Chat Application Design Document

Table of Contents

1. Introduction:

This document describes the design and architecture of a chat application built using microservices architecture. It covers problem statements, requirements, architecture, detailed design and future aspects.

2. Problem statement

The goal is to to develop a real-time chat application that allows users to sign up, log in, send messages and log out. The solution used microservices , kafka for message processing, docker for containerization and for deployment GCP is used.

3. Requirements

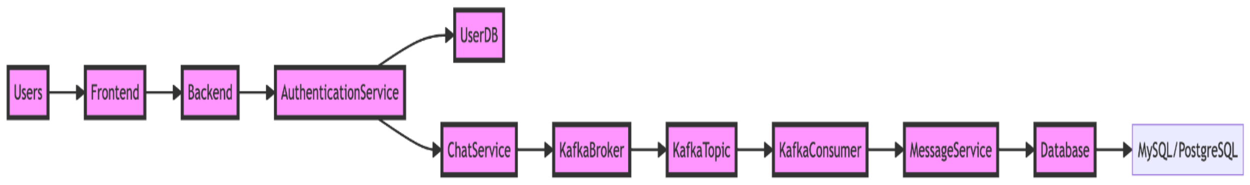
Functional Requirements

- **User Sign-Up:** Allows users to create an account and we are storing user details in the database.
- **User login :** Authenticate users
- **Real time chat:** Enable users to send and receive messages instantly.
- **Log out:** Securely log users out.

Non Functional Requirements

- **Scalability:** Supports a large number of concurrent users
- **Reliability:** Ensure message delivers even in case of failures
- **Performance:** Maintain low latency for message delivery.
- **Security:** Protect user data and ensures secure communication

4. Architecture Overview:



High Level Architecture

The architecture consists of several microservices like authentication service, chat-service, database service each responsible for specific functionality, communicating through kafka message broker and deployed using docker.

Component Interaction

- **Frontend:** A Play Framework static web app that interacts with backend API.
- **Backend API:** Handles user authentication, message routing, and exposes Restful endpoints.
- **Microservices:** Perform background tasks and message processing using the Play Framework.
- **Kafka:** It manages message queues where it sends message topics to consumers, consumer listens it and processes the task asynchronously.
- **Databases:** Stores users data and chat messages persistently.

5. Technology Stack

- Scala: Primary language with a functional approach.
- SBT: Build tool for Scala.
- Play Framework: For frontend and backend API.
- Akka: For background tasks and microservices.
- Kafka: For messaging.
- MySQL: Database.
- Docker: Containerization.
- GCP: Hosting.
- GitHub: Codebase management.

6. Detailed Design

Frontend

The frontend is a static web app built using the Play Framework. It provides user interfaces for sign-up, login, chatting, and logout.

Backend API

The backend API, built using the Play Framework, exposes RESTful endpoints for user operations and message routing. It handles requests from the frontend and communicates with microservices via Kafka.

Microservices

Microservices, implemented using the Akka framework, handle background tasks such as processing chat messages and managing user sessions.

Kafka

Kafka is used for asynchronous communication between microservices. It includes:

- Producer: Publishes messages to Kafka topics.
- Consumer: Subscribes to topics and processes messages.
- Broker: Manages topics and ensures message durability.

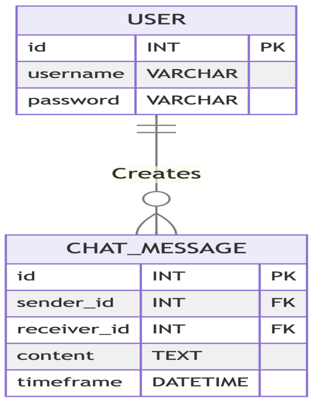
API Endpoints Explanation:

- **/login -** This endpoint is being used for login purpose
- **/chat?username=$username** - This will open the chat application for the user who has logged in
- **/logout** - It will redirect to login page again

Database design:

The database includes two main tables:

- USER: Stores user information.
- CHAT_MESSAGE: Stores chat messages.



USER		
id	INT	PK
username	VARCHAR	
password	VARCHAR	

CHAT_MESSAGE		
id	INT	PK
sender_id	INT	FK
receiver_id	INT	FK
content	TEXT	
timeframe	DATETIME	

8. Error Handling

Error handling is implemented at multiple levels:

- **Frontend**: Displays user-friendly error messages.
- **Backend API**: Returns appropriate HTTP status codes and error messages.
- **Microservices:** In our microservices architecture implemented within the Play Framework, error handling is achieved through proper design patterns and practices. Each microservice is encapsulated within a separate Play application, allowing for modularity and isolation of failures. When an error occurs within a microservice, Play's built-in mechanisms for error handling, such as exception handling and recovery strategies, are utilized to manage failures gracefully.
- **Kafka:** Ensures message delivery through retries and logging.

9. Future Enhancements

Potential enhancements include:

- Message encryption for added security.
- Notification systems for offline users.
- Integration with external services for additional features.

10. Conclusion

The designed chat application leverages a robust and scalable microservices architecture, utilizing Kafka for message processing and Docker for containerization. It ensures real-time messaging capabilities while maintaining reliability, performance, and security.