SocialQ

Project Documentation

This is the document explaining SocialQ.

Members

Ajero, Judah Jess T.
(Post Module)

Canturias, Christian G.
(Auth Module)

Guido, Darryl Adrian D.
(Feed Module)

Instructor

Mr. Armando Jr. Saguin

Course

Information Management (CC 105)
T/TH (10:00–11:00) • Lec
T/TH (13:00–14:30) • Lab

2. Table of Contents

3. Introduction

Project Background

This is a simple social media system through which users can post content among a community. The system is intended to support user generated content and discovery of content in the form of a feed. The system intends to build a simple yet interactive platform for users to see each another's activity without the complexity of messaging features.

Objectives

- Provide a platform where users can create accounts.
- Allow users to post content.
- Offer feed of all posts and all posts of a specific user.

Scope

Included Features:

- User registration and login system.
- Post creation.
- Post deletion.
- All posts feed and all posts of a specific user feed.

Excluded Features:

- Private messaging or chat system.
- Image and video uploads or multimedia stories.
- Likes and comments.
- Advanced post analytics or insights.

4. System Overview

System Architecture

The system follows a client-server architecture using a XAMPP stack (cross-platform, Apache, MariaDB, PHP). The development environment is hosted locally using XAMPP, which provides an integrated server stack for running PHP scripts and managing MariaDB (MySQL) databases. The backend uses object-oriented PHP with PDO to interact with the MariaDB database.

Components:

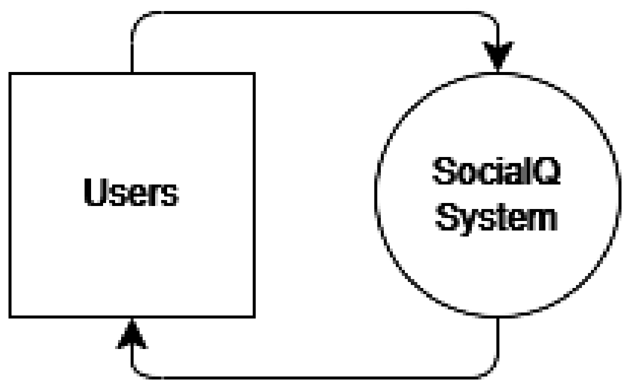
Backend (Web Server with PHP): Processes requests, interacts with the database, manages sessions, and serves content.
Database (MariaDB): Stores user information, posts, likes, comments, and follower relationships.
Make requests: We make requests in Postman directly.

Target Users

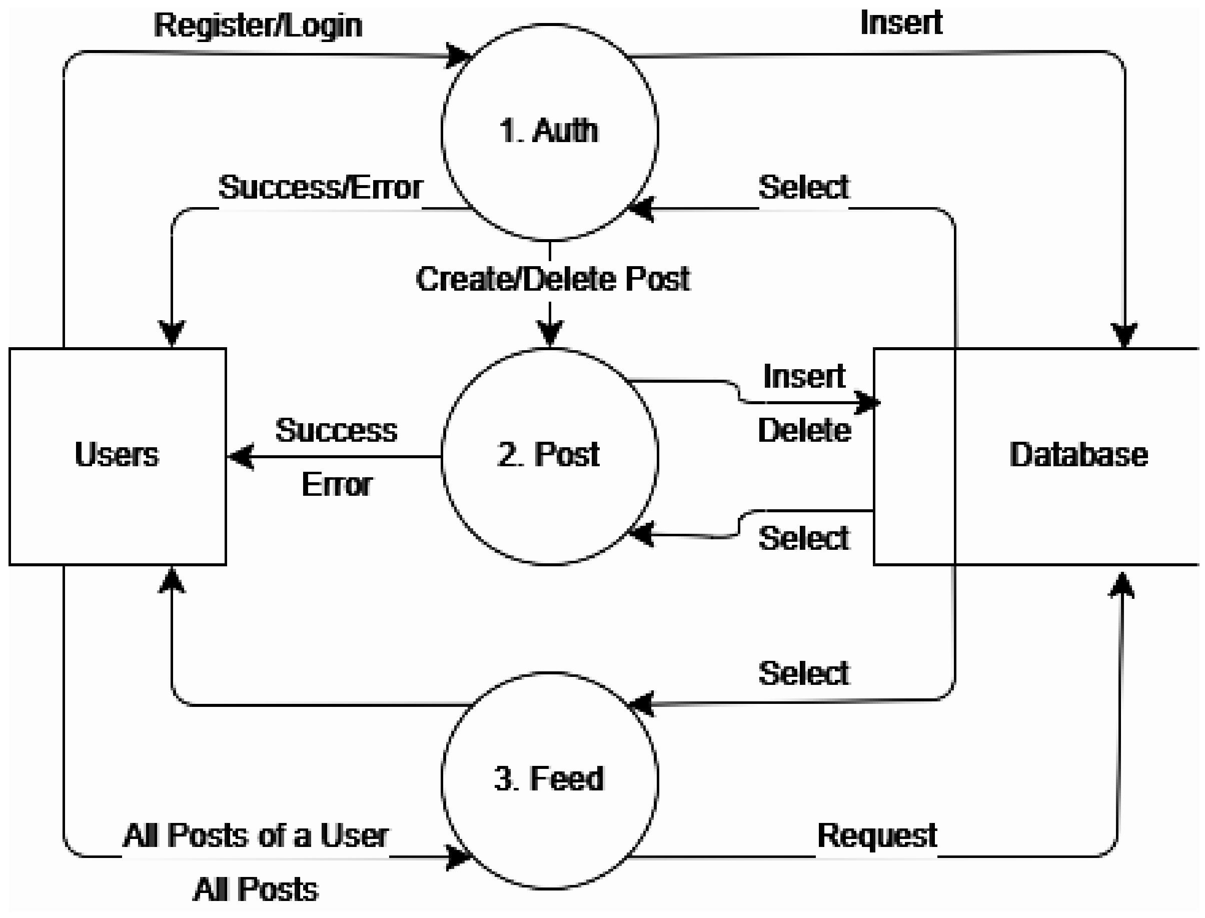
General Users
They can register, log in, post content, delete posts, view posts, etc.

5. Data Modeling

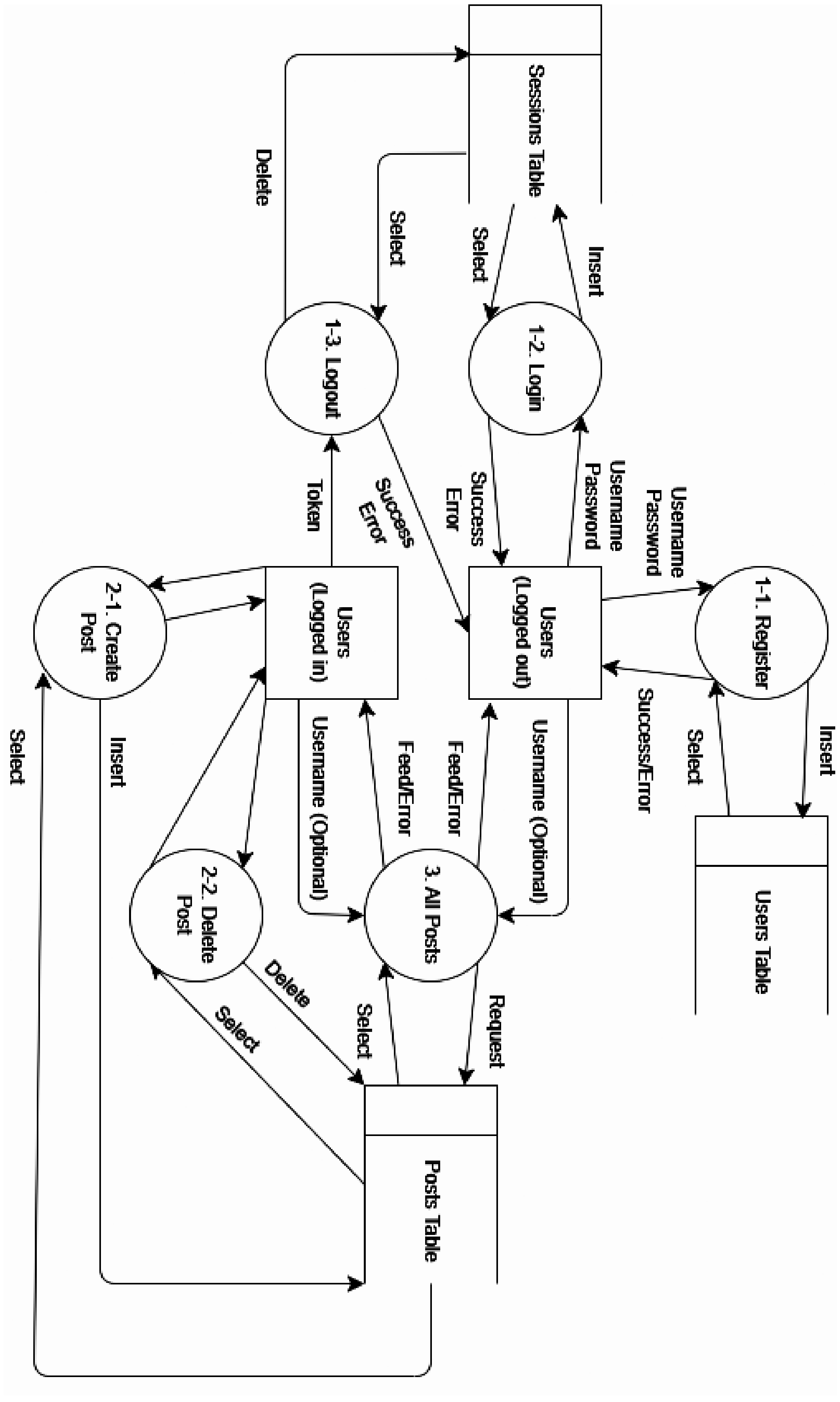
Context Diagram



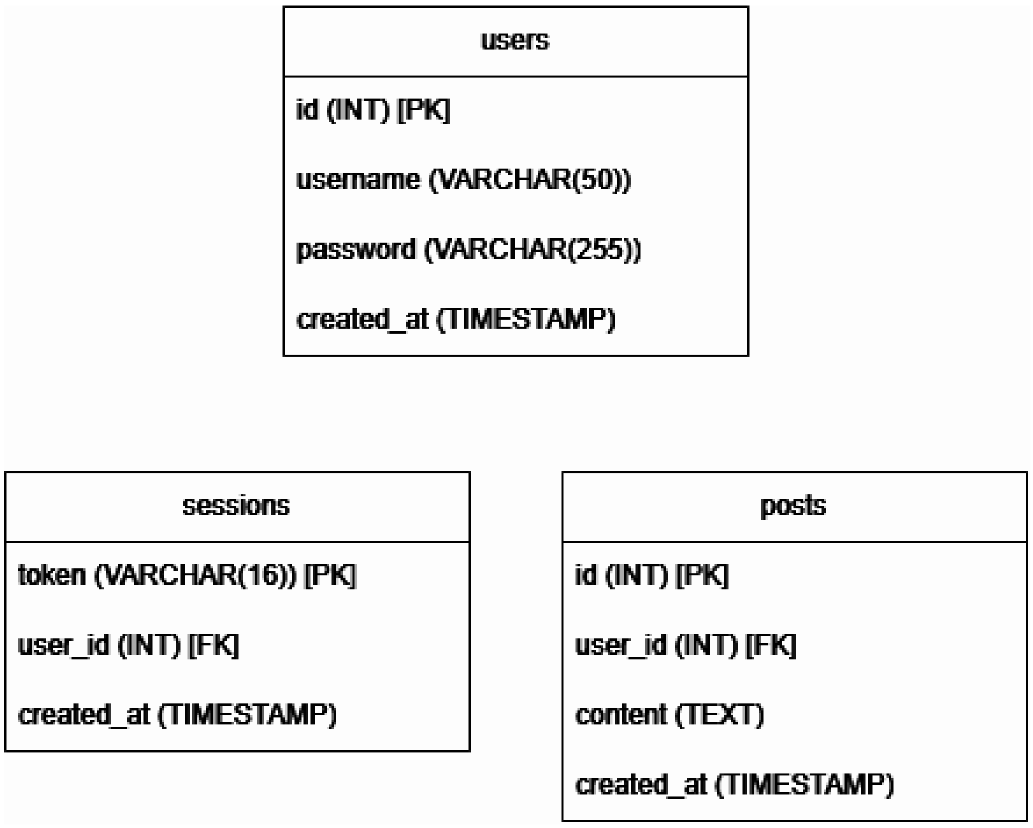
Level 0 Diagram



Level 1 Diagram



Entity Relational Diagram



Data Dictionary

Users Table

Table	Field	Type	Description
users	id	INT, PK	Unique identifier for each user
	username	VARCHAR(50)	Unique username
	password_hash	VARCHAR(255)	Hashed password
	created_at	TIMESTAMP	Account creation time

Posts Table

Table	Field	Type	Description
posts	id	INT, PK	Unique post ID
	user_id	INT, FK → users.id	Author of the post
	content	TEXT	Post content
	created_at	TIMESTAMP	Time of posting

Sessions Table

Table	Field	Type	Description
sessions	token	VARCHAR(16), PK	Unique session token
	user_id	INT, FK → users.id	User to whom the session belongs
	created_at	TIMESTAMP	When the session was created

6. Modules

Module List:

Auth Module, Post Module, Feed Module

Module Details:

Module Name: Auth (Canturias)
Description: Handles user registration, login, and logout

Functionality:

1. Register users

User_Register procedure:

```
BEGIN
  DECLARE user_count INT DEFAULT 0;

  SELECT COUNT(*) INTO user_count FROM users WHERE username = p_username;

  IF user_count > 0 THEN
    SELECT 'Error: Username already exists' AS Record_Status;
  ELSE
    INSERT INTO users (username, password, joined_date)
    VALUES (p_username, p_password, NOW());
    SELECT 'Success' AS Record_Status;
  END IF;
END
```

Prevent duplicate usernames

RegisterUser function in model

```
public static function RegisterUser($username, $password) {
    if(empty($username) || empty($password)) {
      echo json_encode(["Status" => "Error: Username and Password are
required."]);
      return;
    }
    $hashedPassword = password_hash($password, PASSWORD_DEFAULT);
    $params = [$username, $hashedPassword];

    $result = PdoMySQL::ExecuteDML_Query(
      Application::$DBase,
      "CALL User_Register(?, ?)",
      $params
    );
    echo $result;
}
```

2. Login users

User_Login procedure

```
BEGIN
    SELECT id, username, password FROM users WHERE username = p_username
LIMIT 1;
END
```

Select user’s info where usernames match

User_CreateSession procedure

```
BEGIN
    INSERT INTO sessions (user_id, token, login_date)
    VALUES (p_user_id, p_token, NOW())
    ON DUPLICATE KEY UPDATE
      token = VALUES(token),
      login_date = VALUES(login_date);
    SELECT 'Success' AS Record_Status;
END
```

Creates new session for current user

LoginUser function in model

```
public static function LoginUser($username, $password) {
    if(empty($username) || empty($password)) {
      echo json_encode(["Status" => "Error: Username and Password are
required."]);
      return;
    }
    $params = [$username];

    $userDataRaw = PdoMySQL::ExecuteDML_Query(
      Application::$DBase,
      "CALL User_Login(?)",
      $params
    );
    $userData = json_decode($userDataRaw);

    if(empty($userData)) {
      echo json_encode(["Status" => "Error: Invalid username or
password."]);
      return;
    }
    // Verify login
    $storedHash = $userData[0]->password ?? null;
    if(!$storedHash || !password_verify($password, $storedHash)) {
      echo json_encode(["Status" => "Error: Invalid username or
password."]);
      return;
    }
    // If login verified
    $token = bin2hex(random_bytes(8));
    $userId = $userData[0]->id;
    $params = [$userId, $token];

    $sessionResult = PdoMySQL::ExecuteDML_Query(
      Application::$DBase,
      "CALL User_CreateSession(?, ?)",
      $params
    );
    $sessionResultDecoded = json_decode($sessionResult);

    if(isset($sessionResultDecoded[0]->Record_Status) &&
$sessionResultDecoded[0]->Record_Status == "Success") {
      echo json_encode(["Status" => "Login successful.", "Token" =>
$token]);
    }
    else {
      echo json_encode(["Status" => "Error: Failed to create session."]);
    }
  }
```

3. Logout user

User_Logout procedure

```
User_Logout:
BEGIN
    DELETE FROM sessions WHERE token = p_token;
    SELECT 'Success' AS Record_Status;
END
```

Deletes the current session token

LogoutUser function in model

```
public static function LogoutUser($token) {
    if(empty($token)) {
      echo json_encode(["Status" => "Error: Token is required."]);
      return;
    }
    $params = [$token];

    $result = PdoMySQL::ExecuteDML_Query(
      Application::$DBase,
      "CALL User_Logout(?)",
      $params
    );
    echo $result;
  }
```

Module Name: Post (Ajero)
Description: Enables users to create and delete posts

Functionality:

1. Create posts

Post_Create procedure

```
BEGIN
    DECLARE v_user_id INT;

    SELECT user_id INTO v_user_id FROM sessions WHERE token = p_token
LIMIT 1;

    IF v_user_id IS NULL THEN
        SELECT 'Error: Invalid or expired token' AS Record_Status;
    ELSE
        INSERT INTO posts (user_id, post_content, posted_date)
        VALUES (v_user_id, p_post_content, NOW());

        SELECT 'Success: Post created' AS Record_Status;
    END IF;
END
```

Inserts the new post into the posts table with current timestamp using NOW() function.

CreatePost function in model

```
public static function CreatePost($token, $post_content)
  {
    if (empty($token) || empty($post_content))
    {
      echo json_encode(["Status" => "Error: Token and post content are
required."]);
      return;
    }

    try
    {
      $params = [$token, $post_content];
      $result = PdoMySQL::ExecuteDML_Query(Application::$DBase, "CALL
Post_Create(?, ?)", $params);

      if (!empty(trim($result)))
      {
        echo $result;
      }
      else
      {
        echo json_encode(["Status" => "Error: Failed to create post."]);
      }
    }
    catch (Exception $error)
    {
      echo json_encode(["Status" => "Error: " . $error->getMessage()]);
    }
}
```

---

2. Delete posts

Post_Delete procedure

```
BEGIN
    DECLARE v_user_id INT;
    DECLARE v_user_post_id INT;

    SELECT user_id INTO v_user_id FROM sessions WHERE token = p_token
LIMIT 1;
    SELECT user_id INTO v_user_post_id FROM posts WHERE id = p_post_id AND
user_id = v_user_id LIMIT 1;

    IF v_user_id IS NULL THEN
        SELECT 'Error: Invalid or expired token' AS Record_Status;
    ELSEIF v_user_post_id IS NULL THEN
        SELECT 'Error: This is not your post' AS Record_Status;
    ELSE
        DELETE FROM posts WHERE id = p_post_id;
        SELECT 'Success: Post deleted' AS Record_Status;
    END IF;
END
```

Deletes a post in the posts table if the user owns it.

CreatePost function in model

```
public static function DeletePost($token, $post_id)
  {
    if (empty($token) || empty($post_id))
    {
      echo json_encode(["Status" => "Error: Token and post ID are
required."]);
      return;
    }

    try
    {
      $params = [$token, $post_id];
      $result = PdoMySQL::ExecuteDML_Query(Application::$DBase, "CALL
Post_Delete(?, ?)", $params);

      $decodedResult = json_decode($result, true);
      if (!empty($decodedResult) && is_array($decodedResult))
      {
        $record = $decodedResult[0]['Record_Status'] ?? null;

        if (!empty($record))
        {
          echo json_encode(["Status" => $record]);
        }
        else
        {
          echo json_encode(["Status" => "Error: Unexpected response
format."]);
        }
      }
      else
      {
        echo json_encode(["Status" => "Error: Failed to delete post."]);
      }
    }
    catch (Exception $error)
    {
      echo json_encode(["Status" => "Error: " . $error->getMessage()]);
    }
}
```

Module Name: Feed (Guido)
Description: Enables users to create posts

Functionality:

1. View all posts of specific user

User_Post procedure

```
BEGIN
  SELECT
    p.id AS post_id,
    u.username,
    p.post_content,
    p.posted_date
  FROM posts p
  JOIN users u ON p.user_id = u.id
  WHERE u.username = in_username
  ORDER BY p.posted_date DESC;
END
```

UserPosts function in model

```
public static function UserPosts($username) {
    if(empty($username)) {
      echo json_encode(["Status" => "Error: Username parameter is
required."], JSON_UNESCAPED_UNICODE);
      return;
    }

    try {
      $result = PdoMySQL::ExecuteDML_Query(Application::$DBase, "CALL
User_Posts(?)", [$username]);

      if (!empty(trim($result))) {
        echo $result;
      }
      else {
        echo json_encode(["Status" => "Error: No posts found for this
user."], JSON_UNESCAPED_UNICODE);
      }
    }
    catch(Exception $e) {
      echo json_encode(["Status" => "Error: " . $e->getMessage()],
JSON_UNESCAPED_UNICODE);
    }
}
```

---

2. View all posts of all users

All_Posts procedure

```
BEGIN
  SELECT
    p.id AS post_id,
    u.username,
    p.post_content,
    p.posted_date
  FROM posts p
  JOIN users u ON p.user_id = u.id
  ORDER BY p.posted_date DESC;
END
```

AllPosts function in model

```
public static function AllPosts() {
    try {
      $result = PdoMySQL::ExecuteDML_Query(Application::$DBase, "CALL
All_Posts()");

      if(!empty(trim($result))) {
        echo $result;
      }
      else {
        echo json_encode(["Status" => "Error: No posts found."],
JSON_UNESCAPED_UNICODE);
      }
    }
    catch(Exception $e) {
      echo json_encode(["Status" => "Error: " . $e->getMessage()],
JSON_UNESCAPED_UNICODE);
    }
}
```

7. Transaction Entries

Transaction Entry List

- User Registration
- User Login
- Create Post
- View All Posts
- View All of a User’s Posts
- Logout

Transaction Entry Details

Transaction Name: User Registration
Description: Registers a new user
Input Data: Username, Password
Process: User_Register procedure checks for availability, then inserts
Output: Success or Username already exists status

Transaction Name: User Login
Description: Authenticates user
Input Data: Username, Password
Process: User_Login selects user info, LoginUser function verifies, User_CreateSession creates a session or Invalid username or password
Output: Success or incorrect username/password, session token

Transaction Name: Create Post
Description: Creates a new post
Input Data: Session Token, Content
Process: Post_Create inserts record into posts
Output: Success post created or invalid token

Transaction Name: Delete Post
Description: Deletes a post
Input Data: Session Token, Post ID
Process: Post_Delete checks if you own the post before deleting
Output: Success post deleted or invalid token or not allowed

Transaction Name: View All Posts
Description: Retrieves all posts from database
Input Data: None
Process: All_posts
Output: List of all posts

Transaction Name: View All of a User’s Posts
Description: Retrieves all posts of a specific user from database
Input Data: Username
Process: User_posts
Output: List of all posts from a specific user or invalid username

Transaction Name: Logout
Description: Ends session
Input Data: Token
Process: User_Login deletes session from sessions tab;e
Output: Session removed or invalid token

8. Reports

Transaction Reports:

All Posts Report

Report Details:

Report Name: All Posts Report
Description: Lists all posts made on the platform from newest to oldest
Data Sources: posts, users
Report Format: JSON
Frequency: On-demand, can be generated by users

---

9. GUI and API Manuals

Refer to the API Documentation