Project: Online Food Delivery System

1. Introduction

The purpose of this document is to provide a detailed Low-Level Design (LLD) for the Online Food Delivery System. The platform allows customers to browse food menus, place orders, and track deliveries. It includes features for restaurant management, customer management, and delivery tracking. This system utilizes a REST API-based backend and Angular or React for the frontend.

The design supports both Java (Spring Boot) and .NET (ASP.NET Core) frameworks.

2. Module Overview

The project consists of the following modules:

2.1 Customer Management

Handles user registration, authentication, and profile management.

2.2 Menu Management

Allows restaurants to update and manage their food menus.

2.3 Order Management

Facilitates order placement, tracking, and status updates.

2.4 Delivery Management

Tracks the delivery status, assigns delivery agents, and updates customers.

2.5 Payment Management

Handles payment transactions for food orders.

3. Architecture Overview

3.1 Architectural Style

- **Frontend**: Angular or React
- **Backend**: REST API-based architecture
- **Database**: Relational Database (MySQL/PostgreSQL/SQL Server)

3.2 Component Interaction

- The frontend interacts with the backend API for all user-related actions.
- The backend manages the business logic, processes orders, and stores information in the database.

- Payment gateway interactions will be handled by the backend.

4. Module-Wise Design

4.1 Customer Management Module

4.1.1 Features

- Register and login as a customer.
- View and update user profile details.

4.1.2 Data Flow

1. Customers register/login via the frontend.
2. Frontend sends user data to the backend for authentication and profile management.
3. Backend processes the requests and stores data in the database.

4.1.3 Entities

- **Customer**
    - CustomerID
    - Name
    - Email
    - Phone
    - Address

4.2 Menu Management Module

4.2.1 Features

- Restaurants can add, update, and remove menu items.
- Menus are displayed for customers to browse and select.

4.2.2 Data Flow

1. Restaurant owners add/update their menus via the frontend.
2. The frontend sends data to the backend API to store the menu items.
3. The menu is fetched from the backend and displayed to customers.

4.2.3 Entities

- **MenuItem**
    - ItemID
    - Name
    - Description
    - Price
    - RestaurantID

4.3 Order Management Module

4.3.1 Features

- Customers can place orders for selected menu items.
- Track the status of placed orders.

4.3.2 Data Flow

1. Customers select items and place an order via the frontend.

2. The frontend sends the order details to the backend API.
3. The backend processes the order and updates the order status.
4. Customers can track their order through the frontend.

4.3.3 Entities

- **Order**
    - OrderID
    - CustomerID
    - RestaurantID
    - Status (Pending, Accepted, Preparing, Delivered)
    - TotalAmount

4.4 Delivery Management Module

4.4.1 Features

- Assign delivery agents to orders.
- Track the delivery progress and notify customers.

4.4.2 Data Flow

1. Delivery agents are assigned to orders by the backend.
2. Delivery status is updated (e.g., out for delivery, delivered).
3. Customers are notified of order progress.

4.4.3 Entities

- **Delivery**
    - DeliveryID
    - OrderID
    - AgentID
    - Status (In Progress, Delivered)
    - EstimatedTimeOfArrival

4.5 Payment Management Module

4.5.1 Features

- Process payments for placed orders.
- Support multiple payment methods (credit card, wallet, etc.).

4.5.2 Data Flow

1. Customers proceed to payment after placing an order.
2. The frontend sends payment details to the backend API.
3. The backend processes the payment with a payment gateway (third-party integration).
4. Payment confirmation is sent back to the frontend.

4.5.3 Entities

- **Payment**
    - PaymentID
    - OrderID
    - PaymentMethod (Card, Wallet, etc.)
    - Amount
    - Status (Successful, Failed)

5. Deployment Strategy

5.1 Local Deployment

- **Frontend**: Angular/React development server for testing.
- **Backend**: REST API deployed using Spring Boot/ASP.NET Core.
- **Database**: Local database setup for development and testing.

6. Database Design

6.1 Tables and Relationships

- **Customer**
  - Primary Key: CustomerID
- **MenuItem**
  - Primary Key: ItemID
  - Foreign Key: RestaurantID
- **Order**
  - Primary Key: OrderID
  - Foreign Key: CustomerID, RestaurantID
- **Delivery**
  - Primary Key: DeliveryID
  - Foreign Key: OrderID, AgentID
- **Payment**
  - Primary Key: PaymentID
  - Foreign Key: OrderID

7. User Interface Design

7.1 Wireframes

- **Customer Dashboard**: View and manage orders, profile details.
- **Restaurant Management Interface**: Add and update menu items.
- **Order Tracking Page**: Monitor order progress in real-time.
- **Payment Page**: Enter payment details and confirm payment.

8. Non-Functional Requirements

8.1 Performance

- The system should be able to process 100+ orders per minute with minimal delay.

8.2 Scalability

- Easily scalable to handle large numbers of users in a production environment.

8.3 Security

- Secure handling of payment data using encryption and PCI-DSS compliance.

8.4 Usability

- The user interface must be mobile-responsive and user-friendly for both customers and restaurants.

9. Assumptions and Constraints

9.1 Assumptions

- The system will operate in a local environment during initial development and testing.

9.2 Constraints

- No integration with external third-party services other than payment gateways in the first release.