Microservice Architecture for QR and Face Recognition-based Room Access System

1. Introduction

We present a microservice architecture for the development of a QR and Face Recognition-based Room Access System. This system will handle user authentication, access control, and logging, allowing secure access to rooms through the combination of QR codes and facial recognition.

The microservices approach will ensure scalability, flexibility, and fault tolerance, making the system robust and maintainable as it grows.

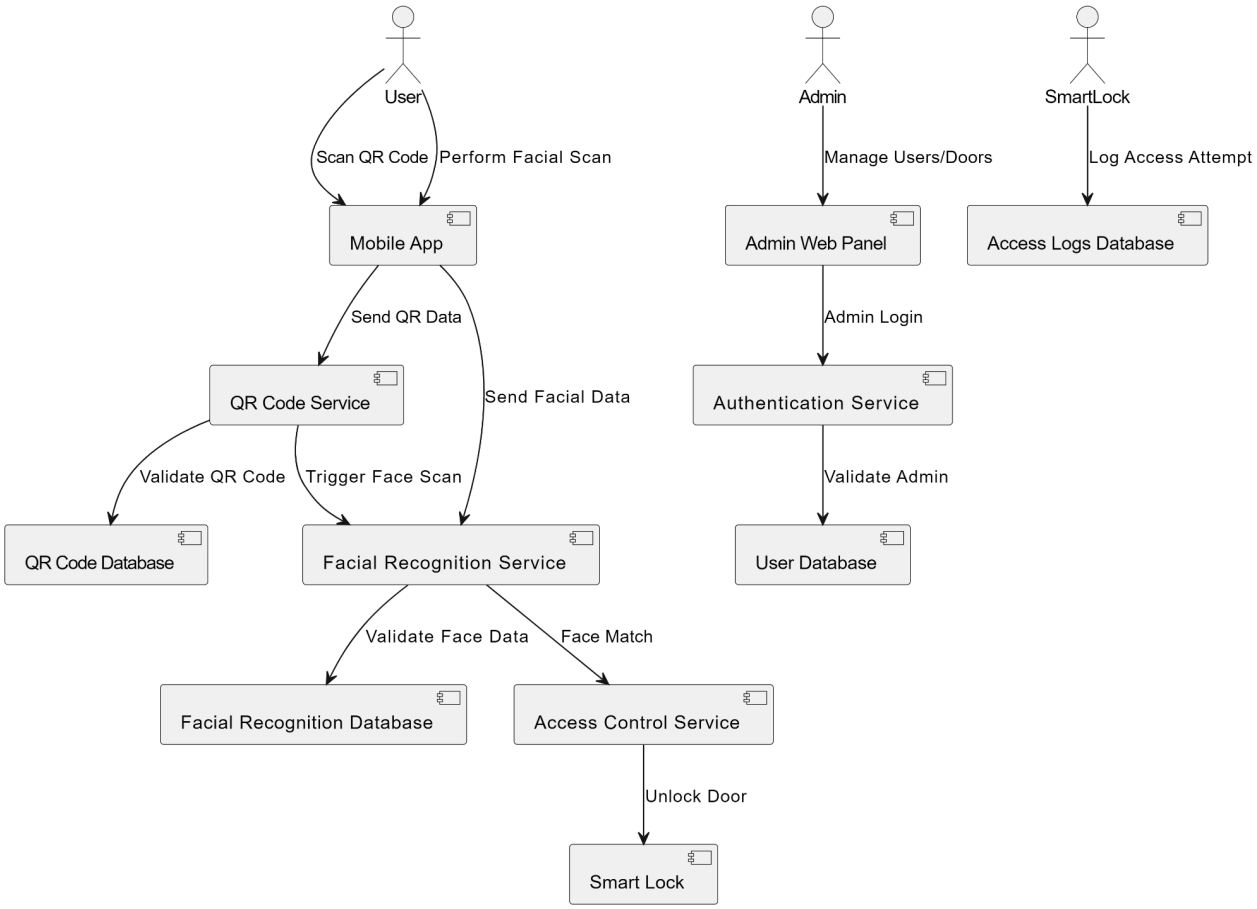
2. Service Breakdown

Services	Responsibilities	Technologies
QR Code Service (Generates and validates QR codes for room access)	• Generate dynamic QR codes for each room • Validate scanned QR codes against the database.. • Trigger the Facial Recognition Service if the QR code is valid.	• Backend: Node.js • Database: MongoDB (to store QR code data) • Communication: REST API
Facial Recognition Service (Handles the facial recognition process to authenticate the user)	• Capture and process facial biometric data. • Compare the captured face with stored data. • Send validation results to the Access Control Service.	• Libraries/APIs: • Database: PostgreSQL (to store facial data securely). • Communication: REST API for receiving user requests.

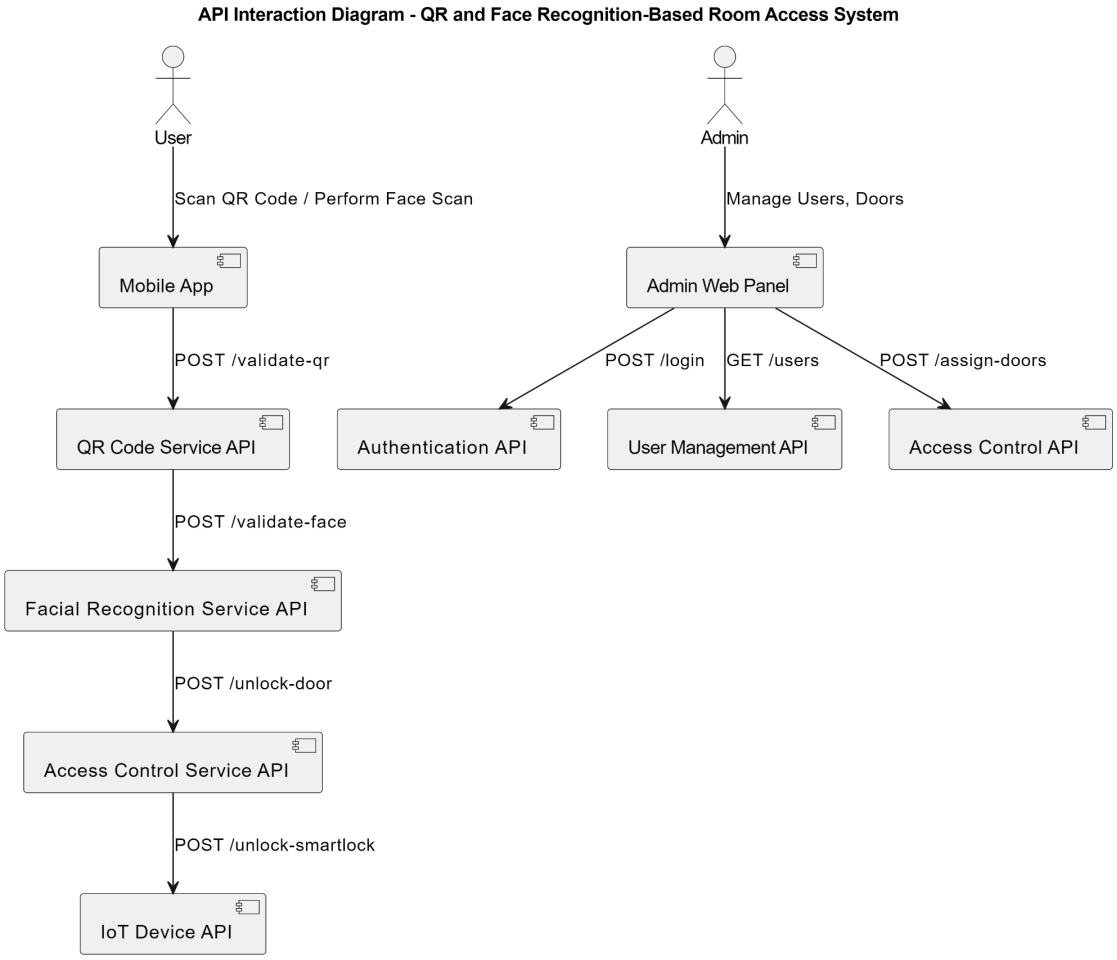
Authentication Service (Manages user authentication and session management)	• Handle user login and registration (e.g., via mobile app). • Issue secure authentication tokens (e.g., JWT tokens) to validate user sessions.	• Backend: Node.js • Database: MongoDB for storing user credentials.
Access Control Service (Integrates both QR code validation and facial recognition to control room access. Responsibilities)	• Receive validated data from both QR Code Service and Facial Recognition Service. • Trigger smart locks to grant or deny access. • Log the access events (successful or failed attempts).	• Backend: Node.js • IoT Integration • Database: PostgreSQL for storing access logs.
Logging and Monitoring Service (Records and monitors all access attempts, including failed ones)	• Log every user access event (successful or failed). • Provide a monitoring dashboard for system administrators. • Trigger alerts for unauthorized access attempts.	• Backend: Node.js • Database: • Visualization:

3. Architecture Diagram

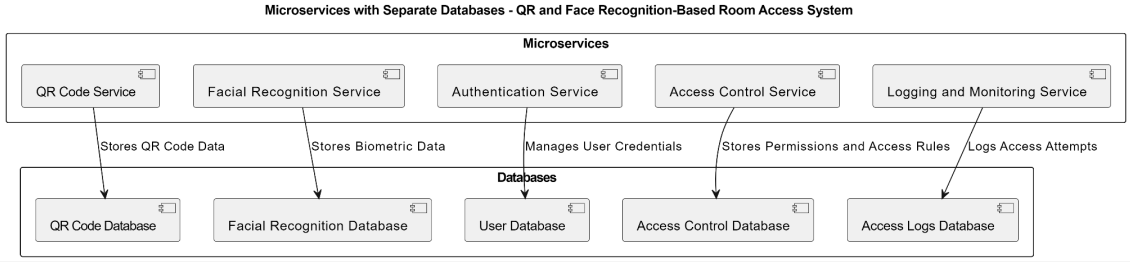
3.1. Microservices (QR Code Service, Facial Recognition Service, Authentication Service, Access Control Service, Logging Service).



3.2. APIs and how each service communicates with the others.



3.3. Databases: Show how each service might have its own database, ensuring modularity.



4. Development Plan

4.1 Technologies and Tools

1. Frontend:
   - Mobile App: Flutter or React Native for cross-platform development.
   - Admin Web: React for a responsive interface.

2. Backend:
   - Web Application: Node.js
   - API Communication: RESTful APIs between services.
   - IoT Integration

3. Databases:
   - MongoDB: For storing user and QR code data.
   - PostgreSQL: For access logs and facial data.

4.2 Development Stages

Stage 1

Frontend Development

- Develop a mobile app for scanning QR codes and capturing facial data.
- Create the admin web panel for user and access management.

Stage 2

Microservices Setup

- Build and containerize each microservice (QR Code, Facial Recognition, Authentication, Access Control, Logging) using Docker.

Stage 3

API Integration

- Connect the mobile app and web panel to backend services via REST APIs.

Stage 4

IoT Integration

- Set up the smart locks and integrate them with the Access Control Service.

Stage 5

Security and Testing

- Implement encryption for sensitive data such as facial biometric data.
- Test the system thoroughly for user access and potential security breaches.

Stage 6

Deployment

- Deploy all services using Kubernetes for orchestration, ensuring that each service can scale independently.