MODULE: 13 React – Applying Redux

* **What is Redux?**

Ans : Redux : state management library

Redux is an open-source JavaScript library used to manage application state. React uses Redux for building the user interface. It was first introduced by Dan Abramov and Andrew Clark in 2015.

React Redux is the official React binding for Redux. It allows React components to read data from a Redux Store, and dispatch Actions to the Store to update data.

Redux was inspired by Flux. Redux studied the Flux architecture and omitted unnecessary complexity.

* Redux does not have Dispatcher concept.
* Redux has an only Store whereas Flux has many Stores.
* The Action objects will be received and handled directly by Store

**Why use React Redux?**

React Redux is the official UI bindings for react Application. It is kept up-to-date with any API changes to ensure that your React components behave as expected.

**STORE**: A Store is a place where the entire state of your application lists. It manages the status of the application and has a dispatch(action) function. It is like a brain responsible for all moving parts in Redux.

**ACTION**: Action is sent or dispatched from the view which are payloads that can be read by Reducers. It is a pure object created to store the information of the user's event. It includes information such as type of action, time of occurrence, location of occurrence, its coordinates, and which state it aims to change.

**REDUCER**: Reducer read the payloads from the actions and then updates the store via the state accordingly. It is a pure function to return a new state from the initial state.

* **What is Redux Thunk used for?**

Ans : Redux Thunk is a middleware for Redux that allows you to write action creators that return a function instead of an action. This function can then be used to perform asynchronous operations, like fetching data from an API or performing complex logic before dispatching an action to the store.

Key Uses of Redux Thunk:

* Handling Asynchronous Actions:

Thunks can dispatch actions before and after an asynchronous operation.

For example, you can dispatch an action to indicate a loading state before fetching data from an API and then dispatch another action with the data once the fetch is complete.

* Conditional Dispatching:

Thunks can contain logic to conditionally dispatch actions. This is useful when actions depend on the current state or other factors.

For example, you might check the current state to avoid making unnecessary API calls.

* Complex Action Creators:

Thunks allow for more complex action creators that can include multiple steps or side effects.

* **What is Pure Component? When to use Pure Component over Component?**

**Ans :** PureComponent is similar to Component but it skips re-renders for same props and state. Class components are still supported by React, but we don’t recommend using them in new code.

Pure components are specifically designed for performance optimization by minimizing unnecessary renders through automatic prop comparison. Regular components, both functional and class-based, might re-render even if props or state don't change, leading to potential performance issues.

* **Optimizing Performance:**

Frequent Re-renders: If your component re-renders frequently and the props and state rarely change, using PureComponent can prevent unnecessary re-renders and improve performance.

Large Component Trees: In a complex application with deep component trees, minimizing unnecessary re-renders in lower components can significantly improve overall performance.

* **Simple and Immutable Data Structures:**

Primitive Data Types: If your component deals with simple props and state involving primitive data types (like strings, numbers, booleans), PureComponent can efficiently manage shallow comparisons.

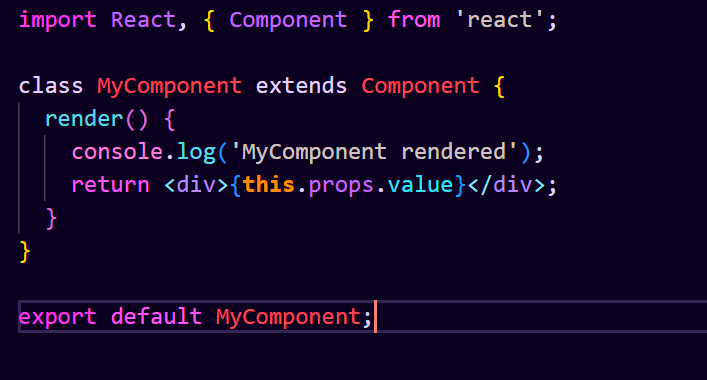
Immutable Data: If your application uses immutable data structures, PureComponent can be very effective. Since immutable data ensures that a new reference is created when data changes, the shallow comparison in PureComponent will reliably detect changes.

* **Consistency and Predictability:**

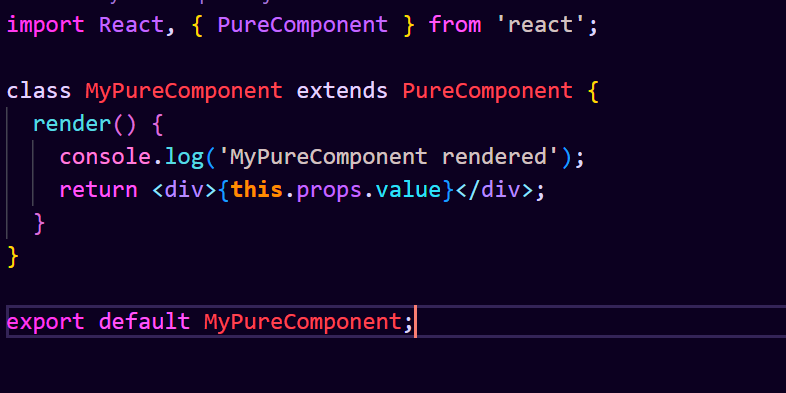
Pure Functions: If your component behaves like a pure function, where the same inputs (props and state) always produce the same output (rendered result), PureComponent can help ensure that the component only re-renders when necessary.

Example Usage of PureComponent :

**Regular Component :**

****

**Pure Component :**

****

* **What is the second argument that can optionally be passed tosetState and what is its purpose?**

**Ans :** In the context of Redux, there is no direct equivalent to React's setState method, since Redux uses a different paradigm for managing state. Redux relies on a central store, actions, and reducers to handle state changes.

Using Redux Thunk

With redux-thunk, you can dispatch a function (thunk) instead of an action. This function can perform asynchronous operations and dispatch other actions based on the results of these operations.

**Purpose of the Callback**

The purpose of the callback in this context is to perform additional actions once the Redux state has been updated. This is similar to the setState callback in React, ensuring that the code inside the callback runs after the state update.

* **Asynchronous** **actions**: The callback pattern is useful for handling asynchronous operations where you need to perform actions based on the updated state.
* **Middleware**: Middleware like redux-thunk enables you to handle side effects and perform actions that depend on state updates.
* **Redux**-**saga**: Another approach is using redux-saga, which provides more powerful handling of side effects through generator functions.