02 Inji Verify Offline Verification

Empowering VC Verification in Disconnected Environments

Build a browser-based or mobile-friendly application using the Inji Verify SDK to enable offline Verifiable Credential (VC) verification via QR codes, with local result storage and automatic sync when connectivity is restored.

Complexity Level: High

Overview:
In regions with poor or no internet access, verifying Verifiable Credentials (VCs) online becomes difficult. This project proposes building an offline verification solution using the Inji Verify SDK. The solution allows scanning QR codes, performing offline cryptographic verification, storing results locally, and syncing with a server once connectivity returns. It is particularly useful in field verifications, remote areas, or disaster relief scenarios.

Exact Task:

Design, develop, and test a browser-based / mobile-friendly application that integrates with the Inji Verify SDK to:

- Utilize a webcam or phone camera to scan QR codes containing Verifiable Credentials.
- Perform complete VC verification logic offline using the Inji Verify SDK.
- Store all verification results (success/failure, timestamps, hash, etc.) securely and persistently in local storage.
- Automatically detect internet connectivity and synchronize the stored verification results to a pre-defined mock server when online. Backend service should be able to synchronise data over mobile data or wifi settings appropriately.

Mandatory Tasks:

- **UI/UX Design:** Create a user-friendly interface for scanning, displaying results, and managing logs.
- **QR Code Scanner Integration:** Implement a reliable QR code scanner capable of reading VC QR codes using webcam/phone camera.

- **SDK Integration:** Successfully integrate the Inji Verify SDK for offline VC validation (signature check, schema validation, revocation status if pre-cached).
- **Local Data Storage:** Implement robust local storage for storing verification results persistently.
- **Connectivity Detection:** Develop logic to accurately detect online/offline status.
- **Synchronization Mechanism:** Implement a mechanism to automatically upload stored logs to a mock server once connectivity is restored.
- **Error Handling:** Provide clear feedback to the user for invalid VCs, scanning errors, or sync failures.
- **Basic Security:** Implement basic security practices for the client-side application (e.g., input validation, secure data handling).

Good-to-have Tasks:

- **Revocation Handling (Pre-cached):** Implement a mechanism to pre-cache small, manageable revocation lists (e.g., a whitelist/blacklist of known revoked VCs/issuers) for offline checks.
- **Configurable Sync Endpoint:** Allow configuration of the mock server endpoint.
- **Filtering/Searching Logs:** Enable users to filter or search through locally stored verification logs.
- **Export Logs:** Provide an option to export local logs (e.g., as CSV/JSON).
- **Multi-VC Scanning:** Allow scanning multiple VCs in sequence.
- **Dockerization:** Dockertization of deployable components

Bonus Tasks:

- **DID Resolution (Limited Offline):** Explore caching strategies for DIDs to enable limited offline DID resolution (e.g., for frequently used issuers).
- **Proof-of-Concept for Native Mobile:** Build a basic native Android/iOS version using the Inji Verify SDK if available for those platforms.

- **Performance Benchmarking:** Measure the performance of offline verification and synchronization processes.

- **Offline Credential Presentation:** If possible, explore basic offline credential presentation capabilities (e.g., generating a QR code for a specific VC for another offline verifier).

Recommended Tech Stack:

- **Frontend (Browser-based PWA or Mobile app):**
  - **Framework:** ReactJS.
  - **Offline Capabilities:** Service Workers (for caching assets and API responses), IndexedDB (for robust local data storage).
  - **QR Scanner:** zxing-wasm library / native mobile camera access

- **Offline VC Verification:** Inji Verify SDK (TypeScript for web, Java/Kotlin for Android, Swift for iOS).

- **Data Synchronization:** RESTful API for syncing data to a backend mock server. Background Sync API (for Service Workers) for progressive web apps, or native background tasks for mobile.

- **Backend (Mock Server for Sync):** Node.js (Express.js), Python (Flask/Django) - for receiving and storing verification logs.

- **Database (Mock Server):** SQLite (for local testing), PostgreSQL (for robust server-side storage).

Deliverables:

- **Working Application:** A fully functional, deployed (or deployable) browser-based PWA or mobile-friendly application.

- **Source Code:** Clean, well-commented, and maintainable source code for the application and mock server.

- **Technical Design Document:** Detailing architecture, data flow, security considerations, and major components.

- **User Guide:** Simple instructions on how to use the offline verifier.

- **Test Cases & Results (optional):** Comprehensive test cases covering offline/online scenarios, valid/invalid VCs, and sync functionality.

- **Demo Video:** A short video showcasing the application's key features.

Resources

1. INJI Official Documentation: https://docs.inji.io/

2. Inji Verify SDK Documentation:
    - Official Docs : https://docs.inji.io/inji-verify/technical-overview/integration-guides/openid4vp-vp-verification-integration-guide
    - GitHub: https://github.com/mosip/inji-verify

3. OpenID4VCI (OpenID for Verifiable Credential Issuance) Specification: https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0-12.html

4. W3C Verifiable Credentials Data Model: https://www.w3.org/TR/vc-data-model/

5. Service Workers MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

6. IndexedDB MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

7. zxing-wasm : https://github.com/Sec-ant/zxing-wasm