Subscribing to Store Updates

Up until here we have dispatched our actions, made sure the reducer received them and then responded with updates. The next step is to receive a notification of the update using store.subscribe().

When you visit the bank, let the Cashier know your intended WITHDRAWAL action, and successfully receive your money, what next?

Most likely, you will receive an alert via email/text or some other mobile notification saying you have performed a transaction, and your new account balance is so and so.

If you don't receive mobile notifications, you'll definitely receive some sort of 'personal receipt' to show that a successful transaction was carried out on your account.

Okay, note the flow. An action was initiated, you received your money, you got an alert for a successful transaction.

We seem to be having a problem with our Redux code. An action has been successfully initiated, we've received money (state), but hey, where's the alert for a successful state update?

We've got none.

Well, there's a solution. Where I come from, you subscribe to receive transaction notifications from the bank either by email/text.

The same is true for Redux. If you want the updates, you've got to subscribe to it. But how?

The Redux store, whatever store you create has a subscribe method called like this: **store.subscribe()**.

Well named function, if you ask me!

The argument passed into store.subscribe() is a function, and it will be invoked whenever there’s a state update. For what it’s worth, please

remember that the argument passed into store.subscribe() should be a **function**. Okay?

Now let’s take advantage of this.

Think about it. After the state is updated, what do we want or expect? We expect a re-render, right?

So, state has been updated. Redux, please, re-render the app with the new state values.

Let’s have a look at where the app is being rendered in **index.js** Here’s what we’ve got.

```
ReactDOM.render(<App />, document.getElementById("root")
```

This is the line that renders the entire application.

It takes the <App/> component and renders it in the DOM. The **root** ID to be specific.

First, let’s abstract this into a function. See this:

```
const render = function() {
  ReactDOM.render(<App />, document.getElementById("root")
}
```

Since this is now within a function, we have to invoke the function to render the app.

```
const render = function() {
      ReactDOM.render(<App />, document.getElementById("root")
}
render()
```

Now, the <App /> will be rendered just like before. Using some ES6 goodies, the function can be made simpler.

```
const render = () => ReactDOM.render(<App />,
   document.getElementById("root"));
```

```
render();
```

Having the rendering of the <App/> wrapped within a function means we can now subscribe to updates to the store like this:

```
store.subscribe(render);
```

Where render is the entire render logic for the <App /> - the one we just refactored.

You understand what's happening here, right?

Anytime there's a successful update to the store, the <App/> will now be re-rendered with the new state values.

For clarity, here's the <App/> component:

```
class App extends Component {
  render() {
    return [
      <HelloWorld key={1} tech={store.getState().tech} />,
      <ButtonGroup key={2} technologies={["React", "Elm", "React-redux"]} />
    ];
  }
}
```

App.js

Whenever a re-render occurs, **store.getState()** on line 4 will now fetch the updated state.

Let's see if the app now works as expected.

```
export const setTechnology = tech => ({ type: "SET_TECHNOLOGY", tech });
```

Hell yeah! This works, and I knew we could do this!

We are successfully dispatching an action, receiving money from the Cashier, and then subscribing to receive notifications. Perfect!