Internet Programming 622

RICHFIELD
richfield.ac.za

Name and Surname: Novithan Naidoo

Student ITS No: 402412304

Qualification: Bachelor of Science Information Technology **Year of Study:** 2025 **Semester:** 2

Assignment due date: 31 October 2025 **Date submitted:** 31 October 2025

QUESTION	EXAMINER MARKS	MODERATOR MARKS	REMARKS

ASSIGNMENT INSTRUCTIONS

Please tick each box to confirm completion. (10 Marks)

Use Times New Roman font, size 12, with 1.5 line spacing throughout the document. Apply Harvard Referencing Style for all citations and references.

For essay-style assignments, please include the following sections:

Table of Contents Introduction

Main Body (with relevant subheadings) Conclusion

References

Submit the assignment in PDF format on Moodle. Use the specified cover page provided.

Include a signed declaration of originality.

DECLARATION OF ORIGINALITY:

I hereby declare that this assignment is my own work and has not been copied from any other source except where due acknowledgment is made. I affirm that all sources used have been properly cited and that this submission complies with the institution's policies on academic integrity and plagiarism.

Student Signature: Nnaidoo Date: 31 October 2025

Table of Contents

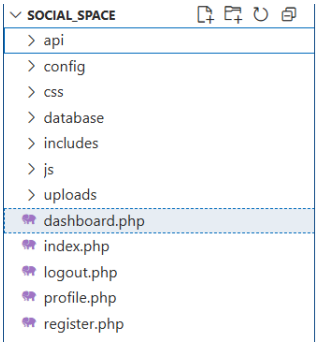
Introduction

This project is a full-stack social media application, developed over the course of two months of intensive trial and error. Throughout the development process, I faced numerous challenges that required me to delete and restart multiple components of the project, making it more challenging and a truly demanding yet rewarding experience. The complexity of building a fully functional social media platform is not easy and certainly pushed me to solve problems constantly and deepen my understanding of web development.

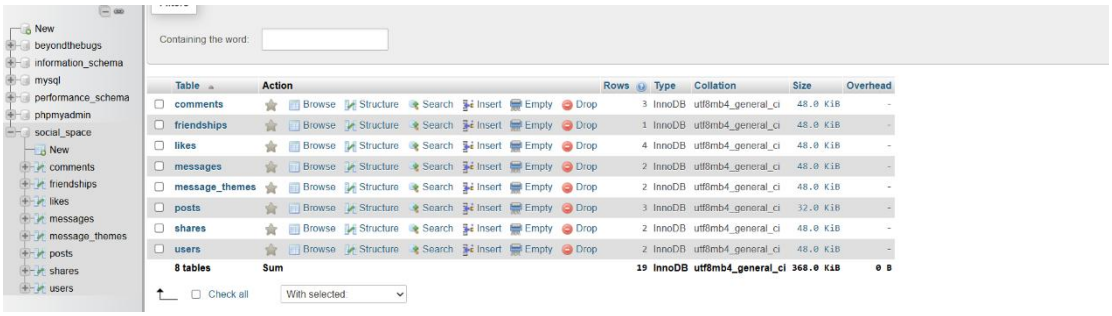
The application as built using a combination of HTML, CSS, JavaScript, PHP, SQL with phpMyAdmin used to manage the database. This technology stack allowed me to implement both the front -end interface and the back-end functionality, providing users with a seamless and interactive experience. Overall, this project not only tested my technical skills but also my problem-solving skills because that is what writing code I all about, solving problems, I really failed and tried very hard with this project.

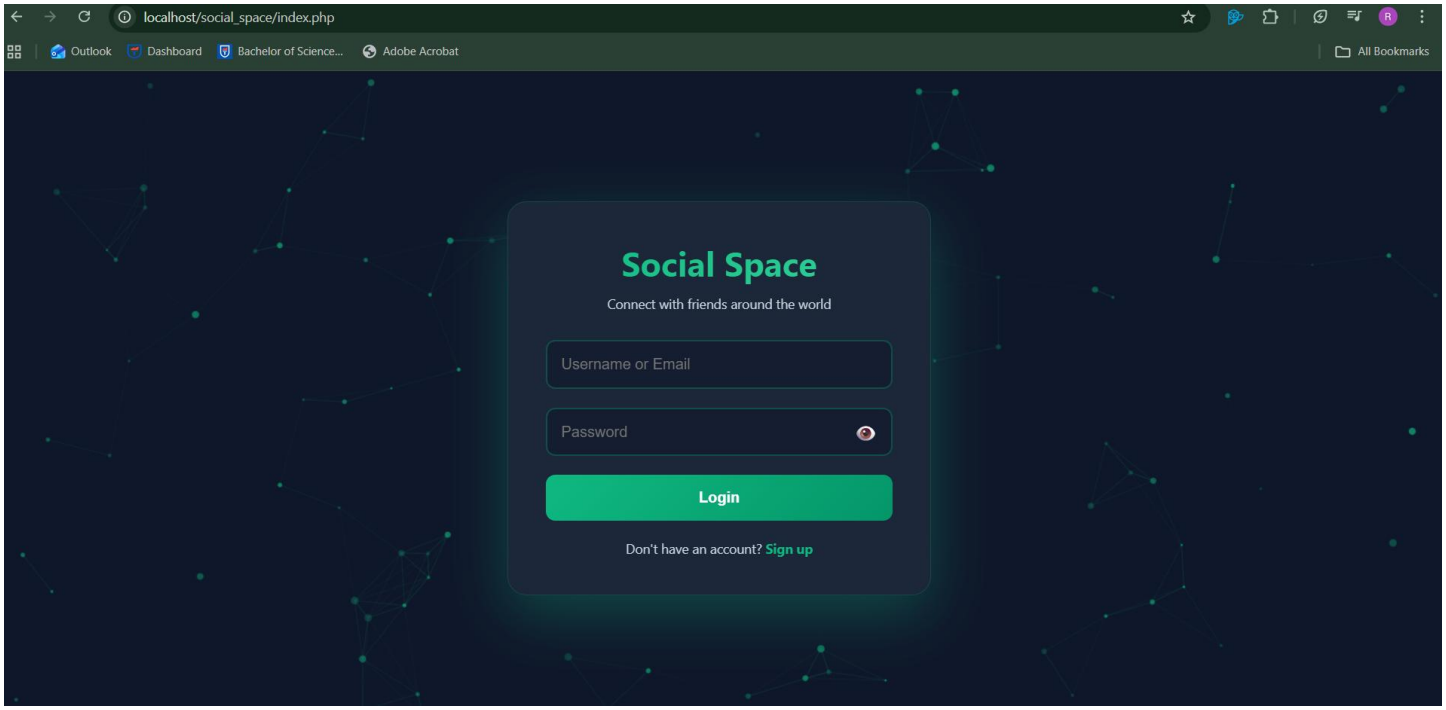
Question 1

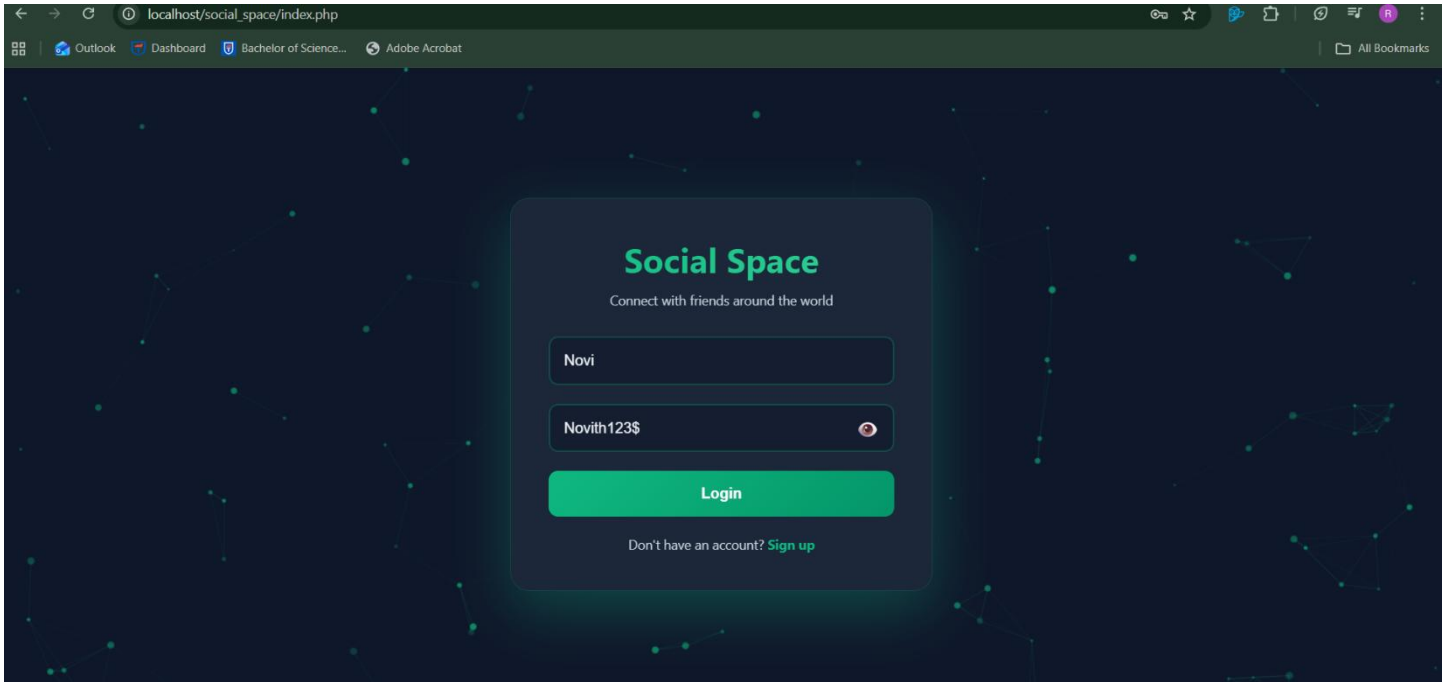
File Structure

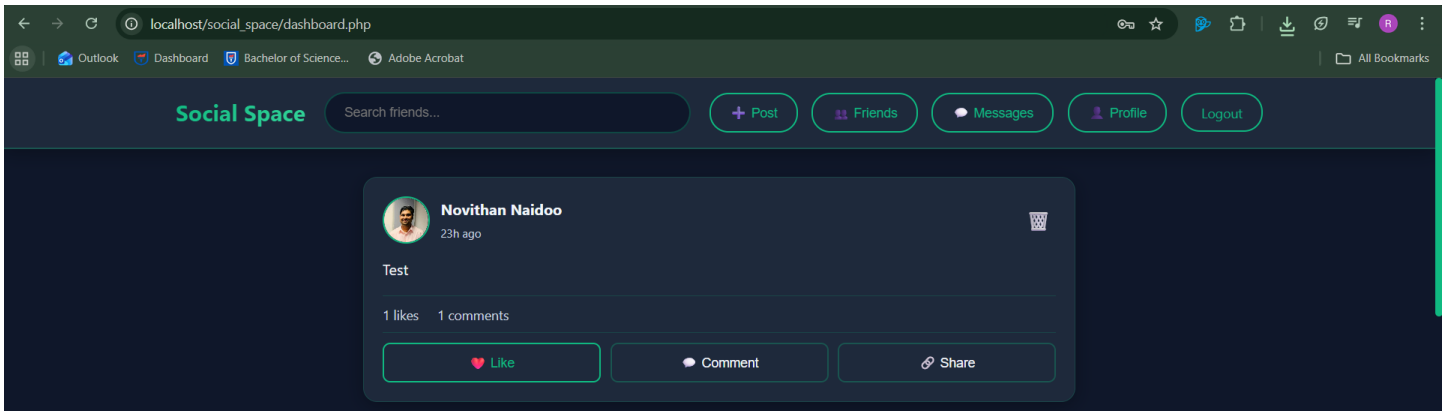


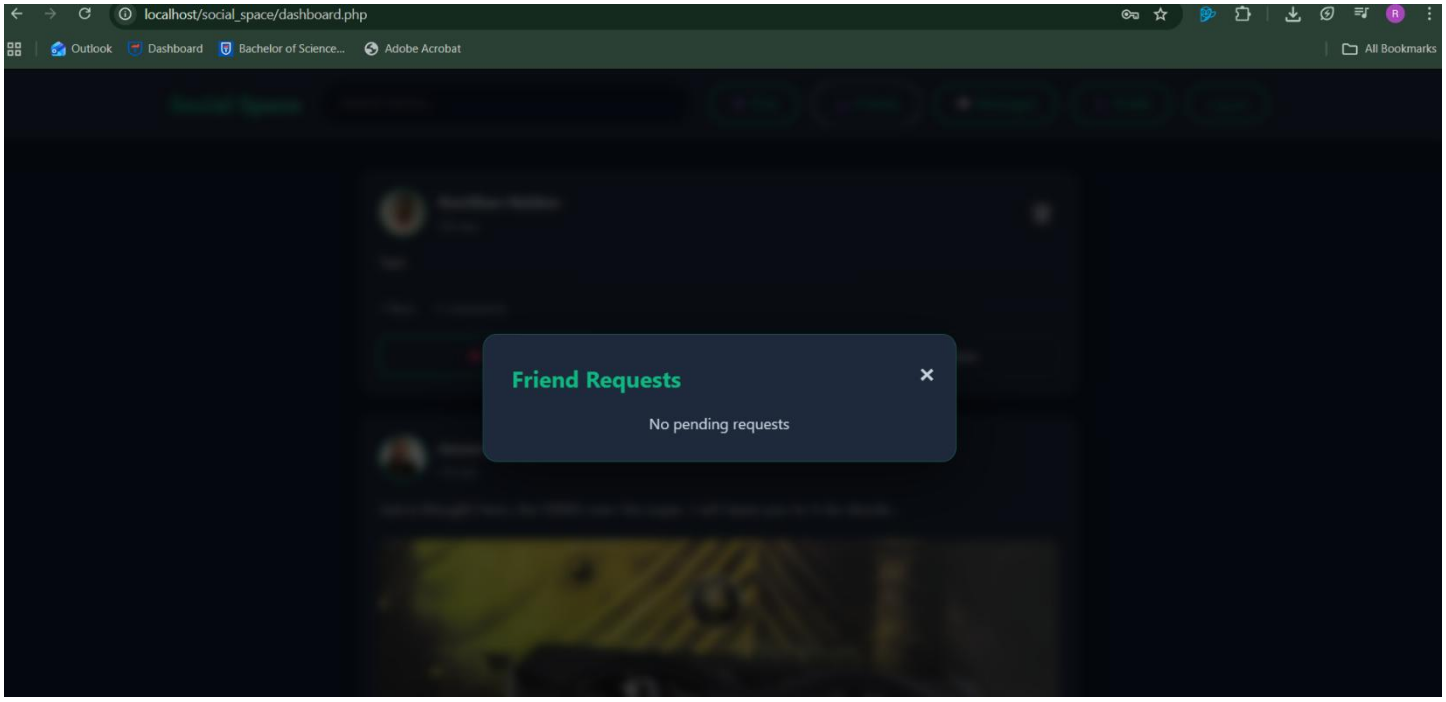
Database Structure

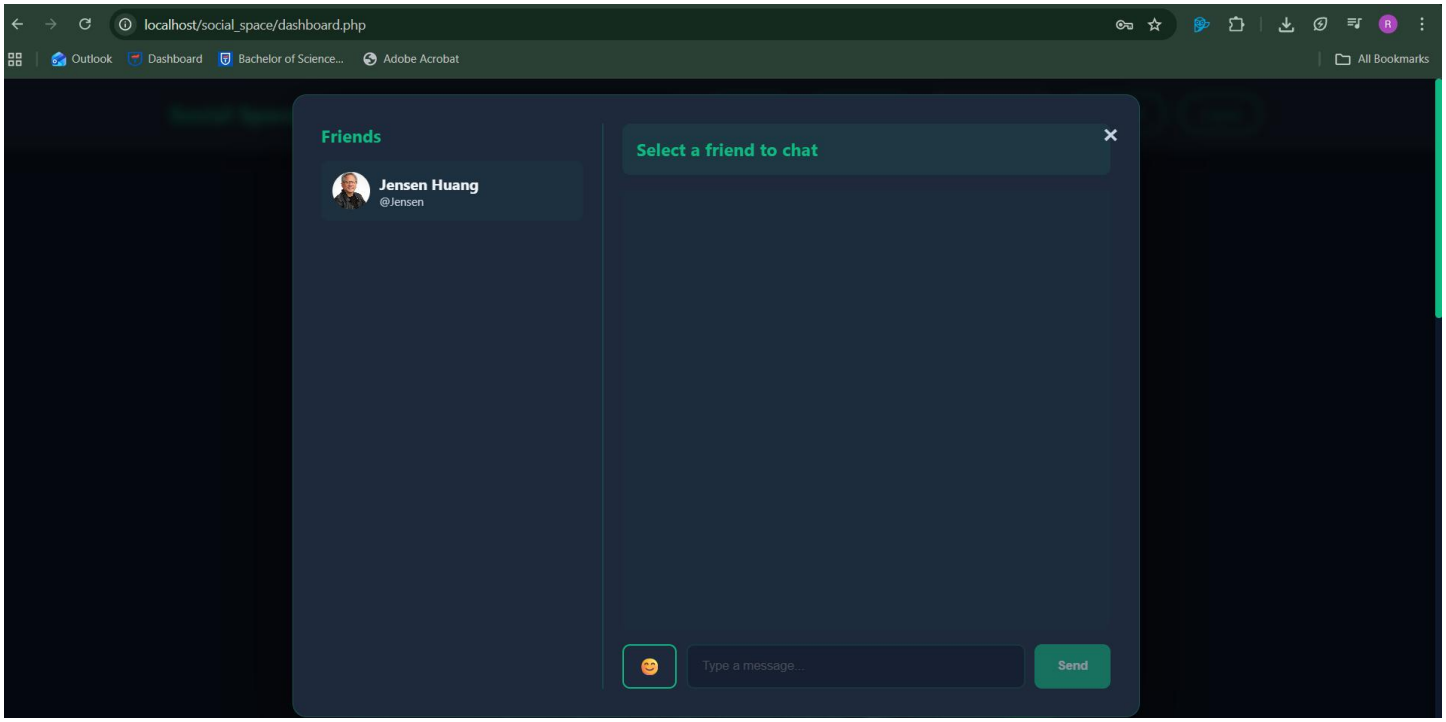


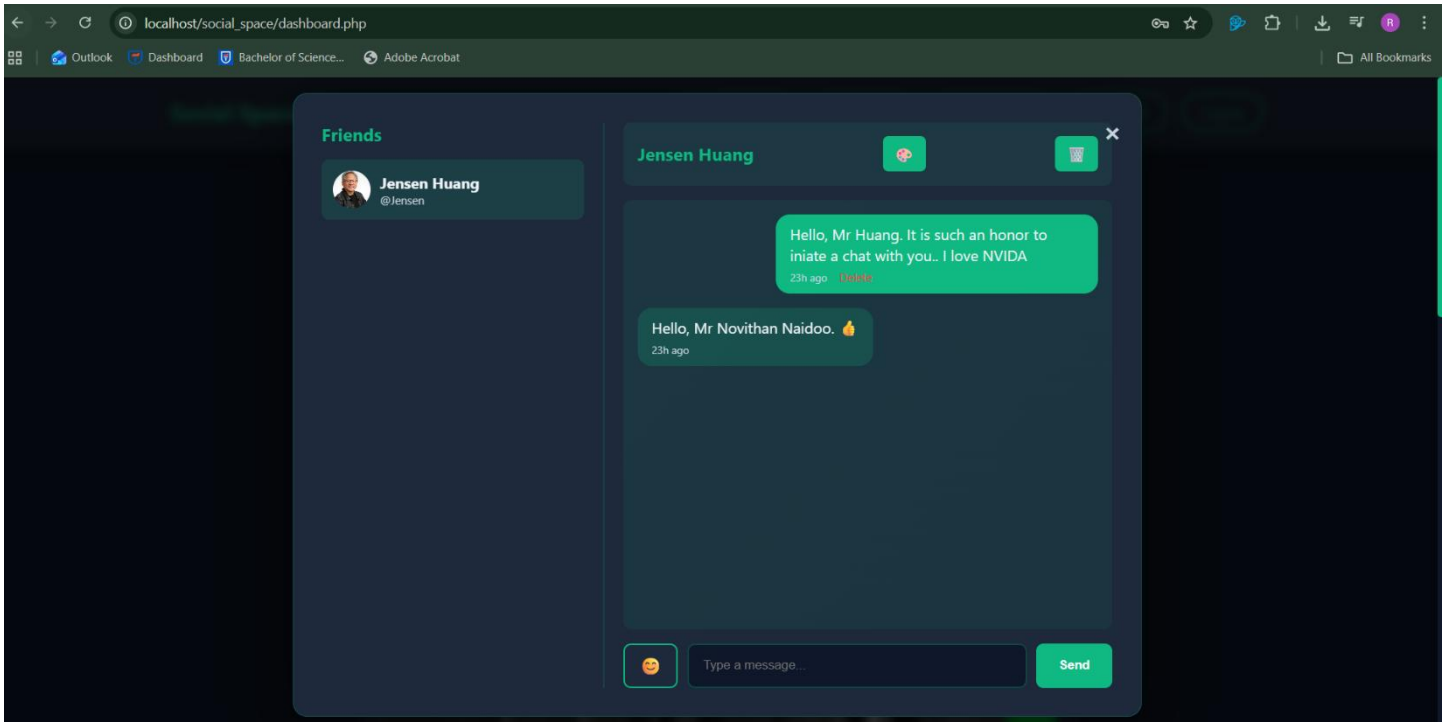


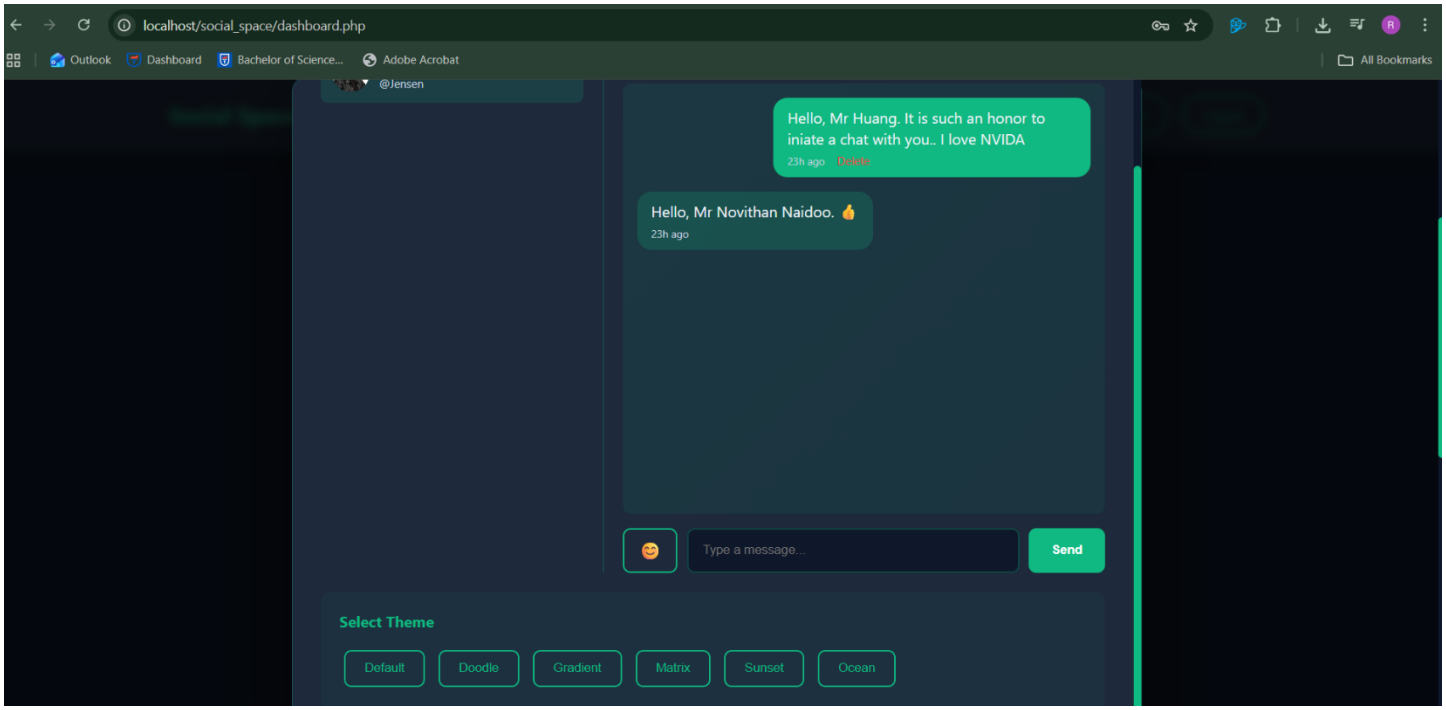

localhost/social_space/index.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Social Space
Connect with friends around the world
Novi
Novith123$
Login
Don't have an account? Sign up

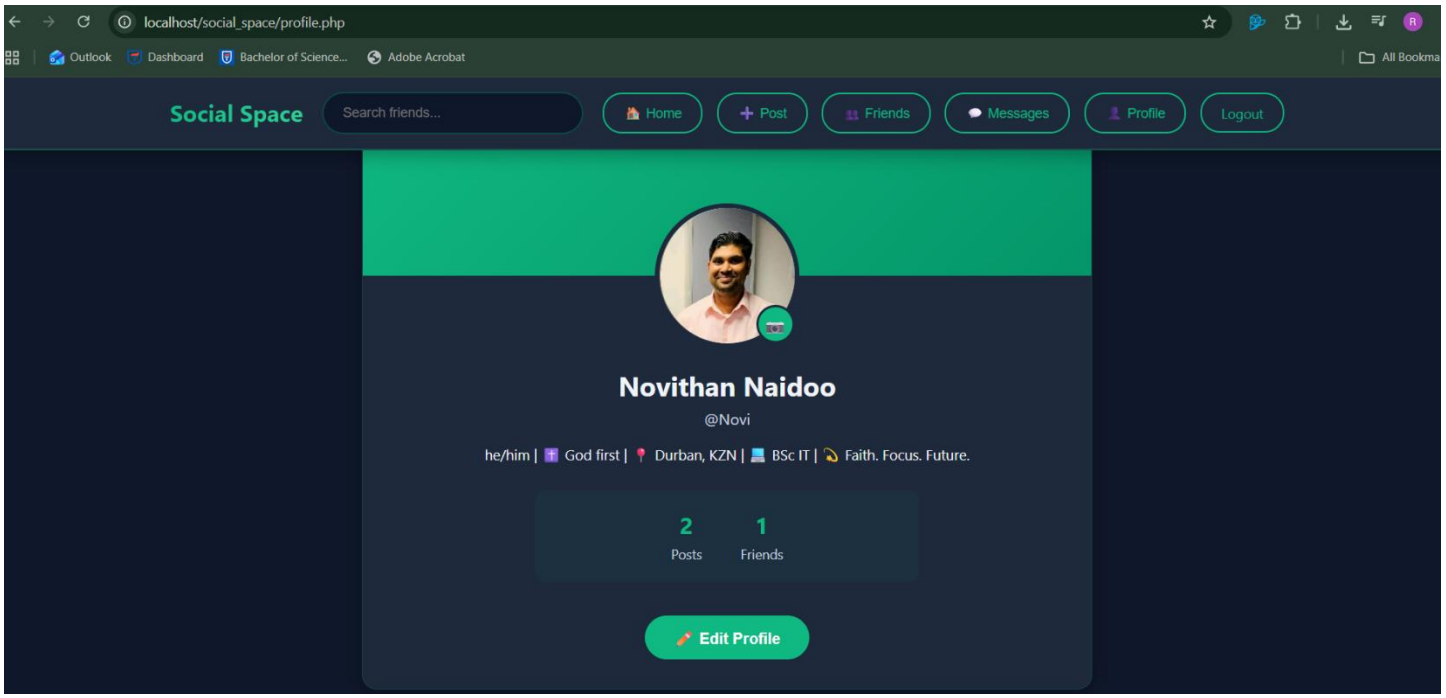

localhost/social_space/dashboard.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Social Space
Search friends...
Post
Friends
Messages
Profile
Logout
Novithan Naidoo
23h ago
Test
1 likes
1 comments
Like
Comment
Share

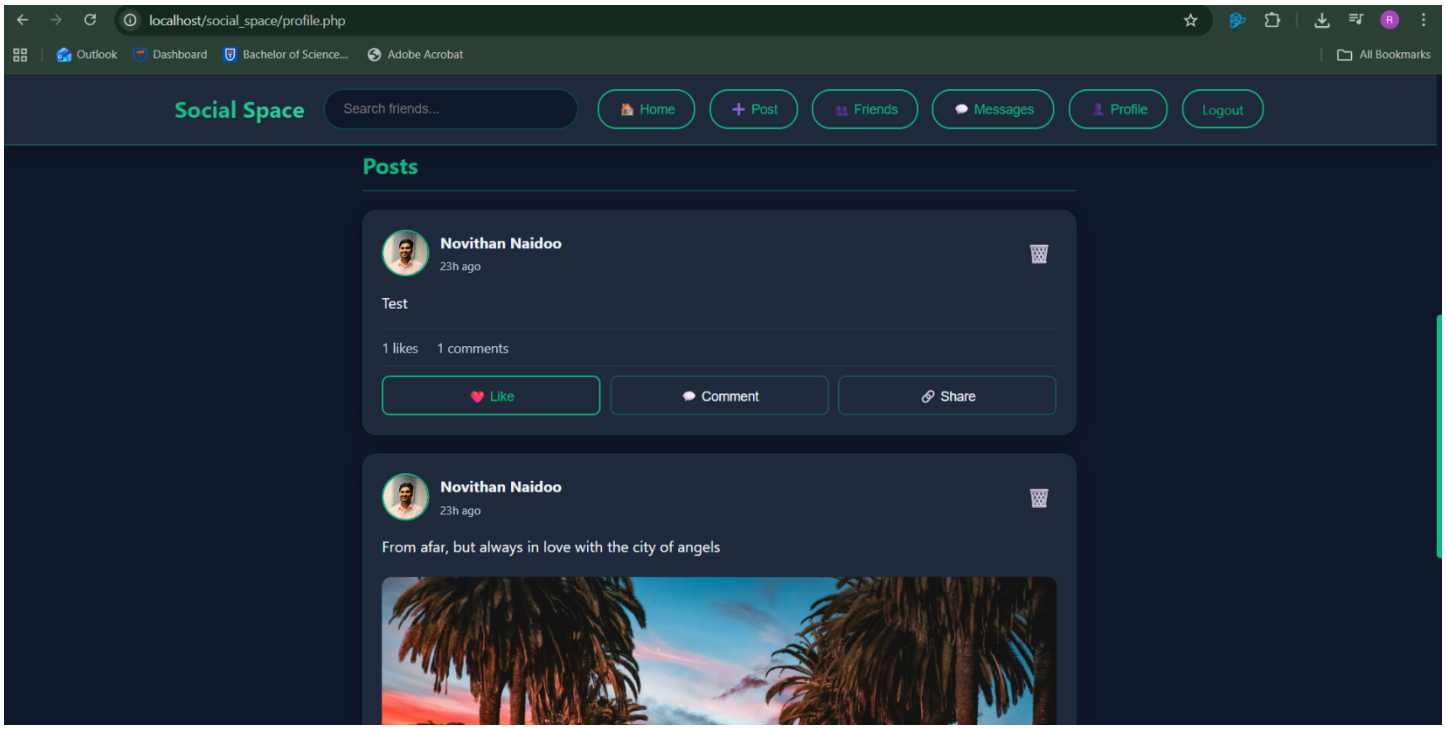

localhost/social_space/dashboard.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Friend Requests
No pending requests

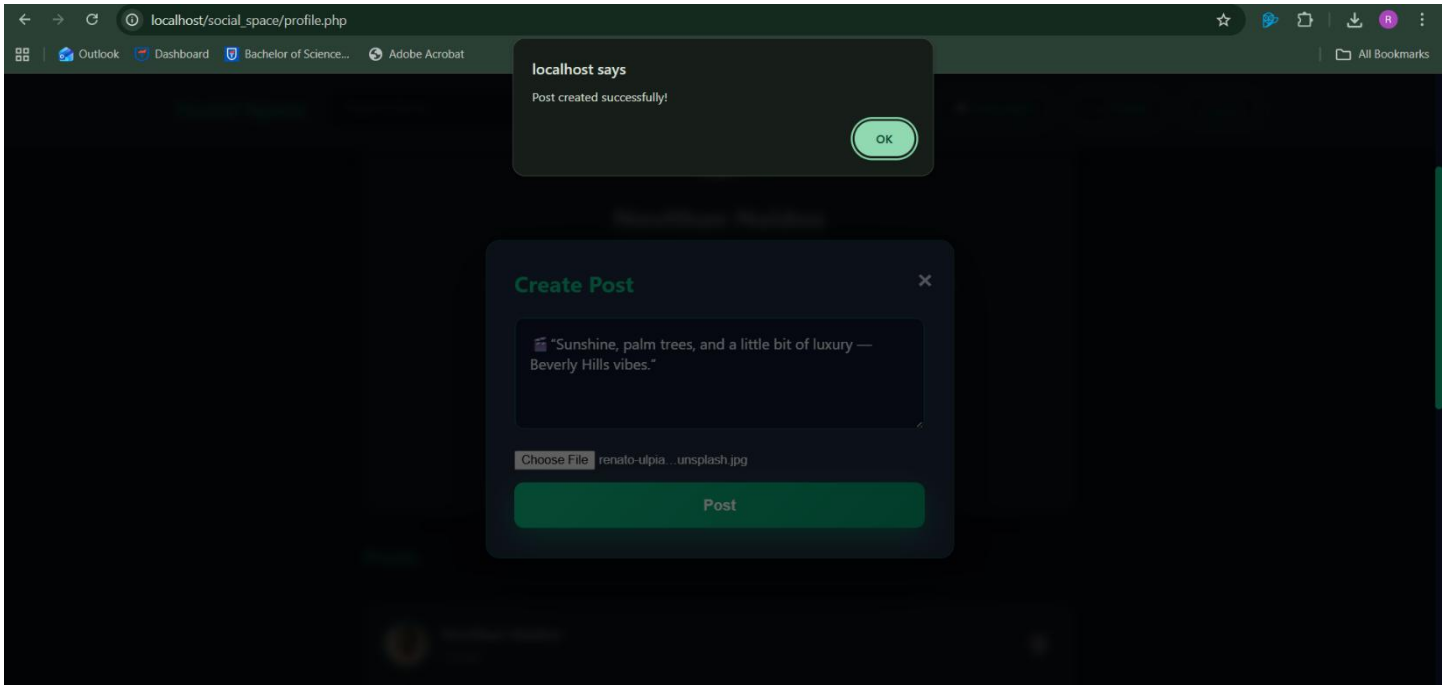

localhost/social_space/dashboard.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Friends
Jensen Huang
@Jensen
Select a friend to chat
Type a message...
Send

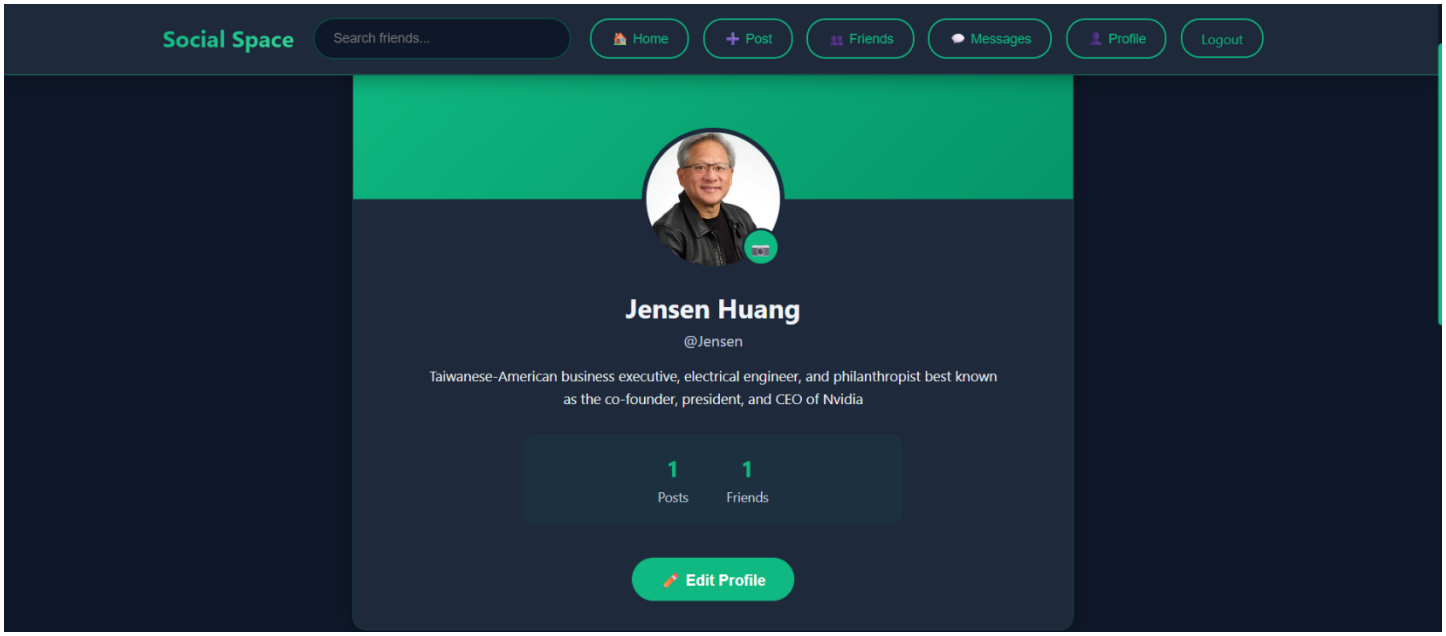

localhost/social_space/dashboard.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Friends
Jensen Huang
@Jensen
Jensen Huang
Hello, Mr Huang. It is such an honor to iniate a chat with you.. I love NVIDA
23h ago
Hello, Mr Novithan Naidoo.
23h ago
Type a message...
Send

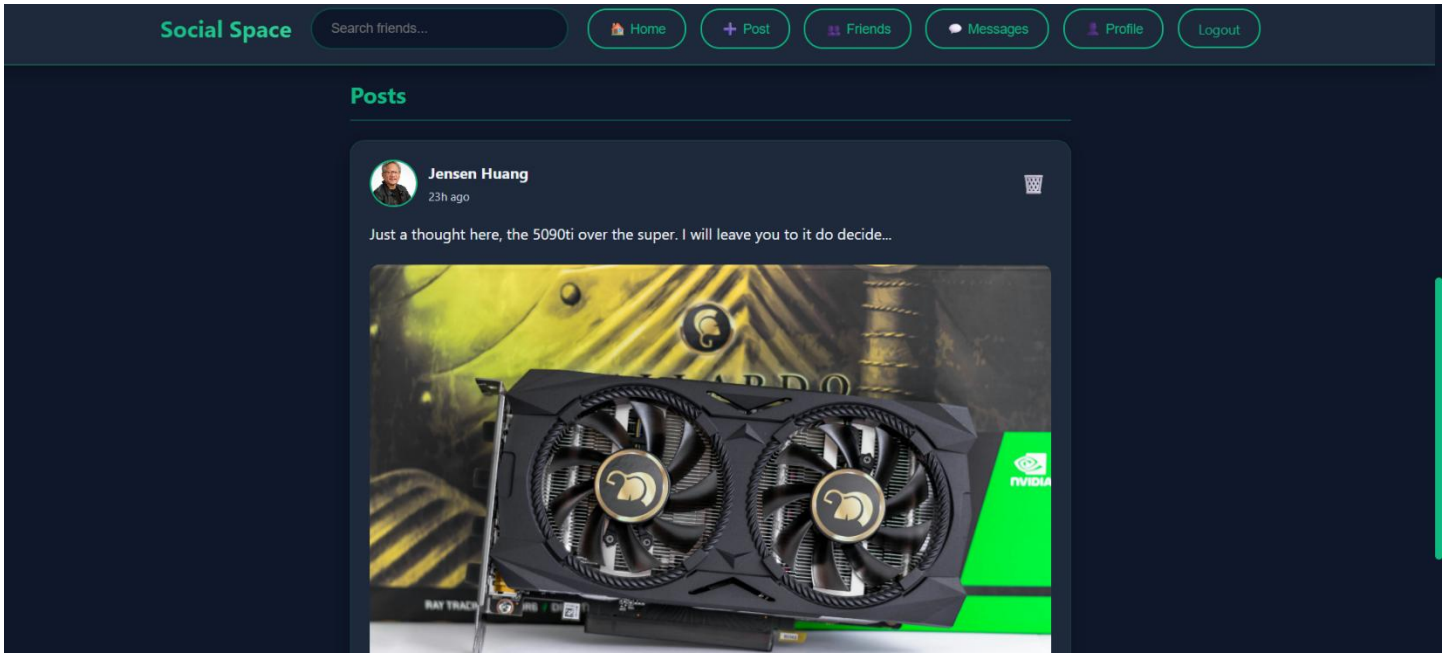

localhost/social_space/dashboard.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
@Jensen
Hello, Mr Huang. It is such an honor to iniate a chat with you.. I love NVIDA
23h ago
Delete
Hello, Mr Novithan Naidoo.
23h ago
Type a message...
Send
Select Theme
Default
Doodle
Gradient
Matrix
Sunset
Ocean

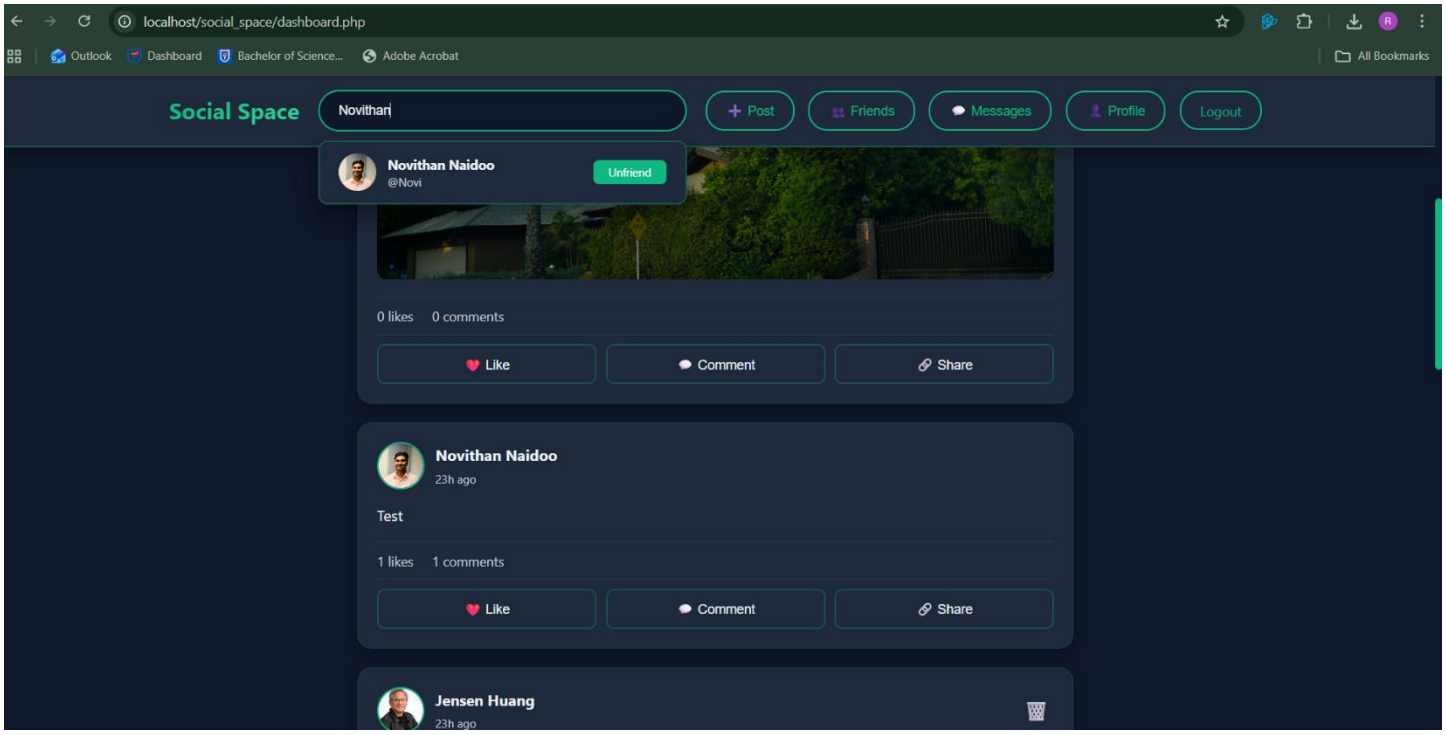

localhost/social_space/profile.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
Social Space
Search friends...
Home
Post
Friends
Messages
Profile
Logout
Novithan Naidoo
@Novi
he/him | God first | Durban, KZN | BSc IT | Faith. Focus. Future.
2
Posts
1
Friends
Edit Profile

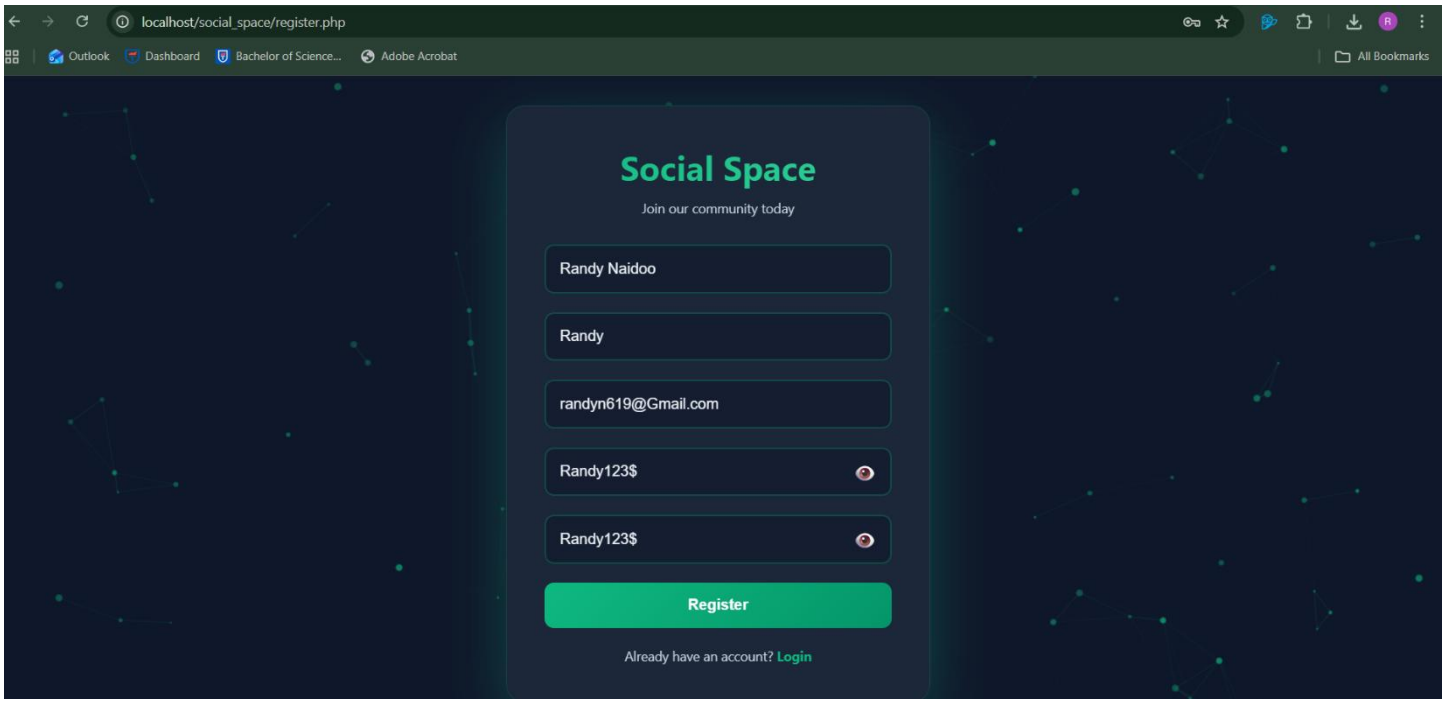

localhost/social_space/profile.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Social Space
Search friends...
Home
Post
Friends
Messages
Profile
Logout
Posts
Novithan Naidoo
23h ago
Test
1 likes
1 comments
Like
Comment
Share
Novithan Naidoo
23h ago
From afar, but always in love with the city of angels

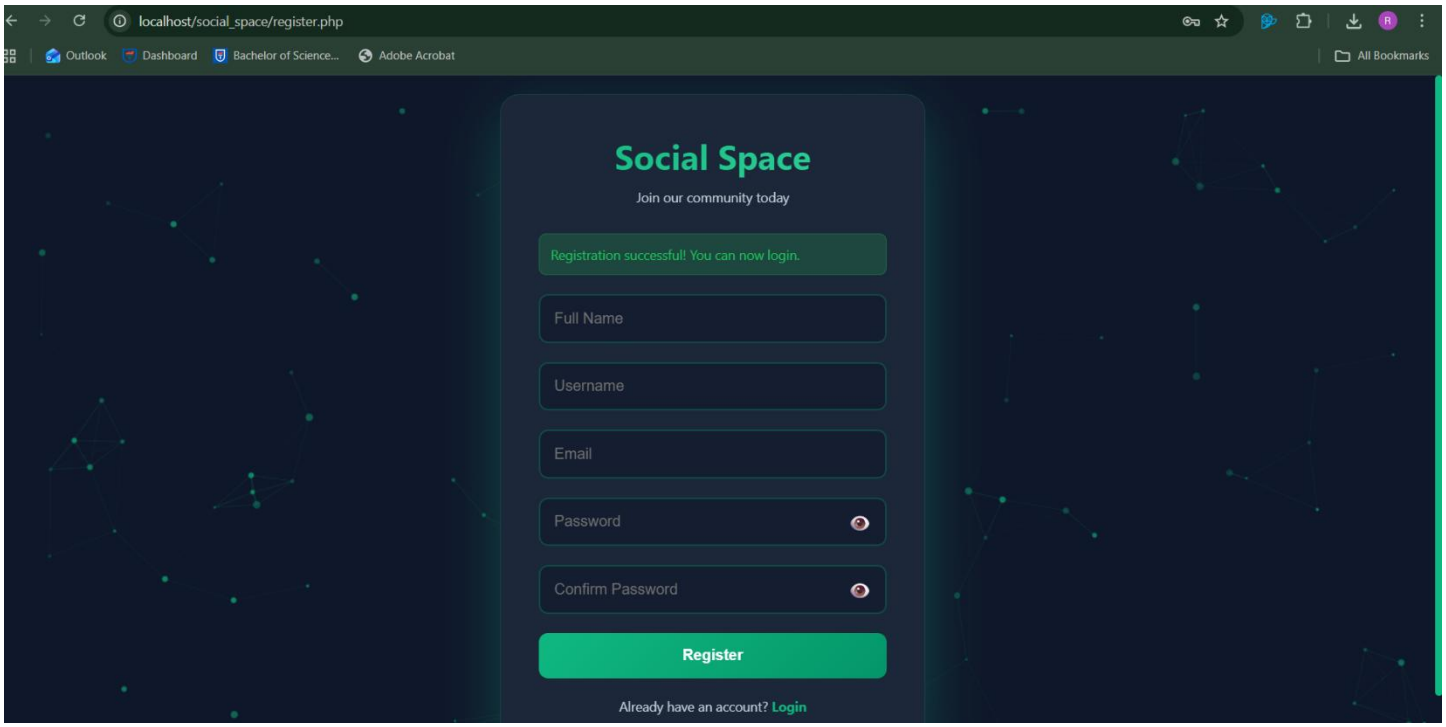

localhost/social_space/profile.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
localhost says
Post created successfully!
OK
Create Post
"Sunshine, palm trees, and a little bit of luxury — Beverly Hills vibes."
Choose File
renato-ulpia...unsplash.jpg
Post

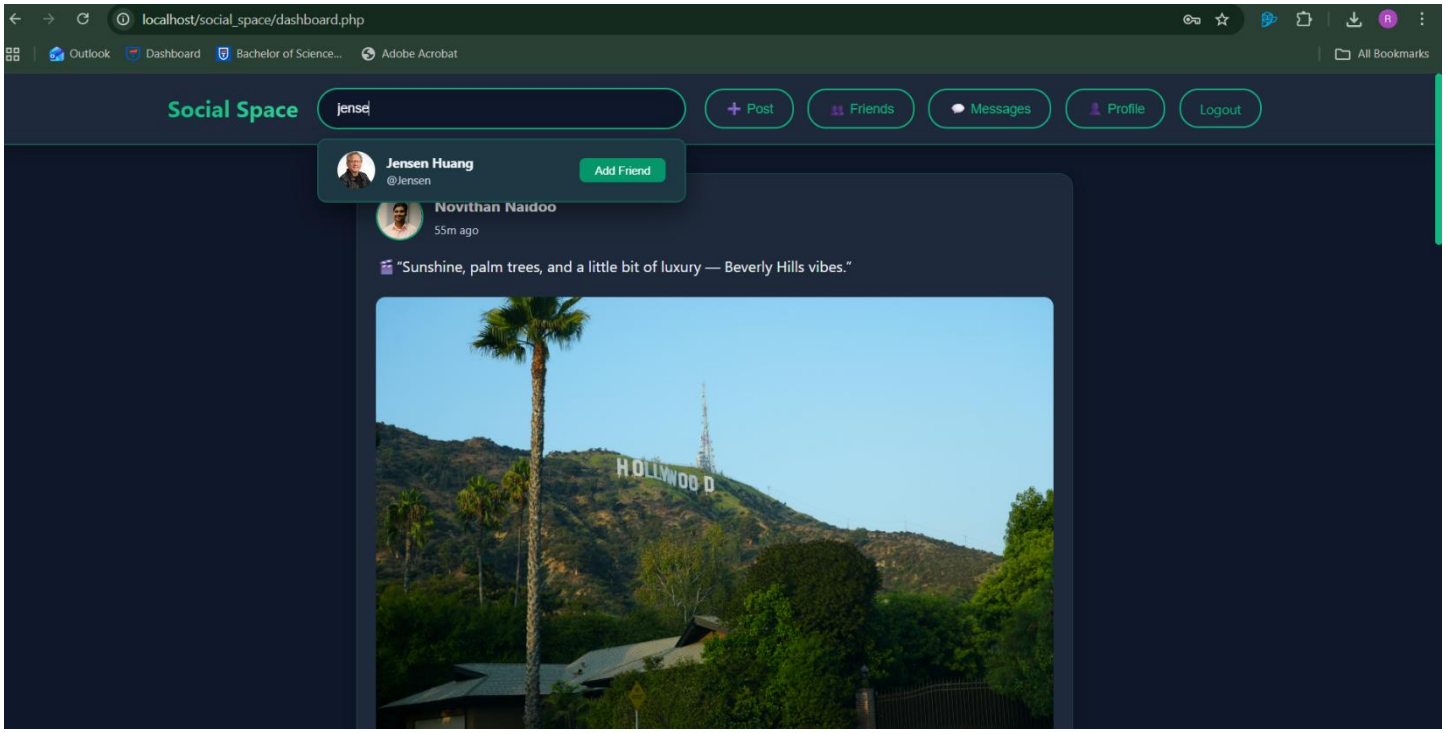

Social Space
Search friends...
Home
Post
Friends
Messages
Profile
Logout
Jensen Huang
@Jensen
Taiwanese-American business executive, electrical engineer, and philanthropist best known as the co-founder, president, and CEO of Nvidia
1
Posts
1
Friends
Edit Profile

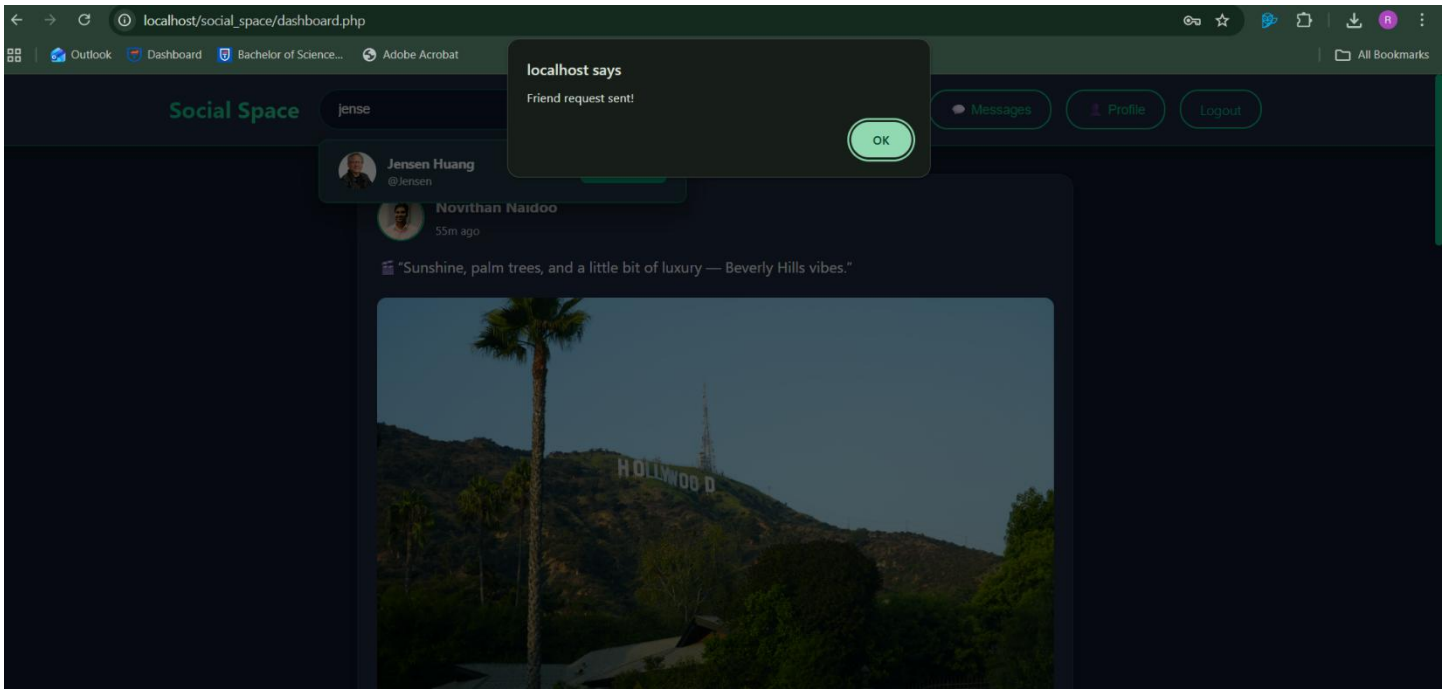

Social Space
Search friends...
Home
Post
Friends
Messages
Profile
Logout
Posts
Jensen Huang
23h ago
Just a thought here, the 5090ti over the super. I will leave you to it do decide...

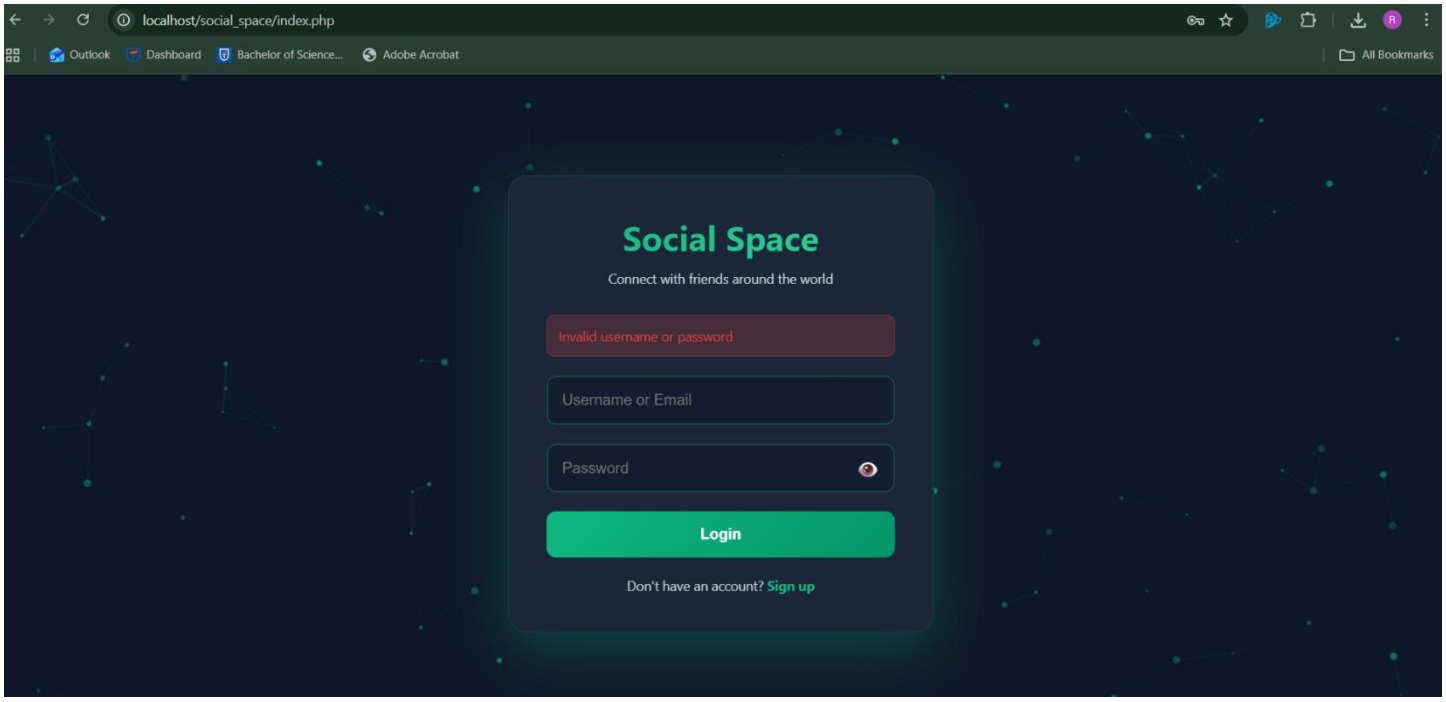

localhost/social_space/dashboard.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Social Space
Novithan
Post
Friends
Messages
Profile
Logout
Novithan Naidoo
@Novi
Unfriend
0 likes
0 comments
Like
Comment
Share
Novithan Naidoo
23h ago
Test
1 likes
1 comments
Like
Comment
Share
Jensen Huang
23h ago

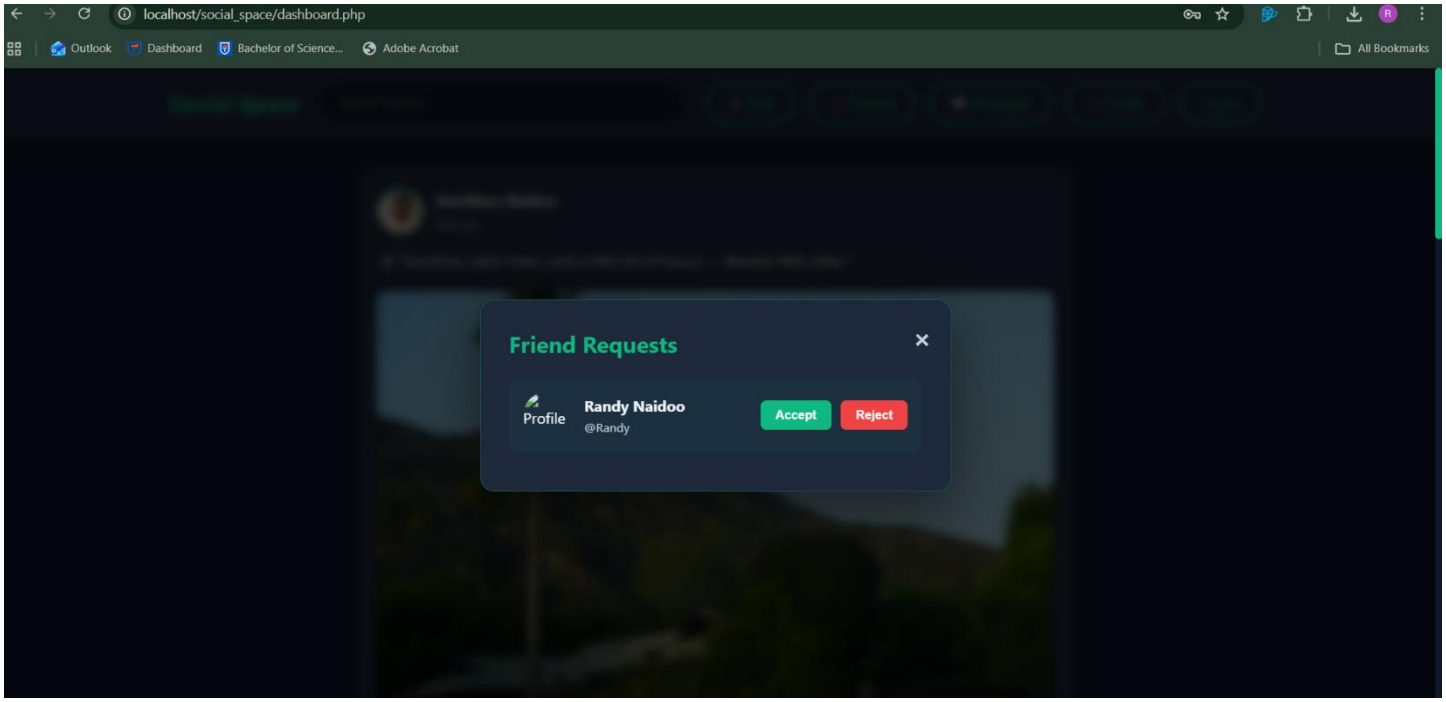

localhost/social_space/register.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Social Space
Join our community today
Randy Naidoo
Randy
randyn619@Gmail.com
Randy123$
Randy123$
Register
Already have an account? Login

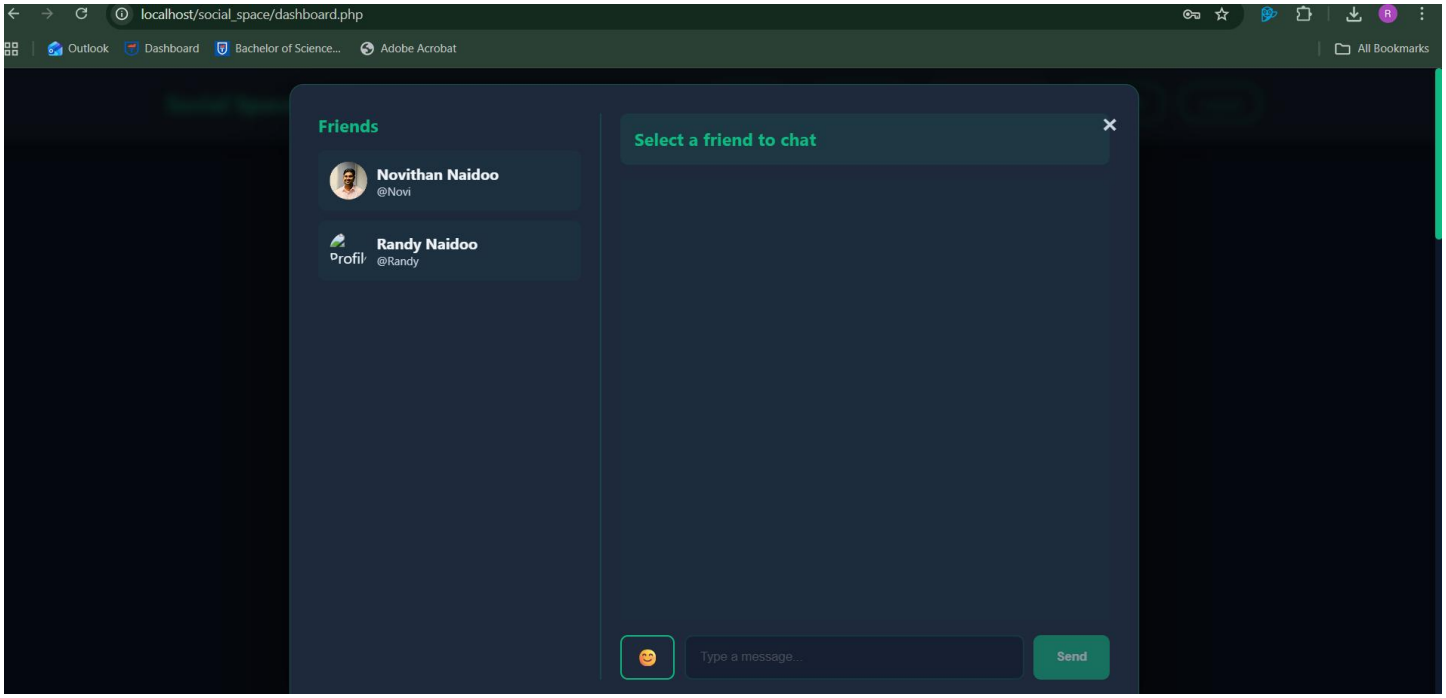

localhost/social_space/register.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Social Space
Join our community today
Registration successful! You can now login.
Full Name
Username
Email
Password
Confirm Password
Register
Already have an account? Login

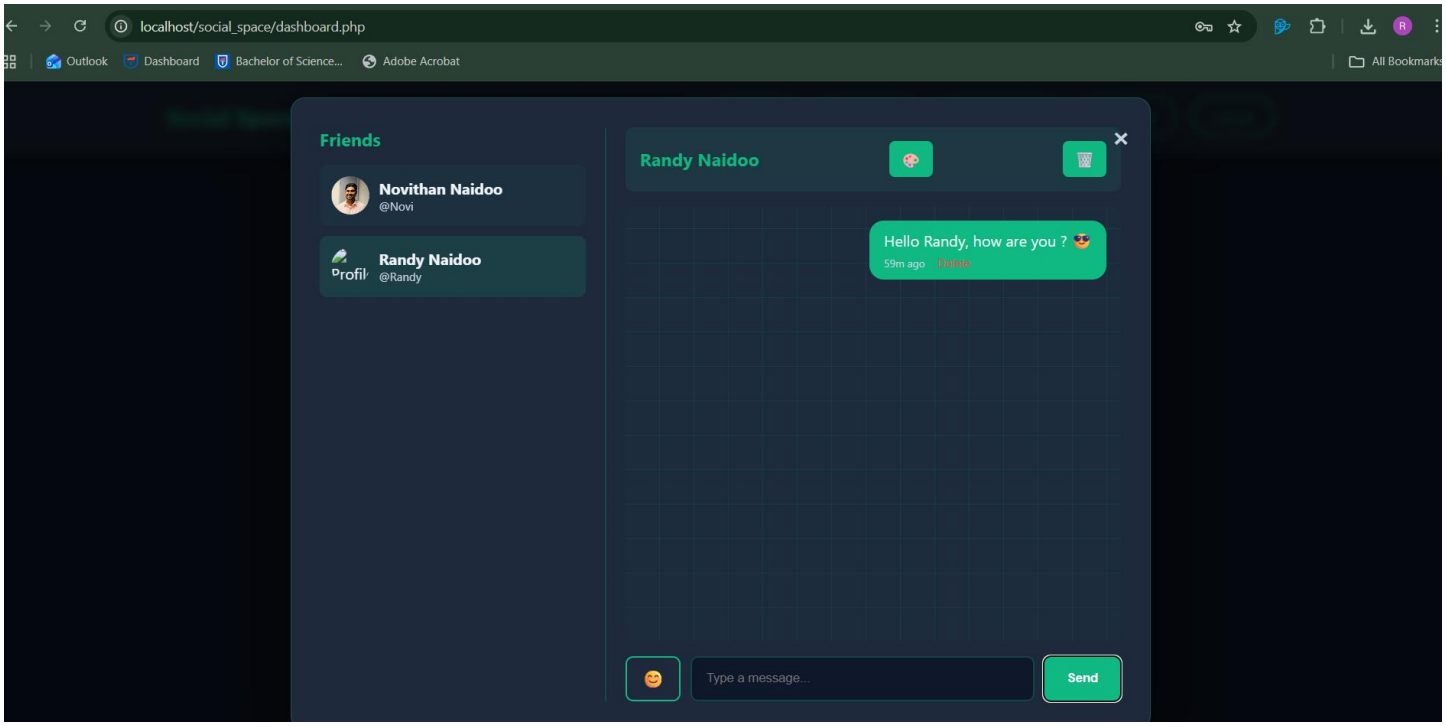

localhost/social_space/dashboard.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Social Space
jense
+ Post
Friends
Messages
Profile
Logout
Jensen Huang
@Jensen
Add Friend
Novithan Naidoo
55m ago
"Sunshine, palm trees, and a little bit of luxury — Beverly Hills vibes."
HOLLYWOOD

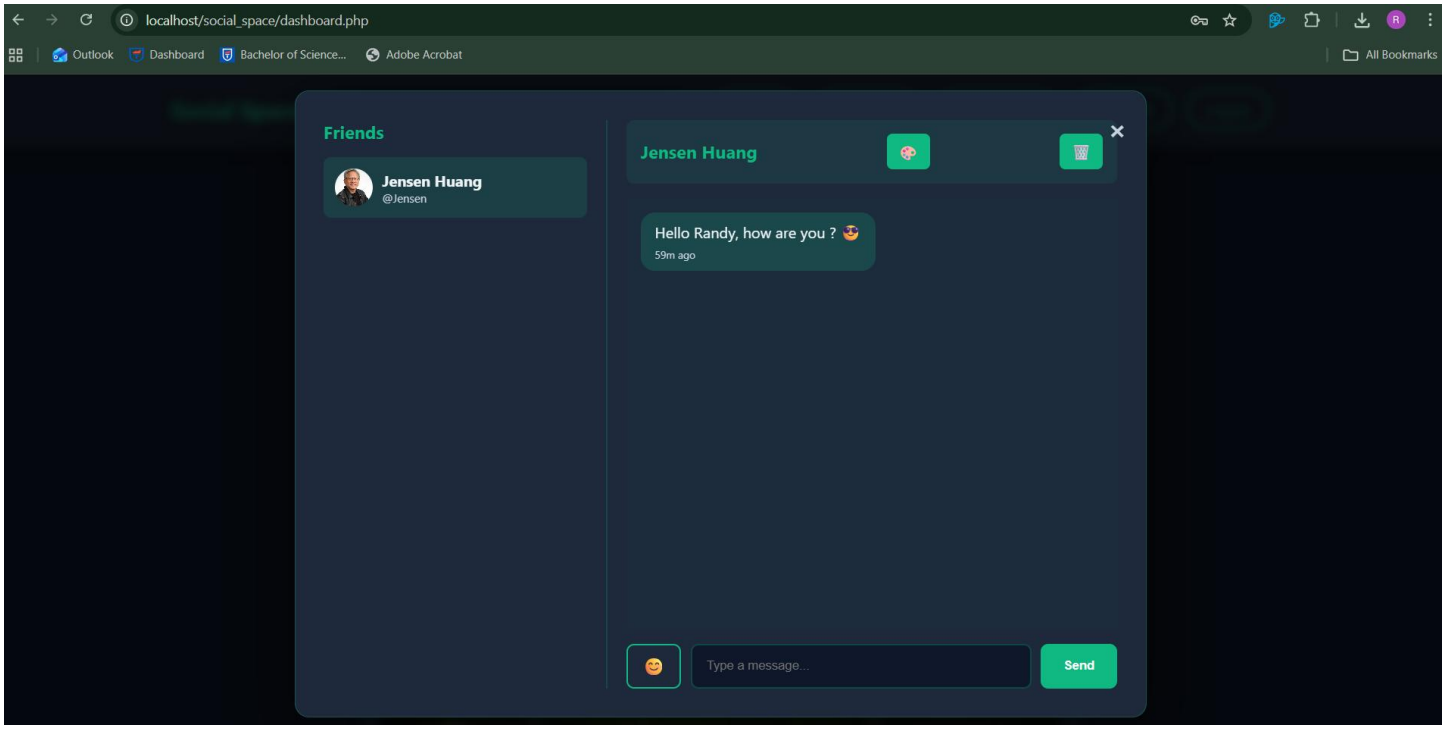

localhost/social_space/dashboard.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
localhost says
Friend request sent!
OK
Social Space
jense
Messages
Profile
Logout
Jensen Huang
@jensen
Novrthan Naidoo
55m ago
"Sunshine, palm trees, and a little bit of luxury — Beverly Hills vibes."
HOLLYWOOD

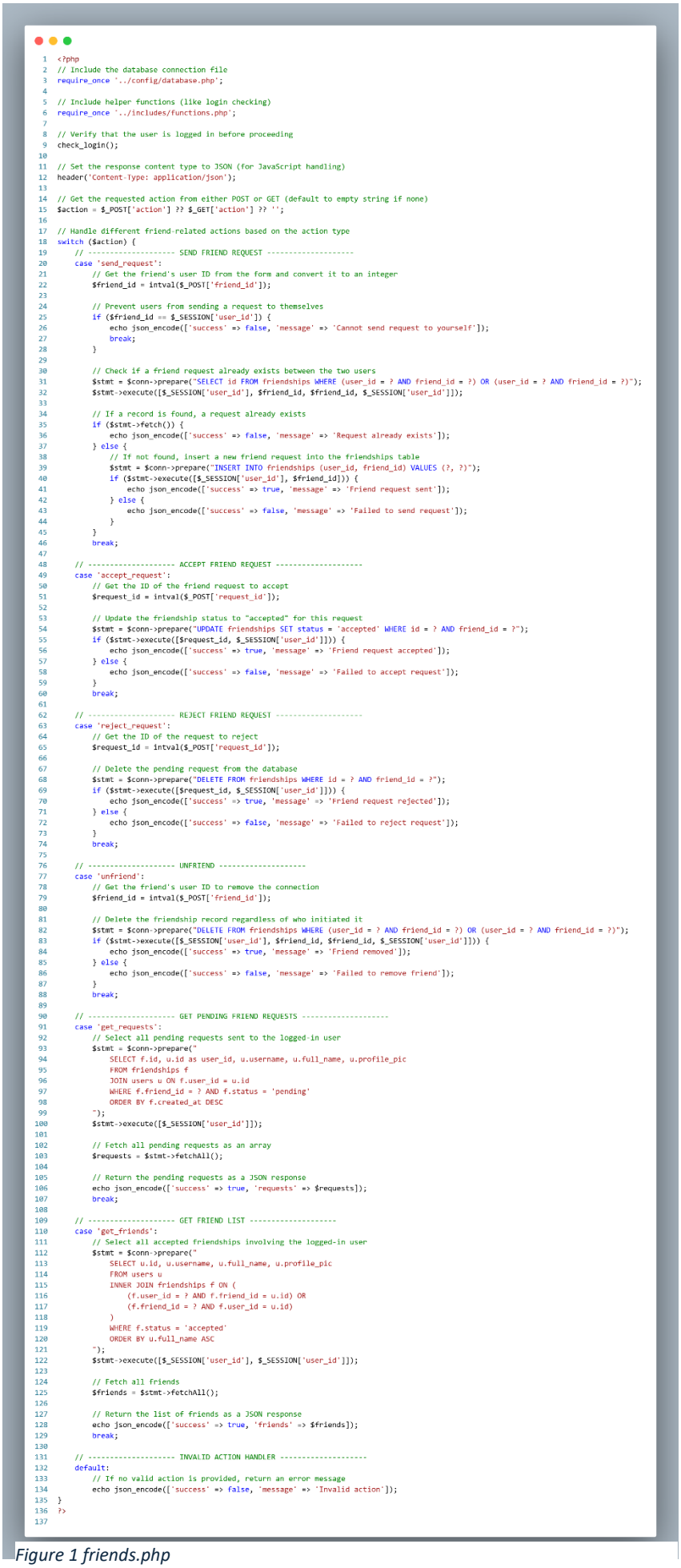

localhost/social_space/index.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Social Space
Connect with friends around the world
Invalid username or password
Username or Email
Password
Login
Don't have an account? Sign up


localhost/social_space/dashboard.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Friend Requests
Profile
Randy Naidoo
@Randy
Accept
Reject


localhost/social_space/dashboard.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Friends
Novithan Naidoo
@Novi
Randy Naidoo
@Randy
Select a friend to chat
Type a message...
Send


localhost/social_space/dashboard.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Friends
Novithan Naidoo
@Novi
Randy Naidoo
@Randy
Randy Naidoo
Hello Randy, how are you ?
59m ago
Delete
Type a message...
Send


localhost/social_space/dashboard.php
Outlook
Dashboard
Bachelor of Science...
Adobe Acrobat
All Bookmarks
Friends
Jensen Huang
@Jensen
Jensen Huang
Hello Randy, how are you ?
59m ago
Type a message...
Send



```php
<?php
// Include the database connection file
require_once '../config/database.php';

// Include helper functions (like login checking)
require_once '../includes/functions.php';

// Verify that the user is logged in before proceeding
check_login();

// Set the response content type to JSON (for JavaScript handling)
header('Content-Type: application/json');

// Get the requested action from either POST or GET (default to empty string if none)
$action = $_POST['action'] ?? $_GET['action'] ?? '';

// Handle different friend-related actions based on the action type
switch ($action) {
    // ------------------- SEND FRIEND REQUEST -------------------
    case 'send_request':
        // Get the friend's user ID from the form and convert it to an integer
        $friend_id = intval($_POST['friend_id']);

        // Prevent users from sending a request to themselves
        if ($friend_id == $_SESSION['user_id']) {
            echo json_encode(['success' => false, 'message' => 'Cannot send request to yourself']);
            break;
        }

        // Check if a friend request already exists between the two users
        $stmt = $conn->prepare("SELECT id FROM friendships WHERE (user_id = ? AND friend_id = ?) OR (user_id = ? AND friend_id = ?)");
        $stmt->execute([$_SESSION['user_id'], $friend_id, $friend_id, $_SESSION['user_id']]);

        // If a record is found, a request already exists
        if ($stmt->fetch()) {
            echo json_encode(['success' => false, 'message' => 'Request already exists']);
        } else {
            // If not found, insert a new friend request into the friendships table
            $stmt = $conn->prepare("INSERT INTO friendships (user_id, friend_id) VALUES (?, ?)");
            if ($stmt->execute([$_SESSION['user_id'], $friend_id])) {
                echo json_encode(['success' => true, 'message' => 'Friend request sent']);
            } else {
                echo json_encode(['success' => false, 'message' => 'Failed to send request']);
            }
        }
        break;

    // ------------------- ACCEPT FRIEND REQUEST -------------------
    case 'accept_request':
        // Get the ID of the friend request to accept
        $request_id = intval($_POST['request_id']);

        // Update the friendship status to "accepted" for this request
        $stmt = $conn->prepare("UPDATE friendships SET status = 'accepted' WHERE id = ? AND friend_id = ?");
        if ($stmt->execute([$request_id, $_SESSION['user_id']])) {
            echo json_encode(['success' => true, 'message' => 'Friend request accepted']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to accept request']);
        }
        break;

    // ------------------- REJECT FRIEND REQUEST -------------------
    case 'reject_request':
        // Get the ID of the request to reject
        $request_id = intval($_POST['request_id']);

        // Delete the pending request from the database
        $stmt = $conn->prepare("DELETE FROM friendships WHERE id = ? AND friend_id = ?");
        if ($stmt->execute([$request_id, $_SESSION['user_id']])) {
            echo json_encode(['success' => true, 'message' => 'Friend request rejected']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to reject request']);
        }
        break;

    // ------------------- UNFRIEND -------------------
    case 'unfriend':
        // Get the friend's user ID to remove the connection
        $friend_id = intval($_POST['friend_id']);

        // Delete the friendship record regardless of who initiated it
        $stmt = $conn->prepare("DELETE FROM friendships WHERE (user_id = ? AND friend_id = ?) OR (user_id = ? AND friend_id = ?)");
        if ($stmt->execute([$_SESSION['user_id'], $friend_id, $friend_id, $_SESSION['user_id']])) {
            echo json_encode(['success' => true, 'message' => 'Friend removed']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to remove friend']);
        }
        break;

    // ------------------- GET PENDING FRIEND REQUESTS -------------------
    case 'get_requests':
        // Select all pending requests sent to the logged-in user
        $stmt = $conn->prepare("
            SELECT f.id, u.id as user_id, u.username, u.full_name, u.profile_pic
            FROM friendships f
            JOIN users u ON f.user_id = u.id
            WHERE f.friend_id = ? AND f.status = 'pending'
            ORDER BY f.created_at DESC
        ");
        $stmt->execute([$_SESSION['user_id']]);

        // Fetch all pending requests as an array
        $requests = $stmt->fetchAll();

        // Return the pending requests as a JSON response
        echo json_encode(['success' => true, 'requests' => $requests]);
        break;

    // ------------------- GET FRIEND LIST -------------------
    case 'get_friends':
        // Select all accepted friendships involving the logged-in user
        $stmt = $conn->prepare("
            SELECT u.id, u.username, u.full_name, u.profile_pic
            FROM users u
            INNER JOIN friendships f ON (
                (f.user_id = ? AND f.friend_id = u.id) OR
                (f.friend_id = ? AND f.user_id = u.id)
            )
            WHERE f.status = 'accepted'
            ORDER BY u.full_name ASC
        ");
        $stmt->execute([$_SESSION['user_id'], $_SESSION['user_id']]);

        // Fetch all friends
        $friends = $stmt->fetchAll();

        // Return the list of friends as a JSON response
        echo json_encode(['success' => true, 'friends' => $friends]);
        break;

    // ------------------- INVALID ACTION HANDLER -------------------
    default:
        // If no valid action is provided, return an error message
        echo json_encode(['success' => false, 'message' => 'Invalid action']);
}
?>
```

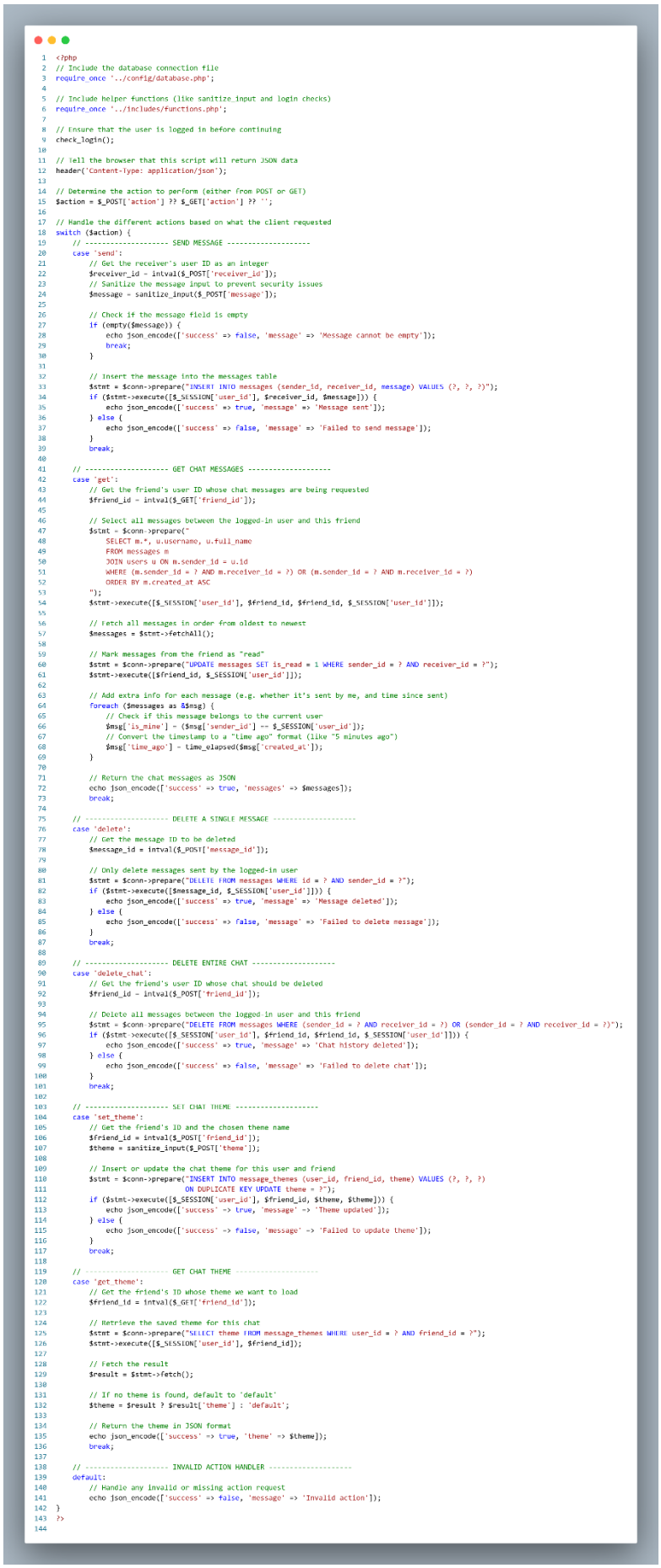
Figure 1 friends.php

```php
<?php
// Include the database connection file
require_once '../config/database.php';

// Include helper functions (like login checking)
require_once '../includes/functions.php';

// Verify that the user is logged in before proceeding
check_login();

// Set the response content type to JSON (for JavaScript handling)
header('Content-Type: application/json');

// Get the requested action from either POST or GET (default to empty string if none)
$action = $_POST['action'] ?? $_GET['action'] ?? '';

// Handle different friend-related actions based on the action type
switch ($action) {
   // ---------- SEND FRIEND REQUEST ----------
   case 'send_request':
      // Get the friend's user ID from the form and convert it to an integer
      $friend_id = intval($_POST['friend_id']);

      // Prevent users from sending a request to themselves
      if ($friend_id == $_SESSION['user_id']) {
         echo json_encode(['success' => false, 'message' => 'Cannot send request to yourself']);
         break;
      }

      // Check if a friend request already exists between the two users
      $stmt = $conn->prepare("SELECT id FROM friendships WHERE (user_id = ? AND friend_id = ?) OR (user_id =
? AND friend_id = ?)");
      $stmt->execute([$_SESSION['user_id'], $friend_id, $friend_id, $_SESSION['user_id']]);

      // If a record is found, a request already exists
      if ($stmt->fetch()) {
         echo json_encode(['success' => false, 'message' => 'Request already exists']);
      } else {
         // If not found, insert a new friend request into the friendships table
         $stmt = $conn->prepare("INSERT INTO friendships (user_id, friend_id) VALUES (?, ?)");
         if ($stmt->execute([$_SESSION['user_id'], $friend_id])) {
            echo json_encode(['success' => true, 'message' => 'Friend request sent']);
         } else {
            echo json_encode(['success' => false, 'message' => 'Failed to send request']);
         }
      }
      break;
```

(PHP.net, 2025)

```
// -------------------- GET PENDING FRIEND REQUESTS --------------------
   case 'get_requests':
      // Select all pending requests sent to the logged-in user
      $stmt = $conn->prepare("
         SELECT f.id, u.id as user_id, u.username, u.full_name, u.profile_pic
         FROM friendships f
         JOIN users u ON f.user_id = u.id
         WHERE f.friend_id = ? AND f.status = 'pending'
         ORDER BY f.created_at DESC
      ");
      $stmt->execute([$_SESSION['user_id']]);

      // Fetch all pending requests as an array
      $requests = $stmt->fetchAll();

      // Return the pending requests as a JSON response
      echo json_encode(['success' => true, 'requests' => $requests]);
      break;

   // -------------------- GET FRIEND LIST --------------------
   case 'get_friends':
      // Select all accepted friendships involving the logged-in user
      $stmt = $conn->prepare("
         SELECT u.id, u.username, u.full_name, u.profile_pic
         FROM users u
         INNER JOIN friendships f ON (
            (f.user_id = ? AND f.friend_id = u.id) OR
            (f.friend_id = ? AND f.user_id = u.id)
         )
         WHERE f.status = 'accepted'
         ORDER BY u.full_name ASC
      ");
      $stmt->execute([$_SESSION['user_id'], $_SESSION['user_id']]);

      // Fetch all friends
      $friends = $stmt->fetchAll();

      // Return the list of friends as a JSON response
      echo json_encode(['success' => true, 'friends' => $friends]);
      break;

   // -------------------- INVALID ACTION HANDLER --------------------
   default:
      // If no valid action is provided, return an error message
      echo json_encode(['success' => false, 'message' => 'Invalid action']);
}
?>
```

(PHP.net, 2025)



```php
<?php
// Include the database connection file
require_once '../config/database.php';

// Include helper functions (like sanitize_input and login checks)
require_once '../includes/functions.php';

// Ensure that the user is logged in before continuing
check_login();

// Tell the browser that this script will return JSON data
header('Content-Type: application/json');

// Determine the action to perform (either from POST or GET)
$action = $_POST['action'] ?? $_GET['action'] ?? '';

// Handle the different actions based on what the client requested
switch ($action) {
    // -------------------- SEND MESSAGE --------------------
    case 'send':
        // Get the receiver's user ID as an integer
        $receiver_id = intval($_POST['receiver_id']);
        // Sanitize the message input to prevent security issues
        $message = sanitize_input($_POST['message']);

        // Check if the message field is empty
        if (empty($message)) {
            echo json_encode(['success' => false, 'message' => 'Message cannot be empty']);
            break;
        }

        // Insert the message into the messages table
        $stmt = $conn->prepare("INSERT INTO messages (sender_id, receiver_id, message) VALUES (?, ?, ?)");
        if ($stmt->execute([$_SESSION['user_id'], $receiver_id, $message])) {
            echo json_encode(['success' => true, 'message' => 'Message sent']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to send message']);
        }
        break;

    // -------------------- GET CHAT MESSAGES --------------------
    case 'get':
        // Get the friend's user ID whose chat messages are being requested
        $friend_id = intval($_GET['friend_id']);

        // Select all messages between the logged-in user and this friend
        $stmt = $conn->prepare("
            SELECT m.*, u.username, u.full_name
            FROM messages m
            JOIN users u ON m.sender_id = u.id
            WHERE (m.sender_id = ? AND m.receiver_id = ?) OR (m.sender_id = ? AND m.receiver_id = ?)
            ORDER BY m.created_at ASC
        ");
        $stmt->execute([$_SESSION['user_id'], $friend_id, $friend_id, $_SESSION['user_id']]);

        // Fetch all messages in order from oldest to newest
        $messages = $stmt->fetchAll();

        // Mark messages from the friend as "read"
        $stmt = $conn->prepare("UPDATE messages SET is_read = 1 WHERE sender_id = ? AND receiver_id = ?");
        $stmt->execute([$friend_id, $_SESSION['user_id']]);

        // Add extra info for each message (e.g. whether it's sent by me, and time since sent)
        foreach ($messages as &$msg) {
            // Check if this message belongs to the current user
            $msg['is_mine'] = ($msg['sender_id'] == $_SESSION['user_id']);
            // Convert the timestamp to a "time ago" format (like "5 minutes ago")
            $msg['time_ago'] = time_elapsed($msg['created_at']);
        }

        // Return the chat messages as JSON
        echo json_encode(['success' => true, 'messages' => $messages]);
        break;

    // -------------------- DELETE A SINGLE MESSAGE --------------------
    case 'delete':
        // Get the message ID to be deleted
        $message_id = intval($_POST['message_id']);

        // Only delete messages sent by the logged-in user
        $stmt = $conn->prepare("DELETE FROM messages WHERE id = ? AND sender_id = ?");
        if ($stmt->execute([$message_id, $_SESSION['user_id']])) {
            echo json_encode(['success' => true, 'message' => 'Message deleted']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to delete message']);
        }
        break;

    // -------------------- DELETE ENTIRE CHAT --------------------
    case 'delete_chat':
        // Get the friend's user ID whose chat should be deleted
        $friend_id = intval($_POST['friend_id']);

        // Delete all messages between the logged-in user and this friend
        $stmt = $conn->prepare("DELETE FROM messages WHERE (sender_id = ? AND receiver_id = ?) OR (sender_id = ? AND receiver_id = ?)");
        if ($stmt->execute([$_SESSION['user_id'], $friend_id, $friend_id, $_SESSION['user_id']])) {
            echo json_encode(['success' => true, 'message' => 'Chat history deleted']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to delete chat']);
        }
        break;

    // -------------------- SET CHAT THEME --------------------
    case 'set_theme':
        // Get the friend's ID and the chosen theme name
        $friend_id = intval($_POST['friend_id']);
        $theme = sanitize_input($_POST['theme']);

        // Insert or update the chat theme for this user and friend
        $stmt = $conn->prepare("INSERT INTO message_themes (user_id, friend_id, theme) VALUES (?, ?, ?)
                                ON DUPLICATE KEY UPDATE theme = ?");
        if ($stmt->execute([$_SESSION['user_id'], $friend_id, $theme, $theme])) {
            echo json_encode(['success' => true, 'message' => 'Theme updated']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to update theme']);
        }
        break;

    // -------------------- GET CHAT THEME --------------------
    case 'get_theme':
        // Get the friend's ID whose theme we want to load
        $friend_id = intval($_GET['friend_id']);

        // Retrieve the saved theme for this chat
        $stmt = $conn->prepare("SELECT theme FROM message_themes WHERE user_id = ? AND friend_id = ?");
        $stmt->execute([$_SESSION['user_id'], $friend_id]);

        // Fetch the result
        $result = $stmt->fetch();

        // If no theme is found, default to 'default'
        $theme = $result ? $result['theme'] : 'default';

        // Return the theme in JSON format
        echo json_encode(['success' => true, 'theme' => $theme]);
        break;

    // -------------------- INVALID ACTION HANDLER --------------------
    default:
        // Handle any invalid or missing action request
        echo json_encode(['success' => false, 'message' => 'Invalid action']);
}
?>
```

Figure 2messages.php

```php
<?php
// Include the database connection file
require_once '../config/database.php';

// Include helper functions (like sanitize_input and login checks)
require_once '../includes/functions.php';

// Ensure that the user is logged in before continuing
check_login();

// Tell the browser that this script will return JSON data
header('Content-Type: application/json');

// Determine the action to perform (either from POST or GET)
$action = $_POST['action'] ?? $_GET['action'] ?? '';

// Handle the different actions based on what the client requested
switch ($action) {
    // ---------- SEND MESSAGE ----------
    case 'send':
        // Get the receiver's user ID as an integer
        $receiver_id = intval($_POST['receiver_id']);
        // Sanitize the message input to prevent security issues
        $message = sanitize_input($_POST['message']);

        // Check if the message field is empty
        if (empty($message)) {
            echo json_encode(['success' => false, 'message' => 'Message cannot be empty']);
            break;
        }

        // Insert the message into the messages table
        $stmt = $conn->prepare("INSERT INTO messages (sender_id, receiver_id, message) VALUES (?, ?, ?)");
        if ($stmt->execute([$_SESSION['user_id'], $receiver_id, $message])) {
            echo json_encode(['success' => true, 'message' => 'Message sent']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to send message']);
        }
        break;

// ---------- GET CHAT MESSAGES ----------
    case 'get':
        // Get the friend's user ID whose chat messages are being requested
        $friend_id = intval($_GET['friend_id']);

        // Select all messages between the logged-in user and this friend
        $stmt = $conn->prepare("
            SELECT m.*, u.username, u.full_name
            FROM messages m
            JOIN users u ON m.sender_id = u.id
            WHERE (m.sender_id = ? AND m.receiver_id = ?) OR (m.sender_id = ? AND m.receiver_id = ?)
            ORDER BY m.created_at ASC
        ");
        $stmt->execute([$_SESSION['user_id'], $friend_id, $friend_id, $_SESSION['user_id']]);
```

```
        // Fetch all messages in order from oldest to newest
        $messages = $stmt->fetchAll();

        // Mark messages from the friend as "read"
        $stmt = $conn->prepare("UPDATE messages SET is_read = 1 WHERE sender_id = ? AND receiver_id = ?");
        $stmt->execute([$friend_id, $_SESSION['user_id']]);

        // Add extra info for each message (e.g. whether it's sent by me, and time since sent)
        foreach ($messages as &$msg) {
            // Check if this message belongs to the current user
            $msg['is_mine'] = ($msg['sender_id'] == $_SESSION['user_id']);
            // Convert the timestamp to a "time ago" format (like "5 minutes ago")
            $msg['time_ago'] = time_elapsed($msg['created_at']);
        }

        // Return the chat messages as JSON
        echo json_encode(['success' => true, 'messages' => $messages]);
        break;

// ---------- DELETE A SINGLE MESSAGE ----------
    case 'delete':
        // Get the message ID to be deleted
        $message_id = intval($_POST['message_id']);

        // Only delete messages sent by the logged-in user
        $stmt = $conn->prepare("DELETE FROM messages WHERE id = ? AND sender_id = ?");
        if ($stmt->execute([$message_id, $_SESSION['user_id']])) {
            echo json_encode(['success' => true, 'message' => 'Message deleted']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to delete message']);
        }
        break;

// ---------- DELETE ENTIRE CHAT ----------
    case 'delete_chat':
        // Get the friend's user ID whose chat should be deleted
        $friend_id = intval($_POST['friend_id']);

        // Delete all messages between the logged-in user and this friend
        $stmt = $conn->prepare("DELETE FROM messages WHERE (sender_id = ? AND receiver_id = ?) OR (sender_id
= ? AND receiver_id = ?)");
        if ($stmt->execute([$_SESSION['user_id'], $friend_id, $friend_id, $_SESSION['user_id']])) {
            echo json_encode(['success' => true, 'message' => 'Chat history deleted']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to delete chat']);
        }
        break;
```

(PHP.net, 2025)

```
// ---------- SET CHAT THEME ----------
   case 'set_theme':
      // Get the friend's ID and the chosen theme name
      $friend_id = intval($_POST['friend_id']);
      $theme = sanitize_input($_POST['theme']);

      // Insert or update the chat theme for this user and friend
      $stmt = $conn->prepare("INSERT INTO message_themes (user_id, friend_id, theme) VALUES (?, ?, ?)
                        ON DUPLICATE KEY UPDATE theme = ?");
      if ($stmt->execute([$_SESSION['user_id'], $friend_id, $theme, $theme])) {
         echo json_encode(['success' => true, 'message' => 'Theme updated']);
      } else {
         echo json_encode(['success' => false, 'message' => 'Failed to update theme']);
      }
      break;

   // ---------- GET CHAT THEME ----------
   case 'get_theme':
      // Get the friend's ID whose theme we want to load
      $friend_id = intval($_GET['friend_id']);

      // Retrieve the saved theme for this chat
      $stmt = $conn->prepare("SELECT theme FROM message_themes WHERE user_id = ? AND friend_id = ?");
      $stmt->execute([$_SESSION['user_id'], $friend_id]);

      // Fetch the result
      $result = $stmt->fetch();

      // If no theme is found, default to 'default'
      $theme = $result ? $result['theme'] : 'default';

      // Return the theme in JSON format
      echo json_encode(['success' => true, 'theme' => $theme]);
      break;

   // ---------- INVALID ACTION HANDLER ----------
   default:
      // Handle any invalid or missing action request
      echo json_encode(['success' => false, 'message' => 'Invalid action']);
}
?>
```

```php
<?php
// Include the database connection file
require_once '../config/database.php';

// Include helper functions (like sanitize_input and check_login)
require_once '../includes/functions.php';

// Ensure the user is logged in before using this script
check_login();

// Set the content type to JSON since responses will be in JSON format
header('Content-Type: application/json');

// Determine what action to perform based on the request (POST or GET)
$action = $_POST['action'] ?? $_GET['action'] ?? '';

// Handle different post-related actions
switch ($action) {

    // -------------------- CREATE A POST --------------------
    case 'create':
        // Sanitize the post content
        $content = sanitize_input($_POST['content']);
        // Initialize image variable as null
        $image = null;

        // Check if an image was uploaded and there are no upload errors
        if (isset($_FILES['image']) && $_FILES['image']['error'] == 0) {
            // Allowed image file types
            $allowed = ['jpg', 'jpeg', 'png', 'gif'];
            // Get the original uploaded file name
            $filename = $_FILES['image']['name'];
            // Extract the file extension
            $ext = strtolower(pathinfo($filename, PATHINFO_EXTENSION));

            // Validate the file extension
            if (in_array($ext, $allowed)) {
                // Generate a unique filename for the uploaded image
                $new_filename = uniqid() . '.' . $ext;
                // Define the upload path
                $upload_path = '../uploads/posts/' . $new_filename;

                // Move the uploaded file to the uploads directory
                if (move_uploaded_file($_FILES['image']['tmp_name'], $upload_path)) {
                    // Store the image name to save in the database
                    $image = $new_filename;
                }
            }
        }

        // Insert the new post into the database
        $stmt = $conn->prepare("INSERT INTO posts (user_id, content, image) VALUES (?, ?, ?)");
        if ($stmt->execute([$_SESSION['user_id'], $content, $image])) {
            echo json_encode(['success' => true, 'message' => 'Post created successfully']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to create post']);
        }
        break;

    // -------------------- DELETE A POST --------------------
    case 'delete':
        // Get the post ID to delete
        $post_id = intval($_POST['post_id']);

        // Fetch the post details to check ownership and get the image name
        $stmt = $conn->prepare("SELECT user_id, image FROM posts WHERE id = ?");
        $stmt->execute([$post_id]);
        $post = $stmt->fetch();

        // Check if the post exists and belongs to the logged-in user
        if ($post && $post['user_id'] == $_SESSION['user_id']) {
            // Delete the post image from the server if it exists
            if ($post['image'] && file_exists('../uploads/posts/' . $post['image'])) {
                unlink('../uploads/posts/' . $post['image']);
            }

            // Delete the post record from the database
            $stmt = $conn->prepare("DELETE FROM posts WHERE id = ?");
            if ($stmt->execute([$post_id])) {
                echo json_encode(['success' => true, 'message' => 'Post deleted']);
            } else {
                echo json_encode(['success' => false, 'message' => 'Failed to delete post']);
            }
        } else {
            // Unauthorized attempt to delete a post
            echo json_encode(['success' => false, 'message' => 'Unauthorized']);
        }
        break;

    // -------------------- LIKE OR UNLIKE A POST --------------------
    case 'like':
        // Get the post ID to like/unlike
        $post_id = intval($_POST['post_id']);

        // Check if the user already liked this post
        $stmt = $conn->prepare("SELECT id FROM likes WHERE post_id = ? AND user_id = ?");
        $stmt->execute([$post_id, $_SESSION['user_id']]);

        // If already liked, remove the like (unlike)
        if ($stmt->fetch()) {
            $stmt = $conn->prepare("DELETE FROM likes WHERE post_id = ? AND user_id = ?");
            $stmt->execute([$post_id, $_SESSION['user_id']]);
            $liked = false;
        } else {
            // Otherwise, add a new like
            $stmt = $conn->prepare("INSERT INTO likes (post_id, user_id) VALUES (?, ?)");
            $stmt->execute([$post_id, $_SESSION['user_id']]);
            $liked = true;
        }

        // Get the updated number of likes for the post
        $stmt = $conn->prepare("SELECT COUNT(*) as count FROM likes WHERE post_id = ?");
        $stmt->execute([$post_id]);
        $result = $stmt->fetch();

        // Return the like status and count
        echo json_encode(['success' => true, 'liked' => $liked, 'count' => $result['count']]);
        break;

    // -------------------- ADD A COMMENT --------------------
    case 'comment':
        // Get the post ID and the comment text
        $post_id = intval($_POST['post_id']);
        $comment = sanitize_input($_POST['comment']);

        // Insert the comment into the comments table
        $stmt = $conn->prepare("INSERT INTO comments (post_id, user_id, comment) VALUES (?, ?, ?)");
        if ($stmt->execute([$post_id, $_SESSION['user_id'], $comment])) {
            echo json_encode(['success' => true, 'message' => 'Comment added']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to add comment']);
        }
        break;

    // -------------------- FETCH COMMENTS FOR A POST --------------------
    case 'get_comments':
        // Get the post ID to fetch comments for
        $post_id = intval($_GET['post_id']);

        // Retrieve all comments for this post with user details
        $stmt = $conn->prepare("
            SELECT c.*, u.username, u.full_name, u.profile_pic
            FROM comments c
            JOIN users u ON c.user_id = u.id
            WHERE c.post_id = ?
            ORDER BY c.created_at ASC
        ");
        $stmt->execute([$post_id]);
        $comments = $stmt->fetchAll();

        // Add time and ownership details to each comment
        foreach ($comments as &$comment) {
            $comment['time_ago'] = time_elapsed($comment['created_at']); // Format time (e.g., "2h ago")
            $comment['can_delete'] = ($comment['user_id'] == $_SESSION['user_id']); // Check if user can delete it
        }

        // Return all comments as a JSON response
        echo json_encode(['success' => true, 'comments' => $comments]);
        break;

    // -------------------- DELETE A COMMENT --------------------
    case 'delete_comment':
        // Get the comment ID to delete
        $comment_id = intval($_POST['comment_id']);

        // Check who owns the comment
        $stmt = $conn->prepare("SELECT user_id FROM comments WHERE id = ?");
        $stmt->execute([$comment_id]);
        $comment = $stmt->fetch();

        // Allow deletion only if the logged-in user is the comment owner
        if ($comment && $comment['user_id'] == $_SESSION['user_id']) {
            $stmt = $conn->prepare("DELETE FROM comments WHERE id = ?");
            if ($stmt->execute([$comment_id])) {
                echo json_encode(['success' => true, 'message' => 'Comment deleted']);
            } else {
                echo json_encode(['success' => false, 'message' => 'Failed to delete comment']);
            }
        } else {
            // Unauthorized attempt
            echo json_encode(['success' => false, 'message' => 'Unauthorized']);
        }
        break;

    // -------------------- SHARE A POST --------------------
    case 'share':
        // Get the post ID to share
        $post_id = intval($_POST['post_id']);

        // Insert a new record in the shares table
        $stmt = $conn->prepare("INSERT INTO shares (post_id, user_id) VALUES (?, ?)");
        if ($stmt->execute([$post_id, $_SESSION['user_id']])) {
            echo json_encode(['success' => true, 'message' => 'Post shared']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to share post']);
        }
        break;

    // -------------------- INVALID ACTION HANDLER --------------------
    default:
        // Handle invalid or unknown action requests
        echo json_encode(['success' => false, 'message' => 'Invalid action']);
}
?>
```

Figure 3posts.php

```php
<?php
// Include the database connection file
require_once '../config/database.php';

// Include helper functions (like sanitize_input and check_login)
require_once '../includes/functions.php';

// Ensure the user is logged in before using this script
check_login();

// Set the content type to JSON since responses will be in JSON format
header('Content-Type: application/json');

// Determine what action to perform based on the request (POST or GET)
$action = $_POST['action'] ?? $_GET['action'] ?? '';

// Handle different post-related actions
switch ($action) {

   // ---------- CREATE A POST ----------
   case 'create':
      // Sanitize the post content
      $content = sanitize_input($_POST['content']);
      // Initialize image variable as null
      $image = null;

      // Check if an image was uploaded and there are no upload errors
      if (isset($_FILES['image']) && $_FILES['image']['error'] == 0) {
         // Allowed image file types
         $allowed = ['jpg', 'jpeg', 'png', 'gif'];
         // Get the original uploaded file name
         $filename = $_FILES['image']['name'];
         // Extract the file extension
         $ext = strtolower(pathinfo($filename, PATHINFO_EXTENSION));

         // Validate the file extension
         if (in_array($ext, $allowed)) {
            // Generate a unique filename for the uploaded image
            $new_filename = uniqid() . '.' . $ext;
            // Define the upload path
            $upload_path = '../uploads/posts/' . $new_filename;

            // Move the uploaded file to the uploads directory
            if (move_uploaded_file($_FILES['image']['tmp_name'], $upload_path)) {
               // Store the image name to save in the database
               $image = $new_filename;
            }
         }
      }
```

(PHP.net, 2025)

```
// Insert the new post into the database
      $stmt = $conn->prepare("INSERT INTO posts (user_id, content, image) VALUES (?, ?, ?)");
      if ($stmt->execute([$_SESSION['user_id'], $content, $image])) {
         echo json_encode(['success' => true, 'message' => 'Post created successfully']);
      } else {
         echo json_encode(['success' => false, 'message' => 'Failed to create post']);
      }
      break;

   // ---------- DELETE A POST ----------
   case 'delete':
      // Get the post ID to delete
      $post_id = intval($_POST['post_id']);

      // Fetch the post details to check ownership and get the image name
      $stmt = $conn->prepare("SELECT user_id, image FROM posts WHERE id = ?");
      $stmt->execute([$post_id]);
      $post = $stmt->fetch();

      // Check if the post exists and belongs to the logged-in user
      if ($post && $post['user_id'] == $_SESSION['user_id']) {
         // Delete the post image from the server if it exists
         if ($post['image'] && file_exists('../uploads/posts/' . $post['image'])) {
            unlink('../uploads/posts/' . $post['image']);
         }

         // Delete the post record from the database
         $stmt = $conn->prepare("DELETE FROM posts WHERE id = ?");
         if ($stmt->execute([$post_id])) {
            echo json_encode(['success' => true, 'message' => 'Post deleted']);
         } else {
            echo json_encode(['success' => false, 'message' => 'Failed to delete post']);
         }
      } else {
         // Unauthorized attempt to delete a post
         echo json_encode(['success' => false, 'message' => 'Unauthorized']);
      }
      break;


   // ---------- LIKE OR UNLIKE A POST ----------
   case 'like':
      // Get the post ID to like/unlike
      $post_id = intval($_POST['post_id']);

      // Check if the user already liked this post
      $stmt = $conn->prepare("SELECT id FROM likes WHERE post_id = ? AND user_id = ?");
      $stmt->execute([$post_id, $_SESSION['user_id']]);
```

```
// If already liked, remove the like (unlike)
      if ($stmt->fetch()) {
         $stmt = $conn->prepare("DELETE FROM likes WHERE post_id = ? AND user_id = ?");
         $stmt->execute([$post_id, $_SESSION['user_id']]);
         $liked = false;
      } else {
         // Otherwise, add a new like
         $stmt = $conn->prepare("INSERT INTO likes (post_id, user_id) VALUES (?, ?)");
         $stmt->execute([$post_id, $_SESSION['user_id']]);
         $liked = true;
      }

      // Get the updated number of likes for the post
      $stmt = $conn->prepare("SELECT COUNT(*) as count FROM likes WHERE post_id = ?");
      $stmt->execute([$post_id]);
      $result = $stmt->fetch();

      // Return the like status and count
      echo json_encode(['success' => true, 'liked' => $liked, 'count' => $result['count']]);
      break;

   // ---------- ADD A COMMENT ----------
   case 'comment':
      // Get the post ID and the comment text
      $post_id = intval($_POST['post_id']);
      $comment = sanitize_input($_POST['comment']);

      // Insert the comment into the comments table
      $stmt = $conn->prepare("INSERT INTO comments (post_id, user_id, comment) VALUES (?, ?, ?)");
      if ($stmt->execute([$post_id, $_SESSION['user_id'], $comment])) {
         echo json_encode(['success' => true, 'message' => 'Comment added']);
      } else {
         echo json_encode(['success' => false, 'message' => 'Failed to add comment']);
      }
      break;

   // ---------- FETCH COMMENTS FOR A POST ----------
   case 'get_comments':
      // Get the post ID to fetch comments for
      $post_id = intval($_GET['post_id']);

  // Retrieve all comments for this post with user details
      $stmt = $conn->prepare("
         SELECT c.*, u.username, u.full_name, u.profile_pic
         FROM comments c
         JOIN users u ON c.user_id = u.id
         WHERE c.post_id = ?
         ORDER BY c.created_at ASC
      ");
      $stmt->execute([$post_id]);
      $comments = $stmt->fetchAll();
```

```
// Add time and ownership details to each comment
    foreach ($comments as &$comment) {
        $comment['time_ago'] = time_elapsed($comment['created_at']); // Format time (e.g., "2h ago")
        $comment['can_delete'] = ($comment['user_id'] == $_SESSION['user_id']); // Check if user can delete it
    }

    // Return all comments as a JSON response
    echo json_encode(['success' => true, 'comments' => $comments]);
    break;

  // ---------- DELETE A COMMENT ----------
  case 'delete_comment':
    // Get the comment ID to delete
    $comment_id = intval($_POST['comment_id']);

    // Check who owns the comment
    $stmt = $conn->prepare("SELECT user_id FROM comments WHERE id = ?");
    $stmt->execute([$comment_id]);
    $comment = $stmt->fetch();

    // Allow deletion only if the logged-in user is the comment owner
    if ($comment && $comment['user_id'] == $_SESSION['user_id']) {
        $stmt = $conn->prepare("DELETE FROM comments WHERE id = ?");
        if ($stmt->execute([$comment_id])) {
            echo json_encode(['success' => true, 'message' => 'Comment deleted']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Failed to delete comment']);
        }
    } else {
        // Unauthorized attempt
        echo json_encode(['success' => false, 'message' => 'Unauthorized']);
    }
    break;

// ---------- SHARE A POST ----------
  case 'share':
    // Get the post ID to share
    $post_id = intval($_POST['post_id']);

    // Insert a new record in the shares table
    $stmt = $conn->prepare("INSERT INTO shares (post_id, user_id) VALUES (?, ?)");
    if ($stmt->execute([$post_id, $_SESSION['user_id']])) {
        echo json_encode(['success' => true, 'message' => 'Post shared']);
    } else {
        echo json_encode(['success' => false, 'message' => 'Failed to share post']);
    }
    break;

  // ---------- INVALID ACTION HANDLER ----------
  default:
    // Handle invalid or unknown action requests
    echo json_encode(['success' => false, 'message' => 'Invalid action']);
}
?>
```

```php
<?php
require_once '../config/database.php';
require_once '../includes/functions.php';

// Ensure the user is logged in before proceeding
check_login();

// Return data as JSON
header('Content-Type: application/json');

// Get the action from POST request
$action = $_POST['action'] ?? '';

// Handle profile update action
if ($action == 'update') {
    try {
        // Sanitize user input for security
        $full_name = sanitize_input($_POST['full_name']);
        $bio = sanitize_input($_POST['bio']);
        $profile_pic = null;

        // ------------------------------
        // PROFILE PICTURE UPLOAD SECTION
        // ------------------------------
        if (isset($_FILES['profile_pic']) && $_FILES['profile_pic']['error'] == 0) {
            $allowed = ['jpg', 'jpeg', 'png', 'gif']; // Allowed file types
            $filename = $_FILES['profile_pic']['name'];
            $ext = strtolower(pathinfo($filename, PATHINFO_EXTENSION)); // Get file extension

            // Check if file type is valid
            if (in_array($ext, $allowed)) {
                // Create the upload directory if it doesn't exist
                if (!file_exists('../uploads/profiles/')) {
                    mkdir('../uploads/profiles/', 0777, true);
                }

                // Generate a unique filename to avoid duplicates
                $new_filename = 'profile_' . $_SESSION['user_id'] . '_' . time() . '.' . $ext;
                $upload_path = '../uploads/profiles/' . $new_filename;

                // Move uploaded file to the destination folder
                if (move_uploaded_file($_FILES['profile_pic']['tmp_name'], $upload_path)) {

                    // Fetch current profile picture to delete the old one (if not default)
                    $stmt = $conn->prepare("SELECT profile_pic FROM users WHERE id = ?");
                    $stmt->execute([$_SESSION['user_id']]);
                    $old_pic = $stmt->fetch()['profile_pic'];

                    // Delete old image file if it exists and isn’t default
                    if ($old_pic != 'default.jpg' && file_exists('../uploads/profiles/' . $old_pic)) {
                        unlink('../uploads/profiles/' . $old_pic);
                    }

                    // Store new profile image filename
                    $profile_pic = $new_filename;
                } else {
                    // File upload failed (permissions or path issue)
                    echo json_encode(['success' => false, 'message' => 'Failed to upload image. Check folder permissions.']);
                    exit;
                }
            } else {
                // Invalid file extension
                echo json_encode(['success' => false, 'message' => 'Invalid file type. Only JPG, PNG, GIF allowed.']);
                exit;
            }
        }

        // --------------------------
        // UPDATE USER PROFILE INFO
        // --------------------------
        if ($profile_pic) {
            // Update including profile picture
            $stmt = $conn->prepare("UPDATE users SET full_name = ?, bio = ?, profile_pic = ? WHERE id = ?");
            $result = $stmt->execute([$full_name, $bio, $profile_pic, $_SESSION['user_id']]);
        } else {
            // Update without profile picture
            $stmt = $conn->prepare("UPDATE users SET full_name = ?, bio = ? WHERE id = ?");
            $result = $stmt->execute([$full_name, $bio, $_SESSION['user_id']]);
        }

        // --------------------------
        // SEND JSON RESPONSE
        // --------------------------
        if ($result) {
            echo json_encode(['success' => true, 'message' => 'Profile updated successfully']);
        } else {
            echo json_encode(['success' => false, 'message' => 'Database update failed']);
        }

    } catch (Exception $e) {
        // Catch any PHP or database error
        echo json_encode(['success' => false, 'message' => 'Error: ' . $e->getMessage()]);
    }
} else {
    // Invalid action was passed
    echo json_encode(['success' => false, 'message' => 'Invalid action']);
}
?> -->

```

Figure 4profile.php

```
<?php
require_once '../config/database.php';
require_once '../includes/functions.php';

// Ensure the user is logged in before proceeding
check_login();

// Return data as JSON
header('Content-Type: application/json');

// Get the action from POST request
$action = $_POST['action'] ?? '';

// Handle profile update action
if ($action == 'update') {
   try {
      // Sanitize user input for security
      $full_name = sanitize_input($_POST['full_name']);
      $bio = sanitize_input($_POST['bio']);
      $profile_pic = null;

// ------------------------------
      // PROFILE PICTURE UPLOAD SECTION
      // ------------------------------
      if (isset($_FILES['profile_pic']) && $_FILES['profile_pic']['error'] == 0) {
         $allowed = ['jpg', 'jpeg', 'png', 'gif']; // Allowed file types
         $filename = $_FILES['profile_pic']['name'];
         $ext = strtolower(pathinfo($filename, PATHINFO_EXTENSION)); // Get file extension

         // Check if file type is valid
         if (in_array($ext, $allowed)) {
            // Create the upload directory if it doesn't exist
            if (!file_exists('../uploads/profiles/')) {
               mkdir('../uploads/profiles/', 0777, true);
            }

// Generate a unique filename to avoid duplicates
            $new_filename = 'profile_' . $_SESSION['user_id'] . '_' . time() . '.' . $ext;
            $upload_path = '../uploads/profiles/' . $new_filename;

            // Move uploaded file to the destination folder
            if (move_uploaded_file($_FILES['profile_pic']['tmp_name'], $upload_path)) {

               // Fetch current profile picture to delete the old one (if not default)
               $stmt = $conn->prepare("SELECT profile_pic FROM users WHERE id = ?");
               $stmt->execute([$_SESSION['user_id']]);
               $old_pic = $stmt->fetch()['profile_pic'];
```

(PHP.net, 2025)

```
// Delete old image file if it exists and isn't default
                if ($old_pic != 'default.jpg' && file_exists('../uploads/profiles/' . $old_pic)) {
                    unlink('../uploads/profiles/' . $old_pic);
                }

                // Store new profile image filename
                $profile_pic = $new_filename;
            } else {
                // File upload failed (permissions or path issue)
                echo json_encode(['success' => false, 'message' => 'Failed to upload image. Check folder
permissions.']);
                exit;
            }
        } else {
            // Invalid file extension
            echo json_encode(['success' => false, 'message' => 'Invalid file type. Only JPG, PNG, GIF allowed.']);
            exit;
        }
    }
// --------------------------
    // UPDATE USER PROFILE INFO
    // --------------------------
    if ($profile_pic) {
        // Update including profile picture
        $stmt = $conn->prepare("UPDATE users SET full_name = ?, bio = ?, profile_pic = ? WHERE id = ?");
        $result = $stmt->execute([$full_name, $bio, $profile_pic, $_SESSION['user_id']]);
    } else {
        // Update without profile picture
        $stmt = $conn->prepare("UPDATE users SET full_name = ?, bio = ? WHERE id = ?");
        $result = $stmt->execute([$full_name, $bio, $_SESSION['user_id']]);
    }

    // --------------------------
    // SEND JSON RESPONSE
    // --------------------------
    if ($result) {
        echo json_encode(['success' => true, 'message' => 'Profile updated successfully']);
    } else {
        echo json_encode(['success' => false, 'message' => 'Database update failed']);
    }
} catch (Exception $e) {
    // Catch any PHP or database error
    echo json_encode(['success' => false, 'message' => 'Error: ' . $e->getMessage()]);
  }
} else {
  // Invalid action was passed
  echo json_encode(['success' => false, 'message' => 'Invalid action']);
}
?>
```

```php
<?php
// Include the database connection file
require_once '../config/database.php';

// Include helper functions
require_once '../includes/functions.php';

// Check if the user is logged in
check_login();

// Set response type to JSON (for AJAX calls)
header('Content-Type: application/json');

// Get the search query from the URL and sanitize it
$query = sanitize_input($_GET['query'] ?? '');

// If the query is shorter than 2 characters, return an empty result
if (strlen($query) < 2) {
    echo json_encode(['success' => true, 'users' => []]);
    exit();
}

// Add wildcard symbols (%) for SQL LIKE search
$search = "%{$query}%";

// Prepare SQL statement to search users and check friendship status
$stmt = $conn->prepare("
    SELECT
        u.id,
        u.username,
        u.full_name,
        u.profile_pic,
        CASE
            WHEN f.status = 'accepted' THEN 'friend'                    -- Already friends
            WHEN f.status = 'pending' AND f.user_id = ? THEN 'pending_sent' -- Request sent by you
            WHEN f.status = 'pending' AND f.friend_id = ? THEN 'pending_received' -- Request received from them
            ELSE 'none'                                                 -- No friendship yet
        END as friendship_status
    FROM users u
    LEFT JOIN friendships f ON (
        (f.user_id = ? AND f.friend_id = u.id) OR
        (f.friend_id = ? AND f.user_id = u.id)
    )
    WHERE u.id != ? AND (u.username LIKE ? OR u.full_name LIKE ?)
    LIMIT 10
");

// Execute query with session user ID and search term
$stmt->execute([
    $_SESSION['user_id'], $_SESSION['user_id'],
    $_SESSION['user_id'], $_SESSION['user_id'],
    $_SESSION['user_id'], $search, $search
]);

// Fetch all matching users
$users = $stmt->fetchAll();

// Return search results as JSON
echo json_encode(['success' => true, 'users' => $users]);
?>
```

Figure 5search.php

```php
<?php
// Include the database connection file
require_once '../config/database.php';

// Include helper functions
require_once '../includes/functions.php';

// Check if the user is logged in
check_login();

// Set response type to JSON (for AJAX calls)
header('Content-Type: application/json');

// Get the search query from the URL and sanitize it
$query = sanitize_input($_GET['query'] ?? '');

// If the query is shorter than 2 characters, return an empty result
if (strlen($query) < 2) {
    echo json_encode(['success' => true, 'users' => []]);
    exit();
}

// Add wildcard symbols (%) for SQL LIKE search
$search = "%{$query}%";

// Prepare SQL statement to search users and check friendship status
$stmt = $conn->prepare("

SELECT
        u.id,
        u.username,
        u.full_name,
        u.profile_pic,
        CASE
            WHEN f.status = 'accepted' THEN 'friend'                    -- Already friends
            WHEN f.status = 'pending' AND f.user_id = ? THEN 'pending_sent' -- Request sent by you
            WHEN f.status = 'pending' AND f.friend_id = ? THEN 'pending_received' -- Request received from them
            ELSE 'none'                                         -- No friendship yet
        END as friendship_status
    FROM users u
    LEFT JOIN friendships f ON (
        (f.user_id = ? AND f.friend_id = u.id) OR
        (f.friend_id = ? AND f.user_id = u.id)
    )
    WHERE u.id != ? AND (u.username LIKE ? OR u.full_name LIKE ?)
    LIMIT 10
");
```

(PHP.net, 2025)

```
// Execute query with session user ID and search term
$stmt->execute([
   $_SESSION['user_id'], $_SESSION['user_id'],
   $_SESSION['user_id'], $_SESSION['user_id'],
   $_SESSION['user_id'], $search, $search
]);

// Fetch all matching users
$users = $stmt->fetchAll();

// Return search results as JSON
echo json_encode(['success' => true, 'users' => $users]);
?>
```

Config folder

```php
<?php

session_start();
//Define constraints for database connection details
define('DB_HOST', 'localhost');
define('DB_USER', 'root');
define('DB_PASS', '');
define('DB_NAME', 'social_space');

try {
    // create a new PDO connection to the MySQL database
    $conn = new PDO("mysql:host=" . DB_HOST . ";dbname=" . DB_NAME, DB_USER, DB_PASS);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $conn->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
} catch(PDOException $e) {
    // if the connection fails, stop the script and show the error message
    die("Connection failed: " . $e->getMessage());
}
?>
```

Figure 6database.php


```
<?php

session_start();
//Define constraints for database connection details
define('DB_HOST', 'localhost');
define('DB_USER', 'root');
define('DB_PASS', '');
define('DB_NAME', 'social_space');

try {
   // create a new PDO connection to the MySQL database
   $conn = new PDO("mysql:host=" . DB_HOST . ";dbname=" . DB_NAME, DB_USER, DB_PASS);
   $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   $conn->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
} catch(PDOException $e) {
   // if the connection fails, stop the script and show the error message
   die("Connection failed: " . $e->getMessage());
}
?>
```


css folder with the css screenshots and profile, style scripts. The profile and style scripts are split because they are too long, however the entire application is included apart from this assignment as a zip file.

```css
/* Profile header container with rounded corners and shadow */
.profile-header {
    background: var(--bg-light);
    border-radius: 15px;
    overflow: hidden;
    margin-bottom: 30px;
    box-shadow: 0 4px 20px rgba(0, 0, 0, 0.3);
    border: 1px solid rgba(16, 185, 129, 0.2);
}

/* Cover photo area at the top (green gradient background) */
.profile-cover {
    height: 200px;
    background: linear-gradient(135deg, var(--primary-green), var(--dark-green));
    position: relative;
}

/* Container for profile info below the cover photo */
.profile-info-section {
    padding: 0 30px 30px;
    position: relative;
}

/* Container that holds the profile picture */
.profile-picture-container {
    position: relative;
    width: 150px;
    height: 150px;
    margin: -75px auto 20px; /* Negative margin pulls it up over the cover photo */
}

/* The actual profile picture (circular) */
.profile-picture {
    width: 150px;
    height: 150px;
    border-radius: 50%; /* Makes it circular */
    border: 5px solid var(--bg-light);
    object-fit: cover; /* Makes sure image fits nicely */
    display: block;
}

/* Small camera button on profile picture */
.edit-profile-pic {
    position: absolute;
    bottom: 5px;
    right: 5px;
    width: 40px;
    height: 40px;
    border-radius: 50%; /* Makes it circular */
    background: var(--primary-green);
    border: 3px solid var(--bg-light);
    color: white;
    font-size: 18px;
    cursor: pointer;
    display: flex;
    align-items: center;
    justify-content: center;
    transition: all 0.3s ease; /* Smooth animation */
}

/* When hovering over the camera button */
.edit-profile-pic:hover {
    background: var(--dark-green);
    transform: scale(1.1); /* Makes it slightly bigger */
}

/* Container for name, bio, stats (centered text) */
.profile-details {
    text-align: center;
}

/* User's full name */
.profile-details h1 {
    font-size: 28px;
    margin-bottom: 5px;
    color: var(--text-light);
}

/* Username with @ symbol */
.username-text {
    color: var(--text-gray);
    font-size: 16px;
    margin-bottom: 15px;
}

/* Bio text */
.bio-text {
    color: var(--text-light);
    font-size: 15px;
    line-height: 1.6;
    margin-bottom: 20px;
    max-width: 600px;
    margin-left: auto;
    margin-right: auto;
```

Figure 7profile.php

```
/* Stats box showing posts and friends count */
.profile-stats {
    display: flex;
    justify-content: center;
    gap: 40px;
    margin: 25px 0;
    padding: 20px;
    background: rgba(16, 185, 129, 0.05);
    border-radius: 10px;
    max-width: 400px;
    margin-left: auto;
    margin-right: auto;
}

/* Each individual stat (posts or friends) */
.stat-item {
    display: flex;
    flex-direction: column; /* Stack number above label */
    align-items: center;
}

/* The big number (like "25") */
.stat-number {
    font-size: 24px;
    font-weight: bold;
    color: var(--primary-green);
}

/* The label (like "Posts" or "Friends") */
.stat-label {
    font-size: 14px;
    color: var(--text-gray);
    margin-top: 5px;
}

/* Edit profile button */
.btn-edit-profile {
    padding: 12px 30px;
    background: var(--primary-green);
    border: none;
    border-radius: 25px; /* Rounded pill shape */
    color: white;
    font-size: 16px;
    font-weight: bold;
    cursor: pointer;
    transition: all 0.3s ease;
    margin-top: 10px;
}
```

Figure 8profile.php

```
145
146 /* When hovering over edit profile button */
147 .btn-edit-profile:hover {
148     background: var(--dark-green);
149     transform: translateY(-2px); /* Lifts up slightly */
150     box-shadow: 0 10px 30px rgba(16, 185, 129, 0.4);
151 }
152
153 /* Container for action buttons (Message, Unfriend, Add Friend) */
154 .profile-actions {
155     display: flex;
156     gap: 15px;
157     justify-content: center;
158     margin-top: 20px;
159 }
160
161 /* Secondary button style (transparent with border) */
162 .btn-secondary {
163     padding: 12px 30px;
164     background: transparent;
165     border: 2px solid var(--primary-green);
166     border-radius: 25px;
167     color: var(--primary-green);
168     font-size: 16px;
169     font-weight: bold;
170     cursor: pointer;
171     transition: all 0.3s ease;
172 }
173
174 /* When hovering over secondary button */
175 .btn-secondary:hover {
176     background: var(--primary-green);
177     color: white;
178 }
179
180 /* When button is disabled (like "Request Sent") */
181 .btn-secondary:disabled {
182     opacity: 0.5;
183     cursor: not-allowed;
184 }
185
186 /* Container for posts section */
187 .profile-content {
188     max-width: 800px;
189     margin: 0 auto;
190 }
```

Figure 9profile.php

```css
/* "Posts" section title */
.section-title {
    font-size: 24px;
    margin-bottom: 20px;
    color: var(--primary-green);
    padding-bottom: 10px;
    border-bottom: 2px solid rgba(16, 185, 129, 0.3);
}

/* Message shown when user has no posts */
.no-posts {
    text-align: center;
    padding: 60px 20px;
    background: var(--bg-light);
    border-radius: 15px;
    border: 1px solid rgba(16, 185, 129, 0.2);
}

/* Text inside no posts message */
.no-posts p {
    color: var(--text-gray);
    font-size: 18px;
}

/* Each form field group (label + input) */
.form-group {
    margin-bottom: 20px;
}

/* Form field labels */
.form-group label {
    display: block;
    margin-bottom: 8px;
    color: var(--text-light);
    font-weight: 500;
}

/* Text inputs and textareas */
.form-group input[type="text"],
.form-group textarea {
    width: 100%;
    padding: 12px 15px;
    border: 2px solid rgba(16, 185, 129, 0.3);
    border-radius: 8px;
    background: var(--bg-dark);
    color: var(--text-light);
    font-size: 15px;
    font-family: inherit;
}

/* When input is focused (clicked on) */
.form-group input[type="text"]:focus,
.form-group textarea:focus {
    outline: none;
    border-color: var(--primary-green);
}

/* File upload inputs */
.form-group input[type="file"] {
    color: var(--text-light);
    padding: 10px 0;
}

/* Small helper text under inputs */
.form-group small {
    display: block;
    margin-top: 5px;
    color: var(--text-gray);
    font-size: 13px;
}

/* Make profile look good on phones and tablets */
@media (max-width: 768px) {
    /* Shorter cover photo on mobile */
    .profile-cover {
        height: 150px;
    }

    /* Smaller profile picture on mobile */
    .profile-picture-container {
        width: 120px;
        height: 120px;
        margin: -60px auto 20px;
    }

    .profile-picture {
        width: 120px;
        height: 120px;
    }

    /* Smaller name text on mobile */
    .profile-details h1 {
        font-size: 24px;
    }

    /* Less space between stats on mobile */
    .profile-stats {
        gap: 20px;
    }

    /* Stack buttons vertically on mobile */
    .profile-actions {
        flex-direction: column;
    }

    /* Make buttons full width on mobile */
    .profile-actions button {
        width: 100%;
    }
}
```

Figure 10profile.php

profile.php code

```
/* Profile header container with rounded corners and shadow */
.profile-header {
    background: var(--bg-light);
    border-radius: 15px;
    overflow: hidden;
    margin-bottom: 30px;
    box-shadow: 0 4px 20px rgba(0, 0, 0, 0.3);
    border: 1px solid rgba(16, 185, 129, 0.2);
}

/* Cover photo area at the top (green gradient background) */
.profile-cover {
    height: 200px;
    background: linear-gradient(135deg, var(--primary-green), var(--dark-green));
    position: relative;
}

/* Container for profile info below the cover photo */
.profile-info-section {
    padding: 0 30px 30px;
    position: relative;
}

/* Container that holds the profile picture */
.profile-picture-container {
    position: relative;
    width: 150px;
    height: 150px;
    margin: -75px auto 20px; /* Negative margin pulls it up over the cover photo */
}

/* The actual profile picture (circular) */
.profile-picture {
    width: 150px;
    height: 150px;
    border-radius: 50%; /* Makes it circular */
    border: 5px solid var(--bg-light);
    object-fit: cover; /* Makes sure image fits nicely */
    display: block;
}
```

(World Wide Web Consortium, 2024)

```
/* Small camera button on profile picture */
.edit-profile-pic {
    position: absolute;
    bottom: 5px;
    right: 5px;
    width: 40px;
    height: 40px;
    border-radius: 50%; /* Makes it circular */
    background: var(--primary-green);
    border: 3px solid var(--bg-light);
    color: white;
    font-size: 18px;
    cursor: pointer;
    display: flex;
    align-items: center;
    justify-content: center;
    transition: all 0.3s ease; /* Smooth animation */
}

/* When hovering over the camera button */
.edit-profile-pic:hover {
    background: var(--dark-green);
    transform: scale(1.1); /* Makes it slightly bigger */
}

/* Container for name, bio, stats (centered text) */
.profile-details {
    text-align: center;
}

/* User's full name */
.profile-details h1 {
    font-size: 28px;
    margin-bottom: 5px;
    color: var(--text-light);
}

/* Username with @ symbol */
.username-text {
    color: var(--text-gray);
    font-size: 16px;
    margin-bottom: 15px;
}
```

(World Wide Web Consortium, 2024)

```
/* Bio text */
.bio-text {
    color: var(--text-light);
    font-size: 15px;
    line-height: 1.6;
    margin-bottom: 20px;
    max-width: 600px;
    margin-left: auto;
    margin-right: auto;
}

/* Stats box showing posts and friends count */
.profile-stats {
    display: flex;
    justify-content: center;
    gap: 40px;
    margin: 25px 0;
    padding: 20px;
    background: rgba(16, 185, 129, 0.05);
    border-radius: 10px;
    max-width: 400px;
    margin-left: auto;
    margin-right: auto;
}

/* Each individual stat (posts or friends) */
.stat-item {
    display: flex;
    flex-direction: column; /* Stack number above label */
    align-items: center;
}

/* The big number (like "25") */
.stat-number {
    font-size: 24px;
    font-weight: bold;
    color: var(--primary-green);
}

/* The label (like "Posts" or "Friends") */
.stat-label {
    font-size: 14px;
    color: var(--text-gray);
    margin-top: 5px;
}

/* Edit profile button */
.btn-edit-profile {
    padding: 12px 30px;
    background: var(--primary-green);
    border: none;
    border-radius: 25px; /* Rounded pill shape */
    color: white;
    font-size: 16px;
    font-weight: bold;
    cursor: pointer;
    transition: all 0.3s ease;
    margin-top: 10px;
}
```

```
/* When hovering over edit profile button */
.btn-edit-profile:hover {
    background: var(--dark-green);
    transform: translateY(-2px); /* Lifts up slightly */
    box-shadow: 0 10px 30px rgba(16, 185, 129, 0.4);
}

/* Container for action buttons (Message, Unfriend, Add Friend) */
.profile-actions {
    display: flex;
    gap: 15px;
    justify-content: center;
    margin-top: 20px;
}

/* Secondary button style (transparent with border) */
.btn-secondary {
    padding: 12px 30px;
    background: transparent;
    border: 2px solid var(--primary-green);
    border-radius: 25px;
    color: var(--primary-green);
    font-size: 16px;
    font-weight: bold;
    cursor: pointer;
    transition: all 0.3s ease;
}

/* When hovering over secondary button */
.btn-secondary:hover {
    background: var(--primary-green);
    color: white;
}
/* When button is disabled (like "Request Sent") */
.btn-secondary:disabled {
    opacity: 0.5;
    cursor: not-allowed;
}

/* Container for posts section */
.profile-content {
    max-width: 800px;
    margin: 0 auto;
}

/* "Posts" section title */
.section-title {
    font-size: 24px;
    margin-bottom: 20px;
    color: var(--primary-green);
    padding-bottom: 10px;
    border-bottom: 2px solid rgba(16, 185, 129, 0.3);
}
```

```
/* Message shown when user has no posts */
.no-posts {
    text-align: center;
    padding: 60px 20px;
    background: var(--bg-light);
    border-radius: 15px;
    border: 1px solid rgba(16, 185, 129, 0.2);
}

/* Text inside no posts message */
.no-posts p {
    color: var(--text-gray);
    font-size: 18px;
}

/* Each form field group (label + input) */
.form-group {
    margin-bottom: 20px;
}

/* Form field labels */
.form-group label {
    display: block;
    margin-bottom: 8px;
    color: var(--text-light);
    font-weight: 500;
}

/* Text inputs and textareas */
.form-group input[type="text"],
.form-group textarea {
    width: 100%;
    padding: 12px 15px;
    border: 2px solid rgba(16, 185, 129, 0.3);
    border-radius: 8px;
    background: var(--bg-dark);
    color: var(--text-light);
    font-size: 15px;
    font-family: inherit;
}
/* When input is focused (clicked on) */
.form-group input[type="text"]:focus,
.form-group textarea:focus {
    outline: none;
    border-color: var(--primary-green);
}

/* File upload inputs */
.form-group input[type="file"] {
    color: var(--text-light);
    padding: 10px 0;
}
```

```
/* Small helper text under inputs */
.form-group small {
    display: block;
    margin-top: 5px;
    color: var(--text-gray);
    font-size: 13px;
}

/* Make profile look good on phones and tablets */
@media (max-width: 768px) {
    /* Shorter cover photo on mobile */
    .profile-cover {
        height: 150px;
    }

    /* Smaller profile picture on mobile */
    .profile-picture-container {
        width: 120px;
        height: 120px;
        margin: -60px auto 20px;
    }

    .profile-picture {
        width: 120px;
        height: 120px;
    }

    /* Smaller name text on mobile */
    .profile-details h1 {
        font-size: 24px;
    }

    /* Less space between stats on mobile */
    .profile-stats {
        gap: 20px;
    }

    /* Stack buttons vertically on mobile */
    .profile-actions {
        flex-direction: column;
    }

    /* Make buttons full width on mobile */
    .profile-actions button {
        width: 100%;
    }
}
```

The style.css script won't be included because there are 1000 lines of code which is included in the zip file. Thank you

The database folder with social_app.sql screenshot and script

```sql
--
-- Database: `social_space`
--

-- --------------------------------------------------------

--
-- Table structure for table `comments`
--

CREATE TABLE `comments` (
  `id` int(11) NOT NULL,
  `post_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `comment` text NOT NULL,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `friendships`
--

CREATE TABLE `friendships` (
  `id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `friend_id` int(11) NOT NULL,
  `status` enum('pending','accepted','rejected') DEFAULT 'pending',
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `likes`
--

```

Figure 11social_space.sql

(MySQL , 2024)

```sql
CREATE TABLE `likes` (
  `id` int(11) NOT NULL,
  `post_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `messages`
--

CREATE TABLE `messages` (
  `id` int(11) NOT NULL,
  `sender_id` int(11) NOT NULL,
  `receiver_id` int(11) NOT NULL,
  `message` text NOT NULL,
  `is_read` tinyint(1) DEFAULT 0,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `message_themes`
--

CREATE TABLE `message_themes` (
  `id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `friend_id` int(11) NOT NULL,
  `theme` varchar(50) DEFAULT 'default'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `posts`
--

CREATE TABLE `posts` (
  `id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `content` text NOT NULL,
  `image` varchar(255) DEFAULT NULL,
  `likes_count` int(11) DEFAULT 0,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

```

Figure 12social_space.sql

```sql
-- Table structure for table `shares`
--

CREATE TABLE `shares` (
  `id` int(11) NOT NULL,
  `post_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `users`
--

CREATE TABLE `users` (
  `id` int(11) NOT NULL,
  `username` varchar(50) NOT NULL,
  `email` varchar(100) NOT NULL,
  `password` varchar(255) NOT NULL,
  `full_name` varchar(100) NOT NULL,
  `profile_pic` varchar(255) DEFAULT 'default.jpg',
  `bio` text DEFAULT NULL,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--
-- Indexes for dumped tables
--

--
-- Indexes for table `comments`
--
ALTER TABLE `comments`
  ADD PRIMARY KEY (`id`),
  ADD KEY `post_id` (`post_id`),
  ADD KEY `user_id` (`user_id`);

--
-- Indexes for table `friendships`
--
ALTER TABLE `friendships`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `unique_friendship` (`user_id`,`friend_id`),
  ADD KEY `friend_id` (`friend_id`);

--
-- Indexes for table `likes`
--
ALTER TABLE `likes`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `unique_like` (`post_id`,`user_id`),
  ADD KEY `user_id` (`user_id`);

```

(MySQL , 2024)

Figure 13social_space.sql

```sql
--
-- Indexes for table `messages`
--
ALTER TABLE `messages`
  ADD PRIMARY KEY (`id`),
  ADD KEY `sender_id` (`sender_id`),
  ADD KEY `receiver_id` (`receiver_id`);

--
-- Indexes for table `message_themes`
--
ALTER TABLE `message_themes`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `unique_theme` (`user_id`,`friend_id`),
  ADD KEY `friend_id` (`friend_id`);

--
-- Indexes for table `posts`
--
ALTER TABLE `posts`
  ADD PRIMARY KEY (`id`),
  ADD KEY `user_id` (`user_id`);

--
-- Indexes for table `shares`
--
ALTER TABLE `shares`
  ADD PRIMARY KEY (`id`),
  ADD KEY `post_id` (`post_id`),
  ADD KEY `user_id` (`user_id`);

--
-- Indexes for table `users`
--
ALTER TABLE `users`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `username` (`username`),
  ADD UNIQUE KEY `email` (`email`);

--
-- AUTO_INCREMENT for dumped tables
--

--
-- AUTO_INCREMENT for table `comments`
--
ALTER TABLE `comments`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

```

Figure 14social_space.sql

(MySQL , 2024)

```
--
-- AUTO_INCREMENT for table `friendships`
--
ALTER TABLE `friendships`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `likes`
--
ALTER TABLE `likes`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `messages`
--
ALTER TABLE `messages`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `message_themes`
--
ALTER TABLE `message_themes`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `posts`
--
ALTER TABLE `posts`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `shares`
--
ALTER TABLE `shares`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `users`
--
ALTER TABLE `users`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- Constraints for dumped tables
--

--
-- Constraints for table `comments`
--
```

Figure 15social_space.sql

(MySQL , 2024)

```
ALTER TABLE `comments`
  ADD CONSTRAINT `comments_ibfk_1` FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `comments_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Constraints for table `friendships`
--
ALTER TABLE `friendships`
  ADD CONSTRAINT `friendships_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `friendships_ibfk_2` FOREIGN KEY (`friend_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Constraints for table `likes`
--
ALTER TABLE `likes`
  ADD CONSTRAINT `likes_ibfk_1` FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `likes_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Constraints for table `messages`
--
ALTER TABLE `messages`
  ADD CONSTRAINT `messages_ibfk_1` FOREIGN KEY (`sender_id`) REFERENCES `users` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `messages_ibfk_2` FOREIGN KEY (`receiver_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Constraints for table `message_themes`
--
ALTER TABLE `message_themes`
  ADD CONSTRAINT `message_themes_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `message_themes_ibfk_2` FOREIGN KEY (`friend_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Constraints for table `posts`
--
ALTER TABLE `posts`
  ADD CONSTRAINT `posts_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Constraints for table `shares`
--
ALTER TABLE `shares`
  ADD CONSTRAINT `shares_ibfk_1` FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `shares_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;
COMMIT;
```

Figure 16social_space.sql

(MySQL , 2024)

```
-
-- Database: `social_space`
--

-- --------------------------------------------------------

--
-- Table structure for table `comments`
--

CREATE TABLE `comments` (
  `id` int(11) NOT NULL,
  `post_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `comment` text NOT NULL,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `friendships`
--

CREATE TABLE `friendships` (
  `id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `friend_id` int(11) NOT NULL,
  `status` enum('pending','accepted','rejected') DEFAULT 'pending',
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `likes`
--

CREATE TABLE `likes` (
  `id` int(11) NOT NULL,
  `post_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

(MySQL , 2024)
```

```
--
-- Table structure for table `messages`
--

CREATE TABLE `messages` (
  `id` int(11) NOT NULL,
  `sender_id` int(11) NOT NULL,
  `receiver_id` int(11) NOT NULL,
  `message` text NOT NULL,
  `is_read` tinyint(1) DEFAULT 0,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `message_themes`
--

CREATE TABLE `message_themes` (
  `id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `friend_id` int(11) NOT NULL,
  `theme` varchar(50) DEFAULT 'default'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `posts`
--

CREATE TABLE `posts` (
  `id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `content` text NOT NULL,
  `image` varchar(255) DEFAULT NULL,
  `likes_count` int(11) DEFAULT 0,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `shares`
--
```

```
CREATE TABLE `shares` (
  `id` int(11) NOT NULL,
  `post_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `users`
--

CREATE TABLE `users` (
  `id` int(11) NOT NULL,
  `username` varchar(50) NOT NULL,
  `email` varchar(100) NOT NULL,
  `password` varchar(255) NOT NULL,
  `full_name` varchar(100) NOT NULL,
  `profile_pic` varchar(255) DEFAULT 'default.jpg',
  `bio` text DEFAULT NULL,
  `created_at` timestamp NOT NULL DEFAULT current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--
-- Indexes for dumped tables
--

--
-- Indexes for table `comments`
--
ALTER TABLE `comments`
  ADD PRIMARY KEY (`id`),
  ADD KEY `post_id` (`post_id`),
  ADD KEY `user_id` (`user_id`);

--
-- Indexes for table `friendships`
--
ALTER TABLE `friendships`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `unique_friendship` (`user_id`,`friend_id`),
  ADD KEY `friend_id` (`friend_id`);
```

```
--
-- Indexes for table `likes`
--
ALTER TABLE `likes`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `unique_like` (`post_id`,`user_id`),
  ADD KEY `user_id` (`user_id`);

--
-- Indexes for table `messages`
--
ALTER TABLE `messages`
  ADD PRIMARY KEY (`id`),
  ADD KEY `sender_id` (`sender_id`),
  ADD KEY `receiver_id` (`receiver_id`);

--
-- Indexes for table `message_themes`
--
ALTER TABLE `message_themes`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `unique_theme` (`user_id`,`friend_id`),
  ADD KEY `friend_id` (`friend_id`);

--
-- Indexes for table `posts`
--
ALTER TABLE `posts`
  ADD PRIMARY KEY (`id`),
  ADD KEY `user_id` (`user_id`);

--
-- Indexes for table `shares`
--
ALTER TABLE `shares`
  ADD PRIMARY KEY (`id`),
  ADD KEY `post_id` (`post_id`),
  ADD KEY `user_id` (`user_id`);

--
-- Indexes for table `users`
--
ALTER TABLE `users`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `username` (`username`),
  ADD UNIQUE KEY `email` (`email`);

--
-- AUTO_INCREMENT for dumped tables
--
(MySQL , 2024)
```

```
--
-- AUTO_INCREMENT for table `comments`
--
ALTER TABLE `comments`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `friendships`
--
ALTER TABLE `friendships`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `likes`
--
ALTER TABLE `likes`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `messages`
--
ALTER TABLE `messages`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `message_themes`
--
ALTER TABLE `message_themes`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `posts`
--
ALTER TABLE `posts`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `shares`
--
ALTER TABLE `shares`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

--
-- AUTO_INCREMENT for table `users`
--
ALTER TABLE `users`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
```

(MySQL , 2024)

```
-
-- Constraints for dumped tables
--

--
-- Constraints for table `comments`
--
ALTER TABLE `comments`
  ADD CONSTRAINT `comments_ibfk_1` FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `comments_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Constraints for table `friendships`
--
ALTER TABLE `friendships`
  ADD CONSTRAINT `friendships_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `friendships_ibfk_2` FOREIGN KEY (`friend_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Constraints for table `likes`
--
ALTER TABLE `likes`
  ADD CONSTRAINT `likes_ibfk_1` FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `likes_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Constraints for table `messages`
--
ALTER TABLE `messages`
  ADD CONSTRAINT `messages_ibfk_1` FOREIGN KEY (`sender_id`) REFERENCES `users` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `messages_ibfk_2` FOREIGN KEY (`receiver_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Constraints for table `message_themes`
--
ALTER TABLE `message_themes`
  ADD CONSTRAINT `message_themes_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE
CASCADE,
  ADD CONSTRAINT `message_themes_ibfk_2` FOREIGN KEY (`friend_id`) REFERENCES `users` (`id`) ON DELETE
CASCADE;

--
-- Constraints for table `posts`
--
ALTER TABLE `posts`
  ADD CONSTRAINT `posts_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;

--
-- Constraints for table `shares`
--
ALTER TABLE `shares`
  ADD CONSTRAINT `shares_ibfk_1` FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE CASCADE,
  ADD CONSTRAINT `shares_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE;
COMMIT;
```

This is the includes folder with the screenshot and script file: functions.

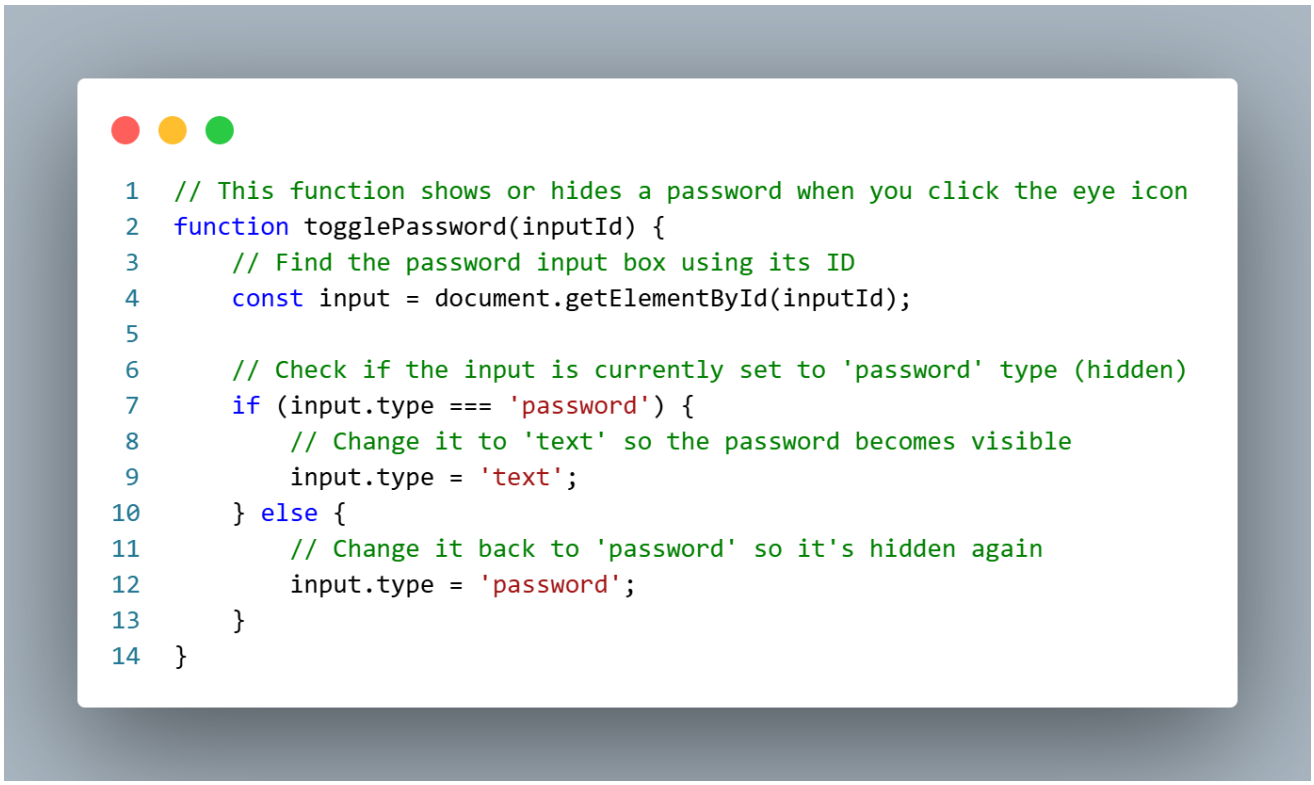
```php
<?php
// Clean up user input to prevent harmful code
function sanitize_input($data) {
    // Remove extra spaces from beginning and end
    $data = trim($data);
    // Remove backslashes
    $data = stripslashes($data);
    // Convert special characters to safe HTML (prevents hacking)
    $data = htmlspecialchars($data, ENT_QUOTES, 'UTF-8');
    // Return the cleaned data
    return $data;
}

// Check if user is logged in, if not send them to login page
function check_login() {
    // If user_id is not in the session (not logged in)
    if (!isset($_SESSION['user_id'])) {
        // Send them to the login page
        header("Location: index.php");
        // Stop running any more code
        exit();
    }
}

// Convert a timestamp to "time ago" format (like "2h ago")
function time_elapsed($datetime) {
    // Get the current time
    $now = new DateTime;
    // Get the time when something happened
    $ago = new DateTime($datetime);
    // Calculate the difference between now and then
    $diff = $now->diff($ago);

    // If more than a year ago, show years
    if ($diff->y > 0) return $diff->y . 'y ago';
    // If more than a month ago, show months
    if ($diff->m > 0) return $diff->m . 'mo ago';
    // If more than a day ago, show days
    if ($diff->d > 0) return $diff->d . 'd ago';
    // If more than an hour ago, show hours
    if ($diff->h > 0) return $diff->h . 'h ago';
    // If more than a minute ago, show minutes
    if ($diff->i > 0) return $diff->i . 'm ago';
    // If less than a minute, show "Just now"
    return 'Just now';
}

// Get a user's basic information from the database
function get_user_info($conn, $user_id) {
    // Prepare a query to get user info
    $stmt = $conn->prepare("SELECT id, username, full_name, profile_pic FROM users WHERE id = ?");
    // Run the query with the user's ID
    $stmt->execute([$user_id]);
    // Return the user's information
    return $stmt->fetch();
}
?>
```

Figure 17functions

```php
<?php
// Clean up user input to prevent harmful code
function sanitize_input($data) {
   // Remove extra spaces from beginning and end
   $data = trim($data);
   // Remove backslashes
   $data = stripslashes($data);
   // Convert special characters to safe HTML (prevents hacking)
   $data = htmlspecialchars($data, ENT_QUOTES, 'UTF-8');
   // Return the cleaned data
   return $data;
}

// Check if user is logged in, if not send them to login page
function check_login() {
   // If user_id is not in the session (not logged in)
   if (!isset($_SESSION['user_id'])) {
      // Send them to the login page
      header("Location: index.php");
      // Stop running any more code
      exit();
   }
}

// Convert a timestamp to "time ago" format (like "2h ago")
function time_elapsed($datetime) {
   // Get the current time
   $now = new DateTime;
   // Get the time when something happened
   $ago = new DateTime($datetime);
   // Calculate the difference between now and then
   $diff = $now->diff($ago);

   // If more than a year ago, show years
   if ($diff->y > 0) return $diff->y . 'y ago';
   // If more than a month ago, show months
   if ($diff->m > 0) return $diff->m . 'mo ago';
   // If more than a day ago, show days
   if ($diff->d > 0) return $diff->d . 'd ago';
   // If more than an hour ago, show hours
   if ($diff->h > 0) return $diff->h . 'h ago';
   // If more than a minute ago, show minutes
   if ($diff->i > 0) return $diff->i . 'm ago';
   // If less than a minute, show "Just now"
   return 'Just now';
}
```
(Developer Mozilla, 2024)

```
// Get a user's basic information from the database
function get_user_info($conn, $user_id) {
   // Prepare a query to get user info
   $stmt = $conn->prepare("SELECT id, username, full_name, profile_pic FROM users WHERE id = ?");
   // Run the query with the user's ID
   $stmt->execute([$user_id]);
   // Return the user's information
   return $stmt->fetch();
}
?>
```

This is the js folder with the screenshots and scripts namely: auth, dashboard, particles and profile



```
// This function shows or hides a password when you click the eye icon
function togglePassword(inputId) {
    // Find the password input box using its ID
    const input = document.getElementById(inputId);

    // Check if the input is currently set to 'password' type (hidden)
    if (input.type === 'password') {
        // Change it to 'text' so the password becomes visible
        input.type = 'text';
    } else {
        // Change it back to 'password' so it's hidden again
        input.type = 'password';
    }
}
```

Figure 18auth.js

```
// This function shows or hides a password when you click the eye icon
function togglePassword(inputId) {
   // Find the password input box using its ID
   const input = document.getElementById(inputId);

   // Check if the input is currently set to 'password' type (hidden)
   if (input.type === 'password') {
      // Change it to 'text' so the password becomes visible
      input.type = 'text';
   } else {
      // Change it back to 'password' so it's hidden again
      input.type = 'password';
   }
}
```

```javascript
// Show the edit profile popup
function showEditProfile() {
    // Find the edit profile modal and add 'active' class to show it
    document.getElementById('edit-profile-modal').classList.add('active');
}

// When the edit profile form is submitted
document.getElementById('edit-profile-form').addEventListener('submit', async (e) => {
    // Stop the form from refreshing the page
    e.preventDefault();

    // Get all the form data (profile picture, name, bio)
    const formData = new FormData(e.target);
    // Add an action to tell the server we want to update the profile
    formData.append('action', 'update');

    try {
        // Send the profile data to the server
        const response = await fetch('api/profile.php', {
            method: 'POST',
            body: formData
        });

        // Get the response from the server
        const data = await response.json();

        // Check if the profile was updated successfully
        if (data.success) {
            alert('Profile updated successfully!');
            // Refresh the page to show the updated profile
            location.reload();
        } else {
            // Show error message from server
            alert(data.message);
        }
    } catch (error) {
        // If something went wrong, log it and show an error
        console.error('Error:', error);
        alert('Failed to update profile');
    }
});
```

Figure 19profile.js

(Developer Mozilla, 2024)

```
// Show the edit profile popup
function showEditProfile() {
   // Find the edit profile modal and add 'active' class to show it
   document.getElementById('edit-profile-modal').classList.add('active');
}

// When the edit profile form is submitted
document.getElementById('edit-profile-form').addEventListener('submit', async (e) => {
   // Stop the form from refreshing the page
   e.preventDefault();

   // Get all the form data (profile picture, name, bio)
   const formData = new FormData(e.target);
   // Add an action to tell the server we want to update the profile
   formData.append('action', 'update');

   try {
      // Send the profile data to the server
      const response = await fetch('api/profile.php', {
         method: 'POST',
         body: formData
      });

      // Get the response from the server
      const data = await response.json();

      // Check if the profile was updated successfully
      if (data.success) {
         alert('Profile updated successfully!');
         // Refresh the page to show the updated profile
         location.reload();
      } else {
         // Show error message from server
         alert(data.message);
      }
   } catch (error) {
      // If something went wrong, log it and show an error
      console.error('Error:', error);
      alert('Failed to update profile');
   }
});
```

(Developer Mozilla, 2024)

```
// Get the canvas element where we'll draw the particles
const canvas = document.getElementById('particles-canvas');
// Get the drawing context (like a paintbrush) for the canvas
const ctx = canvas.getContext('2d');

// Make the canvas fill the whole browser window
canvas.width = window.innerWidth;
canvas.height = window.innerHeight;

// Create an empty array to store all our particles
const particles = [];
// Decide how many particles we want (100 floating dots)
const particleCount = 100;

// This is a blueprint for creating particles
class Particle {
    // When we create a new particle, set up its properties
    constructor() {
        // Put it at a random position on the screen (X coordinate)
        this.x = Math.random() * canvas.width;
        // Put it at a random position on the screen (Y coordinate)
        this.y = Math.random() * canvas.height;
        // Give it a random size between 1 and 4 pixels
        this.size = Math.random() * 3 + 1;
        // Give it a random horizontal speed (can move left or right)
        this.speedX = Math.random() * 2 - 1;
        // Give it a random vertical speed (can move up or down)
        this.speedY = Math.random() * 2 - 1;
        // Give it a green color with random transparency
        this.color = `rgba(16, 185, 129, ${Math.random() * 0.5 + 0.2})`;
    }

    // Move the particle
    update() {
        // Move it horizontally based on its speed
        this.x += this.speedX;
        // Move it vertically based on its speed
        this.y += this.speedY;

```

Figure 20particles.js

(Developer Mozilla, 2024)

```
        // If particle goes off the right edge, bring it back on the left
        if (this.x > canvas.width) this.x = 0;
        // If particle goes off the left edge, bring it back on the right
        if (this.x < 0) this.x = canvas.width;
        // If particle goes off the bottom, bring it back on the top
        if (this.y > canvas.height) this.y = 0;
        // If particle goes off the top, bring it back on the bottom
        if (this.y < 0) this.y = canvas.height;
    }

    // Draw the particle on the canvas
    draw() {
        // Set the color for this particle
        ctx.fillStyle = this.color;
        // Start drawing a shape
        ctx.beginPath();
        // Draw a circle at the particle's position
        ctx.arc(this.x, this.y, this.size, 0, Math.PI * 2);
        // Fill the circle with color
        ctx.fill();
    }
}

// Create all the particles when the page loads
function init() {
    // Loop 100 times
    for (let i = 0; i < particleCount; i++) {
        // Create a new particle and add it to our array
        particles.push(new Particle());
    }
}
```

Figure 21particles.js

(Developer Mozilla, 2024)

```
72  // Draw lines between particles that are close to each other
73  function connectParticles() {
74      // Loop through each particle
75      for (let i = 0; i < particles.length; i++) {
76          // Compare it with every other particle
77          for (let j = i + 1; j < particles.length; j++) {
78              // Calculate horizontal distance between the two particles
79              const dx = particles[i].x - particles[j].x;
80              // Calculate vertical distance between the two particles
81              const dy = particles[i].y - particles[j].y;
82              // Calculate the actual distance using Pythagorean theorem
83              const distance = Math.sqrt(dx * dx + dy * dy);
84
85              // If particles are close enough (less than 120 pixels apart)
86              if (distance < 120) {
87                  // Set line color - gets more transparent as distance increases
88                  ctx.strokeStyle = `rgba(16, 185, 129, ${0.2 - distance / 600})`;
89                  // Set line thickness to 1 pixel
90                  ctx.lineWidth = 1;
91                  // Start drawing a line
92                  ctx.beginPath();
93                  // Start the line at the first particle
94                  ctx.moveTo(particles[i].x, particles[i].y);
95                  // Draw the line to the second particle
96                  ctx.lineTo(particles[j].x, particles[j].y);
97                  // Actually draw the line on the canvas
98                  ctx.stroke();
99              }
100         }
101     }
102 }
```

Figure 22particles.js

(Developer Mozilla, 2024)

```
// This function runs over and over to create animation
function animate() {
    // Clear the entire canvas (erase everything)
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    // Loop through all particles
    for (let i = 0; i < particles.length; i++) {
        // Move this particle
        particles[i].update();
        // Draw this particle
        particles[i].draw();
    }

    // Draw lines connecting nearby particles
    connectParticles();
    // Call this function again on the next frame (creates smooth animation)
    requestAnimationFrame(animate);
}

// When the browser window is resized
window.addEventListener('resize', () => {
    // Update canvas width to match new window size
    canvas.width = window.innerWidth;
    // Update canvas height to match new window size
    canvas.height = window.innerHeight;
});

// Create all the particles
init();
// Start the animation loop
animate();
```

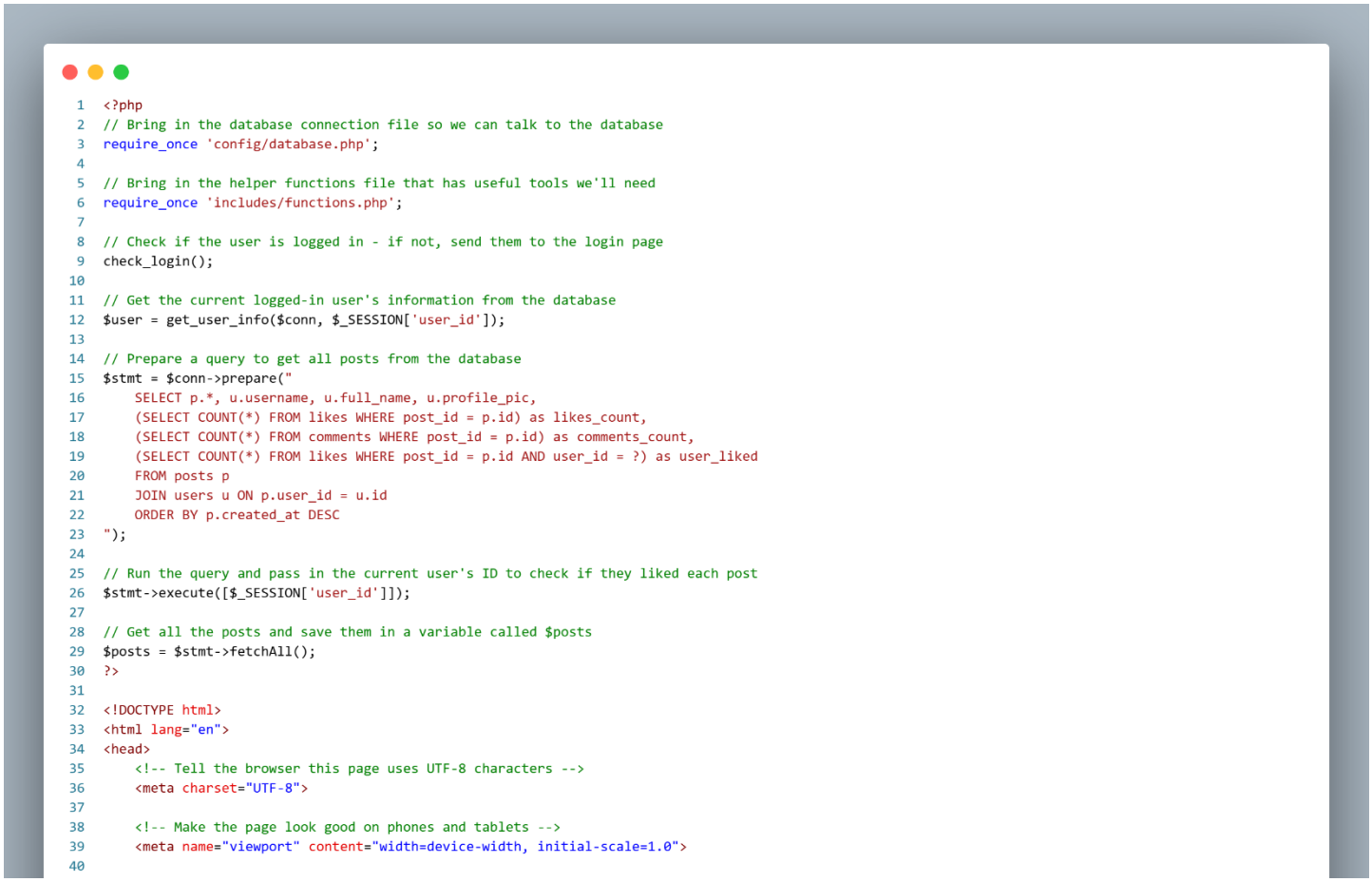
Figure 23particles.js

Please note that the dashboard.js will not be included because there is a lot of code, but it is in the zip file.

(Developer Mozilla, 2024)

Root folder, scripts that are in the root folder.

- dashboard
- index
- logout
- profile
- register
- structure



```php
<?php
// Bring in the database connection file so we can talk to the database
require_once 'config/database.php';

// Bring in the helper functions file that has useful tools we'll need
require_once 'includes/functions.php';

// Check if the user is logged in - if not, send them to the login page
check_login();

// Get the current logged-in user's information from the database
$user = get_user_info($conn, $_SESSION['user_id']);

// Prepare a query to get all posts from the database
$stmt = $conn->prepare("
    SELECT p.*, u.username, u.full_name, u.profile_pic,
    (SELECT COUNT(*) FROM likes WHERE post_id = p.id) as likes_count,
    (SELECT COUNT(*) FROM comments WHERE post_id = p.id) as comments_count,
    (SELECT COUNT(*) FROM likes WHERE post_id = p.id AND user_id = ?) as user_liked
    FROM posts p
    JOIN users u ON p.user_id = u.id
    ORDER BY p.created_at DESC
");

// Run the query and pass in the current user's ID to check if they liked each post
$stmt->execute([$_SESSION['user_id']]);

// Get all the posts and save them in a variable called $posts
$posts = $stmt->fetchAll();
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Tell the browser this page uses UTF-8 characters -->
    <meta charset="UTF-8">

    <!-- Make the page look good on phones and tablets -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

```

Figure 24dashboard.php

```
    <!-- Set the title that shows in the browser tab -->
    <title>Social Space - Dashboard</title>

    <!-- Load the main style sheet for the whole site -->
    <link rel="stylesheet" href="css/style.css">

    <!-- Load the dashboard-specific style sheet -->
    <link rel="stylesheet" href="css/dashboard.css">
</head>
<body>
    <!-- This is the navigation bar at the top of the page -->
    <nav class="navbar">
        <!-- Container to hold everything in the navbar -->
        <div class="nav-container">
            <!-- The Social Space logo/title -->
            <h1 class="nav-logo">Social Space</h1>

            <!-- Search bar section -->
            <div class="nav-search">
                <!-- Input box where users can type to search for friends -->
                <input type="text" id="search-users" placeholder="Search friends..." autocomplete="off">

                <!-- This div will show the search results when user types -->
                <div id="search-results" class="search-results"></div>
            </div>

            <!-- Navigation menu with all the buttons -->
            <div class="nav-menu">
                <!-- Button to create a new post -->
                <button onclick="showCreatePost()" class="nav-btn">➕ Post</button>

                <!-- Button to see friend requests -->
                <button onclick="showFriendRequests()" class="nav-btn">👥 Friends</button>

                <!-- Button to open messages -->
                <button onclick="showMessages()" class="nav-btn">💬 Messages</button>

                <!-- Button to go to user's profile page -->
                <button onclick="location.href='profile.php'" class="nav-btn">👤 Profile</button>

                <!-- Link to log out of the account -->
                <a href="logout.php" class="nav-btn">Logout</a>
            </div>
        </div>
    </nav>
```

Figure 26dashboard.php

```
    <!-- Main container that holds all the content on the page -->
    <div class="main-container">
        <!-- Container specifically for the posts feed -->
        <div class="feed-container">
            <!-- This div will contain all the posts -->
            <div id="posts-feed">
                <!-- Loop through each post and display it -->
                <?php foreach ($posts as $post): ?>
                    <!-- Each post is wrapped in this card -->
                    <div class="post-card" data-post-id="<?php echo $post['id']; ?>">
                        <!-- Header section of the post (shows who posted it) -->
                        <div class="post-header">
                            <!-- User information section -->
                            <div class="post-user">
                                <!-- Show the user's profile picture -->
                                <img src="uploads/profiles/<?php echo htmlspecialchars($post['profile_pic']); ?>" alt="Profile" class="post-avatar">

                                <!-- User's name and time posted -->
                                <div>
                                    <!-- Display the user's full name -->
                                    <h4><?php echo htmlspecialchars($post['full_name']); ?></h4>

                                    <!-- Show how long ago the post was created (like "2 hours ago") -->
                                    <span class="post-time"><?php echo time_elapsed($post['created_at']); ?></span>
                                </div>
                            </div>

                            <!-- If this is the current user's post, show delete button -->
                            <?php if ($post['user_id'] == $_SESSION['user_id']): ?>
                                <!-- Delete button (trash can icon) -->
                                <button class="delete-btn" onclick="deletePost(<?php echo $post['id']; ?>)">🗑</button>
                            <?php endif; ?>
                        </div>

                        <!-- The actual content of the post -->
                        <div class="post-content">
                            <!-- Display the post text (nl2br makes line breaks work) -->
                            <p><?php echo nl2br(htmlspecialchars($post['content'])); ?></p>

                            <!-- If the post has an image, show it -->
                            <?php if ($post['image']): ?>
                                <img src="uploads/posts/<?php echo htmlspecialchars($post['image']); ?>" alt="Post" class="post-image">
                            <?php endif; ?>
                        </div>
```

Figure 25dashboard.php

```
                        <!-- Stats showing number of likes and comments -->
                        <div class="post-stats">
                            <!-- Show how many likes this post has -->
                            <span><?php echo $post['likes_count']; ?> likes</span>

                            <!-- Show how many comments this post has -->
                            <span><?php echo $post['comments_count']; ?> comments</span>
                        </div>

                        <!-- Action buttons (Like, Comment, Share) -->
                        <div class="post-actions">
                            <!-- Like button - add 'liked' class if user already liked it -->
                            <button onclick="likePost(<?php echo $post['id']; ?>)" class="action-btn <?php echo $post['user_liked'] ? 'liked' : ''; ?>">
                                ❤️ Like
                            </button>

                            <!-- Comment button - opens the comments section -->
                            <button onclick="toggleComments(<?php echo $post['id']; ?>)" class="action-btn">💬 Comment</button>

                            <!-- Share button - lets users share the post -->
                            <button onclick="sharePost(<?php echo $post['id']; ?>)" class="action-btn">🔗 Share</button>
                        </div>

                        <!-- Comments section (hidden by default) -->
                        <div class="comments-section" id="comments-<?php echo $post['id']; ?>" style="display:none;">
                            <!-- This div will be filled with comments using JavaScript -->
                            <div class="comments-list"></div>

                            <!-- Input area to write a new comment -->
                            <div class="comment-input">
                                <!-- Text box to type the comment -->
                                <input type="text" placeholder="Write a comment..." id="comment-input-<?php echo $post['id']; ?>">

                                <!-- Button to send the comment -->
                                <button onclick="addComment(<?php echo $post['id']; ?>)">Send</button>
                            </div>
                        </div>
                    </div>
                <?php endforeach; ?>
            </div>
        </div>
    </div>
```

Figure 28dashboard.php

```
    <!-- Modal/popup for creating a new post -->
    <div id="create-post-modal" class="modal">
        <!-- Content inside the modal -->
        <div class="modal-content">
            <!-- X button to close the modal -->
            <span class="close" onclick="closeModal('create-post-modal')">&times;</span>

            <!-- Title of the modal -->
            <h2>Create Post</h2>

            <!-- Form to submit a new post -->
            <form id="create-post-form" enctype="multipart/form-data">
                <!-- Text area where user types their post -->
                <textarea name="content" placeholder="What's on your mind?" required></textarea>

                <!-- File input to upload an image with the post -->
                <input type="file" name="image" accept="image/*">

                <!-- Submit button to post -->
                <button type="submit" class="btn-primary">Post</button>
            </form>
        </div>
    </div>

    <!-- Modal/popup for viewing friend requests -->
    <div id="friend-requests-modal" class="modal">
        <!-- Content inside the modal -->
        <div class="modal-content">
            <!-- X button to close the modal -->
            <span class="close" onclick="closeModal('friend-requests-modal')">&times;</span>

            <!-- Title of the modal -->
            <h2>Friend Requests</h2>

            <!-- This div will be filled with friend requests using JavaScript -->
            <div id="friend-requests-list"></div>
        </div>
    </div>

    <!-- Modal/popup for messaging friends -->
    <div id="messages-modal" class="modal">
        <!-- Content inside the modal (wider for chat layout) -->
        <div class="modal-content messages-modal-content">
            <!-- X button to close the modal -->
            <span class="close" onclick="closeModal('messages-modal')">&times;</span>

            <!-- Container for the messaging interface -->
            <div class="messages-container">
                <!-- Left side: List of friends to chat with -->
                <div class="friends-list">
                    <!-- Title -->
                    <h3>Friends</h3>
```

Figure 27dashboard.php

```
227
228                     <!-- This div will be filled with friend list using JavaScript -->
229                     <div id="friends-list"></div>
230                 </div>
231
232                 <!-- Right side: Chat area -->
233                 <div class="chat-area">
234                     <!-- Top of chat showing who you're talking to -->
235                     <div class="chat-header">
236                         <!-- Name of the friend you're chatting with -->
237                         <h3 id="chat-username">Select a friend to chat</h3>
238
239                         <!-- Button to change chat theme (hidden until you select a friend) -->
240                         <button id="theme-btn" onclick="showThemeSelector()" style="display:none;">🎨</button>
241
242                         <!-- Button to delete the entire chat history (hidden until you select a friend) -->
243                         <button id="delete-chat-btn" onclick="deleteChat()" style="display:none;">🗑️</button>
244                     </div>
245
246                     <!-- Area where all the messages will appear -->
247                     <div id="chat-messages" class="chat-messages"></div>
248
249                     <!-- Bottom section where you type and send messages -->
250                     <div class="chat-input">
251                         <!-- Button to open emoji picker -->
252                         <button onclick="showEmojiPicker()">😊</button>
253
254                         <!-- Input box to type your message -->
255                         <input type="text" id="message-input" placeholder="Type a message..." disabled>
256
257                         <!-- Button to send the message -->
258                         <button onclick="sendMessage()" id="send-btn" disabled>Send</button>
259                     </div>
260
261                     <!-- Emoji picker popup (hidden by default) -->
262                     <div id="emoji-picker" class="emoji-picker" style="display:none;">
263                         <!-- Each emoji is clickable to insert it into the message -->
264                         <span onclick="insertEmoji('😊')">😊</span>
265                         <span onclick="insertEmoji('😂')">😂</span>
266                         <span onclick="insertEmoji('❤️')">❤️</span>
267                         <span onclick="insertEmoji('👍')">👍</span>
268                         <span onclick="insertEmoji('🎉')">🎉</span>
269                         <span onclick="insertEmoji('🔥')">🔥</span>
270                         <span onclick="insertEmoji('😎')">😎</span>
271                         <span onclick="insertEmoji('🤗')">🤗</span>
272                     </div>
273                 </div>
274             </div>
275
276             <!-- Theme selector popup (hidden by default) -->
277             <div id="theme-selector" class="theme-selector" style="display:none;">
278                 <!-- Title -->
279                 <h4>Select Theme</h4>
280
281                 <!-- Buttons to choose different chat backgrounds -->
282                 <button onclick="setTheme('default')">Default</button>
283                 <button onclick="setTheme('doodle')">Doodle</button>
284                 <button onclick="setTheme('gradient')">Gradient</button>
285                 <button onclick="setTheme('matrix')">Matrix</button>
286                 <button onclick="setTheme('sunset')">Sunset</button>
287                 <button onclick="setTheme('ocean')">Ocean</button>
288             </div>
289         </div>
290     </div>
291
292     <!-- Load the JavaScript file that makes everything interactive -->
293     <script src="js/dashboard.js"></script>
294 </body>
295 </html>
```

Figure 29dashboard.php

```php
<?php
// Bring in the database connection file so we can talk to the database
require_once 'config/database.php';

// Bring in the helper functions file that has useful tools we'll need
require_once 'includes/functions.php';

// Check if the user is logged in - if not, send them to the login page
check_login();

// Get the current logged-in user's information from the database
$user = get_user_info($conn, $_SESSION['user_id']);

// Prepare a query to get all posts from the database
$stmt = $conn->prepare("
   SELECT p.*, u.username, u.full_name, u.profile_pic,
   (SELECT COUNT(*) FROM likes WHERE post_id = p.id) as likes_count,
   (SELECT COUNT(*) FROM comments WHERE post_id = p.id) as comments_count,
   (SELECT COUNT(*) FROM likes WHERE post_id = p.id AND user_id = ?) as user_liked
   FROM posts p
   JOIN users u ON p.user_id = u.id
   ORDER BY p.created_at DESC
");

// Run the query and pass in the current user's ID to check if they liked each post
$stmt->execute([$_SESSION['user_id']]);

// Get all the posts and save them in a variable called $posts
$posts = $stmt->fetchAll();
?>

<!DOCTYPE html>
<html lang="en">
<head>
   <!-- Tell the browser this page uses UTF-8 characters -->
   <meta charset="UTF-8">

   <!-- Make the page look good on phones and tablets -->
   <meta name="viewport" content="width=device-width, initial-scale=1.0">

   <!-- Set the title that shows in the browser tab -->
   <title>Social Space - Dashboard</title>

   <!-- Load the main style sheet for the whole site -->
   <link rel="stylesheet" href="css/style.css">

   <!-- Load the dashboard-specific style sheet -->
   <link rel="stylesheet" href="css/dashboard.css">
```

```
</head>
<body>
  <!-- This is the navigation bar at the top of the page -->
  <nav class="navbar">
    <!-- Container to hold everything in the navbar -->
    <div class="nav-container">
      <!-- The Social Space logo/title -->
      <h1 class="nav-logo">Social Space</h1>

      <!-- Search bar section -->
      <div class="nav-search">
        <!-- Input box where users can type to search for friends -->
        <input type="text" id="search-users" placeholder="Search friends..." autocomplete="off">

        <!-- This div will show the search results when user types -->
        <div id="search-results" class="search-results"></div>
      </div>

      <!-- Navigation menu with all the buttons -->
      <div class="nav-menu">
        <!-- Button to create a new post -->
        <button onclick="showCreatePost()" class="nav-btn">➕ Post</button>

        <!-- Button to see friend requests -->
        <button onclick="showFriendRequests()" class="nav-btn">👥 Friends</button>

        <!-- Button to open messages -->
        <button onclick="showMessages()" class="nav-btn">💬 Messages</button>

        <!-- Button to go to user's profile page -->
        <button onclick="location.href='profile.php'" class="nav-btn">👤 Profile</button>

        <!-- Link to log out of the account -->
        <a href="logout.php" class="nav-btn">Logout</a>
      </div>
    </div>
  </nav>
```

```
<!-- Main container that holds all the content on the page -->
   <div class="main-container">
      <!-- Container specifically for the posts feed -->
      <div class="feed-container">
         <!-- This div will contain all the posts -->
         <div id="posts-feed">
            <!-- Loop through each post and display it -->
            <?php foreach ($posts as $post): ?>
               <!-- Each post is wrapped in this card -->
               <div class="post-card" data-post-id="<?php echo $post['id']; ?>">
                  <!-- Header section of the post (shows who posted it) -->
                  <div class="post-header">
                     <!-- User information section -->
                     <div class="post-user">
                        <!-- Show the user's profile picture -->
                        <img src="uploads/profiles/<?php echo htmlspecialchars($post['profile_pic']); ?>" alt="Profile"
class="post-avatar">

                        <!-- User's name and time posted -->
                        <div>
                           <!-- Display the user's full name -->
                           <h4><?php echo htmlspecialchars($post['full_name']); ?></h4>

                           <!-- Show how long ago the post was created (like "2 hours ago") -->
                           <span class="post-time"><?php echo time_elapsed($post['created_at']); ?></span>
                        </div>
                     </div>

                     <!-- If this is the current user's post, show delete button -->
                     <?php if ($post['user_id'] == $_SESSION['user_id']): ?>
                        <!-- Delete button (trash can icon) -->
                        <button class="delete-btn" onclick="deletePost(<?php echo $post['id']; ?>)">🗑️ </button>
                     <?php endif; ?>
                  </div>

<!-- The actual content of the post -->
                  <div class="post-content">
                     <!-- Display the post text (nl2br makes line breaks work) -->
                     <p><?php echo nl2br(htmlspecialchars($post['content'])); ?></p>

                     <!-- If the post has an image, show it -->
                     <?php if ($post['image']): ?>
                        <img src="uploads/posts/<?php echo htmlspecialchars($post['image']); ?>" alt="Post" class="post-
image">
                     <?php endif; ?>
                  </div>
```

```
<!-- Stats showing number of likes and comments -->
                <div class="post-stats">
                  <!-- Show how many likes this post has -->
                  <span><?php echo $post['likes_count']; ?> likes</span>

                  <!-- Show how many comments this post has -->
                  <span><?php echo $post['comments_count']; ?> comments</span>
                </div>

                <!-- Action buttons (Like, Comment, Share) -->
                <div class="post-actions">
                  <!-- Like button - add 'liked' class if user already liked it -->
                  <button onclick="likePost(<?php echo $post['id']; ?>)" class="action-btn <?php echo
$post['user_liked'] ? 'liked' : ''; ?>">
                    ❤️ Like
                  </button>

                  <!-- Comment button - opens the comments section -->
                  <button onclick="toggleComments(<?php echo $post['id']; ?>)" class="action-btn">💬
Comment</button>

                  <!-- Share button - lets users share the post -->
                  <button onclick="sharePost(<?php echo $post['id']; ?>)" class="action-btn">🔗 Share</button>
                </div>

                <!-- Comments section (hidden by default) -->
                <div class="comments-section" id="comments-<?php echo $post['id']; ?>" style="display:none;">
                  <!-- This div will be filled with comments using JavaScript -->
                  <div class="comments-list"></div>

                  <!-- Input area to write a new comment -->
                  <div class="comment-input">
                    <!-- Text box to type the comment -->
                    <input type="text" placeholder="Write a comment..." id="comment-input-<?php echo $post['id'];
?>">

                    <!-- Button to send the comment -->
                    <button onclick="addComment(<?php echo $post['id']; ?>)">Send</button>
                  </div>
                </div>
              </div>
            <?php endforeach; ?>
          </div>
        </div>
      </div>
```

```
<!-- Modal/popup for creating a new post -->
  <div id="create-post-modal" class="modal">
    <!-- Content inside the modal -->
    <div class="modal-content">
      <!-- X button to close the modal -->
      <span class="close" onclick="closeModal('create-post-modal')">&times;</span>

      <!-- Title of the modal -->
      <h2>Create Post</h2>

      <!-- Form to submit a new post -->
      <form id="create-post-form" enctype="multipart/form-data">
        <!-- Text area where user types their post -->
        <textarea name="content" placeholder="What's on your mind?" required></textarea>

        <!-- File input to upload an image with the post -->
        <input type="file" name="image" accept="image/*">

        <!-- Submit button to post -->
        <button type="submit" class="btn-primary">Post</button>
      </form>
    </div>
  </div>

  <!-- Modal/popup for viewing friend requests -->
  <div id="friend-requests-modal" class="modal">
    <!-- Content inside the modal -->
    <div class="modal-content">
      <!-- X button to close the modal -->
      <span class="close" onclick="closeModal('friend-requests-modal')">&times;</span>

      <!-- Title of the modal -->
      <h2>Friend Requests</h2>

      <!-- This div will be filled with friend requests using JavaScript -->
      <div id="friend-requests-list"></div>
    </div>
  </div>

<!-- Modal/popup for messaging friends -->
  <div id="messages-modal" class="modal">
    <!-- Content inside the modal (wider for chat layout) -->
    <div class="modal-content messages-modal-content">
      <!-- X button to close the modal -->
      <span class="close" onclick="closeModal('messages-modal')">&times;</span>

      <!-- Container for the messaging interface -->
      <div class="messages-container">
        <!-- Left side: List of friends to chat with -->
        <div class="friends-list">
          <!-- Title -->
          <h3>Friends</h3>
```

```
<!-- This div will be filled with friend list using JavaScript -->
            <div id="friends-list"></div>
        </div>

        <!-- Right side: Chat area -->
        <div class="chat-area">
            <!-- Top of chat showing who you're talking to -->
            <div class="chat-header">
                <!-- Name of the friend you're chatting with -->
                <h3 id="chat-username">Select a friend to chat</h3>

                <!-- Button to change chat theme (hidden until you select a friend) -->
                <button id="theme-btn" onclick="showThemeSelector()" style="display:none;">🎨</button>

                <!-- Button to delete the entire chat history (hidden until you select a friend) -->
                <button id="delete-chat-btn" onclick="deleteChat()" style="display:none;">🗑️</button>
            </div>

            <!-- Area where all the messages will appear -->
            <div id="chat-messages" class="chat-messages"></div>

            <!-- Bottom section where you type and send messages -->
            <div class="chat-input">
                <!-- Button to open emoji picker -->
                <button onclick="showEmojiPicker()">😊</button>

                <!-- Input box to type your message -->
                <input type="text" id="message-input" placeholder="Type a message..." disabled>

                <!-- Button to send the message -->
                <button onclick="sendMessage()" id="send-btn" disabled>Send</button>
            </div>

            <!-- Emoji picker popup (hidden by default) -->
            <div id="emoji-picker" class="emoji-picker" style="display:none;">
                <!-- Each emoji is clickable to insert it into the message -->
                <span onclick="insertEmoji('😊')">😊</span>
                <span onclick="insertEmoji('😂')">😂</span>
                <span onclick="insertEmoji('❤️')">❤️</span>
                <span onclick="insertEmoji('👍')">👍</span>
                <span onclick="insertEmoji('🎉')">🎉</span>
                <span onclick="insertEmoji('🔥')">🔥</span>
                <span onclick="insertEmoji('😎')">😎</span>
                <span onclick="insertEmoji('🤗')">🤗</span>
            </div>
        </div>
    </div>

    <!-- Theme selector popup (hidden by default) -->
    <div id="theme-selector" class="theme-selector" style="display:none;">
        <!-- Title -->
        <h4>Select Theme</h4>
```

```
<!-- Buttons to choose different chat backgrounds -->
            <button onclick="setTheme('default')">Default</button>
            <button onclick="setTheme('doodle')">Doodle</button>
            <button onclick="setTheme('gradient')">Gradient</button>
            <button onclick="setTheme('matrix')">Matrix</button>
            <button onclick="setTheme('sunset')">Sunset</button>
            <button onclick="setTheme('ocean')">Ocean</button>
        </div>
      </div>
   </div>

   <!-- Load the JavaScript file that makes everything interactive -->
   <script src="js/dashboard.js"></script>
</body>
</html>
```

```php
<?php
// Bring in the database connection file so we can talk to the database
require_once 'config/database.php';

// Bring in the helper functions we need for this page
require_once 'includes/functions.php';

// Check if user is already logged in
if (isset($_SESSION['user_id'])) {
    // If they're logged in, send them to the dashboard instead
    header("Location: dashboard.php");
    exit(); // Stop the rest of the code from running
}

// Create a variable to store any error messages (starts empty)
$error = '';

// Check if the form was submitted (user clicked the Login button)
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Get the username from the form and clean it up
    $username = sanitize_input($_POST['username']);

    // Get the password from the form (don't clean this one, we need it as-is)
    $password = $_POST['password'];

    // Check if either field is empty
    if (empty($username) || empty($password)) {
        // Show an error message if they forgot to fill something in
        $error = "All fields are required";
    } else {
        // Prepare a query to look for this user in the database
        $stmt = $conn->prepare("SELECT id, username, password FROM users WHERE username = ? OR email = ?");

        // Run the query - check if username OR email matches what they typed
        $stmt->execute([$username, $username]);

        // Get the user's information from the database
        $user = $stmt->fetch();

        // Check if we found a user AND if the password is correct
        if ($user && password_verify($password, $user['password'])) {
            // Password is correct! Save their user ID in the session
            $_SESSION['user_id'] = $user['id'];

            // Also save their username in the session
            $_SESSION['username'] = $user['username'];

            // Send them to the dashboard
            header("Location: dashboard.php");
            exit(); // Stop the code here
        } else {
            // Username or password is wrong - show error message
            $error = "Invalid username or password";
        }
    }
}
```

Figure 30index.php

```
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Tell the browser this page uses UTF-8 characters -->
    <meta charset="UTF-8">

    <!-- Make the page look good on phones and tablets -->
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!-- Set the title that shows in the browser tab -->
    <title>Social Space - Login</title>

    <!-- Load the main stylesheet for styling this page -->
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <!-- Canvas for the animated particle background effect -->
    <canvas id="particles-canvas"></canvas>

    <!-- Container that holds the login box in the center of the page -->
    <div class="auth-container">
        <!-- The actual login box -->
        <div class="auth-box">
            <!-- Big "Social Space" logo at the top -->
            <h1 class="logo">Social Space</h1>

            <!-- Tagline text under the logo -->
            <p class="tagline">Connect with friends around the world</p>

            <!-- Only show error message if there is one -->
            <?php if ($error): ?>
                <!-- Red box showing the error message -->
                <div class="error-message"><?php echo $error; ?></div>
            <?php endif; ?>

            <!-- The login form - when submitted, it posts to this same page -->
            <form method="POST" action="">
                <!-- Username/Email input field -->
                <div class="input-group">
                    <!-- Input box where user types their username or email -->
                    <input type="text" name="username" placeholder="Username or Email" required>
                </div>

                <!-- Password input field with eye icon -->
                <div class="input-group password-group">
                    <!-- Input box where user types their password (hidden by default) -->
                    <input type="password" id="password" name="password" placeholder="Password" required>

                    <!-- Eye icon that lets you show/hide the password -->
                    <span class="toggle-password" onclick="togglePassword('password')"> </span>
                </div>

                <!-- Login button -->
                <button type="submit" class="btn-primary">Login</button>
            </form>

            <!-- Link to registration page for new users -->
            <p class="switch-auth">Don't have an account? <a href="register.php">Sign up</a></p>
        </div>
    </div>

    <!-- Load the JavaScript file that makes the particle animation work -->
    <script src="js/particles.js"></script>

    <!-- Load the JavaScript file that makes the password eye icon work -->
    <script src="js/auth.js"></script>
</body>
</html>
```

Figure 31index.php

```php
<?php
// Bring in the database connection file so we can talk to the database
require_once 'config/database.php';

// Bring in the helper functions we need for this page
require_once 'includes/functions.php';

// Check if user is already logged in
if (isset($_SESSION['user_id'])) {
   // If they're logged in, send them to the dashboard instead
   header("Location: dashboard.php");
   exit(); // Stop the rest of the code from running
}

// Create a variable to store any error messages (starts empty)
$error = '';

// Check if the form was submitted (user clicked the Login button)
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
   // Get the username from the form and clean it up
   $username = sanitize_input($_POST['username']);

   // Get the password from the form (don't clean this one, we need it as-is)
   $password = $_POST['password'];

   // Check if either field is empty
   if (empty($username) || empty($password)) {
      // Show an error message if they forgot to fill something in
      $error = "All fields are required";
   } else {
      // Prepare a query to look for this user in the database
      $stmt = $conn->prepare("SELECT id, username, password FROM users WHERE username = ? OR email = ?");

      // Run the query - check if username OR email matches what they typed
      $stmt->execute([$username, $username]);

      // Get the user's information from the database
      $user = $stmt->fetch();

      // Check if we found a user AND if the password is correct
      if ($user && password_verify($password, $user['password'])) {
         // Password is correct! Save their user ID in the session
         $_SESSION['user_id'] = $user['id'];
```

(PHP.net, 2025)

```
 // Also save their username in the session
        $_SESSION['username'] = $user['username'];

        // Send them to the dashboard
        header("Location: dashboard.php");
        exit(); // Stop the code here
      } else {
        // Username or password is wrong - show error message
        $error = "Invalid username or password";
      }
    }
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Tell the browser this page uses UTF-8 characters -->
  <meta charset="UTF-8">

  <!-- Make the page look good on phones and tablets -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <!-- Set the title that shows in the browser tab -->
  <title>Social Space - Login</title>

  <!-- Load the main stylesheet for styling this page -->
  <link rel="stylesheet" href="css/style.css">
</head>
<body>
  <!-- Canvas for the animated particle background effect -->
  <canvas id="particles-canvas"></canvas>

  <!-- Container that holds the login box in the center of the page -->
  <div class="auth-container">
    <!-- The actual login box -->
    <div class="auth-box">
      <!-- Big "Social Space" logo at the top -->
      <h1 class="logo">Social Space</h1>

      <!-- Tagline text under the logo -->
      <p class="tagline">Connect with friends around the world</p>

      <!-- Only show error message if there is one -->
      <?php if ($error): ?>
        <!-- Red box showing the error message -->
        <div class="error-message"><?php echo $error; ?></div>
      <?php endif; ?>
```

```
<!-- The login form - when submitted, it posts to this same page -->
        <form method="POST" action="">
          <!-- Username/Email input field -->
          <div class="input-group">
            <!-- Input box where user types their username or email -->
            <input type="text" name="username" placeholder="Username or Email" required>
          </div>

          <!-- Password input field with eye icon -->
          <div class="input-group password-group">
            <!-- Input box where user types their password (hidden by default) -->
            <input type="password" id="password" name="password" placeholder="Password" required>

            <!-- Eye icon that lets you show/hide the password -->
            <span class="toggle-password" onclick="togglePassword('password')"> 👁 </span>
          </div>

          <!-- Login button -->
          <button type="submit" class="btn-primary">Login</button>
        </form>

        <!-- Link to registration page for new users -->
        <p class="switch-auth">Don't have an account? <a href="register.php">Sign up</a></p>
      </div>
    </div>

    <!-- Load the JavaScript file that makes the particle animation work -->
    <script src="js/particles.js"></script>

    <!-- Load the JavaScript file that makes the password eye icon work -->
    <script src="js/auth.js"></script>
</body>
</html>
```

```php
<?php
// Start the session so we can access the user's login information
session_start();

// Destroy the session - this logs the user out by deleting all their login info
session_destroy();

// Send the user back to the login page
header("Location: index.php");

// Stopping the code here - don't run anything else
exit();
?>
```

Figure 32logout.php


```
<?php
// Start the session so we can access the user's login information
session_start();

// Destroy the session - this logs the user out by deleting all their login info
session_destroy();

// Send the user back to the login page
header("Location: index.php");

// Stopping the code here - don't run anything else
exit();
?>
```


Please note that profile and register will not be included because it is long as it will be in the zip file.

Question 2

2.1 CEO Fraud: Attack Vendors and Preventative Cybersecurity Strategies

- **Phishing CEO fraud**: A cybercriminal will craft carefully researched phishing emails targeting CEOs of various companies to deceive them into clicking a link that leads to a harmful website or malware file. They can subsequently access the victim's account and contact list to send emails, deceiving other downstream employees into transferring funds or confidential information. (terranovasecurity, 2023)
- **Spear phishing**: In this attack, cybercriminals collect data about their victims online prior to dispatching a meticulously crafted email, portraying themselves as a company or person they interact with or mentioning events or projects the victims have been involved in. Cyber perpetrators then attempt to deceive the receiver into supplying the sought-after details, enabling them to execute future offenses. (terranovasecurity, 2023)
- **Social engineering:** In a social engineering scheme, the scammers utilize a tailored email, text message, or phone call to build rapport with the victim and persuade them to provide sensitive information or authorize a wire transfer. (terranovasecurity, 2023)
- **Executive whaling**: A form of cyber danger in which a criminal pretends to be an executive and tries to coerce employees into rapidly providing information, uploading tax files, or transferring money without confirming the request with another co-worker. (terranovasecurity, 2023)

How to prevent CEO Fraud:

1. **Educate executives and their teams on CEO fraud tactics**: Utilize complimentary phishing simulation tools to train employees or recognizing phishing, social engineering, and CEO fraud efforts. In this manner, they are less prone to being deceived into giving away personal or confidential information. (terranovasecurity, 2023)
2. **Take advantage of security awareness training**: The optimal safeguard is making sure that these attack threats remain a priority for employees. Conduct frequent seminars and training sessions to keep employees informed about the latest tactics used by scammers. Establish internal cyber security advocates dedicated to maintaining your organization's cyber safety. (terranovasecurity, 2023)
3. **Monitor employee security and fraud awareness**: Consistently evaluate employee awareness through phishing simulations and assist struggling employees with CEO fraud training modules to teach, educate and modify essential behaviors. (terranovasecurity, 2023)
4. **Provide ongoing security campaigns**: Provide employees with continuous communication initiatives regarding security best practices, CEO fraud, and various social engineering threats, which include enforcing robust password policies and informing employees about the danger of clicking on questionable URLs and attachments. (terranovasecurity, 2023)

5. **Create network access rules to limit the use of personal devices**: Create a solid network structure to limit the use of personal devices in your setting and manage the ways employees exchange information beyond your corporate network. (terranovasecurity, 2023)

6. **Update your infrastructure**: Make certain that all applications, operating systems, network utilities, and internal software remain current and protected. This involves setting up malware defenses and anti-spam applications on endpoints. (terranovasecurity, 2023)

2.2 Developing a Comprehensive IT Security Policy for Sam Dunn Enterprises

1. Introduction

This IT Security Policy aims to safeguard Sam Dunn Enterprises against cyber risks, specifically targeting Business Email Compromise (BEC) and CEO fraud. CEO fraud happens when attackers mimic executives to deceive employees into transferring money or revealing confidential information. Enforcing this policy guarantees that employees are aware of their duties, systems remain secure, and risks are reduced. (StaySafeOnline, 2023)

2. Objectives

The primary goals of this policy are:

- Restrict access to company systems and data to authorized personnel only.
- Safeguard confidential financial, operational, and client data.
- Train staff to recognize phishing, spoofing, and social engineering threats.
- Create protocols for validating requests, particularly concerning finances and sensitive data.
- Observe. Identify, and react to questionable actions swiftly.
- Adhere to cybersecurity rules and leading industry standards. (Cisa, 2023)

3. Scope

This policy is relevant to:

- Every staff member of Sam Dunn Enterprises.
- Contractors and advisors who utilize company systems.
- External vendors who have access to corporate data or email systems.
- All devices owned by the company, email accounts, and network systems.

4. Roles & Responsibilities

IT Department:

- Manage email security solutions, firewalls, and intrusion detection systems.
- Apply security updates and patches across all servers, endpoints, and applications.
- Watch unusual behavior and notify management of possible violations. (NetDiligence, 2024)

Employees:

- Immediately report any emails, requests, or attachments that seem suspicious.
- Adhere to verification protocols for monetary transactions and private inquiries.
- Finish all necessary cybersecurity training and confirm comprehension of this. (Cisa, 2023)

Management:

- Ensure compliance with this policy throughout all divisions.
- Make certain that employees understand cybersecurity threats and optimal practices.
- Perform routine inspections to confirm adherence.

5. Email Security Measures

Email Authentication:

- Utilize SPF, DKIM, and DMARC protocols to authenticate incoming emails and stop domain spoofing.
- Consistently check authentication reports to detect deceptive emails. (StaySafeOnline, 2023)

Email Filtering & Anti-Phishing Tools:

- Utilize sophisticated email filtering tools to prevent potentially harmful attachments and links.
- Emails identified as potentially harmful are automatically quarantined. (Redsift, 2025)

Alerts for External Emails:

- Set up email systems to distinctly identify emails from external sources to notify employees.
- Instruct employees to handle any unusual requests from outside sources carefully. (NCUA, 2021)

6. Access Management and user Verification

Multi-Factor Authentication:

- Mandate MFA for every email account, VPN access, and systems that contain sensitive information.
- Regularly update MFA methods to ensure security is maintained. (StaySafeOnline, 2023)

Password Policies:

- Implement strict password criteria (at least 15 characters including letters, digits, and symbols).
- Mandate password changes every 90 days and disallows the use of previous passwords. (LastPass, 2024)

User Access Management:

- Restrict access to financial and confidential information solely to those employees who need it.
- Consistently assess user access rights and revoke access for former staff or changes in roles.

7. **Employee Training and Awareness**
    - Implement yearly cybersecurity awareness training that emphasizes phishing, spoofing, and CEO fraud.
    - Create phishing email simulations to assess employee's reactions.
    - Instruct staff in the correct ways to report any suspicious behavior. (Cisa, 2023)

8. **Verification of Financial Transactions**
    - Mandate dual verification (email + verbal or face-to-face confirmation) for all wire transfers exceeding a specified limit.
    - All typical or urgent requests must be confirmed by a minimum of two authorized individuals.
    - Maintain documentation of verification for auditing reasons. (Sjouwerman , 2020)

9. **Protected Communication Pathways**
    - Utilize encrypted messaging or secure email for confidential communication and sensitive transactions.
    - Do not transmit financial details or passwords through standard email. (LastPass, 2024)

10. **Supervising, Inspecting, and Recording**
    - Observe network traffic and email records for any irregular behavior.
    - Perform routine internal evaluations of financial transactions and access records.
    - Set up notifications for unusual activities, like repeated unsuccessful login attempts or abnormal transaction trends. (Redsift, 2025)

11. **Emergency Response Strategy**

    In case of a suspected BEC or CEO fraud incident:
    - Promptly inform the IT department and management.
    - Separate impacted accounts or devices to stop additional access.
    - Confirm the authenticity of any requested transactions or information.
    - Inform the police or appropriate authorities if needed.
    - Record the event, detailing the actions taken and the outcome.
    - Assess and revise security protocols to avoid future incidents. (NetDiligence, 2024)

12. Management of Vendors and Security of Third Parties

- Make certain that all external vendors adhere to the organization's cybersecurity requirements.
- Incorporate cybersecurity provisions in agreements to guarantee vendor responsibility.
- Regularly assessor vendor access and permissions to reduce risk. (NCUA, 2021)

13. Policy Enforcement

- Failure to adhere to this policy could lead to disciplinary measures, which may include the termination of employment or contracts.
- Employees are required to sign a form confirming they have reviewed and comprehended this policy.

14. Review and Update of Policies

- This policy will undergo a review at least annually and be revised as necessary to respond to new threats.
- All employees will be informed of updates, and training will be modified as needed.

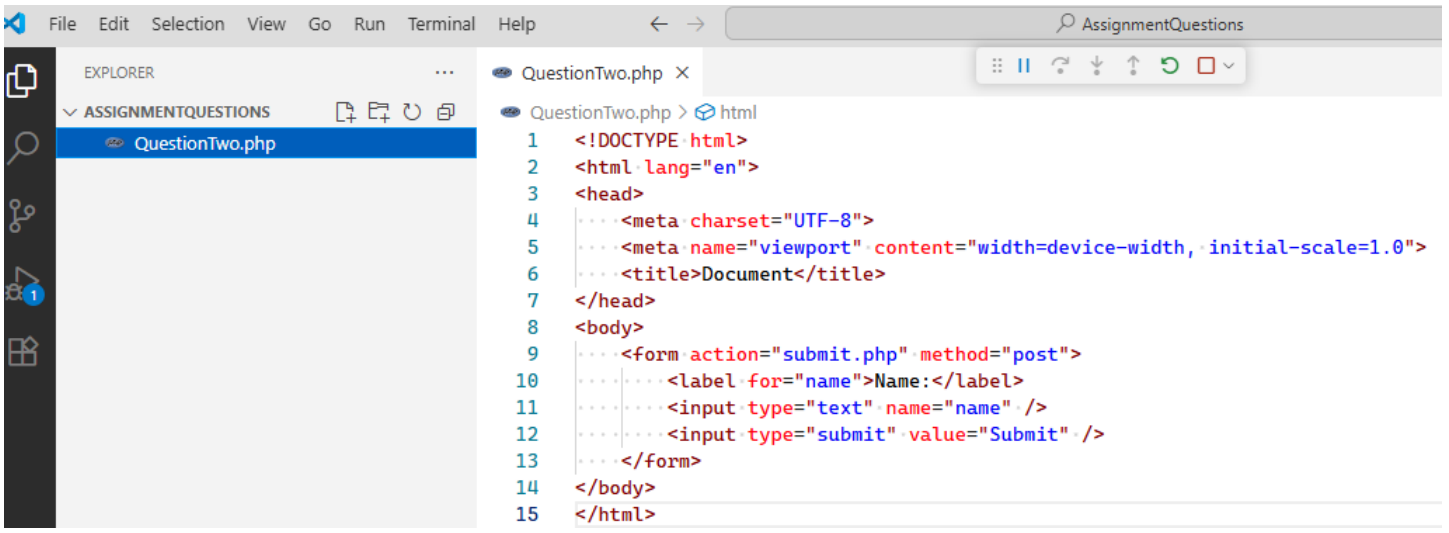
2.3 Understanding PHP Form Handling: Key Considerations for Real-World Applications

Handling forms in PHP is an essential part of web development. It encompasses the procedure of gathering, handling, and overseeing form data provided by users via web forms. These forms can range from simple, such as a contact form, to complex, like multi-step forms for online applications. Grasping PHP Form Handling is crucial for developing interactive, user-friendly, and secure web applications. (eflair, 2024)

Forms serve as a key method for users to engage with websites. They enable users to enter information, which the server processes to execute different tasks, including creating accounts, submitting reviews, or making purchases. PHP, as a strong server-side scripting language, offers solid capabilities for managing form data effectively and securely. (eflair, 2024)

PHP can manage data sent through HTML forms. The PHP superglobals $_GET and $_POST facilitate the processing of form data through GET and POST HTTP request methods. Start by building a basic HTML form. This will contain one text input area and a submit button: (Sebhastian, 2022)

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
</head>
<body>
   <form action="submit.php" method="post">
      <label for="name">Name:</label>
      <input type="text" name="name" />
      <input type="submit" value="Submit" />
   </form>
</body>
</html>
```



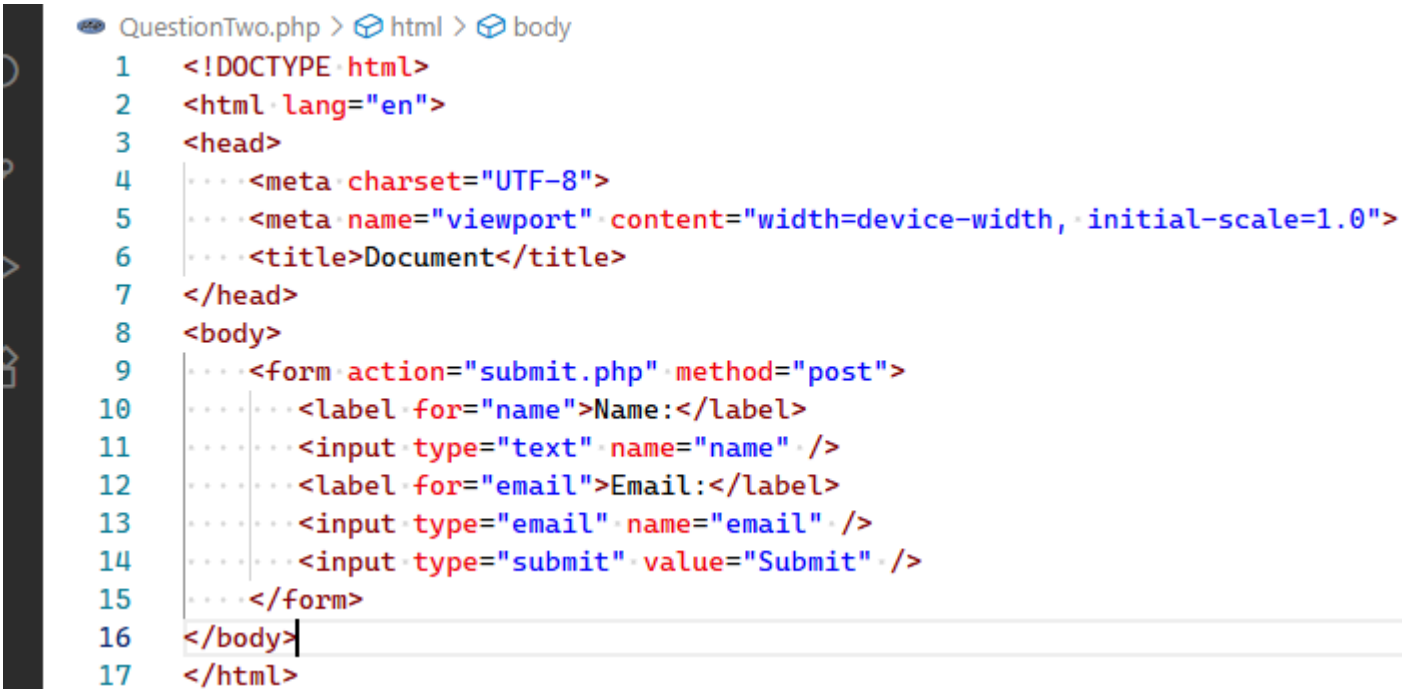
The action attribute indicates the HTTP request method employed to send the form data. In this instance, we are utilizing the POST method, which is safer compared to the GET method.

Below is the “submit.php” script and the form data will be sent through the $_POST: (Sebhastian, 2022)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    Hello, <?php echo $_POST["name"]; ?>
</body>
</html>
```

This form has a few more input fields, so that way the data can be easily accessed using the $_POST:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form action="submit.php" method="post">
        <label for="name">Name:</label>
        <input type="text" name="name" />
        <label for="email">Email:</label>
        <input type="email" name="email" />
        <input type="submit" value="Submit" />
    </form>
</body>
</html>
```



To process form data using the GET method, you can utilize the $_GET superglobal array similarly to how you would use the $_POST array for managing form data with the POST method. Below is the “GET” method: (Sebhastian, 2022)

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<body>

  <form action="submit.php" method="GET">

    <label for="name">Name:</label>

    <input type="text" name="name" />

    <label for="email">Email:</label>

    <input type="email" name="email" />

    <input type="submit" value="Submit" />

  </form>
```

</body>

</html>

This is the code to submit the form:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

Hello, <?php echo $_GET["name"]; ?>

</body>

</html>

The GET method is not as secure as the POST method since the form data is part of the URL and can be seen by the user. The POST method transmits the form data within the request body, making it hidden from the user. For this reason, the POST method is usually advised for managing sensitive information, like passwords or credit card details. The GET method is suitable for non-sensitive information like a search query. When you conduct a Google search, the terms you entered the search box can be seen in the URL. Once you have the data in the $_POST or $_GET array, you can manipulate the data based on your needs. (Sebhastian, 2022)

The challenges involved in implementing secure and scalable form processing:

- SQL Injection: SQL injection is a security flaw that enables attackers to alter SQL statements and obtain unauthorized access to a web application's database. This happens when the user input is directly incorporated into SQL queries without adequate validation, allowing attackers to manipulate database functions. SQL injections also occur when applications take unfiltered input in areas such as login forms, search fields, or URL parameters. (Chawdhary, 2025)
- Insecure authentication and Weak Passwords: This happens when a web application does not effectively validate user identities, enabling attackers to obtain unauthorized entry. Insecure passwords, absence of multi-factor authentication (MFA), and inadequate session management all lead to authentication weaknesses. Attackers can take advantage of these vulnerabilities by using brute-force attacks, credential stuffing, and session hijacking, resulting in unauthorized access to user accounts and confidential information. (Chawdhary, 2025)
- Security misconfigurations: Security misconfigurations happen when a web application, server, or database ins incorrectly configured, exposing it to potential attacks. This may involve default passwords, visible error messages, excessive permissions, or obsolete security configurations. Intruders take advantage of these vulnerabilities to obtain unauthorized entry, retrieve confidential information, or alter system operations. Misconfigurations rank among the leading reasons for

security breaches, frequently arising from neglected settings during the development or deployment phases. (Chawdhary, 2025)

- Insecure API Endpoints: Vulnerable API endpoints reveal confidential information or capabilities to unauthorized individuals, rendering them a key target for attackers. APIs frequently enable communication among web applications, however, without adequate security measures, attackers may exploit them to reach unauthorized data, alter system functionality, or execute denial-of-service (DoS) attacks. Frequent vulnerabilities consist of inadequate authentication, unnecessary data exposure, and absence of rate limiting. (Chawdhary, 2025)

Conclusion

Working on this full-stack media application has been an eye-opening experience especially when it comes to backend development. During my first year, I learned HTML, CSS, and JavaScript for front-end development, which gave me solid coding with PHP and SQL, allowing me to understand and implement the server-side logic and database management that power a complete application.

This prompt challenged me, pushed me to think critically, and helped me discover a genuine passion for backend development. I now have a deeper understanding of how complex systems are built and maintained. Experiencing the full stack from front-end design to backend functionality has inspired me to continue learning and improving my skills, opening the pathway to creating more robust, dynamic, and interactive applications in the future.

References

Chawdhary, . G., 2025. *8 Web Application Security Issues With Solutions.* [Online]
Available at: https://www.securitycompass.com/blog/web-application-security-issues-solutions/
[Accessed Saturday 27 September 2025].
Cisa, 2023. *Teach Employees to Avoid Phishing.* [Online]
Available at: https://www.cisa.gov/secure-our-world/teach-employees-avoid-phishin
[Accessed Sunday 21 September 2025].
Developer Mozilla, 2024. *Javascript.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript
[Accessed Friday 31 October 2025].
eflair, 2024. *PHP Form Handling: A Comprehensive Guide.* [Online]
Available at: https://eflairwebtech.com/php-form-handling/
[Accessed Saturday 27 September 2025].
LastPass, 2024. *Protecting Against CEO Fraud.* [Online]
Available at: https://blog.lastpass.com/posts/ceo-fraud
[Accessed Sunday 21 September 2025].
MySQL , 2024. *MySQL 8.0 Reference Manual.* [Online]
Available at: https://dev.mysql.com/doc/refman/8.0/en/
[Accessed Friday 31 October 2025].
NCUA, 2021. *Business Email Compromise through Exploitation of Cloud-Based Email Services.* [Online]
Available at: https://ncua.gov/regulation-supervision/letters-credit-unions-other-guidance/business-email-compromise-through-exploitation-cloud-based-email-service
[Accessed Sunday 21 September 2025].
NetDiligence, 2024. *Business Email Compromise Incident Response and Containment: An Essential Guide.* [Online]
Available at: https://netdiligence.com/blog/2024/10/how-to-prevent-business-email-compromise
[Accessed Sunday 21 September 2025].
PHP.net, 2025. *PHP Manual.* [Online]
Available at: https://www.php.net/manual/en/index.php
[Accessed Friday 31 October 2025].
Redsift, 2025. *Business Email Compromise: Ultimate prevention guide.* [Online]
Available at: https://redsift.com/guides/business-email-compromise-guide
[Accessed Sunday 21 September 2025].
Sebhastian, N., 2022. *Handling form data using PHP (GET and POST).* [Online]
Available at: https://sebhastian.com/php-form-handling/
[Accessed Saturday 27 September 2025].
Sjouwerman , S., 2020. *5 Tips to Prevent CEO Fraud.* [Online]
Available at: https://blog.knowbe4.com/5-tips-to-prevent-ceo-fraud
[Accessed Sunday 21 September 2025].
StaySafeOnline, 2023. *Business Email Compromise: What It Is and How to Prevent It.* [Online]
Available at: https://www.staysafeonline.org/articles/business-email-compromise-what-it-is-and-how-to-prevent-it
[Accessed Saturday 21 September 2025].
terranovasecurity, 2023. *4 Examples of CEO Fraud and How to Prevent Them.* [Online]
Available at: https://www.terranovasecurity.com/blog/examples-of-ceo-fraud-attacks
[Accessed Sunday 21 September 2025].
World Wide Web Consortium, 2024. *CSS Snapshot 2024.* [Online]
Available at: https://www.w3.org/TR/css-2024/
[Accessed Friday 31 October 2025].