Microservices Overview

Our travel web application use the following tech stack -

1. ReactJS - Frontend
2. NodeJS - Backend
3. SQLite - Database

Microservice architecture is an approach of developing software applications through modular components or services. Here are three common microservice architectures that we considered for our project:

1. **API Gateway with Microservices:**
    - Description: In this architecture, there is an API Gateway that handles client requests and routes them to various microservices responsible for different functionalities. Each microservice handles a specific aspect of the application (e.g., flight booking, coupon management, user information, analytics) and communicates with the database as needed.
    - Advantages: This architecture provides a clear separation of concerns, allowing the application to independently scale and update each microservice. It's a good fit for complex applications with various features.
    - Challenges: Managing inter-service communication and handling API versioning can be challenging.
2. **Event-Driven Architecture:**
    - Description: In this architecture, microservices communicate through events or messages. When an event occurs, such as a flight booking, an event is published and consumed by relevant microservices (e.g., coupon validation or user notification). A message broker like RabbitMQ or Apache Kafka can be used to manage events.
    - Advantages: Event-driven architecture is highly scalable and decoupled, making it suitable for applications with real-time or asynchronous requirements.
    - Challenges: It may introduce complexity due to event handling, and debugging can be more challenging.
3. **BFF (Backend for Frontend):**
    - Description: BFF is an architecture where specific backend services are tailored for the needs of each frontend client. In our case, we would have a separate backend for the web application built with ReactJS. These backends can communicate with a shared set of microservices.
    - Advantages: BFF allows you to optimize the backend services for each client, providing a tailored experience. It simplifies frontend-backend communication.
    - Challenges: Managing multiple backends can be more complex, and we need to ensure proper coordination between backends and shared microservices.

However, for our application, we have decided not to use a microservices approach, instead deciding to go for the client-server architecture. Listed below are the factors that motivated this decision:

1. **Project Size and Complexity:** Microservices are typically more beneficial for large and complex applications where various components need to scale and evolve independently. For a smaller project with straightforward requirements such as ours, a monolithic architecture is more straightforward and easier to manage.
2. **Team Expertise:** Building and managing microservices requires a good understanding of distributed systems and cloud infrastructure.Since our team does not have experience with microservices, it doesn’t make sense to use them else it may lead to unnecessary complexity and challenges.
3. **Operational Overhead:** Microservices come with operational complexity where we'll need to manage services, communication, and orchestration tools.
4. **Time Constraints:** Microservices can be more time-consuming to implement, given the additional infrastructure and maintenance requirements.