How does authentication work?

Identity theft is not a joke, Jim

Justin
Mar 15, 2021

TL;DR

Authentication is how the apps you use **know that you are who you say you are**: through something as simple as a password, or as complex as biometrics.

- Accessing your apps and data on the web relies on identity: **only you** should be able to see your stuff!
- The most popular authentication method is **username and password**, but schemes like **single-sign-on** and **magic links** are getting more popular
- On the backend, authentication can work through **cookies**, **session tokens**, and/or other combinations of esoteric cryptography

Pretty much every app developer has to build authentication if they want people to use what they're making. Auth0 – which lets developers outsource authentication – just exited for $6.5B (!). So the space is probably worth understanding.

Why user accounts exist in the first place

If you're using an app or service, you probably need to create an account and log in. But what's going on behind the scenes when you type in your username and password? What does it mean to "log in with Google?" And how do developers actually build this stuff?

Authentication is just the process of **verifying that someone is, indeed, who they say they are** (is?). In the context of an app, it means creating an account, and then when you log in, proving that you are the person who created said account. It's worth taking a step back to talk about why authentication, or honestly just user accounts, needs to exist.

Most apps you used are customized for you - your emails in Gmail, your timeline on Twitter, and your spreadsheets in Google Drive and *yours*. These apps associate specific data with you - who you follow, the words you've muted, etc. - and apply those when you log in. Without an account, the concept of following someone on Twitter doesn't really make sense - because there's no "you" (in the practical sense, of course. Whether the self really exists is beyond the scope of this newsletter).

In practice, when you follow someone on Twitter, Twitter stores that in a database somewhere - you can imagine some sort of table that looks like this:

user	follows_user	follow_date
justin	elonmusk	1/1/2020
justin	grimes	1/2/2020

This is obviously an oversimplification, but in general this is how apps work - the data gets stored in a database, and gets queried when you load different pages. Once I'm logged in as justin, Twitter can query this table, know who I follow, and show in my browser that I already follow them. My entire experience on Twitter is powered by data that's specific to me, and that's why I need an account.

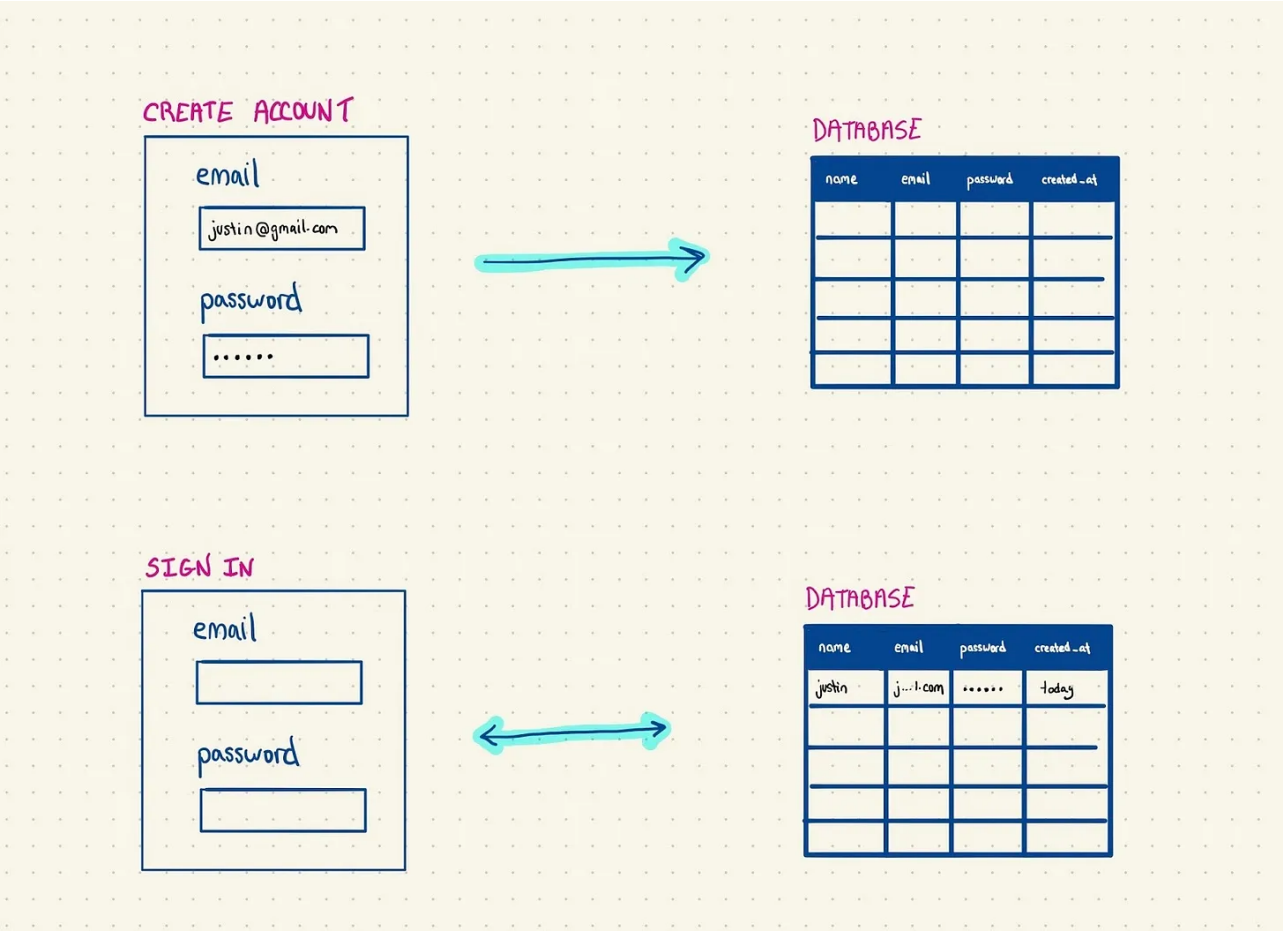
How authentication works on the backend

Now that we understand why user accounts exist, the next step is **protecting access** to those accounts. If you've got financial information saved (e.g. your Amazon account), the ability to create (incriminating) content (e.g. Twitter), or anything else important that exists in your account, it's in your best interest to make sure that you and only you can access it.

The most common implementation of this kind of security check is a username and password. The basic steps here:

1. When you create an account, your username (or email) and password get stored in a database

2. When you try to log in after that, you're prompted to enter your username and password - if they match what's in the database, you're good to go



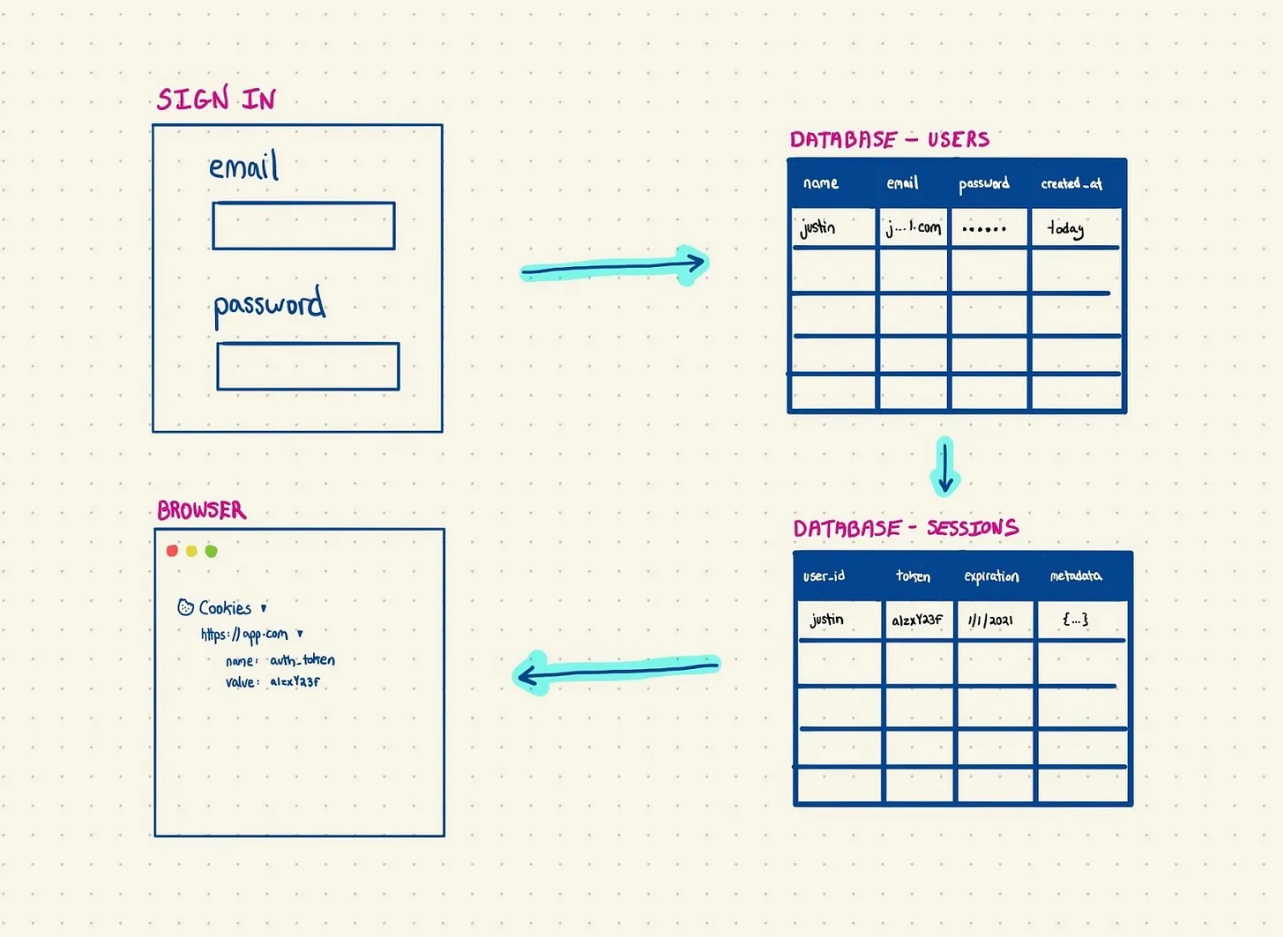
It's pretty basic when you think about it - what makes things complicated, as always, are the details. But that's what you're here for.

Sessions, cookies, and tokens (oh my)

One thing you've probably noticed is that once you log in, you *stay logged in* for a while - depending on the app, it could even be months before you need to enter your credentials again. The period you're logged in for without being asked to re-authenticate is typically called a **session**.

The most popular way of building this is via **browser cookies**. You've probably heard about these before (tbh idk why they're called cookies) - they're just data attached to your browser. If the app you're using is running authentication via cookies, when you log in, two things happen:

- On the server (database), the app created a “token” - just a really long string of random characters – and stores it along with your user information
- On the client (your browser), the app sets a cookie with that token attached



Every time you load a page that needs to verify you are you, the app will try to compare the token in the cookie to the token that exists in the database. If they match, you’re good to go.

After a set amount of time (a few days, weeks, whatever) the app will *destroy* the token on the server, and probably the cookie in the browser as well - and then, without tokens verifying you’re legit, you’ll need to log in again, and this whole thing gets restarted.

> **🧪 Try this at home 🧪**
>
> You can see your browser cookies. Just open your browser’s inspector (right click, then “inspect”), find “application” and then you should see a tab for

> cookies. If you click on one, you'll see a list of data - each entry has a name, a value, and an expiration date (plus a few other things).
>
> **🧪 Try this at home 🧪**

There's a bit of a balance to strike here - on the one hand, needing to log in all the time is really f*ckn annoying. But the longer you let session cookies persist, the more vulnerable your account is - if someone gets access to your computer, they'll also have access to your accounts, even without needing to log in.

> **🔍 Deeper Look 🔍**
>
> Another approach to keeping sessions live is called **token based authentication**. Instead of the server creating a token and a cookie and matching them, the app signs a token via cryptography (different kind of token) when you sign in. One of the common ways this is done is via JSON Web Tokens.
>
> **🔍 Deeper Look 🔍**

Single Sign On

Over the past few years, a lot of our favorite apps have started offering options that let you log in with Google, Facebook, Github, and now, even Apple. This is called Single Sign On (SSO for short), and while it's generally a feature reserved for larger enterprises and their identity providers (like Okta), it can also make things easier for the common folk like us. So how does it work?

It's actually really simple - if the point of authentication is just to prove that you are who you say you are so that an app can show you your data, you can **outsource** that to other services who *already know your identity*. When you log into Google and verify that you are, indeed, the owner of "coolboy69@gmail.com" you can then *bring that identity with you* elsewhere.

More concretely, when you head over to Dropbox and click "login with Google," you get redirected to Google's site, where you'll log into your Google account. Once you've successfully authenticated, you get redirected back to Dropbox - and since your Google account is associated with a specific email, Dropbox now

knows that you are who you say you are. They still need to create an account for you on the backend and all of that jazz, but this greatly simplifies the work a developer needs to do to implement auth.

I wrote more about SSO on the WorkOS blog if you want a deeper dive (heads up - it's for a developer audience).

New methods for the cool kids

Usernames and passwords kinda suck. For two main reasons:

1. **Ease of use** – having to create and remember a username and password is annoying. Especially when you forget your password.
2. **Security** – passwords are easily hackable, especially if the apps you use aren't careful about encryption.

The ecosystem is starting to get wise to this, and we're seeing tons of new companies popping up that offer password-less alternatives. A few popular methods:

- **Magic links** – verify your identity through email or SMS instead of a password (e.g. Magic.link).
- **SSO** – covered above. Outsource identification to a trusted third party, like Google (e.g. WorkOS).
- **Authenticator apps** – use apps like Google Authenticator or Authy to generate one-time codes to sign in.
- **Push** – authenticate through push notifications to your phone or computer (e.g. Stytch).
- **Biometric** – biological authentication, like TouchID and FaceID.

A lot of these are faster *and* more secure than passwords, which is a win-win for everyone.

Final thing to note: more and more companies are popping up that let developers **outsource** authentication instead of building it themselves. The best example here – and now, super valuable exit – is Auth0, recently acquired by Okta. Auth0 (on a basic level) just does all of the work for you – providing nice

APIs for authenticating, as well as a basic UI for entering a username/password or connecting via SSO to something like Google.

But beyond Auth0, there are some **great new options** around: Firebase (actually kinda OG), Supabase, Stytch, Magic.link, and many others. And companies like WorkOS operate a level higher, making it easy for developers to build SSO into their apps too.

5 Comments

Write a comment...

ER-Software Nerd Apr 6, 2021

Great stuff Justin, I am wondering if you have an opinion on what the Okta purchase of Auth0 means for the company, would you expect the sales power of Okta to drive awareness and adoption of customer identity tools like Auth0 for the market or will Auth0's developer-focused culture clash with Okta? Also curious if you think that Identity as a stand-alone cloud offering can rise to a "primary cloud" on par with say, Infrastructure, CRM, collaboration, ERP, etc

♡ 3 Reply Gift a subscription Collapse •••

2 replies by Justin and others

S. Nakamoto Mar 16, 2021

Great article, Justin! Worth mentioning the future direction of authentication that is currently materializing: decentralised identities, self-sovereignty identity, dEvine identity... Check tide.org!

♡ 2 Reply Gift a subscription Collapse •••

3 more comments...