React Demo 1: Creating a Simple React App

Let's take a look at how we can write a simple React front end app using the Rails API-only app we wrote for our blog demo backend. In particular, for now we'll focus on implementing the basic CRUID interface to our authors entity.

Prerequsitites:

You need several software packages to get started:

Node.js

First, make sure you've added installed the latest version of Node.js in your development environment.

create-react-app

Once you have node, we're going to install a utility that will make it easier to create a stubbed out functional React project with all the dependencies and moving parts in place.

```
npm install -g create-react-app
```

Atom Editor

Also, we recommend you use a text editor that can do proper syntax highlighting of React code, and ES6 in particular. We are particularly fond of the Atom editor by GitHub. If you happen to use Atom, then you'll need to add a number of third party "Community Packages" (e.g. plugins) under the editors preferences. These are the ones we use for React and related web development (use the latest versions of these in case they've been updated since thit was written):

```
autoclose-html@0.23.0
autocomplete-modules@1.12.0
busy-signal@1.4.3
es6-javascript@1.0.0
intentions@1.1.5
language-babel@2.84.0
linter@2.2.0
linter-eslint@8.4.1
linter-ui-default@1.7.1
```

React Developer Tools Plugin for Chrome

Last but not least, we'd recommend you use Chrome to test your web app, and make sure you've installed the React Developer Tools plugin to faciltate debugging / inspecting React apps while they are running.

You should now be ready to do some React coding!

1. Let's start by building a stubbed out react app with dependencies:

   ```
   create-react-app blog-frontend
   ```

Then change directory to the created sub-directory for the remaining steps.

```
cd blog-frontend
```

2. You can run the app with the following command:

```
npm start
```

Go ahead and poke around in the JavaScript in the src directory, and you can see that the index.js instantiates an App component which is defined in App.js.

3. We're going to build a front end app for our Author CRUD operations that provides us a form to create new and edit existing author entities, and another component that simply lists out all the authors in the system. Note in the mockup we sort of have 3 components - an AuthorForm, an AuthorList, and then an outer component Authors that will act as the container of these two child components.
Go ahead and delete all files App.js, App.test. and App.css from the src directory, as well as the related imports in the index.js file.

4. We'll start with a couple of really simple JavaScript functions that scaffold out the AuthorForm and AuthorList components. Add these functions to src/index.js.

```
const AuthorForm = (props) => {
  return (
    <div className="author-form">
      Our Author Form Goes Here.
    </div>
  );
}

const AuthorList = (props) => {
  return (
    <div className="author-list">
      Our Author List Goes Here.
    </div>
  );
}
```

5. Next, we're going to define our Authors component (the container of the above two components) as an ES6 class. We're going to do this because this top level container is going to hold the state of our front-end. This is a common approach in building a React App.

```
class Authors extends React.Component {

  constructor(props) {
    super(props);
    this.state = {
      authors: [],
    };
  }

  render() {
    return (
      <div className="authors">
        <AuthorForm />
        <AuthorList />
      </div>
    );
  }
}
```

Note that our constructor defines the initial state - at this point, just an empty array of authors.

6. In order to get this new hierarchy of components to render, we need to replace the App element in the ReactDOM.render call at the bottom of index.js with an Authors element.

```
ReactDOM.render(<Authors />, document.getElementById('root'));
```

At this point, make sure your web server is running and take a look at your web page. You should see the placeholder text for each of the two components displaying.

7. Let's begin fleshing out the AuthorList demo. However, before we do that, we're going to import the Bootstrap CSS framework into our project to keep things pretty:

```
npm install --save react-bootstrap bootstrap@3
```

Then add the following imports to the top of your index.js:

```
import 'bootstrap/dist/css/bootstrap.css';
import 'bootstrap/dist/css/bootstrap-theme.css';
```

8. We can't really display any authors until we have some, so for now we will just introduce a simple function to our Authors class that generates a hardcoded array of author objects and adds to our state. Later on, we'll learn how to grab this data from our API:

```
  loadAuthors() {
    this.setState({
      authors: [
        {fname: "sam", lname: "iam", email: "sam@aol.com"},
        {fname: "jane", lname: "doe", email: "jane@aol.com"},
        {fname: "fred", lname: "bear", email: "fred@aol.com"},
        {fname: "ted", lname: "tooy", email: "ted@aol.com"},
      ]}
    );
  }
```

Given how function context binding works in JavaScript, we also need to explicitly bind this function by adding this line of code to the constructor or Authors:

```
this.loadAuthors = this.loadAuthors.bind(this);
```

9. The next question you might have is, when does this function get called? It turns out we are going use the React component's lifecycle method componentDidMount to accomplish this. We can override this method in our Authors class definition, and React will automaticaly call it when the component is initially created:

```
  componentDidMount() {
    console.log('Authors mounted!')
    this.loadAuthors();
  }
```

10. However, note that at this point, only our top level Authors component knows aobut the state. If AuthorList is going to display our authors, somehow we need to communicate this information to the AuthorList component. We can do this by adding an attribute to the AuthorList element in the JSX in the render of our Authors component:

```
<AuthorList authors={this.state.authors} />
```

That is, we're passing our state variable to the AuthorList component as a property!

11. Now we're ready to flesh out our AuthorList component. Looking more closely at our user interface mockup, notice that each row in our author listing is identical! So to practice the DRY principle, we're going to introduce an AuthorListComponent that simply renders a single row, based on the properties we send it:

```
const AuthorListItem  = (props) =>  {
  return (
    <tr>
        <td className="col-md-3">{props.fname}</td>
        <td className="col-md-3">{props.lname}</td>
        <td className="col-md-3">{props.email}</td>
        <td className="col-md-3 btn-toolbar">
          <button className="btn btn-success btn-sm" onClick={event => props.onEdit("edit",props)}
>
            <i className="glyphicon glyphicon-pencil"></i> Edit
          </button>
          <button className="btn btn-danger btn-sm" onClick={event => props.onDelete(props.id)}>
            <i className="glyphicon glyphicon-remove"></i> Delete
          </button>
        </td>
      </tr>
  );
}
```

Note that this component is simply rendering straightforward HTML with bootstrap styling. Since we are already familiar with that we'll gloss over the details, only to point out that the onClick event handlers for the edit /delete buttons we're calling here will also need to be sent in eventually via the properties we send this component.

12. With the AuthorListItem component in place, we can now rewrite our AuthorList component in terms of it:

```
const AuthorList = (props) => {
  const authorItems = props.authors.map((author)  => {
    return (
      <AuthorListItem
        fname={author.fname}
        lname={author.lname}
        email={author.email}
        id={author.id}
        key={author.id}
        onDelete={props.onDelete}
        onEdit={props.onEdit}
       />
    )
  });

  return (
    <div className="author-list">
      <table className="table table-hover">
        <thead>
          <tr>
            <th className="col-md-3">First Name</th>
            <th className="col-md-3">Last Name</th>
            <th className="col-md-3">Email</th>
            <th className="col-md-3">Actions</th>
          </tr>
        </thead>
        <tbody>
          {authorItems}
        </tbody>
      </table>
    </div>
  );
}
```

Note the references to the onDelete/onEdit properties. These aren't populated yet, so if we press the buttons we'll get an error. We'll take care of those later. For now, take a look at your handiwork in the web browser.

13. Let's add some margin around our components so they don't get pushed up against the sides of the browser. Add to the content of the src/index.css with the following rules:

```
.author-form  {
  margin: 20px;
}

.author-list  {
  margin: 20px;
}
```