Contents

[WHAT IS REACT? 2](#_Toc77058953)

[UNDERSTANDING REDUX FLOW: 3](#_Toc77058954)

[SETTING UP REDUCES AND STORE: 3](#_Toc77058955)

[Dispatching Actions: 6](#_Toc77058956)

[Adding Subscription: 7](#_Toc77058957)

START : [Connecting Redux to React: 9](#_Toc77058958)

[Connecting store to react 11](#_Toc77058959)

[Dispatching actions from within components: 13](#_Toc77058960)

[PASSING AND RETRIEVING DATA WITH ACTIONS 18](#_Toc77058961)

[SWITCH CASE IN REDUCER: 19](#_Toc77058962)

[UPDATING STATE IMMUTABLY 20](#_Toc77058963)

[UPDATING ARRAYS IMMUTABLY[DELETE RESULT]: 24](#_Toc77058964)

[OUTSOURCING ACTION TYPES [IMPORTANT FOR HUGE APP] 28](#_Toc77058965)

[COMBIING MULTIPLE REDUCERS 30](#_Toc77058966)

[UNDERSTANDING TYPES OF STATE [where we should use redux to manage the state] 34](#_Toc77058967)

INSTALLATION: TWO PACKAGES

npm install - -save redux react-redux

STEPS:

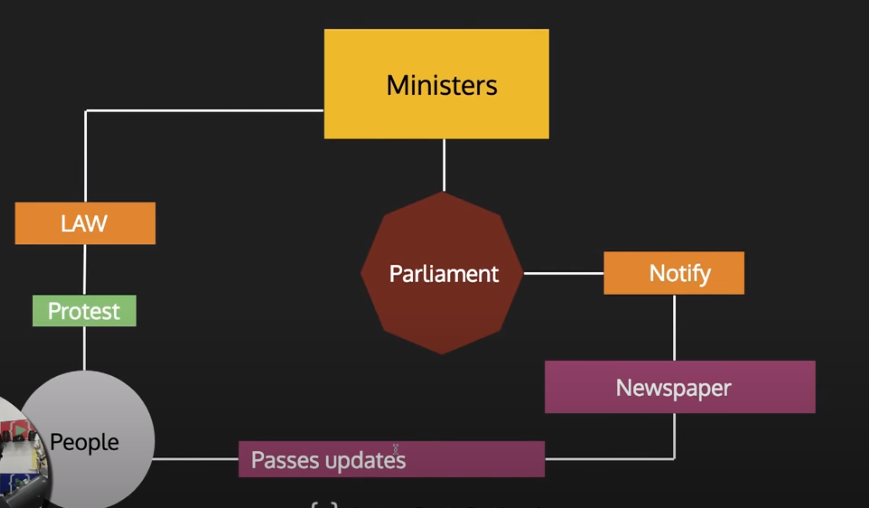
1. Create reducer.js (switch cases: managing states) inside store folder.
2. Create actions.js (action types constant) inside store folder.
3. Connect reducer and importing {createStore} from redux and {Provider} from react-redux in index.js.
4. Tips: …state in reducer.js is to copy old array into new array.
5. Create mapStateToProps for access to our state and mapDispatchToProps to dispatch actions in state.
6. Connect these two const functions to component.

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

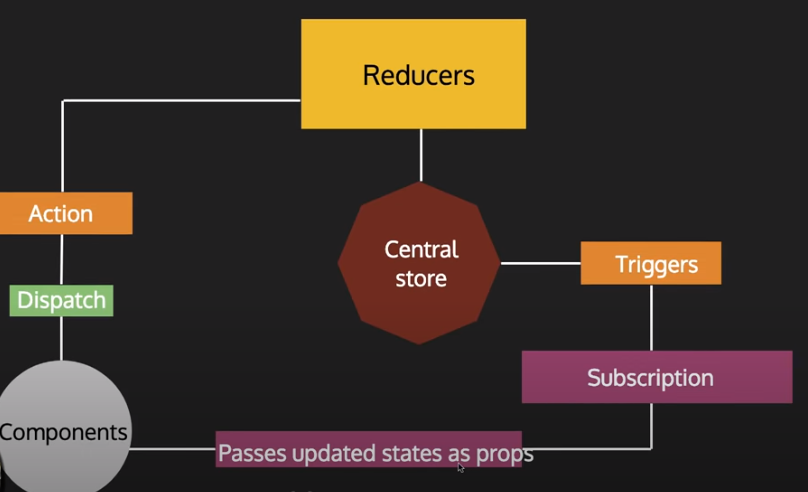
1. Index.js : ReactDOM.render(<Provider store={store}><App />………….

# WHAT IS REACT?

Example: If we normal PEOPLE want to change the LAW. We have to PROTEST to change the LAW. These PROTEST will be reach by MINISTERS. MINISTERS will tries to update the law in PARLIAMENT. It will be notify to NEWSPAPER. We normal PEOPLE are already have link with NEWSPAPER then PEOPLE will get updates about new LAW.

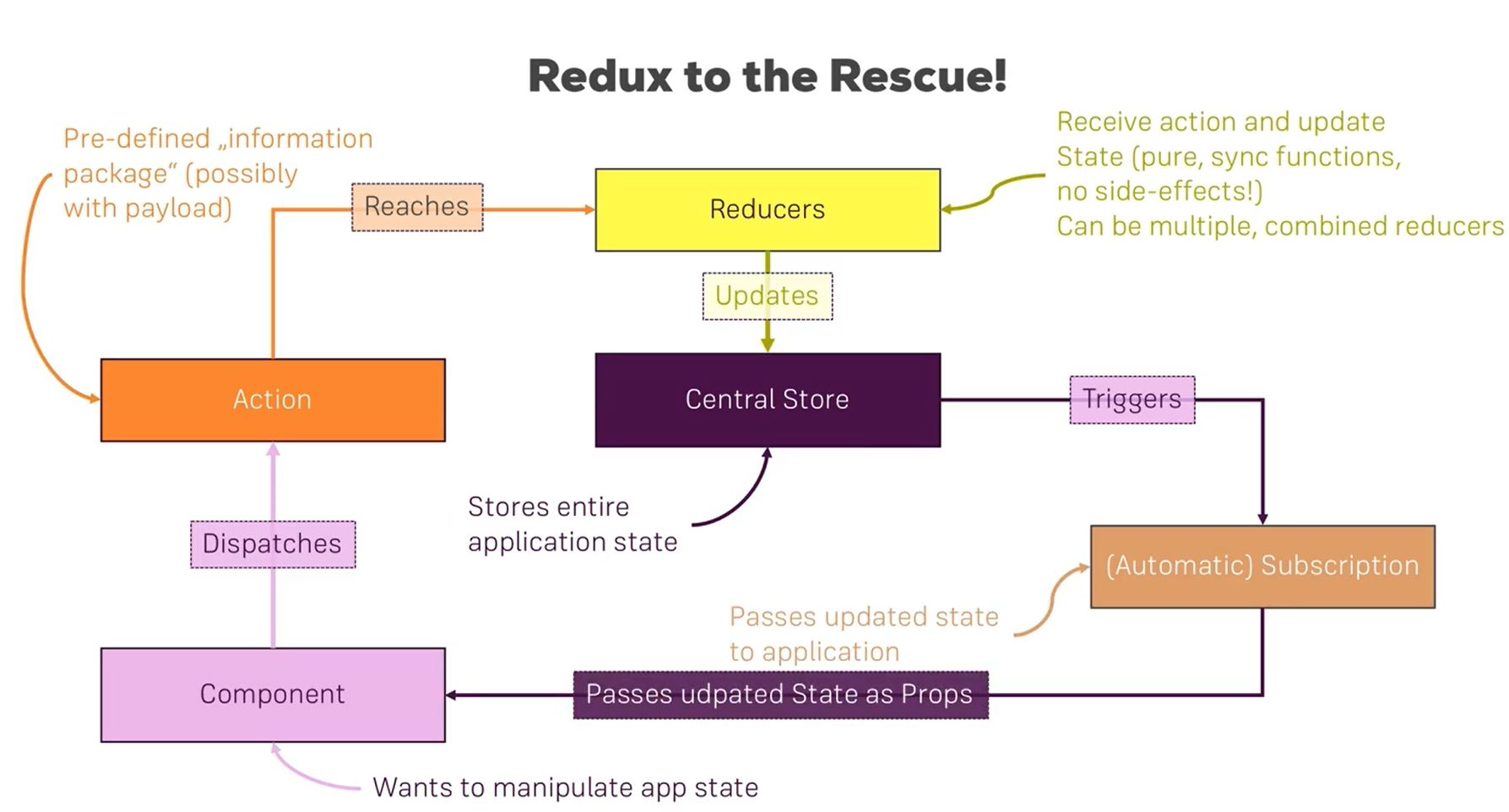


Actual REDUX:



Dispatch Meaning: the sending of someone or something to a destination or for a purpose.

# UNDERSTANDING REDUX FLOW:

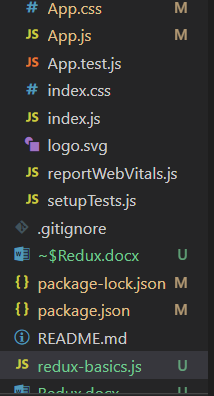


# SETTING UP REDUCES AND STORE:

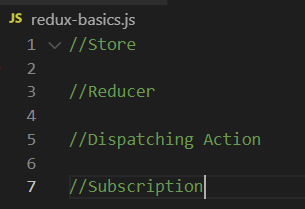
1. INSTALLATION OF REDUX LIBRARY

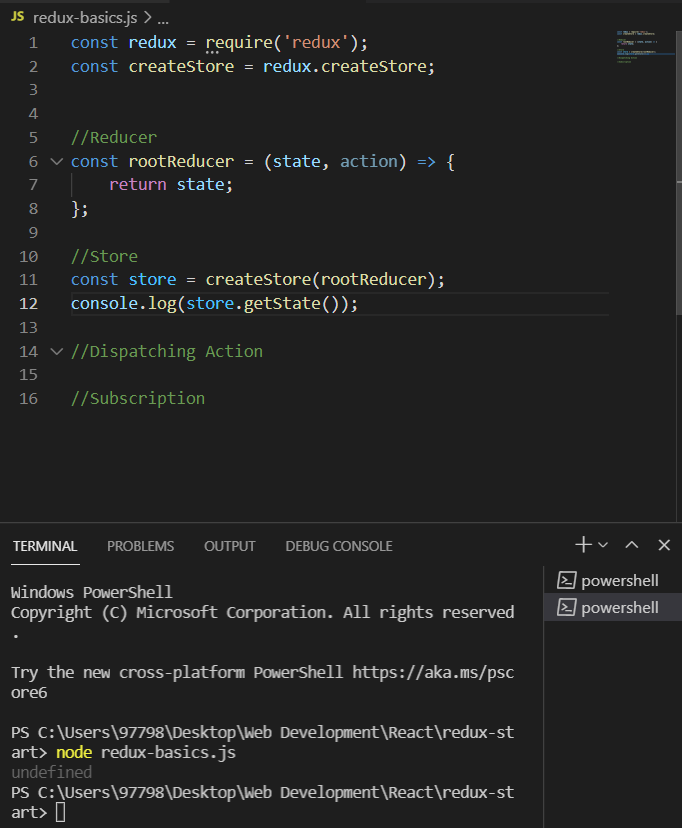
npm install –save redux

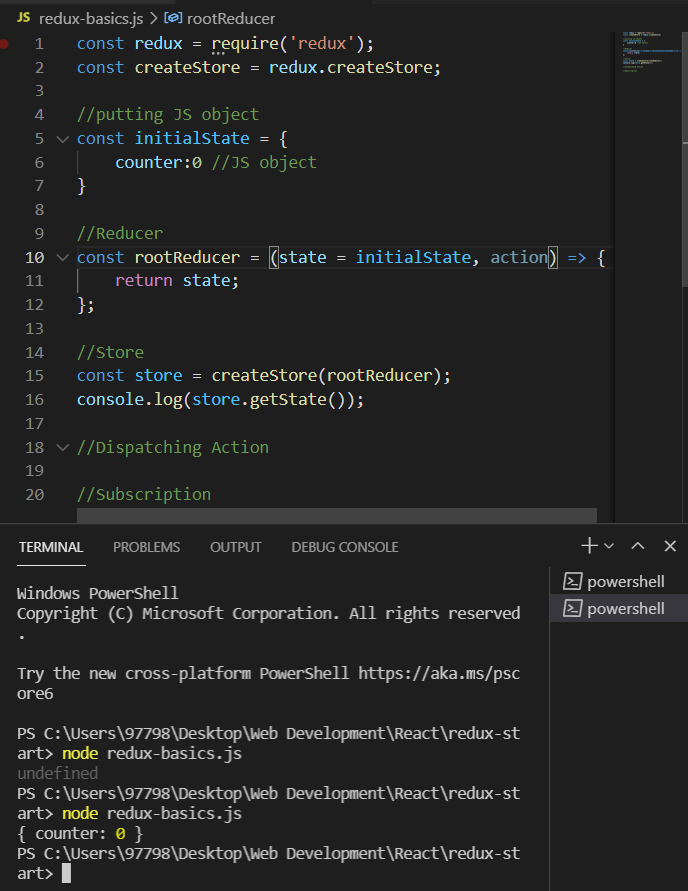
1. Create redux-basics.js file



1. This file does not included inside main project. This is just to make you understand show how redux works through single file.



1. Require will import redux. It will run by node js.  
   
2. Finally



# Dispatching Actions:

Redux-basics.js

const redux = require('redux');

const createStore = redux.createStore;

//putting JS object

const initialState = {

    counter:0 //JS object

}

//Reducer

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

    if(action.type === 'INC\_COUNTER'){

        return {

            //new JS object

            ...state,

            counter: state.counter+1 //This is also JS object. getting access to old state counter

        };

    }

    if(action.type === 'ADD\_COUNTER'){

        return {

            //new JS object

            ...state,

            counter: state.counter+action.value //This is also JS object. getting access to old state counter

        };

    }

    return state;

};

//Store

const store = createStore(rootReducer);

console.log(store.getState());

//Dispatching Action

store.dispatch({type: 'INC\_COUNTER'});//ALL UPPERCASE

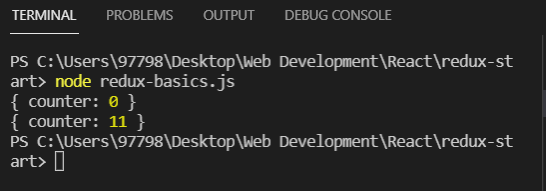
// store.dispatch({type: 'ADD\_COUNTER',value,name,id});//ALL UPPERCASE

store.dispatch({type: 'ADD\_COUNTER',value: 10});//ALL UPPERCASE

console.log(store.getState())

//Subscription

Console:



# Adding Subscription:

const redux = require('redux');

const createStore = redux.createStore;

//putting JS object

const initialState = {

    counter:0 //JS object

}

//Reducer

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

    if(action.type === 'INC\_COUNTER'){

        return {

            //new JS object

            ...state,

            counter: state.counter+1 //This is also JS object. getting access to old state counter

        };

    }

    if(action.type === 'ADD\_COUNTER'){

        return {

            //new JS object

            ...state,

            counter: state.counter+action.value //This is also JS object. getting access to old state counter

        };

    }

    return state;

};

//Store

const store = createStore(rootReducer);

console.log(store.getState());

//Subscription

store.subscribe(() => {

    //any code to update state

    console.log('[Subscription]', store.getState());

});

//Dispatching Action

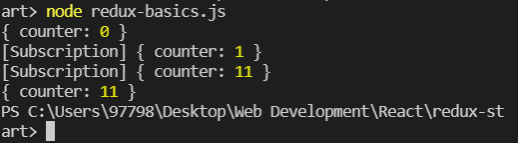
store.dispatch({type: 'INC\_COUNTER'});//ALL UPPERCASE

// store.dispatch({type: 'ADD\_COUNTER',value,name,id});//ALL UPPERCASE

store.dispatch({type: 'ADD\_COUNTER',value: 10});//ALL UPPERCASE

console.log(store.getState())

Console:



# Connecting Redux to React(Dispatches):

Index.js

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import {createStore}  from 'redux';

//create store

const store = createStore();

ReactDOM.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>,

  document.getElementById('root')

);

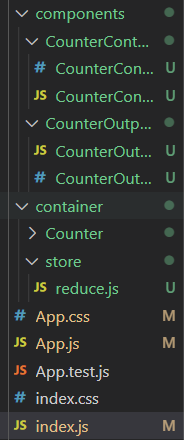
// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

Create store folder where we will create reduce.js file.



Reduce.js

const initialState={

    counter: 0

}

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

    return state;

};

export default reducer;

Finally, Index.js

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import {createStore}  from 'redux';

import reducer from './container/store/reduce'

//create store

const store = createStore(reducer);

ReactDOM.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>,

  document.getElementById('root')

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

# COUNTER: Connecting store to react

Installation:

npm install --save react-redux

Index.js

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import {createStore}  from 'redux';

import { Provider } from 'react-redux';

import reducer from './container/store/reduce'

//create store

const store = createStore(reducer);

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById('root')

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

Counter.js

NOTE: connect works as subscription

import React, { Component } from 'react';

import CounterControl from '../../components/CounterControl/CounterControl';

import CounterOutput from '../../components/CounterOutput/CounterOutpu';

import {connect} from 'react-redux';

class Counter extends Component {

    state = {

        counter: 0

    }

    counterChangedHandler = ( action, value ) => {

        switch ( action ) {

            case 'inc':

                this.setState( ( prevState ) => { return { counter: prevState.counter + 1 } } )

                break;

            case 'dec':

                this.setState( ( prevState ) => { return { counter: prevState.counter - 1 } } )

                break;

            case 'add':

                this.setState( ( prevState ) => { return { counter: prevState.counter + value } } )

                break;

            case 'sub':

                this.setState( ( prevState ) => { return { counter: prevState.counter - value } } )

                break;

        }

    }

    render () {

        return (

            <div>

                <CounterOutput value={this.props.ctr} />

                <CounterControl label="Increment" clicked={() => this.counterChangedHandler( 'inc' )} />

                <CounterControl label="Decrement" clicked={() => this.counterChangedHandler( 'dec' )}  />

                <CounterControl label="Add 5" clicked={() => this.counterChangedHandler( 'add', 5 )}  />

                <CounterControl label="Subtract 5" clicked={() => this.counterChangedHandler( 'sub', 5 )}  />

            </div>

        );

    }

}

const mapStateToProps = state =>{

    return{

        ctr: state.counter

    };

}

export default connect(mapStateToProps)(Counter);

# Dispatching actions from within components:

Counter.js

import React, { Component } from 'react';

import CounterControl from '../../components/CounterControl/CounterControl';

import CounterOutput from '../../components/CounterOutput/CounterOutpu';

import {connect} from 'react-redux';

class Counter extends Component {

    state = {

        counter: 0

    }

    counterChangedHandler = ( action, value ) => {

        switch ( action ) {

            case 'inc':

                this.setState( ( prevState ) => { return { counter: prevState.counter + 1 } } )

                break;

            case 'dec':

                this.setState( ( prevState ) => { return { counter: prevState.counter - 1 } } )

                break;

            case 'add':

                this.setState( ( prevState ) => { return { counter: prevState.counter + value } } )

                break;

            case 'sub':

                this.setState( ( prevState ) => { return { counter: prevState.counter - value } } )

                break;

        }

    }

    render () {

        return (

            <div>

                <CounterOutput value={this.props.ctr} />

                <CounterControl label="Increment" clicked={this.props.onIncrementCounter} />

                <CounterControl label="Decrement" clicked={() => this.counterChangedHandler( 'dec' )}  />

                <CounterControl label="Add 5" clicked={() => this.counterChangedHandler( 'add', 5 )}  />

                <CounterControl label="Subtract 5" clicked={() => this.counterChangedHandler( 'sub', 5 )}  />

            </div>

        );

    }

}

const mapStateToProps = state =>{

    return{

        ctr: state.counter

    };

}

const mapDispatchToProps = dispatch => {

    return {

        //onIncrementCounter=properties

        onIncrementCounter: () => dispatch({

            type: 'INCREMENT'

        }) //Anayamous Function

    };

}

// export default connect(null,mapDispatchToProps)(Counter);

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

Reducer.js

const initialState={

    counter: 0

}

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

    if(action.type === 'INCREMENT'){

        return{

            counter: state.counter + 1

        }

    }

    return state;

};

export default reducer;

FINAL

Counter.js

import React, { Component } from 'react';

import CounterControl from '../../components/CounterControl/CounterControl';

import CounterOutput from '../../components/CounterOutput/CounterOutpu';

import {connect} from 'react-redux';

class Counter extends Component {

    state = {

        counter: 0

    }

    counterChangedHandler = ( action, value ) => {

        switch ( action ) {

            case 'inc':

                this.setState( ( prevState ) => { return { counter: prevState.counter + 1 } } )

                break;

            case 'dec':

                this.setState( ( prevState ) => { return { counter: prevState.counter - 1 } } )

                break;

            case 'add':

                this.setState( ( prevState ) => { return { counter: prevState.counter + value } } )

                break;

            case 'sub':

                this.setState( ( prevState ) => { return { counter: prevState.counter - value } } )

                break;

        }

    }

    render () {

        return (

            <div>

                <CounterOutput value={this.props.ctr} />

                <CounterControl label="Increment" clicked={this.props.onIncrementCounter} />

                <CounterControl label="Decrement" clicked={this.props.onDecrementCounter}  />

                <CounterControl label="Add 5" clicked={this.props.onAddCounter}  />

                <CounterControl label="Subtract 5" clicked={this.props.onSubtractCounter}  />

            </div>

        );

    }

}

const mapStateToProps = state =>{

    return{

        ctr: state.counter

    };

}

const mapDispatchToProps = dispatch => {

    return {

        //onIncrementCounter=properties

        onIncrementCounter: () => dispatch({

            type: 'INCREMENT'

        }),

        //Anayamous Function

        //onIncrementCounter=properties

        onDecrementCounter: () => dispatch({

            type: 'DECREMENT'

        }),

         //Anayamous Function

        //onIncrementCounter=properties

        onAddCounter: () => dispatch({

            type: 'ADD'

        }),

        //Anayamous Function

        //onIncrementCounter=properties

        onSubtractCounter: () => dispatch({

            type: 'SUBTRACT'

        }) //Anayamous Function

    };

}

// export default connect(null,mapDispatchToProps)(Counter);

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

Reducer.js

const initialState={

    counter: 0

}

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

    if(action.type === 'INCREMENT'){

        return{

            counter: state.counter + 1

        }

    }

    if(action.type === 'DECREMENT'){

        return{

            counter: state.counter - 1

        }

    }

    if(action.type === 'ADD'){

        return{

            counter: state.counter + 10

        }

    }

    if(action.type === 'SUBTRACT'){

        return{

            counter: state.counter - 8

        }

    }

    return state;

};

export default reducer;

# PASSING AND RETRIEVING DATA WITH ACTIONS

Counter.js

const mapDispatchToProps = dispatch => {

    return {

        //onIncrementCounter=properties

        onIncrementCounter: () => dispatch({

            type: 'INCREMENT'

        }),

        //Anayamous Function

        //onIncrementCounter=properties

        onDecrementCounter: () => dispatch({

            type: 'DECREMENT'

        }),

         //Anayamous Function

        //onIncrementCounter=properties

        onAddCounter: () => dispatch({

            type: 'ADD',

            value: 10,

            name:'Shakti'

        }),

        //Anayamous Function

        //onIncrementCounter=properties

        onSubtractCounter: () => dispatch({

            type: 'SUBTRACT',

            value: 15

        }) //Anayamous Function

    };

}

Reducer.js

if(action.type === 'ADD'){

        return{

            counter: state.counter + action.value

        }

    }

    if(action.type === 'SUBTRACT'){

        return{

            counter: state.counter - action.value

        }

    }

# SWITCH CASE IN REDUCER:

Reducer.js

const initialState={

    counter: 0

}

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

    switch(action.type){

        case 'INCREMENT':

            return{

                counter: state.counter + 1

            }

        case 'DECREMENT':

            return{

                counter: state.counter - 1

            }

        case 'ADD':

            return{

                counter: state.counter + action.value

            }

        case 'SUBTRACT':

            return{

                counter: state.counter - action.value

            }

    }

    return state;

};

export default reducer;

# UPDATING STATE IMMUTABLY

Counter.js

import React, { Component } from 'react';

import CounterControl from '../../components/CounterControl/CounterControl';

import CounterOutput from '../../components/CounterOutput/CounterOutpu';

import {connect} from 'react-redux';

class Counter extends Component {

    state = {

        counter: 0

    }

    counterChangedHandler = ( action, value ) => {

        switch ( action ) {

            case 'inc':

                this.setState( ( prevState ) => { return { counter: prevState.counter + 1 } } )

                break;

            case 'dec':

                this.setState( ( prevState ) => { return { counter: prevState.counter - 1 } } )

                break;

            case 'add':

                this.setState( ( prevState ) => { return { counter: prevState.counter + value } } )

                break;

            case 'sub':

                this.setState( ( prevState ) => { return { counter: prevState.counter - value } } )

                break;

        }

    }

    render () {

        return (

            <div>

                <CounterOutput value={this.props.ctr} />

                <CounterControl label="Increment" clicked={this.props.onIncrementCounter} />

                <CounterControl label="Decrement" clicked={this.props.onDecrementCounter}  />

                <CounterControl label="Add 5" clicked={this.props.onAddCounter}  />

                <CounterControl label="Subtract 15" clicked={this.props.onSubtractCounter}  />

                <hr/>

                <button onClick={this.props.onStoreResult}>Store Result</button>

                <ul>

                    {this.props.storeResults.map(strResult => (

                        <li key={strResult.id}

onClick={this.props.onDeleteResult}>

{strResult.value}

</li>

                    ))}

                </ul>

            </div>

        );

    }

}

const mapStateToProps = state =>{

    return{

        ctr: state.counter,

        storeResults: state.results

    };

}

const mapDispatchToProps = dispatch => {

    return {

        //onIncrementCounter=properties

        onIncrementCounter: () => dispatch({

            type: 'INCREMENT'

        }),

        //Anayamous Function

        //onIncrementCounter=properties

        onDecrementCounter: () => dispatch({

            type: 'DECREMENT'

        }),

         //Anayamous Function

        //onIncrementCounter=properties

        onAddCounter: () => dispatch({

            type: 'ADD',

            value: 10,

            name:'Shakti'

        }),

        //Anayamous Function

        //onIncrementCounter=properties

        onSubtractCounter: () => dispatch({

            type: 'SUBTRACT',

            value: 15

        }), //Anayamous Function

        onStoreResult: () => dispatch({

            type:'STORE\_RESULT'

        }),

        onDeleteResult: () => dispatch({

            type:'DELETE\_RESULT'

        })

    };

}

// export default connect(null,mapDispatchToProps)(Counter);

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

Reducer.js

const initialState={

    counter: 0,

    //new state

    results: []

}

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

    switch(action.type){

        case 'INCREMENT':

            // return{

            //     counter: state.counter + 1

            // }

            //cloning the state deeply

            const newState = Object.assign({},state);

            newState.counter = state.counter + 1;

            return newState;

        case 'DECREMENT':

            return{

                ...state,//Updating State immutably

                counter: state.counter - 1

            }

        case 'ADD':

            return{

                ...state, //Updating State immutably

                counter: state.counter + action.value

            }

        case 'SUBTRACT':

            return{

                ...state, //Updating State immutably

                counter: state.counter - action.value

            }

        case 'STORE\_RESULT':

            return{

                //return all properties of state

                ...state, //Updating State immutably

                //do not use .push() it will affect original array

                results: state.results.concat({

id: new Date(),

value: state.counter})

            }

        // case 'DELETE\_RESULT':

        //     return{

        //     }

    }

    return state;

};

export default reducer;

# UPDATING ARRAYS IMMUTABLY[DELETE RESULT]:

Reducer.js

const initialState={

    counter: 0,

    //new state

    results: []

}

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

    switch(action.type){

        case 'INCREMENT':

            // return{

            //     counter: state.counter + 1

            // }

            //cloning the state deeply

            const newState = Object.assign({},state);

            newState.counter = state.counter + 1;

            return newState;

        case 'DECREMENT':

            return{

                ...state,

                counter: state.counter - 1

            }

        case 'ADD':

            return{

                ...state,

                counter: state.counter + action.value

            }

        case 'SUBTRACT':

            return{

                ...state,

                counter: state.counter - action.value

            }

        case 'STORE\_RESULT':

            return{

                //return all properties of state

                ...state,

                //do not use .push() it will affect original array

                results: state.results.concat({id: new Date(), value: state.counter})

            }

        case 'DELETE\_RESULT':

            // const id = 2;

            // //copying aray into new array.THis is immutably

            // const newArray = [...state.results]

            // newArray.splice(id,1)//1= amount of data. This is not immutably

            //Another way to update array

            // result => true shortcut for arrow function

            // const updatedArray = state.results.filter((result,index) => index !== id);//filter will return new array

            const updatedArray = state.results.filter(result => result.id !== action.resultElid);//filter will return new array

            return{

                ...state,

                // results: newArray

                results: updatedArray

            }

    }

    return state;

};

export default reducer;

Counter.js

import React, { Component } from 'react';

import CounterControl from '../../components/CounterControl/CounterControl';

import CounterOutput from '../../components/CounterOutput/CounterOutpu';

import {connect} from 'react-redux';

class Counter extends Component {

    state = {

        counter: 0

    }

    counterChangedHandler = ( action, value ) => {

        switch ( action ) {

            case 'inc':

                this.setState( ( prevState ) => { return { counter: prevState.counter + 1 } } )

                break;

            case 'dec':

                this.setState( ( prevState ) => { return { counter: prevState.counter - 1 } } )

                break;

            case 'add':

                this.setState( ( prevState ) => { return { counter: prevState.counter + value } } )

                break;

            case 'sub':

                this.setState( ( prevState ) => { return { counter: prevState.counter - value } } )

                break;

        }

    }

    render () {

        return (

            <div>

                <CounterOutput value={this.props.ctr} />

                <CounterControl label="Increment"

clicked={this.props.onIncrementCounter} />

                <CounterControl label="Decrement" clicked={this.props.onDecrementCounter}  />

                <CounterControl label="Add 5" clicked={this.props.onAddCounter}  />

                <CounterControl label="Subtract 15" clicked={this.props.onSubtractCounter}  />

                <hr/>

                <button onClick={this.props.onStoreResult}>Store Result</button>

                <ul>

                    {this.props.storeResults.map(strResult => (

                        <li key={strResult.id} onClick={() => this.props.onDeleteResult(strResult.id)}>{strResult.value}</li>

                    ))}

                </ul>

            </div>

        );

    }

}

const mapStateToProps = state =>{

    return{

        ctr: state.counter,

        storeResults: state.results

    };

}

const mapDispatchToProps = dispatch => {

    return {

        //onIncrementCounter=properties

        onIncrementCounter: () => dispatch({

            type: 'INCREMENT'

        }),

        //Anayamous Function

        //onIncrementCounter=properties

        onDecrementCounter: () => dispatch({

            type: 'DECREMENT'

        }),

         //Anayamous Function

        //onIncrementCounter=properties

        onAddCounter: () => dispatch({

            type: 'ADD',

            value: 10,

            name:'Shakti'

        }),

        //Anayamous Function

        //onIncrementCounter=properties

        onSubtractCounter: () => dispatch({

            type: 'SUBTRACT',

            value: 15

        }), //Anayamous Function

        onStoreResult: () => dispatch({

            type:'STORE\_RESULT'

        }),

        onDeleteResult: (id) => dispatch({

            type:'DELETE\_RESULT',

            resultElid: id

        })

    };

}

// export default connect(null,mapDispatchToProps)(Counter);

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

NOTE:

// const id = 2;

            // state.results.splice(id,1)//1= amount of elements we want to delete. This is will mutate the elements and it is not immutably.

FOR IMMUTABLE WAY:

//copying original array into new array.

            // const newArray = [...state.results];

            // newArray.splice(id,1)//1= amount of elements we want to delete from updated array.

return{

…state,

results: newArray

}

            //Another way to update array

NOTE: filter() will return new array. Filter will input function in which there will be condition to remove the elements from the updated array or not. result => true is shortcut for arrow function to return true or false.

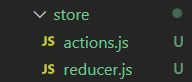
            const updatedArray = state.results.filter((result,index) => index !== id);//this is not proper way

NOTE: result.id is state id whereas resultElid is action id that we are getting from onClick element id action. If these two are not equal then it will return true.

            const updatedArray = state.results.filter(result => result.id !== action.resultElid);

# OUTSOURCING ACTION TYPES [IMPORTANT FOR HUGE APP]

File:



Action.js

export const INCREMENT = 'INCREMENT';

export const DECREMENT = 'DECREMENT';

export const ADD = 'ADD';

export const SUBTRACT = 'SUBTRACT';

export const STORE\_RESULT = 'STORE\_RESULT';

export const DELETE\_RESULT = 'DELETE\_RESULT';

reducer.js

import \* as actionTypes from './actions';

const initialState={

    counter: 0,

    //new state

    results: []

}

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

    switch(action.type){

        case actionTypes.INCREMENT:

            const newState = Object.assign({},state);

            newState.counter = state.counter + 1;

            return newState;

Counter.js

const mapDispatchToProps = dispatch => {

    return {

        //onIncrementCounter=properties

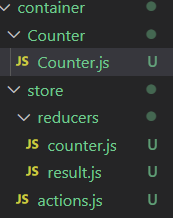
        onIncrementCounter: () => dispatch({

            type: actionTypes.INCREMENT

        }),

# COMBIING MULTIPLE REDUCERS

Create two reducer files named counter.js and result.js.



Counter.js for managing counter state.

import \* as actionTypes from '../actions';

const initialState={

    counter: 0,

}

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

    switch(action.type){

        case actionTypes.INCREMENT:

            const newState = Object.assign({},state);

            newState.counter = state.counter + 1;

            return newState;

        case actionTypes.DECREMENT:

            return{

                ...state,

                counter: state.counter - 1

            }

        case actionTypes.ADD:

            return{

                ...state,

                counter: state.counter + action.value

            }

        case actionTypes.SUBTRACT :

            return{

                ...state,

                counter: state.counter - action.value

            }

    }

    return state;

};

export default reducer;

result.js for managing result state even though they are related to counter’s state. NOTE: results: state.results.concat({id: new Date(), value: state.counter}). We need to update this later on.

import \* as actionTypes from '../actions';

const initialState={

    results: []

}

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

    switch(action.type){

        case actionTypes.STORE\_RESULT:

            return{

                //return all properties of state

                ...state,

                //do not use .push() it will affect original array

                results: state.results.concat({id: new Date(), value: action.result}) //getting counter value gobal state

            }

        case actionTypes.DELETE\_RESULT:

            // const id = 2;

            // //copying aray into new array.THis is immutably

            // const newArray = [...state.results]

            // newArray.splice(id,1)//1= amount of data. This is not immutably

            //Another way to update array

            // result => true shortcut for arrow function

            // const updatedArray = state.results.filter((result,index) => index !== id);//filter will return new array

            const updatedArray = state.results.filter(result => result.id !== action.resultElid);//filter will return new array

            return{

                ...state,

                // results: newArray

                results: updatedArray

            }

    }

    return state;

};

export default reducer;

Index.js for combining two reducers.

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import {createStore,combineReducers}  from 'redux';

import { Provider } from 'react-redux';

import  counterReducer from './container/store/reducers/counter'

import resultReducer from './container/store/reducers/result';

//combine reducer

const rootReducer = combineReducers({

  ctr: counterReducer,

  res: resultReducer

});

//create store

const store = createStore(rootReducer);

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById('root')

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

You may find not map error in your application.

SO, we have to modify our code.

Counter.js

const mapStateToProps = state =>{

    return{

        ctr: state.ctr.counter,

        storeResults: state.res.results

    };

}

Result.js

//Getting value from Global state through actions

                results: state.results.concat({id: new Date(), value: action.result}) //getting counter value gobal state

Counter.js

onStoreResult: (result) => dispatch({

            type:'STORE\_RESULT',

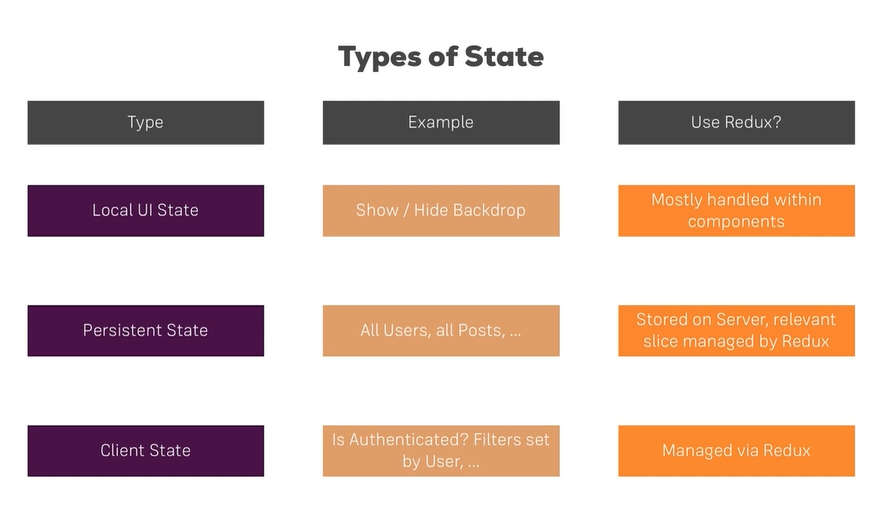
            result: result

        }),

Finally,

 <button onClick={() => this.props.onStoreResult(this.props.ctr)}>Store Result</button>

# UNDERSTANDING TYPES OF STATE [where we should use redux to manage the state]



# ASSINGMENT: