Redux is a state management tool

In general React the state block data is passed to only child components

it happen only if the application is small , in case of large applications

sharing data from parent component becomes some messy and the state block data cannot be shared by the other components which are not the

part of the parent components i.e , sibling components cannot be shared

This Redux stores all the data of state ,which is live in a store

and can be shared anything which is in state from this store to

other components

The way Redux works is simple. There is a central store that holds the entire state of the application. Each component can access the stored state without having to send down props from one component to another.

There are three building parts: actions, store, and reducers.

1. Actions are the events, this is the only way we can send the data from application to the redux store, the data can be from ui part ,api calls, form submissions

Actions are sent through store.dispatch( ) method

Actions are plain java script object which contains type property and payload

Type property is what type of action to be carried out

Payload which has the information to be worked on by the action

{

type: "LOGIN",

payload: {

username: "foo",

password: "bar"

}

}

Actions are created via action creator:

Here is the example of action creator:

const setLoginStatus = (name, password) => {

return {

type: "LOGIN",

payload: {

username: "foo",

password: "bar"

}

}

}

1. Reducers in Redux

It is a pure function , it takes a current state ,perform action over it and returns new state, these states are stored as objects and they specify how the state of a applications changes in response to an action sent to the store.

It is based on the reduce function in JavaScript, where a single value is calculated from multiple values after a callback function has been carried out.

Here is an example of how reducers work in Redux:

const LoginComponent = (state = initialState, action) => {

switch (action.type) {

// This reducer handles any action with type "LOGIN"

case "LOGIN":

return state.map(user => {

if (user.username !== action.username) {

return user;

}

if (user.password == action.password) {

return {

...user,

login\_status: "LOGGED IN"

}

}

});

*Note:Reducers take the previous state of the app and return a new state based on the action passed to it.*

As pure functions, they do not change the data in the object passed to them or perform any side effect in the application. Given the same object, they should always produce the same result.

3.Store

The store holds the application state. There is only one store in any Redux application. You can access the state stored, update the state, and register or unregister listeners via helper methods.

Let’s create a store for our login app:

const store = createStore(LoginComponent);

const {createStore} =Redux;

const initState={

todos:[],

posts:[]

}

const myReducer=(state=initState,action)=>{

if(action.type=='AddTodo'){

return{

...state,

todos:[...state.todos,action.payload.todo]

}

}

if(action.type=='AddPost'){

return{

...state,

posts:[...state.posts,action.payload.post]

}

}

}

const store=createStore(myReducer);

store.subscribe(()=>{

console.log('state updated');

console.log(store.getState())

})

store.dispatch({type:'AddTodo',payload:{todo:"Brink milk"}});

store.dispatch({type:'AddTodo',payload:{todo:"Brink eggs"}});

store.dispatch({type:'AddPost',payload:{post:"we will go for hunt"}});

Thunk:

It is a middleware in between action and reducer .it’s lika a middleman anything before gng to reducer it will move to thunk

A thunk is a function that returns another function. For example:

function say() {  
 return function something() {  
 //code here  
 };  
}

To access the store value we use connect

Import connect from ‘react-redux’;

mapStateToProps=(state)=>{

return{

data:state

}

Wrap the export value with connect

Export default connect (mapStateToProps)(App);

1. i) Create a store, which is store acts as central data and import thunk for api calls i.e, database
2. It takes reducer and any middleware like thunk

i.e, createStore

import {createStore, applyMiddleware} from 'redux';

import asyncreducer from './reducers/rootReducer';

import thunk from 'redux-thunk';

const store=createStore(asyncreducer,applyMiddleware(thunk));

export default store;

2.Create reducers

I)Which has state and actions as parameters and export that reducer.

ii)Next in store ,add this reducer,

3.Create Actions:

In this write and export the actions with ‘type’ and if any data present pass this to the action i.e, function in it.

export const fetch\_post = () => {

  return {

    type: "FETCH\_USER"

  };

};

export const receive\_post = post => {

  return {

    type: "FETCHED\_USER",

    data: post

  };

};

export const receive\_error = () => {

  return {

    type: "RECEIVE\_ERROR"

  };

};

export const thunk\_action\_creator=username=>{

    const user=username.replace(/\s/g,"");

    store.dispatch(fetch\_post());

    return function(dispatch,getState){

        return    fetch(`https://api.github.com/users/${user}`).then(data=>data.json()).then(data=>{

            if(data.message==="Not found"){

                throw new Error("No succh user found!")

                        }

            else{

            dispatch(receive\_post(data));

            }

        }).catch(err=>dispatch(receive\_error()))

    }

}

3.Next import Provider ,which sends the data from store to other components

i)Wrap the App component with Provider in index.js

import {Provider} from'react-redux';

ReactDOM.render(<Provider  *store*={store}><App /></Provider>, document.getElementById('root'));

4.Go to other components where you want to access the store data,in that component:

import connect,which gives access to dispatch method which takes the actions as parameter

import {connect} from 'react-redux';

*this*.props.dispatch(thunk\_action\_creator(username));

Next in the export write the connect and pass required parameters and wrap the comp with brackets and define the parameter

const mapStateToProps = state => {

  return {

    data: state

  };

};

export default connect(mapStateToProps,mapDispatchToProps)(App);