🍔 Online Food Ordering System (OFOS) – Case Study

Scope

The Online Food Ordering System (OFOS) enables customers to order food from multiple restaurants, supports real-time order tracking, and integrates with online payment systems. It connects **Admin, Restaurant Owners, and Customers** with a seamless experience.

Roles & Modules

Admin

- Manage restaurants, users, and payments.
- Generate reports (sales, customer activity).
- Handle refunds and disputes.

Restaurant Owner

- Manage restaurant profile & menus.
- View & process incoming orders.
- Update order status (Preparing, Out for Delivery, Delivered).

Customer

- Browse menus with search & filters.
- Place orders & make online payments.
- Track delivery status in real-time.
- Rate & review restaurants.

Features

✅ Menu management & advanced search
✅ Secure JWT-based authentication for all roles
✅ Online payment integration (Stripe/PayPal API)
✅ Order tracking with live status updates (WebSocket/REST Polling)

✅ Restaurant ratings & customer reviews
✅ Reports (top restaurants, most ordered dishes)

Tech Stack

- **Backend**: Spring Boot (REST APIs, Swagger for documentation)
- **Frontend**: React + Bootstrap (responsive UI)
- **Database**: MySQL
- **Authentication**: JWT Security
- **External API Integrations**:
  - Payment API (Stripe/PayPal)
  - Google Maps API (delivery tracking & restaurant location)
  - Notification API (Twilio/SendGrid for SMS/Email updates)

API Integrations

Auth & User APIs

- `POST /api/auth/register` → Register new user (Admin/Restaurant/Customer)
- `POST /api/auth/login` → Login & get JWT token
- `GET /api/users/profile` → View profile

Restaurant APIs

- `POST /api/restaurants` → Add new restaurant (Admin)
- `GET /api/restaurants` → Get all restaurants (Customer search)
- `PUT /api/restaurants/{id}` → Update restaurant details (Owner/Admin)
- `DELETE /api/restaurants/{id}` → Remove restaurant (Admin)

Menu APIs

- `POST /api/menus` → Add menu item (Owner)
- `GET /api/menus/{restaurantId}` → Get menu for restaurant

- `PUT /api/menus/{id}` → Update menu item
- `DELETE /api/menus/{id}` → Delete menu item

Order APIs

- `POST /api/orders` → Place order (Customer)
- `GET /api/orders/{id}` → Get order details
- `PUT /api/orders/{id}/status` → Update status (Restaurant Owner)
- `GET /api/orders/user/{userId}` → Get order history for a user

Payment APIs (Integration with Stripe/PayPal)

- `POST /api/payments/initiate` → Start payment
- `POST /api/payments/confirm` → Confirm payment after gateway callback
- `GET /api/payments/history` → Payment history

Review APIs

- `POST /api/reviews` → Add restaurant review
- `GET /api/reviews/{restaurantId}` → Get reviews for restaurant

Swagger Integration

- `http://localhost:8080/swagger-ui.html` → Interactive API documentation.
- Each API endpoint includes request/response schema & test UI.

Security (JWT Flow)

1. User logs in → Spring Boot generates JWT token.
2. React frontend stores token (in `localStorage`).
3. All secured APIs require JWT in `Authorization: Bearer <token>`.
4. Role-based authorization (Admin, Restaurant Owner, Customer).

React + Bootstrap Frontend

- **Customer UI**: Restaurant listing, menu browsing, cart, order tracking.
- **Restaurant UI**: Dashboard with active orders, menu editor.
- **Admin UI**: User management, payment reports, dispute resolution.

Example Workflow

1. **Customer** registers and logs in via `/auth/register`.
2. **Customer** browses restaurants → selects items → places order (`/orders`).
3. **Payment** initiated via `/payments/initiate` → integrated with Stripe API.
4. **Restaurant Owner** gets notification of order → updates status (`/orders/{id}/status`).
5. **Customer** tracks order in real-time (polling or WebSocket).
6. **Customer** leaves review via `/reviews`.
7. **Admin** monitors reports using `/reports`.

Reports & Analytics

- Most ordered dishes API → `/reports/top-dishes`
- Top restaurants API → `/reports/top-restaurants`
- Monthly sales API → `/reports/sales?month=2025-08`