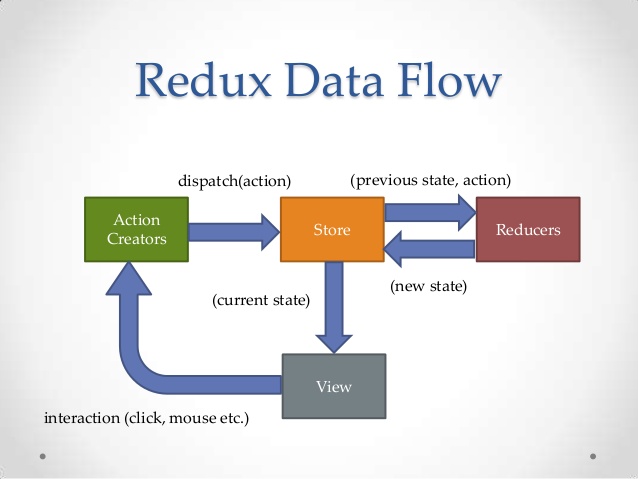
**REDUX**



*Npm install –save redux react-redux redux-thunk*

Redux

Redux is a predictable state container which contains the state of the application

React-redux

Redux can be used with any JS library. React-redux is the official redux UI binding library for React.

React-thunk

Redux thunk middleware allows you to write creators that return a function instead of an action

**The 3 Principles**

Single source of truth

The state of your whole application is stored in an object tree within a single store. A store is an object that holds the applications state tree.

State is read only

The only way to change the state is to emit an action, an object describing what happened. An action is a plain object that represents an intention to change the state. Actions are the only way to get data in to the store

Changes are made with pure functions

To specify how the state tree is transformed by actions, you write pure reducers. A reducer is a function that accepts an accumulation and a value and returns a new accumulation. They are used to reduce a collection of values down to a single value.

**Example**

*app.js*

import React from “react”;

import store from “./store”

import { Provider } from “react-redux”; // a component which acts as the glue between redux and react

class App extends React.Component {

render() {

return (

<Provider store={store}>

// other JSX here

</Provider>

);

}

}

*Store.js*

Import { createStore, applyMiddleware } from “redux”;

Import thunk from “redux-thunk”;

Import rootReducer from “./reducers”;

Const initialState = {};

Const middleware = [thunk];

Const store = createStore(rootReducer, initialState, applyMiddleware(…middlware));

Export default Store;

*Reducers > index.js (root reducer)*

Import { combineReducers } from “redux”;

// import reducers here – e.g. import postReducer from “./postReducer”;

Export default combineReducers({

Posts: postReducer

});

*Reducers > postReducer.js*

Import { FETCH\_POSTS, NEW\_POST } from “../actions/types”;

Const initialState = {

Items: [],

Item: {}

}

Export default function(initialState, action) {

Switch (action.type) {

Case FETCH\_POSTS:

Return {

…state,

Items: action.payload

}

Default:

Return state;

}

}

Actions > types.js

Export const FETCH\_POSTS = ‘FETCH\_POSTS’;

Export const NEW\_POST = ‘NEW\_POST’;

*Actions > postAction.js*

Import { FETCH\_POSTS, NEW\_POST } from “./types”;

Export function fetchPosts() {

Return function (dispatch) {

Fetch (url)

.then(res => res.json())

.then(data => dispatch({

Type: FETCH\_POSTS,

Payload: data

}));

}

}

}

*posts.js*

import React from “react”;

import { connect } from “react-redux”;

import { fetchPosts } from “../actions/postActions”;

class Posts extends React.component {

componentDidMount() {

this.props.fetchPosts();

}

render() {

const postItems = this.props.posts.map(post => (

<div key={post.id}>

<h3>{post.title}</h3>

<p>{post.body}</p>

</div>

));

return(

<div>

{postItems}

</div>

);

}

}

Const mapStateToProps = state => ({

Posts: state.posts.items; // posts as this is what its called in root reducer

})

Export default connect(mapStateToProps, { fetchPosts } )(Posts);

*\*\* connect() connects a react component to the redux store. It can take up to 4 optional*

*arguments: mapStateToProps, mapDispatchToProps, mergeProps, options \*\**

**Redux dev tools (chrome extension)**

Set this extension up to view helpful redux actions / debugging.