1. Overall Software Architecture Overview

Purpose

The architecture manages the end-to-end process of access control using QR and facial recognition. The system will validate the QR code and facial data, then send a signal to open the door.

The key components

- Frontend (Mobile & Web): Handles user interactions like scanning QR codes, facial recognition, and administration.
- Backend: Manages user data, access control logic, QR code generation, and facial recognition verification.
- IoT Layer: Integrates the smartphone and backend to communicate with access points like doors.

2. Frontend Architecture

The frontend is responsible for interactions like scanning QR codes, facial recognition, and communicating with the backend.

Frontend Components

Mobile Application (React Native / Flutter)

- QR Code Scanner: Scans QR codes shown near the door.
- Facial Recognition Module: Uses the mobile camera to capture a live image for facial recognition.
- User Authentication: Handles user login and registration.
- Access Request Interface: Interface for requesting access to rooms or events.
- Communication Module: Sends QR and face data to the backend for validation.
- Access Feedback: Notifies the user of access approval/denial.

Web Dashboard (React / Angular)

- Admin Dashboard: For Admin to approve users, manage access rights, and view logs.
- User Management Interface: Allows admins to add, remove, or modify user information.
- Access Logs Viewer: Displays logs of user access to various rooms.

Flow

1. User scans the QR code with the mobile app.
2. Mobile app initiates face recognition and sends both data to the backend.
3. Backend validates the data and returns the result (access granted or denied).
4. The mobile app displays the result.

3. Backend Architecture

The backend is responsible for managing user data, generating QR codes, validating facial recognition, and controlling access.

Backend Components

Authentication Service

- Handles user registration and authentication using JWT or OAuth2.
- Manages roles (admin, and user).

Access Control Service

- QR Code Generator: Dynamically generates and validates QR codes tied to specific access points.
- Facial Recognition API: Integrates with a third-party API (e.g., AWS Rekognition or Azure Face API) to verify user identity based on facial data.
- Access Policy Engine: Contains logic for who is allowed access to specific rooms or events.
- Visitor Management Module: Allows hosts to approve or deny visitor requests.

Database

- Stores user data, access logs, facial recognition data, and QR codes.
- NoSQL (e.g., MongoDB) or SQL (e.g., PostgreSQL) depending on scalability needs.

API Gateway

- RESTful API (Node.js / Express or Django / Flask): Provides endpoints for the frontend to request QR code validation, facial recognition validation, and access control results.

Event Queue (Optional for large-scale systems):

- Implements asynchronous access validation using an event-based architecture (e.g., RabbitMQ or Kafka) to handle multiple requests simultaneously and improve performance.

IoT Communication Layer:

- WebSockets / MQTT: For real-time communication between the backend and mobile devices.
- Door Control System: Sends the signal to unlock the door after successful validation.

Flow:

1. User data and access policies are stored in the backend.
2. QR codes are dynamically generated based on user requests.
3. Facial recognition is validated against stored data using third-party APIs.
4. If both QR and face data match, the backend sends a signal via the IoT communication layer to unlock the door.

4. IoT Integration

The IoT layer manages the real-time communication between the mobile device and the door access point.

IoT Components

Smartphone as IoT Device

- The mobile app acts as a scanner (for QR and facial recognition) and communicates with the backend over HTTPS or WebSockets.
- The smartphone sends data to the backend and acts as a client device for sending access signals.

IoT Hub / MQTT Broker

- MQTT Broker (e.g., Mosquitto): Handles real-time, lightweight communication between mobile devices and backend.
- IoT Hub (AWS IoT Core / Azure IoT Hub): Manages connections from smartphones and controls interactions with backend services.

Door Lock System (for a future implementation with physical locks)

- Receives a signal from the backend to open doors upon successful validation.
- The system can be extended to include actual IoT-based door locks that communicate with the backend.

Flow:

1. After validation, the mobile device sends a request to unlock the door via the IoT hub.
2. If integrated with physical doors, the hub sends a signal to the door lock to unlock it.
3. The system can also handle fallback methods if the mobile device loses connectivity.

5. Security Considerations

Data Encryption

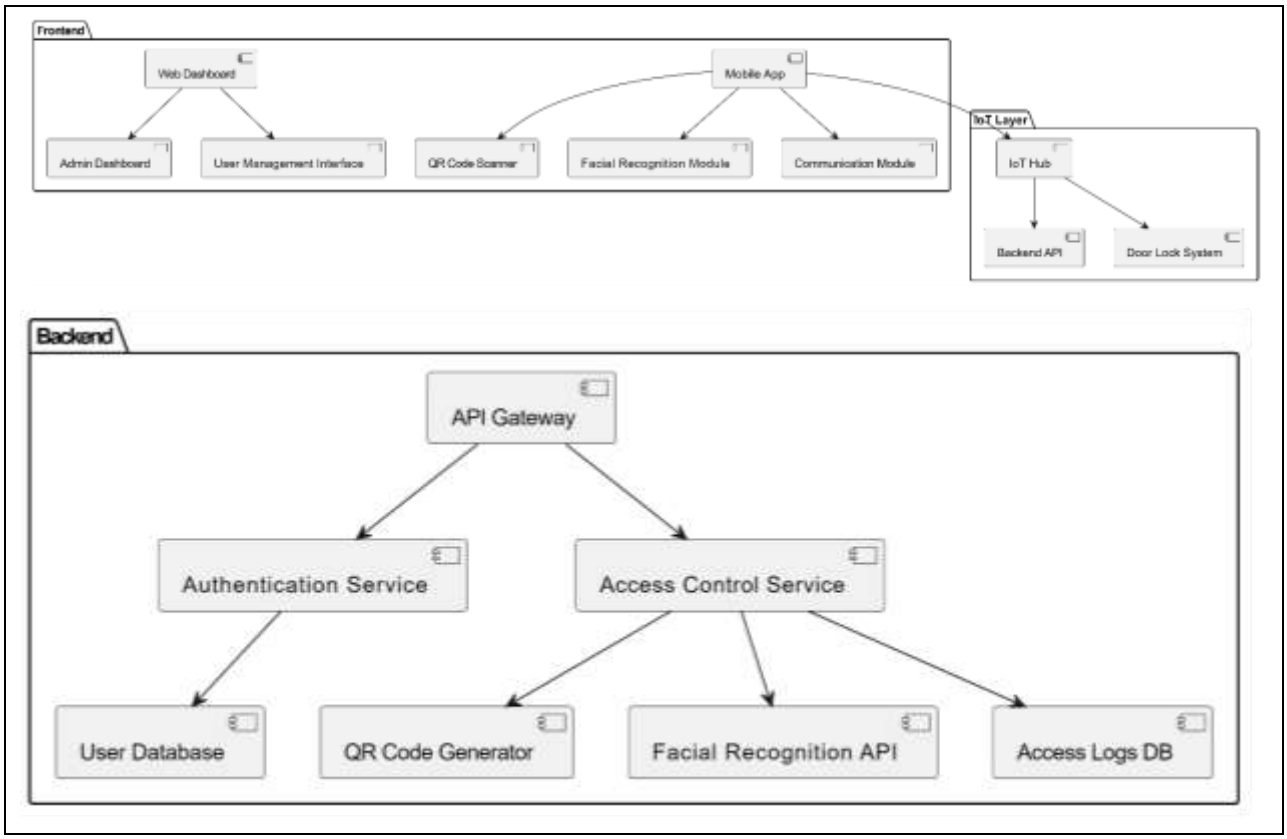
- Use TLS for secure communication between mobile apps, the backend, and IoT devices.
- Encrypt sensitive data (QR code, facial recognition data) both at rest and in transit.

Authentication & Authorization

- Implement OAuth2 or JWT for securing API requests.
- Use role-based access control (RBAC) to manage user roles (admin, visitor, etc.).

Data Privacy

- Ensure compliance with data privacy regulations like GDPR by securely storing and managing personal data such as facial recognition information.


Frontend
Web Dashboard
Mobile App
Admin Dashboard
User Management Interface
QR Code Scanner
Facial Recognition Module
Communication Module
IoT Layer
IoT Hub
Backend API
Door Lock System
Backend
API Gateway
Authentication Service
Access Control Service
User Database
QR Code Generator
Facial Recognition API
Access Logs DB