FusionAuth

The Modern Guide to OAuth

EBOOK

The Modern Guide to OAuth

Brian Pontarelli and Dan Moore

This book is available at
https://leanpub.com/themodernguidetooauth

This version was published on 2025-03-11

Contents

Introduction

I know what you are thinking, is this really another guide to OAuth 2.0?

Well, yes and no. This guide is different than most of the others out there because it covers all of the ways that we actually use OAuth. It also covers all of the details you need to be an OAuth expert without reading all the specifications or writing your own OAuth server. This guide is based on hundreds of conversations and client implementations as well as our experience building FusionAuth, an OAuth server which has been downloaded over a million times.

If that sounds good to you, keep reading!

OAuth overview

OAuth 2.0 is a set of specifications that allow developers to easily delegate the authentication and authorization of their users to someone else. While the specifications don't specifically cover authentication, in practice this is a core piece of OAuth, so we will cover it in depth (because that's how we roll).

What does that mean, really? It means that your application sends the user over to an OAuth server, the user logs in, and then the user is sent back to your application. However, there are a couple of different twists and goals of this process. Let's cover those next.

OAuth modes

None of the specifications cover how OAuth is actually integrated into applications. Whoops! But as a developer, that's what you care about. They also don't cover the different workflows or processes that leverage OAuth. They leave almost everything up to the implementer (the person who writes the OAuth Server) and integrator (the person who integrates their application with that OAuth server).

Rather than just reword the information in the specifications (yet again), let's create a vocabulary for real-world integrations and implementations of OAuth. We'll call them **OAuth modes**.

There are eight OAuth modes in common use today. These real world OAuth modes are:

1. Local login and registration
2. Third-party login and registration (federated identity)
3. First-party login and registration (reverse federated identity)
4. Enterprise login and registration (federated identity with a twist)
5. Third-party service authorization
6. First-party service authorization
7. Machine-to-machine authentication and authorization
8. Device login and registration

I've included notation on a few of the items above specifying which are federated identity workflows. The reason that I've changed the names here from just "federated identity" is that each case is slightly different. Plus, the term federated identity is often overloaded and misunderstood. To help clarify terms, I'm using "login" instead. However, this is generally the same as "federated identity" in that the user's identity is stored in an OAuth server and the authentication/authorization is delegated to that server.

Let's discuss each mode in a bit more detail, but first, a cheat sheet.

Which OAuth mode is right for you?

Wow, that's a lot of different ways you can use OAuth. That's the power and the danger of OAuth, to be honest with you. It is so flexible that people new to it can be overwhelmed. So, here's a handy set of questions for you to ask yourself.

- Are you looking to outsource your authentication and authorization to a safe, secure and standards-friendly auth system? You'll want Local login and registration in that case.
- Trying to avoid storing any credentials because you don't want responsibility for passwords? Third-party login and registration is where it's at.
- Are you selling to Enterprise customers? Folks who hear terms like SAML and SOC2 and are comforted, rather than disturbed? Scoot on over to Enterprise login and registration.
- Are you building service to service communication with no user involved? If so, you are looking for Machine-to-machine authorization.
- Are you trying to let a user log in from a separate device? That is, from a TV or similar device without a friendly typing interface? If this is so, check out Device login and registration.
- Are you building a platform and want to allow other developers to ask for permissions to make calls to APIs or services on your platform? Put on your hoodie and review First-party login and registration and First-party service authorization.
- Do you have a user store already integrated and only need to access a third party service on your users' behalf? Read up on Third-party service authorization.

With that overview done, let’s examine each of these modes in more detail.

Local login and registration

The **Local login and registration** mode is when you are using an OAuth workflow to register or log users into your application. In this mode, you own both the OAuth server and the application. You might not have written the OAuth server (if you are using a product such as FusionAuth), but you control it. In fact, this mode usually feels like the user is signing up or logging directly into your application via **native forms** and there is no delegation at all.

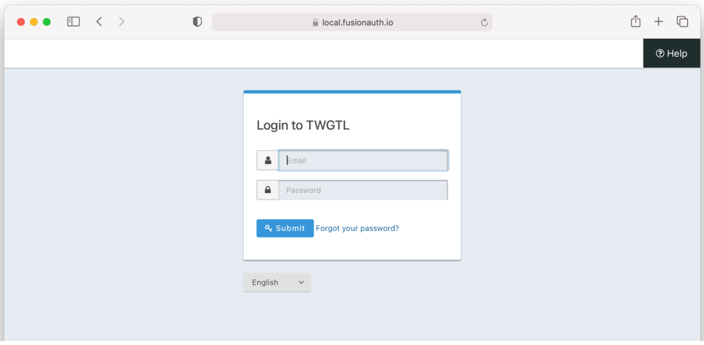
What do we mean by native forms? Most developers have at one time written their own login and registration forms directly into an application. They create a table called `users` and it stores `usernames` and `passwords`. Then they write the registration and the login forms (HTML or some other UI). The registration form collects the `username` and `password` and checks if the user exists in the database. If they don’t, the application inserts the new user into the database. The login form collects the `username` and `password` and checks if the account exists in the database and logs the user in if it does. This type of implementation is what we call native forms.

The only difference between native forms and the **Local login and registration** OAuth mode is that with the latter you delegate the login and registration process to an OAuth server rather than writing everything by hand. Additionally, since you control the OAuth server and your application, it would be odd to ask the user to “authorize” your application. Therefore, this mode does not include the permission grant screens that often are mentioned in OAuth tutorials. Never fear; we’ll cover these in the next few sections.

So, how does this work in practice? Let’s take a look at the steps for a fictitious web application called “The World’s Greatest ToDo List” or “TWGTL” (pronounced Twig-Til):

1. A user visits TWGTL and wants to sign up and manage their ToDos.
2. They click the “Sign Up” button on the homepage.
3. This button takes them over to the OAuth server. In fact, it takes them directly to the registration form that is included as part of the OAuth workflow (specifically the Authorization Code grant which is covered later in this guide).
4. They fill out the registration form and click “Submit”.
5. The OAuth server ensures this is a new user and creates their account.
6. The OAuth server redirects the browser back to TWGTL, which logs the user in.
7. The user uses TWGTL and adds their current ToDos. Yay!
8. The user stops using TWGTL; they head off and do some ToDos.
9. Later, the user comes back to TWGTL and needs to sign in to check off some ToDos. They click the `My Account` link at the top of the page.
10. This takes the user to the OAuth server’s login page.
11. The user types in their username and password.
12. The OAuth server confirms their identity.
13. The OAuth server redirects the browser back to TWGTL, which logs the user in.
14. The user interacts with the TWGTL application, merrily checking off ToDos.

That’s it. The user feels like they are registering and logging into TWGTL directly, but in fact, TWGTL is delegating this functionality to the OAuth server. The user is none-the-wiser so this is why we call this mode *Local login and registration*.



I bet your login screen will be much prettier.

An aside about this mode and mobile applications

The details of this mode has implications for the security best practices recommended by some of the standards bodies. In particular, the OAuth 2.0 for Native Apps[1] Best Current Practices (BCP) recommends against using a webview:

> This best current practice requires that native apps MUST NOT use embedded user-agents to perform authorization requests…

This is because the "embedded user-agents", also known as webviews, are under control of the mobile application developer in a way that the system browser is not.

If you are operating in a mode where the OAuth server is under a different party's control, such as the third-party login that we'll cover next, this prohibition makes sense. But in this mode, you control everything. In that case, the chances of a malicious webview being able to do extra damage is minimal, and must be weighed

[1]https://tools.ietf.org/html/rfc8252

against the user interface issues associated with popping out to a system browser for authentication.

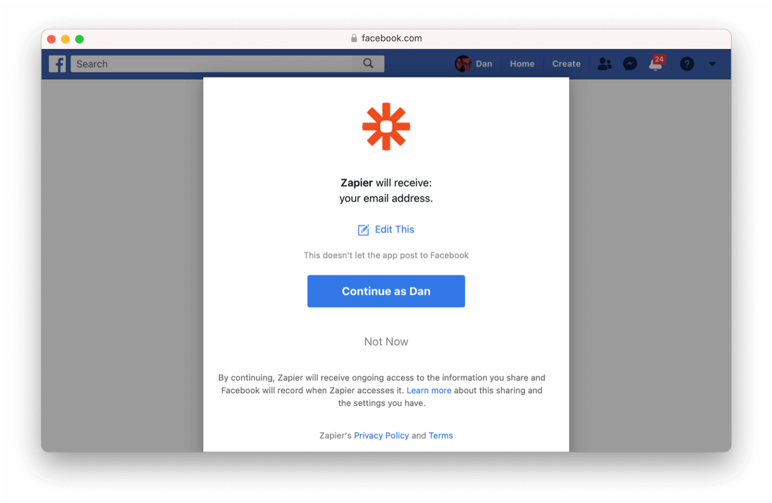
Third-party login and registration

The **Third-party login and registration** mode is typically implemented with the classic “Login with …” buttons you see in many applications. These buttons let users sign up or log in to your application by logging into one of their other accounts (i.e. Facebook or Google). Here, your application sends the user over to Facebook or Google to log in.

Let’s use Facebook as an example OAuth provider. In most cases, your application will need to use one or more APIs from the OAuth provider in order to retrieve information about the user or do things on behalf of the user (for example sending a message on behalf of the user). In order to use those APIs, the user has to grant your application permissions. To accomplish this, the third-party service usually shows the user a screen that asks for certain permissions. We’ll refer to these screens as the “permission grant screen” throughout the rest of the guide.

For example, Facebook will present a screen asking the user to share their email address with your application. Once the user grants these permissions, your application can call the Facebook APIs using an access token (which we will cover later in this guide).

Here’s an example of the Facebook permission grant screen, where Zapier would like to access a user’s email address:



Zapier will receive:
your email address.

This doesn't let the app post to Facebook

By continuing, Zapier will receive ongoing access to the information you share and Facebook will record when Zapier accesses it. Learn more about this sharing and the settings you have.

Zapier's Privacy Policy and Terms

The Facebook permissions grant screen for Zapier.

After the user has logged into the third-party OAuth server and granted your application permissions, they are redirected back to your application and logged into it.

This mode is different from the previous mode because the user logged in but also granted your application permissions to the service (Facebook). This is one reason so many applications leverage "Login with Facebook" or other social integrations. It not only logs the user in, but also gives them access to call the Facebook APIs on the user's behalf.

Social logins are the most common examples of this mode, but there are plenty of other third-party OAuth servers beyond social networks (GitHub or Discord for example).

This mode is a good example of federated identity. Here, the user's identity (username and password) is stored in the third-party system. They are using that system to register or log in to your application.

So, how does this work in practice? Let's take a look at the steps for our TWGTL application if we want to use Facebook to register and log users in:

1. A user visits TWGTL and wants to sign up and manage their ToDos.
2. They click the “Sign Up” button on the homepage.
3. On the login and registration screen, the user clicks the “Login with Facebook” button.
4. This button takes them over to Facebook’s OAuth server.
5. They log in to Facebook (if they aren’t already logged in).
6. Facebook presents the user with the permission grant screen based on the permissions TWGTL needs. This is done using OAuth scopes, which we will cover later in this guide.
7. Facebook redirects the browser back to TWGTL, which logs the user in. TWGTL also calls Facebook APIs to retrieve the user’s information.
8. The user begins using TWGTL and adds their current ToDos.
9. The user stops using TWGTL; they head off and do some ToDos.
10. Later, the user comes back to TWGTL and needs to log in to check off some of their ToDos. They click the `My Account` link at the top of the page.
11. This takes the user to the TWGTL login screen that contains the “Login with Facebook” button.
12. Clicking this takes the user back to Facebook and they repeat the same process as above.

You might be wondering if the **Third-party login and registration** mode can work with the **Local login and registration** mode. Absolutely! This is what I like to call **Nested federated identity** (it’s like a hot pocket in a hot pocket[2]). Basically, your application delegates its registration and login forms to an OAuth server like FusionAuth. Your application also allows users to sign in with Facebook by enabling that feature of the OAuth server (FusionAuth calls this

[2]https://www.youtube.com/watch?v=N-i9GXbptog

the **Facebook Identity Provider**). It's a little more complex, but the flow looks something like this:

1. A user visits TWGTL and wants to sign up and manage their ToDos.
2. They click the "Sign Up" button on the homepage.
3. This button takes them over to the OAuth server's login page.
4. On this page, there is a button to "Login with Facebook" and the user clicks that.
5. This button takes them over to Facebook's OAuth server.
6. They log in to Facebook.
7. Facebook presents the user with the permission grant screen.
8. The user authorizes the requested permissions.
9. Facebook redirects the browser back to TWGTL's OAuth server, which reconciles out the user's account.
10. TWGTL's OAuth server redirects the user back to the TWGTL application.
11. The user is logged into TWGTL.

> What does "reconcile out" mean? OAuth has its jargon, oh yes. To reconcile a user with a remote system means optionally creating a local account and then attaching data and identity from a remote data source like Facebooik to that account. The remote account is the authority and the local account is modified as needed to reflect remote data.

The nice part about this workflow is that TWGTL doesn't have to worry about integrating with Facebook (or any other provider) or reconciling the user's account. That's handled by the OAuth server. It's also possible to delegate to additional OAuth servers, easily adding "Login with Google" or "Login with Apple". You can also nest deeper than the 2 levels illustrated here.

First-party login and registration

The **First-party login and registration** mode is the inverse of the **Third-party login and registration** mode. Basically, if you happen to be Facebook (hi Zuck!) in the examples above and your customer is TWGTL, you are providing the OAuth server to TWGTL. You are also providing a way for them to call your APIs on behalf of your users.

This type of setup is not just reserved for the massive social networks run by Silicon Valley moguls; more and more companies are offering this to their customers and partners, therefore becoming platforms.

In many cases, companies are also leveraging easily integratable auth systems like FusionAuth to provide this feature.

Enterprise login and registration

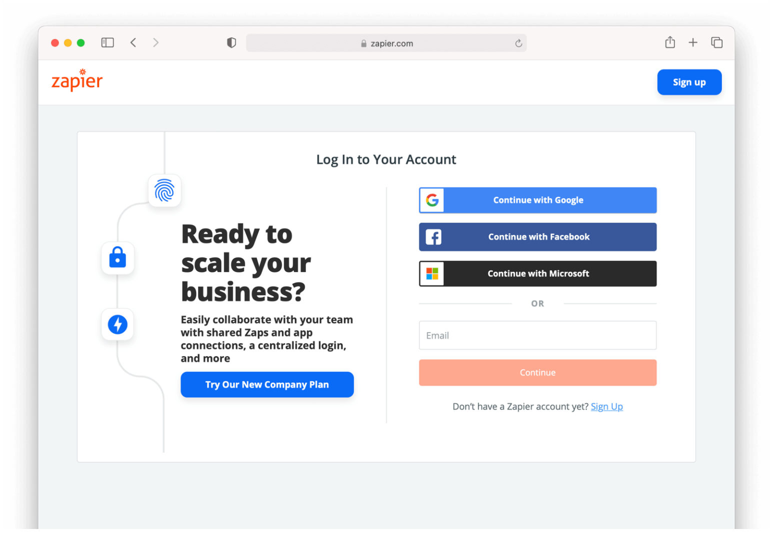
The **Enterprise login and registration** mode is when your application allows users to sign up or log in with an enterprise identity provider such as a corporate Active Directory. This mode is very similar to the **Third-party login and registration** mode, but with a few salient differences.

First, it rarely requires the user to grant permissions to your application using a permission grant screen. Typically, a user does not have the option to grant or restrict permissions for your application. These permissions are usually managed by IT in an enterprise directory or in your application.

Second, this mode does not apply to all users of an application. In most cases, this mode is only available to the subset of users who exist in the enterprise directory. The rest of your users will either log in directly to your application using **Local login and**

registration or through the **Third-party login and registration** mode. In some cases, the user's email address determines the authentication source.

You might have noticed some login forms only ask for your email on the first step like this:



Ready to scale your business?

Easily collaborate with your team with shared Zaps and app connections, a centralized login, and more

Don't have a Zapier account yet? Sign Up

For Zapier, the user's email address is requested before any password.

Knowing a user's email domain allows the OAuth server to determine where to send the user to log in or if they should log in locally. If you work at Example Company, proud purveyors of TWGTL, providing `brian@example.com` to the login screen allows the OAuth server to know you are an employee and should be authenticated against a corporate authentication source. If instead you enter `dan@gmail.com`, you won't be authenticated against that directory.

Outside of these differences, this mode behaves much the same as the **Third-party login and registration** mode.

This is the final mode where users can register and log in to your application. The remaining modes are used entirely for authorization, usually to application programming interfaces (APIs). We'll cover these modes next.

Third-party service authorization

The third-party service authorization mode is quite different from the **Third-party login and registration** mode; don't be deceived by the similar names. Here, the user is already logged into your application. The login could have been through a native form (as discussed above) or using the **Local login and registration** mode, the **Third-party login and registration** mode, or the **Enterprise login and registration** mode. Since the user is already logged in, all they are doing is granting access for your application to call third-party's APIs on their behalf.

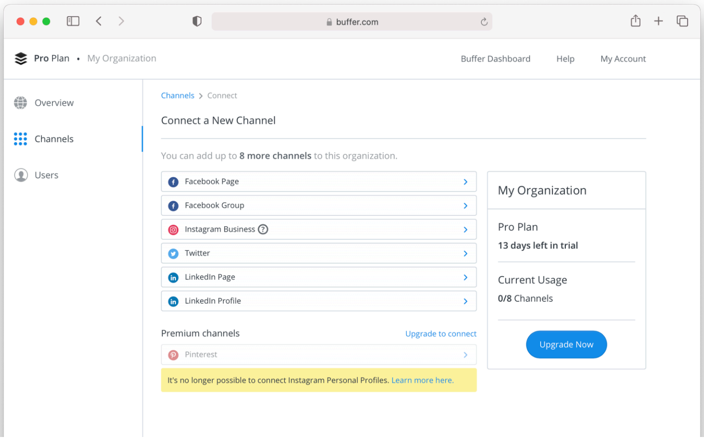
For example, let's say a user has an account with TWGTL, but each time they complete a ToDo, they want to let their WUPHF[3] followers know. (WUPHF is an up and coming social network; sign up at getwuphf.com.) To accomplish this, TWGTL provides an integration that will automatically send a WUPHF when the user completes a ToDo. The integration uses the WUPHF APIs and calling those requires an access token. In order to get an access token, the TWGTL application needs to log the user into WUPHF via OAuth.

To hook all of this up, TWGTL needs to add a button to the user's profile page that says "Connect your WUPHF account". Notice it doesn't say "Login with WUPHF" since the user is already logged in; the user's identity for TWGTL is not delegated to WUPHF. Once the user clicks this button, they will be taken to WUPHF's OAuth server to log in and grant the necessary permissions for TWGTL to WUPHF for them.

Since WUPHF doesn't actually exist, here's an example screenshot

[3]https://www.youtube.com/watch?v=yL1z1ZHD0K4

from Buffer, a service which posts to your social media accounts such as Twitter.



Connect a New Channel

You can add up to **8 more channels** to this organization.

It's no longer possible to connect Instagram Personal Profiles. Learn more here.

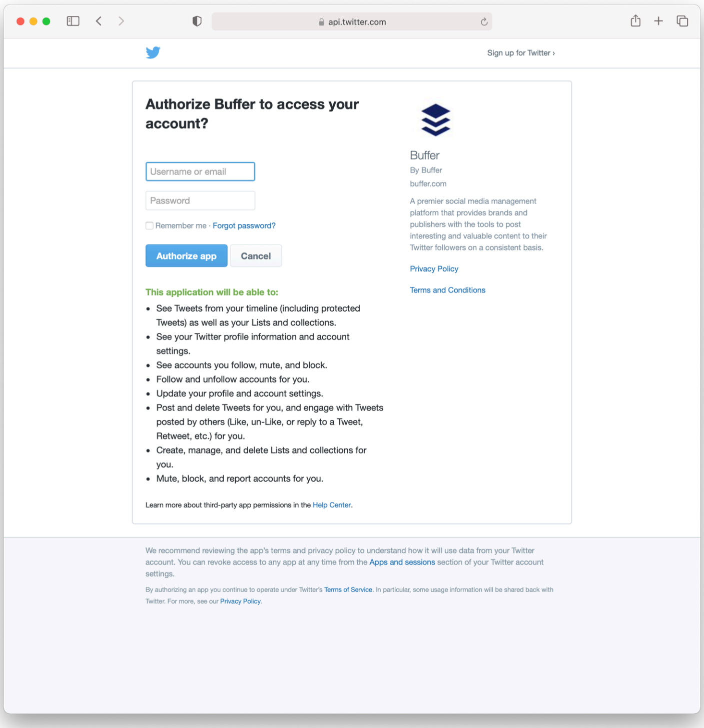
My Organization

Pro Plan
13 days left in trial

Current Usage
0/8 Channels

Buffer would like to connect to your accounts. Pretty please?

When you connect a Twitter account to Buffer, you'll see a screen like this:



Authorize Buffer to access your account?

This application will be able to:

- See Tweets from your timeline (including protected Tweets) as well as your Lists and collections.
- See your Twitter profile information and account settings.
- See accounts you follow, mute, and block.
- Follow and unfollow accounts for you.
- Update your profile and account settings.
- Post and delete Tweets for you, and engage with Tweets posted by others (Like, un-Like, or reply to a Tweet, Retweet, etc.) for you.
- Create, manage, and delete Lists and collections for you.
- Mute, block, and report accounts for you.

Learn more about third-party app permissions in the Help Center.

Buffer
By Buffer
buffer.com
A premier social media management platform that provides brands and publishers with the tools to post interesting and valuable content to their Twitter followers on a consistent basis.

Privacy Policy

Terms and Conditions

We recommend reviewing the app's terms and privacy policy to understand how it will use data from your Twitter account. You can revoke access to any app at any time from the Apps and sessions section of your Twitter account settings.

By authorizing an app you continue to operate under Twitter's Terms of Service. In particular, some usage information will be shared back with Twitter. For more, see our Privacy Policy.

Buffer would like to connect to your Twitter account. Tweeting is so hot right now.

The workflow for this mode looks like this:

1. A user visits TWGTL and logs into their account.
2. They click the "My Profile" link.
3. On their account page, they click the "Connect your WUPHF account" button.
4. This button takes them over to WUPHF's OAuth server.
5. They log in to WUPHF.
6. WUPHF presents the user with the "permission grant screen" and asks if TWGTL can WUPHF on their behalf.

7. The user grants TWGTL this permission.
8. WUPHF redirects the browser back to TWGTL where it calls WUPHF's OAuth server to get an access token.
9. TWGTL stores the access token in its database and can now call WUPHF APIs on behalf of the user. Success!

First-party service authorization

The **First-party service authorization** mode is the inverse of the **Third-party service authorization** mode. When another application wishes to call your APIs on behalf of one of your users, you are in this mode. Here, your application is the "third-party service" discussed above. Your application asks the user if they want to grant the other application specific permissions. Basically, if you are building the next Facebook and want developers to be able to call your APIs on behalf of their users, you'll need to support this OAuth mode.

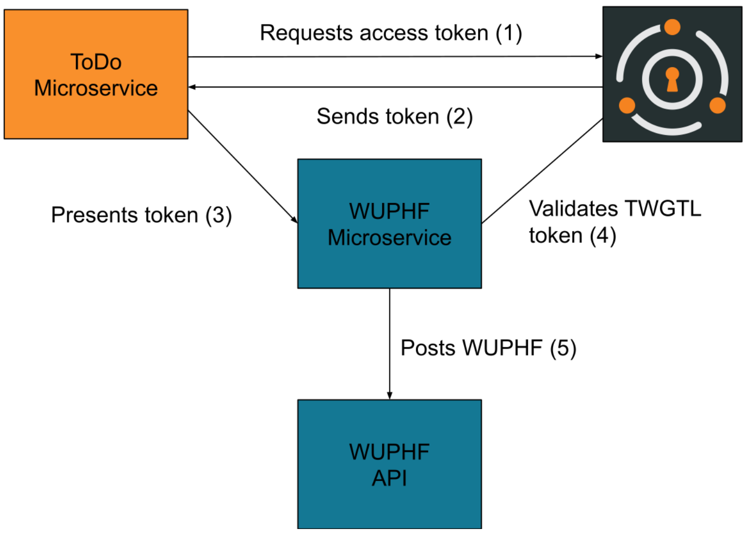
With this mode, your OAuth server might display a "permission grant screen" to the user asking if they want to grant the third-party application permissions to your APIs. This isn't strictly necessary and depends on your requirements.

Machine-to-machine authorization

The **Machine-to-machine authorization** OAuth mode is different from the previous modes we've covered. This mode does not involve users at all. Rather, it allows an application to interact with another application. Normally, this is backend services communicating with each other via APIs.

Here, one backend needs to be granted access to the other. We'll call the first backend the source and the second backend the target. To accomplish this, the source authenticates with the OAuth server. The OAuth server confirms the identity of the source and then returns a token that the source will use to call the target. This token can also include permissions that are used by the target to authorize the call the source is making.

Using our TWGTL example, let's say that TWGTL has two microservices: one to manage ToDos and another to send WUPHFs. Overengineering is fun! The ToDo microservice needs to call the WUPHF microservice. The WUPHF microservice needs to ensure that any caller is allowed to use its APIs before it WUPHFs.



The WUPHF microservice needs to ensure the TWGTL microservice is authorized.

The workflow for this mode looks like this:

1. The ToDo microservice authenticates with the OAuth server.
2. The OAuth server returns a token to the ToDo microservice.
3. The ToDo microservice calls an API in the WUPHF microservice and includes the token in the request.

4. The WUPHF microservice verifies the token by calling the OAuth server (or verifying the token itself if the token is a JWT).
5. If the token is valid, the WUPHF microservice performs the operation.

Device login and registration

The **Device login and registration** mode is used to log in to (or register) a user's account on a device that doesn't have a rich input device like a keyboard. In this case, a user connects the device to their account, usually to ensure their account is active and the device is allowed to use it.

A good example of this mode is setting up a streaming app on an Apple TV, smart TV, or other device such as a Roku. In order to ensure you have a subscription to the streaming service, the app needs to verify the user's identity and connect to their account. The app on the Apple TV device displays a code and a URL and asks the user to visit the URL. The workflow for this mode is as follows:

1. The user opens the app on the Apple TV.
2. The app displays a code and a URL.
3. The user types in the URL displayed by the Apple TV on their phone or computer.
4. The user is taken to the OAuth server and asked for the code.
5. The user submits this form and is taken to the login page.
6. The user logs into the OAuth server.
7. The user is taken to a "Finished" screen.
8. A few seconds later, the device is connected to the user's account.

This mode often takes a bit of time to complete because the app on the Apple TV is polling the OAuth server.

OAuth Grants

Authorization Code grant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

Login/register buttons

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

Authorize endpoint parameters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

Logging in

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

Redirect and retrieve the tokens

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themoderngguidetooauth.

Tokens

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themoderngguidetooauth.

User and token information

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themoderngguidetooauth.

The Introspect endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themoderngguidetooauth.

The UserInfo endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themoderngguidetooauth.

Local login and registration with the Authorization Code grant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

Storing tokens as cookies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

Storing tokens in the session

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

Refreshing the access token

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

Third-party login and registration (also Enterprise login and registration) with the Authorization Code grant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

Third-party authorization with the Authorization Code grant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

First-party login and registration and first-party service authorization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

Implicit grant in OAuth 2.0

Resource Owner's Password Credentials grant

Client Credentials grant

Device grant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

The Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

The Solution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

Device Grant User Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

How It Works

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/themodernguidetooauth.

Conclusion