The Firebase Blog

Jacob Wenger
Core Developer

Using Firebase with ReactJS

May 1, 2014

ReactJS is a JavaScript library built by Facebook and Instagram which makes it easy to build large, complex user interfaces. It is intended to be used in combination with another framework which provides the backend. While Angular, Ember, and Backbone are popular choices for that, Firebase provides the easiest and quickest integration of a persistent, realtime backend into a React app - and it takes only a few lines of JavaScript.

Note: This post requires a basic knowledge of React. You can get up to speed quickly by looking through the short tutorial on the React homepage.

The creators of React describe it as the "V[iew] in MVC." It is not meant to be a replacement for Angular or Ember; instead, it is meant to extend their functionality by providing a high-performance way to keep a view up-to-date with JavaScript. Its special sauce is that it renders HTML using an incredibly fast virtual DOM diff algorithm, providing much better performance than competing platforms. It has a "one-way reactive data flow" which is much simpler to understand than traditional data-binding. Components - the basic building blocks of React apps - are organized in a tree hierarchy in which parent components send data down to their children through the `props` variable. Each component also has a `state` variable which determines the current data for its view. Whenever `state` is changed, the component's `render()` method is called and React figures out the most efficient way to update the DOM.

Since React's main focus is on the user interface, apps made with it need something else to act as their backend. That is where Firebase comes in. It adds the "M[odel] and C[ontroller] in MVC" to React apps, making them fully functioning apps. Using React's straightforward binding system, it is easy to integrate Firebase in a native way with only a small amount of code.

Adding Firebase to a React app

Let's look at the Todo app on the React homepage. Within the `TodoApp` component, `this.state` is used to keep track of the input text and the list of Todo items. While React ensures that the DOM stays in sync with any changes to `this.state`, the changes do not persist beyond the life

items disappear! This is because React has no mechanism for storing data beyond the scope of the current page session. It relies on being used with another framework to do that.

Firebase is a natural complement to React as it provides React apps with a persistent, realtime backend. The first thing we need to do is add Firebase to the project:

```
<head>
  <!-- React JS -->
  <script src="http://fb.me/react-0.10.0.min.js"></script>
  <script src="http://fb.me/JSXTransformer-0.10.0.js"></script>

  <!-- Firebase JS -->
  <script src="https://cdn.firebase.com/js/client/1.0.17/firebase.js"
</head>
```

Now that we have included Firebase, we can populate the list of Todo items by reading them from Firebase. We do this by hooking into the `componentWillMount()` method of the `TodoApp` component which is run once, immediately before the initial rendering of the component:

```
componentWillMount: function() {
  this.firebaseRef = new Firebase("https://ReactFireTodoApp.firebasei
  this.firebaseRef.on("child_added", function(dataSnapshot) {
    this.items.push(dataSnapshot.val());
    this.setState({
      items: this.items
    });
  }.bind(this));
}
```

This code first gets a reference to the `items` node at the root of the Firebase. The call to `on()` will be run every time a node is added under

be fired for every item under the `items` node in Firebase, not just new ones that are added to it. Therefore, when the page is loaded, every existing child under the `items` node will fire a `child_added` event, meaning they can easily be iterated over and added to `this.state.items`. Note that the call at the end to `bind()` just sets the scope of callback function to `this`.

Now what about adding new Todo items to the Firebase? That code is just as easy:

```
handleSubmit: function(e) {
  e.preventDefault();
  this.firebaseRef.push({
    text: this.state.text
  });
  this.setState({text: ""});
}
```

Within `handleSubmit()` a new item is pushed onto the saved Firebase reference which appends it to the end of the `items` node. The call to `setState()` updates `this.state.text` but does not need to update `this.state.items` as it did in the original React code. This is because the `child_added` event handler from `componentWillMount()` will be fired when a new child is pushed onto the `items` Firebase node and that code will update `this.state.items`.

The last thing that needs to happen is cleaning up the Firebase event handler:

```
componentWillUnmount: function() {
  this.firebaseRef.off();
}
```

updated in realtime. Best of all, the items stick around if the page is refreshed! You can even open multiple tabs pointed at the same page and see them all update simultaneously, with Firebase doing all the heavy lifting. Take some time to view the code for this example and play around with a live demo (Note: it's the second one on the demo page).

Introducing the ReactFireMixin

Although integrating Firebase into a React app only takes a few lines of code out of the box, we wanted to make it even easier. That is why we built the ReactFireMixin for React which makes it simple to keep `this.state` in sync with a Firebase node.

We suggest you download ReactFire directly from our CDN:

```
<!-- ReactFire -->
<script src="https://cdn.firebase.com/libs/reactfire/0.1.6/reactfire.
```

ReactFire is also available from npm via `npm install reactfire` or from Bower via `bower install reactfire`.

To then use the ReactFireMixin in the `TodoApp` component, add it to the component's mixins property:

```
var TodoApp = React.createClass({
  mixins: [ReactFireMixin],
  ...
});
```

The ReactFireMixin extends the functionality of the `TodoApp` component, adding additional Firebase-specific methods to it. To keep

Firebase, make the following change in `componentWillMount()`:

```
componentWillMount: function() {
  this.bindAsArray(new Firebase("https://ReactFireTodoApp.firebaseio.
}
```

We simply specify that we want to bind a particular Firebase reference to `this.state.items` of the React component. The ReactFireMixin allows binding to a Firebase node as an array or as a regular JavaScript object. This creates a one-way binding from the Firebase reference to `this.state.items`, meaning that if the data in the Firebase changes, so will `this.state.items`. However, if we update `this.state.items`, the Firebase will not change. Therefore, changes should be made directly to the Firebase and not by calling `setState()`:

```
handleSubmit: function(e) {
  e.preventDefault();
  this.firebaseRefs["items"].push({
    text: this.state.text
  });
  this.setState({text: ""});
}
```

The ReactFireMixin allows for binding to multiple things at once. Firebase ensures that this is all done in an efficient manner. To access the Firebase reference which is bound to `this.state.items`, we can reference `this.firebaseRefs["items"]` which is provided by the ReactFireMixin. Finally, calling `this.firebaseRef.off()` is no longer needed in `componentWillUnmount()` as the mixin handles this behind the scenes.

(Note: it's the last one on the demo page).

Conclusion

ReactJS is a wonderful framework for creating user interfaces. When picking a complementary tool to use alongside it as a backend, Firebase is the easiest and most powerful solution. In just a few lines of code you can get a React app syncing data across hundreds of clients at once. Our new ReactFireMixin makes this that much easier, getting rid of even more boilerplate code.

We encourage you to try out the new ReactFireMixin for React and let us know what you think. If you have any questions or feedback, post them in our Firebase Google Group or email me at jacob@firebase.com. Submit a pull request to our examples repo if you have any Firebase + React examples you would like to share.

We are excited to see what you create!


Firebase