**Full Stack Development with MERN**

**Project Documentation**

**1. Introduction**

**Project Title:** “Grocery Web App”

You will also introduce the “MERN stack”, explaining that it uses:

* MongoDB for the database.
* Express.js as the backend framework.
* React.js for the frontend interface.
* Node.js to handle server-side logic.

**2. Project Overview**

**Grocery web app is a modern, full-stack web application designed to simplify grocery shopping by connecting customers with local stores.**

**Purpose:** Provide a convenient platform for online grocery

• Enable seamless inventory and order management for vendors.

• Enhance user experience through intuitive design and features.

**Features:**

•User registration and login with secure authentication.

•Intuitive product browsing and search capabilities.

•Shopping cart with add, update, and remove options.

•Secure checkout and payment processing.

•Admin dashboard for managing inventory, orders, and users

**3. Architecture**

**Frontend:**

**1.Component Structure:**

• Use a modular design with reusable components like Header, Footer, ProductCard, and CartItem.

• Manage state efficiently using React hooks and a global context for authentication and cart data.

• Separate container components (handle logic) from presentational components (UI rendering).

**2. Routing:**

• Implement React Router for client-side navigation with routes like Home, Products, Cart, and Checkout.

• Use protected routes to restrict access to authenticated users.

• Optimize performance with lazy loading for route components.

**3. API Integration:**

• Centralize API calls in a services/ directory using Axios for HTTP requests.

• Implement interceptors for handling authentication tokens and error management.

**4. Performance Optimization:**

• Use code splitting and lazy loading to reduce initial load time.

• Optimize rendering with React.memo and useMemo for expensive computations.

• Virtualize long lists using libraries like react-window.

**Backend:**

**1.Server Setup:**

• Use Express.js as the web framework for handling HTTP requests and middleware.

• Configure essential middleware like body-parser for parsing requests, CORS for cross-origin requests, and morgan for logging.

• Connect to the database using Mongoose (for MongoDB) or a similar ORM/ODM.

**2. API Structure:**

• Follow RESTful design principles with endpoints for users, products, orders, etc. Example:

GET /api/products - Fetch all products.

POST /api/orders - Place a new order.

Organize logic into controllers and define routes in a modular structure (e.g., routes/userRoutes.js).

**3. Data Validation:**

• Validate incoming data using libraries like Joi or express-validator.

• Implement middleware for validation to ensure request data integrity before processing.

**4. Security:**

• Use JWT (JSON Web Tokens) for secure user authentication and session management.

• Implement rate limiting to prevent abuse and helmet middleware for securing HTTP headers.

• Protect against CSRF and sanitize inputs to avoid injection attacks.

**Database:**

**Data Schema**

**1.User Schema:**

**Fields:**

id (Primary Key)

name (string)

email (unique, string)

password (hashed, string)

address (string)

phone (string)

created\_at (timestamp)

updated\_at (timestamp)

**2. Product Schema:**

**Fields:**

id (Primary Key)

name (string)

description (string)

price (number)

category (string)

stock (number)

image\_url (string)

is\_available (boolean)

created\_at (timestamp)

updated\_at (timestamp)

**3. Order Schema:**

**Fields:**

id (Primary Key)

user\_id (Foreign Key, references User)

items (array of objects: { product\_id, quantity })

total\_amount (number)

status (string: pending, shipped, delivered)

delivery\_address (string)

created\_at (timestamp)

updated\_at (timestamp)

**4. Category Schema:**

**Fields:**

id (Primary Key)

name (string)

description (string)

**5. Review Schema:**

**Fields:**

id (Primary Key)

user\_id (Foreign Key, references User)

product\_id (Foreign Key, references Product)

rating (number)

comment (string)

created\_at (timestamp)

**Database Interactions:**

**1. Connecting to the Database**

• Use Mongoose to connect to MongoDB with configurations for error handling and connection retries.

**2. User Operations**

• Create: Save user details (name, email, hashed password, etc.) during registration.

• Read: Query the database to fetch user details by unique fields like email for login or profile access.

**3. Product Operations**

• Create: Add new products with details like name, description, price, and stock.

• Read: Retrieve products by category or availability for display to users.

**4. Order Operations**

• Create: Save order details like user ID, products ordered, total amount, and status.

• Read: Fetch order history for a user or manage order status updates.

**5. Review Operations**

• Create: Save reviews with user ID, product ID, rating, and comments.

• Read: Fetch reviews for a specific product to display ratings and feedback.

**4. Setup Instructions**

**Frontend Setup:**

1. Navigate to the client directory:

cd client

2. Install dependencies:

npm install

3. Start the development server:

npm run dev

Backend Setup:

1. Navigate to the server directory:

cd server

2. Install dependencies:

npm install

3. Start the backend server:

npm start

**5. Folder Structure**

**Client:**

public/: Contains static files like index.html.

src/: Main source directory for the frontend.

components/: Reusable UI components.

index.js: Entry point for the React application.

postcss.config.js: Configuration for PostCSS.

vercel.json: Deployment configuration for Vercel.

jsconfig.json: Configuration file for JavaScript settings.

.gitignore: Specifies ignored files for Git.

vite.config.js: Configuration for Vite build tool.

package.json: Contains project dependencies and scripts.

package-lock.json: Locks dependency versions.

**Server**

src/: Source files for backend logic.

API endpoints for user, product, and order management.

vercel.json: Deployment settings for Vercel.

gitignore: Specifies ignored files for Git.

package.json: Backend dependencies and scripts.

package-lock.json: Locks dependency versions.

**6. Running the Application**

**Frontend:**

cd .\ Harvest Hub shop\NxtGen grocery-main\NxtGen\_grocery-main\client\

npm install

npm run dev

http://localhost:5174/

**Backend:**

npm run dev

http://localhost:5000](http://localhost:5000)

npm run

**7. API Documentation**

**User management**

**Registration:** A POST request to /api/register allows users to register by providing their username, email, and password.

**Login:** A POST request to /api/login authenticates users and returns a JWT token.

**Profile:** A GET request to /api/user retrieves the profile information of the logged-in user using the token in the request header.

A user submits the following data to log in:

Email: [john@example.com](mailto:john@example.com)

Password: "securePassword123"

**Product Management**

**Get Products:** A GET request to /api/products returns a list of all products.

**Add Product:** A POST request to /api/products allows admins to add new products by providing product details.

**Update Product:** A PUT request to /api/products/:id updates the details of an existing product.

**Get Single Product:** A GET request to /api/products/:id retrieves details of a specific product by its ID.

**Order Management**

**Get Orders:** A GET request to /api/orders retrieves a list of orders placed by the logged-in user.

**Place Order:** A POST request to /api/orders creates a new order by providing product IDs and quantities.

**Get Order Details:** A GET request to /api/orders/:id retrieves details of a specific order.

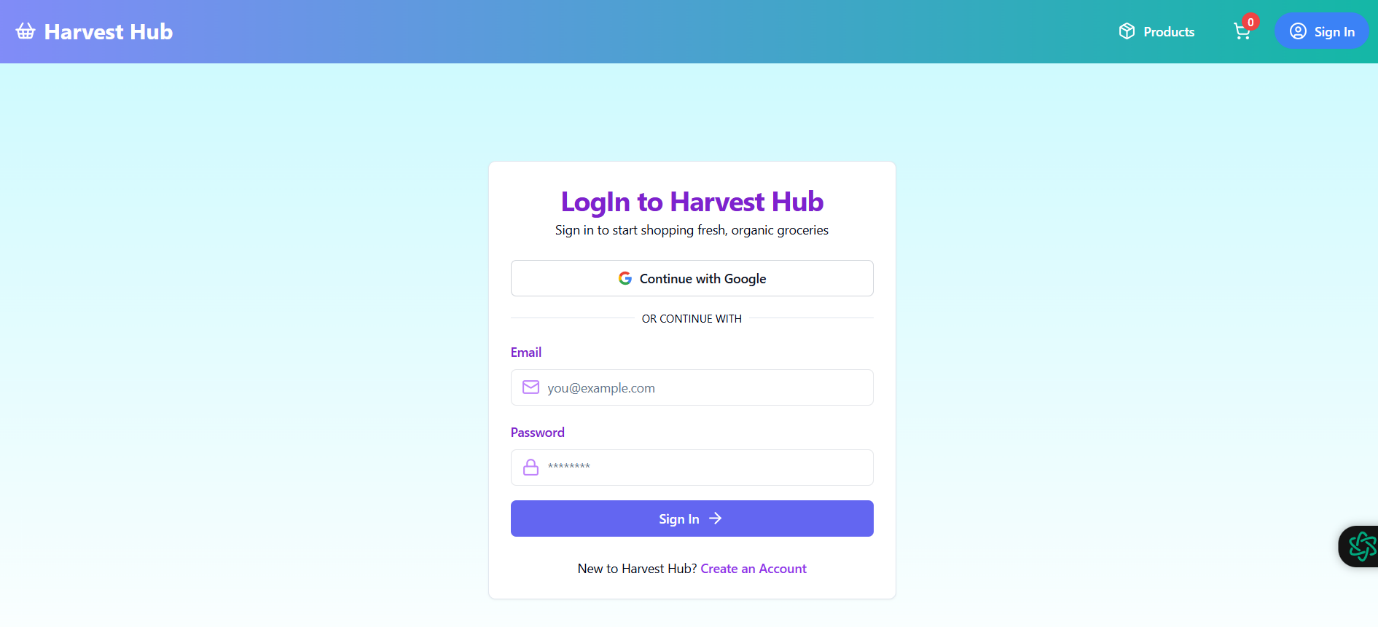
**8. Authentication**

Authentication is the process of verifying the identity of a user. In this project, authentication is primarily handled through a combination of username/email and password.

**Process:**

* **User Registration:** When a new user registers, they provide a username, email, and password. The password is securely hashed before being stored in the database to protect user credentials.
* **User Login:** During the login process, users submit their email and password. The backend verifies these credentials by comparing the provided password (after hashing) with the stored hash in the database.
* **Token Generation:** Upon successful authentication, the server generates a JSON Web Token (JWT). This token contains encoded user information and is signed to prevent tampering. The token is then sent back to the client.

**9. User Interface**



**10. Testing**

**User Management Tests:**

Registration: Verify that users can successfully register with valid data and receive appropriate error responses for invalid data.

Login: Ensure that valid login requests return a JWT token, and invalid credentials result in an error.

Profile: Test that the profile endpoint correctly retrieves user data when a valid token is provided, and returns an error for missing or invalid tokens.

**Product Management Tests:**

Product Listing: Ensure that the GET /api/products endpoint returns the correct list of products.

Add Product: Test adding products with valid data and handle invalid inputs or missing fields.

Update Product: Verify that product updates work and return the correct response.

Single Product: Check that a valid product ID returns the correct product details.

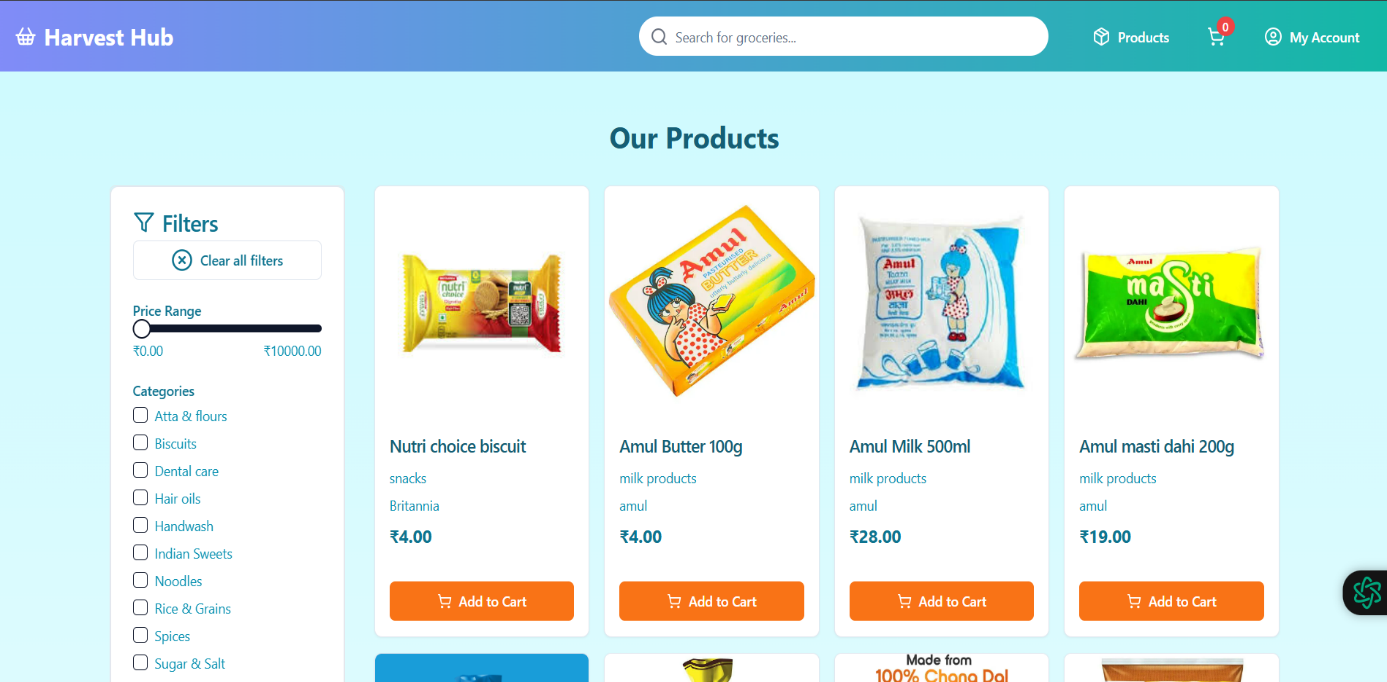
**Order Management Tests:**

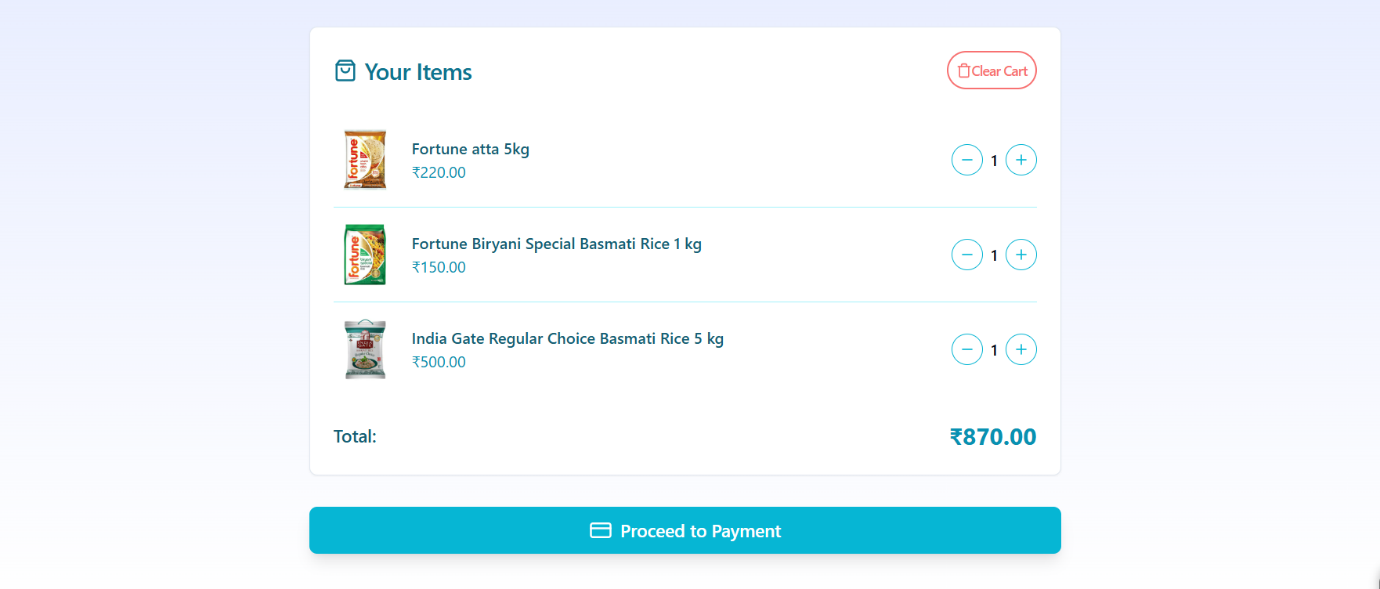
Get Orders: Verify that the GET /api/orders endpoint returns orders for the logged-in user.

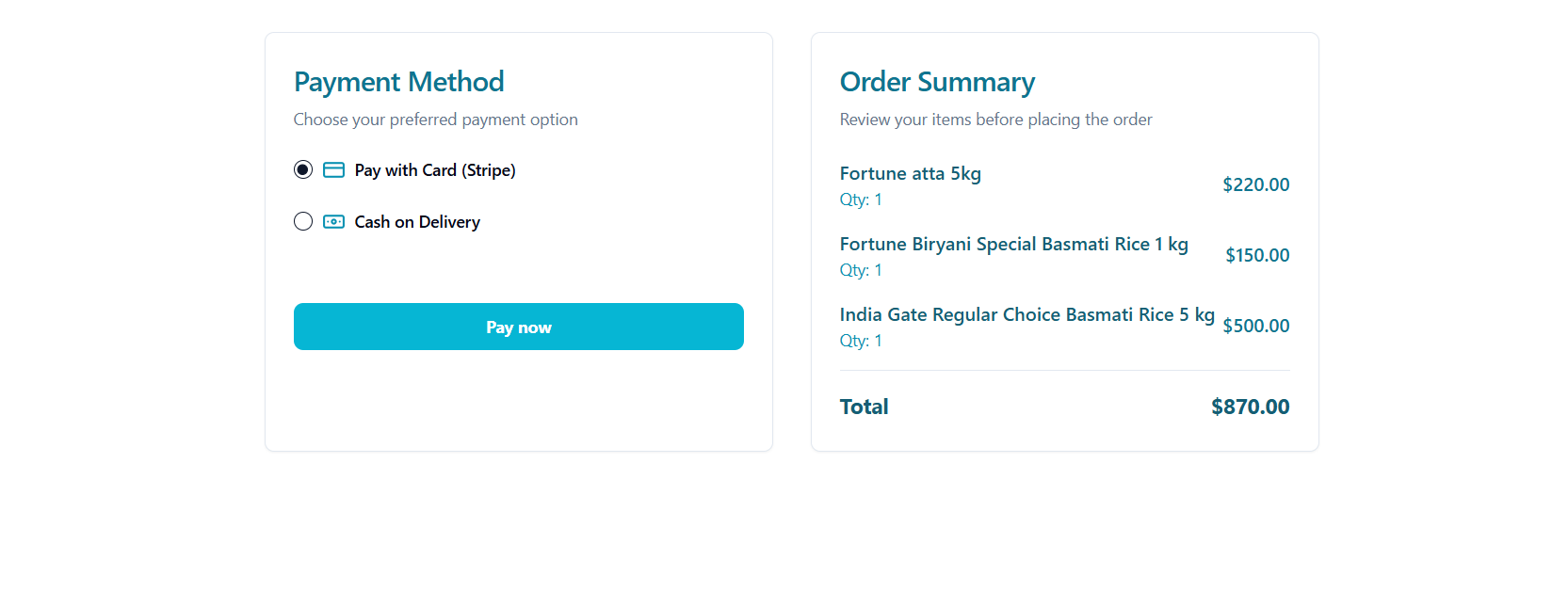
Place Order: Ensure orders can be placed correctly, and appropriate error handling is in place for invalid product data.

Order Details: Test retrieving details for specific orders by ID.

**11. Screenshots or Demo**



****

****

**12. Known Issues**

**Session Timeouts:**

**Issue:** Users may experience unexpected logouts after a period of inactivity.

**Possible Solution:** Adjust session timeout settings on the server or implement a refresh token mechanism to maintain user sessions.

**Payment Processing Delays:**

**Issue:** Some payment transactions may take longer than expected to process, especially during high traffic periods.

**Possible Solution:** Implement a status indicator for users to track payment progress, and consider optimizing the payment gateway integration.

**Mobile Responsiveness:**

**Issue:** The app may not be fully responsive on smaller screens or certain mobile devices.

**Possible Solution:** Review and refine the CSS for mobile responsiveness using media queries.

**13. Future Enhancements**

**1. AI Recommendations:** Introduce personalized product suggestions based on user behaviuor.

**2. Voice Search:** Enable voice-activated search for better accessibility.

**3. Dark Mode:** Add a dark mode option for user interface customization.

**4. Multi-language Support:** Provide the app in multiple languages to cater to a broader audience.

**5. Advanced Filtering:** Implement more refined product filters (e.g., by dietary preferences, ratings).

**6. Subscription Service:** Offer subscription options for recurring orders, such as weekly grocery deliveries.