zühlke
empowering ideas

Get intimate with OAuth

Milan Starčević | 02.11.2023 | Heapcon

Agenda

01 | The Why? (The problem OAuth solves)

02 | Learn the names (Terminology)

03 | Going with the flow (How to choose an OIDC Flow)

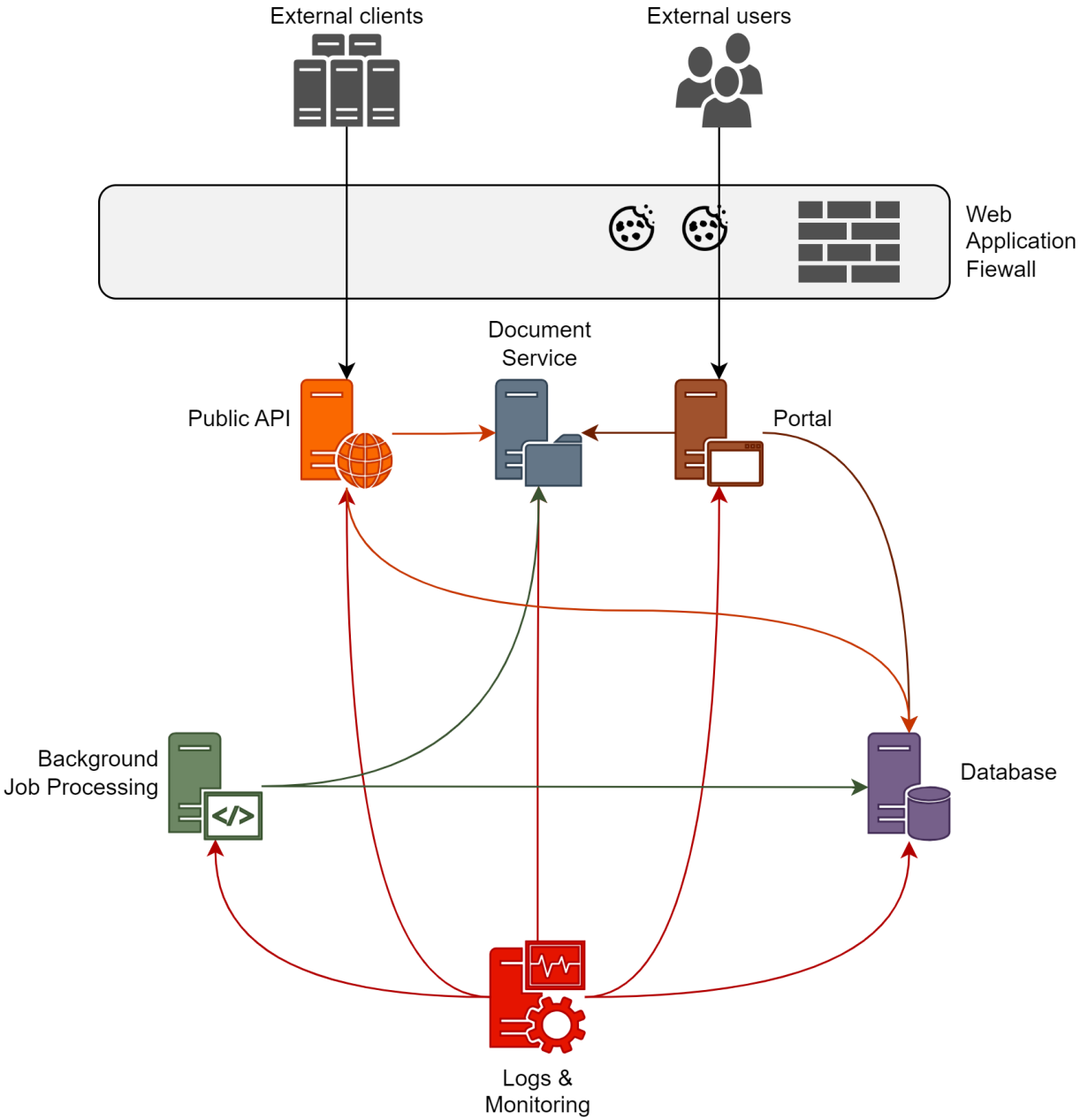
04 | Mind the gap (Implementing and common pitfalls)

01

The Why

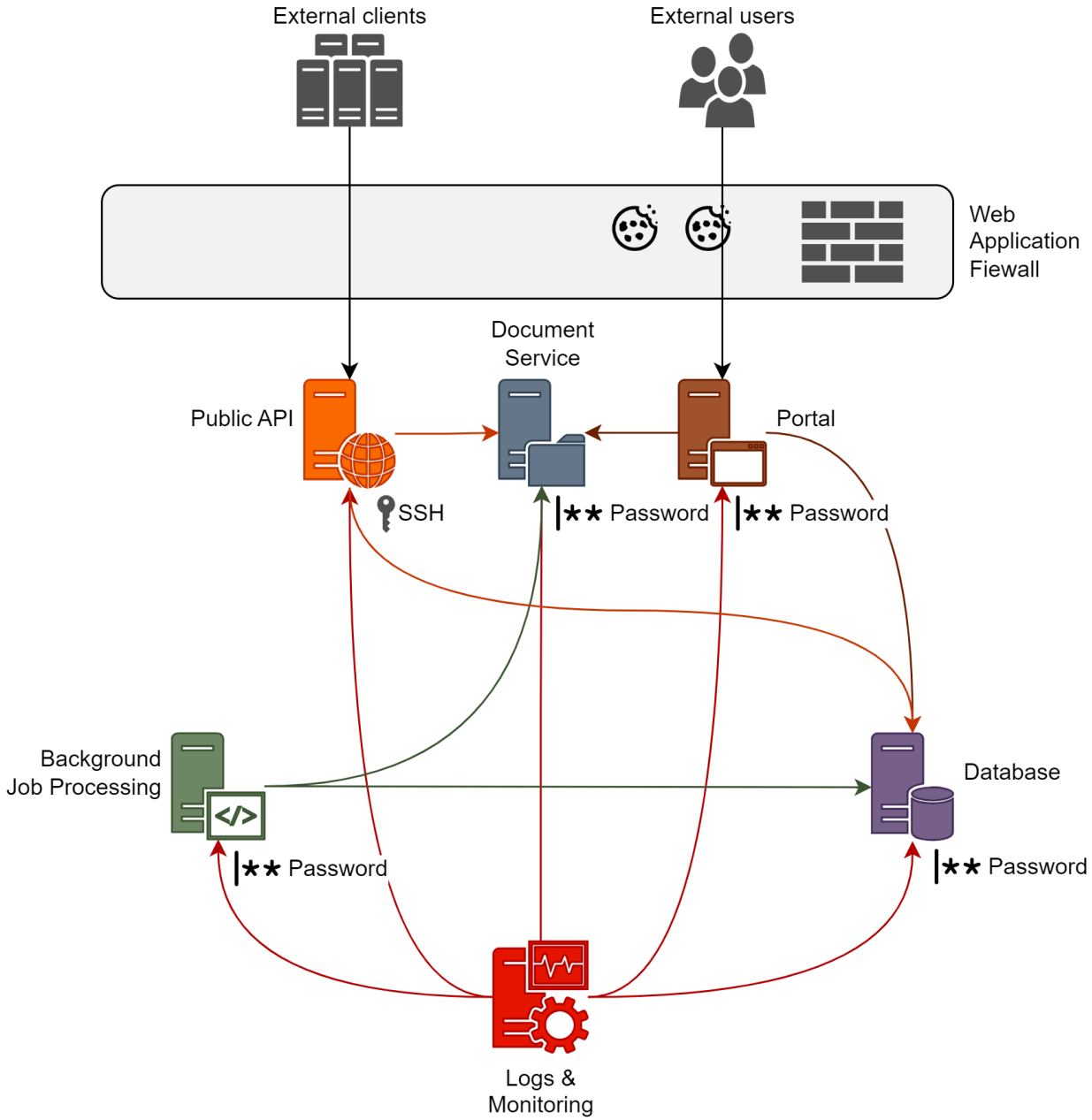
The problem OAuth solves

System


External clients
External users
Web
Application
Fiewall
Document
Service
Public API
Portal
Background
Job Processing
Database
Logs &
Monitoring

Authentication

1. Each component must authenticate its clients in order to trust them
2. Different authentication mechanism

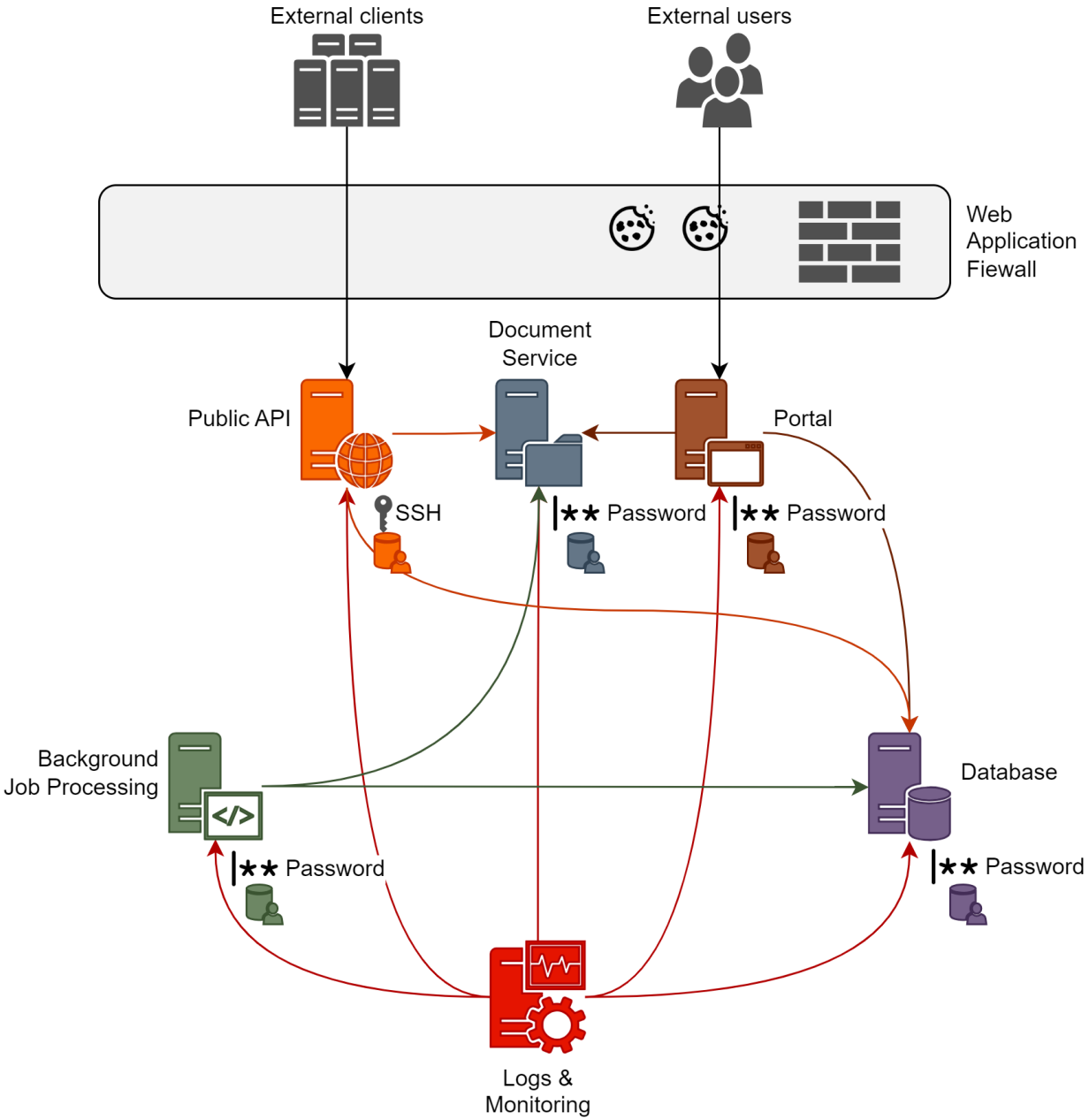


No single source of truth

Each component has to keep a separate identity stores.

It has to know them in order to trust them.

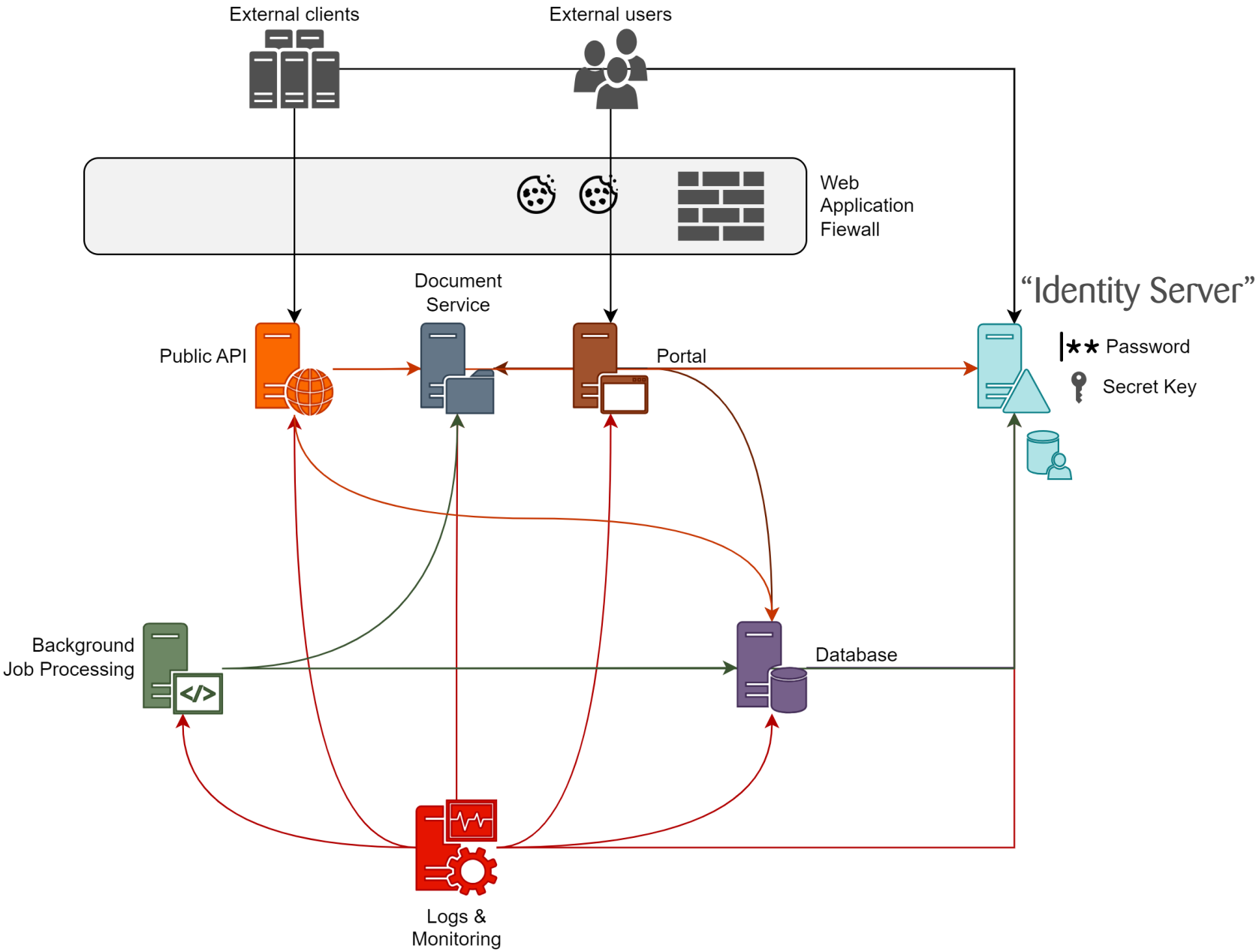
And it has to exchange secrets with them.



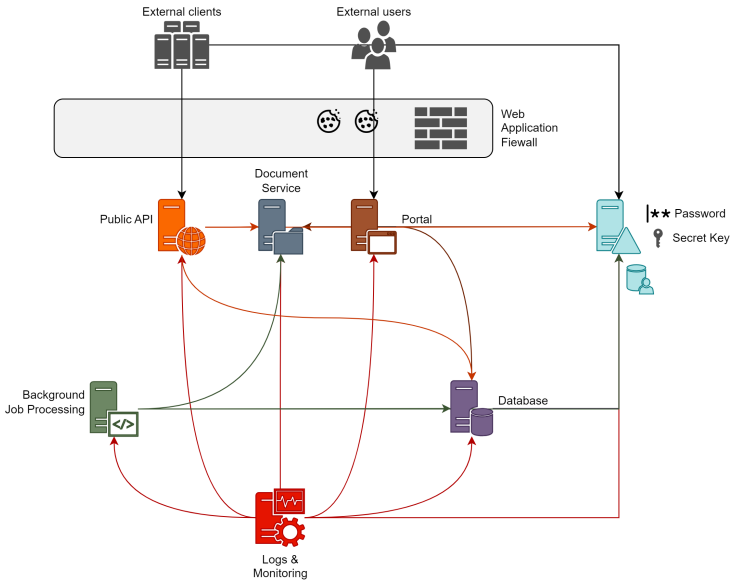
Single responsibility

Let's introduce a **single component for authentication.**

You now have a **single identity store.**



Single sign-on (SSO)



SSO is an authentication scheme that allows a user or server to **sign-on once** and then with a **single identity** access many software systems.

It can be achieved with OAuth, Kerberos, SAML.

Example: I login to Google account and then use this to get authorized in other services.

Reference

Comparison of SSO protocols

For application-level SSO of web and mobile use OIDC

Criteria	OAuth/OpenID Connect	SAML	Kerberos (also with LDAP SASL Bind)
Description	Identity layer on top of OAuth 2.0 that verifies end-user identity.	Allows identity providers to pass authorization credentials to service providers.	Network authentication protocol using secret-key cryptography.
Best Use Case	**Web** and **mobile** apps with Single Sign-On across multiple services.	**Enterprise** and **B2B** scenarios with Single Sign-On across multiple services.	Controlled **corporate** or **campus** environments for **network-level** authentication.
Data Format	JSON	XML	Binary
Commonly Used In	**Web** and **mobile** applications.	**Enterprise** web applications	**Windows** Active Directory, **Unix** networks
Cons	Not suitable for network-level authentication.	More complex to implement, XML makes it less suitable for mobile and web apps.	Complexity in managing and setting up Kerberos servers, limited to controlled environments.

02

Learn the names

Terminology

Identity

OpenID Connect (OIDC)

JWT

Authentication

OAuth

Claims

Authorization

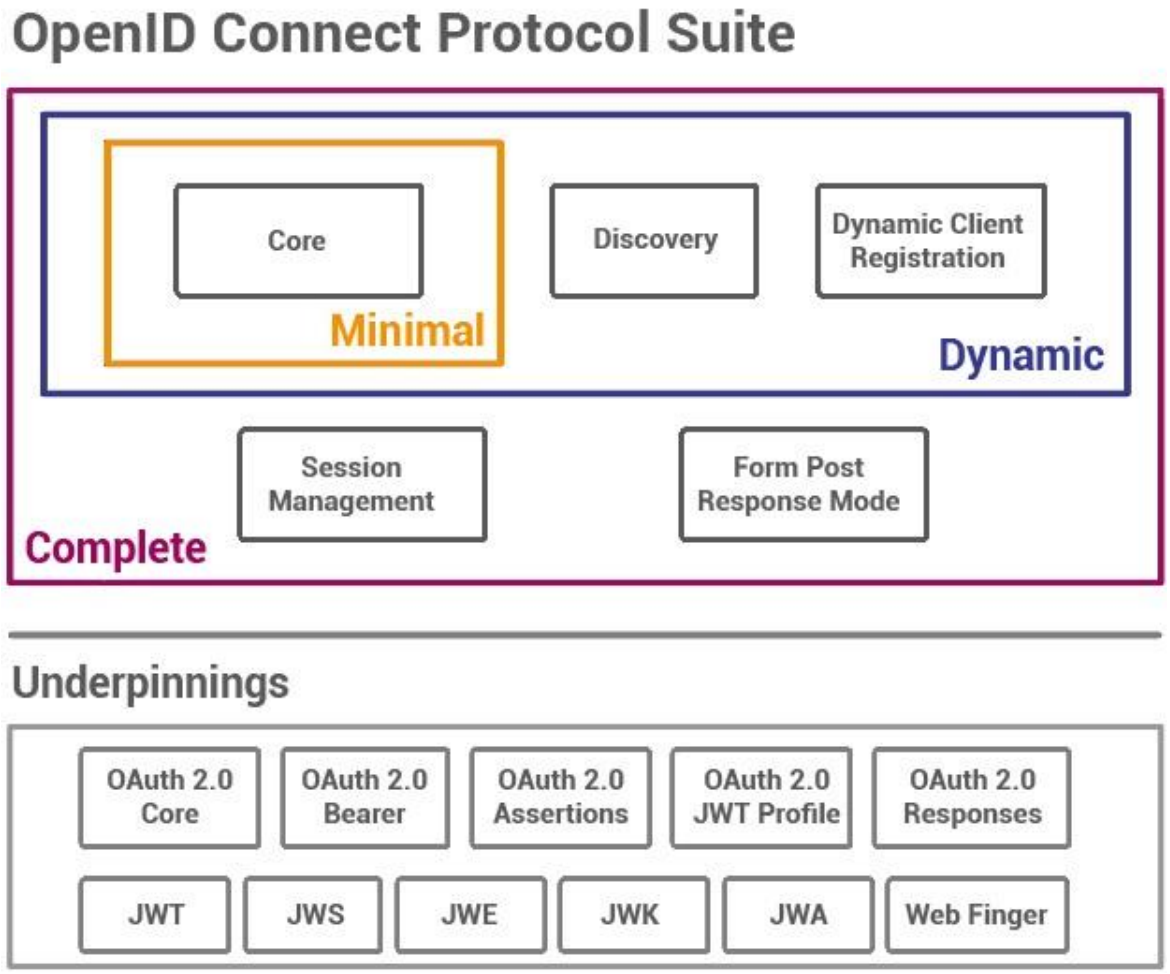
Access Token

Scopes

Single sign-on (SSO)

Bearer Token

OAuth VS OpenID Connect



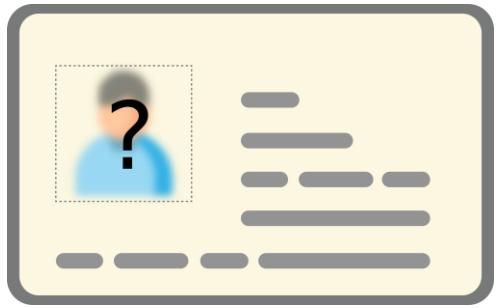
Authentication VS Authorization

Identity and Access Management (IAM)

Authentication

validates that users are whom they claim to be. It establishes their **Identity**.

Not valid → Status Code: 401

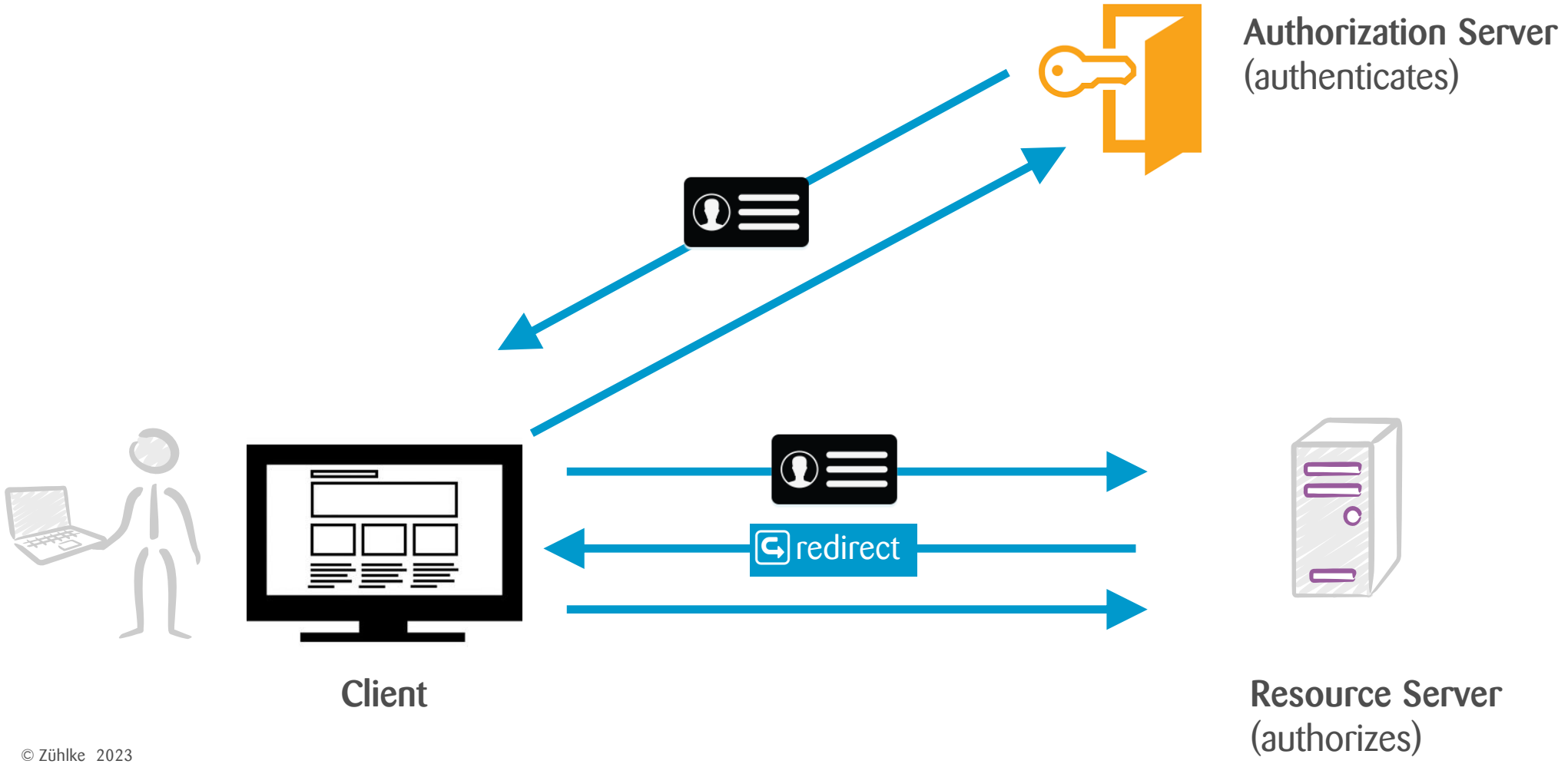


Authorization (Access Management)

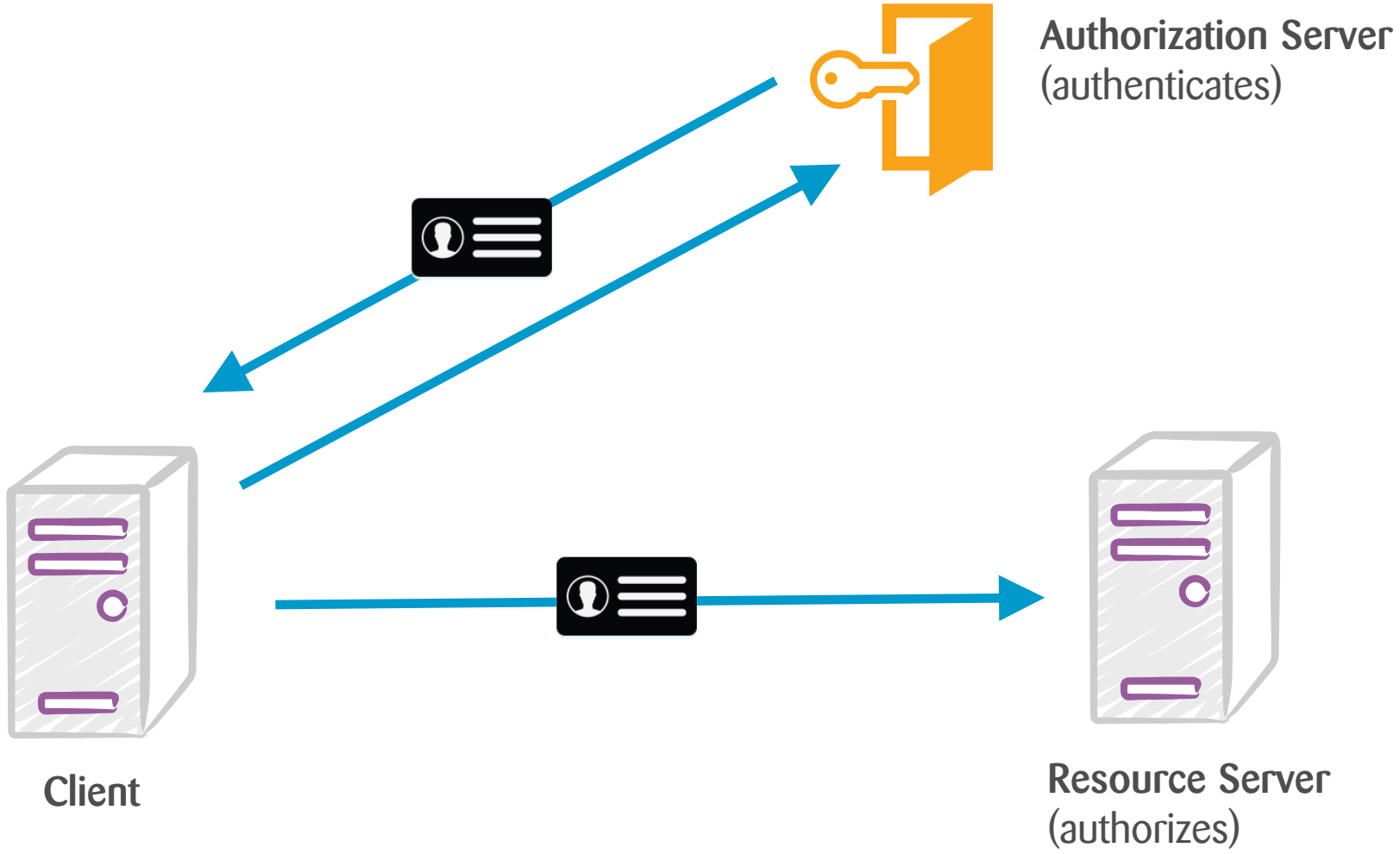
determines if the user has **permission** to access a specific resource or function.

Not permitted → Status Code: 403

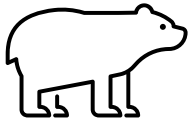
How does it work? Concept with a human.



How does it work? Concept without a human (just machines).

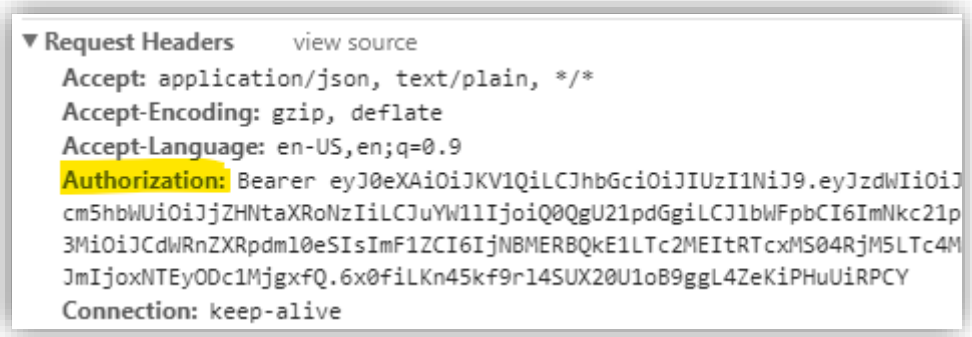


Bearer Token

Not  but 

A **Bearer Token** is an opaque string, not intended to have any meaning to clients using it.

Some tokens are a short string of hexadecimal characters, others may be structured such as JSON Web Tokens.



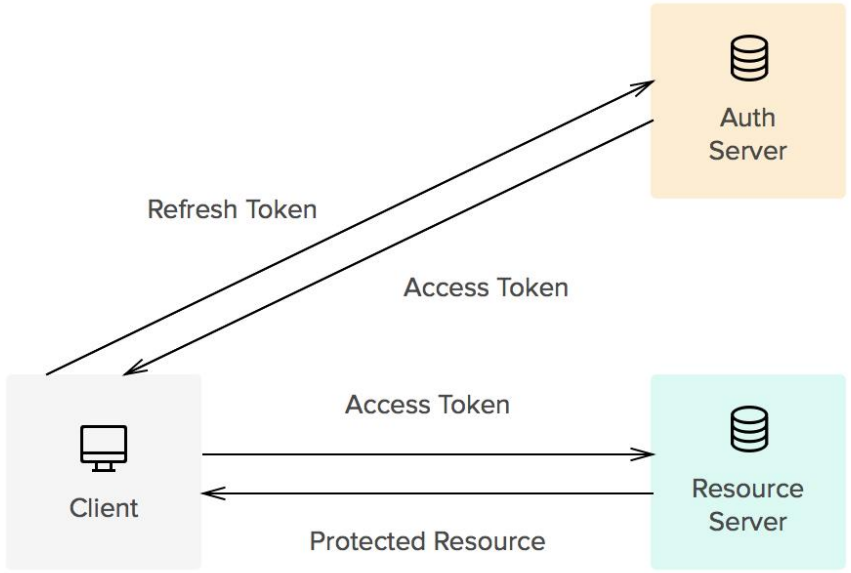
Bearer Token

OpenId Connect token recognizes three types

- **access_tokens** (used by resource servers to **authorize or deny requests**)
- **id_token** (used by client applications to **identify users**)
- **refresh_token** (used by client applications to **get new token**)

Refresh token

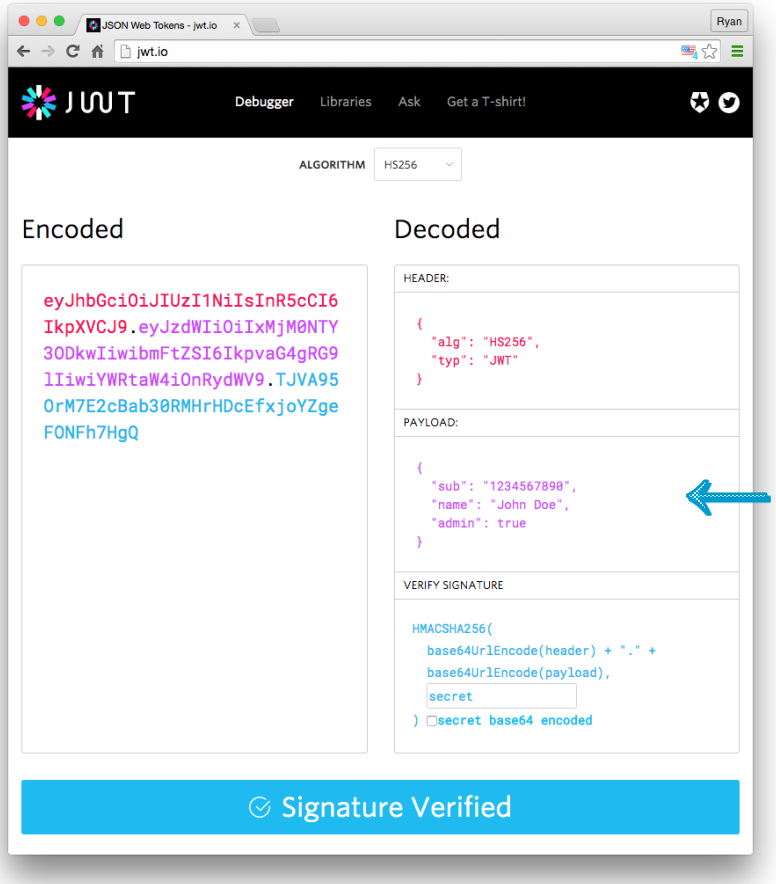
Refresh tokens carry the information necessary to get a new access token.



ID/Access	Refresh
Short lived	Long lived
Expires after period of inactivity	Lives during whole session
Storage less secure	Storage more secure
Used for auth	Used for getting new access token

JSON Web Tokens (JWT) – format

JSON Web Token is a way to encode claims in a JSON document that is then signed.
Claims give information to the Resource Server about the Client.



Claims

Claims are name/value pairs that contain **information about a user**, and meta-information about the OIDC service.

The "**sub**" (subject) claim identifies the **identity** that is the subject of the JWT.

```
{
    "family_name": "Silverman",
    "given_name": "Micah",
    "locale": "en-US",
    "name": "Micah Silverman",
    "preferred_username": "micah.silverman@okta.com",
    "sub": "msilver",
    "updated_at": 1490198843,
    "zoneinfo": "America/Los_Angeles"
}
```

Scopes

Scopes are space-separated lists of identifiers used to specify what claims are being requested by the **client**.

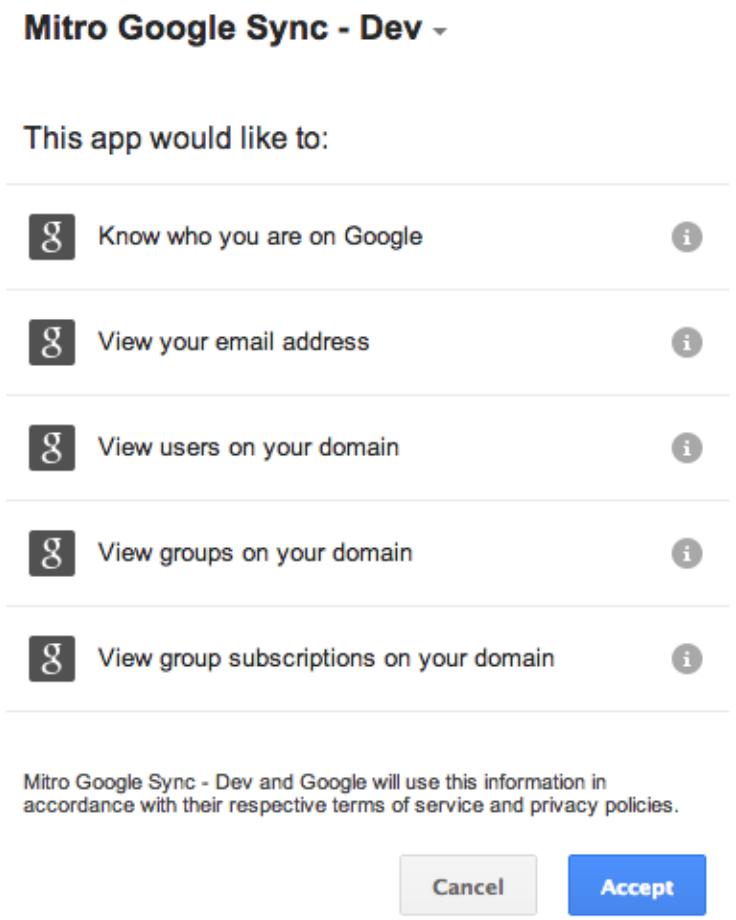
Identity Server will grant them if this **client** is allowed to know them.

scope	purpose
profile	requests access to default profile claims
email	requests access to email and email_verified claims
address	requests access to address claim
phone	requests access to phone_number and phone_number_verified claims

The default **profile** claims are:

- name
- family_name
- given_name
- middle_name
- nickname
- preferred_username
- profile
- picture
- website
- gender
- birthdate
- zoneinfo
- locale
- updated_at

Consent form



03

Going with the flow

How to choose an OIDC Flow / Grant

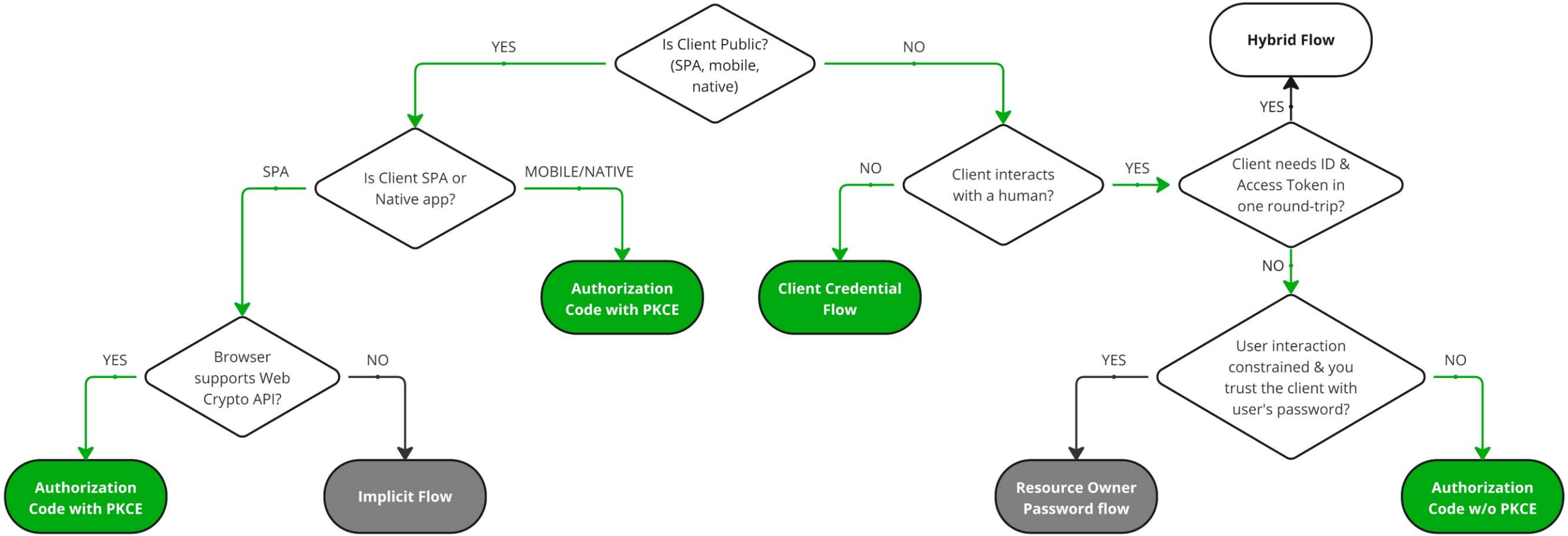
Flows

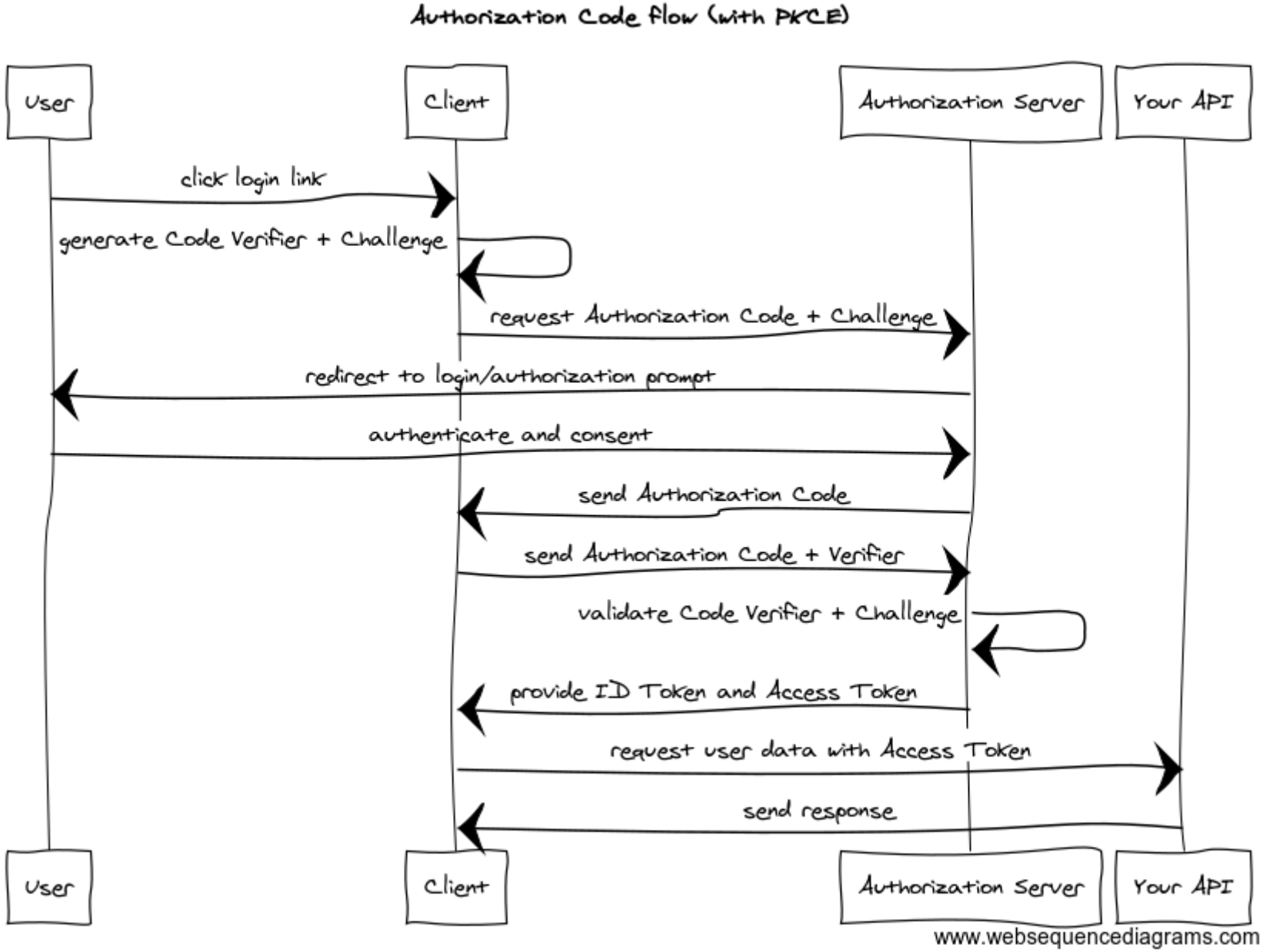
The OAuth 2.0 supports several different **flows** (or **grants**).

Flows are ways of retrieving an Access or Identity Token.

The selection of the flow depends

- application type
- level of trust for the client
- experience you want your users to have…


Is Client Public? (SPA, mobile, native)
YES
NO
Is Client SPA or Native app?
SPA
MOBILE/NATIVE
Authorization Code with PKCE
Browser supports Web Crypto API?
YES
NO
Authorization Code with PKCE
Implicit Flow
Client interacts with a human?
NO
YES
Client Credential Flow
Client needs ID & Access Token in one round-trip?
YES
Hybrid Flow
NO
User interaction constrained & you trust the client with user's password?
YES
NO
Resource Owner Password flow
Authorization Code w/o PKCE


Authorization Code flow (with PKCE)
User
Client
Authorization Server
Your API
click login link
generate Code Verifier + Challenge
request Authorization Code + Challenge
redirect to login/authorization prompt
authenticate and consent
send Authorization Code
send Authorization Code + Verifier
validate Code Verifier + Challenge
provide ID Token and Access Token
request user data with Access Token
send response
User
Client
Authorization Server
Your API
www.websequencediagrams.com

04

Mind the gap

Implementing and common pitfalls

What do you need for implementation?

A library per Actor

* All 3 components can use differnt technologies. OIDC is a specification - **technology agnostic**.

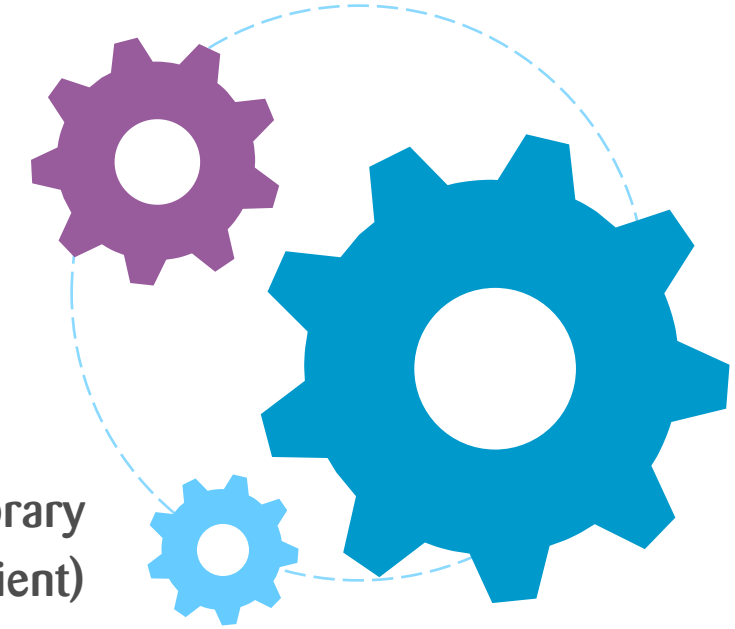
Authorization Server

Auth0

Identity Server

Azure AD

Google Identity



Backend library (for Resource Server or Client)

SPRING: spring-security-oauth2-client

.NET Core: IdentityModel

Frontend library (for Client)

angular-auth-oidc-client

oidc-react

AppAuth-Android

Flutter: openid_client

Implementation libraries

OpenID Connect keeps a list of all libraries:

https://openid.net/developers/libraries/

Configuring

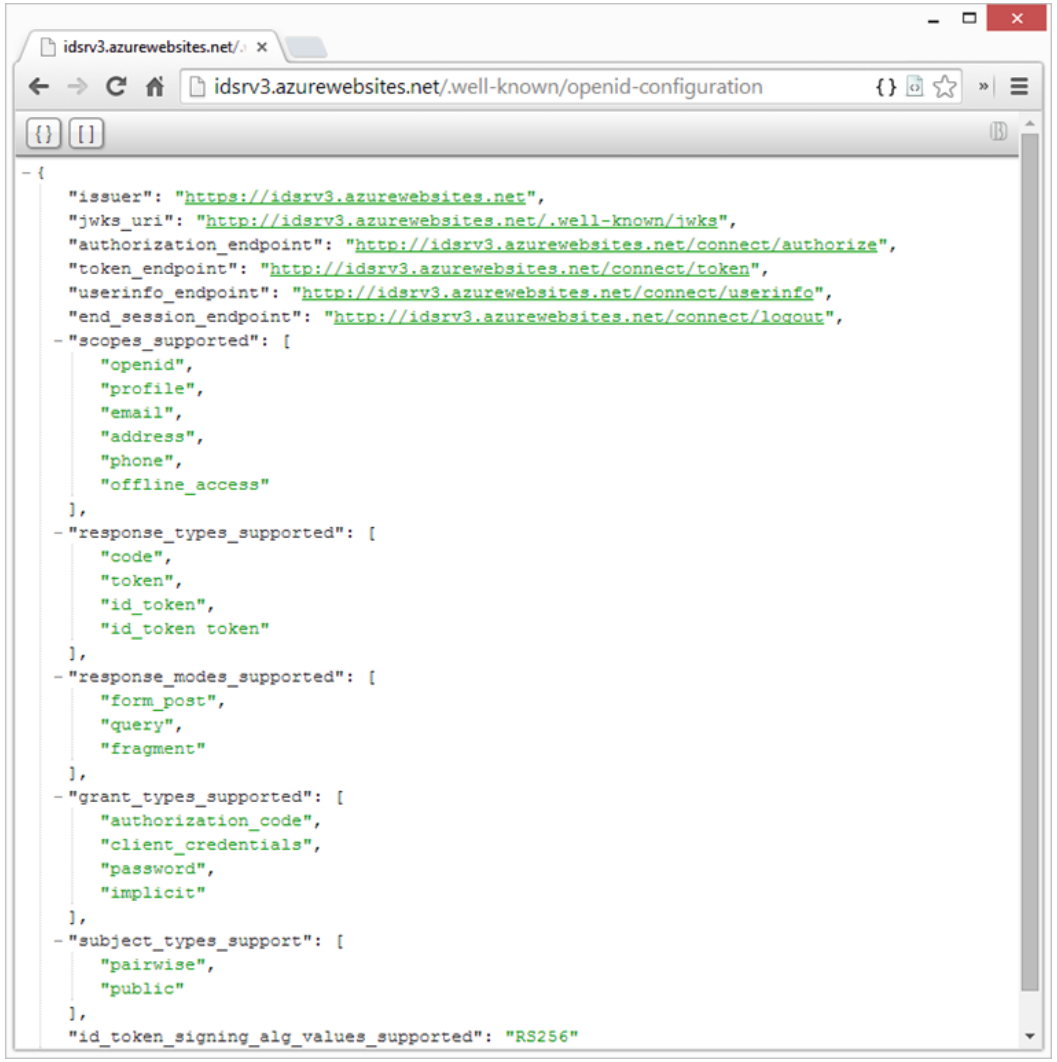
Depends on the Flow/Grant

Read the chosen library docs for that flow!

Resources Server and Client Server can auto configure using the Discovery endpoint that the Authorization Server will provide

Example: `https://server.com/.well-known/openid-configuration`

.well-known/openid-configuration



Common mistakes 1/2

…and how to avoid them

Learn the HTTP basics

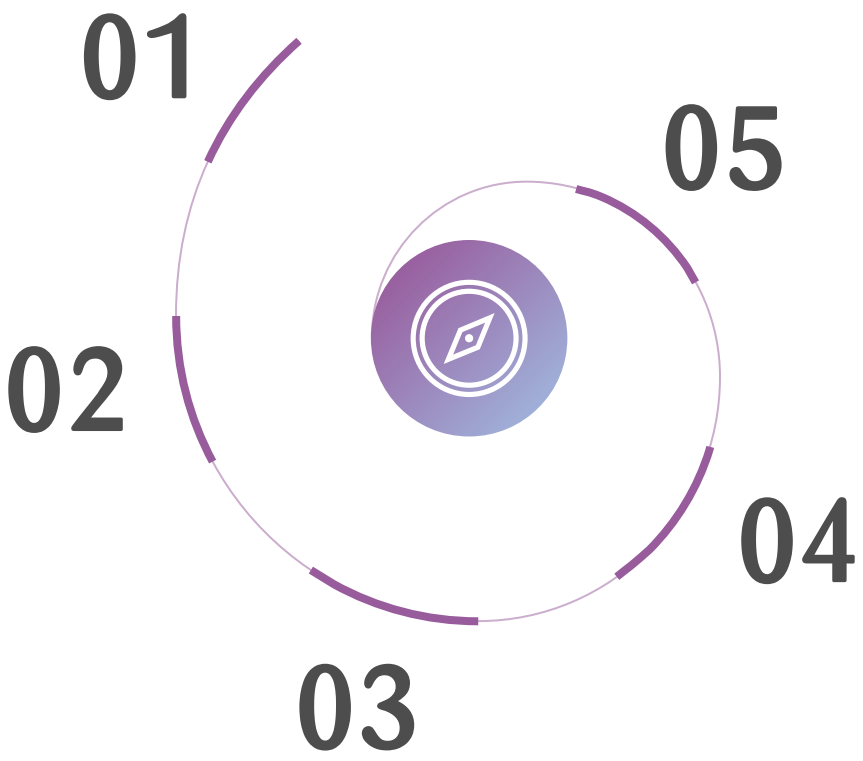
Understand HTTP request/ response exchange and headers vs body.

Understand the flow

Once you selected the flow – learn the HTTP request/response sequence diagram of the flow.

Understand the role of Actors

Know the difference between Client, Resource and Authorization Server.



Handle ID/access token expiry

Check if ID/access token is expired. If yes, get new by using refresh token.

⚠ **Match configuration between actors exactly**

redirect_url: https://myresourceserver.com/resource/
scopes: profile

Common mistakes 2/2

…and how to avoid them

Handle refresh token expiry

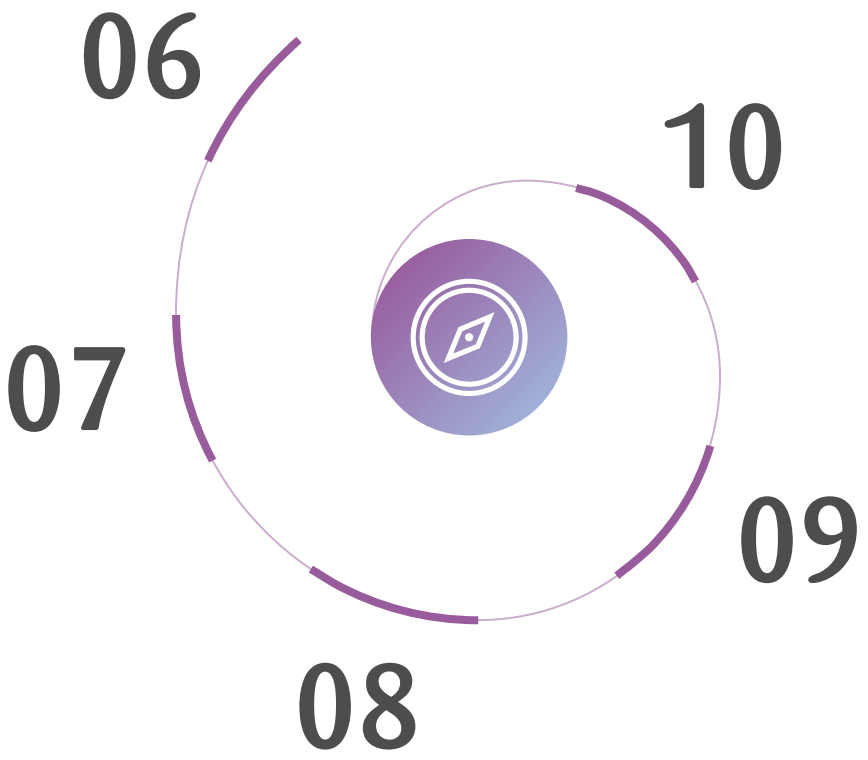
Check if refresh token expired. If yes, re-authenticate user or show “Session expired, click to login”.

Provide logout

In case user doesn’t use their own device to login.

Logging is important!

OIDC is complicated. Logs will save you a lot of time when debugging.



Be aware of intermediate actors

When debugging, be aware of firewalls, proxies etc.

Each environment is different

Dev usually: mock Auth Server, mock components, missing firewall.

What works in Dev might not work in Production!

Use feature toggles on all components when first rolling out OIDC.