Curso de React y React Native

Clase 14

Agenda de la clase

Agenda

- Autenticación en React.
- Ejercicio.

Autenticación en React

Autenticación (1/14)

A la hora de crear una webapp, a veces es necesario proteger ciertas rutas respecto a los usuarios que no tienen la “autorización” adecuada. Aunque React Router no proporciona ninguna funcionalidad para esto, fue hecho bajo el paradigma de componentes reutilizables, lo que significa que agregarlo por la nuestra es bastante sencillo.

Incluso, **antes de crear nuestras rutas protegidas**, vamos a necesitar una forma de averiguar si el usuario está autenticado.

Autenticación (2/14)

El “permiso” para acceder a los datos de cualquier aplicación proviene de un servidor. Por este motivo, nuestra forma de manejar la autenticación a nivel de UI debe ser flexible para soportar las distintas formas que existen.

Algunos servidores otorgan JWT como forma de acceso, otros cookies, entre otros métodos.

En cualquier caso, la lógica para detectar la presencia y validez es **ajena a React y a nuestras rutas**.

Autenticación (3/14)

Por este motivo, vamos a ver un ejemplo con una autenticación “artificial”, de esta forma será trivial que cada uno incorpore la autenticación y detección de autenticación que corresponda a su caso de uso. De vuelta: Nuestra forma de manejar la autenticación será flexible para soportar las distintas formas de auth que existen.

Autenticación (4/14)

```
const fakeAuth = {
 isAuthenticated: false,  // Acá podríamos indicar la presencia de un JWT o cookie
 authenticate(cb) {
   this.isAuthenticated = true
   setTimeout(cb, 100) // Simulamos la demora de una llamada a una API
 },
 signout(cb) {
   this.isAuthenticated = false
   setTimeout(cb, 100) // Simulamos la demora de una llamada a una API
 }
}
```

Autenticación (5/14)

Ahora que ya tenemos una “simulación” de autenticación, vamos a construir los componentes que serán renderizados por React Router para mostrar según la autenticación.

Son tres:

- Componente público
- Componente privado
- Un componente de “inicio de sesión”

Autenticación (6/14)

```
const Public = () => <h3>Público</h3>

const Private = () => <h3>Privado</h3>

class Login extends React.Component {
 render() {
   return (
     <div>
       Login
     </div>
   )
 }
}
```

Autenticación (7/14)

Ahora que tenemos algunos componentes, el siguiente paso es empezar a renderizar las Rutas. Antes de empezar a preocuparnos por la creación de rutas privadas, creemos las Rutas `/public` y `/login` y los links para `/public` y `/private`.

Autenticación (8/14)

```
const App = () => (
 <Router>
   <div>
     <ul>
       <li><Link to="/public">Link a página pública</Link></li>
       <li><Link to="/private">Link a página privada</Link></li>
     </ul>
     <Route path="/public" component={Public} />
     <Route path="/login" component={Login} />
   </div>
 </Router>
);
```

Autenticación (9/14)

La idea es que cualquier usuario pueda acceder a `/public` (y por lo tanto ver el componente `<Public />`), pero si no estando *loggeado* intenta acceder a `/private`, que sea redirigido a `/login`.

El siguiente paso es renderizar un `<Route />` que apunte a `/private`. El problema es que al renderizar un `<Route />` normal, cualquiera podrá acceder a él, lo cual no queremos.

Estaría bueno un componente, por ejemplo `<PrivateRoute />`, ¿no?

Autenticación (10/14)

Bueno, no existe.

Autenticación (11/14)

Y está bien que no exista, porque involucra, como decíamos anteriormente, detalles de implementación de nuestro lado y React Router, como cualquier librería, debe mantenerse agnóstica a nuestros detalles de implementación.

Igualmente, vamos a crear `<PrivateRoute />` y vamos a ver que es sencillo y que simplemente se apoya en otros componentes que expone React Router.

La idea, es que `<PrivateRoute />` se tenga la misma API que `<Route />` tradicional, y que se vea de esta forma:

Autenticación (12/14)

```
<Route path="/public" component={Public} />

<Route path="/login" component={Login} />

<PrivateRoute path='/private' component={Private} />
```

Autenticación (13/14)

Entonces, nuestro componente `<PrivateRoute />`:

- Va a tener la misma API que `<Route />`.
- Internamente comprueba si el usuario está autenticado, haciendo uso de nuestra propia lógica de autenticación.
- Usa una `<Route />` tradicional y le pasa todas las props que recibió **excepto** `component`.
- En la prop `render` de dicha `<Route />`, si el usuario está autenticado mostramos `component`, si no está autenticado, lo redirigimos a `/login`, utilizando el componente `<Redirect />` que expone React Router.

Autenticación (14/14)

```
const PrivateRoute = ({ component: Component, ...otherProps }) => (
 <Route {...otherProps} render={props => (
   fakeAuth.isAuthenticated === true
     ? <Component {...props} />
     : <Redirect to='/login' />
 )} />
);
```

Ejercicio

Ejercicio

1. Crear un proyecto de React.
2. Crear tres rutas:
   a. `/` la cual será pública (una pantalla de bienvenida con links a `/private` y `/login`).
   b. `/private` la cual será privada (ante la ausencia de autorización, redirige a `/login`).
   c. `/login` la cual será pública.
3. En `/login`, crear un formulario de autenticación, el cual hará uso del microservicio https://ha-auth.now.sh/auth para obtener un JWT al enviarle las credenciales. El `username` es `hack` el `password` es `academy`.
4. Utilizaremos el localStorage para almacenar el token obtenido, cuestión de, cuando exista dicho token, incorporarlo por defecto a los headers para interactuar con endpoints privados en la APIs a consumir, y así mantener la autorización a través de sesiones.
5. En `/private` haremos un GET al microservicio https://ha-auth.now.sh/private el cual solo responderá OK 200 ante la presencia del token en el header, y mostramos el mensaje secreto 👀 que nos responda la API.