THE DEFINITIVE GUIDE TO SINGLE SIGN ON (SSO)

By Ado Kukic

Auth0

Table of Contents

The Definitive Guide to Single Sign On (SSO)

Foreword

Thank you for downloading The Definitive Guide to Single Sign On (SSO). Whether you are learning about SSO for the first time, wishing to get the latest trends and best practices, or are in the process of implementing Single Sign On within your app or organization, this book is for you. In this book we'll touch upon many authorization and identity topics with the understanding that you, the reader, have some general technical knowledge and experience. With that said, I will be sure to include many references and resources to aid you further for each topic discussed. The goals of this book are:

- To help you learn and understand what Single Sign On is and how it works
- Understand the benefits of adding Single Sign On within your organization
- Understand the benefits of adding Single Sign On as a premium offering to customers of your apps
- Learn about the various Identity Protocols used in conjunction with SSO
- And finally implement Single Sign On in a guided tutorial

We have a lot to learn so let's get started.

Introduction to Single Sign On (SSO)

Whether you are a manager, a programmer, or copy-writer, you've likely come across identity and authentication. Maybe you were in charge of securing an application or were a frustrated end-user having to remember yet another set of credentials and thinking "is there not a better way?". As we move more and more of our lives online, proper identity management is becoming increasingly important. Single Sign On (SSO) aims to solve the identity crisis for organizations.

Single Sign On (SSO) is a type of authentication in which a user logs in to one system and is automatically granted access to other services. Single Sign On is typically found in enterprise environments where employees access numerous apps and services on a daily basis. Rather than having an employee create a separate set of credentials for each app, they simply login once and can access any app the IT administrator has given them access to. While SSO is promiment in the enterprise landscape, the use case is applicable to organizations of any size and is gaining adoption in organizations of all sizes.

You have likely come across Single Sign On before, even if you didn't know it at the time. Take Google for example. Upon logging in to one Google service such as Gmail, you are automatically authenticated to YouTube, AdSense, Google Analytics, and other Google apps. Likewise, if you log out of your Gmail or other Google apps, you are automatically logged out of all the apps.

Before explaining how Single Sign On works, let's take a brief look at the history of authentication and identity management. Through seeing a brief evolution of authentication, you'll gain a better understanding of where Single Sign On fits in and why it's so important.

A Brief History of Authentication and Identity

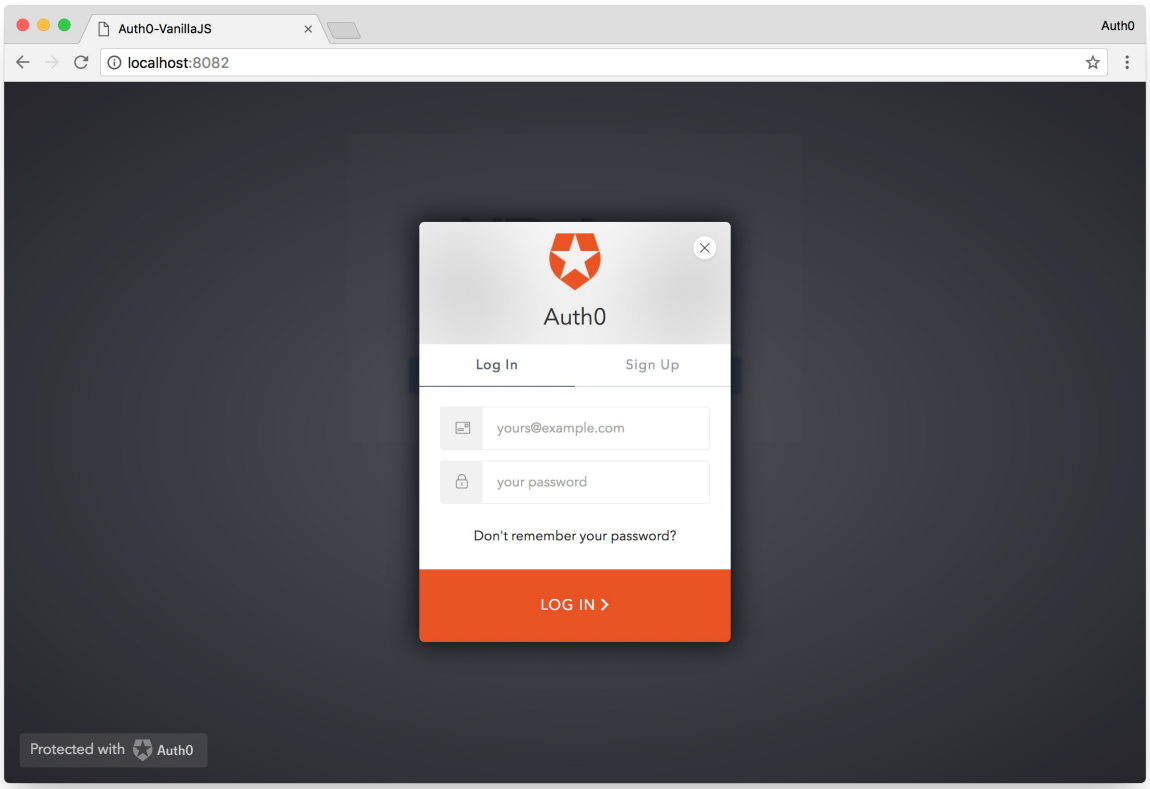
For as long as we've built applications, we've had the need to secure and protect access to them. In the not too distant past, most applications were developed in-house and deployed locally. Security was important, but it was fairly easy to manage.

The rise of the web changed the way software was consumed. Software as a Service (SaaS) has become the standard by which modern applications and services are accessed. Multi-tenant configurations housing thousands of clients alongside their data became the norm. The SaaS model allowed for many start-ups to build specialized products that focused on a singular task and do it well. Rather than a monolithic HR application that handled everything payroll, employee information, and expense reports, a modern organization is likely to have a SaaS application that handles payroll, a SaaS app that handles employee information, and another SaaS app that allows employees to enter their expenses. This proliferation of specialized software has led many enterprises adopting tens or even hundreds of smaller apps to handle day to day operations.

Delegating vital operations to SaaS companies required putting a tremendous amount of trust into these vendors. SaaS vendors, on the other hand, had the responsibility of securing and protecting the data they were entrusted with. While in the past a traditional organization getting hacked would risk leaking its own data, a SaaS vendor getting hacked could put all of its customers at risk. Hacks and data breaches are all too common and security researchers and organizations are constantly fighting the battle to provide better and more secure access.

The approaches to providing stronger authentication and identity management are constantly evolving. Let's examine how authentication has evolved and where we're at today.

Username and Password

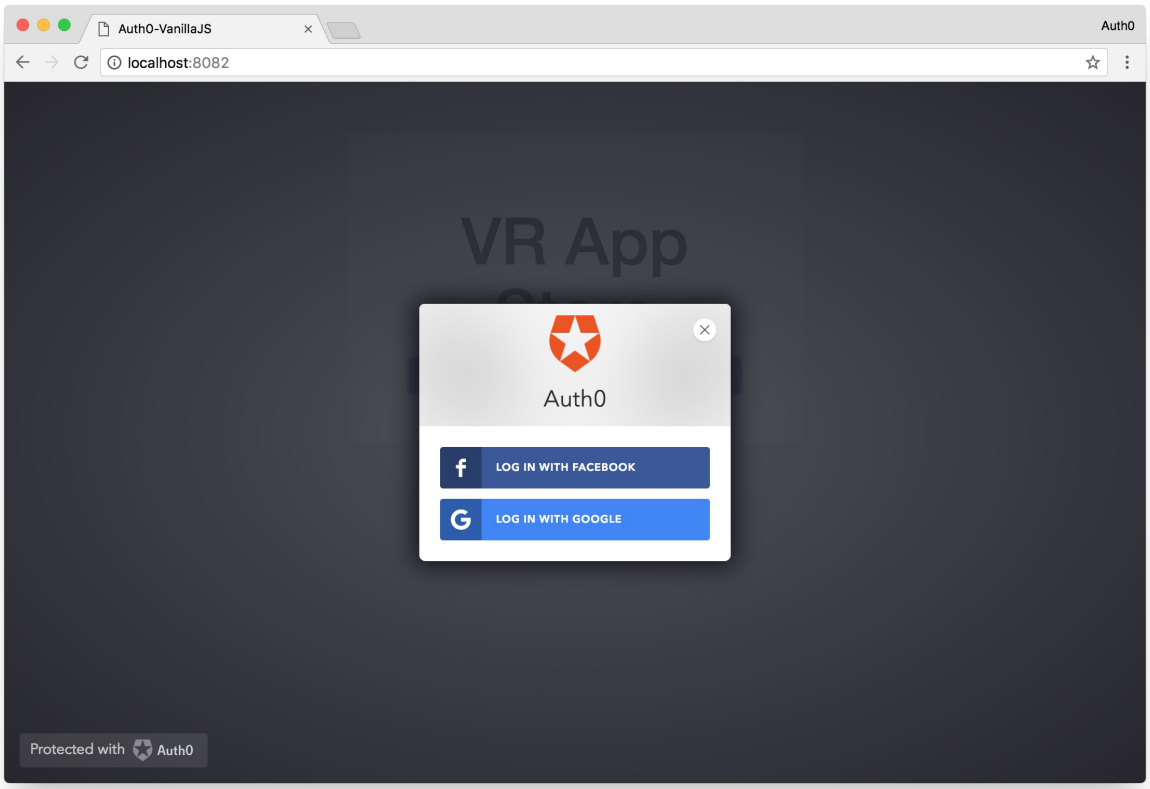


Username and password authentication is the tried and true method of protecting applications. In this type of authentication, a user simply provides their username and password, which is checked against a database of users, and if the credentials match, the user is authenticated. In recent years, it has become commonplace for the username field to be replaced with email or mobile number, but the flow of authentication remains the same.

Username and password authentication is commonplace to this day because it provides a simple experience that most people are familiar with. Since the credentials for this type of authentication are stored in a database, passwords must be hashed and salted. Storing plaintext passwords is very dangerous, because if the database is hacked, the attacker will likely be able to access numerous accounts as users typically reuse the same set of credentials for multiple apps.

Enhancing the username and password authentication flow comes in the flavor of enforcing strong password requirements, forcing password changes every so often, and preventing password reuse. Strong password requirements can range from password length to requirements of special characters, numbers, etc.

Social



Social authentication has gained prominence in the last few years because it allows organizations to authenticate users with existing accounts. Social authentication gets its name from the fact that companies that implement this type of authentication usually allow users to login with social network accounts such as Facebook, LinkedIn, or Twitter. Social networks are by no means the only providers of this type of authentication; any app can become a provider for social connections.

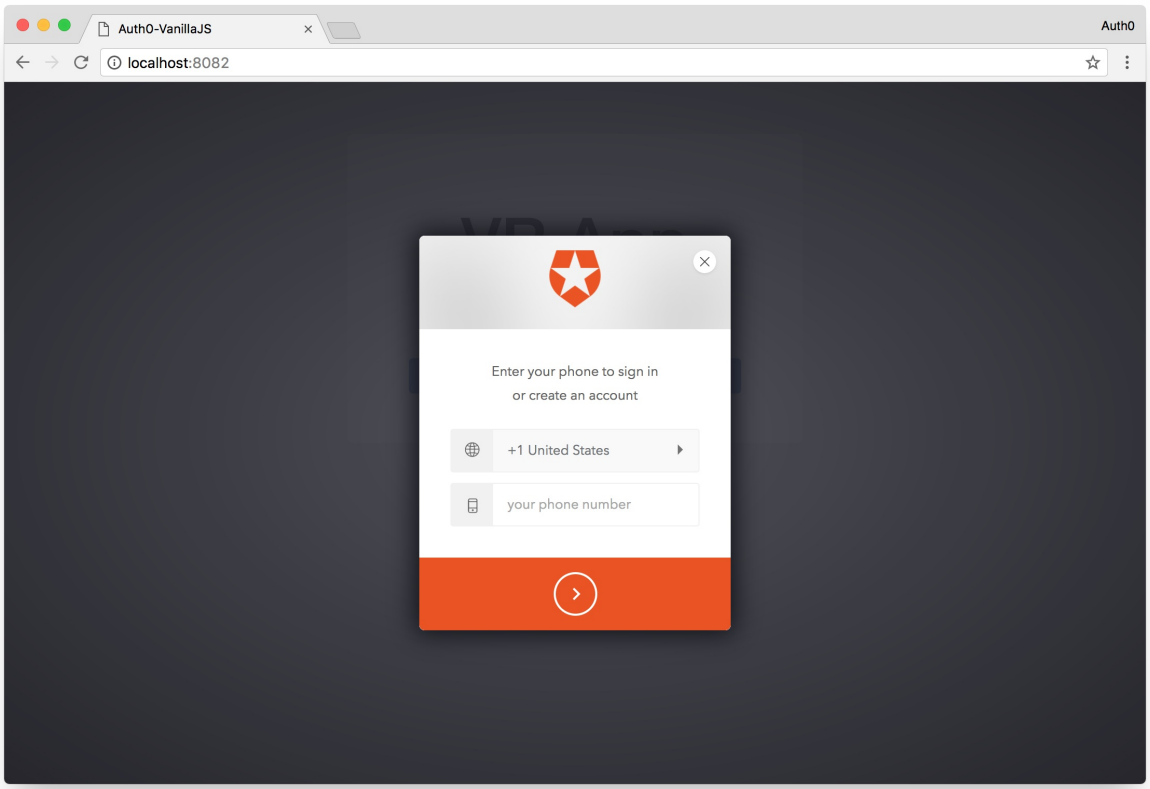
In this type of authentication flow, the user chooses which service they would like to authenticate with. They are then redirected to that application where they login with their credentials. Once verified they are redirected back to the original system. Typically, the user has to choose which sets of data the system can access from the application by explicitly granting them.

Social authentication has the benefit of allowing an app to offload many of the authentication, user data storage, and security duties to the social authentication provider. An organization can simply take the unique id from the social provider and use it as the identifier in their application.

While social authentication provides many benefits, it also has some drawbacks. The biggest drawback is the ability for the social authentication provider to stop providing access which would break the systems authentication system. Organizations have come

up with different ways of dealing with this, such as still requiring the user to create a username and password that is linked to the social account in their system. This is referred to as account linking.

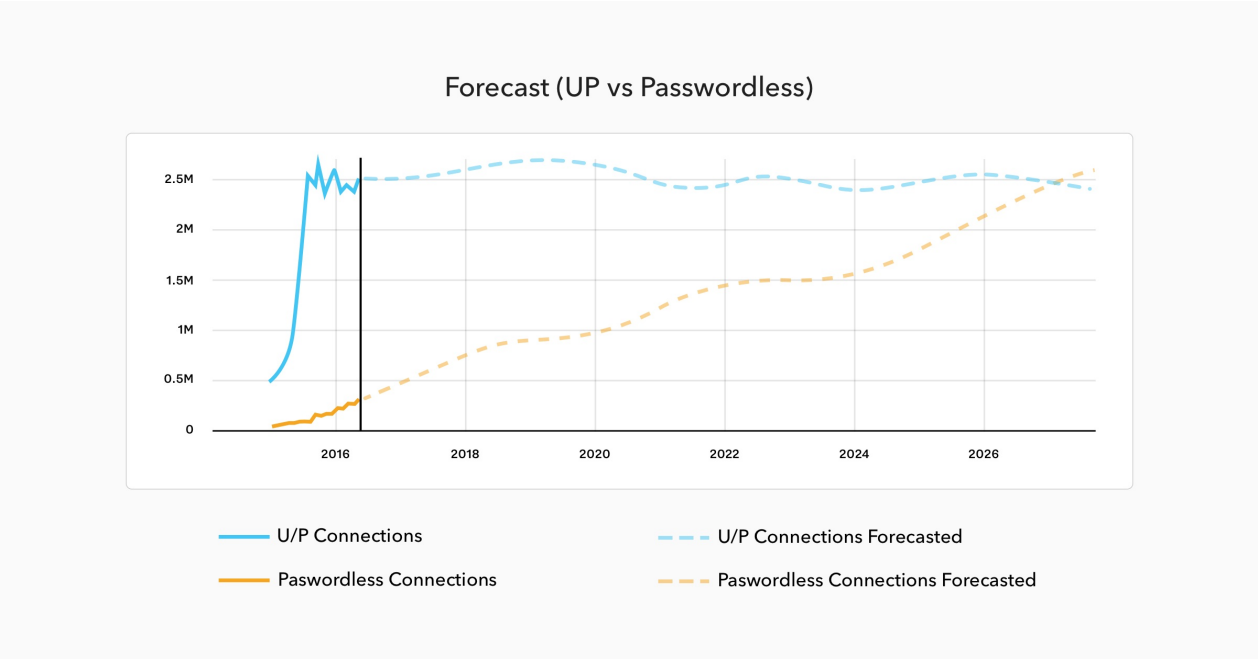
Passwordless



Passwordless authentication is one of the up-and-coming methods of authentication that aims to improve the traditional username and password experience. Wherein username and password authentication a user is required to provide both a username and password, with passwordless authentication, the user simply provides their username. With the username provided, the system generates a one time passcode, referred to as a OTOP, and delivers it to the user via email, SMS, or a dedicated app. The user then provides this code back to the system and the system verifies that the code provided is the one the system issued. If it is, the user is authenticated.

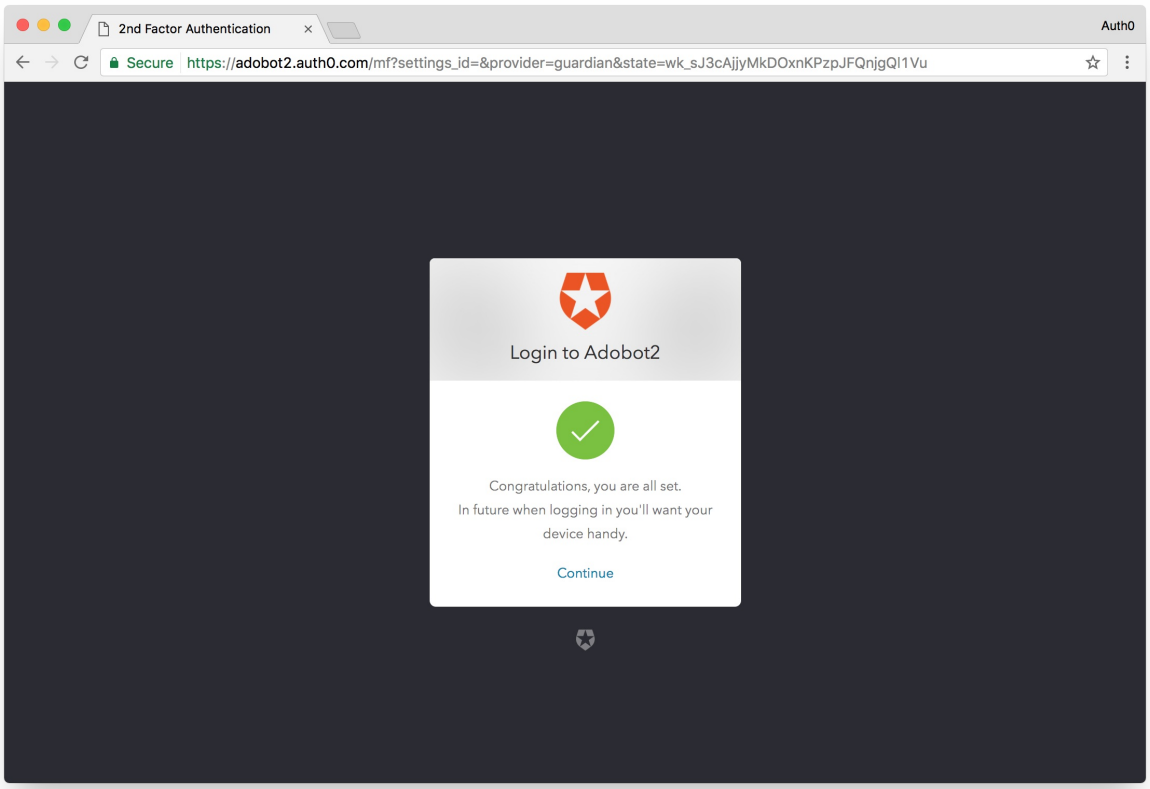
The benefits of passwordless authentication are twofold. One, the owner of the system does not have to take the burden of storing and protecting user passwords. Two, users are not required to remember yet another password.

Data breaches, leaks, and hacks are all too common and the less sensitive data you store about your users the better. It is one of the reasons social authentication has gained prominence and why passwordless authentication adoption is growing.



In a passwordless authentication system the user never has to set or remember a password. A new and random passcode is generated each time they login. The process of entering your username, waiting for a passcode, entering the passcode, and then getting access to a system may seem burdensome and detrimental to a good user experience so companies have found creative ways to better the experience while still getting all the benefits from passwordless authentication. One example is the concept of magic links. Rather than sending the user a passcode, the system sends the user a link via email or SMS, and the user just has to click the link to be authenticated. Users tend to reuse the same password for most services meaning that if they get hacked on one service and their credentials become compromised, all of their other services are at risk. Passwordless authentication negates this risk.

Multifactor Authentication



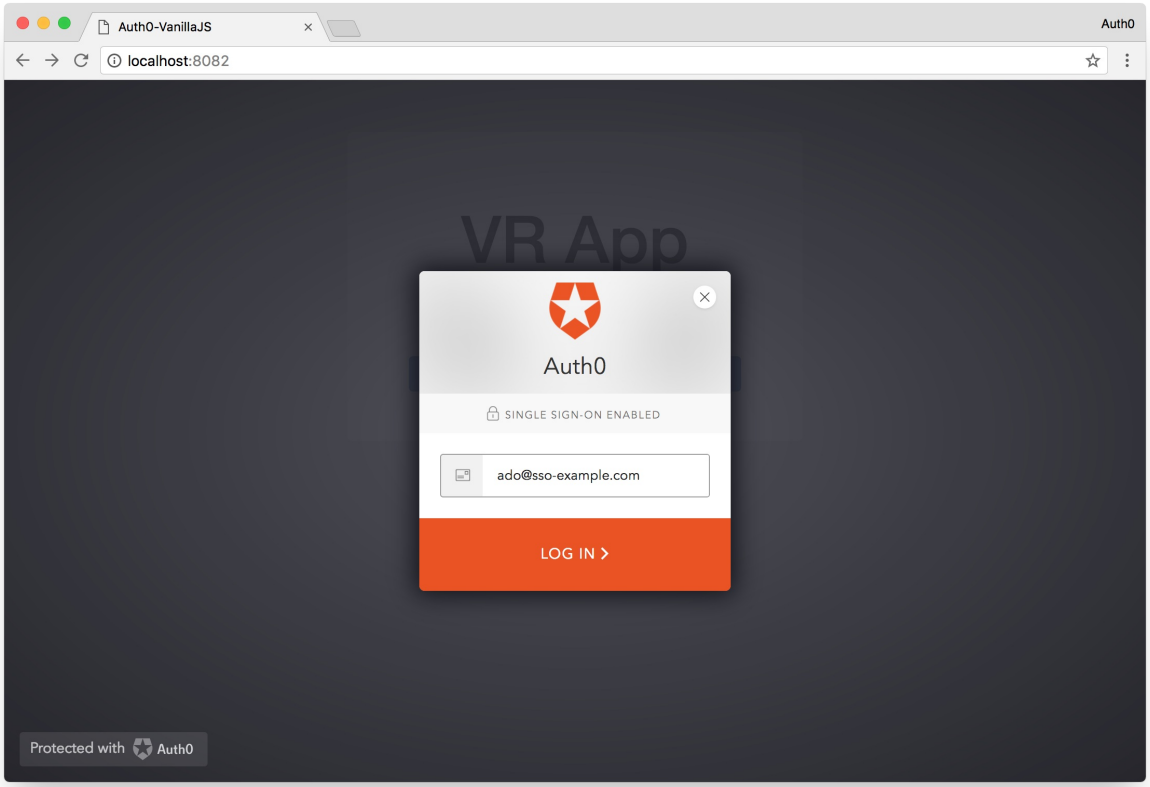
Although multifactor authentication is not a type of authentication in the traditional sense, it deserves special mention as it augments existing authentication methods and makes them more secure. Multifactor authentication aims to enhance security of applications by requiring the user to enter an additional set of credentials before they are granted access to an application. The most common type of multifactor authentication is two-factor authentication (2FA).

In a multifactor authentication flow, once a user has provided a correct set of credentials, they are then tasked with providing an additional piece of data before being allowed access into a system. The additional piece of data can be something the user knows, something the user is, or something the user has. For example, it is common for banks to issue their customers token generator devices that generate a new passcode every minute. When a user logs into their bank account, before they are given access, they must additionally provide the passcode from the token generator. If they cannot provide the code, they are denied access. Another common method of 2FA is upon receiving a correct set of credentials, the system will email or send an SMS with a one time passcode that the user must provide so that they can access the system.

Username and password, social connections, and passwordless authentication are the most common methods of providing secure and privileged access to apps and services. That is not to say that these are the only methods of authentication. Fingerprint or retina

based authentication is gaining popularity as more and more mobile devices are equipped with the hardware to support it.

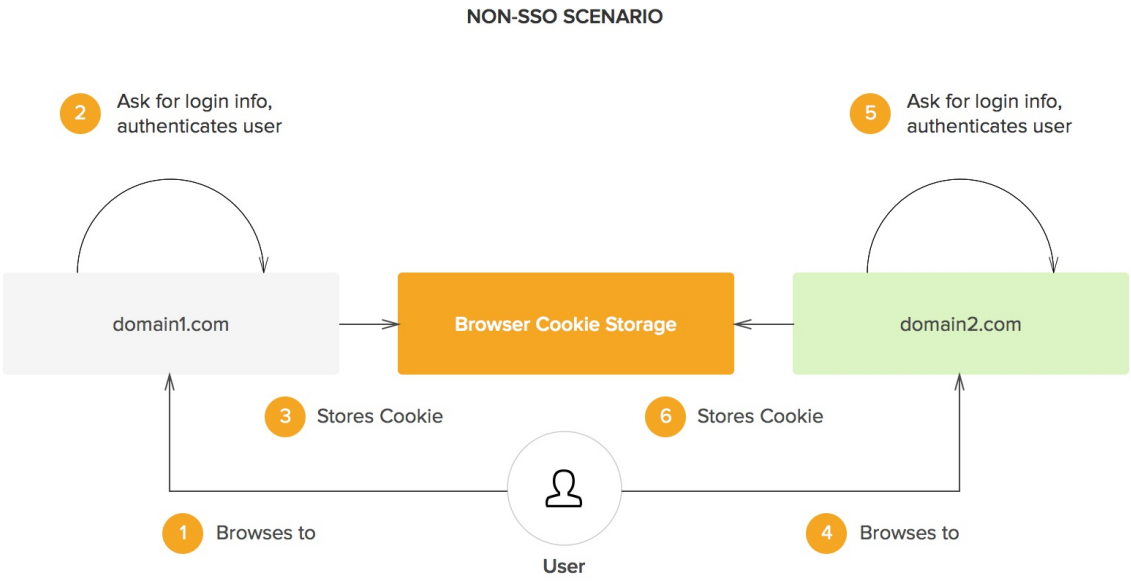
Single Sign On and Identity Federation



Identity federation and Single Sign On go hand-in-hand. As we've mentioned in the introduction, Single Sign On allows a user to login once and gain access to multiple disparate apps and services. Identity federation deals with managing user identities and granting them the rights and privileges to log into these disparate apps and services.

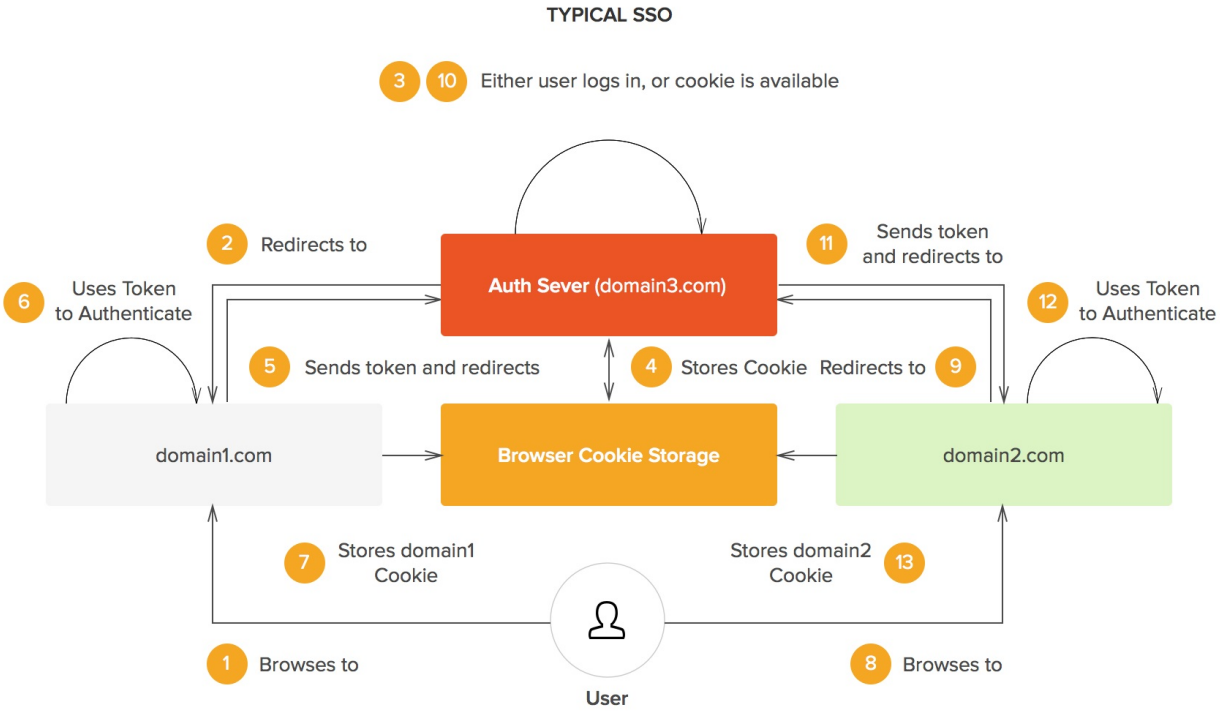
Single Sign On enables a user to login with one of the above mentioned authentication workflows and access multiple applications. This is possible through identity federation and a centralized authentication server. In a non-SSO application, a user will login and their credentials will be sent to the backend system for verification. This backend system is usually the actual application. In the SSO use case, the user credentials are sent to a centralized authentication server, and upon verification, this centralized server grants the user the right to access the application they are attempting to log in to.

The diagram below represents a typical non-SSO scenario in which a user is required to authenticate with different credentials on each domain. In this instance, due to the same-origin browser policy, domain2 would have no way of knowing or accessing data from domain1, thus the need to reauthenticate is present.



The need for a centralized authentication server stems from the same-origin policy that forbids browsers from sharing cookies between domains. Having a centralized authorization system on the other hand, allows many apps to talk directly to the authorization server to see if a user is authenticated and if so grant them access. There are multiple ways this can also be done. In some scenarios, like Google, once a user is logged in to one app they are automatically logged in to all services. In other scenarios, even though a user is logged in to one app, they may still be required to provide some information before getting access.

This diagram shows a typical SSO scenario between two application. The authorization server lives on **domain3**. **Domain1** and **domain2** interface with **domain3** to both authenticate a user and also to verify that the user is authenticated. This may seem very complex, but it'll make a lot more sense as we go through the different chapters of this book and get into the nitty gritty details.



In a Single Sign On implementation, the user will still login with one of the above-mentioned authentication types. The difference is, they will only have to provide this set of credentials once, and the SSO system will take care of granting them correct access to all the other applications. We'll learn much more about this in the coming chapters.

Glossary of SSO and Authentication Terms

Before continuing on, I want to familiarize you with various authentication, identity, and single sign on terminology. This will help put together the various pieces of the identity discussed throughout the rest of this book.

This section is optional and aimed at readers who are new to topics in authentication, so feel free to skip if you were able to follow everything presented so far.

Access Management - Administering the logins and passwords of users across a range of apps and resources—typically contained inside a single organization.

Authentication - Validating an identity as true or false—generally used to verify that a user is who he/she says they are. Most commonly achieved through a username/password combination, but the same principle applies to other forms of authentication like secret questions, secret links, bio-metric identification, etc.

Authorization - Specifying which resources a user (with a given identity) should be allowed to access.

Back-end Server - A server where user information is processed that the user cannot access—where Auth0 authenticates users, for instance.

Claim - A declaration about a subject that is supposed to be true and trusted depending on the identity provider. This declaration could be an attribute such as name, role, or permission.

Client - An application that obtains information from a server for local use.

Credentials - Usernames, passwords, email addresses—any of a variety of means for communicating parties to generate or obtain security tokens.

Delegation - Calling external APIs to authenticate and authorize users. Keeps apps and services from having to store passwords and user information on-site.

Domain - A network where all resources and users are linked to a centralized database on which all authentication and authorization takes place.

Factor - In authentication, a vector through which identity can be confirmed. There are three basic categories— knowledge factors (a password, PIN, security answer), ownership factors (security token, ID card), and inherence factors (fingerprint, DNA, retinal scan).

Federated Identity Management - A system of shared protocols that allows user identities to be managed across organizations.

Federation Provider - An identity provider that provides single sign on, consistency in authorization practices, attributes exchange practices, and user management practices between identity providers (issuers) and relying parties (applications).

Forest - A collection of domains overseen by one central authority.

Identification - The process by which a user's information is received, collected, and taken up for authentication.

Identity Provider (IdP) - A website, app, or service responsible for coordinating identities between users and clients. An IdP can provide a user with identifying information and serve that information to services when the user requests access.

Kerberos - A ticket-based protocol for authentication built on symmetric-key cryptography.

Multifactor Authentication - An authentication process that takes into account multiple factors. Commonly used in reference to two-factor authentication, which most commonly appears in the form of an SMS code sent to a supplement a user's username/password login.

Multitenancy - A term in software architecture referring to the serving of many users (tenants) from a single instance of an application. The most common form for SaaS products, which exist as a single instance but have dedicated shares served to many companies and teams.

OAuth - An open standard for authorization. Development began in 2006 as employees from companies like Twitter and Google saw the need for a set of shared protocols dictating how web services should authorize other web apps to access to their users' information. At its most simple, it works like this:

1. User is prompted to authorize the client, or not, for a specific need (access to your Facebook friends list, say)
2. Proof of that authorization is sent to an (external) authentication server
3. Authentication server gives the client a token representing access to the user's friends list

OpenID - An open standard for authentication. Allows third-party services to verify that users are who they say they are without clients needing to collect, store, and therefore become liable for a user's login information. At its most simple, it works like this:

1. User selects OpenID option upon login
2. Client sends external server (your Facebook, Google, Twitter, etc.) an authentication request
3. External server verifies the identity of the user, sending proof to user if successful
4. User sends proof of authentication to the client
5. Client approves or denies access

Passwordless - A form of authentication based on tokens, most commonly received and sent through SMS, email (magic links) or biometric sensors. Entirely based on inherence and ownership factors, making passwordless more secure than traditional username/password logins.

Role - An aspect of a user's identity that gives them certain permissions.

Role-Based Access Control (RBAC) - A model for authorization based on users gaining certain permissions based on their roles.

Security Assertion Markup Language (SAML) - An authentication and authorization standard commonly found in the enterprise, SAML differs from Open ID in that it does not dynamically discover and accept authentication from new identity providers. The IdPs that a service wants to trust must be specified and hard-coded into each login event. Typically used to give the users of a corporate network access to a specific 3rd party service—for instance, so you don't have to sign in again when you click a link to Salesforce on your company's intranet.

Single Sign On - A subset of federated identity management, a means through which authentication and interoperability can be achieved in a federated system.

Social Identity Provider (Social IdP) - A term used to refer to identity providers originating in social services like Facebook, Google, Twitter, etc.

Software as a Service (SaaS) - A model for software purchasing that relies on monthly subscriptions rather than the one-time purchase of a license.

Web Identity - Identifying characters extracted from an HTTP request, often an authenticated email address.

Windows Identity - How Active Directory organizes user information.

WS-Federation - A federated standard or common infrastructure for identity, used both by web services and browsers on Windows Identity Foundation.

For a more exahustive list of authentication and identity terms and definitions, be sure to check out Auth0's Identity Glossary.

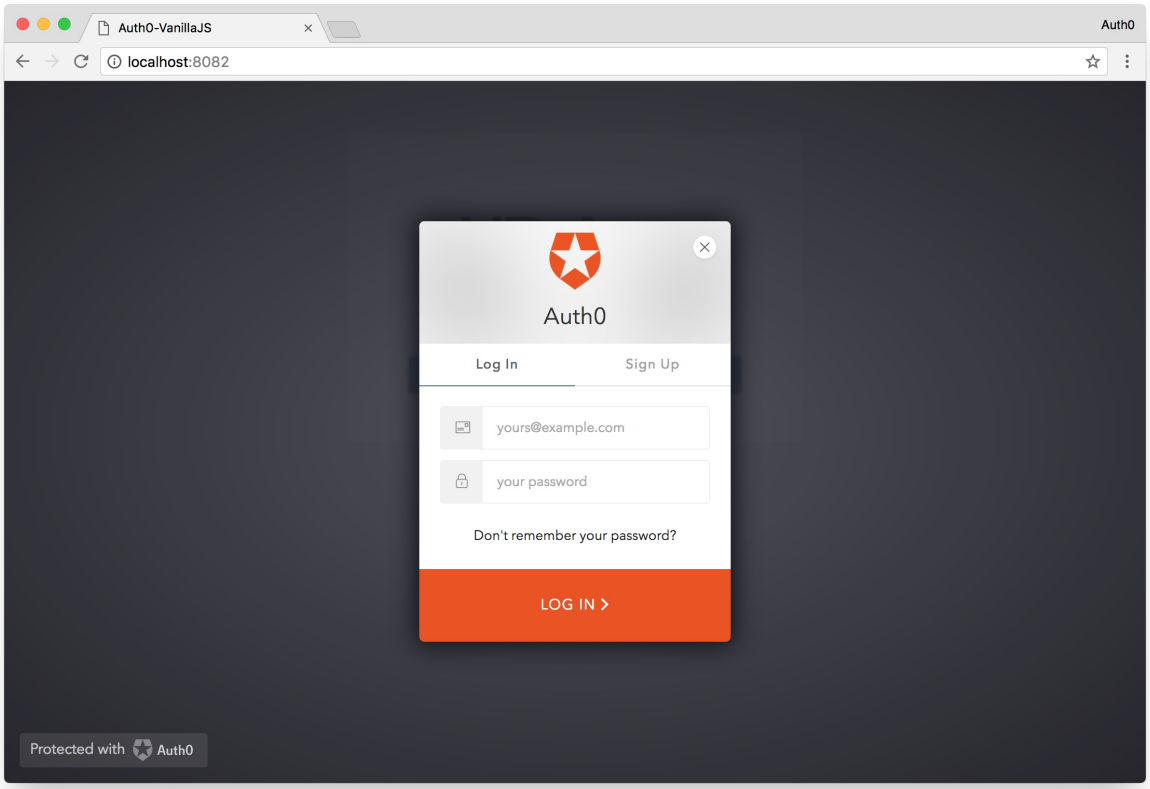
Road to Single Sign On

Now that we understand what Single Sign On is and have somewhat of an idea of how it works, we are ready to start the discussion on modern identity management. The next chapter will focus on what modern identity management is and how Single Sign On achieves the goal of making modern identity management simple and secure.

Identity for the Modern World

Authentication plays a pivotal role in verifying that users are who they say they are. This is just a small piece of modern identity management. Modern identity management is concerned with **authentication**, **authorization**, and **management** of users within a system or organization.

Authentication



Authentication, as we've seen, deals with verifying user identity. We've discussed various types of authentication such as username and password, social, and passwordless authentication. We also looked at how organizations enhance their authentication capabilities with multifactor authentication. Authentication and identity is synonymous with the login screen and many people believe that that's where identity starts and ends. After all, once you're past the login screen, you very rarely have to think about your identity.

A good authentication experience can seem at odds with properly implemented security. You'll often find applications requiring you to enter a password that's at least eight characters long, has an uppercase letter, a lowercase letter, a special character, and other requirements that seem very inconvenient. On top of that, many organizations, require frequent password changes at set intervals like 30 or 90 days. On the other hand, if an app does not enforce any password requirements it is much more susceptible to data breaches and other security issues.

Keeping up with these requirements for one application is inconvenient enough, but doing so for five or ten applications leads users to adopt some pretty bad practices like reusing the same password for each app, writing down their passwords on a sticky note and taping it to their monitor, and so on.

Single Sign On can provide a great experience that will keep the end-users happy while providing strong security measures that can be implemented across an entire organization. Since in an SSO implementation there is a single source of identity, IT administrators can have a central system for enforcing password requirements, spotting authentication anomalies, and so on.

Authorization

Authorization deals with ensuring that users have the correct levels of access within a system. Once a user is successfully authenticated, they are given certain permissions for what they can do within an application. For example, in a payroll application an accountant is able to process payroll and see details for all of the employees, while an individual employee can just see their details and change their direct deposit information.

Ensuring the right level of access is very important and is difficult to do in a non-SSO implementation. Keeping track of who has access to what is likely to keep IT administrators very busy. Different applications also present different levels of granularity when it comes to user permissions which makes authorization even more difficult.

Single Sign On aims to solve the authorization component of modern identity by providing a centralized source of truth for each individual user. Once a user is authenticated, the identity provider can send each application the users identity and permissions for each application. IT administrators now have a single location where they can manage user roles and permissions for the entire organization. This makes it easier to audit and ensure the right users have the right access at all times.

Identity Management

Identity management is concerned with creating, updating, and deleting users as they move throughout an organization. Let's look at identity management through the lens of an employee. On day one, an employee cheerfully walks into the office and is welcomed by their boss. They are given access to the various apps and tools they will need to get their job done. Six months later, the employee is promoted due to their excellent work ethic and performance and is given access to more tools. Two years later, the employee is hired by a competitor and leaves the organization.

In an organization that does not have Single Sign On implemented, the employee is likely required to set up three, four or more sets of credentials on their first day of work. These credentials are likely to reuse the same password, unless each app the employee access has different requirements, in which case the employee might just walk out. Once they are promoted, they are given access to more apps, which means more credentials to set and remember. Finally, when the employee leaves the organization, the task of tracking down each account the employee had access to and deactivating it can be very challenging. In many instances, the employee may retain access for months or even years after they leave an organization.

In an SSO environment, on day one the employee is provided with a set of credentials that are mapped to a user account in a centralized database. The IT administrator grants the user permissions to the apps that their boss has approved for them. Once the employee is promoted, the IT admin simply updates the user account with new roles and permissions. When the user finally decides to leave the organization, the IT administrator can in one fell swoop disable access to all of the organizations resources for that particular account.

Single Sign On and Modern Identity Management Go Hand in Hand

All three fundamental concepts of modern identity are improved with Single Sign On. In a modern setting, a user is unlikely to only need access to a single app. A single employee in an organization may have upwards of ten, twenty, or even more applications that they need access to in order to complete their tasks. If all these disparate apps required their own sets of credentials two things would likely happen. One, the user would end up reusing the same password across most of these apps. Two, the user would not care to use many of the apps unless absolutely necessary. One

is especially dangerous, because if any of the apps the user created an account for is compromised, hackers could gain access to all the apps. Two affects performance and utilization. You've spent time, money, and other resources building apps, you want them to be used as often as possible.

Aside from the security and user experience perspectives, a third side-effect of user management occurs. Managing access to incoming and outgoing employees can be very challenging without a centralized location for user identity.

Single Sign On provides a solution for all three components of modern identity. Through identity federation, an IT administrator can provision and deprovision user accounts once, grant them varying levels of access for each app, and enforce security requirements all from a single source of truth. The authentication experience is simplified for the user, while the organization benefits from better security. Authorization and identity management is handled from a centralized location and can be easily managed.

Single Sign On can solve an organization's identity and user management woes, but implementing SSO is not an easy task. There are many competing standards and protocols as well as costs associated with implementing a SSO solution. Since this is such a valuable feature, SaaS vendors are able to charge additional fees for enabling SSO. Take Slack for example. Slack paid user accounts start at $8/mo per account when billed monthly. Organizations wishing to use Slack with Single Sign On, on the other hand, need to pay for a "Plus" account which costs $15/mo per account when billed monthly. This is almost a 50% increase and goes to show that Single Sign On is not only beneficial, but sought after as well.

Organizations will pay extra for Single Sign On and identity federation because it allows them to retain control of their users. This presents additional revenue opportunities for your organization. Modern identity management is an increasing requirement for many organizations wishing to sell SaaS. Single Sign On is a key consideration for many organizations and may even be a disqualifying attribute if it's not available.

In the next chapter we are going to discuss the various identity protocols for implementing Single Sign On.

Identity Protocols and Providers

In the previous chapter we showed the benefits and value proposition of implementing and offering Single Sign On in your organization. The good news is that SSO provides many benefits and can be a key selling point for your application, the not so good news is that there are many different ways in which SSO can be implemented. There are many competing standards and protocols and knowing which one to implement for your organization is half the battle. If you are selling SaaS, then it is likely that you may need to support multiple of these protocols which will make your job that much harder.

This chapter is dedicated to highlighting the common protocols used in SSO, their benefits and downsides, and examples where these protocols are used. In this chapter, we'll also discuss various identity providers and the pros and cons of each. We'll dive right in with one of the most widely used protocols for managing federated identity: **SAML**.

Security Assertion Markup Language (SAML)

Security Assertion Markup Language[1] or SAML is one of the most widely used protocols when it comes to Single Sign On implementations. SAML dates back to 2001, with the last major revision to SAML 2.0 being released in 2005 and for a long time was the de-facto protocol for implementing SSO.

The SAML protocol exchanges authorization and authentication data in XML format. The three components to this data exchange are the **user**, **identity provider**, and **service provider**. In this scenario, a user requests a resource from the service provider. The service provider, before giving the user access to the resources, checks with the identity provider if the user should have access to this resource. The identity provider, acting as the single source of truth, verifies the users identity, typically through a set of credentials like username and password, and if valid asserts back to the service provider that the user should have access along with the users identity.

This exchange is facilitated by **assertions**. The service provider makes request assertions to the identity provider, and the identity provider, upon verifying the credentials, returns a response assertion with the user data. An example response assertion can be seen below.

```
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" ID="_e5c34c8e3dc3ec7adcdf"
```

```
<saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">urn:adobot2.auth0.com</saml
<samlp:Status>
  <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
</samlp:Status>
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" Version="2.0" ID="_4zEfq
  <saml:Issuer>urn:adobot2.auth0.com</saml:Issuer>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="#_4zEfqA7qOuqvhdjm5eSk1UGXhamc0YKd">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
          <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>LluBcZGrhjPV9yI3zsSb6KY7VHA=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>...</SignatureValue>
    <KeyInfo>
      <X509Data>
        <X509Certificate>...</X509Certificate>
      </X509Data>
    </KeyInfo>
  </Signature>
  <saml:Subject>
    <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">auth0|5898d
    <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
      <saml:SubjectConfirmationData NotOnOrAfter="2017-02-14T23:02:32.047Z" Recipient="http
    </saml:SubjectConfirmation>
  </saml:Subject>
  <saml:Conditions NotBefore="2017-02-14T22:02:32.047Z" NotOnOrAfter="2017-02-14T23:02:32.0
    <saml:AudienceRestriction>
      <saml:Audience>urn:auth0:adobot:SAMLP-Auth0-SSO</saml:Audience>
    </saml:AudienceRestriction>
  </saml:Conditions>
  <saml:AttributeStatement xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://ww
    <saml:Attribute Name="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentif
      <saml:AttributeValue xsi:type="xs:string">auth0|5898d895f7b5312a57f85696</saml:Attrib
    </saml:Attribute>
    <saml:Attribute Name="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddres
      <saml:AttributeValue xsi:type="xs:string">ado@sso-example.com</saml:AttributeValue>
    </saml:Attribute>
    <saml:Attribute Name="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name" NameF
      <saml:AttributeValue xsi:type="xs:string">ado@sso-example.com</saml:AttributeValue>
    </saml:Attribute>
    <saml:Attribute Name="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn" NameFo
      <saml:AttributeValue xsi:type="xs:string">ado@sso-example.com</saml:AttributeValue>
    </saml:Attribute>
    <saml:Attribute Name="http://schemas.auth0.com/identities/default/provider" NameFormat=
      <saml:AttributeValue xsi:type="xs:string">auth0</saml:AttributeValue>
```

```
        </saml:Attribute>
        <saml:Attribute Name="http://schemas.auth0.com/identities/default/connection" NameForma
          <saml:AttributeValue xsi:type="xs:string">Username-Password-Authentication</saml:Attr
        </saml:Attribute>
        <saml:Attribute Name="http://schemas.auth0.com/identities/default/isSocial" NameFormat=
          <saml:AttributeValue xsi:type="xs:boolean">false</saml:AttributeValue>
        </saml:Attribute>
        <saml:Attribute Name="http://schemas.auth0.com/nickname" NameFormat="urn:oasis:names:tc
          <saml:AttributeValue xsi:type="xs:string">ado</saml:AttributeValue>
        </saml:Attribute>
        <saml:Attribute Name="http://schemas.auth0.com/updated_at" NameFormat="urn:oasis:names:
          <saml:AttributeValue xsi:type="xs:anyType">Tue Feb 14 2017 22:02:32 GMT+0000 (UTC)</s
        </saml:Attribute>
        <saml:Attribute Name="http://schemas.auth0.com/created_at" NameFormat="urn:oasis:names:
          <saml:AttributeValue xsi:type="xs:anyType">Mon Feb 06 2017 20:12:05 GMT+0000 (UTC)</s
        </saml:Attribute>
      </saml:AttributeStatement>
      <saml:AuthnStatement AuthnInstant="2017-02-14T22:02:32.047Z" SessionIndex="_9E2mFWD7D8cHW
        <saml:AuthnContext>
          <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified</saml:A
        </saml:AuthnContext>
      </saml:AuthnStatement>
    </saml:Assertion>
  </samlp:Response>
```

These XML request and response assertions are self-encompassing and independent of the protocol used to transfer them so developers can send these requests in the body of a POST request for example or wrap them in a SOAP binding. This flexibility means that SAML can be introduced in many different environments and just work.

One of the major benefits of SAML is that the protocol is only concerned with handing the exchange between the identity provider and service provider. The authentication aspect is not defined in the protocol, so developers are free to choose how users authenticate. For example, if you have an existing database of users, you can have them validate against that database rather than rebuilding your identity management system from scratch. You can also enable enhanced security features like multifactor authentication and your SAML implementation will not need to change.

With SAML, you are not limited to the number of identity or service providers. One identity provider can interact with many service providers and one service provider can interact with many identity providers. This capability ensures that your organization will be able to interact with not just your own internal applications, but also third-party applications which you may be using.

WS-Federation

WS-Federation[2] or WS-Fed is another commonly found protocol used in Single Sign On implementations. For the most part, it is very similar to SAML. WS-Fed comes from the WS-* series of services, falling under the WS-Security branch. WS or Web Services aim to provide a standardized way of handling online communications with varying degrees of success and maturity.

If your organization is already using WS-* services for security and trust, WS-Fed will allow you to extend these capabilities to Single Sign On scenarios. WS-Fed is built on top of WS-Trust which uses the Security Token Services (STS) model to facilitate the transfer of claims data between services. WS-Federation, like SAML, is built around the concept of an Identity Provider and Service Provider, in which a Service Provider requests identity information from an Identity Provider, while the Identity Provider is tasked with verifying incoming requests have the correct set of credentials.

The WS-Federation protocol is also XML based looks very similar to that of SAML. An example of federation metadata can be seen below.

```
<fed:FederationMetadata>
  <fed:Federation>
    <fed:TokenIssuerName>
      http://www.auth0.com
    </fed:TokenIssuerName>
    <fed:TokenIssuerEndpoint>
      <wsa:Address>http://www.auth0.com/STS</wsa:Address>
    </fed:TokenIssuerEndpoint>
    <fed:TokenTypesOffered>
      ...
    </fed:TokenTypesOffered>
    <fed:SingleSignOutNotificationEndpoint>
      <wsa:Address>http://www.auth0.com/SignOut</wsa:Address>
    </fed:SingleSignOutNotificationEndpoint>
    <fed:UriNamedClaimTypesOffered>
      <fed:ClaimType Uri="http://schemas.xmlsoap.org/claims/EmailAddress">
        <fed:DisplayName>Email</fed:DisplayName>
      </fed:ClaimType>
      <fed:ClaimType Uri="http://schemas.xmlsoap.org/claims/UPN">
        <fed:DisplayName>Name</fed:DisplayName>
      </fed:ClaimType>
    </fed:UriNamedClaimTypesOffered>
    <fed:TokenSigningKeyInfo>
      <wsse:SecurityTokenReference>
        <ds:X509Data>
          <ds:X509Certificate>
            ...
```

```
            </ds:X509Certificate>
          </ds:X509Data>
        </wsse:SecurityTokenReference>
      </fed:TokenSigningKeyInfo>
    </fed:Federation>
  </fed:FederationMetadata>
```

WS-Federation shares many benefits already discussed in the SAML protocol. This protocol was developed to extend the capabilities of the existing WS-Security model. WS-Federation is commonly found in large enterprise companies, especially those embedded in the Microsoft ecosystem.

OpenID Connect / OAuth

OpenID Connect (OIDC)[3] is an authentication protocol, based on the OAuth 2.0 family of specifications. It handles authentication via JSON Web Tokens (JWTs) delivered via the OAuth 2.0 protocol. OpenID Connect is a fairly recent protocol, with version 1.0 of the framework being adopted in 2014.

While OAuth 2.0 is concerned with resource access and sharing, OIDC is all about user authentication. Like the previous protocols, OIDC facilitates access through a centralized identity provider. A user wishing to gain access to an application utilizing OIDC, will first be redirected to the OIDC provider, authenticate, the the identity provider will return the users identity to the application.

OpenID Connect is typically used by social networks that allow developers to use them as identity providers. The flow for OpenID Connect is different from that of SAML and WS-Federation. With OpenID Connect, an application sends a request to an identity provider. The identity provider, verifies the user and upon successful verification, prompts the user to grant data access to the application that initiated the request. Once a user agrees to share the data, the identity provider generates a token, called an `id_token` and returns it to the application. This token contains user identity information and the application consumes this token and grants the user access.

JSON Web Tokens are widely used in OAuth and OIDC. A JSON Web Token or JWT is comprised of three parts. The **header**, **payload**, and **signature.** Each JWT is signed by an algorithm such as HMAC SHA256 and can only be verified by the correct key. While token verification can only be done with the right credentials, JWT's are only encoded, not encrypted. This means that they can be decoded and their contents read by

anybody. For this reason, it is recommended that sensitive information never be stored in these tokens. To learn more about JSON Web Tokens, be sure to check out the introduction at https://jwt.io/introduction.

For the purposes of OIDC, each token contains identity data for the user in the form of claims. The OIDC spec defines a series of reserved claims, public claims and private claims. Reserved claims are typically reserved for metadata, for example `iss` is for the issuance date of the token. Public claims are agreed upon in the IANA JSON Web Token Registry, where claims like `givenName` are defined for the users first name. This is to help with interoperability. Finally private claims are those that an application or identity provider defines themselves. These can be named anything and represent any type of data the identity provider wants to represent.

The format of JSON Web Tokens is that of:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRt
```

When decoded the claims represented in the payload of the JWT are:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

OpenID Connect is a fairly modern approach to Single Sign On but is widely adopted in the consumer section. If you've ever logged into a web or mobile application with your Google or Facebook account, you've used OpenID Connect. Although Facebook has developed it's own protocol called Facebook Connect that shares many aspects of OIDC. OIDC is not as common in the enterprise environment as it is with consumer facing applications.

Lightweight Directory Access Protocol (LDAP)

The **Lightweight Directory Access Protocol (LDAP)**[4] is an application protocol, used for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network. The LDAP protocol allows access to a centralized directory of credentials which can be shared amongst multiple applications. The LDAP protocol is the oldest one we'll look at in this book, dating back to 1996.

LDAP works by enabling access to an existing directory of users. LDAP is typically paired with Active Directory and allows administrators to have a centralized location for user identity. Application clients send requests to the LDAP server and in return the server responds with information about user identity. There are a number of different actions a client can request such as authenticating a user, updating user information, deleting users, and more.

Information exchanged over LDAP is done over LDAP Data Interchange Format (LDIF) which looks looks like this:

```
dn: cn=John Doe,dc=example,dc=com
cn: John Doe
givenName: John
sn: Doe
mail: john@example.com
manager: cn=Jane Doe,dc=example,dc=com
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
```

Companies may use the LDAP protocol to provide SSO services to their employees within an intranet. LDAP based SSO is not common outside of company intranets.

Identity Providers

The identity protocols dictate how the flow of Single Sign On works within an organization. Identity providers are concerned with actually storing and managing user identities. In this section, we are going to look at the different ways of storing user identity.

Database / Local

A common way of managing identity is through a local database which is only concerned with users and their roles. Organizations requiring full control over their data will often opt for this option for their identity provider. Users typically log into the database via username and password authentication, although organization do tend to enforce multifactor authentication and other enhanced security measures.

Microsoft - Active Directory, ADFS, Azure AD

Microsoft is no stranger to federated identity and has a long history of offering various products and solutions for managing user identity. Active Directory is one of the most popular identity providers in the Microsoft Enterprise space. Active Directory worked with Windows Server technologies to provide Single Sign On functionality to not just web applications but the entire Windows ecosystem. In recent years, Microsoft has begun offering their various online services like Azure Active Directory, bringing their tried and true identity provider to the cloud.

Social Providers

Social providers can often make for great identity providers. Social providers, like Facebook or Google, typically make use of the OAuth protocol for managing user identity. The benefit of using a large social provider as an identity provider is that these organizations typically have some of the best security standards for user accounts in the world. For organizations like Facebook, that hold billions of identities, top-notch security is a must to the point where many of these organizations are writing the rulebooks for managing modern identity.

Using social providers as your identity provider is not without risk though. Social providers typically have all the control and can revoke access, change their API, or stop providing identity services altogether in the blink of an eye. With great convenience comes great risk in the identity game.

Other

There are many companies, like Auth0 and SSO Circle, that can become identity providers for your organization. These companies, amongst many others, specialize in identity management and single sign on and often provide additional services to make identity management simple for your organization.

Organizations both large and small are demanding Single Sign On and it has become a requirement for companies wishing to compete in today's competitive landscape. SSO should be a check-the-box item for your organization during due diligence rather than a capability your organization must demonstrate to a potential customer. In the build vs buy debate, the argument has always been to build and focus on what makes your product unique and buy everything else. Identity management is a prime candidate that can be delegated to a company specializing in authentication and modern identity management.

> [1]. Security Markup Assertion Language Wikipedia Page ↩

> [2]. Understanding WS-Federation ↩

> [3]. OpenID Connect ↩

> [4]. Lightweight Directory Access Protocol Wikipedia Page ↩

Use Cases

In this chapter we are going to examine if Single Sign On is right for you and your organization. The benefits of SSO cannot be understated, but it's also important to ensure that SSO is a right fit for your organization. So far we've looked at Single Sign On through the value proposition and technological lenses, but now let's look at it from a real world use case lens.

Single Sign On (SSO) for Your Organization

Whether you are a small startup of ten or large enterprise of thousands, Single Sign On enables the consolidation of user identity and management. Think of how many different applications you use every day. Email, issue tracker, file hosting, CRM software, and others. If you require a unique set of credentials for each one of these apps, then you likely have a very fragmented identity system. Many software vendors offer SSO integrations, but they do come at a higher price.

If you wish to get your identity management consolidated under one roof, Single Sign On can help. Consolidating identity in an organization through SSO will require a centralized identity provider and depending on your existing infrastructure you are likely going to want to use SAML or WS-Federation.

Single Sign On (SSO) for Your Applications

If your organization develops applications, Single Sign On can be both a differentiator and a requirement. Adding this feature can seem like a daunting task, but companies like Auth0 can help make the integration simple. Let's examine how Single Sign On can benefit your business.

Business to Consumer (B2C)

Building applications for consumers, such as e-commerce or media applications, means getting the user experience right. Consumers have an abundance of choice and a bad first impression is enough to get them to leave and never come back. The login screen is

typically the first experience users have with your application and offering a Single Sign On solution here can mean the difference between a signup and an abandonment.

Offering Single Sign On through social logins is a great way to get new signups and additional data from users. The experience for consumers is great as they do not have to remember another set of credentials; they just login with their existing social media account. The ease of use and low barrier to entry makes it an ideal candidate for B2C applications. OAuth and OpenID Connect based SSO is the implementation that will achieve this goal.

The challenge with providing Single Sign On solutions for the B2C market is choosing which identity providers to connect to. Large IdP's like Facebook, Google, and Twitter are the obvious choices, but depending on your specific applications, offering GitHub if your application is focused on developers or a more niche IdP may also be important. Supporting multiple IdP's and giving consumers choice is common but the task can be time and cost intensive. Identity providers may change their API's or policies requiring your organization to move fast to keep up with the changes.

For your employees, a traditional SAML based SSO solution may also be applicable. This would allow your employees to access the B2C application without having to create a separate account. This would make managing internal identity a lot easier for your IT team.

Business to Business (B2B)

The demand for Single Sign On in the B2B space is on the rise. This represents both business opportunity as well as increased revenue potential. As we've seen in previous chapters, companies that do offer SSO solutions are able to charge more for the service. Taking a look at the Slack example again, a standard account costs $8 per user per month, while a plus account which supports SSO and allows your employees to login with existing credentials costs $15 per user per month. Slack is just one example, but the majority of B2B companies like Trello, Box, and Jira offer this type of tiered pricing system.

Managing identity is very important for businesses and they are willing to pay for it. For many organizations, maintaining the ownership of their user identities is a requirement and the only way to do that is through Single Sign On. If your organization does not have this capability, it may disqualify your app from further evaluation.

For B2B software, your organization will typically need to support SAML, LDAP or WS-Federation SSO. SAML based SSO is most commonly found in B2B SaaS, so that is a great starting point, but the more protocols your organization can support the better.

Business to Enterprise (B2E)

Single Sign On is pretty much a requirement when selling to the enterprise. Large enterprises demand governance over their users, and the only way to ensure compliance is through SSO. SAML, WS-Federation and LDAP based SSO are the standards here, although OAuth and OpenID Connect are gaining ground.

Supporting many different variations of SSO is essential to success when selling to the enterprise as each enterprise is likely to have unique requirements. Large enterprises typically run many custom applications so having a versatile SSO solution that supports many different protocols and IdP's is very important.

Single Sign On: Build vs Buy

No discussion on use cases would be complete without discussing how to achieve the goal. We've seen how Single Sign On works and learned about the different protocols, but one element that we did not talk about is how to achieve Single Sign On throughout your organization. Do you build it out internally and manage it yourself or is it better to purchase SaaS that provides SSO solutions that can be integrated into your products?

The Argument for Build

Building your own solution, whether it be a blog, framework, or identity can be beneficial because you can build to your exact specifications. Reducing dependencies on external vendors can also be seen as a positive. One less service to worry about going down. Building the solution gives you complete control over all of the data as well as how and where it is stored. Finally, from an expense stand point, you can manage costs by only building what you need.

The arguments against building the solution in-house, especially something as important as identity, is the expertise required to get it done right. Identity is very complex and understanding the ins and outs of all the moving pieces is beyond the realm of many development teams. Adding Single Sign On in the mix adds yet another layer of complexity. Identity is constantly changing, new standards get defined, bugs get identified, and needs of businesses evolve. By building an identity solution in-house your

organization will be tasked with maintaining, patching, and evolving your identity solution, which will take time, money, and other resources away from your core business.

The Argument for Buy

The proliferation of SaaS has allowed many companies to be built around solving one problem and solving it the best. SaaS is so ubiquitous that is has become an adage in the technology sector to "focus on building what makes your product unique, buy the rest."

In the past it was commonplace for companies to build their own database solutions, their own email servers, and so on, but in today's modern environment businesses are offloading as many tasks as they can and just focusing on their core differentiator. If you are an e-commerce company, you may use one of the many off-the-shelf products like Shopify to manage the store front and just focus on getting your products out there. When building software, delegating identity management to a third-party vendor may be very beneficial.

The big reasons to buy your identity management platform is that you can get your solution up and running in no time. Saving time versus building, testing, qa'ing, and deploying a home-grown solution can mean the difference between getting to market in weeks or months. Identity is for the most part a solved problem, so building your own solution is just another way of reinventing the wheel. Having identity management and Single Sign On is more of a standard rather than a differentiator. The technical know-how of implementing identity is another consideration. Identity and Single Sign On are complex solutions and a wrong implementation can leak user data which could be disastrous for your organization. Identity management companies are laser-focused on implementing secure solutions and are on top of their game when it comes to evolving standards.

The disadvantage to buying an off the shelf solution for identity management and Single Sign On is that you lose some control. Your customer's identity data is stored on third-party servers. Adding an off the shelf solution also means that you are introducing another dependency to your technology stack. If the third party service experiences downtime, your application will also be affected. Vendor lock-in is another consideration. Although many identity vendors offer standards based solutions, there may be specific features and functionality that you would lose if you decided to switch vendors. Finally, the third party service may not be able to customize their solution for your exact needs.

There are many considerations to make when deciding to build or buy your Single Sign On solution and each case is different. Typically, if you don't have extensive resources for a dedicated identity management team, you are better off buying a secure off the shelf solution. In the next chapter, we will look at how you can implement Single Sign On with a third party identity management vendor.

Implementing Single Sign On (SSO) Exercise

So far we've covered many topics including the what, why, where, and when to use Single Sign On, and in this chapter we'll focus on the **how**. Theory is great, but I want to supplement your knowledge with a little bit of practice. The game plan is:

- Implement SAML based SSO
- Use Auth0 as the Identity Provider
- Authenticate various clients via SSO

We will implement SAML based SSO in our application with Auth0 serving as our Identity Provider. We've chosen this implementation as SAML is an open standard while Auth0 provides an easy to use identity framework with with many features we've discussed in this book, so we'll get a chance to actually see how some of these features work in practice.

Building a Modern Application

Modern applications are made up of two parts: backend and frontend. The backend is the brains of the operation and handles operations like authentication, authorization, database access, and so on, while the frontend client is what the users interact with. The frontend makes API calls to the backend, while the backend processes those calls, gathers the data and sends a response back to the frontend.

In our exercise we are going to build various front end clients to demonstrate SSO functionality. We'll omit building the backend in the interest of time. Since we want to demonstrate SSO functionality, let's keep it simple and build our applications with plain JavaScript.

Our Organization

You have just joined a new startup that focuses on selling virtual reality apps and games. The company, still new but quickly growing, has three custom applications for managing the product and employees. There is an Admin Dashboard application that allows employees to manage the store-front, add products, set prices, and other

administrative tasks. The VR App Store app is used by customers as well as employees. Customers can create accounts and make purchases, while employees can login and purchase applications at a discount. Finally, the HR App allows employees to login and manage their company documents, request time off, and other human resources related functions.

In a non-SSO environment, you would need a separate set of credentials for each application. That's three logins to remember, three sets of credentials to keep safe, and three identities for the IT administrator to take care of. The company is also looking to build a Analytics app to better understand how users are using the platform. Another set of credentials to remember. No bueno.

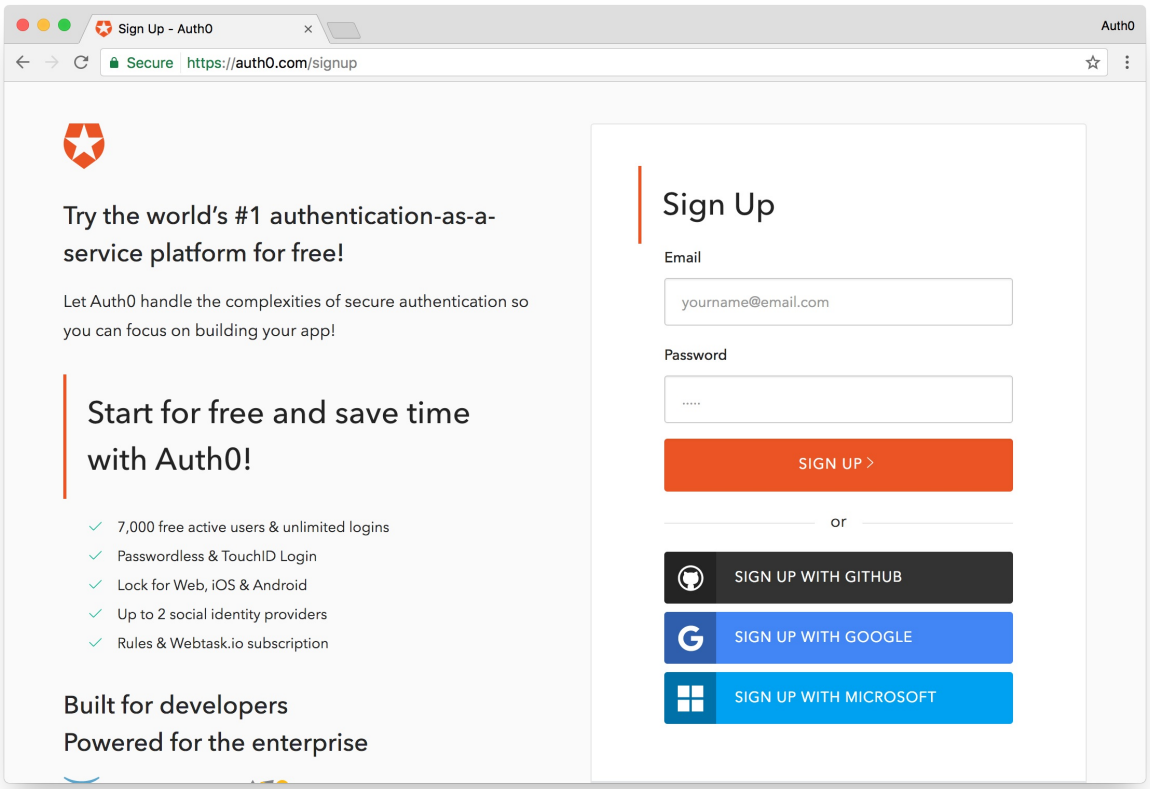
Luckily, the company hasn't launched yet, and is in the exploratory stages of identity and authentication, so you suggest they explore a Single Sign On solution to consolidate the user accounts across all these applications. Your idea is approved, but you've been tasked with implementing it. Let's get into it.

Deciding on the Identity Provider

The first step to setting up Single Sign On in deciding on an identity provider. Remember from our earlier chapters that the identity provider is the source of truth for identity in SSO. All accounts are managed by the Identity Provider or IdP.

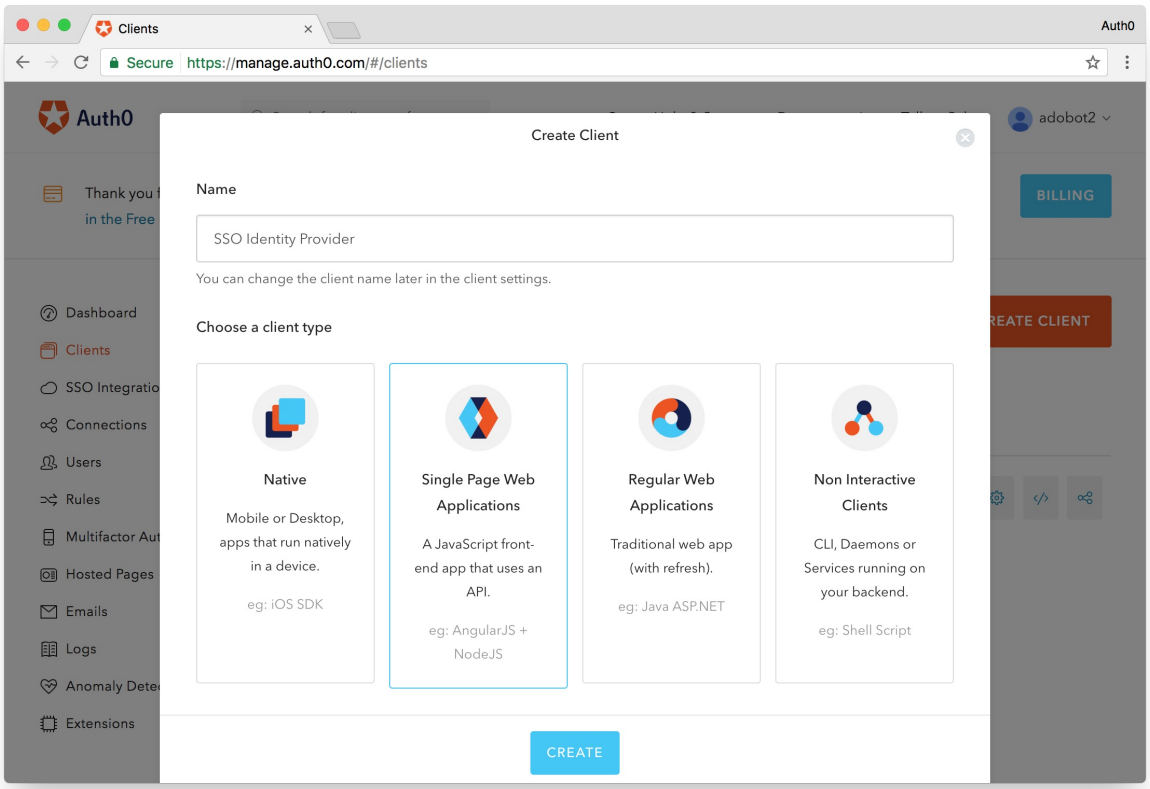
There are many options and considerations deciding on an IdP. If you visit this Wikipedia page https://en.wikipedia.org/wiki/SAML-based_products_and_services\) you can see a large number of different companies and services that provide IdP services. Some work with proprietary protocols like Active Directory, while others provide open source alternatives through SAML, OpenID Connect, and so on.

For our Identity Provider, we'll use Auth0, and we'll build our SSO based on SAML. To get started, sign up for a free Auth0 account at https://auth0.com/signup

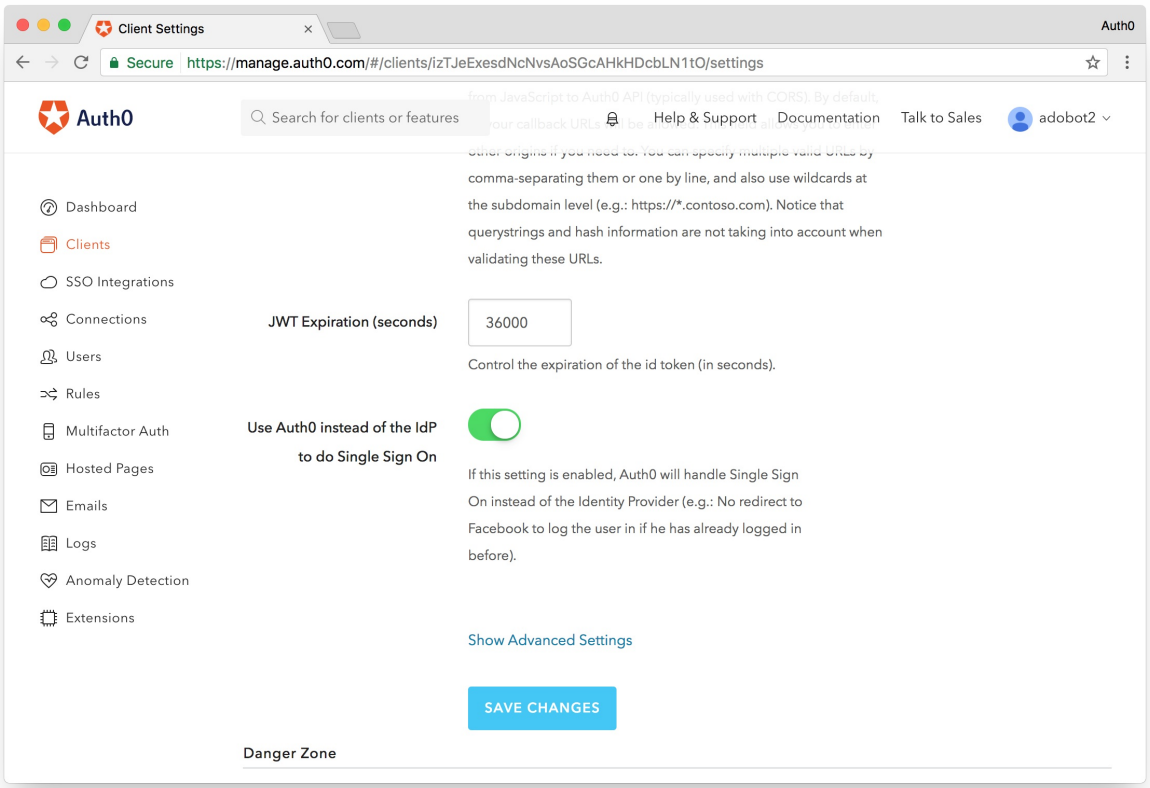


Once you have your Auth0 account created, navigate to the **Clients** section in the Auth0 dashboard and click on the **Create Client** button. A client in Auth0 is essentially a sandbox that exposes authentication and authorization functionality based on your configuration.

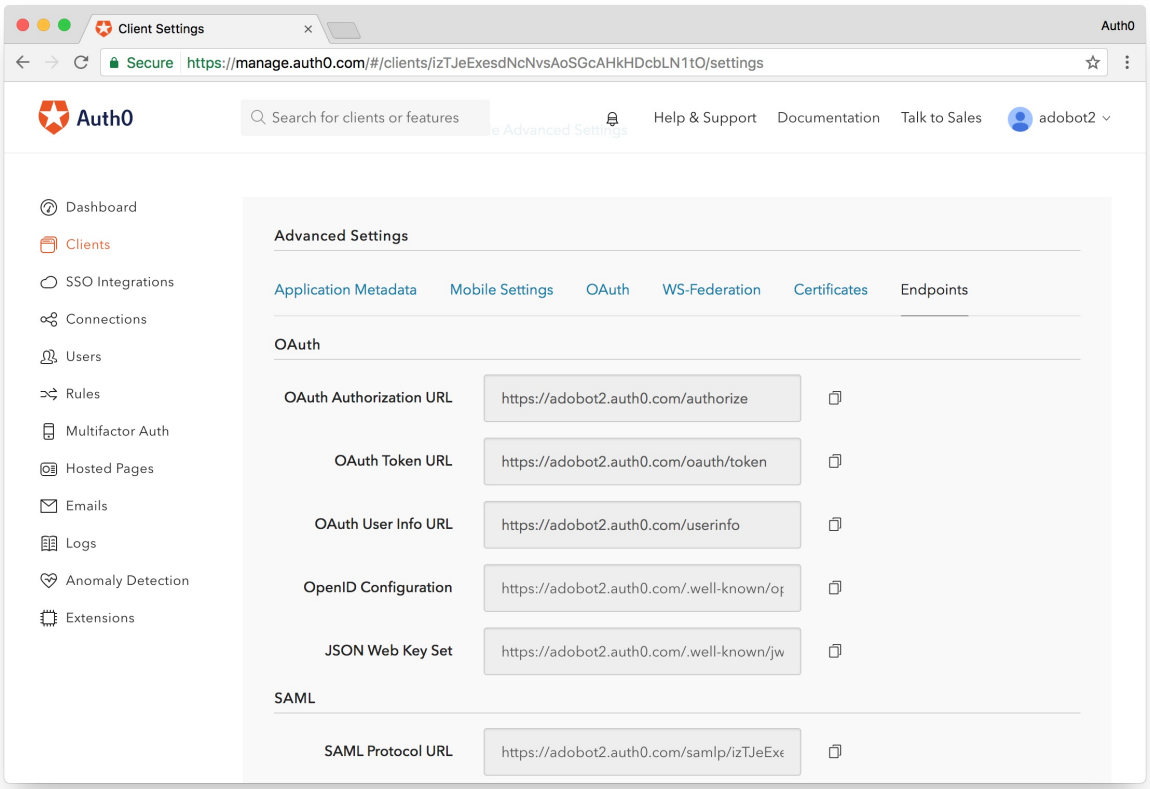
You can name your client whatever you want, I'll keep it descriptive and name the client *SSO Identity Provider*. Select any option for the **Client Type** and finally click the **Create** button.



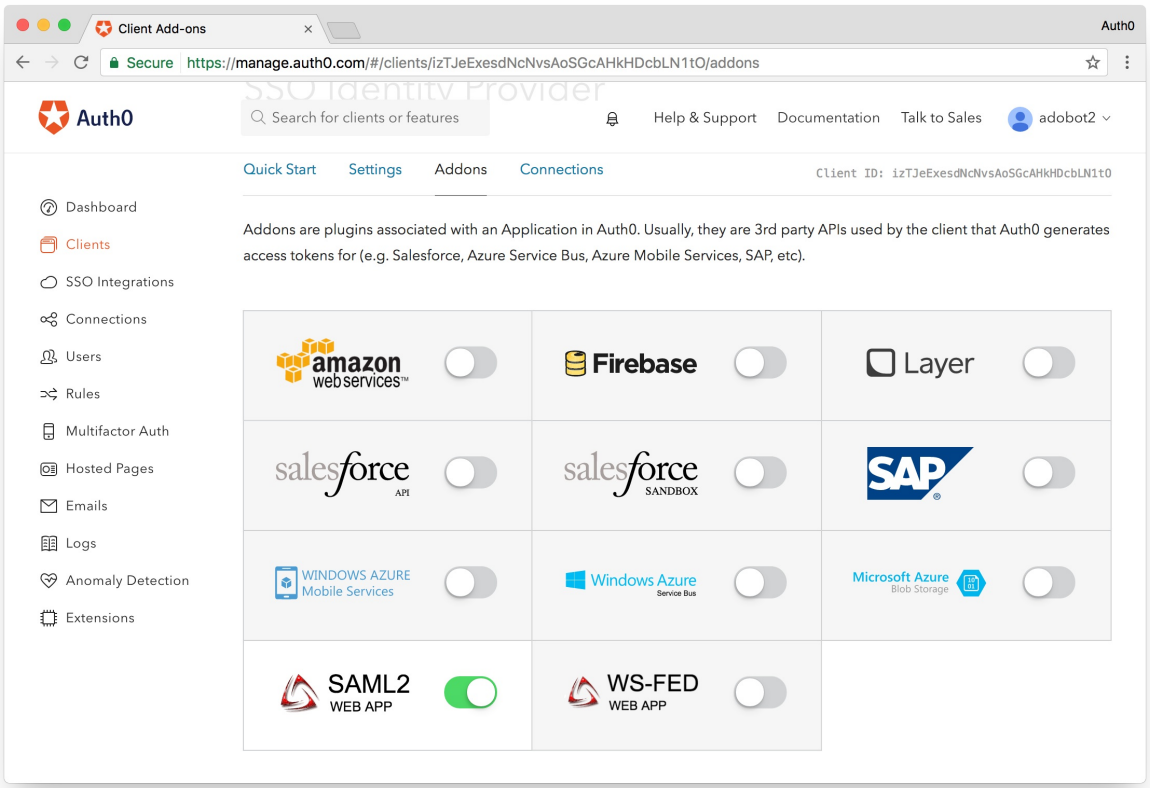
Clients in Auth0 are used for authentication. We'll need to make a few configuration tweaks to enable this client to act as our SSO IdP. The first thing we want to do is enable the **Use Auth0 instead of the IdP to do Single Sign On** switch, which can be found by navigating to the bottom of the client page. All you need to do is flip the switch and make sure it's green. Click the **Save Changes** button to save our change.



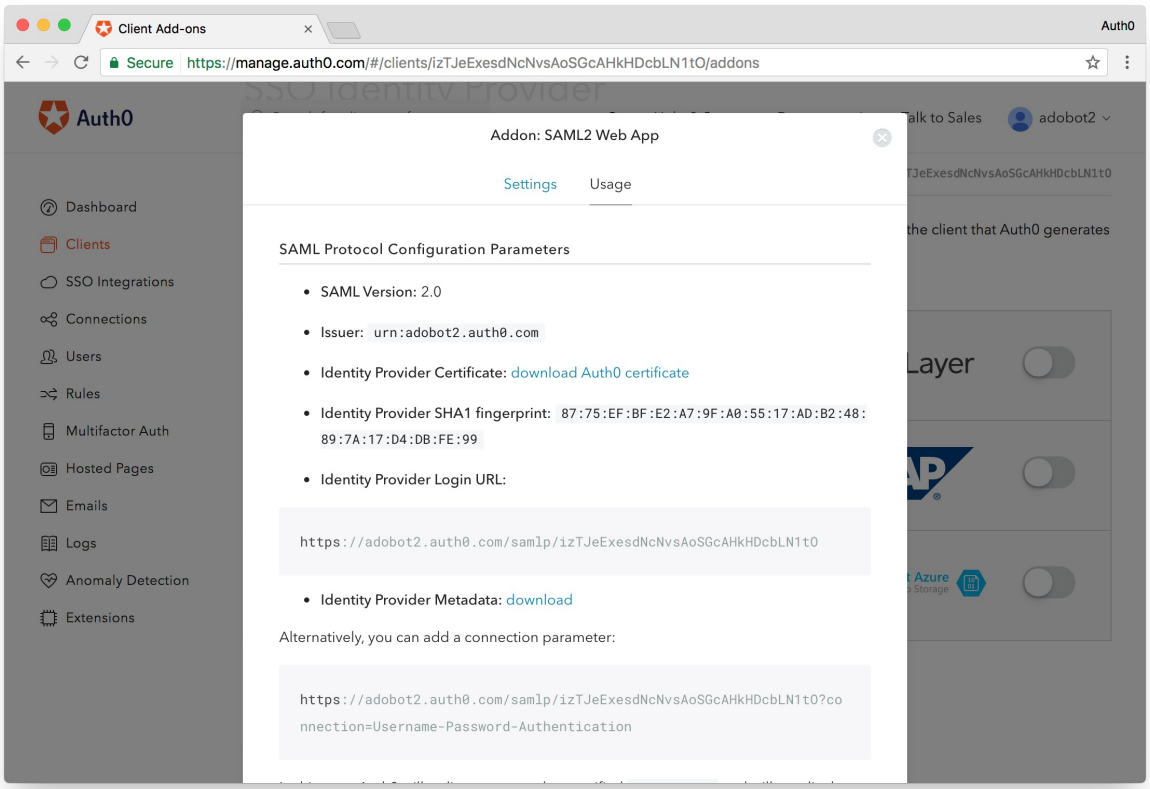
Next, we are going to export our endpoints for this connection. To do this, click on the **Show Advanced Settings** link, then navigate to the **Endpoints** tab. Here, you will see different endpoints that will allow you to interact with this client through various protocols. The one we care about is **SAML**. Copy the **SAML Protocol URL** as we will need it shortly when we configure our **SAML Connection.**



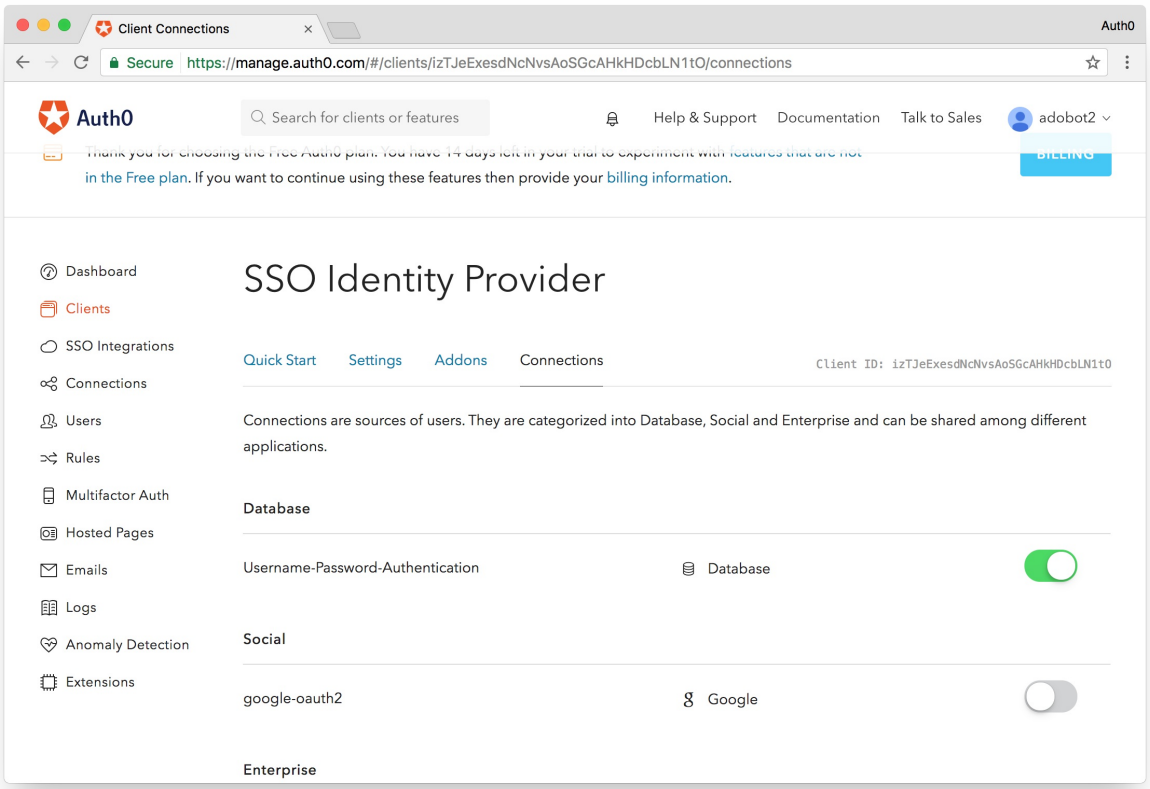
Our next order of business is to enable the SAML add on which will allow our client to handle and process SAML requests and issue responses. To do this, scroll to the top of the page of your client, and select the **Addons** tab. From here, select **SAML2**.



To enable the **SAML2** addon simply flip the switch and a modal dialog will popup prompting you to configure various options. For now, leave all the fields as-is and click on the **Usage** tab. You'll want to download the **Identity Provider Certificate** by clicking the link titled **download Auth0 certificate**. You can optionally download the identity provider metadata, but we won't be needing it here.

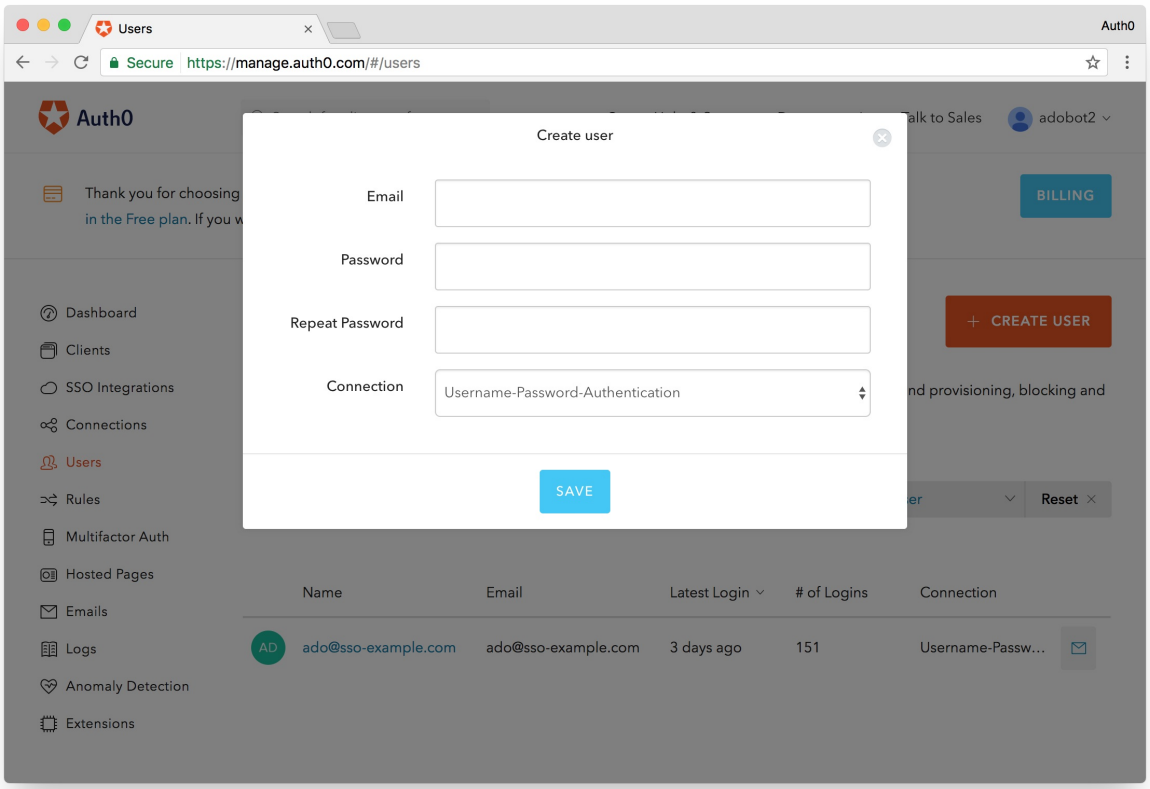


To conclude the section on setting up the Identity Provider we will need to setup how users will authenticate into this client. This is done by setting up a connection. When you created an Auth0 account, a **Username-Password-Authentication** connection was created by default. This connection allows a user to login using an email and password combination. Since we are setting up a generic SAML connection this will work great. To ensure that the client can be authenticated via the **Username-Password-Authentication** connection, open the **Connections** tab and make sure the **Username-Password-Authentication** connection switch is flipped to the on or green state.



Once you've done this, the last step is to create a user or users that will login to the organization. To create a new user, click on the **Users** tab in the main menu, and then select **Create User**. Add an e-mail, password, and ensure that the connection is set to **Username-Password-Authentication**. This will create a user and store them in the database allowing any Auth0 client which enables the **Username-Password-Authentication** connection in your account to login with the credentials you just created.

For the exercise, we are going to assume that the domain for this organization is vr-appstore-example.com, so when you create the user, make sure you give them that email domain extension. For example my user will be **ado@vr-appstore-example.com**. You will see why this is important soon.

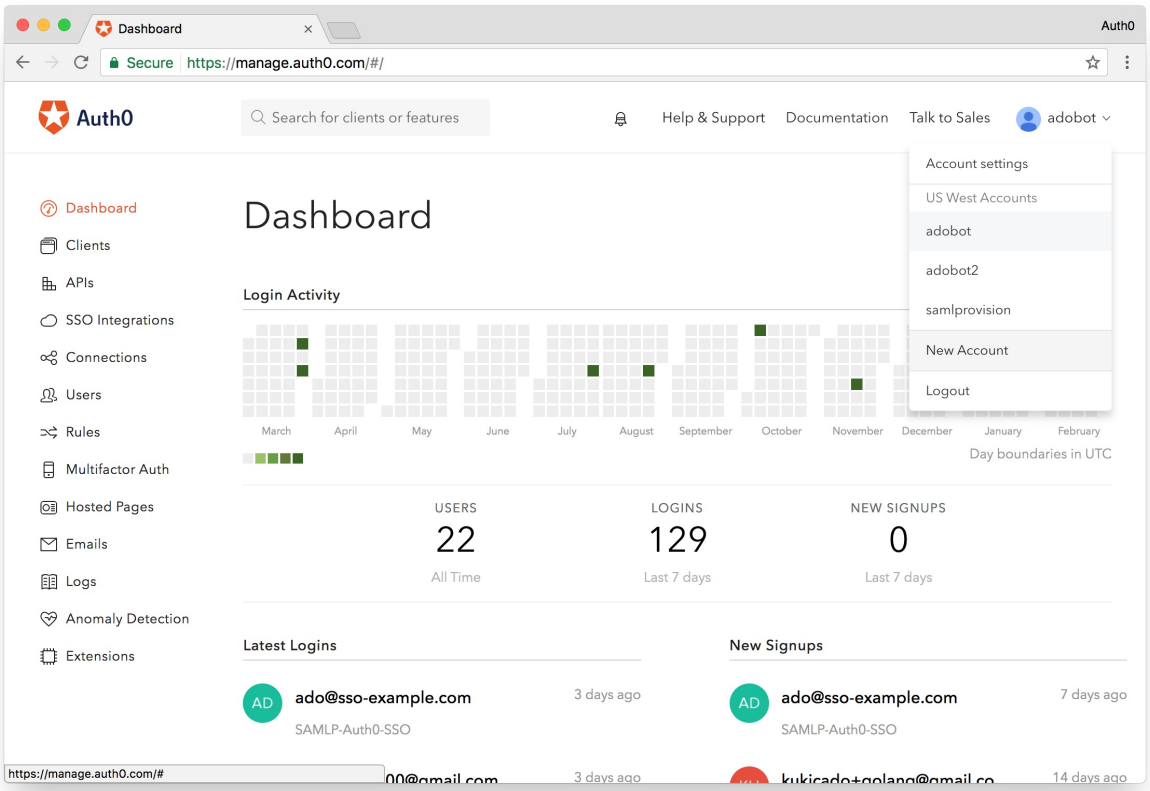


This will wrap up the section on setting up our identity provider. If you curious to learn more about the **Username-Password-Authentication** connection you can find it in the **Connections** -> **Database** section of the Auth0 dashboard. Here you can change various settings like disabling new sign ups, enforcing stricter password requirements, etc. We won't make any changes for this exercise.

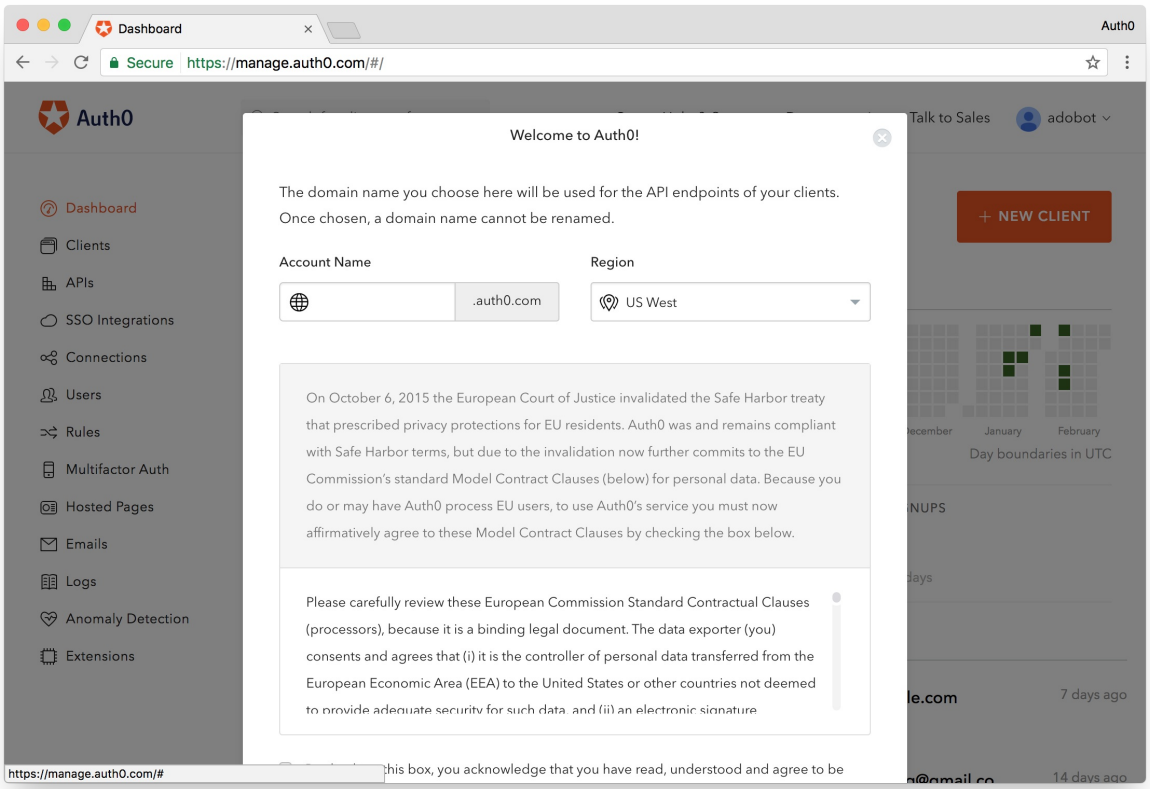
Setting Up Single Sign On

We have our Auth0 IdP setup. Next we are going to setup an Enterprise SSO connection that will connection to the IdP we just created. We are going to create a separate Auth0 account to do this. Luckily, you can create a new account direct from your dashboard and switch back and forth easily.

To create a new Auth0 account, click on your username in the top right corner and then click on the **New Account** button.

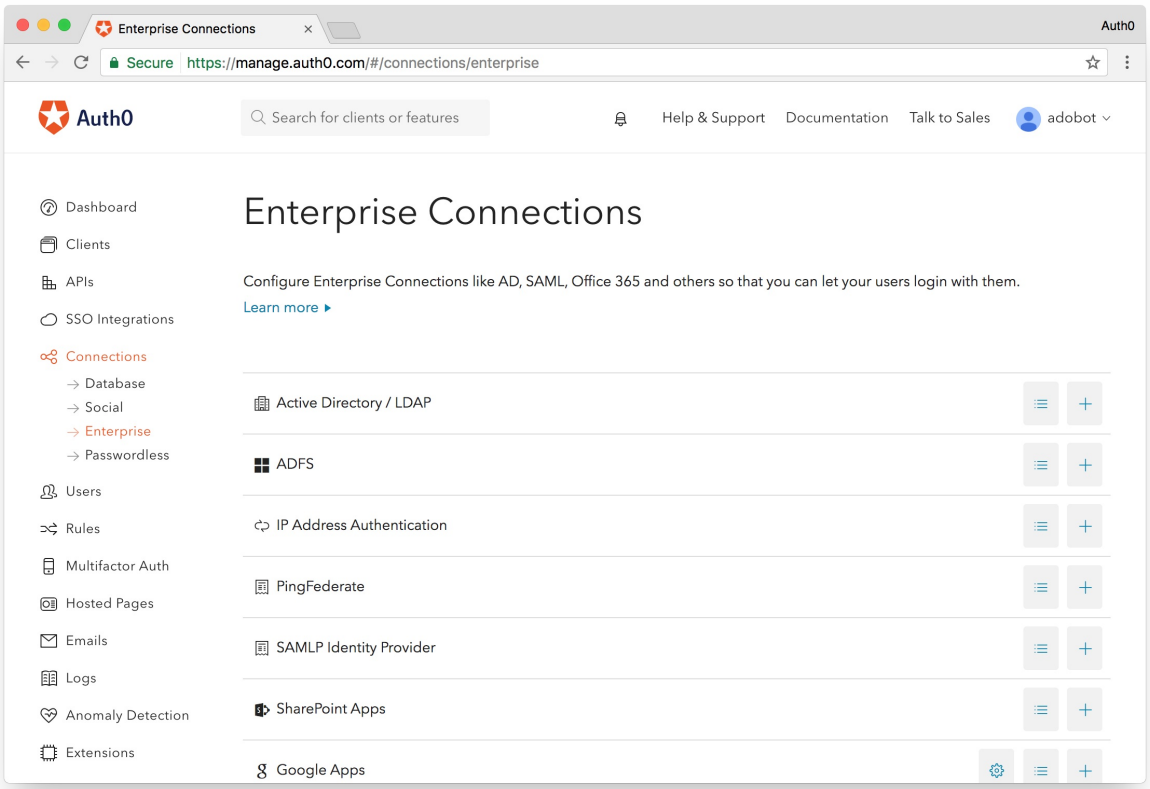


This will bring up a modal allowing you to name the new account and select it's region. I recommend creating the account in the same region as the previous one. Name the account whatever you want, and click **Create**.



Your new Auth0 account operates pretty much the same as the old one. The difference is, while with the previous account, we created a client that will act as our IdP, in this account, we will create clients that interact with our web and mobile applications. We'll also connect this second account to the first via an enterprise SAML connection.

We'll start off with creating a new connection. This Auth0 account, like the first one, created a default **Username-Password-Authentication** connection, but we do not want our users to authenticate in that manner, we want them to sign in via Single Sign On using the IdP client we created.



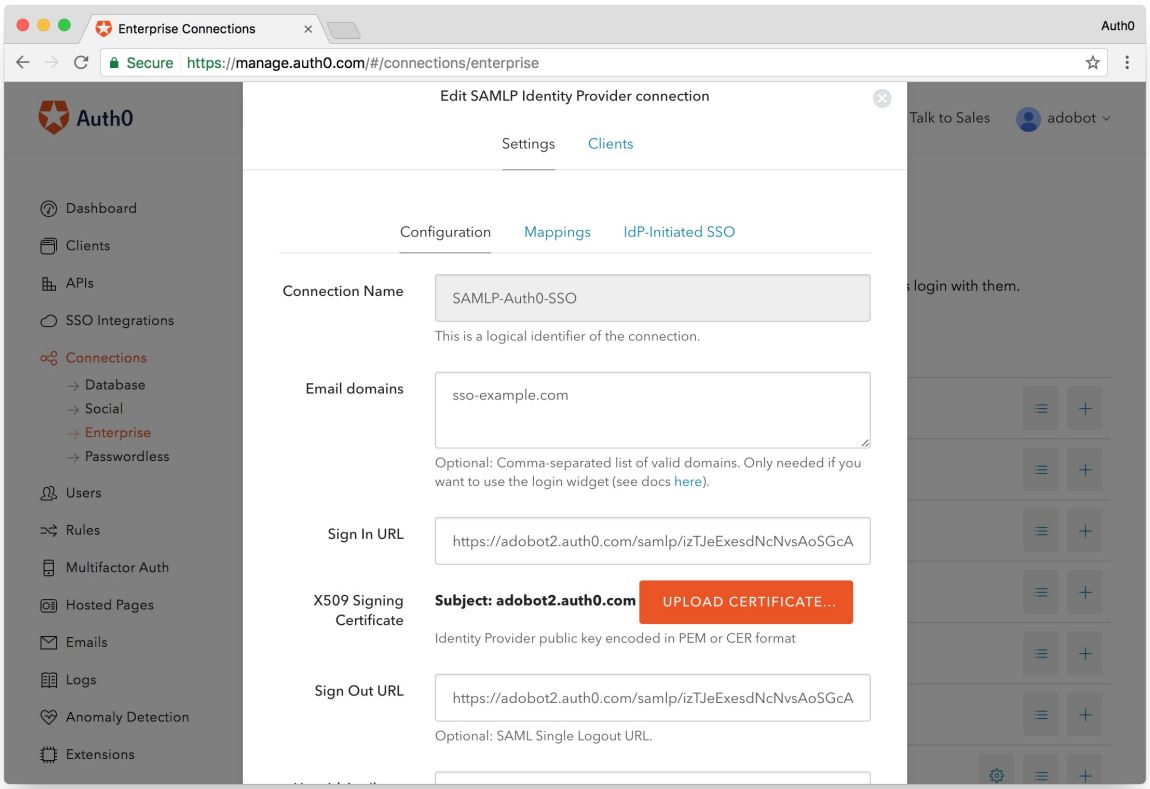
To create a new SAML connection, navigate to the **Connections** -> **Enterprise** section of the Auth0 dashboard. From here, find **SAMLP Identity Provider** and click on the **+** button to create a new SAML connection.

We will have to define a number of different settings here. First, the **Connection Name** can be anything. I've chosen **SAMLP-Auth0-SSO**, but you can feel free to name the connection how you see fit. Let's set the **Email domains** field to **@sso-example.com**, which as you may remember is the email domain of the user we created earlier. For the **Sign In URL**, you will want to paste the **SAML Endpoint** URL you copied from the client you created in the first account.

If you need to find the url, switch over to the first account, by clicking on the username, and selecting the name of the first account, navigate to the **Clients** tab, to the IdP Client you created, scrolling down and clicking on the **Show advanced settings** link, switching to the **Endpoints** tab and finally finding the **SAML Protocol URL** and copying it.

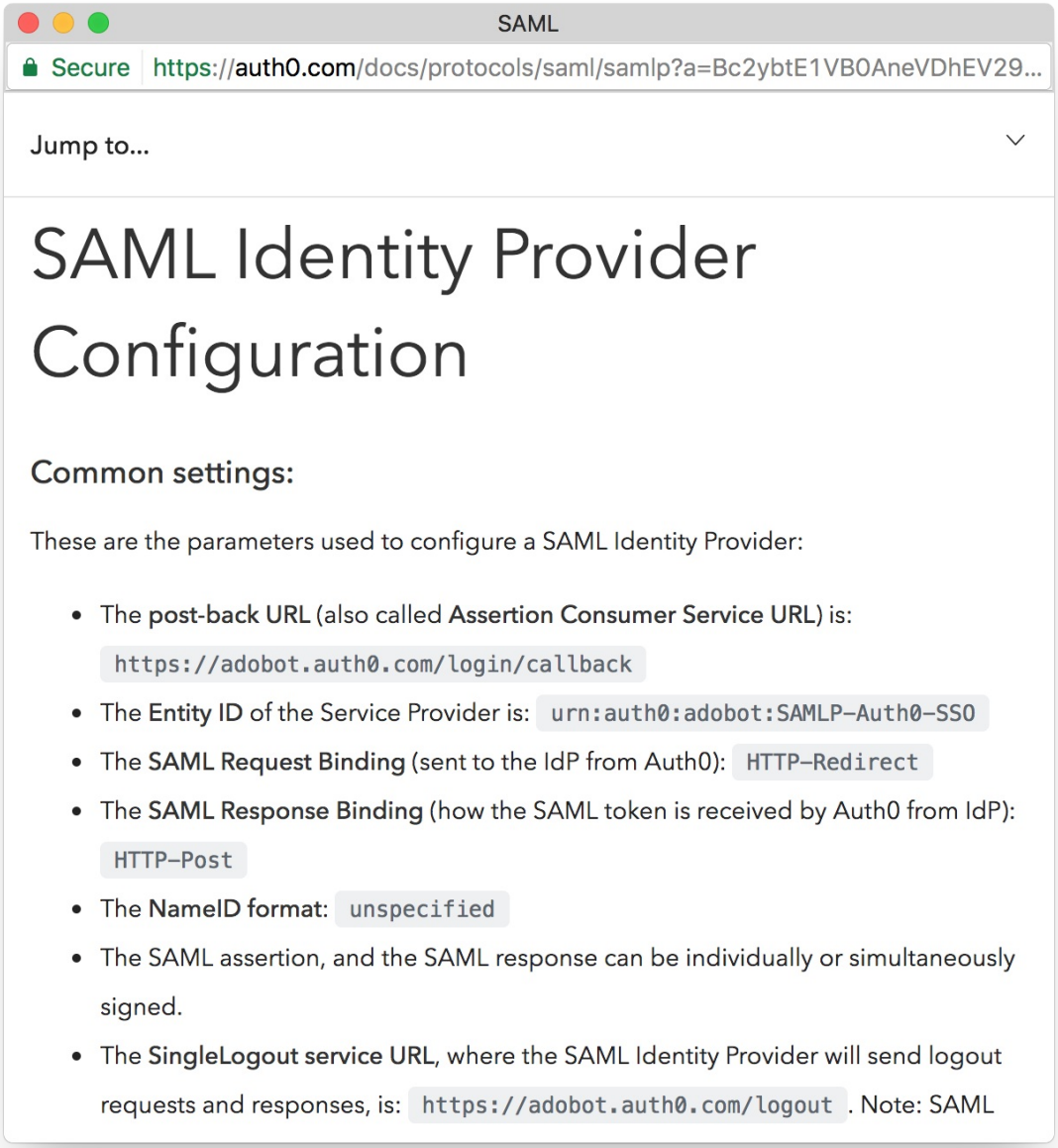
The **Sign Out URL** is going to be whatever the **Sign In URL** is with */logout* append to it. So if your **SAML Protocol URL** is *https://username.auth0.com/samlp/12345*, the **Sign Out URL** will be *https://username.auth0.com/samlp/12345/logout*.

Finally, the **X509 Signing Certificate** we already downloaded from our IdP client in the earlier section. Upload it here and we should be good to go. There are a number of other settings we can configure, but these are not required, so we'll omit them here. Scroll the the bottom of this modal and click the **Save** button.

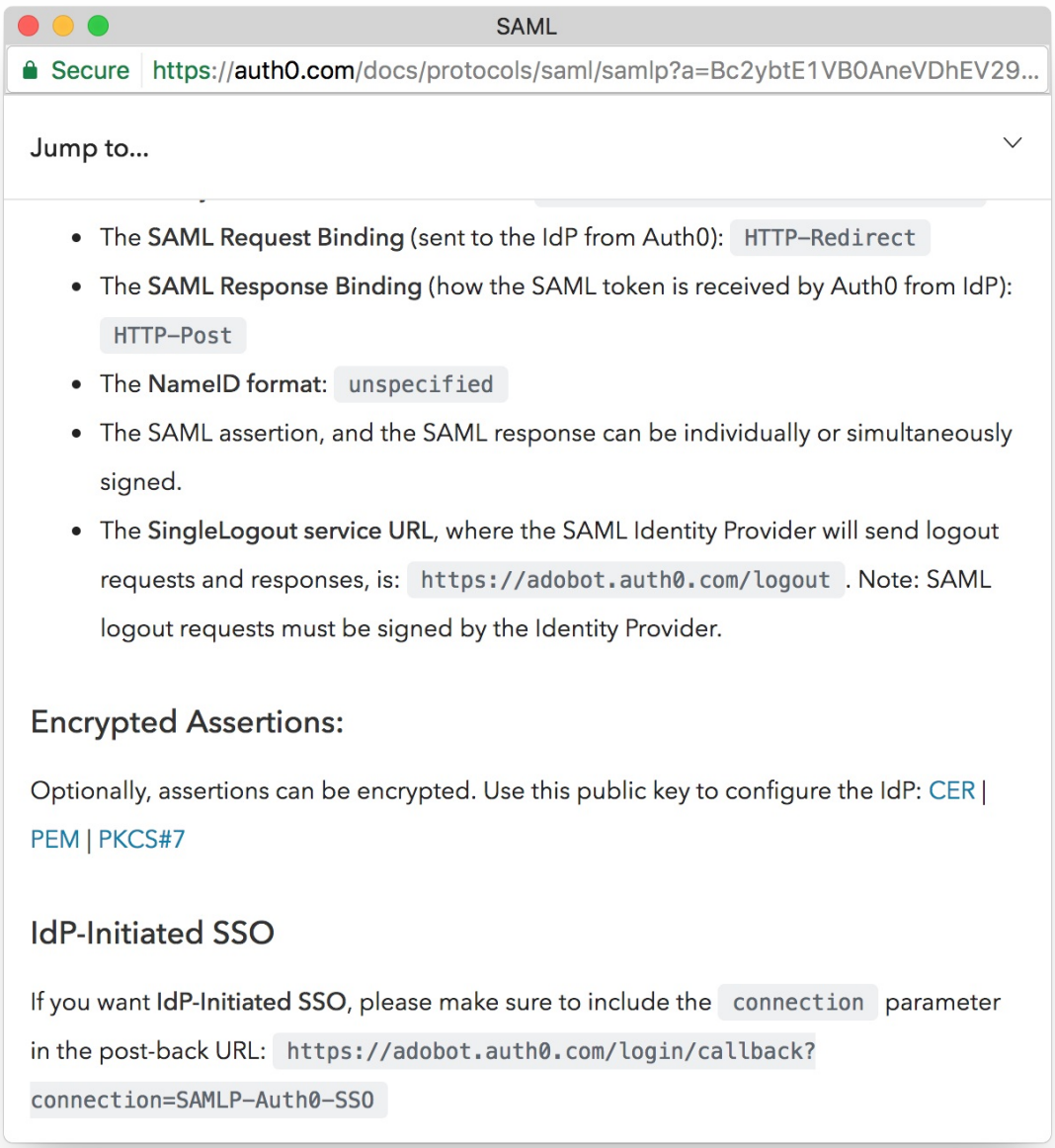


A new window will open up that will provide details on next steps for setting up the SSO connection. We'll have to add these changes in the first account. Let's switch back to the first account and go into our **SSO Identity Provider** or whatever you chose to name client.

The new window that opened up will have details we will need to provide to our IdP. It will look something like this:



The information that you'll want to copy here is the **Entity ID**. We also want to copy the **SingleLogout service URL** which will enable us to Single Logout out of all our applications at once. Since we are going to be using this connection for SSO, we'll also need to get the **IdP-Initiated SSO** value which can be found by scrolling down on this page to the IdP-Initiated section.



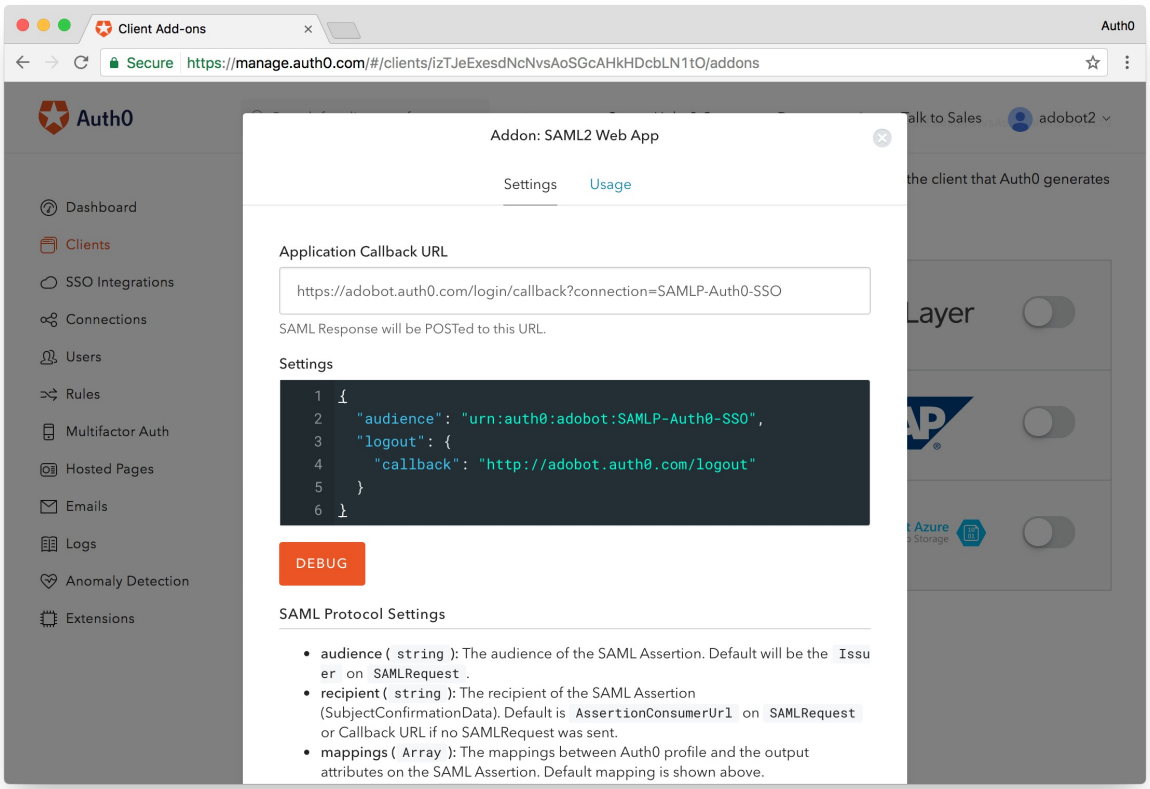
To successfully connect the SAML connection to the IdP, navigate to the **Addons** section for the IdP client (again in the initial account). The **SAML App** addon will already have been enabled, click it again so that we can add the values that we now have.

The completed settings should look like the screenshot below. For the **Application Callback URL,** we set the **IdP-Initiated SSO** url. In the settings, we are going to define two values. **Audience** will be the **Entity ID** we copied earlier. The second setting we'll define is a property **logout**, that will contain a property called **callback**, which we will set to to our **SingleLogout service URL**. The JSON should look like the following:

```
{
  "audience" : "urn:auth0:adobot:SAMLP-Auth0-SSO",
```

```
  "logout" : {
    "callback" : "http://adobot.auth0.com/logout"
  }
}
```

Remember to change the values for your own and scroll to the bottom to save the changes.



The final step here will be to add the **Application Callback URL** to the list of allowed callback URL's for the client. To do this in your IdP SSO client, go to the **Settings** tab and scroll until you find a textarea for **Allowed Callback URLs**. Simply paste the **Application Callback URL** and scroll the the bottom to save your change.

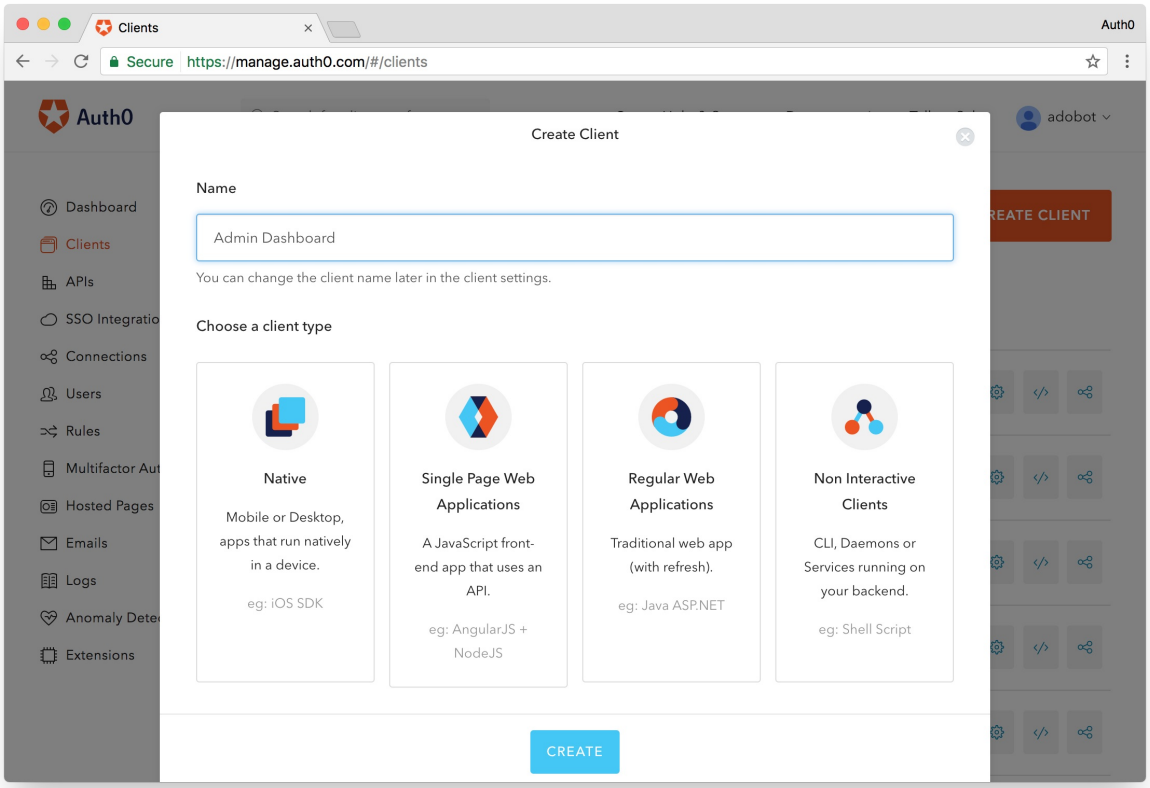
This completes the section on creating an enterprise SSO connection and connecting it to an IdP. Auth0 through separate accounts will act as both our Identity Provider and Service Provider. In the next section, we will implement Single Sign On in our applications.

Creating Application Clients

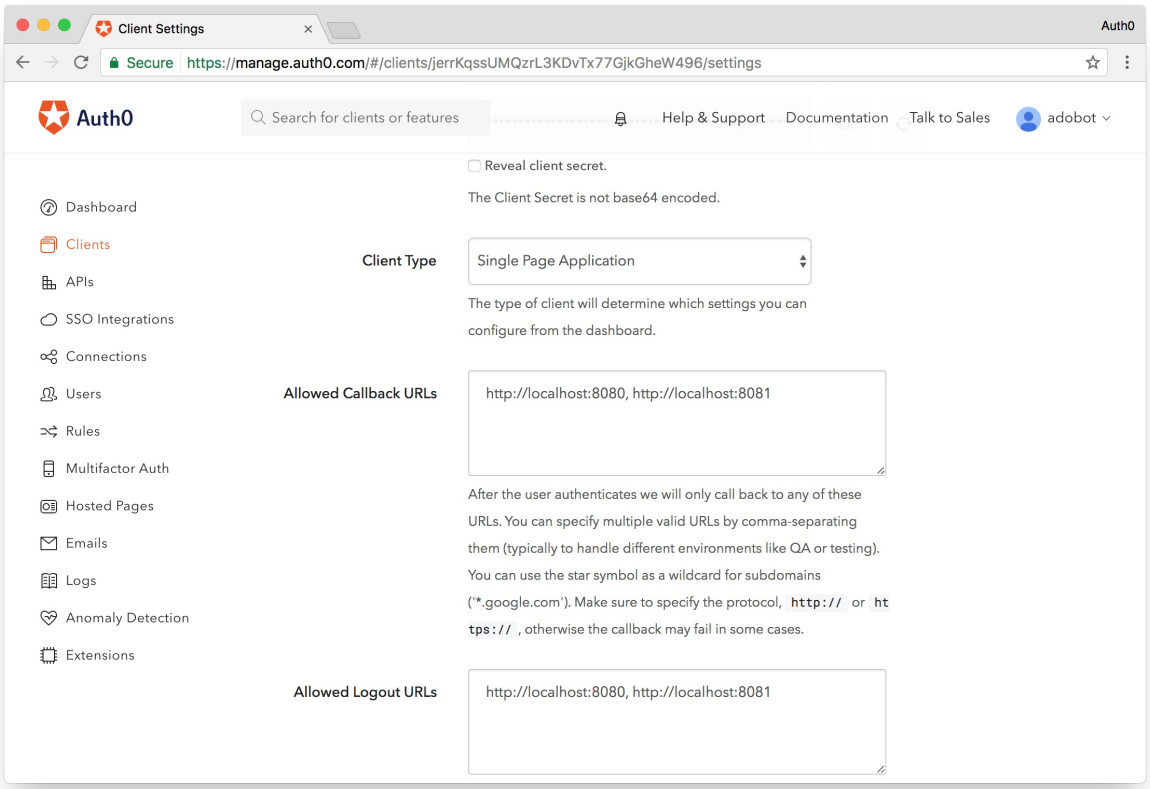
Before we can implement SSO into our applications, we need to have an Auth0 client that can talk to these applications. We will create two clients in our second Auth0 account, the one that has the SAML Enterprise connection setup, which we will be integrated into our applications.

The reason we are creating two clients is because one of the clients will only be used for internal users and employees, while the other will allow customers to sign up and login in addition to the employees of the organization. In a real world scenario you would create as many clients as you see fit for your needs.

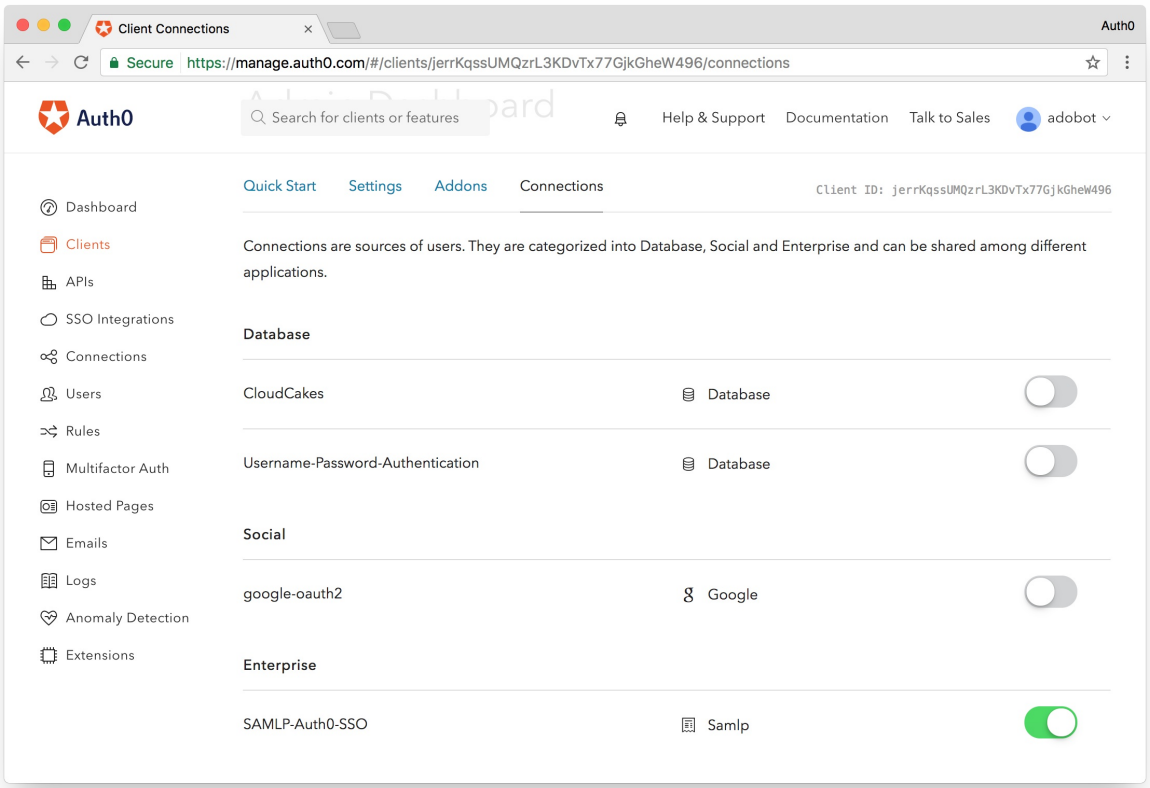
To create a new client, navigate to the **Clients** tab in your Auth0 dashboard and click on the **Create Client** button. Let's name the first client **Admin Dashboard**. For the **Client Type** we can choose **Single Page Web Applications**.



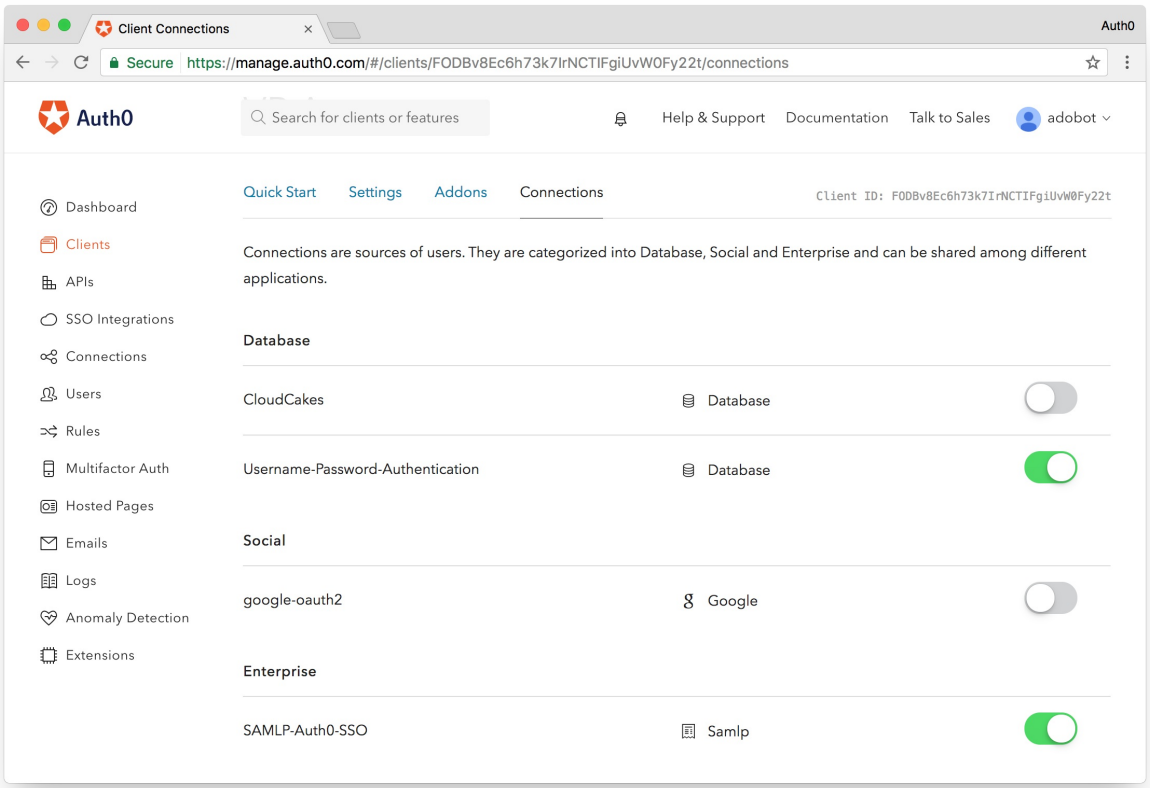
Once the client is created, we are going configure a few different settings. First we are going to set the **Allowed Callback URLs** and **Allowed Logout URLs** fields. For our exercise, our applications will only run locally, and the Admin Dashboard app will run on *http://localhost:8080*, while the HR app will run on *http://localhost:8081*. If you are planning on using different ports or URLs, make the changes here accordingly, otherwise the connection will not work properly.



The next thing we'll do is enable the SSO connection for the client. To do this, navigate to the **Connections** tab for the client and enable the **SAMLP-Auth0-SSO** or whatever you named the SAMLP connection. Since we want this client to only authenticate our employees, and thus SSO users, be sure to disable all of the other connections. Only the SSO switch should be green.



We are going to create a second client that will work with our VR Store application. This client, in addition to letting the SSO users login, will also allow users to login and signup via a database connection. Follow the same steps as above to create a new connection. Call this one VR Store. For the **Allowed Callback URLs** and **Allowed Logout URLs** set the value to *localhost:8082* as our VR App store will operate on *localhost:8082*. For this clients **Connections** enable the **SAMLP-Auth0-SSO** connection as well as the default **Username-Password-Authentication** connection.



To close out this section, go into both of the newly created clients and jot down the **Client ID** and **Domain**. We will be using these values in our applications to connect to Auth0. Make sure to remember which is which.

Implementing Single Sign On in Our Applications

As mentioned in the intro of this chapter, our fictional organization currently has three different applications. This is not uncommon and organizations typically have ten or more disparate apps that employees and customers use. For the exercise, we are going to keep it as simple as possible. Each application will have a default screen presented when a user is not logged in as well as a different screen presented when they are in fact logged in. Let's start with the **Admin Dashboard** application.

Pre-Requisite

We will have three different applications running locally. To serve these applications we'll use a simple NodeJS server. You can download and install NodeJS from https://nodejs.org, make sure the Node Package Manager (npm) is installed as well.

With NodeJS installed, run the command **npm install http-server -g**, to install a simple NodeJS server globally. Once http-server is installed you are ready to start developing the applications.

VR Store App

Our application will have two files. An **HTML** file that will contain the user interface and a **JavaScript** file that will contain the app logic and functionality. Create a new directory titled **app** and in here create two files: **index.html** and **app.js**. For the **index.html** file our code will look like:

```
<!DOCTYPE html>
<html>
  <head>
    <title>VR Appstore</title>
    <meta charset="utf-8">

    <!-- Import dependencies Auth0 Lock, Auth0 JS, and Bootstrap -->
    <script src="//cdn.auth0.com/js/lock/10.3.0/lock.min.js"></script>
    <script src="https://cdn.auth0.com/js/auth0/8.0.4/auth0.min.js"></script>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstra

    <!-- Load our app logic -->
    <script src="app.js"></script>

    <!-- Define some basic styles to pretty up our page -->
    <style>
      body { background: #DEDEDE; margin-top: 50px; }
    </style>
  </head>
  <body>

    <div class="container">
      <div class="col-sm-6 col-sm-offset-3">

        <!-- Display this UI if the user IS NOT logged in -->
        <div class="jumbotron text-center" id="logged_out" style="display: none;">
          <h1>VR App Store</h1>
          <button type="submit" id="btn-login" class="btn btn-block btn-primary">Sign In</but
        </div>

        <!-- Display this UI if the user IS logged in -->
        <div class="panel panel-default" id="logged_in" style="display: none;">
          <div class="panel-heading">
            <h3 class="panel-title">Welcome: <span id="nickname"></span></h3>
          </div>
          <div class="panel-body">
            <img alt="avatar" id="avatar" class="center-block">
```

```
            <br />
            <button type="submit" id="btn-continue" class="btn btn-sm btn-success center-bloc
          </div>
          <div class="panel-footer">
            <button type="submit" id="btn-logout" class="btn btn-sm btn-danger center-block">
          </div>
        </div>

      </div>
    </div>

  </body>
</html>
```

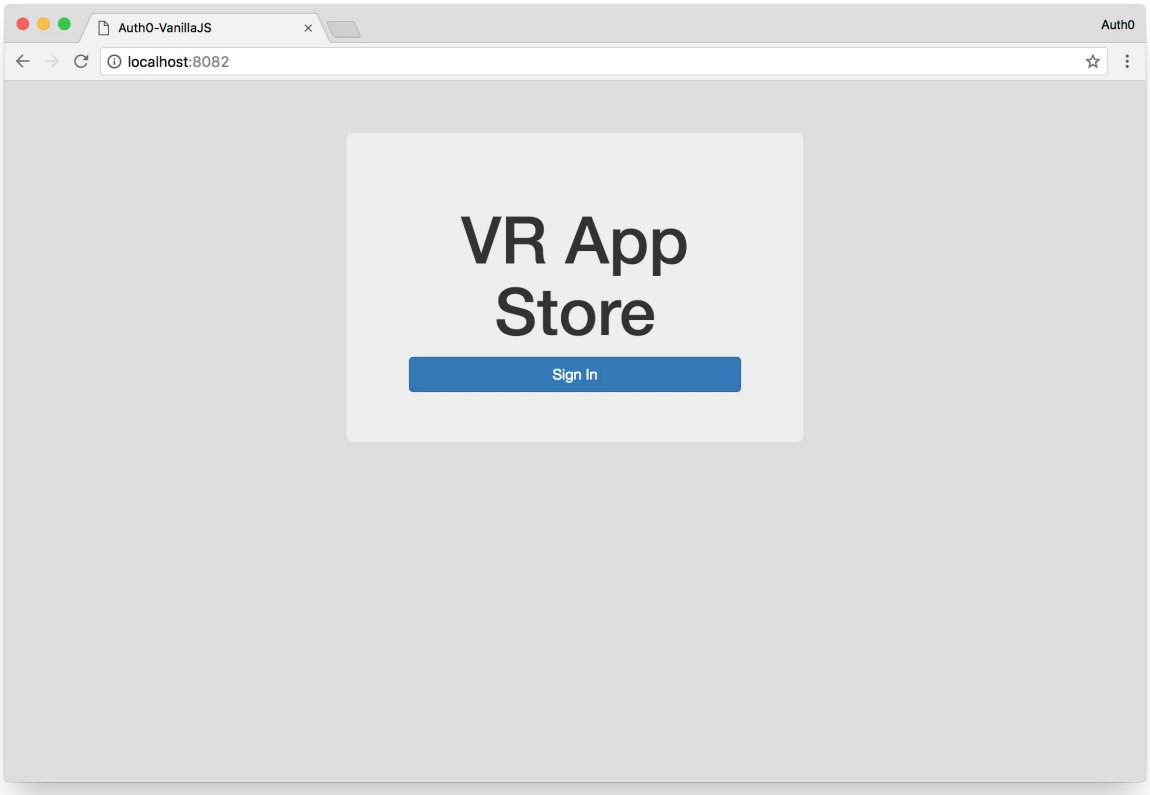
The HTML presented is fairly straightforward. We have two blocks that will be conditionally displayed depending on whether the user is logged in or not. To make this work, let's add the **app.js** logic.

```
window.addEventListener('load', function() {

  // Initialize Auth0 Lock, WebAuth and Auth0 libraries with our Client Credentials
  // For this application you will want to use the Client ID and Domain from the VR App Store
  var lock = new Auth0Lock('YOUR-CLIENT-ID', 'YOUR-DOMAIN');
  var webAuth = new auth0.WebAuth({domain:'YOUR-DOMAIN', clientID:'YOUR-CLIENT-ID'});
  var auth = new Auth0({domain:'YOUR-DOMAIN', clientID:'YOUR-CLIENT-ID'});

  // Set up listeners for our buttons in the UI
  var btn_login = document.getElementById('btn-login');
  var btn_logout = document.getElementById('btn-logout');
  var btn_continue = document.getElementById('btn-continue');
  logged_in.style.display = "none";
  logged_out.style.display = "block";

  // Display the Lock login widget when the user clicks the sign in button
  btn_login.addEventListener('click', function() {
    lock.show();
  });

  // Call the logout function when the user clicks the Logout button
  btn_logout.addEventListener('click', function() {
    logout();
  });

  // Display an alert when the user clicks on the continue button once they are authenticated
  btn_continue.addEventListener('click', function(){
    alert("Let's go shopping!");
  })
```

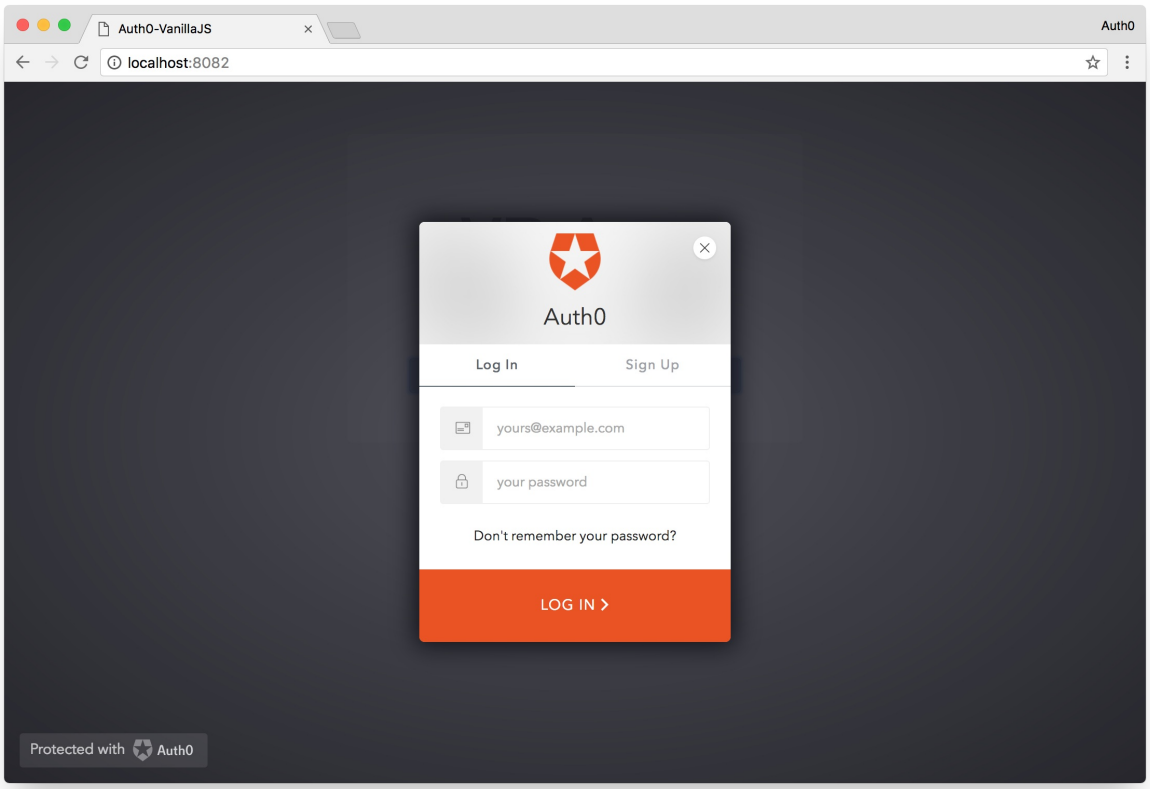
```
// Once a user is authenticated, log their profile information to the console and store the
lock.on("authenticated", function(authResult) {
  lock.getProfile(authResult.idToken, function(error, profile) {
    if (error) {
      // Handle error
      return;
    }
    console.log(profile);
    localStorage.setItem('id_token', authResult.idToken);
    // Display user information
    show_profile_info(profile);
  });
});

//retrieve the profile if the user is logged in.
var retrieve_profile = function() {
  var id_token = localStorage.getItem('id_token');
  if (id_token) {
    lock.getProfile(id_token, function (err, profile) {
      if (err) {
        return alert('There was an error getting the profile: ' + err.message);
      }
      // Display user information
      show_profile_info(profile);
    });
  }
};

var show_profile_info = function(profile) {
  var avatar = document.getElementById('avatar');
  document.getElementById('nickname').textContent = profile.nickname;
  btn_login.style.display = "none";
  avatar.src = profile.picture;
  btn_logout.style.display = "block";
  logged_in.style.display = "block";
  logged_out.style.display = "none";
};

// Logout the user and return to the domain.
var logout = function() {
  localStorage.removeItem('id_token');
  webAuth.logout({
    federated: true,
    returnTo: "http://localhost:8080"
  });
};

retrieve_profile();

// Watch to see if the user is still logged in. If the user has logged out from any SSO app
setInterval(function() {
  if (!localStorage.getItem('id_token')) return;
```

```
    auth.getSSOData(function (err, data) {
      // if there is still a session, do nothing
      if (err || (data && data.sso)){
        return;
      };

      // if we get here, it means there is no session on Auth0,
      // then remove the token and redirect to #login
      localStorage.removeItem('id_token');
      window.location.href = '/'

    });
  }, 2000)
});
```

With this code in place, we are ready to serve our application. Open up the console or terminal of your choice, navigate to the directory where you created these files, and run **http-server -p 8082**. This will create a local server and serve the files in the current directory on port 8082. Navigate to **localhost:8082** in your browser to see the Admin Dashboard app in action.

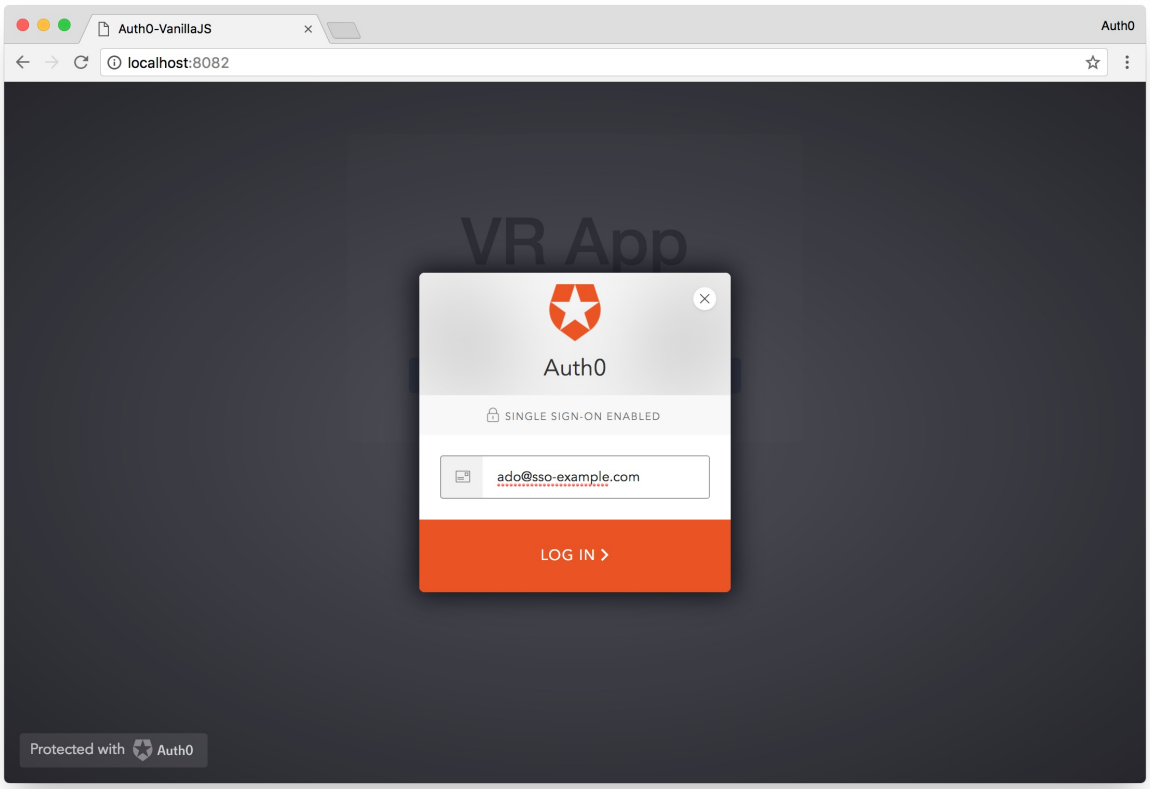


So far so good. Let's try logging in. Click on the **Sign In** button.

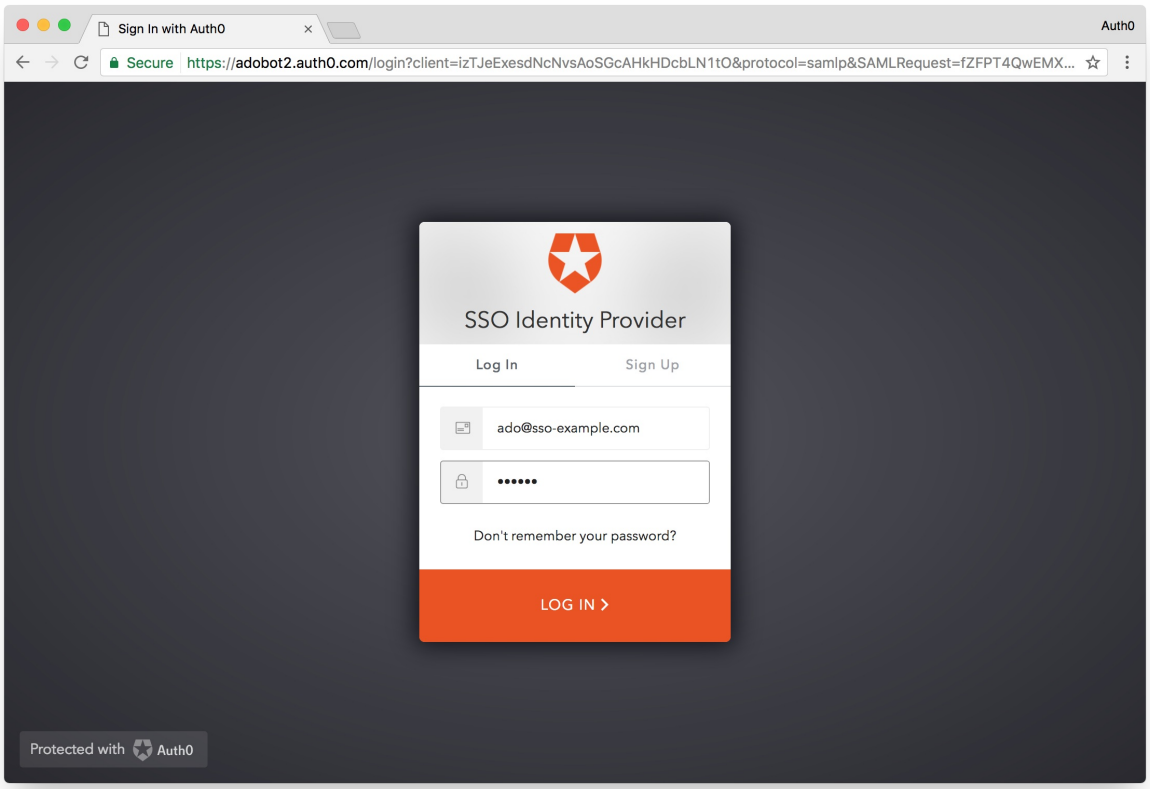


Clicking on the **Sign In** button brings up the standard Auth0 Lock login widget. The user is asked to enter their email and password and can sign up for an account if they don't already have one. This is the functionality we want as the VR App Store app is going to have both customers and employees logging in.

Let's see what happens when an employee attempts to login. Remember that I created a user with the email **ado@sso-example.com**. Let's try to login with that user.

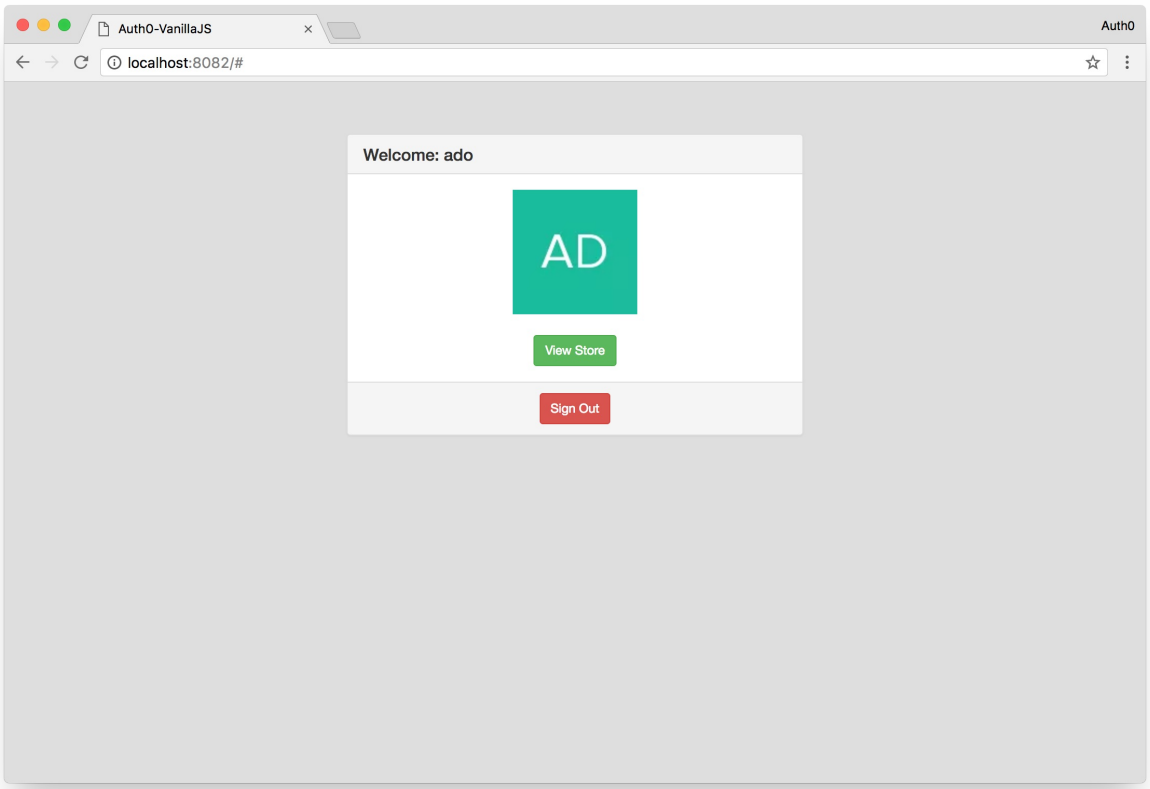


The Auth0 Lock widget recognizes the domain and instantly lets the user know that their account is Single Sign-On enabled. They still see the **Log In** button, but are no longer prompted for a password. Click on the **Log In** button.



The user is now redirected to the Single Sign On provider login screen. Since we are having Auth0 be both our IdP and Service Provider, we see the familiar Lock login widget, but if you look at the URL, you'll see that we've moved to a different domain, and the name of the provider is also displayed, in my case **SSO Identity Provider** which is the name I gave to my IdP client.

Let's login with the credentials we created for the user. Upon successfully entering the credentials, we are redirected back to the app and are now logged in. We see our username and can take further actions in the app.



The VR App Store app is now an SSO enabled application. The benefit of SSO may not be obvious yet, since we only have one application. Before we move on, click on the **Sign Out** button to Single Sign Out, again the benefit of this isn't obvious yet. This time, create a new account, by clicking on the **Sign In** button, and instead of putting in credentials, click the **Signup** tab and enter an email and password of your choice. This demonstrates that the VR App Store application will allow users to login and register, even if they are not coming through from SSO.

Admin Dashboard

Our Admin Dashboard application will for the most part be identical to the VR App Store app. Create a new directory titled **admin** and in here you can copy and paste the contents of the **app** application. The Admin Dashboard will also have two files an **index.html** and an **app.js** file. The **index.html** we'll modify to look like:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Admin Dashboard</title>
    <meta charset="utf-8">

    <!-- Import dependencies Auth0 Lock, Auth0 JS, and Bootstrap -->
```

```
    <script src="//cdn.auth0.com/js/lock/10.3.0/lock.min.js"></script>
    <script src="https://cdn.auth0.com/js/auth0/8.0.4/auth0.min.js"></script>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstra

    <!-- Load our app logic -->
    <script src="app.js"></script>

    <!-- Define some basic styles to pretty up our page -->
    <style>
      body { background: #DEDEDE; margin-top: 50px; }
    </style>
  </head>
  <body>

    <div class="container">
      <div class="col-sm-6 col-sm-offset-3">

        <!-- Display this UI if the user IS NOT logged in -->
        <div class="jumbotron text-center" id="logged_out" style="display: none;">
          <h1>Admin Dashboard</h1>
          <button type="submit" id="btn-login" class="btn btn-block btn-primary">Sign In</but
        </div>

        <!-- Display this UI if the user IS logged in -->
        <div class="panel panel-default" id="logged_in" style="display: none;">
          <div class="panel-heading">
            <h3 class="panel-title">Welcome: <span id="nickname"></span></h3>
          </div>
          <div class="panel-body">
            <img alt="avatar" id="avatar" class="center-block">
            <br />
            <button type="submit" id="btn-continue" class="btn btn-sm btn-success center-bloc
          </div>
          <div class="panel-footer">
            <button type="submit" id="btn-logout" class="btn btn-sm btn-danger center-block">
          </div>
        </div>

      </div>
    </div>

  </body>
</html>
```

The only change we made was the **title** tag in the **head**, the **h1** in the **jumbotron** and the action **button**. Next let's modify the **app.js** file.

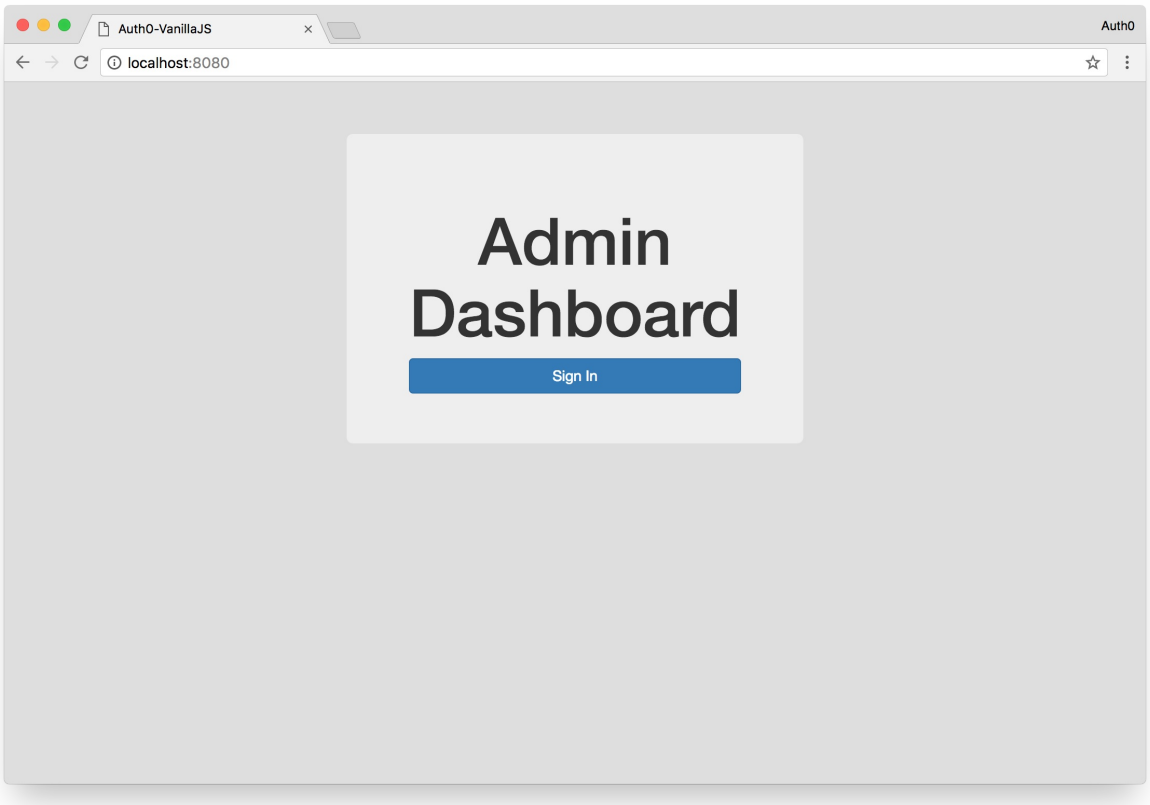
```
window.addEventListener('load', function() {
```

```
// Initialize Auth0 Lock, WebAuth and Auth0 libraries with our Client Credentials
// For this application you will want to use the Client ID and Domain from the Admin Dashbo
var lock = new Auth0Lock('YOUR-CLIENT-ID', 'YOUR-DOMAIN');
var webAuth = new auth0.WebAuth({domain:'YOUR-DOMAIN', clientID:'YOUR-CLIENT-ID'});
var auth = new Auth0({domain:'YOUR-DOMAIN', clientID:'YOUR-CLIENT-ID'});

// Set up listeners for our buttons in the UI
var btn_login = document.getElementById('btn-login');
var btn_logout = document.getElementById('btn-logout');
var btn_continue = document.getElementById('btn-continue');
logged_in.style.display = "none";
logged_out.style.display = "block";

// Display the Lock login widget when the user clicks the sign in button
btn_login.addEventListener('click', function() {
  lock.show();
});

// Call the logout function when the user clicks the Logout button
btn_logout.addEventListener('click', function() {
  logout();
});

// Display an alert when the user clicks on the continue button once they are authenticated
btn_continue.addEventListener('click', function(){
  alert("Time to get work done!");
})

// Once a user is authenticated, log their profile information to the console and store the
lock.on("authenticated", function(authResult) {
  lock.getProfile(authResult.idToken, function(error, profile) {
    if (error) {
      // Handle error
      return;
    }
    console.log(profile);
    localStorage.setItem('id_token', authResult.idToken);
    // Display user information
    show_profile_info(profile);
  });
});

//retrieve the profile if the user is logged in.
var retrieve_profile = function() {
  var id_token = localStorage.getItem('id_token');
  if (id_token) {
    lock.getProfile(id_token, function (err, profile) {
      if (err) {
        return alert('There was an error getting the profile: ' + err.message);
      }
      // Display user information
      show_profile_info(profile);
```

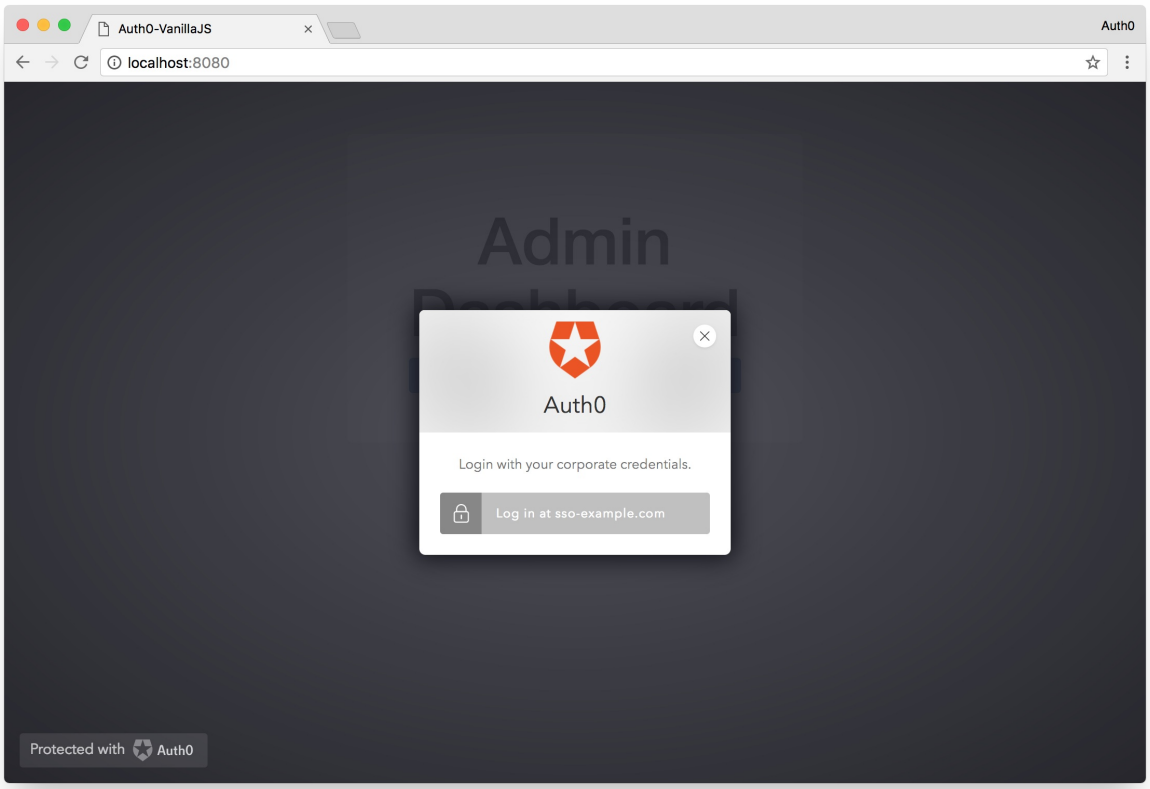
```
      });
    }
  };

  var show_profile_info = function(profile) {
    var avatar = document.getElementById('avatar');
    document.getElementById('nickname').textContent = profile.nickname;
    btn_login.style.display = "none";
    avatar.src = profile.picture;
    btn_logout.style.display = "block";
    logged_in.style.display = "block";
    logged_out.style.display = "none";
  };

  // Logout the user and return to the domain.
  var logout = function() {
    localStorage.removeItem('id_token');
    webAuth.logout({
      federated: true,
      returnTo: "http://localhost:8080"
    });
  };

  retrieve_profile();

  // Watch to see if the user is still logged in. If the user has logged out from any SSO app
  setInterval(function() {
    if (!localStorage.getItem('id_token')) return;

    auth.getSSOData(function (err, data) {
      // if there is still a session, do nothing
      if (err || (data && data.sso)){
        return;
      };

      // if we get here, it means there is no session on Auth0,
      // then remove the token and redirect to #login
      localStorage.removeItem('id_token');
      window.location.href = '/'

    });
  }, 2000)
});
```

The only change we are making is that we are changing the **Client ID** from the VR App Store client, to the Admin Dashboard client from our Auth0 dashboard. To launch this application, open up a new console or terminal of your choice, navigate to the directory

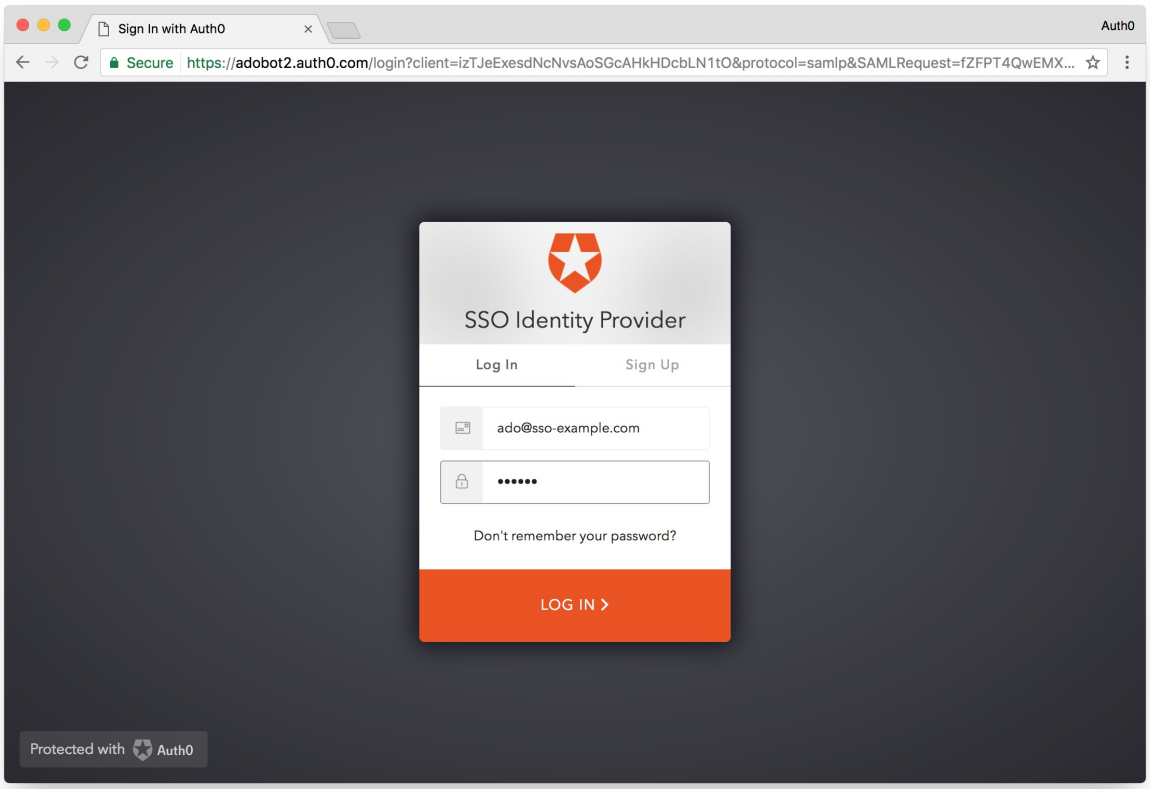
where you created these files, and run **http-server -p 8080**. Navigate to **localhost:8080** in your browser to see the Admin Dashboard app in action.



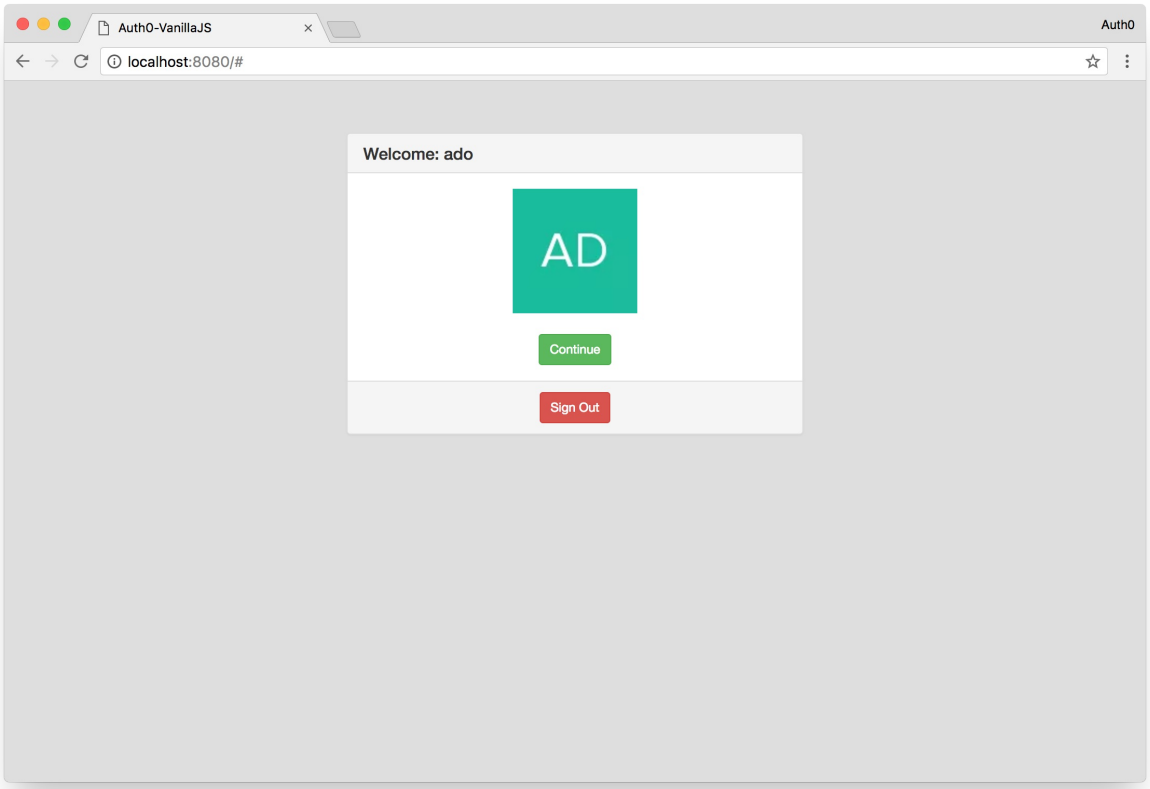
So far so good. Let's try logging in. Click on the **Sign In** button.



Clicking the **Sign In** button opened up a modal dialog using the Auth0 Lock widget. Since we configured this client to only work with the SSO connection, we don't see the option to input an email and password. We just see a button telling us to login with our corporate credentials and a button to login. Clicking this button will take us to the familiar SSO Identity Provider login screen that we saw when we logged in via SSO in the VR App Store application.



Let's login here, again with the account you created in the Auth0 dashboard, and we'll be able to see the logged in view for the Admin Dashboard.



We have now authenticated via SSO into the Admin Dashboard. How does this affect the VR App Store? Let's find out. Navigate to **localhost:8082** and you will see the logged out experience for the VR App Store app. Click on the **Sign In** button, and enter the email for your SSO user. Click **Log In**. The app will redirect to the SSO login page, but only momentarily, since you are already authenticated, you will not be required to provide any credentials, you will simply be redirected back to **localhost:8082** and will be logged in.

This is the magic of Single Sign On. The user is only required to actually provide their credentials once, and they are logged in to all the different applications. They only have to remember one set of credentials but with that one set of credentials they can access as many applications as the IT admin grants them (and as long as they are setup to work with SSO).

HR App

Finally, we are also going to quickly build the third application, the HR Application where employees can request time off and perform other managerial tasks. This app for all intents and purposes will be very similar to the Admin Dashboard. Create a new directory titled **hr** and copy and paste the **index.html** and **app.js** files from the **admin** directory.

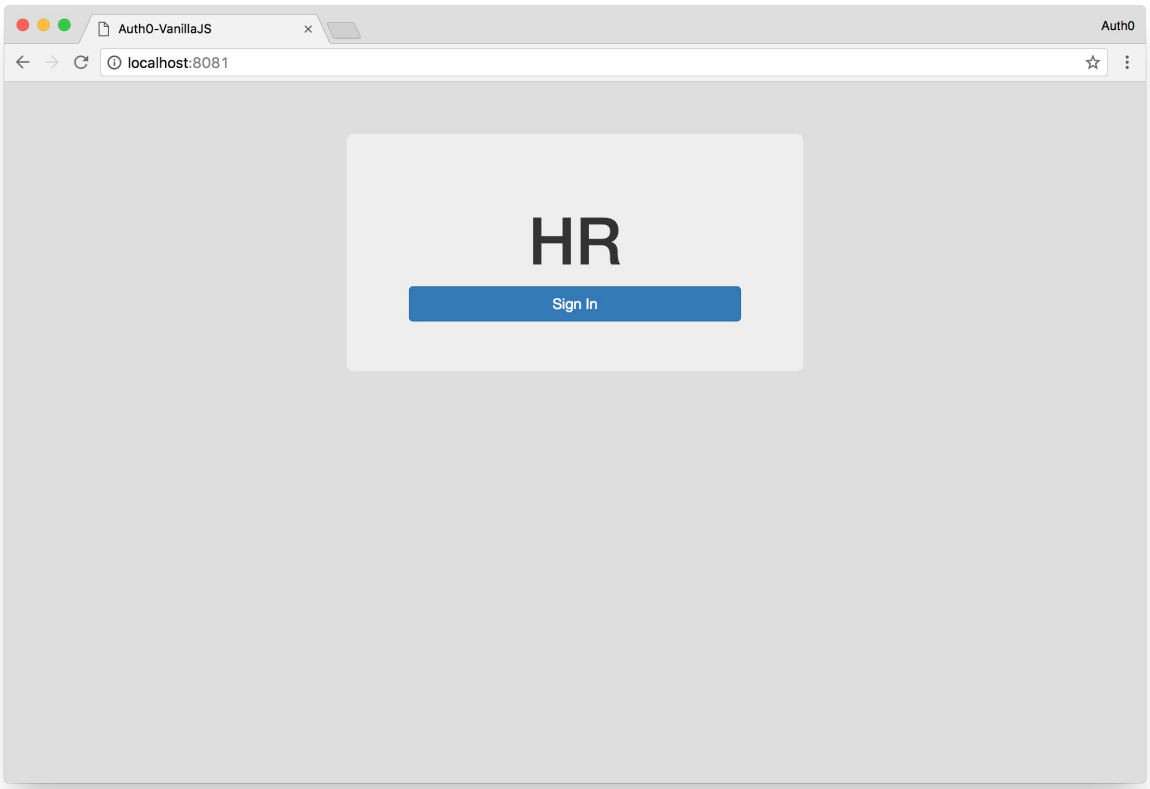
For posterity we will omit redundant code and only show the changes. For **index.html**:

```
...
 <title>Human Resources</title>
...
 <h1>Human Resources</h1>
...
 <button type="submit" id="btn-continue" class="btn btn-sm btn-success center-block">>Reques
```

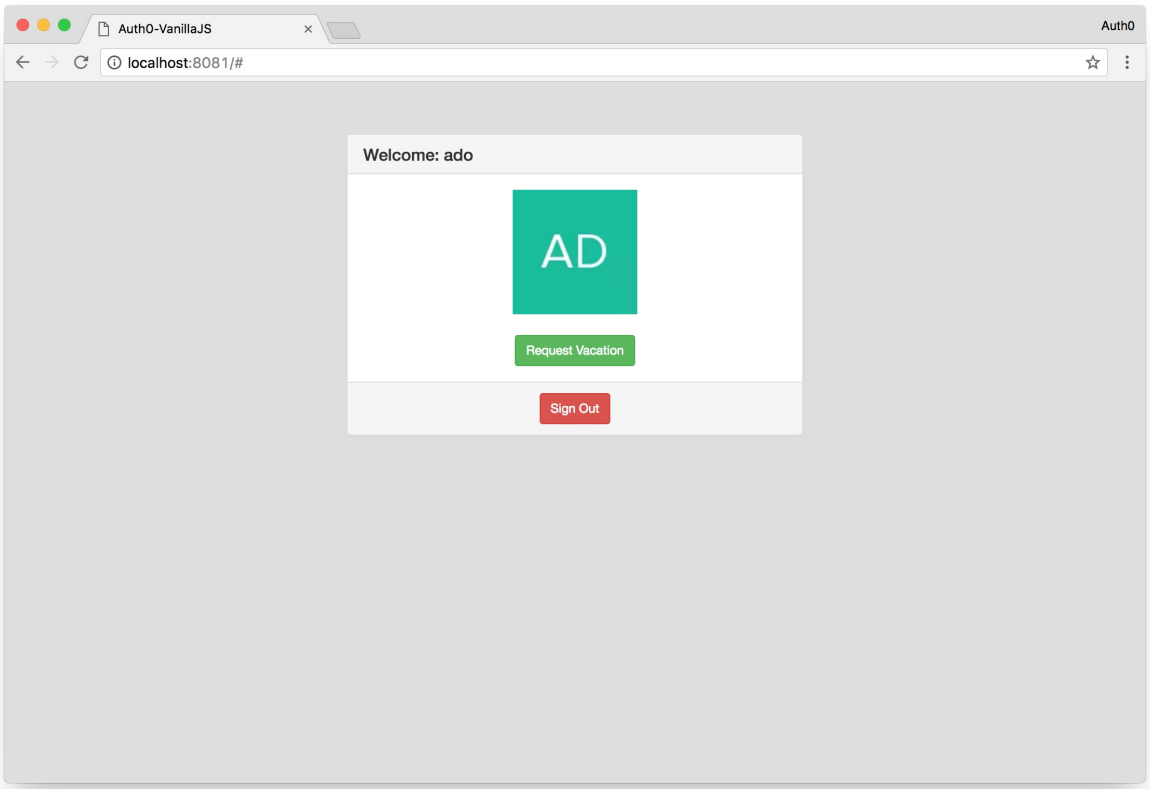
In the **app.js**, we'll only change the message for when a user requests a vacation, as the **Client ID** is identical to that of the Admin Dashboard app.

```
...
 alert("See you in 2 weeks! :)");
...
```

Let's start this application up by running **http-server -p 8081** and navigating to **localhost:8081** in our browser.



Clicking on the **Sign In** button will predictably display the **Login with your corporate credentials** option. Let's login to the HR app with our SSO user. We'll get a familiar logged in screen.



Now navigate to **localhost:8080**, which is the Admin Dashboard, and click the **Sign In** button. Since you are already logged in via SSO, you will simply be logged in to the admin dashboard. No extra steps required. Go to **localhost:8082** and login as well. Since this app is consumer facing as well, you'll need to enter the SSO email and click **Log In** but will again be logged in to the application without having to provide a set of credentials.

Click the **Sign Out** button on either of the three apps and you will be signed out of all three automatically.

The last functionality I want to explain is if you log in to the VR Store App, at **localhost:8082**, through a non-SSO email and password. When you do this, the user account is retrieved from the **Username-Password-Authentication** database in the second Auth0 account you created, not in the same database where the Single Sign On users reside. Likewise, logging in with this user will not grant you access to the other applications. If you were to SSO in say the HR app, and then logout, you would not be automatically logged out of the VR Store App, since the accounts are controlled by different connections.

A Word on Single Logout

In this exercise we covered not only Single Sign On, but also Single Logout. Single Logout is a powerful feature of SSO that allows a user to log out of all the applications they are logged into by simply logging out of one of the applications. This is a feature of SSO and you do not have to implement it in your implementation of SSO if you don't want that functionality. Popular services like Google employ Single Logout so that when you log out of GMail, you are automatically logged out of YouTube and all the other Google services as well.

Conclusion

In this chapter we put theory to practice by implementing Single Sign On across a fictional organization. We set up our own Identity Provider, a SAML SSO connection, and various clients that interacted with each other to save employees of the VR App Store startup time as well as enforce good security practices by centralizing identity.

I hope that this chapter was able to show you the benefit of SSO and got you thinking about how you could implement Single Sign On in your organization. The example we went through used Auth0 for both the IdP and Service Provider, but you do not have to use Auth0 for either if you don't want to. There are many different solutions to implementing SSO but they all strive for the same goal which is to make identity management easier and more secure.

Conclusion and Next Steps

Single Sign On is a very complicated topic but increasingly important. We've covered a lot in this book, but we've just scratched the surface. The goals of this book were:

- To help you learn and understand what Single Sign On is and how it works
- Understand the role Single Sign On plays in modern identity management
- Learn about the various Identity Protocols used in conjunction with SSO
- Understand the benefits of adding Single Sign On as a premium offering to customers of your apps
- And finally implement Single Sign On in a guided tutorial

Across the five chapters, I believe we've met our goals. You should now know what Single Sign On is and how it works. The benefits of Single Sign On and federated identity discussed are applicable for organizations of all sizes. There are many different implementations of Single Sign On and finding the right solution for your organization is important. Our discussion of the different protocols brought insight on the different permutations of SSO. Depending on your use case, you might need to support more than one implementation of SSO. Our use cases section focused on when to use what type of Single Sign On implementation for your application. Depending on your type of business you may need to support one or more types of SSO. Finally, we put theory to practice and showed how SSO works by implementing it across a fictional organization which had multiple apps that otherwise would have required a separate set of credentials each.

For next steps, if this book has convinced you or at least got you thinking about how Single Sign On can benefit your organization, why not give it a try. There are many vendors, like Auth0, offering SSO services that can be enabled with just the flip of a switch.