The Multi-Restaurant Food Ordering System - A Synopsis

I. Introduction to the project:

The Multi-Restaurant Food Ordering System is a scalable, efficient, and modular platform that allows users to browse multiple restaurants, place food orders, make payments, and track deliveries.
The system follows a Microservices Architecture, where each core functionality—such as user management, restaurant services, order processing, and delivery tracking—is handled by an independent service.
This architecture ensures high availability, scalability, and better fault isolation while enabling teams to develop, deploy, and maintain different components independently.

II. Problem statement:

Traditional food ordering systems often face scalability, performance bottlenecks, and single-point failures due to a monolithic architecture. As order volumes increase, systems become slower, harder to maintain, and more prone to failures.
Moreover, integrating multiple restaurants, payment gateways, and delivery management into a single platform leads to complex dependencies, delays, and inefficiencies.

Thus, there is a need for a microservices-based food ordering system that:

- Scales efficiently as order demand grows.
- Improves fault tolerance, ensuring a single service failure doesn't crash the entire system.
- Enhances performance, with optimized restaurant listing, order processing, and delivery tracking.
- Simplifies system maintenance by enabling independent updates and deployments.

III. Objectives:

- To develop a modular and scalable food ordering system using microservices.
- To enable efficient order processing, real-time order tracking, and secure payments.
- To provide a seamless user experience through an interactive UI and real-time updates.
- To integrate multiple restaurants, menu management, and delivery services into one system.
- To ensure high availability and fault tolerance with distributed services.

IV. Scope of the project:

- User Management: Users can register, log in, and manage their profiles.
- Restaurant Management: Restaurants can list their menus, update availability, and accept orders.
- Order Processing: Users can place and track orders in real time.
- Payment Integration (Optional): Secure online payment processing.
- Delivery Tracking: Assigning delivery personnel and tracking order status.
- Review System: Users can rate and review restaurants and food.
- Microservices-Based Approach: Independent modules for user, restaurant, order, and delivery management.
- Scalable Deployment: Docker & Kubernetes-based deployment for cloud scalability.

V. Technologies used:

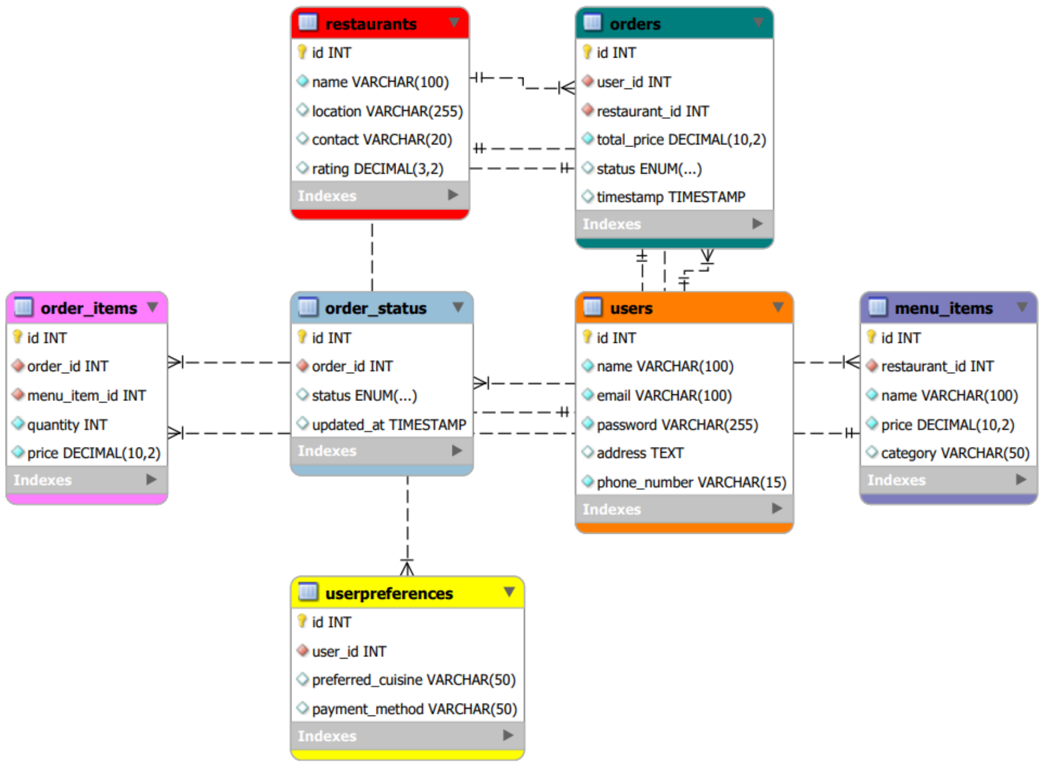
- Backend (Microservices & APIs)
  - Spring Boot (Java) – Backend framework for building REST APIs.
  - Spring Cloud – Microservices communication and service discovery.
  - RESTful APIs – Enables interaction between microservices.
- Frontend
  - React.js / Angular – Frontend framework for an interactive UI.

- Database & Storage
    - MySQL – Relational database for structured data.
- Security & Authentication
    - JWT (JSON Web Token) – For secure authentication.
- Deployment & Containerization
    - Docker – To containerize microservices.
    - Kubernetes – For deploying and managing microservices efficiently.

VI. Expected outcome:

- A scalable and modular multi-restaurant food ordering platform.
- Real-time order tracking and delivery assignment.
- Secure payment processing and transaction handling.
- Seamless user experience with an interactive UI.
- Microservices architecture, ensuring fault tolerance and independent scalability.

ERD (Entity Relationship Diagram)



VII. Relationships between entities :

- *User Service (Manages User Accounts and Preferences)*
  - Users Table (id, name, email, password, address, phone_number)
  - User Preferences Table (id, user_id, preferred_cuisine, payment_method)

Relationships:

➔ Users (1:M) Orders – One user can place multiple orders.
➔ Users (1:M) Payments – A user can make multiple payments.
➔ Users (1:M) Reviews – A user can leave multiple reviews.

- *Restaurant Service (Stores Restaurant Details and Menu Items)*

  - Restaurants Table (id, name, location, contact, rating)
  - Menu_Items Table (id, restaurant_id, name, price, category)

  Relationships:

  - ➔ Restaurants (1:M) Menu_Items – A restaurant can have multiple menu items.
  - ➔ Restaurants (1:M) Orders – A restaurant can receive multiple orders.
  - ➔ Restaurants (1:M) Reviews – Users can review multiple restaurants.

- *Order Service (Manages Orders and Tracking)*
  - Orders Table (id, user_id, restaurant_id, total_price, status, timestamp)
  - Order_Items Table (id, order_id, menu_item_id, quantity, price)
  - Order_Status Table (id, order_id, status, updated_at)

  Relationships:

  - ➔ Orders (1:M) Order_Items – An order can contain multiple items.
  - ➔ Orders (1:M) Order_Status – An order can have multiple status updates.
  - ➔ Orders (1:1) Deliveries – Each order has one delivery entry.
  - ➔ Orders (1:1) Payments – Each order has one payment entry.

- *Delivery Service (Manages Drivers and Deliveries)*
  - Drivers Table (id, name, phone_number, vehicle_details, status)
  - Deliveries Table (id, order_id, driver_id, status, estimated_time)

  Relationships:

  - ➔ Drivers (1:M) Deliveries – A driver can deliver multiple orders.
  - ➔ Deliveries (1:1) Orders – Each order is assigned to one delivery.

- *Payment Service (Handles Payment Transactions)*
  - Payments Table (id, order_id, user_id, amount, status, payment_method, transaction_id)

  - **Relationships:**
  - ➔ Payments (1:1) Orders – Each order has one payment.
  - ➔ Payments (M:1) Users – A user can make multiple payments.

VIII. Normalization of the database:

The database schema for the project follows the principles of normalization to eliminate redundancy and ensure data integrity. The Users, Restaurants, Orders, Deliveries, and Payments tables are designed in at least Third Normal Form (3NF) by ensuring that each table has a primary key, dependencies are fully functional, and there are no transitive dependencies.

The Users Table maintains unique user details, while User Preferences are stored separately to avoid duplication.
The Restaurants Table keeps restaurant details distinct from menu items, which are stored in a related table to prevent data repetition.
The Orders Table is linked to Order_Items to ensure a normalized many-to-many relationship between orders and menu items.
Similarly, Order_Status tracks order updates separately, reducing redundancy.
The Deliveries Table maintains a 1:1 relationship with Orders, ensuring that each order has a unique delivery record while allowing multiple deliveries per driver.
Payments are associated with orders and users, ensuring transaction integrity.

Document created by:
- B019 Riya Bhattacharya (B. Tech CE - B)
- B057 Precious Sukhi (B. Tech CE - B)
- B061 Harshita Ranka (B. Tech CE - B)
- B062 Sanjana Surana (B. Tech CE - B)