ChatFlow

A PROJECT REPORT
for
Mini Project-II (K24MCA18P)
Session (2024-25)

Submitted by

Mahima Goyal (202410116100112)
Kunal Prajapati (202410116100108)
Kartikey Dwivedi (202410116100098)
Krishna (202410116100092)

Submitted in partial fulfilment of the Requirements for the Degree of

MASTER OF COMPUTER APPLICATIONS

Under the Supervision of
Dr. Prashant Aggarwal
Associate Professor
Ms. Shruti Aggarwal
Assistant Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad Uttar Pradesh - 201206 (MAY-2025)

CERTIFICATE

Certified that **Mahima Goyal (202410116100112), Krishna (202410116100103) and Kartikey Dwivedi (202410116100098)** and **Kunal Prajapati (202410116100108)** has/ have carried out the project work having "**Chatflow**" (**Mini Project-II**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Dr Prashant Aggarwal	**Dr. Akash Rajak**
Associate professor	**Dean**
Department of Computer Applications	**Department of Computer Applications**
KIET Group of Institutions, Ghaziabad	**KIET Group of Institutions, Ghaziabad**

ABSTRACT

ChatFlow is a real-time, anonymous chat website designed to facilitate secure and private communication without requiring users to register or share personal details. Built using **Java Spring Boot** for the backend, **JavaScript** for the frontend, and a **relational database (MySQL/PostgreSQL)** for session management, the platform leverages **WebSocket technology** for instant, low-latency messaging.

The project focuses on providing a **seamless and anonymous** chatting experience while ensuring data security through **encryption, session-based storage, and automatic message deletion**. Key features include **instant user pairing, a responsive UI, spam prevention mechanisms, and cross-device compatibility**.

As the project is currently in development, challenges such as **real-time performance optimization, handling concurrent users, and implementing robust security measures** are being addressed. Planned enhancements include **group chat support, end-to-end encryption, a mobile application, and AI-based spam detection**.

ChatFlow aims to be a **scalable, user-friendly, and privacy-centric** communication platform, ensuring an engaging and safe chatting experience. The final product will provide a **secure, real-time messaging solution** that balances anonymity with functionality.

ACKNOWLEDGEMENT

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Dr Prashant Aggarwal** for her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to Dr. Akash Rajak, Professor and Dean, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions. Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Mahima Goyal

Krishna

Kunal Prajapati

Kartikey Dwivedi

TABLE OF CONTENTS

Introduction

In an era where digital communication is essential, privacy and user anonymity are becoming increasingly important. Most existing chat platforms require users to register, provide personal information, or connect via social media, which can deter users who prefer confidentiality. **ChatFlow** addresses this concern by offering a **real-time, anonymous chat platform** that allows users to communicate without disclosing their identity. It is a lightweight, privacy-first solution aimed at creating a secure and seamless chatting experience

Privacy and Anonymity

ChatFlow's core goal is to prioritize user anonymity. No registration or login is required, and users are randomly paired for conversations. The platform avoids collecting or storing personally identifiable information.

- No login or signup required
- Random user pairing
- Temporary, session-based data storage
- No permanent logs maintained

Real-Time Messaging

To ensure low-latency, responsive conversations, ChatFlow uses **WebSocket technology**. This enables bidirectional communication between clients and the server, unlike traditional HTTP requests.

- Instant message delivery
- Real-time, bi-directional communication
- Lag-free chatting experience
- No need to refresh the page for updates

Robust Technology Stack

ChatFlow is developed using modern, open-source technologies that ensure scalability and flexibility. This makes it easy to deploy, maintain, and scale as needed.

- **Backend**: Firebase
- **Frontend**: HTML, CSS, JavaScript (React optional)
- **Database**: PostgreSQL/MySQL
- **Deployment**: AWS/DigitalOcean using Apache/Nginx
- **Development Tools**: Visual Studio Code, Postman

Features and Functionality

The platform includes several built-in features that improve usability and security. It supports instant communication while implementing basic safety controls.

- Simple and clean UI design
- Automatic anonymous pairing
- CAPTCHA for spam prevention
- Designed for both desktop and mobile use
- Plans for group chat, AI moderation, and encryption

Use Case Scenarios

ChatFlow is ideal for scenarios where privacy is essential, or where users want to communicate without forming long-term connections.

- Mental health support and anonymous counseling
- Peer-to-peer student discussion forums
- Employee suggestion portals
- Temporary chat rooms for events or webinars

Future Enhancements

As part of ongoing development, ChatFlow aims to introduce advanced features that strengthen security and improve user engagement.

- End-to-end encryption
- Group chats and public rooms
- AI-based spam detection and moderation
- Mobile application for Android/iOS

Summary

ChatFlow is designed to bridge the gap between **user privacy** and **real-time communication**. By using modern technologies, it ensures that users have a smooth, secure, and anonymous chatting experience. It offers a unique solution for individuals or organizations seeking private online communication without compromising on performance or ease of use

Feasibility Study

A feasibility study is conducted to determine whether the proposed system—**ChatFlow**—can be developed and deployed successfully within the available technical, economic, social, and operational constraints. It ensures that the project is **viable, practical, and beneficial** before investing significant time and resources.

1. Technical Feasibility

This aspect evaluates whether the current technology stack and development tools are sufficient to build the application effectively.

Key Points:

- **Proven Tech Stack**: ChatFlow uses Java Spring Boot, WebSocket, PostgreSQL, and JavaScript—all reliable and widely adopted technologies.
- **WebSocket Support**: Supported by modern browsers and backends, enabling real-time communication without additional libraries.
- **Compatibility**: Works across different operating systems (Windows, Linux, macOS) and modern browsers (Chrome, Firefox, Safari, Edge).
- **Cloud Deployment Ready**: Supports deployment on AWS, DigitalOcean, or local servers with Apache/Nginx.
- **Frontend Flexibility**: HTML, CSS, JS (and optionally React) enable responsive UI design and fast interaction.

Conclusion: The project is **technically feasible** due to the availability of required tools, developer expertise, and platform compatibility.

2. Economic Feasibility

This considers whether the project is cost-effective and affordable with respect to development, deployment, and maintenance.

Key Points:

- **Open-Source Tools**: ChatFlow is built using free and open-source tools (Firebase, PostgreSQL, WebSocket API, VSC), reducing licensing costs.
- **Minimal Hardware Investment**: Only basic hardware (8-core CPU, 16 GB RAM) is needed for hosting, and end-users require only internet-connected devices.

- **Cloud-Friendly**: Hosting on platforms like AWS or DigitalOcean can be done within student project budgets or using free tiers.
- **Low Maintenance**: Session-based architecture means no long-term data storage, reducing storage and backup costs.

Conclusion: ChatFlow is **economically feasible** for both academic use and scalable deployment with minimal costs.

3. Operational Feasibility

This evaluates how easily the system can be used, maintained, and integrated into users' daily activities.

Key Points:

- **User-Friendly Interface**: Clean, simple UI ensures ease of use even for non-technical users.
- **Minimal Training Required**: No instructions needed; users simply visit the website and start chatting.
- **Automatic Pairing**: Eliminates the need for manual user selection or contact list management.
- **Anonymous Interaction**: Reduces user hesitation and increases participation, especially in sensitive discussions.

Conclusion: The system is **operationally feasible** as it is intuitive, self-explanatory, and requires no administrative setup for use.

4. Schedule Feasibility

This checks whether the project can be developed within the available time frame and resource constraints.

Key Points:

- **Two-Month Plan**: The timeline includes UI design, backend integration, security implementation, and deployment.
- **Team of 4 Developers**: Tasks can be distributed efficiently, improving development speed.
- **Agile-Like Workflow**: Allows iteration and testing at every stage to ensure progress without delays.

Conclusion: With proper planning and division of tasks, the project is **feasible within the allotted schedule**.

5. Social Feasibility

This explores the potential acceptance and usefulness of the system by its intended users and society.

Key Points:

- **Promotes Safe Communication**: Helps users engage in sensitive or private discussions without fear.
- **Educational Use Cases**: Enables anonymous feedback and student support chats in schools/colleges.
- **Mental Health Applications**: Encourages people to speak freely about mental health and personal issues.
- **Ethical Design**: Avoids data mining and respects user anonymity, building trust with users.

Project Objectives

The primary objective of the **ChatFlow** project is to build a secure, real-time, and anonymous communication platform that allows users to interact without disclosing personal information.
In today's digital landscape, privacy concerns are growing rapidly. Most existing chat platforms require users to register with email IDs, phone numbers, or social media accounts practices that compromise anonymity and expose users to risks.

ChatFlow challenges this approach by offering a safe, anonymous space for users to engage in conversations without any form of identity tracking.

Real-Time Experience
The project ensures a **seamless and responsive chatting experience** by utilizing **WebSocket technology**, which enables **bi-directional, low-latency** data transmission between client and server.
Unlike traditional HTTP request-response methods, WebSockets allow **instant messaging**, creating a natural flow of conversation without noticeable delays.

Backend and Frontend Architecture
The backend is powered by **Firebase** a reliable and scalable framework that supports efficient handling of multiple concurrent users.
On the frontend, **JavaScript** (with optional **React**) is used to develop a clean, responsive UI that works smoothly across all modern devices and screen sizes.
This choice of technologies ensures both **performance and adaptability**.

Security and Privacy
Security is a key pillar of ChatFlow. All data exchanged over the platform is **encrypted** to prevent unauthorized access and eavesdropping.
The system employs **session-based data storage**, meaning no chat history is saved permanently. Once a session ends, all associated messages are automatically deleted.
This significantly enhances privacy while reducing server storage needs.

Automatic User Pairing and Moderation
To simulate real-world anonymous communication, ChatFlow includes an **automatic user pairing system**.
Users are randomly matched with others for spontaneous one-on-one conversations without needing to browse profiles or initiate friend requests.
To maintain a safe and respectful environment, **spam prevention mechanisms** such as **CAPTCHA**, **rate-limiting**, and **AI moderation tools** are incorporated or planned for future updates.

Scalability and Future Scope
The system is built with **future growth in mind**. ChatFlow is designed to support features like:

- **Group chats**
- **End-to-end encryption**
- **Mobile app integration**
- **AI-driven content moderation**

This ensures the platform remains relevant, robust, and scalable as technology and user needs evolve.

Key Project Objectives

- Enable secure and anonymous chatting without registration
- Implement WebSocket for real-time communication
- Use Java Spring Boot and JavaScript for scalable development
- Protect user privacy through encrypted, session-based architecture
- Design a responsive and intuitive UI
- Incorporate spam filters and security measures
- Support spontaneous user pairing for anonymous interaction
- Provide flexibility for future enhancements
- Promote ethical, safe, and inclusive communication
- Lay the groundwork for AI-powered moderation

Hardware and Software Requirements

Developing and deploying a real-time, anonymous chat application like ChatFlow requires a well-planned set of hardware and software tools. These tools must ensure smooth development, testing, deployment, and efficient runtime performance. Below is a detailed breakdown of both hardware and software requirements along with their roles in the project.

Hardware Requirements

- **Server-Side Hardware Requirements**

The server is responsible for handling real-time communication, managing user sessions, encrypting data, and storing temporary chat messages. Hence, the following specifications are recommended:

Minimum 8-Core Processor (Intel Xeon or AMD Ryzen preferred):

A multi-core processor ensures parallel processing, which is essential for managing multiple chat sessions simultaneously. Real-time applications demand high concurrency, and a powerful CPU helps maintain low latency.

16 GB RAM or Higher:

Adequate memory is crucial for efficient multitasking, particularly when handling WebSocket connections, user authentication, session tracking, and database interactions concurrently.

500 GB SSD Storage:
Although ChatFlow uses session-based messaging (temporary storage), an SSD ensures fast read/write speeds, which benefits real-time performance, server boot-up, and database operations.

High-Speed Internet Connection:

WebSocket requires a persistent TCP connection between client and server. A fast, stable internet connection with low latency is necessary for reliable real-time communication.

Load Balancer (Optional for Scaling):

For future scalability, a load balancer can distribute user traffic across multiple servers, enhancing availability and performance. Tools like HAProxy or NGINX can serve this purpose.

- **Client-Side Hardware Requirements**

The end-users of ChatFlow do not require advanced hardware. The following is sufficient:

Any Modern Device (PC, Laptop, Tablet, Smartphone):

Since ChatFlow is web-based, users can access it from any internet-enabled device with a web browser.

Internet Connection:

A stable internet connection ensures uninterrupted communication and WebSocket connectivity.

Updated Web Browser (Chrome, Firefox, Edge, Safari):

These modern browsers support WebSocket APIs natively and handle frontend JavaScript code efficiently, ensuring smooth user interaction and responsive UI behavior.

Software Requirements

The software tools used for ChatFlow are selected to support development efficiency, maintainability, cross-platform compatibility, and future scalability.

- **Operating System**

Windows, macOS, or Linux:

The development and hosting environment for ChatFlow is platform-independent. Java and its frameworks (Spring Boot, Hibernate) run across all major OSs, making it flexible for developers to work on any system. Linux is often preferred for production servers due to its stability and resource efficiency.

- **Backend Development**

Superbase:

Supabase is chosen for this project due to its simplicity, modern toolset, and ability to manage real-time interactions without requiring custom server-side coding. It reduces backend development effort and focuses on security and performance.

WebSocket API:

WebSocket enables full-duplex communication between the server and client. Unlike HTTP, which works on request-response cycles, WebSockets maintain a persistent connection, allowing real-time message delivery without polling the server repeatedly.

Hibernate (ORM for Database Management):

Hibernate handles object-relational mapping (ORM), translating Java objects into database tables and vice versa. This reduces boilerplate JDBC code and simplifies interactions with relational databases such as MySQL or PostgreSQL.

- **Frontend Development**

HTML & CSS:

HTML structures the web pages while CSS is used for styling and layout. These technologies together ensure the chat interface is accessible, visually appealing, and responsive.

JavaScript:

JavaScript is used to implement client-side logic such as message sending/receiving, WebSocket handling, form validations, and UI interactions. It ensures the chat interface responds in real time to user inputs and server messages.

React:

If used, React helps in building reusable UI components. It improves performance with a virtual DOM and makes frontend code easier to manage and scale as the application grows.

WebSocket Client Implementation:

JavaScript provides built-in support for WebSocket connections (via WebSocket object). It allows the frontend to send messages to and receive messages from the backend server in real time.

- **Database Management**

MongoDB:

Although a relational database is suggested in some parts of the project, MongoDB offers a flexible, document-oriented storage model. It stores data in JSON-like documents which is ideal for unstructured or semi-structured chat messages. Session-based temporary storage is easier and faster to implement with MongoDB.

- **Server and Hosting**

Apache/Nginx:

These are high-performance web servers used to serve frontend content, manage reverse proxying to backend services, and handle security configurations like SSL certificates. NGINX is also used for load balancing.

AWS / DigitalOcean:

Cloud platforms like Amazon Web Services and DigitalOcean offer scalable infrastructure for deployment. They provide hosting, storage, bandwidth, and auto-scaling features. They also support continuous deployment and security features like firewalls and domain management.

- **Development Tools**

IntelliJ IDEA / Eclipse:

These are Integrated Development Environments (IDEs) for Java development. They support debugging, syntax highlighting, version control integration

Timeline for Completion

The development of **ChatFlow** is planned to be completed over a period of **two months**, following a structured and systematic approach. The timeline ensures a balance between frontend and backend development, integration, testing, optimization, and deployment. With four team members, tasks can be efficiently distributed to achieve high productivity while maintaining quality. Each week is assigned specific objectives to guarantee steady progress and timely delivery of the project.

Month 1: Frontend Development and Backend Setup

Week 1–2: UI/UX Design and Initial Frontend Development

The initial two weeks of development are dedicated to planning, designing, and building the frontend of the application.

- **Wireframing and Prototyping**: During this phase, visual wireframes and interactive UI mockups will be created using tools such as Figma or Adobe XD. These prototypes help visualize the layout, chat flow, and user interactions.
- **User-Centric Design**: The primary goal is to ensure the interface is clean, minimal, and intuitive, allowing users to chat anonymously without any confusion. Focus will be placed on responsive design to support various devices including desktops, tablets, and smartphones.
- **Feedback Collection**: Preliminary UI prototypes will be shown to a small group of users or peers to gather early feedback. Their suggestions will be used to refine the design before full implementation.
- **Frontend Development**: The final UI will be implemented using **HTML**, **CSS**, and **JavaScript**, with optional integration of **React** for component-based UI development and better code management.

Week 3–4: WebSocket Integration and Backend Setup

These two weeks focus on establishing the server-side architecture and integrating real-time communication capabilities.

- **WebSocket Backend Development**: A dedicated module using WebSocket will be built to allow bi-directional communication between the client and the server for real-time chat functionality.
- **Backend Setup**: The core backend application will be developed using **Superbase**, allowing easy API management, routing, and service integration.
- **Database Configuration**: A **relational database** (MySQL or PostgreSQL) will be configured to manage session data and temporary message storage. Proper schema design will ensure efficient data handling.

- **Frontend-Backend Communication**: By the end of Week 4, the frontend and backend should
- be successfully connected. Users should be able to initiate chat sessions, send and receive messages in real-time.

Month 2: Testing, Security Enhancements, and Deployment

Week 5: Security Enhancements and Performance Optimization

This week is focused on strengthening the application against vulnerabilities and improving its performance.

- **Message Encryption**: Implementing encryption algorithms (e.g., AES) to ensure that all messages sent over the WebSocket channel are secure and unreadable to unauthorized users.
- **Authentication & Spam Control**: Basic token-based authentication will be integrated. CAPTCHA and rate-limiting mechanisms will be applied to prevent spam or bot abuse.
- **Performance Tuning**: Backend services and database queries will be optimized for low latency and high throughput. This includes query indexing, connection pooling, and asynchronous processing where needed.

Week 6: System Testing and Debugging

This phase ensures the application functions as expected under various scenarios.

- **Load Testing**: Simulate multiple concurrent users using tools like Apache JMeter or Locust to evaluate how the application behaves under stress.
- **Integration Testing**: Ensure that all components (frontend, backend, database, WebSocket) work cohesively. All APIs and real-time features will be tested end-to-end.
- **Bug Fixing**: Identify and resolve issues found during testing. Logging mechanisms and debugging tools will be used for tracking system behavior.

Week 7: Deployment and Documentation

Once the system is stable and reliable, the focus shifts to deployment and project documentation.

- **Cloud Deployment**: The application will be deployed on cloud platforms like **AWS**, **DigitalOcean**, or **Render** using a web server such as Apache or Nginx.
- **CI/CD Integration**: Implement Continuous Integration and Continuous Deployment pipelines using tools like GitHub Actions or Jenkins to automate testing and deployment processes.
- **Documentation Preparation**: Write user guides, technical documentation, and API references. This documentation will help both users and future developers understand and use the system efficiently.

Week 8: Final Optimization and Project Submission

The final week is reserved for fine-tuning, final validation, and preparation for evaluation.

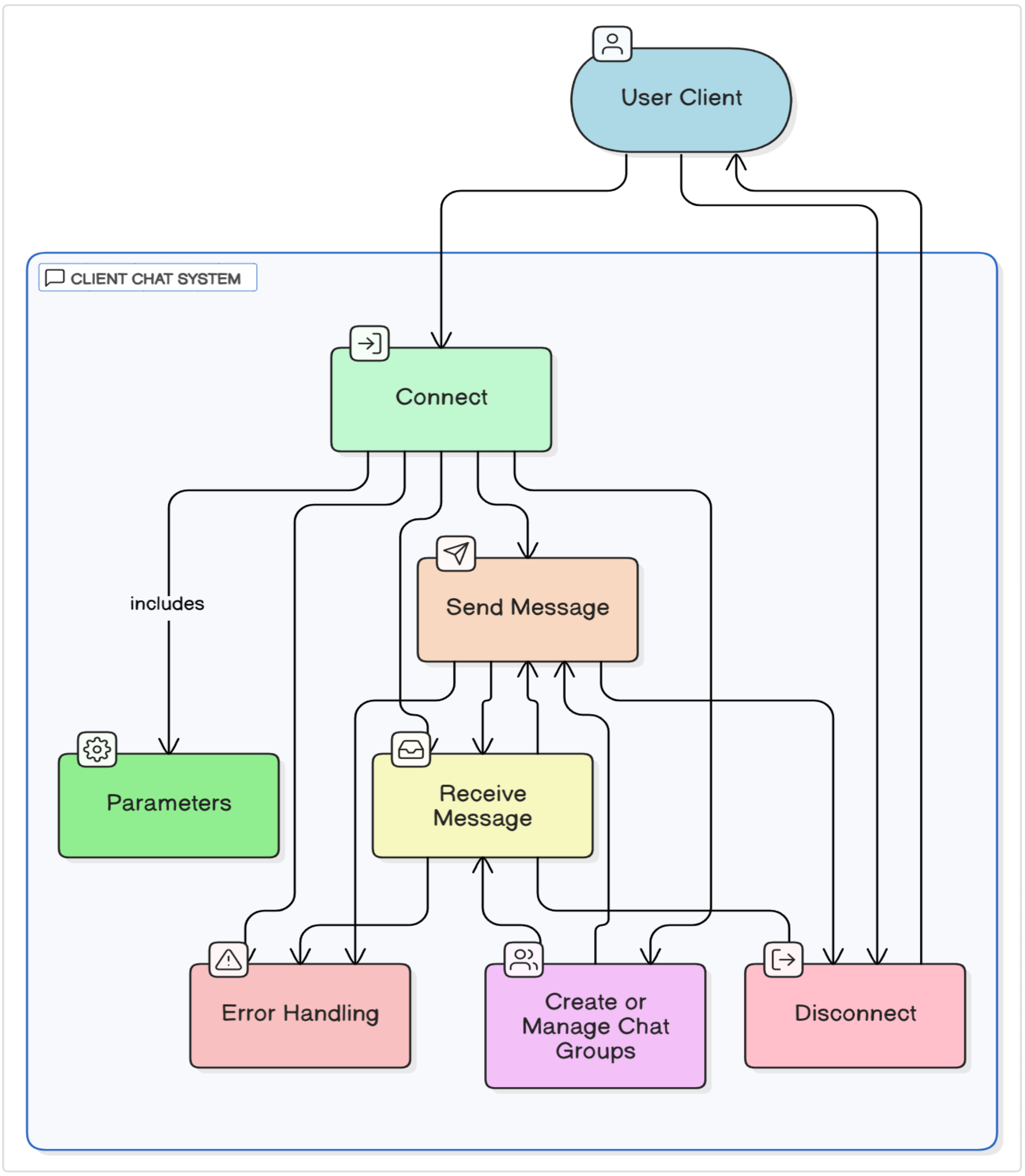
- **Quality Assurance (QA)**: Conduct final QA checks to ensure that the system is secure, stable, and performs well under real-world conditions.
- **Feedback-Based Improvements**: Any feedback received from testing sessions or mentors will be incorporated into the final version of the project.
- **Project Demonstration Readiness**: Finalize the application for the academic demonstration, ensuring that all required features are working correctly and the report is complete.

Conclusion

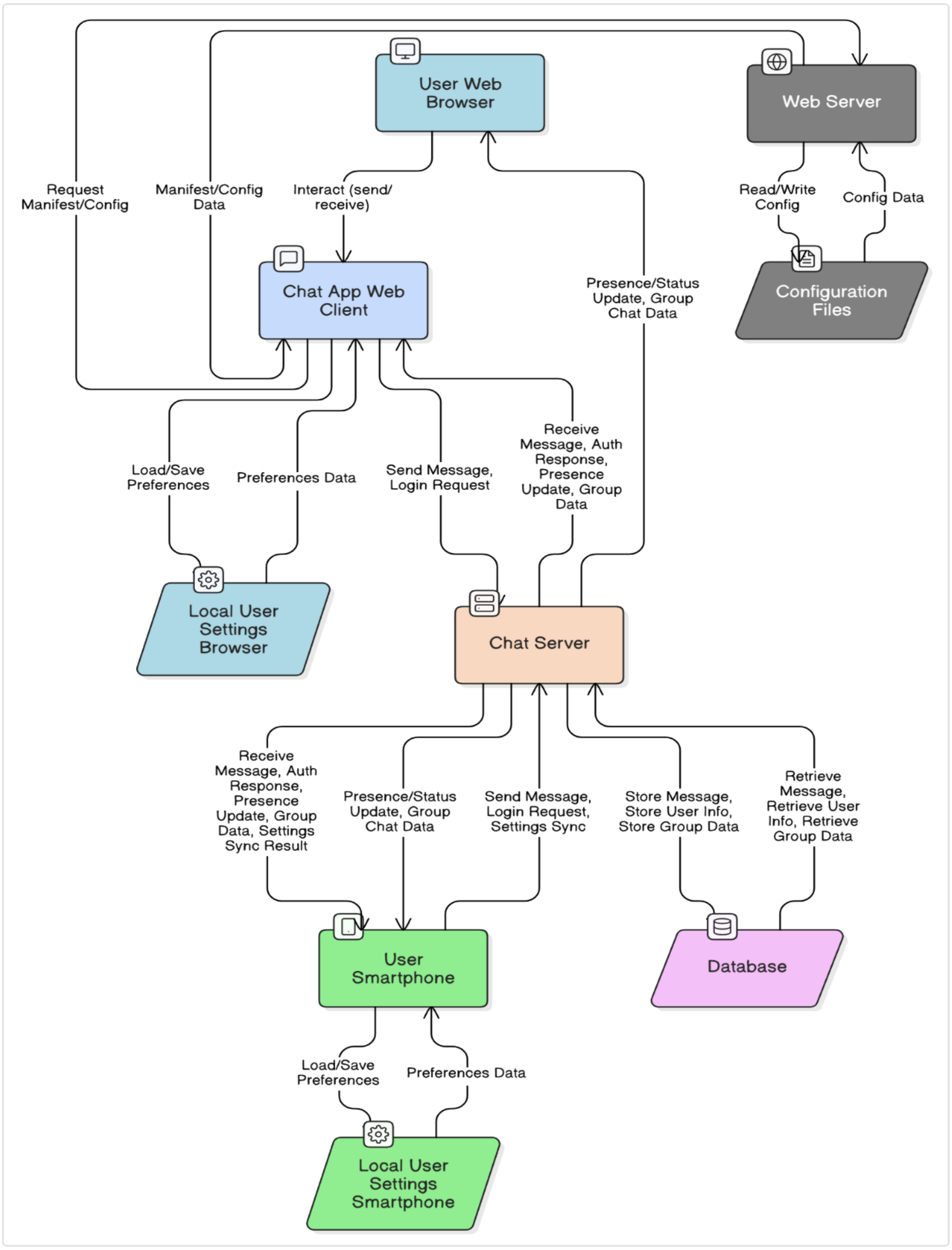
This detailed 8-week development timeline ensures that the project progresses in a logical, organized, and efficient manner. By dividing tasks across team members and following best practices in software development, **ChatFlow can be successfully designed, built, tested, and deployed** within the allocated timeframe, fulfilling both functional and academic expectations.

DIAGRAM

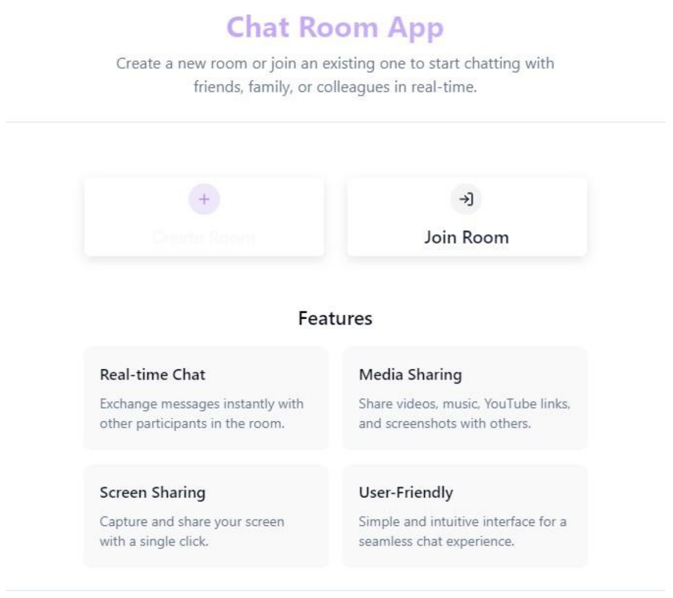
Use Case Diagram



Data Flow Diagram


User Web Browser
Web Server
Request Manifest/Config
Manifest/Config Data
Interact (send/ receive)
Read/Write Config
Config Data
Chat App Web Client
Configuration Files
Presence/Status Update, Group Chat Data
Load/Save Preferences
Preferences Data
Send Message, Login Request
Receive Message, Auth Response, Presence Update, Group Data
Local User Settings Browser
Chat Server
Receive Message, Auth Response, Presence Update, Group Data, Settings Sync Result
Presence/Status Update, Group Chat Data
Send Message, Login Request, Settings Sync
Store Message, Store User Info, Store Group Data
Retrieve Message, Retrieve User Info, Retrieve Group Data
User Smartphone
Database
Load/Save Preferences
Preferences Data
Local User Settings Smartphone

RESULTS



Chat Room App

Create a new room or join an existing one to start chatting with friends, family, or colleagues in real-time.

Create Room

Join Room

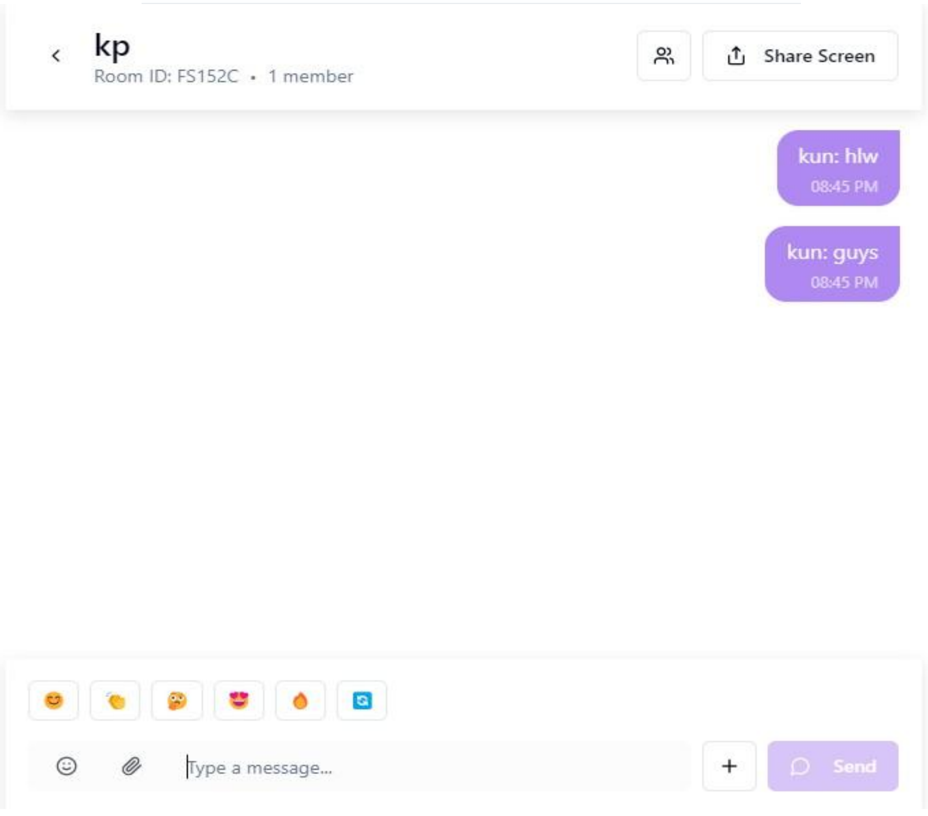
Features

Real-time Chat
Exchange messages instantly with other participants in the room.

Media Sharing
Share videos, music, YouTube links, and screenshots with others.

Screen Sharing
Capture and share your screen with a single click.

User-Friendly
Simple and intuitive interface for a seamless chat experience.



kp
Room ID: FS152C • 1 member

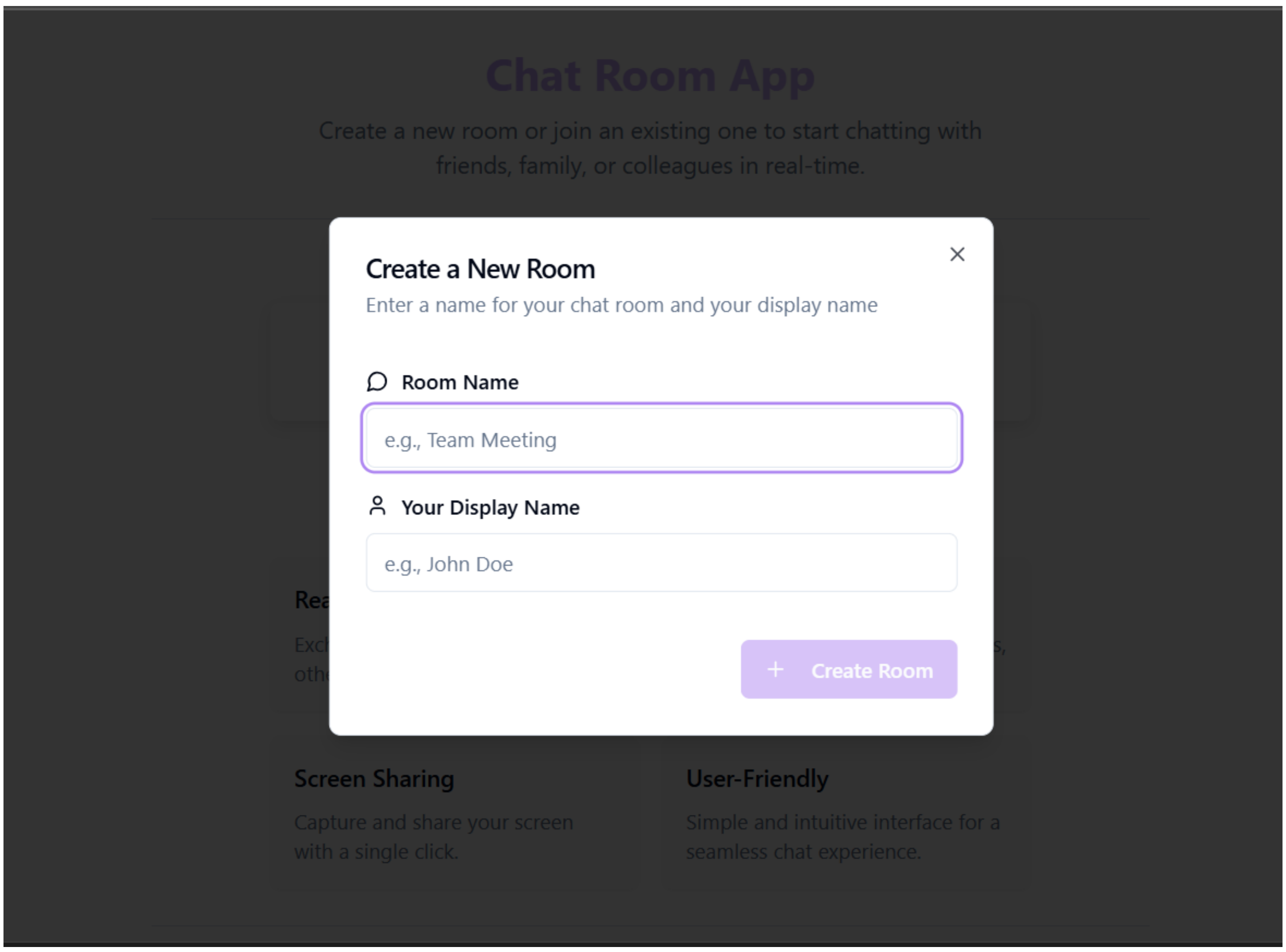
Share Screen

kun: hlw
08:45 PM

kun: guys
08:45 PM

Type a message...

Send


Chat Room App

Create a new room or join an existing one to start chatting with friends, family, or colleagues in real-time.

Create a New Room

Enter a name for your chat room and your display name

Room Name

e.g., Team Meeting

Your Display Name

e.g., John Doe

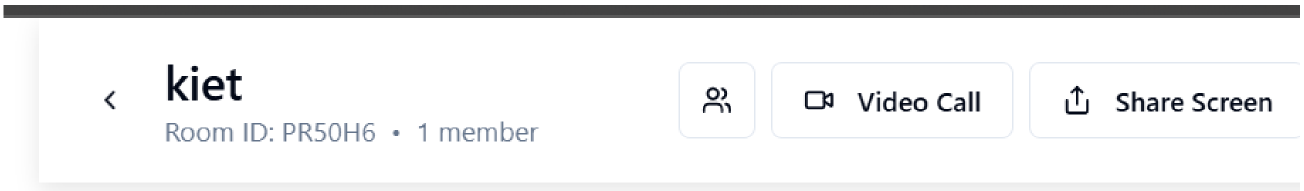
Screen Sharing

Capture and share your screen with a single click.

User-Friendly

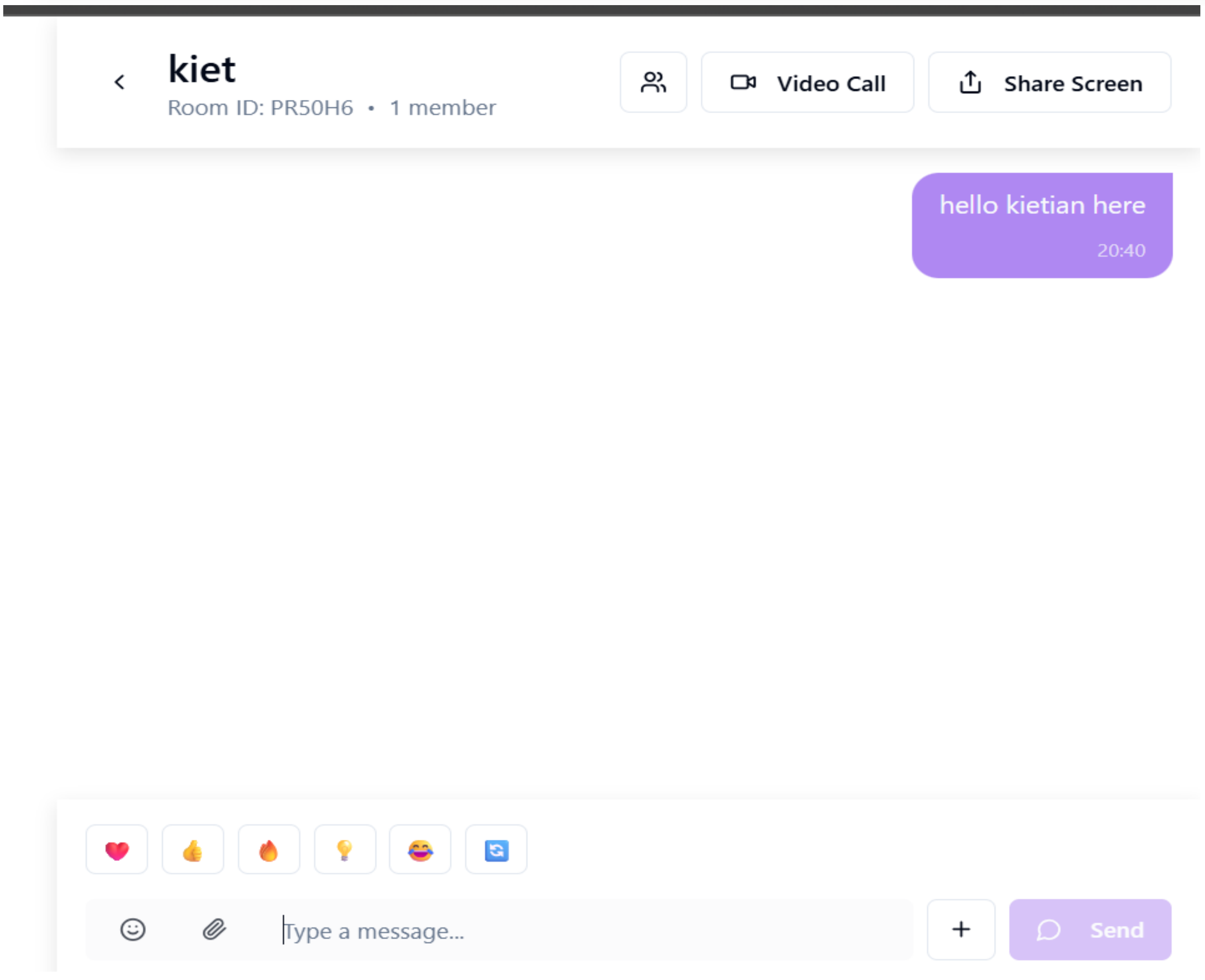
Simple and intuitive interface for a seamless chat experience.

LogIn via secure room Id



kiet

Room ID: PR50H6 • 1 member



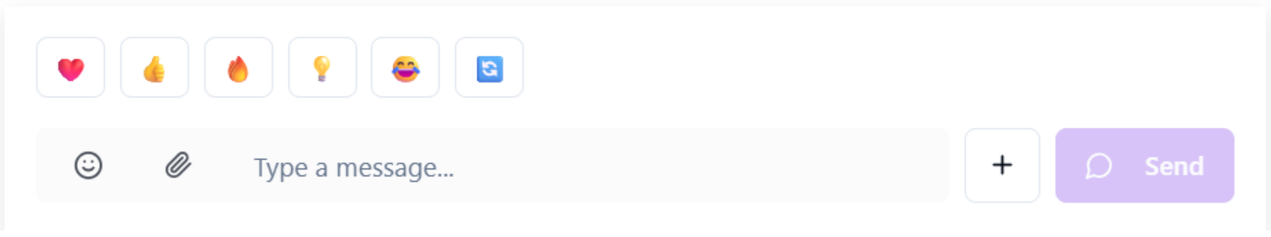
kiet

Room ID: PR50H6 • 1 member

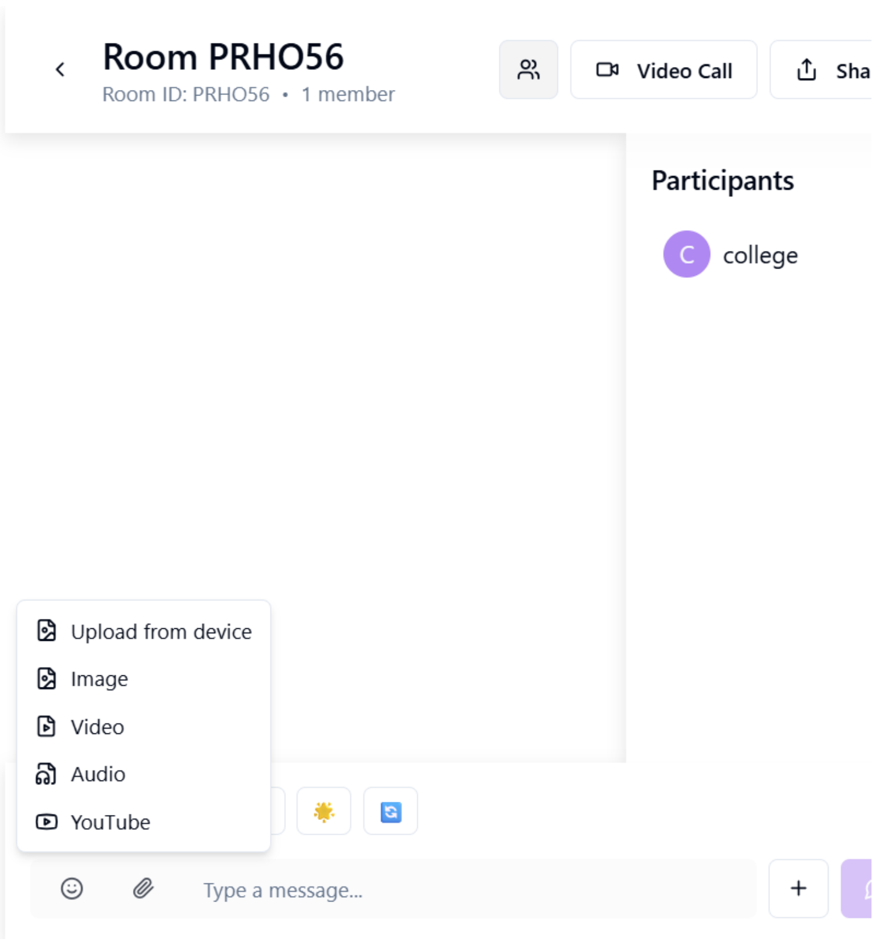
hello kietian here

20:40

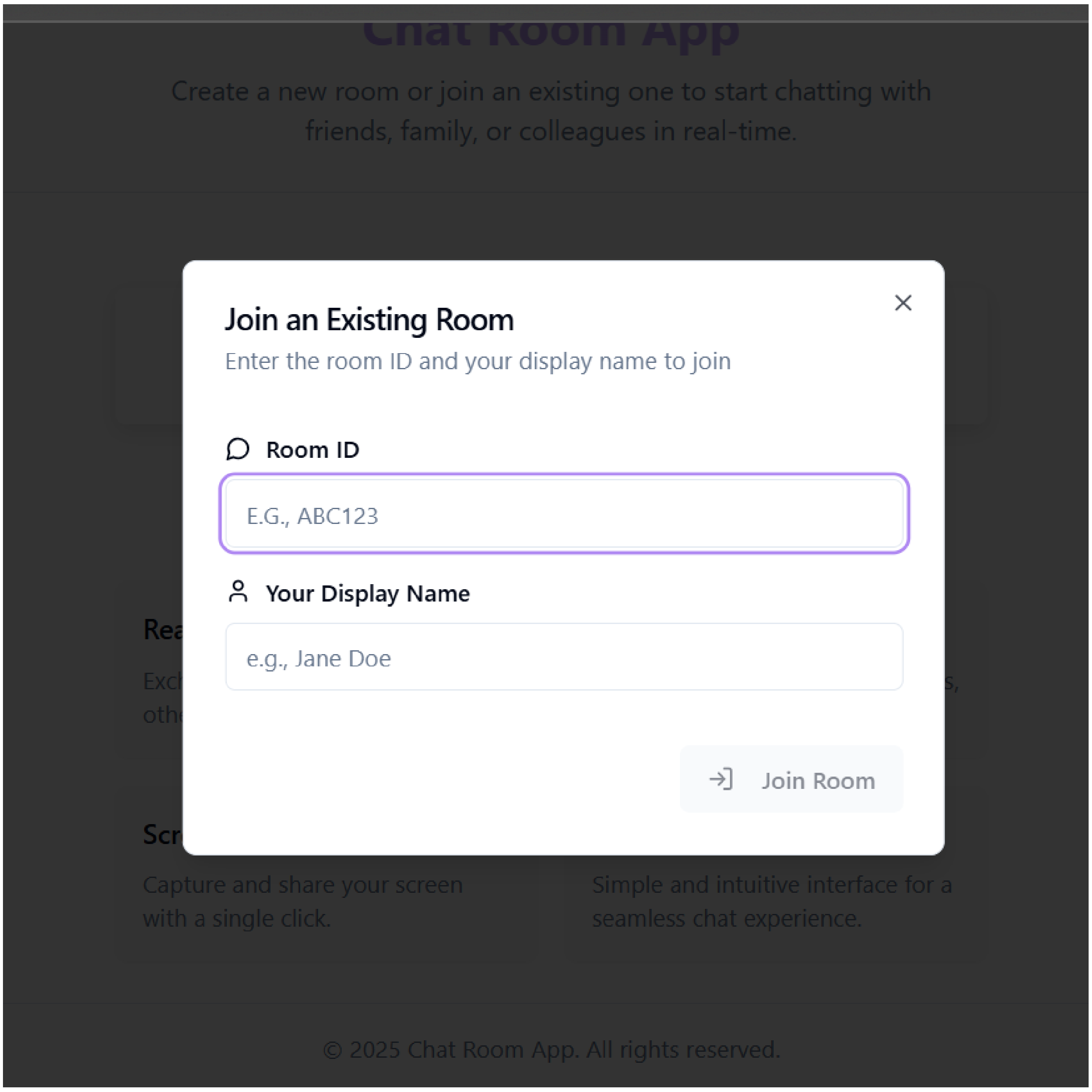
Real time chat feature:



Multimedia support:



Join a chatroom with specific id



Chat Room App

Create a new room or join an existing one to start chatting with friends, family, or colleagues in real-time.

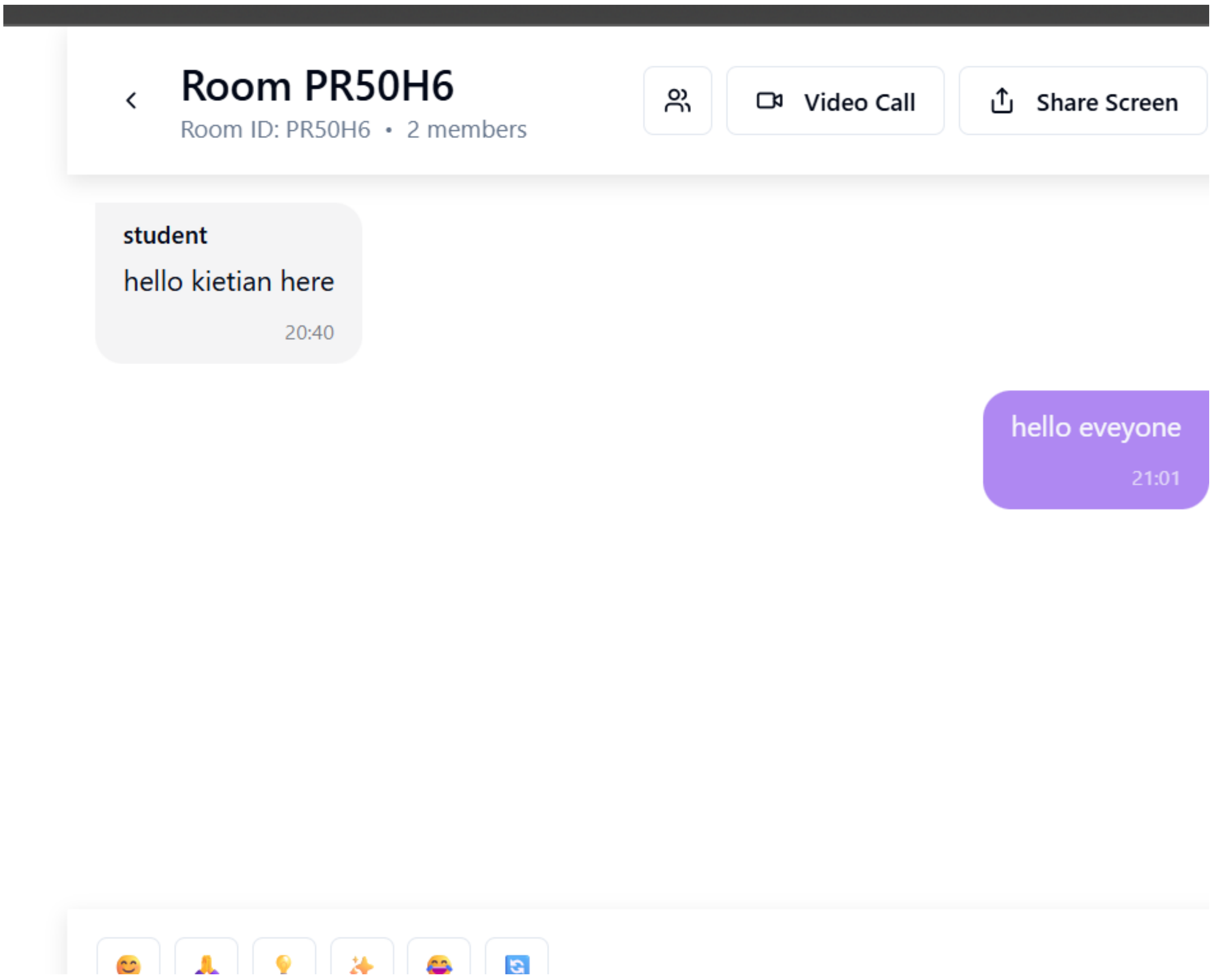
Join an Existing Room

Enter the room ID and your display name to join

Capture and share your screen with a single click.

Simple and intuitive interface for a seamless chat experience.

© 2025 Chat Room App. All rights reserved.


Room PR50H6

Room ID: PR50H6 • 2 members

student

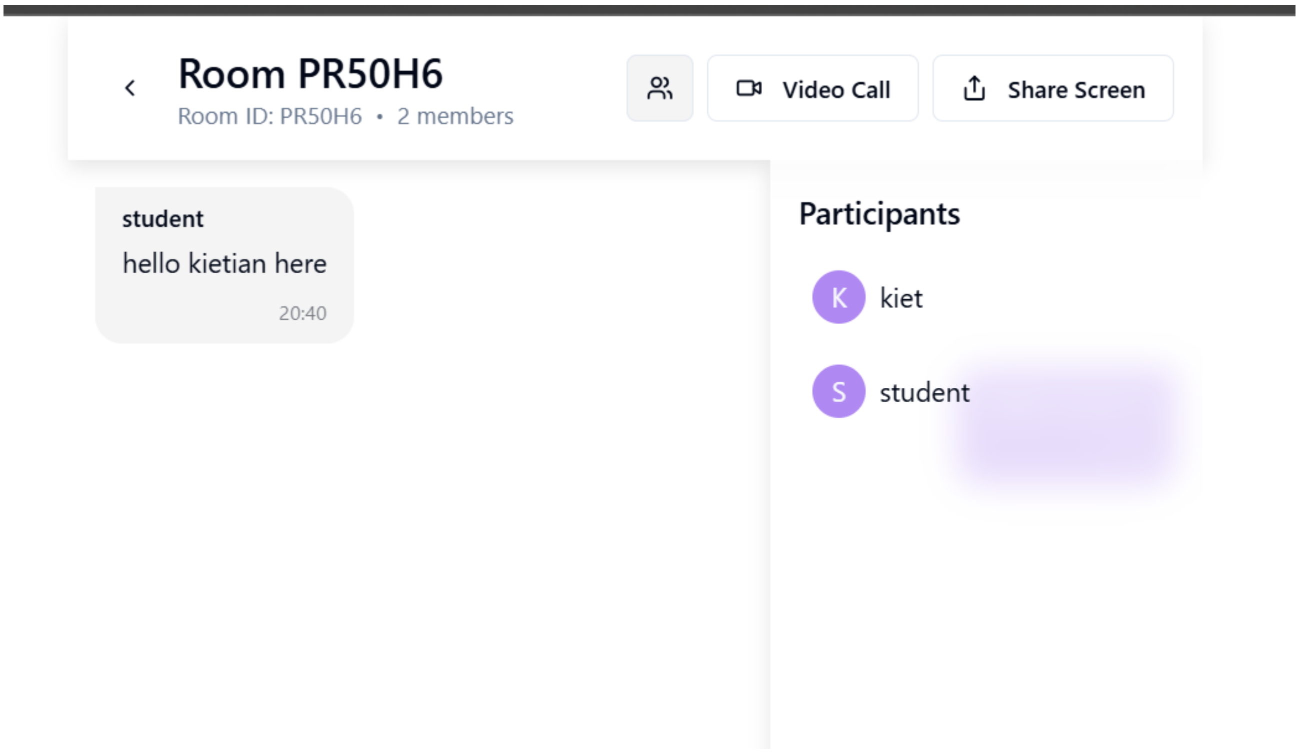
hello kietian here

20:40

hello eveyone

21:01

Check active user:



student
hello kietian here
20:40

Participants

CONCLUSION

The Chat Application project successfully demonstrates the core functionalities required for real-time communication between users over a network. Through the implementation of essential features such as user connection, message sending and receiving, and secure disconnection, the system ensures a seamless and interactive user experience. The architecture integrates both client-side and server-side components, enabling compatibility across web browsers and smartphones.

This project also highlights key aspects of modern application development, including responsive design, data flow between components, and secure user data management. By storing messages and user preferences effectively and ensuring smooth communication via a centralized chat server, the application proves to be both scalable and user-friendly.

In summary, the Chat Application serves as a reliable platform for instant messaging and provides a solid foundation for future enhancements such as group chats, media sharing, and end-to-end encryption, making it a promising solution for real-time digital communication needs.

REFERENCES

1. Smith, J., & Lee, K. (2022). Real-Time Messaging Using WebSocket: A Comparative Study. International Journal of Computer Science, 45(3), 120-135.
2. Brown, T., & Williams, D. (2021). Ensuring Privacy in Anonymous Chat Systems. Cybersecurity Journal, 10(2), 89-102.
3. Johnson, R. (2020). Building Scalable Backend Systems with Spring Boot. Software Engineering Journal, 33(4), 56-72.
4. **Spring Boot WebSocket Guide** - Official Spring documentation on implementing WebSockets: https://spring.io/guides/gs/messagingstompwebsocket/
5. **React WebSocket Integration** - MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
6. **AWS Deployment Best Practices** - https://docs.aws.amazon.com/