# **Restaurant-Foodie Platform: S.D Document**

## **High-Level Overview**

A food delivery app requirements:

1. A restaurant can register itself on the platform with the menu and pricing
2. A foodie can search for a restaurant by either its name, a dish they wish to savour or simply a cuisine they would like to indulge in.
3. The foodie can order different dishes from different restaurants at the same time.
4. We have tied up with a digital wallet partner and would be relying on their APIs to credit/debit the customer’s wallet.
5. Restaurant should be able to update their menu and inventory and only accept the orders they’re capable of serving
6. The freshly prepared food will be collected and delivered by our trusted delivery partners.
7. The foodies can track the real time location of the delivery partners while eagerly waiting for their meal.

**Tech stack**

| Layer | Technology |
| --- | --- |
| Backend | Ruby on Rails, |
| API Gateway | AWS API Gateway, GraphQL, Rails rest api |
| Database | PostgreSQL, Redis, MongoDB |
| Search Engine | Elasticsearch, pg\_search (faster development) |
| Real-Time Tracking | WebSockets / Action cable |
| Payments | Razorpay/Stripe |
| Messaging | SQS/SNS/Action Cable |
| Hosting | AWS/ Docker |
| Deployment | GitHub Actions |
| Monitoring | AWS CloudWatch / open telemetry with honeycomb / New relic/ Scout |

## **3. Service Architecture**

### **Microservices Architecture**

* **User Service:** Manages user and restaurant accounts.
* **Restaurant Service:** Handles restaurant menus and inventory updates.
* **Order Service:** Supports multi-restaurant ordering.
* **Payment Service:** Integrates with digital wallet APIs.
* **Delivery Service:** Assigns delivery partners and tracks real-time location.
* **Search Service:** Full-text search powered by Elasticsearch/ PG Search.
* **Notification Service:** Sends push notifications and SMS updates.

## **4. Database Models**

### **Users Table - has many orders**

| Field | Type |
| --- | --- |
| id | UUID -PK |
| name | String |
| email | String |
| phone\_number | String |
| user\_type | ENUM (foodie, restaurant\_owner, delivery\_partner) |
| wallet\_id | String (from wallet partner) |

### **Restaurants Table - has many menu items**

| Field | Type |
| --- | --- |
| id | UUID - PK |
| name | String |
| address | String |
| cuisine | String[] |
| status | ENUM (active, inactive) |

### **Menu Items Table - belongs to restaurants**

| Field | Type |
| --- | --- |
| id | UUID - PK |
| restaurant\_id | UUID - FK |
| name | String |
| description | String |
| price | Decimal |
| available\_quantity | Integer |

### **Orders Table - belongs to uses, has many order items**

| Field | Type |
| --- | --- |
| id | UUID - PK |
| user\_id | UUID - FK |
| total\_amount | Decimal |
| status | ENUM (pending, confirmed, preparing, out\_for\_delivery, delivered, cancelled) |
| payment\_status | ENUM (pending, completed, failed) |
| created\_at | Timestamp |

### **Order Items Table - belongs to order**

| Field | Type |
| --- | --- |
| id | UUID -PK |
| order\_id | UUID |
| menu\_item\_id | UUID |
| restaurant\_id | UUID |
| quantity | Integer |
| price | Decimal |

### **Delivery Table - belongs to order, belongs to user**

| Field | Type |
| --- | --- |
| id | UUID |
| order\_id | UUID |
| delivery\_partner\_id | UUID |
| status | ENUM (assigned, picked\_up, delivered) |
| current\_location | Geopoint |

## **5. Service Communication**

* **Synchronous Communication:** REST APIs for CRUD operations and order management.
* **Asynchronous Communication:** Message queues (SNS/SQS)/Action cable for payment, delivery, and order status updates.

## **6. Search Flow**

* Foodies can search by restaurant name, dish, or cuisine.
* Elasticsearch/PG search indexes restaurant, menu, and cuisine data for fast retrieval.

## **7. Order Flow**

1. Foodie selects dishes from multiple restaurants.
2. System splits the order per restaurant.
3. Payment is processed via digital wallet APIs.
4. Restaurants confirm orders and start preparation.
5. Delivery partner is assigned.
6. Real-time location tracking is enabled for the foodie.

## **8. Real-Time Tracking**

* Delivery partners update their location via the mobile app.
* WebSockets or Firebase streams the delivery location to the foodie.
* Push notifications and SMS updates are sent for order status changes.

## **9. Scalability Considerations**

* Redis caching for frequently accessed data (menus, user sessions).
* Database sharding for large user and order datasets.
* Horizontal scaling using docker swarm, We can use K8s for better scaling
* Separate read/write replicas for high-traffic databases Using RDS