**Technical Design and Implementation Document: Integration of Atala Prism (Hyperledger Identus) with the "Digital Voter ID" System**

**Project:** Digital Voter ID using Atala Prism (Proof-of-Concept)
**Author:** David Tacuri
**Date:** Aug 4, 2025
**Version:** 1.0

## 1. Introduction to the Digital Voter ID System and Self-Sovereign Identity (SSI)

### 1.1 Purpose and Scope of the Document

The main purpose of this document is to serve as a comprehensive technical guide for integrating Hyperledger Identus with the "Digital Voter ID" project.
It aims to provide an in-depth understanding of the architecture, workflows, underlying technologies, and practical considerations for its implementation.
This report is intended for developers, software architects, and technical leaders involved in the design, development, and deployment of decentralized identity solutions.

### 1.2 Fundamental Concepts of Self-Sovereign Identity (SSI) and Their Relevance

Self-Sovereign Identity (SSI) is a digital identity paradigm that grants individuals full control over their personal information and credentials.
Unlike traditional identity systems that rely on centralized authorities, SSI allows users to decide when, with whom, and what data to share, enhancing privacy and reducing reliance on third parties.

Key components of SSI include:

- **Decentralized Identifiers (DIDs):** Unique, user-controlled identifiers that do not depend on a centralized authority for registration.
  A DID is associated with a DID Document that contains cryptographic material (such as public keys), verification methods, and services that allow a DID controller to prove control over it.
  DIDs are resolved in a Verifiable Data Registry (VDR).
- **Verifiable Credentials (VCs):** Cryptographically signed digital claims issued by an issuer, attesting to attributes about a subject.
  VCs are designed to be cryptographically secure, privacy-respecting, and machine-verifiable.
- **Identity Agents:** Software components that manage DIDs, VCs, and communication within the SSI ecosystem.
  There are three main roles: **Issuer Agent** (issues VCs), **Holder Agent** (stores and presents them), and **Verifier Agent** (requests and validates them).

- **Secure Communication Protocols (DIDComm):** Enable private and secure message exchange between different agents.

The relevance of SSI for the "Digital Voter ID" project is fundamental, as it ensures security, scalability, transparency, and decentralization in managing voters' digital identities, reducing the risk of identity fraud.

### 1.3 Overview of the "Digital Voter ID" Project and Its Objectives

The "Digital Voter ID" project aims to transform political affiliation and voting processes through a digital system that improves transparency, security, and accessibility. It proposes implementing a political affiliation system based on SSI, using Atala Prism (now known as Hyperledger Identus) and the Cardano blockchain.

The specific technical objectives of integrating Hyperledger Identus are:

- **Implement data sovereignty:** Allow citizens (Holders) full control over their personal information and decide when and with whom to share their identity credentials.
- **Provide verifiable digital identity:** Create verifiable credentials (VCs) for voter digital identification, based on the W3C standard, that can be issued and verified cryptographically.
- **Use the Cardano blockchain as a VDR:** Leverage Cardano's immutability and permissionless nature to register and resolve issuer DIDs, ensuring trust and transparency.
- **Establish secure communication channels:** Enable secure peer-to-peer communication between parties (issuers, holders, verifiers) using the DIDComm protocol.
- **Facilitate the management of political affiliations:** Integrate credential issuance and verification capabilities to securely and transparently manage party affiliations and disaffiliations.

## 2. General System Architecture and Key Components

### 2.1 General Architecture Diagram

The "Digital Voter ID" system is based on a distributed architecture that integrates centralized components with the decentralized ecosystem of Hyperledger Identus and Cardano.

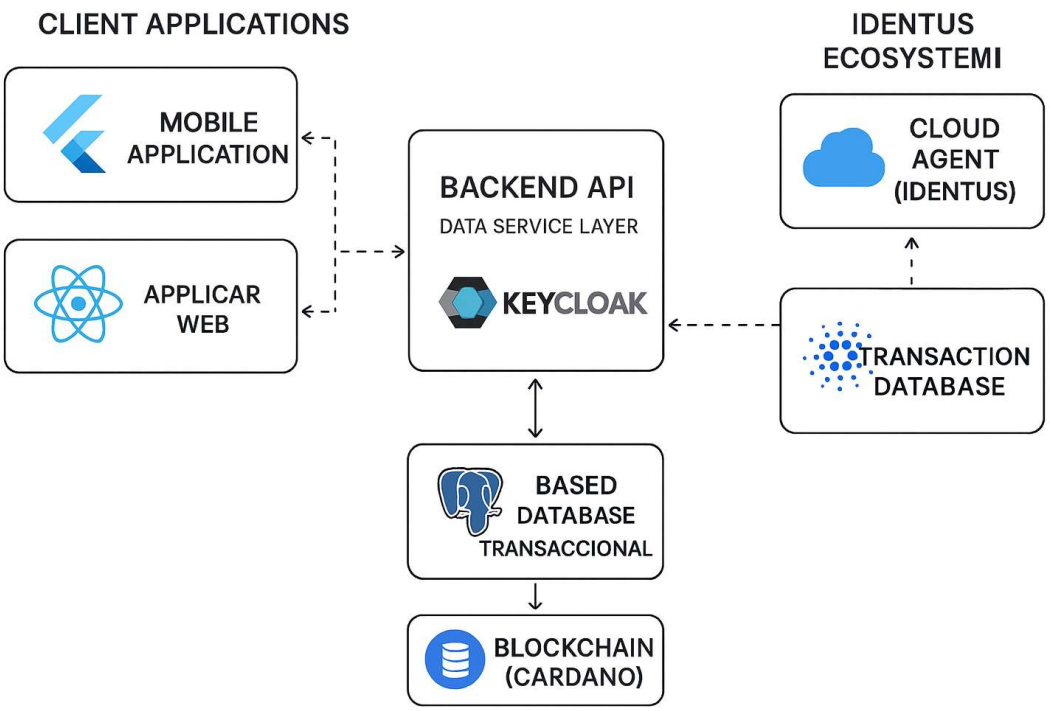


*Diagram description:* The main components and their interactions are structured as follows:

- **Client Applications:** Include a Mobile Application (developed in Flutter for citizens) and a Web Application (developed in React for political organization management and electoral administration).
- **Backend API (Data Service Layer):** A backend API layer that handles business logic and acts as a bridge between the client applications, the Identus ecosystem, and the database (transactional backend built with ASP.NET Core).
- **Transactional Database (PostgreSQL):** For storing operational system data (users, affiliations, organizations, temporary credential backups).
- **Authentication System (Keycloak):** Manages users and roles, providing authentication and authorization with Role-Based Access Control (RBAC) and multi-tenancy support.
- **Identus Ecosystem:** The Cloud Agent (Identus) is the central SSI component, interacting with the Cardano Blockchain as a Verifiable Data Registry (VDR).

At a high level, client applications interact with the Backend API for business logic and authentication.
The Backend API, in turn, communicates with the Identus Cloud Agent for all SSI-related operations (DID and VC management) and with the Cardano Blockchain (either through the Cloud Agent or directly for certain operations) for immutable data registration and

verification.
Keycloak handles user authentication and authorization for application functionalities.

### 2.2 Components of the "Digital Voter ID" System

#### 2.2.1 Client Applications (Flutter Mobile and React Web)

The client applications are the direct interaction points for system users and administrators.

**Mobile Application (Flutter App):** Designed for the citizen.

- **Framework:** Flutter SDK (Dart).
- **Key Features:**
  User registration, creation and management of the decentralized wallet and associated DID, political party selection, viewing affiliation credentials, and presenting them for verification.
  Also sends instant alerts about changes in affiliation status.
- **Security Best Practices:**
  - **Secure Storage:** Use flutter_secure_storage to securely store private keys, DIDs, and VCs on the user's device, leveraging the OS secure storage capabilities (Keychain in iOS, Keystore in Android). Avoid hardcoding secrets in code.
  - **Biometric Authentication:** Integrate libraries such as biometric_storage or local_auth to protect access to sensitive data and SSI operations via fingerprint or facial recognition.
  - **Secure Communication:** All network communications must be performed exclusively over HTTPS to ensure encryption in transit and prevent man-in-the-middle attacks.
  - **Input Validation:** Implement rigorous validation of all user input on both client and server sides to prevent vulnerabilities.
  - **Code Obfuscation:** Flutter's build process allows obfuscating Dart code, making reverse engineering more difficult.
  - **Dependency Updates:** Keep the Flutter SDK and all dependencies updated to include security patches. Compromising the mobile wallet would put the user's identity sovereignty at risk.

**Web Application (React Web App):** Designed for internal and external users, enabling advanced political affiliation management.

- **Framework:** React.js (TypeScript).
- **Key Features:** Interface for political organizations to issue affiliation credentials, audit tools to verify affiliation status, and an electoral administration dashboard. Citizens can check their registration status and access statistics.
- **Security Best Practices:**
    - **Integration with Identus SDKs:** The Identus TypeScript SDK (@hyperledger/identus-sdk) is essential for the web application to interact with the Cloud Agent for DID and VC operations, including modules such as Apollo (cryptography), Castor (DIDs), Mercury (DIDComm V2), and Pluto (agnostic storage).
    - **State Management:** Use libraries such as Redux, MobX, Recoil, or Zustand for global state management.
    - **Authentication:** Integrate with Keycloak using libraries like Auth0 React SDK or JWT to handle authentication and route protection.
    - **XSS/CSRF Protection:** While React automatically escapes HTML content, manual sanitization with sanitize-html is necessary when untrusted HTML is allowed. Implement CSRF tokens for backend requests.
    - **Use of HTTPS:** All communications with the backend and the Identus Agent must be over HTTPS.
    - **Minimizing Sensitive Data in Frontend:** Avoid serializing sensitive data directly in the frontend and ensure that authorization logic resides in the backend. Treat the web app as a "thin client" for security, delegating critical SSI operations to the backend and Identus Cloud Agent.

**2.2.2 Backend API and Transactional Database (PostgreSQL)**

These components form the logical and persistence core of the "Digital Voter ID" system.

**Backend API (Data Service Layer):** Acts as the central business logic.

- **Functions:** Validate political affiliations, generate verifiable credentials, manage revocation processes, associate DIDs with internal system users, and serve as an integration point for communication with the Identus Agent.
- **Suggested Frameworks:**
    - **Node.js:** NestJS (TypeScript-first, modular, scalable) or Express.js (minimalist and flexible). Interaction with Identus can be performed using the @hyperledger/identus-cloud-agent-client package.

- **ASP.NET Core:** C# for high performance, strong typing, native integration with Entity Framework Core for PostgreSQL, advanced security support, and API documentation with Swagger/OpenAPI to facilitate testing and client integration.
- **Java:** Spring Boot (simplifies Spring applications, microservices, REST APIs, JPA). Interaction with Identus would be through direct HTTP calls to its REST API.

**Transactional Database (PostgreSQL):** Stores internal system data such as users, affiliations, organizations, and temporary credential backups.

- It is crucial that this database does **not** store complete VCs or DID private keys, nor biometric information (fingerprints, facial images, facial patterns, etc.), since these must be processed and verified locally on the citizen's device and discarded after validation.
- Its role is to manage transactional data and metadata that link traditional business logic with the SSI ecosystem.
- VCs are stored in the Holder Agent (citizen's mobile application) and DIDs are published to the VDR (Cardano blockchain).
- **Schema Design:** Will require tables for users (with a field for the associated DID), political_organizations, affiliations (linking user and organization), and possibly tables to manage internal revocation status or VC metadata.

### 2.2.3 Authentication and Identity Management System (Keycloak)

Keycloak is an open-source Identity and Access Management (IAM) solution that provides authentication and authorization with Role-Based Access Control (RBAC), user management, and multi-tenancy support.

- **Integration:** Keycloak can integrate with the backend and client applications through standard protocols such as OpenID Connect (OIDC) and OAuth 2.0. This enables Single Sign-On (SSO), user federation, and centralized management of access policies.
- **Custom SPIs (Service Provider Interfaces):** Keycloak allows the creation of custom SPIs.
  This opens the possibility of developing an SPI that authenticates users using DIDs and Verifiable Credentials instead of traditional credentials.
  For the Digital Voter ID project, we will use an **Authenticator SPI**.

## 2.3 Hyperledger Identus Ecosystem (Atala PRISM)

Hyperledger Identus, built on the robust Atala PRISM protocol, is a comprehensive suite of tools for developing decentralized identity solutions.
It operates as a Layer-2 solution over a Distributed Ledger Technology (DLT), using the blockchain as a **Verifiable Data Registry (VDR)**.

### 2.3.1 Identus Agent (Cloud Agent): Functions and Modules

The Identus Agent is the central component for all SSI operations within the system, acting as a container that manages DIDs and the issuance, holding, and verification of credentials.
Its key modules, according to the TypeScript SDK structure, include:

- **Apollo:** Provides the necessary cryptographic operations, including support for curves such as Ed25519.
- **Castor:** Manages all DID-related operations, including creation, management, and resolution.
- **Mercury:** Handles DIDComm V2 message processing, enabling secure communication between agents.
- **Pluto:** Offers an agnostic interface for storage operations, allowing flexibility in the persistence of the agent's internal data.
- **Agent (high-level component):** Combines the functionalities of the above modules to provide basic edge agent capabilities, including DIDComm V2 protocol implementation.

### 2.3.2 DIDComm V2 Protocol: Secure Peer-to-Peer Communication

DIDComm is the secure and private communication protocol used by Identus agents to exchange credential messages and connection invitations.
It is fundamental for establishing verifiable interactions between individuals, organizations, and devices.

A key component of DIDComm in Identus is the **Mediator**.
The Mediator facilitates the routing of encrypted messages between agents, especially when they are not directly online (such as mobile applications).
It allows messages to be stored and retrieved when the Holder reconnects, which is essential for the usability and reliability of the "Digital Voter ID" mobile application.

### 2.3.3 PRISM Node and Its Interaction with the Cardano Blockchain

The PRISM Node (or the internal node within the Identus Agent) is the module responsible for interacting directly with the Cardano blockchain, which functions as the system's Verifiable Data Registry (VDR).

Cardano is chosen as the VDR for its low transaction costs, high security, and compatibility with self-sovereign identity (SSI) systems.

Technically, Cardano contributes to the "Digital Voter ID" project in several ways:

- **Immutable Affiliation Records:** The Cardano blockchain is used to immutably register citizens' affiliations with political organizations, ensuring traceability, transparency, and auditability of each transaction.
- **DID Publication:** Cardano serves as the registry where issuers publish their DIDs, which can be accessed by anyone to verify the issuer's identity.
- **Smart Contracts:** Cardano smart contracts (written in Aiken) can be used to automate validations and audits related to affiliation registration.
- **Decentralized Trust Mechanism:** The public, permissionless nature of Cardano means anyone can join the network and verify the existence of a published DID, reinforcing trust in the system by eliminating the need for centralized intermediaries for identity validation.

To function as a VDR, several Cardano components are configured:

- **Cardano Node:** Establishes direct communication with the Cardano blockchain, serving as the entry point for interacting with the ledger.
- **Cardano DB-Sync:** Includes a PostgreSQL database and a process that reads all block data from Cardano and stores it in the database, enabling efficient blockchain data queries and retrieval.
- **Cardano Wallet:** Required to cover small transaction fees when publishing DIDs or performing other blockchain operations. This wallet is internal to the Identus Agent.
- **Node IPC (Inter-Process Communication):** A socket connection (node socket) facilitates communication between the Cardano Wallet, DB-Sync, and Cardano Node, allowing for transaction submission and block data retrieval.

#### 2.3.4 Identus Internal Database (PostgreSQL) and Vault Server

The Identus Agent uses its own PostgreSQL instance for storing internal operational data. This database is separate from Cardano DB-Sync and is used to manage DIDs, credentials, and internal connection states of the agent.

Additionally, the **Vault Server** is an optional but critical component for production environments, used for securely storing sensitive credential information such as private keys.

**Key distinction:**

- The internal database stores the agent's operational state (locally generated DIDs, active DIDComm connections, held VCs).
- Cardano DB-Sync is a replica of public blockchain data.

#### 2.3.5 API Gateway and Swagger UI

The Identus Agent exposes its functionalities through an API Gateway (API 6), acting as a reverse proxy. This gateway routes traffic to the agent's various subcomponents and exposes RESTful APIs for external interaction.

**Swagger UI** is provided alongside the API Gateway as a graphical interface to explore and test the REST APIs exposed by the Identus Agent.
This tool is invaluable during development and debugging phases, mitigating the lack of extensive external documentation.

## 3. Detailed Technical Implementation Guide

### 3.1 Cardano Environment Setup as a Verifiable Data Registry (VDR)

Implementing the VDR with Identus on Cardano requires deploying and configuring several Cardano components, with **Docker Compose** being the recommended method to simplify the process.

#### 3.1.1 Deploying Cardano Components (Node, DB-Sync, Wallet) with Docker Compose

The easiest way to configure the Cardano environment for Identus is by using Docker Compose.
This approach allows you to spin up all required services in isolated containers, making environment management and replication easier.

**General deployment steps:**

1. **Install Docker and Docker Compose:** Ensure Docker (v17.06.2-ce or higher) and Docker Compose (v1.14.0 or higher) are installed.
2. **Clone the Identus Repository:** The main Hyperledger Identus repository (hyperledger-identus/hyperledger-identus) contains the Docker Compose configuration in the identus-docker directory.
3. **Configure Environment Variables:** The Identus Agent requires specific environment variables to interact with Cardano.
    - NODE_LEDGER → specifies that Cardano will be the VDR.
    - DOCKER_HOST → allows communication with Cardano services running on the Docker host.

- Keycloak integration variables: KEYCLOAK_ENABLED, KEYCLOAK_URL, KEYCLOAK_REALM, KEYCLOAK_CLIENT_ID, KEYCLOAK_CLIENT_SECRET.

4. **Run Docker Compose:** Use the provided docker-compose.yml file to start the following services:
    - PostgreSQL (for DB-Sync)
    - Cardano Node
    - Cardano Wallet (internal to Identus Agent)
    - Icarus (GUI for the wallet)
5. **Synchronize the Cardano Node:**
   Once services are up, the Cardano Node must sync with the network (preprod or mainnet).
   This process can take hours but can be accelerated using snapshots.
    - Check sync status: cardano-cli query tip --testnet-magic 1.
    - Monitor progress: via node container logs or repeated tip queries.
6. **Set up and synchronize Cardano DB-Sync with PostgreSQL:**
    - Prepare PostgreSQL: install, run, and create a database and user with privileges (using .pgpass is recommended).
    - Start DB-Sync: reads blocks from the preprod genesis and populates the PostgreSQL database. Monitor progress in logs.
    - On-chain queries via GraphQL: deploy **Cardano GraphQL** (with Hasura) on top of DB-Sync for efficient blockchain data queries.

### 3.1.2 Funding the Cardano Wallet for Transaction Fees

It is essential for the Cardano wallet associated with the Identus Agent (e.g., prism-wallet-issuer) to have sufficient ADA to cover small transaction fees when publishing DIDs or performing other blockchain operations.

**Process:**

1. **Create the Wallet:** The wallet can be created using the Icarus interface or via Cardano CLI commands.
2. **Send Test Funds:** In development or test environments, send **test ADA** to a wallet address, often obtained from a Cardano Testnet faucet.

3. **Configure the Identus Agent:** Assign the Cardano wallet ID and passphrase to the Identus Agent environment variables, along with a payment address. This allows the agent to autonomously perform on-chain transactions.

### 3.2 Decentralized Identifier (DID) Lifecycle Management

Managing DIDs is a central aspect of any SSI implementation.

#### 3.2.1 DID Creation and Publication (Long-Form to Short-Form)

- **Local Creation:** The process begins with the DID Controller (backend or client application) interacting with the Identus Cloud Agent.
  The Identus Agent manages the private keys and creation operation content, allowing the Controller to specify public keys and associated services.
  PRISM DIDs can be created fully offline without blockchain interaction.
- **Publication on Cardano:** Once created locally, the DID can be published on the Cardano blockchain.
  This publication transforms the DID from a **long-form** (self-contained) to a **short-form** DID (requiring blockchain resolution to obtain the full DID Document).
- **Mechanism:** The Identus Agent uses the DID's MASTER key to sign the publication operation and sends it to the Cardano blockchain.
  This is asynchronous and requires a specific number of block confirmations (currently 112) for the operation to be processed and considered published.
- **Monitoring:** Publication progress can be monitored through Identus Agent logs and blockchain queries until the DID status changes to **PUBLISHED**.

#### 3.2.2 DID Resolution and Issuer Verification

DID resolution is the process of retrieving the DID Document associated with a DID.

- Any client can query a PRISM Node (or an Identus Agent) to obtain the DID's associated information.
- DID Resolvers use PRISM Node output to build the current DID Document, which contains public keys and services needed to interact with the DID subject.
- In credential verification, resolving the issuer's DID on the Cardano blockchain is a critical step in establishing trust in the credential.

#### 3.2.3 DID Update, Key Rotation, and Deactivation

DIDs are not static; their associated documents may require updates.

- **Update:** Users can update DID Documents by publishing on-chain update operations (e.g., adding/modifying services or rotating cryptographic keys).

- **Key Rotation:** Rotating the keys associated with a DID is a fundamental security practice, enabling replacement of compromised keys without changing the DID, thereby proactively mitigating risk.
- **Deactivation:** A DID can be deactivated if the subject no longer wishes to use it or if a severe compromise is detected. Deactivation is performed by publishing an on-chain deactivation operation.
  Once deactivated, all DID Document content is removed.

**Table: DID Operations and Implementation with Identus (APIs/SDKs)**

| DID Operation | Identus/Cardano Component | Relevant API/SDK | Technical Description | Blockchain Status |
|---|---|---|---|---|
| **Creation** | Cloud Agent (Castor module) | Cloud Agent REST API, Identus SDKs (TypeScript Castor.createDID()) | DID Controller interacts with Cloud Agent to generate DID and keys locally. | Local, not on-chain until published. |
| **Publication** | Cloud Agent (PRISM Node), Cardano VDR | Cloud Agent REST API, Identus SDKs (TypeScript Castor.publishDID()) | Agent signs publication with MASTER key and sends it to Cardano, converting DID from long-form to short-form. | On-chain, requires 112 confirmations. |
| **Resolution** | PRISM Node, Cardano DB-Sync | DID Resolvers, Cloud Agent REST API (for local/cached resolution) | Any client or agent queries PRISM Node to obtain DID Document. DB-Sync accelerates on-chain lookups. | On-chain (for published DID Document) and off-chain (for faster resolution). |

| | | | | |
|---|---|---|---|---|
| **Update** | Cloud Agent (PRISM Node), Cardano VDR | Cloud Agent REST API, Identus SDKs (TypeScript Castor.updateDID()) | Publishes update operations on-chain to modify DID Document. | On-chain, requires confirmation. |
| **Key Rotation** | Cloud Agent (PRISM Node), Cardano VDR | Cloud Agent REST API, Identus SDKs (TypeScript Castor.rotateKey()) | A specific update replacing associated cryptographic keys. | On-chain, requires confirmation. |
| **Deactivation** | Cloud Agent (PRISM Node), Cardano VDR | Cloud Agent REST API, Identus SDKs (TypeScript Castor.deactivateDID()) | Publishes on-chain operation marking DID as deactivated, preventing future use. | |

### 3.3 Verifiable Credential (VC) Lifecycle Implementation

Verifiable Credentials are the means by which claims about a voter's identity are issued, stored, presented, and verified in the system.

#### 3.3.1 Issuing Affiliation Credentials (Claims, Digital Signature)

- **Flow:** The issuance process begins when a citizen selects a political party in the mobile application.
  The backend validates that no prior affiliations exist.
  The Issuer Agent generates a Verifiable Credential (VC) containing the user's claims, such as name, affiliation date, political party, and a consent hash.
- **Digital Signature:** The VC is cryptographically signed by the Issuer Agent to ensure its authenticity and integrity.
  Once signed, the VC is sent to the Holder Agent (citizen's wallet) using the DIDComm protocol, ensuring secure and private transmission.
- **Standards:** Claims inside the VC must adhere to the W3C Verifiable Credentials Data Model.

#### 3.3.2 Secure VC Storage in the Holder Agent

The Holder Agent, residing in the citizen's mobile application, is responsible for securely storing received VCs.
This storage is critical to maintaining the citizen's sovereignty over their identity.

- **Mechanism:** In the Flutter mobile application, the flutter_secure_storage library will be used to store VCs encrypted directly on the device, leveraging the secure storage capabilities of the operating system (Keychain on iOS, Keystore on Android).

### 3.3.3 Credential Verification by Third Parties (Signature, Revocation, Issuer DID)

Verification is the process by which an entity (Verifier) validates the authenticity and validity of a VC presented by a Holder.

- **Flow:** A verifying entity (e.g., an electoral authority) requests validation of a citizen's affiliation.
  The user, through their mobile application, approves sharing the VC.
  The Identus Verifier Agent performs the following checks:

  1. **Digital Signature Validation:** Ensures the VC has not been altered and was signed by the legitimate Issuer Agent.
  2. **Revocation Status:** Checks the VC's revocation status to confirm it has not been invalidated.
  3. **Issuer DID Resolution:** Resolves the issuer's DID on the Cardano blockchain to obtain its DID Document and verify the public key used for signing.

### 3.3.4 Credential Revocation (StatusList2021)

Credential revocation is essential to invalidate VCs that are no longer valid due to fraud, state change (e.g., disaffiliation), or expiration.

- If fraud is detected or the user wishes to disaffiliate, the VC is revoked using a **StatusList2021** mechanism, a W3C standard.
- **Process:** The backend updates its internal status to reflect the revocation, and the Issuer Agent publishes the revocation on the Cardano blockchain. Verifiers querying that VC will find it marked as revoked.
  - Each revocable VC includes a credentialStatus field pointing to a public **Status List Credential**, which contains an encoded bitstring. Each position in the bitstring represents the status of a specific VC (1 for revoked, 0 for valid).

- **Verification process (Verifier side):** The verifier resolves the StatusListCredential, verifies its authenticity and integrity (signature), decodes the bitstring, and checks the bit corresponding to the VC.
- **Revocation process (Issuer side):** Only the issuer can revoke a credential. The issuer agent updates the corresponding bit in the StatusListCredential, and the update is passively propagated when verifiers query the updated resource.

**Table: VC Lifecycle and Implementation Methods (APIs/SDKs)**

| VC Operation | Identus/ Application Component | Relevant API/SDK | Technical Description | Standards/Protocols |
|---|---|---|---|---|
| **Issuance** | Issuer Agent (Cloud Agent), Backend API | Cloud Agent REST API, Identus SDKs (TypeScript Agent.issueVC()), Mercury for DIDComm | The Issuer Agent generates and cryptographically signs a VC with the Holder's claims, sending it via DIDComm. | W3C Verifiable Credentials, DIDComm V2 |
| **Storage (Holder)** | Holder Agent (Mobile App) | flutter_secure_storage (Flutter), Pluto module (TS SDK) | VC is securely stored in the Holder's wallet on the user's device. | OS secure storage |
| **Presentation (Holder)** | Holder Agent (Mobile App) | Identus SDKs (TypeScript Agent.presentVC()), Mercury for DIDComm | Holder creates a Verifiable Presentation (VP) from one or more VCs, signs it with their DID key, and presents it to a Verifier. | W3C Verifiable Presentations, DIDComm V2 |
| **Verification** | Verifier Agent (Cloud Agent), Backend API | Cloud Agent REST API, Identus SDKs (TypeScript Agent.verifyVC()) | Verifier validates VC/VP signature, resolves issuer DID in VDR, and checks revocation status. | W3C Verifiable Credentials, W3C DIDs |

| | | | | |
|---|---|---|---|---|
| **Revocation** | Issuer Agent (Cloud Agent), Backend API | Cloud Agent REST API (revocation management) | Issuer publishes revocation status for a specific VC in VDR, invalidating it using StatusList2021. | StatusList2021 (W3C) |

### 3.4 Specific Application Component Integration

Effective integration of the different components of the "Digital Voter ID" system is essential for its functionality and performance.

#### 3.4.1 Flutter Mobile Application: Use of Identus SDKs, Secure Storage, Biometric Authentication

The Flutter mobile application is the primary interface for the citizen (Holder).

- **Interaction with Identus:**
  For DID creation, wallet management, and VC issuance/presentation, the Flutter application will interact with the Identus Agent using the **Kotlin Multiplatform (KMP) SDK** for Android and the **Swift SDK** for iOS provided by Hyperledger Identus.
- **Secure Storage:**
  flutter_secure_storage will be implemented to protect the user's DID private keys and Verifiable Credentials on the device.
- **Biometric Authentication:**
  Integration of local_auth or biometric_storage will allow users to authenticate via fingerprint or facial recognition before accessing sensitive data or performing cryptographic operations with their DIDs/VCs.

#### 3.4.2 React Web Application: Use of Identus SDKs (TypeScript), State Management, Frontend Security Best Practices (XSS, CSRF)

The React web application will serve as the interface for administrators and issuing/verifying organizations.

- **Interaction with Identus:** The React web application will primarily interact with the Identus Cloud Agent via the Backend API.
  For direct frontend functionalities, the Identus TypeScript SDK (@hyperledger/identus-sdk) can be used.
- **State Management:** Recommended libraries include **Redux Toolkit**, **MobX**, **Recoil**, or **Zustand** for global state management.

- **Frontend Security:**
    - **XSS Protection:** While React escapes HTML content by default, manual sanitization using sanitize-html is necessary when rendering untrusted HTML.
    - **CSRF Protection:** Implement CSRF tokens for backend requests to prevent cross-site request forgery.

### 3.4.3 Backend API (Node.js/Python/Java): Framework Selection (NestJS/FastAPI/Spring Boot), Interaction with Identus Agent API

The Backend API is the central orchestrator of the "Digital Voter ID" system.

- **Framework Options:**
    - **NestJS (TypeScript)** and **FastAPI (Python)** are suitable for high-performance API-centric microservices.
- **Interaction with Identus:**
  For Node.js backends (TypeScript/JavaScript), the @hyperledger/identus-cloud-agent-client package is recommended.
  For Python and Java, interaction will be performed via direct HTTP calls to the REST API (as no official SDKs exist in these languages based on current research).

### 3.4.4 Transactional PostgreSQL Database: Schema Design for Users, Affiliations, DIDs, and VCs

The PostgreSQL database will store the traditional business logic data of the "Digital Voter ID" system — not the DIDs or VCs themselves, but their metadata and relationships with users and affiliations.

- **Schema Design Considerations:**
  Tables will include:
    - Users
    - DIDs
    - Political_Organizations
    - Affiliations
    - Optionally, Credential_Schemas

### 3.4.5 Keycloak Integration: Role-Based Authentication and Authorization, Custom SPIs for DID Authentication

Keycloak is critical for user and organization authentication and authorization.

- **Traditional Authentication and Role-Based Authorization:** Keycloak will manage user authentication and role-based access control for authorization.
- **Extension with Custom SPIs for DID Authentication:** A **custom AuthenticatorFactory** and **Authenticator SPI** can be developed in Java to allow user authentication by presenting a **Verifiable Presentation (VP)** signed with their DID instead of a traditional username and password.

**Proposed Flow for DID Authentication with Keycloak:**

1. **Login and DID Challenge:** User initiates login and receives a DID authentication challenge.
2. **VP Generation:** The user's SSI wallet generates a VP.
3. **VP Submission:** The VP is sent to Keycloak.
4. **VP Verification:** The custom SPI verifies the VP, including signature validation, presenter DID resolution, and revocation status check.
5. **JWT Issuance:** Upon success, Keycloak issues a standard JWT token for application access.

**4. Development Considerations and Best Practices**

**4.1 Detailed Configuration of the Identus Agent (Environment Variables, Wallet, DIDComm)**

Proper configuration of the Identus Agent is essential for its operation.

- **Environment Variables:** Configure NODE_LEDGER for Cardano VDR and DOCKER_HOST for communication between Docker services.
- **Cardano Wallet:** Assign the Cardano wallet ID and passphrase to the Identus Agent environment variables, along with a payment address.
- **DIDComm:** Configure the DIDComm protocol for secure communication, including mediators if agents are not always online.
- **Development Mode:** Enable during development and testing phases, but strictly disable in production due to security implications.

**4.2 Interaction with the Identus Agent API (Swagger UI and SDK Usage Examples)**

Programmatic interaction with the Identus Agent is fundamental for both backend and client applications.

- **Swagger UI:** An essential tool for exploring endpoints, understanding data models, and testing API interactions directly — especially valuable given the limited availability of detailed external documentation.
- **SDK Usage:** Identus SDKs (TypeScript, Kotlin, Swift) are the preferred way to interact programmatically with the Cloud Agent, simplifying DID/VC management and DIDComm communications.

**4.3 Scalability Strategies and Performance Optimization**

Scalability and performance are critical considerations for a national identity system.

- **Microservices:** Adopting a microservices architecture allows independent scaling of each service.
- **Stateless Services:** Designing backend services to be stateless facilitates horizontal scaling.
- **Databases:** Implement database scaling strategies such as sharding, read replicas, or NoSQL databases.
- **Asynchronous Processing & Message Queues (optional for PoC):** Use message queues (Kafka, RabbitMQ) for tasks not requiring immediate responses (e.g., DID publication).
- **Load Balancers & API Gateways (optional for PoC):** Implement load balancers to distribute traffic across multiple backend and Identus Agent instances.
- **Caching (optional for PoC):** Implement caching strategies for frequently accessed data.

**4.4 Private Key Security and Seed Phrase Management**

Private key security and seed phrase management are critical for the integrity and trust of an SSI system.

- **Secure MASTER Key Storage:**
  The DID's MASTER key is essential for signing operations and must be protected with the highest level of security.
  In production, use a Vault Server or Hardware Security Module (HSM).
- **Seed Phrase Management:**
  For Holder Agents (mobile apps), managing seed phrases is crucial. Losing a seed phrase may mean irreversible loss of access to the digital identity.

Implement robust identity recovery mechanisms (e.g., social recovery, multiparty computation) and thoroughly educate users on secure seed storage.

- **Key Rotation:** Implement mechanisms for periodic DID key rotation as allowed by the PRISM DID method.
- **Continuous Monitoring:** Private key security, seed management, key rotation, and system/blockchain log monitoring require active, ongoing operations.

**5. Critical System Analysis**

**5.1 Detailed Advantages of SSI Integration with Identus and Cardano**

Integrating Hyperledger Identus with Cardano for the "Digital Voter ID" system offers several significant advantages:

- **Citizen Sovereignty:** Complete control over their identity and personal data, eliminating reliance on centralized entities.
- **Transparent and Immutable Audit:** All critical operations are immutably recorded on the Cardano blockchain, ensuring traceability and public auditability.
- **Advanced Cryptographic Security:** Utilization of cutting-edge cryptographic technologies, including Ed25519 curves, DIDComm, JWT, and digital signatures.
- **Modular and Multi-Tenant Scalability:** Identus is designed with a modular architecture that supports multiple wallets and tenants, facilitating adoption and scaling efficiently.
- **Global Interoperability:** The system is based on widely accepted open standards (W3C DID, VC-JWT, DIDComm V2).
- **Cardano Backing:** Cardano’s robustness, security, and decentralized nature as a Verifiable Data Registry provide a solid foundation of trust for the entire system.

**5.2 Disadvantages and Technical Challenges Identified**

Despite its advantages, implementing an SSI system with Identus and Cardano presents technical and operational challenges:

- **Technological Complexity:** Requires specialized knowledge in blockchain, cryptography, Docker containers, and distributed architecture.
- **Blockchain Latency:** Cardano transaction confirmation can take between 3–10 minutes (112 blocks), potentially affecting user experience.
- **Digital Access Barrier:** Citizens must have access to smart devices and stable internet connections, potentially creating a digital divide.
- **Agent Learning Curve:** The interactions and flows between different agent roles (issuer, holder, verifier) are new and require initial training.
- **Dependence on Third-Party Services:** While decentralized, the system may still depend on the availability and performance of certain services (Cardano nodes, Identus Agent APIs).
- **Continuous Oversight Required:** Private key security, seed phrase management, key rotation, and constant monitoring require active operation by specialized personnel.
- **Limited Detailed Documentation:** A recurring observation is the difficulty of finding detailed guides and code examples for key Hyperledger Identus components, increasing the learning curve and implementation time.

### 5.3 Potential Risks and Mitigation Strategies

Below is a detailed table of identified risks and mitigation strategies according to the proposed architecture for the "Digital Voter ID" system.

**Table: Identified Risks and Mitigation Strategies**

| Risk | Description | Potential Impact | Mitigation Strategy |
|---|---|---|---|

| | | | |
|---|---|---|---|
| **Cardano Node Desynchronization** | The local Cardano node or the Identus Agent node loses sync with the main network. | Failure to publish/validate DIDs and VCs, service disruption. | Proactive node sync monitoring. Implement fallback to trusted public nodes or external DID resolution services. Use snapshots to speed recovery. |
| **Seed Phrase Loss (Holder)** | Citizens lose the seed phrase for their SSI wallet. | Irreversible loss of access to their digital identity and credentials. | Implement identity recovery mechanisms (e.g., social recovery, multi-signature custodianship). Provide thorough user education on secure seed handling. |
| **Private Key Compromise (Issuer/Verifier)** | Issuer or Verifier Agent private keys are stolen/compromised. | Fraudulent VC issuance, malicious revocation, identity spoofing. | Store keys in Vault Server or HSM in production. Implement periodic DID key rotation. Monitor key activity and trigger alerts for anomalies. |
| **Attacks on Identus Agent API** | Vulnerabilities in Identus Agent REST API or API Gateway. | Unauthorized DID/VC operations, data manipulation. | Implement strong authentication (API Keys, OAuth 2.0) and fine-grained authorization. Use WAF. Conduct regular security audits and penetration testing. |

| | | | |
|---|---|---|---|
| **Cardano Confirmation Latency** | Long transaction confirmation times affect UX. | User frustration, slower business processes. | Design asynchronous flows with clear visual feedback. Cache frequently read data. Optimize business logic to avoid immediate confirmation dependency. |
| **Technical Documentation Inaccessibility** | Difficulty finding detailed guides and code examples. | Increased development time, higher learning curve, possible errors. | Promote active exploration of Identus Agent Swagger UI. Participate in Hyperledger Identus community (Discord, forums). Contribute to open-source documentation. |
| **DIDComm Mediator Dependency** | Mediator service downtime/unavailability. | Disruption in DIDComm communication between agents (especially mobile). | Deploy Mediator with high availability and redundancy. Monitor Mediator health. Support multiple Mediators or allow direct communication when possible. |
| **Identity Spoofing Attacks (Apps)** | Fraudulent use of identity at the app level (e.g., compromised mobile app). | Unauthorized access to system functions, fraudulent credential use. | Implement strong biometric authentication. Use flutter_secure_storage for sensitive data. Obfuscate code and detect app tampering. |

| | | | |
|---|---|---|---|
| **Backend API Vulnerabilities** | Security flaws in business logic or in Identus/Keycloak interaction. | Data exposure, affiliation manipulation, unauthorized access. | Strict adherence to framework security best practices (input validation, XSS/CSRF protection, secure token management). Code audits and penetration testing. |
| **Interoperability Issues** | Difficulty integrating with other SSI systems or future standard versions. | Adoption limitations, need for redevelopment. | Strict adherence to W3C standards (DID, VC, DIDComm V2). Participate in standards communities. Continuous interoperability testing. |

## 6. Testing and Validation

### 6.1 Testing Strategy

#### 6.1.1 Unit Testing for Critical Components

Unit tests will be carried out for key components such as the credential issuance service, validating business logic and the generation of consent hashes.

#### 6.1.2 Integration Testing for Key Flows

Integration tests will be performed to validate the complete flow of identity registration and credential issuance, including:

1. Citizen registration.
2. Affiliation request.
3. Credential issuance.
4. Credential verification.
5. Credential revocation.
6. Verification of revoked status.

### 6.2 Standards Compliance Validation

#### 6.2.1 W3C DID Compliance

Technical validation for DID compliance will include:

1. DID structure (did:prism:<namespace>:<unique-id>).
2. DID resolution and existence of the DID Document.
3. Presence of authentication methods.
4. Key format and type (e.g., Ed25519VerificationKey2018).

#### 6.2.2 W3C Verifiable Credentials Compliance

Validation for credential compliance will include:

1. Verification of the W3C Verifiable Credentials context.
2. Presence of the VerifiableCredential type.
3. Presence of credentialSubject.
4. Verification of the political schema (PoliticalAffiliationCredential).
5. Structure of credentialSubject and required fields.