**npm i redux**

it contains methods to create store for our application

createStore and applyMiddleware and combineReducers

**npm i @redux-devtools/extension**

it contains extension to see our store in redux devtools on our browser

**npm i react-redux**

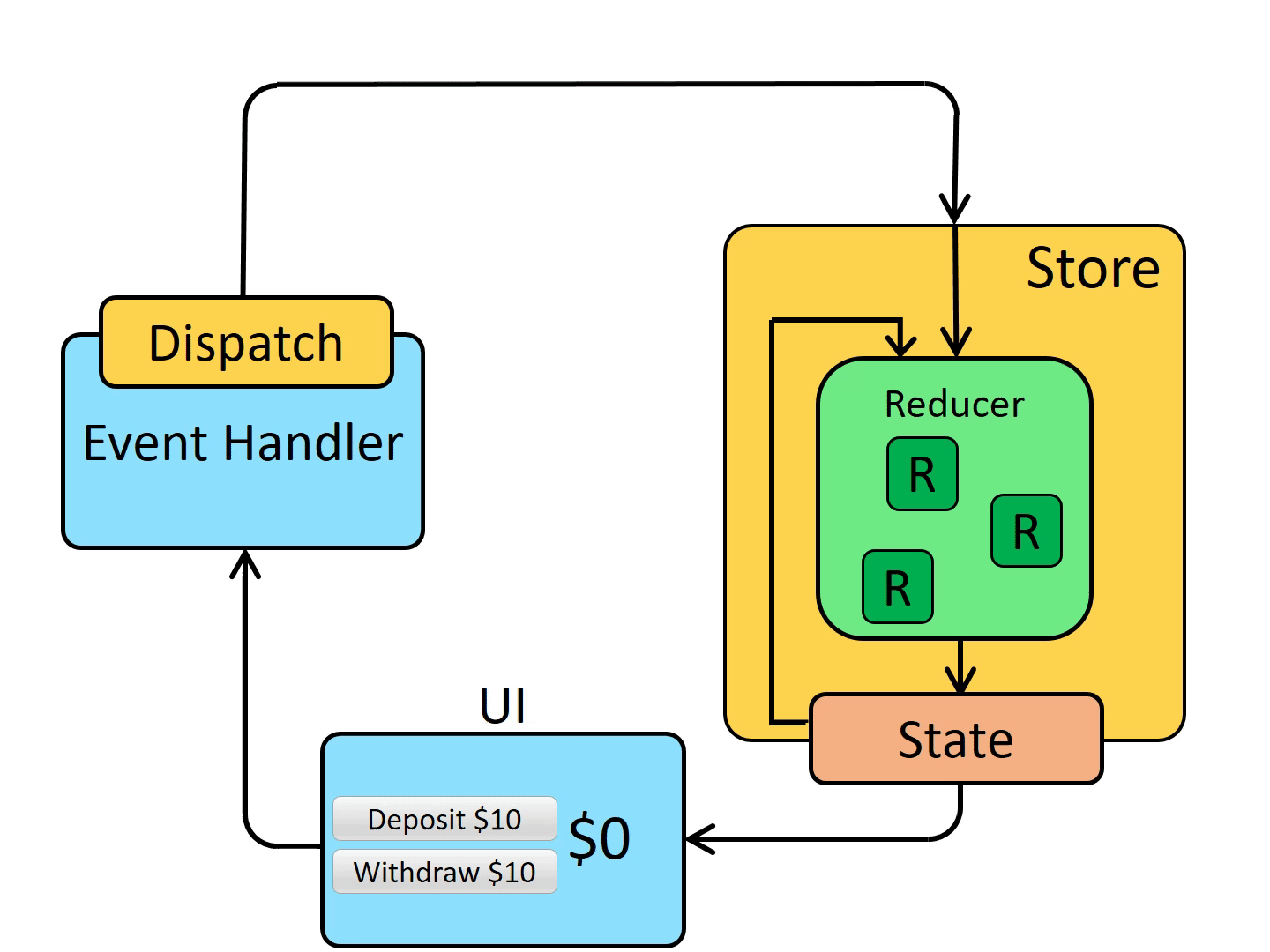
it is used to connect our react with redux store

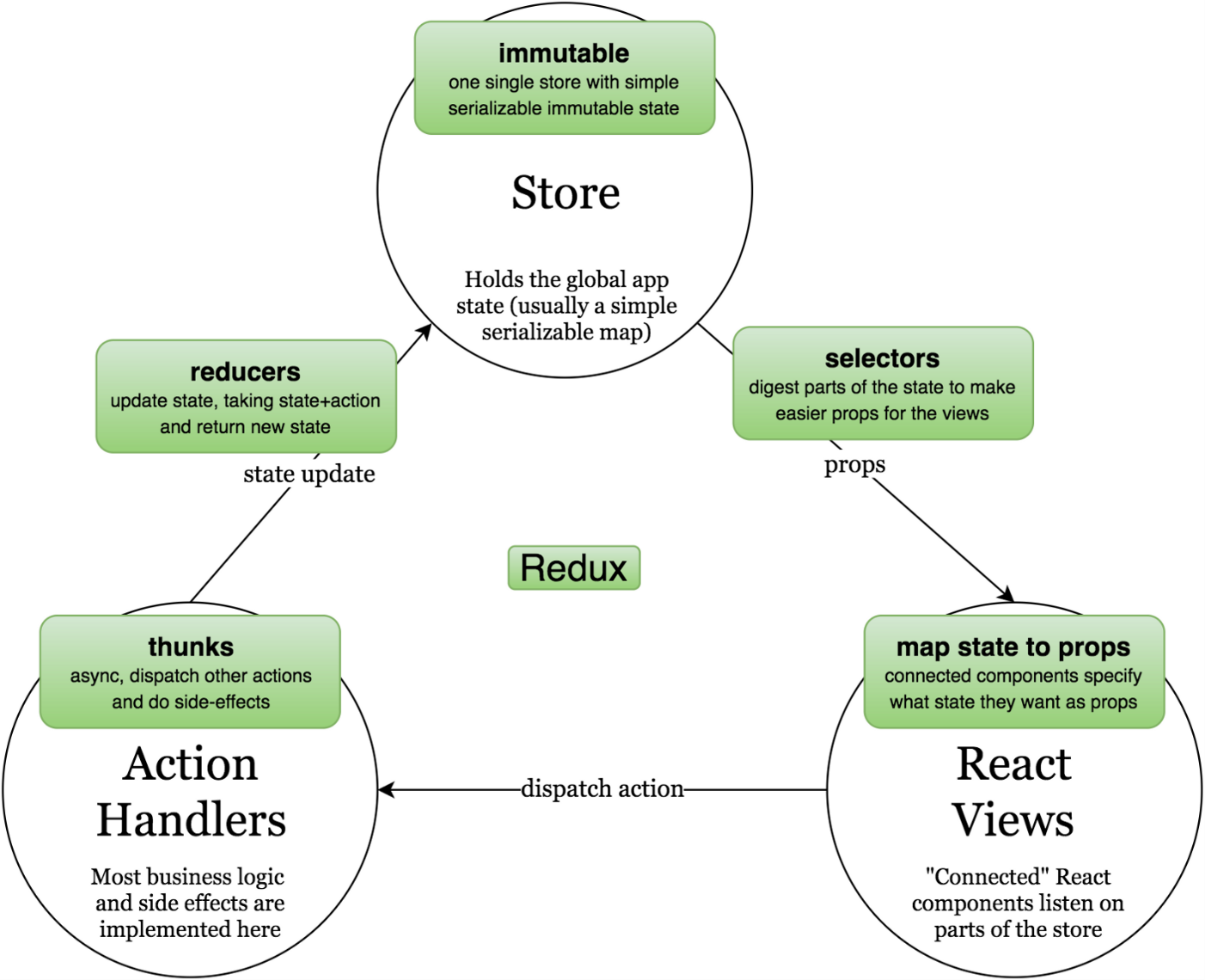
it includes <Provider /> that makes our store available to rest of our app. It has property store where we provide our store

it includes mapStateToProps() method that takes state(basically the store) and passes it our current component.

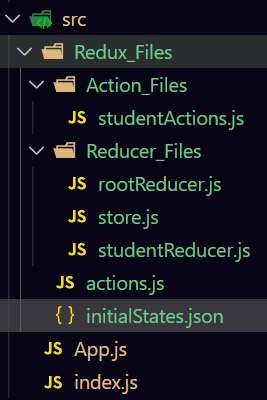
It includes mapDispatchToProps() method that allows us to use the working methods written in action files.

It includes connect() method which is used to connect mapStateToProps() amd mapDispatchToProps() with our component(which is going to access our store data)

****



**Folder Structure**

****

***initialStates*.js**

{

  "students": []

}

***action*.*js***

export const ADD\_STUDENT = "ADD\_STUDENT"

export const DELETE\_STUDENT = "DELETE\_STUDENT"

***studentActions.js***

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

// add bats Action function

export const addStudentActionFunction = (newStudent) => {

  return {

    type: actionTypes.ADD\_STUDENT,

    payload: { newStudent: newStudent },

  };

};

//

//

//

//

// delete bat Action function

export const deleteStudentActionFunction = (studentName) => {

  return {

    type: actionTypes.DELETE\_STUDENT,

    payload: { studentName: studentName },

  };

};

***studentReducer*.*js***

import initialStates from "../initialStates.json";

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

const studentReducer = (state = initialStates, actionState) => {

  switch (actionState.type) {

    case actionTypes.ADD\_STUDENT:

      return {

        ...state,

        students: [...state.students, actionState.payload.newStudent],

      };

    case actionTypes.DELETE\_STUDENT:

      return {

        ...state,

        students: state.students.filter(

          (studentName) => studentName !== actionState.payload.studentName

        ),

      };

    default:

      return state;

  }

};

export default studentReducer;

***rootReducer.js***

import { combineReducers } from "redux";

import studentReducer from "./studentReducer";

const rootReducer = combineReducers({

  studentsNames: studentReducer,

});

export default rootReducer;

***store.js***

import { createStore, applyMiddleware } from "redux";

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

import rootReducer from "./rootReducer";

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

export default store;

***index.js***

import React from "react";

import ReactDOM from "react-dom";

import { Provider } from "react-redux";

import store from "./Redux\_Files/Reducer\_Files/store";

import App from "./App";

ReactDOM.render(

  <Provider store={store}>

    <App></App>

  </Provider>,

  document.querySelector("#root")

);

***App.js***

import React, { useState } from "react";

import { connect } from "react-redux";

import {

  addStudentActionFunction,

  deleteStudentActionFunction,

} from "./Redux\_Files/Action\_Files/studentActions";

function App({ students, addStudentFunc, deleteStudentFunc }) {

  const [name, setName] = useState("");

  return (

    <>

      <div>

        Total Number of Bats available :{" "}

        {students.students.map((name) => (

          <div>{name}</div>

        ))}

      </div>

      <br></br>

      <br></br>

      <br></br>

      <input

        type="text"

        value={name}

        onChange={(e) => {

          setName(e.target.value);

        }}

      ></input>

      <button

        onClick={() => {

          addStudentFunc(name);

          setName("");

        }}

      >

        Add Student

      </button>

      <br></br>

      <br></br>

      <button

        onClick={() => {

          deleteStudentFunc(name);

          setName("");

        }}

      >

        Remove Student

      </button>

    </>

  );

}

const mapStateToProps = (state) => {

  return {

    students: state.studentsNames,

  };

};

const mapDispatchToProps = (dispatch) => {

  return {

    addStudentFunc: (newStudent) =>

      dispatch(addStudentActionFunction(newStudent)),

    deleteStudentFunc: (studentName) =>

      dispatch(deleteStudentActionFunction(studentName)),

  };

};

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

***Thunk***

**Reducer functions are pure function that is we do not perform any side effect work here**

**Because this is the only file that makes changes to our state synchronously while using api or any other side effect work is asynchronous works so using both at same place can change our state values in wrong way.**

**So we do all our asynchronous work in action creator function files using thunk**

**file structure:-**

**actions.js**

export const FETCH\_REQ = 'FETCH\_REQ';

export const FETCH\_SUC = 'FETCH\_SUC';

export const FETCH\_FAIL = 'FETCH\_FAIL';

**userReducer.js**

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

const initialState = {

  loading: false,

  error: "",

  users: [],

};

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

  switch (action.type) {

    case actionTypes.FETCH\_REQ:

      return {

        ...state,

        loading: true,

      };

    case actionTypes.FETCH\_SUC:

      return {

        ...state,

        loading: false,

        users: [...action.payload],

      };

    case actionTypes.FETCH\_FAIL:

      return {

        ...state,

        loading: false,

        error: action.payload,

      };

    default:

      return state;

  }

};

export default userReducer;

**userAction.js**

import axios from "axios";

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

const fetchReq = () => {

  return {

    type: actionTypes.FETCH\_REQ,

  };

};

const fetchSuc = (users) => {

  return {

    type: actionTypes.FETCH\_SUC,

    payload: users,

  };

};

const fetchFail = (err) => {

  return {

    type: actionTypes.FETCH\_FAIL,

    payload: err.message,

  };

};

export const fetchUsers = () => {

  return async (dispatch) => {

    dispatch(fetchReq());

    try {

      let res = await axios.get("https://jsonplaceholder.typicode.com/users");

      let data = res.data;

      dispatch(fetchSuc(data));

    } catch (e) {

      dispatch(fetchFail(e));

    }

  };

};

**rootReducer.js**

import { combineReducers } from "redux";

import userReducer from "./userReducer";

const rootReducer = combineReducers({

  user: userReducer,

});

export default rootReducer;

**store.js**

import { createStore, applyMiddleware } from "redux";

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

import thunk from "redux-thunk";

import rootReducer from "./rootReducer";

const store = createStore(

  rootReducer,

  composeWithDevTools(applyMiddleware(thunk))

);

export default store;

**App.js**

import React from "react";

import { Provider } from "react-redux";

import store from "./redux/store";

import Users from "./Components/Users";

function App() {

  return (

    <>

      <Provider store={store}>

        <Users></Users>

      </Provider>

    </>

  );

}

export default App;

**Users.js**

import React, { useState, useEffect } from "react";

import { connect } from "react-redux";

import { fetchUsers } from "../redux/userAction";

function Users({ userData, fetchUsers }) {

  useEffect(() => {

    fetchUsers();

  }, []);

  return (

    <>

      {userData.loading ? (

        <h1>Loading...</h1>

      ) : userData.error !== "" ? (

        <h1>{userData.error.message}</h1>

      ) : (

        <ul>

          {userData.users.length > 0 &&

            userData.users.map((user) => {

              return <li>{user.name}</li>;

            })}

        </ul>

      )}

    </>

  );

}

const mapStateToProps = (state) => {

  return {

    userData: state.user,

  };

};

const mapDispatchToProps = (dispatch) => {

  return {

    fetchUsers: () => dispatch(fetchUsers()),

  };

};

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