Contents

[Adding Middleware 2](#_Toc77612526)

[USING THE REDUX DEVTOOLS [USEFUL FOR DEBUGGING] 4](#_Toc77612527)

[EXECUTING ASYNCHRONOUS CODE 4](#_Toc77612528)

[What does asynchronous mean? 4](#_Toc77612529)

[INTRODUCTION TO ACTION CREATORS: 4](#_Toc77612530)

[ACTION CREATORS ASYNC CODE 5](#_Toc77612531)

[HANDLING ASYNCHRONOUS CODE [redux thunk]- render action after certain time 8](#_Toc77612532)

[RESTRUCTING ACTIONS[ it can be helpful in big and complex project] 9](#_Toc77612533)

[WHERE TO PUT DATA TRANSFORMING LOGIC 14](#_Toc77612534)

[USING ACTION CREATORS AND GET STATE 15](#_Toc77612535)

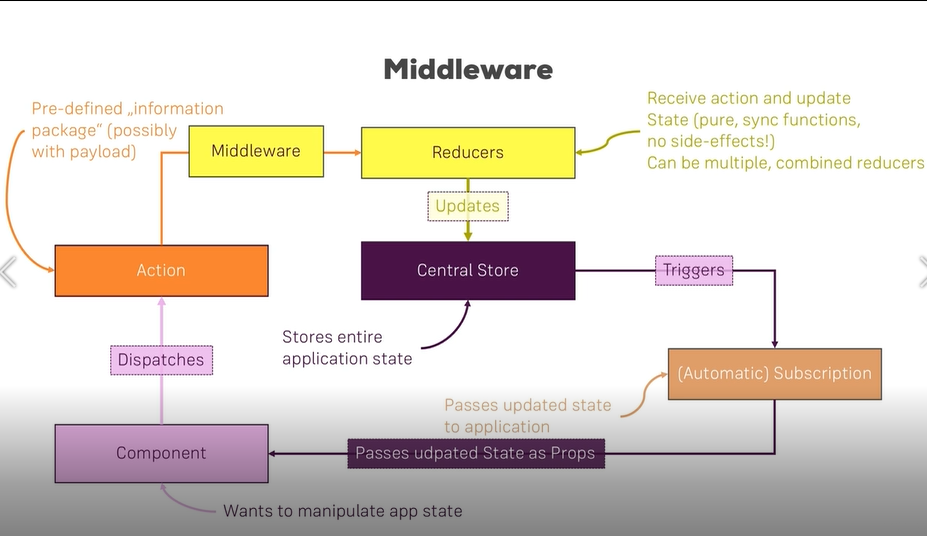
[USING UTLILITY FUNCTIONS 16](#_Toc77612536)

[A LEANER SWITCH CASE STATEMENT [ES6 function to make reducer more understandable] 18](#_Toc77612537)

[Visit Redux Documentation 19](#_Toc77612538)

# Adding Middleware

NOTE: showing how middleware works through index.js



Index.js

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import {createStore,combineReducers,applyMiddleware}  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

});

//middleware

const logger = store => {

  //another function

  return next => {

    //return another function

    return action => {

        console.log('[Middleware] Dispatchin', action);

        const result = next(action);

        console.log('[Middleware] next state', store.getState());

        return result;

      }

  }

}

//create store

const store = createStore(rootReducer, applyMiddleware(logger));

ReactDOM.render(

  <Provider store={store}>

    <App />

  </Provider>,

  document.getElementById('root')

);

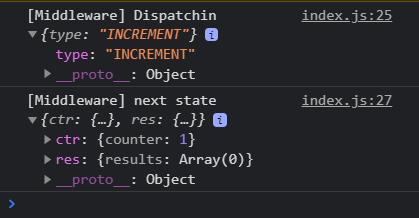
// If you want to start measuring performance in your app, pass a function

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

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

reportWebVitals();

OUTPUT: action and next state is outputted on console.



# USING THE REDUX DEVTOOLS [USEFUL FOR DEBUGGING]

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

# EXECUTING ASYNCHRONOUS CODE

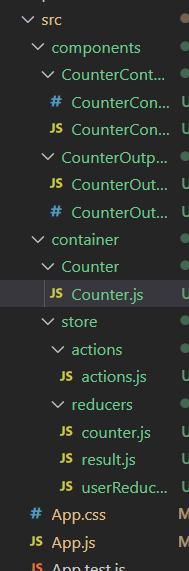
### What does asynchronous mean?

More specifically, asynchronous describes the relationship between two or more events/objects that do interact within the same system but do not occur at predetermined intervals and do not necessarily rely on each other's existence to function. They are not coordinated with each other, meaning they could occur simultaneously or not because they have their own separate agenda.

In computer programming, asynchronous operation means that a process operates independently of other processes, whereas synchronous operation means that the process runs only as a result of some other process being completed or handed off.

# INTRODUCTION TO ACTION CREATORS:

Create sub folder actions inside store.



Actions.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';

//actions for asynchonrous code

//argument can be payload

export const increment = () => {

    // return actions

    return{

        type: INCREMENT

    }

};

Counter.js

import { increment } from '../store/actions/actions';

const mapDispatchToProps = dispatch => {

    return {

        //onIncrementCounter=properties

        onIncrementCounter: () => dispatch(increment()),

# ACTION CREATORS ASYNC CODE

Actions.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';

//actions for asynchonrous code

export const increment = () => {

    // return actions

    return{

        type: INCREMENT

    }

};

export const decrement = () => {

    // return actions

    return{

        type: DECREMENT

    }

};

export const add = (value) => {

    // return actions

    return{

        type: ADD,

        value: value

    }

};

export const substract = (value) => {

    // return actions

    return{

        type: SUBTRACT,

        value: value

    }

};

export const storeResult = (result) => {

    // return actions

    return{

        type: STORE\_RESULT,

        result: result

    }

};

export const deleteResult = (resElId) => {

    // return actions

    return{

        type: DELETE\_RESULT,

        resultElid : resElId

    }

};

Counter.js

// import { increment,decrement,add,substract,storeResult,deleteResult } from '../store/actions/actions';

import \* as actionCreators from '../store/actions/actions';

const mapDispatchToProps = dispatch => {

    return {

        //onIncrementCounter=properties

        onIncrementCounter: () => dispatch(actionCreators.increment()),

        //Anayamous Function

        //onIncrementCounter=properties

        onDecrementCounter: () => dispatch(actionCreators.decrement()),

         //Anayamous Function

        //onIncrementCounter=properties

        onAddCounter: () => dispatch(actionCreators.add(10)),

        //Anayamous Function

        //onIncrementCounter=properties

        onSubtractCounter: () => dispatch(actionCreators.substract(15)), //Anayamous Function

        onStoreResult: (result) => dispatch(

            actionCreators.storeResult(result)

        ),

        onDeleteResult: (id) => dispatch(

            actionCreators.deleteResult(id))

    };

}

ERROR EOE in reducer counter.js

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

# HANDLING ASYNCHRONOUS CODE [redux thunk]- render action after certain time

REDUX THUNK: It is special middleware.

Installation of redux thunk library

npm install –save redux-thunk

index.js

import thunk from 'redux-thunk';

const store = createStore(rootReducer, applyMiddleware(logger,thunk));

actions.js

//new action

export const saveResult = (result ) => {

    return{

        type:STORE\_RESULT,

        result:result

    };

}

export const storeResult = (result) => {

//we get this dispatch with help of redux thunk

    return dispatch => {

        setTimeout(()=>{

            dispatch(saveResult(result));

        },60000)

    }

    // return actions

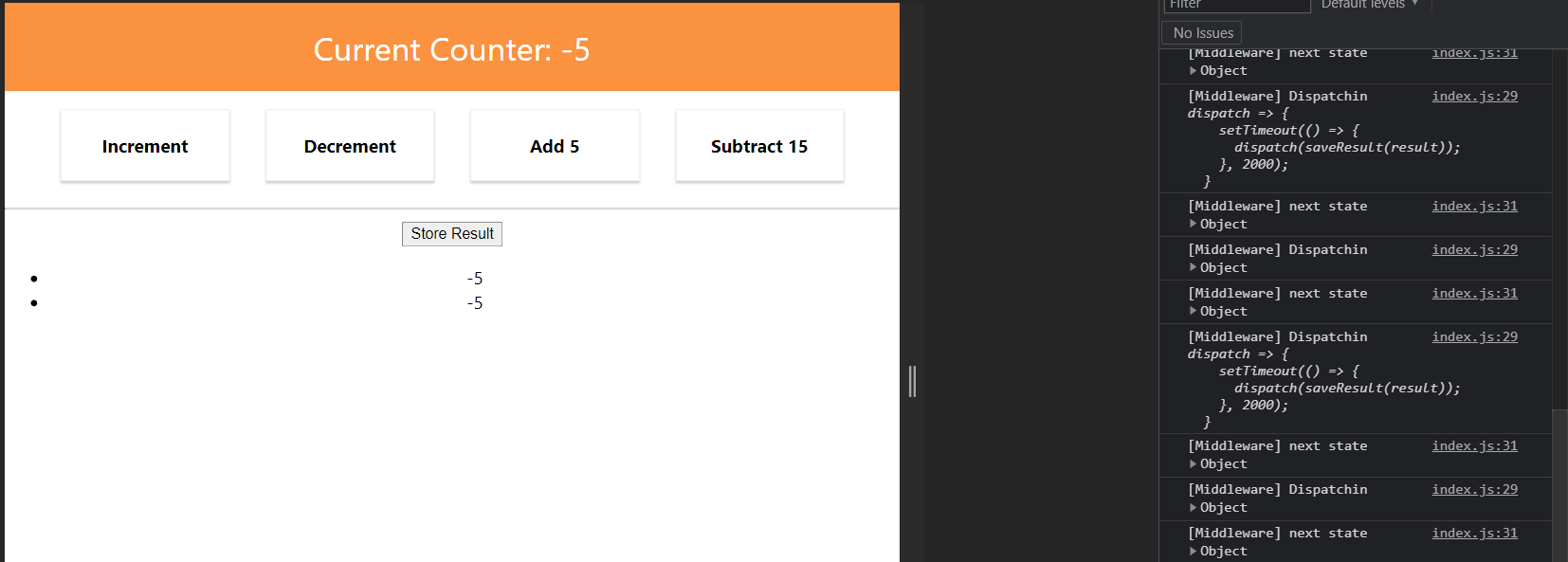
    //return{

    //    type: STORE\_RESULT,

    //    result: result

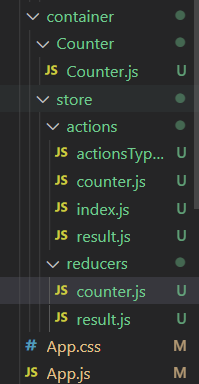
    //}

};



# RESTRUCTING ACTIONS[ it can be helpful in big and complex project]

Folder structure:



actionTypes.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';

counter.js in store

import \* as actionTypes from './actionsTypes'

//actions for asynchonrous code

export const increment = () => {

    // return actions

    return{

        type: actionTypes.INCREMENT

    }

};

export const decrement = () => {

    // return actions

    return{

        type: actionTypes.DECREMENT

    }

};

export const add = (value) => {

    // return actions

    return{

        type: actionTypes.ADD,

        value: value

    }

};

export const substract = (value) => {

    // return actions

    return{

        type: actionTypes.SUBTRACT,

        value: value

    }

};

Result.js in store

import \* as actionTypes from './actionsTypes'

//new action

export const saveResult = (result ) => {

    return{

        type:actionTypes.STORE\_RESULT,

        result:result

    };

}

export const storeResult = (result) => {

    return dispatch => {

        setTimeout(()=>{

            //Async Code

            dispatch(saveResult(result));

        },2000)

    }

};

export const deleteResult = (resElId) => {

    // return actions

    return{

        type: actionTypes.DELETE\_RESULT,

        resultElid : resElId

    }

};

Index.js in store:

export {

    add,

    substract,

    increment,

    decrement

} from './counter';

export {

    storeResult,

    deleteResult

} from './result';

Counter.js in reducer

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

result.js in reducer

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

Counter.js

import React, { Component } from 'react';

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

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

import {connect} from 'react-redux';

// import { increment,decrement,add,substract,storeResult,deleteResult } from '../store/actions/actions';

import \* as actionCreators from '../store/actions/index';

class Counter extends Component {

    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(this.props.ctr)}>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.ctr.counter,

        storeResults: state.res.results

    };

}

const mapDispatchToProps = dispatch => {

    return {

        //onIncrementCounter=properties

        onIncrementCounter: () => dispatch(actionCreators.increment()),

        //Anayamous Function

        //onIncrementCounter=properties

        onDecrementCounter: () => dispatch(actionCreators.decrement()),

         //Anayamous Function

        //onIncrementCounter=properties

        onAddCounter: () => dispatch(actionCreators.add(10)),

        //Anayamous Function

        //onIncrementCounter=properties

        onSubtractCounter: () => dispatch(actionCreators.substract(15)), //Anayamous Function

        onStoreResult: (result) => dispatch(

            actionCreators.storeResult(result)

        ),

        onDeleteResult: (id) => dispatch(

            actionCreators.deleteResult(id))

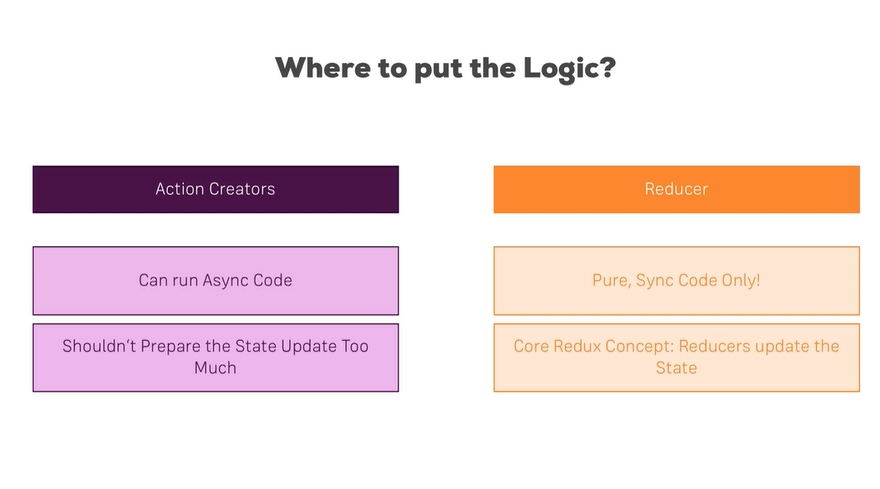
    };

}

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

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

# WHERE TO PUT DATA TRANSFORMING LOGIC



Result.js[Reducer]

import \* as actionTypes from './actionsTypes'

//new action

export const saveResult = (result ) => {

    const updatedResult = result \* 2;

    return{

        type:actionTypes.STORE\_RESULT,

        result:updatedResult

    };

}

export const storeResult = (result) => {

    return dispatch => {

        setTimeout(()=>{

            //Async Code

            dispatch(saveResult(result));

        },2000)

    }

};

export const deleteResult = (resElId) => {

    // return actions

    return{

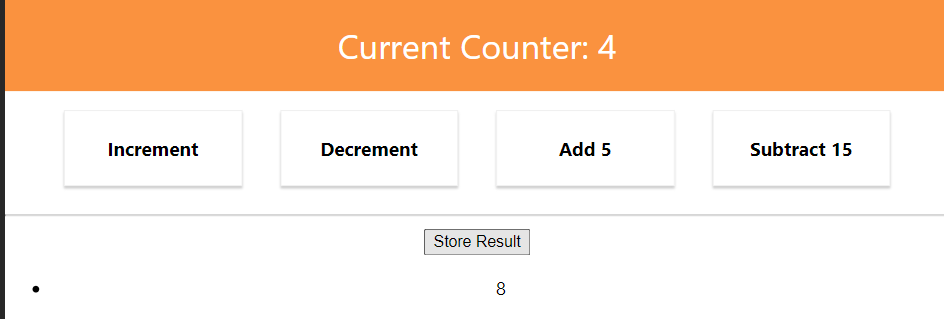
        type: actionTypes.DELETE\_RESULT,

        resultElid : resElId

    }

};

Output:



# USING ACTION CREATORS AND GET STATE

import \* as actionTypes from './actionsTypes'

//new action

export const saveResult = (result ) => {

    return{

        type:actionTypes.STORE\_RESULT,

        result:result

    };

}

export const storeResult = (result) => {

    return (dispatch,getState) => {

        setTimeout(()=>{

            const oldCounter = getState().ctr.counter;

            console.log(oldCounter);

            //Async Code

            dispatch(saveResult(result));

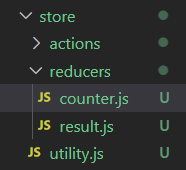
        },2000)

    }

};

# USING UTLILITY FUNCTIONS

Create Utility file:



Utilily.js

//Utility function

export const updateObject = (oldObject, updatedValues) => {

    return{

        ...oldObject,

        //JS object

        ...updatedValues

    }

};

Counter.js (reducer)

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

import {updateObject } from '../utility';

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;

            return updateObject(state, {  counter: state.counter + 1 })

        case actionTypes.DECREMENT:

            return updateObject(state, {  counter: state.counter - 1 })

            // return{

            //     ...state,

            //     counter: state.counter - 1

            // }

        case actionTypes.ADD:

            return updateObject(state, { counter: state.counter + action.value})

            // return{

            //     ...state,

            //     counter: state.counter + action.value

            // }

        case actionTypes.SUBTRACT :

            return updateObject(state, { counter: state.counter - action.value})

            // return{

            //     ...state,

            //     counter: state.counter - action.value

            // }

    }

    return state;

};

export default reducer;

# A LEANER SWITCH CASE STATEMENT [ES6 function to make reducer more understandable]

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

import {updateObject } from '../utility';

const initialState={

    counter: 0,

};

//new ES6 function to make leaner

const addNumber = (state, action) => {

    return updateObject(state, { counter: state.counter + action.value});

}

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

    switch(action.type){

        case actionTypes.INCREMENT:

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

            // newState.counter = state.counter + 1;

            // return newState;

            return updateObject(state, {  counter: state.counter + 1 })

        case actionTypes.DECREMENT:

            return updateObject(state, {  counter: state.counter - 1 })

            // return{

            //     ...state,

            //     counter: state.counter - 1

            // }

        case actionTypes.ADD:

            return addNumber(state,action);

            // return{

            //     ...state,

            //     counter: state.counter + action.value

            // }

# Visit Redux Documentation

<https://redux.js.org/introduction/getting-started>

Important Link: https://redux.js.org/usage/structuring-reducers/immutable-update-patterns