Food Delivery System - Node.js + Express API

Table of Contents

Introduction

This document provides a **Node.js + Express** backend API specification for a Food Delivery System. It covers:

- A recommended project structure
- JWT-based authentication & role-based authorization
- Complete RESTful endpoints for:
    - Customer (mobile app)
    - Restaurant Owner (mobile/web)
    - Delivery Partner (mobile)
    - Super Admin (web dashboard)

- Brief implementation notes (status flows, etc.)

Project Structure

A typical layout for a Node.js + Express project:

```
food-delivery-backend/
 ├─ src/
 │   ├─ config/
 │   │   └─ db.js
 │   ├─ controllers/
 │   │   ├─ authController.js
 │   │   ├─ userController.js
 │   │   ├─ restaurantController.js
 │   │   ├─ menuController.js
 │   │   ├─ orderController.js
 │   │   ├─ deliveryController.js
 │   │   └─ ...
 │   ├─ middlewares/
 │   │   └─ authMiddleware.js
 │   ├─ models/
 │   ├─ routes/
 │   │   ├─ authRoutes.js
 │   │   ├─ userRoutes.js
 │   │   ├─ restaurantRoutes.js
 │   │   ├─ ...
 │   ├─ utils/
 │   └─ app.js
 ├─ package.json
 └─ README.md
```

Authentication & Authorization

- **JWT-based** authentication:
  - **Register/Login** for Customers, Restaurant Owners
  - **Delivery Partners** usually created by Super Admin
  - **Super Admin** might be created via script
- Each user has `role ∈ { CUSTOMER, RESTAURANT_OWNER, DELIVERY_PARTNER, SUPER_ADMIN }`.
- Role-based middleware checks `req.user.role` for endpoint access.

API Endpoints

> **Base URL**: Typically `https://your-domain/api/v1` (omitted in tables).

Auth Module

METHOD	ENDPOINT	DESCRIPTION	BODY/PARAMS	ACCESS
POST	`/auth/register`	Register (Customer or Restaurant Owner)	**Body**: { `fullName`, `email`, `phone`, `password`, `role` }	Public
POST	`/auth/login`	User login, returns JWT	**Body**: { `email`, `password` }	Public
POST	`/auth/logout`	Invalidate/remove token (e.g., blacklisting)	Possibly empty body.	Auth (Any)
POST	`/auth/refresh`	Refresh token flow (optional)	**Body**: { `refreshToken` }	Auth (Any)

User Management (Super Admin)

METHOD	ENDPOINT	DESCRIPTION	BODY/PARAMS	ACCESS
GET	`/users`	List all users. Filter by `?role=RESTAURANT_OWNER`, etc.	Query: `?role=...`, `?isActive=...`	SUPER_ADMIN
GET	`/users/:userId`	Get specific user details	URL param: `:userId`	SUPER_ADMIN
PATCH	`/users/:userId/activate`	Activate user (e.g., Restaurant Owner who was pending)	**Body**: { `is_active`: true }	SUPER_ADMIN
PATCH	`/users/:userId/deactivate`	Deactivate user	**Body**: { `is_active`: false }	SUPER_ADMIN
POST	`/users/delivery-partner`	Create a Delivery Partner (credentials assigned by Admin)	**Body**: { `fullName`, `email`, `phone`, `password`}	SUPER_ADMIN

Profile & Addresses (All Roles)

Route Group: `/profile`

METHOD	ENDPOINT	DESCRIPTION	BODY/PARAMS	ACCESS
GET	`/profile`	Get current user's profile	N/A	Auth (Any)
PATCH	`/profile`	Update profile (name, phone, etc.)	{ `fullName`, `phone`, ... }	Auth (Any)
GET	`/profile/addresses`	List all addresses of current user	N/A	Auth (Any)
POST	`/profile/addresses`	Create new address	{ `addressLine1`, `city`, `latitude`, `longitude`, ... }	Auth (Any)
GET	`/profile/addresses/:addressId`	Get specific address info	URL param: `:addressId`	Auth (Owner)
PATCH	`/profile/addresses/:addressId`	Update address	Updatable fields	Auth (Owner)
DELETE	`/profile/addresses/:addressId`	Remove an address	N/A	Auth (Owner)

Restaurants (Restaurant Owner + Admin)

METHOD	ENDPOINT	DESCRIPTION	BODY/PARAMS	ACCESS
GET	`/restaurants`	Publicly list **active** restaurants	Optional query: `?lat=...,lon=...,radius=...` for nearby search	Public
GET	`/restaurants/:id`	Get details of a specific restaurant	URL param: `:id`	Public
POST	`/restaurants`	Create a new restaurant	{ `name`, `description`, `addressLine1`, ...}	RESTAURANT_OWNER, SUPER_ADMIN
PATCH	`/restaurants/:id`	Update restaurant details	{ `name`, `description`, ... }	Owner of that restaurant / SUPER_ADMIN
DELETE	`/restaurants/:id`	Deactivate or delete a restaurant	N/A	Owner or SUPER_ADMIN

Menus & Menu Items

Menus

Route Group: `/restaurants/:restaurantId/menus`

METHOD	ENDPOINT	DESCRIPTION	BODY/PARAMS	ACCESS
GET	`/:restaurantId/menus`	List all menus for a restaurant	URL param: `:restaurantId`	Public
POST	`/:restaurantId/menus`	Create a new menu (e.g., "Breakfast")	{ `menuName`, `is_active` }	Owner of restaurant / SUPER_ADMIN
GET	`/:restaurantId/menus/:menuId`	Get a specific menu's details	URL param: `:menuId`	Public
PATCH	`/:restaurantId/menus/:menuId`	Update a menu	{ `menuName`, `is_active` }	Owner or SUPER_ADMIN
DELETE	`/:restaurantId/menus/:menuId`	Delete or deactivate a menu	N/A	Owner or SUPER_ADMIN

Menu Items

Route Group: `/restaurants/:restaurantId/menus/:menuId/items`

METHOD	ENDPOINT	DESCRIPTION	BODY/PARAMS	ACCESS
GET	`/:restaurantId/menus/:menuId/items`	List items in a menu	URL param: `:menuId`	Public
POST	`/:restaurantId/menus/:menuId/items`	Create a menu item	{ `itemName`, `description`, `price`, `is_active` }	Owner or SUPER_ADMIN
GET	`/:restaurantId/menus/:menuId/items/:itemId`	Get details of an item	URL param: `:itemId`	Public
PATCH	`/:restaurantId/menus/:menuId/items/:itemId`	Update item details	{ `itemName`, `description`, `price` }	Owner or SUPER_ADMIN
DELETE	`/:restaurantId/menus/:menuId/items/:itemId`	Delete/deactivate item	N/A	Owner or SUPER_ADMIN

Coupons (Restaurant Owner + Admin)

Route Group: `/restaurants/:restaurantId/coupons`

METHOD	ENDPOINT	DESCRIPTION	BODY/PARAMS	ACCESS
GET	`/:restaurantId/coupons`	List coupons for a restaurant	URL param: `:restaurantId`	Owner or SUPER_ADMIN
POST	`/:restaurantId/coupons`	Create a coupon	{ `code`, `discountType`, `discountValue`, `validFrom`, `validTo` }	Owner or SUPER_ADMIN
GET	`/:restaurantId/coupons/:couponId`	Get coupon details	URL param: `:couponId`	Owner or SUPER_ADMIN
PATCH	`/:restaurantId/coupons/:couponId`	Update coupon	{ `discountValue`, `validTo`, ... }	Owner or SUPER_ADMIN
DELETE	`/:restaurantId/coupons/:couponId`	Remove/deactivate coupon	N/A	Owner or SUPER_ADMIN

Orders

Customer Endpoints

METHOD	ENDPOINT	DESCRIPTION	BODY/PARAMS	ACCESS
GET	`/orders`	List orders for the **logged-in** customer	Optionally filter by `?status=PENDING`	CUSTOMER
POST	`/orders`	Create a new order	{ `restaurantId`, `addressId`, `couponId?`, `paymentMethod`, `items`: [{ `itemId`, `quantity` }]}	CUSTOMER
GET	`/orders/:id`	Get details of one order (must belong to current customer)	URL param: `:id`	CUSTOMER
PATCH	`/orders/:id`	Cancel an order (if `PENDING` or allowed)	{ `action`: "cancel" } or a dedicated `/cancel` endpoint	CUSTOMER

Restaurant Owner Endpoints

METHOD	ENDPOINT	DESCRIPTION	BODY/PARAMS	ACCESS
GET	`/orders/restaurant`	List orders for the owner's restaurant(s).	Possibly `?status=...`	RESTAURANT_OWNER
GET	`/orders/:id`	Get details (must belong to the owner's restaurant).	URL param: `:id`	RESTAURANT_OWNER
PATCH	`/orders/:id/accept`	Accept order (set `ACCEPTED`).	N/A	RESTAURANT_OWNER
PATCH	`/orders/:id/reject`	Reject order (set `REJECTED`).	{ `reason` } (optional)	RESTAURANT_OWNER
PATCH	`/orders/:id/prepare`	Set order status to `PREPARING`.	N/A	RESTAURANT_OWNER
PATCH	`/orders/:id/ready-for-pickup`	Set order status to `READY_FOR_PICKUP`.	N/A	RESTAURANT_OWNER

Delivery Partner Endpoints

METHOD	ENDPOINT	DESCRIPTION	BODY/PARAMS	ACCESS
GET	`/orders/delivery`	List orders with status `READY_FOR_PICKUP` near partner	Possibly `?lat=..., lon=..., radius=...`	DELIVERY_PARTNER
PATCH	`/orders/:id/accept-delivery`	Partner claims order, set `OUT_FOR_DELIVERY`	N/A	DELIVERY_PARTNER
PATCH	`/orders/:id/delivered`	Confirm delivery, set `DELIVERED`	{ `paymentCollected`: true } if COD	DELIVERY_PARTNER

Payments

METHOD	ENDPOINT	DESCRIPTION	BODY/PARAMS	ACCESS
POST	`/payments/razorpay/webhook`	Webhook for payment status updates (Razorpay)	Payload from Razorpay (verify signature)	Public (secure verify)
GET	`/payments/:orderId`	Get payment info for a specific order	URL param: `:orderId`	Auth (Owner/Admin)
POST	`/payments/pay` (optional)	Initiate payment session if using a gateway flow	{ `orderId`, `amount`, ... }	CUSTOMER

Ratings & Reviews

METHOD	ENDPOINT	DESCRIPTION	BODY/PARAMS	ACCESS
POST	`/ratings`	Submit rating for a delivered order (restaurant + delivery)	{ `orderId`, `restaurantRating`, `deliveryRating`, `comment` }	CUSTOMER
GET	`/ratings/restaurant/:rid`	Get all ratings for a restaurant	URL param: `:rid`	Public/Owner/Admin
GET	`/ratings/delivery-partner/:pid`	Get all ratings for a Delivery Partner	URL param: `:pid`	Delivery Partner (self)/Admin

Super Admin (Advanced)

Additional advanced routes might include:

- `GET /admin/dashboard`: aggregated stats
- `GET /admin/orders`: all orders in the system
- `GET /admin/restaurants`: manage all restaurants
- `PATCH /admin/...`: system-level changes, etc.

Implementation Notes

- **Order Status Flow**: Typically `PENDING` → `ACCEPTED` / `REJECTED` → `PREPARING` → `READY_FOR_PICKUP` → `OUT_FOR_DELIVERY` → `DELIVERED` (or `CANCELED`)
- **Role-based Middleware**: check `req.user.role` to allow only relevant actions.
- **Payments**: Integrate with your chosen gateway (Razorpay, Stripe, etc.).
- **Geolocation**: Use lat/long for customers, restaurants, and (optionally) real-time delivery partner location.
- **Coupons**: Validate during order creation or with a dedicated coupon validation endpoint.

Conclusion

This Markdown document:

1. **Wraps text** in the “Description” and “Body/Params” columns.
2. Avoids large code samples that might break table formatting.
3. Lets you easily export to PDF from **Google Docs** or **MS Word** once you paste these tables (and ensure “AutoFit” / “Wrap text” is enabled).
4. Or you can open it in a **Markdown editor** (Typora, Obsidian, etc.) that supports PDF export.

With this, you have a **full Node.js + Express API specification** for your Food Delivery System—no more overflowing table cells!