2024

PUIG CREDENTIALS

STUDENT :

Rubén Modamio García

EDUCATIONAL INSTITUTION :

Institut Puig Castellar

1. Introduction

1.1 Document Objective

The objective of this document is to describe in detail the process of issuing and validating digital credentials according to the standards OpenID for Verifiable Credential Issuance (VCI) and Verifiable Presentation (VP). Additionally, it will delve into the system architecture and the components involved in this process. This includes a detailed overview of the roles of participants, the workflow, the data model, and the algorithms used, as well as the deployment and testing strategies implemented.

1.2 Project Scope

The project covers the entire lifecycle from the initial creation of a digital credential to its secure delivery and storage in a Wallet, providing a comprehensive solution for the issuance and presentation of verifiable credentials. Advanced technologies and open standards are used to ensure the security, interoperability, and efficiency of the system.

1.3 System Overview

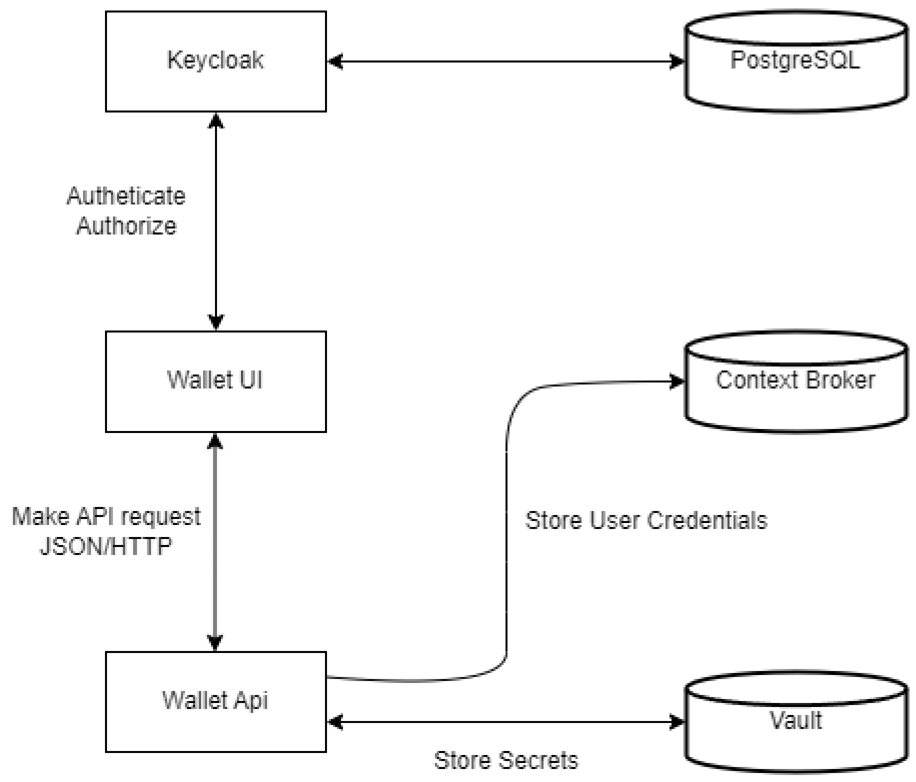
The system is composed of several key components that work together to issue, manage, and present verifiable credentials. These components include the Issuer + Authorization Server, the Wallet, and the Verifier, each with specific and critical functions for the success of the process.

2. Wallet Architecture

2.1 Introduction

The Wallet is a fundamental piece in the system's development as it is responsible for storing the verifiable credentials of a user, whether a natural or legal person. Additionally, it acts as an interaction tool between the different components in the implementation of the credential issuance and presentation flows.

2.2 Wallet Components



2.2.1 Wallet API

The Wallet API provides several services related to the application's business logic. These services allow interaction with other system components, facilitating the issuance, storage, and verification of credentials. Java with Spring WebFlux has been chosen to implement this API due to its capability to efficiently handle reactive and highly concurrent applications.

2.2.2 Wallet UI

The User Interface (UI) is developed in Angular, the solution through which users interact with the Wallet web application. Its main goal is to provide an intuitive and user-friendly experience for end users, allowing them to manage their credentials efficiently. Angular is used for the development of this UI, a platform that enables the building of dynamic and robust web applications.

2.2.3. Identity Access Management (IAM)

IAM encompasses comprehensive identity management and access control. It includes an Authorization Server that facilitates user authentication and authorization.

- Keycloak

  Keycloak has been chosen as the IAM solution due to its multiple robust and flexible features. Keycloak is an open-source tool that provides identity management, identity federation, and access management, simplifying user and permission handling. It offers integration with multiple authentication protocols (OpenID Connect, OAuth 2.0, and SAML), support for multi-factor authentication (MFA), and session management, ensuring optimal security and usability.

- WebAuthn

  WebAuthn is a web standard developed by the World Wide Web Consortium (W3C) that enables secure online authentication without solely relying on passwords. It provides an API that allows users to authenticate using more secure methods, such as cryptographic keys, biometric authentication, or external security devices. Authentication is performed through the generation and verification of digital signatures, offering greater protection against threats such as phishing and password theft.

2.2.4 Personal Data Space

The Personal Data Space is a Context Broker used to store the data that the user owns, including verifiable credentials and other personal data. This space ensures that the user's data is organized and securely accessible.

Scorpio. has been chosen for the Personal Data Space, an open-source implementation of the NGSI-LD standard for contextual data management and exchange. It is highly scalable and provides an efficient way to manage contextual data, facilitating interoperability between different systems and applications.

2.2.5 Vault

The Vault is essential for securely storing key pairs. These key pairs are used to sign verifiable presentations or credentials, ensuring their integrity and authenticity.

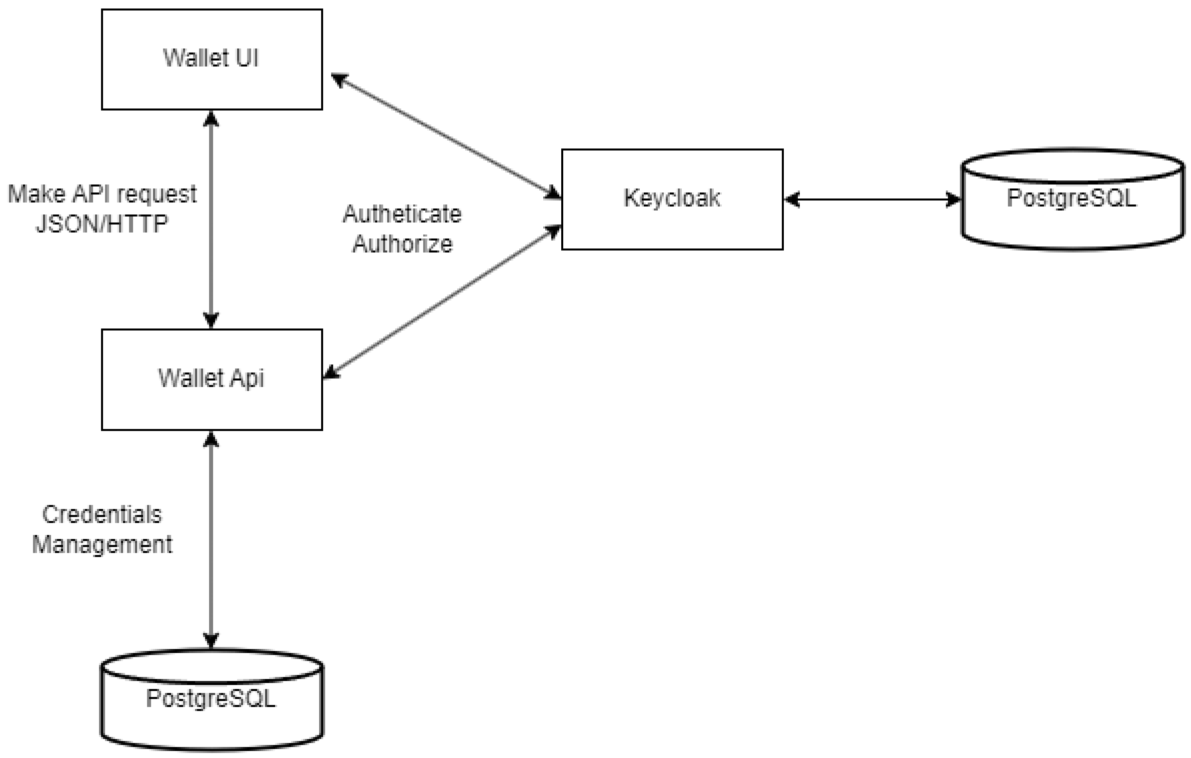
HashiCorp Vault has been chosen for this function due to its ability to robustly manage secrets and protect sensitive data. Vault allows for the secure storage of secrets, providing detailed access control, auditing, and dynamic secret management. Its integration with multiple systems and high security make it an ideal choice for this purpose.

3. Architecture of the Issuer and Authorization Server

3.1 Introduction

The Issuer and the Authorization Server are key components of the system responsible for issuing credentials and managing user authorization and authentication. Both components work together to ensure that credentials are issued securely and efficiently.

3.2 Components of the Issuer and Authorization Server



3.2.1 Issuer UI

The user interface (UI) of the Issuer is developed in Angular, providing an intuitive and efficient user experience. It allows administrators to manage credential issuance through an interactive web platform.

3.2.2 Issuer API

The Issuer's API is developed in Java with Spring WebFlux, similar to the Wallet's API. This choice allows for handling reactive and highly concurrent applications, ensuring a quick and efficient response to credential issuance requests

3.2.3 Database

PostgreSQL is used as the database to manage the issued credentials. PostgreSQL is an open-source relational database that offers robustness, flexibility, and performance, suitable for handling critical data and complex transactions.

3.2.4 Issuer Keycloak

Issuer Keycloak is an extension of Keycloak that allows customization of the OAuth 2.0 flow for users in a verifiable credentials issuance process (VC flow). This customization is crucial for adapting the authorization and authentication mechanisms to the specific needs of the credential issuance process.

4. Process of Issuing a Credential

4.1 Overview

This section describes the issuance process of A Credential using a specific profile of the [OpenID.VCI] protocol. This profile simplifies the standard by limiting the available options and implementing a set with predefined values, suitable for our use case but still maintaining overall flexibility.

4.2 Participants:

End-User: The recipient of the Credential, a student, who communicates securely with the clerk department.

Wallet(Phone): A Web application used as Identity Wallet. In this profile we assume that the wallet is not previously registered with the Issuer and that the wallet does not expose public endpoints that are called by the Issuer. In other words, from the point of view of the Issuer, the wallet in this profile is almost indistinguishable from a full mobile wallet, and it does not assume a previous relationship with the Wallet.

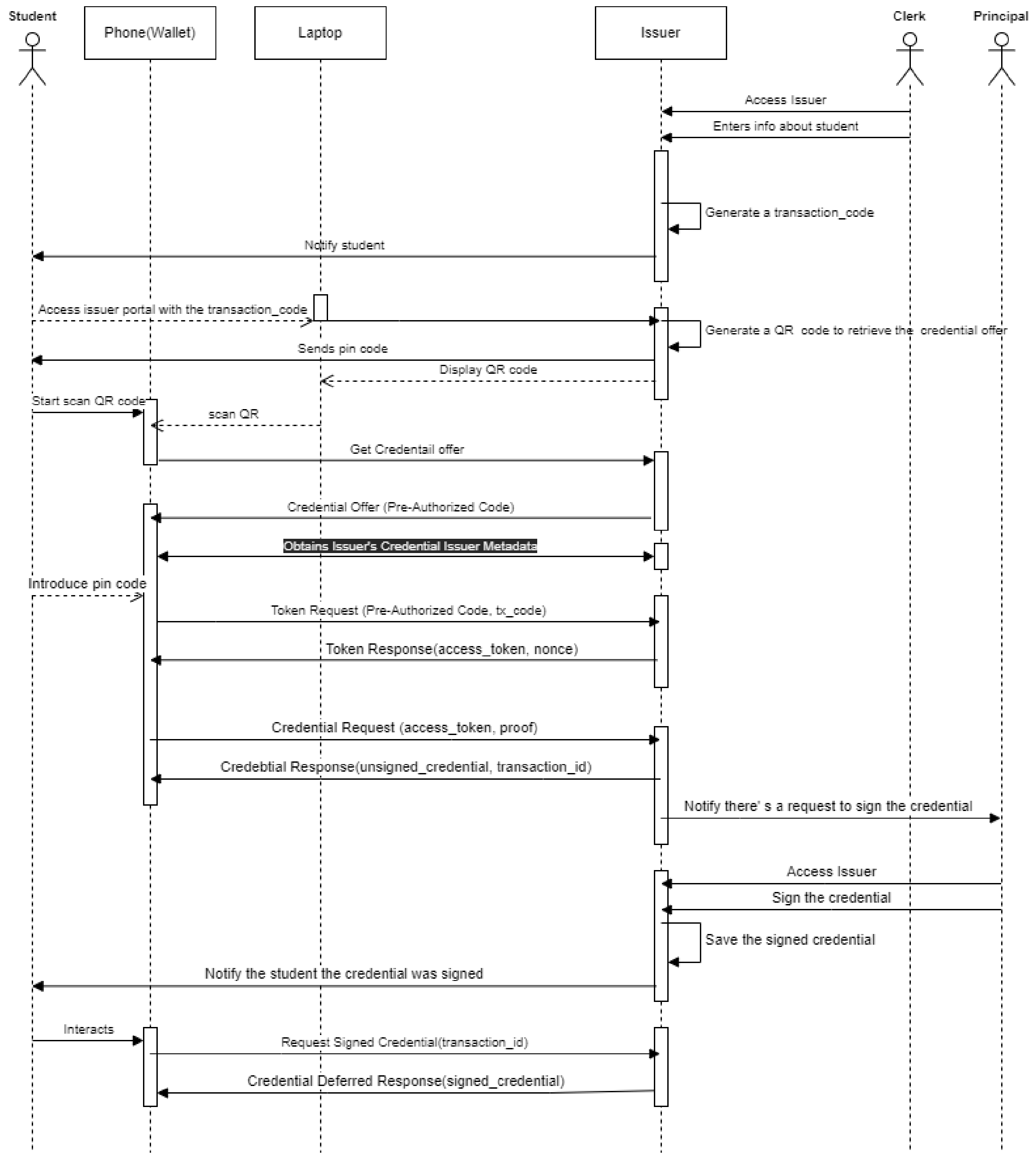
Issuer: In this profile we assume that the Issuer is composed of two components:

- **Issuer backend**: the main server implementing the business logic of the Issuer as a web application and additional backend APIs required for issuance of credentials.

- **Authorization server**: the backend component implementing the existing authentication/authorization functionalities for the Issuer entity.

  We also assume that for external entities, like the Wallet, the two components are under the same domain so they look like one single server entity to the Wallet.

Overall Credential issuance flow diagram


Student
Phone(Wallet)
Laptop
Issuer
Clerk
Principal
Access Issuer
Enters info about student
Generate a transaction_code
Notify student
Access issuer portal with the transaction_code
Generate a QR code to retrieve the credential offer
Sends pin code
Display QR code
Start scan QR code
scan QR
Get Credentail offer
Credential Offer (Pre-Authorized Code)
Obtains Issuer's Credential Issuer Metadata
Introduce pin code
Token Request (Pre-Authorized Code, tx_code)
Token Response(access_token, nonce)
Credential Request (access_token, proof)
Credebtial Response(unsigned_credential, transaction_id)
Notify there' s a request to sign the credential
Access Issuer
Sign the credential
Save the signed credential
Notify the student the credential was signed
Interacts
Request Signed Credential(transaction_id)
Credential Deferred Response(signed_credential)

4.3 Detailed Flow

The clerk accesses the Issuer portal

The Clerk accesses the Credential Issuer Portal using a web browser.

2. The Credential Issuer Portal shows the Home Page with the option to log in.

3. The clerk clicks on the Log in | Sing up button.

4. The Issuer redirects the clerk to the Authorization Server.

The clerk registers a new credential for a specific Student of their organization using a form.

1. The Issuer shows a form to the Clerk to register a new credential to the student. The data related to the Mandator is pre-filled using the data from the Digital Certificate.

2. The clerk fills Mandatee section with the data of the Student that will receive the Credential using the form.

3. The clerk sets the role that the Student will have, for now student, but many others could be added in the future.

4. The clerk clicks on the Create Credential button.

5. The Issuer validates the data and creates a new credential.

NOTE: This credential has the final format, but the Cryptographic Binding is not set and the credential is not signed yet.

6. The Issuer creates a new Credential Procedure entity. The Credential Procedure is in the status WITHDRAWN and sets the attribute credential_decoded with the Credential received.

NOTE: The Credential Procedure is a way to manage the Credential Issuance process. It includes the Credential, the status of the Credential Issuance, and the organization identifier, etc. You can find more information about the Credential Procedure in the Data Model - Credential Procedure section.

7. The Credential Issuer creates a new Deferred Credential Metadata. It includes the Transaction Code, as transaction_code, and the id of the Credential Procedure, procedure_id. This Transaction Code is a nonce that will be used to bind the Credential Offer with the Credential Request and to create the URI that will be sent to the Student via email.

NOTE: The Deferred Credential Metadata is a way to manage the metadata needed to manage the deferred process. It includes the Transaction Code, the Credential Procedure id, etc. You can find more information about the Deferred Credential Metadata in the Data Model

The Credential Issuer notifies the student with the link needed to start the credential issuance process.

When the Credential Issuer finishes the registration of the Credential Student, it sends an email to the Student with the link to start the credential issuance process. The link includes the Transaction Code as a query parameter.

This is a non-normative example of the link that the Student will receive:

https://issuer/credentials?transaction_code=oaKazRN8I0IbtZ0C7JuMn5

NOTE: The Credential Issuer sends the email using an SMTP server. The email includes a template with the link to start the credential issuance process and short but descriptive documentation about the process that the student will follow. The email is sent to the email address of the Student that was set in the Credential Student form.

The Student accesses the Credential Issuer executing the link attached in the received email.

The Students reads the email and clicks on the link to start the process of receiving the Credential.

The Student is redirected to the Credential Offer Page in the Credential Issuer Portal.

The Issuer validates the transaction_code and retrieves the Credential Procedure.

The Credential Issuer makes a Credential Offer and updates the Deferred Credential Metadata.

The Credential Issuer makes a Credential Offer:

- The Credential Issuer fetches the Authorization Server to create a pre-authorized_code.
- The Credential Issuer creates a tx_code, which is a PIN that will be sent to the Student via email.

The Credential Offer updates the Deferred Credential Metadata. The pre-authorized_code is added to the Credential Procedure as auth_server_nonce attribute.

The Credential Issuer sends the tx_code (PIN) to the Student via email.

The tx_code created during the Credential Offer is sent to the Student via email. We will use a template to send the email to the Student.

The Credential Issuer displays a QR code

The Credential Issuer builds a Credential Offer Uri (section 4.1.3). This URI is a link that points to the Credential Offer. This credential_offer_uri is displayed as a QR code.

The Credential Offer Uri uses a nonce value to bind the Credential Offer with the Credential Offer Uri.

The nonce is saved as a key value in memory cache, and the Credential Offer as the value. This is burnt after the Credential

Offer is retrieved by executing the Credential Offer Uri by the Wallet.

This is a non-normative example of the Credential Offer Uri:

credential_offer_uri=https%3A%2F%2Fserver%2Eexample%2Ecom%2F84dr684f51jfdbj

The Student accesses the Wallet with their credentials.

The Student opens the Wallet application on their device.

The Student logs in to the Wallet using their credentials (username and password).

The Student scans the QR code with the Wallet.

The Student clicks on the button Scan.

The camera of the device is activated.

The Student scans the QR code displayed by the Credential Issuer.

The Wallet reads the QR code content, interprets that it is a Credential Offer Uri, and executes the Credential Offer Uri.

The Wallet fetches the Credential Offer.

The Wallet fetches the Credential Offer executing tha parsed credential_offer_uri.

This is a non-normative example of the Credential Offer:

```
{
   "credential_issuer": "https://credential-issuer.example.com",
   "credential_configuration_ids": [
      "UniversityDegreeCredential",
      "org.iso.18013.5.1.mDL"
   ],
   "grants": {
      "urn:ietf:params:oauth:grant-type:pre-authorized_code": {
         "pre-authorized_code": "oaKazRN8I0IbtZ0C7JuMn5",
         "tx_code": {
            "length": 4,
            "input_mode": "numeric",
            "description": "Please provide the one-time code that was sent via e-mail"
         }
      }
   }
}
```

NOTE: The Credential Offer Response includes the header Accept-Language to indicate the language preferred for display. The language(s) in HTTP Accept-Language and Content-Language Headers MUST use the values defined in [RFC3066].

JUSTIFICATION: We do not implement the Accept-Language header in the Wallet's Credential Issuer Metadata request.

The Wallet fetches the Credential Issuer's Metadata and the Authorization Server's Metadata.

The Wallet fetches the Credential Issuer's Metadata (section 11.2) creating a dynamic URL using the parameter credential_issuer and concatenating the path /.well-known/openid-credential-issuer.

NOTE: The communication with the Issuer Metadata Endpoint MUST use TLS.

NOTE: The request MUST be an HTTP request using GET method and the URL SHOULD NOT contain any query parameters.

NOTE: The Credential Issuer MUST return a JSON document compliant with this specification using the application/json media type and the HTTP Status Code 200.

NOTE: The Credential Issuer Metadata response includes the header Accept-Language to indicate the language preferred for display. The language(s) in HTTP Accept-Language and

Content-Language Headers MUST use the values defined in [RFC3066].

JUSTIFICATION: We do not implement the Accept-Language header in the Wallet's Credential Issuer Metadata request.

The Wallet fetches the Authorization Server's Metadata (section 11.3) creating a dynamic URL using the parameter authorisation_server and concatenating the path /.well-known/openid-configuration.

The Student interacts with the Wallet adding the tx_code received in the email.

Previously to start the Token Request, the Student must add the tx_code received in the email to the Wallet.

This is a security measure to ensure that the Student is the one that has received the email and is the one that is interacting with the Wallet.

The Wallet shows a modal to the Student to add the tx_code.

The Students add the tx_code to the Wallet.

The Wallet sends a Token Request to the Credential Issuer. The Token Request contains the Pre-Authorized Code obtained in the Credential Offer and the tx_code added by the Student.

The Wallet sends a Token Request to the Credential Issuer's Token Endpoint, retrieved from the Authorization Server Metadata info. The Token Request contains the Pre-Authorized Code obtained in the Credential Offer and the tx_code added by the Student.

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn:ietf:params:oauth:grant-type:pre-authorized_code
&pre-authorized_code=SplxlOBeZQQYbYS6WxSbIA
&tx_code=493536
```

The Authorisation server validates the Token Request.

The Authorisation Server sends the pre-authorized_code and the access_token to the Credential Issuer.

The Credential Issuer gets the Deferred Credential Metadata by the pre-authorized_code and updates the auth_server_nonce with the access_token.

The Authorization Server sends the Token Response (section 6.2) to the Wallet.

The Wallet sends a Credential Request to the Credential Issuer's Credential Endpoint. It contains the Access Token and the proof of possession of the private key of a key pair to which the Credential Issuer should bind the issued Credential to.

The Wallet generates a Credential Request (section 7.2). To do that, the Wallet needs:

The Wallet creates a did:key to the Student. This means the creation of a key pair, generating a DID, and storing the private key in a secure enclave such as HashiCorp Vault.

The Wallet creates a proof

The Wallet sends a Credential Request to the Credential Issuer's Credential Endpoint with the Access Token and the proof. This proof is the proof of possession of the private key of a key pair to which the Credential Issuer should bind the issued Credential to.

The Credential Issuer sends and emails to the Principal to notify them that they have a new credential pending to be signed.

The Credential Issuer sends an email to the Principal to notify they have a new credential pending to be signed.

The Credential Issuer returns a Credential response

The credential Issuer returns a Credential response containing the unsigned credential and a transaction_id, necessary to retrieve the signed credential when the Student gets notified.

The Principal start the process to sign the credential

The Principal reads the notification email

The principal logs in to the Local Signature component and gets the pending credentials from the Credential Issuer.

The principal selects one or more credentials to sing.

The Local Signature component signs the Credential and sends it back to the Credential Issuer.

The Credential Issuer sends an email to the Student notifying that the Credential is ready to be retrieved.

The Credential Issuer receives the signed Credentials from the Local Signature Component.

The Credential Issuer validates the Authorization header.

The Credential Issuer updates the credential_encoded of the Credential Procedure with the signed Credential and sets a new status of the Credential Issuance (VALID).

The Credential Issuer updates the vc attribute of the Deferred Credential Metadata with the signed Credential.

The Credential Issuer sends an email to the Student notifying that the Credential is ready to be retrieved.

The Wallet retrieves the Credential using a Deferred Credential Request (access_token, transaction_id).

The Student reads the email and accesses the Wallet.

The Student clicks on the button Retrieve Credential of the representation of the credential not signed in their wallet.

The Wallet sends a Deferred Credential Request to the Credential Issuer's Deferred Credential Endpoint with the Access Token and the Transaction ID.

The Credential Issuer validates the Access Token and the Transaction ID

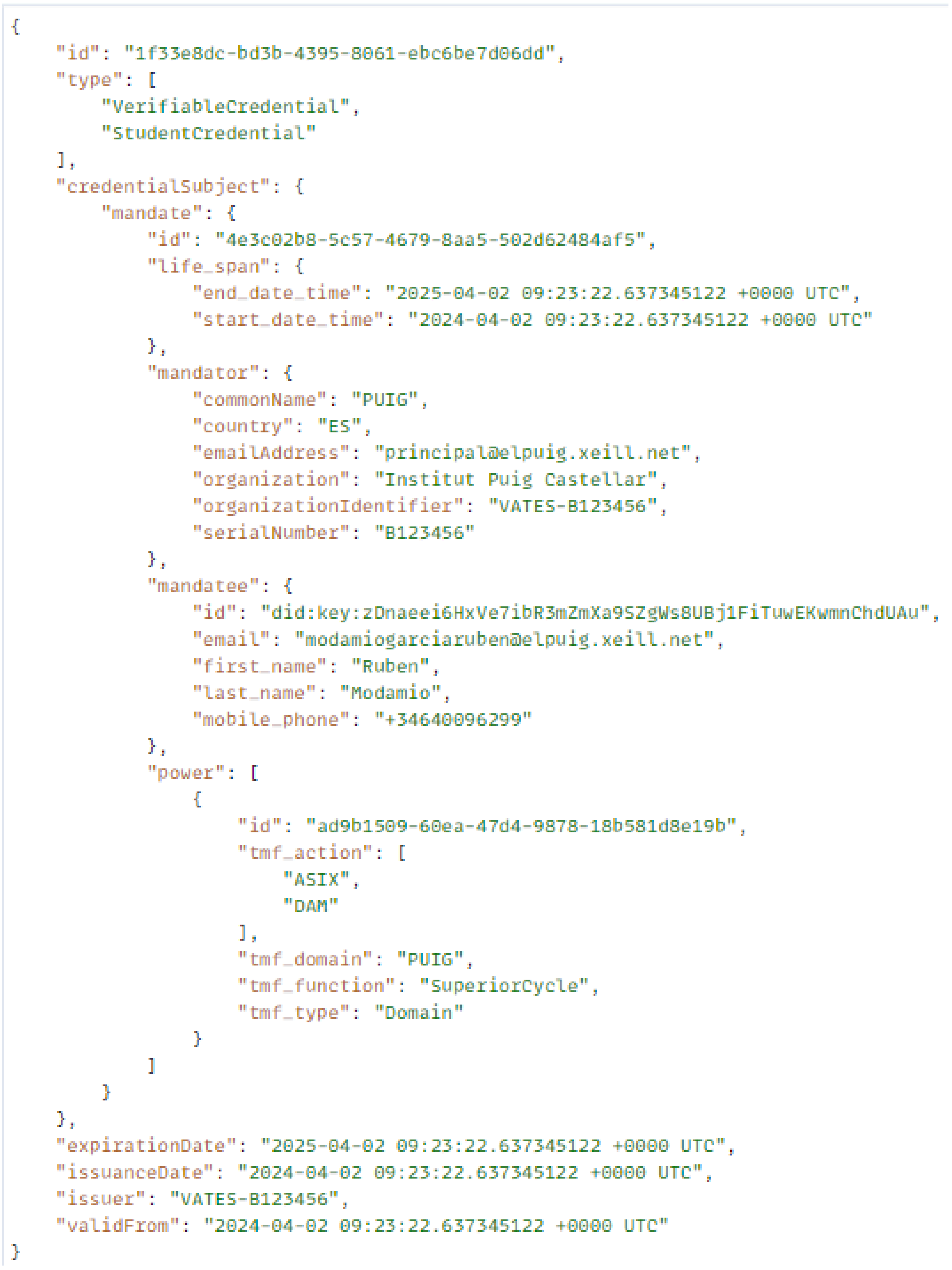
- If the Access Token and the Transaction ID are valid, but the Credential is not ready, the Credential Issuer:
    - burns the Transaction ID
    - generates a new Transaction ID
    - persists the Transaction ID in memory
    - updates the Deferred Transaction Metadata
    - returns a Deferred Response with the new Transaction ID
- If the Access Token and the Transaction ID are valid and the Credential is ready, the Credential Issuer:
    - retrieves the Credential from the Credential Procedure
    - returns a Deferred Response with the Credential

The Wallet receives the Deferred Response and retrieves the Credential and stores it in the Wallet, in case the credential is not signed the wallet saves the new transaction ID.

5. Data Model

5.1 Example of the Student Credential

Below is a detailed example of how a student credential is structured:



```
{
    "id": "1f33e8dc-bd3b-4395-8061-ebc6be7d06dd",
    "type": [
        "VerifiableCredential",
        "StudentCredential"
    ],
    "credentialSubject": {
        "mandate": {
            "id": "4e3c02b8-5c57-4679-8aa5-502d62484af5",
            "life_span": {
                "end_date_time": "2025-04-02 09:23:22.637345122 +0000 UTC",
                "start_date_time": "2024-04-02 09:23:22.637345122 +0000 UTC"
            },
            "mandator": {
                "commonName": "PUIG",
                "country": "ES",
                "emailAddress": "principal@elpuig.xeill.net",
                "organization": "Institut Puig Castellar",
                "organizationIdentifier": "VATES-B123456",
                "serialNumber": "B123456"
            },
            "mandatee": {
                "id": "did:key:zDnaeei6HxVe7ibR3mZmXa9SZgWs8UBj1FiTuwEKwmnChdUAu",
                "email": "modamiogarciaruben@elpuig.xeill.net",
                "first_name": "Ruben",
                "last_name": "Modamio",
                "mobile_phone": "+34640096299"
            },
            "power": [
                {
                    "id": "ad9b1509-60ea-47d4-9878-18b581d8e19b",
                    "tmf_action": [
                        "ASIX",
                        "DAM"
                    ],
                    "tmf_domain": "PUIG",
                    "tmf_function": "SuperiorCycle",
                    "tmf_type": "Domain"
                }
            ]
        }
    },
    "expirationDate": "2025-04-02 09:23:22.637345122 +0000 UTC",
    "issuanceDate": "2024-04-02 09:23:22.637345122 +0000 UTC",
    "issuer": "VATES-B123456",
    "validFrom": "2024-04-02 09:23:22.637345122 +0000 UTC"
}
```

5.2 Data Model Components

Each component in the data model has a specific function within the system. Below are descriptions of each of these components and their functions:

id

- **Description:** Unique identifier of the credential.
- **Function:** Allows for the unique identification of each issued credential.

type

- **Description:** List of credential types, in this case, including "VerifiableCredential" and "StudentCredential".
- **Function:** Specifies the type of credential, providing context about its purpose and use.

credentialSubject

- **Description:** Contains specific information about the credential subject, in this case, a mandate describing the relationship between the mandator and the mandatee.
- **Function:** Provides details of the subject referred to by the credential, including personal information and the scope of the mandate.

Mandate

- **id:** Unique identifier of the mandate.
- **life_span:** Duration of the mandate, including start and end dates.
- **mandator:** Information about the mandator (the one who grants the mandate).
    - **commonName:** Common name.
    - **country:** Country of origin.
    - **emailAddress:** Email address.
    - **organization:** Organization to which the mandator belongs.
    - **organizationIdentifier:** Organization identifier.
    - **serialNumber:** Serial number of the mandator.
- **mandatee:** Information about the mandatee (the one who receives the mandate).
    - **id:** Identifier of the mandatee (did:key).
    - **email:** Email address.
    - **first_name:** First name of the mandatee.
    - **last_name:** Last name of the mandatee.
    - **mobile_phone:** Mobile phone of the mandatee.
- **power:** List of powers granted in the mandate.
    - **id:** Unique identifier of the power.
    - **tmf_action:** Actions allowed by the mandate.
    - **tmf_domain:** Domain in which the mandate applies.
    - **tmf_function:** Function of the mandatee within the domain.
    - **tmf_type:** Type of domain.

expirationDate

- **Description:** Expiration date of the credential.

- **Function:** Indicates until when the credential is valid.

issuanceDate

- **Description:** Issuance date of the credential.
- **Function:** Indicates when the credential was issued.

issuer

- **Description:** Identifier of the credential issuer.
- **Function:** Provides information about the entity that issued the credential.

validFrom

- **Description:** Date from which the credential is valid.
- **Function:** Indicates the start of the credential's validity.

6. Signing Algorithm

To ensure the security and integrity of verifiable credentials, the secp256r1 signing algorithm is used. This algorithm is part of elliptic curve cryptography (ECC), which relies on the difficulty of solving mathematical problems related to elliptic curves over finite fields.

6.1 Key Generation Process

- A pair of keys (public key and private key) is generated using the secp256r1 curve.
- The DID is generated based on the public key of the key pair, following the did standard.
- The DID and key pair are stored as a key-value pair in the vault.

6.2 Considerations for Choosing the Algorithm

Originally, secp256k1 was considered as the signing algorithm, similar to the one used in Bitcoin. However, secp256r1 was chosen due to the lack of support in the JWSigner library for secp256k1. The differences and reasons behind the decision to change the signing algorithm are explained below.

6.2.1 Differences Between secp256k1 and secp256r1

1) Curve Structure
    - **secp256k1**: A Koblitz curve, defined in a characteristic 2 finite field.
    - **secp256r1**: A prime field curve, defined in a prime characteristic finite field.
2) Randomness:
    - **secp256k1:** A non-random curve.
    - **secp256r1:** A pseudo-random curve, structured by NIST.

6.2.2 Controversy over the Security of secp256r1

The article titled "Why Did Satoshi Decide To Use Secp256k1 Instead Of Secp256r1?" suggests that NIST's choice of secp256r1 could have been influenced by the NSA to introduce a potential backdoor in the pseudo-random curves. This concern arises from documents leaked by Edward Snowden, which suggest that the NSA could have manipulated the choice of curves to facilitate the insertion of weaknesses in cryptographic systems.

Excerpt from the Article:

"The main difference between secp256k1 and secp256r1 is that secp256k1 is a Koblitz curve which is defined in a characteristic 2 finite field, while secp256r1 is a prime field curve. Secp256k1 curves are non-random while secp256r1 is pseudo-randomly structured. Ironically, this is generally believed as the reason why Satoshi did not use secp256r1. In particular, the leaked documents by the National Security Agency contractor and whistleblower Edward Snowden suggested that the NSA had used its influence over NIST to insert a backdoor into a random number generator used in elliptic curve cryptography standards."

This information highlights the importance of considering the security and potential implications of the choice of signing algorithm. Despite these concerns, secp256r1 was chosen due to the lack of support in the JWSigner library for secp256k1.

6.3 Security Provider

To implement cryptographic operations related to key generation and signatures, BouncyCastle is used as the security provider. BouncyCastle is an open-source cryptography library that provides a wide range of cryptographic algorithms and tools.

6.4 Decentralized Identifiers (DID)

6.4.1 What are Decentralized Identifiers (DID)?

Decentralized Identifiers (DID) are a new type of identifier that allows for decentralized and secure identity verification. Unlike traditional identifiers, which depend on central authorities (such as government-issued identification numbers), DIDs are designed to be controlled directly by the identity owner, without intermediaries.

DIDs are part of the self-sovereign identity (SSI) ecosystem, which promotes user control over their own identities and personal data. Each DID is unique and is associated with a DID document that contains the public information necessary to interact with the DID, such as public keys and service endpoints.

6.4.2 DID Key

The did:key method is one of the simplest and most straightforward methods for generating DIDs. It uses a public key to derive a DID, providing an easy and quick way to create identifiers and is the one we use.

7.Testing and Code Quality

7.1 Unit Tests

To ensure the quality and reliability of the code, comprehensive unit tests have been developed across the project's various repositories. These unit tests have been implemented as follows:

- Backend:
    - JUnit and Reactor Test were used for unit testing the backend components.
    - JaCoCo was implemented to generate code coverage reports, ensuring that the tests cover a wide spectrum of the application's code.

- Frontend:
    - For the frontend, Karma was used as the test runner, providing a robust and efficient testing environment for Angular applications.

The use of these tools and frameworks has allowed for the early identification and correction of errors, thus improving the software's stability and quality.

7.2 Integration with SonarCloud

The organization's GitHub repositories have been linked with SonarCloud to perform continuous code quality analysis. SonarCloud offers several key benefits:

Vulnerability Detection:

- SonarCloud analyzes the code for potential vulnerabilities and security issues. Thanks to these analyses, most detected vulnerabilities have been identified and corrected.

Quality Metrics:

- SonarCloud provides detailed metrics on code quality, such as test coverage, duplicated code, bugs, and code smells. These metrics are crucial for maintaining and improving code quality over time.

Here you can see the metrics of the repositories.

7.3 CI/CD Pipeline

Continuous Integration and Continuous Deployment (CI/CD) pipelines have been implemented in the GitHub repositories. These pipelines automate several critical tasks, including integration with SonarCloud and deployment to DockerHub. The process is described below

7.3.1 SonarCloud Analysis:

When a pull request is created or changes are made to the main branch, the pipeline launches an automatic analysis of the repository in SonarCloud. This analysis helps detect code quality and security issues before they are integrated into the main codebase.

7.3.2 Image Building and Deployment:

After the SonarCloud analysis, the pipeline builds an image of the project using Docker and automatically deploys it to DockerHub. This ensures that there is always an updated and ready-to-deploy version of the project.

7.3.3 Secret Configuration:

- To perform these tasks securely, a series of secrets need to be configured in the repositories. These secrets include credentials required to interact with SonarCloud and DockerHub.

- The secrets are configured in the repository's settings section on GitHub and dynamically retrieved in the pipeline through environment variables. This ensures that sensitive credentials are not exposed in the source code.

Here you can see how the pipeline is configured.

8. Solution Deployment

8.1 Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is a fundamental practice in deploying our solution, allowing for the management and provisioning of technological resources through code definitions. In this project, Docker Compose is used to define and manage the containers necessary for deploying the services.

8.2 Scripts Implemented to Facilitate Tasks

These scripts automate crucial and repetitive tasks, allowing developers to focus on developing and testing new features without worrying about the manual configuration of these components.

8.2.1 Script to Initialize Vault and Store Token and Keys

This script starts the Vault server, initializes it, and stores the token and unseal keys to facilitate testing, you can find the script here.

8.2.2 Script to Set the Token in the Wallet API:

This script waits for the root token to be generated in Vault and then exports it as an environment variable for use in the Wallet API and you can find the script here.

8.3 Deployment Instructions in the Repository README

This repository README provides a detailed guide on how to set up the various system components and test the complete flow. These instructions ensure that anyone can deploy and test the solution without issues.

9. Conclusions

9.1 Results Obtained

Wallet:

- The Wallet component has been developed with high efficiency, providing a robust tool for managing verifiable credentials. The Wallet allows interaction with other system components, meeting the expected functionalities.

Issuer:

- Despite not being fully developed and having some visual bugs, the Issuer meets the basic functionalities necessary to complete the credential issuance process. Although not all advanced credential management functionalities are implemented, the component is functional for basic processes.

9.2 Next Steps and Improvements

To continue improving and completing the system, the following steps and improvements are identified:

Completion of Issuer Functionalities:

- Complete the pending advanced credential management functionalities.
- Resolve existing visual bugs to enhance the user experience.

Development of a Graphical Interface for the Signer:

- Create a graphical interface for the signer that allows smooth communication with the Issuer.
- Facilitate the credential signing process through a user-friendly interface.

Enabling Certificate Login in Keycloak:

- Implement certificate-based login in Keycloak to improve security and authenticity of access to the Issuer.
- Configure automatic extraction of Mandator data from the digital certificate used for login.

Development of the Verifier:

- Implement a Verifier following the OpenID for Verifiable Presentations (OPENIDVP) flow.
- Develop a portal that allows the use of verifiable credentials for various transactions.

Increasing Test Coverage:

- Ensure 80% test coverage across all system components.
- Implement additional tests to guarantee the system's robustness and reliability.

Deployment of the Complete Infrastructure:

- Once all functionalities are fully implemented and tested, proceed with the deployment of the complete infrastructure.

9.3 Conclusion

Significant progress has been made in the system's development, especially in the Wallet component. Although the Issuer still requires improvements and completion, solid foundations for its basic functionality have been established.

With the identified next steps and improvements, the system is expected to be completed and optimized, providing a comprehensive and efficient solution for the issuance and management of verifiable credentials.

10. Appendices

10.1 GitHub Repositories

The GitHub repositories contain the source code for the different components of the project, as well as the documentation and scripts necessary for deployment and operation. Below are the main repositories:

Wallet Api:

- URL: https://github.com/puigcredentials/wallet-api
- Description: Contains the code for the Wallet API developed in Java with Spring WebFlux.

Wallet UI:

- URL: https://github.com/puigcredentials/wallet-ui
- Description: Contains the code for the Wallet user interface developed in Angular.

Wallet keycloak:

- URL: https://github.com/puigcredentials/wallet-keycloak
- Description: Contains a customized keycloak image.

Issuer Api:

- URL: https://github.com/puigcredentials/issuer-api.git
- Description: Contains the code for the Issuer API developed in Java with Spring WebFlux.

Issuer UI:

- URL: https://github.com/puigcredentials/issuer-ui.git
- Description: Contains the code for the Issuer user interface developed in Angular.

Issuer Keycloak:

- URL: https://github.com/puigcredentials/issuer-keycloak
- Description: Contains the code for the Issuer keycloak plugin developed in Java with Spring Boot.

IAC:

- URL: https://github.com/puigcredentials/IAC
- Description: Contains the infrastructure as code to deploy all components.

10.2 SonarCloud Organization

SonarCloud is used for continuous code quality analysis, detecting potential vulnerabilities and providing detailed metrics. The SonarCloud organization is set up to automatically analyze the linked GitHub repositories.

Organization URL on SonarCloud:
https://sonarcloud.io/organizations/puigcredentials

10.3 DockerHub Organization

DockerHub is used to store and distribute Docker container images for the various project components. The DockerHub organization allows for easy management and access to the images needed for system deployment.

Organization URL on DockerHub: https://hub.docker.com/u/puigcredentials

10.4 References

For more details on how to clone, configure, and deploy the project components, please refer to the respective READMEs in the GitHub repositories. Additionally, continuous integration and code quality metrics can be reviewed on the SonarCloud dashboard, and the necessary Docker images for deployment are available on DockerHub.