

**POLITECNICO DI TORINO**

**MASTER's Degree in COMPUTER ENGINEERING**



**MASTER's Degree Thesis**

**LOW-CODE APPROACH FOR  
WEB-BASED  
ACCESS MANAGEMENT**

**Supervisor**

**Prof. Fulvio VALENZA**

**Company Advisor**

**Ing. Pietro SANTORO**

**Candidate**

**MATTEO CARBONE**

**Academic Year 2021/2022**



# Summary

For years, Identity and Access Management has been considered a monster to tame: every little change to the security policy turned into a huge impact to Front End of all integrated applications. Traditional software development is the result of a long, painstaking and detailed effort. Developers write individual lines of code representing instructions and data. They organise that code into functional routines and modules that provide the features and functionality of the software. This approach requires detailed knowledge of aspects across the application development spectrum: development languages, integrated development environments and compilers, testing and distribution tools, and the various policies and practices used to approach coding, testing and distribution

The goal of the thesis is to develop new access management experiences using this brand low-code new approach and to study its effectiveness. This was done by using orchestration software, PingOne DaVinci, which allows IAM experts to use a drag-and-drop interface in order to design smooth user experiences and secure business logic without needing developer resources. Low code development is a new approach to programming, which allows applications (even enterprise-level applications) to be created in a relatively short time. It is a method that limits the need to write code to a minimum, eliminating it altogether in some cases. Low-code development employs specific platforms (there are now several on the market), which allow programmers to create applications with a visual and logical approach, via the interface of these platforms. This means that the developers' task is no longer to write proprietary code, but their focus becomes describing the basic principles and functions of the application, leaving it to the platform itself to translate this into code.



# Table of Contents

<b>List of Figures</b>	VI
<b>1 Introduction</b>	1
1.1 Thesis Description . . . . .	3
<b>2 Key Concepts</b>	5
2.1 Http Protocol . . . . .	5
2.1.1 Http Request . . . . .	6
2.1.2 Http Response . . . . .	8
2.2 API . . . . .	9
2.2.1 Access Manager . . . . .	13
<b>3 Access Management Processes</b>	20
3.1 Authentication . . . . .	20
3.1.1 Basic Authentication . . . . .	21
3.1.2 Username and Password Authentication . . . . .	22
3.1.3 Identity Federation . . . . .	24
3.2 Authorization . . . . .	37
3.2.1 RBAC and ABAC . . . . .	37
3.2.2 Session Cookie . . . . .	39
3.2.3 OAuth 2.0 . . . . .	39
<b>4 Thesis Objectives</b>	48
<b>5 Low-Code Approach</b>	51
5.1 PingOne DaVinci . . . . .	53
5.1.1 PingOne DaVinci Components . . . . .	54
<b>6 Access Management flows with orchestrator</b>	58
6.1 Registration Process . . . . .	58
6.2 Authentication Process . . . . .	62

6.3	Authorization Process . . . . .	67
<b>7</b>	<b>Use Cases</b>	<b>69</b>
7.1	Integration in a Single Page Application . . . . .	69
7.2	Integration with Redirect Method . . . . .	74
<b>8</b>	<b>Conclusions</b>	<b>79</b>
	<b>Bibliography</b>	<b>82</b>

# List of Figures

2.1	A simple representation of message exchanged between client and server in HTTP . . . . .	5
2.2	Example of an Http request . . . . .	8
2.3	Example of an Http request . . . . .	9
2.4	How several entities interact with an API . . . . .	10
2.5	Classification of Channel APIs . . . . .	11
2.6	Channel User APIs . . . . .	12
2.7	Example of a simple API . . . . .	13
2.8	Possible Architecture for an access manager software . . . . .	15
2.9	How access manager works when a web resource is requested . . . . .	16
2.10	Example of B2C model interaction . . . . .	17
2.11	Example of B2B model interaction . . . . .	18
2.12	Example of B2E model interaction . . . . .	18
3.1	How client interact when using Password authN . . . . .	23
3.2	SAML Architecture . . . . .	25
3.3	Assertion with Authentication Statement . . . . .	27
3.4	SP-Initiated SSO: Redirect/POST Bindings . . . . .	29
3.5	Fig 18-SP-Initiated SSO: POST/Artifact Bindings . . . . .	31
3.6	Example of JWT . . . . .	33
3.7	OIDC-Authorization Code Flow . . . . .	35
3.8	OIDC-Implicit Flow . . . . .	36
3.9	RBAC and ABAC policies . . . . .	38
3.10	Session Cookies . . . . .	39
3.11	Client Credential flow . . . . .	41
3.12	ROPC flow . . . . .	42
3.13	Authorization Code Flow . . . . .	44
3.14	Implicit Grant Flow . . . . .	45
3.15	Authorization Code Grant Flow with PKCE . . . . .	46
3.16	How to choose an Oauth flow . . . . .	46

5.1	PingOne Cloud Platform . . . . .	53
5.2	Example of a DaVinci flow. . . . .	55
5.3	Output of successful example flow . . . . .	55
5.4	Example flow error output . . . . .	56
5.5	Multiple Paths in a flow . . . . .	56
5.6	Http connector configuration . . . . .	57
5.7	Functions Connector Configuration . . . . .	57
6.1	First part of registration flow . . . . .	59
6.2	Custom Http form for credentials . . . . .	59
6.3	Google connector configuration . . . . .	61
6.4	Second part of the registration flow . . . . .	61
6.5	Example of screen connector output . . . . .	62
6.6	First part of login flow . . . . .	63
6.7	Second Part of login flow . . . . .	63
6.8	Login Form . . . . .	64
6.9	Duo security example . . . . .	65
6.10	Example of Recovery code form . . . . .	66
6.11	Authorization flow with PingOne Authorize . . . . .	68
7.1	Screen of Single Page Application on Glitch . . . . .	70
7.2	How to create an App on DaVinci . . . . .	71
7.3	Attaching a flow in an application . . . . .	71
7.4	Script for integrate a flow in an SPA . . . . .	72
7.5	Input schema for a DaVinci flow . . . . .	73
7.6	How to perform A/B testing . . . . .	74
7.7	Configuration of an external Idp in PingOne SSO . . . . .	75
7.8	AuthN Policy . . . . .	75
7.9	PingOne SSo SP and DaVinci as Idp . . . . .	76
7.10	Oidc call to SP in HTTP message . . . . .	76
7.11	Oidc- Call to Sp with AuthZ code . . . . .	77



# Chapter 1

## Introduction

Over the past few years, the advancement and increasing spread of cloud, IoT and mobile technology have made corporate strategies insecure in an attempt to resist increasingly sophisticated and targeted cyber attacks. In today's hyper-connected society, the fundamental security principle of a company is no longer where data and resources are located, but how a given user accesses these resources. Business leaders and IT departments are under increased regulatory and organizational pressure to protect access to corporate resources. As a result, they can no longer rely on manual and error-prone processes to assign and track user privileges. For these reasons, comprehensive attention to digital identity management is required: that is why a proper Identity and Access Management (IAM) policy is appropriate.[1]

Identity and access management is a digital identity and access management system that allows those entitled to it to access certain resources of an organization. By simplifying access to these resources, the overall level of security is increased and the cost of managing all users is reduced. Thanks to this system, each person who is identified within an organization is assigned a digital identity characterized by a number of appropriately valued attributes. Organization is able to have better control over the authentication, authorization and auditing processes, and thus privacy rules can be enforced, security can be monitored and reports can be made. Thus, the IAM not only effectively enables a logical perimeter that allows for a real and profound digital transformation, but also guarantees the protection of corporate resources and data. It must also be considered that the spread of increasingly powerful computers allows hackers to break even the most complex passwords, rendering them completely inadequate. This is why biometrics and behavioral authentication techniques are gradually supplanting passwords in certain areas such as high-risk transactions. The IAM frameworks thus attempt to answer two fundamental questions: "who has access to what resource?" and "how to strengthen access policies?".

These systems prevent an attacker from gaining access to policies and resources even if he were able to break the password of a particular user. Furthermore, an IAM system also allows for a good separation of roles within the corporation by giving access privileges to resources according to the position held by a given user within the corporation and by having that person access only what he or she actually has privileges to do. An IAM system also provides valuable information on how employees and customers interacted with applications and thus how they accessed them. Such information is not only useful from a security point of view, but also to compile interaction patterns, analysing both employee work and customer behaviour.[2]

Identity and access management is a broad topic. It is divided into macro-categories that are the 3 technologies at the core of IAM:

- User Lifecycle Management(ULM) Access Governance(AG):
  - User Lifecycle Management means handling of a user from the moment in which he accepts the job and until he leaves the company. It is referred to which resources he can access during job.
  - Access governance (AG) is an aspect of information technology (IT) security management that seeks to reduce the risks associated with end users who have unnecessary access privileges. The need for access governance has grown in significance as organizations seek to comply with regulatory compliance mandates and manage risk in a more a strategic manner. An important goal of access governance is to reduce the cost and effort that's involved in overseeing and enforcing access policies and management procedures, including recertification.
- Access Management and Federation: An access management federation (or federation, for short) provides a trust framework in which identity providers (such as library organizations) and service providers (such as publishers) agree to policies for the sharing of encrypted user information to provide easy access to online content. When a user attempts to access a resource, it is the user's home organization that authenticates him or her by vouching for their identity. All personal information and credentials remain by default with their home organization, preserving user privacy. Using one username and password to access resources across different platforms, applications and locations is called single sign-on (SSO). It has significantly reduced the administrative burden on institutions and removed barriers to access. Members of a federation share metadata files, establishing trusted connections between libraries and

publishers. This shared, trusted network of metadata means configuring single sign-on to a resource is more efficient and doesn't require ongoing IT assistance.

- Privileged Access Management(PAM): Organizations implement privileged access management (PAM) strategies to protect against threats such as credential theft or illegitimate use of privileges. The acronym PAM refers to a comprehensive information security strategy that includes people, processes, and technologies and involves the control, monitoring, protection, and verification of all privileged identities, human or non-human, that collectively exist in an enterprise IT environment. PAM programs are based on the principle of least privilege, whereby each user is assigned the minimum level of access required to perform his or her duties.

Being very broad as a topic the significant part of identity and access management that is addressed in this thesis work is that of access management Federation. Here a brief description of the following chapters. In the first chapters (chapter 2 through 5), topics that underlie access management are presented. In the sixth and seventh, the various AM flows are analyzed first in a standard way and then with the use of the orchestrator. In the eight I go on to analyze a real-life case of application.

## 1.1 Thesis Description

- Chapter 2: Presents some key concepts that are necessary to understand access management and how interactions between the entities involved occur.
- Chapter 3: Goes on to explain what are the two main processes of an access management software. First I talk about Authentication process, including different techniques and protocols underlying this mechanism. Then we move on to the concept of authorization and how to implement it.
- Chapter 4: In this chapter, after a background on the access management world, are explained the objectives of the thesis.
- Chapter 5: This chapter introduces the low-code approach and its main advantages. After that, the PingOne DaVinci platform, its basic components and features are introduced.
- Chapter 6: In this chapter the access management flows that were created on the DaVinci platform and how they were constructed are explained. In particular, the registration, authentication, and authorization flows will be analyzed.

- Chapter 7: Two use cases of flow integration will be seen in this chapter. In a first case, how to integrate a flow developed on PingOne DaVinci within a Single Page Application will be analyzed. In a second case it will be seen how to integrate a flow within an existing application that provides access to the PingOne testing environment.
- Chapter 8:

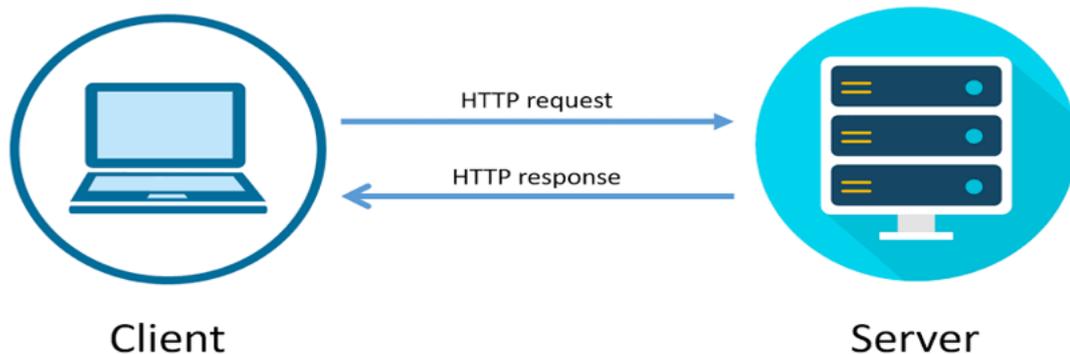
# Chapter 2

## Key Concepts

This chapter will expose key concepts such as the http protocol to understand how various entities interact in a web access management context and also the concept of Api security to understand how access to resources occurs.

### 2.1 Http Protocol

The Hypertext Transfer Protocol (HTTP) is a protocol of the last layer of the ISO-OSI stack and was designed to be implemented above the lower layer protocols. It describes the Web interactions that occur between a client and a server. The client is the one who initiates the communication and then sends a request to the server, which, later sends a response message. Because it is a request-response



**Figure 2.1:** A simple representation of message exchanged between client and server in HTTP

protocol, HTTP is a way for a client to access resources ( such as can be HTML files) via the Web that are precisely hosted on a server. Clients usually use at the

transport layer a TCP connection to communicate with the server.

Each communication between client and server begins with a request, a text message created by the client in a specific format, that of HTTP.

### 2.1.1 Http request

A request is composed by following parts:

- Uniform Resource Identifiers (URIs) are created to access physical or abstract resources on the Internet, which can be of various types depending on the situation: websites, email senders or recipients. A uri can be composed at most of five parts, but only two are mandatory. These parts are:
  - Scheme :give informations on which protocol is used
  - Authority: identifies the domain
  - Path: shows the path of the resource
  - Query: represent the request
  - Fragment: specify a partial aspect of a certain resource

Only schema and path must be mandatory in an identifier. In the URI syntax, all components are listed one after the other and separated by specific, defined characters.

Here there is an example:

`https://example.org/test/test1?search=test-questionpart2`

In this case:

- Scheme: `https`
- Authority: `example.org`
- Path: `test/test1`
- Query: `search=test-question`
- Fragment: `part2`

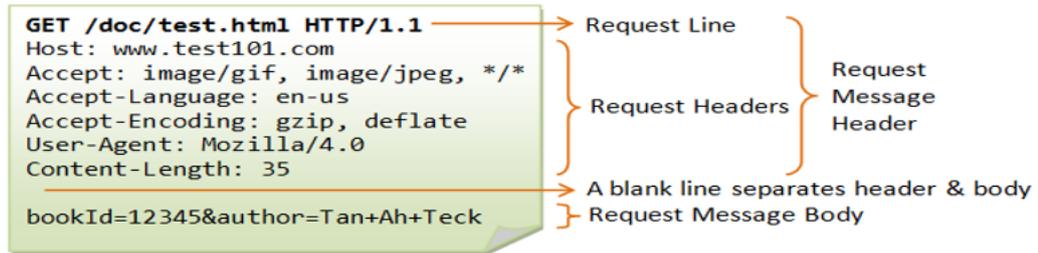
So this means that the part in question (`part2`) is accessible via HTTP, is located on a device with the identifier `example.org`, and, wanting to do a search, can be found under the specified path.

- Http Method: An HTTP method, sometimes referred to as an HTTP verb, indicates the action that the HTTP request expects from the queried server. The main methods are:

- GET : The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
- POST: A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms. The POST method is most often utilized to create new resources.
- HEAD: Same as GET, but transfers the status line and header section only.
- PUT: Replaces all current representations of the target resource with the uploaded content.
- DELETE: Removes all current representations of the target resource given by a URI.

The most commonly used methods are GET and POST.

- An HTTP request contains other information called request headers. The headers provide much additional information such as the host where the resource is located (Host), the response formats accepted by the client (Accept), and the application used by the client to make the request (User-Agent). Some important headers parameters are:
  - Accept:Format file that are accepted by the client.
  - Authorization: Identifies the permission level for the browser.
  - Content-Length: Length of the body request.
  - Date and time when the message was generated.
  - Host: Specifies the server where the requested resource is hosted.
  - Referer: Address of the previous web page from which a link to the currently requested page was followed.
  - User-Agent: Specifies which application (browser) is making the request.
- The request body part is optional for an HTTP message but if it is available then it is used to carry the entity-body.  
Here there is an example of HTTP request:



**Figure 2.2:** Example of an Http request

## 2.1.2 Http Response

A response message is similar to the request but not totally equal and it's sent by the server. The aim of the response is to provide the client with the resource it requested, or inform the client that the action it requested has been carried out; or else to inform the client that an error occurred in processing its request. An HTTP response is composed by following parts:

- Status code: The most important parameter of the response. Specifies which is the status of the response, and how the server had reacted to the previous corresponding request. Possible codes for this field are:
  - 1xx Informational
  - 2xx Success
  - 3xx Redirection
  - 4xx Client Error
  - 5xx Server Error

The “xx” refers to different numbers between 00 and 99. Status codes starting with the number ‘2’ indicate a success. For example, after a client requests a web page, the most commonly seen responses have a status code of ‘200 OK’, indicating that the request was properly completed.

If the response starts with a ‘4’ or a ‘5’ that means there was an error and the webpage will not be displayed. A status code that begins with a ‘4’ indicates a client-side error (It's very common to encounter a ‘404 NOT FOUND’ status code when making a typo in a URL). A status code beginning in ‘5’ means something went wrong on the server side. Status codes can also begin with a ‘1’ or a ‘3’, which indicate an informational response and a redirect, respectively.

- Also here we can find headers that are helpful information for the client. Some possible headers for a response are:

- Date: The date and time that the message was sent.
  - Expires: Gives the date/time after which the response is considered stale.
  - Set-cookie: A cookie is set in this field when the server wants to send to the client, either by setting or updating it.
  - Location: Used in redirection(status code 3xx), or when a new resource has been created. The Location response header indicates the URL to redirect a page to.
- o The response body contains, if there is no error, the requested resource by the client. Here there is an example of http response.

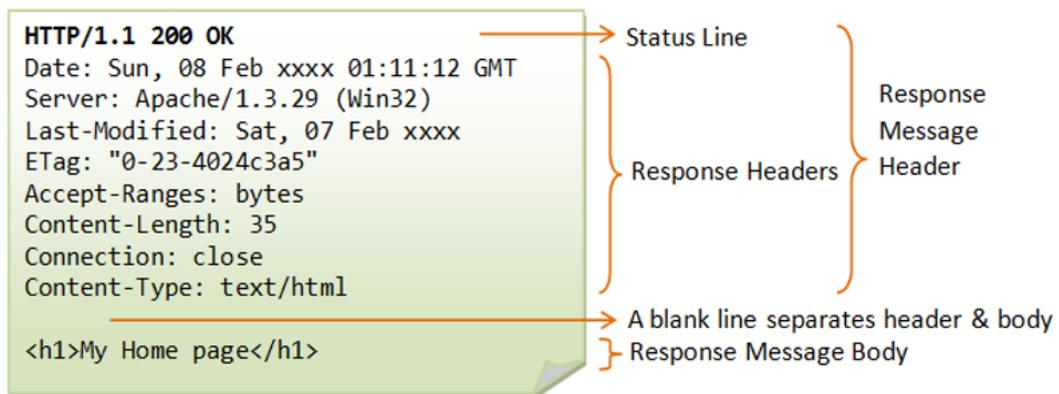


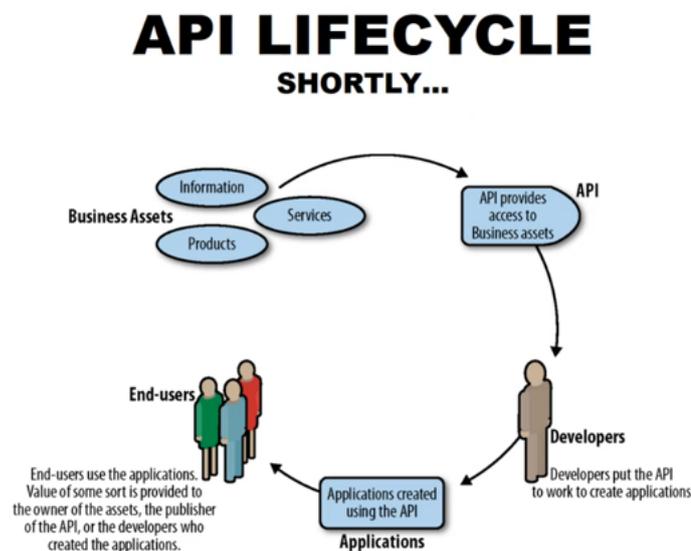
Figure 2.3: Example of an Http request

To have a full knowledge of request and response message it's possible to see this article. [3]

## 2.2 API

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. APIs allow products and services to communicate in a particular way with each other, without the need for them to know how the other party is implemented internally. This goes a long way toward simplifying the work for both time and money savings. When creating new tools and products or managing existing ones, APIs offer flexibility, simplify design, administration and use, and provide opportunities for innovation. Sometimes APIs can be thought of as a form of contract in that before communication between 2 parties occurs these describe how the 2 parties should communicate

with each other and thus represent a form of agreement between 2 parties: if Party A sends a remote request structured in a certain way, Party B's software will respond in another determined way. By simplifying the integration of new application components into an existing architecture, APIs promote collaboration between business and IT teams. Cloud-native application development, based on linking a microservices application architecture through APIs, enables accelerated development speed [4]. So thanks to the API a company is able to connect the infrastructure thanks to cloud-based apps, and also share data with partners or other external users. The following figure shows how APIs can be used by an end-user and also what is their lifecycle.



**Figure 2.4:** How several entities interact with an API

As you can see an API goes to make available to an end-user (or even a partner of a given company) to access a resource. The resource can be about a given product or service, this depends on the core business of the company in question. The developer creates an API that allows access to a resource, places this API within an application that will then be used by a user. In this way the end user does not have direct access to the resource in question but does so through an intermediary mechanism, namely an API. There exist many kinds of APIs and so it's possible to do a categorization of them. First, however, a distinction should be made between client and user in the Web context.

The client or Consumer represents the technical actor, which can be a client

application( browser, a mobile app) or a server application. This is usually authenticated with a client-id ( in technical jargon also api-key) and possibly a client-secret( in technical jargon also api-secret). This must always be recognized.

The user, on the other hand, is precisely the human actor requesting a certain resource at a certain instant. Typically it is authenticated with a username and password. Authentication by the user is not always required since the API does not necessarily carry sensitive information. Now we can categorize APIs from a channel point of view and from a user point of view. With channel APIs you want to make a differentiation from the point of view of the consumer or client accessing a certain resource, a certain api. That is, to what kind of application a certain business company is exposing one of its APIs and what kind of association there is between the one exposing the api and the one who wants to exploit it. Channel APIs are divided into Open API, Partner API and Internal API. Open APIs are available to everyone; they are those that can be accessed simply by browsing Internet. The second ones are those that are exposed to a particular partner of the company as a result of an agreement between the 2 entities. Internal APIs refer to APIs that can be accessed if you are an internal employee of the company. Below is a schematic showing the characteristics for each individual group of API channel.

	Available for anyone to register?	Agreement between parties?	Dedicated supports?	Pay for usage?
Open API	Yes	No	Yes	Depends on the business strategy
Partner API	No	Yes	Depends on the partnership agreements	Depends on the partnership agreements
Internal API	No, internal consumer only	No	Depends on API business value	No

**Figure 2.5:** Classification of Channel APIs

While when we talk about User APIs we refer to what kind of resources can be accessed. User APIs are differentiated into Public, Private and technical APIs. The former are those that are available to anyone and therefore do not carry any sensitive information or information about any particular user. Private APIs, in contrast, have access to private resources and for that as can also be seen in the diagram below also require user authentication. Finally, there are technical APIs, which do not expose public domain information but at the same time do not require

user authentication. To give an example of a technical API one can think of a company, which makes quotes, and which applies a certain discount on a certain quote only to a single partner. Since it does not want to let other partners know about this discount it does not want this information to be public. Below is an outline, not stringent as then each company and each developer can decide who and what can access one of its resources, that allows for an overview between Channel APIs and user APIs.

	Consumer		
	OPEN	PARTNER	INTERNAL
Public APIs	<i>Consumer authentication only</i>	<i>Consumer authentication only</i>	<i>Consumer authentication only</i>
Private APIs	<i>Consumer and user authentication</i>	<i>Consumer and user authentication</i>	<i>Consumer and user authentication</i>
Technical APIs	<i>There's no such thing!</i>	<i>Consumer STRONG authentication</i>	<i>Consumer STRONG authentication</i>

**Figure 2.6:** Channel User APIs

Another type of APIs that we use whenever we want to access a resource on the Internet are web APIs. Web APIs, as the name suggests, can be accessed using the HTTP protocol. It is a framework that helps you to create and develop HTTP based RESTFUL services. Web API is used in either a web server or a web browser. Basically Web API is a web development concept. It is limited to Web Application's client-side and also it does not include a web server or web browser details. If an application is to be used on a distributed system and to provide services on different devices like laptops, mobiles, etc then web API services are used. Web API is the enhanced form of the web application.

A very important concept that needs to be discussed is how to protect APIs and their security. Through APIs, companies transfer data and offer services. Damaged, exposed or cyber-attacked APIs are the cause of serious data breaches that put sensitive data such as medical, financial and personal information in the public domain. However, it must also be remembered that not all data that is transferred over the Internet is of equal value and therefore does not all require the

same level of protection [5]. API security is implemented through 3 protocols that will be addressed in later chapters: OAuth, SAML, OpenId Connect. Generally speaking, it can be said that we need to be careful about some fundamental points when it comes to API security:

- Token Implementation: you need to define trusted identities and allow them access to the API through the use of tokens(this concept will also be explained more fully in later chapters)
- Cryptography adoption: You need to implement digitally sign and possibly encrypt messages. This can be done through the use of the TLS protocol.
- Identifies vulnerabilities: Keep your operating system, your network, your drivers, and your APIs constantly monitored. Check the operation of all components and identify any vulnerabilities that could be exploited to access your APIs.
- It's preferred to use a gateway API. They are the main coordination point for APIs. Allows control and analysis of requests and how APIs are used.

Here is possible to see an example of API. Let's imagine we have an application where at some point the client wants to retrieve the list of exams from the server. This can be an API that is called to obtain the list of exams.

```
async function getAllExams() {
  // call: GET /api/exams
  const response = await fetch(BASEURL + '/exams');
  const examsJson = await response.json();
  if (response.ok) {
    return examsJson.map((ex) => Exam.from(ex));
  } else {
    throw examsJson; // an object with the error coming from the server
  }
}
```

**Figure 2.7:** Example of a simple API

### 2.2.1 Access Manager

The access manager is a solution in access management that provides secure access to web-based applications, SaaS services, and business-to-business federation

interactions. The access manager, also provides, authorized, context-aware, and secure access to Internet applications from anywhere, anytime. Access Manager uses industry standards, such as SAML, OAuth, OpenID Connect, and Federation to deliver federated single sign-on and supports multi-factor authentication, role-based access control, and data encryption. Then are listed possible use cases of Access Manager:

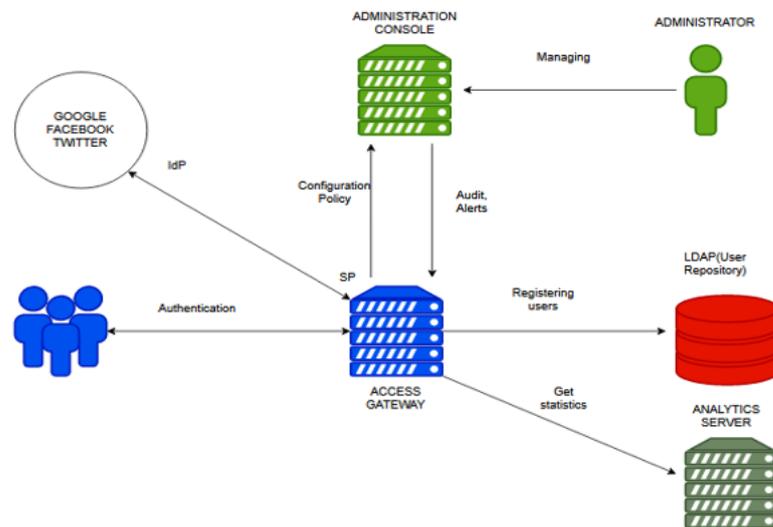
- **Secure Web Access Management:** Allows organizations to regulate their users' access based on authentication context, role-based policies, multifactor access, and more granular access control.
- **Effective Partner Collaboration:** Provides secure access to companies that are in partnership. This allows access not to all corporate resources but only to those applications for which it has been determined that they can be accessed.
- **Simple and Secure Consumer Access Management:** Delivers robust access management including self-service on-boarding and SSO for your customers. It enables your customers to sign-up and set up their own accounts using social identities, such as Facebook, Google, Twitter, and LinkedIn.

To better understand these possible scenarios, it is good to understand what some terms regarding access management mean. This chapter then also illustrates possible solutions from both an architectural and a functional appoint of view.

- **Single Sign On (SSO):** It is a technical solution, which can be implemented in a variety of ways, which allows a user to authenticate to several applications without the need to remember and/or enter multiple credentials for each application they want to access. One application is accessed and then through the implementation of a protocol the others can be accessed as well. This has the advantage not only from a time perspective but also the fact that one does not have to remember as many credentials as there are applications to which one wants to log in. You could store the credentials somewhere but this would then carry the risk that someone could steal them.
- **Identity Federation:** In times where increasingly everything is interconnected, companies need to share data about resources and users. This can be done through the concept of identity federation, which allows a user to authenticate on one system and pass the authentication context to another system, from another corporate. We are not only talking about authentication, but as will be seen in the next chapters, delegated authorization can also be implemented through this concept. In this way both partners in the agreement can share common information regarding a user.

Although similar, the 2 concepts listed above are not quite the same . In that one does not imply the other and vice versa. When SSO implies Identity Federation it is just the case that maybe the 2 applications involved in the SSO procedure belong to different web domains. Even in the opposite case we should not think that identity federation implies mandatory SSO, as we can also use it to delegate authentication or accounts to a third party system, thus without doing SSO.

Here it's possible to see an example of architecture of access management to better understand what are the main components. These concepts were taken from the following article.[6]



**Figure 2.8:** Possible Architecture for an access manager software

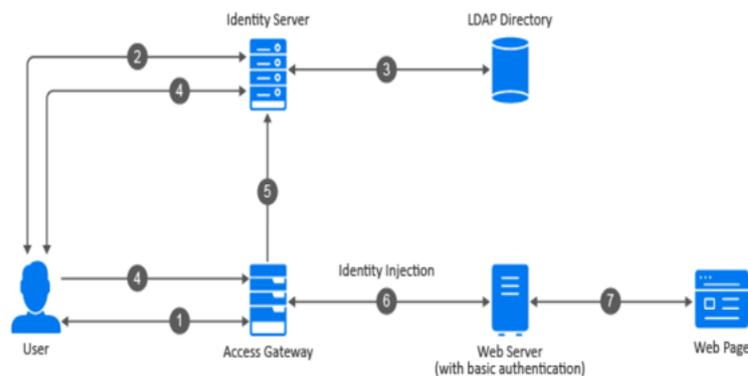
It's possible to see several components:

- **Administration console:** provides a unified console for configuring and managing all components of Access Manager. Some of its tasks can be Resource Management( such as policies and certificates), Health and Statistics monitoring, Persistent configuration store.
- **Access Gateway:** It can be seen as the point of contact between the Internet and access management software. It provides security services (authorization, single sign-on, and data encryption) integrated with the identity and policy services of Access Manager. Some of its key features are: SSO to protected web services (it assumes the role of Identity Provider in a Federation with an external company); Authorization to authenticated users; Multi-homing that enables to use a single public IP address to protect multiple types of web

resources; Caching functions in order to avoid that user must always be sent to the web server.

- Analytics server: Analytics Server analyses usage, performance, and events of Access Manager. It captures, filters, and analyzes the events that are generated by Access Gateway and Identity Server.
- LDAP: It is the database containing the users that are registered by the access gateway. It is very important and must be kept secure. Accesses to it must be controlled.

This can be seen as a general access management architecture. Then in practice each access manager implements its own model from an architectural point of view. The example shown below is based on the architecture used by Netiq access manager in which it can be seen that the access gateway and the identity server are perceived as 2 separate entities . This is a possible flow of protecting a web resource to understand how access management works:



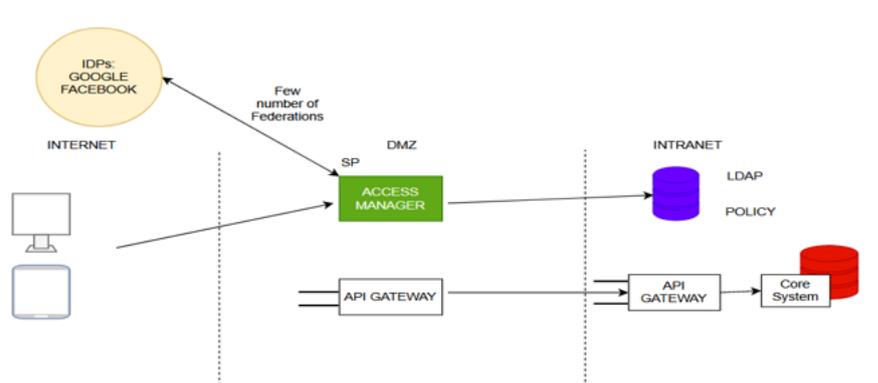
**Figure 2.9:** How access manager works when a web resource is requested

1. A user wants to access to some resource so sends a request to the access gateway.
2. Access gateway redirects the user to the identity server, which asks to the user for its credentials.
3. After the user inserts its credentials, the identity server verify them by asking to the LDAP directory.
4. Identity Server returns an authentication artifact( a pointer in which is specified where to take the information on the Identity Server) to the access gateway through the browser in a query string.

5. Access Gateway retrieves the user's credentials from Identity Server through a direct channel between them.
6. Access Gateway injects the basic authentication information into the HTTP header.
7. The web server validates the authentication information and returns the requested web page.

In addition to the use cases for which an access manager can be used, it is also good to make a distinction about the possible interaction patterns that an access manager must handle in relation to the type of user who wants to access a company's resources. Distinctions can be made between B2B, B2C and B2E.

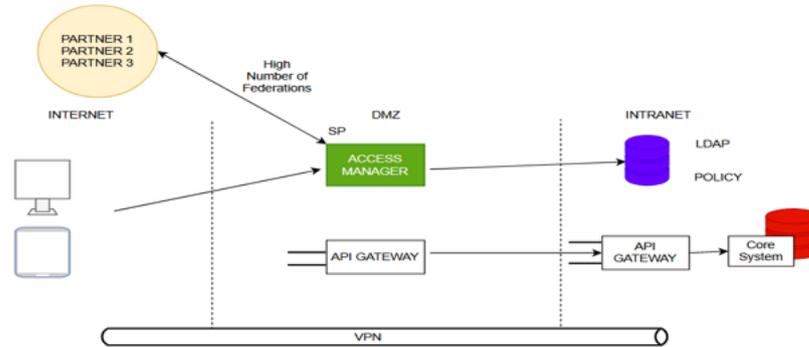
- B2C: When we talk about B2C (Business to customer) we mean an interaction between customer and business i.e., when a customer of a company wants to access its resources.



**Figure 2.10:** Example of B2C model interaction

What can be seen from this image is that a user (who wants to log in to a particular company to obtain a particular resource) can either log in by going and giving his credentials (which will then be verified by the access manager on the LDAP) or he can do so through social login. The latter option involves going in with a Facebook or Google account or some other type of Identity Provider. Obviously to achieve this requires that the access manager of the business unit be in federation with Facebook or Google and then that some form of "agreement" between these companies has already been pre-established. In such a context what must be put a foreground is yes security so that the user has access securely, but user-experience must also prevail as the user may decide if not to become a customer of another company.

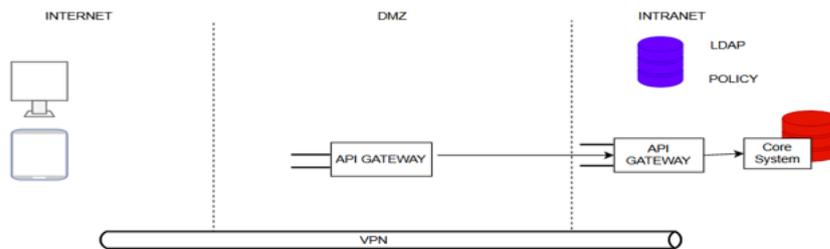
- When we talk about B2B(Business to Business), we mean communication between a company and its business partners.



**Figure 2.11:** Example of B2B model interaction

A first thing that stands out here is the fact that the number of federations that the access manager has to establish is greater than in the previous case. Also, in this context, user-experience is no longer so crucial since one does not have to attract a specific user customer in this case. Note also that the core business part is more or less the same in that what changes is the way the access manager has to interact with outside the company.

- B2E(Business to employee) refers to the way in which an employee of the company has to access a resource within the company.



**Figure 2.12:** Example of B2E model interaction

Here, therefore, the concept of federation is lacking since within a company a certain employee accesses it through credentials issued by the company itself. Here security is very important in that a certain employee( obviously depending on the position they hold within the company) might want to access sensitive information. So a main point in this context is strong-AuthN( maybe

going to use Multi-factor AuthN). The concept of SSO here is important but within the company in that you can think of a given user wanting to access multiple services without always going to re-enter credentials. So the internal services have to be federated with each other.

## Chapter 3

# Access Management Processes

In this chapter, the main processes underlying access management software, namely those of authentication and authorisation, will be explained. It is indeed necessary to distinguish the 2 processes in a clearly defined manner. If the former indicates who has access, the latter refers to what is authorised to after access has been granted. To give a concrete example, let us suppose that a person wants to take his car from the condominium garage (therefore shared by several people). Access to the garage is granted by the authentication process. Only persons who can do this should have access to the garage, so for simplicity of reasoning, only those who live in the building. But after gaining access, a person living in the building is only allowed to take his own car, not those of others. So this is where the authorisation process comes in, i.e. the consent for a particular person to only be able to take his or her own car.

### 3.1 Authentication

There are different definitions of Authentication:

- RFC-4949 (Internet security glossary)  
“the process of verifying a claim that a system entity or system resource has a certain attribute value”
- NIST IR 7298 (Glossary of Key Information Security Terms):  
“verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system”.

An important common point of these two definitions is that when we talk about the authentication process we define authentication of an actor meaning that it could

be not only a human being (interacting via software running on hardware), but also a software component or a hardware element (interacting via software).

While authenticating an actor, there are three categories of factors that can be used:

- Knowledge: authentication relies on something that the user knows, e.g. static passphrase, code, personal identification number. The associated risk is in the storage, in the way it is possible to demonstrate that knowledge and in the way it is transmitted.
- Ownership: authentication relies something only the user possesses (often called an "authenticator"), e.g. token, smart card, smartphone. The associated risks can be in the authenticator itself: it can be infected with a malware, or it can be manufactured in a country that imposes some government control on it, or it can be stolen, cloned, or used without the owner's authorization.
- Inherence: something the user is, e.g. a biometric characteristic (such as a fingerprint). The associated risks can be in counterfeiting and privacy: it is much worse than the previous cases, because for example a biometric characteristic cannot be replaced when "compromised".

### 3.1.1 Basic Authentication

HTTP basic authentication is a simple challenge and response mechanism with which a server can request authentication information (a user ID and a password) from a client. Basic authentication is considered the simplest method of access control access to web resources as this does not require the use of cookies, session ids and login pages. In addition, this authentication methodology does not require prior to credential exchange the handshake of a connection. The client will have to authenticate itself for each realm. A realm means the scope, that is, it identifies a set of resources for which access can be gained with certain credentials. Going to analyze in more detail what happens:

- The client wants to make access to a resource on a server, and so you type in the url for that resource without any kind of credentials.
- • The server, which wants that resource to be protected, then lets the client know this by going and sending an HTTP 401 response. That response contains an HTTP header called WWW-Authenticate. The header for such a ripsposta is made this way:  
WWW-Authenticate: Basic Realm ="realmexample"
- The client in turn then enters its credentials. The user will enter them within a modal that opens on the page on which he is browsing. The credentials are

b-64 encoded and then inserted within an HTTP request in the Authorization header in this manner:

```
Authorization: b64("username:password")
```

Web clients can store the authentication information for each realm so that users do not need to retype the information for every request. When the web client has obtained a user ID and password, it resends the original request with an Authorization header. Alternatively, the client can send the Authorization header when it makes its original request, and this header might be accepted by the server, avoiding the challenge and response process.

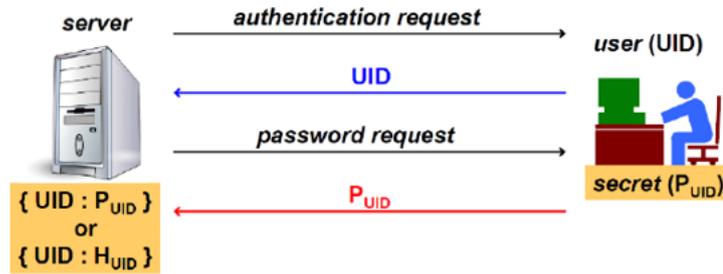
From a security point of view this schema is not considered to be a secure method because the user ID and password are passed only b64-encoded over untrusted network. So this schema can be considered secure only in the case the connection between client and server is secure, for example if TLS protocol is used. For more information about this topic is possible to see the document.[7]

### 3.1.2 Username and Password Authentication

Passwords throughout history have been used to prove whether the user was really who they claimed to be. They have always played a key role within security as they are the key to access sensitive data and resources. When a user intends to register for a site, he or she is required to choose a password that he or she must then re-enter when he or she wants to log in. There may be policies on password choice, i.e., going to use a minimum number of alphabetic characters (upper and lower case), having to enter numbers and/or special characters or not. These considerations are obviously made by the one managing the system to which you want to log in, and you decide this based on the sensitivity of the data to be protected. Obviously there has to be a balance between the security requirements demanded for the choice of password and the user experience. Now let's see from a more technical point of view how password is sent to the server and how it's stored on the server side.

We have to imagine that on the server side there is a table containing username and password in clear or with a function  $H$  computed over the password. So there is first an authentication request from the server, the client answer with its UID and then the server challenge the client with a password request and the client response with its secret. Analyzing this configuration from a security point of view there are 2 question that we need to ask ourselves:

1. "How the password is transmitted ?"
2. "How the password is stored on the server side ?"



**Figure 3.1:** How client interact when using Password authN

To answer to the first question the password can be sent in clear but then everybody can read it and it is very dangerous. To avoid this kind of situation we have to use a secure channel between client and server, so for example using TLS protocol. In order to increase the security on the server side, it must be made sure that the password is not stored in clear on the database. Of course if it were so then the comparison with the secret coming from the client and the one stored on the database would be easier and immediate but not at all secure. So what one has to do is to store a digest of the password, using a hash function. To understand what a digest and a hash function is I refer to this article where it is explained in a simple way.[8]

In this case even if the database with all the passwords is stolen then anyway the attacker is not able immediately to read all the passwords.

Password-based authentication is usually convenient for the user, but only as long as he has to remember just one password, so a reusable one. The current situation is unfortunate, because in some applications there is the need of several passwords, that cannot be kept in mind by a person, so they would need to be stored user-side, and there insecurity starts. Some disadvantages of password-based authentication are:

- The user-side password storage: it could be written on a post-it or on a client-side password manager (also called password wallet), that stores it encrypted typically using only one passphrase;
- Guessable passwords;
- Server-side password storage: the server must know the password in cleartext or in an unprotected digest of it (an important attack for this problem is the “dictionary attack”.[9]);
- Password can be sniffed while it is sent across the network;

So the best practice to use when is used this kind of authentication should be:

- You should use a mixture of alphabetic characters (uppercase and lowercase), including digits and special characters.
- Use a long password, at least 8 characters.
- Never use dictionary words ,because the attacker uses dictionary from all languages.
- Frequently change the password. If the same password is kept for a long time then the attacker has more time to do the computation of it.

### 3.1.3 Identity Federation

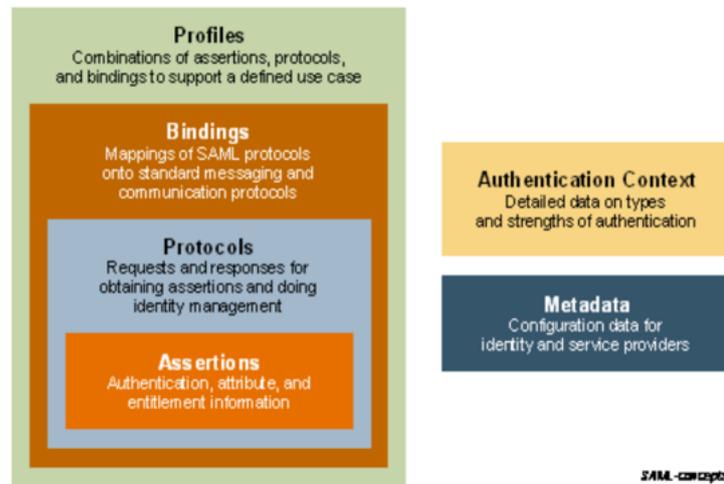
The concept of identity federation was already introduced in the second chapter. Here we will expand on that concept, first descriptively and then going on to talk about 2 standards on which this concept relies with regard to authentication, such as SAML and OpenId Connect. We also talk about federation when we talk about authorization but this will be addressed later in this chapter. To better understand this concept, it is good to first understand the roles that are played.

When we talk about the Relying Party or Service Provider (SP) we are talking about the actor to whom a given user makes access request for a given resource. Whereas when we talk about Authentication Server or Identity Provider (IdP) we are talking about the actor who grants the user permission to be able to access the requested resource. To be more precise we usually talk about Relying Party and Authentication Server when these belong to the same Web domain and this relates more to the case of a delegated Authentication. While we talk about SP and IdP when these do not belong to the same domain (same web environment), and in this case we talk about federated authentication. So underlying the concept of federated authentication there has to be an awareness of an agreement between a service provider(maybe one business company) and an IDP (for example another business company). Today an enterprise has a huge number of employers, partners and customers so has to find a way to handle all these access in a simplified way. Federated identity gives the possibility to some individual to access to various Web sites with just one sign in. Companies nowadays have to share between them a lot of informations and so have to establish between them business-agreement and policies for securely sharing information; they must also comply with federal laws regarding the exchange of personal data. Since September 2001, the Liberty Alliance ([www.projectliberty.org](http://www.projectliberty.org)),a global consortium of more than 150 companies and nonprofit organizations, has been developing open standards for federated identity and identity-based Web Services. Here the complete article.[10]. Now we can see how federated authentication is implemented and which standards are used.

## SAML

The Security Assertion Markup Language (SAML) standard defines an XML-based framework for the secure exchange of information across enterprise boundaries between business partnerships. This is done by issuing assertions that one application sends to another across the enterprise boundary. In addition, the SAML standard also defines the rules and syntax for creating and communicating with these assertions.

To understand this complex protocol is a good way to start from its architecture:



**Figure 3.2:** SAML Architecture

The core of SAML protocol is the assertion or also the response that is emitted from an Identity Provider (the request from a Relying Party is not mandatory). SAML assertions carry statements about a principal that an asserting party claims to be true. The valid structure and contents of an assertion are defined by the SAML assertion XML schema. At the heart of the framework defined by the SAML standard are assertions i.e., the response issued by an Identity Provider (the request from the Relying Party, as will be seen in a stream later, is not necessary). Then there are the SAML protocols that are used to cause an appropriate response tied to a given request to be returned. Then we find the bindings that define which protocol (HTTP or SOAP) is used to transpose these SAML messages. And finally, at the highest level we find the profiles, which by putting together assertions, messages, and bindings, define which business case is being analyzed and thus for which situation it is intended to use the SAML standard. On the right side of the

figure are two other important concepts namely:

- **Metadata:** It is an xml file that is exchanged between the 2 partners before communication begins. This specifies who the Idp is within a federation, who the SP is, and the keys that are used for signing and encrypting messages.
- **Authentication context:** This concept is used when an SP wants to know how a user is authenticated on an Idp. It is good to remember that the authentication scheme used between the user and the Idp is not something defined within the SAML standard but free and independent of it. To provide such information an authentication context can be used in the authentication statement of an assertion (or referenced). It may also happen that it is the SP that requires a certain authentication context, and this can be done by going to declare within a SAML request that the user is to be authenticated with a certain mechanism, such as with multi-factor authentication.

- 

Now that we have seen what the SAML architecture is in general now we can see what the various components of that architecture refer to by going to see in more detail what values they can take on:

- **Assertion:** SAML allows for one party to assert security information in the form of statements about a subject. SAML defines three kinds of statements that can be carried within an assertion:
  - **Authentication statement:** This assertion is related to the authentication performed by an identity provider. An issuer declares that: A certain subject S, at time T, was authenticated with the mechanism M. With this definition is important to specify that SAML doesn't perform any kind of authentication (it's not part of this protocol).
  - **Attribute statement:** This assertion is made when an Idp wants declare that: the subject S is associated with one or more attributes that currently have specific values. For example a certain user "Matteo" in a specific security domain "polito.it" is associated with "Department" and the value is "DAUIN".
  - **Authorization statement:** This assertion is sent from an Idp when it wants declare that when it wants to declare that a certain decision has been made regarding access to the request made by a subject S for a certain resource R of a type T based on evidence E

To better understand what an assertion really is here there is an example of Authentication statement and what are the meaning of different lines.[Fig 3.3]

```

1: <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
2:   Version="2.0"
3:   IssueInstant="2005-01-31T12:00:00Z">
4:   <saml:Issuer Format="urn:oasis:names:SAML:2.0:nameid-format:entity">
5:     http://idp.example.org
6:   </saml:Issuer>
7:   <saml:Subject>
8:     <saml:NameID
9:       Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
10:      j.doe@example.com
11:     </saml:NameID>
12:   </saml:Subject>
13:   <saml:Conditions
14:     NotBefore="2005-01-31T12:00:00Z"
15:     NotOnOrAfter="2005-01-31T12:10:00Z">
16:   </saml:Conditions>
17:   <saml:AuthnStatement
18:     AuthnInstant="2005-01-31T12:00:00Z" SessionIndex="67775277772">
19:     <saml:AuthnContext>
20:       <saml:AuthnContextClassRef>
21:         urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
22:       </saml:AuthnContextClassRef>
23:     </saml:AuthnContext>
24:   </saml:AuthnStatement>
25: </saml:Assertion>

```

**Figure 3.3:** Assertion with Authentication Statement

The first line begins the assertion and contains the SAML assertion namespace. From the second line to the sixth one are represented informations about the assertion, such as protocol's version, when the assertion was created and who is the issuer. From the seventh to twelfth line there are informations about the subject of the assertion. Then it's possible to see conditions of the assertion. In this case these include the period of validity of the assertion. And finally there is the authentication statement. In this case this indicates that the subject was already authenticated on the Idp using a password-protected transport mechanism at given date and time.

- • Protocols: SAML, as described earlier, describes a set of generalized request/response protocols. Here by way of example only 2 will be mentioned:
  - Authentication Request Protocol: defines a way in which a sender can request an authentication request about a user and also, optionally, an attribute request. This protocol is used by the SSO Web Browser when einderizing a user from an SP to an IdP when it needs to obtain an assertion to establish a security context for the user at the SP.
  - Single Logout Protocol: It serves when the user has multiple active sessions on different applications. This protocol causes the user to log out simultaneously from all applications. This process can be initiated directly by the user in a manual mdo or also caused by the SP or Idp due to a session timeout or administrator command.

- Bindings: A binding defines how a message is carried through the underlying protocols. Here we see 3 main types of bindings:
  - HTTP Redirect Binding: It is used when a message is transported using HTTP redirect messages (302 status code).
  - HTTP Post Binding: Defines how SAML messages are transported within a b64-encoded content of an HTML form control.
  - HTTP Artifact Binding: Used in the Artifact Resolution Protocol. In this case, the assertion is not sent directly with the response to the authentication request, but an artifact (a small fixed-length value representing only a reference) is passed first. The artifact receiver uses the Artifact protocol to request the message creator to dereference the artifact and return the actual protocol message. The artifact is usually sent via an HTTP redirect, while the request and resolution response occur via a synchronous binding, such as SOAP.
- Profiles: SAML profiles define how assertions, protocols, and bindings combine to provide a greater form of interoperability. Here we will look at 2 in particular:
  - Web Browser SSO profile : defines how 2 entities exchange an authentication context to achieve the single-sign-on concept with a Web browser. An example of this profile will then be presented.
  - Name Identifier Mapping Profile: Defines how Name Identifier Mapping Protocol is used. This protocol allows to map a name identifier into another one. It permits, for example, one SP to request from an IdP an identifier for a user that the SP can use at another SP in an application integration scenario.

Now to have a full vision of the protocol is proposed an example of a type of profiles, the Web Single Sign On Profile.

This profile provides a wide variety of options, primarily having to do with two dimensions of choice: first whether the message flows are IdP-initiated or SP-initiated, and second, which bindings are used to deliver messages between the IdP and the SP. When it comes to SP-initiated, it is easy to guess that first there will be a SAML request from the SP with subsequent response/assertion from the Identity Provider. Whereas when we talk about IdP-initiated we are talking about a flow that does not involve any request from the Service Provider but directly the issuance of an assertion by the IdP.

A second type of choice that can be implemented depending on the context is the

type of bindings i.e., as described earlier, how the SAML messages are transposed, i.e., both the request ( if any) and the assertion. It should first be noted that the transport of the two messages does not necessarily have to be the same e.g. you may have the request being made through an HTTP Redirect while the assertion (or response) is sent through an HTTP Post. Two cases will now be analyzed regarding the Web-SSo Profile. A first in which the request is made via HTTP Redirect and the response via HTTP Post. A second in which the request is via HTTP Post and the response via HTTP Artifact. Both will be SP-initiated.

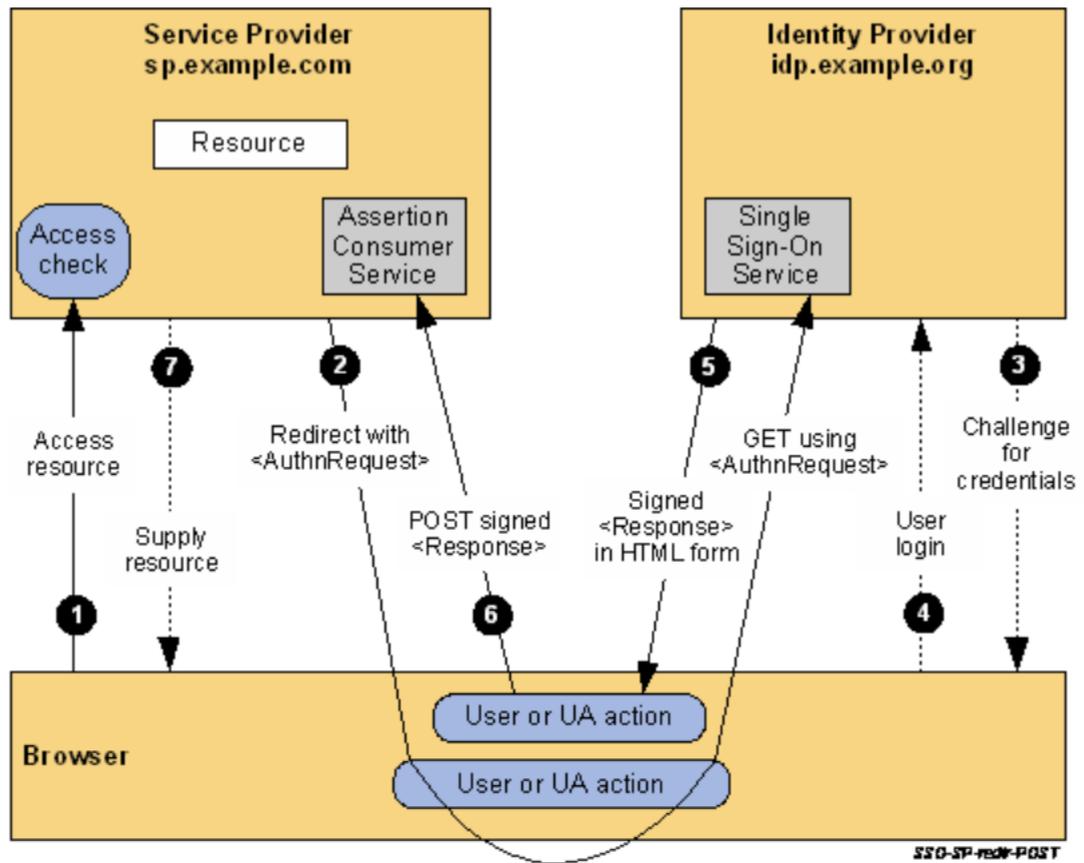


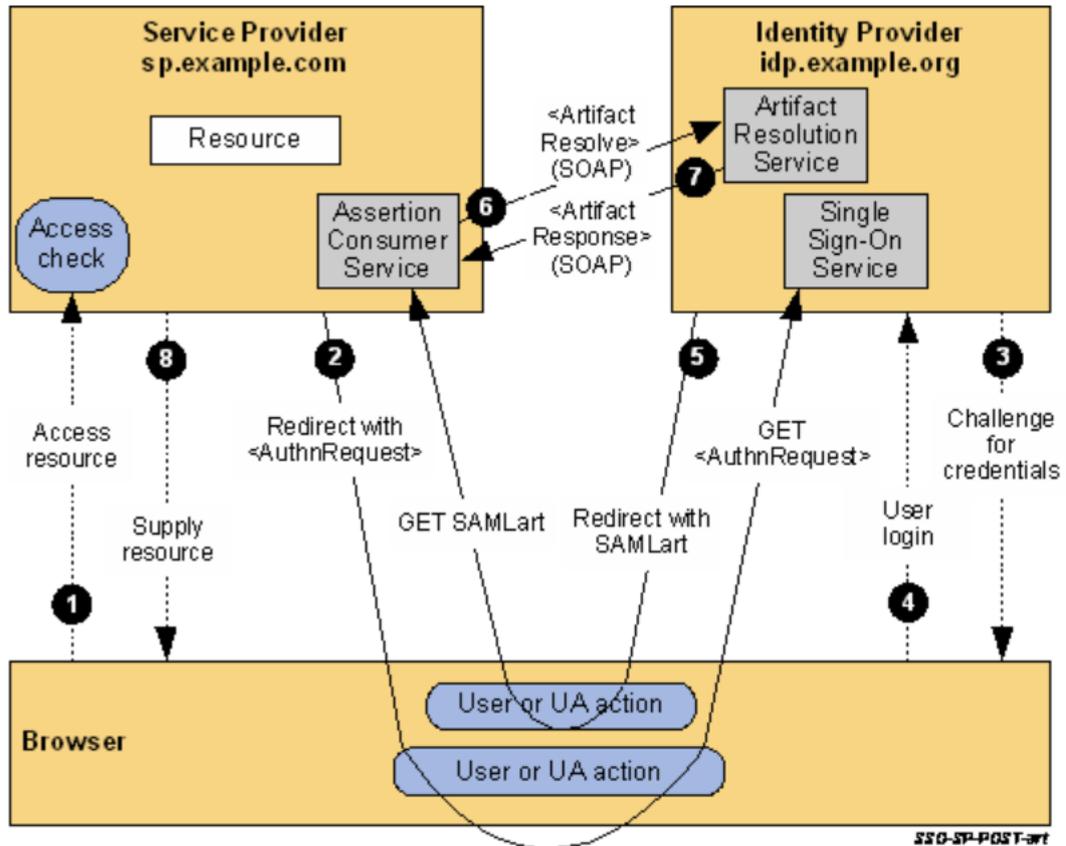
Figure 3.4: SP-Initiated SSO: Redirect/POST Bindings

1. The flow begins with the user attempting to visit sp.example.com. Since he has no logon context on this site, he must authenticate himself.

2. At this point the SP sends a Redirect(code 3xx) message to the browser, there where the message header contains the URI destination of the Idp along with an Authentication Request encoded as a URL query variable named SAMLRequest.
3. The Single-Sign-On service determines whether the user has an authentication context that is to comply with the default or what is requested within the AuthNRquest, i.e., the authentication requirements. If it does not have it then the user is challenged for it to perform the correct log-in on the IdP.
4. The user provide valid credentials for authentication process.
5. The Single-Sign-On service constructs a SAML assertion, which represents the user's authentication context. Since an HTTP POST is used, then the assertion is digitally signed and placed inside a SAML <Response>. The <Response> message is then placed in an HTTP form as a hidden form.
6. The browser sends an HTTP Post to the SP's Assertion Consumer Service.
7. An access check is made to establish if the user has the authorization to access to the desired resource.

Now I'm going to analyze the case where the request is made through HTTP POST and the response is transported through HTTP Artifact. It is useful to send the authentication Request via post and not redirect when there may be space issues as the URL has length limits.

1. As before the user wants to access to some resource on the SP's site but doesn't have a valid logon session.
2. The SP sends an HTTP response (code 200 ok) to the browser. The HTTP form contains a SAML <AuthNRquest> encoded as the value of a hidden form.
3. As before the SSO determines if the user has already a valid logon session on the Idp. If not the user challenge the user to provide valid credentials.
4. The user provides a valid username and password.
5. The Identity Provier creates an arifact, containing the id of www.example.org and a reference to the <Response> message. You can see how the Artifact binding then allows this message to be given to the Sp either with HTTP Redirect or with HTTP Post. Note also how since the response is not invaded



**Figure 3.5:** Fig 18-SP-Initiated SSO: POST/Artifact Bindings

at this time it is not necessary to go in and digitally sign that message. However, if this is not done then , if the SP in the future needed to prove the assertion, this would not be possible since it is not signed.

6. The SP's Assertion Consumer Service now sends a SAML `<ArtifactResolve>` message containing the artifact to the IdP's Artifact Resolution Service endpoint.
7. The IdP's artifact resolution service excises the message from the artifact and locates the corresponding response message associated with it. This is placed within a SAML `<Artifact Response>`, which is returned to the SP. The SP, at that point, extracts and processes the Request, and based on it , creates or does not create an authentication context for the user in question.
8. An access check is made to establish if the user has the authorization to access to the desired resource.

This case is simpler (keys or certificates are not needed) but it takes a bit more time because it is needed to open a separate network channel

This was just a small description of a fairly complex protocol used for multiple purposes. Here only a general level description was given to understand how the concept of Identity Federation can be implemented and specifically regarding the Single Sign On Web Browser case. To go deeper into the protocol and to understand its practical aspects I report here the reading of this document.[11]

SAML is XML-based. XML is simple but quite heavy, so SAML is typically used in PC or server-based environments. This means that is difficult to support in light/mobile environments. Some people use SAML, while others use OpenID-connect, which makes things very similar to SAML but uses JSON instead of XML. So then there is the description of OpenId-Connect but first to better understand this protocol there is a short introduction on what is a JSON Web Token.

### **Json Web Token**

The Json Web Token(JWT), pronounced “jot”, is an open standard, which defines a way to securely and contentually transmit information over an insecure network, in the form of JSON objects. Due to its small length, this can be transmitted in various ways: in a URL, in an HTTP POST, in an HTTP header. It is also faster in transmission, and less verbose than xml. The result is that when encoded this information is smaller than in a SAML assertion. A Json Web Token can be used in several ways:

- **Authentication:** After a user has successfully authenticated to an application this can receive a token ID, within which is information about the user.
- **Authorization:** When a user gives permission to an application to access a resource on his behalf, that application through a mechanism that we will see later in this chapter( Oauth) receives an access token. This access token is precisely what will allow access to the application and this can be a JWT.
- **Information Exchange:** The JWT is also useful for exchanging information between two partners. This is because being digitally signed then one of the two partners can understand that this token, and therefore this information, is from the correct partners and that it is not someone else who has modified it.

A JWT consists of three concatenated Base64url-encoded strings, separated by dots(.):

- **JOSE Header:** contains metadata and algorithms to digitally sign the token.

- JWS Payload: contains what contains the real content of the token, that is, what you want to transport. For example, it can transpose a user's identity and the permissions he or she has.
- used to validate that the token is trustworthy and has not been tampered with. When you use a JWT, you must check its signature before storing and using it.

An example of JWT is given in [Fig 3.6].

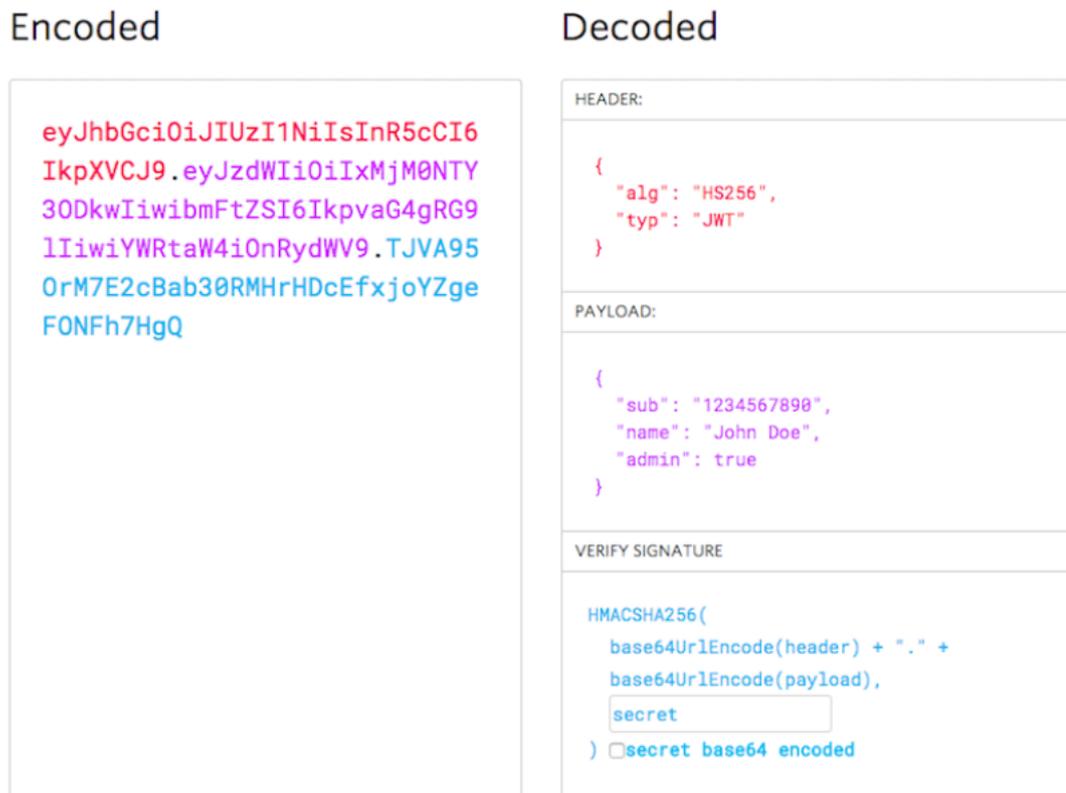


Figure 3.6: Example of JWT

On the right part of the image is possible to see how it results when it is decoded and what are the several fields of the three parts.

## OpenID-Connect

OpenID-Connect is a simple identity layer built on top of the OAuth 2.0 protocol. OAuth protocol will be discussed later in this chapter. It enables clients to verify the identity of End-User based on the authentication performed by an Authorization Server. OpenID Connect allows clients of all types, including Web-based, mobile, and JavaScript clients, to request and receive information about authenticated sessions and end-users. OpenID Connect implements authentication as an extension of the OAuth 2.0 authorization process. Use of this extension is requested by Clients by including the openid scope value in the Authorization Request. Information about the authentication performed is returned in a JWT called an ID Token. Different terms are used in this protocol to refer to the various entities involved. The client in this case is the relying party (RP) that wishes to use OpenID-Connect for authentication. The server, which is the OpenID-Provider (OP), is conceptually similar to the IdP, has various endpoints:

- Authorization endpoint: it is called authorization, but it performs authentication.
- Token Endpoint: something that verifies if a certain token generated during the protocol is valid or not.
- UserInfo endpoint: if the user has given the consent, then the client can retrieve information about the user.

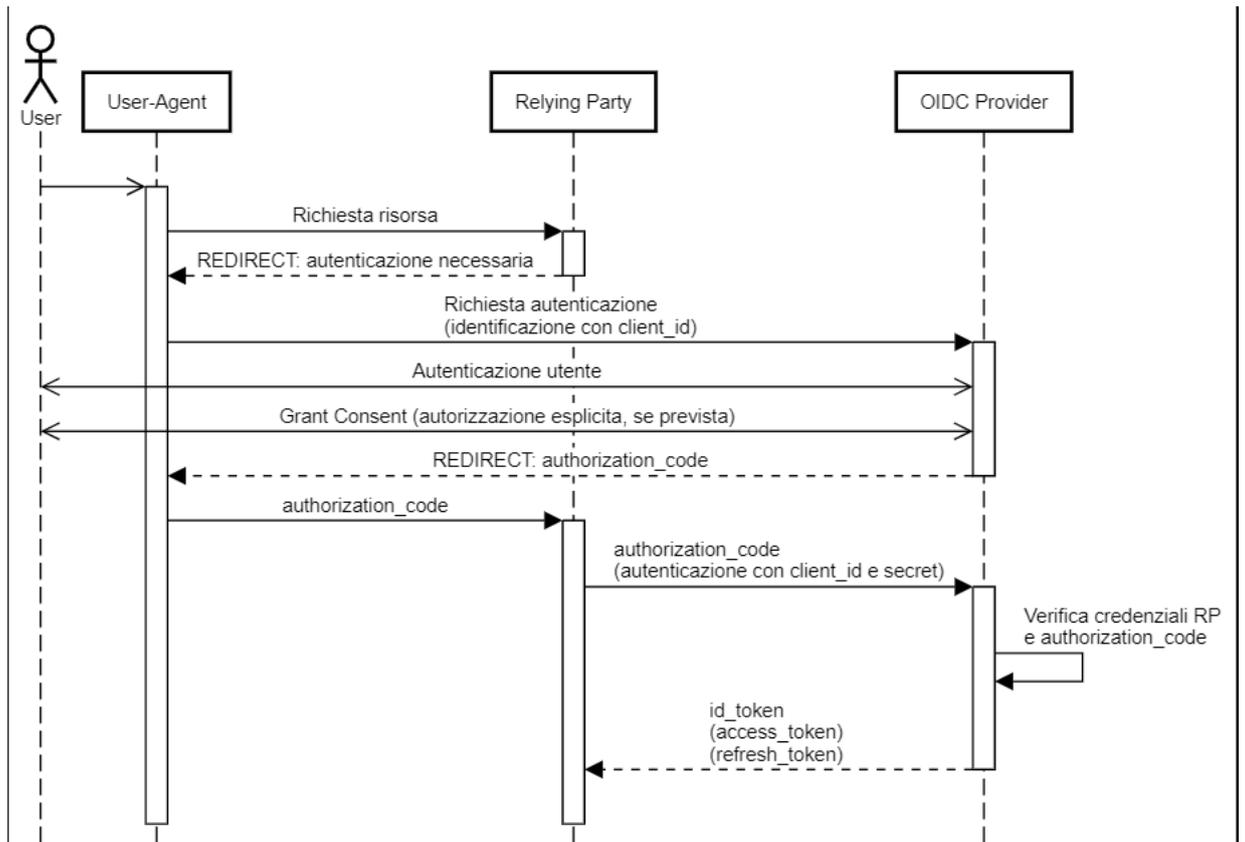
There are 3 possible flows that characterize the protocol and they are:

- Authentication code flow
- Implicit flow
- Hybrid flow

The flow used is determined by the response-type value contained in the Authorization Request.

Let us first analyze the case of Authorization Code Flow.[Fig 3.7]

- As you can see, the flow starts from a user agent who wants to access a resource on the Relying Party.
- The latter not having an authentication context for that particular user redirects the user on the OIDC Provider.
- The IdP challenges the user to authenticate and then there may be a request to the user to authorize the client to act on the user's behalf.

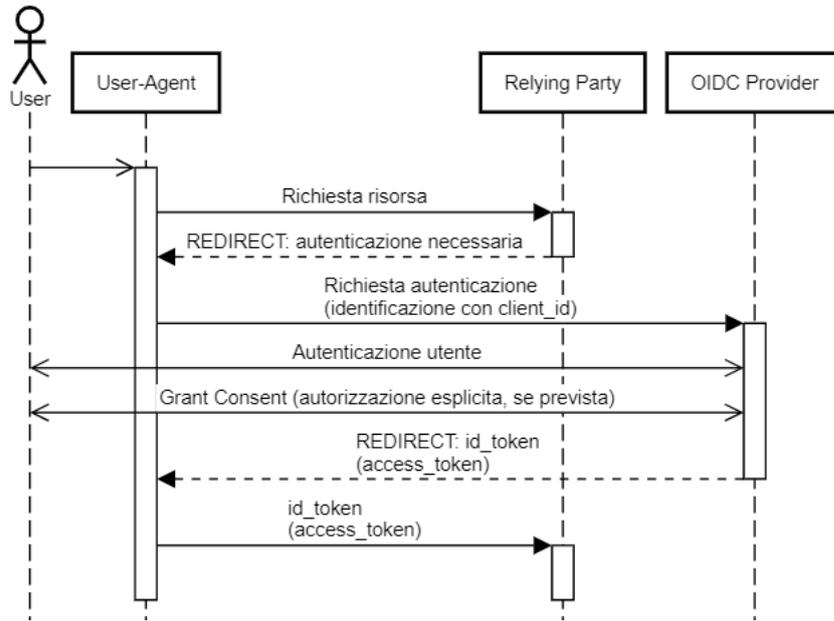


**Figure 3.7:** OIDC-Authorization Code Flow

- Once the user has given consent, the provider grants an authorization code. This code is given to a user agent for a specific client. After that this code will be precisely used by the RP to show that it is the client in question.
- Then the client using this code requests a response to the Token Endpoint. Then it receives a response with the ID Token and an access Token in the response body.
- Client validates the ID token and retrieves the End-User's Subject Identifier.

In this flow it can be seen that a token is not directly issued by the OIDC Provider but an authorization code is first granted. This is useful in situations where you do not want a token to be passed to a user agent and possibly other malicious applications with access to the User Agent. In this flow The OIDC Provider can also authenticate the client before exchanging the Authorization Code for an Access token. This flow is useful for clients who can keep a client secret between themselves and the Authorization Server.

Now let's analyze the implicit flow,[Fig 3.8].



**Figure 3.8:** OIDC-Implicit Flow

- The client prepares an authentication request and sends it to the Authorization Server
- Authorization Server authenticates the end-user and obtains his authorization/consent
- Authorization Server sends the End-User back to the Client with an ID Token and, if requested, an Access Token. Then, the client validates the ID token and retrieves the End-User's Subject Identifier.

The Implicit Flow is mainly used by Clients implemented in a browser using a scripting language. The Access Token and ID Token are returned directly to the Client, which may expose them to the End-User and applications that have access to the End-User's User Agent. The Authorization Server does not perform Client Authentication. This flow eliminates a direct communication between Service Provider and Identity Provider just as was done in SAML by making use of HTTP GET and HTTP POST. A good document about OpenId connect and his features can be found here.[12]

## 3.2 Authorization

So far I have talked about what the authentication process is, that is, going to verify who is the actor who wants to access a given system. In this section I will go over the authorization process and the standards that can be adopted to implement it. This process is such that a decision (positive or negative) is able to be given on the request by a user for a given resource present within the system. Authorization is thus the process by which access to a given resource is granted or denied. Assuming that someone has logged in to a computer operating system or application, the system or application may want to identify what resources the user can be given during this session. Authorization is seen both as the preliminary process for values to be set for access to a given resource (by the administrator) and the verification of those values when a request (by the user) arrives to gain access to that resource. Logically the authorization process is preceded by authentication.

To give an example, we can suppose that Bob wants to gain access to a garage inside which there will also be his car that he wants to open. Bob is able to gain access to the garage as he has credentials to do so. And that is the authentication process. But it is not pretended here in that he not having permission to be able to open any machine in the garage has to be regulated by some mechanisms (policy) to make sure that he can only access ( and therefore can open) the resources for which he is authorized to do so ( and therefore only the machines that he is authorized to open). And this second process is the authorization process.

Typically, authorization to a resource is determined by security policies through evaluation of which it is possible to understand whether or not a user is allowed to gain access to it. There are different types of security policies that also allow different granularity of authorization.

The first example of user policies are ACLs( Access Control List). They were based on lists for each user where it said what resources each user could access. However, this created maintenance problems due to the increasing number of resources. Other types of policies have since replaced them.

### 3.2.1 RBAC and ABAC

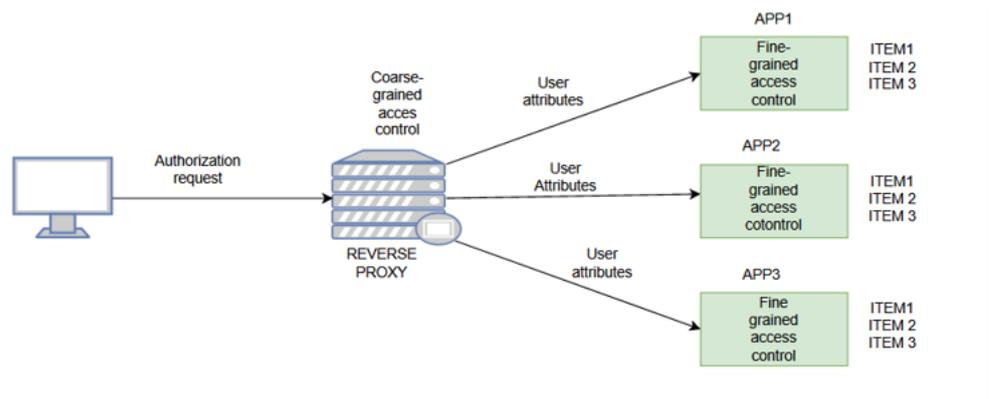
Authorization policies are thus a way to be able to set authorization values so that a request can be either fulfilled or denied.

A first example of policies are those that go by the name of Role-Based Access Control (RBAC). These are the simplest and are based on granting or denying authorization based on the role that the requesting user plays. So in such a scenario you have that one user can also have several roles and can access various resources.

While attached to the resources there will be policies that specify which people in a particular role can access them. These are easy to adopt but they only provide coarse-grained control access in that they only differentiate access based on the role of a given user and thus make the accesses also quite static. In addition, there is also a scalability problem here in that as resources increase (which require the creation of new roles to access them) one would always have to go and make updates. Maintaining roles becomes challenging as more resources are added to networks, causing role explosion. Another problem that might be there is when a worker need to make access to a resource for a given period of time but the scope of access to that resource is outside the permissions for this user's role. With the RBAC policy this could not be achieved since it has a static view of accesses.

Another type of policies are those called Attribute Base Access Control(ABAC). These security policies consider not only the role a user plays but are based on different attributes of the user, attributes of the resource and context attributes. These allow a more granular level of access as they are based on several factors. For example, you may have that a user in a certain role can access a certain resource but only for a period of the day and not 24/7. Examples of user attributes are ID, name, organization, role, security clearance, nationality, etc. Resource attributes include name, owner, data creation date, etc. Examples of environmental attributes are access location, access time, and threat levels. These policies are more secure in that they restrict access but are more complicated to manage than RBACs.

Both types of policies have advantages and disadvantages.[13]. They can also be used in a complementary way by going to use RBACs for a first access control and ABACs for a second as shown in the figure.[Fig 3.9] As you can see in the figure



**Figure 3.9:** RBAC and ABAC policies

you have the implementation of both types of policies. On the reverse proxy where

requests must be forwarded for all applications protected by it the control can be more coarse-grained and fast. While then there must be another control on the servers hosting the applications so that this control is more targeted and secure i.e. respecting what are the ABAC policies.

### 3.2.2 Session Cookie

With policies then you can give access to a resource for a user. The mechanism of first authenticating and then authorizing is not something that has to happen all the time. In fact, if a user had to, for each request, first authenticate and then have each request authorized this would affect the speed and performance of a software and also of client-server communication. For this reason, a first mechanism that arose to make sure that the server hosting the resource can remember the information received from the client are the so-called session cookies. Session cookies contain a Session-id that the server sends to the client once the client has made a request for a resource. From there, the client whenever it needs to access that resource can present the session cookie to the server and retrieve a resource again. So, as you can see in Fig 3.10, the server once it receives a request

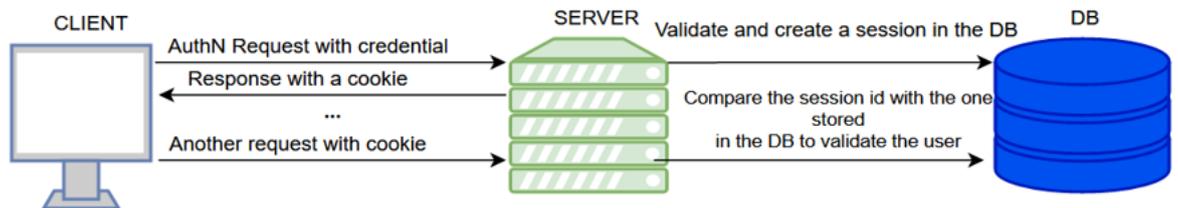


Figure 3.10: Session Cookies

from the client, either for authentication or authorization for a resource, goes and creates a session for the user by going and saving the parameters (that the user has decided to send it) in a database. Once this is done it sends a session id in the cookie (to the client) thanks to which at the next request it can go and retrieve the session information within the database. Cookies, however, are not free of security problems. In fact, there are several attacks that can occur against cookies. For more in-depth reading on these types of attacks I recommend this article.[14]

### 3.2.3 Oauth 2.0

After talking about how security policies can be applied to a user and seeing a first way to manage a session, we are now going to look at the Oauth2.0 protocol.

This protocol, as already mentioned in the last chapter talking about Oidc-connect, was born to implement delegated authorization. Oauth2.0 is a delegation protocol that allows a resource owner to authorize software to access the resource on his or her behalf without impersonating that person. It is good to give an example to better understand what we are talking about. Suppose there is a user, Bob, who is playing Candy Crash and has his own Facebook account. At some point in the game Bob wants to post his achievement (obtained on Candy Crash) on Facebook. So, what he wants to do is to allow a software (in this case Candy Crash) to be able to access a resource ( his Facebook account) without, however, being impersonated by Candy Crash, or rather without such software going to possess John's login credentials, for security reasons that will be analyzed shortly.

In order to implement this process even in the past there were mechanisms that however presented problems such as credential sharing. This is because perhaps in the past if a software wanted to access the resource on behalf of the user then they could ask the user to enter their credentials and then present these to the protected resource. Here 2 problems arise. The first is that such a mechanism required the user to have the same login credentials to access the software and the protected resource. The second is that the resource owner must trust the software that wants to access the resource since it is sharing its credentials. Moreover in the domain of the protected resource there can be no distinction in access between the resource owner and the software (the client) since both use the same credentials in the same way.

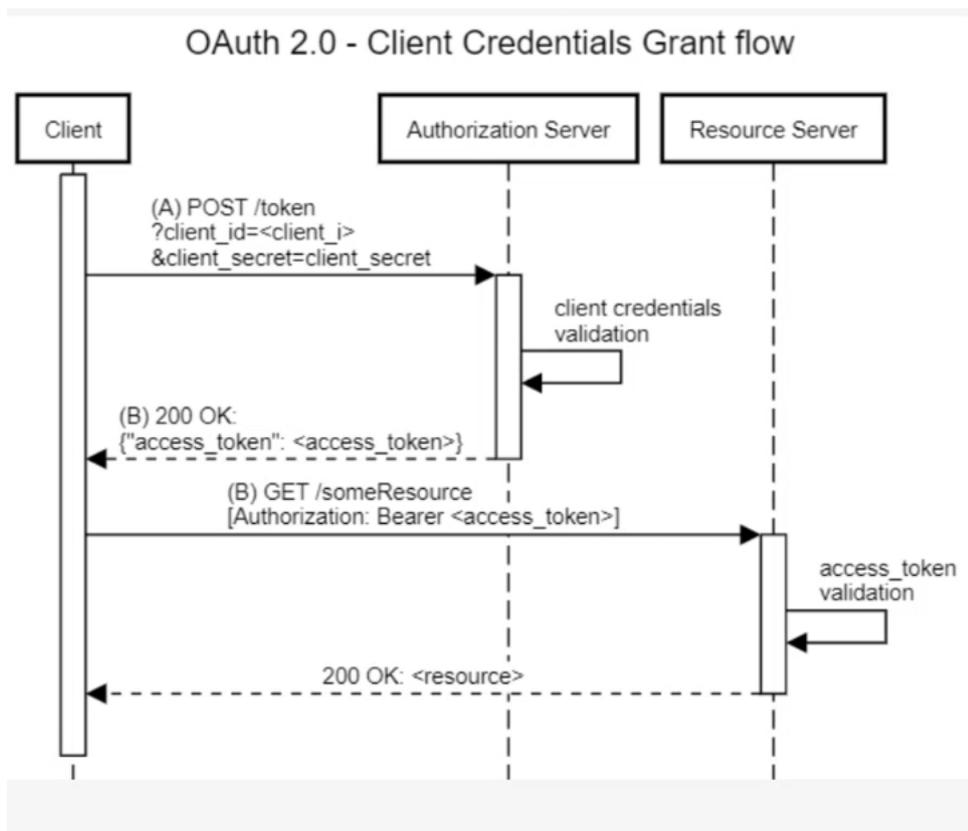
Another possibility would be to give a univresal key that allows the software(client) to be able to access the resource domain by being able to impersonate any user it wants. This would solve the credential sharing problem but would create an even bigger one in that now the client can access all its resources. For these reasons, the Oauth2.0 protocol was born, an evolution of Oauth1.0 (abandoned due to security concerns) that aims precisely at delegating access from a resource owner to a client but in a secure way. Before we delve into what the Oauth2.0 flows are, we need to understand what entities are involved.

- Resource Owner: As the name implies it is the owner of what you want to access. And he is the one who wants to delegate a client(called software first) to access on his behalf.
- Oauth Client: The client is the one who will make access to the protected resource having obtained an authorization.
- Authorization server: The authorization server is the component that authenticates, if necessary, the user in that perhaps he is yes registered on that domain but does not have an active session at the moment, after which it grants the client an access token that is the key to the whole Oauth2.0 protocol.

- Resource Server: The server where the desired resource is hosted.

Now it is time to move on to analyze the various Oauth flows. There are 5 of them, but actually today Implicit Grant Flow is not used much anymore.

A first flow that is going to be analyzed is that of the Client Credential Flow (Fig 3.11). If only this flow existed Oauth would not make sense since here the resource owner coincides with the client and so as you will see it is exactly the same process as when the server grants the client cookies. In this case, the client begins

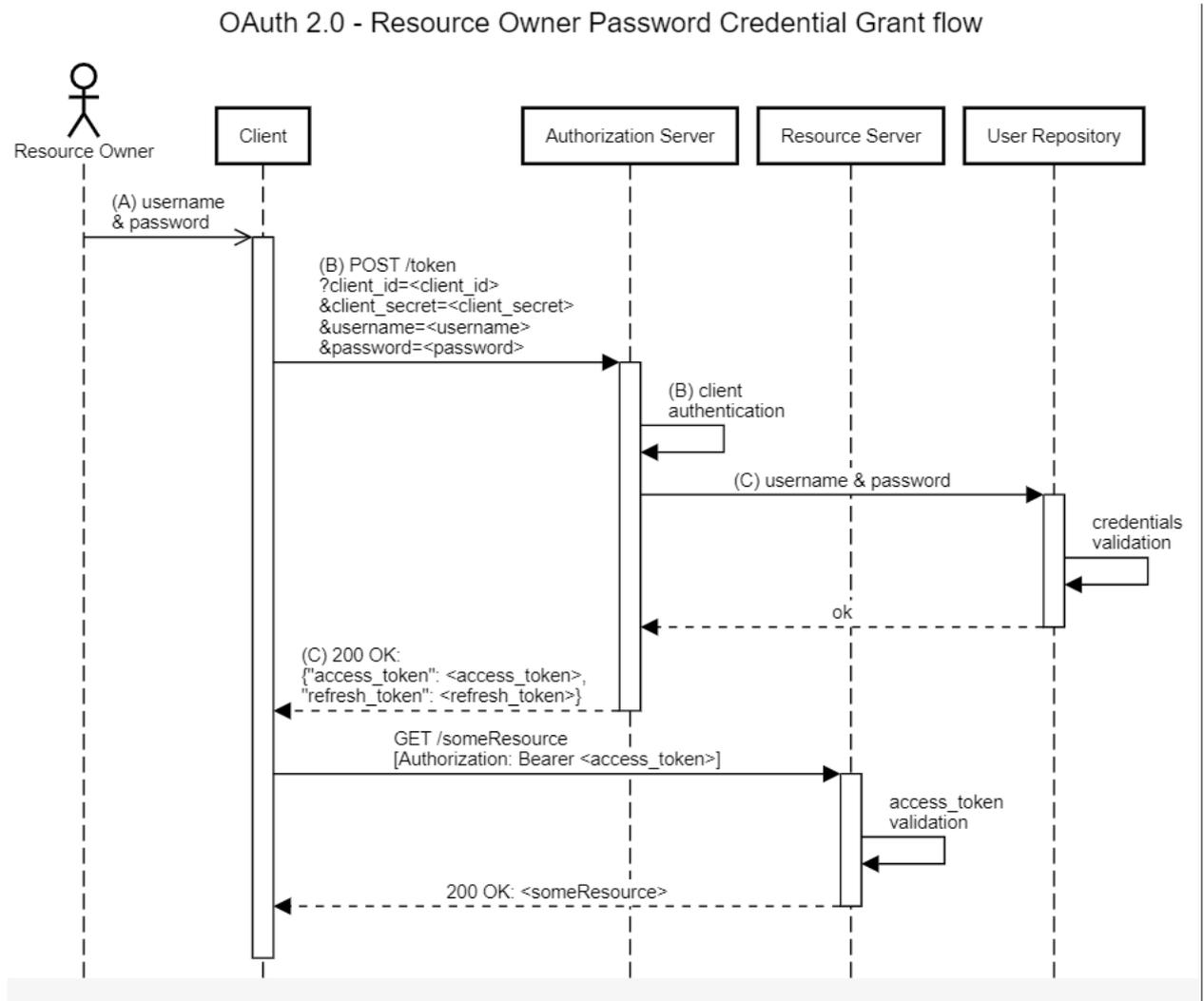


**Figure 3.11:** Client Credential flow

communication by presenting the client id and client secret. After validating the client’s credentials, the authorization server grants an access token to the client, which in turn will use it when it needs to make a request for a resource. At that point the resource server will validate the access token by going to check whether or not it is valid and if so will give access to the resource to the client. The validation of the access token, as the Oauth standard should be done by the Authorization server, but since Oauth2.0 is a flexible standard there are several implementations

of it. This is an advantage of OAuth2.0 and consequently of OpenId Connect over SAML which is a more rigid standard.

Now we see another stream called Resource Owner Password Credential (ROPC)(Fig 3.12). In that flow actually goes to implement the concept of credential sharing in that the resource Owner, this time a separate entity from the client goes to enter its credentials on the client . For this reason such a flow is used when there is complete trust in the client being used. First thing different from the previous flow



**Figure 3.12:** ROPC flow

is that here you have the actual presence of a resource owner. Also noticeable is the presence of a user repository where user credentials are checked.

What happens is that the resource owner gives its credentials to the client. The client for verification sends its credentials and those of the resource owner to the Authorization Server, which verifies them, and if they are correct returns an access token and a refresh token to the client. The access token as in the previous flow is used to gain access to the resource. The refresh token will be used to obtain a new access token when the previous one has expired.

As it is possible to notice both in the previous stream and in the ROPC, there is no user-agent present. This is because these 2 streams are used for back-channel communication. Instead, we will speak of front-channel communication when the communication also involves a user agent. Back-channel communication is more secure. Think of communication between 2 servers communicating with each other through the backend. Front-channel communication is used to interact with the user.

Now we are going to talk about the flow that represents the real essence of the OAuth2.0 protocol as it is the most comprehensive. The flow is called Authorization Code Grant Flow (Fig 3.13). In this flow the user begins communication. From the client he wants to access a resource that is on the server and then is redirected to the Authorization Server where he presents the client id. The first thing the authorization server does is to see if that user is authenticated. If not, it challenges the user to authenticate himself. Important thing to note is that OAuth2.0 does not define an authentication protocol but that is outside the scope of the standard. If authentication is successful then the client is granted the Authorization Code, a base-64 encoded string, which will then be used by the client to show the Authorization Server that it is indeed the client in question for whom the authorization code was given to the resource owner.

The client then, when it wants to apply for an access token, will send its own credentials to the AuthZ server in addition to that code. If the validation is correct then an access token will be released to the client which will then be used to access the resource. Also here you can see the release of the refresh token as well, which will then be used to be able to renew the access token. So this Authorization Code appeared in this flow is a demonstration of the delegation of the resource owner to the client to be able to access a given resource. Note then how while the Authorization Code is exchanged on the front-channel, the access and refresh tokens are given by the server directly to the client in the back-channel, this is because it is not necessary, and may be even dangerous, for the access tokens to go through a user-agent.

Not present in this schema but another important concept in the standard is that the access token has a parameter which is the scope. This is to indicate the access scope of a given token and therefore what resources can be accessed with

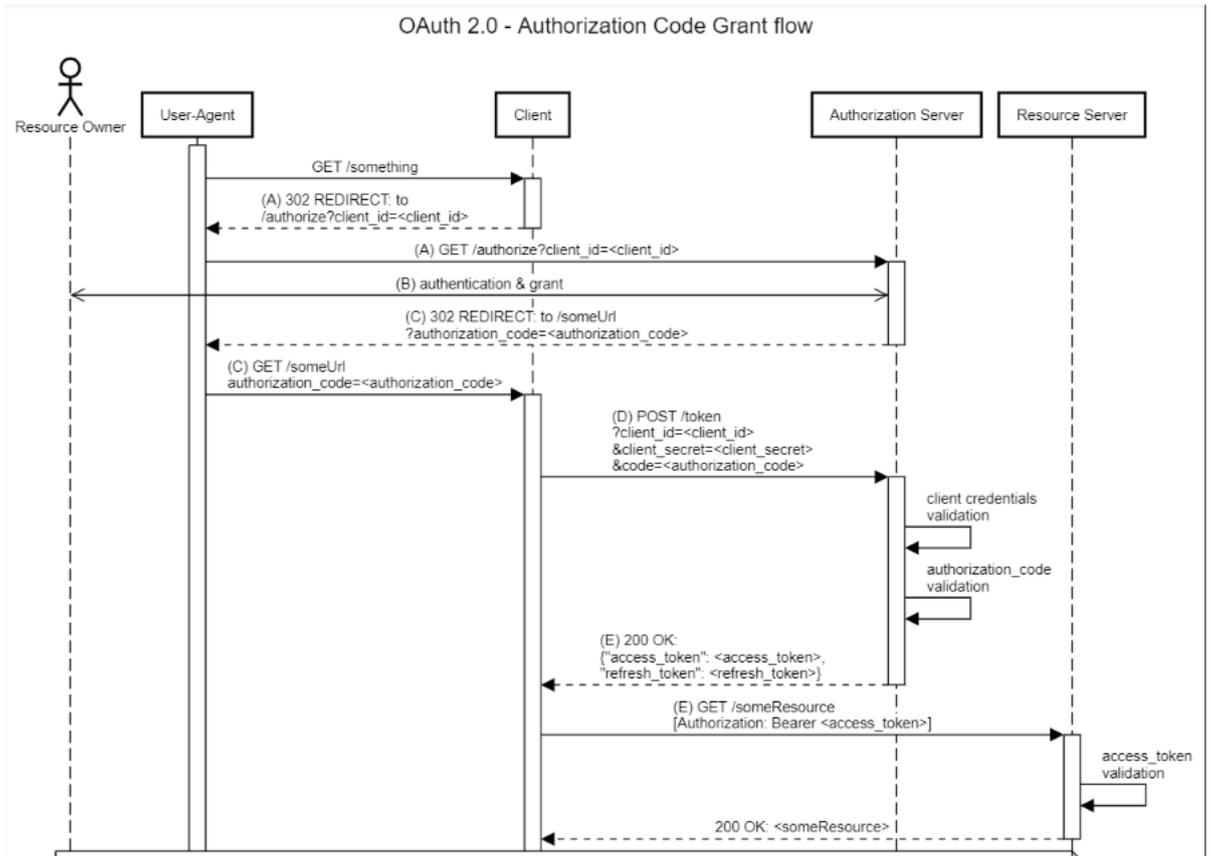
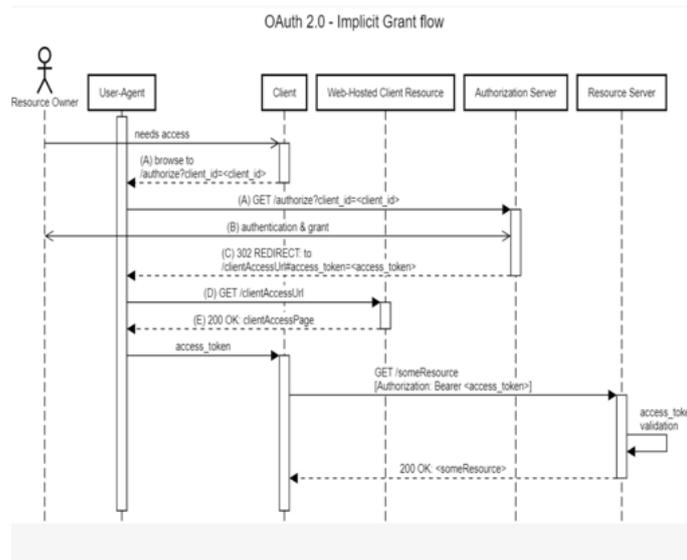


Figure 3.13: Authorization Code Flow

that token.

Now we are going to discuss a flow, the Implicit Grant flow (Fig 3.14), thought of when the client's confidentiality cannot be guaranteed. In previous flows the client could guarantee its confidentiality to the server by going to give its client-secret. As can be seen here, after the initial phase (up to the possible authentication of the user on the authorization server), the issuing of an access token is done directly. This is because, as already mentioned, the flow is designed for public and therefore non-confidential clients. Examples of public clients can be Mobile App or single-page application. A single-page Application does not have a backend server and when then cannot perform back-channel communications. In the AuthZ code grant flow it is true that the authorization code is exchanged on the front-channel but the secret key then to decrypt it was exchanged on the back-channel and stored on the back-end of the client. Here since there is no back-end there is no place to



**Figure 3.14:** Implicit Grant Flow

secretly store the client-secret or even the key to decrypt the code. So for this the issuing of an access token is done directly. For the same reason a refresh token is not given either, since the client is unable to store it. Return access token in the front channel is bad because the front channel is not a secure environment, user can be tricked to install a malicious browser extension, which can listen to user's network or access to browser history or physically just simple as stand behind user's back and snooping for the token. In addition, the lack of a refresh token causes the access token to have a long lifetime, and this as we will see later in this chapter is not a good practice in terms of security.

To make up for this lack of implicit flow in recent years, the Authorization Code flow with PKCE (Proof Key for Code Exchange) was born. This flow is like the regular Authorization Code flow, except PKCE replaces the client secret used in the standard Authorization Code flow with a one-time code challenge. This means the client app doesn't have to store a client secret (Fig 3.15). So in this case as you can see to make up for the lack of a client-secret a code-verifier is also generated , at the initial moment of the flow(in the redirect), by the client. This code verifier is then applied to it by the hash algorithm and is sent as to the server. At the time the client then makes a request for an access token the client will present the code-verifier ( string in plaintext) to the server to verify its "confidentiality". The rest of the flow is the same as what was the Authorization Code Grant flow. A recommended text for an in-depth look at oauth and the implementation of various flows ,even from a more technical point of view, is the following. [15]

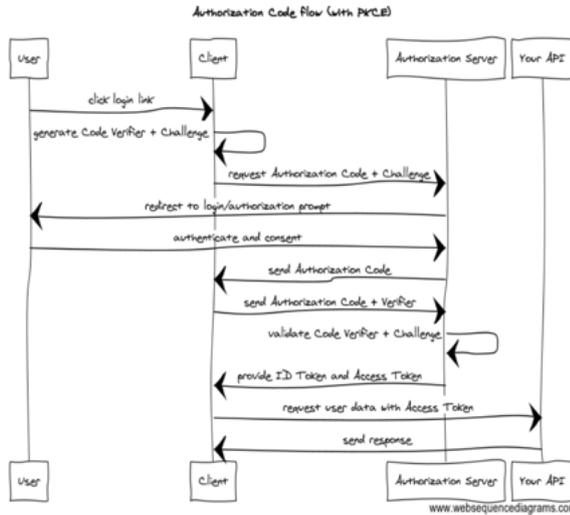


Figure 3.15: Authorization Code Grant Flow with PKCE

Below(Fig 3.16) is a general outline of when to use one stream or another depending on the context. We have thus seen that both Oauth and OpenId connect

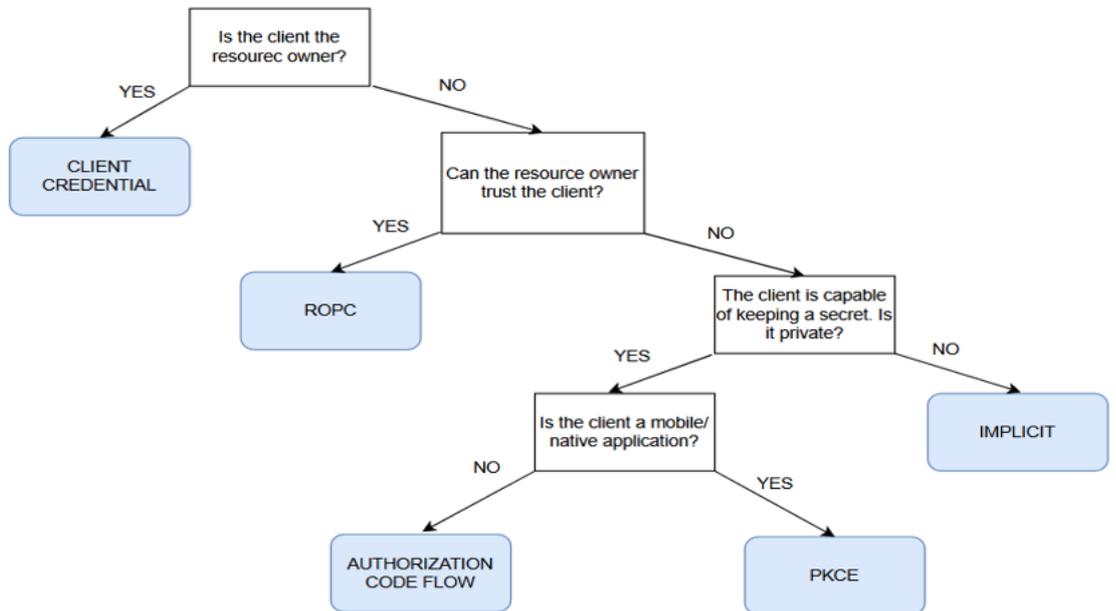


Figure 3.16: How to choose an Oauth flow

make extensive use of tokens. Tokens can be of 2 types:

- Opaque(a.k.a session tokens – a long random meaningless string which is a reference to some information stored in a database).
- Non-opaque (contains some meaningful information like a userID, encoded in base64), like JWT.

The advantages and disadvantages of a JWT over an opaque token are now analyzed. For example, JWTs have the advantage of not performing any database checks that might affect the latency of a communication since they are self-signed. In addition, thanks to this they do not take up any additional space on the database. Whereas with regard to opaque tokens these do not carry information but must be thought of as a string that acts as an index within a database from which session information is then retrieved, which can vary in size. On the other hand, however, these have the disadvantage of being difficult to revoke before their expiration compared to opaque tokens. This is because since the jwt is self-signed there is no information trail on a database. One way to revoke them would be to revoke the key with which they are issued. But doing so would create another problem, namely, that all tokens issued with that key are revoked. Whereas, to invalidate an opaque token, you simply go and delete the information corresponding to it within the database.

These reasonings are also reflected in what an access token and a refresh token should look like. As previously analyzed an access token is the means by which one is able to gain access to a resource while the refresh token is the token by which one gets a new access token when the old one expires is invalidated. What would be preferable to have is a short duration access token and a long duration refresh token. That way even if the access token were to be stolen and an attacker tried to use it to obtain a resource this would only be valid for a short period of time. Whereas in the case of a refresh token this has a long lifetime, as it must be used to obtain new access tokens . Thus it is preferable to have an access token that does not need every time a request is made for a resource to be checked against a database, and so with that it is preferable to have an access token that is a JWT. While as for the refresh token if it is stolen then the attacker could have as many access tokens detached as he wants (until the refresh expires) . So it is preferable to have a refresh token that is easily revocable and thus is an opaque token. An interesting article on how access tokens and refresh tokens can be managed is here.[16]

## Chapter 4

# Thesis Objectives

We have seen, therefore, in the previous chapters, the main methods used in the world of access management and its mechanisms. However, this also leads one to think of all the problems that may arise, especially within large companies, which must interact with other business companies, must manage a large number of internal users, and must also give customers who want to access the resources for which they are authorized the right to do so frictionlessly.

So, if one thinks of medium to large companies, the problems that can arise are many. First of all, one must take into account the fact that there are always more services, and therefore applications, in a company. These applications will be linked to those that already exist, especially now that the concept of microservice development is becoming increasingly widespread. These applications will, like the others, have to guarantee access to resources for the various users and thus, as seen in the previous chapter, will have to share data and have a mechanism for access. This does not only apply to multiple nascent applications, but also to the integration of legacy applications. Given the ever-increasing amount of demand for new services on the market, companies are finding it increasingly difficult to find suitable and competent people to develop applications, both on the front-end side and in terms of the security logic behind them. This, therefore, requires both an economic effort and internal company resources.

The goals of this thesis are about going to study the benefits that low-code philosophy can bring within the world of access management. This was done by going to study and understand how the (cloud-based) PingOne DaVinci software works. Such software provides a drag-and-drop interface through which one can implement the flows that characterize the user experience regarding access management, such as a registration, authentication and authorization process.

Here, working in a training environment, we wish to analyse the benefits of low-code

development compared to a traditional approach. This is to be seen for the three main mechanisms underlying identity and access management software, namely registration, authentication, and authorisation. We must therefore first see in the construction phase of the flows how much this new technique can make significant differences with respect to what is commonly done, based on programming in some language.

- For the registration phase, how quickly the fundamental blocks of any process leading to the creation of a new account on a site can be set up. Such fundamental blocks are, for instance, the forms for entering credentials, verification of e-mail.
- Then we will move on to the authentication process, the most substantial part of the work, where the multiple ways of authenticating oneself in an application will be analysed. If you think about any application today, be it web app or native, you will see that there are many methods of authentication. From the simple use of username and password, to social login, to the concept of SSO, through the use of federations, and Multi Factor Authentication. All this shows the difficulty of constructing and integrating such methods and managing all the information about users that an application must support.
- We then want to see how the method of access to a single resource is managed. Remembering what was said in the chapter authentication and authorisation are 2 very separate concepts. The resources that an application must protect are many. For each of them, it must be determined if and when a user may access them. Writing and modifying the resulting code involves a great deal of time as well as resources. It must be considered that resources always increase, especially when new users are added to the system. This must not only be analysed statically, but also from a dynamic point of view. that is why we do not only want to analyse an authorisation process that allows a user to access a resource either at any time or not at all. In the security sphere, context-access, or access based on the instantaneous attributes of a user, such as the network from which he is trying to gain access, his current IP address, or the geographical position in which he is at that moment, is becoming increasingly popular. These are all aspects that need to be analysed to allow secure yet frictionless authorization.

Before moving on to existing applications, while still remaining in the development environment (low-code of course), it must also be ascertained how much the testing phase can be accelerated in comparison with a traditional approach, and how much the security of the entire mechanism is not compromised.

You will also want to consider and simulate all those situations in which a customer, who makes use of this development, changes his or her mind about the desired

implementation and thus how much this may affect the adjustment time.

Following the implementation of these flows, it is also important to show how access to these flows by an application is facilitated. Access by an application to these flows is achieved by making use of the standards characterizing the world of access management without having to have developer knowledge.

The aim is to analyse how short the integration time of a logic within an application is. Furthermore, after doing so, it is necessary to verify how reusable a flow can be within different applications so that it does not have to be modified when inserted in different contexts.

You will want to see how the logic behind the applications can change and thus its complexity by checking the integration within them.

It will also have to be checked whether it is possible, thanks to the platform, to create federations between 2 applications. We will also want to check to what extent it is possible to set up different types of federations, e.g. SAML and OIDC. How the end user is affected by such a change or whether it is completely indifferent to the user-experience.

The development and security of applications, through this study, can be modified, saving a lot of time and crucial resources.

The integration part is therefore the one that will really give an assessment of the work done and the objectives assumed, as it is here that it will be seen whether this approach can make substantial changes or not.

## Chapter 5

# Low-Code Approach

Low-code development is a new method of programming, which replaces the complexity of writing thousands and thousands of lines of code in favor of less development complexity based on "visual programming." Through this concept, application software can be created through graphical and configuration interfaces. These then allow applications to be created in a simpler way and with significantly reduced time. This is made possible precisely because of such interfaces that offer a "drag-and-drop" mode of development, and then they themselves translate what has been done by the user into source code so that it can be compiled and sent to execution. Thus, the ability to develop applications, even simple ones, becomes within the reach of everyone without the requirement of having a technical background and advanced programming skills.[17]

In today's growing digital marketplace, the low-code approach fits perfectly in three respects:

- Speed. Today, the release of software requires increasingly tight deadlines. Low-code allows this without going to the detriment of efficiency and reliability.
- Enterprise Reliability. Low-code allows minimizing the risk of data loss during software upgrades or possible crashes. This is very important especially for those companies that do constant auditing.
- Complex business logic. Custom logic can be managed while reducing complexity, this is also due to the visibility of the logic itself.

Therefore, low-code is very important in the business environment because it allows you to create applications that are tailored to your business. It also leads to greater ability to talk to one's customers, and greater ability to attract more of them, as it allows one to meet their specific requirements without having to go and change

what is overall software development.

The adoption of the low-code approach will be even greater in the years to come, and this is according to a number of aspects:

- **Distribution of software.** According to a recent report of CIOs, almost all of them believe that software and software updates will need to be released at an increasing speed in the future. Reason being, low-code development may become increasingly central within a company as it allows for standardization of the development approach and reduction of maintenance complexity. Not only that, it also allows you to reduce the possibility of error due to writing lots of lines of code. Thanks to this simplified development, one can also assign tasks to less qualified people and see them accomplished quickly.
- **Visual software development.** Because of its visual approach, the logic behind an application can be understood quickly and easily by any user, from "citizen developer" to senior developer. Through this a variety of user groups can build applications of their choice. Such applications can then be downloaded and started to use within hours. Given the fact that there are not enough developers for the growing demand in the digital market, companies can take advantage of this approach and outsource tasks to less qualified people in order to deliver their projects on time. In addition, thanks to the low-code approach, software can also be tested immediately, which allows them to receive immediate feedback on its operation and any problems.
- **Lifecycle management.** Screenshots that are created with low code development can be started and deployed with a single click. The application is then executed in a few seconds in the browser.

However, there are some common mistakes that people make when thinking about low code. First of all, one should not think that there is no need for a development team. That might be fine for simple applications that are developed through this approach. But not for complex ones that impact business value. Complex applications that have enterprise-wide visibility still need to be integrated with the information systems currently in place, and this requires the presence of a highly skilled team.

Low-code platforms are designed not only for the development of simple applications, but also for major ones. They are indeed able to scale and support thousands of users and datasets.

Also, one should not think that low-code development in a platform is limited only to the basic functionality that that specific platform offers. But these also give the opportunity to create new connectors that allow external services to be integrated and thus allow the application to talk to them.[18]

## 5.1 PingOne DaVinci

PingOne DaVinci is an orchestration platform that enables the creation of user experiences through development-low code approach. Created by the company PingIdentity, the intelligent Identity solution for enterprise, it allows, through a drag-and-drop interface, to model a series of tasks that make up the basic steps for various access management flows.

At the core of PingIdentity’s cloud platform, the orchestrator simplifies the integration and deployment of identity services by going to facilitate the construction of digital paths that allow a user to interact with multiple applications.

PingOne DaVinci, allows dynamic user paths to be designed for any use case within a unified identity fabric. From a single interface, from a single canvas, an entire access management user experience can be identified. It is then possible to model registration, authentication, authorization, verification, risk, fraud, and privileged access flows. It is then possible to immediately test what you have done and integrate those flows, those compositions, within an application in no time.

Andrey Durand, CEO and founder of Ping Identity says that although security in digital services is the first point to respect, nowadays, in order to be competitive in the market you have to put of equal merit also being at the forefront on delivering better digital experiences to customers. He also says that offering frictionless end-user experiences has become the new goal to achieve if you want to beat the competition.[19] Through the orchestrator it is then possible to integrate, in a



**Figure 5.1:** PingOne Cloud Platform

simple way, the various services that PingIdentity offers to perform a series of actions that eventually lead the user to authenticate and have access to the APIs that an application offers. Not only does this allow for the integration of internal services within the company, but because of its interconnections, and its multiple

partnerships, this also allows for the integration of external services, as will be seen below. In fact, Ping DaVinci offers a library of 100+ out-of-the-box connectors for a range of identity, IT, and automation services.

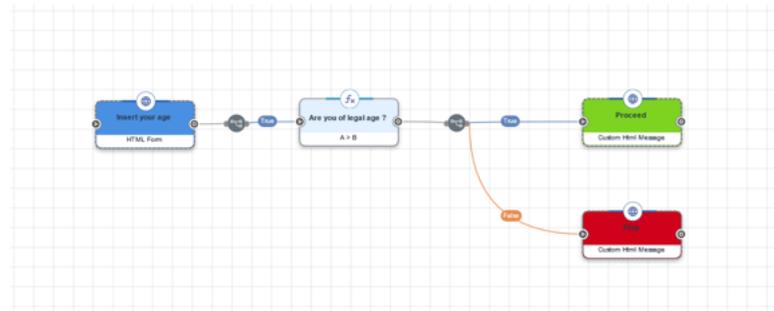
### 5.1.1 PingOne DaVinci Components

At the core of these operations are flows. A flow is a series of interconnected blocks that allows you to create a custom user experience as needed. There are different blocks that have different functionalities. Each block or node can either have purely graphical functionality, visual user interaction, or even communicate something to the backend to say what logic needs to be implemented. Nodes can also refer to external services, thus going to retrieve information from third-party services, change the value of variables, or other parameters. Integration with external services is, however, done, to be secure, through the security standards discussed in previous chapters, such as SAML, OidC, and OAUTH. These nodes are connected to each other through logical operators, which interconnect these nodes based on what the outcome of the action performed on the previous node is, and this determines what path to follow within a flow.

Thus, there are 3 basic foundational components for building a user experience in DaVinci:

- Flow. It represents a user journey that can be registration, authentication or authorization. The flow is composed of nodes and logical operators. The flow starts at the leftmost node and progresses to the right until an error is found or the end of the path is reached.
- Node. It represents the action of a given blocker. This action can result in true, false, or it can return an unexpected error. In case of an unexpected error, the flow stops.
- Logical Operator. This determines the path to follow after a node's action is completed, and the decision is made based on the result returned by the previous node.

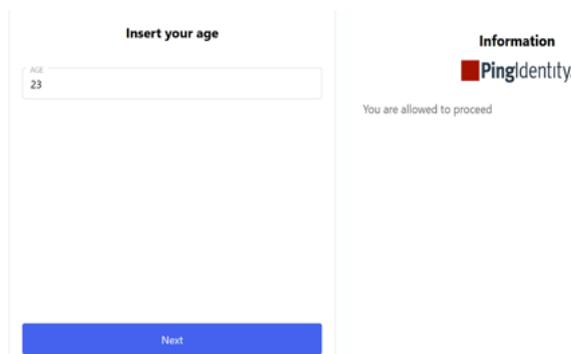
In Fig 5.2 there is an example to try to better understand what we are talking about. This simple flow is intended to perform one action or another based on the age of the interacting user. As previously written, a flow starts from the leftmost block. As for nodes we have 3 nodes that have the functionality of interacting with the user and one that has a logical functionality, which therefore has no visual interaction. The 3 nodes, the blue, the green and the red, are part of the HTTP Form category (one of the standard nodes present in PingOne DaVinci). In the blue one I set the "HTML form" feature, which makes available, without adding



**Figure 5.2:** Example of a DaVinci flow.

any line of code, an html form that allows you to enter your age. In the other 2 a message appears that I set within the block configuration. In that case the configuration is the "Custom HTML Message" configuration.

Then you have the middle block( the Function block, also one of the standard DaVinci blocks) that allows you to do in practice an if statement. In that case I compare the age entered with the number "18". Then you have the logical operators that precisely connect the nodes together. As you can see if the function block returns true then one path will be followed if not another path. The possible outputs of the flow are in Fig 5.3. Or if you enter an age of less than eighteen years (Fig 5.4).



**Figure 5.3:** Output of successful example flow

However, in this simplified flow we do not see the full potential of DaVinci. It could also happen that a flow performs multiple paths in parallel. As in this example (Fig 5.5). As you can see following Google authentication triggers 2 actions, again if Google authentication is successful. On the one hand a token is generated for the user and on the other hand an email is sent to the user. Another feature of DaVinci concerns logical operators. These don't necessarily have to be preceded by a single node and then have to establish the path(s) to follow based on the outcome of a single node. Rather, there can also be several nodes that flow into a logical operator.

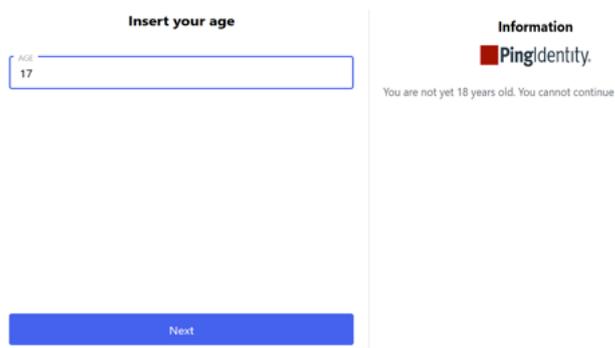


Figure 5.4: Example flow error output

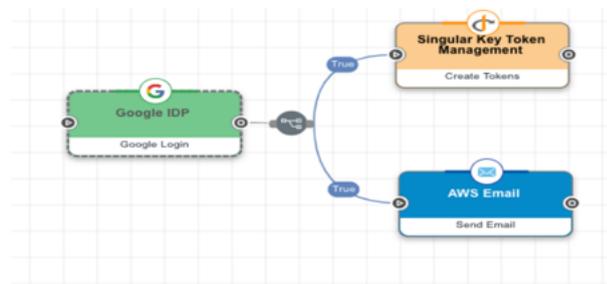


Figure 5.5: Multiple Paths in a flow

As could be seen, no lines of code had to be written to achieve this. To keep track of the values entered you can take advantage, again, of the DaVinci interface. In fact within the configuration of an HTTP block there is the possibility to enter values in a variable and then DaVinci will keep track of them. To retrieve such a variable in a later node you will simply have to select the one you want, as shown in Fig 5.6. To gain more knowledge about the basic use of the DaVinci environment I attach this documentation.[20]



Figure 5.6: Http connector configuration

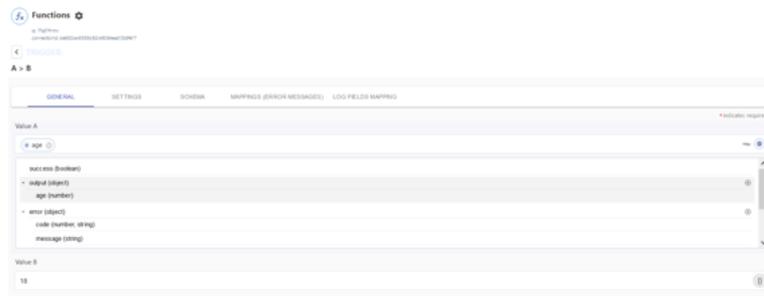


Figure 5.7: Functions Connector Configuration

## Chapter 6

# Access Management flows with orchestrator

Now we are going to analyze how access management flows can be achieved using the PingOne DaVinci environment. In the particular we are going to analyze the registration, authentication and authorization flows. It will analyze the possibilities that DaVinci offers in building these user experiences in simple and linear way.

### 6.1 Registration Process

Normally when we attempt to sign up on a site we are asked to enter our credentials (such as username and password), or even to enter additional information such as First name, Last Name, Date of Birth, etc..

This can be accomplished quickly and easily with the PingOne DaVinci environment by first going to create a credential entry form(username and password) and then going to perform various operations based on what you want to do. With the low-code approach offered by the platform, custom user experiences can be created to suit the company and the customers it intends to attract. And all this can be done in a simple and straightforward way.

The 2 figures(Fig 6.1 and Fig 6.4) show a custom registration flow where the various nodes representing the main steps of a registration flow are present. The flow is divided into the 2 figures for the sake of visibility and clarity for the reader. The first one shows the first part of the registration flow where, proceeding from left to right you have several nodes:

- The first is called Variables. This gives the opportunity to set variables that

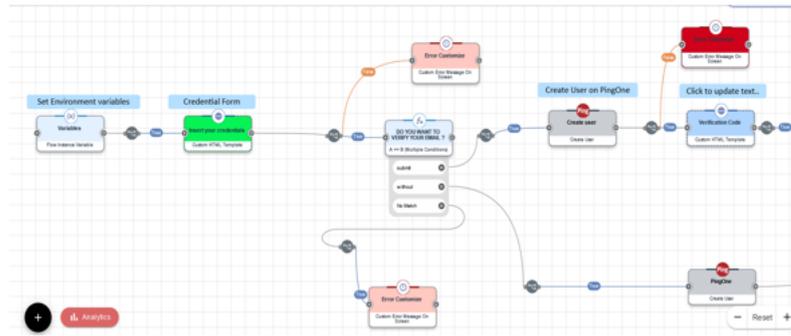


Figure 6.1: First part of registration flow

can be environment and therefore common to all flows in the environment. Or they can be relative to the individual stream and that is how it is used here. The variable that will be set is that of Population Id, since as will be seen shortly it will be used to make it clear to which Population the user created should belong.

- Http form. More precisely in this case we speak of Custom Http Template as the node configuration, slightly different from the one seen in the previous chapter. In this case you have the option of adding html within the node to customize the page where you enter the credentials, as shown in the Fig 6.2.

Figure 6.2: Custom Http form for credentials

- The function block. Already seen in the previous chapter. In this case it is used to check which button the user clicked in the previous step. If he chose

to continue without verification email or even by having an email arrive on the entered email address. Note how if there is no match with the possible values of the button variable (which has as its value what was chosen by the user) then there is an error and the stream must stop.

- The PingOne block. This is where the core of this flow and the potential of DaVinci lies. What has been chosen to implement here is to have PingOne Identity as the IdP and then go and leverage the company's own service integration. Basically PingOne DaVinci passes information to PingOne which stores within its own database the user's information. Then, as will be seen in the login flow this information will be retrieved through the same node, but with different configuration. Here the configuration chosen is "Create User." Within the node there is then a need to go and enter username, password and population Id (which are easily retrieved, with a simple click, as variables saved by DaVinci and set in previous nodes). Another advantage, is that each block, even the non-standard ones, offer a multitude of possible configurations, and you do not have to write and/or rewrite any lines of code to choose them. Another thing you may notice is the fact that when you enter a password that does not meet the necessary security requirements, this is directly communicated to us through the output of this block. And even these security policies are not set in DaVinci but rather dependent on the IdP being used, and therefore retrievable in an immediate way by the orchestrator, that is, through the addition of an "Error Customize" node that has as its message precisely the output of the previous node, in this case that of PingOne.

Sometimes proposing a registration flow where a lot of information is asked of the user causes the user to abandon the process on a given website. Not only that, but having to enter credentials may also result in the user not finding a positive response and preferring to veer elsewhere where registration is not required, as it is explained in this article.[21] If, however, you want the user to log in to a site wanting to keep track of it but without it entering credentials, you can have the user log in with a certain provider and retrieve the information from there. The potential of DaVinci, even compared to other competitors offering low-code platforms, is that it has interconnections with a great many partners. This allows, referring to the stream in question, to be able to switch the IdP from PingOne to any other, which may be Google, Facebook, Slack and many others. To achieve this you just need to have an application on the Providers just mentioned and go and enter the credentials of that application into the configuration of the chosen node as can be noted in the following figure. Note how also the redirect uri is established by DaVinci and you then have to copy it within the application you are talking to on Google. Now let's move on to the second part of this canvas representing registration flow (Fig 6.4). As can be seen if the user previously chose



Figure 6.3: Google connector configuration

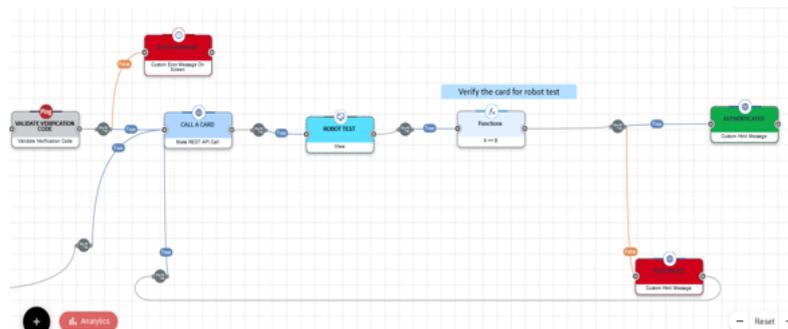


Figure 6.4: Second part of the registration flow

to continue without the verification email this will be registered on the PingOne site and then they will have to do the robot test directly. Whereas if the user had chosen the email verification option, this will receive the code on the email, enter it within a form made available in the figure 32. Then there are other blocks.

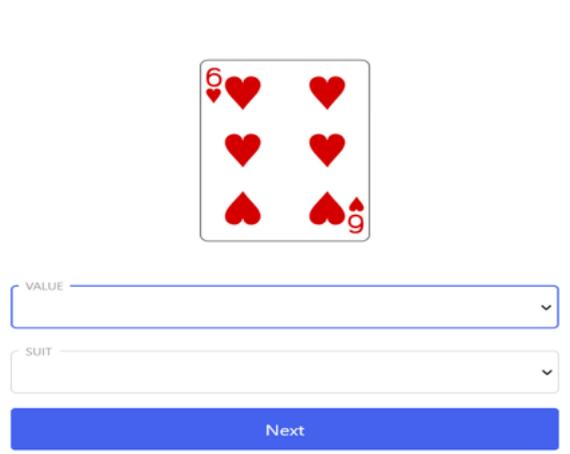
- Validate Verification Code. Another option made available by the PingOne node. Inside you will need to enter the username of the user in question and the code entered in the previous HTTP node. The verification is internal to the IdP and the response then will be available to subsequent nodes. If there is an error then it will be reported appropriately if not then it will proceed forward.

Now there are 2 blocks that perform a custom robot test in which a card is shown and the user has to prove that they can recognize that card.

- Make a rest API call. This type of HTTP block configuration allows a call to API. You then perform a call to a public API. The url is "https://www.deckofcardsapi.com/api" api was taken from the following site.[22]  
Here for example I chose to use the api which allows you to draw one card from the deck, but you can also draw multiple cards. This node receives an

output in JSON format, from that then it is possible to retrieve the values for the url of the image, the value of the card and the corresponding suit, which will be used for the next block.

- Screen Connector. Display forms and customized UI to retrieve information from a user or show flow progress. In this case it is used to show the card drawn from the deck and perform the robot test as seen in Fig 6.5.



**Figure 6.5:** Example of screen connector output

If the test is successful then you are finally registered on the application within which you are going to integrate this stream. If not, if the test fails as can be seen in the figure, there is a loop between the blocks from which you exit only when the test is performed correctly.

## 6.2 Authentication Process

In the area of access management, after registration we talk about access to an application, and then authentication. This section will look precisely at the sign-in process. This will be done through a flow, created on the orchestrator, that allows a user to authenticate to a Provider through the use of credentials. Then in the next chapter we will see how to use this flow in a practical setting by going to consider DaVinci as an Identity Provider and using the OidC protocol. Also for this flow we'll see two pictures, two parts, for convenience in reading the flow( Fig 6.6 and Fig 6.7). The flow starts with the custom template, one of the configurations of the http form, connector already seen earlier for the registration flow. This time the custom template allows you to enter your credentials for authentication and

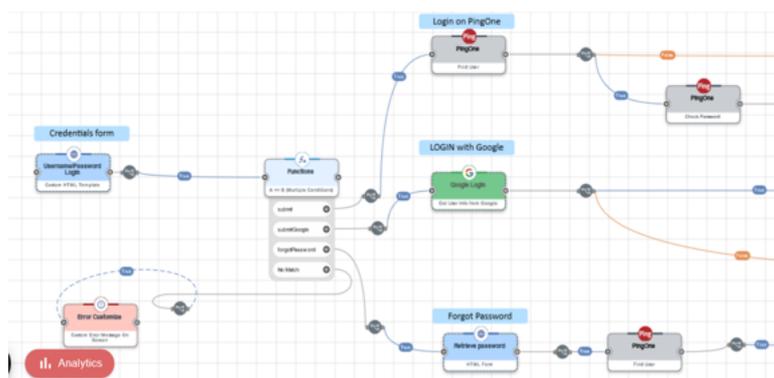


Figure 6.6: First part of login flow

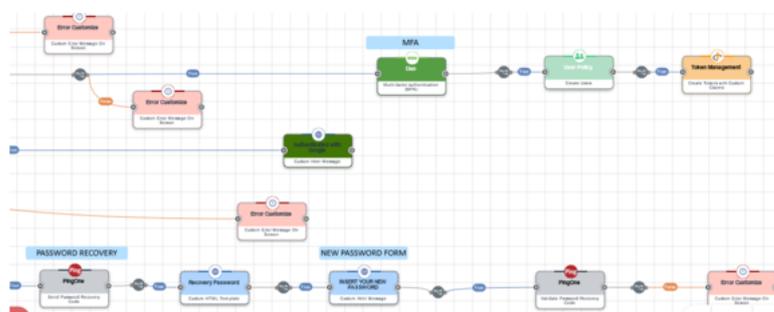


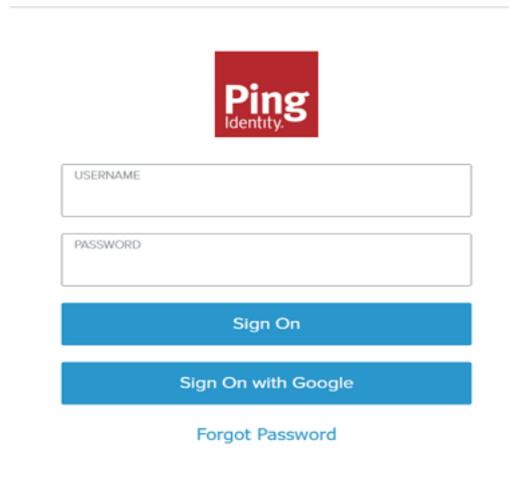
Figure 6.7: Second Part of login flow

then choose whether you want to authenticate with PingOne or Google. Or if you want to recover your password as you forgot it for PingOne. This is shown in Fig 6.8 . Obviously it is assumed that on both providers, whether it is Google or PingOne you already have a registered account. After the connector that allows you to enter credentials there is the function connector, also seen earlier, in which you go evaluate the value you entered inside the button variable. Here again you can see an advantage of DaVinci. As soon as the Http Form connector is created, with the custom Template configuration, automatically the platform has created a variable, called "button," to which the value is associated according to the option chosen by the user for authentication. And then the function node goes to evaluate which path to follow based on the value of the variable.

So there can be 3 possible paths to follow, that of authentication on PingOne, that of authentication via Google, and that of password recovery.

Starting to analyze the first available path, we notice some connectors:

- The first is PingOne, previously seen for the recording flow, but this time with

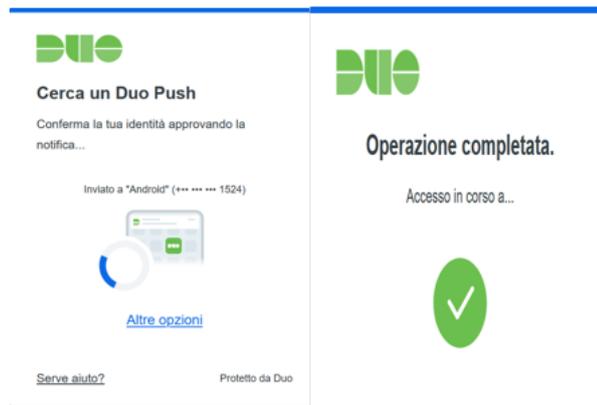


**Figure 6.8:** Login Form

a different configuration. In fact, here we are going to look for the user, if it exists, so if there is actually in the PingOne database, a user related to the email entered, which will then act as the primary key within the DB. If it is present then you will continue on the "true" path, if not you will continue on the "false" path going to show an error on the screen thanks to the "Error Customize" connector, as you can see in Fig 6.7.

- If the user has been found then a password check will take place. Other configuration always of the PingOne connector. It does a password check on the password loaded in the HTTP form and the one stored on the PingOne DB. This connector returns true if the check is valid, otherwise an error message will appear.
- If you continue on the true path, you will encounter a new connector. It is the Duo Security one that enables multi-factor authentication. To do this all you have to do is add an appropriate connector, precisely the "Duo connector" and configure it by going to enter in the settings of the connector the Client Id, Client Secret and Api hostname of an application present on Duo. To do this I followed the following documentation.<sup>[23]</sup>

What implements this block is visible in Fig 6.9. Basically once the user is registered on the Duo Security application this, he chooses the method by which he wants to perform the MFA. Here it was chosen to receive a notification on a mobile app on their smartphone, but there was also the option of sending a code to a previously registered phone number. So you can see how to have MFA all you have to do is add a simple connector. This is because DaVinci supports a lot of partners, to which one can connect quickly



**Figure 6.9:** Duo security example

using the connectors provided by the platform. Here we have chosen to use Duo Security, but it is just one of many partners that allows MFA to be done in a DaVinci flow. MFA can also be done using a connector from PingOne, called the PingOne MFA connector for short. In practice in that case you are going to integrate an in-house service. The PingOne MFA connector supports all of PingOne MFA's authentication methods including more secure methods like Mobile SDK, FIDO bound biometrics, and security keys, generic TOTP Apps (e.g. Google authenticator), as well as traditional sending of OTP (one-time passcode) by SMS, voice, and email. Even to set up an MFA connector is not a time-consuming task. It is possible to set it up by following this documentation.[24]

Once the MFA is done then this path ends here' as the authentication is successful. As seen also to build a successful authentication flow is a quick process that does not require developer skills. Now we move on to the second path available within the flow, which is the Social Login path.

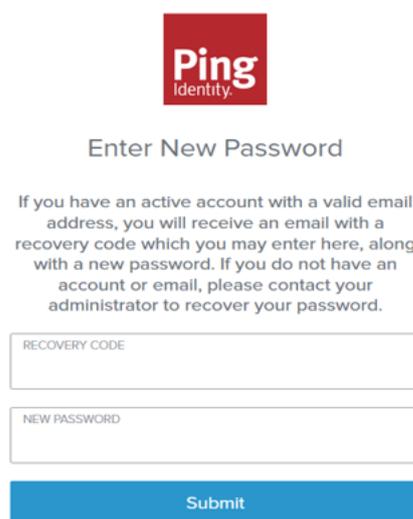
- After the function block, as seen in Fig 6.6 you have the Google login connector, which precisely allows you to authenticate through Google. Here again we take advantage of the partnership that PingOne has with Google, so that you then have to take a few steps to set up the connector properly. You have to create a Google Api app, take the secret client and ID client of that app and put them into the block setting. Not only that, in order for Google to then know where to go back to in the service provider, in this case PingOne DaVinci, you have to go into the Google app and set up the redirect URI. For more information on exactly how to go about this step, I consulted that documentation.[25]

So through this simple mechanism, when you use this connector you set up

the OpenId Connect protocol between DaVinci and Google where the former plays the role of Client and the latter plays the role of OpenId Provider. Not only Google, but also here' you can take advantage of the many partnerships PingIdentity has with other companies so that you can do social login with other connectors. For example, one could use Facebook, Slack, Amazon, and many others. The process is similar, and to achieve this you can always consult the respective section in the documentation.

- If the Google Login Connector is unsuccessful then an error is shown on the screen, otherwise this comes back with an Id token and is authenticated. In the flow then follows an HTTP message block showing the authentication message.

Now we move on to analyze the third possible path within the flow. That of password recovery. First there is an http form block, where you can enter your username. After that you find another connector, already seen earlier, which goes to verify on the PingOne DB if indeed such a username exists. If the verification is successful, a connector, again of the PingOne type, is used, this time set as "Send Recovery password." A code is sent to the entered email address. It then finds then a connector again of type HTTP, of type Custom HTML Template, into which it enters the code received by email and the new password, as it can be seen in Fig 43. Finally there is a last PingOne connector, configured as "Validate Password recovery Code," which verifies the code entered (if it is the same as the one emailed), and if correct then it will store the new password for the user. All



**Ping**  
Identity

**Enter New Password**

If you have an active account with a valid email address, you will receive an email with a recovery code which you may enter here, along with a new password. If you do not have an account or email, please contact your administrator to recover your password.

RECOVERY CODE

NEW PASSWORD

Submit

**Figure 6.10:** Example of Recovery code form

of these PingOne-type connectors, that we have seen both in registration and in

login process, allow operations with the PingOne SSo service, where the utilities used in these flows are present. They thus give the ability to make access to the PingOne API. To do this, it was necessary to create a worker application [26] on PingOne and assign roles to it. A worker application is an application that allows administration roles to be performed without direct human intervention. The roles that are assigned to the application determine what kind of actions that application, on behalf of the user using it, can perform. In this case it was needed to add “Identity Data Admin Role” and “Environment role” to enable the application to do requested tasks.

After that, configure the PingOne connector with Client Id and Client secret of this application.

Another advantage of DaVinci is that once the Client Id and Client secret are set within a PingOne connector, automatically all other PingOne connectors are also automatically set with the same credentials as the Client in question, without having to go and configure all such nodes one by one.

### **6.3 Authorization Process**

To implement an authorization process, it was decided to leverage the integration of a PingIdentity service, called PingOne Authorize. This service allows authorization decisions to be made to access services and/or data. This allows decisions to be made centrally by going to set policies that evaluate identity attributes, entitlements, and other context information. All of this allows for going to simplify the decision processes within a company, as it will no longer rely on "hard-baked" access control by teams. PingOne Authorize provides a trust framework to which those leveraging this service can pass attributes on which policies will be based. Going to use this service, one can then integrate it within a PingOne DaVinci flow through the appropriate block provided by the platform. To do this, it is first necessary to go set up within the training environment the PingOne Authorize service as well. To get a complete overview of the service offered by PingOne Authorize I recommend reading this document.[27]

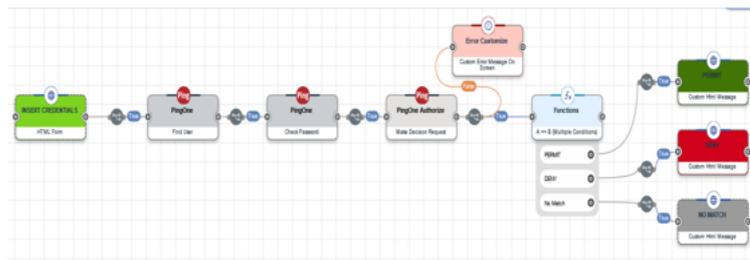
What is then done is to go and create an attribute in PingOne Authorize. Based on the value of that attribute a real-time decision will be made within the DaVinci flow. The attribute in question is the ID of the user once it has authenticated within PingOne, and that is assigned to it by PingOne itself.

First you have to have a worker application on the PingOne environment. The one that you have already used for the registration and authentication flows is leveraged. After that you have to create the PingID attribute on PingOne Authorize.

Finally still on the PingOne Authorize environment you have to go and create a policy associated with that attribute. The service provides various operations that allow you to check whether the attribute in question reflects the requirements for which the policy is met or not. Finally, one has to figure out what to do when the requirement or requirements of the policy are matched, whether to allow the next action or deny it.

Once the policy has been created to make it available and usable we need to make it public on one of the existing endpoints for the PingOne Authorize service.

We now turn to analyze the DaVinci flow in question(Fig 6.11). As it is possible to



**Figure 6.11:** Authorization flow with PingOne Authorize

see from Fig 6.11 first we repeat the blocks that allow the authentication of a user on PingOne. This is due to the fact that authorization succeeds the authentication process and it would not even make sense to talk about it if not in this logical order.

Then the new block that is found is just the PingOne Authorize block, which was configured with Client id and Client Secret of the worker application and also the url of the endpoint on which the policy was published to PingOne Authorize. Within that block you go right to set the value you want to give to the attribute on PingOne Authorize protected by the policy in question. This blocker will give either "PERMIT" or "DENY" as the response.

Then we find a function block that based on the result of the policy, goes to show on the screen, just by way of example the result. Obviously in a practical application you go to exploit the "PERMIT" output of the function block to go and perform the desired action that you are authorized to perform.

# Chapter 7

## Use Cases

This chapter will show use cases of integrating flows within an application. First, it will be shown how to integrate flows developed on the DaVinci environment within a single-page application. Then in a second use case it will be shown how to easily integrate a DaVinci authentication flow within an application that allows access to the PingIdentity environment administration console. DaVinci supports 2 main methods of integrating a flow: the Widget Method, and the Redirect Method. These methods will be used in the first and second use cases, respectively.

### 7.1 Integration in a Single Page Application

Let us see how a flow can be integrated within a Single-Page-Application. First of all, you need to have an application. To do this, the application was developed on a server application, called "Glitch".[28]

Many applications can be developed within this environment, from a pure frontend application to an application that has also a backend part. For simplicity for such a use case, a client-side single-page-application was preferred since the goal is to show how to start a flow outside the DaVinci testing environment. To do this, a series of steps had to be taken. Specify that through the widget method used for embedding the flow in this use case you are going to embed the flow within a modal that opens when you click the button present in the application.

- Create the flow that you want to integrate within the single-page-application. This was seen in the previous chapter. One must always first test the flow in the testing environment. Testing a flow in the DaVinci environment is extremely easy in that you just save, deploy, and with a simple click go test the flow. This is another great advantage that the platform offers in that even if you want to make a change, in a few seconds you can go and test the effect of the change on the flow.

- You obviously need to have a single-page-application. To do this, an application was used, which takes the poker game theme, where there are 2 buttons, one for login and one for registration as is visible in Fig 7.1. To make this single page

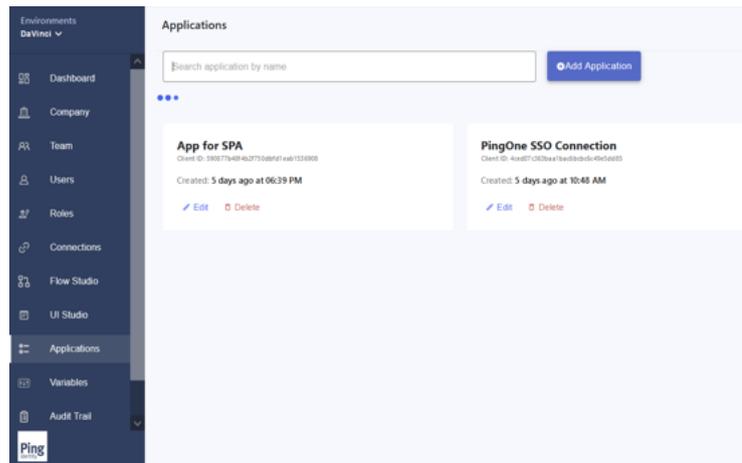


**Figure 7.1:** Screen of Single Page Application on Glitch

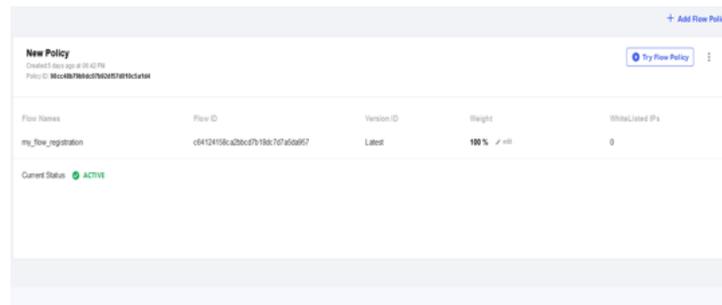
application, CSS found within the Glitch application was used. Here we must make a point and go to note something that may represent a disadvantage for DaVinci. In the previous chapter, when streams were developed and tested, a linked CSS file was always used in the stream settings within the DaVinci environment. When going to integrate a flow within an application, the delegation of the look and style must be in charge of the calling application. This allows the design team, in case of changes, to not have to go and change something in the testing environment, but within their app. This is because, flows developed in the testing environment, can be used and embedded within multiple applications, so it is good practice that the styling and management of this is relative to the calling app rather than the flow, which let's say should represent more of the logical part.

For this reason, so when integrating a flow within an APP, you must not have any CSS linked, neither within the flow settings, nor even within display blocks, such as the HTTP and SCREEN blocks can be.

- After creating a single-page-application you must create an application on the DaVinci environment that acts as a gateway for the flow you choose to integrate. To do this simply go to the PingOne DaVinci environment and press the "Create Application" button as can be seen in Figure 45. Then going into the configuration settings you can see several fields, which are necessary for the integration to work and also the endpoints that are used to access the flow in question.
- After creating the app on DaVinci we need to set the desired flow as the flow policy as in the respective section (Fig 7.3). Once this is done we need to go to



**Figure 7.2:** How to create an App on DaVinci



**Figure 7.3:** Attaching a flow in an application

the single page application and enter some parameters inside a Javascript file that can be found inside the following documentation.[28]

In particular, we need to go pass the parameters of Environment Id, API Key , Policy Id, which can be retrieved on the DaVinci environment, as visible from Figures 45 and 46. Not only that but within the single page application then it was necessary to create an html element contain the widget so that when the button is clicked the flow is invoked. You can see, in the figure below what is the parameterized script that allows you to go and call the DaVinci flow. From a logical point of view, what happens is first a call to the sdkToken endpoint of DaVinci . In fact, thanks to the retrieval of the API key, from the test environment, the application makes the following call:

“GET https://api.singularkey.com/v1/  
company/039c7a35-8a3e-4173-b90e-d601bbccd635/sdkToken”

```

function loadWidget(policyId, renderComponent) {
  const tokenURL = "https://api.singularex.com";
  const apiUrl = "https://auth.pingone.com";
  const companyId = "039c7a35-8a3e-4173-b90e-d601bbccd635";
  const skApiKey =
    "50097b73721a51c4378e79b6796fa7fc2c0b0c448b782af1c296d93b2b0b62d6e9812f083e5f037d09f9ca5963e56e11acc838aa91133821888d2e792917b5d5dbcb8899c4326a23a7c277d6caae7d70d1b1703e464953b3";
  const skAccessToken =
    tokenURL + "/api/company/" + companyId + "/sk/token";
  console.log(skAccessToken);
  // Add the API Key from your Davinci Application.
  var headers = new Headers();
  headers.append("X-SK-API-KEY", skApiKey);

  var requestOptions = {
    method: "GET",
    headers: headers,
    redirect: "follow",
  };
  // Retrieve Davinci Token
  fetch(skAccessToken, requestOptions)
    .then((response) => response.json())
    .then((responseData) => {
      var props = {
        widget: {
          method: "startFlow",
          endpoint: apiUrl,
          accessToken: responseData.access_token,
          companyId: companyId,
          policyId: policyId,
        },
        onSuccess: false,
        successCallback:
          successCallback,
        errorCallback:
          errorCallback,
      };
      // Invoke Davinci Widget
      davinci.showWidget({
        document: getLatestByUrl(renderComponent),
        props
      });
    });
}

```

**Figure 7.4:** Script for integrate a flow in an SPA

In response to this request DaVinci issues an access token to the application, which will later use it to make a request to access the stream. Here the request via the Post method, that includes the access token:

“POST <https://auth.pingone.com/039c7a35-8a3e-4173-b90e-d601bbccd635/davinci/policy/98cc48b79b9dc07b92df57d010c5a1d4/start>”

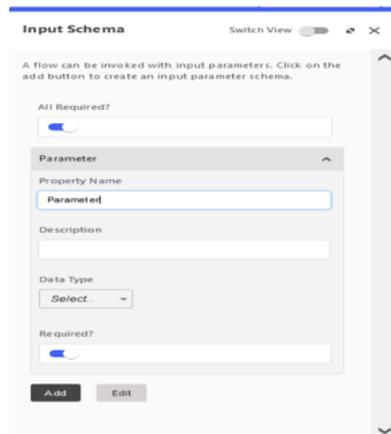
In fact When you use the widget method to integrate DaVinci flow policies into your application, the /start request uses the resource server endpoint “auth.pingone.com”.

After that, you go test the application and you see that when you click the record button it opens the initial screen of the chosen stream. Then going forward in the flow you execute the logic exactly as you do within the testing environment.

Through such a method not only can the flow be started, but a flow can also retrieve values from a present file within the application, so DaVinci can accelerate development when integrating with backend services and APIs, enriching the overall user experience. To do this one has to make use of 2 important nodes, already seen earlier which are the Variables one, where inside one can set the URL of the file from which one wants to retrieve information . The second block that then allows you to send data to the application and the HTTP block, in "Send Success Response" configuration that allows you to send some JSON to the application. The latter must have a method to call the stream, and link that method to the webpage Onload event.

Another advantage of DaVinci is that a given flow can also receive input parameters, even before it starts, from the application that integrates it. To do this, a series of steps had to be followed:

- First in the settings of the flow you have to go to the "Input Schema" section (Fig 7.5), where you have to set the desired parameter as "required." That way if the flow is called without the input of that parameter then an error will be shown.



**Figure 7.5:** Input schema for a DaVinci flow

- • In the application, you must also pass the parameter in question to the loadwidget function, seen earlier (Fig 7.4). Within that function it is then necessary, in the "fetch" to add as a property also that parameter.

As noted then also passing parameters, to a DaVinci flow, becomes something very simple to accomplish.

Moreover, DaVinci offers the advantage of being able to perform A/B testing quickly.[29]

This way you can test the new features and changes on a subset of users and see if it increases or decreases user abandonment. DaVinci allows us to granularly define flow policies to serve up different flows or flow versions based on a distribution percentage or IP whitelisting. This is achieved very easily. Just perform 2 simple steps:

- Do the export of the stream in question. Create a new one by doing the import of the source stream. Then you go in and change what you want to change, and this, as seen so far requires very little effort, after you get a little familiar with the orchestrator.

- You then need to go to the DaVinci environment and select the application that you had previously chosen. Go to the flow policy section. Click on the policy that you had previously and in addition to the flow that was already there also attach the new one, as shown in Fig 7.6.

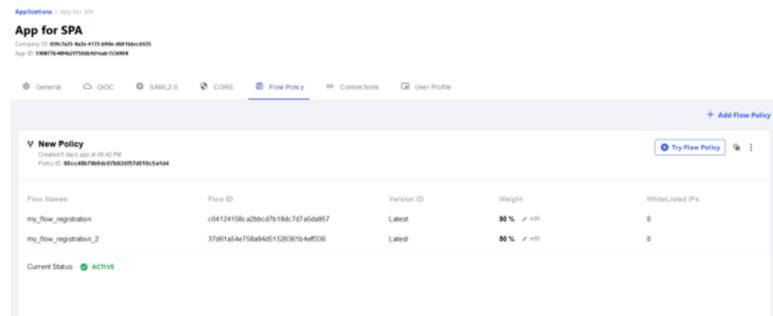


Figure 7.6: How to perform A/B testing

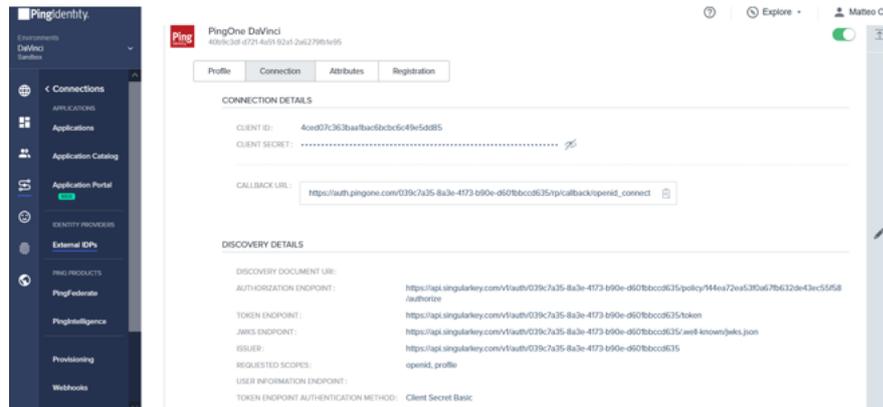
You can see how you can perform in a short time, going to save important resources and time for a company during software development, going both to test a flow and perform A/B testing.

## 7.2 Integration with Redirect Method

In this second use case, I used the "Redirect" method to integrate a DaVinci flow within an application. Through this method the flow is no longer shown within a modal that opens on the page but new application page. For this example, I integrated the authentication flow, explained in Chapter 6, within an existing application in the PingIdentity training environment, which is also the one that allows access to the environment's administration console. Basically what has been accomplished is to have an application in PingOne SSO acting as a Service Provider and an application in DaVinci acting as an Identity Provider.

These communicate using the OpenId Connect protocol. To do this , the following steps were performed:

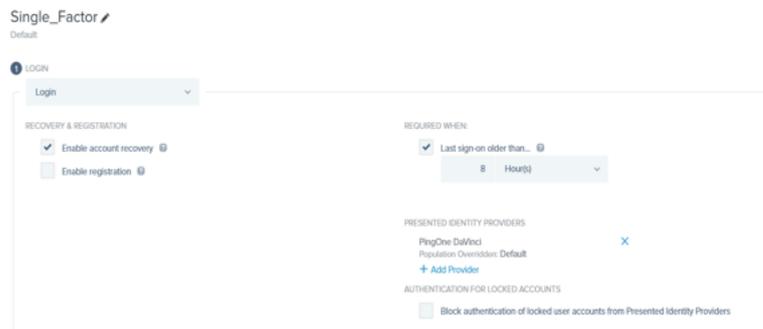
- Have a flow in the DaVinci environment. This time as an example flow we will use, the one named "my-flow-login," explained in Chapter 6(Fig 6.6 and 6.7).
- Have an application in PingOne DaVinci, which as in the previous case,has as flow policy a flow, in this case the one "my-flow-login".
- Connect that application as an external Idp in PingIdentity and set the correct endpoints. This is shown in Fig 7.7. Information for correctly setting these



**Figure 7.7:** Configuration of an external Idp in PingOne SSO

fields was taken from the application present in PingOne DaVinci. Information was taken regarding the client Id, client secret, and the various endpoints of PingOne DaVinci.

- After creating an external IdP in PingOne you need to go and add it as an authentication policy. This is done within a section of the PingOne environment. Specifically, I added that external Identity Provider in the authentication policy called "Single-Factor," which is also the default policy for applications in the environment. It can be seen in Fig 7.8.
- You need to have an application that hides resources behind it and allows you to do authentication to access those resources. To do this, as mentioned earlier , I leveraged an existing application in PingOne SSO that allows you to do console access.



**Figure 7.8:** AuthN Policy

After integrating PingOne DaVinci as an Idp, we can see from Fig 7.9, how indeed it is possible to make access through both credentials and PingOne DaVinci. Since

**Figure 7.9:** PingOne SSo SP and DaVinci as Idp

the second case is more interesting, let us analyze it from a logical point of view, following what are the HTTP messages that the applications exchange (Fig 7.10 and Fig 7.11) . They refer to the OpenId Connect protocol , specifically the Authorization Code Flow, explained in Chapter 3. A user who wants to connect

```

▼ GET
Scheme: https
Host: api.singularkey.com
Filename: /v1/auth/039c7a35-8a3e-4173-b90e-d601bbccd635/policy/739f4c61528a9fdbab24ee222b1e712/authorize

client_id: 4ced07c363baa1bac6bcb6c49e5dd85
redirect_uri: https://auth.pingone.com/039c7a35-8a3e-4173-b90e-d601bbccd635/rp/callback/openid_connect
state: 00becdfb-b8c6-40ac-9b4b-ae5b1b8971ec
response_type: code
scope: openid profile
nonce: 00becdfb-b8c6-40ac-9b4b-ae5b1b8971ec

```

**Figure 7.10:** Oidc call to SP in HTTP message

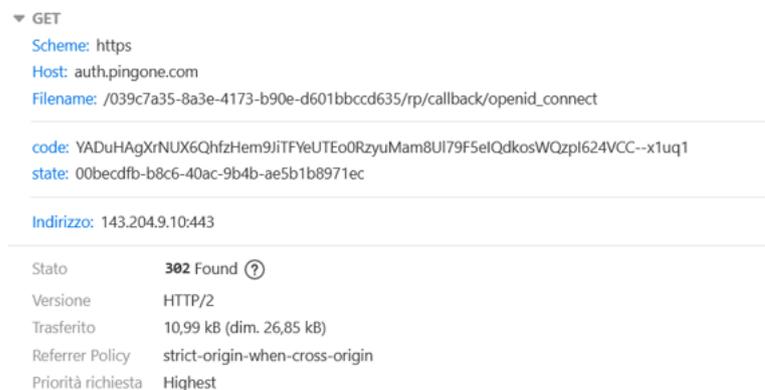
to the administration console on PingOne SSo, decides to do so through PingOne DaVinci, then this is redirected to the chosen identity provider. This can be seen through a call (Fig 7.10) to the very application on DaVinci. Indeed, we note that the client Id is that of the app on DaVinci and the policy

Id is that of the flow that was set as the flow policy in the app. In order to do this, however, it was necessary to set in PingOne SSO, in the "External Idp" section, the authorize endpoint, to which to go and send this request. To set it correctly, it was necessary just to enter the values of client id and policy id (Fig 7.7).

Then downstream of this request, if all goes well, there will be an authorization code in the response.

That authorization code will be used (Fig 7.11) when you are redirected back to the service provider, thanks to the redirect uri.

Note also the "state" parameter that keeps track of the OpenId connect execution flow. It is a kind of "nonce," used to prevent some cyber attacks, such as the "reply attack." Then, as per the protocol, this authorization code will be used, by PingOne



**Figure 7.11:** Oidc- Call to Sp with AuthZ code

SSO, to retrieve information about the user, and then exchange it and get a Json Web Token detached from DaVinci, inside which is user information, which allows Single Sign On authentication to the user.

To exchange the authorization code for a token, the call is made to the token endpoint. The various endpoints were set manually on the Service Provider (Fig 7.7).

Returning to the DaVinci flow in question (Fig 3.7), to implement the openid connect protocol, fundamental importance acquires the final connector of that flow, namely the Token Management one. This gives us the possibility, once the user is authenticated, to create a token with his information. Like almost all connectors, this one offers multiple possible configurations. In this case, the "Custom Claims with Redirect" configuration was chosen. Thanks to this it is not only possible to create custom tokens, without adding any line of code, in fact it is only necessary

to enter the desired variables inside the space provided, and then this will be translated into code. But an important advantage that this configuration offers is that it performs the redirect to the caller without setting any redirect uri, as was the case, for example, when Google Login and Duo Security blocks were used. In fact in such cases, they were the ones taking the role of Identity Provider, and needed within their applications, to set the redirect uri to go back to the Service Provider.

# Chapter 8

## Conclusions

The work of this thesis focused on the possibility of developing typical access management flows through the approach called "low-code "and studying its advantages.

First then, it looked at how access management works and what are the strengths in having an access control system. In that part then, we looked at what are the main features of a software access manager such as access policies, cookie management for a given user session, main authentication methods for a service, multi-factor authentication, and role management for users accessing a given service. Then going deeper, the concepts of Identity federation, delegated Authorization, Single-Sign-On ( including the concept of social login), the concept of API and API security were studied. To implement the federations, the protocols that underlie this such as SAML, OpenId-Connect and Oauth were studied and used.

After seeing these concepts by going to use a "standard" method, the concept of the low-code approach was seen and the benefits it can bring were studied. The flows that characterize accesses to services and resources were then implemented on a platform that exploits the concept of the low-code approach, made available by PingIdentity, namely PingOne DaVinci. It was seen what the main features of this software are and how to implement customized flows that meet the security requirements of the protocols used.

The objectives to be achieved in this part concerned the benefits that could be found, especially in terms of saving time and resources, in the construction and testing phase of the various flows. From what also emerged in the previous chapters, the results were excellent.

For the registration flow, it was noted that setting up both the front-end part and the logic behind it is much simpler than a traditional approach. One can make use

of multiple identity providers by simply learning how to set up a blockchain that allows one to connect to it. All this is handled directly by the Low-Code platform. This also applies to the verification validation of an e-mail. Also for the front-end part, there is the possibility of re-setting templates of already existing code. This is not a great advantage compared to a traditional approach there, where there is a great reuse of already existing code anyway.

For the authentication flow, it was noted that it is possible to go and choose several identity providers to choose the desired authentication method by setting up federations, where the different applications are able to exchange information with each other. All this is done without writing lines of code and with relative speed. Integration between the various methods is not a problem, since this platform has connections to more than a hundred different external services. It has also been discussed in the previous chapters, how one can set up several and more Multi-Factor-Authentication methods in a simple way and without the end user being aware of it during his user-journey. It was also noted how the management of variables is no longer a concern of the developer, but everything is managed by the LCAP platform (in this case PingOne DaVinci).

In the authorisation process, it was seen how it is possible to set policies, and above all to be able to make decisions even on several attributes.

Remaining in the training environment, it was seen how the testing of such flows is much reduced compared to a traditional approach. There, one has to compile and execute code. Not only that, most of the time, there is also the need to use third-party libraries. In that case, it is enough to launch the stream on the browser with a simple click.

It was also noted that, thanks to the services offered by PingIdentity, these can be integrated quite easily within a DaVinci flow. Thanks to this fact, the user does not need to create complicated customised logic to achieve his purpose. He simply takes advantage of the services offered. Even when another person wants to read the logic of the constructed flow, not only is the visual approach more visible, but thanks to the integration of these services with simple blocks, greater clarity also follows. This also allows for a distinct division of roles. For example, there are those who can take care of writing and setting policies, and then those who develop the flow can use them without having to go into the details of them.

To validate the study of this approach and implementations of these flows, the latter were integrated within 2 applications.

The first integration was done within a simple Single-Page-Application. The second within a pre-existing application that allows access to a PingOne training environment and thus access the services made available. Here, too, the results are considered positive. In the integration of a first application, a single-page, we went to see how the use of a flow leads to the addition of a longer or shorter script. This also depends on the parameters one wishes to pass to the stream when calling it. This integration causes the stream to be displayed via a widget that opens in the window. What was relevant is the fact that an application can simply integrate these developed streams without worrying about the logic of them, which is handled instead by the LCAP platform.

For the second integration, it could be seen that integrating a flow within a pre-existing application was not complicated, nor did the application itself have to be modified. Instead, the pre-existing application and the application gateway to the flow were merged into an OIdc and SAML federation. Here too, the difference on the user side did not cause any friction.

For possible future developments there is the possibility of implementing even more advanced concepts such as risk-based registration and authentication and even dynamic authorization by going more and more to integrate the services that are offered by PingIdentity or external services to which the company is connected. The low-code approach will be increasingly used in the future and in application development. In fact, research by "Gartner" determined that the number of applications that will be developed using the low-code approach will be more than 65 percent.[30] It also highlights which are the main platforms that allow the low-code approach to be exploited nowadays, and what are their strengths. This will enable companies to be more and more in step with the growing market demand for innovation and continuous creation of different services. This will be possible because of the main feature of the low-code approach, which is that they do not have to have people with extensive technical backgrounds to develop applications. This will allow experts to concentrate on something else, and the speed of production will be greater.

# Bibliography

- [1] Froehlic. *Why Identity and Access Management? Guide to IAM*. URL: <https://www.techtarget.com/searchsecurity/definition/identity-access-management-IAM-system.html> (cit. on p. 1).
- [2] Paolo Tarsistano. *IAM, perchè passa da qui la nuova sicurezza dei dati aziendali*. URL: <https://www.cybersecurity360.it/soluzioni-aziendali/identity-and-access-management-iam-perche-passa-da-qui-la-nuova-sicurezza-dei-dati-aziendali.html> (cit. on p. 2).
- [3] Flavio Biscaldi. *HTTP: cos'è e come funziona*. URL: <https://www.flaviobiscaldi.it/blog/protocollo-http-cosa-e-come-funziona> (cit. on p. 9).
- [4] Redhat.com. *What is an API?* URL: <https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces> (cit. on p. 10).
- [5] Redhat.com. *API security*. URL: <https://www.redhat.com/en/topics/security/api-security#why-is-api-security-important> (cit. on p. 13).
- [6] Netiq.com. *Access Manager Overview*. URL: <https://www.netiq.com/documentation/access-manager-45/product-overview/data/product-overview.html> (cit. on p. 15).
- [7] *RFC-261*. URL: <https://datatracker.ietf.org/doc/html/rfc2617> (cit. on p. 22).
- [8] Geeksforgeeks.org. *Message digest in Information security*. URL: <https://www.geeksforgeeks.org/message-digest-in-information-security/> (cit. on p. 23).
- [9] BeyondIdentity.com. *What is Dictionary Attack ?* URL: <https://www.beyondidentity.com/glossary/dictionary-attack> (cit. on p. 23).
- [10] Pendyala Shim Bhalla. *Federated Identity Management*. URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1556498> (cit. on p. 24).
- [11] Philpott Ragouzis Hughes. *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. URL: <http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html> (cit. on p. 32).

- 
- [12] Bradley Sakimura. *OpenIdConnect Core 1.0*. URL: [https://openid.net/specs/openid-connect-core-1\\_0-final.html](https://openid.net/specs/openid-connect-core-1_0-final.html) (cit. on p. 36).
- [13] Styra.com. *What You Need to Know About Fine-Grained vs. Coarse-Grained Authorization*. URL: <https://www.styra.com/blog/fine-grained-vs-coarse-grained-authorization/> (cit. on p. 38).
- [14] Infosecinstitute.com. *Risk associated with cookies*. URL: [URL:https://resources.infosecinstitute.com/topic/risk-associated-cookies/](https://resources.infosecinstitute.com/topic/risk-associated-cookies/) (cit. on p. 39).
- [15] Antonio Sanso Justin Richer. *Oauth2.0 in action*. International series of monographs on physics. Manning Publications Co., 2017. ISBN: 9781617293276 (cit. on p. 45).
- [16] Supertokens.com. *The best way to securely manage a user session*. URL: <https://supertokens.com/blog/the-best-way-to-securely-manage-user-sessions> (cit. on p. 47).
- [17] Webmarketingpro.it. *Guida allo sviluppo di app aziendali low code/no code*. URL: <https://www.webmarketingpro.it/sviluppo-app-mobili/guida-allo-sviluppo-di-app-aziendali-low-code-no-code/> (cit. on p. 51).
- [18] LowCodeItalia. *Le piattaforme Low-Code e No-Code rappresentano la prossima grande novità nel settore IT?* URL: [:https://www.lowcodeitalia.it/articoli/le-piattaforme-low-code-e-no-code-rappresentano-la-prossima-grande-novita224-nel-settore-it](https://www.lowcodeitalia.it/articoli/le-piattaforme-low-code-e-no-code-rappresentano-la-prossima-grande-novita224-nel-settore-it) (cit. on p. 52).
- [19] Prnewswire. *Ping Identity Launches PingOne DaVinci*. URL: [:https://www.prnewswire.com/news-releases/ping-identity-launches-pingone-davinci-a-no-code-identity-orchestration-service-for-delivering-seamless-digital-identity-experiences-301472576.html](https://www.prnewswire.com/news-releases/ping-identity-launches-pingone-davinci-a-no-code-identity-orchestration-service-for-delivering-seamless-digital-identity-experiences-301472576.html) (cit. on p. 53).
- [20] Ping Identity. *Getting started with DaVinci*. URL: <https://docs.pingidentity.com/bundle/davinci/page/tre1635461489038.html> (cit. on p. 56).
- [21] Knowband.com. *8 modi per creare un processo di registrazione/registrazione intuitivo*. URL: <https://www.knowband.com/blog/it/ecommerce-it/8-modi-per-creare-un-processo-di-registrazione-registrazione-intuitivo/> (cit. on p. 60).
- [22] DeckofCards. *deckofcardsapi*. URL: <https://www.deckofcardsapi.com/> (cit. on p. 61).
- [23] Ping Identity. *PingOne documentation for Duo security*. URL: [URL:https://docs.pingidentity.com/bundle/davinci-duo-connector/page/djf1646155630645.html](https://docs.pingidentity.com/bundle/davinci-duo-connector/page/djf1646155630645.html) (cit. on p. 64).

## BIBLIOGRAPHY

---

- [24] Ping Identity. *PingOne MFA Connector Documentation*. URL: <https://docs.pingidentity.com/bundle/davinci-pingone-mfa-connector/page/atm1642800835174.html> (cit. on p. 65).
- [25] Ping Identity. *Google Identity Provider*. URL: <https://docs.pingidentity.com/bundle/pingone/page/wdf1567784211161.html> (cit. on p. 65).
- [26] Ping Identity. *Worker Application definition*. URL: <https://docs.pingidentity.com/bundle/pingone/page/mst1564020489720.html> (cit. on p. 67).
- [27] Ping Identity. *PingOne Authorize Service*. URL: <https://hub.pingidentity.com/datasheets/3608-pingone-authorize> (cit. on p. 67).
- [28] Ping Identity. *Integration of a DaVinci Flow using Widget-Method*. URL: <https://apidocs.pingidentity.com/pingone/main/v1/api/#widget-method> (cit. on pp. 69, 71).
- [29] Qualtrics.com. *A/B Testing*. URL: <https://www.qualtrics.com/it/experience-management/ricerca/ab-testing/> (cit. on p. 73).
- [30] M.Driver P.Vincent K.Lijima. «Magic Quadrant for Enterprise Low-Code Application Platforms». In: (2019) (cit. on p. 81).