Colucci Alessandro

Promo 2020 – Communication System Security

Outpost24 France

Sophia Antipolis, France

2019, Aug 1st – 2020, Jan 31st

Federated Identity within Single Sign-On Systems

Authentication & Authorization for LEXIS Project

EURECOM Advisor: Marc Dacier – Professor – Track Responsible

Company Advisor: Frédéric Donnat – Cloud Technical Assistant

POLITO Advisor: Antonio Lioy - Professor

Rapport de stage confidentiel / Confidential thesis report
OUI / YES ☐ *NON / NO ☒*

EURECOM

EURECOM
Sophia Antipolis

DECLARATION POUR LE RAPPORT DE STAGE
DECLARATION FOR THE MASTER'S THESIS

Je garantis que le rapport est mon travail original et que je n'ai pas reçu d'aide extérieure. Seules les sources citées ont été utilisées dans ce projet. Les parties qui sont des citations directes ou des paraphrases sont identifiées comme telles.

I warrant, that the thesis is my original work and that I have not received outside assistance. Only the sources cited have been used in this report. Parts that are direct quotes or paraphrases are identified as such.

À : Antibes
Date : 31/01/2020

Nom Prénom : Colucci Alessandro
Name First Name

Signature :

POLITECNICO DI TORINO & EURECOM - TÉLÉCOM PARIS

Master Degree course in Computer Engineering

Master Degree Thesis

Federated Identity within Single Sign-On Systems

Authentication & Authorization for LEXIS Project

Supervisors
Prof. Marc Dacier
Prof. Antonio Lioy

Candidate
Alessandro Colucci

Company supervisor
Outpost24 France
Eng. Frédéric Donnat

Academic year 2019-2020

To my mother and my family, who always supported me to follow my path.

† To my father and my sister Giulia, who always look down upon me.

† To my grandmother, who taught me to wear a smile every day.

To Marzia, for having been an important part of my life and supported me even when I didn't believe in myself.

To Antonio, Lisa and Mateo, always staying by my side and accompanying me on this journey.

To my friends from "Prater-Eurecom", the best Italian group Eurecom has ever hosted.

To all the people I met on this journey, making it even more unique.

Summary

English version

The *LEXIS Project* aims at building an advanced, geographically-distributed, HPC infrastructure for Big Data analytics that will support the execution of large-scale test-beds in various industrial sectors. This work contains my contribution to the creation of the AAI system securing the whole LEXIS infrastructure (chapter 2). After comparing several Single Sign-On solutions (section 3.7) based on various analysis criteria (subsection 3.7.1), the Keycloak system was chosen representing the best fit for the project, thanks to its security features (subsection 3.7.2). The server was deployed through the implementation of an Ansible Playbook (section 4.2), in charge of installing all the system requirements and configuring the basic setup over the server or cluster nodes specified. Further studies were done on the Authentication and Authorization mechanisms supported by Keycloak (section 4.3), in particular on the configuration of the Keycloak *Clients* and the usage of JWT tokens (subsection 4.3.3). An hybrid approach was adopted to handle the Authorization in Keycloak (subsection 4.3.2) for LEXIS: an *RBAC matrix* was designed to provide the right set of permissions for users and groups in the system, merged with an *ABAC approach* for building up a finer-grained Access Control scheme. Finally, some research was done towards the assessment of possible vulnerabilities in the Identity and Access Tokens management through token forgery (subsection 4.3.4), eventually not identifying any flaw.

Version française

Le *Project LEXIS* vise à construire une infrastructure HPC avancée et géographiquement distribuée pour l'analyse des Big Data qui soutiendra l'exécution de bancs d'essai à grande échelle dans divers secteurs industriels. Ce travail contient ma contribution à la création du système AAI sécurisant l'ensemble infrastructure LEXIS (chapter 2). Après avoir comparé plusieurs Single Sign-On solutions (section 3.7) sur la base de divers critères d'analyse (subsection 3.7.1), le système Keycloak a été choisi comme le mieux adapté au projet, grâce à ses caractéristiques de sécurité (subsection 3.7.2). Le serveur a été déployé via la implémentation d'un Ansible Playbook (section 4.2), en charge de l'installation de toutes les exigences système et de la configuration de la configuration de base sur le serveur ou les noeuds de cluster spécifiés. D'autres études ont été effectuées sur les mécanismes d'authentification et d'autorisation (section 4.3) pris en charge par Keycloak, en particulier sur la configuration de Keycloak *Clients* et l'utilisation de jetons JWT (subsection 4.3.3). Une approche hybride a été adoptée pour gérer l-Autorisation dans Keycloak (subsection 4.3.2) pour LEXIS: une *RBAC Matrix* a été conçue pour fournir le bon ensemble d'autorisations pour les utilisateurs et les groupes du système, fusionné avec un *approche ABAC* pour construire un schéma de contrôle d'accès plus fin. Enfin, des recherches ont été effectuées en vue d'évaluer les vulnérabilités possibles dans la gestion des jetons d'identité et d'accès via falsification des jetons (subsection 4.3.4), sans finalement identifier aucune faille.

Acknowledgements

The work described in this Thesis was produced under the supervision of:

- Prof. Marc Dacier (academic supervisor from *EURECOM - Télécom Paris*)
- Eng. Frédéric Donnat (company supervisor from *Outpost24 France*)
- Prof. Antonio Lioy (academic supervisor from *Politecnico di Torino*)

A special thanks to all the Outpost24 colleagues for helping me in this activity and creating a friendly environment to work in.

Contents

List of Figures

Chapter 1

Introduction

The LEXIS Project [1] represents the answer to the H2020 project ICT 11 call: Large-scale EXecution for Industry and Society.

LEXIS aims at building an advanced, geographically-distributed, HPC infrastructure for Big Data analytics that will support the execution of large-scale test-beds in various industrial sectors. To ease the design and implementation of the infrastructure, LEXIS will be managed by three major components:

- cross-site infrastructure
- data management
- cloud provisioning services

The design and development of HPC/CLOUD/BD technologies in LEXIS is strongly oriented to provide a significant support to the pilot test-beds. The overall architecture of LEXIS has been designed as three integrated layers: Infrastructure Layer, Data Layer and Cloud services.

15 partners take part in the LEXIS Project:

- HPC Center: LRZ, IT4, ECMWF;
- Pilots: Avio Aero, CEA, CIMA, GFZ, AWI;
- Dissemination: Eiffage Teseo;
- Security: Outpost24;
- R&D: IT4I, LRZ, Bull, Cyclops, ECMWF, BayncoreLabs.

The LEXIS project aim to have its own AAI module/system to manage the LEXIS user and their access. The LEXIS infrastructure will be distributed among 2 different datacenters hosted in LRZ (Germany) and IT4I (Czech Republic). The AAI system will be used to store the user credentials (password, SSH keys, etc..) that are needed to access the LEXIS infrastructure and to store the permissions to access all part of LEXIS including:

- Distributed Data layer;

- Compute Layer both in HPC center and on OpenStack platform.

The goal is that each module of LEXIS project can use the AAI system to ensure the user have the proper permission to execute the request. For instance, the OpenStack platform can control that the user is allowed to start a virtual machine or run specific tasks. Another use case is that the Data Layer can ensure that the user has read access to some specific datasets.

The AAI module also needs to include federation as users shouldn't be able to authenticate once on LEXIS portal and then be able to run any simulation on any datasets that he has access.

The goal is to make a study on Open-source IAM system to identify the best choice for LEXIS project. Several Open-source implementations have been identified and can be found at [2].

The thesis will cover the State of the art of such SSO Open-source module that can be used with pros and cons. According to the constraints dictated by LEXIS Project, some recommendation will be required on how to use, deploy and setup such SSO system over the 2 Datacenters available in LEXIS.

The goal will be to make a "paper" analysis of such SSO system to short-list the 3 best fitting solutions to LEXIS' needs. Once the best solution (fitting the needs and requirements of LEXIS project) has been found, it will be necessary to install it on the distributed infrastructure spanning the 2 different Datacenters.

Some recommendations will be requested on how to properly use the SSO features to properly handle the access management (role, permission, RBAC, UBAC, ABAC, etc.). It will be required to identify different ways of using permissions and roles for users and groups in order to provide recommendation for properly configuring the SSO system according to LEXIS project needs and requirements.
The different scenarios will cover the following infrastructure part:

- Distributed Data Layer system among at least 2 different Datacenter
- HPC on 2 different Datacenter
- OpenStack on 2 different Datacenter

Chapter 2

The LEXIS AAI Infrastructure

2.1 LEXIS Infrastructure

This section will briefly describe the LEXIS Infrastructure and its components. All the information provided has been reported in the *Deliverable 4.1 of the LEXIS Project* [3], co-authored by myself.

From co-design activities started at the beginning of the LEXIS project, partner companies have identified and described the "LEXIS AAI" not only from a design perspective but also from a functional approach, taking into consideration the security aspects of the overall LEXIS platform. Figure 2.1

The goal for the LEXIS AAI is to provide a federated authentication and authorization system distributed among different data-centers (IT4I, LRZ), providing Federated Identity, Access and Data management. Moreover, the LEXIS AAI system will need to ensure that all the building blocks of the LEXIS infrastructure will be able to check for authentication and authorization for any user or process accessing LEXIS infrastructural resources (i.e. network, compute or storage resources). To this end, the following use-cases are covered:

- Restricting network access based on LEXIS AAI: A user can be allowed to get access resources on one specific data-center only;
- Restricting computation access based on LEXIS AAI: A user can be granted to use OpenStack resources in one data-center only or spanning several data-centers;
- Restricting data layer access based on LEXIS AAI: A user can be granted to get access to some specific datasets (e.g. weather and climate dataset) among all data-centers.

In terms of security, the design takes into account the fact that each building block of the LEXIS infrastructure is able to verify authentication and authorization for a user or process, avoiding service accounts that are security breaches for user isolation. If there is any security hole in a service account running a process for a user or another process, then it makes possible to elevate privileges and gain access to resources (network, compute, storage) that should not be accessed by the original user or process.

Figure 2.1 depicts the integration of the AAI system within the overall LEXIS platform. Interaction among different components of the LEXIS platform is also shown.

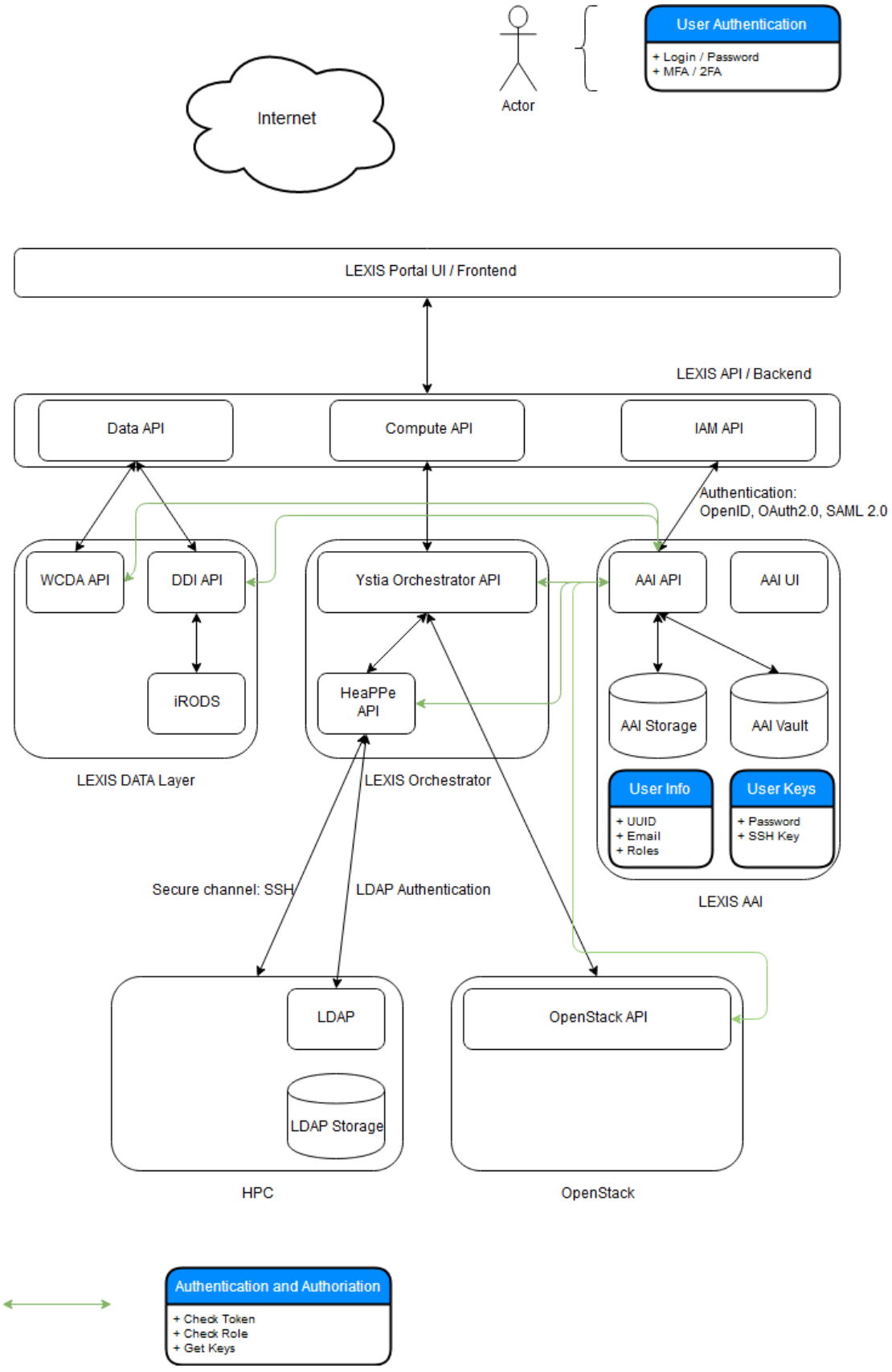

Figure 2.1. Authentication, authorization and identity system architecture.

As a matter of fact, usage of any LDAP server has been avoided in the design of the system, as it wouldn't properly provide:

- Real-time replication of data: LDAP systems are designed to work as master-slave components and thus they aren't designed to operate as a clustered system that can be distributed among different data-centers (although there are several ways of creating an LDAP cluster nowadays, it remains a non-optimal choice);
- Native Single Sign-On (SSO) support:
  - OpenID Connect;
  - SAML;
- Easy integration with web-services: the LEXIS portal will extensively use web services as the LEXIS platform is designed to work with both Cloud (OpenStack) and HPC technologies.

In case of creation of a pure and extensive platform based on HPC systems only, LDAP servers would have been an easier and better choice; however, as a hybrid platform was devised (composed of both HPC and Cloud resources which are based on REST API), then LDAP servers are definitely not the way to go. Conversely, the highly scalable system is built on top of a micro-services architecture which includes easy authentication and authorization services made for web services, such as Keycloak, Unity, OpenIAM, and others.

2.2 LEXIS Objectives and Requirements

This section describes the LEXIS objectives and requirements. All the information provided is reported in the Deliverable document [3].

2.2.1 AAI System for the LEXIS Platform: Objectives

The goal is to provide a federated Authentication & Authorization Infrastructure (AAI), able to manage the access to the LEXIS platform by users which can use resources in 2 different data-centers (IT4I and LRZ).

As a key element in the co-designed LEXIS architecture, the AAI is linked to 3 different parts of the LEXIS infrastructure, which define objectives to achieve with the designed AAI:

- *LEXIS Portal*: Front-End & Back-End portal will expose LEXIS functionalities and features to the end-user;
- *LEXIS Computational Resources*: Computational layer of LEXIS project which is hosted on 2 different supercomputing centers (IT4I and LRZ) and based on 2 different computational units in both infrastructures: HPC resources and OpenStack-based Cloud resources. To simplify the design phase, the VMware deployment has been considered as part of the OpenStack Cloud. In this layer, the AAI will support orchestration system, which is based on software modules, namely YSTIA & Alien4Cloud (provided by partner Bull/Atos) and HEAppE middle-ware (provided by partner IT4I);

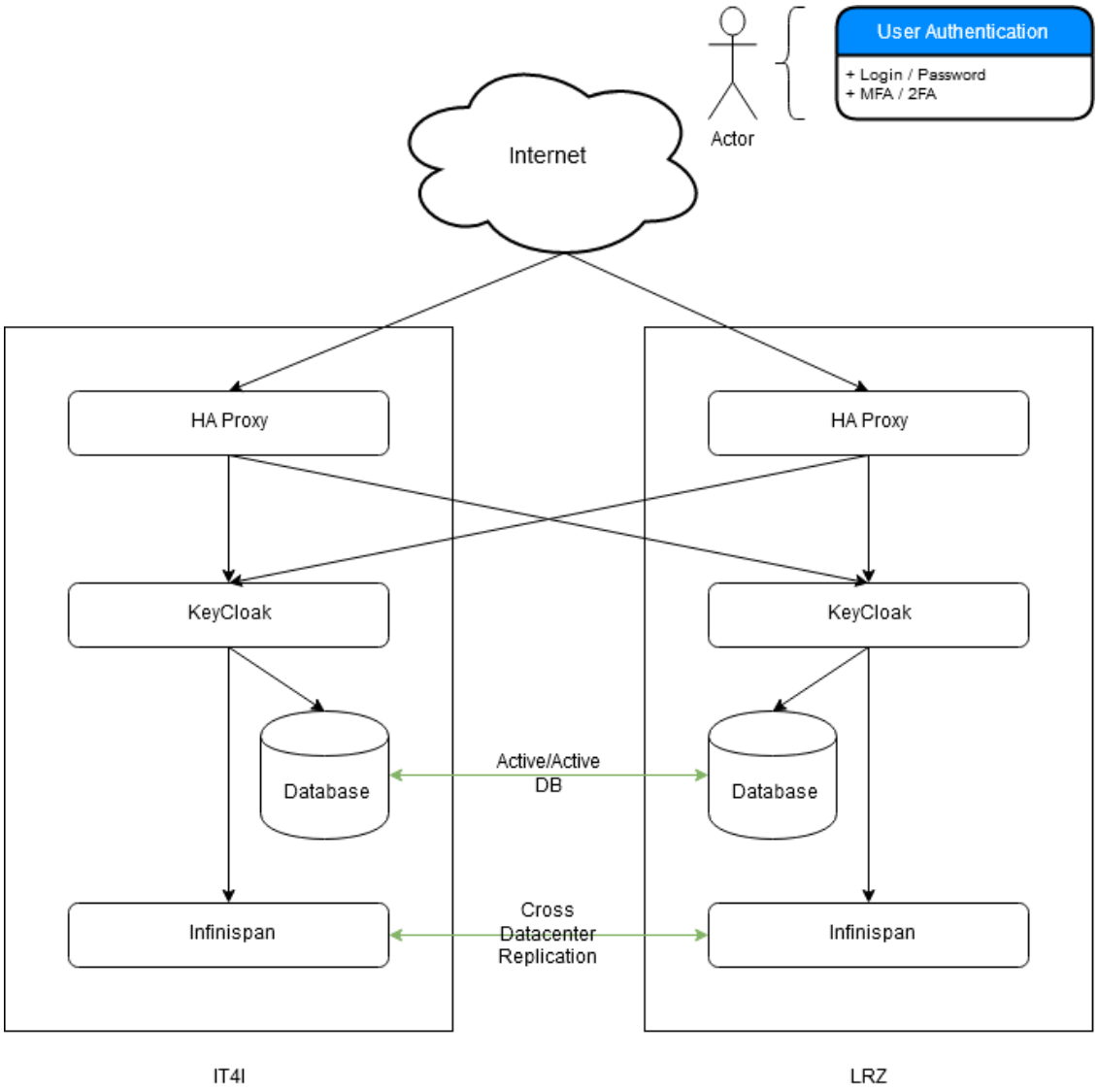

Figure 2.2. Cross-site AAI architectural configuration.

- *LEXIS Data Layer*: the Data layer of LEXIS platform spans over 3 different sites (i.e. IT4I, LRZ and ECMWF) and is based on several storage solutions, such as Ceph and iRODS.

A further set-up will be added for LEXIS AAI, as a fully-operational module following best recommendation and practice for production deployment. Figure 2.2 shows the cross-site architectural configuration for the devised AAI system, which allows keeping information synchronized across different infrastructures (i.e. data-centers).

As reported in the following sections, the AAI will achieve such main objectives by integrating a set of key technologies which have been selected in a way that they can cover all the requirements and specifications provided as input by the other components of the LEXIS project. To this end, the analysis made in this section helps in understanding the in-depth analysis of all relevant technologies reported in section 3.7.

2.2.2 Requirements and Specifications

This section contains the synthesis of the requirements that have been gathered for the 3 main building blocks of the LEXIS platform and that will interact with the LEXIS AAI:

- Compute Layer: This includes Cloud resources (with OpenStack) and HPC resources with all the relevant software tools;
- Data Layer: This includes DDI (Distributed Data Infrastructure) and the selected software tools;
- Portal Layer: It refers to the back-end portal connecting to LEXIS AAI.

Worth of noting that the first two layers (Compute and Data) will also contain the constraints brought by the existing infrastructure at IT4I and LRZ data-centers.

Cloud Computational Resources (Openstack)

OpenStack is the main technology at the basis of building the Cloud part of the LEXIS platform, and it is installed on both LEXIS federated infrastructures (IT4I and LRZ). OpenStack provides its own service for authorization and authentication, namely OpenStack Keystone (subsection 3.7.3), but it can also rely on other existing authorization and authentication services such as Keycloak (subsection 3.7.2).

OpenStack supports several Identity backends, such as LDAP, but it also supports 2 models for federation identity, especially the one called Keystone as a Service Provider which uses external identity provider (such as Keycloak or Google) as identity source and authentication method [4]. Also, OpenStack Keystone supports offloading authentication using SAML2.0 and OpenID Connect protocols.

From this standpoint, the requirements concerning the Cloud computational resources (i.e. OpenStack) can be summarized as follows:

- SAML2.0 or OpenID Connect protocol support, to be used in the LEXIS AAI front-end authentication protocol.

HPC Computational Resources

The HPC resources represents one of the main components of the LEXIS platform, as they are demanded to support a large portion of the pilot applications. The HPC unit of the LEXIS platform requires access to the users stored in LDAP systems installed in IT4I and LRZ infrastructures. Thus, the only requirement is to be able to configure an LDAP server on both data-centers. Moreover, HEAppE middle-ware will be used for executing tasks on behalf of the LEXIS user on the HPC computational part.
HEAppE will then use one of the protocols provided by LEXIS AAI to ensure authentication and access rights.

To summarize, the requirements for HPC part are the following:

- SAML2.0 or OpenID Connect protocol support for LEXIS AAI front-end authentication protocol.

Orchestration Layer (Ystia)

YSTIA is the solution selected as the orchestration tool in LEXIS. It provides the orchestration service (Yorc) and a frontend interface, namely Alien4Cloud. Here, YSTIA is referred to as the combination of orchestration service and the frontend. YSTIA supports any SAML Identity Provider or LDAP server for authentication through its front-end interface (Alien4Cloud).

However, YSTIA does not support any external provider for authorization. To have a complete Federated Identity Management system providing both authentication and authorization, a way has to be identified to map YSTIA roles with LEXIS AAI authorization system.

To summarize, the requirements for the orchestration layer are the following:

- SAML2.0 protocol support for LEXIS AAI front-end authentication protocol;
- or LDAP support for LEXIS AAI front-end authentication.

Middle-ware Solution (HEAppE)

Due to security reasons, there is no (easy) way to get access to the HPC computational resources from the outside of the HPC infrastructures. To solve this issue, an intermediary system must be put in place, in order to map internal HPC cluster users with external ones that want to access the resources. Furthermore, there is a need to keep track of used resources within the cluster (for monitoring reasons). To this end, LEXIS platform uses a middleware solution, namely HEAppE [5]. It provides all such capabilities, i.e. it creates a (dynamic) mapping between internal HPC cluster users that are allowed to use resources and external LEXIS users, and it is able to monitor resource usage (also for billing purposes).

To summarize, the requirements concerning the LEXIS middle-ware solution are the following:

- Support the mapping of internal HPC cluster users with LEXIS users.

Distributed Data Layer (iRODS)

The DDI (Distributed Data Infrastructure) layer will use iRODS [6] open source component. This software uses passwords as the default method for users authentication, stored in an iCAT database (i.e. a layer laying on top of a relational database used to store meta-data). iRODS also supports other authentication methods via specific plugins [7]:

- GSI: Specific authentication plugin for Grid Security Infrastructure;
- Kerberos: Usage of a Key Distribution Center and a Kerberos admin server;
- PAM: Linux Pluggable Authentication Modules is supported by iRODS and it can be configured to support LDAP server;
- OpenID: An experimental OpenID plugin exists for iRODS that allows support for OpenID protocol.

To summarize, the requirements for DDI system are the following:

- Kerberos support for LEXIS AAI front-end authentication;
- or LDAP support for LEXIS AAI front-end authentication;
- or OpenID protocol support for LEXIS AAI front-end authentication protocol.

Although iRODS plugins can cover all the current needs of LEXIS project, potentially other ones will be developed to facilitate and automate the integration of iRODS within the LEXIS platform.

LEXIS Platform Portal

The LEXIS back-end portal will be flexible and adapt to LEXIS AAI. As a matter of fact, commonly used software and protocols in IT (such as SAML2.0 or OpenID Connect) will be adopted.

Chapter 3

State of the Art

This section will briefly describe the main actors and characteristics in the context of Single Sign-On technology. Some sections have been already reported in the *Deliverable 4.1 of the LEXIS Project* [3] , co-authored by myself.

Visit RedHat's documentation on *Key Concepts and Terms* in the SSO environment [8] for further details.

3.1 Identity Federation

Identity federation is a way of linking a person's multiple digital identities, collecting all their attributes under the same entity, which can be shared among different domains to access several applications and services.

Identity federation is essential to Single Sign-On (SSO, cf. also NIST FIPS-800-63), where users' access is allowed after verifying their identity and issuing a single authentication 'ticket' - called *token* - vouching for their identity and allowing access across multiple systems and organizations. "SSO is a subset of federated identity management, as it relates only to authentication, it is understood on the level of technical interoperability and it would not be possible without some sort of federation" [9].

3.2 Single Sign-On

Single Sign-On (SSO) is a security mechanism for enhancing user's experience and security, as users' authentication is requested only once for different applications federated within the same *Identity Provider* and keeping user's credentials safe in the SSO server, preventing them from being cached by the actual service requiring them.
In general, this is achieved when the SSO Identity Provider (IDP) authenticates the user and then issues a security token which is sent to the target applications (i.e. *SSO Service Provider*) asserting the successful user's authentication and the reliability of their identity. This scenario can also involve an *Identity Broker*, consisting in an intermediary service with the purpose of creating a trust relationship between multiple Service providers with different Identity providers.

3.2.1 Identity Provider

An Identity Provider is a system in charge of providing identity management and authentication within a federated or distributed infrastructure. Upon successful user authentication, the Identity Provider returns an authentication token that can be used as proof of the user identity.
The Identity Provider usually authenticates a user by validating a username/password (with or without any MFA) but it can also rely on another method or even another trusted Identity Provider for authenticating the user.
The Identity Provider is in charge of managing the users' identities during its life-cycle (from creation to deletion). This system usually offers an API to facilitate integration with other applications such as web applications.

3.2.2 Service Provider

In this context a Service Provider is a system providing a "services" to an end user such as storage or processing. In modern software architecture (micro-services and not monolithic application), this system relies on another system called Identity Provider to provide Identity Management (authentication, authorization and management of the identity).
In most cases the Service Provider completely relies on the Identity Provider to provides any user's attributes (not only authorization), but it may happen that the Service Provider also manages some very specific user's attributes that are only used locally.
It's important to notice that such an architecture usually provides enhance usability from user perspective (usually coupled with SSO, user only needs to authenticate once) and better security (from Identity Management perspective, it avoids maintaining several identities in several places for the same user, thus reducing the attack surface).

3.2.3 Identity Broker

The Identity Broker is an intermediary service in charge of creating a trusted connection among different Service providers with external Identity providers. When the user tries to access a resource, the Service Provider redirects them to the Identity Broker, which is in charge of providing a list of available Identity providers to the user. Upon a successful authentication, the chosen Identity Provider issues a security token that will be used by the Service Provider to trust the authentication vouched by the Identity Provider and retrieve information about the user.

3.3 Authentication mechanisms and protocols

This section will briefly describe the large variety of authentication mechanisms offered by Single Sign-On systems, along with the standard protocols adopted.

Authentication is verifying the identity of a user, process or device which is often a prerequisite to allowing access to resources in an information system. This (cf. e.g. also NIST FIPS-200 [10]) is often a pre-requisite to giving access to a resource in an information system.

Usually, it's possible to differentiate between 3 authentication types:

- something you know: this includes passwords, PINs, secret passphrases, etc. In few words, it refers to anything you can remember and (then) say, do or perform when requested;
- something you have: this includes physical objects, such as keys, smartphones, USB sticks, smart cards or token devices. In few words, it refers to anything you own that can be provided when requested.
- something you are: this includes any part of the human body, such as the fingerprint, face, iris, etc. In few words, it refers to any part of the human body that can support the verification process when requested.

3.3.1 Multi-Factor Authentication

Nowadays, most of the information systems are using one of these authentication types and the number of systems using Two-Factor Authentication (2FA) or Multi-Factor Authentication (MFA) methods are growing. Two-Factor Authentication is the combination of 2 of these authentication types (similarly, Multi-Factor Authentication systems combine more than 2 of the above-mentioned mechanisms). For instance, in such a system, most of the time you need to provide a password and then a secret number generated on your physical device or smartphone.

According to Cryptography experts (cf. Handbook of Applied Cryptography [11]), it's better to be very careful in using fixed passwords and personal identification numbers (PINs), since their working scheme makes them fall under the category of 'symmetric key techniques providing unilateral authentication'. One-time password-based techniques are a step forward towards having a strong authentication mechanism in place.

On the other hand, strong authentication is provided using the *Challenge-Response* scheme, where the idea is that the user, process or device (claimant entity) proves its identity to the system (verifier entity) by demonstrating knowledge of a secret without revealing or providing it. This is usually done by providing a response to a time-variant challenge (usually random and secret) where the response depends on both the secret and the challenge.

Strong authentication is often confused with 2FA or MFA; however, unless multiple authentication factors are used, such as something you have and something you are, it cannot be considered an MFA.

3.3.2 Federation of web services and Social Login

Federation of web services, be it, e.g., social career platforms (e.g. LinkedIn, Xing) or code repository services (e.g. GitHub) is based on connection to trusted (usually federated) Identity providers (e.g. A. Singhal et al., *Guide to Secure Web Services* [12]). Using social network services or big online stores and Service providers (e.g. Facebook, Google, Amazon, etc.) as an Identity Provider, it is nowadays possible for a user to log into web services and portals with the credentials he uses to access the Identity Provider's main services (e.g. social network); this mechanism will be further referred as **Social Login** below.

Social login negates the need for the end-user to remember login information for multiple web sites and services, while providing site owners with uniform demographic

information as provided by the Identity Provider (e.g. basic data from the users' Facebook or Amazon profiles).
Social network-based login mechanisms are often deployed alongside the traditional registration and login mechanisms, representing a viable way for new users to register even if they don't have an account on a supported social network.

While Social login can be extended to some corporate websites, in the context of strictly secure applications it is often impossible to trust a third-party Identity Provider (e.g. banking or health system management). On the other hand, usage of Social login for suitable applications (e.g. forum, e-commerce or career network websites) is strongly increasing, as it can provide additional social features such as commenting, sharing, reactions and gamification. This is usually possible when Social login is implemented using the OAuth framework, even if it's always possible to implement it as a bare authentication system following the OpenID or SAML standard.

3.3.3 Standard Protocols

All analysed SSO systems are based on standard protocols and provide support for SAML (2.0), OpenID Connect and/or OAuth (2.0).

Usually, standard protocols provide an *Access token*, which is a token that can be provided as part of an HTTP request that grants access to the service being invoked on. This is part of the OpenID Connect and OAuth 2.0 specification.

SAML

Security Assertion Markup Language (SAML) is an XML-based open standard provided by OASIS for exchanging authentication and authorization data between different parties or entities. [13][14]

In its second version (SAML 2.0) this standard allows to even pass such data across what they called *security domains*. A security domain is a set of servers or computers that belong to the same domain and are using the same identity provider. In concrete terms, it allows to deploy Single Sign-On across different web applications among different domains.

The SAML 2.0 language allow to pass token from an Identity Provider to a Service Provider that authenticates the user and also contains some sort of attribute that describes the authorizations for the user (also called *assertions*).
Identity providers and Service providers need to agree on the SAML configuration to use: in order to work, both endpoints need to adopt the same configuration for the SAML authentication.

OpenID Connect

OpenID Connect (OIDC) [15] is an identity layer on top of an authorization framework that uses OAuth 2.0 protocol. It specifies a REST API using JWT (JSON Web token), allowing a Service Provider to communicate with an Identity Provider in order to authenticate and authorize a user.
The REST API enables all web-based applications, mobile applications, and similar (which are using secure channel over HTTP (HTTPS) for communication) to verify the

identity of the user through an Identity Provider and also retrieve additional information available in the IDP.

OpenID Connect uses more robust signing and encryption mechanism than OpenID 2.0 (integration of OAuth 1.0a with an extension) due to the integration of OAuth 2.0 in the protocol itself.

Another important feature is that OpenID Connect provides mechanisms for automatically discover Identity Provider (using some metadata) and allows to dynamically trust an external Identity Provider based on the trust for a common trusted third party.

OAuth

OAuth is a secure authorization protocol for *Access delegation*, achieved by the authorization server issuing an access token to a third-party client over HTTP services.
The protocol allows a user to grant access to their data (stored in a Service Provider) for web, desktop and mobile applications, without sharing their identity and credentials.

Compared to SAML, OAuth is a slightly newer standard, providing similar flows and functionalities with a much simpler client implementation, along with better support for mobile (and Internet application in general), by using JSON formatted communication.
On the other hand, SAML is a more widely applicable protocol, covering identity federation and management and authentication, most deployed in Centralized identity sources and Enterprise SSO scenarios. Apart from requiring an accurate configuration, SAML allows better control over SSO systems.

The first version of the protocol (OAuth 1.0 [16]) was first released in December 2007 and it was rapidly adopted by most industrial web-based applications for access delegation. However, a security flaw [17] required a minor revision (OAuth 1.0 Revision A [18]), published in June 2008.

The new version OAuth 2.0 [19] was introduced in 2012 and it's completely redesigned, not backwards compatible with the previous versions, although maintaining the general approach.
OAuth 2.0 introduces some important improvements to the protocol over the previous versions [20]:

- it allows the application to request authentication by sending the access token over HTTPS, without the need of signed requests using HMAC and token secrets;
- new methods (*OAuth Flows*) for obtaining an access token, to differentiate use cases and allow better support for non-browser based applications;
- simplified signatures, removing the need for special parsing, encoding, and sorting of parameters;
- new Refresh token, allowing the server to issue short-lived access tokens while clients can retrieve a new access token without requiring the user to grant access again;
- separation of roles between the authorization server issuing credentials and the resource server handling API calls, allowing better performances with an increase in scalability.

Kerberos

Kerberos [21] is a cross-platform authentication protocol, especially representing the default authorization technology used to login in Windows systems as well as one of the most adopted standards for web sites and Single Sign-On implementations.

Kerberos is based on a ticketing service, relying on a trusted entity called *Key Distribution Center (KDC)*, whose main task is issuing a *ticket-granting ticket (TGT)* to the client upon user's authentication by the *Authentication Server (AS)*. When trying to access a certain service, identified by its *Service Principal Name (SPN)*, the client gives back the TGT to the corresponding *ticket-granting service (TGS)*, which is time-stamped and encrypted with the TGS's secret key - and transparently renewed upon expiration by the *Local Session Manager (LSM)*.
Once verifying the ticket validity and checking access rights, the user is granted access to the *Service Server (SS)*.

Kerberos was first released in the 1980s, but it consisted of a huge improvement over previous technologies and still represents one of the best and most deployed protocols for authentication.
However, it has some drawbacks and limitations, mainly related to time constraints (time-stamped TGT has a narrow validity period, thus host's and server's clocks need to be synchronized) and the trust relationship between parties, especially relying on a central server's availability for ticket management and verification.

Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (LDAP) [22] is an open standard protocol for authentication based on *X.500 Directory Services*, which represent a way of storing and organizing network resources mapped to their corresponding addresses, arranging them in a naming structure and make them available within the network itself.

LDAP was created as a lightweight alternative to X.500 Directory Access Protocol (DAP), relying on the simpler TCP/IP protocol stack in place of the full Open Systems Interconnection (OSI) protocol stack.

LDAP is widely deployed to access usernames and passwords, stored in a central storage (*LDAP Server*), from different applications and services in order to validate users' identities.

Active Directory

Active Directory (AD) [23] represents Microsoft's directory server implemented using the LDAP protocol (version 2 and 3), designed to work with Microsoft Exchange Server, and Windows Domains in general. Like LDAP, AD is a directory service allowing third-party applications and services to access a centralized directory for user's authentication, providing also group and policy management.

Active Directory requires a central *Active Directory Domain Service (AD DS)*, usually called *Domain Controller*, which is in charge of managing all resources in the Windows Domain such as devices and users, authenticating and authorizing them and enforcing security policies in the network.
It also includes several other services, such as:

- Lightweight Directory Services (AD LDS): similar AD DS functionalities, but without requiring the definition of domains;
- Certificate Services (AD CS): acts as a PKI, in charge of issuing, validating and revoking certificates for files, emails and network traffic encryption;
- Federation Services (AD FS): complements AD DS with identity federation, enabling for Single Sign-On functionalities;
- Rights Management Services (AD RMS): an information rights management, allowing authorization control over corporate documents.

3.3.4 Consent

Commonly services' clients require access to the user identity. Once a user is successfully authenticated, the client might need to request their consent before being able to link to the user's account and retrieve their information.
This is usually done with a prompting screen, eventually specifying the information the application wants to access (such as name, email, etc.) and allowing the user to grant the request and decide which fields can be accessed by the client.

3.4 Client APIs

Client APIs (often called *Client Adapters*) represent exposed communication interfaces that can be used by applications and services to interact with the Identity Provider and request for user authentication.
Typically clients implement APIs to provide Single Sign-On functionalities to their services, being able to retrieve identity information or an access token to securely communicate with other services in the same SSO system.

3.4.1 Account Console

The Account or User console is the means through which users can manage their accounts (updating their profile information or changing their credentials), as well as handle linked accounts (when using Social login or Identity brokering, allow them to authenticate using different Identity providers), oversee sessions and set up advanced security mechanisms (e.g. Multi-Factor Authentication).

Self registration and User validation

Certain Single Sign-On systems provide Self-registration capabilities, which allow users to create their account without the need of an administrator's validation (thus easing and speeding up the workload for the account activation procedure). This functionality, that can be enabled by the system administrators, is usually based on an email validation mechanism, where a confirmation email is sent to the user, containing an activation code or link to provide in order to validate the user's identity and complete the registration process.

3.4.2 Admin Console

Users with administration rights can have access to the system's configuration, managing all aspects of the Single Sign-On server such as setting up Identity Brokering and User Federation; applications and services creation and management, and definition of fine-grained authorization policies; users management, including permissions and sessions.

Usually, these functionalities are available either as additional features in a custom User console or through a dedicated Admin console, which can be a graphical interface (GUI) or a command-line interface (CLI).

3.5 Authorization Services

Authorization Services are in charge of protecting application resources, managing user permissions and enforcing access policies.
Most of SSO systems support *fine-grained authorization policies*, usually offering different Access Control Mechanisms (ACM), generally based on the following schemes:

User-based Access Control (UBAC): permissions are granted at the individual level. Despite allowing more granular control of the system, this scheme leads easily to management overload, as access rights have to be defined for each user separately;

Role-based Access Control (RBAC): users are assigned a role (or a combination of roles), based on their job functions and privileges, determining which permissions they are granted. This scheme allows a more structured way of granting access, organizing roles in a hierarchy where higher-level roles subsume permissions owned by sub-roles;

Attribute-based Access Control (ABAC): access rights are granted to users through the combination of different types of attributes (such as user attributes, resource attributes and environment conditions), determining permissions based on who the users are instead of what they do (like in RBAC);

Context-based Access Control (CBAC): with the support of firewall software features that intelligently filter TCP and UDP packets based on the application layer session information, access rights are granted to the user according to a specific contextual attributes (such as organization, application, network resources, etc.) that the user is attempting to gain access to.

3.6 Auditing

Auditing is a key security functionality in a Single Sign-On system: it allows to keep track of users' activity (e.g. sign-on/sign-off, resource access sessions, privileges elevation, etc.) in a set of chronological records, which can be used for reporting, monitoring, statistical usage analysis or legal record keeping.

It's generally possible to configure the system selecting types or classes of events triggering the auditing mechanism, such as authentication-related events (e.g. login, registration, logout, token management, etc.) or account-related events (e.g. account linking, credentials or user information update, etc.). [24]

In addition to the Auditing system, SSO servers can provide their systems with *Event Listeners*, special procedures enabling to perform specific actions when a certain event occurs.

Systems offering this functionality usually come with predefined listeners, allowing to configure additional listeners based on custom events (e.g. in [24]).

Event records contain various information related to the performed action, such as the belonging category (e.g. appliance realm that generated the audit trail), the severity (in terms of security-relevant operations), time and outcome of the action (whether it was successful or a failure), along with the parties involved. [25]

3.7 Existing SSO Systems

This chapter will focus on the analysis of existing Single Sign-On systems, taking into consideration their benefits and disadvantages according to the defined requirements (section 2.2) and criteria (subsection 3.7.1).

3.7.1 Analysis criteria

This section describes the criteria used to analyse the considered solutions. This information is available in the Deliverable document [3] too.

Aiming at both looking at the most relevant functionalities of the AAI systems and establishing a frame of reference to compare them, the different solutions have been analysed under the following criteria:

- **License**: all the considered solutions being open-source, this criteria is only used to take trace of the license under which the corresponding solution is released;
- **Requirements**: it's the set of hardware and software requirements for the given solution, e.g. some solutions require the Java 8 JDK, for others just the installation of a RDBMS is necessary to deploy the system;
- **Clustering / Scalability**: high availability is one of the key features for a distributed system. A clustering set-up increments flexibility and reliability over failures and allows failover mechanisms;
- **Distributed / Multi site**: load balancing and multi-nodes deployment allow for a capillary coverage of the system network and generally better performances for the whole system;
- **Disaster Recovery / Backup**: this criteria is used to identify error recovery and backup functionalities, able to restore the state of the system after a fault occurs;
- **Authentication protocols**: relevant importance is taken from the authentication and authorization protocols used in the implementation.
  All the analysed solutions are based on most common standard protocols: SAML (generally 2.0), OpenID, OpenID Connect, OAuth (versions 1.0a [18, 17] and 2.0). Commonly used protocols are also Kerberos, WS-Federation (Passive), and others;
- **Authentication Integration**: this criteria defines all the functionalities offered by the authentication implementation. Usually these include: default username&password login (against local database), X.509/SSL certificate-based authentication, LDAP authentication, OTP/TOTP, MFA (2FA), Social login.
  In some cases it can make use of pre-defined implementations (eg. Shibboleth IDP , Spring Security framework [26], FIDO, and others);
- **Authorization Services**: it generally refers to the authorization scheme adopted by the system, which is commonly implemented either by RBAC, ABAC or UBAC models.
  Some solutions provide more advanced policy schemes, such as through XACML or Advanced Hybrid RBAC [27], offering an hybrid approach between RBAC and ABAC.
  More rare are the implementation based on ACL (Access Control List) or HBAC (Host and service-based Access Control model);

- **Functionalities**: relevant additional functionalities that may be offered by some implementations, such as caching; users grouping and management; lifecycle management for users, groups, identities and roles;
- **Auditing**: log all the security-relevant events and actions performed on the system by both users and administrators;
- **Storage**: storage type and features provided by the system;
- **Admin API**: API functions and UI available for administration purposes. It usually involves users management, roles assignment, system administration;
- **User API**: API functions and UI available for end-user, to manage their account and system resources they have access to;
- **Self registration / User validation**: option that enables end-user to create and activate autonomously their new account.
  This functionality can lighten the administration overhead provided by hand-checking every user, which however could be a stronger constraint for the system integrity;
- **Comments**: any other related functionality or comparison not involved in the previous analysis.

Every analysis has been structured in the same way, in order to ease the comparison among different solutions.

3.7.2 Keycloak

'Developed by JBoss (a Red Hat division), Keycloak [28] is an open-source Single Sign-On (SSO) solution with Identity and Access Management. It also provides Single Sign-On functionalities, managing user's logout in place of the applications belonging to the same realm. Keycloak also has **built-in support** to connect to existing LDAP or Active Directory servers or custom providers in other stores, such as a relational database (i.e., an RDBMS such as SQL Oracle or PostgreSQL). [29]

Besides the *Standalone Mode*, the program provides 3 different operating modes supporting **Clustering**:

- *Standalone Clustered Mode*: It requires a copy of the Keycloak distribution on each node in the cluster;
- *Domain Clustered Mode*: It provides a central place to store and publish configurations, common to all the nodes in the cluster;
- *Cross-Datacenter Replication Mode*: It allows to run Keycloak in a cluster across multiple data centers, typically located in different geographical regions. Each data center has its cluster in this mode.

Keycloak provides a **replication mechanism**, achieved by the usage of distributed caches among the nodes in the cluster.
A number of nodes (configurable as an attribute in the cluster's settings) are chosen as owner of the data, which are indeed not replicated to every single node in the cluster; thus any node has to query the cluster to obtain a specific cache entry that it doesn't own. For this reason, the availability of a specific piece of data is related to the nodes

that are hosting it: if all those nodes go down, the data is lost permanently. In these cases, the users will be logged out automatically and asked to login again.

Keycloak provides client adapters for several platforms and programming languages, but it is built on **standard protocols** like OpenID Connect, OAuth 2.0 and SAML 2.0, thus allowing integration with any application and service.
The **authentication** process is built upon different mechanisms: local user with password policy, OTP policy, Kerberos, X509 certificate + LDAP + Kerberos.
From the viewpoint of **authorization** policies, Keycloak supports the following ones: Attribute-based Access Control (ABAC), Role-based Access Control (RBAC), User-based Access Control (UBAC), Context-based Access Control (CBAC), Rule-based Access Control, Time-based Access Control + Support for custom Access Control Mechanisms (ACMs) through a Policy Provider Service Provider Interface (SPI). [30]

Keycloak also provides **Identity Brokering** features, as it provides Social login and authentication with existing OpenID Connect or SAML 2.0 Identity providers, configurable through the **Admin Console**, which provides functionalities for user management too.
Also, Keycloak provides a rich set of **auditing** capabilities [24], recording every user login action or even admin actions (e.g. configuration changes), which can be stored in the database and reviewed in the Admin Console. Furthermore, plugins can listen for these events through an additional listener SPI, allowing to interact with it and perform the appropriate action. Built-in listeners offer some basic auditing features, including a simple logger and the ability to send an email when specific events occur.
It is also possible to enable **User self-registration**.

Some further comments can be found in *StackHPC's article on Federation and identity brokering using Keycloak* [31].'

3.7.3 OpenStack Keystone

'Among the OpenStack services, Keystone [32] is the identity service offering authentication and authorization mechanisms, such as "API client authentication, service discovery, and distributed multi-tenant authorization through OpenStack's Identity API" [33].

Keystone maintains a central directory which keeps a mapping of the users with the services they can access, but it can also integrate existing backend directories (e.g. LDAP).
It supports multiple forms of **authentication** including standard username&password credentials, token-based systems, multi-factor authentication (MFA), time-based one-time passwords (TOTP), HTTPD authentication for mod_mellon and mod_shibboleth, X.509 Tokenless authorization. It supports all **standard protocols** such as LDAP, OAuth, oidc, SAML and SQL.
Additionally, users and third-party applications can determine accessible resources through a queryable list of all the services deployed in the corresponding OpenStack Cloud.

Like most of the OpenStack projects, Keystone defines Role-based Access Control (RBAC) policy rules. However, *group-based role assignments* are needed in the **authorization** scheme to facilitate **federation of users** by the Identity Service: groups objects will be defined, mapping all the belonging users to their local role assignments.

Keystone provides **enhanced auditing capabilities** through the implementation of the *PyCADF library*, capable of uttering notifications according to the DMTF CADF specification [34]: this standard provides "compliance with security, operational, and

business processes and supports normalized and categorized event data for federation and aggregation" [35].

There are two supported **clients**: *python-keystoneclient* project provides python bindings and *python-openstackclient* provides a command-line interface.'

3.7.4 Unity

'Unity [36] is an authentication service with Single Sign-On (SSO), providing identity management capabilities and offering federation and inter-federation management features.
It can be configured to integrate a **storage backend**, which can be:

- A typical relational database backend (RDBMS), such as SQL, MySQL, H2;
- Hazelcast distributed in-memory data grid (HZ) - overlay over RDBMS, offering in-memory operations computing, optimal for clustering and managing large traffic of data.

Unity does not provide **Auditing** functionalities yet. Nevertheless, it provides management of identities and entities, groups and attributes.
Authorization is indeed based on Role-based Access Control (RBAC), also supporting authorization on the group level.

Unity provides an Authentication system based on OAuth 2.0, OpenID Connect, SAML endpoints (Web and SOAP) and External LDAP, allowing users to login using a password or X509 certificate.

The **Web Admin UI** facilitates the operations of server management. The most important features of the Web Admin UI are:

- Management of attribute types, attribute classes, credential types and credential requirements (Schema management tab);
- Possibility to manage groups, their attribute classes and attribute statements (Contents management tab);
- Control over entities and identities and their group membership (Contents management tab);
- Full attribute control (Contents management tab);
- Management of registration forms, along with the possibility to list them instantly from the Web Admin UI and to manage the received requests (Registrations management tab);
- Possibility to create and load database dumps and to browse and trigger reconfiguration of endpoints, authenticators and translation profiles (Server management tab).

On the other hand, ordinary users can manage their profiles through the **Web User Home UI**, a simple interface to update their credentials and information about them.
User registration is allowed through *customizable registration forms*, that can be used to collect enrollment information about the user (typical use case) or retrieve it from a remote IDP (in case the user has been authenticated in such way), simply defining automated actions to be performed on newly created accounts.'

3.7.5 Other evaluated systems

The following systems have been fully analysed according to their documentation, in order to determine if they would satisfy the requirements for LEXIS project.
Even if the analysis of such solutions covered a large portion of time, a deep description of their features would be out of the purpose of this thesis. The reader can find all those features listed in the current subsection, as well as deepen the analysis with the cited references.

Apache Syncope

Apache Syncope [37]

Criteria	Description
License	Apache Licenses [38]
Requirements	HW requirements, Java JDK and JRE, Java EE Container, RDBMS
Authentication protocols	Extensions: SAML 2.0, OpenID Connect
Authentication Integration	based on *Spring Security* framework [26]; based on *ConnId* [39] for communication with Identity Stores
Authorization Services	based on *Spring Security* framework [26]
Functionalities	Users, Groups and Any Objects; Roles, Policies, Resources; Realms
Auditing	Audit [40] in the documentation
Storage	Identity Stores: Flat files (XML,CSV, ..), LDAP, RDBMS (MySQLm Oracle, ..), platform-specific (Microsoft Active Directory, FreeIPA, PowerShell, ..), Web services (REST, SOAP, ..), Cloud providers and more
Admin API	Admin console [41] in the documentation
User API	End-user application [42] in the documentation
Self registration / User validation	self-registration, self-service and password reset through End-User UI

Features table for Apache Syncope.

WSO2

WSO2 [43]

Criteria	Description
License	WSO2 Licenses [44] in the documentation
Requirements	Installation Prerequisites [45] in the documentation
Clustering / Scalability	Deployment Patterns [46] in the documentation
Authentication protocols	SAML2, OpenID Connect and *Web Services Federation (WS-Federation) Passive* [47])
Authentication Integration	X.509 certificate, IWA with Kerberos, Fast IDentity Online (FIDO), Time-based One-Time Password (TOTP); LDAP (ApacheDS, an external LDAP, Microsoft Active Directory, or any JDBC database), MFA, Adaptive Authentication
Authorization Services	Role-based access control (RBAC), eXtensible Access Control Markup Language (XACML) 2.0/3.0
Functionalities	User, Group management
Admin API	Calling Admin Services [48] in the documentation
Self registration / User validation	Self-Service Registration [49] in the documentation

Features table for WSO2.

OpenIAM

OpenIAM [50][51]

Criteria	Description
License	OpenIAM Identity and Access Governance - Licenses [52]
Requirements	1.2 System requirements [53] in the documentation
Clustering / Scalability	the OpenIAM deployment architecture allows you to select from either: - Application server-based clustering - Hardware load balancer in front of the UI layer and/or the Service layer. Both models will allow the load to be balanced across nodes and to failover in case a node in a cluster goes down.
Authentication protocols	SAML, OAuth 2.0, OpenID Connect; *Web Services Federation (WS-Federation)* [47] specification, defined in the *Web Services Security (WS-Security)* framework [54]
Authentication Integration	(Enterprise version) password, Directory (AD/LDAP), SSO protocols (SAML, OAuth 2.0, OpenID Connect), OTP policy (SMS/email/mobile with push notification), social authentication, Kerberos, Certificate-based authentication, Custom login module, Adaptive (Contextual) Authentication
Authorization Services	based on *Spring Security* framework [26]; RBAC Access control policies, Attribute-based Access Control- XACML (add-on)
Functionalities	User lifecycle management, Identity lifecycle management, Role lifecycle management (cf. OpenIAM Features [55] in the documentation)
Auditing	NoSQL audit repository (optional). "The audit service consists of the following components: - Event collectors capture audit events across different parts of the solution; - Queue, where audit events are published; - Audit Service, which takes care of logging the events. Also, audit events are signed so that any tampering of events can be detected" (from *Audit* in the *Technical Architecture Overview* [56], pp. 11-12).
Admin API	Administration Guide [57] in the documentation
User API	OpenIAM API [58] in the documentation
Self registration / User validation	Self-Service Guide [59] in the documentation

Features table for OpenIAM.

Gluu

The Gluu Server [60]

Criteria	Description
License	Gluu License [61]
Clustering / Scalability	Clustering for HA [62] in the documentation
Disaster Recovery / Backup	Gluu Server Backup [63] in the documentation
Authentication protocols	*Shibboleth SAML IDP* [64], OAuth 2.0 federation standards like OpenID Connect & UMA
Authentication Integration	SCIM, U2F, FIDO 2.0/WebAuthn, LDAP [65]
Authorization Services	Authorization policies with: User-based Access Control (UBAC), Attribute-based Access Control (ABAC), with Group-based and Role-based policies as a natural subset (cf. *User Managed Access (UMA) 2.0* [66])
Auditing	Audit Logging Configuration [67] in the documentation
Admin API	Admin Guide - Accessing the UI [68] in the documentation
Self registration / User validation	User Registration [69] in the documentation
Comments	Shibboleth for SAML & oxAuth for OAuth 2.0 [70]

Features table for Gluu.

Evolveum MidPoint

Evolveum MidPoint [71]

Criteria	Description
License	midPoint Licensing [72]
Clustering / Scalability	High Availability and Load Balancing [73] in the documentation
Authentication protocols	based on *Spring Security* framework [26]: Kerberos, OAuth 1(a) and 2.0, SAML 2.0
Authentication Integration	based on *Spring Security* framework [26]
Authorization Services	*Advanced Hybrid RBAC* [27] to tackle the problem of *Role explosion* [74]
Auditing	SQL repository + auditing to PostgreSQL, audit also to logs
Admin API	Admin/User Interface [75] in the documentation
Self registration / User validation	Self-Registration Configuration [76] in the documentation
Comments	RBAC/ABAC Hybrid Solution [77]

Features table for Evolveum MidPoint.

eduGAIN

EDUcation Global Authentication INfrastructure (eduGAIN) [78]

Criteria	Description
License	From custom repository: eduGAIN License [79]
Requirements	RDBMS (MariaDB)
Authentication protocols	SAML Shibboleth IdP, midPoint
Functionalities	eg. Shibboleth, Jetty, LDAP, MySQL
Self registration / User validation	eduGAIN Access Check [80] in the documentation
Comments	TNC-2018 INTERNET2 GEANT Campus IdP v1.0 [81] (Platform architecture on slide 11)

Features table for eduGAIN.

Soffid

Soffid [82]

Criteria	Description
License	Soffid License [83]
Requirements	RDBMS
Clustering / Scalability	Multi-master MariaDB cluster [84] in the documentation
Disaster Recovery / Backup	System backup [85] in the documentation
Authentication protocols	SAML and OpenID bridge [86]
Authentication Integration	LDAP directories, MS Active Directory, RDBMS and most common Operating Systems; Two-Factor Authentication (2FA)
Authorization Services	RBAC; also XACML optional module available to define Attribute-based control policy (ABAC)
Functionalities	fine tuning permissions based on organisation role, organisation unit or granted roles
Auditing	IAM: Audit & Compliance [87] in the documentation
Storage	it supports certain number of RDBMS including MariaDB, MySQL, Oracle and SQL Server
User API	A single, simple and intuitive web interface for the end user to manage their own profile, request passwords, access directly to their applications and manage their own business processes.
Self registration / User validation	Web services reference [88] in the documentation

Features table for Soffid.

JOSSO

JOSSO [89]

Criteria	Description
License	JOSSO2 License [90]
Requirements	JRE 8 or newer
Clustering / Scalability	About High Availability [91] in the documentation
Authentication protocols	SAML 2.0, OAuth 2.0, OpenID 2.0, SSL for certificate-based, ID confirmation with OAuth 2.0 Access Token Issuance
Authentication Integration	(following protocols order) LDAP for Directory-based auth., Kerberos for Integrated Windows auth., WiKID Two-Factor Authentication (2FA) with OTP, SSO Domino Auth. with Lightweight Third-Party Authentication (LTPA from IBM), Certificate-based auth. via SSL, JBoss Enterprise Portal Platform (or JBoss EPP)
Authorization Services	Role-based Access Control (RBAC) with both accounts and groups provisioning; Identity Appliance Life Cycle Management, Account and Entitlement Management
Auditing	Auditing [25] in the documentation
Admin API	Java API for RESTful Services (JAX-RS) [92] in the documentation
User API	JAX-RS - Sample Client code [93] in the documentation
Comments	based on *J2EE* [94] and *Atricore IAM Platform* [95][96]

Features table for JOSSO.

Shibboleth

Shibboleth [97]

Criteria	Description
License	Apache License Version 2.0 [98]
Requirements	System Requirements [99] in the documentation
Clustering / Scalability	Clustering [100] in the documentation
Authentication protocols	SAML 1.1 and 2.0, CAS 2 [101]
Authentication Integration	LDAP, Kerberos, JAAS, X.509, SPNEGO, Duo Security, and container-based authentication systems
Authorization Services	Authorization policies with Access Control (type not specified)
Functionalities	Interesting Features [102] in the documentation
Auditing	Audit Logging Configuration [103] in the documentation
Admin API	Administrative Configuration [104] in the documentation
Self registration / User validation	no user registration system
Comments	Shibboleth for SAML & oxAuth for OAuth 2.0 [105]

Features table for Shibboleth.

Apereo CAS

Apereo CAS [106] CAS Protocol 2.0 specification [101]

Criteria	Description
License	CAS License [107]
Requirements	Installation Requirements [108] in the documentation
Clustering / Scalability	High Availability Guide (HA/Clustering) [109] in the documentation
Distributed / Multi site	Multiple CAS Server Nodes [110] in the documentation
Authentication protocols	CAS Protocol version 1,2 and 3 (exclusive for CAS), SAML 1.1, OAuth 1.0 and 2.0, OpenID Connect, SCIM and WS-Fed; possible to integrate with Shibboleth IdP (SSO for Shibboleth IdP [111] in the documentation)
Authentication Integration	Database, JAAS, LDAP, OAuth 1.0/2.0, OpenID, RADIUS, SPNEGO (Windows), Trusted (REMOTE_USER), X.509 client SSL certificate, Remote Address, YubiKey, Apache Shiro, pac4j; Multi-Factor Authentication (MFA) (cf. Authentication Handlers [112] in the documentation)
Authorization Services	CAS ABAC, Custom ABAC, LDAP support (cf. Securing Access and Authorization [113] in the documentation)
Functionalities	Java (Spring Webflow/MVC servlet) server component; Pluggable authentication support (LDAP, database, X.509, 2-factor); Support for multiple protocols (CAS, SAML, OAuth, OpenID); Cross-platform client support (Java, .Net, PHP, Perl, Apache, etc); Integrates with uPortal, Liferay, BlueSocket, Moodle, and Google Apps to name a few
Auditing	Audits [114] in the documentation

Features table for Apereo CAS.

Perun

Perun AAI [115]

Criteria	Description
License	Perun License [116]
Requirements	Database (SQL, Oracle, PostreSQL), JAVA
Authentication protocols	Kerberos, Shibboleth IdP, Certificate or REMOTE_USER like Apache config
Authentication Integration	LDAP, Import (XML, CSV), SQL DB, G Suite Connector is outdated (using P12 file instead of JSON)
Authorization Services	privilegies associated with the user (cf. Perun RPC API [117])
Functionalities	User, Group, (No roles), User lifecycle management
Auditing	Audit trail, but nothing about how to consume or export it
Storage	Database (SQL, Oracle, PostgreSQL)
Admin API	Perun RPC API (Not REST) [117]
Self registration / User validation	User management [118] in the documentation (under *Perun benefits - Virtual organizations managers*)
Comments	GitHub repository [119]

Features table for Perun.

FreeIPA

FreeIPA [120]

Criteria	Description
License	FreeIPA License [121]
Clustering / Scalability	Assuming that clients do intelligent caching, there is no need for many FreeIPA servers to handle the load from all clients. (cf. Scalability [122] in the documentation)
Disaster Recovery / Backup	Backup and Restore [123] in the documentation
Authentication protocols	SAML; LDAP
Authentication Integration	System Security Services Daemon (SSSD); Kerberos (with mod_auth_gssapi or mod_auth_kerb); Pure Application Level, Kerberos SSO (ticket), SAML-based, Certificate-based; Login form-based (cf. Web App Authentication [124] in the documentation)
Authorization Services	Host and service based access control (HBAC), with access control check to the Kerberos authentication method on the Apache level in order to prevent access to unauthorized users that are able to get the Kerberos ticket (Note: default rule allow_all grants access from anywhere to anywhere to any user and service. It needs to be disabled)
Functionalities	The FreeIPA Directory Service is built on the 389 DS LDAP server, acting as data backend for all identity, authentication (Kerberos) and authorization services and other policies
Auditing	Session recording - Audit recording details [125] in the documentation
Admin API	Web UI or CLI
User API Self registration / User validation	Self-Service Password Reset [126] in the documentation

Features table for FreeIPA.

OpenAM

OpenAM [127]

Criteria	Description
License	OpenAM License [128]
Requirements	fully qualified domain name (FQDN); Java JRE; Docker and Apache HTTP Server
Clustering / Scalability	"OpenAM provides both system failover and session failover. These two key features help to ensure that no single point of failure exists in the deployment, and that the OpenAM service is always available to end-users. Redundant OpenAM servers, policy agents, and load balancers prevent a single point of failure. Session failover ensures the user's session continues uninterrupted, and no user data is lost." (from section *High availability* in OpenAM Features - Wikipedia [129])
Authentication protocols	SAML, OAuth 2.0, OpenID Connect 1; possible setup of Web-Authn standard by W3C and FIDO
Authentication Integration	Authentication modules [130] in the documentation
Authorization Services	"Authorization policy from basic, simple, coarse-grained rules to highly advanced, fine-grained entitlements based on XACML (eXtensible Access Control Mark-Up Language). Authorization policies are abstracted from the application, allowing developers to quickly add or change policy as needed without modification to the underlying application." (from section *Authorization* in *OpenAM Features - Wikipedia* [129])
Auditing	OpenAM Audit Logging [131] in the documentation
Admin API	OpenAM provides client application programming interfaces with Java and C APIs and a RESTful API that can return JSON or XML over HTTP, allowing users to access authentication, authorization, and identity services from web applications using REST clients in their language of choice. OAuth 2.0 also provides a REST Interface for the modern, lightweight federation and authorization protocol.
Self registration / User validation	Legacy User Self-Service [132] in the documentation

Features table for OpenAM.

Univention Corporate Server UCS

Univention Corporate Server [133]

Criteria	Description
License	Univention Corporate Server License [134]
Requirements	1GB memory and 8GB hard drive space
Distributed / Multi site	Fault-tolerant domain setup [135] in the documentation
Disaster Recovery / Backup	Domain controller backup [136] in the documentation
Authentication protocols	SAML, Kerberos, SSL certificate, LDAP, OpenID Connect; RADIUS; UMCP 2.0 (based on JSON)
Authentication Integration	LDAP, Kerberos, Two-Factor Authentication (2FA) (for example a TAN generated randomly each time), Certificate-based auth. via SSL
Authorization Services	Access to the information contained in the LDAP directory is controlled by Access Control Lists (ACLs) on the server side; RADIUS
Functionalities	Group and computer management, IP and network management, file share management
Auditing	possible, on a share-by-share basis even. You'll have to enable the audit VFS module for the share (from UCS Forum)
Admin API	Univention Management Console (UMC) Documentation [137]
User API	"Web interface of UCS system. Microsoft Windows clients and Mac OS X systems are integrated via a Samba-based, AD-compatible Windows domain; most Linux distros (Ubuntu, Debian, SUSE or RedHat) can also be integrated into the domain" (cf. UCS Clients [138] in the documentation). Web browser with Dojo/UMC JavaScript API, communication to UMC HTTP server via AJAX and JSON (UMC Architecture [139])
Self registration / User validation	UCS Self Services [140] in the documentation
Comments	Release of UCS 4.1 with Docker, Single Sign-On Mechanism and Two-Factor Authentication [141] in the documentation

Features table for Univention Corporate Server (UCS).

Aerobase IAM

Aerobase IAM [142]

Criteria	Description
License	Aerobase IAM License [143]
Requirements	Java 8 JDK, HW requirements (512M RAM + 1GB storage), a shared external database (PostgreSQL, MySQL, Oracle, etc) for Cluster mode
Clustering / Scalability	Clustered Mode [144] in the documentation
Authentication protocols	OpenID Connect, OAuth 2.0, SAML 2.0; LDAP, Kerberos
Authentication Integration	LDAP, Identity brokering, Social login (Google, GitHub, Facebook, etc), Kerberos bridging, Two-Factor Authentication (2FA)
Functionalities	central management of users, roles, role mappings, clients and configuration through Admin console
Admin API	Admin REST API [145] in the documentation
User API	web apps and RESTful web services
Comments	Server Features [146] in the documentation

Features table for Aerobase IAM.

Chapter 4

Solution Design

4.1 The chosen solution: Keycloak

After a detailed comparison among the three main above-mentioned solutions, **Keycloak** has been chosen as the solution to be benchmarked, since it appears to cover all the requirements and specifications collected (cf. subsection 3.7.2), while providing some additional benefits with respect to OpenStack Keystone and Unity:

- it's an open-source solution, supported by RedHat and a very active community, allowing to perform User Federation, Authentication and Authorization in a single framework. The provided user federation functionalities allow to connect with external databases through LDAP and Active Directory;
- it provides Identity Brokering functionalities, allowing to use external Identity providers or Social Networks (such as Google, GitHub, Facebook and Twitter) to authenticate to Keycloak, with built-in support for OpenID Connect and SAML 2.0 protocols;
- it allows to define different Clients for different purposes(cf. subsection 4.3.1), which can be used e.g. to interface with web applications or back-end services;
- it allows to define actions (*Client Scopes*) that Clients can perform over some resources, and *Roles* to associate such actions with related permissions;
- it embeds further authentication functionalities, such as declaring requirements for different OIDC Flows (which can be linked to specific actions on the server, e.g. registration, browser authentication or credentials reset), specifying required actions (to be performed at each new user registration or requested by the server administration), and defining policies for password and OTP.

Thanks to its versatility, Keycloak can interface with all the components involved in the LEXIS project. It's worth to mention that **iRODS integration**, responsible of data management and virtualization, required some adjustments to connect with Keycloak: through an authentication plugin, iRODS can take advantage of OpenID login using tokens provided by an OpenID provider; however, JWT Tokens used by Keycloak are generally too long to be passed through the iRODS client-server interface.
As exposed in [147], two solutions were implemented to address such problem:

- for web-based applications (directly interfacing with the user), a helper thread is launched to retrieve the authentication redirection URL and pass it to the main thread, which will be performing the user authentication, and then redirecting to the web portal. The helper thread will be also performing a query to the iRODS system and store the result only after successful authentication of the user;
- for back-end applications, the iRODS authentication microservice has been modified implementing opaque tokens, simply consisting of the hash of the actual token, small enough to be passed through the iRODS client-server interface. Upon receiving a hash, the microservice database is used to retrieve the token and perform the verification operations.

The Proof of Concept with Keycloak had the purpose of excluding major obstacles to the usage of Keycloak in the LEXIS ecosystem. Due to the success of its PoC, Keycloak remained the selected solution.

4.2 Deployment with Ansible

The need for a uniform and automated method of deployment for the AAI system led to the creation of an **Ansible** script, allowing to standardize the installation across multiple server or cluster nodes, using the same configuration.

Ansible is a software based on *YAML language* allowing to specify a list of tasks to be performed over the given hosts (either locale or remote, eventually divided into operational groups), in a concise and schematic fashion.
Ansible allows running both ad-hoc commands or scripts, the latter called **Playbooks**, representing a collection of *plays* which in turn consist of a group of *tasks* to be performed over different hosts or groups of hosts. Tasks can also be grouped in so-called *roles*, grouping tasks to be performed together.

Ansible comes with an additional utility called *Ansible-Galaxy*, allowing to share roles with other users in the Ansible community. I used some of the shared Roles available through Ansible-Galaxy, in particular:

- Mariadb by adfinis-sygroup [148]
- Keycloak by andrewrothstein [149]

These roles represented a useful starting point for the system deployment, despite needing some additional changes in its setup and configuration.

I first worked on writing the **Ansible Playbook** to deploy the system, achieving the following tasks:

- install system requirements, such as Python and Java JRE:

```
## [site.yml]
pre_tasks:
  - name: install python 2
    raw: test -e /usr/bin/python || (apt -y update && apt install
    -y python-minimal)
  - name: install java jre
    raw: test -e /usr/bin/java || (apt -y update && apt install -y
     default-jre)
```

- install and configure MariaDB, according to [148]:

```
## [site.yml]
- name: configure mariadb servers
  hosts: dbservers
  become: yes
  roles:
    - ansible-role-mariadb
```

- install the Keycloak server, according to [149], and set the system into a basic configuration, e.g. creation of 'root' user in the 'master' realm:

```
## [site.yml]
- name: configure keycloak servers
  hosts: ssoservers
  become: yes
  roles:
    - ansible-keycloak

## [roles/ansible-keycloak/tasks/main.yml]
# create first user
- name: create root user
# Note: change password once logged-in
  shell: "{{ keycloak_install_dir }}/bin/add-user-keycloak.sh -u
    root -p root"
  notify: restart standalone service
```

- set-up and launch the server as a system service, automatically starting at booting time.

```
## [roles/ansible-keycloak/tasks/main.yml]
- name: copy service configuration
  with_items:
    - f: keycloak.standalone.service
      d: /etc/systemd/system
  template:
    src: "{{ item.f }}.j2"
    dest: "{{ item.d }}/{{ item.f }}"
    mode: "{{ item.m|default("0644") }}"
  notify: reload systemctl

## [roles/ansible-keycloak/handlers/main.yml]
- name: reload systemctl
  become: true
  command: systemctl daemon-reload
- name: restart standalone service
  become: true
  service:
    name: keycloak.standalone.service
    state: restarted
```

For the Ansible deployment guide please refer to Appendix A.

4.3 Authentication and Authorization mechanisms

Keycloak allows to create different **Realms**, sets of *Users* belonging to it that can be organized in *Groups* and *Roles*, allowing to manage access with various Authentication mechanisms (subsection 4.3.2). Every realm is isolated from the other ones hosted on the same system, thus users can only authenticate on a realm they are belonging to.

Authentication and Authorization for a user is handled by some APIs, called **Clients** in the Keycloak terminology, which are compliant to Standard Protocols (subsection 3.3.3) and can define different ways to manage identities and access rights: they represent the means for applications and services to interface with the server and the Realm, authenticate a user and request an Identity or Access Token (subsection 4.3.3). These tokens represent the building blocks of this process and they are used in conjunction with OpenID Connect to provide the system with Single Sign-On functionalities.

4.3.1 Authentication in Keycloak

In Keycloak, the **Authentication** of the user is performed through *Clients*, offering different mechanisms depending on their **Access Type**:

- **public**: suitable for client-side applications, thus requiring to restrict their access on the server-side by proper URIs redirection, since it would be impossible to maintain the confidentiality of their credentials and securely authenticate them;
- **confidential**: designed for server-side applications, this type of Clients needs to additionally submit their credentials (issued by the authorization server) in order to authenticate themselves and be able to perform an *Access Token Request*;
- **bearer-only**: differently from the previous types (designed for browser login), it only allows Bearer token requests, making it suitable for services which are never initiating a login (e.g. an application connecting to a database).

Clients can authenticate using OpenID Connect or SAML, depending on the type of applications they are securing: OpenID Connect is a newer protocol, designed for the web through HTML5 and JavaScript and offering many predefined features, making it easier to integrate and implement on the client-side; SAML, however, is more mature and has been chosen as a standard for many years, offering more flexibility and control over many features later introduced in the OIDC standard, and being retrofitted to work on top of the web, despite being more verbose than OIDC.
For its better compatibility with web applications, **OpenID Connect** has been selected as the **recommended standard in Keycloak**.

OIDC Clients can perform different types of authentication based on the **OIDC/OA2 standard flows** enabled [150]:

- **Authorization Code Flow**: it's the recommended approach for browser-based applications, as it is heavily depending on URL redirections: when attempting to login, the user visits the application, which will redirect the browser to Keycloak for authentication; upon successful authentication, Keycloak will use a callback URL (*redirect URI*), previously provided by the application, to issue a *temporary authorization code* (as a query parameter in the callback), to be used for retrieving an *Identity*, *Access* and *Refresh token* through a REST invocation. In Keycloak, this flow is available by enabling the *Standard Flow* option;

- **Implicit Flow**: it's designed for browser-based application as well. It's very similar to the previous flow, except for the usage of the temporary code: upon authentication, Keycloak redirects to the application through the callback, directly passing the *Identity* and *Access tokens* as query parameters. Since no *Refresh token* is issued, access tokens have to be very long-lived, representing a possible security issue; the only solution to cobble this problem would be to re-authenticate the user after their access token expired. In Keycloak, this flow is available by enabling the *Implicit Flow* option;
- **Resource Owner Password Credentials Grant**: it allows REST applications to retrieve a token on behalf of the user, having access to their credentials and submitting them via an HTTP POST request; the corresponding *Identity*, *Access* and *Refresh tokens* are issued. In Keycloak, this flow is available by enabling the *Direct Access Grants* option;
- **Client Credentials Grant**: it allows REST clients to request a token dedicated to themselves, after authenticating their credentials; for this reason, this type of Clients need to be *confidential*. In Keycloak, this flow is available by enabling the *Service Accounts* option.

When needed, a confidential OIDC Client can authenticate against the Keycloak server in 4 different ways:

- *Client ID and Client Secret*: the Keycloak server appoints the Client with a secret, which needs to be know from both parties to successfully authenticate the application;
- *X509 Certificate*: the Client can be authenticated by signing messages with its X509 Certificate, vouching for its identity;
- *Signed JWT*: during authentication, the Client generates a JWT token and signs it using its private key, which can be verified by the Keycloak server using the Client's public key or certificate. [151] The public key or certificate can be either directly uploaded on the server (hardcoding it in the server configuration) or configuring a *JWKS URL* from where the Keycloak server can download the Client's public key: the latter method allows the Client to regularly change its keys, since the server will always download the valid ones through the provided URL;
- *Signed JWT with Client Secret*: similar to the previous method, a signed JWT token is generated by the Client and verified by the server. However, despite using a private key and certificate, the token will be signed using the Client's secret, which needs to be known from both parties.

Multiple Clients can be defined within a single Realm, allowing to setup different interfaces for different applications and services.
By default, every newly created Realm defines some default Clients (*account*, *admin-cli*, *broker*, *realm-management*, *security-admin-console*), generally used to connect with internal components.

4.3.2 Authorization in Keycloak

Regarding the **Authorization** scheme in Keycloak, there are 2 viable ways:

- **Realm level**: it's possible to define some authorization rules by defining *Realm Roles* and related attributes, that can be later assigned to the users and granting them permissions regardless of the Client (API) used;
- **Client level**: it's possible to define some fine-grained access control rules (*Permissions*) over certain *Resources*, characterized by different *Authorization Scopes* and linked by defined *Policies*. These rules are client-dependent and can provide some additional control with different API usage.

Additionally, some *Client Scopes* can be defined, allowing to specify some *Mappers* (to be newly created or imported from the built-in set) and **mapping permissions from specific scopes** (*Realm Roles*, *Client Roles*, or a combination of the two - called *Composite Roles*).
This is crucial to **access some resources from a certain client**, e.g. a client without the Mappers for {'realm roles', 'client roles'} is unable to access the set of roles assigned to the user, hence denying them access even to resources they would be allowed to.

Building up the Authorization scheme for LEXIS, the first step has been to design an **RBAC matrix**, defining roles to be created and permissions to be granted to each role. Role-based Access Control (RBAC) represents the most commonly deployed Access Control model, allowing to manage users' access based on the their job responsibilities, hence identified by the assigned Roles.

However, an **hybrid solution using both RBAC and ABAC approaches** would introduce a finer-grained management of access rights (cf. *IDM365 RBAC/ABAC Hybrid Solution* [77]), based on who the user is (i.e. Attributes like user's location, company or project they work in) rather than what they do (i.e. Roles).

The following subsections will illustrate how these aspects of the Access Management system have been designed for LEXIS.

Roles: RBAC Matrix

Designing the appropriate access control for LEXIS Project requires to identify the correct mapping between job functions and the corresponding authority level.
To this extent, the following **Roles** have been identified [152]:

- *LEXIS Administrator* (**lex_adm**): The LEXIS cloud admins have higher level (pre-allocated resource pools for LEXIS) control over LEXIS-only infrastructures. Full control is under responsibility of local production teams on both clouds locations. YORC infrastructure components will be created out of the LEXIS portal. Scope: LEXIS Administration.
- *LEXIS Support* (**lex_sup**): The LEXIS platform may require some "support" persons that can access some information about LEXIS Customer/Client without having full access to the LEXIS platform.Can be restricted to some Organization only. Scope: LEXIS Support.
- *LEXIS Organisation Manager* (**org_mgr**): From LEXIS Portal perspective, it's crucial to identify some Organization (can be a Company for instance) hierarchy in order to be able to handle the fact that an Organization can have several projects.Restricted to the Organization by default. Scope: LEXIS Customer/-Client.

- *LEXIS Financial Manager* (**fin_mgr**): Financial Manager has control over all financial (billing & payment) informations.Restricted to the Organization by default. It can be restricted to some projects only. Scope: LEXIS Customer/Client.
- *LEXIS License Manager* (**lic_mgr**): Licensing Manager has control over all licensing informations.Restricted to the Organization by default. It can be restricted to some projects only. Scope: LEXIS Customer/Client.
- *LEXIS Project Manager* (**prj_mgr**): Project manager has control over all projects within a LEXIS client organization.Restricted to the Organization by default. It can be restricted to some projects only. Scope: LEXIS Customer/Client.
- *LEXIS Workflow Manager* (**wfl_mgr**): Workflow manager has control other all workflows, can create, delete and execute them within an organization. The Workflow manager can also assign resources to the workflow like computational hours from a specific Project of an Organization.Restricted to the project by default, it can be restricted to some workflow only. Scope: LEXIS Customer/Client.
- *LEXIS IAM Manager* (**iam_mgr**): Users manager has control over one or more group of users within a LEXIS client organization.Restricted to the Organization by default. It can be restricted to some projects only. Scope: LEXIS Customer/Client.
- *LEXIS User* (**end_usr**): End users have control over their own jobs and data only. Can only know about group-level information for groups they are members of.Restricted to the Organization by default. It can be restricted to some projects only. Scope: LEXIS Customer/Client.

Roles have been assigned different types of permissions, according to possible restrictions related to their attributes:

- **F**: Full Access;
- **P**: Partial Access, restricted by attributes based on the Organization (**PO**), Project (**PP**) or Workflow (**PW**) to which they belong.

The full RBAC Matrix is represented in Figure 4.1. [152]

Attributes: ABAC Approach in Keycloak

Building a hybrid access control requires to integrate an adequate Attribute-based Access Control (ABAC) approach on top of the above-mentioned *RBAC scheme*.

Attributes represent some kind of metadata specifying additional information over an object, element or file. In Keycloak, it's possible to **assign Attributes to Users, Groups or Roles**.

Although it's possible and might be useful to assign *Attributes to specific Roles*, it would bring additional complexity to the RBAC model, as it would require to grant the Client with access to all the Realm roles and filtering for the wanted role and/or specific attribute might be tricky, potentially introducing performance issues to the access control management.

The other viable methods are assigning Attributes for Users or Groups. However, a member of a group inherits all the attributes defined for that group, in addition to the attributes defined for the user itself.

LEXIS PERMISSIONS	List Users	Read details of a User	Create/Delete/Update a User	List Organizations	Read details of an Organization	Create/Delete/Update an Organization	List Billing&Payment informations	Read details of a Billing&Payment information	Create/Delete/Update a Billing&Payment information	List Licensing informations	Read details of a Licensing information	Create/Delete/Update a Licensing information
	iam_list	iam_read	iam_write	org_list	org_read	org_write	bil_list	bil_read	bil_write	lic_list	lic_read	lic_write
	Identity & Access Management			Organization Management			Billing Management			Licensing Management		
lex_adm	F	F	F	F	F	F	F	F	F	F	F	F
lex_sup	P (PO)	P (PO)		P (PO)	P (PO)		P (PO)			P (PO)		
org_mgr	P (PO)			P (PO)	P (PO)	P (PO)	P (PO)			P (PO)		
fin_mgr				P (PO)			P (PO)	P (PO)	P (PO)			
lic_mgr				P (PO)						P (PO)	P (PO)	P (PO)
prj_mgr	P (PO, PP)			P (PO, PP)								
wfl_mgr	P (PO, PW)			P (PO, PW)								
iam_mgr	P (PO)	P (PO)	P (PO)	P (PO)								
end_usr	P (PO, PP, PW)			P (PO, PP, PW)								

LEXIS PERMISSIONS	List Projects	Read details of a Project	Create/Delete/Update a Project	List Workflows	Read details of a Workflow	Create/Delete/Update/Start/Stop a Workflow	List Computations	Read details of a Computation	Create/Delete/Update/Start/Stop a Computation	List Datasets	Read details of a Dataset	Create/Delete/Update/Import/Export a Dataset
	prj_list	prj_read	prj_write	wfl_list	wfl_read	wfl_write	cpu_list	cpu_read	cpu_write	dat_list	dat_read	dat_write
	Project Management			Workflow Management			Computation Management (jobs, tasks of differents systems OpenStack/YORC/HEAppE)			Data Management (iRODS DDI and WCDA)		
lex_adm	F	F	F	F	F	F	F		F	F		F
lex_sup	P (PO)	P (PO)		P (PO)	P (PO)		P (PP)			P (PP)		
org_mgr	P (PO)											
fin_mgr	P (PO)											
lic_mgr	P (PO)											
prj_mgr	P (PO, PP)	P (PO, PP)	P (PO, PP)	P (PO, PP)			P (PP)			P (PP)		
wfl_mgr	P (PO, PW)			P (PO, PW)	P (PO, PW)	P (PO, PW)	P (PP)			P (PP)		
iam_mgr	P (PO)											
end_usr	P (PO, PP, PW)	P (PO, PP, PW)		P (PO, PP, PW)	P (PO, PP, PW)		P (PP)	P (PP)	P (PP)	P (PP)	P (PP)	P (PP)

Figure 4.1. RBAC Matrix for LEXIS.

Hence, the most logical approach results in **assigning Attributes to the Groups**, as it allows to define common attributes for all the members of the group instead of directly appointing them to each user.
In this way, grouping the same type of users becomes very useful: for example, it's possible to create a Group for each Project, so that different users assigned with the same role (e.g. "Admin") can be distinguished based on the Project (Group) they are working in.

Attributes assigned to Users and Groups can be displayed in the access token by defining a **User Attribute Mapper** for a given *Client* or *Client Scope*, that can be later associated to multiple Clients.

In Keycloak, it's possible to define multiple values for the same attribute, simply concatenating them with the delimiter "##". To display all the values, the *Multivalued* option needs to be enabled in the *Mapper* settings, otherwise only the first one will be displayed.
User and Group Attributes can also overlap, assigning additional values to one another. In order to display any appointed value to the given attribute, despite being a User or Group attribute, the *Aggregated attribute values* option needs to be enabled in the *Mapper* settings. Duplicated values will be discarded, obtaining a unique set of all the effective attributes assigned to the user. Using OpenID Connect mappers, the *Multivalued* option needs to be enabled as well.

4.3.3 JWT Tokens

The Keycloak IAM makes use of *Identity and Access tokens* in the **JSON Web Token (JWT) format** [153]: these tokens have proven to be very versatile, with a compact and url-safe design, particularly useful in the SSO context.

JWTs are signed tokens, structured in 3 parts:

- a *Header*, containing useful information such as the signature algorithm used and the type of token. The server could be using different keys for different algorithms, labelling them with a `kid` flag (cf. subsection 4.3.4);
- a *Payload*, containing a set of *Claims* - i.e. standard and custom fields - with information related to the token itself, such as details of the user requesting the token and the system issuing it;
- a *Signature*, used to verify the integrity of the token - in case of using an asymmetric signature method, it can be validated with the public key of the issuer.

These sections are encoded through the *Base64Url Encoding* algorithm and then concatenated, separated by dots:

$$token = base64urlEnc(header) \oplus '.' \oplus base64urlEnc(payload) \oplus '.' \oplus base64urlEnc(signature)$$

the signature being:

$$signature = H(base64urlEnc(header) \oplus '.' \oplus base64urlEnc(payload), secret)$$

Through various Authentication mechanisms (subsection 4.3.1), the Client will retrieve different type of tokens:

- **Identity token**: asserting the user's identity information;

- **Access token**: stating their access rights to connect with other secured resources on the network. Once authenticated, the Client will be able to use this token to make authenticated calls to secure APIs;
- **Refresh token**: allowing an authenticated Client to request a renewed access token when the previous one expires: once the user is authenticated, the following re-authentications will be transparent to them, as long as the Refresh token is valid. However, not all the OIDC Flows will issue this type of token (cf. subsection 4.3.1).

Once the user is authenticated and retrieved the access token, every request for accessing secured services or resources will include this token, granting them access according to their permissions: this is typically achieved by including the token in the Authorization header, following the *HTTP Bearer authentication scheme*. [154]

4.3.4 Token Forgery

I've been analysing these tokens, especially in relation to the service APIs (*Clients*) available in Keycloak and their Authentication and Authorization mechanisms: regardless of the client (API) used to retrieve the token, once the user's credentials are verified, **a valid token can be used by any client** defined in the same *Realm* the user belongs to. It is also possible to retrieve an Access token referred to a specific client instead of a user, through the *Client Credentials Grant* OIDC flow (cf. subsection 4.3.1).

It's possible to **retrieve tokens** in different ways:

- directly from the *cookies on a web session* (after authenticating on Keycloak);
- using the '*cUrl*' command from the command-line interface or using other libraries (e.g. *gocloak library* [155] in Go language);
- using an API development platform and software, such as '*Postman*'.

For the performed tests, **Postman** has been chosen as the token retrieving method to use for its simplicity and provided functionalities. To see the software configuration used, please refer to Appendix B.
The token content can be easily decoded using any compatible libraries (e.g. *PyJWT library* for Python) or browsing to `https://jwt.io/`.

As a simple example, I've been testing access tokens by retrieving a new token from the system and performing a request to get user information or display all the users in the realm, with different combinations of user's and client's permissions.
The following code snippet (in GO language) retrieves an access token for `username` (upon successful authentication through `clientA`) and uses it to get user information such as their name, ID number and e-mail, through `clientB` (in the same realm):

```
1 clientA := gocloak.NewClient(url)
2 userToken, err := clientA.Login(client1, secret1, realm, username,
     password)
3 if err != nil {
4   panic("Something wrong with userlogin:" + err.Error())
5 }
6 fmt.Printf("Token from service A (%s): %v\n", client1, userToken.
     AccessToken)
```

```
fmt.Printf("\n Connecting to clientB (%s):\n", client2)
clientB := gocloak.NewClient(url)
_, err = clientB.LoginClient(client2, secret2, realm)
if err != nil {
  panic("Something wrong with clientlogin:" + err.Error())
}
user, err := clientB.GetUserInfo(userToken.AccessToken, realm)
if err != nil {
  panic("Something wrong with checking usertoken:" + err.Error())
}
fmt.Printf("User Name: %v\nUser ID: %v\nUser Email: %v\n", user.
    PreferredUsername, user.Sub, user.Email)
```

It's important to notice how the outcome of this simple operation is largely dependent on the Client Roles, Client Scopes and Mappers assigned to the given Client, as well as the choice of the Client itself: **an unauthorized Client**, despite using a valid token granting access to the related user, **will be denied from accessing** the requested resources.

The token analysis moved to evaluate the eventual reuse of tokens in term of security and try to **evaluate the system resistance against forged tokens**.

I made some research on **known vulnerabilities and misconfiguration in JWT libraries** [156][157]. I've been trying to perform one of the most common exploits against JWT tokens: sometimes the server issuing JWTs can avoid specifying a default signature algorithm (set to '*none*') to act with greater flexibility and allow different types of tokens for different uses; on the other hand, not specifying the algorithm to be used for token verification leaves this choice to the algorithm used for the token signature, stated in the token header.
However, accessing this value before verifying the token represents a huge security issue, as the token needs to be validated first before being trustworthy, **allowing the attacker to dictate which algorithm to be used for the signature verification**.

In this way, an attacker can forge a token using an arbitrary algorithm, inducing the server to verify it with a different one: for example, the attacker can sign the token with a symmetric algorithm (e.g. HMAC-SHA256) using the server's asymmetric public key (e.g. RSA public key, which is exposed and well-known) as the secret, **driving the server - expecting an RSA-signed token - to verify it with its public key, used as a HMAC secret key**. By referring to the algorithm stated in the token's header, the server will be tricked into considering it valid.
This flaw would allow any attacker to sign a token with an arbitrary payload, that would be accepted by the server and allow them access to discretionary resources.

However, this type of attack **can be easily prevented by avoiding using the 'none' algorithm**, which is intended to be used for already validated tokens (interestingly, 'none' and 'HS256' are the only 2 mandatory algorithms to be implemented, according to the JWT standard), **and specifying the verification algorithm to be used at server level**, preventing the attacker to choose it arbitrarily.
Additionally, the JWT standard provides a ***kid* flag in the token header**, used to label the server keys and relate them to the corresponding algorithm. In this way, the server can't be tricked to verify the token with an algorithm different from the one supposed to be used with the given key.

Keycloak implements all the possible countermeasures to these attacks:

- injecting arbitrary payload claims in a valid token: the signature would prevent an attacker to retrieve or hijack a valid token and then change its payload with the Base64Url Encoding of an arbitrary one, as its integrity would be compromised and the token rejected;
- forging a new token: Keycloak includes the `kid` identifier in the tokens, making the above-mentioned attack unfeasible.

Chapter 5

Conclusion

This thesis aimed to **identify and build an appropriate Authentication and Authorization Infrastructure with Single Sign-On capabilities for the LEXIS project**. Based on a qualitative analysis of the security features provided by several open-source IAM solutions, the best fitting system has been selected as the Identity Management system to secure the LEXIS Infrastructure and all its components.
Unfortunately, the lack of time and the periodic deadlines imposed for a shared project, as LEXIS is, didn't allow to accomplish a comprehensive study of all the analysed solutions. However, the **motivating results** obtained by benchmarking the selected solution reflected how suited it is for the integration in such a complex system.

Designing an appropriate Authorization scheme is crucial to the purpose of the project, as it represents the basis on which the **interconnection of all the secured components involved** is founded.
Identifying the perfect Access Control scheme to be adopted represents a tangled challenge. Thus, **embracing an hybrid approach** merging different mechanisms is surely a bold choice, which however could represent the **best working combination** for meeting all the requirements defined by LEXIS.

The project is moving forward to a new stage, as the work done by different teams in the LEXIS project (e.g. Data model, Orchestrator, etc) is **converging to a common point**. This process will surely take some time, as the due date for the project is expected to be in June 2021.
Regarding the AAI sytem, the future work will surely include the **deployment of the defined Access Control scheme** in the system and the integration with the other components of the project.

Appendix A

Ansible deployment guide

Deployment of widely distributed systems like Keycloak for LEXIS Project can be automated via deployment tools such as **Ansible**. This guide refers to Ubuntu OS.

To install Ansible, run the following command:

```
sudo apt-get install ansible
```

The installation package also contains **Ansible-Galaxy**, usefui tool for installing community-shared *Ansible Roles*.

The deployment can be performed on either local or remote machines, specified in the **hosts** file, passed as an argument to the `ansible` command using the flag `-i`. In this file, hosts can be divided in groups and some additional information can be provided, such as ssh key location or other connection parameters.
An example of its format is available in the "*hosts*" file provided in the source files.

The system deployment is executed by running an **Ansible-Playbook** called "*site.yml*", provided in the source files. Playbooks group different types of operations, mainly: *tasks* and *pre-tasks*, grouped in *plays* and representing commands to be executed when running the playbook; *handlers*, representing operations to be performed when a `notify` command is used in tasks or pre-tasks; *templates*, representing files in '*.j2*' format, which can be easily copied in the destination machine using the command `template`.
Nevertheless, the file defines **some requirements on the machine running the playbook**:

- installation of Python:

  ```
  sudo apt-get install python
  ```

- installation of Java JRE:

  ```
  sudo apt-get install default-jre
  ```

- installation of 2 Ansible-Galaxy shared Roles (cf. section 4.2):

  - adfinis-sygroup.mariadb [148]
  - andrewrothstein.keycloak [149]

  Such Roles could be installed by running:

  ```
  ansible-galaxy install adfinis-sygroup.mariadb
  ansible-galaxy install andrewrothstein.keycloak
  ```

However, they have been slightly modified to adapt to the required setup and configuration; additionally, installing the roles as presented above would require the installation of `ansible` on each destination machine. For this reason, to deploy the server I preferred to clone the GitHub repositories (available at [158] and [159]), in accordance with GitHub repositories licensing, then modifying the roles as shown in section 4.2. The installed roles are provided under the "*roles*" folder;

- finally, the installation can be executed by running the following command (optional `-vvvvv` for verbose):

```
ansible-playbook site.yml -i hosts -u username --ask-become-pass
    [-vvvvv]
```

which executes the "*site.yml*" playbook over the hosts specified in the "*hosts*" file, runned by "*username*" as administrator (the path to the SSH key can be specified in the same file).

Once the script is successfully terminated, the server is ready and fully operational. The administration console can be reached at: `http://<server_ip_address>:<port_number>/auth/admin/`, generally listening on port 8080. Default username and password for the first admin user can be set in the file `roles/ansible-keycloak/tasks/main.yml`, using the `-u` and `-p` options.

Appendix B

Postman configuration and testing

Postman is a popular collaboration platform for API development and testing, allowing to organize test suites in Collections and subfolders and define different Environments and related variables. Collections and Environments can be easily exported and shared, e.g. within a testing team, or duplicated for different uses. Moreover, it provides high availability as it just requires the user to login in the application to access all their saved tests.

I have identified 3 main requests to perform:

- *Retrieving the token for a user*: it's an HTTP POST request of type `application/x-www-form-urlencoded`. Enter the token-endpoint in the request URL and fill in the *Body* tab as in Figure B.1; as shown in the figure, environment variables (e.g. *master-user*, *master-pswd*) can be used to set values.

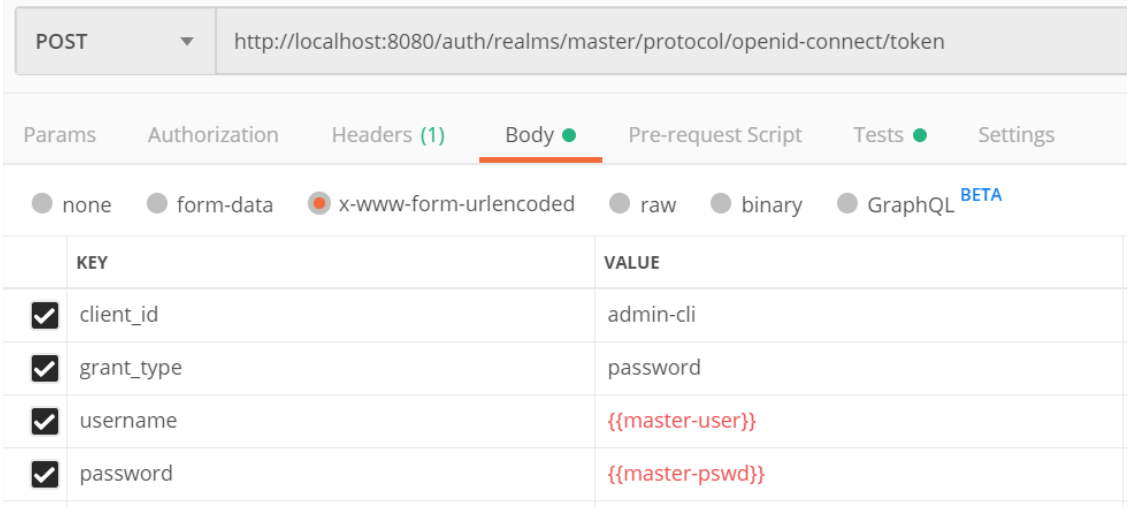

Figure B.1. Token request via Postman (user).

Additionally, a short JavaScript code snippet allows to capture the Access token retrieved from the request:

```
var jsonData = JSON.parse(responseBody);
postman.setEnvironmentVariable("master-token", jsonData.
    access_token);
```

- *Retrieving the token for a client*: it's similar to the previous case, but using a different `grant_type` value and associated OIDC Flow (cf. Figure B.2);

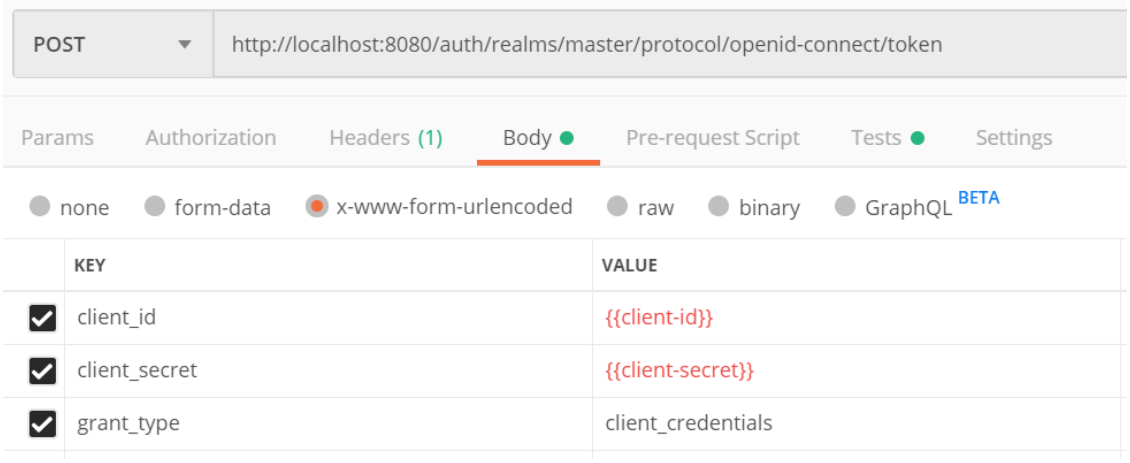

Figure B.2. Token request via Postman (client).

- *Using the token to access the server*: once retrieved a valid Access token with one of the previous requests, its value will be saved to the variable `master-token` by the above-mentioned JavaScript code. This value can then be used to access protected resources, according to the rights of the user or client linked to the issued token. For example, the request in Figure B.3 shows the request to perform to list all the users in the relarted realm (assuming both user and client have the right roles and permissions to access this values).

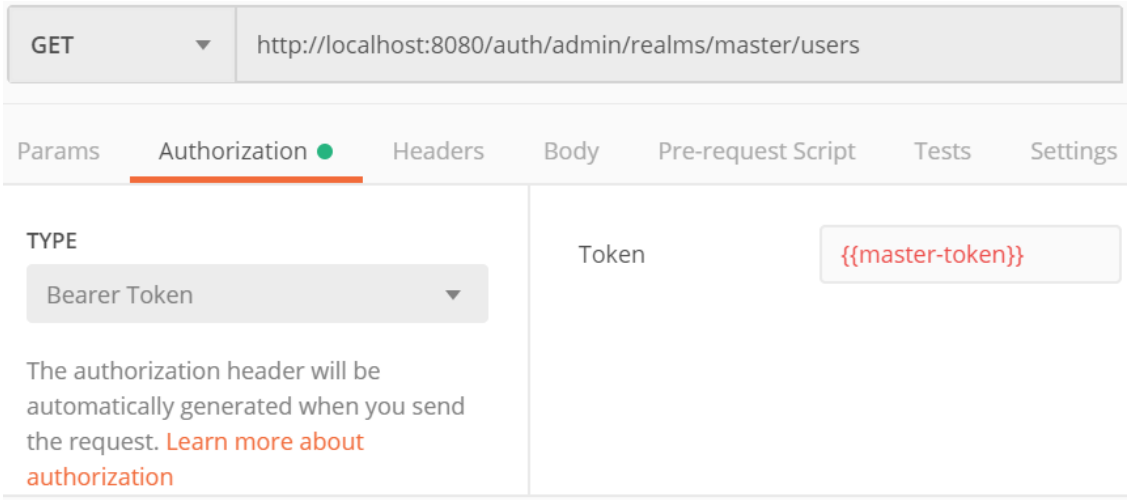

Figure B.3. Request for displaying users in the realm.

Bibliography

[1] LEXIS Project website, https://lexis-project.eu/web/

[2] List of Single Sign-On implementations - Wikipedia, https://en.wikipedia.org/wiki/List_of_single_sign-on_implementations

[3] LEXIS Deliverable D4.1, *Analysis of mechanisms for securing federated infrastructure.*

[4] Introduction to Keystone Federation, https://docs.openstack.org/keystone/latest/admin/federation/introduction.html

[5] HEAppE Middleware, https://code.it4i.cz/ADAS/HEAppE/Middleware/wikis/home

[6] iRODS website, https://irods.org

[7] iRODS Authentication, https://docs.irods.org/4.1.1/manual/authentication/

[8] Redhat's documentation on Key Concepts and Terms in the SSO environment, https://access.redhat.com/documentation/en-us/red_hat_single_sign-on/7.0/html/server_administration_guide/overview#core_concepts_and_terms

[9] Federated Identity - Wikipedia, https://en.wikipedia.org/wiki/Federated_identity

[10] National Institute of Standards and Technology, FIPS 200, *Minimum Security Requirements for Federal Information and Information Systems*, March 2006, https://csrc.nist.gov/publications/detail/fips/200/final.

[11] J. Katz and et al., *Handbook of applied cryptography*, CRC press, 1996.

[12] A. Singhal, T. Winograd and K. Scarfone, *Guide to Secure Web Services*, August 2007, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-95.pdf.

[13] SAML specification, http://saml.xml.org/saml-specifications/

[14] SAML - Cover Pages by OASIS, http://xml.coverpages.org/saml.html

[15] OpenID Connect specification, https://openid.net/connect/

[16] E. E. Hammer-Lahav, "The OAuth 1.0 Protocol." RFC-5849, April 2010, DOI 10.17487/RFC5849

[17] Session Fixation Attack against OAuth 1.0 Request Token approval flow, https://oauth.net/advisories/2009-1/

[18] OAuth 1.0 Revision A, https://oauth.net/core/1.0a/

[19] E. D. Hardt, "The OAuth 2.0 Authorization Framework." RFC-6749, October 2012, DOI 10.17487/RFC6749

[20] Introducing OAuth 2.0, https://hueniverse.com/introducing-oauth-2-0-b5681da60ce2#04c9

[21] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The Kerberos Network Authentication Service (V5)." RFC-4120, July 2005, DOI 10.17487/RFC4120

[22] E. J. Sermersheim, "Lightweight Directory Access Protocol (LDAP): The Protocol." RFC-4511, June 2006, DOI 10.17487/RFC4511
[23] Active Directory Technical Specification, https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/d2435927-0999-4c62-8c6d-13ba31a52e1a/
[24] Keycloak Auditing and Events, https://www.keycloak.org/docs/latest/server_admin/index.html#auditing-and-events
[25] JOSSO 2.4 Auditing, http://docs.atricore.com/josso2/2.4/tutorials/josso-auditing-tutorial/html/en-US/JOSSO_Tutorial_Auditing.html
[26] Spring Security framework website, https://spring.io/projects/spring-security
[27] Advanced Hybrid RBAC, https://wiki.evolveum.com/display/midPoint/Advanced+Hybrid+RBAC
[28] Keycloak website, https://www.keycloak.org/
[29] Keycloak - Server Admin documentation, https://www.keycloak.org/docs/latest/server_admin
[30] Keycloak Authorization Services, https://www.keycloak.org/docs/latest/authorization_services/
[31] StackHPC's article on Federation and identity brokering using Keycloak, https://www.stackhpc.com/federation-and-identity-brokering-using-keycloak.html
[32] OpenStack Keystone website, https://docs.openstack.org/keystone/latest/
[33] Openstack Keystone documentation, https://www.openstack.org/software/releases/ocata/components/keystone
[34] DMTF CADF specification, https://www.dmtf.org/standards/cadf
[35] Auditing with CADF, https://docs.openstack.org/mitaka/config-reference/identity/auditing.html
[36] Unity website, https://www.unity-idm.eu/
[37] Apache Syncope website, https://syncope.apache.org/
[38] Apache Licenses, https://www.apache.org/licenses/
[39] ConnId website, http://connid.tirasa.net/
[40] Apache Syncope Audit documentation, https://syncope.apache.org/docs/2.1/reference-guide.html#audit
[41] Apache Syncope Admin console, https://syncope.apache.org/docs/2.1/reference-guide.html#admin-console
[42] Apache Syncope End-user application, https://syncope.apache.org/docs/2.1/reference-guide.html#enduser-application
[43] WSO2 website, https://wso2.com/identity-and-access-management/
[44] WSO2 Licenses, https://wso2.com/licenses
[45] WSO2 Installation Prerequisites, https://docs.wso2.com/display/IS580/Installation+Prerequisites
[46] WSO2 Deployment Patterns, https://docs.wso2.com/display/IS580/Deployment+Patterns
[47] Web Services Federation - OASIS standard, http://docs.oasis-open.org/wsfed/federation/v1.2/ws-federation.html
[48] WSO2 - Calling Admin Services, https://docs.wso2.com/display/IS580/Calling+Admin+Services
[49] WSO2 Self-Service Registration, https://wso2.com/whitepapers/customer-identity-and-access-management-a-wso2-reference-architecture/#2111
[50] OpenIAM website, https://www.openiam.com/

[51] OpenIAM Documentation, http://doc.openiam.com/
[52] OpenIAM Identity and Access Governance - Licenses, https://www.openhub.net/p/openiam-idm-ce
[53] 1.2 System requirements - OpenIAM Documentation, http://docs.openiam.com/installation/about.htm?tocpath=Installation%20Guide%7C1.%20About%20installing%20OpenIAM%7C_____0#1._About_installing_OpenIAM
[54] Web Services Security - OASIS standard, https://www.oasis-open.org/committees/wss
[55] OpenIAM - Features, https://www.openiam.com/products/identity-governance/features/
[56] OpenIAM Identity and Access Manager Technical Architecture Overview, https://www.openiam.com/wp-content/uploads/TechnicalArchitecture-v3-A.pdf
[57] OpenIAM - Administration Guide, http://docs.openiam.com/administration/index.htm
[58] OpenIAM - User API, https://www.openiam.com/products/identity-governance/features/api/
[59] OpenIAM - Self-Service Guide, http://docs.openiam.com/self-service/index.htm
[60] Gluu website, https://gluu.org/docs
[61] Gluu License, https://gluu.org/docs/ce/4.0/#license
[62] Gluu - Clustering for HA, https://gluu.org/docs/ce/4.0/installation-guide/cluster/
[63] Gluu Server Backup, https://gluu.org/docs/ce/4.0/operation/backup/
[64] Gluu Server: Shibboleth for SAML & oxAuth for OAuth 2.0, https://www.gluu.org/shibboleth-idp/
[65] Gluu Authentication methods, https://gluu.org/docs/ce/4.0/authn-guide/intro/
[66] Gluu User Managed Access (UMA) 2.0 Authorization Server, https://gluu.org/docs/ce/4.0/admin-guide/uma/
[67] Gluu - Audit Logging Configuration, https://wiki.shibboleth.net/confluence/display/IDP30/AuditLoggingConfiguration
[68] Gluu Admin Guide - Accessing the UI, https://gluu.org/docs/ce/4.0/admin-guide/oxtrust-ui/#accessing-the-ui
[69] Gluu - User Registration, https://gluu.org/docs/ce/user-management/user-registration/
[70] Shibboleth for SAML & oxAuth for OAuth 2.0, https://www.gluu.org/shibboleth-idp/
[71] Evolveum MidPoint website, https://evolveum.com/midpoint/
[72] midPoint Licensing, https://wiki.evolveum.com/display/midPoint/Licensing
[73] Evolveum midPoint - High Availability and Load Balancing, https://wiki.evolveum.com/display/midPoint/High+Availability+and+Load+Balancing
[74] Role Explosion on midPoint guide, https://wiki.evolveum.com/display/midPoint/Role+Explosion
[75] Evolveum midPoint - Admin and User Interface, https://wiki.evolveum.com/display/midPoint/User+Interface
[76] Evolveum midPoint - Self-Registration Configuration, https://wiki.evolveum.com/display/midPoint/Self+Registration+Configuration
[77] IDM365 RBAC/ABAC Hybrid Solution, https://idm365.com/idm365-the-rbac-abac-hybrid-solution/
[78] EduGAIN website, https://edugain.org/

[79] eduGAIN License, https://github.com/biancini/edugain-connectivity-check/blob/master/LICENSE
[80] eduGAIN Access Check, https://wiki.geant.org/display/eduGAIN/eduGAIN+Access+Check
[81] TNC-2018 INTERNET2 GEANT Campus IdP v1.0, https://docs.google.com/presentation/d/1y8NySPa1mQPEFKdCYodWMUv-a9zeK9dyZ6_VsZ6C_7o/edit?usp=sharing
[82] Soffid website, https://www.soffid.com/
[83] Soffid License, http://www.soffid.com/doc/console/iam-core/license.html
[84] Soffid - Multi-master MariaDB cluster, http://confluence.soffid.org/display/SOF/Creating+a+multi-master+MariaDB+cluster
[85] Soffid - System backup, http://confluence.soffid.org/display/SOF/System+backup
[86] Bridging OpenID and SAML 2.0, https://www.terena.org/activities/eurocamp/november07/slides/solberg-open-id.pdf
[87] Soffid - IAM: Audit & Compliance, http://www.soffid.com/our-solutions/#identity-governance-audit
[88] Soffid - Web services reference, http://confluence.soffid.org/display/SOF/Web+services+reference
[89] JOSSO website, http://www.josso.org/
[90] JOSSO2 License, https://github.com/atricore/josso2/blob/2.4.3/LICENSE
[91] About High Availability, http://docs.atricore.com/josso2/2.4.0/josso-reference-guide/html/en-US/JOSSO_Reference.html#About_High_Availability
[92] JOSSO Admin API - Java API for RESTful Services (JAX-RS), http://docs.atricore.com/josso2/2.4/tutorials/josso-jaxrs-tutorial/html/en-US/JOSSO_Tutorial_JAXRS.html
[93] JOSSO - JAX-RS Sample Client code, http://docs.atricore.com/josso2/2.4/tutorials/josso-jaxrs-tutorial/html/en-US/JOSSO_Tutorial_JAXRS.html#_sample_client_code
[94] Java Enterprise Edition (Java EE/J2EE), https://www.oracle.com/java/technologies/java-ee-glance.html
[95] JOSSO Architecture, http://www.josso.org/architecture.html
[96] JOSSO - Atricore IAM Platform, http://www.atricore.com
[97] Shibboleth website, https://www.shibboleth.net/
[98] Apache License Version 2.0, http://apache.org/licenses/LICENSE-2.0.html
[99] Shibboleth - System Requirements, https://wiki.shibboleth.net/confluence/display/IDP30/SystemRequirements
[100] Shibboleth - Clustering, https://wiki.shibboleth.net/confluence/display/IDP30/Clustering
[101] CAS Protocol 2.0 specification, https://apereo.github.io/cas/6.0.x/protocol/CAS-Protocol-V2-Specification.html
[102] Shibboleth - Interesting Features, https://wiki.shibboleth.net/confluence/display/IDP30/InterestingFeatures
[103] Shibboleth - Audit Logging Configuration, https://wiki.shibboleth.net/confluence/display/IDP30/AuditLoggingConfiguration
[104] Shibboleth - Administrative Configuration, https://wiki.shibboleth.net/confluence/display/IDP30/AdministrativeConfiguration
[105] Shibboleth for SAML & oxAuth for OAuth 2.0, https://www.gluu.org/shibboleth-idp/
[106] Apereo CAS website, https://www.apereo.org/projects/cas

[107] Apereo CAS License, https://apereo.github.io/cas/4.2.x/protocol/CAS-Protocol-Specification.html#appendix-e-cas-license
[108] Apereo CAS - Installation Requirements, https://apereo.github.io/cas/4.2.x/planning/Installation-Requirements.html
[109] Apereo CAS - High Availability Guide, https://apereo.github.io/cas/4.2.x/planning/High-Availability-Guide.html#high-availability-guide-haclustering
[110] Apereo CAS - Multiple Server Nodes, https://apereo.github.io/cas/4.2.x/planning/High-Availability-Guide.html#multiple-cas-server-nodes
[111] Apereo CAS - SSO for Shibboleth IdP, https://apereo.github.io/cas/4.2.x/integration/Shibboleth.html
[112] Apereo CAS - Authentication Handlers, https://apereo.github.io/cas/4.2.x/installation/Configuring-Authentication-Components.html#authentication-handlers
[113] Apereo CAS - Securing Access and Authorization, https://apereo.github.io/cas/4.2.x/installation/Installing-ServicesMgmt-Webapp.html#securing-access-and-authorization
[114] Apereo CAS - Audit, https://apereo.github.io/cas/4.2.x/installation/Audits.html
[115] Perun AAI website, https://perun-aai.org
[116] Perun License, https://github.com/CESNET/perun#license
[117] Perun RPC API, https://perun-aai.org/documentation/technical-documentation/rpc-api/index.html
[118] Perun - User management & Self-registration, https://perun-aai.org/documentation/technical-documentation/rpc-api/index.html
[119] Perun GitHub repository, https://perun-aai.org/documentation/technical-documentation/rpc-api/index.html
[120] FreeIPA website, https://www.freeipa.org/page/Main_Page
[121] FreeIPA - License, https://www.freeipa.org/page/License
[122] FreeIPA - Scalability, https://www.freeipa.org/page/Goals/Scalability
[123] FreeIPA - Backup and Restore, https://www.freeipa.org/page/Backup_and_Restore
[124] FreeIPA - Web App Authentication, https://www.freeipa.org/page/Web_App_Authentication
[125] FreeIPA - Audit session recording details, https://www.freeipa.org/page/Session_Recording#Audit_recording_details
[126] FreeIPA - Self-Service Password Reset, https://www.freeipa.org/page/Self-Service_Password_Reset
[127] OpenAM website, https://www.openidentityplatform.org/
[128] OpenAM License, https://github.com/OpenIdentityPlatform/OpenAM/blob/master/LICENSE.md
[129] OpenAM Features - Wikipedia, https://en.wikipedia.org/wiki/OpenAM#Features
[130] OpenAM - Authentication modules, https://github.com/OpenIdentityPlatform/OpenAM/wiki/Authentication-modules
[131] OpenAM - Audit Logging, https://backstage.forgerock.com/docs/openam/13.5/reference/#chap-audit-log-messages
[132] OpenAM - Legacy User Self-Service, https://backstage.forgerock.com/docs/openam/13.5/reference/#legacy-user-self-service
[133] Univention Corporate Server website, https://github.com/univention/univention-corporate-server

[134] Univention Corporate Server License, https://github.com/univention/univention-corporate-server/blob/4.4-1/LICENSE
[135] UCS Fault-tolerant domain setup, https://docs.software-univention.de/manual-4.4.html#domain:fault-tolerant
[136] UCS Domain controller backup, https://docs.software-univention.de/manual-4.4.html#domain-ldap:Domain_controller_backup
[137] Univention Management Console (UMC) Documentation, https://docs.software-univention.de/manual-4.4.html#central:user-interface
[138] Univention Corporate Server documentation - Clients, https://docs.software-univention.de/quickstart-en.html#quickstart:clients
[139] UMC Architecture, https://docs.software-univention.de/developer-reference-4.4.html#umc:architecture
[140] UCS Self Services, https://www.univention.com/blog-en/2019/04/ucs-4-4-self-services-new-features/
[141] Release of UCS 4.1 with Docker, Single Sign-On Mechanism and Two-Factor Authentication, https://www.univention.com/blog-en/2015/11/release-of-ucs-4-1-with-docker-single-sign-on-mechanism-and-two-factor-authenti
[142] Aerobase IAM website, https://aerobase.io/iam
[143] Aerobase IAM License, https://github.com/aerobase/unifiedpush-server/blob/master/LICENSE.txt
[144] Aerobase IAM - Clustered Mode, https://aerobase.io/docs/installation/index.html#_standalone-ha-mode
[145] A. I. A. R. API, https://aerobase.io/docs/server_development/index.html#admin-rest-api
[146] Aerobase IAM - Server Features, https://aerobase.io/docs/server_admin/index.html#features
[147] R. J. García-Hernández and M. Golasowski, "Supporting keycloak in irods systems with openidauthentication", Presented at the CS3 2020 - Workshop on Cloud Storage Synchronization and Sharing Services, Copenhagen (Denmark), Jan 27-29, 2020, pp. 1–6
[148] Mariadb by adfinis-sygroup - Ansible Galaxy, https://galaxy.ansible.com/adfinis-sygroup/mariadb
[149] Keycloak by andrewrothstein - Ansible Galaxy, https://galaxy.ansible.com/andrewrothstein/keycloak
[150] OpenID Connect Authentication Flows, https://www.keycloak.org/docs/latest/server_admin/#_oidc-auth-flows
[151] M. Jones, B. Campbell, and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants." RFC-7523, May 2015, DOI 10.17487/RFC7523
[152] LEXIS RBAC Matrix v3.2, January 2020
[153] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)." RFC-7519, May 2015, DOI 10.17487/RFC7519
[154] M. Jones and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage." RFC-6750, October 2012, DOI 10.17487/RFC6750
[155] Gocloak library in Go language, https://github.com/Nerzal/gocloak
[156] Auth0 - Critical vulnerabilities in JSON Web Token libraries, https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/
[157] WebSecurify - Hacking JSON Web Tokens, https://blog.websecurify.com/2017/02/hacking-json-web-tokens.html
[158] Mariadb by adfinis-sygroup - GitHub, https://github.com/adfinis-sygroup/ansible-role-mariadb

[159] Keycloak by andrewrothstein - GitHub, https://github.com/andrewrothstein/ansible-keycloak