

Masters in Cybersecurity

Identification, Authentication and Authorization

Privacy Preserving Authentication with Verifiable Credentials

Delivery 2

João Luís - 107403

Luís Leal - 103511

Ricardo Quintaneiro - 110056

June 7, 2025

Contents

1	Introduction	1
2	Scenarios	2
3	Use Cases	3
4	Requirements	4
4.1	Functional Requirements	4
4.2	Security Requirements	5
5	VC Data Model	6
5.1	Client VC Data Model	6
5.2	Product VC Data Model	6
6	System Architecture	8
6.1	Overview	8
6.2	Technologies and libraries model	8
6.2.1	Online Wine Shop	8
6.2.2	VC Issuer (Client and Product)	8
6.2.3	Product VC Verifier	8
6.2.4	ZKP Prover	9
6.2.5	ZKP Verifier	9
6.2.6	Browser Extension	9
7	Design and Implementation	9
7.1	Online Wine Shop - UI	9
7.2	Online Wine Shop - API	10
7.3	Online Wine Shop - Storage	10
7.4	Online Wine Shop - VC Holder	11
7.4.1	Document Loader Configuration	11
7.4.2	DID Methods and Resolver	11
7.4.3	Key Management	11
7.4.4	Presentation Creation and Signing	12
7.4.5	Storage and Batch Processing	12
7.5	Browser extension	12

7.6	VC Issuer (Client and Product)	14
7.6.1	DID and Key Management	14
7.6.2	Document Loader and Contexts	14
7.6.3	Credential Generation and Signing	14
7.6.4	Output	15
7.6.5	Public Key Storage	15
7.7	Product VC Verifier	16
7.7.1	Verification Workflow	16
7.7.2	API Endpoint	18
7.8	ZKP Prover	19
7.8.1	Circuit design	19
7.8.2	ZKP Prover API service	20
7.9	ZKP Verifier	21
8	Validation	22
8.1	Scenario 1: Proof of legal age	22
8.2	Scenario 2: Verify authenticity of wine	24
9	Analysis of Results	27
10	References	28

1 Introduction

The goal of this project is to design and build an authentication system that utilizes Verifiable Credentials (VC's) in conjunction with the eIDAS 2.0 principles. Our technology is made to give users access to an online platform while enhancing their control over personal data and reducing needless data sharing.

The selected use cases focus on an online marketplace that offers alcoholic beverages, where stringent age verification is required. The solution enables users to demonstrate their legal age without disclosing any other personal information, including their date of birth, by incorporating Zero Knowledge Proofs (ZKPs) into the authentication process. The user can also check to see if the beverages they are purchasing have the necessary qualities they say they have. This method not only complies with privacy-by-design guidelines, but it also shows how VC approaches are used in real-world internet transactions.

In order to do this, our approach mimics an ecosystem that is in line with eIDAS by integrating two VC Issuers — one for individuals and another for other products — as well as a VC Verifier to verify the product credentials and a VC Holder to produce the Verifiable Presentations. By facilitating communication between the user, the VC Issuer, and the VC Verifier, this design ensures that credentials can be issued, stored, and verified securely by all parties.

Our goal is to show how contemporary authentication systems may strike a compromise between strict security regulations and cutting-edge privacy safeguards.

The source code for the project can be consulted at [this repository](#).

The demonstration videos for the implemented use cases and the complete system flow can be found [here](#).

2 Scenarios

The two main scenarios we idealized for our project are depicted above.

Scenario 1: Age Verification to Purchase Alcohol

John is a customer who wants to buy a bottle of whiskey from an online store that sells alcoholic beverages. Due to legal regulations, the store must ensure that all buyers meet the minimum age requirement without collecting unnecessary personal information.

Scenario 2: Verifying Beverage Authenticity

Jane is a conscious consumer who wants to ensure that the wine she is purchasing is truly organic and from the advertised region. The online store provides Verifiable Credentials for products, ensuring transparency and trust.

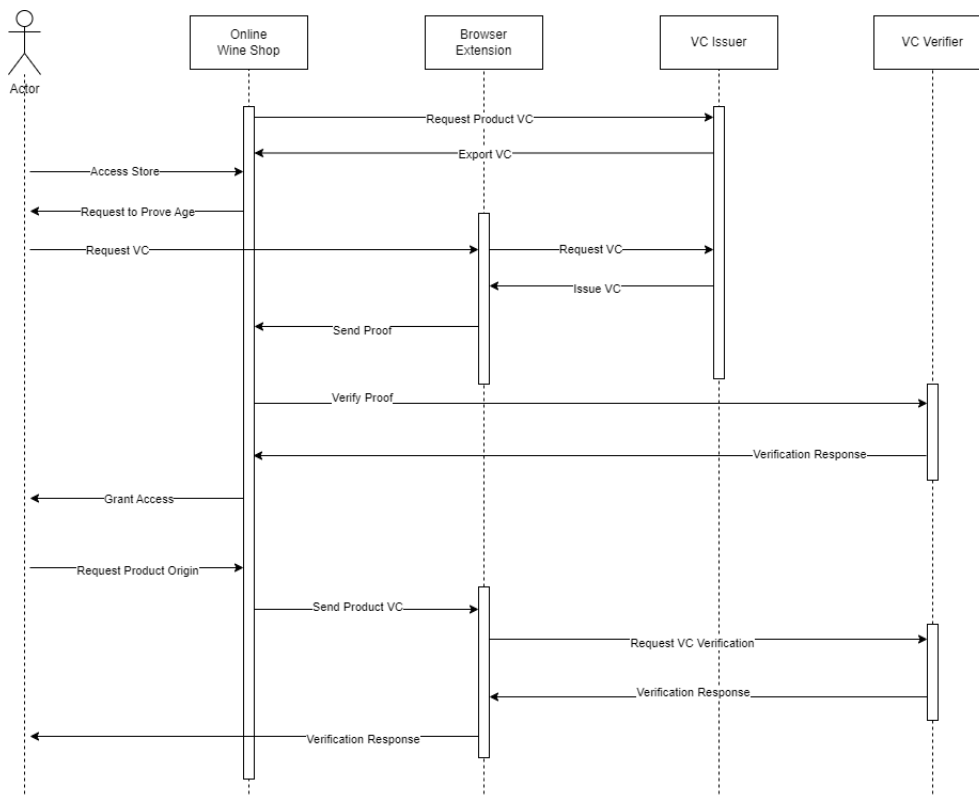


Figure 1: Sequence Diagram

3 Use Cases

Based on the previously described scenarios, the following use cases have been identified.

Use Case 1: As John, I want to prove that I am of legal drinking age without revealing my date of birth, so that I can purchase alcohol online while maintaining my privacy.

Use Case 2: As Jane, I want to verify that the organic wine I am purchasing is authentic and certified, so that I can make an informed decision without solely relying on the store's claims.

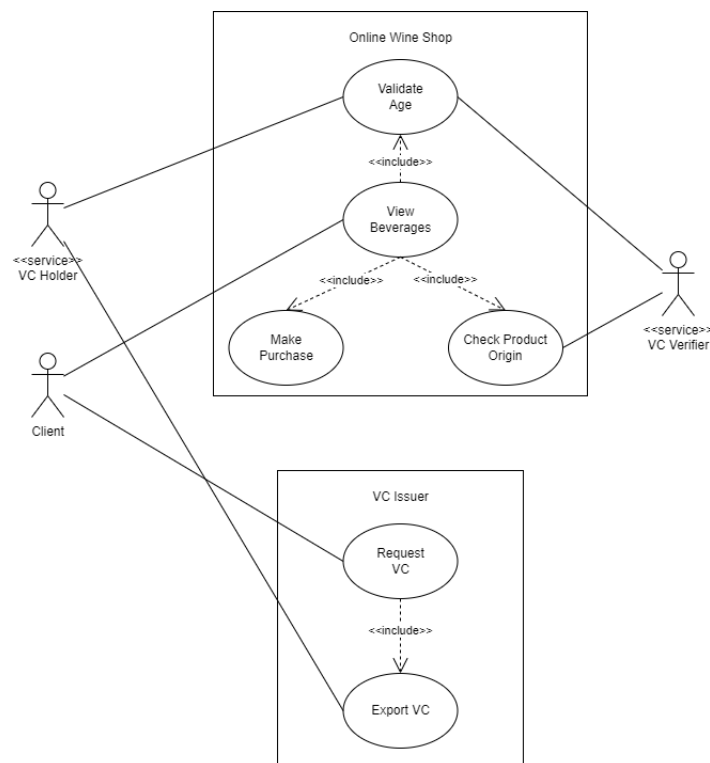


Figure 2: Use Case Diagram

4 Requirements

This chapter outlines the functional and security requirements of the system.

As the primary objective of this project is to present a proof of concept, some requirements are intended for future implementation.

4.1 Functional Requirements

- Client-related requirements:
 - FR1: A client must be able to prove that they are of legal age to access the platform by selectively disclosing only the necessary information. This requirement **was implemented**.
 - FR2: A client must be able to export their VC's. This requirement is **future work**.
 - FR3: A client must be able to verify the origin/characteristics of a product listed on the website. This requirement **was implemented**.
- Authentication-related requirements:
 - FR4: In order to view the list of products, a client must first authenticate on the platform. This requirement **was implemented**.
- Shop-related requirements:
 - FR5: The shop can only present its full functionality of listing alcohol products, etc. after the clients have proven they are of legal age. This requirement **was implemented**.

4.2 Security Requirements

<i>Confidentiality</i>	REQ-01	User authentication passphrases shall be stored in a hashed format using the Password-Based Key Derivation Function 2 with the HMAC-SHA-256 standard using 600000 iterations, as recommended by OWASP [1]. [Implemented]
	REQ-02	User authentication passphrase salts shall be obtained using a cryptographically-secure pseudo-random number generator that relies on the operating system built-in functionality. [Implemented]
	REQ-03	All communications between system components, including client-server interactions and data exchanges with external systems, shall be encrypted using version 1.3 of Transport Layer Security. [Future Work]
	REQ-04	The system shall only allow the following TLS ciphersuites, as required by eIDAS Cryptographic Requirements [2]: <ul style="list-style-type: none"> 1. TLS_AES_128_GCM_SHA256 2. TLS_AES_256_GCM_SHA384 3. TLS_CHACHA20_POLY1305_SHA256 [Future Work]
<i>Authentication</i>	REQ-05	User-created passwords shall be no less than 8 characters in length, allowing the usage of all printable characters, as specified by UTF-8, and whitespaces, according to OWASP recommendations [3]. [Implemented]
	REQ-06	The system shall enforce the use of a different password for all registrations found to be holding a blacklisted password, as revealed by the Have I Been Pwned (HIBP) API [4]. [Future Work]
	REQ-07	The system shall use a password strength meter to help users create a more complex password. [Future Work]
<i>General</i>	REQ-08	User inputs should be sanitized to avoid common security problems such as XSS, SQL injection or CSRF. [Implemented]

5 VC Data Model

This chapter presents the selected attributes for the *credentialSubject* field in both the Client VC and the Product VC.

5.1 Client VC Data Model

Since the *Website Verifier* only needs proof that the client is of legal age, we opted for a minimalist Client VC, containing the client's ID, name and date of birth.

```
"credentialSubject": {  
  "clientID": "<The Decentralized Identifier (DID) or other unique identifier of the person  
  ↪ receiving the credential>",  
  "clientName": "<Full name of the credential owner>",  
  "dateOfBirth": "<The date of birth of the credential holder, formatted as YYYY-MM-DD>"  
}
```

5.2 Product VC Data Model

The Product VC contains the product's intrinsic details. Therefore, a wine-related product VC includes a set of attributes that describe the wine's identity, origin, composition, and certifications.

A comprehensive set of attributes enables verifiable traceability, transparency, and product integrity throughout the supply chain.

```
"credentialSubject": {  
  "productId": "urn:epc:id:sgln:<barcodeValue>.<bottleNumber>",  
  "productName": "<Commercial product name>",  
  "productType": "<Product classification>",  
  "wineType": "<Product specific classification>",  
  "origin": {  
    "country": "<Origin country>",  
    "region": "<Origin region>",  
    "subRegion": "<Origin sub region>",  
    "producer": { "name": "<Producer name>", "did": "did:<Producer ID>" }  
  },  
  "certifications": [  
    {  
      "certificationType": [<List of certifications>],  
      "certifyingAuthority": "<Certification authority name>",  
    }  
  ]  
}
```

```
    "certificateId": "<Certification authority ID>",
    "certificationDate": "<Certification date>"
  }
],
"grapeVarieties": ["<List of grape varieties>"],
"alcoholContent": { "value": "<Alcohol value>", "unit": "%" },
"volume": { "value": "<Volume value>", "unit": "ml" },
"analyticalData": {
  "fixedAcidity": { "value": "<Fixed acidity value>", "unit": "g/L" },
  "volatileAcidity": { "value": "<Volatile acidity value>", "unit": "g/L" },
  "citricAcid": { "value": "<Citric acid value>", "unit": "g/L" },
  "residualSugar": { "value": "<Residual sugar value>", "unit": "g/L" },
  "chlorides": { "value": "<Chlorides value>", "unit": "g/L" },
  "freeSulfurDioxide": { "value": "<Free sulfur dioxide value>", "unit": "mg/L" },
  "totalSulfurDioxide": { "value": "<Total sulfur dioxide value>", "unit": "mg/L" },
  "density": { "value": "<Density value>", "unit": "g/cm³" },
  "pH": "<pH value>",
  "sulphates": { "value": "<Sulphates value>", "unit": "g/L" }
},
"vintage": "<Production year>",
"batchCode": "<Batch code>"
}
```

6 System Architecture

6.1 Overview

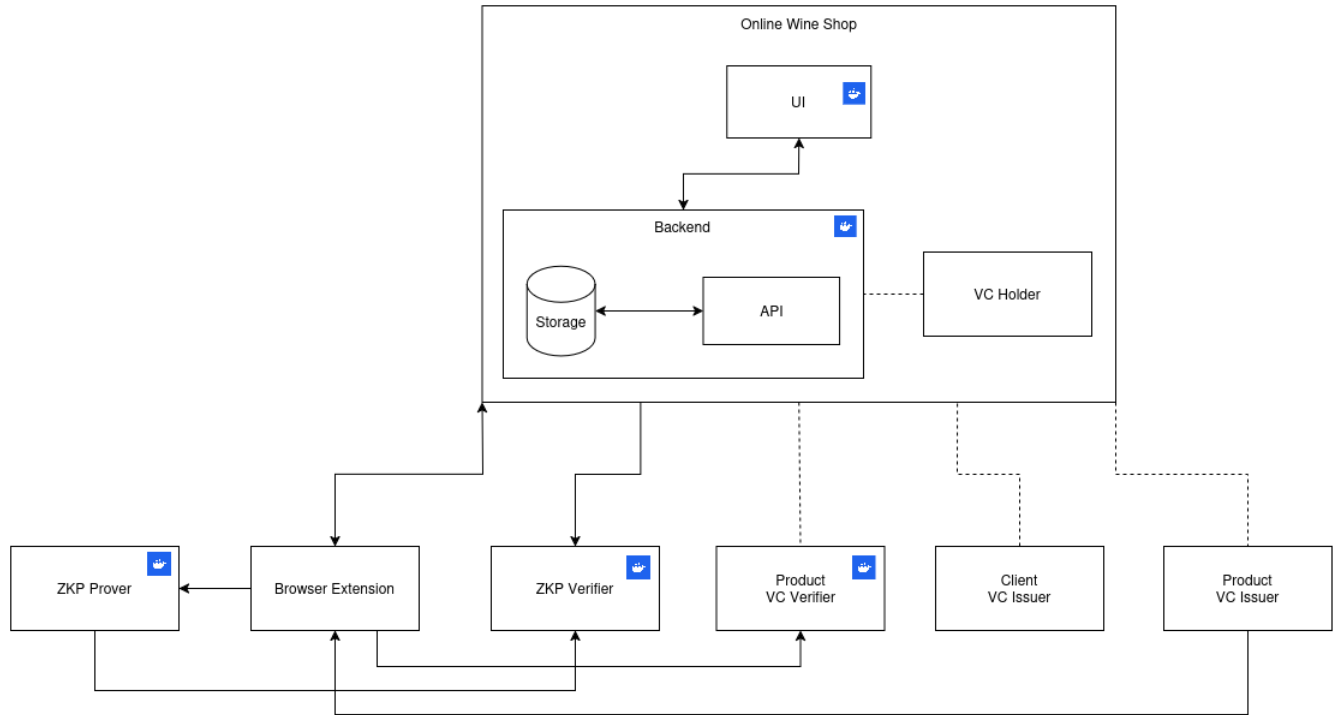


Figure 3: System Architecture

6.2 Technologies and libraries model

6.2.1 Online Wine Shop

- UI: ReactJS and Vite with Hero UI components [5]
- API: Flask RESTful
- Storage: SQLite
- VC Holder: Node.js with *digitalbazaar/vc*[6]

6.2.2 VC Issuer (Client and Product)

- Node.js with *digitalbazaar/vc*

6.2.3 Product VC Verifier

- Node.js Express with *digitalbazaar/vc*

6.2.4 ZKP Prover

- Node.js Express with *snarkjs* and *circom*

6.2.5 ZKP Verifier

- Node.js Express with *snarkjs*

6.2.6 Browser Extension

- ReactJS and Vite

7 Design and Implementation

After designing the system architecture and selecting the appropriate technologies and libraries, the previously mentioned services and modules were implemented. This section describes the implementation and functioning details of each.

The connections between the VC Holder, Client VC Issuer, and Product VC Issuer are represented by dashed lines, as these components were implemented as modules rather than services due to time constraints. Consequently, they execute once and store the resulting data, instead of responding to requests via endpoints as originally intended.

The Product VC Verifier, on the other hand, is implemented as a service providing an endpoint for Verifiable Presentation (VP) verification. However, since the VP's are loaded from local storage, the connection between the Verifier service and the Online Wine Shop is also represented by a dashed line.

7.1 Online Wine Shop - UI

As mentioned on 6.2 section, we used HeroUI, which is a ReactJS component library, to make the frontend development easier. We developed the following pages/modals in our POC:

- Age Verification Page, which communicates with the Browser Extension;
- Main Page, with a Navbar and a Footer, and one card for each available wine. This list of wines is currently available on the *frontend* project directly. In the future, this data should be behind the API.
- Sign Up component that is communicating with the API via `/register` endpoint. HeroUI allows to do direct frontend validation, such as inputting a correct format email or comply

with the password length requirement defined on 4.2 section. However, we also do this validation on the API endpoint.

- Wine Details modal, that displays the VC information stored in a JSON in a more friendly way. Again, this information is retrieved directly from the /public folder inside the frontend project folder, but in the future should be behind the API. In this modal it's also possible to check the Certificates enrolled with the VC's.

7.2 Online Wine Shop - API

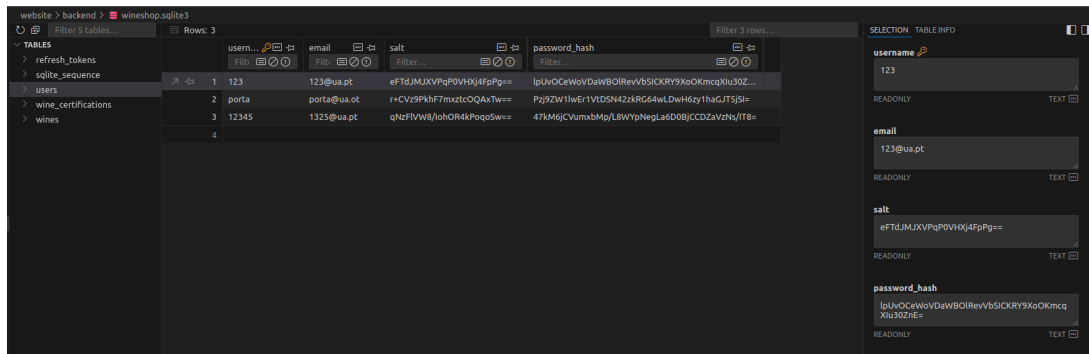
As the project developed, we ended up using the API only for authentication. Our Flask-Restful API implements JWT-based authentication with refresh token capabilities and a CORS policy for local development.

The authentication system uses PBKDF2 with SHA-256 hashing, employing 600,000 iterations for key stretching along with cryptographic salts and an additional pepper for enhanced password security. The dual-token approach utilizes short-lived JWT access tokens with 15-minute expiration and longer-lived refresh tokens lasting 7 days, stored in the database for session management.

Three endpoints handle the authentication flow: /register for user creation with password validation, /login for credential verification returning both token types, and /refresh for obtaining new access tokens without re-authentication. Security measures include timing attack protection through constant-time comparison and secure storage of sensitive configuration in environment variables.

7.3 Online Wine Shop - Storage

Our SQLite DB only stores auth data too. This database is created anew when docker compose is built.



id	username	email	salt	password_hash
123	123@ua.pt	123@ua.pt	eFTdJMJXVPqP0VHj4fPpPg==	lpUvOCeWwVdaWBOlRevVbSICKRy9XoOKmcqXlu30Z...
12345	porta	porta@ua.pt	r+CvZ9PkbF7mxztCQAXtw==	Pq9ZW1tWeF1VDSN42zRG64wLDwH6zy1haGJT5Sf=
12345	12345@ua.pt	12345@ua.pt	qNzFTVwS/ohOR4kPoqoSw==	47kM6jCVumxbMp/LBWYPneqLa6D0BjCCD2aVzNq/TT8=

Figure 4: DB structure

7.4 Online Wine Shop - VC Holder

The **VC Holder** is responsible for creating and signing **Verifiable Presentations** (VP's) from existing **Verifiable Credentials** (VC's). After generating the VP's, those will be ready to be presented to a **Verifier**. This implementation utilizes the *@digitalbazaar/vc* library along with Decentralized Identifier (DID) method drivers and cryptographic suites to ensure compliance with W3C standards and secure proof generation.

The developed code was adapted from this course's '*Lab - Authentication with ZNP and Verifiable Credentials*'[7].

7.4.1 Document Loader Configuration

A custom JSON-LD document loader is configured to resolve necessary security contexts including W3C Verifiable Credentials, Data Integrity, and a domain-specific wine certification context.

7.4.2 DID Methods and Resolver

Multiple DID method drivers (*did:key* and *did:web*) are instantiated with support for ECDSA multikey cryptography on the P-384 elliptic curve. These drivers are registered with a *CachedResolver*, which enables efficient resolution of DID Documents required during credential and presentation processing.

7.4.3 Key Management

The holder generates an ECDSA key pair from a P-384 elliptic curve for signing presentations using the *@digitalbazaar/ecdsa-multikey* library. This key pair is then associated with

a *did:web* DID corresponding to the holder's identity. Then, the public keys along with the product batch codes extracted from the credential subject are stored in a JSON file (*public-keys/vpEcdsaKeyPairs.json*) for key tracking and reuse.

7.4.4 Presentation Creation and Signing

The VP's are generated by encapsulating an existing VC alongside the holder's DID. To prevent replay attacks during verification, a challenge string is incorporated and, although the challenge is hardcoded in this implementation, in a production environment it should be randomly generated for each VP and securely stored, for example, as an environment variable.

The VP is then cryptographically signed using the *DataIntegrityProof* suite, which embeds a proof that cryptographically binds the holder's authentication key and the challenge to the presentation, ensuring the authenticity and integrity of the VP.

7.4.5 Storage and Batch Processing

Signed presentations are saved in a dedicated directory for subsequent use by verifiers but, in a production environment should be stored in a database, much like the the public signing keys, which are maintained in a JSON. This issue is addressed later in the **VC Issuer** (Client and Product) subsection [7.6.5].

This architecture ensures that the **VC Holder** fulfills its role by proving possession and control over the credentials it presents, using decentralized identifiers and cryptographic proofs aligned with W3C specifications.

7.5 Browser extension

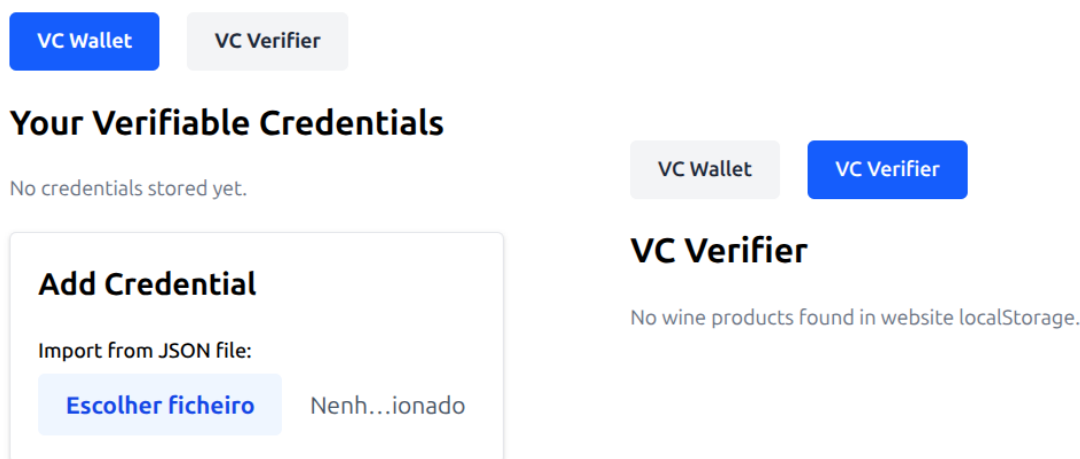
The **VC Wallet** browser extension serves as a wallet for the user to manage VC's and interact with Zero-Knowledge Proof services. It allows users to store, generate and present cryptographic proofs of their credentials, as well as verify product VP's directly from the website.

As modeled in the previous chapter, the browser extension was written in ReactJS with the Vite framework and was tested as a built Chromium-based extension only. This is because there are implementation differences in the way extensions interact with browser windows and store things in their own local storage (for Chromium, `chrome.storage.local`). The three main functionalities of this extension are:

1. Importing and storing Verifiable Credentials

2. Make proofs about these credentials - if a person is of age - and send them to the website
3. Verify the product Verifiable Proofs that the website provides

The extension has two tabs, one for the wallet and proof generation and the other for the verification of the products VP's.



The extension, to connect to the website, uses a content script and has listeners that are waiting for messages related to events in the extension UI. For example, when the user opens the verifier view, the extension requests all *wine_verification_** localStorage entries from the website, presenting them as a clickable list for verification.

```
chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
  if (message.type === "AGE_PROOF" && message.proofObj) {
    localStorage.setItem("ageProof", JSON.stringify(message.proofObj));
    sendResponse({ status: "ok" });
  } else if (message.type === "GET_WINE_VPs") {
    const wineVPs = [];
    for (let i = 0; i < localStorage.length; i++) {
      const key = localStorage.key(i);
      if (key && key.startsWith("wine_verification_")) {
        try {
          wineVPs.push(JSON.parse(localStorage.getItem(key)));
        } catch {}
      }
    }
    sendResponse({ wineVPs });
  }
});
```


To install the extension one must first run `"npm run build"` and then import it into a Chromium-based browser through the created `manifest.json`. More details about its features will follow in the validation section.

7.6 VC Issuer (Client and Product)

The **VC Issuer** module is responsible for generating Verifiable Credentials for both products and clients and it uses `@digitalbazaar`[6] to handle DID generation, credential issuance, signing, and custom context management.

The developed code was adapted from this course's *'Lab - Authentication with ZNP and Verifiable Credentials'*[7].

7.6.1 DID and Key Management

To ensure interoperability and data integrity, each issuer uses a DID based on the `did:key` or `did:web` methods. The keys are then generated using the `ecdsa-multikey` library with a P-256 elliptic curve. Afterwards, the keys are used to produce DID documents and configure the signing suite:

- A key pair is created using `EcdsaMultikey.generate()`.
- A DID document is derived from the key using `didKeyDriver.fromKeyPair()`.
- The key's identifier (`id`) and controller are extracted from the DID document.

7.6.2 Document Loader and Contexts

A custom `documentLoader` is configured to support both W3C standard credentials and application-specific contexts:

- Standard context: `https://www.w3.org/2018/credentials/v1`
- Product-specific context: `https://example.com/wine-context/v1`
- Client-specific context: `https://example.com/client-context/v1`

These contexts are added statically to the loader to ensure fast resolution and offline operability.

7.6.3 Credential Generation and Signing

For each product or client, a credential is created with the following fields:

- *@context*: Includes standard and custom contexts
- *type*: Credential type, e.g., *WineCertificationCredential* or *AgeVerificationCredential*
- *id*: A UUID-based URN
- *issuer*: The controller DID of the key pair
- *issuanceDate* and *expirationDate*
- *credentialSubject*: The subject data loaded from a dataset

The credential is signed using a *DataIntegrityProof* with the *ecdsa-rdfc-2019* cryptographic suite, and the result is a W3C-compliant Verifiable Credential.

7.6.4 Output

The module reads input data from JSON datasets:

- *wine-dataset.json* for product VC's
- *client-dataset.json* for client VC's

Each credential is saved in a designated output directory:

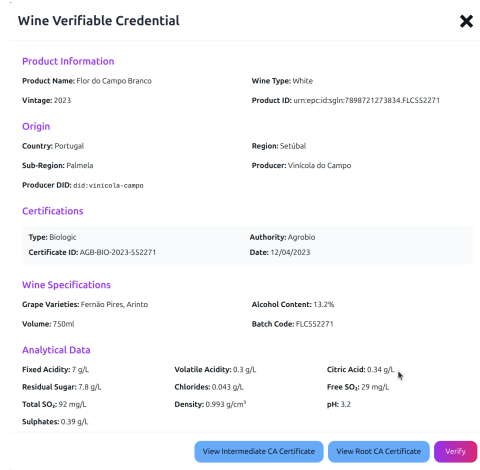
- *../wine-VCs/* for product credentials
- *../client-VC'/* for client credentials

7.6.5 Public Key Storage

In a production environment, public keys would be organized within a certification chain, where the Root Certificate Authority (Root CA) acts as the trusted root of authority, and the Issuer functions as an Intermediate CA directly subordinate to the Root CA in the hierarchy. Their certificates would be accessible for verification in a trusted database.

We attempted to simulate this setup by creating a certification chain '*product-issuer/certification-chain/create-ca.sh*' and importing the keys accordingly. However, due to formatting errors in the Node.js module *JSON Object Signing and Encryption (JOSE)*[8], this solution could not be fully implemented.

Nonetheless, the certificates can still be viewed within the product modal via the buttons labeled '*View Intermediate CA Certificate*' and '*View Root CA Certificate*', as depicted in the figure below.



The image shows a 'Wine Verifiable Credential' modal card. It is a light gray box with a close button (X) in the top right corner. The card is divided into several sections with purple headers: 'Product Information', 'Origin', 'Certifications', 'Wine Specifications', and 'Analytical Data'. Each section contains key-value pairs of information. At the bottom, there are three buttons: 'View Intermediate CA Certificate', 'View Root CA Certificate', and 'Verify'.

Product Information	
Product Name: Flor do Campo Branco	Wine Type: White
Vintage: 2023	Product ID: urn:opcids:gin:7898721273834:FLC552271

Origin	
Country: Portugal	Region: Setúbal
Sub-Region: Palmela	Producer: Vinícola do Campo
Producer DID: did:vinicola-campo	

Certifications	
Type: Biologic	Authority: Agrobio
Certificate ID: AGB-BIO-2023-552271	Date: 12/04/2023

Wine Specifications	
Grape Varieties: Fernão Pires, Arinto	Alcohol Content: 13.2%
Volume: 750ml	Batch Code: FLC552271

Analytical Data		
Fixed Acidity: 7 g/L	Volatile Acidity: 0.3 g/L	Citric Acid: 0.34 g/L
Residual Sugar: 7.8 g/L	Chlorides: 0.043 g/L	Free SO ₂ : 29 mg/L
Total SO ₂ : 92 mg/L	Density: 0.993 g/cm ³	pH: 3.2
Sulphates: 0.39 g/L		

Buttons: View Intermediate CA Certificate, View Root CA Certificate, Verify

Figure 5: Product Modal Card

7.7 Product VC Verifier

The **Verifier** is implemented as a RESTful API using *Express.js*, which listens for HTTP POST requests on the `/verify` endpoint. This service verifies the product VP's upon the client's request, ensuring the authenticity and integrity of the claims they contain.

7.7.1 Verification Workflow

Upon receiving a VP in a verification request:

1. **Extract Batch Code and Retrieve Public Key:** The batch code embedded within the VP's credential subject is used to locate the corresponding public key from a locally stored JSON file containing pre-registered keys (*public-keys/vpEcdsaKeyPairs.json*). This simulates a trusted key database.
2. **Reconstruct DID Document and Verification Method:** Using the retrieved key pair, a DID Document is generated via the *did:web* driver, including the authentication method used to sign the VP. This DID Document is added to the *documentLoader* to facilitate cryptographic proof validation.
3. **Configure Verification Suite:** The service uses the *DataIntegrityProof* suite with the *ecdsa-rdfc-2019* cryptographic suite to verify the cryptographic signature on the VP.
4. **Verify the Presentation:** The VP is verified against the challenge string (hardcoded in this prototype, as mentioned here [7.4.4]), the loaded DID documents, and the cryptographic suite to confirm that the signature is valid.

```
async function verifyPresentation({ verifiablePresentation, documentLoader, challenge }) {  
  // setup example for did:web  
  const VP_DID = "did:web:wineshop.pt:issuer:123";  
  const VP_DID_URL = "https://wineshop.pt/issuer/123";  
  
  // Load the key pair from the keys list  
  const batchCode = verifiablePresentation.verifiableCredential[0].credentialSubject.batchCode;  
  const keysFile = "./public-keys/vpEcdsaKeyPairs.json";  
  const keysList = JSON.parse(fs.readFileSync(keysFile, "utf8"));  
  
  const keyData = keysList.find(k => k.batchCode === batchCode);  
  if (!keyData) {  
    throw new Error(`No key found for batch code ${batchCode}`);  
  }  
  
  const vpEcdsaKeyPair = await EcdsaMultikey.from(keyData.keyPair);  
  
  // Create DID Document  
  const { didDocument: vpDidDocument, methodFor: vpMethodFor } =  
    await didWebDriver.fromKeyPair({  
      url: VP_DID_URL,  
      verificationKeyPair: vpEcdsaKeyPair,  
    });  
  
  // Get the authentication key and ensure it matches the one in the presentation  
  const didWebKey = vpMethodFor({ purpose: "authentication" });  
  vpEcdsaKeyPair.id = didWebKey.id;  
  vpEcdsaKeyPair.controller = vpDidDocument.id;  
  
  // setup VP ecdsa-rdfc-2019 verifying suite  
  const vpVerifyingSuite = new DataIntegrityProof({  
    cryptosuite: ecdsaRdfc2019Cryptosuite,  
  });  
  
  // Add both the DID document and the verification method to the loader  
  loader.addStatic(VP_DID, vpDidDocument);  
  
  const vpDidVm = {  
    ...vpDidDocument.verificationMethod[0],  
    "@context": [  
      "https://www.w3.org/ns/did/v1",  
      "https://w3id.org/security/multikey/v1"  
    ]  
  };  
  loader.addStatic(vpDidVm.id, vpDidVm);  
  
  // verify signed presentation  
  const verifyPresentationResult = await vc.verify({  
    presentation: verifiablePresentation,  
    challenge,  
  });  
}
```

```
    suite: vpVerifyingSuite,  
    documentLoader,  
  });  
  
  return verifyPresentationResult;  
}
```

7.7.2 API Endpoint

The `/verify` endpoint accepts a JSON payload containing the VP, parses it, and calls the verification workflow. The response includes the verification result.

```
// API endpoint for verification  
app.post("/verify", async (req, res) => {  
  try {  
    const verifiablePresentation = typeof req.body === "string"  
      ? JSON.parse(req.body)  
      : req.body.verifiablePresentation || req.body;  
    const challenge = "wine-challenge-321"; // Hardcoded for development purposes  
  
    if (!verifiablePresentation) {  
      return res  
        .status(400)  
        .json({ error: "No verifiable presentation provided" });  
    }  
  
    console.log("VERIFIABLE PRESENTATION:", verifiablePresentation);  
  
    const result = await verifyPresentation({  
      verifiablePresentation,  
      documentLoader,  
      challenge,  
    });  
  
    if (result.error) {  
      return res.status(400).json({  
        verified: false,  
        error: result.error.errors[0].message,  
      });  
    }  
  
    res.json({  
      verified: result.verified,  
      results: result,  
    });  
  } catch (error) {  
    console.error(error);  
    res.status(500).json({ error: "Internal server error" });  
  }  
});
```

```
}  
});  
  
app.listen(port, () => {  
  console.log(`Verifier API listening on port ${port}`);  
});
```

7.8 ZKP Prover

The ZKP Prover is provided as a Node.js API service. It is designed to generate zero-knowledge proofs for age verification using the Groth16 protocol and Circom circuits. It is a core component that the VC Wallet extension will need, as it enables the generation of privacy-preserving credential proofs for its users.

7.8.1 Circuit design

The core circuit, *age_check.circom*, is implemented in Circom 2.0. It takes as input:

- birthTimestamp: Unix timestamp of the user's date of birth.
- currentTimestamp: Unix timestamp of the current date.

The circuit makes sure:

- That currentTimestamp is not before birthTimestamp.
- That the difference between the two is at least 18 years (in seconds).
- The output signal isOldEnough is set to 1 if the user is at least 18 years old.

```
pragma circom 2.0.0;  
include "circomlib/circuits/comparators.circom";  
  
template AgeCheckTimestamp() {  
  signal input birthTimestamp; // Unix timestamp of date of birth  
  signal input currentTimestamp; // Unix timestamp of "now"  
  signal output isOldEnough; // 1 if yes, 0 otherwise  
  
  var eighteenYearsInSeconds = 568036800;  
  
  // Check if currentTimestamp >= birthTimestamp  
  component lt1 = LessThan(64);  
  lt1.in[0] <== currentTimestamp;  
  lt1.in[1] <== birthTimestamp;  
  signal isValidTimestamp;  
  isValidTimestamp <== 1 - lt1.out; // 1 if current >= birth
```

```

// Calculate ageDiff only if valid
signal ageDiff;
ageDiff <== isValidTimestamp * (currentTimestamp - birthTimestamp);

// Check if ageDiff >= eighteenYearsInSeconds
component lt2 = LessThan(64);
lt2.in[0] <== ageDiff;
lt2.in[1] <== eighteenYearsInSeconds;
signal isGreaterOrEqual;
isGreaterOrEqual <== 1 - lt2.out; // 1 if ageDiff >= threshold

isOldEnough <== isGreaterOrEqual;
}

component main = AgeCheckTimestamp();

```

The circuit then needs to be compiled to generate the needed files for proof and verification. It's also needed to generate a Powers of Tau file to then use it for making the Zero-Knowledge key, also used in verification. Detailed instructions are in the README file at the *extension/prover* directory.

7.8.2 ZKP Prover API service

The service is implemented in Node.js using Express and it exposes a */generate-proof* HTTP endpoint. It loads the compiled WASM circuit and .zkey proving key at runtime. Then it uses the *snarkjs* library for witness calculation and proof generation.

```

app.post('/generate-proof', async (req, res) => {
  try {
    const input = req.body;

    const wasmBuffer = fs.readFileSync(wasmPath);
    const zkeyBuffer = fs.readFileSync(zkeyPath);

    const wc = await witnessCalculator(wasmBuffer);
    const witness = await wc.calculateWTNSBin(input, 0);

    const { proof, publicSignals } = await snarkjs.groth16.prove(zkeyPath, witness);

    res.json({ proof, publicSignals });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

```

7.9 ZKP Verifier

The ZKP verifier is also a Node.js microservice, responsible for verifying zero-knowledge proofs generated by the prover service. As well as the prover, it's designed to work with Groth16 proofs and the verification key (verification_key.json) produced during the prover setup. It exposes a */verify-proof* HTTP endpoint for proof verification.

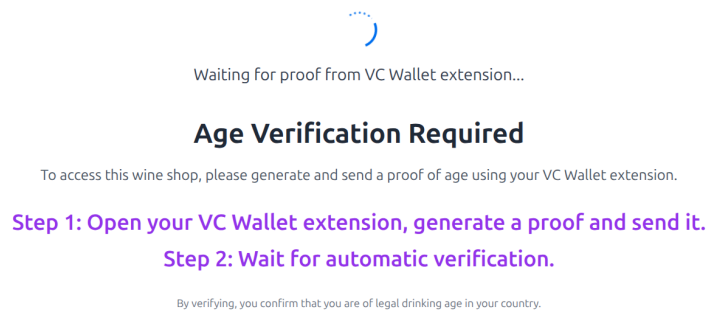
```
const vkey = JSON.parse(fs.readFileSync(path.join(__dirname, 'verification_key.json')));

app.post('/verify-proof', async (req, res) => {
  try {
    const { proof, publicSignals } = req.body;
    const verified = await snarkjs.groth16.verify(vkey, publicSignals, proof);
    res.json({ verified });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

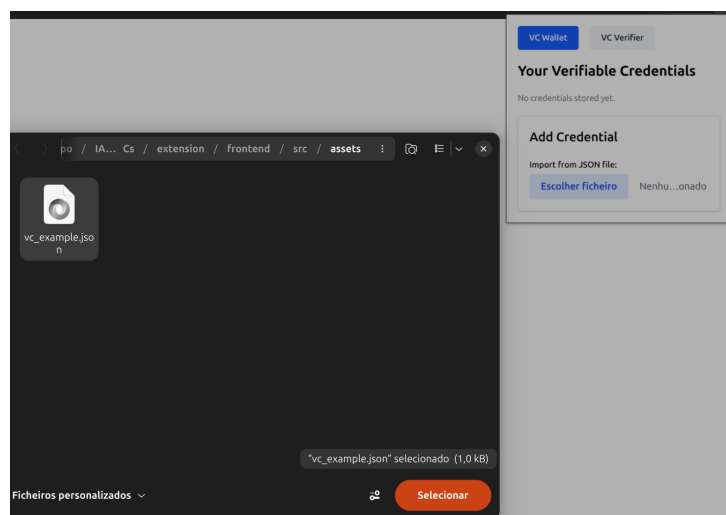

8 Validation

8.1 Scenario 1: Proof of legal age

When a user enters the website for the first time, it's greeted with a page requiring age verification.



Then the user will have to import a VC and generate a proof of that VC.



VC Wallet
VC Verifier

Your Verifiable Credentials

urn:uuid:16e737e6-f97c-4ac7-a034-618414bb6de1 from John Doe

Remove

Generate Proof

Issued: 2025-06-03 - 17:21
Expires: 2026-06-03 - 17:21

Add Credential

Import from JSON file:

Escolher ficheiro

Nenh...ionado

VC Wallet
VC Verifier

320px × 498.67px

Your Verifiable Credentials

Proof generated and stored!

urn:uuid:16e737e6-f97c-4ac7-a034-618414bb6de1 from John Doe

Send Proof to Website

Remove

Generate Proof

Issued: 2025-06-03 - 17:21
Expires: 2026-06-03 - 17:21

Add Credential

Import from JSON file:

Escolher ficheiro

Nenh...ionado

After that the user can access the site normally, as the proof that he is of age was verified by the website and it let him in.

Login
Sign

Sign In

Please enter your credentials to sign in.

Username*
Enter your username

Password*
Enter your password

Sign In
Reset

VC Wallet
VC Verifier

Your Verifiable Credentials

Proof sent to website! You can now continue on the site.

urn:uuid:16e737e6-f97c-4ac7-a034-618414bb6de1 from John Doe

Send Proof to Website

Remove

Generate Proof

Issued: 2025-06-03 - 17:21
Expires: 2026-06-03 - 17:21

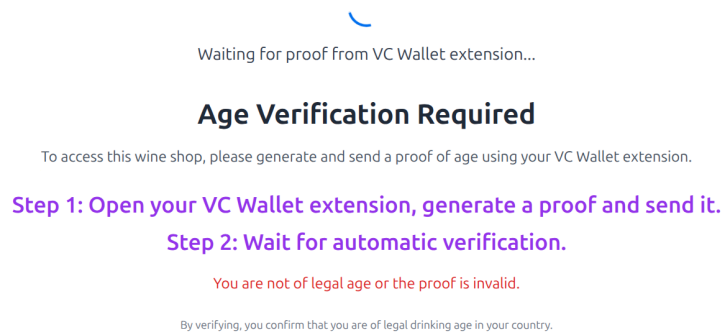
Add Credential

Import from JSON file:

Escolher ficheiro

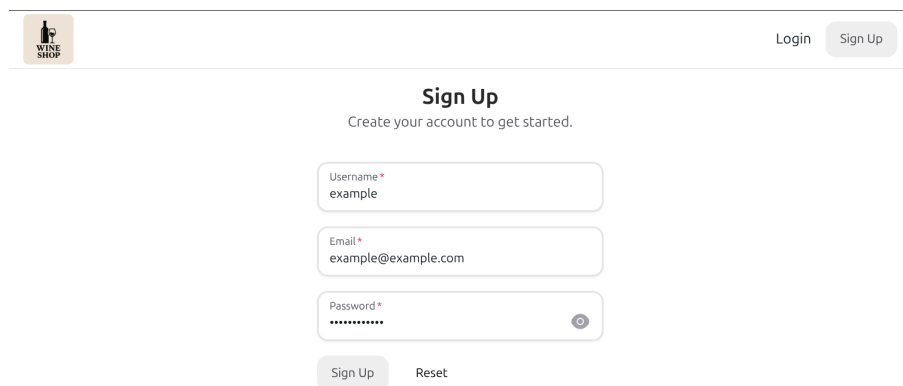
Nenh...ionado

Otherwise, if the user is not of age it will present an error:




8.2 Scenario 2: Verify authenticity of wine

To access the list of products, the user first has to sign in. If the user has no account it needs to register one:



The screenshot shows a web interface for a wine shop. At the top left is a logo with a wine glass and the text "WINE SHOP". At the top right are "Login" and "Sign Up" buttons. The main heading is "Sign Up" with the subtext "Create your account to get started." Below this are three input fields: "Username *" with the value "example", "Email *" with the value "example@example.com", and "Password *" with masked characters and a toggle icon. At the bottom are "Sign Up" and "Reset" buttons.

After that, the user can sign in to the account:



LoginSign Up

Sign In

Please enter your credentials to sign in.

Username *

example


Password *

••••••••

👁

Sign InReset

When the user is signed in he will see the list of available wines, which he can then click details and then click verify. This will send a VP of the wine to local storage which is then listed in the extension.




Welcome, example ✕

Featured Wines

Discover our curated selection of premium wines

Sale



☆☆☆☆ (4.9)


Pomadão Alvarinho
White • 2023
Minho, Portugal
Elegant and complex with notes of blackcurrant and cedar

\$299.99

~~\$349.99~~

Add to CartDetails

New




☆☆☆☆ (4.7)

Vale Secreto Tinto
Red • 2022
Borba, Portugal
Rich and powerful with earthy undertones and cherry notes

\$89.99

Add to CartDetails



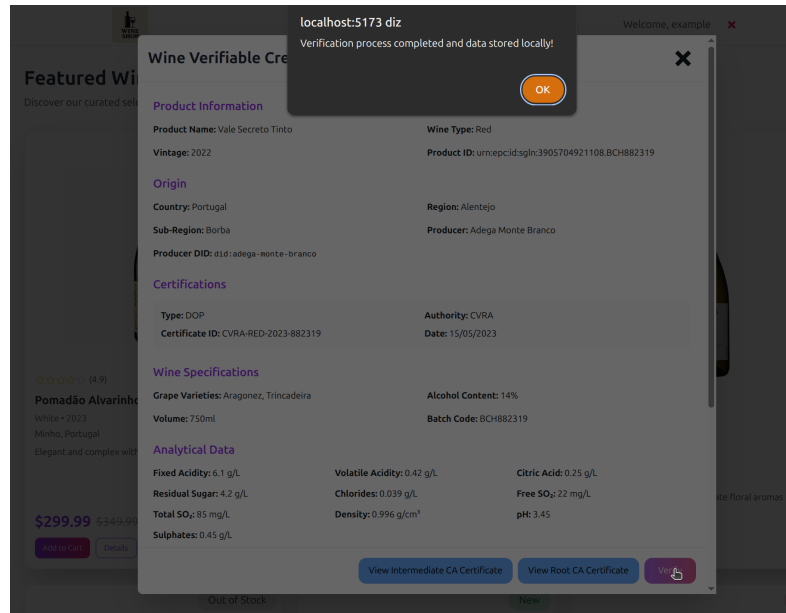
☆☆☆☆ (4.8)

Flor do Campo Branco
White • 2023
Palmela, Portugal
Crisp and effervescent with delicate floral aromas

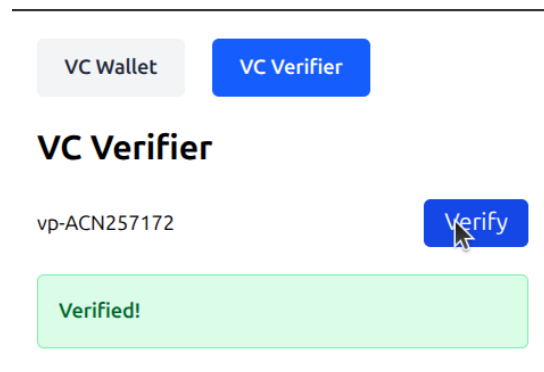
\$199.99

Add to CartDetails

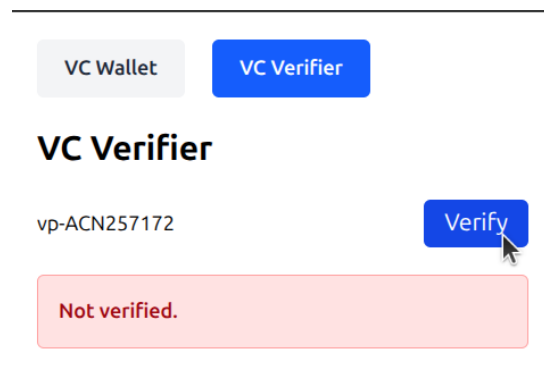
25



When listed in the extension, the wines VP's can be verified as true:



Or false (in this case, for demonstration, changing a proofValue in the localStorage by hand to originate a false verification):



9 Analysis of Results

Overall, we feel like this work has exemplified well the practical application for using Verifiable Credentials (VC's) to establish trust between consumers and online store platforms. Not only did we manage to guarantee that there is a way for the shop to allow only legal-age customers to interact with it, but we also allowed the customers to know if the products they were buying had guarantees of being real and authentic. This dual verification - of both user and product - has highlighted the potential of VC's to increase security and data integrity across digital commerce platforms. And the ability for users to verify the authenticity of the products via associated VC's contributes to greater transparency and consumer trust.

While the current implementation meets the intended goals, there is room for improvement. Future work should focus on the issuers and website holder being distributed services that can be queried when the user or website needs VC's/VP's. It should also focus on improving the security of the system as whole, such as encrypting communications between services with TLS 1.3. This would increase scalability and usability, and also align the system more closely with real-world eIDAS-compliant infrastructures.

10 References

- [1] OWASP. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#pbkdf2.
- [2] eIDAS eID Technical Subgroup, *Eidas cryptographic requirements for the interoperability framework*. [Online]. Available: https://ec.europa.eu/digital-building-blocks/sites/display/DIGITAL/eID+Profile?preview=/467109280/817168531/eIDAS%20Cryptographic%20Requirement%20v.1.4.1_final.pdf.
- [3] OWASP. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#implement-proper-password-strength-controls.
- [4] T. Hunt. “Pwned passwords api (v3).” (2024), [Online]. Available: <https://haveibeenpwned.com/API/v3#PwnedPasswords>.
- [5] N. Inc, *Hero ui*. [Online]. Available: <https://www.heroui.com/>.
- [6] D. Bazaar, *Verifiable credentials (vc) library*. [Online]. Available: <https://github.com/digitalbazaar/vc>.
- [7] J. P. Barraca, *Lab - authentication with znp and verifiable credentials*. [Online]. Available: https://sweet.ua.pt/jpbarraca/course/iaa/lab3_zkp/#verifiable-credentials.
- [8] Cisc0, *Node-jose*. [Online]. Available: <https://github.com/cisco/node-jose>.