Session Handling with Higher-Order Components

This lesson will go through Session Handling using Higher-order components.

WE'LL COVER THE FOLLOWING

- React Context
- Creating a Higher-Order Component

We implemented a basic version of session handling in the previous lesson. However, the authenticated user still needs to be passed down from the App component to other components interested in using it.

This can become tedious over time because the authenticated user has to be passed through all components until it reaches all the leaf components.

Previously, we used the **React Context API** to pass down the *Firebase instance* to other components. Now, we will do the same for the authenticated user.

React Context

In a **new** `src/components/` **`Session/context.js`** file, place the following new React Context for the session (authenticated user):

```
import React from 'react';

const AuthUserContext = React.createContext(null);

export default AuthUserContext;
```

Session/context.js

Next, we *import* and *export* it from the `src/components/` **`Session/index.js`** file which is the entry point to this module:

```
import AuthUserContext from './context';
```

```
export { AuthUserContext };
```

Session/index.js

The App component can use the **new context** in order to provide the authenticated user to components that are interested in using it by adding the following code:

```
...

import { AuthUserContext } from '../Session';

class App extends Component {
  ...

  render() {
    return (
      <AuthUserContext.Provider value={this.state.authUser}>
        <Router>
          <div>
            <Navigation />

            <hr />

            ...
          </div>
        </Router>
      </AuthUserContext.Provider>
    );
  }
}

export default withFirebase(App);
```

App/index.js

The `authUser` does not need to be passed to the Navigation component anymore. Instead, the Navigation component uses the new context to consume the authenticated user:

```
...

import { AuthUserContext } from '../Session';

const Navigation = () => (
  <div>
    <AuthUserContext.Consumer>
      {authUser =>
        authUser ? <NavigationAuth /> : <NavigationNonAuth />
      }
    </AuthUserContext.Consumer>
  </div>
);
```

Navigation/index.js

Creating a Higher-Order Component

The application works the same way as before, except that any component can simply use **React's Context** to use the authenticated user. To keep the App component clean and concise, we prefer to extract the session handling for the authenticated user part to a separate higher-order component in a **new** `src/components/` **`Session/withAuthentication.js`** file:

```
import React from 'react';

const withAuthentication = Component => {
  class WithAuthentication extends React.Component {
    render() {
      return <Component {...this.props} />;
    }
  }

  return WithAuthentication;
};

export default withAuthentication;
```

Session/withAuthentication.js

We move all the logic that deals with the authenticated user from the App component to the `src/components/` **`Session/withAuthentication.js`** file as shown in the code snippet below:

```
import React from 'react';

import AuthUserContext from './context';
import { withFirebase } from '../Firebase';

const withAuthentication = Component => {
  class WithAuthentication extends React.Component {
    constructor(props) {
      super(props);

      this.state = {
        authUser: null,
      };
    }

    componentDidMount() {
      this.listener = this.props.firebase.auth.onAuthStateChanged(
        authUser => {
          authUser
```

```
            ? this.setState({ authUser })
            : this.setState({ authUser: null });
        },

      );
    }

    componentWillUnmount() {
      this.listener();
    }

    render() {
      return (
        <AuthUserContext.Provider value={this.state.authUser}>
          <Component {...this.props} />
        </AuthUserContext.Provider>
      );
    }
  }

  return withFirebase(WithAuthentication);
};

export default withAuthentication;
```

Session/withAuthentication.js

As seen from the code above, the *withAuthentication component* also uses the new React Context in order to provide the authenticated user. The App component will not be in charge of it anymore.

Next, we *export* the higher-order component from the `src/components/` **`Session/index.js`** file, so that it can be used in the App component afterward:

```
import AuthUserContext from './context';
import withAuthentication from './withAuthentication';

export { AuthUserContext, withAuthentication };
```

Session/index.js

Finally, the App component has successfully become a function component again, without the additional business logic related to the authenticated user. Now it uses the higher-order component to make the authenticated user available for all other components.

The code snippet below shows how the `src/components/` **`App/index.js`** file (App component) should look like after all the modifications:

```
import React from 'react';
import { BrowserRouter as Router, Route } from 'react-router-dom';

import Navigation from '../Navigation';
import LandingPage from '../Landing';
import SignUpPage from '../SignUp';
import SignInPage from '../SignIn';
import PasswordForgetPage from '../PasswordForget';
import HomePage from '../Home';
import AccountPage from '../Account';
import AdminPage from '../Admin';

import * as ROUTES from '../../constants/routes';
import { withAuthentication } from '../Session';

const App = () => (
  <Router>
    <div>
      <Navigation />

      <hr />

      <Route exact path={ROUTES.LANDING} component={LandingPage} />
      <Route path={ROUTES.SIGN_UP} component={SignUpPage} />
      <Route path={ROUTES.SIGN_IN} component={SignInPage} />
      <Route
        path={ROUTES.PASSWORD_FORGET}
        component={PasswordForgetPage}
      />
      <Route path={ROUTES.HOME} component={HomePage} />
      <Route path={ROUTES.ACCOUNT} component={AccountPage} />
      <Route path={ROUTES.ADMIN} component={AdminPage} />
    </div>
  </Router>
);

export default withAuthentication(App);
```

App/index.js

Start the application to verify that it still works. Note that we did not change any behavior in this lesson, but shielded away from the more complex logic into a higher-order component. Also, the application now implicitly passes the authenticated user via React's Context, rather than explicitly via the component tree using props.

Next, we will run and verify our app.