YUShuttles.com

Built in

React.js

Mordechai Djavaheri

Web Development

Fall 2015



YUShuttles - React.js

# The Need

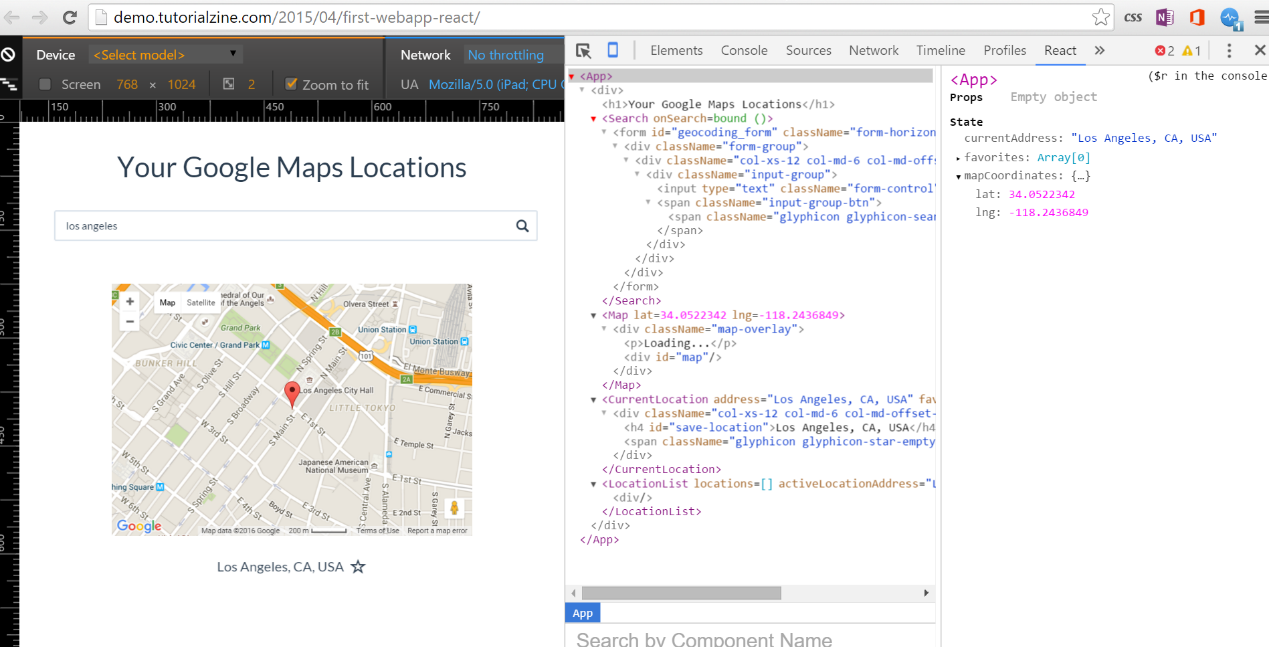
Let’s face it: the YU Shuttles web app deployment was pretty messy, resulting in numerous issues preventing students from registering for their ticket to skipping the subway (and perhaps finding their soulmates). Even in its current working implementation, the front end is just miserable. The color scheme is detestable, and screen real estate is not allocated efficiently. Let’s not forget that one is first prompted to make a WordPress account on the site first, via the standard WordPress GUI, before actually entering the site itself.

In effort to offer a new user interface and experience when signing up for the shuttle, while totally redoing the backend, React.js plays a critical role in presenting and handling the shuttle timetables and other data. Also known as a pure V in the MVC, React presents and maintains the data with a flux architecture that creates and rerenders components as data is requested and user input is fielded.

The highlights of this project include two-way data binding and page routing, as well as a cleaner user interface, and a new user experience. The relevant source code can be found in the shuttles.js file.

My project is accessible from ycfac197.mc.yu.edu/~djavaher/project/index.html

# React Component Structure

 React creates components that handle data and render HTML objects that can create even more components, resulting in a DOM tree like hierarchy. Specifically, React has its own Virtual DOM, accessible via a Chrome developer tools plugin, broken down by components.

# Two Way Data Binding

Not only can data be passed from parent components to children, children can return values that allow it to alter the parents state (similar to private fields), thereby causing the parent to update and rerender any child components that use that data.

In the shuttle app, for example, the Controller component “owns” the origin data and passes it on to its child components, TimeTable, Options, and SeatSelector; therefore, origin is declared as a state value of Controller. The child components will be able to use the origin as a parameter stored in this.props.

For reverse data flow, Controller passes an update method to Options. When a user uses the select input to switch origins from “Wilf” to “Beren,” Options will pass the new value back up to Controller via that method and the value of Controller.state.origin will be updated. React will then rerender both TimeTable and SeatSelector to show the corresponding schedule and seat availability. See source code (lines 68-129).

# Page Routing

With so many components representing different parts of the DOM, using separate html files for different pages becomes anachronistic. One can more efficiently present views of different pages by just generating different components in a parent-child routing system. React-router does just that by declaring parent pages and their children, in a quasi-template schemed view of parent template pages and their content-children, if any. Instead of ReactDOM.render(<RootComponent />, which is followed by document.getElementById(“#foo”)), we replace <RootComponent /> with the routing tree, which includes a page history component, as well.

ReactDOM.render(

<ReactRouter.Router history={ReactRouter.browserHistory}>

<ReactRouter.Route path="/" component={App}>

<ReactRouter.IndexRoute component={Home}/>

<ReactRouter.Route path="shuttles" component={Controller}/>

</ReactRouter.Route>

</ReactRouter.Router>,

document.getElementById('container')

);

# Seat Selector

A cute feature that separates the shuttles from the busses is the ability to pick your seat. Perhaps this is just overdoing it and unnecessary, but innovation has to start somewhere. When a user clicks on a ShuttleRow component, it toggles the shuttleNumber state in Controller, which subsequently updates all the child components, including SeatSelector. SeatSelector then displays divs color coded for open and closed. Once a user chooses a seat, he hits Register, which send the info to Controller to update the seat availability for that shuttle.

The default seat is set to -1, so the relevant components can check its value before allowing one to register. Also, the default shuttle is set to 0, so that there’s some shuttle displayed when the user lands on the page.

Instead of having these all one above the other, a side by side presentation was designed to take full advantage of the available screen real estate.

Due to mysterious technical difficulties, mainly a non-rerendering Controller component, the SeatSelector module hasn’t been implemented successfully, and, in its place, one simply registers for a seat by tapping on a ShuttleRow.

# Experience

## Takeaways

Firstly, a major takeaway from the project was learning about routing. To have a script with tons of components on one page doing different things also change the URL was foreign to me. Being introduced to the concept of a router was an interesting experience.

Also, I had heard the name Babel here and there, but it usually got mixed in with all the other terms I’m unfamiliar with, like LESS and SASS. Given the opportunity in this project to use the JSX format and precompile or compile in production was sort of eyebrow raising. It’s just so much easier to write in JSX, and I’m glad that I can!

## Setbacks

At first, I had some issues passing the schedule JSON around to the right child components, but I eventually got it to work for the most TimeTable, but not the SeatSelector. I may have been missing something about implementing the architecture they describe (perhaps due to insufficient documentation, but I can’t point fingers when working under pressure – I may have missed something or didn’t Google enough).

Unfortunately, the current project is unable to run, because of some issue I could not understand. At first, it said that there was some unexpected <!— character prepended to my script, but I couldn’t find any, despite using numerous text editors, in case it was hidden. I determined that it must have been the JSX Transformer compiler, which Facebook did say should only be used in production and was intended only for the tutorial and otherwise deprecated.

Switching to the Babel.js precompiler, unfortunately, in addition to not helping, forced me to work with a ton of errors when being forced to switch from .jsx to .js. Eventually I decided to just keep my work in the .jsx file, do the precompiling myself on the Babel website, and just upload the output in a separate .js file.

It didn’t help, and my once positive future of automating the shuttle reservation system stalled before it got out of the garage. If all went well, it would’ve looked something like this:

