

Original Equipment Manufacturer (OEM)

1. Project Overview

Application Purpose

The application is designed to streamline and manage warranty-related processes for users and administrators. It provides a comprehensive platform for handling various aspects of warranty management, including tracking, updating, and processing warranty claims.

Key Features

- **Warranty Tracking:** Monitor the status and details of warranties for different products.
- **Claim Processing:** Submit and manage warranty claims, including reviewing, approving, or rejecting claims.
- **Product Management:** Add, update, and delete product information related to warranties.
- **Batch Management:** Manage batches of products, including tracking serial numbers, batch renewals, and balance checks.
- **Vendor Management:** Oversee service centers and suppliers involved in the warranty process.
- **Settings Configuration:** Customize warranty settings and preferences based on organizational requirements.
- **Notifications:** Receive updates and notifications related to warranty status and claims.

Target Users

- **Customers:** Individuals who need to file warranty claims or track warranty information for their purchased products.
- **Administrators:** Personnel responsible for managing warranty processes, including reviewing claims, updating product details, and configuring settings.
- **Service Centers:** Vendors and service centers involved in processing and servicing warranty claims.

Technology Stack

- **Frontend:** React for building a dynamic and responsive user interface.
- **Backend:** API integration with various endpoints for managing warranty data.
- **State Management:** Redux for handling application state related to warranties and other features.
- **Styling:** SASS, Styled Components, and Material UI for a cohesive and maintainable design.

Goals

- **Efficiency:** Simplify and automate warranty management processes to reduce manual effort and errors.
- **Accessibility:** Provide an intuitive interface for users and administrators to interact with warranty information and processes.
- **Integration:** Seamlessly integrate with existing systems and processes to enhance overall functionality and user experience.

Th

2. Installation and Setup

- **Prerequisites:** Node.js, npm or yarn.
- **Installation Steps:**
 1. Navigate to the project directory: `cd warranty_application`
 2. Install dependencies: `npm install` or `yarn install`
- **Environment Variables:**
 - Not required.
- **Running the Application:**
 - For development: `npm start` or `yarn start`
 - For production build: `npm run build` or `yarn build`

3. Project Structure

- **Directory Structure:**

```
public/  
src/  
  assets/  
  Components/  
  Pages/  
  Sagas/  
  Slices/  
  Store/  
  App.jsx  
  index.js  
  ...  
package.json
```

4. Components and Pages

Overview

This section outlines the components and pages used in the application, along with their associated routes and functionalities.

Routing Configuration

Public Routes

- **Login:**
 - **Path:** /Login
 - **Component:** Login
 - **Purpose:** Allows users to log into the application.
- **Forgot Password:**
 - **Path:** /Forgot-Password
 - **Component:** ForgotPassword
 - **Purpose:** Provides functionality for users to reset their passwords.
- **Account Recovery:**
 - **Path:** /Account-Recovery
 - **Component:** RecoverAccount
 - **Purpose:** Facilitates account recovery for users.

Private Routes

- **Dashboard:**
 - **Path:** /
 - **Component:** Dashboard
 - **Purpose:** Main dashboard displaying an overview of the application.

Products

- **Product List:**
 - * **Path:** /Product-List
 - * **Component:** ProductView
 - * **Purpose:** Displays a list of products.
- **Add/Edit Product:**
 - * **Path:** /Product/:type
 - * **Path:** /Product/:type/:id
 - * **Component:** AddProduct
 - * **Purpose:** Adds or edits product details.

Warranty

- **Warranty List:**
 - * **Path:** /Warranty-List
 - * **Component:** WarrantyList
 - * **Purpose:** Shows a list of warranty packages.
- **Warranty Claim List:**
 - * **Path:** /Warranty-Claim-List
 - * **Component:** WarrantyClaimList
 - * **Purpose:** Lists warranty claims.

Batch

- **Batch List:**
 - * **Path:** /Batch-List
 - * **Component:** BatchList
 - * **Purpose:** Displays a list of product batches.
- **Add/Edit Batch:**
 - * **Path:** /Batch/:type
 - * **Path:** /Batch/:type/:id
 - * **Component:** AddBatch
 - * **Purpose:** Adds or edits batch details.

Vendors

- **Service Center List:**
 - * **Path:** /Service-Center-List
 - * **Component:** ServiceCenterList
 - * **Purpose:** Lists service centers.
- **Sales Partners List:**
 - * **Path:** /Sales-Partners-List
 - * **Component:** SalesPartnersList
 - * **Purpose:** Displays a list of sales partners.
- **Supplier List:**
 - * **Path:** /Supplier-List
 - * **Component:** SupplierList
 - * **Purpose:** Shows a list of suppliers.

Subscriptions

- **Subscription:**
 - * **Path:** /Subscription
 - * **Path:** /Subscription/:type/:mth/:nop
 - * **Component:** Subscription
 - * **Purpose:** Manages subscription information and packages.
- **Subscription List:**
 - * **Path:** /Subscription-List
 - * **Component:** SubscriptionList
 - * **Purpose:** Lists all subscriptions.
- **Subscription Package List:**
 - * **Path:** /Subscription-Package-List
 - * **Component:** PackageList
 - * **Purpose:** Shows available subscription packages.

OEM

- **OEM List:**
 - * **Path:** /OEM-List
 - * **Component:** OEMList
 - * **Purpose:** Lists OEMs (Original Equipment Manufacturers).
- **Add/Edit OEM:**
 - * **Path:** /OEM/:type

- * **Path:** /OEM/:type/:id
- * **Component:** AddOEM
- * **Purpose:** Adds or edits OEM details.

Supplier

- **OEM Supplier List:**
 - * **Path:** /OEM-Supplier-List
 - * **Component:** OEMSupplierList
 - * **Purpose:** Lists suppliers associated with OEMs.

Settings

- **Settings List:**
 - * **Path:** /Settings-List
 - * **Path:** /Settings-List/:type
 - * **Component:** SettingsList
 - * **Purpose:** Manages application settings and configurations.

Components

- **PublicRoute:** Handles routes that do not require authentication.
- **PrivateRoute:** Manages routes that require user authentication to access.

This document provides an overview of the routes and components used in the application, specifying their purposes and how they are organized within the routing configuration.

5. State Management (if using Redux or Context API)

State Management Approach

The application uses Redux for state management, with a combination of multiple slices to manage different parts of the application's state.

Store Structure

The Redux store is structured using slices, each responsible for managing the state related to a specific feature or domain. The `combineReducers` function is used to combine all the individual slices into a single root reducer.

Reducers and Actions

Below is an overview of the reducers used in the application:

- **authentication:** Managed by `AuthenticationSlice`, handles user authentication state.
- **product:** Managed by `ProductSlice`, handles product-related data.

- **warranty:** Managed by `WarrantySlice`, handles warranty-related information.
- **batch:** Managed by `BatchSlice`, handles batch-related data.
- **vendor:** Managed by `VendorSlice`, handles vendor information.
- **settings:** Managed by `SettingsSlice`, handles application settings.
- **header:** Managed by `HeaderSlice`, handles the state for header-related components.
- **OEM:** Managed by `OEMSlice`, handles OEM-specific data.
- **supplier:** Managed by `SupplierSlice`, handles supplier-related information.
- **subscription:** Managed by `SubscriptionSlice`, handles subscription services.

Root Reducer

The root reducer is created by combining all the individual reducers:

6. API Integration

Overview

The application uses Axios for making HTTP requests to interact with various backend APIs. Axios interceptors are used to handle authentication and set up request headers. The APIs handle CRUD operations across different modules like products, settings, warranty, batch, vendor, and more.

Request Interceptors

- **Authorization:** Before each request, an interceptor retrieves the token from `localStorage` and adds it to the `Authorization` header as a Bearer token.
- **Content-Type:** The request headers are set to `multipart/form-data` to handle file uploads.

Response Handling

- **Success Responses:** Successful responses are those with status codes in the range of 200 to 304. These responses are processed, and the result is returned.
- **Error Handling:** If the request fails, the error response is caught and returned with an appropriate error message and status code.

API Methods

The following methods are used to perform API operations:

- **apiPost:** Handles HTTP POST requests to create or update resources.
- **apiPut:** Handles HTTP PUT requests for updating existing resources.

- **apiGet:** Handles HTTP GET requests to fetch data from the server.
- **apiDelete:** Handles HTTP DELETE requests to remove resources from the server.

7. Styling

Overview

The application uses a combination of SASS, Styled Components, and Material UI to handle its styling needs. This approach allows for modular, reusable, and themeable components that enhance the UI/UX of the dashboard.

SASS (Syntactically Awesome Style Sheets)

- **Purpose:** SASS is used for writing modular and maintainable CSS with features like variables, nested rules, and mixins.
- **File Structure:** The SASS files are organized by feature or component, with partials used for common styles like variables and mixins.
- **Variables:** Global variables are defined in `_variables.scss` to maintain consistency across the application.
- **Mixins:** Reusable mixins are stored in `_mixins.scss` for common patterns like flexbox, media queries, etc.

Styled Components

- **Purpose:** Styled Components are used to create encapsulated and themeable components with dynamic styling based on props.
- **Usage:** Each React component has an associated styled component that handles its styling. The styled components are often defined within the same file as the component for better cohesion.
- **Theming:** A global theme is provided using `ThemeProvider` from `styled-components`, allowing easy management of colors, fonts, and spacing throughout the application.

Material UI

- **Purpose:** Material UI (MUI) provides a set of pre-built components that follow Google's Material Design guidelines, which are used for consistent and responsive UI elements.
- **Customization:** MUI components are customized using the `sx` prop, styled components, and the MUI theme for consistent design language across the app.
- **Theming:** MUI's `ThemeProvider` is used alongside Styled Components' `ThemeProvider` to manage the application's theme. The MUI theme is extended to include custom styles and overrides as needed.

CSS Modules

- **CSS Modules:** When SASS is used in combination with CSS modules, styles are scoped locally by default, preventing conflicts with other components and improving maintainability.

This combination of SASS, Styled Components, and Material UI ensures a flexible, maintainable, and consistent styling approach across the application.

8. Testing

Overview

Testing in this project is performed by both developers and testers to ensure the correctness of API endpoints and the overall functionality of the UI. The testing approach includes manual testing using Postman for APIs and manual testing of the UI without any additional libraries.

API Testing

- **Tool Used:** Postman
- **Purpose:** To test the functionality, reliability, and performance of API endpoints.
- **Process:**
 1. **Endpoint Testing:** Each API endpoint is tested to ensure it responds correctly to various types of requests (GET, POST, PUT, DELETE).
 2. **Request Validation:** The request payload, headers, and parameters are validated to ensure proper functionality.
 3. **Response Validation:** The responses are checked for correct status codes, data structure, and content.
 4. **Error Handling:** Different scenarios, including error responses and edge cases, are tested to ensure proper handling and messages.
 5. **Environment:** Testing is performed in both development and staging environments to mimic real-world scenarios.

UI Testing

- **Tool Used:** Manual testing (no libraries)
- **Purpose:** To verify the visual and functional aspects of the user interface.
- **Process:**
 1. **Visual Inspection:** The UI is checked to ensure that all elements are rendered correctly and align with the design specifications.
 2. **Functionality Testing:** User interactions, such as button clicks, form submissions, and navigation, are tested to ensure they work as expected.
 3. **Responsiveness:** The UI is tested across different devices and screen sizes to ensure responsiveness and proper layout adjustments.

4. **Accessibility:** Basic accessibility features are checked, such as keyboard navigation and screen reader support.
5. **Cross-Browser Testing:** The application is tested in multiple browsers to ensure compatibility.

Summary

- **API Testing:** Conducted using Postman to ensure all API endpoints function correctly and handle various scenarios.
- **UI Testing:** Performed manually to verify visual and functional aspects without the use of automated testing libraries.

This approach ensures that both the backend and frontend of the application are thoroughly tested for reliability and usability.

9. Deployment

Overview

Deployment of the application is managed using cPanel. The process involves uploading the build files to the server and ensuring that the application is accessible and functioning correctly.

Deployment Process

1. Build Generation:

- The application is built using a build command (e.g., `npm run build` or `yarn build`).
- This generates a production-ready set of static files in the `build` or `dist` directory.

2. Accessing cPanel:

- Log in to cPanel using the provided credentials.

3. File Management:

- **Navigate to File Manager:** Go to the “File Manager” section in cPanel.
- **Upload Build Files:** Upload the contents of the `build` or `dist` directory to the desired directory on the server (e.g., `public_html` for the main domain).

4. Configuration:

- **Set Document Root:** Ensure the document root is set to the directory where the build files are uploaded.
- **Update Environment Variables:** If necessary, update environment variables or configuration settings to match the production environment.

5. **Testing:**

- **Access Application:** Visit the application's URL to verify that it is loading correctly.
- **Check Functionality:** Test key functionalities to ensure that everything is working as expected after deployment.

6. **Troubleshooting:**

- **Error Logs:** Check server error logs in cPanel for any issues during deployment.
- **File Permissions:** Ensure that file permissions are set correctly for the uploaded files.

Summary

- **Build Generation:** Use build tools to prepare production files.
- **cPanel:** Utilize cPanel's File Manager to upload and manage build files.
- **Testing:** Verify that the application works correctly post-deployment.

This approach ensures that the application is successfully deployed and accessible in the production environment.

14. Contact Information

- **Company:** Opine Infotech Pvt Ltd
- **Email:** support@opine.in
- **Phone:** +91 9894 112 506
- **Company Website:** Opine Infotech Pvt Ltd