Software Requirements Specification (SRS)

Campus Room Schedule and Management System

Course: CSAI 203 – Introduction to Software Engineering

Instructor: Mohamed Rakha

TAs: Kholoud Osman, Maram Alazab, Mohamed Nabil, Shorouq Odaiba, Nada Mostafa

Team Members:

Ahmed Ayman Mostafa – 202401612

Mohamed Ahmed Fouad – 202401032

Yousef Hossameldin Mohamed – 202402592

Prepared for CSAI 203 Project – Zewail City University of Science and Technology

Table of Contents

1.Introduction

1.1 Purpose

The purpose of this project is to solve a recurring operational problem at the university the inefficient manual management of classroom schedules using Excel files or printed sheets. This manual process wastes time, creates confusion during room openings and closings, and makes it difficult to track maintenance issues accurately.

The proposed system provides an integrated web platform that imports room schedules directly from Excel, displays real-time room statuses with live countdowns, and enables staff, TAs, and professors to report and monitor maintenance requests seamlessly.

By digitizing this workflow, the project aims to streamline daily operations, reduce human error, and provide transparency across all involved roles. In essence, it transforms a repetitive administrative task into an efficient, automated, and traceable process aligned with modern software engineering practices.

1.2 Scope

This project is a web-based application that imports campus room schedules from Excel files and assists hall staff, TAs, and professors in managing room operations, including opening/closing and maintenance requests. It provides real-time dashboards, countdowns, and tracking logs.

In Scope (Core Functionality):

1. Import and parse room schedules from uploaded Excel files.
2. Display live room status (Open/Closed/Scheduled) with countdown timers.
3. Implement role-based access control (RBAC) for different user types.
4. Allow manual override of room states by authorized staff with audit logs.
5. Provide maintenance/fault reporting and tracking system.
6. Store all system data persistently (users, rooms, schedules, issues).

Out of Scope (MVP/Phase 4):

1. Automated email or SMS notifications.
2. Integration with university Single Sign-On (SSO).
3. Automatic synchronization of Excel imports.
4. Advanced analytics or report generation.

1.3 Definitions, Acronyms, and Abbreviations

Acronym	**Definition**
SRS	Software Requirements Specification
FR	Functional Requirement
NFR	Non-Functional Requirement
UC	Use Case
Admin	System Administrator
Staff	Hall Management Staff
TA	Teaching Assistant
Prof	Professor
MVC	Model-View-Controller
MVP	Minimum Viable Product
RBAC	Role-Based Access Control

1.4 References

1. CSAI203 Course Project Requirements (CSAI203_Course Project.pdf)

2.Project Idea Details & Specification (Project Idea Details.pdf)

3.CSAI 203, Lecture 4: Software Processes

4.CSAI 203, Lecture 5: Requirements Engineering

5.Perforce. (2025, July 11). How to write a software requirements specification (SRS) document. https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document

1.5 Overview

Section 1 introduces the project's purpose and scope. Section 2 provides an overview of the product, its environment, and users. Section 3 defines the specific functional and non-functional requirements. Section 4 contains appendices and supporting documents.

2. Overall Description

2.1 Product Perspective

The Campus Room Schedule and Management System is a new, standalone web application developed to replace the current manual and fragmented process of managing university room schedules. At present, hall management staff rely on Excel sheets and paper notes to determine which rooms should be opened, closed, or serviced a process that is time-consuming, error-prone, and lacks real-time coordination between users.

This product provides a unified digital platform that automates these daily operations. It enables users to import room schedules directly from Excel files, visualize live room statuses with countdown timers, and submit or track maintenance reports in a single interface.

The system operates independently but can be extended or integrated with existing university infrastructure in the future, such as centralized authentication or facility management systems. It represents a shift from static manual management to an interactive, data-driven workflow that supports efficiency, accountability, and transparency across all involved roles.

2.2 Product Functions

1. User Management: Secure authentication and role-based authorization.
2. Schedule Import: Upload and parse Excel schedule files.
3. Room Management: Create and manage room details (ID, building, capacity).
4. Live Dashboard: Display active rooms with countdown timers.
5. Manual Control: Enable staff to manually override room status with logs.
6. Issue Reporting: Allow users to report issues for specific rooms.
7. Issue Tracking: Enable staff to view and update issue statuses.
8. Audit Logging: Record system actions such as status changes and reports.
9. Data Export: Export schedules and logs to CSV.
10. Search & Filter: Filter rooms by building, status, or capacity.

2.3 User Classes and Characteristics

1. Hall Management Staff: Primary users responsible for daily room operations, tracking schedules, and managing issues.
2. Students: review Lectures and labs Rooms and report issues
3. TAs and Professors: View room schedules and report issues related to classes.
4. Maintenance Team: Review reported issues and update their resolution status.
5. Administrator: Manage users, roles, and system configurations.

2.4 Operating Environment

Backend: Implemented using Python Flask framework.
Frontend: Developed using HTML only.
Database: Persistent storage through database or JSON files.
Access: Available via modern web browsers such as Google Chrome and Mozilla Firefox.

2.5 Design and Implementation Constraints

• Backend must use Python Flask framework.

• Frontend must use HTML only (no external frameworks).

• Architecture must follow the MVC design pattern.

• Version control through GitHub with meaningful commits by all members.

2.6 User Documentation

A User Manual and Technical Documentation will accompany the final project. Each will describe system functions and usage for different user roles (Admin, Staff, TA/Prof).

2.7 Assumptions and Dependencies

• Staff can provide campus schedules in structured Excel (.xlsx) or .csv formats.

• Users can map Excel columns to system fields (e.g., Room ID, Open Time).

• System authentication uses local credentials (not SSO).

• System deployed on a single demonstration server (non-production).

3. Specific Requirements

3.1 Functional Requirements

FR-1 — User authentication & role-based authorization
The system shall provide secure username/password authentication and support the following roles: Administrator, Hall Staff, TA/Professor, and Maintenance. Each route and UI action shall enforce role-based permissions (ex: only Admin can manage users, Staff/Admin can perform manual Open/Close actions). Passwords shall be stored hashed. Authentication state is required to access any protected pages.

FR-2 — Upload and parse Excel/CSV schedule
The system shall allow authorized users (HallStaff) to upload `.xlsx` or `.csv` schedule files. After uploading a mapping UI will let the user map file columns to internal fields (Room ID, Date, Open time, Close time). The system validates times, detects obvious conflicts, and creates or updates Room and Schedule records. Mappings can be saved for repeated imports.

FR-3 — Room entity management
The system shall create and persist Room entities (building, room_id ,Locked?, capacity, notes, etc). Admins may add/edit/delete rooms; imports will create/merge rooms automatically when a matching identifier is found. Room detail pages show metadata and schedule history.

FR-4 — Dashboard with today's schedule & countdown timers
The system shall display a dashboard listing today's scheduled rooms and each room's current status (Open / Closed / Scheduled). Each room widget shall show a live countdown (client-side JS using server-provided UTC times) until the next scheduled change. The dashboard must refresh status on page load or user action.

FR-5 — Manual room status override
Authorized staff shall be able to manually change a room's status to Open or Closed and provide an optional reason. Each manual change must be recorded in an Schedule

FR-6 — Report an Issue (issue creation)
Any authorized user (TA/Prof/Staff/Representative student) shall be able to submit a room issue (issue type, description, room, optional attachments). The system stores the issue with status `New` and timestamps, and displays it on the Issue Log.

FR-7 — Issue tracking & lifecycle management
Issues shall have statuses (`New`, `In Progress`, `Resolved`) and allow assignment to a Maintenance user. Issue detail pages show a timeline of comments and status changes. Filters (by status, room, date) and sorting options must be available on the Issue Log page.

FR-8 — Search & filter rooms
The system shall provide search and filters on rooms and schedules (by building, capacity, status) so staff can quickly find a room and its schedule. Search results link to Room Detail pages.

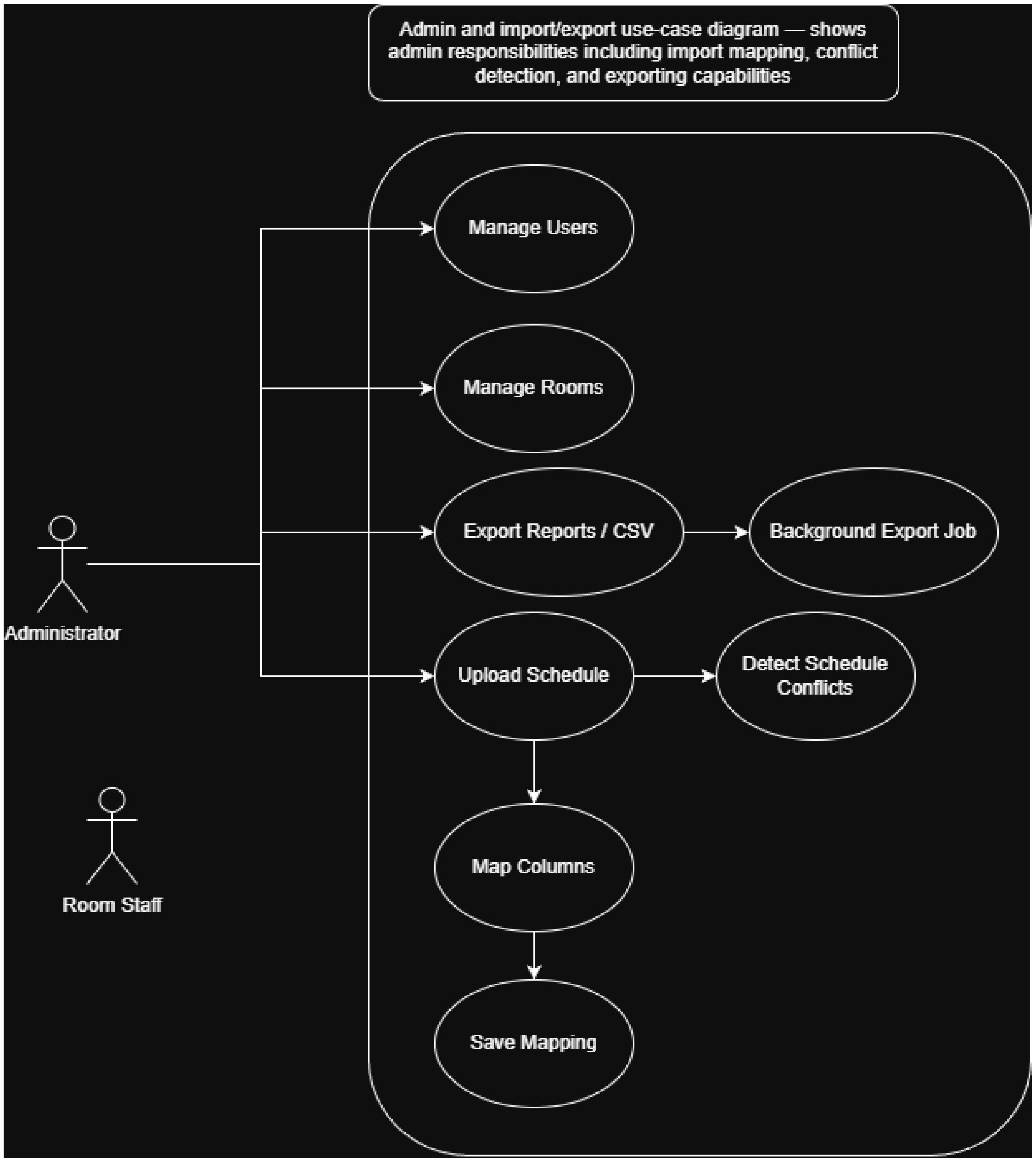
FR-9 — Export schedules & issue logs
Authorized users shall be able to export schedules and issue logs to CSV for reporting and printing (daily sheet for on-the-go staff). Exports respect applied filters (date range, building, status).

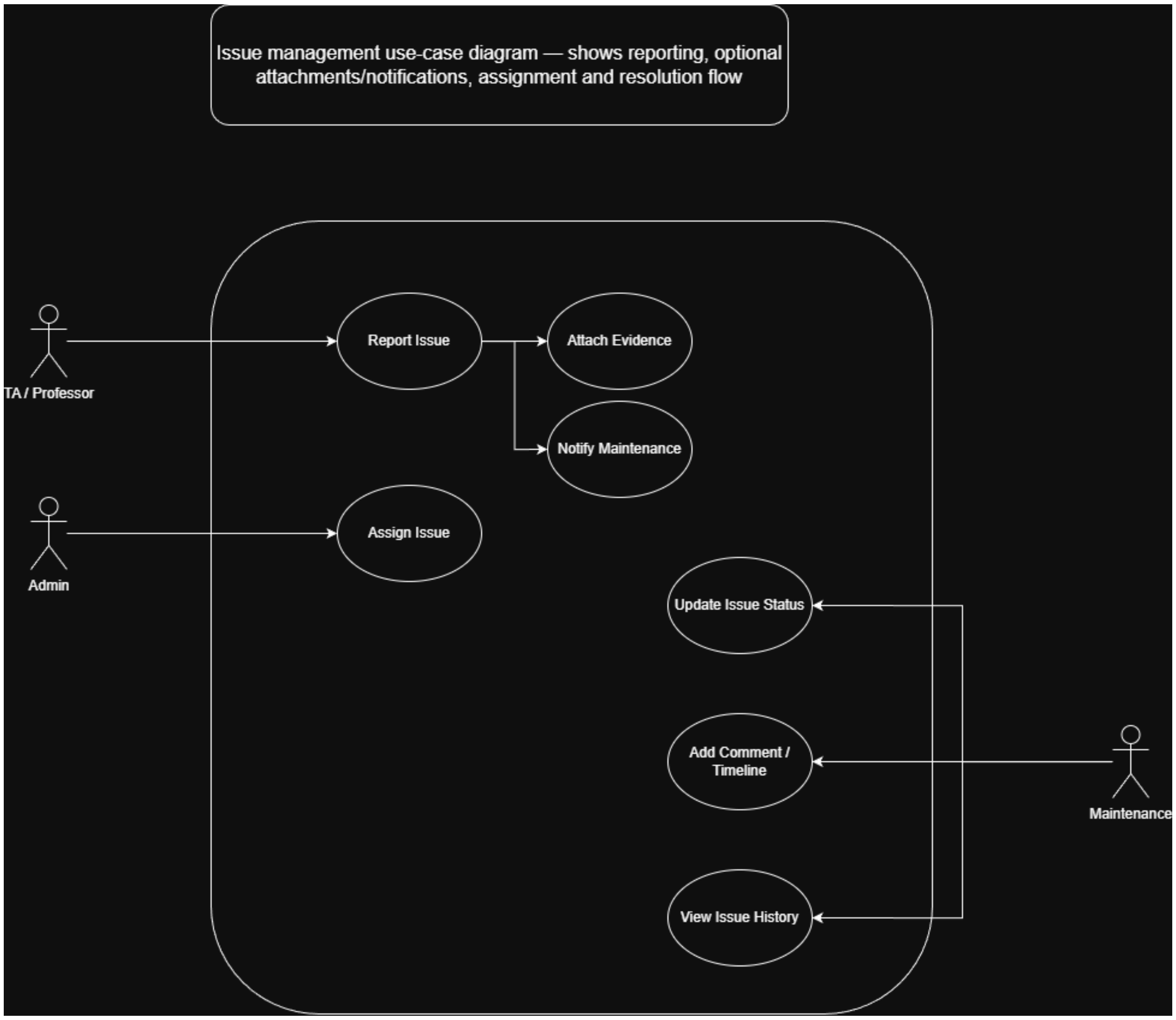
FR-10 — Notifications (optional / MVP stretch)
When a critical issue is reported (configurable by Admin), the system shall optionally send an email notification to the maintenance team. (This feature is optional for MVP; advertise in SRS as "MVP-optional / Phase-5".)

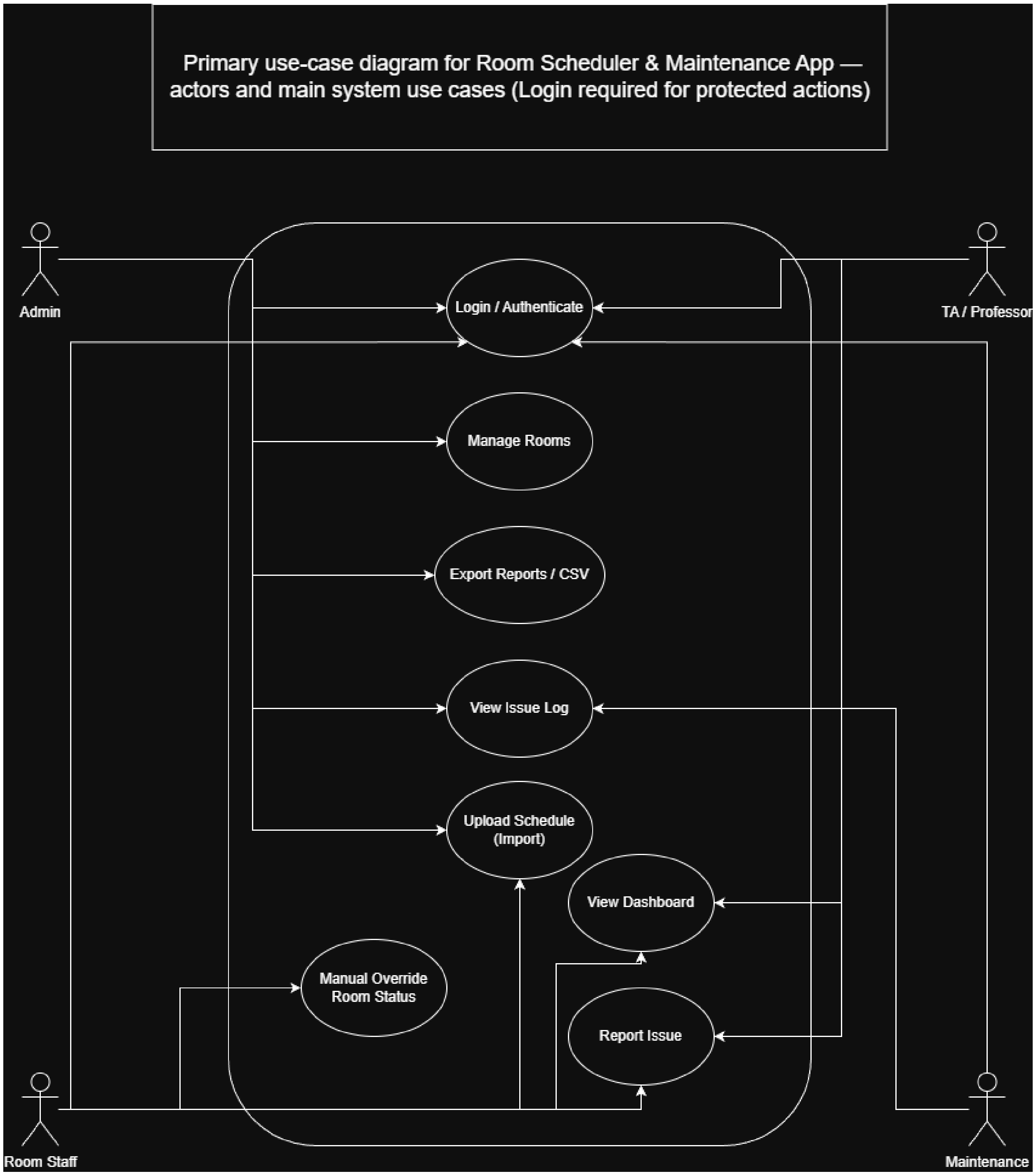
FR-11 — Persistence & basic tests
All domain data (Users, Rooms, Schedules, Issues, Imports) must be persisted in a database (or JSON file for a simple demo). The system shall include unit tests (Excel parser, schedule state transitions, issue creation) and at least one integration test for a key route (e.g., import upload or issue creation) to satisfy course requirements.

3.2 Use Case Model

3.2.1 Use Case Diagram:


Admin and import/export use-case diagram — shows admin responsibilities including import mapping, conflict detection, and exporting capabilities
Administrator
Room Staff
Manage Users
Manage Rooms
Export Reports / CSV
Background Export Job
Upload Schedule
Detect Schedule Conflicts
Map Columns
Save Mapping


Issue management use-case diagram — shows reporting, optional attachments/notifications, assignment and resolution flow
TA / Professor
Report Issue
Attach Evidence
Notify Maintenance
Admin
Assign Issue
Update Issue Status
Add Comment / Timeline
View Issue History
Maintenance


Primary use-case diagram for Room Scheduler & Maintenance App — actors and main system use cases (Login required for protected actions)
Admin
TA / Professor
Login / Authenticate
Manage Rooms
Export Reports / CSV
View Issue Log
Upload Schedule (Import)
View Dashboard
Manual Override Room Status
Report Issue
Room Staff
Maintenance

3.2.2 Use Case Descriptions:

Table 3.2.2-1 Use Case Description for Primary System Use Cases

Template Section	Description
Use Case ID	UC-01
Use Case Name	Manage Daily Room Operations
Primary Actor	Administrator, Hall Staff, TA/Professor, Maintenance
Stakeholders and Interests	Staff want an easy dashboard to see room status and perform actions quickly; Admin wants accurate records; TAs and professors need to check rooms and report issues; Maintenance needs updates on problems.
Preconditions	The user is authenticated via Login use case; schedule has been imported.
Postconditions	Room status and issue data are updated and persisted; audit entries created.
Main Success Scenario (Basic Flow)	1 User logs in → 2 System shows Dashboard → 3 User views rooms and countdowns → 4 User performs actions (override status or report issue) → 5 System updates database and audit log.
Extensions (Alternative Flows)	3a If no schedule is imported → Dashboard shows “no data”. 4a If

	user unauthorized → access denied message.
Includes / Extends (Relationships)	include: Login / Authenticate; extend: Report Issue (optional for TA/Prof).
Special Requirements	Dashboard loads under 2 seconds; countdowns must update live.
Assumptions	Stable network connection on campus; accurate system clock.
Frequency of Use	Multiple times daily by staff and TAs.

Table 3.2.2-2 Use Case Description for Issue Management Workflow

Template Section	**Description**
Use Case ID	UC-02
Use Case Name	Report and Resolve Maintenance Issue
Primary Actor	Reporter (TA/Professor or Staff), Admin, Maintenance
Stakeholders and Interests	Reporters need quick fault submission; Maintenance needs detailed info to fix issues fast; Admin monitors response times.
Preconditions	User authenticated; room exists in system.
Postconditions	Issue record created or updated; status and assignment logged;

	optional notifications sent.
Main Success Scenario (Basic Flow)	1 Reporter opens “Report Issue” form → 2 Enters room, type, description, attachments → 3 Submits form → 4 System stores issue (New) and optionally notifies maintenance → 5 Admin assigns issue → 6 Maintenance marks In Progress → 7 Maintenance resolves issue and adds comment.
Extensions (Alternative Flows)	2a Missing fields → system shows validation errors. 4a Email fails → log notification failure. 6a Issue cannot be reproduced → Admin closes without repair.
Includes / Extends (Relationships)	Extend: Attach Evidence (optional); Include: Notify Maintenance (automatic on critical issues);
Special Requirements	Must record timestamps and user IDs for each state change; attachment size ≤ 10 MB.
Assumptions	Maintenance staff check the system dashboard regularly or receive email alerts.
Frequency of Use	Whenever room fault occurs (daily to weekly).

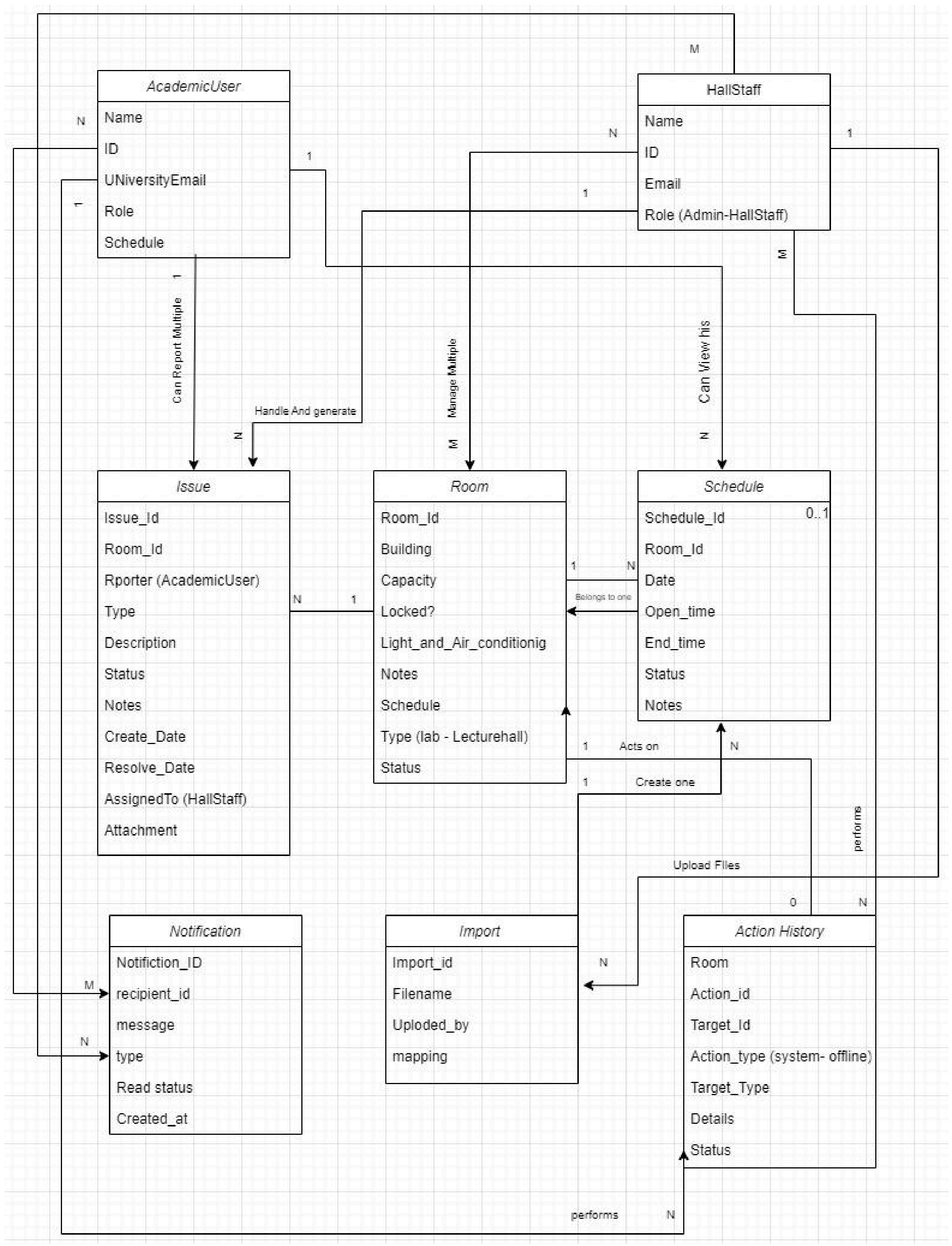
Table 3.2.2-3 Use Case Description for Admin Import/Export Use Cases

Template Section	**Description**
Use Case ID	UC-03
Use Case Name	Manage Imports and Exports (Admin Operations)
Primary Actor	Administrator (and Hall Staff for imports if authorized)
Stakeholders and Interests	Admins need accurate schedules and data backups; Staff want to avoid manual entry errors.
Preconditions	User authenticated as Admin or Staff with import rights.
Postconditions	New schedules imported and validated; exports generated for reporting.
Main Success Scenario (Basic Flow)	1 Admin selects Upload Schedule → 2 System parses file and shows mapping UI → 3 Admin maps columns and saves mapping → 4 System validates and imports data → 5 System detects conflicts and shows warnings → 6 Admin reviews and confirms → 7 Later, Admin exports reports or issue logs to CSV.
Extensions (Alternative Flows)	2a Invalid file format → error message. 5a Conflict detected → user chooses skip or override. 7a

	Large export → system creates background jobs.
Includes / Extends (Relationships)	Include: Map Columns; Save Mapping; Detect Schedule Conflicts; Extend: Background Export Job (optional for large datasets).
Special Requirements	Import parses ≤ 5 seconds for ≤ 1000 rows; export must produce UTF-8 CSV; errors logged for traceability.
Assumptions	Excel files are well-structured and follow campus format; Admin has file access.
Frequency of Use	Import weekly / Export monthly or on demand.

3.3 Domain Model

3.3.1 Conceptual Class Diagram


AcademicUser
Name
ID
UNiversityEmail
Role
Schedule
HallStaff
Name
ID
Email
Role (Admin-HallStaff)
Issue
Issue_Id
Room_Id
Rporter (AcademicUser)
Type
Description
Status
Notes
Create_Date
Resolve_Date
AssignedTo (HallStaff)
Attachment
Room
Room_Id
Building
Capacity
Locked?
Light_and_Air_conditionig
Notes
Schedule
Type (lab - Lecturehall)
Status
Schedule
Schedule_Id
Room_Id
Date
Open_time
End_time
Status
Notes
Notification
Notifiction_ID
recipient_id
message
type
Read status
Created_at
Import
Import_id
Filename
Uploded_by
mapping
Action History
Room
Action_id
Target_Id
Action_type (system- offline)
Target_Type
Details
Status
Can Report Multiple
Handle And generate
Manage Multiple
Can View his
Belongs to one
Acts on
Create one
Upload Files
performs
performs
M
N
1
N
1
1
M
1
N
M
N
N
1
1
N
0..1
1
N
1
N
1
0
N
N
M
N
N

3.3.2 Conceptual Class Diagram

3.3.2 Class Descriptions

This section describes each conceptual class identified in the system's class diagram.
Every class represents a distinct system entity, defining its purpose, attributes, and relationships with other classes

1. AcademicUser

Description:
Represents academic users of the system, including students, teaching assistants (TAs), and professors.
These users can view room schedules and report issues related to classrooms or facilities.

Attributes:

1. ID: Unique identifier for the user.
2. Name: Full name of the user.
3. UniversityEmail: Official university email used for login and communication.
4. Role: Indicates the user's role (Student / TA / Professor).
5. Schedule: References the schedules associated with the user's classes.

Relationships:

- Can report multiple **Issues**.
- Can view multiple **Schedules**.
- Can perform multiple **Actions** (recorded in **ActionHistory**).
- Can receive multiple **Notifications**.

2. HallStaff

Description:
Represents staff and administrators who manage rooms, handle issues, and oversee imported schedules.

They are responsible for daily operations such as opening/closing rooms and maintaining schedule integrity.

Attributes:

1. ID: Unique identifier for each staff member.
2. Name: Full name of the staff member.
3. Email: Institutional contact email.
4. Role: Defines whether the user is an Admin or HallStaff.

Relationships:

- Manages multiple **Rooms** and their **Schedules**.
- Handles and resolves multiple **Issues**.
- Uploads multiple **Imports** of Excel files.
- Performs multiple **Actions** (recorded in **ActionHistory**).
- Receives multiple **Notifications**.

3. Room

Description:
Represents a physical classroom, lab, or hall within the campus.
Each room has identifiable properties and is linked to schedules and issues reported by users.

Attributes:

1. Room_ID: Unique identifier for the room.
2. Building: Name or code of the building.
3. Capacity: Maximum occupancy capacity.
4. Locked?: Indicates whether the room is currently locked.

5. Light_and_Air_conditioning: Environmental features such as lighting and ventilation.
6. Notes: Additional remarks or details about the room.

Relationships:

- One **Room** can have multiple **Schedules**.
- One **Room** can have multiple **Issues**.
- Referenced by **ActionHistory** whenever an action targets a room.

4. Schedule

Description:
Defines when a room is scheduled to be open or closed on a given date.
Schedules are generated from imported files and may be adjusted manually by staff.

Attributes:

1. Schedule_ID: Unique identifier for each schedule entry.
2. Room_ID: Reference to the assigned room.
3. Date: Date for which the schedule is valid.
4. Open_time: Room opening time.
5. End_time: Room closing time.
6. Status: Current state (Open / Closed / Scheduled).
7. Notes: Additional comments or special instructions.

Relationships:

- Each **Schedule** belongs to one **Room**.
- Each **Schedule** is created through one **Import**.

- May be modified by **HallStaff** (actions recorded in **ActionHistory**).
- Can trigger **Notifications** to users (e.g., when updated).

5. Issue

Description:
Represents a maintenance or equipment problem reported for a specific room.
Academic users submit issues, and staff handle and resolve them.

Attributes:

1. Issue_ID: Unique identifier for each issue.
2. Room_ID: Room where the issue occurred.
3. Reporter: The AcademicUser who submitted the issue.
4. Type: Category of the problem (e.g., Projector, Lighting, AC).
5. Description: Detailed summary of the issue.
6. Status: (New, In Progress, Resolved).
7. Notes: Optional remarks by staff during resolution.
8. Create_Date: Timestamp of submission.
9. Resolve_Date: Timestamp when resolved.
10. AssignedTo: The HallStaff member handling the issue.
11. Attachment: Optional photo or file supporting the report.

Relationships:

- Reported by one **AcademicUser**.
- Assigned to one **HallStaff**.
- Linked to one **Room**.

- Actions on issues recorded in **ActionHistory**.
- Status changes may generate **Notifications**.

6. Import

Description:
Represents each event where an Excel or CSV file is uploaded to create or update room schedules.
This ensures that imported data is traceable to the responsible staff member.

Attributes:

1. Import_ID: Unique identifier for the import record.
2. Filename: Name of the uploaded file.
3. Uploaded_by: The HallStaff who performed the import.
4. Mapping: Column-mapping details between the file and system fields.

Relationships:

- One **Import** can generate multiple **Schedules**.
- Each **Import** is uploaded by one **HallStaff**.
- Import activities are logged in **ActionHistory**.

7. ActionHistory

Description:
Keeps a chronological record of all significant actions performed by any user within the system.
This includes manual room operations, schedule imports, issue updates, and other tracked behaviors for auditing and accountability.

Attributes:

1. Action_ID: Unique identifier for each action record.
2. Actor_ID: Reference to the AcademicUser or HallStaff who performed the action.

3. Action_Type: Short description of the action (e.g., Open Room, Import Schedule, Resolve Issue).

4. Target_Type: Entity type affected (Room, Schedule, Issue, Import).

5. Target_ID: ID of the affected entity.

6. Timestamp: Date and time of the action.

7. Details: Optional notes or justification for the action.

Relationships:

- Many **Actions** can be performed by one **AcademicUser** or **HallStaff**.

- Each **Action** can reference one target entity (**Room**, **Schedule**, **Issue**, or **Import**).

8. Notification

Description:
Represents alerts or messages sent by the system to inform users or staff of important events, such as issue updates or schedule changes.

Attributes:

1. Notification_ID: Unique identifier for each notification.

2. Recipient_ID: Reference to the AcademicUser or HallStaff receiving the notification.

3. Message: The content of the notification.

4. Type: Type of event (Issue, Schedule, System, Maintenance).

5. Created_At: Date and time the notification was generated.

6. Read_Status: Indicates whether the notification has been viewed.

Relationships:

- Each **Notification** is sent to one **AcademicUser** or **HallStaff**.

- May be triggered by changes recorded in **ActionHistory** or updates to **Issue** and **Schedule**.

3.4 Non-Functional Requirements

NFR-1 — Usability

- **Requirement:** The UI shall be simple and intuitive: staff should be able to complete primary tasks (find today's open rooms, override room status, report issue) within 3 clicks from the dashboard. Provide clear labels, consistent navigation, and mobile-friendly layout (responsive).

- **How to test:** Perform a small usability study with 3–5 representative users (hall staff/TAs). Measure number of clicks and time-to-complete for three tasks: (a) identify which rooms close within 10 minutes, (b) mark a room closed with reason, (c) submit an issue. Acceptance: median time < 2 min and ≤ 3 clicks for each task; no critical usability block reported. Also run simple keyboard/tab navigation tests for accessibility.

NFR-2 — Performance

- **Requirement:** Dashboard page load time shall be under 2 seconds on a typical campus LAN for a dataset of up to 200 rooms and associated schedules. Server-side API responses for typical queries must return within 300 ms under light load. Client countdowns must not block UI.

- **How to test:** Use synthetic load testing (e.g., ApacheBench or a lightweight script) on the deployed demo server with a dataset of 200 room records. Measure page load times and API response times; verify dashboard loads < 2s and API endpoints < 300 ms. Provide screenshots/logs in SRS test report.

NFR-3 — Reliability / Data Integrity

- **Requirement:** Critical operations (schedule import, manual override, issue creation) must be durable: writes will be retried on transient failures and never silently lost. The system shall maintain an AuditLog for manual changes.

- **How to test:** Simulate network or DB transient failures during import and verify that the system retries or fails gracefully with a clear error. Confirm that for each manual override and issue creation an AuditLog entry exists with correct metadata. Unit tests should cover retry behavior for parser and DB write functions.

NFR-4 — Security

- **Requirement:** Passwords must be hashed (e.g., bcrypt). All role-protected routes must enforce authorization server-side. Inputs from files or forms must be validated and sanitized to prevent injection. Optional: TLS in deployment.

- **How to test:** (1) Code review to confirm password hashing used. (2) Pen-test-style checks: attempt to access Admin actions while logged in as a Staff/TA account (should be forbidden). (3) Upload malformed CSV with SQL-special characters and verify no injection occurs. (4) Use an authentication flow test to ensure sessions expire appropriately.

NFR-5 — Maintainability / Testability (bonus NFR)

- **Requirement:** Codebase must follow MVC structure and be modular (app factory, services/repositories) so components are unit-testable. Unit test coverage target: at least 60% for core modules (parser, schedule status logic, issue creation).

- **How to test:** Provide unit test reports showing coverage statistics; run tests in CI (or local runner) and show passing test suite. Demonstrate that the parser module can be run independently with sample files.

3.5 External Interface Requirements

3.5.1 User Interface

The system is a web application. The user interface will be built using HTML and accessed through a standard web browser (like Chrome or Firefox). The UI will be simple, focusing on functionality with forms, buttons, and tables to display information.

3.5.2 Hardware Interface

None. This system is a pure web application and does not interface with any custom hardware.

3.5.3 Software Interface

None. The system is self-contained. The MVP (Minimum Viable Product) will not integrate with any external software systems, such as the university's Single Sign-On (SSO).

3.5.4 Communication Interface

Communication between the client (web browser) and the server (Flask backend) will use the HTTP protocol. HTTPS is recommended for secure communication in a production environment.

4. Appendices

4.1 Appendix A: Data Dictionary

N/A (Not Applicable). The main data concepts and attributes are already defined in Section 3.3 (Domain Model).

4.2 Appendix B: Glossary

Please refer to Section 1.3: Definitions, Acronyms, and Abbreviations for a complete glossary of terms used in this document.