**Design Document: Food Delivery Microservice**

**1. Introduction**

This document outlines the design of a food delivery microservice built using Java, Spring Boot, and a message queue (RabbitMQ). The microservice is responsible for handling the core processes involved in placing, processing, and tracking food orders. It is designed for scalability, maintainability, and ease of integration with other systems.

**2. Goals**

* Provide a reliable and efficient system for processing food orders.
* Ensure scalability to handle varying traffic loads.
* Maintain data consistency across distributed components.
* Enable seamless integration with external services (e.g., payment gateways, delivery providers).

**3. High-Level Architecture Diagram**

HighLevel Architecture Diagram of the Food Delivery Microservice

The diagram should illustrate the following components and their interactions:

* **API Gateway:** Entry point for external clients (web, mobile apps). Routes requests to appropriate microservices.
* **Order Microservice:** Handles order placement, status updates, and communication with restaurants.
* **Restaurant Microservice:** Manages restaurant information, menus, and order preparation.
* **Customer Microservice:** Manages customer information and order history.
* **Message Queue (RabbitMQ/Kafka):** Facilitates asynchronous communication between microservices.
* **Database:** Stores data for orders, customers, restaurants, and menu items.

**4. Microservice Details**

**4.1. Order Microservice**

* **Responsibilities:**
  + Receives order placement requests from the API Gateway.
  + Validates order data and checks for restaurant availability.
  + Persists order information in the database.
  + Sends order notifications to restaurants via the message queue.
  + Updates order status based on events from restaurants and delivery providers.
* **API Endpoints (Examples):**
  + POST /orders: Place a new order.
  + GET /orders/{orderId}: Get order details.
  + PUT /orders/{orderId}/status: Update order status.

**4.2. Restaurant Microservice**

* **Responsibilities:**
  + Manages restaurant profiles, menus, and operating hours.
  + Receives order notifications from the message queue.
  + Updates order status (e.g., accepted, preparing, ready).
  + Sends notifications to the Order Microservice when an order is ready.
* **API Endpoints (Examples):**
  + GET /restaurants: Get a list of restaurants.
  + GET /restaurants/{restaurantId}/menu: Get a restaurant's menu.

**4.3. Customer Microservice**

* **Responsibilities:**
  + Manages customer profiles and order history.
  + Provides endpoints for customers to view their orders and update their information.
* **API Endpoints (Examples):**
  + GET /customers/{customerId}/orders: Get a customer's order history.
  + PUT /customers/{customerId}: Update customer information.

**5. Data Model**

|  |  |  |
| --- | --- | --- |
| **Entity** | **Attributes** | **Relationships** |
| Order | id, customer, orderItems, orderStatus, ... | Many-to-One with Customer, One-to-Many with OrderItem |
| Customer | id, name, email, phone, address, orders | One-to-Many with Order |
| OrderItem | id, order, menuItem, quantity | Many-to-One with Order, Many-to-One with MenuItem |
| MenuItem | id, name, description, price, restaurant | Many-to-One with Restaurant |
| Restaurant | id, name, address, menuItems, ... | One-to-Many with MenuItem |

**6. Communication**

* **Synchronous:** RESTful API calls for direct communication between microservices and the API Gateway.
* **Asynchronous:** Message queues for handling order notifications, status updates, and other events.

**7. Technologies**

* Java
* Spring Boot
* Spring Data JPA (for database interaction)
* RabbitMQ or Kafka (message queue)
* OpenAPI/Swagger (API documentation)
* Docker (for containerization - to be added later)
* Kubernetes (for orchestration and scaling - to be added later)

**8. Future Enhancements**

* Implement a payment gateway integration.
* Integrate with delivery providers for real-time tracking.
* Implement a rating and review system.