A Peek at State Management with MobX - Exploring MobX state management with React

Written by Seth Corker on Benevolent Bytes

What is MobX?

MobX is a state management library which can be used with React much like Redux, its tagline is "Simple, scalable state management" and it definitely delivers on the first promise. Although it is not as popular and widespread as Redux, it is slowly gaining popularity so I thought I'd give it a go.

MobX has some pretty interesting ideas and takes some cues from reactive programming. It uses core concepts you may be familiar with if you've used RxJS. Values can be made observable so that when the value changes, anything that uses that value will update automatically. This is a pretty simple to grasp concept that can make working with data much more intuitive and that's why MobX offers something different to other state management libraries.

Should I switch to MobX?

Whether you should make the switch to MobX is up to you but the answer for an existing project is often no. Should you give it ago? Of course, it's an interesting approach that might make it easier to consume data and have your front-end react to those changes like magic.

MobX, like Redux, is another state management library and should be treated as such. You can often make do with React's `setState` in the early stages of an application if you don't think the state will grow too large. For tiny projects, it may not make a much sense but you'll know when you need a state management library, and when that time comes it might be a good idea to see if MobX is a good fit. You could of course always use MobX or Redux for smaller projects but it's more challenging to see the value when a lot of the problems of state management have yet to rear their ugly head.

My first impressions of MobX are good, it forgoes the large amount of setup that is often associated with Redux which makes it easy to understand as well as easy to get up and running. It is easy to integrate into parts of applications so it's not necessary to refactor large parts of your application to start with MobX.

Demo

import clip from "./1*Khu_eS44mhKOn2g644YnwQ.webm"

<AutoplayVideo src={[{ src: "s3-bucket://build/2018-05-14-a-peek-at-state-management-with-mobx/1*Khu_eS44mhKOn2g6
type: "video/webm", },]} poster="s3-bucket://build/2018-05-14-a-peek-at-state-management-with-mobx/1*Khu_eS44mhKOn2g644YnwQ-poster.jpg" />

State managed with MobX

I made a small contrived example to show what MobX can do and how it can be achieved with minimal effort.

The following CodeSandbox is an authentication front end that just takes an email address (No it's not secure but it's also not a real app). *notspiderman@email.com* is the only valid email which will progress the app to a logged in state.Try to login and see how the UI responds

This sample is made using react-create-app which is a great tool for starting react projects without the hassle of build configuration. One downside to this approach is the lack of support for decorators. MobX documentation makes extensive use of the JavaScript decorator syntax to make code more concise. It is possible to use MobX without decorators which is what I've done here. I think the longer syntax is fine and if you've come from Redux, you'll feel at home with this separation.

The bulk of the setup is done within the decorate function. For this example, it resides in the same file as the `App` itself but in a real project, it can easily be separated much like `mapStateToProps` in Redux, to make it easier to test or reuse components.

Below is the main logic for the simple example.

```
import React, { Component } from "react"
import { observer } from "mobx-react"
import { observable, computed, decorate, action } from "mobx"
import "./App.css"
```

```
decorate(App,
  {
    authenticated: observable,
    name: computed,
    message: observable,
    email: observable,
    loginEmail: observable,
    login: action,
    logout: action,
    setEmail: action,
  });

export default observer(App);
```

Figure 1: Decorating our app and wrapping it in an observer

```
const users = {
  "notspiderman@email.com": { name: "Peter Parker" },
}

class App extends Component {
  get name() {
    return this.email && users[this.email]
      ? users[this.email].name
      : "Not logged in"
  }

  setEmail = ({ target }) => {
    this.loginEmail = target.value
  }

  login = () => {
    if (this.loginEmail && users[this.loginEmail]) {
      this.email = this.loginEmail
      this.authenticated = true
      this.message = null
    } else {
      this.message = "You're not allowed in!"
      this.authenticated = false
    }
  }

  logout = () => {
    this.email = null
    this.message = "You have been logged out"
    this.authenticated = false
  }

  render() {
```

```
    return (
      <div className="container">
        <h2>{this.message}</h2>
        <header>
          <h5>{this.email}</h5>
          <h6>{this.name}</h6>
        </header>
        <main>
          {this.authenticated ? (
            <button onClick={this.logout}>Logout</button>
          ) : (
            <div>
              <input type="email" onChange={this.setEmail}></input>
              <button onClick={this.login}>Login</button>
            </div>
          )}
        </main>
      </div>
    )
  }
}

decorate(App, {
  authenticated: observable,
  name: computed,
  message: observable,
  email: observable,
  loginEmail: observable,
  login: action,
  logout: action,
  setEmail: action,
})

export default observer(App)
```

The app itself is very simple but you'll notice how simple it is to get working. There is no `setState` and we deal directly with class properties. We define some actions such as `login`, `logout` and `setEmail` which will affect some observables which we have also defined in `decorate()`, MobX handles the rest. The only other thing of note is that the `name: computed` on #60, this tells MobX that the value of 'name' is derived from other observable(s) and we are not changing it directly.

Wrapping up

I think MobX is a library with really great potential because it is far simpler to get started with a solution like Redux. The documentation is great and there quite a few learning resources to get you up to speed quickly. The code sample above can be expressed more concisely with the new JavaScript decorator syntax although as of the time of writing this, create-react-app doesn't support this syntax out of the box.

The biggest complaint I have about the MobX workflow is testing, probably due to my lack of understanding, it's not immediately obvious how components should be broken up for testing. It is a bit easier without using the decorator syntax to separate the logic from the view, however, the documentation doesn't touch on how this works when using the decorator syntax. The devtools are much easier to add to a project than the Redux devtools however they are not as fully featured. The benefit I suppose is that much of the digging you would be doing in Redux is not as necessary due to the fewer number of moving parts that are available to you through MobX.

Take a look at the MobX documentation to get started quickly or if you'd like to take a look at thefull source for the example, it's available here.

Thanks for taking the time to read about my experience with MobX. I hope it was useful in evaluating some benefits of MobX and how it works. What do you think of MobX and would you witch from Redux?

Photo by elizabeth lies on Unsplash and MobX logo from https://mobx.js.org/