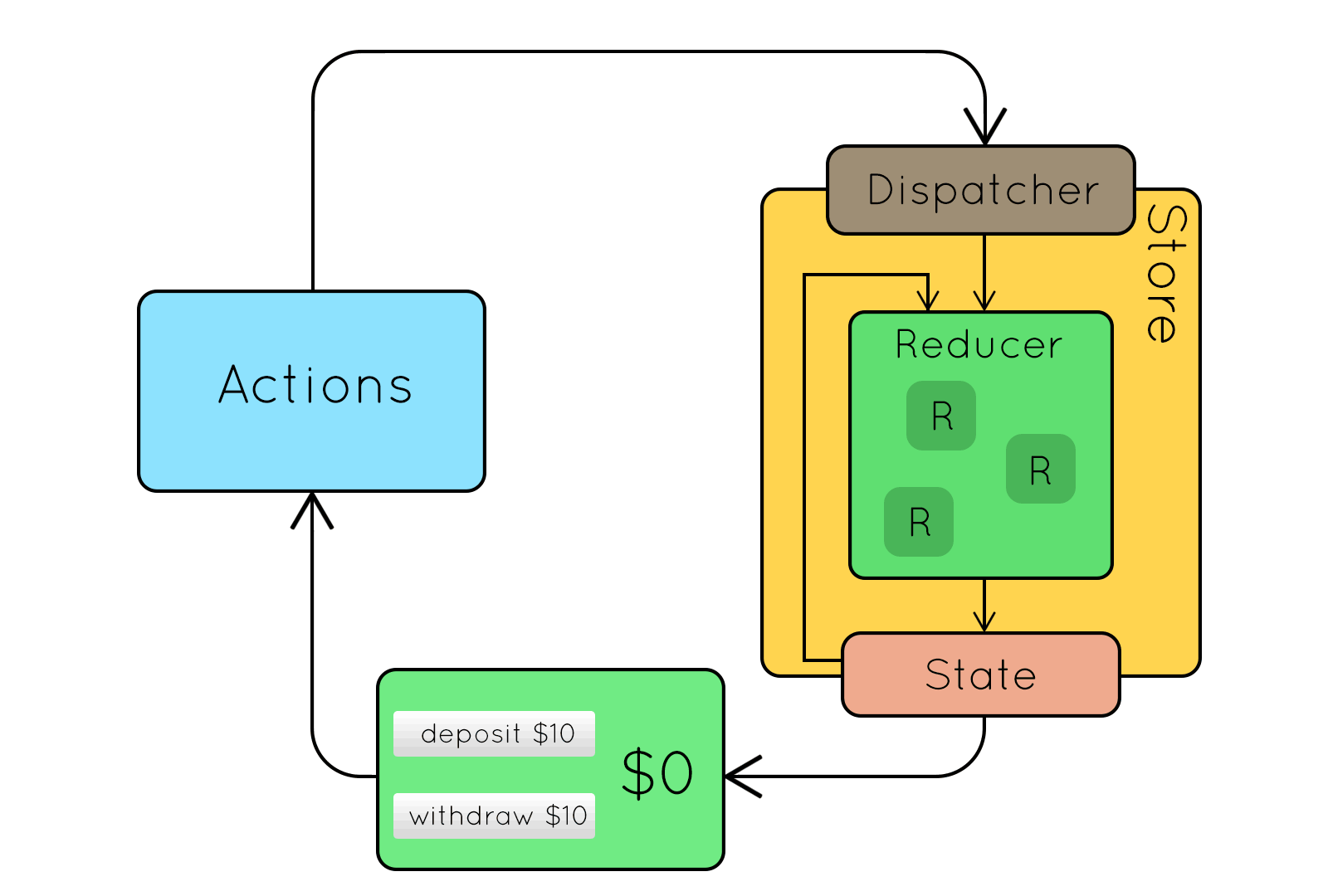
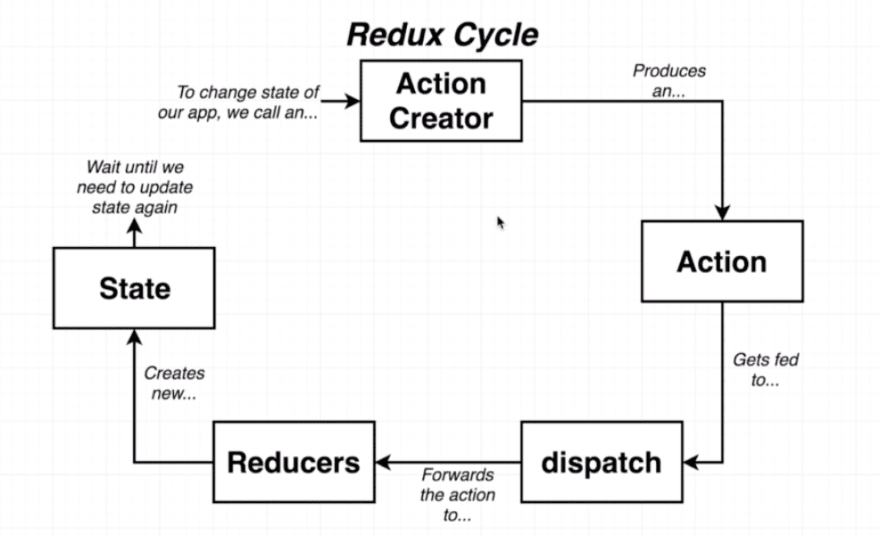
REMINDER OF REDUX FLOW





We will tackle this flow below:

Action Creator > Action > Dispatch > Reducers > Redux Store

I always start to create the boiler plate for redux as follows:

1. Install the library and create the folders for redux
2. Create the store and import our future reducers to it.
3. Hooking up our App to get access to the redux store.

Then when I need to store something in the redux store I do the

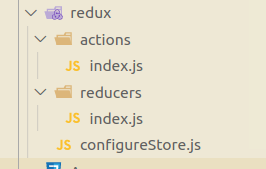
following:

1. Create an Action Creator.
2. Create an Action which gets passed the value from the Action Creator.
3. Create a reducer that will handle that action.

**Lets create a simple app that will get a joke from the chuck norris api and store it to redux state.**

1. **Install the redux library and create the folders**

* npm install redux react-redux
* Create a redux folder.
* Create a configureStore.js
* Create a actions and reducers folder with an index.js



Action Creator > Action > Dispatch > Reducers > Redux Store

1. **Redux Store:**

Create the store and attach our future reducers to it.

* configureStore.js

import { createStore } from "redux";

import reducers from "../redux/reducers/index";

export **let** store = createStore(reducers);

* Create a store and attach all our future reducers to it.
* Export our store which we will import in our index.js.

1. **Hooking our store to our React Components.**

Wrap our store to the highest component of our react application so all its children can access the store. Which is naturally our App component which lives in index.js

* index.js

import React from "react";

import ReactDOM from "react-dom";

import { Provider } from "react-redux";

import { store } from "./redux/configureStore";

import App from "./App";

ReactDOM.render(

<React.StrictMode>

<App>

<Provider store={store}>

<App />

</Provider>

</React.StrictMode>,

document.getElementById("root")

);

* The *Provider* is a component that has a reference to the *Store* and provides the data from the *Store* to the component it wraps.
* Give our store as a prop to the Provider component
* Wrap the Provider component to our App component so all components under App can access the store.

Action Creator > Action > Dispatch > Reducers > Redux Store

1. **Reducers:**

* reducers/index.js

import { combineReducers } from "redux";

*//we will import our future reducers here*

export default combineReducers({

//and we will export them here

});

* Import combineReducers from redux. This will combine all the reducers into one reducer.
* Export our future reducers to our store.

We took care of our setup. Wasn’t that bad, was it?

Lets now fetch a Chuck Norris joke in our App.js, then create an action creator that will take the joke value and pass it to our action which in turn will dispatch the value to our selected reducer. Finally, in our Joke component we will fetch the joke from our store and display it in our JSX.

Fetch a joke from our App component

**import React from "react";**

**import axios from "axios";**

**import Joke from "./components/joke";**

**class** App **extends** React.Component {

**async** componentDidMount() {

**const** data = await axios.get("https://api.chucknorris.io/jokes/random");

}

render() {

return (

<div>

<Joke></Joke>

</div>

);

}

}

export default App;

We will now need to wrap our App component to the *connection* HOC component that we get from redux to have access to all the redux goodies.

Importing the connect HOC:

import { connect } from "react-redux";

Wrapping the connect HOC to our App component

export default connect()(App);

connect gets two arguments. First one is to access our store and the 2nd one is to send our action creator to our dispatch.

Since we don’t need to access our store at this given time, we only want to dispatch an action, we can leave it as null. By convention we call our 2nd argument as “mapStateToDispatch”.

export default connect(null, mapStateToDispatch)(App);

Action Creator > Action > Dispatch > Reducers > Redux Store

1. **Action Creator:**

A function that creates the action

Now we create our mapStateToDispatch object which will create our Action Creator

**const** mapStateToDispatch = {

chuckNorrisJoke: (joke) **=>** fetchJoke(joke),

};

We created an **action creator** called *chuckNorrisJoke* which we get access via props.

The value of chuckNorrisJoke is the action which we will soon create called *fetchJoke.*

The action also gets one argument, a joke. We will pass the joke we get from the API to the Action eventually.

This is how our component looks now

**class** App **extends** React.Component {

**async** componentDidMount() {

**const** fetchData = await axios.get("https://api.chucknorris.io/jokes/random");

this.props.chuckNorrisJoke(fetchData.data.value);

}

render() {

return (

<div>

<Joke></Joke>

</div>

);

}

}

**const** mapStateToDispatch = {

chuckNorrisJoke: (joke) **=>** fetchJoke(joke),

};

export default connect(null, mapStateToDispatch)(App);

After we got our Joke from the API, we will pass the joke to our Action Creator which in turn will pass it to the Action.

this.props.chuckNorrisJoke(fetchData.data.value);

We need to import our Action called fetchJoke which we will soon create in our actions folder

import { fetchJoke } from "./redux/actions/fetchJoke";

So the overview of our component and how everything is hooked up as follows:

import React from "react";

import axios from "axios";

import { connect } from "react-redux";

import { fetchJoke } from "./redux/actions/fetchJoke";

import Joke from "./components/joke";

**class** App **extends** React.Component {

**async** componentDidMount() {

**const** fetchData = await axios.get("https://api.chucknorris.io/jokes/random");

this.props.chuckNorrisJoke(fetchData.data.value);

}

render() {

return (

<div>

<Joke></Joke>

</div>

);

}

}

**const** mapStateToDispatch = {

chuckNorrisjoke: (joke) **=>** fetchJoke(joke),

};

export default connect(null, mapStateToDispatch)(App);

Action Creator > Action > Dispatch > Reducers > Redux Store

1. **Action:**

An object that contains information about how we want to change some data within our central state

Let’s create a new Action in the action folder called fetchJoke and export default it.

We can safely remove the index.js from the action folder. We want to think our application will have dozens of actions, so to keep things organized, we create each action to a separate file.

fetchJoke.js

export **const** fetchJoke = (joke) **=>** {

return {

type: "FETCH\_JOKE",

payload: joke,

};

};

We create a type called “FETCH JOKE” which our dispatch will look for the reducer for the same type.

Our payload will be the joke we passed from our Action Creator.

Action Creator > Action > Dispatch > Reducers > Redux Store

**Dispatch:**

A function that takes in an action,makes copies of the action, and sends them to our reducers.

Good news is we don’t have to send them to our reducers. It will automatically create copies of our Action and pass them to all of our reducers to find a match.

Let's create our Reducer.

As we did with our actions, let's modify our reducer file structure a little bit.

* Let’s change our reducers/index.js to reducers/rootReducer
* Let's create a reducer called fetchJokereducer.js

fetchJokeReducer.js

**const** fetchJokeReducer = (state = “”, action) **=>** {

if (action.type === "FETCH\_MOVIES") {

return action.payload;

}

return state;

};

export default fetchJokeReducer;

Reducers get two arguments. The state which has by default an initial state, an empty string/boolean/array and the action which is passed from our dispatch automatically.

We check if the action.type from our action is “FETCH\_JOKE” then we are in the right reducer.

Then return the payload which is our joke in this case.

If it is not the correct reducer it will return the reducers state. Initially it is an empty string but there can be data inside, depending if this reducer saved any state previously.

We then import this to our rooteReducer like so:

rootreducer.js

import { combineReducers } from "redux";

import fetchJokeReducer from "./fetchJokeReducer";

export default

combineReducers({

joke: fetchJokeReducer,

Now we have our joke inside the redux store!

Next thing to do is actually use that value in our Joke component.

We import our connect HOC and wrap it in our Joke component.

export default connect()(Joke);

We need to access our store so we need our first argument in the connect HOC. We can leave the second argument empty or null.

By convention the first argument will be called mapStateToProps

export default connect(mapStateToProps)(Joke);

mapStateToProps function can take two parameters, the state of the store and the components own props.

//not understanble

We will only be using the store’s state. We then create an object and set a key and fetch for a particular state we want from the store as the value

const mapStateToProps = (state) => ({

joke: state.joke,

});

We then get access to this state via props and we can inject this to our JSX

return <div>{this.props.joke}</div>;

Our joke component looks like this and how it connects to our store:

Joke.js

import react from "react";

import { connect } from "react-redux";

**class** Joke **extends** react.Component {

render() {

return <div>{this.props.joke}</div>;

}

}

**const** mapStateToProps = (state) **=>** ({

joke: state.joke,

});

export default connect(mapStateToProps)(Joke);

**CONCLUSION**

Action Creator > Action > Dispatch > Reducers > Redux Store

Action Creator

Creating the action creator

**const** mapStateToDispatch = {

chuckNorrisJoke: (joke) **=>** fetchJoke(joke),

};

Calling the action creator

this.props.chuckNorrisJoke(fetchData.data.value);

**Action :**

Assigning the action creator to an action

**const** mapStateToDispatch = {

chuckNorrisJoke: (joke) **=>** fetchJoke(joke),

};

Creating the action

export **const** fetchJoke = (joke) **=>** {

return {

type: "FETCH\_JOKE",

payload: joke,

};

};

**Dispatch:**

MAGIC!!!

**Reducers:**

**const** fetchJokeReducer = (state = [], action) **=>** {

if (action.type === "FETCH\_JOKE") {

return action.payload;

}

return state;

};

export default fetchJokeReducer;

Redux Store

import { createStore } from "redux";

import reducers from "../redux/reducers/index";

export **let** store = createStore(reducers);

**EXTENSIONS**

* Apply extensions into configStore.js

.

REDUX-DEVTOOLS-EXTENSION

* Install google chromes redux devtools extension [here](https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd?hl=en)
* npm install redux-devtools-extension

import { createStore } from "redux";

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

import reducers from "../redux/reducers/rootReducer";

export **let** store = createStore(reducers, composeWithDevTools());

* Import redux dev tools
* Apply the composeWithSDevTools to our store.