Development Guide

Shula Neighborhood Equipment Library

GitHub: https://github.com/TymorIbrahim/ShulaWebApp2

Introduction

The Online Rental Store is a web-based platform designed to offer a flexible alternative to traditional retail by enabling customers to rent a variety of products. The application is built using a modern technology stack, with **ReactJS** serving as the foundation for a responsive and interactive frontend, and **MongoDB** as the NoSQL database for flexible and scalable data management. The development environment is **VSCode**, which facilitates efficient coding and debugging. This guide provides a comprehensive overview of the system's architecture, key components, and functions, focusing on the data flow and interdependencies that create a robust and user-friendly platform.

Project Structure

The application is structured into distinct, logical components, each handling a specific set of functionalities and user interactions.

1. **homepage.jsx**
    - **Purpose**: The main entry point for users, designed to engage visitors and provide a clear overview of the store's offerings.
    - **Key Functions**:
        - **Featured Products**: Displays a curated selection of products to highlight popular or newly added items.
        - **Promotional Content**: Showcases special offers, seasonal discounts, or rental package deals.
        - **Navigation**: Provides an intuitive navigation menu, directing users to the product catalog, customer dashboard, and other essential pages.
2. **productPage.jsx**
    - **Purpose**: The primary interface for Browse and interacting with the entire product catalog.
    - **Key Functions**:

        - **Product Listings**: Renders a dynamic grid or list of all available products for rent, fetching data from the backend.
        - **Search and Filter**: Implements search functionality by product name and allows filtering by categories (e.g., electronics, tools, furniture), price range, and availability status.
        - **Pagination**: Manages the display of a large number of products, loading data in chunks to improve performance.
        - **Detail View**: Clicking on a product navigates to a dedicated productDetails.jsx page (a sub-component of the main structure) for a more in-depth look.
3. **customerDashboard.jsx**
    - **Purpose**: A personalized hub for authenticated users, providing a comprehensive view of their account and rental activity.
    - **Key Functions**:
        - **Rental History**: Displays a list of both active and past rentals, including product details, rental dates, and total costs.
        - **Profile Management**: Allows customers to view and update their personal information, such as name, email, and billing address.
        - **Transaction Log**: Provides a detailed history of all transactions, including payments and refunds.
4. **adminPanel.jsx**
    - **Purpose**: An exclusive, protected interface for administrators to manage all aspects of the store's operations.
    - **Key Functions**:
        - **Inventory Management**: Enables adding new products, editing existing product details (price, description, images), and removing products from the catalog.
        - **User Management**: Provides tools to view, edit, or deactivate customer accounts.
        - **Rental Oversight**: Monitors all ongoing rentals, with the ability to manually update rental statuses or resolve disputes.
        - **Analytics Dashboard**: Displays key business metrics, such as total revenue, popular rental items, and customer trends.

Connections

The platform’s core functionality is powered by the communication between its frontend, backend, and database.

1. **Frontend-Backend Connection (using ReactJS and a REST API)**
    - The frontend is a ReactJS application that makes HTTP requests to a backend server.
    - **API Endpoints**: The backend exposes various REST API endpoints to handle different functionalities. For example:
        - GET /api/products: Fetches all products.
        - GET /api/products/:id: Retrieves a single product by its ID.
        - POST /api/rentals: Creates a new rental order.
        - PUT /api/customers/:id: Updates a customer's profile.
    - **Data Flow**: When a user performs an action for example, “add to cart”, the React component dispatches an action that makes an asynchronous API call. The backend processes the request, interacts with the database, and sends a JSON response back to the frontend, which then updates the UI.
2. **Backend-Database Connection (using MongoDB)**
    - The backend server, Node.js with Express.js, connects to the MongoDB database.
    - **Mongoose**: An ODM library like Mongoose is used to simplify database interactions. It defines structured schemas for each collection, ensuring data consistency. For example, a ProductSchema would define fields like productName (String), pricePerDay (Number), and availabilityStatus (Boolean).
    - **CRUD Operations**: The backend’s API handlers use Mongoose models to perform CRUD operations on the MongoDB collections, such as Product.find({}) to get all products or Rental.create(newRentalData) to save a new rental order.
3. **Session Management and Authentication (using JWT)**
    - **Login Flow**: When a customer logs in, the backend authenticates their credentials against the customers collection. If successful, it generates a **JSON Web Token (JWT)** containing the user's ID and role (customer or admin) and sends it to the frontend.
    - **Token Storage**: The frontend stores this JWT securely, often in localStorage.
    - **Protected Routes**: For any API requests to sensitive data (the customerDashboard or adminPanel), the frontend includes the JWT in the Authorization header of the request. The backend middleware intercepts these requests, validates the token, and grants or denies access based on the user's authentication status and role.

Data

The platform's data is organized into three primary MongoDB collections, with defined relationships between them.

1. **Product Data**
    - **Collection**: products

- **Fields**:
  - _id (ObjectId): The unique identifier for the product.
  - name (String): The name of the product.
  - description (String): A detailed description of the product.
  - pricePerDay (Number): The daily rental price.
  - category (String): The product category for example, "Power Tools," "Camping Gear".
  - imageUrl (String): The URL of the product image.
  - availabilityStatus (Boolean): Indicates whether the product is available for rent.
  - currentRenter (ObjectId, Ref: customers): A reference to the customer currently renting the item (if applicable).

2. **Customer Data**
   - **Collection**: customers
   - **Fields**:
     - _id (ObjectId): The unique identifier for the customer.
     - firstName (String): The customer's first name.
     - lastName (String): The customer's last name.
     - email (String, Unique): The customer's email address, used for login.
     - password (String): The hashed password.
     - rentalHistory ([ObjectId], Ref: rentals): An array of references to past and current rentals.
3. **Rental Data**
   - **Collection**: rentals
   - **Fields**:
     - _id (ObjectId): The unique identifier for the rental.
     - productId (ObjectId, Ref: products): A reference to the rented product.
     - customerId (ObjectId, Ref: customers): A reference to the customer who rented the item.
     - rentalDate (Date): The date the rental started.
     - returnDate (Date): The date the rental is scheduled to end.
     - totalPrice (Number): The calculated total cost of the rental.
     - status (String): The current status of the rental ("Active," "Returned," "Overdue").

Dependencies

The robust nature of the application is built on the interconnectedness of its components and data.

- **Product-Rental Status Synchronization**: There is a critical dependency between the products and rentals collections. When a new rental is created, the availabilityStatus of the corresponding product must be updated to false, and the currentRenter field should be populated. This change must be immediately reflected on the productPage.jsx to prevent double-booking.
- **Customer-Rental History**: The customerDashboard.jsx depends on the rentals collection to display a customer's rental history. The backend must query the rentals collection using the customer's ID to fetch all relevant rental documents and then use the productId references to retrieve the corresponding product details from the products collection.
- **Data Validation and Integrity**: Before any data is saved to the MongoDB database, the backend must perform validation to ensure consistency. This includes checking for required fields, verifying data types like ensuring pricePerDay is a number, and validating the format of dates and emails.

Security Best Practices

- **Password Hashing**: Instead of storing plain-text passwords, they must be hashed using a strong, one-way algorithm like **bcrypt**. This protects user credentials even if the database is compromised.
- **Access Control**: The adminPanel and its associated API endpoints must be protected by a strict access control mechanism. The backend must verify that the user's JWT contains an admin role before processing any administrative requests.
- **Input Sanitization**: To prevent common web vulnerabilities like **Cross-Site Scripting (XSS)** and **NoSQL Injection**, all user-generated input must be sanitized and validated on both the frontend and the backend. This ensures that malicious code cannot be executed or that database queries cannot be manipulated.