Documentation: HomePage Component

The HomePage component represents the home page of the application. It displays a set of options and actions available to the user based on their role and authentication status.

Component Structure

The component consists of the following main parts:

Import Statement: Imports the necessary modules and hooks from React and other libraries.
State Management: Uses the useState hook to manage the state variables roles.
Redux State and Dispatch: Uses the useSelector hook to retrieve data from the Redux store and the useDispatch hook to dispatch actions.
Authentication and Role Checking: Checks the user's authentication status and role to determine the available options and actions.
Event Handlers: Defines event handler functions for different actions and options available on the home page.
Rendered Content: Renders the user interface elements, including links, buttons, and dynamic content such as order count and spinner.
Conditional Rendering: Conditionally renders content based on the authentication status, role, and loading/error state of the data.
Toast Notifications: Utilizes the react-hot-toast library to display toast notifications for errors and other messages.

Component Functionality

The useNavigate hook from React Router is used to access the navigation functionality.
The useDispatch and useSelector hooks are used to interact with the Redux store and retrieve necessary data.
The component checks if the user is logged in. If not, it redirects them to the login page using the useNavigate hook.
The CountOrders action is dispatched to get the total count of orders, and the result is stored in the OrderCount state variable.
The isLoading, OrderCount, and isError variables are extracted from the Redux store using the useSelector hook to manage the loading and error state of the order count.
The setRoles function is used to update the roles state variable with the user's role retrieved from the Redux store.
Two event handler functions, checkRoles and checkRoles2, are defined to handle different actions based on the user's role.
The component renders a set of UI elements, including links, buttons, and dynamic content:
Links: The Link component from React Router is used to create links to different pages within the application.
Buttons: Regular HTML buttons are used to trigger actions and navigate to different pages.
Dynamic Content: The order count is displayed dynamically based on the loading and error state of the data. A spinner is shown while the count is being fetched, an error message is displayed if there's an error, and the actual count is displayed if available.

Toast notifications are displayed using the react-hot-toast library to provide feedback to the user for success and error messages.

Usage

To use the HomePage component, you need to integrate it into your application.

Example usage:

jsx Copy code

```jsx
import { HomePage } from "./HomePage";

const App = () => {
  return (
    <div>
      {/* Other content */}
      <HomePage />
      {/* Rest of the application */}
    </div>
  );
};

export default App;
```

Documentation: LoginPage Component

The LoginPage component is responsible for rendering the login page of the application. It includes a login form where users can enter their username and password to authenticate.

Component Structure

The component consists of the following main parts:

Import Statements: Imports necessary modules, components, and actions from the Redux store.
Navigation Handling: Handles navigation events using the useNavigate hook.
Redux Integration: Retrieves user-related data and dispatches actions using the useSelector and useDispatch hooks.
Effect Hook: Executes logic based on the state changes of isError, user, isSuccess, message, and dispatch.
Form Handling: Manages form data using the useState hook and handles form submission.
Loading Spinner: Displays a loading spinner when the login request is being processed.

Rendered Content: Renders the login form, including input fields for the username and password.

Component Functionality

The navigate function from the useNavigate hook is used to handle navigation events.
The dispatch function from the useDispatch hook is used to dispatch actions to the Redux store.
The user, isLoading, isError, isSuccess, and message variables are retrieved from the Redux store using the useSelector hook.
The useEffect hook is used to handle the following scenarios:
If an error occurs during login (isError is true), an error toast notification is displayed.
If the login is successful (isSuccess is true or user is not null), the user is navigated to the home page and the page is reloaded.
The handleSubmit function is called when the login form is submitted. It dispatches the loginUserFunction action with the provided username and password.
The handleChange function updates the formData state object when the input values change.
The loading spinner is displayed when the isLoading variable is true.
The styles object defines the size of the loading spinner.
The login form includes input fields for the username and password, as well as a submit button.
The Toaster component from the react-hot-toast library is used to display toast notifications.

jsx Copy code

```jsx
import { LoginPage } from "./LoginPage";

const App = () => {
  return (
    <div>
      <LoginPage />
      {/* Rest of the application */}
    </div>
  );
};

export default App;
```

Documentation: Header Component

The Header component is responsible for rendering the header section of the application. It includes the application logo, navigation links, user information, and a mobile-friendly menu.

Component Structure

The component consists of the following main parts:

Import Statements: Imports necessary modules and components.
Component State: Manages component-level state using the useState hook.
Redux Integration: Retrieves user data and dispatches actions using the useSelector and useDispatch hooks.
Navigation State: Manages the visibility of the mobile-friendly menu.
Navigation Handling: Handles navigation events using the useNavigate hook.
Logout Handling: Handles the logout functionality.
Rendered Content: Renders the header section with the logo, navigation links, user information, and menu.

Component Functionality

The component initializes the nav state as false to manage the visibility of the mobile-friendly menu.
It uses the useDispatch hook to get access to the Redux store's dispatch function.
The component retrieves user-related data from the Redux store using the useSelector hook.
The handleNav function toggles the value of the nav state to show or hide the mobile-friendly menu.
The handleLogout function dispatches the logoutUserFunction action to log out the user.
The function also dispatches the reset action to reset the user-related state in the Redux store.
The navigate function from the useNavigate hook is used to navigate to the login page after logout.
The rendered content includes the application logo, navigation links, user information, logout button, and user icon.
The mobile-friendly menu is displayed when the user clicks on the menu icon. It includes navigation links and a logout link.
The visibility of the mobile-friendly menu is controlled based on the value of the nav state.

jsx Copy code

```jsx
import { Header } from "./Header";

const App = () => {
  return (
    <div>
      <Header />
      {/* Rest of the application */}
    </div>
  );
};

export default App;
```

Documentation: AllOrdersTable Component

The AllOrdersTable component is responsible for displaying a table of orders. It retrieves order data from the Redux store using the useSelector hook and dispatches actions to get all orders and mark orders as complete.

Component Structure

The component consists of the following main parts:

Import Statements: Imports necessary modules and components.
Component State: Manages component-level state using the useState hook.
Redux Integration: Retrieves order data and dispatches actions using the useSelector and useDispatch hooks.

Date and Order Search: Handles searching orders by date and order number.
Modal: Manages the visibility and data for the modal component.
Complete Mark: Handles marking orders as complete.
Table Columns: Defines the columns for the data table.
Custom Styles: Defines custom styles for the data table.
Loading and Error Handling: Displays a spinner or error message if loading or error states occur.
PDF Generation: Generates a PDF document with order details.
Rendered Content: Renders the table, search inputs, and buttons.

Component Functionality

The component initializes the isOpen state as false to manage the visibility of the modal.
It uses the useDispatch hook to get access to the Redux store's dispatch function.
The component retrieves order-related data from the Redux store using the useSelector hook.
The search functionality allows the user to search for orders by order number.
The date search feature filters orders based on the selected date.
The openModal function sets the isOpen state to true, opening the modal.
The closeModal function sets the isOpen state to false, closing the modal.
The handleViewClick function sets the modal data and opens the modal when the "View" button is clicked.
The completeMark function dispatches the CompleteMarkOrder action to mark an order as complete.
The generatePDF function generates a PDF document with order details using the jsPDF library.
The customStyles object defines custom styles for the data table.
The loading and error states are handled to display a spinner or error message.
The rendered content includes the search inputs, buttons, and the data table.

jsx Copy code

```jsx
import { AllOrdersTable } from "./AllOrdersTable";

const OrdersPage = () => {
  return (
    <div>
      <h1>All Orders</h1>
      <AllOrdersTable />
    </div>
  );
};

export default OrdersPage;
```

Documentation: Modal Component

The Modal component is responsible for rendering a modal dialog box that displays additional information and allows the user to update specific details of an order.

Component Structure

The component consists of the following main parts:

Import Statements: Imports necessary modules, components, and actions from the Redux store.
Conditional Rendering: If the isOpen prop is false, the component returns null and does not render anything.
Modal Classes: Defines the CSS classes for the modal dialog. (Please note that the classes are currently missing in the provided code snippet and need to be added by the user.)
Form Handling: Manages the form data using the useState hook and handles the form submission.
Redux Integration: Retrieves the necessary data from the Redux store using the useSelector hook and dispatches actions using the useDispatch hook.
Event Handlers: Defines event handler functions for updating the order and closing the modal.
Rendered Content: Renders the modal dialog box, including order details and an update form.

Component Functionality

The isOpen prop controls the visibility of the modal. If it is false, the component returns null and does not render anything.
The modalClasses variable should be updated with appropriate CSS classes for the modal dialog.
The updateFormData state object manages the form data for updating the order's shipping cost and tracking number.
The dispatch function from the useDispatch hook is used to dispatch the updateOrder and getAllOrders actions to update the order and fetch all orders after the update.
The handleUpdateOrder function is called when the update form is submitted. It dispatches the updateOrder action with the order ID (data._id) and the updated form data. It then dispatches the getAllOrders action to fetch all orders to reflect the changes in the table. Finally, it closes the modal.
The handleUpdateInput function updates the updateFormData state object when the input values change.
The modal dialog box includes order details such as the order number, the order creator's username, the order creation date, the marketplace order ID, the tracking number, and the complete marked status. It also displays a "Close" button to close the modal.
The update form allows the user to enter the shipping cost and tracking number for the order. Upon submission, the form data is sent to the handleUpdateOrder function.
The Toaster component from the react-hot-toast library is used to display toast notifications.

Usage

To use the Modal component, import it into your desired component and include it in the JSX code. Pass the necessary props to the component: isOpen, closeModal, className, and data.

Example usage:

```jsx
import { Modal } from "./Modal";

const App = () => {
  const isOpen = true;
  const closeModal = () => {
    // close modal logic
  };
  const className = "custom-modal-styles";
  const data = {
    // order data
  };

  return (
    <div>
      {/* Other content */}
      <Modal isOpen={isOpen} closeModal={closeModal} className={className}
      {/* Rest of the application */}
    </div>
  );
```

Ensure that the Redux store and necessary actions are properly set up to handle order-related functionality.

Note: Make sure to add appropriate CSS classes to the modalClasses variable to style the modal dialog box according to your application's design.

That's it! You now have a modal component that displays order details and allows the user to update specific order information.

Documentation: ProtectedRoutes Component

The ProtectedRoutes component is used to create protected routes in your application that can only be accessed by authenticated users. It redirects unauthenticated users to the login page.

Component Structure

The component consists of the following main parts:

Import Statements: Imports necessary modules and hooks from React and React Router.

Routing Logic: Uses the useNavigate hook from React Router to manage navigation. Retrieves the user information from the Redux store using the useSelector hook.
Effect Hook: Uses the useEffect hook to check if the user is authenticated. If the user is not authenticated (user is null or undefined), it redirects the user to the login page using the navigate function.
Rendered Content: Renders the protected component specified by the Component prop.
Component Functionality
The navigate function from the useNavigate hook is used to navigate to different routes.
The user object is retrieved from the Redux store using the useSelector hook. It represents the currently logged-in user.
The useEffect hook is used to perform a check when the component mounts. If the user object is null or undefined, indicating that the user is not authenticated, the component redirects the user to the login page by calling navigate("/login").
The Component prop is destructured from props. It represents the protected component that should be rendered if the user is authenticated.
The protected component specified by the Component prop is rendered.

Usage

To use the ProtectedRoutes component, wrap your protected routes with it. Pass the protected component as a prop named Component.

Example usage:

jsx Copy code

```jsx
import { ProtectedRoutes } from "./ProtectedRoutes";
import Dashboard from "./Dashboard";

const App = () => {
  return (
    <div>
      {/* Other content */}
      <ProtectedRoutes Component={Dashboard} />
      {/* Rest of the application */}
    </div>
  );
};

export default App;
```

Ensure that you have properly set up the Redux store and authentication-related functionality to determine the user's authentication status.

That's it! You now have a protected route that redirects unauthenticated users to the login page.

Documentation: AddNewOrders Component

The AddNewOrders component is responsible for creating new orders in the application. It provides a form for users to input order details and handles the submission of the form data.

Component Structure

The component consists of the following main parts:

Import Statements: Imports necessary modules, components, and hooks from React and Redux.
State Management: Uses the useState hook to manage state variables for various form fields and other UI-related states.
Dispatch and Selector: Uses the useDispatch and useSelector hooks from React Redux to dispatch actions and retrieve data from the Redux store.
Effect Hook: Uses the useEffect hook to handle side effects, such as displaying toasts for success or error messages and resetting form data and Redux state.
Form Input Handlers: Defines functions to handle form input changes, add new line forms, and toggle tracking number input fields.
Form Data Management: Manages the form data by updating the state whenever there is a change in any of the form fields.
Rendered Content: Renders the form inputs, including dynamically generated line forms and child tracking forms, and handles form submission.
Child Components: Defines the NewLineForm and ChildTrackingForm components that are used within the AddNewOrders component.

Component Functionality

The useState hook is used to manage various state variables, such as shape, newLines, showTrackingBox, and childTracking, which control the visibility and values of different form inputs.
The useDispatch and useSelector hooks are used to interact with the Redux store. The component retrieves relevant data and dispatches actions to create orders.
The useEffect hook is used to handle side effects. It listens for changes in Redux state variables like isLoading, isError, isSuccess, and createdOrder. If there is an error, it displays an error toast message. If the order is successfully created, it displays a success toast message. The Redux state is then reset.
The component defines various event handlers to handle form input changes, add new line forms, and toggle the visibility of tracking number input fields.
The form data is managed by updating the state whenever there is a change in any of the form fields. The form data is used to dispatch the CreateOrders action.
The component renders the form inputs, including order details like order number, market place order ID, shape, thickness, length, width, diameter, and quantity. It also dynamically generates line forms based on the number of newLines and renders child tracking forms.

The NewLineForm component renders the form inputs for a new line, including thickness, shape, length, width, diameter, and quantity.
The ChildTrackingForm component is a placeholder for child tracking forms.
The component renders a spinner while the form is submitting (isLoading is true).
Usage
To use the AddNewOrders component, you need to integrate it into your application, including setting up the Redux store and related actions and reducers.

Example usage:

jsx Copy code

```jsx
import { AddNewOrders } from "./AddNewOrders";

const OrdersPage = () => {
  return (
    <div>
      {/* Other content */}
      <AddNewOrders />
      {/* Rest of the page */}
    </div>
  );
};

export default OrdersPage;
```

Ensure that you have properly set up the Redux store and actions/reducers related to creating orders.

That's it! You now have a form for users to add new orders, including multiple line items and tracking numbers.

Documentation: AdminPage Component

The AdminPage component represents a page where administrators can perform administrative tasks, such as creating new users. It provides a form for creating new users and includes options to view all users.

Component Structure

The component consists of the following main parts:

Import Statement: Imports the necessary modules and hooks from React.

State Management: Uses the useState hook to manage the state variable viewForm, which determines whether the user creation form is visible or not.
Event Handlers: Defines the handleViewForm function to toggle the visibility of the user creation form.
Rendered Content: Renders the user interface elements, including buttons to create a new user and view all users, and a form for creating a new user.
Form Inputs: Renders form inputs for user creation, including fields for username, email, password, and role selection.
Button Actions: Defines the actions to be performed when buttons are clicked, such as toggling the visibility of the user creation form and submitting the form data (not implemented in the provided code).

Component Functionality

The useState hook is used to manage the viewForm state variable. It determines whether the user creation form is visible or not.
The handleViewForm function is responsible for toggling the visibility of the user creation form when the "Create New User" button is clicked. It updates the viewForm state variable and logs its current value to the console.
The component renders a container section with a maximum width, containing two buttons: "Create New User" and "View All Users".
When the "Create New User" button is clicked, the user creation form is displayed below the buttons. The form includes input fields for username, email, password, and role selection.
The role selection field is a dropdown menu with options for different user roles, including "User," "Admin," and "Super User."
The form does not currently have a submit button or implementation for form submission. This needs to be implemented separately based on the desired functionality of creating a new user.
When the form is submitted (not implemented in the provided code), it should trigger the appropriate action or function to create a new user based on the entered data.
Usage

To use the AdminPage component, you need to integrate it into your application.**Exampleusage:**

```
export const AdminPage = () => {
  const [viewForm, setViewForm] = useState(false);
  const handleViewForm = () => {
    setViewForm(!viewForm);
    console.log(viewForm);
  };
  return (
    <>
      <section className="container  mx-auto h-screen max-w-6xl p-4  z-10 m-8">
        <div className="flex gap-10 justify-evenly">
          <button
            onClick={handleViewForm}
            className=" hover:bg-orange-600 w-72 h-24  rounded-full flex items-c
          >
            <p className="text-2xl">Create New User</p>
          </button>

          <button className=" hover:bg-orange-600 w-72 h-24  rounded-full flex i
            <p className="text-2xl">View All Users</p>
          </button>
```

Documentation: UserProfile Component

The UserProfile component represents the user's dashboard or profile page. It displays the user's information, such as UserID, email, username, and roles. Additionally, it provides the functionality to update the user's password.

Component Structure

The component consists of the following main parts:

Import Statement: Imports the necessary modules and hooks from React, React Redux, React Router DOM, and toast notification library.
State Management: Uses the useState hook to manage the password state variable.
Redux State and Dispatch: Uses the useSelector hook to retrieve the user data from the Redux store and the useDispatch hook to dispatch the updatePassword action.
Navigation: Uses the useNavigate hook from React Router DOM to access the navigation functionality.
Event Handlers: Defines the updateUserPassword event handler function to handle the password update action.
Rendered Content: Renders the user interface elements, including the user information, password update form, and toast notifications.
Toast Notifications: Utilizes the react-hot-toast library to display a success message after a successful password update.

Component Functionality

The useNavigate hook is used to access the navigation functionality, and the useDispatch and useSelector hooks are used to interact with the Redux store and retrieve the necessary data.

The password state variable is used to store the user's new password input.
The updateUserPassword event handler function is called when the user clicks the "Update Password" button. It dispatches the updatePassword action with the new password and user ID, and displays a success toast notification.
The component renders the user's dashboard or profile information, including the UserID, email, username, and roles.
It also renders a form for updating the password, consisting of an input field for the new password and a button to trigger the password update.
Toast notifications are displayed using the react-hot-toast library to provide feedback to the user for a successful password update.
Usage
To use the UserProfile component, you need to integrate it into your application.

Example usage:

jsx Copy code

```jsx
import { UserProfile } from "./UserProfile";

const App = () => {
  return (
    <div>
      {/* Other content */}
      <UserProfile />
      {/* Rest of the application */}
    </div>
  );
};

export default App;
```