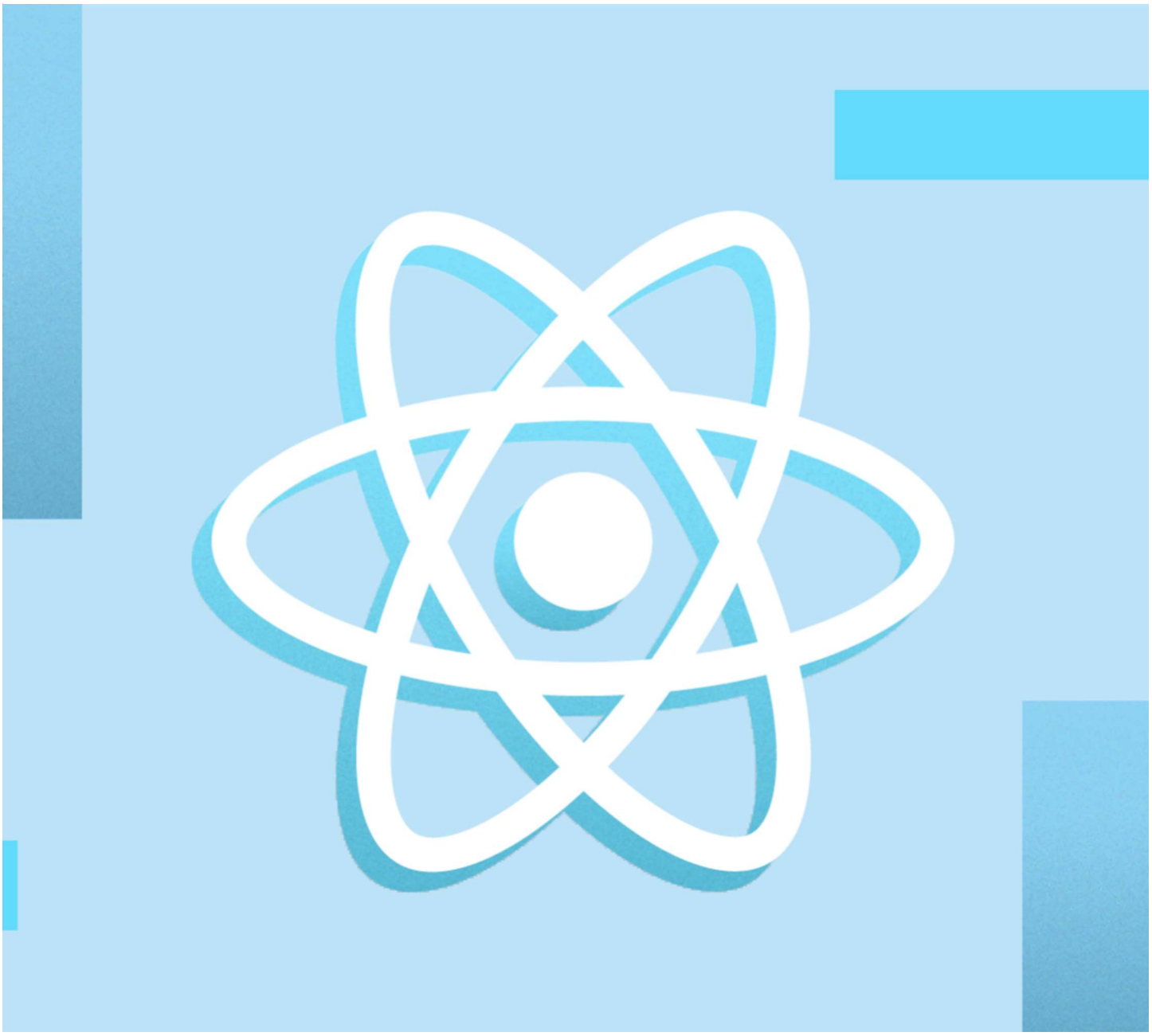


REACT

The Complete Guide to React User Authentication with Auth0

Learn how to add user authentication to React using Context and Hooks

Dan Arias
R&D Content Engineer

Last Updated On: November 02, 2020

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

> Look for the 🛠 emoji if you'd like to skim through the content while focusing on the build steps.

The focus of this tutorial is to help developers learn how to secure a React application by implementing user authentication. You'll enhance a starter React application to practice the following security concepts:

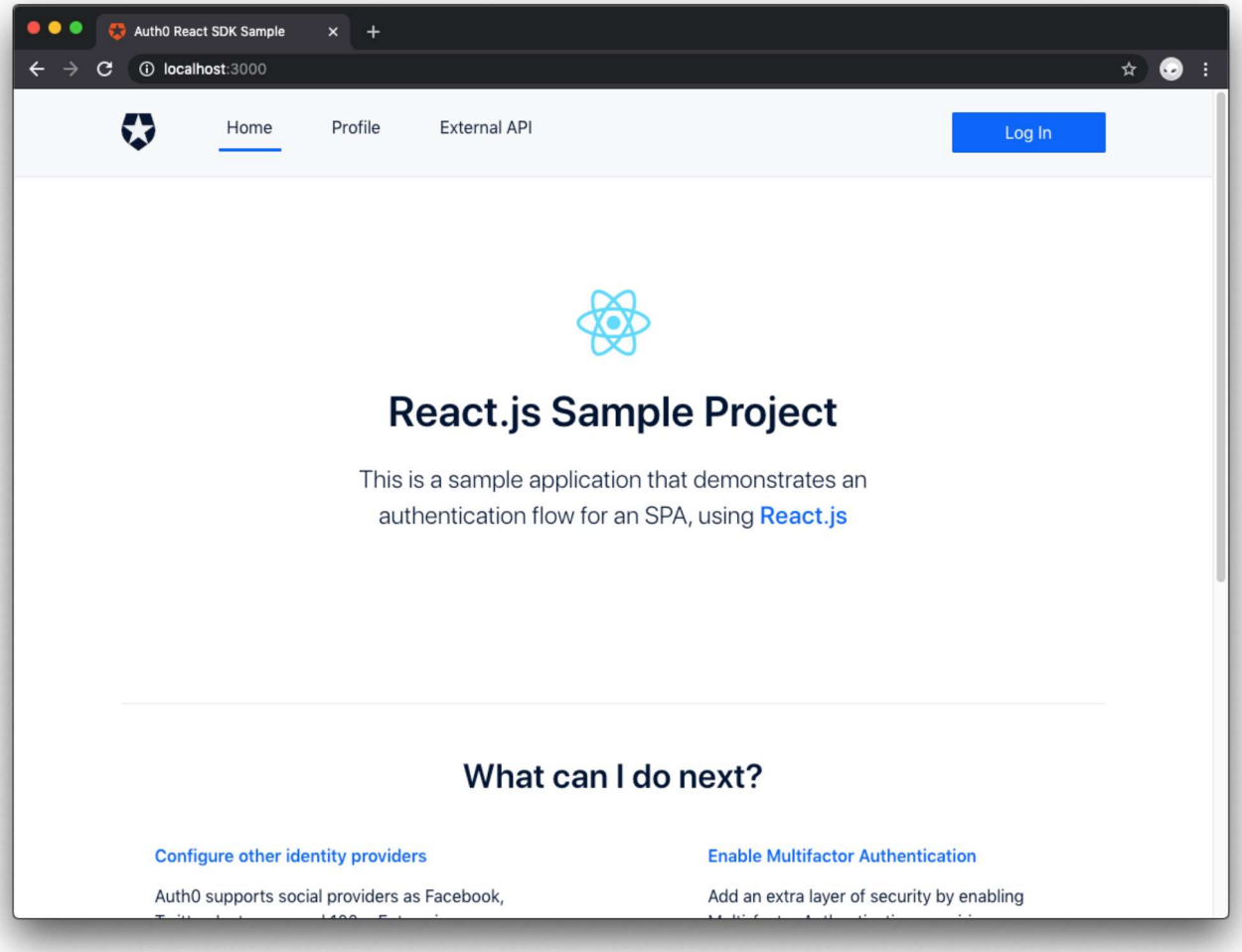
- Add user login and logout.
- Retrieve user information.
- Protect application routes.
- Call protected endpoints from an API.

This guide uses the Auth0 React SDK to secure React applications, which provides React developers with an easier way to **add user authentication to React applications using a hooks-centric approach.** The Auth0 React SDK provides a high-level API to handle a lot of authentication implementation details. You can now secure your React applications using security best practices while writing less code.

⚠️ This guide uses React Hooks and function components to build a secure React application. **If you need to implement any component from this guide using JavaScript classes**, check out the `auth0-react-sample-classes` repo as you read along. There is an equivalent `class`-based file for every file created in this guide.

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1



How does Auth0 work?

With the help of Auth0, **you don't need to be an expert on identity protocols, such as OAuth 2.0 or OpenID Connect, to understand how to secure your web application stack.** You first integrate your application with Auth0. Your application will then redirect users to an Auth0 customizable login page when they need to log in. Once your users log in successfully, Auth0 redirects them back to your app, returning JSON Web Tokens (JWTs) with their authentication and user information.

> ⏰⚡ If you are short of time, check out the Auth0 React Quickstart to get up and running with user authentication for React in just a few minutes. You may also check out our React and Auth0 YouTube Playlist.

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!
1

Get the Starter Application

We have created a starter project using `create-react-app` to help you learn React security concepts through hands-on practice. The starter application uses Bootstrap with a custom theme to take care of the styling and layout of your application. You can focus on building React components to secure your application.

🛠 As such, clone the `auth0-react-sample` repository on its `starter` branch to get started:

```
git clone -b starter git@github.com:auth0-blog/auth0-react-sample.git
```

🛠 Once you clone the repo, make `auth0-react-sample` your current directory:

```
cd auth0-react-sample
```

🛠 Install the React project dependencies:

```
npm install
```

Connect React with Auth0

The best part of the Auth0 platform is how strean g t
steps:

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

Sign up and create an Auth0 Application

If you haven't already, **sign up for a free Auth0 account →**

A free account offers you:

- 7,000 free active users and unlimited logins.
- Auth0 Universal Login for Web, iOS & Android.
- Up to 2 social identity providers like Google, GitHub, and Twitter.
- Unlimited Serverless Rules to customize and extend Auth0's capabilities.

During the sign-up process, you create something called an **Auth0 Tenant**, representing the product or service to which you are adding authentication.

🛠 Once you sign in, Auth0 takes you to the Dashboard. In the left sidebar menu, click on "Applications".

🛠 Then, click the "Create Application" button. A modal opens up with a form to provide a name for the application and choose its type.

- **Name:**

```
Auth0 React Sample
```

- **Application Type:** Single Page Web Applications

🛠 Click the "Create" button to complete the process. Your Auth0 application page loads up.

In the next step, you'll learn how to help React and Auth0 communicate.

▶ **What's the relationship between Auth0 Tenants and Auth0 Applications?**

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

Create a communication bridge between React and Auth0

When you use Auth0, you don't have to build login forms. Auth0 offers a Universal Login page to reduce the overhead of adding and managing authentication.

How does Universal Login work?

Your React application will redirect users to Auth0 whenever they trigger an authentication request. Auth0 will present them with a login page. Once they log in, Auth0 will redirect them back to your React application. For that redirecting to happen securely, you must specify in your **Auth0 Application Settings** the URLs to which Auth0 can redirect users once it authenticates them.

🛠 As such, click on the "Settings" tab of your Auth0 Application page and fill in the following values:

🛠 Allowed Callback URLs

```
http://localhost:4040
```

The above value is the URL that Auth0 can use to redirect your users **after they successfully log in.**

🛠 Allowed Logout URLs

```
http://localhost:4040
```

The above value is the URL that Auth0 can use to redirect your users **after they log out.**

🛠 Allowed Web Origins

```
http://localhost:4040
```

Using the Auth0 React SDK, your React appli[illegible]od to an Auth0 URL to handle authentication requests. As[illegible]app origin URL to avoid Cross-Origin Resource Sharin[illegible]

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

🛠 **Scroll down and click the "Save Changes" button.**

🛠 **Do not close this page yet**. You'll need some of its information in the next section.

Add the Auth0 configuration variables to React

From the Auth0 Application Settings page, you need the Auth0 Domain and Client ID values to allow your React application to use the communication bridge you created.

▶ **What exactly is an Auth0 Domain and an Auth0 Client ID?**

🛠 Open the React starter project, `auth0-react-sample`, and create a `.env` file under the project directory:

```
touch .env
```

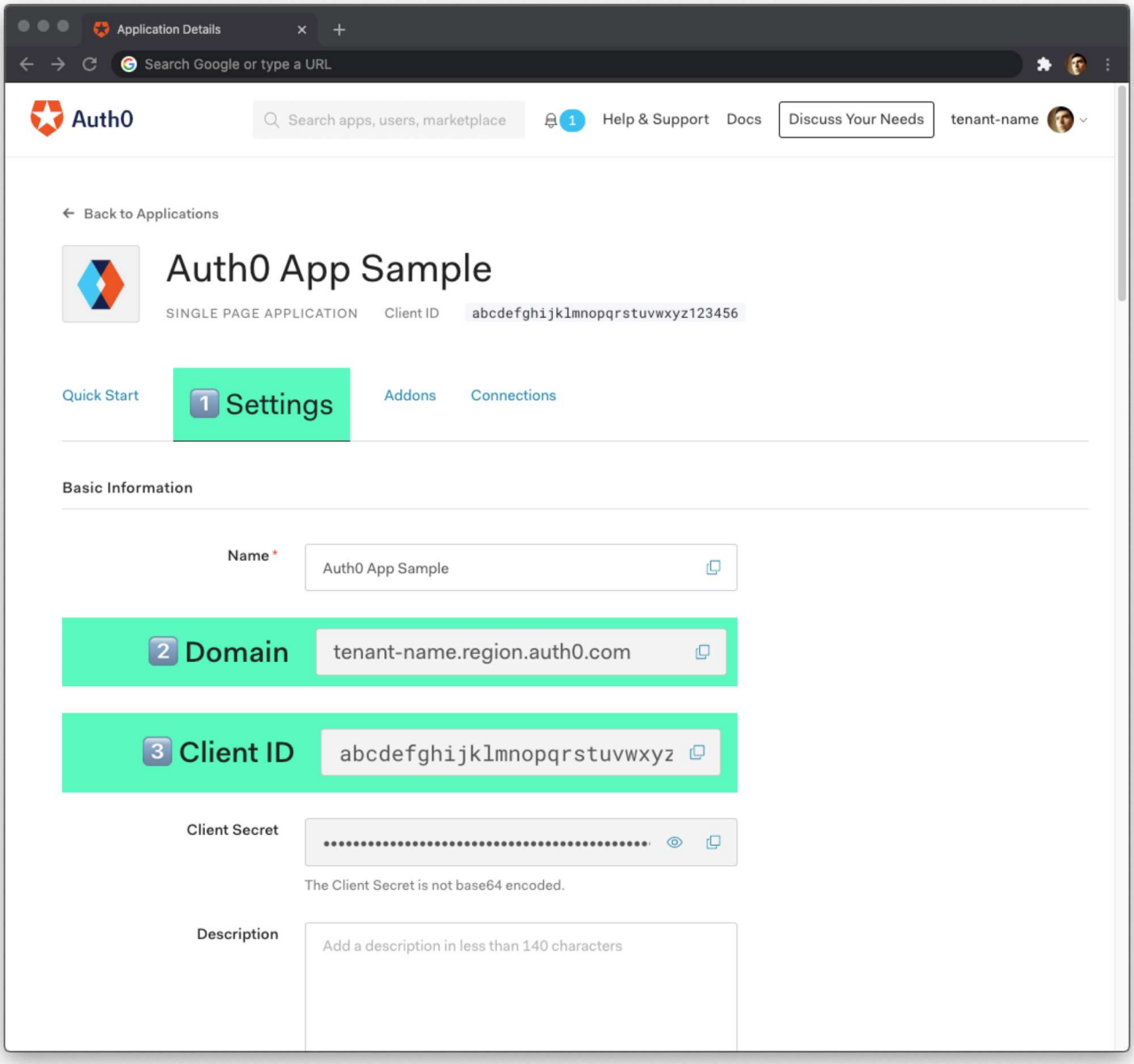
🛠 Populate `.env` as follows:

```
REACT_APP_AUTH0_DOMAIN=
REACT_APP_AUTH0_CLIENT_ID=
```

🛠 Head back to your Auth0 application page. Follow these steps to get the `REACT_APP_AUTH0_DOMAIN` and `REACT_APP_AUTH0_CLIENT_ID` values:

Psst! 👏We have a quiz to determine if Auth0 works for your identity needs!

1



1. 🛠 Click on the "Settings" tab, if you haven't already.

2. 🛠 Use the "Domain" value from the "Settings" as the value of `REACT_APP_AUTH0_DOMAIN` in `.env`.

3. 🛠 Use the "Client ID" value from the "Settings" as the value of `REACT_APP_AUTH0_CLIENT_ID` in `.env`.

These variables let your React application identify itself as an authorized party to interact with the Auth0 authentication server.

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

Auth0 and React connection se

You have completed setting up an authentication service that your React application can consume. All that is left is for you to continue building up the starter project throughout this guide by implementing components to trigger and manage the authentication flow.

Feel free to dive deeper into the Auth0 Documentation to learn more about how Auth0 helps you save time on implementing and managing identity.

Set Up the Auth0 React SDK

🛠 You need to follow these steps to integrate the Auth0 React SDK with your React application.

Install the Auth0 React SDK

🛠 Execute the following command:

```
npm install @auth0/auth0-react
```

Configure the Auth0Provider component

Under the hood, the Auth0 React SDK uses React Context. The SDK uses an `Auth0Context` component to manage the authentication state of your users. In turn, the SDK exposes the `Auth0Provider` component that provides that `Auth0Context` to its child components. As such, you can wrap your root component, such as `App`, with `Auth0Provider` to integrate Auth0 with your React app.

```
<Auth0Provider>
  <App />
</Auth0Provider>
```

However, user authentication is a mechanism to monitor who is accessing your application and control what they can do. For example, you can prevent users who have not logged in from accessing parts of your application. In that scenario, Auth0 can act as your *application bouncer*.

> A bouncer is a person employed by a … is … to prevent troublemakers from ente… th…

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

> premises. React security is not too different from nightclub security.

If users want to enter a protected route from your application, Auth0 will stop them and ask them to present their credentials. If Auth0 can verify who they are and that they are supposed to go in there, Auth0 will let them in. Otherwise, Auth0 will take them back to a public application route.

Now, it's important to reiterate that the authentication process won't happen within your application layer. Your React application will redirect your users to the Auth0 Universal Login page, where Auth0 asks for credentials and redirects the user back to your application with the result of the authentication process.

The `Auth0Provider` remembers where the user wanted to go and, if authentication were successful, it takes the user to that route. As such, the `Auth0Provider` **needs to have access to the session history of the application**. The starter React app uses React Router to manage its routing. React Router exposes a React Hook that makes it easy for you to access the session history through a `history` object, `useHistory()`.

Consequently, you need to wrap the `Auth0Provider` with `BrowserRouter` from React Router, which uses a `RouterContext.Provider` component under the hood to maintain routing state:

```
<BrowserRouter>
  <Auth0Provider>
    <App />
  </Auth0Provider>
</BrowserRouter>
```

Here, what you see at play is a pillar of React's architecture: you extend components, not through inheritance but composition.

How do you create an `Auth0Provider` with access to the application session history?

🛠 Start by creating an `auth` directory under

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
mkdir src/auth
```

🛠 Create an `auth0-provider-with-history.js` file under the `src/auth` directory to define an `Auth0ProviderWithHistory` component, which uses composition to make React Router Hooks available to `Auth0Provider`:

```
touch src/auth/auth0-provider-with-history.js
```

🛠 Populate `src/auth/auth0-provider-with-history.js` with the following:

```
// src/auth/auth0-provider-with-history.js

import React from 'react';
import { useHistory } from 'react-router-dom';
import { Auth0Provider } from '@auth0/auth0-react';

const Auth0ProviderWithHistory = ({ children }) => {
  const domain = process.env.REACT_APP_AUTH0_DOMAIN;
  const clientId = process.env.REACT_APP_AUTH0_CLIENT_ID;

  const history = useHistory();

  const onRedirectCallback = (appState) => {
    history.push(appState?.returnTo || window.location.pathname);
  };

  return (
    <Auth0Provider
      domain={domain}
      clientId={clientId}
      redirectUri={window.location.origin}
      onRedirectCallback={onRedirectCallback}
    >
      {children}
    </Auth0Provider>
  );
};

export default Auth0ProviderWithHistory;
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

What is happening within `Auth0ProviderWithHistory`?

- You need the Auth0 React SDK to connect with the correct Auth0 Application to process authentication. As such, you need to Auth0 Domain and Client ID to configure the `Auth0Provider`.

- You use the `onRedirectCallback()` method to handle the event where Auth0 redirects your users from the Auth0 Universal Login page to your React application. You use the `useHistory()` hook to get the `history` object from React Router. You use the `history.push()` method to take users back to the route they intended to access before authentication.

That's it! Wrapping any component tree with `Auth0ProviderWithHistory` will give it access to the `Auth0Context`.

How do you use `Auth0ProviderWithHistory`?

The `Auth0ProviderWithHistory` requires the `BrowserRouter` component from React Router to be its parent, grandparent, or great-great-great-grandparent.

The `Context` from React Router must be present in the component tree at a higher level for `Auth0ProviderWithHistory` to access the `useHistory()` hook from React Router.

🛠 Open `src/index.js` and update it as follows to build the proper component tree to power the routing and user authentication features of your React application:

```
// src/index.js

import React from 'react';
import ReactDOM from 'react-dom';
import App from './app';
import { BrowserRouter as Router } from 'react-router-dom';
import Auth0ProviderWithHistory from './au

import './index.css';
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
ReactDOM.render(
  <Router>
    <Auth0ProviderWithHistory>
      <App />
    </Auth0ProviderWithHistory>
  </Router>,
  document.getElementById('root'),
);
```

🛠 Execute the following command to run your React application:

```
npm start
```

The Auth0 React SDK is all set up. You are ready to implement user authentication in the next section.

Add User Authentication

You need to provide UI elements for your users to trigger authentication events: login, logout, and sign up.

Create a login button

🛠 Create a `login-button.js` file under the `src/components/` directory:

```
touch src/components/login-button.js
```

🛠 Populate `src/components/login-button.js` like so:

```
// src/components/login-button.js

import React from 'react';
import { useAuth0 } from '@auth0/auth0-react';

const LoginButton = () => {
  const { loginWithRedirect } = useAuth0();
  return (
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
      <button
        className="btn btn-primary btn-block"
        onClick={() => loginWithRedirect()}
      >
        Log In
      </button>
    );
};

export default LoginButton;
```

`loginWithRedirect()` is a method exposed by the `Auth0Context`. Calling this method prompts a user to authenticate and provide consent for your React application to access certain data on behalf of that user. In your current architecture, this means that your React application redirects the user to the Auth0 Universal Login page to carry out the authentication process. You'll see this in action in the next sections.

You can pass a configuration object to `loginWithRedirect()` to customize the login experience. For example, you can pass options to redirect users to an Auth0 Universal Login page optimized for signing up for your React application. See `RedirectLoginOptions` for more details on these options.

Create a sign-up button

You can make users land directly on a sign-up page instead of a login page by specifying the `screen_hint=signup` property in the configuration object of `loginWithRedirect()`:

```
{
  screen_hint: "signup",
}
```

🛠 Create a `signup-button.js` file under the `src/components/` directory:

```
touch src/components/signup-button.js
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

🛠 Populate `src/components/signup-button.j` component:

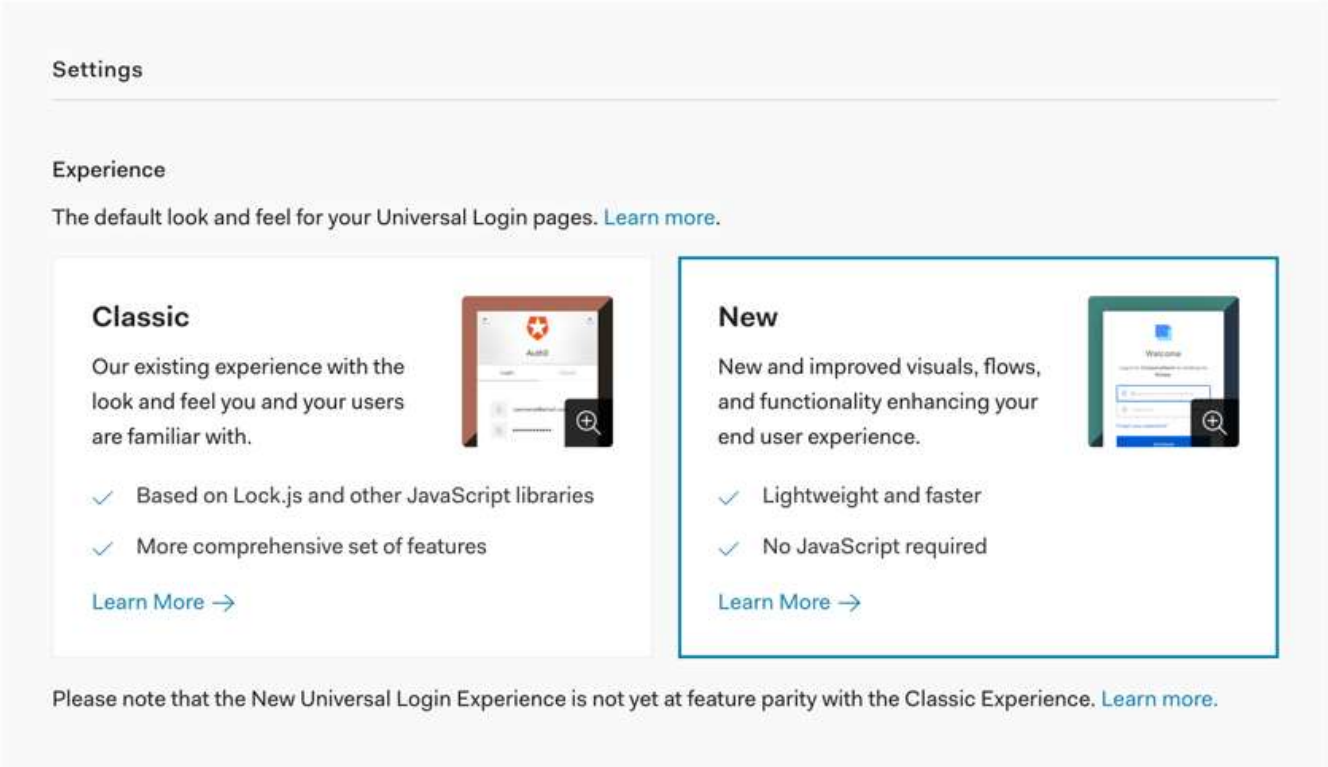
```
// src/components/signup-button.js

import React from 'react';
import { useAuth0 } from '@auth0/auth0-react';

const SignupButton = () => {
  const { loginWithRedirect } = useAuth0();
  return (
    <button
      className="btn btn-primary btn-block"
      onClick={() =>
        loginWithRedirect({
          screen_hint: 'signup',
        })
      }
    >
      Sign Up
    </button>
  );
};

export default SignupButton;
```

Using the Signup feature requires you to enable the Auth0 New Universal Login Experience in your tenant.

🛠 Open the Universal Login section of the Auth0 Dashboard and choose the "New" option under the "Experience" subsection.

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1



🛠 **Scroll down and click on the "Save Changes" button.**

The difference between the `LoginButton` and `SignupButton` user experience will be more evident once you integrate those components with your React application and see them in action. You'll do that in the next sections.

Create a logout button

🛠 Create a `logout-button.js` file under the `src/components/` directory:

```
touch src/components/logout-button.js
```

Populate `src/components/logout-button.js` like so:

```
// src/components/logout-button.js

import React from 'react';
import { useAuth0 } from '@auth0/auth0-r
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

```
const LogoutButton = () => {
  const { logout } = useAuth0();
```

```
  return (
    <button
      className="btn btn-danger btn-block"
      onClick={() =>
        logout({
          returnTo: window.location.origin,
        })
      }
    >
      Log Out
    </button>
  );
};

export default LogoutButton;
```

The `logout()` method exposed by `Auth0Context` clears the application session and redirects to the Auth0 `/v2/logout` endpoint to clear the Auth0 session. As with the login methods, you can pass an object argument to `logout()` to define parameters for the `/v2/logout` call. This process is fairly invisible to the user. See `LogoutOptions` for more details.

Here, you pass the `returnTo` option to specify the URL where Auth0 should redirect your users after they logout. Right now, you are working locally, and your Auth0 application's "Allowed Logout URLs" point to `http://localhost:4040`.

However, **if you were to deploy your React application to production**, you need to add the production logout URL to the "Allowed Logout URLs" list and ensure that Auth0 redirects your users to that production URL and not `localhost`. Setting `returnTo` to `window.location.origin` will do just that.

> **Read more about how Logout works at Auth0.**

Integrate the login and logout b

Let's wrap the `LoginButton` and `LogoutButtor` `AuthenticationButton`.

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

🛠 Create an `authentication-button.js` file under the `src/components/` directory:

```
touch src/components/authentication-button.js
```

🛠 Populate `src/components/authentication-button.js` with the following code:

```
// src/components/authentication-button.js

import React from 'react';

import LoginButton from './login-button';
import LogoutButton from './logout-button';

import { useAuth0 } from '@auth0/auth0-react';

const AuthenticationButton = () => {
  const { isAuthenticated } = useAuth0();

  return isAuthenticated ? <LogoutButton /> : <LoginButton />;
};

export default AuthenticationButton;
```

`isAuthenticated` is a boolean value exposed by the `Auth0Context`. Its value is `true` when Auth0 has authenticated the user and `false` when it hasn't.

There are some advantages to using this `AuthenticationButton` component wrapper:

You can build flexible interfaces. `AuthenticationButton` serves as a "log in/log out" switch that you can put anywhere you need that switch functionality. However, you still have separate `LoginButton` and `LogoutButton` components for cases when you need their functionality in isolation. For example, you may have a `LogoutButton` on a page that only authenticated users can see.

You can build extensible interfaces. You can eas ne the `SignupButton` component in `AuthenticationButton` to create a "sign up/log out"

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

switch. You could also wrap the "sign up/log out" switch in a `NewAuthenticationButton` component.

You can build declarative interfaces. Using `AuthenticationButton`, you can add login and logout functionality to your `NavBar` component, for example, without thinking about the implementation details of how the authentication switch works.

🛠 With that in mind, create an `auth-nav.js` file under the `src/components/` directory:

```
touch src/components/auth-nav.js
```

🛠 Populate `src/components/auth-nav.js` like so:

```
// src/components/auth-nav.js

import React from 'react';
import AuthenticationButton from './authentication-button';

const AuthNav = () => (
  <div className="navbar-nav ml-auto">
    <AuthenticationButton />
  </div>
);

export default AuthNav;
```

🛠 Finally, open `nav-bar.js` under the `src/components/` directory and update it like so:

```
// src/components/nav-bar.js

import React from 'react';

import MainNav from './main-nav';
import AuthNav from './auth-nav';

const NavBar = () => {
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

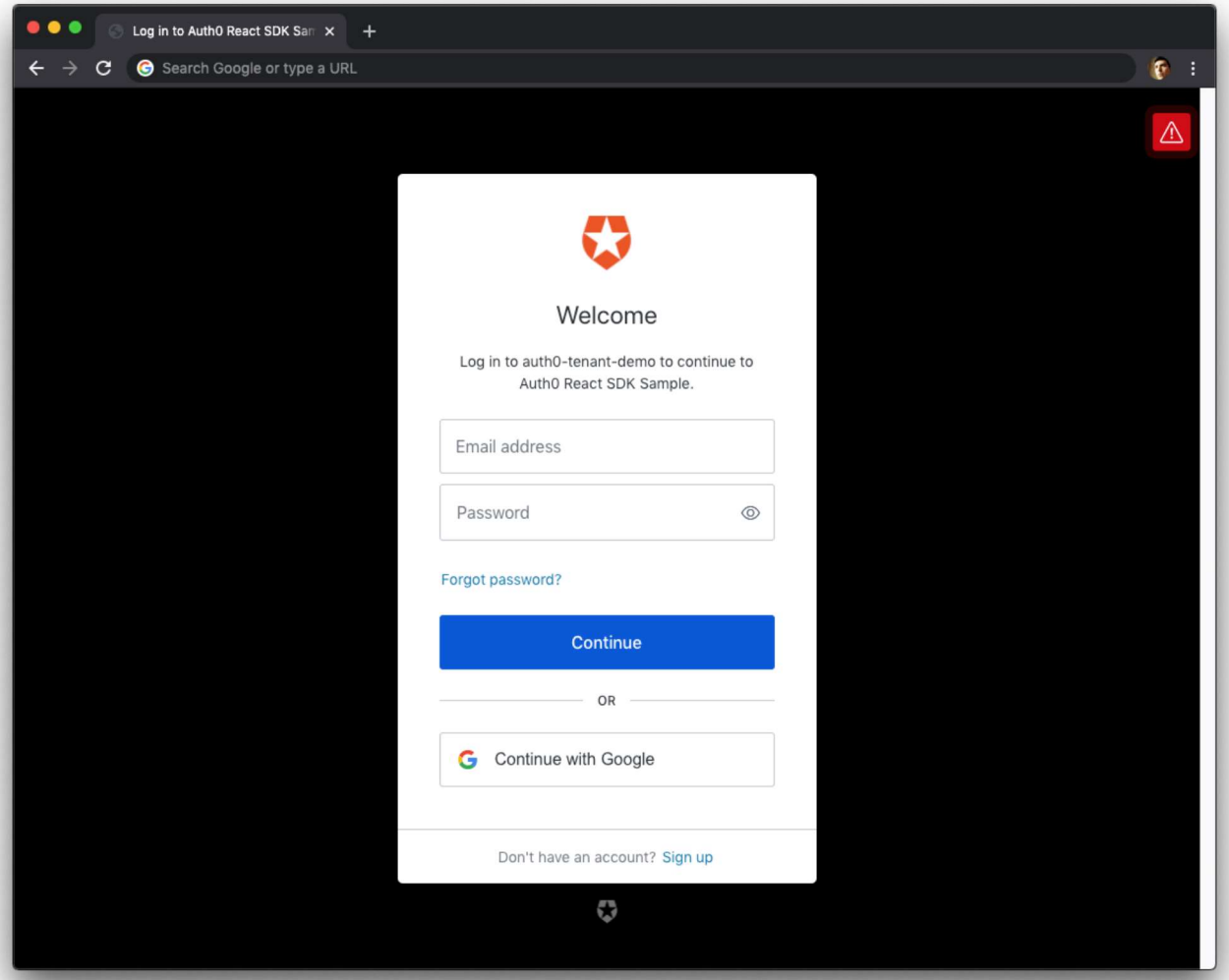
```
  return (
    <div className="nav-container mb-3">
      <nav className="navbar navbar-expand-md navbar-light bg-light">
        <div className="container">
          <div className="navbar-brand logo" />
          <MainNav />
          <AuthNav />
        </div>
      </nav>
    </div>
  );
};

export default NavBar;
```

By having different types of navigation bar subcomponents, you can extend each as you need without reopening and modifying the main `NavBar` component.

🛠 Go ahead and try to log in. Your React application redirects you to the Auth0 Universal Login page. You can use a form to log in with a username and password or a social identity provider like Google. Notice that this login page also gives you the option to sign up.

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1



▶

Experiment: Use the `SignupButton` component

> You can customize the appearance of New Universal Login pages. You can also override any text in the New Experience using the Text Customization API.

Notice that when you finish logging in and Auth0 redirects you to your React app, the login button briefly shows up (blue color), and then the logout button renders (red color).

The user interface flashes because your Reac p ticated the user yet. Your app will know the user authenticati OK

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!
1

🛠 To fix that UI flashing, use the `isLoading` boolean value exposed by the `Auth0Context` to render the `App` component once the Auth0 React SDK has finished loading.

🛠 Open `src/app.js` and update it as follows:

```
// src/app.js

import React from 'react';
import { Route, Switch } from 'react-router-dom';
import { useAuth0 } from '@auth0/auth0-react';

import { NavBar, Footer, Loading } from './components';
import { Home, Profile, ExternalApi } from './views';

import './app.css';

const App = () => {
  const { isLoading } = useAuth0();

  if (isLoading) {
    return <Loading />;
  }

  return (
    <div id="app" className="d-flex flex-column h-100">
      <NavBar />
      <div className="container flex-grow-1">
        <Switch>
          <Route path="/" exact component={Home} />
          <Route path="/profile" component={Profile} />
          <Route path="/external-api" component={ExternalApi} />
        </Switch>
      </div>
      <Footer />
    </div>
  );
};
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
export default App;
```

While the SDK is loading, the `Loading` component, which has a cool animation, renders.

Retrieving User Information

After a user successfully logs in, Auth0 sends an ID token to your React application. Authentication systems, such as Auth0, use ID Tokens in token-based authentication to cache user profile information and provide it to a client application. The caching of ID tokens can contribute to improvements in performance and responsiveness for your React application.

You can use the data from the ID token to personalize the user interface of your React application. The Auth0 React SDK decodes the ID token and stores its data in the `user` object exposed by the `Auth0Context`. Some of the ID token information includes the name, nickname, picture, and email of the logged-in user.

How can you use the ID token to create a profile page for your users?

🛠 Update the `Profile` component in `src/views/profile.js` as follows:

```
// src/views/profile.js

import React from 'react';

import { useAuth0 } from '@auth0/auth0-react';

const Profile = () => {
  const { user } = useAuth0();
  const { name, picture, email } = user;

  return (
    <div>
      <div className="row align-items-cent
        <div className="col-md-2 mb-3">
          <img
            src={picture}
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
            alt="Profile"
            className="rounded-circle img-fluid profile-picture mb-3 mb-md-0"
          />
        </div>
        <div className="col-md text-center text-md-left">
          <h2>{name}</h2>
          <p className="lead text-muted">{email}</p>
        </div>
      </div>
      <div className="row">
        <pre className="col-12 text-light bg-dark p-4">
          {JSON.stringify(user, null, 2)}
        </pre>
      </div>
    </div>
  );
};

export default Profile;
```

What's happening within the `Profile` component?

- You destructure the `user` object to obtain the user `name`, `picture`, and `email`.
- You then display these three properties in the user interface. Since the data comes from a simple object, you don't have to fetch it using any asynchronous calls.
- Finally, you display the full content of the decoded ID token within a code box. You can now see all the other properties available for you to use.

The `Profile` component renders user information that you could consider protected. Additionally, the `user` property is `null` if there is no logged-in user. So either way, this component should only render if Auth0 has authenticated the user.

As such, you should protect the route that render
`http://localhost:4040/profile`. You'll learn

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

Protecting Routes

The Auth0 React SDK exposes a `withAuthenticationRequired` Higher-Order Component (HOC) that you can use to protect routes. You can also use `withAuthenticationRequired` to create a `ProtectedRoute` component to protect routes in a more declarative way using React Router.

Use a Higher-Order Component to protect a route

🛠 Open `src/views/profile.js` and update it as follows:

```
// src/views/profile.js

import React from 'react';

import { useAuth0, withAuthenticationRequired } from '@auth0/auth0-react';
import { Loading } from '../components';

const Profile = () => {
  const { user } = useAuth0();
  const { name, picture, email } = user;

  return (
    <div>
      <div className="row align-items-center profile-header">
        <div className="col-md-2 mb-3">
          <img
            src={picture}
            alt="Profile"
            className="rounded-circle img-fluid profile-picture mb-3 mb-md-0"
          />
        </div>
        <div className="col-md text-center text-md-left">
          <h2>{name}</h2>
          <p className="lead text-muted">{
        </div>
      </div>
      <div className="row">
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
        <pre className="col-12 text-light bg-dark p-4">
          {JSON.stringify(user, null, 2)}
        </pre>
      </div>
    </div>
  );
};

export default withAuthenticationRequired(Profile, {
  onRedirecting: () => <Loading />,
});
```

When you wrap your components in the `withAuthenticationRequired` Higher-Order Component and users who have not logged in visit a page that renders that component, your React application will redirect that user to the login page. After the user logs in, Auth0 will redirect the user to your React application, and the `Auth0Provider` will take the users to the page they intended to access before login.

`withAuthenticationRequired` takes the following arguments:

- The component that you want to protect.
- A configuration object to customize the authentication flow, `WithAuthenticationRequiredOptions`. This object takes the following optional properties:
  - `loginOptions`: It behaves exactly like the configuration options you can pass to `loginWithRedirect()` to customize the login experience.
  - `returnTo`: Lets you specify a path for React to redirect a user after the login transaction that the user triggered in this component completes.
  - `onRedirecting`: It renders a component **while your React application redirects the user to the login page.**

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

In the example above, users who have not logged in see the `Loading` component as soon they hit the `/profile` route:

```
export default withAuthenticationRequired(Profile, {
  onRedirecting: () => <Loading />,
});
```

The `onRedirecting` component improves the user experience by avoiding any flashing of mixed UI components (protected and public components).

🛠 Try this out. Log out and visit `http://localhost:4040/profile`. Your React application should redirect you to the Auth0 Universal Login page.

What if you are using React Router?

Using `withAuthenticationRequired` to wrap the component directly is not the most declarative way to build a React application. If you were to look at the routes defined in the `App` component, you wouldn't be able to tell which routes are protected and which routes are public.

```
const App = () => {
  const { isLoading } = useAuth0();

  if (isLoading) {
    return <Loading />;
  }

  return (
    <div id="app" className="d-flex flex-column h-100">
      <NavBar />
      <div className="container flex-grow-1">
        <Switch>
          <Route path="/" exact component={Home} />
          <Route path="/profile" compon t
          <Route path="/external-api" comp
        </Switch>
      </div>
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
      <Footer />
    </div>
  );
};
```

🛠 Instead of using `withAuthenticationRequired` directly, you can wrap it in a `ProtectedRoute` component that leverages the features of React Router.

Create a component to protect React Router paths

> **The starter application uses React Router as its routing library. This example applies only to that library.**

In this section, you'll create a `ProtectedRoute` component that uses the `Route` component from React Router to render the `withAuthenticationRequired` Higher-Order Component. The advantage of this approach is that your `ProtectedRoute` will have the same API as an out-of-the-box `Route` component. As such, you can compose `ProtectedRoute` with other React Router components organically.

🛠 To start, create a `protected-route.js` file under the `src/auth` directory:

```
touch src/auth/protected-route.js
```

🛠 Populate `src/auth/protected-route.js` as follows:

```
// src/auth/protected-route.js

import React from 'react';
import { Route } from 'react-router-dom';
import { withAuthenticationRequired } from '@auth0/auth0-react';
import { Loading } from '../components/index';

const ProtectedRoute = ({ component, ...ar
  <Route
    component={withAuthenticationRequired(
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
      onRedirecting: () => <Loading />,
    })}
    {...args}
  />
);

export default ProtectedRoute;
```

🛠 Finally, open the `src/app.js` file. Locate the `Switch` component and change the `Route` components for the `/profile` and `/external-api` paths to a `ProtectedRoute` component:

```
// src/app.js

import React from 'react';
import { Route, Switch } from 'react-router-dom';
import { useAuth0 } from '@auth0/auth0-react';

import { NavBar, Footer, Loading } from './components';
import { Home, Profile, ExternalApi } from './views';
import ProtectedRoute from './auth/protected-route';

import './app.css';

const App = () => {
  const { isLoading } = useAuth0();

  if (isLoading) {
    return <Loading />;
  }

  return (
    <div id="app" className="d-flex flex-column h-100">
      <NavBar />
      <div className="container flex-grow-1">
        <Switch>
          <Route path="/" exact component=
          <ProtectedRoute path="/profile"
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
          <ProtectedRoute path="/external-api" component={ExternalApi} />
        </Switch>
      </div>
      <Footer />
    </div>
  );
};

export default App;
```

You don't need to use the `withAuthenticationRequired` HOC directly in the `Profile` component any longer.

🛠 Open `src/views/profile.js` and revert the file to its previous content:

```
// src/views/profile.js

import React from 'react';

import { useAuth0 } from '@auth0/auth0-react';

const Profile = () => {
  const { user } = useAuth0();
  const { name, picture, email } = user;

  return (
    <div>
      <div className="row align-items-center profile-header">
        <div className="col-md-2 mb-3">
          <img
            src={picture}
            alt="Profile"
            className="rounded-circle img-fluid profile-picture mb-3 mb-md-0"
          />
        </div>
        <div className="col-md text-center
          <h2>{name}</h2>
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
          <p className="lead text-muted">{email}</p>
        </div>
      </div>
      <div className="row">
        <pre className="col-12 text-light bg-dark p-4">
          {JSON.stringify(user, null, 2)}
        </pre>
      </div>
    </div>
  );
};

export default Profile;
```

🛠 You can now test that these two paths require users to log in before they can access them. **Log out** and try to access the Profile or External API page. If it works, React redirects you to log in with Auth0.

▶ **Client-side guards improve the user experience of your React application, not its security.**

> **Which route protection strategy would you prefer to use in your React applications? The `withAuthenticationRequired` HOC or the `ProtectedRoute` component? Please, let me know in the comments below.**

Calling an API

This section focuses on showing you how to get an access token in your React application and how to use it to make API calls to protected API endpoints.

When you use Auth0, you delegate the authentication process to a centralized service. Auth0 provides you with functionality to log in and log o n. How , your application may need to access protected re

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

You can also protect an API with Auth0. There are multiple API quickstarts to help you integrate Auth0 with your backend platform.

When you use Auth0 *to protect your API*, you also delegate the authorization process to a centralized service that ensures only approved client applications can access protected resources on behalf of a user.

How can you make secure API calls from React?

Your React application authenticates the user and receives an access token from Auth0. The application can then pass that access token to your API as a credential. In turn, your API can use Auth0 libraries to verify the access token it receives from the calling application and issue a response with the desired data.

Instead of creating an API from scratch to test the authentication and authorization flows between the client and the server, you'll use a demo Express API that I've prepared for you.

Get the Express API demo

🛠 Open a new terminal window and clone the `auth0-express-js-sample` repo somewhere in your system. **Ensure that you clone it outside your React project directory.**

```
git clone git@github.com:auth0-blog/auth0-express-js-sample.git
```

🛠 Once you clone this repo, **make the `auth0-express-js-sample` directory your current directory**:

```
cd auth0-express-js-sample
```

🛠 Install the Node.js project dependencies:

```
npm install
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

Connect the Express API with A

Create a communication bridge between Express and Auth0

This process is similar to how you connected React with Auth0.

🛠 Head to the APIs section in the Auth0 Dashboard, and click the "Create API" button.

🛠 Then, in the form that Auth0 shows:

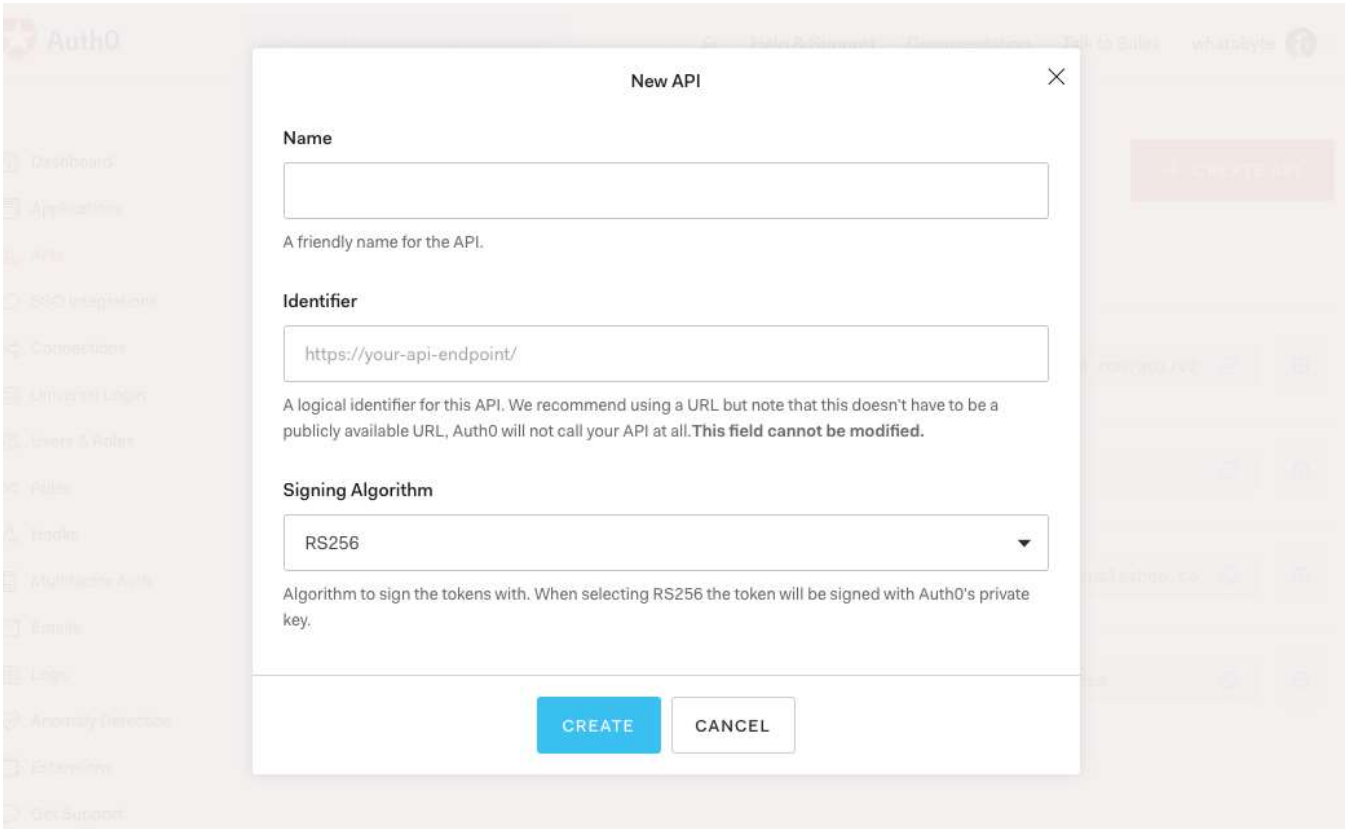
- Add a **Name** to your API:

```
Auth0 Express Sample
```

- Set its **Identifier** value:

```
https://express.sample
```

- Leave the signing algorithm as `RS256` as it's the best option from a security standpoint.



Identifiers are unique strings that help Auth0 differentiate between your different APIs. We recommend using URLs to facilitate creating uni , Auth0 **never calls these URLs.**

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!
1

🛠 With these values in place, hit the "Create" button. Keep this page open as you'll need some of its values in the next section.

Add the Auth0 configuration variables to Express

🛠 Create a `.env` file for the API Server under the `auth0-express-js-sample` directory:

```
touch .env
```

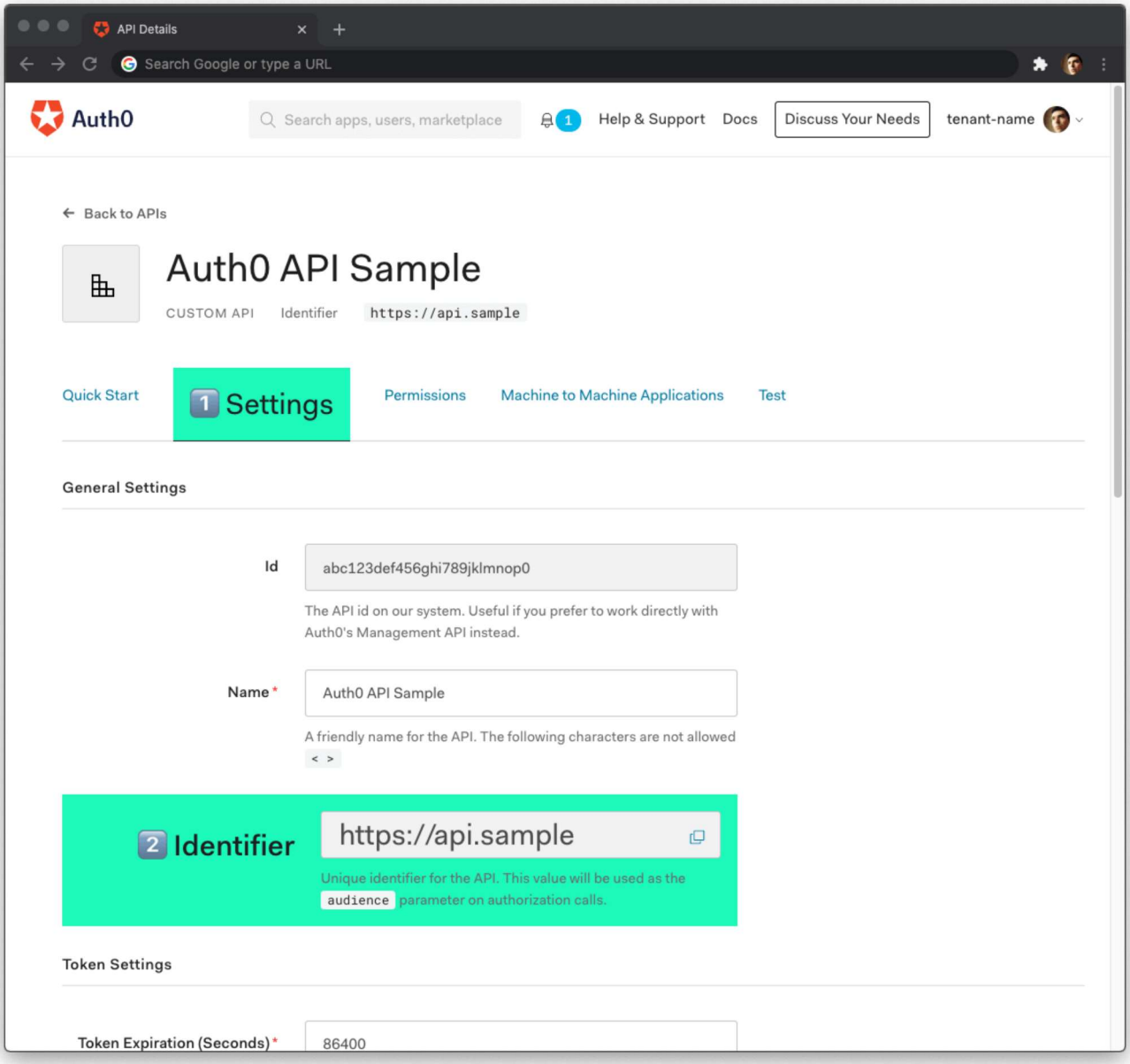
🛠 Populate this `auth0-express-js-sample/.env` file as follows:

```
SERVER_PORT=6060
CLIENT_ORIGIN_URL=http://localhost:4040
AUTH0_AUDIENCE=
AUTH0_DOMAIN=
```

🛠 Head back to your Auth0 API page, and **follow these steps to get the Auth0 Audience**:

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1



1. 🛠 Click on the **"Settings"** tab.

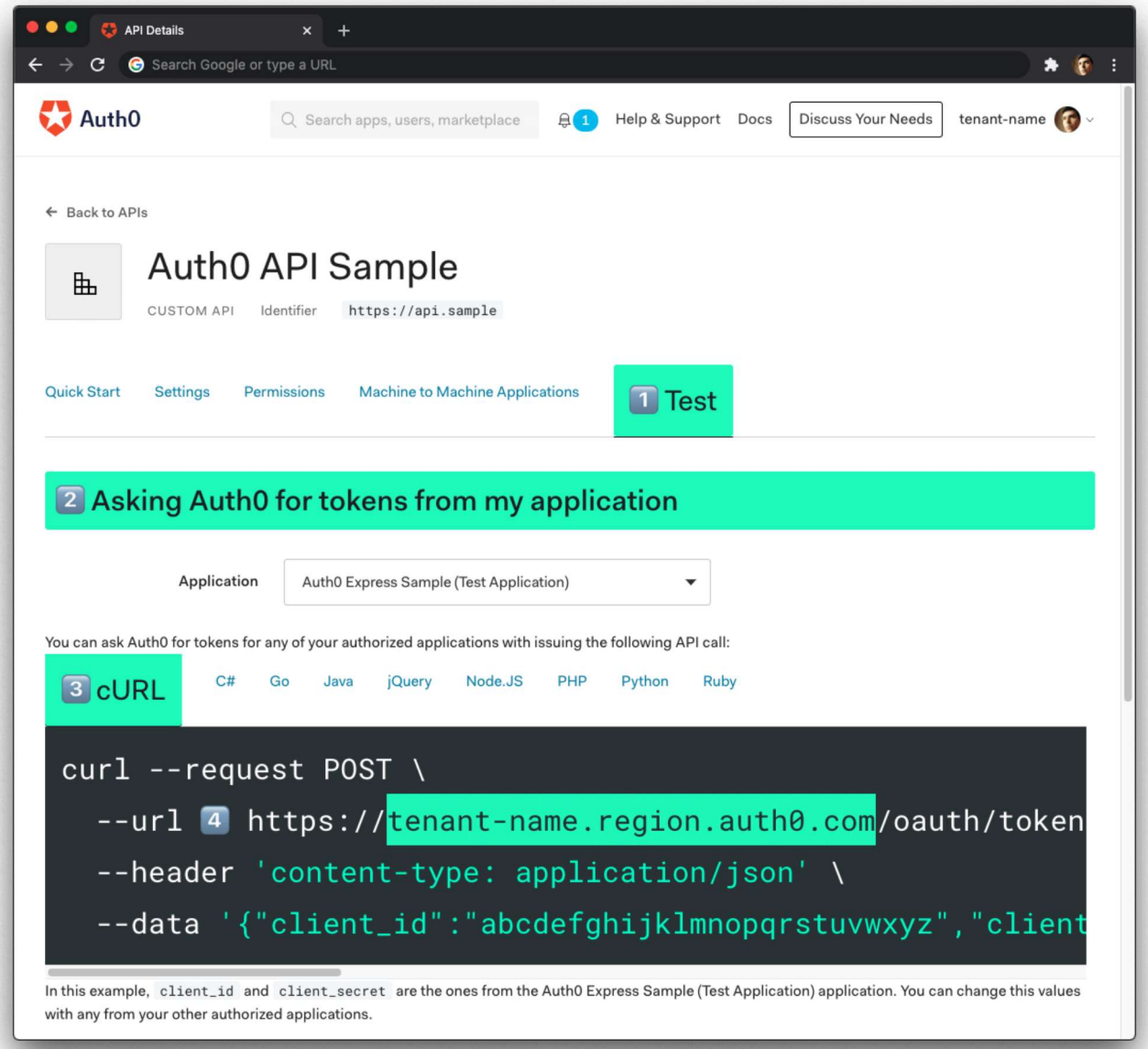
2. 🛠 Locate the **"Identifier"** field and copy its value.

3. 🛠 Paste the "Identifier" value as the value of `AUTH0_AUDIENCE` in `.env`.

Now, **follow these steps to get the Auth0 Domain value:**

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1



1. 🛠 Click on the **"Test"** tab.
2. 🛠 Locate the section called **"Asking Auth0 for tokens from my application"**.
3. 🛠 Click on the **cURL** tab to show a mock `POST` request.
4. 🛠 Copy your Auth0 domain, which is *part* of the `--url` parameter value: `tenant-name.region.auth0.com`.
5. 🛠 Paste the Auth0 domain value as the value of `AUTH0_DOMAIN` in `.env`.

▶ **Tips to get the Auth0 Domain**

🛠 With the `.env` configuration values set, run th ng command:

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

```
npm start
```

Configure React to connect with the Express API

🛠 Head back to the `auth0-react-sample` project directory that stores your React application.

🛠 Locate the `auth0-react-sample/.env` file and add your Auth0 Audience and Server URL values to it:

```
REACT_APP_AUTH0_DOMAIN=YOUR-AUTH0-DOMAIN
REACT_APP_AUTH0_CLIENT_ID=YOUR-AUTH0-APP-CLIENT-ID
REACT_APP_AUTH0_AUDIENCE=https://express.sample
REACT_APP_SERVER_URL=http://localhost:6060
```

🛠 The value of `REACT_APP_AUTH0_AUDIENCE` is the same as `AUTH0_AUDIENCE` from `auth0-express-js-sample/.env`.

▶ **Why do all variables in the React `.env` file start with `REACT_APP_`?**

Your React application needs to pass an access token when it calls a target API to access protected resources. You can request an access token in a format that the API can verify by passing the audience and scope props to `Auth0Provider`.

Any changes that you make to React environment variables require you to restart the development server if it is running.

🛠 **Restart your React application so that it can use the new values you've set in `auth0-react-sample/.env`.**

🛠 Update the `auth0-provider-with-histo` `sample/src/auth` directory to add the `audienc`

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
// src/auth/auth0-provider-with-history.js

import React from 'react';
import { useHistory } from 'react-router-dom';
import { Auth0Provider } from '@auth0/auth0-react';

const Auth0ProviderWithHistory = ({ children }) => {
  const history = useHistory();
  const domain = process.env.REACT_APP_AUTH0_DOMAIN;
  const clientId = process.env.REACT_APP_AUTH0_CLIENT_ID;
  const audience = process.env.REACT_APP_AUTH0_AUDIENCE;

  const onRedirectCallback = (appState) => {
    history.push(appState?.returnTo || window.location.pathname);
  };

  return (
    <Auth0Provider
      domain={domain}
      clientId={clientId}
      redirectUri={window.location.origin}
      onRedirectCallback={onRedirectCallback}
      audience={audience}
    >
      {children}
    </Auth0Provider>
  );
};

export default Auth0ProviderWithHistory;
```

Why is the Auth0 Audience value the same for both apps? Auth0 uses the value of the `audience` prop to determine which resource server (API) the user is authorizing your React application to access. It's like a phone number. You want to ensure that your React application "texts the right API".

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

The actions that your React application can perform on the API depend on the scopes that your access token contains, which you define as the value of a `scope` props in `Auth0Provider`.

Remember that screen you saw when you first logged in with Auth0 asking you for permission to access your profile information? Your React application will request authorization from the user to access the requested scopes, and the user will approve or deny the request. You may have seen something similar when sharing your contacts or photos from a social media platform with a third-party application.

When you don't pass a `scope` prop to `Auth0Provider` as in the example above, the React SDK defaults to the OpenID Connect Scopes: `openid profile email`.

- `openid`: This scope informs the Auth0 Authorization Server that the Client is making an OpenID Connect (OIDC) request to verify the user's identity. OpenID Connect is an authentication protocol.

- `profile`: This scope value requests access to the user's default profile information, such as `name`, `nickname`, and `picture`.

- `email`: This scope value requests access to the `email` and `email_verified` information.

The details of the OpenID Connect Scopes go into the ID Token. However, you can define custom API scopes to implement access control. You'll identify those custom scopes in the calls that your client applications make to that API. Auth0 includes API scopes in the access token as the `scope` claim value.

> **The concepts about API scopes or permissions are better covered in an Auth0 API tutorial such as "Use TypeScript to Create a Secure API with Node.js and Express: Role-Based Access Control".**

🛠 Update `src/views/external-api.js` as f

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
// src/views/external-api.js
```

```
import React, { useState } from 'react';
import { useAuth0 } from '@auth0/auth0-react';

const ExternalApi = () => {
  const [message, setMessage] = useState('');
  const serverUrl = process.env.REACT_APP_SERVER_URL;

  const { getAccessTokenSilently } = useAuth0();

  const callApi = async () => {
    try {
      const response = await fetch(`${serverUrl}/api/messages/public-message`);

      const responseData = await response.json();

      setMessage(responseData.message);
    } catch (error) {
      setMessage(error.message);
    }
  };

  const callSecureApi = async () => {
    try {
      const token = await getAccessTokenSilently();

      const response = await fetch(
        `${serverUrl}/api/messages/protected-message`,
        {
          headers: {
            Authorization: `Bearer ${token}`,
          },
        },
      );

      const responseData = await respon

      setMessage(responseData.message);
    } catch (error) {
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
      setMessage(error.message);
    }
  };

  return (
    <div className="container">
      <h1>External API</h1>
      <p>
        Use these buttons to call an external API. The protected API call has an
        access token in its authorization header. The API server will validate
        the access token using the Auth0 Audience value.
      </p>
      <div
        className="btn-group mt-5"
        role="group"
        aria-label="External API Requests Examples"
      >
        <button type="button" className="btn btn-primary" onClick={callApi}>
          Get Public Message
        </button>
        <button
          type="button"
          className="btn btn-primary"
          onClick={callSecureApi}
        >
          Get Protected Message
        </button>
      </div>
      {message && (
        <div className="mt-5">
          <h6 className="muted">Result</h6>
          <div className="container-fluid">
            <div className="row">
              <code className="col-12 text-light bg-dark p-4">{message}</code>
            </div>
          </div>
        </div>
      )}
```

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
    </div>
  );
};

export default ExternalApi;
```

What is happening now within the `ExternalApi` component?

- You add a `callApi()` method that performs a public API request.
- You add a `callSecureApi()` method that performs a secure API request as follows:
  - **(a) Get the access token from Auth0** using the `getAccessTokenSilently` method, which gets your React application a new access token under the hood without requiring the user to log in again.
  - **(b) Pass that access token** as a bearer credential in the authorization header of the request.
- You use the `useState()` React hook to update the user interface whenever any of the described API calls complete successfully.

Your previous login request did not include an audience parameter. As such, the React SDK doesn't have an access token stored in memory.

▶ **You should not store tokens in `localStorage`. Why?**

🛠 Log out and log back in to get a new access token from Auth0 that includes the audience information.

🛠 Visit `http://localhost:4040/external-api` to access the External API page to test the responses.

Psst! 👋 We have a quiz to determine if Auth0 works for your identity needs!

Get Public Message:

```
The API doesn't require an access token to share this message.
```

Get Protected Message:

```
The API successfully validated your access token.
```

Conclusion

You have implemented user authentication in React to identify your users, get user profile information, and control the content that your users can access by protecting routes and API resources.

This tutorial covered the most common authentication use case for a React application: simple login and logout. However, Auth0 is an extensible and flexible platform that can help you achieve even more. If you have a more complex use case, check out the Auth0 Architecture Scenarios to learn more about the typical architecture scenarios we have identified when working with customers on implementing Auth0.

In a follow-up guide, we'll cover advanced authentication patterns and tooling, such as using a pop-up instead of a redirect to log in users, adding permissions to ID tokens, using metadata to enhance user profiles, and much more.

Let me know in the comments below what you thought of this tutorial. Thank you for reading and stay tuned, please.

Dan Arias
R&D CONTENT ENGINEER

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

Howdy! 🤠 I do technology research at Auth0 with a focus on security and identity and develop apps to showcase the advantages or pitfalls of such technology. I also contribute to the development of our SDKs, documentation, and design systems, such as **Cosmos.**

The majority of my engineering work revolves around AWS, React, and Node, but my research and content development involves a wide range of topics such as Golang, performance, and cryptography. Additionally, I am one of the core maintainers of this blog. Running a blog at scale with **over 600,000 unique visitors per month** is quite challenging!

I was an Auth0 customer before I became an employee, and I've always loved how much easier it is to implement authentication with Auth0. Curious to try it out? **Sign up for a free account** ⚡.

VIEW PROFILE ▸

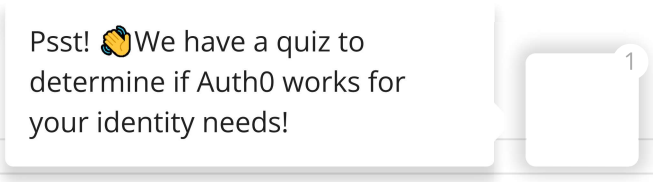
More like this

TYPESCRIPT

Use TypeScript to Create a Secure API with Node.js and Express: Getting Started

AUTH0

Auth0 Features to Fall in Love With



NODE

Node.js and Express Authentication Using Passport

Follow the conversation

Powered by the Auth0 Community. Sign up now to join the discussion. **Community links will open in a new window.**

CONTINUE DISCUSSION 160 replies

avala ↩ Reply to comment

This is, by far, the most comprehensive guide to getting Auth0 set up on your React project. I would have greatly appreciated this link being included in the Quick Start React guide when I was getting started.

I ran into one problem however where components making an API call with useEffect create an infinitie loop only when using Private Routes. I have a form component protected by the Private Route as setup in this guide. The form component has a useEffect hook with an empty array dependency to only run on componentDidMount. This useEffect hook calls a basic Axios.get function to populate a global context component which in turn populates our form fields. This was working before adding the Private Routes, and continues to work if using a normal route. Under the private route, the browser begins to chug and throws:

```
Uncaught (in promise) Error: Maximum update depth exceeded. This can happen when a component repeatedly calls setState inside componentWillUpdate or componentDidUpdate. React limits the number of nested updates to prevent infinite loops.
```

Is there a better way to construct this? Is it pos e? I've combed through the many different forms of R to Private routes with API calls. This seems like something that should be easy to setup.

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1 reply

dan-auth0 Auth0 Employee ▶ avala ↩ Reply to comment

Howdy, Avala. Thank you for joining the Auth0 Community and for your feedback 🙌

If I understand correctly, the example that you'd like to see is as follows:

- You have a component that makes API calls within a `useEffect()` hook.
- The component stores the response from the API in a global Context component.
- You limit access to the component by wrapping it up in a `PrivateRoute` Higher-Order Component.

Is that right? If so, I definitely have planned to provide an example of this in the close future.

Where it may get tricky is the following:

> This useEffect hook calls a basic Axios.get function to populate a global context component which in turn populates our form fields.

This is more of an implementation detail of state management that could be done in different ways and could be the source of the performance error. For example, I default to Formik when I need to use forms in React. I have an app that has private routes to gate components with forms in them but I have not experienced an error like the one you pointed out yet, which makes me think the source of the error is not related to the presence of the `PrivateRoute` per se but something else 🤔

2 replies

AV

avala ▶ dan-auth0 ↩ Reply to comment

@dan-auth0 Thanks for the quick response on a Friday! It looks like you have the scenario correct. I've tinkered with the app further and it seems like the issue stems from attempting to manipulate the Global State context. We're using Formik as well, but have some funky use-cases with a material stepper and a repeating table that makes this project interesting.

After further testing, I can run a fetch request and update local state with expected behavior, but when we attempt to read or edit the global state context (in useEffect or a callback) the stuff hits the fan again. Long story short, attempting to dispatch to the Global State Reducer generates an infinite loop when using Private Routes, but normal Routes work fine

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

```
const authReducer = (state, action) => {
    switch (action.type) {
        case "FETCH_FIELDS_REQUEST":
            return {
                ...state,
                isFetching: true
```

```
            isFetching: true,
            hasError: false,
        };
    case "FETCH_FIELDS_SUCCESS":
        return {
            ...state,
            isFetching: false,
            accessoryValues: action.payload.Accessory.sort(compareValues('Name')),
            injectorValues: action.payload.Injector.sort(compareValues('Name')),
            nozzleValues: action.payload.Nozzle.sort(compareValues('Name')),
            fieldsRetrieved: true
        };
    case "FETCH_FIELDS_FAILURE":
        return {
            ...state,
            hasError: true,
            isFetching: false
        };
        case "UPDATE_FORM":
            return {
                ...state,
                ...action.payload
            }
    default:
        return state;
  }
};
```

```
const context = useContext(QuoteContext)

useEffect(() => {
    const fetchFormData = async () => {
            context.dispatch({
                type: "FETCH_FIELDS_REQUEST"
            });
            try {
                const response = await axios.get("someAPI");
                context.dispatch({
                    type: "FETCH_FIELDS_SUCCESS",
                    payload: response.data
                });
            } catch (e) {
                context.dispatch({
                    type: "FETCH_FIELDS_FAILURE"
                });
            }

    };
    fetchFormData();
}, []);
```

AV **avala** ▶ dan-auth0 ↩ Reply to comment

@dan-auth0 you were completely correct. I to vider and refactored my app reducer and callbacks in th ha r this great write up!

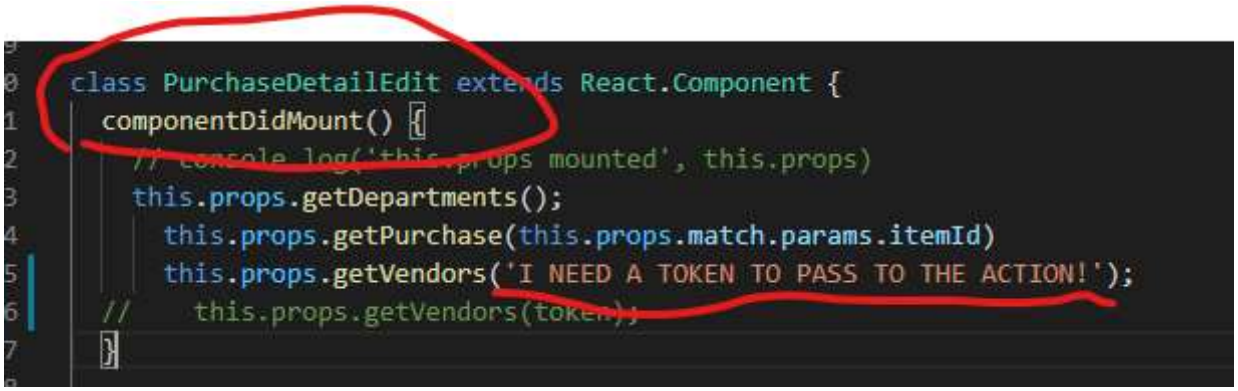
Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1 reply

MA **macm** ↩ Reply to comment

I have to admit this tutorial got me fast forwarded massively on getting auth0 working. Thanks Dan. Of course though I ran into issues when trying to integrate it into my react-redux app. I am wondering if you might be able to provide some insight into the following…

1: passing/getting a token when in a class based component



```
class PurchaseDetailEdit extends React.Component {
  componentDidMount() {
    // console.log('this.props mounted', this.props)
    this.props.getDepartments();
      this.props.getPurchase(this.props.match.params.itemId)
      this.props.getVendors('I NEED A TOKEN TO PASS TO THE ACTION!');
  //    this.props.getVendors(token);
  }
```

2: Setting the user profile data into a reducer and assigning the props?

1 reply

 **dan-auth0** Auth0 Employee ▶ avala ↩ Reply to comment

Hey, Avala! Glad to read that 🎉 You are welcome, thank you for reading the guide 🙂

 **dan-auth0** Auth0 Employee ▶ macm ↩ Reply to comment

Howdy, Macm! Welcome to our Auth0 Community and thank you for reading the post.

With the guide that we just published, we are only scratching the surface of the React ecosystem. There are different uses cases that I am planning to cover in the coming weeks/months. The most important ones I have on my list are: React + TS version, using class components, and React + Redux.

I am in the process right now of formulating our Redux strategy, specially when the application architecture gets more complex using side effects.

In the meantime, this section of the SDK Docs, Use with a Class Component, can provide you insight on how to integrate the new Auth0 React SDK with class components 🙂 In a nutshell: You use the `withAuth0` higher-order compone th onent. Then, all the SDK props are available to you through

I'll keep this Community Topic updated as we cos

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

dc.gaudium ↩ Reply to comment

What is the difference between `@auth0/auth0-spa-js` and `@auth0/auth0-react`?

The SPA sdk is older while the React sdk is very new, but how are they different in usage and underlying implementation? Are there differences/benefits that would make it easier to use one over the other?

Also, in terms of the conversation on a previous blog post (https://auth0.com/blog/securing-gatsby-with-auth0/), which sdk would you recommend for usage with Gatsby given that Gatsby is geared towards SPAs built with React?

1 reply

dan-auth0 Auth0 Employee ▶ dc.gaudium ↩ Reply to comment

David, that's a great question 🙂 I'll be creating much more content on React Security and Identity in the coming weeks/months and this is a use cause that I want to highlight! React SDK vs Auth0 SPA SDK.

Technically, what happens under the hood is still the same. However, prior to the introduction of the React SDK, we prompted developers to create a React Component to manage their Auth0 Authentication integration. You'd copy and paste a large section of code into a file that would be your "React SDK".

What we did was to package all that integration into its own consumable package following the idioms of React. Now, you download the SDK and think in terms of React constructs at a high level (Hooks, Higher-Order Components) instead of having to implement authentication functionality from scratch.

You could also say that the React SDK is opinionated. We provide you with the high-level functions to power the authentication flow. If you were to use the Auth0 SPA SDK, you assemble the flow perhaps in a different way and create different hooks or wrappers.

To my knowledge, if you need to access the Auth0 Client outside of the React Realm, you may benefit from using the Auth0 SPA SDK.

Regarding Gatsby:

We are in the process of updating our Gatsby recommend that you try out the React SDK with Gatsby as it HOC to wrap the components you want to pro cor for the router that Gatsby uses. On my personal experience, I always ended up migrating to React

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

Router as I find it easier to create that private route component.

solace ↩ Reply to comment

Hi,

Thanks for this comprehensive guide.

I've set up an app which uses `@auth0/auth0-react` for the front-end as described, and `auth0` in the back to perform management actions including updating `user_metadata` and `email`. With updating email, it requires you to log in again so that's one workaround, but when you update metadata there doesn't seem to be a way to update the cached user object in the front-end to reflect the changes without logging out and back in again.

Is there a way to re-fetch the user, or (less optimally) manipulate the cached data to reflect the changes?

Thanks

1 reply

dan-auth0 Auth0 Employee ▶ solace ↩ Reply to comment

Howdy, Solace! Welcome to the Auth0 Community and thank you for reading the guide! This is an excellent question. Let me consult with my team internally to see what strategy could work here 🙂

1 reply

solace ▶ dan-auth0 ↩ Reply to comment

Thanks. Looking forward to hear what you come up with.

1 reply

asdFletcher ↩ Reply to comment

I'm working through the react mega tutorial and I can now successfully login using username and password.

For curiosity's sake, I am interested in viewing the JWT that the client receives after authentication.

Where is this JWT stored? As a developer, ho

SDK: @auth0/auth0-react 1.0.0

Thank you!

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

1 reply

konrad.sopala Community Engineer ↩ Reply to comment

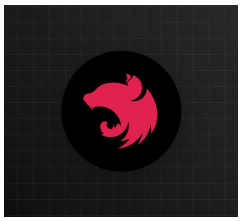
Hey there!

Moving your message here as this si the thread related to handling issues and questions regarding that blog article. Thank you!

dan-auth0 Auth0 Employee ↩ Reply to comment

Howdy, asdFletcher! Check out the “Hack Yourself” section of the “Developing a Secure API with NestJS: Managing Roles” which shows you how to get the access token. Once you have the token, you can use https://jwt.io/ to decode it.

For background: the client demo app that we use in the NestJS tutorial is a React app, but this process of inspecting the access token is the same for any client app.

> Auth0 - Blog
>
> 
>
> **Developing a Secure API with NestJS: Managing Roles**
>
> Learn how to use NestJS, a Node.js framework powered by TypeScript, to build a secure API.

1 reply

dan-auth0 Auth0 Employee ▶ solace ↩ Reply to comment

Just wanted to follow up with you: I am still researching what guidance we can provide on getting user metadata. Let me please ask you: how frequent do you estimate that data will change?

1 reply

solace ▶ dan-auth0 ↩ Reply to comment

Thanks for checking in.

I was planning on using user_metadata for pr[illegible], and possibly settings/preferences if that helps.

If this is not a recommended use of that property let me know.

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

Thanks!

1 reply

FL **asdFletcher** ▶ dan-auth0 ↩ Reply to comment

Hi Dan, thanks for the quick response.

I have retrieved the client id JWT from the “token” response that I receive on authentication, I decoded it and it appears to be different from the access token I get when I call:

```
getAccessTokenSilently()
```

However, the information when decoded is the exact same. I realize there’s not really a question there.

A followup question would be, how does my node.js backend verify that the JWT is valid? All the information it has is environment variables:

```
AUTH0_AUDIENCE=
AUTH0_ISSUER=
```

How does my backend know that an authenticated client hasn’t changed scopes to include “admin”, for instance. Or is that what the secret key is for?

2 replies

 **dan-auth0** Auth0 Employee ▶ asdFletcher ↩ Reply to comment

You are welcome. I am putting together an answer for you in relation to token integrity and trust on its claims 🙂

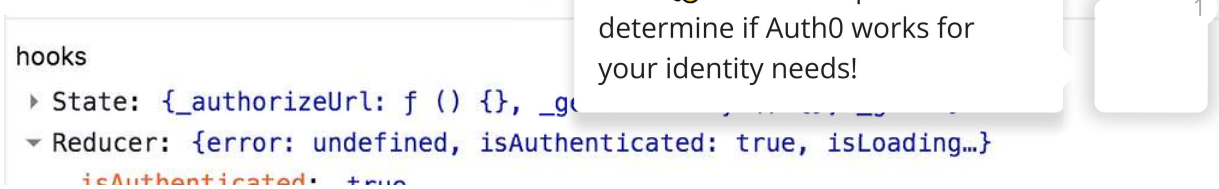
M **m.j.j.golda** ↩ Reply to comment

Hi, I used React Auth0 SDK according to https://auth0.com/blog/complete-guide-to-react-user-authentication/#Get-the-Starter-Application with my custom app. Everything seems to be fine.

However when I open react dev tools in chrome I can modify Auth0Provider state so I can change reducer and isAuthenticated to true. This enable me to get into app without login and protected routes are also not protected as long as y re webpack.

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1



```
isAuthenticated: true
isLoading: false
user: false
error: undefined
Effect: ƒ () {}
```

Can I somehow protect this state in react devtools??

Thanks

konrad.sopala Community Engineer ↩ Reply to comment

Moving your topic to this thread as this is the one to raise questions and issues regarding this blog article.

1 reply

M

m.j.j.golda ▶ konrad.sopala ↩ Reply to comment

Thank you.

I turned off react dev tools for my production build.
However I am curious if you have other solution for this issue?
Otherwise it would be nice to warn users about enabling React dev tools in their app.

It looks like security bug in React SDK.

2 replies

dan-auth0 Auth0 Employee ▶ m.j.j.golda ↩ Reply to comment

Howdy, Marcin. Welcome to the Auth0 Community. Thank you for reading the blog post and for sharing this feedback.

I'll share your question with the team who developed the SDK and get back to you as soon as I have an answer 👍

dan-auth0 Auth0 Employee ▶ m.j.j.golda ↩ Reply to comment

Hey, Marcin. I did some research about your concern. In essence, it is safe to assume that any of the client-side code can be modified as it r[illegible]st, the client-side routing is implemented for UX purposes a[illegible]

We should not see the client-side router as a r[illegible]ra tool for navigation and information presentation.

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

You secure data in a SPA by putting it behind an API that is protected by an access token. Any content you deploy with your SPA will be visible to anonymous users who can find it using the browser dev tools or by viewing its source. I checked with the team and we determined that the

Secure access for everyone. But not just anyone.

TRY AUTH0 FOR FREE

TALK TO SALES

BLOG

Developers
Identity & Security
Business
Leadership
Culture
Engineering
Announcements

COMPANY

About Us
Customers
Security
Careers
Partners
Press
Status
Legal
Privacy Policy
Terms

PRODUCT

Single Sign-On
Password Detection
Guardian
M2M
Universal Login
Passwordless

MORE

Auth0.com
Ambassador Program
Guest Author Program
Auth0 Community
Resources

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1

© 2013-2022 Auth0 Inc. All Rights Reserved.

Psst! 👋We have a quiz to determine if Auth0 works for your identity needs!

1