OAuth 2.0

Backend and APIs Knowledge

Learn this authorization framework operates behind the scenes. **I think you already have heard of it right? If not, get it to know here!**

@mauricioperez

→

@mauricioperez

01

What is OAuth 2.0?

Let's put it simply, it's an authorization framework where it already **has well-predefined authentication and authorization flows** from different providers and lets you access data from **third-party providers** (applications).

It operates through a requesting process where the **resource owner** gives the permission to access data from application A to application B, and the rest is all automated and you'll know it here.

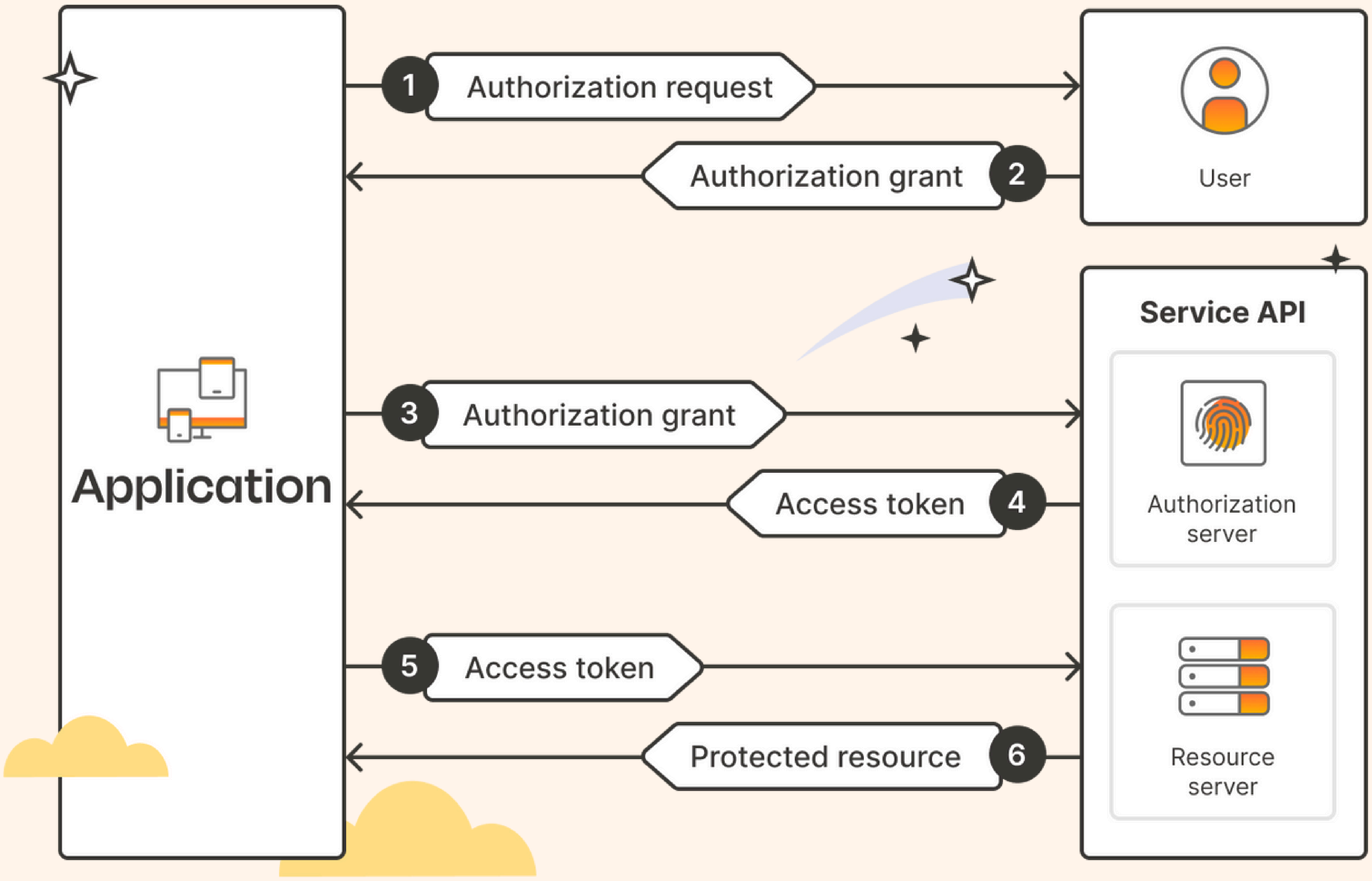
The real world example is when you make available to users to log in with Google credentials but you never see their passwords or credentials, but they already have access to your application.

"**Resource owner**"?? What the hell is that? Let's see in the next layer

@mauricioperez

02 Architecture of OAuth 2.0

Let's see each component and its working process



https://blog.postman.com/what-is-oauth-2-0/

@mauricioperez

03 Architecture of OAuth 2.0

But first of all before starting we need to define some terminology that are the fundamental pieces in this game:

- **Resource Owner**
- **Client**
- **Authorization Server**
- **Resource Server**

Okay, the dynamic goes like this: I'll explain what each of these terms means and then I'll explain you the architecture with all of them compounded.

@mauricioperez

04

Who is the Resource Owner?

Okay, let's get into the dip.

The **Resource Owner** is you, and as a person you are viewed as the **entity with the greatest power in the process** of authentication and data trespassing through services accross the network, in other words, **without your permission no process of OAuth 2.0 is into existence**.

Look, whenever an application requests to get some of your data, might be some data from your Google Account, or your Github account, **whatever of these applications whatsoever are going to be riged to your word**, and they can't access your data if you don't give permission...

You got it? Let me know in the comments if not..

@mauricioperez

05

Who is the Client?

The **Client** is the application which first request the permission to the **Resource Owner** and will be the responsible that catches the data as a first instance, to posterior processing of it.

But the Client will not have the permission at all by this first step it is just requesting to the **Resource Server** to get the **Resource Owner** data, now if the user gives the consent is the process that we are going to be following.

But I am not going to explain you all of this yet, because now I want to clarify you what is the **Resource Server**.

@mauricioperez

06

Who is the Resource Server?

Now the **Resource Server** is the server responsible to provide you the data that you are trying to get from the user or better known in this process: **The Resource Owner**.

We are going to leave from rounds up, let's put it simple, **imagine you are trying to get user data from the User's Google account**, now the resource server is the one that will provide you this data, **who you think it is in this example??**

If you guessed **the google server** you are very right in this question! **Congratulations!** So it's simple, the resource server is the one that will give you the data that you need. **But look is not that simple**, I'll tell you this process...

Who is the Authorization Server?

Guess what!! We hadn't already finished! There's still left to explain the **Auth Server**.

Now the **Authorization Server** is the one that will prove authenticity from your services that you are loyal and the legitimate application that will use responsible the **Resource owner's data (the user)**.

How you are supposed to prove you are legitimate to use user's data? Well that is possible if you own the third-party credentials **that were provided to you** upon your app registration **on the third-party API**.

Simply **you had mandatory to have registered your application in the external service that is going to ask for data** and the external service will provide you credentials and you have to use them for it, no more. Okay?

@mauricioperez

Let’s explain the architecture

I will divide this process into five parts:

1. **Client** requests for **Resource Owner’s** data to external API
2. **Resource Owner** accepts or declines
3. **Client** sends code data to **Authorization Server**
4. **Authorization Server** exchanging process with **Resource Server**
5. **Resource Server** provides the content

@mauricioperez

Phase 1: "Client requesting..."

You are in the application "MeetForHealth" but you want to create an account via Google. Now you see a button "**Sign in with Google**", so you clicked it (or tap it whatever).

Now it will redirect you to a window which is already prepared from Google API, but this is provided upon before registration you made to google API through the Google Cloud Console. (To see this process search in Google how this is done).

But this process will consist if the Resource Owner (user) accepts all of the terms and scope pretended to access for **Resource Owner's account** and if so the process will continue to phase 2...

@mauricioperez

Phase 2: “Resouce Owner accepts or declines”

Now if the user accepted the terms and the data that the app “MeetForHealth” (The Client) is trying to access, the **Resource Server** will respond with a code which the **Client** is supposed to exchange it somehow to access the data.

Now this code provided by the Resource Server, in this case is Google, **will not contain any data**, this code is only for purposes **for the Client to see if this is the application is the legitimate one** that was registered in the Google Services to consume user’s data. **The type of codes provided vary dependending of what is the type of application you are doing.**

To see more about this, take a look at official docs.

@mauricioperez

Phase 3: “Client and Authorization Server”

Now with the code in hand, what the **Client** is going to do is to send the code received from the **Resource Server** to **Authorization Server** to somehow decipher it and obtain user’s data.

Now to put this into practice, let’s say that the “MeetForHealth” app is built with Spring Boot + NextJS development stack, now NextJS will send the code to the backend which is on Spring Boot, and will tell to it that it owns the code but don’t have the credentials to prove legitimacy for Google Services. **Can you guess here which is the Authorization Server?** Yes, it’s **Spring Boot in this case**, because it owns the credentials provided.

So what Spring Boot will do here is an exchanging process with the Resource Server

Phase 4: “Exchanging process”

The exchanging process will take place between the **Authorization Server** and the **Resource Server**.

The process is simple, the **Client** sent the code to the **Authorization Server**, then the **Authorization Server sends a request of exchange to the Resource Server along with the credentials,** like the Secret and ID of the app, now the Resouce Server will respond in case it is valid **as long as the secret and the ID are true** to what it has registered for the Client accessing the Resource Owner’s data, with a token.

The Resource Server will respond with a token of response, this is a token that must be sent then to the Google Services in this case to get the data, but this is just a simple process then of requesting data with the token already gotten.

@mauricioperez

Phase 5: “Obtaining user data”

Now that all the process is almost done, to access the data, we need to send the token sent by the **Resource Server** from the **Authorization Server** to get the users data.

To make this **you have to put alongside your request with the token provided** and then simply catch the data returned in a JSON format.

Then what you need to do is to **send the data returned by the Resource Server** (which is from Google in this example) **to the Client again (in NextJS) and display the user data**.

There are more actions to do, like to **save the user logged in** as a new record **in your database** or **issue a JWT token**, but this is another topic for another post.

Thank you!

@mauricioperez

Did you get all from OAuth 2.0 let me know in the comments!

Like Celebrate Support Love Insightful Funny