On this page

Sign users in to your SPA using redirect and Auth JS

The Okta JavaScript Auth SDK (Auth JS) helps implement many Web Authentication solutions for both the redirect and embedded models. This guide creates a simple redirect authentication solution using Auth JS. Once created, you can drop the solution into just about any front-end or server-side web app.

Learning outcomes

- Understand how to implement a simple sign-in redirect using Auth JS.
- Understand basic installation and code configurations using Auth JS.

What you need

About the Okta Auth JavaScript SDK

The Okta Auth JavaScript SDK builds on top of the following:

Sign-in experiences

These experiences include fully branded embedded authentication, as with Auth JS fundamentals and redirect authentication. Auth JS is used by the Okta Sign-In Widget, which powers the default Okta sign-in page. Your app initializes the SDK, which automatically redirects you to this authentication page. Okta hosts this page and enforces the policies you configured for your sign-in experience.

Auth JS also powers our other redirect SDKs that provide simple authentication for server-side web apps and single-page JavaScript apps (SPA). See the Quickstart guides.

> **Note**: For your app to work reliably in all browsers, you must configure your Okta org with a custom domain. This prevents browser privacy features from blocking Okta's session cookies and causing sign-in or sign-out failures.

Auth JS and redirect authentication

In this guide, you don't need to use an Okta-supported server-side or front-end framework for redirect authentication. It's possible to use Auth JS to create a drop-in solution. This solution works with most web apps. It also works whether you're adding a centralized sign-in flow to a new app or retrofitting it to an existing app.

Create an Okta app integration

An Okta app integration represents your app in your Okta org. The integration configures how your app integrates with Okta services, which include the following:

- Users and groups that have access
- App sign-in policies
- Token refresh requirements
- Redirect URLs

The integration includes configuration information required by the app to access Okta.

1. Sign in to your Okta organization with your administrator account.
2. Click **Admin** in the upper-right corner of the page.
3. Go to **Applications** > **Applications**.
4. Click **Create App Integration**, and then select a **Sign-in method** of **OIDC - OpenID Connect**.
5. Select an **Application type** of **Single-Page Application**, and then click **Next**.

> **Note:** If you choose an inappropriate app type, it breaks the sign-in or sign-out flow. It breaks the flow by requiring the verification of a client secret, which public clients don't have.

3. Enter the **Sign-out redirect URIs** for local development. For this sample, use `http://localhost:9000`.
4. Enter the **Base URIs** for the trusted origin. For this sample, use `http://localhost:9000`. See Trusted origins.
5. In the **Assignments** section, define the type of **Controlled access** for your app. Select **Allow everyone in your organization to access**. See Assign app integrations.
6. Clear the **Enable immediate access with Federation Broker Mode** checkbox.
7. Click **Save** to create the app integration. The configuration pane for the integration opens after it's saved. Keep this pane open as you need to copy the **Client ID** and your org domain name when configuring your app.

About trusted origins

Reduce possible attack vectors by defining trusted origins, which are the websites allowed to access the Okta API for your app integration. Cross-Origin Resource Sharing (CORS) enables JavaScript requests using `XMLHttpRequest` with the Okta session cookie. See Grant cross-origin access to websites.

> **Note:** To reduce risk, only grant access to the Okta API to specific websites (origins) that you both control and trust.

To review or set trusted origins go to **Security** > **API** and select the **Trusted Origins** tab. See Enable trusted origins.

Create a basic app

Review the following sections to build out the sample app, or see the full sample app code in the Add a sign-out function section.

Create a simple HTML UI

In your app directory, create an HTML file called `index.html`. Add the following markup to this file:

```
<html>
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Okta Auth JS - Redirect SPA</title>
</head>

<body>
 <b>Okta Auth JS Simple Redirect App</b>
 <hr />
 <div id="content-jwt"></div>
</body>

</html>
```

The content in the body tags represents a simple app. The `content-jwt` reference displays user information after you add some upcoming JavaScript.

```
<!-- Latest CDN production Auth JS SDK -->
<script src="https://global.oktacdn.com/okta-auth-js/7.7.0/okta-auth-js.min.js" type="text/javasc
```

> **Note:** If you're using a package manager, you can also install it through the appropriate command, for example `yarn add @okta/okta-auth-js` or `npm install @okta/okta-auth-js`.

Add JavaScript to initialize the SDK

In the same `index.html` file, add the following JavaScript code after the Auth JS SDK reference. Update the `baseOktaURL` to your org and the `appClientID` to the client ID from the Create an Okta app integration section.

```
<script type="text/javascript">

// UPDATE THESE FOR YOUR OKTA TENANT
var baseOktaURL = "https:${yourOktaDomain}"; //For example, https://yourdomain.com
var appClientID = "${yourClientID}"; // For example, 0oa73hm5sh9jf6s5e1d6

// Bootstrap the AuthJS Client
const authClient = new OktaAuth({
  // Required Fields for OIDC client
  url: baseOktaURL,
  clientId: appClientID,
  redirectUri: "http://localhost:9000/", //or the redirect URI for your app
```

This code initializes the SDK by creating an instance of the `OktaAuth()` object. The object stores all the necessary config information for your auth session and is used to control subsequent steps of the process.

Get info about the user

Include the following JavaScript within the `<script>` tags and after the code that initializes the SDK:

```
if (authClient.isLoginRedirect()) {
  // Parse token from redirect url
  console.log("Parse token from redirect url");
  authClient.token.parseFromUrl()
    .then(data => {
      const { idToken } = data.tokens;
      // Display the Token
      const str1 = document.createElement('p');
      str1.innerHTML = `<b>${idToken.claims.email}</b> (email)<br /><b>${idToken.claims.sub}</b>
      document.getElementById('content-jwt').appendChild(str1);
    });
} else {
  // Always Redirect to get a "Fresh JWT" - Skipping the Token Manager in this example
  console.log("Attempt to retrieve ID Token from redirect");
  authClient.token.getWithRedirect({
          responseType: ['id_token']
```

The previous JavaScript parses the ID token and prints the data to your web page. See token.parseFromUrl() in the Auth JS SDK.

Add a sign-out function

Include the following function within the `body` tags after the `content-jwt` reference:

```
 <hr />
 <div id="uxActiveOptions">
  <b>Functions:</b>
  <br /><button onclick="authClient.signOut();">Close Okta Session</button>
</div>
```

This function signs the user out of the Okta session. See signOut() in the Auth JS SDK.

> **Note:** If users have already signed out or there's no active session, the browser redirects to the **Sign-out redirect URI** defined in your app integration.

After adding the sign-out function, the sample app is ready to test. Your sample app code appears as follows:

```
<html>
<head>
```

```
<script src="https://global.oktacdn.com/okta-auth-js/7.7.0/okta-auth-js.min.js" type="text/javas
<script type="text/javascript">


  // UPDATE THESE FOR YOUR OKTA TENANT
  var baseOktaURL = "https:${yourOktaDomain}"; //For example, "https://yourdomain.com"
  var appClientID = "${yourClientID}"; // For example, 0oa73hm5sh9jf6s5e1d6


  // Bootstrap the AuthJS Client
  const authClient = new OktaAuth({
  // Required Fields for OIDC client
    url: baseOktaURL,
    clientId: appClientID,
    redirectUri: "http://localhost:9000/", //or the redirect URI for your app
    issuer: baseOktaURL , // oidc
    scopes: ['openid', 'profile', 'email']
  });


  if (authClient.isLoginRedirect()) {
      // Parse token from redirect url
      console.log("Parse token from redirect url");
      authClient.token.parseFromUrl()
          .then(data => {
```

```
        });
    } else {
        // Always Redirect to get a "Fresh JWT" - Skipping the Token Manager in this example
        console.log("Attempt to retrieve ID Token from redirect");
        authClient.token.getWithRedirect({
            responseType: ['id_token']
        });
    }
</script>
</head>


<body>
<b>Okta Auth JS Simple Redirect App</b>
<hr />
<div id="content-jwt"></div>
<hr />
<div id="uxActiveOptions">
<b>Functions:</b>
<br /><button onclick="authClient.signOut();">Close Okta Session</button>
</div>
</body>
</html>
```

```
python3 -m http.server 9000
```

Open a private or incognito browser window and go to `http://localhost:9000` or the port for your local web server. The sample app starts the redirect flow when the page opens, and the Sign-In Widget for your org appears. Sign in with a user assigned to your app, and with a successful authentication, your app's `index.html` page displays the ID token for the signed-in user.

Troubleshoot your app

If your app isn't functional, ensure that:

- Your org URL is accurate and formatted correctly, including the secure protocol, `https://` .
- Your client ID is accurate from your Okta app integration.
- Your `redirectUri` is accurate or the port number for your local web server is correct.
- You've enabled a trusted origin for `http://localhost:9000` . See **About trusted origins**.
- If your app is bypassing the Okta Sign-In Widget, your user is already signed in. Use a new private or incognito browser window or optionally set the app sign-in policy for your app to always sign in. That is, the **Re-authentication frequency is** set to **Every sign-in attempt**.

Enable profile enrollment (self-service registration)

The profile enrollment or self-service registration feature provides a **Sign-up** link on the Sign-In Widget. Your end users use this link to register their user profile and sign in to your app.

b. Click the **Assignments** tab.

c. Click the **Groups** filter.

d. If the Everyone group isn't assigned, add it by clicking **Assign** > **Assign to Groups**, and assigning to the Everyone group.

2. Go to **Security** > **User Profile Policies** and edit the **Default Policy**.
3. Notice in the **Profile Enrollment** section, **Denied** is selected for **Self-service registration** by default. This setting removes the self-registration option for all apps assigned to the default policy.
4. Test your app and note that the **Sign-up** link doesn't appear under the sign-in page.
5. Return to the Admin Console, and then select **Back to all user profile policies** to return to the **Security** > **User Profile Policies** page.
6. Click **Add user profile policy**, and then create a name for the policy (for example, "App self-service registration").
7. Edit the new policy and note that self-service registration is **Allowed** by default.
8. Clear the **Email verification** checkbox for ease of testing and to allow your new user to sign in to the app immediately. Click **Save**.
9. Click **Manage Apps** and then **Add an App to This Policy**. Add or apply your sample app to this new policy.
10. Test your app again and note that the text **Don't have an account? Sign up** link now appears for your app under the Sign-In Widget. Click the link to add a user:

    a. Enter a first name, family name, and email address and click **Sign up**.

> **Note:** All new users through the self-registration process receive a welcome email. This email activates user access to the apps on your Integrator Free Plan org and demonstrates ownership of the email authenticator. If you complete this process, ensure you're in the same browser window as the app sign-in tab.

Enable progressive profile enrollment

Progressive profile enrollment builds out a user's profile incrementally during sign-in. The user profile policy is evaluated every time a user signs in. Based on the profile fields you want to add, this data is requested from users before signing in. At least one field must be set as required to enable the progressive profile enrollment feature. If a user's profile already has the requested data, the user signs in directly.

1. Go to **Security** > **User Profile Policies** and click **Add user profile policy**.
2. Create a name for the policy and **Save**.
3. Click edit from the **Actions** column for your new policy.
4. Click **Edit** in the policy and then for **Progressive Profiling**, select the **Enabled** option.

> **Note:** You can also enable profile enrollment (self-service registration) and progressive profile enrollment with the same policy. Select **Allowed** for the **Self-service registration** option of your user profile policy. New

6. Add the other user profile fields that you want existing users to provide in the **Profile enrollment form**. In this example, add the city field:

    a. Click **Add form input** and select the **City (city)** field. If the field is read only, you must change the attribute permission. See Create a profile enrollment form.

    b. Repeat this step for the number of fields that you want to add. At least one of these fields must be set as **Required**.

7. Click **Manage Apps** and then **Add an App to This Policy**. Add or apply your sample app to this new policy.

8. Test your app. Sign in with a user that doesn't have a city added to their profile.

    a. After entering the user's credentials, a new dialog requests the required user profile data. In this scenario, the **City** field. Add a city to this user's profile and fill out any other required or optional fields that you configured. After the user adds the data, they're signed in as normal.

    b. Sign out of the sample app by clicking **Close Okta Session**.

    c. Sign in again with the same user. With the data already added to the user's profile, the user is signed in directly.

Add MFA with a mandatory second factor

By default, your Integrator Free Plan org isn't configured for multifactor authentication. Use the following steps to understand the policy configurations and set up this use case. This setup requires an end user to authenticate with a password and a phone authenticator.

1. Go to **Security** > **Authenticators** and ensure that the phone authenticator is available in the **Authenticators** list on the **Setup** tab.

2. Go to **Security** > **Authentication policies** > **App sign-in**, click **Create policy**.
3. Add a name for the policy. For example: **Mandatory MFA**.
4. Click **Edit** under the **Actions** dropdown menu.
5. Select **Password / IdP + Another factor** for **User must authenticate with**. Ensure that the **Possession factor constraints are** doesn't have the **Exclude phone and email authenticators** checkbox selected. Click **Save**.
6. Go to **Applications** > **Applications** and select your app.
7. Click the **Sign On** tab, scroll down to the **User authentication** section, and click **Edit**.
8. Select your new app sign-in policy, **Mandatory MFA**, from the **App sign-in policy** dropdown menu, and click **Save**.
9. Test the new configurations by signing in to your app. If your test user doesn't have a phone number enrolled, the user is prompted for the enrollment during sign-in. Enroll the test user, add the SMS code, and the user is signed-in to your sample app.

After your users have enrolled in the phone authenticator, future sign-in flows require both a password and SMS code to access your app.

Enable password recovery with email magic link

By default, the Integrator Free Plan org is configured for a self-service password reset. Review the following steps to understand the policy configurations and to enable your sample app users to self-recover their password through an email magic link.

b. Set **This authenticator can be used for** to **Recovery**, and click **Add**.

2. Go to **Security** > **Authenticators** and edit the **Password** authenticator by clicking **Actions** > **Edit**.
3. Scroll to the bottom of the **Default Policy** and edit the **Default Rule**.
4. Ensure that **Password reset** is selected for the **Users can perform self-service** section.
5. Ensure that **Email** is selected for the **Users can initiate recovery with** section.
6. Click **Not required** for the **additional verification is** question, and then click **Update rule**.
7. If you previously set the sign-on policy for your app as Mandatory MFA, go to **Applications** > **Applications** and select your app. Click the **Sign On** tab and scroll down to the **User authentication** section and click **Edit**. From the **App sign-in policy** dropdown menu, select **One factor access** and click **Save**.

Test the new configurations by recovering a password for a user of your sample app:

1. Start your app. On the Sign-In Widget, click the **Forgot password?** link.
2. Add the email address for your test user.
3. Click the **Send me an email** link.
4. Check your test user's email inbox and wait for the **Account password reset** email from your Integrator Free Plan org.

> **Note:** Stay in the same browser window as the app sign-in tab.

See Self-service account recovery.

Enable passwordless authentication

Enable passwordless authentication for your existing users by configuring your Okta org's authenticator enrollment policy, app sign-in policy, and global session policy. This example uses the email authenticator to authenticate your users instead of a password. For full details and other passwordless implementation options, see **Set up passwordless sign-in experience**.

1. Go to **Directory** > **Groups** and click **Add group**. Give the group a name, for example, "Passwordless users", and click **Save**.
2. Select your new group, and click **Assign people** from the **People** tab. Add one or more users to your new group.
3. Go to **Security** > **Authenticators** and edit or ensure that the **Email** authenticator is set to **Authentication and recovery**.
4. Click the **Enrollment** tab, and then click **Add a policy** to add an enrollment policy targeted at your new group. Configure the following fields, and then click **Create Policy**:
   - **Policy name**: Any name for this policy, for example, "Passwordless enrollment"
   - **Assign to Groups**: Your new group, "Passwordless users"
   - **Email** authenticator: Set to required
   - **Password** authenticator: Set to disabled
5. Add a rule name, for example, "Passwordless enrollment rule", and click **Create rule** to complete the enrollment policy setup.

- **User must authenticate with**: Any one factor type or IdP

7. Click **Global Session Policy** and click **Add policy**. Give the policy a name, for example, "Global Passwordless policy", and assign the policy to your new group, "Passwordless users". Click **Create policy and add rule**.
8. Configure the following fields in the **Add rule** dialog and then click **Create rule**:
   - **Rule name**: Any name for this rule, for example, "Global Passwordless rule"
   - **Establish the session with**: Any factor used to meet the app sign-in policy requirements

Test the new configurations by signing in to your sample app with a user added to your "Passwordless users" group:

1. Start your app. On the Okta sign-in page, add the email address of your test user. Notice that there's no password field on the page.
2. Add the email address for your test user and click **Next**.
3. Click the **Send me an email** link to receive a verification email.
4. Open the email and use either the verification code or the email link to verify the user. The user is signed in to your sample app without a password.

See also

- Sign in to SPA
- Redirect auth in the sample app
- Auth JS fundamentals