1 Introdution

This file is a set of notes that I write as I go through my Journey with Javascript/React/Redux and anything that comes along the way. I will try my best to have some organized structured but since this is mostly for my education I wont have it as my first priority. Instead this as notes from a person that has some experience with html,css,JavaScript and is only looking to get better at it.

BIG P.S. Do NOT assume this is free of typos or general mistakes. Please tread with care.

2 Le Hello Worlds(without the "hello" or "world")

Seems like the very basics starts with a classic html file which imports the `React` and teh `ReactDOM` libraries using `script` html tags

```
<html>
<head>
<script src="https://unpkg.com/react@16/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
<!---...more stuff here--->
</head>
<body>
    <div id="root">
    <!--- Our React App will live here -->
    </div>
    <script type="text/babel">
    ReactDOM.render(
        <OurRootComponent />,//could also be a javascript component variable
        document.getElementById("root")//the root div above
    )
    <!--- React Code Goes here --->
    </script>
</body>
</html>
```

2.1 What is DOM?

When a we page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page. This model allows javescript to create dynamic HMTL

DOM IS STANDARD that is established by W3C(World Wide Web Consortium).

2.2 What is Babel?

This is a Javscript compiler(transpiler ?) that includes the ability to compile. JSX into vanilla JavaScript.

2.3 What is JSX ?

JSX is a syntax extension to JavaScript. It is used with React to describe what the UI should look like.

JSX produces React "elements".

2.3.1 Embedding Expressions in JSX:

We can embed variables inside of JSX by wrapping it in curly braces:

```
const name = 'My Name';
const element = <h1> Hello, {name}</h1>

ReacDOM.render(
element,
document.getElementById('root)
)
```

What else can be embedded ?

```
Any valid JavaScript *expression*. That means any arithmetic expressions
or even function returns.
```

Where else can we place JSX?

```
It seems to work wherever a normal JavasScript funciton calls works.
They end up being evalueated as JavaSript objects.
```

Can it prevent Injection Attacks?

```
It seems so, since React DOM excapes any value embedded in
JSX before rendering them. Preventing XSS
```

2.4 Components in React

Most of our logic in react will be based on *components*. These can be created by extending upon the `React-Component` class:

```
class MyComponent extends React.Component {
    render () {
        return <h1>Hello World!</h1>
        //The above statement is returning a React Component(whose syntax
        //  I believe to be JSX)
    }
}
```

Most(if not all) the components in React Js will use the `render` method to return a description that details how our component should be rendered in the UI.

2.5 Data in React:

React has two main types of data:

1. **props**: Props are public, and are not controlled by the React component to which it corresponds. IT is passed down from parent components doing all the data controlling.
2. **state**: this bois is private and can be changed only from within the React Component

2.5.1 Props

So basically, the main purpose of props is to allow for the reusability that stands as one of React's pillar principles.

Passing props to a component is as easy as :

```
ReactDOM.render(
    <MyComponent aProperty="I am a property" />,
    document.getElementById("root")
);
```

As you can see above JSX allows us to specify `props` similarly to html attributes. This prop above can be accessed inside the component as follows:

```
class MyComponent extends React.Component{
    render () {
        return <h1> One of my properties is : {this.props.aProperty}</h1>;
    }
}
```

2.5.2 State

Can be initialized inside the components constructor. **State has only one key called `message`**

```
class Hello extends React.Component {
    constructor() {
        super();//super has to be called for $t$ his` object to be initialized
        this.state = {
        message: "This is the initialization of our state"
        }
    }
    render() {
        return <h1> Dis is my state:{this.state.message};
```

```
    }
}

To change a state we simply calll `this.setState()`, where we pass
the new state *object* as the argument.


## Event Handlers

render () { return (

    <h1> Press button below to add {this.state.name} to your cart</h1>
    <button onClick={this.addToCart}>Add to Cart</button>
    </div>
)

}

## Addendum

So far it seems like im missing on what binding does and why its needed
to make the `this` object available to methods. Maybe it somewhat "private-izes" the method
thinking too much

# REACTDom

`react-dom` package provides DOM-specific methods that can be used at the
top level of your app and as an "escape hatch" to get outside of the
React model. (Though this is not commonly needed for most components)

So called DOM-specific methods:

1. [`ReactDOM.render(element, container[, callback])`](https://reactjs.org/docs/react-dom.ht

    * This renders a *React element* into the *DOM* int he supplied
    container and returns a *references* to the component.
    * If *React element* has been previously rendered than it will update it.
    * If (optional) `callback` is provided it will be called after
        *React element* has been rendered or updated

2. [`ReacDOM.hydate(element, container[,callback])`](https://reactjs.org/docs/react-dom.html

    * Ah I cant care for this right now.
    * There are more methods. Up for you to search

# What are Redux Reducers for?

Important about them:
```

```
* They should be designed to make them a *single source of Truth*.

    (i.e. stored in a single place so that if we need to access that stat
    anywhere else in in our app then we can do so by a single
    reference to this source of truth)
* Every state can have their own **reducer**.
* To alleviate a mess we can combine them in one **root reduce**, which
    ends up defining the store
*

# Implementing Actions

Once you have designed your reducers you can jump into
implementing actions. *Actions will make sure we make correct calls
to our API*

Actions seem to *serve* API calls. Ideally it does so through
three defined stages:

1. Loading State
2. Success
3. Failure

They are supposed to provide assurance of correct data flow from the API.

Once of the examples:
```

export function fetchSearchData(args) { return async (dispatch) => { // Initiate loading state dispatch({ type: FETCH_SEARCH_DATA }); try { // Call the API const result = await fetchSearchData(args.pageCount, args.itemsPerPage);

// Update payload in reducer on success dispatch({ type: FETCH_SEARCH_SUCCESS, payload: result, currentPage: args.pageCount }); } catch (err) { // Update error in reducer on failure dispatch({ type: FETCH_SEARCH_FAILURE, error: err }); } }; }

```
# Integrating React with Redux(React-Redux)

`react-redux` is its package

## Integration

`react-redux`'s `provider` class: It helps letting allowing the app
access all of the states provided by it
```

// src/index.js // // ... import React from 'react'; import ReactDOM from 'react-dom'; import { Provider } from 'react-redux'; const App =

React-Redux eCommerce app

; ReactDOM.render({ App } , document.getElementById(‘root’));

```
# `Connect` Component

THe `Connect` Component allows us to retrieve data or change it
by dispatching an action

## Example
```

import React, { Component } from ‘react’; import { connect } from ‘react-redux’ import fetchSearchData from ‘./action/fetchSearchData’; import SearchData from ‘./SearchData’; const Search = (props) => ( ); const mapStateToProps = (state) => ({ search: state.header.search.payload }); const mapDispatchToProps = { fetchSearchData }; export default connect(mapStateToProps, mapDispatch-ToProps)(Search) “‘

You can see in the code above that both `mapStateToProps` and `mapDispatchToProps` are functions that are provided a *state*

3 Sidenotes

- JavaScript’s `export` statement is used when creating JavaScript modules to export live bindings to functions, objects, or primitive values from the module so they can be used by other programs **with the `import`** statement.

4 References:

- How to build an eCommerce Website using React-Redux
- Pete Hunt: React: Rething best pratices – JSConf EU
- Introducing JSX
- Learning React in 5 minutes