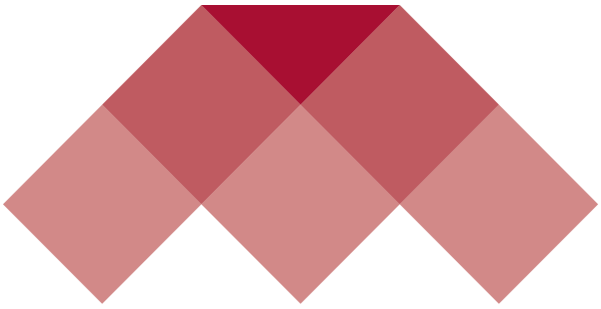
TÉCNICO LISBOA



FenixEdu Connect

An Identity Management System for Academic Organizations

Paulo Ricardo Conde Branco

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. João Nuno de Oliveira e Silva
Eng. David Jorge Lopes Batista Martinho

Examination Committee

Chairperson: Prof. António Manuel Raminhos Cordeiro Grilo
Supervisor: Prof. João Nuno de Oliveira e Silva
Member of the Committee: Prof. Fernando Henrique Côrte-Real Mira da Silva

November 2018

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

First and foremost, I would like to thank my adviser, Prof. Doutor João Nuno de Oliveira Silva for trusting me with the development of this subject. It is highly uncommon for a student to be given the creative freedom to pursue his own vision for a solution with this level of complexity. Thank you for allowing me to do just that, while still going out of your way to finding a solution whenever a problem arose.

To everyone at IST's IT Services team, with a special note for Ricardo Barata, Sérgio Silva and Luís Cruz for welcoming me and allowing me to grow within one of the best software development teams the country has ever known. Without your insights and countless hours of dedication to my personal growth none of this would be possible.

I would like to thank my parents, Teresa and Paulo for the constant words of encouragement to follow my true passion. I will be forever grateful for your unbelievable amount of support during these past five years, without which this endeavour would be impossible to carry out.

To the old life-long friends, João, Tiago and Rui for forcing me to go out for a drink (or two), even when it all seemed lost.

To the new life-long friends, José, Duarte and Frederico, with whom I had the pleasure to share the long nights, the deadlines, and all the small precious momments that made our journey at IST unique.

I would also like to thank Ana Varanda for, perhaps unknowingly, being the role model that brought me to this point. Thank you for showing me the taste for academic excellence and the slight perfectionist behaviour that comes with it. Your radical honesty, along with your unique way of rationalising the world that surrounds us were paramount for allowing me to solve some of the most significant challenges during these last months.

Finally, but arguably the most important, I would like to thank my co-advisor Eng. David Jorge Lopes Batista Martinho for reasons that words alone will never be able to express. Thank you for your contagious cheerful personality. I have yet to know someone with the same ability to lighten up a room, and honestly doubt I ever will. Thank you for teaching me that occasional failure is part of the process. Thank you for believing in me, when I did not. For always finding a solution when it seemed there was none. For striving for excellence in everything you did, and teaching everyone around you to follow suit. For taking the time to teach me everything I needed to know. Without your constant support this thesis would simply not be possible.

Thank you

Lisboa, October 2018

Paulo Ricardo Conde Branco

Resumo

O aumento das consequências de falhas de segurança tem pressionado organizações por todo o mundo a desenvolver medidas de proteção adicionais contra este tipo de ameaça. Os sistemas de gestão de acessos e identidades (GAI) estão na linha da frente desta proteção ao disponibilizar um conjunto de regras e processos para gerir as identidades digitais dos seus utilizadores. No entanto, as necessidades específicas das instituições académicas têm prevenido a adoção das mais recentes tecnologias e processos. Esta tese propõe uma solução de GAI com o objetivo de solucionar os desafios de autenticação, autorização e gestão de identidades das instituições de ensino superior com base na definição de uma *framework* que permite a cada instituição adaptar o produto aos seus requisitos. O modelo de implementação proposto é depois validado através de um caso de estudo da sua possível implementação no Instituto Superior Técnico, como substituição do sistema de gestão de identidades existente no mesmo.

Palavras-chave: Gestão de Acessos e Identidades, Autenticação, Autorização, *Json Web Token*, *OAuth*, *OpenID Connect*, Desenvolvimento de Software, Integração

Abstract

As security breaches result in ever increasing damages to organizations worldwide the pressure is on to develop additional safeguards against these types of cyberattacks. Identity and Access Management systems sit at the forefront of this protection by providing system administrators with a consistent set of rules and processes for managing the digital identities of their users. However, the specific needs of academic institutions have for long prevented the adoption of the latest state of art technologies and practices at these organizations. This thesis proposes an open-source IAM solution designed to meet the authentication, authorization and identity management challenges of higher education institutions by describing a framework that allows each client institution to tailor the product to its specific requirements. The proposed implementation model is then validated against a possible deployment at Instituto Superior Técnico, as a replacement for the existent identity management product.

Keywords: Identity and Access Management, Authentication, Authorization, Json Web Token, OAuth, OpenID Connect, Software Development, Integration

Contents

List of Tables

List of Figures

Glossary

2FA 2-Factor Authentication.

ACL Access Control Lists.

ACS Assertion Consumer Service.

API Application Programming Interface.

ATM Automatic Teller Machine.

CA Certificate Authority.

CAS Central Authentication Service.

CRL Certificate Revocation List.

CRM Customer Relationship Management.

CRUD Create-Read-Update-Delete.

CSRF Cross Site Request Forgery.

ERP Enterprise Resource Planning.

FdIM Federated Identity Management.

HEI Higher Education Institution.

HMAC Keyed-Hash Message Authenticator Code.

HRP Human Relations and Payroll.

HTTP Hypertext Transfer Protocol.

IAM Identity and Access Management.

IANA Internet Assigned Numbers Authority.

IDaaS Identity as a Service.

IdP Identity Provider.

IETF Internet Engineering Task Force.

IoC Inversion of Control.

IT Information Technology.

JAR Java Archive.

JPA Java Persistence API.

JSON Javascript Object Notation.

JWE JSON Web Encryption.

JWK JSON Web Key.

JWS JSON Web Signature.

JWT JSON Web Token.

LDAP Lightweight Directory Access Protocol.

MFA Multi-Factor Authentication.

MOOCs Massive Online Open Courses.

MVC Model View Controller.

OIDC OpenID Connect.

PDP Policy Decision Point.

PEP Policy Enforcement Point.

PIN Personal Identification Number.

PKI Public Key Infrastructure.

POJO Plain Old Java Object.

RBAC Role Based Access Control.

REST Representative State Transfer.

RFC Request For Comment.

RP Relying Party.

SAML Security Assertion Markup Language.

SLAs Service Level Agreements.

SOAP Simple Object Access Protocol.

SP Service Provider.

SSL Secure Sockets Layer.

SSO Single Sign On.

TLS Transport Layer Security.

TOTP Time-based One Time Password.

U2F Universal Second Factor.

UML Universal Modeling Language.

URL Universal Resource Locator.

XML Extensible Markup Language.

XSS Cross Site Scripting.

Chapter 1

Introduction

The past few years have seen a significant increase in the number of services available through digital channels. This is especially true in academic institutions where international regulation is pushing for a higher use of electronic documents with the goal of cutting costs and reducing waste. The ever-increasing number of information systems required to support this migration poses a challenge for the traditional identity management processes that are still in place in many organizations. While the current systems focus on solving the problem of authentication, often deploying Single Sign On (SSO) solutions to reduce the number of credentials that users have to memorize, little is done when it comes to managing the digital identities of each user.

Academic institutions are required to manage accesses for an extremely dynamic user base, often consisting of large thousands of status changes per yer as students start and complete their educational programmes. In addition, roles are often fluid: a student can simultaneously be a part-time teaching assistant or an employee at the university's Information Technology (IT) services. This poses a challenge for the traditional role-based access control technologies in place in these organizations.

Discrete Identity and Access Management (IAM) solutions play a pivotal role in modern organizations, both through the obvious security layer they entail as through the accountability and traceability they can offer. Organizations of all sizes can take advantage of advanced IAM solutions to improve productivity, reduce downtime and ensure compliance with data protection laws and/or internal business rules.

As organizations began to adopt the cloud-based service format the need for interoperable solutions for identity management in these environments paved the way for the first offerings of Cloud-based Identity and Access Management systems, also known as Identity as a Service (IDaaS). The significant cost savings associated with this architecture are commonly cited as one of the main drivers for this migration [1], as organizations are no longer required to employ development and system administration teams and to maintain the computational resources required to run them. Additionally, cloud based services often offer guarantees of performance and uptime, through Service Level Agreements (SLAs) protecting organizations from failures in their provided services.

Whereas typical in-house IAM solutions are usually designed to meet the specific requirements of

the target institution, IDaaS solutions are built to accommodate a wide array of well known applications and authentication protocols. As new standards gain traction, IDaaS providers race to include support for them in their solutions. This presents a clear benefit for their client organizations which benefit from the peace of mind of having support for the new protocols whenever they need to implement them without the need to develop additional integration modules. However, the reverse is also true: large scale organizations tend to rely, on a varying scale, on legacy applications which may use archaic authentication protocols no longer supported by the current IDaaS providers. The adoption of one of these IAM solutions requires companies to design and implement additional systems (often in the form of another purpose-built IAM system) to maintain compatibility with the legacy applications.

Despite the clear advantages of the cloud model, its adoption has not kept up with the estimates [2]. Organizations are still reluctant to hand over their user identity data to a third party [3]. The core responsibilities of an IDaaS solution demand that it necessarily takes part in every data exchange transaction and often that it acts as a storage provider for user accounts and personal data. This poses severe challenges to the safety of this data if the appropriate measures are not followed by the vendor. In extreme cases if the IDaaS provider operates a multi-tenant environment, an organization's user database may even be stored alongside other organizations' data increasing the consequences of a security breach. While some of these challenges affect every cloud computing solution, and are not restricted to IDaaS providers, the type of information secured by these systems is often more sensitive to data breaches and thus, requires additional security measures to protect it against unauthorized access.

1.1 The FenixEdu Project

As the number of connected households increased, higher education institutions looked for ways to provide learning resources beyond the scope of the traditional classroom model. This led to the development of the first Learning Management System (LMS), applications where educational resources could be made available to the academic community over a network, often being shared online. As these systems grew in popularity, so did the number of features available in each one, with universities often resorting to in-house development teams to integrate other types of services (such as tuition payments or official document requests), with direct consequences to the product's complexity and maintainability. Both commercial (Blackboard [1]) and non-profit (Moodle [2]) organizations attempted to develop learning management systems that could be customized to each institution's needs.

The FenixEdu Project, started in 2002 at IST aimed to provide a complete and integrated set of open-source software platforms for academic organization management, including a LMS, a Student Management System (SMS) and a Content Management System (CMS) [4]. FenixEdu's extensible and modular approach to the development of these tools was fundamental to the adoption of its products outside IST, allowing each implementing institution to tailor it to its specific requirements.

However, the decoupled nature of this solution also creates its unique set of challenges as a SSO

[1] https://www.blackboard.com/
[2] https://moodle.org/

solution must be deployed to ensure users can roam between the different applications without being asked to re-authenticate with each one. While support for third-party SSO systems (namely, the Central Authentication Service (CAS)) is already implemented in FenixEdu's suite of products, identity management is still bundled with one of the products (FenixEdu Academic) and only a small set of features is available. These features extend beyond the purpose of FenixEdu Academic and thus, should be replaced by a complete IAM solution that offers a central management point for authentication and user identities across the FenixEdu suite of products.

1.2 Objectives

This thesis aims to offer an hybrid approach to the IAM problem by describing and implementing a software solution to allow organizations to take advantage of the ease of integration and extensibility of cloud-based identity providers while still ensuring the safety and security of the data it is responsible for managing, which should be stored on-premises and thus never leave the organization's IT perimeter.

The implemented solution, FenixEdu Connect, leverages the latest authentication and authorization standards (such as OAuth, OpenID Connect and SAML) to provide a unified gateway to user identity data in academic organizations. Its architecture is designed for extensibility and interoperability with external applications, allowing organizations to easily integrate it with existing systems including legacy applications.

In addition, FenixEdu Connect aims to relieve individual applications from the responsibility of having to maintain their own OAuth Authorization server infrastructure to enable access delegation to their resources by providing a centralized OAuth server for the organization, enabling developers to request access to resources from multiple OAuth-enabled applications using the same set of credentials.

As part of the FenixEdu project, the implemented application is naturally open-source with support and contributions being maintained by the project's team.

1.3 Thesis Outline

This thesis's report is divided into five chapters. This section concludes the Introduction chapter where the motivation for the current work is described, along with a description of the problem it aims to solve.

Chapter 2 introduces the set of conceptual foundations required to contextualize the problem this thesis aims to solve.

Chapter 3 describes the proposed solution, along with the main objectives it aims to accomplish.

Chapter 4 provides an evaluation of the implemented solution based on a concrete case study that details the implementation constraints and expected outcomes of the deployment of FenixEdu Connect in a major Portuguese University.

Chapter 5 concludes the report with a brief summary of the obtained results, how they pertain to the initial objectives and a set of suggestions for future developments.

Chapter 2

Related Work

Following the introduction of the problem in the previous chapter, some contextual foundations must be established to sustain the object of study of this thesis. This chapter begins with an overview of the most important theoretical concepts that make up the foundations of this field of study. Then, the key technologies that power the proposed solution are analysed. Finally, three of the leading IAM solutions are briefly analysed.

2.1 Theoretical Overview

In order to contextualize the proposed solution within its field of study, this section will briefly highlight the main theoretical concepts that make up the foundations of IAM systems.

2.1.1 Digital Identity

Identity existed long before humans learned to speak, or records started to be kept. Back then identity was based on a set of easily recognizable attributes, such as physical features, behavior, or common gestures. With the advent of the first languages, the concept of identity was also updated: entities could now be referred by their names. Nowadays, when asked to provide a proof of identity most people will resort to personally identifiable documents, such as a driver's license, or a social-security card. These documents, often issued by a trusted party such as a state or government, carry identifiers (usually a number or a string) that are unique for every person (within that context), enabling third-party entities to identify their owner.

When a customer uses a credit card to pay she is required to enter a Personal Identification Number (PIN) to authorize the purchase. While the card's number is unique worldwide (no two cards have the same number), the user is still required to enter an additional piece of information to ensure the card is being used by its legitimate owner. This constitutes a simple example of credentials, that is, a set of private (and optionally public) data that can be used to assert the authenticity of a given claim (in this case: ”the owner is in possession of the card”).

In addition to identifiers and credentials, there is often a set of information that, while not unique, is at the core of an entity's digital identity: attributes. Attributes define characteristics associated with a given entity, which can either be based on personal traits, such as fingerprint data and eye color, or temporary, such as an email address or a student number. Attributes, unlike identifiers, are not expected to be used to assert a subject's identity. Rather they make up, along with identifiers and credentials, the foundations for a digital identity.

The similarities between the main components of a digital identity and its paper-based counterpart were fundamental to the exponential growth of online services based on the premise of digital identities. Users quickly embraced the possibility of maintaining a set of digital personas, each one with its set of attributes and connections to other users [5].

As digital identities became more prevalent, so did the attempts to use Digital Identity Systems as a risk management tool [6]. One of the most notable example is the creation of No-Flight lists, databases containing personal identifiers of people who should not be allowed to board an airplane in a given territory for security reasons. Throughout the world, airlines rely on these lists to pre-screen passengers and decide if access to each flight should be granted. While these lists are used to manage the risk of violent behavior on flights, this can only be possible if the systems which maintain them are less subject to flaw than the organization's security measures against untrusted adversaries. Otherwise, an attacker would simply shift her focus from penetrating the secure system into forging an identity that would allow her to access the protected resources. Ensuring the integrity of digital identities is thus of paramount importance whenever they are expected to have a unary relationship with a physical subject.

While the policies for digital identity management should ideally be designed in the way that balances the organization's security policies and its business requirements this hasn't always been the case. Historically, IT departments have relied on firewalls and access control technologies to maintain a secure perimeter around the organization's infrastructure, ensuring that any sensitive data would be kept inside. As the Wold Wide Web surfaced, so did a wide array of new business opportunities that forced organizations to abandon the secure perimeter model for the adoption of interoperability standards, such as identity federation or identity aggregation. An example of this change is the advent of the Automatic Teller Machine (ATM), which allowed bank costumers to perform a wide array of bank operations without the presence of a teller. To implement them, banks were required to find a way to authenticate their users in remote locations without the presence of an employee, a problem that was solved by issuing costumers a combination of a card and a unique code (PIN). ATMs forced banks to expand their secure perimeters beyond the realm of the organization's infrastructure to support these remote devices. This change was driven by business requirements as banks quickly realized that ATMs could operate permanently, increasing the number of costumers that could be served simultaneously.

Naturally, the impact of the widespread adoption of the World Wide Web isn't restricted to the banking sector. Universities have continuously leveraged the latest advances in the digital revolution to fuel new learning opportunities and to expand their impact on societies. The rise in the number of institutions providing Massive Online Open Courses (MOOCs) is challenging the traditional classroom model with universities increasing the number of learning platforms and content they share with the community

through digital media. This poses challenges to the identity systems in place at these institutions, which must now support an ever increasing number of users with only partial access to the organization's resources.

Digital identities are, similarly to their physical counterparts, dependent of a trust relationship between all the involved parties. *Windley* [7] defines trust as "a firm belief in the veracity, good faith and honesty of another party with respect to a transaction that involves some risk". Multiple identities for the same user can bear different levels of trust depending on the set of attributes they contain or the context they are originated from. As an example, whereas both can lay the same claims, a shopkeeper is more likely to accept a driver's license as a proof of date of birth than a library card, since it places a higher degree of trust on the government agency that issued the driver's license than it does on the library. In the same way, before asserting a user's identity, a digital system must be able to trust the provided credentials and, ideally, that they are being held by the correct entity.

2.1.2 Identity Management

Identity Management, commonly referred to as Identity and Access Management refers to the combination of policies, processes and technology to ensure the identification and authorization of the users accessing an organization's resources [8]. Policies define constraints and/or standards in order to comply with existing legislation or internal business rules. Processes implement the steps necessary to complete a business process or task. Finally, technology encompasses the set of tools that are used to accomplish business goals in an efficient way that still ensures all security constraints are met. Typical IAM solutions encompass a set of components including, but not limited to:

1. Single Sign-On
2. Access control and authorization
3. User account repository
4. Auditing and reporting

2.1.2.1 Single Sign-On

As the Web moves to a more service-oriented architecture users are faced with the burden of creating and managing an ever increasing number of passwords. Wash et al. [9] conducted a study on the re-use of passwords across different websites where it found that the average user uses 12 distinct passwords to access her online services. While the use of different passwords is encouraged by security experts [9] users also reported engaging in a number of behaviors that degraded the security of their passwords, such as intentionally re-using credentials for frequently used services. Web Single Sign-On systems attempt to tackle this problem by providing users with the ability to use a single set of credentials to authenticate with multiple services. In an SSO architecture, authentication is performed by an Identity Provider (IdP) which can be a separate entity from the actual service hosting the requested resources, usually known as the Relying Party (RP) or Service Provider (SP). While the sequence of steps for an

SSO authentication differs according to the implemented solution, the abstract flow involves the following actions:

1. The user attempts to access a protected resource in a website that is part of an SSO system. This website acts as the service provider. Since the user did not provide a valid set of credentials, the SP redirects her to the Identity Provider for authentication.

2. The user authenticates with the Identity Provider using her credentials. The IdP redirects the user back to the SP, passing in an authenticated token that asserts the user's identity.

3. The SP validates the provided token and authorizes the user to access the resource.

Single Sign-On systems can be deployed as a standalone solution with some even bundling in some basic access control features. However, these systems lack the flexibility to deal with account lifecycle events such as automatic provisioning / deprovisioning and completely leave out the management of the users' profile information that can only be achieved with a complete IAM solution. In addition to providing users with a unified set of credentials (and often a common web interface) for authenticating with the organization's services, IAM systems often rely on SSO solutions to enable users from outside the organization to access internal resources using their own credentials. This is known as Identity Federation and will be discussed in the next section.

2.1.2.2 Access control and authorization

Access control is the process of granting certain subjects access to a resource while denying others access, that is, to use an entity's previously verified identity to control the actions it can take on a set of protected resources [7]. Access control systems are the direct result of business policies. Their task is mainly to automate the enforcement of an existing policy that governs an entity's access to a set of resources. These systems are often founded on the principle of least privilege, which states that users should be given no more access to resources than they need to accomplish necessary functions [10]. How well this is accomplished depends heavily on the granularity of the implemented permission system. There are multiple approaches to permission systems:

- The User-based permission system, popularized by the Unix file system. In this architecture, resources can be directly owned by either users or groups, with users belonging to groups. In addition, the system defines a set of three permission flags: read, write and execute. It is possible to assign these flags to any user or group. A file can, for instance, be owned by a given user, which has complete control (read, write and execute), be readable by users of a group A and both readable and writable by users of a group B.

  While widely used, this system suffers from a set of challenges related to behaviour inconsistencies when a resource is protected by rules targeting both groups and users. As an example, a user can have direct permissions to read a file and be part of a group with permissions to read the same file. Removing access to the group does not remove access to the user who still has his direct

permission flag. This makes it very difficult to assert which resources can be accessed by each user.

- Access Control Lists (ACLs) were designed to tackle this problem by providing an additional abstraction level. Instead of tying users and groups directly to the protected resource, the resource holds a set of ACLs that specify the permission level for each user and group. However, being maintained alongside the protected resource they still pose a maintenance challenge: revoking a user's access to a set of resources requires updating the ACL for each one.

- Role Based Access Control (RBAC) has become the leading architecture of access control for medium to large companies [11]. RBAC is based on two main components: roles and permissions. Users are assigned a set of roles, which can define an hierarchical relation (a user with the *Teacher* role is also granted the *Employee* role). Each role is associated with a set of permissions. Resource owners define the required permissions to access the resources. User's with a role that satisfies the required permissions for a resource are allowed to access it. Different operations on the same resource can require different permissions. A user with the *Student* role may be able to read the online material for a course but she cannot modify it, as that requires a permission only granted to users with the *Teacher* role.

There are other relevant access control architectures not covered in this chapter. However, they all share a common set of components that allows their analysis to be abstracted to a set of interactions between them. Figure 2.1 highlights this abstract architecture. The Policy Enforcement Point (PEP) operates as the frontend for the authorization transaction. It is there that the user directs her request, and it is the component responsible for conveying the reply. The Policy Decision Points (PDPs) are the components where the decision actually takes place, often by consulting a set of stored policies.

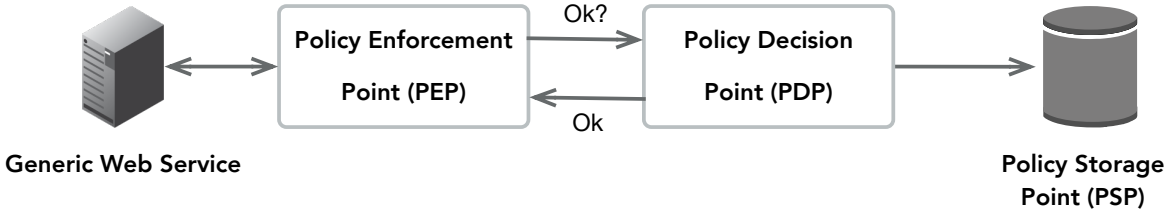


Figure 2.1: Abstract authorization architecture diagram.

2.1.2.3 User account repository

Windley [7] presents an abstract overview for the digital identity lifecycle, consisting of 5 actions:

1. **Provision**: The process under which an IT system is prepared to be accessed by a new user or entity. Depending on the degree of system integration in the organization this action might involve the creation of a single account with a set of roles/permissions or require intervention in multiple systems.

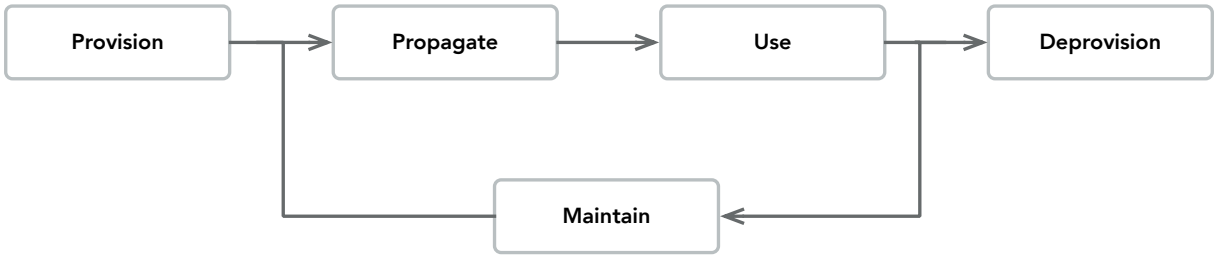


Figure 2.2: The main actions of the digital identity lifecycle.

2. **Propagation**: In organizations that do not use a centralized directory to store user accounts it may be necessary to propagate information on the newly created accounts for multiple systems. Even if this type of account repository is used, some systems need to be made aware of the creation of new accounts to provision other resources, such as creating directories, or databases.

3. **Using**: Once an identity is created and propagated across the necessary systems, it is used until needed.

4. **Maintaining**: Similarly to their physical counterparts, digital identities are rarely static. As attributes change, systems update the stored identity records and repropagation occurs, if needed.

5. **Deprovisioning**: The process under which an identity is removed from the IT system. Systems need to be notified of the change so that any resources associated with the entity can be released. As with provisioning, the degree of system integration has a direct influence in the time and resources required to perform this task.

A good Identity and Access Management system should, therefore, implement the processes to assist the organization in the five phases of the lifecycle. As an example, an employee joining the organization should have instant access to the internal resources without the need to wait for lengthly manual enrollment processes. Similarly, an employee that is leaving the organization should have her privileges revoked in a timely manner. These features have been coined as *zero-day start/stop* by multiple IAM providers and greatly reduce the on-boarding time for new accounts while providing additional protection against misuse of existing systems by ex-employees.

IAM systems provide a common gateway to access information that may be spread throughout multiple systems. As an example, It is not uncommon for an organization to have a Human Relations and Payroll (HRP) system to manage staff, a Customer Relationship Management (CRM) solution to manage engagements with clients and an Enterprise Resource Planning (ERP) platform to manage its business practices, all of which maintaining their own silos of identity data. In the physical world, users perform the mental task of aggregating their identity relationships. A user will hardly think of herself as a student at IST with a specific ID. Instead, she sees her role at the university as just one of the many identities that she possesses. IAM systems should assist in the aggregation of identity attributes from multiple systems in the organization by providing a common interface to manage them.

2.1.2.4 Auditing and reporting

Identity and Access Management systems are often at the core of an organization's IT infrastructure with most services, if not all, relying on these systems to provide them with critical information for the security of the organization's digital resources. As states and governments push for an increase in the regulatory standards and policies that organizations must implement, the pressure to provide audit trails for the personal information these organizations store is forcing IAM systems to collect and monitor data on how this information is being accessed. As the gateways for user digital identities, IAM systems should be able to provide accurate trails to account for every data transaction processed. Ideally, these trails should be able to answer the five Ws (a journalistic concept consisting of five questions that start with the letter W) for every transaction: (1) what data was accessed? (2) when was it accessed ? (3) who accessed it ? (4) why was it accessed ? (5) where was it accessed from ?.

2.1.2.5 Discussion

The myriad of identity management standards and frameworks, often times independently developed, coupled with the lack of a common ground for how identity should be created, managed and distributed within the organizations leads to brittle approaches to identity management. As service-oriented architectures become more popular [12] and organizations move more resources to cloud-based environments, the concept of Identity as a Service gains traction as the preferred architecture of access control at scale. IAM systems are now forced to adapt to this new paradigm or risk perish.

Lewis [7] describes the next generation of Identity and Access Management systems as a combination of: "(...) directory services, role-based user provisioning, delegated administration and self-service administration for passwords and other attributes. General-purpose, strong authentication systems, along with good credential management." The implemented solution will lend on this concept to provide an IAM system capable of supporting the needs of academic institutions.

2.1.3 Identity Federation

Any reader that has ever used a citizen card or a driver's license to assert her identity to a third party has experienced identity federation. The service trusts that the provided identity claim has only been granted after the user passed a set of vetting checks from the government entity that issued the card. By trusting the government the service is able to authenticate the user, even though the presented credentials have been issued by another identity provider. This presents an obvious advantage to the user, who is not required to carry identification cards for every service that she wishes to use and can instead carry a single ID card. However, there are also significant advantages to the service provider when it chooses to use federated identities: by offloading the authentication process to a trusted third party with a greater vetting capacity not only is it simplifying its authentication responsibilities as it enjoys the additional security offered by the increased user scrutiny of the identity provider (in this case, a government agency, which is able to perform identity checks more thoroughly than any private organization).

The increase in the number of Internet-based services led to a new set of issues revolving around he need to secure protected resources across multiple services while still providing an easy way to access them. While the initial SSO systems alleviated this problem, enterprise users were still faced with the challenge of having to access systems in another organization, which was not part of their SSO realm. This access would have to be limited in scope to the essential systems and often required the creation of a set of guest credentials, which, in turn, would have to be memorized for every external service the user required access to. While organizations usually had security policies in-place to manage internal passwords there was little control over these external credentials. An employee whose employment had been terminated would have his internal credentials suspended, but could easily retain access to the external services.

The solution came in the form of a decentralized IAM architecture, known as Federated Identity Management (FdIM). Federated Identity refers to the portability of identity information across multiple domains, allowing users from one organization to access data or systems of another domain using the same set of credentials.

The *eduroam*[1] initiative, which aims to provide secure access to wireless networks at academic and research organizations in Europe, is an example of the use of identity federation. When a student from a given institution attempts to access the *eduroam* network at another organization, she is asked to provide a pair of credentials for authentication. Instead of contacting the organization's IT services to obtain a pair of guest credentials, the student can simply use the same username and password that she already uses in her home institution. The organization's *eduroam* infrastructure analyzes the provided user ID (which has a fixed format of username@institutionDomain) and determines that it is not responsible for that domain. As such, it proxies the request to a national entity that then forwards the request to the user's home institution, which validates the credentials and provides an authorization decision. Naturally, this is only possible if the organization trusts the student's original institution and if it can be sure that the authorization decision will come from the correct entity. Both in the *eduroam* initiative and in typical web-based identity federations this is accomplished through the use of digital certificates, but other mechanisms are available.

Identity Federation relies on a set of common standards to achieve interoperability among systems, mainly OAuth, SAML and OpenID, which will be analyzed in the next sections.

2.1.4 Public Key Cryptography

While the history of cryptography can be traced back to 1900 BC [13] as a way to hide information from eavesdroppers it has always been based in the existence of a common secret, shared between the sender and receiver through an alternative medium and used both to encrypt and decrypt the data. This is known as symmetric cryptography and has been subject to significant advances (both in algorithm strength and performance) which has made it the preferred way to encrypt large volumes of data. However, the issue of key distribution poses severe challenges to the use of these algorithms, since physical

[1] https://www.eduroam.org/

media often involves a courier or requires the key to be printed. On the other hand, if a phone or online media is used, there is also the possibility of it being eavesdropped.

The solution came in the form of public key cryptography, which relies on a set of related keys (known as *public* and *private*) with one being used for encrypting the data and the other to decrypt it. While it is possible to infer the public key from the private key these systems rely on the infeasibility to perform the inverse action. The public key can, therefore, be published to a known key repository or be made available in some other form allowing any one to generate a ciphertext that can only be decrypted by the intended party.

Rivest, Shamir and Adleman described, in 1977, the first algorithm to implement these principles. It was named after the author's initials: RSA. In it, the authors describe four base properties for public-key crypto systems that correlate a generic encryption procedure E with its corresponding decryption procedure D and the plaintext message to encrypt M.

1. Deciphering the enciphered form of a message M yields M. Formally,

$$D(E(M)) = M \tag{2.1}$$

2. Both E and D are easy to compute.

3. By publicly revealing E the user does not reveal an easy way to compute D. This means that in practice only she can decrypt messages encrypted with E, or compute D efficiently.

4. If a message M is first deciphered and then enciphered, M is the result. Formally,

$$E(D(M)) = M \tag{2.2}$$

A function which satisfies the four properties is known as a trap-door one-way permutation, a concept originally introduced by Diffie and Hellman [14]. A trap door function is easy to compute in one direction but very difficult to compute in the opposite, that is, until private "trap-door" information is available, in which case the inversion becomes easy to perform.

Figure 2.3 depicts a common situation where Alice wants to send an encrypted message to Bob. Let E_A, E_B be the encryption procedures that use Alice's and Bob's public keys respectively. Conversely, let D_A and D_B be the respective decryption procedures which use their private keys. The following list outlines the steps necessary to achieve this under a public cryptography system:

1. Bob has previously generated a pair of keys and shared his public key with Alice (for instance, by publishing it to a central key repository). Alice retrieves Bob's public key E_B.

2. Alice encrypts the plaintext M with Bob's public key, retrieved from the repository and sends the resulting chiphertext to Bob.

$$C = E_B(M) \tag{2.3}$$

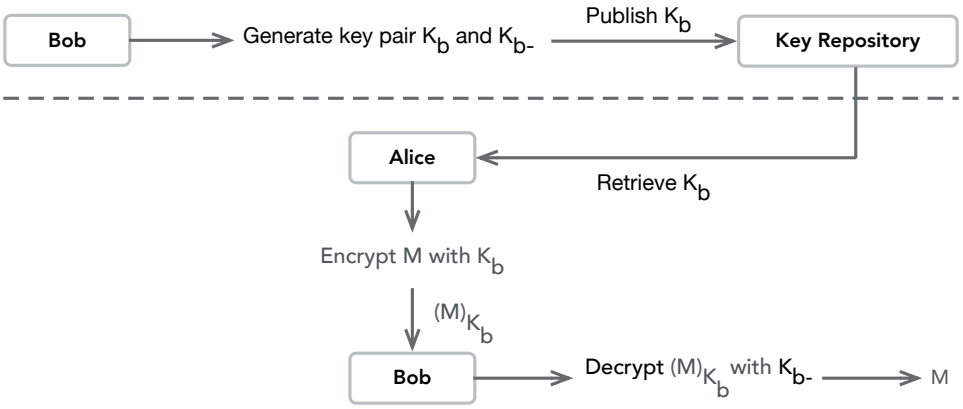


Figure 2.3: An overview of the process of sending a message using public key cryptography.

3. Bob receives the message and decrypts it with his own private key, obtaining the original plaintext.

$$D_B(E_B(M)) = D_B(C) = M \tag{2.4}$$

2.1.5 Digital Signatures

The last property of public key crypto systems, highlighted in equation 4 allows for an additional use of these systems: digital signatures. As with their traditional counterparts, digital signatures assert the identity of a given document's owner. Additionally, they are able to guarantee the integrity of the signed content, ensuring that is hasn't been modified after it has been signed.

For a given user Alice to send a signed message to Bob the following sequence of steps should be carried out:

- Alice starts by computing the message's signature S by using her private key, D_A.

$$S = D_A(M) \tag{2.5}$$

- The resulting signature is then encrypted using Bob's public key, E_B and sent to Bob.

$$P = E_B(S) \tag{2.6}$$

- Bob reverts the encryption step, using its decryption procedure D_B and thus retrieving the signed message S.

$$S = D_B(S) \tag{2.7}$$

- Bob attempts to recover the original message using Alice's public key, E_A ensuring that it came from Alice (as only Alice, in possession of her private key could have generated a payload that could be decrypted with her public key).

$$M = E_A(S) \tag{2.8}$$

While these principles paved the way for the use of digital signatures they were vulnerable to multiple attacks [15]. New techniques then evolved to signing a hash of the document instead of its data which, among other advantages, allowed recipients to read the document without the presence of the validation key. It also decoupled the signature from the signed document itself.

2.1.6 Digital Certificates and Public Key Infrastructure

These simple examples outlined the main advantage of public key cryptography systems: the private keys never have to be distributed, rendering attacks based on eavesdropping useless. However, it should also be clear that the entire system relies on the security of the key repository. An attacker that is able to infiltrate the key repository would be able to replace Alice and Bob's public keys for its own which would allow her to perform a Man in the Middle attack by reading the messages sent between Alice and Bob and replaying them as if they had been sent by the other party. Digital Certificates help to alleviate this issue by bundling an entity's public key and some identifying information in a self-contained structure that is then digitally signed by a trusted entity. These entities, known as Certificate Authorities (CAs) perform additional layers of validation before issuing a certificate, such as verifying the requestor's identity through a government issued ID card [7].

In addition to subject authentication and certificate generation/distribution, a CA is also responsible for ensuring that revoked certificates are no longer usable. Revoked certificates are placed on a Certificate Revocation List (CRL) which is signed by the CA. However, applications still need to know if a certificate is in the CRL. There are three main approaches for solving this problem:

- Every time an application wants to use a certificate, it checks with the CA that issued it if it is still valid. This may be incompatible with the application's performance requirements.

- An application may subscribe to a service from the CA that sends the CRL whenever it is updated. However, these messages may be blocked without the application ever being aware of it.

- The application may periodically query an online service provided by the CA.

Certificate Authorities can develop a hierarchical tree of trust, whereby the keys used to sign certificates issued by CA B can be digitally signed by another certificate authority, CA A, expressing a trust relation between the two: a user who trusts CA B also trusts CA A. In this simple hierarchy CA A would be known as the root CA. The infrastructure needed to operate public key cryptography at scale is formed by a web of CAs and is known as Public Key Infrastructure (PKI) [7].

Since public key cryptography can take from 100 to 1000 times more time to perform than its symmetric counterpart [7] it is rarely used to encrypt large amounts of data. Instead, it is often used to securely negotiate a secret symmetric key between the parties in the beginning of the communication. Transport Layer Security (TLS), also known as Secure Sockets Layer (SSL) is a common example of this use case.

2.2 Technologies

2.2.1 OAuth

As the number of online services increased so did the need for simple and efficient data exchange and integration amongst them. The rise of web Application Programming Interfaces (APIs) as a privileged interface between systems has been accompanied by its own set of challenges. Applications that needed to access user's data controlled by a third party service were forced to request (and often store) the user's access credentials. Not only was this a significant vulnerability (as a data breach could expose all the users' passwords) as these credentials allowed for full, unrestricted access to the user's accounts, which was well beyond the realm of information that was required by most services. This forced users to carefully balance the advantages of using those third party services with the possible security risks associated with sharing their personal account details.

The solution came in the form of OAuth, an open and standardized web resource authorization protocol which allows users to grant third party entities access to a subset of their protected resources without directly having to share their credentials [16]. OAuth's origins can be traced back to 2006 and stem from a desire to allow the use of OpenID (a distributed authentication protocol) to secure Twitter's public API. The lack of a viable open standard for API access delegation led to the drafting of the first proposal that would, a year later, become OAuth 1.0.

A simplified version of the OAuth 1.0 flow [17] is as follows:

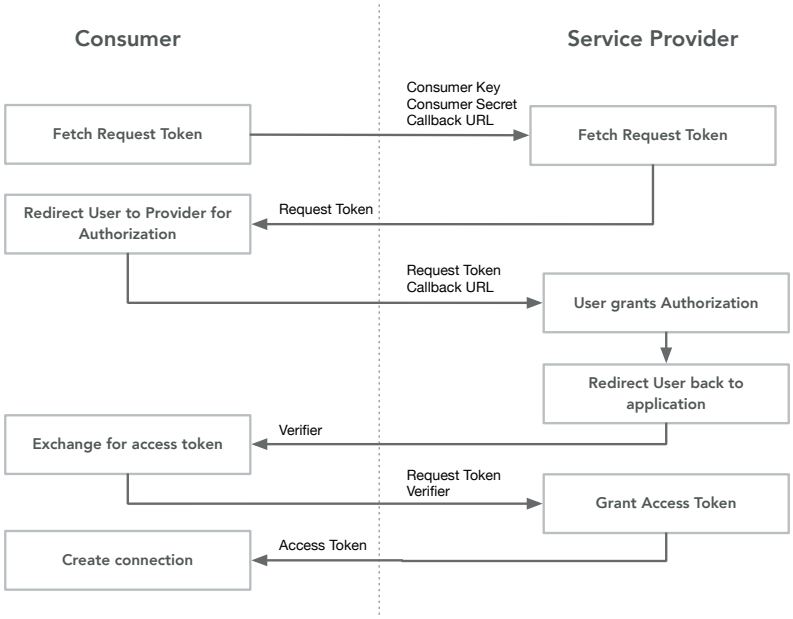


Figure 2.4: A sequence diagram of an authorization transaction using the OAuth 1.0 protocol.

1. The client application sends a signed request to the OAuth-enabled server containing its consumer key and a callback Universal Resource Locator (URL).
2. The server generates a request token and an associated secret, sending them back to the client application.

3. The application directs the user to the authorization server, along with the request token. The user is asked to authenticate with her credentials and authorize access to one or more sets of information (scopes).

4. If access to the required scopes is granted, the server will redirect the user back to the application, along with the original OAuth request token and a set of temporary credentials, known as an OAuth verifier.

5. The application can then exchange this verifier for an access token and, optionally, a refresh token. The access token can now be used access OAuth protected resources on behalf of the authenticated user.

The lack of adoption of SSL/TLS at the time forced developers to design the protocol around digitally signed requests that could be transmitted over an insecure medium. This led to an increase in the implementation complexity of the algorithm, since both parties had to perform cryptographic operations on the exchanged data. In addition, since there was no mandated signature method each implementation was free to define its own requirements, which posed severe challenges on the use of OAuth between different service providers.

While multiple services used OAuth 1.0 to secure their APIs, the significant complexity of its authorization flow and the lack of support for the next generation of mobile and/or desktop app clients led to the development of the second iteration: OAuth 2.0 [18]. By removing the need for digitally signing the requests from the client applications and instead relying on SSL to ensure the messages could not be tampered with while in-flight the new version of the OAuth protocol completely removed the need for the use of cryptography in the client applications.

OAuth 2.0 defines a set of five flows, commonly known as grants, for applications to obtain access tokens [18]:

- **Authorization Code Grant:** Similar to the OAuth 1.0 authorization flow. The user is redirected to the OAuth server for authentication and authorization granting. The server redirects the user back to the service provider with a token (referred to as an authorization code) that can be exchanged for the access/refresh token pair.

- **Implicit Grant:** Provides an abbreviated flow where after the user authentication the server immediately redirects the access token during the redirect to the service provider.

- **Resource Owner (Password) Grant:** Allows the service provider to directly exchange the user's credentials for an access token.

- **Client Credentials Grant:** Allows the client application to obtain an access token on behalf of itself, outside of the context of a user.

- **Refresh Token Grant:** Allows a client application to exchange a previously obtained refresh token for a valid access token.

The authorization code grant constitutes the most common implementation of the OAuth 2.0 framework. Before any application is allowed to take part in any of these flows it must first be registered with the OAuth authorization server, which issues a pair of credentials (client ID and client secret) that identify the application.

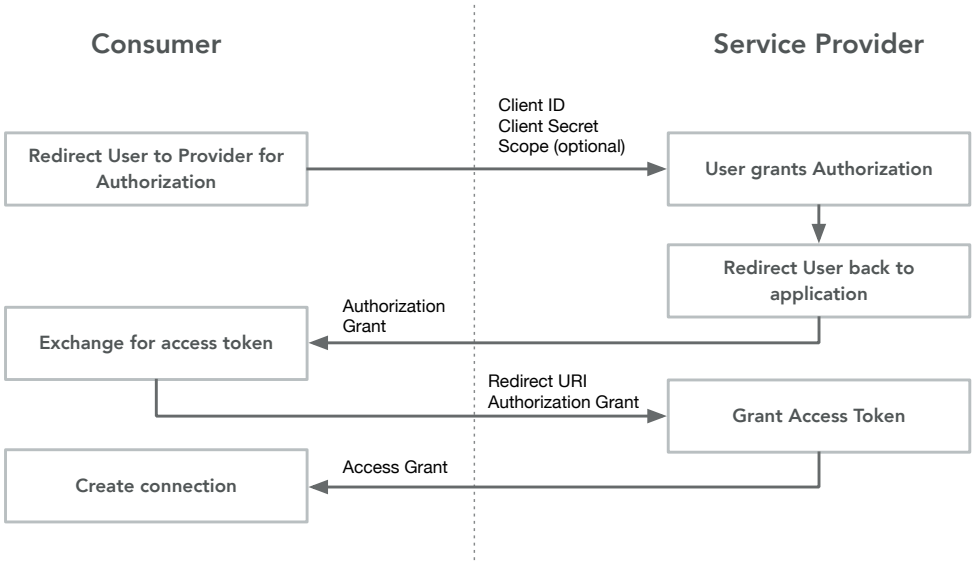


Figure 2.5: A sequence diagram of an authorization transaction using the OAuth 2 protocol.

Figure 2.5 depicts an overview of the OAuth 2.0 flow where the Service Provider and the OAuth authorization server have been bundled in the same entity. The authentication flow starts with the initial user redirect from the application to the identity provider, passing in its client ID, client secret and a redirect URI that should match the one provided when the application was registered. If the user authentication is successful and access to the required scopes has been granted, the OAuth server will forward the user to the specified redirect URI, appending a *code* parameter that represents the user's session. Finally the application uses the provided code along with its client secret to obtain the final set of credentials: the access token and an optional refresh token. Future requests to protected resources use the provided access token for authentication. If the OAuth implementation uses short lived tokens (which expire after a predetermined amount of time) the refresh token can be used to obtain a new access token from the OAuth server's token exchange endpoint.

2.2.1.1 OAuth 2.0 as a Single-Sign On solution

The flexibility of OAuth 2.0 allowed developers to extend its use beyond API authorization. The widespread adoption from companies with significant user bases, such as Facebook[19], Google[20] or Twitter[21] led to the development of the first SSO systems based on the OAuth protocol. The significant business incentive [22] for client applications to tap into these massive user bases has led to the proliferation of OAuth as a Single-Sign On service. Most implementations rely on the regular OAuth 2.0 flows with an additional last step where the client application queries the IdP's API for the user's personal information and creates a local session. Since authentication is federated to the IdP client applications are no longer responsible for having to manage user credentials or implementing password reset strategies. Access

to the user's profile information and social graph is also often cited as a strong business advantage of using these OAuth-based SSO providers [22]. However, since the OAuth 2.0 framework was designed to provide authorization decisions (and not authentication) some of the earlier implementations of these systems have been shown to be vulnerable to Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF) attacks [23] that exploit some of the protocol's features that are at the base of an SSO implementation, such as the automatic authorization granting of a previously authorized application request. Any system aiming at using OAuth as a base for its authentication layer should, therefore, be designed to mitigate these vulnerabilities.

2.2.2 OpenID

Traditional SSO solutions are based on a fixed gateway that is used by multiple services when a user needs to authenticate herself. A developer waiting to use a given SSO provider in his application will need to go through a two-step process where it registers the application in the identity provider (thus adding it to its list of allowed clients) and configures the application to redirect users to the desired IdP.

While the rationale behind this approach is easy to understand it poses significant challenges [24] to services which need to support multiple SSO providers. A blog page that wants to authenticate users with Facebook, Google and Twitter will need to be registered with the three providers and manage their distinct authentication flows. A solution to this problem was purposed by the open source community in 2005 under the name of OpenID. OpenID is an open and decentralized authentication protocol created with the aim of simplifying the use of multiple identity providers [25].

A user starts by enrolling into any Identity Provider that supports the OpenID protocol and collects her unique OpenID URL. When attempting to access a service which supports this authentication protocol the user provides it with her unique URL and is redirected to the associated IdP for authentication. If this step is successful, the IdP redirects the user back to the service, along with her OpenID identifier and profile information, digitally signed by the IdP.

While OpenID simplifies the process of IdP discovery (by integrating it into the protocol itself) the distributed nature of the protocol necessarily leaves trust outside of its scope of responsibilities. Since any IdP can become an OpenID provider services that rely on strict user verification (such as banking or retail) need to perform additional checks to assert each user's identity. This poses a significant challenge to the widespread adoption of OpenID which has prevented it from being used in high security settings due to the increased overhead required to establish a trust relation with the signed-in user.

2.2.3 OpenID Connect

OIDC, published in 2012, is an authentication layer built on top of the authorization capabilities provided by the OAuth 2.0 framework [26]. OIDC builds upon the foundations of OpenID and provides an interoperable protocol, based on RESTful requests to provide user authentication.

When a user attempts to access a protected resource the service provider will send a regular OAuth request to the authorization server, requesting access to the `openid` scope. The authorization server

(which may or may not also act as the IdP) will request the user to authenticate with an existing set of credentials and to authorize the requested access to personal information by the service provider. If the user grants authorization to this data she will be redirected back to the service provider, along with an access token (standard for an OAuth authorization request) and an additional object, known as the ID Token. The ID Token is packaged as a JSON Web Token (JWT) and contains a set of information that allows the service provider to assert the user's identity. This JWT is digitally signed by the IdP to prevent it from being modified in transit, if an unsecured connection is used. In addition to user information the ID tokens can also contain records that identify the issuing party, the target application of that token (commonly known as its audience), creation and expiration timestamps and optional security features (such as nonces).

In addition to the ID token, the standard also defines a set of endpoints [26] to ensure interoperability between the different service providers and IdPs:

- `/authorization`: The authentication entrypoint. The service provider redirects the end user to this endpoint, along with its client ID and redirect URI. The user performs the initial authentication and is asked to authorize access to the requesting application.
- `/token`: The token exchange endpoint. Allows exchanging an OAuth 2.0 grant (such as an authorization code or the user's credentials) for a valid access token and ID Token.
- `/userinfo`: Allows applications to access the user's profile information (the same that is present in the provided ID token). Since the ID token may contain an expiration date (after which it must not be consumed by the service provider) the applications may use the access token to access this token and get the user's updated profile information.

The OpenID Connect specification also defines additional extensions with optional endpoints for token introspection (allowing a service provider to check if a given token is valid without having to perform the validation locally), token revocation, and client application registration. Since OIDC implementations are free to map each of these endpoints to any path an additional extension was created to allow client applications to automatically determine the correct endpoint locations. Published as the OpenID Connect Discovery protocol [27] the specification defines a fixed endpoint, mapped to the `/.well-known/openid-configuration` path where IdPs which opt-in to the discovery protocol must expose a JSON object (with a fixed schema) describing their OIDC configuration (including endpoint URLs) allowing for the dynamic discovery of their OIDC configuration by developers and client applications.

2.2.4 JSON Web Token (JWT)

JSON Web Token is an open standard that defines a compact and self-contained way for securely transferring information between parties as a JSON object [28]. The standard defines an object comprised of three individual sections with the first one, commonly referred to as the header, providing information on the cryptographic operations to apply to the token's contents by listing the algorithms used and whether the JWT is signed or encrypted.

The second section, also know and the body or the claims set is comprised of a set of key/value pairs that make up the set of assertions that the sending party wishes to expose. The JWT specification defines a number of claim names with specific meanings, which are reserved. Each implementation is, however, free to expand this existing set with as many user defined claims as required. Since most JWTs are issued with a specific purpose (and a set of target users/applications) claim name collisions are not a significant problem. Nevertheless, in the interest of achieving the highest possible level of interoperability between systems and allowing JWTs to act as a universal container for user claim transfers the Internet Assigned Numbers Authority (IANA) currently holds a public registry for the reserved JSON Web Token claim names.

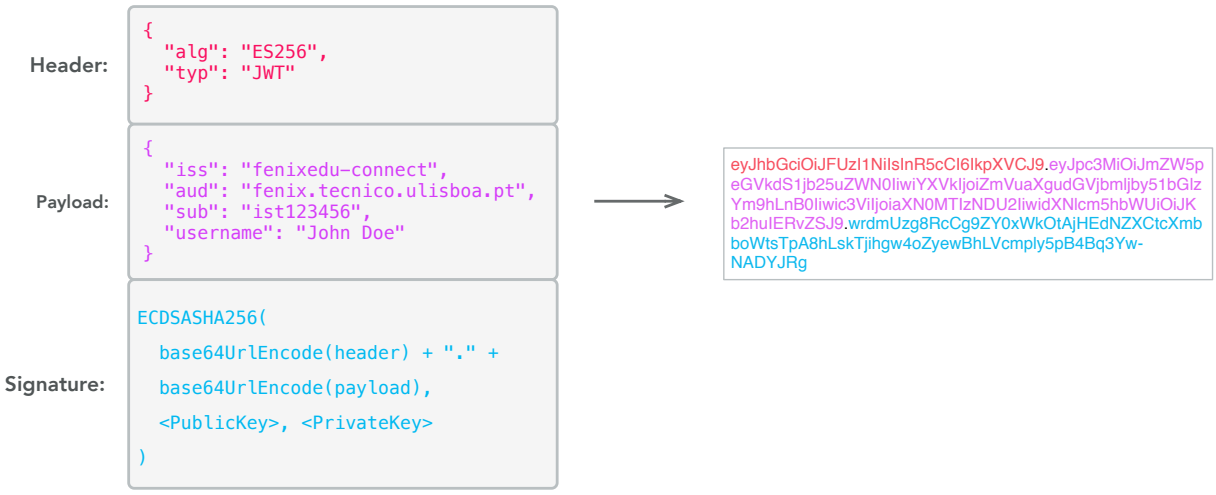


Figure 2.6: A breakdown of the three sections that make up a JSON Web Token.

To create a JWT the first and second sections and individually encoded with base64 (to ensure its safe transfer over Hypertext Transfer Protocol (HTTP)) and appended with a single period ('.') as a separation character. The resulting payload can now either be signed by the issuing party, allowing the receiver to validate the token's authenticity and integrity, or encrypted, ensuring the token's contents remain secret until its delivery to the indented audience. Both methods generate additional data which is appended to the previous two sections with another period as a separation character. While the JWT specification allows for the creation of unsigned and unencrypted tokens (also known as Unsecured JWTs) this severely limits the range of applicable scenarios for JWT's usage and thus, they are not frequently used [29].

2.2.4.1 The JOSE Framework: JSON Web Signature (JWS), JSON Web Encryption (JWE) and JSON Web Key (JWK)

While JWT defines the base container for transferring user claims between multiple parties one of its most useful feature is the support for easily signing or encrypting the token's contents.

JSON Web Signatures [30] defines a set of algorithms for signing JWTs which allow its intended audience to validate the token's authenticity and/or integrity. Once a signed token is received, the header

can be inspected to determine the algorithm used to generate its signature data. The consumer application should be in possession of the cryptographic secret required to validate the signature (which can either be a shared secret, in case a symmetric algorithm was used or the issuer's public key if the token was signed with an asymmetric algorithm). By re-signing the header and payload fields with this cryptographic secret and comparing the obtained signature with the signature data found in the token the consumer party can validate the token has been issued by the intended party and not modified in transit. Signatures are usually performed using a Keyed-Hash Message Authenticator Code (HMAC) a symmetric algorithm which combines the data to sign with a secret using a cryptographic hash function. The resulting signature can only be verified in the presence of this shared secret. In addition to HMAC-based algorithms JWS also supports signatures using RSA-based algorithms where a private key is used to sign (and optionally verify) a JWT while its public counterpart is limited to the latter. In a shared-secret environment, where all the entities are in possession of the cryptographic key used to sign the tokens, it is impossible to reliably establish the identity of a specific message's sender. When using asymmetric cryptography (and under a controlled set of conditions) no party is able to create signed messages on behalf of another, since the private key used to sign the tokens is only know to its respective owner.

JSON Web Encryption aims to provide a way to ensure the confidentiality of the token's claims by encrypting it, which prevents it from being inspected in-transit [31]. Once again two schemes are available: algorithms based on a shared secret and the asymmetric alternatives. While symmetric algorithms offer some performance advantages they are subject to the significant challenge of key distribution, since the disclosure of the encryption key exposes all the previously shared messages. The public/private-key algorithms are implemented as the reverse of the JWS standard. The issuer encrypts the message with the consumer's public key. This creates an encrypted payload that can only be decrypted by the receiver's private key, thus allowing the token to be sent over unsecured channels and still retain its confidentiality. However, unlike JWS where the message's authenticity can be verified by any interested party (provided that a suitable PKI is in place) in JWE since the message is encrypted with the target's public key there is no way to validate the identity of its sender.

Both JWS and JWE rely on the consuming party's ability to retrieve the appropriate signature / encryption key from the issuing entity. Since JWTs are designed to act as an interoperable container for claim transfers between distinct systems it was necessary to define a common format for exchanging these keys. JSON Web Key defines a standard way to describe a key in a JSON object, with a fixed set of key values to represent its properties [32]. An identity provider that issues JWTs will often expose an endpoint with the keys used to sign its tokens, in the form of JWKs, allowing consuming parties to validate the token's authenticity. Since JWT implementations often resort to multiple keys to sign its tokens, the Internet Engineering Task Force (IETF) also defines a standard for sharing a set of JWK objects: the JSON Web Key Set (JWKS), which simply wraps the keys in a JSON object with a single ”keys” array field.

2.2.5 SAML

The inherent complexity level of a Single Sign-On solution greatly depends on the location of the applications that need to share authentication data. While two applications that reside on the same domain (for instance `A.foo.org` and `B.foo.org`) can easily share a cookie with a `.foo.org` domain this is not possible if one of the applications needs to be accessed through a different domain. These cases require the presence of a separate service, responsible for managing authentication requests, known as an identity provider. The Security Assertion Markup Language is an open standard, based on the Extensible Markup Language (XML) for the exchange of authentication and authorization data between an IdP and a Service Provider. [33].

SAML's architecture is made up of four major components:

- **Assertions:** Similar to claims in a JWT, SAML assertions specify the set of information that is to be transferred between the two parties involved in the SAML transaction. They can contain authentication data (such as the authenticated principal's name or user ID), identity attributes (such as the authenticated user's name or email address) or authorization decisions (such as if the authenticated user has access to the requested resource).

- **Protocols:** Describe the request/response flows and the valid SAML assertions that make up each message in a transaction.

- **Bindings:** Specify how SAML messages are transported. SAML does not enforce the use of any transport mechanism, delegating the decision to each implementation which can support one or more bindings according to its requirements. Common SAML bindings are the Browser POST, which transmits the messages in a hidden field in a form sent from a browser or the Simple Object Access Protocol (SOAP) Binding which uses SOAP messages to carry the SAML payload.

- **Profiles:** Combine assertions, protocols and bindings to perform a predefined task. The Web Browser SSO Profile is one of the most common profiles in the SAML specification and it describes the details of a set of request/response pairs that allow applications to delegate authentication to a third party service (the IdP) requesting, optionally, additional attributes to be returned along with the authenticated user's ID.

Figure 2.7 presents a diagram of the sequence of steps required to authenticate the user of an application with a centralized IdP using SAML's Web Browser SSO Profile. The user starts by requesting access to a protected resource. Since the service provider is unable to detect a valid authentication, the user is redirected to the IdP where she is authenticated. The IdP redirects the user back to a URL previously registered by the service provider, known as the Assertion Consumer Service (ACS) endpoint along with a SAML response containing an Authentication assertion that provides it with the user's ID and optional identity attributes. The ACS endpoint redirects the user to the original endpoint which, in the presence of a valid authentication, returns the requested resource to the end user.

SAML powers some of the biggest identity federations worldwide, with a strong presence in the

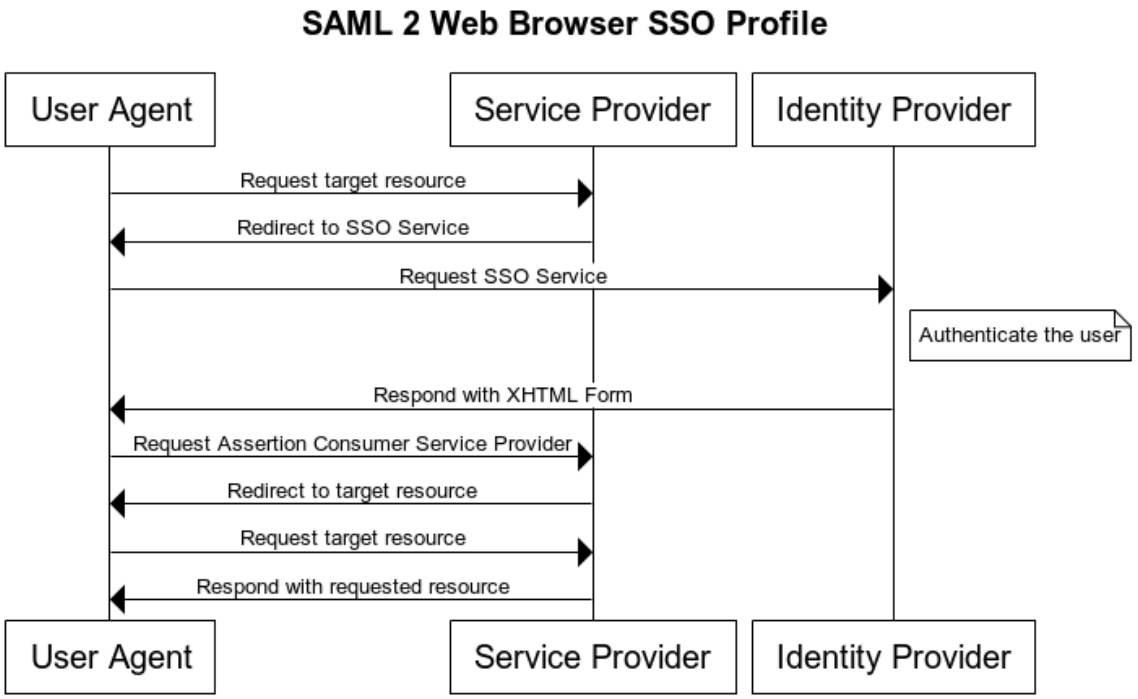


Figure 2.7: A sequence diagram of SAML's Web Browser SSO Profile.

education and public sectors through initiatives such as the InCommon Federation[2], which provides a common authentication and authorization layer for educational and research institutions in the United States or the European Union's project STORK[3] which aims to allow users to access public services in every member country using the same credentials.

2.3 Existing Solutions

The following section will explore three of the main IAM solutions currently in use on enterprise scenarios: Okta, Auth0 and Shibboleth. While both Okta and Auth0 were selected as a representative sample from a large set of comercial IAM products Shibboleth is the only open-source solution with a relevant level of adoption that allows it to be featured in this comparison.

2.3.1 Okta

Okta provides a modular Identity and Access Management solution spanning across six products that can be adjusted to the organization's needs [34]. Although it makes available some integrations with on-premise user directories it is mostly a cloud-based solution running on a hosted environment.

Okta is based around the concept of a customizable web-portal that acts as a gateway for registered applications, with support for SAML and OpenID Connect. For applications that require static credentials

[2]https://www.incommon.org/federation/
[3]https://www.eid-stork2.eu/

(and thus are unable to use identity federation protocols) Okta offers administrators the possibility of saving these credentials in an encrypted form using unique keys for each user.

The system can be configured either via the included administration dashboard or its RESTful APIs. Okta's authentication API allows developers to deeply integrate it within existing applications with login happening purely over RESTful requests, thus removing the need to redirect the user to the IdP and back. There is support for the most common MFA factors such as SMS, Push Notifications, Time-based One Time Password (TOTP) and Universal Second Factor (U2F).

Okta provides built-in integrations for more than five thousand applications simplifying the initial setup flow. Its *Universal Directory* feature allows for two-way mappings to be setup for the profile information exchanged with each application. As an example, a user accessing Salesforce can have its username exported as a combination of his first and last name with the same attribute being set to an internal identifier when accessing the organization's expense tracker software. The *Lifecycle Management* solution allows for complex workflows to be defined enabling, for instance, to implement a multi-step approval process for access to some protected resource.

In terms of extensibility, the comercial nature of this product severely limits the possibilities beyond the existing Representative State Transfer (REST) APIs. Okta has a complex pricing scheme that depends heavily on the number of products subscribed, reaching 10$/user/month if all six products are used.

2.3.2 Auth0

Auth0 is an integrated cloud-based IAM solution with a strong focus on enterprise integrations. However, unlike Okta, it also offers a solution that runs on-premise in a virtualized environment partially controlled by Auth0.

At its heart is the Auth0 Dashboard, a web-based application that combines both the user and administration portals. Beyond the typical IAM features, Auth0 innovates by providing an additional configuration layer that allows for the real time customization of the system, without any downtime. This feature is known as *Rules* and is made up of simple Javascript functions that are executed as part of a transaction whenever a user authenticates to an application. Rules can be used to customize the returned User Profile, create complex authorization conditions, notify external systems of user activity and even conditionally require 2-factor authentication for specific requests (for instance, when a suspicious login attempt is performed). Rules have complete access to the logged-in user's profile, the authentication context (incluiding IP address, location and target application) and Auth0's Management API and run after a successful primary authentication but before the authorization step (when the user gains access to the intended resource). As an example, the following snippet implements a rule that restricts single sign-on to users with a verified email address.

Auth0 is a comercial platform with a complex pricing structure that depends on the feature sets required and the maximum number of active users per month. Pricing is decided on a case-by-case basis but, as an example, it would range between 2 and 4$/month/user for an organization with an average ten thousand active users per month.

```
function (user, context, callback) {
  if (!user.email_verified) {
    return callback(new UnauthorizedError('Please verify your email before logging in.'));
  } else {
    return callback(null, user, context);
  }
}
```

Listing 1: An example of an Auth0 Rule that restricts single sign-on to users with a verified email address.

2.3.3 Shibboleth

Shibboleth is an open source implementation of an Identity and Access Management solution based on SAML which is distributed across three main components: the Identity Provider (IdP), the Service Provider (SP) and the Discovery Service (DS). The three components can be independently deployed to fit the organization's needs [35].

Since it only provides an identity layer on top of the existing infrastructure Shibboleth does not have the concept of a built-in, unified user store. Instead, LDAP is used to connect to an existing directory on the organization's network. Additionally, there is no built-in dashboard for CRUD-like operations. Identities need to be managed in a separate component that interacts with the LDAP store directly.

Shibboleth is implemented as a Java appplication, running on an application container (such as Tomcat or Jetty) and is based on the Spring framework, specifically, Spring Web Flow. A flow definition contains an ordered sequence of steps that are individually executed to authenticate a given user, ranging from the initial form presentation to the validation of the provided credentials. While this approach allows organizations to deploy complex authentication strategies it comes with the increased cost of having to implement the required flows from the ground up.

The open source nature of Shibboleth makes it inherently extensible. Nevertheless, there is an active effort to provide developers with an ever increasing number of extensibility points with well-documented examples that allow organizations to plug in simple pieces of logic without the need to completely fork from the application's main codebase.

Chapter 3

Proposed Solution

This chapter describes the proposed solution. Section 3.1 outlines the main requirements that were defined after the initial problem analysis. Section 3.2 provides an overview of how Connect integrates with existing technology architectures. Section 3.3 provides an overview of FenixEdu Connect's architecture while section 3.4 focuses on the implementation details of the proposed solution.

3.1 System Requirements

After an initial analysis of the main issues that surround IAM systems at Higher Education Institutions (HEIs) in general and Instituto Superior Técnico (IST) in particular, which included valuable insights from the IST's IT Services team, the following requirements for the proposed solution were defined:

- Architectural
  - **R1.1**: Modular design, ensuring the separation between the core logic and additional components through clear interfaces and component registries (plug-in architecture).
  - **R1.2**: Be Extensible. Each implementing HEI should be able to customize as much of Connect's functionality as possible without compromising the core module's logic. Features should be designed to allow for future extensibility whenever possible.
  - **R1.3**: Use Generic concepts. It should not be tightly coupled with any specific service, even from the remaining FenixEdu ecosystem.
  - **R1.4**: There should be a clear separation between the backend and frontend, with communication between the two layers occuring in a stateless way, through HTTP requests.
  - **R1.5**: The backend should persist its information in a relational database which ideally should not be constrained to any given provider.
- Authentication:
  - **R2.1**: Provide authentication for the other applications in the organization, using a claim transfer media that allows clients to validate its validity without having to contact the Connect

instance.

  - **R2.2**: Support for multiple primary authentication providers, with enough built-in options to deploy the system standalone.
  - **R2.3**: Support for two-factor authentication with built in implementations for the most common providers. Users should be able to manage their own authentication factors.

- Identity Management:
  - **R3.1**: Connect should be able to aggregate the profile information for a given user from multiple systems in the organization and present it according to the requesting party's level of privilege.
  - **R3.2**: There should be a flexible account management system that allows administrators to perform maintenance actions on any user account.
  - **R3.3**: It should be possible to create user accounts ahead of time with 0-day provisioning (invite system).

- Access Delegation:
  - **R4.1**: The ability to act as an OAuth authorization server, with the option to also be a resource server. It should provide a central location for registering OAuth applications and allow developers to manage their applications.
  - **R4.2**: Support for authentication over OAuth (OpenID Connect).

- External Integrations:
  - **R5.1**: It should be possible to base Connect on an existing Lightweight Directory Access Protocol (LDAP) directory for account synchronization with legacy systems.
  - **R5.2**: It should be possible to inject authentication filters at runtime, that are executed after every successful authentication transaction, similarly to Auth0's Rules system.
  - **R5.3**: Connect should expose a SAML IdP interface for integration with identity federations.

- Security:
  - **R6.1**: Signing keys should be generated and automatically distributed in a secure way that allows them to be accessible for multiple application instances.
  - **R6.2**: Users should be able to access a history of relevant security events pertaining to their account (such as granted authorizations, failed login attempts or password changes).
  - **R6.3**: It should be possible to manage individual sessions, incluiding remotely terminating any open session.

- Monitoring and Auditing:
  - **R7.1**: It should be possible to audit any resource managed by the Connect system and trace back every change made over its lifetime.

- **R7.2**: It should support multiple logging facades, allowing it to be selected at deployment time.

3.2 Integration Goals

FenixEdu Connect tackles the challenges of managing the vast digital identities of the academic community in higher education institutions while also providing a centralized authentication gateway for all applications in the organization. A complete implementation of Connect within an organization's infrastructure would naturally place it in the role of the authoritative source for user information with every application querying it through one of the supported protocols. While this may be possible in new deployments or in the case of organizations where applications are mostly open source this is rarely the case in established Higher Education Institutions where the use of proprietary software can be a deterrent to the adoption of this solution. Connect was thus designed for ease of extensibility, allowing each organization to develop the necessary logic to adapt it to its necessities, rather than focusing on delivering a one-size-fits-all solution which would, undoubtedly, leave out features that could be of significant importance to some organizations.

Section 4.2 provides a case study for the deployment of FenixEdu Connect at Instituto Superior Técnico, highlighting the necessary development efforts that would be required to support this IAM solution along with a comparison with the existing solutions.

3.3 Architecture

3.3.1 Overview

FenixEdu Connect follows a traditional client-server architecture whereby processing is divided between two sets of entities: the servers, which host all of the system's resources and the clients, who request access to a resource or service hosted by the servers. More specifically, it follows a three-tier architecture, organizing the applications's responsibilities between three separate layers:

1. **Presentation**: The topmost layer, responsible for displaying information to the end user. This layer is commonly known as the application's frontend.
2. **Application**: Where the application's services and business logic run.
3. **Persistence**: The bottom layer, responsible for providing data access and abstracting the application layer from the persistence mechanism used to store it.

One of Connect's design requirements was to enforce a clear separation between the frontend (the presentation layer) and the backend (the remaining two). While it would be possible to develop these two modules as part of the provided work it was decided to drop the frontend module completely as part of the FenixEdu Connect solution and this thesis. The rationale behind this decision is twofold:

- Higher Education Institutions often have specific design systems in place to ensure the coherency of their applications' interfaces. While the FenixEdu ecosystem of applications takes this into account by providing extensive support for theming there are still limitations in the way the UI can be customized.
- Connect's extensible approach allows each institution to tailor the implemented solution to its individual needs. These extensions often require additional interfaces to be designed. Constraining this interfaces to a limited set of layout options could be a significant challenge to their deployment. On the opposite side, organizations may wish to restrict specific features (such as not offering support for a built-in authentication factor, like SMS codes) and should be able to not include these features in the User Interface (UI).

The provided solution, as part of the FenixEdu family is, therefore, restricted to the backend module, which is to be maintained by the FenixEdu team and centrally versioned. Each organization should then implement a frontend module capable of supporting its specific design guidelines and and business requirements. Decoupling these two layers allows for the independent development of each one. Organizations can then easily update the backend module to add a new feature or pull a security patch from the FenixEdu team without disrupting the frontend layer.

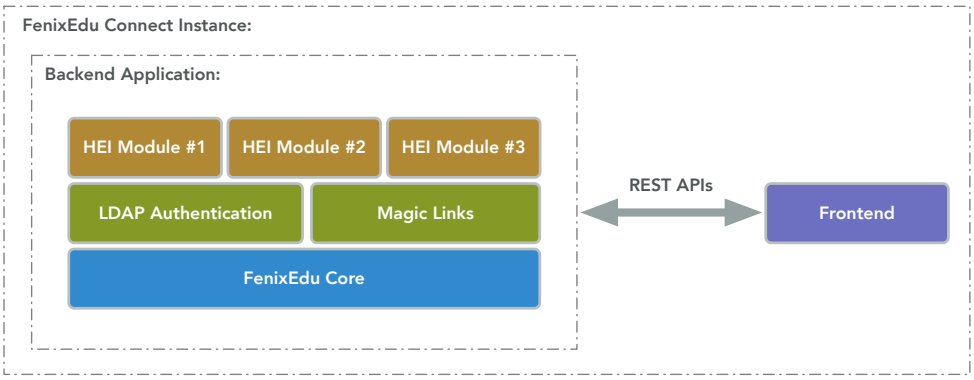


Figure 3.1: A breakdown of the modular architecture of FenixEdu Connect. Brown modules are specific to the client institution while the green modules are generic.

Figure 3.1 provides an overview of this architecture. The core module provides a number of extensibility points to be used by each organization. Being an open source application, it becomes possible, for exemple, to have an organization pull in an authentication module developed at another institution. In the depicted situation, the Connect deployment uses two modules obtained from the open source community and three modules developed in-house to make up the final backend application. The frontend was naturally implemented by the organization, with communication between the two components occurring through well defined REST APIs.

Redundancy is a major concern for any system that is part of the core IT infrastructure of an organization. FenixEdu Connect can easily scale vertically (by increasing the computational resources it can access) or horizontally (by replicating it across multiple nodes). Since the core module operates in a stateless way it can easily be replicated across as many instances as needed to suit the performance requirements of the organization.

3.3.2 Backend

The implemented solution is a Java application, built on top of the Spring framework. Spring is a free and open source framework for the development of Java applications based on the Inversion of Control (IoC) design pattern. IoC differs from the traditional design methodologies by inverting the call flow between the application's logic and its supporting libraries: instead of having the application's own code call into its dependencies it is the underlying application framework that calls the application code in specific events. The Spring framework relies on dependency injection, whereby an object is responsible for providing (or injecting) the required dependencies of another object for implementing its IoC container. The separation between a client's dependencies and its behavior allow for a loosely coupled architecture where components can easily have their implementation replaced without breaking changes.

The Spring initiative offers more than 20 libraries in active development, accelerating the development of Java applications by providing solutions for configuration, persistence, security, enterprise integrations or web services. Since Connect has been built on top of some of these projects, the remainder of this section will provide an overview of the most relevant ones and how they enabled the development of the features described in later sections.

The presentation layer is mostly handlded by Spring's *MVC* framework, which provides the necessary servlet infrastructure to implement a Model View Controller (MVC) pattern. Figure 3.2 provides an overview of the HTTP request/response flow implemented by Spring MVC:

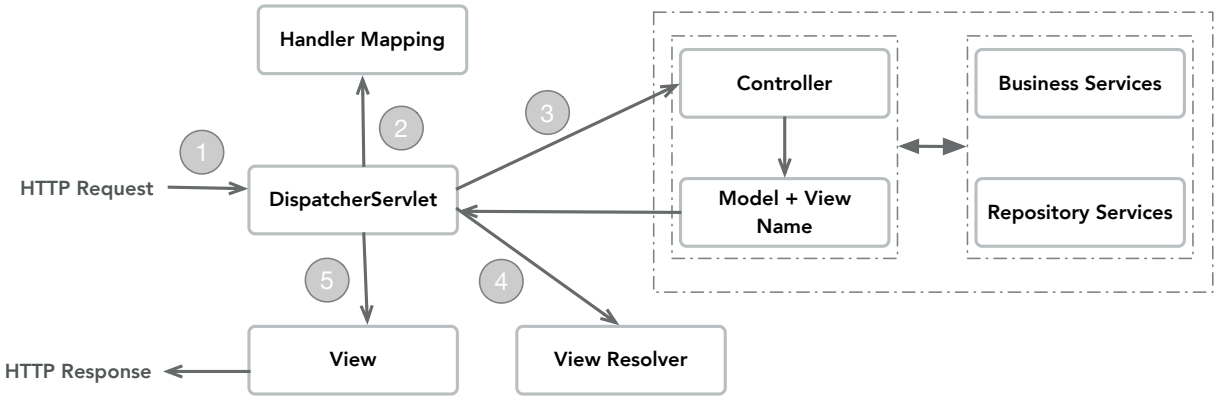


Figure 3.2: Overview of the request handling flow implemented by Spring MVC.

1. Incoming HTTP requests are handled by the front facing `DispatcherServlet`.
2. Controllers register the set of paths they should handle in the `HandlerMapping` component at application startup. This component is then queried for the `Controller` component that matches each incoming request and returns it to the `DispatcherServlet`.
3. The controller is called with the incoming request data. Spring provides automatic type conversion between the HTTP body and simple Java objects (often referred to as Plain Old Java Objects (POJOs)). The controller accesses the business layer and returns a `ModelAndView` object, encompassing the name of the view that should be rendered and associated model data.

4. The `ViewResolver` is queried for the view with the specified name.

5. The returned view is rendered, and the resulting data is sent back as an HTTP response.

While the core component only exposes REST endpoints, where the use of a `ModelAndView` object is not required, Spring MVC also provides framework support for this type of controller, including the automatic serialization of model objects to JSON or XML.

For persistence Spring offers the the Spring Data JPA library as a wrapper for the Java Persistence API (JPA) that offers repositories with basic Create-Read-Update-Delete (CRUD) methods out of the box, as well as the automatic generation of queries from user defined interfaces. Hibernate[1] is an Object-Relational Mapper library that is often used with Spring Data as a JPA implementation. In addition to abstracting the developer from the persistence provider (allowing it to be defined for each deployment) these libraries often offer performance improvements, such as multiple cache levels or the ability to lazily load objects.

Security is another major concern for every application. The Spring Security framework aims to ease the challenges of implementing authentication and access control in Spring applications. Since security is often a cross-cutting concern, Spring Security is not focused on any specific component providing, instead, a global umbrella over the presentation and business layers. These features range from basic support for multiple authentication providers to optional integrations with Spring MVC which allow controllers to directly specify access-control rules through an internal expression language. Spring's authentication system is designed for extensibility, with developers being able to easily inject their own logic into the base scaffolding provided by the framework.

Developing Connect on top of an already existing security framework is not without its challenges. While Spring's frameworks are often easy to extend, there are always specific scenarios in which the existing extensibility points are not enough to implement the required logic. However, relying on a mature, well-tested framework greatly reduces the attack surface that would otherwise have to be audited for security risks. Being an widely-used open source framework ensures that security vulnerabilities are reported and fixed in a timely manner.

Figure 3.3 provides an overview of Spring security's authentication flow.

1. The user's HTTP request is filtered through the servlet filter chain until it hits an Authentication Filter.

2. The `AuthenticationFilter` is responsible for converting the request to an object that implements the `Authentication` interface. This is usually accomplished by parsing the request and extracting the credentials that were passed in (which can either be a token in the authorization header, username and password parameters, a key in the request's body or some custom credentials schema). Spring provides some built-in implementations, such as the `UsernamePasswordToken` which holds a pair of username/password credentials.

[1] https://hibernate.org/

3. The authentication object is forwarded to the `Authentication Manager`, which acts as a centralized container for the multiple authentication providers that might be available in the application. The manager queries each registered provider to check for support for the incoming authentication type. Providers that support the object are asked to attempt to authenticate it.

4. Each provider implements the internal logic required to authenticate the principal from the information available in the incoming authentication object.

5. Providers may query a `UserDetailsService` to obtain additional information for a given user id (such as retrieving its password) before issuing an authentication response.

6. If no exceptions were thrown, an authentication object is forwarded up the chain until the initial Authentication Filter.

7. The filter then sets the received authentication object in Spring Security's Security Context, which marks the user as authenticated.

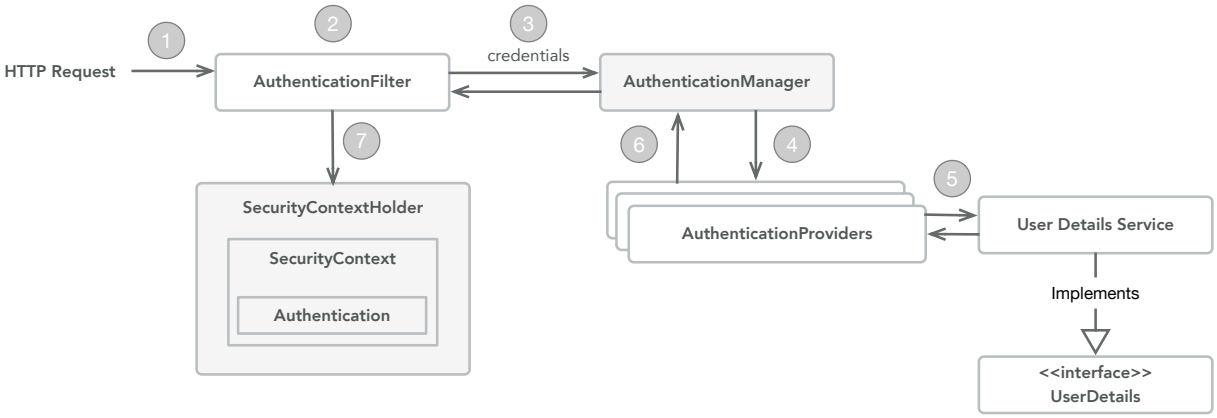


Figure 3.3: Overview of the authentication flow implemented by Spring Security.

While using the wide array of frameworks provided by the Spring initiative results in a significantly faster development cycle there are some concerns which must be addressed beforehand. Each framework relies on a set of dependencies, some of them pinned to specific versions. Managing these dependencies while ensuring that version conflicts do not arise can be difficult. Additionally, since Spring frameworks are designed to be applicable to a wide array of scenarios they often require significant configuration before they can be used. The Spring Boot project aims to alleviate this issues by taking an opinionated approach to dependency management and attempting to auto configure the integration between the different spring frameworks. One of the most relevant examples is the automatic configuration of an embedded Tomcat server, which is included in the Java Archive (JAR) files generated by Spring Boot. The application can thus be run anywhere with a simple `java -jar` command. Connect takes advantage of Spring Boot to accelerate the development times and simplify dependency management.

3.4 Implementation Details

Now that the overall architecture has been defined, this section will focus on describing in further detail the main features of the proposed solution, along with additional details on how they were implemented.

3.4.1 Authentication

Authentication is at the foundation of every identity management system. It is the process under which a previously registered user is able to establish her identity by presenting one or more sets of credentials. Credentials can come in many forms, with most usually grouped in three types [36]:

1. **Knowledge factors**: requiring the user to provide a set of information that only she knows (i.e. a username/password combination or a PIN code).

2. **Ownership factors**: requiring the user to provide something that only she should be in possession of, such as a hardware key or a code sent to the user's mobile phone.

3. **Inherence factors**: requiring the user to provide some form of personal trait identifier, such as a fingerprint or an iris scan.

The use of a single authentication component, generally known as a single-factor authentication has been proven to not provide enough protection against malicious attempts to access secured resources, especially in high-security contexts where personal data is at stake [37]. To alleviate this issue, experts have been advocating for the use of multiple authentication components, ideally through the combination of at least two factors from different groups. This is commonly known as Two-Factor Authentication. The implemented solution separates the authentication flow into two consecutive flows: primary authentication and secondary authentication.

3.4.1.1 Primary Authentication

Connect defines primary authentication as the first (and mandatory) step in establishing a user's identity. The implemented solution takes an opinionated approach to the type of factor that should be used for this step by shipping a complete solution for username and password credentials (a knowledge factor). Nevertheless, the final implementation provides a high degree of provider flexibility, allowing other types of authentication factors (such as a smart card) to be used for primary authentication.

While Spring Security already provides a very significant authentication API with support for multiple authentication providers it clashes with Connect's design philosophy of a solution centered around easy extensibility through the addition of small modules that allow each HEI to tailor the system to its own preferred authentication provider. Spring's authentication APIs were designed to be easy to customize, with less regard for their usage in a modular application. To overcome this limitation, Connect proxies Spring Security's `AuthenticationProvider` by implementing its interface in the `DelegatingAuthenticationProvider` (DAP) class, which is the only authentication provider that gets registered with Spring's

Security Manager. The DAP is responsible for maintaining a set of registered authentication provider instances, which can be added at any point of the application's lifecycle.

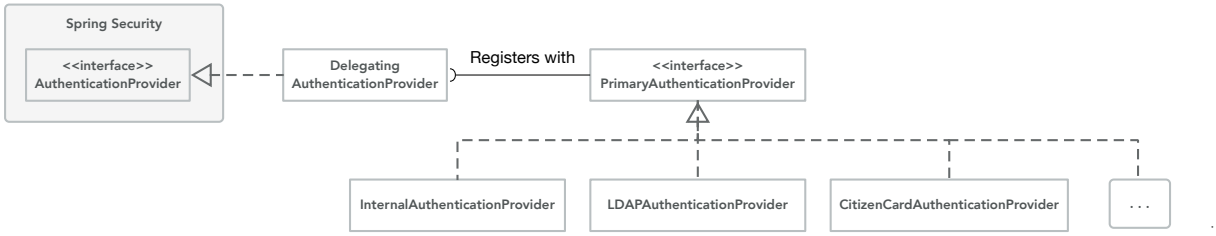


Figure 3.4: A UML representation of the primary authentication provider classes and their interaction with the infrastructure provided by Spring Security.

The `PrimaryAuthenticationProvider` abstract class is a lightweight wrapper for Spring Security's `AuthenticationProvider` interface that defines a set of common logic to be available for every authentication provider, such as the ability to toggle a provider on/off at runtime. New authentication providers can be implemented by simply extending the abstract class since a servlet initializer hook will take care of registering it with the DAP automatically on application startup.

While Connect supports multiple authentication providers they are only expected to process objects conforming to Spring's `Authentication` interface. Connect implements a built-in entry point for authentication credentials by exposing a RESTful endpoint at `/api/auth` that consumes a JSON object with `username` and `password` fields and takes care of parsing and translating the input object into a valid Authentication object that can be processed by the registered providers. Additional providers that require credentials to be supplied in other formats (such as an endpoint with a different URL or consuming other content types) will need to implement it by extending Spring's `AbstractAuthenticationProcessingFilter`.

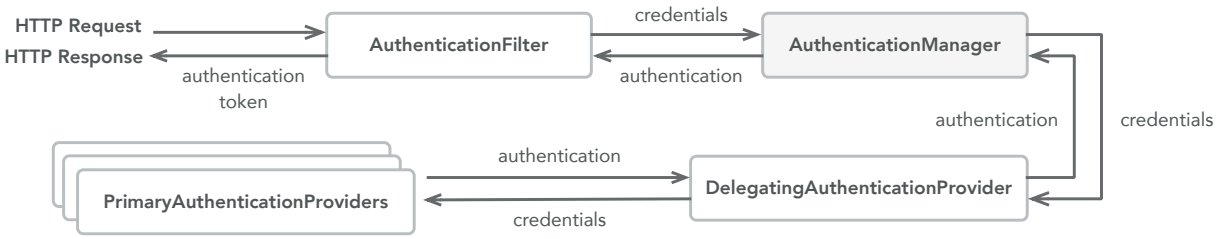


Figure 3.5: An overview of the authentication flow using the `DelegatingAuthenticationProvider` component.

Figure 3.5 provides an overview of the primary authentication flow and how it interacts with Spring Security's infrastructure. Multiple implementations of the `AbstractAuthenticationProcessingFilter` class can be registered in Spring's filter chain. They provide an entry point for a given type of credentials and are responsible for translating them into `Authentication` objects. These objects are then passed on to Spring's Authentication Manager which is responsible for querying the registered authentication providers and returning an authenticated principal. Since the DAP is the only provider registered with Spring's Security Manager this is the only class to be queried. The DAP will, in turn, forward the au-

thentication request to all of its registered providers that support it. Whether a valid authentication is accomplished or an exception has been thrown, the result is carried upstream, with the DAP informing the authentication manager of its decision. The manager then returns the result to the `AbstractAuthenticationProcessingFilter` object that initiated the call which is then responsible for returning an appropriate response (either an error message, or a some type of credential representing that session).

Authenticating a user is a two step process, with the authentication provider asserting whether or not the user's credentials are valid and the authentication filter then using this decision to issue the credentials. This separation of concerns allows Connect to provide different types of credentials for each situation. An organization wanting to extend Connect to authenticate users in its 802.11 network could, for instance, implement a simple RADIUS[2] server that delegated authentication to Connect's providers.

One of the possible types of credentials granted by Connect (in fact, the only one implemented by the provided solution) is a JSON Web Token. A successful authentication results in a JWT being issued by the authentication filter. Listing 2 provides an example of the body (or claim set) of a JWT issued by Connect. The `iat` and `exp` represent, respectively, the token's issue and expiration dates. `jti` is the unique identifier for the token. `sub` asserts the token's subject, which can either be a user or an OAuth application. The `aud` field lists the services which should accept this token. Finally, the `authorities` field asserts the user's roles and permissions.

```
{
  "iss": "fenixedu-connect",
  "iat": 1533897834,
  "exp": 1533919434,
  "jti": "NB9IGug7sqExJVb8iKhy8xFdjbzVQRP4",
  "sub": "ist178401",
  "grant_type": "jwt",
  "authorities": [
    "ADMIN", "STUDENT", "GRANT_OWNER"
  ],
  "aud": "fenixedu-connect"
}
```

Listing 2: The body section of a JWT authentication token issued by Connect

When a user attempts to access Connect passing in a JWT in the Authorization header the `AuthenticationTokenFilter` intercepts the request and attempts to authenticate the principal. The JWT is extracted from the header, and its signature is validated against Connect's signing keys. If the token is valid, a `JWTAuthentication` object is created from the token's data and loaded into Spring's Security context.

Connect records all issued tokens along with some additional metadata (such as IP Address or user agent) in the form of an `AccessToken` object, which is persisted in the backend. This allows for an historical overview of a given user's sessions, and is made available in the `api/v1/users/{user}/authorizations` endpoint. Users can manually revoke a single token or all of the active ones. In both cases, tokens are added to a public revocation list which will be analyzed in section 3.4.5.

[2] https://tools.ietf.org/html/rfc2865

3.4.1.2 Secondary Authentication

While a user is considered authenticated after a successful primary authentication step (with a JWT being issued to assert the principal's identity) a secondary step is available to reduce the chances of a malicious attack on a user's identity.

Spring Security does not provide clear support for two factor authentication out of the box. To accomplish this, Connect provides a registration service, implemented in the `MFAManagementService` class which maintains a set of available 2FA providers and is responsible for returning the correct instance when a provider of a specific type is required.

Secondary authentication factors require a small set of operations to be available:

- **Create**: Create a new factor of a given type.
- **Activate**: Activate a previously created factor.
- **Start Verification [Optional]**: Initiate the verification process of a previously activated profile, if its verification process implements a two-step challenge/response.
- **Verify**: Complete the provider's challenge/response verification cycle.
- **Delete**: Remove the provider from the user's list of authorized factors.

The `AuthenticationFactorProvider` generic abstract class declares the five required actions that should be implemented in each of its subclasses. Since each subclass is only the provider for a given type of authenticator factor implementations are also required to bind the generic type to a specific implementation of the `AuthenticationFactor` interface.

Figure 3.6 presents an architectural overview of the MFA subsystem with the `MFAManagementService` component implementing a central repository of second factor authentication providers. Subclasses of the `AuthenticationFactorProvider` class implement the required logic to manage a given type of factor (i.e. to manage `AuthenticationFactor` objects). Similarly to the `PrimaryAuthenticationProvider` class, a servlet initialization hook automatically registers all subclasses of the `AuthenticationFactorProvider` class in the top level `MFAManagementService`.

Implementing a new type of 2FA is a two step process:

1. Extend the `AuthenticationFactor` abstract class to persist the required information for each instance of the authentication factor (such a seed value or a secret). Let this be `AuthFactorImpl`.
2. Extend the `AuthenticationFactorProvider` abstract class specializing it for the `AuthFactorImpl` type. This class should implement the logic required to perform the five main actions described beforehand on objects of `AuthFactorImpl`.

The core package provides built-in support for three of the most common secondary authentication factors:

- **Time-base One Time Password (TOTP)**: An algorithm that generates a one-time password from a secret seed value and the current time [38]. Implemented in the `TOTPAuthenticationFactorProvider` class.

Figure 3.6: A UML representation of the classes that implement the 2FA subsystem.

- **Universal 2nd Factor (U2F)**: An open standard for hardware security keys [39]. Implemented in the `U2FAuthenticationFactorProvider` class.
- **SMS verification**: Where a secret code is sent to a previously registered mobile phone. Implemented in the `SMSAuthenticationFactorProvider` class.

After a successful secondary authentication the user is issued a new JWT token with an additional `MFA_VERIFIED` authority, which can be used by Connect instances or resource serves to restrict access to some resources to users who were authenticated with two different factors. This is especially helpful to secure sensitive areas such as application administration portals.

3.4.1.3 Passwordless Authentication

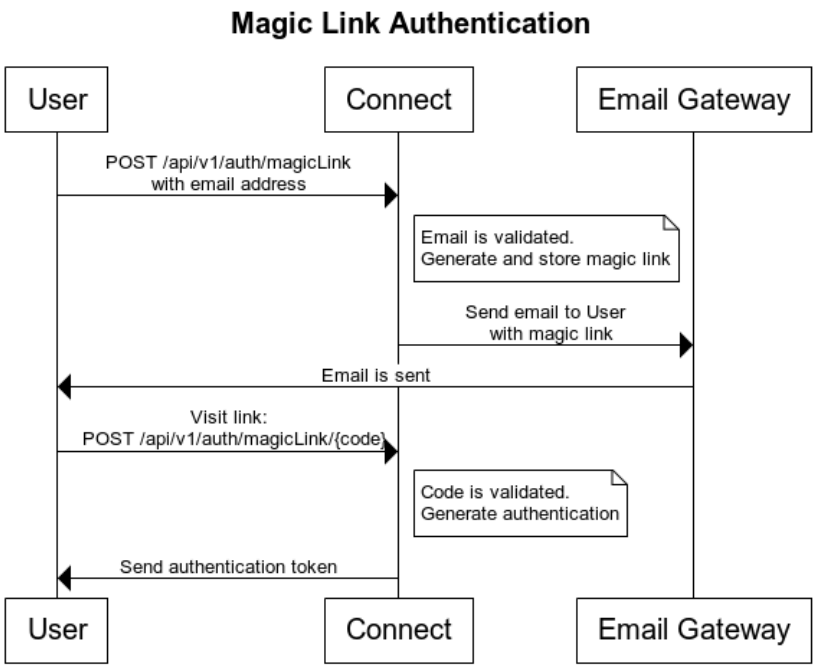
Web applications have traditionally resorted to username/password pairs to perform user authentication due to their inherent simplicity over other methods such as digital certificates. However, there are two main reasons why these systems often fall short on the security they provide. [40]

- When faced with the need to memorize multiple passwords, users tend to opt for using simpler combinations which get reused on multiple applications.
- Developers often fail to implement the necessary security policies when managing these credentials storing them in plaintext or using insecure hashing algorithms.

To alleviate these issues, a new type of authentication known as *Passwordless* Authentication has emerged. As the name suggests, *passwordless* authentication is based on the premise of using alternative authentication providers to avoid storing a user's password in the application's backend.

There are multiple strategies for *passwordless* authentication with one of the most common being the use of Magic Links. With this form of authentication, when a user wishes to access a protected system she is asked for her previously registered email address. The application checks its user database and, if a match is found, generates a unique token which is appended to a preset URL belonging to the application. This URL is then sent to the user's email address and the user is directed to check her inbox. When the user clicks on the link sent to her email it is exchanged by a regular authentication token, as if the user had logged in with her username and password combination. As long as the magic links have a short life span and the servers implement measures to prevent brute-force attacks, this type of authentication can provide a similar level of security to the regular approach.

The provided solution includes an optional module, `connect-magic-links`, that implements *passwordless* authentication through magic links. In addition, it also serves the purpose of acting as a template for how an organization could implement an additional authentication provider and use it to extend Connect's built in feature set.



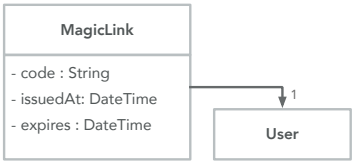


Figure 3.8: Magic Links UML diagram

Figure 3.7: A sequence diagram of the magic link authentication flow.

Figure 3.7 provides an overview of this authentication strategy. The user sends her email to the Connect instance, which queries the internal database through the `UserManagementService` service. If a user is found, a `MagicLink` object is created and persisted in the database. The constructor for this object generates a secure random string which, when appended to the path `/api/v1/auth/magicLink`

forms the complete magic link that is sent to the user's email. When Connect intercepts a magic link request, through the `MagicLinksAuthenticationFilter` it extracts the code from the path, attempts to retrieve the previously stored `MagicLink` object and generates a JWT for the user's session.

3.4.2 User Management

At the heart of every Identity Management system lies a directory responsible for maintaining user accounts and their associated data. The core module defines a `UserRepository` interface with a set of base methods required by its internal logic. An implementation of this interface is automatically generated by Spring Data JPA however, organizations are free to extend this interface to suit the requirements of additional modules, with Spring's dependency injection automatically using the extended version in place of the provided one. FenixEdu Connect limits the scope of the personal information it stores directly, delegating profile data to a set of external providers. As a consequence, the user model (shown in figure 3.9) is quite simple:

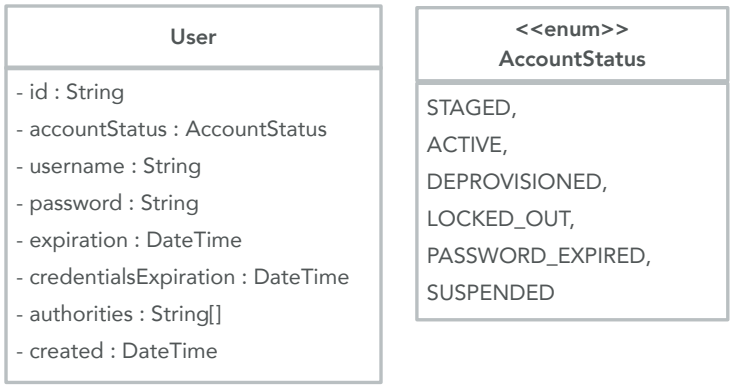


Figure 3.9: A UML representation of the User class.

In addition to the unique user identifier, credentials and account metadata (creation and expiration timestamps) there is only an additional field, which holds the set of user authorities. Authorities (often also referred as *roles*) represent a permission to perform a certain action or to access a protected resource. User accounts are created with an initial set of authorities, which may be updated over time to reflect changes in the user's permissions (i.e. students may have a `STUDENT` authority granted when first enrolling in the institution which grants access to some educational resources. Upon completing their degree, the authority is removed from the account, preventing them from accessing the same protected resources). Connect provides a way to define a hierarchical set of authorities, such that if a resource is protected with a required authority of `HELP_DESK_EMPLOYEE` it is automatically available to a user with a `SYSTEM_ADMIN` authority without it having to be made explicit (assuming a system administrator should be able to perform all the actions of an IT help desk employee).

3.4.2.1 Account Lifecycle

As users join, leave and progress within the organization their digital identities must be updated to reflect these changes in status. The set of possible states and the transitions between them are often referred

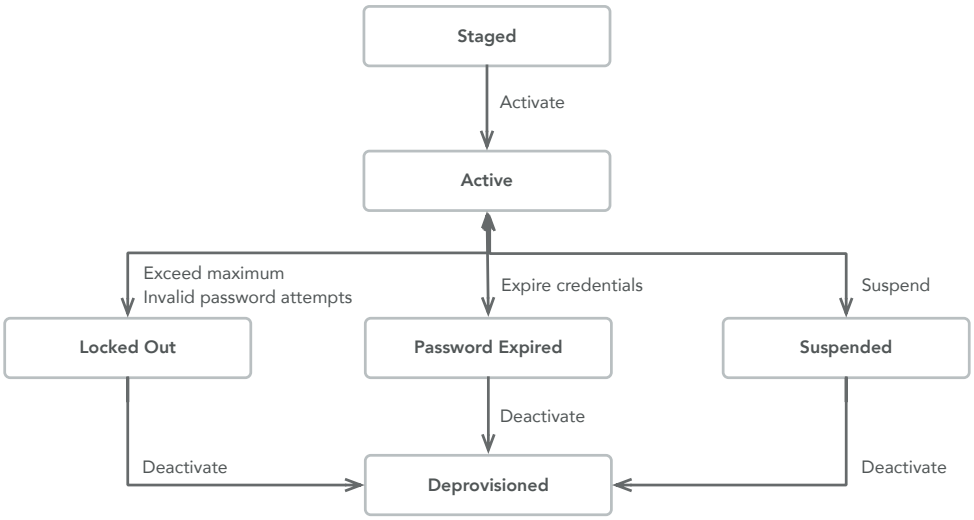


Figure 3.10: The possible account states and the lifecycle transitions between them.

to as the organization's account lifecycle. Figure 3.10 provides an overview of FenixEdu Connect's account lifecycle. Accounts can be created in any state, but they will usually start at either the `Staged` or `Active` states and progress through the remaining states through well-defined lifecycle events.

- **Staged**: The account has been created, but still requires activation before it can be used.
- **Active**: The account is active and ready to use.
- **Locked out**: The account has been locked for security reasons (e.g. the maximum number of invalid login attempts has been reached, or suspicious activity has been detected).
- **Password Expired**: The credentials have expired.
- **Suspended**: The account has been manually suspended by staff.
- **Deprovisoned**: The account has been permanently terminated.

FenixEdu Connect implements an invite system whereby an authorized user can send account invites, allowing external users to create accounts without the need for generating (and distributing) temporary credentials. The invited user will receive a short lived token that can be used to sign up to Connect. The account is not created until the user visits the link and selects her own credentials. In addition, when creating an invite it is possible to define the set of authorities to grant to the created account, which allows users to have access to allowed resources as soon as their account is activated. This is a clear improvement over the traditional systems in place at many HEIs where not only are invited users emailed a pair of temporary credentials as the required accesses are rarely created ahead of time, forcing them to request the necessary permissions after the account has been set up. Figure 3.11 illustrates the two flows.

While most account lifecycle transitions will be the result of user intervention or automatic events FenixEdu Connect exposes a complete RESTful API for account management, which naturally includes controlling the transition between lifecycle states. Authorized personal (such as IT Help Desk employees

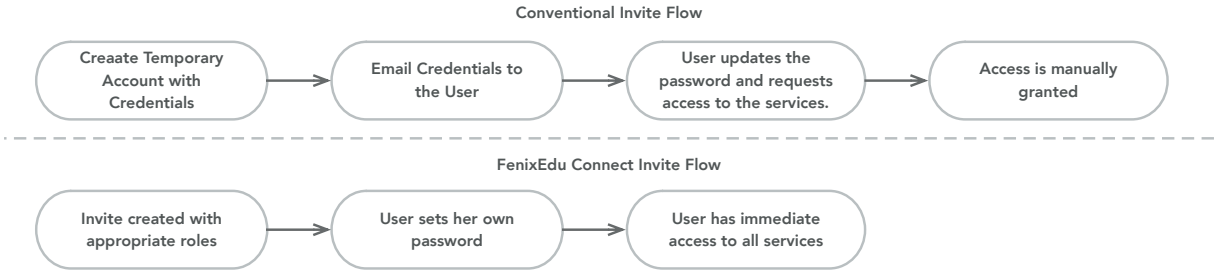


Figure 3.11: A comparison between the conventional invite flow and FenixEdu Connect's.

or system administrators) can easily suspend/unsuspend accounts, force credentials to expire, unlock accounts and reset passwords.

3.4.2.2 Password Reset

As previously discussed, password reset requests are responsible for a significant portion of the total volume of IT Help Desk support tickets. FenixEdu Connect implements a flexible system of self-served password reset strategies that allows implementing organizations to extend the built-in account recovery methods.

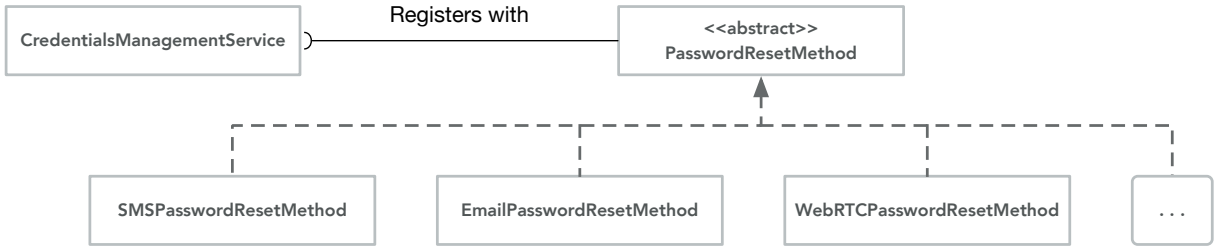


Figure 3.12: A UML diagram of the classes that implement the password reset service.

A central service holds a registry of the password reset methods currently in use. This is implemented in the `CredentialsManagementService` class. All valid methods should extend the `PasswordResetMethod` abstract class, which defines a simple interface to abstract the password reset behavior. Subclasses are automatically registered with the credentials management service at application start.

The core module defines a common entry point for all password reset requests, independent of the requested reset strategy. This endpoint consumes a JSON object whose only required field is the unique identifier of the reset strategy to use. From this information, the credential management service performs some initial validation checks (such as validating the account is in state that allows credentials to be recovered) and forwards the request to the appropriate `PasswordResetMethod` which is responsible for performing any additional steps that may be required. If the strategy requires a second step to be performed, it should generate and persist a `PasswordResetToken` which is automatically managed by the core module. An additional endpoint ensures that users are able to exchange the generated token for a new password.

Figure 3.13: An overview of the password reset flow.

Figure 3.13 illustrates this two-step process and how the request is handled internally. The user, Alice, wishes to reset her password and chooses the SMS recovery type. To accomplish this, Alice sends an initial request to the password reset endpoint with the specified payload. The request is forwarded internally to the `SMSPasswordResetMethod` service, which fetches Alice's phone number from her profile information and uses an SMS gateway to send her the recovery code. Alice then makes a second request to the appropriate endpoint passing in the recovery code (which authorizes the reset operation) and her new password.

3.4.2.3 User Profile Management

While it could be argued that the Identity Management system should maintain all the user's profile information in a self-contained fashion that is rarely the case. Higher Education Institutions, as most medium to large scale organizations rely on a multitude of applications responsible for maintaining semi-isolated clusters of information. These applications may either be the authoritative source for this data, or just consumers of some previously replicated dataset. In a FenixEdu ecosystem, for instance, FenixEdu Academic is the authoritative source for the user's completed courses, while FenixEdu Sotis can provide an authoritative answer regarding the user's ORCID number (a researcher identifier used in scientific publications). While Connect should be able to provide both pieces of profile information, it would be infeasible in most situations to refactor all the existing applications to store their user profile information in a Connect instance.

To overcome this limitation, Connect defines an additional level of abstraction, through the `User-`

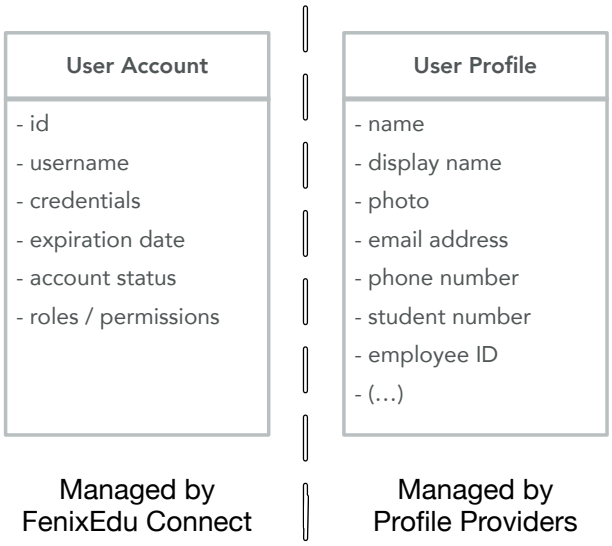


Figure 3.14: FenixEdu Connect enforces a clear separation of responsibilities between account data and profile information.

`ProfileProvider` interface. This interface defines the required set of methods to interface between the Connect instance and a second entity, responsible for maintaining the user's profile information. Each client institution can provide an implementation of this interface, through an additional module, to act as the interface between FenixEdu Connect and its internal applications that are responsible for managing user information. However, to facilitate the deployment of a Connect solution, a concrete implementation of this interface was included in the core module, through the `DelegatingProfileProvider (DPP)` class.

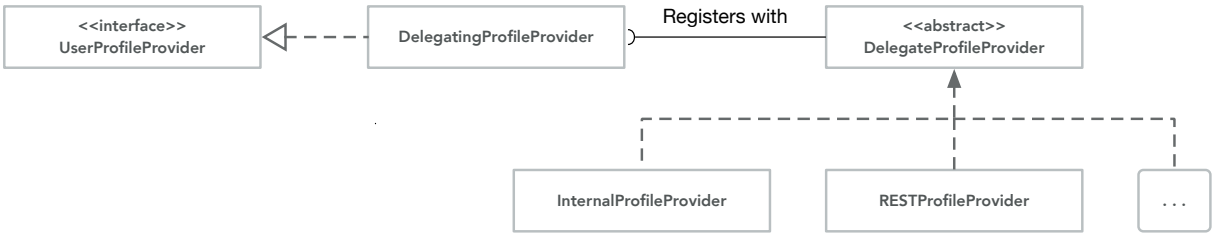


Figure 3.15: A UML diagram of the classes responsible for managing profile information in FenixEdu Connect.

The DPP service allows Connect to be used in an environment where applications want to maintain control over the user's profile information they are authoritative sources for. A set of delegates, extending the `DelegateProfileProvider` abstract class are registered with the DPP service. Each delegate is able to inform its parent service of whether or not it is an authoritative source for a given attribute key. When the delegating service receives a request for a set of profile attributes it demultiplexes it into sub-requests querying the delegates for the attributes they are authoritative for and assembling the final result. Figure 3.16 provides an overview of this flow. It depicts two concrete implementations of the `DelegateProfileProvider` abstract class:

- The `InternalProfileProvider` class provides a simple key/value store backed by a Spring JPA repository. This is the ideal solution for deployment scenarios where it is possible to import all the user's profile information into Connect, since it can be stored in the same relational database as the user's account data. In the event that this is not possible, an instance of this class can still be used to allow Connect to persist profile information that is not managed by any other application.

- The `RESTDelegateProfileProvider` aims to ease the deployment of profile providers in organizations with a strong presence of open source tools (such as the FenixEdu ecosystem). To accomplish this, Connect defines the specification for a simple API which, if implemented by the client applications, allows profile provider delegate instances to be automatically generated from only a few configuration properties (such as the client's API base URL and a shared secret). The specification for the client API is made up of two simple endpoints:

  - POST `/read`: Which consumes a JSON array of attribute names to retrieve. Listing 3 provides an example of a request to this endpoint.
  - POST `/write`: Which consumes a JSON object of attribute key/value pairs to set. An example is provided in listing 4.

  When Connect is requesting profile attributes on behalf of the user, the authentication token used when contacting Connect is forwarded to the profile provider as the authorization header for the `read` and `write` endpoints. This allows each provider to set its own security policies, i.e. define the set of roles/permissions required to access each attribute. When attributes are requested on behalf of another trusted application or Connect itself, the authorization header carries a JWT issued directly to FenixEdu Connect, in which case providers should ignore any additional security checks and provide the requested attributes.

```
{
        "userId": "ist123456",
        "attrs:" [
                "fenix:profile:name",
                "fenix:profile:gender",
                "fenix:profile:birthdate",
                "fenix:courses:completed",
                "fenix:degrees"
        ]
}
```

Listing 3: A request to a client's `/read` endpoint

The delegate interface provides a pluggable architecture for organizations to expose profile attributes from any application. HEIs can easily deploy a module to interface with a HRP system, such as SAP, to expose the user's employee number in Connect's profile information. In a traditional system, applications that require access to this attribute would be integrated with the HRP system, usually resorting to its public APIs. If this system was replaced or the APIs deprecated all the client applications would have to be updated to reference the new one. In a Connect-enabled environment only the profile provider

```
{
        "userId": "ist123456",
        "attrs:" {
                "fenix:profile:name" : "John Doe",
                "fenix:profile:gender" : "Male",
                "fenix:profile:birthdate" : "01-01-1970",
                "fenix:profile:emails": [
                        "john.doe@doe.org",
                        "jdoe@tencico.ulisboa.pt"
                ]
        }
}
```

Listing 4: A request to a client's `/write` endpoint

module would have to be made aware of the change for all client applications to continue to have access to the employee number attribute.

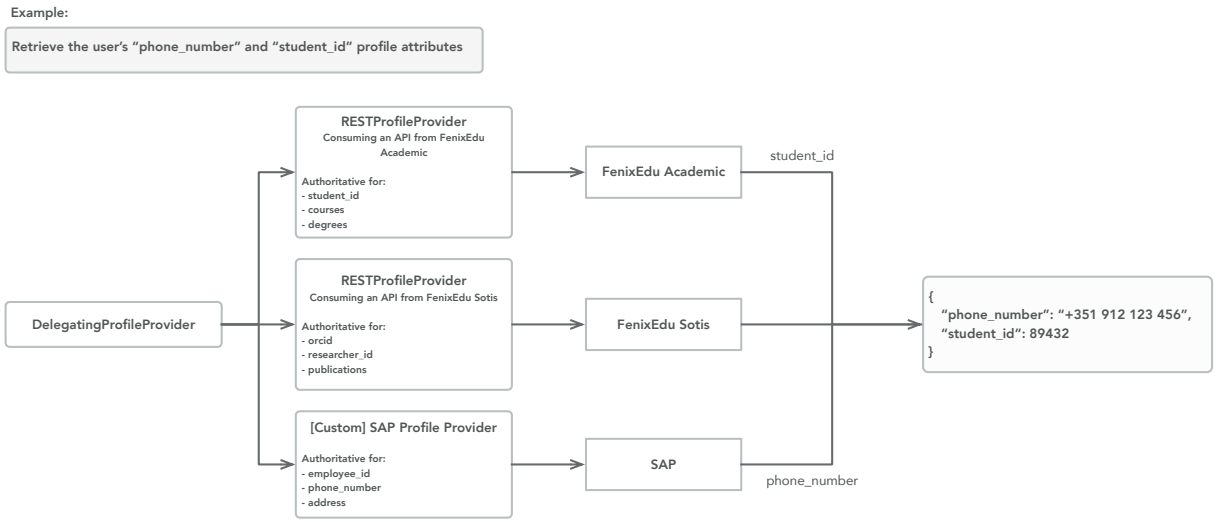


Figure 3.16: An example of how multiple profile providers can be used to aggregate user information and expose it through FenixEdu Connect.

3.4.3 OAuth Authorization Server

In addition to offering Identity Management features Connect's design philosophy rests on it simultaneously acting as a central point for the registration and authorization of OAuth applications. To achieve this, Connect relies on the foundations offered by the Spring Security framework to implement an OAuth Authorization Server. Figure 3.17 documents the domain model for the OAuth authorization server features. The `OAuthApplication` class is used to persist the relevant information for the registered OAuth applications including basic metadata, credentials, authorized grant types and required scopes. A distinction is made between required and optional scopes, allowing developers to fine-tune the requested permissions to the essential resources while still allowing users to opt-in for providing additional information. The `OAuthScope` class represents an OAuth Scope with an option to restrict it to applications created by users with an administration role. This allows for the use of OAuth for internal applications, as

regular users will not see administrator-only scopes and thus, will be unable to create OAuthApplications which require these scopes.

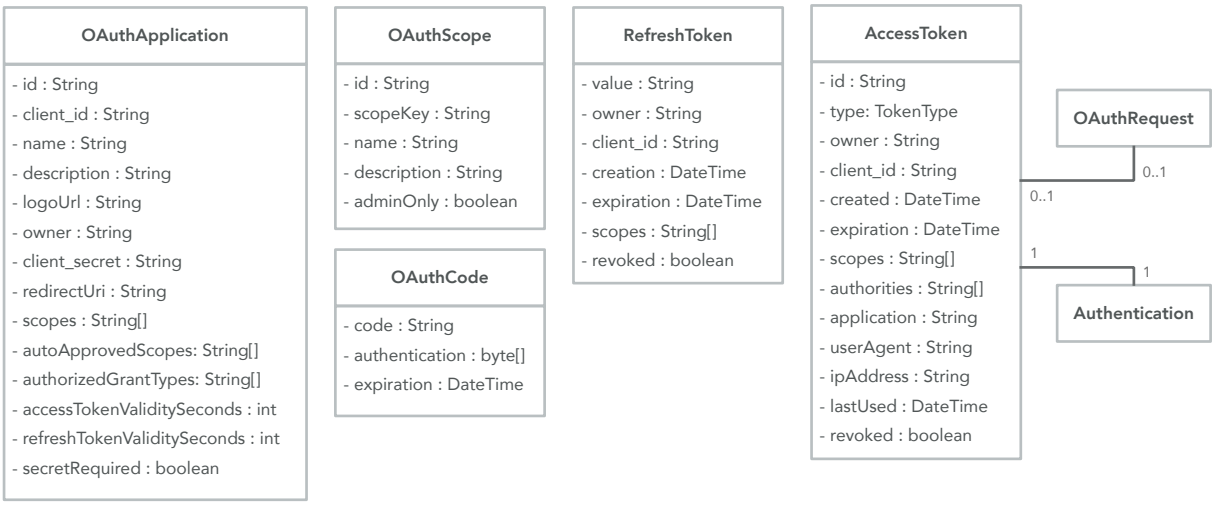


Figure 3.17: An UML diagram of the classes that support the OAuth implementation in FenixEdu Connect.

As previously discussed in section 2.2.1, the OAuth 2.0 Core Request For Comment (RFC) does not specify how the client applications should perform their initial registration in the authorization server. While a standard has been recently published [41] to provide a protocol for dynamic client registrations FenixEdu Connect only implements support for a subset of the required client metadata. There is, nevertheless, a complete API for client application registration allowing for CRUD-like operations on OAuth Applications and scopes.

The OAuth 2.0 Core RFC only specifies two endpoints as required for any protocol implementation, usually referred to as the authorization and token exchange endpoints. When using some grant types, such as the `authorization_code`, the resource owner's user agent interacts with the authorization endpoint, where the user is asked to provide a valid set of credentials and authorize access to the client application before an authorization code can be issued. This step naturally involves the UI layer, both through the authentication form (if the user isn't yet authenticated), the authorization step and the final redirect to the client application. This contradicts the main architectural decision of restricting the core module's exposure and forcing all communication to happen over RESTful endpoints. While Spring Security provides ready-made OAuth authorization and token exchange endpoints, a significant part of their logic had to be refactored to maintain this purely RESTful interface while still allowing the frontend to perform the necessary steps of the authorization flow.

Figure 3.18 provides an overview of the original flow, offered by Spring Security, and the resulting version, with the frontend and core module working together to establish a coherent flow while still only resorting to RESTful calls for communication between the two layers.

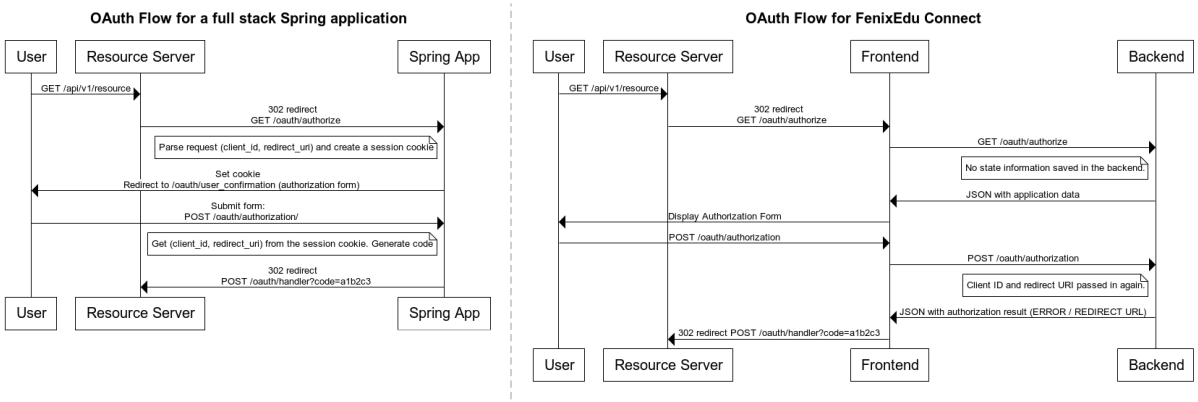


Figure 3.18: A comparison between the conventional OAuth authorization flow and FenixEdu Connect's.

3.4.3.1 Access Tokens

The OAuth 2.0 RFC intentionally left out a specification for how access tokens should be formed, what information they should contain and how they should be used. While some of these issues have been addressed in a later specification [42], which defines how tokens should be sent in an HTTP request there is still some ambiguity on how they should be constructed with some implementations relying on opaque strings (used to reference some authentication details previously stored) and others opting for sending all the required information in the token itself, along with some type of signature for verification. Spring provides an implementation of the former by providing a built-in solution for opaque access tokens. However, Connect's architecture relies on the use of JWTs whenever possible as a way to decouple the verification of the issued tokens from the authentication server. To achieve this, it takes advantage of a feature offered by the Spring Security framework that allows developers to enhance the opaque access tokens issued by the underlying OAuth implementation to convert them to signed JWTs with additional information on the authenticated principal.

The issued access tokens contain information on the authenticated user (the subject), the client application for which the token is issued (known as the token's audience) and the scopes whose access has been granted by the authenticated principal. Since all the information is contained in the token itself, client applications can both validate the token's authenticity and access the authenticated user's identifier and allowed scopes without having to contact the authorization server.

The use of opaque strings for access tokens requires authorization servers to implement a protocol that allows client applications to validate if a given token is authentic, that it hasn't been revoked and to fetch the list of authorized scopes. Once again, this step was left out of the original OAuth 2.0 RFC. A later specification proposed the creation of a token introspection endpoint implemented by authorization servers to allow protected resources to check the active state of access/refresh tokens and to obtain additional information about them [43]. This endpoint has been designed to bridge a specification gap in the implementation of opaque tokens and is unlikely to offer significant advantages when JWTs are used as access tokens, since the endpoint will simply return the same information that was already included in the token itself. Nevertheless, Connect implements an introspection endpoint as a fallback

verification strategy for client applications that are unable to deploy the necessary logic to validate and decode the issued JWTs. Instead of periodically refreshing the signing keys used by the Connect server and validating the access tokens' signatures against them these applications can simply send the token to the introspection endpoint and parse the resulting JSON response. This emulates the behavior of opaque tokens at the expense of nullifying the main advantage of using JWTs as access tokens.

Since this validation strategy may result in unnecessary load to the Connect servers (with developers abusing the introspection endpoint when local validation was possible) client organizations can easily opt to remove this feature.

3.4.3.2 OpenID Connect Provider

In addition to acting as an OAuth 2 authorization server for access delegation Connect also implements an OpenID Connect provider, allowing client applications to authenticate users using the same OAuth flow that was already being used for API authorization. As discussed in section 2.2.3, OIDC introduces a reserved `openid` scope which, when requested by a client application, causes the OAuth authorization server to treat it as an authentication request. The OAuth response should include, in addition to the access and optional refresh tokens, an ID Token asserting the authenticated user's identity. This is the only modification to the regular OAuth authorization flow. As such, when access to the `openid` scope is granted, Connect leverages the token enhancer that was already put in place to convert the opaque access tokens generated by Spring Security into signed JWTs to inject the additional `id_token` field in the resulting OAuth response. Figure 3.19 provides an overview of the enhancement flow.

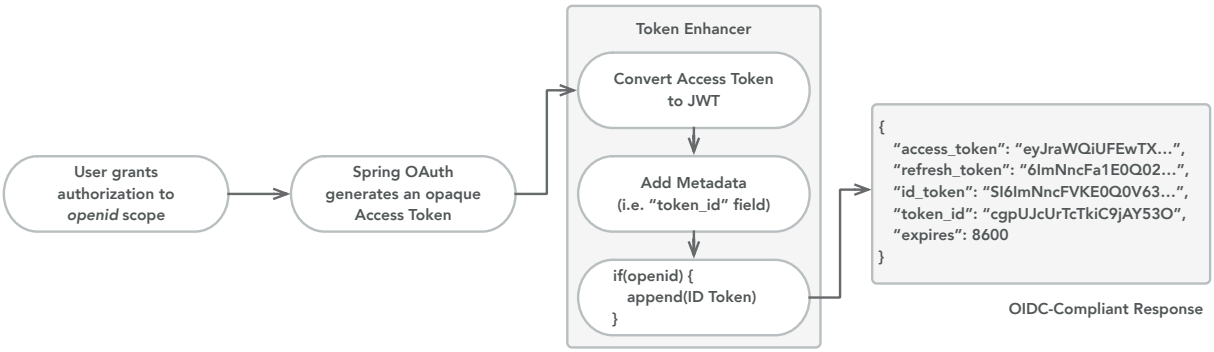


Figure 3.19: An overview of the OAuth token enhancement flow for OIDC requests.

3.4.4 Security

3.4.4.1 Key Management

Connect's reliance on JWT as the transport container for its authentication assertions requires a careful approach to key management. The JSON Web Signature specification allows for the use of both symmetric (HMAC) and asymmetric (ECDSA and RSA) algorithms in token signatures. However, the use of symmetric tokens would present additional challenges in key generation and distribution when deploying

new client applications (known as resource servers). An additional protocol would have to be designed to exchange symmetric keys between the Connect instances and the applications that consume its tokens. As a consequence, it was decided to restrict Connect to only use asymmetric algorithms with the final product supporting two main classes: Elliptic Curve (EC) and RSA.

At any given point, Connect maintains a set of valid signing keys which are made available to a trusted set of applications, described in section 3.4.5.1. One of those keys is considered the active key and is used to sign the issued tokens but any token signed with a key from the valid keys set should still be accepted by the resource servers.

Due to Connect's distributed nature, which allows it to scale horizontally, instances must store the Public and Private Keys that are used to sign the tokens. This is accomplished by persisting the Private Keys in Connect's database. To reduce the risk of exposure in case the database is compromised, the Private Keys are persisted encrypted using, by default, AES-CBC with 128-bit keys. The encryption keys are provided in the application's configuration files and multiple keys can be specified. The `KeyEncryptionService` encompasses all the logic required to create, validate and securely fetch the keys used to sign the issued JWTs.

To create a new key, a system administrator selects the desired algorithm, key size and a display name. Connect generates the key pair and searches the configuration files for the encryption key to use to encrypt the newly generated private key. The configuration properties follow a known pattern of the type `connect.key.password.<keyName> = <keyPasswordData>`. A `ConnectKey` object is then created to hold the key pair and associated metadata including the name of the key used to encrypt the Private Key.

Whenever a valid key is required to perform a signature operation the process is reversed. The `KeyEncryptionService` searches the persisted `ConnectKey` objects for the active one, attempts to retrieve its associated encryption key from the application's configuration properties, decrypts the key and hands it to the JWT builder.

System administrators have the ability to manage Connect's signing keys through a set of API endpoints. These allow for the creation of new keys, revocation of an existing key or setting the current active key.

3.4.4.2 Security Events

As the core component for the users' account security FenixEdu Connect must be able to provide users with an historical overview of the events that may have had a security impact on their accounts. These are typically associated with changes in the user's profile data or credentials. An attacker who was able to obtain temporary access to a user's account could, for instance, register a secondary authentication factor that was in her possession without the user ever becoming aware. Beyond the regular logging and auditing information, which is only available for system operators, Connect keeps a special record of these actions, which is made available to the user through the `api/v1/users/{user}/securityEvents` API endpoint. Users are able to query the endpoint to get a complete list of security events pertaining to their account, or limit the scope to a single type of event or date.

To implement this feature, Connect leverages Spring's built-in support for signals. Figure 3.20 provides an overview of the global flow. Connect services, such as the `UserManagementService` or the `MFAManagementService` create `AccountAuditEvent` objects whenever one of these key account actions occurs. At the same time, they forward these newly created objects to Spring's application event system, which takes care of distributing them for all the registered listeners. External modules can opt to listen to these signals but, in any case, Spring provides a built in listener that consumes all signals and stores the event objects in a known repository. Connect's security events API endpoint simply consumes this repository and makes the events available to the user.

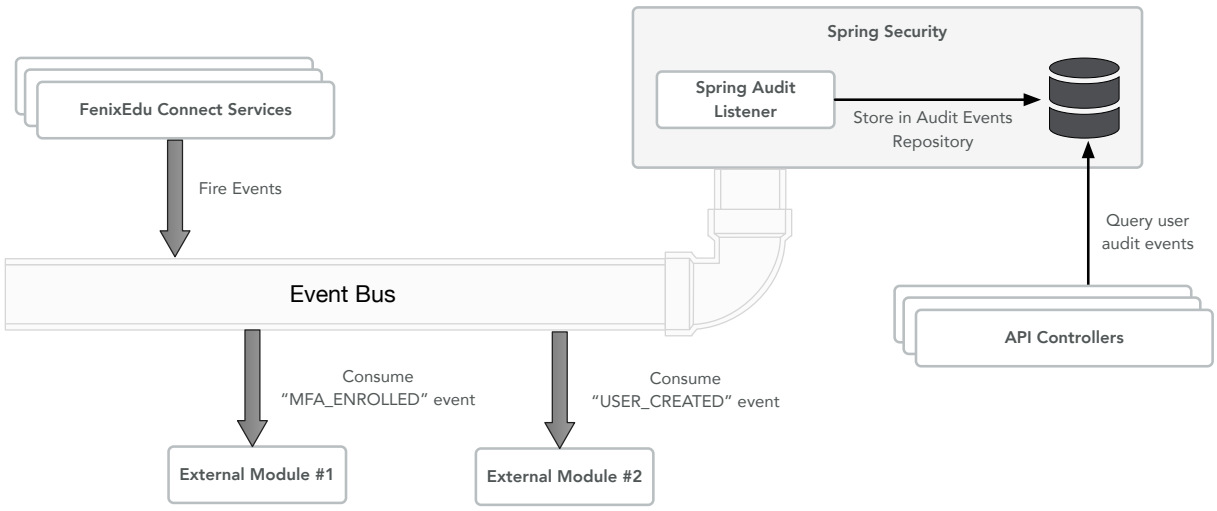


Figure 3.20: An overview of the flow of audit events from the source services to the possible consumers.

3.4.4.3 SecurityManager

While the most security-conscious users will likely opt to use two-factor authentication whenever it is available, the often voluntary nature of this security feature tends to result in less than ideal adoption rates. With the goal of increasing security and reducing the chances of an account becoming compromised FenixEdu Connect actively monitors authentication attempts against suspicious behaviours. Through a direct integration with the `DelegatingAuthenticationProvider` Connect's `SecurityManager` service validates each authentication event against a set of checks:

- **Repeated attempts with invalid credentials:** Leveraging the auditing infrastructure outlined in section 3.4.4.2 Connect limits the number of invalid authentication attempts to a configurable value, after which the offending IP address is blocked from making further requests.
- **Significant location changes between login events:** IP location is used to obtain an approximate distance between the current authentication event and the last. If this distance is incompatible with the time required to travel it at a preset speed the account is suspended and the user is notified.

As with most Connect components, the provided implementation of the `SecurityManager` service

can easily be extended or even replaced by an organization which requires a different set of account security verifications.

3.4.4.4 Connect Extensions

Today's business environment requires organizations to undergo changes almost constantly. Authentication systems must be able to adapt to this fast pace with little to no downtime. While Connect's architecture allows organizations to develop additional modules to fine tune its behavior to the general requirements of the organization there are some scenarios in which the development of a separate module may not be justifiable. Connect Extensions offers system administrators the ability to customize the application's behavior in runtime, without the need to perform changes to the codebase.

An extension is a regular java class, which implements an interface from a specific set, known as Extension Points. The proposed solution implements two extension points, as a proof-of-concept.

- **Authentication**: Through the `AuthenticationExtension` interface, administrators have the chance to run code synchronously as part of every successful authentication transaction.
- **Pre-User Registration**: Through the `PreUserRegistration` interface, allows administrators to run code immediately before the creation of a new user account.

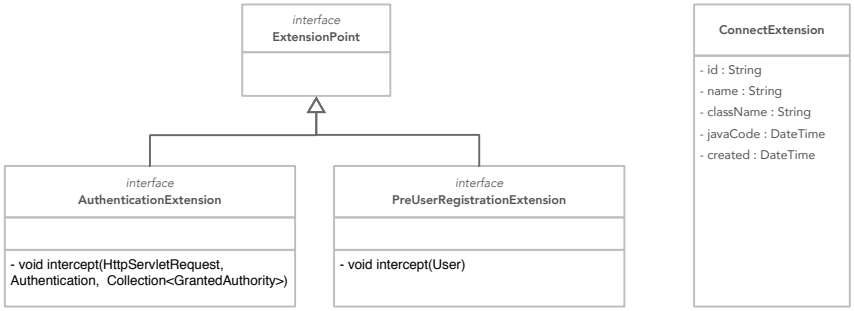


Figure 3.21: A UML diagram of the classes that make up Connect's Extension Points feature.

Extension point interfaces extend the top level `ExtensionPoint`. Connect exposes a set of API endpoints to retrieve the current extensions or add new ones. New extensions can be added by sending a POST request to the `/api/v1/extensions` endpoint with the extension's type, full class name, a display name and the class' code encoded with base64. Connect then performs an initial validation by attempting to compile the code. If no exceptions were thrown, it is encapsulated in a `ConnectExtension` object which, being a domain entity, is persisted to the database. At the same time an instance of the provided class is created and maintained in memory, ready to use. When the application is started, `ConnectExtension` objects are loaded from the database, compiled and instantiated into memory.

Listing 5 provides an example for a possible authentication extension. In this example, a change in the organization's legal requirements has required users to give explicit consent to its updated privacy policy, which is managed in a separate micro-service. This system exposes an API that allows callers to check whether or not a given user (identified by its username) has consented to the updated privacy

policy. The IT Services team has been ordered to deny users access to all systems until they consent to the new privacy policy.

```java
package org.fenixedu.connect.core.security;

import org.fenixedu.connect.core.security.exception.ConnectAuthenticationException;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.web.client.RestTemplate;

import javax.servlet.http.HttpServletRequest;
import java.util.Collection;
import java.util.Collections;

public class GDPRConsentAuthenticationCheck implements AuthenticationExtension {

    private static final String GDPR_CONSENT_HOLDER_SERVICE_URL = "https://acme.org/api/v1/gdpr/consent";

    @Override
    public void intercept(HttpServletRequest request, Authentication authentication,
     ↪  Collection<GrantedAuthority> grantedAuthorities) {
        String username = authentication.getName();

        String path = GDPR_CONSENT_HOLDER_SERVICE_URL.concat("/").concat(username);
        RestTemplate restTemplate = new RestTemplate();
        ResponseEntity<Boolean> responseEntity = restTemplate.getForEntity(path, Boolean.class,
         ↪  Collections.emptyMap());

        if(responseEntity.getBody() == false) {
            throw new ConnectAuthenticationException(HttpStatus.PRECONDITION_REQUIRED, "GDPR_REQUIRED",
            "You need to consent to the organization's privacy policy before logging in");
        }
    }
}
```

Listing 5: An authentication extension that queries a remote service to check if the user has already consented to the organization's privacy policy. If not, an authentication exception is thrown.

The simple example highlights the flexibility of Connect Extensions. Having the ability to run code in a synchronous way after a successful primary authentication allowed Connect administrators to implement the required block without the need to create a separate module to implement this check and with no downtime. The Pre-User-Registration extension point could, for instance, be used to send a welcome email if the user belonged to a specific role, or to notify an external system that a new user had been created.

The ability to inject Java code at runtime is certainly powerful, but it is not without its risks. There are two possible attack vectors to abuse this feature: the authorization checks for the API endpoints can fail, allowing a user without the right permissions to inject an extension in the system or an account with administrative privileges may be compromised. If the first were to happen, the security of the remainder endpoints would also have been breached, since Connect delegates the authorization checks to the Spring Security framework. On the other hand, if an administrator's account were to be compromised, a significant amount of user data would also be available through the regular API endpoints.

It could be argued that Connect Extensions should run in a controlled, sandboxed environment, isolated from the rest of the application. While this would certainly be a necessity if the classes came from an untrusted source, this approach would severely limit the feature's flexibility. A developer wanting to

write a rule to send an SMS or an email under certain conditions can just resort to Spring's dependency injection features to `@Autowire` a `EmailGateway` or an `SMSGateway` from the core module instead of having to copy the relevant code from this classes to the extension. However, considering the potential security implications of this feature, which may not be acceptable for some organizations, it can easily be turned on/off from the application's configuration properties.

3.4.5 External Integrations

3.4.5.1 Trusted Applications

While Connect implements flexible and comprehensive support for OAuth applications, enabling users to grant granular permissions to their secure resources this security model is naturally designed for access delegation. Internal applications still require a way to authenticate users who visit their web pages directly. These applications are known as *Trusted Applications* and have access to the following set of features:

- **Key Synchronization**: While Connect's authentication API is open, the public counterpart of the signing keys used to sign the issued tokens is a protected resource, only available for trusted applications.
- **Revoked Token Synchronization**: One of the main disadvantages of using JWT as authentication tokens is the inability to revoke them in real time once they have been issued. When a token is manually revoked (by the owner or a system administrator) it is added to a list of revoked (but not yet expired) tokens that is accessible to trusted applications. Consumers should periodically refresh this list and reject any token whose ID matches one of the list.
- **Single Log-off protocol**: Trusted applications are allowed to take part in Connect's single log-off protocol.

3.4.5.2 Single log-off protocol

To ensure that trusted applications reject recently revoked authentication tokens without having to rely on the periodic synchronization of the token revocation lists Connect implements a single log-off protocol. This implementation is compliant with a recently published draft for a back-channel logout protocol for OpenID Connect [44].

Trusted applications define a logout handler URL as part of their Connect registration info. When a user performs a logout on a Connect-enabled application the request is forwarded to a Connect instance which POSTs the now invalid token information to all registered trusted applications. which can now add it to their internal revocation lists. If a user attempts to access one of these applications with the old token it will be found in the client's revocation list and thus, access will be denied, causing the user to have to re-authenticate.

The revocation notice is sent as JWT, digitally signed by a valid Connect key, with a `sub` field corresponding to the user's id, a `tid` claim with the revoked token's ID and a predetermined claim name to

identify the event type. Since the token ID uniquely identifies the JWT (thus allowing client applications to reject it, if found) there is no need to expose the entire token to the network which could be exploited to perform man in the middle attacks on vulnerable applications. While applications are not required to implement the logout handler (and thus take part in the Single Log Off protocol) it is highly recommended they do, to prevent recently revoked tokens from being used to impersonate users (which may happen until the client applications update their internal revocation lists).

3.4.5.3 SAML Identity Provider

As countries expand their offer for digital services, so does the need for reliable user authentication across multiple realms. Cross-country identity federation has usually been carried out through the SAML protocol, which is currently used in some of the most common projects, such as the InCommon Federation[3] in the United States or the European STORK 2.0[4] project. Academic organizations wanting to take part in these identity federations need to implement the required support for the SAML protocol as Identity Providers.

To alleviate this issue, Connect provides an optional module that allows it to act as a SAML IdP, implemented in the `connect-saml` package. Since institutions may have the need to take part in multiple identity federations this module was designed around the concept of SAML Connectors. A SAML Connector represents a SAML IdP configuration for a specific purpose. Rather than creating SAML Connectors ad-hoc, each one must be associated with a Trusted Application, which provides additional context for the integration.

In the SAML protocol, the attributes to be provided after a successful authentication are statically defined at the time of the initial configuration of the federation. To maintain this list of attributes, Connect defines the `SAMLConnector` entity. However, SAML attributes are often identified using Uniform Resource Names (URNs) which may not correspond to the attribute keys used by Connect's Profile Providers. To allow federations to request access to the users's profile attributes each SAML Connector specifies a list of mappings between the internal attribute names and the desired URN with which they should be exposed in the SAML Response. Figure 3.22 provides an overview of how this attribute map is used in the SAML SSO request flow.

3.4.6 Monitoring and Auditing

3.4.6.1 Logging

Connect takes advantage of Spring's built in integration with SLF4J[5] which provides a common facade for logs while allowing developers to select from a wide range of concrete logging implementations. This allows Connect's log output strategies to vary from simple console output to sophisticated log analysis tools, such as Greylog[6] or Logstash[7] with only minor changes to the configuration files and included

[3] https://www.incommon.org/federation/
[4] https://www.eid-stork2.eu/
[5] https://www.slf4j.org/
[6] https://www.graylog.org/
[7] https://www.elastic.co/products/logstash

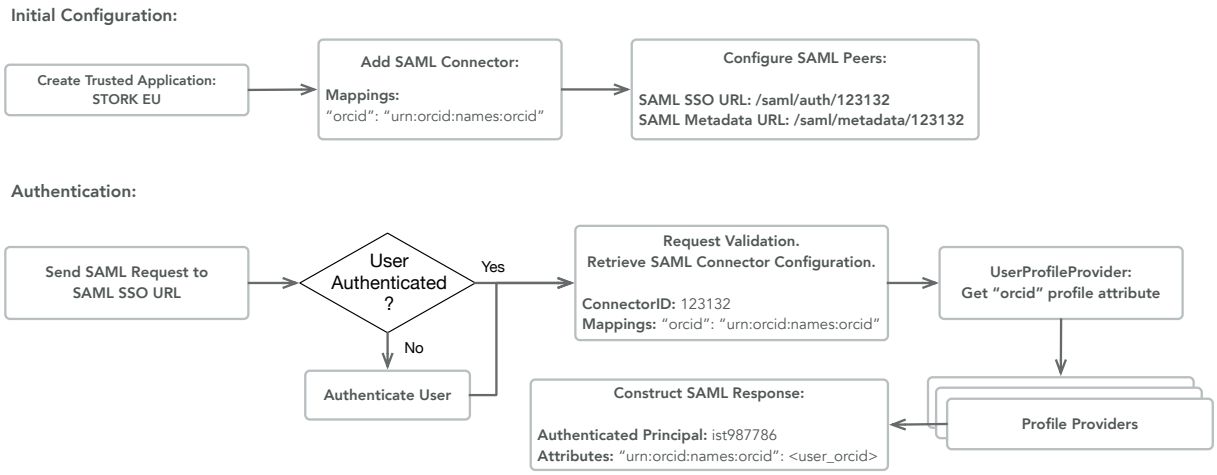


Figure 3.22: An overview of the configuration and authentication flows of the SAML Connector.

dependencies. Most of the relevant actions are logged. While the format of these logs is configurable by each institution (through configuration properties) Connect implements some additional logic to supply relevant information to the logging agents:

- The use of a Mapped Diagnostics Context (MDC[8]) allows the application to maintain a thread local map of information that can be accessed by the logging framework. Connect injects the current user's id and the token ID used for authentication in the MDC, which is then appended to each log line generated by that thread.
- The user's id is injected in the HTTP session after a successful authentication allowing the application server to populate the access logs with the authenticated user.

The use of these techniques reduces the need to manually add the author information to each logger call, simplifying the log statements. In addition, since this information is part of the log message's format, it can easily be parsed and extracted by log analysis tools, allowing system administrators or auditors to track the sequence of requests made by a given user at some point in time.

3.4.6.2 Audit Log

Application logs are an invaluable tool for monitoring the real-time state of any software component as well as a major contribution for troubleshooting issues or retracing user behaviour. However, when performing these actions there is often the need to access information that was only partially logged, or not logged at all. Developers are left with no choice but to introduce changes to the application's codebase that increase the log coverage and wait for the issue to happen again. This might not be possible in the event of a security breach, where the missing log data could be essential for determining the extent of the damages. To help to minimize this issue, organizations often resort to audit logs, which provide an historical view of the application's data over time. Maintaining an audit log presents a clear set of advantages:

[8]https://www.slf4j.org/api/org/slf4j/MDC.html

- **Troubleshooting/debugging**: Having the ability to get a snapshot from the application's database for a specific point in time or being able to trace all the changes made to a particular object is an invaluable tool for tracking down issues that may be otherwise hard to reproduce.

- **Usage metrics**: While application logs might be enough for gathering basic usage metrics having the ability to drill down in snapshots of the database may allow for the collection of detailed metrics that would otherwise not be possible to generate from the logged data.

- **Compliance**: Having the ability to track changes to user accounts, credentials and roles is essential for compliance with the most common security standards and frameworks, such as the Payment Card Industry Data Security Standard (PCI DSS[9]) or the General Data Protection Regulation (GDPR[10]).

However, the extra information provided by an audit log also poses a few challenges on the audited applications and their supporting infrastructure. Audit logs generate a significant amount of data, which has to be persisted to a secure location. The most common audit tools use the same database as the main application. If the application's databases are backed up on regular intervals, the same audit information will be persisted over multiple DB backups. In addition, database writes incur in a natural performance hit, since there is a need to create additional entries every time a row is modified.

Spring Data JPA ships with basic support for auditing, allowing some metadata, such as an object's last modification date and the responsible user to be automatically persisted. While this may be enough for simpler applications, it is not the case of Connect, where its core role in an organization's security layer demands a higher degree of control and accountability over database changes. The solution came in the form of Envers[11], a Hibernate module that provides an automatic audit log for all JPA entities annotated with `@Audited`.

Envers works by maintaining a history of revisions, with each one corresponding to a transaction made to the application's database. Each transaction is assigned a sequential identifier, which is global in scope. Figure 3.23 depicts a simple use case. For every object, it is possible to query its revision history, allowing auditors to track down changes over time. Connect takes advantage of Spring Security's authentication context data to augments Envers' standard revision metadata with the user who originated it, providing accountability over every change made to the database.

[9] https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf
[10] https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:02016R0679-20160504&from=EN
[11] http://hibernate.org/orm/envers/

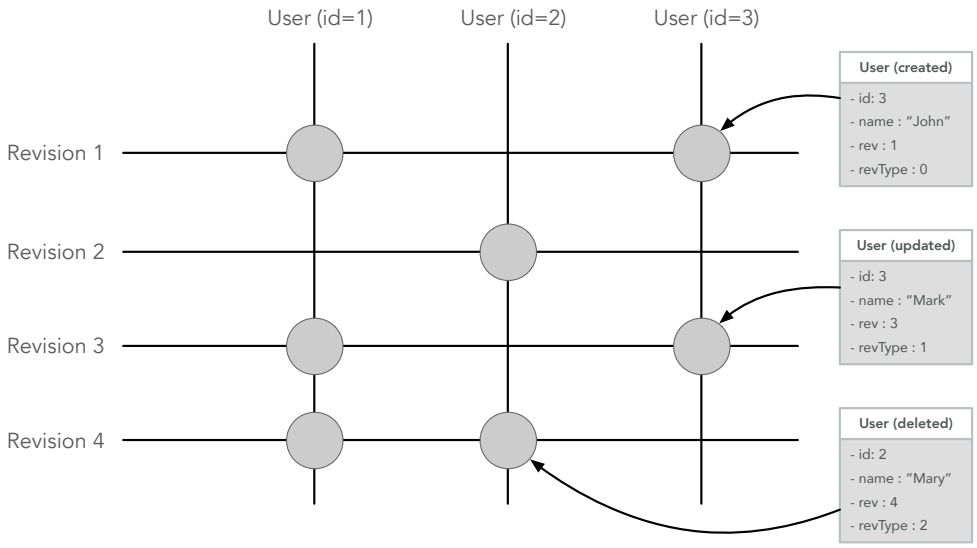


Figure 3.23: An example of how Envers uses the concept of revisions to track changes to a set of objects.

Overall View

Figure 3.24 provides an overall view of how the aforementioned components make up the complete architecture. The authentication and OAuth sub-systems, being the most directly exposed to the end user make up the interface layer, where external integrations such as SAML were also included. These services share a common data source for user data, provided by the components in the identity layer, which operate on a lower abstraction level and manage accounts, credentials and the direct interface with the profile providers. The security layer encompasses the components that are manage the user sessions, issued tokens and overall key management. Finally, as a cross-cutting concern, the audit layer includes the services responsible for logging all relevant user actions, both through regular logs, audit logs, and account security events, which are made available to the end user.

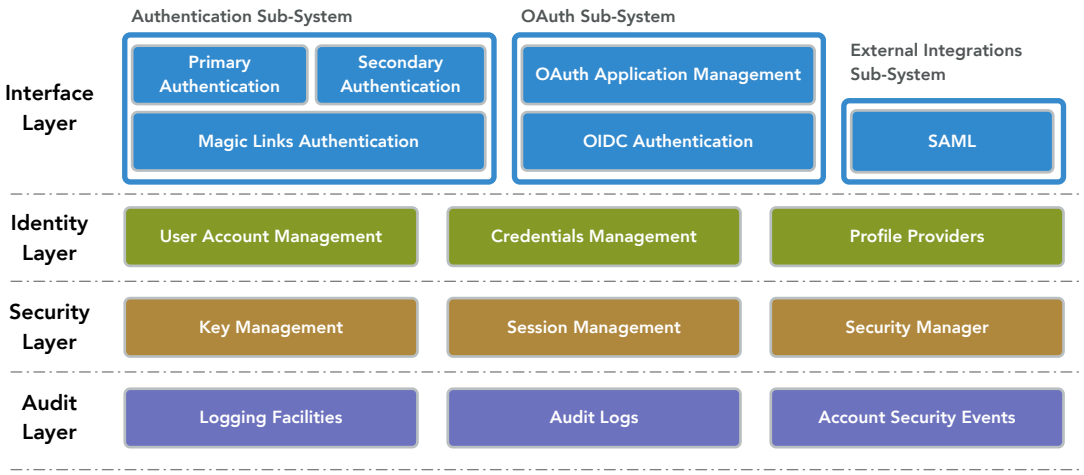


Figure 3.24: An architectural overview of Connect's main components and their associated abstract layers.

Chapter 4

Evaluation

This chapter discusses the validation of the implemented solution. While methods based on the quantitative analysis of key metrics (such as the total execution time or resource requirements of a given task) are often preferred for the evaluation of software applications there were a set of constraints that limited the effectiveness of these methods in the evaluation of the implemented solution:

- As previously discussed in chapter 1 regarding the motivation of this thesis's work, FenixEdu Connect is based on the premise of expanding the realm of features that are typically available in the IAM systems of higher education institutions. As such, it is not expected of this solution to provide a noticeable improvement in the execution time of common tasks, such as authentication or access decisions, preventing any comparison with the existing systems' performance from providing a relevant result.

- Most IAM systems which offer a similar feature set to the one implemented in FenixEdu Connect are proprietary and require comercial licenses that depend on the number of active users. This renders comparisons with these solutions unfeasible from an economic standpoint, as the appropriate licenses would have to be purchased for a significant number of users to be able to conduct a relevant study.

The most significant validation results would only be achieved for a deployment of this solution in a production environment, where it could be subject to the common load and usage patterns. Only then would some of Connect's main advantages, such as the reduction in the number of support tickets achieved from the multiple self-served password reset strategies have a measurable (and thus comparable) impact.

While the initial schedule included the deployment of FenixEdu Connect at Instituto Superior Técnico as part of this thesis, where it would work alongside the existing system for a limited period acting as the IAM solution for a pre-approved set of non-vital applications, unforeseen circumstances from the part of IST's IT department prevented this from happening. The decision to roll out a new IAM system is never one to be made lightly, due to the sensitive nature of these applications and the data they are responsible for protecting. The imposed requirement of performing a thorough security audit of the

project's codebase and internal delays in providing a frontend compatible with IST's design guidelines dictated this decision.

The remainder of this section will then focus on the evaluation of the implemented solution by analyzing feature completion, that is, which of the requirements specified in section 3.1 were successfully implemented in the final product. Following this discussion, a case study outlining a possible Connect deployment at Instituto Superior Técnico will be presented where the appropriate comparisons with the comercial alternatives will be made.

4.1 Feature Completion Analysis

This analysis is broken down by functional areas, following a similar structure to section 3.4 where the solution's implementation details were outlined.

4.1.1 Authentication

Authentication			
Primary Authentication			
Feature	**Completed?**	**Requirement**	**Okta/Auth0**
JWT Authentication Tokens	Yes	R2.1	Yes / Yes
Authentication API	Yes	R2.1	Yes / Yes
Extensible Primary Authentication Providers	Yes	R2.2	No / No
Internal Authentication Provider	Yes	R2.2	Yes / Yes
LDAP Authentication Provider	Yes	R2.2	Yes / Yes
Passwordless authentication provider - Magic Links	Yes	R2.2	Yes / Yes
Multi-factor Authentication (MFA)			
Feature	**Completed?**	**Requirement**	**Okta/Auth0**
MFA Management API	Yes	R2.3	Yes / Yes
Extensible MFA Providers	Yes	R2.3	No / No
Time-based One-Time Password (TOTP) Factor	Yes	R2.3	Yes / Yes
Universal 2nd Factor (U2F)	Yes	R2.3	Yes / Yes
SMS Verification Factor	Yes	R2.3	Yes / Yes

Table 4.1: Authentication Features

Table 4.1 presents an overview of Connect's authentication features. As described in section 3.4.1.3 *passwordless* authentication is implemented in a separate package allowing institutions to opt in to use this provider. Regarding the secondary authentication options, support for the three main types of providers is available as well as a comprehensive API for managing these factors (creation, removal and

verification).

4.1.2 Identity Management

Identity Management

Feature	Completed?	Requirement	Okta/Auth0
User Management API	Yes	R3.2	Yes / Yes
User Search API	Yes	R3.2	Yes / Yes
User Invite API	Yes	R3.3	Yes / Yes
Extensible support for profile providers	Yes	R3.1	No / No
Internal profile provider	Yes	R3.1	Yes / Yes

Table 4.2: Identity Management Features

Connect implements the necessary infrastructure to allow organizations to delegate the responsibility over sets of profile attributes to multiple applications within the organization. These attributes are made available for both OAuth applications and external connections (from SAML clients, including the required conversion between its namespace and Connect's). Additionally, Connect also allows for profile attributes to be stored in its database. There are, however, no security constraints for these fields (i.e. no specific role is required to access profile attributes stored directly in the Connect instance).

4.1.3 Access Delegation

Access Delegation

Feature	Completed?	Requirement	Okta/Auth0
OAuth Authorization Server	Yes	R4.1	Yes / Yes
OAuth Application registration & management API	Yes	R4.1	Yes / Yes
Scopes management API	Yes	R4.1	Yes / Yes
Support for all OAuth 2.0 grant types	Yes	R4.1	Yes / Yes
OpenID Connect Identity Provider	Yes	R4.2	Yes / Yes

Table 4.3: Access Delegation Features

Connect implements a feature-complete OAuth 2 Authorization server with a RESTful API for application, scope and grant type management. Support for the required endpoints of OpenID Connect is available, although some of the optional endpoints (such as the discovery and session management features) were not implemented.

4.1.4 External Integrations

External Integrations

Feature	Completed?	Requirement	Okta/Auth0
Management of authentication interceptors in runtime (Connect Extensions)	Yes	R5.2	No / Yes
SAML Identity Provider	Yes	R5.3	Yes / Yes
LDAP account synchronization	Yes	R5.1	Yes / Yes
LDAP Interface (for external queries)	No	R5.1	Yes / No

Table 4.4: External Integrations

At the heart of Connect's requirements was the need to support a wide array of existing systems with different authentication protocols. This was mainly translated in the need to act as a SAML IdP and a CAS Server to ensure proprietary applications, which could not be updated to use the authentication API, would still work in an organization with Connect as its IAM system. While support for SAML was included in a separate package, `connect-saml`, it was not possible to accommodate support for the CAS protocol in the project's timeframe.

Extension points, which allow system administrators to inject custom logic in specific actions at runtime (such as after a successful authentication) were not part of the initial project requirements. They were, however, included in the final implementation. We strongly believe the possible applications of this feature far outweigh the additional development time, which negatively impacted the completeness of other features such as the LDAP interface, designed to allow external systems to synchronize with Connect's user database using the LDAP protocol.

4.1.5 Security

Security

Feature	Completed?	Requirement	Okta/Auth0
Secure Key Generation	Yes	R6.1	- / -
Independent key encryption passwords	Yes	R6.1	- / -
Security event listings for end users	Yes	R6.2	Yes / Yes
Session Management with remote session termination	Yes	R6.3	Yes / Yes
Single log-off protocol	Yes	R6.3	Yes / Yes
Key segregation (separation between keys used for authentication from keys used to sign OAuth tokens	No	R6.1	- / -

Table 4.5: Security Features

Most of Connect's initial security requirements were accomplished. Support for remote session termination is built-in but, naturally, requires the client applications to take part in Connect's Single Log-off protocol to be effective. Support for specifying the uses of each of Connect's signing keys was not completed. Some of the requirements pertain to specific implementation details that are not available in closed-source products. For this reason, it was not possible to ascertain if the commercial solutions complied with a set of requirements.

4.1.6 Monitoring & Auditing

Monitoring & Auditing

Feature	Completed?	Requirement	Okta/Auth0
Audit Logs for all domain objects tagged with the authenticated principal for every change event	Yes	R7.1	No / No
Comprehensive logging of all relevant activity tagged with the authenticated principal	Yes	R7.2	Yes / Yes
Support for multiple logging implementations and data analysis tools (Graylog, Logstash, etc...)	Yes	R7.2	No / No

Table 4.6: Monitoring and Auditing Features

The implementation complied with the established monitoring and auditing requirements.

4.2 Case study: FenixEdu Connect at Instituto Superior Técnico

This section will contextualize the implemented solution in a real life scenario by exploring its possible deployment at Instituto Superior Técnico as a replacement for the current IAM solution. Seven use cases will be analyzed, following a similar structure to the previous section.

4.2.1 General Overview of the IAM scenario at IST

On any given day more than one hundred services run on IST's IT infrastructure, relying on a set of authentication providers (CAS, OAuth, JWTs, SAML) to authenticate its users. The causes for this fragmentation are twofold: (1) the decentralized nature of the development process, which has been carried out by a multitude of organizations within IST ranging from student groups to research units and (2) the lack of a publicly accessible authentication gateway that allows developers outside the university's IT department to authenticate its users.

With the aim of centralizing authentication and providing a consistent experience across its applications IST introduced a CAS service which provides SSO capabilities to the most used applications within the organization. While this was clearly a step forward, CAS's adoption has been less the ideal, with a significant number of services still running their own authentication logic. This can both be attributed to the administrative restrictions imposed on the applications that can use the CAS system (which is seen as an internal tool that should not be used by third party developers) as well as the complex setup and development cost required to support the CAS protocol as an authentication provider.

The current authentication system is depicted in figure 4.1. At its heart lies an LDAP directory, which acts the single source of truth for user identities across all IST systems. Services such as the ID gateway are implemented on top of this user store to provide authentication mechanisms suitable for each application's needs. While every year more than one thousand students enrol into IST for the first time (which requires user accounts to be setup) there is not yet a process to create these accounts ahead of time. Users enter their personal information on one of the client applications (FenixEdu Academic) upon their arrival for the first time. This system is set up to periodically export a JSON file with the account information of all its local users. The file is then periodically queried by an application that is responsible for creating the appropriate records in the LDAP directory, updating existing ones when necessary.

4.2.2 Authentication use case

While the use of 2-Factor Authentication has become commonplace on most enterprise organizations it is still not part of Técnico Lisboa's centralized authentication. Support for 2FA must be implemented at the application level, separating the authentication concerns between two entities with CAS handling the first (more general) authentication layer and the application implementing custom logic to support 2FA and handling its verification. Not only does this put an additional burden on the users, who are forced to register their 2FA tokens in multiple services as it forces developers to deal with unnecessary

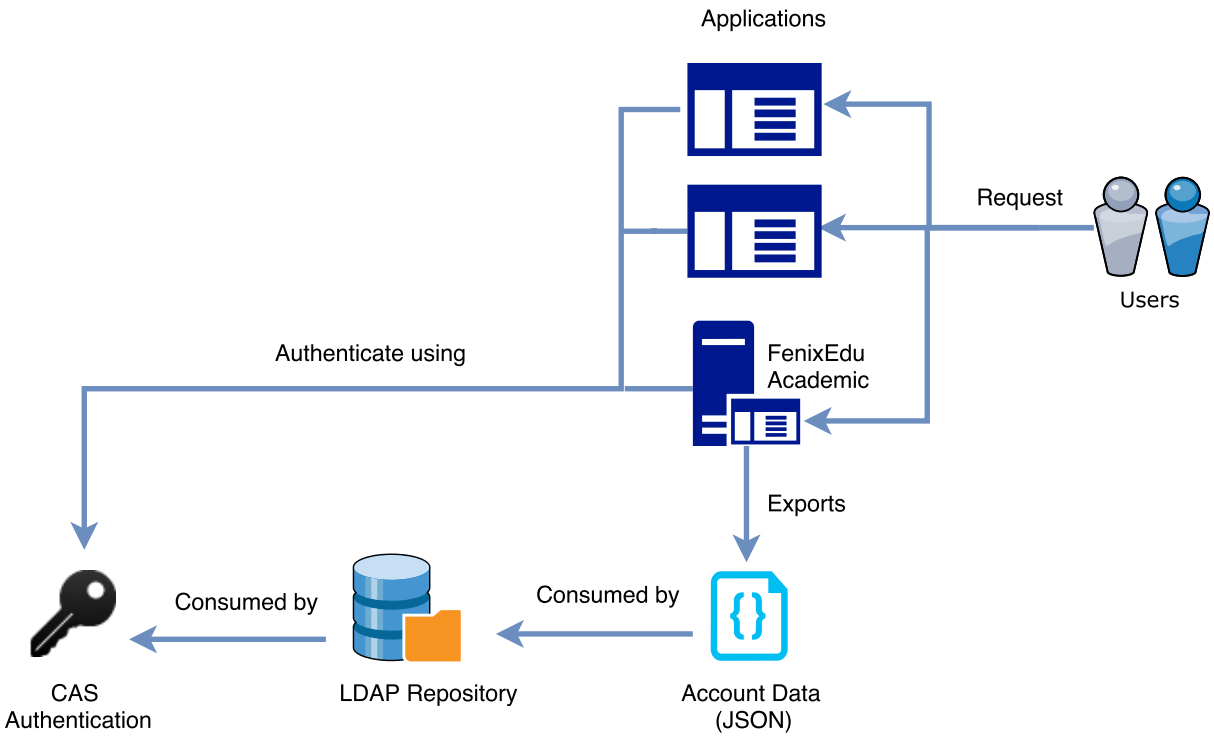


Figure 4.1: An overview of the authentication system at IST.

authentication concerns (such as token registration and validation).

Applications that want to enforce 2FA verification to protect sensitive areas (such as administration portals) or as an additional confirmation step before important actions are taken (such as submitting a course's marksheet) are thus faced with the technical challenges of deploying internal support for these authentication factors. This is a clear violation of the Separation of Concerns principle. When operating in an environment protected by a SSO system applications should not have to perform custom authentication logic and should, instead, rely on the existing infrastructure to validate the requestor's identity and provide any required personal information.

FenixEdu Connect allows users to register a number of 2FA factors ranging from Time-based One-Time Passwords, SMS phone numbers or U2F tokens in a centralized application that are made available on every subsequent authentication attempt. Figure 4.2 provides an overview of how Connect would be used with one of the existing applications (FenixEdu Academic) to enforce 2FA verification when users attempted to access the application's administration portal.

4.2.2.1 Comparison with Commercial Offers

The relevance of 2FA is the current security ecosystem has made it commonplace in most commercial IAM offerings. Both Okta and Auth0 have comprehensive support for the registration, management and verification of the main types of 2FA providers. However, there is no support for custom provider types. The versatility of FenixEdu Connect allows its possible use cases to extend beyond the realm of software authentication allowing it, for instance, to be integrated with a building's access control system. In this

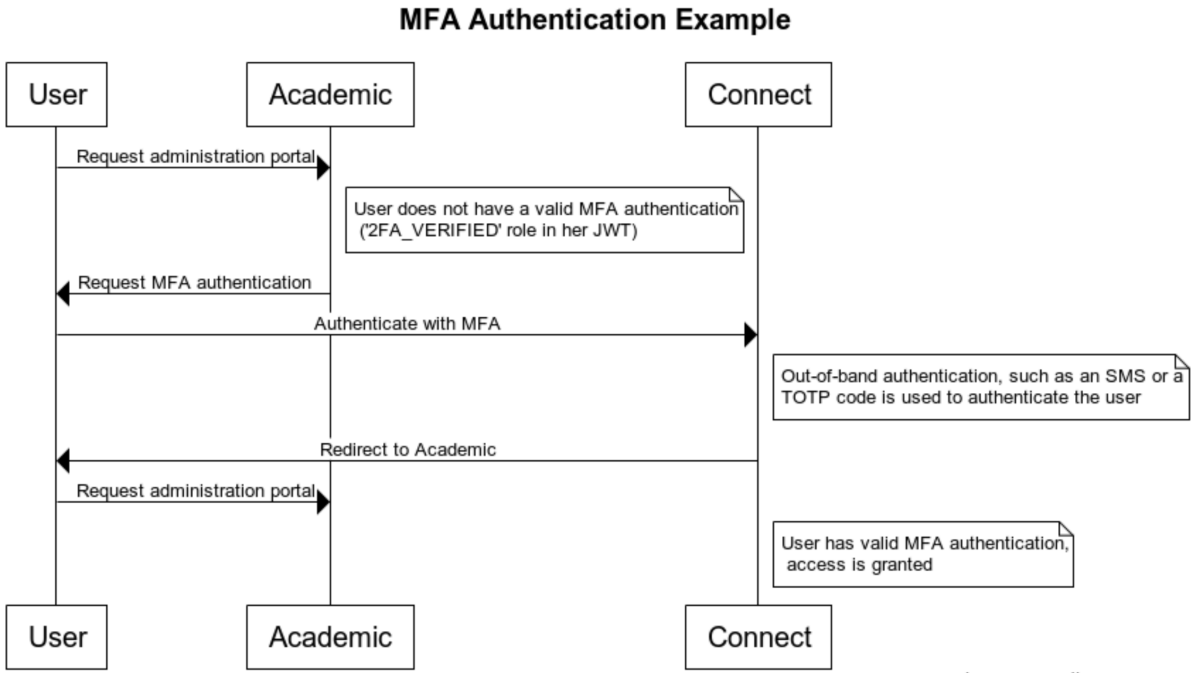


Figure 4.2: An overview of how Connect can be used to provide global MFA support for the client applications at IST.

specific situation, it would be feasible to require a combination of three authentication factors (such as a password, smart-card and TOTP code) to access a especially sensitive area (such as a datacenter). Commercial solutions lack the versatility to implement these types of advanced identity verification.

4.2.3 Account Management use case

As the number of applications that used the centralized authentication system increased so did the need to provide users with a quicker and easier way to recover their credentials. The ID project (`https://id.tecnico.ulisboa.pt`) aimed to provide both end users and developers with a common gateway for user authentication augmenting the existing CAS service with additional account management tools including a self-served password recovery system. There are currently two reset strategies: (1) an SMS code sent to the user's registered mobile phone or (2) the use of the Portuguese Citizen Card for identity verification. While these strategies played a significant role in the reduction of the IT Help Desk support tickets for password resets they have had varying degrees of success in the different types of IST users. As an example, students rarely update their profile when their phone number changes. This is especially significant for international students which often purchase Portuguese mobile SIM cards on arrival that are only active during their stay. Future attempts to reset their password will require access to the registered phone, which is often no longer in service. Since the only alternative to this strategy is to visit the university's IT services, students are often left without access to their IST accounts for extended periods of time.

FenixEdu Connect extends the existing password recovery strategies by offering users the ability to reset their credentials by email, using an alternative address that may have been previously registered

by the user. Since in a Connect-enabled environment applications are offered a unified version of the user's profile information it becomes possible for multiple systems to contribute contact information to the user's identity record, i.e. both the email provided by the HR management system and the one registered in the academic management system will be available for use as a password reset option. When the registered contact information cannot be used for password recovery (either because it is missing or no longer available) FenixEdu Connect provides an alternative approach consisting of a video call with a member of the IT support staff, implemented on top of the WebRTC protocol suite. During this call the support staff member will validate the user's identity by a predetermined protocol (such as requiring the user to display her ID card and comparing its photo with the user). If validation is successful, the member of the support staff will be able to add an additional contact point (an email address or phone number) to be used by the user for password recovery.

4.2.3.1 Comparison with Commercial Offers

Both Okta and Auth0 have built-in support for account recovery options based on the traditional strategies (SMS and email). There is not, however, support for any custom implementations. FenixEdu's extensible approach to password reset strategies allows, for instance, for the development of self-service solutions. It would be possible to deploy Kiosks across the university where by authenticating with their national citizen card students would be able to, for instance, reset their lost passwords without having to visit the university's IT department with clear advantages to the efficiency of these services.

4.2.4 Identity Provider use case

At IST user identities live in semi-isolated silos maintained by each application, with attributes commonly duplicated in multiple applications, in some instances with conflicting values. Students who enrol in a degree fill out their personal information in FenixEdu Academic, which exports a small subset of it to the core LDAP directory. Other applications may query the LDAP directory for these attributes, but they are unable to access the information that was left in FenixEdu Academic. As an example, while FenixEdu Sotis stores the user's ORCID number, since it isn't part of the information exported to the LDAP directory, the user is forced to fill it in FenixEdu Academic if she wants it to be exposed in her public profile page.

FenixEdu Connect would allow for all applications at IST to share a global view of the user's profile data through the use of small add-on modules, known as profile providers, responsible for establishing an interface between the Connect service and the internal applications. In this scenario, the information gathered during the student account provisioning would be split and distributed by the systems that have declared to be authoritative over those specific attributes. As an example, figure 4.3 provides an overview of how data collected is handed over to multiple systems according to the attributes they are authoritative over.

Connect provides an abstraction layer over where the profile data is located. Applications are not made aware of where a specific profile attribute is stored. Rather they request it to be read or written

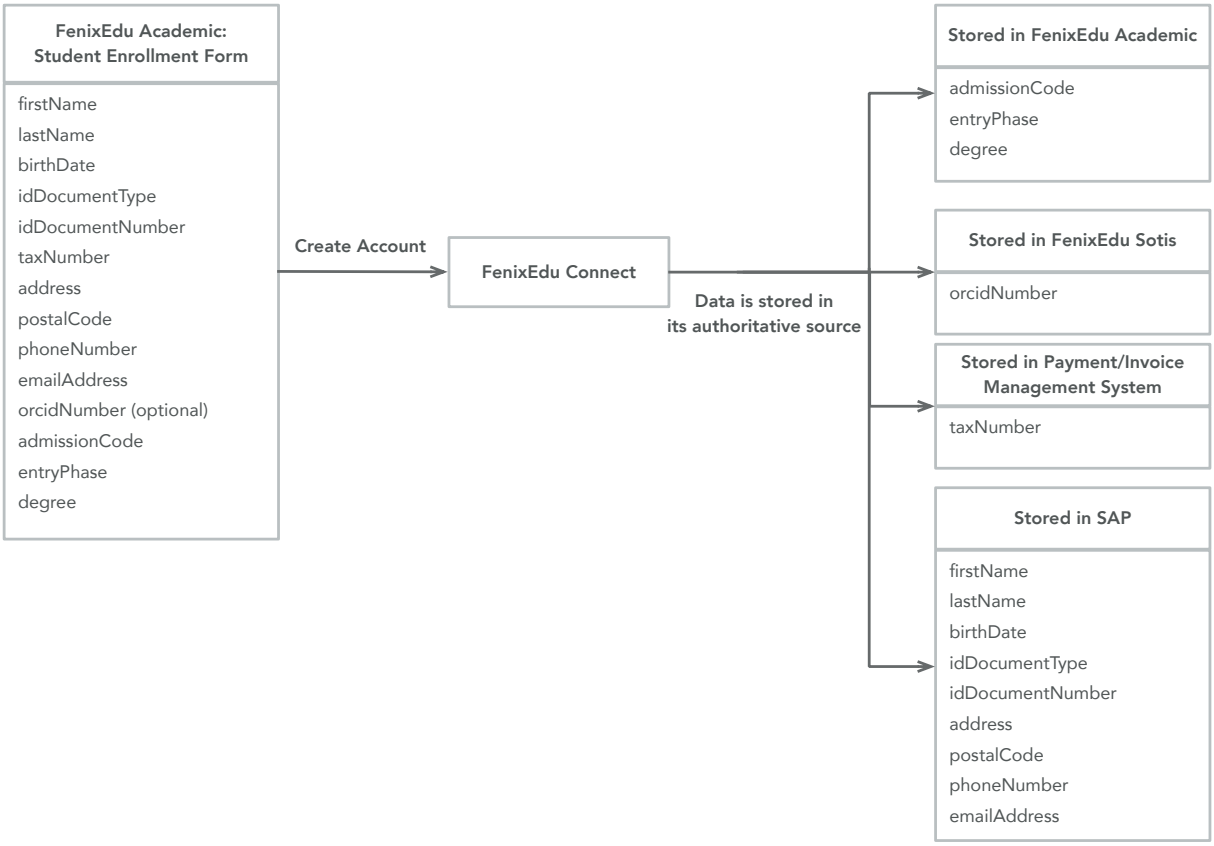


Figure 4.3: An overview of how data collected in one application can be handed over to multiple systems according to the attributes they are authoritative over.

through a Connect instance. This flexibility allows for data to be moved between authoritative sources, without any disruption to the existing applications. As an example, while currently FenixEdu Academic holds the postal addresses of the registered students, and thus is the authoritative source for this attribute, in the event of the development of a micro-service to hold basic profile data only Connect would have to be notified about the change in the authoritative source. Since all the applications query the Connect instance to obtain the user's address, every system would continue to behave in the same way, without the need for any further updates. Figure 4.4 provides an overview of how Connect proxies multiple applications to expose a single user identity to the client applications.

4.2.4.1 Comparison with Commercial Offers

While Auth0 has support for custom user stores, allowing system administrators to define a set of scripts that run whenever an authentication attempt is made (which allows Auth0 to proxy a remote directory) all the profile information must be stored in the cloud service itself. Unlike FenixEdu Connect, it isn't possible to have Auth0 distribute profile data over a set of authoritative sources.

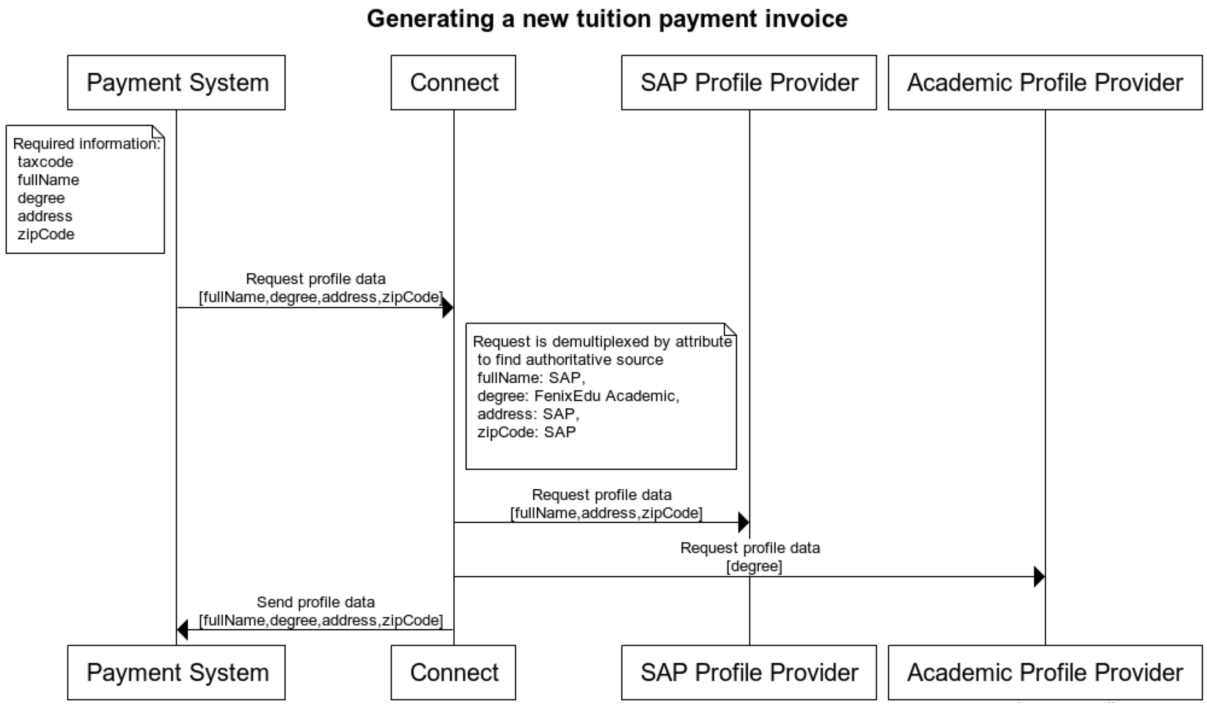


Figure 4.4: An overview of how Connect proxies multiple applications to expose a single user identity to the client applications.

4.2.5 OAuth use case

Instituto Superior Técnico offers its academic community mobile applications for the two major operating systems (iOS and Android) [1] enabling users to access their most relevant information, ranging from course announcements, evaluations or pending payments to canteen menus and car park occupancy levels. These applications consume a RESTful API exposed in a contribution module compiled alongside FenixEdu Academic and secured with OAuth, through the Bennu OAuth project. The existing OAuth implementation relies on Academic to act as an authorization server, responsible for the registration of client applications, scopes and authorizations. It also exposes all the protected resources in the same application resulting in a self-contained system.

IST offers a free shuttle service [45] to transport students and academic staff between its two campi. The shuttle service runs multiple routes, on varying schedules and the required capacity changes every day. To alleviate this issue an online platform, the Shuttle Management Service (SMS), has been developed to allow students to book their rides ahead of time, allowing IST personal to reserve enough buses for each day.

For the mobile applications to be able to offer the ability to reserve shuttle tickets would require them to be registered in two OAuth authorization servers (FenixEdu Academic and IST's Shuttle Management System). Even in the event of both servers sharing the same authentication gateway (allowing the use of the same credentials to login) the user would still have to authorize access to both applications, each with its own set of scopes. If the user decided to revoke access to all the data that has been accessed by

[1] https://tecnico.ulisboa.pt/en/campus-life/services/tecnico-mobile-app/

the mobile applications it would be forced to revoke access on both FenixEdu Academic and the shuttle management service.

The deployment of FenixEdu Connect would severely reduce the complexity of this use case:

1. Both FenixEdu Academic's and SMS' developers would register their intended scopes in the organization's Connect server, ideally in an hierarchical structure such as `academic:read:evaluations` and `shutle:bookTickets`.

2. The mobile application developers would register an OAuth application in Connect, requesting access to both scopes.

3. Users who downloaded the applications would be asked authenticate in Connect and authorize access to both scopes. The application would be granted a signed access token, which would be used to access the protected resources on both applications.

While developers are still required to be aware of the different locations of each resource (i.e. academic resources are available in Academic's API while shuttle information is served from the SMS's API) the ability to access both APIs using the same credentials is a fundamental step forward that allows for the development of OAuth applications that consume information from multiple IST systems.

In addition to the aforementioned features, Connect has native support for OpenID Connect, allowing developers of OAuth applications to securely authenticate their users while still remaining isolated from Connect's Authentication API, reserved for internal applications.

4.2.5.1 Comparison with Commercial Offers

Both comercial solutions offer similar OAuth authorization server and OpenID Connect capabilities.

4.2.6 Session Management use case

Under the current SSO system in place at Instituto Superior Técnico it is impossible to list the current active sessions for each user or to remotely terminate a single session (or all of them). A user whose account credentials have been stolen may be unaware of this fact until unusual changes are made to the account. There is no security warnings for significant location changes between subsequent authentications or any limit on the amount of invalid password attempts that can be performed on a specific account. While the existing solution offers single logout within the same session (i.e. all the sessions started within that browser window are terminated) it is based on an iframe redirect to the client applications' specific logout URLs. This is known as a front-channel protocol and requires the user to remain in the logout page for enough time to visit all the required application logout URLs.

FenixEdu Connect offers complete session management allowing users to, for instance, remotely terminate any of their active sessions from any device. Users would be able to remotely end a session that was accidentally left open in a public computer from their device or end all active sessions simultaneously. In addition, the implemented solution ships with some basic security measures that detect

brute-force attacks and block the originating IP address after a number of incorrect authentication attempts. There is also protection against significant location changes in subsequent login attempts. All of these events are logged and are made available to the end user who can then evaluate if further security measures (such as a password reset) are necessary.

4.2.6.1 Comparison with Commercial Offers

Both solutions offer opinionated (and experimental) features of suspicious behaviour detection ranging from brute force prevention or IP distance checking between logins to password breach detection systems (which monitor known websites where stolen passwords are often published). These systems, while currently broader in scope than the ones offered in FenixEdu Connect, suffer from the significant disadvantage of being static. Developers are not able to customize them to their needs or expand them to comply with specific security requirements of the organization, a feature which FenixEdu Connect makes available from the start.

4.2.7 External Integrations use case

Beyond the mentioned authentication mechanisms IST actively takes part in a number of identity federations, mainly to interface with external vendors (such as Microsoft's Office365) or scientific publication repositories. These federations are based on the SAML 2 protocol, previously described in section 2.2.5. Additionally, CAS and SAML are often seen as competing authentication providers and IST is currently undergoing a push to phase out CAS within the next years replacing it for SAML in all the applications that still use this protocol to authenticate its users. The existing SAML IdP lives in the ID Project and, while effective, is severely limited in the attributes it can provide to the client applications, as its only attribute source is the university's LDAP directory.

FenixEdu Connect's SAML IdP leverages the extensible profile provider infrastructure to allow SAML client applications to request access to any profile attribute that is managed by Connect, regardless of the application responsible for maintaining it. Since SAML authentication requests resort to Uniform Resource Names (URNs) for attribute names and Connect profile attributes may use a more user friendly syntax, when creating a new SAML client application in Connect developers are able to specify the mapping between the internal attribute names and the URN they wish it is shared with in SAML responses.

4.2.7.1 Comparison with Commercial Offers

Both commercial solutions offer similar SAML IdP features incluiding the ability to map between SAML's URN attribute names and internal representations.

Discussion

The previous use cases were a representative, but far from extensive, overview of the IAM challenges faced at IST and how FenixEdu Connect would help to solve them. Naturally, the commercial nature

of the closed-source products requires them to support the latest standards and protocols, which has shown to be true. However they lack, in many instances, the flexibility to allow developers to extend the built-in behavior to fit their individual needs, which is exactly one of Connect's design goals. The analysed products' pricing scheme follows a subscription model which can range from 2 to 10$/user/month.

4.2.8 Rollout

The rollout of a complete IAM solution requires careful planning to ensure the least amount of downtime and reduce the attack surface stemming from incorrect application configurations. As such, we purpose a rollout plan consisting of four deployment phases that should ensure the gradual transition from the existing ID system to FenixEdu Connect while still allowing for gradual improvements and security fixes to be applied without major system disruption.

4.2.8.1 Phase 1: Proof of concept

The initial rollout phase of FenixEdu Connect at Instituto Superior Técnico is designed to validate the performance profile of the solution under real load conditions. This step is expected to provide valuable insights on possible optimizations while also limiting the extent of the damages brought on by any unidentified security vulnerabilities that may be disclosed during this trial period. While the final list of applications to be enrolled in this phase would always be subject to approval from the IT department, both the Group Server and Ticketing applications are prime candidates for the trial phase, due to the limited amount of personal data they hold. During this period FenixEdu Connect would run alongside the existing authentication solution, TécnicoID, feeding its user information of the university's LDAP directory.

4.2.8.2 Phase 2: General rollout

Once the required performance optimizations were identified and carried out the remaining applications can be switched over to FenixEdu Connect. The development of a Bennu Authentication Provider module would instantly allow all Bennu-based applications (such as the FenixEdu suite of products and the DOT applications) to use Connect, without any further modifications to their codebases. SAML integrations, including Office365, OpenStack or the Portuguese Citizen Card would also be instantly supported by FenixEdu Connect, since it exposes a SAML 2.0 IdP. The remaining applications that rely on CAS to perform authentication would have to be refactored to use SAML, OpenID Connect, or consume Connect's authentication API directly. At this point, Connect should be used by the majority of the active applications at IST. TécnicoID would still be available to support legacy applications, but it should be limited to serving requests from those products.

4.2.8.3 Phase 3: Migration of legacy systems

The third deployment phase would consist of the identification of the legacy applications that cannot be directly ported to FenixEdu Connect and the development of auxiliary systems to overcome this

challenge. Solutions may include the development of additional authentication modules, such as support for Kerberos[2] or NTLM[3]. However, the high adoption rate of open source software at Instituto Superior Técnico should severely limit the number of applications at this stage.

4.2.8.4 Phase 4: Integration with external entities

The last phase would allow Connect to act as the authentication provider for external systems, that are rarely included in IAM solutions:

- The implementation of a wrapper around FreeRadius[4] would allow Connect to take on the role of authenticating network access to IST's wireless network, eduroam.
- An integration module could be developed to allow Connect to interact with the campi's physical security system. This would allow, for example, to have a guest user's account provisioned with privileges to access a certain room for a limited period of time from day zero. As more rooms have networked locks installed, so do the inherent advantages of having Connect handle this system. A simple integration with FenixEdu Gears would allow, for instance, to setup an approval process (consisting of multiple authorization steps by different people) to grant access to a room. For sensitive areas (such as a vault or a data center), two-factor authentication (or even the combined authentication of two or more people) could be implemented with the effort being limited to the development of a connector module to act as the interface between FenixEdu Connect and the target system.

[2] https://web.mit.edu/kerberos/
[3] https://msdn.microsoft.com/en-us/library/cc236699.aspx
[4] https://freeradius.org/

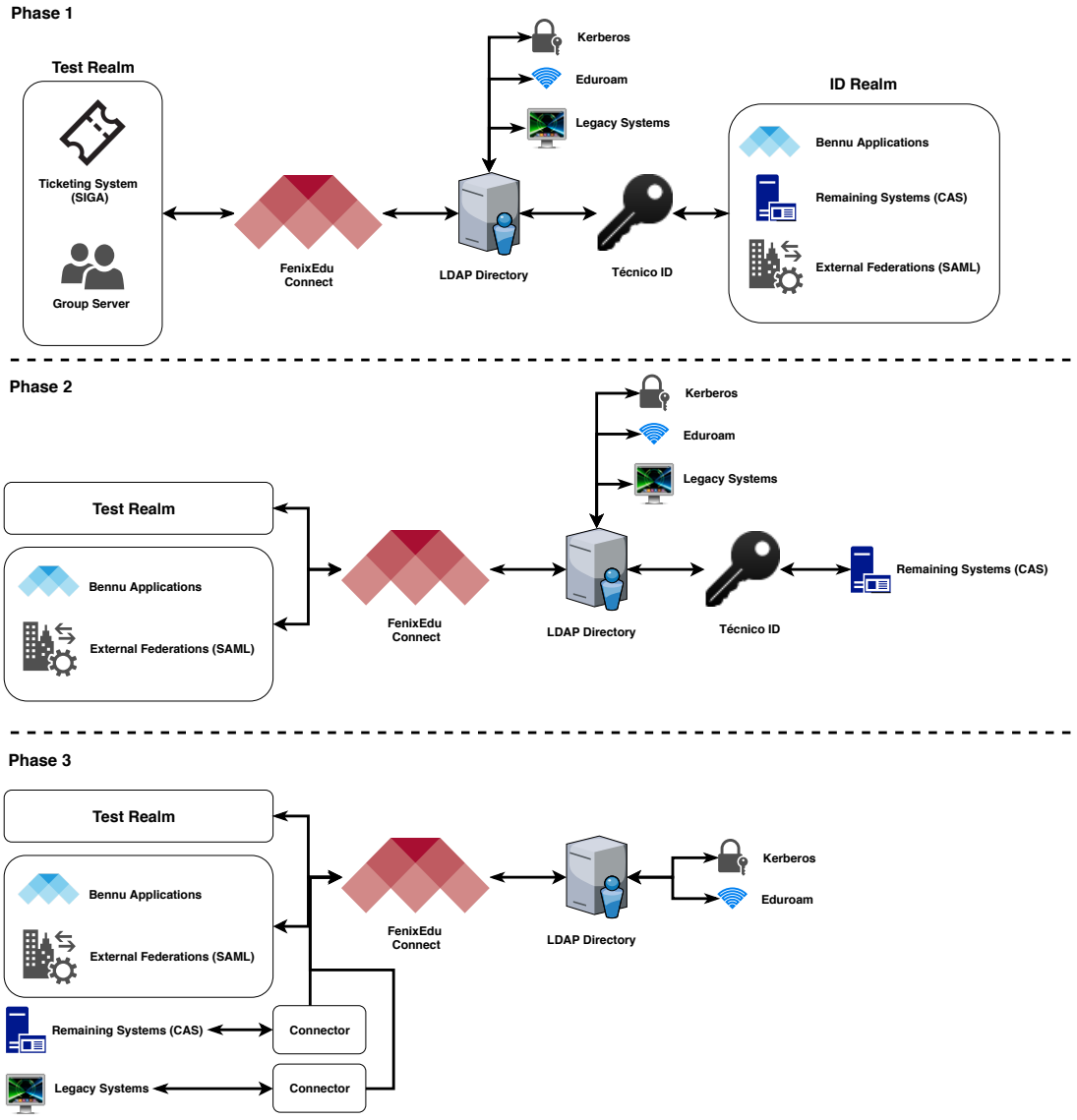


Figure 4.5: Evolution of FenixEdu Connect's deployment at IST.

Chapter 5

Conclusions

5.1 Conclusions

The constant pressure of migrating an ever increasing number of services to the virtual world has left organizations struggling with ways to transition existing structures and business practices while still ensuring the same level of availability, security and accountability offered by their offline counterparts. IT architectures who were previously limited to serving a small subset of users safely shielded behind enterprise firewalls had to be reimagined to accommodate the cross-domain nature of today's systems. Identity and Access Management systems, through the use of adaptive authentication strategies, clear authorization and access control rules and advanced reporting and auditing tools will be a decisive factor in determining the success of this transition, and ultimately the organization's ability to remain competitive and relevant in a virtual world.

Commercial vendors have been taking advantage of the demand for IAM solutions by offering tailor-made solutions based on the client's specific needs, often in per-user subscription model. While most of these solutions support the latest authentication and access control protocols the budgetary constraints faced by academic organizations often prevent them from taking advantage of these services. In addition, closed-source solutions inherently lack the flexibility to be extended and adapted to the organization's actual needs.

This thesis set out to design and implement the foundations for a customizable Identity and Access Management system that was able to meet the requirements of a wide array of academic institutions. While the initial requirement analysis was heavily inspired by the specific challenges faced at Instituto Superior Técnico, its architecture was designed around the premise of ease of extensibility, limiting the opinionated decisions to no more than a few sensitive defaults whenever it was justified. Both small to medium HEIs should be able to customize the final product to fit their specific requirements.

The motivation behind this product was never to develop an entire new authentication technology or protocol. Rather, it focused on solving a real-world need of academic institutions: to reliably authenticate its users across a variety of channels and ensure all applications have secure access to their identity. Its outcome is thus, two-fold: from a theoretical standpoint, it provided the community with a framework for

the design of extensible IAM solutions, capable of supporting the custom business practices in place at each institution. From a practical standpoint, it provided a working solution for how such systems could be implemented.

As mentioned in section 3.3, the final product still requires a frontend layer to be implemented on top of the provided Connect Core. This work is already underway at IST's IT department with the first trials of the complete Connect solution scheduled to begin in early 2019.

5.2 Future Work

The cross-cutting nature of IAM systems directly translates in the impact these solutions have in the entire organization's IT infrastructure. The concrete implementation provided with this thesis focused on providing a functionally-ready solution that can be used in the initial system trials at IST. However, as it is expected of a solution that is responsible for protecting a wide range of personal information, FenixEdu Connect must still undergo a thorough security validation by IST's IT department before it can be used in a production environment. Following this, future developments should occur in two concurrent vectors:

- **Fostering the adoption of the system by third party developers:** This can be accomplished by developing Connector modules for the most common frameworks and / or programming languages. In the presence of a simple solution to authenticate with FenixEdu Connect developers may prefer it over a custom authentication scheme, with the added benefit of being able to take part in Connect's SSO realm.
- **Extend the system's supported authentication factors:** Connect's vision is one where it is the single authentication and identity provider of each implementing institution. While the provided solution focused on implementing support for the most common authentication protocols, there is still room for the development of additional features, such as the ability to accept CAS or Kerberos authentication.

Bibliography

[1] M. Stieninger, D. Nedbal, W. Wetzlinger, G. Wagner, and M. A. Erskine. Impacts on the organizational adoption of cloud computing: A reconceptualization of influencing factors. *Procedia Technology*, 16:85 – 93, 2014. ISSN 2212-0173. doi: https://doi.org/10.1016/j.protcy.2014.10.071. URL http://www.sciencedirect.com/science/article/pii/S2212017314002989. CENTERIS 2014 - Conference on ENTERprise Information Systems / ProjMAN 2014 - International Conference on Project MANagement / HCIST 2014 - International Conference on Health and Social Care Information Systems and Technologies.

[2] U. Habiba, R. Masood, M. A. Shibli, and M. A. Niazi. Cloud identity management security issues & solutions: a taxonomy. *Complex Adaptive Systems Modeling*, 2(1):5, Nov 2014. ISSN 2194-3206. doi: 10.1186/s40294-014-0005-9. URL https://doi.org/10.1186/s40294-014-0005-9.

[3] I. Indu, P. R. Anand, and V. Bhaskar. Identity and access management in cloud environment: Mechanisms and challenges. *Engineering Science and Technology, an International Journal*, 21 (4):574 – 588, 2018. ISSN 2215-0986. doi: https://doi.org/10.1016/j.jestch.2018.05.010. URL http://www.sciencedirect.com/science/article/pii/S2215098617316750.

[4] I. S. Técnico. The fenixedu project: an open-source academic information platform, March 2001.

[5] K. Shu, S. Wang, J. Tang, R. Zafarani, and H. Liu. User identity linkage across online social networks: A review. *SIGKDD Explor. Newsl.*, 18(2):5–17, Mar. 2017. ISSN 1931-0145. doi: 10.1145/3068777.3068781. URL http://doi.acm.org/10.1145/3068777.3068781.

[6] J. L. Camp. Digital identity. *IEEE Technology and Society Magazine*, 23(3):34–41, Fall 2004. ISSN 0278-0097. doi: 10.1109/MTAS.2004.1337889.

[7] P. Windley. *Digital Identity*. O'Reilly Media, Inc., 2005. ISBN 0596008783.

[8] Microsoft Corporation. Identity and access management, 2004. https://msdn.microsoft.com/en-us/library/aa480030.aspx, Last accessed on 2018-09-04.

[9] R. Wash, E. Rader, R. Berman, and Z. Wellmer. Understanding password choices: How frequently entered passwords are re-used across websites. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 175–188, Denver, CO, 2016. USENIX Association. ISBN 978-1-931971-31-7. URL https://www.usenix.org/conference/soups2016/technical-sessions/presentation/wash.

[10] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems, 1975.

[11] D. Ferraiolo and R. Kuhn. Role-based access control. In *In 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[12] C. Emig, F. Brandt, S. Kreuzer, and S. Abeck. Identity as a service – towards a service-oriented identity management architecture. In A. Pras and M. van Sinderen, editors, *Dependable and Adaptable Networks and Services*, pages 1–8, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-73530-4.

[13] Cypher Research Laboratories. A brief history of cryptography, 2006. `http://www.cypher.com.au/crypto_history.htm`, Last accessed on 2018-09-04.

[14] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976. ISSN 0018-9448. doi: 10.1109/TIT.1976.1055638.

[15] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996. ISBN 0849385237.

[16] S.-T. Sun and K. Beznosov. The devil is in the (implementation) details: An empirical analysis of oauth sso systems. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 378–390, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1651-4. URL `http://doi.acm.org/10.1145/2382196.2382238`.

[17] E. Hammer-Lahav. The OAuth 1.0 Protocol. RFC 5849, April 2010. URL `https://tools.ietf.org/html/rfc5849`.

[18] D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, October 2012. URL `https://tools.ietf.org/html/rfc6749`.

[19] Facebook. Facebook: Access tokens - facebook login - documentation, 2017. URL `https://developers.facebook.com/docs/facebook-login/access-tokens/`.

[20] Google. Using oauth 2.0 to access google apis — google identity platform, 2016. URL `https://developers.google.com/identity/protocols/OAuth2`.

[21] Twitter. Twitter: Oauth twitter developers, 2017. URL `https://dev.twitter.com/oauth`.

[22] S.-T. Sun, Y. Boshmaf, K. Hawkey, and K. Beznosov. A billion keys, but few locks: The crisis of web single sign-on. In *Proceedings of the 2010 New Security Paradigms Workshop*, NSPW '10. ACM, 2010. ISBN 978-1-4503-0415-3. URL `http://doi.acm.org/10.1145/1900546.1900556`.

[23] S.-T. Sun. Simple but not secure: An empirical security analysis of oauth 2.0-based single sign-on systems. 01 2018.

[24] S.-T. Sun, E. Pospisil, I. Muslukhov, N. Dindar, K. Hawkey, and K. Beznosov. What makes users refuse web single sign-on?: An empirical investigation of openid. In *Proceedings of the Seventh*

Symposium on Usable Privacy and Security, SOUPS '11, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0911-0. URL http://doi.acm.org/10.1145/2078827.2078833.

[25] T. O. Foundation. OpenID Authentication 2.0. Specification, OIDF, December 2007. URL http://openid.net/specs/openid-authentication-2_0.html.

[26] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. OpenID Connect Core 1.0. Specification, OIDF, November 2014. URL http://openid.net/specs/openid-connect-core-1_0.html.

[27] N. Sakimura, J. Bradley, M. Jones, and E. Jay. OpenID Connect Discovery 1.0. Specification, OIDF, November 2014. URL http://openid.net/specs/openid-connect-discovery-1_0.html.

[28] M. Jones, J. Bradley, and N. Sakimura. JSON Web Token (JWT). RFC 7519, IETF, May 2015. URL https://tools.ietf.org/html/rfc7519.

[29] S. Peyrott. *The JWT Handbook*. Auth0, 2016.

[30] M. Jones, J. Bradley, and N. Sakimura. JSON Web Signature (JWS). RFC 7515, IETF, May 2015. URL https://tools.ietf.org/html/rfc7515.

[31] M. Jones and J. Hildebrand. JSON Web Encryption (JWE). RFC 7516, IETF, May 2016. URL https://tools.ietf.org/html/rfc7516.

[32] M. Jones. JSON Web Key (JWK). RFC 7517, IETF, May 2015. URL https://tools.ietf.org/html/rfc7517.

[33] Organization for the Advancement of Structured Information Standards. Security assertion markup language (saml) v2.0, 2005.

[34] Okta. Okta. https://www.okta.com. Accessed: 18-12-2017.

[35] S. Consortium. Shibboleth. https://www.shibboleth.net/. Accessed: 18-12-2017.

[36] E. Emandii. Authentication - threats and countermeasures. *Mircea cel Batran, Naval Academy Scientific Bulletin*, XIX(1):4, Nov 2016. doi: 10.21279/1454-864X-16-I1-063. URL https://doi.org/10.21279/1454-864X-16-I1-063.

[37] E. Ikhalia and C. Imafidon. The need for two factor authentication in social media. In *Proceedings of the International Conference on Future Trends in Computing and Comunication*, FTCC 2013, pages 76–82, 2013. doi: 10.3850/978-981-07-7021-1_16.

[38] D. M'Raihi, S. Machani, M. Pei, and J. Rydell. TOTP: Time-Based One-Time Password Algorithm. RFC 6238, IETF, May 2011. URL https://tools.ietf.org/html/rfc6238.

[39] S. S., B. D., and T. E. Fido universal 2nd factor (u2f) overview, version v1. 0-rd-20140209, 2014. URL https://fidoalliance.org/specs/fido-u2f-v1.0-rd-20140209/fido-u2f-overview-v1.0-rd-20140209.pdf.

[40] K. Sælensminde and V. Boonjing. A simple password less authentication system for web sites. In *2010 Seventh International Conference on Information Technology: New Generations*, pages 132–137, April 2010. doi: 10.1109/ITNG.2010.154.

[41] M. Jones, J. Bradley, M. Machulak, and P. Hunt. OAuth 2.0 Dynamic Client Registration Protocol. RFC 7591, July 2015. URL `https://tools.ietf.org/html/rfc7591`.

[42] M. Jones and D. Hardt. The OAuth 2.0 Authorization Framework: Bearer Token Usage. RFC 6750, October 2012. URL `https://tools.ietf.org/html/rfc6750`.

[43] J. Richer. OAuth 2.0 Token Introspection. RFC 7662, October 2015. URL `https://tools.ietf.org/html/rfc7662`.

[44] M. Jones and J. Bradley. OpenID Connect Back-Channel Logout 1.0 - draft 04. Specification, OIDF, January 2017. URL `http://openid.net/specs/openid-connect-backchannel-1_0.html`.

[45] I. S. Técnico. Mobility and transports. `https://tecnico.ulisboa.pt/en/campus-life/services/mobility-and-transports/`. Accessed: 2019-10-14.

Appendix A

API Endpoints

The following section outlines the complete list of API endpoints exposed by FenixEdu Connect.

Note: The base path (`/api/v1/`) has been omitted for brevity.

Authentication

Method	Path	Description
POST	`/auth`	Performs primary authentication
GET	`/auth/factors`	Returns the available MFA factors
GET	`/users/{userId}/factors`	Returns the user's enrolled MFA factors
POST	`/users/{userId}/factors`	Enrols a new MFA factor
DELETE	`/users/{userId}/factors`	Removes an enrolled MFA factor
POST	`/users/{userId}/factors/{factorId}/activate`	Activates an enrolled MFA factor
POST	`/users/{userId}/factors/{factorId}/startVerification`	Begins the verification transaction of an enrolled MFA factor
POST	`/users/{userId}/factors/{factorId}/verify`	Concludes the verification transaction of an enrolled MFA factor
POST	`/auth/magicLink`	Requests a magic link
POST	`/auth/magicLink/{code}`	Redeems a magic link

Session Management

Method	Path	Description
GET	`/users/{userId}/authorizations`	Lists the user's active sessions
DELETE	`/users/{userId}/authorizations`	Revokes all active sessions
DELETE	`/users/{userId}/authorizations/{sessionId}`	Revokes a session

Account Management

Method	Path	Description
GET	`/users`	Returns a list of all users
POST	`/users`	Creates a new user account
POST	`/users/invite`	Creates a new user account invite
GET	`/users/current`	Returns the authenticated user
GET	`/users/find`	Performs a user search
PUT	`/users/{userId}/credentials/password`	Updates the user's password
POST	`/users/{userId}/credentials/reset_password`	Requests a password reset
POST	`/users/{userId}/credentials/reset_password/{code}`	Concludes a password reset
PUT	`/users/{userId}/username`	Updates the user's username
PUT	`/users/{userId}/roles`	Updates the user's roles
POST	`/users/{userId}/lifecycle/activate`	Activates an account
POST	`/users/{userId}/lifecycle/suspend`	Suspends an account
POST	`/users/{userId}/lifecycle/reactivate`	Reactivates an account
POST	`/users/{userId}/lifecycle/deprovision`	Deprovisions an account
POST	`/users/{userId}/lifecycle/deprovision`	Deprovisions an account

Trusted Applications

Method	Path	Description
GET	`/trusted-applications`	Returns a list of all trusted applications
POST	`/trusted-applications/`	Create a new trusted application
DELETE	`/trusted-applications/`	Remove an existing trusted application
PUT	`/trusted-applications/{appId}`	Regenerate API key for a trusted application
POST	`/trusted-applications/{appId}/connectors/saml`	Create a SAML connector for a trusted application
GET	`/sync/keys`	Sync signing keys
GET	`/sync/token/revoked`	Sync revoked tokens

OAuth

Method	Path	Description
GET	`/oauth/applications/all`	Returns a list of all registered applications
GET	`/oauth/applications`	Returns the authenticated user's applications
POST	`/oauth/applications`	Creates a new OAuth application
PUT	`/oauth/applications/{appId}`	Updates an OAuth application's data
POST	`/oauth/applications/{appId}/scopes/{scopeId}`	Adds a scope to an application
DELETE	`/oauth/applications/{appId}/scopes/{scopeId}`	Removes a scope from an application
POST	`/oauth/applications/{appId}/grant/{grantId}`	Adds a grant type to an application
DELETE	`/oauth/applications/{appId}/grant/{grantId}`	Removes a grant type from an application
GET	`/oauth/scopes`	Returns the available scopes
POST	`/oauth/scopes`	Creates a new scope
PUT	`/oauth/scopes/{scopeId}`	Edits an existing scope
DELETE	`/oauth/scopes/{scopeId}`	Removes an existing scope
POST	`/oauth/authorize`	Get data for authorization request form
POST	`/oauth/confirm_access`	Authorize an OAuth application
POST	`/oauth/token`	Exchange a grant for OAuth credentials