React Assignment Documentation

Yağız Can Çolak

Table of Contents

Project Overview

This React project is designed to demonstrate best practices and advanced web development concepts. It focuses on creating a scalable, user-friendly application with robust authentication, dynamic data handling, and simple testing. The application allows users to log in, view a list of products, and interact with detailed product information, including commenting and rating functionalities.

Technologies Used

- **React**: For building the user interface using functional components and hooks.
- **TypeScript**: Provides static typing to enhance code quality and maintainability.
- **Material-UI (MUI)**: Offers a set of React components for faster and easier web development with a consistent design.
- **Formik & Yup**: Manage form state and validation seamlessly.
- **Axios**: Handles HTTP requests with interceptors for authentication and error handling.
- **JWT (JSON Web Tokens)**: Implements secure authentication mechanisms.
- **React Router DOM**: Manages client-side routing for navigation between pages.
- **React Context API**: Manages global state for authentication, theming, and currency preferences.
- **Jest & React Testing Library**: Facilitates unit and integration testing
- **Axios Mock Adapter**: Mocks API requests for testing without a real backend.

Architecture and Design Decisions

1. **Component-Based Structure**:

   - **Separation of Concerns**: Different functionalities are broken down into reusable components (e.g., CommentCard, ProductCard, CustomAppBar), promoting maintainability and scalability.

2. **State Management with Context API**:

   - **Global State**: Utilized React Context for managing authentication (AuthContext), theming (ColorModeContext), and currency preferences (CurrencyContext), allowing state to be accessible across the application without prop drilling.

3. **Routing with React Router**:

   - **Protected Routes**: Implemented PrivateRoute to guard sensitive pages, ensuring only authenticated users can access the product list and detail pages.

   - **Dynamic Routing**: Used dynamic parameters in routes (e.g., /products/:id) to navigate to specific product details.

4. **Authentication with JWT**:

   - **Secure Sessions**: Employed JWT tokens for authenticating users, stored in localStorage to persist sessions across page refreshes. Although it'd be more secure to store it in a httponly, same-site cookie, I've decided localStorage is sufficient for the scope of this project.

   - **Mock Backend**: Used axios-mock-adapter to simulate backend responses, facilitating development and testing without a real server.

5. **Form Handling and Validation**:

   - **Formik & Yup**: Chose Formik for efficient form state management and Yup for schema-based validation.

6. **HTTP Request Management with Axios**:

   - **Interceptors**: Configured Axios interceptors to automatically attach JWT tokens to requests and handle unauthorized responses globally.

7. **UI/UX Considerations with Material-UI**:

   - **Consistent Design**: Leveraged MUI components to maintain a cohesive and responsive design across the application.

   - **User Feedback**: Incorporated components like LoadingIndicator, ErrorAlert, and NotificationSnackbar to provide real-time feedback to users.

8. **Testing Strategy**:

   - **Testing**: Employed Jest and React Testing Library to write simple unit and integration tests, ensuring components behave as expected under various scenarios.

   - **Mocking API Calls**: Used axios-mock-adapter to mock API responses, allowing tests to run in isolation without external dependencies.

Features Implemented

1. **Authentication and Session Handling**:
   - **Login Functionality**: Users can log in with predefined credentials (username: user, password: user123).
   - **Session Persistence**: Maintains user sessions across page refreshes using JWT tokens stored in localStorage.
   - **Logout Functionality**: Users can log out, which clears the session and redirects them to the login page.
2. **Product List Page**:
   - **Product Display**: Shows a list of products with name, price, rating (stars), and an image.
   - **Navigation**: Clicking on a product redirects users to the product detail page.
3. **Product Detail Page**:
   - **Detailed Information**: Displays a larger image, detailed description, formatted price with currency symbol, arrival date, total comments, and average rating.
   - **Tabs for Details and Comments**:
     - **Details Tab**: Provides comprehensive information about the product.
     - **Comments Tab**: Shows existing comments and allows users to add new comments and ratings.
   - **Image Slider**: Features a custom-built image slider to navigate through product images without external libraries.
4. **Form Validation**:
   - **Comment Form**: Uses Formik and Yup for managing and validating user comments and ratings.
   - **Login Form**: Integrates Formik and Yup to validate user input for login credentials.
5. **Global State Management**:
   - **Theme Management**: Allows toggling between light and dark modes using ColorModeContext.
   - **Currency Selection**: Manages currency preferences through CurrencyContext.
6. **Error Handling and Notifications**:
   - **Error Alerts**: Displays error messages for failed API requests.
   - **Snackbar Notifications**: Provides feedback for successful actions like adding comments.

Folder Structure

```
src/
├── App.css
├── App.tsx
├── PrivateRoute.tsx
├── __tests__
│   ├── Login.test.tsx
│   ├── ProductDetail.test.tsx
│   └── ProductList.test.tsx
├── api
│   └── axiosInstance.ts
├── assets
│   ├── images
│   └── logo.svg
├── components
│   ├── CommentCard.tsx
│   ├── CommentForm.tsx
│   ├── CustomAppBar.tsx
│   ├── ErrorAlert.tsx
│   ├── ImageSlider.tsx
│   ├── LoadingIndicator.tsx
│   ├── NotificationSnackbar.tsx
│   └── ProductCard.tsx
├── config.ts
├── context
│   ├── AuthContext.tsx
│   ├── ColorModeContext.tsx
│   └── CurrencyContext.tsx
├── custom.d.ts
├── data
│   └── products.ts
├── index.css
├── index.tsx
├── logo.svg
├── pages
│   ├── Login.tsx
│   ├── ProductDetail.tsx
│   └── ProductList.tsx
├── react-app-env.d.ts
├── reportWebVitals.ts
├── setupTests.ts
├── types
│   └── Product.ts
└── utils
    └── currencyUtils.ts
```

Key Directories and Files

- components/: Contains reusable UI components like CommentCard, ProductCard, and CustomAppBar.
- pages/: Houses page-level components such as Login, ProductList, and ProductDetail.
- context/: Implements React Contexts for authentication, theming, and currency management.
- api/axiosInstance.ts: Configures Axios with interceptors and mock adapters for handling HTTP requests.
- __tests__/: Contains test files for page components.
- utils/: Includes utility functions like currencyUtils.ts for currency conversions.
- data/products.ts: Provides mock data for products used throughout the application.
- setupTests.ts: Configures the testing environment, including mocking console methods to suppress unwanted logs during tests.

Testing

Testing Tools Used

- **Jest**: A JavaScript testing framework used for running tests.
- **React Testing Library**: Facilitates testing React components by focusing on user interactions and component behavior.
- **Axios Mock Adapter**: Mocks API requests to test components in isolation without relying on a real backend.

Implemented Tests

1. **Login Page (**Login.test.tsx**)**:
    - **Render Check**: Confirms username, password fields, and sign-in button render.
    - **Input Functionality**: Verifies user can type into username and password fields.
    - **Form Submission**: Ensures login function is called with correct input on submit.

2. **Product List Page (**ProductList.test.tsx**)**:
    - **Loading Indicator**: Verifies a loading indicator appears during product fetch.
    - **Error Handling**: Checks that an error message is shown on fetch failure.
    - **Product Rendering**: Ensures product names and prices display on successful fetch.

- **Navigation to Detail**: Confirms navigation to product detail page on product click.

3. **Product Detail Page (**ProductDetail.test.tsx**):**
   - **Loading Indicator**: Verifies a loading indicator appears during product fetch.
   - **Error Handling**: Checks that an error message is shown on fetch failure.
   - **Detail Rendering**: Ensures product details (name, price, description) display on successful fetch.
   - **Navigation**: Checks navigation behavior when no product is selected, showing products list.

Setup and Running

Prerequisites

- **Node.js** (version 14 or higher recommended)
- **npm** (comes with Node.js)

Installation Steps

1. **Clone the Repository**

- git clone https://github.com/yagizcolak/crea-store.git
- cd crea-store

2. **Install Dependencies**

- npm install

3. **Run the Application**

- npm start
- The application will start in development mode at http://localhost:3000

4. **Run Tests**

- npm test

Authentication Credentials

- **Username**: user
- **Password**: user123