**Project Documentation**

**1. Project Overview**

This project is a marketplace application with separate backend and frontend implementations. The backend handles server-side logic, APIs, and database interactions, while the frontend provides a user-friendly interface for interacting with the marketplace.

**Technologies Used:**

* Backend: Node.js, Express, MongoDB, Redis, Stripe, Cloudinary
* Frontend: React, Tailwind CSS, Vite, Axios

**2. Backend**

The backend is organized to handle various functionalities efficiently using controllers, middleware, models, and routes.

**2.1 Application Flow**

Start Server (server.js)

|

+--> Load Middleware (auth.middleware.js, etc.)

|

+--> Connect to Database (db.js, redis.js)

|

+--> Handle API Requests

|

+--> Route to Controllers

|

+--> Controller Logic (auth.controller.js, product.controller.js, etc.)

|

+--> Access Models (user.model.js, product.model.js)

|

+--> External Integrations (stripe.js, cloudinary.js)

**2.2 Controllers**

Controllers manage specific aspects of the application's backend functionality.

* **analytics.controller.js**: Handles analytics data and metrics.
* **auth.controller.js**: Manages user authentication and authorization processes.
* **cart.controller.js**: Handles shopping cart functionalities.
* **coupon.controller.js**: Manages creation, validation, and application of coupons.
* **payment.controller.js**: Integrates payment gateways like Stripe.
* **product.controller.js**: Manages product-related operations such as listing and details.

**2.3 Libraries**

Libraries include custom or third-party modules for external integrations:

* **cloudinary.js**: Handles image uploads and storage.
* **db.js**: Manages database connections (MongoDB).
* **redis.js**: Provides Redis-based caching solutions.
* **stripe.js**: Integrates Stripe for payment handling.

**2.4 Middleware**

Middleware adds extra functionality, such as request validation and user authentication:

* **auth.middleware.js**: Ensures protected routes are accessed only by authorized users.

**2.5 Models**

Data models define schemas for MongoDB:

* **coupon.model.js**: Schema for coupon data.
* **order.model.js**: Schema for orders.
* **product.model.js**: Schema for products.
* **user.model.js**: Schema for user data.

**2.6 Routes**

Routes define API endpoints for the application:

* **analytics.route.js**: Endpoints for analytics operations.
* **auth.route.js**: Endpoints for user login, registration, and token management.
* **cart.route.js**: Endpoints for managing the user’s cart.
* **coupon.route.js**: Endpoints for coupon-related operations.
* **payment.route.js**: Endpoints for handling payments.
* **product.route.js**: Endpoints for product operations.

**3. Frontend**

The frontend is a React-based application with modular and reusable components.

**3.1 Application Flow**

Start Application (main.jsx)

|

+--> Load Root Component (App.jsx)

|

+--> Configure Routing (react-router-dom)

|

+--> Render Pages (HomePage.jsx, CartPage.jsx, etc.)

|

+--> Use Components (Navbar.jsx, ProductCard.jsx)

|

+--> Trigger API Calls (axios.js)

|

+--> Update State (useCartStore.js, useProductStore.js)

**3.2 Components**

Reusable UI components:

* **AnalyticsTab.jsx**: Displays analytics-related data.
* **CartItem.jsx**: Represents an item in the shopping cart.
* **CategoryItem.jsx**: Displays a single category in the marketplace.
* **CreateProductForm.jsx**: Form for adding new products.
* **GiftCouponCard.jsx**: Displays gift coupons.
* **loadingSpinner.jsx**: A loading spinner for asynchronous operations.
* **Navbar.jsx**: The top navigation bar.
* **OrderSummary.jsx**: Summary of the user’s order.
* **PeopleAlsoBought.jsx**: Suggests products that others have bought.
* **ProductCard.jsx**: Displays a product's details.
* **ProductsList.jsx**: Lists multiple products in a category.

**3.3 Pages**

Pages define the structure of different views:

* **AdminPage.jsx**: Admin panel for managing the marketplace.
* **CartPage.jsx**: User’s shopping cart.
* **CategoryPage.jsx**: Displays products within a category.
* **HomePage.jsx**: Main landing page.
* **LoginPage.jsx**: User login.
* **PurchaseCancelPage.jsx**: Payment failure page.
* **PurchaseSuccessPage.jsx**: Payment success page.
* **SignUpPage.jsx**: User signup.

**3.4 State Management**

State management is achieved using custom hooks:

* **useCartStore.js**: Manages cart state.
* **useProductStore.js**: Manages product state.
* **useUserStore.js**: Manages user state.

**3.5 Configuration Files**

* **tailwind.config.js**: Tailwind CSS configuration.
* **vite.config.js**: Configuration for Vite, the frontend bundler.

**4. Common Files**

* **.env**: Stores environment variables (ignored in version control).
* **.gitignore**: Specifies files and directories to exclude from git.
* **LICENSE**: Contains licensing information.

**5. Public Resources**

Static assets used in the application:

* Images: Represent categories and product visuals (e.g., electronics.jpg, groceries.jpg).
* **vite.svg**: Logo for Vite, used in the frontend.

**6. API Documentation**

Detailed descriptions for each API endpoint:

* **Auth APIs**:
  + **POST /auth/login**: Logs in a user, returning a JWT token.
    - Request: { email: string, password: string }
    - Response: { token: string, user: object }
    - Errors: 401 Unauthorized, 400 Bad Request
  + **POST /auth/register**: Registers a new user.
    - Request: { name: string, email: string, password: string }
    - Response: { message: 'User registered successfully.' }
    - Errors: 400 Bad Request
* **Product APIs**:
  + **GET /products**: Fetches a list of products.
    - Response: [ { id: string, name: string, price: number, ... } ]
  + **GET /products/:id**: Fetches details of a single product.
    - Response: { id: string, name: string, price: number, description: string, ... }

**7. Database Schema Diagrams**

User:

- id: ObjectId

- name: String

- email: String

- password: String

- created\_at: Date

Product:

- id: ObjectId

- name: String

- price: Number

- category: String

- stock: Number

Order:

- id: ObjectId

- user\_id: ObjectId

- products: Array of { product\_id, quantity }

- total\_price: Number

**8. Dependencies**

Key dependencies listed in package.json include:

* **axios**: For API calls.
* **react-router-dom**: For routing in React.
* **redux**: For state management.
* **tailwindcss**: For styling.
* **vite**: Frontend bundler.

Additional backend dependencies:

* **express**: Web framework.
* **mongoose**: MongoDB object modeling.
* **redis**: Caching.
* **stripe**: Payment processing.
* **cloudinary**: Image uploads.

**9. Future Improvements**

* Enhance state management for scalability.
* Add unit tests for critical components and APIs.
* Optimize static asset delivery using a CDN.
* Automate deployment workflows using CI/CD pipelines.

This documentation outlines the project's current structure, flow, and functionality comprehensively.