FULL STACK PROJECT IDEA

## # Multi-Vendor Shop System Documentation

**1. Introduction**

The Multi-Vendor Shop System is a full-stack web application where customers can browse and purchase products from various vendors. Vendors manage their own shops, and a superadmin oversees all operations. The system includes location-based filtering to help users find nearby shops.

**2. Technology Stack**

* **Frontend:** React (Vite), Tailwind CSS, React Router, Axios
* **Backend:** Node.js, Express.js
* **Database:** MongoDB (MongoDB Atlas)
* **Authentication:** JWT (JSON Web Tokens), bcrypt.js (password hashing)
* **State Management:** Context API / Redux
* **Additional Features:** Geolocation API, MongoDB Geospatial Queries

**3. System Features & Roles**

**User Roles:**

1. **Customer**:
   * Browse shops & products
   * Add products to cart & checkout
   * View past orders
2. **Vendor**:
   * Manage their shop (add/edit products, update stock)
   * View customer orders for their shop
3. **Superadmin**:
   * Manage all shops & vendors
   * Remove products or shops
   * Manage users (approve vendors, remove accounts)

**4. Project Roadmap**

**Phase 1: Project Setup**

1. Initialize Backend (Node.js + Express)
2. Initialize Frontend (React + Vite, Tailwind CSS)
3. Set up MongoDB Atlas database

**Phase 2: Authentication & Authorization**

1. Implement role-based authentication (JWT, bcrypt.js)
2. Create protected API routes for each role

**Phase 3: Shop & Product Management**

1. Create shops API (CRUD operations, location filtering)
2. Implement product management (vendors can only edit their own products)

**Phase 4: Customer Features**

1. Build cart & checkout system
2. Implement orders API for customers and vendors

**Phase 5: Location-Based Filtering**

1. Integrate Geolocation API for shop filtering
2. Implement MongoDB Geospatial Queries to find nearby shops

**Phase 6: Admin & Vendor Dashboard**

1. Superadmin dashboard for shop and vendor management
2. Vendor dashboard for managing products and orders

**Phase 7: Deployment & Testing**

1. Test APIs using Postman
2. Deploy backend (Render/Railway/Vercel) and frontend (Vercel/Netlify)
3. Deploy MongoDB on MongoDB Atlas

**5. Future Enhancements**

* **Real-time Order Updates** (WebSockets / Socket.io for instant notifications)
* **Payment Gateway Integration** (Stripe/Razorpay for online payments)
* **Google Maps API** (Visual representation of shop locations)
* **Admin Analytics Dashboard** (Track sales, user activities, and shop performance)

**6. Conclusion**

The Multi-Vendor Shop System provides a seamless experience for customers, vendors, and administrators. With role-based access control and location-based filtering, the platform enhances user convenience while ensuring data security and operational efficiency. Future enhancements such as real-time order updates and payment integration will further improve usability and performance.

**7. References & Tools**

* **Postman** (API Testing)
* **MongoDB Atlas** (Cloud Database Management)
* **Vercel / Netlify** (Frontend Deployment)
* **Render / Railway** (Backend Deployment)
* **GitHub** (Version Control)

**📂 Final Optimized Directory Structure with Details:**

LOCALCART/

├── backend/ <-- Backend API (Node.js + Express)

│ ├── controllers/ <-- Handles business logic for each route

│ │ ├── authController.js <-- Login, signup, and user auth

│ │ ├── productController.js <-- CRUD for products

│ │ └── userController.js <-- User profile and management

│ ├── library/ <-- Custom helper functions

│ │ └── helper.js <-- Utility functions for common tasks

│ ├── models/ <-- Database models/schemas (MongoDB)

│ │ ├── Order.js <-- Order schema

│ │ ├── Product.js <-- Product schema

│ │ └── User.js <-- User schema

│ ├── routes/ <-- Defines API endpoints

│ │ ├── authRoutes.js <-- Auth-related routes (login, signup)

│ │ ├── productRoutes.js <-- Product-related routes

│ │ └── userRoutes.js <-- User-related routes

│ ├── middlewares/ <-- Custom middleware functions

│ │ ├── authMiddleware.js <-- Protects routes (JWT-based auth)

│ │ └── errorHandler.js <-- Error handling middleware

│ ├── utils/ <-- Utility/helper functions

│ │ └── logger.js <-- Logs errors and events

│ ├── config/ <-- Configuration files

│ │ └── database.js <-- MongoDB connection setup

│ ├── index.js <-- Main server entry point

│ ├── package-lock.json <-- Auto-generated by npm

│ └── package.json <-- Backend dependencies and metadata

├── frontend/ <-- Frontend (React + Vite)

│ ├── public/ <-- Public assets (static files)

│ │ └── vite.svg <-- Vite logo (optional)

│ ├── src/ <-- Main frontend source folder

│ │ ├── assets/ <-- Static assets (images/icons)

│ │ │ ├── images/ <-- Store images

│ │ │ └── icons/ <-- SVG/PNG icons

│ │ ├── components/ <-- Reusable UI components

│ │ │ ├── Card.jsx <-- Product display card

│ │ │ ├── Navbar.jsx <-- Navigation bar

│ │ │ └── Footer.jsx <-- Footer section

│ │ ├── pages/ <-- Page-level components

│ │ │ ├── Home.jsx <-- Homepage layout

│ │ │ ├── Cart.jsx <-- Shopping cart page

│ │ │ └── ProductDetail.jsx <-- Product details page

│ │ ├── hooks/ <-- Custom hooks

│ │ │ └── useFetch.js <-- Fetch API data easily

│ │ ├── context/ <-- Context API for global state

│ │ │ └── CartContext.jsx <-- Manages cart state globally

│ │ ├── services/ <-- API services and utilities

│ │ │ ├── api.js <-- Handles API requests

│ │ │ └── authService.js <-- Manages auth requests

│ │ ├── utils/ <-- Utility/helper functions

│ │ │ └── formatDate.js <-- Formats date for UI

│ │ ├── App.css <-- Global CSS styles

│ │ ├── App.jsx <-- Main React app component

│ │ ├── index.css <-- TailwindCSS/index styles

│ │ └── main.jsx <-- Entry point for rendering app

│ ├── .gitignore <-- Ignores files in Git

│ ├── eslint.config.js <-- Linter configuration

│ ├── index.html <-- Main HTML template

│ ├── package-lock.json <-- Auto-generated by npm

│ ├── package.json <-- Frontend dependencies and metadata

│ ├── README.md <-- Project overview and instructions

│ └── vite.config.js <-- Vite configuration

├── docs/ <-- Documentation

│ └── API\_DOCS.md <-- API documentation

├── FULL STACK PROJECT IDEA.docx <-- Project idea/proposal

├── LICENSE.md <-- License information

└── README.md <-- Main project overview

**📚 Folder/Files Explanation in Simple Terms:**

**🎯 Backend:**

* controllers/ – Handles what happens when a request hits an API endpoint.
* library/ – Custom helper functions used in multiple places.
* models/ – Defines MongoDB schemas for Orders, Products, and Users.
* routes/ – API endpoints that map URLs to controllers.
* middlewares/ – Handles things like checking if a user is logged in or error handling.
* utils/ – Miscellaneous functions (e.g., logging events).
* config/ – Contains DB connection setup and other configurations.

**⚡️ Frontend:**

* public/ – Stores static files like logos or favicons.
* src/ – Main folder where the frontend code lives.
* assets/ – Images/icons used in the UI.
* components/ – Reusable UI elements like Card or Navbar.
* pages/ – Main page components (Home, Cart, ProductDetail).
* hooks/ – Custom hooks for reusing logic (useFetch).
* context/ – Manages global state like Cart or Auth.
* services/ – API requests and external service handlers.
* utils/ – Utility functions like date formatting.
* 📚 Other Important Folders/Files:
* docs/ – Contains project documentation like API guides.
* LICENSE.md – License file to protect your work.
* README.md – Project overview, setup, and usage instructions.