NestJS Food Delivery Backend

A production-ready, fully-featured backend for a food delivery platform built using **NestJS** and **MySQL**, complete with **JWT authentication**, **role-based access**, **WebSocket real-time updates**, and **Swagger API documentation**.

> "Build once, scale infinitely."

Features

Roles & Permissions

- **Customer**: Browse restaurants and menus, place orders, track status.
- **Restaurant Owner**: Manage own restaurants and menu items.
- **Delivery Rider**: Accept and deliver orders, update status.

Authentication handled via **JWT + Passport**
Role-based access control using custom guards

Restaurant & Menu Management

- Owners can create/update/delete:
    - Restaurants
    - Menu items (linked to their own restaurants)
- Access is tightly scoped to authenticated owners

Order Management

- Customers can:
    - View restaurants and menus
    - Place orders (mocked payment)
    - View order history & details
- Orders include:
    - **Delivery status** (`pending`, `accepted`, `picked_up`, `delivered`)
    - **Assigned rider info**
    - **Delivery timestamp**

Delivery Workflow

- Riders can:
    - View unassigned orders

    - Accept one order at a time
    - Update order status as they deliver

Real-Time Order Updates

- Implemented using **WebSocket (Socket.IO)**
- Customers receive **live delivery updates**
- Secure, room-based subscriptions by user

API Documentation

- Fully documented with **Swagger UI**
- Available at: http://localhost:3000/api-docs
- Auto-generated schemas, request bodies, role info

Tech Stack

Category	Stack
Language	TypeScript
Framework	NestJS
Database	MySQL
ORM	TypeORM
Authentication	JWT, Passport
Real-time	WebSocket (Socket.IO)
API Docs	Swagger (`@nestjs/swagger`)
DevOps	Docker, docker-compose

Project Structure

```
src/
├── auth/               // Auth logic, JWT, guards, roles
├── user/               // User entity & service
├── restaurant/         // Restaurant CRUD for owners
├── menu/               // Menu items CRUD
├── order/              // Order placing, history, delivery updates
├── websocket/          // WebSocket gateway for real-time updates
├── common/             // Interceptors, decorators, utils
├── main.ts             // App bootstrap
```

Endpoints Overview

Auth

- `POST /auth/register`

- `POST /auth/login`
- `GET /auth/me`

Restaurant

- `POST /restaurant` (owner only)
- `GET /restaurant` (public)
- `PATCH /restaurant/:id` (owner only)
- `DELETE /restaurant/:id` (owner only)

Menu

- `POST /menu/:restaurantId` (owner only)
- `PATCH /menu/:id` (owner only)
- `DELETE /menu/:id` (owner only)
- `GET /menu/restaurant/:restaurantId` (public)

Orders

- `POST /orders` (customer only)
- `GET /orders/history` (customer only)
- `GET /orders` (rider only)
- `POST /orders/:id/accept` (rider only)
- `POST /orders/:id/status` (rider only)

Local Setup (Dockerized)

Prerequisites

- Docker + Docker Compose

Steps

```
git clone https://github.com/your-username/food-delivery-backend.git
cd food-delivery-backend
cp .env.example .env
docker-compose up --build
```

Result

- Backend: http://localhost:3000
- Swagger: http://localhost:3000/api-docs

- MySQL DB: `localhost:3306`

  Automatically runs DB migration & seeds initial data

Environment Variables

`.env.example`:

```
DB_HOST=db
DB_PORT=3306
DB_USERNAME=root
DB_PASSWORD=secret
DB_NAME=food_delivery
JWT_SECRET=your_jwt_secret
JWT_EXPIRES_IN=3600s
```

Scripts

```
# Start in development
docker-compose up --build

# Run migration manually
npm run typeorm migration:run

# Seed script (on boot or manually)
npm run seed
```

Future Enhancements

- Email notifications
- Payment integration
- Admin dashboard
- Unit & e2e testing

Author

Kapil Patel
GitHub: **Kapil-k-git**

> "Clean code. Clear design. Complete features."
