CSC3170 Project Report: Dormitory Management System

Xing Qiliang - 123090669

1 Project Description

1.1 System Overview

This project implements a Dormitory Management System using modern web architecture with strict frontend-backend separation. The backend uses FastAPI (Python) with MySQL database providing RESTful APIs, while the frontend uses Vue.js 3 with Element Plus UI framework, communicating via HTTP/HTTPS with JWT authentication. The system supports two roles (Student and Administrator) with 21+ functional modules covering dormitory management, student registration, room assignment, maintenance requests, bill management, and approval workflows.

2 Requirements Analysis

2.1 Student Role (8 Functions)

(1) Authentication with Argon2 password hashing; (2) Profile management and password change; (3) View dormitory details and roommates; (4) View bills with payment status; (5) Submit or modify room change requests; (6) Submit or modify maintenance requests with priority; (7) Track request status; (8) Dashboard overview.

2.2 Administrator Role (12+ Functions)

(1) CRUD operations on students; (2) Dormitory management with occupancy stats; (3) Room assignment/reassignment; (4) View resident information; (5) Approve/reject room change requests; (6) Process maintenance requests; (7) Bill management; (8) Statistical dashboard with system-wide metrics; (9) Profile management; (10) Batch operations with pagination; (11) Monitor student distribution.

The system implements 21+ distinct operations exceeding the 9-function requirement, with all database operations handled through backend APIs ensuring security and integrity.

3 Database and SQL Design

3.1 Database Schema and ER Model

The system uses 6 normalized tables following Third Normal Form (3NF) principles:

Field	Type	Constraint	Description
Students Table			
student_id	VARCHAR(20)	PK	Student ID
password	VARCHAR(255)	NOT NULL	Hashed pwd
name	VARCHAR(100)	NOT NULL	Full name
gender	ENUM	NOT NULL	Male/Female
nationality	VARCHAR(50)	NOT NULL	Nationality
college	VARCHAR(50)	NOT NULL	College
enrollment_year	INT	NOT NULL	Enroll year
email	VARCHAR(100)	UNIQUE	Email
dorm_id	INT	FK (null)	Dorm ID
created_at, updated_at	DATETIME	AUTO	Timestamps
Rel: Many-to-One with Dormitories			

Field	Type	Constraint	Description
Dormitories Table			
dorm_id	INT	PK, AUTO	Dorm ID
building_no	VARCHAR(10)	NOT NULL	Building
floor_no	INT	NOT NULL	Floor
room_no	VARCHAR(20)	UNIQUE	Room No.
gender_type	ENUM	NOT NULL	Male/Female
total_beds	INT	DEFAULT 4	Capacity
occupied_beds	INT	DEFAULT 0	Occupancy
created_at, updated_at	DATETIME	AUTO	Timestamps
Rel: One-to-Many with Students, Bills			

Field	Type	Constraint	Description
Dorm Change Requests			
request_id	INT	PK, AUTO	Request ID
student_id	VARCHAR(20)	FK	Student
current_dorm_id	INT	FK	Current dorm
target_dorm_id	INT	FK	Target dorm
reason	TEXT		Reason
status	ENUM	pending	Status
admin_id	INT	FK (null)	Admin
admin_comment	TEXT		Comment
created_at, updated_at	DATETIME	AUTO	Timestamps
Rel: M-to-1 with Students, Admins, Dorms			

Field	Type	Constraint	Description
Bills Table			
bill_id	INT	PK, AUTO	Bill ID
dorm_id	INT	FK	Dormitory
bill_type	VARCHAR(50)	NOT NULL	Type
amount	DECIMAL(10,2)	NOT NULL	Amount
billing_month	VARCHAR(7)	NOT NULL	YYYY-MM
due_date	DATE	NOT NULL	Due date
status	ENUM	unpaid	Status
paid_at	DATETIME		Paid time
created_at, updated_at	DATETIME	AUTO	Timestamps
Rel: Many-to-One with Dormitories			

Field	Type	Constraint	Description
Maintenance Requests			
request_id	INT	PK, AUTO	Request ID
student_id	VARCHAR(20)	FK	Student
dorm_id	INT	FK	Dormitory
issue_type	VARCHAR(50)	NOT NULL	Issue type
description	TEXT	NOT NULL	Details
status	ENUM	pending	Status
priority	ENUM	medium	Priority
admin_id	INT	FK (null)	Admin
admin_comment	TEXT		Comment
completed_at	DATETIME		Completed
created_at, updated_at	DATETIME	AUTO	Timestamps
Rel: M-to-1 with Students, Dorms, Admins			

Field	Type	Constraint	Description
Administrators Table			
admin_id	INT	PK, AUTO	Admin ID
username	VARCHAR(50)	UNIQUE	Username
password	VARCHAR(255)	NOT NULL	Hashed pwd
name	VARCHAR(100)	NOT NULL	Full name
email	VARCHAR(100)	UNIQUE	Email
role	ENUM	admin	Role
phone	VARCHAR(20)		Phone
is_active	BOOLEAN	TRUE	Active
last_login	DATETIME		Last login
created_at, updated_at	DATETIME	AUTO	Timestamps
Rel: 1-to-M with DormChange, Maintenance			

3.2 Normalization and Constraints

The schema eliminates redundancy through proper normalization: student information is not duplicated in request tables (using foreign keys instead), dormitory details are centralized, and all relationships use referential integrity constraints. Cascade rules ensure data consistency: deleting a student cascades to their requests (ON DELETE CASCADE), while deleting a dormitory sets student dorm_id to NULL (ON DELETE SET NULL) to preserve historical records.

3.3 Key SQL Examples

3.3.1 Dormitory Search with Filters

```
SELECT d.dorm_id, d.building_no, d.floor_no, d.room_no, d.gender_type,
       d.total_beds, d.occupied_beds, (d.total_beds - d.occupied_beds) AS available_beds
FROM dormitories d
WHERE d.gender_type = ? AND (d.total_beds - d.occupied_beds) > 0
  AND (d.building_no = ? OR ? IS NULL)
ORDER BY d.building_no, d.floor_no, d.room_no LIMIT ? OFFSET ?;
```

3.3.2 Statistical Dashboard Query

```
SELECT COUNT(DISTINCT s.student_id) AS total_students,
       COUNT(DISTINCT d.dorm_id) AS total_dorms,
       SUM(d.total_beds - d.occupied_beds) AS available_beds,
       COUNT(CASE WHEN b.status = 'unpaid' THEN 1 END) AS unpaid_bills,
       COUNT(CASE WHEN dr.status = 'pending' THEN 1 END) AS pending_changes,
       COUNT(CASE WHEN s.gender = 'Male' THEN 1 END) AS male_students
FROM students s
LEFT JOIN dormitories d ON s.dorm_id = d.dorm_id
LEFT JOIN bills b ON d.dorm_id = b.dorm_id
LEFT JOIN dorm_change_requests dr ON s.student_id = dr.student_id;
```

3.3.3 Room Change Approval Transaction

```
START TRANSACTION;
UPDATE dorm_change_requests SET status = 'approved', admin_id = ?,
  admin_comment = ? WHERE request_id = ? AND status = 'pending';
UPDATE students SET dorm_id = (SELECT target_dorm_id FROM dorm_change_requests
  WHERE request_id = ?) WHERE student_id = (SELECT student_id FROM dorm_change_requests
      WHERE request_id = ?);
UPDATE dormitories SET occupied_beds = occupied_beds - 1
  WHERE dorm_id = (SELECT current_dorm_id FROM dorm_change_requests WHERE request_id = ?)
      ;
```

```
UPDATE dormitories SET occupied_beds = occupied_beds + 1
  WHERE dorm_id = (SELECT target_dorm_id FROM dorm_change_requests WHERE request_id = ?);
COMMIT;
```

4 Frontend and Backend Implementation

4.1 Backend Architecture (FastAPI + MySQL)

Technology Stack: FastAPI 0.104.1 (async web framework), SQLAlchemy 2.0.23 (ORM), PyMySQL 1.1.0 (MySQL driver), MySQL 8.0+ (database), python-jose 3.3.0 (JWT), argon2-cffi 23.1.0 (password hashing), Pydantic 2.5.0 (validation), python-dotenv 1.0.0 (configuration).

Three-Layer Architecture:

(1) *Data Layer (models.py):* SQLAlchemy ORM models define database schema with declarative base. Each model includes relationships using `relationship()` and `back_populates`, cascade rules (CASCADE for dependent data, SET NULL for optional references), and automatic timestamp tracking. Enum types ensure data consistency for status fields.

(2) *Business Logic Layer (routers):* Three router modules implement RESTful endpoints:

- `auth.py` (4 endpoints): POST /login (authenticate and issue JWT), POST /register (create new student account), GET /me (retrieve current user info), POST /logout (invalidate token).
- `students.py` (12 endpoints): Profile management (GET/PUT /profile, PUT /password), dormitory operations (GET /dormitory, /roommates, /dormitories with filters), financial (GET /bills with status filter), requests (GET/POST /dorm-change, GET/POST /maintenance, PUT /maintenance/{id}).
- `admin.py` (21 endpoints): Student management (GET/GET/PUT/DELETE /students), dormitory management (GET/GET/PUT /dormitories, GET /dormitories/{id}/students), request processing (GET/PUT /dorm-change, POST /dorm-change/{id}/approve, POST /dorm-change/{id}/reject, GET/PUT /maintenance), billing (GET/POST/PUT/DELETE /bills), statistics (GET /statistics), profile (PUT /profile, PUT /password).

(3) *Validation Layer (schemas.py):* Pydantic models enforce type safety and validation rules. Examples: student_id must be exactly 9 characters, passwords minimum 6 characters with , emails validated with EmailStr type, enrollment_year constrained to realistic range (2020-2030), amounts positive with 2 decimal precision.

Security Implementation:

Password Security: Argon2id algorithm with parameters memory_cost=65536 KB (64MB), time_cost=3 iterations, parallelism=4 threads. Salt automatically generated per password.

Authentication Flow: (1) User submits credentials; (2) Backend queries database and verifies password hash; (3) On success, generates JWT with payload {user_id, user_type, exp}; (4) Client stores token in localStorage; (5) Subsequent requests include token in Authorization header (Bearer scheme); (6) Protected endpoints use dependency injection (`Depends(get_current_student)`) to verify token and extract user identity.

Authorization: Role-based access control ensures students can only access their own data (checked by comparing token user_id with resource owner), while administrators have elevated privileges verified through user_type claim in JWT payload.

4.2 Frontend Architecture (Vue.js 3)

Technology Stack: Vue.js 3.4.0 (Composition API with `<script setup>` syntax), Element Plus 2.5.0 (UI components), Vue Router 4.2.5 (SPA routing), Axios 1.6.2 (HTTP client), Vite 5.0.8 (build tool with HMR), ECharts 5.4.3 (data visualization).

Component Architecture:

Layout Components: Two master layouts provide consistent navigation and user experience. `AdminLayout.vue` includes top navigation bar with system name, user info dropdown (profile/logout), and side menu (Dashboard, Students, Dormitories, Requests, Maintenance, Bills). `StudentLayout.vue` features simplified navigation focused on student-specific functions.

View Components (15 pages): Student interface includes Dashboard , Profile , Dormitory , Bills , DormChange , Maintenance . Admin interface includes enhanced Dashboard , Students management , Dormitories , DormRequests , Maintenance processing , Bills management .

API Service Layer: Centralized API functions in `/api` directory with modules for auth, student, admin operations. Each module exports typed async functions that handle request construction, error catching, and response parsing.

State Management and Routing:

Vue Router implements protected routes with `beforeEach` navigation guard checking localStorage for valid JWT token and user type. Unauthorized access redirects to login page. Authentication state managed via composable utilities (`useAuth.js`) providing reactive refs for `isLoggedIn`, `userType`, `currentUser`.

User Experience Features:

Form validation provides instant feedback using Element Plus validation rules (required fields, email format, numeric ranges, custom validators). Loading states display skeleton screens during data fetching. Error handling shows user-friendly toast messages parsed from backend error responses. Confirmation dialogs prevent accidental deletions or rejections. Search and filter functionality includes debouncing (300ms delay) to reduce API calls. Responsive design adapts to mobile/tablet/desktop using CSS media queries and Element Plus responsive grid system. Pagination with customizable page sizes (10/20/50 items) handles large datasets efficiently.

4.3 Communication Protocol and Integration

RESTful API Design: All endpoints follow REST principles with semantic HTTP methods (GET for retrieval, POST for creation, PUT for updates, DELETE for removal) and resource-based URLs (e.g., `/api/students/{id}` for individual student). Request/response bodies use JSON format with camelCase naming convention.

Request Flow: (1) User action triggers frontend function; (2) Axios interceptor adds JWT token to Authorization header (`Bearer <token>`); (3) Request sent to backend with JSON payload; (4) Backend validates token, checks permissions, processes request; (5) Response returned with HTTP status code and JSON body; (6) Axios response interceptor catches errors globally; (7) Frontend updates UI with success message or displays error notification.

Error Handling Strategy: Backend returns consistent error format: `{"detail": "Error message"}` with appropriate HTTP status (400 for validation errors, 401 for authentication failures, 403 for authorization denials, 404 for not found, 500 for server errors). Frontend interceptor maps status codes to user-friendly messages and displays toast notifications. Token expiration (401) triggers automatic redirect to login page.

CORS Configuration: Backend enables Cross-Origin Resource Sharing to allow frontend (running on different port during development) to access API. Configured with `allow_origins=["*"]`, `allow_credentials=True`, `allow_methods=["*"]`, `allow_headers=["*"]`.

API Documentation: FastAPI automatically generates interactive documentation accessible at `/docs` (Swagger UI) and `/redoc` (ReDoc). Developers can test endpoints directly through browser interface with example request bodies and responses.

5 System Deployment Guide

This section provides step-by-step instructions for TA to set up and run the system.

5.1 Database Setup

Step 1: Create Database

```
CREATE DATABASE dormitory_management_system
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE dormitory_management_system;
```

Step 2: Import SQL Scripts

Navigate to the `sql/` folder and execute the SQL files in order:

```
mysql -u root -p dormitory_management_system < sql/01_create_tables.sql
mysql -u root -p dormitory_management_system < sql/02_insert_data.sql
```

The database will be populated with 4,354 students, 1,350 dormitories, sample bills, and request records. Most data are real-world data aquired from SYSU. Thank bakabaka9405 for providing the dataset.

5.2 Backend Setup and Launch

Step 1: Install Dependencies

```
cd backend
pip install -r requirements.txt
```

Step 2: Configure Database Connection
Edit `backend/app/database.py` to set MySQL credentials:

```
DATABASE_URL = "mysql+pymysql://root:password@localhost:3306/dormitory_management_system"
```

Step 3: Start Backend Server

```
uvicorn app.main:app --reload --host 0.0.0.0 --port 8000
```

Backend will run at `http://localhost:8000`. API documentation available at `http://localhost:8000/docs`.

5.3 Frontend Setup and Launch

Step 1: Install Dependencies

```
cd frontend
npm install
```

Step 2: Start Development Server

```
npm run dev
```

Frontend will run at `http://localhost:5173`.

5.4 Test Accounts

Student Accounts: All students use default password `123456`. Login with any student ID from the database (e.g., `123090669`, `123090123`).

Administrator Accounts:

- Username: `admin`, Password: `admin123`
- Username: `dorm_manager`, Password: `manager123`
- Username: `maintenance`, Password: `maint123`

6 Summary and Harvest

6.1 Achievements

Successfully implemented a production-ready system with 21+ functions across 2 roles, frontend-backend separation, normalized 6-table database, Argon2+JWT security, and responsive web interface.

6.2 Skills Acquired

Backend: Mastered FastAPI async/await patterns, SQLAlchemy relationships and query optimization, JWT authentication, RESTful API design, and database transaction management with ACID properties.

Frontend: Proficient in Vue 3 Composition API (`ref`, `reactive`, `computed`), Vue Router with navigation guards, Element Plus components, responsive layouts (CSS Grid/Flexbox), and asynchronous state management.

Database: Applied normalization (3NF), designed ER models, wrote complex SQL (joins, aggregates, subqueries), optimized with indexing, and implemented atomic transactions.

Engineering: Adopted layered architecture, Git version control, comprehensive error handling, API documentation (Swagger), and testing (unit, integration, UAT).

Security: Implemented parameterized queries (SQL injection prevention), Argon2 password hashing, HTTPS encryption, authentication/authorization, and least privilege principle.

6.3 Key Challenges and Solutions

Migrated to Argon2 password hashing for security; implemented atomic transactions with rollback for data consistency; centralized authentication with localStorage; standardized error handling using Axios interceptors.

6.4 Conclusion

This project provided hands-on full-stack development experience, demonstrating how database, backend API, and frontend UI integrate cohesively. The modular architecture with clear interfaces ensures maintainability and scalability, preparing me for future software engineering challenges.