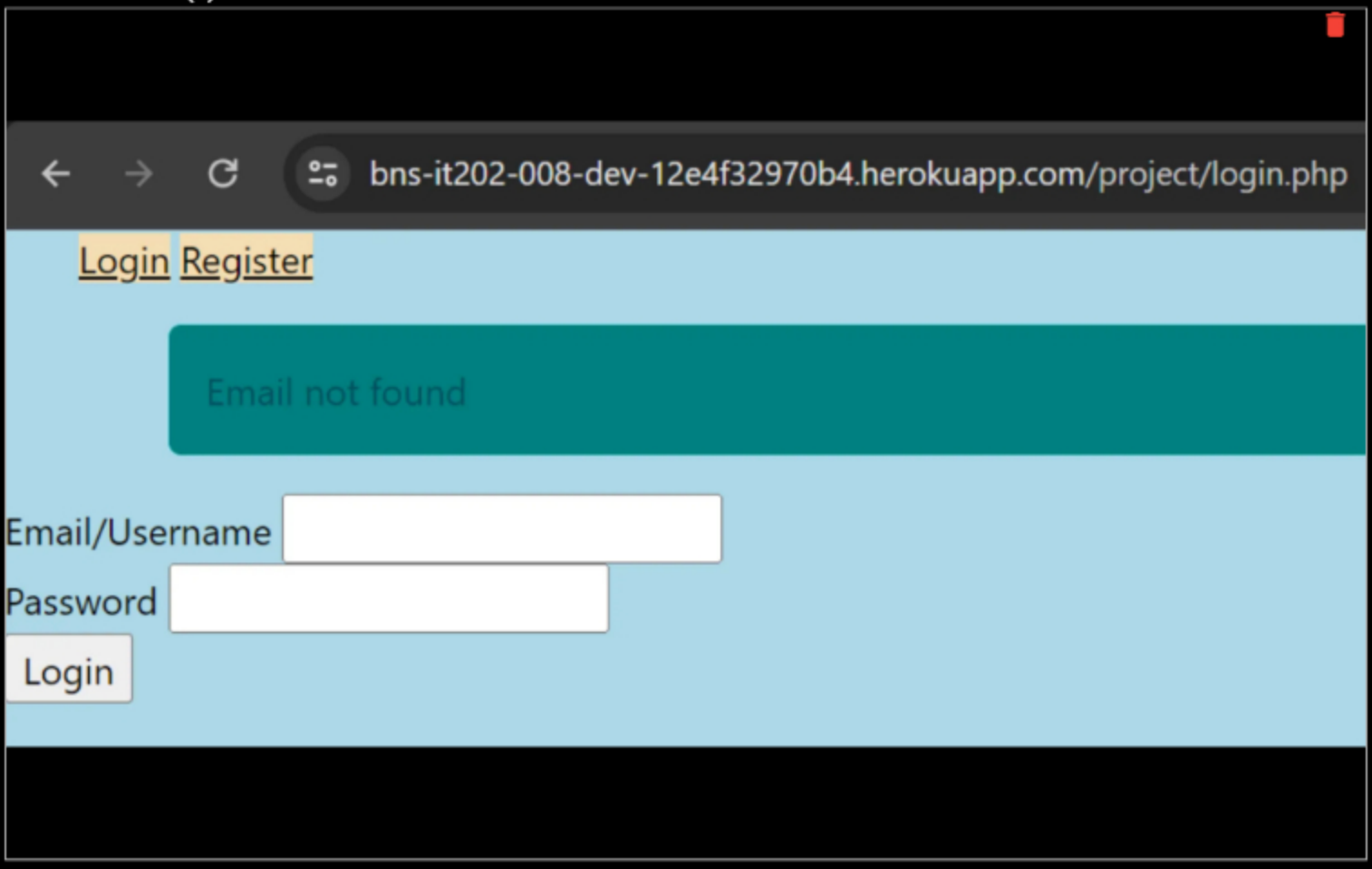
Correct data for using email to log in

Checklist Items (0)



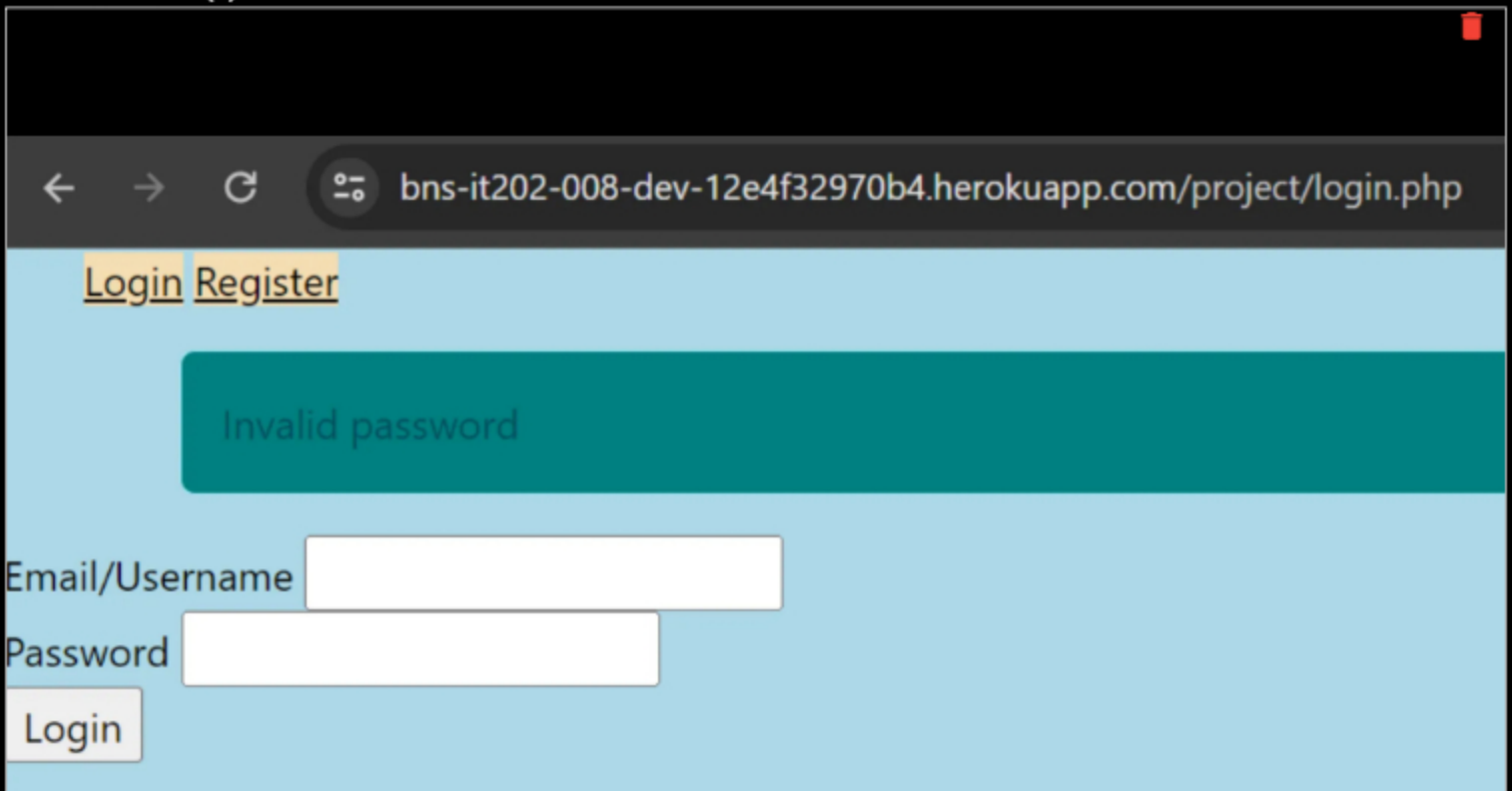
Correct data when using username to log in

Checklist Items (0)



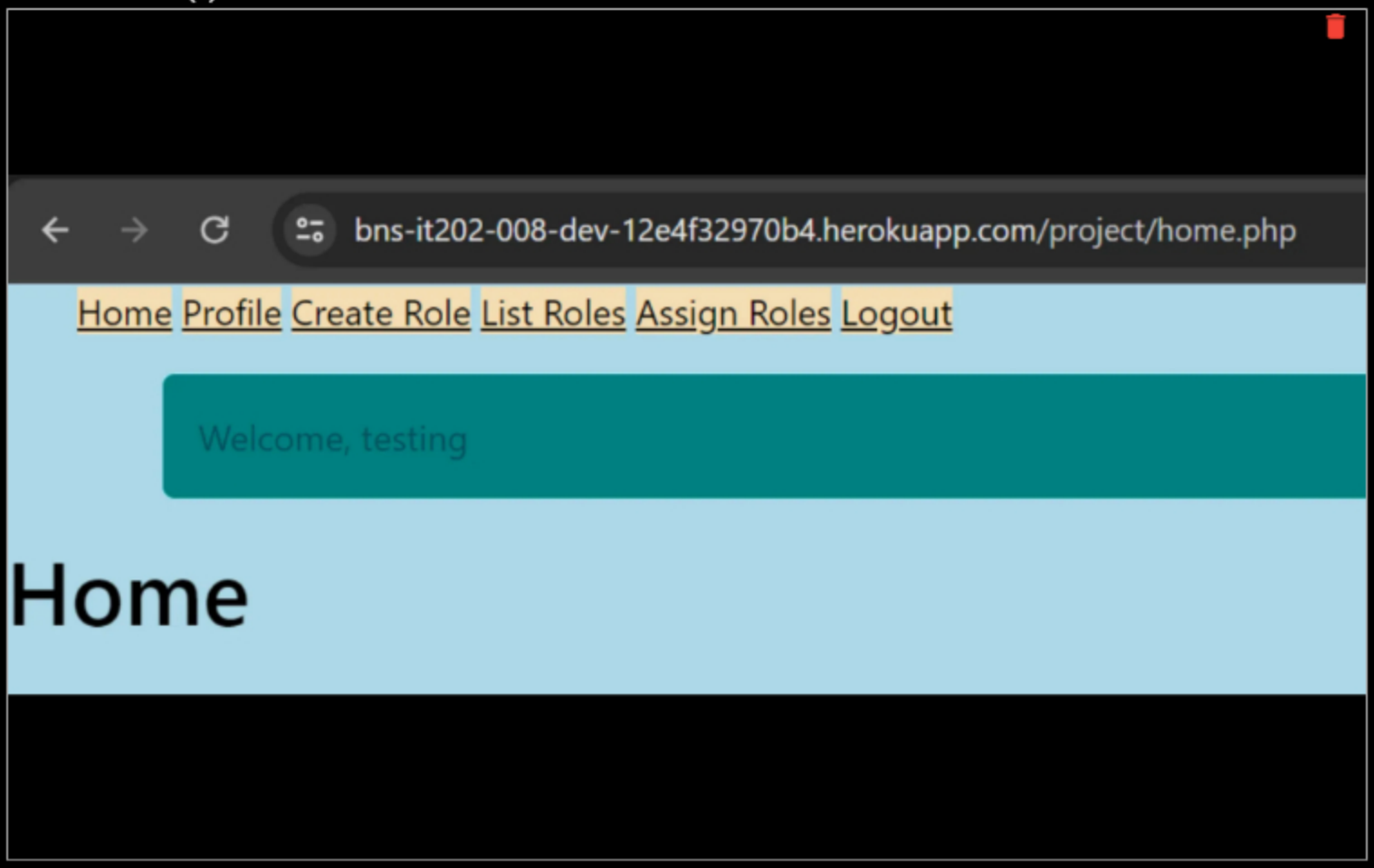
When the account is not found

Checklist Items (0)



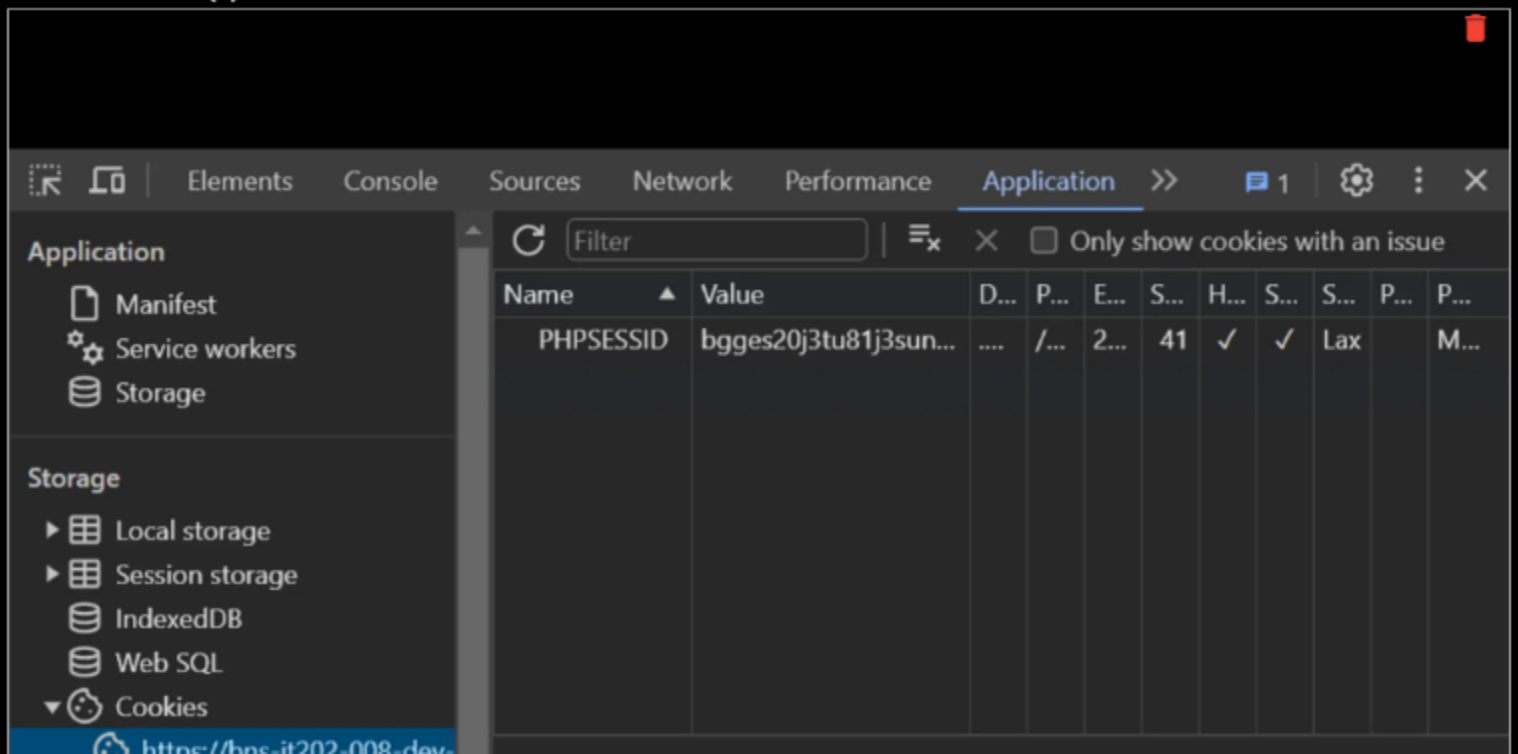
Password doesn't match

Checklist Items (0)



Success message and the destination page

Checklist Items (0)



Private state tokens

Session data being set, I think this is what I needed to submit

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Screenshot of the form code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Should have proper input types for the fields (note in the caption if you're using two fields for Username/Email or a combined field)
#2	1	Show JavaScript validations (include any extra files related)
#3	1	Show PHP validations (include any lib content)
#4	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```
<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>
```

Proper input types for the login page. Username and email are combined.

Checklist Items (0)

```
<script>
    function validate(form) {
        //TODO 1: implement JavaScript validation
        //ensure it returns false for an error and true for success bns24 04/01/24
        var email = document.getElementsByName('email')[0].value;
        var password = document.getElementsByName('password')[0].value;

        if(email == ""){
            flash("[JS] Email cannot be empty.", "warning");
            return false;
        }

        if(email.includes("@")){
            if(!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)){
                flash("[JS] Invalid email format", "warning");
                return false;
            }
        }else{
            if(!/^[a-z0-9_-]{3,16}$/.test(email)){
                flash("[JS] Username must only contain 3-16 characters a-z, 0-9, _, or -", "warning");
                return false;
            }
        } <- #32-37 else

        if(password == ""){
            flash("[JS] Password cannot be empty.", "warning");
            return false;
        }
```

(SCREENSHOT 1) Javascript Validation

Checklist Items (0)

```
        if(password.length < 8){
            flash("[JS] Password must be atleast 8 characters.", "warning");
            return false;
        }

        //TODO update clientside validation to check if it should
        //valid email or username bns24 04/01/24
        return true;
    } <- #16-52 function validate(form)
</script>
```

(SCREENSHOT 2) Javascript Validation

Checklist Items (0)

```php
<?php
//TODO 2: add PHP Code
if (isset($_POST["email"]) && isset($_POST["password"])) {
    $email = se($_POST, "email", "", false);
    $password = se($_POST, "password", "", false);

    //TODO 3
    $hasError = false;
    if (empty($email)) {
        flash("Email must not be empty");
        $hasError = true;
    }
    if (str_contains($email, "@")) {
        //sanitize bns24 04/01/24
        //$email = filter_var($email, FILTER_SANITIZE_EMAIL);
        $email = sanitize_email($email);
        //validate
        /*if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            flash("Invalid email address");
            $hasError = true;
        }*/
        if (!is_valid_email($email)) {
            flash("Invalid email address");
            $hasError = true;
        }
    } else {
        if (!is_valid_username($email)) {
            flash("Invalid username");
            $hasError = true;
        }
    } <- #79-84 else
```

(SCREENSHOT 1) PHP Validation

Checklist Items (0)

```php
if (empty($password)) {
    flash("password must not be empty");
    $hasError = true;
}
if (!is_valid_password($password)) {
    flash("Password too short");
    $hasError = true;
}
if (!$hasError) {
    //flash("Welcome, $email");
    //TODO 4 bns24 04/01/24
    $db = getDB();
    $stmt = $db->prepare("SELECT id, email, username, password from Users
    where email = :email or username = :email");
    try {
        $r = $stmt->execute([":email" => $email]);
        if ($r) {
            $user = $stmt->fetch(PDO::FETCH_ASSOC);
            if ($user) {
                $hash = $user["password"];
                unset($user["password"]);
                if (password_verify($password, $hash)) {
                    //flash("Weclome $email");
                    $_SESSION["user"] = $user; //sets our session data from db
                    //lookup potential roles
                    $stmt = $db->prepare("SELECT Roles.name FROM Roles
                    JOIN UserRoles on Roles.id = UserRoles.role_id
                    where UserRoles.user_id = :user_id and Roles.is_active = 1 and UserRoles.is_active = 1");
                    $stmt->execute([":user_id" => $user["id"]]);
                    $roles = $stmt->fetchAll(PDO::FETCH_ASSOC); //fetch all since we'll want multiple
```

```
//save roles or empty array
if ($roles) {
```

(SCREENSHOT 2) PHP Validation

Checklist Items (0)

```
                    if ($roles) {
                        $_SESSION["user"]["roles"] = $roles; //at least 1 role
                    } else {
                        $_SESSION["user"]["roles"] = []; //no roles bns24 04/01/24
                    }
                    flash("Welcome, " . get_username());
                    die(header("Location: home.php"));
                } else {
                    flash("Invalid password");
                }
            } else {
                flash("Email not found");
            }
        } <- #103-129 if ($user)
    } catch (Exception $e) {
        flash("<pre>" . var_export($e, true) . "</pre>");
    }
} <- #99-133 try
} <- #97-134 $stmt = $db->prepare
```

(SCREENSHOT 3) PHP Validation

Checklist Items (0)

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

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Don't just show code, translate things to plain English
#2	1	Explain how the session works and why/how it's used

Response:

First the code requires the nav file for the navigation layout. It has a form with two fields, an email/username and the password. Both are required, password must be minimum 8 characters. When the information is submitted through a POST method, it goes through a validation function.

First it does client-side validation, where it checks if email field is empty. Then it checks if the email field has an "@". If it does, then that means that what is being typed is most likely an email, so we check to see if email is formatted

correctly. If not, a message is flashed and it returns false. If there is no "@", then it checks for username format, and if it is incorrect then a flash message appears and it returns false. It also checks if password is empty, and if password is less than 8 characters, and flashes messages respectively if there is an error.

Then it goes through server-side validation, where it goes through similar checks. What's different is that it prepares the database and first sees if the username/email exists in the database. If it does, then we check the hashed version of the password in the database with the hash version of what was typed, and if it matches then it was a success. We start a session for the user so that they can stay logged in for a certain time period. We assign them their respective roles and bring them to home.php and flash a welcome message for a success. If the password doesn't match, then we flash a message saying the password was not found. If there wasn't even an email/username in the database, then we flash an appropriate message.

COLLAPSE

Task #4 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/BruceSiaNJIT/bns24-it202-008/pull/14

User Logout (1 pt.)

COLLAPSE

Task #1 - Points: 1

Text: Capture the following screenshots

Checklist

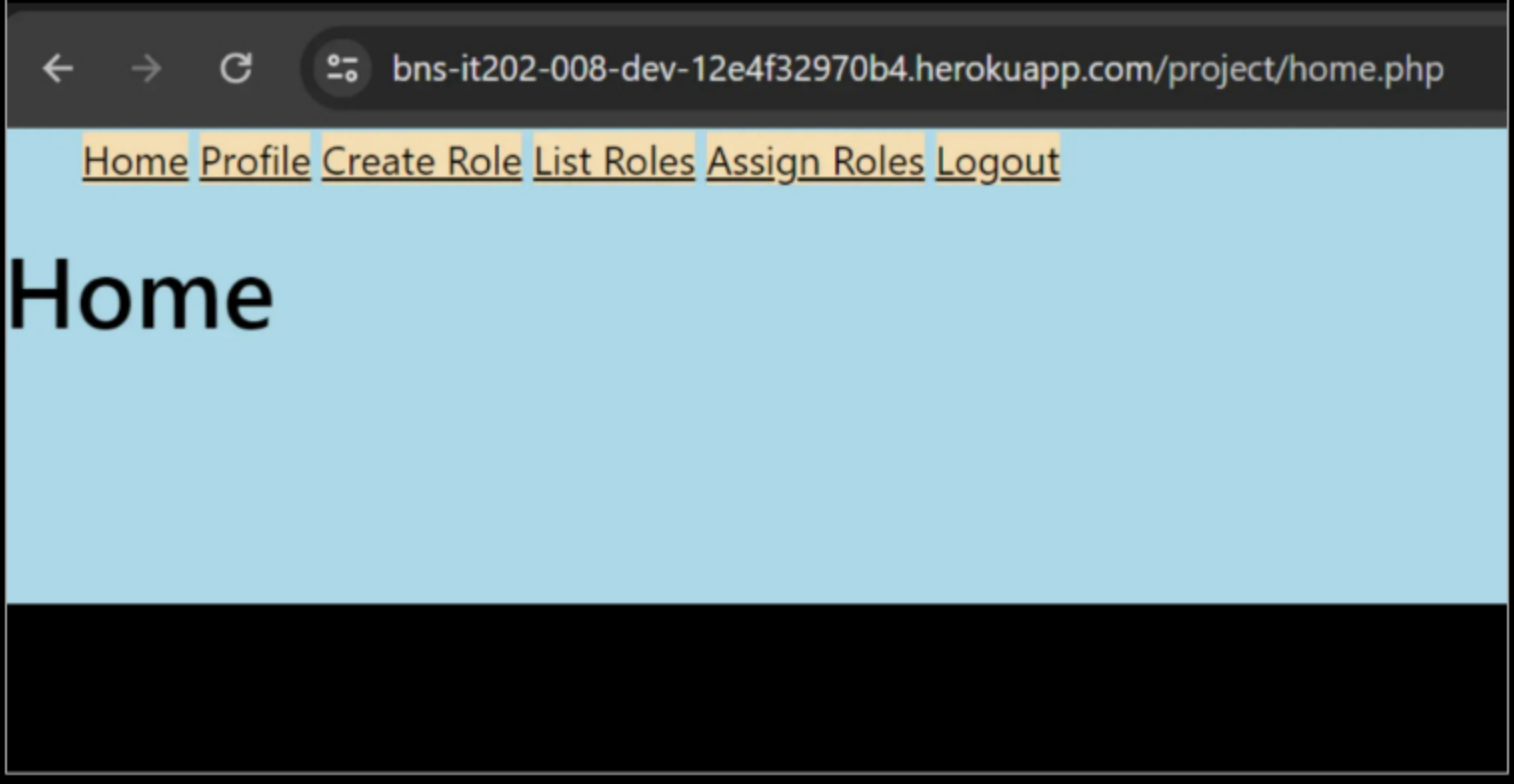
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Screenshot of the navigation when logged in (site)
#2	1	Screenshot of the redirect to login with the user-friendly logged-out message (site)
#3	1	Screenshot of the logout-related code showing the session is destroyed (code). Ensure ucid/date comment is present.

Task Screenshots:

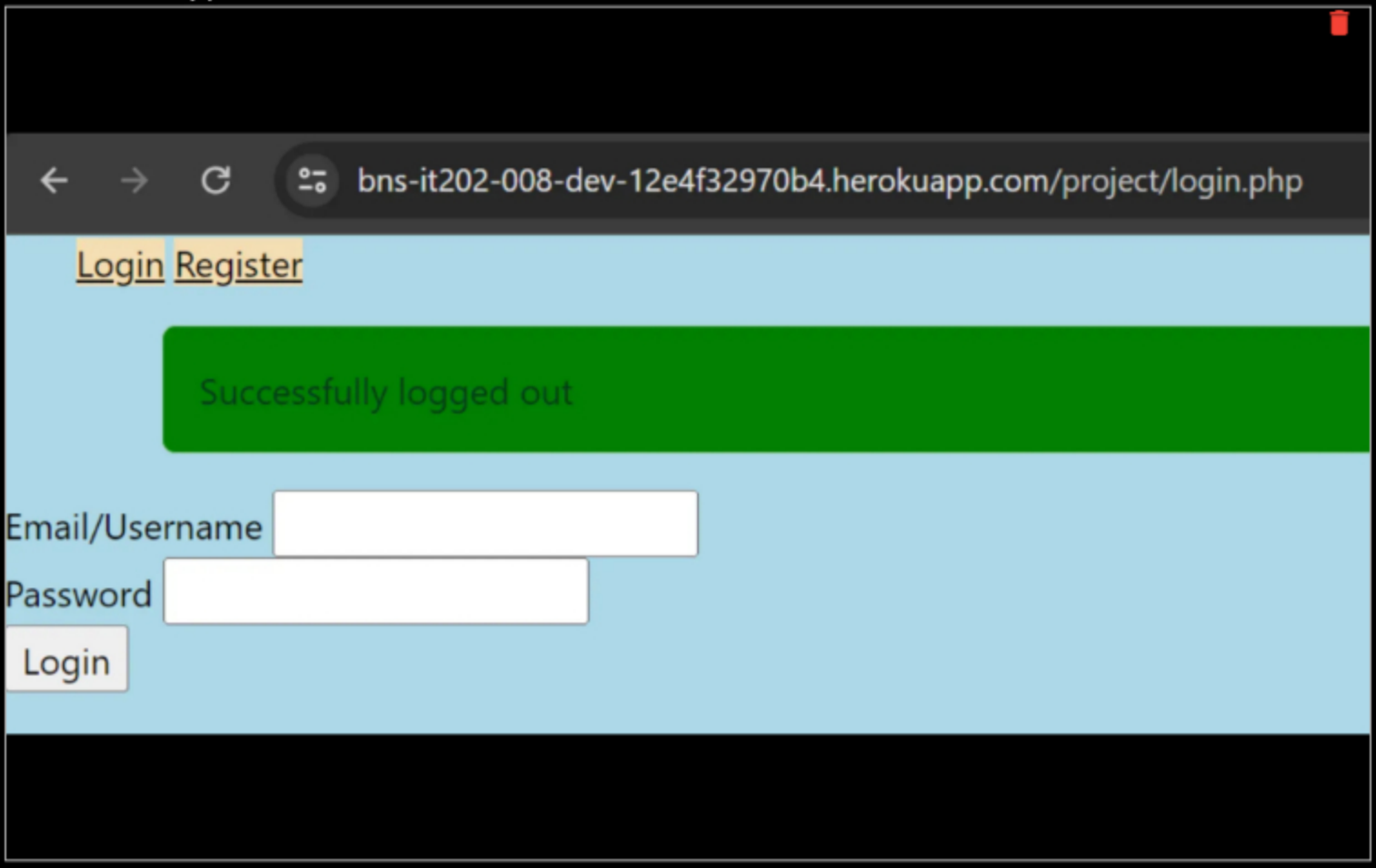
Gallery Style: Large View

Small Medium Large



User is logged in

Checklist Items (0)



User is logged out and is brought to the login page with a flash message.

Checklist Items (0)

public_html > project > logout.php

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

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

//bns24 04/01/24
```

Code

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/BruceSiaNJIT/bns24-it202-008/pull/14

Basic Security Rules and Roles (2 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Authentication Screenshots

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Screenshot of the function that checks if a user is logged in
		Screenshot of the login check function being used (i.e., profile likely). Also caption what pages

#2	1	it's used on
#3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)
#4	1	Demonstrate the user-friendly message of trying to manually access a login-protected page while being logged out

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```php
function is_logged_in($redirect = false, $destination = "login.php")
{
    $isLoggedIn = isset($_SESSION["user"]);
    if ($redirect && !$isLoggedIn) {
        //if this triggers, the calling script won't receive a reply since die()/exit() terminates it
        //bns24 04/01/24
        flash("You must be logged in to view this page", "warning");
        die(header("Location: $destination"));
    }
    return $isLoggedIn;
}
```

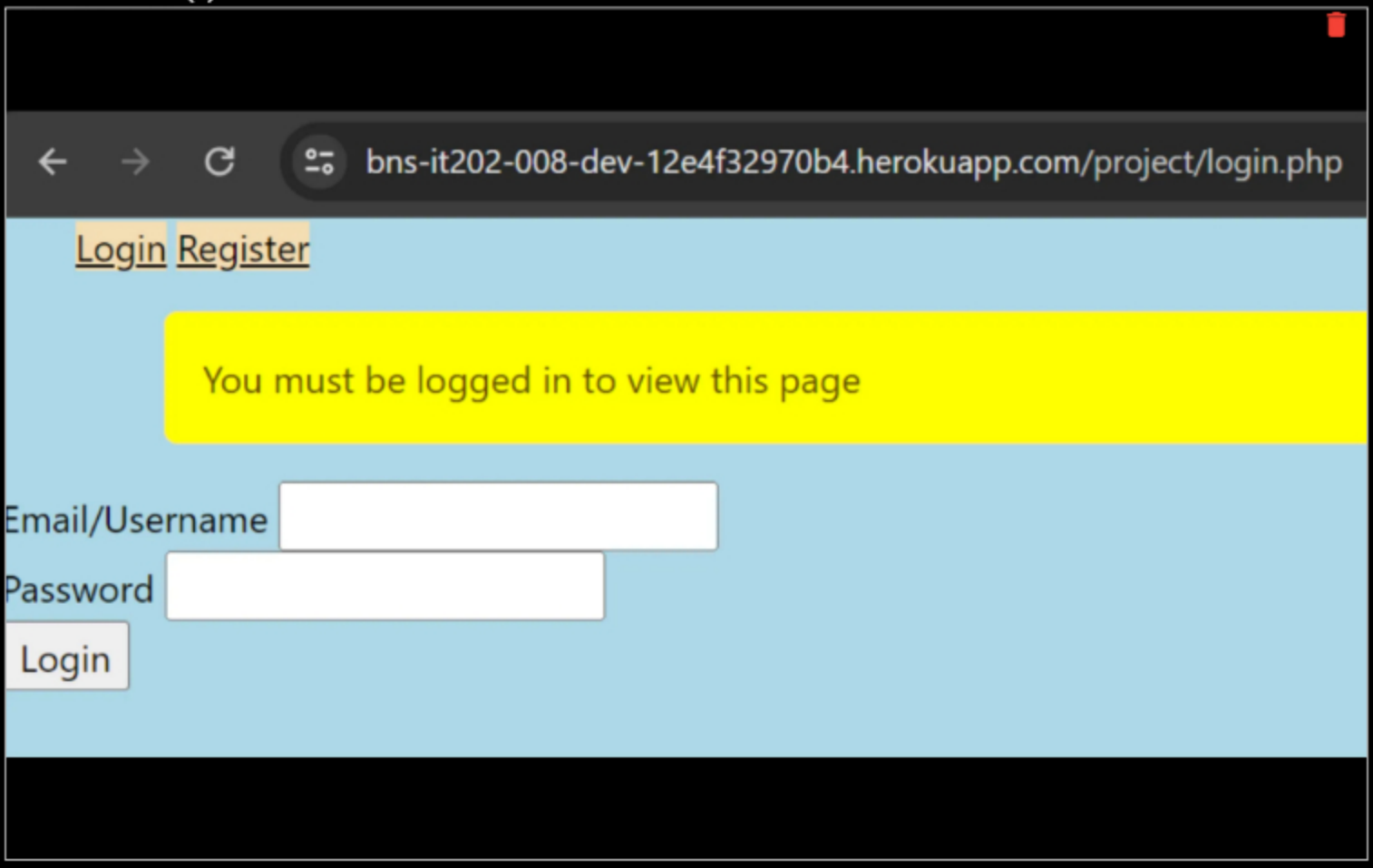
Function that checks if user is logged in

Checklist Items (0)

```php
<?php
require_once(__DIR__ . "/../../partials/nav.php");
is_logged_in(true);
//bns24 04/01/24
?>
```

Is logged in check being used

Checklist Items (0)



User friendly message when trying to access home.php while logged out

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Authorization Screenshots

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Screenshot of the function that checks for a specific role
#2	1	Screenshot of the role check function being used. Also caption what pages it's used on
#3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)
#4	1	Demonstrate the user-friendly message of trying to manually access a role-protected page

#4	while being logged out

Task Screenshots:

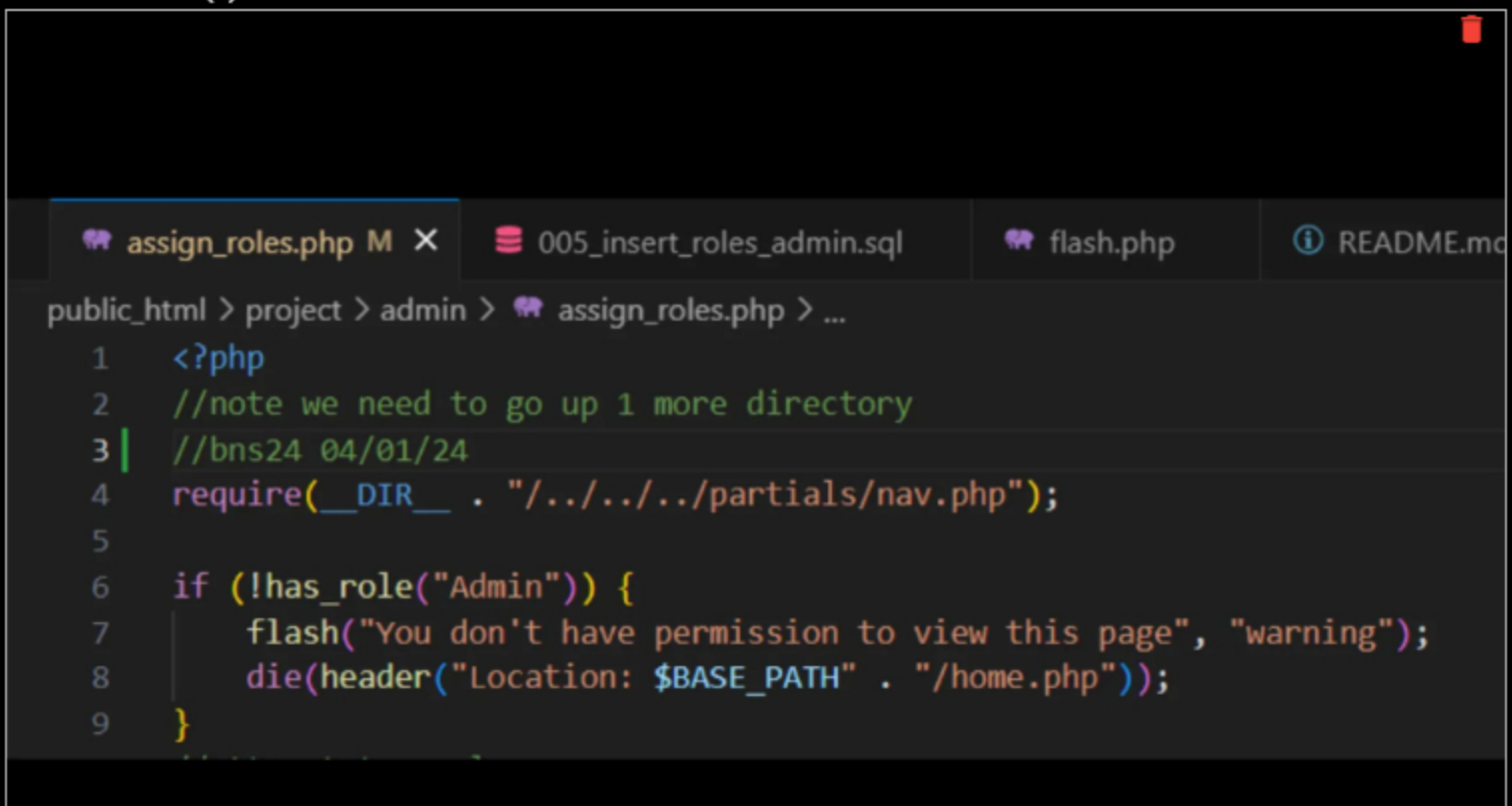
Gallery Style: Large View

Small Medium Large

```php
function has_role($role)
{
    if (is_logged_in() && isset($_SESSION["user"]["roles"])) {
        foreach ($_SESSION["user"]["roles"] as $r) {
            if ($r["name"] === $role) {
                return true;
                //bns24 04/01/24
            }
        } <- #21-25 foreach ($_SESSION["user"]["roles"] as $r)
    } <- #20-26 if (is_logged_in() && isset($_SESSION["user"]["ro
    return false;
} <- #19-28 function has_role($role)
```

has_role function

Checklist Items (0)



Role check being used in assign_roles.php

Checklist Items (0)

create_role.php M | login.php | Users | nav.php ...\M6 | create_t

public_html > project > admin > create_role.php > ...

```php
<?php
//note we need to go up 1 more directory
//bns24 04/01/24
require(__DIR__ . "/../../../partials/nav.php");

if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: " . get_url("home.php")));
}

```

role check being used in create_role.php

Checklist Items (0)

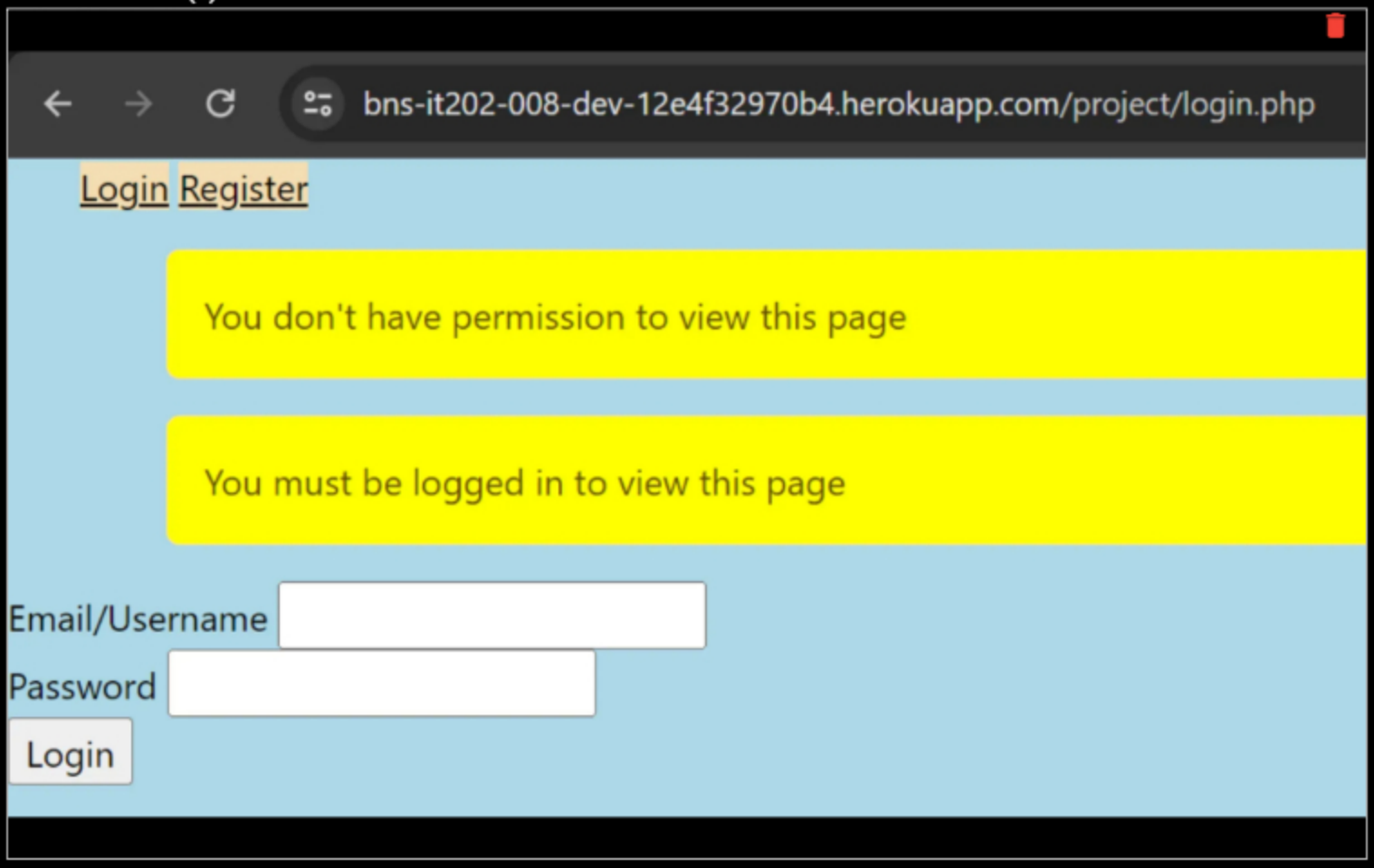
public_html > project > admin > list_roles.php > ...

```php
<?php
//note we need to go up 1 more directory
//bns24 04/01/24
require(__DIR__ . "/../../../partials/nav.php");

if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: " . get_url("home.php")));
}
```

role check being used in list_roles.php

Checklist Items (0)



User-friendly message when trying to access role-protected page while being logged out

Checklist Items (0)

COLLAPSE

Task #3 - Points: 1

Text: Screenshots of UserRoles and Roles Tables

Checklist *The checkboxes are for your own tracking

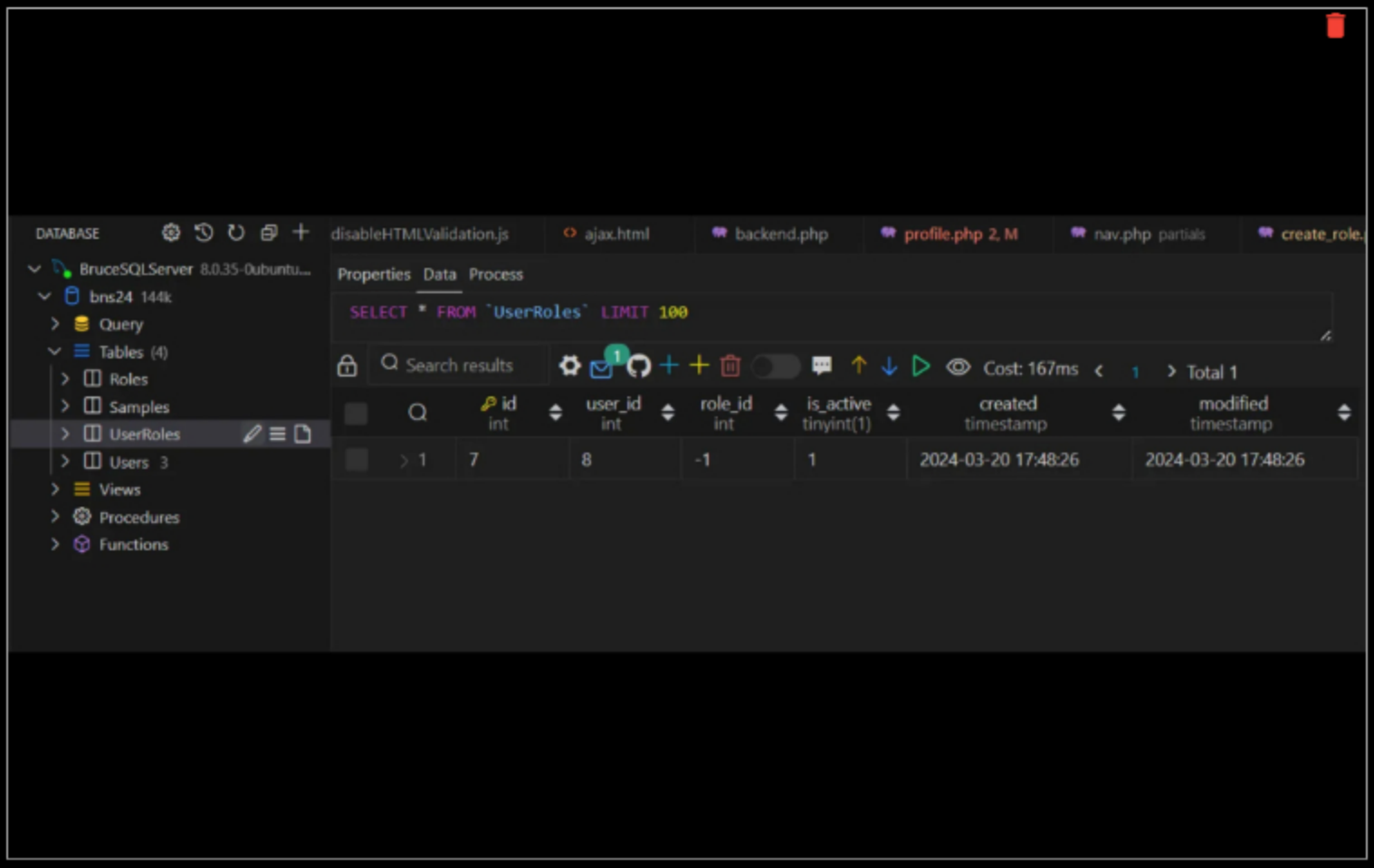
#	Points	Details
#1	1	At least one valid and enabled User->Role reference (UserRoles table)
#2	1	UserRoles Table should have id, user_id, role_id, is_active, created, and modified columns
#3	1	Roles Table should have id, name, description, is_active, modified, and created columns
#4	1	At least one valid and enabled Role (Roles table)
#5	1	Ensure left panel or database name is present in each table screenshot (should contain your ucid)

Task Screenshots:

Task Screenshots:

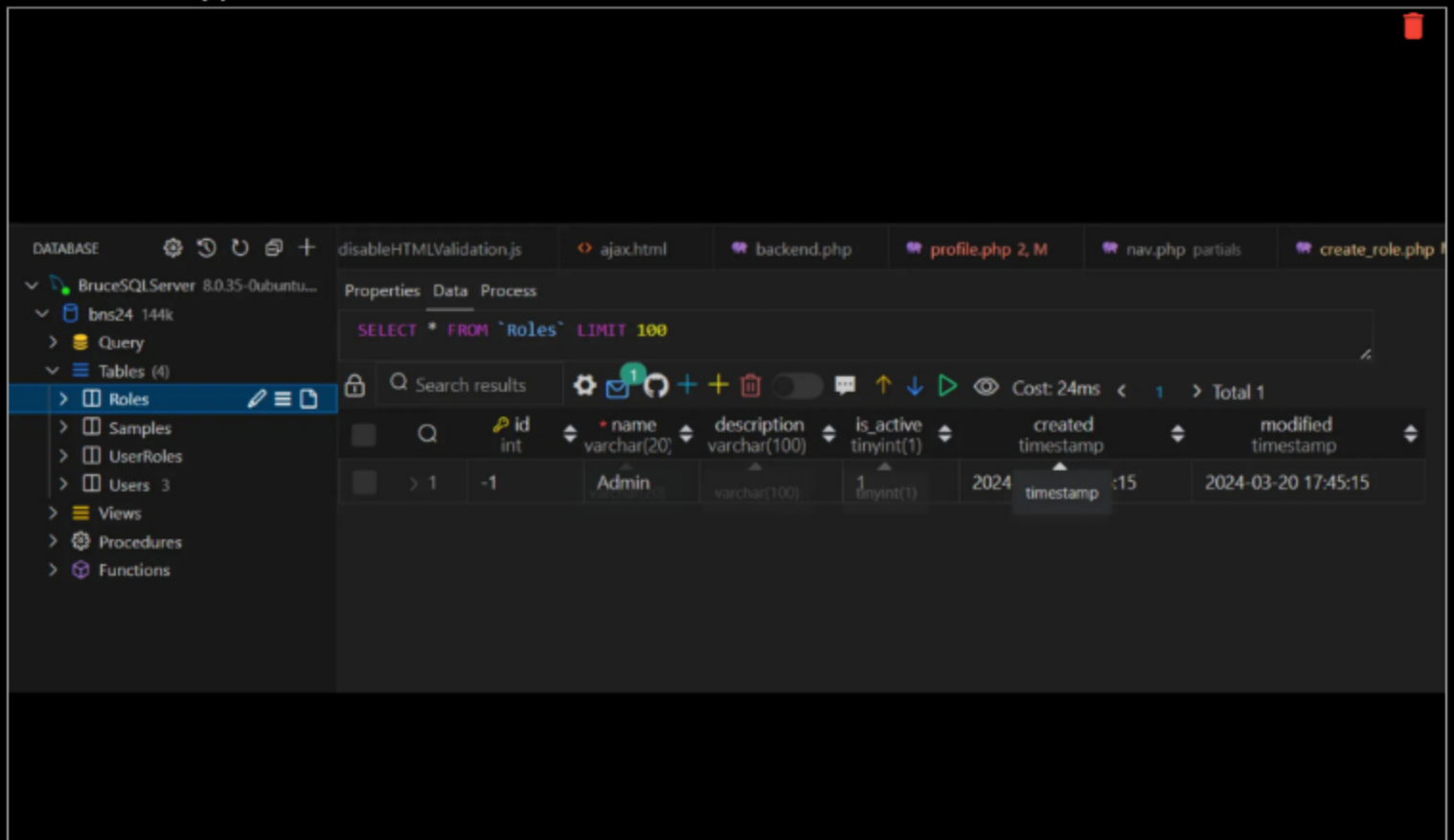
Gallery Style: Large View

Small Medium Large



UserRoles Table

Checklist Items (0)



Roles Table

Checklist Items (0)

COLLAPSE

Task #4 - Points: 1

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

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	What's the purpose of the UserRoles table?
#2	1	How does Roles.is_active differ from UserRoles.is_active?

Response:

If the Roles and UserRoles tables have not been created, they are going to be created. The Roles table has properties of a primary key id, name, description, is_active, created, modified. The UserRoles table is what helps connect the Roles table and the Users table. It has a foreign key for the user_id and the role_id.

We created a role called "Admin" in a different sql script and gave it a role_id of -1. In my case with my database, I manually gave admin to the user with id 8 through MySQL. This gives the UserRoles an ID of 7 because we combine the two. So Roles.is_active is basically in charge of making sure that the roles work and are there. The Roles table in general is just a big table that lists all possible roles that a user can have. UserRoles.is_active tells whether or not a User has an active roll because it combines the user_id with the role_id to give a user an active role. And we can disable this role for such a user.

COLLAPSE

Task #5 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/BruceSiaNJIT/bns24-it202-008/pull/19

User Profile (2 pts.)

COLLAPSE

Task #1 - Points: 1

Text: View Profile Website Page

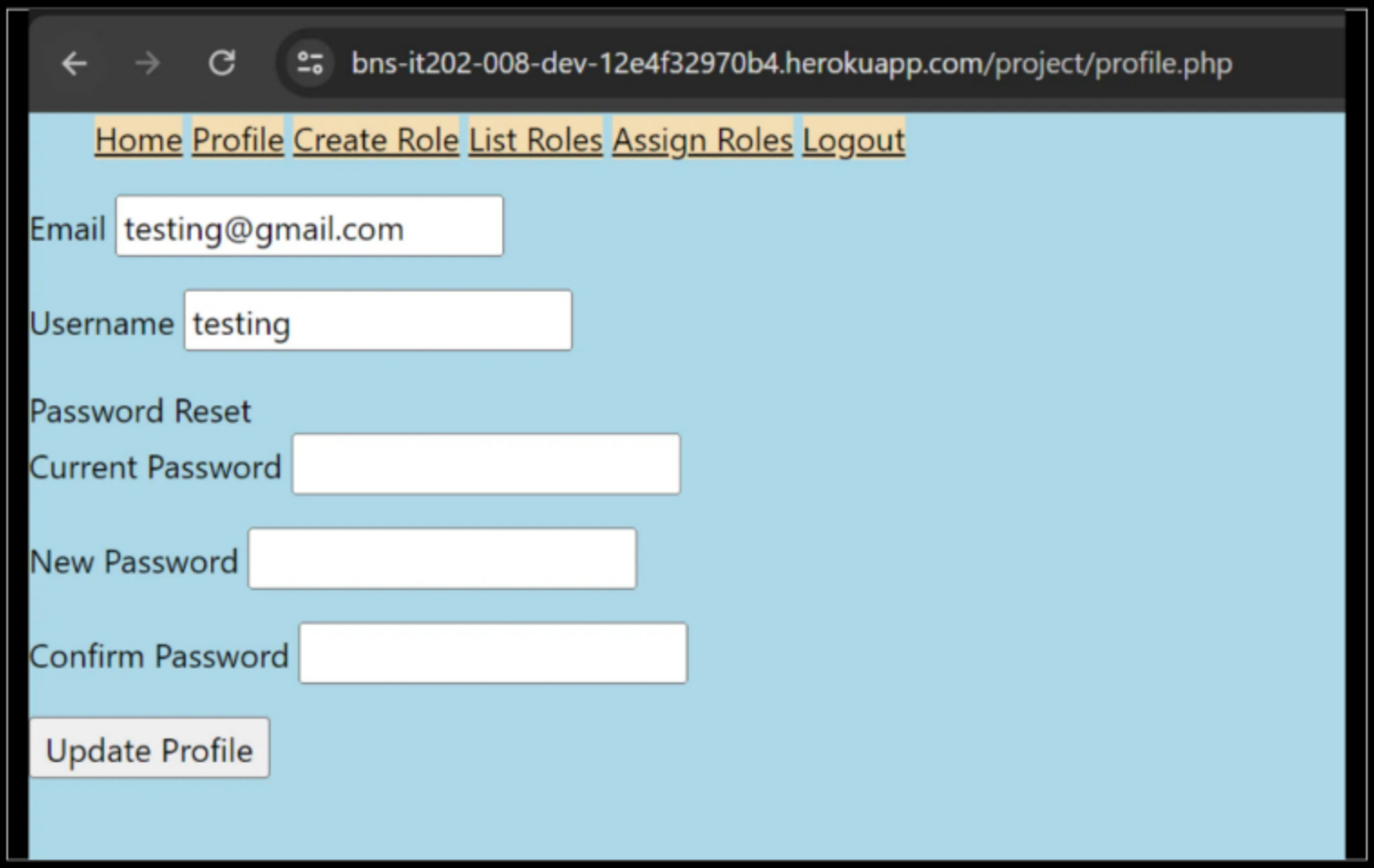
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Heroku dev url should be present in the address bar
#2	1	Should have thoughtful CSS applied
#3	1	Show the profile form correctly populated on page load (username, email)
#4	1	Should have the following fields: username, email, current password, new password, confirm password (or similar)

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Has Create Role, List Roles, and Assign Roles because this is an admin account.

Checklist Items (0)

COLLAPSE

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

Details:

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

Response:

When profile page is visited, it first checks if you are logged in. It then displays the form, with fields with the email, which displays the $email value using the get_user_email() helper function when it loads. It has the username field, which displays the $username value using the get_username() helper function when it loads. It also has a current password field, new password field, and confirm new password field.

When this form is submitted it goes through validation, client-side first. It checks if the email is formatted wrong in case the user wants to change it, then it checks if new username is formatted wrong in case user wants to change it. It then checks to see if the new password is atleast 8 characters long. It then checks to see if the new password and confirm new password values match.

Then it goes through server-side validation. It checks the same fields as client-side. Then it prepares the database and if there are no errors, it saves the new profile with the new info, otherwise it checks to see if user already exists, and if they do, then a flash message error is displayed. It also checks to make sure that the current password entered in the field is the same as the current password in the database, and if it isn't, then a flash message is displayed.

COLLAPSE

Task #3 - Points: 1

Text: Edit Profile Website Page

Checklist

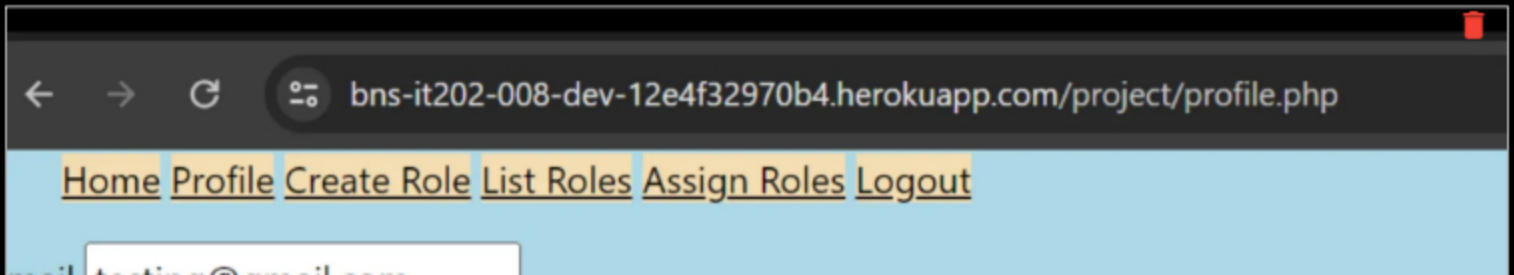
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Heroku dev url should be present in the address bar
#2	1	Should have thoughtful CSS applied
#3	1	Demonstrate with before and after of a username change (including success message)
#4	1	Demonstrate with a before and after of an email change (including success message)
#5	1	Demonstrate the success message of updating password
#6	1	Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)
#7	1	Demonstrate PHP user-friendly validation messages (desired email is already in use, desired username is already in use, current password doesn't match what's in the DB)

Task Screenshots:

Gallery Style: Large View

Small Medium Large



mail testing@gmail.com

Username testing

assword Reset

urrent Password

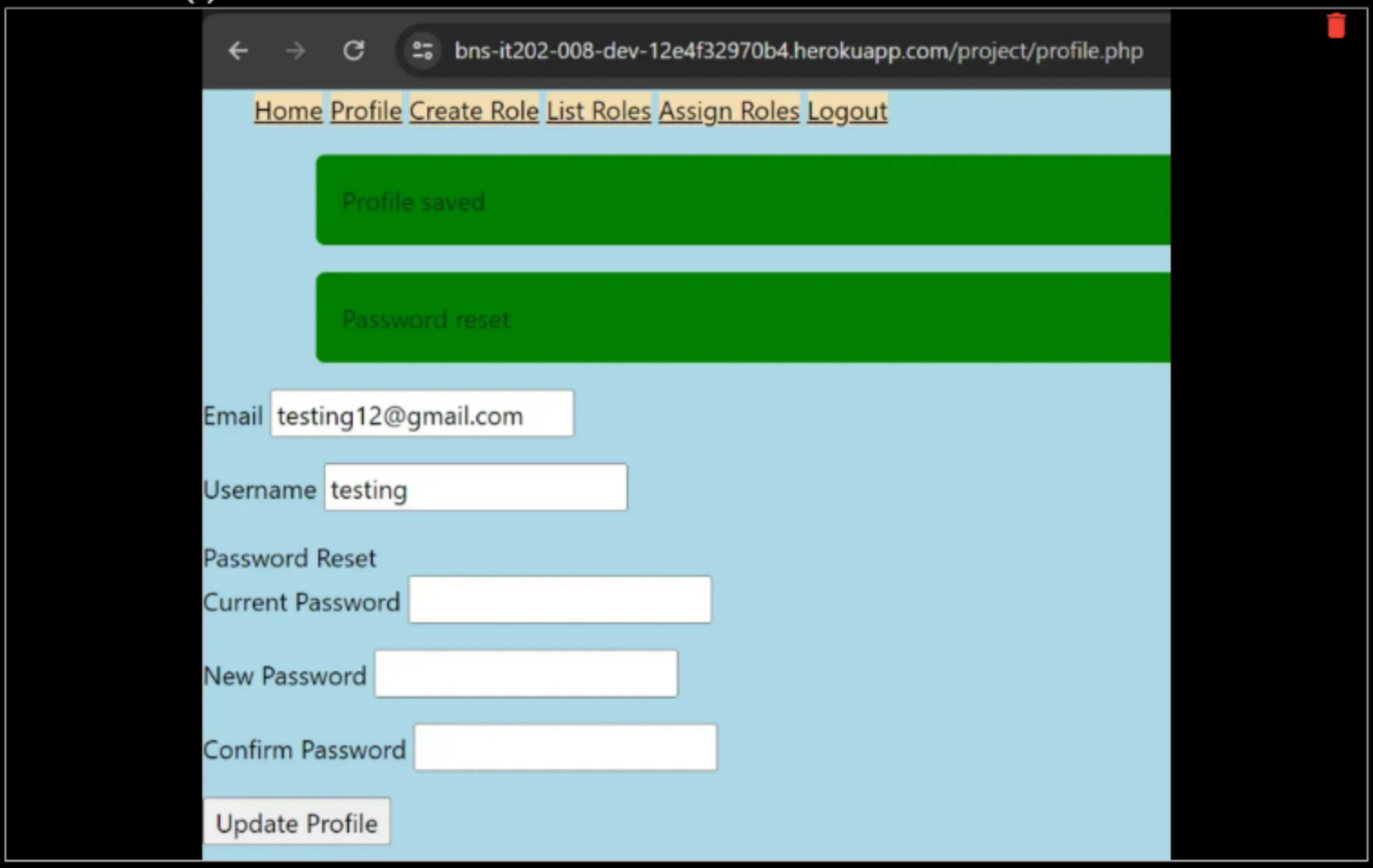
lew Password

onfirm Password

Update Profile

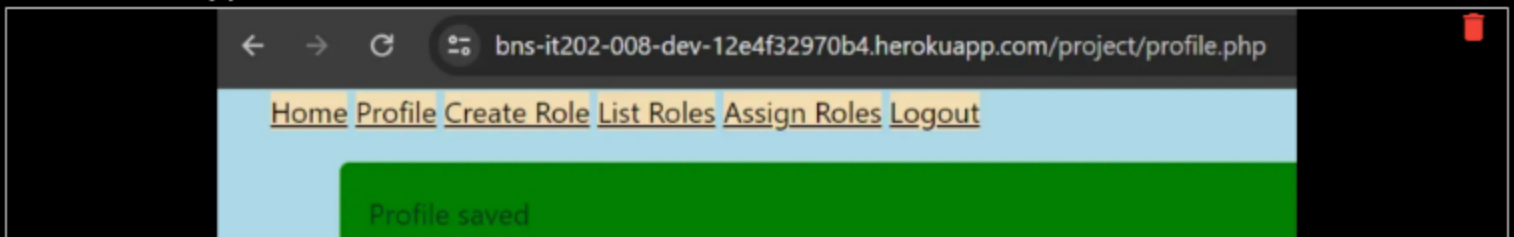
Before changing email

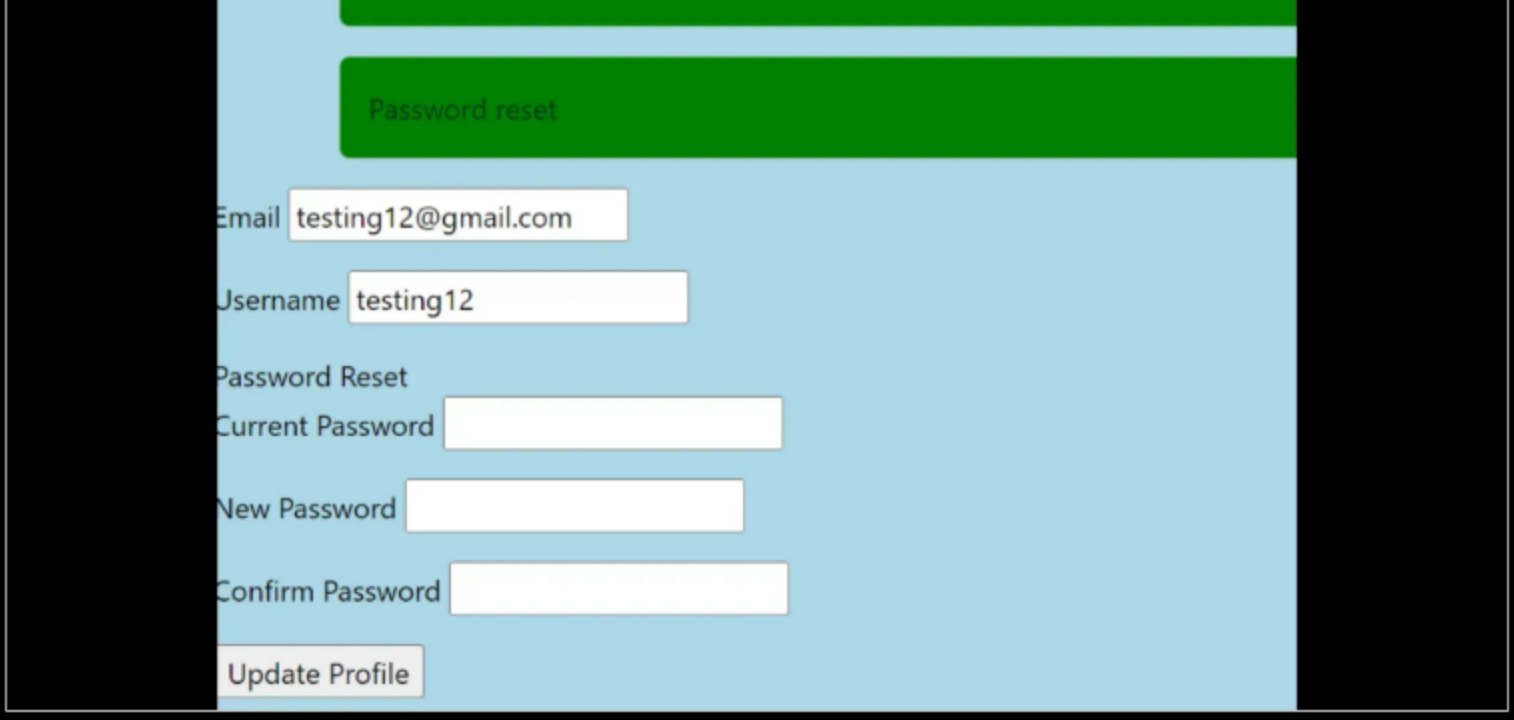
Checklist Items (0)



After email change

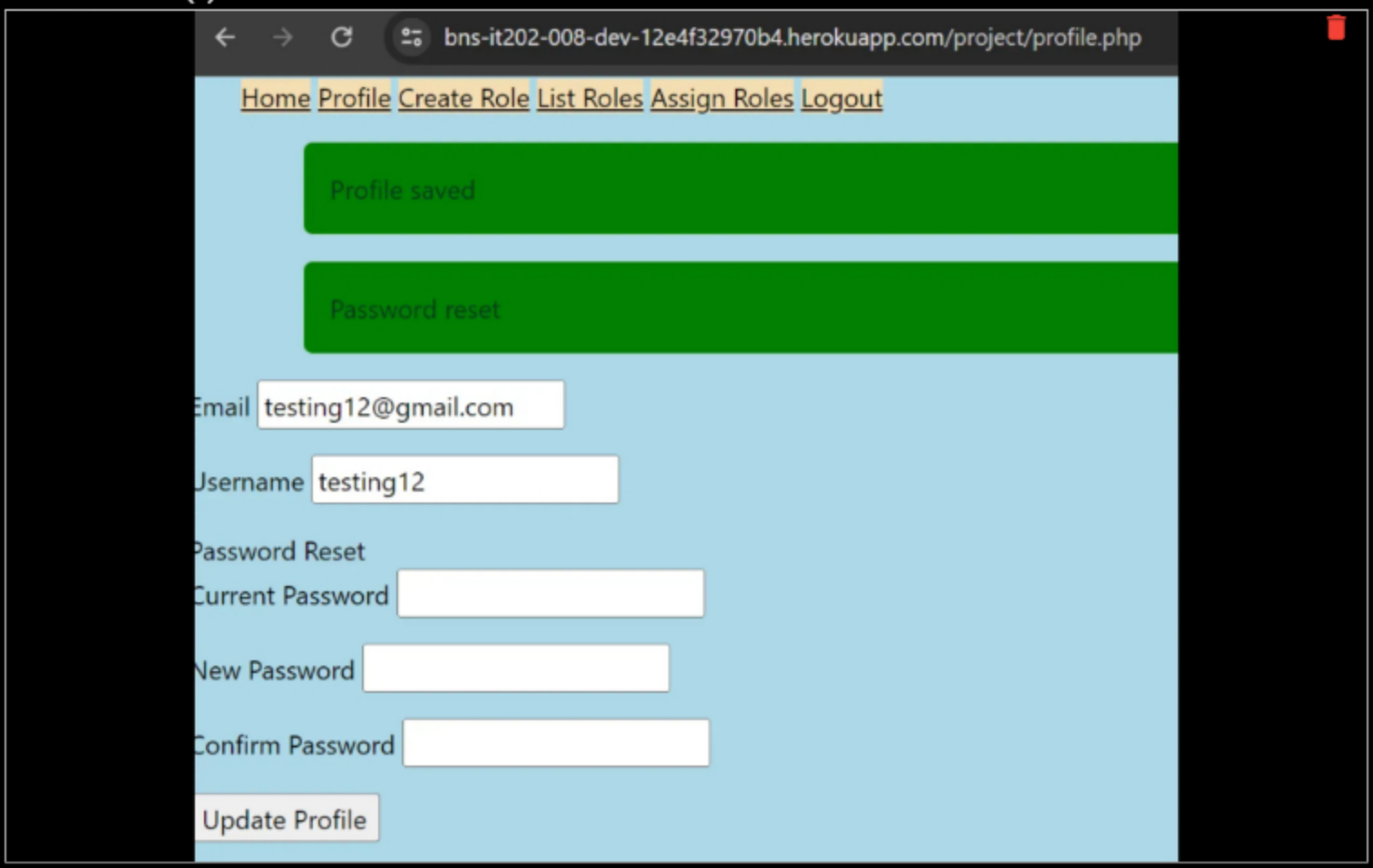
Checklist Items (0)





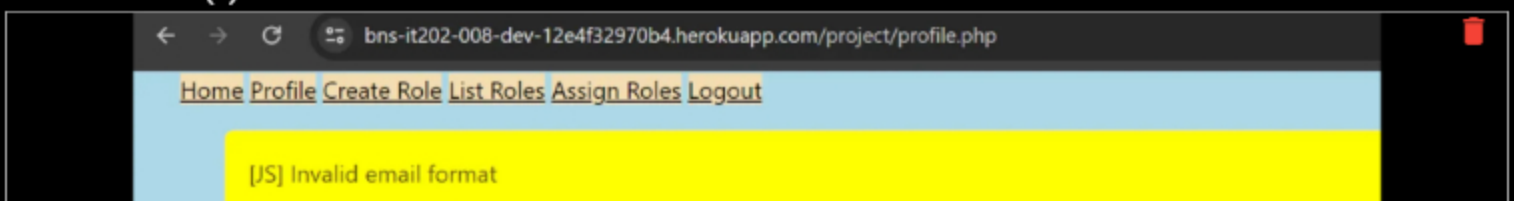
After username change

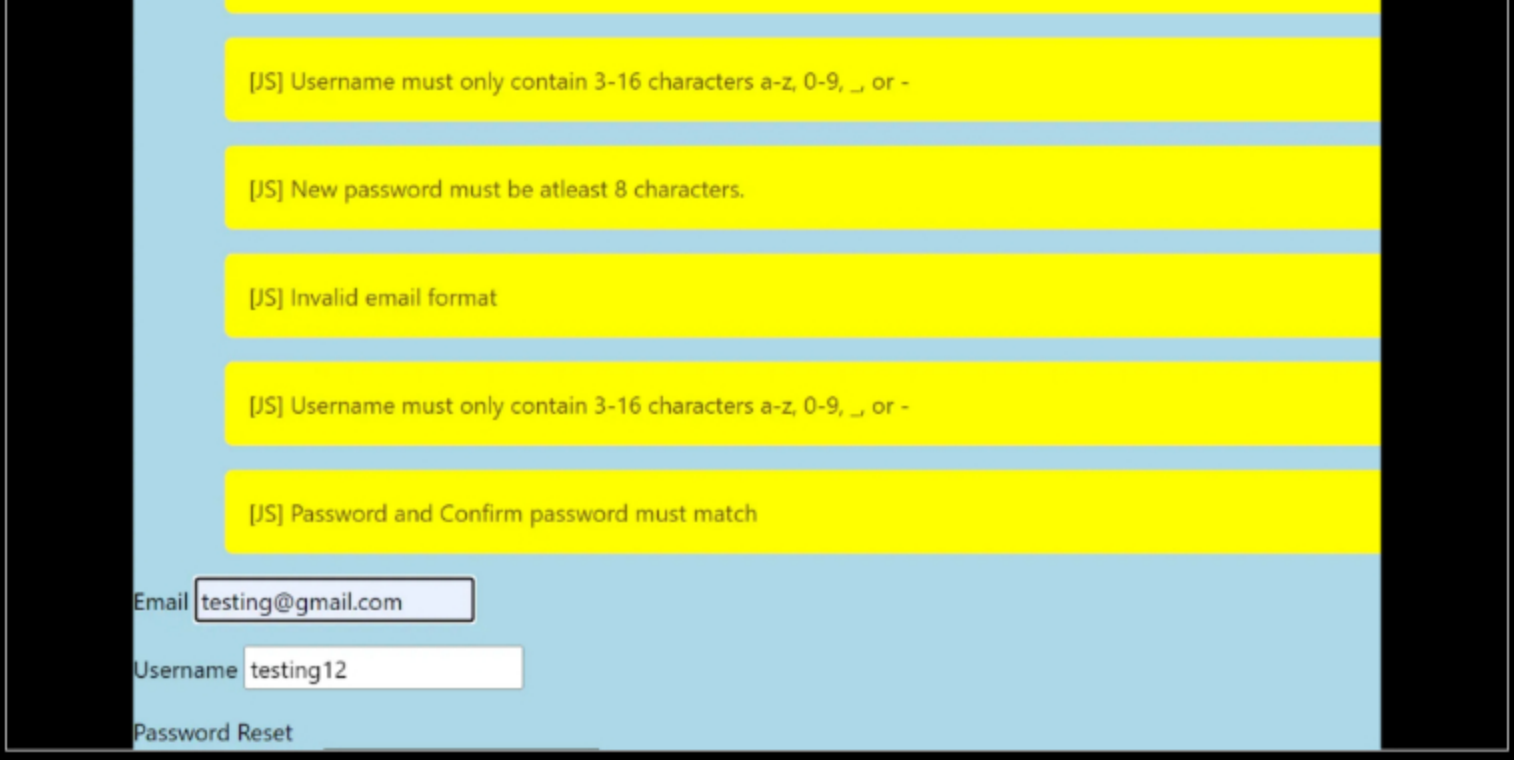
Checklist Items (0)



After password reset

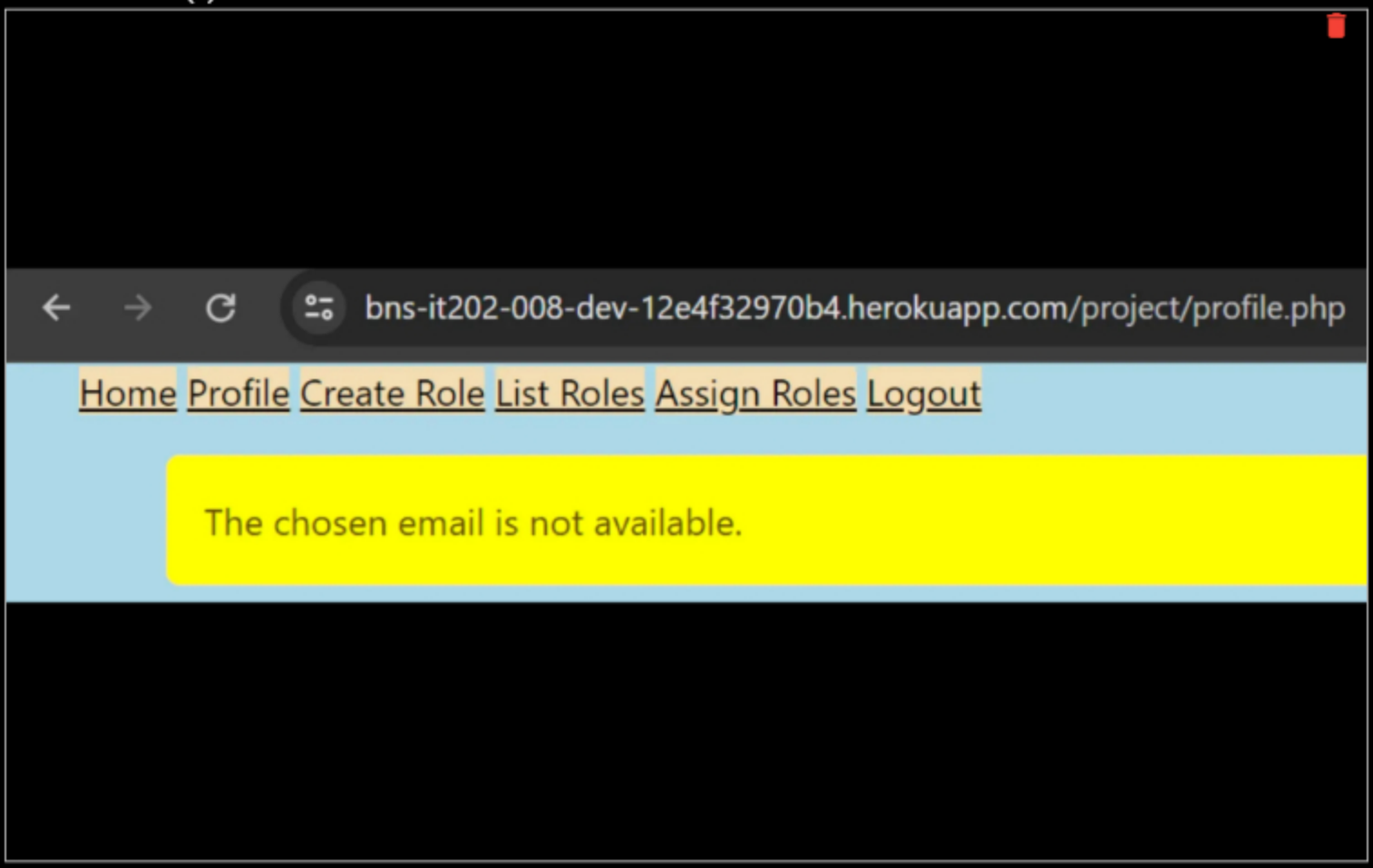
Checklist Items (0)





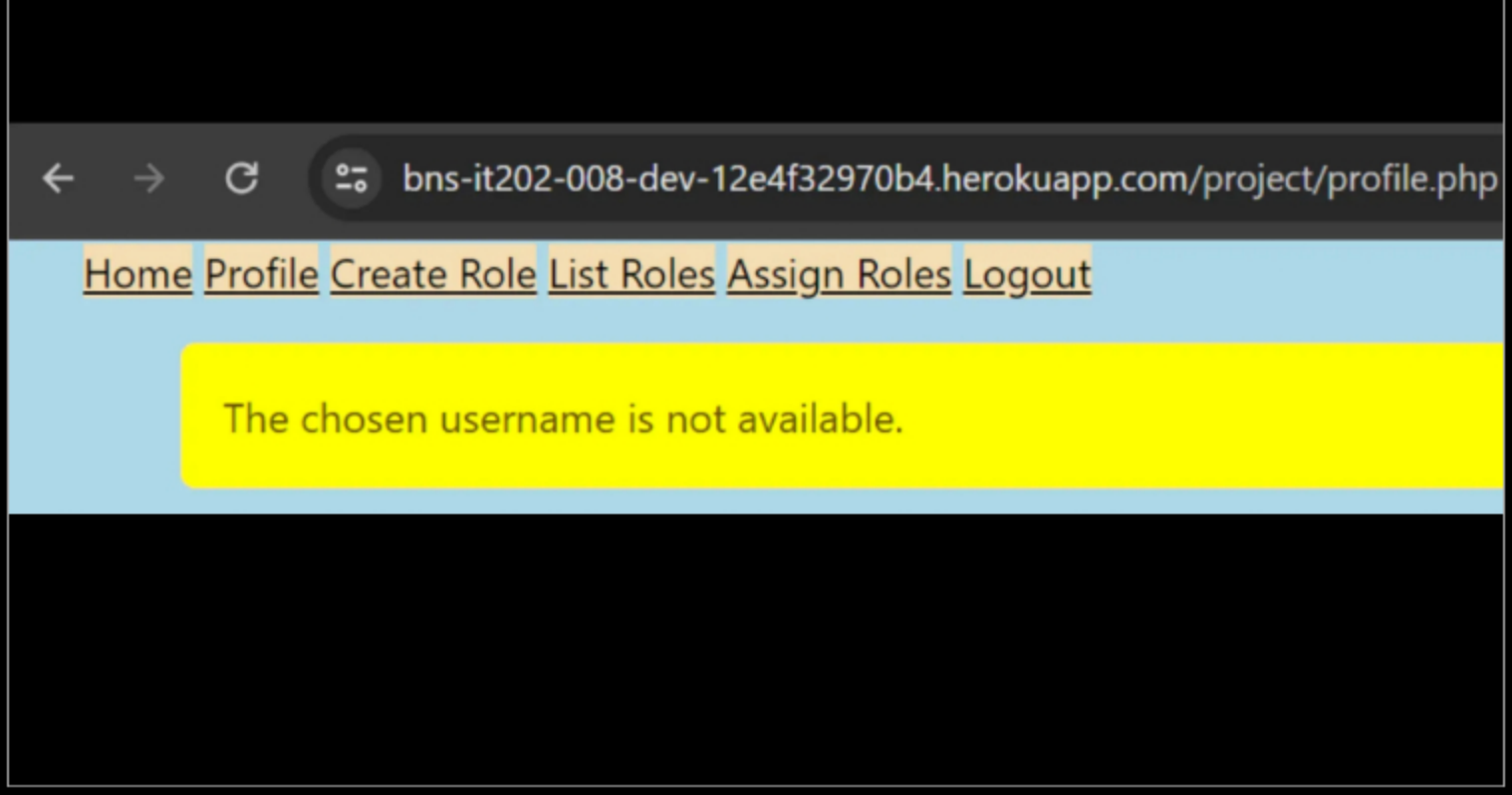
Javascript validation

Checklist Items (0)



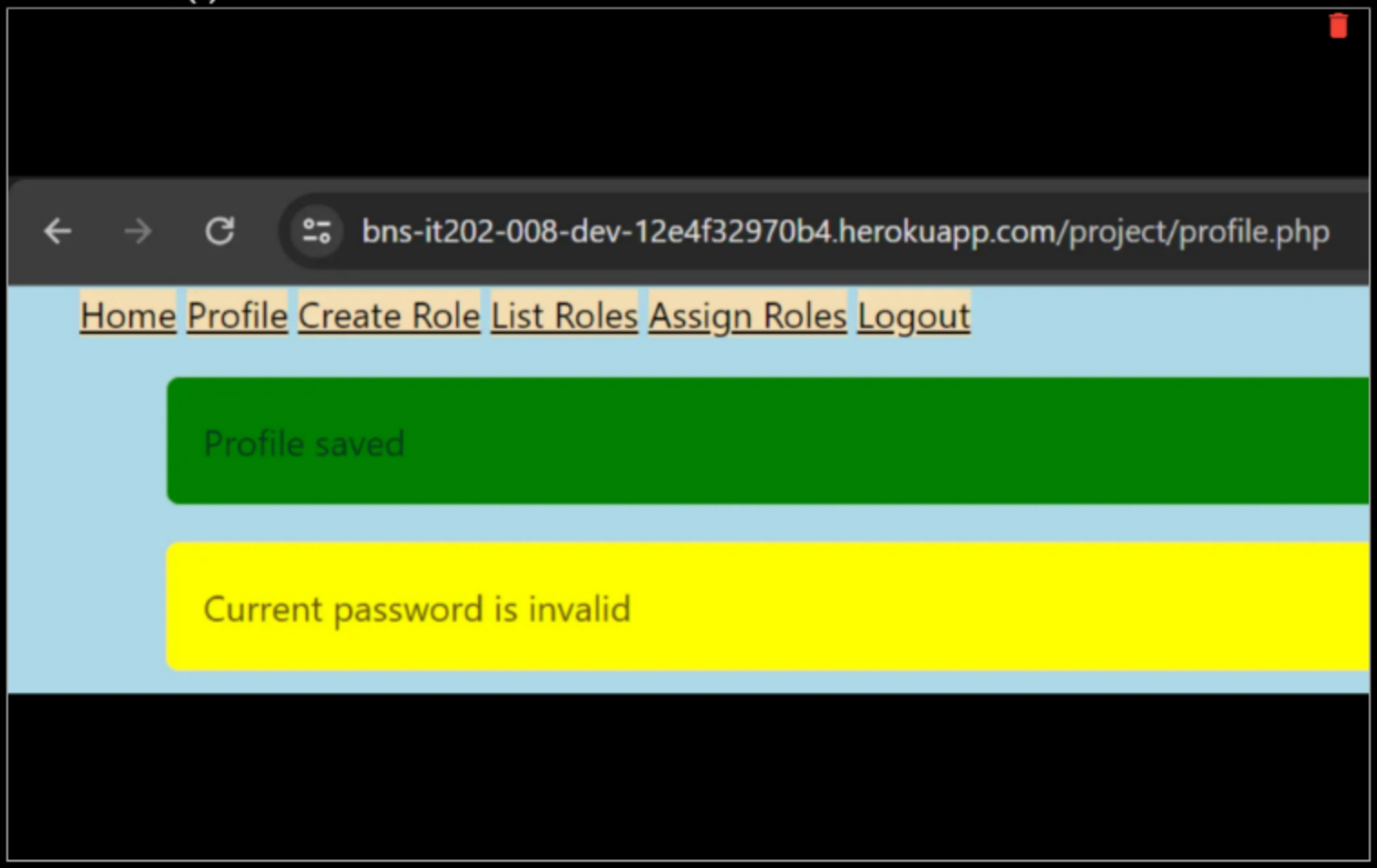
Email in use.

Checklist Items (0)



Username in use

Checklist Items (0)



Current Password is invalid

Checklist Items (0)

Task #4 - Points: 1

COLLAPSE

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

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Updating Username/Email
#2	1	Updating password
#3	1	Don't just show code, translate things to plain English

Response:

When the update button is clicked, the form is sent and goes through client side validation.

Client side validation checks if the new email is formatted correctly and if the new username is formatted correctly. It also checks to see if the new password has at least 8 characters, and makes sure that the new password and the confirm new password values match.

Then it goes to server side validation. Here it checks for similar things. It then checks to see if the input that was entered in the "Current Password" field matches with the current password of the user from the database. If they don't match, a flash message appears informing the user of that. If there are no further errors, it then prepares the database and updates the profile and resets the password.

COLLAPSE

Task #5 - Points: 1

Text: Include pull request links related to this feature

Details:
Should end in /pull/#

URL #1
https://github.com/BruceSiaNJIT/bns24-it202-008/pull/18

COLLAPSE

Misc (1 pt.)

COLLAPSE

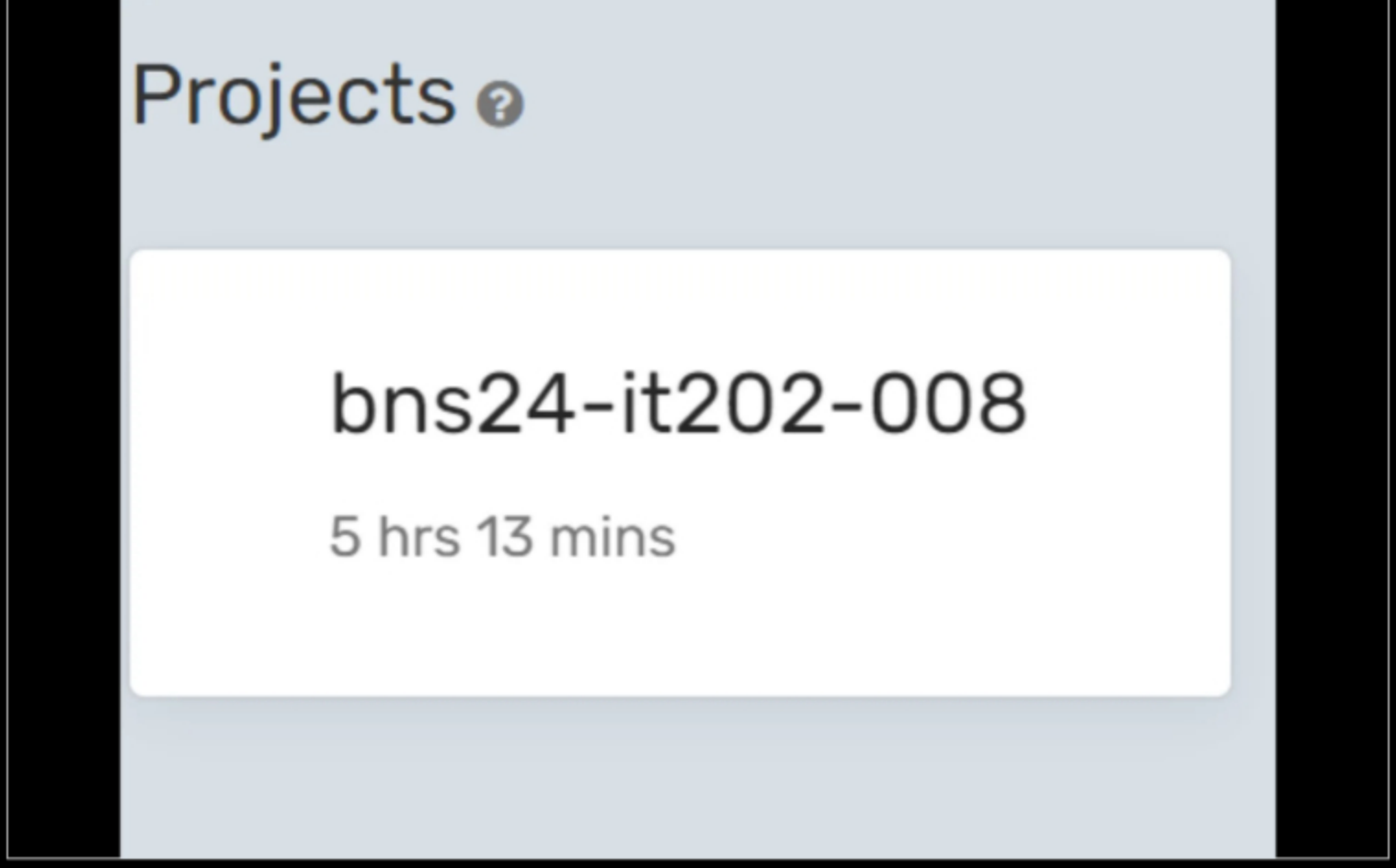
Task #1 - Points: 1

Text: Screenshot of wakatime

Task Screenshots:

Gallery Style: Large View

Small Medium Large



WakaTime screenshot

COLLAPSE

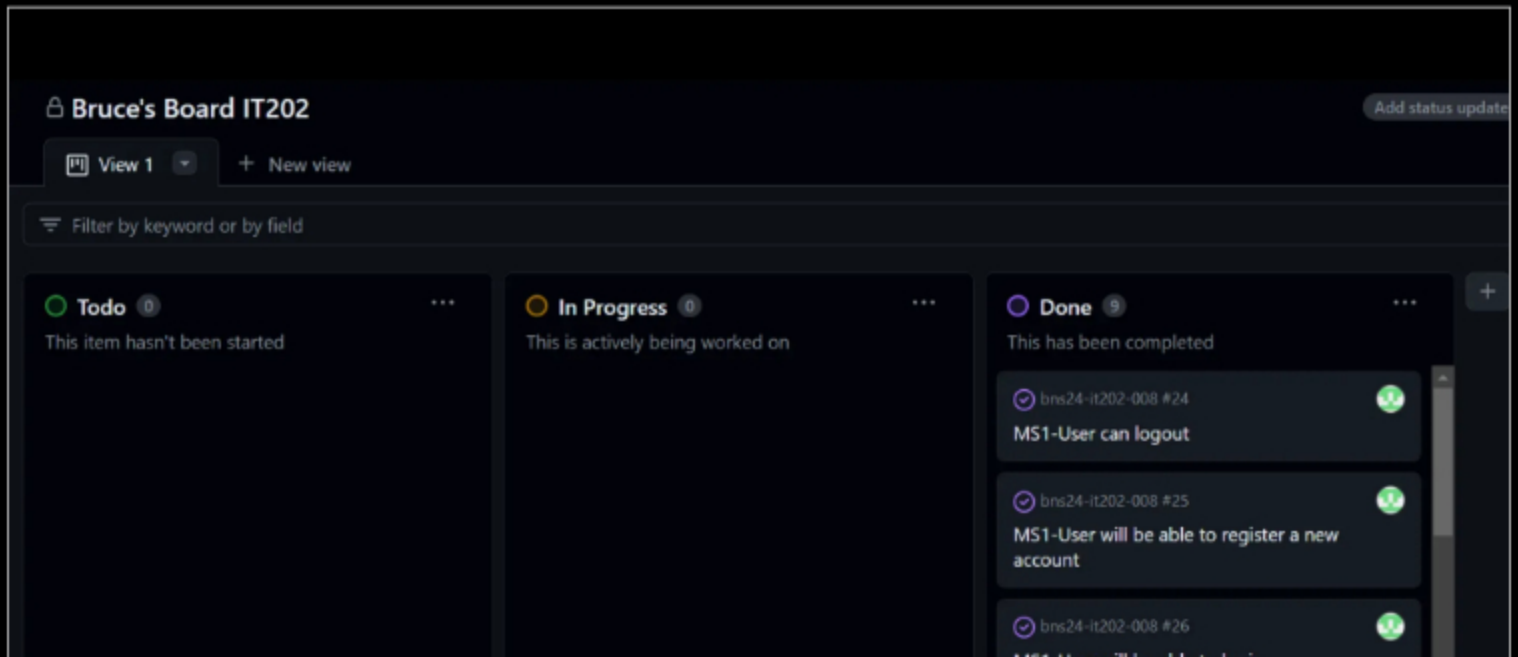
Task #2 - Points: 1

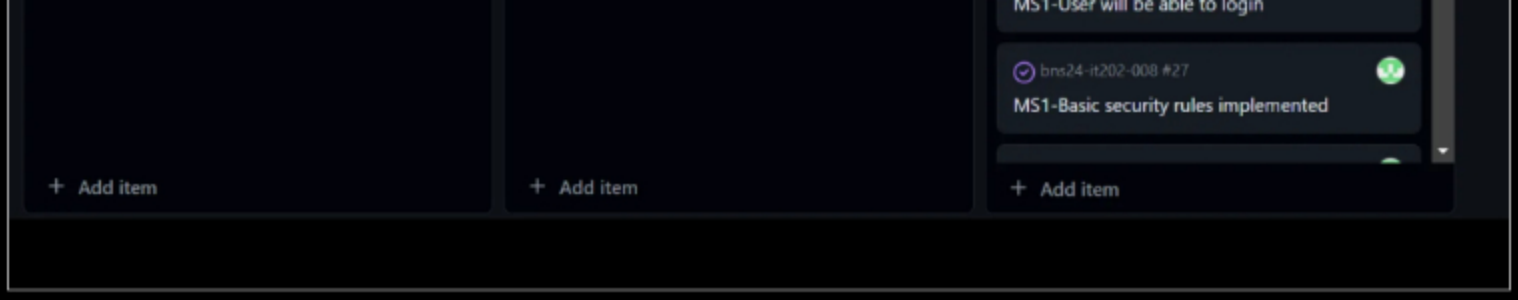
Text: Screenshot of your project board from GitHub (tasks should be in the proper column)

Task Screenshots:

Gallery Style: Large View

Small Medium Large





GIthub board

COLLAPSE

Task #3 - Points: 1

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

URL #1

https://github.com/users/BruceSiaNJIT/projects/1/views/1

COLLAPSE

Task #4 - Points: 1

Text: Provide a direct link to the login page from your prod instance

URL #1

https://bns24-it202-008-prod-4d42e45e31d0.herokuapp.com/project/login.php

End of Assignment