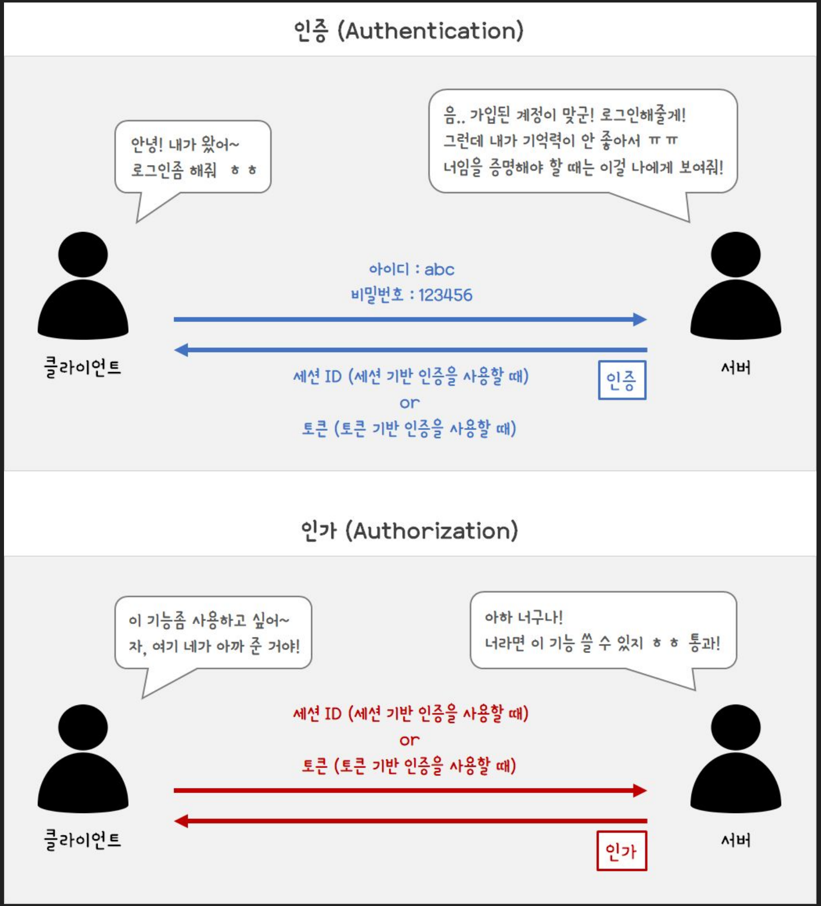
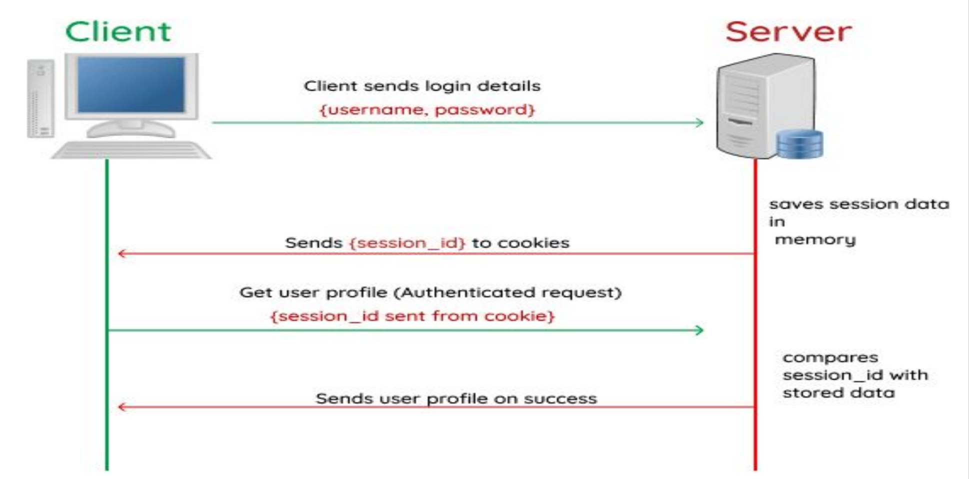
Authentication and Authorization

COSKA AWS Study

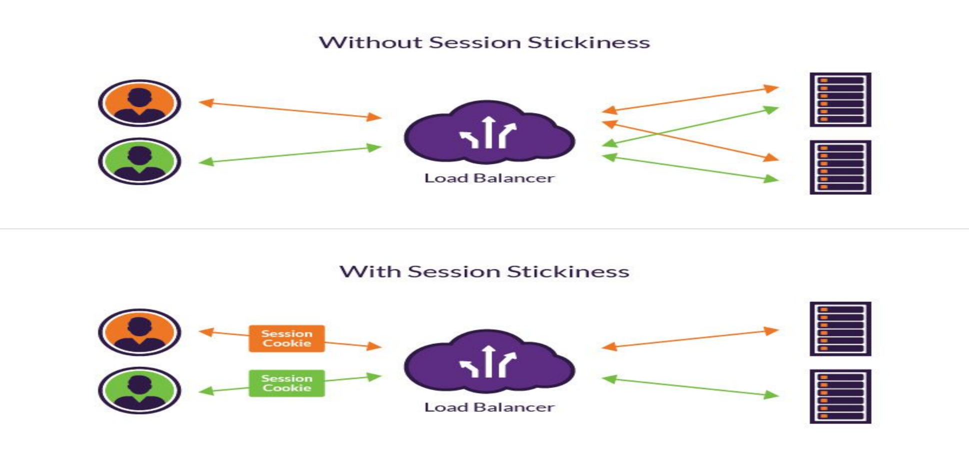
Authentication?

Authentication is the process of verifying the identity of a user, device, or system to ensure that only authorized access is granted to protected resources. This is typically done by requiring the user to provide a password or some other form of authentication token, which is compared against a database of authorized users. The goal of authentication is to prevent unauthorized access to sensitive information and ensure the confidentiality, integrity, and availability of data and systems.

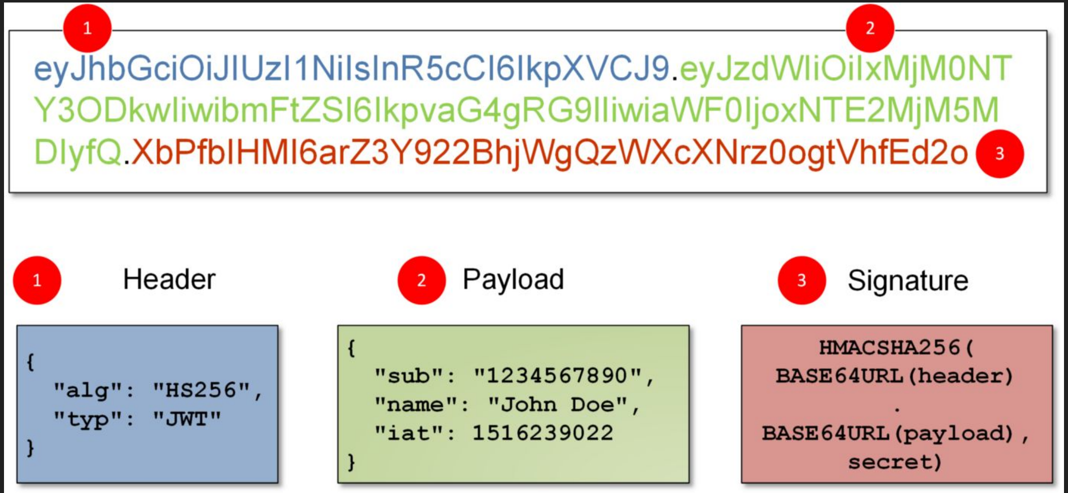

인증 (Authentication)
안녕! 내가 왔어~
로그인좀 해줘 ㅎㅎ
음.. 가입된 계정이 맞군! 로그인해줄게!
그런데 내가 기억력이 안 좋아서 ㅠㅠ
너임을 증명해야 할 때는 이걸 나에게 보여줘!
아이디 : abc
비밀번호 : 123456
클라이언트
서버
세션 ID (세션 기반 인증을 사용할 때)
or
토큰 (토큰 기반 인증을 사용할 때)
인증
인가 (Authorization)
이 기능좀 사용하고 싶어~
자, 여기 네가 아까 준 거야!
아하 너구나!
너라면 이 기능 쓸 수 있지 ㅎㅎ 통과!
세션 ID (세션 기반 인증을 사용할 때)
or
토큰 (토큰 기반 인증을 사용할 때)
클라이언트
서버
인가


Client
Server
Client sends login details
{username, password}
saves session data in memory
Sends {session_id} to cookies
Get user profile (Authenticated request)
{session_id sent from cookie}
compares session_id with stored data
Sends user profile on success

Session based authentication



- cookie, session, (local storage)
- password encryption
- MFA
- session id
- request header
- session verification
- session extend
- server resource, session hijacking
- session timeout, https
- delegated server
- sticky session
- session sync
- clustering
- failover
- session storage
- elasticache (redis)
- token, jwt (jason web token)

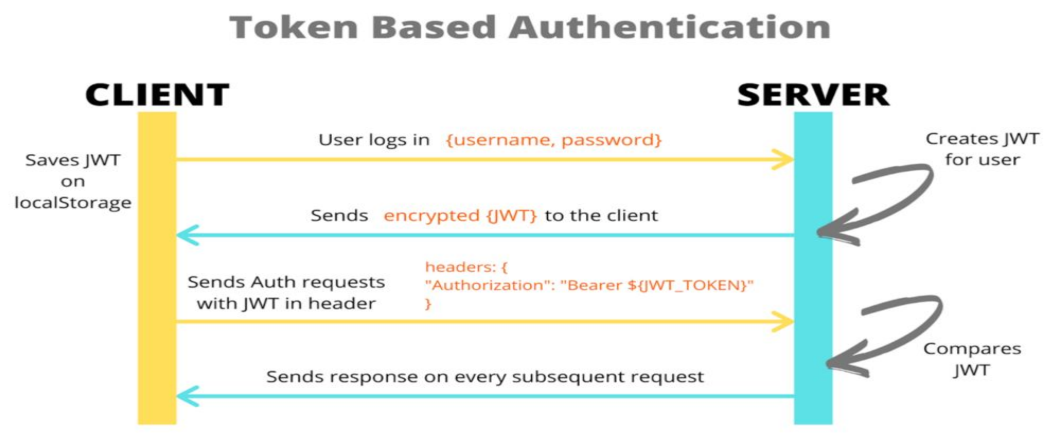


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NT
Y3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5M
DIyfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

```
HMACSHA256(
BASE64URL(header)
.
BASE64URL(payload),
secret)
```

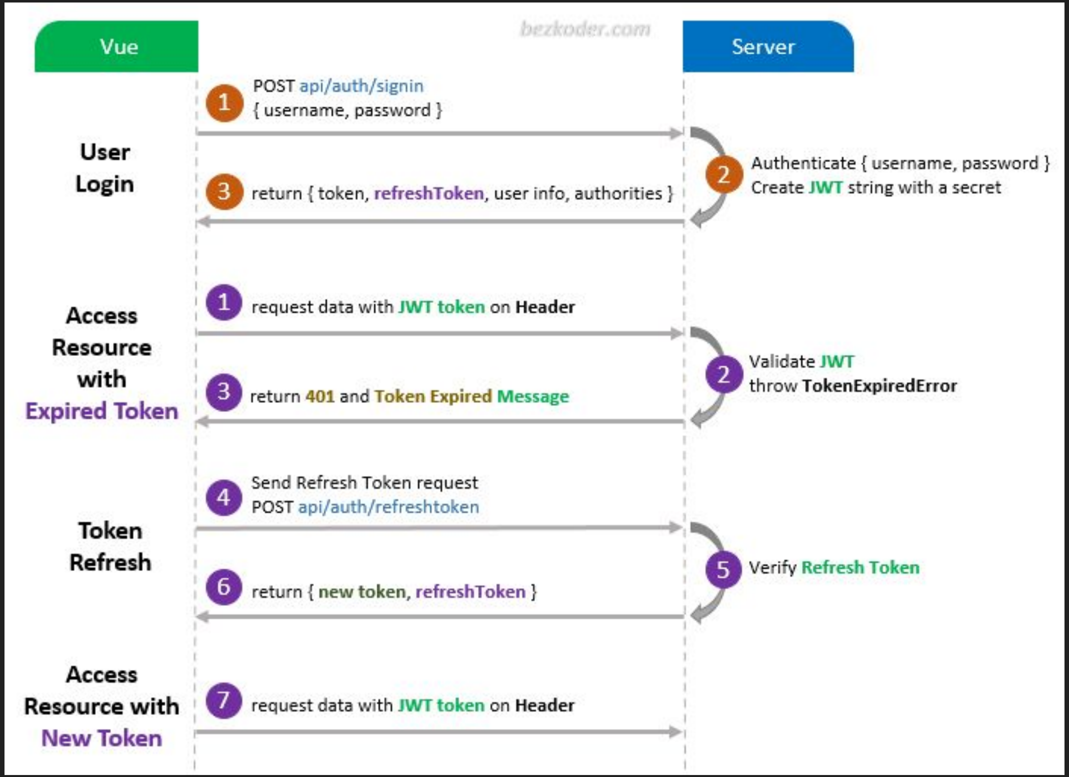


Token Based Authentication
CLIENT
SERVER
Saves JWT on localStorage
User logs in {username, password}
Creates JWT for user
Sends encrypted {JWT} to the client
Sends Auth requests with JWT in header
headers: {
"Authorization": "Bearer ${JWT_TOKEN}"
}
Compares JWT
Sends response on every subsequent request

JWT (Json Web Token)

XXXXXXX.YYYYYYY.ZZZZZZZ
header.payload.signature

- jwt in cookie or local storage
- https://jwt.io/
- payload (user info)
- session (user info in session storage, server)
- no storage
- stateless (server scale out)
- revocable (session reset)
- expiration time
- refresh token (safe)
- openai.com token



bezkoder.com

Vue

Server

User Login

1 POST api/auth/signin
{ username, password }

2 Authenticate { username, password }
Create JWT string with a secret

3 return { token, refreshToken, user info, authorities }

Access Resource with Expired Token

1 request data with JWT token on Header

2 Validate JWT
throw TokenExpiredError

3 return 401 and Token Expired Message

Token Refresh

4 Send Refresh Token request
POST api/auth/refreshtoken

5 Verify Refresh Token

6 return { new token, refreshToken }

Access Resource with New Token

7 request data with JWT token on Header

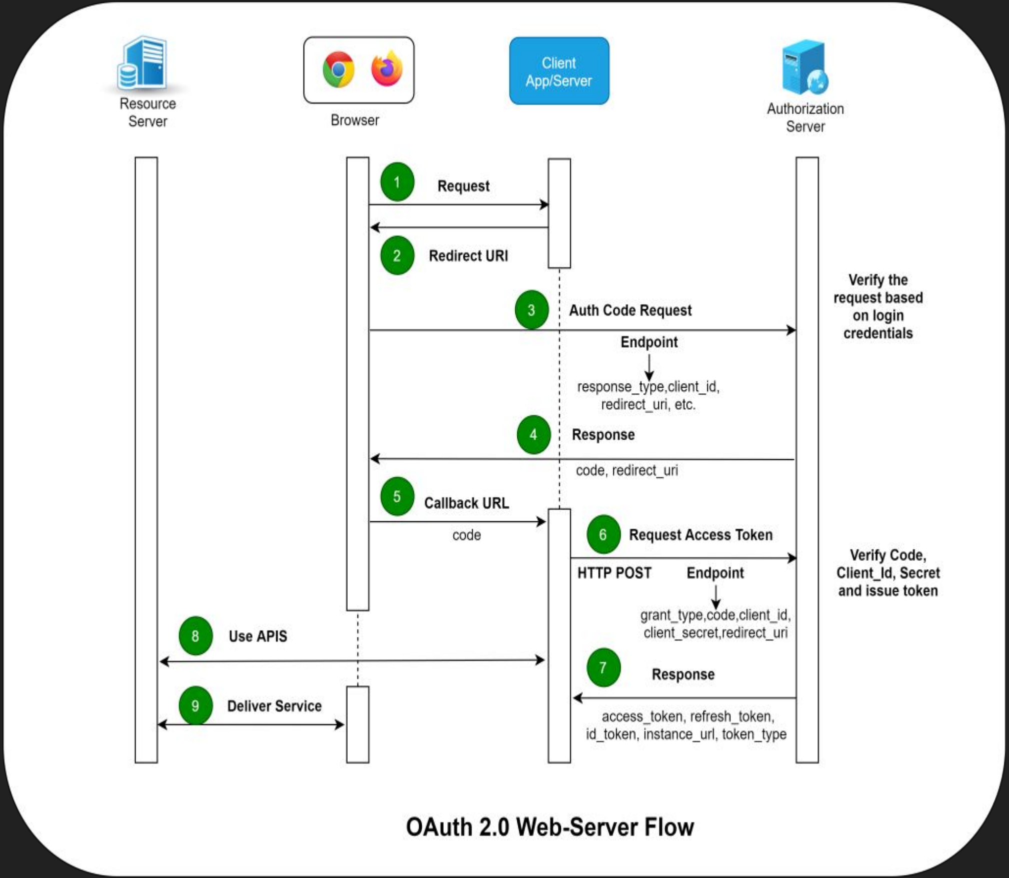
- expired token (front end)
- refresh token in DB

SSO

Single Sign-On (SSO) is a process that allows a user to access multiple applications or services with just one set of login credentials (username and password), without having to log in again for each separate application. The user is authenticated once, and the authentication is then securely shared across all applications and systems that the user wants to access. The goal of SSO is to provide a seamless and secure user experience, reduce the number of passwords that users need to remember, and increase security by reducing the number of times a user has to enter their password.

- one time login
- ID / password (forgot, reset)
- saves IT team time
- improves end-user experience, for both employees and customers
- makes your systems more secure, and decreases attack surface

Oauth / SAML / OIDC


Resource Server
Browser
Client App/Server
Authorization Server
1 Request
2 Redirect URI
3 Auth Code Request
Endpoint
response_type,client_id, redirect_uri, etc.
Verify the request based on login credentials
4 Response
code, redirect_uri
5 Callback URL
code
6 Request Access Token
HTTP POST
Endpoint
grant_type,code,client_id, client_secret,redirect_uri
Verify Code, Client_Id, Secret and issue token
7 Response
access_token, refresh_token, id_token, instance_url, token_type
8 Use APIS
9 Deliver Service

OAuth 2.0 Web-Server Flow

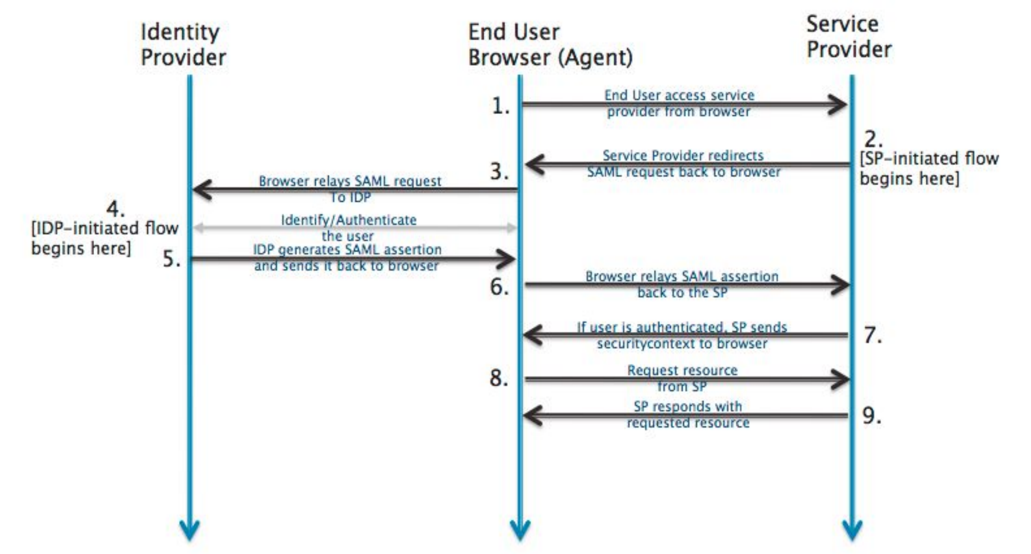
Oauth

OAuth (Open Authorization): OAuth is an open standard for authorization, allowing users to grant third-party applications access to their resources (e.g., a user granting a to-do list app access to their Google calendar) without having to share their login credentials. It uses a token-based approach and is typically less centralized than SAML, relying on a combination of an authorization server, resource server, and client.

- openai.com signup
- clash of clans (facebook connect, friends)
- mobile
- https
- authorization
- authentication (user login to Facebook)
- access token, refresh token

- authorization code grant (response_type=code)
- implicit grant
- resource owner password credentials grant
- client credentials grant

- openai.com (google oauth)

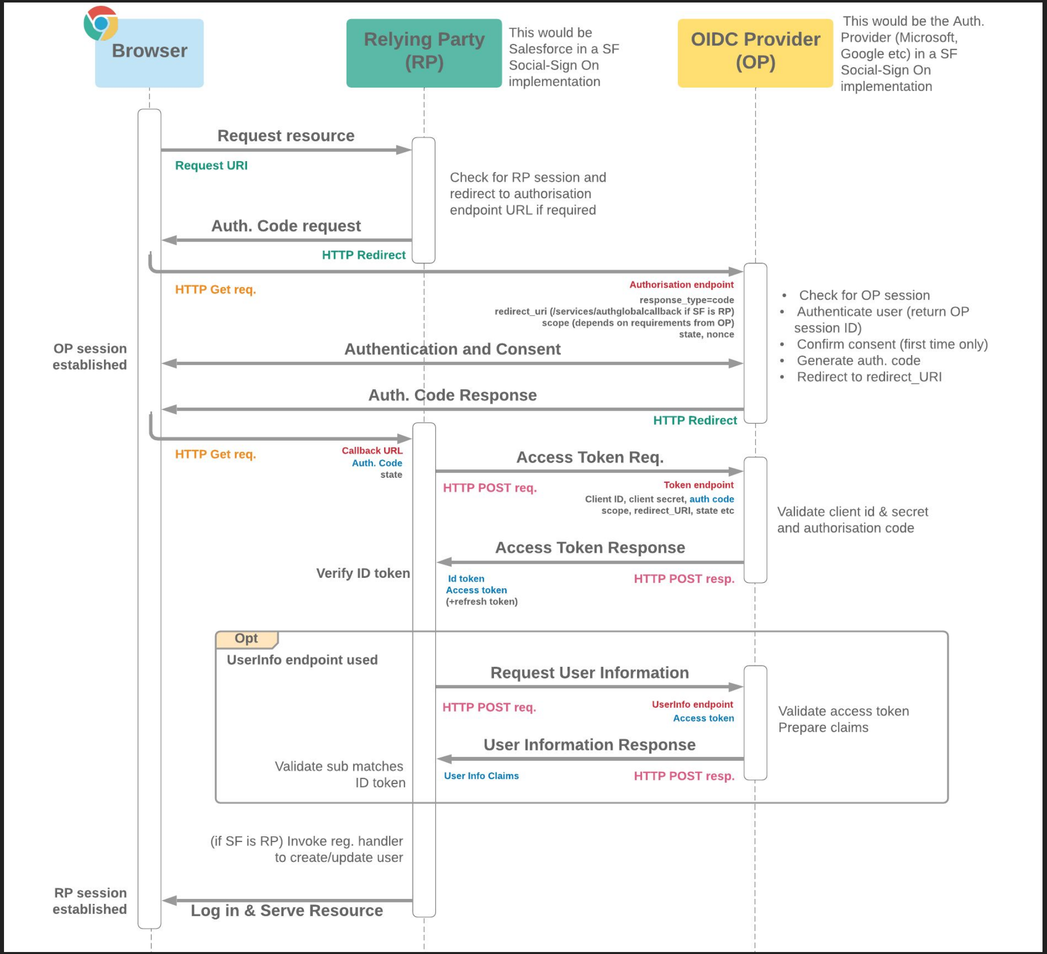

Identity Provider
End User Browser (Agent)
Service Provider
1.
End User access service provider from browser
2.
[SP-initiated flow begins here]
3.
Service Provider redirects SAML request back to browser
Browser relays SAML request To IDP
4.
[IDP-initiated flow begins here]
Identify/Authenticate the user
5.
IDP generates SAML assertion and sends it back to browser
6.
Browser relays SAML assertion back to the SP
7.
If user is authenticated, SP sends securitycontext to browser
8.
Request resource from SP
9.
SP responds with requested resource

SAML

Security Assertion Markup Language (SAML) is an XML-based open standard for exchanging authentication and authorization data between parties, in particular, between an identity provider (IdP) and a service provider (SP). It enables SSO by allowing a user to authenticate with an IdP and then securely pass the authentication information to an SP, allowing the user to access multiple applications without having to log in again. SAML defines the structure and format of the authentication and authorization data and provides a secure way for an IdP to pass this information to an SP. It is widely used in enterprise and cloud environments, and is supported by many software products, including web browsers, identity management systems, and cloud-based applications.

- multiple applications (coska chat, coska booking)
- without oauth (google / facebook)

OIDC



Browser

Relying Party (RP)

This would be Salesforce in a SF Social-Sign On implementation

OIDC Provider (OP)

This would be the Auth. Provider (Microsoft, Google etc) in a SF Social-Sign On implementation

Request resource

Request URI

Check for RP session and redirect to authorisation endpoint URL if required

Auth. Code request

HTTP Redirect

HTTP Get req.

Authorisation endpoint
response_type=code
redirect_uri (/services/authglobalcallback if SF is RP)
scope (depends on requirements from OP)
state, nonce

- Check for OP session
- Authenticate user (return OP session ID)
- Confirm consent (first time only)
- Generate auth. code
- Redirect to redirect_URI

OP session established

Authentication and Consent

Auth. Code Response

HTTP Redirect

HTTP Get req.

Callback URL
Auth. Code
state

Access Token Req.

HTTP POST req.

Token endpoint
Client ID, client secret, auth code
scope, redirect_URI, state etc

Validate client id & secret and authorisation code

Access Token Response

Verify ID token

Id token
Access token
(+refresh token)

HTTP POST resp.

Opt

UserInfo endpoint used

Request User Information

HTTP POST req.

UserInfo endpoint
Access token

Validate access token
Prepare claims

User Information Response

Validate sub matches ID token

User Info Claims

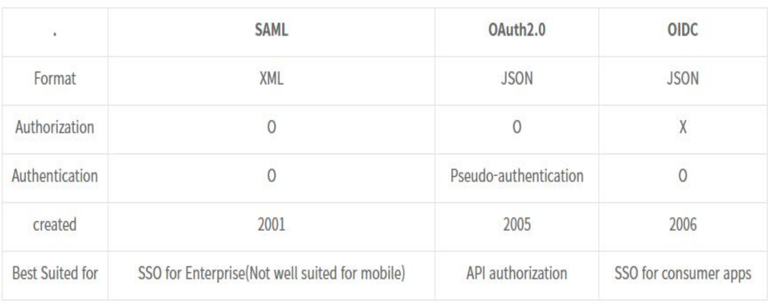
HTTP POST resp.

(if SF is RP) Invoke reg. handler to create/update user

RP session established

Log in & Serve Resource

OpenID Connect is an open standard for authentication that is built on top of OAuth 2.0. It provides a simple and secure way for users to authenticate with an identity provider (IdP) and then access resources at a service provider (SP) without having to repeatedly enter their credentials. OpenID Connect extends OAuth 2.0 by adding an identity layer that provides user information to the client in the form of an ID token. The ID token contains information about the user, such as their name and email address, and is signed by the IdP to prove its authenticity. OpenID Connect is designed to be easy to use and integrate into modern applications and is supported by many popular identity providers and cloud-based services.



.	SAML	OAuth2.0	OIDC
Format	XML	JSON	JSON
Authorization	O	O	X
Authentication	O	Pseudo-authentication	O
created	2001	2005	2006
Best Suited for	SSO for Enterprise(Not well suited for mobile)	API authorization	SSO for consumer apps