**HIGH LEVEL DESIGN DOCUMENT**

**ZOMATO APPLICATION**

**Document Version:** 1.0

**Date:** 24/02/2024

**Prepared By:** Vajjala Srinandhini

Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Author** | **Changes** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**TABLE OF CONTENTS**

1. **Introduction**

* Why this HLD?
* Scope
* Definitions
* Overview

1. **Features**
2. **Architecture Overview**
   * Technology Stack
   * Design Principles
   * Scalability Considerations
   * Activity Diagram
3. **Component Details**
   * Authentication Service
   * Restaurant Service
   * Order Service
   * Payment Gateway Service
   * Review Service
4. **Data Storage**
   * User Profiles
   * Restaurant Data
   * Orders and Transactions
   * Reviews and Ratings
5. **Security Considerations**
   * Data Encryption
   * Authentication Mechanisms
   * Authorization Policies
   * Secure Payment Processing
6. **Deployment Strategy**
   * Containerization with Docker
   * Orchestration with Kubernetes
   * CI/CD Pipeline
7. **Conclusion**
8. **Introduction**

* Why this HLD document?

This High-Level Design (HLD) document is created to provide a structured plan and overview of the architecture, features, and key components of a Zomato application. The HLD document plays a crucial role in guiding the development process, aligning stakeholders, mitigating risks, ensuring compliance and security, and ultimately, delivering a successful Zomato-like application that meets the needs and expectations of its users.

* Scope

This HLD documentation is comprehensive, covering various aspects essential for the successful design, development, and deployment of a Zomato application. It provides a holistic view of the application's architecture, features, data management, security, and deployment strategy, serving as a guide for the development team and stakeholders involved in the project.

* Definitions
* High Level Design (HLD) – A document that provides an overview of the architecture, features and key components of a software system.
* Microservices architecture – An architectural style that structures an application as a collection of loosely coupled services, each encapsulating a specific business function.
* Containerization – Process of encapsulating an application into containers which can be deployed across different environments.
* Continuous Integration/ Continuous Deployment (CI/CD) – These pipelines automate the build, test and deployment processes, enabling the rapid and reliable software delivery.
* Relational Databases – Organizes data into tables (rows and columns) with pre-defined relation between them.
* NoSQL Database - A non-relational database that provides flexible data models and scalability for handling large volumes of unstructured or semi-structured data.
* Authentication – The process of verifying the identity of users or systems accessing a software application.
* Authorization – The process of granting or denying access to specific resources or functionalities based on authenticated user’s permissions.
* Payment Gateway Integration – The integration of third-party payment processing services into a software application to facilitate secure and convenient payment transactions.
* Overview

The HLD will:

* Present all of the design aspects and define them in detail
* Describe the user interface being implemented
* Describe the hardware and software interfaces
* Describe the performance requirements
* Include design features and the architecture of the project
* List and describe the non-functional attributes like:

1. Scalability
2. Reliability
3. Security
4. Performance
5. Usability
6. Compliance
7. Maintainability
8. Monitoring and Logging
9. **Features**
10. **User Authentication and Profiles:** Enable users to create accounts, log in securely, and manage their profiles, including personal information, payment details, and order history.
11. **Restaurant Listings and Search:** Provide a vast database of restaurants categorized by cuisine, location, and user ratings, allowing users to discover dining options that match their preferences.
12. **Order Placement and Tracking:** Facilitate the seamless process of placing orders, customizing items, and tracking the status of deliveries in real-time, ensuring transparency and convenience for users.
13. **Payment Gateway Integration:** Integrate with leading payment gateways to facilitate secure and hassle-free transactions, safeguarding user data and ensuring smooth payment processing.
14. **Review and Rating System:** Empower users to share their dining experiences by leaving reviews and ratings for restaurants, fostering transparency, and enabling others to make informed decisions.
15. **Architectural Overview**

The Zomato application will be built on a robust microservices architecture, leveraging modern technologies and design principles to ensure scalability, reliability, and performance. The architecture will comprise various backend services responsible for different aspects of the application, each encapsulating specific functionality and interacting with others through well-defined APIs.

1. **Technology Stack:**

The technology stack refers to the combination of programming languages, frameworks, libraries, databases, and other tools used to build the Zomato-like application. Here's how each component of the technology stack might be utilized:

1. **Backend Technologies:**
   * **Programming Language:** Java
   * **Framework:** Spring Boot (for Java) which provides a robust foundation for building RESTful APIs and handling business logic.
   * **Database:** A combination of relational ( MySQL) and NoSQL (MongoDB) databases for storing different types of data efficiently.
2. **Frontend Technologies:**
   * **JavaScript Framework:** Angular for building interactive and responsive user interfaces.
3. **Containerization and Orchestration:**
   * **Containerization Platform:** Docker for packaging applications and their dependencies into containers, ensuring consistency across development, testing, and production environments.
   * **Orchestration Platform:** Kubernetes for automating deployment, scaling, and management of containerized applications, providing resilience and scalability.
4. **Design Principles:**

The architectural design of the Zomato application adheres to several key principles to ensure modularity, scalability, and maintainability:

* 1. **Microservices Architecture:** Decomposes the application into smaller, loosely coupled services, each responsible for specific business functions. This promotes modularity, agility, and independent deployment of services.
  2. **Service-Oriented Design:** Services are designed around specific business capabilities and expose well-defined APIs for interaction with other services and clients. This fosters reusability, flexibility, and separation of concerns.
  3. **API-First Approach:** Design APIs before implementing functionality, ensuring consistency, interoperability, and ease of integration with frontend and third-party systems.
  4. **Fault Tolerance and Resilience:** Services are designed to handle failures gracefully, with built-in redundancy, retry mechanisms, and circuit breakers to maintain system stability and availability.

1. **Scalability Considerations:**

Scalability is a critical aspect of the Zomato application's architecture, enabling it to handle increasing user demand and workload efficiently. Here's how scalability is addressed:

1. **Horizontal Scaling:** Services are designed to scale horizontally by adding more instances or replicas to distribute load and handle increased traffic effectively. This allows the application to scale out across multiple servers or containers.
2. **Load Balancing:** Load balancers are employed to evenly distribute incoming requests across multiple instances or replicas of services, ensuring optimal resource utilization and preventing overload on individual components.
3. **Caching Mechanisms:** Caching is used to reduce latency and improve performance by storing frequently accessed data in memory or distributed caches (e.g., Redis). This helps offload the backend systems and handle spikes in traffic more efficiently.
4. **Activity Diagram:**

A diagram of a software application

Description automatically generated

* User Interface: The frontend layer where users interact with the application.
* API Gateway: Acts as a single entry point for client applications to access backend services.
* Microservices: Backend services responsible for specific functionalities like authentication, restaurant management, order handling, payment processing, etc.
* Authentication Service: Handles user authentication and profile management.
* Restaurant Service: Manages restaurant data including listings, menus, and reviews.
* Order Service: Facilitates order placement, tracking, and fulfillment.
* Payment Gateway Service: Integrates with third-party payment gateways for secure payment processing.

1. **Component Details**
2. **Authentication Service:** Manages user authentication, registration, and profile management, ensuring secure access to the platform.
3. **Restaurant Service:** Curates and maintains a comprehensive database of restaurants, menus, and location data, enabling users to explore dining options effortlessly.
4. **Order Service:** Handles the end-to-end order lifecycle, including order placement, customization, and delivery tracking, to provide users with a seamless ordering experience.
5. **Payment Gateway Service:** Integrates with leading payment gateways to facilitate secure and efficient payment processing, supporting various payment methods and ensuring compliance with industry standards.
6. **Review Service:** Enables users to submit reviews and ratings for restaurants, implements moderation mechanisms to maintain quality, and fosters a vibrant community of diners.
7. **Data Storage**
8. Data storage for the Zomato application involves a combination of relational and NoSQL databases to accommodate various data types and access patterns efficiently.
9. Relational databases are suitable for structured data requiring ACID compliance and complex queries, while NoSQL databases are ideal for semi-structured and rapidly changing data.
10. Schema design, normalization, denormalization, and indexing techniques are employed to ensure data integrity, optimize query performance, and provide a seamless user experience.

**1. User Profiles:**

* **Data Model:**
  + User profiles include attributes such as username, email, password (hashed), profile picture, contact information, and preferences (e.g., favorite cuisines, dietary restrictions).
* **Storage:**
  + User profile data can be stored in a relational database (e.g., PostgreSQL or MySQL) with a dedicated table for users. Each row in the table represents a user, and columns correspond to user attributes.
* **Normalization:**
  + User data may be normalized to avoid redundancy and maintain data integrity. For instance, separate tables can be created for user information, contact details, and preferences, linked by foreign key relationships.
* **Indexes:**
  + Indexes can be applied to columns frequently used for querying, such as username or email, to improve search performance.

**2. Restaurant Data:**

* **Data Model:**
  + Restaurant data includes attributes such as restaurant name, location (address, coordinates), cuisine type, menu items, opening hours, and ratings.
* **Storage:**
  + Restaurant data can be stored in a combination of relational and NoSQL databases.
  + Relational databases can be used for structured data (e.g., restaurant details, ratings), while NoSQL databases (e.g., MongoDB) can be used for semi-structured data like menu items and reviews.
* **Schema Design:**
  + Relational tables can be designed to store basic restaurant information, while MongoDB collections can store nested documents for menu items and reviews.
* **Denormalization:**
  + Denormalization techniques may be applied to improve query performance for frequently accessed data, such as caching restaurant details and popular menu items.

**3. Orders and Transactions:**

* **Data Model:**
  + Orders and transactions include attributes such as order ID, user ID, restaurant ID, order items, total amount, status (e.g., pending, delivered), and timestamps.
* **Storage:**
  + Order data can be stored in a relational database, with dedicated tables for orders, order items, and transactions.
* **Normalization:**
  + Normalization can be applied to avoid data redundancy and ensure consistency. Separate tables can store order details, item details, and transaction information, linked by foreign key relationships.
* **Indexes:**
  + Indexes can be created on columns used for filtering and sorting orders, such as order date or restaurant ID, to improve query performance.

**4. Reviews and Ratings:**

* **Data Model:**
  + Reviews and ratings include attributes such as review ID, user ID, restaurant ID, rating (e.g., 1-5 stars), review text, and timestamps.
* **Storage:**
  + Review data can be stored in a NoSQL database like MongoDB, which allows for flexible schema design and efficient storage of semi-structured data.
* **Schema Design:**
  + MongoDB collections can store documents representing reviews, with fields for user ID, restaurant ID, rating, review text, and timestamps.
* **Indexes:**
  + Indexes can be applied to fields used for filtering and sorting reviews, such as restaurant ID or timestamp, to optimize query performance.

DATABASE DESIGN:

A diagram of a diagram

Description automatically generated

### ****5. Security Considerations****

Security will be paramount in the design and implementation of the Zomato-like application, with measures in place to protect user data, authenticate users securely, and ensure secure payment processing. Encryption, authentication mechanisms, authorization policies, and compliance with industry standards such as PCI DSS will be implemented to safeguard user information and maintain the integrity of the platform.

### ****Deployment Strategy****

The application will be deployed using containerization with Docker for packaging and Kubernetes for orchestration, enabling seamless scaling, deployment, and management of containerized microservices. Continuous Integration/Continuous Deployment (CI/CD) pipelines will be implemented to automate the deployment process, ensuring rapid delivery of updates and enhancements to the platform.

### ****Conclusion****

The Zomato-like application represents a groundbreaking solution to meet the evolving needs of modern consumers in the digital food delivery landscape. By offering a comprehensive platform with user-friendly features, robust architecture, and stringent security measures, the application aims to redefine the dining experience and become the go-to destination for users seeking culinary delights at their fingertips.