POLITECNICO DI TORINO

Master Degree Course in Computer Engineering

Master Degree Thesis

# Enhanced attribute retrieval and provisioning through the eIDAS digital identity infrastructure

**Supervisors**
Prof. Antonio Lioy
Assist. Prof. Diana Berbecaru

**Candidate**
Sahar saadatmandi

July 2023

*To my family who always encouraging me pursue my aspirations.*

*To my love who his unwavering support has meant the world to me.*

# Summary

This thesis explores the integration of eIDAS and OAuth 2.0 frameworks for secure digital identity verification, access control, and attribute retrieval. eIDAS, a European Union regulation, establishes a standardized framework for electronic identification and trust services, while OAuth 2.0 is an authorization framework widely used in web and mobile applications.

The combination of eIDAS and OAuth 2.0 enhances the security and functionality of digital services. eIDAS serves as the primary authentication mechanism, while OAuth 2.0 provides a standardized protocol for secure authorization and access delegation. This integration enables controlled access to user resources while leveraging secure identity verification.

The thesis extends the eIDAS node's Specific part to support additional attribute retrieval. It introduces the AP Connector interface within the IdP Proxy, implementing AP-Proxy and AP-OAuth2 versions. These connectors integrate OAuth 2.0 with the eIDAS network, facilitating attribute retrieval and transfer. Modifications are made to the IdP Proxy to retrieve additional attributes from a national Attribute Provider, enhancing authentication and authorization processes.

The thesis also emphasizes the integration of eIDAS and OAuth 2.0 with the SPID system in Italy, providing a standardized and secure digital identity system. SPID attributes are securely exchanged using eIDAS and OAuth 2.0, enhancing trust, security, and interoperability within the system.

Additionally, the thesis highlights the role of JSON Web Tokens (JWT) in facilitating information exchange within the system. JWT is a compact format used for securely transmitting information as a JSON object, commonly used in authentication and authorization scenarios.

# Acknowledgements

I would like to express my sincere gratitude to Prof. Diana Berbecaru and Cesare Cameroni for their invaluable support and guidance throughout the development and writing of this thesis. Their expertise, patience, and availability have been instrumental in shaping the direction of this research and enhancing its quality. I am truly grateful for their mentorship and the knowledge they have imparted to me during this process. Their contributions have greatly enriched my understanding and have been crucial in the successful completion of this thesis.

I would like to express my heartfelt gratitude to my sister. Your constant encouragement and unwavering support have played a vital role in shaping my journey. Without your push, I would not have reached the heights I have achieved today. Thank you for always believing in me, motivating me, and pushing me to pursue my goals. Your presence in my life has been truly invaluable.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Authorization rules are a fundamental aspect of Identity and Access Management (IAM) within computer systems. IAM encompasses various practices and technologies that assist system managers in controlling user access to system resources and establishing client privileges. The purpose of authorization, within the IAM framework, is to determine and enforce the permissions and privileges granted to individuals or entities, ensuring that they can access the appropriate resources.

To better comprehend the concept of authorization and its application in computer systems, real-world examples can be useful in illustrating its meaning and usage. One such example is owning a house. As the owner, you have unrestricted access to the house, which represents the resource. However, as the owner, you also possess the authority to grant permission to others to use it. In this context, you can be seen as the authority figure granting individuals the authorization to access the house. By using this straightforward example, we can further explore the idea of authorization within the context of permissions.

For instance, entering the house is a permission, representing an activity that individuals are allowed to carry out on the resource. Additionally, other activities related to the house, such as furniture arrangement, cleaning, or repairs, may require specific approval or permission. These activities require additional permissions, and when granted, they become privileges or rights associated with those permissions.

However, it's important to note that individuals seeking permission to perform certain actions may need your consent or approval beforehand. Continuing with the house ownership analogy, consider the scenario where you hire an interior decorator to furnish your home. In this case, the action the decorator wants to carry out in your home represents the scope of the permission request. Before granting permission, you may need to provide your consent for the decorator to proceed.

The connection between identification and authorization can also be observed in certain instances. Let's consider the boarding procedure for a plane. According to your boarding pass, you are permitted to board the aircraft. However, the gate agent won't allow you to board solely based on the boarding pass. In addition to the boarding pass, you must possess a passport with an identity page. The gate agent verifies whether the name on your passport matches the name on your boarding pass before granting you access to the plane.

In the context of authorization, your name serves as an attribute of your identification. Other attributes, such as age, linguistic ability, credit card details, or any other relevant information, may be considered depending on the specific situation. Your name on the passport serves as a claim or declaration that you possess that particular attribute. The trust placed in the government that issued your passport ensures that anyone who reads your name from it can be certain that it belongs to you.

In the boarding process, the boarding pass functions as an "access token" that combines access rights to board the aircraft with evidence of customers' identities. It serves as a form of authentication and authorization, ensuring that only individuals with valid boarding passes and matching identification are granted access.[3]

Expanding beyond these real-world examples, the eIDAS network provides a standardized framework for electronic identification and trust services across the European Union (EU). It aims to ensure secure and reliable digital transactions while reducing costs and risks associated with additional verification procedures. The eIDAS network allows individuals to authenticate with government-issued electronic identities, transferring core personal attributes to service providers (SPs). However, for long-term applications or specific use cases, additional attributes may be required, and the eIDAS network allows for the request and authorization of these additional attributes. By leveraging authorization mechanisms within the eIDAS network, SPs can establish a higher level of trust and security for transactions, eliminating the need for separate verification procedures to collect additional data.

In summary, authorization rules within IAM play a crucial role in limiting user access to system resources and establishing client privileges. Real-world examples, such as owning a house and the boarding process for a plane, help illustrate the concepts of permission, privilege, scope, identification, and attributes within the context of authorization. Additionally, the eIDAS network offers a standardized framework for electronic identification and trust services, ensuring secure and reliable digital transactions. By implementing effective authorization mechanisms, organizations can ensure the secure and controlled access to their IT systems, protecting sensitive resources from unauthorized use while enabling trustworthy digital interactions.

# Chapter 2

# Background

In today's digital age, electronic transactions are becoming more and more common. However, with this increase in online activity comes an increased need for security measures to ensure the safety and privacy of personal information. One such measure is the use of electronic identification, authentication, and trust services, commonly referred to as **eIDAS**.

## 2.1 What is eIDAS?

**eIDAS** is a framework for **E**lectronic **I**dentification, **A**uthentication and Trust **S**ervices that is recognized across the European Union (EU). The framework is designed to enable secure and convenient access to digital services by establishing a legal and technical framework for electronic identification and trust services.

The eIDAS Regulation was introduced in 2014 to replace the previous EU regulation on electronic signatures. The regulation applies to all EU member states and aims to ensure the seamless use of electronic identification and trust services across the EU.

The eIDAS Regulation aims to provide a harmonized legal framework for electronic identification and trust services across the EU. The regulation establishes a network of trusted service providers (TSPs) that offer electronic identification and trust services to users. This network is known as the eIDAS network.[4]

### 2.1.1 Introduction

Nowadays, using the internet for an expanding range of services is standard practice. However, robust and user-friendly authentication and identification are required to securely access remote services in various domains (such as health, finance, or academia) in order to prevent attacks like data leakage or identity theft. The European Member State (MS) nations have improved their eID management methods with this end in mind since the late 1990s. A digital representation of a natural or legal person is known as an electronic (or digital) identity.[1] The European governments have begun issuing government eID credentials to citizens, such as national eID cards with a digital (public key) certificate on-board, to enable residents to confirm who they are (with high confidence level) in public or private services [5]-[6].

After completing numerous identification processes, such as reviewing national registers or identity documents, these credentials are granted. As a result, they offer a high level of assurance regarding a person's identification. There are now credentials that are easier to use, like one-time passwords that are combined with personal devices [7]. Citizens can access a Service Provider (SP) by using these credentials to authenticate at an Identity Provider (IdP), which may give some basic identification information about the person. In this situation, bilateral agreements or national digital identification systems are used to establish trust between the IdP and SP. As an illustration, organizations in Italy such as InfoCert [8] and Poste Italiane [9] provide authentication credentials that are accepted by the SPID (Sistema Pubblico di Identità Digitale, or Public System for Digital Identity) [10] national system.

The citizens might use these credentials to gain access to public or private services that support SPID, such registering kids for school or filing taxes. The mobile sector made a substantial contribution to the development of digital identity systems in other nations, including Estonia, Finland, Norway, and Switzerland [11]. The European eIDAS network integrating the eID systems of many EU countries has been established and put into place to enable mutual recognition of eIDs and to promote people' access to foreign services with national credentials. By allowing cross-border authentication with national credentials for both legal and natural people [12], this network complies with the eIDAS Regulation 910/2014 [13]. It takes advantage of the SAML 2.0-based national eIDAS nodes that communicate using the eIDAS protocol [14]. The national identity infrastructure of each nation is interacted with by the eIDAS node utilizing certain technologies and protocols. The eIDAS network only transmits a limited set of attributes for natural persons known as the eIDAS Minimum Data Set (MDS) [14]. *FirstName*, *FamilyName*, *DateOfBirth*, and *PersonIdentifier* are the four required attributes in this set. BirthName, PlaceOfBirth, CurrentAddress, and Gender are the four optional attributes.

The SPs may utilize the eIDAS MDS properties in one-off services without any additional requirements. On the other hand, in addition to the MDS traits, the SPs also require other citizen attributes for long-term services. For identification matching purposes, they could want information about the nationality or photo, for instance [15]. Domain-specific information, such as professional credentials, financial standing, academic record, or present employment, is useful in delivering targeted services. Unfortunately, self-assessed user data cannot be trusted, so research is now being done on how to safely and privately collect such attributes from authoritative parties. More information could be retrieved by the eIDAS network from reliable national sources. The eIDAS nodes may facilitate the exchange of various attributes, following the eIDAS specification [14].

It is possible to establish new attributes, however those that are not listed in the eIDAS attribute profile [12] can call for a bilateral agreement. To put it another way, nations may select if they wish to improve the node so that they can exchange other attributes (in addition to the MDS) with other nations. The European Commission (EC) recently adopted the European Digital Identity framework [16], which emphasizes the need to expand the eIDAS network with additional cross-border recognized person identification data sets to support identity matching. Additionally, this EC report offers three options to support citizens' and businesses' use of electronic IDs and electronic attestation of the credentials and attributes associated with those eIDs.The attributes and credentials might theoretically be connected to the users' eIDs via a trust service provider or a digital wallet.

**Methodology**: In order to retrieve and send more properties across the eIDAS network, we have overcome a number of obstacles:

1. Support for attribute retrieval and new attribute addition on the eIDAS node. An key topic is deciding what kind of extra data (personal or domain-specific) and how much data the eIDAS network should transport. The eIDAS nodes must minimize the amount of data they retrieve (data minimization) and limit the types of data they gather (collection limitation) in accordance with the privacy by default principle [17]. We decided which extra academic and personal characteristics to make available on the eIDAS node. Then, we retrieved them from an Attribute Provider (AP) different from the IdP(s) connected to the eIDAS network using two methods, *AP Proxy* and *AP-OAuth2*.

2. AP Connector design and installation. The user agent (browser), the SPs providing services, the eIDAS nodes, the IdPs authenticating citizens with national credentials, and possibly the APs providing additional person or domain-specific attributes, are all involved in authentication and attribute retrieval via the eIDAS network. Interoperability is a difficult task in this situation [18]. Local entities favor "lighter" protocols like Open Authorization (OAuth) 2.0 [19] and more straightforward formats for attribute transfer, like JSON [20], for managing user consent and authorizing users. There are adapters in a logical AP Connector module to translate the eIDAS protocol messages to other particular protocols.

3. Privacy and user consent concerns. The user's consent is necessary for the data collection even when more data is enabled in the eIDAS network.Citizens must agree on the data to be retrieved and to whom it will be released, hence user consent management needs to be taken seriously. Users must not, however, find the user consent dialogs annoying or repetitious when it comes to their privacy. We examine user consent management, identifying various eIDAS process steps where consent is obtained. Additionally, the parties involved in the proposed AP Connector implementations handle user consent in a different way.

**Contribution**: Our primary contributions are (1) a detailed analysis of the (natural person) attributes supported by the eIDAS network currently, and justifications for supporting new ones to address user identification and service needs; (2) a discussion on the utilization of eIDAS MDS attributes in one-off and long-term services; (3) the presentation of two AP Connector models used by the eIDAS nodes for attribute retrieval; the description of two potential AP Connector implementations and their testing in experimental testbeds. The eIDAS node creates a direct HTTPS backend channel for communication with the AP in the first one, dubbed AP Proxy. The node in the second one, known as AP-OAuth2, takes advantage of the OAuth 2.0 protocol to authorize the disclosed attributes.

The government eIDs issued through schemes notified under eIDAS are supported by mutual cross-border recognition under the eIDAS Regulation. The nations who have notified their eID schemes must recognize the eIDs of those nations.

According to [16], the MS nations voluntarily notify the European Commission of their eID scheme(s). The European Commission then asks MS specialists to conduct a peer assessment of the scheme, determining if it complies with the standards outlined in the eIDAS Regulation, implementing acts, and guidelines [21]. The scheme is published on a special list of "notified eID schemes" after notification and peer assessment are complete [22]. For instance, the *SPID* system and the *CIE* (Carta d'Identità Elettronica) are two schemes that Italy has announced.

The eIDAS interoperability network, made up of national "eIDAS nodes," has been put up and is currently in use to support the mutual eID recognition in practice. The three levels of assurance (LoA) that eIDAS defines *low*, *considerable*, and *high* refer to a restricted, significant, or high level of confidence in the identification of a person being claimed or asserted. The procedures used to issue credentials to citizens, verify their identities, and manage credentials are all covered at the LoA level. The LoA levels are internally mapped by the nations into the national authentication credentials of their eID schemes.

In general, the LoA level is high when using a national smart card for authentication. The SPs must accept considerable and high levels of verification from citizens and may also accept lower levels of authentication.

## Privacy and User Consent Issue

The ability of citizens to generate, manage, and share information relating to their eID(s) is a key component of digital identity systems, and Satchell et al. [23] examined this topic. They generally desire to have many identities that may overlap and prefer to remain "anonymous" during transactions. Another study using three well-known digital identity providers (Google, Facebook, and Google+) found that most users (specifically, 399 out of 424) felt that having control over what information an IdP gave to an SP was "very" or "extremely important" [24].

In addition, 50% of the study's participants favored using numerous IdPs over a single one. Another noteworthy point raised by the same study is that the participants were unclear as to what data had been transmitted to the SPs. However, as more data was sent, they became aware that more attributes had been sent, even though they were unsure of which ones. The consent dialogs had little impact on their propensity to "log in," but privacy worries did. For instance, the majority of users reported feeling "uncomfortable" sending their friend list and images, whether they were given to a reputable website or not.

As a result, when more data travels across the eIDAS network, adequate care must be taken in choosing the requested attributes. The requirements for the service must be clearly identified and described. If too much information is requested, users may be reluctant to use the site because of privacy concerns. In order to provide users control over the spread of their personal information, Gomi created a framework for tracking the history of identity information transfers across various domains [25]. Other methods, like My Data [26], put the individual at the center of data use. MyData raises awareness of the value of personal data and encourages its more moral application. They offer technical guidelines as well as encouraging users' and businesses' consciousness.

The most recent EC report [27] examines how the eIDAS Regulation supports the needs for customer data portability and provides information on how the user may control their credentials through the verifiable claims that are emerging in various initiatives, including the European Blockchain Service Infrastructure. Taniguchi et al. in [28] offered a plan to use anonymity and pseudonymity to defend the privacy of digital identities. The concept is that while a person can act in an anonymous manner in most situations, they can reveal their identity in unusual circumstances.

### Identities and User Identifier(s)

User identities were divided into two categories by Bhargav-Spantzel et al. [29]: *weak* and *strong*. A population's members can all be identified by a weak identifier, whereas a strong identifier can be used to identify a single member of the population. The population size and the uniqueness of the identifying attribute determine whether an identifier is strong or weak. The same authors noted that a unique identification could result from combining many weak identities. This research proves that a number of traits are often required for person identification, which is helpful in the context of this study.explored the issue of homonyms (people with the same first and last names as well as the same birthdate) in the person identification procedure [1].

### Attribute Aggregation

Different models of attribute aggregation in federated identity management systems were examined by Ferdous and Poet [30]. They talked on the criteria for trust while modeling attribute aggregation on each side, taking into account the traditional actors in the federated architecture (SP, IdP, and the client). We pay attention to the "identity proxying model," in which the SP enables the user to combine attributes from many IdPs using a very reliable IdP (a sort of "super IdP").

In this model, the user is first sent to the trusted IdP, which then sends them on to additional IdPs. The user returns to the trusted IdP with an assertion containing the evaluated characteristics after being separately authenticated at each IdP. Finally, the trusted IdP combines the attribute values after retrieving and validating each assertion. The trustworthy IdP may supplement the combined set with its user characteristics before reasserting all user attributes to the SP. The SP and the trusted IdP have a trust relationship, and the SP is unaware of the other IdPs from which the attributes have been gathered. The Italian IdP Proxy (a component of the eIDAS node) described in this thesis and the trusted IdP in [30] have certain similarities. They both serve as trusted attribute aggregators and collectors. In contrast, the user only needs to authenticate (using eIDAS) once at an IdPs that uses a recognized eID scheme, and the SP creates a trust relationship with the local eIDAS node.

### Trust Models

The trust requirements of various identity management solutions were examined by Jsang et al. [31] using a condensed model made up of clients, service providers, and identifier & credential providers. The authors categorized the architectures into 4 categories based on how the aforementioned entities interacted: *isolated*, *federated*, *centralized*, and *personal*. The primary trust criteria for each architecture have been outlined. The eIDAS node, which actively connects with the other nodes and the national bodies, is a new addition to the eIDAS network compared to the earlier models.

### 2.1.2 Characteristics

The eIDAS network satisfies a number of crucial requirements, including security and decentralization. Additionally, it might expand to include more sectors in the future. The eIDAS network, which was first developed as a prototype within the framework of the STORK and STORK 2.0 [32]-[33] European-funded projects, is currently the de facto Pan-European eID interoperability framework connecting the digital identity systems of several EU countries to enable the integration of e-services in a variety of domains. [34]

#### Decentralization

The eIDAS network lacks a central (control or data storage) point. Through dedicated (national) eIDAS nodes that are part of a circle of trust, the authentication requests and responses are transmitted. The nodes communicate eIDAS metadata, also known as SAML metadata, on a bilateral basis for the trust establishment.

#### Security

Given that the many components of the eIDAS network are operated by different companies and that the network is decentralized, its overall security is not straightforward. To prevent such potential attacks, the eIDAS node operators (national agencies or public ministries) must adhere to tight cryptographic specifications [35] for eIDAS message protection and TLS channel formation. For instance, the eIDAS nodes are required to utilize cipher suites with forward secrecy and qualified X.509 certificates in the TLS connections [36].

The processing and protection of data transported through the eIDAS network must be sufficient [37]. "The security and privacy of the user identification information, both certified and uncertified, are of highest importance nowadays," as stated in [29], When natural person attributes are retrieved and transferred through the eIDAS nodes, security prevents theft and impersonation, while privacy guards against the attributes disclosure. "Node operators of eIDAS nodes shall prove that...the node fulfills the requirements of standard ISO/IEC 27001 by certification, by equivalent methods of assessment, or by complying with national legislation," states the eIDAS specification. In addition, "Nodes shall not keep any transaction data including personal data beyond that needed by Article 9(3) of [4]" for privacy concerns. In short, the eIDAS node operator must maintain (just) the data necessary to reconstruct the sequence of the message exchange in the case of an incident, allowing for the location and kind of the incident to be determined. This information consists of the eIDAS node's identifier, a message identification, and the date and time of the message. While there are hints for the security of messages sent to other eIDAS nodes, the protection of information sent to national SPs and IdPs is dependent on the country.

#### Cross-sectorial

The cross-sectorial functionality allows for the use of attributes from one (particular) domain that are made available over the eIDAS network in other application domains. For instance, a person's academic standing (such as that of a student) might be utilized in the public transportation sector services to gain customized discounts.

### 2.1.3 Components and Attributes of the eIDAS Node

The eIDAS Connector and the eIDAS Proxy Service are the two primary logical components of the eIDAS node. The nation(s) host one or more eIDAS Connector(s) that offer either public or private services, as well as one or more proxy services. Germany offers a Middleware program to be deployed and used on the eIDAS nodes of the other MS nations in place of a Proxy Service. The eIDAS authentication request (eIDAS-Auth-Req) is obtained from the national service provider and forwarded to the foreign eIDAS Proxy Service in the sending MS country by the (eIDAS) Connector in the so-called Receiving MS country. The national IdP(s) are contacted using MS-specific protocols when the eIDAS Proxy Service processes the eIDAS-Auth-Req, often by translating it into a request (in local format). When eIDs recognized by eIDAS are used for authentication, the IdP creates an authentication response in local format and sends it back to the neighborhood proxy service. The Proxy Service creates an eIDAS authentication response (eIDASAuth-Res) based on the received response and transmits it to the foreign Connector for further processing. If the SP supports the eIDAS protocol, the Connector may return the eIDAS-Auth-Res in eIDAS format; otherwise, it translates the answer into the particular protocol that the SP supports. In more detail, a Generic portion and a Specific part make up both the Connector and the Proxy Service. The Generic component enables eIDAS protocol connection with the counterpart nodes [14]. The Specific component is engaged in the eIDAS node's communication with the national SP(s) and IdPs. The list of requested characteristics is one of the contents of the eIDAS-Auth-Req. Eight MDS properties for natural individuals are defined by the eIDAS attribute profile [12] (Table 2.1), of which four are required, namely *PersonIdentifier*, *FamilyName*, *FirstName*, and *DateOfBirth*. The remaining four attributes are optional.

| Name | Description | Type |
|---|---|---|
| PersonIdentifier* | Unique identifier. | String |
| FamilyName* | Family name. | String |
| FirstName* | First name. | String |
| DateOfBirth | Date of birth. | Date |
| BirthName | First name or family name at birth. | String |
| PlaceOfBirth | Place of birth. | String |
| CurrentAddress | Current address. | String |
| Gender | Gender. | String |

Table 2.1. eIDAS natural person attributes.

### 2.1.4 Versions Of the Code and Running Environments

The eIDAS specification must be followed by the operating eIDAS nodes [14]. While some nations have created their own ad hoc implementation, others are using the eIDAS code(s) made available by the European Commission and modified in the Specific section. The eIDAS code version 1.4.x (branch) and the eIDAS code version 2.x (branch), both written in Java, have both been made available. We will only go into depth about the key changes between the two code branches because every release includes improvements.

The Specific components transfer messages between the format of the national eID scheme and the eIDAS format in the eIDAS code version 1.4.x. Through the eIDAS protocol, the Generic components are communicated with. The Generic and Specific components are separate services in this version, and they may even operate on different servers. Instead, the translation of authentication messages in the eIDAS code version 2.x is divided into two parts: the Specific part translates between the national eID scheme and an intermediate format, and the Generic part translates between the intermediate format and the eIDAS one. The Specific and Generic parts of the eIDAS 2.x version communicate with each other using a so-called lightweight protocol, and they are intended to run on the same server.

eIDAS code is often implemented in a variety of settings. While the experimental nodes are operating in test settings, the official nodes are accessible in production environments. Tests on a near-production setting are often conducted in the pre-production or Quality Assurance (QA) environment. The eIDAS code is used in many nations' production settings, and the majority of them have separate test and production environments. The eIDAS node's support for extra natural person traits has been put to the test in a pre-production setting. Only the eIDAS MDS properties are currently supported by the eIDAS nodes running in production environments.

### 2.1.5 Management of Trust

To maintain a continuous chain of trust, the eIDAS nodes are securely recognized using eIDAS metadata sharing [38]. There isn't a single trust anchor for all Member States, like at the website of the European Commission; rather, each MS acts as the trust anchor for its own eIDAS node. The MS nations trade bilaterally the trust anchors. To ensure that the signature on its eIDAS metadata file is authentic, each MS nation operating an eIDAS node securely distributes the eIDAS metadata signing certificate (CerteIDAS_metadata_sign). [38] provides a description of the processes utilized for eIDAS information sharing, verification, pre-fetching, and caching. The X.509 digital certificates required by the other nodes for eIDAS are contained in the eIDAS metadata file.

The X.509 digital certificates needed by the other nodes to decode the attribute values in the eIDAS-Auth-Res messages and to verify the signatures on the eIDAS-Auth-Req and eIDAS-Auth-Res messages are stored in the eIDAS metadata file. Each node's eIDAS metadata specifically includes the following information:

- the CerteIDAS_message_sign certificate required to validate the digital signatures on the eIDAS-Auth-Req and eIDAS-Auth-Res messages produced by that eIDAS node.
- the user characteristics that are encrypted in eIDAS-Auth-Res messages delivered to other eIDAS nodes using the certificate (CerteIDAS_attributes_encrypt).

### 2.1.6 A Short Summary of a few Notified eID Schemes

Presently, eIDAS supports cross-border authentication and identification using 19 notified eID schemes from 15 distinct MS countries [39]. We highlight a few of these schemes in this section, but [40] has a complete list of schemes together with the LoA levels of the credentials recognized by eIDAS. Be aware that certain nations (such as Slovenia or Austria) have not disclosed their plans for a variety of reasons that are mentioned in [16]. This fact means that other nations may, but are not required to, accept authentication using eIDs from non-notified scheme(s).

The first generation of eID cards were initially issued in Belgium in 2003. The deployment of the second generation of eID cards began in 2014, with an estimated 2 million cards issued annually [41]. In 2020, more than 2.5 million individuals would have utilized the mobile-based "itsme" authentication system [42], up from 49% of households in 2018 [43]. In the Czech Republic, owners of the national eID can use it to log into legal firms, online gambling and betting sites, and health insurance providers [16]. Around 4.7 million Danes use the NemID scheme for online banking authentication, resulting in more than 55 million transactions each month [44]. The national eID card is owned by 98% of Estonians, and 67% of them frequently use it [45]. In instance, 60% of eID card holders in 2018 utilized their cards at least once for authentication or signature [46]. Up to 2018, 53 million eID cards were distributed in Germany, but the goal is to reach all eligible citizens by 2020 [5]. More than 19.5 million Italians hold an electronic identity card, or CIE (Carta d'Identità Elettronica) [47]. Over 18 million SPID credentials have been issued in a short period of time, growing exponentially [48]. The use of eID cards is now voluntary in Latvia, but they will be required by 2023 [49].

The eID cards are optional and recommended to ID card applicants in Luxembourg as well [50]. Since Portugal's national eID cards (Carto de Citado) have been in use since 2008, about 45% of cardholders have activated the digital certificate needed for authentication and signature [6]. However, the 'Chave Mvel Digital' mobile authentication and signature solution is now more widely used, with 160,000 users in 2018 [7]. The rollout of eID cards in Slovakia began in December 2013 [51] and is expected to exceed 600,000 monthly authentications in 2019 [52]. In 2006, Spain started issuing eID cards (DNIe - Document Nacional de Identidad electronico), and in 2015 it started issuing DNIe 3.0, which included NFC functionality [53].

In the Netherlands, 13.8 million active users used the DigiD authentication system in 2018, resulting in more than 307 million authentications [54].

### 2.1.7 eID Identifiers

#### eIDAS Identifiers, Natural Persons, and Digital Identities

What exactly is a digital identity? We have taken into consideration the definitions supplied by ITU-T and the eIDAS Regulation in our work, despite the fact that there are other meanings for the word "digital identity" that exist. Digital identity is defined by the ITU-T Focus Group on Identity and Authentication as "mechanisms that assert and verify personal data attributes in the context of digital services and transactions, based on three processes: identification, authentication, and authorization." [11]. Three categories of digital identities *foundational*, *functional*, and *transactional* were identified by the ITU-T Group [11]. A foundational (core) digital identity is created as part of a national digital identity scheme or something comparable, and it is based on the "formal establishment of identity through the examination of qualifying (breeder) documents such as birth records, marriage certificates, and social security documents." The functional digital identity responds to the unique requirements of a certain industry, like healthcare. "Intended to facilitate the performance of financial or other transactions across several sectors," the transactional identity. According to the same assessment, state-issued eID serves as a solid, trustworthy fundamental identity. Additionally, it shows that there are two categories of attributes that can be used to define a person's digital identity: biographic attributes, such as name, age, and gender, and biometric attributes, such as fingerprints, iris texture, voice, or facial geometry. When civil registration systems are nonexistent or official birth certificates are not present, as is the case in underdeveloped nations, biometric features are essential to uniquely identifying a person [55].

Digital ID is "the process of using personal identification data in electronic form to uniquely represent either a natural or legal person, or a natural person representing a legal person," according to the eIDAS Regulation. Since the national eID schemes obtain the information needed to identify a person from national registries or official (identification) documents, eIDAS deals with biographic attributes for natural persons. We conclude that the eIDAS network deals with the transfer of biographic traits associated with fundamental digital identity in light of the aforementioned definitions.

eIDAS PersonIdentifier discussion: The eIDAS MDS characteristics are an element of a person's fundamental digital identity, with the exception of the eIDAS PersonIdentifier. The eIDAS PersonIdentifier, what about it? There is a specific syntax for this eIDAS characteristic, which is briefly given in (Table 2.2) [12]. With the exception of nationality codes, each MS country chooses how to construct it, therefore it is possible but not required that it be derived from the national identification. In fact, it may even be a transaction-specific pseudonym [56]. Multiple eIDAS PersonIdentifiers are possible for a natural person. These identification numbers are ensured to be exclusive in that no two individuals inside the EU may share the same eIDAS PersonIdentifier. We note that an individual's eIDAS PersonIdentifier is not often the same as his national identification number.

| Description | Example |
|---|---|
| Country's nationality code, followed by a slash | IT/ |
| Nationality code of the nation supplying the service, then a slash | AT/ |
| A string of legible characters that distinguishes the claimed identity in the country of origin | 02635542Y |

Table 2.2. An explanation of the eIDAS PersonIdentifier property.

Additional information might be required to provide the service when a citizen presents his eIDAS PersonIdentifier (along with his name, surname, and date of birth) to a foreign SP. The SP may request extra personal information from the citizen or require them to register with a national registry in order to get long-term services. Instead, for short-term (or one-off) services (as described in Section IV-B), the eIDAS MDS attributes may be deemed adequate to deliver the service. From a functional standpoint, the SPs demand that each European citizen have a single, unique, and (potentially) permanent person identification, comparable to those that are employed at the national level. Despite the fact that the eIDAS PersonIdentifier falls short of meeting all SP expectations, it has been regarded as a viable option. Even though the eIDAS PersonIdentifier falls short of meeting all SP standards, it has been accepted since each MS nation can determine the value as long as it continues to be unique throughout the EU. The following are some potential explanations: a) the eIDAS PersonIdentifier is derived from the national identifier, so its value would be as persistent as the national identifier; b) the national IdP(s) performing the eIDAS authentication assigns or derives it; and c) it is an eID pseudonym that changes from one transaction to the next.

### 2.1.8 A Service Using eIDAS

While other services need stringent identification of a person, some do not. For instance, being able to identify the natural person specifically is necessary for responsibility. We distinguish between two different categories of services: one-time and ongoing. We go over the criteria in terms of the necessary eIDAS properties for each type of service.

*One-Shot Services*: These services, like renting a bike or registering for ski passes, are only meant to be used once. The eIDAS MDS properties in such services have to be adequate to deliver the service. Because the SP may store the eIDAS MDS attributes and offer the service, the authentication through the eIDAS network, for instance, may allow a foreign citizen to reserve the bike in the case of a bike-sharing service. When age-based discounts are given (such as when issuing ski pass cards), the confirmed characteristic with the date of birth is sufficient. Based on the eIDAS MDS characteristics, the SP may request additional information on the citizen's identity in his country of origin in the event of a disagreement (for example, in the instance of bike damage). The organization that assigned the eIDAS PersonIdentifier may resolve potential homonyms, for instance, by searching its regional database or a national registration of local residents.

*Long-Term Services*: These services use the eIDAS MDS characteristics to deliver essential, reliable information about a person. In any case, the SP often needs more information for the service or must install further identification checks or processes before offering the service. For instance, the Tax Agency may need the foreign citizen to register on the specific online portal in order to file a tax return in an eIDAS-enabled service. As part of this procedure, the Tax Agency would generate an identifier (in the national format) for the foreign citizen, and the newly formed profile would include the eIDAS PersonIdentifier as well as the other information acquired from the eIDAS network. For foreign citizens, the whole data may be kept in a national registration.[1]

The "Login with eIDAS" service for academic staff is another illustration of a long-term service [19]. In this instance, the university (acting as SP) has already indexed the individuals based on their national identify and registered them in the internal database. The individual must link the current eIDAS PersonIdentifier to his national identifier in his profile in order to permit eIDAS authentication. The individual must update the identification in his profile if the eIDAS PersonIdentifier changes, as can occur, for instance, in Italy if a person authenticates with several SPID IdPs. Otherwise, the eIDAS authentication will fail. We note that the SPs frequently request identity matching for long-term services [15]. This calls on them to determine whether the person who provided their eIDAS network authentication matches one of the individuals who have previously been registered on their end. To do this, they might compare the data from the eIDAS MDS with the records or data stored locally in the SP's database or obtained from a centralized national source, such as the national civil registry. In spite of this, homonym or transliteration issues might result in false positives or false negatives for this method. When two people share the same name, last name, and birth date, they are said to be homophones. Furthermore, the SP operator may need to manually check for transliteration issues (i.e., minor discrepancies found in the registered names or surnames). This problem is more prevalent when surnames and given names contain accented or unique characters. The eIDAS MDS characteristics might be expanded to store additional identifying information, such as the passport or eID card number, a picture, or the European Health Insurance Card number, in order to enhance identity matching at the SP.

## 2.2 Connecting Attribute Providers to the eIDAS Network

Services typically require more information about people, such as their profession (teacher, doctor), position within a company (manager, director), or country of origin. Such characteristics may be retrieved by the eIDAS node from either a specific AP or an IdP serving as an AP. In our work, we only take into account the APs in the citizen's home nation where he has successfully authenticated using his eIDAS eID.

Additionally, the APs might also need to perform identity matching. The eIDAS MDS properties by themselves might not be adequate due to homonym and transliteration issues, hence the AP requires additional attributes to prevent instances where a person's profile is mapped to someone else's or when access to his profile is prohibited. The AP often demands information like a Tax Reference number, a Passport or ID card number, a combination of them, and a unique national identification.

### 2.2.1 Attributes Classification

Due to the possibility that eIDAS nodes may obtain characteristics from several APs, we addressed attributes categorization in our study. We categorize the attributes' properties into two groups: (1) general properties, which define the attributes' innate features; and (2) attribute value properties, which provide information on how the attributes are valued. The attributes' general characteristics include, for instance: (a) Category, which can be personal or industry-specific; (b) Persistence (permanent, non-permanent), depending on whether its value may change over time (for the same person); and (c) Strength (strong or weak, where a strong attribute can be used to uniquely identify a user while many different weak attributes may be required at the same time to lower the risk of incorrect identification).

For instance, although the passport number is personal, non-permanent, and weak, the Italian fiscal number also known as a "tax reference number" or "codice fiscale" is powerful, permanent, and personal. Attribute Level of Assurance (ALOA) and Source of Authority (SoA) are two examples of attribute value characteristics. Different ALOA levels (low, medium, or high) could be defined in the future for the degree of credibility of the attribute values. Instead, the SoA offers details on the organization (entity) held and responsible for an attribute value. The URI that may be used to access the attribute value and the name of the entity or authority are two examples of the subparts that, in our opinion, make up the SoA.[1]

### 2.2.2 AP Connector Models

We developed new attributes for the eID4U project [57] (Table 2.3) and a logical AP Connector component to retrieve the new characteristics from the APs [58]. This component, which use several technologies to connect to the national APs and the eIDAS nodes, can be operated by many organizations. In the eID4U project, there are now two AP Connector models [59]-[60]. In the first, the AP Connector interacts with the Specific part of the eIDAS Proxy Service to carry out attribute processing, such as attribute retrieval, filtering or aggregation with other attributes valued by the IdP, and conversion into eIDAS format. The AP Connector in this paradigm can either be a standalone element outside the node or a part of the Specific section of the eIDAS Proxy Service. In a variation of this model, the AP Connector communicates with the eIDAS Proxy Service as a whole rather than just with its Specific component. MS nations that have created and implemented their own code for the eIDAS node may use this solution. As a result, they might not have distinct pieces for the generic and specific components. The second model does not either collect attributes or translate the attributes to/from the eIDAS format to national formats; instead, it just minimally modifies the eIDAS node to accommodate new characteristics and transfer them to the counterpart nodes. In this instance, the eIDAS node does not transform the messages into any other particular forms and is agnostic to the attribute names (formats) used inside the nation. The IdP primarily handles the aforementioned functions while one or more APs provide extra qualities. In this instance, the AP Connector is positioned between the IdP and the AP.Keep in mind that the IdP's contact with the national APs is MS-specific and can be carried out in a variety of ways. For instance, a protocol based on SAML 2.0 might be utilized by both the IdP and the APs.

| **eIDAS attributes** | **Description** | **Type** |
|---|---|---|
| TaxReference | Fiscal number | String |
| IdType | Passport or national identity | String |
| IdNumber | Document ID number | String |
| IdIssuer | Document issuer | String |
| IdExpiryDate | Date in the format YYYY-MM-DD | Date |
| EhicId | European Health Insurance Card ID | String |
| Nationality | ISO 3166-1 alpha-2 code of the country | String |
| Citizenship | ISO 3166-1 alpha-2 code of the country | String |
| MaritalState | Marital status | String |
| CountryOfBirth | ISO 3166-1 alpha-2 code of the country | String |
| CurrentPhoto | Picture | Document |
| TemporaryAddress | Current address | String |
| Email | Email address | String |
| Phone | Phone number | String |
| HomeInstitutionName | Name of the home institution | String |
| HomeInstitutionIdentifier | Erasmus code of the home institution | String |
| HomeInstitutionCountry | ISO 3166-1 alpha-2 code of the country | String |
| HomeInstitutionAddress | Current address | String |
| CurrentLevelOfStudy | ISCED code representing the level of current study | Integer |
| FieldOfStudy | ISCED code representing the field of study | Integer |
| CurrentDegree | Name of the degree the user is attending at home institution | String |
| Degree | ISCED code representing previously achieved level of study | Integer |
| DegreeAwardingInstitution | Name of the degree awarding institution | String |
| GraduationYear | Graduation year | Integer |
| DegreeCountry | ISO 3166-1 alpha-2 code of the country | String |
| LanguageProficiency | Base64 encoded Europass 3.3 compliant declaration of language proficiency | europass3 |
| LanguageCertificates | Base64 encoded list of documents binaries | Document |

Table 2.3. The eID4U project defined additional eIDAS attributes (personal and academic-specific).(Source: [1])

### 2.2.3 Integration of the AP Connector with the Italian eIDAS Node

This section explains how the Italian eIDAS infrastructure was integrated with the AP Connector and the SPID identification system. The IdP Proxy, a specific component of the node's Specific section, was created and built by the FICEP project [61] to enable communication between the Italian eIDAS node and the SPID IdPs. The citizen can choose the SPID IdP to use for authentication by using this component. Additionally, it converts eIDAS communications into SPID messages (and the other way around).

First, we added support for the properties in Table 2.3 by extending the eIDAS code. After that, we altered the IdP Proxy to obtain the extra characteristics from a national AP, combine them with those that the SPID IdP valued, and then transform them into the eIDAS format. Figure 2.2 depicts the components of the Generic and Specific sections of the eIDAS Proxy Service that are engaged in this process. Sections VI-B and VI-C detail the processes that are run in the Generic and Specific parts of the eIDAS node, respectively. We used the (Italian) fiscal number issued by the national Tax Agency with the eIDAS MDS characteristics to identify the natural people throughout the attribute retrieval procedure.The fiscal number is one-of-a-kind, permanent, and its value is constant across time. The AP Connector interface, which we created for the IdP Proxy, is seen in Figure 2.1[1].

The eIDAS node interacts directly over a specialized backed channel with a so-called AP Proxy module to get the properties in the initial AP Connector implementation, dubbed AP-Proxy. The AP returns to the AP Proxy module more properties than required in this implementation. The AP Proxy module filters the additional properties as a result. The AP and the eIDAS node support the OAuth 2.0 protocol in the second AP Connector version, known as AP-OAuth2, for improved authorisation of the data released and management of user permission.

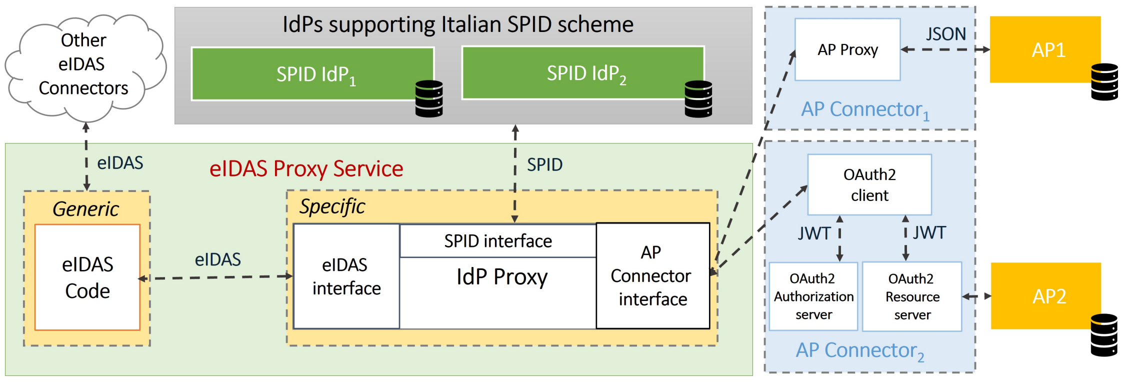


Figure 2.1. AP connections (logical) integration with the Italian eIDAS node . (Source:[1])

## 2.2.4 Italian SPID System

To take use of the SPID system [10], a citizen must register with an authorized SPID IdP, who must verify his identity before issuing an SPID credential. You should be aware that a citizen may register with many SPID IdPs and get multiple SPID credentials with various levels of security. The citizen can use the services offered by the SPs that support SPIDs after receiving an SPID credential. Numerous Italian organizations, including towns, universities, the tax authority, hospitals, or other governmental bodies, now permit citizen identification via the SPID system since it is now required that public services implement SPID [1].

SPID Protocol: Technically speaking, the SPID system is based on the SAML 2.0 standard as well; the operational modes are those offered by SAML v2 for the "Web Browser SSO" profile. When a citizen uses a service that supports SPID, the SP creates a digitally signed SPID authentication request (SPID-Auth-Req) and transmits it to the SPID IdP, where the citizen authenticates using his SPID credential. After a successful authentication, the SPID IdP gives the SP a digitally signed SPID authentication response (SPID-AuthRes) message, and the SP decides whether to grant or deny access to the service depending on the answer.

SPID Metadata: Specific SAML metadata is used to disseminate the certificate(s) that are utilized to validate the signatures of SPID messages. In reality, AgID (Agenzia per l'Italia Digitale) [62] manages an SPID Registry [63] via which the SPID IdPs and SPs share the (SPID) SAML metadata. This organization conducts verifications to prevent anyone other than accredited IdPs and validated SPs from connecting to the SPID system. It also specifies the SPID messages format. SPID Characteristics The IdP sends the so-called SPID properties to the SPs at authentication time. We provide the terms defined for natural people in Table 2.4 [64]. In SPID attribute sets, the attributes are organized. Each SP defines one or more attribute sets in its own SAML metadata, and each set is recognized.

SPID attributes: The IdP sends the so-called SPID characteristics to the SPs at the moment of authentication. We provide the terms defined for natural people in Table 2.4 [64]. In SPID attribute sets, the attributes are organized. Each SP defines one or more attribute sets in its own SAML metadata, and each set is given a unique numerical index. The index number of an SPID attribute set is added to the request when the SP constructs the SPID-Auth-Req. By using the attribute set index present in the SP's SAML metadata, the SPID IdP is able to identify the desired attributes. We have developed a novel concept for the interaction between the IdP Proxy serving as an SPID SP in the communication with the SPID IdPs. We have created a new attribute set and the matching index for the interaction between the IdP Proxy serving as an SPID SP in the communication with the SPID IdPs. This set includes a person's fiscal number together with the bare minimum of reliable information about them (name, surname, and date of birth).[1]

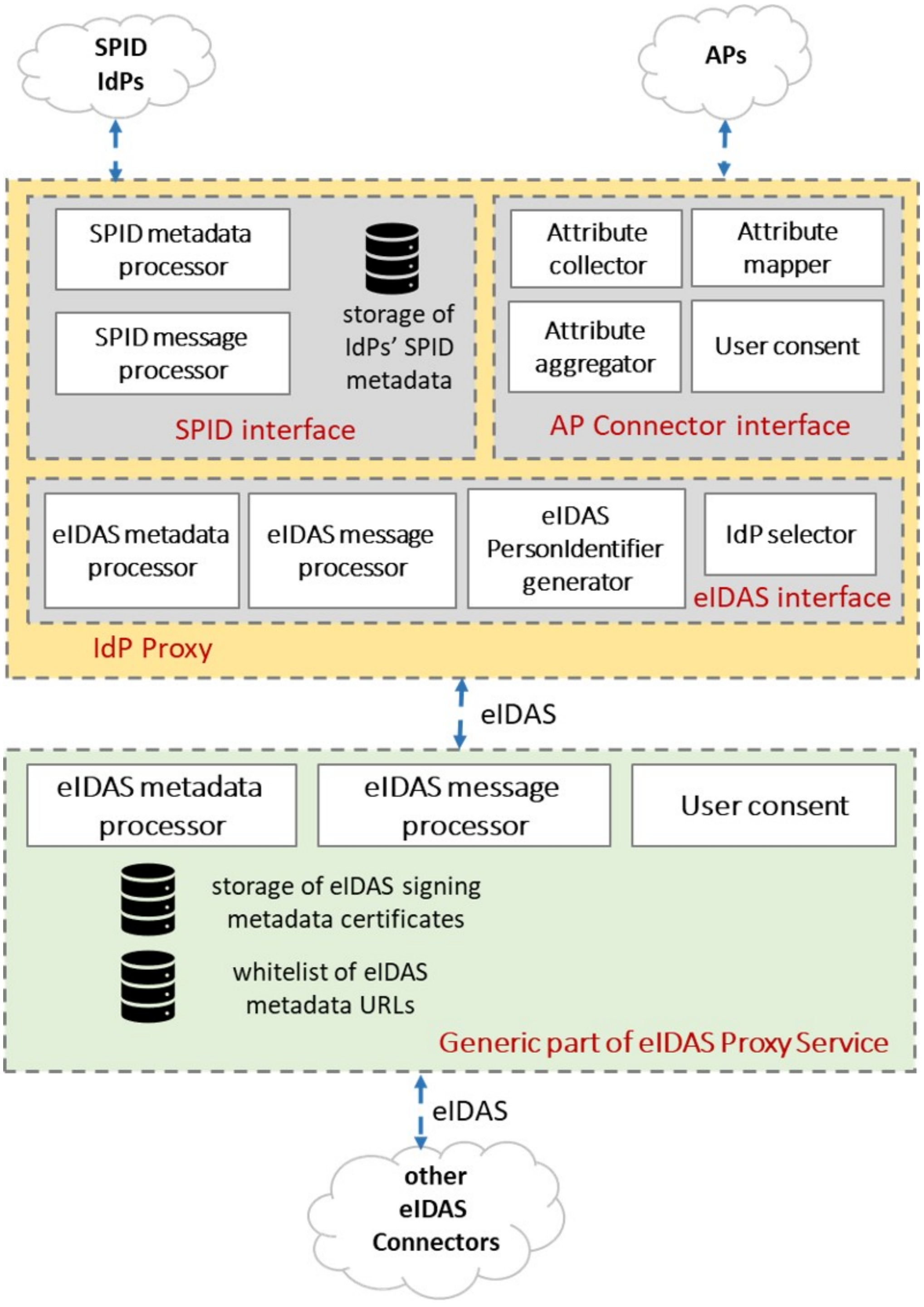


Figure 2.2. Internal view of the Italian eIDAS node showing both the general and particular eIDAS proxy service components. . (Source: [1])

### 2.2.5 The Italian eIDAS Node's Generic Part

Using the node components depicted in Figure 2.2, the Italian eIDAS node processes an eIDAS-Auth-Req obtained from a related eIDAS Connector by carrying out the procedures. The node must first get the CerteIDAS_message_sign certificate of the eIDAS Connector since the request is digitally signed. The Connector's eIDAS information, which includes this certificate, is retrieved and verified using the eIDAS metadata processing module. The eIDAS message processor module then gets ready to prepare the eIDAS-AuthReq if the eIDAS metadata has been properly checked.

| **Name** | **Description** | **Type** |
|---|---|---|
| spidCode | Identification code assigned by the SPID IdP, must be unique in SPID system. | String |
| familyName* | String of one or more non-empty substrings | String |
| name* | String of one or more non-empty substrings | String |
| dateOfBirth* | Date in the format YYYY-MM-DD | Date |
| countryOfBirth* | Two-characters code of the italian province of birth | String |
| placeOfBirth* | Four-characters code of the italian municipality of birth | String |
| address* | String of one or more non-empty substrings | String |
| gender* | Gender F/M | String |
| fiscalNumber* | Unique, single and permanent fiscal identification number | String |
| mobilePhone* | Phone number | String |
| email* | Email address | String |
| idCard* | String of four fields: idType, idNumber, idIssuer, idIssueDate, idExpirationDate | String |
| expirationDate | Date in the format YYYY-MM-DD | Date |
| digitalAddress* | Certified email address | String |

Table 2.4. SPID attributes. (Source: [1])

In actuality, the node generates a fresh eIDAS-Auth-Req that is sent to the IdP Proxy. The procedures are followed by the node to retrieve and validate the eIDAS metadata from the IdP Proxy. The next step is to see if the metadata of the IdP Proxy contains the desired properties. The new eIDAS-Auth-Req is delivered to the IdP Proxy, which processes it in accordance with Section VI-C. It is digitally signed using the private key corresponding to the Generic part's CerteIDAS_message_sign. The node receives and verifies IdP Proxy metadata after receiving an eIDAS-Auth-Res from the IdP Proxy. It then uses the associated encryption certificate to decode the eIDAS attribute values, and for each of the decrypted attributes, it creates a user consent form. It then produces a new eIDAS-Auth-Res containing the attributes encrypted using the CerteIDAS_attributes_encrypt certificate of the Connector after retrieving and validating the Connector's eIDAS information. The digitally signed eIDAS-Auth-Res is returned to the eIDAS Connector in the last step [1].

### 2.2.6 Idp Proxy, a Specific Part of the Italian eIDAS Node

The IdP Proxy interacts with the SPID IdPs and retrieves additional characteristics. The eIDAS-AuthReq obtained from the Generic portion is first verified using the associated eIDAS information, which is downloaded and processed instantly. The IdP Proxy then changes the characteristics' format from eIDAS to SPID. The IdP Proxy adds the SPID fiscalNumber to the list of requested characteristics to identify the citizen in the attribute retrieval phase if the request contains attributes that the SPID IdP is unable to value. The IdP Proxy then downloads the SPID IdP information and verifies that it contains the desired properties. The SPID IdP receives a signed SPIDAuth-Req that was created after choosing the SPID attribute set index.

The received SPID-Auth-Res is handled in the manner. First, the message is verified by utilizing the associated SPID IdP information. The next step is to get any properties that the SPID IdP hasn't valued from the AP using either the AP Proxy (discussed in Section VII-B) or the AP-OAuth2 method (discussed in Section VII-C). The equivalent CerteIDAS_attributes_encrypt of the Generic component is used to transform the returned attributes into the eIDAS format and encrypt them. The IdP Proxy then generates a fresh eIDASAuth-Res that has been digitally signed and transmitted to the Generic section of the eIDAS node.[1]

### 2.2.7 Details of AP Connector Implementation

This section provides information on the proposed AP Connector implementations as well as the changes made to the Generic and Specific sections of the eIDAS code. We utilized the Apache Tomcat 8 and Apache Struts 2 framework-based [65] eIDAS code version 1.4.4 [66], [67]. On the eIDAS node and certain machines at our location, we have installed the implemented AP Connector components. Keep in mind that we have no influence over the SPID IdP or the AP backend (i.e., the university database).

### 2.2.8 New Attributes are Available on the eIDAS Node

The Generic part of the eIDAS node allows support for new attributes of common types. We filled in the empty configuration file named saml-engine-additional-attributes.xml with the details of the attributes defined in [59]-[68] and reported in Table 2.3. For example, we defined the TaxReference attribute for the natural person in the following way [59]:

```
<entry key="14.NameUri">
        http://eidas.europa.eu/attributes/
        naturalperson/TaxReference
</entry>
<entry key="14.FriendlyName">
        TaxReference
</entry>
<entry key="14.PersonType">
        NaturalPerson
</entry>
<entry key="14.Required">false</entry>
<entry key="14.XmlType.NamespaceUri">
        http://eidas.europa.eu/attributes/
        naturalperson
</entry>
<entry key="14.XmlType.LocalPart">
        TaxReferenceType
</entry>
<entry key="14.XmlType.NamespacePrefix">
        eidas-natural
</entry>
<entry key="14.AttributeValueMarshaller">
        eu.eidas.auth.commons.attribute.impl.
        LiteralStringAttributeValueMarshaller
</entry>
```

We changed the IdP Proxy classes to accommodate the AP Proxy approach's attribute extraction from the AP, attribute aggregation, and user consent. Three distinct classes make up the IdP Proxy: In order to communicate with the eIDAS Proxy-Service, the **EIDASController** class implements the eIDAS interface. In order to communicate with SPID IdPs, the **SPIDController** class implements the SPID interface, and the IdpProxyService class converts messages from the eIDAS protocol to the SPID protocol and vice versa. We introduced additional logic to the **IdpProxyService** class to provide user consent on the IdP Proxy [1].

### 2.2.9 AP Proxy

For this strategy, we created the remedy displayed in Figure 2.3 . Below are more explanations of both the implementation specifics and the sequence diagram. Using Python's Flask web application framework [69], we developed the AP Proxy application as a RESTful Web service [70]. We have introduced a new class named APProxyRequestData to the AP Connector interface of the IdP Proxy on the eIDAS node.The SPIDController class calls the aforementioned class by sending it the person's fiscal number and the list of desired properties. The APProxyRequestData then creates a mutually authorized TLS connection with the AP Proxy application. The program makes an HTTP GET request with the fiscal number and the extra desired information across the TLS secured channel. The fiscal number is read by the AP Proxy application and sent through an HTTP GET request to a specific web service operating at the Student Service Office backend. The JSON response's attributes are returned using the same technique [69]. AP Proxy application characteristics, for instance, are displayed below in JSON format:

```
{
        "Citizenship": "IT",
```

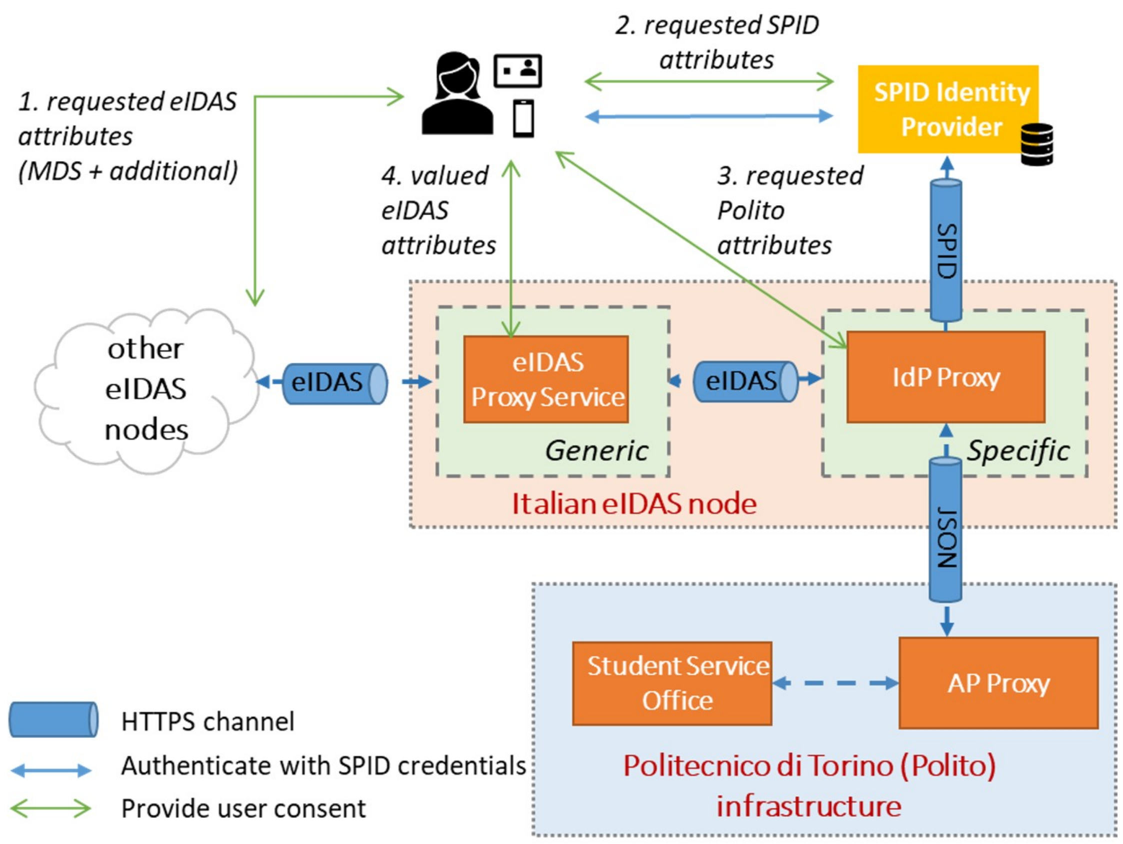

1. requested eIDAS attributes (MDS + additional)
2. requested SPID attributes
SPID Identity Provider
3. requested Polito attributes
4. valued eIDAS attributes
other eIDAS nodes
eIDAS
eIDAS Proxy Service
Generic
eIDAS
IdP Proxy
Specific
SPID
Italian eIDAS node
JSON
Student Service Office
AP Proxy
Politecnico di Torino (Polito) infrastructure
HTTPS channel
Authenticate with SPID credentials
Provide user consent

Figure 2.3. Internal view of the Italian eIDAS node showing both the general and particular eIDAS proxy service components. . (Source: [1])

```
        "CountryOfBirth": "IT",
        "CurrentAddress": "QklUVEk=",
        "CurrentDegree": "",
        "CurrentFamilyName": "ROSSI",
        "CurrentGivenName": "MARCO",
        "DateOfBirth": "1994-03-29",
        "Email": "s111122@studenti.polito.it",
        "Gender": "Male",
        "HomeInstitutionAddress": "Q2..JUxZ\f\n",
        "HomeInstitutionCountry": "IT",
        "HomeInstitutionIdentifier": "POLITO",
        "HomeInstitutionName":
        "Politecnico di Torino",
        "IdIssuer": "MCTC-NU",
        "IdNumber": "NU53415682",
        "MaritalState": "N/A",
        "Nationality": "IT",
        "PersonIdentifier": "176311",
        "Phone": "3465678312345",
        "PlaceOfBirth": "Milano",
        "TemporaryAddress": "QklUVEk="
}
```

The AP Proxy program discards any characteristics that were not requested, while converting any that were to a format that is compatible with eIDAS. As an illustration, the characteristics with addresses (like CurrentAddress) are added to tags. The characteristics are transmitted back to the IdP Proxy in JSON format. The SPIDController class receives the valued properties once the APProxyRequestData class has parsed the JSON response. We have gotten TLS certificates from Let's Encrypt Certification Authority (CA) and configured them into the deployed components in order to establish mutually authenticated TLS connections between IdP Proxy and AP Proxy. A TLS termination proxy is provided by the NGINX reverse proxy [71] that is put in front of the AP proxy [?]. We also inserted the TLS certificates from the IdP Proxy and the reverse proxy, respectively, into the dedicated certificate keystore of the IdP Proxy. According to Figure 2.3, the user's consent is requested in four separate phases. For the necessary eIDAS MDS characteristics, as well as the optional eIDAS MDS attributes and the extra ones, the eIDAS Proxy Service creates user consent pages. For the requested SPID characteristics, the SPID IdP provides a user consent page, whereas the IdP Proxy creates user consent pages for the requested AP attributes. Finally, before transferring the valued attributes to the complementary eIDAS node, the eIDAS Proxy Service requests the user's permission).

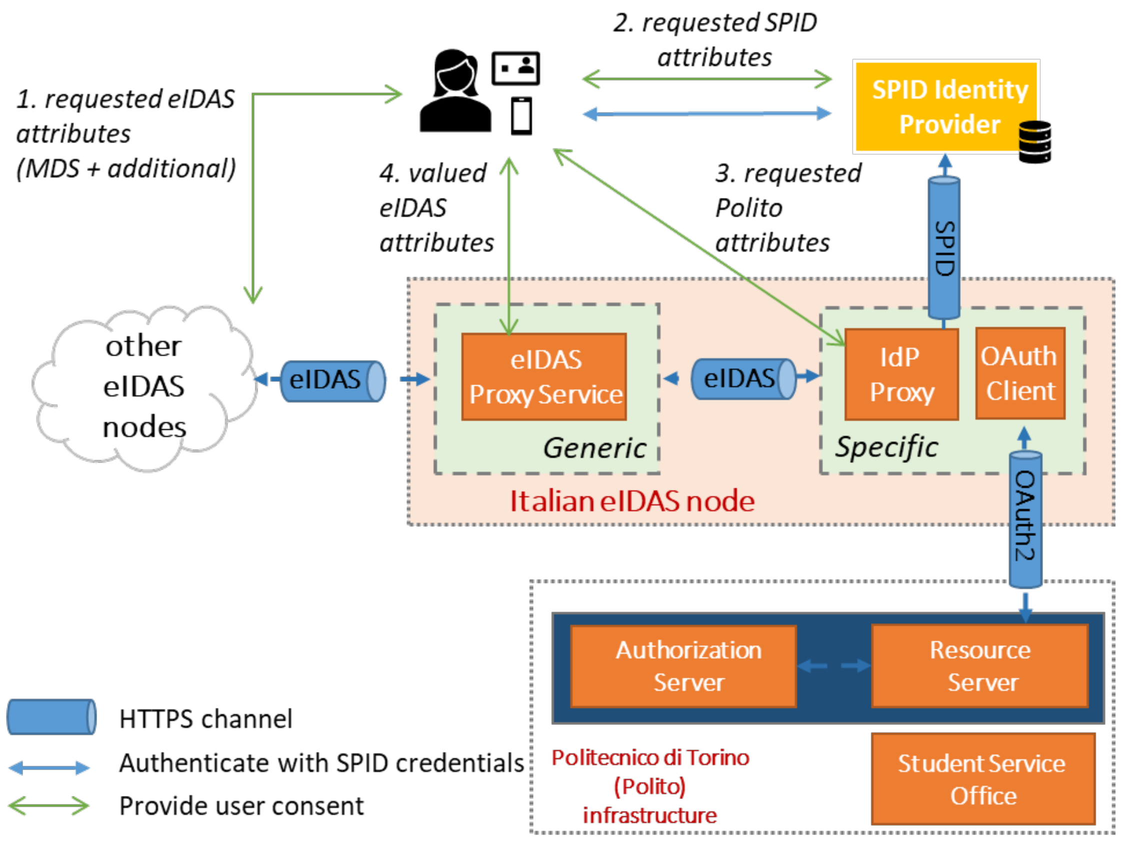


Figure 2.4. Components involved in the AP connector implementation exploiting OAuth 2.0 protocol . (Source: [1])

### 2.2.10 AP OAuth2

To assist with this strategy, we created and put into action the architecture depicted in Figure 2.4, which takes use of the OAuth 2.0 protocol's Authorization (AuthZ) Code Grant procedure. We will explain in more detail in Chapter 3.

## 2.3 OAuth 2.0

Open Authorization 2.0 (OAuth 2.0)[72] is a standard that enables a website or application to access resources hosted by other web apps on behalf of a user. In 2012, it took the place of

OAuth 1.0 and is currently the de facto industry standard for online authorization. Without ever disclosing the user's credentials, OAuth 2.0 offers agreed-upon access and limits the activities the client app can do on the user's behalf on resources.

### 2.3.1 Principles of OAuth 2.0

OAuth 2.0 is a protocol for authorization, NOT for authentication. As a result, its main purpose is to enable access to a range of resources, such as external APIs or user data.

OAuth 2.0 makes use of Access Tokens. The authorization to access resources on behalf of the end user is represented by an **access token**, which is a piece of data. A specific format for Access Tokens is not specified by OAuth 2.0. JSON Web Token (JWT) format, however, is frequently used in specific situations. As a result, data can be included in the token itself by token issuers. Access Tokens could have an expiration date as well for security reasons.

### 2.3.2 OAuth 2.0 Roles

The OAuth2.0 authorization framework's main definition includes the concept of roles. The following list identifies the fundamental parts of an OAuth 2.0 system:

- **Resource Owner:** A user or system with ownership of and access control over protected resources.
- **Client:** The system that needs access to the protected resources is referred to as the client. The Client must possess the proper Access Token in order to access resources.
- **Authorization Server:** Requests for Access Tokens are received from the Client by this server, which then provides them upon successful resource owner authorization and authentication. The Authorization endpoint, which manages the user's interactive authentication and consent, and the Token endpoint, which is involved in a machine-to-machine connection, are the two endpoints that the Authorization server exposes.
- **Resource Server:** A server that handles access requests from the client and safeguards the user's resources. The appropriate resources are then returned to the Client after accepting and validating an Access Token from them.

### 2.3.3 OAuth 2.0 Scopes

The concept of scopes is crucial to OAuth 2.0. They are used to precisely define the grounds for granting access to resources. The Resource Server determines what resources are relevant to acceptable scope values.

### 2.3.4 OAuth 2.0 Access Tokens and Authorization Code

After the Resource Owner has given permission for access, the OAuth 2.0 Authorization server might not immediately return an Access Token. An Access Token is then exchanged for an Authorization Code in place of this, which would provide greater security. Along with the Access Token, the Authorization server may also issue a Refresh Token. Refresh Tokens, in contrast to Access Tokens, typically have long expiration dates and can be traded for new Access Tokens when the latter expires. Due to these characteristics, clients must securely store Refresh Tokens.

### 2.3.5 OAuth 2.0: How Does It Operate?

At the most fundamental level, before OAuth 2.0 can be utilized, the Client must obtain its own credentials, a *client id* and *client secret* from the Authorization Server in order to identify and authenticate itself when making a request for an Access Token.

Access requests are made via OAuth 2.0 by the Client, such as a desktop application, smart TV app, website, mobile app, etc. The overall flow of the token request, exchange, and response is as follows:

1. The client submits an authorization request to the authorization server, identifying themselves by providing their client id and secret. They also include the scopes and an endpoint URI (*redirect URI*) to which the access token or authorization code should be sent.
2. The Authorization server validates the Client's identity and confirms that the requested scopes are legal.
3. The owner of the resource communicates with the authorization server to provide access.
4. Depending on the grant type, as described in the following section, the authorization server redirects back to the client with either an authorization code or an access token. Additionally, a Refresh Token might be given back.
5. The Client asks the Resource server for access to the resource using the Access Token.

### 2.3.6 Grant Types in OAuth 2.0

**Grants** are the series of actions a Client must take in order to obtain resource access authorization in OAuth 2.0. The authorization framework offers a number of grant types to handle various scenarios:

- **Authorization Code grant:** The Client exchanges the single-use **Authorization Code** received from the Authorization server for an Access Token. For conventional web programs, where the exchange may safely take place on the server side, this is the ideal choice. Mobile/native apps as well as Single Page Apps (SPA) may employ the Authorization Code flow. However, because the client secret cannot be safely kept here, just the client id can be used for authentication throughout the exchange. The *Authorization Code* with *PKCE* grant, as below, is a preferable substitute.
- **Implicit Grant:** A more straightforward flow where the Client is given the Access Token directly. The authorization server may provide the Access Token in response to a form post or as a parameter in the callback URI in the implicit flow. Because of a possible token leak, the first option has been deprecated.
- **Authorization Code Grant with Proof Key for Code Exchange (PKCE):** This authorization flow is comparable to the Authorization Code grant, but it includes extra steps to increase security for native and mobile apps as well as SPAs.
- **Resource Owner Credentials Grant Type:** The credentials of the resource owner must first be obtained by the Client and sent to the Authorization server in order to receive this permission. As a result, it is only available to those who can be entirely trusted. It has the benefit of not requiring a redirect to the Authorization server, making it useful in use cases where a redirect is infeasible.
- **Client Credentials Grant Type:** Used for non-interactive applications, such as automated workflows and microservices. Using its client id and secret, the application in this instance is authenticated per se.
- **Device Authorization Flow:** A grant that makes it possible for apps to be used on input-limited devices, including smart TVs.
- **Refresh Token Grant:** The flow that entails trading a Refresh Token for a new Access Token.

## 2.4 JWT

JSON Web Token (JWT) [2] is an open standard (RFC 7519) [73] that specifies a condensed and independent method for sending information securely between parties as a JSON object. Due to its digital signature, this information can be checked and trusted. A secret can be used to sign JWTs using the HMAC algorithm, or a public/private key pair can be used to sign JWTs using RSA or ECDSA.

Despite the fact that JWTs can be encrypted to additionally offer confidentiality between parties, we will concentrate on signed tokens. While encrypted tokens conceal those claims from outside parties, signed tokens allow the integrity of the claims they contain to be verified. When signing tokens with public/private key pairs, the signature also attests that the token was signed by only the party that possesses the private key.

Here are a few situations in which JSON Web Tokens are helpful:

- **Authorization:** The most typical use case for JWT is this one. Once logged in, the user can access the routes, services, and resources that are made available with that token by including the JWT in each request going forward. Due to JWT's low overhead and simplicity of use across several domains, single sign-on is a feature that is frequently utilized nowadays.
- **Information Exchange:** Information can be securely transmitted between parties using JSON Web Tokens. You can be certain that the senders are who they claim to be because JWTs can be signed (for instance, using public/private key pairs). You may also confirm that the content hasn't been altered because the signature is created using the header and the payload.

### 2.4.1 JSON Web Token Structure

JSON Web Tokens are made up of three sections that are separated by dots (.). These three components are as follows:

- Header
- Payload
- Signature

As a result, this is how a JWT usually appears.

```
xxxxx.yyyyy.zzzzz
```

Let's examine each component separately.

- **Header:** The type of the token, which is often a JWT, and the signature algorithm being used, like HMAC SHA256 or RSA, are both typically included in the header.

  ```
  {
          "alg": "HS256",
          "typ": "JWT"
  }
  ```

  The first component of the JWT is created by **Base64Url** encoding this JSON.

- **Payload:** The payload, which includes the claims, makes up the token's second component. Claims are assertions about a subject (usually the user) and supplementary information. *Registered*, *public*, and *private* claims are the three different categories of claims.

- **Registered Claims:** In order to provide a set of useful, interoperable claims, this set of preconfigured claims is provided. They are not required, but they are suggested. To name a few of them: **iss** (issuer), **exp** (expiration time), **sub** (subject), **aud** (audience), and others.

  JWT is designed to be small, therefore you'll see that the claim names are only three characters long.
- **Public Claims:** Those that use JWTs have complete freedom to specify these. The IANA JSON Web Token Registry or a URI that contains a collision resistant namespace should be used to define them in order to prevent collisions.
- **Private claims:** These are not registered or public claims; rather, they are private claims made specifically to transmit information between persons who consent to utilizing them.

```
{
        "sub": "1234567890",
        "name": "John Doe",
        "admin": true
}
```

The second component of the JSON Web Token is the payload, which is then **Base64Url** encoded.

Please be aware that with signed tokens, this information is accessible to anybody even though it is protected from tampering. Unless it is encrypted, private information should not be included in the payload or header sections of a JWT.

- **Signature:** You must sign the encoded header, encoded payload, secret, and algorithm indicated in the header to construct the signature portion.

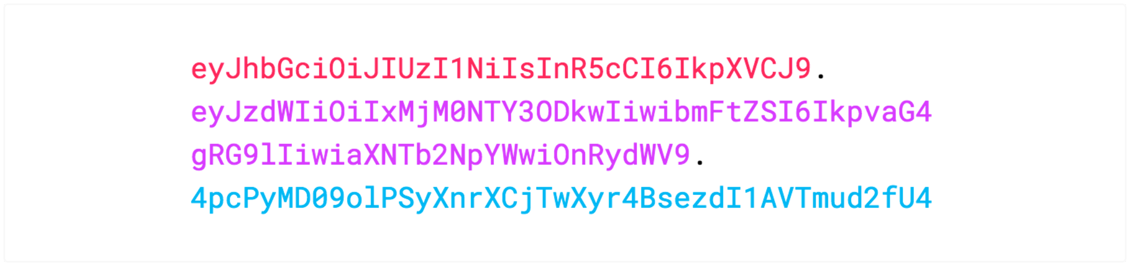
  For instance, if you choose to employ the HMAC SHA256 method, the signature will be generated as follows:

```
HMACSHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),
secret)
```

The signature is used to ensure that the message was not altered along the route, and in the case of tokens signed with a private key, it can also ensure that the JWT was sent by the specified party.

When compared to XML-based standards like SAML, the result consists of three Base64-URL strings separated by dots that may be given with ease in HTML and HTTP settings.

The JWT that is displayed in the example below is signed using a secret and has the header and payload from the preceding JWT encoded.



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Figure 2.5. Encoded JWT example. (Source: [2] )

### 2.4.2 JSON Web Tokens: How Do They Operate?

A JSON Web Token will be returned in the authentication process after the user successfully logs in with their credentials. Tokens are credentials, thus extreme caution must be exercised to avoid security problems. Tokens shouldn't generally be kept any longer than necessary.

Due to a lack of security, you should also avoid storing important session data in the browser's storage.

The user agent should provide the JWT whenever the user tries to access a protected route or resource, often in the **Authorization** header using the **Bearer** schema. The header's text should contain the following information:

```
Authorization: Bearer <token>
```

In some circumstances, this may serve as a stateless permission mechanism. The user will be permitted to access protected resources if a valid JWT is found in the **Authorization** header, which is checked by the server's protected routes. The requirement to query the database for some activities may be diminished if the JWT contains the required data, albeit this may not always be the case.

Keep in mind that you should try to keep JWT tokens from growing too large if you send them using HTTP headers. Some servers will only accept headers up to 8 KB in size. A different approach, such as Auth0 Fine-Grained Authorization, may be required if you are trying to embed excessive amounts of data in a JWT token, such as by incorporating all of the user's rights.

Cross-Origin Resource Sharing (CORS), which doesn't employ cookies, won't be a problem if the token is sent in the Authorization header.

A JWT can be acquired and used to access APIs or services according to the diagram below:

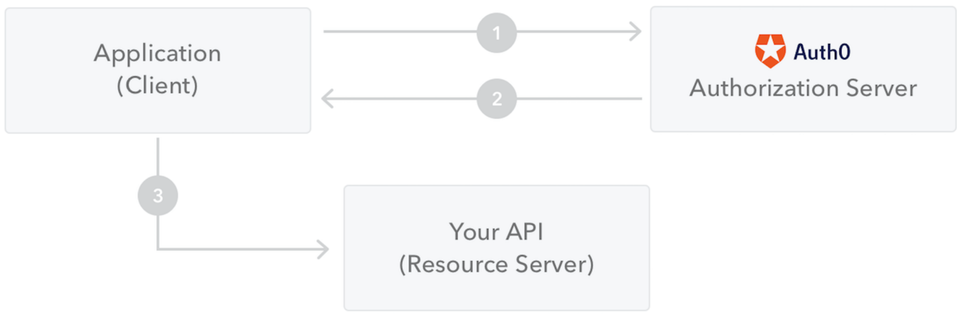


Figure 2.6. Client Credentials Grant. (Source: [2])

1. The application or client asks the authorization server for permission. One of the several authorization flows is used for this. For instance, a typical OpenID Connect compliant web site will use the authorization code flow to pass through the /authorize endpoint.
2. The authorization server provides the application with an access token after approving the request.
3. A protected resource is accessed by the application using the access token (like an API).

Be aware that with signed tokens, even while users or other parties cannot change the information, it is still completely visible to them. This indicates that you shouldn't include sensitive information in the token.

### 2.4.3 Why Should We Use JSON Web Tokens?

Comparing **JSON Web Tokens (JWT)** to **Simple Web Tokens (SWT)** and **Security Assertion Markup Language Tokens (SAML)**, let's discuss the advantages of each.

JWT is more compact than SAML because JSON is less verbose than XML and has a reduced overall file size when encoded. JWT is an excellent option for passing in HTML and HTTP environments because of this.

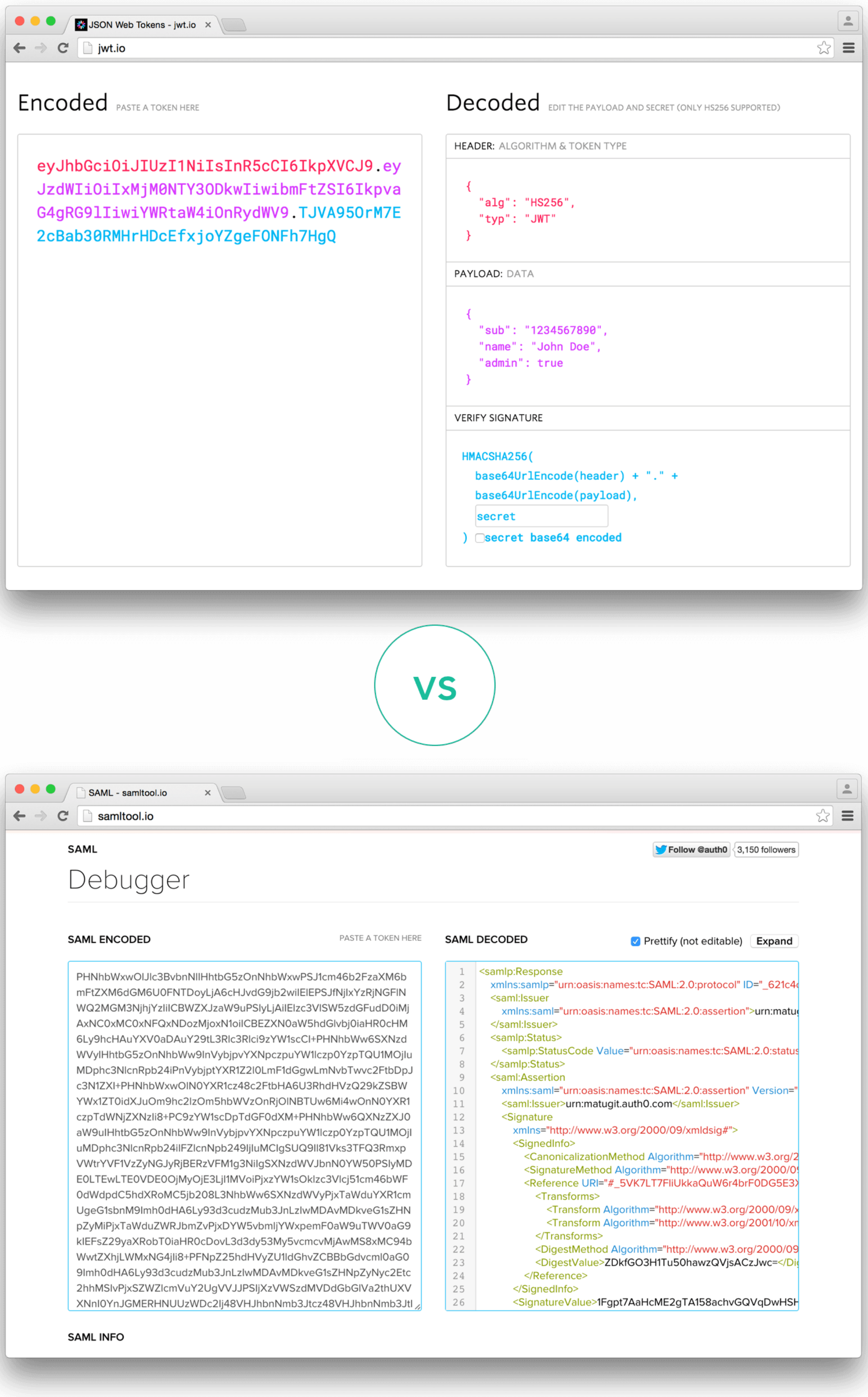
Security-wise, SWT can only be symmetrically signed using the HMAC algorithm by a shared secret. However, a public/private key pair in the form of an X.509 certificate can be used to sign JWT and SAML tokens. In contrast to the ease of signing JSON, signing XML with an XML Digital Signature is quite difficult to do without creating hidden security flaws.

Most computer languages employ JSON parsers because of how well it maps to objects. On the other hand, there is no inherent document-to-object mapping in XML. As a result, working with JWT is simpler than SAML claims.

JWT is utilized on the Internet at a large extent. This demonstrates how the JSON Web token can be processed easily on the client-side across many platforms, particularly mobile.

## 2.5 Redis

Redis [74] is a distributed, in-memory key/value database, cache, and message broker with optional durability that is used as an in-memory data structure store. Various abstract data structures, including strings, lists, maps, sets, sorted sets, HyperLogLogs, bitmaps, streams, and spatial indices, are supported by Redis.



eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) ☐secret base64 encoded
```

PHNhbWxwOlJlc3BvbnNlIHhtbG5zOnNhbWxwPSJ1cm46b2FzaXM6b
mFtZXM6dGM6U0FNTDoyLjA6cHJvdG9jb2wiIElEPSJfNjIxYzRjNGFlN
WQ2MGM3NjhjYzIiICBWZXJzaW9uPSIyLjAiIElzc3VlSW5zdGFudD0iMj
AxNC0xMC0xNFQxNDozMjoxN1oiICBEZXN0aW5hdGlvbj0iaHR0cHM
6Ly9hcHAuYXV0aDAuY29tL3Rlc3Rlci9zYW1scCI+PHNhbWw6SXNzd
WVyIHhtbG5zOnNhbWw9InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIu
MDphc3NlcnRpb24iPnVybjptYXR1Z2l0LmF1dGgwLmNvbTwvc2FtbDpJ
c3N1ZXI+PHNhbWxwOlN0YXR1cz48c2FtbHA6U3RhdHVzQ29kZSBW
YWx1ZT0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOnN0YXR1
czpTdWNjZXNzIi8+PC9zYW1scDpTdGF0dXM+PHNhbWw6QXNzZXJ0
aW9uIHhtbG5zOnNhbWw9InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjI
uMDphc3NlcnRpb24iIFZlcnNpb249IjIuMCIgSUQ9Il81Vks3TFQ3Rmxp
VWtrYVF1VzZyNGJyRjBERzVFM1g3NiIgSXNzdWVJbnN0YW50PSIyMD
E0LTEwLTE0VDE0OjMyOjE3LjI1MVoiPjxzYW1sOklzc3Vlcj51cm46bWF
0dWdpdC5hdXRoMC5jb208L3NhbWw6SXNzdWVyPjxTaWduYXR1cm
UgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvMDkveG1sZHN
pZyMiPjxTaWduZWRJbmZvPjxDYW5vbmljYWxpemF0aW9uTWV0aG9
kIEFsZ29yaXRobT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS8xMC94b
WwtZXhjLWMxNG4jIi8+PFNpZ25hdHVyZU1ldGhvZCBBbGdvcml0aG0
9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvMDkveG1sZHNpZyNyc2Etc
2hhMSIvPjxSZWZlcmVuY2UgVVJJPSIjXzVWSzdMVDdGbGlVa2thUXV
XNnI0YnJGMERHNUUzWDc2Ij48VHJhbnNmb3Jtcz48VHJhbnNmb3JtI

```
<samlp:Response
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" ID="_621c4
  <saml:Issuer
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">urn:matu
  </saml:Issuer>
  <samlp:Status>
    <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status
  </samlp:Status>
  <saml:Assertion
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" Version="
    <saml:Issuer>urn:matugit.auth0.com</saml:Issuer>
    <Signature
      xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/2
        <SignatureMethod Algorithm="http://www.w3.org/2000/0
        <Reference URI="#_5VK7LT7FliUkkaQuW6r4brF0DG5E3
          <Transforms>
            <Transform Algorithm="http://www.w3.org/2000/09/
            <Transform Algorithm="http://www.w3.org/2001/10/x
          </Transforms>
          <DigestMethod Algorithm="http://www.w3.org/2000/09
          <DigestValue>ZDkfGO3H1Tu50hawzQVjsACzJwc=</Di
        </Reference>
      </SignedInfo>
      <SignatureValue>1Fgpt7AaHcME2gTA158achvGQVqDwHSH
```

Figure 2.7. Comparison of the length of an encoded JWT and an encoded SAML.(Source: [2])

# Chapter 3

# Design and Implementation

In order to facilitate the attribute retrieval and authorization process within our system, we made several modifications to the existing components. The primary focus was on enhancing the functionality of the IdP Proxy, specifically by updating the processResponse method in the SPIDController class.

The processResponse method plays a pivotal role in handling the SPID-Auth-Res received from the SPID IdP. It extracts the citizen's identification data, such as the SPID fiscal number, and initiates the attribute retrieval process using the OAuth2 protocol. The OAuth client is invoked to handle this attribute retrieval, and upon receiving the attribute response, it sends the data back to the SPIDController class.

Within the SPIDController class, further processing takes place. The retrieved attributes are converted into the eIDAS format, and they are combined with the previously obtained SPID attributes. This step involves attribute aggregation and formatting to prepare the data for the next stage. Finally, the IdP Proxy prepares an eIDAS-Auth-Res message, which is then transmitted to the Generic part of the eIDAS Proxy Service.

In addition to the modifications made to the IdP Proxy, we have implemented several other components within our system. These include the OAuth client, the Authorization Server (AuthZ server), and the Resource Server. Each component plays a specific role in the overall attribute retrieval and authorization process.

For project management and handling dependencies, we utilized Maven [75]. Our implementation is based on Jakarta EE [75] - [76] with MicroProfile, and Open Liberty serves as the server runtime environment. To manage JWT (JSON Web Token) tokens, we employed the Nimbus JOSE (Javascript Object Signing and Encryption) and JWT library [77].

By adopting this implementation approach, we have created a robust system that efficiently handles attribute retrieval, authorization, and token management using well-established technologies and frameworks.

## 3.1 Implementation Diagram

The provided steps outline a series of actions related to attribute request, authorization, and data exchange within a system. These steps play a crucial role in securely retrieving and authorizing citizen identification data. Here is a description of each step:

1. **Attributes Request:** Upon a successful SPID login, the IdP Proxy retrieves the academic requested attributes along with the MDS (name, familyName, dateOfBirth, identifier, and fiscalNumber) from the eIDAS response. It then prepares a signed JWT using a private key and sends it as an id token to the Client.

2. **Authorization Code Request:** The Client receives the id token and decodes it using the public key. The Client stores the MDS (name, familyName, dateOfBirth, and fiscalNumber) in Redis. It then generates an authorization code for the Polito AP client to create an access token later in step 4. This step involves starting the OAuth 2.0 flow for the eidas client by calling the HTTP GET /authorize endpoint of the Authorization Server. The required parameters and academic requested attributes are passed in a scope.

3. **Citizen Identification Data Request:** The logged-in user's information is kept in Redis on the client side. The Authorization Server needs to obtain this information, so it uses an access token for data exchange. The Authorization Server calls the HTTP GET /token endpoint, providing the authorization code (openidconnect code) generated in step 2.

4. **Citizen Identification Data Response:** The Client generates an access token and includes an id token as part of the response in JWT format.

5. **Validate Citizen Information:** The Authorization Server receives the id token and stores it in Redis. It then validates the user's identity based on the received token.

6. **Show Authorization Form:** After successful validation, the user is redirected to an authorization form.

7. **Authorize Attributes:** In order to receive a response from the Resource Server in the subsequent steps, the user checks the requested attributes and grants permission for their access.

8. **Send User Choices:** The Authorization Server receives the user's chosen attributes along with the approval status.

9. **Authorization Code Response:** To prevent CSRF (Cross-Site Request Forgery), the Authorization Server sends the user to the redirect URI defined in step 2, including the code and state parameter.

10. **Access Token Request:** The client obtains the authorization code, verifies that the state parameter matches the one sent earlier, and prepares an HTTP POST to the authorization token endpoint. The client encodes the secret in Base64url format and includes the authorization code, redirection URI, and grant type as form items.

11. **Access Token Response:** The Authorization Server decodes the authorization header, verifies the credentials, extracts the code, and generates an access token. It loads the Redis datastore, retrieves the fiscal number, and includes it as a claim in the access token. The response also includes the signed and serialized JWT for access, the expiration time (30 minutes), the token type, and the authorized scopes in the JSON response.

12. **Resources Access Request:** The Client prepares an HTTP GET request to the /api/resource/read endpoint, including the access token in the authorization header. The Client extracts the signed serialized JWT from the access token variable in the JSON response and verifies its signature using the public key. The Client then stores the access token to request resources from the Resource Server.

13. **Attributes Request:** The Resource Server, with the public key available locally, verifies the JWT provided by the Client. It uses the Private Claim of the group to map the Jakarta RR Roles on the requested resources. In this case, since resource.read is the only relevant group, access is granted for the resource read request. The Resource Server prepares the response by including a welcome message for the user in HTML format.

14. **Attributes Response:** The Resource Server calls an endpoint (which can be a JSON file) to retrieve the information related to the authorized fiscal code based on the requested scopes. It reads the subject from the access token, decodes it with the public key, and retrieves the fiscal number.

15. **Resources Access Response:** The Resource Server responds to the Client with the requested attributes in a POST form.

16. **Attributes Response:** The Client receives the response and provides the IdP Proxy with the requested information from step 1.

These implementation steps provide a comprehensive overview of the attribute retrieval, authorization, and data exchange process within the system.

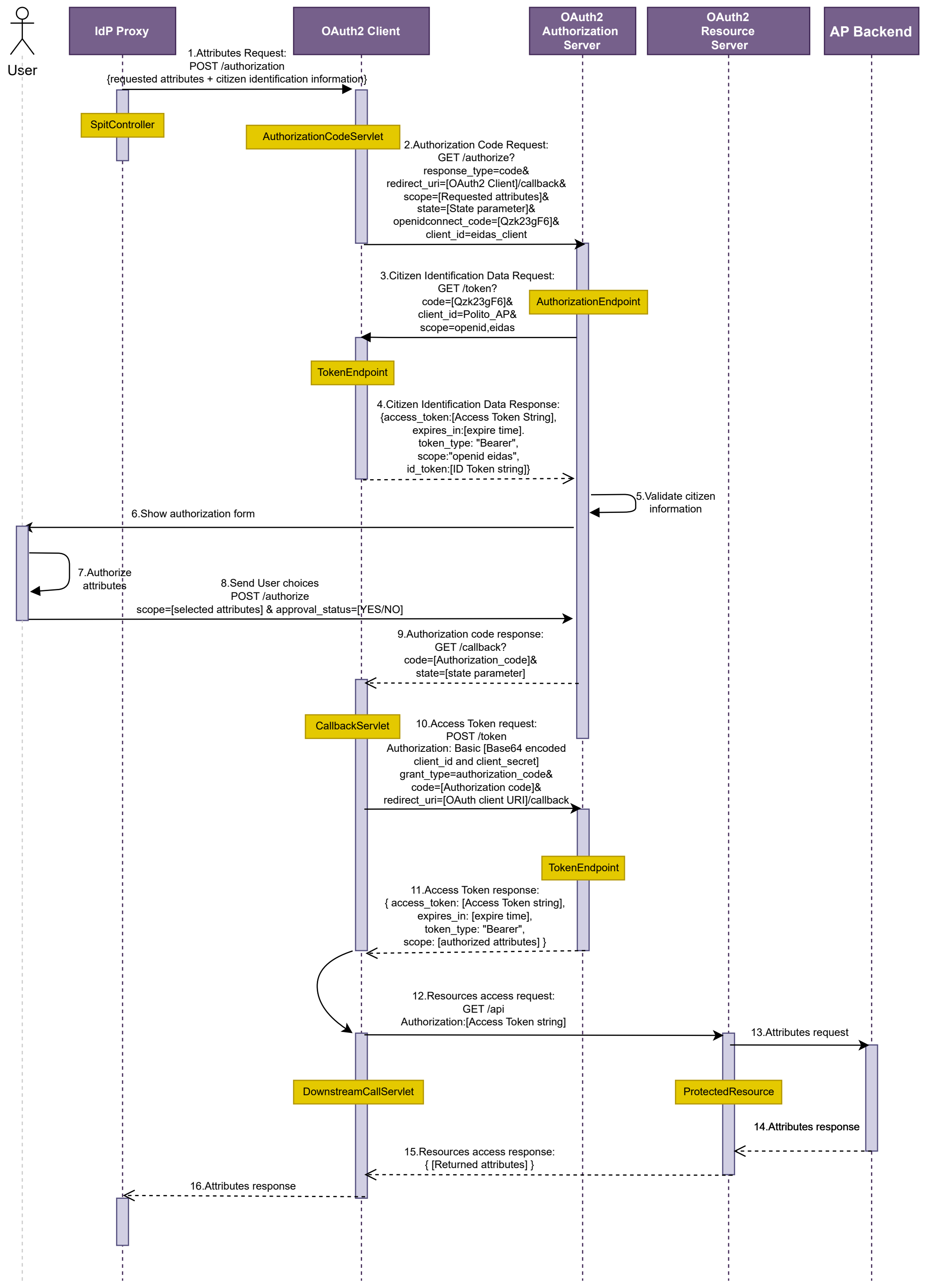

User
IdP Proxy
OAuth2 Client
OAuth2 Authorization Server
OAuth2 Resource Server
AP Backend
1.Attributes Request:
POST /authorization
{requested attributes + citizen identification information}
SpitController
AuthorizationCodeServlet
2.Authorization Code Request:
GET /authorize?
response_type=code&
redirect_uri=[OAuth2 Client]/callback&
scope=[Requested attributes]&
state=[State parameter]&
openidconnect_code=[Qzk23gF6]&
client_id=eidas_client
3.Citizen Identification Data Request:
GET /token?
code=[Qzk23gF6]&
client_id=Polito_AP&
scope=openid,eidas
AuthorizationEndpoint
TokenEndpoint
4.Citizen Identification Data Response:
{access_token:[Access Token String],
expires_in:[expire time].
token_type: "Bearer",
scope:"openid eidas",
id_token:[ID Token string]}
5.Validate citizen information
6.Show authorization form
7.Authorize attributes
8.Send User choices
POST /authorize
scope=[selected attributes] & approval_status=[YES/NO]
9.Authorization code response:
GET /callback?
code=[Authorization_code]&
state=[state parameter]
CallbackServlet
10.Access Token request:
POST /token
Authorization: Basic [Base64 encoded client_id and client_secret]
grant_type=authorization_code&
code=[Authorization code]&
redirect_uri=[OAuth client URI]/callback
TokenEndpoint
11.Access Token response:
{ access_token: [Access Token string],
expires_in: [expire time],
token_type: "Bearer",
scope: [authorized attributes] }
12.Resources access request:
GET /api
Authorization:[Access Token string]
13.Attributes request
DownstreamCallServlet
ProtectedResource
14.Attributes response
15.Resources access response:
{ [Returned attributes] }
16.Attributes response

Figure 3.1. Flow diagram.

## 3.2 Idpproxy

The request is actually handled by the idpproxy, which finishes the necessary translations, creates an SPID Authentication Request, and then transmits it to an Idp that is a component of the SPID domain. It then receives the response with the requested attributes. consists of a gateway that communicates with SPID Idp and eIDAS Node as national and EU SP, respectively.

### 3.2.1 Spit

It is an SPID Proxy Service Provider that can communicate with SPID Idps to collect attributes and authenticate users. This suggests that Spit will transmit requests from an eIDAS Node proxy service, handled by other sub_modules, to an SPID Node, enabling the Spit interface to communicate with an SPID Idp for user authentication.

It is technically a java servlet [78], a Java program that runs on a web server or application server that supports Java. It is utilized to manage requests received from web servers, process those requests, generate responses, and then transmit those responses back to the web servers.

After SPID authentication, we receive the requested attributes in *SpitController.java*. Since the requested attributes do not contain a fiscal number, we then verify the missing attributes and put the fiscal number there.

Take note that we create a default fiscal number in the case that the attribute provider does not provide the fiscal number. This is only for testing the flow; in a production environment, it must be removed [1].

```
idpproxy/test/src/idpproxy-eid4u-taxreference/src/main/java/it/
    telecomitalia/ficep/idpproxy/spit/controller/SpitController.java



@RequestMapping(value = PROCESS_RESPONSE, method = RequestMethod.POST)
public ModelAndView
    processResponse(@RequestParam(Constants.SPID_SAMLRESPONSE_PARAM_NAME)
    String samlSpidREsponse,
@RequestParam(Constants.SPID_RELAYSTATE_PARAM_NAME) String relayState,
    ModelMap model,
HttpServletRequest request) {
/..

List<String> first_name = dt.getAttributeMapWithValues().get("name");
List<String> family_name = dt.getAttributeMapWithValues().get("familyName");
List<String> date_of_birth =
    dt.getAttributeMapWithValues().get("dateOfBirth");
List<String> fiscalNumber =
    dt.getAttributeMapWithValues().get("fiscalNumber");

String academicRequestedAttributes = missingAttributes.toString();
academicRequestedAttributes = academicRequestedAttributes.replace("[","");
academicRequestedAttributes = academicRequestedAttributes.replace("]","");

try {
       String jwtIdpproxy =
           JWT.createJWTPV(first_name.get(0),family_name.get(0),date_of_birth.get(0),
       fiscalNumber.get(0),academicRequestedAttributes);
       model.put("id_token", jwtIdpproxy);
}
catch (Exception ex){
        System.out.println(ex.getMessage());
        request.setAttribute("error", ex.getMessage());
}


view = new ModelAndView("oauth2",model);
}
```

It's time to send these pieces of information to the *oauth2-client*. There is a Java library called **jsonwebtoken**[79] that can be used to create and sign JWTs, as we discussed in the background. However, there are some dependencies that need to be installed, they need to be added to the *pom.xml* file.

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.11.2</version>
</dependency>

<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-impl</artifactId>
  <version>0.11.2</version>
  <scope>runtime</scope>
</dependency>

<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-jackson</artifactId> <!-- or jjwt-gson if Gson is
      preferred -->
  <version>0.11.2</version>
  <scope>runtime</scope>
</dependency>
```

Setting up the claims and other information needed to construct a JWT is the initial step, followed by signing it with an RSA private key.

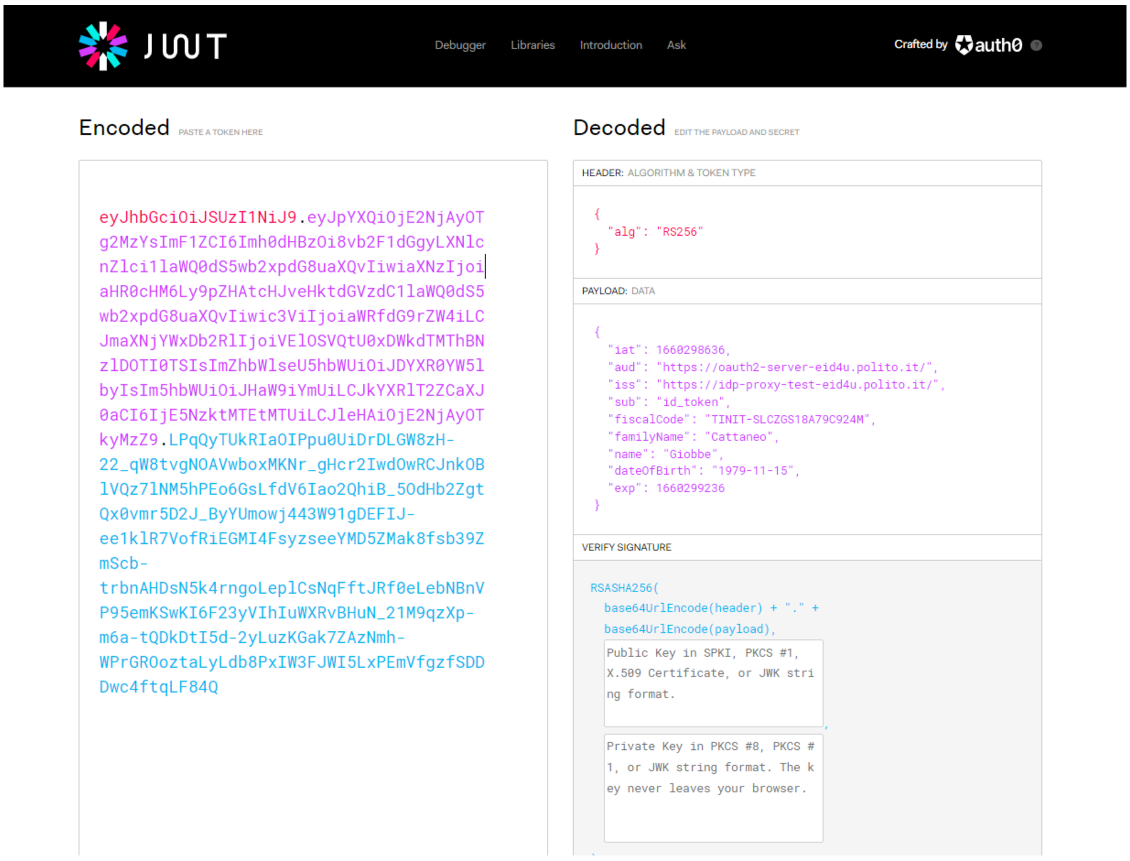
The *JWTDemo.java* code contains a function named *createJWTPV()* that we use to accomplish it.

```
idpproxy/test/src/idpproxy-eid4u-taxreference/src/main/java/it/
    telecomitalia/ficep/idpproxy/spit/utils/JWTDemo.java


 public static String createJWTPV(String name,String familyName,String
     dateOfBirth,String fiscalNumber, String requestedAttributes) throws
     Exception {

        String pemEncodedRSAPrivateKey =
            readKeyAsString("/META-INF/private-key-jwt1.pem");
        RSAKey rsaKey = (RSAKey)
            JWK.parseFromPEMEncodedObjects(pemEncodedRSAPrivateKey);
        PrivateKey privateKey = rsaKey.toRSAPrivateKey();

        Instant now = Instant.now();
        Date expirationTime = Date.from(now.plus(expiresInMin,
            ChronoUnit.MINUTES));
        JwtBuilder builder = Jwts.builder()
        .setIssuedAt(Date.from(now))
        .setAudience("https://oauth2-client-eid4u.polito.it/")
        .setIssuer("https://idp-proxy-test-eid4u.polito.it/")
        .setSubject("id_token")
        .claim("fiscalNumber", fiscalNumber)
        .claim("name", name)
        .claim("familyName", familyName)
        .claim("dateOfBirth", dateOfBirth)
        .claim("requestedAttributes", requestedAttributes)
        .setExpiration(expirationTime)
        .signWith(privateKey);



        //Builds the JWT and serializes it to a compact, URL-safe string
        return builder.compact();

}
```

The JWT (*id_token*) has finally been generated and serialized into a *String*.

In the following section, RSA Private and Public Key (Section 3.3.8), I will describe how the private and public keys are generated using OpenSSL.

After being created and signed by the private key in the *idpproxy*, the *id_token* will then be validated and decoded by the public key in the *oauth2-authorization-server*.



eyJhbGciOiJSUzI1NiJ9.eyJpYXQiOjE2NjAyOT
g2MzYsImF1ZCI6Imh0dHBzOi8vb2F1dGgyLXNlc
nZlci1laWQ0dS5wb2xpdG8uaXQvIiwiaXNzIjoi
aHR0cHM6Ly9pZHAtcHJveHktdGVzdC1laWQ0dS5
wb2xpdG8uaXQvIiwic3ViIjoiaWRfdG9rZW4iLC
JmaXNjYWxDb2RlIjoiVElOSVQtU0xDWkdTMThBN
zlDOTI0TSIsImZhbWlseU5hbWUiOiJDYXR0YW5l
byIsIm5hbWUiOiJHaW9iYmUiLCJkYXRlT2ZCaXJ
0aCI6IjE5NzktMTEtMTUiLCJleHAiOjE2NjAyOT
kyMzZ9.LPqQyTUkRIaOIPpu0UiDrDLGW8zH-
22_qW8tvgNOAVwboxMKNr_gHcr2IwdOwRCJnkOB
lVQz7lNM5hPEo6GsLfdV6Iao2QhiB_5OdHb2Zgt
Qx0vmr5D2J_ByYUmowj443W91gDEFIJ-
ee1klR7VofRiEGMI4FsyzseeYMD5ZMak8fsb39Z
mScb-
trbnAHDsN5k4rngoLeplCsNqFftJRf0eLebNBnV
P95emKSwKI6F23yVIhIuWXRvBHuN_21M9qzXp-
m6a-tQDkDtI5d-2yLuzKGak7ZAzNmh-
WPrGROoztaLyLdb8PxIW3FJWI5LxPEmVfgzfSDD
Dwc4ftqLF84Q

```
{
  "alg": "RS256"
}
```

```
{
  "iat": 1660298636,
  "aud": "https://oauth2-server-eid4u.polito.it/",
  "iss": "https://idp-proxy-test-eid4u.polito.it/",
  "sub": "id_token",
  "fiscalCode": "TINIT-SLCZGS18A79C924M",
  "familyName": "Cattaneo",
  "name": "Giobbe",
  "dateOfBirth": "1979-11-15",
  "exp": 1660299236
}
```

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Public Key in SPKI, PKCS #1,
  X.509 Certificate, or JWK stri
  ng format.
  ,
  Private Key in PKCS #8, PKCS #
  1, or JWK string format. The k
  ey never leaves your browser.
```

Figure 3.2. id_token JWT structure. (Source: [2])

We may transmit the *id_token* to the oauth2-client using POST now that we have it. (Figure 5.1 - Step 1).

It's important to note that the *id_token* expires after 10 minutes.

Before I explain the OAuth 2.0 flow, I should mention that this solution uses two OAuth 2.0 flows, which are divided by the client name:

- **eidas_client:** This is the primary OAuth 2.0 flow that requests the *access_token* in order to access the resources.
- **Polito_AP:** This will be used for the exchange of *id_token*.

## 3.3 OAuth2 Authorization Server

With the help of Java EE and MicroProfile, we will offer an implementation of the OAuth 2.0 Authorization Framework[80] . Most crucially, use the Authorization Code grant type to implement the interaction between OAuth 2.0 roles.

We will implement the Authorization Endpoint, the Token Endpoint, as well as the JWK Key Endpoint, which is helpful for the Resource Server to retrieve the public key, for the most crucial job, the Authorization Server.

We are going to use a pre-registered store of clients and users, and obviously a JWT store for access tokens, because we want the solution to be straightforward and simple for a quick setup.

### 3.3.1 Authorization Code Grant Flow

The authorization server's */authorize* endpoint is redirected to by the client in order to request permission. The program issues a callback to this endpoint.

The end-user, or resource owner, is typically asked for approval by the authorization server. If the end user agrees, the authorization server then sends a code back to the callback.

After receiving this code, the application calls the authorization server's */token* endpoint with authentication. We define "authenticated" as the application presenting identification as part of this call. The token is returned by the authorization server if everything looks to be in order.

The resource server's API will verify the token when the program submits its request with the token in hand. Using the /introspect endpoint, it can request that the authorization server confirm the token. Alternatively, the resource server can optimize by locally confirming the signature of the token if it is self-contained, like with JWT.

We will concentrate on the most used grant type in this implementation, the **Authorization Code**.

### 3.3.2 Client and User Registration

Before it can allow a request, an authorization server need to know about the client and user. Also typical is the presence of a user interface on an authorization server.

But for the sake of simplicity, we will use the pre-configured client, which is saved in *src/-main/resources/data.sql.template* :

- eidas_client

```
INSERT INTO clients (client_id, client_secret, redirect_uri, scope,
    authorized_grant_types)
VALUES ('eidas_client', 'eidasclientsecret',
    'https://oauth2-client-eid4u.polito.it/callback',
'Citizenship CountryOfBirth CurrentDegree Email HomeInstitutionCountry
    HomeInstitutionIdentifier HomeInstitutionName IdIssuer IdNumber
    MaritalState Nationality Phone', 'authorization_code');
```

And a pre-configured user:

```
INSERT INTO users (user_id, password, roles, scopes)
VALUES ('user_server', 'usersecret', 'USER', 'Citizenship CountryOfBirth
    CurrentDegree Email HomeInstitutionCountry HomeInstitutionIdentifier
    HomeInstitutionName IdIssuer IdNumber MaritalState Nationality Phone',
    'authorization_code');
```

For the purposes of this implementation, it should be noted that we used plain-text passwords; however, in a production environment, passwords should be hashed.

### 3.3.3 Authorization Endpoint

The primary function of the authorization endpoint is to authenticate the user before requesting the scopes or permissions that the application requires.

However, since eIDAS has previously verified our identity, we may skip this step and still regard the user as authorized in our scenario.

The authorization endpoint can now begin processing the application's request (Figure 5.1 - Step 2), which must include the parameters **response_type, client_id, redirect_uri, scope, and state**. Additionally, there is a parameter called **openidconnect_code**.

In our scenario, the *client_id* should be a legitimate client from the client database table (**eidas_client**).

Additionally, the *redirect_uri*, if specified, must coincide with the information in the clients database record (**https://oauth2-client-eid4u.polito.it/callback**).

The *openidconnect_code*, which is the **Polito_AP** client's *authorization_code*, is manually generated in the *oauth2-client*'s *IndexServlet.java* file. We will utilize it in (Figure 5.1 - Step 3) to obtain the *id_token* and *access_token.*

Furthermore, *response_type* is **code** since we are using an authorization code.

We can temporally save these values in the session because permission is a multi-step process:

```
request.getSession().setAttribute("ORIGINAL_PARAMS", params);
```

- **HTTP GET Endpoint** that validates the secrets, the redirect URI, and the requested scope, saves all of these parameters in the session, and then redirects the user to *authorize.jsp*. It calls the client */token* endpoint in the meantime to obtain the *id_token* as we will discuss in Section 3.3.4, while the user can confirm their consent.
- **HTTP POST Endpoint** that, after receiving the user's consent from *authorize.jsp*, uses the previously saved parameters to populate a new authorization code model with the *authorization_code*, the *client_id*, the *approved scopes*, the *expiration date*, and so on, and then redirects back to the client on the *redirect_uri* by adding the new code and the state obtained at the beginning to the query parameters in order to prevent CSRF. The table must contain the *redirect_uri* as an allowed URI for redirection.

### 3.3.4 Citizen Identification Data Request

There is an additional step to pass it through OAuth 2.0 using *Polito_AP* client at this point (Figure 5.1 - Step 3), as we need the *id_token* in the authorization server.

Furthermore, since we already know the *authorization_code* for this client (*openidconnect_code*), we can just call the */token* endpoint to obtain the *id_token* as a json parameter with *access_token*. In this case, there is an HTTP GET request from the authorization server to the client, followed by an HTTP POST and JAX RS Client implementation. More information on these steps is provided in Section 3.4.4. (Figure 5.1 - Step 3,4).

In order to use the *id_token* in upcoming steps, we additionally used Redis to store it for a short period of time with a 2-minute expiration time.

```
Jedis jedis = new Jedis("redis");
jedis.set("id_token",id_token);
jedis.expire("id_token", 120); // Deleted after two minutes
jedis.close();
```

### 3.3.5 Authorization Form

For *eidas_client*, we get ready to redirect to the *authorize.jsp* page and ask the user what permissions the application may use:

```
String allowedScopes = checkUserScopes(user.getScopes(), requestedScope);
request.setAttribute("scopes", allowedScopes);
request.getRequestDispatcher("/authorize.jsp").forward(request, response);
```

### 3.3.6 User Scopes Approval

The user makes a choice at this stage (Figure 5.1 - Steps 6, 7, and 8), when the browser shows an authorized UI for them. The user's choice is then submitted by the browser via an HTTP POST:

```
@POST
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.TEXT_HTML)
public Response doPost(@Context HttpServletRequest request, @Context
    HttpServletResponse response,
MultivaluedMap<String, String> params) throws Exception {
        MultivaluedMap<String, String> originalParams =
        (MultivaluedMap<String, String>)
            request.getSession().getAttribute("ORIGINAL_PARAMS");

        // ...

        String approvalStatus = params.getFirst("approval_status"); // YES OR
            NO

        // ... if YES

        List<String> approvedScopes = params.get("scope");

        // ...
}
```

We then create a temporary code that makes reference to the **user_id, client_id, and redirect_uri**, all of which the application will utilize once it reaches the token endpoint.

So let's make an *AuthorizationCode JPA* Entity with an auto-generated id:

```
@Entity
@Table(name ="authorization_code")
public class AuthorizationCode {
        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        @Column(name = "code")
        private String code;

        //...

}
```

Then fill it with data:

```
AuthorizationCode authorizationCode = new AuthorizationCode();
authorizationCode.setClientId(clientId);
authorizationCode.setUserId(userId);
authorizationCode.setApprovedScopes(String.join(" ", authorizedScopes));
authorizationCode.setExpirationDate(LocalDateTime.now().plusMinutes(2));
authorizationCode.setRedirectUri(redirectUri);
```

The code attribute is automatically filled up when the bean is saved, allowing us to retrieve it and send it back to the client:

```
appDataRepository.save(authorizationCode);
String code = authorizationCode.getCode();
```

It's important to keep in mind that our permission number will expire in two minutes, so we should be as careful as we can with it. It can be brief because the client will immediately exchange it for an access token.

After that, we redirect to the application's *redirect_uri* (Figure 5.1 - Step 9), passing it the *code* and any *state* parameters it included in its */authorize* request:

```
StringBuilder sb = new StringBuilder(redirectUri);
// ...

sb.append("?code=").append(code);
String state = params.getFirst("state");
if (state != null) {
        sb.append("&state=").append(state);
}
URI location = UriBuilder.fromUri(sb.toString()).build();
return Response.seeOther(location).build();
```

Again, keep in mind that **redirectUri** is not the *redirect_uri* request parameter but rather whatever is present in the clients database.

In order to proceed, the client must first get this code and then use the token endpoint to convert it into an access token.

### 3.3.7 Token Endpoint

The token endpoint, in contrast to the authorization endpoint, does not require a browser to communicate with the client, so we will implement it as a JAX-RS endpoint (JAX-RS[81] provides portable APIs for creating, making available, and using Web applications created and implemented in accordance with the principles of REST architectural style):

```
@Path("token")
public class TokenEndpoint {
        List<String> supportedGrantTypes =
            Collections.singletonList("authorization_code");

        @Inject
        private AppDataRepository appDataRepository;

        @Inject
        Instance<AuthorizationGrantTypeHandler>
            authorizationGrantTypeHandlers;

        @POST
        @Produces(MediaType.APPLICATION_JSON)
        @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
        public Response token(MultivaluedMap<String, String> params,
        @HeaderParam(HttpHeaders.AUTHORIZATION) String authHeader) throws
            JOSEException {
                //...
        }
}
```

Both a POST and the application/x-www-form-urlencoded media type are required for the token endpoint's parameters.

As we previously explained, we will only support the grant type for an authorization code:

```
List<String> supportedGrantTypes =
    Collections.singletonList("authorization_code");
```

Therefore, it is necessary to support the received grant_type as a required parameter:

```
String grantType = params.getFirst("grant_type");
Objects.requireNonNull(grantType, "grant_type params is required");
if (!supportedGrantTypes.contains(grantType)) {
        JsonObject error = Json.createObjectBuilder()
        .add("error", "unsupported_grant_type")
        .add("error_description", "grant type should be one of :" +
            supportedGrantTypes)
        .build();
        return Response.status(Response.Status.BAD_REQUEST)
        .entity(error).build();
}
```

Then, using HTTP Basic authentication, we verify the client authentication. Specifically, we determine whether the *client_id* and *client_secret* supplied through the Authorization header correspond to a client that has already been registered:

```
String[] clientCredentials = extract(authHeader);
String clientId = clientCredentials[0];
String clientSecret = clientCredentials[1];
Client client = appDataRepository.getClient(clientId);
if (client == null || clientSecret == null ||
    !clientSecret.equals(client.getClientSecret())) {
        JsonObject error = Json.createObjectBuilder()
        .add("error", "invalid_client")
        .build();
        return Response.status(Response.Status.UNAUTHORIZED)
        .entity(error).build();
}
```

Finally, we assign a corresponding grant type handler to produce the *TokenResponse*:

```
public interface AuthorizationGrantTypeHandler {
        TokenResponse createAccessToken(String clientId,
            MultivaluedMap<String, String> params) throws Exception;
}
```

Since the authorization code grant type is the one in which we are most interested, we have offered a suitable implementation as a CDI bean and decorated it with the Named annotation:

```
@Named("authorization_code")
```

The CDI Instance mechanism at runtime activates the appropriate implementation in accordance with the received grant type value:

```
String grantType = params.getFirst("grant_type");
//...
AuthorizationGrantTypeHandler authorizationGrantTypeHandler =
authorizationGrantTypeHandlers.select(NamedLiteral.of(grantType)).get();
```

Time to create */token*'s response is now. (Figure 5.1 - Step 10).

### 3.3.8 RSA Private and Public Keys

**An RSA private key is required before creating the token so it can be signed.**

We will employ **OpenSSL** for this objective:

```
# PRIVATE KEY
openssl genpkey -algorithm RSA -out private-key.pem -pkeyopt
    rsa_keygen_bits:2048
```

Using the file *META-INF/microprofile-config.properties*, the server receives the *private-key.pem* through the MicroProfile Config signingKey property:

```
signingkey=/META-INF/private-key.pem
```

The inserted Config object allows the server to read the property:

```
String signingkey = config.getValue("signingkey", String.class);
```

A similar public key can be created by following these steps:

```
# PUBLIC KEY
openssl rsa -pubout -in private-key.pem -out public-key.pem
```

And to read it, use the MicroProfile Config verificationKey:

```
verificationkey=/META-INF/public-key.pem
```

The resource server should be able to access it on the server so that it can be verified. Through a JWK endpoint, this is done.

A useful library in this case is *Nimbus JOSE+JWT*. Adding the nimbus-jose-jwt dependency first:

```
<dependency>
       <groupId>com.nimbusds</groupId>
       <artifactId>nimbus-jose-jwt</artifactId>
       <version>7.7</version>
</dependency>
```

We can now make use of Nimbus's JWK support to make our endpoint simpler:

```
@Path("jwk")
@ApplicationScoped
public class JWKEndpoint {
@GET
public Response getKey(@QueryParam("format") String format) throws Exception {
//...
String verificationkey = config.getValue("verificationkey", String.class);
String pemEncodedRSAPublicKey = PEMKeyUtils.readKeyAsString(verificationkey);
if (format == null || format.equals("jwk")) {
       JWK jwk = JWK.parseFromPEMEncodedObjects(pemEncodedRSAPublicKey);
       return
           Response.ok(jwk.toJSONString()).type(MediaType.APPLICATION_JSON).build();
} else if (format.equals("pem")) {
       return Response.ok(pemEncodedRSAPublicKey).build();
}

//...
}
}
```

The format parameter has been used to alternate between the PEM and JWK formats. Both of these forms are supported by the MicroProfile JWT that we'll use to create the resource server.

### 3.3.9 Token Endpoint Response

Now the token response needs to be created by the specified *AuthorizationGrantTypeHandler*. We will only support the structured JWT Tokens in this implementation.

We will use the *Nimbus JOSE+JWT* library once more to create a token in this manner, although there are many more JWT libraries as well [80].

Therefore, in order to create a signed JWT, the JWT header must first be created:

```
JWSHeader jwsHeader = new
    JWSHeader.Builder(JWSAlgorithm.RS256).type(JOSEObjectType.JWT).build();
```

Then, we construct the payload, which is a collection of pre-built and original claims:

```
Instant now = Instant.now();
Long expiresInMin = 30L;
Date in30Min = Date.from(now.plus(expiresInMin, ChronoUnit.MINUTES));

JWTClaimsSet jwtClaims = new JWTClaimsSet.Builder()
.issuer("http://oauth2-server-eid4u.polito.it")
.subject(authorizationCode.getUserId())
.claim("upn", authorizationCode.getUserId())
.audience("http://oauth2-resource-eid4u.polito.it")
.claim("scope", authorizationCode.getApprovedScopes())
.claim("groups", Arrays.asList(authorizationCode.getApprovedScopes().split("
    ")))
```

```
.claim("fiscal_number", fiscal_number)
.expirationTime(in30Min)
.notBeforeTime(Date.from(now))
.issueTime(Date.from(now))
.jwtID(UUID.randomUUID().toString())
.build();
SignedJWT signedJWT = new SignedJWT(jwsHeader, jwtClaims);
```

The MicroProfile JWT requires *upn* and *groups* in addition to the regular JWT claims, thus we have added those as well. Both the *upn* and the *groups* will be mapped to Jakarta EE Roles and the Security *CallerPrincipal* in that platform.

Additionally, we have introduced a new claim called *fiscal_number*, which we will utilize in the Resource Server to retrieve academic data about the logged-in user that is denoted by the *fiscal_number*.

Due to the fact that *id_token* is a JWT, in order to obtain the *fiscal_number* from it, we must import the public key for that JWT into the authorization server. Once there, we can use the public key to verify the *fiscal_number* and decode it.

its public key can be found in:

```
resources/META-INF/public-key-jwt.pem
```

In the *JWTDemo.java* class, *decodeToken()* is used to both verify and decode it.

```
handler/JWTDemo.java

  public static Claims decodeToken(String token) throws Exception {

        String pemEncodedRSAPublicKey =
            readKeyAsString("/META-INF/public-key-jwt.pem");
        RSAKey rsaPublicKey = (RSAKey)
            JWK.parseFromPEMEncodedObjects(pemEncodedRSAPublicKey);
        PublicKey publicKey = rsaPublicKey.toRSAPublicKey();

        Claims claims = Jwts.parser()
        .setSigningKey(publicKey)
        .parseClaimsJws(token).getBody();
        return claims;

}
```

We must sign the *access_token* using an RSA private key now that we have the header and the content. The resource server can utilize the matching RSA public key to check the access token by exposing it over the JWK endpoint or making it accessible in another way.

Since the private key was given in PEM format, we should obtain it and convert it to an *RSAPrivateKey*:

```
SignedJWT signedJWT = new SignedJWT(jwsHeader, jwtClaims);
//...
String signingkey = config.getValue("signingkey", String.class);
String pemEncodedRSAPrivateKey =
    PEMKeyUtils.readKeyAsString(signingkey);
RSAKey rsaKey = (RSAKey)
    JWK.parseFromPEMEncodedObjects(pemEncodedRSAPrivateKey);
```

The JWT is then serialized and signed:

```
signedJWT.sign(new RSASSASigner(rsaKey.toRSAPrivateKey()));
String accessToken = signedJWT.serialize();
```

We then create a token response:

```
return Json.createObjectBuilder()
.add("token_type", "Bearer")
.add("access_token", accessToken)
.add("expires_in", expiresInMin * 60)
.add("scope", authorizationCode.getApprovedScopes())
.build();
```

which, due to JSON-P, is serialized to JSON format and transmitted to the client:

```
{
        "access_token": "acb6803a48114d9fb4761e403c17f812",
        "token_type": "Bearer",
        "expires_in": 1800,
        "scope": "Citizenship CountryOfBirth CurrentDegree Email
            HomeInstitutionCountry HomeInstitutionIdentifier
            HomeInstitutionName IdIssuer IdNumber MaritalState Nationality
            Phone"
}
```

## 3.4 OAuth2 Client

The Servlet, MicroProfile Config, and JAX RS Client APIs will be used in this part to create a web-based OAuth 2.0 Client.

To be more specific, we will be putting in place two main servlets: one for requesting the authorization server's authorization endpoint and obtaining a code using the authorization code grant type, and another servlet for using the code obtained to obtain an access token from the authorization server's token endpoint.

One more servlet will also be put into place to allow access to the resource server's APIs [80].

### 3.4.1 Client and User Registration

To authorize its requests, the Client (which is actually the authorization server for the second client *Polito_AP*) needs to be aware of the Client and User.

The pre-configured client is kept in *src/main/resources/data.sql.template*.

- Polito_AP

```
INSERT INTO clients (client_id, client_secret, redirect_uri, scope,
    authorized_grant_types)
VALUES ('Polito_AP', 'Polito_secret',
    'https://oauth2-client-eid4u.polito.it/token',
'openid eidas', 'authorization_code');

INSERT INTO users (user_id, password, roles, scopes)
VALUES ('user_client', 'usersecret', 'USER', 'openid eidas');
```

For the purposes of this implementation, it should be noted that we used plain-text passwords; however, in a production environment, passwords should be hashed.

### 3.4.2 OAuth 2.0 Client Details

Considering that the client is already registered with the authorization server, we must first provide the client registration details:

- *client_id*: Client Identifier is often given out by the authorization server at the time of registration.
- *client_secret*: Client Secret.
- *redirect_uri*: Location where the authorization code can be obtained.
- *scope*: Client requested permissions.

The client should also be aware of the authorization and token endpoints of the authorization server:

- *authorization_uri*: Location of the authorization server authorization endpoint that we can use to get a code.
- *token_uri*: Location of the authorization server token endpoint that we can use to obtain a token.

The client side's META-INF/microprofile-config.properties file, which contains all of this data, provides it:

```
# Client registration
client.clientId=eidas_client
client.clientSecret=eidasclientsecret
client.redirectUri=http://oauth2-client-eid4u.polito.it/callback
client.scope=Citizenship CountryOfBirth CurrentDegree Email
    HomeInstitutionCountry HomeInstitutionIdentifier HomeInstitutionName
    IdIssuer IdNumber MaritalState Nationality Phone
client.downstream=http://oauth2-client-eid4u.polito.it/downstream?action=read

# Polito Client registration
client.polito.clientId=Polito_AP
client.polito.scope=openid eidas
client.polito.redirectUri=http://oauth2-client-eid4u.polito.it/token


# Provider
provider.authorizationUri=https://oauth2-server-eid4u.polito.it/authorize
provider.tokenUri=https://oauth2-server-eid4u.polito.it/token

# Resource
resource.serverUri=https://oauth2-resource-eid4u.polito.it/api
```

#### openidconnect_code

We receive the *id_token* from the *idpproxy* in the *IndexServlet.java*, store it in Redis for a short period of time that will expire in 2 minutes, and use it in the subsequent Section 3.4.4.

```
Jedis jedis = new Jedis("redis");
jedis.set("id_token",request.getParameter("id_token"));
jedis.expire("id_token", 120); // Deleted after two minutes
jedis.close();
```

The client then produces an *authorization_code*, which is a random string, to be able to use the */token* endpoint in order to give the *id_token* to the authorization server because it is already registered within the oauth2-client (which functions as an authorization server for the *Polito_AP* client).

```
//generate a random unique code
String openidconnect_code = UUID.randomUUID().toString();

//creating the second client as Polito_AP and assign openid code
final AuthorizationCode authorizationCode = new AuthorizationCode();
authorizationCode.setClientId(config.getValue("client.polito.clientId",
    String.class));
authorizationCode.setUserId("user_client");
authorizationCode.setApprovedScopes(config.getValue("client.polito.scope",
    String.class));
authorizationCode.setExpirationDate(LocalDateTime.now().plusMinutes(2));
authorizationCode.setRedirectUri(config.getValue("client.polito.redirectUri",
    String.class));
authorizationCode.setCode(openidconnect_code);
appDataRepository.save(authorizationCode);
```

### 3.4.3 Authorization Code Request

The process of obtaining an *authorization_code* for the *eidas_client* begins at this point.

In order to begin, we must create and save a session's security state value:

```
String state = UUID.randomUUID().toString();
request.getSession().setAttribute("CLIENT_LOCAL_STATE", state);
```

Next, we obtain the client configuration data:

```
String authorizationUri = config.getValue("provider.authorizationUri",
    String.class);
String clientId = config.getValue("client.clientId", String.class);
String redirectUri = config.getValue("client.redirectUri", String.class);
String scope = config.getValue("client.scope", String.class);

String openidconnect_code = openidconnect_code;
```

Then, we will add these details as query parameters to the */authorize* endpoint of the authorization server:

```
String authorizationLocation = authorizationUri + "?response_type=code"
+ "&client_id=" + clientId
+ "&redirect_uri=" + redirectUri
+ "&scope=" + scope
+ "&state=" + state
+ "&openidconnect_code=" + openidconnect_code;
```

The browser will then be sent to this URL as a last step:

```
response.sendRedirect(authorizationLocation);
```

### 3.4.4 Citizen Identification Data Request

At this point, the *Polito_AP* client-related process will be carried out by the authorization server using the *openidconnect_code* (which serves as *Polito_AP*'s authorization code), which will call:

```
https://oauth2-client-eid4u.polito.it/token?code=A123&client_id=Polito_AP
    &scope=openid,eidas.
```

And it's actually calling the *GET /token* endpoint which will call an internal function to create an *access_token* with the *id_token*.

In fact, it's contacting the *GET /token* endpoint, which calls a procedure inside to generate an *access_token* using the *id_token*.

The *access_token* creation function is located in:

```
oauth2-client/src/main/java/com/homework/oauth2/client/handler/
    AuthorizationCodeGrantTypeHandler.java
```

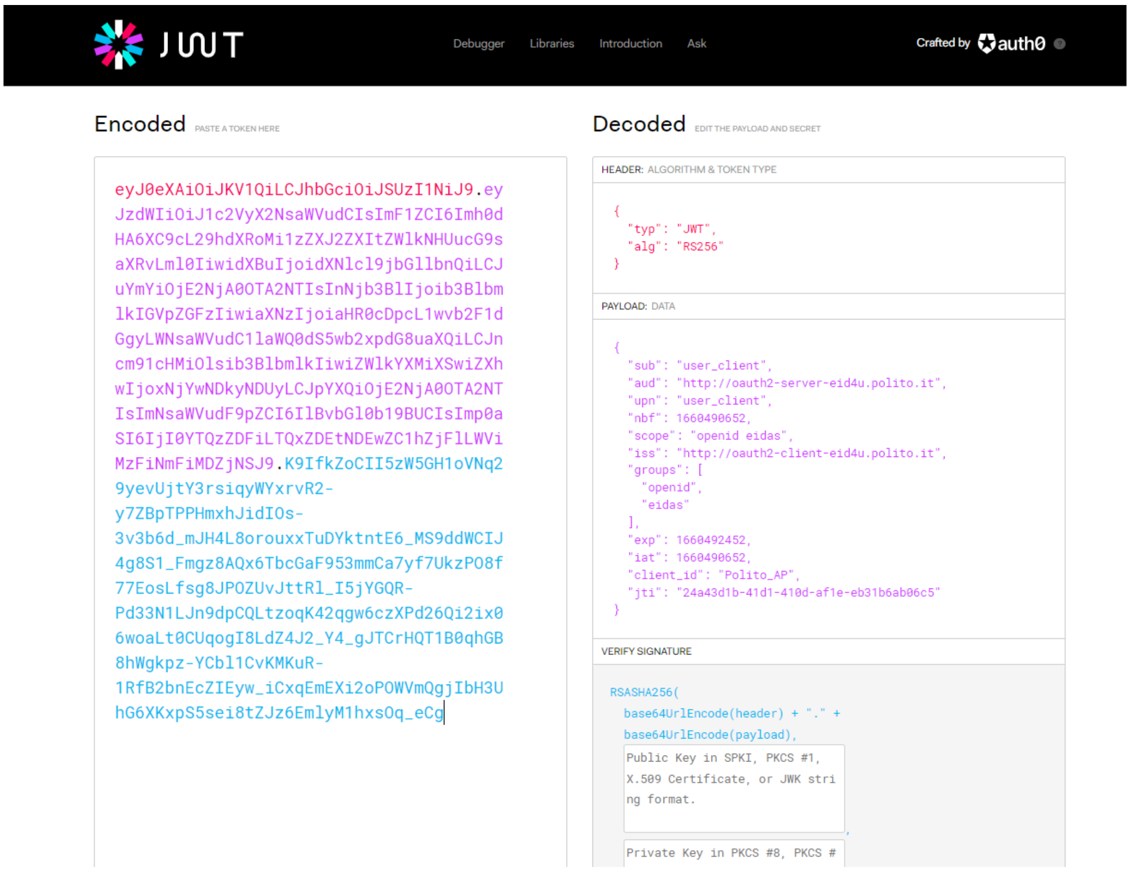
### 3.4.5 Citizen Identification Data Response

The *id_token* is first retrieved from Redis, after which the *access_token* is used to build the JWT Payload, and finally the *id_token* is added to the JSON response.

```
//get id_token
String id_token = "";
Jedis jedis = new Jedis("redis");
if (jedis.exists("id_token")) {
        id_token = jedis.get("id_token");
        jedis.del("id_token");
        jedis.close();
}
jedis.close();

//3. JWT Payload or claims
String accessToken = getAccessToken(clientId, authorizationCode.getUserId(),
    authorizationCode.getApprovedScopes());


return Json.createObjectBuilder()
.add("token_type", "Bearer")
.add("access_token", accessToken)
.add("expires_in", expiresInMin * 60)
.add("scope", authorizationCode.getApprovedScopes())
.add("id_token",id_token)
.build();
```

And will send it back to the server of authorization.

```
{
"token_type": "Bearer",
"access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJ1c2VyX2NsaWV
udCIsImF1ZCI6Imh 0dHA6XC9cL29hdXRoMi1zZXJ2ZXItZWlkNHUucG9saXRvLml0IiwidXBuIjoi
dXNlcl9jbGllbnQi LCJuYmYiOjE2NjA0OTA2NTIsInNjb3BlIjoib3BlbmlkIGVpZGFzIiwiaXNz
IjoiaHR0cDpcL1wvb 2F1dGgyLWNsaWVudC1laWQ0dS5wb2xpdG8uaXQiLCJncm91cHMiOlsib3Bl
bmlkIiwiZWlkYXMiXS wiZXhwIjoxNjYwNDkyNDUyLCJpYXQiOjE2NjA0OTA2NTIsImNsaWVudF9p
ZCI6IlBvbGl0b19BUCI sImp0aSI6IjI0YTQzZDFiLTQxZDEtNDEwZC1hZjFlLWViMzFiNmFiMDZjN
SJ9.K9IfkZoCII5zW5GH1oVNq29yevUjtY3rsiqyWYxrvR2-y7ZBpTPPHmxhJidIOs-3v3b6d_mJH4L
8orouxxTuDYktntE6 _MS9ddWCIJ4g8S1_Fmgz8AQx6TbcGaF953mmCa7yf7UkzPO8f77EosLfsg8
JPOZUvJttRl_I5jYGQ R-Pd33N1LJn9dpCQLtzoqK42qgw6czXPd26Qi2ix06woaLt0CUqogI8LdZ
4J2_Y4_gJTCrHQT1B0qhGB8hWgkpz-YCbl1CvKMKuR-1RfB2bnEcZIEyw_iCxqEmEXi2oPOWVmQgjI
bH3UhG6XKxpS5sei8t ZJz6EmlyM1hxsOq_eCg",
"expires_in": 1800,
"scope": "openid eidas",
"id_token": "eyJhbGciOiJSUzI1NiJ9.eyJpYXQiOjE2NjA0OTA2NDQsImF1ZCI6ImhOdHBzOi8
vb2F1dGgyLXN lcnZlci1laWQ0dS5wb2xpdG8uaXQvIiwiaXNzIjoiaHR0cHM6Ly9pZHAtcHJveHk
tdGVzdC1laWQ0 dS5wb2xpdG8uaXQvIiwic3ViIjoiaWRfdG9rZW4iLCJmaXNjYWxDb2RlIjoiVEl
OSVQtU0xDWkdTM ThBNzlDOTI0TSIsImZhbWlseU5hbWUiOiJDYXR0YW5lbyIsIm5hbWUiOiJHaW9
iYmUiLCJkYXRlT2 ZCaXJ0aCI6IjE5NzktMTEtMTUiLCJleHAiOjE2NjA0OTEyNDR9.BL7Iv1rZ1d
Ebsheib1eznTjCDU 9CpnhHjtaYKufsaDEqVSMtrWK7ir6g5LOiEOuKfgW6krvZDs4zQhk9jy9g23
ZCeqMeU5yy_-RxO2RUA9edvMBkWkpltAZjQqj3k3a8NQkEFjqa8i8LYbdybKnhKuzyor4QzvMPTS-i
65WpKEdabD1bzD62 o4jDOgOViz5-WCOWXyTmmz_xdnBFcRQcDyUkjadnLCx8vXGgVtHtYTPC3NZa3
kD9yYHUA7olj9IHd MNngVwxcoN37vRQpOz9HrtS-SGkgRLQfx2IeqVJRE7ZpTQNi3sVCRb4g6j3yt
RUoTFQapotPsY93O i4EmSDSw"
}
```



```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.ey
JzdWIiOiJ1c2VyX2NsaWVudCIsImF1ZCI6Imh0d
HA6XC9cL29hdXRoMi1zZXJ2ZXItZWlkNHUucG9s
aXRvLml0IiwidXBuIjoidXNlcl9jbGllbnQiLCJ
uYmYiOjE2NjA0OTA2NTIsInNjb3BlIjoib3Blbm
lkIGVpZGFzIiwiaXNzIjoiaHR0cDpcL1wvb2F1d
GgyLWNsaWVudC1laWQ0dS5wb2xpdG8uaXQiLCJn
cm91cHMiOlsib3BlbmlkIiwiZWlkYXMiXSwiZXh
wIjoxNjYwNDkyNDUyLCJpYXQiOjE2NjA0OTA2NT
IsImNsaWVudF9pZCI6IlBvbGl0b19BUCIsImp0a
SI6IjI0YTQzZDFiLTQxZDEtNDEwZC1hZjFlLWVi
MzFiNmFiMDZjNSJ9.K9IfkZoCII5zW5GH1oVNq2
9yevUjtY3rsiqyWYxrvR2-
y7ZBpTPPHmxhJidIOs-
3v3b6d_mJH4L8orouxxTuDYktntE6_MS9ddWCIJ
4g8S1_Fmgz8AQx6TbcGaF953mmCa7yf7UkzPO8f
77EosLfsg8JPOZUvJttRl_I5jYGQR-
Pd33N1LJn9dpCQLtzoqK42qgw6czXPd26Qi2ix0
6woaLt0CUqogI8LdZ4J2_Y4_gJTCrHQT1B0qhGB
8hWgkpz-YCbl1CvKMKuR-
1RfB2bnEcZIEyw_iCxqEmEXi2oPOWVmQgjIbH3U
hG6XKxpS5sei8tZJz6EmlyM1hxsOq_eCg
```

```
{
  "typ": "JWT",
  "alg": "RS256"
}
```

```
{
  "sub": "user_client",
  "aud": "http://oauth2-server-eid4u.polito.it",
  "upn": "user_client",
  "nbf": 1660490652,
  "scope": "openid eidas",
  "iss": "http://oauth2-client-eid4u.polito.it",
  "groups": [
    "openid",
    "eidas"
  ],
  "exp": 1660492452,
  "iat": 1660490652,
  "client_id": "Polito_AP",
  "jti": "24a43d1b-41d1-410d-af1e-eb31b6ab06c5"
}
```

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Public Key in SPKI, PKCS #1,
  X.509 Certificate, or JWK stri
  ng format.
  ,
  Private Key in PKCS #8, PKCS #
```

Figure 3.3. Access Token of Polito_AP. (Source: [2])

### 3.4.6 Authorization Code Response

After handling the request, the authorization endpoint of the authorization server will generate and add a code, along with the received state argument, to the *redirect_uri* and reroute the browser.

```
https://oauth2-client-eid4u.polito.it/callback?code=A123&state=Y.
```

### 3.4.7 Access Token Request

The *CallbackServlet.java* client's */callback* initializes by verifying the received state:

```
String localState = (String)
    request.getSession().getAttribute("CLIENT_LOCAL_STATE");
if (!localState.equals(request.getParameter("state"))) {
       request.setAttribute("error", "The state attribute doesn't match!");
       dispatch("/", request, response);
       return;
}
```

We will next use the code we previously got to make a token endpoint request for an access token from the authorization server:

```
String code = request.getParameter("code");
Client client = ClientBuilder.newClient();
WebTarget target = client.target(config.getValue("provider.tokenUri",
    String.class));

Form form = new Form();
form.param("grant_type", "authorization_code");
form.param("code", code);
form.param("redirect_uri", config.getValue("client.redirectUri",
    String.class));

TokenResponse tokenResponse = target.request(MediaType.APPLICATION_JSON_TYPE)
.header(HttpHeaders.AUTHORIZATION, getAuthorizationHeaderValue())
.post(Entity.entity(form, MediaType.APPLICATION_FORM_URLENCODED_TYPE),
    TokenResponse.class);
```

As we can see, there is no interaction with the browser during this call; instead, an HTTP POST request is made directly utilizing the JAX-RS client API.

The *client_id* and *client_secret* have been added to the Authorization header since the token endpoint needs client authentication.

The client can call the resource server APIs, which are covered in the following section, using this access token.

### 3.4.8 Protected Resource Access

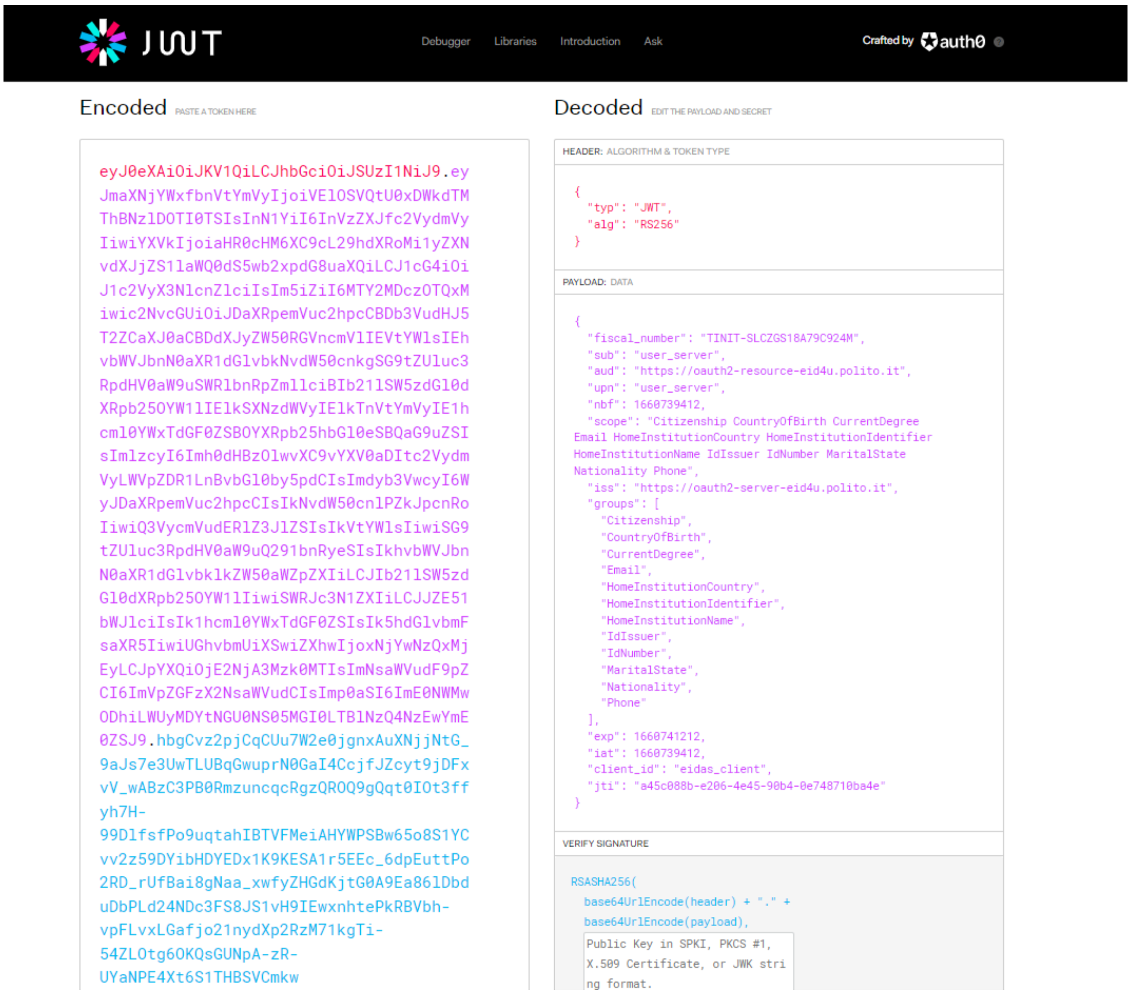
Now that we have a valid *access_token* (Figure 5.1 - Step 11), we can use the */read* APIs on the resource server.

The Authorization header must be provided in order to accomplish that. This is easily accomplished by using the Invocation.Builder header() method of the JAX-RS Client API:

```
resourceWebTarget = webTarget.path("resource/read");
Invocation.Builder invocationBuilder = resourceWebTarget.request();
response = invocationBuilder
.header("authorization", tokenResponse.getString("access_token"))
.get(String.class);
```

## 3.5 OAuth2 Resource Server

Building a secure online application using JAX-RS, MicroProfile JWT, and MicroProfile Config will be the focus of this part. The MicroProfile JWT is responsible for validating the incoming JWT and mapping the JWT scopes to roles in the Jakarta EE [80].



```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.ey
JmaXNjYWxfbnVtYmVyIjoiVElOSVQtU0xDWkdTM
ThBNzlDOTI0TSIsInN1YiI6InVzZXJfc2VydmVy
IiwiYXVkIjoiaHR0cHM6XC9cL29hdXRoMi1yZXN
vdXJjZS1laWQ0dS5wb2xpdG8uaXQiLCJ1cG4iOi
J1c2VyX3NlcnZlciIsIm5iZiI6MTY2MDczOTQxM
iwic2NvcGUiOiJDaXRpemVuc2hpcCBDb3VudHJ5
T2ZCaXJ0aCBDdXJyZW50RGVncmVlIEVtYWlsIEh
vbWVJbnN0aXR1dGlvbkNvdW50cnkgSG9tZUluc3
RpdHV0aW9uSWRlbnRpZmllciBIb21lSW5zdGl0d
XRpb25OYW1lIElkSXNzdWVyIElkTnVtYmVyIE1h
cml0YWxTdGF0ZSBOYXRpb25hbGl0eSBQaG9uZSI
sImlzcyI6Imh0dHBzOlwvXC9vYXV0aDItc2Vydm
VyLWVpZDR1LnBvbGl0by5pdCIsImdyb3VwcyI6W
yJDaXRpemVuc2hpcCIsIkNvdW50cnlPZkJpcnRo
IiwiQ3VycmVudERlZ3JlZSIsIkVtYWlsIiwiSG9
tZUluc3RpdHV0aW9uQ291bnRyeSIsIkhvbWVJbn
N0aXR1dGlvbklkZW50aWZpZXIiLCJIb21lSW5zd
Gl0dXRpb25OYW1lIiwiSWRJc3N1ZXIiLCJJZE51
bWJlciIsIk1hcml0YWxTdGF0ZSIsIk5hdGlvbmF
saXR5IiwiUGhvbmUiXSwiZXhwIjoxNjYwNzQxMj
EyLCJpYXQiOjE2NjA3Mzk0MTIsImNsaWVudF9pZ
CI6ImVpZGFzX2NsaWVudCIsImp0aSI6ImE0NWMw
ODhiLWUyMDYtNGU0NS05MGI0LTBlNzQ4NzEwYmE
0ZSJ9.hbgCvz2pjCqCUu7W2e0jgnxAuXNjjNtG_
9aJs7e3UwTLUBqGwuprN0GaI4CcjfJZcyt9jDFx
vV_wABzC3PB0RmzuncqcRgzQROQ9gQqt0IOt3ff
yh7H-
99DlfsfPo9uqtahIBTVFMeiAHYWPSBw65o8S1YC
vv2z59DYibHDYEDx1K9KESA1r5EEc_6dpEuttPo
2RD_rUfBai8gNaa_xwfyZHGdKjtG0A9Ea86lDbd
uDbPLd24NDc3FS8JS1vH9IEwxnhtePkRBVbh-
vpFLvxLGafjo21nydXp2RzM71kgTi-
54ZLOtg6OKQsGUNpA-zR-
UYaNPE4Xt6S1THBSVCmkw
```

```
{
  "typ": "JWT",
  "alg": "RS256"
}
```

```
{
  "fiscal_number": "TINIT-SLCZGS18A79C924M",
  "sub": "user_server",
  "aud": "https://oauth2-resource-eid4u.polito.it",
  "upn": "user_server",
  "nbf": 1660739412,
  "scope": "Citizenship CountryOfBirth CurrentDegree
Email HomeInstitutionCountry HomeInstitutionIdentifier
HomeInstitutionName IdIssuer IdNumber MaritalState
Nationality Phone",
  "iss": "https://oauth2-server-eid4u.polito.it",
  "groups": [
    "Citizenship",
    "CountryOfBirth",
    "CurrentDegree",
    "Email",
    "HomeInstitutionCountry",
    "HomeInstitutionIdentifier",
    "HomeInstitutionName",
    "IdIssuer",
    "IdNumber",
    "MaritalState",
    "Nationality",
    "Phone"
  ],
  "exp": 1660741212,
  "iat": 1660739412,
  "client_id": "eidas_client",
  "jti": "a45c088b-e206-4e45-90b4-0e748710ba4e"
}
```

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Public Key in SPKI, PKCS #1,
  X.509 Certificate, or JWK stri
  ng format.
```

Figure 3.4. Access Token of eidas_client. (Source: [2])

### 3.5.1 Maven Dependencies

Furthermore to the dependency Java EE Web API [82], we need also the MicroProfile Config [83] and MicroProfile JWT [84] APIs:

```
<dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-web-api</artifactId>
  <version>8.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.eclipse.microprofile.config</groupId>
  <artifactId>microprofile-config-api</artifactId>
  <version>1.3</version>
</dependency>
  <groupId>org.eclipse.microprofile.jwt</groupId>
  <artifactId>microprofile-jwt-auth-api</artifactId>
  <version>1.1</version>
</dependency>
```

### 3.5.2 JWT Authentication Mechanism

The Bearer Token Authentication technique is implemented by the MicroProfile JWT. The Authorization header's JWT is processed, a Jakarta EE Security Principal is made accessible as a JsonWebToken holding the JWT claims, and the scopes are mapped to Jakarta EE roles. For further information, look at the Jakarta EE Security API [76].

The LoginConfig annotation must be added to the JAX-RS application in order to enable the server's JWT authentication mechanism:

```
@ApplicationPath("/api")
@DeclareRoles({"Citizenship CountryOfBirth CurrentDegree Email
    HomeInstitutionCountry HomeInstitutionIdentifier HomeInstitutionName
    IdIssuer IdNumber MaritalState Nationality Phone"})
@LoginConfig(authMethod = "MP-JWT")
public class OAuth2ResourceServerApplication extends Application {
}
```

The RSA public key is also required by MicroProfile JWT in order to validate the JWT signature. Introspection or, for sake of simplicity, manually copying the key from the authorization server might be used to supply this. In both cases, we must specify where the public key is stored:

```
mp.jwt.verify.publickey.location=/META-INF/public-key.pem
```

Last but not least, the MicroProfile JWT must confirm the incoming JWT's iss claim, which must be present and match the value of the MicroProfile Config property:

```
mp.jwt.verify.issuer=https://oauth2-server-eid4u.polito.it
```

Typically, this is where the Authorization Server is situated.

### 3.5.3 The Secured Endpoints

We will build a resource API with a single endpoint just to show how it works.

Through the @RolesAllowed annotation, the scopes are restricted:

```
@Path("/resource")
@RequestScoped
public class ProtectedResource {

        @Inject
        private JsonWebToken principal;

        @GET
        @RolesAllowed("Citizenship CountryOfBirth CurrentDegree Email
            HomeInstitutionCountry HomeInstitutionIdentifier
            HomeInstitutionName IdIssuer IdNumber MaritalState Nationality
            Phone")
        @Path("/read")
        public String read() {
                return "Protected Resource accessed by : " +
                    principal.getName();
        }

}
```

Now that we have access to the resources, we can call an endpoint to retrieve academic information about the logged-in user. Since this information uses the fiscal number as a key, we read the *fiscal number* from the claim of the *access_token* before using a json file to search inside of it to retrieve the authorized attributes instead of having access to the Polito database. Finally, we return this information to the idpproxy by POST.

```
public Response read() {
String outputString = "";
JSONObject json = new JSONObject();
try {

        JSONParser parser = new JSONParser();

        Object obj = parser.parse(new FileReader("/AP.json"));

        JSONObject jsonObject = (JSONObject) obj;

        JSONArray users = (JSONArray) jsonObject.get("users");

        String[] requestedAttributes = ((String)
            principal.getClaim("scope")).split(" ");
        String fiscalNumber = (String) principal.getClaim("fiscal_number");

        for (int i = 0; i < users.size(); i++) {
                JSONObject user = (JSONObject) users.get(i);
                String fn = (String) user.get("fiscalNumber");
                if ( fn.equals(fiscalNumber)) {
                        for (String attribute : requestedAttributes) {
                                json.put(attribute, user.get(attribute));
                        }
                }
        }

        outputString = json.toString();
        System.out.println(outputString);

} catch (Exception e) {
        e.printStackTrace();
}

return Response.ok(outputString).build();
}
```

## 3.6 Dockerization

There is a Dockerfile for each server module that is only utilized by the docker-compose.yml to fully build and operate the solution locally [85]. The project includes Open Liberty [86] as a Maven dependency, which is used by the three servers as their server run-time. To properly output a customized version of the configuration file, several arguments are modifiable from the composition (e.g. for the User credentials definition or for the Client credentials definition).

docker-compose.yml:

```
# AuthZ service
app-server:
# Configuration for building the docker image for the authz server
image: eid4u/app-server
build:
```

```
context: .
dockerfile: ./oauth2/oauth2-authorization-server/Dockerfile
args:
# Secrets
client_id: eidas_client
client_secret: eidasclientsecret
ports:
- "9080:9080" # Forward the exposed port 9080 on the container to port 9080
    on the host machine
networks:
eidas_net:
aliases:
- app-server-0
restart: on-failure

# Resource service
app-res:
# Configuration for building the docker image for the client server
image: eid4u/app-res
build:
context: .
dockerfile: ./oauth2/oauth2-resource-server/Dockerfile
ports:
- "9280:9280" # Map the exposed port 9280 on the container to port 9280 on
    the host machine
depends_on:
- app-server
networks:
eidas_net:
aliases:
- app-res-0
restart: on-failure

# Client Service
app-client:
# Configuration for building the docker image for the resource server
image: eid4u/app-client
build:
context: .
dockerfile: ./oauth2/oauth2-client/Dockerfile
args:
# Secrets
client_id: eidas_client
client_secret: eidasclientsecret

server_client_id: Polito_AP
server_client_secret: Politosecret
ports:
- "9180:9180" # Map the exposed port 9180 on the container to port 9180 on
    the host machine
depends_on:
- app-res
networks:
eidas_net:
aliases:
- app-client-0
restart: on-failure
```

Additionally, we update nginx to include the virtual hosts for each server, each of which has a unique configuration that can be found in:

```
nginx/test/eid4u/conf/050oauth2.conf

nginx/test/eid4u/conf/060oauth2.conf

nginx/test/eid4u/conf/070oauth2.conf
```

They are added to the docker-compose.yml in reverse proxy part:

```
#
# Reverse proxy in front of eIDAS components
#
revproxy:
image: ficep/nginx:test
build: ./nginx/test
networks:
eidas_net:
aliases:
/..
- oauth2-server-eid4u.polito.it
- oauth2-client-eid4u.polito.it
- oauth2-resource-eid4u.polito.it
ports:
- "80:80"
- "443:443"
depends_on:
/..
- app-server
- app-res
- app-client
volumes:
/..
- "./nginx/test/eid4u/conf/050oauth2.conf:/etc/nginx/conf.d/050oauth2.conf:ro"
- "./nginx/test/eid4u/conf/060oauth2.conf:/etc/nginx/conf.d/060oauth2.conf:ro"
- "./nginx/test/eid4u/conf/070oauth2.conf:/etc/nginx/conf.d/070oauth2.conf:ro"
restart: on-failure
```

# Chapter 4

# User's Manual

The development environment for the entire manual will be a Linux system based on Ubuntu. Based on this presumption, all commands and processes are used.

## 4.1 Software Dependencies

There are a few fundamental prerequisites that must be installed before installing and running the implementation: Install Docker and Docker-Compose after installing Linux on the computer.

How to correctly install Docker on Ubuntu is described in the Docker documentation [85].

1. Install docker:

   ```
   sudo apt-get install docker
   ```

2. Install docker-compose:

   ```
   sudo apt-get install docker-compose
   ```

3. Set up docker run without sudo:

   ```
   sudo usermod -aG docker ${USER}
   sudo service docker restart
   su - ${USER}
   ```

   Docker install done.

## 4.2 Source Code

The repository on Github has the source code available:

1. Clone the git repository

   ```
   git clone https://github.com/torsec/eidas-oauth2.git
   ```

2. Run eIDAS OAuth2 using docker-compose

   ```
   sudo docker-compose up --build -d && docker-compose logs -f
   ```

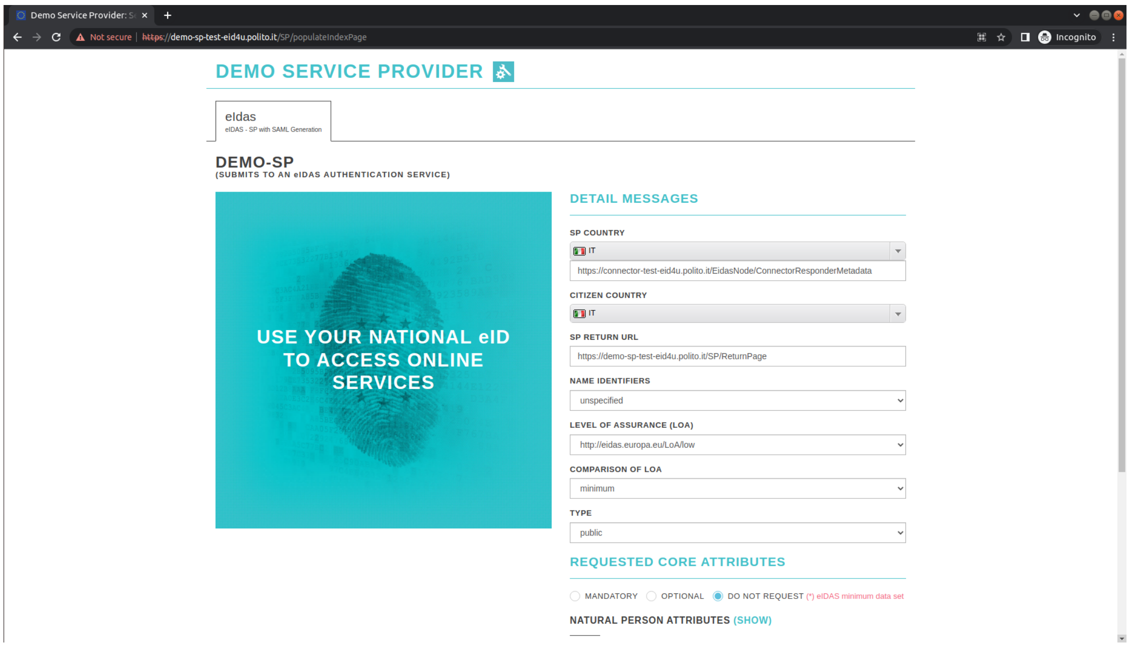
3. Since we have several domain names on a single server, we use Apache Virtual Host [87]. In order to do that, we need to set up nginx which we'll go over in the section on dockerization Section 3.6. The only thing left to do right now is to add domain names to the etc/hosts file after executing all of the docker containers, but first, we need to find out the gateway information for the running containers. To do this, we can use the docker network as shown below:

```
docker network ls
docker network inspect thesis_github_eidas_net

Add the following domains to the /etc/hosts file after copying the
    gateway:

sudo gedit /etc/hosts

*.*.*.* demo-sp-test-eid4u.polito.it
*.*.*.* service-test-eid4u.polito.it
*.*.*.* connector-test-eid4u.polito.it
*.*.*.* idp-proxy-test-eid4u.polito.it
*.*.*.* demo-ap-test-eid4u.polito.it
*.*.*.* demo-idp-test-eid4u.polito.it
*.*.*.* oauth2-server-eid4u.polito.it
*.*.*.* oauth2-client-eid4u.polito.it
*.*.*.* oauth2-resource-eid4u.polito.it
```

4. Launch the browser and enter the URL.

```
https://demo-sp-test-eid4u.polito.it/SP
```

In order to run the project you should follow these steps as shown in the following screens:


Demo Service Provider: S

Not secure | https://demo-sp-test-eid4u.polito.it/SP/populateIndexPage

DEMO SERVICE PROVIDER

eldas
eIDAS - SP with SAML Generation

DEMO-SP
(SUBMITS TO AN eIDAS AUTHENTICATION SERVICE)

USE YOUR NATIONAL eID TO ACCESS ONLINE SERVICES

DETAIL MESSAGES

SP COUNTRY
IT
https://connector-test-eid4u.polito.it/EidasNode/ConnectorResponderMetadata

CITIZEN COUNTRY
IT

SP RETURN URL
https://demo-sp-test-eid4u.polito.it/SP/ReturnPage

NAME IDENTIFIERS
unspecified

LEVEL OF ASSURANCE (LOA)
http://eidas.europa.eu/LoA/low

COMPARISON OF LOA
minimum

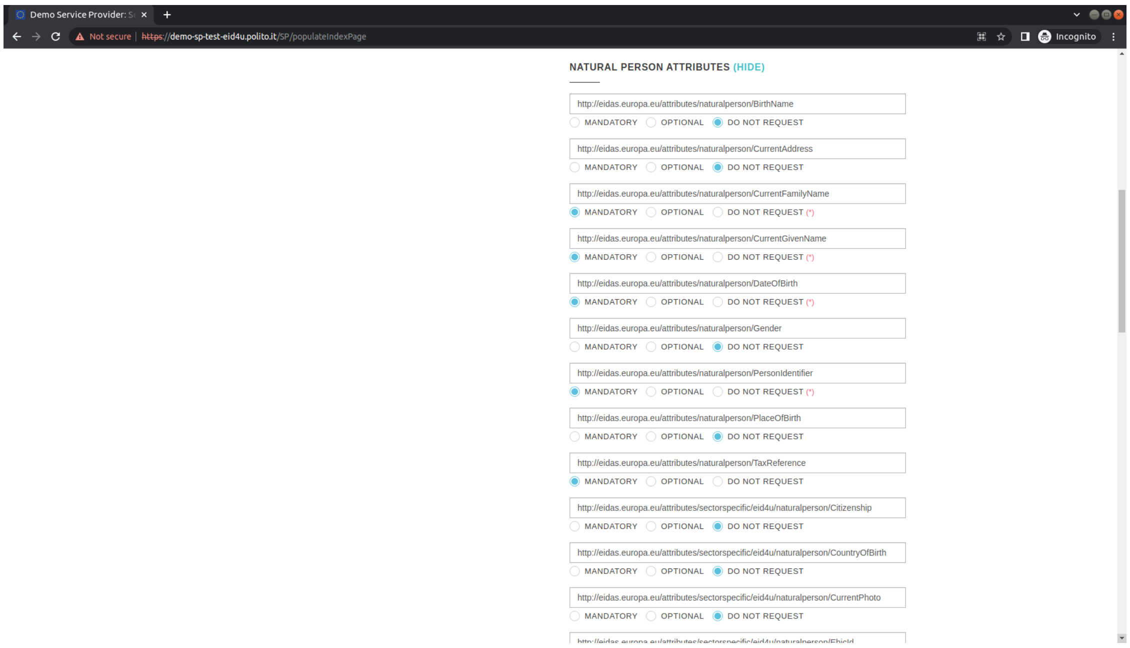
TYPE
public

REQUESTED CORE ATTRIBUTES

MANDATORY OPTIONAL [✓ DO NOT REQUEST] (*) eIDAS minimum data set

NATURAL PERSON ATTRIBUTES (SHOW)

Figure 4.1. Screen Step-1-0.



Demo Service Provider: S

Not secure | https://demo-sp-test-eid4u.polito.it/SP/populateIndexPage

Incognito

NATURAL PERSON ATTRIBUTES (HIDE)

http://eidas.europa.eu/attributes/naturalperson/BirthName
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/naturalperson/CurrentAddress
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/naturalperson/CurrentFamilyName
[✓ MANDATORY] OPTIONAL DO NOT REQUEST (*)

http://eidas.europa.eu/attributes/naturalperson/CurrentGivenName
[✓ MANDATORY] OPTIONAL DO NOT REQUEST (*)

http://eidas.europa.eu/attributes/naturalperson/DateOfBirth
[✓ MANDATORY] OPTIONAL DO NOT REQUEST (*)

http://eidas.europa.eu/attributes/naturalperson/Gender
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/naturalperson/PersonIdentifier
[✓ MANDATORY] OPTIONAL DO NOT REQUEST (*)

http://eidas.europa.eu/attributes/naturalperson/PlaceOfBirth
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

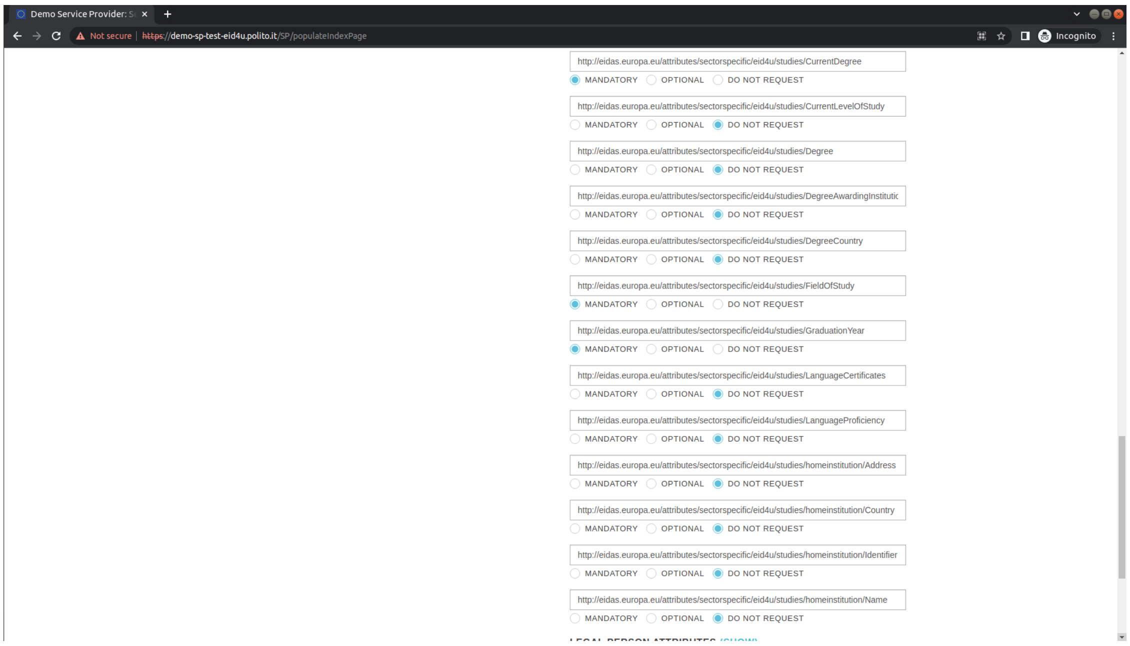
http://eidas.europa.eu/attributes/naturalperson/TaxReference
[✓ MANDATORY] OPTIONAL DO NOT REQUEST

http://eidas.europa.eu/attributes/sectorspecific/eid4u/naturalperson/Citizenship
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/naturalperson/CountryOfBirth
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/naturalperson/CurrentPhoto
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

Figure 4.2. Screen Step 1-1.



Demo Service Provider: S

Not secure | https://demo-sp-test-eid4u.polito.it/SP/populateIndexPage

Incognito

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/CurrentDegree
[✓ MANDATORY] OPTIONAL DO NOT REQUEST

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/CurrentLevelOfStudy
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/Degree
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/DegreeAwardingInstitutic
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/DegreeCountry
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/FieldOfStudy
[✓ MANDATORY] OPTIONAL DO NOT REQUEST

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/GraduationYear
[✓ MANDATORY] OPTIONAL DO NOT REQUEST

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/LanguageCertificates
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/LanguageProficiency
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

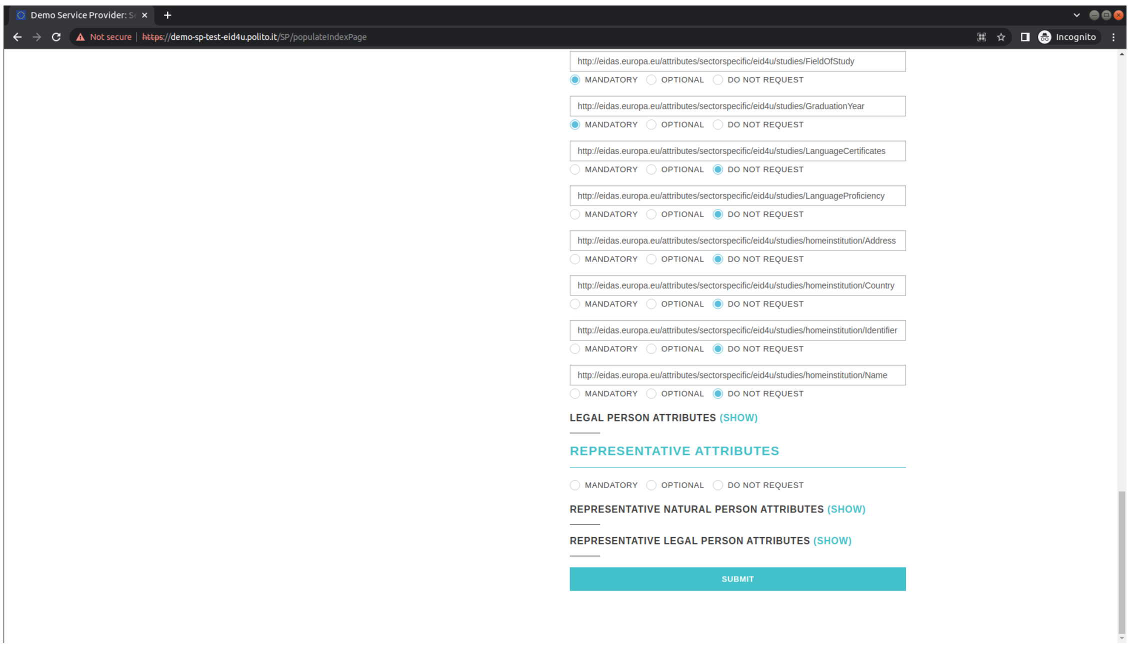
http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/homeinstitution/Address
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/homeinstitution/Country
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/homeinstitution/Identifier
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/homeinstitution/Name
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

Figure 4.3. Screen Step 1-2.



Demo Service Provider: S
Not secure | ~~https~~://demo-sp-test-eid4u.polito.it/SP/populateIndexPage

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/FieldOfStudy
[✓ MANDATORY] OPTIONAL DO NOT REQUEST

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/GraduationYear
[✓ MANDATORY] OPTIONAL DO NOT REQUEST

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/LanguageCertificates
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/LanguageProficiency
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/homeinstitution/Address
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/homeinstitution/Country
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/homeinstitution/Identifier
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

http://eidas.europa.eu/attributes/sectorspecific/eid4u/studies/homeinstitution/Name
MANDATORY OPTIONAL [✓ DO NOT REQUEST]

LEGAL PERSON ATTRIBUTES (SHOW)

REPRESENTATIVE ATTRIBUTES

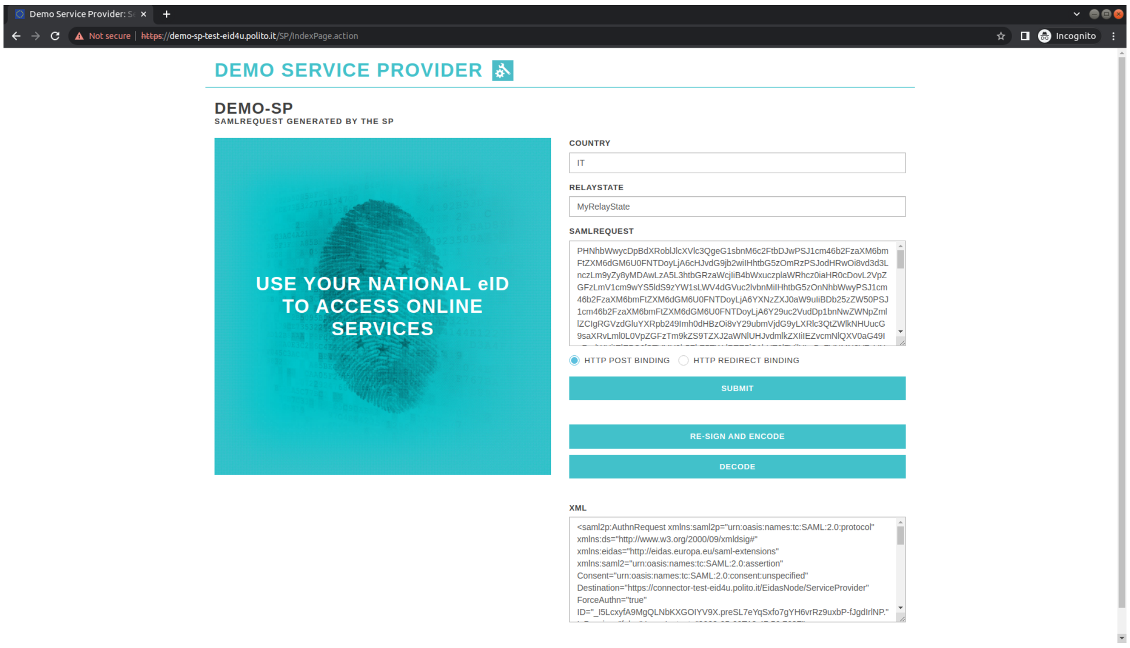
MANDATORY OPTIONAL DO NOT REQUEST

REPRESENTATIVE NATURAL PERSON ATTRIBUTES (SHOW)

REPRESENTATIVE LEGAL PERSON ATTRIBUTES (SHOW)

SUBMIT

Figure 4.4. Screen Step 1-3.



Demo Service Provider: S
Not secure | ~~https~~://demo-sp-test-eid4u.polito.it/SP/IndexPage.action

DEMO SERVICE PROVIDER

DEMO-SP
SAMLREQUEST GENERATED BY THE SP

COUNTRY
IT

RELAYSTATE
MyRelayState

SAMLREQUEST

```
PHNhbWwycDpBdXRoblJlcXVlc3QgeG1sbnM6c2FtbDJwPSJ1cm46b2FzaXM6bm
FtZXM6dGM6U0FNTDoyLjA6cHJvdG9jb2wiIHhtbG5zOmRzPSJodHRwOi8vd3d3L
nczLm9yZy8yMDAwLzA5L3htbGRzaWcjIiB4bWxuczplaWRhcz0iaHR0cDovL2VpZ
GFzLmV1cm9wYS5ldS9zYW1sLWV4dGVuc2lvbnMiIHhtbG5zOnNhbWwyPSJ1cm
46b2FzaXM6bmFtZXM6dGM6U0FNTDoyLjA6YXNzZXJ0aW9uIiBDb25zZW50PSJ
1cm46b2FzaXM6bmFtZXM6dGM6U0FNTDoyLjA6Y29uc2VudDp1bnNwZWNpZml
lZCIgRGVzdGluYXRpb249Imh0dHBzOi8vY29ubmVjdG9yLXRlc3QtZWlkNHUucG
9saXRvLml0L0VpZGFzTm9kZS9TZXJ2aWNlUHJvdmlkZXIiIEZvcmNlQXV0aG49I
```

[✓ HTTP POST BINDING] HTTP REDIRECT BINDING

SUBMIT

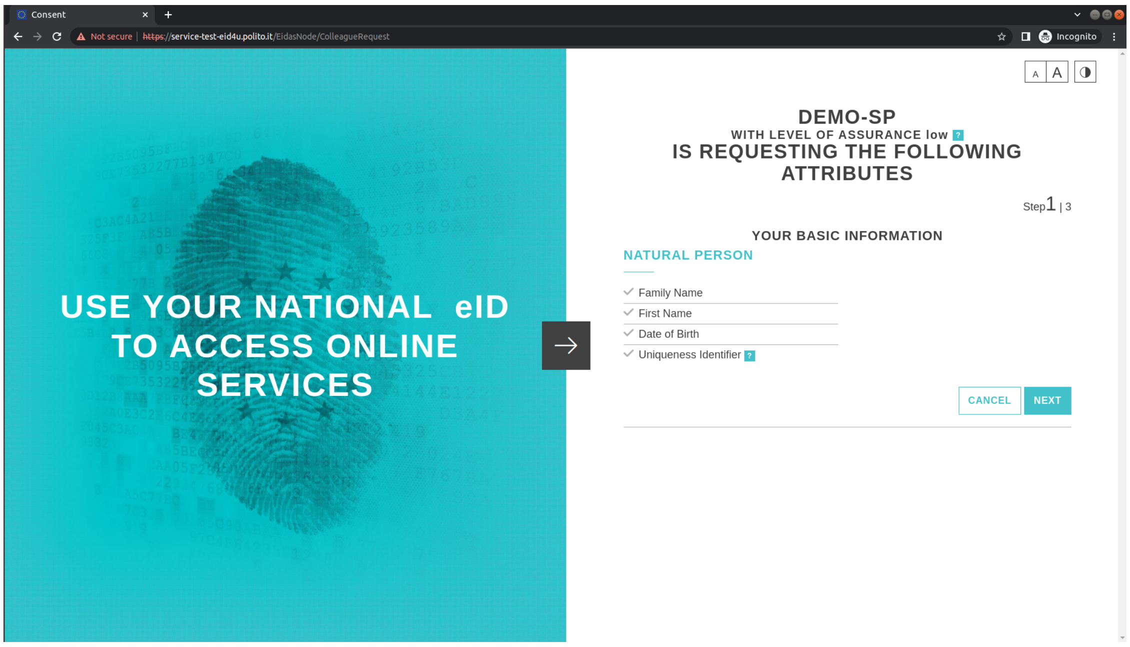
RE-SIGN AND ENCODE

DECODE

XML

```
<saml2p:AuthnRequest xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:eidas="http://eidas.europa.eu/saml-extensions"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
Consent="urn:oasis:names:tc:SAML:2.0:consent:unspecified"
Destination="https://connector-test-eid4u.polito.it/EidasNode/ServiceProvider"
ForceAuthn="true"
ID="_I5LcxyfA9MgQLNbKXGOIYV9X.preSL7eYqSxfo7gYH6vrRz9uxbP-fJgdIrlNP."
```

Figure 4.5. Screen Step 2.



Consent

Not secure | https://service-test-eid4u.polito.it/EidasNode/ColleagueRequest

USE YOUR NATIONAL eID TO ACCESS ONLINE SERVICES

DEMO-SP
WITH LEVEL OF ASSURANCE low
IS REQUESTING THE FOLLOWING ATTRIBUTES

Step 1 | 3

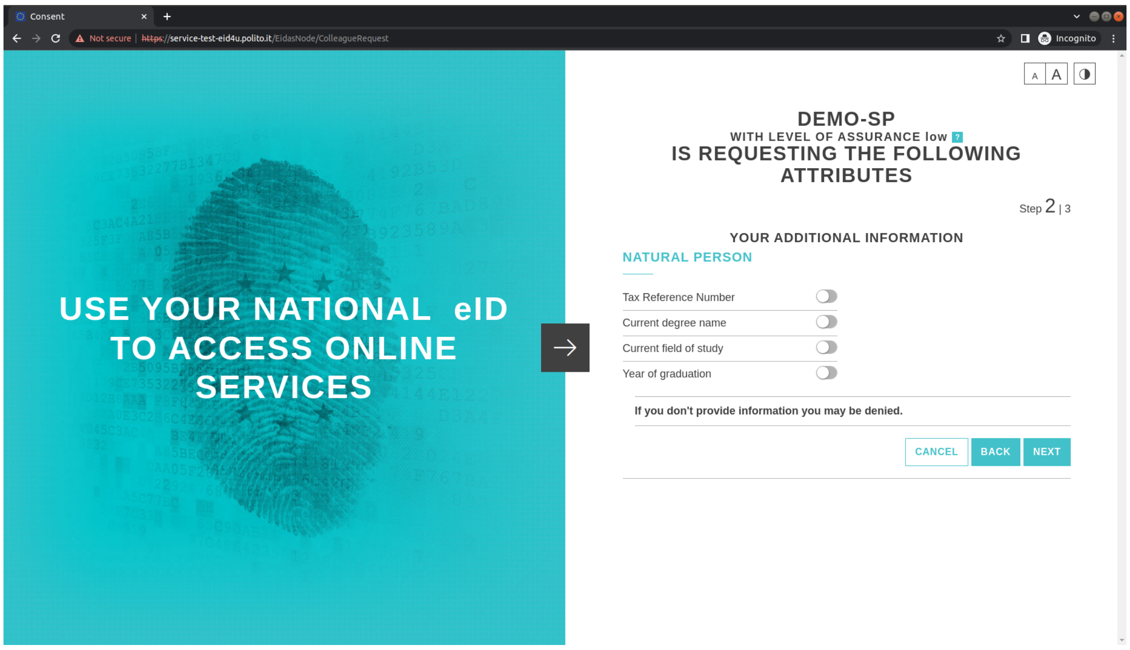
YOUR BASIC INFORMATION

NATURAL PERSON

✓ Family Name
✓ First Name
✓ Date of Birth
✓ Uniqueness Identifier

CANCEL NEXT

Figure 4.6. Screen Step 3-0.



Consent

Not secure | https://service-test-eid4u.polito.it/EidasNode/ColleagueRequest

USE YOUR NATIONAL eID TO ACCESS ONLINE SERVICES

DEMO-SP
WITH LEVEL OF ASSURANCE low
IS REQUESTING THE FOLLOWING ATTRIBUTES

Step 2 | 3

YOUR ADDITIONAL INFORMATION

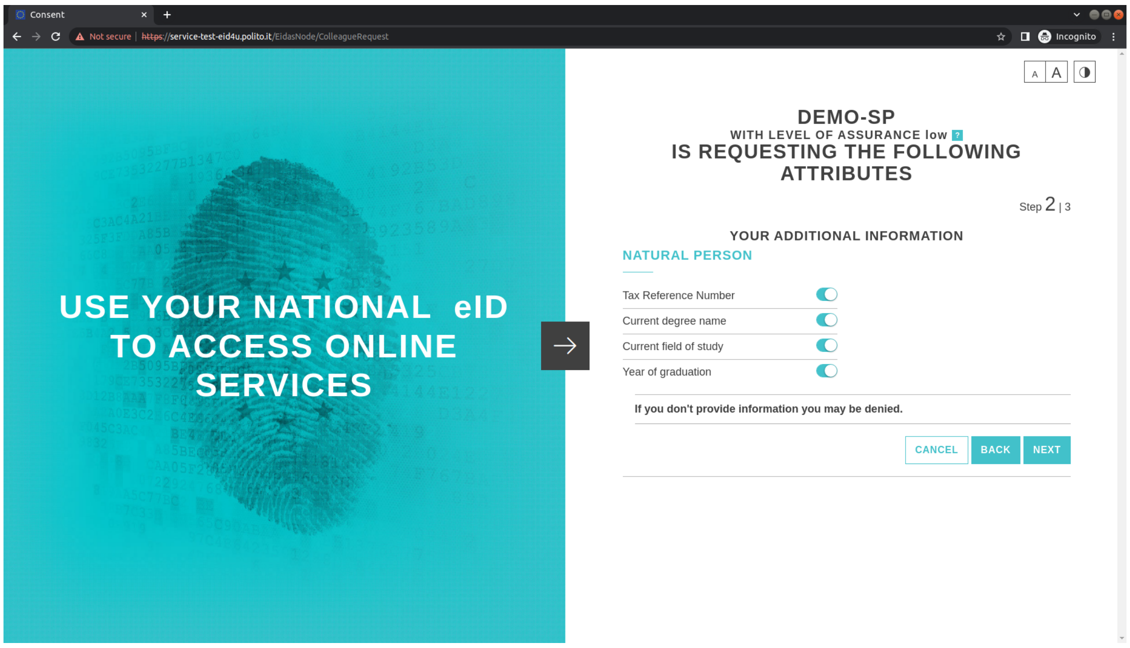
NATURAL PERSON

Tax Reference Number
Current degree name
Current field of study
Year of graduation

If you don't provide information you may be denied.

CANCEL BACK NEXT

Figure 4.7. Screen Step 3-1.


Consent
Not secure | https://service-test-eid4u.polito.it/EidasNode/ColleagueRequest
Incognito
USE YOUR NATIONAL eID TO ACCESS ONLINE SERVICES

DEMO-SP
WITH LEVEL OF ASSURANCE low
IS REQUESTING THE FOLLOWING ATTRIBUTES

Step 2 | 3

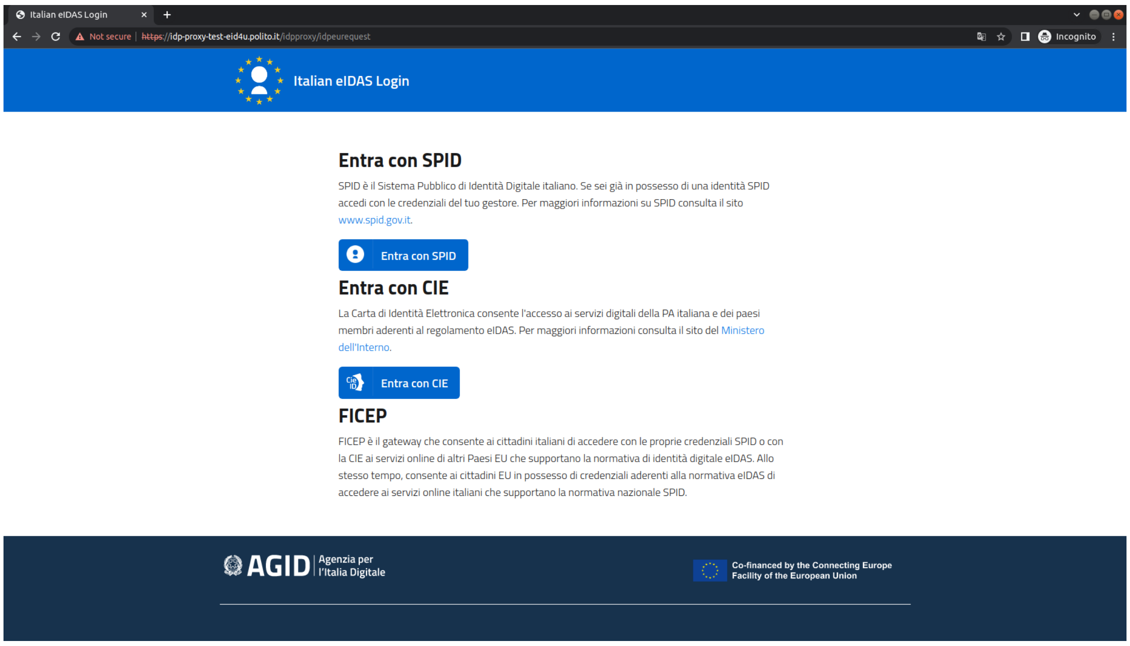
YOUR ADDITIONAL INFORMATION

NATURAL PERSON

Tax Reference Number
Current degree name
Current field of study
Year of graduation

If you don't provide information you may be denied.

Figure 4.8. Screen Step 3-2.


Italian eIDAS Login
Not secure | https://idp-proxy-test-eid4u.polito.it/idpproxy/idpeurequest
Incognito
Italian eIDAS Login

**Entra con SPID**

SPID è il Sistema Pubblico di Identità Digitale italiano. Se sei già in possesso di una identità SPID accedi con le credenziali del tuo gestore. Per maggiori informazioni su SPID consulta il sito www.spid.gov.it.

**Entra con CIE**

La Carta di Identità Elettronica consente l'accesso ai servizi digitali della PA italiana e dei paesi membri aderenti al regolamento eIDAS. Per maggiori informazioni consulta il sito del Ministero dell'Interno.

**FICEP**

FICEP è il gateway che consente ai cittadini italiani di accedere con le proprie credenziali SPID o con la CIE ai servizi online di altri Paesi EU che supportano la normativa di identità digitale eIDAS. Allo stesso tempo, consente ai cittadini EU in possesso di credenziali aderenti alla normativa eIDAS di accedere ai servizi online italiani che supportano la normativa nazionale SPID.

AGID Agenzia per l'Italia Digitale
Co-financed by the Connecting Europe Facility of the European Union

Figure 4.9. Screen Step 4.

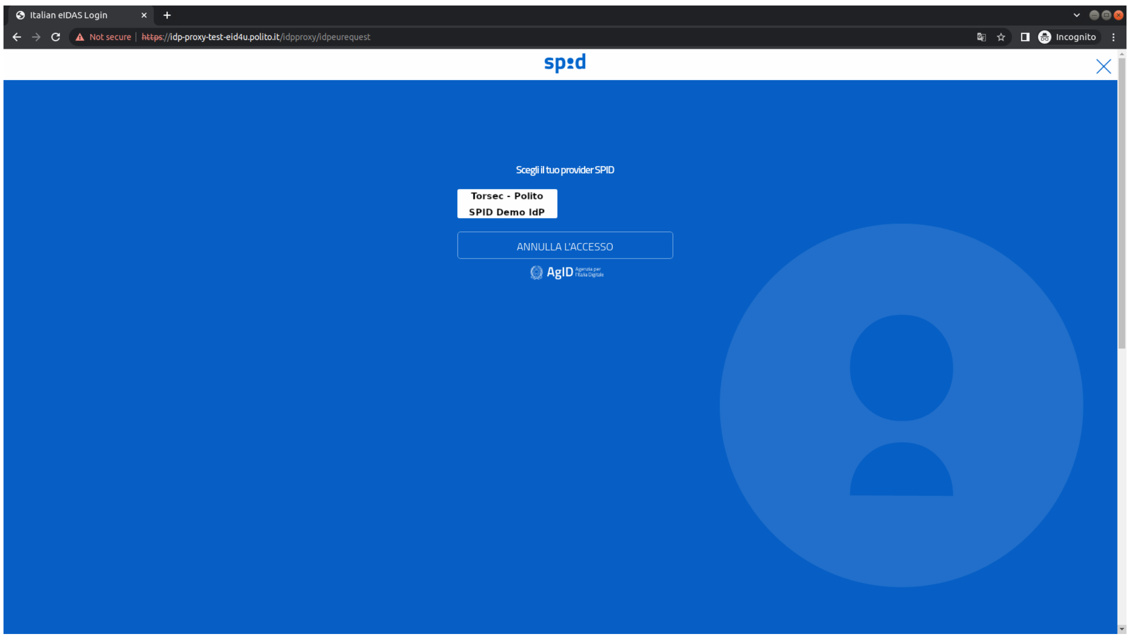
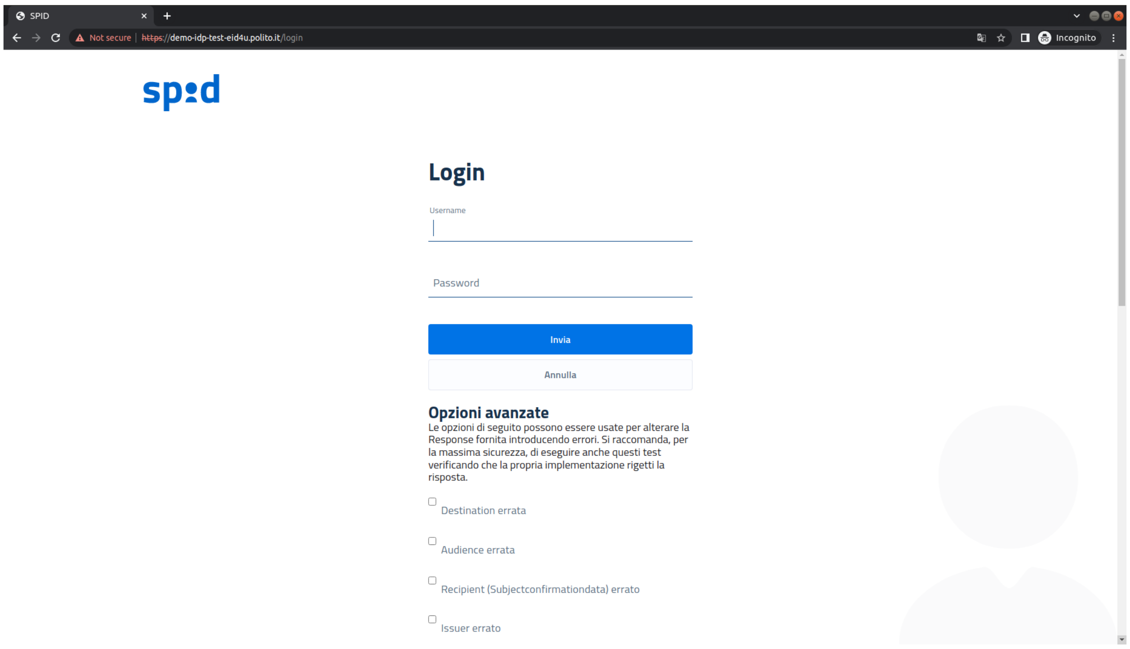

Italian eIDAS Login
Not secure | https://idp-proxy-test-eid4u.polito.it/idpproxy/idpeurequest
Incognito
spid
Scegli il tuo provider SPID
Torsec - Polito
SPID Demo IdP
ANNULLA L'ACCESSO
AgID

Figure 4.10. Screen Step 5.

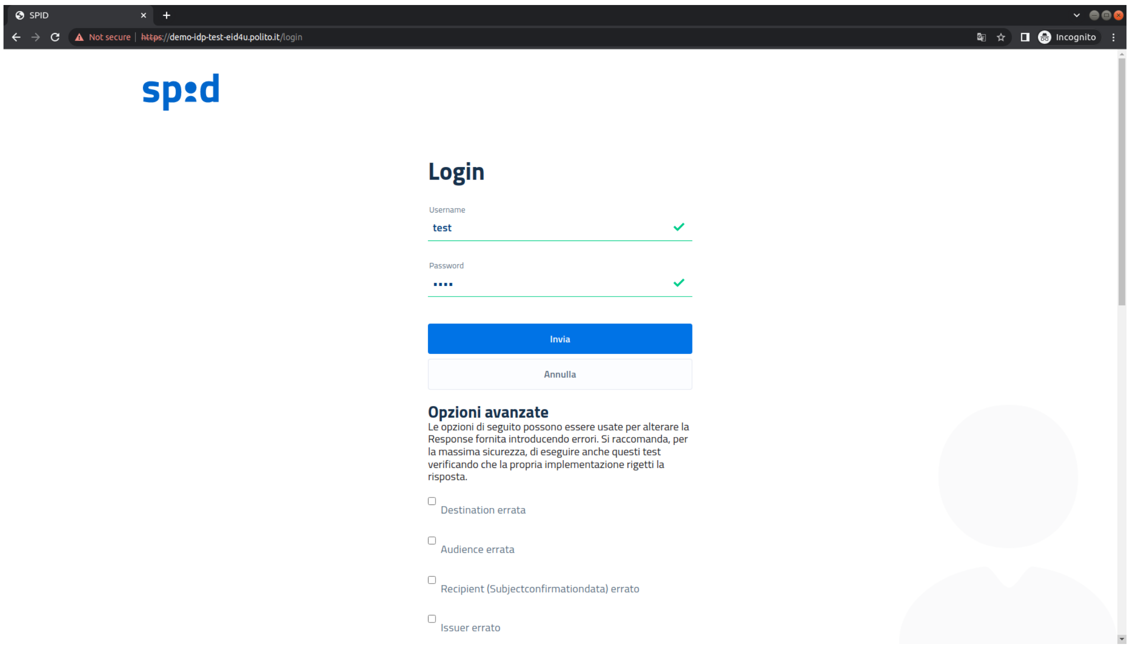


Opzioni avanzate

Le opzioni di seguito possono essere usate per alterare la Response fornita introducendo errori. Si raccomanda, per la massima sicurezza, di eseguire anche questi test verificando che la propria implementazione rigetti la risposta.

- [ ] Destination errata
- [ ] Audience errata
- [ ] Recipient (Subjectconfirmationdata) errato
- [ ] Issuer errato

Figure 4.11. Screen Step 6-0.



**Login**

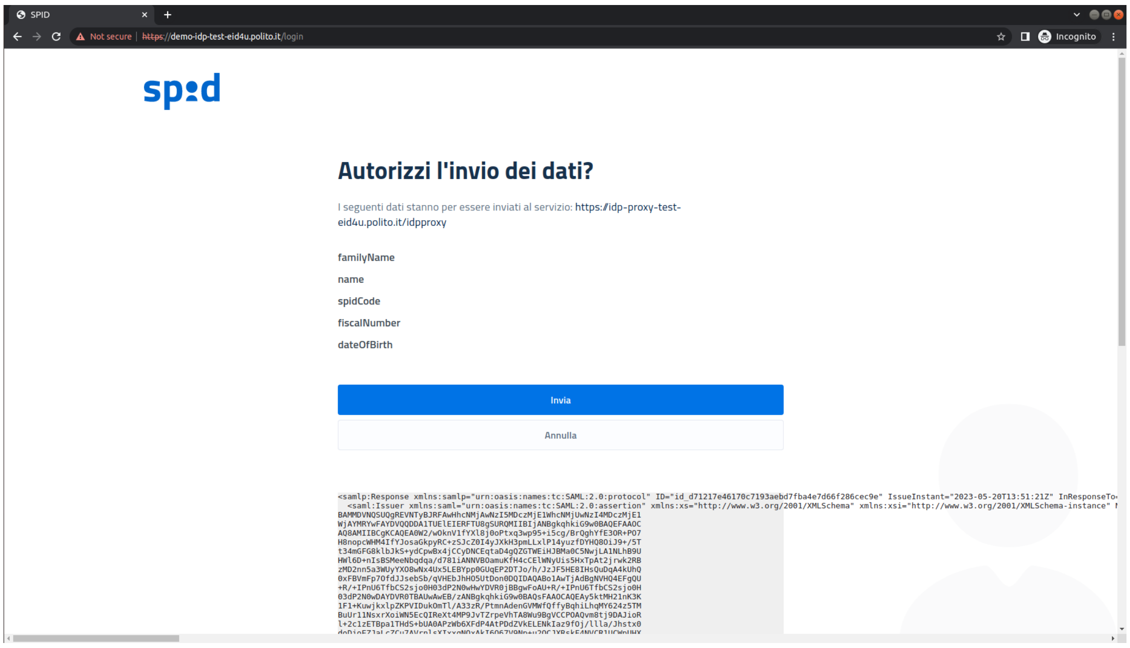
Username
test

Password
••••

**Opzioni avanzate**
Le opzioni di seguito possono essere usate per alterare la Response fornita introducendo errori. Si raccomanda, per la massima sicurezza, di eseguire anche questi test verificando che la propria implementazione rigetti la risposta.

- [ ] Destination errata
- [ ] Audience errata
- [ ] Recipient (Subjectconfirmationdata) errato
- [ ] Issuer errato

Figure 4.12. Screen Step 6-1.



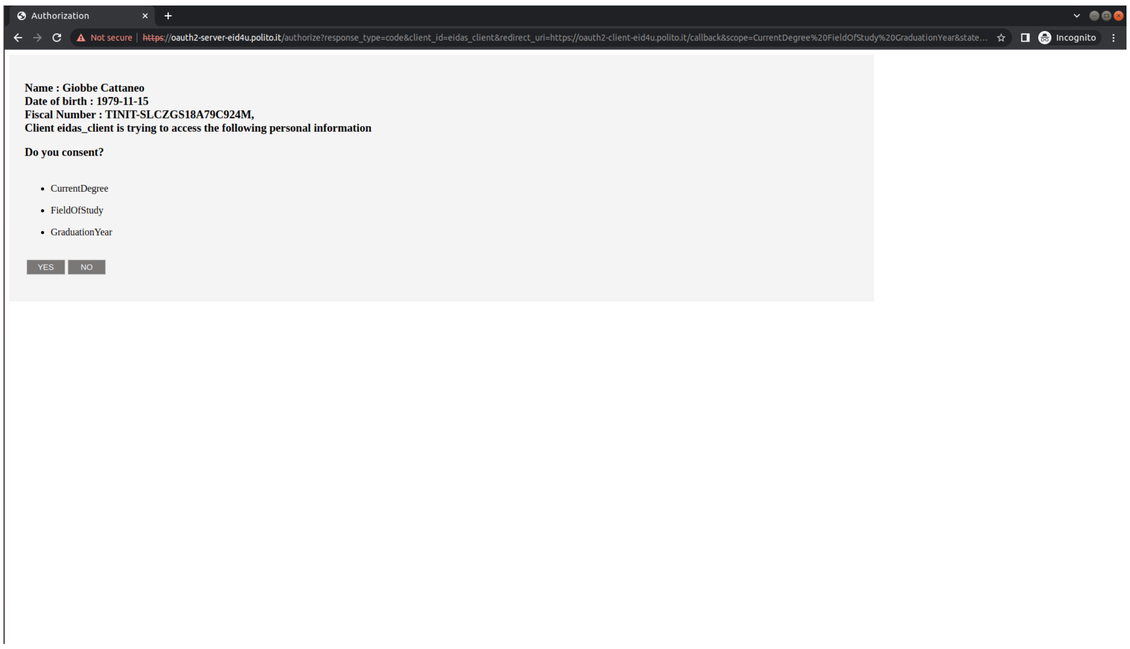
**Autorizzi l'invio dei dati?**

I seguenti dati stanno per essere inviati al servizio: https://idp-proxy-test-eid4u.polito.it/idpproxy

familyName
name
spidCode
fiscalNumber
dateOfBirth

```
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" ID="id_d71217e46170c7193aebd7fba4e7d66f286cec9e" IssueInstant="2023-05-20T13:51:21Z" InResponseTo=
  <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
BAMMDVNQSUQgREVNTyBJRFAwHhcNMjAwNzI5MDczMjE1WhcNMjUwNzI4MDczMjE1
WjAYMRYwFAYDVQQDDA1TUElEIERFTU8gSURQMIIBIjANBgkqhkiG9w0BAQEFAAOC
AQ8AMIIBCgKCAQEA0W2/wOknV1fYXl8j0oPtxq3wp95+i5cg/BrQghYfE3OR+PO7
H8nopcWHM4IfYJosaGkpyRC+zSJcZ0I4yJXkH3pmLLxlP14yuzfDYHQ8OiJ9+/5T
t34mGFG8klbJkS+ydCpwBx4jCCyDNCEqtaD4gQZGTWEiHJBMa0C5NwjLA1NLhB9U
HWl6D+nIsBSMeeNbqdqa/d781iANNVBOamuKfH4cCElWNyUis5HxTpAt2jrwk2RB
zMD2nn5a3WUyYXO8wNx4Ux5LEBYpp0GUqEP2DTJo/h/JzJF5HE8IHsQuDqA4kUhQ
0xFBVmFp7OfdJJsebSb/qVHEbJhHO5UtDon0DQIDAQABo1AwTjAdBgNVHQ4EFgQU
+R/+IPnU6TfbCS2sjo0H03dP2N0wHwYDVR0jBBgwFoAU+R/+IPnU6TfbCS2sjo0H
03dP2N0wDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAy5ktMH21nK3K
1F1+KuwjkxlpZKPVIDukOmTl/A33zR/PtmnAdenGVMWfQffyBqhiLhqMY624z5TM
BuUr11NsxrXoiWN5EcQIReXt4MP9JvTZrpeVhTA8Wu9BgVCCPOAQvm8tj9DAJioR
l+2c1zETBpa1THdS+bUA0APzWb6XFdP4AtPDdZVkELENkIaz9fOj/llla/Jhstx0
```

Figure 4.13. Screen Step 7.


Authorization

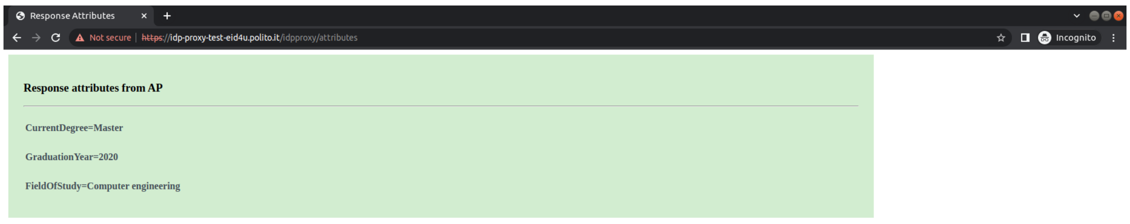
Not secure | ~~https~~://oauth2-server-eid4u.polito.it/authorize?response_type=code&client_id=eidas_client&redirect_uri=https://oauth2-client-eid4u.polito.it/callback&scope=CurrentDegree%20FieldOfStudy%20GraduationYear&state…

**Name : Giobbe Cattaneo**
**Date of birth : 1979-11-15**
**Fiscal Number : TINIT-SLCZGS18A79C924M,**
**Client eidas_client is trying to access the following personal information**

**Do you consent?**

- CurrentDegree
- FieldOfStudy
- GraduationYear

Figure 4.14.  Screen Step 8.


Response Attributes

Not secure | ~~https~~://idp-proxy-test-eid4u.polito.it/idpproxy/attributes

**Response attributes from AP**

**CurrentDegree=Master**

**GraduationYear=2020**

**FieldOfStudy=Computer engineering**

Figure 4.15.  Screen Step 9.

# Chapter 5

# Developer's Manual

## 5.1 Frameworks, Libraries and Environment

The installation of Java JDK8 [88] is the only prerequisite. As a build automation tool, Apache Maven [89] was used. In addition, since Maven is already included in the project, there is no need to install it on the computer (by utilizing maven-wrapper).

Last but not least, installing Docker and Docker-Compose is necessary to dockerize the solutions.

### 5.1.1 Modules

- **idpproxy**: It is based on the Spring Framework [90].

```
/
├── ...
├── idpproxy
│   ├── config
│   │   └── ...
│   └── test
│       ├── ...
│       └── src
│           ├── ...
│           └── idpproxy-eid4u-taxreference
│               ├── ...
│               └── src
│                   └── main
│                       ├── java
│                       │   └── it
│                       │       ├── ...
│                       │       └── telecomitalia
│                       │           └── ficep
│                       │               ├── idpproxy
│                       │               │   └── spit
│                       │               │       ├── controller
│                       │               │       │   └── SpitController.java
│                       │               │       ├── discovery
│                       │               │       │   └── ...
│                       │               │       └── utils
│                       │               │           └── JWT.java
│                       │               └── ...
│                       ├── resources
│                       │   └── META-INF
│                       │       ├── private-key.jwt1.pem
│                       │       └── ...
│                       └── webapp
│                           ├── ...
│                           ├── js
│                           │   ├── ...
│                           │   └── oauth2.js
│                           └── pages
│                               ├── ...
│                               └── oauth2.jsp
└── ...
```

- **oauth2-client**: A Jakarta EE [91] implementation of the OAuth2 Client.

```
/
└── ...
    └── oauth2
        └── oauth2-client
            ├── src
            │   └── main
            │       ├── java
            │       │   └── com
            │       │       └── homework
            │       │           └── oauth2
            │       │               └── client
            │       │                   ├── api
            │       │                   │   └── TokenEndpoint.java
            │       │                   ├── handler
            │       │                   │   ├── AbstractGrantTypeHandler.java
            │       │                   │   ├── AuthorizationCodeGrantTypeHandler.java
            │       │                   │   ├── AuthorizationGrantTypeHandler.java
            │       │                   │   └── JWT.java
            │       │                   ├── model
            │       │                   │   └── ...
            │       │                   ├── security
            │       │                   │   └── ...
            │       │                   ├── AuthorizationCodeServlet.java
            │       │                   ├── CallbackServlet.java
            │       │                   ├── DownstreamCallServlet.java
            │       │                   └── ..
            │       ├── liberty
            │       │   └── ...
            │       ├── resources
            │       │   ├── META-INF
            │       │   │   ├── microprofile-config.properties.template
            │       │   │   ├── persistence.xml
            │       │   │   ├── private-key1.pem
            │       │   │   ├── private-key-jwt2.pem
            │       │   │   └── public-key-jwt1.pem
            │       │   ├── security
            │       │   │   └── keyclient.jks
            │       │   └── data.sql.template
            │       └── webapp
            │           └── WEB-INF
            │               ├── attributes.jsp
            │               ├── eid.jsp
            │               └── index.jsp
            └── ...
```

- **oauth2-authorization-server** : A Jakarta EE implementation of the OAuth2 Authorization Server.

```
/
└── ...
    └── oauth2
        └── oauth2-authorization-server
            └── src
                └── main
                    ├── java
                    │   └── com
                    │       └── homework
                    │           └── oauth2
                    │               └── authorization
                    │                   └── server
                    │                       ├── api
                    │                       │   ├── AuthorizationEndpoint.java
                    │                       │   ├── JWKEndpoint.java
                    │                       │   └── TokenEndpoint.java
                    │                       ├── handler
                    │                       │   ├── AbstractGrantTypeHandler.java
                    │                       │   ├── AuthorizationCodeGrantTypeHandler.java
                    │                       │   ├── AuthorizationGrantTypeHandler.java
                    │                       │   ├── JWT.java
                    │                       │   └── RefreshTokenGrantTypeHandler.java
                    │                       ├── model
                    │                       │   └── ...
                    │                       └── security
                    │                           └── ...
                    ├── liberty
                    │   └── ...
                    ├── resources
                    │   ├── META-INF
                    │   │   ├── microprofile-config.properties
                    │   │   ├── private-key2.pem
                    │   │   ├── public-key1.pem
                    │   │   └── public-key-jwt2.pem
                    │   ├── security
                    │   └── data.sql.template
                    └── webapp
                        ├── WEB-INF
                        ├── authorize.jsp
                        ├── error.jsp
                        ├── login.jsp
                        └── login-error.jsp
```

- **oauth2-resource-server** : A Jakarta EE implementation of the OAuth2 Resource Server.

```
/
├── ...
│   └── oauth2
│       ├── oauth2-resource-server
│       │   ├── src
│       │   │   └── main
│       │   │       ├── java
│       │   │       │   └── com
│       │   │       │       └── homework
│       │   │       │           └── oauth2
│       │   │       │               └── resource
│       │   │       │                   └── server
│       │   │       │                       ├── secure
│       │   │       │                       │   └── ProtectedResource.java
│       │   │       │                       └── Oauth2RecourseServerApplication.java
│       │   │       ├── liberty
│       │   │       ├── resources
│       │   │       │   └── META-INF
│       │   │       │       ├── AP.json
│       │   │       │       ├── microprofile-config.properties
│       │   │       │       └── public-key2.pem
│       │   │       └── webapp
│       │   │           └── ...
│       │   └── ...
│       └── ...
└── ...
```

### 5.1.2 IntelliJ IDEA

A Java-based integrated development environment (IDE) called IntelliJ IDEA [92] is used to create applications in JAR-based languages including Groovy, Kotlin, and Java. JetBrains developed it (formerly known as IntelliJ).

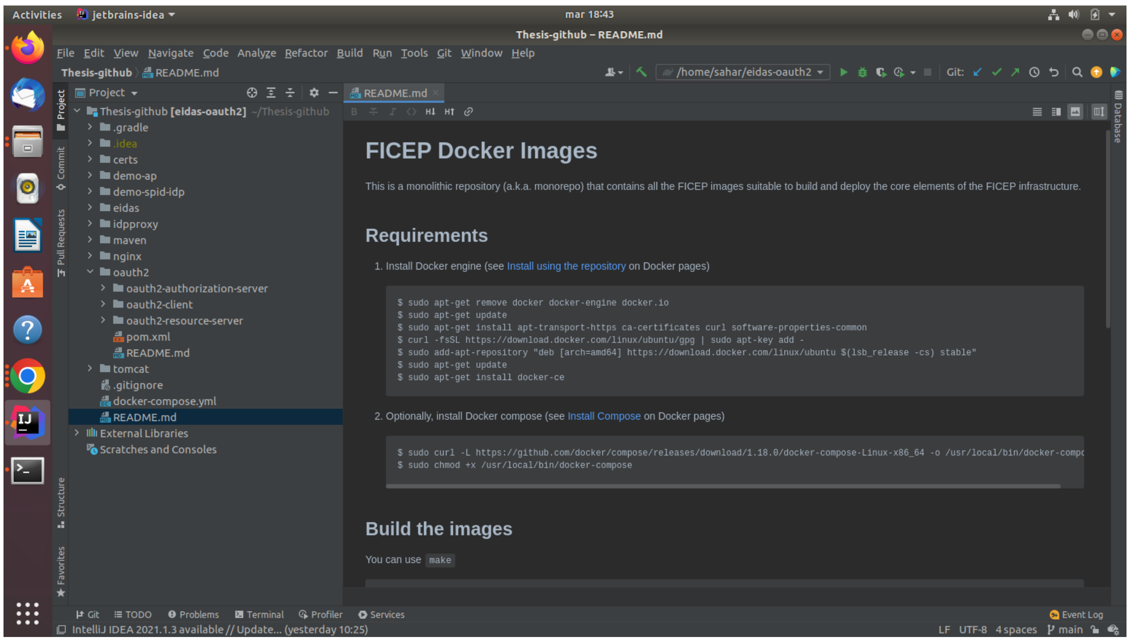
It is a robust IDE with a good user interface that was quite helpful to me when I was developing.

#### Installation

Access the linux version here:

`https://www.jetbrains.com/idea/download/#section=linux`

It is not a free tool and only offers a 30-day free trial, although there are certain exclusive deals that grant students a full year of free use.



FICEP Docker Images

This is a monolithic repository (a.k.a. monorepo) that contains all the FICEP images suitable to build and deploy the core elements of the FICEP infrastructure.

Requirements

1. Install Docker engine (see Install using the repository on Docker pages)

```
$ sudo apt-get remove docker docker-engine docker.io
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
$ sudo apt-get update
$ sudo apt-get install docker-ce
```

2. Optionally, install Docker compose (see Install Compose on Docker pages)

```
$ sudo curl -L https://github.com/docker/compose/releases/download/1.18.0/docker-compose-Linux-x86_64 -o /usr/local/bin/docker-compo
$ sudo chmod +x /usr/local/bin/docker-compose
```

Build the images

You can use `make`

Figure 5.1. IntelliJ IDEA.

# Chapter 6

# Conclusion

In conclusion, the extension of the specific part of the eIDAS architecture with attributes is a significant enhancement that enables organizations to implement robust authorization and access control mechanisms. This expansion empowers organizations to effectively manage user access and privileges within their systems, aligning the eIDAS network with specific industry requirements. By incorporating these attributes, the eIDAS framework becomes even more adaptable, ensuring that authorization mechanisms are tailored to various sectors and use cases.

Authorization rules within Identity and Access Management (IAM) are essential for controlling user access and establishing client privileges in computer systems. Real-world examples, such as house ownership and the boarding process for a plane, provide tangible illustrations of authorization concepts like permissions, privileges, scope, identification, and attributes.

The eIDAS network represents a significant advancement in electronic identification and trust services across the European Union. Its standardized framework aims to facilitate secure and reliable digital transactions while reducing costs and risks associated with additional verification procedures. By leveraging government-issued electronic identities, individuals can authenticate themselves and transfer core personal attributes to service providers (SPs).

Moreover, the eIDAS network allows for the authorization and request of additional attributes, which may be necessary for specific use cases or long-term applications. This flexibility enhances the trust and security of transactions conducted through the network. By incorporating effective authorization mechanisms within the eIDAS framework, SPs can establish a higher level of trust, eliminating the need for separate verification procedures and reducing the collection of additional data.

Connecting this to digital wallets, the eIDAS network plays a vital role in enhancing the security and usability of digital wallet systems. Digital wallets serve as virtual repositories for various forms of digital assets, such as cryptocurrencies, payment credentials, and identity information. By leveraging the eIDAS network's standardized framework for authentication and authorization, digital wallet providers can strengthen the security of their platforms. This includes verifying the identity of users, ensuring proper authorization for accessing wallet contents, and facilitating secure and reliable transactions.

With the integration of the eIDAS network's authorization mechanisms, digital wallets can offer a higher level of trust and security to their users. This empowers individuals to confidently manage their digital assets, knowing that access to their wallet and the transactions they conduct are protected by robust authentication and authorization protocols. Additionally, the standardized framework provided by eIDAS simplifies cross-border digital wallet usage within the EU, fostering seamless and secure digital transactions across member states.

In summary, the extension of the specific part of the eIDAS architecture with attributes provides organizations with the tools to implement effective authorization and access control mechanisms. This enhances the adaptability of the eIDAS network, aligning its authorization mechanisms with specific industry requirements. By incorporating these attributes, organizations can effectively manage user access and privileges within their systems, ultimately contributing to enhanced security and trust in digital transactions.

# Bibliography

[1] D. G. Berbecaru, A. Lioy, and C. Cameroni, “On Enabling Additional Natural Person and Domain-Specific Attributes in the eIDAS Network”, IEEE Access, vol. 9, September 2021, pp. 134096–134121, DOI 10.1109/ACCESS.2021.3115853

[2] “JWT”, https://jwt.io/introduction, Accessed: 06-02-2022

[3] “What is Authorization”, https://auth0.com/intro-to-iam/what-is-authorization/, Accessed: 10-10-2021

[4] “EU: Commission Implementing Regulation (EU) 2015/1501 of 8 September 2015 on the Interoperability Framework Pursuant to Article 12(8) of Regulation (EU) No 910/2014 of the European Parliament and of the Council on Electronic Identification and Trust Services for Electronic Transactions in the Internal Market”, https://eur-lex.europa.eu/legalcontent/EN/TXT/?uri=CELEX%3A02015R1501-20150909, Accessed: 10-05-2021

[5] “Overview of the German Identity Card Project and Lessons Learned”, https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/inspired/eid-in-germany, Accessed: 18-05-2021

[6] “15 years of eID: Portugal’s citizen card”, https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/customer-cases/portugal-id, Accessed: 18-05-2021

[7] “Portugal Lets Citizens Sign Documents With a Smartphone”, https://joinup.ec.europa.eu/collection/joinup/news/digital-mobile-key, Accessed: 18-05-2021

[8] “Infocert”, https://www.infocert.it/, Accessed: 18-05-2021

[9] “Poste Italiane”, https://www.poste.it/, Accessed: 10-05-2021

[10] “Sistema Pubblico di Identità Digitale”, https://www.spid.gov.it/?lang=en-001, Accessed: 10-05-2021

[11] ITU Report, “Digital Identity in the ICT Ecosystem”, https://www.itu.int/dms_pub/itu-d/opb/pref/D-PREF-BB.ID01-2018-PDF-E.pdf, Accessed: 18-05-2021

[12] “eIDAS SAML Attribute Profile Version 1.2”, https://ec.europa.eu/cefdigital/wiki/download/attachments/82773108/eIDAS%20SAML%20Attribute%20Profile%20v1.2%20Final.pdf, Accessed: 10-05-2021

[13] J. Dumortier, “Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on Electronic Identification and Trust Services for Electronic Transactions in the Internal Market and Repealing Directive 1999/93/ec”, SSRN Electronic Journal, 2016

[14] “eIDAS SAML Attribute Profile Version 1.2”, https://ec.europa.eu/cefdigital/wiki/download/attachments/82773108/eIDAS%20SAML%20Message%20Format%20v.1.2%20Final.pdf, Accessed: 10-05-2021

[15] “How Can Identity Matching improve the Experience of Citizens on Online Public Services”, https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/2019/06/27/How+can+Identity+Matching+improve+the+experience+of+citizens+on+online+public+services, Accessed: 18-05-2021

[16] “Commission Staff Working Document Impact Assessment Report Accompanying the Document Proposal for a Regulation of the European Parliament and of the Council amending Regulation (EU) No 910/2014 as Regards Establishing a Framework for a European Digital Identity”, https://op.europa.eu/en/publication-detail/-/publication/6f30628d-c458-11eb-a925-01aa75ed71a1/language-en, Accessed: 18-05-2021

[17] A. Cavoukian, “Privacy by Design—The 7 Foundational Principles”, https://iapp.org/resources/article/privacy-by-design-the-7-foundational-principles/, Accessed: 18-05-2021

[18] S. Stalla-Bourdillon, H. Pearce, and N. Tsakalakis, “The GDPR: A game changer for electronic identification schemes? The case study of Gov.UK Verify”, Computer Law Security Review, vol. 34, August 2018, pp. 784–805, DOI 10.1016/j.clsr.2018.05.012

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

[20] “JavaScript Object Notation”, https://www.json.org/json-en.html, Accessed: 10-05-2021

[21] E. Union, “Commission implementing regulation (eu) 2015/1501 of 8 september 2015 on th interoperability framework: Commission implementing regulation (eu) 2015/1502 of 8 september 2015 on setting out minimu technical specifications and procedures for assurance levels for electronic identification; commission implementing decisio (eu) 2015/1984 of 3 november 2015 defining the circumstances, formats and procedures of notification”, https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32015R1502, Accessed: 05-03-2021

[22] S. Lips, N. Bharosa, and D. Draheim, “eIDAS Implementation Challenges: The Case of Estonia and the Netherlands”, Electronic Governance and Open Society: Challenges in Eurasia, (Springer, Switzerland), January 2021, pp. 75–89, DOI 10.1007/978-3-030-67238-6_6

[23] C. Satchell, G. Shanks, S. Howard, and J. Murphy, “Beyond Security: Implications for the Future of Federated Digital Identity Management Systems”, Proceedings of the 18th Australia Conference on Computer-Human Interaction: Design: Activities, Artefacts and Environments, (New York, NY, USA), 2006, p. 313–316, DOI 10.1145/1228175.1228231

[24] L. Bauer, C. Bravo-Lillo, E. Fragkaki, and W. Melicher, “A Comparison of Users’ Perceptions of and Willingness to Use Google, Facebook, and Google+ Single-Sign-on Functionality”, Proceedings of the 2013 ACM Workshop on Digital Identity Management, (New York, NY, USA), 2013, p. 25–36, DOI 10.1145/2517881.2517886

[25] H. Gomi, “User-centric identity governance across domain boundaries”, Proceedings of the 5th ACM Workshop on Digital Identity Management, (New York, NY, USA), 2009, p. 35–44, DOI 10.1145/1655028.1655038

[26] “MyData Global”, https://mydata.org/, Accessed: 13-07-2021

[27] “Commission Staff Working Document Accompanying the Document Report From the Commission to the European Parliament and the Council on the Evaluation of Regulation (EU) No 910/2014 on Electronic Identification and Trust Services for Electronic Transactions in the Internal Market (eIDAS)”, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A02015R1501-20150909, Accessed: 13-07-2021

[28] N. Taniguchi, K. Chida, O. Shionoiri, and A. Kanai, “DECIDE: A Scheme for Decentralized Identity Escrow”, Proceedings of the 2005 Workshop on Digital Identity Management, (New York, NY, USA), November 2005, p. 37–45, DOI 10.1145/1102486.1102493

[29] A. Bhargav-Spantzel, A. C. Squicciarini, and E. Bertino, “Establishing and protecting digital identity in federation systems”, Journal of Computer Security, vol. 14, June 2006, pp. 269–300, DOI 10.3233/jcs-2006-14303

[30] M. S. Ferdous and R. Poet, “Analysing Attribute Aggregation Models in Federated Identity Management”, Proceedings of the 6th International Conference on Security of Information and Networks, (New York, NY, USA), 2013, p. 181–188, DOI 10.1145/2523514.2526998

[31] A. Jøsang, J. Fabre, B. Hay, J. Dalziel, and S. Pope, “Trust Requirements in Identity Management”, https://crpit.scem.westernsydney.edu.au/confpapers/CRPITV44Josang.pdf, Accessed: 28-09-2021

[32] D. G. Berbecaru, A. Lioy, and C. Cameroni, “Providing digital identity and academic attributes through European eID infrastructures: Results achieved, limitations, and future steps”, Software: Practice and Experience, vol. 49, August 2019, pp. 1643–1662, DOI 10.1002/spe.2738

[33] D. G. Berbecaru and A. Lioy, “On the design, implementation and integration of an Attribute Provider in the Pan-European eID infrastructure”, IEEE Symposium on Computers and Communication (ISCC), (Messina, Italy), June 2016, pp. 1263–1269, DOI 10.1109/iscc.2016.7543910

[34] “What can eID do for You”, https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eID+for+You, Accessed: 10-05-2021

[35] “eIDAS Cryptographic Requirements for the Interoperability Framework, TLS and

SAML, Version 1.0”, https://ec.europa.eu/cefdigital/wiki/download/attachments/82773108/eidas__crypto_requirements_for_the_eidas_interoperability_framework_v1.0.pdf?version=1&modificationDate=1497252920224&api=v2, Accessed: 10-05-2021

[36] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, RFC-8446, August 2018, DOI 10.17487/RFC8446

[37] D. G. Berbecaru, A. Atzeni, M. De Benedictis, and P. Smiraglia, “Towards Stronger Data Security in an eID Management Infrastructure”, 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), (St. Petersburg, Russia), 2017, pp. 391–395, DOI 10.1109/pdp.2017.90

[38] “eIDAS Interoperability Architecture Version 1.2”, https://ec.europa.eu/cefdigital/wiki/download/attachments/82773108/eIDAS%20Interoperability%20Architecture%20v.1.2%20Final.pdf, Accessed: 10-05-2021

[39] “New Notified eID Schemes in 2020”, https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/2020/12/17/New+notified+eID+schemes+in+2020

[40] “Overview of Pre-Notified and Notified eID Schemes Under eIDAS”, https://ec.europa.eu/cefdigital/wiki/display/EIDCOMMUNITY/Overview+of+pre-notified+and+notified+eID+schemes+under+eIDAS, Accessed: 10-05-2021

[41] “Electronic ID cards in Belgium: The keystone of eGovernment”, https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/customer-cases/belgium, Accessed: 10-05-2021

[42] “Itsme”, https://www.itsme.be/en/blog, Accessed: 10-05-2021

[43] “Belgium: share of households with an eID card reader 2008-2015”, https://www.statista.com/statistics/558946/share-of-households-with-an-eid-card-reader-inbelgium/, Accessed: 10-05-2021

[44] “Next Generation NemID”, https://en.digst.dk/digitisation/eid/next-generation-nemid/, Accessed: 10-05-2021

[45] “e-Estonia.Com E-Identity”, https://e-estonia.com/solutions/e-identity/id-card/, Accessed: 10-05-2021

[46] “Estonian Information System Authority Means of eID”, https://e-estonia.com/solutions/eidentity/id-card/, Accessed: 10-05-2021

[47] “Carta d’Identità Elettronica”, https://www.cartaidentita.interno.gov.it/, Accessed: 18-05-2021

[48] “Avanzamento Trasformazione Digitale, SPID”, https://avanzamentodigitale.italia.it/it/progetto/spid, Accessed: 10-05-2021

[49] “Public Broadcasting of Latvia, eID Cards to Become Mandatory Identification Documents in 2023”, https://eng.lsm.lv/article/society/society/eid-cards-to-becomemandatory-identification-documents-in-2023.a290382/, Accessed: 10-05-2021

[50] “LuxTrust, What Exactly is an Electronic Identity (eID)”, https://www.luxtrust.com/whatexactly-is-an-electronic-identity-eid/, Accessed: 10-05-2020

[51] “The Year of the Slovakian eID”, https://silicontrust.org/2014/03/31/2014-the-year-of-the-slovakian-eid/, Accessed: 10-05-2020

[52] “Slovakia Uses eID Card for Safe Digital Public Services”, https://thrive.dxc.technology/eur/2019/05/16/slovakia-uses-eid-card-for-safe-digital-public-services/, Accessed: 10-05-2020

[53] “Spanish ID Cards, Evolution and Meaning of DNI 3.0 Fields”, https://www.mobbeel.com/en/blog/spanish-id-cards-evolution-and-meaning-of-dni-3-0-fields/, Accessed: 10-05-2020

[54] “Digital Government Factsheet 2019-The Netherlands”, https://joinup.ec.europa.eu/sites/default/files/inline-files/Digital_Government_Factsheets_Netherlands_2019_0.pdf, Accessed: 10-05-2020

[55] U. B. Mir, A. K. Kar, Y. K. Dwivedi, M. Gupta, and R. Sharma, “Realizing digital identity in government: Prioritizing design and implementation objectives for Aadhaar in India”, Government Information Quarterly, vol. 37, April 2020, p. 101442, DOI 10.1016/j.giq.2019.101442

[56] N. Tsakalakisz, S. Stalla-Bourdillon, and K. O'Hara, "Identity Assurance in the UK: technical implementation and legal implications under eIDAS", Journal of Web Science, vol. 3, December 2017, pp. 32–46, DOI 10.1561/106.00000010

[57] "eID4U Project", https://ec.europa.eu/inea/en/connecting-europe-facility/cef-telecom/2017-eu-ia-0051, Accessed: 10-05-2021

[58] D. G. Berbecaru and C. Cameroni, "On integration of academic attributes in the eIDAS infrastructure to support cross-border services", 22nd International Conference on System Theory, Control and Computing (ICSTCC), (Sinaia, Romania), 2018, pp. 691–696, DOI 10.1109/ICSTCC.2018.8540674

[59] D. G. Berbecaru, A. Lioy, and C. Cameroni, "Electronic Identification for Universities: Building Cross-Border Services Based on the eIDAS Infrastructure", Information, vol. 10, June 2019, p. 210, DOI 10.3390/info10060210

[60] D. G. Berbecaru, A. Lioy, and C. Cameroni, "Providing login and Wi-Fi access services with the eIDAS network: A practical approach", IEEE Access, vol. 8, 2020, pp. 126186–126200, DOI 10.1109/access.2020.3007998

[61] " FICEP First Italian Crossborder eIDAS Proxy Server", https://www.eid.gov.it/presentazioneprogetto?lang=en-001, Accessed: 28-09-2021

[62] "Agenzia Per l'Italia Digitale", https://www.agid.gov.it/, Accessed: 10-05-2021

[63] "SPID Registry", https://registry.spid.gov.it/, Accessed: 10-05-2021

[64] "SPID Regole Tecniche", https://docs.italia.it/italia/spid/spid-regole-tecniche/it/stabile/attributi.html, Accessed: 10-05-2021

[65] "Apache Struts 2", https://struts.apache.org/, Accessed: 10-05-2021

[66] "eIDAS Node Version 1.4.4 Source Code", https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eIDAS-Node+version+1.4.4, Accessed: 10-05-2021

[67] "eIDAS-Node Installation Manual v1.4.4", https://ec.europa.eu/cefdigital/wiki/download/attachments/84421967/eIDAS-Node%20Installation%20Manual%20v1.4.4.pdf, Accessed: 10-05-2021

[68] D. G. Berbecaru and C. Cameroni, "ATEMA: An attribute enablement module for attribute retrieval and transfer through the eIDAS network", 24th International Conference on System Theory, Control and Computing (ICSTCC), (Sinaia, Romania), October 2020, pp. 532–539, DOI 10.1109/icstcc50638.2020.9259642

[69] "NGINX Reverse Proxy", https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/, Accessed: 28-09-2021

[70] A. Rodriguez, "Restful Web Services: The Basics. IBM developerWorks", https://cs.calvin.edu/courses/cs/262/kvlinden/references/rodriguez-restfulWS.pdf, Accessed: 10-05-2021

[71] "NGINX SSL Termination", https://docs.nginx.com/nginx/admin-guide/security-controls/terminatingssl-http/, Accessed: 10-05-2021

[72] "What is OAuth 2.0", https://auth0.com/intro-to-iam/what-is-oauth-2/, Accessed: 10-10-2021

[73] M. B. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)", RFC-7519, May 2015, DOI 10.17487/RFC7519

[74] "Introduction to Redis", https://redis.io/docs/about/, Accessed: 09-05-2021

[75] "Jakarta EE", https://jakarta.ee/, Accessed: 10-05-2021

[76] "Jakarta EE Security API", https://www.baeldung.com/java-ee-8-security, Accessed: 10-10-2021

[77] "Nimbus JOSE + JWT Library", https://connect2id.com/products/nimbus-jose-jwt, Accessed: 10-05-2021

[78] "Introduction to Java Servlets", https://www.geeksforgeeks.org/introduction-java-servlets/, Accessed: 14-02-2022

[79] "JSON Web Token with Java", https://github.com/oktadev/okta-java-jwt-example, Accessed: 14-02-2022

[80] "The OAuth 2.0 Authorization Framework Using Jakarta EE", https://www.baeldung.com/java-ee-oauth2-implementation, Accessed: 14-02-2022

[81] "JAX-RS", https://restfulapi.net/create-rest-apis-with-jax-rs/, Accessed: 14-02-2022

[82] “Java EE Web API”, https://search.maven.org/search?q=g:javax%20AND%20a:javaee-web-api&core=gav/, Accessed: 10-10-2021
[83] “MicroProfile Config”, https://search.maven.org/search?q=g:org.eclipse.microprofile.config%20AND%20a:microprofile-config-api&core=gav, Accessed: 10-10-2021
[84] “MicroProfile JWT”, https://search.maven.org/search?q=g:org.eclipse.microprofile.config%20AND%20a:microprofile-config-api&core=gav, Accessed: 10-10-2021
[85] “Docker”, https://docs.docker.com/engine/install/ubuntu/, Accessed: 09-05-2021
[86] “Open Liberty Project”, https://openliberty.io/, Accessed: 10-05-2021
[87] “Apache Virtual Host”, https://httpd.apache.org/docs/2.4/vhosts/, Accessed: 10-10-2021
[88] “JDK”, https://www.oracle.com/it/java/technologies/javase/javase8-archive-downloads.html, Accessed: 14-02-2022
[89] “Apache Maven”, https://maven.apache.org/, Accessed: 14-02-2022
[90] “Spring Framework”, https://spring.io/, Accessed: 14-02-2022
[91] “JAKARTA EE”, https://jakarta.ee/, Accessed: 14-02-2022
[92] “IntelliJ IDEA”, https://www.jetbrains.com/idea/, Accessed: 13-11-2021