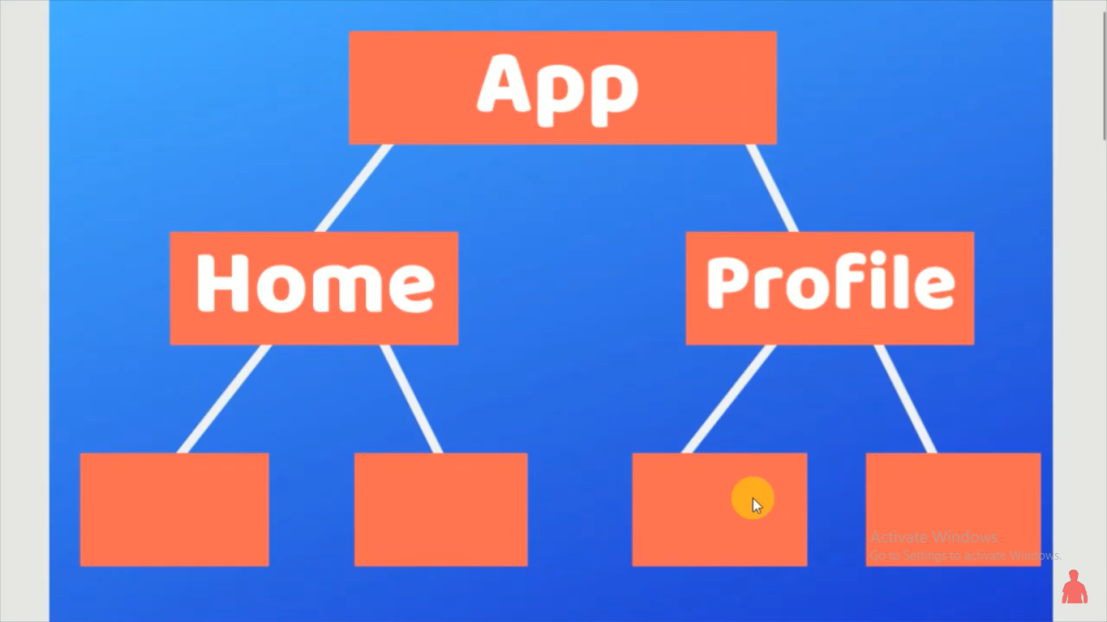
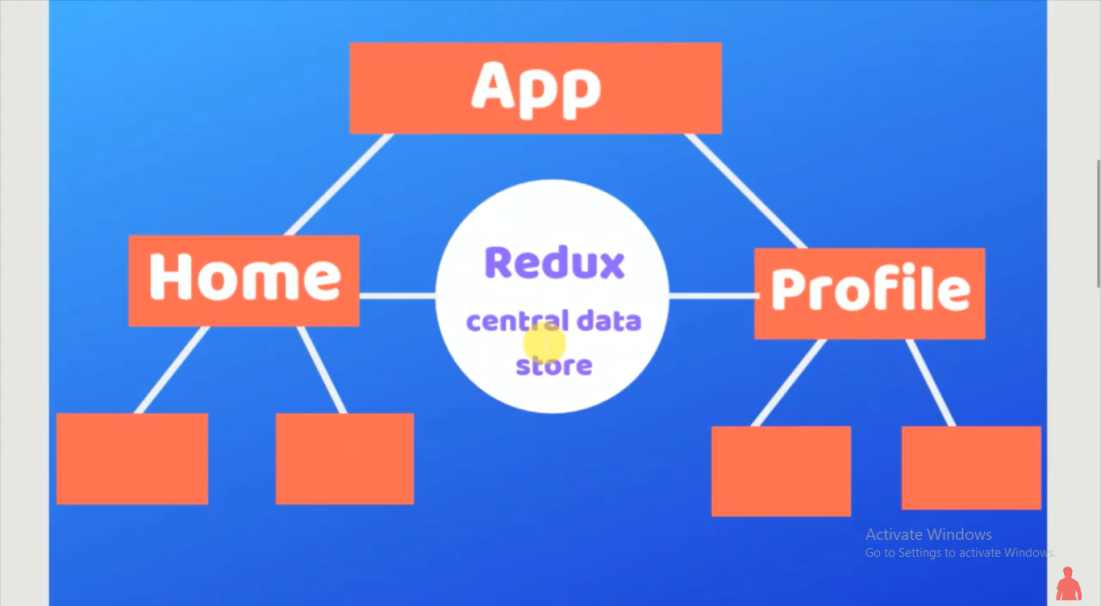
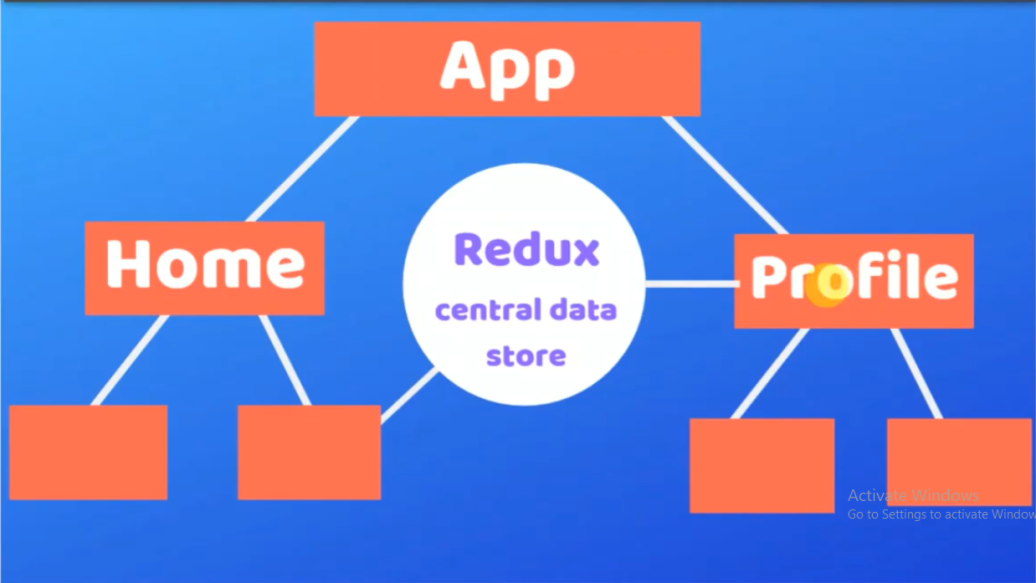
Redux

* It is the state management

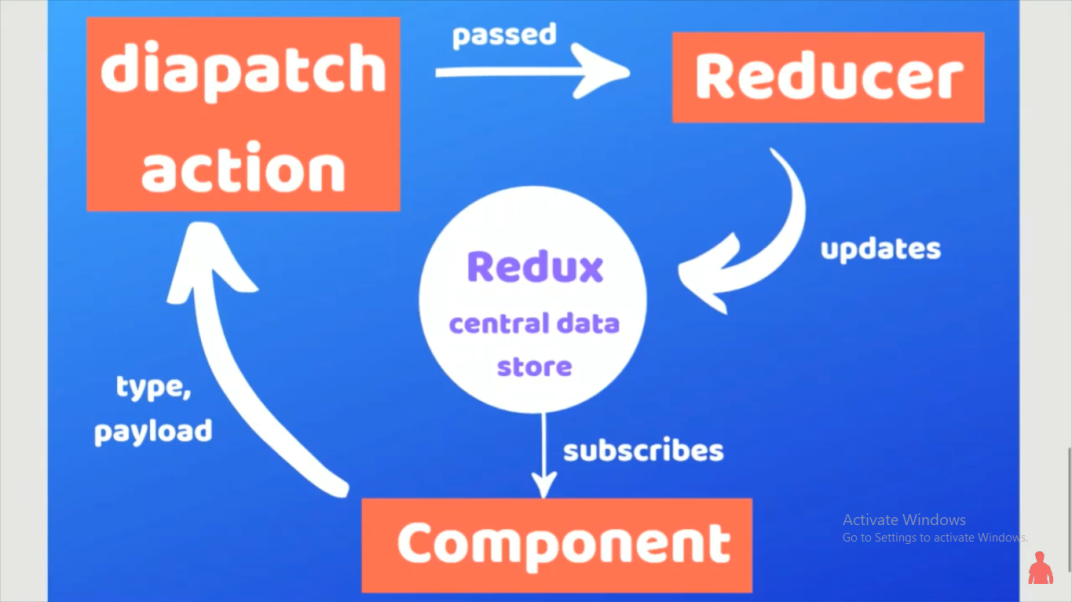
Suppose we want to pass data from Home to Profile. We need to pass from Home to App and App to Profile or if we want to pass from grandchild to another grand child, it will be too complicated. Hence comes the concept of Redux. 

* So Redux stores data in the central storage (i.e. reducer) any component can use data





* Redux makes data transfer easier between any component
* If Profile component want to update the state, it cannot directly update the state. It follows redux steps



STEP1 – If component want to use some data he should subscribe to Redux

STEP2 – Next to update state component have to dispatch action. Now for action dispatch we pass following parameters:

Type – state to update eg. name

Payload – new data assign to name

STEP3 – data from dispatch action goes to Reducer and reducer updates state value to Redux

Installation:

* For this two package needs to install redux and react-redux

Npm install redux react-redux

STEP1 – Create the store (where data needs to store)

Index.js

import React from "react";

import ReactDOM from "react-dom";

import "./index.css";

import App from "./App";

import \* as serviceWorker from "./serviceWorker";

import { createStore } from "redux";

import { Provider } from "react-redux";

const store = createStore(); // Created a central store

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById("root")

);

// If you want your app to work offline and load faster, yc

// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();

STEP2: Create the Reducer

* Now createStore accepts reducer as a parameter.
* We will create reducer (a functional component) in src/reducers/reducer.js

Reducer.js

const iState = {

  name: "Aamir",

  wished: "Workout, code"

};

const reducer = (state = iState, action) => {

  return state;

};

export default reducer;

index.js

import React from "react";

import ReactDOM from "react-dom";

import "./index.css";

import App from "./App";

import \* as serviceWorker from "./serviceWorker";

import { createStore } from "redux";

import { Provider } from "react-redux";

import reducers from "./reducers/reducer";

const store = createStore(reducers);   // create store accepts reducers as parameter i.e. the initial data

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById("root")

);

// If you want your app to work offline and load faster, you can change

// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();

STEP3: App Component Subscribe data from Reducer

* This is just a way to access data from our central data i.e. reducer

App.js

import React from "react";

import "./App.css";

import { connect } from "react-redux";

function App(props) {

  console.log(props); // getting below data as props

  return <div className="App">App Components</div>;

}

const mapStatetoProps = state => {

  return {                  // all data inside reducer is accessed by state

    myname: state.name,     // Inside App component is used as props

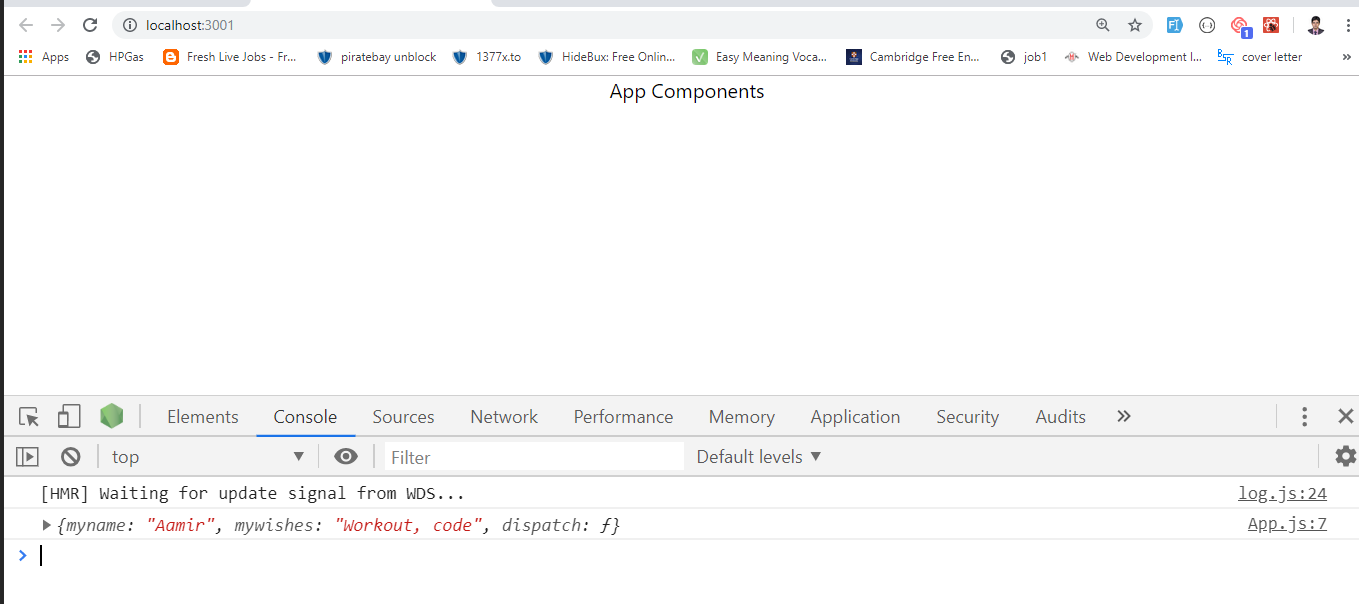
    mywishes: state.wishes

  };

};

// export default App;

export default connect(mapStatetoProps)(App); // connect is Higher Order Component(H.O.C)



STEP4: Update data in reducer

* To update data in reducer we have to create a new function mapDispatchToProps in which function defn return is written in return object.
* Type – key, payload – new value to update

App.js

import React from "react";

import "./App.css";

import { connect } from "react-redux";

function App(props) {

  console.log(props);

  return (

    <div className="App">

      <h2>App Components</h2>

      <h2> My name is {props.myname}</h2>

      <button onClick={() => {props.changeName('Suresh')}}>Change it</button>

    </div>

  );

}

const mapStatetoProps = state => {

  return {

    // all data inside reducer is accessed by state

    myname: state.name, // Inside App component is used as props

    mywishes: state.wishes

  };

};

// The dispatch method takes the changing data which triggers on button click

const mapDispatchToProps = dispatch => {  // changeName is auto saved as props

  return { // Inside name parameter data is received from btn click

    changeName: (name) => { // LHS = function name, RHS = arrow function which updates data

      dispatch({ type: "CHANGE\_NAME", payload: name });

    }

  };

};

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

reducer.js – The changed data is received by reducer as action

const iState = {

  name: "Aamir",

  wishes: "Workout, code"

};

const reducer = (state = iState, action) => {

  console.log(action); // here we get ChangeName on button click

  return state;

};

export default reducer;

STEP5 – Now as we have data in reducer action, we will write logic to update the state

const iState = {

  name: "Aamir",

  wishes: "Workout, code"

};

const reducer = (state = iState, action) => {

  // console.log(action); - normally people use switch when many data is used

if (action.type === "CHANGE\_NAME") {

    return {

… state, // this gets prev data as well

      name: action.payload

    };

 }

  return state;

};

export default reducer;

STEP6 - Add Redux Debugging Tool to easy check

STEP A - install chrome extension - Redux dev tools (initially grey)

STEP B - npm install --save redux-devtools-extension

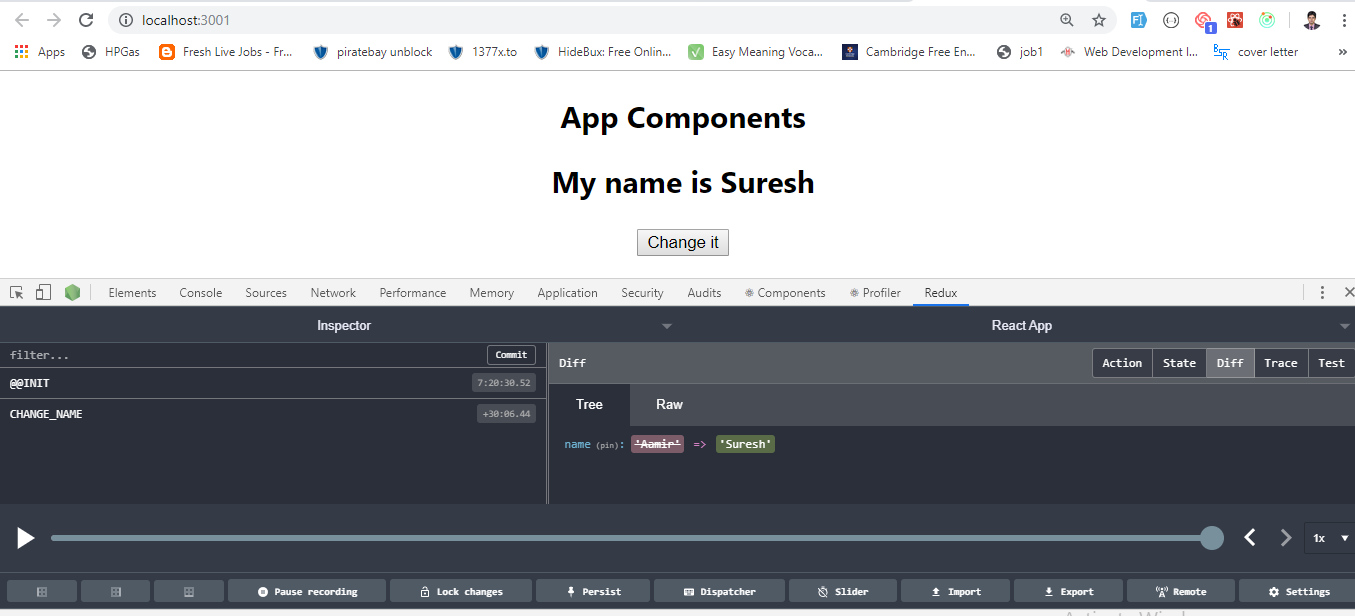
STEP C- In your store page (i.e. index.js)

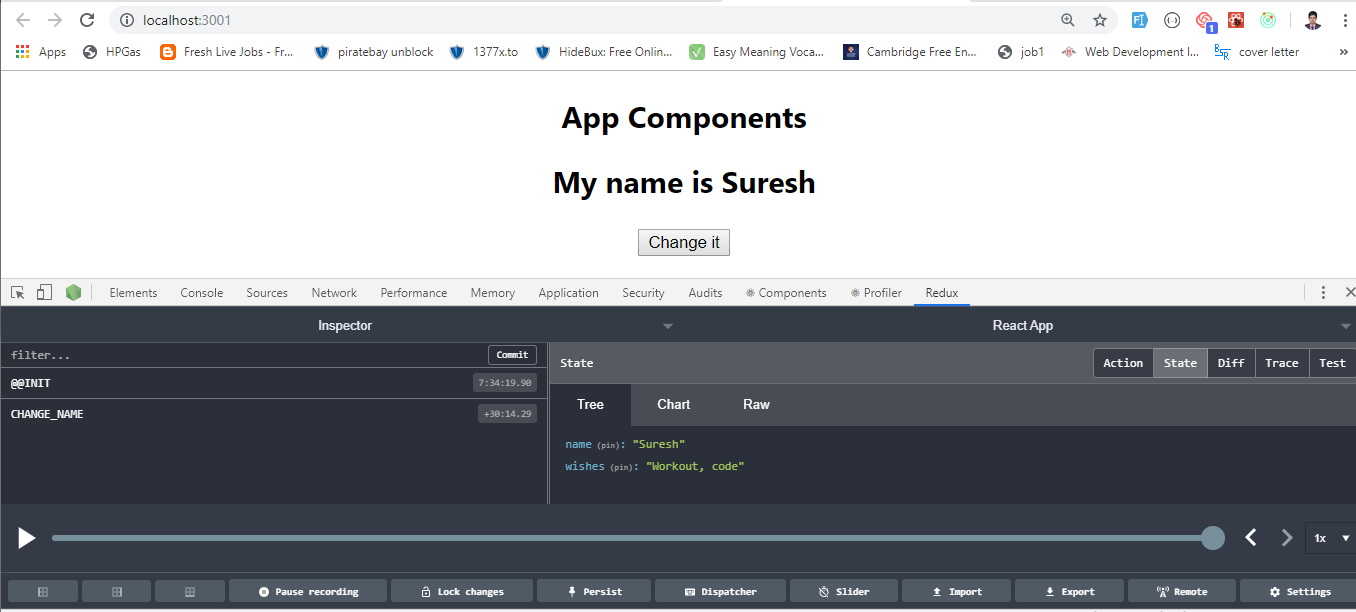
import { createStore } from "redux";

import { composeWithDevTools } from "redux-devtools-extension";

const store = createStore(reducers, composeWithDevTools());

STEP D - You extension icon color should be green now. Inspect Element --> Redux --> Now change any state it will show there





**Action Creator**

* Now whatever we have written in dispatch i.e. object of type and payload can be written in separate page (i.e. action creator) and imported here
* Create a new directory inside src/actions/myaction.js

myaction.js

export const anotherName = name => {

  return {

    type: "CHANGE\_NAME",

    payload: name

  };

};

App.js

import React from "react";

import "./App.css";

import { connect } from "react-redux";

import { anotherName } from "./actions/myaction";

function App(props) {

  console.log(props);

  return (

    <div className="App">

      <h2>App Components</h2>

      <h2> My name is {props.myname}</h2>

      <button

        onClick={() => {

          props.changeName("Suresh");

        }}

      >

        Change it

      </button>

    </div>

  );

}

const mapStatetoProps = state => {

  return {

    // all data inside reducer is accessed by state

    myname: state.name, // Inside App component is used as props

    mywishes: state.wishes

  };

};

// The dispatch method will takes the changing data which will trigger on button click

const mapDispatchToProps = dispatch => {

  return {

    changeName: name => {

      dispatch(anotherName(name)); // replacd by action creator

    }

    /\* changeName: name => {

      dispatch({ type: "CHANGE\_NAME", payload: name });

    } \*/

  };

};

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

* Main purpose to make action creator is to make reusable.
* Action creator is widely used with redux-thunk package.
* Thunk package is basically used to handle asynchronous operations

Redux Thunk

* Redux thunk is basically used when we want to fetch data asynchronously

npm install redux-thunk

STEP1: First update import thunk + redux dev tool

Index.js (snippet)

import { Provider } from "react-redux";

import reducers from "./reducers/reducer";

import thunk from "redux-thunk"; // added thunk

import { createStore, applyMiddleware } from "redux"; // applyMiddleware for async operation

import { composeWithDevTools } from "redux-devtools-extension"; // we will wrap middleware with this to use debugging tool

// const store = createStore(reducers, composeWithDevTools());

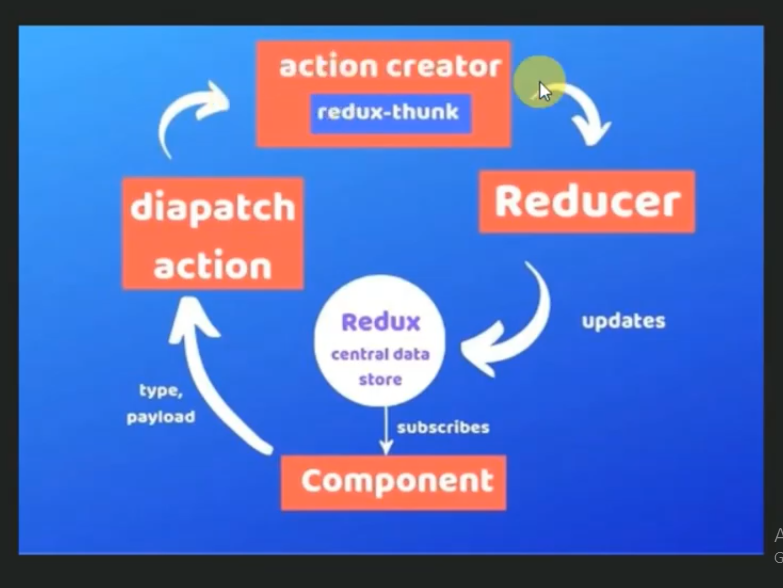
const store = createStore(

  reducers,

  composeWithDevTools(applyMiddleware(thunk))

);

Follow 1.3 section -<https://github.com/zalmoxisus/redux-devtools-extension>



* Earlier what we were doing data was passing directly from **dispatch to reducer** (i.e. mapDispatchtoProps to action parameter of reducer)
* Now **asycn operation** is done between dispatch and reducer cycle and data will be transferred once dispatch is done (i.e. data is fetched). This is done with the help of redux thunk
* Changes are done in action creator page (i.e. myaction.js)

myaction.js

// Earlier we were returning an object

/\* export const anotherName = name => {

  return {

    type: "CHANGE\_NAME",

    payload: name

  };

};

 \*/

// (1st Way) upload data using fetch async - Fetching dummy data from json placeholder (Promise)

export const anotherName = name => {

  return dispatch => {

    fetch("https://jsonplaceholder.typicode.com/users")

      .then(res => res.json())

      .then(res => {

        dispatch({ type: "CHANGE\_NAME", payload: res[0].name });

      });

  };

};

//(2nd Way) upload data using fetch async - Fetching dummy data from json placeholder (ES8 async/await)

export const anotherName = name => {

  return async dispatch => {

    const res = await (await fetch("https://jsonplaceholder.typicode.com/users")).json();

    dispatch({ type: "CHANGE\_NAME", payload: res[0].name });

  };

};

<https://www.loom.com/share/2c8f9912aacb4faab07a1e650ee59ad6>

Multiple Reducers / Combining Reducers

Step1: Create more than one reducer say nameReducer.js and wishReducer

nameReducer.js – here we have removed state default data and set to empty

const nameReducer = (state = "", action) => {

if (action.type === "CHANGE\_NAME") {

    return action.payload; // multiple reducer

    /\* return { // single reducer

      ...state,

      name: action.payload

    }; \*/

  }

  return state;

};

export default nameReducer;

wishReducer.js

const wishReducer = (state = [], action) => {

return state;

};

export default wishReducer;

index.js

* Here we are importing both the reducer and using **combineReducers** to merge

import React from "react";

import ReactDOM from "react-dom";

import "./index.css";

import App from "./App";

import \* as serviceWorker from "./serviceWorker";

import { Provider } from "react-redux";

import thunk from "redux-thunk"; // added thunk

import { createStore, applyMiddleware, combineReducers } from "redux"; // applyMiddleware for async operation

import { composeWithDevTools } from "redux-devtools-extension"; // we will wrap middleware with this to use debugging tool

import nameReducers from "./reducers/nameReducer"; // Reducer 1

import wishReducers from "./reducers/wishReducer"; // Reducer 2

// combining reducers

const masterReducer = combineReducers({

  name: nameReducers,

  wish: wishReducers

});

const store = createStore(

  masterReducer, // combined reducer

  { name: "Amir Mustafa", wish: ["eat", "sleep"] }, // default data for reducer - earlier we passed in reducer page itself. Withot this like data will show empty initially

  composeWithDevTools(applyMiddleware(thunk))

);

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById("root")

);

// If you want your app to work offline and load faster, you can change

// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();

Eg2: We are making

Index.js (nothing changed)

import React from "react";

import ReactDOM from "react-dom";

import "./index.css";

import App from "./App";

import \* as serviceWorker from "./serviceWorker";

import { Provider } from "react-redux";

import thunk from "redux-thunk"; // added thunk

import { createStore, applyMiddleware, combineReducers } from "redux"; // applyMiddleware for async operation

import { composeWithDevTools } from "redux-devtools-extension"; // we will wrap middleware with this to use debugging tool

import nameReducers from "./reducers/nameReducer";

import wishReducers from "./reducers/wishReducer";

// combining reducers

const masterReducer = combineReducers({

  name: nameReducers,

  wish: wishReducers

});

const store = createStore(

  masterReducer, // combined reducer

  { name: "Amir Mustafa", wish: ["eat", "sleep"] }, // default data for reducer - earlier we passed in reducer page itself

  composeWithDevTools(applyMiddleware(thunk))

);

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById("root")

);

// If you want your app to work offline and load faster, you can change

// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();

nameReducer.js (Reducer 1 - nothing changed)

const nameReducer = (state = "", action) => {

  // console.log(action); - normally people use switch when many data is used

  if (action.type === "CHANGE\_NAME") {

    return action.payload; // multiple reducer

}

  return state;

};

export default nameReducer;

wishReducer.js

const wishReducer = (state = [], action) => {

  if (action.type === "ADD\_WISH") { // when updating

    return [...state, "code"];

  }

  return state; // at the time of load

};

export default wishReducer;

myaction.js – Action Creator (here we are passing type and payload value)

export const anotherName = name => {

  return dispatch => {

    fetch("https://jsonplaceholder.typicode.com/users")

      .then(res => res.json())

      .then(res => {

        dispatch({ type: "CHANGE\_NAME", payload: res[0].name });

      });

  };

};

export const addwish = wish => {// value of wish is coming from props of App component

  return {

    type: "ADD\_WISH",

    payload: wish

  };

};

App.js

import React from "react";

import "./App.css";

import { connect } from "react-redux";

import { anotherName, addwish } from "./actions/myaction";

function App(props) {

  const myWish = props.mywishes.map((current, i) => {

    return <li key={i}>{current}</li>;

  });

  console.log(props);

  return (

    <div className="App">

      <h2>App Components</h2>

      <h2> My name is {props.myname}</h2>

      <p>My Wishes list</p>

      <ul>{myWish}</ul>

      <button

        onClick={() => {

          props.changeName("Suresh");

        }}

      >

        Change it

      </button>

      <button

        onClick={() => {

          props.addWish("Code");

        }}

      >

        Add wish

      </button>

    </div>

  );

}

const mapStatetoProps = state => {

  return {

    // all data inside reducer is accessed by state

    myname: state.name, // Inside App component is used as props

    mywish: state.wish

  };

};

// The dispatch method will takes the changing data which will trigger on button click

const mapDispatchToProps = dispatch => {

  return {

    changeName: name => {

      dispatch(anotherName(name)); // replacd by action creators

    },

    addWish: newwish => {

      dispatch(addwish(newwish)); // data we are writing in action creator

    }

  };

};

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

Video - <https://www.loom.com/share/1256b2e268cd4210abb115bf6b05c26c>

Ref - <https://www.youtube.com/watch?v=ahZPLgj1tnY&list=PLB97yPrFwo5gpct1dOsirrotuPef1xBWS&index=1>