Ain Shams University Faculty of Engineering

CSE351: Computer Networks

Under the surveillance of Professor Ayman Bahaa.

Peer-to-Peer Multi-User Chatting Application

Group 7

Kareem Wael Hasan Ahmed	2001151
Malak ahmed yehia sherif	2001350
hussien ahmad abdelgelil mohammed khalifa	2000459
Mahmoud Talaat El-sayed Rezk	2001366

Abstract

In an era of digital connectivity, a novel peer-to-peer multi-user chat application emerges, built upon the foundation of Python and socket programming. It is a p2p chat application that uses centralized index approach. Through the creation and management of chat rooms, users can engage in both individual and group conversations. The application safeguards users with a secure authentication system, ensuring privacy and trust within the virtual environment. Beyond basic text-based messaging, the platform offers functionalities like text formatting and hyperlink sharing, enhancing the overall communication experience. By prioritizing user-friendliness through a simple command-line interface and visually distinct color-coded messages, the application caters to diverse user preferences. Moreover, robust error handling and automatic network reconnection ensure that user experience remains seamless and uninterrupted. This project promises a significant contribution to the realm of online communication, offering a secure, reliable, and engaging platform for users to connect and share their experiences.

Keywords: peer-to-peer, Python, socket programming, centralized index approach, authentication system , text-based messaging, network reconnection .

Table of Contents

List of Figures

Project Proposal

1. Executive Summary

1.1 Project Title

Peer-to-Peer Multi-User Chatting Application

1.2 Project Overview

The project aims to develop a robust and user-friendly Peer-to-Peer Multi-User Chatting Application using Python and sockets. It is a p2p chat application that uses a centralized index approach. This application will enable users to authenticate, create and join chat rooms, send messages, initiate one-to-one chat sessions, and use basic text formatting. The implementation will also include features such as sharing hyperlinks, error handling, automatic reconnection, a command-line interface, and color-coded messages for enhanced visual distinction.

2. Objectives and Scope

2.1 Objectives

I. Implement a **SECURE AUTHENTICATION** system for users with unique usernames and passwords.

II. Develop a basic server application capable of handling **MULTIPLE CLIENT CONNECTIONS**.

III. Allow users to **CREATE AND JOIN** chat rooms for group communication.

IV. Enable users to **SEND AND RECEIVE MESSAGES** within chat rooms.

V. Facilitate **ONE-TO-ONE CHAT** sessions between users.

VI. Support basic **TEXT FORMATTING** (e.g., bold, italics) in messages.

VII. Allow users to share **HYPERLINKS** in messages.

VIII. Implement robust **ERROR HANDLING MECHANISMS** for unexpected scenarios.

IX. Automatically **RECONNECT** users in case of network interruptions.

X. Develop a user-friendly **COMMAND-LINE INTERFACE**.

XI. Enhance visual distinction with **COLOR-CODED** messages.

2.2 Scope

The project will focus on the development of a command-line-based chat application with the specified features. The application will support communication between multiple users in real-time, providing a seamless and secure chatting experience. The scope includes the implementation of a server component to manage user connections, chat rooms, and message distribution.

3. Goals

3.1 Primary Goals

I. Create a reliable and secure authentication mechanism.

II. Establish a functional server to handle multiple client connections concurrently.

III. Enable users to create, join, and communicate within chat rooms.

IV. Implement one-to-one chat sessions for private communication.

V. Support basic text formatting and hyperlink sharing.

3.2 Secondary Goals

I. Enhance user experience with a robust error handling system.

II. Implement automatic reconnection for users in case of network interruptions.

III. Develop a user-friendly command-line interface.

IV. Add color-coded messages for better visual distinction.

4. Functionalities

I. User Authentication

- ♦ *Unique username and password authentication for users.*

II. Server Application

- ♦ *Manage multiple client connections simultaneously.*
- ♦ *Facilitate communication between users and chat rooms.*

III. Chat Rooms

- ♦ *Allow users to create and join chat rooms.*

IV. Messaging

- ♦ *Enable users to send and receive messages within chat rooms.*
- ♦ *Support one-to-one chat sessions.*

V. Text Formatting

- ♦ *Implement basic text formatting (e.g., bold, italics) in messages.*

VI. Hyperlink Sharing

- ♦ *Allow users to share hyperlinks in messages.*

VII. Error Handling

- ♦ *Implement robust error handling for unexpected scenarios.*

VIII. Automatic Reconnection

- ♦ *Automatically reconnect users in case of network interruptions.*

IX. Command-Line Interface

- ♦ *Develop a user-friendly command-line interface for simplicity.*

X. Color-Coded Messages

- ♦ *Enhance visual distinction with color-coded messages.*

5. Timeline

Phase 1: Design and development of core functionalities

Phase 2: Implementation of user authentication and chat room management

Phase 3: Integration of messaging functionalities and text formatting

Phase 4: Development of one-to-one chat and hyperlink sharing

Phase 5: Implementation of error handling and network reconnection

Phase 6: Design and implementation of the command-line interface

Phase 7: Testing, debugging, and refinement

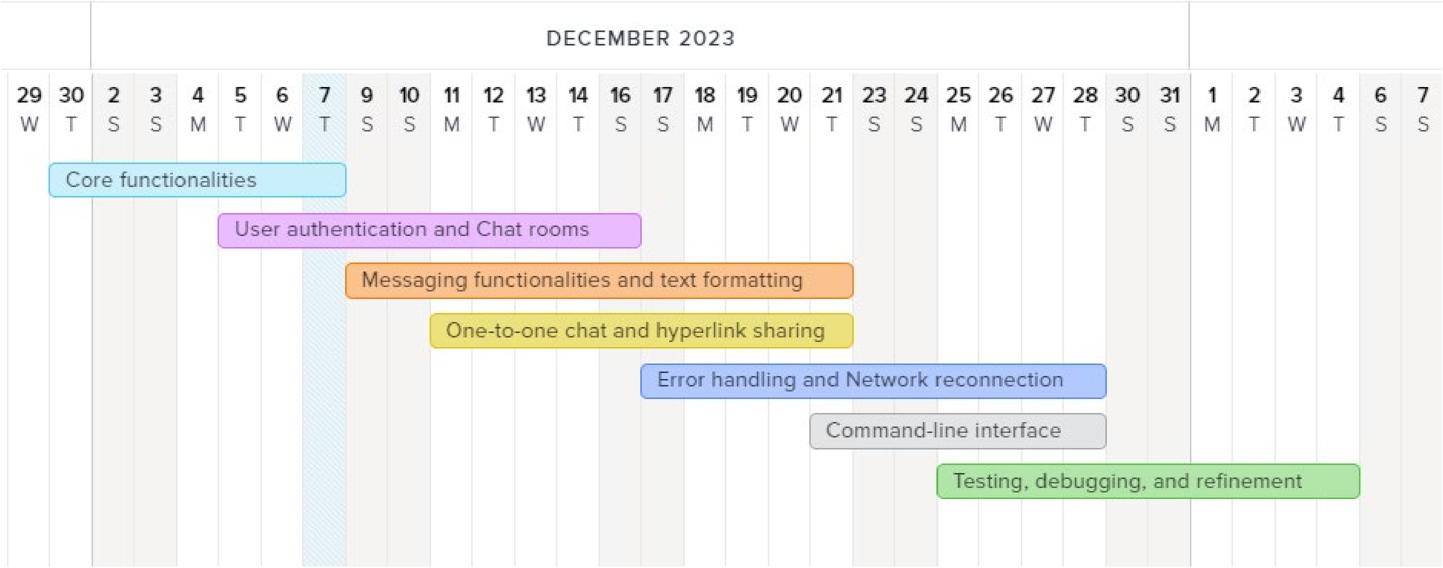

Figure 1:Gnatt Chart

6. System Architecture Overview

The Peer-to-Peer Multi-User Chatting Application will follow a distributed system architecture to ensure scalability, reliability, and responsiveness. The key components include:

6.1 Client-Side Components:

- **User Interface:** Facilitates user interactions, authentication, and input/output.
- **Chat Client:** Manages communication with the server and other clients.
- **Authentication Module:** Validates user credentials.

6.2 Server-Side Components:

- **Chat Server:** Manages multiple client connections, chat rooms, and messaging.
- **User Manager:** Handles user authentication, creation, and management.
- **Chat Room Manager:** Manages the creation and functionality of chat rooms.

6.3 Communication Protocols:

- **Authentication Protocol:** Specifies how clients authenticate with the server.
- **Chat Protocol:** Defines the format and rules for exchanging messages between clients and the server.

6.4 Error Handling and Reconnection:

- **Error Handling Module:** Manages unexpected scenarios and provides feedback to users.
- **Reconnection Module:** Handles the automatic reconnection of users in case of network interruptions.

6.5 User Interface Elements:

- **Command-Line Interface (CLI):** Represents the user interface for simplicity.
- **Message Formatting:** Includes elements for text formatting and hyperlink sharing.
- **Color-Coding Module:** Adds color-coded messages for visual distinction.

6.6 Security Measures:

- **Encryption Module:** Ensures secure communication between clients and the server.
- **Authentication Security Layer:** Implements robust user authentication mechanisms.

7. Non-Functional Requirements

7.1 Performance

- **Response Time:** The system should respond to user actions within 1 second.
- **Scalability:** The application should gracefully handle an increasing number of simultaneous users without compromising performance.

7.2 Reliability

- **Availability:** The system should be available 99.9% of the time.
- **Fault Tolerance:** The application should gracefully handle server failures and recover without data loss.

7.3 Security

- **Data Encryption:** All communications between clients and the server should be encrypted using industry-standard protocols.
- **Authentication Security:** User authentication should follow best practices to prevent unauthorized access.

7.4 Usability

- **Intuitiveness:** The user interface should be intuitive, requiring minimal training for new users.
- **Accessibility:** The application should be accessible to users with disabilities.

7.5 Maintainability

- **Code Maintainability:** Code should be well-documented and follow coding standards for ease of maintenance.
- **Server Logs:** Maintain detailed logs for monitoring and debugging purposes.

8. Challenges Faced

The development of the Peer-to-Peer Multi-User Chatting Application may encounter the following challenges:

- **Network Variability:** Managing communication in diverse network conditions and ensuring a consistent user experience.
- **Cross-Browser Compatibility:** Ensuring consistent behavior across different web browsers.
- **Real-Time Communication:** Implementing efficient real-time communication between clients and the server using web sockets.
- **Cross-Platform Compatibility:** Addressing differences in platform-specific guidelines and APIs for mobile development.
- **Background Execution:** Handling restrictions on background tasks for both mobile and web environments.

9. Cost Analysis

This document uses basic COCOMO model for the cost analysis since this is a small scale project.

Assumptions:

1. **Size:** We'll estimate the size of the software project using function points (FP). Let's assume a total of 10 function points for the P2P chat network application.
2. **Effort Multipliers:** We'll consider a set of effort multipliers to account for various factors influencing the development effort. These assumptions fits a typical small project for undergrads:
   - Product attributes: 0.8
   - Personnel attributes: 0.8
   - Project attributes: 0.8
   - Computer attributes: 0.8
   - Language and tools: 0.8
   - Development flexibility: 0.8
3. **Development Team:** The development team consists of 4 members.
4. **Average Monthly Cost per Team Member:** We'll assume an average monthly cost of $500 per team member, including salaries, benefits, and overhead (Average fresh grad software engineering payroll in Egypt).

Analysis:

1. **Effort Multiplier:** 0.8 * 0.8* 0.8 * 0.8 * 0.8 * 0.8 ≈ 0.26
2. **Effort:**

   Effort (person-months) = 2.4 * (Size in function points)^1.05 * Effort Multiplier

   Effort = 2.4 * (10)^1.05 * 0.26 ≈ 7 person-months
3. **Development Time:** Assuming a team of 4 members:

   Development Time (months) = Effort / Team Size

   Development Time = 7 / 4 = 1.75 months
4. **Determine Project Cost:**

   Project Cost = Development Time * (Average Monthly Cost per Team Member * Team Size)

   Project Cost = 1.75 * ($500 * 4) = $3,500

So, based on the assumptions and using the Basic COCOMO model, the estimated effort for the development of the P2P chatting network application is approximately 7 person-months. The estimated development time is less than 2 months, and the projected project cost is approximately $3,500$.

10. Conclusion

The development of a Peer-to-Peer Multi-User Chatting Application will address the need for a secure and feature-rich communication platform. The proposed features will enhance user experience and provide a versatile chatting environment for both personal and professional use. The successful completion of this project will result in a valuable tool for real-time communication.

Ain Shams University Faculty of Engineering
CSE351: Computer Networks
Under the surveillance of Professor Ayman Bahaa.

Peer-to-Peer Multi-User Chatting Application

Group 7

Kareem Wael Hasan Ahmed	2001151
Malak ahmed yehia sherif	2001350
hussien ahmad abdelgelil mohammed khalifa	2000459
Mahmoud Talaat El-sayed Rezk	2001366

Abstract

In an era of digital connectivity, a novel peer-to-peer multi-user chat application emerges, built upon the foundation of Python and socket programming. It is a p2p chat application that uses centralized index approach. Through the creation and management of chat rooms, users can engage in both individual and group conversations. The application safeguards users with a secure authentication system, ensuring privacy and trust within the virtual environment. Beyond basic text-based messaging, the platform offers functionalities like text formatting and hyperlink sharing, enhancing the overall communication experience. By prioritizing user-friendliness through a simple command-line interface and visually distinct color-coded messages, the application caters to diverse user preferences. Moreover, robust error handling and automatic network reconnection ensure that user experience remains seamless and uninterrupted. This project promises a significant contribution to the realm of online communication, offering a secure, reliable, and engaging platform for users to connect and share their experiences.

Keywords: peer-to-peer, Python, socket programming, centralized index approach, authentication system , text-based messaging, network reconnection .

Table of Contents

Table Of Figures

Design Document

1. Security measurements

As a network application, implementing robust security measurement is crucial for the success of the software as a product, this is achieved using 3 measurements:

1. Hashing the password transferring over the network:

   Authentication is a crucial aspect of the peer-to-peer chatting application to ensure secure and authorized access. The system employs the **SHA-256 Cryptographic Hash Algorithm** authentication method:

   Client modules will communicate with the authentication utility module that is responsible for hashing the password to ensure secure transfer of sensitive information over the network.

   When the client registers, his password is hashed using SHA-256 algorithm and sent hashed over the network to the registry server, it then validates that the user have a unique username then create the user record in the database, when the user logs in, the password they type is yet again hashed using the same technique and sent hashed over the network, the registry server compares the hashed password sent using the login message with the hashed password stored in the database which should be identical in case of a correct password.

   The advantage of using a hashing technique over encryption is that hashing produces a result that cannot be decrypted back to the original message, making the transfer of hashed message secure even if the packet got sniffed.

   refer to the authentication sequence in Figure 6: User Registration (including authentication)

2. Storing the password hashed in the Database:

   This prevents DBA and developers from accessing users' sensitive information.

3. Using SSL/TLS to securely transfer messages over the network:

   SSL (Secure Sockets Layer) encryption, and its more modern and secure replacement, TLS (Transport Layer Security) encryption, protect data (not only passwords) sent over the internet or a computer network.

2. System Components

The system architecture defines the structure and organization of the peer-to-peer chatting application. It includes components, their interactions, and the overall design of the system.

2.1 Components

2.1.1 Peer Client

- **Description:** Represents an individual user of the application.
- **Responsibilities:** Manages user interface, sending messages, and interactions within the application.
- **Components:**
  - User Interface
  - Communication Module
  - Peer Client Logic

2.1.2 Peer Server

- **Description:** Replaces the server in the traditional Client-Server architecture
- **Responsibilities:** Receiving messages and sending notifications.
- **Components:**
  - Client handler
  - Chat room hosting

2.1.3 Registry Server

- **Description:** Centralized server managing user accounts, online status, and user searches.
- **Responsibilities:** Handles user authentication, search operations, and online status management.
- **Components:**
  - Registry Logic
  - Database

2.1.4 Authentication utility

- **Description:** Provides the functionality of hashing passwords
- **Responsibilities:** Hashing Passwords.
- **Components:**
  - Hashing module

2.1.5 Database

- **Description:** Stores user data, and other relevant information.
- **Responsibilities:** Manages user accounts and chat-related data storage.
- **Components:**
  - Accounts Data
  - Chat rooms data

2.2 Interactions

- **Peer-to-Peer Communication:** Peers communicate directly for real-time chat.
- **Registry-Database Interaction:** Registry interacts with the database for user account management.
- **Peer-Registry Interaction:** Peers interact with the registry for authentication and user-related operations.
- **Peer-Chat Server Interaction:** Peers communicate with the chat server for chat room management and messaging.

2.3 Error Handling

Robust error handling mechanisms are implemented throughout the application to ensure graceful handling of unexpected scenarios. Each component incorporates error detection, reporting, and recovery mechanisms to enhance the application's reliability.

3. System Architecture and Design diagrams

This section presents visual representations of the system architecture through various diagrams, aiding in the understanding of the architecture and interactions.

4.1 Component Diagram

The UML component diagram provides a visual representation of the various software and hardware components in the system and their relations

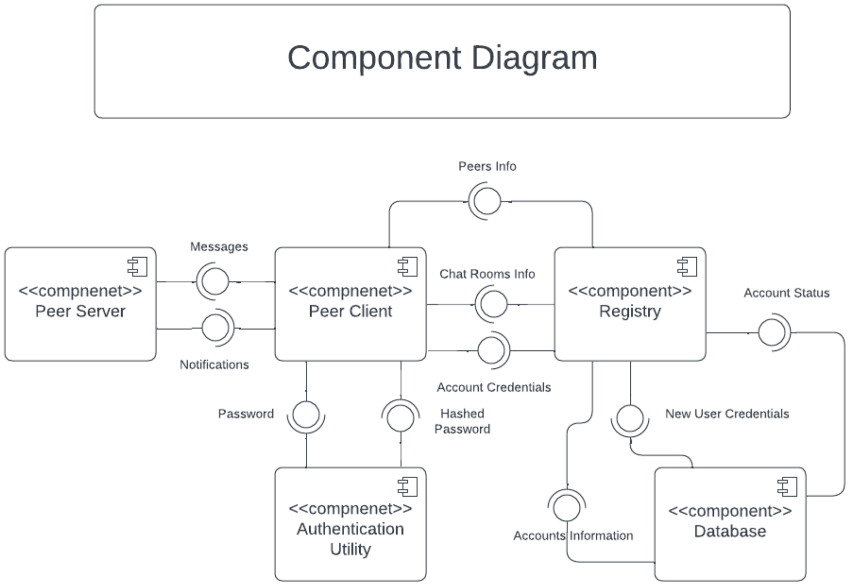

Figure 1: Component Diagram

4.2 Layered Architecture

The system is designed in layered architecture where every layer communicates only with the layer above/below it, this helps with the scalability of the application and makes it easier for modification and maintenance.

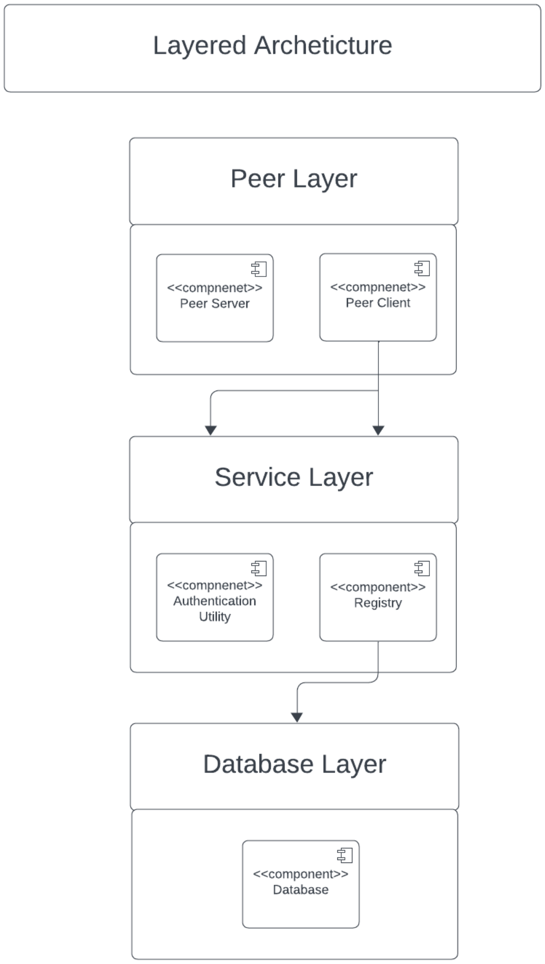

Figure 2: Layered architecture

The topmost layer represents the core components which the user interacts with as a peer in the application, the middle layer represents the service components which implements authentication at the client side and application organization at the server side, the bottom layer represents the interface to the database.

4.3 Sequence Diagrams

This sequence diagram details the sequence of interactions during the search operation, demonstrating how components collaborate to fulfill this specific functionality.

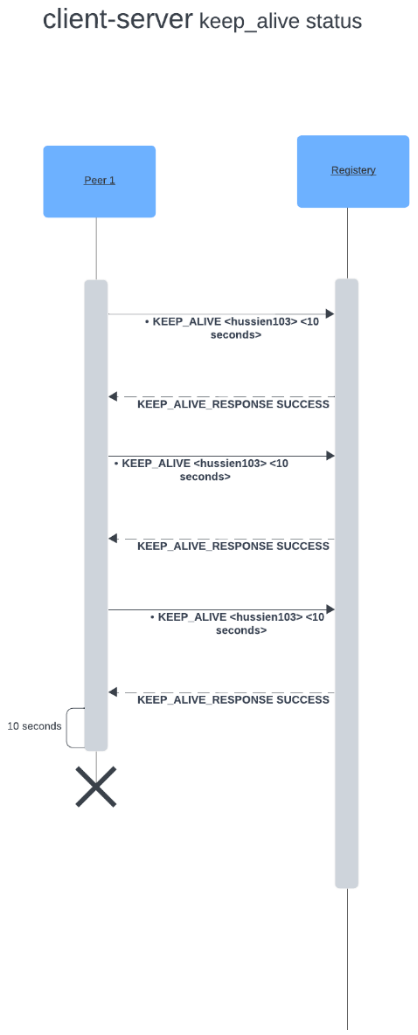
client-server keep_alive status
Peer 1
Registery
• KEEP_ALIVE <hussien103> <10 seconds>
KEEP_ALIVE_RESPONSE SUCCESS
• KEEP_ALIVE <hussien103> <10 seconds>
KEEP_ALIVE_RESPONSE SUCCESS
• KEEP_ALIVE <hussien103> <10 seconds>
KEEP_ALIVE_RESPONSE SUCCESS
10 seconds

Figure 3: Client Server Keep Alive Status

With this mechanism the registry can detect when a user becomes offline for any reason, as the user have to send a message every x second to confirm that they are still there.

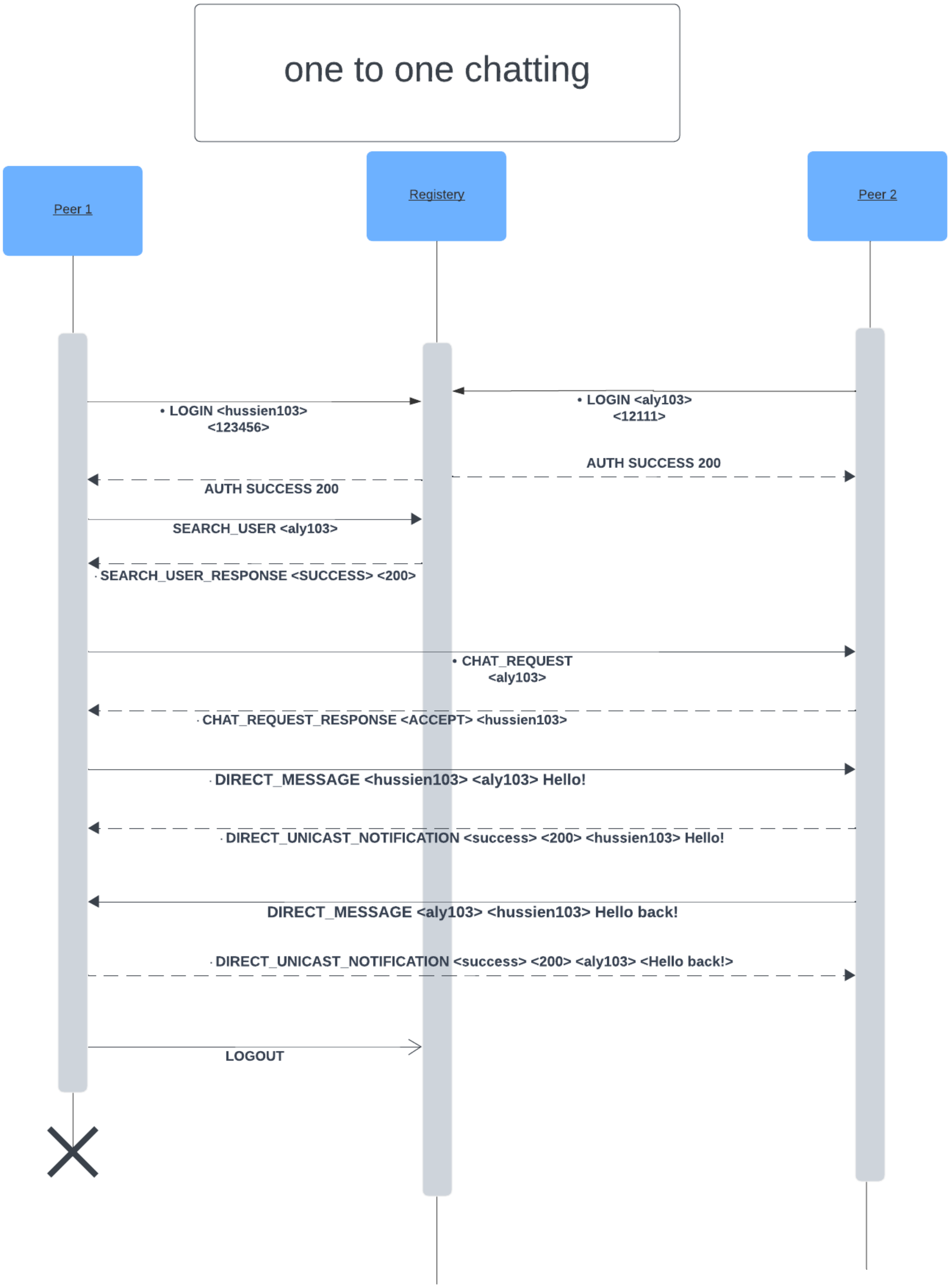
one to one chatting
Peer 1
Registery
Peer 2
• LOGIN <aly103>
<12111>
• LOGIN <hussien103>
<123456>
AUTH SUCCESS 200
AUTH SUCCESS 200
SEARCH_USER <aly103>
SEARCH_USER_RESPONSE <SUCCESS> <200>
• CHAT_REQUEST
<aly103>
CHAT_REQUEST_RESPONSE <ACCEPT> <hussien103>
DIRECT_MESSAGE <hussien103> <aly103> Hello!
DIRECT_UNICAST_NOTIFICATION <success> <200> <hussien103> Hello!
DIRECT_MESSAGE <aly103> <hussien103> Hello back!
DIRECT_UNICAST_NOTIFICATION <success> <200> <aly103> <Hello back!>
LOGOUT

Figure 4:One to one Chatting

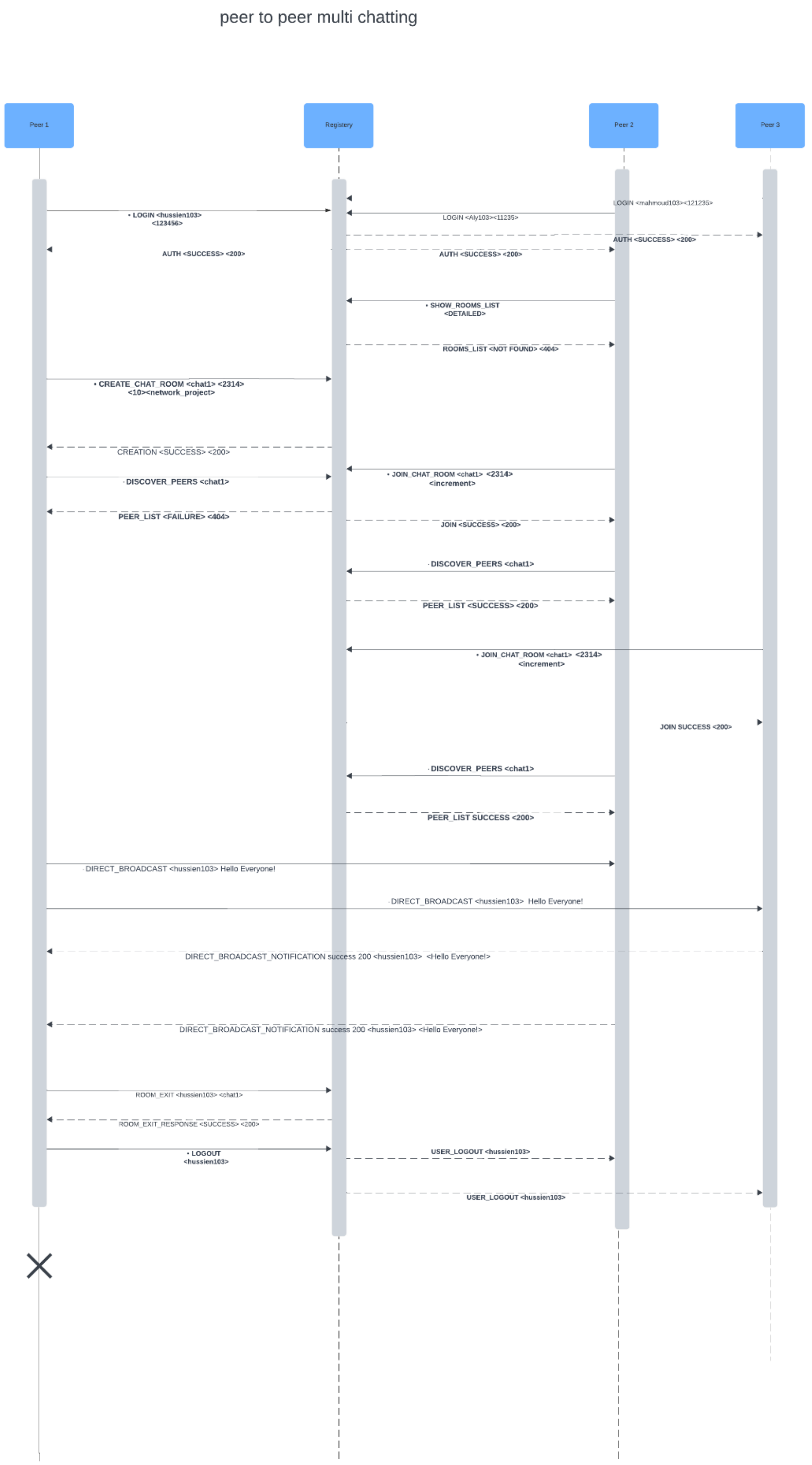
peer to peer multi chatting
Peer 1
Registery
Peer 2
Peer 3
LOGIN <mahmoud103><121235>
• LOGIN <hussien103> <123456>
LOGIN <Aly103><11235>
AUTH <SUCCESS> <200>
AUTH <SUCCESS> <200>
AUTH <SUCCESS> <200>
• SHOW_ROOMS_LIST <DETAILED>
ROOMS_LIST <NOT FOUND> <404>
• CREATE_CHAT_ROOM <chat1> <2314> <10><network_project>
CREATION <SUCCESS> <200>
• JOIN_CHAT_ROOM <chat1> <2314> <increment>
DISCOVER_PEERS <chat1>
PEER_LIST <FAILURE> <404>
JOIN <SUCCESS> <200>
DISCOVER_PEERS <chat1>
PEER_LIST <SUCCESS> <200>
• JOIN_CHAT_ROOM <chat1> <2314> <increment>
JOIN SUCCESS <200>
DISCOVER_PEERS <chat1>
PEER_LIST SUCCESS <200>
DIRECT_BROADCAST <hussien103> Hello Everyone!
DIRECT_BROADCAST <hussien103> Hello Everyone!
DIRECT_BROADCAST_NOTIFICATION success 200 <hussien103> <Hello Everyone!>
DIRECT_BROADCAST_NOTIFICATION success 200 <hussien103> <Hello Everyone!>
ROOM_EXIT <hussien103> <chat1>
ROOM_EXIT_RESPONSE <SUCCESS> <200>
• LOGOUT <hussien103>
USER_LOGOUT <hussien103>
USER_LOGOUT <hussien103>

Figure 5:Multi Chatting

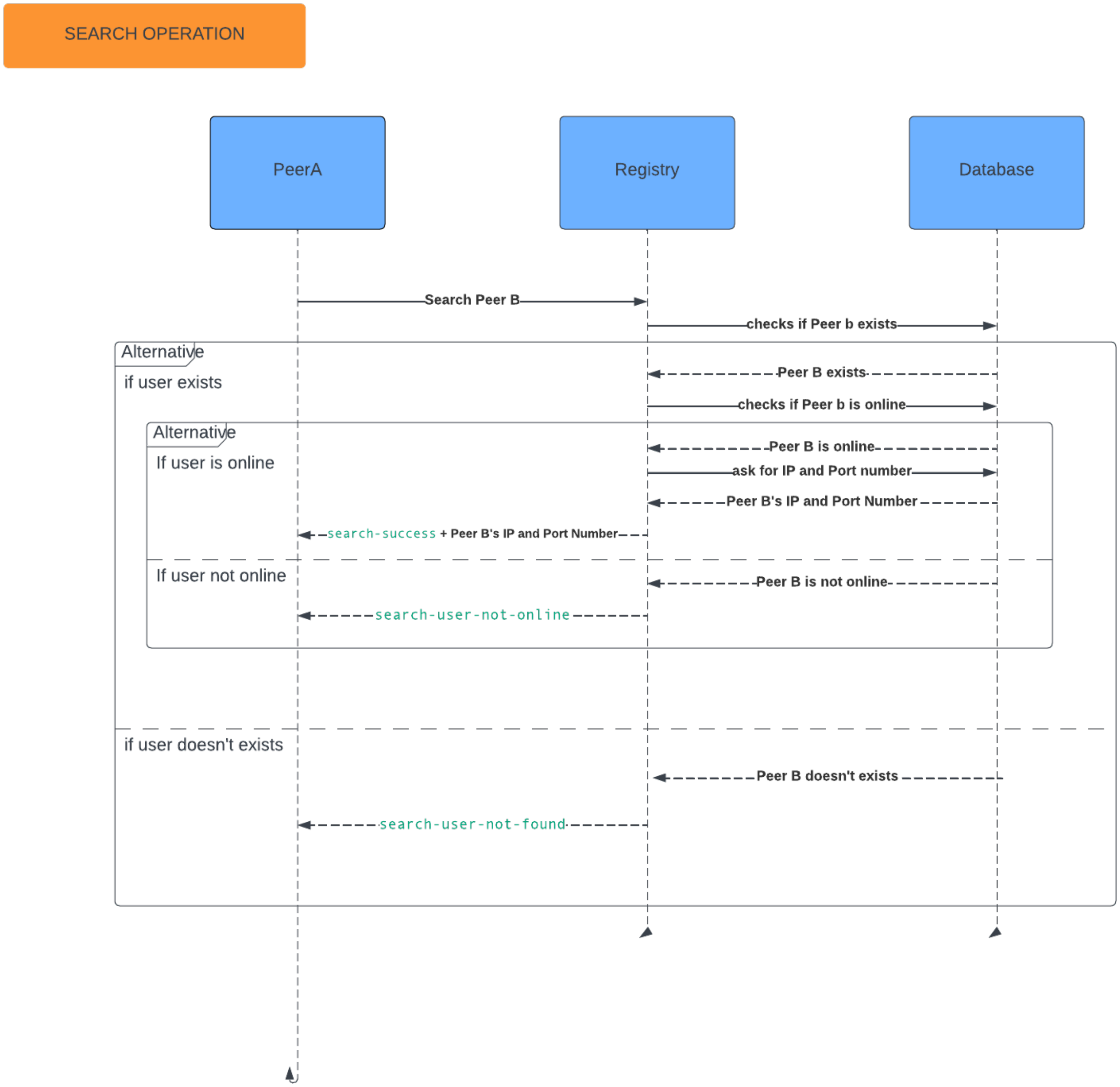

Figure 6: Search Operation

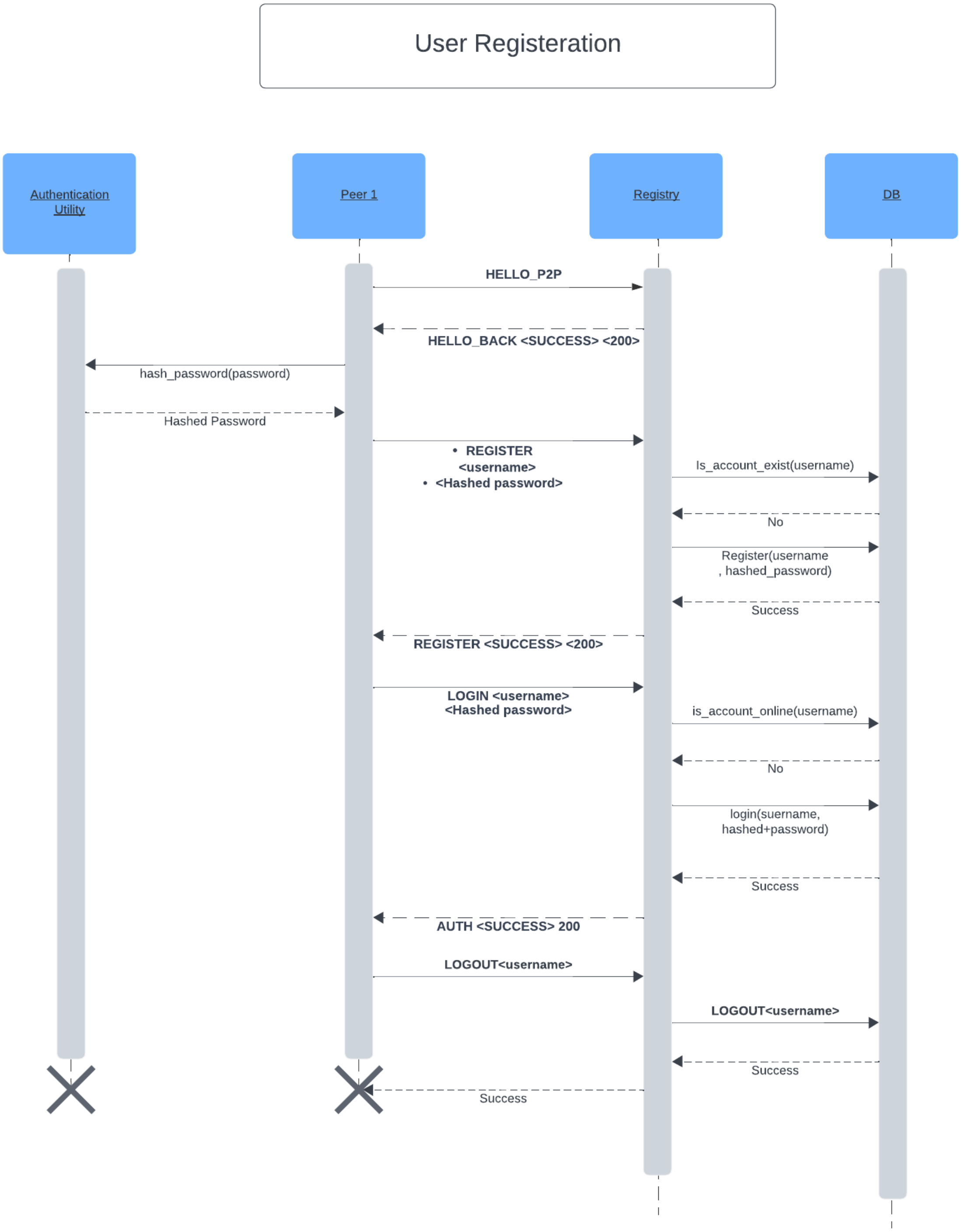

Figure 7: User Registration (including authentication)

4.4 Context Diagram

The context diagram describes the relation between the application and external sources/sinks, in the p2p chatting application the only external source/sink is the user using all functionality of the system.

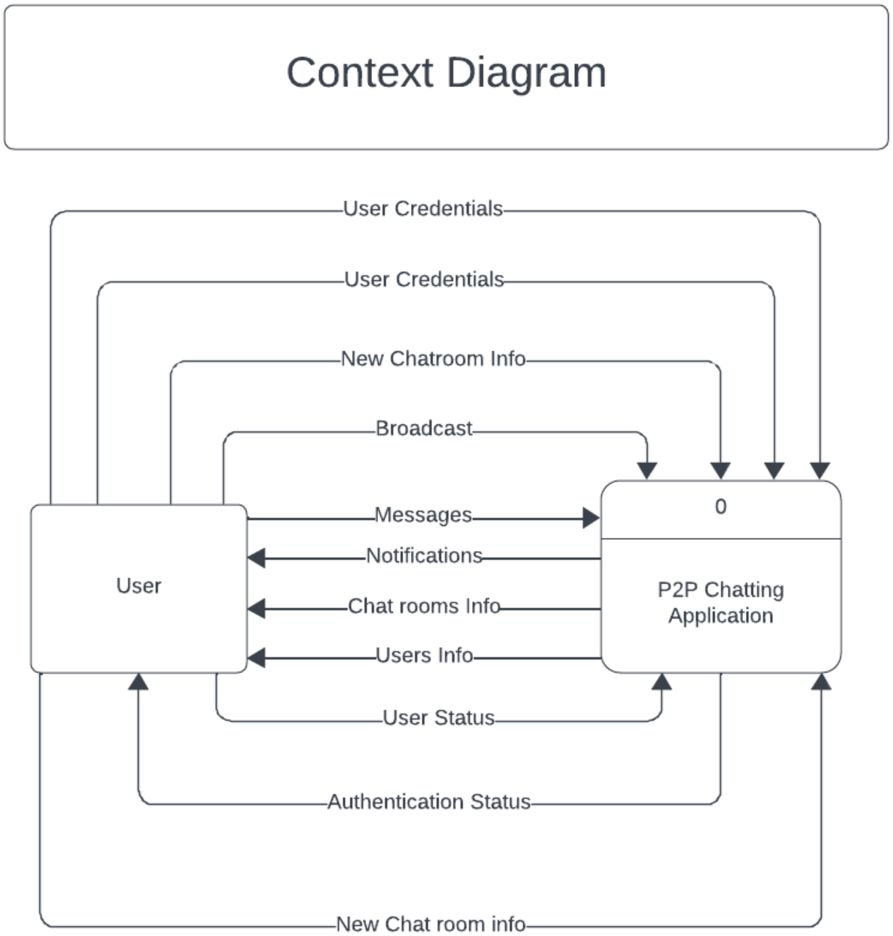

Figure 8: Context Diagram

4.5 Data Flow Diagram

The data flow diagram outlines the flow of information between different components of the system, providing a high-level view of how data moves through the application.

An overview of the main processes and data exchanged between them can be represented with Level 0 DFD:

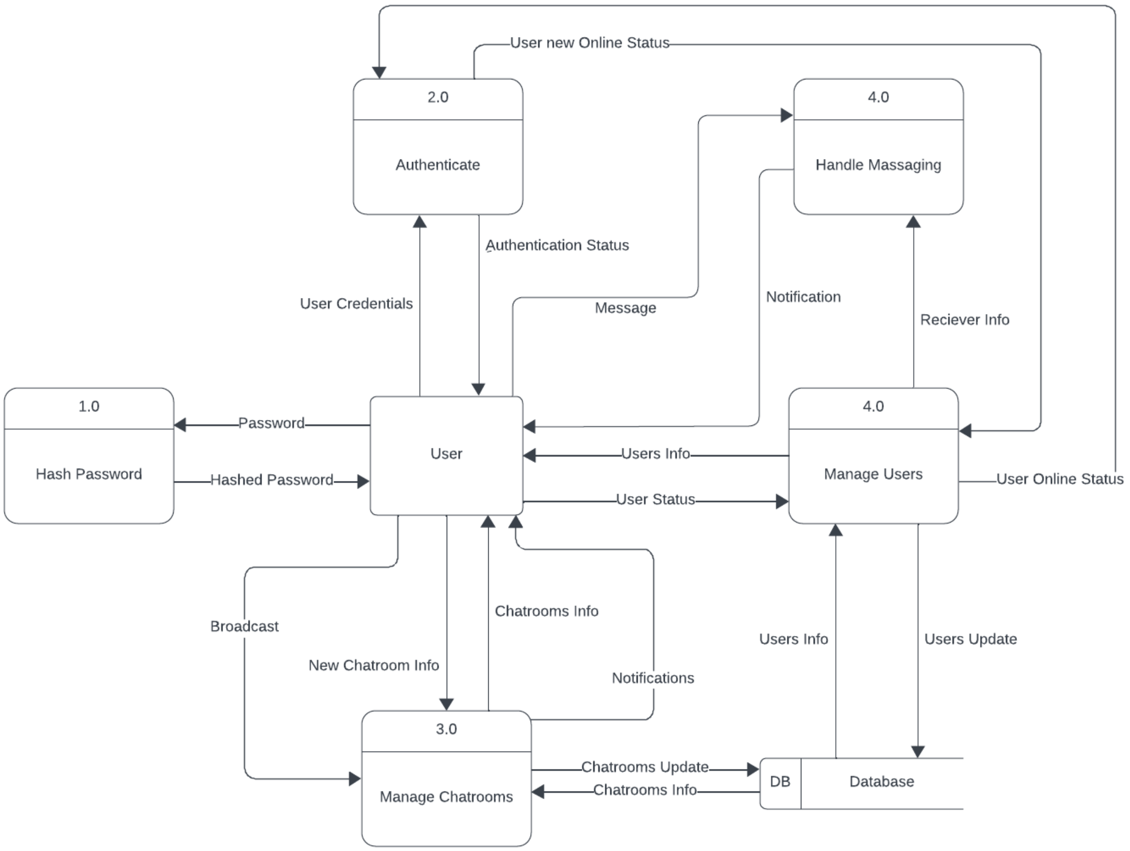

Figure 9: Level 0 DFD

4. Communication Protocols

I. Hello Message

- **Peer to Registry:**
  - TCP based Message: **HELLO_P2P**
  - Purpose: Notifies the registry about the peer's presence.
  - **Example request : HELLO_P2P**

[header lines]

command : HELLO_P2P

[data payload]

(empty)

- **Registry to Peer:**
  - TCP based Response: **HELLO_BACK <SUCCESS> <200> / HELLO <FAILURE> <404>**
  - Purpose: Confirms successful connection with the registry

.[header Lines]

Status line : HELLO_BACK

Status phrase : SUCCESS/FAILURE

Status code : 200 / 404

[data payload]

(empty)

II. User Registration

- **Client to Registry:**

- TCP based Message: **REGISTER <username> <password>**
- Purpose: Initiates the registration process with the registry.
- Example Request: **REGISTER <hussien> <123456>**

[Header Lines]

Command: REGISTER

Username: hussien

Password: 123456

[Payload]

(empty)

- **Registry to Client:**
  - TCP based Response: **REGISTER <SUCCESS> <200> or REGISTER <FAILURE> <404> or REGISTER <EXIST> <300>**
  - Purpose: Notifies the client about the success or failure of the registration.
  - Example Response:

[Header Lines]

Status Line: REGISTER

Status Phrase: SUCCESS/FAILURE/EXIST

Status Code: 200 / 404/300

[Payload]

(empty)

III. User Authentication:

- **Client to Registry:**
  - TCP based Message: **LOGIN <username> <hashed password>**
  - Purpose: Initiates the authentication process with the registry.
  - **Example request : login <hussien> <123456>**

[Header Lines]

Command: LOGIN

Username: hussien

Hashed Password: 123456

[Payload]

(empty)

- **Registry to Client:**
  - TCP based Response: **AUTH <SUCCESS> <200>** or **AUTH <FAILURE> <404> or AUTH <ONLINE> <300>**
  - Purpose: Notifies the client about the success or failure of the authentication.
  - **Example response:**

[Header Lines]

Status Line : AUTH

Status Phrase : SUCCESS/FAILURE/ONLINE

Status Code : 200 / 404 / 300

[Payload]

(empty)

IV. Creating Chat Room:

Client to Registry:

- TCP based Message: **CREATE_CHAT_ROOM <room_name> <room_password> <room-capacity><room_description>**
- Purpose: Requests the registry to create a new chat room with the specified name.
- **Example request :** CREATE_CHAT_ROOM <chat1> <123456> <10> <network_project>

[Header Lines]

command : CREATE_CHAT_ROOM

room_name : chat1

room_password:123456

room_capacity: 10

room_description : network_project

[Data Payload]

(empty)

- **Registry to Client:**
  - TCP based Response: CREATION <SUCCESS> or CREATION <FAILURE>
  - Purpose: Notifies the client about the success or failure of creating the chat room.
  - **Example response:** CREATION <SUCCESS> <200>

[Header Lines]

Status Line : CREATION

Status phrase : SUCCESS / FAILURE

Status code : 200/404

[Data payload]

(empty)

V. Joining Chat Room:

- **Client to Registry:**
  - TCP based Message: **JOIN_CHAT_ROOM <room_name> <room_password> <room_members_count>**
  - Purpose: Requests to join a specific chat room.
  - **Example request:** JOIN_CHAT_ROOM <chat1> <123455> <increment>

[Header Lines]

Command : JOIN_CHAT_ROOM

room_name : chat1

room_password : 123455

room_members_count : increment

[data payload]

(empty)

- **Registry to Client:**
  - TCP based Response: **JOIN <SUCCESS> <200>** or **JOIN <FAILURE> <404>**
  - Purpose: Notifies the client about the success or failure of joining the chat room.
  - Example Response: JOIN <FAILURE> <404>

[Header Lines]

Command : JOIN

Status phrase : FAILURE

Status code : 404

[data payload]

(empty)

VI. Leaving Chat Room:

- **Client to Registry:**
  - TCP based Message: **ROOM_EXIT <user_name><room_name>**
  - Purpose: Requests to leave a specific chat room.
  - Example Request: **ROOM_EXIT <hussien103> <chat1>**

[Header Lines]

Command : ROOM_EXIT

User_name : hussien103

Room_name : chat1

[data payload]

(empty)

- **Registry to Client:**
  - TCP based Response: **ROOM_EXIT_RESPONSE <SUCCESS> <200> or ROOM_EXIT_RESPONSE <FAILURE> <404>**
  - Purpose: Notifies the client about the success or failure of leaving the chat room.
  - Example Response: **ROOM_EXIT_RESPONSE <SUCCESS> <200>**

[Header Lines]

Status line: ROOM_EXIT_RESPONSE

Status Phrase: SUCCESS

Status Code: 200

[Payload]

(empty)

VII. Showing list of rooms :

- **Client to Registry:**
  - TCP based Message: **SHOW_ROOMS_LIST <type>**
  - Purpose: Show list of rooms with details or not.
  - **Example Request : SHOW_ROOMS_LIST <DETAILED>**

[Header Lines]

Command : SHOW_ROOMS_LIST

Type : DETAILED/PARTIAL

[data payload]

(empty)

- **Registry to Client:**
  - TCP based Response : **ROOMS_LIST <status_phrase> <status_code> + data**
  - Purpose: respond with list of rooms if found and not found response if not found.
  - Example response : ROOMS_LIST FOUND <200>

[Header Lines]

Status Line : ROOMS_LIST

Status Phrase : FOUND

Status Code: 200

[data payload]

<room1_name> <room1_description> <room1_members_count> <room1_capacity> ,

<room2_name> <room2_description> <room2_members_count> <room2_capacity> , …

VIII. Search User:

- **Client to Server:**
    - TCP based Message: **SEARCH_USER <username>**
    - Purpose : find a specified user
    - Example Request: **SEARCH_USER <aly103>**

[Header Lines]

Command: SEARCH_USER

Username: aly103

[data Payload]

(empty)

- **Server to Client:**
    - TCP based Response: **SEARCH_USER_RESPONSE <SUCCESS> <200> + data / SEARCH_USER_RESPONSE <NOT_ONLINE> <300> / SEARCH_USER_RESPONSE <NOT_FOUND ><404>**
    - Purpose : response with success or failure
    - Example Response: **SEARCH_USER_RESPONSE <SUCCESS> <200>**

[Header Lines]

Status Line : SEARCH_USER_RESPONSE

Status Phrase: SUCCESS/NOT_ONLINE/NOT_FOUND

Status Code : 200/300/404

[Data Payload]

<peer_username><peer_port><peer_IP>

IX. Start a Chat :

- **Peer to Peer(sender to reciever):**
    - TCP based Message: **CHAT_REQUEST < recipient_username>**
    - Purpose : requesting the reciever to start a chat
    - Example Request: **CHAT_REQUEST <aly103>**

[Header Lines]

Command: CHAT_REQUEST

recipient_Username: ahmed

[Payload]

(empty)

Peer to peer(recipient to sender):

- TCP based response: **CHAT_REQUEST_RESPONSE <ACCEPT> <200> < sender_username > / CHAT_REQUEST_RESPONSE <REJECT> <404>**
- Purpose : accepting or rejecting the chat request
- Example Request: **CHAT_REQUEST_RESPONSE <ACCEPT> <hussien103>**

[Header Lines]

Status line: CHAT_REQUEST_RESPONSE

Status phrase : ACCEPT / REJECT

Status code : 200 / 404

sender_username: hussien103

[data Payload]

(empty)

X. Online peer discovery

- **Peer to Registry:**
  - TCP based Message: **DISCOVER_PEERS <type>**
  - Purpose: Requests a list of online peers from the registry.
  - **Example request : DISCOVER_PEERS <DETAILED>**

[header lines]

Command : DISCOVER_PEERS

[data payload]

(empty)

- **Registry to Peer:**
  - TCP based Response: **PEER_LIST <SUCCESS>/<FAILURE> <200>/<404> + peer1, peer2 ...**
  - Purpose: Provides a list of online peers.

[header lines]

Status line : PEER_LIST

Status phrase : success

Status code : 404

[data payload]

<peer1_username> <peer1_roomname> <peer1_port><peer1_ip>,

<peer2_username> <peer2_roomname> <peer2_port><peer2_ip>, …

XI. Direct message in one to one chat

- **Peer to Peer (Direct Message):**
  - Direct TCP Message: **DIRECT_MESSAGE <sender_username> <recipient_username> + message**
  - Purpose: Allows users to send messages in one to one chat.
  - **Example Message: DIRECT_MESSAGE <hussien103> <aly103> Hello!**

[Header Lines]

Command: DIRECT_MESSAGE

sender_username: hussien103

recipient_username: aly103

[Data Payload]

Hello!

Peer to Sender (unicast notification) :

- TCP based Notification: DIRECT_UNICAST_NOTIFICATION <SUCCESS>/<FAILURE> <200>/<404> <sender_username> <message_content>
- Purpose: Notifies sender about receiving a unicast message or not.
- **.Example Notification: DIRECT_UNICAST_NOTIFICATION <SUCCESS> <200> <hussien103> <Hello!>**
- **[Header Lines]**

Status Line: DIRECT_BROADCAST_NOTIFICATION

Status phrase : SUCCESS/FAILURE

Status code : 200/404

Sender_username : hussien103

Message_content : Hello!

[Payload]

(empty)

XII. User Logout:

- **Client to Registry:**
  - TCP based Message: **LOGOUT**
  - Purpose: Informs the registry that the user is logging out.

[header links]

Command : LOGOUT

[data payload]

(empty)

- **Registry to peer (in the same room):**
  - TCP based Broadcast: **USER_LOGOUT <user_name>**
  - Purpose: Notifies other users in the same chat room about the user's logout.
  - **Example message : USER_LOUGOUT <hussien>**

[Header Links]

status line : USER_LOGOUT

user_name : hussien

[data payload]

(empty)

XIII. Keep-Alive Message:

- **Peer to Registry:**
  - UDP based Message: **KEEP_ALIVE <user_name> <timeout>**
  - Purpose: Periodically sent by peers to maintain their online status.

[header lines]

command : KEEP_ALIVE

user_name : hussien103

timeout : 10

[data payload]

(empty)

- **Registry to Peer:**
  - UDP based Response: **KEEP_ALIVE_RESPONSE <SUCCESS> /< FAILURE> <200>/<404>**
  - Purpose: Confirms the receipt of the keep-alive message.

[header lines]

Status line : KEEP_ALIVE_RESPONSE

Status phrase : SUCCESS / FAILURE

Status code : 200/404

[data payload]

(empty)

XIV. Timeout Notification:

- **Registry to Peer:**
  - UDP based Notification: **TIMEOUT <username>**
  - Purpose: Notifies the peer about a timeout (e.g., no keep-alive received).

[header lines]

notification line : TIMEOUT

user_name : hussien103

[data payload]

(empty)

XV. Room Peers Discovery Protocol:

Peer to Registry:

- TCP based Message: **DISCOVER_PEERS <current_room_name>**
- Purpose: Requests a list of online peers in a specific chat room.
- **Example Request : DISCOVER_PEERS <chat1>**

[Header Lines]

Command : DISCOVER_PEERS

Current_room_name : chat1

[data payload]

(empty)

- **Registry to Peer (Discovery Response):**
  - TCP based Response: **PEER_LIST <SUCCESS>/<FAILURE> <200>/<404> <room_name> + data**
  - Purpose: Provides a list of online peers in the specified chat room if it exists or has members.
  - **Example Response:**

[header lines]

Status line : PEER_LIST

Status phrase: SUCCESS/FAILURE

Status code : 200/404

[data payload]

<room_name> <peer1><peer2>

XVI. Direct Peer Messaging:

- **Peer to Peers:**
  - TCP based Message: DIRECT_BROADCAST <sender_username> + message
  - Purpose: Allows a peer to broadcast a message to all other peers in the same chat room.
  - **Example Request: DIRECT_BROADCAST <hussien103> Hello Everyone!**

[Header Lines]

Command: DIRECT_BROADCAST

Sender_username : hussien103

[data Payload]

Hello, everyone!

- **Peers to Sender (broadcast notification) :**
  - TCP based Notification: DIRECT_BROADCAST_NOTIFICATION <SUCCESS>/<FAILURE> <200>/<404><sender_username> <message_content>
  - Purpose: Notifies the sender about a broadcast message
  - **.Example Notification: DIRECT_BROADCAST_NOTIFICATION <SUCCESS> <200> <hussien103> <Hello Everyone!>**

[Header Lines]

Status Line: DIRECT_BROADCAST_NOTIFICATION

Status phrase : success/failure

Status code : 200/404

Sender_username : hussien103

Message_content : Hello Everyone!

[Payload]

(empty)

5. System Scalability

The application is designed with scalability in mind to accommodate a growing user base. The system architecture allows for the seamless addition of resources, ensuring optimal performance even as the number of users increases.

The layered system architecture discussed in the previous section facilitates system scalability as managing increasing load can be easily done by updating the bottlenecked layer without affecting the other layers or the interfaces between them.