The University of Faisalabad

SRS DOCUMENT

Prepared for Mam Irsha Qureshi

Prepared By:

Waleed 054

Hanzla 046

Taha 090

Ali 038

Software Requirements Specification (SRS)
Social Media Feed Simulation

1.Introduction

1.1 Purpose

The purpose of this SRS is to define the requirements and functionalities for a **C++-based Social Feed Application**. The application includes:

- A **server** that manages user accounts, posts, comments, friend relationships, polls, and direct messages (DM).
- A **Win32 GUI client** (or any socket-based client) that connects to the server, enabling users to interact with the system (create posts, view feed, DM friends, etc.).

1.2 Scope

The project scope covers:

- **User authentication** (register and login).
- **Feed management** (create/read posts, comments, likes/dislikes).
- **Direct messaging** between friends.
- **Friend system** (send/accept friend requests).
- **Poll creation** and voting.
- **GUI client** that strips ANSI color codes (sent by the server) and displays only plain text.
- **Server** (in C++) that stores and processes user data, feed/posts, and polls.

This SRS does **not** include advanced security measures (like encrypted transport), advanced data persistence (like a full database), or highly-scalable architecture. It is intended as a **basic** social platform.

1.3 Definitions, Acronyms, and Abbreviations

- **GUI**: Graphical User Interface
- **SRS**: Software Requirements Specification
- **DM**: Direct Message
- **ANSI Codes**: Special terminal escape sequences for color output, removed on the client side

1.4 References

- Win32 API documentation: Microsoft Docs
- Socket programming in C++: Beej's Guide
- ANSI color codes: Wikipedia: ANSI Escape Code

1.5 Overview

The remainder of this SRS describes in detail:

1. Overall product description.
2. System features (server-side and client-side).
3. Use cases and data flow.
4. Non-functional requirements (performance, reliability, etc.).

2.Overall Description

2.1 Product Perspective

This project is a **client-server** system:

- **Server**: A socket-based C++ application that manages all data (users, posts, comments, polls, friends).
- **Client**: A Win32 GUI in C++ that connects to the server via TCP sockets. Previously, a console client existed, but the scope now includes a **GUI** client.

2.2 Product Features (High-level)

1. **User Account Management**:
    - Registration (unique username, password).
    - Login (username, password).
2. **Feed**:
    - Create posts (/post <content>::<mediaURL>).
    - View feed (/feed command shows all posts).
    - Comment on posts, like/dislike posts.
3. **Friend System**:
    - Send friend requests (/friend request <username>).
    - Accept friend requests (/friend accept <username>).
    - Only friends can DM each other.

4. **Direct Messaging**:
    - /dm <friendUsername> <message>.
5. **Polls**:
    - Create polls with multiple options.
    - Vote on polls; view poll results.
6. **ANSI Code Removal** (Client-Side):
    - The server may send color codes for console output. The GUI strips these for plain-text display.

2.3 User Classes and Characteristics

- **Regular User**: Can register, log in, create/view posts, DM friends, comment, like/dislike, create/join polls.
- **Admin** (optional extension): Could have privileges like kicking users, but that might not be fully implemented in the scope.

2.4 Operating Environment

- **Server**:
    - Runs on Windows (compiled with MinGW or MSVC) or Linux (with minimal changes) as a console-based C++ app.
    - Requires open port 54000 (default) for incoming TCP connections.
- **Client**:
    - Win32 GUI (C++).
    - Built with MinGW or MSVC.
    - Requires network access to the server's IP/port.

2.5 Design and Implementation Constraints

- **Language**: C++17 or higher.
- **Socket-based**: using WinSock2 on Windows.
- **No advanced database**: basic file-based persistence or in-memory data structures.
- **No built-in encryption**: data sent in plaintext over sockets.
- **GUI**: Standard Win32 Edit controls for feed, no color rendering (ANSI codes are stripped).

2.6 User Documentation

A simple README or help screen describing:

- How to **register** or **login**.
- Command usage (e.g., /feed, /post, /dm, etc.).
- Basic troubleshooting (e.g., firewall issues, server not running).

2.7 Assumptions and Dependencies

- The user will have basic network connectivity.
- The server is running before the client connects.
- The server's IP/port is correct in the client code (127.0.0.1:54000).
- Minimal concurrency constraints; the server handles multiple clients but not extremely large scale.

3.System Features

3.1 Registration G Login

Description: Allows a new user to create an account or an existing user to authenticate.

- **Inputs**: Username, password, plus an option R (register) or L (login).
- **Processing**: The server checks if username exists or not, verifies password, and responds.
- **Outputs**: "Success" (login/register) or an error message ("Invalid username or password", etc.).

3.2Feed Management

3.2.1 Create Post

Description: A user can create a post with optional media URL.

- **Command**: /post <text>::<mediaURL>.
- **Server**: Stores post with an auto-increment post ID, username, timestamp.
- **Output**: "Post created successfully! (ID: X)".

3.2.2 View Feed

Description: Retrieves all posts in reverse chronological order.

- **Command**: /feed.

- **Server**: Sends plain-text listing of each post: ID, timestamp, username, content, media URL, like/dislike counts.
- **Client**: Strips ANSI if present, displays text in feed box.

3.2.3 Comments, Likes, Dislikes

Description: Comments on posts (/comment <postID> <text>), or like/dislike (/like <postID>).

- **Server**: Associates comment with a post. For likes/dislikes, adds user's name to respective sets.
- **Output**: Confirmation or error.

3.3 Friend System

Description: Users can send/accept friend requests. Only friends can DM each other.

- **Commands**:
  - /friend request <username>
  - /friend accept <username>
- **Server**: Tracks friend relationships in memory or a file.
- **Output**: Notification of success or "User not found" / "No friend request pending."

3.4 Direct Messaging (DM)

Description: Send private messages to a friend.

- **Command**: /dm <friendUsername> <message>.
- **Server**: Forwards the message to friendUsername's socket if online; otherwise, an error or ignore.
- **Output**: "DM sent to X" or "User 'X' is not online" or "You are not friends with X."

3.5 Polls

Description: Create or vote in a poll.

- **Command**:
  - /poll create <question>::opt1,opt2,opt3...
  - /poll vote <pollID> <optionIndex>
  - /poll results <pollID>

- **Server**: Maintains poll question, options, and votes in memory.
- **Output**: Confirmation and real-time vote counts.

3.6 Edit/Delete (Time-Limited)

Description: Creator can edit or delete their own posts/comments within a certain time (e.g., 120s).

- **Commands**:
  - /editpost <postID> <newContent>
  - /deletepost <postID>
  - /editcomment <cmtID> <newText>
  - /deletecomment <cmtID>
- **Server**: Checks ownership and timestamp before allowing.
- **Output**: "Post edited successfully" or "Time limit expired."

4.External Interface Requirements

4.1 User Interface

- **Client**: Win32 GUI with text fields for:
  - Username/password
  - Radio buttons (Register/Login)
  - Connect button
  - Large read-only feed (Edit control)
  - Command input line + "Send" button
- **Server**: Console-based, logs status messages (like "User X connected," "New post ID=12," etc.).

4.2 Hardware Interfaces

- Minimal: any modern Windows PC for the client, plus a PC or local host for the server.
- Network: local loopback (127.0.0.1) or LAN. Requires port 54000 open.

4.3 Software Interfaces

- **WinSock2** for networking on Windows.

- **File I/O** or in-memory vectors for storing posts, users, polls.

5.System Requirements

5.1 Functional

- **FR1**: The system shall allow new user registration.
- **FR2**: The system shall authenticate existing users by username/password.
- **FR3**: The system shall store and display a feed of user posts.
- **FR4**: The system shall allow creation of polls and voting.
- **FR5**: The system shall allow direct messaging only between friends.
- **FR6**: The system shall allow friend requests and acceptance.
- **FR7**: The client shall remove ANSI color codes before displaying text.

5.2 Non-functional

- **NFR1**: **Performance**: The server should handle multiple concurrent clients (at least a few) without crashing.
- **NFR2**: **Reliability**: Basic error-handling for invalid commands or invalid user input.
- **NFR3**: **Usability**: The GUI client must be straightforward: user credentials, “Connect,” then a command field.
- **NFR4**: **Maintainability**: Code structured in separate modules (server vs. client).
- **NFR5**: **Portability**: The server code should compile on Windows or potentially other OS (with minimal changes for sockets), while the client is Windows-specific (Win32).

6.Use Cases

6.1 UC1: Register

- **Actor**: New user.
- **Precondition**: Server is running.
- **Flow**:

1. User enters “R .”
2. Server checks if user exists. If not, creates user.
3. Server returns “Success.”

- **Postcondition**: New account is stored.

6.2 UC2: Login

- **Actor**: Existing user.
- **Flow**:
  1. User enters “L .”
  2. Server verifies credentials.
  3. Server returns “Success” or error.
- **Postcondition**: User is authenticated.

6.3 UC3: Post to Feed

- **Actor**: Logged-in user.
- **Flow**:
  1. User types /post Hello::http://image.com/img.png.
  2. Server creates a new post (ID, user, content, optional media).
  3. Returns “Post created (ID=X).”

6.4 UC4: View Feed

- **Actor**: Logged-in user.
- **Flow**:
  1. User types /feed.
  2. Server sends a list of recent posts.
  3. Client displays them in the feed box.

6.5 UC5: Friend Request

- **Actor**: Logged-in user.
- **Flow**:
  1. User types /friend request Bob.
  2. Server logs that “Alice” requested Bob.

3. Bob can type /friend accept Alice to confirm.

4. **UC6: DM**
    - **Actor**: Logged-in user, must be friends.
    - **Flow**:
        1. User types /dm Bob Hey buddy!
        2. Server checks if Bob is a friend. If yes, sends “Hey buddy!” to Bob’s client.

7.Non-Functional Requirements

1. Performance

- The server must handle at least 5 simultaneous clients without major slowdown.

2. Reliability

- The server logs errors if a command fails.
- The client does not crash if the server disconnects.

3. Usability

- The GUI should be minimal: username, password, a connect button, a feed area, and a command line.

4. Maintainability

- C++ code should be separated: “server.cpp” for server logic, “client.cpp” for the Win32 GUI.

5. Security (Limited)

- Passwords stored in plaintext or lightly hashed (not fully secure).
- No encryption on the socket (plaintext).

8.Other Requirements

- **ANSI Code Removal**: The GUI must remove ESC [... m sequences so the feed is plain text.
- **Time-limited Edit/Delete**: Optionally, the server enforces a small window (like 120 seconds) to edit or delete a post/comment.

- **File-based** or **In-memory** data store: Basic .txt files for storing user credentials and posts.

Conclusion

This SRS outlines the **functional** and **non-functional** requirements for the **C++ Social Feed App** (server + Win32 GUI client). With user authentication, feed posting, friend system, DM, polls, and basic GUI features, it aims to be a **mini social platform** for learning purposes.

This **SRS** ensures a clear baseline for future iterations and testing.

End of Document
