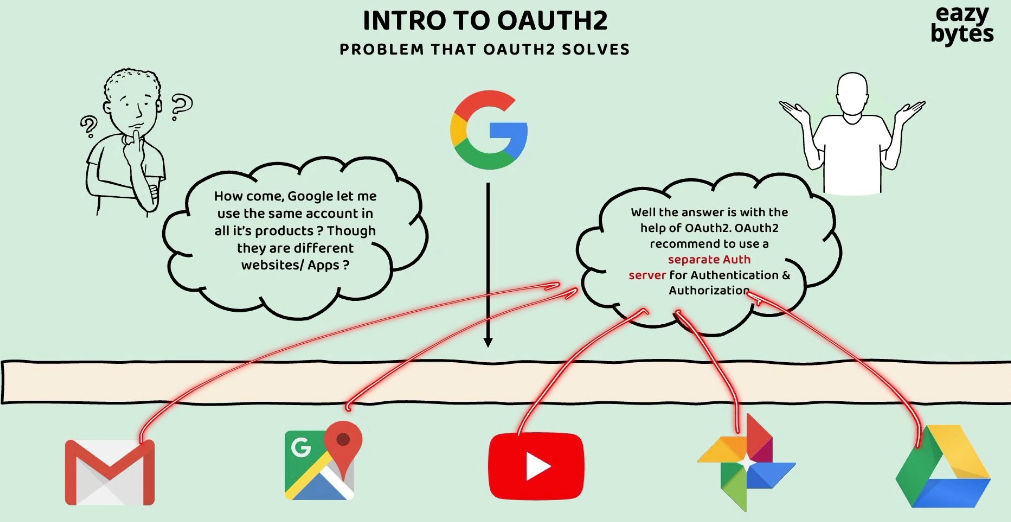
**12. OAUTH2 and openid connect**

what are the most common problems that theis OAuth2 framework

is trying to solve inside the industry.



We all use this various Google products. Google has primarily Gmail as a product.

They also have Google Maps, YouTube, Google Photo, Google Drive.

So similarly, there are many other applications which are developed and maintained by Google or Alphabet.

All these applications of Google, they have their own separate code base.

All these web applications are mobile applications. They're going to be deployed

in different, different servers. But have you anytime wondered how all these Google products, they allow the same credentials as an input?

We as an end user, whenever we want to use any Google product, we need to create our own account with the help of Gmail. So the same Gmail credentials you can use

to authenticate in all the remaining applications.

For example, if I want to use YouTube to see the videos or to upload some videos,

I can log in into these YouTube mobile application or web application with the Gmail credentials that I have created previously. The same applies for the remaining Google products.

So how this is possible is, behind the scenes, Google as an organization,

it is following the OAuth2 standard or OAuth2 framework.

So whenever someone is following the OAuth2, it is going to recommend them

to use a separate authorization server which is responsible for storing the end user credentials, and the same Auth server is also responsible for end user authentication and authorization.

So initially, when an end user register into the Gmail account, a separate Auth server

will handle the registration process and will store the end user credentials.

The same Auth server will be used by all the applications inside Google

to authenticate the user and to perform authorization of the end user.

This way, we as an end users of Google products, we don't have to create

different, different accounts in different products and we don't have to maintain different credentials for different products of Google.

So the responsibility of this Auth server is, it is going to perform the authentication of the end user and it is going to issue the tokens during the login operation.

The same token can be used with the other applications as well.

For example, inside one of my browser tab, think like, I have logged in into my Gmail account. With the help of Auth server, I'm going to get an token. Now, this token will be saved somewhere inside my browser. Maybe after a few hours or maybe after few days if I try to open one of the different product of Google like Maps or YouTube or Photos or Drive, it is not going to ask me the credentials again, it is simply going to use the same token that is available inside my browser.

And with this token, the authentication and authorization is going to happen

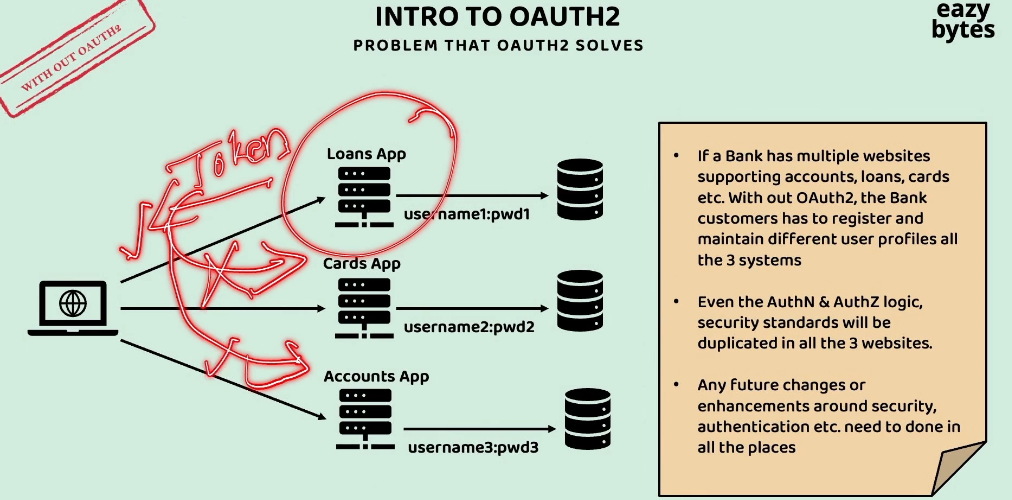
without me entering any credentials.

So this is possible because all these web applications are mobile applications. They're pointing to the same Auth server which issued the token originally.

So this is a very first major problem that OAuth2 is solving inside the industry.

Always OAuth2 framework is going to recommend to maintain separate authentication server inside an organization.

Let's try to take one more scenario and try to understand what are the advantages that we get if you try to maintain a separate Auth server inside an organization.



Think like, there is a bank application which has multiple websites supporting accounts, loans, and cards.

If these bank application, if they're not following the OAuth2 framework,

then the bank customers, they have to register and maintain different user profiles

in all the three systems.

Inside each of these applications, if they have their own authentication and authorization logic, then the end user will end up creating different credentials for different applications, and all these credentials are going to be stored in different storage systems.

So here, clearly, these people, they're not following the OAuth2 framework

inside the bank organization. With these, we are going to have multiple challenges.

The very first challenge or drawback is the security-related logic

like authentication and authorization, it is getting duplicated in multiple places.

So in future, if there is a requirement change around the authentication and authorization, the same has to be done in all these three places.

The next challenge is, think like, this loan application has issued a token

during the login process to the client application.

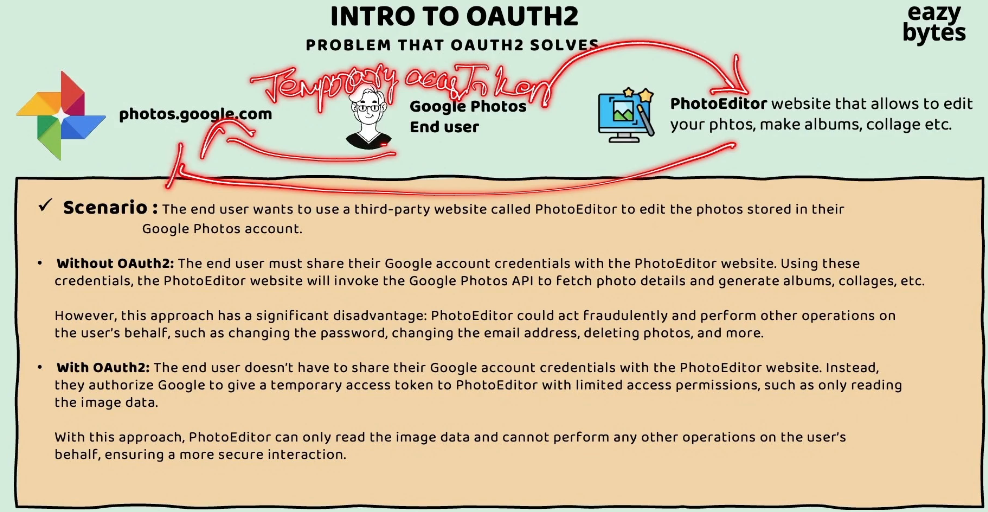
So the same token is not going to make any sense to the other applications.

That's why they're going to reject that token. They always accept their own credentials, and they're going to issue their own tokens during the authentication and authorization process.

Always try to separate your security-related logic into an Auth server

so that your individual applications, they are going to simply have the logic

related to the business logic.



The next common problem that OAuth2 framework try to solve inside the industry is,

Delegated Authentication or Authorization.

Think like you are an end user who maintain all your photos inside the Google Photos, and there is a third-party app with the name PhotoEditor.

So this PhotoEditor third-party app, it is going to allow you to edit your photos

with the help of artificial intelligence, and it is also going to help you

to create albums out of your photos. It is also going to help you to create collage of your photos.

Since this third-party app is providing all these features, you decided to use this third-party app.

So let's try to imagine this scenario

without OAuth2 and with OAuth2.

In a very common basic scenario, what you're going to do?

You're going to download the photo from your Google Photo.

The same you're going to upload into the PhotoEditor for performing some edits.

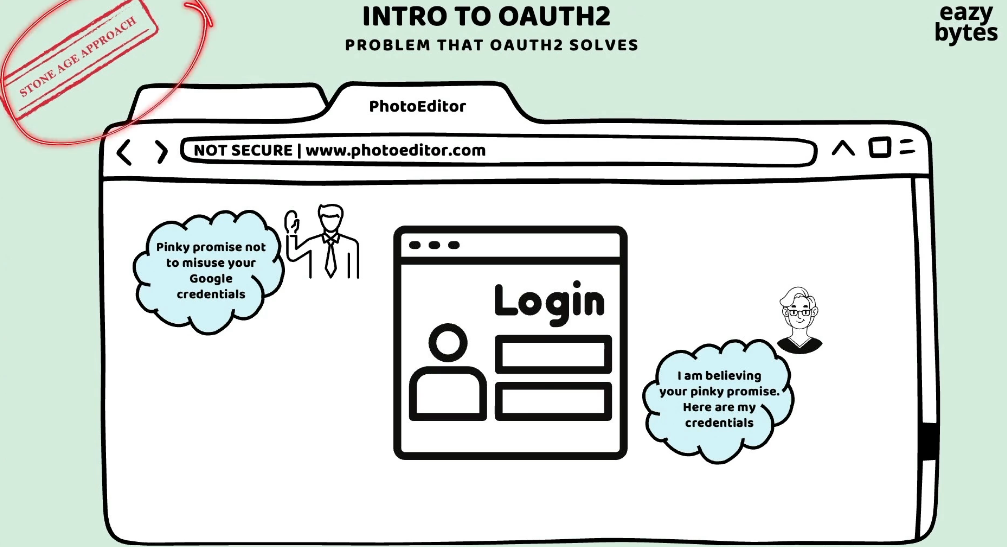
So the same process you need to do for all the photos.

So this is going to be very cumbersome process.

So to make your life easy, PhotoEditor, they supports integration with the Google Photos.

So this integration with the OAuth2, it is going to be supported in two different styles.

One is without OAuth2 and the other one is with OAuth2 framework.



So let's try to understand what is going to happen inside the without OAuth2 scenario.

In this scenario, the PhotoEditor third-party app, it is going to ask the end user

to provide his Google Photo credentials.

Though sharing the Google credentials directly to this website is a risky option,

still this end user want to share his credentials because he liked the features

provided by this website a lot. They're going to use the credentials of this end user,

and they're going to read the photos from the Google Photos website for editing.

So this used to be a stone age approach where people used to follow these

before OAuth2 introduced.

So whatever we discussed, this is a very happy scenario.

But there is a significant disadvantage in this approach, which is, the PhotoEditor application, they may do some fraudulent activities by performing other operations on user's behalf. They can change the password or they can change the email address because they know the actual credentials of the end user.

Now, let's try to imagine the same scenario with the OAuth2.

So with OAuth2 what is going to happen is, the end user will not share his Google credentials to the third-party application. Instead, he's going to let Google Photos

to issue an temporary access token. Using this temporary access token, the PhotoEditor app, they can only read the photos from the end user Google Photos account. With the access token that is issued by the Google Photos

on behalf of the end user, it is only useful to read the images or to load the images.

Apart from reading, any other operations will straight away rejected by the Google photos because the access token issued to the third-party app

does not have enough privileges.

So whenever we are using OAuth2, what is happening?

The end user delegating the authentication and authorization

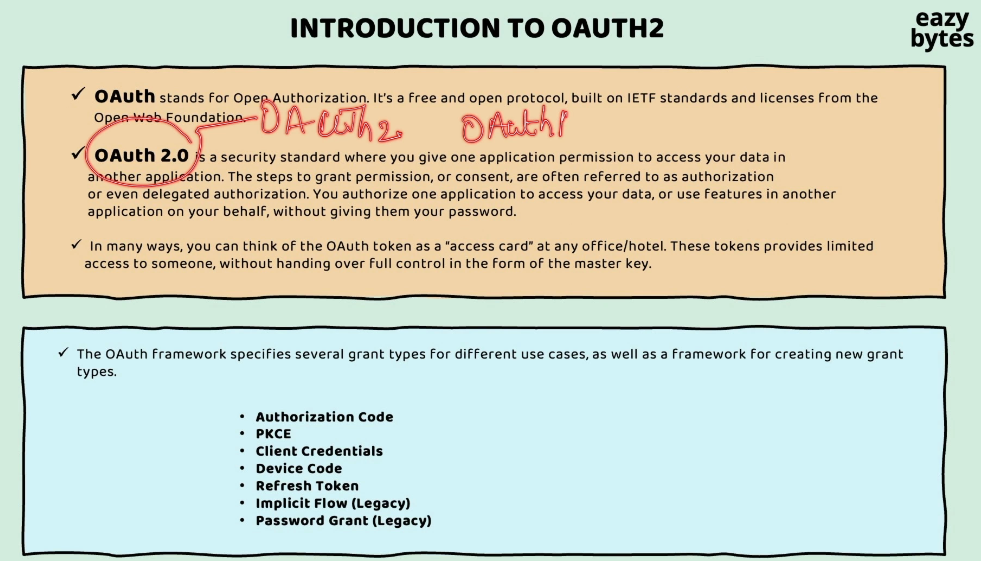
to the PhotoEditor app to read his photos from the Google Photos.

And whenever the end user feel like he no more needed the services of this website,

he can easily revoke the permission that he has initially issued to this website

by logging into the Google account.

Introduction to OAUTH2



OAuth 2 is a free and open source protocol that is built by the open source community. When I say OAuth 2, it is a standard, or it is a protocol,

or it is a specification that every organization has to follow

whenever they're trying to implement authentication

and authorization inside their organization.

Whenever an access token is issued to a client application or a third-party application, with the help of that access token, they should be able to read that data of the end user. So with the help of this access token, they should be able to perform only a limited functionality.

you need to follow one of the grant types that are supported by the OAuth 2 framework.

For example, if an end user is involved during the authentication and authorization process, then we can use either of the Authorization Code or PKCE grant type flow

based upon the type of our application.

If your application is built using the JavaScript frameworks, like, Angular, React, or mobile applications, in such scenarios, you need to use PKCE.

Otherwise, you can safely use Authorization Code.

The next grant type flow that we have here is Client Credentials.

So these Client Credentials, we need to use whenever two different backend applications or two different APIs, they're trying to communicate with each other.

So in these scenarios, the end user is not involved.

whenever two different devices or IoT applications, if they're trying to communicate with each other, we need to use the Device Code grant type flow.

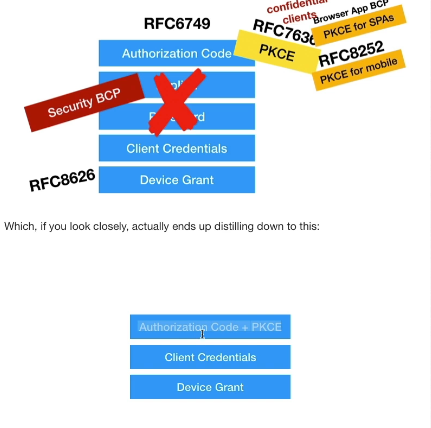
Similarly, there is a Refresh Token grant type flow that can be used in the scenarios whenever an access token is expired and if you're looking to get a new access token.

Implicit flow is very similar to the Authorization Code grant type flow, but it's a older version of Authorization Code grant type flow, which has some drawbacks.

That's why this Implicit flow is deprecated right now.

This Password grant type flow is also a flavor of Authorization Code grant type flow,

and this is also deprecated due to its drawbacks.

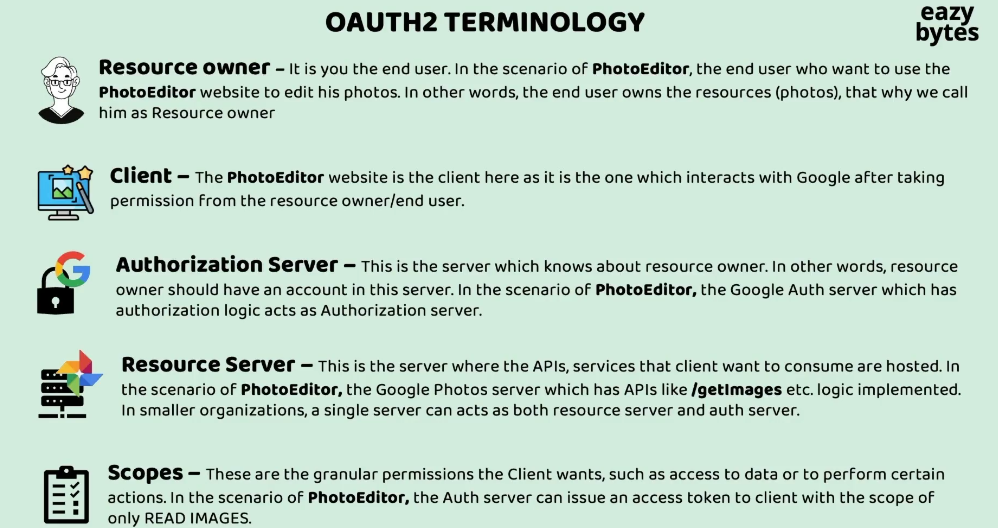


As part of OAuth 2.1, the open source community, they're trying to reduce the number of grant type flows to Authorization Code plus PKCE and Client Credentials and Device grant.

With that, we are only going to have three grant type flows in the future when this 2.1 is officially released.

So they are recommending to use the Authorization Code with PKCE instead of using them separately.

OAuth2 terminologies or jargons



So in the scenario of PhotoEditor, the PhotoEditor application,

it is going to invoke the /getimages API, which is exposed by the Google Photos resource server.

So when client application asks for the resources, obviously, the resource server will expect an access token before providing the response.

So how to get the access token, the client first has to reach out to the authorization server to get the access token. Once the access token is received, using the same access token, it can reach out to the Resource Server to fetch the resources.

In smaller organizations, there is a good possibility that a single server can act

as both resource server and auth server.

So just like how we have authorities and roles concept inside the Spring Security framework, very similarly, inside the OAuth 2 standards, we call them with the name or with the jargon scopes.

Demo of OAuth2 Sample flow