Hack to Hire 2024

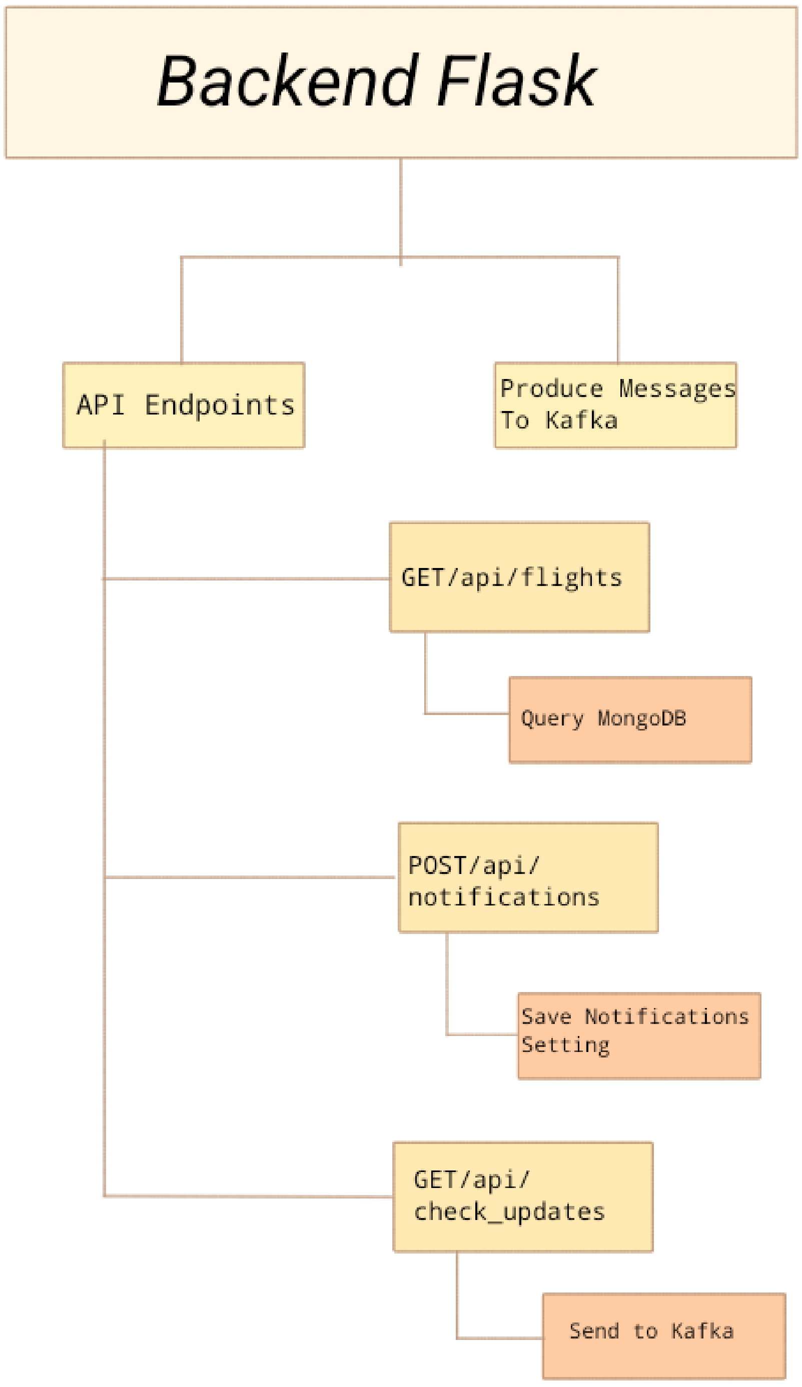
Objective : Develop a system to provide real-time flight status updates and notifications to passengers, integrating with airport systems and utilizing technologies like Kafka for push notifications.

Solution:

In this project, I set out to create a real-time flight status update and notification system for "SampleAirlines." The goal was to provide passengers with timely updates about their flights, including status changes, delays, and gate information. To achieve this, I used a combination of modern web technologies, including React.js for the frontend, Flask for the backend, MongoDB for data storage, and Apache Kafka for real-time messaging. Additionally, I explored both polling and push-based approaches to ensure real-time updates.

To solve this problem, I first considered the requirements for real-time notifications and the need for an efficient and scalable system. My initial consideration was the polling mechanism, where the client would periodically request updates from the server. While this method is straightforward and easy to implement, it can become inefficient due to the constant communication between the client and server, which can lead to unnecessary network traffic and increased server load.

To address these concerns, I decided to implement a push-based approach using Apache Kafka and WebSockets. This method allows the server to push updates to the client only when there are changes, significantly reducing network traffic and improving efficiency. Kafka, a high-throughput distributed messaging system, was used to handle real-time updates, ensuring that messages are delivered promptly and reliably. WebSockets provided a persistent connection between the client and server, enabling instant communication and real-time updates.


Backend Flask
API Endpoints
Produce Messages To Kafka
GET/api/flights
Query MongoDB
POST/api/notifications
Save Notifications Setting
GET/api/check_updates
Send to Kafka

Frontend Implementation

The frontend was developed using React.js to create a dynamic and responsive user interface. I structured the application into several key components:

- Header: This component includes the search bar and location permissions.
- Offers: This component displays current offers and promotions.
- Services: This component lists the various services provided by the airline.
- FlightDashboard: This component shows flight information and notifications.
- NotificationSettings : This component allows users to enter their email and phone number for receiving notifications.

To ensure the UI was user-friendly and visually appealing, I applied CSS styling across all components. The `FlightDashboard` component was designed to fetch flight data from the backend and update the state and UI with the latest information. This component also established a WebSocket connection to receive real-time updates.

Backend Implementation

The backend was implemented using Flask, a lightweight and flexible web framework for Python. I set up API endpoints to handle requests from the frontend:

- GET /api/flights: This endpoint queries the MongoDB database for the latest flight data and returns it to the client.
- POST /api/notifications: This endpoint saves user notification settings to the database.
- GET /api/check_updates: This endpoint simulates checking for updates and sends data to Kafka.

I configured Flask to connect to a MongoDB database using the `Flask-PyMongo` library. The MongoDB database stores flight information, including flight number, origin, destination, status, and gate. I populated the database with sample data for Indigo Airlines' domestic flights in India using MongoDB Compass.

Real-Time Messaging with Kafka

To handle real-time updates, I integrated Apache Kafka into the backend. Kafka served as a message broker, managing the flow of messages between the server and the clients. I created a Kafka topic named `flight_updates` to handle flight status updates.

In the Flask application, I set up a Kafka producer to send messages to the `flight_updates` topic whenever there were changes in flight status. On the client side, I created a WebSocket server to consume messages from Kafka and push them to the frontend. This server maintained connections with multiple clients, ensuring that all connected users received updates simultaneously.

Polling Mechanism

Initially, I considered a polling mechanism where the client would send requests to the server every second to check for updates. This approach involved the following steps:

- The `FlightDashboard` component used JavaScript's `setInterval` function to make periodic API calls to the backend.
- The server queried the MongoDB database for any new updates and returned them to the client.
- The frontend updated its state and re-rendered the UI with the latest data.

While polling ensures regular updates, it can be inefficient due to the constant requests, leading to increased network traffic and server load. This inefficiency prompted me to explore the push-based approach.

Push-Based Approach with Kafka and WebSockets

The push-based approach provided several advantages over polling:

- Efficiency: Updates are sent to the client only when there are changes, reducing unnecessary network traffic and server load.
- Real-Time Updates: Users receive notifications instantly as changes occur, enhancing the user experience.
- Scalability: Kafka handles high-throughput messaging efficiently, making it suitable for applications with many users.

By using Kafka and WebSockets, I ensured that the system could handle real-time updates efficiently and scale as the number of users increased.

Through this project, I demonstrated the benefits of combining modern web technologies to build a robust and efficient real-time flight status update and notification system. By implementing both polling and push-based approaches, I gained insights into their respective advantages and drawbacks. The push-based approach using Kafka and WebSockets proved to be more efficient and scalable, providing real-time updates with minimal latency. This project showcases how leveraging different technologies can deliver an optimal user experience and meet the demands of real-time applications.This detailed explanation of the system design, approach, and implementation provides a comprehensive understanding of how I tackled the problem and developed a solution for real-time flight status updates and notifications for "SampleAirlines."