**Home Security System with LIFI**

Intro to Embedded Systems – CSE221

**Submitted to**

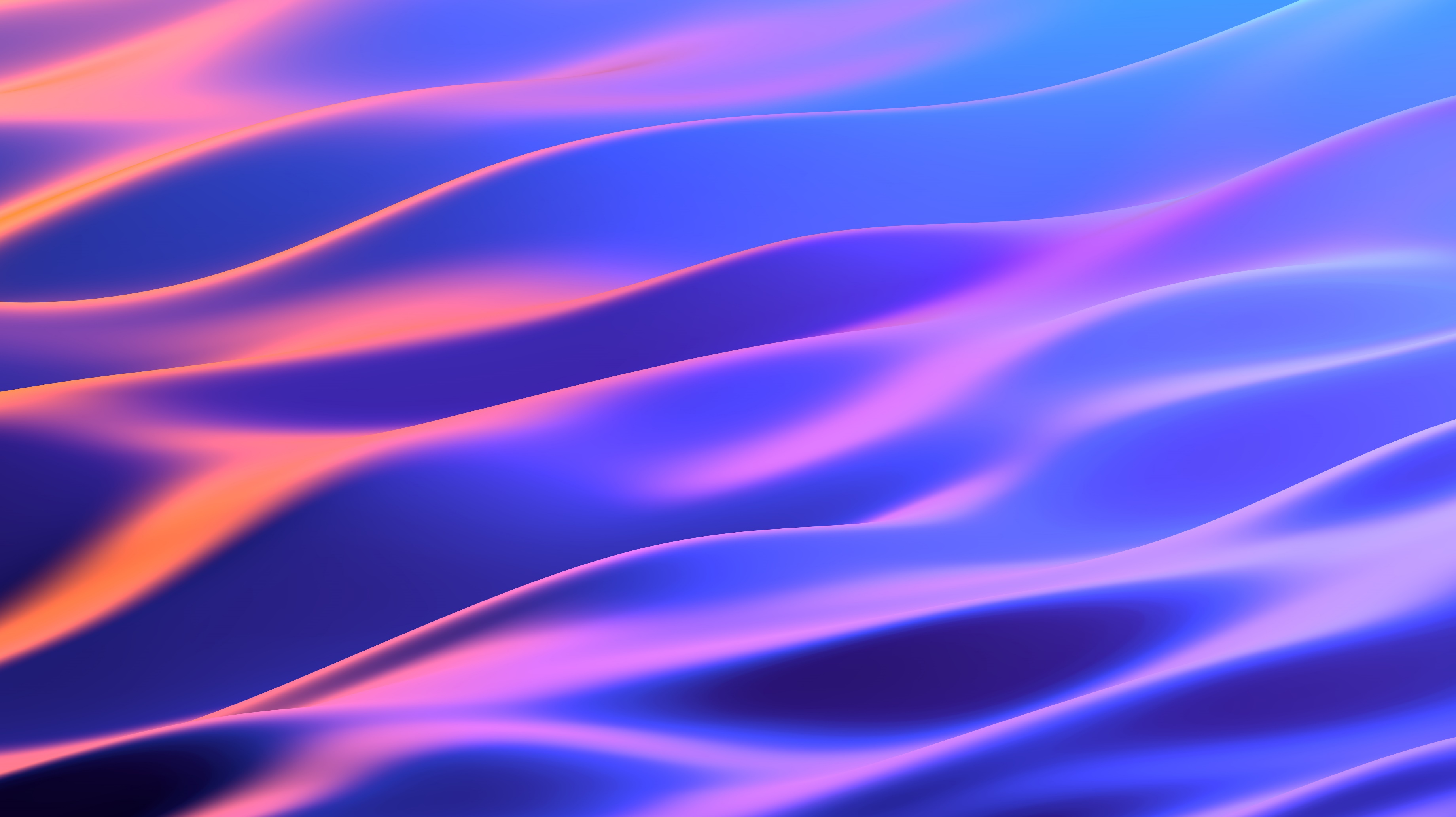
Prof. Sherif Hammad

Eng. Mohamed Tarek

**Group: 20**

|  |  |
| --- | --- |
| Ahmed Wael Ibrahim Mohamed | 20P3343 |
| Donia Sameh Farouk Mohamed | 20P3424 |
| Joliana Emad Kamal Naguib | 20P3292 |
| Mohamed Amr El-Mallah | 20P3485 |





Contents

[Executive Summary 5](#_Toc152974062)

[Focus 5](#_Toc152974063)

[Project Overview 6](#_Toc152974064)

[Hardware and Software Requirements 6](#_Toc152974065)

[Operating System 6](#_Toc152974066)

[Programming Language 6](#_Toc152974067)

[Operating System 6](#_Toc152974068)

[Additional Software Products 6](#_Toc152974069)

[Project Scope 7](#_Toc152974070)

[Project Objectives 7](#_Toc152974071)

[Phases of Implementation 7](#_Toc152974072)

[Constraints and Assumptions 7](#_Toc152974073)

[Key Functionalities 7](#_Toc152974074)

[Overview 10](#_Toc152974075)

[User Stories 11](#_Toc152974079)

[Cost Analysis 13](#_Toc152974080)

[User Authentication 16](#_Toc152974081)

[Authentication Mechanisms for P2P application 16](#_Toc152974082)

[Types of User authentication for P2P Multi-User Chatting Application 18](#_Toc152974083)

[Key Considerations for Selection 19](#_Toc152974084)

[System Architecture 20](#_Toc152974085)

[Client-Server Model 20](#_Toc152974086)

[Definition and characteristics 20](#_Toc152974087)

[Components of Client-Server Architecture 20](#_Toc152974088)

[Peer-to-Peer Model 21](#_Toc152974089)

[Definition and Characteristics 21](#_Toc152974090)

[Components of P2P Architecture 22](#_Toc152974091)

[Software Components 24](#_Toc152974092)

[Component Diagram 24](#_Toc152974093)

[Client-Peer Component 24](#_Toc152974094)

[Chat Room Component 25](#_Toc152974095)

[Database Component 25](#_Toc152974096)

[Server-Peer Component 26](#_Toc152974097)

[Sequence Diagrams 27](#_Toc152974098)

[Login 27](#_Toc152974099)

[Logout 28](#_Toc152974100)

[Signup 29](#_Toc152974101)

[One-to-One Chatting 30](#_Toc152974102)

[Group Chatting 31](#_Toc152974103)

[Data Flow Diagram 32](#_Toc152974104)

[Communication Protocols 33](#_Toc152974105)

[Client-Server Protocol 33](#_Toc152974106)

[Peer-to-Peer Protocol 34](#_Toc152974107)

[Message Formats 39](#_Toc152974108)

Project Proposal

# Executive Summary

The modern era, shaped by globalization and technological advancements, has significantly altered human life, notably through the emergence of social networking. As an integral facet of contemporary existence, this evolution necessitates innovative technologies and software to facilitate seamless human interaction. Our project responds to this need by developing a sophisticated peer-to-peer messaging application, offering users worldwide the ability to communicate effortlessly via a user-friendly command-line interface.

Leveraging the capabilities of Python and Socket APIs alongside the Computer Networks protocol stack, our application seeks to enable global communication. Users will access private and group chat functionalities within chat rooms, empowering them to exchange messages seamlessly. Furthermore, the application offers the added feature of sending formatted text, encompassing options such as Bold, Italic, and more.

This project's primary goal is to create an intuitive, robust platform that harnesses contemporary technologies to foster enhanced global communication while prioritizing user convenience and versatility in communication styles.

# Focus

After a thorough but efficient preparation period, our team will create the Peer-to-Peer Messaging Application.

We will build this with a focus on **Ability to authenticate users**, **Ability to send/receive messages** both in private and in chat rooms and **Ability to send formatted messages** through a user-friendly command-line interface.

The resulting software will achieve:

* Sophisticated yet user-friendly user interface.
* Secure and reliable connection between clients.
* Secure and sustainable user authentication protocols.
* Efficient User communication protocols.
* Error handling and resilience.

# Project Overview

We are committed to developing and delivering a product that fulfils the provided requirement specifications in the best possible methods. As such, we have put together a plan to develop the Peer-to-Peer Messaging Application. The software solution will fulfil the requirements mentioned in the **Key Functionalities** section.

## Hardware and Software Requirements

This section will discuss the operational requirements for developing and operating the finished software solution.

|  |  |
| --- | --- |
| Operating System  * OS that supports running Python Scripts * Windows – MacOs - Linux | Programming Language  * Python Programming 3.9 or later * Soket Programming APIs |
| Operating System  * Desktop Computers * Personal Computers (Laptops) * Computing devices supporting Python Scripts | Additional Software Products  * Any further products will be discussed later when needed. |

# Project Scope

## Project Objectives

The project aims to develop a robust Peer-to-Peer Multi-User Chatting Application using Python programming language and socket programming APIs. The project focuses on text-based communication while utilizing the peer-to-peer network topology, TCP and UDP protocols. The project progresses through three technical phases that will be mentioned in a later section.

## Phases of Implementation

|  |
| --- |
| **Phase (1) – Basic Client-Server Setup**  Implement client-side and server-side software capable of communicating together using Python and Socket API. |
| **Phase (2) – Peer-to-Peer Architecture and Chat Rooms**  Integrate peer-to-peer architecture and implement the ability to create chat rooms. |
| **Phase (3) – One-to-One Chatting and Protocol Optimization**  Implement one-to-one chatting feature between the user and another selected user. In addition to protocol optimizations and performance enhancements. |

## Constraints and Assumptions

* Both server and client software run on windows machines with python packages and interpreter installed.
* Both server and client software will run on a pre-specified port number.
* Users will interact through the system using Command-line interface (CLI).

## Key Functionalities

1. **User Authentication**

Users can authenticate the user’s credentials (username and password).

1. **Chat Rooms**

Users can create chat rooms where that other users can join or join a pre-created chat room.

1. **Group Messaging in Chat Rooms**

Users can send or read messages sent in a chat room. All messages are visible to all the users.

1. **One-to-One Chatting**

Users can participate in a private chat with another user. The messages are visible to the participants only and no other user can see the messages.

1. **Message Formatting**

Users can send formatted messages (i.e., bold, italic).

1. **User Interface**

Users can interact with the software using a user-friendly command-line interface (CLI).

Design Document

# Overview

Project Duration

4 weeks until January 4th, 2024.

Development Team

* Ahmed Wael Ibrahim
* Mohamed Amr
* Donia Sameh
* Joliana Emad

Purpose

The development of a Peer-to-Peer Messaging application serves as a major task submission for the Computer Networks course instructed by Prof. Ayman Bahaa El-Dine and Eng. Noha Wahdan.

Dependencies

Development will depend on the code base provided by prof. Ayman on LMS.

Methodology

The project will adhere to the **Agile Software Development methodology**. Agile is an iterative and collaborative approach to software development, emphasizing adaptability and continuous improvement. It involves breaking the project into manageable iterations or "sprints" to facilitate ongoing development, feedback incorporation, and adjustments. This methodology promotes close teamwork, frequent reassessment of priorities, and adaptation to changes. Its core focus lies in delivering functional software that aligns with evolving requirements while ensuring customer satisfaction through effective communication and teamwork.

# User Stories

1. User Authentication:

* **Requirement:** Users must authenticate with a unique username and password.
* **User Story:** As a user, I want to create an account with a unique username and password.
* **User Story:** As a user, I want to log in using my registered username and password.

2. Basic Client-Server Setup:

* **Requirement:** Implement a basic server application to handle multiple client connections.
* **User Story:** As a user, I want to connect to the server using a client application.
* **User Story:** As a user, I want to see a list of online users.

3. Chat Room Functionality:

* **Requirement:** Users can create and join chat rooms.
* **User Story:** As a user, I want to create a new chat room.
* **User Story:** As a user, I want to join an existing chat room.
* **User Story:** As a user, I want to see a list of available chat rooms.

4. Group Messaging in Chat Rooms:

* **Requirement:** Users can send and receive messages within a chat room.
* **User Story:** As a user, I want to send a message to everyone in the chat room.
* **User Story:** As a user, I want to see messages from other users in the chat room.
* **User Story:** As a user, I want to receive notifications for new messages.

5. One-to-One Chat Functionality:

* **Requirement:** Users can initiate one-to-one chat sessions.
* **User Story:** As a user, I want to send a private message to another user.
* **User Story:** As a user, I want to receive private messages from others.
* **User Story:** As a user, I want to be notified of new private messages.

6. Message Formatting and Features:

* **Requirement:** Support basic text formatting (e.g., bold, italics) in messages.
* **User Story:** As a user, I want to format my messages to emphasize certain words.
* **Requirement:** Users can share hyperlinks in messages.
* **User Story:** As a user, I want to click on a hyperlink shared in a message to open a browser.

7. Error Handling and Resilience:

* **Requirement:** Implement robust error handling for unexpected scenarios.
* **User Story:** As a user, I want to receive meaningful error messages for troubleshooting.
* **Requirement:** Automatically reconnect users in case of a network interruption.

8. User Interface (UI) Enhancements:

* **Requirement:** Develop a command-line interface for simplicity.
* **User Story:** As a user, I want a clean and intuitive command-line interface.
* **Requirement:** Add color-coded messages for better visual distinction.
* **User Story:** As a user, I want to easily identify different types of messages.

9. Documentation:

* **Requirement:** Create user documentation covering installation, configuration, and usage.
* **User Story:** As a user, I want a comprehensive guide to set up and use the application.
* **Requirement:** Technical documentation detailing system architecture, protocols, and codebase structure.

10. Testing:

* **Requirement:** Conduct unit testing for each implemented feature.
* **User Story:** As a developer, I want to ensure each component functions correctly in isolation.
* **Requirement:** Perform integration testing to ensure seamless interactions between different components.
* **Requirement:** Conduct stress testing to evaluate system performance under high loads.

11. Scalability:

* **Requirement:** Design the system to handle an increasing number of users and chat rooms efficiently.
* **User Story:** As a developer, I want to optimize data structures and algorithms for scalability.

# Cost Analysis

Developing a robust Peer-to-Peer Multi-User Chatting Application involves various costs, including:

**Development Costs:**

* **Software Development:**
  + **Resource Time:**
    - Programming (Python) - Analyzing, designing, coding, and testing the application.
    - Network Engineering - Designing and implementing the network architecture.
    - Security Engineering - Implementing secure authentication and communication protocols.
  + **Tools and Frameworks:**
    - Python development environment (IDE)
    - Specific libraries for networking, cryptography, and UI/UX
    - Cloud-based development platform (optional)
  + **Testing and Quality Assurance:**
    - Unit testing, integration testing, and user acceptance testing
    - Automated testing tools
* **Documentation:**
  + Creating user manuals, API documentation, and internal technical documentation

**Infrastructure Costs:**

* **Deployment:**
  + Cloud-based server (optional)
  + Domain name and SSL certificate
  + Load balancer (optional)
* **Hosting:**
  + Monthly or annual fees for cloud server or other hosting services
  + Bandwidth costs depending on user activity

**Maintenance Costs:**

* Bug Fixes: Addressing issues reported by users
* Feature Enhancements: Implementing new features and functionality
* Security Updates: Maintaining security patches and updates for libraries and frameworks
* Version Control: Managing code changes and releases

**Additional Costs:**

* Project Management: Planning, scheduling, and coordinating development activities
* Legal and Regulatory Compliance: Ensuring compliance with data privacy regulations
* Third-Party Services: APIs, libraries, or other paid services
* Marketing and Promotion: Advertising and promoting the application to attract users

**Cost Estimation:**

Due to the project's scope and varying factors, providing a definitive cost estimate is difficult. However, here's a rough breakdown:

* Development: $5,000 - $20,000+
* Infrastructure: $500 - $2,000+ per month
* Maintenance: $1,000 - $5,000+ per month

**Cost Optimization Strategies:**

* **Open-source libraries and frameworks:**
  + Utilize freely available libraries and frameworks for various functionalities, reducing licensing costs.
* **Cloud-based development and hosting:**
  + Leverage cloud platforms for development and deployment to reduce infrastructure costs and maintenance overhead.
* **Agile development methodology:**
  + Focus on rapid prototyping and iterative development to ensure resource efficiency and early feedback.
* **Community-driven development:**
  + Encourage contributions from open-source communities to leverage shared resources and expertise.

Overall, the cost of developing and maintaining the Peer-to-Peer Multi-User Chatting Application will depend on various factors like project complexity, team size, and chosen technologies. Implementing cost-optimization strategies can significantly reduce expenses and ensure project viability.

# User Authentication

## Authentication Mechanisms for P2P application

1. Username and Password:
   * **Description:** This is the fundamental authentication method where users provide a unique username and password during both registration and login. The application stores and verifies these credentials to ensure secure access.
   * **Implementation:** The passwords are securely hashed before storage, using strong cryptographic algorithms. This prevents storing plaintext passwords and enhances overall security.
2. Account Registration:
   * **Description:** Users can create an account by selecting a unique username and establishing a password. This step is essential for setting up user identities within the application.
   * **Implementation:** During registration, ensure that the chosen username is unique and not already in use. Store user information securely, possibly in a database, and enforce validation rules for password strength.
3. Password Encryption:
   * **Description:** Passwords are encrypted using robust cryptographic algorithms to protect them from unauthorized access, ensuring they are not stored in a readable format.
   * **Implementation:** Implement industry-standard encryption techniques (e.g., bcrypt, Argon2) to hash and store passwords securely. This safeguards user credentials even in the event of a data breach.
4. Multi-Factor Authentication (MFA):
   * **Description:** To enhance security, MFA requires users to provide an additional layer of verification beyond usernames and passwords.
   * **Implementation:** Integrate MFA methods such as one-time codes sent through email or SMS, or the use of authenticator apps. This adds an extra layer of protection, making it harder for unauthorized users to gain access.
5. Session Management:
   * **Description:** Once authenticated, the application needs to manage user sessions securely to prevent unauthorized access.
   * **Implementation:** Use secure session tokens that expire after a set period. Implement mechanisms for users to log out and ensure that inactive sessions are automatically terminated.
6. Account Recovery:
   * **Description:** Users should have a secure way to recover their accounts if they forget passwords or lose access.
   * **Implementation:** Implement a secure account recovery process, such as sending password reset links via email with token validation. This process should be designed to prevent unauthorized access during account recovery.
7. Rate Limiting and Lockout Policies:
   * **Description:** These mechanisms are crucial for preventing brute force attacks by limiting the number of login attempts.
   * **Implementation:** Implement rate limiting to restrict the number of login attempts within a specific time frame. Introduce account lockout policies, temporarily disabling accounts after repeated unsuccessful login attempts.
8. Secure Communication Protocols:
   * **Description:** Ensure that all communication, including authentication, is encrypted over secure channels to protect user data during transmission.
   * **Implementation:** Utilize secure socket protocols (e.g., TLS/SSL) to encrypt data exchanged between clients and servers. This prevents eavesdropping and man-in-the-middle attacks.
9. Logging and Auditing:
   * **Description:** Logging authentication events is essential for monitoring and auditing purposes.
   * **Implementation:** Implement logging mechanisms to record authentication events, including successful logins and failed login attempts. These logs can be valuable for security analysis and compliance purposes.
10. Account Deactivation:
    * **Description:** Users should have the option to deactivate or close their accounts when needed.
    * **Implementation:** Develop a secure account deactivation process that ensures user data is handled appropriately. This may involve anonymizing or securely deleting user data while adhering to privacy and regulatory considerations.

## Types of User authentication for P2P Multi-User Chatting Application

1. Centralized Authentication Server:
   * Employs a central server to store user credentials.
   * Users register and login by communicating with this server.
   * Authenticates credentials using secure hashing algorithms (e.g., bcrypt, scrypt).
   * Offers centralized control for managing user accounts and security measures.
   * Might introduce a single point of failure and potential privacy concerns.
2. Distributed Authentication with a Trusted Node:
   * Distributes authentication responsibility among trusted nodes in the peer-to-peer network.
   * Users register with a trusted node, which stores their credentials securely.
   * Login involves authentication against a trusted node, not a central server.
   * Enhances resilience and privacy compared to a centralized approach.
   * Relies on the trustworthiness of designated nodes.
3. Password-Less Authentication:
   * Eliminates password storage and transmission.
   * Options include:
     + Public Key Infrastructure (PKI): Relies on cryptographic keys for authentication. Users generate a public-private key pair, share their public key, and prove identity using their private key.
     + One-Time Passwords (OTPs): Generate temporary passwords for each login attempt, often sent via email or SMS.

## Key Considerations for Selection

* Security

Protect user credentials and prevent unauthorized access.

* Scalability

Handle increasing numbers of users effectively.

* Decentralization

Aim for decentralized control to align with peer-to-peer architecture.

* User Experience

Ensure a smooth and seamless authentication process for users.

* Implementation Complexity

Choose a method that can be implemented within project constraints.

# System Architecture

In this section, the different architectures that might be used in the design of the system will be discussed and compared.

## Client-Server Model

A diagram of a cloud computing

Description automatically generated

### Definition and characteristics

* A client-server model is a networking computing system design that illustrates a relationship between two or more clients, where the client computers (Hosts) request and receive services or resources from a powerful centralized server computer.
* It enables several clients to access files from a single server, preserving consistency across all devices where Data is stored at server side only.
* This model is widely used in various computing applications, ranging from simple file sharing to complex web-based applications.

### Components of Client-Server Architecture

1. **Client**
   1. Client is end-user device or application that initiates requests for services or resources from a powerful centralized server.
   2. Clients can be computers, smartphones, tablets, or any other device capable of making requests.
2. **Server**
   1. The server is a powerful computer or software application that provides services or resources in response to client requests.
   2. Servers are designed to handle multiple requests simultaneously and can be specialized for specific tasks such as web hosting, database management, or file storage.
3. **Network**
   1. Is the communication infrastructure that allows clients and servers with each other.
   2. This can be a local area network (LAN), a wide area network (WAN), or the internet.
   3. The network facilitates the transmission of data between clients and servers.

## Peer-to-Peer Model

A diagram of a computer network

Description automatically generated

### Definition and Characteristics

* A decentralized network architecture where each participant in the network, referred to as a peer.
* There is no dedicated server and clients. Each node(peer) acts as a server and as well as client which means peers communicate directly with each other to share resources.

So, each node in the network can request a service and can provide a service & store its own data.

A diagram of a computer connection

Description automatically generated

* One-to one Chatting where Peer1 can directly communicate with Peer2 through P2P Connection after Authentication.

A diagram of a network of computers

Description automatically generated

* Chat Rooms where All Peers can communicate directly with each other through P2P Connection by sending Broad messages.

### Components of P2P Architecture

1. **Client**
   1. Each participant in the P2P network is referred to as a “peer”, Each peer has the capability to request resources or services from other peers as well as provide resources or services to others. Peers are equal in terms of functionality, and there is no centralized control.
   2. Peers communicate with each other using P2P communication protocols.
   3. Peers interact with server only for authentication purposes, when a peer joins the network, it might need to authenticate itself before participating fully & for chat room management.

**Examples of Request sent:**

* Chat requests in order to initiate peer-to-peer chat
* Create/Join chat room requests in order to create/join specific chat room
* Sending/Receiving messages in chat room requests

**Examples of Requests sent (Authentication):**

* User login request
* User logout request
* User signup request 🡪 Register if no account exists

1. **Server**
   1. In P2P Network, Server is not a Centralized Entity - Instead, It typically serves specific roles Such as :
      1. Authentication**:** The server plays a role in authenticating peers. When a peer wants to join the network or access certain services, it may send authentication requests to the server to verify its identity. The server can verify the identity of the peer based on credentials, ensuring secure access to the P2P network.
      2. Chat Room Management**:**  The server manages the process of joining/leaving rooms. Peers may request to join/leave specific rooms, and the server coordinates these requests, ensuring that the processes is controlled and organized. And provides a list of Available Chat rooms.
   2. Server facilitates the establishment of P2P connections.
2. **Database**
   1. The database component is responsible for managing and storing data shared among peers. Unlike traditional client-server architectures where a centralized server manages the database, P2P databases are often distributed across multiple peers in a decentralized manner.
   2. Server interacts with database to verify login data of client and store account if user is not registered.
   3. Database provides a list of online peers.

# Software Components

## Component Diagram

A diagram of a computer

Description automatically generated

Assumptions

* Server is intended only for Authentication and Chat Room Management
* Client 1 (acts as client) provides login data to Client peer for sending messages to server peer either one-to one or group chatting ,Client 2(acts as a server) on the other hand provides login data to Server peer component in order to exchange messages through p2p connection.

In the upcoming section each component will be described separately.

## Client-Peer Component

Description

Represents Server- side application responsible for user authentication, Managing p2p connections and Handling Chat Rooms Requests by client peer.

Roles

1. **Authentication Request Management**

Request Authentication of client to database for validation of stored accounts to ensure secure and valid access to the P2P network.

1. **Chat Room Joining Management**

When user requests joining chat room, server will provide a list of available chat rooms to client peer to be able to join target chat room.

1. **P2P Communication Management**

Request user status to database whether server peer is online/offline so that client peer can initiate P2P connection if server peer is online.

## Chat Room Component

Description

Manages direct communication between peers in p2p connection by exchanging messages across it.

Roles

1. **Handling Messages of Client Peer**

Manages processing, and presentation of messages sent by the Client-Peer Component to Ensure effective handling of messages, including displaying them to the user, managing message history, and providing a seamless messaging experience within the P2P network.

1. **Facilitates Messages Transmission**

Enables the client to communicate messages to the Server peer without relying on a centralized messaging channel. This feature supports efficient and direct data transfer between the client and the server peer through p2p connection.

## Database Component

Description

System responsible for managing ,organizing and storing data within a peer-to-peer (P2P) architecture.

Roles

1. **User Registration:**

When a user joins the P2P network or requests to access specific resources, they need to register with the system.

Registration involves providing necessary information (username, password, etc.), and this information is securely stored in the distributed database.

1. **Authentication Requests:**

When a user attempts to log in, the peer initiates an authentication request. This request was sent to database for verification.

1. **Provides List of Online Peers:**

When a client peer wants to send a message to server peer, it must be online So, request is sent by server to database to check status and then database respond by sending list of online peers.

## Server-Peer Component

Description

emphasize the server's role in a peer-to-peer (P2P) architecture**.**

Roles

1. **P2P Communication**

Server peer interact with client-peer component when client-peer sends a message through P2P connection.

1. **Chat Room Interactions**

Server peer also can join chat room by providing username of client.

## Sequence Diagrams

### Login

A screenshot of a computer screen

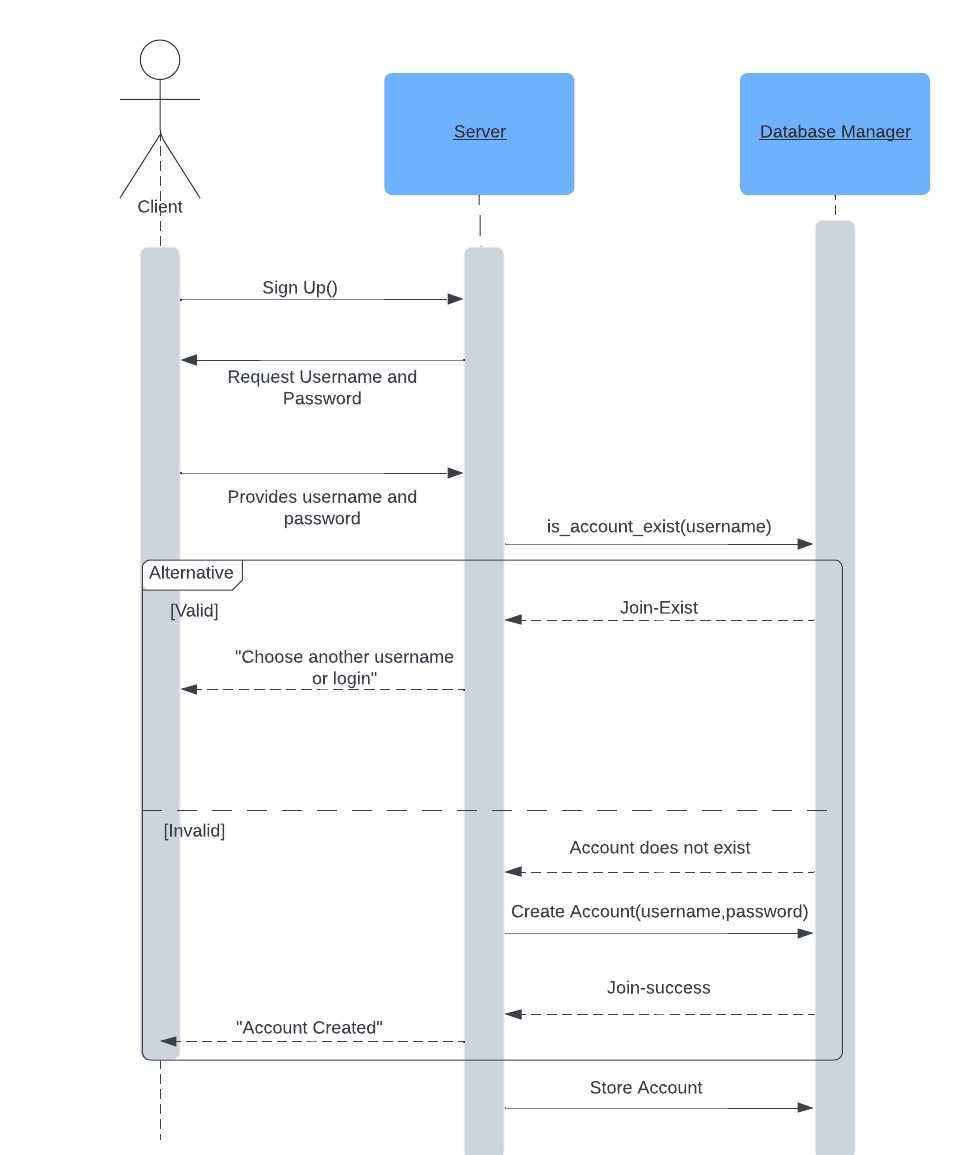
Description automatically generated

### Logout

A diagram of a login

Description automatically generated

### Signup



### One-to-One Chatting

A diagram of a project

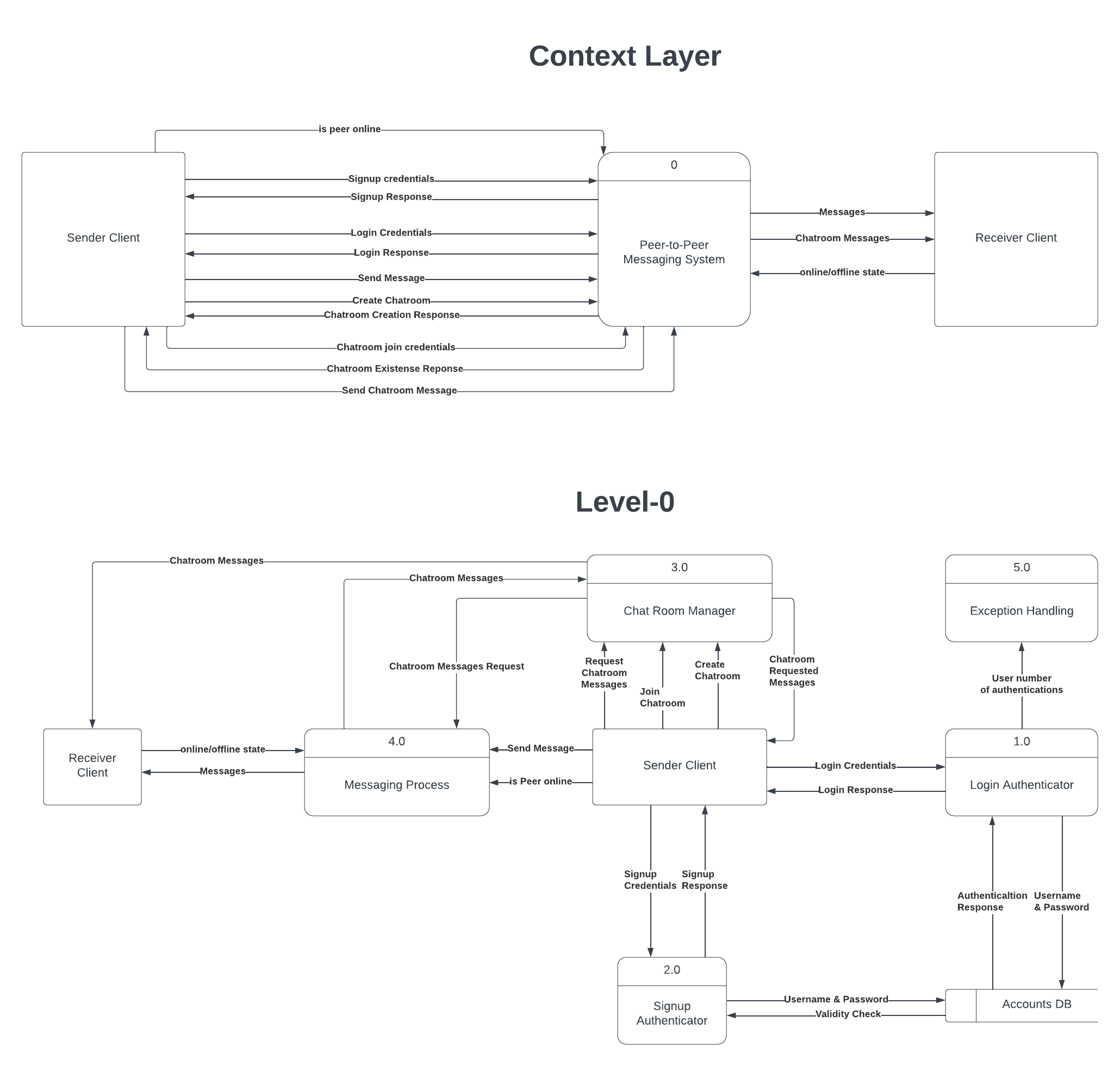
Description automatically generated

### Group Chatting

A blueprint of a building

Description automatically generated

## Data Flow Diagram



# Communication Protocols

## Client-Server Protocol

**Establishing Client Server Connections**

Establish a TCP connection first, and it should be established successfully after that.

Client requests to create ana account to the server then server sends that account is created.

To create authentication, clients first send requests to log in. If the username and password are incorrect, the server rejects the request (404 not found). If the login details are correct, the server responds with an acceptance (200 ok).

Server maintain a list of available online peers.

Handshaking occurs, the client requests to send a message to server, and server checks if the other client is online. If the other client is online the server which delivers message. If the other client is offline, then the request is rejected.

Client sends requests to create or join chat rooms then server manage the creation and joining of chat rooms.

**A diagram of a network

Description automatically generated**

## Peer-to-Peer Protocol

Establishing Peer-to-Peer Connections

Each user in the chat room will act as a peer and have their own instance of the database, representing their local database connection. Peers need a way to discover and connect with each other. This can be achieved by implementing a peer discovery mechanism, where peers exchange their IP addresses and port numbers.

You can modify the database to discover and connect with other peers in the chat room to retrieve a list of online peers from the database.

Joining the Chat Room

When a peer wants to join the chat room, they can use the peer discovery mechanism to find other peers. Once the peer obtains the IP addresses and port numbers of other peers, they can establish direct connections with them.

The peer can use the database to mark themselves as online and store their IP address and port number in the online peer’s collection.

Peer-to-Peer Message Exchange

With peer connections established, peers can send and receive messages directly to/from each other. You can create a messaging system where peers exchange messages using their established connections. Peers can use the database to retrieve the IP address and port number of a specific peer before sending a message to them.

Consider implementing a message format to determine the structure of the messages exchanged between peers. This format should include information such as the type of message, sender, receiver, and the actual content of the message. You can define a message format using a specific syntax, such as JSON or plain text.

Broadcasting Messages to the Group

To implement group chat functionality, each peer needs to broadcast messages to all other peers in the chat room. You can modify the code to include a method that sends a message to all connected peers. This method can iterate through the list of connected peers obtained from the online peer’s collection and use their IP addresses and port numbers to send the message.

Leaving the Chat Room

When a peer wants to leave the chat room, they can use the logout method of the database to mark themselves as offline and remove their entry from the online peers collection. Additionally, the peer should close their UDP socket and terminate any established connections with other peers.

Create an account

Client is requesting from the server to create an account then the server stores the username and password in the database.

A screen shot of a computer screen

Description automatically generated

Login to an account

The client requests from the server to login then server checks in database the username, password, IP address and port number.

If login details are accepted, then the database returns the list of online peers and server sends to the client that it is accepted (200 ok).

A screen shot of a computer

Description automatically generated

If login details are rejected, then the database returns invalid details and server sends to the client that it is rejected (404 not found).

A screen shot of a computer

Description automatically generated

One to one chat

To Establish peer connections, each peer should have a mechanism to establish connections with other peers. This can be done using IP addresses and port numbers. Peers can connect to each other using sockets, such as TCP or UDP sockets. The server socket is used to accept incoming connections from other peers. Using TCP connection for one-to-one chat.

Client 2 must be online (logged in) to be able to receive the message so the server checks in database if client 2 is online and server sends to client 1 that request accepted (200 ok) and the connection will be initiated.

If client 1 requests to send chat to client 2 and client 2 is offline the server will reject the request and server will send to the client a rejection response (404 not found).

If client 1 wants to start a chat with client 2, it sends to server a request to chat with client 2 then the server sends IP address and port number of client 2 to client 1 and sends IP address and port number of client 1 to client 2 to make a peer connection.

When two peers establish a connection, they need to perform a handshake to negotiate the communication parameters. This handshake can include exchanging information about supported protocols, capabilities, and establishing a shared encryption key.

Client 1’s message will be encrypted and sent to client 2 and will be decrypted to be read.

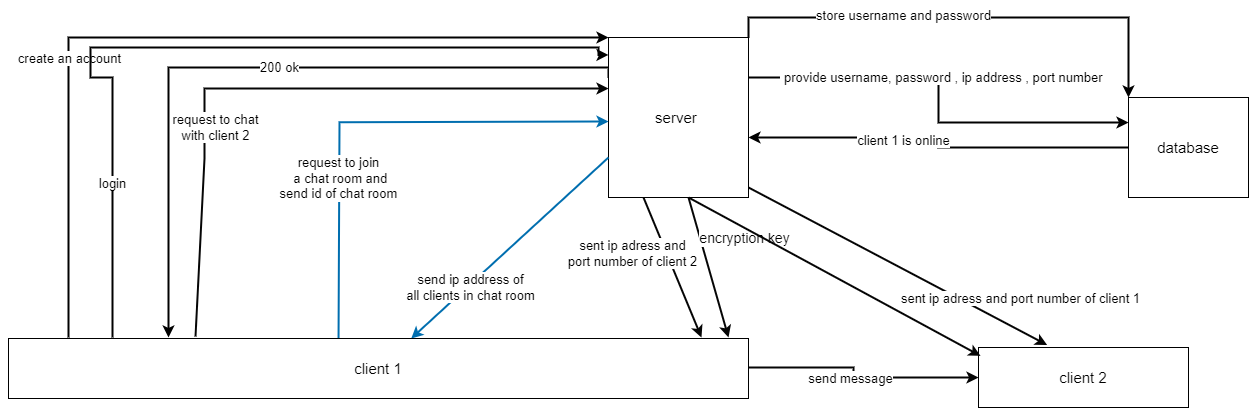
If client 1 requests to start a chat with client 2, server checks the list of online peers in database, if client 2 is busy with other chat the server sends to client 1 that the request is rejected, and connection will not be initiated.

**A diagram of a diagram

Description automatically generated**

Chat room

Client 1 requests to join a chat room and id of group chat that wants to join from a server then server sends IP address of all clients in chat room then it starts the broadcasting to send the messages to all the clients in the group and starts a UDP connection.



Client 1 requests to create a new chat room then the server adds the new chat room to the database.

Client 1 requests to see a list of available chat rooms then the server requests the list from the database then the database sends the list to the server which is resent to Client 1.

A diagram of a building

Description automatically generated

Logout

Client 1 requests to logout from server then server sends the username to the database then database send that the user is offline the server sends to the client 200 ok (logged out successfully)

A diagram of a diagram

Description automatically generated

## Message Formats

|  |  |
| --- | --- |
| Chat request message Format | Chat Request:  {  “Action”: "CHAT-REQUEST",  “Username\_sender”: “Medhat2990”,  “Username\_receiver”: “Fawzy9000”  }  Acceptance Response:  {  “Status”: “OK”,  “Message”: "Client with OK message is created... and sending messages"  }  Chat Response:  {  “Status”: “REJECT”,  “Message”: "client of requester is closing..."  }  Chat Response:  {  “Status”: “BUSY”,  “Message”: "Receiver peer is busy"  } |
| Quit Message Format | Quit request:  “Action”: ":q",  “Username\_sender”: “Medhat2990”,  “Username\_receiver”: “Fawzy9000”  “Message”: “Hello, how are you?”  }  Received message response is empty  {  “Status”: “empty-message-not-chatting”,  “Message”: “chat suddenly ended”  }  Message response is not an empty  {  “Status”: “:q-is-sent”,  “Message”: “User you're chatting with ended the chat”  } |
| Create Account Message Format | Create account request:  {  “Action”: "JOIN",  “username”: “medhat2990”,  “password”: “pass2990”  }  Create Account Response:  {  “Status”: "join-success",  “Message”: "Account created..."  }  Create Account Response:  {  “Status”: "join-exist",  “Message”: "choose another username or login..."  } |
| Login Message Format | Login request:  {  “Action”: " LOGIN",  “username”: “medhat2990”,  “password”: “pass2990”  }  Login response:  {  “Status”: "login-success",  “Message”: "Logged in successfully..."  }  Login response:  {  “Status”: "login-account-not-exist",  “Message”: "Account does not exist..."  }  Login response:  {  “Status”: "login-online",  “Message”: " Account is already online..."  }  Login response:  {  “Status”: "login-wrong-password",  “Message”: "Wrong password..."  } |
| Logout Message Format | Login request:  {  “Action”: " LOGOUT",  “username”: “medhat2990”,  }  Logout response:  {  “Status”: “user is offline”,  “Message”: “Logout successfully”  } |
| Search User Message Format | Search user request:  {  “Action”: " SEARCH-username"  }  Search user response:  {  “Status”: "search-success",  “Message”: " is found successfully..."  }  Search user response:  {  “Status”: "search-user-not-online",  “Message”: " is not online..."  }  Search user response:  {  “Status”: "search-user-not-found",  “Message”: " is not found"  } |
| Hello Message Format | Hello message request:  {  “Action”: " HELLO username",  } |
| User authentication | user request:  {  “Action”: "AUTHENTICATE",  “username”: “medhat2990”,  “password”: “pass2990”  }  user response:  {  “Status”: " authentication-check",  “Message”: " authentication done successfully "  } |
| Joining a chat room message format | Join request:  {  “Action”: "JOIN-CHAT-ROOM",  “chat room id”: “general ”  }  Join Response:  {  “Status”: "join-success",  “Message”: "joined chat room successfully", |
| Send a message to chat room | Send request:  {  “Action”: " SEND-TO-CHAT-ROOM",  “chat\_room\_id”: “general ”,  “Message”: “Hi, everyone”  }  Send response:  {  “Status”: "send-success",  “Message”: "sent successfully"  } |
| Send a message to one-to-one chat | Send request:  {  “Action”: " SEND-TO-ONE-TO-ONE-CHAT",  “Username\_sender”: “Medhat2990”,  “Username\_receiver”: “Fawzy9000”  “Message”: “Hello, how are you?”  }  Send response:  {  “Status”: "send-success",  “Message”: "sent successfully"  } |