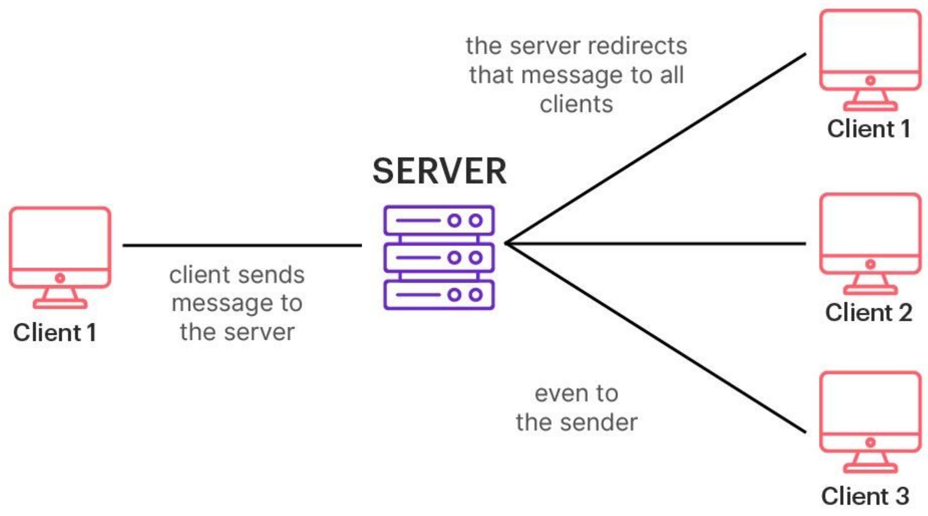
LOW LEVEL DESIGN

CHATTING APPLICATION

NAME: SAHIL PARAG KINDO



the server redirects that message to all clients
SERVER
Client 1
client sends message to the server
Client 1
Client 2
even to the sender
Client 3

cometchat

1. Client-Server Model:

The architecture should follow the client-server model, where clients (users) communicate with a central server that manages user connections, messages, and data storage.

2. Client-Side Components:

User Interface (UI): The client application presents the chat interface to users, allowing them to send and receive messages.

Client Logic: This component handles user interactions, message formatting, and communication with the server.

Communication Layer: Manages communication between the client and the server, often using WebSocket or HTTP protocols for real-time messaging.

3. Server-Side Components:

Application Server: Receives and processes client requests, manages user connections, and enforces security.

Database: Stores user profiles, message history, and other persistent data.

Message Broker: In a scalable system, a message broker like Apache Kafka or RabbitMQ can be used to manage real-time message distribution.

Authentication and Authorization: Handles user authentication and authorization, ensuring that users can only access their own data.

4. Database Design:

User Data: Store user profiles, including usernames, passwords (hashed and salted), and other relevant information.

Message History: Store messages with metadata (sender, receiver, timestamp) for chat history.

Group Data: If supporting group chats, store group information and memberships.

5. Real-Time Messaging:

Use WebSocket or similar technologies to enable real-time communication between clients and the server.

Implement message queuing for efficient message distribution to multiple clients in group chats.

6. Security:

Implement secure communication using HTTPS for web clients.

Encrypt sensitive data like passwords and messages.

Implement authentication and authorization mechanisms to protect user data.

Guard against common security threats like SQL injection and cross-site scripting (XSS).

7. Scalability:

Design the system to scale horizontally by adding more server instances or message brokers as the user base grows.

Implement load balancing to distribute client requests evenly.

8. Caching:

Use caching mechanisms to reduce database load, especially for frequently accessed data like user profiles and recent messages.

9. Logging and Monitoring:

Implement logging to record system events and errors for debugging and auditing.

Use monitoring tools to track server health, performance, and user activity.

10. Deployment:

Deploy the application on a cloud infrastructure or on-premises servers, depending on your requirements.

Ensure high availability and disaster recovery strategies.