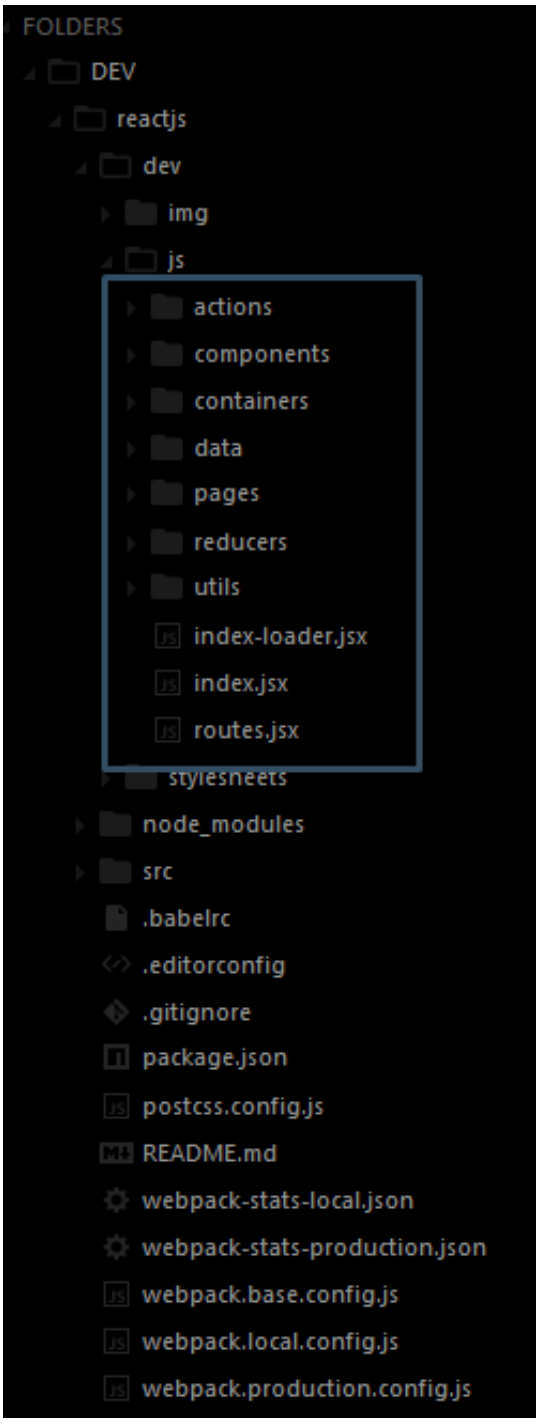
Redux Example Workflow

example of folder structure:



SETUP:

Creating a `State` Object through

`combineReducers()`

- first we need combine the reducers in `reducers/index.jsx` to create `rootReducer` for `App.jsx`
- load a reducer from individual reducer file `import yourReducer from './yourReducer'`
- put that reducer into the `combineReducers` method.
- **NOTE:** each reducer will be an `Object` in the state, the state of your application may want to be thought out before starting serious development. This means when a reducer method is dispatched via an action…. that reducer will ONLY affect the `Object` it was defined in, in the `combineReducers` method.

```
1.  import {combineReducers} from 'redux';
2.  import {routerReducer} from 'react-router-redux';
3.  import htmlClassesReducer from './htmlClassesReducer'
4.
5.  /*
6.      We combine all reducers into a single object before updated d
    ata is dispatched (sent) to store
7.      Your entire applications state (store) is just whatever gets
    returned from all your reducers
8.  */
9.  const allReducers = combineReducers({
10.     htmlClasses: htmlClassesReducer,
11.     routing: routerReducer,
12. });
13.
14. export default allReducers
```

Now the `state` has a new key *(ie: `htmlClasses`)* which is an object that should only be manipulated by its `reducers` which are triggered by `actions`

Make the `state` object and `actions` globally accessible.

- open `container/App.jsx`
- Namespace how you'd like to access that data:

```
// these values become available to `this.props` in components.
const mapStateToProps = state => {
    return {
        redux: {
            htmlClasses: state.htmlClasses
        }
    }
}
```

The above will make the `state` available anywhere in the App like so:

`this.props.redux.htmlClasses`

- we need to import the `actions` and then do the same as above, but we are mapping `actions` this time.
    - first, import your action files at the top of `App.jsx`. Each action in the actions file will have `export const` prepended to it, so we need to import **all** of the methods individually in the file, eg:
        - `import * as htmlClassesActions from 'actions/htmlClassesActions'`

- **Next**, we need to map the `actions` to `props` so that the actions are globally accessible:
    - *NOTE:* by convention, this variable is named `mapDispatchToProps` but `mapActionsToProps` is kind of what it *means*… or is syntactically easier to understand/read. `Actions` are a **noun** and actions get `Dispatched` – the verb here.

```
const mapDispatchToProps = dispatch => {
    return {
        actions: {
            ...bindActionCreators({
                ...testActions,
                ...headerActions,
                ...htmlClassesActions
            }, dispatch)
        }
    }
}
```

The above will make the `actions` available anywhere in the App like so:

`this.props.actions.yourActionMethod`

Define the `defaultState` for the object in the `state`

- the value will return `undefined` if not defined in the `defaultState` and used in a React component.
- to create a default state for the object open the file `Index.jsx` and define the initial state there:

```
const defaultState = {
    htmlClasses: {
        body: "notLoaded"
    },
    // anotherStateObject: {},
    // yetAnother: {},
}
```

First

1) First, an **action** needs to *get **dispatched***–this is simply when an event takes place, when a user performs an *action* for example.

- happens in the `jsx` files, ie: *(components, containers, pages)*
- example: When a user navigates to a “page”, change the state, so that the `<body>` tag adds a special className for the tag. We will add a state object `htmlClasses`. We will trigger actions that will let us add/remove classes to any element in the DOM. `htmlClasses` will be namespaced and elements can be added/removed on the fly of course.
    - we will setup the state like this:

```
state = {
    htmlClasses: {
        body: 'default-classname',
        header: 'header',
        sidebar: 'secondary-stuff'
    }
}
```

EXAMPLE ACTION:

```
onComponentDidMount(){
    this.props.actions.addClassName();
}
```

Second

2) Create the action file. (`htmlClassesActions.js`) Example:

```
export const ADD_CLASS_NAME = "ADD_CLASS_NAME"

export const addClassName = (key, classToAdd) => {
    return {
        type: ADD_CLASS_NAME,
        key,
        classToAdd
    }
}
```

Third

2) Create the reducer file(`htmlClassesReducer.js`). Example:

```
import * as actions from 'actions/htmlClassesActions';

const addClassName = (state , action) => {
    return {
        ...state,
        [action.key]: action.classToAdd
    }
}

const runReducer = ( state={}, action={} ) => {
    switch (action.type) {
        case actions.ADD_CLASS_NAME:
            return addClassName(state, action)
        default:
            return state
    }
}
export default runReducer
```

So now we can fire use our action to update the state:
Example:

```
// onClick...
this.props.actions.addClassName( "body", "intro-loaded");
```

and we need to place the `state` *somewhere so that when the action is dispatched, the value is updated*
Find the `<body>` tag–or a different app *wrapper* className of your app and place in `this.props.redux.htmlClasses.body` ..
…

```
<div className={"app-wrapper" + " " + this.props.redux.htmlClasses.body}>
```

GOOD TO GO!