
Salesforce External Identity Implementation Guide

Salesforce, Winter '19



CONTENTS

EXTERNAL IDENTITY FOR CUSTOMERS AND PARTNERS GUIDE	1
HOW CAN I USE SALESFORCE IDENTITY FOR EXTERNAL USERS?	2
Acquire and Engage New Users	2
Deliver a Consistent Experience with a Single Identity Across All Channels	3
Secure and Manage Your Customer and Partner Ecosystems	3
Integrate and Customize to Your Business Needs	4
Extend External Identity to Your Website	4
What Is Salesforce Identity for External Users?	4
External Identity Licenses	4
What Features Are Available with the External Identity License	5
External Identity and Communities	6
Learn More with Trailhead	7
EXTERNAL IDENTITY IMPLEMENTATION CHECKLIST	8
CREATE A TYPICAL EXTERNAL IDENTITY IMPLEMENTATION	10
Prepare Your Org	10
Create a Developer Org	11
Set Up My Domain	11
Control Authorization with Custom Profiles and Roles	11
Create an Account for CRM Integration	12
Create a Branded Login Page	12
Use the Aloha Template to Create Your Community	13
Control External User Access to Your Community	15
Brand Your Community Using the Login & Registration Page	16
Customize Your Login Pages with Community Builder	19
Activate Your Community	21
Enable Self-Registration	21
Add a Self-Registration Page	21
Add Fields to Collect Additional Information	22
Add a Password Field to Enable Login Directly During Registration	24
Enable Self-Registration for B2C Users (Optional)	25
Enable Person Accounts	26
Configure Self-Registration for Person Accounts	26
SET UP SSO FOR YOUR USERS	28
Social Sign-On	28
Create an Auth. Provider	29

Customize Your Registration Handler	29
Enable Your Auth. Provider in Your Community	30
Accept User Identity with SAML and Just-In-Time Provisioning	30
SET UP SSO FOR YOUR APPS	31
Set Up SSO and Access for Mobile Apps	31
Create a Connected App for Your Mobile App	31
Install the Salesforce Mobile SDK	32
Create a Mobile App	32
Configure the Mobile App to Point to Your Community	32
More About Single Sign-On for Your Mobile App	33
Set Up SSO and Access for Your Web App	33
Create a Connected App for Your Web App	34
Create a Sample Service Provider on Heroku	34
Configure Salesforce Identity to Provide Identity for Your App	35
Authorize Your Web App	35
Configure Your App to Trust Salesforce Identity	35
Personalize Your App with Custom Attributes	36
More About Single Sign-On for Your Web App	36
QUICKER AND EASIER LOGINS	37
Choose a Login Page Type	38
Use the Discoverable Login Page for Simpler Login	39
Extend the Login Discovery Handler in Apex	41
Choose a Self-Registration Page Type	43
Use the Configurable Self-Reg Page for Easy Sign-Up	44
Extend the Configurable Self-Reg Handler in Apex	46
View Your Users' Identity Verification Methods	47
Create Custom Error Messages in the Login Discovery and Self-Registration Handlers	48
DYNAMIC URLS: DETERMINE YOUR BRAND AT RUN TIME	50
Dynamic Branding for Multiple Brands	50
Dynamic Branding Using the Login & Registration Page	52
Set Up Dynamic URLs	53
CUSTOMIZE LOGIN PAGES IN APEX FOR FULL CONTROL	54
Create a Custom Login Page	55
Create a Custom Login Page from a Github Project	57
Dynamic Branding Using Custom Login Pages	58
Extend an Endpoint with the Experience ID	59
Passwordless Login Page Using Custom Login Pages	60
Passwordless Login Coding Considerations	62

EMBEDDED LOGIN: ALLOW VISITORS TO LOG IN TO YOUR WEB

PAGES	64
Embedded Login in Action	65
How to Implement Embedded Login	68
Implement Embedded Login	69
Step 1: Enable Resource Sharing Across Domains	70
Step 2: Create the Embedded Login Connected App	71
Step 3: Enable Embedded Login on a Web Page	73
Step 4: Write Login and Logout Functions	77
Step 5: Handle the Embedded Login Callback	78
Embedded Login Advanced Authentication Features	83
Embedded Login Considerations	85
Embedded Login Meta Tag Reference	86

MANAGE COMMUNITIES FOR EXTERNAL IDENTITY 90

Extend External Identity Sessions (Beta)	90
Create Lightweight Users Contactless Users (Beta)	91
Upgrade Users Without Contacts to a Community License (Beta)	92

EXTERNAL IDENTITY ON GITHUB AND TRAILBLAZER COMMUNITY 94

INDEX 95

EXTERNAL IDENTITY FOR CUSTOMERS AND PARTNERS GUIDE

Salesforce Identity for Customers and Partners improves your engagement with your external users. With little effort on your part, you can provide your external users a login page that reflects your brand. You can further customize the login process with the provided tools. This product is similar to Salesforce Identity but intended for your external users, whereas Salesforce Identity maintains the identity of your internal users.

Learn how to set up and manage a community for your external users. For brevity, Salesforce Identity for Customers and Partners is referred to as external identity.

HOW CAN I USE SALESFORCE IDENTITY FOR EXTERNAL USERS?

Salesforce Identity for Customers and Partners offers a broad set of capabilities for connecting with your customers and partners, as well as extensive customization and integration options. Here are some common use cases and features.

[Acquire and Engage New Users](#)

Your business is growing, and you need to onboard customers and partners quickly. Salesforce Identity can help enable and scale your customer and partner acquisition processes with self-registration, social sign-on, and CRM integration.

[Deliver a Consistent Experience with a Single Identity Across All Channels](#)

Salesforce Identity lets you engage with your users everywhere. Get a single, 360-degree view of your users while delivering a consistent, streamlined end-user experience for your brand.

[Secure and Manage Your Customer and Partner Ecosystems](#)

By centralizing management of your users, Salesforce Identity makes life easy for your admins. They have a single place to manage Identity users and create reports and dashboards on their access.

[Integrate and Customize to Your Business Needs](#)

Salesforce Identity is integrated into the Salesforce Platform and is fully customizable, extensible, and scalable for any business.

[Extend External Identity to Your Website](#)

Salesforce Identity Embedded Login makes it easy to incorporate authentication into websites. Creating authenticated sessions between your community and website visitors extends your reach with your customers. For example, you can require that your users log in before they access your website. Or when customers change billing information on the website, Salesforce can update their contact information. As you collect information about a user, you can tailor the experience accordingly.

[What Is Salesforce Identity for External Users?](#)

Salesforce Identity is an Identity and Access Management (IAM) service that connects users to your apps, services, and devices. It provides a centralized point of management for your admins and provides a single, trusted identity for your end users. Traditionally, IAM services have focused on employee-facing use cases. Today, companies are using identity as a way to better connect with their customers and partners. We call this external identity.

[External Identity Licenses](#)

External Identity is a type of Salesforce license that enables you to deliver identity services, such as single sign-on (SSO) and social sign-on. External Identity is a standalone license and purchased in blocks of active users. These users are typically consumers of your business, such as purchasers, patients, partners, and dealers.

[External Identity and Communities](#)

Communities are branded spaces for employees, customers, and partners to connect. You can customize and create communities to deliver specific business apps and services, including identity services.

[Learn More with Trailhead](#)

For a fun way to get a better understanding of external identity, take advantage of Trailhead. You can find identity modules in the Secure Identity and Access Management trail.

Acquire and Engage New Users

Your business is growing, and you need to onboard customers and partners quickly. Salesforce Identity can help enable and scale your customer and partner acquisition processes with self-registration, social sign-on, and CRM integration.

Self-Registration

External users can create user accounts quickly and easily with fully branded and customizable registration processes.

Social Sign-On

Customers and prospects can bring their own identity from social networks and public providers, such as Facebook, Google, Amazon, and PayPal.

CRM and Back-Office Integration

You can easily integrate your customers with your Salesforce org. When you run registration on your customer platform, identity data is no longer stuck in an IT system. Enrich your CRM data, create leads, link to your back-office customer records, and drive approval processes by implementing an external identity solution.

Deliver a Consistent Experience with a Single Identity Across All Channels

Salesforce Identity lets you engage with your users everywhere. Get a single, 360-degree view of your users while delivering a consistent, streamlined end-user experience for your brand.

Single Sign-On (SSO)

Save your users' time by letting them log in once to seamlessly access your apps. Uses secure industry standards like SAML, OpenID, and OAuth.

Mobile Identity

Deliver mobile apps to your customers with automatic SSO, authorization, and mobile-specific policies. Salesforce gives you a robust, open-source mobile SDK to easily create your mobile apps.

Cloud Directory Services

Adapt your business with customizable fields, automatable workflows, batch processing, and delegated administration through cloud Cloud directory services.

Secure and Manage Your Customer and Partner Ecosystems

By centralizing management of your users, Salesforce Identity makes life easy for your admins. They have a single place to manage Identity users and create reports and dashboards on their access.

Authorization and Policy Management

Deliver the right experience to your users at the right time and for the right reasons. Built-in access management, authorization, and robust policies make it easy for you to effective identity management.

Multifactor Authentication

Add an extra layer of security when logging in or accessing critical resources using secure, mobile two-factor authentication.

Provisioning and Unprovisioning Apps

Provide access and personalization to your apps with a customizable push-provisioning engine for just-in-time provisioning and single sign-on.

Reporting and Dashboards

Gain visibility into usage, adoption, and security with drag-and-drop customizable reports and dashboards.

Integrate and Customize to Your Business Needs

Salesforce Identity is integrated into the Salesforce Platform and is fully customizable, extensible, and scalable for any business.

Fully Branded

Extend your company's brand securely with drag-and-drop branding for login, self-registration, and federation services. Control and customize branding at run time depending on certain conditions, like who the user is or from where the user is logging in.

Workflows and Business Processes

Scale your administration and integration efforts with visually designed workflow processes.

Open APIs and Open Standards

Take advantage of the full suite of development tools that Salesforce Identity offers. It provides APIs for everything you need and supports major open identity standards, including SAML, OAuth 2.0, OpenID Connect, and SCIM.

Extend External Identity to Your Website

Salesforce Identity Embedded Login makes it easy to incorporate authentication into websites. Creating authenticated sessions between your community and website visitors extends your reach with your customers. For example, you can require that your users log in before they access your website. Or when customers change billing information on the website, Salesforce can update their contact information. As you collect information about a user, you can tailor the experience accordingly.

Your web developers aren't required to know anything about authentication services to add login capabilities to their web pages. They can rely on Embedded Login to take care of the process of authenticating users. Web developers just add a few HTML meta tags to a web page and a JavaScript function to determine what happens when a user successfully logs in. When your website visitors access the page, they enter their credentials in a login form generated by Embedded Login.

What Is Salesforce Identity for External Users?

Salesforce Identity is an Identity and Access Management (IAM) service that connects users to your apps, services, and devices. It provides a centralized point of management for your admins and provides a single, trusted identity for your end users. Traditionally, IAM services have focused on employee-facing use cases. Today, companies are using identity as a way to better connect with their customers and partners. We call this external identity.

When used for external identities, Salesforce Identity transforms CRM contacts into real digital identities that can self-register, log in, update their profile, and securely access web and mobile apps with a single identity. Plus, it's customized to your specific business process and brand using the power of the Salesforce Platform.

By delivering identity services directly from the same platform you use for sales, service, and marketing, you can recognize users across all your digital channels and create a consistent experience for customers and partners across all lines of business. The information and insight gathered converge with your existing CRM data and processes, thus building a single view of all your relationships.

Using Salesforce Identity, you build deeper, richer relationships with customers and partners by creating and maintaining a single identity for interaction across all channels.

External Identity Licenses

External Identity is a type of Salesforce license that enables you to deliver identity services, such as single sign-on (SSO) and social sign-on. External Identity is a standalone license and purchased in blocks of active users. These users are typically consumers of your business, such as purchasers, patients, partners, and dealers.

With an External Identity license, you can access several standard objects and 10 custom objects to deliver powerful self-service applications. The license includes extra data storage and API requests. Make sure that your org has sufficient resources before rolling out your external identity system. For more information, contact your Salesforce representative.

External Identity works with Community licenses. It's also included for free with all paid community user licenses in Enterprise, Performance, and Unlimited Editions. Each Developer Edition org includes 10 External Identity user licenses. You can upgrade the External Identity license to a Community license to benefit from Community features, including Cases, Contracts, Notes, Orders, and Tasks.

These licenses are also available for managing user identities.

Identity Only

Enables use cases similar to External Identity for your internal employees.

Identity Connect

An on-premises component that synchronizes users with Microsoft Active Directory (AD). While not commonly used in external scenarios, occasionally companies store their external users in AD.

Customer Community Plus or Partner Community

For customers who want to implement delegated administration. Admins with either a Customer Community Plus or Partner Community license can manage their users with external identity licenses.



Note: External Identity Allocations

We recommend that the number of External Identity license users in your external identity community not exceed ten million unique users per month. If you require additional user licenses beyond this limit, contact your Salesforce account executive. Exceeding this limit can result in an extra charge and decrease expected functionality.

What Features Are Available with the External Identity License


The Salesforce External Identity license lets you deliver identity services, such as single sign-on (SSO), to your customers and partners. You can purchase this standalone license for blocks of users who are typically consumers of your business, such as purchasers, patients, partners, and dealers.

What Features Are Available with the External Identity License

The Salesforce External Identity license lets you deliver identity services, such as single sign-on (SSO), to your customers and partners. You can purchase this standalone license for blocks of users who are typically consumers of your business, such as purchasers, patients, partners, and dealers.

With the External Identity license, you can store and manage customers and partners and authenticate them through username and password, SSO, or authentication providers. You can set up authentication providers to allow users to log in with their credentials from their social accounts, such as Facebook, Twitter, and LinkedIn. External Identity supports self-registration to easily provision new users.

You can upgrade this External Identity license to a Customer Community or Partner Community license to benefit from Community features.

Feature	External Identity
Accounts	 Read and Edit

EDITIONS

Available in: Salesforce Classic

External Identity licenses are available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions







USER PERMISSIONS

To assign and manage external identity users:

- Manage Users

To enable Communities:

- Customize Application

Feature	External Identity
Assets	 Read, Create, Edit
Chatter	
Contacts	 Read, Create, Edit
Identity	
Cases	
Products	
Orders	
Files	
Chatter Answers	
Ideas	
Knowledge	
Tasks	
Custom Objects	 Ten custom objects per profile, but custom objects in managed packages don't count toward this limit
Notes and Attachments	
Additional Storage	150 MB (25,000 active users) 2 GB (250,000 active users) 10 GB (1,000,000 active users) 60 GB (5,000,000 active users)

We recommend that the number of External Identity license users in your external identity community not exceed ten million unique users per month. If you require additional user licenses beyond this limit, contact your Salesforce account executive. Exceeding this limit can result in an extra charge and decrease expected functionality.

External Identity and Communities

Communities are branded spaces for employees, customers, and partners to connect. You can customize and create communities to deliver specific business apps and services, including identity services.

Salesforce External Identity uses communities for its deployment. Deploying external identity as a community gives you the ability to configure identity for your external users so they can easily recognize your brand. They can also have the same experience regardless of

which app they're accessing or whether they're using their desktop or mobile device. You can act as both a service provider and identity provider for all your apps without your customers realizing that the service runs on Salesforce. Similar to My Domain, a community can have a unique subdomain name, for example, `https://mycompanyname.force.com`. Or, the community can be configured with your custom SSL domain, such as `https://community.mycompanyname.com`.

Don't confuse community user licenses with underlying community capabilities. Salesforce provides community licenses for use cases like customer self-service, but there is no correlation between a community and a community license.

For more information, see [Salesforce Communities Overview](#) in the *Salesforce Help*.

Learn More with Trailhead

For a fun way to get a better understanding of external identity, take advantage of Trailhead. You can find identity modules in the Secure Identity and Access Management trail.



Identity Basics

Get an overview of the features in Salesforce Identity, and see how external identity fits into its feature set. Familiarize yourself with key identity terms, like single sign-on, social sign-on, identity providers, and service providers. Get familiar with the identity protocols, Security Assertion Markup Language, OAuth 2.0, and OpenID Connect.



External Identity for Customers

Walk through the entire process of setting up external identity on a trial developer org.

EXTERNAL IDENTITY IMPLEMENTATION CHECKLIST

Implementing external identity involves several steps. Use this checklist as a reference as you implement external identity in your production org.

1. [Set Up My Domain.](#)

My Domain is required for many Salesforce features, so it's likely that you've already enabled it.

2. [Create an external identity profile to authorize access.](#)

Clone the External Identity user profile, and then customize it for your community.

3. [Create a role structure.](#)

Communities requires a role hierarchy to better manage your community. Assign the community manager role to an internal user with administrative permissions.

4. [Create an account.](#)

Assign external users to a Salesforce account. When external users register for your community, they're assigned to this account.

5. [Use the Aloha Template to Create Your Community.](#) Use the Aloha template to create your community.

Before you create your first community, enable Communities and create your community with the Aloha template. The Aloha template is designed for managing external users.

6. [Create an external identity profile.](#)

Set your community's profile to control external user access.

7. [Brand your community.](#)

On the Workspaces Administration Login & Registration page, customize the default login page to reflect your brand. Add your logo, your colors, background image, and more.

8. [Activate your community.](#)

9. [Create a registration page.](#)

Allow guests to join your community by self-registering.

After you set up your basic external identity implementation, you can take advantage of other features. Enhance your community to drive customer engagement and apply your partner relationships.

- [Create quicker and easier logins.](#)

Let users log in and register using their email address or phone number instead of a username.

- [Use dynamic URLs to customize branding at run time.](#)

With dynamic branding, you can control and customize branding at run time depending on the circumstances, such as who the user is or where the user is logging in from.

- [Add login capabilities to pages on your website.](#)

Add Embedded Login metadata tags to any web page to require users to log in to access it.

- [Manage communities for external identity](#) (beta).

External Identity Implementation Checklist

Manage external identity users to make it easy for them to stay in your community longer. You can also create lightweight external communities by creating users without contacts (contactless users). These features are available only to users with an External Identity license.

CREATE A TYPICAL EXTERNAL IDENTITY IMPLEMENTATION

Let's walk through the process of creating a typical external identity implementation. Salesforce Identity integrates with the customer and partner business processes that you run on Salesforce.

Want to learn by doing? You can create a typical external identity implementation in this Trailhead, [External Identity for Customers](#).

[Prepare Your Org](#)

Because Salesforce External Identity integrates with the customer and partner business processes that you run on Salesforce, you perform a few basic administrative tasks to set up a typical deployment.

[Create a Branded Login Page](#)

You want your customers and partners to experience your brand consistently, whether they're visiting your community for the first time or logging in as a member. The community comes with a default login page that gives users access. You can customize this login page to reflect your brand.

[Enable Self-Registration](#)

You can invite visitors to join your community by having them self-register. You can add self-registration directly from Community Workspaces.

[Enable Self-Registration for B2C Users \(Optional\)](#)

Previously, you enabled self-registration for users in a simple business-to-business (B2B) data model. Each contact was associated with a default account called Customers. You can modify this process to support multiple accounts or even support a business-to-consumer (B2C) data model.

Prepare Your Org

Because Salesforce External Identity integrates with the customer and partner business processes that you run on Salesforce, you perform a few basic administrative tasks to set up a typical deployment.

[Create a Developer Org](#)

A developer org has all the features and licenses you need to get started with Salesforce Identity for Customers and Partners. You can use an existing org, trial org, or sandbox for external identity, but a developer org is a great way to get familiar with the product.

[Set Up My Domain](#)

Add a subdomain to your Salesforce org with My Domain. With a subdomain, you can highlight your company and gain more control over the login and authentication processes. Check if your org already has a subdomain by looking at its URL. If the URL contains your subdomain name, like `https://somethingcool.my.salesforce.com`, you're all set. If the URL contains an instance name, like `https://na30.salesforce.com`, set up My Domain.

[Control Authorization with Custom Profiles and Roles](#)

One important facet of identity and access management is the ability to control who has access to what. To get started, you set up two basic ways to control authorization: profiles and roles.

[Create an Account for CRM Integration](#)

One of the great things about Salesforce Identity for external users is that it's already integrated with your customer success platform. By driving identity on the same platform that you use for managing your customers and partners, you simplify your integration requirements while providing your users a better experience.

Create a Developer Org

A developer org has all the features and licenses you need to get started with Salesforce Identity for Customers and Partners. You can use an existing org, trial org, or sandbox for external identity, but a developer org is a great way to get familiar with the product.

1. Go to <https://developer.salesforce.com/signup>.
2. Enter your contact information.
3. Enter a unique username.
4. Submit the form, and wait for your welcome email.
5. In the welcome email, click the link to set your password.

That's it—you now have your own developer org.

Set Up My Domain

Add a subdomain to your Salesforce org with My Domain. With a subdomain, you can highlight your company and gain more control over the login and authentication processes. Check if your org already has a subdomain by looking at its URL. If the URL contains your subdomain name, like `https://somethingcool.my.salesforce.com`, you're all set. If the URL contains an instance name, like `https://na30.salesforce.com`, set up My Domain.

1. In your developer org, from Setup, enter *My Domain* in the Quick Find box, then select **My Domain**.
2. Enter the subdomain name you want to use within the Salesforce URL. For example, a company called Universal Containers uses the subdomain `universalcontainers`. The company's login URL would be `https://universalcontainers.my.salesforce.com`.
3. Click **Check Availability**. If the name is already taken, choose a different one.
4. Click **Register Domain**. Salesforce updates its domain registries with your new subdomain. When complete, you receive an email message with a subject like, "Your Developer Edition domain ready for testing." It takes just a few minutes.
5. After you receive the email, click the link to go to your subdomain. You're automatically signed in to the domain.
6. Return to Setup, and on the My Domain page, click **Deploy to Users**.

This step is often missed. So make sure that you deploy to users before you continue.

Control Authorization with Custom Profiles and Roles

One important facet of identity and access management is the ability to control who has access to what. To get started, you set up two basic ways to control authorization: profiles and roles.

A profile defines how users access objects and data. Clone and customize the external identity profile to meet your org requirements. A role allows you to control which users can view and edit data. Users can view and edit all data owned by or shared with users below them.

1. In your developer org, from Setup, enter *Profiles* in the Quick Find box, then select **Profiles**.
 - a. Next to External Identity User, click **Clone**.
 - b. Enter a name for the profile. Let's call it *Customers*.
 - c. Click **Save**.
2. Customize your profile.
 - a. Click **Edit**.

- b. Search for API Enabled, and then select the checkbox next to this permission.
API Enabled gives users API access to your org's data through any back-end mechanism.

- c. Click **Save**.

3. Create a role structure.

Beyond the permissions offered by profiles and permission sets, Salesforce lets you specify sharing settings to determine the access that users have to your Salesforce org's data. A user role hierarchy is one option for controlling sharing. You can combine it with sharing settings to segment data visibility.

- a. From Setup, enter *Role* in the Quick Find box, then select **Roles**.
- b. From the dropdown list, select **Product-based Sample**, then select **Set Up Roles**.
- c. Under CEO, click **Add Role**.
- d. For the role label, enter *Customer Manager*.
- e. Click **Save**.

4. Add the external identity role to your user.

- a. From Setup, enter *Users* in the Quick Find box, then select **Users**.
- b. Find your username, and click **Edit**.
- c. From the Role picklist, select **Customer Manager**.
- d. Click **Save**.

You've now learned the basics of authorizing users. For more information about configuring authorization, check out the [Data Security](#) Trailhead module. For more information about creating users and securing access, see the [User Management](#) Trailhead module.

Create an Account for CRM Integration

One of the great things about Salesforce Identity for external users is that it's already integrated with your customer success platform. By driving identity on the same platform that you use for managing your customers and partners, you simplify your integration requirements while providing your users a better experience.

External users are tied into the CRM data model within Salesforce. So when your external users register or update their profile, you get a consistent view of the external user within your Sales and Service processes.

- 1. In your developer org, switch to Sales.
- 2. Click the **Accounts** tab.
- 3. Create an account called *Customers*.
- 4. Click **Save**.

You've now completed all the prerequisites for creating an external identity community. To learn more, check out the [Accounts and Contacts](#) Trailhead module.

Create a Branded Login Page

You want your customers and partners to experience your brand consistently, whether they're visiting your community for the first time or logging in as a member. The community comes with a default login page that gives users access. You can customize this login page to reflect your brand.

To learn how login page branding works, watch the [How to Set Up a Community for Identity and Deploy a Branded Login Page](#) video. Then follow these steps to create your own external identity community.

[Use the Aloha Template to Create Your Community](#)

After you enable Salesforce Communities for your org, you can create your external identity community using the Aloha template that Salesforce provides. Behind the scenes, Salesforce creates default pages to handle the entire login experience, which includes self-registration, forgotten and reset passwords, and identity verification.

[Control External User Access to Your Community](#)

Set your community's profile to control external user access. The profile enables single sign-on access to your community.

[Brand Your Community Using the Login & Registration Page](#)

With little effort, you can brand your community starting with the first pages a user sees when they click your community's URL. Salesforce Identity gives you a point-and-click way to customize the login page that prompts users to log in to your community. These branding options apply to related login pages, including pages for users to verify their identity or reset passwords and for new users to register. If you've set up login flows, they use these branding settings.

[Customize Your Login Pages with Community Builder](#)

Using Community Builder is one way to customize your login pages easily. Community Builder pages are available after you publish your community.

[Activate Your Community](#)

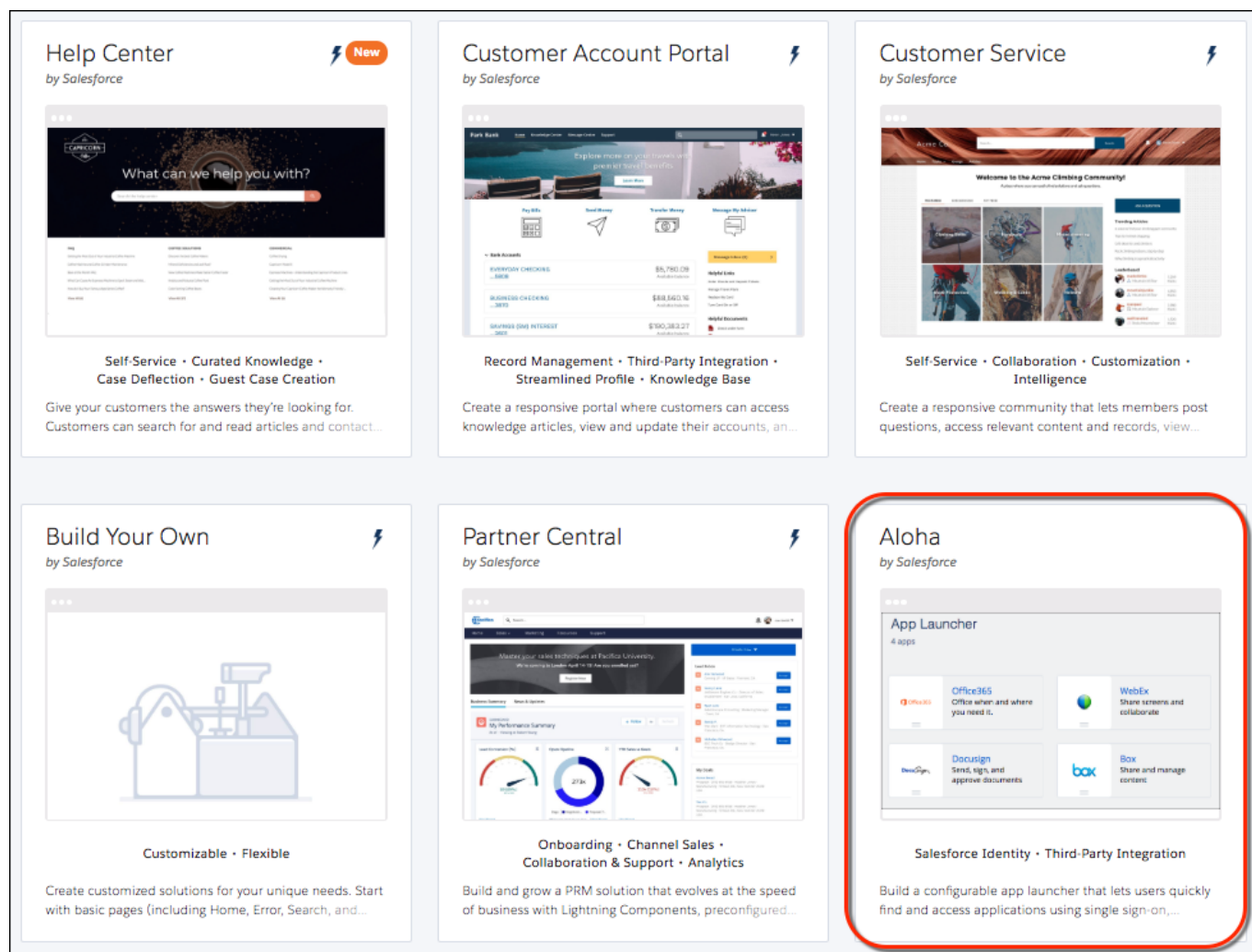
To complete your external identity community setup, you must activate it.

Use the Aloha Template to Create Your Community

After you enable Salesforce Communities for your org, you can create your external identity community using the Aloha template that Salesforce provides. Behind the scenes, Salesforce creates default pages to handle the entire login experience, which includes self-registration, forgotten and reset passwords, and identity verification.

When you enable Communities for your org, you provide a community domain. The community domain collects all your communities under one URL. Typically, your community domain is your company domain.

1. From Setup, enter *Communities* in the Quick Find box, then select **Communities Settings**.
2. Select **Enable communities**.
3. Enter a memorable domain name.
Keep in mind that customers and partners interact with this domain name. After you choose this name, you can't change it. Later on, you can add a [custom SSL domain](#) to have more control over your community branding.
4. Select **Check Availability**.
5. Click **Save**, and then click **OK**.
The Communities feature is now enabled for your org. Next, create your external identity community.
6. Click **New Community**.
You're shown a series of community templates.



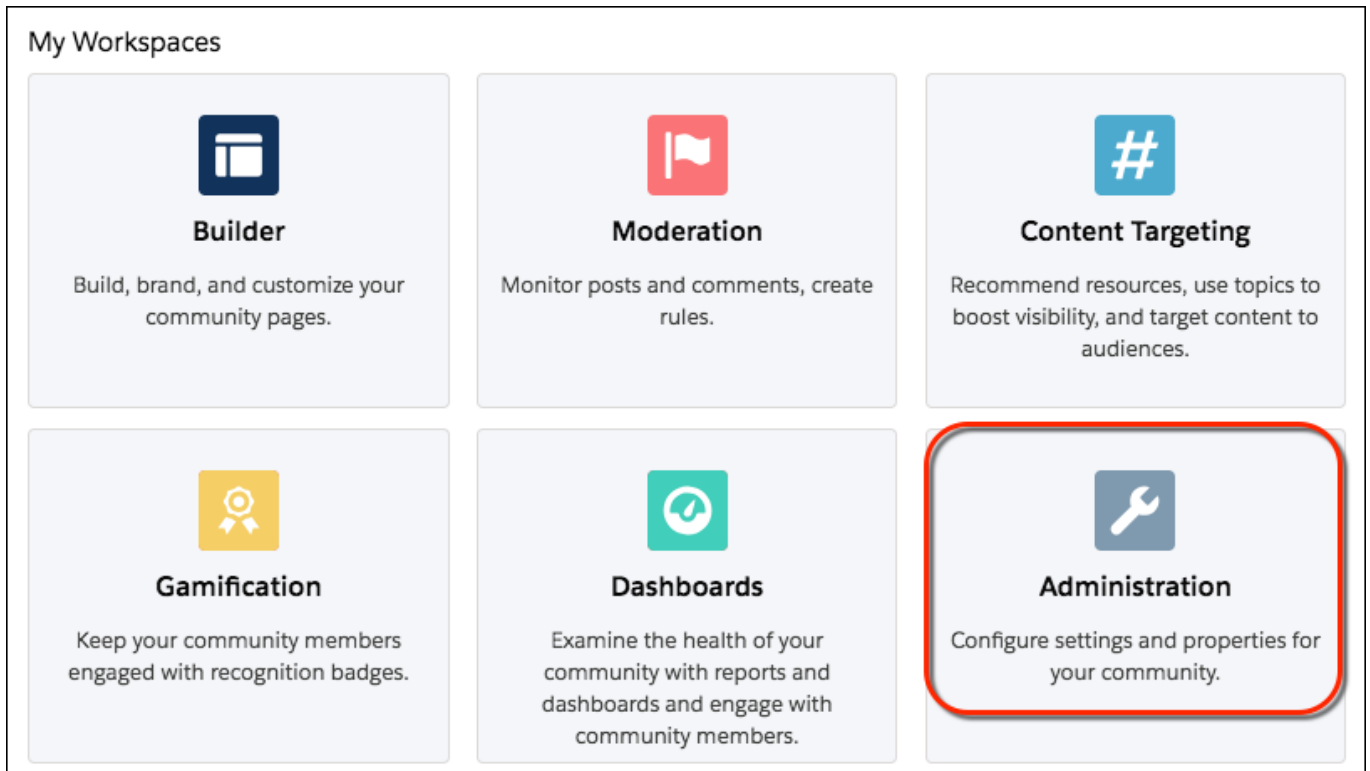
7. Choose the Aloha template, and click **Get Started**.

The Aloha template is designed with external identity in mind. For more information, see the [Getting Started with the Aloha Community Template for Salesforce Identity Guide](#).

8. Give your community a name and URL. For this example, enter *Customers* in both fields.

9. Click **Create**.

After a bit of background work, your new community appears in Community Workspaces. Notice the Administration workspace. Administration is where you configure your community's login experience.



Control External User Access to Your Community

Set your community's profile to control external user access. The profile enables single sign-on access to your community.

After you add the external identity profile, all users assigned to that profile become members of the community. You can also assign new users to the profile when they sign up.

1. If you haven't already done so, clone and customize the external identity profile as described in [Control Authorization with Custom Profiles and Roles](#).
2. Set up community membership from the Members page of the Administration workspace.
 - a. From Setup, enter *All Communities* in the Quick Find box, select **All Communities**, and click **Workspaces** next to your community.
 - b. Select **Members**.
 - c. From the search list, select **All**.
 - d. From the list of available profiles, locate the Customers profile that you created earlier, and click **Add** to add it to the selected profiles.
 - e. Click **Save**.

Salesforce updates the membership for your community. When done, Salesforce sends an email to your community members.

 **Note:** If you prefer, you can control user access using permission sets.

Brand Your Community Using the Login & Registration Page

With little effort, you can brand your community starting with the first pages a user sees when they click your community's URL. Salesforce Identity gives you a point-and-click way to customize the login page that prompts users to log in to your community. These branding options apply to related login pages, including pages for users to verify their identity or reset passwords and for new users to register. If you've set up login flows, they use these branding settings.

For convenience, you can use the sample logos and backgrounds we provide by downloading this file:

<https://www.salesforceidentity.info/ExternalIdentityAssets.zip>. Or use your own.

The branding options you set here apply to all types of login pages, except Community Builder pages.

1. From Setup, enter *All Communities* in the Quick Find box, then click **Workspaces** next to your community.

2. Select **Administration**, and select **Login & Registration**.

3. Next to Choose Logo Type, select **File**.

4. Next to Logo File, upload a logo. If you don't have your own logo, upload `fix-logo.png` from the sample files that you downloaded.

Instead of uploading a file, you can enter a URL where you store your branding assets. This URL can use a dynamic value to determine which logo appears at run time. The URL contains the `expid` placeholder `{expid}`, for example,

`https://www.my-cms.com/{expid}/logo.png`. When customers click a link to log in, how the login page looks depends on the `expid` value indicated in the login URL.

5. Next to Background Type, select **Color**, and select your color.

Optionally, you can enter a Background Image URL to an image or a color gradient. This URL can also use a dynamic value to determine which background image appears at run time.

6. Choose the color of the login buttons. These buttons appear on all login pages, including Log In, Verify, and Sign Up.

7. Optionally, enter a URL for Right Frame URL to display content on the right side of the login page. Leave it blank for now.

The content of the right-frame URL displays in an iframe to the right of the login form. The right-frame URL is useful, for example, to display a branded image or topical announcements.

8. For Footer Text, enter any text, such as a copyright, to display at the bottom of the login form.

9. Click **Save**.

10. Notice that Default Page appears for the login page (1) and password pages (2). These default pages come with the Aloha template.

Logo Type [i](#)

Logo File No file chosen
JPG, GIF or PNG, 100 KB max.

 125 px max
250 px max

Background Type [i](#)

Background [i](#)

Login Button [i](#)

Right Frame URL [i](#)

Footer Text [i](#)

Login Page Setup

Choose a login page type to create a branded login experience. Depending on the login page email, phone number, or other user identifier. [Learn more](#)

Login Page Type [i](#) **1**

☒ Allow internal users to log in directly to the community [i](#)

Select login options to display on the login page. To add more login options, visit [Single](#)

☒ MyNGO username and password

Logout Page URL

Full URL [i](#)

Password Pages

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

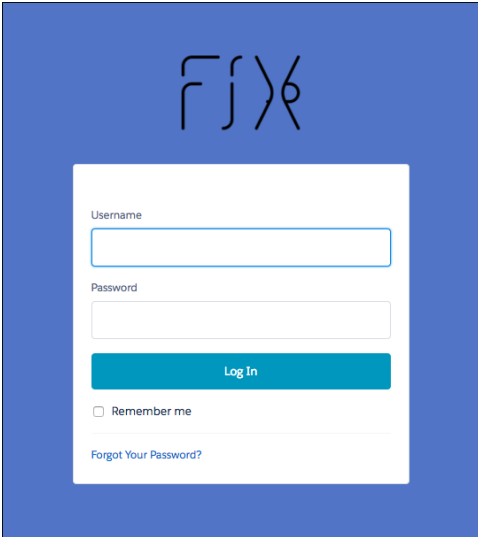
Forgot Password [i](#) **2**

Reset Password [i](#)

11. Let's see how your changes appear on a login page.



- From Administration, select **Settings**, and copy the URL for your community.
- Open a new private (incognito) window, and paste the URL in the address bar.

Here's how the login page looks with the branding options set on the L&R page.

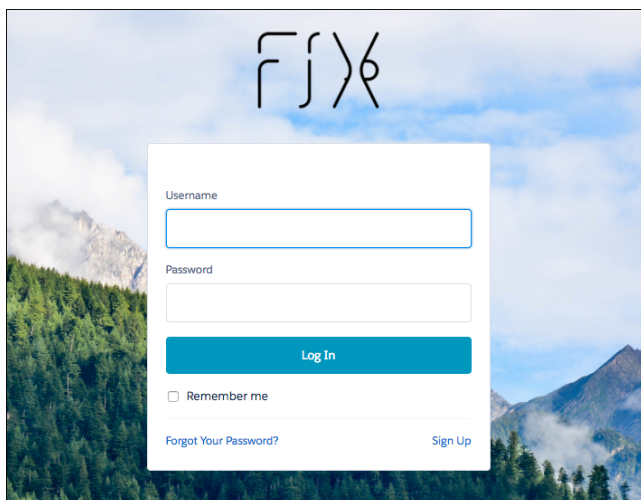


When external users—your customers and maybe your partners—navigate to your community URL, they're greeted with this login page.

To use an image instead of a color for the login form, enter a background image URL on the L & R page.

Background Type	Image URL 
Background Image URL	<input type="text" value="http://identitycms.herokuapp.com/back_url.jpg"/> 

Here's a login page built with a background image URL.



SEE ALSO:


[Set Up Dynamic URLs](#)

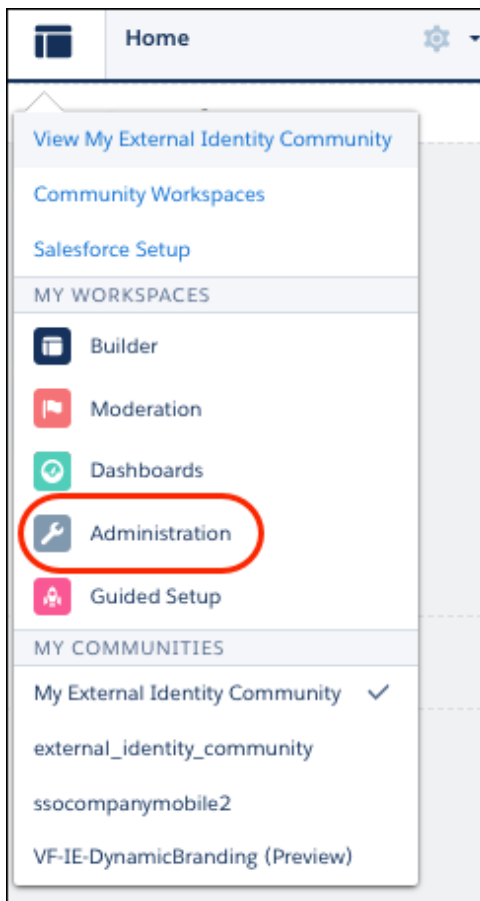
[Customize Your Login Pages with Community Builder](#)


Customize Your Login Pages with Community Builder

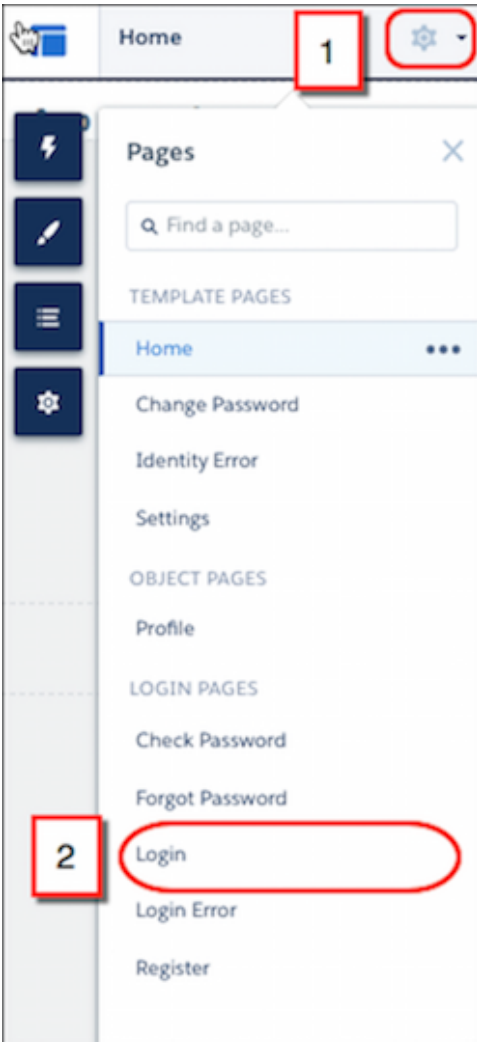
Using Community Builder is one way to customize your login pages easily. Community Builder pages are available after you publish your community.

These steps take you through branding your community using Community Builder. When you brand your community from Community Builder, the branding settings on the Login & Registration Page of the Administration workspace are ignored.


1. From Setup, enter *All Communities* in the Quick Find box, select **All Communities**, and then click **Builder** next to your community.
2. Get the Community Builder pages that come with the Aloha template. They're available after you publish the community.
 - a. In the top right, click **Publish**.
If you see a notification to activate community, ignore it. You activate the community later on.
 - b. Return to your Login & Registration page. In the top left, click , and select **Administration**.



- c. Select **Login & Registration**.
 - d. Under Login Page, select **Community Builder Page**, and then select **login** from the page picker.
 - e. Click **Save**.
 - f. Return to Community Builder. At the top of Workspaces, select **Administration**, then select .
3. At the top of Builder, click the dropdown next to Setup (1), then select **Login** (2).



Your login page now appears in Community Builder. You can further brand it with Community Builder.

4. Select .
5. Under Background Image, select **Clear** and upload a new background image. If you don't have a background image, upload `cupontablesmall.png` from the sample files you downloaded.
6. Click **Publish**.

In just a few steps, you made your external identity community your own. At this point, you can make further customizations. For example, you can control branding based on run-time circumstances. For details, see [Dynamic URLs: Determine Your Brand at Run Time](#).

SEE ALSO:

[Implement Embedded Login](#)

[Dynamic URLs: Determine Your Brand at Run Time](#)

[Brand Your Community Using the Login & Registration Page](#)

Activate Your Community

To complete your external identity community setup, you must activate it.

1. From Setup, enter *All Communities* in the Quick Find box, select **All Communities**, and click **Workspaces** next to your community.
2. Select **Administration**, and select **Settings**.
3. Click **Activate Community**, and click **OK**.
Salesforce sends you an email when the community is activated.
4. To verify that your community is activated, refresh your private (incognito) window .
The browser displays the login page for your community.

Enable Self-Registration

You can invite visitors to join your community by having them self-register. You can add self-registration directly from Community Workspaces.

To learn how, you can watch the [Enabling Self-Registration in a Community](#) video. Then follow the steps.

[Add a Self-Registration Page](#)

You can create a self-registration page for visitors to use to join your community.

[Add Fields to Collect Additional Information](#)

When users register, you often want to ask them for more than just basic information. You can easily add fields to the registration page.

[Add a Password Field to Enable Login Directly During Registration](#)

You can add a password field to your self-registration page to require users to supply a password when they register.


Add a Self-Registration Page

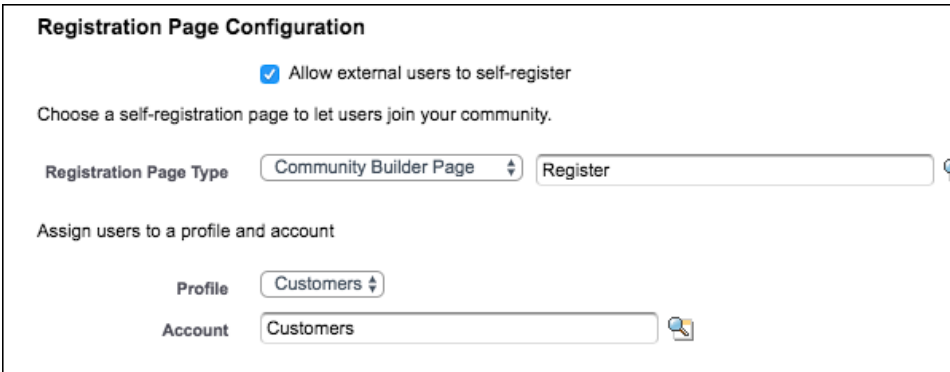
You can create a self-registration page for visitors to use to join your community.



Note: These steps create a self-registration page using the Community Builder registration page type. This self-registration form requires visitors to supply a username and password to join your community. Optionally, you can create a self-registration page that requires only an email address or phone number. For details, see [Use the Configurable Self-Reg Page for Easy Sign-Up](#).

1. From Community Workspaces, select **Administration**, and then select **Login & Registration**.
2. Select **Allow external users to self-register**.

3. For Registration Page Type, select **Community Builder Page**, click , and then select **Register**.
4. For Profile, select **Customers**. This setting gives new users your External Identity user profile.
5. For Account, select **Customers**. Recall that you already created this account as part of preparing your org. Here's what the page looks like when you're done.



Registration Page Configuration

☒ Allow external users to self-register


Choose a self-registration page to let users join your community.

Registration Page Type: Community Builder Page Register

Assign users to a profile and account

Profile: Customers

Account: Customers

6. While we're here, let's use the Community Builder page for Forgot Password.
 - a. Under Password Pages, for Forgot Password, select **Community Builder Page**, click , and then select **Forgot Password**.
7. Click **Save**.
8. Return to the private (incognito) browser, and reload the login page for your community. To register a new user for your community, click **Sign Up**.
9. Check whether self-registration succeeded.
 - a. From Setup, enter *users* in the Quick Find box, and select **Users**. Notice that your new user is listed.
 - b. From the App Launcher, under All Items, select **Accounts**, then select the **Customers** account. Your new user is listed as a contact.

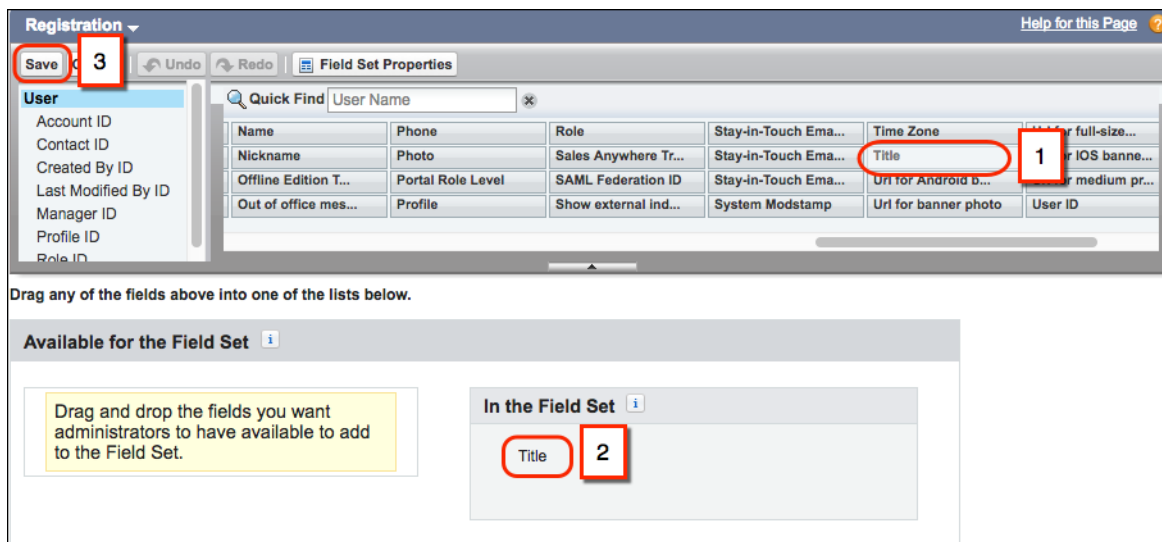
SEE ALSO:

[Embedded Login Considerations](#)

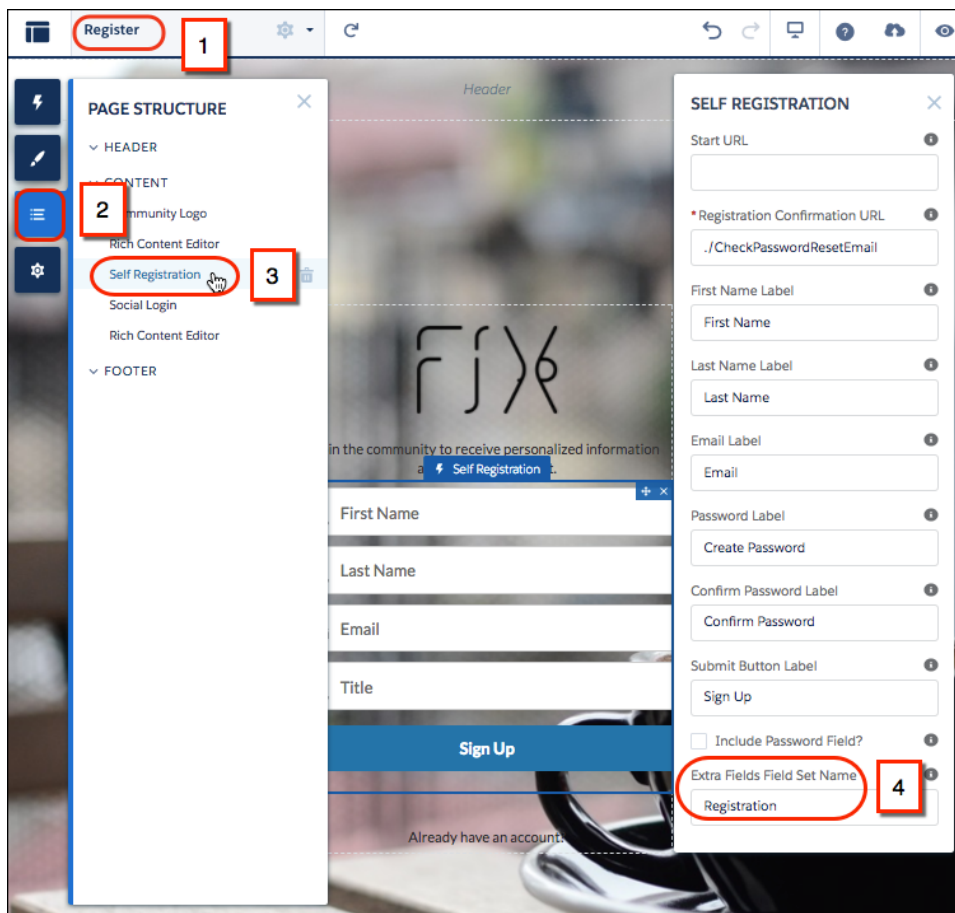
Add Fields to Collect Additional Information

When users register, you often want to ask them for more than just basic information. You can easily add fields to the registration page. Tailoring your registration page involves navigating to a few different areas in the app. First watch the [Enabling Self-Registration in a Community](#) video. Then follow these steps.

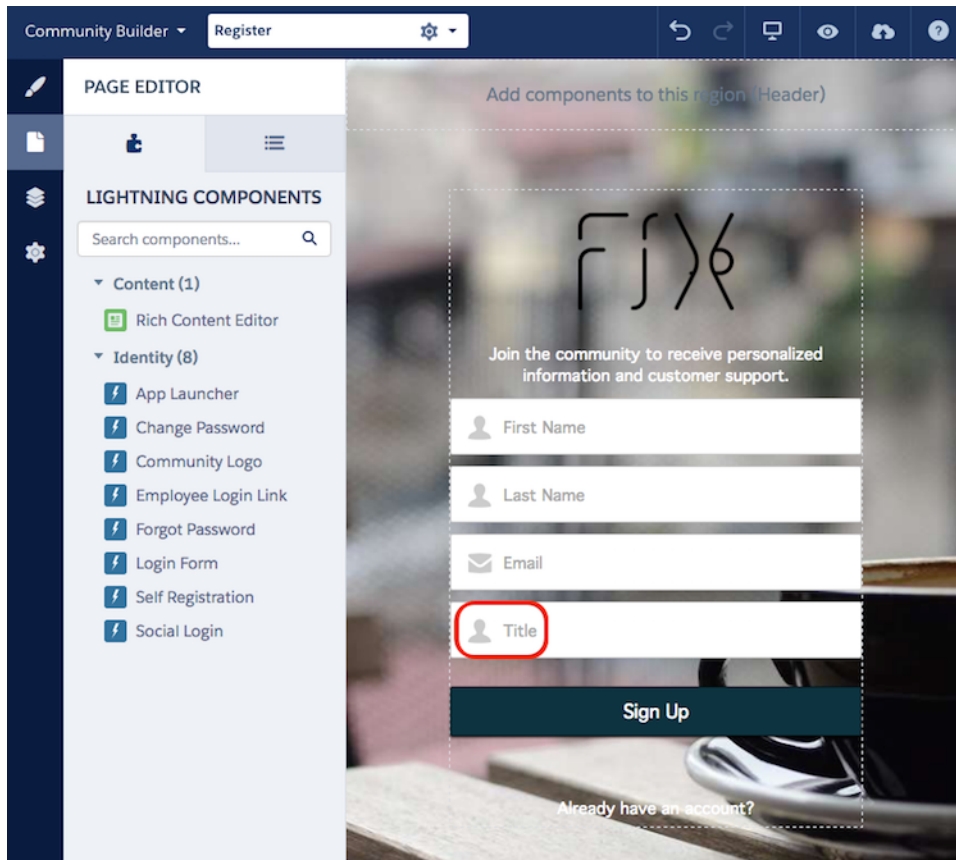
1. In Setup, enter *objects* in the Quick Find box, and select **Object Manager**.
2. Select **User**, and then select **Field Sets**.
3. Click **New**. Name the field *Registration*.
4. For where is this used, enter *registration field set*.
5. Click **Save**.
6. Drag **Title (1)** into the field set (2), and click **Save (3)**.



7. Go back to Community Builder. From Setup, enter *All Communities* in the Quick Find box, then click **Builder** next to your community.
8. From Setup, select **Register** (1), click  (2), and then select **Self Registration** (3).



9. On the right, scroll to Extra Fields Set Name (4), and enter *Registration*. The page reloads and displays your title field.



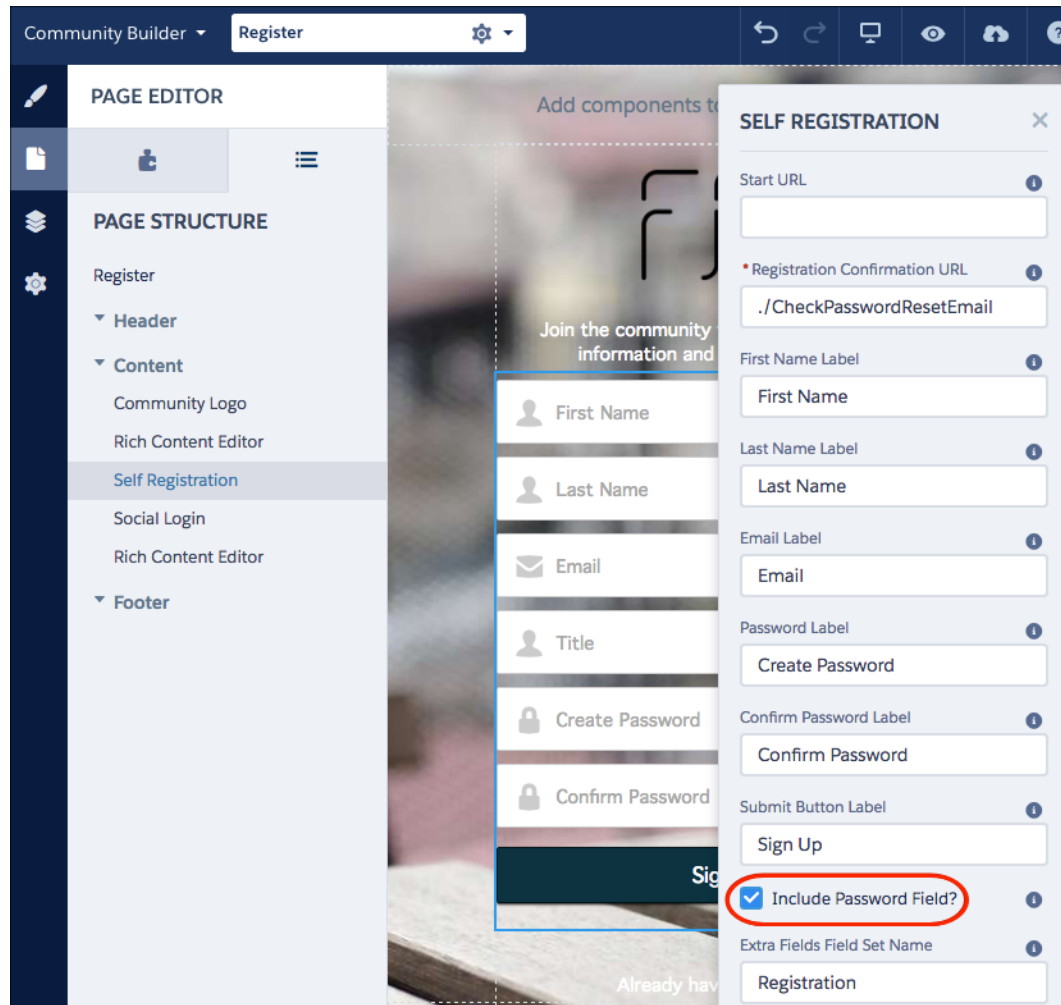
10. Click **Publish**.

Add a Password Field to Enable Login Directly During Registration

You can add a password field to your self-registration page to require users to supply a password when they register.

It's not much effort to require a password because you're already in Community Builder.

1. In Community Builder, select **Include Password Field**. The self-registration page reloads and displays the password field.



2. Click **Publish**.
3. After you receive the confirmation email, go back to your browser and confirm that your self-registration page includes the password field.

Enable Self-Registration for B2C Users (Optional)

Previously, you enabled self-registration for users in a simple business-to-business (B2B) data model. Each contact was associated with a default account called Customers. You can modify this process to support multiple accounts or even support a business-to-consumer (B2C) data model.

Salesforce supports a B2C model through person accounts. The best way to get started with person accounts is to review the [Setting Up Person Accounts Implementation Guide](#).

You can also watch the [Setting up Person Accounts and Enabling Them for Self-Registration in a Community](#) video. It walks you through setting up and enabling person accounts for self-registration.

[Enable Person Accounts](#)

Person accounts store information about individual people by combining certain account and contact fields into one record.

[Configure Self-Registration for Person Accounts](#)


You can use person accounts instead of business accounts for self-registration.

SEE ALSO:

[Salesforce Help: What Is a Person Account?](#)

Enable Person Accounts

Person accounts store information about individual people by combining certain account and contact fields into one record.

 **Important:** After Person Accounts is enabled, it can't be disabled. We recommend that you create a sandbox to preview how Person Accounts affect your Salesforce org.

1. Make sure you meet the following prerequisites.
 - The account object has at least one record type.
 - User profiles that have read permission on accounts have read permission on contacts.
 - The organization-wide default sharing is set so that either Contact is **Controlled by Parent** or both Account and Contact are **Private**.
2. From Setup, enter *Account Settings* in the Quick Find box, and then select **Allow Customer Support to enable Person Accounts**.
We'll verify your org meets the prerequisites, then send you an email with additional information. If you don't see a message verifying you meet the prerequisites, go back to step 1.
3. Contact Salesforce Customer Support by logging a case to enable Person Accounts.
Refer to the email we sent about what to include in the case.
4. After Person Accounts is enabled, a person account record type is created. You can create additional record types for person account if needed.
5. [Assign the person account record type to user profiles.](#)

EDITIONS

Business accounts available in: both Salesforce Classic and Lightning Experience

Business accounts available in: **All** Editions

Person accounts available in: Both Salesforce Classic and Lightning Experience

Person accounts available in: **Professional, Enterprise, Performance, Unlimited, and Developer** Editions

USER PERMISSIONS

To enable person accounts:

- Customize Application

Configure Self-Registration for Person Accounts

You can use person accounts instead of business accounts for self-registration.

1. Assign record types to your community's security profile by updating your community's public-access settings. This step ensures that the security profile that controls anonymous access in your community has access to account record types.
 - a. From Setup, enter *All Communities* in the Quick Find box, then select **All Communities** and click **Manage** next to the Customers community.
 - b. Select **Administration**, select **Pages**, then select **Go to Lightning Platform**.
 - c. Click **Public Access Settings**.
 - d. Under Record Type Settings, click **Edit** next to Accounts.
 - e. Select business and person record types and add them to Selected Record Types.
 - f. Click **Save**.
2. Update the self-registration setting on your login configuration page to use person accounts.

- a. From Setup, enter *All Communities* in the Quick Find box, then select **All Communities** and click **Workspaces** next to the Customers community.
- b. Select **Administration**, and then select **Login & Registration**.
- c. Scroll down to Registration and make sure that the default Account field is empty. By removing the default, new users are created as person accounts.
- d. Click **Save**.

You're done. New users that register with your branded self-registration page are now B2C-style users using person accounts.

SET UP SSO FOR YOUR USERS

While self-registration is a great way to get started, often users exist in your back-office systems or with social providers, such as Facebook, LinkedIn, or Twitter. With Salesforce Identity, you can use these existing sources with single sign-on (SSO) and just-in-time (JIT) provisioning. SSO and JIT provisioning let you create and update user accounts on the fly.

The following methods are available for SSO into Salesforce and communities.

Social sign-on

Salesforce users can authenticate and log in from different social identity providers, such as a Twitter and Facebook. They can also log in through open federation standards like OpenID Connect.

Federated authentication

Use Security Assertion Markup Language (SAML) to send authentication and authorization data between affiliated but unrelated web services.

Delegated authentication

Integrate Salesforce with various legacy authentication technologies.

Both federated authentication and social sign-on let you accept identities from existing identity providers and create users or link to and update existing users. Social sign-on is a common and effective way to engage your customers without having them create accounts.

[Social Sign-On](#)

Salesforce Identity supports a variety of public authentication providers, such as LinkedIn, Google, Facebook, Twitter, and open-standard OpenID Connect through the auth. providers framework. Using these providers, you can accept identity and link to existing Salesforce users. You can also create and update users on the fly using identity information asserted by the provider.

[Create an Auth. Provider](#)

You choose which Auth. providers can access your Salesforce org from Setup. With a few clicks, you can add the option to log in with one or more social accounts. Here's how to set up Facebook as an Auth. provider.

[Customize Your Registration Handler](#)

The registration handler is an Apex class that handles the heavy lifting of creating users, updating users, and linking to existing users, accounts, and contacts. You can also integrate more business processes, such as creating opportunities or calling out to back-office customer systems.

[Enable Your Auth. Provider in Your Community](#)

You created an Auth. provider for Facebook and customized it with a registration handler. Now instruct the login page in your community to display Facebook as an option on your external identity community's login page.

[Accept User Identity with SAML and Just-In-Time Provisioning](#)

With Salesforce Identity, you can bring your own identity from standards-based systems using SAML. You can integrate with existing SAML identity providers, letting users access your community based on your own authentication systems. And you can use SAML with just-in-time (JIT) provisioning to create or update users on the fly as part of the SSO process.

Social Sign-On

Salesforce Identity supports a variety of public authentication providers, such as LinkedIn, Google, Facebook, Twitter, and open-standard OpenID Connect through the auth. providers framework. Using these providers, you can accept identity and link to existing Salesforce users. You can also create and update users on the fly using identity information asserted by the provider.

The video [Setting Up Social Sign-On](#) walks you through setting up social sign-on. To get started, the following steps help you set up social sign-on with Facebook. The process is similar for all providers, so if you don't use Facebook, you can easily substitute another provider.

Create an Auth. Provider

You choose which Auth. providers can access your Salesforce org from Setup. With a few clicks, you can add the option to log in with one or more social accounts. Here's how to set up Facebook as an Auth. provider.

1. In your developer org, from Setup, enter *Auth. Providers* in the Quick Find box, then select **Auth. Providers**.
2. Click **New** and select **Facebook** for the provider type.
3. Name the Auth. provider *Facebook* and enter the URL suffix.
4. For this exercise, leave the Consumer Key, Consumer Secret, User Info Endpoint URL, and Default Scopes fields empty. When you leave these fields empty, Salesforce Identity uses a default app when interacting with Facebook. You can't customize the brand that users see nor the scope of access you request from the provider. In a real deployment, you register an app with the provider and configure your own consumer key (*client_id*) and consumer secret (*client_secret*).
5. For Registration Handler, select **Automatically create a registration handler template**.
6. For Execute Registration As, choose your admin to run the registration handler. Heads up: This step is essential and often gets overlooked.
7. For Icon URL, select **Choose one of our sample icons**.
8. In the new window, find a Facebook icon that you want to use, click it, and copy the URL.
9. Close the window and paste the URL as your Icon URL.
10. Click **Save**.

Customize Your Registration Handler

The registration handler is an Apex class that handles the heavy lifting of creating users, updating users, and linking to existing users, accounts, and contacts. You can also integrate more business processes, such as creating opportunities or calling out to back-office customer systems.

You can edit the generated registration handler. Or to get started, use one of our open-source samples.

1. In another browser window, open the registration handler, <https://github.com/salesforceidentity/IdentityTrail-Module3/blob/master/SimpleFacebookRegistrationHandler.cls>. This class creates an account and contact, and it also creates an opportunity during user creation.
2. Click **raw** and copy the code.
3. Return to your Auth. provider and click **AutoGeneratedRegHandler**.
4. Click **Edit**.
5. Select all the code and paste it over the old code.
6. Click **Save**.

You now have a fully functional Auth. provider that's ready for social sign-on with Facebook.

Enable Your Auth. Provider in Your Community

You created an Auth. provider for Facebook and customized it with a registration handler. Now instruct the login page in your community to display Facebook as an option on your external identity community's login page.

1. From Setup, enter *All Communities* in the Quick Find box, then select **All Communities**.
2. Click **Workspaces** next to your community.
3. Select **Administration**, then select **Login & Registration** and confirm that Facebook shows up in the checkbox under Login.
4. Click **Save**.
5. Test your changes by going to your community in a new private (incognito) browser or window.
6. Reload the login page.
7. Click the Facebook logo.
8. Log in with your Facebook account.

You are immediately granted access to the community. If you return to the browser where you are administering Salesforce, go to Accounts and drill into the Customer account. You find that you show up as a contact. When you view your contact, you see that you have an opportunity associated with the contact. The registration handler created the opportunity.

For more information on configuring social sign-on for various providers, see [Social Sign-On](#). You can find more sample Apex classes that implement the RegistrationHandler interface on the GitHub repository, <https://github.com/salesforceidentity>.

Accept User Identity with SAML and Just-In-Time Provisioning

With Salesforce Identity, you can bring your own identity from standards-based systems using SAML. You can integrate with existing SAML identity providers, letting users access your community based on your own authentication systems. And you can use SAML with just-in-time (JIT) provisioning to create or update users on the fly as part of the SSO process.

We assume that you're already familiar with SAML authentication protocols and you know how to work with your identity provider to configure SSO for your company. For more information on setting up SSO, watch the see [Setting up Single Sign-On \(Salesforce Classic\)](#) video.

To use JIT provisioning, you pass a Salesforce-defined set of attributes in your SAML assertion. Or use Apex provisioning handlers to have Salesforce Identity adapt to existing third-party schemas. For more information, see [Just-in-Time Provisioning for Communities](#) in *Salesforce Help*.

SET UP SSO FOR YOUR APPS

You can use Salesforce Identity to establish identity for your users with self-registration, social sign-on, and branded login services. You can also provide SSO and access to web and mobile apps with identity services like SAML, OpenID Connect, or OAuth engines.

[Set Up SSO and Access for Mobile Apps](#)

Use the Salesforce Mobile SDK to create mobile apps that integrate with your org's identity setup. You use the OAuth protocol to connect Salesforce Identity with mobile apps. OAuth is an open standard used for authorization that provides applications secure, delegated access to services on behalf of a user—without sharing the user's credentials. Fortunately, you don't need to know much about OAuth to use it. Salesforce Identity and Salesforce work together.

[Set Up SSO and Access for Your Web App](#)

Salesforce Identity works as an identity provider to provide SSO to other apps. You can use SAML to set up SSO with web apps.

Set Up SSO and Access for Mobile Apps

Use the Salesforce Mobile SDK to create mobile apps that integrate with your org's identity setup. You use the OAuth protocol to connect Salesforce Identity with mobile apps. OAuth is an open standard used for authorization that provides applications secure, delegated access to services on behalf of a user—without sharing the user's credentials. Fortunately, you don't need to know much about OAuth to use it. Salesforce Identity and Salesforce work together.

To walk through creating a mobile app for SSO, check out the video [How to Create a Sample Mobile App and Take Advantage of Salesforce Identity](#). Then follow these steps to create a sample mobile app. To get started, we create a connected app that supports OAuth. The process is similar to using the SAML protocol.

[Create a Connected App for Your Mobile App](#)

The connected app integrates your mobile app with Salesforce Identity.

[Install the Salesforce Mobile SDK](#)

Salesforce Mobile SDK is an open-source suite of familiar technologies (including a REST API and OAuth 2.0). You use the SDK to rapidly build HTML5, native, and hybrid mobile apps that connect to the Salesforce platform.

[Create a Mobile App](#)

Let's use the Salesforce Mobile SDK to jump-start our app.

[Configure the Mobile App to Point to Your Community](#)

Let's teach the mobile app about your community to finish the identity configuration.

[More About Single Sign-On for Your Mobile App](#)

You've learned the basics of acting as an identity provider for mobile apps. For more information, several Trailhead modules can guide you.

Create a Connected App for Your Mobile App

The connected app integrates your mobile app with Salesforce Identity.

This example assumes that you're using macOS and iOS. The steps for Android are similar.

1. From Setup, enter *Apps* in the Quick Find box, and then select **App Manager**.
2. Click **New Connected App**.
3. Enter a name for your app. Let's call it My Mobile App.
4. For Contact Email, enter your email address.
5. Select **Enable OAuth Settings**.
6. Enter a callback URL. Use `mymobileapp://callback`.
7. For Available OAuth Scopes, select **id**, **openid**, **api**, **refresh_token**, **web**, and **visualforce**.
8. Click **Save**.
9. Click **Continue**.

Install the Salesforce Mobile SDK

Salesforce Mobile SDK is an open-source suite of familiar technologies (including a REST API and OAuth 2.0). You use the SDK to rapidly build HTML5, native, and hybrid mobile apps that connect to the Salesforce platform.

If you don't already have the Salesforce Mobile SDK, follow the installation instructions in the [Salesforce Mobile SDK Development Guide](#) to download it.

Create a Mobile App

Let's use the Salesforce Mobile SDK to jump-start our app.

1. At a command line, change to a directory where you want to create your app assets.
2. Run `forceios create`.
3. For application type, enter *native*.
4. For application name, enter *MyMobileApp*.
5. Press Enter to create the app in the current directory.
6. For package name, enter *com.yourcompany*.
7. For organization name, enter *YourCompany*.
8. Return to the Connected App page in your Developer org and copy the consumer key.
9. Paste the key in the forceios utility as the value for Connected App ID.
10. Return to the Connected App page in your Developer org and copy the callback URL.
11. Paste the URL in the forceios utility as the value for the Callback URI.
12. Press Enter.

The Mobile SDK creates a mobile app project for you.

Configure the Mobile App to Point to Your Community

Let's teach the mobile app about your community to finish the identity configuration.

1. At a command line, change to the app's directory `cd MyMobileApp`.
2. Open your app in XCode: `open MyMobileApp.xcodeproj`.

3. In your Developer org, copy your community URL, omitting `https://`. If you don't recall the URL, from Setup, enter *Communities* and then select **All Communities**.
4. Return to XCode.
5. In the file browser, expand `MyMobileApp > Supporting Files`.
6. Click **MyMobileApp-info.plist**.
7. Select the `SFDCOAuthLoginHost` key value and replace `login.salesforce.com` with your community URL (again, without `https://`).

Now all you have to do is build your app. Click the triangle-shaped button to build your app and watch it connect to your community. You can now log in to your app, even using social sign-on if you want. With Salesforce Identity, you focus on building your app rather than spending resources integrating identity.

More About Single Sign-On for Your Mobile App

You've learned the basics of acting as an identity provider for mobile apps. For more information, several Trailhead modules can guide you.

- [Salesforce Identity How-To](#) video series
- [Mobile Basics](#) Trailhead module
- [Native iOS](#) Trailhead module
- [Native Android](#) Trailhead module
- [HTML5 & Hybrid](#) Trailhead module

Set Up SSO and Access for Your Web App

Salesforce Identity works as an identity provider to provide SSO to other apps. You can use SAML to set up SSO with web apps.

Here's an example of using SAML with a sample app. For an overview, watch the [How to Set Up Single Sign-On with a Sample Application Using SAML](#) video.

[Create a Connected App for Your Web App](#)

A connected app is an application that integrates with Salesforce Identity using APIs and identity services. Connected apps use standard identity protocols like SAML, OAuth, and OpenID Connect to authenticate, provide single sign-on, and provide tokens for use with Salesforce APIs.

[Create a Sample Service Provider on Heroku](#)

To implement single sign-on, you need an app that speaks SAML. We've prepared a free sample that gets you up and running quickly.

[Configure Salesforce Identity to Provide Identity for Your App](#)

Teach Salesforce Identity about the SAML configuration of your new app.

[Authorize Your Web App](#)

The Salesforce Identity SAML identity provider understands your app via the connected app, but your users aren't authorized to access it. You still have to configure authorization.

[Configure Your App to Trust Salesforce Identity](#)

Even though you've described your sample app to Salesforce Identity, your app doesn't yet trust Salesforce to act as an identity provider. You must configure the app to accept SAML messages. This process is known as SAML metadata exchange.

Personalize Your App with Custom Attributes

You might notice that your app displays attributes of the user's identity. These attributes are shared through standard SAML attribute assertions, which is useful when you want to personalize the app by providing more information about the user.

More About Single Sign-On for Your Web App

You've learned the basics of acting as an identity provider for your web app. For more information, use the following resources.

Create a Connected App for Your Web App

A connected app is an application that integrates with Salesforce Identity using APIs and identity services. Connected apps use standard identity protocols like SAML, OAuth, and OpenID Connect to authenticate, provide single sign-on, and provide tokens for use with Salesforce APIs.

Let's create a SAML-based connected app that users can see and administrators can manage.

1. From Setup, enter *Apps* in the Quick Find box, then select **App Manager**.
2. Click **New Connected App**.
3. Give your app a name. Let's call it My SSO App.
4. For Contact Email, enter your email address.
5. Select **Choose one of our sample logos**.
6. In the new window, select a logo you like and copy the URL.
7. Close the window, and paste the URL in the Logo URL field in your Connected App window.
8. Click **Enable SAML**.

You now have the basics of a connected app in place, but you need to connect the app to something. Let's set up another app so you can establish trust between the app and Salesforce Identity for SSO.

Create a Sample Service Provider on Heroku

To implement single sign-on, you need an app that speaks SAML. We've prepared a free sample that gets you up and running quickly. The sample app runs on Heroku. Heroku is a Salesforce Platform offering that provides platform as a service in a wide variety of languages. It also offers an amazing developer experience. As you see, deploying a new app can be as simple as clicking a button. If you don't have a Heroku account, sign up for free at [Heroku](https://heroku.com).

After you have a Heroku account, go to <https://toolbelt.heroku.com> and install the Heroku tool belt. Then follow these steps.

1. In a new browser window, go to <https://github.com/salesforceidentity/heroku-identity-java>.
2. Click **Deploy to Heroku**. A new page in the Heroku Dashboard displays that clones the sample for you.
3. On the dashboard, you can optionally name the app.
4. Click **Deploy for Free**.
5. Heroku copies the app that you'll control. When the copy is complete, click **View**.
6. Click **Login**.

Because you haven't configured the app to trust Salesforce as an identity provider, you see instructions about how to set it up.

Configure Salesforce Identity to Provide Identity for Your App

Teach Salesforce Identity about the SAML configuration of your new app.

1. Copy the Start URL value on your apps page. (It's the same as the Entity ID and ACS URL for this particular app.)
2. Return to your connected app window.
3. Paste the Start URL value into the Start URL, Entity ID, and ACS URL fields.
4. Click **Save**.

You've now configured a connected app with metadata for your sample SAML service provider.

Authorize Your Web App

The Salesforce Identity SAML identity provider understands your app via the connected app, but your users aren't authorized to access it. You still have to configure authorization.

1. In the connected app window, click **Manage**.
2. Scroll to the Profiles section and click **Manage Profiles**.
3. Choose your Customers and System Administrator profiles.
4. Click **Save**.

Anyone with the Customers or System Administrator profile can use SAML to access the app.

Configure Your App to Trust Salesforce Identity

Even though you've described your sample app to Salesforce Identity, your app doesn't yet trust Salesforce to act as an identity provider. You must configure the app to accept SAML messages. This process is known as SAML metadata exchange.

Just as you've provided metadata about the app to Salesforce, you provide metadata about Salesforce to the app. In practice, this process varies from app to app, but the fundamentals remain the same. You give the app:

- A unique name of the identity provider
- URLs where the app runs
- A certificate to validate single sign-on messages from the identity provider

Salesforce Identity exposes standard SAML metadata documents that can be downloaded or accessed via a URL. The sample app you deploy accepts metadata either way. Let's take the easy route and use the URL.

1. Access the SAML metadata through a URL.
 - a. Scroll to the SAML Login Information section and expand the section for your community.
 - b. Copy the Metadata Discovery Endpoint value.
 - c. On a command line, use Heroku Toolbelt to update the configuration of the app: `set --app your_app_name SAML_METADATA=your_metadata_url`

You've now configured your sample service provider to trust your Salesforce Identity IDP.

2. Let's test it!
 - a. Return to your sample app and reload the page.
 - b. You're now automatically signed in as your administrator using SAML.
3. Test the configuration with a user.

- a. In a new private (incognito) browser or window, load your app.
- b. Click **Login**.
- c. Click the Facebook icon. If necessary, log in with Facebook.

You're returned to your app through SSO.

Personalize Your App with Custom Attributes

You might notice that your app displays attributes of the user's identity. These attributes are shared through standard SAML attribute assertions, which is useful when you want to personalize the app by providing more information about the user.

Connected apps let you extend this information through custom attributes. Using custom attributes, you can enrich the data sent to your app declaratively, choosing from attributes of users, their profiles, and their Salesforce org. When the app interacts with Salesforce over SAML or OpenID Connect, these attributes are shared in a standardized way.

1. Go to the Connected Apps page for your app.
2. Scroll to Custom Attributes and click **New**.
3. Set the Attribute key to **Profile**.
4. Click **Insert Field**.
5. Click **\$Profile** and find the name.
6. Click **Insert**.
7. Return to your sample service provider and log out.
8. Click **Login** to get single sign-on, including your new attribute.

Custom attributes are flexible, and you can use the Salesforce formula language to combine or transform attributes for your particular use case. For example, you can create a custom attribute called "IsOver18" with a formula like this.

```
IF(($User.Birthday__c - TODAY() + 6574 ) >= 0, 'false', 'true' )
```

At runtime, the attribute logic looks at a custom date field on the user object, calculates whether the user is over 18, and discloses true or false. This attribute allows you to assert that the user meets a business policy without disclosing the actual birthday to the target application.

For more information on using formulas, review the [Using Formula Fields](#) Trailhead module. You can also construct custom attributes using Apex.

SEE ALSO:

[Retrieve User Information with Custom Attributes](#)

More About Single Sign-On for Your Web App

You've learned the basics of acting as an identity provider for your web app. For more information, use the following resources.

[Setting up Single Sign-On \(Salesforce Classic\) video](#)

[Examples for Setting Up Identity Providers and Service Providers](#)

QUICKER AND EASIER LOGINS

These days, users are given more login options. They can identify themselves using a phone number or email address instead of a username. And, instead of a password, they can verify that their identity with a verification code sent to their email address or mobile device. Salesforce External Identity gives you an easy way to implement these alternate login mechanisms. With a few clicks, you can quickly deploy login and sign-up pages for your community.

You create login pages from Community Workspaces on the Login & Registration (L&R) page of the Administration workspace. Developers can modify existing login pages or create them with Visualforce and Apex.

[Choose a Login Page Type](#)

When you create a community with a template, Salesforce takes care of the login process. You can choose from a few login page types to control the login experience for your community.

[Use the Discoverable Login Page for Simpler Login](#)

If you want users to log in with another identifier than their username, such as a phone number or email address, use the Login Discovery Page. After the user enters the identifier, the process sends a verification code to the user's email address or mobile device. If the user enters the code on the generated Verify page correctly, the user is logged in to your community. Login Discovery supports identity verification via email address and phone number. You can support custom identifiers, such as an employee number or federation ID, in Apex.

[Extend the Login Discovery Handler in Apex](#)

When you select the Login Discovery Page for your login page, Salesforce generates the `AutocreatedDiscLoginHandler`, which contains logic for users to log in with their email address or phone number. You can modify the handler in Apex to customize the login process. For example, you can have users log in with a vendor ID, employee number, or other identifier. And you can customize the handler's lookup logic to invoke the appropriate authentication flow. If your org is configured with multiple identity providers (IdP), your code can "discover" the IdP for the particular user and log the user directly into your community.

[Choose a Self-Registration Page Type](#)

You can choose from a few page types to implement self-registration for your community.

[Use the Configurable Self-Reg Page for Easy Sign-Up](#)

Salesforce Identity provides a default self-registration page that signs up visitors with a username and password. You can give visitors a simpler way to sign up while giving you an easier way to add more value to the self-registration process by using the Configurable Self-Reg Page. For example, you can let users sign up with only an email address or phone number. You choose which information to collect from the users. And you can verify that users are who they say they are before you add them as a member of your community.

[Extend the Configurable Self-Reg Handler in Apex](#)

When you select the Configurable Self-Reg Page registration type on the Login & Registration page, Salesforce generates the `AutoCreatedConfigSelfReg` handler. The handler contains logic for users to register with their email address or phone number. You can modify the handler in Apex to customize self-registration. You might want to add logic to ensure that a new member's email address and phone number are unique in your org. Or you can set up SSO for your new external users.

[View Your Users' Identity Verification Methods](#)

Salesforce maintains information about each user's identity verification history. You can get this information from the user interface or API.

[Create Custom Error Messages in the Login Discovery and Self-Registration Handlers](#)

If a user makes a mistake when trying to log in to or register for your community, the Apex handler displays a generic error message. You can customize the error message that appears on the login, verify, and self-registration pages using the `Auth.DiscoveryCustomErrorException` exception.

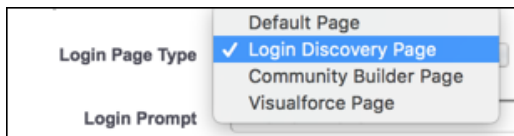
SEE ALSO:

[Customize Login Pages in Apex for Full Control](#)


Choose a Login Page Type

When you create a community with a template, Salesforce takes care of the login process. You can choose from a few login page types to control the login experience for your community.

You choose the login page type on the Login & Registration (L&R) page of the Administration workspace.



Each login page type has a purpose.

- **Default Page**—Provides you a simple login page with basic styling. It's intended to serve as a starting point to customize the look and behavior of the login process.
- **Login Discovery Page**—Gives you a login page that enables users to log in with an identifier other than a username. By default, users can log in with their email address or phone number. This page generates a Login Discovery handler, which you can modify to extend login discovery functionality. The Login Discovery page is branded according to the options set on the L&R page.
- **Community Builder Page**—Lets you use the Community Builder to manage and brand your self-registration page. This page requires users to register a username and password.
- **Visualforce Page**—Lets you enable the custom Visualforce page that you created to control the self-registration process. You can click , and select the page from the list.

1. From Community Workspaces, select **Administration**, and then select **Login & Registration**.
2. From Login Page Type, select the login page type.

If you're trying out various page types, select the Default Page to clear your selections before choosing another page type. To access Community Builder page templates, you first publish your login page in Community Builder.

SEE ALSO:

[Use the Discoverable Login Page for Simpler Login](#)

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To access Community Workspaces:

- Access Community Management OR Manage Communities OR Create and Set Up Communities
- AND is a member of the community

To create and edit Visualforce pages:

- Customize Application

To edit Apex classes:

- Author Apex
- AND Customize Application

Use the Discoverable Login Page for Simpler Login

If you want users to log in with another identifier than their username, such as a phone number or email address, use the Login Discovery Page. After the user enters the identifier, the process sends a verification code to the user's email address or mobile device. If the user enters the code on the generated Verify page correctly, the user is logged in to your community. Login Discovery supports identity verification via email address and phone number. You can support custom identifiers, such as an employee number or federation ID, in Apex.

Create a Login Discovery Page to prompt users to log in with their email address, phone number, or a custom identifier.

1. From Setup, enter *All Communities* in the Quick Find box, then click **Workspaces** next to your community.
2. Select **Administration**, and then select **Login & Registration**.
3. Under Branding Options, customize the login page with your own logo, colors, and background. For details, see *Brand Your Community Using the Login & Registration Page*.
4. For Login Page Type, choose **Login Discovery Page**.

5. For Login Prompt, enter the text that you want to appear as the prompt on your login page. For example, enter *Email Address or Phone Number*.
6. Leave the Login Discovery Handler blank.
7. Click **Create a Login Discovery Handler**.
Salesforce generates a default login discovery handler after you save the page. But continue to fill out the page before you save it.
8. For Execute Login As, choose an admin with Manage Users permission.
9. Make sure **Allow internal users to log in directly to the community** isn't selected.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To access Community Workspaces:

- Access Community Management OR Manage Communities OR Create and Set Up Communities
- AND is a member of the community

To create and edit Visualforce pages:

- Customize Application

To edit Apex classes:

- Author Apex
- AND Customize Application

The Login Discovery Page doesn't support internal users (employees) logging in to the community directly. However, you can customize the handler to redirect users to an IdP.

10. Click **Save.**

Salesforce generates a default login discovery handler and populates the Login Discovery Handler field with its name. This handler accepts an email address or phone number from the login page and emails or texts a verification code to the user. If the user's email or phone number isn't verified, the user is prompted for a password.

You can modify the handler in Apex to extend its functionality.



Note: To support text-message verification, you purchase a license for Identity Verification Credits. This usage-based license provides your org a predetermined number of SMS messages for identity verification.

SEE ALSO:

[View Your Users' Identity Verification Methods](#)

[Passwordless Login Page Using Custom Login Pages](#)

[Extend the Login Discovery Handler in Apex](#)

[Choose a Login Page Type](#)

Extend the Login Discovery Handler in Apex

When you select the Login Discovery Page for your login page, Salesforce generates the `AutocreatedDiscLoginHandler`, which contains logic for users to log in with their email address or phone number. You can modify the handler in Apex to customize the login process. For example, you can have users log in with a vendor ID, employee number, or other identifier. And you can customize the handler's lookup logic to invoke the appropriate authentication flow. If your org is configured with multiple identity providers (IdP), your code can "discover" the IdP for the particular user and log the user directly into your community.

Before you extend Login Discovery, let's review the Apex code for the generated handler. Even if you don't want to change the default implementation, it's helpful to understand how the login discovery process works. The Apex code contains comments where you can customize the code.

1. Find the name of the Login Discovery handler from the Login & Registration page.
 - a. From Setup, enter *All Communities* in the Quick Find box, then click **Workspaces** next to your community.
 - b. Select **Administration**, then select **Login & Registration**.
 - c. Under Login Page Setup, notice the name next to Login Discovery Handler, for example, `AutocreatedDiscLoginHandler1535677786093`.

2. Locate the handler from Setup.

- a. From Setup, enter *Apex* in the Quick Find box, then click **Apex Classes** next to your community.
- b. To view the code, from the list of Apex classes, locate the handler, and click the name to open it.

3. Optionally, change the autocreated handler name.

- a. Click **Edit**.

The name of the Login Discovery handler appears in the first line of code: `global class AutocreatedDiscLoginHandler1535677786093 implements Auth.LoginDiscoveryHandler`.

- b. Replace **AutocreatedDiscLoginHandler1535677786093** with your own, for example, `DiscoveryHandlerMyCommunity`.

- c. Click **Save**.

- d. Refresh your Login & Registration page to see the renamed handler in the Login Discovery Handler field.

4. Now let's look at the Apex code.

This Apex code example implements the `Auth.LoginDiscoveryHandler` interface. It checks whether the user who is logging in has a verified email or phone number, depending on which identifier was supplied on the login page. If verified, with `Auth.VerificationMethod.EMAIL` or `Auth.VerificationMethod.SMS`, we send a challenge to the identifier, either the user's email address or mobile device. If the user enters the code correctly on the verify page, the user is redirected to the community page specified by the start URL. If the user isn't verified, the user must enter a password to log in. The handler also checks that the email and phone number are unique with this code: `users.size() == 1`.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To access Community Workspaces:


- Access Community Management OR Manage Communities OR Create and Set Up Communities
- AND is a member of the community

To create and edit Visualforce pages:

- Customize Application

To edit Apex classes:

- Author Apex
- AND Customize Application

 **Note:** Passwordless login works only with verified methods. You can check the verification status on the User object, for example, with User list view, a report, or the API. Make sure that your solution handles the case where the user doesn't have a verification method. This code example falls back to a password.

The default discoverable login handler checks whether the user entered a valid email address or phone number before redirecting the user to the verification page. If an invalid entry is made, the handler returns an error. Because this behavior is vulnerable to user enumeration attack, make sure that your solution prevents this attack. For example, you can create a dummy page similar to the verification page and redirect the user to the dummy page when invalid user identifier is entered. Also, use generic error messages to avoid providing additional information.

The `discoveryResult` function calls the `Site.passwordlessLogin` method to log the user in with the specified verification method. The `getSsoRedirect` function looks up whether the user logs in with SAML or an Auth Provider. Add the implementation-specific logic to handle the lookup.

SEE ALSO:

[Apex Developer Guide: LoginDiscoveryHandler Interface](#)

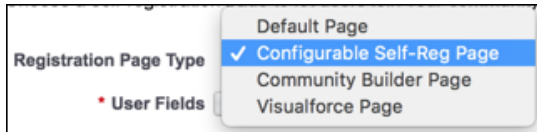
[Passwordless Login Page Using Custom Login Pages](#)

[Use the Discoverable Login Page for Simpler Login](#)


Choose a Self-Registration Page Type

You can choose from a few page types to implement self-registration for your community.

You choose the self-registration page type on the Login & Registration (L&R) page of the Administration workspace.



Each self-registration page type has advantages.

- **Default Page**—Gives you a simple self-registration page with basic styling. It's intended to serve as a starting point to customize the look and behavior of self-registration.
- **Configurable Self-Reg Page**—Gives you the ability to decide which identifiers guests register with, such as an email address, phone number, or username. You choose whether to require them to verify their identity and the information to collect when they register. With this page, you make it easy for guests to sign up with a minimum amount of information. The Configurable Self-Reg Page is branded according to the options set on the L&R page.
- **Community Builder Page**—Lets you use the Community Builder to manage and brand your self-registration page. This page requires users to register a username and password.
- **Visualforce Page**—Lets you enable your custom Visualforce page for self-registration. Click , and select your page from the list.

Here are some points to consider when deciding which registration page to use.

- The Configurable Self-Reg Page generates a handler that you can modify to extend self-registration functionality. It gives you more flexibility while avoiding the need to create a custom page.
- The Community Builder Page is an easy way to set up self-registration using the Community Builder design tool, but limits you to prompt for username and password. If you use another self-registration page, you can still use Community Builder to manage your community.
- The Default and Visualforce pages give you complete control over the look and feel of the self-registration process.

1. From Community Workspaces, select **Administration**, and then select **Login & Registration**.
2. Select the self-registration page type.

If you're trying out various page types, select the Default Page to clear your selections before choosing another page type. To access Community Builder page templates, you first publish your login page in Community Builder.

SEE ALSO:

[Use the Configurable Self-Reg Page for Easy Sign-Up](#)

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To access Community Workspaces:

- Access Community Management OR Manage Communities OR Create and Set Up Communities
- AND is a member of the community

To create and edit Visualforce pages:

- Customize Application

To edit Apex classes:

- Author Apex
- AND Customize Application

Use the Configurable Self-Reg Page for Easy Sign-Up

Salesforce Identity provides a default self-registration page that signs up visitors with a username and password. You can give visitors a simpler way to sign up while giving you an easier way to add more value to the self-registration process by using the Configurable Self-Reg Page. For example, you can let users sign up with only an email address or phone number. You choose which information to collect from the users. And you can verify that users are who they say they are before you add them as a member of your community.

Create a configurable sign-up page from your community's Login & Registration page. You can select Configurable Self-Reg Page when you create the login page using a Login Discovery Page, Visualforce Page, or Default Page. If your login page type is Community Builder Page, Configurable Self-Reg Page isn't available.

1. From Setup, enter *All Communities* in the Quick Find box, then click **Workspaces** next to your community.
2. Select **Administration**, and then select **Login & Registration**.
3. Select **Allow external users to self-register**.
4. For Registration Page Type, select **Configurable Self-Reg Page**.
The L&R page displays fields for setting up self-registration.

5. For User Fields, select which fields to display on the self-registration page.

Users are required to supply this information when they register. If the information isn't supplied, the handler populates the fields with dummy values.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To access Community Workspaces:

- Access Community Management OR Manage Communities OR Create and Set Up Communities
- AND is a member of the community

To create and edit Visualforce pages:

- Customize Application


To edit Apex classes:

- Author Apex
- AND Customize Application

6. Leave Configurable Self-Reg Handler blank.
7. Click **Create a configurable self-reg handler**.
Salesforce generates a self-registration handler after you save the page. But continue to fill out the page before you save it.
8. For Execute Registration As, choose a Salesforce admin with Manage Users permission.
9. For Verification Method, if you want visitors to verify their identity before creating the user, select either **Email** or **Text Message**. If you don't require verification, select **None**.

Require a verification method if you want to create a user with a verified email or phone number. When verified, the user can log in from a login page created with Login Discovery. If you choose **None**, users must supply a password when they log in again.

The verification method is independent of the user fields you selected. For example, if you select **Email** as a user field and **None** as the verification method, Salesforce collects the user's email address, but doesn't send a verification code.
10. If you want users to create a password when they sign up, select **Include Password**.
When selected, the self-registration form contains Password and Confirm Password fields.

 **Note:** If you select **None** as the verification mode, you must select **Include Password**. Otherwise, the new user can't log in.

Collecting a password from the new user gives the user two ways to log in. The password can be helpful, for example, if the admin requires text-message verification but the user leaves the phone at home. The user can fall back to logging in with a password.
11. For Profile, select a profile configured for your org, such as an External Identity, Customer Community User, or Partner Community User profile.
This setting assigns new users a profile to control access. The profile selected here is passed to the Configurable Self-Reg handler. Unless modified, the new user is assigned this profile. If you don't specify a value here, make sure that your handler sets a user profile explicitly before inserting a user. Every user requires a profile.
12. For Account, choose the account to contain your community members. Each member is listed as a contact in the account.
The account selected here is passed to the Configurable Self-Reg handler. Leave the field blank if your org is set up for Person accounts. If you want, you can override Account in Apex. For example, you can disassociate external users from accounts to create contactless users.
13. Click **Save**.
Salesforce generates a configurable self-registration handler and populates the Configurable Self-Reg Handler field with its name. The resulting sign-up form prompts users to register with the user fields that you selected.
14. Check whether self-registration works.
 - a. From a private (incognito) browser, open your community login page.
 - b. Click **Sign Up**, and register a new user.
 - c. If prompted for an email address or phone number, enter your own to complete the verification process.
 - d. If verification succeeds, log in to the community.

You can modify the handler in Apex to extend its functionality. For example, you can add logic to ensure that the new member's email address and phone number are unique in your org. You can also control how the user is created and populate fields on the self-reg form with your values. However, the handler can't change the prompts on the self-registration form. These prompts are determined by the user fields selected on the L&R page, such as First Name, Last Name, and so on.

SEE ALSO:

[Choose a Self-Registration Page Type](#)

Extend the Configurable Self-Reg Handler in Apex

When you select the Configurable Self-Reg Page registration type on the Login & Registration page, Salesforce generates the `AutoCreatedConfigSelfReg` handler. The handler contains logic for users to register with their email address or phone number. You can modify the handler in Apex to customize self-registration. You might want to add logic to ensure that a new member's email address and phone number are unique in your org. Or you can set up SSO for your new external users.

You can find the generated Configurable Self-Reg handler from the Setup Apex Classes page. Its name begins with `AutoCreatedConfigSelfReg`. When Salesforce generates the default handler, it appends digits to ensure that the name is unique. You can rename the handler to something more meaningful.

Before you extend the configurable self-reg handler, let's review the Apex code for the generated handler. Even if you don't want to change the default implementation, it's helpful to understand how the configurable self-registration process works. When you open the handler in Apex, you can see that the code contains comments in places where you can customize the code.

1. Find the name of the Configurable Self-Reg handler on the Login & Registration page.
 - a. From Setup, enter *All Communities* in the Quick Find box, then click **Workspaces** next to your community.
 - b. Select **Administration**, and select **Login & Registration**.
 - c. Under Registration Page Configuration, notice the name next to Configurable Self-Reg Handler, for example, `AutocreatedConfigSelfReg1535145601649`.
2. Locate the handler from Setup.
 - a. From Setup, enter *Apex* in the Quick Find box, then click **Apex Classes** next to your community.
 - b. To view the code, from the list of Apex classes, locate the handler, and click the name to open it.
3. Optionally, rename the handler for your convenience.
 - a. Click **Edit**.
 The name of the Configurable Self-Reg Handler appears in the first line of code: `global class AutocreatedConfigSelfReg1532475901849 implements Auth.ConfigurableSelfRegHandler.`
 - b. Replace the autocreated name with your own, for example, `ConfigureSelfRegHandlerMyCommunity`.
 - c. Click **Save**.
 - d. To see the renamed handler in the Configurable Self-Reg Handler field, refresh your Login & Registration page.
4. Now let's look at the Apex code.

Verification occurs by email if the admin chose Email as the verification method when setting up the Configurable Self-Reg handler on the Login & Registration (L&R) page. When a visitor clicks the sign-up link from the login page, Salesforce prompts for an email address and then sends a one-time password to the specified email address. If the visitor enters the verification code successfully on the verify page, the user is created and logged in. Likewise, if the admin chose Text Message as the verification method on the L&R page, the visitor is prompted to enter a phone number. Salesforce sends a challenge (verification code) via SMS to the user. If successful, the user is created and logged in. Requiring verification before creating a user reduces the number of dummy users cluttering your org.

The `Auth.ConfigurableSelfRegHandler` class contains logic for generating the user fields required to create a user in case the user doesn't supply them. The handler generates default values, ensuring that the values are unique by appending a timestamp. You can modify the handler to make sure that the email address and phone number of the external user are also unique.

SEE ALSO:

[Apex Developer Guide: ConfigurableSelfRegHandler Interface](#)

View Your Users' Identity Verification Methods

Salesforce maintains information about each user's identity verification history. You can get this information from the user interface or API.

User Identity Verification Methods from Setup

You can determine which verification methods your users have registered.

1. From Setup, enter *Users* in the Quick Find box, then select **Users**.
2. Click **Create New View**.
3. Under Available Fields, select these fields that report verification method activity, and click **Save**.

Admin Trusted Mobile Number

Indicates whether the user has a mobile phone number that an admin added or the user self-registered. Salesforce can text a verification code to that number when verifying identity for authenticator apps but not for passwordless login.



Note: As a security measure, when users add or update a mobile number in Advanced User Details, they must log in again to verify their identity. As a result, unsaved changes in the app are lost. To disable this security measure, contact Salesforce Support.

One-Time Password App

Indicates whether the user has connected an authenticator app that generates verification codes, also known as time-based one-time passwords. The user can verify identity by entering a code generated by the app.

Salesforce Authenticator

Indicates whether the user has connected the Salesforce Authenticator mobile app. The user can verify identity by approving a notification sent to the app.

Temporary Code

Indicates whether the user has a temporary verification code. Admins or non-admin users with the Manage Two-Factor Authentication in User Interface permission generate temporary codes and set when the code expires.

U2F Security Key

Indicates whether the user has registered a U2F security key. The user can verify identity by inserting the security key into a USB port.

User Verified Email

Indicates whether the user self-registered and verified an email address. Salesforce can send a verification code to the user at that email address.

User Verified Mobile Number

Indicates whether the user self-registered and verified a mobile phone number. Salesforce can text a verification code to the user at that number.

USER PERMISSIONS

To view and customize the Identity Verifications Methods report from Identity Verification History:

- Manage Two-Factor Authentication in User Interface

To access the Identity Verifications Methods report from the Administrative Reports folder in Reports:

- View Setup and Configuration

To create custom reports and dashboards about identity verification history:

- Manage Two-Factor Authentication in API

API Identity Verification

Determine the user's verification methods with `TwoFactorMethodsInfo`.

Set verification methods for external users with `System.UserManagement.registerVerificationMethod`. This example registers identity verification by text message.

```
System.UserManagement.registerVerificationMethod(Auth.VerificationMethod.SMS, '/');
```

Likewise, you can deregister a verification method, for example, when a user loses access to the mobile device, with the `System.UserManagement.deregisterVerificationMethod`.

```
System.UserManagement.deregisterVerificationMethod(Auth.VerificationMethod.SMS, '/');
```

SEE ALSO:

[Object Reference for Salesforce and Lightning Platform: TwoFactorMethodsInfo](#)

[Apex Developer Guide: UserManagement Class](#)

[Passwordless Login Coding Considerations](#)

[See How Your Users Are Verifying Their Identity](#)


[Use the Discoverable Login Page for Simpler Login](#)

Create Custom Error Messages in the Login Discovery and Self-Registration Handlers

If a user makes a mistake when trying to log in to or register for your community, the Apex handler displays a generic error message. You can customize the error message that appears on the login, verify, and self-registration pages using the `Auth.DiscoveryCustomErrorException` exception.

1. From Setup, enter `Apex` in the Quick Find box, then click **Apex Classes**.
2. From the list of Apex classes, find the class that implements `Auth.LoginDiscoveryHandler` or `Auth.ConfigurableSelfRegHandler`. You can find the name of the handlers on the Login & Registration page of the Administration workspace. If you haven't renamed the handlers, the names are similar to `AutocreatedDiscLoginHandler1535677786093` and `AutocreatedConfigSelfReg1532475901849`.
3. Edit the Apex class to add the custom error message using the `Auth.DiscoveryCustomErrorException` exception.
For example, `throw new Auth.DiscoveryCustomErrorException('Custom error message defined by the admin.');`
4. Optionally, instead of entering the custom error message in the Apex class, you can create the error message as a custom label.
Custom labels are custom text values that can be translated to the user's native language. In this example, `CustomError` is a custom label.

```
throw new
Auth.DiscoveryCustomErrorException(Label.CustomError)
```

 **Note:** You create a custom error message from the Custom Labels Setup page. The error message can be up to 200 characters.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To access Community Workspaces:

- Access Community Management OR Manage Communities OR Create and Set Up Communities
- AND is a member of the community

To create and edit Visualforce pages:

- Customize Application

To edit Apex classes:

- Author Apex
- AND Customize Application

Login Discovery custom error messages appear on the login page. Configurable self-registration error messages appear on the verify page if you selected the Email or Text Message verification method when configuring self-registration. If you selected **None**, the error message appears on the self-registration page.

SEE ALSO:

[Salesforce Help: Custom Labels](#)

[Apex Developer Guide: LoginDiscoveryHandler Interface](#)

[Apex Developer Guide: ConfigurableSelfRegHandler Interface](#)

DYNAMIC URLs: DETERMINE YOUR BRAND AT RUN TIME

Add dynamic branding to your external community to extend the functionality of your login page. From the Login & Registration page of the Administration workspace, you can brand your external identity community login experience on pages used to log in users, verify identities, reset passwords, and register new members. You can replace the Salesforce logo with yours, change the color of your background and login button, and modify the footer text. You can also provide a URL to display content on the right side of your login form. With dynamic branding, you control and customize branding at run time depending on current circumstances, like who the user is or where the user is logging in from.

[Dynamic Branding for Multiple Brands](#)

Dynamic branding is ideal for multi-brand companies. You can create a single login page that determines which brand appears at run time. All brands are served from the same login page that serves the same community.

[Dynamic Branding Using the Login & Registration Page](#)

You can implement dynamic branding for login pages from Community Workspaces on the Login & Registration (L&R) page of the Administration workspace. Create dynamic URLs by including a placeholder that resolves at run time.

SEE ALSO:

[Implement Embedded Login](#)

[Customize Your Login Pages with Community Builder](#)

Dynamic Branding for Multiple Brands

Dynamic branding is ideal for multi-brand companies. You can create a single login page that determines which brand appears at run time. All brands are served from the same login page that serves the same community.

In our scenario, Fix Corporation has a single external community to handle its two brands, Fix Coffee and Fix Chocolate. The corporation uses dynamic branding to customize the login experience for each brand. When customers click a link to log in, Fix presents a login page based on the brand indicated in the login URL.

Fix has a set of resources—including a logo and right-frame URL—for each brand. These resources are hosted on its content management system (CMS). So, for example, the URL to the Fix Coffee logo is `https://www.my-cms.com/coffee/logo.png`. Likewise, the URL to the Fix Chocolate logo is `https://www.my-cms.com/chocolate/logo.png`.

Fix implements dynamic branding by using a placeholder, or experience ID (expid) in the URL to represent each brand. That is, the URL contains the expid dynamic URL, which is then replaced by its value at run time.

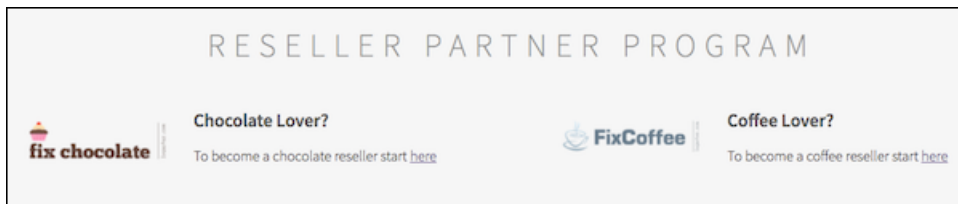
In our scenario, the dynamic URL uses the expid: `https://www.my-cms.com/{expid}/logo.png`. Then {expid} becomes either `https://www.my-cms.com/coffee/logo.png` or `https://www.my-cms.com/chocolate/logo.png` at run time.

If the customer at run time clicks a link to log in to Fix Coffee, the link contains an expid query parameter set to `expid=coffee`:

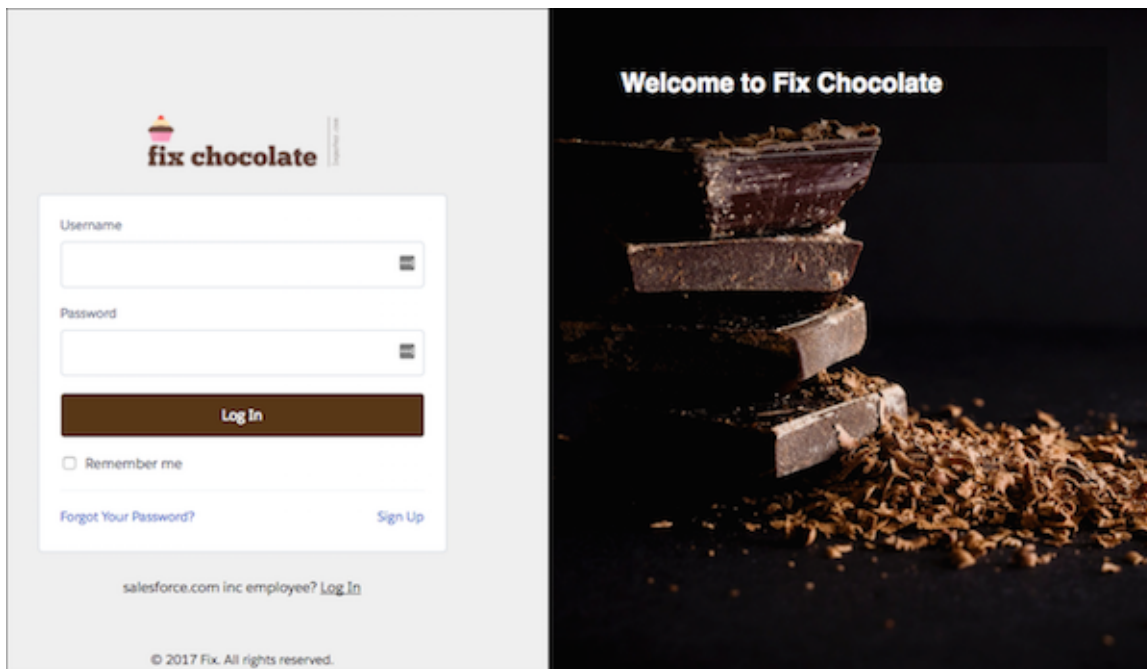
`https://fix.force.com/login?expid=coffee`

This URL is the login endpoint to the community branded as Fix Coffee.

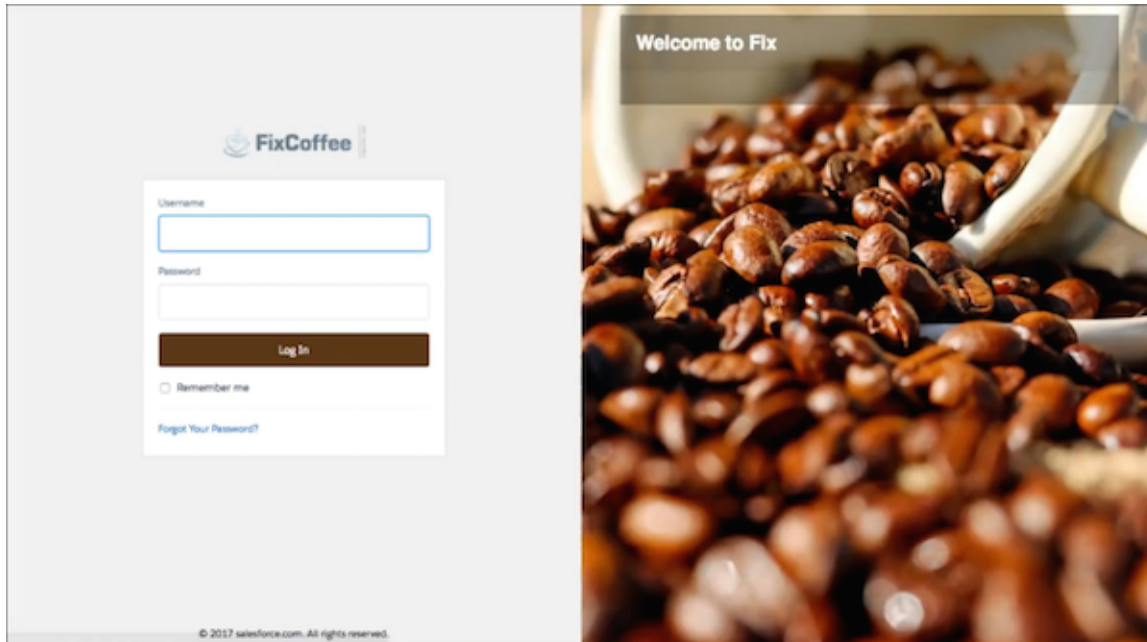
It's up to your implementation to determine how the expid query parameter is set. For example, it can be contained in an email link or set with a click of a button. In our scenario, the user selects the brand from the Reseller Partner Program page.



When the user selects chocolate, the link to the login page is set to `https://fix.force.com/login?expid=chocolate`. Chocolate fans are greeted with the Fix Chocolate login page, complete with its chocolate-branded logo and right-side content.



A click from a coffee fan generates the login page URL `https://fix.force.com/login?expid=coffee`, and the Fix Coffee login page appears with its coffee-branded logo and right-side content.



You use the Login & Registration page of the Administration workspace to set up your community's login page, including dynamic branding. The Login & Registration page controls three types of login pages: the Default page that Salesforce provides, the Community Builder page that comes with Community Builder, and the custom login page that you create with Visualforce.

Dynamic Branding Using the Login & Registration Page

You can implement dynamic branding for login pages from Community Workspaces on the Login & Registration (L&R) page of the Administration workspace. Create dynamic URLs by including a placeholder that resolves at run time.

The dynamic URL contains the `expid` placeholder for your logo and right-frame content, for example, `https://www.my-cms.com/{expid}/logo.png` and `https://www.my-cms.com/{expid}/index.php` respectively. The right-frame URL displays the contents of the URL in an iframe. At run time, the external community replaces the dynamic URL with the brand specified by the `expid` value (for example, `coffee`) in the query parameter.

The brand that appears at run time depends on the `expid` dynamic URL (the `expid` signal) that you specify on the L&R page. For example, given the dynamic URL, `https://www.my-cms.com/{expid}/logo.png`, Fix Coffee appears when the login URL is `https://fix.force.com/?expid=coffee`.

You deliver different brands, or user experiences, based on how the `expid` query parameter gets set in your implementation. You might set it in a URL that you send to your community users in an email. Or, you might set the URL on a login button. You can also set the query parameter programmatically with the `setExperienceId` method of the `System.Site` class.

In addition to using dynamic branding for the login process, you can use it to brand the user's experience during the entire user session. To do so, instruct the community to store the `expid` value in the user's browser. When stored, you can retrieve it as long as the user's logged in to the community.

When specifying your logo and right-frame URLs, keep in mind the following.

- Most browsers don't allow mixing the `http://` and `https://` protocols on the same page.
- The URL for your right-frame content must use the same protocol as your community. Your resource and test or demo servers must also use the same protocol.

Generally, we recommend using https:// for your logo and right-frame URLs. However, in local and test environments, http:// can be more flexible.

[Set Up Dynamic URLs](#)

You set up dynamic URLs for your login pages from the Login & Registration (L&R) page of the Administration workspace.

SEE ALSO:

[Set Up Dynamic URLs](#)

[Dynamic Branding Using Custom Login Pages](#)

Set Up Dynamic URLs

You set up dynamic URLs for your login pages from the Login & Registration (L&R) page of the Administration workspace.

Set up dynamic branding from the Community Workspaces L&R page. You can add custom login pages to your community regardless of the template that you used to create the community.

The screenshot shows the 'Administration' workspace for 'My VF EI Community'. The left sidebar lists various settings: Settings, Preferences, Members, Tabs, Branding, **Login & Registration** (selected), Emails, Pages, and Rich Publisher Apps. The main content area is titled 'Login & Registration'. It contains three sections: 'Logo', 'Background', and 'Right Frame'. In the 'Logo' section, the 'Choose Logo Type' dropdown is set to 'URL', and the 'Logo URL' field contains 'https://www.my-cms.com/{expid}.logo.png'. In the 'Background' section, the 'Background Color' is set to '#B1BAC1'. In the 'Right Frame' section, the 'Right Frame URL' field contains 'https://www.my-cms.com/{expid}.index.php'. Red circles highlight the 'URL' dropdown and the 'Right Frame URL' field.

1. From Setup, enter *Communities* in the Quick Find box, and select **All Communities**.
2. Next to your community, select **Workspaces**.
3. Select **Administration**, and then select **Login & Registration**.
4. Enter dynamic URLs for the logo and right-frame content, for example, `https://www.my-cms.com/{expid}/logo.png` and `https://www.my-cms.com/{expid}/index.php`.

SEE ALSO:

[Brand Your Community Using the Login & Registration Page](#)

[Dynamic Branding Using the Login & Registration Page](#)

CUSTOMIZE LOGIN PAGES IN APEX FOR FULL CONTROL

You can create a custom login page completely in Apex for full control over the look and behavior of your community's login page. Usually, the point-and-click Login & Registration Page gives you most of what you need to handle your community's login experience. But if you want to customize login further, we recommend that you modify the Apex controllers and Visualforce pages that Salesforce provides rather than starting from scratch. For example, you can add dynamic branding so that the login page displayed is decided at run time. Or you can use Apex to implement passwordless login programmatically.

EDITIONS

Available in: **Salesforce Classic** and **Lightning Experience**

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

[Create a Custom Login Page](#)

Create your custom login page in Visualforce, and associate it with your community from the Login & Registration page of the Administration workspace. You can also create custom Visualforce pages for password, forgot password, and self-registration. You can add these login pages to your community regardless of the template you use to create the community.

[Dynamic Branding Using Custom Login Pages](#)

For custom login pages, you can implement dynamic URLs with a custom login Apex controller. Use the Apex methods `getExperienceId` and `setExperienceId` of the `System.Site` class to retrieve and store the expid value.

[Passwordless Login Page Using Custom Login Pages](#)

You use the Login & Registration (L&R) page to implement passwordless login. For complete control, you can implement passwordless login in Apex as a custom login page. The custom login controller includes logic to locate the user based on the identifier entered on the login page. It then checks which verification methods the user has previously registered. The controller logs in the user by whichever method is configured.

Create a Custom Login Page

Create your custom login page in Visualforce, and associate it with your community from the Login & Registration page of the Administration workspace. You can also create custom Visualforce pages for password, forgot password, and self-registration. You can add these login pages to your community regardless of the template you use to create the community.

1. To customize the behavior of the login page, create your own Apex controller. Include the `Site.login()` Apex method. For example:

```
global PageReference login()  
    { return Site.login(username, password, returnUrl); }
```

2. To customize the look of your login page, create a Visualforce page. To determine how the Apex controller gets invoked, include Apex code. For example, when the user clicks the Login button:

```
apex:commandButton action="{!login}" value="Sign in" id="login-submit" styleClass="btn btn-lg btn-primary btn-block"/>
```
3. Set up public access to make your custom objects, Apex controllers, and Visualforce available externally. (This step is often overlooked.)
 - a. To display your org's Visualforce pages publicly, from Community Workspaces, select **Administration**, select **Pages**, and then click **Go to Force.com**.
 - b. Click **Public Access Settings**.
 - c. Under Enabled Visualforce Page Access, click **Edit**.
 - d. Select the Visualforce pages you created, add them to **Enabled Visualforce Pages**, and save your changes.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To create and edit Visualforce pages:

- Customize Application

To edit Apex classes:

- Author Apex



AND

Customize Application

Enable Visualforce Page Access


Select the Visualforce pages that you want to make accessible at this Salesforce site.


Available Visualforce Pages		Enabled Visualforce Pages
AnswersHome	Add ▶	BandwidthExceeded
ChangePassword		CommunitiesLanding
ForgotConfirm	Remove ◀	CommunitiesLogin
IdeasHome		CommunitiesSelfReg
MyProfilePage		CommunitiesSelfRegConfirm
SiteTemplate		CommunitiesTemplate
StdExceptionTemplate		Exception
Unauthorized		FileNotFoundException
		ForgotPassword
		ForgotPasswordConfirm
		InMaintenance
		SiteLogin
		SiteRegister
		SiteRegisterConfirm


- e. If necessary, enable Apex classes for public access. Under Enabled Apex Class Page Access, click **Edit**, select the classes, and add them to **Enabled Apex Classes**.
4. From your Login & Registration page, replace the community's default pages with your custom Visualforce pages.
 - a. From Community Workspaces, select **Administration**, and then select **Login & Registration**.
 - b. Under Login Page Setup, select **Visualforce Page**, click , and then select your custom Visualforce page from the list.
 - c. If you created Visualforce pages for password or registration, select **Visualforce Page**, click , and select your custom Visualforce page from the list.

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: Visualforce Page SiteLogin 

☐ Allow internal users to log in directly to the community 

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


☒ MyNGO username and password


Logout Page URL

Full URL: 

Password Pages

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


Forgot Password: Visualforce Page ForgotPassword 

Reset Password: Visualforce Page ChangePassword 

Registration Page Configuration

☒ Allow external users to self-register

Choose a self-registration page to let users join your community.

Registration Page Type: Visualforce Page SiteRegister 

5. View your custom login page from a browser in private (incognito) mode.

[Create a Custom Login Page from a Github Project](#)

When creating custom code for your login page, it's often helpful to start out with an existing example for guidance. Salesforce Identity provides a Github custom login page project that you can modify.

Create a Custom Login Page from a Github Project

When creating custom code for your login page, it's often helpful to start out with an existing example for guidance. Salesforce Identity provides a Github custom login page project that you can modify.

1. On Github, locate the Salesforce Identity [custom login page project](#).
 - a. Download the project to your computer.
 - b. Upload the zipped `_include` directory from the project to Salesforce as a static resource called `include` with Cache Control set to **Public**.
 - c. Create a field set named `Registration` to contain the user information for your community members, such as the user's name, company name, and email.
 - d. If you want to set up an Auth. provider for social sign-on, create a custom URL field on the User object.
2. Install the Apex controllers and Visualforce pages from the Github project to your org according to the project's README.
3. Make your Visualforce pages available publicly.
 - a. From Community Workspaces, select **Administration**, select **Pages**, and then click **Go to Force.com**.
 - b. Click **Public Access Settings**.
 - c. Under Enabled Visualforce Page Access, click **Edit**.
 - d. Select the Visualforce pages you created, add them to **Enabled Visualforce Pages**, and save your changes.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To create and edit Visualforce pages:

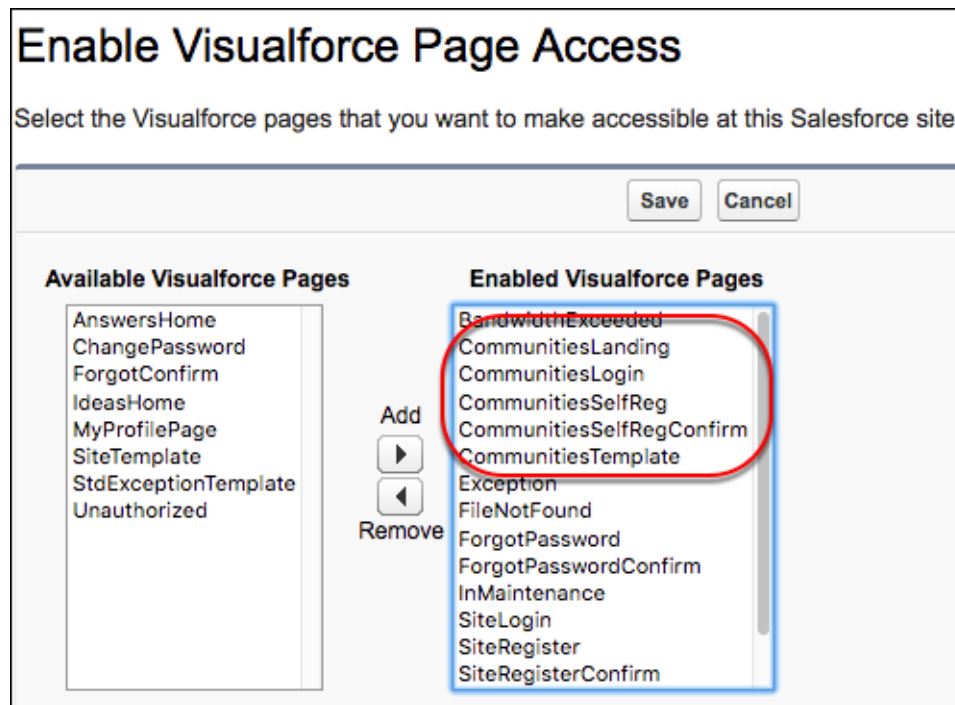
- Customize Application


To edit Apex classes:

- Author Apex

AND

Customize Application



4. Replace the community's default pages with your new Visualforce pages.
 - a. From Community Workspaces, select **Administration**, and then select **Login & Registration**.
 - b. Under Login, select **Visualforce** for the page type, enter the name of your Visualforce page in the search field, and click .
 - c. In the search results window, click the name of your page to select it, and click **Save**.

To revert to the default page, select **Default Page**.
5. View your custom login page from a browser in private mode.
6. Make it your own.

Dynamic Branding Using Custom Login Pages

For custom login pages, you can implement dynamic URLs with a custom login Apex controller. Use the Apex methods `getExperienceId` and `setExperienceId` of the `System.Site` class to retrieve and store the `expid` value.

The brand that appears at run time depends on the `expid` dynamic URL that you specify on the Apex page. The dynamic URL, `https://www.my-cms.com/{expid}/logo.png`, displays the Fix Coffee brand when the login URL is `https://fix.force.com/?expid=coffee`. In this example, the background image of the login page depends on the value of `expid`.

```
<apex:page docType="html-5.0" controller="CustomLoginController" showHeader="false"
  sidebar="false"
  <style>
    body {
      background-image: url("https://www.my-cms.com/{!ExpID}/promo.jpg");
      width: 500px;
```



```

        clear: both;
        margin: 40 px 50px;
        vertical-align: middle;
    }
    h2 { color:#5F9EA0; }
</style>

```

In your custom login controller, include code to extract the expid value from the query parameter.

```

Global CustomLoginController()
{
    ...
    Expid = getExpidFromURL();

    //Get the expid parameter from query string
    Public string getExpidFromURL()
    {
        String expid = '';
        expid = ApexPages.currentPage().getParameters().get('expid');
        Return expid;
    }
    ...
}

```

In your custom login controller, also include the `setExperienceId` method of the `System.Site` class to store the expid value in the user's browser. When stored, you can retrieve it at any time during the lifetime of the user's session.

```

Global CustomLoginController()
{
    ...
    Expid = getExpidFromURL();
    site.SetExperienceId(expId);
    ...
}

```

[Extend an Endpoint with the Experience ID](#)

You can set the experience ID value by extending supported endpoints.

SEE ALSO:

[Dynamic Branding Using the Login & Registration Page](#)

Extend an Endpoint with the Experience ID

You can set the experience ID value by extending supported endpoints.

The value must contain alphanumeric characters only, up to 30 characters.


Extend the following endpoints with `expid_value`.

- **`community-url/services/oauth2/authorize/expid_value`**
- **`community-url/idp/endpoint/HttpPost/expid_value`**
- **`community-url/idp/endpoint/HttpRedirect/expid_value`**

Extend the following endpoints with `expid={value}`.

- `community-url_login_page?expid={value}`
- `community-url/CommunitiesSelfReg?expid={value}`
- `community-url/.well-known/auth-configuration?expid={value}`
- `secur/forgotpassword.jsp?expid={value}`

For example, pass in a different expid value to the self-registration page to deliver a different registration flow for each brand.

 **Note:** The experience ID value is captured in a cookie rather than in the URL. For example, if the `expid_value` is set to `Customers`, an associated cookie has the `expid_Customers` value.

Passwordless Login Page Using Custom Login Pages

You use the Login & Registration (L&R) page to implement passwordless login. For complete control, you can implement passwordless login in Apex as a custom login page. The custom login controller includes logic to locate the user based on the identifier entered on the login page. It then checks which verification methods the user has previously registered. The controller logs in the user by whichever method is configured.

Instead of implementing the passwordless login process entirely in Apex, you have a mid-way option. Create a Login Discovery login page from the L&R page. Then modify the generated LoginDiscovery handler to incorporate your customizations.

Here's the code included in the login controller to define which verification methods to handle. The order in which the methods appear sets the challenge order when a user has registered multiple verification methods. Upon successful verification, the `site.passwordlessLogin` identity verification method is invoked to log in the user.

```
// Add verification methods in priority order
methods = new List<Auth.VerificationMethod>();
if (AvailableMethods.HasUserVerifiedMobileNumber)
    methods.add(Auth.VerificationMethod.SMS);
if (AvailableMethods.HasUserVerifiedEmailAddress)
    methods.add(Auth.VerificationMethod.EMAIL);
if (AvailableMethods.HasSalesforceAuthenticator)
    methods.add(Auth.VerificationMethod.SALESFORCE_AUTHENTICATOR);
if (AvailableMethods.HasTotp)
    methods.add(Auth.VerificationMethod.U2F);
if (AvailableMethods.HasU2F)
    methods.add(Auth.VerificationMethod.U2F);

return Site.passwordlessLogin(uid, methods, startURL);
```

`System.Site.passwordlessLogin` returns a `PageReference` that redirects the user to the Salesforce verification page. Generated verification pages use the logo and background color specified on the Login & Registration page of the Communities Administration workspace.

Apex Methods

The Apex methods you use for identity verification and passwordless login include:

UserManagement.registerVerificationMethod

Registers an identity verification method, such as an email address or phone number. Only users can register verification methods. But you and users can deregister them.

UserManagement.deregisterVerificationMethod

Deregisters an existing identity verification method.

Auth.VerificationMethod enum

Gets the available verification methods, such as EMAIL, SALESFORCE_AUTHENTICATOR, SMS, TOTP, and U2F.

UserManagement.formatPhoneNumber

Formats a mobile phone number to ensure that it's in the format required by Salesforce. After formatting the phone number, you update the mobile field of the user's record with the output of `formatPhoneNumber`.

Site.passwordlessLogin

Logs in a user to a community using an identity verification method that the user previously registered.

For details, see the *Apex Developer Guide*.

PasswordlessLogin Code Example

Here's a simple code example of an Apex controller containing the `passwordlessLogin` method.

```
global with sharing class MFILoginController
{
    //Input variables
    global String input {get; set;}
    public String startURL {get; set;}
    public List<Auth.VerificationMethod> methods;
    public String error;

    global MFILoginController()
    {
    }

    global PageReference login()
    {
        List<User> users = null;

        // Empty input
        if(input == null || input == '')
        {
            error = 'Enter Username';
            return null;
        }
        users = [select name, id, email from User where username=:input];
        if(users == null || users.isEmpty())
        {
            error = 'Can\'t find a user';
            return null;
        }
        ID uid = users[0].id;
        if (startURL == null) startURL = '/';

        // Check which verification methods the user has already registered
        TwoFactorMethodsInfo AvailableMethods = [select
            HasUserVerifiedMobileNumber,
            HasUserVerifiedEmailAddress,
            HasSalesforceAuthenticator,
```

```

        HasTotp,
        HasU2F from TwoFactorMethodInfo where userId=:uid];

// If no verification method is available, prompt the user to enter a password
// and invoke the site.login(usr,pwd) method
// if (AvailableMethods.size() == 0)

// Add verification methods in priority order
methods = new List<Auth.VerificationMethod>();
if (AvailableMethods.HasUserVerifiedMobileNumber)

    methods.add(Auth.VerificationMethod.SMS);
if (AvailableMethods.HasUserVerifiedEmailAddress)
    methods.add(Auth.VerificationMethod.EMAIL);
if (AvailableMethods.HasSalesforceAuthenticator)
    methods.add(Auth.VerificationMethod.
        SALESFORCE_AUTHENTICATOR);
if (AvailableMethods.HasTotp)
    methods.add(Auth.VerificationMethod.U2F);
if (AvailableMethods.HasU2F)
    methods.add(Auth.VerificationMethod.U2F);

return Site.passwordlessLogin(uid, methods, startURL);
}
}

```

[Passwordless Login Coding Considerations](#)

Review these tips and suggestions before implementing passwordless login.

SEE ALSO:

[Use the Discoverable Login Page for Simpler Login](#)

[Extend the Login Discovery Handler in Apex](#)

Passwordless Login Coding Considerations

Review these tips and suggestions before implementing passwordless login.

Check whether the user is verified

Users must be registered and verified before they can log in without a password. They must have a verified phone number to log in with an SMS one-time password. Or, they must have a verified email address to log in with an email one-time password.

You can determine identity verification from the user interface or API. See [View Users Identity Verification History](#) on page 47.

View the verification methods that the user has registered

Determine the user's verification methods with `TwoFactorMethodInfo`.

Handle errors returned by `system.site.passwordlessLogin`

If the `system.site.passwordlessLogin` method returns an error because the user can't be verified, handle the error. For example, redirect the user to an IdP, or use an alternate page to collect the password and invoke the `site.login` method.

Get more identity information after a user log in

Set up a login flow to kick off after the user logs in, for example, to register a phone number. Use the `System.UserManagement` `register` and `deregister` methods to manage user verification registration.

Make sure a user's email address and mobile phone number are unique

The `User` object stores the email address and phone number in `User.Mobilephone` and `User.Email`, respectively. But these fields aren't unique. To ensure uniqueness, store a copy of the email and phone in a unique field, such as `User.CommunityNickname`. Or create a custom field to store the values.

Get two verification methods for each user (for recovery)

We recommend that your users have two verification methods, for example, a phone number and password, or phone number and email address. You can collect this information when users sign up. Or you can collect the information on subsequent logins using a login flow.

Make sure that the phone number is in a format that Salesforce expects

Use the `System.UserManagement.formatPhoneNumber` method to ensure that the phone number is formatted correctly.

Determine how the login controller logs in the user

When implementing passwordless login, build the login controller to locate the user based on the identifier entered on the login page. Then check which verification methods the user has registered. Set up the controller to invoke the login process by whichever mechanism is configured, for example, passwordless login, SSO, or social sign-on.

You can completely remove passwords

Assign users the `Is Single Sign-On Enabled` user permission. This user permission is available when your org is enabled for delegated authentication—contact Salesforce to enable it. After you enable the feature, you can assign users the permission.

SEE ALSO:

[View Your Users' Identity Verification Methods](#)

EMBEDDED LOGIN: ALLOW VISITORS TO LOG IN TO YOUR WEB PAGES

Create authenticated sessions between your community and website visitors to extend your reach with your customers. Add login capabilities to any of your website pages with Salesforce Identity Embedded Login.

You can embed a small component into any app and use that component to manage your authentication and registration needs. Here are some ways to take advantage of Embedded Login.

- Do you want customers to log in before accessing your website? Add Embedded Login.
- Do you want customers to log in to your website before they purchase? Add Embedded Login to the page that contains your shopping cart.
- Do you want to update customers' billing address in their Salesforce contact when they purchase something on your site? Embedded Login integrates your website with your Salesforce back end.
- Do you want to collect information about your visitors to customize their experience? With Embedded Login, you can gather information about users during the login process. Get their email address, time zone, and even their profile picture.



When you configure your community with authentication providers, you've already done most of the work to enable Embedded Login. Website visitors log in using their credentials from any social or identity provider that you configured for your community. Web developers then choose which pages to add login capabilities to. Adding login capabilities consists of:

- Adding HTML meta tags to the login form
- Writing a JavaScript function to determine what happens when a user logs in

Another advantage of Embedded Login is that it provides a way to authenticate users through Salesforce when your website doesn't support authentication through SAML or OpenID Connect protocols.

[Embedded Login in Action](#)

Let's see what your website visitors experience when you add login capabilities to a web page with Embedded Login.

[How to Implement Embedded Login](#)

Adding Embedded Login to a web page takes some coordination between the Salesforce admin who manages the community and the web developer who owns the web page.

[Implement Embedded Login](#)

Let's add Embedded Login to a web page on your website. For your Salesforce community, use the external identity community that you created earlier in this guide. If you haven't created a community yet, you can complete the Identity for Customers Trailhead module to create your external identity community and earn a badge while you're at it.

[Embedded Login Advanced Authentication Features](#)

Embedded Login takes care of authenticating users so that you can add login capabilities to a web page without worrying about the details. You can also take advantage of the advanced authentication features that Salesforce offers.

[Embedded Login Considerations](#)

When implementing Embedded Login, be aware of these considerations.

[Embedded Login Meta Tag Reference](#)

You use these Embedded Login meta tags when adding login capabilities to your website.

Embedded Login in Action

Let's see what your website visitors experience when you add login capabilities to a web page with Embedded Login.

Imagine that you're the owner of Fix Coffee and you've designed a website for your customers to buy your products.

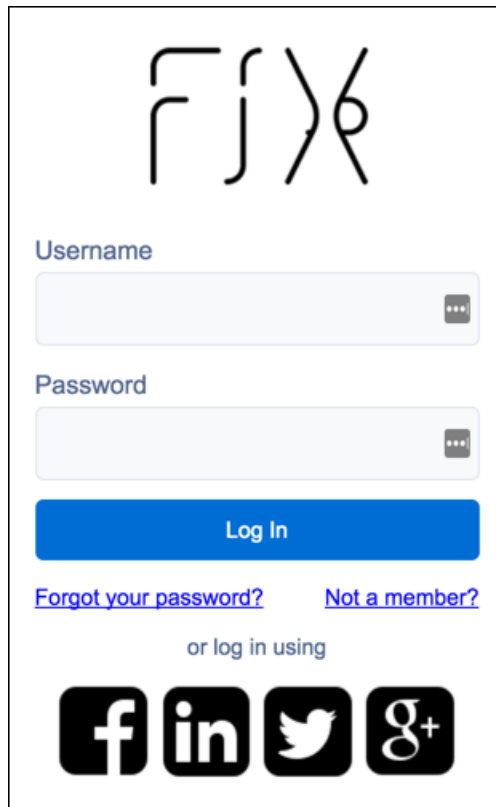
Login Button

When users browse your Fix website, they see a login button at the top of the web page when they try to make a purchase.



Login Page

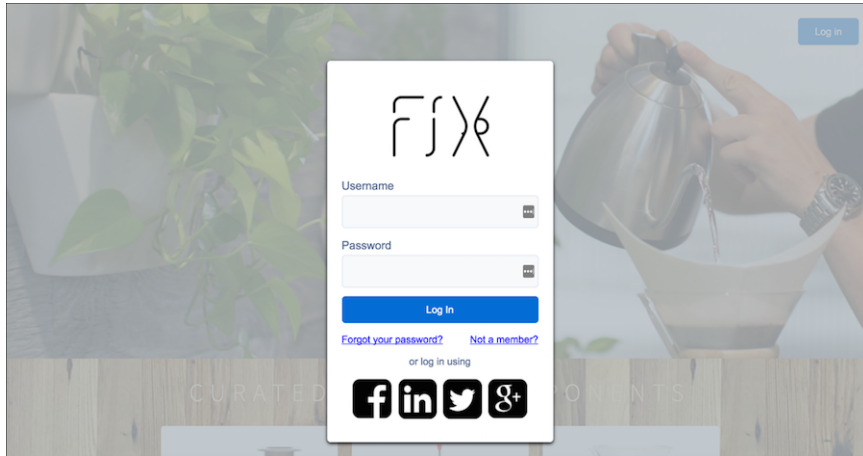
When users click the Login button, Embedded Login displays a login form. Your users can log in using a username and password, or they can sign in with their social account credentials.



Embedded Login Modes

You can use a modal, popup, or inline login form. If the form is a modal or popup, users click the Login button to see the login form. If the form is inline, users see the form when they navigate to the page. Here's a picture of the modal and inline modes. The modal option shows the login form in the foreground.

Here's what the web page looks like when the login form is in modal mode. The login form appears in the foreground at the center of the page.



Here's what the login form looks like in inline mode. The form appears when the user navigates to the web page. This form is just a sample—you can control how the login form looks.



onLogin Function

After your users log in, you control what happens. In our example, we wrote an `onLogin` function to display the user's avatar and email address on a successful login event.

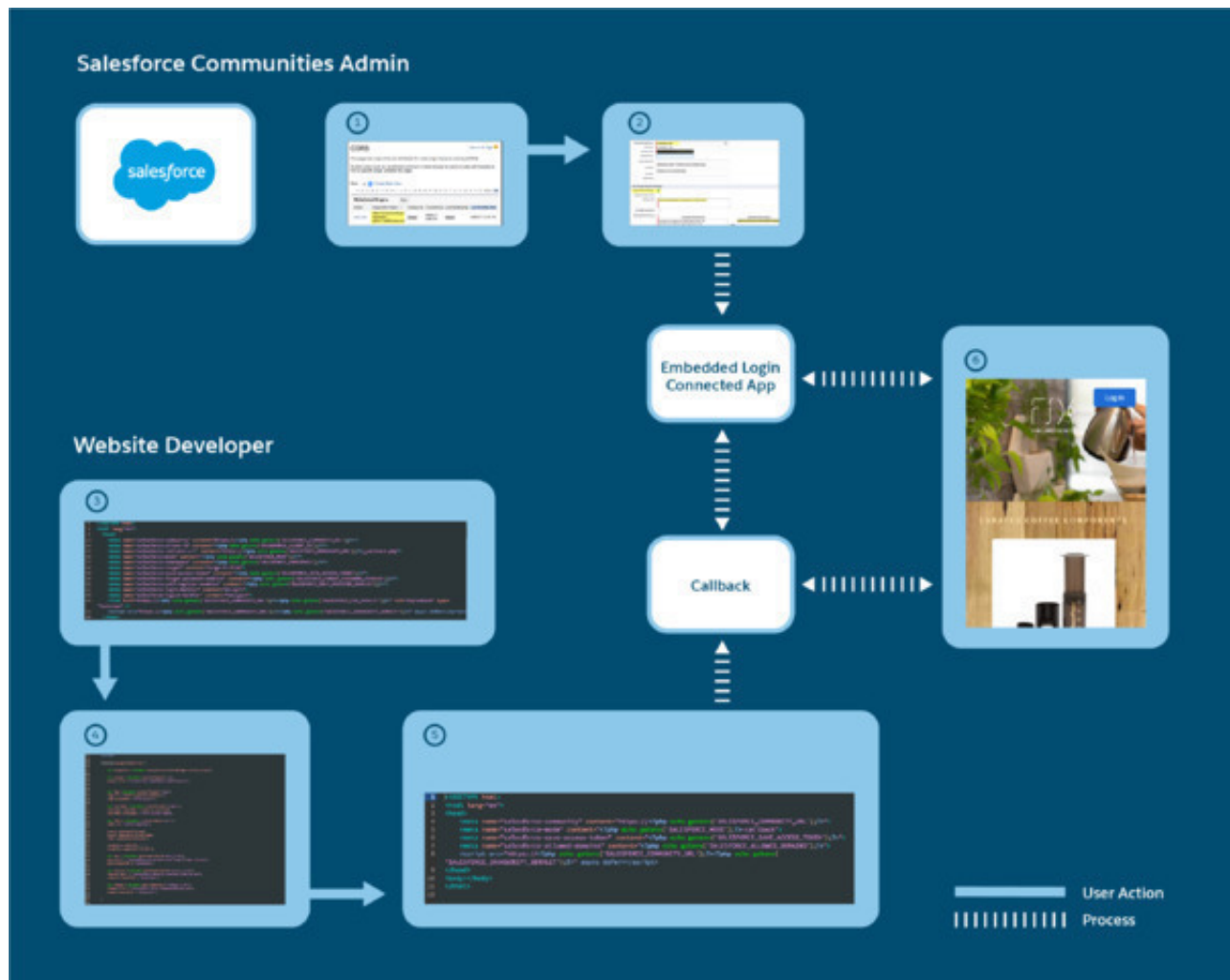


onLogout Function

You can control what your website users see when they log out. In our example, the `onLogout` function returns to the initial state, which displays the login button.

How to Implement Embedded Login

Adding Embedded Login to a web page takes some coordination between the Salesforce admin who manages the community and the web developer who owns the web page.



Who Does What

- The Salesforce admin creates a community, brands the login page, and configures the authentication providers. Then the admin adds the website domain to the Cross Origin Resource Sharing (CORS) whitelist (1).
- The Salesforce admin creates an Embedded Login connected app. The connected app handles the interaction between Salesforce and the website (2).
- The web developer adds Embedded Login meta tags to the web page to display the login form (3).

- The web developer supplies JavaScript functions and the `onLogin` and `onLogout` specifications to determine what happens when the user logs in and out. The logout function is optional. (4).
- The web developer creates a callback to handle the authentication response, specifying callback-specific meta tags (5).
- The result is a web page with login capabilities (6).

What's Happening Behind the Scenes

1. When a user clicks the button and enters credentials in the login form, Salesforce authenticates the user. Then Salesforce checks the connected app to determine the type of access token to grant.
2. Salesforce sends the access token to the callback.
3. The callback uses the access token to pull the user's information from Salesforce and cache it locally.
4. The `onLogin` function determines which information to display to the user.
5. If the website requires ongoing interaction with Salesforce after authentication, the connected app maintains a connection between the web page and the Salesforce community.

Implement Embedded Login

Let's add Embedded Login to a web page on your website. For your Salesforce community, use the external identity community that you created earlier in this guide. If you haven't created a community yet, you can complete the Identity for Customers Trailhead module to create your external identity community and earn a badge while you're at it.

Use your own website to add login capabilities to a web page. Or test it out by creating a simple web app on Heroku in a few minutes. Navigate to <https://heroku.com/apps>, and click **Sign Up for Free**.

Make sure that your community works before you add Embedded Login by testing various login scenarios. For example, can community users log in with their username and password? Can visitors join the community? Can users log in through all the social authentication providers that you set up?



Now you're ready to add login capabilities with Embedded Login.

[Step 1: Enable Resource Sharing Across Domains](#)

Embedded Login entails making web requests outside the website boundaries, but for security reasons, web requests are restricted to the current domain. To enable a website to share resources beyond its boundaries, the Salesforce admin adds trusted website domains to Salesforce's CORS (Cross-Origin-Resource-Sharing) whitelist. CORS is an industry standard that enables web browsers to make requests from origins other than their own.

[Step 2: Create the Embedded Login Connected App](#)

From your Salesforce community, create an Embedded Login connected app to connect your community with your website. The connected app handles communication between your web page and the Salesforce back end.

[Step 3: Enable Embedded Login on a Web Page](#)

On the web page where you want to add login capabilities, enter Embedded Login meta tags inside the `<head>` HTML element at the top of the page.

[Step 4: Write Login and Logout Functions](#)

On the web page, provide a login and optional logout JavaScript function in the body to handle `onLogin` and `onLogout` events. With these functions, you determine what happens when users log in and out successfully. You have full control over what happens.

[Step 5: Handle the Embedded Login Callback](#)

When a user logs in to a website, the Embedded Login callback receives the access token and uses it to retrieve user information. Both the access token and user information can be stored to local storage. Depending on your implementation, you can create a callback to handle the response on either the website (client) or the server.

SEE ALSO:

[Customize Your Login Pages with Community Builder](#)

[Dynamic URLs: Determine Your Brand at Run Time](#)

Step 1: Enable Resource Sharing Across Domains


Embedded Login entails making web requests outside the website boundaries, but for security reasons, web requests are restricted to the current domain. To enable a website to share resources beyond its boundaries, the Salesforce admin adds trusted website domains to Salesforce's CORS (Cross-Origin-Resource-Sharing) whitelist. CORS is an industry standard that enables web browsers to make requests from origins other than their own.

Salesforce CORS Whitelist and Access-Control-Allow-Origin

Embedded Login populates the Access-Control-Allow-Origin response header with the origin (domain plus protocol) specified in the CORS whitelist. If CORS isn't set up, the Access-Control-Allow-Origin header value is set to null, which effectively block all requests.

For Embedded Login to enable resource sharing across boundaries, the origin listed in the CORS whitelist—the Access-Control-Allow-Origin—must match the origin listed in the request. But an exact match isn't required. Here are some examples.

CORS	Request	Match?	Why?
<code>https://salesforce.com</code>	<code>https://salesforce.com:6109</code>	✓	Accept requests from all ports in this domain.
<code>https://salesforce.com:6109</code>	<code>https://salesforce.com</code>		Accept requests only from port 6109.
<code>https://*.salesforce.com</code>	<code>https://trailhead.salesforce.com</code>	✓	Accept requests from all subdomains in this domain.
<code>https://trailhead.salesforce.com</code>	<code>https://salesforce.com</code>		The CORS whitelist must list this domain.
<code>https://salesforce.com</code>	<code>https://salesforce.com/myCommunity</code>	✓	Accept requests from communities in this domain.
<code>https://salesforce.com/myCommunity</code>	<code>https://salesforce.com</code>		Accept requests only from this community.

CORS	Request	Match?	Why?
<code>https://salesforce.com/myCommunity</code>	<code>https://salesforce.com/myCommunity/login</code>		Accept requests from the login page of this community in this domain.
none	<code>https://salesforce.com/myCommunity</code>		The CORS whitelist must be set up to accept requests from other domains.

1. From Setup, enter `CORS` in the Quick Find box, then select **CORS**.

2. Click **New**.

3. Enter the domain where Embedded Login is deployed.

For example, `https://embeddedlogin.herokuapp.com` allows access to all pages hosted on `embeddedlogin.herokuapp.com`.

To handle multiple domains, you can use a regular expression to add them all to the whitelist at once. Or you can list each domain individually.



Note: By default, browsers cache the Embedded Login JavaScript, including your CORS settings, for 24 hours. You can change how often the cache refreshes with the `salesforce-cache-max-age` meta tag. If you change the value, test the change by clearing the cache between each change or using an incognito window.

SEE ALSO:

[Salesforce Help: Add Your Website to the CORS Whitelist](#)

Step 2: Create the Embedded Login Connected App

From your Salesforce community, create an Embedded Login connected app to connect your community with your website. The connected app handles communication between your web page and the Salesforce back end.

The connected app controls how the initial authentication process is handled. Then it continues to handle the interaction between the website and community during the user's active session. When creating the connected app, you supply the callback URL, which is used to retrieve the access token during the initial authentication process.

The Salesforce connected app and callback URL are interconnected, so you have a "chicken or egg" issue. The Embedded Login connected app needs the website's callback URL. The website needs the Embedded Login connected app URL. For now, specify a placeholder. You can come back later to replace it with the correct URL.

Use the Salesforce wizard to create a connected app for Embedded Login. It takes only a few minutes.

1. Start the connected app wizard.

- In Lightning Experience, from Setup, enter `App`, then select **App Manager**. Click **New Connected App**.
- In Classic, from Setup, enter `Apps`, then select **App**. Under Connected Apps, click **New**.

2. Complete these fields.

- App name, for example, Embedded Login
- Your email address

3. Click **Enable OAuth Settings**.

4. For the callback URL, enter `https://your_website/your_webpage/_callback.php`, where `_callback.php` is the name of your future callback.
5. For the OAuth scope, select **Allow access to your unique identifier (openid)**. You can add other options if your web page requires more access to Salesforce, but it isn't necessary.
6. Click **Save**.

The screenshot shows the 'Basic Information' and 'API (Enable OAuth Settings)' sections of the Salesforce Connected App configuration page. In the 'Basic Information' section, the 'Connected App Name' is 'Embedded Login', 'API Name' is 'Embedded_Login', and the 'Callback URL' is 'https://embeddedlogin.herokuapp.com/_callback.php'. In the 'API (Enable OAuth Settings)' section, 'Enable OAuth Settings' is checked, 'Enable for Device Flow' is unchecked, and 'Use digital signatures' is unchecked. The 'Selected OAuth Scopes' list contains 'Allow access to your unique identifier (openid)'. The 'Available OAuth Scopes' list includes various permissions like 'Access and manage your Chatter data', 'Access and manage your Eclair data', 'Access and manage your Wave data', 'Access and manage your data (api)', 'Access custom permissions (custom_permissions)', 'Access your basic information (id, profile, email, address, phone)', 'Full access (full)', 'Perform requests on your behalf at any time (refresh_token, offline_access)', 'Provide access to custom applications (visualforce)', and 'Provide access to your data via the Web (web)'.

It can take a few minutes for the changes to take effect.

7. Click **Continue**.
The new connected app opens and populates the consumer key, which is the app's unique identifier to identify itself to Salesforce.
8. Copy the consumer key for later. You use it when entering meta tags. It's the value for the meta tag `salesforce_client_id`.
9. Click **Manage**, and then click **Edit Policies**.
10. Under OAuth Policies, select **Admin approved users are pre-authorized**.

The screenshot shows the 'Connected App Edit' page. The 'Basic Information' section is visible, showing the 'Start URL' and 'Mobile Start URL'. The 'OAuth policies' section is expanded, showing 'Permitted Users' set to 'Admin approved users are pre-authorized', 'IP Relaxation' set to 'Enforce IP restrictions', and 'Refresh Token Policy' set to 'Immediately expire refresh token'.

11. Click **Yes**.

12. Click **Save**.
13. Under Profiles, click **Manage Profiles** and select the profiles that can access this connected app. Choose the profile you created when you set up your community.
14. Optionally, you can get more user information by adding custom attributes to the connected app.
15. Click **Save**.

[Retrieve User Information with Custom Attributes](#)

As part of the login process, Embedded Login retrieves information from Salesforce about the user who's logging in. You determine what kind of information to collect by creating custom attributes for your Embedded Login connected app.

SEE ALSO:

[Create a Connected App for Your Web App](#)

[Salesforce Help: Connected Apps](#)

[Salesforce Help: Connected App and OAuth Terminology](#)

Retrieve User Information with Custom Attributes

As part of the login process, Embedded Login retrieves information from Salesforce about the user who's logging in. You determine what kind of information to collect by creating custom attributes for your Embedded Login connected app.

You can create custom attributes for your Embedded Login connected app either declaratively or programmatically.

- Declaratively: Go to the Setup page for your Embedded Login connected app. Under Custom Attributes, choose the user information that you want to collect.
- Programmatically: Use the `customAttributes` method of the Apex class `Auth_ConnectedAppPlugin`. For more information, see [ConnectedAppPlugin Class](#).

You can also write JavaScript and use the returned access token to call other Salesforce APIs. However, this option is more difficult to implement and less efficient. For the user information that you can retrieve, see [Identity URLs](#).

SEE ALSO:

[Personalize Your App with Custom Attributes](#)

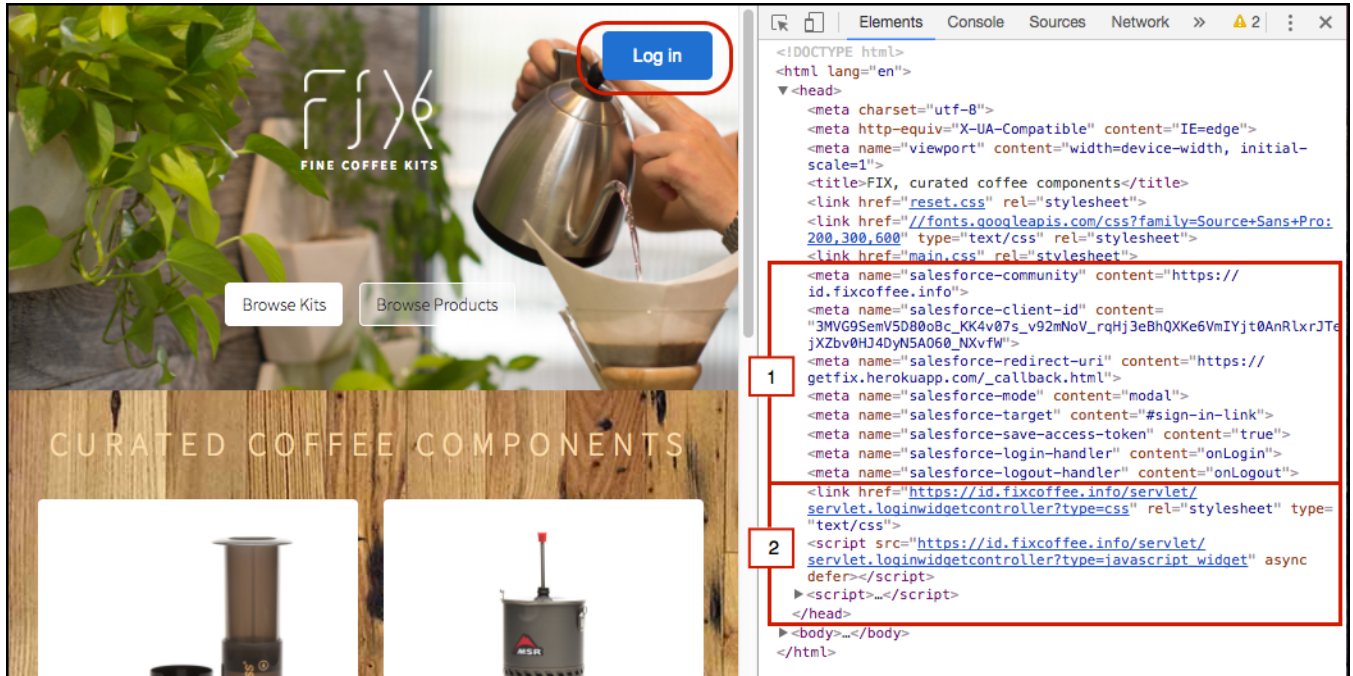
[Salesforce Help: UserInfo Endpoint](#)

[Step 4: Write Login and Logout Functions](#)

Step 3: Enable Embedded Login on a Web Page

On the web page where you want to add login capabilities, enter Embedded Login meta tags inside the `<head>` HTML element at the top of the page.

Here's what you see when you inspect the web page.



The top (1) contains a set of meta tags that specify how to display the login form. The bottom (2) contains a link to your CSS resources and a script to invoke Embedded Login on the server.

Let's take a closer look at the code.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>FIX, curated coffee components</title>

    <link href="reset.css" rel="stylesheet">
    <link href="//fonts.googleapis.com/css?family=Source+Sans+Pro:200,300,600"
    type="text/css" rel="stylesheet">
    <link href="main.css" rel="stylesheet">

    <meta name="salesforce-community" content="https://id.fixcoffee.info">
    <meta name="salesforce-client-id"
    content="3MVG9SemV5D80o8c_KK4v07s_v92mNoV_rqHj3eBhQXKe6VmIYjt0AnRlXrJTjXZbv0HJ4
    DyN5AO60_NXvfw">
    <meta name="salesforce-redirect-uri" content="https://getfix.herokuapp.com/_
    callback.html">
    <meta name="salesforce-mode" content="modal">
    <meta name="salesforce-target" content="#sign-in-link">
    <meta name="salesforce-save-access-token" content="true">
    <meta name="salesforce-login-handler" content="onLogin">
    <meta name="salesforce-logout-handler" content="onLogout">
    <link href="https://id.fixcoffee.info/servlet/servlet.loginwidgetcontroller?
    type=css" rel="stylesheet" type="text/css" />
    <script src="https://id.fixcoffee.info/servlet/
    servlet.loginwidgetcontroller?type=javascript_widget" async defer></script>
  </head>
```

The CSS and servlet URLs reside on static endpoints hosted by your community. You replace `https://embeddedlogin-developer-edition.na99.force.com/demo/` with the path to your community.

```
<link
href="https://embeddedlogin-developer-edition.na99.force.com/demo/servlet/servlet.loginwidgetcontroller?type=css"
rel="stylesheet" type="text/css" />
<script
src="https://embeddedlogin-developer-edition.na99.force.com/demo/servlet/servlet.loginwidgetcontroller?type=javascript_widget"
async defer></script>
```


Next, let's see how to implement Embedded Login by populating the web page `<head>` element.

[Add Embedded Login Meta Tags to Your Web Page](#)

You enter meta tags on your web page where you want to add login capabilities with Embedded Login. You specify the Salesforce community URL, what the login form looks like, the location of the CSS style sheet, and more.

[Replace the Embedded Login CSS with Your Own](#)

Embedded Login comes with a default style sheet, which styles the login button and login form. If you want to replace the default CSS with your own, you have a few options.

[Change How to Invoke Embedded Login on the Server](#)

You can change the default behavior of Embedded Login on the server by modifying the `<script>` tag inside the `<head>` HTML element at the top of the web page.

Add Embedded Login Meta Tags to Your Web Page

You enter meta tags on your web page where you want to add login capabilities with Embedded Login. You specify the Salesforce community URL, what the login form looks like, the location of the CSS style sheet, and more.

On the web page where you want to add to login capabilities, add Embedded Login information within the `<head>` HTML element.

1. Open the web page to which you want to add login capabilities.
2. Within the `<head>` HTML element at the top of the file, enter these required meta tags, specifying the values from your configuration.
 - `salesforce-community`
 - `salesforce-client-id`
 - `salesforce-redirect-uri`
 - `salesforce-mode`
 - `salesforce-target`
 - `salesforce-login-handler`
 - `salesforce-logout-handler`
 - `salesforce-server-callback` (required if you're using a server-side callback)
3. If desired, enter these optional meta tags.
 - `salesforce-forgot-password-enabled`
 - `salesforce-self-register-enabled`
 - `salesforce-mask-redirects`
 - `salesforce-use-min-js`
 - `salesforce-cache-max-age`
 - `salesforce-save-access-token`
4. Enter the link to the location of CSS resources. The CSS resides on a static endpoint hosted by your community.

```
<link
href="https://embeddedlogin-developer-edition.na99.force.com/demo/servlet/servlet.loginwidgetcontroller?type=css"
rel="stylesheet" type="text/css" />
```
5. Add this Embedded Login script, replacing the community URL `https://embeddedlogin-developer-edition.na99.force.com/demo` with your own.

```
<script  
src="https://embeddedlogin-developer-edition.na99.force.com/demo/servlet/servlet.loginwidgetcontroller?type=javascript_widget"  
async defer></script>
```

SEE ALSO:

- [Replace the Embedded Login CSS with Your Own](#)
- [Change How to Invoke Embedded Login on the Server](#)
- [Embedded Login Meta Tag Reference](#)
- [Create an Embedded Login Client-Side Callback](#)

Replace the Embedded Login CSS with Your Own

Embedded Login comes with a default style sheet, which styles the login button and login form. If you want to replace the default CSS with your own, you have a few options.

Use whichever method you prefer.

- Override the CSS directly in the web page.
- Edit the Embedded Login CSS. The style sheet is located in your community URL resource folder.
- Replace the style sheet with your own by updating the link on your web page with the location of your style sheet.



Note: If you replace the style sheet, be sure to define all the necessary styling. Embedded Login doesn't provide default styles when you supply your own CSS.

Change How to Invoke Embedded Login on the Server

You can change the default behavior of Embedded Login on the server by modifying the `<script>` tag inside the `<head>` HTML element at the top of the web page.

The script loads a Java servlet that enables login capabilities. You can add these parameters to the script to change how the Embedded Login servlet behaves.

min=false

Generates a readable JavaScript version. By default, Embedded Login loads the JavaScript in a minimized, lightweight state that's hard to read. Use `min=false` to generate a response that's easier to read.

```
<script src="https://embeddedlogin-developer-edition.na99.force.com/demo/servlet/  
servlet.loginwidgetcontroller?type=javascript_widget&min=false" async defer></script>
```

cacheMaxAge=n

Sets the cache control maximum age header, which the browser uses to determine whether the cached content is fresh or must be refreshed from the server. By default, the cache is cleared every 24 hours. You can change the maximum age, where *n* is the specified number of seconds. To improve performance, increase the cache age. However, as a result, you receive JavaScript updates from Salesforce less frequently because the updates occur when the cache is cleared. Here, the cache refreshes every three days.

```
<script src="https://embeddedlogin-developer-edition.na99.force.com/demo/servlet/  
servlet.loginwidgetcontroller?type=javascript_widget&cacheMaxAge=259200" async  
defer></script>
```

Embedded Login supports localization. For example, it localizes the login form based on browser settings. All labels, errors, and actions match the browser's language setting. See [Embedded Login Advanced Authentication Features](#).

SEE ALSO:

[Embedded Login Advanced Authentication Features](#)

Step 4: Write Login and Logout Functions

On the web page, provide a login and optional logout JavaScript function in the body to handle `onLogin` and `onLogout` events. With these functions, you determine what happens when users log in and out successfully. You have full control over what happens.

1. Add a login function to the body of the web page.

Here's a code example that replaces the login button with user's name and profile picture (avatar) upon successful login. The Embedded Login response returns the user information.

```
function onLogin(identity) {  
  
    var targetDiv = document.querySelector(SFIDWidget.config.target);  
  
    var avatar = document.createElement('a');  
    avatar.href = "javascript:showIdentityOverlay()";  
  
    var img = document.createElement('img');  
    img.src = identity.photos.thumbnail;  
    img.className = "sfid-avatar";  
  
    var username = document.createElement('span');  
    username.innerHTML = identity.username;  
    username.className = "sfid-avatar-name";  
  
    var iddiv = document.createElement('div');  
    iddiv.id = "sfid-identity";  
  
    avatar.appendChild(img);  
    avatar.appendChild(username);  
    iddiv.appendChild(avatar);  
  
    targetDiv.innerHTML = '';  
    targetDiv.appendChild(iddiv);  
  
}
```

2. Optionally, add code to the login function to retrieve user information from Salesforce.
3. Optionally, add a logout function to the body of the web page.

You can write a function to determine what happens when a user logs out. This code example clears the user's Salesforce session and reloads the login button.

```
function onLogout() {  
    SFIDWidget.init();  
}
```

```
}  

```

SEE ALSO:

[Retrieve User Information with Custom Attributes](#)

Step 5: Handle the Embedded Login Callback

When a user logs in to a website, the Embedded Login callback receives the access token and uses it to retrieve user information. Both the access token and user information can be stored to local storage. Depending on your implementation, you can create a callback to handle the response on either the website (client) or the server.

Way back in Step 2, when you created a connected app, you supplied the URL to this callback. The callback is mainly for security, and it's used to receive the access token. It then uses the token to fetch user information from Salesforce and write the information to the local storage, which is on the community domain. After writing user information to local storage—which is equivalent to a successful login, the callback redirects the user back to the index page.

The main difference between a client-side and server-side callback is how the access token is received. For a normal client-side callback that's a web page, the callback receives the access token. For a server-side callback, the server gives the callback a one-time code. Then the callback exchanges the code for an access token.

For server-side callbacks, after using the access token to get the user information, the server writes the information out to the served file, such as an HTML file. After the user information is read on the client, it is written to local storage, which is on the community domain.

For more information about the login authentication process and relationship between the callback and connected app, see [Embedded Login Authentication Features](#).

[Create an Embedded Login Client-Side Callback](#)

To create a client-side callback, you add a web page to your website and specify a few Embedded Login meta tags inside the `<head>` HTML element.

[Create an Embedded Login Server-Side Callback](#)

To create a server-side callback, create a servlet using your preferred language. Use the server-side callback instead of a client-side callback web page to avoid exposing the access token on the client.

Create an Embedded Login Client-Side Callback


To create a client-side callback, you add a web page to your website and specify a few Embedded Login meta tags inside the `<head>` HTML element.

The client-side callback takes the access token from Salesforce and writes it to local browser storage for future access. Regardless of how many web pages that you add login capabilities to, you create only one callback page.



Note: For security, the callback page must be on the same domain as the web pages containing Embedded Login.

1. Create a page on your website and call it `_callback`, for example, `_callback.php`.
2. Enter the following required meta tags inside the `<head>` HTML element of this `_callback` page.
 - `salesforce-community`
 - `salesforce-allowed-domains`
 - `salesforce-mode` (where the value ends in `-callback`)

 **Note:** The value of the `salesforce-mode` meta tag is the same mode specified in the Embedded Login web page with the `-callback` suffix. For example, if `salesforce-mode` on the web page is set to `modal`, the value is `modal-callback`.

3. If desired, enter these optional meta tags.

- `salesforce-save-access-token` with the value `true` to save the access token after initialization. By saving the access token, you can continue to interact with Salesforce during the active user session.
- `salesforce-logout-on-browser-close`

4. Keep the body empty: `<body></body>`.

Example:

```
<html>
<head>
  <meta name="salesforce-community"
content="https://embeddedlogin-developer-edition.na99.force.com/demo">
  <meta name="salesforce-client-id"
content="3MVG9Iu66FKeHhIPrRneLTDFdiuLfgLjycFpg6SbLpZAJScEXuD.oRdaWnJE7QGFWHxunp0ut1">

  <meta name="salesforce-mode" content="inline-callback">
  <meta name="salesforce-save-access-token" content="false">
  <meta name="salesforce-allowed-domains" content="embeddedlogin.heroku.com">
  <meta name="salesforce-redirect-uri"
content="https://embeddedlogin.heroku.com/_callback.html">
  <meta name="salesforce-target" content="#salesforce-login">
  <meta name="salesforce-login-handler" content="onLogin">
  <meta name="salesforce-logout-handler" content="onLogout">

  <script src="https://embeddedlogin-developer-edition.na99.force.com/demo/servlet/

      servlet.loginwidgetcontroller?type=javascript_widget" async defer></script>
</head>
<body></body>
</html>
```

SEE ALSO:


[Embedded Login Meta Tag Reference](#)

[Add Embedded Login Meta Tags to Your Web Page](#)

Create an Embedded Login Server-Side Callback

To create a server-side callback, create a servlet using your preferred language. Use the server-side callback instead of a client-side callback web page to avoid exposing the access token on the client.

When you use the server-side callback, you create a separate servlet to authenticate the user, retrieve user's identity information, and form the HTTP response.

 **Note:** You can use the server-side callback to get a refresh token with the Scope OAuth setting, **Perform requests on your behalf at any time** (refresh token, `offline_access`). Set this scope from Enable OAuth Settings when creating the connected app.

1. Execute an OAuth web-server flow to authenticate the user logging in.

Issue an HTTP POST against the community's token endpoint where the `grant_type` parameter must be `authorization_code`. The server process received the authorization code in an HTTP request and now the process must include the code in this POST. If the HTTP POST completes properly, the user who's logging in is authenticated and Salesforce returns the access token in the JSON body.

2. Parse the results of the OAuth web-server flow.

Use the access token to get identity information about the now authenticated user. If you added customer attributes when you created the Embedded Login connected app (Step 2: Create the Embedded Login Connected App), the custom attributes are included in the JSON body.

3. Form an HTML response.

The response must contain these meta tags.

- `salesforce-community`
- `salesforce-mode` (where the value ends in `-callback`)



Note: The value of the `salesforce-mode` meta tag is the same mode specified in the Embedded Login web page with the `-callback` suffix. For example, if `salesforce-mode` on the web page is set to `modal`, the value is `modal-callback`.

- `salesforce-server-callback` (where the value must be `true`)
- `salesforce-server-response`
- `salesforce-server-starturl`
- `salesforce-target`
- `salesforce-allowed-domains`

You can include the `salesforce-save-access-token` with the value `true` to save the access token after initialization. By saving the access token, you can continue to interact with Salesforce during the active user session.

4. In your Embedded Login web page, specify these meta tags.

- a. Add the `salesforce-server-callback` meta tag with the value `true`. This meta tag indicates that the callback to handle the HTTP response is on the server.

```
<meta name="salesforce-server-callback" content="true">
```

- b. Make sure that the `salesforce-redirect-uri` meta tag references the location of the server-side callback servlet. Use the same URL as specified in the callback URL field of your Embedded Login connected app.

```
<meta name="salesforce-redirect-uri"
content="https://embeddedlogin.heroku.com/servlet/servlet.serversidecallback">
```

- c. Make sure that the `salesforce-mode` on this web page matches the mode on the server-side callback.



Example:



Note: This server callback servlet uses base64 encoding in the server response.

```
package servlet;

import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.methods.GetMethod;
import org.apache.commons.httpclient.methods.PostMethod;

import org.json.JSONObject;

import javax.servlet.ServletConfig;
```

```

import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.URLDecoder;
import java.nio.charset.StandardCharsets;
import java.util.Base64;

@WebServlet(
    name = "CallbackServlet2",
    urlPatterns = {"/_callback"}
)
public class ServerSideCallbacks extends HttpServlet{

    // Client ID
    private static final String CLIENT_ID=
"3MVG9xOCXq4IDluF8V6oKd32SPVi6FHwEOQlQ5BjvaKX.5QZpGe4Z3F4fc6KvMYsQ.fi3l4cp0oZ8KpOBs4Mh";

    // client secret
    private static final String CLIENT_SECRET = "9103416584217247123";

    @Override
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws
        ServletException, IOException {

        String code = request.getParameter("code");
        if (code != null) {
            code = URLDecoder.decode(code, "UTF-8");
        }
        String startURL = request.getParameter("state");
        if (startURL != null) {
            startURL = URLDecoder.decode(startURL, "UTF-8");
        }

        String tokenResponse = null;
        String communityUrl = null;
        HttpClient httpClient = new HttpClient();
        try {
            // community_url parameter passed from redirect uri.
            communityUrl = request.getParameter("sfdc_community_url");

```

```

        // Token endpoint : communityUrl + "/services/oauth2/token";
        PostMethod post = new PostMethod(communityUrl+"/services/oauth2/token");
        post.addParameter("code",code);
        post.addParameter("grant_type","authorization_code");
        // Consumer key of the Connected App.
        post.addParameter("client_id", CLIENT_ID);
        // Consumer Secret of the Connected App.
        post.addParameter("client_secret",CLIENT_SECRET);

        // Callback URL of the Connected App.
        post.addParameter("redirect_uri",
            "https://boiling-brushlands-41143.herokuapp.com/_callback");

        httpClient.executeMethod(post);
        tokenResponse = post.getResponseBodyAsString();
        post.releaseConnection();

        System.err.println("tokenResponse: " + tokenResponse);
    } catch (Exception e) {
        throw new ServletException(e);
    }
}

JSONObject identityJSON = null;
try {
    JSONObject token = new JSONObject(tokenResponse);
    // get the access token from the response
    String accessToken = token.getString("access_token");
    String identity = token.getString("id");
    httpClient = new HttpClient();
    GetMethod get = new GetMethod(identity + "?version=latest");
    get.setFollowRedirects(true);
    get.setRequestHeader("Authorization", "Bearer " + accessToken);

    // get identity information using the access token
    httpClient.executeMethod(get);
    String identityResponse = get.getResponseBodyAsString();
    get.releaseConnection();
    identityJSON = new JSONObject(identityResponse);
    identityJSON.put("access_token", accessToken);
} catch (Exception e) {
    throw new ServletException(e);
}

response.setContentType("text/html; charset=utf-8");
PrintWriter out = response.getWriter();

// Notice that we're using base64 encoded
String outputStr = "<html><head>\n" +
    "<meta name=\"salesforce-community\" content=\"" + communityUrl + "\">\n" +
    // notice the -callback in the salesforce-mode content value
    "<meta name=\"salesforce-mode\" content=\"modal-callback\">\n" +
    "<meta name=\"salesforce-server-callback\" content=\"true\">\n" +

```



```

        // send the identity information back to the Embedded Login
        "<meta name=\"salesforce-server-response\" content=\"" +
        Base64.getEncoder().encodeToString(identityJSON.toString()).
        getBytes(StandardCharsets.UTF_8))+"'\>\n" +
        "<meta name=\"salesforce-server-starturl\" content=\"" + startURL + "'>\n"
+
        "<meta name=\"salesforce-target\" content= \"#salesforce-login\">\n"+
        "<meta name=\"salesforce-allowed-domains\"
content=\"boiling-brushlands-41143.herokuapp.com\">\n" +
        "<script src=\""+ communityUrl +
        "/servlet/servlet.loginwidgetcontroller?type=javascript_widget\" +
        " async defer></script>\n" +
        "</head><body></body></html>";
        out.write(outputStr);
    }
}

```

SEE ALSO:[Step 2: Create the Embedded Login Connected App](#)[Embedded Login Meta Tag Reference](#)[Embedded Login Meta Tag Reference](#)[REST API Developer Guide: Understanding the Web Server OAuth Authentication Flow](#)[REST API Developer Guide: Finding Additional Resources](#)[Salesforce Help: Authenticate Apps with OAuth](#)

Embedded Login Advanced Authentication Features

Embedded Login takes care of authenticating users so that you can add login capabilities to a web page without worrying about the details. You can also take advantage of the advanced authentication features that Salesforce offers.

Multi-Language Support in Embedded Login

Set the locale code parameter dynamically in the Embedded Login script. This example displays the embedded login page in Japanese.

```

<script src="https://embeddedlogin-developer-edition.na99.force.com/demo/servlet/
servlet.loginwidgetcontroller?type=javascript_widget&locale=jp" async defer></script>

```

Use the following Salesforce-defined (nonstandard) locale codes with the locale parameter.

- Chinese (Simplified): `cn`
- Chinese (Traditional): `tw`
- Dutch (Netherlands): `nl`
- English (Australian): `au`
- English (Canada): `ca`

- English (India): `in`
- English (Ireland): `eu`
- English (Singapore): `ap`
- English (US): `us`
- English (UK): `uk`
- Finnish: `fi`
- French (France): `fr`
- French (Canada): `fr-ca`
- German: `de`
- Italian: `it`
- Japanese: `jp`
- Korean: `kr`
- Portuguese (Brazil): `br`
- Spanish (Spain): `es`
- Spanish (Mexico): `mx`
- Swedish: `sv`
- Thai: `th`

Embedded Login Connected App on Your App Launcher

As an option, you can add your Embedded Login-enabled web page to your App Launcher to provide transparent single sign-on. To do so, add a Start URL to the connected app that you want displayed. The Start URL uses a special format.

The start URL format is the community URL's OAuth Authorize endpoint plus these parameters:

- Token response type
- Client ID of the Embedded Login's connected app
- Encoded URL to the callback
- State

Here's an example of the start URL where the community OAuth endpoint is

`https://embeddedlogin-developer-edition.na99.force.com/demo/services/oauth2/authorize`
and the callback to Embedded Login is `https://embeddedlogin.herokuapp.com/_callback`.

```
https://embeddedlogin-developer-edition.na99.force.com/demo/services/oauth2/authorize?response_type=token&
client_id=3MVG9Iu66FKeHhIPrRneLTDFdiuLfgLjycFpg6SbLpZAJScEXuD.oRdaWnJE7QGNEFWHxunp0ut1&
redirect_uri=https%3A%2F%2Fembeddedlogin.herokuapp.com%2F_callback.html&state=%2F
```



Note: The Start URL field is limited to 255 characters. If your site's URL exceeds this limit, you can create a Visualforce page as a relay.

Embedded Login Authentication Process

It's helpful to be familiar with the Salesforce authentication process. Here's an overview.

1. The user enters a username and password or social credentials in the login form on the web page.

2. Salesforce validates the credentials and then redirects the response to the OpenID authorization endpoint `https://login.salesforce.com/services/oauth2/authorize`. The connected app ID of the Embedded Login connected app is passed in, which specifies how the OAuth access token is granted. Salesforce issues the access token based on how the connected app is configured.
3. Salesforce sends the access token to the callback.
4. The callback receives the access token, parses it out as a message, and caches the access token. If it's a client-side callback, the token is cached in the web browser local storage.
5. The callback uses the access token to call the Salesforce Identity service endpoint to pull the required and authorized information about the user.
6. The callback stores the user information with the access token.
7. Embedded Login gets the user information from storage and checks the online function to determine which login information to show on the web page and how.
8. If the website doesn't require ongoing interaction with Salesforce after initial login, the access token can be released. If the web page continues to interact with Salesforce, the access token remains in storage. The connected app maintains the connection between the page and Salesforce and uses the access token to retrieve data from Salesforce.

The authentication process happens in an iframe, and Salesforce sets the `salesforce-mask-redirects` meta tag to true to hide the process from the user. However, if your org uses login flows or two-factor authentication, you set the `salesforce-mask-redirects` meta tag to false. The user takes the journey to Salesforce to complete the login process.

Relationship Between the Embedded Login Callback and Connected App

The Embedded Login connected app is at the core of the authentication process and controls ongoing communication between the website and Salesforce after initial authentication. The callback is involved during the initial authentication process, receiving the access token and the user information passed from Salesforce. When the initial authentication is complete, the connected app takes over. It maintains the connection with your Salesforce community as long as the session is active.

SEE ALSO:

[Salesforce Help: Authenticate Apps with OAuth](#)

[Identity Implementation Guide: Configure and Use the App Launcher](#)

Embedded Login Considerations

When implementing Embedded Login, be aware of these considerations.

Supported Browsers

Embedded Login is supported on all browsers that support Lightning Experience. Embedded Login works on these browsers: Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge, and Internet Explorer 11.

HTTP and HTTPS URLs

Modern browsers don't allow mixing the HTTP and HTTPS protocols on the same page. Your right-frame content is inside an iframe, so the URL must use the same protocol as your community. Generally, we recommend using HTTPS for logo and right-frame URLs. However, in local and test environments, HTTP can be more flexible.

URL Redirection for Self-Registration and Forgot Password Links

Embedded Login handles logging in with username and password or social credentials, but relies on the community to handle self-registration and forgotten passwords. When customers complete the password or self-registration page, they must navigate back to your website. They're not redirected to the website automatically.

OAuth Policies


OAuth policies control how the Embedded Login connected app handles authentication. Website visitors are prompted to approve access when the default OAuth policy allows users to self-authorize. To avoid this interruption, set the OAuth policy to **Admin approved users are pre-authorized**, as mentioned in Step 2: Create the Embedded Login Connected App.

Error Configuring CORS

If you get an error that contains the phrase ... `an ancestor value violates the following Content Security Policy directive`, configure CORS according to instructions in Step 1: Enable Resource Sharing Across Domains.

HTML5 localStorage for Storing Data


Embedded Login stores data locally within the user's browser using `localStorage`, which is more secure than storing data in cookies. `localStorage` can store large amounts of data locally without affecting website performance. `localStorage` is per origin (per domain and protocol). All pages from one origin can store and access the same data. Data in `localStorage` has no expiration date and is cleared only through JavaScript or by clearing the browser cache or locally stored data.

 **Note:** If you get "access denied" messages, check whether `localStorage` is available.

Missing Login Button Due to Limited Access to Cookies

If a user sets the browser to block third-party websites from storing cookies and other data locally, the Login button doesn't appear on the web page. For example, if the Chrome **Block third-party cookies** option is set, the Login button doesn't appear.

While this behavior applies to all browsers, it is a problem for Safari users because of Safari's default settings. The default setting is to block cookies and local storage if the user hasn't yet visited the website, based on current cookies and browsing history. With Embedded Login, content is served from a domain other than the one that the user is on, so it's more likely that the user hasn't visited the website. To enable Embedded Login, the user must change **Allow from websites I visit** to **Always Allow**.

 **Note:** Salesforce issues a session cookie only to record encrypted authentication information during a specific session. The session cookie doesn't include the username and password of the user. Salesforce doesn't use cookies to store other confidential user and session information. Instead Salesforce implements more advanced security methods based on dynamic data and encoded session IDs.

SEE ALSO:

[Add a Self-Registration Page](#)

[Salesforce Help: Recommendations and Requirements for All Browsers](#)

[Salesforce Help: Supported Browsers for Lightning Experience](#)

Embedded Login Meta Tag Reference

You use these Embedded Login meta tags when adding login capabilities to your website.

salesforce-allowed-domains

Specifies domains that can access the access token and user information. Use this meta tag only on callback. The callback must be located in the same domain as the Embedded Login web page. String.

```
<meta name="salesforce-allowed-domains" content="embeddedlogin.herokuapp.com">
```

salesforce-cache-max-age

Sets the cache control maximum age header. Optional. The browser uses this header to determine whether the cached content is fresh or must be refreshed from the server after the specified number of seconds. Integer. By default, the cache is cleared every 24 hours. To improve performance, increase the cache maximum age. However, as a result, you receive JavaScript updates from Salesforce less frequently because updates occur when the cache is cleared. In this example, the cache is cleared every minute.

```
<meta name="salesforce-cache-max-age" content="60">
```

salesforce-client-id

The Embedded Login connected app's consumer key, which is the unique identifier for the connected app. When the Salesforce admin creates the Embedded Login connected app, the app generates a unique identifier in the consumer key field. String.

```
<meta name="salesforce-client-id"
content="3MVG9Iu66FKeHhIPrRneLTDFdiuLfgLjycFpg6SbLpZAJScEXuD.oRdaWnJE7QGFWHxunp0ut1">
```

salesforce-community

Community URL of the Salesforce community hosting your identity services. String.

```
<meta name="salesforce-community"
content="https://embeddedlogin-developer-edition.na99.force.com/demo">
```

salesforce-expid

Specifies the value of the experience ID for the current user session. Use this tag to support dynamic branding on your login page. String.

```
<meta name="salesforce-expid" content="coffee">
```

salesforce-forgot-password-enabled

Indicates whether to show the forgot password link on the login form. Optional. Boolean, `false` by default. If your community enabled the forgot password link, set this link to `true`.

```
<meta name="salesforce-forgot-password-enabled" content="true">
```

salesforce-login-handler

JavaScript function to call on a successful login event. Optional. You have full control over what happens when users log in successfully. For example, you can replace the login button with the user's name and profile picture. String.

```
<meta name="salesforce-login-handler" content="onlogin">
```

salesforce-logout-handler

JavaScript function to call on a successful logout event. Optional. You have full control over what happens when users log out successfully. For example, you can clear the user's session and reload the login button. String.

```
<meta name="salesforce-logout-handler" content="onlogout">
```

salesforce-logout-on-browser-close

Deletes the user's Embedded Login session after they close their browser window. Optional. Boolean, `true` by default. Set to `false` if you want users to remain logged in even after they close the browser.

```
<meta name="salesforce-logout-on-browser-close" content="true">
```

salesforce-mask-redirects

Controls the login process. By default, Embedded Login uses a simple authentication process that's completed inside an iframe and hidden from the user with a mask redirect. Boolean, `true` by default.

Set to `false` to invoke a custom login flow. Your org can use a custom login flow to add more steps to the login process. For example, the flow can prompt the user for a second factor of authentication, collect or update user data, or display a custom logo or message. If your org uses a custom login flow, set `salesforce-mask-redirects` to `false` and work with your Salesforce admin to integrate the login flow with Embedded Login.

```
<meta name="salesforce-mask-redirects" content="true">
```

salesforce-mode

Add this meta tag to the Embedded Login web page to determine whether to display the login form inline or as a modal or popup. With modal and popup modes, the page initially displays a login button. When clicked, the login form appears. With inline mode, the login form appears when the user navigates to the web page. Modal and inline modes render the login form from the website. Popup mode loads your community's login page.

```
<meta name="salesforce-mode" content="inline">
```

Add this meta tag to the server-side callback to determine how the callback displays the login form on the web page. Values can be `modal-callback`, `inline-callback`, or `popup-callback`. This value must match the mode specified on the web page. For example, if your web page mode is `modal`, the callback value must be `modal-callback`.

```
<meta name="salesforce-mode" content="inline-callback">
```

salesforce-redirect-uri

URL of your callback. This URL is the same as the URL that you specify in the `callback URL` field of the Embedded Login connected app. The connected app requires the callback URL to connect Salesforce to your website. String.

```
<meta name="salesforce-redirect-uri"
content="https://embeddedlogin.heroku.com/_callback.php">
```

salesforce-save-access-token

Indicates whether to save the user's access token after the initial login process. Boolean, `false` by default, which doesn't save the access token. Set to `true` to continue interacting with Salesforce during the active user session.

```
<meta name="salesforce-save-access-token" content="true">
```

salesforce-self-register-enabled

Indicates whether to show the self-register link on the login form. Boolean, `false` by default. If your community enabled the self-registration link, set this link to `true`.

```
<meta name="salesforce-register-enabled" content="true">
```

salesforce-server-callback

Indicates that the Embedded Login callback is on the server. Boolean, `false` by default. If you're using a server-side callback, this value must be set to `true`.

```
<meta name="salesforce-server-callback" content="true">
```

salesforce-server-response

The HTML response of the server-side callback. String. It returns a base-64 encoded response from the user info endpoint.

```
<meta name="salesforce-server-response" content=%json%
```

salesforce-server-starturl

The Embedded Login connected app start URL. Specify this meta tag to add your Embedded Login-enabled web page to your App Launcher. String. It returns the state parameter.

```
<meta name="salesforce-server-starturl"
content=https://embeddedlogin-developer-edition.na99.force.com/demo/services/oauth2/authorize?response_type=token&
client_id=3MVG9Iu66FKeHhIPrRneLTDfduLfgLjycFpg6SbIpZAJScEXuD.oRdaWnJE7QGNFWHxunp0ut1&
redirect_uri=https%3A%2F%2Fembeddedlogin.herokuapp.com%2F_callback.html&state=%2F
```

salesforce-target

Identifier of the visible HTML element, such as a button or link, which executes a JavaScript function when clicked. For example, to use a login button as a target, `#salesforce-login` refers to a `<div>` in the body of the web page, `div id=salesforce-login`. String.

```
<meta name="salesforce-target" content="#salesforce-login">
```

salesforce-use-login-page-background-color

Determines the background color of the Embedded Login login form. Boolean, `true` by default. If `true`, it uses the background color specified on the Community Workspaces Administration | Login & Registration page. If `false`, the color specified in the local CSS is used.

```
<meta name="salesforce-use-login-page-background-color" content="false">
```

salesforce-use-login-page-button-color

Determines the color of the login button on the Embedded Login login form. Boolean, `true` by default. If `true`, it uses the button color specified on the Community Workspaces Administration | Login & Registration page. If `false`, the color specified in the local CSS is used.

```
<meta name="salesforce-use-login-button-color" content="false">
```

salesforce-use-min-js

Indicates whether to generate JavaScript in a readable or minimized lightweight version. Boolean, `true` by default. Set to `false` to generate readable JavaScript.

```
<meta name="salesforce-use-min-js" content="false">
```

SEE ALSO:

[Create an Embedded Login Client-Side Callback](#)

MANAGE COMMUNITIES FOR EXTERNAL IDENTITY

Salesforce for Customers and Partners has features for managing communities for users with External Identity licenses.

[Extend External Identity Sessions \(Beta\)](#)

Make it easy for your external identity customers and partners to stay in your community with longer sessions and fewer logins. Allow users to remain logged in even after they close their browser. And keep them logged in for up to seven days of inactivity. To extend sessions, modify the use profile for the external identity users in your community. You can also add more security when external users log in by enabling device activation.

[Create Lightweight Users Contactless Users \(Beta\)](#)


Reduce the overhead of managing external identity users by creating users without contact information. You can add contacts later if you decide that you want them—like when you upgrade to a more full-featured community license. This feature is available only for users with the External Identity license. It's not available with other community licenses.

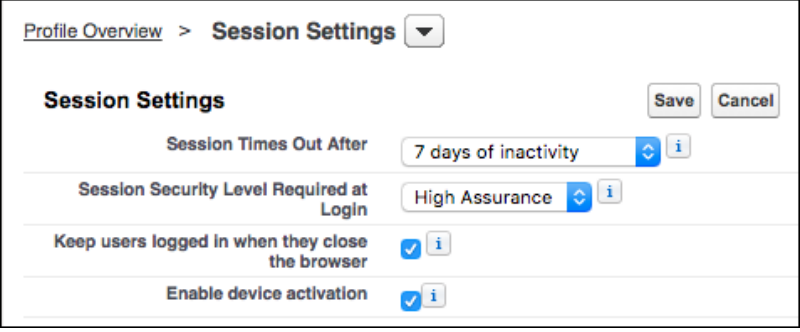
[Upgrade Users Without Contacts to a Community License \(Beta\)](#)

While you can create external identity users without contacts, you must add a contact to the user before you can upgrade the user to a community license. Upgrading an external identity to a community user's license lets you give users access to more data within your community.

Extend External Identity Sessions (Beta)

Make it easy for your external identity customers and partners to stay in your community with longer sessions and fewer logins. Allow users to remain logged in even after they close their browser. And keep them logged in for up to seven days of inactivity. To extend sessions, modify the use profile for the external identity users in your community. You can also add more security when external users log in by enabling device activation.

-  **Note:** This release contains a beta version of Extended Sessions for External Identity Users, which means it's a high-quality feature with known limitations. Extended Sessions for External Identity Users isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. We can't guarantee general availability within any particular time frame or at all. Make your purchase decisions only based on generally available products and features.



[Profile Overview](#) > **Session Settings** ▼

Session Settings Save Cancel

Session Times Out After	7 days of inactivity ⬇ i
Session Security Level Required at Login	High Assurance ⬇ i
Keep users logged in when they close the browser	<input checked="" type="checkbox"/> i
Enable device activation	<input checked="" type="checkbox"/> i

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS


To edit session settings in profiles:

- Manage Profiles and Permission Sets

1. From Setup, enter *Profiles* in the Quick Find box, then select **Profiles**.
2. Next to the external identity profile that you want to use as the basis for the new profile, click **Clone**.
3. Name the profile, and click **Save**.
4. Click **Session Settings**.
5. Under Session Settings, for Session Times Out After, select a timeout value.
Extend the session length to make it easy for your customers and partners to stay in your community. You can extend session timeouts for external identity users for up to seven days. When the session times out, the user must log in again.
6. To keep external identity users logged in until they log out, select **Keep users logged in when they close the browser**.
This setting lets external identity user sessions remain active until users log out of the community or when the session times out. If not selected, external identity users are logged out when they close their browser.
7. To add more security when external users log in, select **Enable device activation**.
When selected, Salesforce requires external users to verify their identity when they log in from a different browser or device. This option applies to all external user licenses—External Identity and community licenses.
8. Click **Save**.

Create Lightweight Users Contactless Users (Beta)

Reduce the overhead of managing external identity users by creating users without contact information. You can add contacts later if you decide that you want them—like when you upgrade to a more full-featured community license. This feature is available only for users with the External Identity license. It's not available with other community licenses.

 **Note:** This release contains a beta version of Contactless External Identity Users, which means it's a high-quality feature with known limitations. Contactless External Identity Users isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. We can't guarantee general availability within any particular time frame or at all. Make your purchase decisions only based on generally available products and features.

Typically, when Salesforce creates an external identity user in a community, it adds the user's contact information. But if your implementation doesn't require contact information for external users, consider going contactless. Without contacts, you don't have to worry about keeping user and contact records in sync. You can simply maintain a user database for identity purposes. For example, you can delay creating contacts for the guests registering for your community until after they're qualified.

You can add contactless users to a new or existing community.

To create a contactless external identity user, you create a user record using Apex, SOAP, or Bulk API.

```
User u = new User();
    u.FirstName = 'Jane';
    u.LastName = 'Doe';
    u.Email = 'janedoe@test.com';
    u.Alias = 'jane';
    u.Username = 'janedoe@test.com';
    u.CommunityNickname = 'Jane';
    u.LocaleSidKey = 'en_US';
```

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To create contactless users:

- Manage User OR Manage External User OR Edit Self-Service User

```

u.TimeZoneSidKey = 'GMT';
u.ProfileID = '00exx000000jvN4'; // Profile that's associated with the EI license
u.LanguageLocaleKey = 'en_US';
u.EmailEncodingKey = 'UTF-8';
insert u;

```

To handle self-registration for contactless users, create a custom self-registration page and create contactless users through the API. Then enable self-registration for your community by selecting **Allow external users to self-register** from the community's Login & Registration page.

To add a contact to the contactless external identity user, create a contact (1), and associate the contact with the user (2).

```

Account a = [SELECT Id FROM Account WHERE Id = '001xx000003DIyf'];
Contact c = new Contact(); // Create contact
c.FirstName = 'James';
c.LastName = 'James';
c.AccountId = a.id;
insert c;
User u = [SELECT Id FROM User WHERE Id = '005xx000001TL1f'];
u.ContactId=c.id // Associate contact with user
update u;

```

Considerations

- The Login As feature isn't supported for contactless users because contacts are required.
- Delegated admins can't manage contactless users.
- System for Cross-Domain Identity Management (SCIM) isn't supported for contactless user creation.
- External identity users without contacts have the same access to objects as users with contacts.

Upgrade Users Without Contacts to a Community License (Beta)

While you can create external identity users without contacts, you must add a contact to the user before you can upgrade the user to a community license. Upgrading an external identity to a community user's license lets you give users access to more data within your community.

- 📌 **Note:** This release contains a beta version of Contactless External Identity Users, which means it's a high-quality feature with known limitations. Contactless External Identity Users isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. We can't guarantee general availability within any particular time frame or at all. Make your purchase decisions only based on generally available products and features.
- 📌 **Note:** The Contactless User feature is available only for the External Identity license, which is available to customers of the Identity for Customers and Partners product.

EDITIONS

Available in: Salesforce Classic and Lightning Experience

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

USER PERMISSIONS

To add a contact to a contactless user:

- Manage User OR Manage External User OR Edit Self-Service User

1. To add a contact to a contactless external identity user, update the user's record with Apex, SOAP, or Bulk API.

```
Account a = [SELECT Id FROM Account WHERE Id = '001xx000003DIyf'];
Contact c = new Contact();
c.FirstName = 'Sonia';
c.LastName = 'James';
c.AccountId = a.id;
insert c;
User u = [SELECT Id FROM User WHERE Id = '005xx000001TL1f'];
u.ContactId=c.id;
update u;
```

2. Upgrade to a community license.
 - a. From Setup, enter *Users* in the Quick Find box, then select **Users**.
 - b. Click **Edit** next to the user you want to upgrade.
 - c. Select the new license, profile, and role for the user.
 - d. Optionally, specify a new profile and role.
 - e. Save your changes.



Note: You can upgrade contactless external identity users to other community licenses, but you can't downgrade community users to contactless users.

EXTERNAL IDENTITY ON GITHUB AND TRAILBLAZER COMMUNITY

We've covered the basics of setting up Salesforce Identity for external users. What you've learned in this guide provides the basis for customizing Salesforce Identity to your specific business goals.

However, you can do much more with Salesforce Identity. For more information on customizing Salesforce Identity for your business, check out our advanced samples on the Salesforce Identity GitHub account.

If you don't see an answer to your question or a solution to your problem, post to the Salesforce Identity group in the Trailblazer Community. The Salesforce Identity Team loves to hear from customers.

INDEX

<head> [73](#)
<script> [73](#), [76](#)
\$Profile [36](#)

A

access token [78–79](#)
account [2](#), [12](#), [90–92](#)
activate community [13](#)
Admin approved users are pre-authorized [71](#), [85](#)
Admin Trusted Mobile Phone [47](#)
All users may self-authorize [85](#)
Allow access to your unique identifier (openID) [71](#)
Aloha template [13](#)
App Launcher [83](#)
App manager [71](#)
Auth. provider [30](#)
Auth. providers [29](#)
authentication [4](#)
authentication process [68](#), [78](#), [83](#)
Authenticator App [47](#)

B

background image [16](#)
base64 encoding [79](#)
Brand [68](#)
browser storage [78](#)

C

cache age [76](#)
callback [79](#)
callback handler [78](#)
checklist [8](#)
clear session [77](#)
Client ID [75](#)
clone external identity profile [11](#)
cloud directory services [3](#)
code example [73](#), [77–79](#), [83](#)
communities [6](#)
Communities
 Apex [60](#)
 branding [55](#)
 custom login page [60](#)
 login [55](#)
 passwordless login [60](#)
 self-registration [55](#)
Community Builder [19](#), [38](#), [43](#)

Community Workspaces menu [13](#)
configurable self-registration
 Login & Registration page [44](#)
configurable self-registration handler [46](#)
configurable self-registration page [44](#)
connected app [31](#), [34](#)
ConnectedAppPlugin class [73](#)
Consumer key [75](#)
CORS [68](#), [70](#), [85](#)
create developer org [11](#)
CRM [2](#), [4](#), [12](#), [90–92](#)
Cross Origin Resource Sharing (CORS) [70](#)
CSS stylesheet [73](#), [76](#)
custom attributes [36](#), [73](#)
custom error exception
 LoginDiscovery handler [48](#)
custom fields [73](#)
custom login page [12](#), [37](#)

D

dashboards [3](#)
Default Page [38](#), [43](#)
definition [4](#)
delegated authentication [28](#)
developer org [11](#)
domain name [11](#)
domain URLs [70](#)
dynamic [55](#), [60](#)

E

Embedded Login [4](#)
encoding [79](#)
Entity ID [35](#)
extend sessions [90](#)
External Identity license [5](#)
external identity profile [15](#)

F

Facebook [28](#), [30](#), [35](#)
federated authentication [28](#)
field sets [22](#)
forceios utility [32](#)
forgot password [16](#)

G

github [57](#)

GitHub [29](#), [94](#)

H

Heroku [34](#)

I

IAM services [1](#), [7](#)

Identity Basics [7](#)

Identity Connect [4](#)

Identity only license [4](#)

Identity providers [64](#)

Identity URLs [73](#)

Identity Verification [47](#)

iframe [16](#)

implementation [8](#)

J

JavaScript [76](#)

JIT [28](#), [30](#)

just-in-time provisioning [3](#), [28](#), [30](#)

L

license [5](#)

licenses [4](#)

login discovery handler [41](#)

login discovery page [39](#)

login page [19](#), [57](#)

login page type [38](#)

logo [16](#), [34](#)

M

mask redirects [75](#)

membership to community [15](#)

Metadata Discovery Endpoint [35](#)

minimize JavaScript [76](#)

mobile identity [3](#), [31](#)

multi-factor authentication [3](#)

My Domain [11](#)

N

Not a member link [21](#), [85](#)

O

OAuth [31](#)

OAuth policies [71](#), [85](#)

One-Time Password App [47](#)

onLogin [77](#)

onLogout [77](#)

OpenID Connect [64](#)

P

password on login page [24](#)

passwordless login

about [37](#)

considerations [62](#)

Login & Registration page [39](#)

verify users [47](#), [62](#)

permission sets [15](#)

person accounts [25](#)

profiles [4](#), [11–12](#), [15](#), [26](#), [35–36](#)

R

readable JavaScript [76](#)

register domain [11](#)

registration handler [29](#)

resources [76](#)

roles [11](#)

S

Salesforce Authenticator [47](#)

SAML [34–35](#), [64](#)

SAML-based connected app [34](#)

sandboxes [11](#), [24](#)

Security Policy directive [85](#)

self-registration [2](#), [4](#), [21](#), [25–26](#), [90–92](#)

self-registration page type [43](#)

server-side callback [79](#)

servlet updates [76](#)

session timeout [90](#)

SFDCOAuthLoginHost [32](#)

single sign-on [3](#), [29](#)

Social sign-on providers [64](#)

start URL [35](#)

Start URL [83](#)

T

Temporary Code [47](#)

title field [22](#)

Trailblazer Community [94](#)

Trailhead [7](#)

U

use cases [4](#), [6](#), [25](#), [28](#)

V

verification methods [47](#), [62](#)

Verified Mobile Phone [47](#)

Visualforce [57](#)

Visualforce Page [38](#), [43](#)

Index

W

website [4](#)
whitelist [70](#)

workflows [3–4](#)

X

XCode [32](#)