Using OAuth2 to Integrate RESTlike Web Services into a SAML-based Federation

– Implementation of a Policy Decision Point –

Version: 1.0 – August 27, 2019

DARIAH-DE III:

Digitale Forschungsinfrastruktur für die Geistes- und Kulturwissenschaften

Dieses Forschungs- und Entwicklungsprojekt wird / wurde mit Mitteln des Bundesministeriums für Bildung und Forschung (BMBF), Förderkennzeichen 01UG1110A bis M, gefördert und vom Projektträger im Deutschen Zentrum für Luft- und Raumfahrt (PT-DLR) betreut.

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

Projekt: DARIAH-DE III: Digitale Forschungsinfrastruktur für die Geistes- und Kulturwissenschaften

BMBF Förderkennzeichen: 01UG1110A bis M

Laufzeit: März 2016 bis Februar 2019

Dokumentstatus: final

Verfügbarkeit: öffentlich

Autoren:

Martin Haase, DAASI International GmbH

Peter Gietz, DAASI International GmbH

Markus Widmer, DAASI International GmbH

David Hübner, DAASI International GmbH

Stefan E. Funk, SUB Göttingen

Ubbo Veentjer, SUB Göttingen

Marcel Hellkamp, GWDG

Progression of Revisions

Date	**Version**	**Author**	**Comments**
Jun 19, 2014		Martin Haase	Grundsätzliches Modell
Jul 24, 2014		Martin Haase, Peter Gietz	Erstellung eines DARIAH internen Papiers
Jul 7, 2015		Martin Haase	Spezifikation der StorageAPI
Aug 7, 2015		Martin Haase, Peter Gietz, Markus Widmer	Detaillierte Angaben zum RBAC-AS und LDAP-Struktur
Nov 13, 2015		Stefan E. Funk	Use-Cases und benötigte Methoden hinzugefügt
Dec 1, 2015		Martin Haase	Kommentare zu den Use Cases und zu Sonstiges
Dec 15, 2015		Stefan E. Funk	Kapitel 4.4 näher spezifiziert, Übersichtsgrafik des DARIAH-DE Repositorys hinzugefügt
Dec 18, 2015		Martin Haase	Reduzierung der Token-Anzahl bei publish (3.1.2), Änderungen akzeptieren
Dec 21, 2015		Stefan E. Funk	Spezifikation der einzelnen REST-Methoden des PDP
Dec 30, 2015		Martin Haase, Markus Widmer	Anpassungen im Zuge der Implementierung: LDAP-Struktur (3.3), Pfad der PDP-Methoden (4.2)
Jan 13, 2016		Martin Haase	Weitere Kommentare zum RBAC als Vorbereitung auf die heutige (v0.10) und morgige (v0.11) Telko
Jan 14, 2016		Martin Haase	Finalisierung nach heutiger Telko (v0.12), für GWDG
Jan 27, 2016		Peter Gietz	Finalisierung nach heutiger Telko (v0.13): Kleinere Änderungen an der REST API in Kap. 4.
Feb 11, 2016		Martin Haase	Token-Bezug (Kap. 4.1)
Jul 13, 2016		Stefan E. Funk	Allgemeine Korrekturen, Zugriff auf CDStar-Instanz ergänzt
Jul 29, 2016		Martin Haase	Kommentare
Sep 21, 2016		Stefan E. Funk	Allgemeine Überarbeitung

Jan 17, 2017		Stefan E. Funk	Weitere Kommentare eingearbeitet
Mar 7, 2017		Stefan E. Funk	Abbildung 3 finalisiert: Flow eines DARIAH-DE Storage-Token
Mar 21, 2017		Martin Haase	Update der PDP-Grafik und Beschreibung
May 12, 2017		Stefan E. Funk	Zusammenführung der letzten Versionen, Kapitel zu dynamischen Gruppen hinzugefügt
Jul 10, 2017		David Hübner	Translation, various OAuth2 clarifications
Sep 29, 2017		Stefan E. Funk	Added DiscussData and Annotator Use-Cases
Oct 13, 2017	0.25	Martin Haase	Feedback on v2.4 and from several tickets
Feb 21, 2018	0.26	Martin Haase	Described functionality of access token management in the SelfService LUI (3.3.4 , 3.5).
Aug 23, 2018	0.28	Stefan E. Funk	Edited to latest Datasheet Editor, Geo-Browser, and DARIAH-DE Repository Issues. We do need them for: (a) isPublic flag and API for setting OwnStorage resources public, and (b) Storage Token generation in DARIAH-DE SelfService to use with DARIAH-DE Repository API.
Aug 23, 2018	0.29	Stefan E. Funk	Removed Annotator and DiscussData use cases, we only need basic DH-rep, PDP, or Shibboleth functionality here.
Dez 11, 2018	0.3	Stefan E. Funk	Storage Token can be revealed in the Publikator, no need to implement this in the SelfService, #setPublic issues commented, ordering of future tasks and TODOs.
Jan 02, 2019	0.31	Martin Haase	#listResources; many comments resolved and created; spelled out TODOs (and sorted tickets)
Jan 07, 2019	0.32	Stefan E. Funk	Resolved PDP/AS/Storage issues: We indeed must query the storage resource for #setPublic and related things, not PDP, storage must implement these for OwnStorage…
Feb 01, 2019	0.33	Stefan E. Funk	Added the ownStorage flag, adapted methods and documentation
Feb 01, 2019	0.34	Stefan E. Funk	Adapted to new Storage API paper v2.0, added Abbildungsverzeichnis
Feb 12, 2019	0.35	Martin Haase	Some clarifications wrt. the API

Feb 25, 2019	0.50	Stefan E. Funk	Adding DARIAH-DE Own- and PublicStorage API spec and their usage of PDP methods.
March 1, 2019	0.51	Stefan E. Funk	Completed OwnStorage and PublicStorage API documentation.
March 6, 2019	0.60	Stefan E. Funk	Removed some refs to storage API paper version 1.0.
March 11, 2019	0.70	Stefan E. Funk	Adopted #checkAccess on Own and PublicStorage.
April 9, 2019	0.82	Stefan E. Funk	Added PDP.php PDP path.
April 11, 2019	0.84	Stefan E. Funk	Added NOT FOUND responses for unknown PDP methods,
May 15, 2019	0.88	Stefan E. Funk	Removed the PDP#checkAccess calls from Own and PublicStorage#create.
August 26, 2019	1.0	Stefan E. Funk	Finalized document.

Content

1 Introduction

The Authentication- and Authorization-Infrastructure (AAI) of DARIAH is based on the SAML protocol and implemented with open-source-solutions, such as OpenLDAP, didmosLUI, Shibboleth Identity Provider and Shibboleth Service Provider and a DARIAH-developed Java Service Provider.

The SAML Web Single Sign-On profile covers most of the requirements of today's web applications. However, a common use-case is a web application (front-end) accessing a web service (back-end) securely in the name of the user, that is logged in at the front-end application. Handing over the SAML assertion securely is rather difficult to realize[1]. Other possibilities include Kerberos Constrained Delegation, which requires a very specific infrastructure or dedicated service-accounts, that are easily compromised and do not scale well.

If the web service is based on the SOAP protocol, authentication usually is done via the SOAP header. An example for this is the SAML Security Token profile[2]. Recent development indicates a shift from SOAP-based to REST-based solutions for back-end web services. The IETF standard OAuth2[3] can be used to cover this use-case.

This document briefly describes a solution, where the "Authorization Code Flow" of OAuth2 is combined with SAML-based authentication to provide secure access to a REST-based back-end. This solution has been deployed successfully by DAASI International in the past and could be re-used for DARIAH. Furthermore this document details how such a flow could be integrated into an external Policy Decision Point (PDP) in order to allow it to be used for the DARIAH Storage API.

1.1 Bibliography

[DARIAH-Storage-APIv1.0]	Stefan E. Funk, Peter Gietz, Martin Haase, Patrick Harms, Andreas Aschenbrenner, Danah Tonne, Jedrzej Rybicki: DARIAH Storage API – A Basic Storage Service API on Bit Preservation Level, Version: 1.0
[DARIAH-Storage-APIv2.0]	Stefan E. Funk, Peter Gietz, Martin Haase, Patrick Harms, Andreas Aschenbrenner, Danah Tonne, Jedrzej Rybicki: DARIAH Storage API – A Basic Storage Service API on Bit Preservation Level, Version: 2.0

1 cf. https://spaces.internet2.edu/display/ShibuPortal/Home
2 http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SAMLTokenProfile.pdf
3 http://tools.ietf.org/html/rfc6749

1.2 Abbreviations

AAI	Authentication and Authorization Infrastructure
API	Application Program Interface
AS	(OAuth2) Authorization Server
ePPN	eduPersonPrincipalName
IdP	Identity Provider
LDAP	Lightweight Directory Access Protocol
PAOS	SOAP, backwards
PDP	Policy Decision Point
RBAC	Role Based Access Control
RS	Resource Server
SAML	Security Assertion Markup Language
SID	Session ID
SOAP	Simple Object Access Protocol
SP	Service Provider
SSO	Single Sign On
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

2 General OAuth2 Authorization Code Flow with SAML

In Figure 1, the "Web Service Client or Portal" is the front-end application mentioned in the introduction, while the "OAuth2 Resource Server (RS)" is the back-end web-service. The "Web Service Client or Portal " will be referred to as external SP. Please note, that this component needs to act as both, a SAML SP towards the SAML IdP, that is used for authentication and as OAuth2 client towards the AS and the RS.

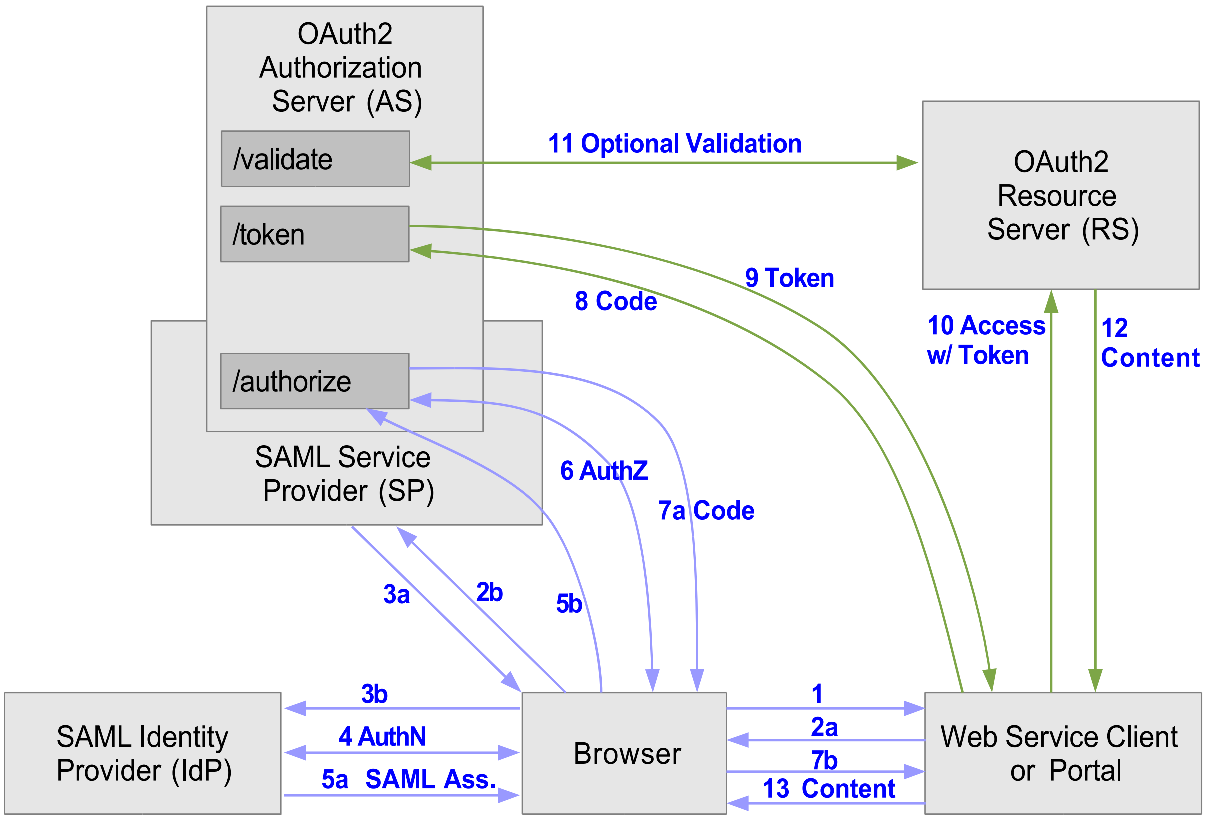

Figure 1: OAuth2 Authorization Code Flow with SAML

Figure 1 shows the access of an external SP (a Web Service Client or Portal) to a REST-based web-service (Resource Server, RS). Most importantly, the external SP eventually gets granted access to information stored at the Resource Server in the name of the user, that is logged in at the external SP. For this, an OAuth2 Access Token is used. This token is only valid for this user and a limited time. Optionally, an additional Refresh Token might be issued, that can be used by the external SP to request a new Access Token after the previous one expired without the need for additional user interaction. Tokens can be invalidated by both the AS and the external SP at any time. The detailed flow is as follows:

1. The user accesses the external SP. Authentication can be with the IdP or by some other means. If it is with an IdP, then the arrows from the diagram denote:
    - 2a: External SP issued Authentication Request and

- 3b: redirects the user to the external IdP.
- 4: Authentication at the external IdP
- 5a: External IdP issues assertion and
- 7b: redirects user back to the external SP.

2. As soon as the external SP wants to request data from the REST-based web-service, it redirects the user to the Authorization endpoint (/authorize) of the OAuth2 Authorization Server (AS) to eventually get an Access Token (2a / 2b). This endpoint is protected by a SAML SP, which will be referred to as AS-SP.. The external SP transmits the following information as part of the HTTP GET parameters:
   1. Scope of the intended operation (read, write, …)
   2. Its client ID
   3. A redirect URI
   4. An identifier for the status

   During step 1 (above) the user already authenticated at the external IdP. This session can be reused by the AS-SP (Single Sign-On). If, for whatever reason, no session can be found, the user has to authenticate at the external IdP again. At the very least, the AS-SP gets an user identifier from the external IdP, that can be used to identify the user at the AS.
3. Authentication Request from the AS-SP to the external IdP (cf. Step 2)
4. The user authenticates towards the IdP. Note that this step will be invisible if the user had authenticated in Step 1 already, due to Single Sign-On.
5. SAML assertion is being transmitted to the AS-SP
6. The AS asks the user to *authorize* the request, showing the user the scope and name of the client of this request. Allowing the user to explicitly allow or deny access to her/his data is one of the main characteristics of the OAuth2 protocol.
7. Assuming the user authorized the request, the AS redirects the user back to the redirect URI (cf. Step 2) at the external SP. Part of the HTTP GET parameters is the so called Authorization Code, which is a short-lived, one-time-use code, that can be used by the external SP to request the Access Token at the AS.
8. Using the Authorization Code, the external SP requests the Access Token directly at the AS via a back-channel request (usually REST-based). The /token-endpoint used for this request is *not* protected by the AS-SP and must not be protected by a firewall that would prevent access.
9. After validating the Authorization Code and request, the AS sends the Access Token (and optionally a Refresh Token) to the external SP.
10. Using the Access Token, the external SP can now send a request to the Resource Server. The Access Token is transferred in the HTTP header. Access to the RS must be allowed for the external SP (e.g. Firewall).
11. The RS should validate the Access Token. This step usually requires interaction

with the AS. The Access Token allows identification of the user at the AS. As an alternative, there are so-called Self-contained tokens. These signed tokens allow the RS to directly validate an Access Token, which prevents latency-problems and processing in the AS. Self-contained tokens are usually short-lived and therefore must be renewed, using Refresh Tokens frequently.

12. The RS returns the requested resources.
13. After processing the information, the external SP can provide adequate feedback to the user.
14. (not shown) After the Access Token expires, the external SP can request a new Access Token at the /token-endpoint of the AS, using the Refresh Token. This request is similar to step 8 and requires no user interaction.
15. The external SP gets a new Access Token (similar to step 9).

In figure 1, the AS, the AS-SP and the RS are individual components. However, these three components might be combined. Some implementations offer any combination of two of these components or all of them. The external IdP might also be part of such a combination.

The following user interaction is required:

- Authentication at the external IdP (e.g. User/Password)
- Step 4 might require authentication of the user at the external IdP, if no SSO-session can be used
- During step 6 the user has to authorize the request. This interaction only happens during the first request

As soon as the external SP has a valid Access Token, steps 5-11 are no longer necessary and can be omitted. Optionally, the AS also issues a Refresh Token for the external SP. In this case, even after the Access Token expires, the external AS can request a new Access Token without any user interaction (steps 14+15).

3 The DARIAH-AAI Flow

While chapter 2 provided an overview of the general OAuth2 Authorization Code Flow, chapter 3 will focus on the integration into the DARIAH infrastructure and the implementation of the centralized Policy Decision Point (PDP), consisting of RBAC and OAuth2 AS, using openRBAC, openContext APIs and the existing DARIAH LDAP server.

3.1 Use Cases in DARIAH-DE

DARIAH-DE provides two storage instances: OwnStorage and PublicStorage.

OwnStorage provides storage for a single user only (#read / #write / #update / #publish / #unpublish) and single resources can be set public by the resources' owner to be available for everyone (#read). This can be reversed by using #unpublish anytime.

PublicStorage provides storage for resources that are publicly available and shall not be deleted anymore. Users can write resources (once) and they are publicly available after writing. No delete or update is provided.

Both storage instances are using the same PDP and AS REST interface, so OwnStorage and PublicStorage resources both are registered in the same PDP.

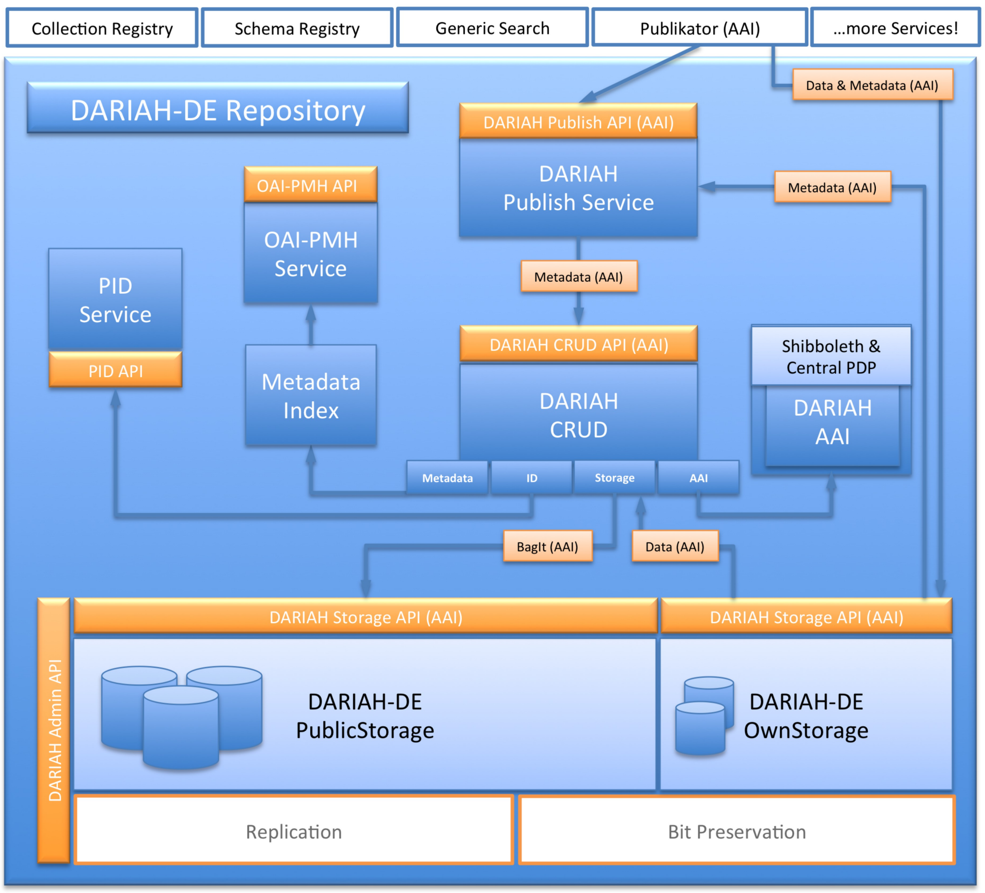

Figure 2: The DARIAH-DE Repository Architecture

3.1.1 Publikator – Browser Application with Java Backend

The DARIAH-DE Publikator[4] provides the workflow to publish research data in the DARIAH-DE repository (cf. DARIAH-DE Repository[5]). Both, the DARIAH-DE OwnStorage and the DARIAH-DE PublicStorage are accessed by different services at different levels of the workflow. All of these calls should be on behalf of and with authorization of the user. There will only be a single Access Token, that will be used to access both, the OwnStorage and the PublicStorage service.

The user's access token can easily be revealed be pressing the "reveal storage token" button, and then be used for storage or higher level service API calls.

The Publikator web application requests that token from the AS-component of the PDP and later hands it over to the Publish-Service. This token will only be used by the individual DARIAH-DE repository-services in a secured environment. Note, that the OAuth2 Access Token is a Bearer Token[6] and therefore can be used to access a resource by anyone who is possession of such a token. It is therefore paramount to make sure, that such a token is only shared with applications, that are privileged to access these resources. It is common to only issue short-lived Access Tokens, in order to contain possible damage. In such a scenario the client, that originally requested such a token, can use a Refresh Token to request new Access Tokens without additional user interaction.

Unlike Access Tokens, these Refresh Tokens do have a specified audience (i.e. the client). For the foreseeable future the DARIAH PDP will likely use long-lived Access Tokens.

4 https://trep.de.dariah.eu/publikator

5 https://repository.de.dariah.eu/doc/services/submodules/publikator/docs/index.html

6 https://tools.ietf.org/html/rfc6750

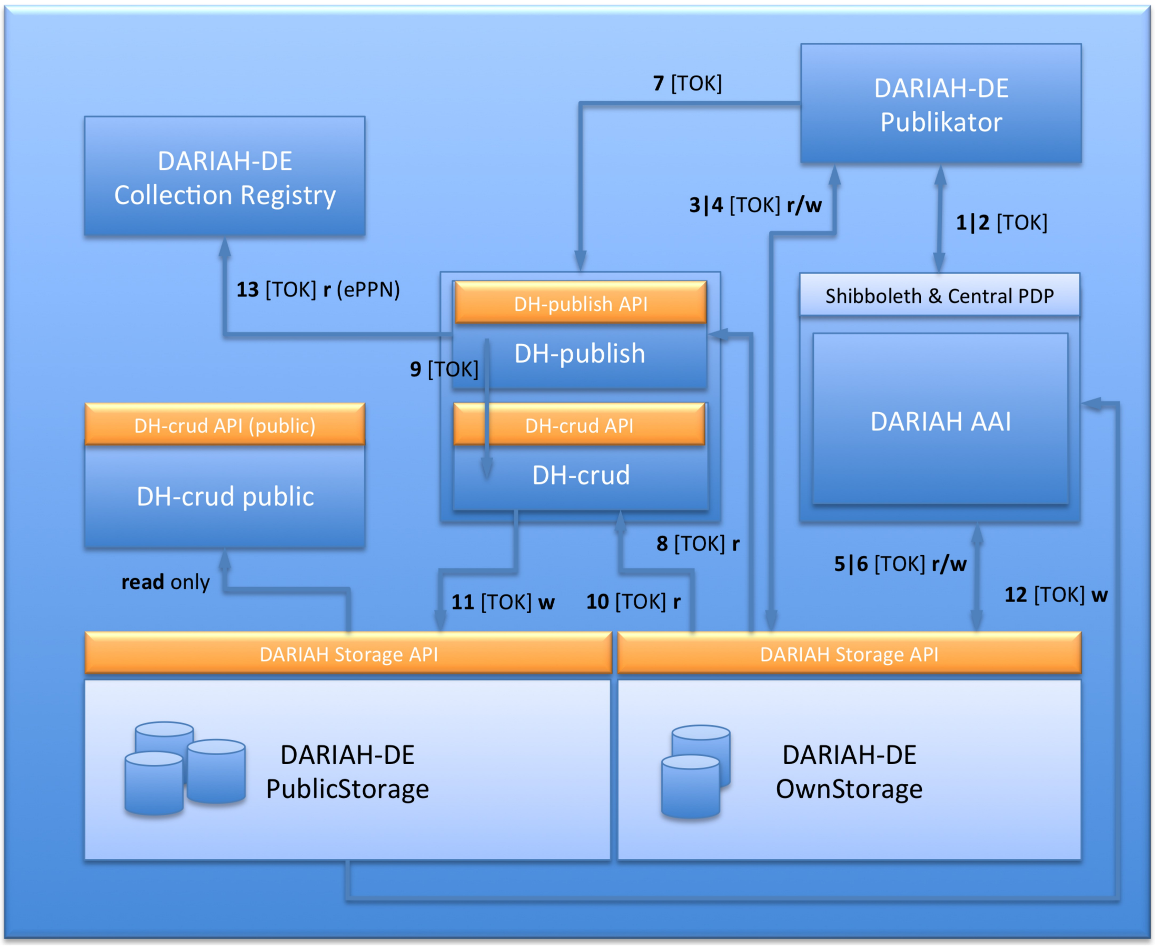

Figure 3: Flow of the DARIAH-DE Storage Token

Figure 3 shows the workflow of the Publikator use-case:

- 1 + 2: The Publikator application requests an Access Token from the PDP:
  - Token T1 for the DARIAH-DE storage (OwnStorage/PublicStorage), client: Publikator, Scope: read/write
- 3 + 4: Publikator writes/updates/deletes the user's data in the OwnStorage, using Token T1 (public=false).
- 5 + 6: The storage-implementation (OwnStorage) requests a policy decision from the PDP, checking if the user's privileges are sufficient for the requested operation. If yes, the write operation is performed.
- 7: Publikator calls DH-publish and hands over Token T1. All clients (Publikator, DH-publish, DH-crud, Collection Registry) are trustworthy and in the same security domain.
- 8: DH-publish reads from OwnStorage, using Token T1 (public=false)
- 9: DH-publish hands over Token T1 to DH-crud
- 10: DH-crud reads from OwnStorage, using Token T1 (public=false)
- 11: DH-crud writes to PublicStorage, using Token T1 (public=true)

- 12: The storage-implementation (PublicStorage) requests a policy decision from the PDP, checking if the user's privileges are sufficient for the requested operation. If yes, the write operation is performed.
- 13: DH-publish creates collection-record for the user in the Collection Registry (CR)
  - CR does not use any tokens, but gets the user's ePPN directly from DH-publish and uses it to connect the record to the user. DH-publish uses an interface with HTTP basic auth to connect to the CR.

3.1.2 Repository API Usage

- Get token T1 from Publikator for storage access (see development service at https://trep.de.dariah.eu/publikator)
- Publish via API using (I) DH-import (koLibRI) or (II) direct DH-publish access as described in the DH-publishAPI documentation (https://repository.de.dariah.eu/doc/services/submodules/kolibri/kolibri-dhpublish-service/docs)
  - DH-publish API takes (i) a storage token and (ii) a storage ID from the OwnStorage, and first checks if the file can be read without error (such as restricted access)
  - If storage access is granted the user is allowed to publish. More testing is then done in DH-publish modules (such as checking mandatory metadata and overall file access).

3.1.3 Geo-Browser and Datasheet Editor – HTML/JavaScript Application

The Geo-Browser web-application (https://geobrowser.de.dariah.eu), including the Datasheet Editor (https://geobrowser.de.dariah.eu/edit), allows to store and access data in the DARIAH-DE storage. Currently everything is being stored in the DARIAH-DE OpenStorage without any access control. The DARIAH-DE OpenStorage used by the Datasheet Editor and the Geo-Browser is provided by the KIT at the moment, and will be shut down end of February 2019.

In order to grant usage to the Datasheet Editor's and Geo-Browser's storage, we do need to change this to (a) store all data in the DARIAH-DE OwnStorage (under the user's name) and (b) allow a user to flag his/her files to be readable by others. This should NOT result in the data being copied to the DARIAH-DE PublicStorage, but rather by setting the PDP's public flag via OwnStorage, indicating that the files should be readable by everyone. All files in the OwnStorage can be edited or deleted by the user and the isPublic-flag can be revoked at any time.

The workflow of the Geo-Browser use-cases follows:

- The Geo-Browser requests an OAuth2 storage token from the AS in order to access the OwnStorage on the user's behalf.
- The Geo-Browser reads/writes/updates files in the OwnStorage, using that token.
- The user is able to set/unset the public flag from within the Geo-Browser, or any

other app that uses the token, querying the storage implementation, OwnStorage in this case (see #publish and #unpublish).

- The user must be able to chose from his files in the OwnStorage to set a certain file's status to public=true and again to public=false. Therefore we do need a list of published and/or unpublished files from the OwnStorage (see #listResources).

Resource-Management:

- In a later version, a GUI to manage all objects of a user at the RBAC is probably needed (including file- and group-permissions, perhaps as part of an extension to the Self-service portal)
- In a later version: lists of objects in the RBAC and an interface between generic search and OwnStorage

3.1.4 Dynamic groups

Dynamic groups should be implemented, that allow users to share (probably only read-access at first) objects in their OwnStorage with other DARIAH users.

One possible use-case is a Geo-Browser user (let's call him Franz) wanting to share one of his files, stored in his OwnStorage, with other DARIAH users:

- Franz logs into the Datasheet Editor via Shibboleth
- Franz creates a new datasheet and uses the Geo-Browser to display it
- Franz creates a new group or chooses an existing group in the LDAP (perhaps via the SelfService)
- Franz adds all DARIAH users to this group, he wants to share his datasheet with. DARIAH users need to explicitly allow other users to find them with such a search (similar to the TextGrid-searchable flag).

3.2 Overview

Figure 4 gives an overview of the proposed architecture. The central PDP-component (consisting of SP, RBAC and OAuth2 AS) is placed in the center.

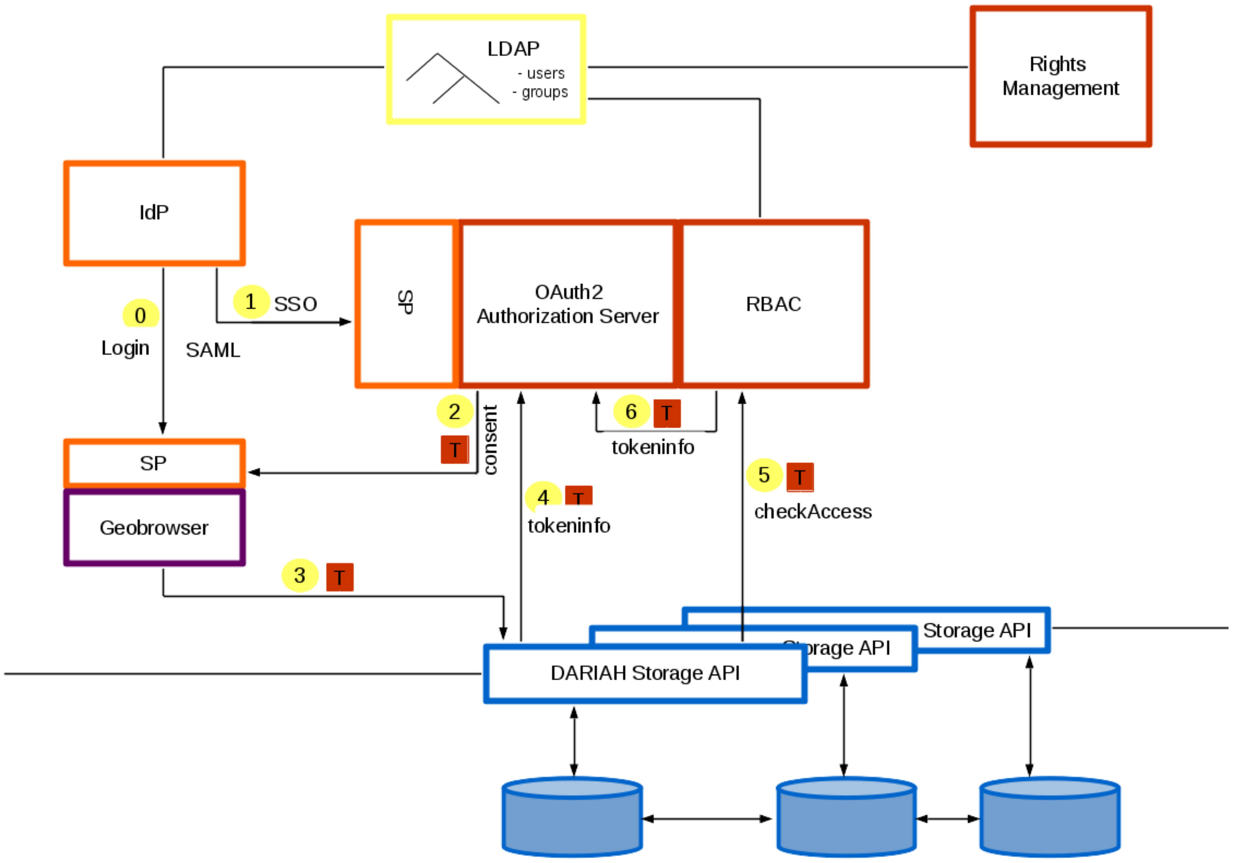

Figure 4: The Policy Decision Point (PDP), consisting of an openRBAC-server and OAuth2 AS, positioned in the DARIAH-AAI

Figure 4 shows the following steps (yellow boxes):

0. The user's browser already established a valid session with an application (here: Geo-Browser)
1. As soon as resources are requested, the user is redirected to the AS-SP, which triggers a SAML Authentication Request at the IdP (as specified in the DARIAH-AAI). Since the user already has a session at the IdP, a SAML-assertion is redirected to the AS-SP (SSO). Now shown: attribute aggregation as specified in the DARIAH-AAI.
2. The AS issues an OAuth2 token T intended to be used by the client (Geo-Browser) to read/write at the StorageAPI and asks the user to authorize. This token is transferred to the application via the OAuth2 Authorization Code Flow (cf. Steps 5-11 in chapter 2).
3. cf. step 12 in chapter 2 (GeoBrowser accesses Resource Server, using T)
4. cf. step 13 in chapter 2. Token validation happens exactly as described in chapter 4.3.
5. Prior to every access to a resource, the StorageAPI calls the checkAccess endpoint at the PDP, using token T.
6. Internally the openRBAC-server validates token T.

7. (not shown) The response from the PDP to the StorageAPI happens as specified in chapter 4.4.

Note, that the access management to share/unshare files, is not shown in Figure 4, because it is independent of the flow shown in Figure 4.

3.3 Usage of the OAuth2 Admin Interface

The Admin interface of the OAuth2 AS component of the PDP is available at https://pdp.de.dariah.eu/oauth2/client/client.html.

3.3.1 Access restrictions

Any DARIAH user can log in, however, only administrative users can see or modify entities created by other users. Adding an administrative user is currently done by adding a user ID / an eduPersonPrincipalName to the configuration file /opt/apis-home/conf/saml.attributes.properties on the PDP server, and restarting the Tomcat servlet container. While this is a working solution, certainly a more dynamic method, e.g. using an LDAP group, would be a desirable future solution.

Non-administrative users would normally not create Resource Servers and Client Applications, but can be expected to have the desire to inspect or delete the tokens that were issued (see 3.5.2).

3.3.2 Resource Servers and Client Applications

The interface has two menu entries for adding OAuth2 Resources (RS) and Clients. There is a 1:N relationship between RS and Clients, i.e. any Client is always attached to exactly one RS.

Adding an RS will define the OAuth2 "scopes" and any descriptive metadata. An RS key and secret will be created by the AS.

For adding a Client, first an RS needs to be selected, and a sub-set of the available RS scopes that this Client would request. You can set here three checkboxes:

- Allow implicit grant: this is for Browser-based/JavaScript applications that omit the background token retrieval step
- Allow client credentials grant: this is for service accounts that need no user permission
- Use refresh tokens: as a security feature, it is recommended to use short-lived access tokens. Using this feature allows the client to receive both an access token and a refresh token. As in the authorization-code/token retrieval step, the client would retrieve another access/refresh token pair after access token expiration, using the refresh token.

The token expiration time is set here as well. A Client key and secret will be created by the AS.

Please refer to the OAuth2 RFC https://tools.ietf.org/html/rfc6749 in order to get the correct definitions of the terms used in this section and in the GUI.

3.3.3 Access Tokens

The "Access Tokens" menu entry is just for browsing the existing tokens; and the user can delete them if needed. Admins can delete other user's tokens (see 3.3.4.2).

The functionality to delete one's own tokens is now available in the DARIAH SelfService (cf. 3.5.2), which makes a priviledged callout (see 3.3.4.1) to the AS. Thus a DARIAH user does not need to handle two interfaces.

3.3.4 REST interface

The Admin Interface is actually a light-weight JavaScript Browser-based client that is secured using OAuth2. All data is being retrieved from the back-end AS using an access token. The capabilities of the REST API the AS offers are not documented but can be deduced from the way the GUI offers its components, and by recording its requests using a Web browser tracing tool, see 3.3.4.2 .

3.3.4.1 Security

Since the administrative functions are supposed to be used interactively, it was decided to secure them by Shibboleth SP in the current deployment. The user would access the Shibboleth-protected /client endpoint first. Having the session established, the JavaScript-based requests to the "/admin" endpoints (see below) will be accessible. The user thus needs

- A session cookie issued by the SP
- An access token created for the administrative client during access of the AS

These functions can be used in an interactive manner. They are coupled with the SP's rather short-lived session duration (which by default times out after 1 hour of web browser inactivity).

For *non-interactive* requests by services such as the didmos LUI SelfService (see 3.5.2), another endpoint "/luiadmin" was created. To access this endpoint, the service needs

- To send its request from a defined IP address configured on the Apache Web Server
- Present Basic Authentication credentials (i.e. username / password) configured on the Apache Web Server
- Present a long-lived administrative access token that has been issued beforehand (e.g. interactively)

For the HTTP Basic Authentication by the Web Server and for the OAuth2 Authentication by the AS, the headers "*Authorization*: Basic ..." and "*Oauthorization*: bearer ..." will be used, respectively.

Both endpoints /admin and /luiadmin are proxied to the AS' Tomcat servlet container using the AJP protocol. Once Basic Authentication has been checked, the Apache Web Server is configured to rename the custom "Oauthorization" request header into the standard "Authorization" request header expected by the AS.

3.3.4.2 Endpoints

The basic endpoints used are:

- `https://pdp(dev).de.dariah.eu/oauth2/admin/` (interactive mode)
- `https://pdp(dev).de.dariah.eu/oauth2/luiadmin/` (service account access)

The following RESTlike operations can be issued. Depending on the HTTP verb, the action is different:

- GET: return a JSON structure containing the data
- DELETE: delete the specified resource(s)
- PUT: create a resource
- POST: modify the specified resource

In order to access concrete resources, the following RESTlike path specifications must be added to the basic endpoints specified above.

- *accessToken* – return/delete all present tokens, including expired ones
- *accesToken/851* – return/delete access token with AS-internal ID "851"
- *accessTokenForOwner/StefanFunk@dariah.eu* – return/delete all tokens of user with resourceOwnerId "StefanFunk@dariah.eu" *(this function constitutes an add-on contributed by DAASI to the AS core code)*
- *resourceServer* – return all OAuth2 resource servers defined in the AS
- *resourceServer/101* – return resource server with internal ID "101"
- *resourceServer/101/client* – return OAuth2 clients connected to this resource server
- *resourceServer/101/client/251* – return client with internal ID "251" connected to this resource server
- *resourceServer/stats* – return statistics of all resource servers

3.4 Structure of the RBAC

The data structure of the LDAP server needs to by slightly modified to store sessions of the openRBAC server. Unlike previous versions of this document, the proposed modification focuses on the highest possible compatibility with the current production-level LDAP server. Significant changes include:

- No additional levels below ou=people. A high number of existing applications operate of this level. Search requests for users should use baseDn: dc=dariah,dc=eu and must use the search filter (&(objectClass=dariahPerson)(!(dariahDeleted=TRUE)).
- No additional levels below ou=groups. Once again, several existing applications rely on this structure. To include support for dynamic groups (cf. chapter 3.1.3) an additional level will be implemented in ou=dynamic-groups,ou=groups, dc=dariah,dc=eu. The RBAC uses the cn of an object in the LDAP directory to

detect role definitions. Our proposal is the use a _dynamicgroup_ prefix for dynamic groups (e.g. cn=_dynamicgroup_<UUID>). To allow filtering such groups the attribute dariahIsDynamic=TRUE could be added to the dariahGroup objectClass.

```
dc=dariah,dc=eu
□ □ □  ou=deleted-people
□   □ □ □  uid=def
□       □ □ □  dariahDeleted=TRUE
□ □ □  ou=dsa
□   □ □ □  uid=jira
□       □ □ □  (!(objectClass=dariahPerson))
□ □ ou=federation-people
□   □ □ □  uid=xyz
□ □ ou=groups
□   □ □ □  cn=group1
□   □   □ □ □  member=uid=abc
□   □ □ ou=dynamic-groups
□       □ □ □  cn=_dynamicgroup_UUID2
□       □ □ □  cn=_dynamicgroup_UUID3
□       □   □ □ □  member=uid=abc
□       □   □ □ □  member=uid=xyz
□       □ □ □  cn=_dynamicgroup_UUID4
□ □ ou=people
□   □ □ □  uid=abc
□ □ □  ou=requests
□ □ □  ou=resources
□   □ □ □  cn=resource1
□       □ □ □  owner=abc@def.de
□       □ □ □  ispublic=FALSE
□       □ □ □  permissions=cn=_dynamicgroup_UUID3:read
□       □ □ □  permissions=cn=group1:write
□ □ □  ou=sessions
    □ □ □  cn=session1
    □   □ □ □  sessionuser=uid=xyz
    □ □ □  cn=session2
```

3.5 SelfService functionality

3.5.1 Creation of access tokens

To create an access token the users can use the DARIAH-DE Publikator. After logging in via DARIAH-AAI, the storage token the Publikator got from the PDP can be revealed easily.

3.5.2 Users manage their own access tokens

In order to have a consistent user experience, management of the access tokens a user was issued will be implemented in the DARIAH SelfService didmos LUI-based portal. A dedicated menu entry will be added to the existing SelfService menu. For each access token, the user sees the following information:

- Date issued
- Expiry date[7]

7 https://github.com/OAuth-Apis/apis/issues/33#issuecomment-450837319

- Application Name (OAuth2 client) the token was issued to
- Scopes (e.g "read", "write") the user has accepted

This structure is mainly informat∫ive. The only action the user can issue is to delete individual tokens.

The SelfService will access the ∫REST API of the AS directly, see 3.3.4 . Users can see and delete only their own tokens∫. Administrators would use the AS's own GUI described under 3.3 (or 3.3.3 specifically) in order to inspect or delete other users' access tokens.

4 Specification of the API

Accessing the StorageAPI [DARIAH-Storage-APIv1.0] via the SAML ECP profile has several disadvantages:

- Delegation (i.e. passing along the SAML “Token”) is not practical
- There are very few IdP in today’s federations that support the SAML ECP profile
- The ECP profile is not wide-spread. Most use-cases are covered by the much more accepted alternative OAuth2
- OAuth2 clients are simpler to implement than clients using the rather complicated ECP profile

Therefore the OAuth2 protocol seems to be a much better fit for the requirements of the DARIAH Storage API, also see[DARIAH-Storage-APIv2.0].

The following chapters are based on the RFC specification for OAuth2[8].

For every API call the header `X-Transaction-ID` can be set containing a logID string. If given, it must be included in the PDP's logging.

4.1 Client requests token

Requesting a token happens according to the OAuth2 specification. The relevant grant types are Authorization Code Grant (for server-side applications) and Implicit Grant (for JavaScript applications running in a browser). This token can be used to access the Resource Server via DARIAH Storage API.

4.1.1 Authorization Code Grant

Cf. Figure 1 and chapter 4.1 of the OAuth2 RFC. The client needs to do the following (here: Test-Client at the Test-AS):

- Before any authentication: registration with client_id, redirect_uri and intended scopes. AS issues a client_secret.
- During every authentication:
    - Redirect to

      `https://pdpdev.de.dariah.eu/oauth2/oauth2/authorize?response_type=code&client_id=pdp-test-client&redirect_uri=https://pdpdev.de.dariah.eu/pdptest/client/StDclient.php&scope=read&state=4675`
    - This endpoint is protected by the AS-SP, meaning the user is redirected to a IdP (if no session exists already). Afterwards the AS has access to the user’s ePPN and replies with

      `https://pdpdev.de.dariah.eu/pdptest/client/StDclient.php?code=379ec0f7-4830-483e-b7e8-776664cb4dd0&state=4675`
    - The client uses this Authorization Code in a REST call and exchanges it for an

8 http://tools.ietf.org/html/rfc6749

Access Token. This request happens directly at the AS, not as a redirect via the browser:

```
POST https://pdpdev.de.dariah.eu/oauth2/oauth2/token
Authorization: Basic <Base64(<client_id>:<client_secret>)>
Content-Type: application/x-www-form-urlencoded
(here the POST body:)
grant_type=authorization_code&redirect_uri=https://pdpdev.de.dar
iah.eu/pdptest/client/StDclient.php&code=79ec0f7-4830-483e-b7e8-
776664cb4dd0
```

- The client receives the Access Token from the AS

```
{"scope":"read","access_token":"47de4b99-408c-42bf-952c-
a9f682b55663","token_type":"bearer","expires_in":63072000}
```

4.1.2 Implicit Grant

In contrast to Figure 1 the Access Token is directly returned to the JavaScript-client (cf. Chapter 4.2 in the OAuth2 RFC). The client needs to do the following:

- Before: registration with client_id, redirect_uni and intended scopes (**no client_secret**) at the AS. **Implicit grant must be checked!**
- During every authentication:
  - Redirect to `https://pdpdev.de.dariah.eu/oauth2/oauth2/authorize?response_type=token&client_id=authorization-server-admin-js-client&scope=read,write&redirect_uri=https://pdpdev.de.dariah.eu/oauth2/client/client.html`
  - This endpoint is protected by the AS-SP, meaning the user is redirected to a IdP (if no session exists already). Afterwards the AS has access to the user's ePPN and redirects back to `https://pdpdev.de.dariah.eu/oauth2/client/client.html#access_token=33f948d8-9285-4f1f-9e9e-9ce2b7c6799c&token_type=bearer&expires_in=0&scope=read,write&principal=MartinHaase@dariah.eu`
  - The access token is already included here.

4.1.3 Using the token

Now the client can perform REST calls on the Resource Server (here: at the Test-RS):

```
GET
https://pdpdev.de.dariah.eu/pdptest/resource/StDResourceServer.php/
data/MartinHaase@dariah.eu/read
Accept application/json
Authorization bearer 33f948d8-9285-4f1f-9e9e-9ce2b7c6799c
```

The Resource Server (PDP) should answer accordingly.

4.2 Accessing resources

Creating, querying, modifying and deleting resources still works with the HTTP methods POST, GET, … as specified in [DARIAH-Storage-APIv1.0]. Also the responses of the Storage API does not change. Only the AAI has changed from PAOS / ECP to OAuth2 tokens to be included in the header, see [DARIAH-Storage-APIv2.0]:

- `Authorization: bearer` ***`<OauthAccessToken>`***

4.3 Validation of the Access Token and Callout to the RBAC PDP

In OAuth2, a Resource Server (here: the Storage API implementation) performs authorization itself. Since the API implementations are distributed and access rules should be maintained centrally, every API implementation must query a central Policy Decision Point (PDP) for such an access decision. In the TextGrid project this was done via SOAP requests at TG-auth*, which used an open source PDP based on openRBAC. This approach should be reused here with some modifications for the DARIAH infrastructure and using the OAuth2 protocol.

A tgCheckAccess request, used in TextGrid, contained a triple of (sessionID, resource, operation). The sessionID parameter will be replaced by the OAuth2 Access Token in this specification.

4.3.1 General Access Format for the PDP's API

A request at the RBAC can be described as follows:

```
<HTTP-VERB> https://<host>/dhauth/rbacRest/PDP.php/<resource>/...
Authorization: Basic <Base64 encoded api_key:api_secret>
Accept: application/json
X-Requested-For: <access_token>
```

After receiving such a request, the RBAC needs to querʃy the AS internally to validate the Access Token…

For unknown request pathes the PDP is responding with `404 Not Found` and the JSON string in the response body: `{"message":"Not found"}`.

4.3.2 Validation of the Access Token

…and receives the user's ePPN (`principal:attributes:name`). For this a HTTP request is used as follows:

```
GET https://<host>/oauth2/v1/tokeninfo?access_token=<access_token>
Authorization: Basic <Base64 encoded api_key:api_secret>
Accept: application/json
```

The AS returns a JSON object, containing the ePPN of the user:

```
{
  "expires_in": 1464251020060,
  "principal": {
    "attributes": {
      "IDENTITY_PROVIDER": "TODO",
      "DISPLAY_NAME": "Martin Haase"
    },
    "adminPrincipal": false,
    "groups": [],
    "roles": [],
    "name": "MartinHaase@dariah.eu"  <<<< ePPN
  },
  "scopes": [
    "read", "write"
  ],
  "audience": "dariah-de-publikator"  <<<< ID of the OAuth2 client
}
```

4.3.3 #checkAccess

Request

```
GET
https://<host>/dhauth/rbacRest/PDP.php/<resource>/checkAccess/<oper
ation>
Authorization: Basic <Base64 encoded api_key:api_secret>
Accept: application/json
X-Requested-For: <access_token>
X-Transaction-ID: <log_id>
```

The StorageAPI is registered at the AS/PDP with two strings, api_key and api_secret, which are used for authorization at the AS when performing requests. Additional parameters are included in a GET request:

- `<access_token>`
- `<resource>` contains an ID for the resource
- `<operation>` (more specifically: the privilege to perform an operation on the resource) as described in chapter 4.6 (e.g. read, write)

The X-Requested-For Header is optional if the operation equals "read" and the resource has the Flag TGisPublic=TRUE.

Response

A HTTP 200/40x/50x status code with a JSON string in the response body:

Success:

- `HTTP 200`

Access denied:

- `HTTP 40x with JSON Response data`

```
{"error":"error","error_description":"human_readable error
description"}
```

Error:

- `HTTP 50x` depending on the exact error, this might also include a JSON error as in `40x`
- Our proposal is to adapt the error codes from http://tools.ietf.org/html/rfc6749#section-4.1.2.1 (e.g. "invalid_grant", "access_denied", etc.)

Function:

What should happen during the checkAccess call?

- TGisPublic=TRUE and read: PERMIT
- TGisPublic=TRUE and OwnStorage=FALSE and write: DENY
- TGisPublic=TRUE and OwnStorage=FALSE and delete: DENY
- TGisPublic=TRUE and OwnStorage=TRUE and Owner matches, regardless of operation: PERMIT
- TGisPublic=FALSE and Owner matches, regardless of operation: PERMITIf no decision is possible using these rules, the group memberships (not roles) in rbacPermissions are checked

For every access decision the RBAC uses the intersection of scopes in the token and allowed operations according to roles. If the requested operation in within this intersection PERMIT, otherwise DENY.

4.3.4 #registerResource

Request

```
POST https://<host>/rbacRest/PDP.php/<ressource>
Authorization: Basic <Base64 encoded api_key:api_secret>
Accept: application/json
Content-Type: application/x-www-form-urlencoded
X-Requested-For: <access_token>
X-Transaction-ID: <log_id>
(hereafter the POST body:)
ownStorage=true
public=false
```

For an OwnStorage implementation `ownStorage=true` must be set in the POST body, for a PublicStorage implementation `ownStorage=false` must be set.

Response

- See #checkAccess

Function

What should happen during registerResource?

    - Set the owner in the resource (ePPN)
    - For `public=true` set **TGisPublic** to TRUE, FALSE otherwise.
    - Set the possible rbacOperations:
        - read
        - write
        - deleted
        - publish
    - Set **dariahIsOwnStorage** to the value of `ownStorage`, if missing, the default is TRUE
    - Currently no specific group permissions are set

4.3.5 #unregisterResource

Request

```
DELETE https://<host>/dhauth/rbacRest/PDP.php/<resource>
Authorization: Basic <Base64 encoded api_key:api_secret>
Accept: application/json
X-Requested-For: <access_token>
X-Transaction-ID: <log_id>
```

Response

- See #checkAccess

4.3.6 #publish

This API method of the PDP must be called, similar to the other PDP methods, by the resource server, i.e. the storage implementation.

Request

```
POST https://<host>/dhauth/rbacRest/PDP.php/<resource>/publish
Authorization: Basic <Base64 encoded api_key:api_secret>
Accept: application/json
X-Requested-For: <access_token>
X-Transaction-ID: <log_id>
```

Response

- See #checkAccess

Function

- Sets the **`TGisPublic`** flag for this resource to TRUE

4.3.7 #unpublish

Request

```
POST https://<host>/dhauth/rbacRest/PDP.php/<resource>/unpublish
Authorization: Basic <Base64 encoded api_key:api_secret>
Accept: application/json
X-Requested-For: <access_token>
X-Transaction-ID: <log_id>
```

Response

- See #checkAccess

Function

- Sets the **`TGisPublic`** flag for this resource to FALSE

4.3.8 #list

Request

```
GET https://<host>/dhauth/rbacRest/PDP.php/resources/list?
public=true&ownStorage=true
Authorization: Basic <Base64 encoded api_key:api_secret>
Acceppt: application/json
X-Requested-For: <access_token>
X-Transaction-ID: <log_id>
```

Response

- JSON Structure with ResourceIDs containing all the resources which are owned by the token holder:

```
[
  { “id”: “EAEA0-4BC3-2E22-246D-0”,
    “ownStorage” : true,
    “public”: true },
  { “id”: “EAEA0-4BC3-2E22-246E-0”,
    “ownStorage” : true,
    “public”: false },
  ...
]
```

- Error responses: see 4.3.3 (#checkAccess)

Function:

- If `public=true` is appended to the URL, the list will contain only public resources.
- If `public=false` is appended, the list will contain only unpublished resources

(registered with `public=false`).

- If `public` is not given, published and unpublished resources of this user are returned.
- If `ownStorage=true` is appended, the list will contain only OwnStorage resources (registered with `ownStorage=true`).
- If `ownStorage=false` is appended, the list will contain PublicStorage resources (registered with `ownStorage=false`).
- If `ownStorage` is not given, Own- and PublicStorage resources of this user are returned.
- Further appenders may be implemented in the future, such as `group=groupABC`, or other keywords, e.g. for administrative usage.

4.4 Response to the client

After a StorageAPI implementation performed a request at the PDP, generally the same responses are returned to the client as specified in the StorageAPI specification (version 2.0).

4.5 Operations on the DARIAH-DE Storage Implementations

It is possible to access the Storage service using an Access Token. This happens via the DARIAH Storage API (version 2).

The storage implementations must have unique IDs over both Storage implementation types. **No two resources (OwnStorage and PublicStorage) may have the same ID!**

4.5.1 OwnStorage Implementation

- URL: `https://cdstar.de.dariah.eu/dariah/`
- The PDP API `ownStorage` parameter must be set to `true`

4.5.1.1 #create

- OwnStorage performs a PDP#registerResource

Request

```
POST https://cdstar.de.dariah.eu/dariah/
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
Content-Type: <content_type>
```

Response

```
201 Created

Location: https://cdstar.de.dariah.eu/dariah/<object_id>

Content-Type: <content_type>
```

Errors

```
401 Unauthorized
```

4.5.1.2 #read

- OwnStorage performs a PDP#checkAccess/read

Request

```
GET https://cdstar.de.dariah.eu/dariah/<object_id>
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
```

Response

```
200 OK
Content-Type: <content_type>
(object data in HTTP body)
```

Errors

```
401 Unauthorized
404 Not Found
```

4.5.1.3 #update

- OwnStorage performs PDP#checkAccess/write

Request

```
PUT https://cdstar.de.dariah.eu/dariah/<object_id>
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
```

Response

```
201 Created
```

Error

```
401 Unauthorized
404 Not Found
```

4.5.1.4 #delete

- OwnStorage performs a PDP#checkAccess/delete and PDP#unregisterResource

Request

```
DELETE https://cdstar.de.dariah.eu/dariah/<object_id>
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
```

Response

```
204 No Content
```

Error

```
403 Forbidden
404 Not Found
```

4.5.1.5 #publish

- OwnStorage performs a PDP#publish

Request

```
POST https://cdstar.de.dariah.eu/dariah/<object_id>/publish
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
```

Response

```
204 No Content
```

Error

```
401 Unauthorized
404 Not Found
```

4.5.1.6 #unpublish

- OwnStorage performs a PDP#unpublish

Request

```
POST https://cdstar.de.dariah.eu/dariah/<object_id>/unpublish
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
```

Response

```
204 No Content
```

Error

```
401 Unauthorized
404 Not Found
```

4.5.1.7 #list

- OwnStorage performs a PDP#checkAccess/read and PDP#list
- The `public` parameter must be forwarded to the PDP as is comes with the #list request. If the parameter is not set in the OwnStorage#list request, if must not be set in the PDP#list request.

Request

```
GET https://cdstar.de.dariah.eu/dariah/list?public=true/false
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
```

Response

See chapter 4.3.8: #list

Error

```
401 Unauthorized
```

4.5.1.8 #info

- OwnStorage performs an OAUTH#tokeninfo

Request

```
GET https://cdstar.de.dariah.eu/dariah/auth/info
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
```

Response

See chapter 4.3.2: Validation of the Access Token

4.5.1.9 #checkAccess

- OwnStorage performs a PDP#checkAccess/<operation> on <object_id>

Request

```
POST https://cdstar.de.dariah.eu/dariah/<object_id>/checkAccess/
<operation>
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
```

Response

```
204 No Content
```

Error

```
401 Unauthorized
```

4.5.2 PublicStorage Implementation

- URL: `https://cdstar.de.dariah.eu/public/`
- The PDP API `ownStorage` parameter must be set to `false`

4.5.2.1 #create

- PublicStorage performs a PDP#registerResource

Request

```
POST https://cdstar.de.dariah.eu/public/
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
Content-Type: <content_type>
```

Response

```
201 Created
Location: https://cdstar.de.dariah.eu/public/<object_id>
Content-Type: <content_type>
```

Error

```
401 Unauthorized
```

4.5.2.2 #read

- PublicStorage does NOT need to perform a PDP#checkAccess

Request

```
GET https://cdstar.de.dariah.eu/public/<object_id>
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
```

Response

```
200 OK
Content-Type: <content_type>
(object data in HTTP body)
```

Error

```
401 Unauthorized
404 Not Found
```

4.5.2.3 #update

- Not implemented

4.5.2.4 #delete

- Not implemented

4.5.2.5 #publish

- Not implemented

4.5.2.6 #unpublish

- Not implemented

4.5.2.7 #list

- PublicStorage performs a PDP#checkAccess/read and a PDP#list
- The `public` parameter must be forwarded to the PDP as is comes with the #list request. If the parameter is not set in the PublicStorage#list request, if must not be set in the PDP#list request.

Request

```
GET https://cdstar.de.dariah.eu/public/list
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
```

Response

See chapter 4.3.8: #list

Error

```
401 Unauthorized
```

4.5.2.8 #checkAccess

- PublicStorage performs a PDP#checkAccess/<operation> on <object_id>

Request

```
GET https://cdstar.de.dariah.eu/public/<object_id>/checkAccess/
<operation>
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
```

Response

```
200 OK
```

Error

```
401 Unauthorized
404 Not Found
```

4.5.2.9 #info

- OwnStorage performs an OAUTH#tokeninfo

Request

```
GET https://cdstar.de.dariah.eu/public/auth/info
Authorization: bearer <access_token>
X-Transaction-ID: <log_id>
```

Response

- See chapter 4.3.2: Validation of the Access Token

4.6 Mapping HTTP operations and scopes

We propose the following mapping between operations in [DARIAH-Storage-API-v1.0_final.pdf], chapter 4 and operations (access rights) in checkAccess at the RBAC:

HTTP Verb	**Meaning in Storage API**	**Operation in RBAC/Scope**
GET	Retrieve Resource	read
POST	Create Resource	write

HEAD	Retrieve Resource Headers	read
PUT	Update Resource	write
DELETE	Delete Resource	delete
OPTIONS	Retrieve Server Options	n/a

This mapping table needs to be configured and documented in the individual StorageAPI implementations (OwnStorage and PublicStorage). Access must only be allowed if the PDP call (checkAccess, registerResource, …) was successful.

Figures