📘 Food Delivery System

📌 1. Project Planning

1.1 Project Title

Online Food Delivery System

1.2 Project Description

The Food Delivery System is a web-based application that allows users to browse restaurants, order food online, make payments, and track their orders.
Admins can manage restaurants, menu items, and orders.

The system is built using:

- Backend: Spring Boot
- Database: MySQL
- API Testing: Swagger
- Version Control: Git & GitHub

📌 2. Project Objectives

- Allow users to register and login
- Allow users to browse restaurants and food items
- Allow users to place and pay for orders
- Allow admin to manage restaurants and orders
- Provide order status tracking (Pending, Confirmed, Delivered)

📌 3. Project Scope

In Scope:

- User Authentication (JWT based login/register)
- Restaurant Management
- Menu Management
- Order Management
- Payment Status Handling
- Admin Panel APIs

- Swagger API Documentation

Out of Scope:

- Real payment gateway integration (Razorpay/Stripe)
- Live GPS tracking
- Mobile application

📌 4. Requirements Analysis

4.1 Functional Requirements

👤 User Module

- User can register
- User can login
- User can view restaurants
- User can view menu items
- User can add items to order
- User can place order
- User can make payment
- User can view their orders

👨‍💼 Admin Module

- Admin can login
- Admin can add restaurants
- Admin can update restaurants
- Admin can delete restaurants
- Admin can manage menu items
- Admin can update order status

4.2 Non-Functional Requirements

- System should respond within 2–3 seconds
- System should support multiple users
- Secure authentication using JWT

- Data stored securely in database
- REST API based architecture
- Proper exception handling

📌 5. System Architecture

Architecture Type:

- Layered Architecture

Layers:

1. Controller Layer (Handles API requests)
2. Service Layer (Business logic)
3. Repository Layer (Database interaction)
4. Database (MySQL)

📌 6. Technology Stack

Component	Technology
Backend	Spring Boot
Database	MySQL
Security	Spring Security + JWT
Build Tool	Maven
API Testing	Swagger
Version Control	Git & GitHub

📌 7. Database Design (Main Entities)

- User
- Role
- Restaurant
- MenuItem
- Order
- OrderItem
- PaymentStatus (Enum)
- OrderStatus (Enum)

📌 9. Risk Management

- Git merge conflicts
- Enum mismatch errors
- Build failures (Maven)
- Dependency version issues

Mitigation:

- Use proper version control
- Clean & rebuild project
- Test APIs regularly

📌 10. Conclusion

The Food Delivery System provides a structured backend solution for online food ordering. It demonstrates REST API development, authentication, database management, and layered architecture using Spring Boot.

📌 Functionalities of All Roles and Modules

🍔 Food Delivery System

👥 USER ROLES & THEIR FUNCTIONALITIES

1️⃣ Admin

The Admin manages the overall system.

- 🔹 **Admin Functionalities:**
  - Login & Logout
  - View Dashboard (Total Users, Orders, Restaurants, Revenue)
  - Add / Update / Delete Restaurants
  - Add / Update / Delete Food Items
  - View All Orders
  - Update Order Status (CONFIRMED, PREPARING, DELIVERED, CANCELLED)
  - Manage Users
  - Filter Orders by Status
  - View Payment Status
  - Manage Categories

2️⃣ Customer (User)

Customers can browse and place food orders.

- 🔹 **Customer Functionalities:**
  - Register & Login
  - View Restaurants
  - View Restaurant Menu
  - Add Items to Cart
  - Update / Remove Items from Cart
  - Place Order
  - Make Payment

- View Order History
- Track Order Status
- Update Profile
- Logout

3️⃣ Restaurant Owner (If implemented)

Manages their own restaurant.

🔹 **Restaurant Functionalities:**

- Login
- Add / Update / Delete Food Items
- View Incoming Orders
- Update Order Status (PREPARING, READY)
- View Sales Reports

🚴 **4️⃣ Delivery Partner Role**

📌 **Responsibilities:**

Handles delivery of orders.

✅ **Functionalities:**

- Register / Login
- View Assigned Orders
- Accept Delivery Request
- Update Delivery Status:
    - Picked Up
    - Out for Delivery
    - Delivered
- View Delivery History
- View Earnings

🧩 SYSTEM MODULES & FUNCTIONALITIES

1️⃣ Authentication Module

- User Registration
- User Login
- JWT Token Generation
- Role-Based Authorization
- Password Encryption
- Secure API Access

2️⃣ User Management Module

- Create User
- Update User Profile
- Delete User (Admin)
- View All Users (Admin)

3️⃣ Restaurant Module

- Add Restaurant (Admin)
- Update Restaurant
- Delete Restaurant
- View All Restaurants
- View Restaurant by ID

4️⃣ Menu / Food Item Module

- Add Food Item
- Update Food Item
- Delete Food Item
- View Food Items by Restaurant

- Filter by Category

5️⃣ Cart Module

- Add Item to Cart
- Remove Item
- Update Quantity
- Calculate Total Price

6️⃣ Order Module

- Place Order
- Get My Orders
- Get All Orders (Admin)
- Filter Orders by Status
- Update Order Status
- View Order Details

7️⃣ Payment Module

- Process Payment
- Update Payment Status (PENDING, COMPLETED, FAILED)
- Link Payment to Order
- Store Payment Information

5️⃣ Delivery Module

- Assign Delivery Partner
- Update Delivery Status
- Track Delivery

8️⃣ Reporting Module (Optional Advanced Feature)

- Total Revenue
- Total Orders
- Daily / Monthly Reports
- Most Ordered Items

🔐 ROLE-BASED ACCESS CONTROL SUMMARY

Module	Admin	Customer	Restaurant
Login/Register	✅	✅	✅
Manage Users	✅	❌	❌
Manage Restaurants	✅	❌	Limited
Manage Menu	✅	❌	✅
Place Order	❌	✅	❌
View All Orders	✅	❌	Own Only
Update Order Status	✅	❌	Partial
Payment Processing	❌	✅	❌

I want you to build a complete **Food Delivery System Backend** using:

- Java 17
- Spring Boot
- Spring Security with JWT
- Spring Data JPA (Hibernate)
- MySQL
- Maven
- Swagger (OpenAPI)
- Proper layered architecture

🎯 Project Overview

Build a production-ready backend for a Food Delivery System with 4 roles:

1. ADMIN
2. CUSTOMER
3. RESTAURANT
4. DELIVERY_AGENT

Use proper package structure, clean architecture, DTOs, exception handling, and validation.

🏗 Required Architecture

Use layered architecture:

```
com.fooddelivery.backend
│
├── config
├── controller
├── service
├── repository
├── model
├── dto
├── enums
├── exception
└── security
```

Follow:

- Controller → Service → Repository pattern
- Use DTOs (never expose entities directly)
- Global exception handler
- Proper validations using @Valid
- Use Lombok

🔐 Authentication & Security

Implement:

- JWT Authentication
- Login & Register
- Role-based authorization
- Secure endpoints based on roles

- Password encryption using BCrypt

📦 Modules & Functionalities

1️⃣ AUTH MODULE

- Register user (with role)
- Login
- Generate JWT token
- Get current logged-in user

2️⃣ ADMIN MODULE

- View all users
- View all restaurants
- Approve / reject restaurants
- View all orders
- Update order status
- View analytics (total users, total orders, total revenue)

3️⃣ CUSTOMER MODULE

- Register & Login
- Browse restaurants
- View menu items
- Add to cart
- Place order
- View my orders
- Make payment
- Track order

4️⃣ RESTAURANT MODULE

- Register restaurant
- Add / Update / Delete menu items
- View incoming orders
- Update order status
- Accept / reject orders

5️⃣ DELIVERY AGENT MODULE

- View assigned orders
- Accept delivery
- Update delivery status
- Mark as delivered

📄 **Entities Required**

Create proper relationships:

- User
- Restaurant
- MenuItem
- Order
- OrderItem
- Cart
- Payment
- Delivery

Use:

- @OneToMany
- @ManyToOne
- @JoinColumn
- Proper cascade types

📊 **Enums Required**

- Role (ADMIN, CUSTOMER, RESTAURANT, DELIVERY_AGENT)
- OrderStatus (PENDING, CONFIRMED, PREPARING, OUT_FOR_DELIVERY, DELIVERED, CANCELLED)
- PaymentStatus (PENDING, COMPLETED, FAILED)

🛠 API Requirements

- RESTful API design
- Proper HTTP status codes
- Pagination support
- Sorting support
- Filtering support
- Swagger documentation
- Clear request/response examples

🗃 Database

- Use MySQL
- Proper foreign key relationships
- Auto table creation
- application.yml configuration

⚙ Non-Functional Requirements

- Clean code
- Proper logging
- Centralized exception handling
- Validation messages
- Professional structure
- No circular dependency
- Production-ready structure

📘 Deliverables

Generate:

1. Complete project structure
2. All entities
3. DTOs
4. Controllers
5. Services
6. Repositories
7. Security configuration
8. JWT utility
9. Global exception handler
10. Swagger configuration
11. application.yml
12. Sample API test examples

🚀 ADVANCED FOOD DELIVERY SYSTEM (PRODUCTION READY)

I want you to build a **production-ready Food Delivery System Backend** using:

- Java 17
- Spring Boot
- Spring Security + JWT
- Spring Data JPA (Hibernate)
- MySQL
- Stripe (or Razorpay) Payment Gateway Integration
- Swagger (OpenAPI)
- Docker support
- Clean Architecture

🎯 Project Goal

Build a scalable, secure, real-world backend system similar to **Swiggy/Zomato** with:

- 4 Roles:
    - ADMIN
    - CUSTOMER
    - RESTAURANT
    - DELIVERY_AGENT
- Secure authentication
- Real payment integration
- Order lifecycle management
- Delivery assignment system
- Production-ready structure

🏗 Architecture Requirements

Use **Layered Architecture**:

controller
service
repository
model
dto
enums
exception
security
config

Follow:

- Controller → Service → Repository
- DTO pattern
- Global Exception Handling
- Validation using @Valid
- Clean RESTful APIs
- No circular dependencies

🔐 Security Requirements

- JWT Authentication
- Role-based access control
- BCrypt password encryption
- Custom JWT filter
- Secure endpoints by role
- Refresh token support (optional advanced)

💳 PAYMENT GATEWAY INTEGRATION

Integrate **Stripe or Razorpay**:

Payment Flow:

1. Customer places order
2. Order status = PENDING
3. Create Payment Intent via Stripe API
4. Return client secret
5. On successful payment:
    - PaymentStatus = COMPLETED
    - OrderStatus = CONFIRMED

Requirements:

- Payment entity
- Store transaction ID
- Store payment timestamp
- Handle payment failure
- Webhook endpoint to verify payment
- Secure secret key handling via application.yml

📦 Core Modules

1️⃣ AUTH MODULE

- Register
- Login
- JWT generation
- Role-based access

2️⃣ CUSTOMER MODULE

- Browse restaurants
- View menu
- Add to cart
- Place order
- Pay online
- Track order
- Cancel order (if not confirmed)

3️⃣ RESTAURANT MODULE

- Add/update/delete menu items
- Accept/reject orders
- Update status (PREPARING → READY)

4️⃣ DELIVERY MODULE

- Auto assign delivery agent
- Accept delivery
- Update delivery status:
    - PICKED_UP
    - OUT_FOR_DELIVERY
    - DELIVERED
- Calculate delivery earnings

5️⃣ ADMIN MODULE

- View analytics
- Manage users
- Manage restaurants
- Monitor payments
- Dashboard statistics:
    - Total Revenue
    - Total Orders
    - Active Users

📊 Order Lifecycle (Advanced)

PENDING
→ PAYMENT_COMPLETED
→ CONFIRMED
→ PREPARING
→ READY
→ OUT_FOR_DELIVERY
→ DELIVERED

📜 Required Entities

- User
- Role
- Restaurant
- MenuItem
- Cart
- Order
- OrderItem
- Payment
- Delivery

Include proper relationships:

- @OneToMany
- @ManyToOne
- Cascade rules
- Fetch types optimized

📈 **Advanced Features**

- Pagination & Sorting
- Order filtering
- Revenue calculation
- Logging using SLF4J
- Global exception handler
- Proper HTTP status codes
- Swagger documentation
- Dockerfile + docker-compose.yml
- Environment-based configuration

🗄 **Database**

- MySQL
- Proper foreign key constraints
- Indexing on frequently queried fields
- Soft delete (optional advanced)

📦 **Deployment Ready**

Include:

- Dockerfile
- docker-compose for MySQL + App
- Production profile
- application-prod.yml

🧪 Testing

- Swagger UI support
- Example API request bodies
- Proper validation error responses

🎯 Ensure:

- No compilation errors
- No enum mismatch
- No LazyInitializationException
- No circular JSON issues
- Clean and readable code
- Production-level folder structure

📘 Deliverables

Generate:

1. Complete project structure
2. All entities
3. Security config
4. JWT utility
5. Controllers
6. Services
7. Repositories
8. Payment integration service
9. Webhook controller
10. Docker configuration
11. Swagger config
12. Sample API test instructions
