Reliable QR Attendance Tracker Final Project Report

Authors: Harshdeep Singh (2022IMT-048), Harsh Jaiswal (2022IMT-047), Gouravi Singh (2022IMT-046)

1. System Architecture Overview

Key Architectural Components:

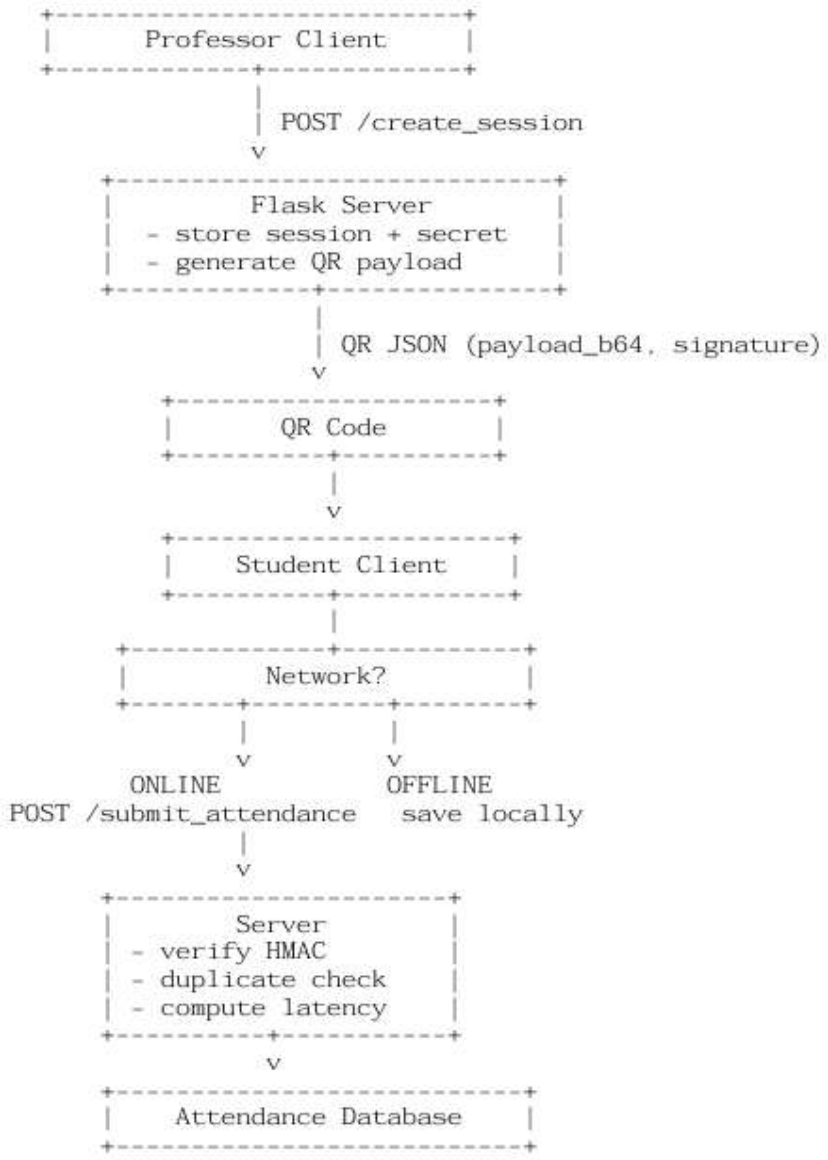
- **Flask Server (Backend):** This is the central hub, responsible for session management, QR payload generation, signature computation, and attendance data storage. It acts as the single source of truth for all attendance records.
    - **Core Functions:** Handling \ (POST /create_session (requests, \ (POST /submit_attendance (requests, and serving attendance viewing pages (e.g.,/attendance/session_id\).
    - **Security:** It generates a unique \ (server_secret (for each session and uses **HMAC (Hash-based Message Authentication Code)** to sign the QR payload, ensuring that only valid, untampered session data is used for submission.
- **Professor Client:** Used to initiate a new class session by sending a request to the server, specifying the course and duration. Its output provides the signed QR payload (JSON) which can be displayed to students.
- **Student Client:** This is the primary user interface for students. It takes the QR payload, extracts the session details, and attempts to submit the student's attendance to the server. Its key distinguishing feature is its **offline queue capability**.

System Flow: Session Creation and Attendance Submission

The process follows a clear sequence, as detailed in the diagram:

1. The Professor Client sends a \(POST (request to the Flask Server (/create_session).
2. The Flask Server generates a unique \(session_id (, signs the session payload using HMAC, and returns the QR JSON (containing \ (payload_b64 (and signature.
3. The Student Client receives the QR JSON (e.g., by scanning the QR code).
4. The Student Client builds a JSON submission including their \(student_id (and the decoded QR payload.
5. If **online**, the client sends the submission via POST (/submit_attendance).

6. If **offline**, the client saves the submission to a local **offline queue** for later synchronization.



2. Implementation Steps and User Workflow

The following steps outline the setup and execution of the system for a typical class attendance scenario:

Step 1: Backend Server Startup

The server environment is initialized using standard Python dependency management:

- Install necessary libraries: pip install -r requirements.txt
- Start the Flask application: python server.py
- The server is confirmed to be running and listening on port **5000** (http://localhost:5000).

Step 2: Creating a Session (Professor Role)

A session is initiated using a \ (curl) command, simulating the Professor Client sending a \ (POST) request to the server:

- **Request:** \ (POST http://localhost:5000/create_session)
- **Body:** ("course":"Networks","duration_minutes":20)
- **Server Action:** The server processes this, generates a \ (session_id) (e.g., \ (69d20aee-9f5e-...)), and creates the digitally signed QR JSON output. This output is then presented to students for attendance marking.

Step 3: Submitting Attendance (Student Role)

The Student Client handles the attendance submission, demonstrating network flexibility:

- **Client Launch:** python student_client.py --student-id 2022IMT-048 --server http://localhost:5000 --network online
- **Action:** The student selects menu option **1) Paste QR payload JSON** and inputs the data received from the professor.
- **Server Validation:** The server verifies the submission by:
  - **HMAC Verification:** Re-calculating the HMAC signature using the stored \ (server_secret) and comparing it to the submitted \ (signature) . This prevents spoofing.
  - **Time Check:** Ensuring the submission is within the \ (duration_minutes) window.
  - **Uniqueness:** Checking for duplicate entries from the same \ (student_id) .
- **Result:** A successful submission yields \ ([+] Submission accepted. latency: X.XX) , confirming the student is marked present.

3. Reliability and Key Feature: Offline Mode

A critical feature that elevates this project beyond simple QR systems is its **reliability in handling network interruptions** via the **Offline Mode and Synchronization Logic**.

Offline Queue Mechanism:

When the Student Client is launched, it can operate in an **offline** or a temporarily disconnected state.

- **Offline Submission:** If the client attempts to submit attendance while unable to reach the Flask Server, it does **not** fail. Instead, the submission JSON is saved locally to an **offline queue** (persistent storage).
- **Data Structure:** Each item in the queue is a complete, ready-to-send attendance submission record.

Auto-Sync Logic:

This feature ensures data eventually reaches the server once connectivity is restored.

- **On Reconnect:** The Student Client periodically or upon network re-establishment attempts to sync the queued items. It sends the saved submissions to the (POST /submit_attendance) endpoint.
- **Server-Side Validation:** Crucially, the server validates these delayed submissions against the original session parameters (HMAC signature and submission window time) to prevent late or fraudulent attendance marking.
- **Queue Management:** Upon successful server validation, the synchronized item is permanently removed from the local offline queue.

This reliable mechanism guarantees that a student can successfully 'mark' attendance during the allocated time, even if they have temporary network issues, thus significantly improving the user experience and the tracker's overall reliability.