

/instituut voor
de Nederlandse
taal/

Keycloak at the INT

A not always straightforward
journey in auth

Koen Mertens

Introduction

- Koen Mertens
- Worked at the INT since 2017
- Started exploring Keycloak end of last year
- Never did Auth before, straight into the deep end

In the past

- Authentication / Identity Management came up from time to time, never as a central focus
- Usually solved per-application in an ad-hoc manner
- Could always fall back on Shibboleth / CLARIN login
- Proper solution becoming more important as we build more software

Current solution



Ad-hoc implementations

Difficulties

- Limiting access to certain users
- User registration
- Roles

- Maintenance
- Setup work again and again
- Bus factor

Requirements



Centralized



Modern standards (OpenID
Connect)



Integrate with CLARIN
federation for existing users

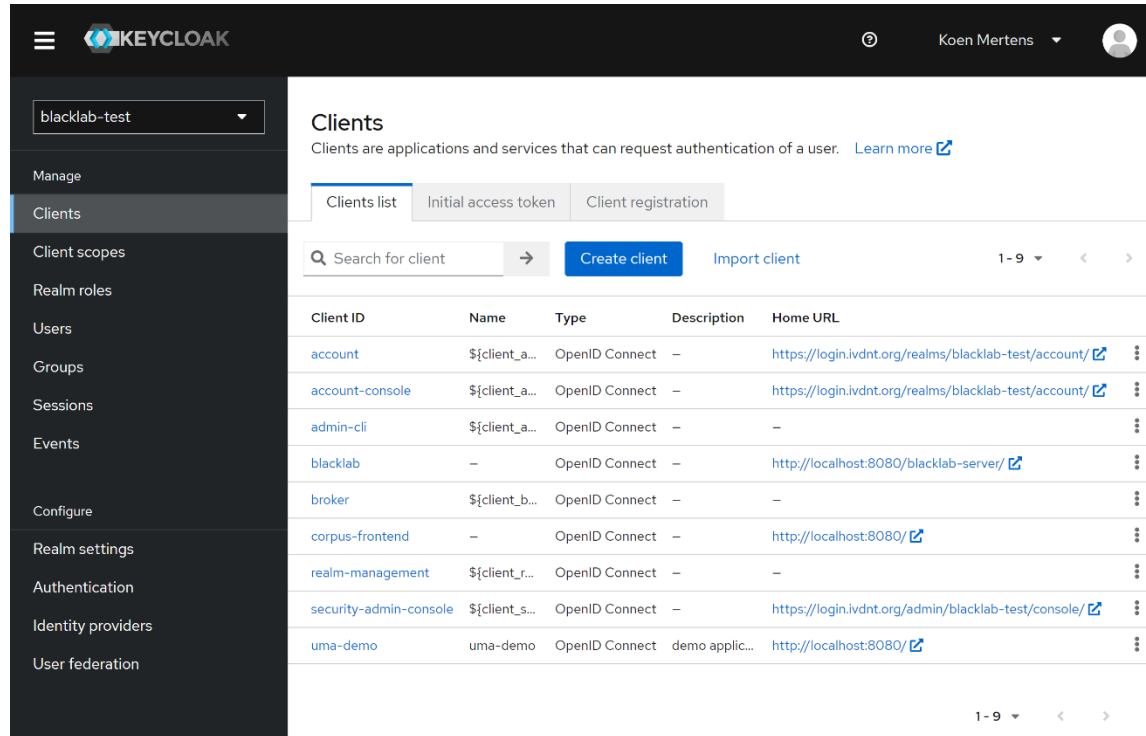


Overview

- Easy to set up (<10 minutes with Docker)
- Frequent releases
- Supports SSO (single-sign-on)
- Supports both OIDC and SAML, good open-source support
- Supports brokering (integrating with social networks, etc)
 - Foreshadowing: Integrating with CLARIN not as straightforward...
- Supports fine grained authorization (user resource sharing)
 - Not as straightforward (again)
- Supports custom themes and extensions

Initial impressions

Nice admin panel!



The screenshot shows the Keycloak admin interface. The top navigation bar includes the Keycloak logo, a user profile for 'Koen Mertens', and a search icon. The left sidebar contains a menu with options like 'Manage', 'Clients', 'Client scopes', 'Realm roles', 'Users', 'Groups', 'Sessions', 'Events', 'Configure', 'Realm settings', 'Authentication', 'Identity providers', and 'User federation'. The main content area is titled 'Clients' and includes a description: 'Clients are applications and services that can request authentication of a user. [Learn more](#)'. Below this, there are tabs for 'Clients list', 'Initial access token', and 'Client registration'. A search bar and buttons for 'Create client' and 'Import client' are visible. A table lists several clients with columns for Client ID, Name, Type, Description, and Home URL.

| Client ID | Name | Type | Description | Home URL |
|------------------------|----------------|----------------|----------------|---|
| account | \${client_a... | OpenID Connect | - | https://login.ivdnt.org/realms/blacklab-test/account/ |
| account-console | \${client_a... | OpenID Connect | - | https://login.ivdnt.org/realms/blacklab-test/account/ |
| admin-cli | \${client_a... | OpenID Connect | - | - |
| blacklab | - | OpenID Connect | - | http://localhost:8080/blacklab-server/ |
| broker | \${client_b... | OpenID Connect | - | - |
| corpus-frontend | - | OpenID Connect | - | http://localhost:8080/ |
| realm-management | \${client_r... | OpenID Connect | - | - |
| security-admin-console | \${client_s... | OpenID Connect | - | https://login.ivdnt.org/admin/blacklab-test/console/ |
| uma-demo | uma-demo | OpenID Connect | demo applic... | http://localhost:8080/ |

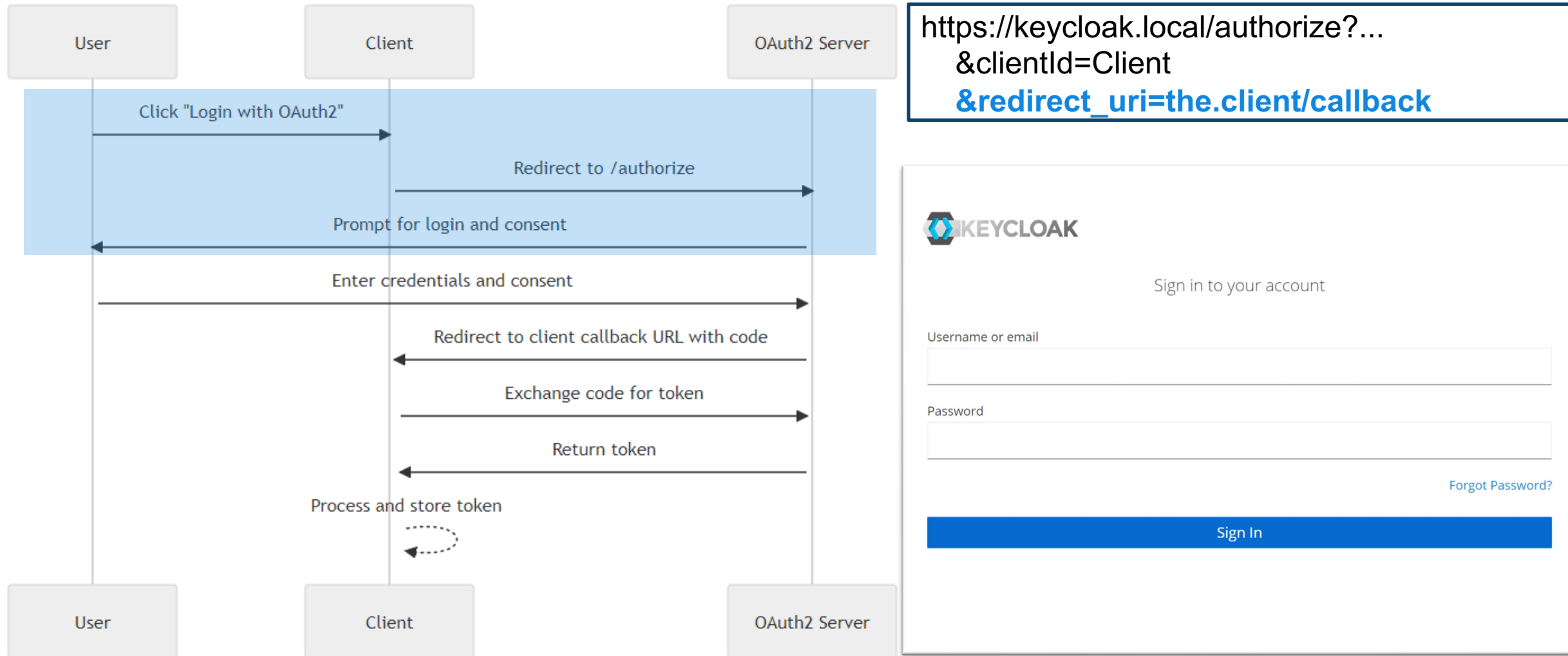
Terminology?

- Realms
- SAML
- OIDC
- Client
- Resource Server
- Identity Provider
- Relying Party
- Scope
- Role
- Group
- Attribute
- Mapper
- Permission
- Service Account
- Providers
- UMA
- Oh my!

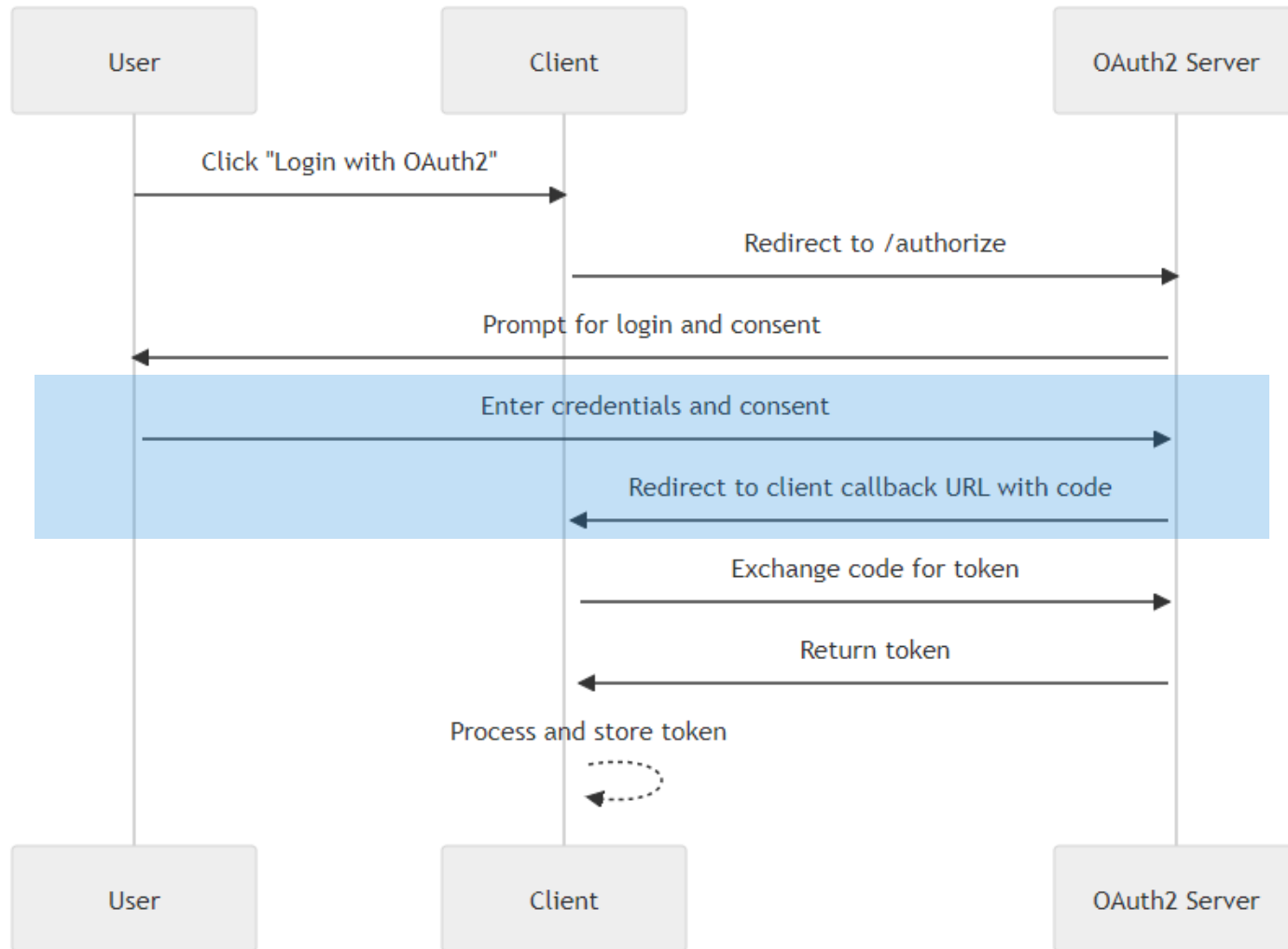
Quick OIDC primer - tokens

- We only care about Tokens
 - Identity Tokens: contain information about the user. Username, email, roles, etc.
 - Don't send this over the web!
 - Access Tokens: for sending to API's, related applications. Can be exchanged with the Identity Provider (Keycloak) to retrieve associated user info and permissions.
- User logs in directly with Identity Provider (Keycloak)
- After login, IDP sends Tokens to applications, representing the User
- Other flows also possible, for example for devices lacking web browser or keyboard.

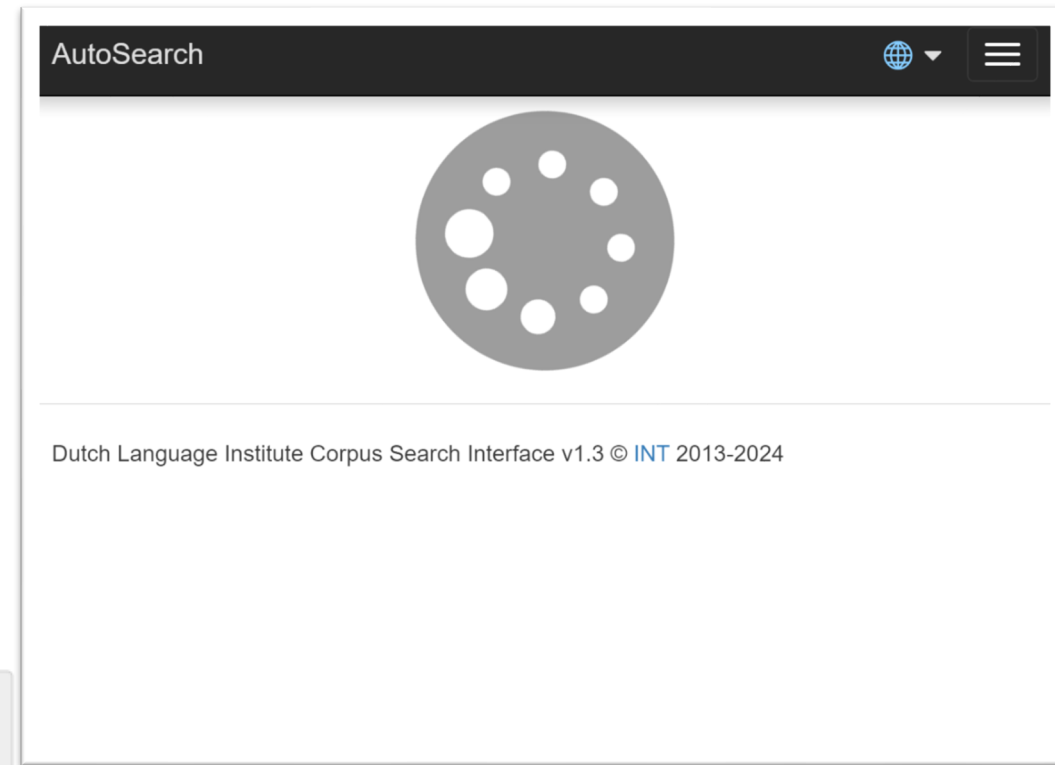
Quick OIDC primer – standard flow



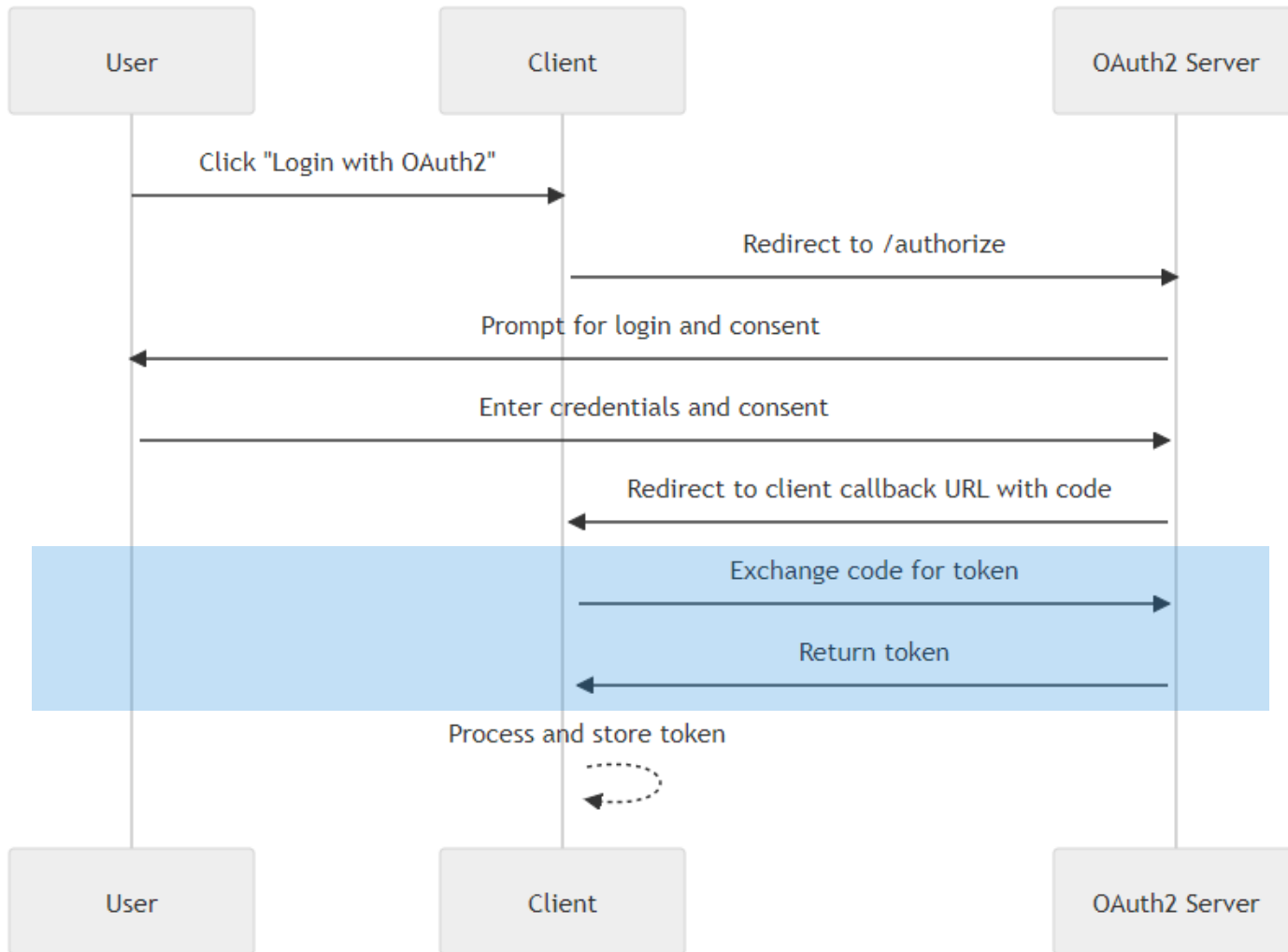
Quick OIDC primer – standard flow



[https://client.local/callback&...
?state=...
?token=...](https://client.local/callback&...?state=...?token=...)



Quick OIDC primer – standard flow

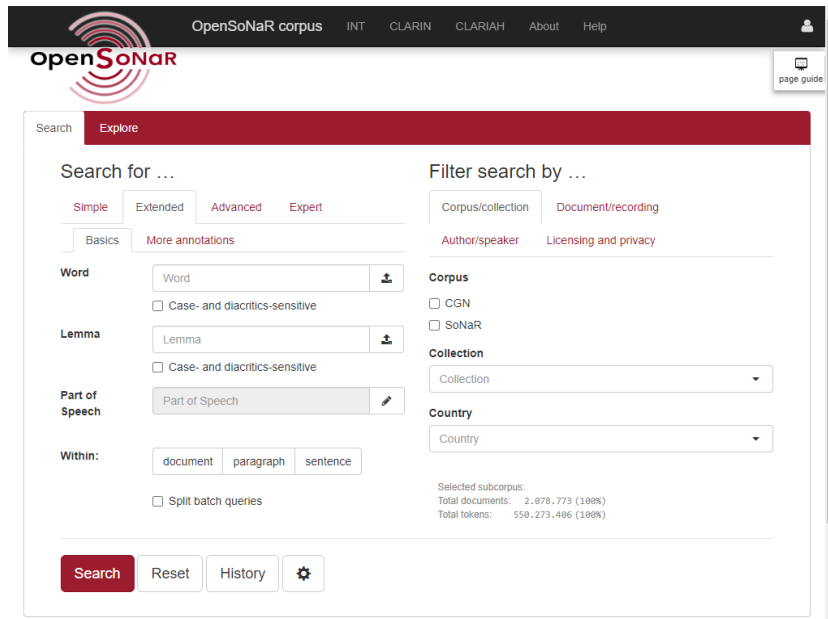


POST <https://keycloak.local/token>

```
code: 205f8878-f509-4a45-83c6-  
c38ac98d6ded.3e66bc99-77d1-4f7e-8183-  
5fd922c6734a.763e740a-25a0-4ec0-800d-  
0b6d0d3dd661  
grant_type: authorization_code
```

```
access_token: ...  
expires_in: 60  
id_token: ...  
not-before-policy: 0  
refresh_expires_in: 3592  
refresh_token: ...  
scope: "openid email profile"  
session_state: "3e66bc99-77d1-4f7e-8183-  
5fd922c6734a"  
token_type: "Bearer"
```

Story: existing applications



- BlackLab + Corpus Frontend, used for publishing our corpora
 - Java servlet creates page scaffold, Vue Javascript frontend
 - Examples: OpenSonar, historical corpora, AutoSearch

- Problem: Keycloak client libraries (adapters) deprecated
 - Need to use opensource alternatives
 - OIDC-client-ts for Javascript frontend
 - Pac4j/Nimbus Jose for Java backend
 - Upside: more generic than just Keycloak

Create a Client

Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

- 1 General Settings
- 2 Capability config
- 3 Login settings

Client type [?] OpenID Connect

Client ID * [?] example

Name [?] example

Description [?] example

Always display in UI [?] Off

[Next](#) [Back](#) [Cancel](#)

- Client here refers to an Application, not an End User

Create a Client

Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

- 1 General Settings
- 2 **Capability config**
- 3 Login settings

Client authentication Off

Authorization Off

Authentication flow

- Standard flow Direct access grants
- Implicit flow Service accounts roles
- OAuth 2.0 Device Authorization Grant
- OIDC CIBA Grant

Next Back Cancel

- Client Authentication will require our application to send a secret when retrieving ID and Access Tokens.
- Only makes sense for server-side applications, as the browser cannot keep a secret.

Create a Client

Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

- 1 General Settings
- 2 Capability config
- 3 Login settings

Root URL ⓘ

Home URL ⓘ

Valid redirect URIs ⓘ -
[+ Add valid redirect URIs](#)

Valid post logout redirect URIs ⓘ -
[+ Add valid post logout redirect URIs](#)

Web origins ⓘ -
[+ Add web origins](#)

- Whitelist the redirect URLs to our application, so the redirect containing the Tokens cannot point outside our application.

Frontend

- Example OIDC-client-ts code
- Doesn't need to do much.
 - Redirect to Keycloak on login
 - When returning, read code, exchange for Access Token, ID Token.
- Simply add Access Token to Authorization header of requests to your API.
- Important enforcement logic lives in the backends.

```
// Common code
const manager = new UserManager({
  metadataUrl: 'https://login.ivdnt.org/realms/example/.well-known/openid-configuration',
  redirect_uri: window.location.origin + CONTEXT_URL + '/callback',
  client_id: 'example',
  authority: 'https://login.ivdnt.org/',
});

// To log in:
// This will redirect us to the Keycloak login,
// After logging in we will be redirected to the redirect_uri set above
manager.signinRedirect({state: window.location.href});

// After logging in:
const url = new URL(window.location.href);

// Error and Code are parameters set by Keycloak
if (url.searchParams.has('error')) {
  alert('Error logging in: ' + url.searchParams.get('error'));
}
if (url.searchParams.has('code')) {
  manager.signinCallback().then(user => {
    if (user) alert(`Logged in as: ${user.profile.name}`);
  })
}
```

Backend first attempt: Pac4J

- BlackLab uses raw Servlet, no framework
- Pac4J is more suited to frameworks, Quarkus, Spring, etc.
- Default setup doesn't work for Access Tokens. Wants to retrieve the Tokens in the backend
 - Safer, but more work. You now need to extend your API just to show the username on the Frontend.
- Pac4J needs lots of massaging to work with Access Tokens
 - Override built-in Validator with no-op implementation (default expects ID tokens)
 - Write custom Authenticator (convert token to User object)
 - Change incompatible defaults
 - Documentation not great on why and how to do these steps
 - Misleading errors in some cases when you make a mistake – compiles doesn't mean it works.
 - Don't get to use many of the features to decorate API functions etc. because there are none.
- Probably don't bother unless using a framework like Spring

Backend second attempt: Nimbus JWT

- Light on code
- Need to manually contact Keycloak for some initial metadata
- Downside: need to manually check usernames and roles in API code.

```
public JWTClaimsSet example(String clientId, String wellKnownURI, HttpServletRequest request)
    throws MalformedURLException, ParseException, BadJOSEException, JOSEException {
    // Fetch the IDP metadata
    OIDCProviderMetadata metadata = getProviderMetadata(wellKnownURI);
    // Fetch public keys for the IDP
    RemoteJWKSet<SecurityContext> keySource = new RemoteJWKSet<>(
        metadata.getJWKSetURI().toURL(),
        new DefaultResourceRetriever(5000, 5000, 0));

    // Create the token parser and validator
    DefaultJWTProcessor<SecurityContext> jwtProcessor = new DefaultJWTProcessor<>();
    jwtProcessor.setJWSVerifier(new DefaultJOSEObjectTypeVerifier<>(JOSEObjectType.JWT));
    jwtProcessor.setJWSKeySelector(new JWSSAlgorithmFamilyJWSKeySelector<>(JWSAlgorithm.Family.SIGNATURE, keySource));
    jwtProcessor.setJWTClaimsSetVerifier(new DefaultJWTClaimsVerifier<>(
        // Audience parameter - check our application is the intended receiver of this token.
        clientId,
        // Params in the token that must match EXACTLY
        new JWTClaimsSet.Builder()
            .issuer(providerMetadata.getIssuer().toString())
            .claim("email_verified", Boolean.TRUE)
            .build(),
        // Params in the token that must exist - but may be anything
        // NOTE: expiration time is checked automatically
        new HashSet<>(Arrays.asList(
            "email",
            JWTClaimNames.ISSUER, // IDP
            JWTClaimNames.SUBJECT, // user id
            JWTClaimNames.ISSUED_AT,
            JWTClaimNames.EXPIRATION_TIME,
            JWTClaimNames.AUDIENCE,
            JWTClaimNames.JWT_ID))
    ));
    // Read the Access Token, decode it, and
    String accessToken = request.getHeader("Authorization").replace("Bearer ", "");
    // Parse and validate the token
    return jwtProcessor.process(JWTParser.parse(accessToken), null);
}
```

Summary for existing applications

- Easy enough for a backend-only application
- Easy enough for a frontend that gets everything from an API
- Extra effort for hybrid applications, where backend serves initial content, but frontend performs additional AJAX requests
 - User info typically only exists on Frontend or Backend, but not both
 - Need to know who is logged in on both sides however
- Some considerations for non-standard setups
 - One frontend, multiple (secured) back-ends (microservices)
 - Backend-to-backend communication

Coupling with CLARIN federation

- We initially expected this to be simple.
- But: Keycloak's SAML broker does not support SAML Discovery Protocol.
 - There is a fork of Keycloak that supports SAML Discovery, but no official support
Seems unwise to depend on for longterm production use.
- Can add every federation one by one with the admin API, but results in 1500+ brokers!
 - The French CLARIN Centre Ortolang does this, we could use their code
 - Doesn't feel quite right
 - Needs a custom Keycloak theme
 - User experience isn't as good as Shibboleth.
- Need something better...

Log in to **ortolang**

Username or email

Password

Remember me [Forgot Password?](#)

[Sign In](#)

New user? [Register](#)

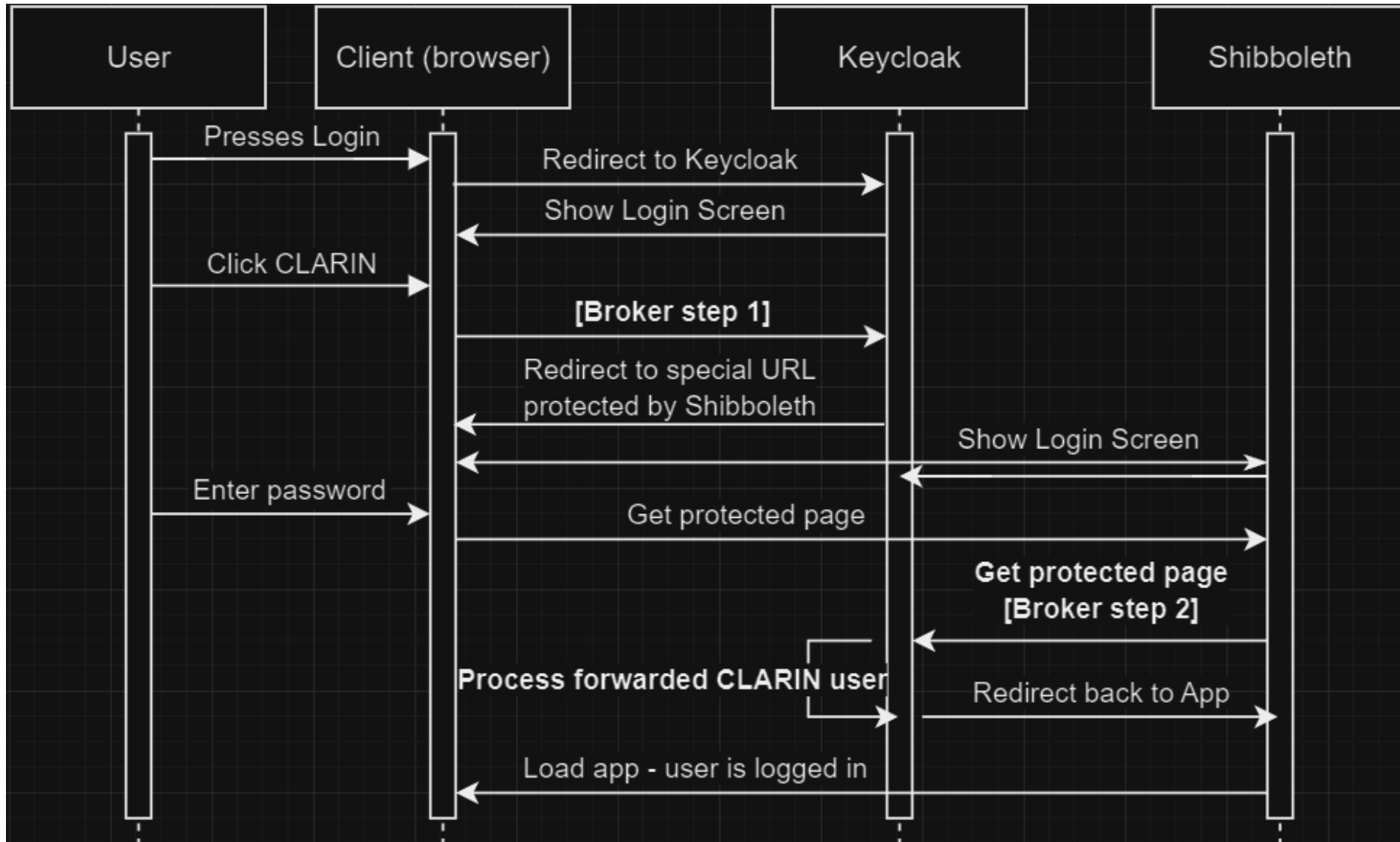
29 Mayıs University

29 Mayıs University
A. T. Still University
A*STAR - Agency for Science, Technology and Research
AAF Virtual Home
aai.lab.maeen.sa
AAI@EduHr Single Sign-On Service

Coupling with CLARIN federation

- We can write a custom Broker for Keycloak
- Same mechanism as Google, Facebook, Github etc. Account linking.
 - Normally a Broker acts like a Client, receives Tokens (from Google, etc.)
 - Reads the username from the Token and logs in the correct Keycloak account.
- We can make our existing CLARIN login portal (Shibboleth) forward requests to our custom Broker
 - Shibboleth will intercept the request, make the user log in with CLARIN, then pass the CLARIN username to our Broker
- Our broker can then read this info and create/login the associated Keycloak account!

Shibboleth/Keycloak flow



Groups, Roles, Scopes, oh my!

- 15 minutes too short to go in depth
- For someone new to Auth, a frustrating and at times confusing topic
- Lots of terminology, very freeform, rules on proper usage seem to be nebulous
 - Keycloak doesn't help by adding extra abstraction layers
 - Groups are an abstraction to apply Roles to users automatically
 - Roles are an abstraction to add attributes to the User's Tokens
 - Scopes are an abstraction to add attributes to the User's Tokens
 - Eventually everything boils down to extra attributes in Tokens
 - Great for re-use, but makes for a steeper learning curve, and overkill for our modest purposes
- Careful suggestion: keep it simple.
 - Ignore Groups and Scopes initially
 - Use Roles only if there is need to differentiate Users by something besides username.
- Important: roles/scopes not suitable for individual permissions per Resource (e.g., uploaded corpora)
 - You can only be an Admin for the entire application, not for only your own data
 - Keycloak has a separate system (UMA – User Managed Access) for this

UMA: User Managed Access

- Can register users' Resources and Permissions on those Resources in Keycloak
 - Example: Alice allows Bob to **read**, but not **edit** a Dictionary she uploaded
 - Bob can lodge a request for permissions with Alice
 - Keycloak won't send Alice an email... Limits real-world usability
- Java library (keycloak-authz-client)
 - Bare-bones, basically a set of classes to interface with Keycloak's API
- Need to build your own UI
 - Keycloak has a bare-bones panel, but it only shows Resources you own
- Important: UMA Resources and Permissions are tied to a single Client
 - A Permission is a combination of: The Client, The User, The Resource, The "verbs" (can be anything)
 - Example, the Dictionary Editor from above can only query Dictionaries, not Corpora,
 - Other Clients cannot see Dictionaries, and cannot query Alice's rights
 - Microservices will need to share the same Client ID
 - Seems more work than using your own database

Questions?