Redux (Javascript)

**Basic Notes::**

* Redux is a popular JavaScript library for managing the state of your application.
* State is an Global Objects of an Application that contain information that can be used for the various purpose in the Application. (State is that shared data of the Application which we can used in more then one component).
  + For example :: Loading is an Boolean value which can be stored in a store as a state so that it can be commonly used (set or unset) by the components at the time of loading the page or data. (toggle the display of the loading indicator)
* There are no rules as to which data should be kept in Redux or store, so, it’s all up to you. However, serializable data should be preferred to kept in store as state.

**Patterns followed by Redux::**

* one pattern that Redux follows is called “Single Source Of Truth”, which means that we have only one place (called Store) where we store the state for the whole application.  
  In other words, one app — one store — one state.
  + However, Components ( such as in react) can also manage their own internal state if it is required to do so.
* Another pattern that Redux follows is called “immutability”, which means we don’t change the state object and its properties directly. Instead, we make a new object, recalculate the new application state and update it with our newly created object. We want to leave our old state object intact.

**Building Blocks of Redux ::**

There are three main Building block or we can say three pillars of Redux :

1. **Store**

* Storehold the state of the application.
* Store contains a few extra things other than your application’s state as well (like functions and other objects).
* The state in Redux is in the form of a JavaScript Object and is often referred to as the “state tree”. You can put whatever values you want to store in it and you can nest them as much as you need.
* it is possible to create multiple stores, but this is against the pattern that Redux follows.
* We have to subscribe the Store Events so that when ever the store get updated then its respective UI get also Updated.
* Creating a store inside the Application is very simple, we just have to create it through **“createStore()”** method of redux library and passing the reducer(root reducer in case of multiple reducers exist) as its first argument and composer as its second argument (Optional).
* To provide that Store Global Access to our application, we import it into our main.js or index.js file (In react we use provider component of react-redux to provide support to entire application). For Details :: <https://github.com/JainNaveen94/shopnix-mobile-store/blob/master/src/index.js>

1. **Actions**
   * Actions are plain JavaScript objects or function (usually called action creators) that describe **WHAT** happened, without specifying how it will happen (what state change but not how state changes ).
   * One important thing to remember is that Redux requires our action objects to contain a type field. This field is used to describe what kind of action we are dispatching and it should usually be a constant that you export from a file.  
     All other fields in the action object are optional and are up to you.
     + For example :: const initTask = (data) => {

type: <action\_type>/Action.INIT\_TASK,

initilizeData: data

}

Here, Action is a file having constants of the action types available in the application.

* + In Single Action File, there may be one or more action of different-2 types similarly there are multiple action files in our application.
  + These action files are basically imported in our respective component file from where we want to dispatches this actions. { please note action can also be dispatched from other action inside action file }. For More Detail :: <https://github.com/JainNaveen94/shopnix-mobile-store/blob/master/src/store/actions/cart.js>.

1. **Reducers**
   * Reducers are **pure** functions that define **HOW** the app state get changes based on action type in a store.
   * Whenever we dispatch an action to our store (calling appropriate type of action from component or other action), the action gets passed to its respective reducer.  
     The reducer function takes two arguments: the previous app state or initial state and the action being dispatched and returns the new app state.
     + initialState : { // state }

Const taskReducer = (state = initialState, action) => {

Switch(action.type) {

Case Action.INIT\_TASK:

return {

…state,

// changes which you want to perform

}

Default:

return state;

}

};

* + Here, above we can define switch case for a single action, so If multiple actions are there then we just need to increase the cases as per our requirnment. For more detail please refer the <https://github.com/JainNaveen94/shopnix-mobile-store/blob/master/src/store/reducers/cart.js>
  + There may be chances that there are more then one reducer exist inside our application so in that case we need to combine them into a single reducer called as **root reducer.** The reducers are combine through “combineReducer()” method of redux liberary in which we pass different-2 reducers in the form of key: value pair under object.
    - For example :: const rootReducer = composeReducer({task: taskReducer})

**Important Note Points ::**

* In Normal Java Script or vanilla Javascript we basically import the store into appropriate components (who need store access) to used its states.
* But in react we have to connect a component (who need store access) to the store by using following steps ::
  1. Import the connect method from “**react-redux**”.
  2. Define the following Two Objects:
     + mapStateToProps – return the detail of respective states which is needed for that component. (it has an access to state object by default due to global store access through providers of react redux).
     + mapDispatcherToProps -return the depatched actions described in action file (imported in component) to its respective method call to dispatch the action to reducers.
  3. After Defining the above two Objects, export those two object along with component under connect function of “react-redux”.
     + For Example :: please refer => <https://github.com/JainNaveen94/shopnix-mobile-store/blob/master/src/containers/Cart/Cart.js>

**Data Flow::**

* Whenever event is triggered by the user to update the state then in that case its respective action is dispatched to the reducer.
  + Here Action is dispatched in Normal Javascript through “dispatched()” method of store object which is imported in that component.
  + But in React, We can define all the dispatched action under “mapDispatcherToProps” Object so that respective action can be called by react itself.
* Reducer will create the new State Object and replace it with old one inside the Store based on the action Type.
* The Listeners which subscribe the store will listen the state update through their subscription and then update the respective UI Component.
  + Here in Normal Javascript, store object have a “subscribe()” function in which we can call the other function which update the UI by getting the updated state through “getState()” method of store.
  + But in react, we can define or subscribe all the required state under the “mapStateToProps” Object so that we can simply access the updated state through this.props.<defined state>.

## 