NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Department of Computer Science

Hostel Management System

Semester Project

Submitted By:

Hamad Karim (23K-2034)

Faaez Mugse (23K-0722)

Uzair Mustafa (23I-0827)

Submitted To:

Sir Syed Ahmad Khan

Table of Contents:

Chapter 1: Introduction

i. System Overview

The **Hostel Management System (HMS)** is a software solution designed to digitalize and streamline hostel-related operations such as student registration, room allocation, fee management, and record maintenance.
It replaces traditional manual and paper-based processes with an efficient, automated, and centralized system. The system helps hostel administrators, wardens, and staff manage daily tasks with accuracy, speed, and minimal effort.

ii. Problem Statement

Most hostels still rely on **manual record-keeping**, which leads to:

- Misplaced or duplicated student records
- Difficulty in room allocation and vacancy tracking
- Delayed or inaccurate fee collection records
- High time consumption for updating logs
- Lack of transparency and real-time information

These issues affect the overall efficiency of hostel operations. Therefore, there is a need for a **fully automated system** that manages hostel activities in an organized and reliable way.

iii. Scope

The scope of the system includes:

- Student registration & storing personal details
- Room allocation, reallocation, and vacancy tracking

- Fee submission, dues tracking, and payment history
- Managing hostel blocks, rooms, and capacities
- Generating reports for admin
- Search and filter functionality for quick access
- Admin authentication & controlled access

Out of Scope (for now):

- Online payment integration
- Biometric attendance
- Visitor management system
- Mobile app

iv. Objectives

The main objectives of the Hostel Management System are:

1. **Improve efficiency** by digitalizing manual hostel tasks.
2. **Centralize student and room data** for easy management.
3. **Ensure accuracy** in fee tracking and record maintenance.
4. **Reduce administrative workload** through automation.
5. **Provide faster access** to real-time hostel information.
6. **Enhance transparency** and reduce chances of human error.

v. Users and Stakeholders

Primary Users

1. **Admin**

    - Full control over system
    - Manages rooms, students, and fees
2. **Warden/Hostel Staff**
    - Allocates rooms
    - Oversees day-to-day hostel operations

Secondary Users

3. **Students**
    - View allotted room
    - Check fee status (in extended versions)

Stakeholders

- Hostel Administration
- University/Institute Management
- IT/Software Development Team
- Finance/Accounts Department

vi. System Requirements Summary

Functional Requirements

- Add, update, delete student records
- Allocate or change rooms
- Manage fee submissions and dues
- Search student records
- Generate reports
- Admin login & authentication

Non-Functional Requirements

- **Usability:** User-friendly UI for admins/wardens
- **Performance:** Fast loading and efficient database operations
- **Security:** Role-based access, secure data storage
- **Reliability:** System should be available without crashes
- **Scalability:** Should support more students and hostels in future

vii. Software Process Model

Chosen Model: *Iterative Model*

We adopted the **Iterative Software Development Life Cycle (SDLC) model** for this project.

Why Iterative Model?

- The system can be built **in small modules**, such as student module, room module, fee module.
- Feedback from team or supervisor can be applied after each iteration.
- Requirements become clearer as the project evolves.
- Easy to add new features (like online payment, biometric) in future versions.

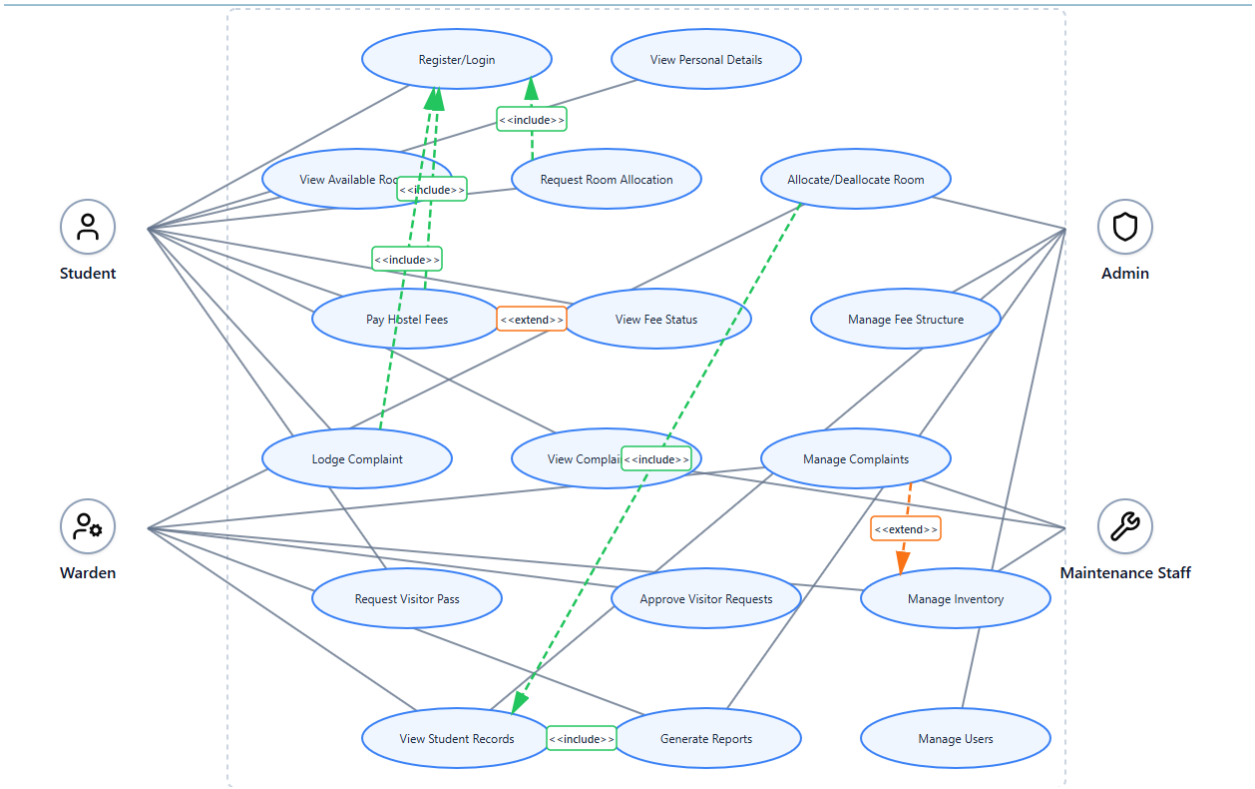
Benefits for This Project

- Suitable for academic projects with evolving requirements
- Allows early demonstration of partial working system

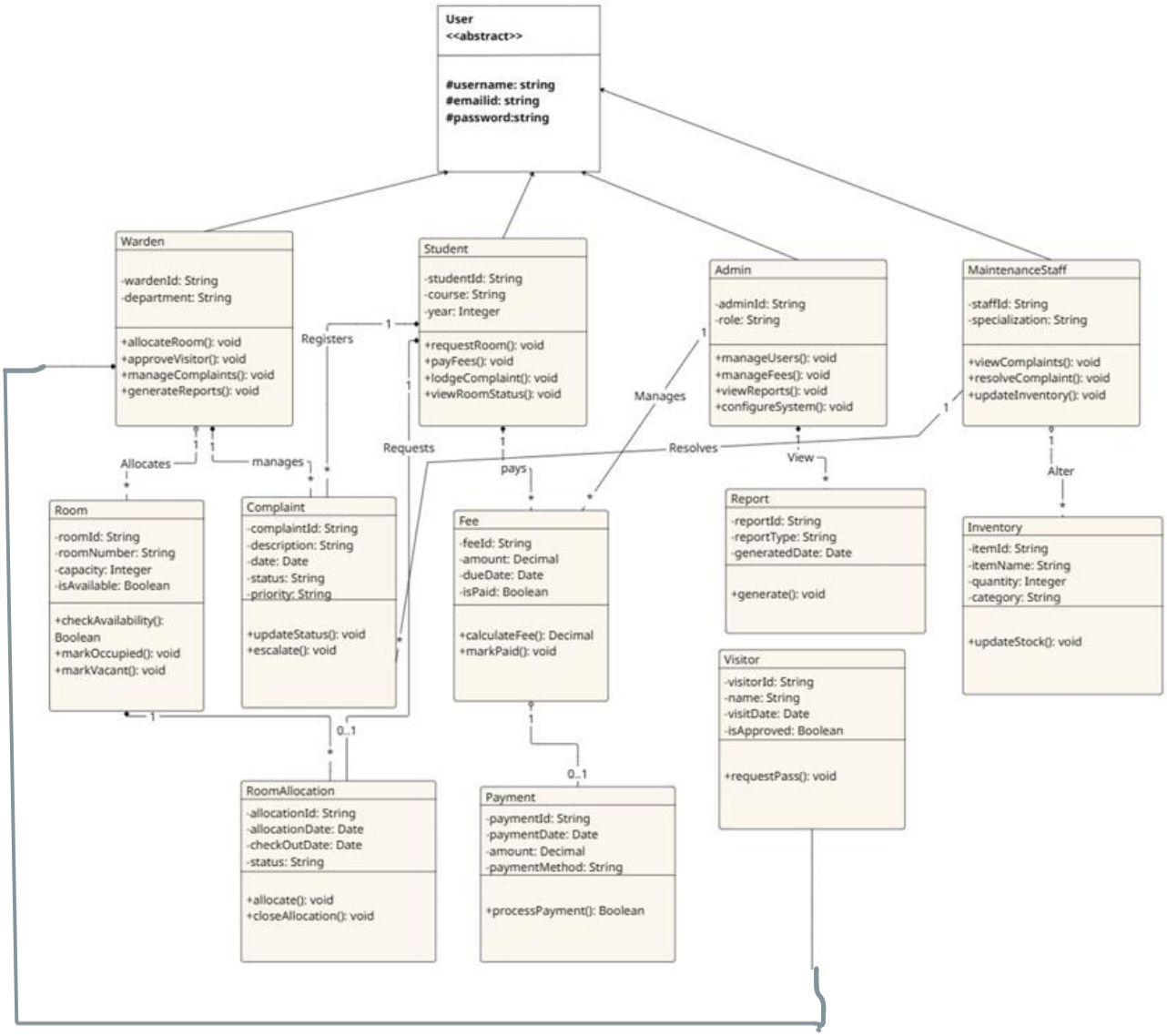
- Reduces risk and improves quality through repeated refinement

Chapter2: System Analysis and Modeling

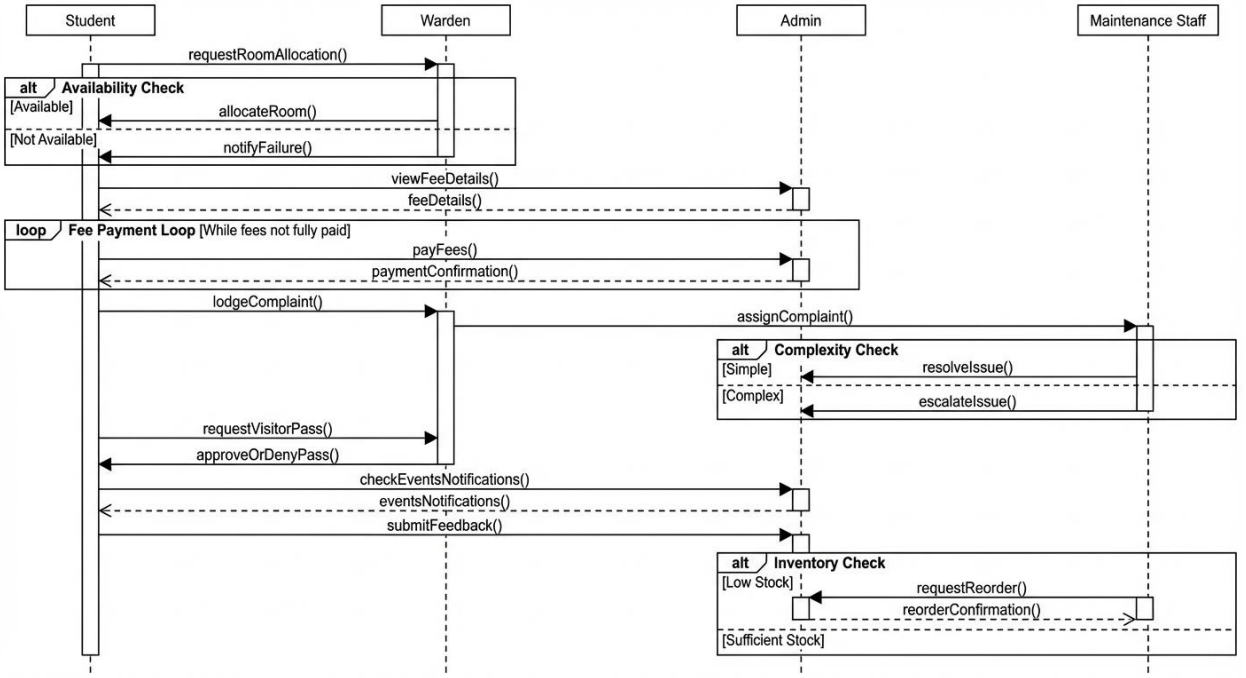
i. Use Case Diagram



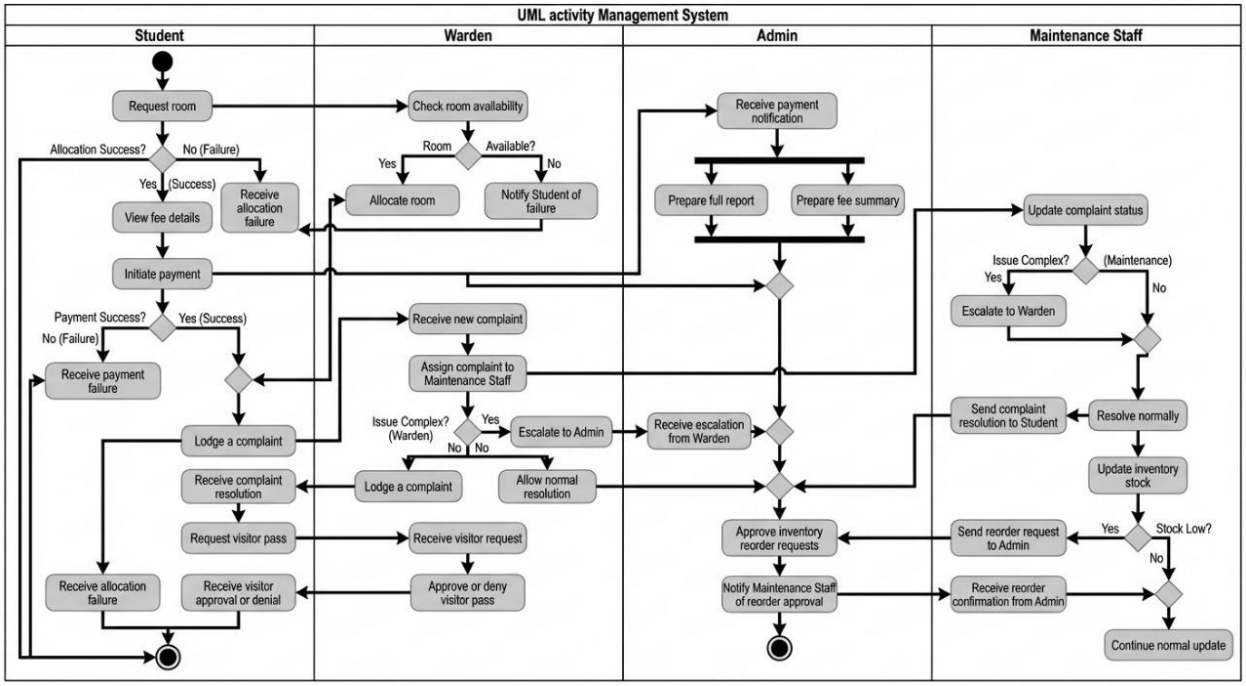
ii. Class Diagram



iii. Sequence Diagram



iv. Activity Diagram



3. Architectural View (4+1 Model)

1. Logical View (What the system contains)

Focus: *Main classes, data structures, and relationships.*

Main Classes

- **Student**
  - Attributes: studentID, name, contact, roomID, feeStatus
  - Methods: applyRoom(), submitComplaint(), payFee()
- **Admin**
  - Attributes: adminID, name

    - Methods: manageStudents(), allocateRoom(), manageFees(), resolveComplaint()
- **Warden**
    - Attributes: wardenID, name
    - Methods: approveApplication(), trackOccupancy(), handleComplaints()
- **Room**
    - Attributes: roomID, type, capacity, status (available/occupied)
    - Methods: checkAvailability(), assignStudent(), vacateRoom()
- **Complaint**
    - Attributes: complaintID, studentID, description, status
    - Methods: submit(), updateStatus()
- **Payment**
    - Attributes: paymentID, studentID, amount, date, status
    - Methods: processPayment(), generateReceipt()
- **Visitor**
    - Attributes: visitorID, studentID, timeIn, timeOut
    - Methods: logVisit()

Key Relationships

- Admin **manages many Students** and **many Rooms**.
- Each Student **occupies one Room**.

- Each Complaint **belongs to one Student**.
- Payments are **linked to Students and Rooms**.
- Warden **approves/rejects** student room applications.

2. Process View (How the system behaves internally)

Focus: *Concurrency, threads, runtime behavior, workflows.*

Main Processes

- **Authentication Process**
  - Handles login for Admin, Student, Warden.
  - Uses secure session/token management.
- **Room Allocation Workflow**
  - Student request → System checks availability → Warden/Admin approval thread.
- **Complaint Handling Process**
  - Student submits complaint → Notification thread → Admin/Warden resolves.
- **Payment Processing Workflow**
  - Student initiates payment → Payment validation service → Receipt generation.

Concurrency / Threads

- **Notification Thread:** Sends alerts (fee due, room updates, complaint status).
- **Report Generation Thread:** Generates fee reports, occupancy reports.

- **Database Update Thread:** Ensures consistent updates to room and student data.

Control Flow Example

- Student → ApplyRoom() → System validates → Warden → Approves → Updates Room & Student objects.

3. Development View (How developers structure the code)

- Focus: *Modules, packages, folder structure.*

Development Characteristics

- Object-Oriented Design (OOD)
- MVC or layered architecture
- UML design maintained in Draw.io / Lucidchart

4. Physical View (What artifacts/models exist physically in the project)

*Since your project focuses on design and documentation only (not full deployment), the Physical View represents the **UML models, tools used, and produced artifacts**, rather than servers or hardware.*

Physical Artifacts Produced

- **Use Case Diagram**
    - Actors: Admin, Student, Warden
    - Major use cases: Register/Login, Allocate Room, Pay Fees, Manage Complaints, View Reports

- **Class Diagram**
    - Core classes: Student, Admin, Room, Complaint, Payment, Visitor, Warden
    - Shows associations and relationships
- **Activity Diagrams**
    - Room allocation workflow
    - Complaint submission and resolution
    - Fee payment workflow
- **Sequence Diagrams**
    - Student applies for room → System checks → Admin/Warden approves
    - Student submits complaint → Admin/Warden resolves

Tools Used for Physical Design Models

- **Figma** – UI mockups and interface workflows
- **Draw.io / Lucid chart** – UML diagrams

5. +1 View — Use Case View (How users interact with system)

Focus: *Functional scenarios from the user's point of view.*

Main Actors

- Student
- Admin
- Warden

- Maintenance Staff

Core Use Cases

- **Register/Login**
  All users authenticate via secure login.
- **Allocate Room**
  Student applies → System validates → Warden/Admin approves.
- **Pay Fees**
  Student pays online → Payment service processes → Admin views reports.
- **Manage Complaints**
  Student submits → Admin/Warden resolves → Status updated.
- **View Reports**
  Admin views room occupancy, payment history, complaints, student records.

Additional Use Cases

- Update profile
- View available rooms
- Generate monthly/annual reports
- Handle visitor logs
- Receive system notifications