Headless Identity Implementation Guide

Version 64.0, Summer '25

Last updated: June 20, 2025

CONTENTS

Chapter 1: What Is Headless Identity? 1

About This Guide 2
Headless Login Overview 2
Headless Registration Overview 3
Headless Forgot Password Overview 5
Building Native Single Sign-On Experiences 6

Chapter 2: Complete Prerequisites for Headless Identity 8

Create a Role to Manage Headless Identity Features 9
Assign the Headless Identity Admin Role to Yourself 10
Set Up a Demo Profile for End Users 10
Set Up Cross-Origin Resource Sharing (CORS) 11
Create an Account for End Users 11
Enable the Authorization Code and Credentials Flow 12
Implement reCAPTCHA on a Web App 12

Chapter 3: Set Up an Authentication Provider for Single Sign-On 14

Create an Authentication Provider Registration Handler 15
Configure a Google Authentication Provider 18

Chapter 4: Set Up an Experience Cloud Site for Headless Identity 19

Enable Digital Experiences and Create Your Site 20
Configure Experience Cloud Site Membership 20
Create a Headless Registration Handler 21
Configure Headless Identity Settings in Experience Cloud 24
Activate Your Site 25

Chapter 5: Set Up a Connected App for Headless Identity 26

Create an OAuth-Enabled Connected App 27
Configure Connected App Policies 27
Get Your Consumer Key 28

Chapter 6: Use Postman to Work with Headless Identity APIs 29

Set Up Your Postman Workspace 30
Headless Registration Flow: Send a Registration Request 31
Headless Registration Flow: Send an Authorization Request 32
Headless Registration Flow: Send a Token Request 33
Headless Login: Send an Authorization Request 33
Headless Login: Send a Token Request 34
Headless Registration and Headless Login Flows: Get User Info 35

Headless Forgot Password Flow: Send a Password Reset Request 35
Headless Forgot Password Flow: Change the User's Password 36
Headlessly Revoke an Access Token 37

Chapter 7: JavaScript Examples for Headless Identity APIs 38

Headless Login JavaScript Examples 39
Headless Registration JavaScript Examples 40
Headless Forgot Password JavaScript Examples 42
Building a Native Single Sign-On Experience JavaScript Examples 43
Getting User Info JavaScript Examples 45

CHAPTER 1 What Is Headless Identity?

In this chapter ...

- About This Guide
- Headless Login Overview
- Headless Registration Overview
- Headless Forgot Password Overview
- Building Native Single Sign-On Experiences

Salesforce Headless Identity gives you the ability to separate back-end authentication processes from front-end identity experiences. By calling Headless Identity APIs, you can use the power of Customer Identity for authentication while maintaining complete control over the user experience in an off-platform or third-party app. Salesforce offers three Headless Identity features: login with Headless Login API, registration with Headless Registration API, and password reset with Headless Forgot Password API. You can also link a single sign-on (SSO) provider to your headless app to create a native SSO experience.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Headless Identity use cases fall into two categories.

- Apps that complement a customer-facing Experience Cloud site. Users fully interact with and log in to the Experience Cloud site and the app. For example, you build a mobile app in addition to your main Experience Cloud site because you want to target mobile-first users. You want to fully design the user experience to suit your company's branding. You can completely control the user experience in your app while Salesforce provides identity services. And because you already have an Experience Cloud site, you can simplify your setup process.
- Standalone apps. Users interact with and log in to your app, but not an Experience Cloud site. For example, your company builds your own customer-facing apps to align with your digital marketing strategy. Because you want to use Salesforce to manage customer outreach and store information, enabling your users to log in and register for your apps is important. But you still want full control over the user experience in your apps. Headless Identity means you can have it all—you can provide identity services to your apps, manage customers in Salesforce, and keep up with your company's digital marketing strategy.

  For use cases in this category, you still create and set up an Experience Cloud site because Headless Identity APIs are exposed and configured through Experience Cloud. The Experience Cloud site also functions as a way to store your customer accounts and contact records and manage access to your app. But your users don't interact with it directly.

About This Guide

This guide walks you though an end-to-end Headless Identity implementation, from completing your Salesforce setup to calling Headless Identity API using a Postman collection. The example implementation in this guide is designed for a single-page app or public client.

The guide takes you through these high-level tasks.

- Complete prerequisites for Headless Identity.
- Create an authentication provider.
- Configure Experience Cloud settings.
- Configure a connected app.
- Use Postman to call Headless Identity APIs.

The guide also includes some JavaScript examples for implementing Headless Identity with a single-page app.

To use this guide, you must have a Salesforce admin account. We recommend completing these steps in a non-production environment, such as a sandbox or developer org, before you customize and deploy your own solution into production.

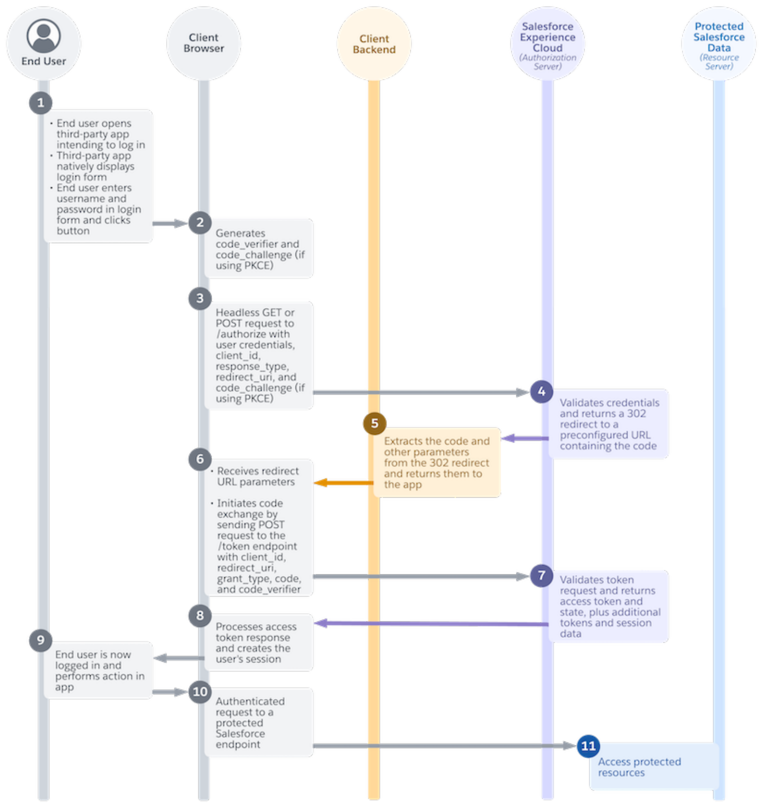
EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Headless Login Overview

You configure headless login via the Authorization Code and Credentials Flow, which is built on the OAuth 2.0 Authorization Code grant type. At the end of this flow, a user is logged in and can access Salesforce data. Here's a high-level overview of how the flow works with a single-page app.


End User
Client Browser
Client Backend
Salesforce Experience Cloud (Authorization Server)
Protected Salesforce Data (Resource Server)
1
• End user opens third-party app intending to log in
• Third-party app natively displays login form
• End user enters username and password in login form and clicks button
2
Generates code_verifier and code_challenge (if using PKCE)
3
Headless GET or POST request to /authorize with user credentials, client_id, response_type, redirect_uri, and code_challenge (if using PKCE)
4
Validates credentials and returns a 302 redirect to a preconfigured URL containing the code
5
Extracts the code and other parameters from the 302 redirect and returns them to the app
6
• Receives redirect URL parameters
• Initiates code exchange by sending POST request to the /token endpoint with client_id, redirect_uri, grant_type, code, and code_verifier
7
Validates token request and returns access token and state, plus additional tokens and session data
8
Processes access token response and creates the user's session
9
End user is now logged in and performs action in app
10
Authenticated request to a protected Salesforce endpoint
11
Access protected resources

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

- Your user goes to your custom app, where your login form is natively displayed, and enters their username and password (1).
- If you're using the Proof Key for Code Exchange (PKCE) extension, the app generates values to verify the authorization code. If you're not using PKCE, your flow skips this step. We strongly recommend that you always use PKCE when implementing this flow for single-page apps (2).
- From the browser, your custom app—via JavaScript—sends a headless authorization request to the Salesforce Headless Login API authorization endpoint on your Experience Cloud site (3).
- Salesforce returns a 302 redirect to a preconfigured URL containing the authorization code. If the flow is executed in the browser, the 302 redirect is processed and the response is delivered headlessly to your callback endpoint. For single-page apps, you can use the OAuth 2.0 echo endpoint, which is designed to make development for this use case easier (4).
- The callback endpoint extracts the authorization code from the 302 redirect and returns it to the app (5).
- The client-side JavaScript receives the redirect URL parameters and initiates the code exchange with a POST request to the token endpoint (6).
- Salesforce Headless Login API validates the request and returns an access token response to the app (7).
- Client-side JavaScript on the app processes the access token and creates the user's session (8).
- The user is now logged in, and they perform an action in your custom app that initiates a request for Salesforce data. For example, they click a button to access their travel booking history, which is stored in the Salesforce Experience Cloud site (9).
- Your custom app makes an authenticated request to a protected Salesforce endpoint, such as a Salesforce API (10).
- The user can now access their protected data in your custom app. For example, they can see their travel booking history (11).

Here are a few key concepts to keep in mind for this flow.

- Your app initiates headless login with an authorization request to Headless Login API.
- You use a callback endpoint to extract the authorization code. To make this process easier, instead of building your own endpoint, use the Salesforce OAuth 2.0 echo endpoint.
- Your app exchanges the code for an access token with a request to the token endpoint.

Headless Registration Overview

The Headless Registration Flow extends the Authorization Code and Credentials Flow. At the end of this flow, a new user is registered and logged in, and they can access Salesforce data. Here's a high-level overview of how the flow works with a single-page app.

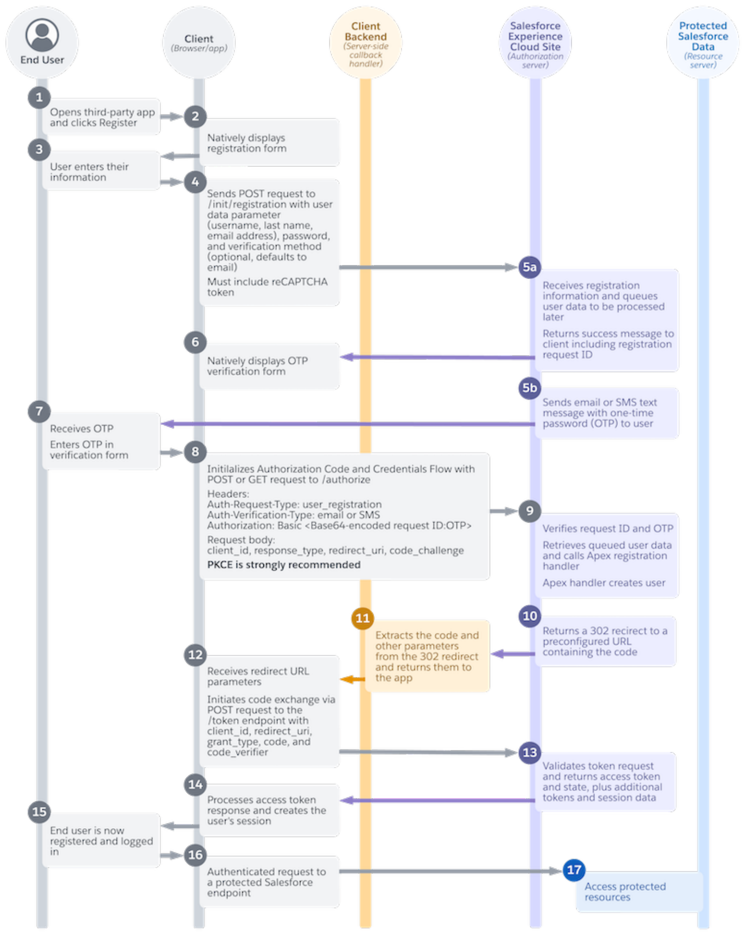
EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

- An end user opens your app and clicks **Register** (1).
- In your app, you natively display a registration form to collect user data. The look and feel of this form is entirely up to you, and you can fully customize what kind of information that you want to collect (2).
- The end user enters their information in the app. For example, they enter their new username, password, and first name (3).
- Your app submits the user information to the Headless Registration API endpoint on your Experience Cloud site (4).
- Salesforce receives the user information and queues it to be processed later (5a).
- Salesforce then sends an email or an SMS text message containing a one-time password (OTP) to the user (5b).
- In your app, you natively display an OTP verification form. Again, it's up to you how you want this form to look (6).
- The user receives their OTP and enters it in the verification form (7).
- Your app then initializes the Authorization Code and Credentials Flow with an authorization code request. The request includes the OTP and the request ID, along with other parameters (8).

- Salesforce verifies the request ID and OTP. It retrieves the queued user data that it stored earlier and calls the headless registration handler. The headless registration handler creates a user in Salesforce (9).
- Salesforce returns a 302 redirect to a preconfigured URL containing the authorization code. If the flow is executed in the browser, the 302 redirect is processed and the response is delivered headlessly to your callback endpoint. For single-page apps, you can use the OAuth 2.0 echo endpoint, which is designed to make development for this use case easier (10).
- The callback endpoint extracts the code and other parameters from the 302 redirect. It returns this information to your app (11).
- Your app initiates the code exchange via a POST request to the token endpoint (12).
- From the token endpoint, Salesforce returns an access token response to your app (13).
- Your app processes the token response and creates the user's session (14).
- The user is now registered and logged in. They perform an action in your custom app that initiates a request for Salesforce data. For example, they click a button to access their travel booking history, which is stored in the Salesforce Experience Cloud site (15).
- Your custom app makes an authenticated request to a protected Salesforce endpoint, such as a Salesforce API (16).
- The customer can now access their protected data in your custom app. For example, they can see their travel booking history (17).


End User
Client (Browser/app)
Client Backend (Server-side callback handler)
Salesforce Experience Cloud Site (Authorization server)
Protected Salesforce Data (Resource server)
1 Opens third-party app and clicks Register
2 Natively displays registration form
3 User enters their information
4 Sends POST request to /init/registration with user data parameter (username, last name, email address), password, and verification method (optional, defaults to email)
Must include reCAPTCHA token
5a Receives registration information and queues user data to be processed later
Returns success message to client including registration request ID
6 Natively displays OTP verification form
5b Sends email or SMS text message with one-time password (OTP) to user
7 Receives OTP
Enters OTP in verification form
8 Initilalizes Authorization Code and Credentials Flow with POST or GET request to /authorize
Headers:
Auth-Request-Type: user_registration
Auth-Verification-Type: email or SMS
Authorization: Basic <Base64-encoded request ID:OTP>
Request body:
client_id, response_type, redirect_uri, code_challenge
PKCE is strongly recommended
9 Verifies request ID and OTP
Retrieves queued user data and calls Apex registration handler
Apex handler creates user
10 Returns a 302 recirect to a preconfigured URL containing the code
11 Extracts the code and other parameters from the 302 redirect and returns them to the app
12 Receives redirect URL parameters
Initiates code exchange via POST request to the /token endpoint with client_id, redirect_uri, grant_type, code, and code_verifier
13 Validates token request and returns access token and state, plus additional tokens and session data
14 Processes access token response and creates the user's session
15 End user is now registered and logged in
16 Authenticated request to a protected Salesforce endpoint
17 Access protected resources

Here are a few key concepts to keep in mind.

- Your app initiates registration with a request to Headless Registration API.
- When Salesforce receives the user information from this request, it queues the information to be processed later. At this point, the user isn't registered yet.

- Your app initializes the Authorization Code and Credentials Flow with an authorization request to Headless Login API.
- The information sent in this authorization request prompts Salesforce to retrieve the queued user information and pass it to the headless registration handler. The registration handler creates the user.
- At the end of the flow, the user is registered and logged in.

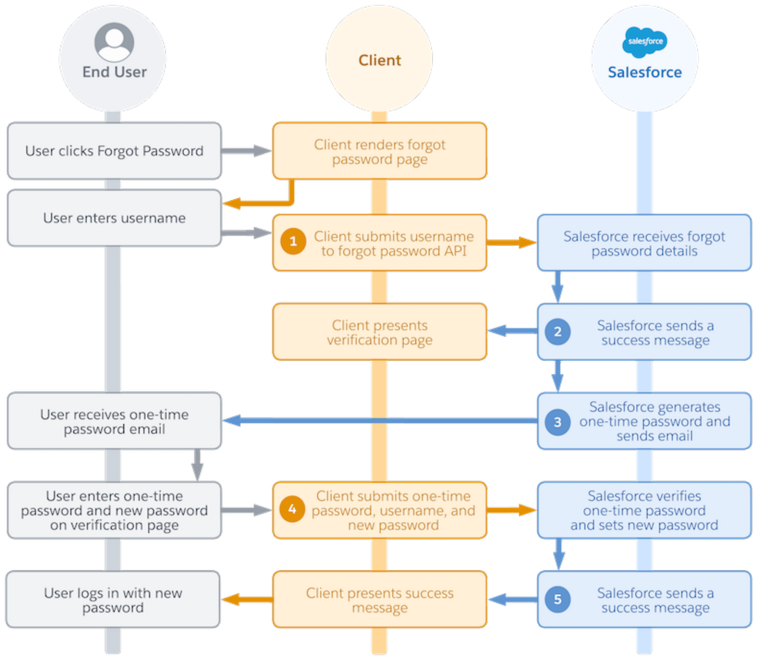
Headless Forgot Password Overview

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

If your users can log in and register, they must also be able to reset their passwords. Here's a high-level overview of how the Headless Forgot Password Flow works with a single-page app.


End User
Client
Salesforce
User clicks Forgot Password
Client renders forgot password page
User enters username
1 Client submits username to forgot password API
Salesforce receives forgot password details
Client presents verification page
2 Salesforce sends a success message
User receives one-time password email
3 Salesforce generates one-time password and sends email
User enters one-time password and new password on verification page
4 Client submits one-time password, username, and new password
Salesforce verifies one-time password and sets new password
User logs in with new password
Client presents success message
5 Salesforce sends a success message

- An end user clicks a password reset link in your app.
- In your app, you natively display a forgot password page.
- The user enters their username.
- Your app initiates the flow with a request to Headless Forgot Password API. The request includes the user's username (1).
- Salesforce receives the forgot password details.
- Salesforce returns a success message to your app (2).
- Immediately after, Salesforce generates a one-time password (OTP) and sends it to the user's email address (3).
- In your app, you natively display an OTP verification form.
- The user receives the OTP email.
- The user enters the OTP and their new password in your verification form.
- Your app finishes resetting the password with another request to Headless Forgot Password API. The request includes the username, OTP, and new password (4).
- Salesforce verifies the OTP and sets a new password.
- Salesforce returns a success message to your app (5).
- The user logs in with their new password.

Here are a few key concepts to keep in mind for this flow.

- This flow sends two requests to Headless Forgot Password API.
- The first request initializes the password reset and kicks off the OTP verification process.
- The second request completes the password reset process.

Building Native Single Sign-On Experiences

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

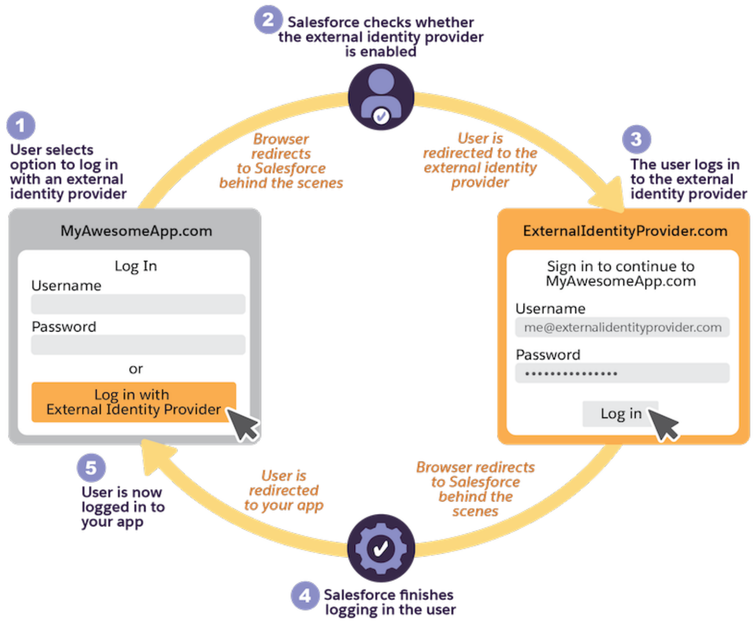
Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

You can configure the OAuth 2.0 web-server and user-agent flows to create a native single sign-on (SSO) experience in your app. Use this feature to add SSO to your Headless Identity implementation.

To create a native single sign-on experience in your app, you configure SSO with an authentication provider or SAML identity provider. You add this provider as a login option from your Experience Cloud settings. Then you configure the OAuth 2.0 web server flow or user-agent flow with your app. You pass in a parameter that specifies the name of the SSO provider you configured. On your app, you also build a button to display the option to log in with the SSO provider.

During the flow, when the user clicks the option to log in with the provider, the browser is redirected to your Experience Cloud site. Here, Salesforce checks for the SSO provider parameter specified in the authorization flow. The browser is then automatically redirected to the SSO provider. The user enters their credentials and the flow briefly redirects to your Experience Cloud site again before the user is redirected back to your app. These redirects happen automatically, giving the user the impression that your app is natively integrated with the external SSO provider—they don't see or interact with your Experience Cloud site.

For example, you want to set up SSO between your app and Google, and you want it to feel like your app is natively integrated with Google. Here's a simplified overview of the flow works when it's configured.


2 Salesforce checks whether the external identity provider is enabled
1 User selects option to log in with an external identity provider
Browser redirects to Salesforce behind the scenes
User is redirected to the external identity provider
3 The user logs in to the external identity provider
MyAwesomeApp.com
Log In
Username
Password
or
Log in with External Identity Provider
ExternalIdentityProvider.com
Sign in to continue to MyAwesomeApp.com
Username
me@externalidentityprovider.com
Password
Log in
5 User is now logged in to your app
User is redirected to your app
Browser redirects to Salesforce behind the scenes
4 Salesforce finishes logging in the user

- A user goes to your app and clicks a button so that they can log in with an external provider, such as Google (1).
- The browser briefly redirects to the Experience Cloud login page.
- Salesforce confirms that a Google SSO provider is enabled for the Experience Cloud site (2).
- The browser automatically redirects to the Google login page.
- The user enters their Google credentials (3).

- Google authenticates the user. The browser is briefly redirected back to the Experience Cloud login page.
- Salesforce finishes logging in the user (4).
- The browser is automatically redirected back to your app.
- The user is now logged in (5).

CHAPTER 2 Complete Prerequisites for Headless Identity

In this chapter ...

- Create a Role to Manage Headless Identity Features
- Assign the Headless Identity Admin Role to Yourself
- Set Up a Demo Profile for End Users
- Set Up Cross-Origin Resource Sharing (CORS)
- Create an Account for End Users
- Enable the Authorization Code and Credentials Flow
- Implement reCAPTCHA on a Web App

Cross off some basic setup steps in Salesforce and Google.

In Salesforce, these prerequisites include creating a role to manage your users, setting up a demo profile, enabling Cross-Origin Resource Sharing (CORS), and setting up an account to contain your users. You must also enable the Authorization Code and Credentials Flow at an org-wide level so that you can set up headless login and registration.

You must also implement Google reCAPTCHA to get a reCAPTCHA token so that you can use in the Postman examples. For single-page apps, with reCAPTCHA, you can secure your flow without passing secret information, like an integration user's access token, in any of your requests to Salesforce.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Create a Role to Manage Headless Identity Features

For identity and access management, it's important to define who can access what. Create a role to ensure that you have the right level of access to manage Headless Identity features.

1. From Setup, in the Quick Find box, enter `Roles`, and then select **Roles**.
2. Click **Set Up Roles**.
3. Under CEO, select **Add Role**.
4. Enter a label for the role, such as `Headless Identity Admin`.

   The Role Name autofills. You can keep it the same or change it.
5. Save your changes.

Stay on this Setup page to complete the next step.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To view roles and role hierarchy:
- View Roles and Role Hierarchy

To create, edit, and delete roles:
- Manage Roles

To assign users to roles:
- Manage Internal Users

Assign the Headless Identity Admin Role to Yourself

Assign the role that you created to your System Administrator user.

1. From your role detail page, select **Assign Users to Role**.

   If you left this page, you can get back to it by entering `Roles` in the Quick Find box, selecting **Roles**, and then selecting the role that you created.

2. If you don't see your admin user in Available Users, use the dropdown and search bar to refine your search.
3. Select your admin user, and add them to Selected Users for End Users.
4. Save your changes.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To view roles and role hierarchy:

- View Roles and Role Hierarchy

To create, edit, and delete roles:

- Manage Roles

To assign users to roles:

- Manage Internal Users

Set Up a Demo Profile for End Users

For Headless Identity, you use profiles to define how your end users access data in Salesforce. New users are automatically assigned to this profile when they register. Create a demo profile so that you can test your headless login, registration, forgot password, and single sign-on processes as an end user.

Instead of creating a new profile, you can clone an existing standard profile to get all of its preconfigured permissions and access settings. You can then customize the profile as needed. Because you use an Experience Cloud site to call Headless Identity APIs and store user data, your end users must be able to access Experience Cloud. We recommend cloning only these standard profiles so that users can log in via an Experience Cloud site.

- Customer Community User
- Customer Community Plus User
- External Identity User
- Partner Community User
- Partner User

For this example, we clone the Customer Community User profile.

1. From Setup, in the Quick Find box, enter `Profiles`, and then select **Profiles**.
2. From the profile list, select the checkbox next to Customer Community User.
3. Click **Clone**.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

4. Name the profile *`Headless Demo Profile`*.
5. Save your changes.

Set Up Cross-Origin Resource Sharing (CORS)

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

USER PERMISSIONS

To create, read, update, and delete:
- Modify All Data

In production, to use Headless Identity features in a web app, you must enable Cross-Origin Resource Sharing (CORS) so that your app can communicate with Salesforce. This step isn't necessary for the example implementation in this guide, so you can skip it for now. But if you want to learn how to set up CORS, here's how it's done.

In production, you set up CORS using the domain of your off-platform app. For this example implementation, if you don't have a test app in mind, you can create one using Heroku—you can try a basic account for free. Later in this guide, you can use the same app when you implement reCAPTCHA.

1. From Setup, in the Quick Find box, enter *`CORS`*, and then select **CORS**.
2. For Allowed Origins List, click **New**.
3. Enter a URL pattern that can identify your web app. For example, if your app is hosted on `myapp.com`, you enter *`https://www.myapp.com`*.
4. Save your changes.
   Your web app can now request resources from Salesforce.

Create an Account for End Users

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

You must use an account to store information about your end users, including their contact records. For Headless Identity, you reference the account in your Apex registration handlers. When new users log in via an authentication provider or sign up directly through your site, they're added as contacts.

Salesforce supports two types of accounts: business accounts, which store information about organizations, and person accounts, which store information about individuals. For this example, you use a business account to keep all your end-user records in one place.

1. From the App Launcher, find and select **Accounts**.
2. Click **New**.
3. For the Account Name, enter *`My Account`*.

   You can leave the rest of the details blank.
4. Save your new account.

Enable the Authorization Code and Credentials Flow

The Authorization Code and Credentials Flow is the foundation of headless login and headless registration. Enable this flow at an org-wide level.

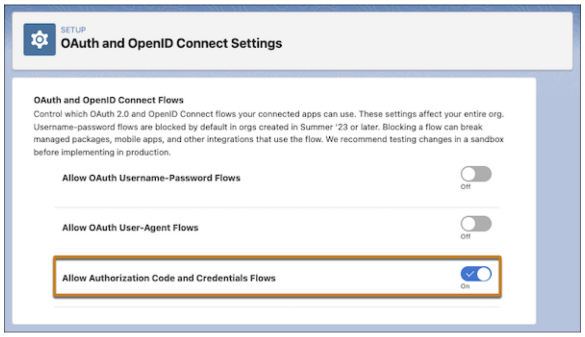
EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

1. From Setup, in the Quick Find box, enter `OAuth`, and then select **OAuth and OpenID Connect Settings**.
2. Turn on **Allow Authorization Code and Credentials Flows**.
   You receive a warning that changing this setting can break your integrations.
3. To accept the warning, click **OK**.
   With this flow enabled, you can access settings to turn it on for a specific connected app.

Example:


SETUP
OAuth and OpenID Connect Settings

OAuth and OpenID Connect Flows
Control which OAuth 2.0 and OpenID Connect flows your connected apps can use. These settings affect your entire org. Username-password flows are blocked by default in orgs created in Summer '23 or later. Blocking a flow can break managed packages, mobile apps, and other integrations that use the flow. We recommend testing changes in a sandbox before implementing in production.

Allow OAuth Username-Password Flows — Off

Allow OAuth User-Agent Flows — Off

Allow Authorization Code and Credentials Flows — On

Implement reCAPTCHA on a Web App

For the Headless Registration Flow and the Headless Forgot Password Flow, you must configure at least one of two security settings on the Experience Cloud Login & Registration page. These settings add requirements to the requests that your app sends to Headless Identity APIs. You can require your app to send an access token issued to an internal integration user. Or you can require your app to send a reCAPTCHA token. For single-page apps, because you can't keep the integration user's access token private in the browser, we recommend that you require your app to send a reCAPTCHA token instead. Requiring reCAPTCHA helps you filter out invalid requests, such as requests from bots. To work with the Headless Identity API Postman collection, you must have a valid reCAPTCHA token.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Though full-blown instructions for implementing reCAPTCHA are outside the scope of this guide, we show you where to go and what information you need. Salesforce supports reCAPTCHA v2 and reCAPTCHA v3 for Headless Identity. For this example, we use reCAPTCHA v3.

To get a reCAPTCHA token, you must set up and host reCAPTCHA on a web app. In production, you implement reCAPTCHA on your off-platform app.

For this example, if you don't have a test web app in mind, you can create one using Heroku—you can try a basic account for free.

To implement reCAPTCHA, see *reCAPTCHA v3* in the reCAPTCHA Developer's Guide at https://developers.google.com/recaptcha/docs/v3.

When you set up reCAPTCHA for this example, follow these guidelines.

- For the domain, enter the URL for your web app, such as `https://www.myapp.com`.

- For reCAPTCHA type, choose reCAPTCHA v3.
- Note your API key pair, which you use later in this guide.
- Ensure that you can get a reCAPTCHA token before you move on.

CHAPTER 3 Set Up an Authentication Provider for Single Sign-On

In this chapter ...

- Create an Authentication Provider Registration Handler
- Configure a Google Authentication Provider

Authentication providers are one way of setting up single sign-on (SSO) from an external identity provider, like Google, into Salesforce. With a single parameter, you can link an authentication provider to your app to create an SSO experience that feels native. For this example, we set up a Google authentication provider so that your users can log in to Google from your app.

This guide shows you how to create a registration handler, set up an authentication provider, and add it to your Experience Cloud site. This guide doesn't cover a Postman example for configuring the OAuth 2.0 web server flow or user-agent flow to use the `sso_provider` parameter. For full setup instructions, see Create a Native Single Sign-On Experience in Your App in Salesforce Help.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Create an Authentication Provider Registration Handler

Create an Apex registration handler to use with your authentication provider. When users log in to your third-party app with an external single sign-on (SSO) provider, the registration handler creates and updates their user records.

1. From Setup, in the Quick Find box, enter *Apex*, and then select **Apex Classes**.
2. Click **New**.
3. Fill the class with your registration handler code. You can paste in the code from the example.
4. Save the class, and note its name.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Example: Here's an example registration handler that you can use with your Google authentication provider. This class is triggered every time a user logs in with Google. If it's the user's first time logging in to your app, the class creates a user in Salesforce and associates them with the Headless Identity Demo profile that you created. The class also added the user to the account that you created earlier. If the user logged in to your app before, the class updates their record with any new information.

Note: This sample code is for demonstration only. Always test code before deploying it to a production environment.

```
/*
 * This class is a sample auth provider registration handler
 for a Headless Identity implementation
 * It creates an external user and associates them with the
Headless Demo Profile
 * It creates or associates the user contact to an Account
called My Account
 * It uses the email as the username
 * */

global class HeadlessDemoGoogleIDPRegistrationHandler
implements Auth.RegistrationHandler{
    static final String headless_account = 'My Account';
    static final String headless_profile = 'Headless Demo
Profile';

 /*
  * Tries to find a user with a username matching the incoming
 email
  * If not it creates one, associates it to an Account and
Profile
  * */
    global User createUser(Id portalId, Auth.UserData data)
{
        //Find an existing user, we are using username to map
 to email as your google email is a google username
        List<User> users = [SELECT Id, firstName, lastName,
email FROM User where Username =:data.email LIMIT 1];
        User u = null;
        if (!users.isEmpty()) {
            u = users[0];
        }
        //If no user is found we create one
        if (u == null) {
            u = new User();
            prepareUserData(data, u);

            //Get the Account, and create it if one is not already present.
            Account a;
           List<Account> accounts = [SELECT Id FROM Account WHERE name='My Account'];

            if(accounts.isEmpty()) {
                a = new Account(name = headless_account);
                insert(a);
            } else {
                a = accounts[0];
            }

            // Get the Profile
            Profile p = [SELECT Id FROM Profile WHERE Name =: headless_profile LIMIT
1];

            //Create the Contact
            Contact c = new Contact();
            c.accountId = a.Id;
            c.firstName = u.firstName;
            c.lastName = u.lastName;
            insert(c);

            //Associate the Contact to the user along with the profile.
            u.profileId = p.Id;
            u.contactId = c.Id;

        } else {
            u.firstName = data.firstName;
            u.lastName = data.lastName;
            u.email = data.email;
            update u;
        }

        return u;

    }

    /*
     * Basic Update User Method
     * */
    global void updateUser(Id userId, Id portalId, Auth.UserData data){
        User u = new User(id=userId);
        u.email = data.email;
        u.lastName = data.lastName;
        u.firstName = data.firstName;
        update(u);
    }
```

```
   /*
 * This method handles filling user data that is required by Salesforce but is not
passed in during registration
 */
    void prepareUserData(Auth.UserData data, User u){

        String name, firstName, lastName, username, alias, email;

        System.debug('----> Passed In User Information');
        System.debug('Email: ' + data.email);
        System.debug('First Name: ' + data.firstName);
        System.debug('Last Name: ' + data.lastName);

        for(string key : data.attributeMap.keySet())
        {
            system.debug('key: ' + key + ' value: ' + data.attributeMap.get(key));
        }
        // Initialize the attributes essential for creating a new user with dummy
values
        // in case they will not be provided by the Auth Provider
        firstName = 'change-me';
        lastName = 'change-me';
        email = 'change@me.com';
        if(data.email != null && data.email != '')
            email = data.email;
        if(data.firstName != null && data.firstName != '')
            firstName = data.firstName;
        if(data.LastName != null && data.lastName != '')
            lastName = data.lastName;
        if(data.attributeMap.containsKey('full_name'))
            name = data.attributeMap.get('full_name');
        if(data.attributeMap.containsKey('name'))
            name = data.attributeMap.get('name');
        if(firstName == 'change-me' && name != '')
            firstName = name.substringBefore(' ');
        if(lastName == 'change-me' && name.substringAfter(' ') != '')
            lastName = name.substringAfter(' ');

        alias = firstName;

        //Alias must be 8 characters or less
        if(alias.length() > 8)
            alias = alias.substring(0, 8);
        u.username = email;
        u.email = email;
        u.lastName = lastName;
        u.firstName = firstName;
        u.alias = alias;
        u.languagelocalekey = UserInfo.getLocale();
        u.localesidkey = UserInfo.getLocale();
        u.emailEncodingKey = 'UTF-8';
        u.timeZoneSidKey = 'America/Los_Angeles';
    }
```

```
}
```

Configure a Google Authentication Provider

For this example, configure a Google authentication provider so that users can log in with Google.

Authentication providers require you to use an app on the external identity provider to communicate with Salesforce. You can either set up an app yourself or use the default global app provided by Salesforce. In production, we recommend that you create your own app. For testing and for this example, you can use the global app, so you're only required to fill in a few fields. Salesforce handles the rest.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

1. From Setup, in the Quick Find box, enter `Auth`, and then select **Auth. Providers**.
2. Click **New**.
3. From the Provider Type dropdown, select **Google**.
4. For the name, enter `Google IDP`.
5. Ensure that the URL suffix is `Google_IDP`.
6. For Registration Handler, click 🔍, and then search for the name of the registration handler Apex class that you created.
7. For Execute Registration As, click 🔍, and then search for a user to run the registration handler Apex class. This user must have the Manage Users permission. For this example, you can use your admin user.
8. Leave the rest of the values blank, and save your changes.

   In production, when you set up your own app on the third-party SSO provider, you must fill in these fields.

CHAPTER 4 Set Up an Experience Cloud Site for Headless Identity

In this chapter ...

- Enable Digital Experiences and Create Your Site
- Configure Experience Cloud Site Membership
- Create a Headless Registration Handler
- Configure Headless Identity Settings in Experience Cloud
- Activate Your Site

Because Headless Identity APIs are exposed through Experience Cloud, creating a site is a crucial part of configuring Headless Identity.

Through your Experience Cloud site, you control enablement and access for headless flows, including your site membership. You also control security settings for the Headless Registration Flow and Headless Forgot Password Flow.

If you already have an Experience Cloud site, we still recommend creating a new one for this example.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Enable Digital Experiences and Create Your Site

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Enabling digital experiences is the first step to creating your Experience Cloud site.

1. From Setup, in the Quick Find box, enter `Digital`, and then select **Digital Experiences | Settings**.
2. Select **Enable Digital Experiences**.
3. If enhanced domains are enabled in your org, you see your digital experiences domain name. Otherwise, enter a name, and then to make sure that it isn't being used, click **Check Availability**.
4. Save your changes.
5. From Setup, in the Quick Find box, enter `Sites`, and then select **All Sites**.
6. Click **New**.
7. Select a site template. For this example, select **Build Your Own (LWR)**.
8. Click **Get Started**.
9. Note: We recommend that you don't provide a URL suffix for your site. That way, you're not required to remember it every time you need the Experience Cloud domain.

   Enter a name for the site. For this example, name the site `Headless Demo`.
10. Click **Create**.
    Your site opens in Experience Workspaces.

Keep your site open in Experience Workspaces for the next step.

Configure Experience Cloud Site Membership

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Because your Experience Cloud site stores all your user information, you must add your end-user profile to the site's membership.

1. From Experience Workspaces, select **Administration**.

   If you're not in Experience Workspaces, here's how you can get back to it. In the Quick Find box, enter `Sites`, select **All Sites**, and then next to your site name, select **Workspaces**.
2. From the Administration page, select **Members**.
3. Search for the Headless Demo Profile that you created earlier, and add it to Selected Profiles. You can use the Search dropdown to filter profile types.
4. Save your changes.

Create a Headless Registration Handler

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Create an Apex class for your registration handler. You reference this Apex class when you configure Experience Cloud settings on the Login & Registration page.

1. From Setup, in the Quick Find box, enter *Apex*, and then select **Apex Classes**.
2. Click **New**.
3. Fill the class with your headless registration handler code. You can paste in the code from the example.
4. Save the class, and note its name.

Example: Here's an example headless registration handler. This registration handler creates a user and links it to the account and profile that you created earlier in this guide.

Note: This sample code is for demonstration only. Always test code before deploying it to a production environment.

```
/*
 * Sample Headless Self Registration Handler class for Headless
 Identity implementation
 */

global class HeadlessSelfRegistrationHandler implements
Auth.HeadlessSelfRegistrationHandler{
    static final String headless_account = 'My Account';

    // Creates a Standard salesforce or a community user
    global User createUser(Id profileId, Auth.UserData data,
 String customUserDataMap, String experienceId, String
password){
        User u = new User();
        //Ensures the user will save as all required fields
are pre-filled in with dummy values
        prepareUserData(data, u);

       //Get the Account, and create it if one is not already
 present.
        Account a;
       List<Account> accounts = [SELECT Id FROM Account WHERE
 name='My Account'];
        if(accounts.isEmpty()) {
            a = new Account(name = headless_account);
            insert(a);
        } else {
            a = accounts[0];
        }

        handleCustomData(customUserDataMap);

        // Create the Contact
        Contact c = new Contact();
        c.accountId = a.Id;
        c.firstName = u.firstName;
        c.lastName = u.lastName;
        insert(c);

        //Associate the Contact to the user along with the profile.
        u.profileId = profileId;
        u.contactId = c.Id;
        return u;
    }

    /*
     * We support the ability to pass in complex structures in the custom user data
map
     * You can build Apex classes that represent your complex structure
     * Then deserialize that structure into your Apex class
     * In this case we have a class at the bottom of this file called
"ContactInformation"
     * This class deserializes the incoming request and prints out the fields
     * */
    void handleCustomData(String customUserDataMap) {
        System.debug('Custom Data: ' + customUserDataMap);
        ContactInformation contactInfo = null;
        try {
            contactInfo =
(HeadlessSelfRegistrationHandler.ContactInformation)JSON.deserialize(customUserDataMap,
 HeadlessSelfRegistrationHandler.ContactInformation.class);
            System.debug('ContactInfo.mobilePhone: ' + contactInfo.mobilePhone);
            System.debug('ContactInfo.streetAddress: ' + contactInfo.streetAddress);

            System.debug('ContactInfo.city: ' + contactInfo.city);
            System.debug('ContactInfo.state: ' + contactInfo.state);
        } catch (Exception e) {
            System.debug('JSON was not formed correctly for the apex class');
        }



    }

    /*
 * This method handles filling user data that is required by Salesforce but is not
passed in during registration
    * It is not strictly necessary but helpful as it centralizes the management of
unnecessary fields to the IDP instead of the client.
 */
    void prepareUserData(Auth.UserData data, User u){

        String name, firstName, lastName, username, alias, email;

        System.debug('----> Passed In User Information');
        System.debug('Email: ' + data.email);
        System.debug('First Name: ' + data.firstName);
        System.debug('Last Name: ' + data.lastName);

        for(String key : data.attributeMap.keySet())
        {
            System.debug('key: ' + key + ' value: ' + data.attributeMap.get(key));
        }
        // Initialize the attributes required to create a new user with dummy values
        // in case they are not provided by the Auth Provider
        firstName = 'change-me';
        lastName = 'change-me';
        email = 'change@me.com';
        if(data.email != null && data.email != '')
            email = data.email;
        if(data.firstName != null && data.firstName != '')
            firstName = data.firstName;
        if(data.LastName != null && data.lastName != '')
            lastName = data.lastName;
        if(data.attributeMap.containsKey('full_name'))
            name = data.attributeMap.get('full_name');
        if(data.attributeMap.containsKey('name'))
            name = data.attributeMap.get('name');
        if(firstName == 'change-me' && name != '')
            firstName = name.substringBefore(' ');
        if(lastName == 'change-me' && name.substringAfter(' ') != '')
            lastName = name.substringAfter(' ');

        // Generate a random username
        Integer rand = Math.round(Math.random()*100000000);
        if(data.attributeMap.containsKey('username')){
            username = data.attributeMap.get('username');
        }else{
            username = lastName + '.' + rand + '@social-sign-on.com';
        }
        alias = firstName;

        //Alias must be 8 characters or less
        if(alias.length() > 8)
            alias = alias.substring(0, 8);
        u.username = username;
        u.email = email;
        u.lastName = lastName;
        u.firstName = firstName;
        u.alias = alias;
        u.languagelocalekey = UserInfo.getLocale();
        u.localesidkey = UserInfo.getLocale();
        u.emailEncodingKey = 'UTF-8';
        u.timeZoneSidKey = 'America/Los_Angeles';
    }

    /*
     * Apex Class Representation of Contact Information
     * which was passed in the custom data map
     * */
    global class ContactInformation {
        String mobilePhone;
        String streetAddress;
        String city;
        String state;
        Boolean privacyPolicy;
    }
}
```

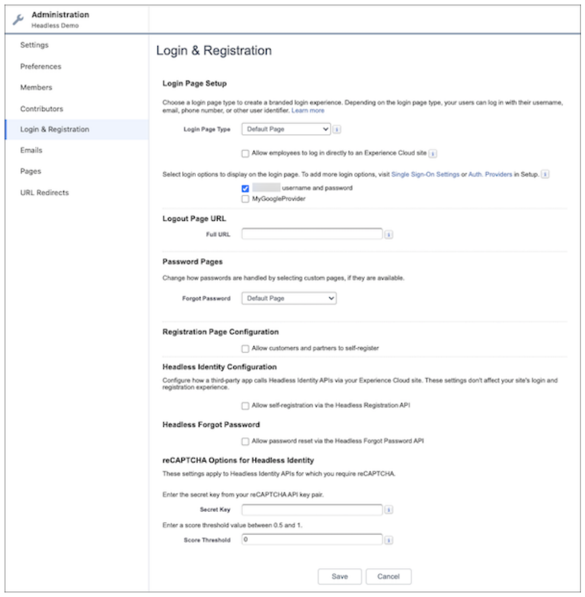
Configure Headless Identity Settings in Experience Cloud

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

You control enablement, access, and security for the Headless Registration Flow and the Headless Forgot Password Flow on the Experience Cloud Login & Registration page. For this example, configure settings to support headless identity for a single-page app.



Login & Registration

Login Page Setup

Choose a login page type to create a branded login experience. Depending on the login page type, your users can log in with their username, email, phone number, or other user identifier. Learn more

Login Page Type: Default Page

☐ Allow employees to log in directly to an Experience Cloud site

Select login options to display on the login page. To add more login options, visit Single Sign-On Settings or Auth. Providers in Setup.

☑ [redacted] username and password

☐ MyGoogleProvider

Logout Page URL

Full URL:

Password Pages

Change how passwords are handled by selecting custom pages, if they are available.

Forgot Password: Default Page

Registration Page Configuration

☐ Allow customers and partners to self-register

Headless Identity Configuration

Configure how a third-party app calls Headless Identity APIs via your Experience Cloud site. These settings don't affect your site's login and registration experience.

☐ Allow self-registration via the Headless Registration API

Headless Forgot Password

☐ Allow password reset via the Headless Forgot Password API

reCAPTCHA Options for Headless Identity

These settings apply to Headless Identity APIs for which you require reCAPTCHA.

Enter the secret key from your reCAPTCHA API key pair.

Secret Key:

Enter a score threshold value between 0.5 and 1.

Score Threshold: 0

Save Cancel

The Login & Registration page is also where you configure identity features for users who interact with your Experience Cloud site directly. Most of the settings on this page, including settings for login, logout, password, and registration pages, affect the identity experiences for your site users.

The settings in the Headless Identity Configuration section are separate and affect how your app calls Headless Identity APIs via your site. Headless Identity Configuration settings don't affect how users interact with your Experience Cloud site.

In general, only the settings in Headless Identity Configuration are relevant for setting up your implementation. There's one exception, though. To create a native single sign-on (SSO) experience in your app, you must add the SSO provider to the Experience Cloud login page. During the flow, the browser is briefly redirected to the login page URL so that Salesforce can check to see if the SSO provider is enabled. The browser is then redirected to the provider. The redirection happens so quickly that the user never sees the Experience Cloud login page. The experience feels like headless SSO, even though it technically isn't.

1. Go to the Login & Registration page. From Setup, in the Quick Find box, enter `Sites`, and then select **All Sites**. Next to your site name, click **Workspaces**, select **Administration**, and then select **Login & Registration**.
2. Add your SSO provider to the Experience Cloud login page. Under Login Page Setup, for login options, enable the Google IDP authentication provider you set up.
3. Enable headless registration.
   a. Select **Allow self-registration via the Headless Registration API**.
   Enabling this setting exposes other settings related to headless registration.
   b. Select **Require reCAPTCHA to access this API**, and leave **Require authentication to access this API** deselected.

These settings control whether you need extra information—either a reCAPTCHA token or an access token— in your initial POST request to Headless Registration API. When you configure headless registration, you must require either authentication or reCAPTCHA—you can't save your settings without at least one of these settings turned on. For this example, which is focused on single-page apps, requiring authentication isn't recommended. A single-page app submits the registration POST request via the browser, and it can't keep an access token safe. So requiring reCAPTCHA is the way to go.

c. For Default Profile, select the Headless Demo profile you set up. This profile gets assigned to new users automatically.

d. For Registration Handler, click 🔍, and then select your headless registration handler class.

e. For Run As, click 🔍, and then select your admin user.

This user runs the headless registration handler. They must be able to access the account that contains your end users, and they must be assigned to the Headless Identity Admin role that you created. For this example, you can select yourself as the Run As user to make testing easier. In production, select a user that isn't tied to a real person. That way, you don't experience service disruptions if someone leaves the company and their account is disabled.

4. Enable headless password reset.
   a. Select **Allow password reset via the Headless Forgot Password API**.
   b. Select **Require reCAPTCHA to access this API**, and leave **Require authentication to access this API** deselected.

      These settings work the same way for headless password reset as they do for headless registration. Similarly, we recommend requiring reCAPTCHA and not authentication for this example.
   c. For the maximum number of password reset attempts, keep the default of 5 attempts.
5. Configure reCAPTCHA options.

   The reCAPTCHA options apply to Headless Registration API and Headless Forgot Password API.
   a. For Secret Key, enter the secret key from your reCAPTCHA API key pair. You get this information from Google when you set up reCAPTCHA.
   b. For Score Threshold, enter 0.7.

      The score threshold is the lowest value that you accept for the reCAPTCHA score issued by Google. This score helps you determine whether new registration requests are valid. Scores closer to 0 are more likely to be bots, while scores closer to 1 are more likely to be valid users. The minimum score threshold that Salesforce allows is 0.5.
6. Save your settings.

Activate Your Site

To complete your Experience Cloud site setup for headless identity, activate your site. This important but often overlooked step is required to expose Headless Identity API endpoints.

1. From Administration workspaces, click **Settings**.
2. Click **Activate**.
3. Click **OK** to accept the warning.

   You receive an email when the site's activated.

Your Experience Cloud site is now fully prepared for headless identity.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

CHAPTER 5 Set Up a Connected App for Headless Identity

In this chapter ...

- Create an OAuth-Enabled Connected App
- Configure Connected App Policies
- Get Your Consumer Key

A connected app is a framework that allows your off-platform app to request data from Salesforce APIs. To integrate your single-page app with Headless Identity APIs, create a connected app and configure its settings and access policies.

As part of the connected app setup, you enable the Authorization Code and Credentials Flow. With this flow, you can set up headless login. Enabling this flow is also a prerequisite for enabling headless registration. You also define what Salesforce data your app can access with scopes and configure additional security settings and access policies. And you define a callback URL, which you use as the `redirect_uri` parameter during headless flows.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

After you create a connected app, you get a consumer key, or `client_id`. During headless flows, your app passes the consumer key to Salesforce to identify itself.

Create an OAuth-Enabled Connected App

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Create your connected app for headless identity, add the required scopes, and enable the Authorization Code and Credentials flow at the app level.

1. From Setup, in the Quick Find box, enter *`App`*, and then select **App Manager**.
2. Click **New Connected App**.
3. Enter a name for your connected app, such as *`Headless Demo App`*.
   The API Name autofills based on the name that you enter.
4. For Contact Email, enter your email address.
5. Under API (Enable OAuth Settings), select **Enable OAuth Settings**.
6. For Callback URL, enter *`https://MyExperienceCloudSite.my.site.com/services/oauth2/echo`*, where `https://MyExperienceCloudSite.my.site.com` is your Experience Cloud site domain.

   This URL points to the Salesforce OAuth 2.0 echo endpoint on your Experience Cloud site. The echo endpoint handles the code extraction step for headless login and headless registration, which saves you the work of writing and hosting your own code extraction endpoint. It returns the authorization code and other parameters from the 302 redirect as a JSON object that you can easily parse.
7. Add the **Manage user data via APIs (api)** and **Access unique user identifiers (openid)** scopes to Selected OAuth scopes.
8. Deselect the **Require Secret for Web Server Flow** and **Require Secret for Refresh Token Flow** settings.

   Because this example is focused on single-page apps, which can't keep information private, you must deselect these settings for security.
9. Select **Enable Authorization Code and Credentials Flow**.
   Enabling the flow exposes another setting to require user credentials in the POST body of your authorization request. Leave this setting deselected—again, your app can't keep this information secret.
10. Save your connected app settings.

Now that you have an OAuth-enabled connected app, you can get your consumer key and consumer secret.

Configure Connected App Policies

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

In a standard OAuth flow, users often see an approval screen where they confirm that an app is allowed to access their Salesforce data. With headless identity flows, you don't want to show users a Salesforce approval screen. To preapprove access, configure OAuth policies on your connected app.

1. Go to your connected app policy page.
   a. From Setup, in the Quick Find box, enter *`App`*, and then select **App Manager**.
   b. Next to your app, click ▾, and then select **Manage**.
2. Click **Edit Policies**.
3. Under OAuth Policies, set the Permitted Users policy to **Admin approved users are pre-authorized**.
4. Save the policy change.
5. On the connected app policy page, scroll down to and select **Manage Profiles**.

6. Select the headless demo profile that you created.
7. Save the policy change.

Get Your Consumer Key

When you create a connected app, it generates a consumer key, also known as a client ID. This value allows Salesforce to identify your third-party app during headless identity flows. Learn how to get your consumer key so that you can use it when configuring headless flows.

1. If you're not on the connected app detail page from the previous step, go to your app's page.
   a. From Setup, in the Quick Find box, enter *App*, and then select **App Manager**.
   b. Next to your app, click ▾, and then select **View**.
2. From the connected app detail page, click **Manage Consumer Details**.
   Salesforce prompts you to verify your identity with one of your registered methods.
3. Complete the identity verification challenge.
   When you complete the challenge, you can see your consumer key.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

CHAPTER 6 Use Postman to Work with Headless Identity APIs

In this chapter ...

- Set Up Your Postman Workspace
- Headless Registration Flow: Send a Registration Request
- Headless Registration Flow: Send an Authorization Request
- Headless Registration Flow: Send a Token Request
- Headless Login: Send an Authorization Request
- Headless Login: Send a Token Request
- Headless Registration and Headless Login Flows: Get User Info
- Headless Forgot Password Flow: Send a Password Reset Request
- Headless Forgot Password Flow: Change the User's Password
- Headlessly Revoke an Access Token

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Now that you configured org-wide settings, Experience Cloud settings, and a connected app, you can configure and test out the headless identity flows. To make this process easier, we provide Postman examples that are customized for this example single-page app implementation. Use these examples to walk through the flows for headless registration, headless login, and headless password reset. The collection also contains examples for calling the User Info endpoint and logging out a user.

Note: For simplicity, these examples don't use the OAuth 2.0 Proof Key for Code Exchange (PKCE) extension. For security, we strongly recommend that you always use PKCE when configuring these flows with public clients.

The examples in this collection are divided into four sections, each with examples representing key steps in the flow.

- Headless Registration Flow—This section contains three examples.
  - Registration - Initialize—Send a registration request to Headless Registration API.
  - Registration - Authorize—Send an authorization request to Headless Login API.
  - Registration - Token Exchange—Send an access token request to the token endpoint.
- Headless Login via the Authorization Code and Credentials Flow—This section contains two examples.
  - Username Password Login - Authorize—Send an authorization request to Headless Login API.
  - Username Password Login - Token Exchange—Send an access token request to the token endpoint.
- Confirming a successful login by calling the User Info endpoint—This section contains one example. You can use it to confirm the success of headless registration and headless login.
  - Get User Info—Send a request to the User Info endpoint.
- Forgot Password Flow—This section contains two examples.
  - Forgot Password - Initialize—Send the initial password reset request to Headless Forgot Password API.
  - Forgot Password - Change Password—Finish changing the password. with another request to Headless Forgot Password API.
- Logging out the user by revoking the access token—This section contains one example. You can use it to revoke the access token you get during headless registration and headless login.
  - Revoke Token—Send a request to the token revocation endpoint.

Set Up Your Postman Workspace

To get started with Postman, fork the public Salesforce Developers collection and set your variables.

To work with this collection, you must have a Postman account that's connected to Salesforce, meaning that the Salesforce Postman collection must be authorized to access your org. If you're using Postman in a browser, you must also add the Postman URL patterns to your Cross-Origin Resource Sharing (CORS) allowlist. For instructions on completing these steps, see the Quick Start: Connect Postman to Salesforce Trailhead module.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Note: If you're using the Postman desktop app, you can skip configuring your CORS allowlist.

1. If you haven't already, fork the Salesforce Platform APIs collection from the public Salesforce Developers workspace. By forking the collection, you can modify it on your local workspace without changing the parent version.
   a. Open the Salesforce Developers workspace.
   b. In Collections, select **Salesforce Platform APIs**, click ..., and then select **Create a Fork**.
   c. Name the fork, add it to your workspace, and click **Fork Collection**.
2. Reset the collection variables to use your own.
   a. Select the forked Salesforce Platform APIs collection in your workspace.
   b. Select the **Variables** tab.

      The variables here apply to the entire Salesforce Platform APIs collection, so only a few of them are relevant to this example. You can ignore the ones that aren't.
   c. Replace the current value for `url` with your Experience Cloud site domain, such as `MyExperienceCloudSite.my.site.com`.
   d. Replace the current value for `clientId` with your connected app consumer key.
   e. Replace the current value for `redirectUrl` with *`https://MyExperienceCloudSite.my.site.com/services/oauth2/echo`*, where `https://MyExperienceCloudSite.my.site.com` is your Experience Cloud site domain. This value must match your connected app callback URL.

   The variable list also contains a `site` variable, which you can use to store an Experience Cloud site suffix if your site has one. For this example implementation, we didn't add a suffix, so leave this value blank.
3. To see the Headless Identity examples, expand the Salesforce Platform APIs collection, select **Auth**, and then select **Headless Identity API Demo**.

Headless Registration Flow: Send a Registration Request

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

To configure the Headless Registration Flow, start with a request to Headless Registration API. This example walks you through sending a request with Postman.

For the Headless Registration Flow, we use these three examples from the headless Postman collection.

- Registration - Initialize
- Registration - Authorize
- Registration - Token Exchange

1. From the Headless Identity API Demo folder in Postman, select **Registration - Initialize**.

   Note the location of the POST request. It's the `/services/auth/headless/init/registration` endpoint on your Experience Cloud site.

2. To see the request body, click **Body**. The body for the initial registration request includes these parameters.
   - `userdata`—Contains basic information about the user, including their first name, last name, email address, and username.
   - `customdata`—Contains custom information that you want to collect in addition the data in the `userdata` parameter. In this example, it contains the user's mobile phone number, street address, city, state, ZIP code, and privacy policy.
   - `password`—The user's password.
   - `recaptcha`—A reCAPTCHA token. Because you configured your flow to require reCAPTCHA, this parameter is required.
   - `verificationmethod`—The method you want to use to verify the user's identity. They can use email or SMS. For this example, we use email.

3. For the request body, update the parameter values that aren't filled out already.
   a. In the `userdata` parameter, change `userFirstName` and `userLastName` to your own first name and last name.
   b. For `email`, enter your own email address.
   c. For `username`, enter a test username. You can use your email address for the username.
   d. For `recaptcha`, enter a valid reCAPTCHA token from Google.

      Generating this token is out of the scope of this guide. Google provides several examples, which you can find at https://developers.google.com/recaptcha/docs/v3.

4. To send the request to Headless Registration API, click **Send**.
   Salesforce receives the request and queues the user data to be processed later. It creates an identifier to track the request. If the request is successful, Salesforce sends you a response that includes your verification method and request identifier. Here's an example response from Postman.

```
{
    "status": "success",
    "email": "testemail@example.com",
    "identifier": "dg3o**********"
}
```

   Because you configured email as the verification method, Salesforce also sends an email containing a one-time password (OTP) to your email address.

5. Check your email for the OTP. You use it in the next step.

Next, you send an authorization request to initialize the Authorization Code and Credentials Flow and log the user in.

Headless Registration Flow: Send an Authorization Request

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

After you send a registration request to Headless Registration API, initialize the Authorization Code and Credentials Flow to complete the registration and log the user in. For this part of the flow, you call Headless Login API.

1. From the Headless Identity API Demo folder in Postman, select **Registration - Authorize**.

   Note the location of the POST request. It's the `/services/oauth2/authorize` endpoint on your Experience Cloud site.

2. To see the headers, click **Headers**. The authorization request for registration includes these headers and values.
   - An `Auth-Request-Type` header set to `user-registration`
   - An `Auth-Verification-Type` header set to `email`. This header specifies the method that was used to verify the user's identity—note that its value matches the `verificationmethod` body parameter from your initial registration request.
   - An `Authorization` header with the value `Basic <base64Encoded identifier:otp>`

3. Update the Authorization header.
   a. In a text editor, paste the request ID from the registration response from the previous step.
   b. After the request ID, enter a colon, and then paste the one-time password (OTP) that you received from Salesforce, such as `identifier:OTP`.
   c. Base64-encode the resulting value. For example, if you Base64-encode the string `identifier:OTP`, you get `aWRlbnRpZmllcjpPVFA=`.
   d. In Postman, paste this value into the Authorization header to replace `<base64Encoded identifier:otp>`.

4. To see the request body, click **Body**. The authorization request for registration includes these parameters, some of which you already entered when you set your variables.
   - `response_type`—The type of response you want to receive. For this flow, it's set to `code_credentials`.
   - `client_id`—The connected app consumer key
   - `redirect_uri`—The connected app callback URL, which points to the OAuth 2.0 echo endpoint on your Experience Cloud site
   - `scope`—An optional comma-separated list of scopes. For this demo, you can leave it blank.

5. To send the request to Headless Login API, click **Send**.
   Salesforce verifies the request ID and OTP and uses the request ID to retrieve the queued user data from the registration request. Salesforce then calls the headless Apex registration handler that you configured in your Experience Cloud site. The registration handler uses the queued data to create a user.

   If the request is successful, Salesforce returns a 302 redirect to a preconfigured URL containing the authorization code. The echo endpoint extracts the code and other parameters from the 302 redirect and returns them to your app in JSON format. Here's an example response in Postman.

```
{
    "code": "aPrxCdr***************",
    "sfdc_community_url": "https://MyExperienceCloudSite.my.site.com",
    "sfdc_community_id": "0DBXXXXXXXXXXXXXXX"
}
```

Next, you exchange the authorization code for an access token.

Headless Registration Flow: Send a Token Request

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

After you receive the authorization code from Salesforce, exchange the code for an access token. This example walks you through initializing the code exchange with Postman.

1. From the Headless Identity API Demo folder in Postman, select **Registration - Token Exchange**.

   Note the location of the POST request. It's the `/services/oauth2/token` endpoint on your Experience Cloud site.

2. To see the request body, click **Body**. The token request for registration includes these parameters.
   - `code`—The authorization code from Salesforce
   - `grant_type`—Defines the OAuth 2.0 grant type. Because the OAuth 2.0 authorization code grant type is the foundation of this flow, this parameter is set to `authorization_code`.
   - `client_id`—The connected app consumer key
   - `redirect_uri`—The connected app callback URL, which points to the OAuth 2.0 echo endpoint on your Experience Cloud site

3. For the `code`, enter the authorization code that you received after sending the authorization request.

4. To send the request to the token endpoint, click **Send**.
   Salesforce validates the token request and returns an access token to your app.

Headless Login: Send an Authorization Request

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

To configure headless login, set up the Authorization Code and Credentials Flow. This Postman example walks you through the Authorization Code and Credentials Flow with a single-page app.

1. From the Headless Identity API Demo folder in Postman, select **Username Password Login - Authorize**.

   Note the location of the POST request. It's the `/services/oauth2/authorize` endpoint on your Experience Cloud site.

2. To see the headers, click **Headers**. The authorization POST request includes these headers and values.
   - An `Auth-Request-Type` header set to `Named-User`
   - An `Authorization` header with the value `Basic <username:password>`, which contains the Base64-encoded username and password value

3. Update the Authorization header.

   a. In a text editor, paste the username that you registered during headless registration.

   b. After the username, enter a colon, and then paste the password that you registered, such as `username:password`.

  c. Base64-encode the resulting value. For example, if you Base64-encode the string `username:password`, you get `dXNlcm5hbWU6cGFzc3dvcmQ=`.
  d. In Postman, paste this value into the Authorization header to replace `<username:password>`.

4. To see the request body, click **Body**. The authorization request for registration includes these parameters.
   - `response_type`—The type of response you want to receive. For this example, it's set to `code_credentials`.
   - `client_id`—The connected app consumer key
   - `redirect_uri`—The connected app callback URL, which points to the OAuth 2.0 echo endpoint on your Experience Cloud site

5. To send the request to Headless Login API, click **Send**.
   Salesforce validates the user credentials and returns a 302 redirect to a preconfigured URL containing the authorization code. Salesforce then automatically sends the redirect response to the redirect URL. The echo endpoint extracts the code and other parameters from the 302 redirect and returns them to your app in JSON format. Here's an example response in Postman.

```
{
    "code": "aPrxCdr***************",
    "sfdc_community_url": "https://MyExperienceCloudSite.my.site.com",
    "sfdc_community_id": "0DBXXXXXXXXXXXXXXX"
}
```

Headless Login: Send a Token Request

After you send the authorization request and get a code, exchange the code for an access token.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

1. From the Headless Identity API Demo folder in Postman, select **Username Password Login - Token Exchange**.
   Note the location of the POST request. It's the `/services/oauth2/token` endpoint on your Experience Cloud site.

2. To see the request body, click **Body**. The token request for headless login includes these parameters.
   - `code`—The authorization code from Salesforce.
   - `grant_type`—Defines the OAuth 2.0 grant type. Because the OAuth 2.0 authorization code grant type is the foundation of this flow, this parameter is set to `authorization_code`.
   - `client_id`—The connected app consumer key.
   - `redirect_uri`—The connected app callback URL, which points to the OAuth 2.0 echo endpoint.

3. For the `code`, enter the authorization code that you received in your response from the previous step.

4. To send the request to the token endpoint, click **Send**.
   Salesforce validates the token request and returns a response to your app. The response contains an access token that can be used to access Salesforce APIs and other identifying parameters. Here's an example access token response in Postman.

```
{
    "access_token": "00DR**********",
    "sfdc_community_url": "https://MyExperienceCloudSite.my.site.com",
    "sfdc_community_id": "0DBXXXXXXXXXXXXXXX",
    "signature": "CPk2JprUcxOmRHND71gJFn+SxyKe7jWqA1rQnFz9zZg=",
    "token_format": "opaque",
    "scope": "openid api",
    "id_token": "eyJr**************",
    "instance_url": "https://yourInstance.salesforce.com",
    "id": "https://yourInstance.salesforce.com/id/00Dxxxxxxxxxxxx/005xxxxxxxxxxxx",
    "token_type": "Bearer",
    "issued_at": "1667600739962"
}
```

Headless Registration and Headless Login Flows: Get User Info

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Both headless registration and headless login are built on the Authorization Code and Credentials Flow, where you exchange an authorization code for an access token. After your app receives the access the token, the next step is processing it and creating the user session. This example walks you through calling the User Info endpoint to confirm that the login was successful and to provide the user information required for your app to create a session. You can use this example for both headless registration and headless login.

1. From the Headless Identity API Demo folder in Postman, select **Get User Info**.

   Note the location of the POST request. It's the `/services/oauth2/userinfo` endpoint on your Experience Cloud site.

2. To see the headers, click **Headers**. The request for User Info includes these headers and values.
   - An `Authorization` header with the value `Bearer <Token>`
   - A `Content-Type` header set to `application/json`

3. For the `Authorization` header, replace `<token>` with the access token that you received during headless registration or headless login, depending on which process you're testing.

4. To send the request to the User Info endpoint, click **Send**.
   If the login was successful, you get a response containing information about the user.

Headless Forgot Password Flow: Send a Password Reset Request

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

To initialize the Headless Forgot Password Flow, send a password reset request. This example walks you through sending a request with Postman.

1. From the Headless Identity API Demo folder in Postman, select **Forgot Password - Initialize**.

   Note the location of the POST request. It's the `/services/auth/headless/forgot_password` endpoint on your Experience Cloud site.

2. To see the request body, click **Body**. The password reset request includes these parameters.
   - `username`—The user's registered username
   - `recaptcha`—A reCAPTCHA token. Because you configured your flow to require reCAPTCHA , you must include this parameter.

3. Replace the body parameter values with your own information.

   a. For `username`, enter the username that you registered and logged in with.

   b. For `recaptcha`, enter a valid reCAPTCHA token from Google. You get this token when you implement reCAPTCHA on your app.

4. To send the request to Headless Forgot Password API, click **Send**.
   If the request is successful, Salesforce sends a one-time password (OTP) to your email address. Here's an example response in Postman. To avoid leaking user information, Salesforce always returns this response if reCAPTCHA validation succeeds, even if the user doesn't exist.

```
{
    "status_code": "otp_sent"
}
```

5. Check your email to confirm that you received the OTP.

Next, use the OTP to finish changing your password.

Headless Forgot Password Flow: Change the User's Password

After you receive a one-time password (OTP) from Salesforce, you can change your password with a new request to the forgot password endpoint. This example walks you through changing the password in Postman.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

1. From the Headless Identity API Demo folder in Postman, select **Forgot Password - Change Password**.

   Note the location of the POST request. It's the `/services/auth/headless/forgot_password` endpoint on your Experience Cloud site, which is the same endpoint you used for the initial reset request.

2. To see the request body, click **Body**. The password reset request includes these parameters.
   - `username`—The user's registered username
   - `newpassword`—The user's new password.
   - `otp`—The one-time password (OTP) sent to the user's email

3. Replace the body parameter values with your own information.

   a. For `username`, enter the username that you registered and logged in with.

   b. For `newpassword`, enter a new password that includes uppercase and lowercase characters, a number, and a special character.

   c. For `otp`, enter the OTP that you received from your initial request to the forgot password endpoint.

4. To send the request to Headless Forgot Password API, click **Send**.
   Salesforce validates the OTP and finishes resetting the password. If the request is successful, Salesforce sends a success response. Here's an example response in Postman.

```
{
    "status_code": "success"
}
```

Headlessly Revoke an Access Token

To log out of your headless app, you can revoke the access token. This example walks you through headlessly revoking a token with Postman.

1. From the Headless Identity API Demo folder in Postman, select **Revoke Token**.

   Note the location of the POST request. It's the `/services/oauth2/revoke` endpoint on your Experience Cloud site, which is the same endpoint that you used for the initial reset request.

2. To see the request body, click **Body**. The revocation request includes only a `token` parameter.
3. For the `token` parameter, replace `<Token>` with an access token that you obtained during headless registration or headless login.
4. To send the request to the revocation endpoint, click **Send**.
   If the request is successful, you're logged out.

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

CHAPTER 7 JavaScript Examples for Headless Identity APIs

In this chapter ...

- Headless Login JavaScript Examples
- Headless Registration JavaScript Examples
- Headless Forgot Password JavaScript Examples
- Building a Native Single Sign-On Experience JavaScript Examples
- Getting User Info JavaScript Examples

EDITIONS

Available in: both Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions

Postman is great for testing and learning how to set up headless identity flows, but it doesn't show you how your app can interact with Headless Identity APIs. Use these high-level JavaScript examples to understand how your app can call these APIs in a real-world implementation. Like the rest of this guide, the examples here apply to single-page apps, also known as public clients. These examples don't show you how to use these flows with client-server apps or private clients.

These examples are for demonstration only and aren't meant to be used in production. Always test code before deploying it to a production environment.

Unlike the Postman examples, these examples use the Proof Key for Code Exchange (PKCE) extension, which improves security. We strongly recommend that you always use PKCE when configuring these flows with public clients.

The examples present a simplified overview of each headless flow for public clients. For in-depth descriptions of the flows for both public and private clients, see these resources in Salesforce Help.

- Headless Identity APIs for Customers and Partners
- Authorization Code and Credentials Flow for Public Clients
- Authorization Code and Credentials Flow for Private Clients
- Headless Registration Flow for Public Clients
- Headless Registration Flow for Private Clients
- Headless Forgot Password Flow—You can use this resource for public and private clients.

Headless Login JavaScript Examples

Use these high-level examples to understand how to implement headless login for a single-page app.

Authorization Request

The Authorization Code and Credentials Flow enables users to log in with a username and password. A core concept of this flow is making an authorization request to Headless Login API. This client-side JavaScript example shows you how to send an authorization request.

```
// Make a POST Request to Authorize
client = new XMLHttpRequest();
client.open("POST", expDomain + authorizationURI, true);
client.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

//Headers for the Username and Password Code and Cred Flow
client.setRequestHeader("Auth-Request-Type", "Named-User");
client.setRequestHeader("Authorization", "Basic " + btoa(username + ':' + password));

//Request Body
requestBody = "response_type=code_credentials&client_id=" +  clientId + "&redirect_uri="
+ callbackURL;

// Add State
if (state != null) {
  requestBody = requestBody + '&state=' + storeState(state);
}

// Add Scopes
if (scopes != null) {
  requestBody = requestBody + '&scope=' + scopes;
}

// PKCE Enabled
requestBody = requestBody + "&code_challenge=" + generateCodeChallenge();

// Send the Authorization Request
client.send(requestBody);

// Handle the Authorization Response
client.onreadystatechange = function() {
  if(this.readyState == 4) {
    if (this.status == 200) {
        //Auth Code has been returned, perform token exchange
      tokenExchange(JSON.parse(client.response), null, authorizeType, uniqueVisitorId);

    } else {
      onError("An Error Occured during Authorize", client.response);
    }
  }
}
```

Token Exchange

After receiving the authorization request, Salesforce validates the username and password and returns an authorization code. The app then calls the token endpoint to exchange the code for an access token. This client-side JavaScript example shows the token exchange.

```
function tokenExchange(response, codeChallenge, authorizeType, uniqueVisitorId) {
  // Get Values from Code Response
  code = response.code;
  stateIdentifier = response.state;
  baseURL = response.sfdc_community_url;

  state = null;
  // validate state if it was present
  if (stateIdentifier != null) {
  state = getState(stateIdentifier, true);
    if (state == null) {
    onError("A state param was sent back but no state was found");
    return;
    }
  }

  // Create Client
  client = new XMLHttpRequest();
  client.open("POST", expDomain + tokenURI, true);
  client.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

  // Build Request Body
  requestBody = "code=" + code + "&grant_type=authorization_code&client_id=" + _this.clientId
 + "&redirect_uri=" + _this.callbackURL;

  // Add PKCE
  requestBody = requestBody + "&code_verifier=" + generateCodeVerifier();

  // Send Request
  client.send(requestBody);
  client.onreadystatechange = function() {
    if(this.readyState == 4) {
      if (this.status == 200) {
        //Access Tokens have been returned
        console.log("Code and Credntial Flow, token response: ");
        console.log(JSON.parse(client.response));
      } else {
          onError("An error occured during token exchange for " + authorizeType,
client.response)
      }
    }
  }
}
```

The result is an access token that you can use to request user information and establish the user's session.

Headless Registration JavaScript Examples

Use these high-level examples to understand how to implement headless registration for a single-page app.

Authorization Request

The authorization request for headless registration is similar to the request for headless login, but instead of passing the username and password in the header, you pass in the request identifier and OTP. The request also contains a few additional headers that aren't required for headless login. Here's an example.

```
  // Make a POST Request to Authorize
  client = new XMLHttpRequest();
  client.open("POST", expDomain + authorizationURI, true);
  client.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

  //Headers for registration variation of the Code and Credentials flow
  client.setRequestHeader("Auth-Request-Type", "user-registration");
  client.setRequestHeader("Auth-Verification-Type", verificationMethod);
  //Request identifier is returned from the /init/registration endpoint, and
requestCredential is the OTP sent in the email or SMS.
  client.setRequestHeader("Authorization", "Basic " + btoa(requestIdentifier + ':' +
requestCredential));

  //Request Body
  requestBody = "response_type=code_credentials&client_id=" +  clientId + "&redirect_uri="
 + callbackURL;

  // Add State
  if (state != null) {
    requestBody = requestBody + '&state=' + storeState(state);
  }

  // Add Scopes
  if (scopes != null) {
    requestBody = requestBody + '&scope=' + scopes;
  }

  // PKCE Enabled
  requestBody = requestBody + "&code_challenge=" + generateCodeChallenge();

  // Send the Authorization Request
  client.send(requestBody);

  // Handle the Authorization Response
  client.onreadystatechange = function() {
    if(this.readyState == 4) {
      if (this.status == 200) {
        //Auth Code has been returned, perform token exchange
        tokenExchange(JSON.parse(client.response), null, authorizeType, uniqueVisitorId);

      } else {
      onError("An Error Occured during Authorize", client.response);
      }
    }
  }
```

Salesforce validates the request identifier and OTP and registers the user by calling the headless registration handler. When the user is created, Salesforce returns an authorization code response. You use a callback endpoint to extract the code and other parameters and return them to your app.

Token Exchange

When your app receives the authorization code, it exchanges the code for an access token. This part of the flow is identical to the token exchange in headless login.

```
function tokenExchange(response, codeChallenge, authorizeType, uniqueVisitorId) {
  // Get Values from Code Response
  code = response.code;
  stateIdentifier = response.state;
  baseURL = response.sfdc_community_url;

  state = null;
  // validate state if it was present
  if (stateIdentifier != null) {
    state = getState(stateIdentifier, true);
    if (state == null) {
      onError("A state param was sent back but no state was found");
      return;
    }
  }

  // Create Client
  client = new XMLHttpRequest();
  client.open("POST", expDomain + tokenURI, true);
  client.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

  // Build Request Body
  requestBody = "code=" + code + "&grant_type=authorization_code&client_id=" + _this.clientId
 + "&redirect_uri=" + _this.callbackURL;

  // Add PKCE
  requestBody = requestBody + "&code_verifier=" + generateCodeVerifier();

  // Send Request
  client.send(requestBody);
  client.onreadystatechange = function() {
    if(this.readyState == 4) {
      if (this.status == 200) {
          //Access Tokens have been returned
          console.log("Code and Credntial Flow, token response: ");
          console.log(JSON.parse(client.response));
        } else {
          onError("An error occured during token exchange for " + authorizeType,
client.response)
        }
      }
    }
  }
```

The result is an access token that you can use to request user information and establish the user's session.

Headless Forgot Password JavaScript Examples

Use these high-level examples to understand how to implement the Headless Forgot Password Flow for a single-page app.

The Headless Forgot Password Flow contains two requests to the same endpoint. You initialize the flow with a request to Headless Forgot Password API. Here's a function that you can call to initialize the password reset.

```
//This is the function call to initialize the forgot password request
forgotPasswordRequest(username, null, null, recapchaToken, forgotPasswordProcess.init,
callbackFunction);
```

This request results in Salesforce sending the user a one-time password (OTP).

For the second request, you pass the username, new password, and OTP to Headless Forgot Password API. Here's a function that you can call to complete the password change.

```
//This is the function call to complete the forgot password request
forgotPasswordRequest(username, password, otp, null, forgotPasswordProcess.changePassword,
 callbackFunction)
```

Because both requests call the same endpoint, you can use one function for both calls.

```
function forgotPasswordRequest(username, password, otp, recapchaToken,
forgotPasswordProcessStep, callbackFunction) {
    client = new XMLHttpRequest();
    client.open("POST", expDomain + forgotPasswordURI, true);
    client.setRequestHeader("Content-Type", "application/json");

    requestBody = {
    username: username,
    newpassword: password,
    otp: otp,
    recaptcha: recapchaToken
    }

    client.send(JSON.stringify(requestBody));

    client.onreadystatechange = function() {
      if(this.readyState == 4) {
        if (this.status == 200) {
         callbackFunction(JSON.parse(client.response), username, forgotPasswordProcessStep)

        } else {
          this.onError("An Error Occured during Forgot Password Step: " +
forgotPasswordProcessStep, client.response);
        }
      }
    }
  }
```

Building a Native Single Sign-On Experience JavaScript Examples

Use these high-level examples to understand how to create a native single sign-on (SSO) experience for a single-page app. This configuration uses a redirect-based flow to make it seem like your app natively integrates with an SSO provider. It isn't technically headless, but the user experience is the same as the headless flows.

Constructing an Authorization URL

A key part of this configuration is the `sso-provider` parameter. You use this parameter to identify the SSO provider configured in Salesforce, whether it's an authentication provider or a SAML identity provider. During the redirect-based flow, Salesforce checks for this parameter and redirects to the SSO provider so the user can log in.

To use the `sso-provider` parameter with a redirect-based flow, you must first construct an authorization URL. This example constructs the URL and redirects the browser to the Experience Cloud site.

```
//Setup the Authorization URL
  redirectURL = expDomain + _authorizationURI;

  //Add Params to the Authorization URL
  redirectURL = redirectURL + '?client_id=' + _this.clientId;
  redirectURL = redirectURL + '&redirect_uri=' + _this.ssoCallbackURL;
  redirectURL = redirectURL + '&state=' + storeState(state);
  redirectURL = redirectURL + '&response_type=code';

  //Specificy the SSO Provider
  redirectURL = redirectURL + '&sso_provider=' + ssoProviderDevName;

  //Add Scopes
  if (scopes!= null) {
      redirectURL = redirectURL + '&scopes=' + scopes;
  }

  //Add Code Challenge
  requestBody = requestBody + "&code_challenge=" + generateCodeChallenge();

  //Redirect the Browser
  window.location.href = redirectURL;
```

The browser is redirected to the Experience Cloud site briefly, so the user never sees the Experience Cloud login page. The browser then automatically redirects to the SSO provider and loads the provider's login page. The user logs in with their credentials from the provider. The browser is again briefly redirected to Salesforce before being automatically redirected to your app.

Token Exchange

The app must process the redirect to get the authorization code.

```
 // Get URL Params from the callback URL
  queryString = window.location.search;
  urlParams = new URLSearchParams(queryString);
  console.log('Loading Callback Params: ' + urlParams);

  //Create the Code Response from the URL params
  codeResponse = new Object;
  codeResponse.code = urlParams.get('code');
  codeResponse.state = urlParams.get('state');
  codeResponse.sfdc_community_url = urlParams.get('sfdc_community_url');

  // Call the common token exhcange method.
  tokenExchange(codeResponse, getCodeChallenge(), authorizationType.SSOLogin, null);
```

In the last line, this example calls a token exchange function. Here's an example of this function.

```
function tokenExchange(response, codeChallenge, authorizeType, uniqueVisitorId) {
  // Get Values from Code Response
  code = response.code;
  stateIdentifier = response.state;
  baseURL = response.sfdc_community_url;

  state = null;
  // validate state if it was present
  if (stateIdentifier != null) {
    state = getState(stateIdentifier, true);
    if (state == null) {
      onError("A state param was sent back but no state was found");
      return;
    }
  }

  // Create Client
  client = new XMLHttpRequest();
  client.open("POST", expDomain + tokenURI, true);
  client.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

  // Build Request Body
  requestBody = "code=" + code + "&grant_type=authorization_code&client_id=" + _this.clientId
+ "&redirect_uri=" + _this.callbackURL;

  // Add PKCE
  requestBody = requestBody + "&code_verifier=" + generateCodeVerifier();

  // Send Request
  client.send(requestBody);
  client.onreadystatechange = function() {
    if(this.readyState == 4) {
      if (this.status == 200) {
          //Access Tokens have been returned
          console.log("Code and Credntial Flow, token response: ");
          console.log(JSON.parse(client.response));
        } else {
          onError("An error occured during token exchange for " + authorizeType,
client.response)
        }
      }
    }
  }
```

Getting User Info JavaScript Examples

When you get an access token via headless registration or headless login, you can retrieve user information with a request to the User Info endpoint. Use this example to understand how.

In this example, the access token is passed in an Authorization Bearer header.

```
 function getUserInfo(accessToken) {
    client = new XMLHttpRequest();
    client.open("GET", expDomain + userInfoURI, true);
    client.setRequestHeader("Content-Type", "application/json");
    client.setRequestHeader("Authorization", 'Bearer '  + accessToken);
    client.send();

    client.onreadystatechange = function() {
        if(this.readyState == 4) {
            if (this.status == 200) {
                //User Info response
                console.log(client.response);
            } else {
                console.log(client.response)
                onError("An Error Occured during Forgot Password Step: " +
forgotPasswordProcessStep, client.response);
            }
        }
    }
}
```