**PROJECT REPORT**

**COMPUTER NETWORKS**

**CHAT-APPLICATION**

**3ST SEM**

**B. Tech Computer Science Engineering Artificial Intelligence**



**Prepared by D-09**

**Teja: [CB.SC.U4AIE23328]**

**K.Devi Aakash: [CB.SC.U4AIE23311]**

**c.srikar : [CB.SC.U4AIE23358]**

**Kalyan Rohit M: [CB.SC.U4AIE23318]**

**Project Advisor Prof J.JAISOORAJ**

**Amrita School Of Engineering**

**Amrita Vishwa Vidyapeetham**

**Coimbatore, 641112**

**1. Introduction**

The chat application developed as part of this project is designed to facilitate real-time communication between users over a local or remote network. This application incorporates features such as text messaging and file sharing, making it ideal for collaborative work, educational purposes, or casual conversations. The primary objective was to create a user-friendly interface with visually appealing elements and robust backend support to manage client-server communication efficiently.

This chat application aims to merge ease of use with effective performance. Its architecture ensures that multiple users can connect, share text messages, and exchange files seamlessly. In addition to the core functionality, special attention has been given to ensure a visually attractive GUI to enhance the user experience.

**2. Working of the Method**

The chat application functions using a client-server architecture built with Python’s socket library. The server runs in the background, listening for connections from multiple clients. Once a client is connected, they can send messages or files that the server then broadcasts to all other connected clients. Below is an outline of the key components and processes involved:

**Client-Server Connection:** The server is programmed to bind to a specific IP address and port, establishing the central communication hub. Upon a client connection, a new thread is spawned for that client, allowing simultaneous interactions and handling multiple clients concurrently.

**Message Handling:** The application distinguishes between text and binary data through tagged protocols. Messages are tagged as "MSG" for standard text and "FILE" for file transfers, enabling the server and clients to manage the data appropriately.

**Broadcasting Mechanism:** The server takes on the role of redistributing received messages or files from one client to all connected clients. This ensures real-time, responsive communication without any noticeable delay.

**Graphical User Interface (GUI):** The client interface, created using the Tkinter library, includes a display panel for chat content, an input field for composing messages, a send button, and an option to upload and send files. Each user’s message and server-generated messages are displayed with distinct colors and formatting for better readability and user interaction.

**Thread Management:** Threads are employed for handling incoming messages and file transfers, running asynchronously to prevent the main program from becoming unresponsive during long operations or multiple client interactions.

The integration of these elements allows the chat application to provide seamless and efficient communication. The combination of backend threading and a responsive GUI ensures that users can interact in real time without interruptions.

**3. Brief Description of Tools/Software Involved**

Several essential tools and libraries were utilized to develop the chat application. Below is a brief description of the key technologies:

**Python:** The programming language chosen for the project due to its versatility and rich ecosystem of libraries.

**Socket Library:** A core module in Python that enables network communication by supporting the TCP/IP protocol necessary for client-server data transmission.

**Tkinter:** The standard GUI toolkit for Python, used for designing the user interface. Tkinter provides an assortment of widgets and tools for layout management, allowing for an intuitive and interactive client interface.

**Threading:** A built-in Python library that manages concurrent operations. This was critical for enabling non-blocking message reception and ensuring that the server could handle multiple clients simultaneously.

**OS Module**: This module was used for file management, such as creating directories to store received files. It ensures that the application can access and manage file paths and directories efficiently, enabling smooth file transfers.

Python served as the backbone for both the server and client-side programming. Its simplicity and comprehensive standard library made it an optimal choice for quick development and iteration. The **socket library** provided the foundation for reliable client-server communication, crucial for handling network connections and transmitting data.

The client interface, crafted with **Tkinter,** aimed to offer an interactive, user-friendly experience. Widgets like text boxes, buttons, and file dialog options were seamlessly integrated for a cohesive layout.

Threading played a vital role in maintaining the responsiveness of the application. By allowing different processes to run concurrently, users could send and receive messages without experiencing delays or freezes. Lastly, the **OS module** was pivotal for handling file operations, enabling features such as file uploads, downloads, and storage.

The combination of these technologies and tools resulted in a robust, feature-rich chat application. Its real-time messaging capabilities, coupled with an easy-to-use interface, make it suitable for various purposes, including personal, educational, and collaborative communication. The well-structured architecture ensures stability and efficiency, providing users with a reliable communication platform.