CSC3170 DBMS – Visitor Management System (VMS) Report

1. **Overall Project Description**

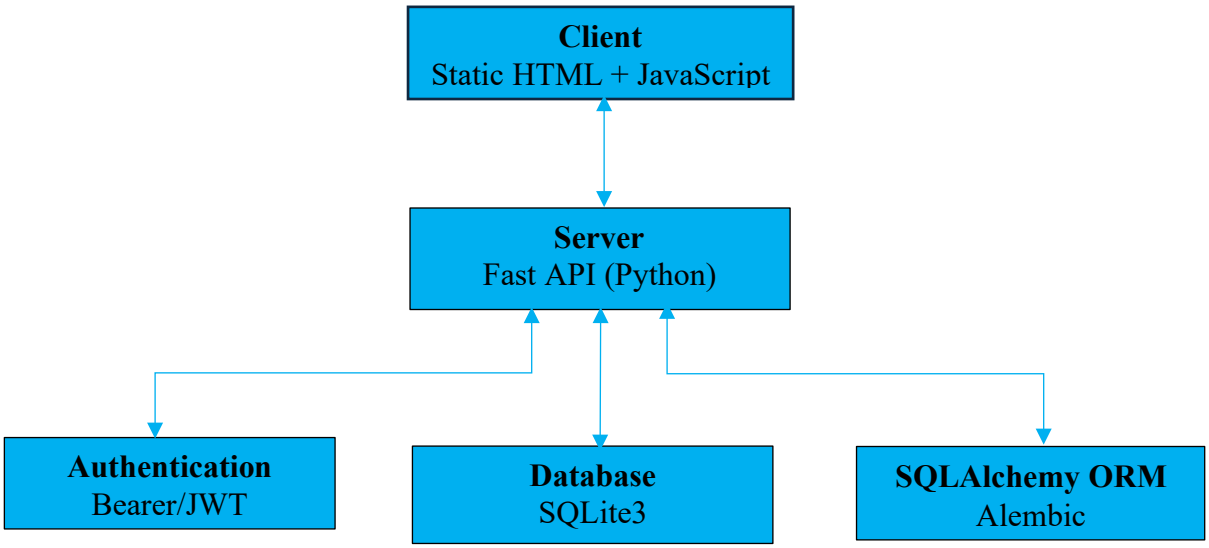
The Visitor Management System (VMS) is designed to digitalize and streamline the visitor registration and approval process within the CUHK(SZ) campus. As the number of visitors, including parents, alumni, corporate partners, and temporary guests—continues to grow, traditional paper-based registration methods have become inefficient and prone to errors.

The proposed system provides an online platform that integrates visitor registration, appointment scheduling, approval management, and real-time notifications. Through this system, visitors can easily create accounts, submit visit requests, modify reservations, and receive approval results without physical paperwork.

Administrators, on the other hand, can review and approve visitor requests, monitor daily visit statistics, and generate reports to support campus security and operational decisions.

The system follows a client-server architecture, consisting of a front-end user interface and a back-end server implemented with a relational database (SQLite). The database manages entities such as visitors, administrators, locations, reservations, and notifications, ensuring data consistency and reliability. Both the visitor and administrator modules communicate through HTTP APIs to perform data operations such as login authentication, CRUD actions, and data queries.

System Architecture



Technology Stack

Client: Static HTML / Java Script / CSS / Fetch API

Server: Python3 / Fast API / Alembic / SQLite 3

DB: DBeaver / SQLite 3

2. **Requirement Analysis**

Project Objectives: The objective of the VMS is to build a secure, efficient, and user-friendly platform that manages campus visitor activities digitally. The system replaces traditional paper-based processes with an online reservation and approval workflow. It allows visitors to create and manage reservations easily, while providing administrators with tools to approve requests, monitor access, and generate statistical reports.

By integrating authentication, reservation management, and data analytics, the system ensures that every visitor entry is authorized and traceable, thereby improving the campus's security, efficiency, and information transparency.

User Role && Requirements

Visitors

Roles: Visitors are external users (such as parents, guests, partners, or alumni) who require campus access. They can register an account, submit visit reservations, and receive approval results from the system.

Functions: Register && Login / Edit Personal Profile / Create Reservation / Edit || Cancel Pending Reservation / View My Reservations / Receive Notifications / Mark Notifications as Read / Reset Password

Admin

Roles: Administrators are authorized staff members responsible for managing visitor requests, reviewing reservations, and ensuring compliance with security policies. They operate through a backend interface that supports approval workflows and statistical monitoring.

Functions: Login / View All Reservations / Approve || Reject Reservations / Filter Reservations by Date or Location / Generate Daily Report / View Visitor History / Modify Profile

Root Admin (Super Admin)

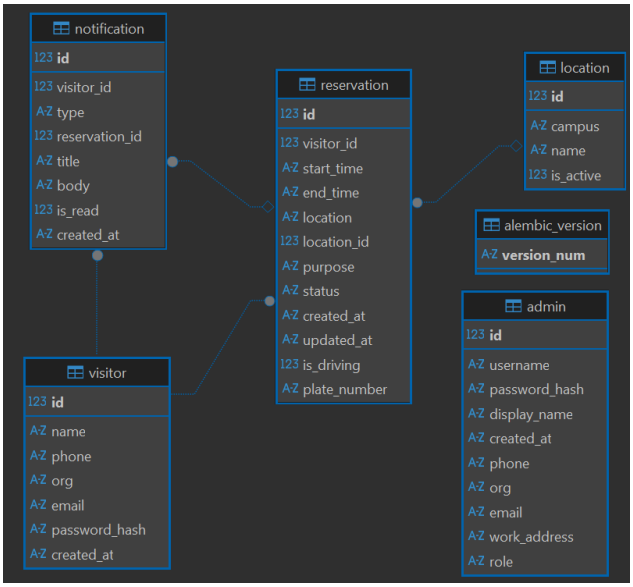
Roles: The Root Admin (or Super Admin) has the highest level of system authority. They are responsible for managing administrator accounts, maintaining location data.

Functions: Create Admin / Manage Locations' Status (Open || Close) / Create || Delete

|| Locations

3. **DB && SQL Design**

DB Schema



Visitor Table:

Stores visitor user info, including name, contact info, organization, and password hash. Supports authentication for external users and links to their reservations and notifications.

Admin Table:

Stores administrator info, including name, display name, contact info, organization, working address role (admin || super admin) and password hash.

Location Table:

Maintains campus location data, including campus name, location name and activation status. Administrators can manage available visiting sites, and reservations reference these locations via foreign keys.

Reservation Table:

Servers as the core transactional table for visiting booking records. Each reservation is linked to a visitor and a location, containing visiting purpose, time range, approval status, and driving info. It supports CRUD operations and is used by administrators for approval workflow and daily reporting.

Notification Table:

Handles system-generated messages such as approval results when visitor login their account. Each one links to a visitor and optionally to a reservation, with read/unread status tracking.

Alembic Version Table:

Automatically managed by Alembic, used to record DB migration version numbers. This ensures schema changes are version-controlled and synchronized across env.

Key SQL Queries

One of the most significant queries is to get the top location of a given day by number of reservations, along with how many of those were approved.

```sql
WITH day AS (
  SELECT :d AS d_start, DATE(:d, '+1 day') AS d_end
)
SELECT r.location_id,
       l.name AS location_name,
       COUNT(r.id) AS reservation_count,
       SUM(CASE WHEN r.status='approved' THEN 1 ELSE 0 END) AS approved_count
FROM reservation r
JOIN location l ON l.id = r.location_id
JOIN day      d ON 1=1
WHERE r.start_time >= d.d_start AND r.start_time < d.d_end
GROUP BY r.location_id, l.name
ORDER BY reservation_count DESC, approved_count DESC
LIMIT 1;

```

Here, we select all location and reservation data for a given day, join them to count total reservations and approved ones for each location, and then order the results by reservation volume to find the most popular location of the day.

4. **Front && Back-end Design && Implementation**

VMS adopts a front–back-end separated architecture, ensuring scalability, modularity, and clarity in data flow. The backend, built with *FastAPI (Python)*, exposes RESTful APIs to handle authentication, reservation management, notifications, and administrative operations. The frontend, implemented using static *HTML, CSS, and JavaScript*, provides distinct web portals for Visitors, Admins, and Super Admins. All data exchange between frontend and backend is conducted through asynchronous REST API calls using the fetch() interface with JSON formatting.

```
├── env.yml
├── Readme.md
├── requirements.txt
├── vms.db
├── .vscode
├── client
│   ├── admin.html
│   ├── admin.js
│   ├── api.js
│   ├── index.html
│   ├── index.js
│   ├── root.html
│   ├── root.js
│   ├── style.css
│   ├── visitor.html
│   └── visitor.js
├── server
│   ├── alembic.ini
│   ├── __pycache__
│   ├── alembic
│   └── app
│       ├── auth.py
│       ├── database.py
│       ├── deps.py
│       ├── locations.py
│       ├── main.py
│       ├── models.py
│       ├── notifications.py
│       ├── reservations.py
│       ├── schemas.py
│       ├── settings.py
│       └── __pycache__
└── sql
    ├── queries.sql
    └── schema.sql
```

Backend Function Implementation Table

Module	Functionality
auth.py	Handles login, registration, JWT authentication, and password management for visitors and admins.
reservations.py	Handles login, registration, JWT authentication, and password management for visitors and admins.
locations.py	Manages location data (campus, name, status); available to both visitors and super admins.
notification.py	Manages location data (campus, name, status); available to both visitors and super admins.
database.py models.py	Manages notifications: creation, retrieval, unread counts, and marking as read.
schemas.py	Define ORM models for all tables and database connection/session setup.
deps.py	Defines Pydantic schemas for data validation and serialization.

Frontend Function Implementation Table

Module	Functionality
index.html index.js	Provides login and registration interface for visitors and admins; routes users to their respective dashboards.
visitor.html visitor.js	Provides login and registration interface for visitors and admins; routes users to their respective dashboards.
admin.html admin.js	Allows visitors to create, edit, cancel, and view reservations, as well as read notifications and manage profiles.
root.html root.js	Enables super admins to manage locations and administrator accounts.
api.js	Wraps fetch() requests to handle headers, JWT tokens, and error responses.

5. Summary && Harvest

Through the development of the VMS, I successfully completed a full-stack project covering DB design, backend API development, and frontend implementation. The system integrates user authentication, reservation management, notification handling, and administrative control into a unified and modular structure. The use of FastAPI and SQLite demonstrated the efficiency of modern lightweight frameworks, while Alembic ensured smooth DB migration and version management. On the frontend, HTML and JavaScript were sufficient to deliver an intuitive and user-friendly experience, fully separated from backend logic via RESTful APIs. From this project, I not only deepened my understanding of DBMS & web architecture, but also practiced how to design data models and manage schema evolution. Overall, the project provided valuable experience bridging theory and real-world implementation——reinforcing the concepts of modularity, maintainability, and user-centered design.