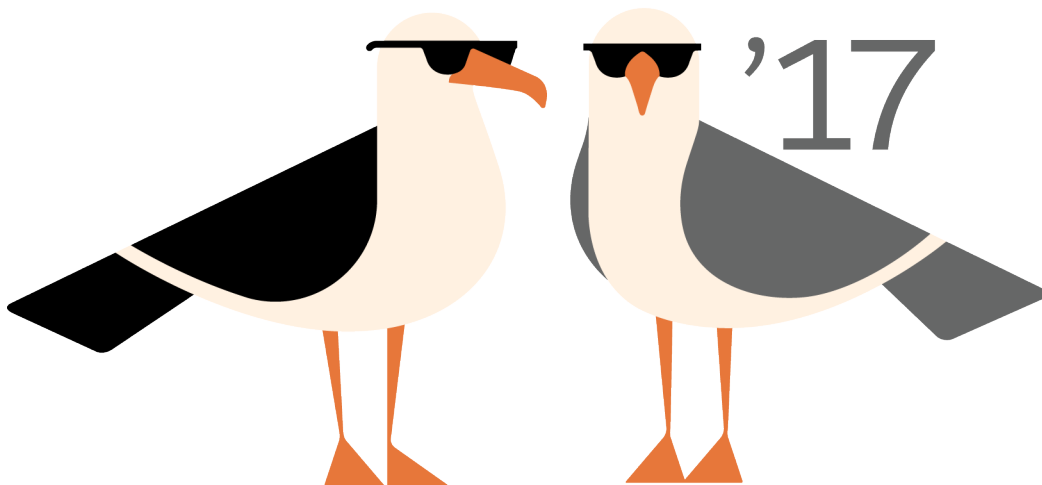




---

# Salesforce External Identity Implementation Guide

Salesforce, Summer '17





# CONTENTS

|  |    |
|--|----|
| <b>SALESFORCE IDENTITY FOR EXTERNAL USERS</b> .....                              | 1  |
| <b>WHAT IS SALESFORCE IDENTITY FOR EXTERNAL USERS?</b> .....                     | 2  |
| <b>HOW CAN I USE SALESFORCE IDENTITY FOR EXTERNAL USERS?</b> .....               | 3  |
| Acquire and Engage New Users .....   | 3  |
| Deliver a Consistent Experience with a Single Identity Across All Channels ..... | 4  |
| Secure and Manage Your Customer and Partner Ecosystems .....                     | 4  |
| Integrate and Customize to Your Business Needs .....                             | 4  |
| Extend External Identity to Your Website .....                                   | 5  |
| <b>EXTERNAL IDENTITY LICENSES</b> .....  | 6  |
| <b>EXTERNAL IDENTITY AND COMMUNITIES</b> .....                                   | 7  |
| <b>PREPARE YOUR ORG</b> .....  | 8  |
| Create a Developer Org .....   | 8  |
| Set Up a My Domain .....   | 8  |
| Control Authorization with Custom Profiles and Roles .....                       | 9  |
| Create an Account for CRM Integration .....                                      | 10 |
| <b>CREATE A BRANDED LOGIN PAGE</b> .....   | 11 |
| Set Up Your Community .....  | 11 |
| Customize Your Community .....   | 13 |
| Create a Branded Login Experience .....  | 14 |
| Activate Your Community .....  | 15 |
| <b>ENABLE SELF-REGISTRATION</b> .....  | 17 |
| Create a Basic Branded Self-Registration Page .....                              | 17 |
| Add Fields to Collect Additional Information .....                               | 18 |
| Add a Password Field to Enable Login Directly During Registration .....          | 19 |
| <b>ENABLE SELF-REGISTRATION FOR B2C USERS (OPTIONAL)</b> .....                   | 21 |
| Enable Person Accounts .....   | 21 |
| Configure Self-Registration for Person Accounts .....                            | 22 |
| <b>ACCEPT IDENTITY FROM AN EXISTING IDENTITY PROVIDER</b> .....                  | 23 |
| Social Sign-On .....   | 23 |
| Create an Auth. Provider .....   | 24 |

- Customize Your Registration Handler . . . . . 24
- Enable Your Auth. Provider in Your Community . . . . . 24
  
- ACCEPT USER IDENTITY WITH SAML AND JUST-IN-TIME PROVISIONING . . . . . 26**
  
- SET UP SSO AND ACCESS FOR YOUR WEB APP . . . . . 27**
  - Create a Connected App for Your Web App . . . . . 27
  - Create a Sample Service Provider on Heroku . . . . . 28
  - Configure Salesforce Identity to Provide Identity for Your App . . . . . 28
  - Authorize Your Web Application . . . . . 28
  - Configure Your App to Trust Salesforce Identity . . . . . 29
  - Personalize Your App with Custom Attributes . . . . . 29
  - More About Single Sign-On for Your Web App . . . . . 30
  
- PROVIDE 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
  
- EMBEDDED LOGIN: AUTHENTICATE YOUR WEBSITE VISITORS . . . . . 34**
  - Embedded Login in Action . . . . . 35
  - How to Implement Embedded Login . . . . . 38
  - Steps to Add Authentication to Your Website with Embedded Login . . . . . 39
    - Step 1: Enable Resource Sharing Across Domains . . . . . 40
    - Step 2: Create the Embedded Login Connected App . . . . . 40
    - Step 3: Enable Embedded Login on a Web Page . . . . . 43
    - Step 4: Write Login and Logout Functions . . . . . 46
    - Step 5: Handle the Embedded Login Callback . . . . . 47
  - Embedded Login Advanced Authentication Features . . . . . 52
  - Embedded Login Considerations . . . . . 54
  - Embedded Login Meta Tag Reference . . . . . 55
  
- EXTERNAL IDENTITY ON GITHUB AND SUCCESS COMMUNITY . . . . . 58**
  
- INDEX . . . . . 59**

# SALESFORCE IDENTITY FOR EXTERNAL USERS

This implementation guide describes how to set up Salesforce Identity for external users. Refer to this guide when you want to set up external identity on your own Salesforce org.

But first, to get a thorough understanding of external identity, there's nothing better than learning by doing. So take advantage of these Trailhead modules.

## **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 (IdP), and service providers (SPs). Get familiar with the identity protocols, Security Assertion Markup Language (SAML), 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. The steps in this guide are similar to the steps you follow in the Trailhead module.

# WHAT IS SALESFORCE IDENTITY FOR EXTERNAL USERS?

Salesforce Identity is an Identity and Access Management (IAM) service that connects users to your applications, 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 applications 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 converges 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.

# HOW CAN I USE SALESFORCE IDENTITY FOR EXTERNAL USERS?

Salesforce Identity for external users 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.

## 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 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.

### 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.

# 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.

# 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 applications and services, including identity services.

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

Salesforce External Identity uses communities for its deployment. Community deployment accommodates for a unique brand and configuration, so you can act as both a service provider and identity provider for all your applications without your customers realizing that the service runs on Salesforce. Similar to My Domain, communities allow for unique DNS domains, either in the format of `https://customname.force.com` or custom SSL domains.

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

# PREPARE YOUR ORG

To begin your walkthrough of a typical external identity implementation, you create a developer org. Because Salesforce 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.

For a refresher on the entire process for creating an external identity community, watch the [How to Get Started Basic Developer Edition Org for Identity](#) video. Then follow the tasks in this section.

## [Create a Developer Org](#)

Developer orgs have all the features and licenses you need to get started with Salesforce Identity. You can use existing orgs, trial orgs, and sandboxes for external identity, but Developer orgs are a great way to get started.

## [Set Up a My Domain](#)

My Domain allows admins to define their own DNS subdomain name and associate it with their Salesforce org. A subdomain name gives you more control over the authentication options for your company and also highlights your company name.

## [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

---

Developer orgs have all the features and licenses you need to get started with Salesforce Identity. You can use existing orgs, trial orgs, and sandboxes for external identity, but Developer orgs are a great way to get started.

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

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

## Set Up a My Domain

---

My Domain allows admins to define their own DNS subdomain name and associate it with their Salesforce org. A subdomain name gives you more control over the authentication options for your company and also highlights your company name.

The Salesforce SAML identity provider requires that the org be configured with a subdomain. Fortunately, setting up a subdomain is easy with 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*. Then the company's login URL is `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 it's done, 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. On the My Domain page, click **Deploy to Users** and you're done!

For more instructions on how you can use My Domain for internal identity use cases, see the [Salesforce Identity Implementation Guide](#).

## 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.

Profiles define how users access objects and data and what they can do in Salesforce and with your connected apps. You can use a default profile, but it's a best practice to clone and customize the default profiles to meet your own organizational requirements.

1. In your developer org, from Setup, enter *Profiles* in the Quick Find box, then select **Profiles**.
  - a. Click **Clone** next to **External Identity User**.
  - 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.
  - 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 user and click **Edit**.
    - c. From the Role picklist, select **Customer Manager**.

- d. Click **Save**.

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

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

1. In your developer org, switch to the Sales application.
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

Salesforce Identity for external users makes it simple to deploy a custom identity experience. This custom experience is important. You want your users to experience your brand consistently, whether they're visiting your website for the first time or signing in to your community. The first thing you set up is a community to support your deployment and configure a branded login page.

To learn how, watch the [How to Set Up a Community for Identity and Deploy a Branded Login Page](#) video. Then walk through the steps in this section.

## [Set Up Your Community](#)

To enable Salesforce Communities for your org, you provide a community domain name, much like the domain name you created when setting up your org. The community domain collects all your communities under one URL. Typically, your community domain name is your company name.

## [Customize Your Community](#)

Let's go through the basics of configuring your community for External Identity use cases. First, use profiles to lock down your data so that external users see only what you want them to see. Then use the Community Manager to customize your users' login experience. This process is where you add your brand to your external identity community.

## [Create a Branded Login Experience](#)

While that page is nice, it doesn't feel like it's optimized for a customer-facing brand, does it? Let's start using Community Builder to see how quickly we can customize the user's branding experience.

## [Activate Your Community](#)

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

## Set Up Your Community

---

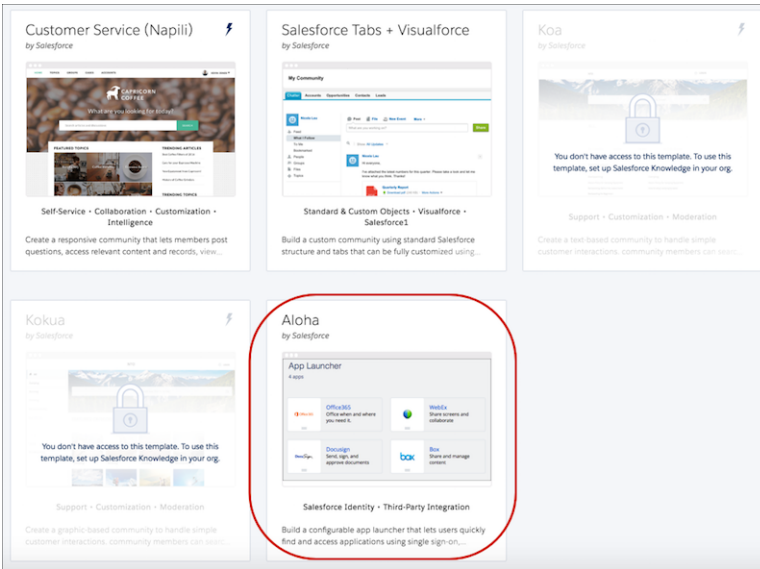
To enable Salesforce Communities for your org, you provide a community domain name, much like the domain name you created when setting up your org. The community domain collects all your communities under one URL. Typically, your community domain name is your company name.

To get started, create a community to host your external identities.

1. In your developer org, 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. Also, after you choose this name, you can't change it. However, for complete control over branding later, you can add a [custom SSL domain](#).
4. Select **Check Availability**.
5. Click **Save**, and then click **OK**.

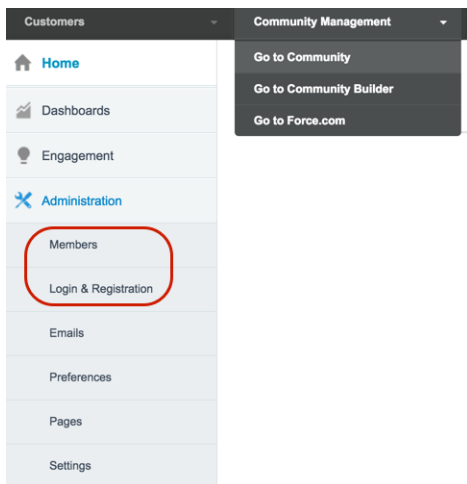
The basic Communities feature is enabled. Next, let's create your external identity community.

6. Click **New Community**.  
You're shown a series of community templates.



7. Choose the Aloha template. 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. Let's call it Customers.
9. Click **Create Community**.

Your new community is now available! Click **Manage & Moderate** to open Community Management. You manage your community and create its login page from here.



SEE ALSO:

[Steps to Add Authentication to Your Website with Embedded Login](#)



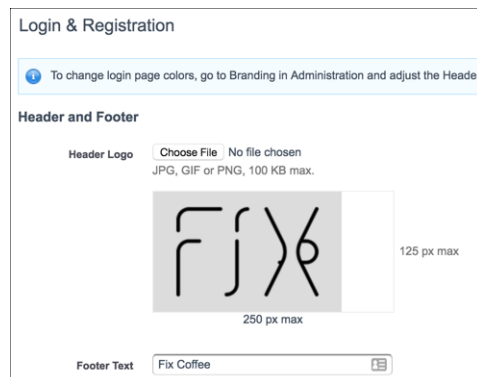
## Customize Your Community

Let's go through the basics of configuring your community for External Identity use cases. First, use profiles to lock down your data so that external users see only what you want them to see. Then use the Community Manager to customize your users' login experience. This process is where you add your brand to your external identity community.

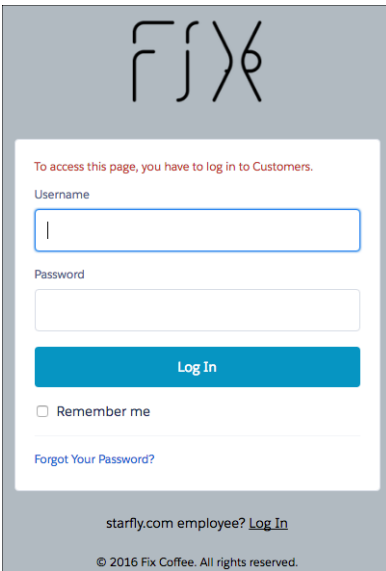
You can use your own images for branding or use ours. Download and unzip this file, which contains a sample logo and backgrounds: <https://www.salesforceidentity.info/ExternalIdentityAssets.zip>.

1. Authorize external identity users access to your community.
  - a. From Community Management, select **Administration**, then select **Members**.  
You can now control what people can access when they log in to your community.
  - b. In the search list, enter *Portal*.
  - c. In the list of Available Profiles, locate the **Customers** profile and then click **Add** to add it to Selected Profiles.
  - d. Click **Save**.  
Salesforce updates membership for your community and sends an email when it's done. Now External Identity users with the Customers profile are authorized to self-register and use single sign-on to access your community.
2. Customize your login page.
  - a. From the Community Manager, select **Login & Registration**.
  - b. Next to Header Logo, click **Choose File**.
  - c. Select a logo. For example, select `fix-logo.png` from the sample files you downloaded.
  - d. Brand the footer text. Let's call it Fix Coffee.
  - e. Click **Save**.

If you used `fix-logo.png` for your logo, the Login & Registration page looks like this.



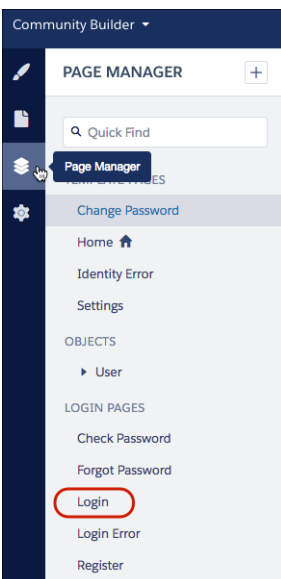
3. Under Login, select **Allow internal users to log in directly to the community**.
4. Under Password, notice that the Forgot Password and Change Password fields are populated.
5. Let's see how your changes appear on a login page.
  - a. Copy the URL of your community—it's in the browser window address bar.
  - b. Open a new browser or incognito window and paste the URL in the address bar.  
Your page looks something like this.



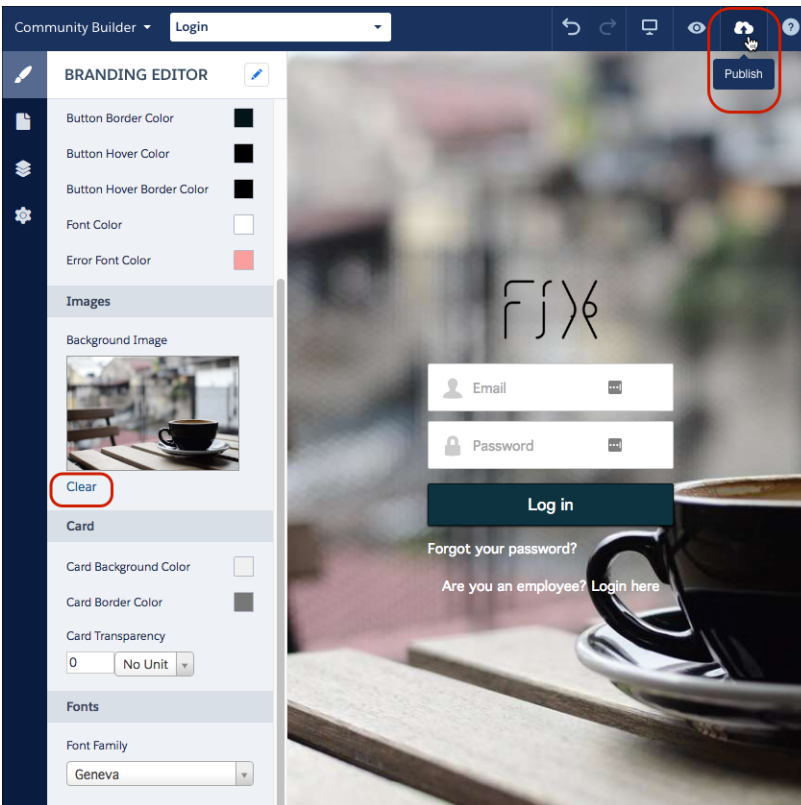
## Create a Branded Login Experience

While that page is nice, it doesn't feel like it's optimized for a customer-facing brand, does it? Let's start using Community Builder to see how quickly we can customize the user's branding experience.

1. From Community Management, click your name in the top-right and then click **Setup**.
2. From Setup, enter *All Communities* in the **Quick Find** box, then select **All Communities** and click the **Manage** link next to your community.
3. From Community Management, click **Go to Community Builder**.
4. Under Page Manager, select **Login**, click **Edit**, then the **Branding Editor** icon (🎨). You can now customize your login page brand.



- 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.
- Tweak your button colors and other options, and make any other changes to brand your login page. Here's an example of what your page can look like after you make some changes.



- When you're done branding your page, click **Publish**.
- From Community Builder, select **Go to Community Management**, select **Administration**, and then select **Login & Registration**.
- Under Login, for Page, select **Community Builder** and then select **login** from the page picker.
- Click **Save**.

#### SEE ALSO:

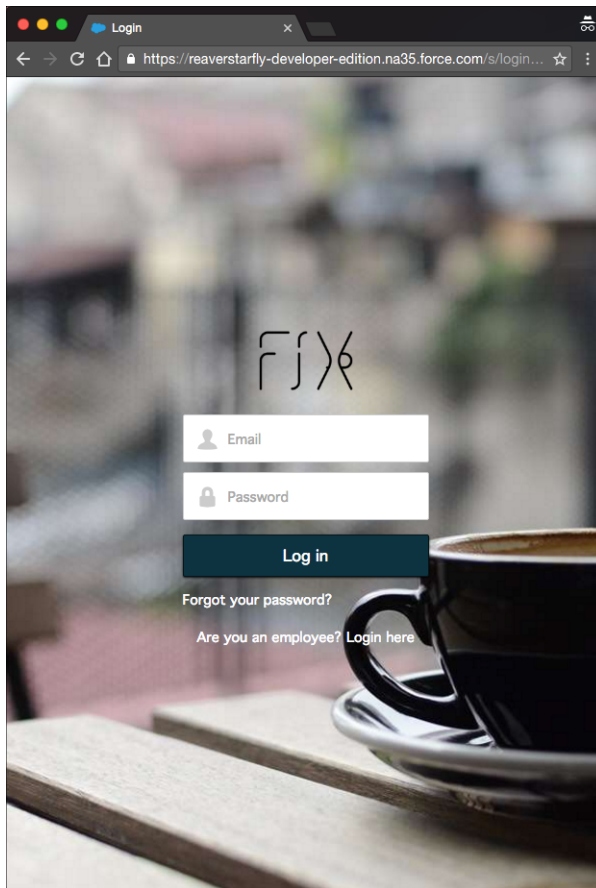
[Steps to Add Authentication to Your Website with Embedded Login](#)

## Activate Your Community

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

- From Setup, enter *All Communities* in the Quick Find box, then select **All Communities** and click the manage link next to your community.
- Select **Administration** and then select **Settings**.

3. Click **Activate Community**, then click **OK**.  
Salesforce sends you an email when the community is activated.
4. Verify that your community is activated.
  - a. From Community Management, select **Administration**, then select **Settings**, and copy the URL of your community.
  - b. Open a new browser or incognito window and paste the URL into the address bar.  
The browser displays the login page for your community, which looks a lot like the login page you just created with Community Builder.



You've completed the basics of branding with Community Builder. As a side benefit, you added links to handle passwords and employee logins. Now, on to setting up a Self-Registration page, which allows you to invite visitors to join your community.

# ENABLE SELF-REGISTRATION

You've deployed a branded login experience, but you don't have any users. Let's invite your users to log in using self-registration. To learn how, you can watch the [Enabling Self-Registration in a Community](#) video. Then follow the steps in this section.

## [Create a Basic Branded Self-Registration Page](#)

Without even knowing it, you've built a branded self-registration page. The Community Builder template and branding you did when you customized your community took care of it automatically. To activate your self-registration page, follow these steps.

## [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](#)

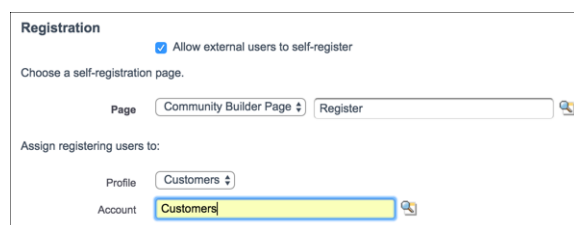
You can add a password field to your self-registration form to require users to supply a password when they register. Because you're already in Community Builder, it's simple to add the field.

## Create a Basic Branded Self-Registration Page

---

Without even knowing it, you've built a branded self-registration page. The Community Builder template and branding you did when you customized your community took care of it automatically. To activate your self-registration page, follow these steps.

1. In Community Management, select **Login & Registration**.
2. Select **Allow external users to self-register**.
3. For page, select **Community Builder Page** and then select **Register**.
4. For Profile, select **Customers**. This setting automatically gives new users your External Identity user profile.
5. For Account, select **Customers**. You already created this account as part of preparing your org. When you're done with the page, it looks like this.



The screenshot shows the 'Registration' configuration interface. At the top, there is a checkbox labeled 'Allow external users to self-register' which is checked. Below this, the instruction 'Choose a self-registration page.' is followed by a 'Page' dropdown menu set to 'Community Builder Page' and a search box containing 'Register'. Underneath, the instruction 'Assign registering users to:' is followed by a 'Profile' dropdown menu set to 'Customers' and an 'Account' dropdown menu also set to 'Customers'.

6. Click **Save**.
7. Return to the browser and reload the login page for your community. Click **Not a Member?** to register for the community.

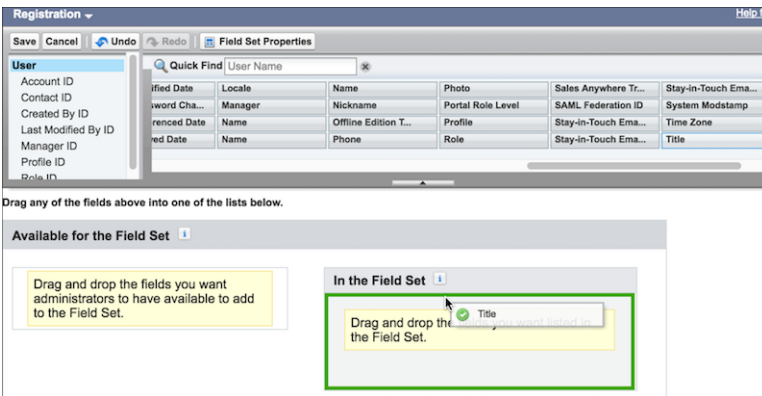
SEE ALSO:

[Embedded Login Considerations](#)

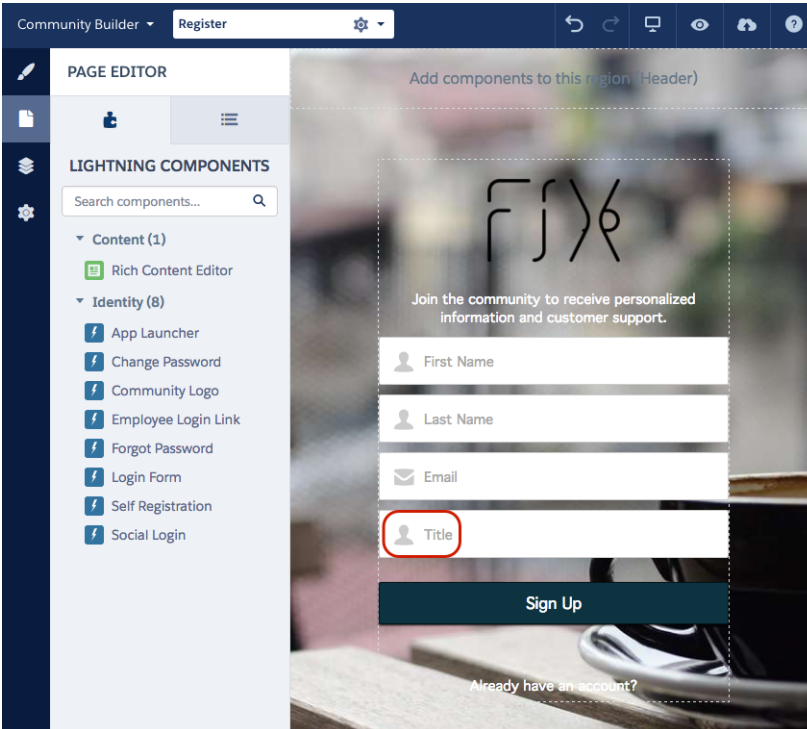
## Add Fields to Collect Additional Information

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 *users* in the Quick Find box, then under Customize Users, select **Field Sets**.
2. Click **New** to create a field set. Name it *Registration*.
3. Under Where is this used, enter *reg field set*.
4. Drag **Title** into the field set, and click **Save**.



5. Go back to Community Builder. From Setup, enter *All Communities* in the Quick Find box, then click **Builder** next to your community.
6. From the dropdown menu at the top, select **Register**, and then select the **Page Editor**.
7. Select the Page Layout icon (☰), and then select **Self Registration**.
8. On the right, scroll to **Extra Fields Field Set Name** and enter *Registration*. The form reloads and displays your title field.



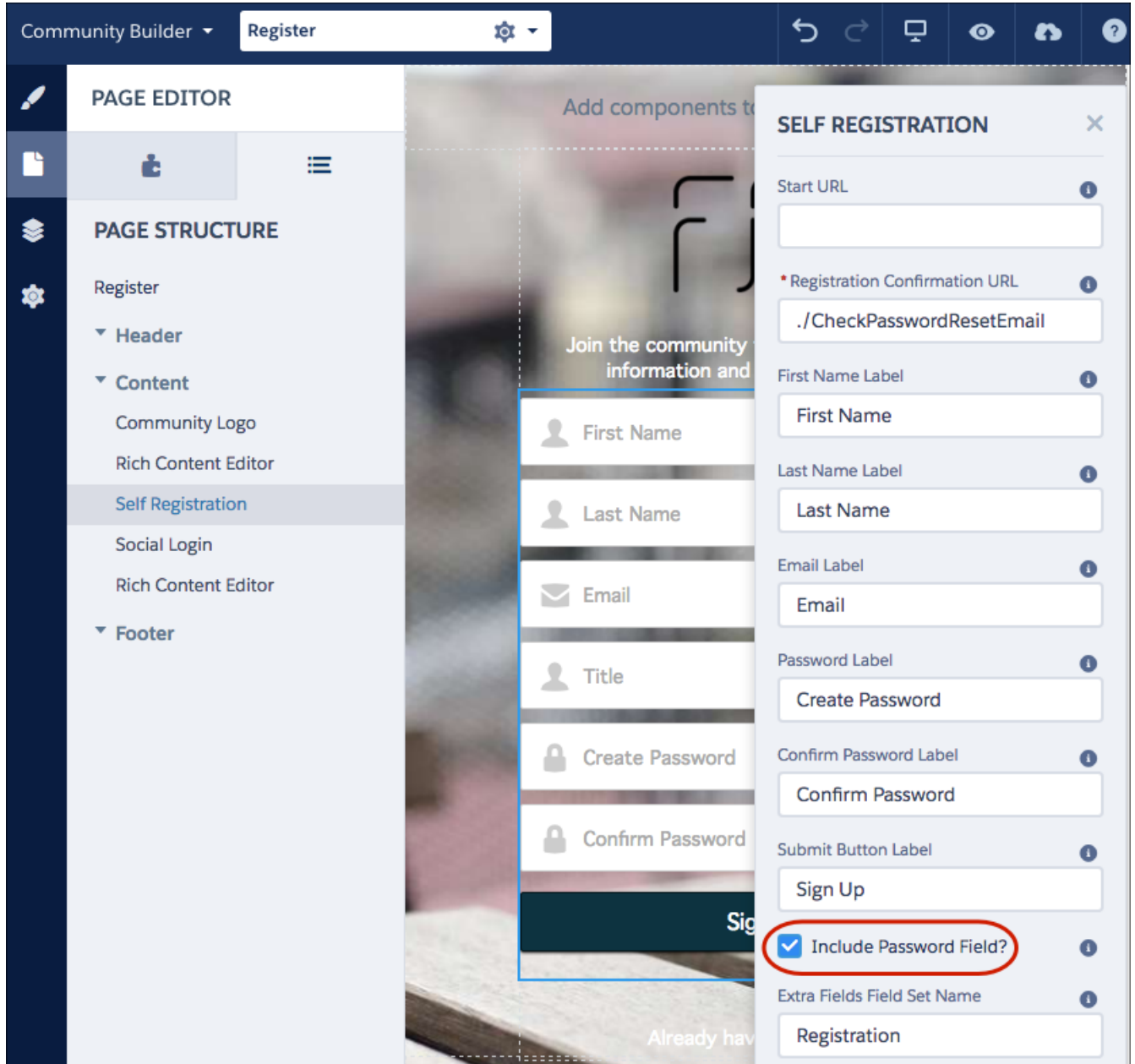
9. Click **Publish**.

## Add a Password Field to Enable Login Directly During Registration

---

You can add a password field to your self-registration form to require users to supply a password when they register. Because you're already in Community Builder, it's simple to add the field.

1. From the Community Builder property manager, select **Include Password Field**.  
The self-registration form reloads and displays the password field.



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

SEE ALSO:

[Embedded Login Considerations](#)



# 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](#)

Before you can use person accounts, you must enable them. It's easy to enable them, but it involves filing a case with Salesforce. Let's walk through it.

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

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

### SEE ALSO:

[Embedded Login Meta Tag Reference](#)

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

## Enable Person Accounts

---

Before you can use person accounts, you must enable them. It's easy to enable them, but it involves filing a case with Salesforce. Let's walk through it.

### 1. Create a record type for business accounts.

You create a record type for business account. You don't create a record type for person account because Salesforce creates it when it enables person accounts.

**a.** From Setup, enter *Accounts* in the Quick Find box, then select **Record Types**.

**b.** Click **New**.

**c.** For Record Type Label, enter *Business Account*, then click **Next**.

**d.** For page layout, select **Account Layout** as the layout to apply to all profiles.

**e.** Click **Save**.

### 2. Verify your sharing settings.

**a.** From Setup, enter *Security Controls* in the Quick Find box, then select **Sharing Settings**.

**b.** Make sure that Contact is set to **Controlled by Parent**.

### 3. Contact Salesforce Customer Support to enable person accounts.

## 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**, then **Pages**, then select **Go to Force.com**.
  - 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 **Manage** 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 form are now B2C-style users using person accounts.

# ACCEPT IDENTITY FROM AN EXISTING IDENTITY PROVIDER

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. Salesforce Identity lets you 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.

## 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 application 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 application 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 user as 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** and click **Manage** next to your community.
2. Select **Administration**, then select **Login & Registration** and confirm that Facebook shows up in the checkbox under Login.
3. Click **Save**.
4. Test your changes by going to your community in a new browser or incognito window.
5. Reload the login page.
6. Click the Facebook logo.
7. 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](#) in the Salesforce Technical Library. 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

Salesforce Identity lets you 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.

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.

With just-in-time (JIT) provisioning, you can use a SAML assertion to create or update users on the fly as part of the SSO process. How? You can either pass a Salesforce-defined set of attributes in your SAML assertion or have Salesforce Identity adapt to existing third-party schemas using Apex provisioning handlers. For more information, see [Just-in-Time Provisioning for Communities in Salesforce Help](#).

# SET UP SSO AND ACCESS FOR YOUR WEB APP

So far we've focused on establishing user identity in Salesforce Identity with self-registration, social sign-on, and branded login services. After you're managing users successfully, the next step is to use identity services like SAML, OpenID Connect, or OAuth engines to provide identity to other applications.

In this section, you set up single sign-on with a sample application using SAML. 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 Application](#)

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 **Apps** and scroll to **Connected Apps**.
2. Click **New**.
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 Application

---

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 have to provide metadata about Salesforce to the app. In practice, this process varies from app to app, but the fundamentals remain the same. You provide the app the unique name of the identity provider, the URLs where it runs, and a certificate to use 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 deployed 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 as follows: 

```
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 browser or incognito window, load your application.
  - b. Click **Login**.
  - c. Click the Facebook icon. If required, log in with Facebook.

You're automatically returned to your application via 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](#)

[Salesforce SSO How Tos](#) in the Salesforce Technical Library

# PROVIDE SSO AND ACCESS FOR MOBILE APPS

Salesforce Identity isn't limited to web applications. You can also use it to provide identity for mobile apps. Without much added work, you can use the Salesforce Mobile SDK to create mobile apps that integrate with everything you've set up.

You interface Salesforce Identity with mobile apps using the OAuth protocol. 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 the Salesforce work together.

To walk through creating a mobile app, 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 the SAML work you just completed.

## [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, but they're not covered here.

## [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, but they're not covered here.

1. In your developer org, from Setup, enter `Apps` in the Quick Find box and then select **Apps**.
2. Scroll to Connected Apps and click **New**.
3. Enter a name for your app. Let's call it My Mobile App.
4. For Contact Email, enter your email address.
5. Click **Enable OAuth Settings**.
6. Enter a callback URL. Use `mymobileapp://callback`.
7. Select the `id`, `openid`, `api`, `refresh_token`, `web`, and `visualforce` scopes.
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

# EMBEDDED LOGIN: AUTHENTICATE YOUR WEBSITE VISITORS

You've created a community to engage your customers and partners. You've branded your login experience and configured the social and identity providers. What's next? You have several options, but here's an easy one with high impact—add login capabilities to any of your website pages with Salesforce Identity Embedded Login.

Creating authenticated sessions between your community and website visitors extends your reach with your customers. 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 a customer's billing address in their Salesforce contact when they purchase something on your site? Embedded Login integrates your website with your Salesforce backend.
- 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.



By configuring 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 just choose which pages to add login capabilities to. Adding login capabilities consists of adding a few HTML meta tags to configure the login form. And then writing a JavaScript function to determine what happens when a user successfully logs in.

Another advantage of Embedded Login is that it provides a way to authenticate users through Salesforce when your website doesn't support user 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. Imagine that you're the owner of Fix Coffee and you've designed a website for your customers to buy your products.

### [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.

### [Steps to Add Authentication to Your Website with Embedded Login](#)

Let's add Embedded Login to a web page in your website. For your Salesforce community, use the external identity community that you created earlier in this guide. If you haven't created 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 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.

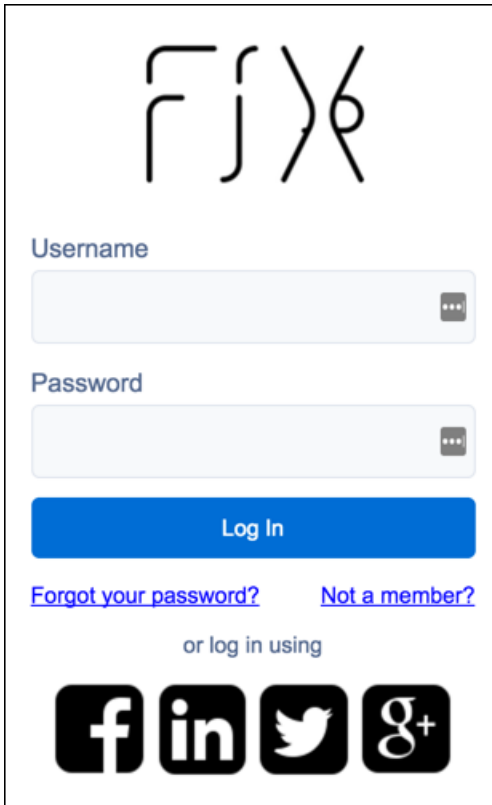
### **Login Button**

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 Form

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.



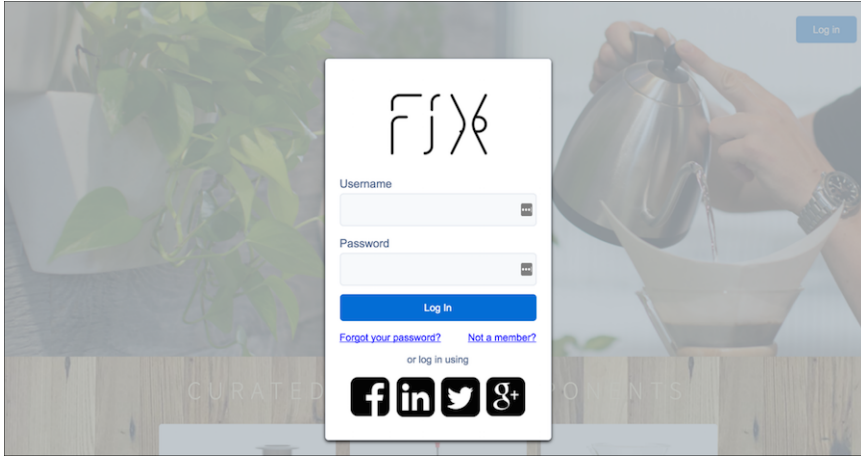
The image shows a login form for a website. At the top, there is a stylized logo consisting of the letters 'F', 'S', and 'X' in a modern, geometric font. Below the logo, there are two input fields: one for 'Username' and one for 'Password'. Each input field has a small 'eye' icon on the right side, indicating a toggle for password visibility. Below the input fields is a blue 'Log In' button. Underneath the button, there are two links: 'Forgot your password?' and 'Not a member?'. Below these links, the text 'or log in using' is displayed. At the bottom, there are four social media icons: Facebook, LinkedIn, Twitter, and Google+.

## 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.

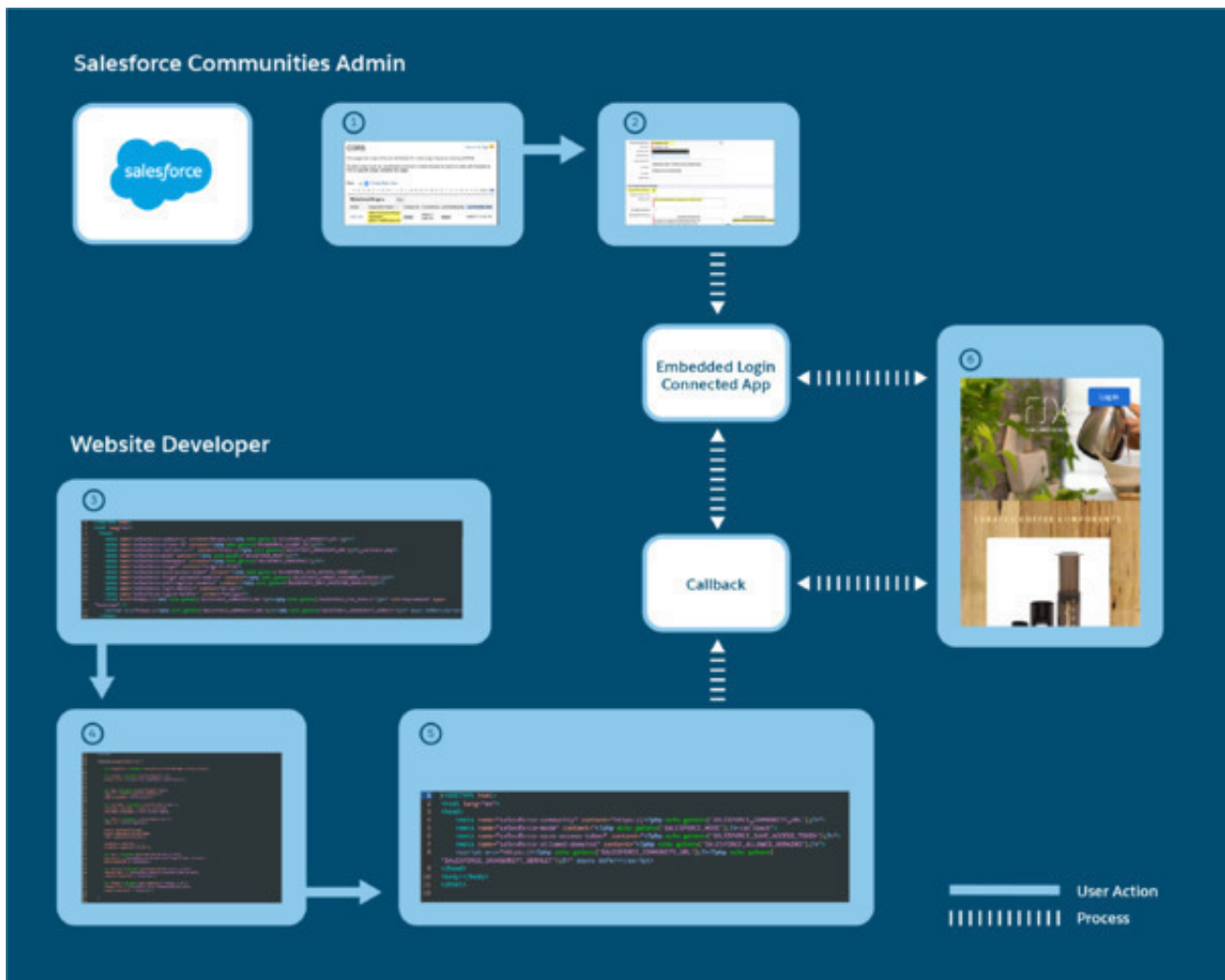


### onLogout Function

You can control what your website users see when they logout. 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.

## Steps to Add Authentication to Your Website with Embedded Login

---

Let's add Embedded Login to a web page in your website. For your Salesforce community, use the external identity community that you created earlier in this guide. If you haven't created 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 backend.

### [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 callback retrieves and stores the access token and user information. Depending on your implementation, you can create a callback to handle the response on either the website or the server.

## SEE ALSO:

[Set Up Your Community](#)[Create a Branded Login Experience](#)

## 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.

1. From Setup, enter `CORS` in the Quick Find box, then select **CORS**.
2. Click **New**.
3. Enter the domain for 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 list them individually.

Browsers cache the Embedded Login JavaScript, including your CORS settings, for 24 hours by default. You can change how often the cache refreshes with the `salesforce-cache-max-age` meta tag. If you're making changes, we recommend that you clear the cache between each change or use an incognito window for testing.

## 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 backend.

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, and 'Allow access to your unique identifier (openid)' is selected as the 'Selected OAuth Scope'.

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 displays the 'Connected App Edit' configuration page. At the top, there is a cloud icon, the text 'Version 1', and a 'Description' field. Below this is the 'Basic Information' section, which contains 'Start URL' and 'Mobile Start URL' input fields. The 'OAuth policies' section is divided into three parts: 'Permitted Users' (a dropdown menu currently showing 'Admin approved users are pre-authorized'), 'IP Relaxation' (a dropdown menu showing 'Enforce IP restrictions'), and 'Refresh Token Policy' (a dropdown menu showing '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:

[Retrieve User Information with Custom Attributes](#)

[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.

This code adds login capabilities to a web page.

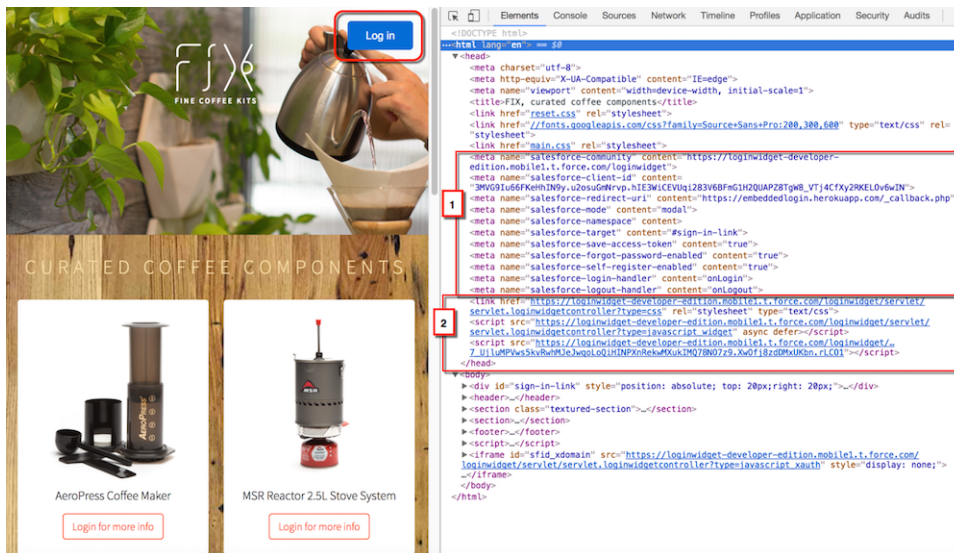
```
<!DOCTYPE html>
<html lang="en">
  <head>

    <title>FIX, curated coffee components</title>

    <!-- Some other meta tags -->
    <meta name="salesforce-community"
content="https://embeddedlogin-developer-edition.na99.force.com/demo">
    <meta name="salesforce-client-id"
content="3MVG9Iu66FKeHhIPrRneLTDfduLfgLjycFpg6SbLpZAJScEXuD.oRdaWnJE7QGnFWHxunp0ut1">
    <meta name="salesforce-redirect-uri"
content="https://embeddedlogin.herokuapp.com/_callback.php">
    <meta name="salesforce-mode" content="modal">
    <meta name="salesforce-target" content="#sign-in-link">
    <meta name="salesforce-login-handler" content="onLogin">
    <meta name="salesforce-logout-handler" content="onLogout">

  <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>
</head>
```

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



At the top (1), a set of meta tags specifies how to display the login form. For example, the `salesforce-community` meta tag specifies the community URL.

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

At the bottom (2), a `<link>` specifies the location of the CSS resources used to style the Embedded Login visual elements. It also contains a `<script>` to invoke Embedded Login on the server.

The CSS and servlet URLs reside on static endpoints hosted by your community. Enter the following lines, replacing `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 \(Optional\)](#)

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 \(Optional\)](#)

## 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.

SEE ALSO:

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

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

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 ALSO:

[Add Embedded Login Meta Tags to Your Web 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.

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 callback retrieves and stores the access token and user information. Depending on your implementation, you can create a callback to handle the response on either the website or the server.

Create a client-side callback by adding a web page to your website. With the client-side callback, the access token and user information are stored in the browser's local storage.

Create a server-side callback servlet on the server to perform the authentication process on the server. With the server-side callback, no access token or user information passes to 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.

[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.

SEE ALSO:

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

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


### 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.oRdaWnJE7QGNFWHxunp0ut1">

  <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=javascipt_widget" async defer></script>
</head>
<body></body>
</html>
```

#### SEE ALSO:

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

## 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.

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 located 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.herokuapp.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.fi314cp0oZ8KpOBs4Mh";

    // 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" +
    // ? Is the comment on the next line at the correct indent ??? Shouldn't it be at
    // the same indent as the next line. Note that this next block is indented from the lines
    // above. Is that correct?
        // 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 5: Handle the Embedded Login Callback](#)

[Force.com REST API Developer Guide: Understanding the Web Server OAuth Authentication Flow](#)

[Force.com 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 Salesforce offers.



## 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.

## Displaying the 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.oRdaWnJE7QGNFWHxunp0ut1&
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.

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 works on all modern browsers: Microsoft® Edge, Google Chrome™, Mozilla® Firefox®, and Apple® Safari®. Embedded Login does not work with Internet® Explorer® 11.

### URL Redirection for Self-registration and Forgotten 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 form, 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.

### Error Configuring CORS

If you get an error that contains this phrase, follow instructions in Step 1 to configure CORS: ... an ancestor value violates the following Content Security Policy directive.

### Missing Login Button

The Login button can't appear on a web page if the user sets the browser to block third-party websites from storing cookies and other data locally. 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 becomes 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, we're serving content 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**.

Embedded Login is supported on all browsers that support Lightning Experience.

SEE ALSO:

[Create a Basic Branded Self-Registration Page](#)

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

[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.oRdaWnJE7QGNFWHxunp0ut1">
```

### 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-forgot-password-enabled

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

```
<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.herokuapp.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. You can set the self-registration link to `true` if your community enabled the feature.

```
<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=3MVG9Iu66FKeHhIPrRneLTDFdiuLfgLjycFpg6SbLpZAJScEXuD.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-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:

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

# EXTERNAL IDENTITY ON GITHUB AND SUCCESS 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 Success Community. The Salesforce Identity Team loves to hear from customers.

# INDEX

<head> 43  
<script> 43, 45  
\$Profile 29

## A

access token 47, 49  
account 3, 10  
activate community 15  
Admin approved users are pre-authorized 40, 54  
All users may self-authorize 54  
Allow access to your unique identifier (openID) 40  
Aloha template 11  
App Launcher 52  
App manager 40  
Auth. provider 24  
Auth. providers 24  
authentication 5  
authentication process 38, 47, 52

## B

B2C 21  
base64 encoding 49  
Brand 38  
browser storage 47  
business accounts 21  
business-to-consumer 21

## C

cache age 45  
callback 49  
callback handler 47  
clear session 46  
Client ID 44  
clone external identity profile 9  
cloud directory services 4  
code example 42–43, 46–47, 49, 52  
communities 7  
Community Builder 14  
Community Management menu 11  
connected app 27, 31  
ConnectedAppPlugin class 42  
Consumer key 44  
CORS 38, 40, 54  
create developer org 8  
CRM 2–3, 10  
Cross Origin Resource Sharing (CORS) 40

CSS stylesheet 43, 45  
custom attributes 29, 42  
custom fields 42  
custom login page 11

## D

dashboards 4  
definition 2  
delegated authentication 23  
developer org 8  
domain name 8  
domain URLs 40

## E

Embedded Login 5  
encoding 49  
Entity ID 28

## F

Facebook 23–24, 29  
federated authentication 23  
field sets 18  
forceios utility 32  
forgot password 13

## G

GitHub 24, 58

## H

Heroku 28

## I

IAM services 1  
Identity Basics 1  
Identity Connect 6  
Identity only license 6  
Identity providers 34  
Identity URLs 42

## J

JavaScript 45  
JIT 23, 26  
just-in-time provisioning 4, 23, 26

### L

licenses 6  
login page 14  
logo 27

### M

mask redirects 44  
membership to community 13  
Metadata Discovery Endpoint 29  
minimize JavaScript 45  
mobile identity 4, 31  
multi-factor authentication 4  
My Domain 8

### N

Not a member link 17, 54

### O

OAuth 31  
OAuth policies 40, 54  
onLogin 46  
onLogout 46  
OpenID Connect 34

### P

password on login page 19  
person accounts 21  
profiles 2, 9–10, 13, 22, 28–29

### R

readable JavaScript 45  
register domain 8

registration handler 24  
resources 45  
roles 9

### S

SAML 28, 34  
SAML-based connected app 27  
sandboxes 8, 19  
Security Policy directive 54  
self-registration 2–3, 17, 21–22  
server-side callback 49  
servlet updates 45  
SFDCOAuthLoginHost 32  
single sign-on 4, 24  
Social sign-on providers 34  
start URL 28  
Start URL 52  
Success Community 58

### T

title field 18  
Trailhead 1

### U

use cases 4, 7, 21, 23

### W

website 5  
whitelist 40  
workflows 4

### X

XCode 32