Search for applications or featu 1 Help & Support Documentation Talk to Sales artfullylondon

Your tenant has the **Legacy Lock API** enabled. Please follow our Deprecation Guide then disable the **Legacy Lock API** in your advanced settings. The Legacy Lock API will be removed on **July 16th, 2018** and your applications will no longer work if you have not fully migrated.

Dashboard
Applications
APIs
SSO Integrations
Connections
Users
Rules
Hooks
Multifactor Auth
Hosted Pages
Emails
Logs
Anomaly Detection
Analytics
Extensions
Get support

Artfully London Staging

Quick Start Settings Addons Connections Client ID: e2f3RlV1TllUyUHKC1YeXAz80LvdIpIb

Single Page App > React

React: Login

By Andres Aguiar
Auth0

Sample Project

Download a sample project specific to this tutorial configured with your Auth0 API Keys.

DOWNLOAD Fork on Github ▸

Show requirements

Configure Callback URLs

A callback URL is a URL in your application where Auth0 redirects the user after they have authenticated.

You need to whitelist the callback URL for your app in the **Allowed Callback URLs** field in your Application Settings. If you do not set any callback URL, your users will see a mismatch error when they log in.

If you are following along with the downloadable sample projects for this tutorial directly, the **Callback URL** should be set to

```
http://localhost:3000/callback
```

Did it help? Yes / No

Install auth0.js

You need the auth0.js library to integrate Auth0 into your application.

Install auth0.js using npm or yarn.

```
# installation with npm
npm install --save auth0-js

# installation with yarn
```

```
yarn add auth0-js
```

Did it help? Yes / No

Once you install auth0.js, add it to your build system or bring it in to your project with a script tag.

```
<script type="text/javascript" src="node_modules/auth0-js/build/auth0.js"></script>
```

Did it help? Yes / No

If you do not want to use a package manager, you can retrieve auth0.js from Auth0's CDN.

```
<script src="https://cdn.auth0.com/js/auth0/9.5.1/auth0.min.js"></script>
```

Did it help? Yes / No

Add Authentication with Auth0

Universal login is the easiest way to set up authentication in your application. We recommend using it for the best experience, best security and the fullest array of features. This guide will use it to provide a way for your users to log in to your React application.

You can also embed the login dialog directly in your application using the Lock widget. If you use this method, some features, such as single sign-on, will not be accessible. To learn how to embed the Lock widget in your application, follow the Embedded Login sample.

When a user logs in, Auth0 returns three items:

- `access_token` : to learn more, see the Access Token documentation
- `id_token` : to learn more, see the ID Token documentation
- `expires_in` : the number of seconds before the Access Token expires

You can use these items in your application to set up and manage authentication.

Create an Authentication Service

Create a service to manage and coordinate user authentication. You can give the service any name. In the examples below, the service is `Auth` and the filename is `Auth.js` .

In the service add an instance of the `auth0.WebAuth` object. When creating that instance, you can specify the following:

- Configuration for your application and domain
- Response type, to show that you need a user's Access Token and an ID Token after authentication
- Audience and scope, which specify that authentication must be OIDC-conformant
- The URL where you want to redirect your users after authentication.

In this tutorial, the route is `/callback`, which is implemented in the Add a Callback Component step.

Add a `login` method that calls the `authorize` method from auth0.js.

```
// src/Auth/Auth.js

import auth0 from 'auth0-js';

export default class Auth {
  auth0 = new auth0.WebAuth({
    domain: 'artfullylondon.eu.auth0.com',
    clientID: 'e2f3RlV1TllUyUHKC1YeXAz80LvdIpIb',
    redirectUri: 'http://localhost:3000/callback',
    audience: 'https://artfullylondon.eu.auth0.com/userinfo',
    responseType: 'token id_token',
    scope: 'openid'
  });

  login() {
    this.auth0.authorize();
  }
}
```

Did it help? Yes / No

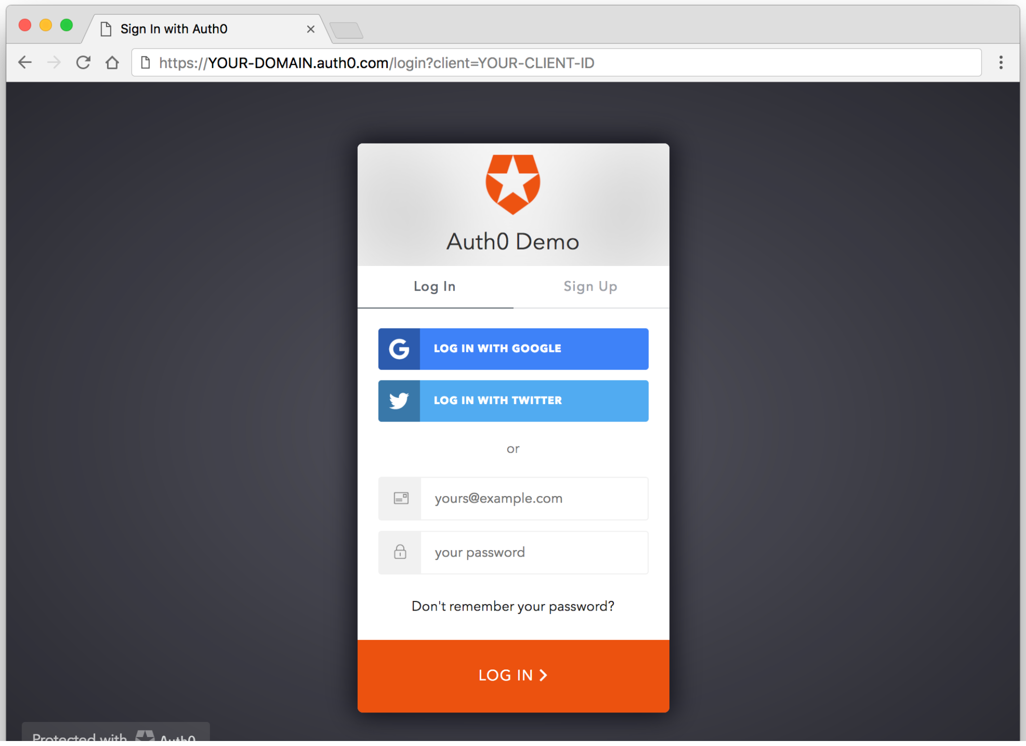
Checkpoint

Try to import the `Auth` service from somewhere in your application. Call the `login` method from the service to see the login page. For example:

```
// App.js
import Auth from './Auth/Auth.js';

const auth = new Auth();
auth.login();
```

Did it help? Yes / No



Dashboard

Applications

APIs

SSO Integrations

Connections

Users

Rules

Hooks

Multifactor Auth

Hosted Pages

Emails

Logs

Anomaly Detection

Analytics

Extensions

Get support

Finish the Service

Add more methods to the `Auth` service to handle authentication in the app.

The example below shows the following methods:

- `handleAuthentication` : looks for the result of authentication in the URL hash. Then, the result is processed with the `parseHash` method from auth0.js
- `setSession` : sets the user's Access Token, ID Token, and the Access Token's expiry time
- `logout` : removes the user's tokens and expiry time from browser storage
- `isAuthenticated` : checks whether the expiry time for the user's Access Token has passed

```
// src/Auth/Auth.js

import history from '../history';

// ...
export default class Auth {
  // ...
  constructor() {
    this.login = this.login.bind(this);
    this.logout = this.logout.bind(this);
    this.handleAuthentication = this.handleAuthentication.bind(this);
    this.isAuthenticated = this.isAuthenticated.bind(this);
  }

  handleAuthentication() {
    this.auth0.parseHash((err, authResult) => {
      if (authResult && authResult.accessToken && authResult.idToken) {
        this.setSession(authResult);
        history.replace('/home');
      } else if (err) {
        history.replace('/home');
        console.log(err);
      }
    });
  }

  setSession(authResult) {
    // Set the time that the Access Token will expire at
    let expiresAt = JSON.stringify((authResult.expiresIn * 1000) + new Date().getTime());
    localStorage.setItem('access_token', authResult.accessToken);
    localStorage.setItem('id_token', authResult.idToken);
    localStorage.setItem('expires_at', expiresAt);
    // navigate to the home route
    history.replace('/home');
  }

  logout() {
    // Clear Access Token and ID Token from local storage
    localStorage.removeItem('access_token');
    localStorage.removeItem('id_token');
    localStorage.removeItem('expires_at');
    // navigate to the home route
    history.replace('/home');
  }

  isAuthenticated() {
    // Check whether the current time is past the
    // Access Token's expiry time
    let expiresAt = JSON.parse(localStorage.getItem('expires_at'));
    return new Date().getTime() < expiresAt;
  }
}
```

Dashboard

Applications

APIs

SSO Integrations

Connections

Users

Rules

Hooks

Multifactor Auth

Hosted Pages

Emails

Logs

Anomaly Detection

Analytics

Extensions

Get support

Did it help? Yes / No

```
// src/history.js

import createHistory from 'history/createBrowserHistory'

export default createHistory()
```

Did it help? Yes / No

Provide a Login Control

Provide a component with controls for the user to log in and log out.

```
// src/App.js

import React, { Component } from 'react';
import { Navbar, Button } from 'react-bootstrap';
import './App.css';

class App extends Component {
  goTo(route) {
    this.props.history.replace(`/${route}`)
  }

  login() {
    this.props.auth.login();
  }

  logout() {
    this.props.auth.logout();
  }

  render() {
    const { isAuthenticated } = this.props.auth;

    return (
      <div>
        <Navbar fluid>
          <Navbar.Header>
            <Navbar.Brand>
              <a href="#">Auth0 - React</a>
            </Navbar.Brand>
            <Button
              bsStyle="primary"
              className="btn-margin"
              onClick={this.goTo.bind(this, 'home')}
            >
              Home
            </Button>
            {
              !isAuthenticated() && (
                  <Button
                    bsStyle="primary"
                    className="btn-margin"
                    onClick={this.login.bind(this)}
                  >
                    Log In
                  </Button>
                )
            }
            {
              isAuthenticated() && (
                  <Button
                    bsStyle="primary"
                    className="btn-margin"
                    onClick={this.logout.bind(this)}
                  >
                    Log Out
                  </Button>
                )
            }
          </Navbar.Header>
        </Navbar>
      </div>
    );
  }
}

export default App;
```

Did it help? Yes / No

This example uses Bootstrap styles. You can use any style library you want, or not use one at all.

Depending on whether the user is authenticated or not, they see the **Log In** or **Log Out** button. The `click` events on the buttons make calls to the `Auth` service to let the user log out or log in. When the user clicks the **Log In** button, they are redirected to the login page.

The login page uses the Lock widget. To learn more about universal login and the login page, see the universal login documentation. To customize the look and feel of the Lock widget, see the Lock customization options documentation.

Add a Callback Component

When you use the login page, your users are taken away from your application. After they authenticate, the users automatically return to your application and a client-side session is set for them.

This example assumes you are using path-based routing with `<BrowserRouter>`. If you are using hash-based routing, you will not be able to specify a dedicated callback route. The URL hash will be used to hold the user's authentication information.

Dashboard

Applications

APIs

SSO Integrations

Connections

Users

Rules

Hooks

Multifactor Auth

Hosted Pages

Emails

Logs

Anomaly Detection

Analytics

Extensions

Get support

You can select any URL in your application for your users to return to. We recommend creating a dedicated callback route. If you create a single callback route:

- You don't have to whitelist many, sometimes unknown, callback URLs.
- You can display a loading indicator while the application sets up a client-side session.

Create a component named `CallbackComponent` and add a loading indicator.

To display a loading indicator, you need a loading spinner or another indicator in the `assets` directory. See the downloadable sample for demonstration.

```
// src/Callback/Callback.js

import React, { Component } from 'react';
import loading from './loading.svg';

class Callback extends Component {
  render() {
    const style = //...

    return (
      <div style={style}>
        <img src={loading} alt="loading"/>
      </div>
    );
  }
}

export default Callback;
```

Did it help? Yes / No

After authentication, your users are taken to the `/callback` route. They see the loading indicator while the application sets up a client-side session for them. After the session is set up, the users are redirected to the `/home` route.

Process the Authentication Result

When a user authenticates at the login page, they are redirected to your application. Their URL contains a hash fragment with their authentication information. The `handleAuthentication` method in the `Auth` service processes the hash.

Call the `handleAuthentication` method after you render the `Callback` route. The method processes the authentication hash fragment when the `Callback` component initializes.

```
// src/routes.js

import React from 'react';
import { Route, Router } from 'react-router-dom';
import App from './App';
import Home from './Home/Home';
import Callback from './Callback/Callback';
import Auth from './Auth/Auth';
import history from './history';

const auth = new Auth();

const handleAuthentication = (nextState, replace) => {
  if (/access_token|id_token|error/.test(nextState.location.hash)) {
    auth.handleAuthentication();
  }
}

export const makeMainRoutes = () => {
  return (
    <Router history={history} component={App}>
      <div>
        <Route path="/" render={(props) => <App auth={auth} {...props} />} />
        <Route path="/home" render={(props) => <Home auth={auth} {...props} />} />
        <Route path="/callback" render={(props) => {
          handleAuthentication(props);
          return <Callback {...props} />
        }}/>
      </div>
    </Router>
  );
}
```

Did it help? Yes / No

More React tutorials

- Login
- User Profile
- Calling an API
- Authorization
- Token Renewal

Did it work?

✓ YES ✗ NO

Any suggestion or typo? Edit on GitHub ▸