Session Handling with Firebase/React

This section will deal with how our React application manages user sessions after a successful sign-in.

WE'LL COVER THE FOLLOWING

This lesson is very crucial for the authentication process. Since we have all the components required to fulfill an authentication round-trip in React, all that is missing now is an *overseer* to overlook and inspect the **session state**.

The information of the currently authenticated user needs to be stored and made accessible to other components. This is the point where developers most often start using a *state management library* like **Redux or MobX**. In this course, we'll make do by using the **global state** instead of state management libraries.

Since we have been building our application under the umbrella of the **App component**, it will be sufficient to manage the session state in the App component using React's local state. The App component only needs to keep track of an authenticated user (session).

The App Component and the Session State

If a user is authenticated, i.e. successfully logged into the app, we either store it in the local state and pass the authenticated user object down to all components that might be interested in using it or pass the authenticated user down as `null`. That way, all components interested in using it can adjust their behavior (e.g. use conditional rendering) based on the session state behavior (e.g. use conditional rendering) based on the session state.

For example, the **Navigation component** would be interested because it needs to show different options to authenticated as well as non-authenticated users. Another example is that the **SignOut component** shouldn't show up for a non-authenticated user (a user that hasn't signed into the app yet).

Implementation

We will implement **session handling** in the App component in the `src/components/App/index.js` file. As the component handles the local state now, we will have to refactor it to a class component, which basically manages the local state of an `authUser` object and then passes it to the **Navigation component**.

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

...

class App extends Component {
  constructor(props) {
    super(props);

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

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

          <hr/>

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

export default App;
```

App/index.js

Authentication-Based Navigation

The **Navigation component** can be made aware of the authenticated user in order to display different options. It should only show links that are available for an authenticated user or a non-authenticated user, respectively.

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

import SignOutButton from '../SignOut';
import * as ROUTES from '../../constants/routes';

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

const NavigationAuth = () => (
  <ul>
    <li>
      <Link to={ROUTES.LANDING}>Landing</Link>
    </li>
    <li>
      <Link to={ROUTES.HOME}>Home</Link>
    </li>
    <li>
      <Link to={ROUTES.ACCOUNT}>Account</Link>
    </li>
    <li>
      <SignOutButton />
    </li>
  </ul>
);

const NavigationNonAuth = () => (
  <ul>
    <li>
      <Link to={ROUTES.LANDING}>Landing</Link>
    </li>
    <li>
      <Link to={ROUTES.SIGN_IN}>Sign In</Link>
    </li>
  </ul>
);

export default Navigation;
```

App/index.js

Now let's see where the `authUser` (authenticated user) comes from in the App component. Firebase offers a **listener function** in order to get the authenticated user from Firebase.

The Firebase Listener

```
...

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

class App extends Component {
  constructor(props) {
    super(props);

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

  componentDidMount() {
    this.props.firebase.auth.onAuthStateChanged(authUser => {
      authUser
        ? this.setState({ authUser })
        : this.setState({ authUser: null });
    });
  }

  ...

}

export default withFirebase(App);
```

App/index.js

Explanation

The helper function `onAuthStateChanged()` receives a function as a parameter that has access to the authenticated user. The passed function is called every time something changes for the authenticated user. It is called when a user signs up, signs in, and signs out. If a user signs out, the `authUser` object becomes null, so the `authUser` property in the local state is set to null and all components depending on it adjust their behavior accordingly (e.g. display different options such as the Navigation component).

We also want to avoid memory leaks that lead to **performance issues**, so we'll remove the listener if the component unmounts.

```
...

class App extends Component {
  ...

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

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

  componentWillUnmount() {
    this.listener();
  }

  ...

}

export default withFirebase(App);
```

App/index.js

Start the application to verify that your sign-up, sign-in, and sign-out functionality works and that the Navigation component displays the options depending on the session state, i.e. renders different displays according to the authenticated or non-authenticated user.

Congratulations, we have successfully implemented the authentication process using Firebase in our React app. We can run and verify this in the next lesson.