High Level Design Document
Swiggy (Food App)

Version: 1.0

Updated: 28/02/2024

Author: Suryateja Kethavarapu

Table of Contents:

1. Introduction
2. System Overview
    2.1 User Interface
    2.2 Backend Services
    2.3 Restaurant Partners
    2.4 Delivery Partners
    2.5 Payment Gateway
3. System Architecture
    3.1 Frontend Layer
    3.2 Backend Layer
    3.3 Data Layer
    3.4 Infrastructure Layer
4. Security Measures
5. Scalability
6. Conclusion

1. Introduction:

This Application is a leading online food delivery platform that connects users with a wide range of restaurants, allowing them to order food from the comfort of their homes. This document outlines the high-level system architecture of Food Application, detailing its various components and their interactions.

2. System Overview:

System consists of several interconnected components working together to facilitate seamless food ordering, delivery, and payment processes. These components include:

2.1 User Interface:

- The user interface comprises web and mobile applications available for both iOS and Android platforms. These interfaces allow users to browse restaurants, view menus, place orders, track deliveries, and make payments.

2.2 Backend Services:

- The backend services form the core of the Food Application system. They manage user authentication, order processing, restaurant management, delivery assignment, payment processing, and other essential functionalities. These services are built using microservices architecture for scalability and maintainability.

2.3 Restaurant Partners:

- Food Application collaborates with various restaurant partners whose menus are listed on the platform. Integration with restaurant systems allows real-time updates on menu items, pricing, and availability.

2.4 Delivery Partners:

- Delivery partners are responsible for picking up orders from restaurants and delivering them to customers.

- This app assigns delivery partners based on proximity and availability, optimizing delivery times.

2.5 Payment Gateway:

- This app integrates with multiple payment gateways to facilitate secure transactions.

- Users can choose from various payment options, including credit/debit cards, digital wallets, and cash on delivery.

3. System Architecture:

The system architecture of this app is designed to be highly scalable, fault-tolerant, and efficient. It follows a distributed architecture with the following key components:

3.1 Frontend Layer:

- Web Application: Built using modern web technologies like React.js, the web application provides an intuitive interface for users to interact with services.

- Mobile Applications: Native iOS and Android apps developed using Swift and Kotlin respectively, offering a seamless experience tailored to mobile devices.

3.2 Backend Layer:

- API Gateway: Acts as a single entry point for all client requests, routing them to appropriate microservices.

- Authentication Service: Handles user authentication and authorization, ensuring secure access to services.

- Order Service: Manages the lifecycle of orders, including order placement, tracking, and cancellation.

- Delivery Service: Assigns delivery partners to orders, optimizes delivery routes, and tracks delivery statuses.

- Payment Service: Integrates with payment gateways to process transactions securely and efficiently.

- Restaurant Service: Manages restaurant data, including menus, availability, and pricing.

- Notification Service: Sends real-time notifications to users regarding order status updates, promotions, and other relevant information.

3.3 Data Layer:

- Database: Utilizes a combination of relational (e.g., PostgreSQL) and NoSQL (e.g., MongoDB) databases to store user data, order information, restaurant details, and other relevant data.

- Caching Layer: Implements caching mechanisms using technologies like Redis to improve system performance and reduce database load.

3.4 Infrastructure Layer:

- Cloud Infrastructure: Hosted on a cloud platform like AWS or Google Cloud, providing scalability, reliability, and flexibility.

- Containerization: Docker containers are used to package and deploy microservices, ensuring consistency across different environments.

- Orchestration: Kubernetes orchestrates containerized deployments, automating scaling, deployment, and management tasks.

4. Security Measures:

Ensuring the security of user data, transactions, and platform integrity. The following security measures are implemented:

- Encryption: Data transmission between client devices and backend servers is encrypted using industry-standard protocols such as HTTPS to prevent unauthorized access.

- Authentication and Authorization: Robust authentication mechanisms, including multi-factor authentication where necessary, are employed to verify user identities. Authorization policies restrict access to sensitive data and functionalities based on user roles.

- Secure Payment Processing: Payment information is handled securely using tokenization and encryption techniques. Compliance with PCI-DSS standards ensures the protection of financial data.

5. Scalability and Performance:

As a user base and order volume continue to grow, scalability and performance are critical considerations.

- Load Balancing: Load balancers distribute incoming traffic across multiple instances of backend services, ensuring optimal resource utilization and preventing overloading of any single component.

- Performance Monitoring: Monitoring tools track system performance metrics in real-time, enabling proactive identification and resolution of performance.

6. Conclusion:

The Swiggy system architecture outlined in this document illustrates a robust and scalable platform designed to meet the demands of online food delivery services. By modern technologies and best practices, Swiggy continues to provide a seamless and delightful experience for its users and partners alike.