Report

> Name: Wenjie Feng ID: 122090112
> **Statement**: Besides test case generation, the implementation of webpage(.vue) also relies heavily on school deployed Qwen3.

1. Overall Project Description

As for this project, I choose scenario-1 to design a dormitory management system. It contains 2 roles: student and administrator. A student can always register a new student account. And after registration, or been input into the database from the start, a student can modify his own personal information, view individual dormitory assignment details, submit dormitory adjustment requests，view dormitory fee overview，and submit or modify dormitory maintenance requests (I add features to allow also modifications on dormitory adjustment request and logout). But the newly registered student won't have an assigned dormitory, so it was set to be default as `NULL` .

As for administrator, he can not only modify his own personal information like a student. He can also modify all students accounts. He can also approve or reject dormitory adjustment request. And he can view all room status, like empty bed number, room availability, dormitory payment status.He can also process all maintenance requests and update their status.

This project adopts a frontend-backend structure. While the backend processes all requests from the backend and interacts with the database, the frontend provides user with a Web UI and send all requests to backend via a self-defined API. As for frontend architecture, I apply `Vite + Vue` . As for frontend backend interaction, I apply `FlaskAPI` . As for the database interaction, I apply `sqlite3` .

2. Requirement analysis

1. Role choice: There should be a welcome page at the start, allowing the user to choose whether to log in as a student or admin or register as a new student.
2. Student personal information modification: To allow personal information modification, the frontend must also fetch the personal information of that logged-in student. To simplify authentication, I just use the `session['student_id']` to store the student_id and `session['student_logged_in']` to avoid illegal bypassing. And all the updated filed will be passed in a `form` but not a `json` .
3. Student view dormitory assignment details: first extract the corresponding `dormitory_no` of that `student_id` . Then, find all other `student_id` s with the same `dormitory_no` as roommates (just use `!=` to rule out himself).
4. Student submit dormitory adjustment requests: Main difficulty here is to find all available rooms. Then to simplify our job, the `dorm_info` table must contain `total_beds` and `occupied_beds` . So that the availability is simple to find. And student mustn't change to its own room. But seeing the whole available room is more of admin features. Because in real word adjustment, one will first find an available room then do the application. I also add the feature of modifying the pending request to let it more sophisticated. And one can't submit a new request when there is a pending one.
5. Student view dormitory fee overview: rather simple. Just maintain a `dorm_fee` table, and use the simple `SELECT` sql to extract the information.
6. Student submit or modify dormitory maintenance requests: Pretty much like the adjustment request. But this can be in 3 stages: pending, processing, finished. While the adjustment request is just pending, approve/reject. And the maintenance request can allow priority tag which is much more smarter.
   Other part is rather the same.
7. Admin login: Similar like student login, to forbid illegal bypassing, use the `session['admin_logged_in']` and `session['admin_id']` to judge in the backend everytime.
8. As for all other admin features, they are all pretty much the same as student. Just need to notice that to allow modifications, the admin must first fetch and show all applications.

3. Database and SQL design

3.1 Database schema

The schema for all tables created are below.The primary key is marked as red and all foreign keys are marked as green.

`dorm_info`：

Field Name	Data Type	Constraints
dormitory_no	INTEGER	PRIMARY KEY AUTOINCREMENT
building_no	TEXT	NOT NULL
floor_no	TEXT	NOT NULL
dormitory_door_no	TEXT	NOT NULL
total_beds	INTEGER	NOT NULL CHECK (total_beds > 0)
occupied_beds	INTEGER	NOT NULL DEFAULT 0 CHECK (occupied_beds >= 0)
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP
updated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP

`student_info`：

Field Name	Data Type	Constraints
student_id	TEXT	PRIMARY KEY
password	TEXT	NOT NULL
name	TEXT	NOT NULL
gender	TEXT	CHECK(gender IN ('男', '女', '其他'))
major	TEXT	NOT NULL
dormitory_no	INTEGER	DEFAULT NULL
email	TEXT	NOT NULL DEFAULT '122090112@link.cuhk.edu.cn'
phone_number	TEXT	NOT NULL
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP
updated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP

`admin_info`：

Field Name	Data Type	Constraints
admin_id	TEXT	PRIMARY KEY
password	TEXT	NOT NULL
name	TEXT	NOT NULL
email	TEXT	NOT NULL DEFAULT '122090112@link.cuhk.edu.cn'
phone_number	TEXT	NOT NULL
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP
updated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP

`dorm_adjustment_requests`：

Field Name	Data Type	Constraints
id	INTEGER	PRIMARY KEY AUTOINCREMENT
student_id	TEXT	NOT NULL
old_dormitory_no	INTEGER	
new_dormitory_no	INTEGER	
reason	TEXT	
status	TEXT	CHECK(status IN ('待审批', '已通过', '已拒绝')) DEFAULT '待审批'
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP
approved_at	DATETIME	DEFAULT NULL

`maintenance_requests`：

Field Name	Data Type	Constraints
id	INTEGER	PRIMARY KEY AUTOINCREMENT
student_id	TEXT	NOT NULL
dormitory_no	INTEGER	NOT NULL
issue	TEXT	NOT NULL
priority	TEXT	CHECK(priority IN ('低', '中', '高')) DEFAULT '中'
status	TEXT	CHECK(status IN ('待处理', '处理中', '已完成')) DEFAULT '待处理'
assigned_to	TEXT	
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP
resolved_at	DATETIME	DEFAULT NULL

`dormitory_fees`：

Field Name	Data Type	Constraints
paymenet_id	INTEGER	PRIMARY KEY AUTOINCREMENT
dormitory_no	INTEGER	NOT NULL
academic_year	TEXT	NOT NULL
semester	TEXT	CHECK(semester IN ('春季','夏季','秋季'))
fee_amount	DECIMAL(10,2)	NOT NULL
paid_amount	DECIMAL(10,2)	DEFAULT 0.00
payment_status	TEXT	CHECK(payment_status IN ('未缴费', '部分缴费', '已缴费')) DEFAULT '未缴费'
payment_date	DATE	NOT NULL
due_date	DATE	NOT NULL
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP
paid_at	TIMESTAMP	DEFAULT NULL

3.2 Database population

Here will just provide you with some example cases and the total number of cases inserted.

Dormitory Information Table

Dormitory No.	Building No.	Floor No.	Dormitory Door No.	Total Beds	Occupied Beds
1	Building A	1F	A101	4	0
2	Building A	1F	A102	4	0
3	Building A	2F	A201	4	0

Total records: 10

Student Information Table

Student ID	Name	Gender	Major	Dormitory No.	Email	Phone Number
S001	Zhang San	Male	Computer Science	5	122090001@link.cuhk.edu.cn	13800138001
S002	Li Si	Female	Electronic Engineering	1	122090002@link.cuhk.edu.cn	13800138002
S003	Wang Wu	Male	Mechanical Engineering	10	122090003@link.cuhk.edu.cn	13800138003

Total records: 10

Admin Information Table

Admin ID	Name	Email	Phone Number
A001	Zhang Admin	admin1@example.com	13800138000
A002	Li Supervisor	admin2@example.com	13900139000

Admin ID	Name	Email	Phone Number
A003	Wang Director	admin3@example.com	13700137000

Total records: 5

Dormitory Adjustment Requests Table

Request ID	Student ID	Old Dorm	New Dorm	Reason	Status
1	S001	5	3	Dormitory noise affects study	Pending
2	S001	5	4	Personal preference	Rejected
3	S002	1	7	Different schedules with roommate	Approved

Total records: 6

Maintenance Requests Table

Request ID	Student ID	Dormitory No.	Issue	Priority	Status	Assigned To
1	S001	5	Air conditioner not cooling	High	Pending	None
2	S001	5	Power outage	High	In Progress	None
3	S006	2	Door lock broken	High	Pending	None

Total records: 7

Dormitory Fees Table

Payment ID	Dormitory No.	Academic Year	Semester	Fee Amount	Paid Amount	Status
1	1	2023-2024	Autumn	1200.00	1200.00	Paid
2	2	2023-2024	Autumn	1200.00	600.00	Partial
3	3	2023-2024	Autumn	1200.00	0.00	Unpaid

Total records: 10

3.3 SQL design

Here we will go through important SQL logic.

1. `get_student_dormitory()` :

```
cursor.execute('''
    SELECT d.building_no, d.floor_no, d.dormitory_door_no, s.name
    FROM student_info s
    JOIN dormitory_info d ON s.dormitory_no = d.dormitory_no
    WHERE s.student_id = ?
''', (student_id,))
own_info = cursor.fetchone()
cursor.execute('''
    SELECT d.building_no, d.floor_no, d.dormitory_door_no, s.name as roommate_name
    FROM student_info s
    JOIN dormitory_info d ON s.dormitory_no = d.dormitory_no
    WHERE s.dormitory_no = (SELECT dormitory_no FROM student_info WHERE student_id = ?)
    AND s.student_id != ?
''', (student_id, student_id))
```

The key here is the `!=`

2. `get_all_dormitory_change_requests_admin()` :

```
cursor.execute("SELECT dormitory_no,(total_beds - occupied_beds) AS available_beds FROM dormitory_info WHERE total_beds > occupied_beds
avaliable_dormitories = cursor.fetchall()
```

3.When admin approves a adjustment request, one need to minus 1 in the original occupied bed counts, and add 1 to the new dormitory.

Other SQL is just ordinary `SELECT` or `UPDATE` .

4. front and back-end design and implementation

Here, I apply `FlaskAPI` and `Vit+Vue` to implement the frontend and backend. As for backend, every function is the backend is assigned with an API, like `@app.route("/api/admin/student/approve_change_requests", methods=["POST"])` and `@app.route("/api/student/dormitory", met` . While only for simple view, we use `GET` , others we use `POST` . And inside each backend function, it will interact with the database via `sqlite3` .

As for frontend, in the `./router/index.js` , how each URL is linked to each webpage in `views/` is listed. And the router can also provide authentication check(which I didn't implement here).In the `views/` , the webpage ie `.vue` files is contained. Each `.vue` file will decide what we can see on the webpage. As for `./api` , they defines how the input for backend function is passed and which API shall they call. As for `./store` , it stores all the return and all the stores will be used as data for `.vue` in `./views` .

5. Gain

In this project, I learn how to implement a frontend backend database. I know in general how a frontend backend arch works. And playing with the Web UI is though painful but also quite fun. The difficult part of this project is not the SQL. SQL is very simple, but the interface betwwen frontend and backend. The `./store/` and `.vue` files took me so many time to learn and modify.