**System Design Document: ChatApp**

1. **Introduction**

The Chat App is a real-time messaging application developed using Python with Flask for the application server, Kafka for message queuing, and MongoDB for data storage. This document outlines the system architecture, components, and key features of the Chat App.

**2. System Overview**

The Chat App enables users to exchange text messages in real-time. It includes user authentication, one-on-one messaging, group chat functionality, and real-time notifications.

**3. System Architecture**

The Chat App architecture consists of several components working together:

**3.1 Client-Side Components**

Developed using HTML, CSS, and JavaScript, these components allow users to access the chat application through web browsers.

**3.2 Server-Side Components**

**Application Server (Python with Flask):**

Handles user authentication, message processing, and business logic.

Utilizes Flask for RESTful API endpoints to communicate with clients.

**Message Queuing (Kafka):**

Uses Apache Kafka for managing real-time message delivery between clients.

Producers send messages, and consumers (server and clients) receive messages from Kafka topics.

**Database (MongoDB):**

MongoDB is used as a NoSQL database to store user profiles, message history, and chat metadata.

Provides high scalability and flexibility for storing JSON-like documents.

**3.3 Real-Time Communication**

WebSocket can be implemented for real-time, bidirectional communication between clients and the server, enhancing user experience and reducing latency.

**4. Key Features**

User Authentication: Users can create accounts and log in securely.

One-on-One Chat: Users can send text messages, images, and multimedia files to individual contacts.

Group Chat: Users can create and participate in group conversations.

Notifications: Users receive real-time notifications for new messages.

Scalability: Kafka and MongoDB enable the system to scale horizontally to handle a growing user base.

**5. Data Flow**

1. User logs in or creates a new account. Flask handles user authentication and validates user credentials against the MongoDB database.
2. If not registered, user can go to registration page to register.
3. Upon successful authentication, the client establishes a connection with the Kafka cluster. Producers send messages to Kafka topics, and consumers (server and clients) receive messages from these topics.
4. Messages are stored in MongoDB for future retrieval.
5. When the client sends a send/fetch message request, the request is passed to flask server through forms in HTML pages.
6. The request is forwarded to the action\_server which sends a query to database and retrieves the data.
7. The data will be passed from action\_server to flask server and then to the HTML pages
8. Clients receive real-time notifications and messages through WebSocket (if implemented) or regular HTTP requests to Flask API endpoints (by clicking the refresh button present in the dashboard page).

**6. Non-Functional Requirements**

**Performance:** The system should handle a large number of concurrent users, ensuring low-latency message delivery.

**Scalability:** Kafka and MongoDB provide horizontal scalability to handle increased user load and message traffic.

**Reliability:** Kafka's fault-tolerant architecture ensures message delivery even in the event of server failures.

**Security:** User data should be encrypted, and Flask should protect against common web vulnerabilities. Kafka can be configured with SSL for secure communication.

**7. Conclusion**

The Chat App leverages Python with Flask for the application server, Kafka for real-time message queuing, and MongoDB for flexible data storage. This combination of technologies provides a robust, scalable, and real-time messaging solution, offering users a seamless communication experience for users.

**Libraries used:**

1. Flask
2. Kafka
3. Pymongo
4. Json
5. Threading
6. Flask\_socketio

**Procedure to run the app:**

1. Install Kafka (if not done) and then run the zookeeper first and then the kafka in command prompt.
2. Run the action\_server with ‘python action\_server.py’ command.
3. In a separate terminal run flask server with ‘flask run’ command.
4. Go to the port given under flask run command to access the app.
5. In the app, register with a random username and login
6. You can fetch the users and groups with their respective buttons on the left side of viewport.
7. You can logout with the button on top-right corner
8. You can fetch previous messages of a chat by clicking fetch messages and refresh buttons respectively.
9. You can get updated messages list by clicking refresh button.
10. You can type new messages and send with the send button.