



SSI EduWallets

Developer documentation

SSI EduWallets

Developer Documentation | Call 17 | Project ID 6344

License CC BY-SA

Contents

1 Course of the work packages.....	4
2 Introduction to SSI EduWallets.....	4
3 What is SSI EduWallets.....	4
4 What is the purpose of SSI EduWallets.....	5
5 How Web 3.0 & SSI (Self Sovereign Identity) paradigm works.....	5
6 Why to use the new paradigm of Web 3.0.....	7
7 SSI EduWallets Components.....	9
Wallets.....	9
User Interface.....	9
Wallet Kit API.....	9
Issuer API (Open API).....	10
Verifiable API (Open API).....	10
8 Standards used.....	10
8.1 W3C Decentralized Identifiers.....	10
8.2 W3C Verifiable Credentials Data Model.....	11
9 Verifiable credentials security.....	12
10 Architecture and deployment.....	13
10.1 Web Wallet proof of concept.....	13
Wallet deployment.....	14
Wallet backend.....	14
Demo wallet UI.....	15
10.2 Demo Application.....	15
Demo Application UI.....	16
Architecture.....	17
UI pages, workflows & requests.....	17
UI components, workflows & requests.....	23
The "issueCredentialsBtn" component.....	23
The "VerifyCredentialsBtn" component.....	24
The "userProfileDialog" component.....	24
How to run.....	25
Demo Application API.....	26
Functionalities.....	26
Architecture.....	27
Software components.....	27
How to run.....	29

Demo Application Web wallet.....	30
Architecture.....	30
Issuance workflow & requests.....	31
Verification workflow & requests.....	38
How to run.....	43
11 Development Environment Setup.....	44
11.1 System Requirements.....	44
11.2 Dependency Installation.....	45
11.3 System Configuration.....	46
11.4 Repositories needed.....	47
12 Open API documentation.....	50
13 Wallet kit API.....	51
Functionalities.....	52
Architecture.....	53
Integration with the Demo Application.....	55
How Wallet Kit verify a verifiable presentation.....	56
14 Wallet Kit configuration.....	57
15 DID and keys creation.....	62
16 Creation of VC schema.....	64
17 Creation of the verifiable credential.....	68
18 Verifiable credentials issuance.....	71
19 Verifiable presentation.....	78
20 Documental sources.....	85

1 Course of the work packages

The Developer's Manual for Self-Sovereign Identity (SSI) EduWallets provides **guidelines and best practices** for developing a stack of applications for the Web 3.0 related to the SSI for the **issuance, verification, and exchange of verifiable credentials**. This manual is designed to **assist developers** in understanding how the SSI paradigm works and the process of issuance and verification of the verifiable credentials and ensuring the proper implementation of related specifications and standards.

2 Introduction to SSI EduWallets

The SSI EduWallets project was born with the purpose of transforming the issuance of traditional educational certificates into **cryptographically secure and interoperable digital documents** called verifiable credentials (VCs), thus **eliminating** problems such as the low **interoperability** of the credentials and the **forgery** of physical documents which are easy to falsify but difficult to verify their authenticity. Adjusting to the new generation of Web 3.0 and the Self-Sovereign Identity (SSI) paradigm through the European Self-Sovereign Identity Framework (ESSIF).

3 What is SSI EduWallets

SSI EduWallets is a software implementation made up of several components which, once integrated into an e-learning platform, allow the use of **self-sovereign identity wallets** within these platforms for the **issuance, verification, and exchange** of verifiable credentials, thus allowing the exchange of verifiable credentials between the user's wallets and the platforms that implement this system.

4 What is the purpose of SSI EduWallets

SSI EduWallets allows transforming any traditional elearning platform to the new self-sovereign identity paradigm of Web 3.0, this allows users **to be the owners of their own information** so they are the ones who manage their information and not third parties.

In addition, this implementation allows the **exchange of verifiable credentials** between the **user's wallets and the platforms** that implement the system, so that users can present certain verifiable credentials to the platforms and then they can **read and verify** them in order to use that information within the platforms streamlining some processes. The platforms will also be able to **issue educational verifiable credentials** to the user's wallet, that certifies that a user has completed and achieved certain knowledge at the end of a course or assessment.

These verifiable credentials will be **stored in the user's wallets** to later be used at any time when necessary. The implementation brings the advantages of **issuing digital and cryptographically secure certificates** that can be **easily verified** by third parties and **interoperable** between other SSI systems based on the same standards ([ESSIF/EBSI¹](#)).

5 How Web 3.0 & SSI (Self Sovereign Identity) paradigm works

The new paradigm of Web 3.0 aims to revolutionize certain areas of previous generations of the web; it represents a more **decentralized, secure**, and user-centered vision of the web, where **users have greater control over their data** and digital experiences.

Specifically, SSI EduWallets focuses on the **self-sovereign identity** of the users, this means that it is the user who stores and manages all their data in a single place, such as a digital

¹ European Commission *European Blockchain Services Infrastructure, Home - EBSI* -. Available at: <https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home> (last accessed: 09 July 2023).

wallet application installed on a smartphone. In such a way that it is the user who manages what data to share and with whom, thus eliminating the third parties that store and decide how to share this data.

In this new paradigm, users possess applications called **self-sovereign identity wallets**, where they **store verifiable credentials in a centralized and digital manner**. These applications function as **virtual wallets** where users hold different verifiable credentials that attest to something, such as identity, driver's license, or academic certificates. These digital verifiable credentials use **asymmetric cryptography** (Public Key Infrastructure) to ensure their security and validity.

This approach does not imply that the data is directly, locally, and physically stored in the wallet of the user. How data is actually stored, will depend on the implementation of the chosen wallet. For instance, the data could be stored by the wallet in a cloud web service, but this implementation detail does not change the following essential key points:

- Wallet solutions are interoperable. The user is able to choose which wallet service to use and store the credentials in any wallet.
- The user is actively choosing which data to share with each third-party platform.
- Wallets and platforms are independent of each other.

These verifiable credentials are generated and issued by platforms implementing the SSI EduWallets system using standards for the generation of **Decentralized Identifiers** (DIDs), and **Verifiable Credentials** (VCs). DIDs are **unique identifiers** that each user can have and generate within the wallets). VCs are documents in **JSON-LD** or **JWT format** that define the information contained in a verifiable credential following a defined **verifiable credential schema**. The credentials schema allows the creation of verifiable credentials with the same body structure, this makes them **interoperable** with other SSI systems.

Once a verifiable credential of a certain type is created, it is shared with the requesting party. The exchange between the issuer and the receiver, in this case, a learning platform and the user's wallet, is done using protocols such as **OIDC and SIOP**.

When a verifiable credential is stored in the user's wallet, the user can subsequently present it to another platform (verifier) or third party that requires a specific type of verifiable credential. Through the verification UI, the user makes a request to create a verifiable presentation (VP), which is a document in JSON-LD or JWT format that **acts as a wrapper for the VCs** that the user wants to deliver to the verifier. The VP includes the user's signature performing the presentation. In this way, VPs contain two different signatures as proof of the whole chain of custody, the original signature from the issuer contained in the VC and the signature of the holder contained in the VP.

Once the verifier receives the verifiable presentation, it verifies the validity of the user's signature in the VP body and then the signature of each VC within the VP ensuring that the content has not been altered. In decentralized systems, this verification is typically done by checking if the DID of the issuer of the verifiable credentials is registered on the ledger along with their public key within the blockchain. This allows the verifier to authenticate a verifiable credential and subsequently verify the correctness of the user's signature by obtaining their public key, usually from the user's DID.

6 Why to use the new paradigm of Web 3.0

The SSI EduWallets provide a series of advantages, these advantages are given by a new paradigm in which **users are the ones who own their data** and not third parties so that users can unify their credentials in one single storage location, the wallet. Examples of credentials could be the national identifier or educational diplomas. In this way, **users can choose which information they want to share and with whom** increasing the user's privacy.

The new paradigm also **improves efficiency** in tasks such as sending verifiable data to the platforms for verification by eliminating third parties that were in charge of carrying out the verification process so that now the recipient is directly in charge of carrying out the verification and then using that information within the platforms, **speeding up the**

process, saving costs and time. When working with digital educational certificates, certain problems derived from the **interoperability, mobility, and forgery** can be removed because once they're digitized they can be shared easily with anyone anytime, they are **interoperable with other systems** just because they are created with a structure under certain standards and a common definition of the content that follows certain rules and they are secure because **asymmetric cryptography** is used for it.

The educational verifiable credentials body definition follows the **ELM v3** (European Learning Model) data model in JSON-LD format to structure and describe the skills and learning opportunities a user gains by completing a course or assessment. The **ESCO** (European Skills, Competences, Qualifications, and Occupations) is used in conjunction with ELM v3 to **classify the learning outcomes** of the users like the skills and the occupations, and with that information, it is possible to classify the knowledge, learning opportunities, and job occupations.

Within the educational verifiable credentials is intended to follow and integrate a Qualification Metadata Schemata (QMS) which documents the skills and qualifications that the person achieves once the course or assessment is completed. The ESCO classification is integrated within the QMS that shows the learning outcomes of the users and with that information it is possible to classify the knowledge, learning opportunities, and job occupations under a European standard.

- Users own their personal data.
- Unify user information in one place.
- Users choose which data they want to share.
- Streamline platform processes like the presentation of documents.
- Achieve interoperability between different SSI systems that use educational verifiable credentials.
- Issue educational verifiable credentials in a standardized manner following a schema.

- Security, the verifiable credentials are secure by Public Key Infrastructure (PKI) encryption, so they're tamper-proof.
- Provide the skills, competencies, qualifications, and occupations that a user has attained upon completion of a course or assessment.

7 SSI EduWallets Components

Wallets

Wallets are the applications where users **store their DIDs and verifiable digital credentials** that users must use to support the SSI EduWallets implementation. Those wallets are developed by third parties under European standards. In the implementation of SSI EduWallets, it is provided a **demo web wallet** that acts as a real web wallet in order to simulate the web wallet workflow, it was also tested the implementation with another compliant wallet provider “**ValidatedID**” and the issuance flow was performed using the cross-device workflow through the scanning of the QR code provided by the issuer platform.

User Interface

The user interface is the part that is **graphically displayed to the users** on platforms that implement SSI EduWallets. This is in charge of graphically showing users an abstraction of the implementation so that users can easily interact with the platform.

The user interface is made up of several graphical components that show the steps of **issuing verifiable credentials and presenting verifiable presentations**. These graphical components make requests to the different APIs to interact and exchange data.

Wallet Kit API

The Wallet Kit API is a third-party integration that uses the SSI EduWallets project to take advantage of the implementation to handle the **entire Web 3.0 and SSI stack**. This API is responsible for DID creation, verifiable credential issuance, verifiable presentations,

verifiable credential security, and the demo web wallet. All those responsibilities are divided into different sets of API endpoints.

This API is not directly accessible but the issuer API and the verifiable API are responsible for communicating with it.

Issuer API (Open API)

The issuer API (Application Programming Interface) is the main component in charge of performing **the issuance process** of the verifiable credentials to the users. This component must be implemented within any platform to allow the issuance of verifiable credentials to the users using a compliant wallet. Once the issuer API is integrated within any platform it can interact directly with the user's wallets and with the wallet kit API to issue an educational verifiable credential to the users.

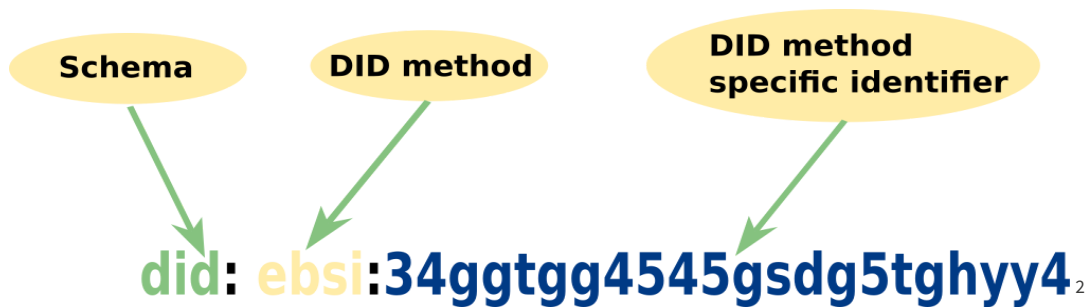
Verifiable API (Open API)

The verifier API is the main component in charge of performing the **verification process of the verifiable credentials** from the users that they present to the platforms. This component must be implemented within any platform to allow the verification of the verifiable presentation from the users that use a compliant wallet. Once the verifier API is integrated within any platform it can interact directly with the user's wallet and with the wallet kit API in order to verify if a verifiable credential is valid or not and then the platform can use the data of the verifiable credentials to perform any actions within the platform.

8 Standards used

8.1 W3C Decentralized Identifiers

Decentralized identifiers (DIDs) are a new type of identifier that enables verifiable, **decentralized digital identifiers**. A DID refers to any subject (e.g., a person, organization, thing, data model, abstract entity, etc.)



DIDs are the cornerstone of self-sovereign identity (SSI). DIDs are **URL-based identifiers** associated with an entity; a DID is just a long string that does not provide any meaningful information about a natural or legal entity. DIDs and DID Documents are generated by their owners with their wallet or back-office systems, these identifiers are most often used in a **verifiable credential** and they are associated with subjects such that a verifiable credential itself can be easily ported from one repository to another without the need to reissue the credential.

The DIDs are composed of the schema or “did:” which is the first part of the definition of a DID, the method that is a mechanism or protocol used to create and manage unique and decentralized identifiers and the DID method-specific identifier that is a completely unique random number that follows method-specific generation rules.

8.2 W3C Verifiable Credentials Data Model

Credentials are a part of our daily lives; driver's licenses are used to assert that we are capable of operating a motor vehicle, university degrees can be used to assert our level of education, and government-issued passports enable us to travel between countries. Verifiable credentials with DIDs are the core of the SSI paradigm, the verifiable credentials provide a **mechanism to express these sorts of credentials digitally on the Web** normally in the format of **JSON-LD or JWT** in a way that is **cryptographically secure, privacy-respecting, and machine-verifiable**.

The verifiable credentials got features like:

² Decentralized Identifier parts

- **Portability:** due they are digital documents that can be safely shared with third parties.
- **Interoperability:** because they are created in the same formats like JSON-LD or JWT and the definition is based on a schema that can be a public standard.
- **Security:** VCs use asymmetric encryption and signatures to keep safe and tamper-proof the credentials.
- **Selective disclosure:** the individual has control over which pieces of information they share in a given context. They can selectively disclose specific attributes without revealing unnecessary personal data.

9 Verifiable credentials security

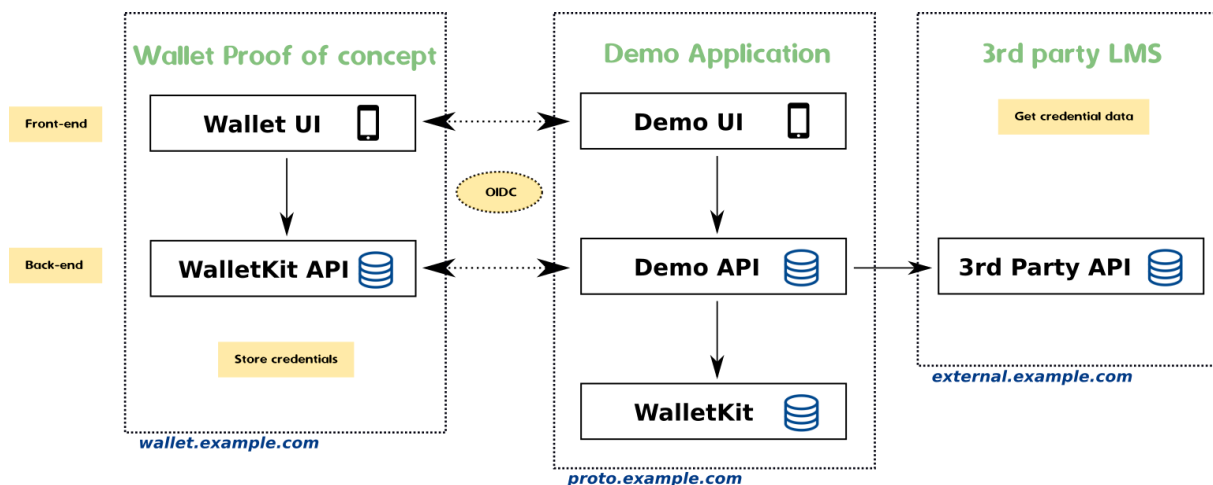
The verifiable credentials are secure and tamper-proof due to the use of asymmetric cryptography (PKI) in which the issuer signs the VC that will be issued to any user with its private key and also a hash of the content of the verifiable credential, this way the verifiable credential can be verified by any third party just by seeking for the DID and public key of the issuer of the VC, in the case of European Blockchain Services Infrastructure (EBSI) ecosystem, the verifier will retrieve the public key and the DID of the issuer from the EBSI ledger. The verifier with the public key that it has just retrieved from the EBSI ledger can decrypt the content of the VC. Then it can hash the content in order to obtain a new hash and compare both two to check if it is the same hash. If the hashes are equal and the verifier can decrypt the content of the VC then the VC is valid³.

When a user creates a VP (verifiable presentation) this will create a wrapper of VCs that will be signed with the private key of the user. The verifier is able to get the public key of the user that is encoded on her/his DID and can decrypt the content of the VP.

³ Decentralised Identifiers DIDs - Walt.id. Available at:
<https://docs.walt.id/v/ssikit/ssi-kit/what-is-ssi/technologies-and-concepts/decentralised-identifiers-dids>
(last accessed: 15 jul 2023)

10 Architecture and deployment

The system is divided into two main applications: **Wallet** and **Demo application**. Each of them is divided into smaller components as described below.



4

10.1 Web Wallet proof of concept

The demo web wallet application provided within the project can be replaced with any other web wallet solution compatible with [ESSIF/EBSI](#)⁵ standards. There are many wallet solutions compatible with our system. For example, the commercial wallet from [ValidatedID](#)⁶ was successfully tested with the project.

The goal of this project is not to develop a new standalone wallet application, but an **open-source wallet solution** was still needed in order to proceed with the implementation and be able to provide a full stack **proof of concept** of the whole project.

⁴ Demo application architecture

⁵ Pastor Matut, C. and Du Seuil, D. *Understanding the European self-sovereign identity framework (ESSIF)*, PPT. Available at: <https://www.slideshare.net/SSIMeetup/understanding-the-european-selfsovereign-identity-framework-essif> (last accessed: 08 July 2023).

⁶ ValidatedID *Validated ID - electronic signature and digital identity providers, Validated ID - Electronic Signature and Digital Identity Providers*. Available at: <https://www.validatedid.com/en> (last accessed: 08 July 2023).

An **extension** of [walt.id Wallet Kit](#)⁷ was implemented and the source code was made available as a public [git repository SSI EduWallets/Wallet Proof Of Concept](#)⁸.

This web wallet application is integrated with the **Demo Application** using links, but it is just a proof of concept, it is intended for demo purposes only and it should not be used in production settings.

Wallet deployment

This demo application is divided into **frontend** and **backend**. Both components are **dockerized** and can be run and deployed using [Kubernetes](#)⁹ and the [Helm chart](#)¹⁰ provided in the git repository. This allows easy deployment and test of the whole application.

The main file with the **configuration** of the Helm chart is located on the path `helm/wallet/values.yaml` of the repository. In this file, the key `wallet.issuer.url` should be changed to the domain where the demo verifier application is going to be served.

Wallet backend

The wallet backend is a plain deployment of [walt.id Wallet Kit](#). A publicly available image is available in [walt.id dockerhub](#)¹¹ under the tag `waltid/walletkit:latest`

⁷ Walt.id, *Walt.id Wallet Kit*, *walt.id*. Available at: <https://github.com/walt-id/waltid-walletkit> (last accessed: 08 July 2023).

⁸ SSI EduWallets / *Wallet Proof Of Concept*. Available at: <https://gitlab.com/ssi-edu-wallets/wallet-proof-of-concept> (last accessed: 08 July 2023).

⁹ Kubernetes *Production-grade container orchestration*, *Kubernetes*. Available at: <https://kubernetes.io/> (last accessed: 09 July 2023).

¹⁰ Helm *Helm*. Available at: <https://helm.sh/> (last accessed: 09 July 2023).

¹¹ Walt.id *Wallet Kit Image*, *Docker*. Available at: <https://hub.docker.com/r/waltid/walletkit/tags> (last accessed: 08 July 2023).

For extra configuration or more details, refer to the [official documentation](#)¹² from wallet-kit.

Demo wallet UI

The demo wallet front-end is a **whitelabel** extension of the [web wallet provided by Walt.id](#)¹³. It is written using the [Nuxt](#)¹⁴ framework and the source code is available in the git repository of the [git repository SSI EduWallets/Wallet Proof Of Concept](#)¹⁵.

A **docker image** is available on the container registry of the same repository and can be pulled publicly: `docker pull`

`registry.gitlab.com/ssi-edu-wallets/wallet-proof-of-concept:wallet-ui-0.5`

10.2 Demo Application

Two different use cases were implemented in this application.

On one hand, the issuer will be able to **issue VCs** ([verifiable credentials](#)¹⁶) that can be stored in the user's wallet. The issuer could be getting the information stored in a 3rd party API, for example, an LMS in order to obtain the data that is going to be contained in the VC.

In the implemented **issuance use case**, the demo application is getting a list of learning resources experienced by the authenticated user and it is issuing a VC (type `UserLearningOutcomes`) containing the list of learning resources.

¹² Walt.id *Introduction, Docs*. Available at: <https://docs.walt.id/v/web-wallet/wallet-kit/readme> (last accessed: 08 July 2023).

¹³ Walt.id *Walt.id Web wallet, walt.id*. Available at: <https://github.com/walt-id/waltid-web-wallet> (last accessed: 08 July 2023).

¹⁴ Nuxt *The intuitive web framework, Nuxt*. Available at: <https://nuxt.com/> (last accessed: 12 July 2023).

¹⁵ SSI EduWallets / *Wallet Proof Of Concept*. Available at: <https://gitlab.com/ssi-edu-wallets/wallet-proof-of-concept> (last accessed: 08 July 2023).

¹⁶ Sporny, M., Longley, D. and Chadwick, D. *Verifiable credentials data model V1.1, W3C*. Available at: <https://www.w3.org/TR/vc-data-model/#abstract> (last accessed: 08 July 2023).

On the other hand, the verifier that will **request VPs** (verifiable presentations) from the verify UI that will request a specific type of VCs to the user's wallet, and then once the user chose the VCs to share it sends the VP to the verifier and then it will check the cryptographic signatures. The data from the VC should be processed and may be forwarded a 3rd party application. It is the responsibility of the 3rd party application to do whatever is required with the data contained in the VP.

In the implemented **verifier use case**, the demo application is getting a VP (type **UserLearningOutcomes**, the same type generated by the issuer workflow) and it is processing the data in order to build a basic profile from the user.

Both use cases were implemented on the same **backend API**, but they are independent from each other. This means that they could run on different **3rd party platforms** in order to **exchange** verifiable credentials between both of them. They can also be extended to generate any type of verifiable credential in order to store any data under the W3C VCs specification.

The Demo Application is divided into 2 main components: the **UI** and the **API**. The API is also connected to a standalone instance of the **walletkit API** in order to use basic functions already implemented like verification of credentials, issuance or exchange of verifiable credentials through the protocol [OIDC \(OpenID Connect\)](#)¹⁷ and it is also connected to a 3rd party API to provide information to be stored in the VCs after issuance.

Demo Application UI

The **UI of the demo application** is a development in order to implement a graphical interface (UI) through which the users can interact with the **demo API (Open API)** to perform both the **verifiable credential issuance** and the **verification of verifiable presentations**.

¹⁷ T. Lodderstedt, K. Yasuda, T. Looker (03/02/2023), *OpenID for Verifiable Credential Issuance*. Available at: https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html#name-introduction.

Architecture

The implementation of this **UI** application has been done using [Vue.js](#)¹⁸ frontend framework, which generates a scaffolding of a frontend application divided into directories and configuration files that compose the whole UI application with the finality of running over a [Node.js](#)¹⁹ server.

Once the UI application is ready to be deployed it is compiled within a Node.js server and then dockerized in order to be deployed using Kubernetes. This UI application is part of a set of applications that form a stack in order to build the final system (**Demo application**).

UI pages, workflows & requests

The implementation of the Demo Application UI is composed of the following pages or views under the "views" directory:

Issuance page

The first is the **issuance page**, from which the flow of issuance of verifiable credentials to the users is carried out. Once the user reaches this page, an **HTTP GET** request is made to the demo application API in order to obtain the types of verifiable credentials that the issuer can issue to users. The request is made to the domain that this UI establishes in its configuration such as:

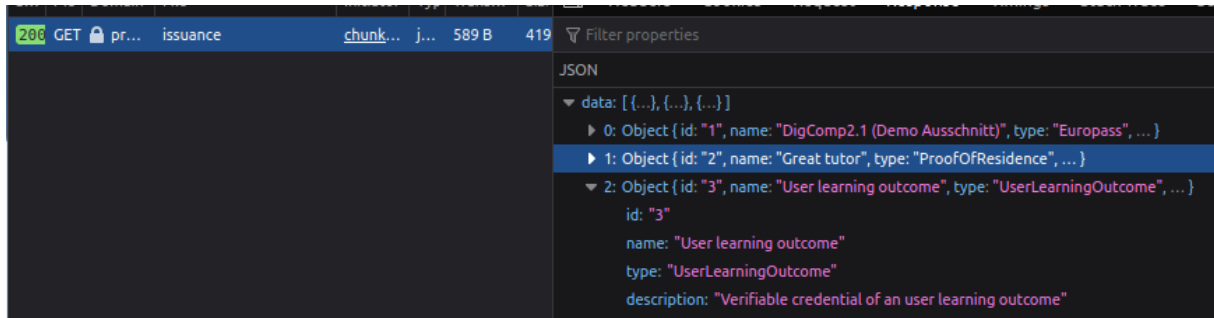
Unset

<https://proto.example.com/edu/api/v1/wallet/users/3/issuance>

This last one is the endpoint on which the demo application API is listening. The response from this endpoint will be an array of objects in JSON format, each of these objects corresponding to the type of verifiable credentials that can be issued.

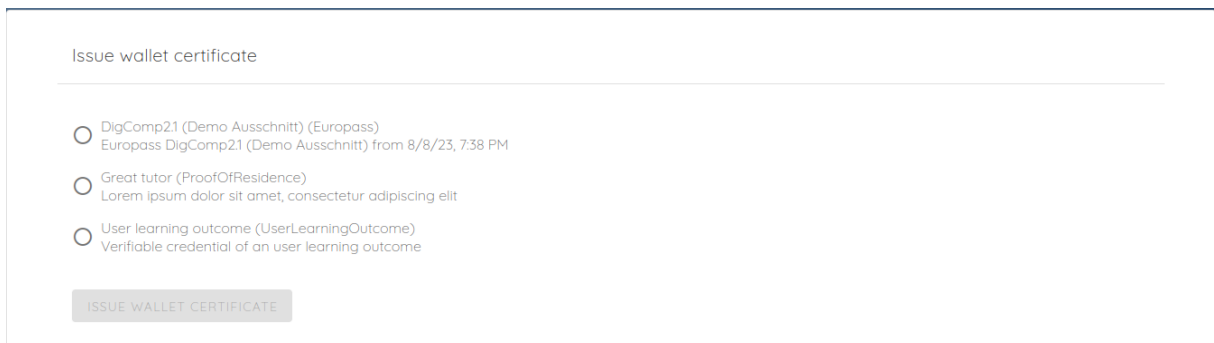
¹⁸ Vue.js *The progressive javascript framework*, *Vue.js - The Progressive JavaScript Framework* | *Vue.js*. Available at: <https://vuejs.org/> (last accessed: 12 July 2023).

¹⁹ Node.js *Node.js*. Available at: <https://nodejs.org/en> (last accessed: 12 July 2023).



20

Once the different **types of verifiable credentials** that can be issued have been obtained, the UI shows a series of radio buttons with the information of the verifiable credentials that it can be issued and a disabled button that performs the issuance over the web flow. Some of these verifiable credentials already exist within the wallet kit API by default the only one that was totally implemented for us through the demo application API is the **“userLearningOutcomes”** verifiable credential in order to showcase a proof of concept of the project over the issuance and verification of an educational verifiable credential.



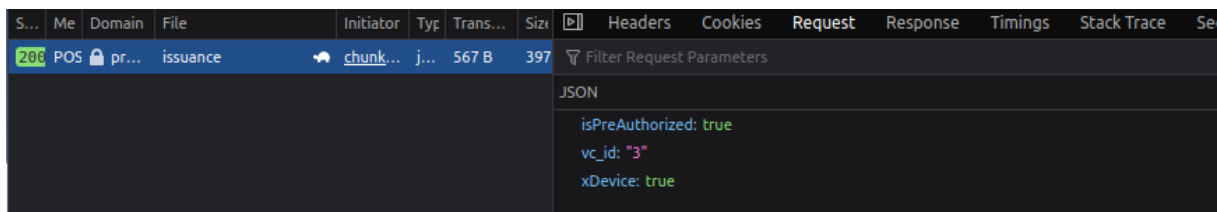
21

When the user selects an option, an **HTTP POST request** is sent to the same endpoint as before of the demo application API, this time, the request contains a **JSON object** with certain parameters **as payload**, such as the **id of the verifiable credential** to be issued and the **type of issuance flow** that can be the web wallet **"xDevice: false"** or cross-device **"xDevice: true"**, in the first case this field is sent with the value **true** and the response of the demo API will be a **JSON object** that will contain a **redirection URI** in order to **encode**

²⁰ Request to get the VC types that can be issued.

²¹ Issuer portal UI

this URI in the **QR code** that is generated by the UI, so just by scanning this QR code from a compatible wallet such as "[ValidatedID](#)²²" the verifiable credential will be issued and exchanged with the user's wallet using the [OIDC4VC](#)²³ protocol and then in the user's wallet it will receive a verifiable credential issuance request from the issuer where the user can check and read the content of the verifiable credential before accepting it on the wallet.



24

In the issuance UI, once the verifiable credential to be issued is chosen, an issue button is shown which is in charge of making the same **previous request** but **changing** the value of the "**xDevice**" field to **false**, this way the demo application API **behaves differently** if the user clicks the button then an issuance web wallet flow is started.


²² ValidatedId Validated ID - electronic signature and digital identity providers, Validated ID - Electronic Signature and Digital Identity Providers. Available at: <https://www.validatedid.com/en> (last accessed: 08 July 2023).

²³ T. Lodderstedt, K. Yasuda, T. Looker (03/02/2023), OpenID for Verifiable Credential Issuance. Available at: https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html#name-introduction.

²⁴ Issuance request for the cross device flow

Issue wallet certificate

Scan the Qr code to start the issuance



☐ DigComp21 (Demo Ausschnitt) (Europass)
Europass DigComp21 (Demo Ausschnitt) from 8/8/23, 7:38 PM
 ☐ Great tutor (ProofOfResidence)
Lorem ipsum dolor sit amet, consectetur adipiscing elit
 ☒ User learning outcome (UserLearningOutcome)
Verifiable credential of an user learning outcome

ISSUE WALLET CERTIFICATE

25

S...	Me	Domain	File	Initiator	Typ	Trans...	Size	Headers	Cookies	Request	Response	Timings	Stack Trace	Security
206	POS	pr...	issuance	chunk...	j...	596 B	426	Filter Request Parameters						
302	GET	wa...	/api/siop/initiateissuance	860.f...	htm	servic...	4.07	JSON						Raw
206	GET	wa...	/initiateissuance/7session	docu...	htm	servic...	4.07	isPreAuthorized: true						
206	GET	wa...	favicon.ico	Favico...	htm	servic...	4.07	vc_id: "3"						
								xDevice: false						

26

Once the **redirection URI** is obtained from the response of the request, the issuer UI redirects the user to this URI, which is in the pre-configured **web wallet**, so the verifiable credential will be issued and exchanged with the user's wallet using the **OIDC4VC** protocol. Then the user's wallet will start the verifiable credential issuance request and the user needs to approve it.

The issuer will retrieve the information needed to add it to the credential. For example, in case of the issuance of the verifiable credential is of type "**UserLearningOutcomes**" then the demo application API internally gets **the information** about the **completed courses** by the user and creates the VC with them formatting the data according to the VC schema.

²⁵ Issuance portal UI, issuance process

²⁶ Issuance request payload from the web wallet flow

Verify page

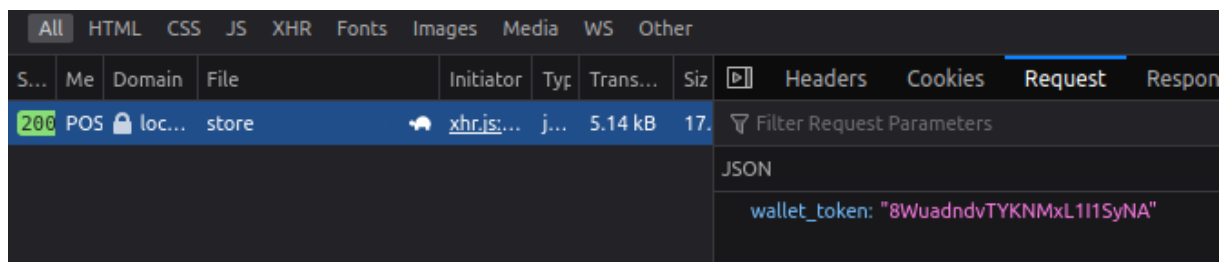
The second main workflow starts with the **verify page**, from which the verification flow is carried out. Once the user will be displayed with an “import” **button** that will start the verification of a **"userLearningOutcomes"** credential through the verification of a **verifiable presentation**.

As soon as the flow is started, the user is redirected to the verifier UI on the page (by default **/verify/success** and this is **not configurable** at the moment) with an **access token** as a parameter in the URL, the response of the demo application API redirects to the next URL:

Unset
https://proto.example.com/assess/en/verify/success/?access_token=8WuadndvTYKNMxL1I1SyNA

From this page, the verification UI reads the **access token** from the redirection URL then it will send a **HTTP Post request** to the demo application API to the endpoint

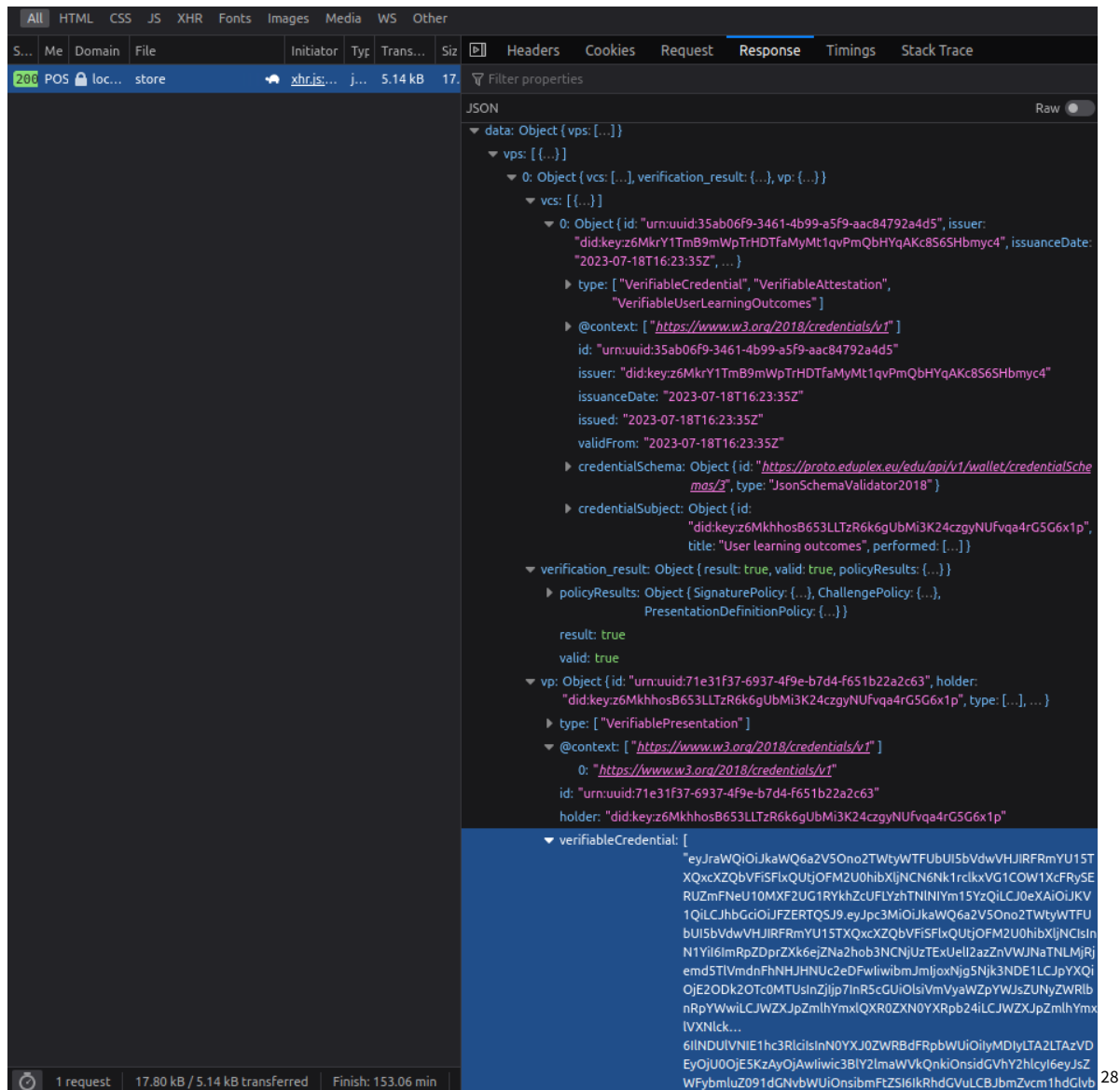
Unset
<https://proto.example.com/edu/api/v1/wallet/verifier/store>



27

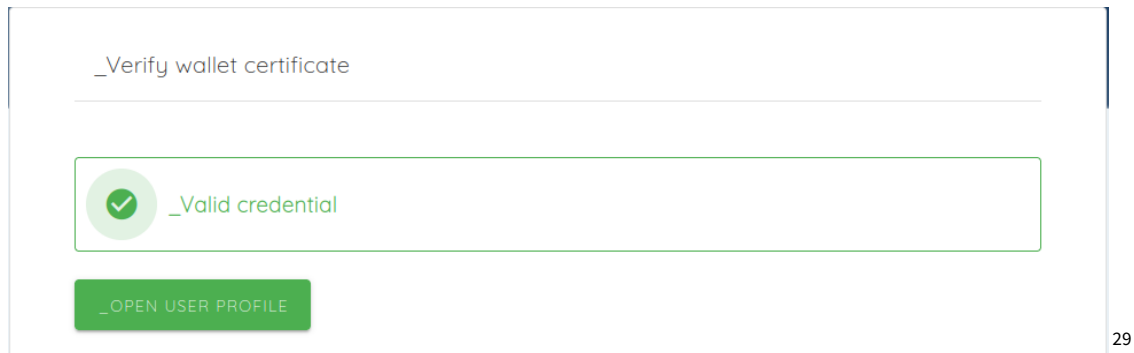
The API will use the access token of the URL parameter to verify the VP, if the VP is valid, it will retrieve the data from the VP and process it to store it as an “imported” user profile. The response of the request is a JSON object with the content of the VP.

²⁷ Verification presentation payload

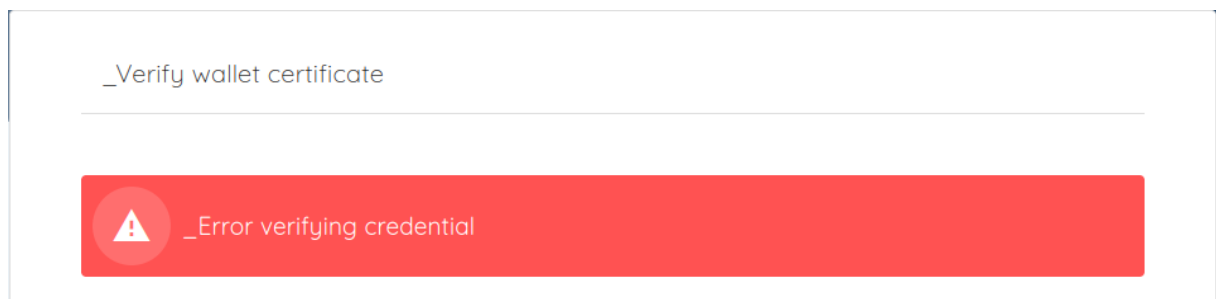


In this case, the content of the response is not displayed to the user through the verification UI, instead, just a success message is displayed together with a button which will show the newly imported profile.

²⁸ Verification presentation response



If the token is not valid then the verification UI will show an error message.



30

UI components, workflows & requests

Three UI components have been created using the Javascript framework [Vue.js](https://vuejs.org/)³¹, two of them in the form of a button. They are in charge of starting the flows of issuance and verification of verifiable credentials. The third component displays a modal in which the user can check their learning outcomes profile and perform the verification and importation of data from a “**userLearningOutcomes**” verifiable credential type from the user's wallet. These components can be imported anywhere else on the platform, such as a custom view or page.

The "issueCredentialsBtn" component

1. The "issueCredentialsBtn" component shows a button that, when clicked on, redirects to the "issuance" view.

²⁹ Valid verifiable presentation message in the verification UI

³⁰ Invalid verifiable presentation message in the verification UI

³¹ Vue.js *The progressive javascript framework*, *Vue.js - The Progressive JavaScript Framework* | *Vue.js*. Available at: <https://vuejs.org/> (last accessed: 12 July 2023).

The "VerifyCredentialsBtn" component

- The "**VerifyCredentialsBtn**" component shows a button with an icon which is currently a link that, when clicked on, makes an **HTTP GET** request to the demo API. This request is made towards an established URL such as

Unset

```
https://proto.example.edu/api/v1/wallet/verifier/present?walletId=YOUR_WALLET_ID&vcType=VerifiableUserLearningOutcomes
```

This request includes the id of the wallet used and the type of the verifiable credential to be verified, in this case, the parameter of the wallet is configured within the configuration of the API wallet kit through the file `verifier-config.json` in the field "wallets" and the type of verifiable credential is "VerifiableUserLearningOutcomes" which is hardcoded in the redirection URL. The demo application was intended to simulate the flow of the issuance and verification of this type of verifiable credential but it can be easily extended in order to use any other credential type.

After this request, the user will be redirected to the web wallet, in order to create a verifiable presentation of the selected verifiable credential that was in the "vcType" parameter. Once the user from her/his wallet choose the VC that she/he wants to present to the verifier and accepts the creation of the verifiable presentation then, he/she is redirected to the verifier platform into the verification UI in

Unset

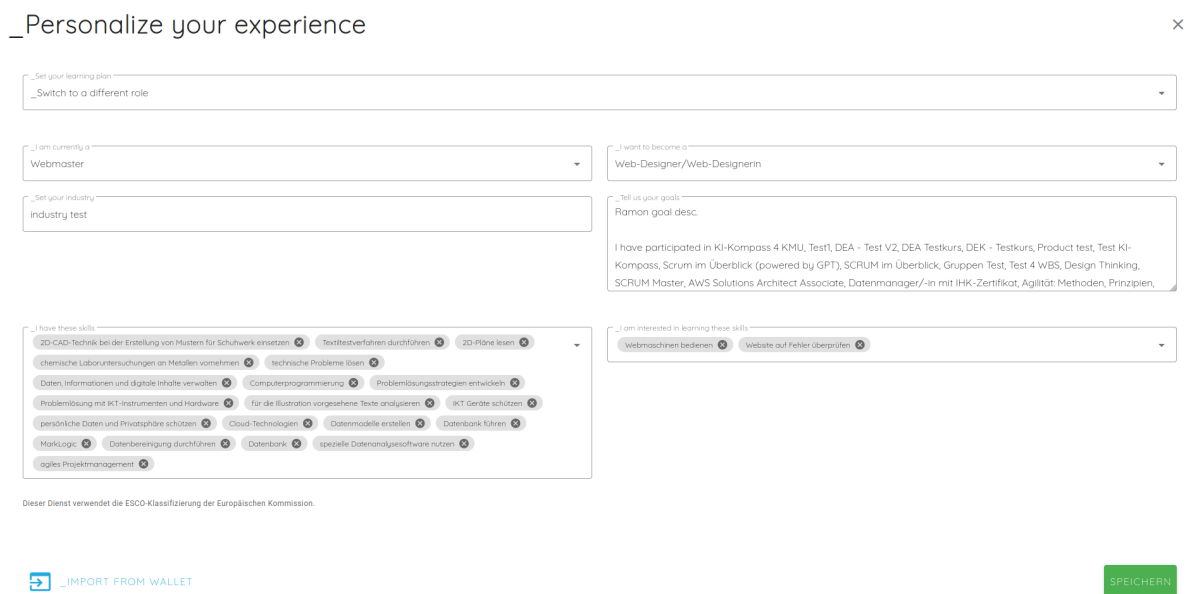
```
verify/success/?access_token=7vT6I4iDSLej-XYbisKAVw
```

The "userProfileDialog" component

- The "**userProfileDialog**" component is composed of a button that, when clicked on, will open a **modal window** that will show a series of fields related to the **ESCO skills obtained** through courses that the user has been completing. When this modal is opened, a request is made to the **third-party API** in order to obtain the user profile data that has been created using the data imported previously of a

"**userLearningOutcomes**" verifiable credential from the user's wallet. Also, if there are defined **ESCO skills**, other requests will be executed to a third-party API in order to obtain information about the possible occupations of each **ESCO skill**.

In addition to these fields, the modal window contains **two buttons** at the bottom, the first is related to the import and validation of the "**userLearningOutcomes**" verifiable credentials that will be imported from the user's wallet, this button is the "**verifyCredentialsBtn**" component, which is in charge of perform verifiable credentials verification flow. The other button is in charge of **saving** the changes that are performed within the fields of this UI component, so once it is clicked it will send an **HTTP request** to the third-party API.



32

How to run

In order to access the issuance and verification graphical interface and make use of the UI components, it is necessary to put in production this repository that can be found [here](#) running under a node.js server in a [docker](#)³³ container for which the project must be

³² User profile UI

³³ Docker (2023) *Accelerated Container Application Development*, Docker. Available at: <https://www.docker.com/> (last accessed: 12 July 2023).

compiled using the yarn package manager with the "**yarn build**" command or "**yarn generate**" in order to create a static project and then deploy it over a docker container, at the end it is only needed to run the container and execute the command "**yarn start**" to use the application.

The other option as a developer is to run the repository in **development mode**, for this it can be done by opening a terminal in the root of the repository and already with node installed at least with version **16.19.1** and with the [yarn package manager](#)³⁴ we execute the "**yarn install**" commands if it is the first time that we run the project, so that the appropriate dependencies will be installed and the set up it is done. Once the last command is finished, the "**yarn dev**" command will be launched to launch the project in development mode.

Once it is running, we can access the pages by putting in the url the next:

```
Unset
{ YOUR_DOMAIN }/assess/ { CURRENT_LANG }/issuance or /verify/success
```

so that the UIs implemented for certain cases will be displayed to the user.

Demo Application API

The **API of the demo application** is a development in order to implement an **Open API** that acts as an intermediary between the user and the **Wallet Kit API** this implementation was necessary in order to customize and manage the request that is forwarded to the Wallet Kit API, this provides an additional **abstraction layer** and **security** because the Wallet Kit API has no private endpoints.

Functionalities

The Demo Application API performs the following tasks:

- Receive users' issuance requests and forward them to the **Wallet Kit API** in order to issue the different verifiable credentials types that it allows:

³⁴ Yarn Home Page, Yarn. Available at: <https://yarnpkg.com/> (last accessed: 12 July 2023).

- **Europass**
- **ProofOfResidence**
- **UserLearningOutcomes**
- Receive user verifiable presentation with the VC type of “**userLearningOutcomes**” requests and forward it to the Wallet Kit API.
- Receive users' requests in order to return the **verifiable credential schemas** linked to each VC type.
- Send requests to **third parties APIs** in order to manage user profiles.

Architecture

The implementation of the **Demo Application API** has been done through the use of [Cake PHP 4](#)³⁵ backend framework which generates a scaffolding of a backend application divided into directories and configuration files that compose the whole API application with the finality of running it over a [Nginx](#)³⁶ server within a docker container in order to manage the user request. The **Demo Application API** is served under the directory “**Wallet**” which is within “**app_rest**” and within “**plugins**” directories, the configurations to run the API and the documentation related can be found on the [demo-api](#) repository.

Once the Demo Application API is ready to be deployed it is dockerized and running within an Nginx server, and then it is deployed using Kubernetes. This Demo Application API is part of a set of applications that form a stack in order to build the final system (**Demo application**).

Software components

The **Demo Application API** or **Open API** is made up of a series of software components that implement the necessary endpoints for users to make use of these **HTTP network requests** through a UI (Demo Application UI). These components called **controllers** are in

³⁵ Cake PHP. *CakePHP cookbook Archivo de Documentación, creado por, Bienvenido - 4.x*. Available at: <https://book.cakephp.org/4/es/index.html> (last accessed: 12 July 2023).

³⁶ Nginx (2023) *Advanced load balancer, web server, & reverse proxy, NGINX*. Available at: <https://www.nginx.com/> (last accessed: 12 July 2023).

charge of defining the routing and managing of the network requests within the API in order to interact with the correct data model. These controllers are a part of the Model View Controller (MVC) design pattern.

Main controllers and workflows should be:

- **Issuance flow:**
 - **WalletIssuanceController:** This controller is responsible for the handling of the issuance requests and perform the next requests:
 - WalletIssuance: **List** and **issue** VCs
 - WalletIssuerOIDC: Set of endpoints to **enable OIDC** issuer flow (proxy forwarded to Wallet Kit API)
- **Verifier flow:**
 - **VerifierPresentProxyController:** This controller is responsible for the handling of the verifiable presentation requests:
 - VerifierPresentProxy: **Init VPs** in order to **forward** to the Wallet Kit API.
 - **VerifierVerifyProxyController:** This controller is responsible for the handling of the verification requests:
 - VerifierVerifyProxy: **Forward verification requests** to Wallet Kit API.
 - **VerifierStoreController:** This controller is responsible for the handling of the verifiable presentations in order to manage the data of the VCs and **store** them in a third-party API and perform the next requests:
 - VerifierStore: **Process** and **store** the data of a “**userLearningOutcomes**” verifiable credential in a third party API.
- **Other controllers:**
 - **WalletCredentialsSchemaController:** This controller is responsible for the handling of the [verifiable credentials schema](https://www.w3.org/TR/vc-json-schema/)³⁷ requests:

³⁷ Cohen, G. and Steele, O. (2023) *Verifiable credentials JSON schema specification, W3C*. Available at: <https://www.w3.org/TR/vc-json-schema/> (last accessed: 09 July 2023)

- **WalletCredentialSchema:** Retrieve the schema definition of a verifiable credential.
- **UserProfiles:** Simple CRUD endpoint of a third party API in order to manage user profiles.

Endpoints can be found documented in [Open API documentation](#)

Some of the endpoints are **directly forwarded** to the **standalone** deployment of Wallet Kit API. Since **Wallet Kit does not perform any kind of authentication, all walletkit endpoints must be running in a private network**. A simple [kubectlyaml](#) is included in the repository to allow easier deployment.

The controllers with the name “**Proxy**” perform a simple forward of the request via PHP. Besides this, a proxy forward needs to be set up for the endpoint `/issuer-api/default/oidc`. This can be easily accomplished using software like [Nginx](#) or using an ingress controller within Kubernetes. The rules definition will look similar to:

```
- path: /issuer-api/default/oidc/.*$
  pathType: Prefix
  backend:
    service:
      name: walletkit-api-svc
      port:
        number: 80
```

How to run

To run the Demo Application API, first it is needed to put it into production running under an **Nginx** server which, through **docker**, is encapsulated in a container where it can be configured and run, to perform the production ready deployment of the API it has been used **docker**, **kubernetes** and **helm** technologies to achieve this, the deployment of the Demo Application API is carried out on a server in the cloud from where the API services

will be provided. To launch or configure the API locally, it can be done through the "`docker-compose-dev.yaml`" file that is located in the root of the repository.

for further information about how to configure and run the API a more detailed documentation can be found on the [Demo Application API](#) repository.

Demo Application Web wallet

The **demo application web wallet** is a **pre-built** frontend application that performs the tasks of a user's web wallet application, it handles **issuance** of verifiable credentials, **verifiable presentations**, performs the **VC exchanging process** between issuers, verifiers and the web wallet itself and **stores the VCs and manages them**. This demo application was **customized** in order to display the information of the new verifiable credential that was created, the "**userLearningOutcomes**", this pre-built solution that acts as a demo web wallet in which a user can make use of it accessing a login UI which the login process is mocked up, in order to test a real workflow.

The demo application web wallet UI implementation and the documentation attached to it can be found in the [Web Wallet](#)³⁸ repository provided by **Walt.id** to make changes and customize.

Architecture

This **UI** was implemented within the **Wallet Kit API** from "walt.id",³⁹ this way the Wallet Kit API provides some UIs in order to perform a full solution, in this case the **issuance** and **verification** UI portals was skipped and only the web wallet UI was customized, this UI perform requests directly to the Wallet Kit API which it also acts as its own API. In the implementation of the **Demo Application** the demo web wallet was **customized** and a image of it was **built** and it is available in the repository

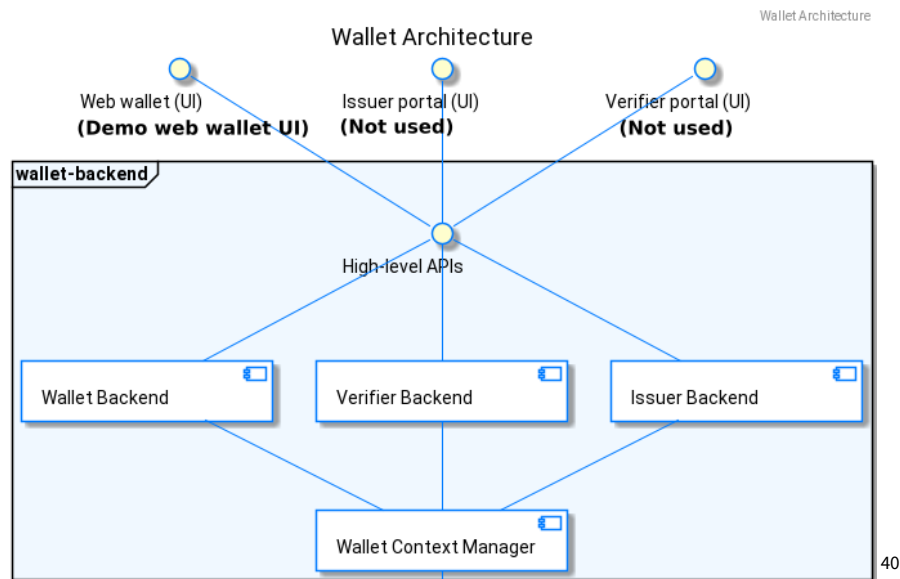
https://gitlab.com/ssi-edu-wallets/wallet-proof-of-concept/container_registry/4437224

³⁸ Walt.id, *Walt.id Web wallet*, walt.id. Available at: <https://github.com/walt-id/waltid-web-wallet> (last accessed: 08 July 2023).

³⁹ Walt.id, *Walt.id Wallet Kit*, walt.id. Available at: <https://github.com/walt-id/waltid-walletkit> (last accessed: 08 July 2023).

with the tag `wallet-ui-0.5`, this image it is being used within the configuration of the **Wallet Kit API** in order to use this web wallet UI.

The image below displays the architecture of the demo web wallet UI and the other pre-built UI components within the Wallet Kit API:



Issuance workflow & requests

The issuance flow starts once a user clicks the **issuance button** from the **issuance UI**, then a **HTTP POST request** will be sent to the **Demo Application API** in order to start the issuance flow and then the response of this request that is a redirection URL will redirect the user to web wallet UI that the issuer platform configure.

Then from the web wallet UI the URL from the previously request response it is sent to the Wallet Kit API and the response of it is a redirection to the “**initiateIssuance**” page of the web wallet UI with the parameter “**sessionId**”, the request is like the next one:

⁴⁰ Walt.id Architecture - Demo web wallet architecture, Docs. Available at: <https://docs.walt.id/v/web-wallet/wallet-kit/issuer-and-verifier-portals/architecture> (last accessed: 12 July 2023).

Unset

```
https://wallet.example.com/api/siop/initiateIssuance/?issuer=https://proto.example.com/issuer-api/default/oidc/&credential_type=UserLearningOutcomes&pre-authorized_code=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI5ODh1MjJlZC04MTc4LTQ1YmUtYTl1Ni0xM2ZjY2NmNWJhNDUiLCJwcmUtYXV0aG9yaXplZCI6dHJ1ZX0.iYwaTegQ4JoesfyigDNqNoy5AgIKkrAqSIn1qVuqhUA&user_pin_required=false
```

and the response is like this:

Unset

```
https://wallet.example.com/InitiateIssuance/?sessionId=8b450ebe-eb6d-4b16-8bb7-768cc37c7519
```

Then the web wallet UI send another HTTP GET request in order to retrieve the list of the DIDs:

Unset

```
https://wallet.example.com/api/wallet/did/list
```

and the response of it:

Unset

```
[{"did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p"}]
```

Also it is sending another HTTP GET request in order to retrieve the information of the VC that it has been issued

Unset

```
https://wallet.example.com/api/wallet/issuance/info?sessionId=8b450ebe-eb6d-4b16-8bb7-768cc37c7519
```

and the response of it:

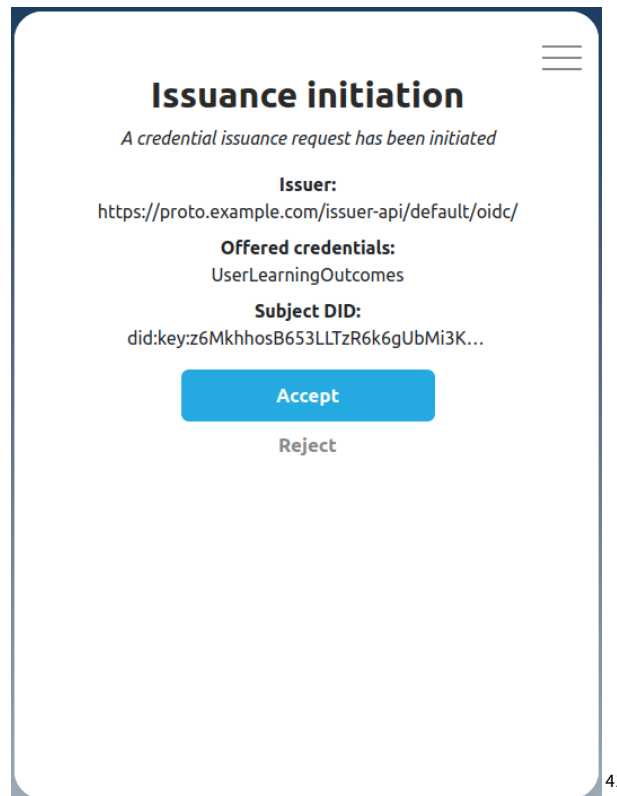
Unset

```
{  
  "credentialTypes": ["UserLearningOutcomes"],  
  "credentials": null,  
}
```



```
"did" : null,  
"id" : "8b450ebe-eb6d-4b16-8bb7-768cc37c7519",  
"isIssuerInitiated" : true,  
"isPreAuthorized" : true,  
"issuerId" : "https://proto.example.com/issuer-api/default/oidc/",  
"lastTokenUpdate" : null, "nonce" : "81d50896-43b3-4280-9f06-37485b604461",  
"opState" : null,  
"preAuthzCode" :  
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI5ODhlMjJlZC04MTc4LTQ1YmUtYT1  
lNi0xM2ZjY2NmNWJhNDUiLCJwcmUtYXV0aG9yaXplZCI6dHJ1ZX0.iYwaTegQ4JoesfyigDNqNoy5  
AgIKkrAqSIn1qVuqhUA",  
"tokenNonce" : null,  
"tokens" : null,  
"user" : null,  
"userPinRequired" : false,  
"walletRedirectUri" : null  
}
```

Then the web wallet UI displays the “Issuance initiation” view.



Once the user **accepts** the issuance initiation the next **HTTP GET** request is sent to the Wallet Kit API in order to save the VC in the web wallet, retrieve the session id of the transaction and the page of the web wallet UI

Unset

GET

```
https://wallet.example.com/api/wallet/issuance/continueIssuerInitiatedIssuance?did=did:key:z6MkhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p&sessionId=424d5584-8e4f-41e9-9563-84a70a2d10a9
```

And the response is the next:

Unset

```
/ReceiveCredential/?sessionId=424d5584-8e4f-41e9-9563-84a70a2d10a9
```

Then the user is redirected to the web wallet page from the response of the request.

⁴¹ Walt.id, *Walt.id Web wallet - issuance initiation, walt.id*. Available at: <https://github.com/walt-id/waltid-web-wallet> (last accessed: 08 July 2023).

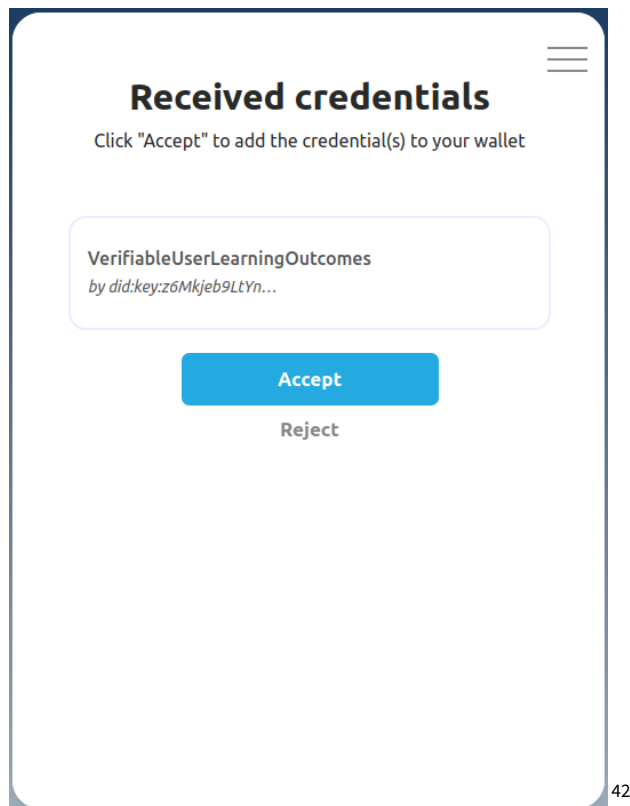
Also another **HTTP GET** request is sent to the Wallet Kit API in order to retrieve the data of the VC that has been issued.

```
Unset
https://wallet.example.com/api/wallet/issuance/info?sessionId=424d5584-8e4f-41e9-9563-84a70a2d10a9
```

And the response of it is the next:

```
Unset
{
  "credentialTypes": ["UserLearningOutcomes"],
  "credentials": [{"The VC that has been issued"}],
  "did": "did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p",
  "id": "424d5584-8e4f-41e9-9563-84a70a2d10a9",
  "isIssuerInitiated": true,
  "isPreAuthorized": true,
  "issuerId": "https://proto.example.com/issuer-api/default/oidc/",
  "lastTokenUpdate": {},
  "nonce": "dd07002b-c7ec-4711-9f50-be7ff00e0729",
  "opState": null,
  "preAuthzCode":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJmNDdmNzhiOC0wMzIzLTQ0YWQtOGZlMS0zMWQ4NTM0ODg0NjEiLCJwcmUtYXV0aG9yaXplZCI6dHJ1ZX0.bVwZBDTUxpVUoLvukX_aznr0FW0f1-IguWecvJdZdM8",
  "tokenNonce": "322b3880-1099-4b56-9f6f-0991d7ffcb0a",
  "tokens": {},
  "user": {"Web wallet user data"},
  "userPinRequired": false,
  "walletRedirectUri": null
}
```

Then in the Web wallet UI we can check the VC that has been issued.



Once the user accepts the VC it is redirected to the “**Credentials**” page of the web wallet UI, and then a **HTTP GET** request is sent in order to retrieve the VCs that the user has stored in her/his wallet.

```
Unset
GET
https://wallet.example.com/api/wallet/credentials/list
```

and the response is a list of VCs that the user has stored in the web wallet.

```
Unset
{
  "list": [
    {
```

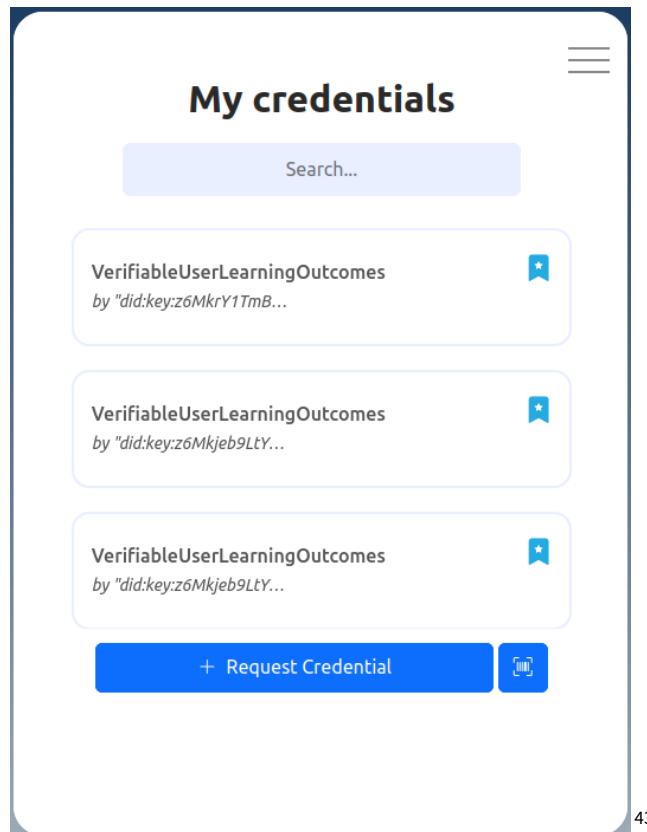
⁴² Walt.id ,Walt.id Web wallet - received credentials, walt.id. Available at: <https://github.com/walt-id/waltid-web-wallet> (last accessed: 08 July 2023)

```

    "type":
["VerifiableCredential", "VerifiableAttestation", "VerifiableUserLearningOutcom
es"],
    "@context": [ "https://www.w3.org/2018/credentials/v1" ],
    "id": "urn:uuid:35ab06f9-3461-4b99-a5f9-aac84792a4d5",
    "issuer": "did:key:z6MkrY1TmB9mWpTrHDTfaMyMt1qvPmQbHYqAKc8S6SHbmyc4",
    "issuanceDate": "2023-07-18T16:23:35Z",
    "issued": "2023-07-18T16:23:35Z",
    "validFrom": "2023-07-18T16:23:35Z",
    "credentialSchema": {
      "id": "https://proto.example.com/edu/api/v1/wallet/credentialSchemas/3",
      "type": "JsonSchemaValidator2018"
    },
    "credentialSubject": {
      "id": "did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p",
      "title": "User learning outcomes",
      "performed": [
        {
          "title": "Test1",
          "startedAtTime": "2023-07-12T17:28:16+02:00",
          "specifiedBy": {
            "teaches": {
              "learningOutcome": {
                "name": "chemische Laboruntersuchungen an Metallen vornehmen",
                "relatedESCOskill":
"http://data.europa.eu/esco/skill/2b60c0cf-6ce6-4f04-9748-0e6d883673d8"
              }
            }
          }
        }
      ]
    }
  }
}

```

The image below displays the VCs that an user has stored in her/his wallet:



Verification workflow & requests

The verification flow starts once a user clicks the **verification button** from the **verification UI**, then a **HTTP GET request** will be send to the **Demo Application API** in order to start the verification flow and then the response of this request that is a redirection URL will redirect the user to web **wallet UI** that the verifier platform has configure.

Once the user is redirected to web wallet UI, this one send an **HTTP GET** request to Wallet Kit API in order to start the verification process, the request is the next:

```
Unset
https://wallet.example.com/api/siop/initiatePresentation/?scope=openid&presentation_definition={"format" : null, "id" : "1", "input_descriptors" :
```

⁴³ Walt.id, *Walt.id Web wallet - verifiable credentials stored / owned by a user, walt.id*. Available at: <https://github.com/walt-id/waltid-web-wallet> (last accessed: 08 July 2023)

```
[{"constraints" : {"fields" : [{"filter" : {"const":  
"VerifiableUserLearningOutcomes"}, "id" : null, "path" : ["$.type"],  
"purpose" : null}]}, "format" : null, "group" : null, "id" : "1", "name" :  
null, "purpose" : null, "schema" : null}], "name" : null, "purpose" : null,  
"submission_requirements" :  
null}&response_type=vp_token&redirect_uri=https://proto.example.com/edu/api/v  
1/wallet/verifier/verify&state=1iP0k21YRPGp1CmPPwIILQ&nonce=1iP0k21YRPGp1CmPP  
wIILQ&client_id=https://proto.example.com/edu/api/v1/wallet/verifier/verify&r  
esponse_mode=form_post
```

The response of this request is a redirection to the “CredentialRequest” page of the web wallet UI with a parameter called “sessionId”, the response is similar to this one:

```
Unset  
https://wallet.example.com/CredentialRequest/?sessionId=eb5fda12-5c26-4e11-89  
e3-fce0444e729f
```

Once this request is sent another one it is also sent in order to retrieve the list of DIDs:

```
Unset  
https://wallet.example.com/api/wallet/did/list
```

and the response is a list of DIDs:

```
Unset  
did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p
```

Then another **HTTP GET** request is sent in order to continue with the VP of the VCs and obtain a redirection URL in order to **verify** the VCs that the VP contains:

```
Unset
```

```
https://wallet.example.com/api/wallet/presentation/continue?sessionId=eb5fda12-5c26-4e11-89e3-fce0444e729f&did=did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p
```

And the response of this request is the next:

```
Unset
{
  "availableIssuers": null,
  "did": "did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p",
  "id": "eb5fda12-5c26-4e11-89e3-fce0444e729f",
  "presentableCredentials": [
    {
      "claimId": "1",
      "credentialId": "urn:uuid:35ab06f9-3461-4b99-a5f9-aac84792a4d5"
    }
  ],
  "presentationDefinition": {
    "format": null,
    "id": "1",
    "input_descriptors": [{}],
    "name": null,
    "purpose": null,
    "submission_requirements": null
  },
  "redirectUri":
  "https://proto.example.com/edu/api/v1/wallet/verifier/verify"
}
```

Then the web wallet UI send another HTTP GET request in order to retrieve all VC of the **“userLearningOutcomes”** type through the next request:

```
Unset
https://wallet.example.com/api/wallet/credentials/list?id=urn:uuid:35ab06f9-3461-4b99-a5f9-aac84792a4d5
```


The response to this request is a list of VC of the “userLearningOutcomes” type

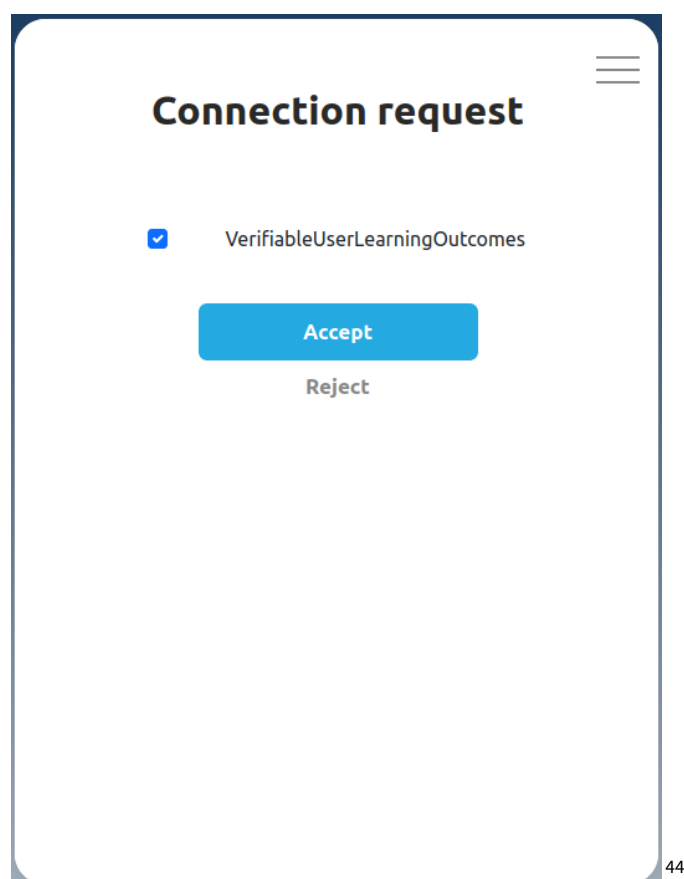
```
Unset
{
  "list": [
    {
      "type":
["VerifiableCredential", "VerifiableAttestation", "VerifiableUserLearningOutcom
es"],
      "@context": [ "https://www.w3.org/2018/credentials/v1" ],
      "id": "urn:uuid:35ab06f9-3461-4b99-a5f9-aac84792a4d5",
      "issuer": "did:key:z6MkrY1TmB9mWpTrHDTfaMyMt1qvPmQbHYqAKc8S6SHbmyc4",
      "issuanceDate": "2023-07-18T16:23:35Z",
      "issued": "2023-07-18T16:23:35Z",
      "validFrom": "2023-07-18T16:23:35Z",
      "credentialSchema": {
        "id": "https://proto.example.com/edu/api/v1/wallet/credentialSchemas/3",
        "type": "JsonSchemaValidator2018"
      },
      "credentialSubject": {
        "id": "did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p",
        "title": "User learning outcomes",
        "performed": [
          {
            "title": "Test1",
            "startedAtTime": "2023-07-12T17:28:16+02:00",
            "specifiedBy": {
              "teaches": {
                "learningOutcome": {
                  "name": "chemische Laboruntersuchungen an Metallen vornehmen",
                  "relatedESCOskill":
"http://data.europa.eu/esco/skill/2b60c0cf-6ce6-4f04-9748-0e6d883673d8"
                }
              }
            }
          }
        ]
      }
    }
  ]
}
```

```

    }
  }
]
}

```

These request represent the connection request between a verifier and the user's wallet, the image below display the UI of the user's web wallet:



Once the user accepts the verifiable presentation then a HTTP POST request is sent to the Wallet Kit API in order to present the selected “userLearningOutcomes” VC through the next endpoint:

⁴⁴ Walt.id, *Walt.id Web wallet -verified presentation from web wallet, walt.id*. Available at: <https://github.com/walt-id/waltid-web-wallet> (last accessed: 08 July 2023)

Unset

```
https://wallet.example.eu/api/wallet/presentation/fulfill?sessionId=24c1f6b0-1b6b-460e-820a-4f11fbfc0012
```

with the next payload:

Unset

```
[
  {
    "claimId": "1",
    "credentialId": "urn:uuid:35ab06f9-3461-4b99-a5f9-aac84792a4d5"
  }
]
```

and the next response:

Unset

```
{
  "fulfilled": false,
  "id_token": null,
  "presentation_submission": "{ \"definition_id\" : \"1\", \"descriptor_map\" : [ { \"format\" : \"jwt_vp\", \"id\" : \"0\", \"path\" : \"$\", \"path_nested\" : { \"format\" : \"jwt_vc\", \"id\" : \"0\", \"path\" : \"$.verifiableCredential[0]\" } } ], \"id\" : \"1\" }",
  "rp_response": null,
  "state": "f1tEm1anQ1G_bfm8WtRmMA",
  "vp_token": "JWT"
}
```

At last the user's web wallet redirects to the verifier platform in order to complete the verification flow.

How to run

In order to run the **demo web wallet UI** in production the **Wallet Kit API** with the other system components like the **Demo application API** and the **Demo application UI** must be deployed. Once the whole application stack is deployed then from the **issuer** or **verifier portal UIs** a user can reach the web wallet, also with the configured domain from which

any user can access through an URL. The web wallet UI is builded within the Wallet Kit API so in order to run it, the Wallet Kit API must be running.

It is also possible to run the web wallet UI in a standalone or development mode, once the repository of the web wallet UI it is downloaded then from a command line interface it can be run over a Node.js server with the version of 16.19.1 and using the "**yarn install**" command in order to install all the dependencies of the web wallet UI and then using the "**yarn dev**" in order to run the **UI** locally.

11 Development Environment Setup

In order to carry out the development of this project, it is necessary to establish the development environment adequately, following some steps to follow in order to use the entire stack that makes up the project.

11.1 System Requirements

This is a list of the system requirements in order to be able to develop and start the application stack:

- A text editor or an **IDE** like “PhpStorm” to be able to open and edit the project's code.
- **GIT** as a version control system in order to track the changes in the code.
- **Docker** in order to execute a container with the project implementation
- [Docker-compose](https://docs.docker.com/compose/)⁴⁵ in order to create and execute a multi-container application.
- A **Node.js** environment, version **16.19.1** in order to execute the Demo-UI in development mode.

⁴⁵ Docker (2023b) *Overview of docker compose, Docker Documentation*. Available at: <https://docs.docker.com/compose/> (last accessed: 12 July 2023).

- Optionally, [Node Version Manager \(NVM\)](#)⁴⁶ in order to manage the versions of Node.js installed or used.
- Optionally, an **API development platform** with a Rest client like "Insomnia" or "Postman" to design, debug, and send requests to the Demo API without a frontend.
- **Yarn** package manager in order to set up and run the project.
- **Kubectl** in order to manage docker clusters and deploy the application
- **Helm** in order to manage the packages for Kubernetes
- A **web browser** like "Mozilla Firefox" or "Google Chrome" in order to execute the application and use the developer tools for the development process.

More information can be found inside the repositories.

11.2 Dependency Installation

The following dependencies will be required in order to use the implementation:

- [Vue.js Framework](#): The main frontend framework used for the development of the **Demo-UI** frontend as a single-page application.
- [Cake PHP 4 Framework](#): The main backend framework used for the development of the **Demo-API** or Open API.
- [Vuetify 2 UI Framework](#)⁴⁷: The main UI framework used for the **development of UI elements** within the Vue.js framework.
- [Pug](#)⁴⁸: A template engine focused on making **HTML** coding faster.
- [Axios](#)⁴⁹: A promise-based **HTTP Client** for node.js.

⁴⁶ Node(no data),*Node version manager*. Available at: <https://github.com/nvm-sh/nvm> (last accessed: 12 July 2023).

⁴⁷ Vuetify *A material design framework for vue.js*, Vuetify. Available at: <https://v2.vuetifyjs.com/en/> (last accessed: 02 July 2023).

⁴⁸ Pug *Getting started*, Pug. Available at: <https://pugjs.org/api/getting-started.html> (last accessed: 12 July 2023).

⁴⁹ Axios *Axios, Starting | Axios Docs*. Available at: <https://axios-http.com/en/docs/intro> (last accessed: 12 July 2023).

- [Qrious](#)⁵⁰: A library for **QR code generation** using canvas.
- [Vue-i18n](#)⁵¹: A plugin that allows the internationalization of a platform using multi-languages.
- [Vue router](#)⁵²: A plugin that allows **routing** between different pages over the Vue.js framework.
- [Vuex](#)⁵³: A **state management** pattern + library for Vue.js.
- [Sass](#)⁵⁴: A **CSS preprocessor** compatible with all its versions.
- [Eslint](#)⁵⁵: A static **code analysis tool** for identifying problematic patterns found in JavaScript code.
- [Typescript](#)⁵⁶: A strongly typed **programming language** that builds on JavaScript.

More information can be found inside the repositories.

11.3 System Configuration

The configurations necessary for the implementation of the project are defined in each `README.md` file of the **used repositories**. For more configurations related to the deployment and resources, there exists a Helm chart file `"issuer-verifier-api.yaml"` in order to manage the whole stack of applications provided once they are deployed.

⁵⁰ Qrious *Qrious*, *npm*. Available at: <https://www.npmjs.com/package/qrious> (last accessed: 10 July 2023).

⁵¹ Vue *Vue I18n*. Available at: <https://kazupon.github.io/vue-i18n/> (last accessed: 12 July 2023).

⁵² Vue *Vue Router*, *Vue Router | The official Router for Vue.js*. Available at: <https://router.vuejs.org/> (last accessed: 10 July 2023).

⁵³ Vue *What is Vuex?*, *Vuex*. Available at: <https://vuex.vuejs.org/> (last accessed: 12 July 2023).

⁵⁴ sass *CSS with superpowers*, *Sass*. Available at: <https://sass-lang.com/> (last accessed: 10 July 2023).

⁵⁵ Nicholas C. Zakas 11 Aug et al. (1970) *Find and fix problems in your JavaScript code - eslint - pluggable JavaScript linter*, *ESLint*. Available at: <https://eslint.org/> (last accessed: 10 July 2023).

⁵⁶ Typescript *JavaScript with syntax for types.*, *TypeScript*. Available at: <https://www.typescriptlang.org/> (last accessed: 14 July 2023).

11.4 Repositories needed

To run the project it is necessary to download certain repositories into which the application is divided:

- [Demo application](#): This is the main repository of the project implementation in which there is a [Kubernetes configuration file](#): This file is intended to provide the **configuration and setup** of the whole Demo application in order to **deploy and run** the whole stack of applications under the same context. This file will create the architecture necessary for the implementation of the whole stack and the configuration of the containers that Kubernetes handles. In this file, it is defined the configuration files of the **Wallet Kit API** and the image to download and use to perform the SSI operations. Also, other configurations like the ports, mount points, or volumes are defined here.

```
Unset
apiVersion: v1
kind: ConfigMap
metadata:
  name: wallet-config
data:
  issuer-config.json: |
    {
      "issuerUiUrl": "https://proto.example.com/assess/en/issuance",
      "issuerApiUrl": "https://proto.example.com/issuer-api/default",
      "wallets": {
        "walt.id": {
          "id": "EduProto",
          "url": "https://wallet.example.com",
          "presentPath": "api/siop/initiatePresentation/",
          "receivePath": "api/siop/initiateIssuance/",
          "description": "EduProto walt.id based web wallet"
        }
      }
    }
```

```

    }
  }
  verifier-config.json: |
    {
      "verifierUiUrl": "https://proto.example.com/assess/en/verify",
      "verifierApiUrl":
"https://proto.example.com/edu/api/v1/wallet/verifier",
      "wallets": {
        "walt.id": {
          "id": "walt.id",
          "url": "https://wallet.walt.id",
          "presentPath": "api/siop/initiatePresentation/",
          "receivePath": "api/siop/initiateIssuance/",
          "description": "walt.id web wallet"
        },
        "EduProto": {
          "id": "EduProto",
          "url": "https://wallet.example.com",
          "presentPath": "api/siop/initiatePresentation/",
          "receivePath": "api/siop/initiateIssuance/",
          "description": "EduProto prototype wallet"
        }
      }
    }

# verifier-config.json "verifierApiUrl":
"https://proto.example.com/verifier-api/default",
https://proto.example.com/edu/api/v1/wallet/verifier
---

kind: Deployment
apiVersion: apps/v1
metadata:
  name: walletkit
spec:
  replicas: 1
  selector:
    matchLabels:

```



```
    app: walletkit
template:
  metadata:
  labels:
  app: walletkit
  annotations:
  deployment/id: "_DEFAULT_DEPLOYMENT_"
  spec:
  containers:
  - name: walletkit
    image: waltid/walletkit:1.2305151432.0
    volumeMounts:
    - name: wallet-config
      mountPath: "/app/dataRoot/config/"
      readOnly: true
    env:
    - name: WALTID_DATA_ROOT
      value: "/app/dataRoot"
    - name: WALTID_WALLET_BACKEND_BIND_ADDRESS
      value: 0.0.0.0
    args:
    - run
    ports:
    - containerPort: 8080
      name: http-api
    volumes:
    - name: wallet-config
      configMap:
        name: wallet-config
  ---
kind: Service
apiVersion: v1
metadata:
  name: walletkit-api-svc
spec:
  ports:
```

```
- name: http
port: 80
targetPort: http-api
protocol: TCP
selector:
  app: walletkit
```

Also there are some repositories that they are in charge of provide a frontend and a backend to the implementation:

- [Demo API](#): This is the repository on which a public API has been implemented, the **Open API**, which is in charge of handling the requests made from the implementation of the Demo UI, **it is acting as an intermediary API between the Demo-UI and the Wallet Kit API**. This Open API manages the requests in order to carry out the issuance flows and verifications of verifiable credentials. Inside this repository exist a `README.md` file which is a more detailed documentation.
- [Demo UI](#): This repository has implemented a **frontend** using the Vue.js and Vuetify UI framework in order to provide a graphical interface (UI) so that users can carry out the processes of claiming and submitting verifiable credentials. . Inside this repository exist a `README.md` file which is a more detailed documentation.

12 Open API documentation

Once the design, development, implementation, integration and start-up of the demo application API has been completed, the final points resulting from its implementation will be documented using the **Open API specification** by means of a file in JSON format that is

available on the [Demo Application repository](#) which will be available to download and imported on tools such as [Swagger Editor](#)⁵⁷ or [Postman](#)⁵⁸ to serve as data input and visualize the documentation graphically and intuitively.

13 Wallet kit API

The Wallet Kit API is an open source solution made by a **third party** specifically "[Walt.id](#)⁵⁹" with the purpose of providing a Self-Sovereign Identity (**SSI**) that enable decentralized identity and digital wallet **infrastructure** to the developers and businesses in order to implement the **SSI paradigm** of the **Web 3**.

The Wallet Kit provides the backend infrastructure to build a custom wallet solution (Demo web wallet) and provides the services of the [SSI Kit API](#)⁶⁰. This API is the **core** of the whole **SSI** functionalities that make it possible to create [DIDs](#)⁶¹, keys, VC and perform the **issuance** and **verification** of verifiable credentials.

The Wallet Kit API is built on top of the **SSI Kit** which is in charge of performing all the SSI-related tasks. The Wallet Kit just extends the SSI Kit In order to extend the functionality providing some components like the **demo web wallet**, an issuance portal UI, and a verification portal UI.

The Wallet Kit API enables to **use of different identity ecosystems** like Europe's emerging identity ecosystem ([ESSIF/EBSI](#)⁶²) in anticipation of a multi-ecosystem future.

⁵⁷ Swagger *Swaggereditor*, *SwaggerEditor*. Available at: <https://editor-next.swagger.io/> (last accessed: 08 July 2023).

⁵⁸ Postman *Postman*. Available at: <https://www.postman.com/> (last accessed: 13 July 2023).

⁵⁹ Walt.id *Identity and NFT infrastructure for developers.*, *walt.id*. Available at: <https://walt.id/> (last accessed: 08 July 2023).

⁶⁰ Walt.id *SSI Kit, Docs*. Available at: <https://docs.walt.id/v/ssikit/ssi-kit/readme> (last accessed: 13 July 2023).

⁶¹ Sporny, M. et al. *Decentralized identifiers (DIDs) v1.0, W3C*. Available at: <https://www.w3.org/TR/did-core/> (last accessed: 08 July 2023).

⁶² European Commission *European Blockchain Services Infrastructure, Home - EBSI* -. Available at: <https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home> (last accessed: 09 July 2023).

The Wallet Kit API and the documentation can be found on the GitHub repository from “walt.id” in [waltid-walletkit](https://github.com/walt-id/waltid-walletkit)⁶³.

Functionalities

The Wallet Kit provides various high-level functionalities. For example:

- **Web wallet app (Demo web wallet application)**
 - Web-based user interface (UI) for managing credentials and DIDs.
- **User context separation**
 - Separation of user contexts in the data stores (key store, credential store, DID store).
- **User data management**
 - DIDs
 - VCs
- **Ecosystems integrations**
 - did:bsi
 - DID creation
 - DID registration
 - did:web
 - DID creation
 - DID web registry
 - did:key
 - DID creation
- **Verifiable Credential Issuance and Presentation exchange**
 - Support for verifiable credential issuance and presentation exchange based on the [OIDC and SIOP specs](#)
 - ⁶⁴OIDC4CI - OIDC for credential issuance

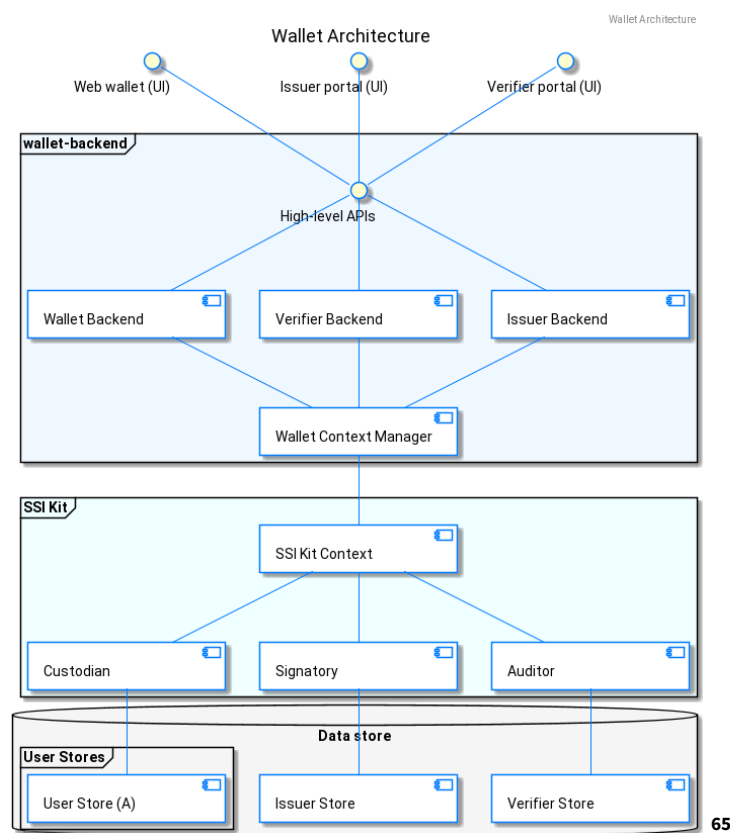
⁶³ GitHub - walt-id/waltid-walletkit - Walt.id. Available at: <https://github.com/walt-id/waltid-walletkit> (last accessed: 12 July 2023).

⁶⁴ Kristina Yasuda, Dr. Torsten Lodderstedt, *OpenID Connect for SSI*, Available at: https://openid.net/wordpress-content/uploads/2021/09/OIDF_OIDC4SSI-Update_Kristina-Yasuda-Torsten-Lodderstedt.pdf.

- OIDC4VP - OIDC for verifiable presentations (SIOP)
- Credential issuance via SIOP protocol (custom protocol)

Architecture

The Wallet Kit API architecture is displayed on the following image:

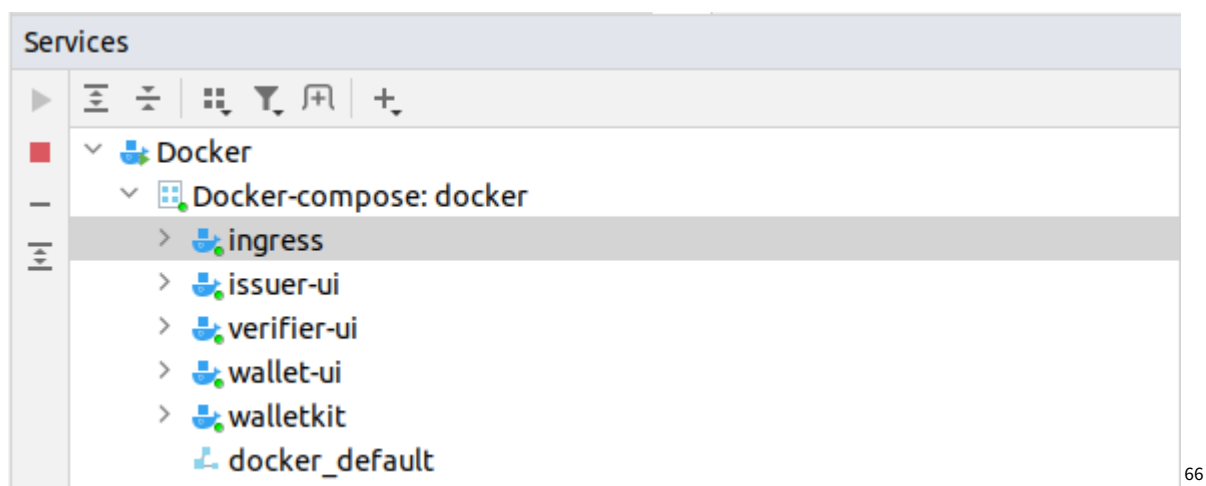


There exists a **layer** where the Wallet Kit API **stores all the data** related to the **issuer**, **verifier** and the **users**, here it stores information like the key pairs, DIDs, VCs and VPs. This layer is **part of the SSI Kit API**, which is composed of three different APIs, the **custodian API** (Demo web wallet), the **signatory API** (issuer platform) and the **auditor API** (verifier platform). Normally the Issuer and the verifier of the implementation are the same.

⁶⁵ Walt.id ,Walt.id Wallet Kit - Architecture, walt.id. Available at: <https://github.com/walt-id/waltid-walletkit> (last accessed: 08 July 2023).

Over these layers, it's built the **Wallet Kit** which is **connected to the SSI Kit** in order to provide the whole **SSI stack** and its own implementation of the three high-level APIs that compose it, the **wallet backend** (Demo web wallet), the **issuer backend** (issuer platform) and the **verifier backend** (verifier platform). These **high-level APIs** serve the different UI that the wallet kit implements, the **web wallet UI** (Demo web wallet), the **issuer portal UI** (not used in this implementation), and the **verifier portal UI** (not used in this implementation). All of these UI components are **pre-built** and ready to work, but in this **Demo Application** it is only used the "Web wallet UI (Demo web wallet)" in order to perform the issuance and verification flow using a web wallet. The pre-built issuance and verification UI were skipped in order to use the **Demo Application UI** to work as an issuer and verifier UI, within this frontend in some components are implemented the necessary request in order to perform the issuance and verification flow.

Once the repository of Wallet Kit API is downloaded under the directory "**docker**" we can find the "**config**" directory in which the configurations for the issuer, verifier and wallet are defined. Also in the "**data**" directory, we can find all the information related to the issuer, users, and verifiers stored like the DIDs and the key pairs. There is also a "**docker-compose.yaml**" file which defines all the images, ports, volumes, and containers that will be used in order to run and build the Wallet Kit API stack in a standalone or development mode. Once this file is running it will run the containers that are displayed below:



⁶⁶ Wallet Kit API running locally

Integration with the Demo Application

The integration of the **Wallet Kit API** with the other components that compose the implementation of the **Demo Application** is done by configuring the "issuer-verifier-api.yaml" file in the "kubernetes" directory found in the [Demo application repository](#) also within this configuration file is possible to manage the configuration of the issuer and verifier from the Wallet Kit API.

```
Unset
spec:
  containers:
    - name: walletkit
      image: waltid/walletkit:1.2305151432.0
      volumeMounts:
        - name: wallet-config
          mountPath: "/app/dataRoot/config/"
          readOnly: true
      env:
        - name: WALTID_DATA_ROOT
          value: "/app/dataRoot"
        - name: WALTID_WALLET_BACKEND_BIND_ADDRESS
          value: 0.0.0.0
      args:
        - run
      ports:
        - containerPort: 8080
          name: http-api
  volumes:
    - name: wallet-config
      configMap:
        name: wallet-config
```

How Wallet Kit verify a verifiable presentation⁶⁷

“The verifier in order to verify any verifiable presentation that a user present to it will go through the following steps to make sure the certificate is valid:

1. Before the validation of the content of the certificate can take place, the VC needs to be parsed from the support **JSON-LD** or the **JWT** format. Depending on the ecosystem used, there will also be a **validation of the schema** of the verifiable credential.
2. **Validate** that the **DID of the holder**, stated in the certificate, is the person presenting the VC.
3. Checking if all the **state** values are valid (expiration date and if the certificate is revoked or not in the case that the system is within any decentralized ecosystem that records the issuance of the VCs).
4. Checking the claims about the subject and if they match the requirements to give the person access to the service they are requesting to get access to.
5. Checking the **signatures of the issuer and the holder**, by getting the DID of the issuer from the registry (in the case that the system is within a decentralized ecosystem like EBSI) and the DID from the holder in their wallet and validating it **using the public keys** presented in the related **DID documents**.”

⁶⁷ Walt.id *Verifiable credentials (VCS), Docs*. Available at: <https://docs.walt.id/v/ssikit/ssi-kit/what-is-ssi/technologies-and-concepts/verifiable-credentials-vcs-and-verifiable-presentations-vps> (last accessed: 14 July 2023).

14 Wallet Kit configuration

Once we have integrated the **Wallet Kit API** into our system, the next step is to configure it to perform a correct integration so that it works correctly. For this, this API provides the following configuration files:

- **docker-compose.yaml:** In this file that is only available under the “docker” directory on the **Wallet Kit API repository** they can be found the configurations to run the Wallet Kit API in a standalone mode or development mode, this file set a bunch of directives in order to launch the **Wallet Kit API**, the containers of “**wallet-ui**”, “**verifier-ui**”, “**issuer-ui**” and the **ingress** which make use of a **Nginx** server in order to setup in which ports the previous containers are served. There is also defined which images are used.

```
Unset
version: "3.3"
services:
  walletkit:
    image: waltid/walletkit:latest # backend docker image
    command:
      - run
    environment:
      WALTID_DATA_ROOT: ./data-root
      WALTID_WALLET_BACKEND_BIND_ADDRESS: 0.0.0.0
      EXTERNAL_HOSTNAME: localhost
      WALTID_WALLET_BACKEND_PORT: 8080
    volumes:
      - ./app/data-root # data store volume incl. config files.
    extra_hosts:
      - "localhost:host-gateway"
  wallet-ui:
    image: ssi-wallet-ui:alex7 # wallet web ui docker image
  verifier-ui:
```

```
    image: waltid/ssikit-verifier-portal:latest # verifier web ui docker image
issuer-ui:
    image: waltid/ssikit-issuer-portal:latest # issuer web ui docker image
ingress:
    image: nginx:1.15.10-alpine
    ports:
      - target: 80
        published: 8080 # wallet ui publish port
        protocol: tcp
        mode: host
      - target: 81
        published: 8081 # verifier ui publish port
        protocol: tcp
        mode: host
      - target: 82
        published: 8082 # issuer ui publish port
        protocol: tcp
        mode: host
    volumes:
      - ./ingress.conf:/etc/nginx/conf.d/default.conf # API gateway
configuration
```

- **issuer-config.json:** In this file, it is possible to change to different configurations of the issuer platform like the UI URL, API URL, the default DID of the issuer, and the web wallets that the issuer platform will support, within each wallet it is possible to set the paths for the issuance and verification flow. The configuration of this file, can be found under the “`docker`” directory on the Wallet Kit API repository, once the Wallet Kit API is integrated on the application stack this files can be found on under the directory “`kubernetes`” within the “`issuer-verifier-api.yaml`” file of the [Demo Application repository](#).

Unset

```
issuer-config.json: |
{
  "issuerUiUrl": "https://proto.example.com/assess/en/issuance",
  "issuerApiUrl": "https://proto.example.com/issuer-api/default",
  "wallets": {
    "walt.id": {
      "id": "EduProto",
      "url": "https://wallet.example.com",
      "presentPath": "api/siop/initiatePresentation/",
      "receivePath": "api/siop/initiateIssuance/",
      "description": "EduProto walt.id based web wallet"
    }
  }
}
```

- **wallet-config.json:** In this file, it is possible to change to different configurations of the web wallet like the UI URL, API URL and the issuers' platforms that it will support, within each issuer, it is possible to set the path for the URL of the issuer. The configuration of this file can be found under the “docker” directory on the Wallet Kit API repository, once the Wallet Kit API is integrated into the application stack this file can be found under the directory “kubernetes” within the “issuer-verifier-api.yaml” file of the [Demo Application repository](#).

Unset

```
{
  "walletUiUrl": "http://wallet.example.com",
  "walletApiUrl": "http://wallet.example.com/api",
  "issuers": {
    "walt.id": {
      "id": "EduProto",
      "url": "http://proto.example.com/issuer-api/default/oidc",

```

```
    "description": "EduProtowalt.id Issuer Portal"
  }
}
```

- **verifier-config.json:** In this file, it is possible to change to different configurations of the verifier platform like the UI URL, API URL and the web wallets that the verifier platform will support, within each wallet it is possible to set the paths for the issuance and verification flow. The configuration of this file, can be found under the “docker” directory on the Wallet Kit API repository, once the Wallet Kit API is integrated on the application stack this files can be found on under the directory “kubernetes” within the “issuer-verifier-api.yaml” file of the [Demo Application repository](#).

Unset

```
verifier-config.json: |
{
  "verifierUiUrl": "https://proto.example.com/assess/en/verify",
  "verifierApiUrl": "https://proto.example.com/edu/api/v1/wallet/verifier",
  "wallets": {
    "walt.id": {
      "id": "walt.id",
      "url": "https://wallet.walt.id",
      "presentPath": "api/siop/initiatePresentation/",
      "receivePath": "api/siop/initiateIssuance/",
      "description": "walt.id web wallet"
    },
    "EduProto": {
      "id": "EduProto",
      "url": "https://wallet.example.com",
```

```
"presentPath": "api/siop/initiatePresentation/",  
"receivePath": "api/siop/initiateIssuance/",  
"description": "EduProto prototype wallet"  
}  
}  
}
```

- **ingress.conf:** This file is intended to perform the configuration of the routes to the containers that compose the **Wallet Kit API**, this file can be found under the “docker” directory on the Wallet Kit API repository, this not directly available once it is integrated on the application stack.

Unset

```
server {  
    listen 80;  
    location ~* /(api|webjars|verifier-api|issuer-api)/ {  
        proxy_pass http://walletkit:8080;  
        proxy_redirect default;  
    }  
    location / {  
        proxy_pass http://wallet-ui:80/;  
        proxy_redirect default;  
    }  
}  
  
server {  
    listen 81;  
    location ~* /(api|webjars|verifier-api|issuer-api)/ {  
        proxy_pass http://walletkit:8080;  
        proxy_redirect default;  
    }  
    location / {
```

```
    proxy_pass http://verifier-ui:80/;
    proxy_redirect default;
}

server {
    listen 82;
    location ~* /(api|webjars|verifier-api|issuer-api)/ {
        proxy_pass http://walletkit:8080;
        proxy_redirect default;
    }
    location / {
        proxy_pass http://issuer-ui:80/;
        proxy_redirect default;
    }
}
```

More info related to the Wallet Kit API can be found on the [GitHub repository](#).

15 DID and keys creation

The creation of **DIDs and keys** is done within the Wallet Kit API, which endpoints are private to the rest of the network, so that only those that are only open through the Open API can be accessed. In this implementation, no endpoint related to the management and creation of keys or DIDs has been developed, so the use of these is limited to the configuration scope of the Wallet Kit API or its previous creation also through the Wallet Kit API directly.

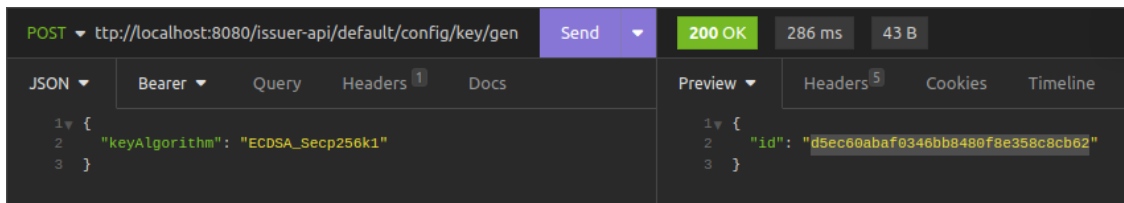
By default the Wallet Kit API already has a preconfigured DID and its keys with which the verifiable credentials will be issued also it is possible once a **DID is generated** to the issuer

to set it up in the configuration file `issuer-config.json` adding the field with the next value `"issuerDid": "did:key:example"` the image below shows the file with the configuration.

```
{
  "issuerUiUrl": "http://$EXTERNAL_HOSTNAME:8082",
  "issuerApiUrl": "http://$EXTERNAL_HOSTNAME:8082/issuer-api/default",
  "issuerDid": "did:key:example",
  "wallets": {
    "walt.id": {
      "id": "walt.id",
      "url": "http://$EXTERNAL_HOSTNAME:8080",
      "presentPath": "api/siop/initiatePresentation/",
      "receivePath": "api/siop/initiateIssuance/",
      "description": "walt.id web wallet"
    }
  }
}
```

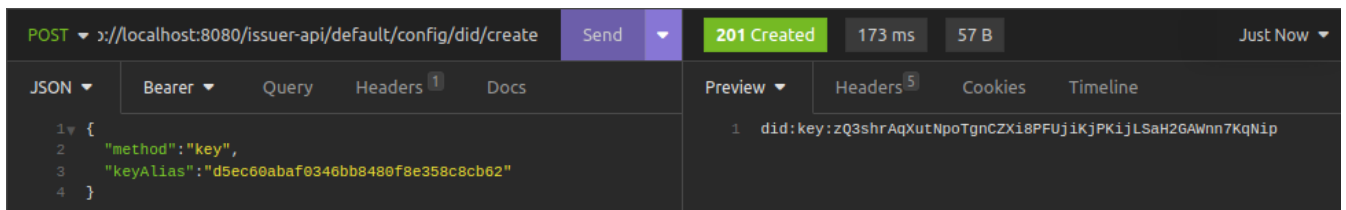
68

In this implementation the **DID method used** is the **"key"** method because the project is **outside of any blockchain ecosystem yet**. In order to use this DID method, first a key pair is generated through the **Wallet Kit API**.



69

Then to create the DID it is necessary to provide the method and a key alias (that is the response of the generated key pair), then it is created the **DID** with the **associated public key** through the **Wallet Kit API**.



70

⁶⁸ Issuer DID configuration file

⁶⁹ Key generation through Wallet Kit API

⁷⁰ DID generation with associated key through Wallet Kit API

This DID method is composed of the corresponding **public key**, which makes it especially useful in scenarios where a **simpler and more direct resolution** is needed because a verifier will only need the public key of the owner of the DID to **verify** the authenticity of the verifiable credential and this public key is **encoded** in the DID normally in a **base58 format**.

16 Creation of VC schema

To issue a personalized verifiable credential, it is first necessary to create a **data model** or schema. For the use case of this demo, a verifiable credential of the **"userLearningOutcomes"** type has been implemented, which conforms to a data schema in **JSON-LD** format.

This scheme **describes the structure, fields and restrictions** that the verifiable credential must follow once it is created. This schema can be published in a public repository so that once a user with a verifiable credential of this type presents it to a third party or **verifier**, the verifier will be in charge of extracting the content of the **"id"** field within the field **"credentialSchema"** from the body of the verifiable credential, this is a **URI** that references the schema of the verifiable credential.

So that the verifier will perform a **check** on whether the **verifiable credential conforms to the scheme that defines it**, this is configured by the verifier in its **verification policies** by default the policy applied is **"JsonSchemaPolicy"** which check that the body of the VC matches the schema, which in this implementation no endpoint has been created over the **Demo Application API** that is in charge of modifying the settings of these policies, the default policies that are configured on the **Wallet Kit API** have been left.

The schema belonging to the verifiable credential created **"userLearningOutcomes"** has been generated within the implementation of the Demo application API, and it has the body of the code below:

```
Unset
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
```



```
"title": "User learning outcomes verifiable accreditation",
"description": "Schema of a user learning outcomes verifiable accreditation",
"type": "object",
"allOf": [{
  "$ref":
  "https://api-pilot.ebsi.eu/trusted-schemas-registry/v2/schemas/0xeb6d81312643
  27f3cbc5ddba9c69cb9afd34732b3b787e4b3e3507a25d3079e9"
},
{
  "properties": {
    "credentialSubject": {
      "description": "Defines additional properties on
credentialSubject to describe the body of the verifiable credential",
      "type": "object",
      "properties": {
        "id": {
          "description": "Defines the did of
the credential subject",
          "type": "string"
        },
        "title": {
          "description": "Title of the
credential subject",
          "type": "string"
        },
        "performed": {
          "description": "Defines the learning
activity that a person participated in or attended",
          "type": "array",
          "items": {
            "$ref": "#/$defs/performed"
          }
        }
      },
      "required": ["id", "title", "performed"]
    }
  }
}]
```

```

    }
  }
}],
"$defs": {
  "performed": {
    "description": "Defines the learning activity that a
person participated in or attended",
    "type": "object",
    "properties": {
      "title": {
        "description": "Defines a title of the
learning achievement",
        "type": "string"
      },
      "startedAtTime": {
        "description": "The date the learner
started the activity",
        "type": "DateTime"
      },
      "endedAtTime": {
        "description": "The date the learner ended
the activity",
        "type": "DateTime"
      },
      "specifiedBy": {
        "definition": "The specification of this
learning activity",
        "type": "object",
        "properties": {
          "teaches": {
            "definition": "The expected
learning outcomes this learning activity specification can lead or contribute
to",
            "type": "array",
            "items": {

```

```

"$ref":
"#/$defs/teaches"
}
}
}
}
},
"required": ["title"]
},
"teaches": {
  "definition": "The expected learning outcomes this
learning activity specification can lead or contribute to",
  "type": "object",
  "properties": {
    "learningOutcome": {
      "description": "The learning outcome of the
learning specification",
      "type": "object",
      "properties": {
        "name": {
          "description": "A legible,
descriptive name for the learning outcome",
          "type": "string"
        },
        "relatedESCOSkill": {
          "description": "A URI to the
related ESCO Skill",
          "type": "object",
          "items": {
            "description": "A URI
to the related ESCO Skills",
            "type": "string",
            "format": "uri"
          }
        }
      }
    }
  }
}

```

```
        },
        "required": ["name", "relatedESCOskills"]
    },
    },
    "required": ["learningOutcome"]
}
}
```

17 Creation of the verifiable credential

During the development of this project, the creation of an **own** verifiable credential called **"userLearningOutcomes"** has been carried out in relation to the **use case of the issuance of educational credentials**. This verifiable credential has been created with the purpose of demonstrating the **knowledge** and **skills** that a user of an **educational platform** has achieved through the course of different educational courses that they have carried out. This verifiable credential was designed following the schema of the [Verifiable Diploma Schema](#)⁷¹ that is defined for EBSI use cases.

This verifiable credential is intended to document certain basic aspects related to the **educational courses** that the user has taken, also this verifiable credential was created with the purpose of making use of the [European Skills, Competences, Qualifications and Occupations \(ESCO\)](#)⁷² classification in order to classify the learnings outcomes of the user under a European standard framework.

⁷¹ European Commission *Verifiable diploma schema, Verifiable Diploma Specifications* -. Available at:

<https://ec.europa.eu/digital-building-blocks/wikis/display/EBSIDOC/Verifiable+Diploma+Schema> (last accessed: 09 July 2023).

Schema - EBSI

⁷² ESCO About Esco, ESCO. Edited by the European Commission. Available at: <https://esco.ec.europa.eu/en/about-esco> (last accessed: 08 July 2023).

The verifiable credential template is created through the **Demo application API** which will create the **schema** linked to it and the **verifiable credential** body if it doesn't exist yet within the **Wallet Kit API**.

The verifiable credential is made up of the following fields:

Field	Description
id	Contains the DID of the user receiving the verifiable credential.
title	Title of the course taken.
performed	Defines the learning activities that a person participated in or attended, within this list we have the different parts of which the course is made up.

Fields within "performed" field	Description
title	Contains the DID of the user receiving the verifiable credential.
startedAtTime	The date the learner started the activity
endAtTime	The date the learner ended the activity
specifiedBy	The specification of this learning activity, within this field we have more fields inside.

Fields within “specifiedBy” field	Description
teaches	The expected learning outcomes this learning activity specification can lead or contribute to, this list contains more fields inside.

Fields within “teaches” field	Description
learningOutcome	The learning outcome of the learning specification, which has more fields inside.

Fields within “learningOutcome” field	Description
name	A readable, descriptive name for the learning outcome
relatedESCOSkill	A URI to the related ESCO Skill

below it is displayed an example of the content of a verifiable credential of the type **"userLearningOutcomes"**

```
Unset
{
  "credentialSubject": {
    "id": "did:key:z6MkhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p",
    "title": "User learning outcomes",
    "performed": [{
```

```

        "title": "Test1",
        "startedAtTime": "2023-08-09T14:39:17+02:00",
        "specifiedBy": {
            "teaches": [{
                "learningOutcome": {
                    "name": "Web-Designer/Web-Designerin",
                    "relatedESCOSkill":
"http://data.europa.eu/esco/occupation/c40a2919-48a9-40ea-b506-1f34f693496d"
                }
            }]
        }
    }
}

```

18 Verifiable credentials issuance

<https://docs.walt.id/v/web-wallet/concepts/oidc/oidc>

During the flow of verifiable credentials issuance, the following requests will be made in the following order:

- The user access to the issuance UI, the first request performed is related to retrieving the verifiable credentials types that the issuer can issue to the users.
 - **GET** <https://proto.example.com/edu/api/v1/wallet/users/{UID}/issuance>
 - **Response:**

Unset

```

{
  "data": [
    {

```

```

    "id": "1",
    "name": "DigComp2.1 (Demo Ausschnitt)",
    "type": "Europass",
    "description": "Europass DigComp2.1 (Demo Ausschnitt) from 08.08.23,
19:38"
  },
  {
    "id": "2",
    "name": "Great tutor",
    "type": "ProofOfResidence",
    "description": "Lorem ipsum dolor sit amet, consectetur adipiscing elit"
  },
  {
    "id": "3",
    "name": "User learning outcome",
    "type": "UserLearningOutcome",
    "description": "Verifiable credential of an user learning outcome"
  }
]
}

```

- The user from the issuer UI choose a VC in order to be issued and perform the next HTTP request in order to obtain a URL where the users will be redirected to a web wallet:
 - **POST** <https://proto.example.com/edu/api/v1/wallet/users/{UID}/issuance>
 - **Payload:**

Unset

```

{
  "vc_id": "3",
  "isPreAuthorized": true,

```



```
"xDevice":false
}
```

○ **Response:**

```
Unset
{
  "data":{
    "redirect_uri":"https://wallet.example.com/api/siop/initiateIssuance/?
issuer=https%3A%2F%2Fproto.example.com%2Fissuer-api%2Fdefault%2Foidc%2
F&credential_type=UserLearningOutcomes&pre-authorized_code=eyJ0eXAiOiJ
KV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI5YjQ2NWQwOC05OWUyLTRlMzYtYjYxMS00
MjJjMWMMyYjRmYjMiLCJwcmUtYXV0aG9yaXplZCI6dHJ1ZX0.qH08WS9BUQkgmWH777LHAA
PJeXvN-CElCaIo6JIIGMk&user_pin_required=false"
  }
}
```

This request once it reach the Demo Application API, will forward to the next endpoint of the Wallet Kit

“/issuer-api/default/credentials/issuance/request?walletId={YOUR_WALLET_ID}&isPreAuthorized=true&xDevice:false” in order to obtain a response and then forward it to the Demo Application API to manage it and send it back to the user.

- Once the user is redirected to a web wallet from the previous response of the the HTTP request, it obtains a URL from where the user will be redirected in order to start the issuance process and obtain a new URL with a sessionId as a param in the headers of the request where to be redirected, the next request is send:

- **GET**
https://wallet.example.com/api/siop/initiateIssuance/?issuer=https://proto.example.com/issuer-api/default/oidc/&credential_type=UserLearningOutcomes&pre-authorized_code=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI5YjQ2NWQwOC05OWUyLTRlMzYtYjYxMS00MjJjMWMMyYjRmYjMiLCJwcmUtYXV0aG9yaXplZCI6dHJ1ZX0.qH08WS9BUQkgmWH777LHAA

*CJwcmUtYXV0aG9yaXplZCI6dHJ1ZX0.qH08WS9BUQkgmWH777LHAAPJeXv
N-CELCalo6JIIGMk&user_pin_required=false*

- **Response:**

Unset

```
https://wallet.example.com/InitiateIssuance/?sessionId=fcbace6d-e7d3-4  
1a9-ad8c-ac7aee163f51
```

- Then from the user's web wallet a new request is sent to the Wallet Kit in order to retrieve a list of DIDs.

- **GET** <https://wallet.example.com/api/wallet/did/list>

- **Response:**

Unset

```
[ "did:key:z6MkhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p" ]
```

- Also from the user's web wallet it is sent a request in order to get the information of the issuance initiation.

- **GET** <https://wallet.example.com/api/wallet/issuance/info?sessionId=fcbace6d-e7d3-41a9-ad8c-ac7aee163f51>

- **Response:**

Unset

```
{  
  "credentialTypes" : [ "UserLearningOutcomes" ],  
  "credentials" : null,  
  "did" : null,  
  "id" : "fcbace6d-e7d3-41a9-ad8c-ac7aee163f51",  
  "isIssuerInitiated" : true,  
  "isPreAuthorized" : true,
```

```

    "issuerId" :
    "https://proto.example.com/issuer-api/default/oidc/",
    "lastTokenUpdate" : null,
    "nonce" : "65fed870-e447-4abb-96b9-7a4daa1384be",
    "opState" : null,
    "preAuthzCode" :
    "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI5YjQ2NWQwOC05O
    WUyLTRlMzYtYjYxMS00MjJjMWMYyRmYjMiLCJwcmUtYXV0aG9yaXplZCI6dHJ1
    ZX0.qH08WS9BUQkgmWH777LHAAPJeXvN-CElCaIo6JIIGMk", "tokenNonce" :
    null, "tokens" : null, "user" : null,
    "userPinRequired" : false,
    "walletRedirectUri" : null
  }

```

- Then once the user accepts to receive the VC that was issued a new request it is sent to the Wallet Kit in order to obtain a page of the web wallet and a sessionId to perform the VC exchange between the issuer and user's wallet.

- **GET**
<https://wallet.example.com/api/wallet/issuance/continuelIssuerInitiatedIssuance?did=did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p&sessionId=b8f052c8-db29-410e-9937-1d744e4ea980>

- **Response:**

Unset

```
/ReceiveCredential/?sessionId=b8f052c8-db29-410e-9937-1d744e4ea980
```

- Then once again the web wallet UI sends a request in order to obtain the issuance information about the VC that has been issued and stored on the user's web wallet.

- GET

<https://wallet.example.eu/api/wallet/issuance/info?sessionId=b8f052c8-db29-410e-9937-1d744e4ea980>

- **Response:**

```
Unset
{
  "credentialTypes" : [ "UserLearningOutcomes" ],
  "credentials" : null,
  "did" : null,
  "id" : "8b450ebe-eb6d-4b16-8bb7-768cc37c7519",
  "isIssuerInitiated" : true,
  "isPreAuthorized" : true,
  "issuerId" : "https://proto.example.com/issuer-api/default/oidc/",
  "lastTokenUpdate" : null, "nonce" : "81d50896-43b3-4280-9f06-37485b604461",
  "opState" : null,
  "preAuthzCode" :
  "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI5ODhlMjJlZC04MTc4LTQ1YmUtYT1lNi0xM2ZjY2NmNWJhNDUiLCJwcmUtYXV0aG9yaXplZCI6dHJ1ZX0.eyJwaTegQ4JoesfyigDNqNoy5AgIKkrAqSIn1qVuqhUA",
  "tokenNonce" : null,
  "tokens" : null,
  "user" : null,
  "userPinRequired" : false,
  "walletRedirectUri" : null
}
```

- At last once the VC is stored on the user's wallet a new request is sent in order to fetch the VC that the user own or has stored on the web wallet
 - GET <https://wallet.example.com/api/wallet/credentials/list>
 - **Response:**

```
Unset
{
  "list": [
    {
      "type":
["VerifiableCredential", "VerifiableAttestation", "VerifiableUserLearningOutcom
es"],
      "@context": [ "https://www.w3.org/2018/credentials/v1" ],
      "id": "urn:uuid:35ab06f9-3461-4b99-a5f9-aac84792a4d5",
      "issuer": "did:key:z6MkrY1TmB9mWpTrHDTfaMyMt1qvPmQbHYqAKc8S6SHbmyc4",
      "issuanceDate": "2023-07-18T16:23:35Z",
      "issued": "2023-07-18T16:23:35Z",
      "validFrom": "2023-07-18T16:23:35Z",
      "credentialSchema": {
        "id": "https://proto.example.com/edu/api/v1/wallet/credentialSchemas/3",
        "type": "JsonSchemaValidator2018"
      },
      "credentialSubject": {
        "id": "did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p",
        "title": "User learning outcomes",
        "performed": [
          {
            "title": "Test1",
            "startedAtTime": "2023-07-12T17:28:16+02:00",
            "specifiedBy": {
              "teaches": {
                "learningOutcome": {
                  "name": "chemische Laboruntersuchungen an Metallen vornehmen",
                  "relatedESCOskill":
"http://data.europa.eu/esco/skill/2b60c0cf-6ce6-4f04-9748-0e6d883673d8"
                }
              }
            }
          }
        ]
      }
    }
  ]
}
```

```
]
}
```

19 Verifiable presentation

During the flow of verifiable presentation, the following requests will be made in the following order:

- The user access to the verifier UI and open her/his user profile in order to list or load the data of the profile that was extracted from the “**userLearningOutcomes**” verifiable credential, if it is the first time that the user open the modal of the profile then she/he can click in the left button from the bottom of the modal, that is called “**Import from wallet**”, this button will make a request in order to start the creation of a verifiable presentation of a “**userLearningOutcomes**” VC.
 - **GET**
https://proto.example.com/edu/api/v1/wallet/verifier/present?walletId=YOUR_WALLET_ID&vcType=VerifiableUserLearningOutcomes
 - **Response:**

Unset

```
https://wallet.example.com/api/siop/initiatePresentation/?scope=openid
&presentation_definition={"format": null, "id": "1", "input_descriptors"
: [{"constraints": {"fields": [{"filter": {"const":
"VerifiableUserLearningOutcomes"}, "id": null, "path": [ "$.type" ],
"purpose": null}]}}, {"format": null, "group": null, "id": "1", "name":
null, "purpose": null, "schema": null}], "name": null, "purpose": null,
"submission_requirements":
null}&response_type=vp_token&redirect_uri=https://proto.example.com/edu
/api/v1/wallet/verifier/verify&state=KhUFKVf7RgSTTYeLG9UDeQ&nonce=KhUF
```

```
KVf7RgSTTYeLG9UDeQ&client_id=https://proto.example.com/edu/api/v1/wallet/verifier/verify&response_mode=form_post
```

Once the Demo Application API handles the request for this endpoint, then this API will create a new HTTP request in order to forward it to the endpoint of the **“/verifier-api/default/present?walletId={YOUR_WALLET_ID}&vcType=VerifiableUserLearningOutcomes”** Wallet Kit and the response of it, it will forward to the Demo Application API in order to send it back to the user.

- With the response of the previous request then the user is redirected to the user’s web wallet, which executes an HTTP request in order to obtain a page of the configured web wallet and a sessionId and the response of it is a redirection URL in the headers of the request.

- **GET**

```
https://wallet.example.com/api/siop/initiatePresentation/?scope=openid&presentation_definition={"format": null, "id": "1", "input_descriptors": [{"constraints": {"fields": [{"filter": {"const": "VerifiableUserLearningOutcomes"}}, {"id": null, "path": [ "$.type" ], "purpose": null } ]}], "format": null, "group": null, "id": "1", "name": null, "purpose": null, "schema": null }], "name": null, "purpose": null, "submission_requirements": null}&response_type=vp_token&redirect_uri=https://proto.example.com/edu/api/v1/wallet/verifier/verify&state=KhUFKVf7RgSTTYeLG9UDeQ&nonce=KhUFKVf7RgSTTYeLG9UDeQ&client_id=https://proto.example.com/edu/api/v1/wallet/verifier/verify&response_mode=form_post
```

- **Response:**

Unset

```
https://wallet.example.com/CredentialRequest/?sessionId=5b149306-33fa-4505-a065-249e4dc20aed
```

- Then from the user's web wallet a request to fetch a list of DIDs is submitted to the Wallet Kit.
 - **GET** <https://wallet.example.com/api/wallet/did/list>
 - **Response:**

Unset

```
[ "did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p" ]
```

- Also from the user's web wallet it is send another request to the Wallet Kit in order to continue with the VP of the VCs and obtain a redirection URL in order to **verify** the VCs that the VP contains
 - **GET**
<https://wallet.example.com/api/wallet/presentation/continue?sessionId=5b149306-33fa-4505-a065-249e4dc20aed&did=did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p>
 - **Response:**

Unset

```
{
  "availableIssuers": null,
  "did": "did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p",
  "id": "eb5fda12-5c26-4e11-89e3-fce0444e729f",
  "presentableCredentials": [
    {
      "claimId": "1",
      "credentialId": "urn:uuid:35ab06f9-3461-4b99-a5f9-aac84792a4d5"
    }
  ]
}
```



```

    }
  ],
  "presentationDefinition": {
    "format": null,
    "id": "1",
    "input_descriptors": [{}],
    "name": null,
    "purpose": null,
    "submission_requirements": null
  },
  "redirectUri":
  "https://proto.example.com/edu/api/v1/wallet/verifier/verify"
}

```

- Then from the user's web wallet it is also send a request in order to retrieve all VC of the **"userLearningOutcomes"** type through the next request

- GET

<https://wallet.example.eu/api/wallet/credentials/list?id=urn:uuid:35ab06f9-3461-4b99-a5f9-aac84792a4d5&id=urn:uuid:8ca4510f-9072-4627-8f5f-4c9a6e49b7c1>

- **Response:**

```

Unset
{
  "list": [
    {
      "type":
["VerifiableCredential", "VerifiableAttestation", "VerifiableUserLearningOutcomes"],
      "@context": ["https://www.w3.org/2018/credentials/v1"],
      "id": "urn:uuid:35ab06f9-3461-4b99-a5f9-aac84792a4d5",
      "issuer": "did:key:z6MkrY1TmB9mWpTrHDTfaMyMt1qvPmQbHYqAKc8S6SHbmyc4",

```

```

"issuanceDate": "2023-07-18T16:23:35Z",
"issued": "2023-07-18T16:23:35Z",
"validFrom": "2023-07-18T16:23:35Z",
"credentialSchema": {
  "id": "https://proto.example.com/edu/api/v1/wallet/credentialSchemas/3",
  "type": "JsonSchemaValidator2018"
},
"credentialSubject": {
  "id": "did:key:z6MkhhosB653LLTzR6k6gUbMi3K24czgyNUfvqa4rG5G6x1p",
  "title": "User learning outcomes",
  "performed": [
    {
      "title": "Test1",
      "startedAtTime": "2023-07-12T17:28:16+02:00",
      "specifiedBy": {
        "teaches": {
          "learningOutcome": {
            "name": "chemische Laboruntersuchungen an Metallen vornehmen",
            "relatedESCOskill":
"http://data.europa.eu/esco/skill/2b60c0cf-6ce6-4f04-9748-0e6d883673d8"
          }
        }
      }
    }
  ]
}

```

- Then when the user accepts to create the verifiable presentation with the VC within it is sent a request to the Wallet Kit in order to fulfill verifiable credentials presentation with the selected verifiable credentials.

- **POST**

<https://wallet.example.eu/api/wallet/presentation/fulfill?sessionId=70382f32-5f47-40b4-abef-05e50687c550>

- **Payload:**

Unset

```
[
  {
    "claimId": "1",
    "credentialId": "urn:uuid:35ab06f9-3461-4b99-a5f9-aac84792a4d5"
  }
]
```

- **Response:**

Unset

```
{
  "fulfilled": false,
  "id_token": null,
  "presentation_submission": "{ \"definition_id\" : \"1\", \"descriptor_map\" : [{ \"format\" : \"jwt_vp\", \"id\" : \"0\", \"path\" : \"$\", \"path_nested\" : { \"format\" : \"jwt_vc\", \"id\" : \"0\", \"path\" : \"$.verifiableCredential[0]\" } }], \"id\" : \"1\" }",
  "rp_response": null,
  "state": "f1tEm1anQ1G_bfm8WtRmMA",
  "vp_token": "JWT"
}
```

- Then from the user's web wallet it is send a form request with the data of the response of the previous request to the Demo Application API in order to verify the VP and obtain a redirection.

- **POST** <https://proto.example.eu/edu/api/v1/wallet/verifier/verify>

- **Payload:**

Unset

```
{
  "fulfilled": false,
  "id_token": null,
  "presentation_submission": "{\"definition_id\" : \"1\", \"descriptor_map\" :
[ {\"format\" : \"jwt_vp\", \"id\" : \"0\", \"path\" : \"$\", \"path_nested\" : {\"format\" :
\"jwt_vc\", \"id\" : \"0\", \"path\" : \"$.verifiableCredential[0]\"} } ], \"id\" : \"1\" }\",
  \"rp_response\": null,
  \"state\": \"f1tEmlanQ1G_bfm8WtRmMA\",
  \"vp_token\": \"JWT\"
}
```

- **Response:**

Unset

http://proto.example.com/assess/en/verify/success/?access_token=HgCfBBiLSSywg2T-mihLQA

Once the Demo Application API handles the previous request then, it will forward a request to the Wallet Kit endpoint “**/verifier-api/default/verify**” in order to verify the VP and obtain a redirection to the page of the verifier platform with an `access_token`, then the response of the Wallet Kit will be forwarded to the Demo Application API in order to send it back to the users.

- Finally once the user is redirected to the verification UI with a valid **access_token** then it is sent a request to a third party API in order to store the VC in the database of the third party API.
 - **POST** <http://proto.example.com/edu/api/v1/wallet/verifier/store>
 - **Payload:**

Unset

```
{\"wallet_token\": \"HgCfBBiLSSywg2T-mihLQA\"}
```

- **Response:**

Unset

```
{
  "data": {
    "vps": [
      {}
    ]
  }
}
```

20 Documental sources

Axios Axios, *Starting | Axios Docs*. Available at: <https://axios-http.com/en/docs/intro> (last accessed: 12 July 2023).

Cake PHP .. *CakePHP cookbook Archivo de Documentación, creado por, Bienvenido - 4.x*. Available at: <https://book.cakephp.org/4/es/index.html> (last accessed: 12 July 2023).

Cohen, G. and Steele, O. (2023) *Verifiable credentials JSON schema specification*, W3C. Available at: <https://www.w3.org/TR/vc-json-schema/> (last accessed: 09 July 2023)

Docker (2023) *Accelerated Container Application Development*, Docker. Available at: <https://www.docker.com/> (last accessed: 12 July 2023).

Docker (2023b) *Overview of docker compose*, Docker Documentation. Available at: <https://docs.docker.com/compose/> (last accessed: 12 July 2023).

ESCO About Esco, ESCO. Edited by the European Commission. Available at: <https://esco.ec.europa.eu/en/about-esco> (last accessed: 08 July 2023).

European Commission *European Blockchain Services Infrastructure, Home - EBSI* -. Available at: <https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home> (last accessed: 09 July 2023).

European Commission *Verifiable diploma schema, Verifiable Diploma Schema - EBSI Specifications* -. Available at:

<https://ec.europa.eu/digital-building-blocks/wikis/display/EBSIDOC/Verifiable+Diploma+Schema> (last accessed: 09 July 2023).

Helm *Helm*. Available at: <https://helm.sh/> (last accessed: 09 July 2023).

<https://www.slideshare.net/SSIMeetup/understanding-the-european-selfsovereign-identity-framework-essif>

Kristina Yasuda, Dr. Torsten Lodderstedt, *OpenID Connect for SSI*, Available at:

https://openid.net/wordpress-content/uploads/2021/09/OIDF_OIDC4SSI-Update_Kristina-Yasuda-Torsten-Lodderstedt.pdf.

Kubernetes *Production-grade container orchestration*, Kubernetes. Available at:

<https://kubernetes.io/> (last accessed: 09 July 2023).

Nginx (2023) *Advanced load balancer, web server, & reverse proxy*, NGINX. Available at:

<https://www.nginx.com/> (last accessed: 12 July 2023).

Nicholas C. Zakas 11 Aug et al. (1970) *Find and fix problems in your JavaScript code - eslint - pluggable JavaScript linter*, ESLint. Available at: <https://eslint.org/> (last accessed: 10 July 2023).

Node.js *Node.js*. Available at: <https://nodejs.org/en> (last accessed: 12 July 2023).

Node(no data), *Node version manager*. Available at: <https://github.com/nvm-sh/nvm> (last accessed: 12 July 2023).

Nuxt *The intuitive web framework*, Nuxt. Available at: <https://nuxt.com/> (last accessed: 12 July 2023).

Postman *Postman*. Available at: <https://www.postman.com/> (last accessed: 13 July 2023).

Pug *Getting started*, Pug. Available at: <https://pugjs.org/api/getting-started.html> (last accessed: 12 July 2023).

Qrious *Qrious*, npm. Available at: <https://www.npmjs.com/package/qrious> (last accessed: 10 July 2023).

sass *CSS with superpowers*, Sass. Available at: <https://sass-lang.com/> (last accessed: 10 July 2023).

Sporny , M., Longley , D. and Chadwick , D. *Verifiable credentials data model V1.1*, W3C. Available at: <https://www.w3.org/TR/vc-data-model/#abstract> (last accessed: 08 July 2023).

Sporny, M. *et al. Decentralized identifiers (DIDs) v1.0*, W3C. Available at: <https://www.w3.org/TR/did-core/> (last accessed: 08 July 2023).

Swagger *Swaggereditor*, *SwaggerEditor*. Available at: <https://editor-next.swagger.io/> (last accessed: 08 July 2023).

T. Lodderstedt, K. Yasuda, T. Looker (03/02/2023), *OpenID for Verifiable Credential Issuance*. Available at:

https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html#name-introduction.

Typescript *JavaScript with syntax for types.*, *TypeScript*. Available at: <https://www.typescriptlang.org/> (last accessed: 14 July 2023).

ValidatedId *Validated ID - electronic signature and digital identity providers*, *Validated ID - Electronic Signature and Digital Identity Providers*. Available at:

<https://www.validatedid.com/en> (last accessed: 08 July 2023).

ValidatedId *Validated ID - electronic signature and digital identity providers*, *Validated ID - Electronic Signature and Digital Identity Providers*. Available at:

<https://www.validatedid.com/en> (last accessed: 08 July 2023).

Vue.js *The progressive javascript framework*, *Vue.js - The Progressive JavaScript Framework* | *Vue.js*. Available at: <https://vuejs.org/> (last accessed: 12 July 2023).

Vue *Vue 118n*. Available at: <https://kazupon.github.io/vue-i18n/> (last accessed: 12 July 2023).

Vue *What is Vuex?*, *Vuex*. Available at: <https://vuex.vuejs.org/> (last accessed: 12 July 2023).

Vue *Vue Router*, *Vue Router* | *The official Router for Vue.js*. Available at:

<https://router.vuejs.org/> (last accessed: 10 July 2023).

Vuetify *A material design framework for vue.js*, *Vuetify*. Available at:

<https://v2.vuetifyjs.com/en/> (last accessed: 02 July 2023).

Walt.id ,*Walt.id Wallet Kit*, *walt.id*. Available at: <https://github.com/walt-id/waltid-walletkit>
(last accessed: 08 July 2023).

Walt.id ,*Walt.id Web wallet*, *walt.id*. Available at:

<https://github.com/walt-id/waltid-web-wallet> (last accessed: 08 July 2023).

Walt.id ,*Walt.id Web wallet*, *walt.id*. Available at:

<https://github.com/walt-id/waltid-web-wallet> (last accessed: 08 July 2023).

Walt.id *Architecture - Demo web wallet architecture*, *Docs*. Available at:

<https://docs.walt.id/v/web-wallet/wallet-kit/issuer-and-verifier-portals/architecture> (last
accessed: 12 July 2023).

Walt.id *Identity and NFT infrastructure for developers.*, *walt.id*. Available at: <https://walt.id/>
(last accessed: 08 July 2023).

Walt.id *Introduction*, *Docs*. Available at:

<https://docs.walt.id/v/web-wallet/wallet-kit/readme> (last accessed: 08 July 2023).

Walt.id *SSI Kit*, *Docs*. Available at: <https://docs.walt.id/v/ssikit/ssi-kit/readme> (last accessed:
13 July 2023).

Walt.id *Verifiable credentials (VCS)*, *Docs*. Available at:

[https://docs.walt.id/v/ssikit/ssi-kit/what-is-ssi/technologies-and-concepts/verifiable-crede
ntials-vcs-and-verifiable-presentations-vps](https://docs.walt.id/v/ssikit/ssi-kit/what-is-ssi/technologies-and-concepts/verifiable-credentials-vcs-and-verifiable-presentations-vps) (last accessed: 14 July 2023).

Walt.id *Wallet Kit Image*, *Docker*. Available at:

<https://hub.docker.com/r/waltid/walletkit/tags> (last accessed: 08 July 2023).

Yarn *Home Page*, *Yarn*. Available at: <https://yarnpkg.com/> (last accessed: 12 July 2023).