Higher-Order Components

Now that we've implemented the sign-up system in our application, let's figure out a way to improve it.

In the previous lesson, we made the sign-up system in our React app work through Firebase. However, we can still make things easier.

A Simpler Way

Instead of using a **render prop** component, which is automatically provided with React's Context Consumer component, it may be simpler to use a **higher-order** component.

Let's implement this higher-order component in the `src/components/` `Firebase/context.js`:

```
import React from 'react';

const FirebaseContext = React.createContext(null);

export const withFirebase = Component => props => (
  <FirebaseContext.Consumer>
    {firebase => <Component {...props} firebase={firebase} />}
  </FirebaseContext.Consumer>
);

export default FirebaseContext;
```

Firebase/context.js

Next, we'll make it available via our Firebase module in the `src/components/` `Firebase/index.js` file:

```
import FirebaseContext, { withFirebase } from './context';
import Firebase from './firebase';


export default Firebase;

export { FirebaseContext, withFirebase };
```

Firebase/index.js

Now, instead of using the Firebase Context directly in the `SignUpPage`, which doesn't need to know about the Firebase instance, we'll use the higher-order components to wrap your `SignUpForm`.

Afterward, the `SignUpForm` has access to the Firebase instance via the higher-order component. It is also possible to use the `SignUpForm` as a standalone component without the SignUpPage because it is responsible for getting the Firebase instance via the higher-order component.

```
import React, { Component } from 'react';
import { Link } from 'react-router-dom';

import { withFirebase } from '../Firebase';
import * as ROUTES from '../../constants/routes';

const SignUpPage = () => (
  <div>
    <h1>SignUp</h1>
    <SignUpForm />
  </div>
);

const INITIAL_STATE = { ... };

class SignUpFormBase extends Component {
  ...
}

const SignUpLink = () => ...

const SignUpForm = withFirebase(SignUpFormBase);

export default SignUpPage;

export { SignUpForm, SignUpLink };
```

SignUp/index.js

Redirecting the User

Like many applications, we may want to redirect the user to another page after he or she signs up. This could be the user's home page which is a protected route for authenticated users only.

We will need the help of the React Router to redirect the user after a successful sign-up.

```
import React, { Component } from 'react';
import { Link, withRouter } from 'react-router-dom';

import { withFirebase } from '../Firebase';
import * as ROUTES from '../../constants/routes';

...

class SignUpFormBase extends Component {

  ...

  onSubmit = (event) => {
    const { username, email, passwordOne } = this.state;

    this.props.firebase
      .doCreateUserWithEmailAndPassword(email, passwordOne)
      .then(authUser => {
        this.setState({ ...INITIAL_STATE });
        this.props.history.push(ROUTES.HOME);
      })
      .catch(error => {
        this.setState({ error });
      });

    event.preventDefault();
  }

  ...
}

...

const SignUpForm = withRouter(withFirebase(SignUpFormBase));

export default SignUpPage;

export { SignUpForm, SignUpLink };
```

SignUp/index.js

Let's take this code block apart. To redirect a user to another page programmatically, we need access to the React Router. Fortunately, the React Router node package offers a higher-order component to make the router properties accessible in the props of a component.

Any component that goes in the `withRouter()` higher-order component gains access to all properties of the router. So, when the enhanced SignUpFormBase component is passed to the `withRouter()` higher-order component, it has access to the props of the router. The relevant property from the router props is the `history` object because it allows us to redirect a user to another page by pushing a route to it.

The `history` object of the router can eventually be used in the `onSubmit()` class method. If a request resolves successfully, we can push any route to the history object. Since the pushed `/home` route is defined in our App component with a matching component to be rendered, the displayed page component will change after the redirect.

We're almost done with the sign-up process. In the next lesson, we will make a few final changes and see if users can sign up to our application.