**April 20,2024**

**EECE 350 Project Report**

**Presented to Professor Sanaa Sharafeddine**

**Group members:**

* **Riad Khoury (202300496)**
* **Mario Khoury (202300707)**
* **Christia Daoud (202303412)**
* **Anthony Haddad (202309103)**

**Workload Distribution:**

* **Riad Khoury (25%)**
* **Mario Khoury (25%)**
* **Christia Daoud (25%)**
* **Anthony Haddad (25%)**

**PART 1: PHASE 1**

**System Architecture Description**

* **Client-Server Architecture:**
  + The system follows a client-server architecture where communication happens over TCP/IP sockets.
  + The server manages user authentication, maintains user data, handles messaging, and facilitates group interactions.
  + The client interacts with the server to perform user authentication, send/receive messages, manage friends, and participate in group chats.
* **Server Components:**
  + **User Management:**
    - Stores user credentials securely.
    - Manages user login and signup processes.
  + **Friendship Management:**
    - Tracks friend relationships between users.
    - Allows adding and removing friends.
  + **Group Management:**
    - Handles group creation and membership.
    - Facilitates joining and leaving groups**.**
  + **Message Handling:**
    - Manages real-time messaging between users.
    - Supports offline message delivery.
  + **Chat History:**
    - Stores chat history between users for future reference.
  + **Threading:**
    - Utilizes threading to handle multiple client connections concurrently.
* **Client Components:**
  + **User Interface (GUI):**
    - Provides a graphical interface for user interaction.
    - Allows users to sign up, log in, view friends, join groups, and send messages.
  + **Functionalities:**
    - Supports signup and login processes.
    - Displays online and offline friends.
    - Facilitates adding friends and creating/joining groups.
    - Provides a chat interface for real-time messaging.
  + **Additional Features:**
    - Includes a notepad, task list, and a simple calculator for user convenience.
    - These features are accessible from the chat interface.
* **Communication Protocol:**
  + Defines specific message formats for different operations (e.g., SIGNUP, LOGIN, SEND\_MESSAGE).
  + Messages are encoded and decoded to ensure proper communication between client and server.
* **Scalability and Extensibility:**
  + The architecture allows for easy scalability by supporting multiple concurrent connections.
  + New features can be added by extending the existing functionalities of the server and client.
* **Security Considerations:**
  + The system should implement secure communication protocols to protect user credentials and messages during transmission.
  + Further enhancements can be made to ensure data integrity and confidentiality.

**Protocol Description**

* This chat program employs socket connections and a unique text-based syntax for its communication mechanism. A specified text format, such as "LOGIN:username:password" for login, is sent by the client when it establishes a connection with the server on port 9999. The multithreaded server responds to these requests by processing them and returning encoded replies.
* Real-time communication and the capacity to manage several clients at once are made possible by this method. A constant and dependable connection for the chat application's communication is ensured by the use of sockets, and the straightforward text-based protocol allows for easy extension and customization.

**Features Overview Table**

|  |  |
| --- | --- |
| Feature | Implemented |
| User Signup | YES |
| User Login | YES |
| Direct Messaging | YES |
| Friend List | YES |
| Add Friend | YES |
| Group Chat | YES |
| Create Group | YES |
| Join Group | YES |
| Offline Messaging | YES |

**Part 2: Phase 2**

**Functionality Implementation**

* Several features were added to this chat application project in Phase Two in order to improve user involvement and conversation. A client-server architecture with socket-based communication was used to provide direct messaging.
* Messages are delivered from the client to the server, which forwards them to the intended recipient. Users can now create and join groups using the newly integrated group chat capability, which also allows the server to handle group message routing and membership maintenance.
* In order to guarantee that users who are not online receive their messages when they log in, offline messaging was introduced.
* One of the implementation challenges was managing several client connections at once. This was overcome by the server-side multithreading, which enabled it to process several requests at once. A robust system for storing and retrieving messages was necessary while dealing with offline users, and a bespoke text-based protocol was needed to ensure smooth communication between the client and server.
* These difficulties were overcome by creating a scalable architecture and employing suitable synchronization methods to prevent inconsistent data, resulting in a dependable and useful chat program.

**Future Networking Project**

* As a group we decided that building on our chat program, a real-time video conferencing platform could be a future networking effort. This would turn our text-based conversation into a video and audio-based virtual meeting or social media platform.
* For peer-to-peer connections, we would leverage WebRTC and modify your current server to function as a signaling server, controlling these connections. The Tkinter-based user interface would need to be updated to provide audio controls and video streaming in order to include video.
* Scalability is also essential and may call for server clusters and load balancing. The increased need for remote communication in social, educational, and business situations is addressed by this project.
* Honestly, we found this to be a very beneficial project that would expand our chatting platform and be able to allow users to communicate not only through text but also through more live interactive ways.

**Conclusion**

* In conclusion, a completely functional client-server communication system was produced by the chat application project, which also achieved important goals like group chat, offline message handling, friend management, direct messaging, and user authentication.
* Users can register, log in, add friends, and take part in chat groups on the client-side, which was created using Tkinter.
* Communication in real time is made possible by the multithreaded server's effective management of several clients. Notable takeaways include the necessity of scalable architecture to sustain performance under load and the necessity of strong error handling in multithreaded systems.
* Although the project's unique text-based protocol offered flexibility, it also necessitated careful planning to guarantee continuous communication. All things considered, the project produced a dependable chat platform with a strong basis for future growth and scalability.