1. **Project Name**: Order Management System

**2. Modules Covered:**

1. User Authentication Module

* Simulated login system for Admin and User
* Role-based redirection after login
* Basic session handling with in-memory storage

2. Admin Dashboard

* Admin-only area after login
* Access to full order management controls (Add, Edit, Delete)
* Option to manage users or other future features

3. User Dashboard

* User-specific dashboard
* View-only access or limited operations (optional customization)
* Displays relevant order information

4. Orders Management Module (CRUD)

* View all freight orders in a table
* Add new freight order
* Edit existing order details
* Delete orders
* All changes synced with backend via Angular HttpClient

5. Backend API Module (Node.js with Express)

* REST API to handle CRUD operations on freight orders
* Uses in-memory JSON data (mocked) — no database
* Endpoints:
  + GET /api/orders
  + POST /api/orders
  + PUT /api/orders/:id
  + DELETE /api/orders/:id

6. Routing and Navigation Module

* Angular Router used to manage navigation between login, dashboard, and order pages
* Route guards to protect access based on login state

7. Service Layer Module

* AuthService: Manages login simulation, stores session data
* OrderService: Communicates with backend to perform API calls

8. UI/UX Module

* Responsive layout using Bootstrap (or Angular Material if used)
* Clean navigation structure
* Interactive table and form interface**s**

**3.Technologies used:**

🔹 Frontend (Angular)

| Technology | Purpose |
| --- | --- |
| Angular 15+ | Frontend framework for building SPA |
| TypeScript | Language used for Angular components/services |
| HTML/CSS | Template and styling of the UI |
| Bootstrap 5 *(optional)* | Responsive design and layout styling |
| Angular Router | Route-based navigation between components |
| HttpClientModule | Making API calls to backend |
| Reactive Forms / FormsModule | Handling form inputs and validation |

🔹 Backend (Node.js + Express)

| Technology | Purpose |
| --- | --- |
| Node.js | JavaScript runtime for server-side code |
| Express.js | Web framework for building REST APIs |
| CORS Middleware | Allow frontend to connect to backend |
| Body-Parser / express.json() | Parses incoming JSON request bodies |

🔹 Authentication (Simulated)

| Technology | Purpose |
| --- | --- |
| Hardcoded Credentials | Simulates admin/user login (no DB) |
| Role-based Routing | Controls access to views based on role |

🔹 Mock Data Handling

| Technology | Purpose |
| --- | --- |
| In-memory JSON Objects | Simulated database for storing orders |
| JavaScript Array | Used for order CRUD without persistence |

🔹 Development & Tools

| Technology | Purpose |
| --- | --- |
| Visual Studio Code | Code editor |
| Node Package Manager (NPM) | Manages packages and dependencies |
| Angular CLI | Generates components, services, modules |
| Postman | API testing tool (for backend routes) |

**4. FLOW DIAGRAM:**

+---------------------+

| User/Admin |

+---------------------+

|

v

+---------------------+

| Login Page |

| Angular Component |

+---------------------+

|

+---------+---------+

| |

+------+-----+ +-------+------+

| Admin Login | | User Login |

+------------+ +--------------+

| |

v v

+--------------------+ +--------------------+

| Admin Dashboard | | User Dashboard |

| (Angular Component)| | (Angular Component)|

+--------------------+ +--------------------+

| |

+---------+---------+

|

v

+--------------------------+

| Orders Component |

| (CRUD: Add/Edit/Delete) |

+--------------------------+

|

v

+-----------------------------------------+

| Angular OrderService (HTTP to backend) |

+-----------------------------------------+

|

v

+----------------------------------+

| Node.js Express Backend API |

| - GET /api/orders |

| - POST /api/orders |

| - PUT /api/orders/:id |

| - DELETE /api/orders/:id |

+----------------------------------+

|

v

+-----------------------------+

| In-memory JSON data |

| |

**5. How each module is developed:**

**1. Login Module (Authentication Simulation)**

* **Purpose:** To simulate secure login for Admin and User.
* **Implementation:**
  + Created a LoginComponent in Angular.
  + Form accepts username and password, validated with Angular forms.
  + On submission, data is passed to AuthService, which checks hardcoded credentials.
  + Depending on role, users are redirected using Angular Router.
* **Tools/Tech:** Angular Forms, TypeScript, Router

**2. Dashboard Module (Role-Based Views)**

* **Purpose:** Provide distinct views for Admin and User after login.
* **Implementation:**
  + Used Angular routing to display AdminDashboardComponent or UserDashboardComponent.
  + Guarded routes using a custom AuthGuard that checks login role.
  + Used AuthService to determine who is logged in.
* **Tools/Tech:** Angular Router, Services, Route Guards

**3. Order Management Module (CRUD Operations)**

* **Purpose:** Allow Admin (and optionally User) to manage freight orders.
* **Implementation:**
  + Angular OrdersComponent handles display and form for adding/editing orders.
  + Connected to backend using Angular HttpClient through OrderService.
  + Backend is a simple Express.js server with mocked order data stored in a JS array.
  + APIs:
    - GET /api/orders – fetch all orders
    - POST /api/orders – add new order
    - PUT /api/orders/:id – edit order
    - DELETE /api/orders/:id – delete order
* **Tools/Tech:** Angular, Node.js, Express, TypeScript, HTML/CSS

**4. Routing Module**

* **Purpose:** Manage navigation and route protection.
* **Implementation:**
  + Defined routes in app-routing.module.ts for login, dashboard, orders, etc.
  + Added route guards to prevent access without login.
* **Tools/Tech:** Angular Router, canActivate guard

**5. Service Layer Module**

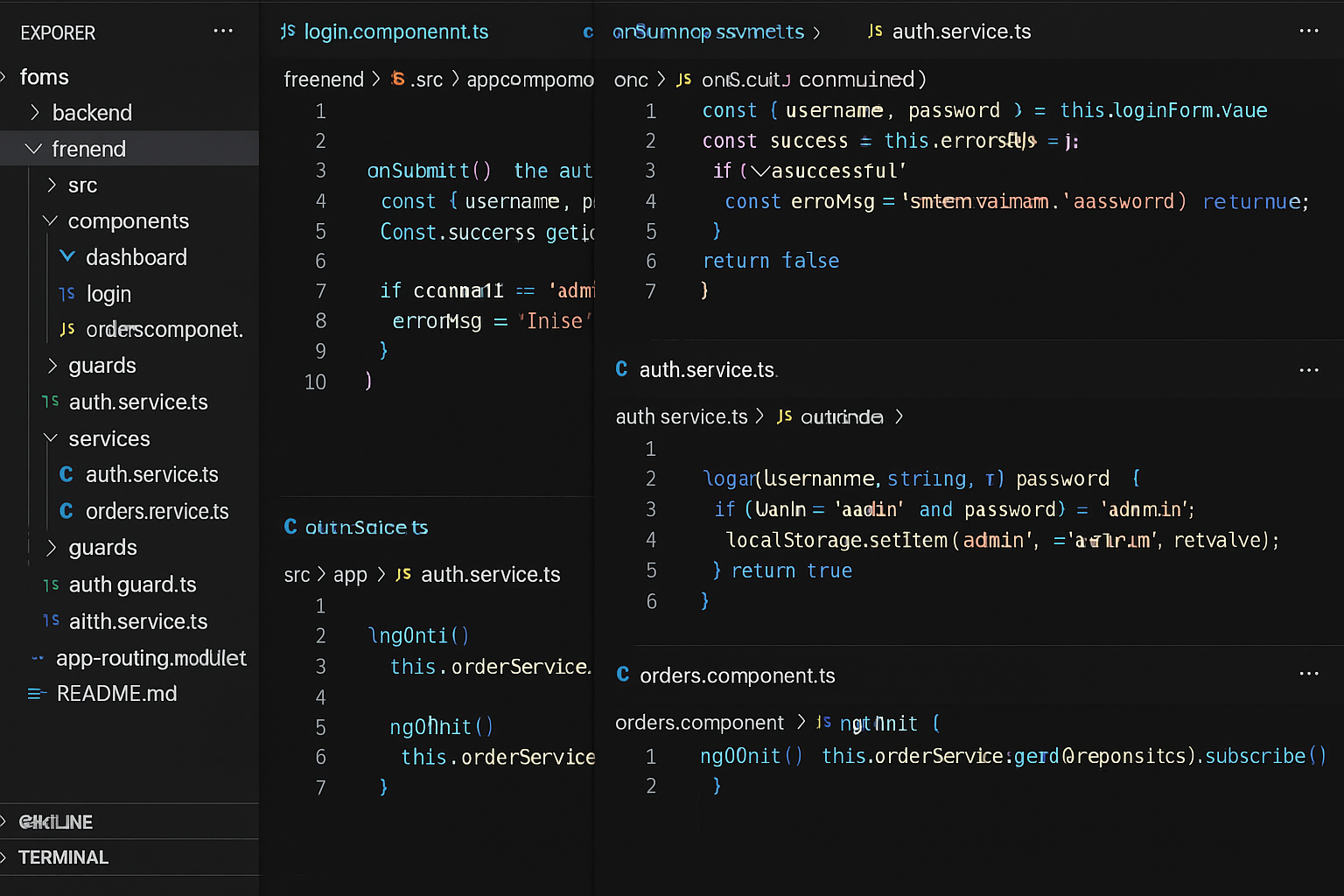
* **Purpose:** Encapsulate logic for authentication and API requests.
* **Implementation:**
  + AuthService handles login state, role detection, and logout.
  + OrderService wraps HttpClient calls to communicate with backend.
  + Both services are injected into components using Angular dependency injection.
* **Tools/Tech:** Angular Services, Dependency Injection

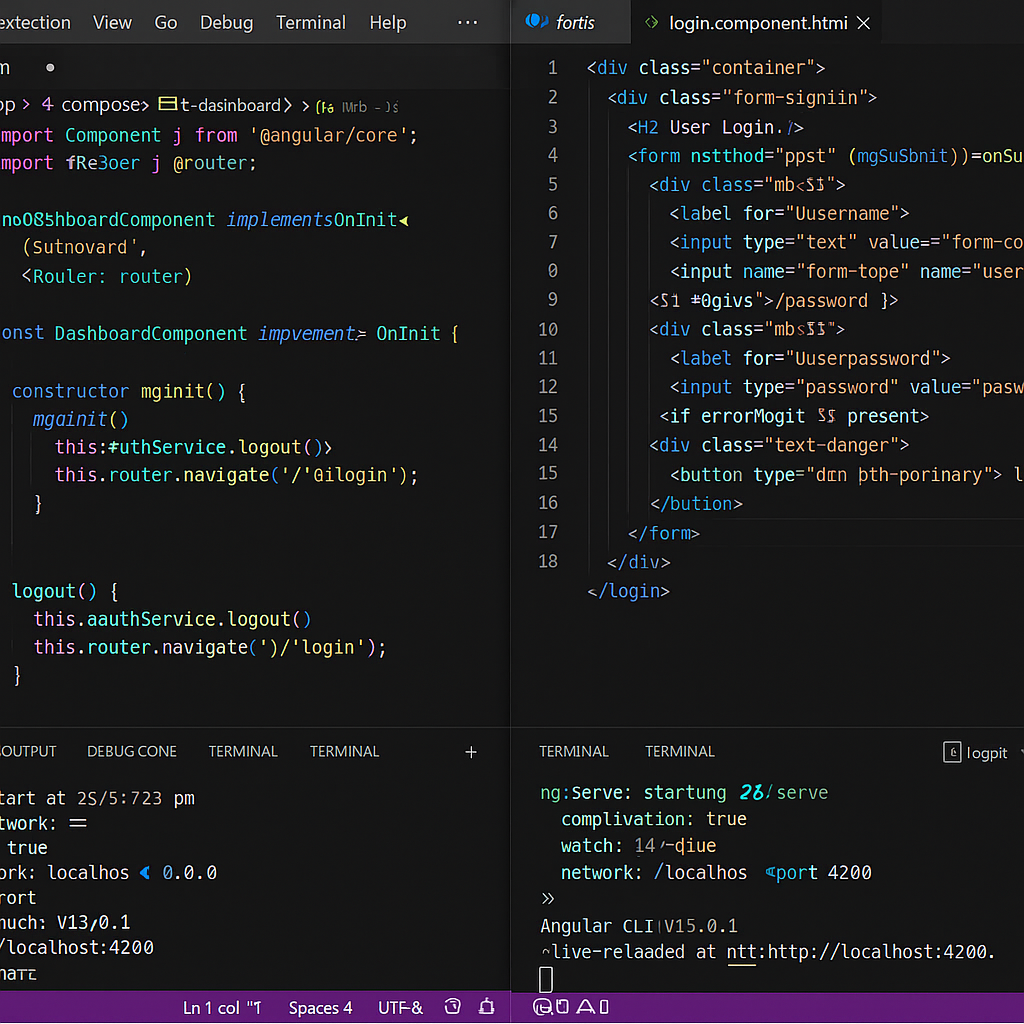
**6. Backend API Module (Mock Backend)**

* **Purpose:** Serve as a REST API to perform CRUD operations.
* **Implementation:**
  + Developed using Express.js.
  + Stored freight orders in an in-memory JS array (mock database).
  + Each route handles corresponding CRUD operation.
  + Used CORS and JSON middleware to allow frontend interaction.
* **Tools/Tech:** Node.js, Express, CORS, body-parser

**7. UI/UX Module**

* **Purpose:** Provide a clean, responsive, user-friendly interface.
* **Implementation:**
  + Used Angular components with Bootstrap for layout and styling.
  + Forms have validation (required fields).
  + Tables used for displaying order data.
* **Tools/Tech:** HTML, CSS, Bootstrap, Angular Template Binding

**6. Screenshots  
  
**

****