Project Documentation: Raftlabs Full-Stack Assessment

This document provides a comprehensive overview of the project structure, architectural decisions, and the development process, including the integration of AI tools.

---

1. Code Walkthrough & Structure

The project is structured as a monorepo-style setup with two primary directories: `Client` (Frontend) and `Server` (Backend).

Backend (Server)

Built with **Node.js, Express, and MongoDB (Mongoose)** using **TypeScript**.

- **`src/models`**: Mongoose schemas defining the data layer (User, Restaurant, Dish, Menu, Order, Review).
- **`src/services`**: The core business logic layer. Services interact with models and perform data transformations, ensuring that controllers remain thin and focused only on request/response handling.
- **`src/controllers`**: Orchestrates the flow between the routes and services.
- **`src/routes`**: Definitive routing structure with semantic naming (e.g., `/api/restaurants`, `/api/orders`).
- **`src/middleware`**: Global and route-specific logic for Authentication (JWT), Error Handling, and Validation (Yup).
- **`src/utils`**: Global utilities like `ApiResponse`, `ApiError`, and database seeding scripts.

Frontend (Client)

Built with **React, Vite, and Tailwind CSS**.

- **`src/shared/api`**: Centralized API client (Axios) with pre-configured interceptors and structured endpoint managers.
- **`src/features`**: Redux Toolkit slices for state management (Auth, Cart, Common).
- **`src/shared/hooks`**: Custom hooks (e.g., `useSocket`, `useDashboard`) that abstract logic away from UI components, following the "Headless UI" pattern.
- **`src/layouts`**: Higher-level layout wrappers for consistent navigation and UI structure.
- **`src/pages`**: Feature-specific views (Dashboard, Restaurant Details, Profile, Cart).
- **`src/components/ui`**: Atomic, reusable UI components like Buttons, Loaders, and the custom

Pagination system.

---

2. Architecture & Design Choices

Design Pattern: Service-Oriented Backend

We chose a **Service Layer Architecture** to decouple business logic from the HTTP layer. This makes the code:

- **Testable**: Services can be unit-tested without mocking the entire Express request/response cycle.
- **Maintainable**: If the data source changes (e.g.) from MongoDB to another DB, only the services need modification.

State Management: Redux + Persistence

Used **Redux Toolkit** for centralized state management.

- **Persistence**: Integrated `redux-persist` for the **Shopping Cart** and **Auth Session**, ensuring user data survives page refreshes—a critical UX requirement for e-commerce.

Real-Time Experience: WebSockets

Implemented **Socket.io** (via native WebSockets) for real-time order tracking.

- **Design Choice**: Instead of polling the server every few seconds, the server "pushes" status updates directly to the client. This reduces server load and provides an "instant" feel to the user interface.

UI/UX: Modern & Premium Aesthetics

- **Tech**: Vanilla CSS combined with Tailwind CSS for rapid, scalable styling.
- **Aesthetics**: Employed modern design principles like **glassmorphism**, high-contrast typography (Inter/Outfit), and subtle micro-animations (hover transitions, loaders) to create a premium feel.
- **Responsiveness**: A mobile-first approach ensures the app works perfectly on devices of all sizes.

---

3. Development with AI Tools (Antigravity)

During development, my AI pair programmer (Antigravity) was utilized as a force multiplier across several key areas:

Automated Code Generation

- **Boilerplate**: Rapidly generated Mongoose models and Express route skeletons.
- **Logic Synthesis**: The AI assisted in brainstorming and implementing the complex pagination logic across the Dashboard and Restaurant details, ensuring the backend queries (`skip` / `limit`) stayed in sync with the frontend state.

Refactoring & Optimization

- **Code Quality**: The AI was used to refactor monolithic components into reusable hooks (like `useDashboard`) and atomic UI components.
- **Linting & Formatting**: Instrumental in setting up `Prettier` and `ESLint` configurations across the entire codebase to maintain a professional, consistent style.

Debugging & Security

- **Issue Isolation**: Helped identify and fix critical bugs, such as the `restaurantId` synchronization error in the Redux Cart slice.
- **Security Audit**: Used to scan for unprotected routes, leading to the implementation of the `authenticate` middleware and the securing of the `/seed` endpoint against production attacks.

---

4. Key Features Implemented

1. **Full Auth Flow**: Secure registration, login, and protected profiles.
2. **Smart Shopping Cart**: Multi-restaurant protection (clears cart if switching restaurants) and automatic quantity handling.
3. **Advanced Filtering**: Multi-select food type filters (Veg/Non-Veg/Vegan) and real-time search on the Dashboard.
4. **Pagination**: Custom-built pagination engine for both restaurants and menu items.
5. **Order Tracking**: A real-time system that simulates order progression (Preparing -> Out for Delivery -> Delivered) with live socket notifications.

Developed for the Raftlabs Full-Stack Assessment - 2026