# Reference links

<https://oauth.net/>

<https://fusionauth.io/articles/oauth/modern-guide-to-oauth>

* [Client Credentials](https://www.oauth.com/oauth2-servers/access-tokens/client-credentials/) (oauth.com)
* [Client Authentication Methods](https://developer.okta.com/docs/reference/api/oidc/#refresh-token) (developer.okta.com)
* [Client Secret](https://www.rfc-editor.org/rfc/rfc6749#section-2.3.1) (RFC 6749 Section 2.3.1) – u can see all RFC here (Request for change)
* All about MTLS <https://datatracker.ietf.org/doc/html/rfc8705>

Udemy course reference links

#### **Resource Server**

* **Repository(Original)**: <https://github.com/simplyi/ResourceServer>
* **Zip(Original)**: <https://github.com/simplyi/ResourceServer/archive/master.zip>
* **Repository(Updated to Spring Boot 3.2.1):**[Git Repository](https://github.com/simplyi/ResourceServer/tree/master-spring-boot-3.2.1-update)
* **Zip(Updated to Spring Boot 3.2.1):**[Zip](https://github.com/simplyi/ResourceServer/archive/refs/heads/master-spring-boot-3.2.1-update.zip)

#### **Albums Resource Server**

**Before configuring it as Eureka Client**

* **Repository:** <https://github.com/simplyi/Albums-Resource-Server/tree/Albums-Resource-Server-Before-Eureka-Client>
* **Zip:** <https://github.com/simplyi/Albums-Resource-Server/archive/Albums-Resource-Server-Before-Eureka-Client.zip>

**After Configuring Albums Resource Server as Eureka Client**

* **Repository:** <https://github.com/simplyi/Albums-Resource-Server/tree/Eureka-Client>
* **Zip:**<https://github.com/simplyi/Albums-Resource-Server/archive/Eureka-Client.zip>
* **Repository:**[Spring Boot 3.2.1 update](https://github.com/simplyi/Albums-Resource-Server/tree/Eureka-Client-Spring-Boot-3.2.1)

#### **Photos Resource Server**

**Before configuring it as Eureka Client**

* **Repository:** <https://github.com/simplyi/Photos-Resource-Server/tree/Photos-Resource-Server-Before-Eureka-Client>
* **Zip:**<https://github.com/simplyi/Photos-Resource-Server/archive/Photos-Resource-Server-Before-Eureka-Client.zip>

**After configuring it as Resource Server**

* **Repository:** <https://github.com/simplyi/Photos-Resource-Server/tree/Eureka-Client>
* **Zip:** <https://github.com/simplyi/Photos-Resource-Server/archive/Eureka-Client.zip>

#### **API Gateway**

* **Repository:** <https://github.com/simplyi/ApiGateway>
* **Zip:** <https://github.com/simplyi/ApiGateway/archive/master.zip>

**API Gateway Before CORS Configuration**

* **Repository**: <https://github.com/simplyi/ApiGateway/tree/Before-CORSE-Configuration>
* **Zip:**<https://github.com/simplyi/ApiGateway/archive/Before-CORSE-Configuration.zip>

**API Gateway After CORS Configuration**

* **Repository:** <https://github.com/simplyi/ApiGateway/tree/CORS-Configuration>
* **Zip:**<https://github.com/simplyi/ApiGateway/archive/CORS-Configuration.zip>

#### **Discovery Service**

* **Repository:** <https://github.com/simplyi/DiscoveryService>
* **Zip:**<https://github.com/simplyi/DiscoveryService/archive/master.zip>

#### **Photo App Spring MVC Web Client Application**

* **Repository:** <https://github.com/simplyi/PhotoAppWebClient>
* **Zip:** <https://github.com/simplyi/PhotoAppWebClient/archive/master.zip>

#### **PKCE Example**

* **Repository:** <https://github.com/simplyi/PKCE>
* **Zip**: <https://github.com/simplyi/PKCE/archive/main.zip>

#### **Social Login/Logout**

* **Repository(Original):** <https://github.com/simplyi/SocialLoginWebClient>
* **Zip(Original):**<https://github.com/simplyi/SocialLoginWebClient/archive/master.zip>
* **Repository(Updated to Spring Boot v 3.1.2):**  [Repository](https://github.com/simplyi/SocialLoginWebClient/tree/master-update-to-Spring-3.1.2)
* **Zip(Updated to Spring Boot v 3.1.2):** [Zip](https://github.com/simplyi/SocialLoginWebClient/archive/refs/heads/master-update-to-Spring-3.1.2.zip)

#### **The PKCE-Enhanced Authorization flow in JavaScript application**

**JavaScript application**

* **Repository**: <https://github.com/simplyi/spa-example>
* **Zip:** <https://github.com/simplyi/spa-example/archive/main.zip>

**API Gateway**(before CORS configuration)

* **Repository:** <https://github.com/simplyi/ApiGateway/tree/Before-CORSE-Configuration>
* **Zip:** <https://github.com/simplyi/ApiGateway/archive/Before-CORSE-Configuration.zip>

**API Gateway**(after CORS configuration)

* **Repository:** <https://github.com/simplyi/ApiGateway/tree/CORS-Configuration>
* **Zip:** <https://github.com/simplyi/ApiGateway/archive/CORS-Configuration.zip>

#### **Resource Server**

**Resource Server**(before CORS configuration)

* **Repository:**<https://github.com/simplyi/ResourceServer/tree/CORS-configuration>
* **Zip:**<https://github.com/simplyi/ResourceServer/archive/CORS-configuration.zip>

**Resource Server**(after CORS configuration)

* **Repository(Original):**<https://github.com/simplyi/ResourceServer/tree/before-CORS-configuration>
* **Zip(Original):**<https://github.com/simplyi/ResourceServer/archive/before-CORS-configuration.zip>
* **Repository(Updated to Spring Boot 3.2.1):**[Git Repository](https://github.com/simplyi/ResourceServer/tree/master-spring-boot-3.2.1-update)
* **Zip(Updated to Spring Boot 3.2.1):**[Zip](https://github.com/simplyi/ResourceServer/archive/refs/heads/master-spring-boot-3.2.1-update.zip)

#### **Keycloak User Storage Provider SPI**

(Remote User Authentication)

**Remote User Storage Provider Implementation**

* **Repository**: <https://github.com/simplyi/KeycloakRemoteUserStorageProvider.git>
* **Zip:**<https://github.com/simplyi/KeycloakRemoteUserStorageProvider/archive/main.zip>

**Alternative to using RestEasy with Spring Boot 3.1+**

* **Repository**: <https://github.com/simplyi/KeycloakRemoteUserStorageProvider/tree/rest-easy-client-fix>
* **Zip**: <https://github.com/simplyi/KeycloakRemoteUserStorageProvider/archive/refs/heads/rest-easy-client-fix.zip>

**Legacy Users Web Service**

* **Repository**: <https://github.com/simplyi/LegacyUsersWebService.git>
* **Zip:**<https://github.com/simplyi/LegacyUsersWebService/archive/main.zip>

#### **The New Spring Authorization Server**

* **Repository:** <https://github.com/simplyi/NewSpringAuthorizationServer.git>
* **Zip:**<https://github.com/simplyi/NewSpringAuthorizationServer/archive/refs/heads/main.zip>
* [Authorization Code Grant Type](https://docs.google.com/presentation/d/1CiAiuay5rd1KDDnYwOyu6ud9xk5ZetSQDOMp9DYUKjs/edit?usp=sharing)
* [Client Credentials](https://docs.google.com/presentation/d/1KEA3i0F0bhB4me1uHfXkbmuaaFeRyxo7rG0ih-MlP68/edit?usp=sharing)
* [Password](https://docs.google.com/presentation/d/1kea9VCSP_QtQSb_NbU7MPOVLEF20iuOgieNW1g1MTwc/edit?usp=sharing)
* [Device flow](https://docs.google.com/presentation/d/1SlGr9z9bFIxYOLzwwZco3ny2W1XZ-9GJ9ORWA73gbjo/edit?usp=sharing)
* [PKCE-Enhanced Authorization code](https://docs.google.com/presentation/d/1yJeYPMoPY2050cZkkBcBu1SL5Z88StV7O7fH5f_6X3A/edit?usp=sharing)
* [Refreshing Access Token](https://docs.google.com/presentation/d/1e0bWzKk5JxxGXlAvAyeWe1q8iYgY804Y6vZ-zCpRAKU/edit?usp=sharing)

OAuth – Open authorization, ok what is opened because of this?

Oauth2 started in 2012

What is OAuth and Why OAuth?

OAuth, or open authorization, is a widely adopted authorization framework that allows you to consent to an application interacting with another on your behalf without having to reveal your password. It does this by providing access tokens to third-party services without exposing user credentials

The core OAuth 2.0 specification defines the "client password" (e.g. client secret) client authentication type, which defines the client\_secret parameter as well as the method of including the client secret in the HTTP Authorization header.

The end goal of Oauth is appln should get an access token from authorization server

OAuth vs. open id connect

Open id connect is built on top of OAuth which adds some user information while redirecting like user photo

OAuth generates only token Top of Form

OAuth provides a way for an application to get an access token to make API requests

Bottom of Form

OAuth is all about accessing api with token open id connect is about identifying the users, means OAuth

OAuth doesn’t provides the application with information about the user’s identity. That’s what OpenID Connect was created to add into the framework.

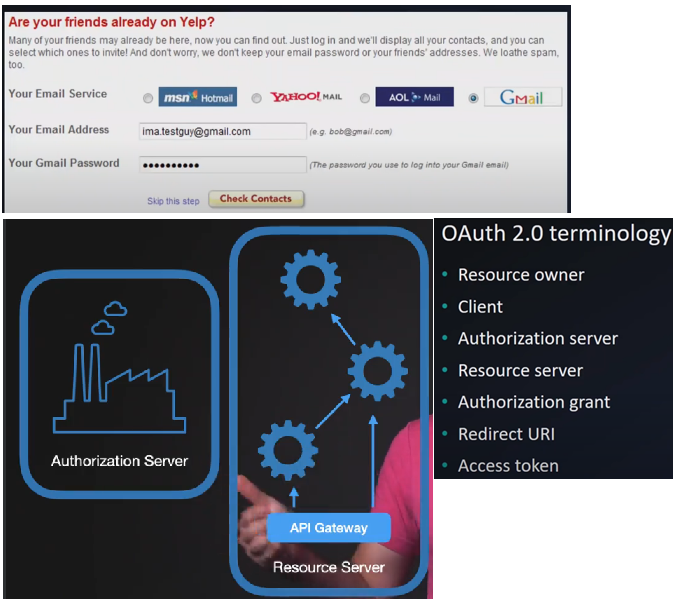
# Before OAuth

Goal of OAuth- 3rd party apps also wants to access the data ,if ticket new app wants data from google, we should not give our google password to ticket new

Before oauth we are supposed to give our credentials to random applications as below

In 2009 .. when 1 appn wants to talk to another application like if ticketnew app wants to get all user details that ticketnew application will take the **gmail user name and password** (which is very bad they can misuse the users Gmail id credentials ) and that application will talk to google and get all his information like name, age, mail id, .. ex:- apps book my show

So the main objective of oauth is not to share the password of google with ticketnew..



## OAuth 2.0 terminologies

**Resource owner**:- who owns the data it is the person who has the data , ex:- These are the rooms that are accessible by the person who has the card

If you are building an API, using an external service as your Authorization Server, which roles are you building? –Ans resource server

**Authorization server** :- Is one who provides authorization token or access token after validating our credentials& our ticket new api will use our token issued by auth server to access the google api

The user will type the passwords only at Authorization server ex:- google auth server

**Analogy**: - when we go to hotel for stay they will verify our payment and they will give us a card, with that card we can access rooms, swimming pool, play area..

As like this card, we will receive the token from auth server once our credentials are verified, we can give that token to 3rd party applications like ticketnew and they will use that token and ask google for their contacts or photos

Ex:- if ticket new application wants to connect our google friends list, then unlike old way we will give our credentials to google oauth server he will verify our credentials(here google server is authorization server)

and give us the token we can give that token to ticket new application

Resource server:- where the resource is like google contacts api

Google contacts API is reusable secured API, if u want to use it authorization is mandatory, and

Similarly if u want to develop an reusable service ur api also should implement security and generally we get many requirements to consume an app on which oauth is enabled

**User agent /OAuth client** – the person or device who is making the request or

**THE API** – is the place where data lives or to whom we request

**Client** – it is nothing but a 3rd party application like ticket new app who wants to get our friends information from google oauth server

Client Secret- it is the password which will be given to the app after registration, it will be given only to confidential clients who can maintain passwords

Public clients: These applications are incapable of maintaining those client secrets like passwords/secret key and these apps are incapable they won’t/shouldn’t maintain any client secrets

If u are writing single page application ur app should not maintain any secret, if u maintain, user can hack that client secret key which will be misused

Note: [PKCE](https://oauth.net/2/pkce/) is not a form of client authentication, and is not an alternative to client authentication. Applications using client authentication should also use PKCE.

Ex:- Mobile apps don't have the ability to be deployed with a client secret, since users of the app would be able to extract the secret. Therefore they are considered public clients.

Mobile apps don’t have the ability to be deployed with credentials, so they are considered public clients. The users of the app are not significant to this distinction.

Confidential clients: Confidential clients are those clients which are capable to maintain the client secrets,

These clients have some passwords like Client secret/private key JWT / MTLS or these client like java applications can maintain the client secrets

Confidential clients authenticate when making requests to the OAuth authorization server.

For first party confidential client authorization server may skip the consent screen because only trusted client can hit the application with the client secret

For confidential clients consent screen is skipped

Ex:- java, .net application can maintain client secrets effectively

Front Channel

Also called implicit flow

Sending data in url like the way which is visible

Using the browser’s address bar to move data between two other pieces of software is using the front channel.

Ex:- we should never send the oauth token in front channel (url )

window.location = 'https://authorization-server.com/authorize?client\_id=example';

here in this way data is visible in url here this is front channel and it is less secured way

Back Channel

Sending the data/oauth token through a program or through headers &

it is possible even in java script also

For the purposes of OAuth, back channel means a secure HTTP connection from any HTTP client to a server Any HTTP client that makes a request to an HTTP server is using the back channel, even if that client is JavaScript code in a browser

Back channel is always secured

fetch("https://authorization-server.com/", {

method: "GET",

headers: {

"Content-Type": "application/json"

}

})

.then((response) => {

return response.json();

});

Here we are sending data in headers hence this is not visible in url so this is called back channel

Client Id

Every appln is uniquely identified by its client id, once we register we will get the client id from authorization server, It is the public information

While authorization we should send our client id, based on that only it will show that particular app’s consent screen to the user

**App registration**

While registration we should give our redirect url so that auth server after authentication they will redirect to our specified url with the auth token

On successful registration we will get our client id (which will be unique to identify any client) and client secret (password)

### What is MTLS

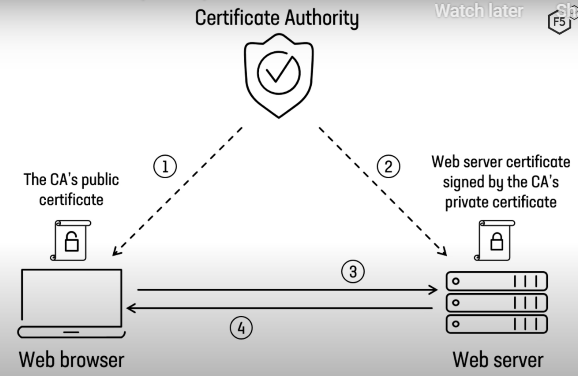
SSL, TLS both are technology used to encrypt the data communication

MTLs- mutual TLS means both server and client will verify other’s certificate to confirm whether they are talking to right applications or not

So in postman when u are hitting an MTLS URL u need to attach certificate

Because both server and client needs to verify others certificate

MTLS (Mutual transport layer security) which is a technology establishes an encrypted connection in which both parties use x509 digital certificates to authenticate and verify each other



Forms of client authentication

Here client means application who wants the resource

These are most common forms of client authentication.

* [Client Secret](https://www.rfc-editor.org/rfc/rfc6749#section-2.3.1) (RFC 6749 Section 2.3.1)
* [Mutual TLS](https://oauth.net/2/mtls/) (RFC 8705) [MTLS](https://tools.ietf.org/html/rfc8705) is a form of [client authentication](https://oauth.net/2/client-authentication) and an extension of OAuth 2.0
* [Private Key JWT](https://oauth.net/private-key-jwt/) (RFC 7521, RFC 7523, OpenID)

1. Using client secret-> Clients Applications sending the client secret to authorization server is not more secure practice (this is most common as this is easy to use as this is even static key),

Ex: Client should send the secret in header as below

Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

Or

lient\_id

REQUIRED. The client identifier issued to the client during

the registration process described by [Section 2.2](https://www.rfc-editor.org/rfc/rfc6749#section-2.2).

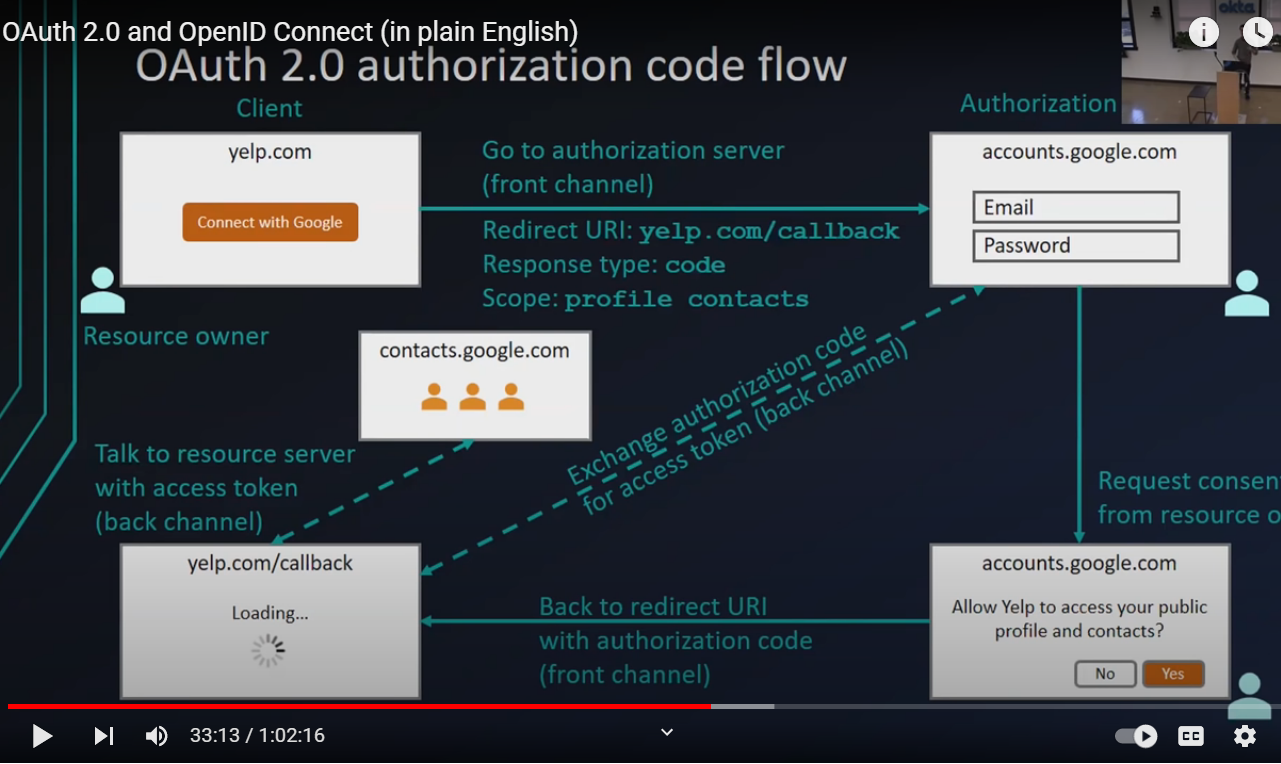
client\_secret

REQUIRED. The client secret. The client MAY omit the

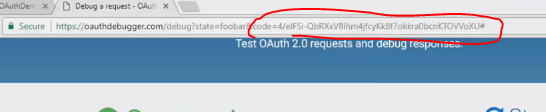
parameter if the client secret is an empty string.

These will be passed as the headers & authorization servers are responsible to decode the key

1. You should use the private key and public key concepts using Mutual TLS (MTLS) or a private key to sign a JWT



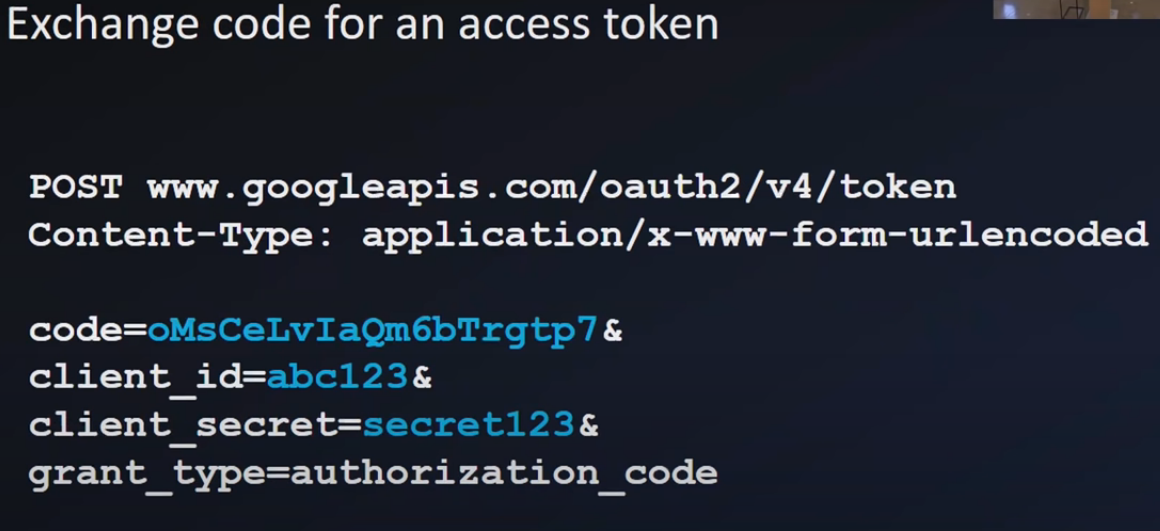
When user provides consent we will get authorization code in the browser address bar, not access token, we will hit again with auth code to get access token and with that we will hit resource server to get contacts



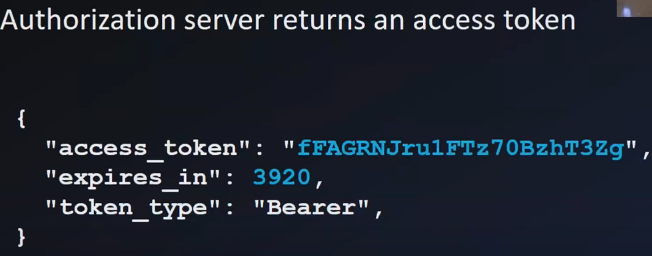
## Procedure to get the access token

* 1. Once user provided his credentials on consent screen, then after authentication oauth server will send auth code in front channel (url/address bar which is not confidential)
  2. With
  3. )auth code and
  4. unique client identifier(for each client) and
  5. with his client secret &
  6. pkce identifier/unique hash code for that flow

With all these details hit the auth server again and get the auth token in back channel (secret information in headers)



Once u got the auth code from the url address bar and hit google api from backend and get the access token,



It has expiration time, as this is valid only for some time and this is valid only for some url like whatever the user have gave the consent if user gave consent to read access to contacts, he can only access them he can’t delete their contacts

When u are completely using front end – react alone or javascript alone then u will directly get the access token in the url I think , because with javascript we may not be able to call backend server to get the access token from the auth code\

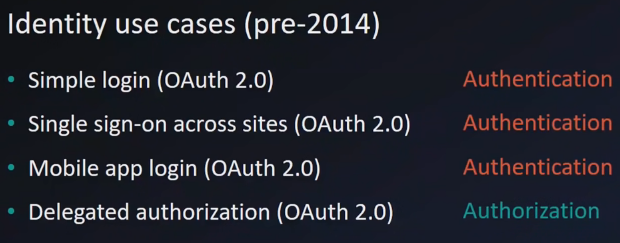
## Use cases of OAUTH

Now auth is used for both authentication and authorization

Like login with facebook , means

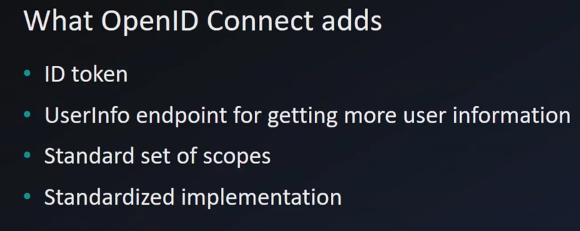
Authorization use case:- to use face book contacts first login and provide the consent and allow access to use contacts

Oauth wasn’t designed to use for authentication, its was designed for authorization only



Now these days they are using open ID connect for authentication and oauth for authorization

OPEN Id connect is just an extension to oauth



If I am talking to an oauth server which can understands open ID connect

I will get access token to talk to resource server and ID token ,,but what s the use of this

And use this access token and hit user info end point and get more details about the use for authorization