*React - Applying Redux*

**What is Redux?**

**Ans** = An open-source JavaScript package called Redux is used to control the state of applications. Redux is used by React to construct the user interface. Dan Abramov and Andrew Clark originally presented it in 2015.   
The official React binding for Redux is called React Redux. It enables React components to send actions to a Redux store in order to update data and read data from the store.  
Redux took inspiration from Flux. Redux examined the Flux architecture and removed pointless intricacy.

• The Dispatcher idea is absent from Redux.   
• Flux has numerous stores, while Redux only has one.   
• Store will receive and handle the Action objects directly.

Use of React Redux:

The official UI bindings for React Applications are called React Redux. It is updated with every modification to the API to guarantee that your React components function as intended.  
STORE: Your application's complete state is listed in a store. It has an action (dispatch) function and controls the application's status. It functions similarly to Redux's brain, controlling every moving component.

REDUCER: After reading the payloads from the actions, Reducer modifies the store's state via the store. The function's sole purpose is to return the initial state as a new one.

ACTION: The view sends or dispatches actions that are payloads that Reducers are able to read. It is a simple object made specifically to hold user event data. It contains details on the kind of activity, when it happens, where it happens, its coordinates, and the state it intends to alter.

**What is Redux Thunk used for?**

**Ans** = Redux Thunk is a middleware that enables you to design action creators that return a function rather than returning an action. After that, this method can be used to perform asynchronous tasks like gathering information from an API or carrying out intricate calculations before sending an action to the store.   
Important Redux Thunk Uses:   
Asynchronous operations can have actions dispatched both before and after by thunks.   
For instance, before retrieving data from an API, you can send one action to signal that it is loading, and after the retrieve is finished, you can dispatch another action with the data.

Redux State and Dispatch Access:  
You can use the getState and dispatch methods to get the Redux store's current state and dispatch actions from within your asynchronous operations by passing them as arguments to the thunk function.

Conditional Dispatching: Logic for conditionally dispatching activities can be found in chunks. When activities are dependent on the situation as it is or other variables, this is helpful.  
Checking the current status, for instance, can help you avoid making pointless API queries.  
  
sophisticated Action Creators: Action creators with numerous phases or side effects can be made more sophisticated using thunks.

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

**Ans** = In React, a Pure Component is a kind of component that determines whether to re-render by doing a cursory comparison between its props and state. Because it avoids needless re-renders, this comparison is automatic and contributes to performance improvement.  
  
Important Pure Component Features:  
  
Shallow Comparison: To compare the previous and next values of props and state, a Pure Component utilizes shallowEqual. It stops the component from re-rendering if nothing changes.  
shallowEqual only compares the attributes at the highest level of the objects. A thorough comparison of nested objects is not carried out by it.  
  
Automatic shouldComponentUpdate: A Pure Component, in contrast to a conventional component, has an inbuilt shouldComponentUpdate function that handles the superficial comparison.

When you want to maximize performance, especially in large applications where needless re-renders can cause the app to lag, use Pure Component.  
Ideal for components with expensive rendering processes or those that render frequently.  
States and Props Are Easy:  
  
Perfect for simple, shallow props and states without a lot of highly nested arrays or objects. This guarantees the efficacy of the superficial analogy.  
Consistent Data Organisations:  
  
When you are certain that the props and state won't change and you know how to handle immutable data, use Pure Component. By doing this, unintentional alterations that might avoid the superficial comparison are prevented.

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

**Ans** = A callback function is the second optional argument that can be supplied to setState in a React class component. After the setState action is finished and the component has re-rendered with the modified state, this callback method is triggered.  
  
The Second Argument's Goal: Post-Update Logic  
  
You can carry out any logic that requires the component to have re-rendered with the updated state by using the callback function. This can be especially helpful for tasks that need to be completed after the DOM has been modified and the state update has been implemented.  
Assuring Proper Sequential Action:

You may need to ensure that certain functions execute only after the state has been updated because setState is asynchronous. The callback ensures that its code will run following the state update.

