Redux

It is a state management library for javascript applications. We can use it with React, Angular,Vue.js and vanilla JS. This is because it is a state management library, therefore it doesn’t care about which library we are using to build the UI.

The problem without a State Management library

Well, if you have ever built an application with a complex UI, you have probably come across a situation where you need to keep different parts of the UI synced. In a complex application, let’s say the user changes data in one part of the application then other parts of the application should be immediately updated to reflect the changes. In more complex scenarios, the data also gets updated as a result of network requests or background tasks. Therefore, we need to write a lot of code to keep it synced.

Why Use Redux - State Management Library?

With the help of Redux, instead of scattering the application state in various components of the application, we will store all the application state inside a central repository, that is a single JS object called the Redux Store. You can assume it as a kind of database for the front end of an application. So with this architecture, the different components of the app have to no longer maintain their own state. Instead, they get what they need from the Redux store. Redux as an architecture also makes it easy to understand how the data changes in our application, if something goes wrong we can see exactly how, why, when, and where the data changed. Remember, Redux performs a similar function as Context API and hence helps in reducing the prop drilling. Hence, Redux is a custom solution for state management.

In a Nutshell - The Need to Use Redux

Redux centralizes the application’s state

It also makes the data flow transparent and predictable

In addition to this, It also makes Testing very easy.

It brings out Consistency throughout the application

Terms that we use in Redux

Actions & Reducers

Actions: An action is a plain object that describes the intention to cause change. Let’s suppose we are creating an iNotebook application, which contains a lot of notes. Now, suppose, we want to change the correspondence of the notes with respect to their Users. To do so, we will bring an action to change the Notes array by adding another note.

Reducers: A reducer is a function that determines changes to an application state. It returns to the new State and tells the store how to do it. For example, we would like to add a new note in the application then, Reducer will determine which variable/array/State will be changed by the addition of the note. In most simple words, It uses the action it receives to determine the change.

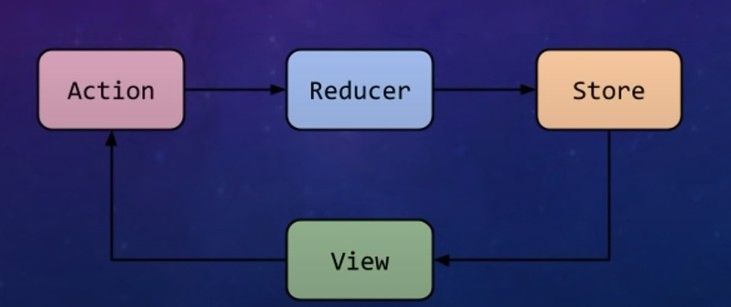


Figure 1.1: Actions and Reducers

The analogy of Actions and Reducers

