**JSON Web Token:** Token á JSON format-i.. base-64 encoded (eyJHbGci).. sent fram og til baka. Skiptist í þrjá hluta.. header (json object).. hvaða algorithmi.. type.. Jason Web Token. Svo kemur næsti hluti.. það eru gögnin... { ‚sub‘: ‚1234567890‘ //userId } (þetta er CLAIM) svo getum við bætt inn þeim gildum sem okkur langar.. gæti verið t.d. name og admin. Síðasti hlutinn (SIGNATURE) er dulkóðaður með þessum algorithma.. með secret sem authentication server-inn veit af. (Sjáum hvort fiktað hafi verið við þetta með þessu).

Historically:

* Client sends request to web app, containing username and password
* Web app looks up the two, if match, creates a **session key** and stores it in a **response cookie**
* Subsequent requests will then use the **cookie** and the contained **session key** to find out who the user is

First, web apps/services SHOULD NOT MAINTAIN THEIR OWN LIST OF USERS AND THEIR CREDENTIALS..

**NOT ALL CLIENTS USE A COOKIE**

**Oauth (AUTHORIZE, NOT AUTHENTICATE)**

Valet key – LIMITED ACCESS..

New websites often offer services which tie together functionality from other sites..

Should not have to give username and password everywhere.. you just expose it to more people..

**Oauth** allows the you as a user, to **GRANT ACCESS TO YOUR PRIVATE RESOURCES ON ONE SITE, TO ANOTHER SITE**..

**OpenID: All about using a SINGLE IDENTITY to SIGN INTO MANY SITES.. Oauth is about GIVING ACCESS to your stuff without sharing your identity at all..**

Oauth:

Say you upload photos to facebook, make them private. (Jane is the **resource owner**, and facebook the **server**). The photos are the **protected resources**.

Wants to share photos with her grandmother, but not the bottle of Scotch.

Uses Beppa, a photo printing service. Beppa is the **client**.

Since photos are private, **Beppa must use Oauth** to gain access to the photos..

A Beppa developer obtained a set of **client credentials (CLIENT IDENTIFIER** and **SECRET**) from Facebook to be used with Facebook‘s Oauth-enabled API.

Beppa requests from Facebook a set of **temporary credentials**. Not resource-owner-specific, can be used by Beppa to gain **resource owner approval** from Jane to access her private photos.

When beppa receives the temp credentials, it redirects Jane to the Facebook OAUth User Authorization URL with the temporary credentials and asks Facebook to redirect Jane back once approval has been granted to: <http://beppa.com/print>

On Facebook she needs to sign into the site.. **Oauth requires that servers first authenticate the resource owner**, and then ask them to grant access to the client..

Oauth allows Jane to **TO NOT SHARE HER USERNAME/PASSWORD WITH ANY OTHER SITE**..

After logging in, Jane is asked to grant access to Beppa, **the client**. Facebook tells her who is requesting access (Beppa) and the **type of access**.. Jane can approve or deny.

Jane makes sure Beppa is getting the **limited access** it needs. Does not want to allow Beppa to change her photos for instance. One time access.. good for one hour..

Once Jane approves.. Facebook marks the **temporary credentials** as **resource-owner-authorized**. She is redirected back to beppa, together with the **temporary credentials identifier**. Beppa can now continue to fetch Jane‘s photos..

Beppa uses the **authorized Request Token** and exchanges it for an **Access Token**.

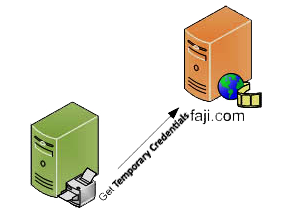
Request Tokens => Good for obtaining User approval.

Access Token => access Protected Resources.

1. **Exchange Request Token for Access Token**
2. **Request the photo (or photos)**

So.. overview.

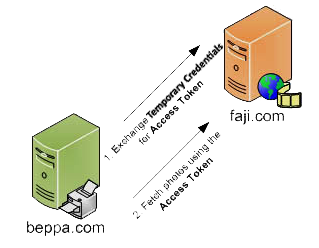
Beppa has client credentials with Facebook.. (**client identifier and secret**).



These need resource-owner-approval.. aren‘t resource-owner specific.

So next.. redirects to Facebook.. where Jane (the user) approves the **type of access**.. then redirect back..

Facebook then tells Beppa who the user is (Jane) and the type of access. I.e. makes them **resource-owner-authorized**.. sends it back with the temp credentials identifier.



The temporary crendetials were good for **obtaining user approval**..

Need Access Token to access protected resources.

Next request would be for the photos.

**OpenID Connect**

OpenID Connect 1.0 is a simple **identity layer** on top of the Oauth 2.0 protocol.

Enables clients to **verify the identity** of the End-User based on the **authentication performed by an Authorization Server**, as well as to obtain **basic profile info** about the End-User.. in an interoperable and REST-like manner.

Uses REST/JSON..

OpenID lets developers **authenticate** their users across websites and apps without having to own and manage password files.

(Identity, Authentication) + Oauth 2.0 = OpenID Connect

**OpenID Connect basically ADDS AUTHENTICATION on top of Oauth**

Oauth defined access tokens.. these tokens were used to access for example, some info from facebook/google. However.. **OpenID** defined **IDENTITY TOKENS** which are used to **identify the user**.. for example to access an application.

OpenID follows the following steps:

1. The Client sends a request to the OpenID Provider
2. The OpenID Provider **authenticates** the end-user and obtains authorization
3. The OpenID Provider responds with an **ID token** and usually an **Access Token**
4. The Client can send a request with the Access Token to the UserInfo endpoint
5. The UserInfo Endpoint returns **Claims** about the end-user

**OpenID Connect basic specs**

**Flows**

The OpenID Connect authentication process.. requires the user to authenticate him/herself through a OpenID provider. This process then returns the info about the user in the form of an **id token**, which contains **scopes** and **claims**, as well as other useful info about the user being authenticated.

The way in which this process is conducted is referred to as a flow.. the main flows are:

* Code Flow
* Implicit Flow
* Hybrid Flow

Flow is specified through the response\_type param in the authorization request.

**Code Flow**

1. The client sends an **authentication request** to the OpenID provider.
2. The OpenID provider **authenticates the user** (on behalf of the client) and responds with an **authorization code** to the client.
3. The client requests an **id token** with the authorization code.
4. The client **validates** the token and **retrieves the user info.**

**Implicit Flow**

1. The client sends an **authentication request** to the OpenID provider.
2. The OpenID provider **authenticates** the user (on behalf of the client) and **redirects him/her to the client with an id token**
3. The client validates the token and **retrieves the user info**

**Hybrid Flow**

This flow is basically a mix of the two, means in some cases the authorization process is returned to the client with a code and sometimes with an id token directly.

**Request params:**

* Client\_id: Client identifier
* Response\_type: the response type. Like ‚id-token‘ or token.
* Scope: Represents the ACCESS or information request. F.ex. openid, profile, email, read, or write
* Redirect\_uri: Location for where the response will be sent
* Response\_mode: Defines the Form Post Response Mode
* State: Maintaining a state between the client and provider
* nonce: This is used to associate a client session with an id token

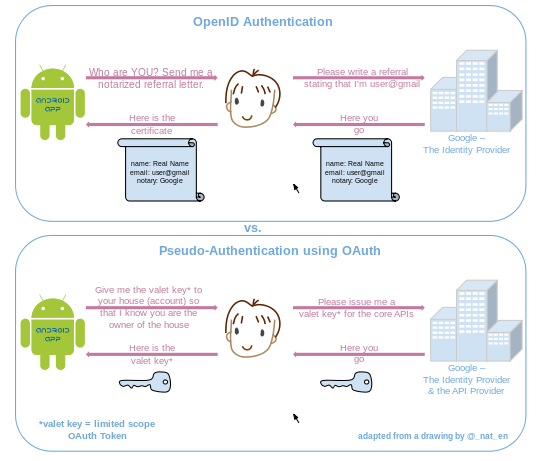
**ID token**

Contains info about the authenticated user in the form of a **claim**. Id tokens as represented in **JSON Web Token** format. Some of the claims contained within an id token are the following:

* iss : Represents the issuer of the response (url)
* sub: This is the subject identifier.. a unique identifier that represents the user.
* Aud: This is the audience which is a client identifier
* Exp: Expiration date of the id token
* Iat: Represents the issue date
* Nonce: Used to associate a client session with an id token
* At\_hash: This represents the hashed acces token

**Claims**

The OpenID connect also defines several sets of claims.



Þekkja:

Munur á Oauth og OpenId: Oauth er bara authorization..

Authorization vs Auth

Flow: Implicit Flow.. viljum ekki að third-party app fái beinan aðgang að lykilorðinu okkar.

Þekkja hvernig JSON Web Token lítur út..