Acknowledgement

We would like to express our deepest gratitude and sincere appreciation to all individuals who contributed directly or indirectly to the successful completion of our final year project, the "Local Area Network (LAN) Based Chat and File Sharing System."

First and foremost, we would like to extend our sincere thanks to our project supervisor, **Tr. Maran Gam Awng** and co-supervisor **Tr. Labya Naw Naw**, for their invaluable guidance, constant encouragement, and insightful feedback throughout this project. Their expertise and patience were instrumental in helping us navigate challenges and refine our work. Without their unwavering support, this project would not have reached its full potential.

We are also immensely grateful to all the teachers and staff of Mai Ja Yang College for providing us with the necessary resources and a conducive learning environment. We would particularly like to acknowledge our Principal, **Dr. Hpala Zau Jat**, and Academic Officer, **Tr Htingnai Ja Bu**, for their administrative support. Our special thanks also go to Head of computer science, **Tr. Shadau Seng Tawng** and all our Computer Science major teachers, compulsory subject teachers, and faculty members from various majors who have imparted valuable knowledge to us. The knowledge and skills imparted during our coursework formed the essential foundation upon which this project was built.

Our sincere thanks go to our friends and classmates for their moral support, stimulating discussions, and for willingly testing our application. Their constructive criticism and valuable suggestions were crucial in improving the system's functionality and user experience.

Lastly, and most importantly, we wish to express our heartfelt gratitude to our families. Their endless love, unwavering belief in our abilities, and immense patience during this demanding period were our greatest source of strength and motivation. This achievement is as much theirs as it is ours.

Abstract

This document presents the design, implementation, and functionality of a "Local Area Network (LAN) Based Chat and File Sharing System." This system addresses this need by leveraging the existing LAN infrastructure to provide real-time messaging and seamless file sharing capabilities among connected users.

The system is developed as a desktop application featuring a centralized client-server architecture. The server application manages user connections, message routing, and file transfer operations, while the client application provides an intuitive graphical user interface (GUI) for users to interact with the system. Core functionalities include multi-user real-time text chatting, private messaging, and drag-and-drop file sharing. The system is designed to be lightweight, efficient, and independent of an external internet connection, ensuring high-speed data transfer and low latency within the network.

Key technical implementations include the use of socket programming for network communications, multi-threading to handle multiple clients concurrently without performance degradation, and a straightforward user authentication mechanism to maintain basic access control. The file sharing module is optimized to handle various file formats and sizes, transferring files directly between users via the server to maintain network integrity.

This project successfully demonstrates the practical application of computer networking principles to create a cost-effective, secure, and efficient collaborative tool

Contents

List of Figures

List of Table

CHAPTER 1
Introduction

Effective and secure communication is the backbone of collaboration in modern environments such as educational institutions, offices, and organizations. While internet-based messaging platforms offer extensive connectivity, they often depend on external infrastructure, pose potential data privacy concerns, and are unusable in scenarios with limited or no internet access. This gap highlights a critical need for a self-reliant, efficient, and secure communication system that operates independently of the global internet.

This project presents the design and implementation of a "Local Area Network (LAN) Based Chat and File Sharing System". The system is engineered to provide a robust, standalone platform for real-time messaging and seamless file transfer within a confined network environment. By leveraging the high-speed, low-latency characteristics of a LAN, the system ensures fast and reliable communication without the need for an active internet connection. This makes it an ideal solution for settings like college campuses, corporate offices, or any localized group where internal communication and resource sharing are paramount.

1.1 Project Overview

Local Area Network (LAN) Based Chat and File Sharing System is designed to enable real-time communication and file sharing among computers connected within a LAN. The system allows data exchange between users without requiring an internet connection, making it efficient and secure for local communication.

The system is to facilitate easy and fast communication and file transfer among team members, departments, office and within a classroom environment. The system enables instant messaging between connected users within the LAN and sending and receiving files (documents, images, etc.) between users or group. Secure login with Username and Password to ensure data privacy. Provides sound or visual alerts for incoming messages and displays online/offline status of user.

This system uses Python language in the backend and HTML/CSS, JavaScript, Json in the frontend. TCP/IP socket programming is built on the Linux Platform.

Develop a chat system that work efficiently over a LAN without the need for internet access and to make file sharing among team members easy and secure. To design a user-friendly interface and reliable communication platform. Lacal Area Network (LAN) Based Chat and File Sharing System provide a standalone LAN-based system capable of real-time messaging and high-speed file transfer among connected computers.

1.2 Project Objectives

This project has several main goals. Its first objective is to learn how to build a personal server within a local area networking LAN (like Proxmox server). In addition, to provide fast and efficient real-time communication for users on the same local network (LAN). Example It aims to make sharing files between these users a simple and quick process and to facilitate the submission of activities, such as school assignments.

1.3 Project Scope and Limitations

1.3.1 Project Scope

"Project Scope" refers to what is included and what the system is capable of doing.

1. User Management

Registration: Users can register an account using a Username and Password.

User List: Ability to view a list of all users currently on the network.

User Status: Ability to see the status of users (e.g., Online, Offline).

2. Real-time Messaging

Private Chat: Users can send private messages to each other.

Group / Broadcast Chat: Users can connect in groups and send messages to all users simultaneously.

Chat History: Users can view their message history.

3. File Sharing

Users can send files, photos, and videos to each other. Supports direct file transfer. File size limits are defined.

4. Administration & Network

Local Network Operation: The system works only within the same local area network (via Wi-Fi or Ethernet).

Centralized Control (Client-Server Model): A central server manages all information and operations.

1.3.2 Project Limitations

"Limitation" refers to the weaknesses of this system or the things it cannot do.

Performance

Limited Users: Depending on the server's performance capacity, the system may become slow if the number of users exceeds 100-500. It may not be possible to support over 600 users. Currently, the video call feature is not yet available.

Large File Issues: There may be restrictions in place that prevent sending very large files, such as those several GB in size or videos.

Memory Usage: Storing users' files and chat history requires significant use of the server's memory (RAM).

1.4 Problem Statement

In today's digital age, educational institutions and organizations rely heavily on effective communication tools to facilitate collaboration, information sharing, and administrative operations. While internet-based messaging platforms like WhatsApp, Telegram, and Facebook Messenger offer widespread connectivity, they present significant limitations in specific contexts, particularly in environments with poor or no internet infrastructure.

Many regions, including our college in Mai Ja Yang, experience unreliable internet connectivity, making cloud-dependent communication platforms impractical for daily use. Furthermore, even when internet access is available, these third-party services raise serious concerns about data privacy, security, and institutional control. Sensitive academic discussions, student data, and internal communications are stored on external servers, potentially exposing them to unauthorized access.

Existing LAN-based chat solutions, while addressing the connectivity issue, often suffer from outdated interfaces, limited features, poor scalability, or complex configuration requirements. Many lack essential functionalities needed in educational settings, such as user management for administrators, file sharing for assignments, and group communication for classrooms.

This project, the "Local Area Network (LAN) Based Chat and File Sharing System (SpeedChat)", addresses these challenges by developing a modern, secure, and efficient communication platform that operates entirely within a local network. It aims to provide a reliable alternative that ensures data sovereignty, reduces operational costs, and offers customization tailored specifically to the needs of educational environments without depending on external internet services.

CHAPTER 2
Background Study and Methodology

2.1 System Requirements

Hardware

- Proxmox Server: Run multiple VM containers with sufficient CPU, Ram and Storage.
- Local network and router: Support LAN distribution with static IP configuration for each service.
- Ethernet cable: support for network or Wifi, connect to router.

Software

- Proxmox VE: Updated to the latest version.
- Linux VM Templates: Use Debian for each container (Flask, File Browser)

2.1.1 What is Local Area Network (LAN)?

A Local Area Network (LAN) is a type of network created to connect computers, printers, phones, and other computer devices located close to each other in a single location (for example, one home, one office, one school) so that they can communicate and be used together.

For example

· If you have Wi-Fi at home, all the phones, computers, and Smart TVs connected to that Wi-Fi are within your home's LAN.

· A computer lab at school with about 50 connected computers, or classrooms with Wi-Fi, are also LANs.

2.1.2 Where are LANs used?

You can find LANs almost everywhere.

1. Homes (Home Networks) - For family members to use the internet, share files, and play games together.

2. Schools - In computer labs, libraries, and teachers' offices.

3. Offices - For employees to communicate with each other, share data, and use a central printer.

4. Hospitals - For patient lookup systems and sharing medical records.

5. Public places like coffee shops - For customers to use the internet.

2.1.3 How does it benefit students?

For a student, a LAN greatly supports your education and daily life.

1. Research & Learning - Using the school's Wi-Fi, you can easily access the internet to search for information and access online libraries and e-books.

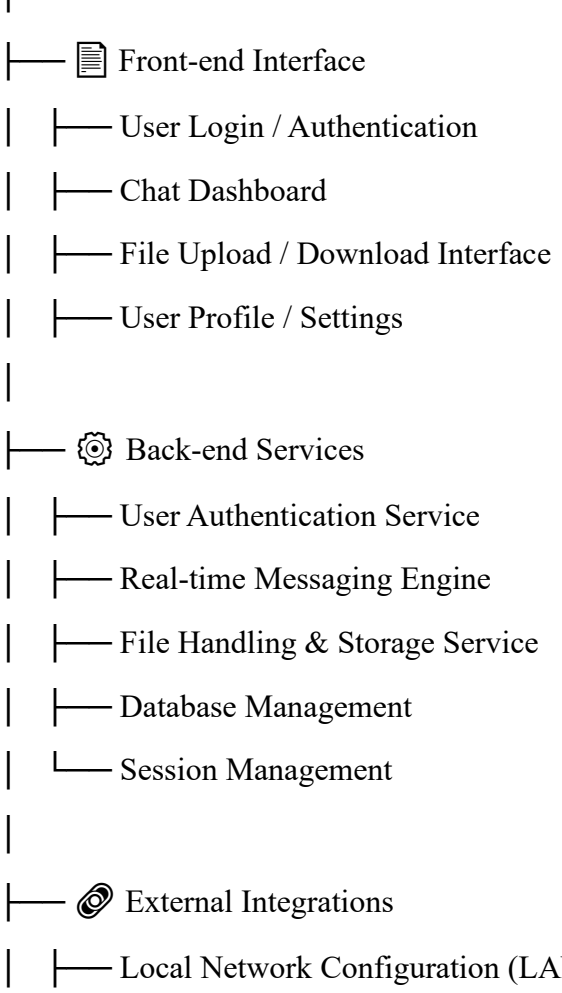
2. Collaboration - Using online platforms like Google Docs or Microsoft Teams via the LAN, you can work on group essays, prepare presentations, and easily send files to other students.

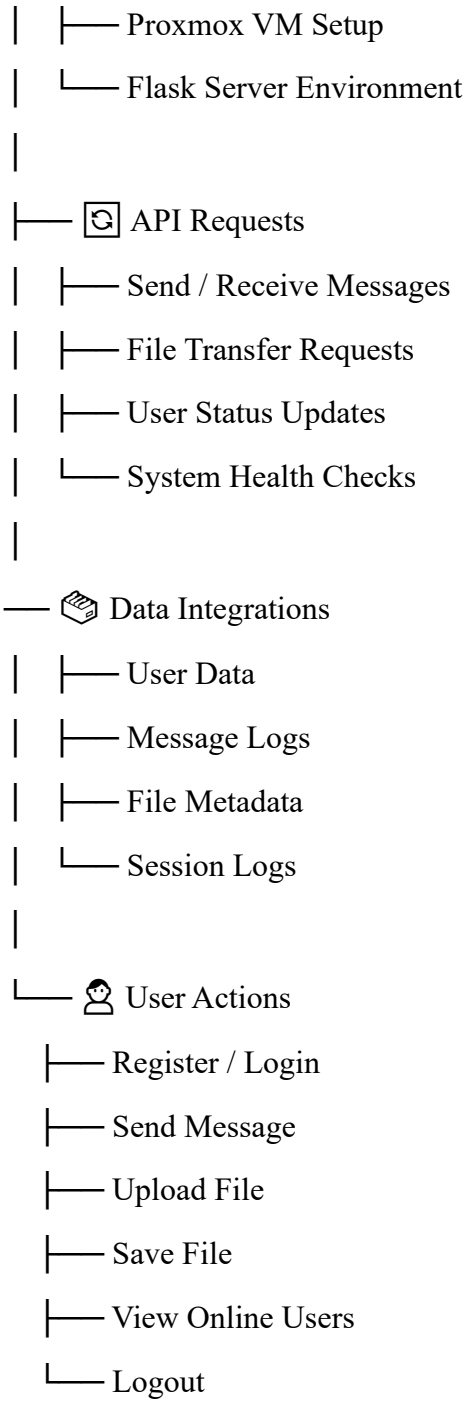
3. Resource Efficiency - To install software in a school computer lab, it doesn't need to be installed on every single computer. It can be distributed to all from one central server.

4. Knowledge Expansion - Understanding LANs provides a good foundation for students interested in technology fields like networking and IT management.

5. Easy Management - Teachers can centrally manage students' computers through the LAN to deliver lessons and assign exercises.

2.3 Site Map





2.3.1 Front-end Interface Layer

The front-end serves as the primary point of interaction for the end-user. It is a dynamic web application that renders the user interface and communicates with the back-end services via a defined API. Its key modules include:

- **User Login / Authentication Interface:** A secure portal that collects user credentials and manages the login/logout life cycle.

- **Chat Dashboard:** The central hub for real-time communication, displaying active conversations, contact lists, and message histories.
- **File Upload / Download Interface:** Provides a drag-and-drop or browser-based mechanism for users to transfer files to and from the system.
- **User Profile / Settings Module:** Allows users to view and modify their personal information and application preferences.

2.3.2 Back-end Services Layer

This layer constitutes the core application logic, built upon the Django web framework. It is composed of several discrete services:

- **User Authentication Service:** Validates user credentials, manages password hashing, and issues session tokens to maintain user state.
- **Real-time Messaging Engine:** Leverages Web Sockets to facilitate instantaneous, bi-directional communication for message delivery and user status updates.
- **File Handling & Storage Service:** Manages the secure reception, storage, retrieval, and access control of user-uploaded files on the server's file-system.
- **Database Management Service:** Handles all Create, Read, Update, and Delete (CRUD) operations for persistent data, abstracting direct database interactions.
- **Session Management Service:** Tracks active user sessions, manages timeouts, and ensures secure access to protected resources.

2.3.3 External Integrations and Deployment Environment

The system's operational context is defined by its integration with specific external platforms and network configurations:

- **Local Network Configuration (LAN):** The system is designed to operate within a LAN, leveraging its high-bandwidth, low-latency characteristics for optimal performance and enhanced security through network isolation.
- **Proxmox VM Setup:** The server component is containerized within a Proxmox Linux (VM), providing a lightweight, isolated, and easily manageable runtime environment.

2.3.4 API Communication Layer

The interaction between the front-end and back-end is standardized through a Restful API and Web Socket connections. The primary API endpoints and actions include:

- **Message Endpoints:** For sending and receiving text messages in real-time.

- **File Transfer Endpoints:** For initiating uploads and downloads, handling multipart/form-data requests.
- **User Status Endpoints:** For updating and fetching the online/offline status of users.
- **System Health Check Endpoint:** A monitoring endpoint used to verify the operational status of the server and its services.

2.3.5 Data Integrations and Persistence

The system interacts with a relational database to persistently store all operational data. The key data entities are:

- **User Data:** Stores user profiles, hashed passwords, and preferences.
- **Message Logs:** Archives all sent and received messages with timestamps and sender/receiver identifiers for history and auditing.
- **File Metadata:** Contains information about stored files, such as filename, size, uploader, upload timestamp, and physical storage path.
- **Session Logs:** Records user login and logout times, IP addresses, and session tokens for security and analytics.

2.3.6 User Interaction Flow

The system is designed around a set of core user actions that define the primary use cases. These actions trigger the aforementioned components in a coordinated sequence:

1. **Register / Login:** A user provides credentials, which are verified by the Authentication Service, leading to session creation.
2. **Send Message:** A message composed in the Chat Dashboard is sent via the API to the Real-time Messaging Engine, which broadcasts it to the intended recipient(s).
3. **Upload File:** A file selected through the interface is transmitted to the File Handling Service, which stores it and records its metadata in the database.
4. **Save File:** A user request to download a file is routed through the File Handling Service, which retrieves the file from storage and initiates the download.

2.4 Compare

Feature	SpeedChat	Traditional Lan messenger	Internet app
Internet	Not Required	Not Required	Required
Data Security	Full Security Data stays within LAN	Weak Poor security	Risky Stored on external servers
Cost	Free	Free	Paid Premium features required
For Schools	Special Design Assignment submission, groups	General use	Limited
Customization	Freely Customizable	Cannot customize	Cannot customize

2.5 Methodology

2.5.1 Proxmox VM, Flask and Networking Setup

Proxmox server is an open-source virtualization software based on Linux (Debian) that lets you run and manage virtual machines (VMs), containers, and storage all on one physical server using a simple web interface. It is widely used for creating private cloud infrastructure at home or in enterprise settings. Building a Robust and Isolated Server Environment.

Main Features:

- Proxmox VE enables you to create and manage multiple virtual machines, allowing you to install any operating system you want (Linux, Windows, etc.).
- It supports lightweight LXC containers for efficient resource usage and fast performance.
- The user-friendly web interface helps you manage VMs, networks, storage, and clusters easily.
- Proxmox VE supports features like clustering and high-availability setups, making it useful for both labs and enterprise environments.

2.5.1.1 Proxmox Virtual Environment (VE) Setup

Proxmox VE was chosen as the primary platform for virtualization. It was installed on a dedicated server to host all the project's virtual machines (VMs) and Linux Containers (LXC). Proxmox provides a powerful web-based dashboard for managing virtualized resources, which simplified tasks such as creating, snapshotting, and managing our containers.

- **Initial Configuration:** The Proxmox host server was configured with a static IP address on the college LAN to ensure consistent accessibility for all team members. Storage was set up and named appropriately (e.g., local-lvm).
- **Cluster (Optional):** A single-node cluster was established, as the project did not require high availability.

2.5.2 Python

Python is a programming language that is widely used in web applications, software development, data science, and machine learning (ML). Developers use Python because it is efficient and easy to learn and can run on many different platforms.

Why it's used:

· Python is easy to write and has rich libraries for Socket Programming, Web Frameworks (Django/Flask), and Data Handling, making it ideal for this project's needs.

· All Backend Services like Real-time Messaging, File Sharing, and User Authentication are written in Python.

Interconnection with other parts:

· Back-end Services Layer: It is the lifeblood of all services like the User Authentication Service, Real-time Messaging Engine, and File Handling Service.

· Abstract: The point "The system is developed using socket API" is implemented using Python Socket Programming.

2.5.3 JavaScript

JavaScript is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else. i.e. anything that moves, refreshes, or otherwise changes on your screen without requiring you to manually reload a web page. Features like: animated graphics. photo slideshows.

Why it's used:

· It is essential for achieving the goal of a User-friendly Interface.

· It allows the Chat Dashboard to display new messages instantly without reloading the page (Real-time Update).

· It is used to call Web Sockets or API Calls to easily send user actions (e.g., Send Message, Upload File) to the Backend Server.

Interconnection with other parts:

· Front-end Interface Layer: It is the technology that brings the Chat Dashboard and File Upload/Download Interface to life.

· API Communication Layer: JavaScript initiates API Requests like "Send / Receive Messages" and "File Transfer Requests."

2.5.4 HTML/CSS

HTML is a markup language used to create static web pages and web applications. CSS is a style sheet language responsible for the presentation of documents written in a markup language. Consists of tags surrounding content. For Example: *Welcome to Simplilearn* .

Why it's used:

· Creating the basic Structure (HTML) and Visual Design (CSS) of the Website.

· They are the foundational building blocks for creating a web-based User Interface.

· HTML is used to structure components like the User Management Interface, Chat Box, and File Upload Button.

· CSS is used to design the interface to be visually appealing and easy to use (User-friendly).

Interconnection with other parts:

· Front-end Interface Layer: The visual part of all components, including the User Login Interface, Chat Dashboard, and File Upload Interface, is built with HTML/CSS.

2.5.5 JSON

JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax. It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).

Why it's used:

· Used as a standardized Format for exchanging Data between the Frontend and Backend.

· It is a lightweight, easy-to-read, and fast-to-process format, perfect for data exchange between JavaScript (Frontend) and Python (Backend).

· For example, when a user sends a message, it's sent to the backend in a JSON format like {"sender": "user1", "message": "Hello", "timestamp": "..."}. The backend also returns data, like a message list, in JSON.

Interconnection with other parts:

· API Communication Layer: It implies that all API Endpoints exchange Request and Response data using the JSON format.

· Data Integrations: JSON acts as the data carrier, transporting data from the database to the frontend via the API.

CHAPTER 3

System Architecture & Graphical User InterfaceS

3.1 Use Case Diagram

User Sign up
This diagram visually shows the steps a user needs to follow to create a new account. It is a kind of "use case diagram" for a registration feature.

Step-by-Step Explanation

- *Upload Profile Photo*: The user uploads a profile picture.
- *Put User Name*: The user enters their username.
- *Put Email*: The user provides their email address.
- *Put New Password*: The user chooses and enters a new password.
- *Confirm New Password*: The user re-enters the chosen password to confirm.
- *Click Create Account*: After filling in all required fields, the user clicks to create the account.
- *Click Back to Login*: The user can also choose to return to the login page instead.

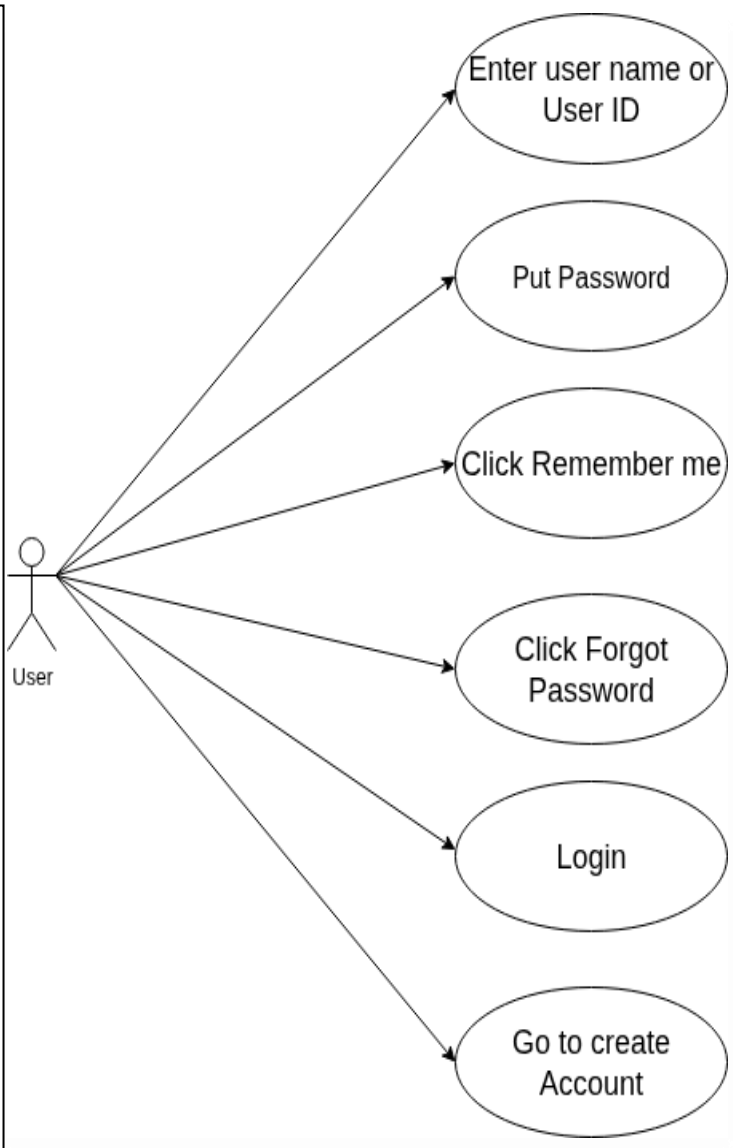


Figure 3.1.1 User Login Form diagram

User Login

This diagram shows the login process for a user, similar to the previous example but focused on logging in instead of account creation.

Step-by-Step Explanation

- *Enter user name OR Admin username*: The user types their unique name or Admin username to identify themselves.
- *Put Password*: The users and Admin enters their account password.
- *Click Remember me*: The user can click to have the system remember their login for convenience next time.
- *Click Forgot Password*: If the user forgot their password, they can start a recovery process by clicking here.
- *Login*: After filling in the required fields, the user clicks to log into their account.
- *Go to create Account*: If the user doesn't have an account, they can go to the account creation page.

NOTE: Admin can log in from this login page. Since there is only one admin who controls the entire app, the login feature here is specifically designed for the admin to access and manage everything.

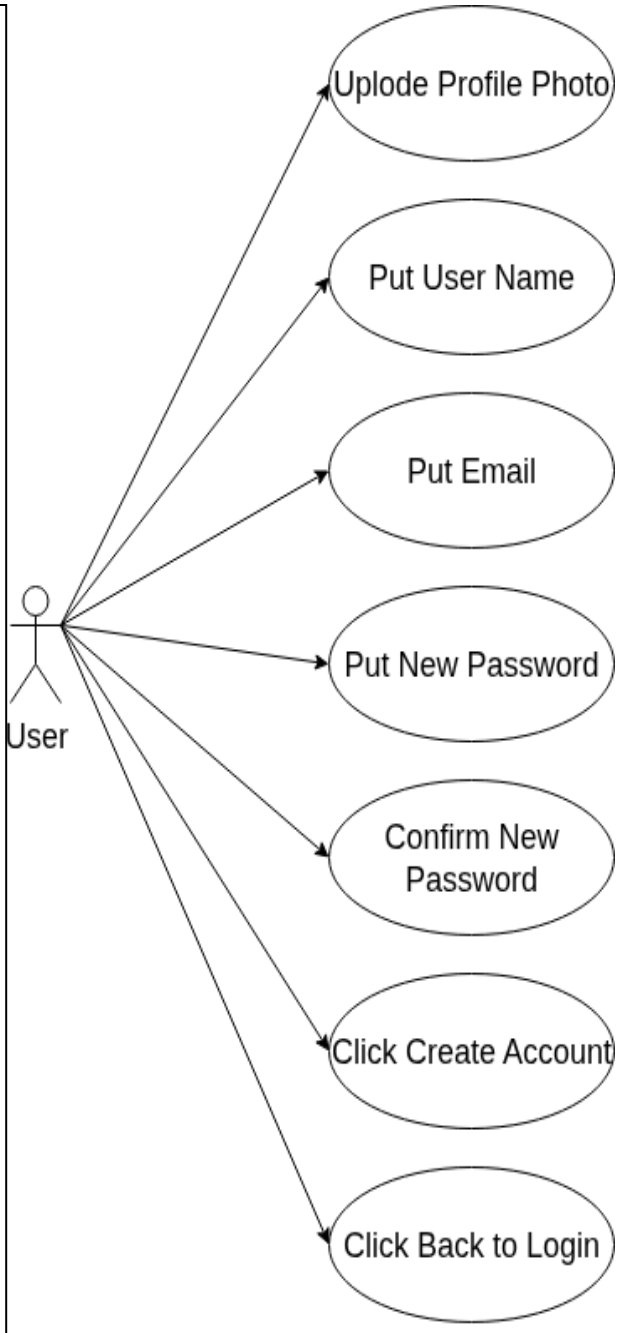


Figure 3.1.2 User Signup Form diagram

Forgot password for reset password
This diagram shows the actions involved in resetting a password. It is a "use case diagram" for the password reset processes.

Step-by-Step Explanation

- *User ID*: The user provides their user ID to identify which account needs the password reset.
- *New Password*: The user chooses and enters a new password for their account.
- *Re-enter new Password*: To confirm, the user types the same new password again to avoid mistakes.
- *Reset Password*: After both passwords match, the user clicks to reset the password.
- *Click Back to Login*: The user can also choose to return to the login screen at any time.

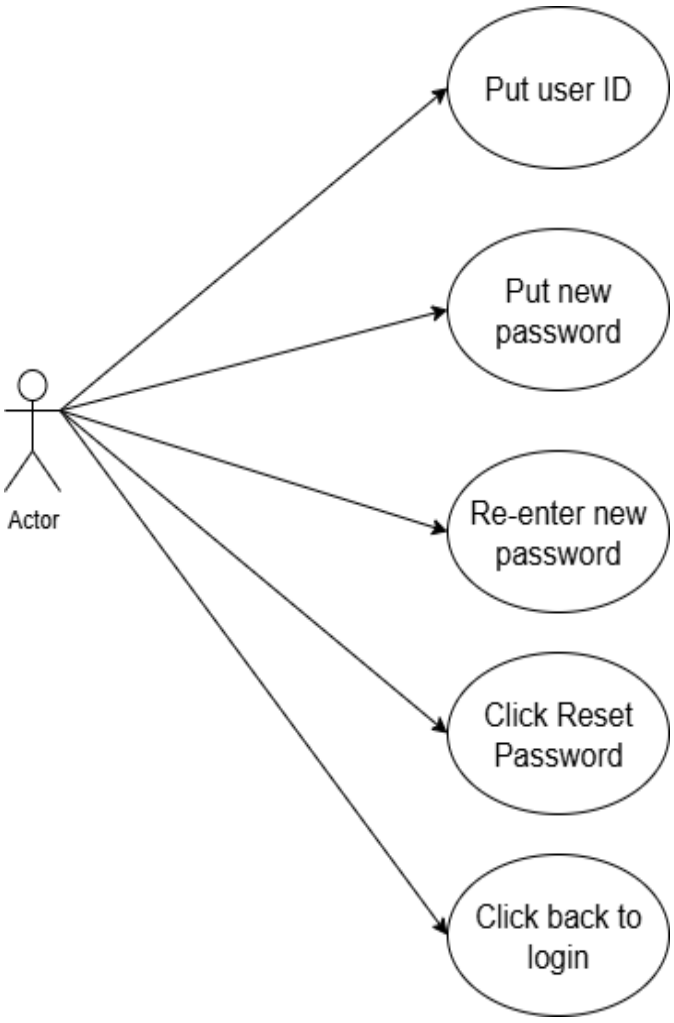


Figure 3.1.3 Forgot Password Form

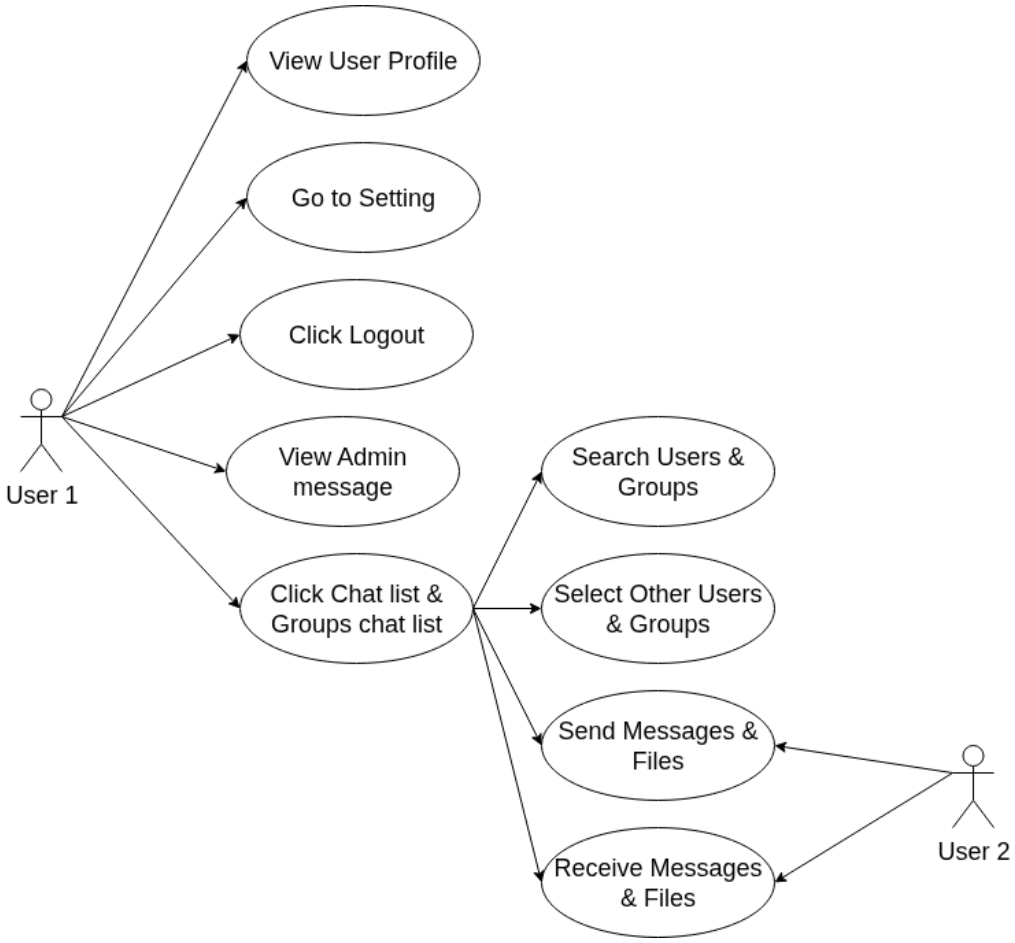


Figure 3.1.4 Home Page Diagram

Home Chat

This diagram shows the actions and workflows a user can perform within a chat application. Each oval is a feature, and arrows represent possible interactions between users and those features [1].

Step-by-Step Explanation

- *View User Profile*: User 1 can view their own profile for information or edits.
- *Go to Setting*: User 1 can access settings to customize the app or their account.
- *Click Logout*: User 1 can log out of the app to end their session.
- *Click Chat list & Groups chat list*: User 1 can access a list of chats and group conversations.
- *Search Users & Groups*: User 1 can look for other users or groups to chat with.

- *Select Other Users & Groups*: After searching, User 1 selects who they want to interact with.
- *Send Messages & Files*: Both User 1 and User 2 can send messages or files via the chat interface.
- *Receive Messages & Files*: Both User 1 and User 2 can receive messages or files from others.

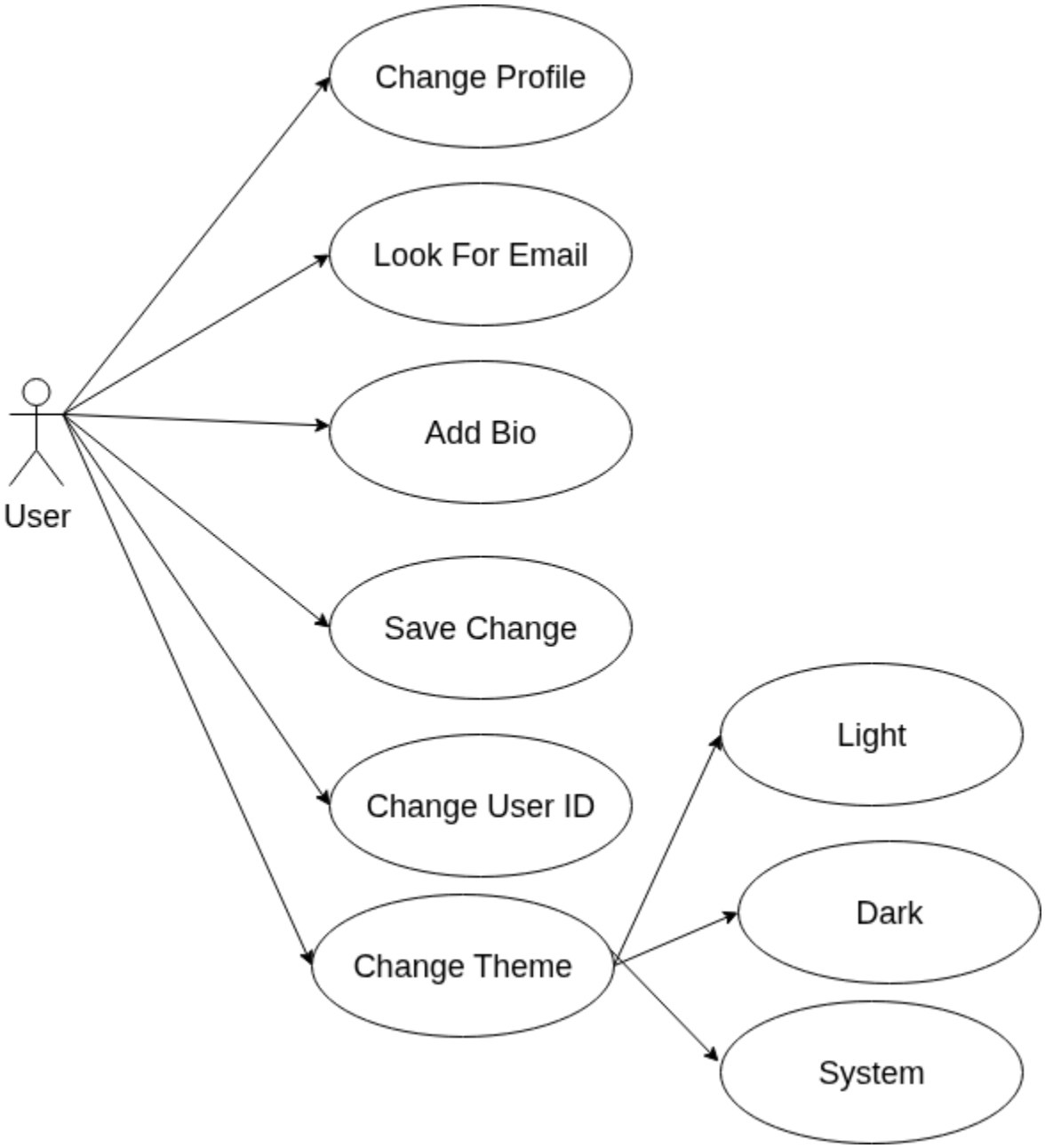


Figure 3.1.5 Setting Diagram1

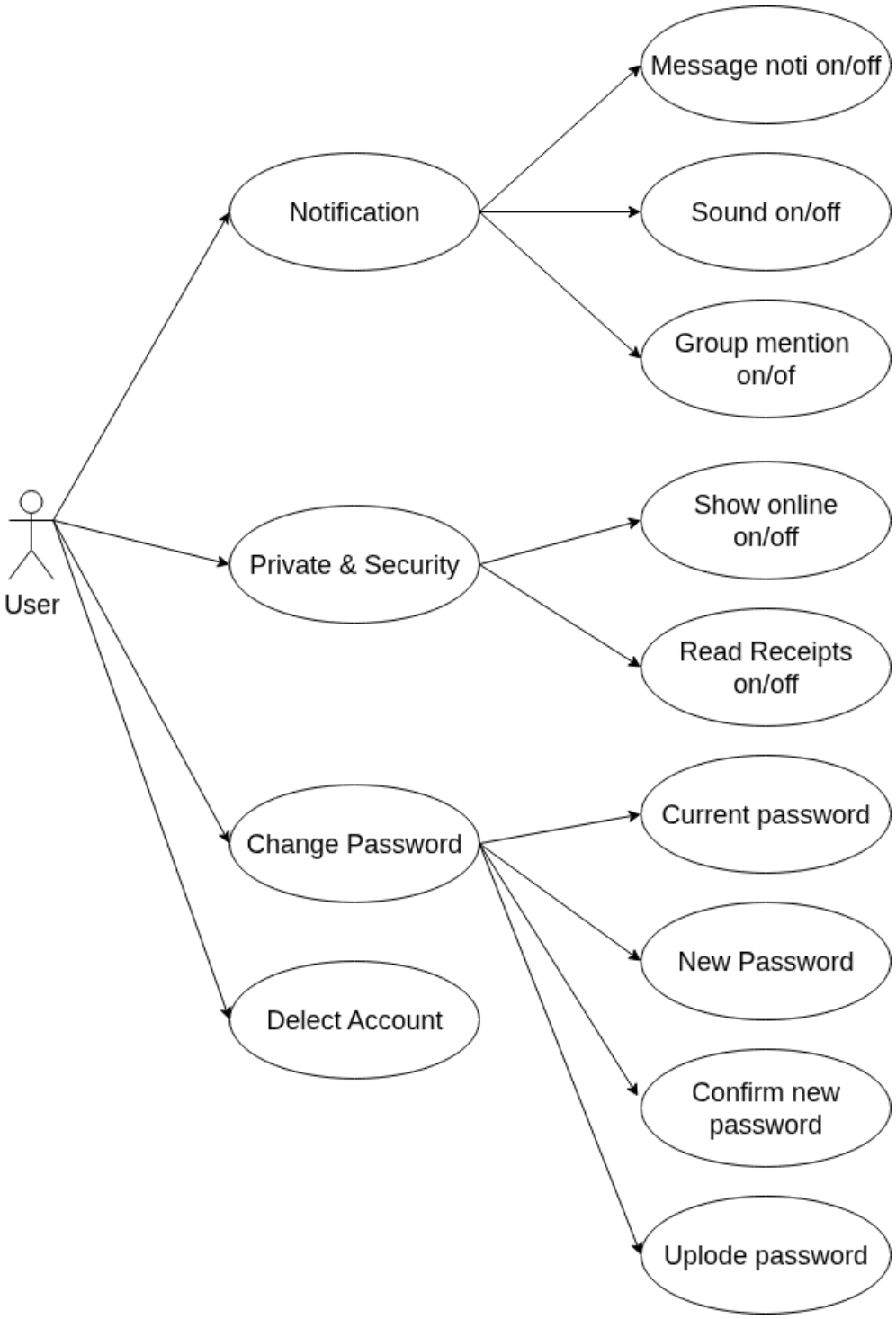


Figure 3.1.6 Setting Diagram2

Setting 1
This diagram shows the user profile and theme settings features available to a user in the app. Arrows point from the user to each function that can be performed [1].

Step-by-Step Explanation

- *Change Profile*: The user updates their profile details such as name or avatar.
- *Look for Email*: The user can view or check their email address.
- *Add Bio*: The user adds a bio or personal introduction to their profile.
- *Save Change*: After making updates, the user saves their changes to apply them.
- *Change Theme*: The user can customize the app's appearance by changing the theme.
    - Light: User selects a light color scheme.
    - Dark: User selects a dark color scheme.
    - System: User chooses to match the app theme to the system default.

Setting 2
This diagram outlines features related to notifications, privacy, and account safety for the user.

Step-by-Step Explanation

- *Notification*: User can adjust:
    - **Message notification on/off**: Enable or disable message notifications.
    - **Sound on/off**: Enable or disable notification sounds.
    - **Group mention on/off**: Get notified for group mentions.
- *Private & Security*:
    - **Show online on/off**: Display/hide online status.
    - **Read Receipts on/off**: Show/hide message read receipts.
- *Change Password**:***
    - **Current password**: Enter current password.
    - **New Password**: Enter a new password.
    - **Confirm new password**: Confirm new password entry.
    - **Uplode password**: Update new password.
- *Delete Account*: User can delete their account.

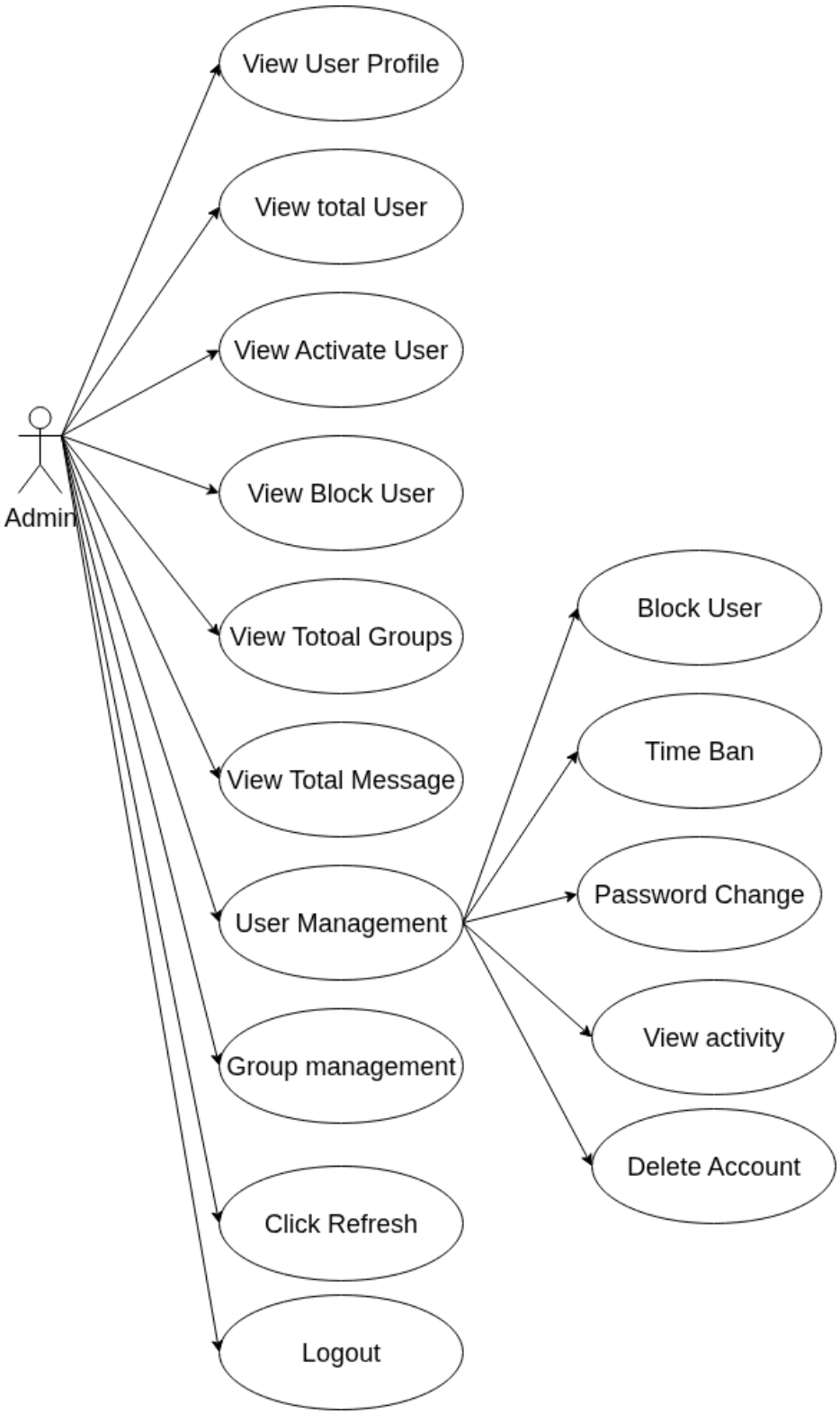


Figure 3.1.7 Admin Form

Admin Use Case Diagram

This diagram shows the main features and controls available to the admin in the system.1000000479.jpg

Step-by-Step Explanation

- *View User Profile*: Admin can see details of any user.
- *View Total User*: Admin can view the total number of users registered.
- *View Activate User*: Admin can check which users are currently activated.
- *View Block User*: Admin can see users that are blocked.
- *View Total Groups*: Admin can see all the groups in the system.
- *View Total Message*: Admin can view the total number of messages.
- *User Management*: Admin can manage users using:
    - *Block User*: Admin blocks a user.
    - *Time Ban*: Admin temporarily bans a user.
    - *Password Change*: Admin can change a user's password.
    - *Delete Account*: Admin can delete a user's account.
- *Group Management*: Admin can manage chat groups.
- *Click Refresh*: Admin refreshes the admin interface.
- *Logout*: Admin logs out of the admin panel.

3.2 Database Diagram

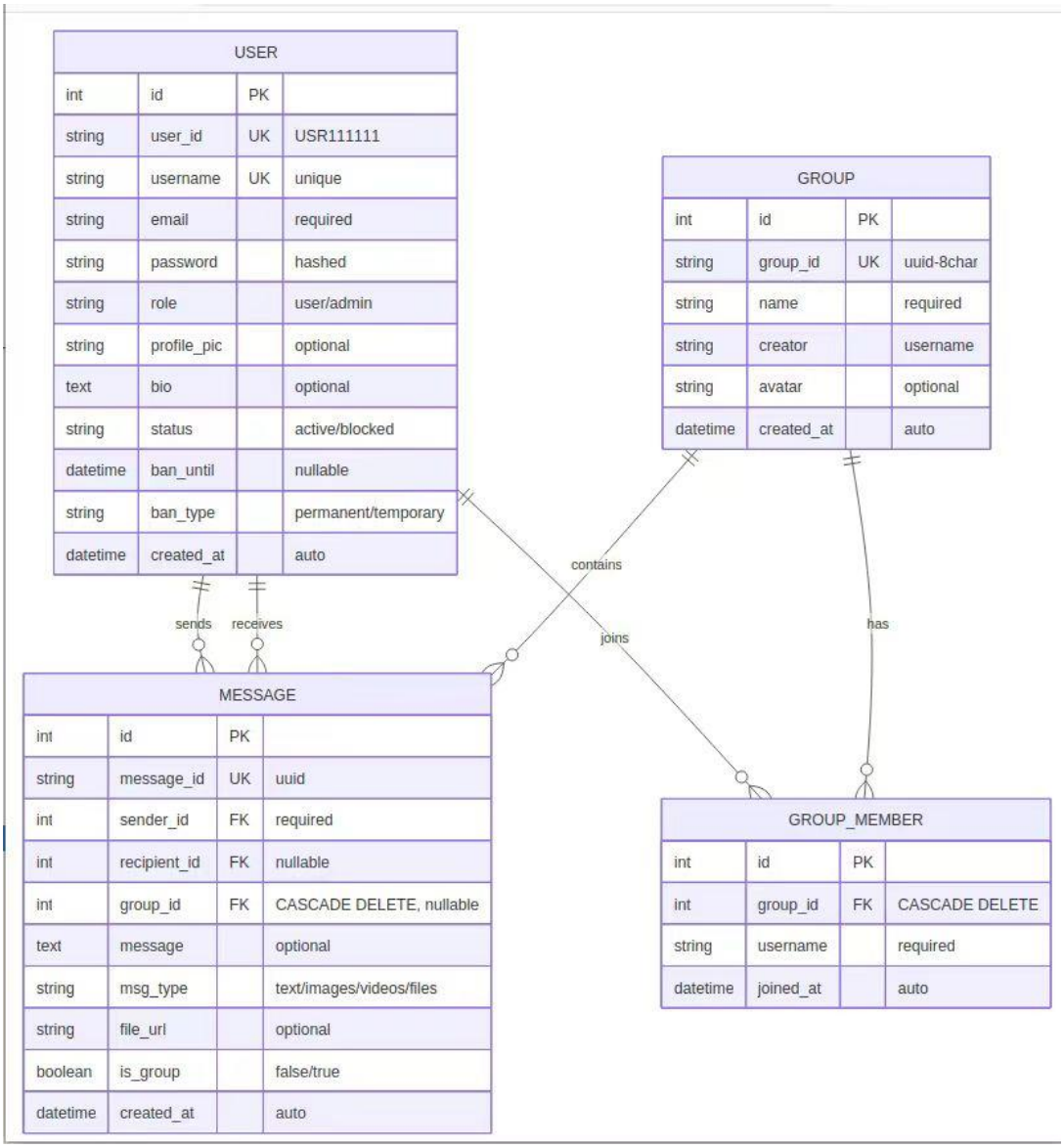


Figure 3.2.1 Database Diagram

Database ERD (Entity Relationship Diagram)

This diagram visualizes how app data is structured using tables and relationships.

Step-by-Step Explanation

- *USER Table*: Stores all user data such as user ID, username, email, password, role, profile pic, bio, status, ban details, and creation time.

- *GROUP Table*: Stores group details including group ID, name, creator, avatar, and creation time.
- *MESSAGE Table*: Each message record includes IDs, sender/recipient, group link, message content, message type, file URL, group status, and timestamps.
- *GROUP_MEMBER Table*: Connects users to groups with their joined time.
- Relationships are shown via Primary Key (PK) and Foreign Key (FK) links, mapping how records connect across these tables.

3.3 Data Flow Diagram

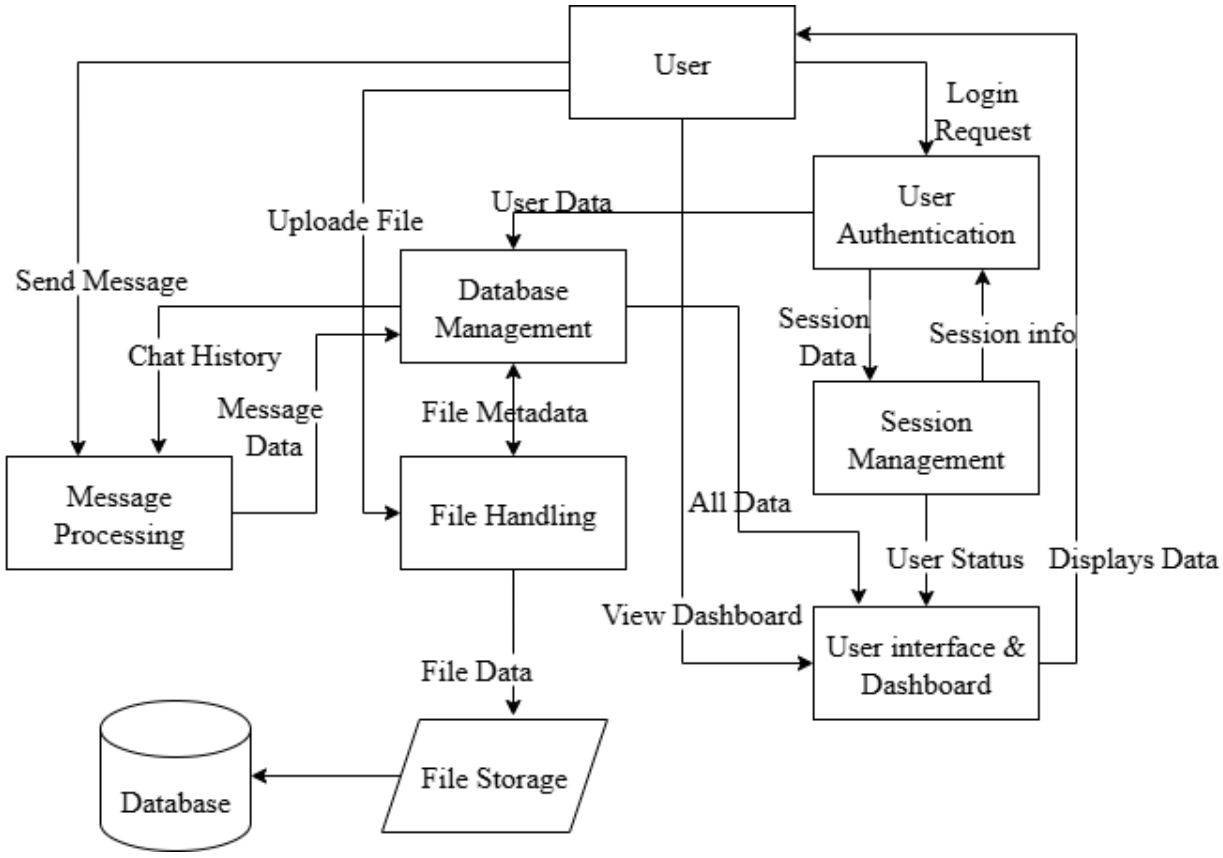


Figure 3.3.1 Data Flow Diagram

Data Flow

This diagram illustrates the main data flow and modular components of a chat application, similar to your previous example but with more detail and visual separation between the processing modules and data stores.

Step-by-Step Explanation

User: Interacts with the system and initiates actions such as login, sending messages, and uploading files.

Login Request: The user sends login credentials for authentication.

1.0 User Authentication: Handles login validation and passes user/session data to other modules.

2.0 Message Processing: Manages sending and receiving messages, passing chat history and message data to the database management module.

3.0 File Handling: Processes file uploads; passes file metadata to database management and stores file data in the file storage system.

4.0 User Interface & Dashboard: Displays data to the user and allows interaction with the dashboard and profile functions.

5.0 Session Management: Tracks session status and user activity, integrating with authentication and dashboard modules.

6.0 Database Management: Collects and organizes all data (user, session, message, file) and coordinates with both file storage and the database.

Database & File Storage: Stores persistent data, including all user records, files, and chat history. Receives data from the database management and file handling modules.

3.4 System Architecture Diagram

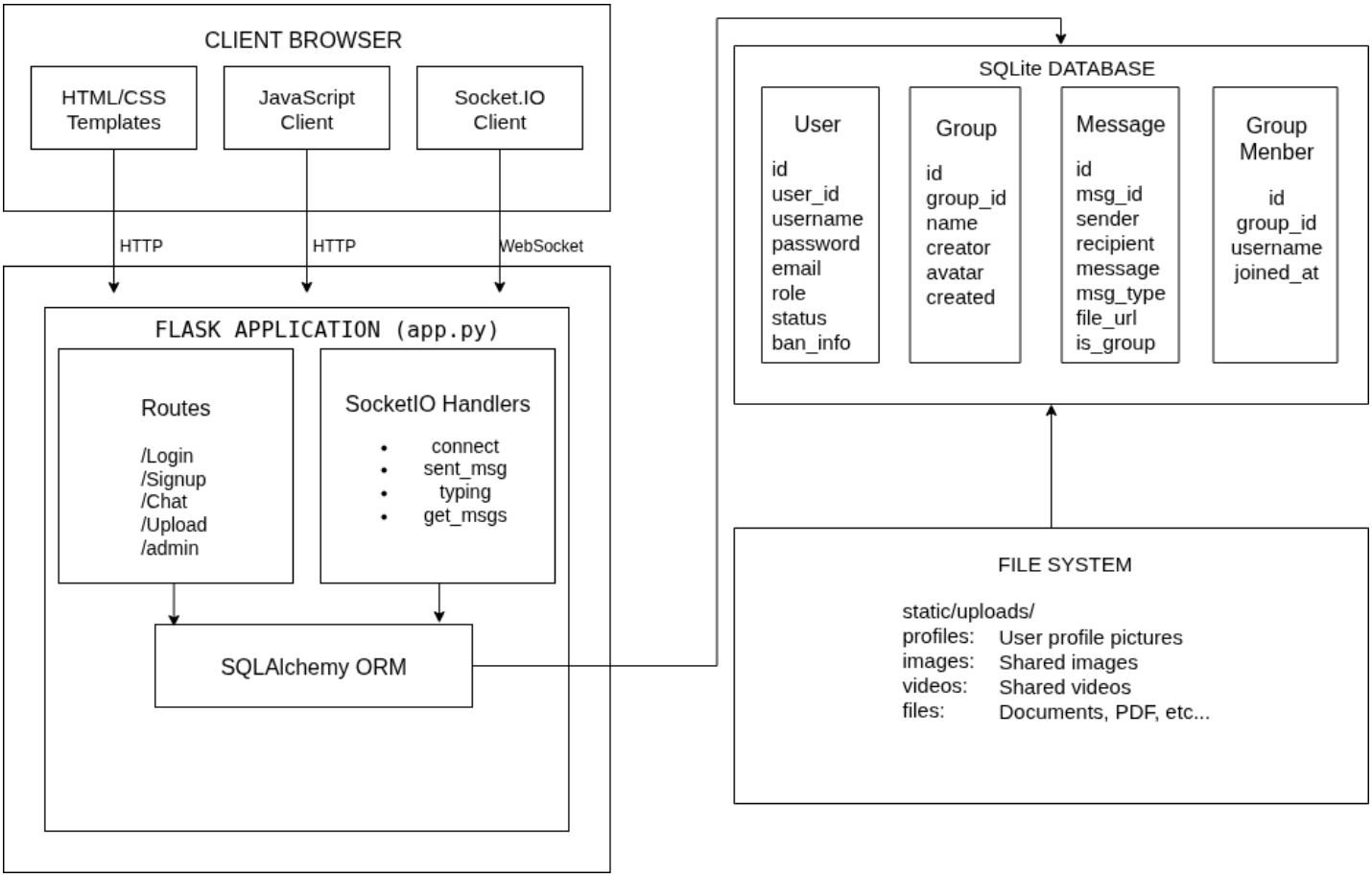


Figure 3.4.1 System Architecture Diagram

Architecture Components Description

Client Browser (Frontend Layer)

- HTML/CSS Templates: To display the basic structure and design of the system.
- JavaScript Client: To handle dynamic content updates, form validation, and user interactions.
- Socket.IO Client: To maintain a WebSocket connection for real-time messaging, file sharing, and user status updates.

Flask Application Server (Backend Layer)

- Routes: Receives HTTP requests and directs them to the appropriate services.
  - login, signup → User authentication
  - chat → Chat interface rendering
  - upload → File upload handling

- admin → Admin panel access
- Socket.IO Handlers: Handles real-time events.
  - connect → New user connection
  - send message → Message sending and broadcasting
  - typing → User typing indicator
  - get messages → Fetch chat history
- SQLAlchemy ORM: Uses an object-relational mapping tool to interact with the database.

SQLite Database (Data Layer)

- User Table: Stores user accounts, roles, status, and ban information.
- Group Table: For creating chat groups.
- Message Table: Stores messages sent between users.
- Group Member Table: Tracks members joining groups.

File System (Storage Layer)

- static/uploads/: Directory structure for storing files uploaded by users.
  - profiles → User profile pictures
  - images → Shared images in chats
  - videos → Shared videos
  - files → Documents, PDFs, and other files

Data Flow Explanation

1. User Authentication:
   - The user fills out the login form in the browser and sends an HTTP POST request to the `/login` route.
   - The Flask server checks the credentials in the User table and initiates a session.

2. Real-time Messaging:
   - The user types a message and clicks Send.
   - The JavaScript sends the `send msg` event to the server via the Socket.IO client.
   - The server saves the message in the database and broadcasts it to the recipient(s).

3. File Sharing:
   - The user selects a file via the upload interface and initiates the upload.
   - The file arrives at the `/upload` route as an HTTP request.

- The server saves the file in the `static/uploads/` folder and records the file metadata in the database.

4. Admin Operations:
   - The admin user accesses the `/admin` route to perform user management, ban/unban, and system monitoring.

3.5 Graphical User Interface

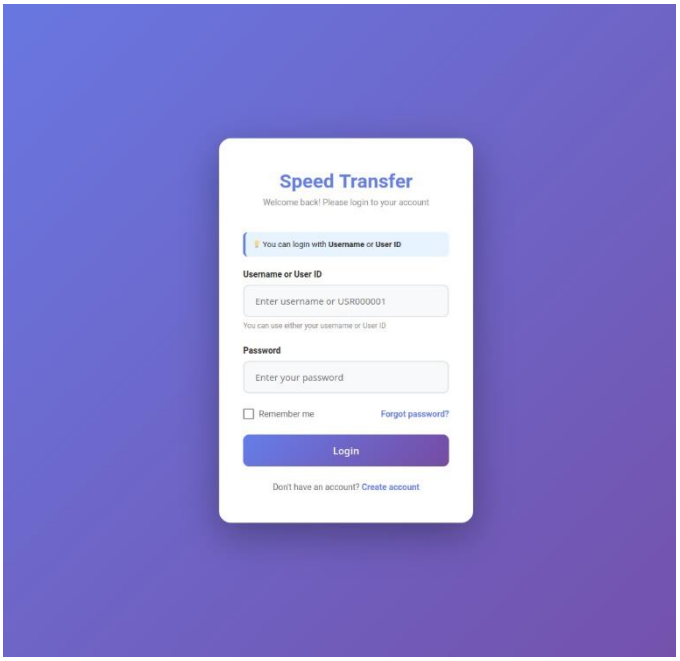


Figure 3.5.1 User Login wireframe

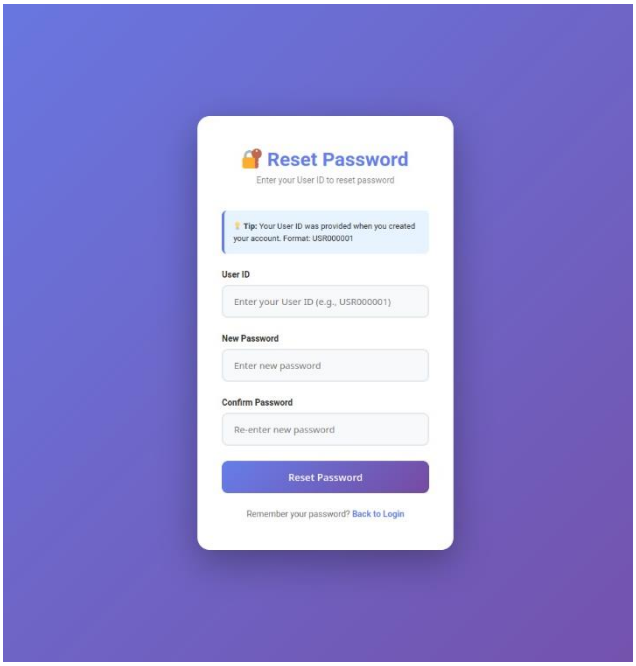


Figure 3.5.2 Forgot Password Wireframe

The first step to access our app is the Login UI. Existing users must enter their information to log in. This includes fields such as Username (or User ID), Password, and options like “Remember Me” and “Forgot Password” before proceeding to login**.**

Users with an existing account can log in. If the user forgets their password, they can reset it by entering their User ID and creating a new password to regain access to their account.

(Note: Password recovery can only be done using the User ID. If the User ID is forgotten, the user cannot reset their password. In such cases, they must contact the Admin to recover or restore account access.)

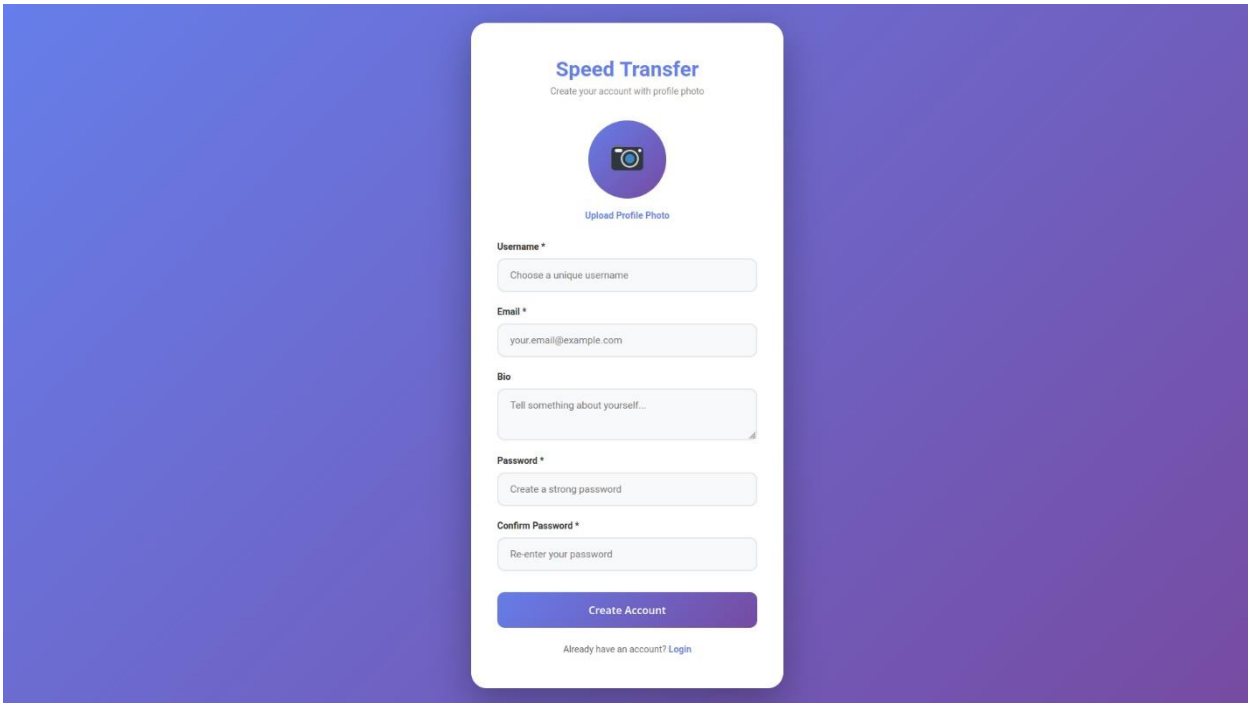


Figure 3.5.3 User Create Account wireframe

This is the Signup form, designed for users who do not have an account yet. To create a new account, users are required to fill in their Username, Email, Bio, and Password.

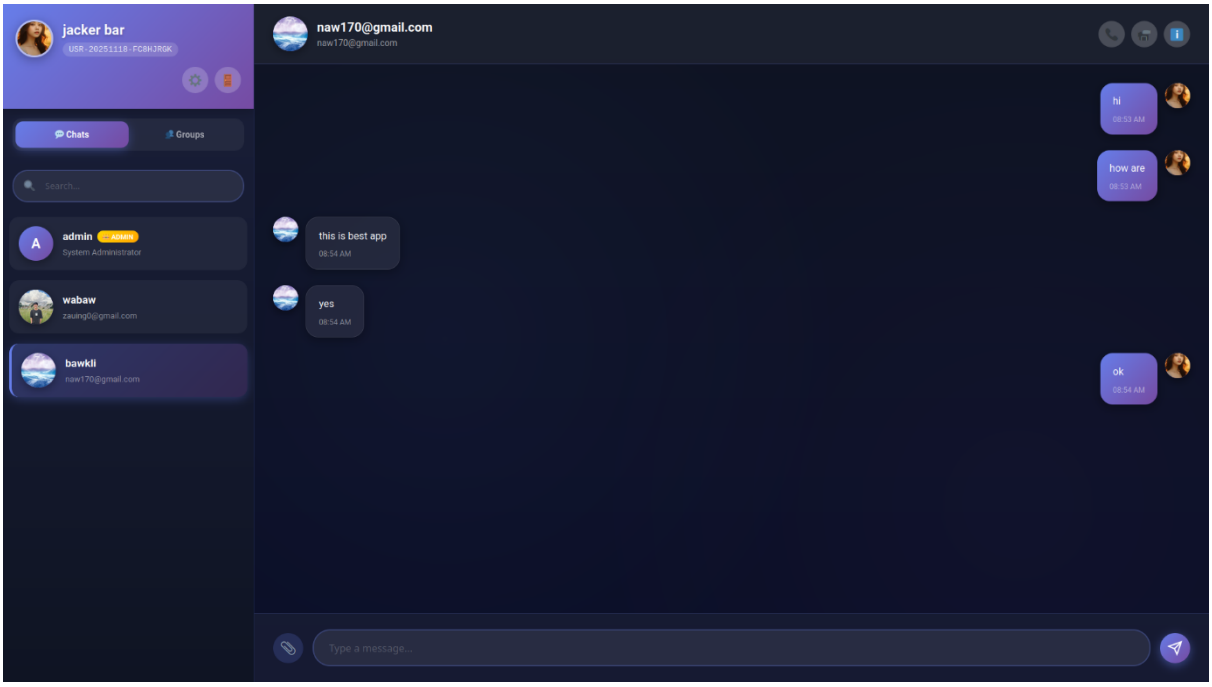


Figure 3.5.4 Chat Page Wireframe

Chat Page Wireframe

This image shows the interface after successfully logging into the account. From here, the user can access various features such as Settings, Searching for Users, Searching for Groups, Create Group, Selecting Users or Groups for chatting with friends, and Logging Out.

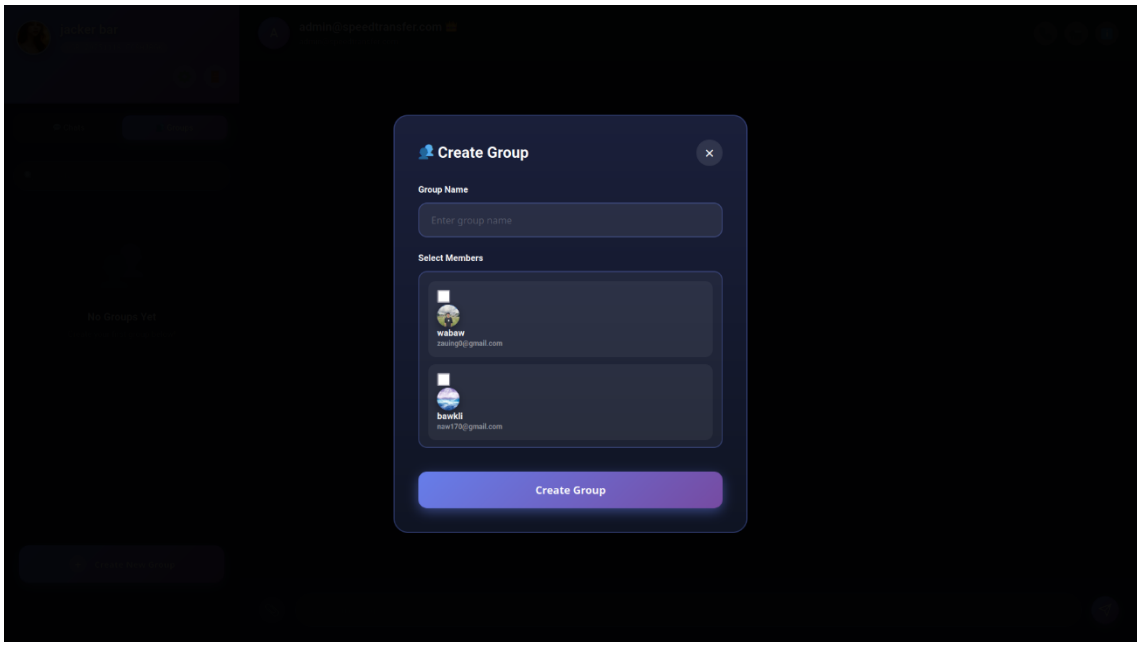


Figure 3.5.5 Create group wireframe

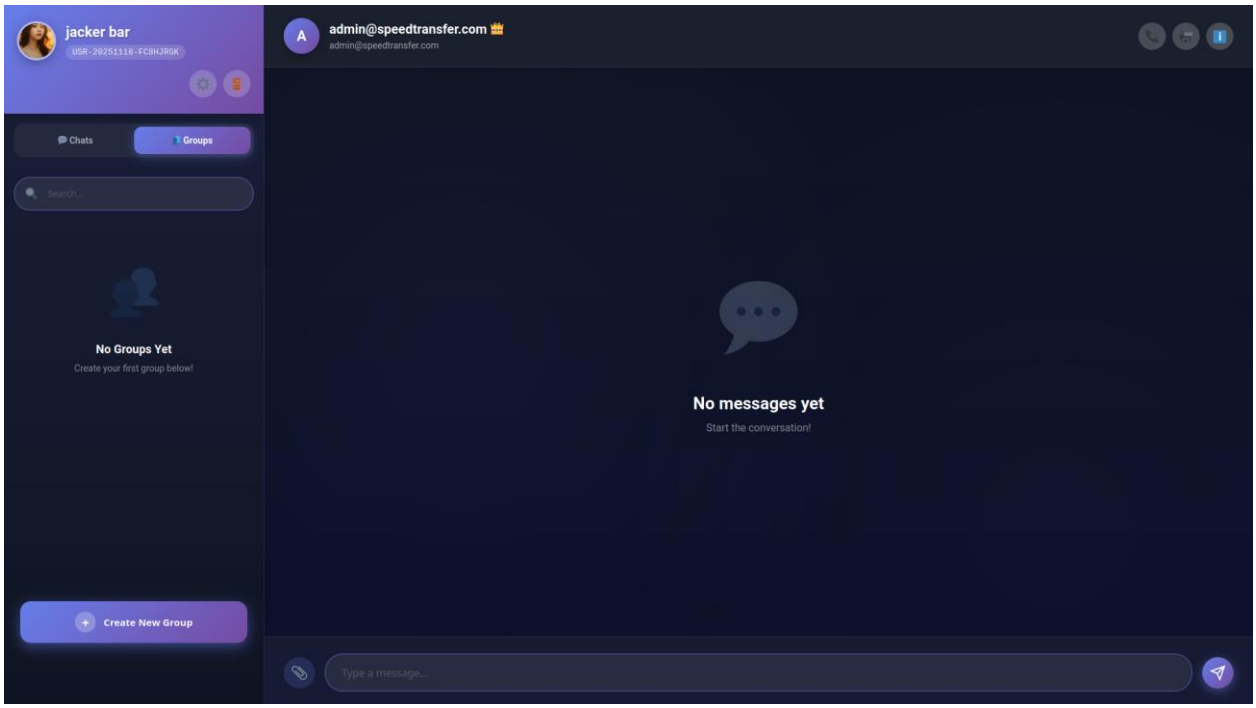


Figure 3.5.6 group chat wireframe

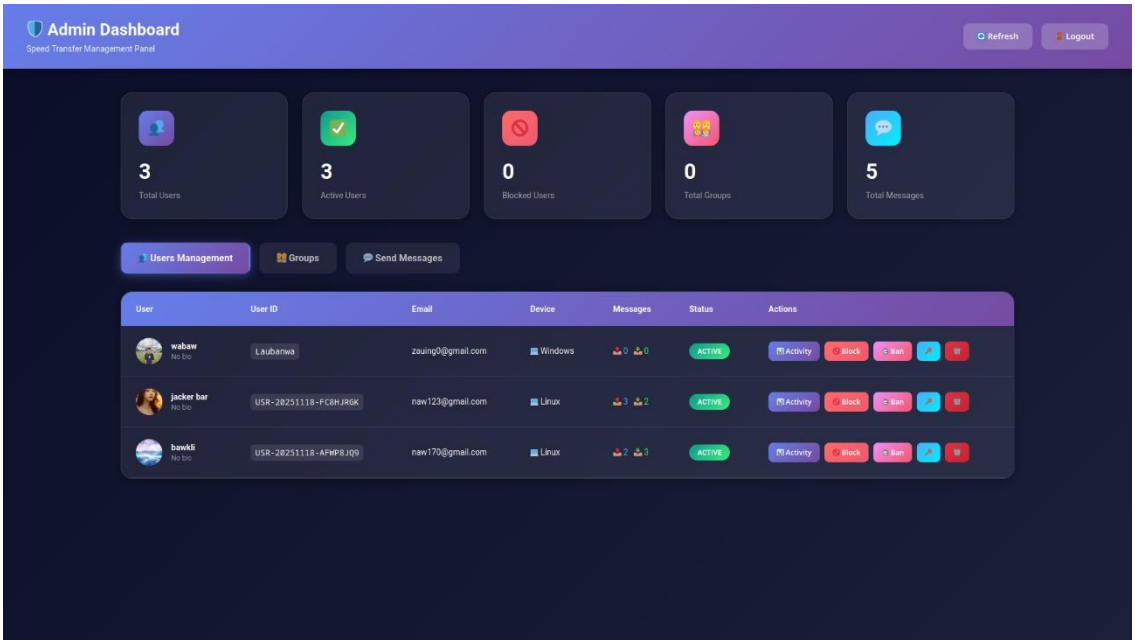


Figure 3.5.7 Admin page1 wireframe

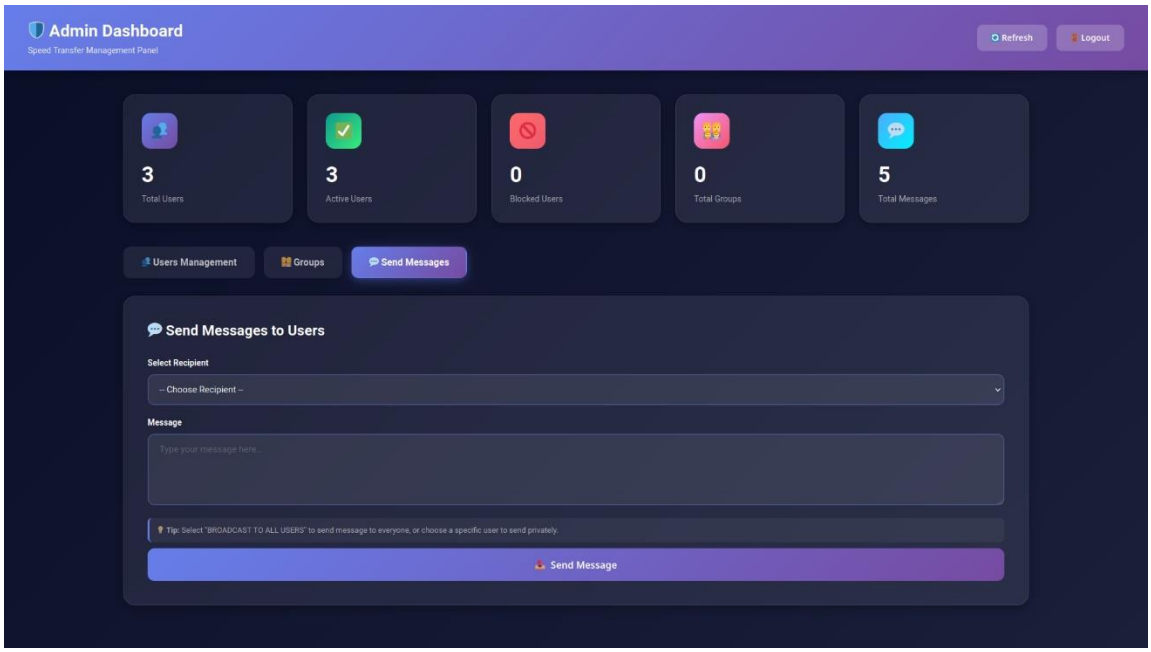


Figure 3.5.8 Admin page2 wireframe

This image represents the Admin Page, which is designed for managing and monitoring users. For monitoring purposes, the admin can view information such as the total number of users, active users, blocked users, total groups, and total messages.

For management, the admin can perform actions such as blocking users, banning users, changing passwords, and deleting accounts. The admin can also send messages to users and manage groups.

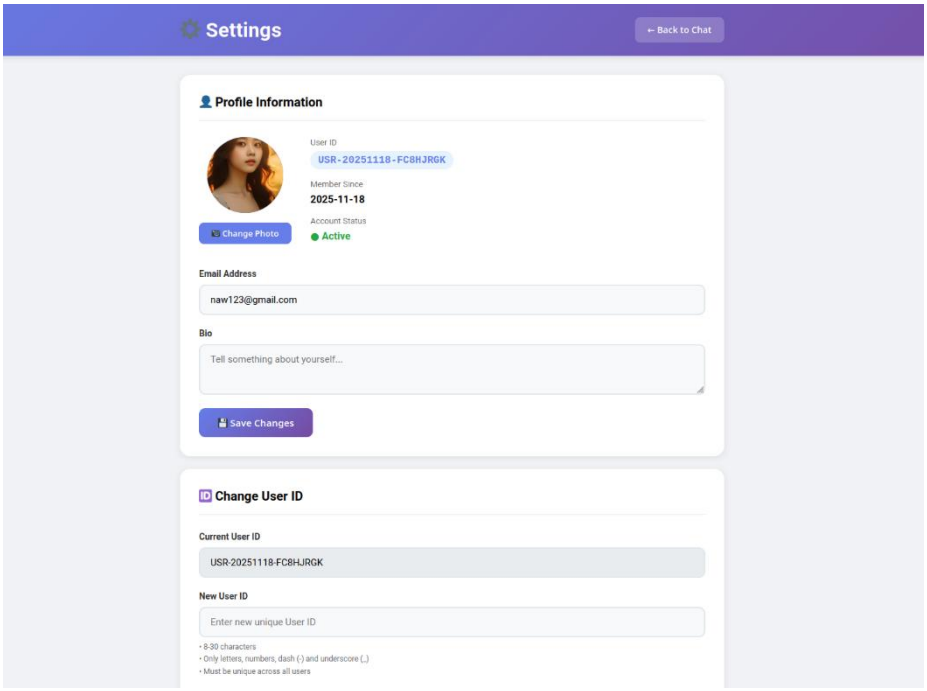


Figure 3.5.9 setting1 wireframe

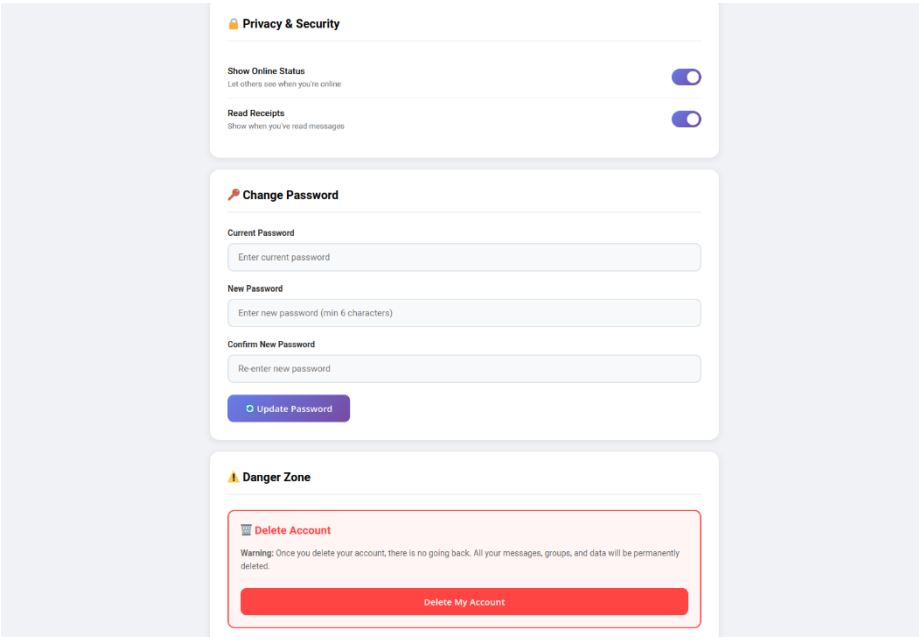


Figure 3.5.10 setting2 wireframe

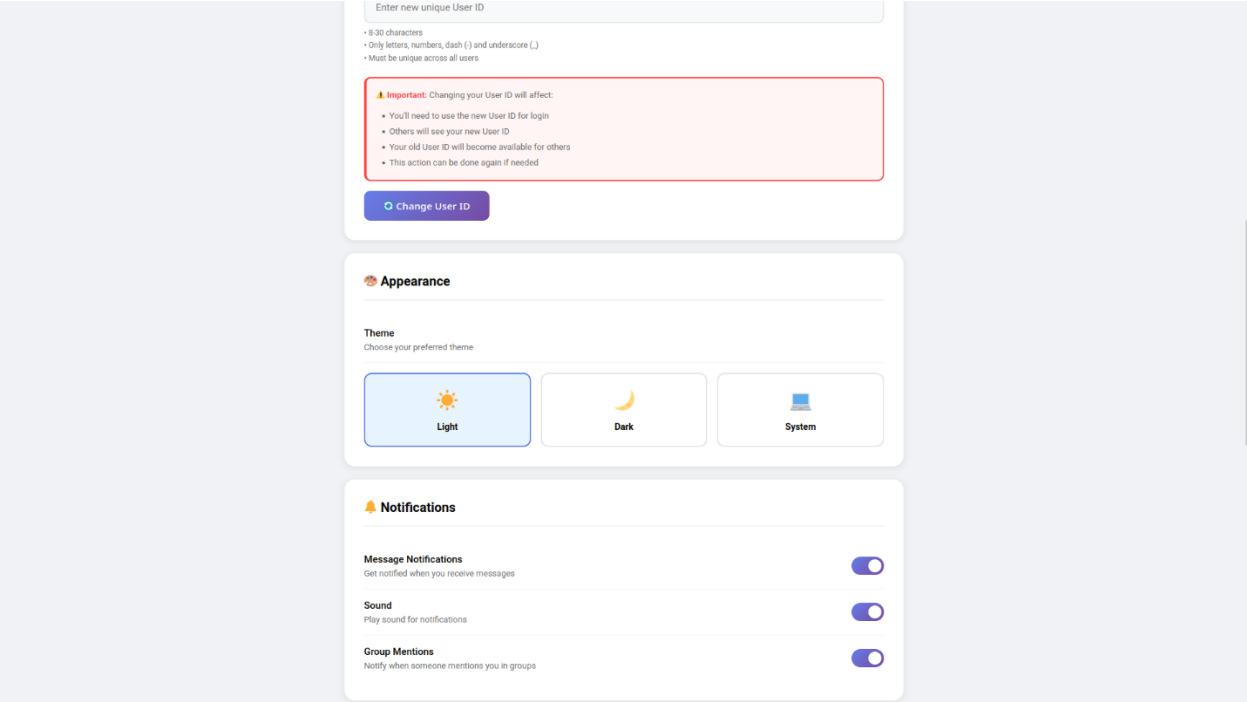


Figure 3.5.11 setting3 wireframe

The figures show the Settings section within the user dashboard. Here, users can customize and modify various settings such as changing their profile, viewing their email, adding a bio, changing their User ID, managing notifications, adjusting privacy & security settings, changing their password, and deleting their account.

If users are not satisfied with their password or User ID, they can easily update and modify them through the Settings section.

CHAPTER 4
Methodology and Technologies

4.1 How to install ProxmoxVE server

Proxmox VE installation is straightforward using the official ISO installer. The process involves creating a bootable USB, running the installer on your server, and completing basic setup through the web interface.

Here is a step-by-step guide to get your Proxmox VE server up and running.

Preparation Before Installation
Before you start, make sure you have the following ready:

· A Dedicated Server: For the best performance and stability, it is recommended to install Proxmox VE on bare-metal hardware rather than as a virtual machine.

· System Requirements: A 64-bit CPU, at least 1 GB of RAM (with additional RAM needed for guest VMs and containers), and a bootable USB stick with at least 1 GB of storage.

· Proxmox VE ISO Image: Download the latest version from the official Proxmox website.

· A Bootable USB Drive: You will need to create one using the downloaded ISO file.

Installation Steps
Follow these steps to install Proxmox VE on your server.

Step 1: Create a Bootable USB Installer

a. Use a tool like Rufus (for Windows) or Etcher (for Windows, macOS, Linux) to write the ISO file to your USB drive.
b. On a Linux system, you can use the dd command. First, identify your USB device name (e.g., /dev/sdb) using lsblk, then run:

```bash
dd bs=1M conv=fdatasync if=./proxmox-ve_*.iso of=/dev/your_usb_device
```

Warning: This will erase all data on the target USB drive, so double-check the device name.

Step 2: Boot and Start the Installation

a. Insert the USB drive into your server.
b. Boot the server and enter the boot menu (typically by pressing Esc, F2, F10, F11, or F12).
c. Select your USB drive as the boot device.
d. From the Proxmox VE menu, select Install Proxmox VE (Graphical) to start the standard installation.

Step 3: Run the Installation Wizard

a. The graphical installer will guide you through several configuration screens:
b. EULA: Read and accept the End User License Agreement.
c. Target Disk: Select the hard disk where Proxmox VE will be installed. The installer will use the entire disk, and all existing data will be erased. You can click Options to choose a filesystem; ext4 is the default.
d. Location and Time Zone: Set your location, time zone, and keyboard layout. The installer usually auto-detects these settings.
e. Administrator Password: Create a strong password for the root user and enter a valid email address for system notifications.
f. Network Configuration:
    i. Management Interface: Select the network card for the server management traffic.
    ii. Hostname: Assign a name for your server (e.g., pve or proxmox).
    iii. IP Address, Gateway, and DNS Server: Configure a static IP address for your server to ensure it doesn't change. During installation, you can configure either IPv4 or IPv6; a dual-stack configuration can be set up later.
g. Installation Summary: Review all your settings. If everything is correct, click Install. The installer will format the disks and copy the necessary files.

Once the installation is complete, remove the USB drive and let the server reboot.

Post-Installation Setup

After the server reboots, you will see a welcome message on the console with a URL.

a. Access the Web Interface: From any computer on the same local network, open a web browser and go to https://[YOUR_SERVER_IP]:8006 (for example, https://192.168.1.100:8006).
b. Log In: Use root as the username and the password you created during installation.
c. Subscription Notice: You may see a pop-up message about no valid subscription. You can safely click OK for now, as the product is free to use with access to public repositories.

Next Steps for Your Chat Application

With Proxmox VE ready, you can now create virtual environments to host your chat application.

a. Create a Virtual Machine: For a Python Flask application, you can efficiently use a full Virtual Machine.
b. In the Proxmox web interface, click "Create VM".
c. Follow the wizard to allocate resources like CPU cores, memory, and disk space.
d. Install Your OS: For the container/VM, choose a lightweight Linux distribution like Ubuntu Server or Debian.
e. Deploy Your Application: Inside your new container or VM, you can now install Python, set up your Flask environment, and deploy your LAN chat application files.

4.2 How to install Linux for Debian

Preparation

System Requirements

- Minimum: 1GB RAM, 10GB disk space
- Recommended: 2GB+ RAM, 20GB+ disk space
- Architecture: AMD64 (64-bit)

Download Debian ISO

1. Visit: debian.org/download
2. Choose:
   - Netinst (Small ~500MB, downloads packages during install) - Recommended
   - Full DVD (~4GB, complete offline installation)

Create Bootable USB

```bash
Using dd on Linux/Mac
sudo dd if=debian-xx.x.x-amd64-netinst.iso of=/dev/sdX bs=4M status=progress
Using Rufus on Windows
1. Download Rufus from rufus.ie
2. Select ISO and USB drive
3. Use DD mode or ISO mode
```

Installation Process

***Step 1*:** Boot from USB

1. Insert bootable USB

2. Restart computer

3. Enter BIOS/UEFI (F2, F12, Del, Esc)

4. Set USB as first boot device

5. Save and exit

***Step 2*:** Debian Installer Startup

You'll see the GRUB menu:

1. Install

2. Graphical install ← Recommended

3. Advanced options...

Select "Graphical install" for easier installation.

***Step 3*:** Language and Location

· Language: English (or your preference)

· Location: Your country

· Keymap: Your keyboard layout

***Step 4*:** Network Configuration

```bash
- Installer will automatically detect network
- For manual configuration:
- Hostname: debian-server (or your preferred name)
- Domain name: local (or your domain)
```

***Step 5*:** User Account Setup

```bash
- Root password - create strong password
  Root password: ********
  Verify: ********
- Create regular user
  Full name: Your Name
  Username: yourusername
  Password: ********
```

Step 6: Disk Partitioning - Important Step

Option A: Guided - Use entire disk (Recommended for beginners)

```
1. Guided - use entire disk
2. All files in one partition
3. Finish partitioning and write changes to disk
```

Option B: Manual Partitioning (Recommended for advanced users)

```
Suggested layout:
- /: 20-30GB (ext4) - Root partition
- /home: Remaining (ext4) - User files
- swap: 2-4GB     - Swap space

Steps:
1. Select free space
2. Create root partition (/) - Primary - Beginning - ext4
3. Create swap partition
4. Create home partition (/home)
5. Finish partitioning
```

Step 7: Package Selection

Software Selection:

· Debian desktop environment (for GUI)

· [✓] SSH server - Important for remote access

· Web server

· Print server

· Standard system utilities

For your chat application server, only select "SSH server"

Step 8: GRUB Bootloader

· Install GRUB to /dev/sda (main disk)

· This enables booting into Debian

Step 9: Finish Installation

· Remove installation media when prompted

· System will reboot into Debian

System Update

```bash
# Update package lists
sudo apt update

# Upgrade installed packages
sudo apt upgrade -y

# Optional: Full system upgrade
sudo apt full-upgrade -y
```

Create VM in Proxmox:

1. VM Settings:

OS: Use Debian ISO

System: Default (Q35, OVMF if UEFI)

Disk: 20GB+, SCSI/VirtIO

CPU: 2+ cores, Type: host

Memory: 2048MB+

Network: VirtIO, vmbr0

2. Install Debian following the main installation steps above

Create Application Environment

```bash
# Create application directory
sudo mkdir -p /opt/chat-app
sudo chown $USER:$USER /opt/chat-app

# Set up Python virtual environment
cd /opt/chat-app
python3 -m venv venv
source venv/bin/activate

# Install Python dependencies
pip install flask flask-socketio python-socketio eventlet

# Clone your application code
git clone <your-repository>.
```

4.3 Implementation

The "SpeedChat" system was built and set up by following a clear, step-by-step plan. The main goal was to create a working chat and file-sharing application that runs on a Local Area Network (LAN).

4.3.1. Setting Up the Server (The Foundation)

First, we needed a strong and reliable server to host our chat application. For this, we used Proxmox VE.

- We installed Proxmox on a dedicated computer that would act as our main server.
- After installation, we accessed its web interface to manage everything.
- Inside Proxmox, we created a Virtual Machine (VM). Think of this as a separate, virtual computer inside our main server.
- On this virtual machine, we installed a lightweight operating system called Debian Linux.

4.3.2. Developing the Chat Application

The application itself has two main parts: The Backend (the brain) and the Frontend (the face).

Backend (Server-Side Logic with Python & Flask):

- We used Python and a simple web framework called Flask to create the server's logic.
- This part handles user logins, sending and receiving messages, and file sharing.
- For real-time chat, we used Socket.IO. This allows messages to appear instantly on everyone's screen without needing to refresh the page.
- SQLite, a simple database, was used to store all user information, messages, and file records.

Frontend (User Interface with HTML, CSS, JavaScript):

- We built the website that users see and interact with using HTML (for structure), CSS (for styling and looks), and JavaScript (for making the page dynamic).
- JavaScript also helps connect the webpage to the server using Web Sockets for live messaging.
- The frontend includes pages for:
  1. Login and Sign-up
  2. A main chat dashboard to see messages and online users
  3. A settings page to change your profile or password

4.3.3. Connecting Everything on the LAN

- The server (the Proxmox machine) was given a static IP address on the local network. This means its address never changes, so users can always find it.
- The Flask application was run on this server.
- Now, any computer, laptop, or phone connected to the same Wi-Fi or network cable (the same LAN) can open a web browser.
- The user just types the server's IP address into their browser (like `http://192.168.1.100`) to access the SpeedChat application and start chatting.

CHAPTER 5
Conclusion & Testing

5.1 Testing

5.1.1 Testing Objectives

- To verify that all system functions operate as specified in the requirements.
- To validate the system's stability and performance under load.
- To check security aspects, including login, authentication, and data protection.
- To ensure the User Interface (UI) works correctly across different browsers and devices.

5.1.2 Functional Testing

Test Case ID	Test Case Description	Expected Result
TC-001	New user registration (Sign Up)	Account is created successfully.
TC-002	Login using either Username or User ID	Successful login, user is redirected to the Chat Dashboard.
TC-003	Attempt to login with an incorrect password	An error message is displayed, and login is denied.
TC-004	Password reset via "Forgot Password"	User can set a new password after providing their User ID.
TC-005	Changing the User ID	User can successfully log in using the new User ID.
TC-006	Editing Profile Information (Email, Bio)	Modified information is saved and displayed correctly after saving.

Table 5.1.2.1 User account & Management

Test Case ID	Test Case Description	Expected Result
TC-007	Sending a Private Message to another user	The message is displayed in both the sender's and recipient's chat interfaces.
TC-008	Sending a message in a Group Chat	The message is received by all members of the group.
TC-009	Viewing Message History	Previously sent and received messages are visible and accessible.
TC-010	Sending a File/Image/Video	Files within the specified size limit are sent and received successfully.

Table 5.1.2.2 Messaging

Test Case ID	Test Case Description	Expected Result
TC-011	Creating a new group	A group can be created by providing a group name and selecting members.
TC-012	Sending a Message/File within a group	The Message/File is delivered to all group members.

Table 5.1.2.3 Group Management

Test Case ID	Test Case Description	Expected Result
TC-007	Admin user login	Successful login and access to the Admin Dashboard.
TC-008	Viewing the list of users	The admin can view details of all users (User ID, Email, Status).
TC-009	Blocking/Unblocking a user	A blocked user cannot log into the system.
TC-010	Sending a Broadcast Message	The message is delivered to all active users.

Table 5.1.2.4 Admin Management

5.1.3 Non-Functional Testing

Performance Testing

- **Concurrent Users:** 10-20 users logging in and chatting simultaneously. The system should not hang or experience significant delay.
- **File Transfer:** Test sending files of various sizes (e.g., 10MB, 50MB, 100MB) to verify performance and adherence to size limits.

Usability Testing

- The UI should be intuitive, clear, and easy to navigate for a new user.
- Verify that the theme can be switched between Light and Dark modes.

Security Testing

- Verify that passwords are stored in a hashed format in the database.
- Check that user sessions expire after a period of inactivity (session timeout).

Compatibility Testing

- **Browsers:** Test core functionality on different browsers (Chrome, Firefox, Edge).
- **Devices:** Verify that the UI is responsive and functional on mobile browsers (Android/iOS).

5.1.4 Test Environment

- **Server:** Proxmox VE environment with Debian VM/Container.
- **Client:** Windows, Linux, and Mac PCs; Android and iOS mobile devices.
- **Network:** Local Area Network (LAN) only. Internet connection should not be required.
- **Browsers:** Latest versions of Chrome, Firefox, and Edge.

5.1.5. Test Data

- A set of 5-10 test user accounts (including one admin account).
- Sample test messages and files (images, PDFs, videos) of various formats and sizes.

5.1.6. Defect Management

- Any bugs found during testing will be recorded in a Bug Tracking Tool (e.g., Excel, Trello, Jira) with the following details:

    1. Bug ID
    2. Description
    3. Steps to Reproduce
    4. Expected vs. Actual Result
    5. Severity (Critical, Major, Minor)
    6. Status (Open, In Progress, Fixed, Closed)

5.2 Conclusion

The Local Area Network (LAN) Based Chat and File Sharing System developed in this project successfully fulfills its primary objectives of providing secure, fast, and reliable communication within a local network environment. By combining Python backend services, WebSocket-based messaging, and a user-friendly front-end interface, the system offers an independent platform that functions even without internet connectivity. This makes it especially useful for schools, offices, and organizations that require efficient internal communication.

Through the use of Proxmox virtualization, LXC containers, and a structured LAN setup, the deployment environment became more stable, manageable, and scalable. Core features such as user authentication, real-time chat, file transfer, and session control were successfully implemented and tested. The system ensures smooth operation on a LAN while maintaining both security and performance.

Overall, the project demonstrates strong technical understanding in networking, Python development, system architecture design, and front-end integration. Although certain limitations exist—such as user capacity, file size restrictions, and the absence of advanced communication features—the system provides a solid foundation for future improvement. It proves that a LAN-based communication platform can operate effectively, deliver real-time interaction, and meet the communication needs of small to medium-sized environments.

5.3 Future Work

Although the system is functional and meets its key objectives, several improvements can enhance performance, usability, and scalability. The following recommendations are proposed for future development:

1. Improve Performance for Larger User Groups

Optimize server processing and database handling.

Implement load balancing for environments with more than exiting users.

Monitor memory usage to reduce resource consumption during heavy file transfers.

2. Add End-to-End Encryption

Implement encryption for messages and file transfers to increase security.

Use SSL/TLS for communication between clients and server.

3. Add Audio, Video Call, and Voice Messaging

Integrate WebRTC for voice/video communication.

Add screen-sharing and group meeting options for schools and offices.

4. Enhance the User Interface (UI/UX)

Introduce a modern responsive design.

Add dark/light theme options with user customization.

Improve notification systems (sound, popup alerts).

5. Expand File Management Features

Add file preview (image, PDF).

Implement file compression before transfer.

Add cloud backup using local NAS or external storage.

6. Develop a Mobile Version

Build an Android app using Flutter or React Native.

Allow cross-device communication across phones and PCs on the same LAN.

7. Include Admin Dashboard Enhancements

Real-time monitoring of all connected users.

Log and audit history for security purposes.

User management tools (ban users, reset passwords, log analysis).

8. Implement Offline Message Storage

Store messages when a user is offline and deliver them once the user logs in.

Ensure message persistence across sessions.

5.4 References

- https://developer.mozilla.org
- https://react.dev
- https://www.w3schools.com
- https://www.plantuml.com/plantuml/umla/SoWkIImgAStDuNBCoKnELT2rKt3AJx9IS2mjoKZDAybCJYp9pCzJ24ejB4qjBk42oYde0jM05MDHLLoGdrUSoeLkM5u-K5sHGY9MGw6ARNHryQb66EwGcfS2T300#google_vignette
- https://www.flaticon.com/
- www.geeksforgeeks.org

Project Documents: https://github.com/JaSengHtoi/document