Creating Simple Counter App with REDUX

We will be creating a simple counter app with help of react-redux and redux.

SO First thing we need to do is install required dependencies in our project.

So we will go to out terminal and,

```
npm install react-redux
npm install redux
```

Now in our src folder, we will create a file counterAction.js

Inside this action file is where we would have all actions and functions.
Action are payload of information that will send data from our application to our store.
They are the only source of information for the store, we can send the, using store.dispatch()

So Actions are the simple functions that returns an object that describes how its going to manipulate the state. The object has type attribute with value of action to be taken.

Inside counterAction.js we will have following code

```
export function increaseCount() {
return { type: 'ADD'}
}

export function decreaseCount() {
return { type: 'SUBSTRACT'}
}
```

Now that we have set up our actions, we can write our reducer function. So we will create new file name it counterReducer.js

A reducer function takes in two argument, the initial state and the action parameter.
The initial state of the reducer contains a single property, count that is initialized at 0.

The way we will be set up our reducer function is with conditions. We are going to implement a switch statement on action parameter to determine how to update the state. We have used switch statement because a switch statement have a default value if no conditions are match, a default value will always be returned.

If the action type is ADD, the reducer creates a new state object via object spread that copies all properties of the existing state object into new object and then changes the count property by incrementing it by 1. If the action type is SUBSTRACT , the reducer creates a state object via object spread that copies all the properties of the existing state object and then changes the count property by decrementing it by 1.
If the action type is neither increment or decrement, the reducer simply return the existing state.

It is important to notice, to maintain the inmutablility of our state, we are using a spread operator. Here we are using **...state**
... is an spread operator syntax in javascript. It is used to create object or array into a new object or array. So here ...state is used to create a new object that copies all properties of the existing state object.

```
export default function counterReducer(state = {count: 0}, action) {
switch(action.type) {
case 'ADD':
return {
...state,
count: state.count + 1
}
case 'SUBSTRACT':
return {
...state,
count: state.count - 1
}
default:
return state
}
}
```

Now we will be creating our ReduxStore

Now we have all we need to create a store for our <App /> component. For this we will go to index.js and in this we will first import createStore from redux,
Because we are using latest version of redux, version 4.2 . CreateStore in redux is depreciated by redux. It is to spread awareness about redux features of toolkit.
So first we will install redux toolkit in our project.

```
npm install @reduxjs/toolkit
```

Now we will import { configureStore } from @reduxjs/toolkit

So now we will set a variable equal to configureStore method. A reducer is a required argument to our configureStore.

```
import { configureStore } from '@reduxjs/toolkit'

import counterReducer from './counterReducer';



const store = configureStore({reducer : counterReducer})
```

Now we will import provider from react-redux.
Provider is used to send down props to other components that are wrapped inside connect function.
We will pass down the store variable as props to our app component.

```
<Provider store={store} >
<App />
</Provider>,
```

Now we will build react component that will render our buttons and other required UI for our webpage. So we will create a new file, name it as CounterInput.js

Now we will simply create a text and button and define its css styling.

```
<div className="container">
<div className="number">
<p> Count : 0 </p>
</div>
<div className="allbuttons">
<button name='Plus' class="button" >
Plus
</button>
<button name='Minus' class="button" >
Minus
</button>

</div>
</div>
</div>
```

Now we will use connect componet from react-redux. This will grand us access to the store and other components in our app.

So

```
import { connect } from 'react-redux'
```

Now we will wrap our app with the connect component and pass null as one of its value. This is because we have not yet accessing the store.

```
export default connect(null)(counterInput);
```

We will import actions that we created earlier.

```
import { increaseCount, decreaseCount } from './counterAction'
```

we will include them in our connect statement at the bottm of the file.

```
export default connect(null, {increaseCount, decreaseCount})(counterInput);
```

Now we will add onClick event listener to our button that points to handleOnClick() function.

```
<button name='Plus' class="button" onClick={e => this.handleOnClick(e)} >
Plus
</button>
<button name='Minus' class="button" onClick={e => this.handleOnClick(e)} >
Minus
</button>
```

Now we will define our handleOnClick()

So we want our handleOnClick() function to trigger an action based on the event's target name. In this case, the target are the buttons and the names are Plus or Minus.

If the click target's name is "Plus" then the increaseCount() action will be dispatch, if not then decrementCount() action will be dispatch.

```
handleOnClick = (e) => {
e.preventDefault()
if(e.target.name === 'Plus'){
this.props.increaseCount()
}
else{
this.props.decreaseCount()
}
}
```

Now our Final Step will be to connect CounterInput component to the store.

Now we are ready to show user count, to do this we need to access to redux store.

We will do this by creating a function which will take the state object from redux store and returns an object containing a count property that we can access the value with state.count
The main purpose of this function is to return our state. This means the count property from the redux store will be available in <App /> component as a prop called count.
We can name it anything , for better understanding we will name it as mapStateToProps.

```
const mapStateToProps = (state) => {
return {
count: state.count
}
}
```

Now we will pass it as argument to connect().

```
export default connect(mapStateToProps, {increaseCount, decreaseCount})(counterInput);
```

Now we can have access to current count from the store. SO we will be calling it to display as needed.

```
<p> Count : {this.props.count}</p>
```

Now we will remove the default template from App.js and import our CounterInput.js to it.