**Here is a list of React hooks :**

useState: Essential for managing component state.

useEffect: Critical for side effects and lifecycle management.

useContext: Important for sharing data across components.

useReducer: Useful for complex state logic.

useRef: Handy for accessing and interacting with DOM elements.

useMemo: Optimizes expensive computations.

useCallback: Optimizes callback functions. (Prevents re-rendering of child component)

useLayoutEffect: Similar to useEffect but runs synchronously. (it fires before the useEffect and before the browser visually updates the page)

useHistory: Access and manipulate the browser's history.

useParams: Retrieve parameters from React Router.

useLocation: Access the current URL location.

useRouteMatch: Match the current URL against a specific route.

`useSelector` is a function provided by the React Redux library, which is commonly used in React applications to access and retrieve data from the Redux store. Redux is a state management library for JavaScript applications, and React Redux is a library that provides integration between Redux and React.

Here's how you can use `useSelector` in a React component:

1. \*\*Import necessary dependencies:\*\*

First, you need to import the `useSelector` function from the `react-redux` library. You'll also need to import any actions or selectors you want to use.

```javascript

import { useSelector } from 'react-redux';

```

2. \*\*Create a component:\*\*

Define a functional component in your React application. This component will use `useSelector` to access data from the Redux store.

```javascript

function MyComponent() {

// Use useSelector to access data from the Redux store

const myData = useSelector((state) => state.myReducer.myData);

return (

<div>

<h1>My Data</h1>

<p>{myData}</p>

</div>

);

}

```

3. \*\*Configure the Redux store:\*\*

Before you can use `useSelector`, you need to configure the Redux store in your application. This involves creating a store, defining reducers, and setting up the store provider at the top level of your application. Here's a simplified example:

```javascript

// Import necessary Redux dependencies

import { createStore, combineReducers } from 'redux';

import { Provider } from 'react-redux';

// Define reducers

const initialState = { myData: '' };

function myReducer(state = initialState, action) {

switch (action.type) {

case 'SET\_MY\_DATA':

return { ...state, myData: action.payload };

default:

return state;

}

}

// Combine reducers

const rootReducer = combineReducers({

myReducer,

});

// Create the Redux store

const store = createStore(rootReducer);

// Wrap your app in the Redux Provider component

ReactDOM.render(

<Provider store={store}>

<App />

</Provider>,

document.getElementById('root')

);

```

4. \*\*Dispatch actions to update state:\*\*

To update the state in your Redux store, you'll typically dispatch actions. In the `MyComponent` example, if you want to update `myData`, you would dispatch an action like this:

```javascript

// Dispatch an action to update myData

dispatch({ type: 'SET\_MY\_DATA', payload: 'New Data' });

```

The Redux reducer (`myReducer` in this example) will handle this action and update the state accordingly. `useSelector` will automatically re-render your component when the Redux store's state changes, so your component will reflect the updated data.

That's the basic usage of `useSelector` in a React Redux application. It allows you to connect your React components to the Redux store, access state, and re-render components when the state changes.