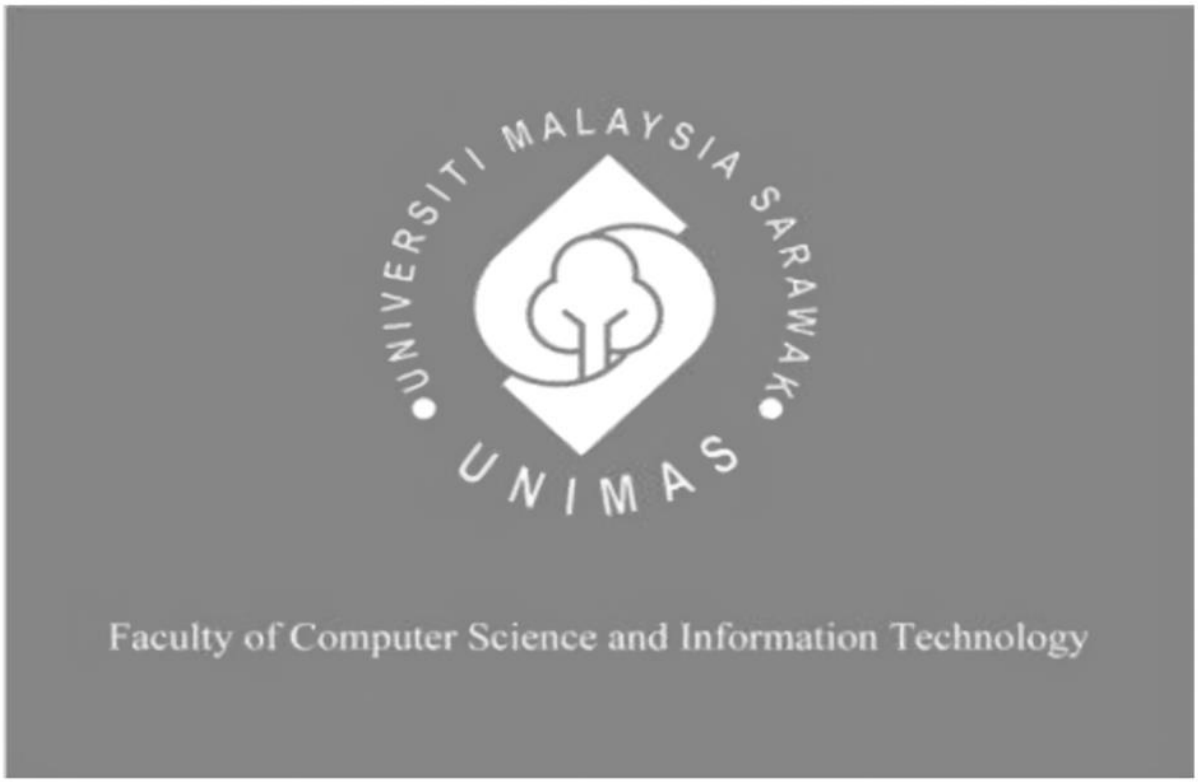


ROOMPRO: A WEB-BASED HOSTEL ROOM ALLOCATION USING GENETIC ALGORITHM AND MANAGEMENT SYSTEM

JOANNE KON XIN YUE

Bachelor of Computer Science with Honours

(Computational Science)

2025

ROOMPRO: A WEB-BASED HOSTEL ROOM ALLOCATION USING GENETIC ALGORITHM AND MANAGEMENT SYSTEM

JOANNE KON XIN YUE

This project is submitted in partial fulfilment of
requirements for the degree of
Bachelor of Computer Science with Honours

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK

2025

ROOMPRO: SISTEM PENGURUSAN DAN PERUNTUKAN BILIK ASRAMA MENGGUNAKAN ALGORITMA GENETIK BERASASKAN WEB

JOANNE KON XIN YUE

Projek ini merupakan salah satu keperluan untuk
Izajah Sarjana Muda Sains Komputer dengan Kepujian

Fakulti Sains Komputer dan Teknologi Maklumat

UNIVERSITI MALAYSIA SARAWAK

2025

UNIVERSITI MALAYSIA SARAWAK

THESIS STATUS ENDORSEMENT FORM

TITLE ROOMPRO: A WEB-BASED HOSTEL ROOM ALLOCATION USING GENETIC ALGORITHM AND MANAGEMENT SYSTEM

ACADEMIC SESSION: 2024/2025

JOANNE KON XIN YUE

(CAPITAL LETTERS)

hereby agree that this Thesis* shall be kept at the Centre for Academic Information Services, Universiti Malaysia Sarawak, subject to the following terms and conditions:

1. The Thesis is solely owned by Universiti Malaysia Sarawak
2. The Centre for Academic Information Services is given full rights to produce copies for educational purposes only
3. The Centre for Academic Information Services is given full rights to do digitization in order to develop local content database
4. The Centre for Academic Information Services is given full rights to produce copies of this Thesis as part of its exchange item program between Higher Learning Institutions [or for the purpose of interlibrary loan between HLI]
5. ** Please tick (√)

	CONFIDENTIAL	(Contains classified information bounded by the OFFICIAL SECRETS ACT 1972)
	RESTRICTED	(Contains restricted information as dictated by the body or organization where the research was conducted)
√	UNRESTRICTED	

Joanne

(AUTHOR'S SIGNATURE)

Validated by

(SUPERVISOR'S SIGNATURE)

Permanent Address

NO. 22, PASAR BATU KAWA, 93250,
KUCHING, SARAWAK

Date: 18 July 2025

Date: 18 July 2025

Note * Thesis refers to PhD, Master, and Bachelor Degree

** For Confidential or Restricted materials, please attach relevant documents from relevant organizations / authorities

DECLARATION

I hereby declare that this project is my original work. I have not copied from any other student's work or from any other sources except where due to reference or acknowledgement is not made explicitly in the text, nor has any part had been written for me by another person.

….....

Joanne Kon Xin Yue 23 July 2025

Matric No: 79684

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to everyone who has provided invaluable support during the process of completing my Final Year Project, which focused on the development of RoomPro: A Web-based Hostel Room Allocation using Genetic Algorithm and Management System**.**

Firstly, I would like to thank my supervisor, Dr. Tiong Wei King, for guiding me throughout the project. He offered valuable suggestions, corrected my mistakes, and provided guidance on areas I was unfamiliar with. I sincerely appreciate his support, which greatly helped me complete my report. Next, I would like to thank Madam Noor Hazlini Bt Borhan, the examiner, for providing constructive feedback on my report. Her insights were instrumental in improving my project.

Besides, I would like to extend my gratitude to the FYP course coordinator, Prof. Dr. Wang Yin Chai, for his lectures and for teaching us the methodologies needed to complete the project. By following his guidance, I was able to complete my project successfully.

I would also like to thank everyone who took the time to respond to my survey. Their responses were extremely helpful for the progress of my project. Lastly, I want to express my heartfelt thanks to my family and friends for their motivation and support throughout this journey. Their encouragement played a significant role in helping me complete my FYP.I sincerely thank everyone who contributed to the success of my project.

ABSTRACT

This project introduces RoomPro, a web-based hostel management system that features an intelligent room allocation mechanism based on a genetic algorithm (GA). Hostels, often favoured by students for their proximity to campus and affordability, frequently encounter challenges such as inefficient manual processes, fragmented communication, and time-consuming room assignments. RoomPro addresses these issues by offering an integrated platform that streamlines administrative tasks, facilitates announcement dissemination, and automates room allocation through an optimized GA approach. System requirements were identified through a user-focused survey, followed by the development of both physical and logical system designs. The room allocation process is designed to run efficiently in the background, minimizing disruption while ensuring optimal room assignments. The system also supports full task management with create, read, update, and delete (CRUD) operations. Functional and usability testing, supported by test cases and user feedback, confirm the system's effectiveness and user-friendliness. The development followed a structured methodology encompassing analysis, design, implementation, testing, and evaluation. The paper concludes with a discussion of the system's accomplishments, its current limitations, and potential areas for future enhancement, establishing RoomPro as a practical and efficient solution for modern hostel management.

ABSTRAK

Projek ini memperkenalkan RoomPro, sebuah sistem pengurusan asrama berasaskan web yang menampilkan mekanisme pengagihan bilik secara pintar menggunakan algoritma genetik (GA). Asrama yang sering menjadi pilihan pelajar kerana kedudukannya yang berhampiran dengan kampus dan yuran yang berpatutan, sering menghadapi cabaran seperti proses manual yang tidak efisien, komunikasi yang tidak terpusat, serta pengagihan bilik yang memerlukan masa yang panjang. RoomPro menangani isu-isu ini dengan menyediakan satu platform bersepadu yang memudahkan tugas pentadbiran, menyampaikan pengumuman dengan lebih sistematik, dan mengautomasikan proses pengagihan bilik melalui pendekatan GA yang dioptimumkan. Keperluan sistem dikenal pasti melalui tinjauan yang memfokuskan kepada pengguna, diikuti dengan pembangunan reka bentuk fizikal dan logikal sistem. Proses pengagihan bilik direka bentuk untuk beroperasi secara efisien di latar belakang, sekali gus mengurangkan gangguan dan memastikan pengagihan bilik yang optimum. Sistem ini juga menyokong pengurusan tugasan penuh melalui operasi cipta, baca, kemas kini dan padam (CRUD). Ujian kefungsian dan kebolehgunaan yang disokong oleh kes ujian dan maklum balas pengguna mengesahkan keberkesanan serta kemudahan penggunaan sistem ini. Pembangunan sistem ini mengikuti metodologi yang tersusun, merangkumi analisis, reka bentuk, pelaksanaan, pengujian dan penilaian. Kertas kerja ini diakhiri dengan perbincangan mengenai pencapaian sistem, kekangan yang dihadapi serta cadangan penambahbaikan pada masa hadapan, sekali gus menjadikan RoomPro sebagai satu penyelesaian yang praktikal dan cekap untuk pengurusan asrama moden.

Table of Contents

List of Tables

List of Figures

CHAPTER 1: INTRODUCTION

1.1 Introduction

The college hostel serves as a temporary accommodation for student during their study, particularly for those coming from distant areas who need a place to stay to pursue their education. The hostel serves as a convenient housing option for students. Effective hostel management is essential to ensure smooth operations and provide a comfortable living environment for the hostel community.

One of the most important procedures in hostel operations is room allocation. Since the number of rooms and residents involved is usually large, manual allocation can easily lead to errors, such as duplicate assignments**,** assigning rooms in poor condition or offering more rooms than are available. These errors are time-consuming and increase the staff's workload. To address this issue, a Genetic Algorithm (GA) is introduced to optimize the room allocation process. GA is a well-known optimization algorithm used for solving complex problems efficiently. By implementing GA, the system can assign the most suitable room to each applicant while considering constraints such as high demand for specific rooms and limited room availability. If an applicant's preferred room is unavailable, the system will allocate the next best option while still meeting other constraints. The GA will be integrated into the system to provide a more efficient and convenient way for staff to allocate rooms. Room assignments will be based on applicants' preferences, and a weightage system will be applied to ensure optimal allocation, maximizing the likelihood that applicants receive their desired rooms.

The Hostel room allocation and management system is a centralized digital platform with advanced room allocation and comprehensive hostel management features. By utilizing genetic algorithm for room allocation, the system assigns rooms that best match students'

preferences without the need for manual assigning. Additionally, the system provides complete hostel management capabilities, streamline all related tasks. Systematic data organization through digital system enhances overall efficiency in managing the hostel.

This project aims to propose a web-based application, namely Hostel Room Allocation and Management System for the hostel community to address the limitations of the current hostel management approach. The proposed system streamlines the hostel tasks by eliminate the need of manual processes, enhance the management efficiency and bring convenient to the hostel community. With advanced management capabilities, the system not only improves hostel operations but also supports a better living environment for the community.

1.2 Problem Statement

The problems lie within the current hostel management approach can be categorized into three parts:

- lack of centralised platform for the hostel community to handle hostel related tasks and keep updated with the latest announcements
- the room is assigned manually by the hostel staff which is time consuming and prone to errors
- the hostel related tasks are managed manually or through paper-based process, which is inefficient and lack of systematic organization for hostel operations

Currently, there are not a centralised platform for the hostel community to manage the hostel activities such as apply for hostel, reporting a complaint or keep updated with the latest hostel announcements. Hostel community need to complete the tasks through different channels which is inconvenient. The community also need to browse through the hostel social

media group chat to find the latest hostel announcements, which can lead to missed updates and potential issues due to overlooked information.

Additionally, the hostel rooms are assigned manually by the hostel staff, which is time-consuming and prone to mistakes. This manual process increases the likelihood of assigning an occupied room to another resident or overlooking assignments entirely, further burdening the staff's workload.

The hostel operations are inefficient as the tasks are managed manually or through paperwork. The resident and housing information are recorded on paper manually and is inefficient for tracking the information and time consuming. There is also a lack of systematic organization for facility management, making it difficult for staff to oversee and maintain facilities effectively.

1.3 Objective

The project aims to provide a centralized platform for the hostel community to enhance management efficiency through a hostel management system that includes an advanced room allocation feature.

The objectives are as below:

- to design and develop a system with an advanced room allocation feature and hostel management functionalities to streamline hostel operation
- to implement a Genetic Algorithm-based room allocation features in the proposed system to optimise the room allocation process
- to evaluate the functionality and usability of the proposed system

1.4 Methodology

The methodology used in this project is the Waterfall methodology. This approach provides a clear project structure and simplifies progress tracking, as each phase is completed sequentially, avoiding cross-functional work. With the Waterfall model, design is based directly on requirements, and implementation follows the established design, reducing the likelihood of errors in later stages.

The Waterfall methodology consists of five sequential phases: requirements, design, implementation, testing, and deployment and maintenance. In the Requirements phase, all project needs are collected and clearly defined to ensure a detailed understanding of the project scope. In the Design phase, developers create a technical solution to meet the requirements. A logical design is developed based on the requirements and then transformed into a physical design. Once the design is complete, the Implementation phase begins, where programmers code the system according to the proposed design. This phase is often shorter, as the research and design have already been completed. If significant changes are needed, the project must revert to the design phase for revisions. In the Testing phase, testing is conducted after implementation to ensure the system functions as expected and good user experience. Both functional and usability testing are performed to identify and address potential issues in the system. Finally, in the Deployment and Maintenance phase, the system is deployed to users, and maintenance begins. This phase involves addressing defects, handling improvement requests, and keeping the system up-to-date with new versions as necessary.

By using this methodology, project progress becomes easier to track, and errors can be identified early, making the project more manageable and effective. Figure 1.1 illustrates the Waterfall methodology.

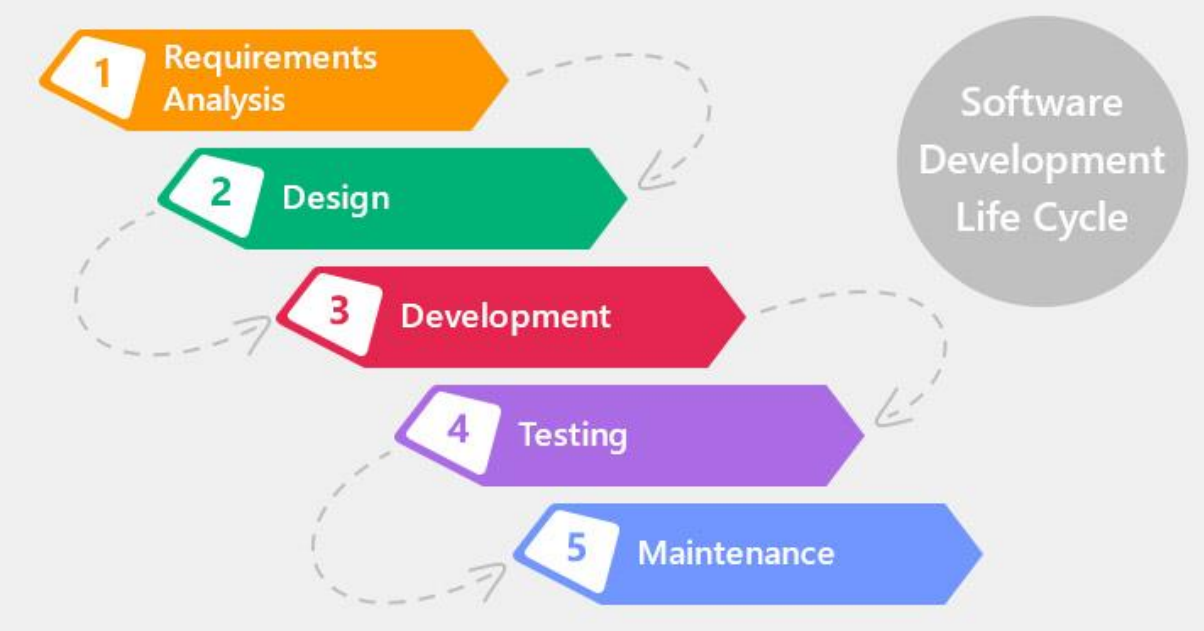


Figure 2.1: Five phase of Waterfall methodology (Team, 2024)

1.5 Scope

The scope of the system can be divided into two main components: **hostel create, read, update and delete (CRUD) functionalities** and **room allocation function.**

1. **Hostel CRUD Functionalities**: The system will include six primary functionalities that utilize CRUD operations to manage data efficiently. These functionalities are:
    - Resident and housing information management
    - Application management
    - Facilities management
    - Visitor tracking
    - Complaint and feedback handling
    - Announcement notices

    Each of these functionalities will incorporate CRUD operations to systematically organize and manage data.

2. **Room Allocation**: A key component of the system is advanced room allocation feature, developed using a defined room allocation algorithm. This algorithm is based on a genetic algorithm, a mathematical approach that optimizes room assignments. The system will allocate rooms to residents in a way that best matches their preferences through the defined room allocation algorithm.

The system will have two types of users which are resident(student) and staff (admin). Staff can access to all the functionalities while resident can access to limited functionalities.

This structured approach ensures that the system meets the needs of hostel management effectively while enhancing operational efficiency.

1.6 Significance of the project

The significance of this project lies in providing a centralized platform that enhances hostel operational efficiency and offers convenience to the hostel community in managing hostel related tasks. By utilizing the system, data can be organized systematically, giving users a clearer overview for monitoring information. The advanced room allocation features of the system reduce staff workload and save time, enabling smoother operations. As a result, the hostel management become more effective and efficient.

1.7 Schedule

Project Gantt Chart:

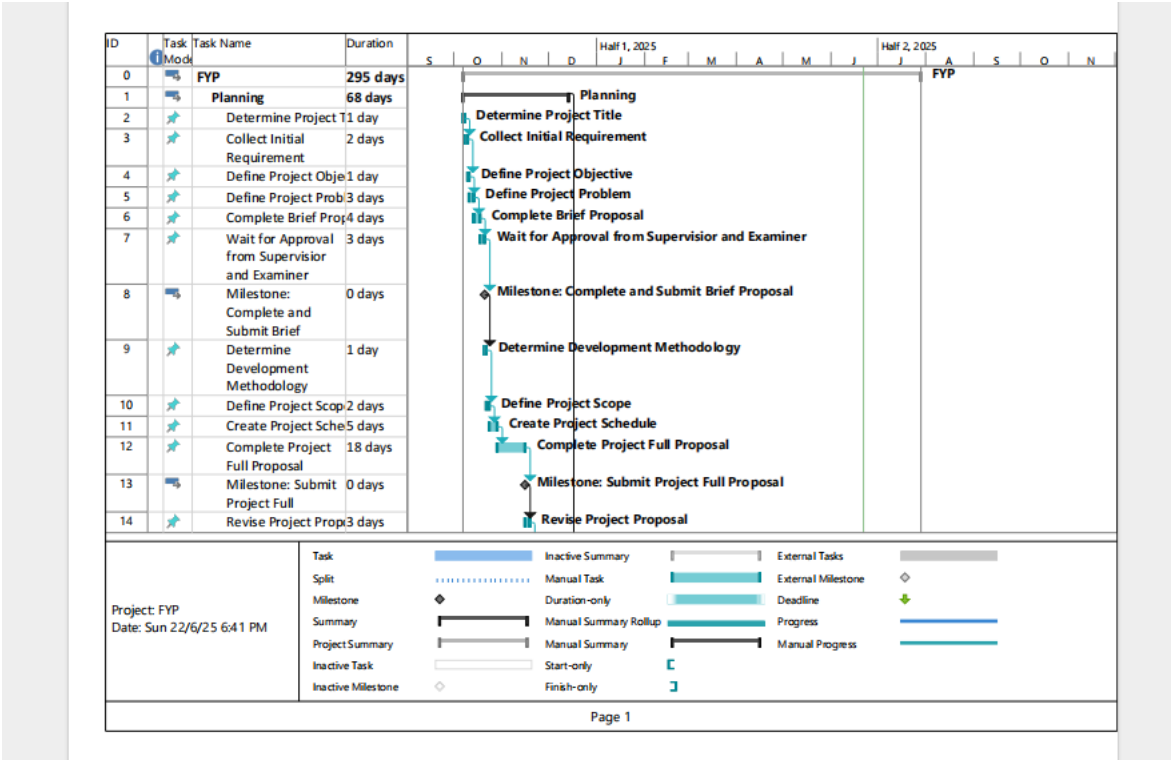


Figure 2.2: Planning Phase of Gantt Chart

Figure 1.2 shows the Gantt chart for planning phase. In planning phase, project title is determined and complete the project full proposal.

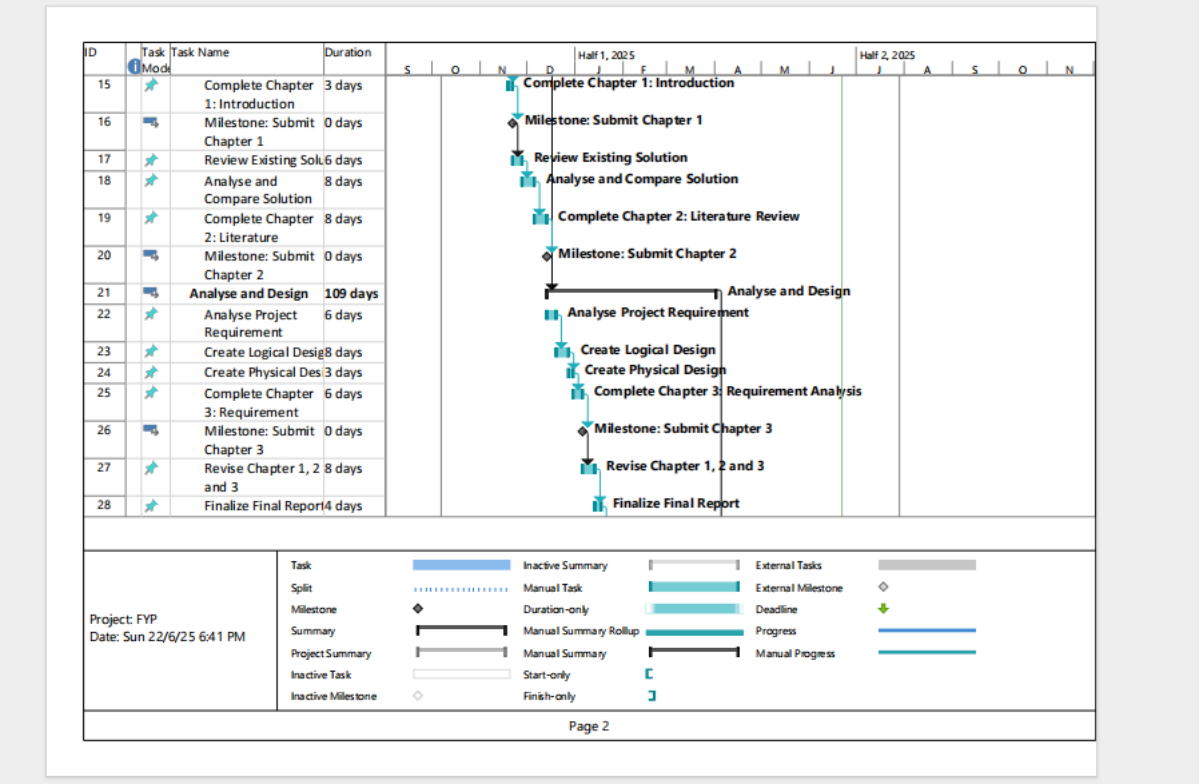


Figure 2.3: Analysis and Design phase of Gantt Chart

Figure 1.3 shows the analysis and design phase of the Gantt Chart. After completes the project proposal, Chapter 1: Introduction and Chapter 2: Literature Review, analysis and design phase is carried out. Requirements are analysed and designs are created. After completing Chapter 3: Requirement Analysis and Design, the FYP 1 report is finalized and submitted.

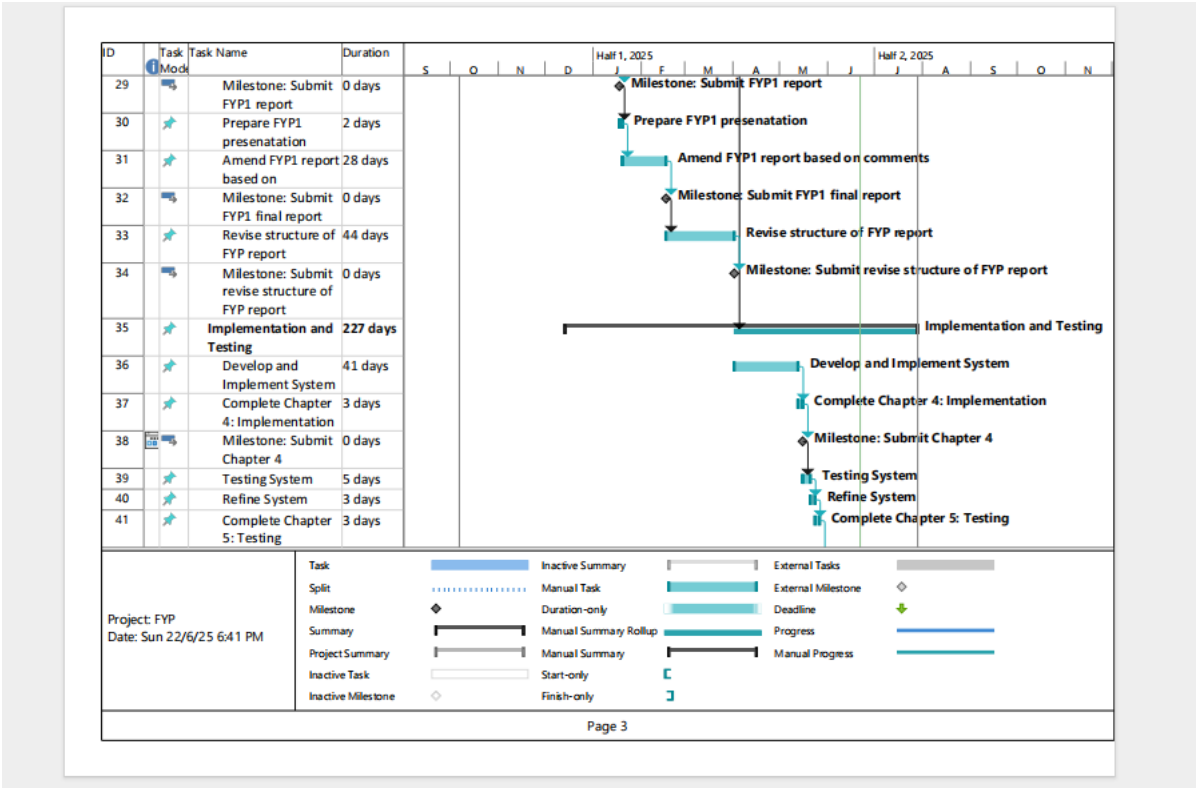


Figure 2.4: Implementation Phase of Gantt Chart

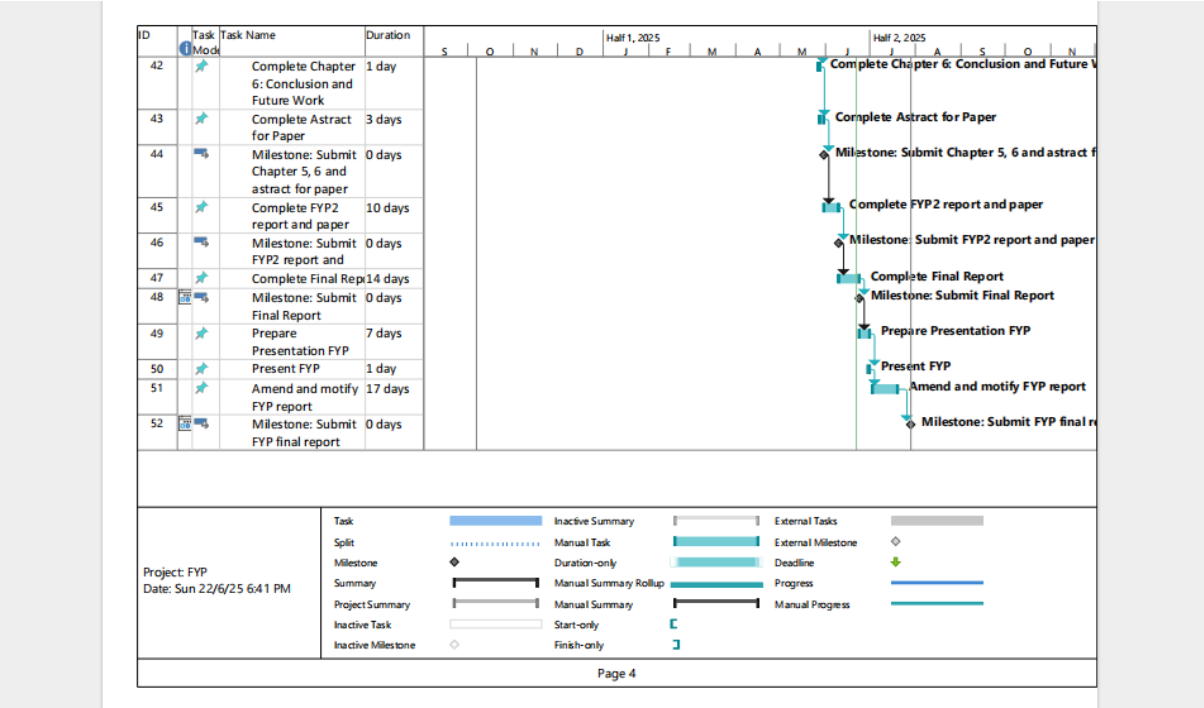


Figure 2.5: Continuous part of Implementation and Testing Phase of Gantt Chart

Figures 1.4 and 1.5 show the Gantt chart for the implementation and testing phase. After the analysis and design phase complete by submit the FYP1 final report, in implementation and testing phase, development and implementation start after the requirements are defined and the designs are created. Once the implementation is completed, testing begins. After completing Chapter 4: Implementation and Chapter 5: Testing, Chapter 6: Conclusion and Future Work begins. Upon the completion of Chapter 6, the FYP 2 report is finalized, the project is presented, and FYP 2 is completed.

1.8 Expected Outcome

The outcome of this project is a web-based system equipped with an advanced room allocation feature and comprehensive hostel management functionality. The room allocation function will allocate the rooms that meet the defined criteria to residents, using an allocation algorithm constructed based on Genetic Algorithm. The system will incorporate six primary hostel management functionalities, all utilizing CRUD operations. These functionalities include application management, resident and housing information management, facilities management, complaint and feedback handling, visitor tracking and announcement notice

1.9 Report Outline

The report will contain six chapters, namely: Introduction, Literature Review, Methodology, Implementation, Testing, and Conclusion.

Chapter 1 provides the introduction and overview of the project. It includes the problem statement, project scope, objectives, methodology, significance, project schedule, expected outcomes, and an outline of the report.

Chapter 2 will present the literature review. Existing similar systems will be reviewed, and a comparison will be made between the current system and the proposed system to identify areas for enhancement. Related research on the algorithm will also be examined to gain a deeper understanding and improve the system's performance.

Chapter 3 focuses on requirement analysis and design. This chapter will define the project methodology with a comprehensive explanation. Requirements will be gathered through surveys and analysed. Both logical and physical designs will be included.

Chapter 4 covers implementation. It will outline the development environment setup and detail the step-by-step process of system implementation.

Chapter 5 is dedicated to testing. Functionality and usability testing will be carried out to verify system performance and ensure good user experience. Any identified bugs will be resolved, and improvements will be made to enhance system performance.

Chapter 6 is the conclusion. This chapter will summarize the project's achievements, outline any constraints encountered, and suggest possible improvements to enhance the system in future work.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

A literature review is a critical examination and assessment of previous research on a certain topic. It is an essential part of academic research, as it summarizes the current knowledge in a specific field, identifies gaps in the literature, and guides future research. The goal of a literature review is to highlight key themes and topics, uncover gaps in existing studies, and explore opportunities for further investigation. It also aims to evaluate and integrate prior research to identify trends and patterns (What Is the Purpose and Importance of Literature Reviews in Research? n.d.). This chapter reviews three similar existing systems and the primary features of the existing and proposed system will be compared. The current manual management methods used by hostel management will also be examined. Additionally, the genetic algorithm, which serves as the core component for constructing the room allocation algorithm in the proposed system, will be reviewed. Furthermore, the technological tools intended for system development will be discussed.

2.2 Review of Manual Approach and Existing Similar Systems

The current UNIMAS hostel management manual approach and existing similar systems are reviewed. The primary features of existing similar systems and proposed system are compared.

2.2.1 Manual Approach of UNIMAS Hostel Management

The hostel at UNIMAS currently relies on manual methods for managing its operations. Student and room information is recorded on paper. Students are required to physically locate the QR code to make a complaint. They must scan the QR code, complete the Microsoft form,

and then submit it to file a complaint. Announcements and important messages are communicated via the WhatsApp social media platform. Hostel registration processes, including check-in and check-out, are managed through Microsoft Forms. The hostel application and payment process are done through different websites.

Figure 3.1: Announcement WhatsApp Group for Kolej Rafflesia UNIMAS

Figure 2.1 shows the Kolej Rafflesia Announcement WhatsApp group, which is dedicated solely to posting hostel-related announcements. Residents are required to join this group and only administrators are allowed to send messages.

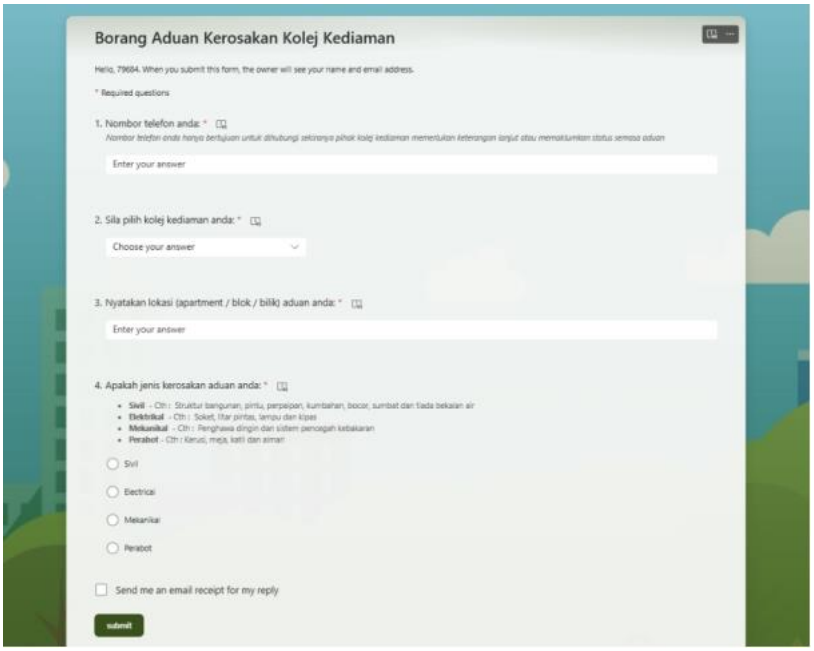


Figure 3.2: Microsoft Complaint Form

Figure 2.2 shows the hostel complaint form created used Microsoft Forms, which residents use to report any issues. Resident need to access the form via a separate link or by scanning the provided QR code.

2.2.2 Hostel Management System

Hostel Management System (HMS) is a web-based application used to manage the student and hostel information. The system is divided into two panels: a user panel for students and an admin panel for administrators, each offering tailored functionalities to meet their respective needs. The user panel allows students to register and log in to the system to access its features. The dashboard provides students with an overview of their personal information and room details for quick access. Through the hostel booking feature, students can book available rooms, while the room details section offers comprehensive information about the rooms. Students can also utilize the complaint registration feature to report issues regarding hostel facilities. Additionally, the feedback feature enables students to share their experiences

and suggestions for improvement. The my profile section allows students to view and update their personal information, while the change password option provides secure credential management. To ensure accountability, the system includes an access log that records student activities within the platform. The admin panel equips administrators with tools to manage the hostel effectively. The dashboard offers a centralized view of student information, room management, course management, complaints, and feedback. Administrators can perform CRUD (Create, Read, Update, Delete) operations on courses and hostel rooms through the course management and room management sections, respectively. They can assist students with registration and room bookings via the student registration management feature. The complaint management section enables administrators to review and address complaints submitted by students. Similarly, the feedback management feature provides access to all student feedback, facilitating the identification of areas for improvement. To ensure transparency, the user access logs allow administrators to monitor user activities within the system.

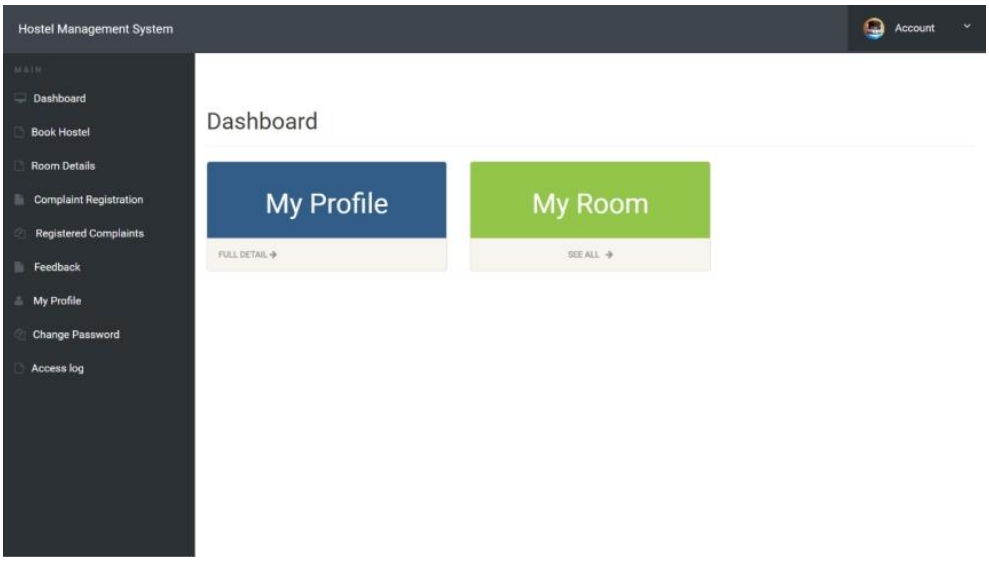


Figure 3.3: Dashboard for Student Panel

Figure 2.3 shows the student panel dashboard, which consists of two main sections: My Profile and My Room. These sections provide quick access for students to navigate to their profile and room details.

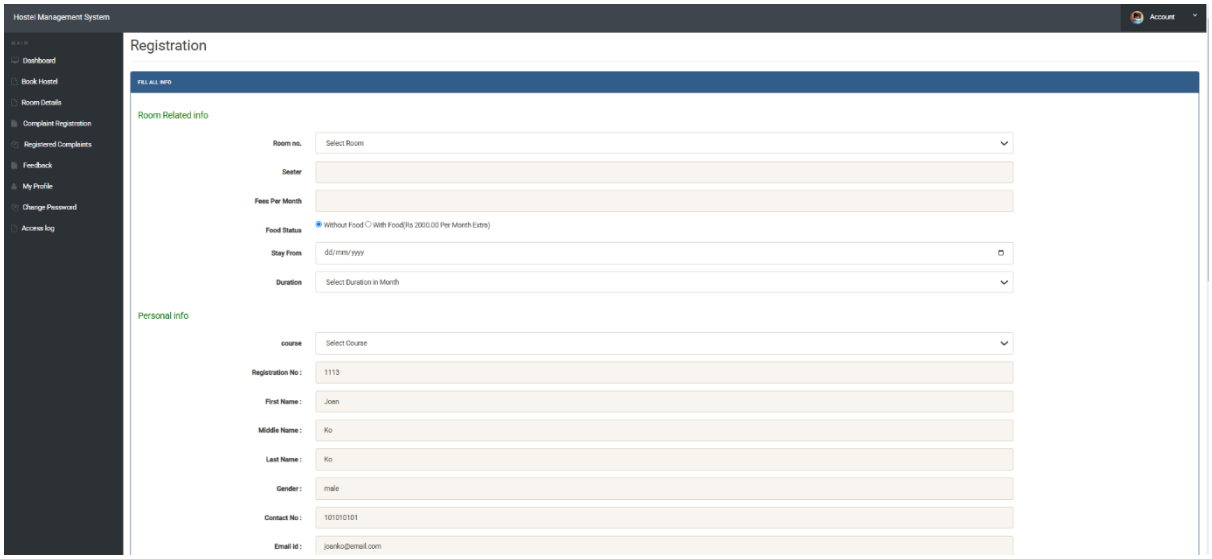


Figure 3.4: Hostel Booking feature and Personal Detail Registration for Student Panel

Figure 2.4 shows the hostel booking and personal details form for the student panel. The personal details are auto-filled. Students are required to complete this form when booking a hostel room.

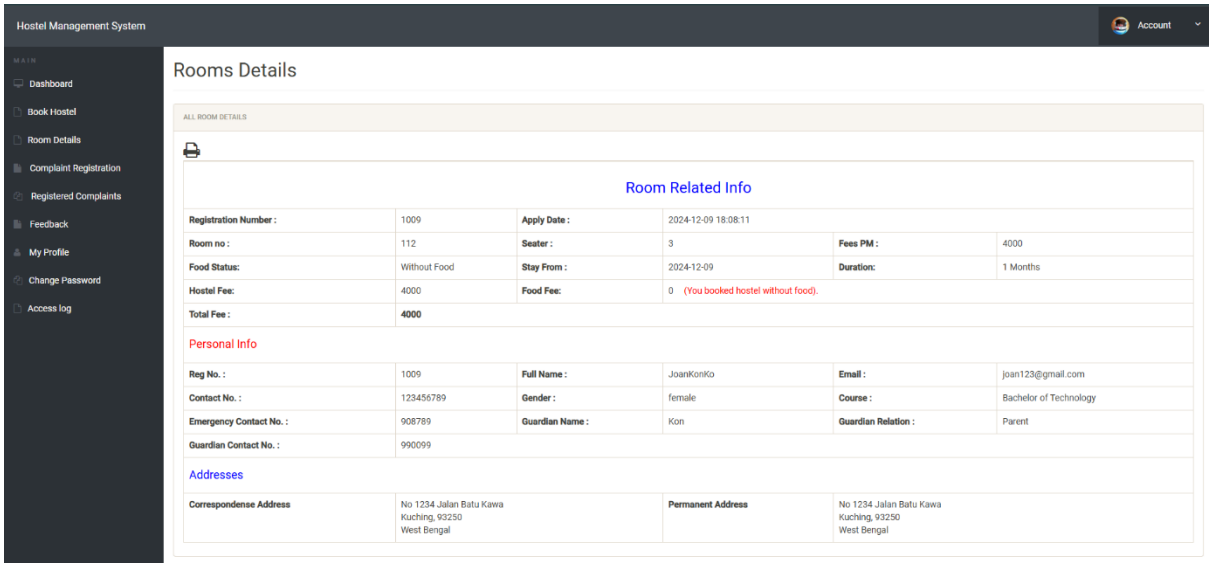


Figure 3.5: Room Details for Student Panel

Figure 2.5 displays the room details section for the student panel. This page presents information such as room type, capacity, duration, and the student's personal information.

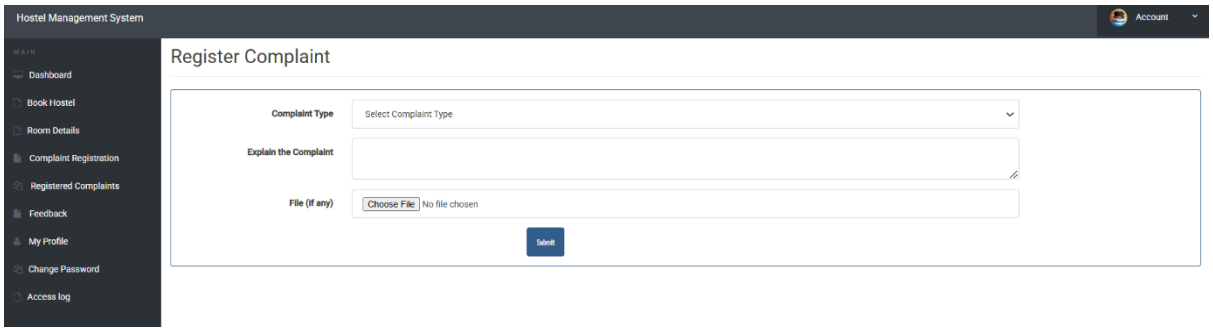


Figure 3.6: Complaint Registration for Student Panel

Figure 2.6 shows the hostel complaint form available in the student panel. Students must fill out this form to submit a complaint by selecting the complaint type, providing details, and uploading an attachment if available.

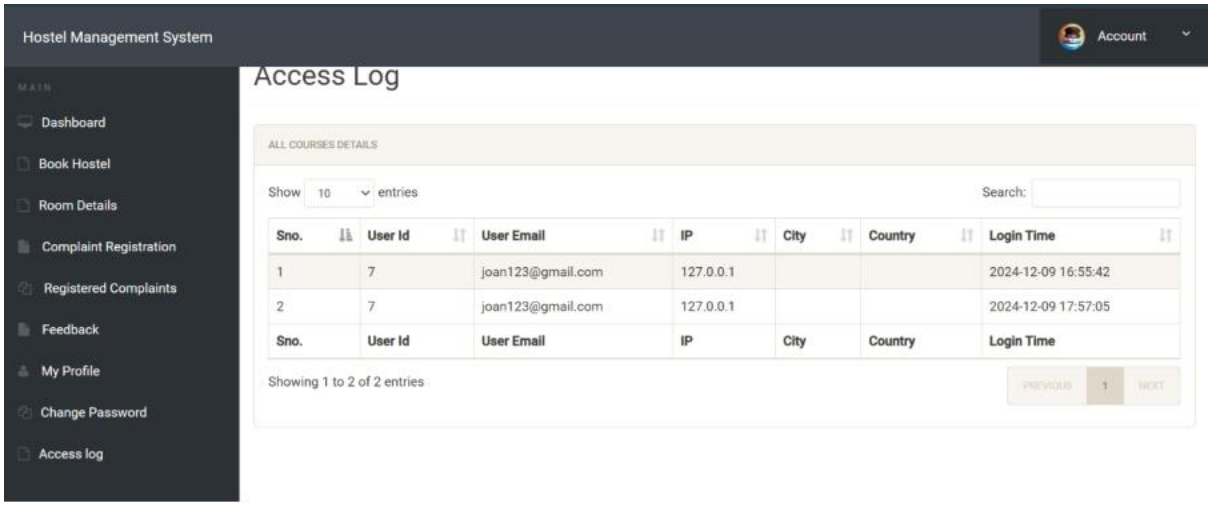


Figure 3.7: Access Log for Student Panel

Figure 2.7 presents the student access log. This will record the student's activity within the system.

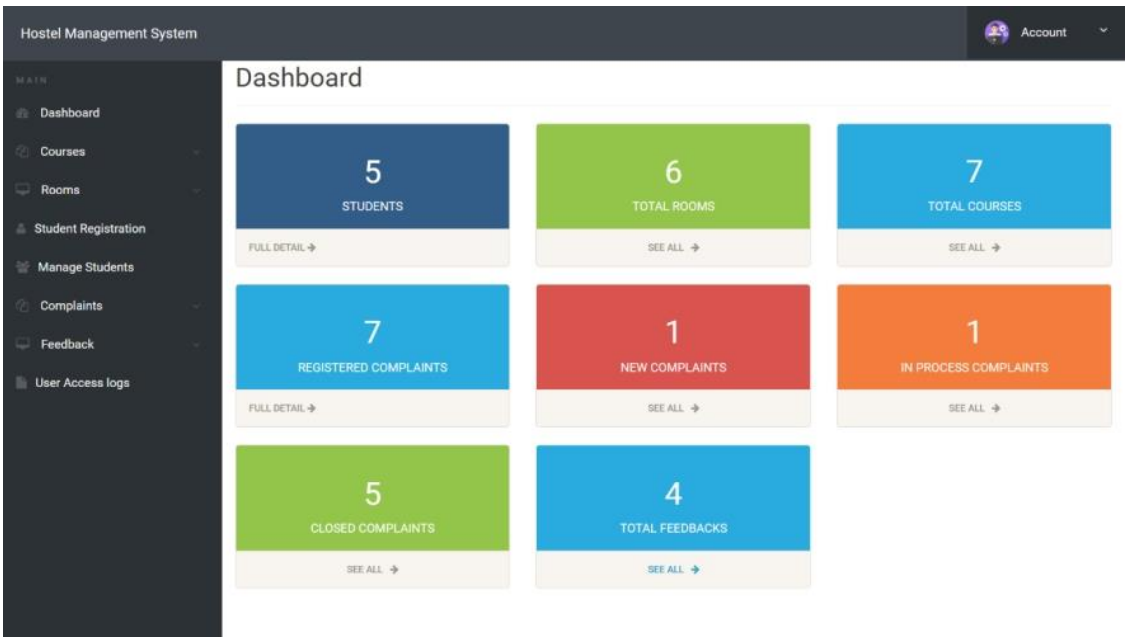


Figure 3.8: Dashboard for Admin Panel

Figure 2.8 shows the admin panel dashboard, which consists of eight sections: *Students*, *Total Rooms*, *Total Courses*, *Registered Complaints*, *New Complaints*, *In Progress Complaints*, *Closed Complaints*, and *Total Feedbacks*. These sections provide quick access for admins to navigate to specific areas.

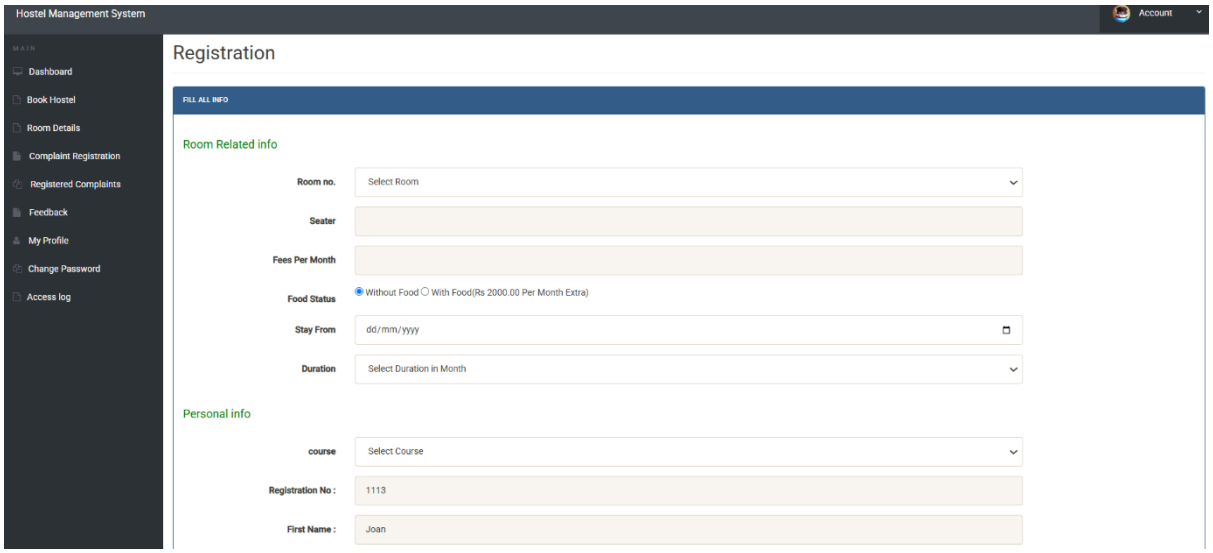


Figure 3.9: Room Allocation Feature and Student Registration for Admin Panel

Figure 2.9 shows the room allocation and student registration feature in the admin panel. Admins can allocate rooms to students and register student information.

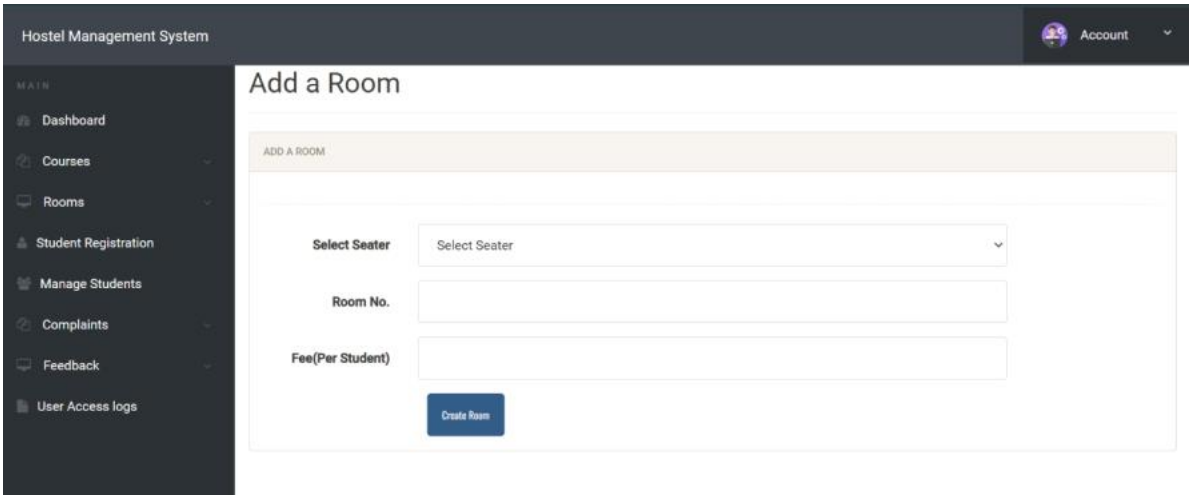


Figure 3.10: Add a room feature for Admin Panel

Figure 2.10 shows the add a new room feature for admin panel. Admin can create a new room by entering its capacity, room number and fee.

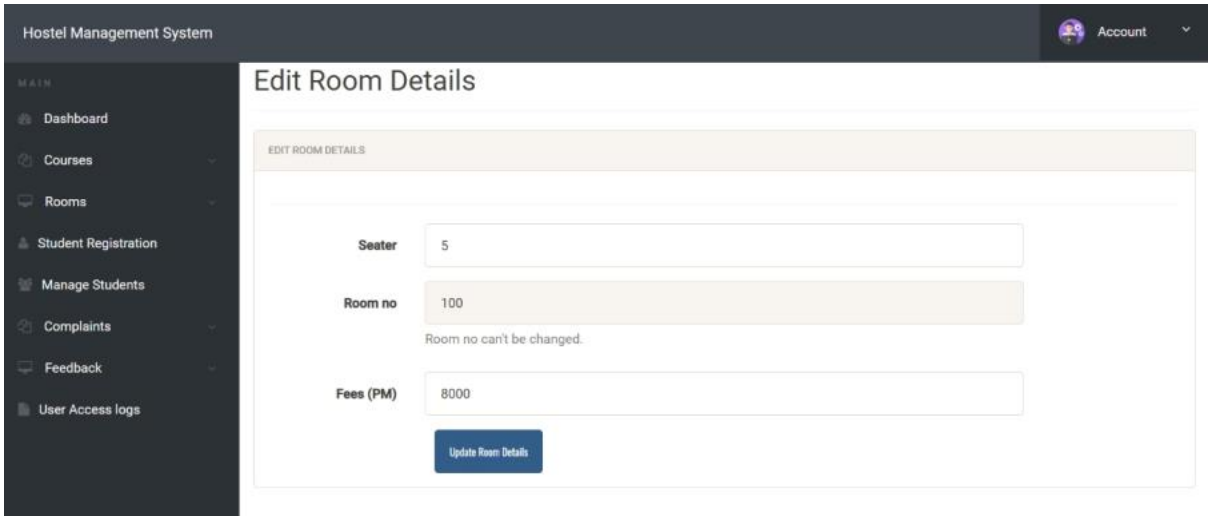


Figure 3.11: Edit room details for Admin Panel

Figure 2.11 shows the edit exiting room details feature for admin panel. Admin can modify the existing room information.

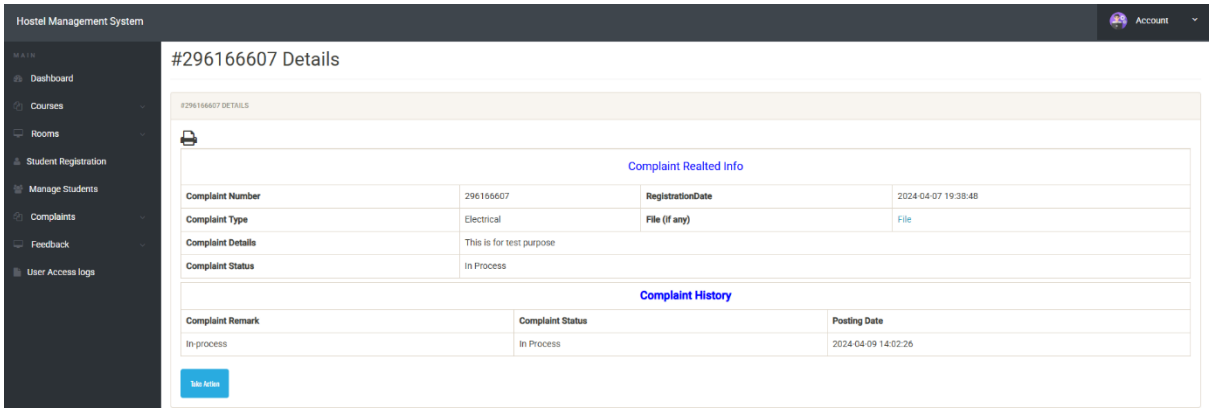


Figure 3.12: Complaint Details for Admin Panel

Figure 2.12 shows the complaint details for admin panel. Admin can view complaint information and respond by clicking the Take Action button.

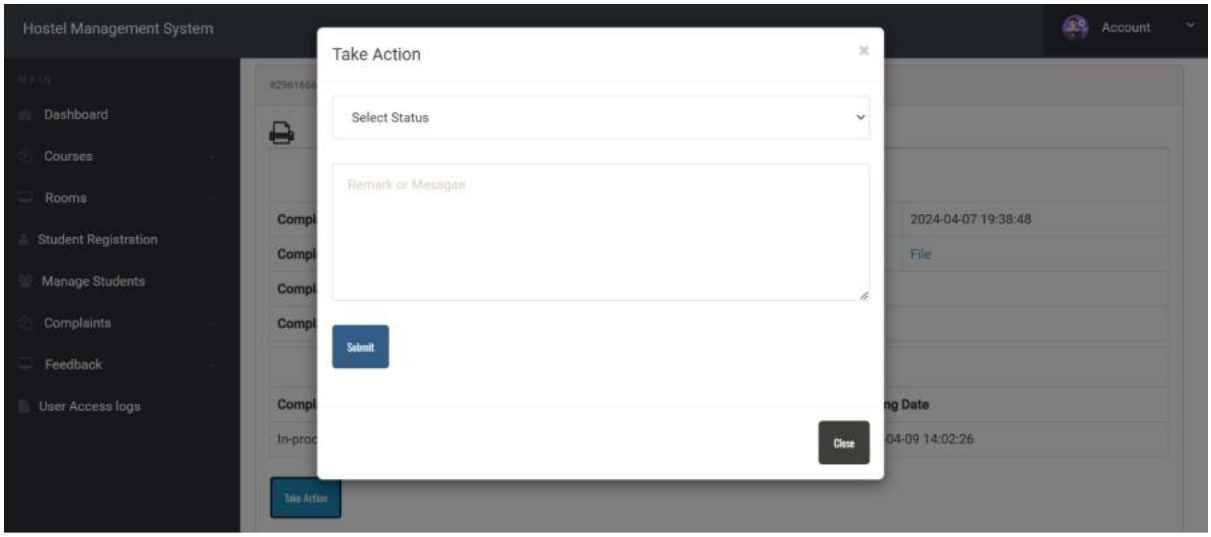


Figure 3.13: Complaint feedback feature for Admin Panel

Figure 2.13 shows the pop-up text box of complaint feedback feature for admin panel. Admin can enter and submit their response to a complaint here.

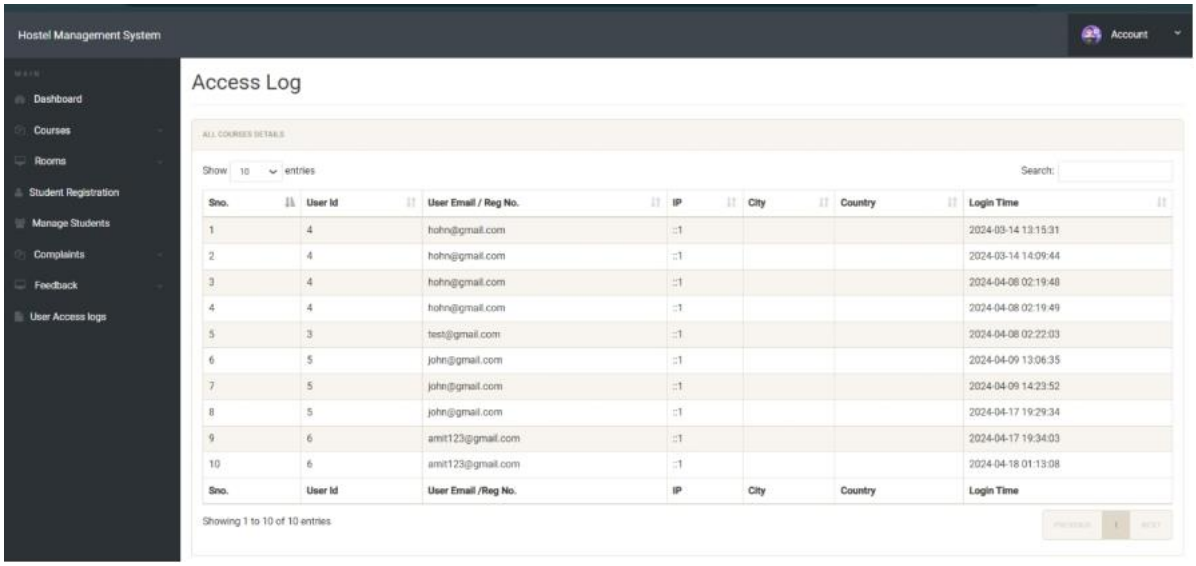


Figure 3.14: User access log for Admin Panel

Figure 2.14 shows the user access log for admin panel. This page records user activity within the system.

2.2.3 Online Hostel Management System

Online Hostel Management System is a web-based system designed to streamline hostel booking and management. It consists of two main panels: the Student Panel and the Admin Panel**,** each offering distinct functionalities tailored to their respective users. The Student Panel is accessible only after the admin registers the student's details in the system. Once logged in, students can use the sidebar to access features such as the dashboard, hostel booking, room details, and system log activities. Through the Hostel Booking feature, students can select available rooms and must complete their personal information before submitting a booking request. Upon successful submission, they can view the details of their assigned room in the My Room Details section. The System Log Activities feature enables students to track their activities within the system. Additionally, students can manage their profiles and update account settings, such as changing their password, by clicking on their profile image in the Account Settings section. The Admin Panel provides a more comprehensive set of tools for managing the system. The Dashboard offers an overview of key metrics, including the total number of registered students, available rooms, booked rooms, featured courses, and student activity. However, it does not include quick access shortcuts to specific functions. Admins can register student accounts, view and manage student details, and assist with hostel bookings by manually entering all necessary information, unlike the student process which auto-fills some details. The Hostel Students section allows admins to view all students residing in the hostel. Furthermore, the system enables CRUD (Create, Read, Update, Delete) operations for Manage Rooms and Manage Courses, providing flexibility in managing room allocations and course offerings. Like students, admins can also update their profiles and change their account passwords in the Account Settings section.

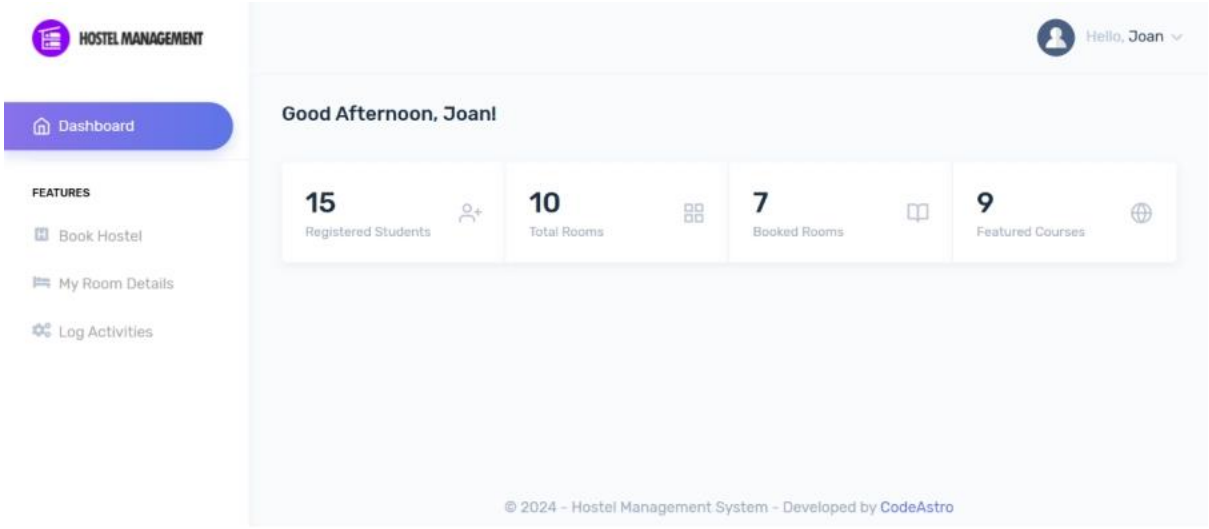


Figure 3.15: Dashboard for Student Panel

Figure 2.15 shows the dashboard of student panel consist of four sections: Registered Students, Total Rooms, Booked Rooms and Featured Courses. These sections provide summarised information only and do not offer quick access links.

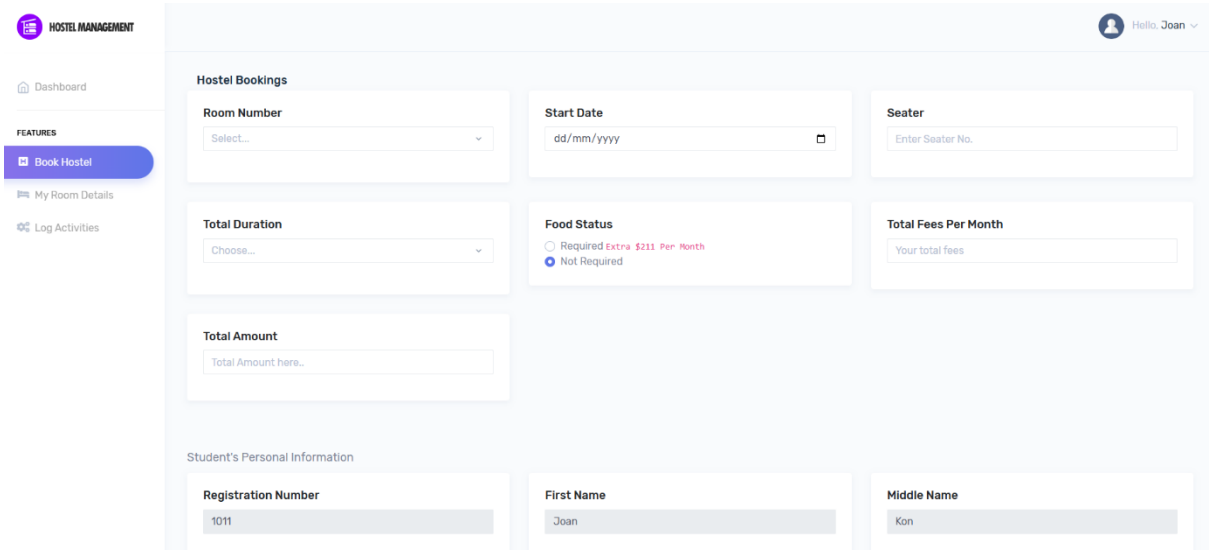


Figure 3.16: Hostel Booking for Student Panel

Figure 2.16 shows the hostel booking for student panel. The personal details are auto-filled. Student fills in the related information to book a hostel.

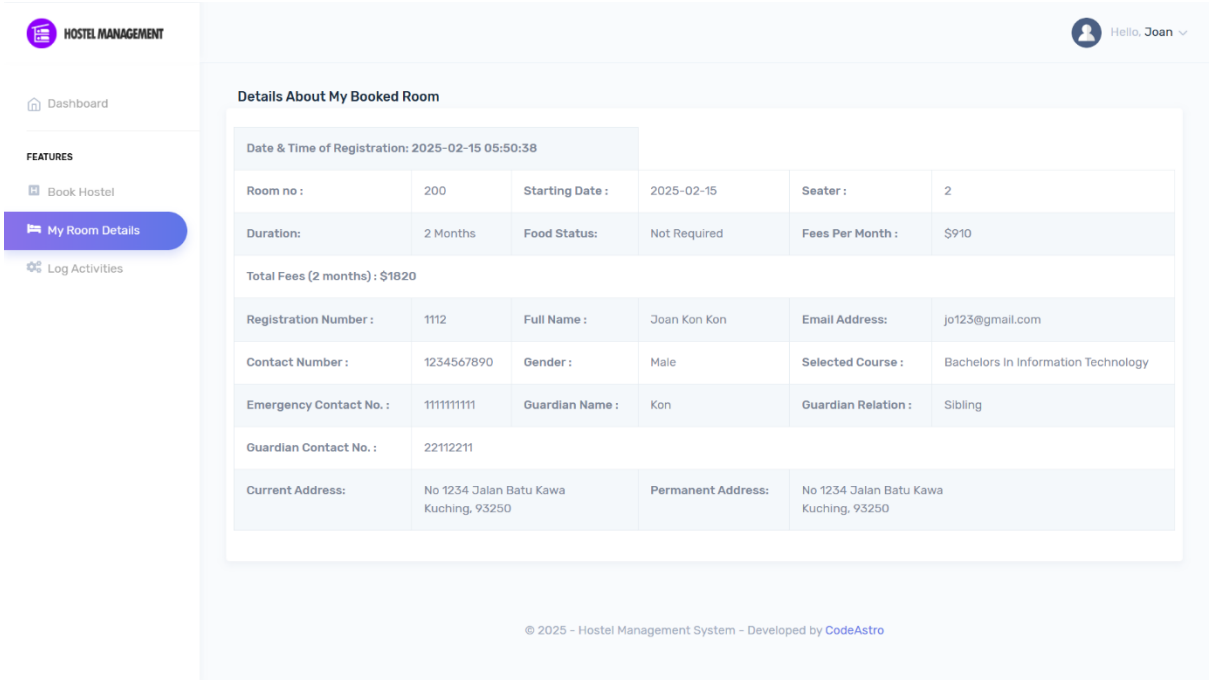


Figure 3.17: Room Details for Student Panel

Figure 2.17 shows the room details for student panel. This page shows the details of booked rooms and the student's personal information.

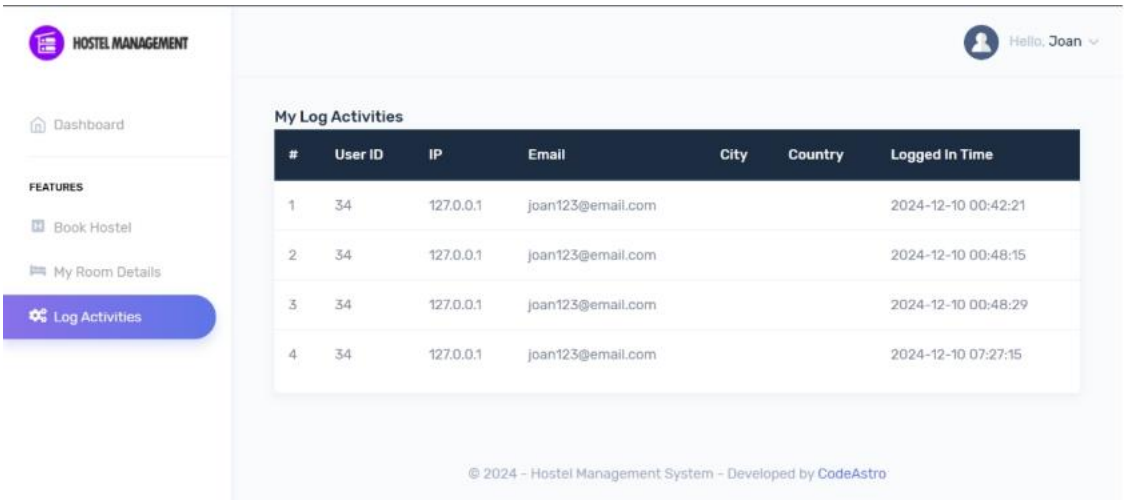


Figure 3.18: User Log Activities for Student Panel

Figure 2.18 shows the user log activities for student panel. This page records the user's activity within the system.

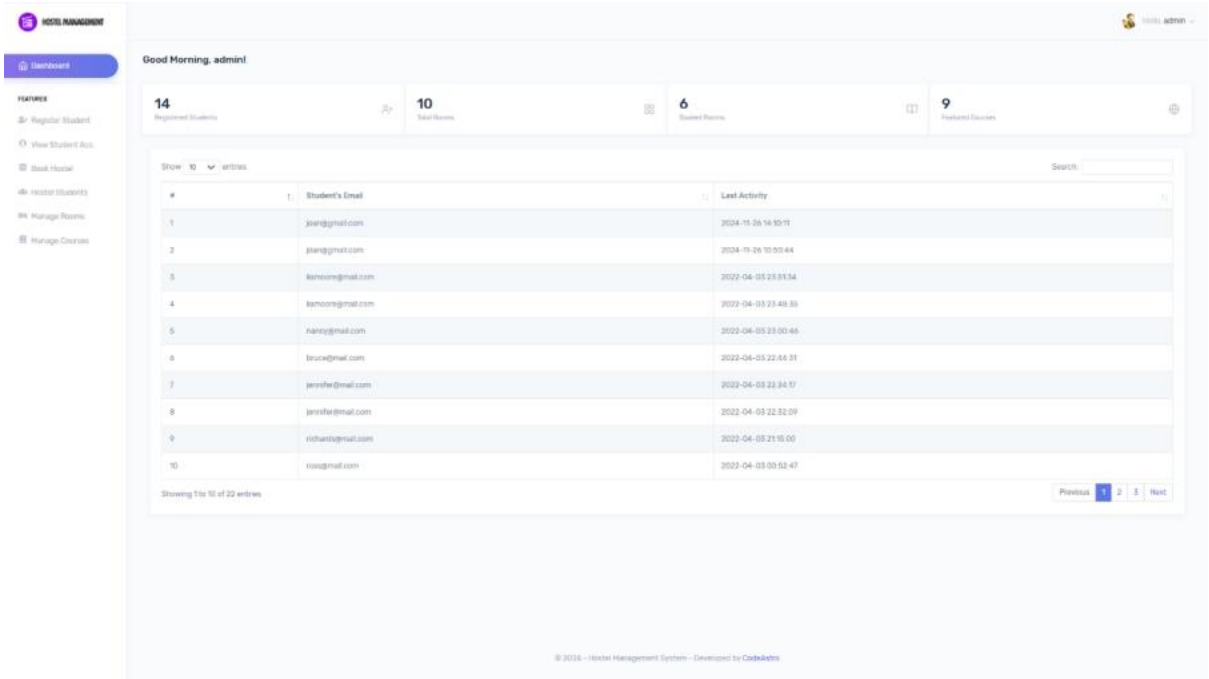


Figure 3.19: Dashboard for Admin Panel

Figure 2.19 shows the dashboard for admin panel consist of five sections: Registered Student, Total Rooms, Booked Rooms, Featured Courses and user access log. It is similar to the student panel dashboard, with the additional section of access logs for each user.

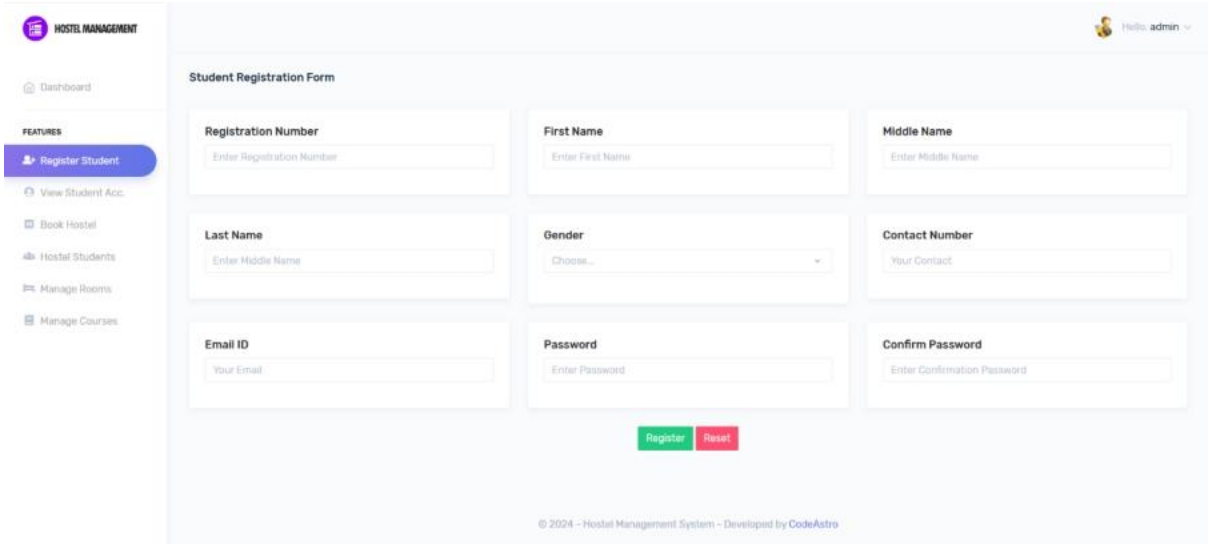


Figure 3.20: Student registration form for Admin Panel

Figure 2.20 shows the student registration feature for admin panel. Admin fills in the related information for student registration.

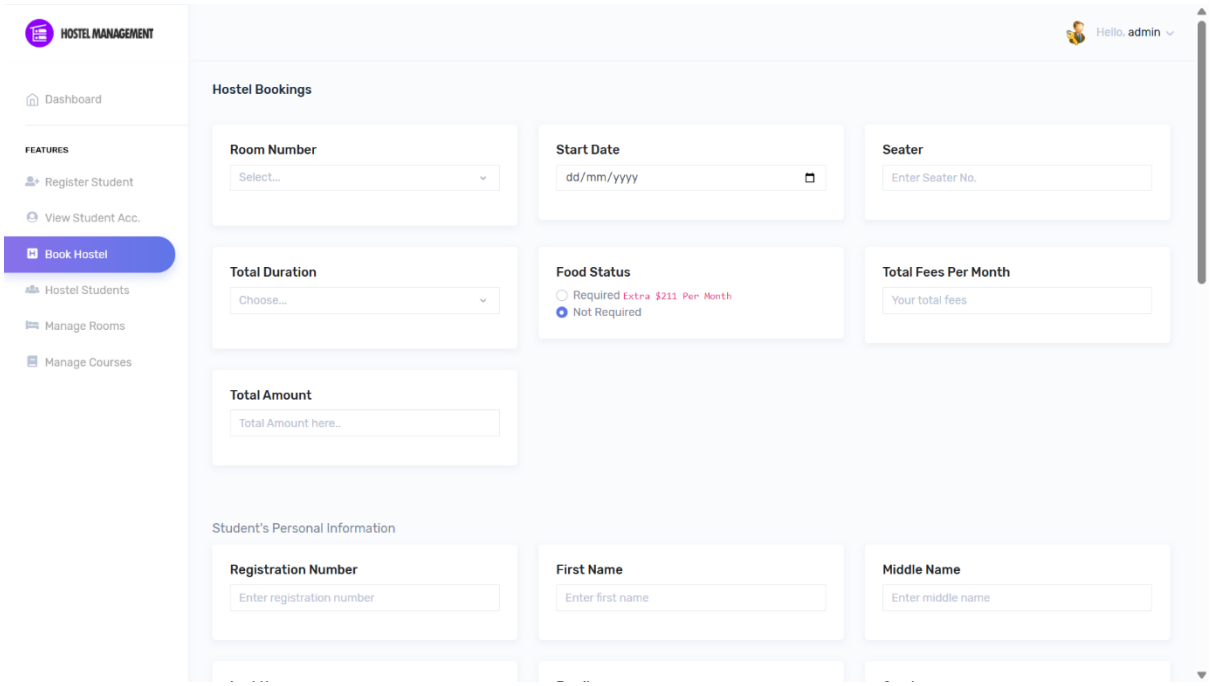


Figure 3.21: Hostel Booking for Admin Panel

Figure 2.21 shows the hostel booking feature for admin panel. Admin can allocate a room for student by fill in related information.

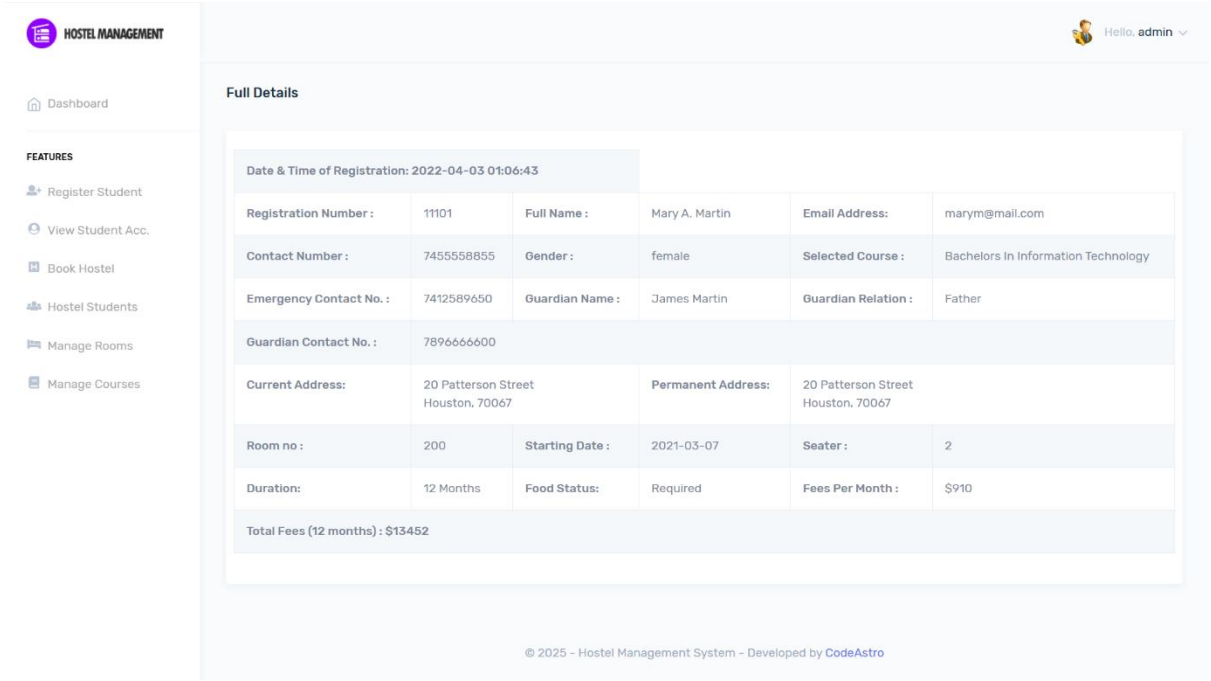


Figure 3.22: Hostel Student Information for Admin panel

Figure 2.22 shows the information of hostel student for admin panel. This section provides both room and resident details.

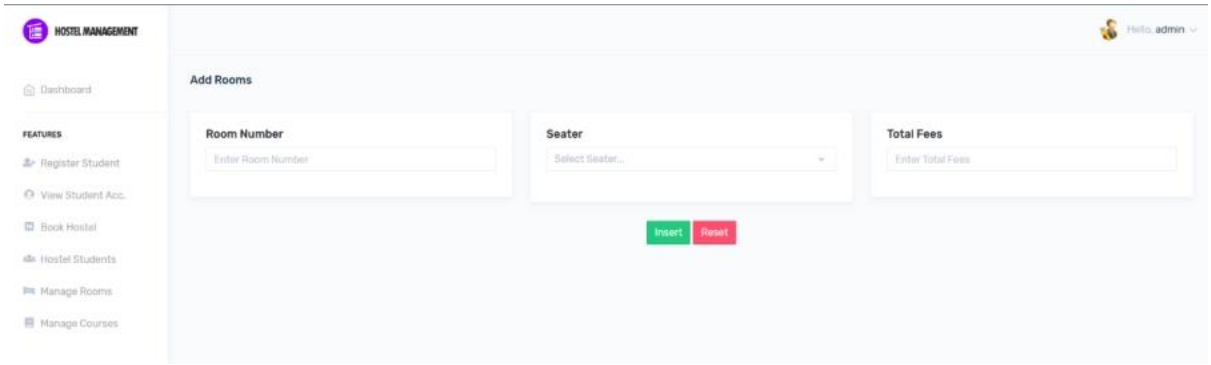


Figure 3.23: Add New Room for Admin Panel

Figure 2.23 shows the add new room feature for admin panel. Admin can add a new room by entering room number, capacity and total fees.

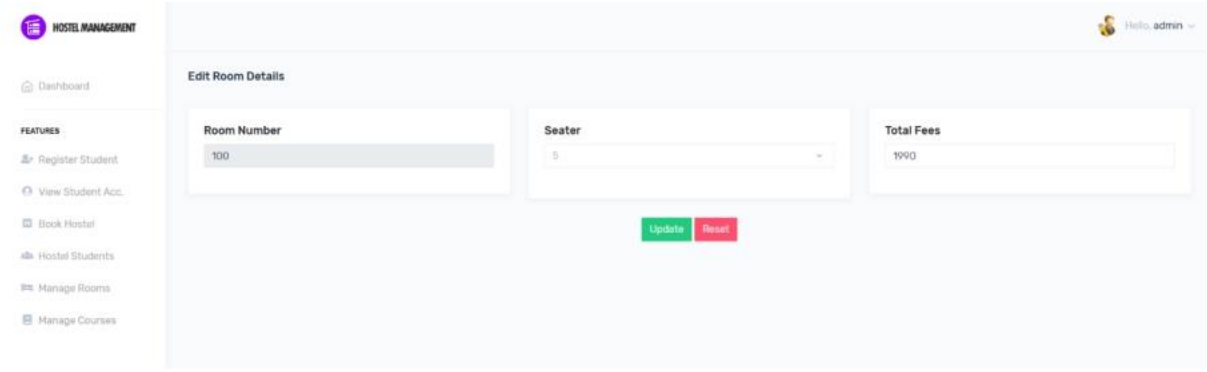


Figure 3.24: Edit room details for Admin Panel

Figure 2.24 shows the edit existing room feature for admin panel. Admin can modify the details of existing rooms.

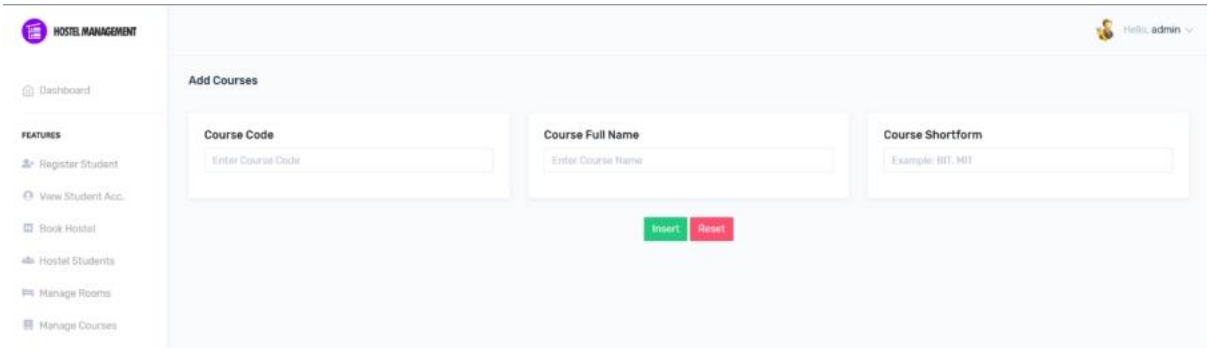


Figure 3.25: Add New Course for Admin Panel

Figure 2.25 shows the add new course for admin panel. Admin can add new course by entering course code, course full name and course short form.

2.2.4 UCSM Hostel Allocation System

Hostel Allocation System is a system that dedicated to the USCM hostel community to handle the hostel room allocation and management. This system is divided into two panels: the Student Panel and the Admin Panel**.** To ensure that users are verified students at the institute, the admin must first register the student's information. Once this is completed, students can sign up for an account and log in to the system. Students have limited functionalities, including

updating their email and phone number, making payments, and changing their account password. The Admin Panel offers a wide range of features, including adding student information, accepting or declining student applications, managing non-allocated students, allocating rooms, searching for students, and viewing student and fee information. Admins can also manage hostel operations through features such as attendance tracking, adding and viewing rooms, and handling gift cards. Room allocation is done manually by the admin on a one-to-one basis, requiring the student number to be entered for each assignment. Admins can add and view student records, decide whether to approve or reject a student's hostel application, and allocate rooms accordingly. Attendance can be tracked by marking students as present using the attendance feature. Additionally, the admin can view detailed information about hostel students and their payment records.

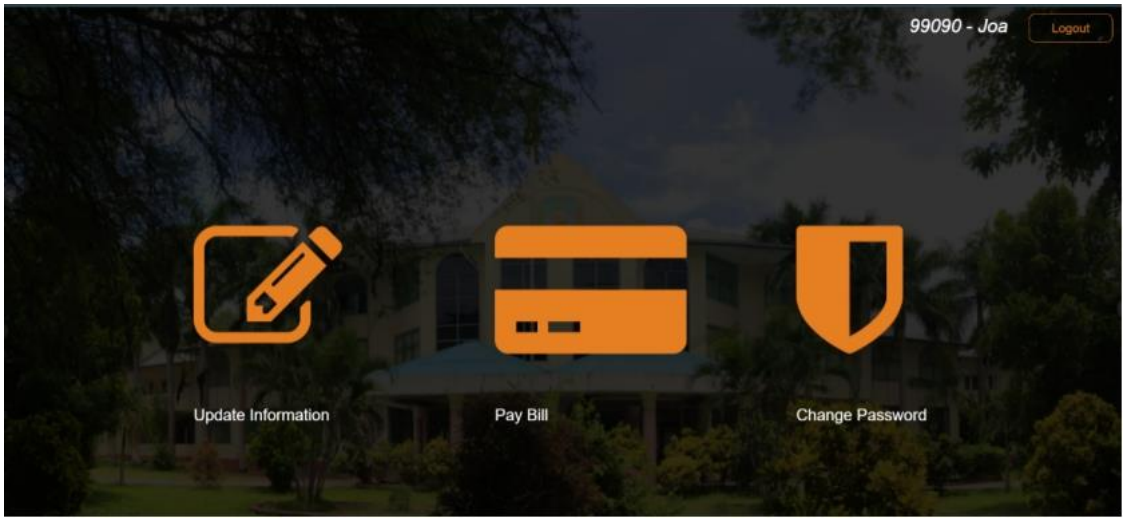


Figure 3.26: Dashboard for Student Panel

Figure 2.26 shows the dashboard for student panel consists of three sections: update information, pay bill and change password. These sections provide quick access for students to navigate the system efficiently.

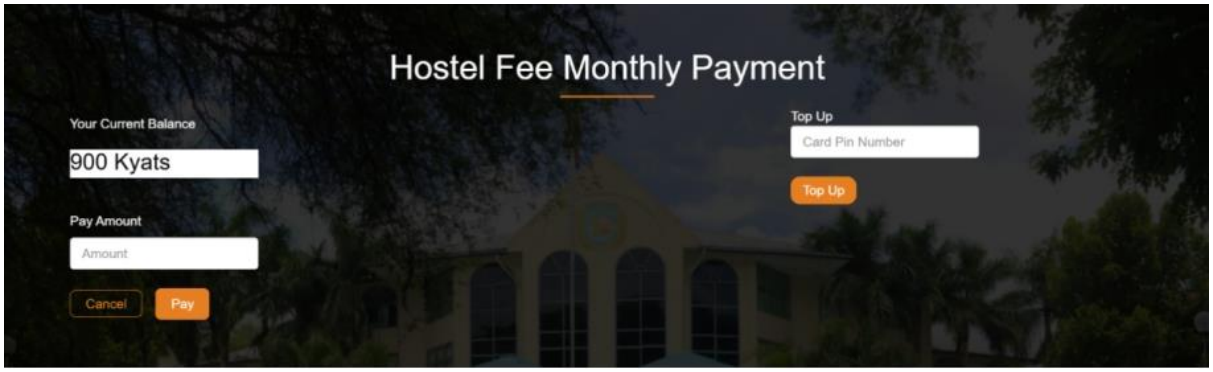


Figure 3.27: Making Payment for Student Panel

Figure 2.27 shows the making payment for hostel fee for student panel. Student can enter the top-up amount and the amount paid.

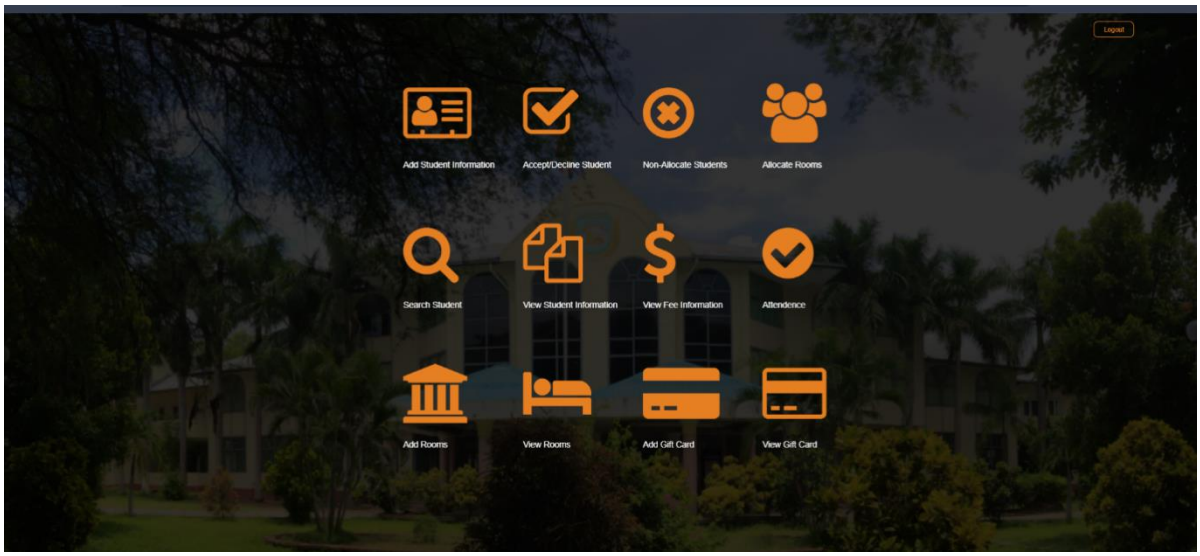


Figure 3.28: Dashboard for Admin Panel

Figure 2.28 shows the dashboard for admin panel consists of 12 sections: Add Student Information, Accept/Decline Student, Non-Allocate Student, Allocate Room, Search Student, View Student Information, View Fee Information, Attendance, Add Room, View Room, Add Gift card and View Gift Card. These sections provide quick access for students to navigate the system efficiently.

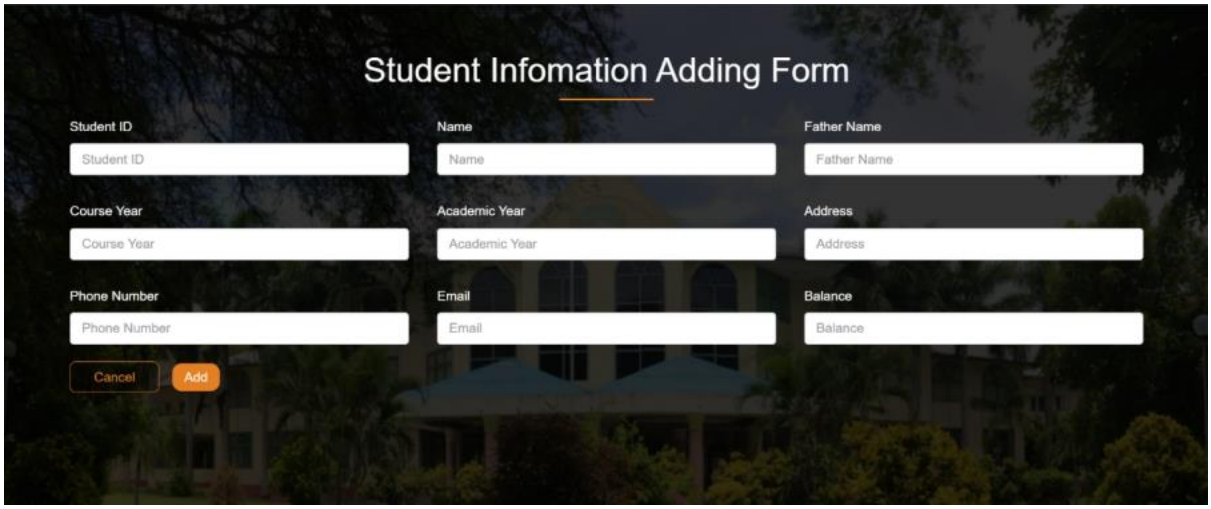


Figure 3.29: Add New Student Information for Admin Panel

Figure 2.29 shows add new student information for admin panel. Admin can enter the student information into the system.

Figure 3.30: Accept or Decline Student for Admin Panel

Figure 2.30 shows the accept or decline student hostel application for admin panel. Admin can accept or decline the applications by clicking the corresponding buttons.

Figure 3.31: Room Allocation for Admin Panel

Figure 2.31 shows the room allocation for admin panel. Admin allocates the room by enter the student id.

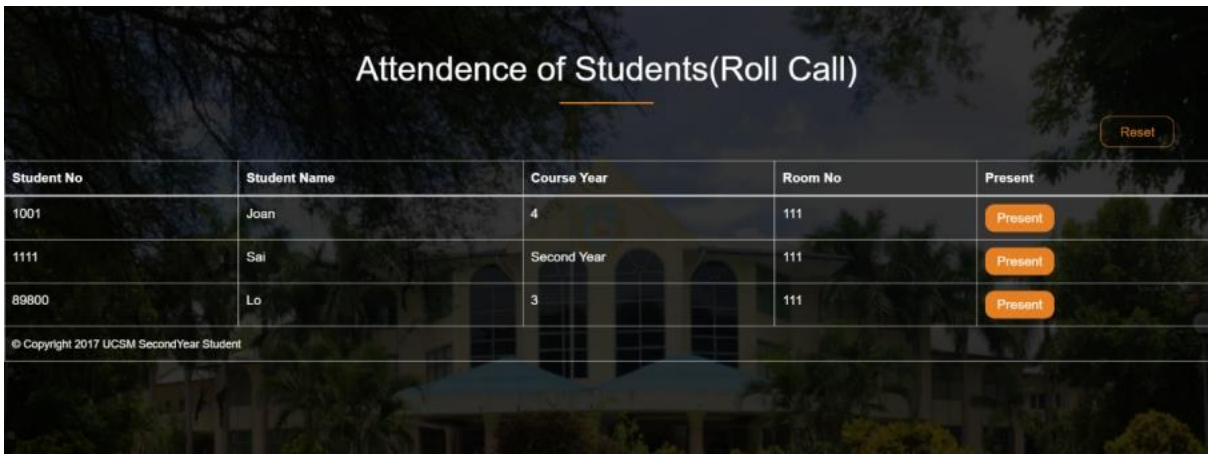


Figure 3.32: Student Attendance for Admin Panel

Figure 2.32 shows the student attendance for admin panel. Admin clicks the Present button when student checks in for the hostel to mark the attendance.

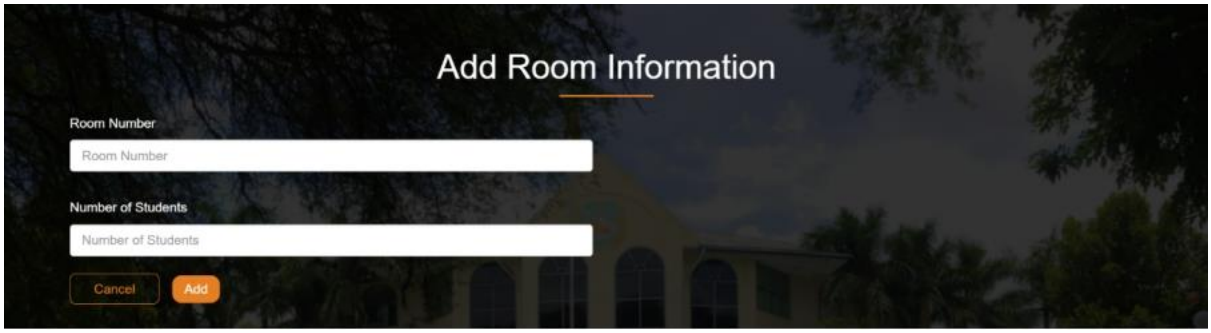


Figure 3.33: Add New Room for Admin Panel

Figure 2.33 shows the add new room feature for admin panel. Admin can create a new room by entering the room number and capacity.

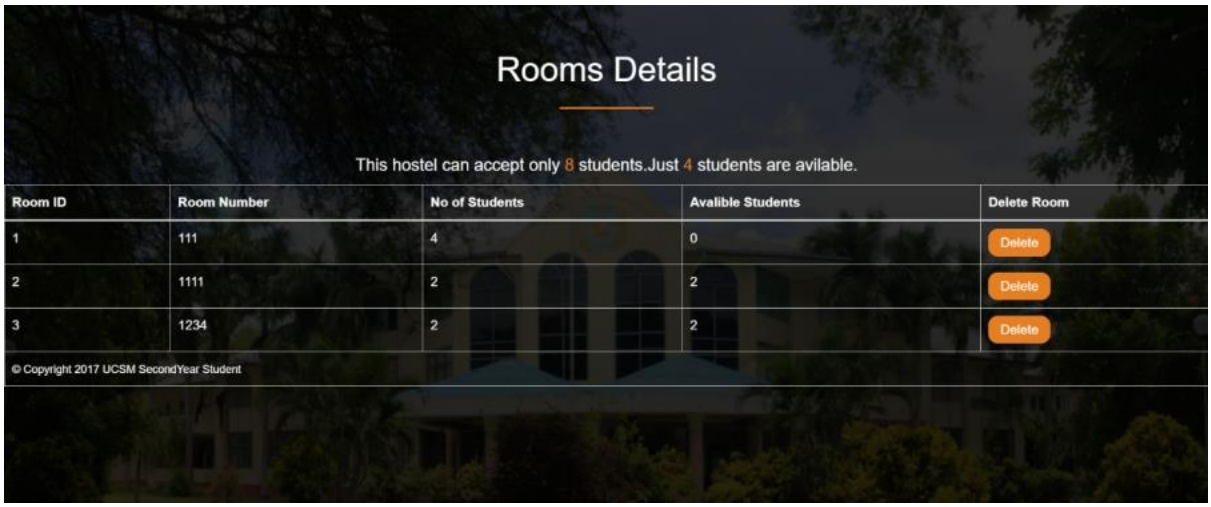


Figure 3.34: Room Details for Admin Panel

Figure 2.34 shows the room details for admin panel. Admin can view the details of the rooms in this section.

2.2.5 Comparison between Existing Similar System and Proposed System

Table 3.1: Table of comparison between exiting similar system and proposed system

Features	Hostel Management System	Online Hostel Management System	UCSM Hostel Allocation System	Proposed System

Student Panel and Admin Panel	√	√	√	√
Account register and log in	√	√	√	√
Dashboard	√	√	√	√
Student Information Management	√	√	√	√
Hostel Room Information	√	√	√	√
Course Management	√	√	×	×
Room Allocation	√	√	√	√
Complaint Management	√	×	×	√
Facilities Management	×	×	×	√
Hostel Application management	×	×	√	√
Notice Announcement	×	×	×	√
Making Payment	×	×	√	×
Visitor Tracking Management	×	×	×	√
User Access Log	√	√	×	×

After reviewing the existing system, it was determined that two separate panels are required: one for students and another for administrators. This distinction is necessary because the system's target users, the hostel community, consist of students and staff, each with different access needs. Account registration and login will be mandatory to ensure user authentication

and data security. The proposed system will include a dashboard to provide an overview of key information and quick access to essential features. Since the system is designed to handle hostel-related tasks, it will include features for managing student information and hostel room details. Several issues related to hostel facilities, such as electricity, cleanliness, plumbing, and more, highlight the need for a complaint management feature. This will enable users to report and track complaints efficiently. Room allocation, one of the primary tasks of hostel management, will also be a core feature. Considering the large number of students and their varying room preferences, room allocation can be challenging and time-consuming. While similar systems allocate rooms manually, the proposed system will implement a Genetic Algorithm (GA)-based room allocation feature to assign rooms in few steps, improving efficiency and accuracy. The existing system offers a course management feature, but this is unnecessary for hostel management. Course information can be incorporated into student profiles, so the proposed system will not include a separate course management feature. The system will also not include a feature of making payment for hostel fee. However, to maintain a well-functioning hostel environment, the proposed system will include a facility management feature that was missing in the existing system. To improve communication, the proposed system will include a notice announcement feature, allowing the hostel community to stay updated with important notices. Additionally, a visitor management feature will be introduced to record and track visitors, ensuring the safety of hostel residents. To enhance convenience, the system will provide an application management feature, enabling students to apply for hostel accommodations and administrators to manage applications effectively. Finally, unlike the existing system, the proposed system excludes the user access log feature, as it was considered unnecessary for the current scope.

2.3 Review of Genetic Algorithm

Genetic algorithms (GA) are a subset of the broader category of evolutionary algorithms (EA), which solve optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. GA, introduced by J. Holland in the 1970s, are search and optimization algorithms based on the principles of natural selection and genetics, drawing inspiration from the biological evolution of living organisms. These algorithms represent the problem space as a population of individuals and iteratively explore the fittest individual by generating successive generations. Each individual in the population corresponds to a potential solution to the problem (Haldurai, Madhubala, & Rajalakshmi, 2016).

The process of GA begins with the generation of an initial population of randomly created chromosomes (individuals). These individuals are evaluated using a fitness function, which quantitatively measures their suitability in solving the given problem. The algorithm then selects the most fit individuals to act as parents, combining their genetic information through a process called crossover to produce offspring. To maintain genetic diversity and avoid premature convergence, random changes or mutations, are applied to the offspring. The newly generated individuals replace part or all the current population to form the next generation, and this cycle repeats until the termination criteria, such as a predefined number of generations or achieving a desired fitness level are met. Ultimately, the best individual in the final population is considered the solution to the problem (Baipai & Kumar, 2010).

The core operations of GA include selection, crossover, and mutation. Selection involves choosing individuals for reproduction based on their fitness, ensuring that more fit individuals have a higher probability of passing their genes to the next generation. Crossover is the process of combining the genetic material of two parents to create offspring that inherit traits from both, introducing variability into the population. Mutation involves random

alterations to certain genes with a specific probability, which helps preserve genetic diversity and prevents the algorithm from getting stuck in local maxima (Yadav & Prajapati, 2012).

Optimization is the science of finding decisions that satisfy given constraints and meet a specific goal at its optimal value. The objective of global optimization is to the find the best possible solution and the classical optimization techniques have difficult in dealing with global optimization as they can easily trap in local minima. GA solves optimization problems by mimicking the principle of biological evolution and due to its random nature, there has high chances to find the global solution (Baipai & Kumar, 2010). Thus, GA is efficient in finding global optimal solutions and be used in the process of room allocation.

GA is highly versatile can be applied to a wide range of optimization problems. Room allocation is an optimization problem as there are many ways to allocate the rooms and finding the best ways which is best solution obtained through GA is desirable. GA provides an optimal solution that enhance the room allocation process by allocate the room to the student that best matches their preference. By applying GA to the room allocation process, the workload of the hotel staff can be reduced while satisfying student preference.

2.4 Review of technologies and tools used

2.4.1 Laravel

Laravel is a web application PHP framework with expressive and elegant syntax. It follows the model-view-controller (MVC) design pattern and offers a rich set of functionalities, incorporating features from PHP frameworks like CodeIgniter and Yii, as well as concepts from other programming languages like Ruby on Rails. Laravel has several advantages, such as scalability, allowing web applications to grow efficiently. It saves considerable time during the design process and includes namespaces and interfaces to organize and manage resources

effectively. Artisan, Laravel's command-line interface, provides a set of commands that assist in building web applications. Laravel also offers numerous key features that are highly beneficial for web application development, including modularity, testability, routing, configuration management, query builder and ORM, and authentication. Laravel's built-in libraries and modules, integrated with the Composer dependency manager, enhance application development. Laravel emphasizes testability, supporting the creation and execution of test cases. It provides a flexible approach to defining routes and a consistent method for managing configurations across different environments. Authentication, a fundamental requirement for web applications, is simplified with Laravel's tools, enabling developers to implement secure and efficient authentication systems quickly (Laravel - Overview, n.d.). One of Laravel's most powerful features is its Eloquent ORM (Object-Relational Mapping), which simplifies database interactions. Eloquent is model-driven, meaning each database table corresponds to a specific model. It offers a fluent and expressive query builder, making it easy for developers to construct complex database queries with simple and readable syntax (Shukla, 2023). In summary, Laravel is a powerful framework that empowers developers to create highly performant and comprehensive web applications with ease.

2.4.2 Visual Studio Code

Visual Studio (VS) Code is a free code editor that supports programming in various languages. While primarily a lightweight editor, it provides features comparable to a complete Integrated Development Environment (IDE). VS Code integrates seamlessly with version control systems like Git, allowing developers to manage code repositories efficiently (Faulds, n.d.). VS Code offers syntax highlighting and extends functionality with IntelliSense, which provides code completion, code hinting, and parameter information to enhance productivity. It also includes a robust debugging feature, enabling developers to attach to running applications

and debug using breakp1oints. The extension system in VS Code allows developers to browse, install, and use a wide variety of extensions to customize their development environment. Additionally, the integrated terminal enables developers to perform tasks directly within the editor, replicating the functionality of a standalone terminal. Another powerful feature is the Command Palette, which provides quick access to commands and allows developers to execute tasks efficiently (Blumer, 2018). In summary, VS Code is a versatile and user-friendly IDE suitable for developing applications across a wide range of programming languages.

2.4.3 Laragon

Laragon is a development environment designed for local use and is compatible only with Windows. It is portable, isolated, fast, and powerful, providing a universal platform for building and managing web applications based on PHP, Node.js, Python, Java, Go, and Ruby. Laragon operates as an isolated environment with its own service orchestration, independent of Windows services. This design ensures asynchronous and non-blocking service management, allowing applications to run smoothly and efficiently. Laragon is favored for its pretty URLs, portability, isolated environment that keeps the system clean, and ease of operation. Its modern and powerful features make it an excellent choice for local development (Documentation, n.d.).

2.5 Summary

A literature review is crucial as it summarizes existing knowledge, identifies gaps in the literature, and provides direction for future studies to facilitate improvement. By reviewing current approaches, whether system-based or manual methods, it becomes possible to determine which components are essential to retain in the proposed system and which areas require enhancement. Careful evaluation of the knowledge related to the use of Genetic

Algorithms (GA) is necessary to fully understand their application and decide if they are suitable for the system. Additionally, reviewing the technological tools is vital to ensure the system is developed effectively.

CHAPTER 3: REQUIREMENTS ANALYSIS AND DESIGN

3.1 Introduction

In this chapter, the requirements for the proposed hostel management system will be collected and analysed. Once the requirements are defined, the design phase will follow. The requirements are gathered through a questionnaire, and the design will encompass both logical and physical design elements.

3.2 Methodology

The methodology used in the Waterfall methodology, which consists of five phases: requirement, design, implementation, testing and deployment and maintenance. The Waterfall methodology is chosen because it provides a clear project structure (Atlassian, 2015). By defining the requirements at the early stage, it offers valuable insights that help in creating a better design, leading to more effective implementation. The Waterfall methodology dictates that the phases are carried out in sequential order, ensuring that clearly defined requirements and a well-thought-out design simplify the implementation process. Therefore, requirement analysis is crucial, and the design must be tailored to meet those requirements.

3.3 Requirement Analysis

The requirements of the proposed system are collected through questionnaire. The questionnaire is presented in google form and the respondents is the UNIMAS students. Total of 30 respondents have answered the questionnaire.

3.3.1 Questionnaire Result

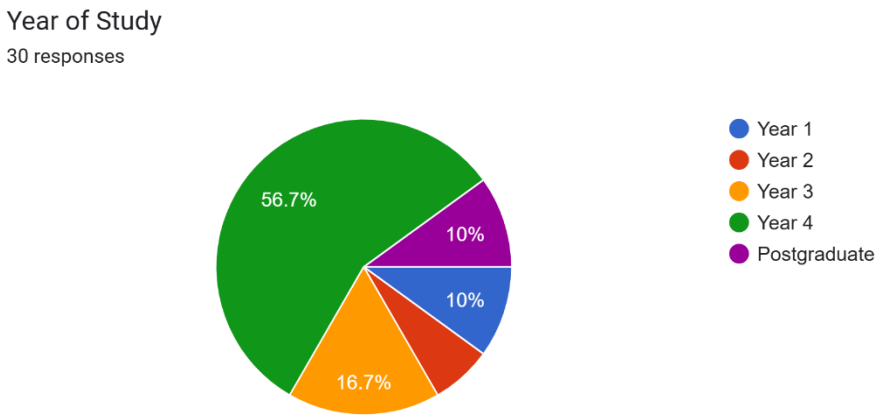


Figure 4.1: Pie chart of year of study of respondents

Figure 3.1 shows the pie chart of the year of study of respondents. Based on the chart in figure 41, it shows 17 respondents from Year 4, 5 from Year 3, 2 from Year 2 and 3 each from Year 1 and postgraduate studies.

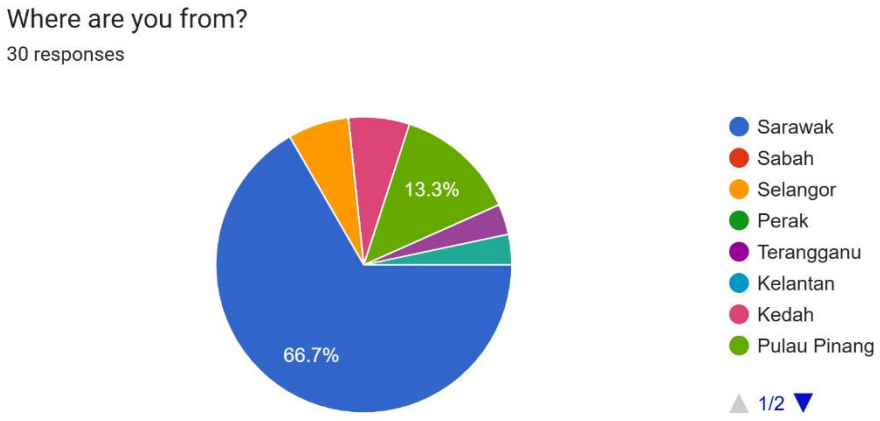


Figure 4.2: Pie chart of where are you from of respondents

Figure 3.2 shows the pie chary of where are you from of respondents. Based on the chart in figure 42, most respondents, which are 20 respondents are from Sarawak, followed by

4 from Pulau Pinang, 2 each from Selangor and Kedah, and 1 each from Johor and Wilayah Persekutuan Kuala Lumpur.

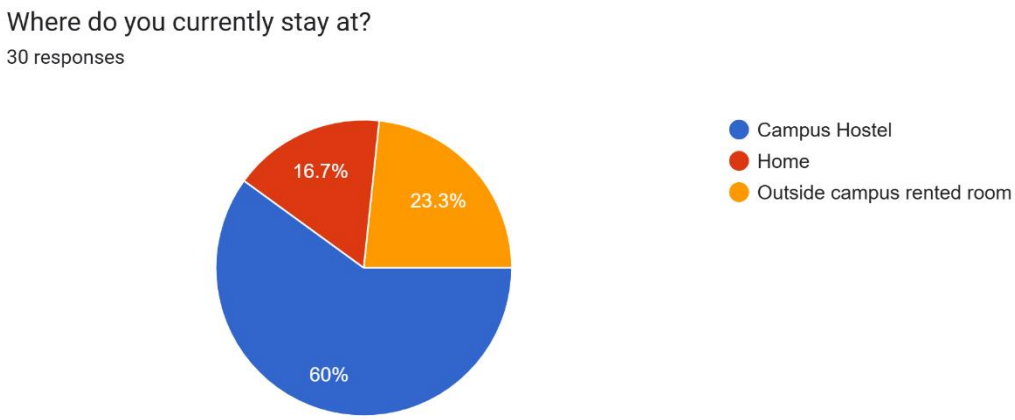


Figure 4.3: Pie chart of where you currently stay at of respondents

Figure 3.3 shows the pie chart of where you currently stay at of respondents. Based on the chart in figure 43, 18 respondents currently stay in campus hostels, 7 in rented rooms outside campus, and 5 at home.

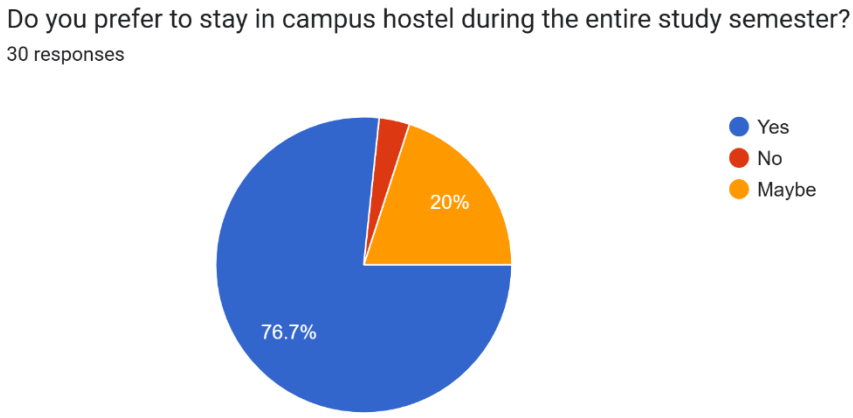


Figure 4.4: Pie chart of result of do you prefer to stay in campus hostel during the entire study semester of respondents

Figure 3.4 show the pie chart of result of do you prefer to stay in campus hostel during the entire study semester of respondents. Based on the chart in figure 44, 23 respondents prefer to stay on campus during the entire study semester, 6 are unsure, and 1 prefers not to. Most respondents prefer staying on campus due to its convenience and proximity to lectures.

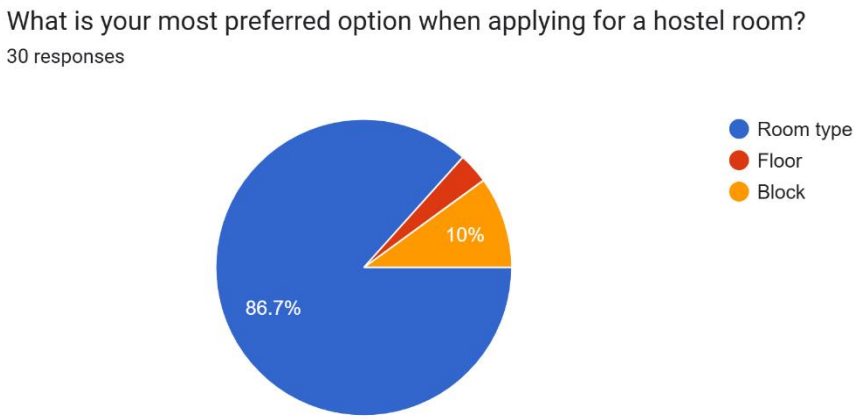


Figure 4.5: Pie chart of result of what is your most prefer option when applying for a hostel room of respondents

Figure 3.5 shows the pie chart of result of what is your most prefer option when applying for a hostel room of respondents. Based on the chart in figure 45, when applying for a hostel, only 1 respondent prioritizes floor selection, 3 prefer block, while the rest prefer room type. Most respondents prefer certain room types, either for staying alone or with friends.

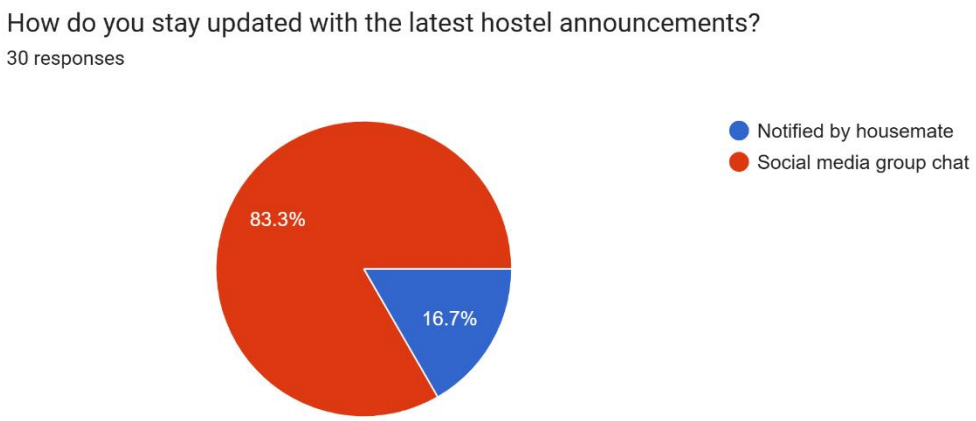


Figure 4.6: Pie chart of result of how do you stay updated with the latest hostel announcement of respondents

Figure 3.6 shows a pie chart of result of how do you stay updated with the latest hostel announcement of respondents. Based on the chart in figure 46, 5 respondent receives hostel announcements via housemates, while the rest rely on social media group chats. Most respondents prefer using social media group chats for immediate notification, while others prefer being notified by roommates to avoid missing messages.

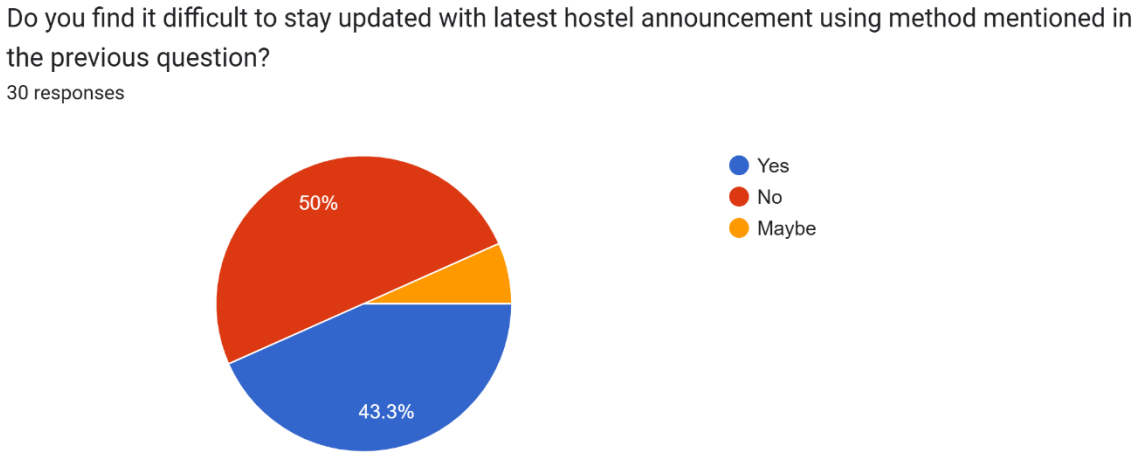


Figure 4.7: Pie chart of results of do you find it difficult to stay updated with latest hostel announcement using method mentioned in the previous question of respondents

Figure 3.7 shows the pie chart of result of do you find it difficult to stay updated with the latest hostel announcement using method mentioned in the previous question of respondents. Based on the chart in figure 47, 15 respondents find it easy to stay updated with hostel announcements through the current method, 13 find it difficult, and 2 are unsure. Several respondents may be overlooked for the messages when posted in the social media group chat.

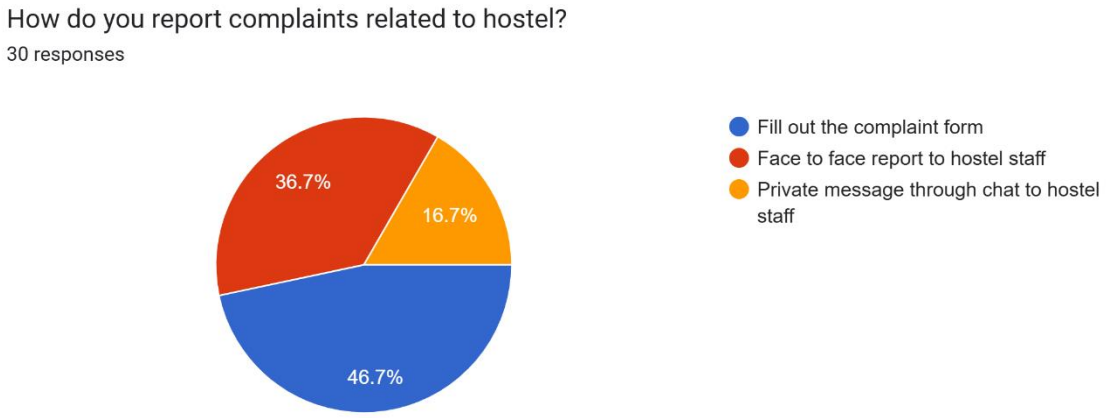


Figure 4.8: Pie chart of results how do you report complaints related to hostel of respondents

Figure 3.8 shows the pie chart of results how do you report complaints related to hostel of respondents. Based on the chart in figure 48, to report complaints, 14 respondents prefer filling out complaint forms, 11 prefer face-to-face reports, and 5 prefer private messages to hostel staff. Respondents choose filling out complaints forms to provide full complaint details, face-to-face reports in emergencies and private messages for emergencies when staff are off-duty.

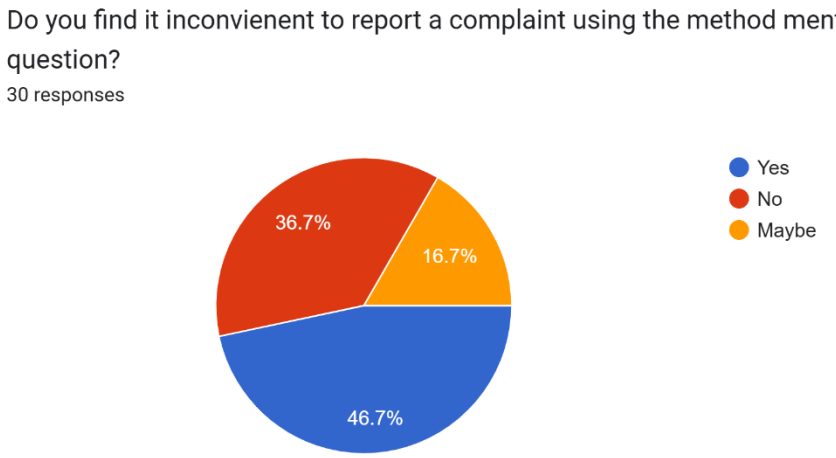


Figure 4.9: Pie chart of results of do you find it inconvenient to report a complaint using method mentioned in the previous question of respondents

Figure 3.9 shows the pie chat of do you find it inconvenient to report a complaint using method mentioned in the previous question of respondents. Based on the chart in figure 49, 14 respondents find the current complaint reporting approach is inconvenient, 11 find it convenient, and 5 are unsure. Several respondents find it inconvenient that they need to physically locate the complaint forms or ask for the staff's contact to report complaint.

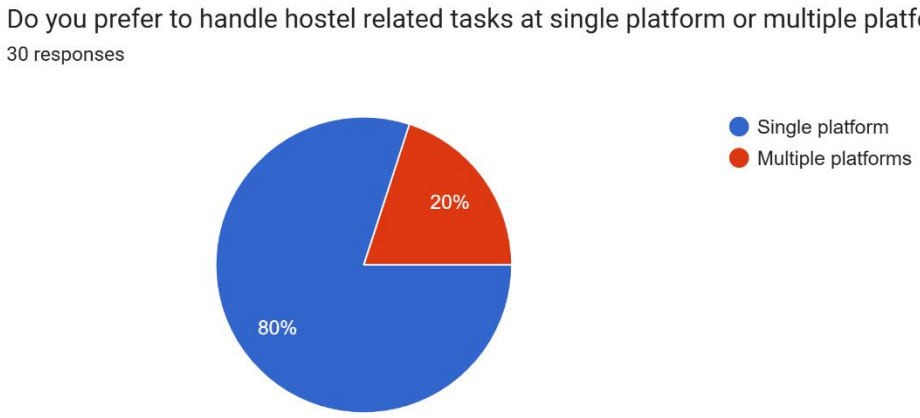


Figure 4.10: Pie chart of results of do you prefer to handle hostel related tasks at single platform or multiple platforms of respondents

Figure 3.10 shows the pie chart of results of do you prefer to handle hostel related tasks at single platform or multiple platforms of respondents. Based on the chart in figure 50, 24 respondents prefer handling hostel tasks on a single platform, while 6 prefer multiple platforms. Most respondents prefer a single platform over multiple platforms due to its convenience.

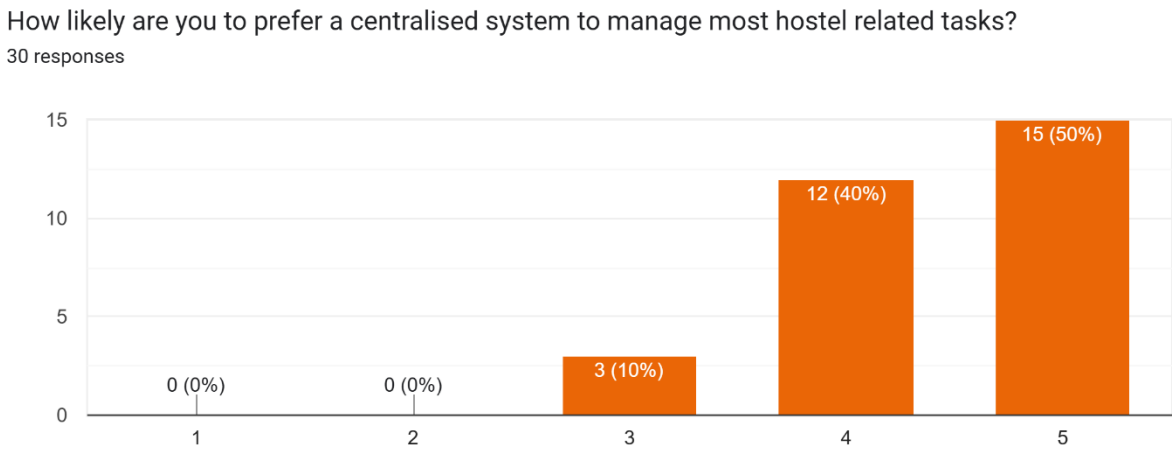


Figure 4.11: Bar chart of results of how likely are you to prefer a centralise system to manage most of the hostel related tasks of respondents

Figure 3.11 shows the bar chart of results of how likely are you to prefer a centralise system to manage most of the hostel related tasks of respondents. Based on the chart in figure 51, 50% of respondents strongly prefer a centralized system for managing hostel tasks, 40% prefer it and 10% are neutral. More respondents prefer a centralized system to manage hostel tasks, as single platform is more convenient than using multiple platforms.

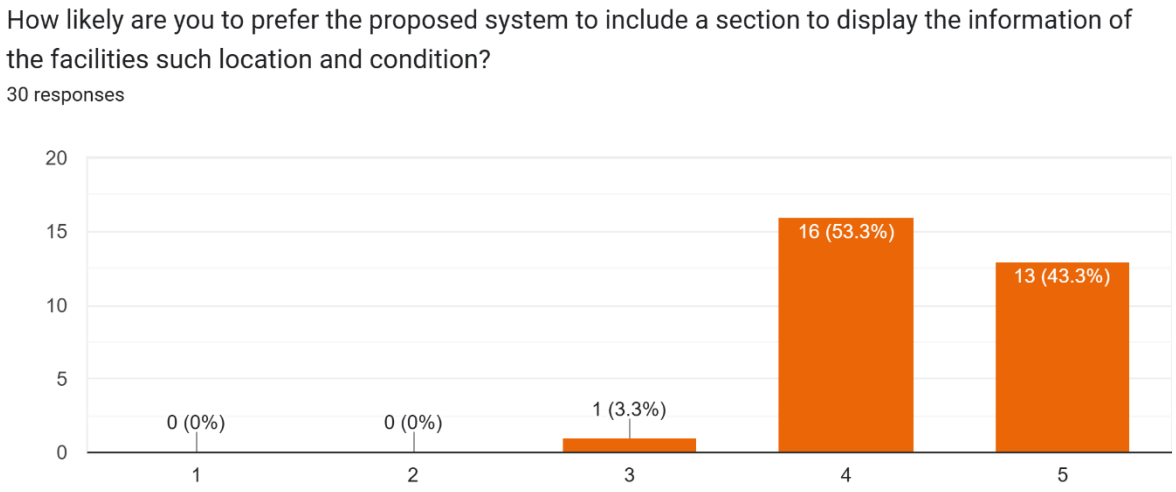


Figure 4.12: Bar chart of results of how likely are you to prefer the proposed system to include a section to display the information of the facilities such as location and condition

Figure 3.12 show the bar chart of results of how likely are you to prefer the proposed system to include a section to display the information of the facilities such as location and condition. Based on the chart in figure 52, 43.3% strongly prefer a section in the proposed system to display hostel facilities information, 53.3% prefer it and 3.3% are neutral. Most respondents prefer including a facility section in the system, as it allows users to access facility information, including location and condition, when needed.

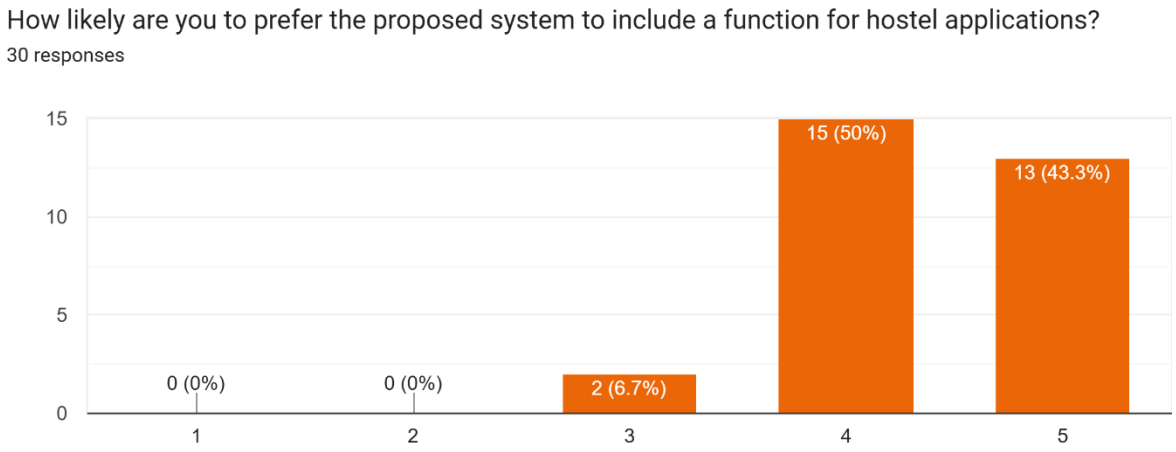


Figure 4.13: Bar chart of how likely are you to prefer the proposed system to include a function for hostel application of respondents

Figure 3.13 shows the bar chart of how likely are you to prefer the proposed system to include a function for hostel application of respondents. Based on the chart in figure 53, 43.3% strongly prefer a hostel application function, 50% prefer it, and 6.7% are neutral. Most respondents prefer having a hostel application in the system for easier handling of hostel-related tasks. This way, students can apply for the hostel more easily without having to access different platforms.

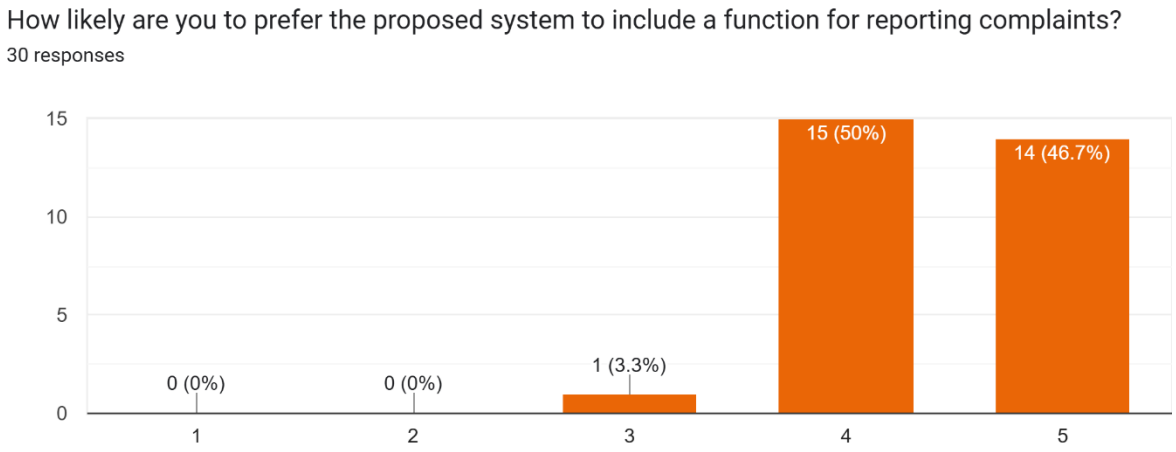


Figure 4.14: Bar chart of results of how likely are you to prefer the proposed system to include a function for reporting complaints of respondents

Figure 3.14 shows the bar chart of results of how likely are you to prefer the proposed system to include a function for reporting complaints of respondents. Based on the chart in figure 54, 46.7% strongly prefer a complaint reporting function, 50% prefer it and 3.3% are neutral. Most respondents prefer a complaint function, as it allows students to file complaints more conveniently, compared to the current manual method.

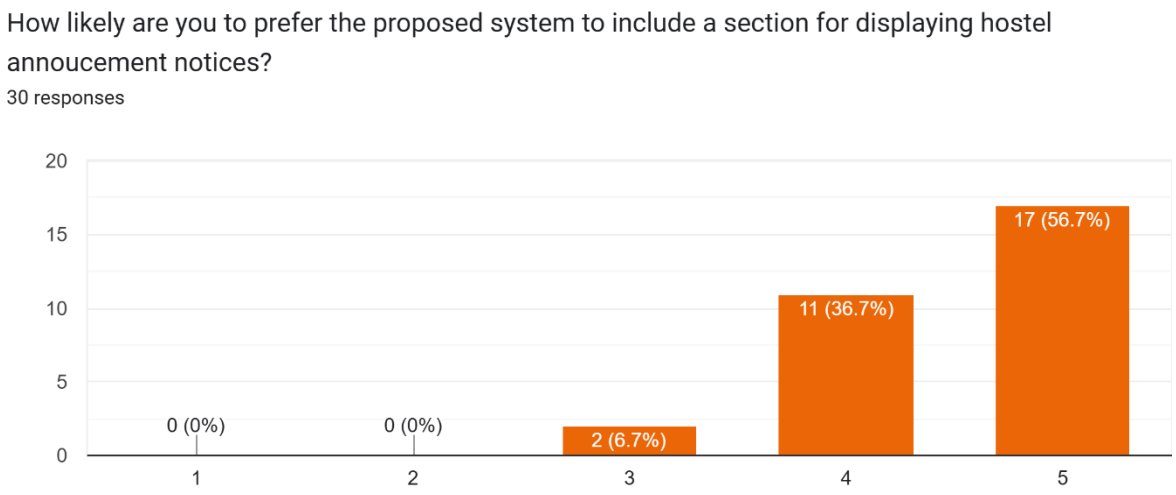


Figure 4.15: Bar chart of results of how likely are you to prefer the proposed system to include a section to display the hostel announcement notice of respondents

Figure 3.15 shows a bar chart of results of how likely are you to prefer the proposed system to include a section to display the hostel announcement notice of respondents. Based on the chart in figure 55, 56.7% strongly prefer a section for hostel announcement notices, 36.7% prefer it and 6.7% are neutral. Most respondents prefer an announcement section in the system, as they may overlook announcements on social media or physical notice boards. This way, they can easily check for updates.

3.3.2 Result Analysis

Based in the responses from the questionnaire, most respondents indicated a preference for staying in the campus hostel. Among the key considerations, the room type emerged as the most preferred option when applying for a hostel. This highlights the need to develop a hostel management system to better handle hostel-related tasks, giving room type a higher weightage in the room allocation process.

Additionally, several respondents reported challenges with the current hostel management approach. They expressed difficulty in staying updated with hostel

announcements and reporting complaints. They preferred a centralized platform where most hostel-related tasks could be managed conveniently.

From the system preference section of the questionnaire, the respondents provided insights into the functionalities they value in the proposed system. Most respondents favoured a centralized platform to manage hostel-related tasks, citing its convenience. They also expressed a preference for the system to include a feature displaying information about hostel facilities, such as their condition and location, which would save time and improve accessibility.

Respondents highly supported the inclusion of functionalities for hostel applications and complaint reporting. They noted that the current process for applying for hostels and reporting complaints is cumbersome, often requiring access to separate links or external channels. Integrating these functionalities into a single platform would eliminate the inconvenience and streamline the process.

Another widely preferred feature was a dedicated section for hostel announcements. Currently, most announcements are shared in social media group chats, where they can be easily overlooked, leading to missed information. A dedicated announcement section in the proposed system would ensure that residents can quickly and reliably access important updates.

In conclusion, the responses indicate that essential functionalities for the proposed hostel management system include hostel applications, complaint reporting, announcement viewing, and facilities information. By integrating these features into a centralized platform, the system will provide a more efficient and convenient way for the hostel community to manage their tasks.

3.4 Design

The design phase starts after the requirements have been collected and defined. The design of the proposed system can be categorized into two types: logical design and physical design. The logical design includes a use case diagram, use case descriptions, activity diagrams, and an entity relationship diagram. Meanwhile, the physical design will focus on creating the wireframe.

3.4.1 Logical Design

The logical design includes a use case diagram with detailed descriptions for each use case. Additionally, two activity diagrams are created to illustrate the process flow for hostel applications and complaint reporting. The system database design is represented using an entity relationship diagram. The term ‘admin’ also refers to ‘staff’.

3.4.1.1 Use Case Diagram

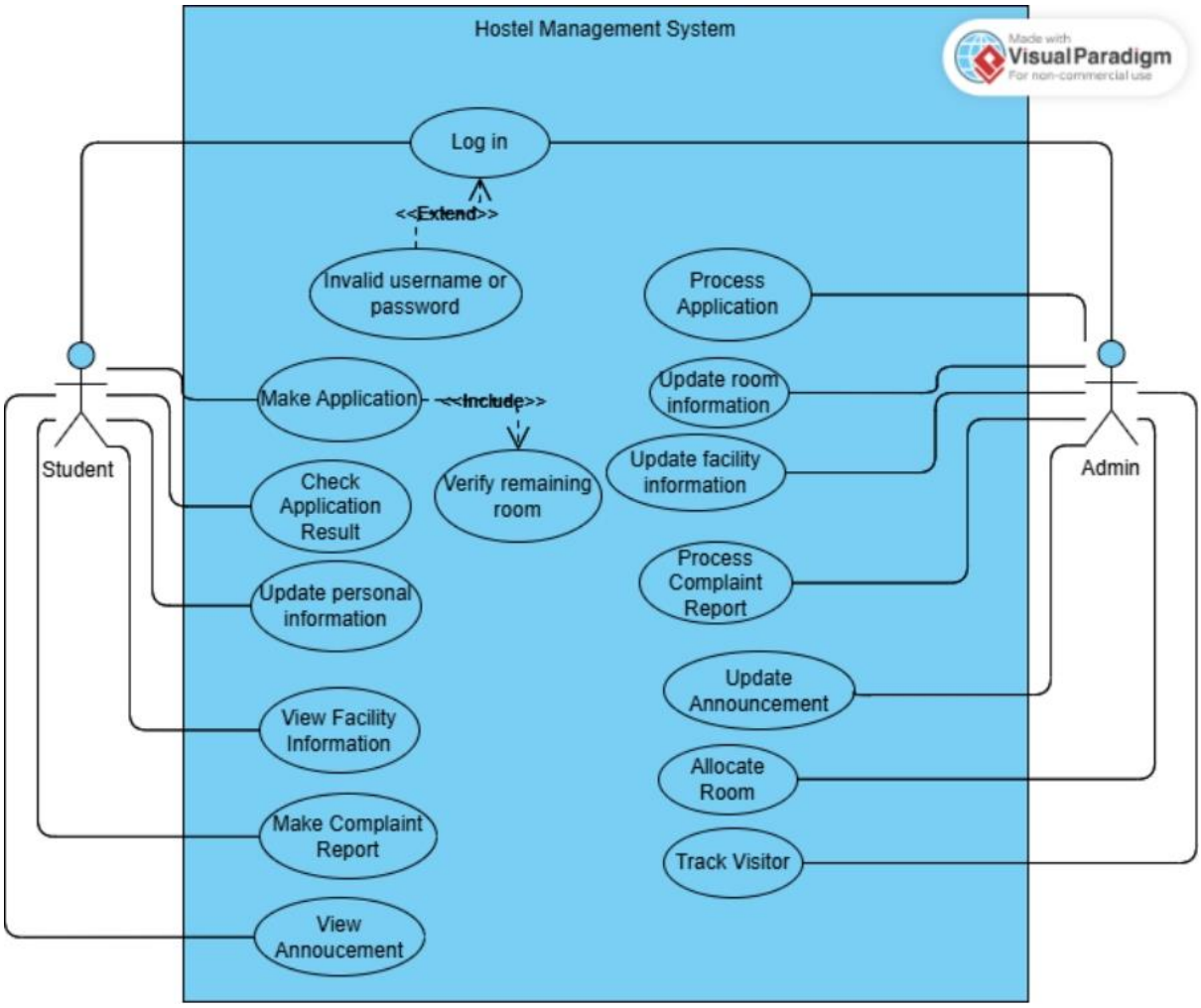


Figure 4.16: Use Case Diagram for proposed hostel management system

Figure 3.16 shows the use case diagram of the proposed hostel management system. There are two actors: student and admin. The functions that each actor can access are illustrated in the use case diagram.

3.4.1.2 Use Case Description

Table 4.1: Use case description for log in use case

Use Case Name	Log in
Actor(s)	Student, Admin

Summary Description	The actor logs in to the system by enter the username and password.
Pre-Conditions/Entry Conditions	The actor had registered an account.
Post-Conditions/Exit Conditions	The actor successfully logged in to the system.
Basic Path/Flow of Events	1. The actor accesses to the log in page. 2. The actor enters the username and password. 3. The system verifies the username and password. 4. The actor successfully logs in to the system. 5. After logging in, the actor will see the dashboard page of the system. 6. The use case ends.
Alternative path (if exist)	2.1 The actor clicks the forget password option. 2.2 The actor enters the email address. 2.3 The actor enters the verification code. 2.4 The actor resets the password. 2.5 The actor logs in to the system by enter the username and new password.
Exceptional path (if exist)	2.1 The actor enters incorrect username or password. 2.2 The system displays the error message to notify the actor. 2.3 The actor re-enters the correct username and password.

Table 3.1 shows the use description for log in. This use case describes how student and staff log in to the system using their registered credentials. It includes both standard login flow and options for password recovery in case of forgotten credentials.

Table 4.2: Use case description for make application use case

Use Case Name	Make Application
Actor(s)	Student
Summary Description	The actor fills the application form to apply for hostel in the system.
Pre-Conditions/Entry Conditions	The actor had logged in to the system and not a hostel resident.
Post-Conditions/Exit Conditions	The actor successfully submitted the application if there have room available. The actor is notified by the system if no room is available and cannot proceed the application.
Basic Path/Flow of Events	1. The actor accesses to the application page. 2. The system checks if there are remaining room available. 3. The actor enters the corresponding application information. 4. The system validates the information. 5. The actor submits the application. 6. The use case ends.
Alternative path (if exist)	2.1 No room is available. 2.2 The system notifies the actor. 2.3 The actor cannot proceed the application. 2.4 The use case ends.

Exceptional path (if exist)	3.1 The actor enters invalid information. 3.2 The system displays the error message to notify the actor. 3.3 The actor re-enters the valid information.

Table 3.2 shows the use case description for make application. This use case outlines how students submit hostel applications. It includes validation of available rooms and user input to ensure that only eligible students can proceed with the application process.

Table 4.3:Use case description for check application status use case

Use Case Name	Check Application Result
Actor(s)	Student
Summary Description	The actor checks the application result and decides to accept or reject the application offer.
Pre-Conditions/Entry Conditions	The actor had submitted the application.
Post-Conditions/Exit Conditions	If the actor accepted the application, the actor gains access to other functionalities. If the actor rejected the application, the actor cannot access to other functionalities.
Basic Path/Flow of Events	1. The actor accesses to the application page. 2. The actor checks the application status. 3. The system displays the application as approved. 4. The actor accepts the application offers. 5. The use case ends.
Alternative path (if exist)	3.1 The system displays the application as rejected and reason.

	3.2 The use case ends. 4.1 The actor rejects the application offers and enters the reject application offer reason. 4.2 The actor submits the reason. 4.3 The use case ends.
Exceptional path (if exist)	N/A

Table 3.3 shows the use case description for check application status. This use case allows students to check the status of their hostel applications and respond to the offer by either accepting or rejecting it. Their decision effects access to other system features.

Table 4.4: Use case description for update personal information use case

Use Case Name	Update personal information
Actor(s)	Student
Summary Description	The actor updates the personal information including check in, check out date and housing information in the system for record purpose.
Pre-Conditions/Entry Conditions	The actor accepted the application offer.
Post-Conditions/Exit Conditions	The information is successfully updated and recorded in the system.
Basic Path/Flow of Events	1. The actor accesses to the personal information page. 2. The actor enters the related information in the system. 3. The actor updates the information by clicking updates button.

	4. The information is successfully updated and recorded. 5. The use case ends.
Alternative path (if exist)	N/A
Exceptional path (if exist)	2.1 The actor enters invalid information. 2.2 The system displays error message to notify the actor the enter valid information. 2.3 The actor re-enters valid information.

Table 3.4 shows the use case description for update personal information. This use case covers how residents update personal details such as check-in or check-out and housing information for record-keeping and system tracking.

Table 4.5: Use case description of make complaint report use case

Use Case Name	Make Complaint Report
Actor(s)	Student
Summary Description	The actor fills the complaint form to report a complaint related to the hostel.
Pre-Conditions/Entry Conditions	The actor accepted the application offer.
Post-Conditions/Exit Conditions	The actor successfully submitted the complaint report.
Basic Path/Flow of Events	1. The actor accesses to the make complaint page. 2. The actor fills the complaint form. 3. The actor submits the complaint form. 4. The use case ends.
Alternative path (if exist)	N/A
Exceptional path (if exist)	2.1 The actor enters invalid information.

	2.2 The system displays the error message to notify the actor. 2.3 The actor re-enters the valid information.

Table 3.5 shows the use case description for make complaint report. This use case enables residents to submit complaints related to hostel facilities or services. It ensures complaints are properly recorded and sent to staff foe action.

Table 4.6: Use case description for view facility information use case

Use Case Name	View Facility Information
Actor(s)	Student
Summary Description	The actor views the information related to the facilities around the hostel.
Pre-Conditions/Entry Conditions	The actor had accepted the application offer and become the resident.
Post-Conditions/Exit Conditions	N/A
Basic Path/Flow of Events	1. The actor accesses to hostel facilities page. 2. The actor views the information of the facilities. 3. The use case ends.
Alternative path (if exist)	N/A
Exceptional path (if exist)	N/A

Table 3.6 shows the use case description for view facility information. This use case allows residents to view information about the available hostel facilities, such as their location and status to support daily activities and planning.

Table 4.7: Use case description of view announcement use case

Use Case Name	View Announcement
Actor(s)	Student
Summary Description	The actor views the announcement to stay updated with the hostel notices.
Pre-Conditions/Entry Conditions	The actor had accepted the application offer and become the resident.
Post-Conditions/Exit Conditions	N/A
Basic Path/Flow of Events	1. The actor accesses to the announcement page. 2. The actor views the announcements. 3. The use case ends.
Alternative path (if exist)	N/A
Exceptional path (if exist)	N/A

Table 3.7 shows the use case description for view announcement. This use case allows residents to view hostel-related announcements and updates. It ensures residents stay informed of important events or changes.

Table 4.8: Use case description of process application use case

Use Case Name	Process Application
Actor(s)	Admin
Summary Description	The actor processes the application to approve or reject the application.
Pre-Conditions/Entry Conditions	The student has submitted the application.
Post-Conditions/Exit Conditions	The application had been processed and the application status is updated.
Basic Path/Flow of Events	1. The actor accesses to the application management page. 2. The actor reviews the application.

	3. The actor approves the application. 4. The system updates the application status. 5. The use case ends.
Alternative path (if exist)	3.1 The actor rejects the application. 3.2 The actor replies with the reason of rejects application.
Exceptional path (if exist)	N/A

Table 3.8 shows the use case description for process applications. This use case explains how staff review and process student applications by approving or rejecting them, along with providing reasons for rejection if necessary.

Table 4.9: Use case description of update room information use case

Use Case Name	Update room information
Actor(s)	Admin
Summary Description	The actor updates the information of the hostel room in the system.
Pre-Conditions/Entry Conditions	The actor had log in to the system.
Post-Conditions/Exit Conditions	The information of the hostel room is updated and recorded in the system.
Basic Path/Flow of Events	6. The actor accesses to the room management page. 7. The actor enters the room information. 8. The actor clicks the save button to updates and saves the room information in the system. 9. The use case ends.
Alternative path (if exist)	N/A
Exceptional path (if exist)	N/A

Table 3.9 shows the use case description for update room information. This use case outlines how staff update hostel room information, including room capacity, availability and other related details in the system.

Table 4.10: Use case description of update facility information use case

Use Case Name	Update facility information
Actor(s)	Admin
Summary Description	The actor updates the information of the facility in the system.
Pre-Conditions/Entry Conditions	The actor had log in to the system.
Post-Conditions/Exit Conditions	The information of the facility is updated and recorded in the system.
Basic Path/Flow of Events	1. The actor accesses to the facility management page. 2. The actor enters the facility information. 3. The actor clicks the save button to updates and saves the facility information in the system. 4. The use case ends.
Alternative path (if exist)	N/A
Exceptional path (if exist)	N/A

Table 3.10 shows the use case description for update facility information. This use case explains how staff manage and update information related to hostel facilities, ensuring data accuracy and transparency for residents.

Table 4.11: Use case description of process complaint report use case

Use Case Name	Process complaint report
Actor(s)	Admin

Summary Description	The actor processes the complaint report by review the report and replies with feedback.
Pre-Conditions/Entry Conditions	The student submitted the complaint report.
Post-Conditions/Exit Conditions	The complaint report had been processed by reply feedback.
Basic Path/Flow of Events	1. The actor accesses to the complaint management page. 2. The actor reviews the complaint report and replies with feedback. 3. The use case ends.
Alternative path (if exist)	N/A
Exceptional path (if exist)	N/A

Table 3.11 shows the use case description for process complaint report. This use case description allows staff to process submitted complaints by reviewing them and providing feedback to residents, thereby addressing hostel issues.

Table 4.12: Use case description of update announcement use case

Use Case Name	Update Announcement
Actor(s)	Admin
Summary Description	The actor updates the latest announcement in the system.
Pre-Conditions/Entry Conditions	The actor had log in to the system.
Post-Conditions/Exit Conditions	The announcement is updated and displayed in the system.
Basic Path/Flow of Events	1. The actor accesses to the announcement page. 2. The actor updates the announcement.

	3. The system displays the announcement at the announcement section. 4. The use case ends.
Alternative path (if exist)	N/A
Exceptional path (if exist)	N/A

Table 3.12 shows the use case description for update announcement. This use case describes how staffs update or publish announcements in the system so that residents receive timely and relevant notifications.

Table 4.13: Use case description of allocate room use case

Use Case Name	Allocate room
Actor(s)	Admin
Summary Description	The actor allocates the room to each resident.
Pre-Conditions/Entry Conditions	The room information had been recorded in the system and the student had submitted the application.
Post-Conditions/Exit Conditions	The room is allocated to the student where the application is approved.
Basic Path/Flow of Events	1. The actor accesses to the room allocation page. 2. The actor clicks the allocate room button. 3. The system allocates the room based on the defined criteria. 4. The system displays the room allocation result. 5. The use case ends.
Alternative path (if exist)	N/A
Exceptional path (if exist)	N/A

Table 3.13 shows the use case description for allocate room. This use case covers how staffs allocate rooms to students whose applications have been approved. The process involves a room assignment algorithm and displays results to admins.

Table 4.14: Use case description of track visitor use case

Use Case Name	Track visitor
Actor(s)	Admin
Summary Description	The actor tracks the outside visitor by record the visitation information in the system.
Pre-Conditions/Entry Conditions	Outside visitor had visit the hostel.
Post-Conditions/Exit Conditions	The visitation information is recorded in the system.
Basic Path/Flow of Events	1. The actor accesses to the visitation management page. 2. The actor reviews the visitation information. 3. The use case ends.
Alternative path (if exist)	2.1 The actor fills the visitation form if the visitor did not fill the visitation form. 2.2 The actor submits the visitation form.
Exceptional path (if exist)	N/A

Table 3.14 show the use case description for track visitor. This use case details how staffs track visitor information by recording visitation data manually or reviewing visitor-submitted forms, supporting hostel security and monitoring.

3.4.1.3 Activity Diagram

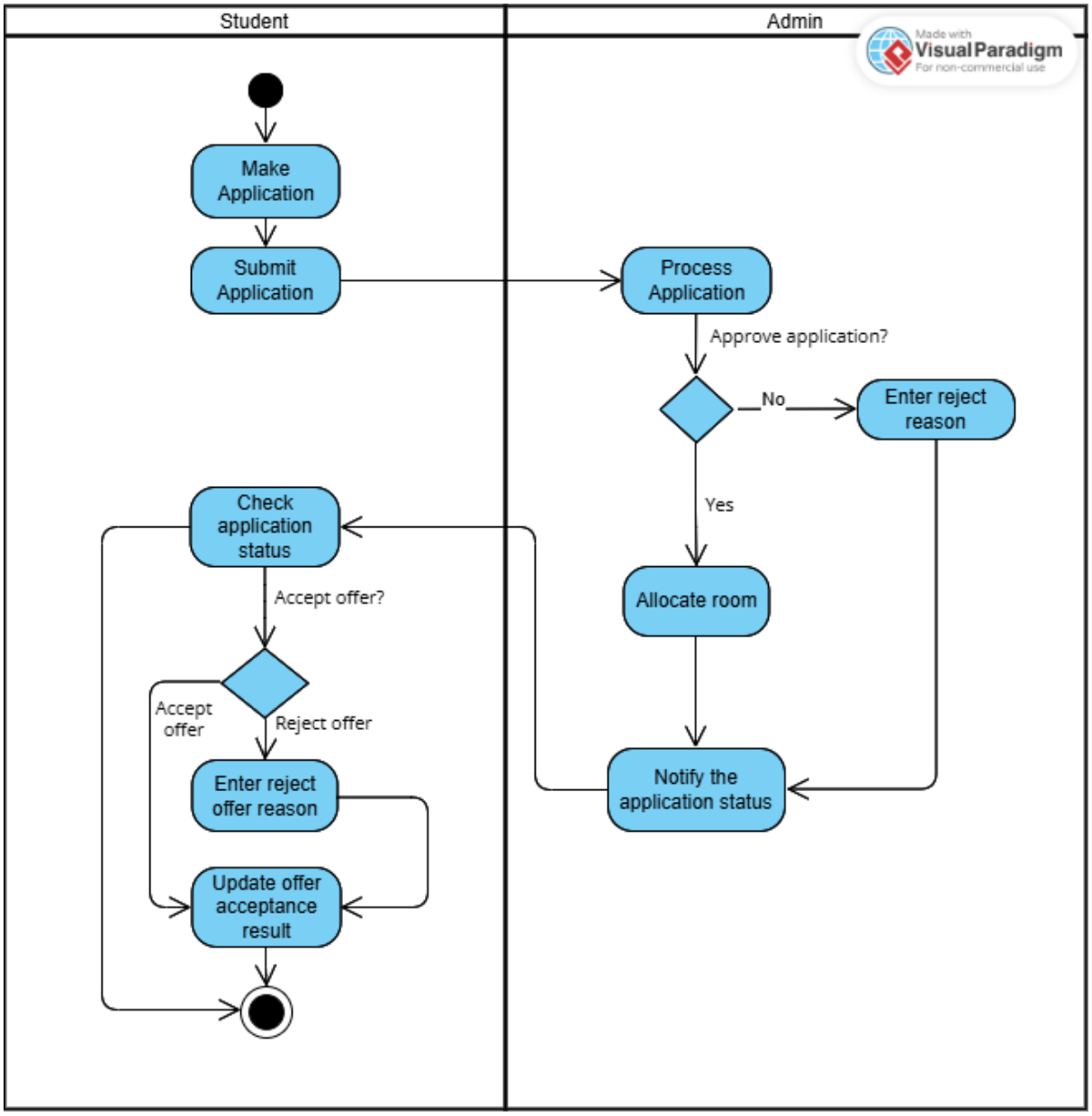


Figure 4.17: Activity diagram for the process flow for hostel application

Figure 3.17 shows the activity diagram for the process flow for hostel applications. The process begins when a student makes an application. After completing the application form, the student submits it. Once submitted, the admin processes the application by either approving or rejecting it. If the application is approved, the admin allocates a room, and the system notifies the student of the application status. The student then checks the status. If the status is approved, the student must update the offer status by either accepting or rejecting it. If the student accepts,

the system updates the offer acceptance result, and the process ends. If the student rejects, the student must provide a reason, and the system updates the offer acceptance result before ending the process. If the application is rejected, the system notifies the student, who then checks the status, and the process ends.

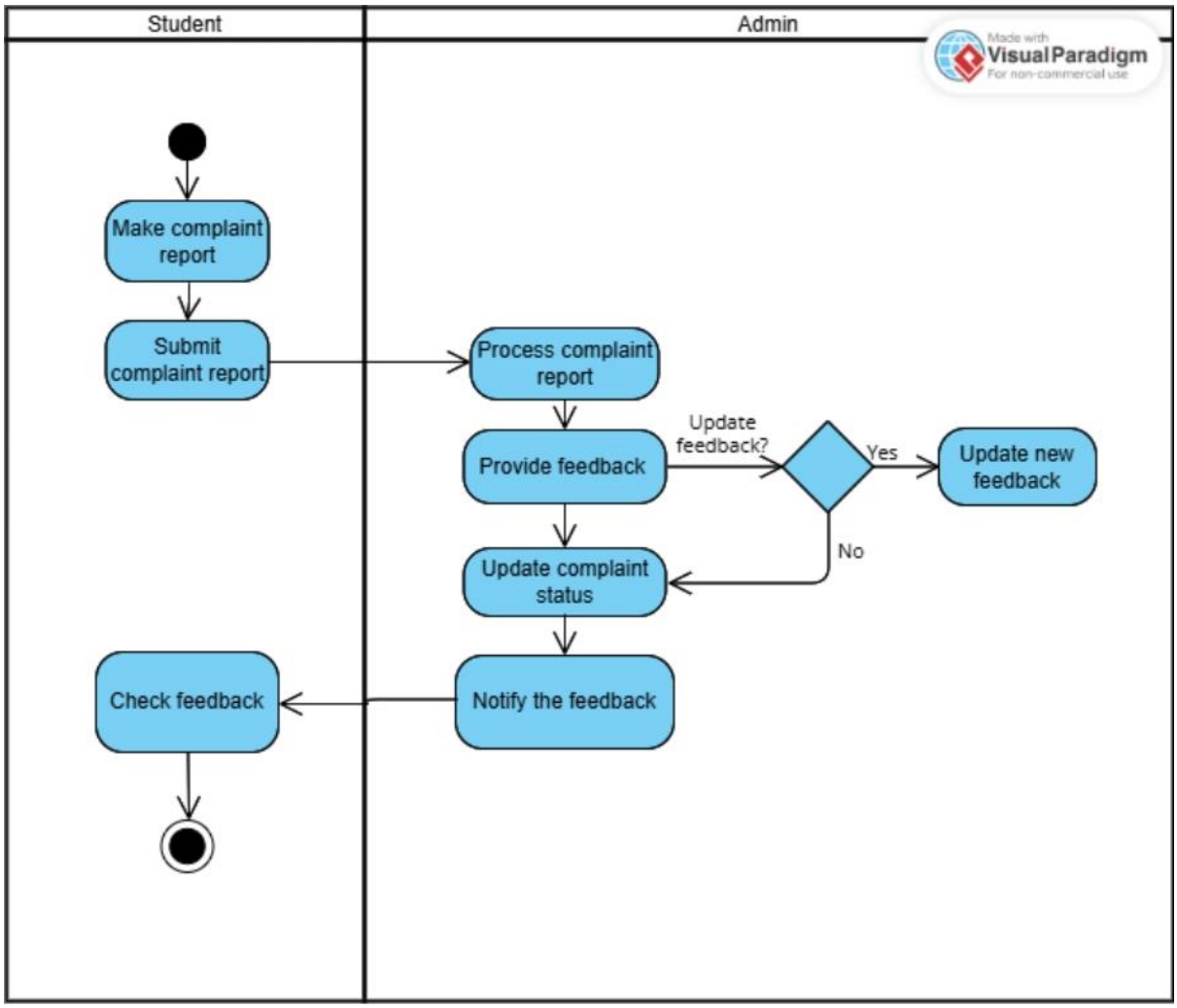


Figure 4.18: Activity diagram for the process flow for report a complaint

Figure 3.18 shows the activity diagram for the process flow for report a complaint. The process starts when the student makes complaint report. After that, student submits the complaint report. Once submitted, admin process the complaint report. Admin provides the feedback for the complaint report. If admin want to update the latest feedback, admin replies with the latest feedback. After that, the system updates the complaint status and notifies the

feedback. If there is not new feedback to update, the system updates the complaint status and notifies the feedback. After that, student check the feedback and the process end.

3.4.1.4 Entity Relationship Diagram

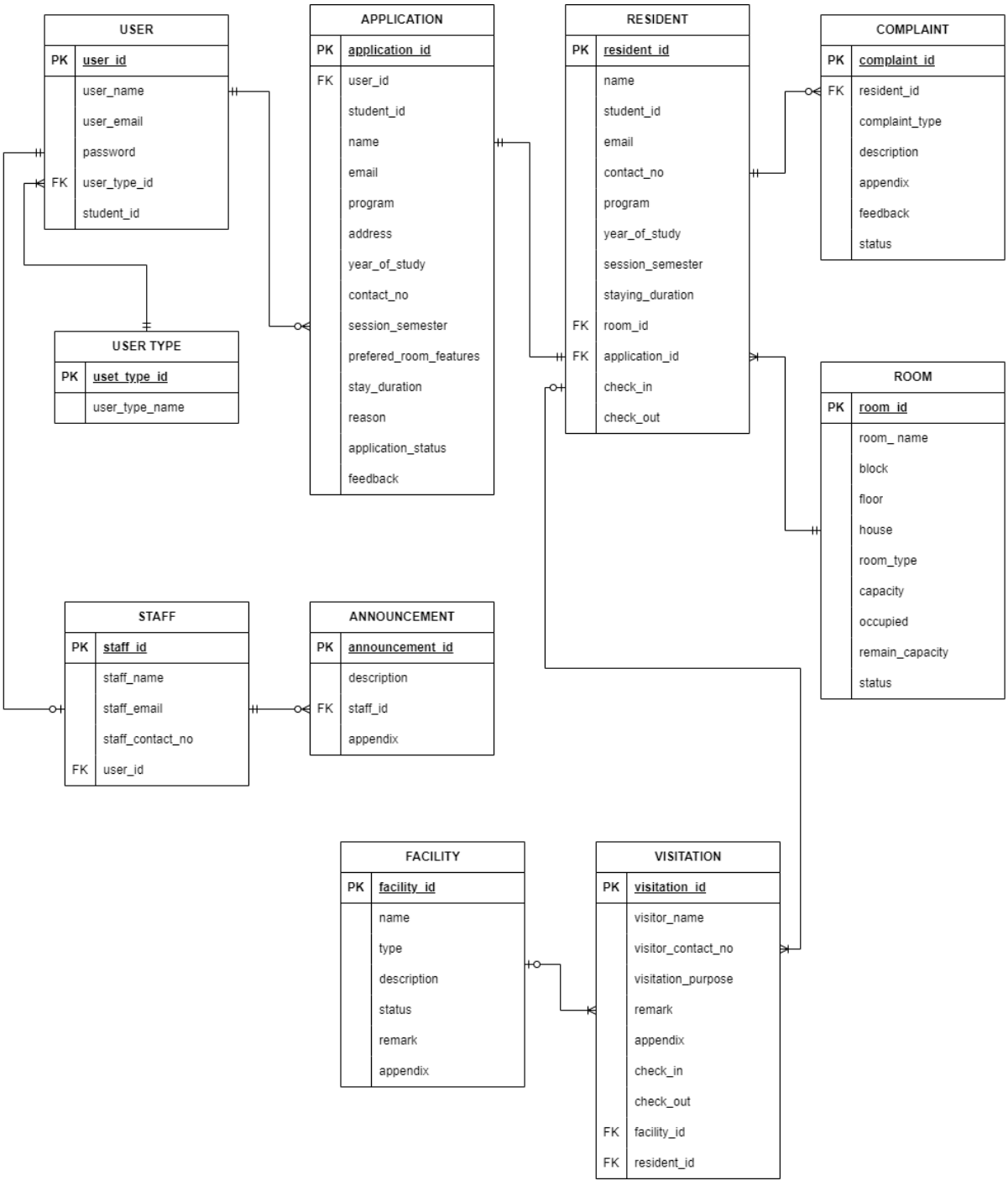


Figure 4.19: Entity relationship diagram for hostel management system

Figure 3.19 shows the entity-relationship diagram (ERD) for the proposed hostel management system. In this ERD, there are 10 entities: User, User Type, Application, Resident, Complaint, Announcement, Staff, Visitation, Facility, and Room. These entities represent distinct tables in the system database. Each table has corresponding attributes to record data. The User table stores information related to system users, whether they are students or admins. The User Type table defines the types of users, which consist of admin and student. In the User table, users are differentiated by the user type ID. The Staff table stores information about staff members. For Application, Complaint, Resident, Facility, Room, and Announcement, the related data is recorded in their respective tables.

3.4.1.5 Class Diagram

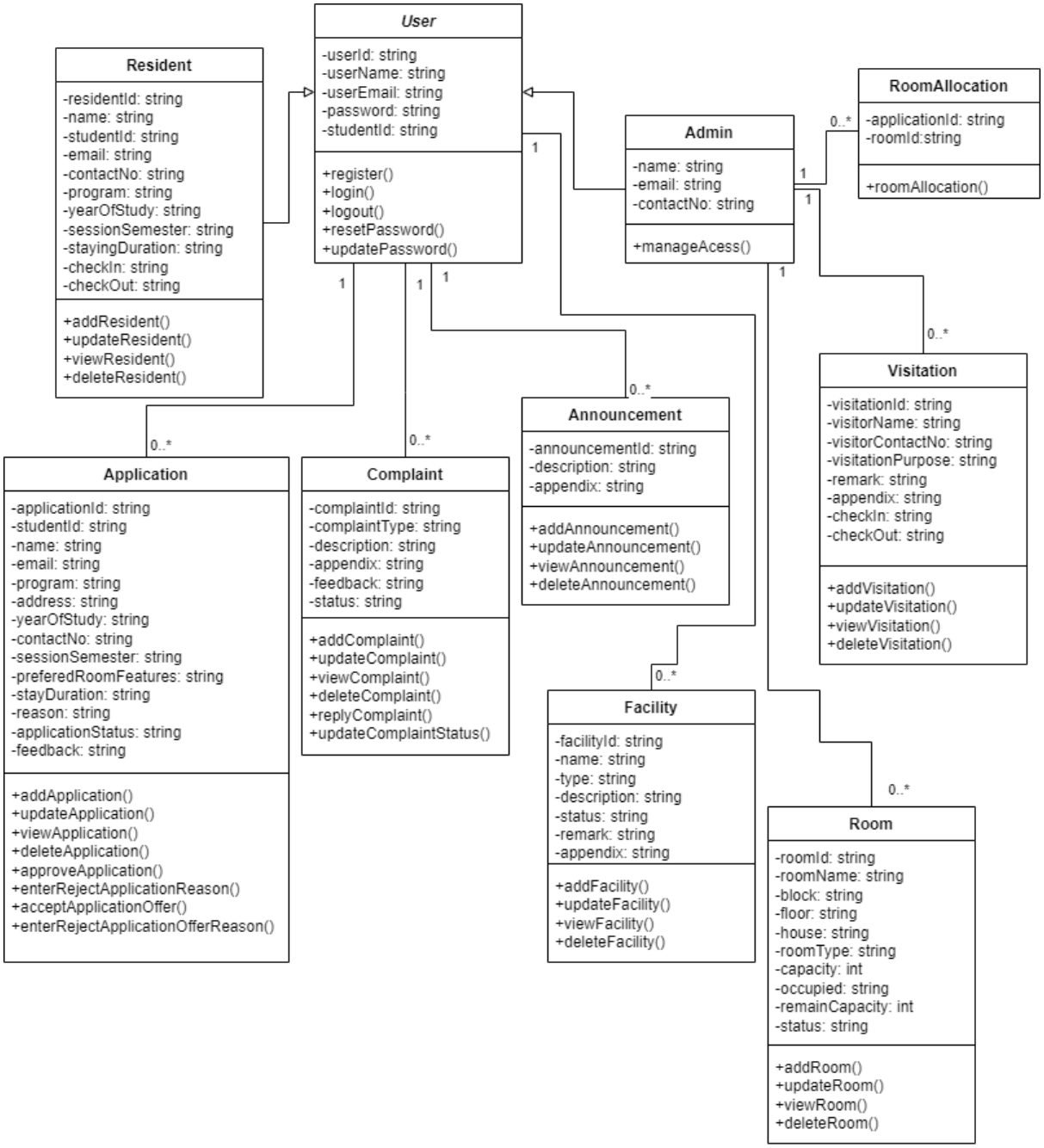


Figure 4.20: Class Diagram of proposed hostel management system

Figure 3.20 shows the Class Diagram for the proposed system. User is the superclass, with Resident and Admin inheriting from the User class. Other classes include Application, Complaint, Facility, Visitation, Room, RoomAllocation and Announcement. Room, RoomAllocation, and Visitation can only be accessed by the admin**,** while the other classes can

be accessed by both types of users. The corresponding attributes and functions of each class are shown in the figure.

3.4.2 Physical Design

The physical design focuses on creating the wireframe for the proposed system. The wireframe will outline the main functionalities of the system for two types of users.

3.4.2.1 Wireframe

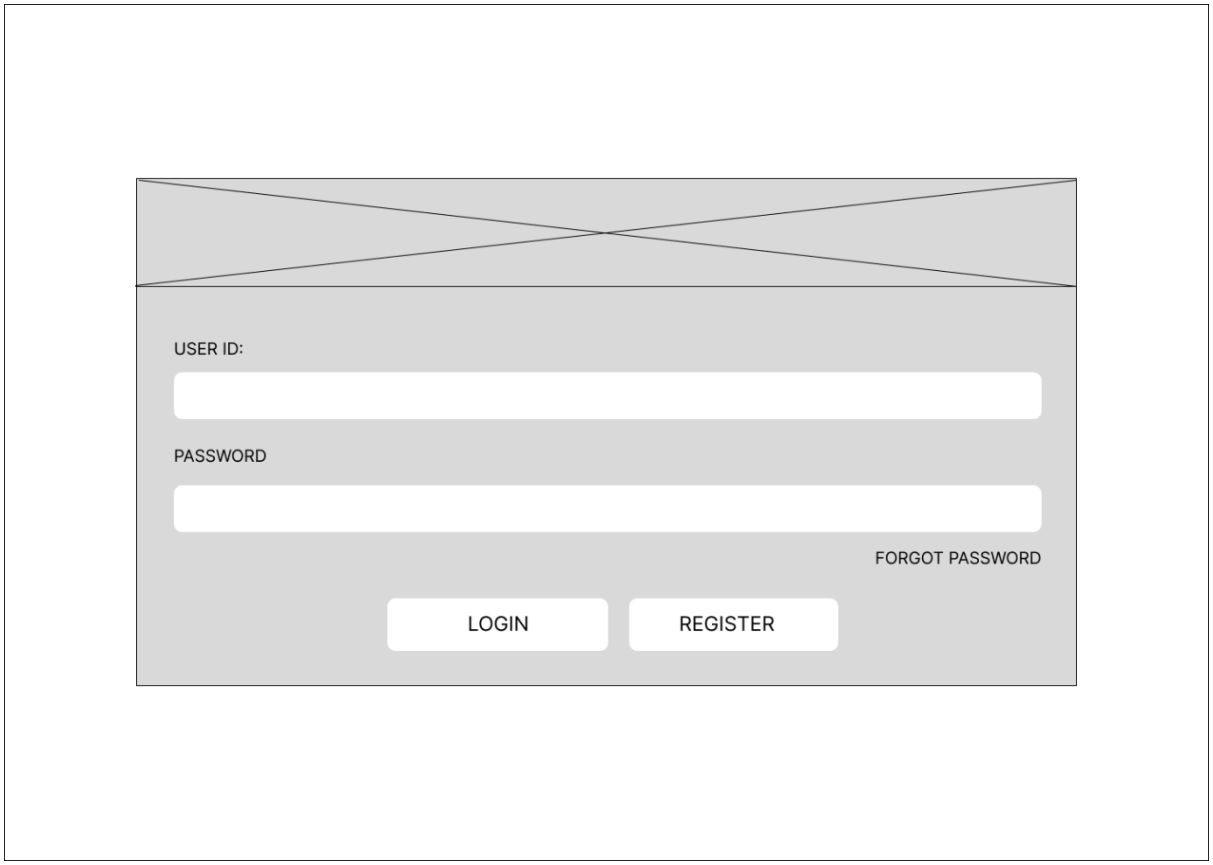


Figure 4.21: Wireframe of log in page

Figure 3.21 illustrates the wireframe of log in page. After the user have enter the user id and password, user can log in to the system by clicking the LOGIN button. If user haven't

had an account, user can register an account by clicking the REGISTER button. If user forgot the password, user can click on the FORGOT PASSWORD text to reset the password.

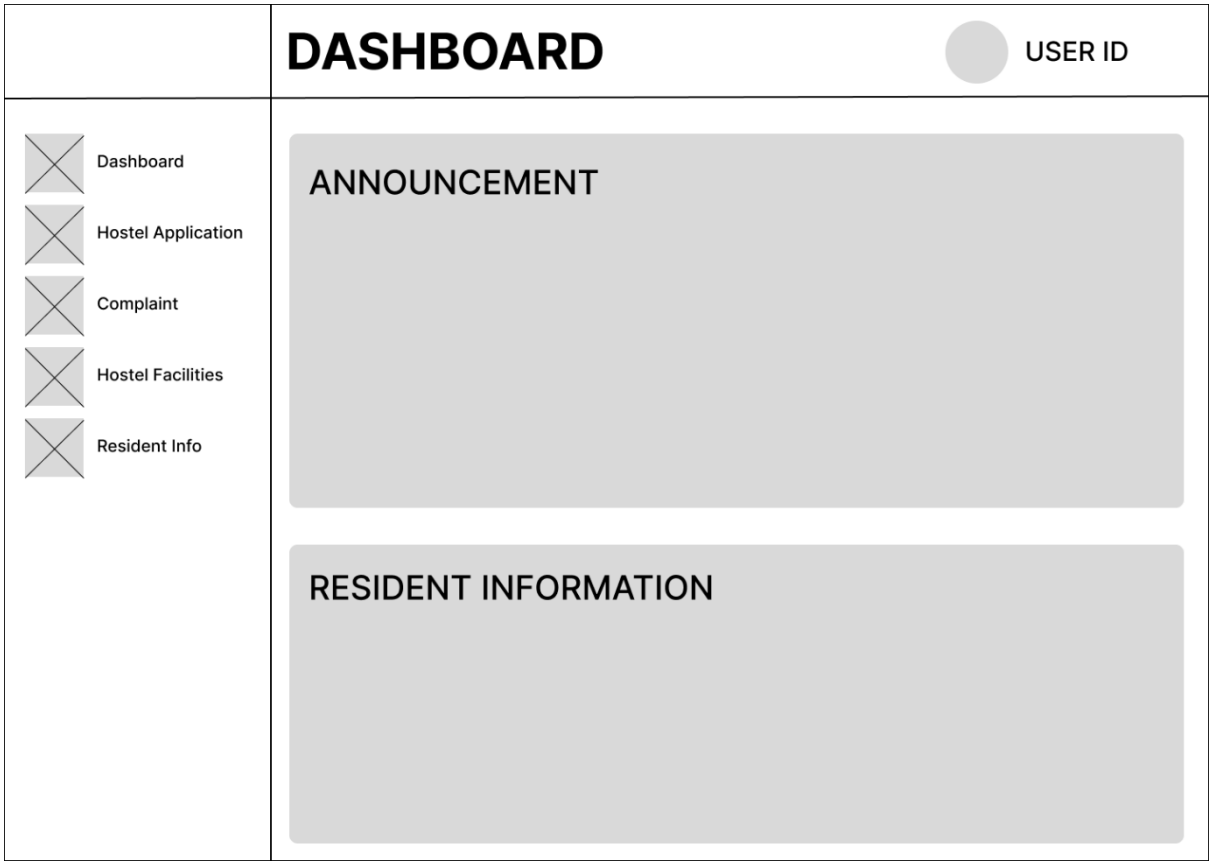


Figure 4.22: Wireframe of dashboard for student panel

Figure 3.22 illustrates the wireframe of dashboard for student panel. The system will have a side bar at the left side to provide access for the user to navigate to certain page. The dashboard will display the hostel notice announcements and the resident information. the resident information will display the detail of the user and housemates. At the top right corner, it is the user profile section.

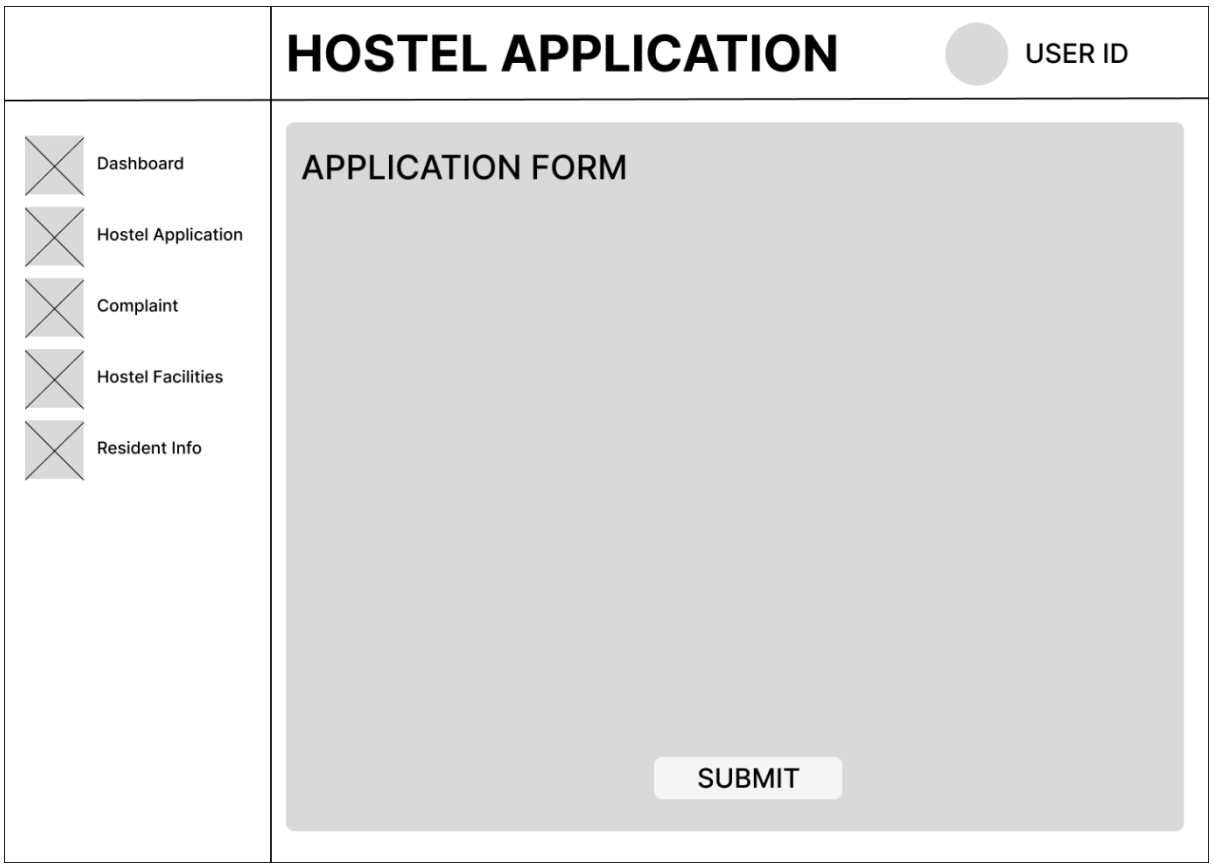


Figure 4.23: Wireframe of apply for hostel for student panel

Figure 3.23 illustrates the wireframe of hostel application for student panel. The hostel application form is displayed. After the application form is filled, applicant submit the application by clicking the SUBMIT button.

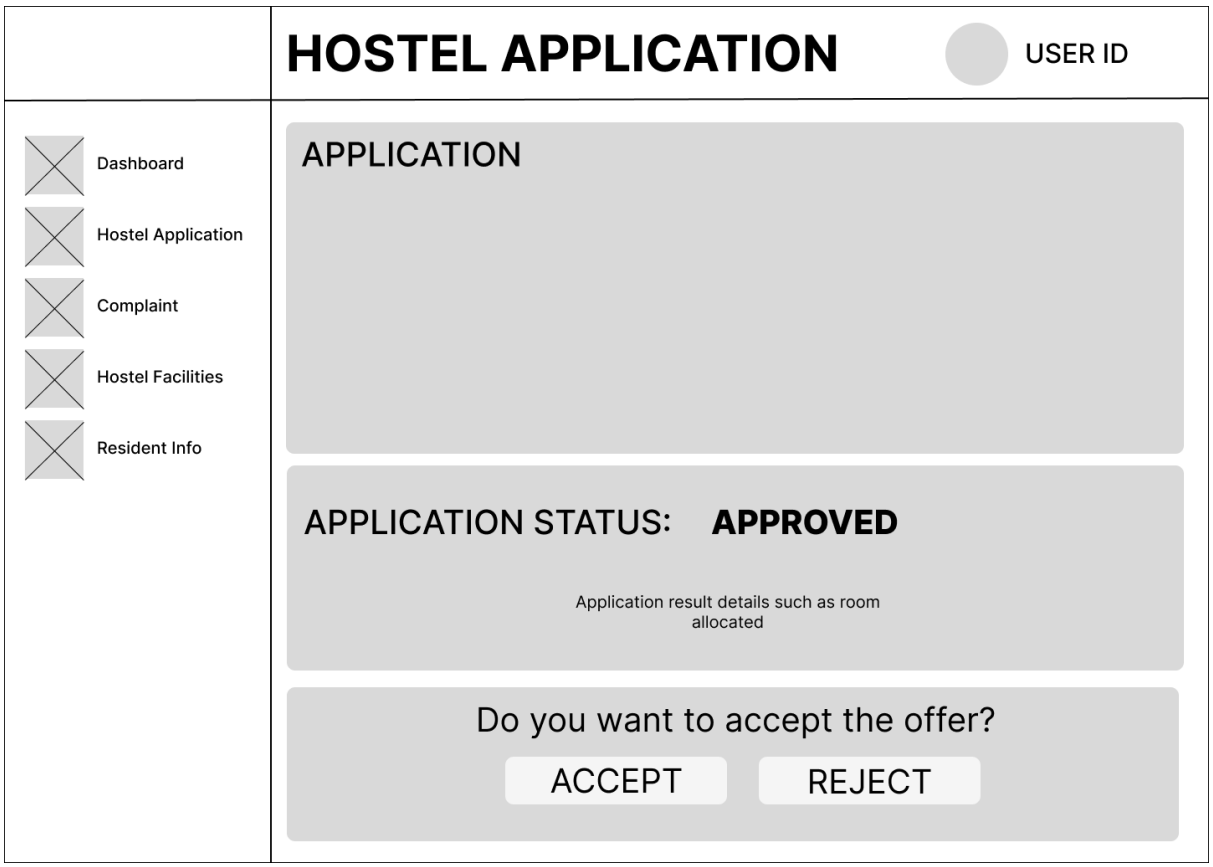


Figure 4.24: Wireframe of checks application and updates offer status

Figure 3.24 illustrates the wireframe of checks application and updates offer status. After the application is submitted, the applicant still can review the application form. Applicant can check for the application status. If the status is approved, the application result details such as room allocated is displayed. If the status is rejected, the reason will be displayed. The applicant needs to update the offer status by either accept or reject the offer by clicking on the corresponding button.

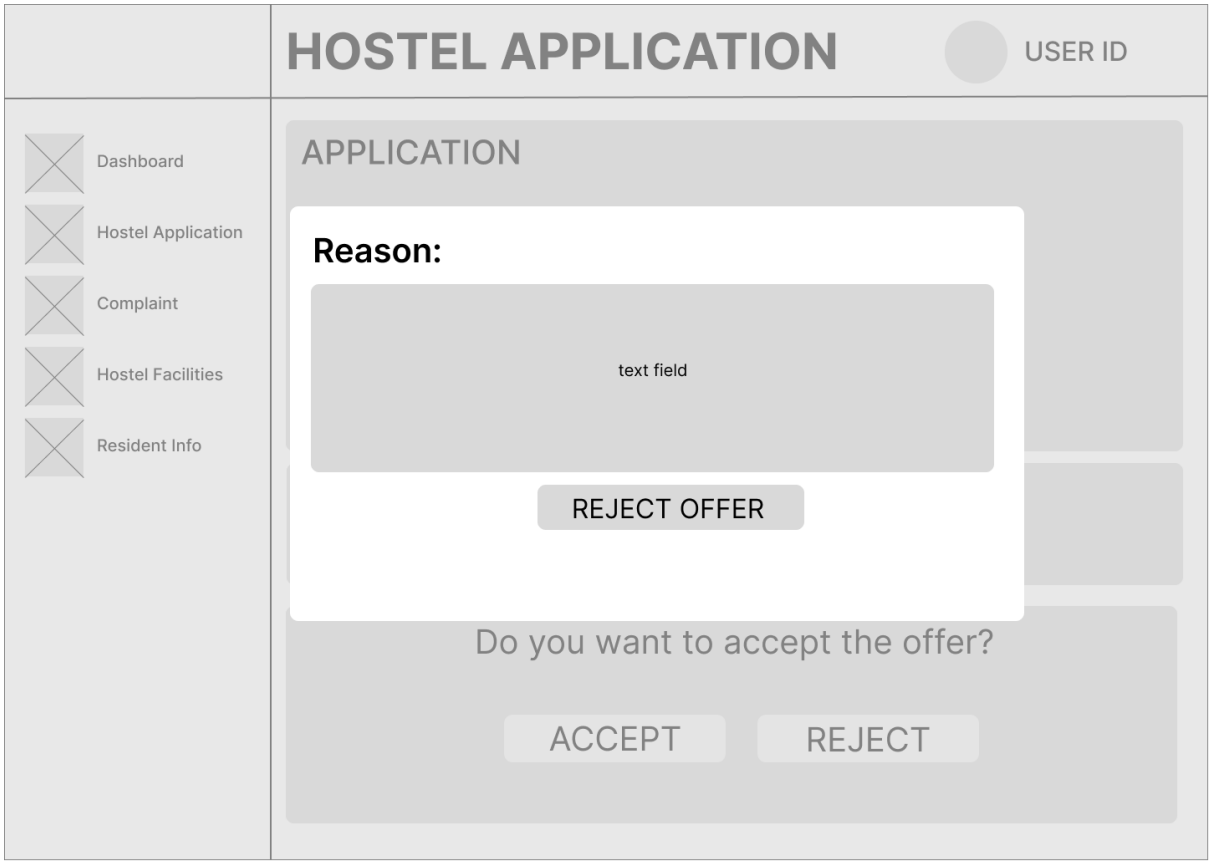


Figure 4.25: Wireframe of enter reject offer reason for student panel

Figure 3.25 illustrated the wireframe of the enter reject offer reason for student panel. If the applicant clicks the REJECT button, a pop-up text box will appear to prompt the applicant to enter the reject reason. After entering the reason, the applicant clicks on the REJECT OFFER button to update the offer acceptance status.

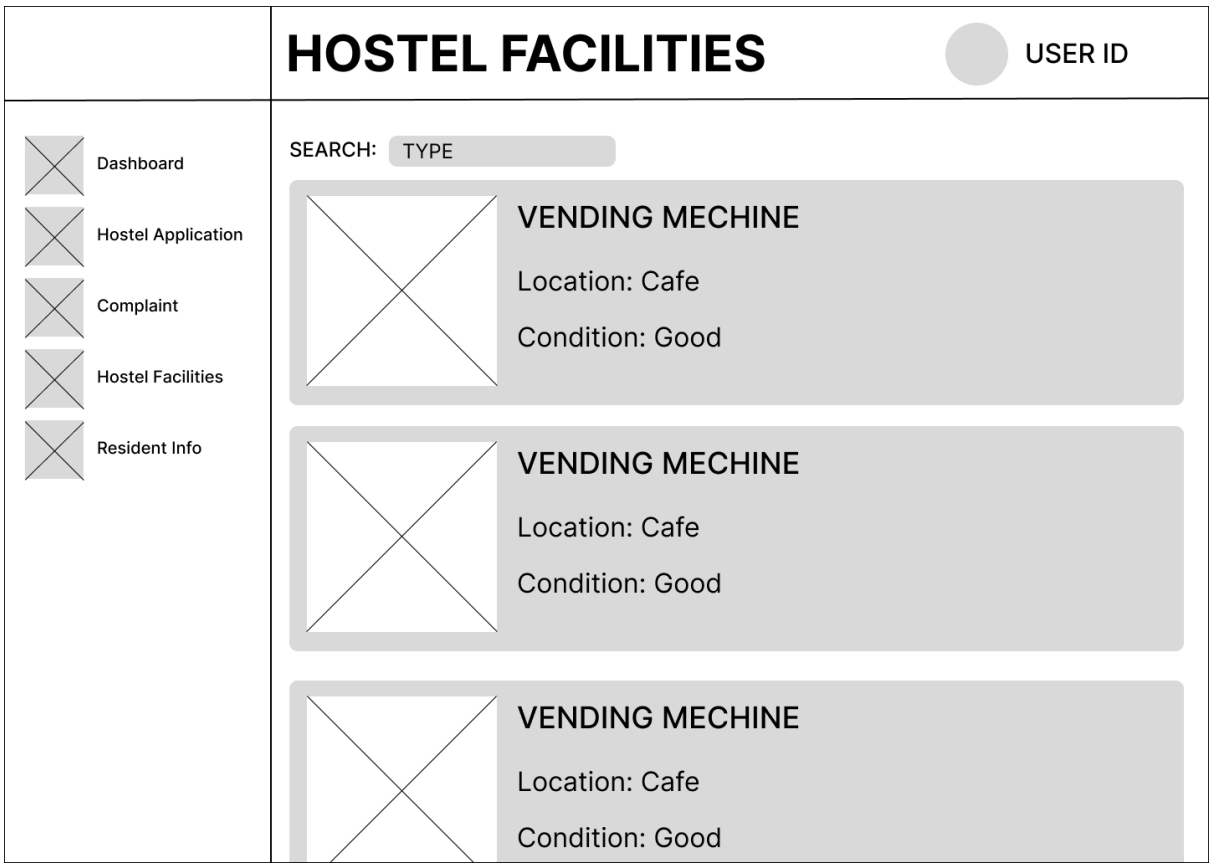


Figure 4.26: Wireframe of views hostel facilities information for student panel

Figure 3.26 illustrates the wireframe of views hostel facilities information for student panel. The facilities information will be displayed with the pictures. By this, when user want to use the facilities, they can navigate to this page to look for the information such as location and conditions. User can search the facilities by select the type of facilities on the drop-down menu of the search section.

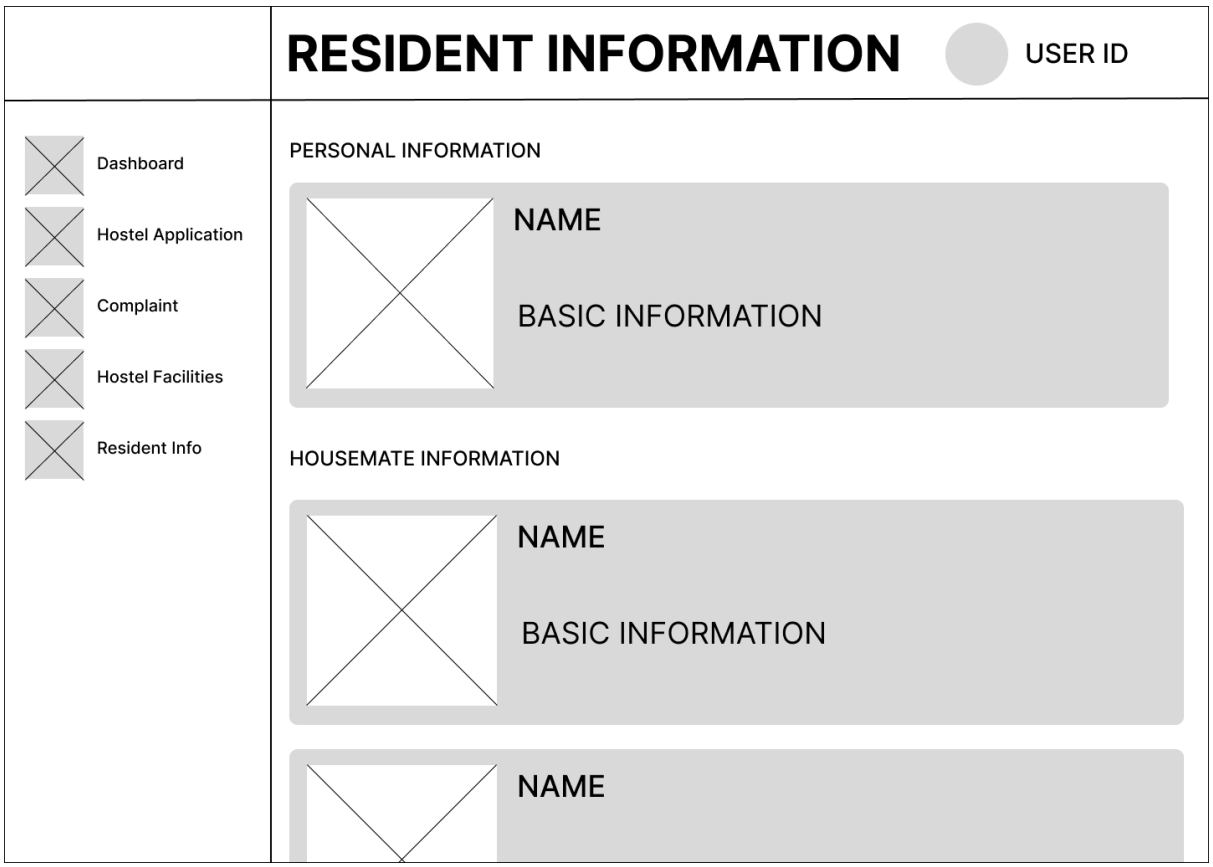


Figure 4.27: Wireframe of views and updates the resident information for student panel

Figure 3.27 illustrates the wireframe of views and updates the resident information for student panel. At the page, the resident personal information will be displayed along with the housemate's information.

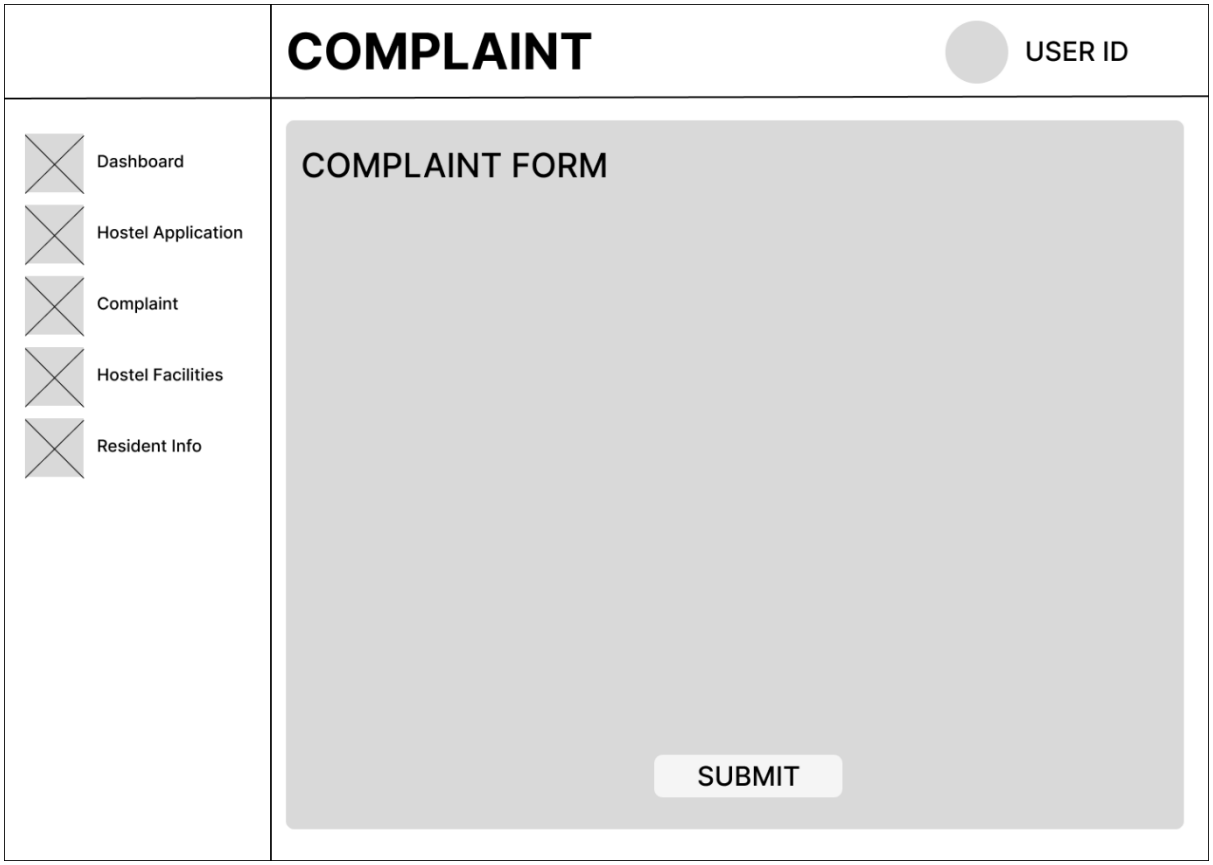


Figure 4.28: Wireframe of report a complaint for student panel

Figure 3.28 illustrates the wireframe of report complaint for student panel. After the user had reported a complaint, the user can review the complaint. The user can check for the complaint feedback at the feedback section. The feedback will be displayed along with the timeline.

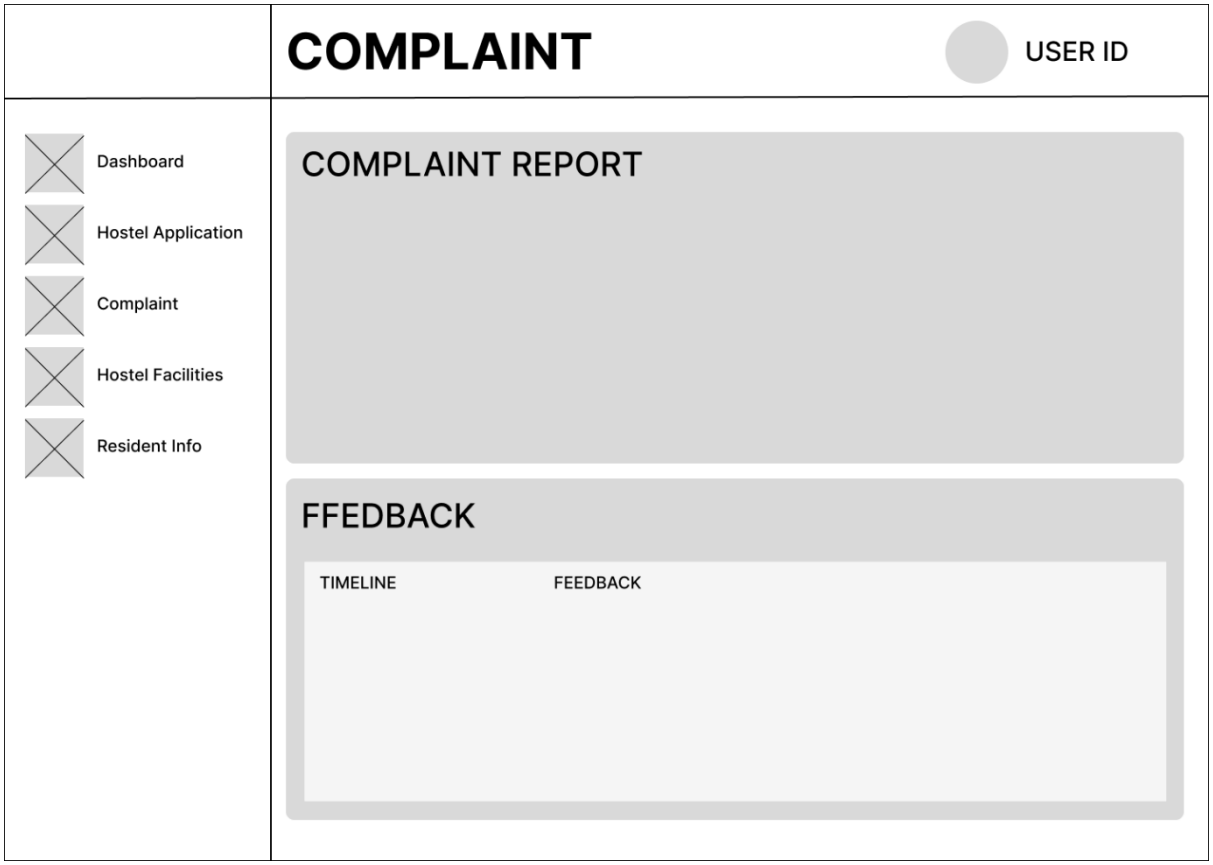


Figure 4.29: Wireframe of review complain feedback for student panel

Figure 3.29 illustrates the wireframe of review complaint feedback for student panel. The complaint form is displayed. After the complaint form is filled, the user submits the complaint form by clicking the SUBMIT button.

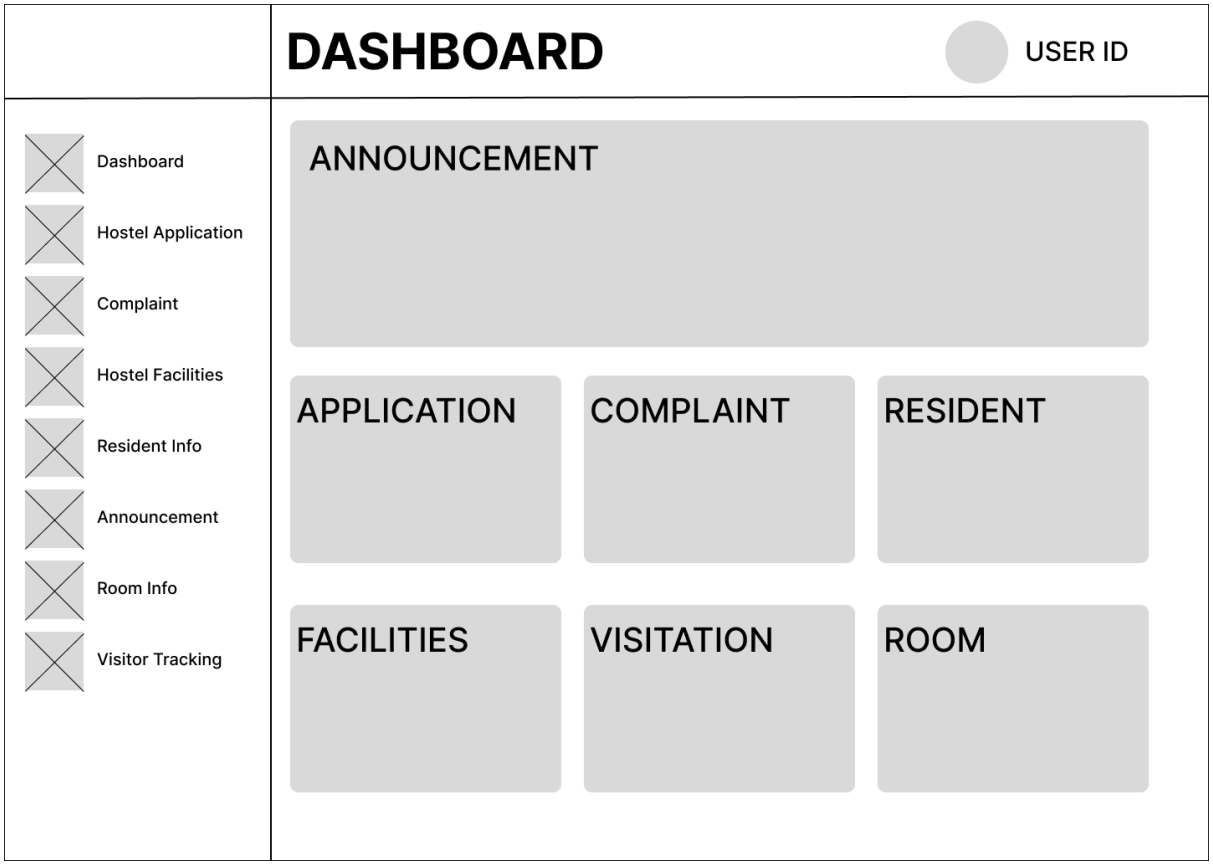


Figure 4.30: Wireframe of dashboard for admin panel

Figure 3.30 illustrates the wireframe of dashboard for admin panel. The dashboard will display the hostel notice announcements and summary of application, complaint report, resident, facilities, visitation and hostel room.

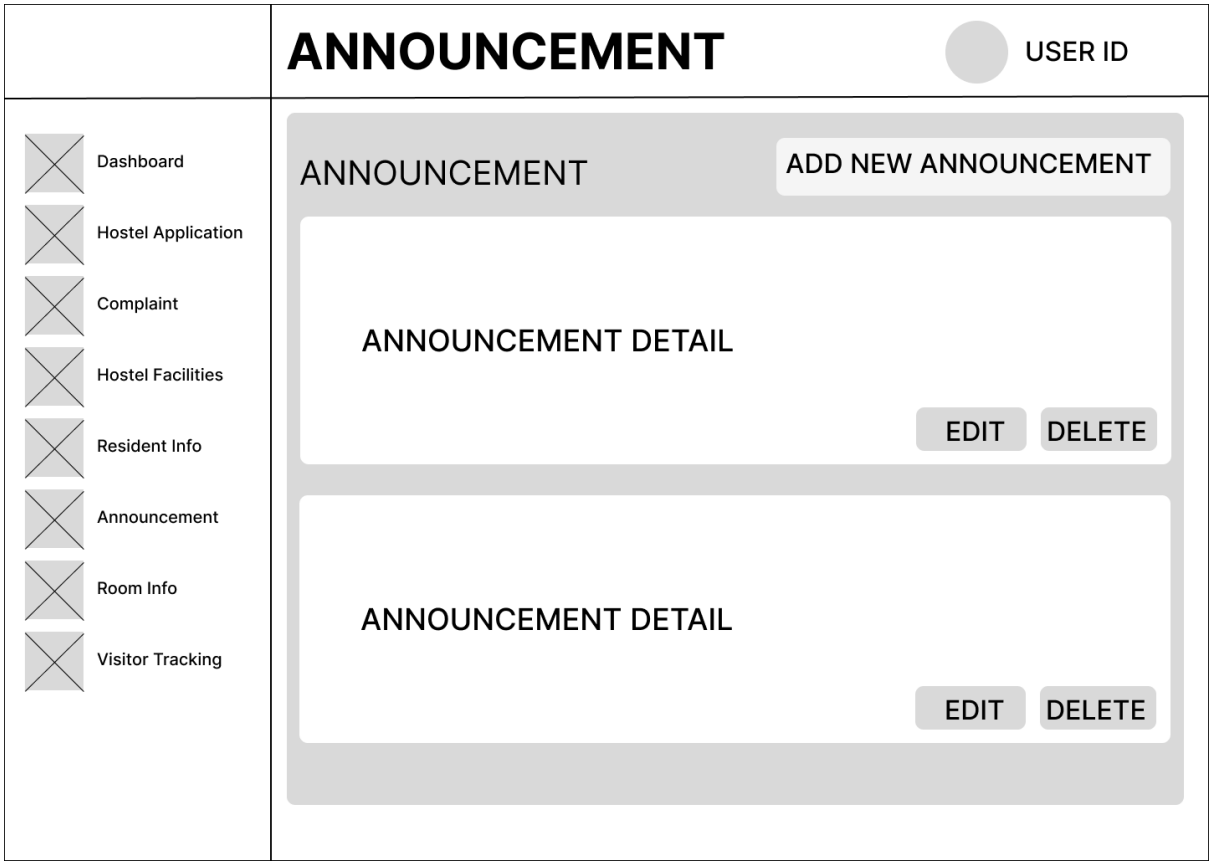


Figure 4.31: Wireframe of updates announcements for admin panel

Figure 3.31 illustrates the wireframe of updates announcements for admin panel. admin can view the announcements. Admin can add, edit and delete the announcement by clicking the corresponding button.

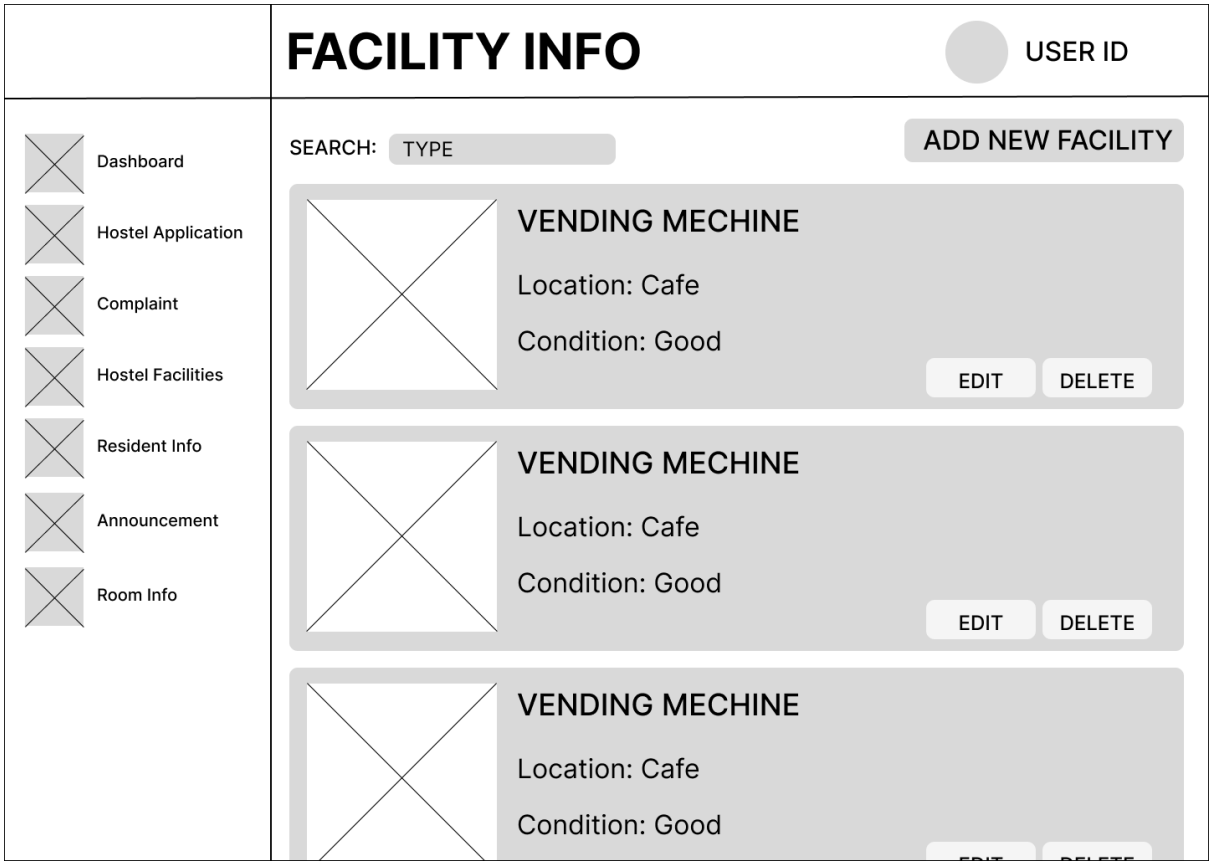


Figure 4.32: Wireframe of updates facilities information for admin panel

Figure 3.32 illustrates the wireframe of updates facilities information for admin panel. Admin can view for the facilities information. Admin can add, edit and delete by clicking the corresponding button.

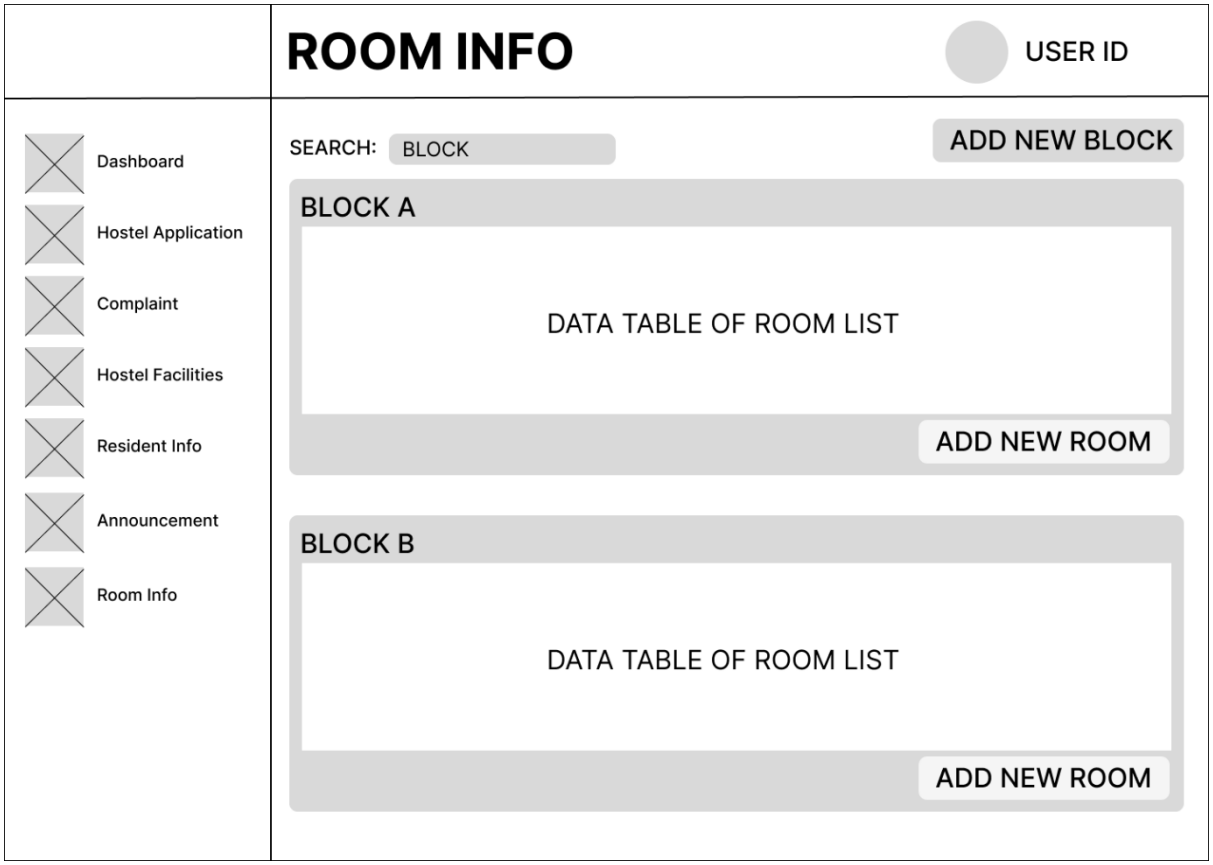


Figure 4.33:Wireframe of updates room information for admin panel

Figure 3.33 illustrates the wireframe of updates room information for admin panel. The room is classified by block and record in data table. Admin can add new block and new room by clicking the corresponding button. Admin edit and delete the room record by clicking the corresponding icon.

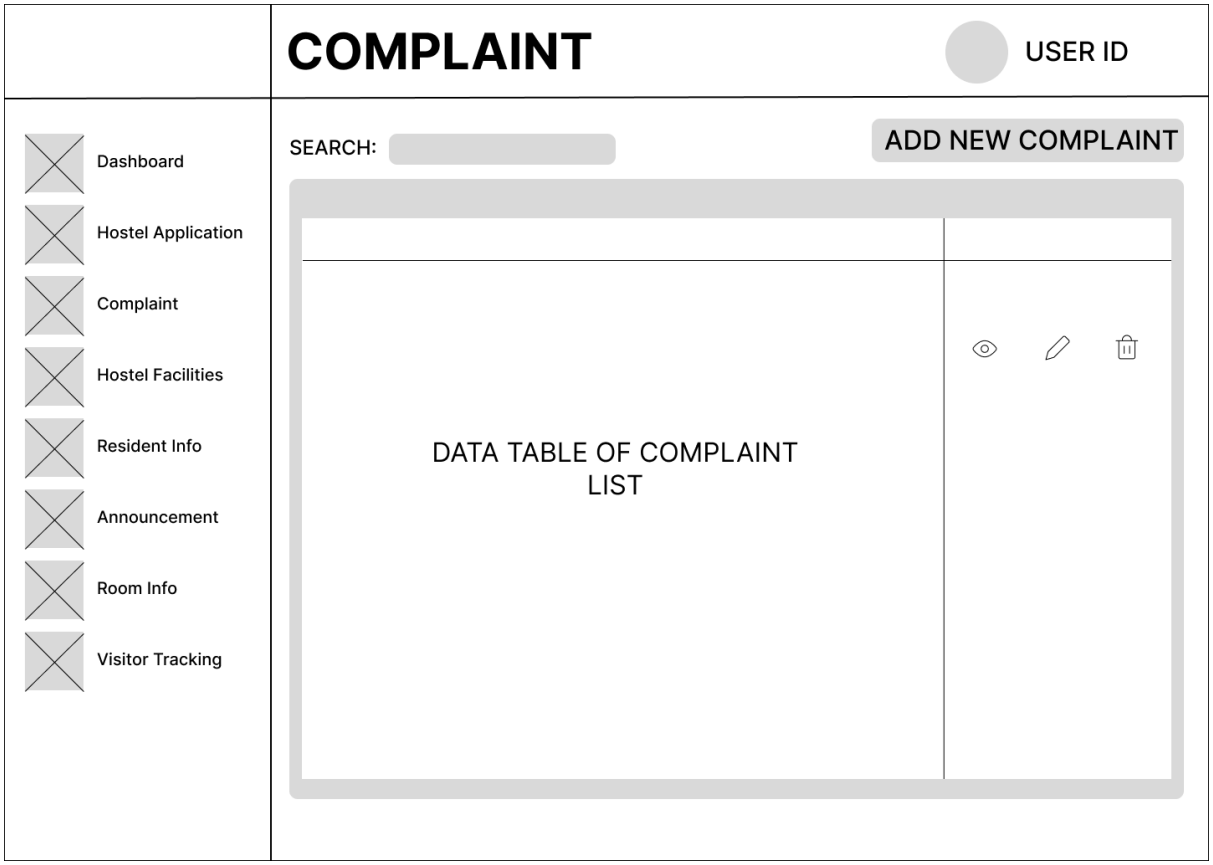


Figure 4.34: Wireframe of complaint list for admin panel

Figure 3.34 illustrates the wireframe of complaint list for admin panel. The room information are recorded in the data table. Admin can add, view, edit and delete the complaint list by clicking the corresponding button or icon.

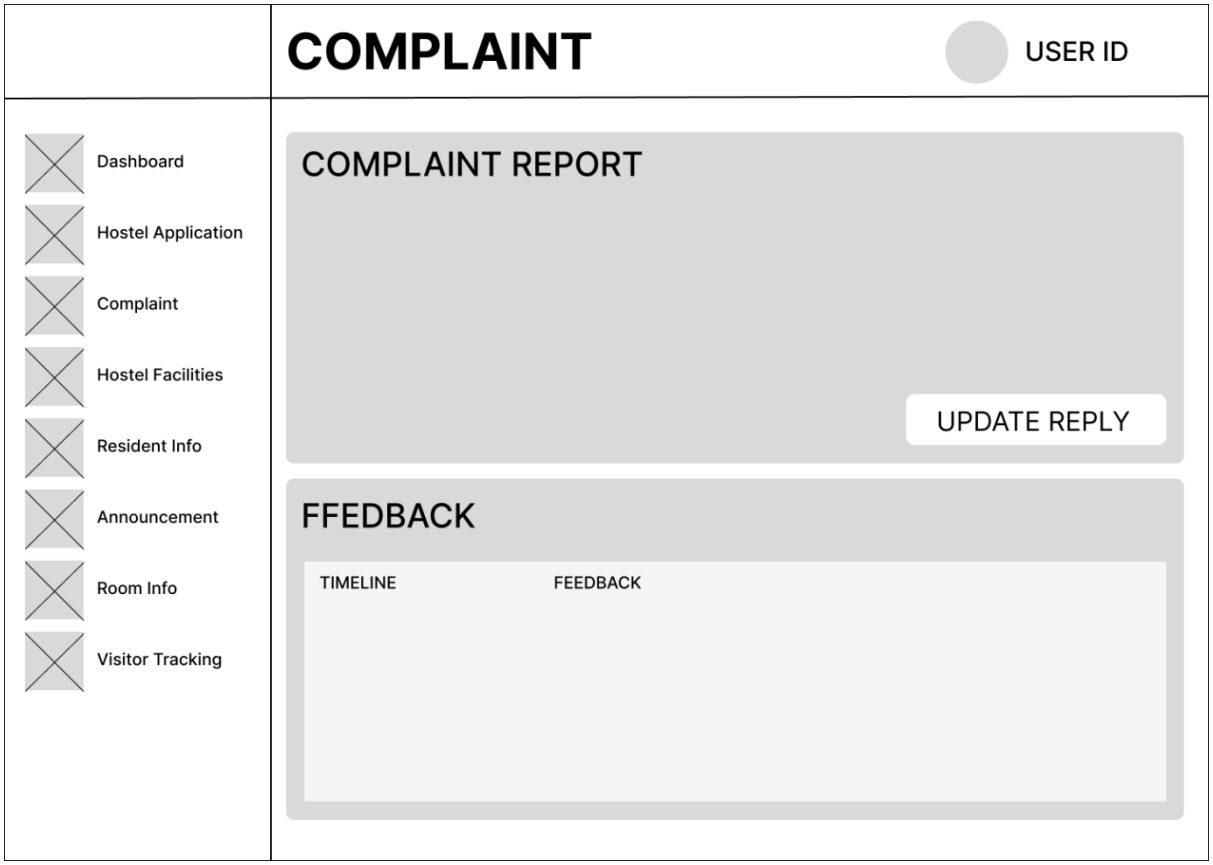


Figure 4.35: Wireframe of review complaint report for admin panel

Figure 3.35 illustrate the wireframe of review complaint report for admin panel. admin can review the complaint and reply feedback by clicking on the UPDATE REPLY button.

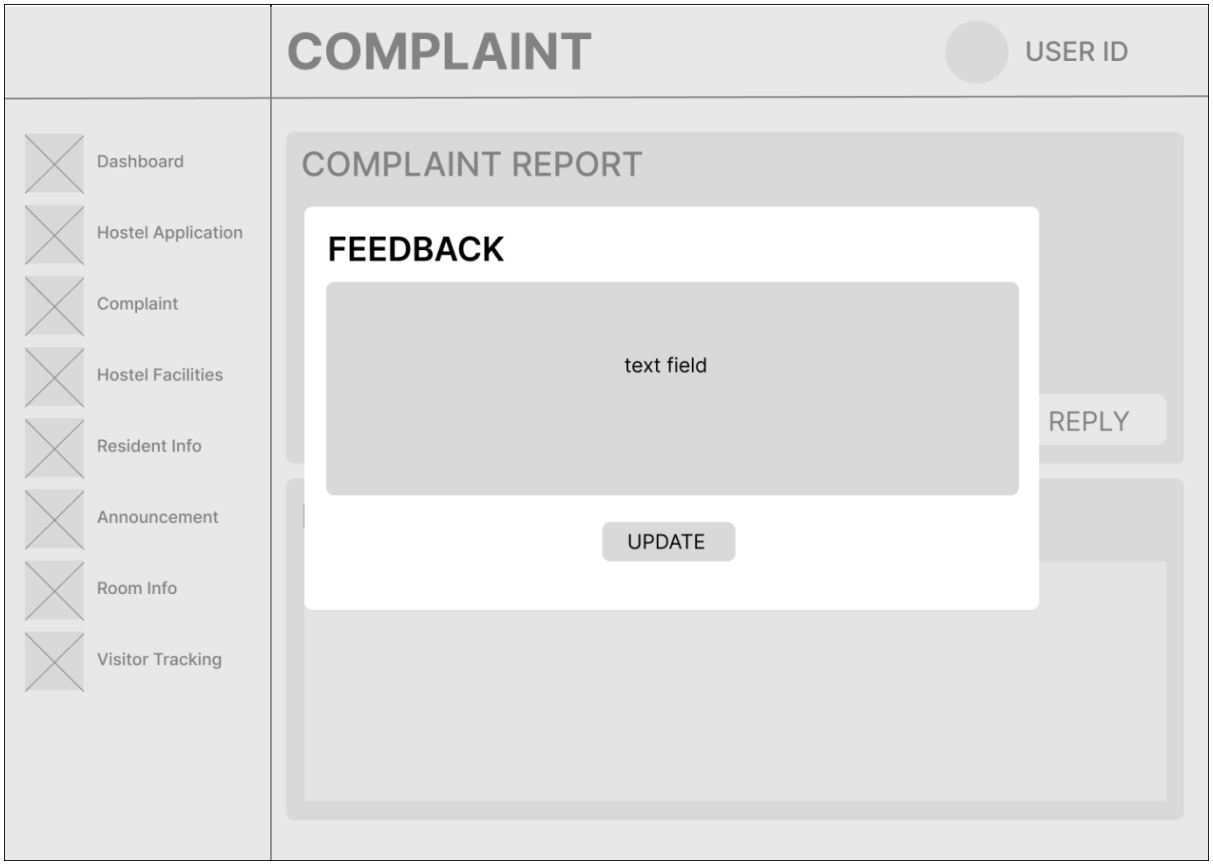


Figure 4.36: Wireframe of complaint feedback reply for admin panel

Figure 3.36 illustrate the wireframe of complaint feedback reply for admin panel. After the admin clicks the UPDATE REPLY button, a popup text box will appear to prompt the user to enter the feedback. After admin had entered the feedback, admin update the feedback by clicking UPDATE button.

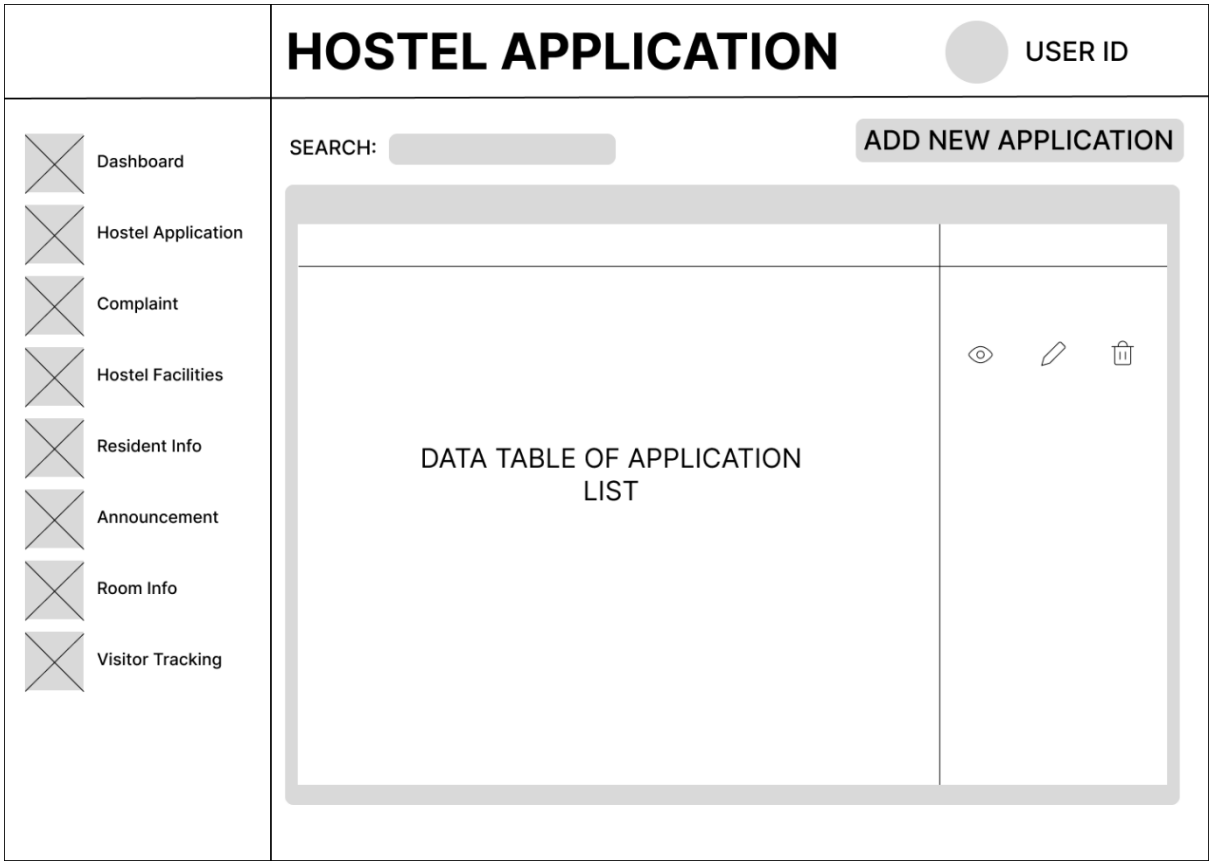


Figure 4.37: Wireframe of application list for admin panel

Figure 3.37 illustrates the wireframe of application list for admin panel. The application list is recorded in the data table. Admin can add, view, edit and delete the application by clicking the corresponding button or icon.

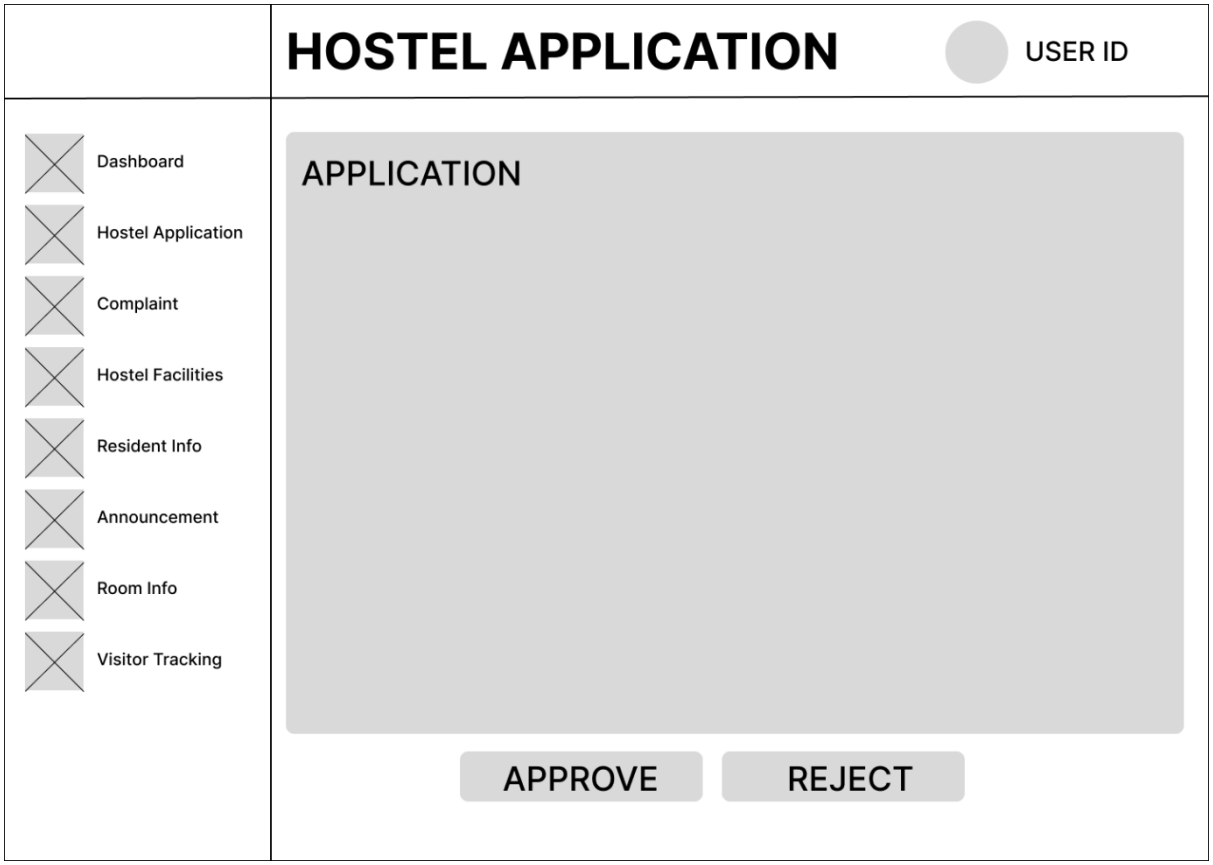


Figure 4.38: Wireframe of review application for admin panel

Figure 3.38 illustrate the wireframe of review application for admin panel. Admin review the application and decide to either approve or reject the application by clicking on the corresponding button.

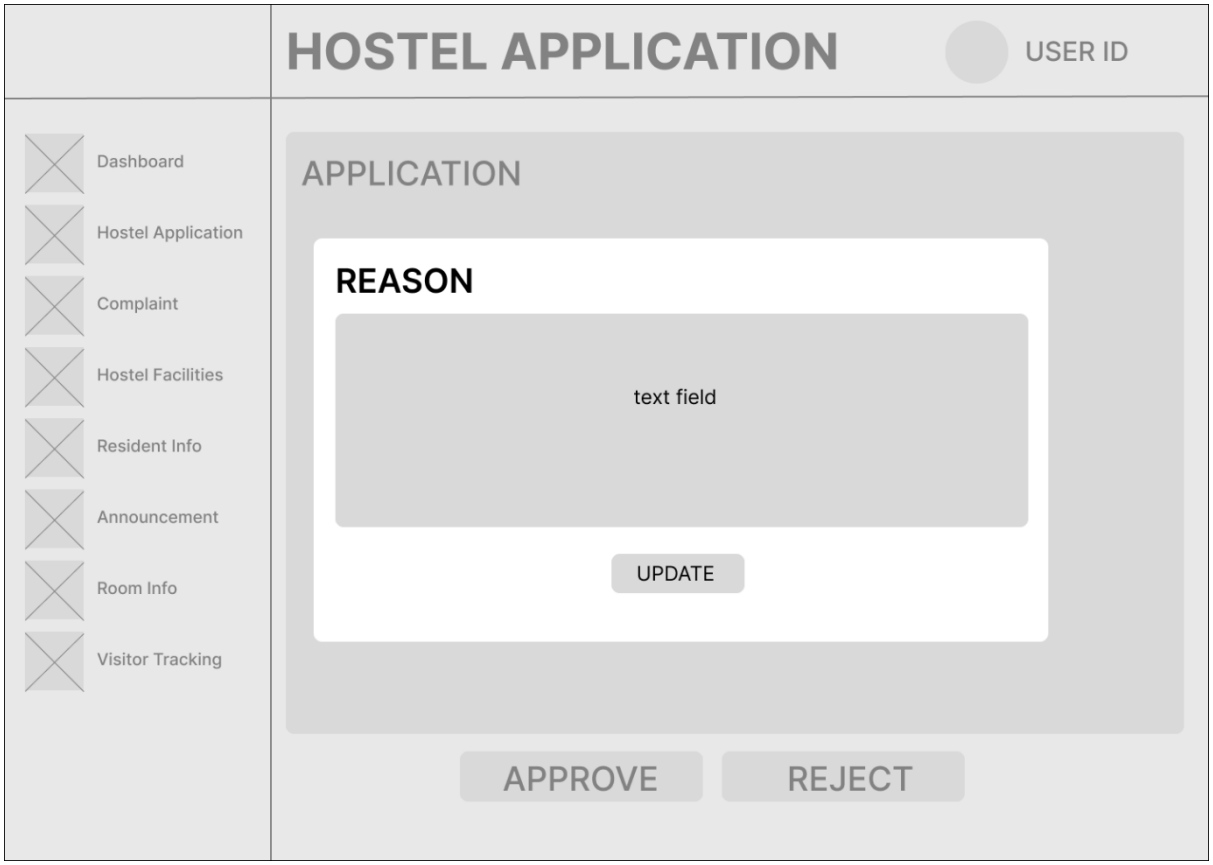


Figure 4.39: Wireframe of rejects application for admin panel

Figure 3.39 illustrates the wireframe of rejects application for admin panel. After the REJECT button is clicked, a pop-up text box will appear to prompt the user to enter the reject reason. After enters the reason, admin updates the status by clicking the UPDATE button.

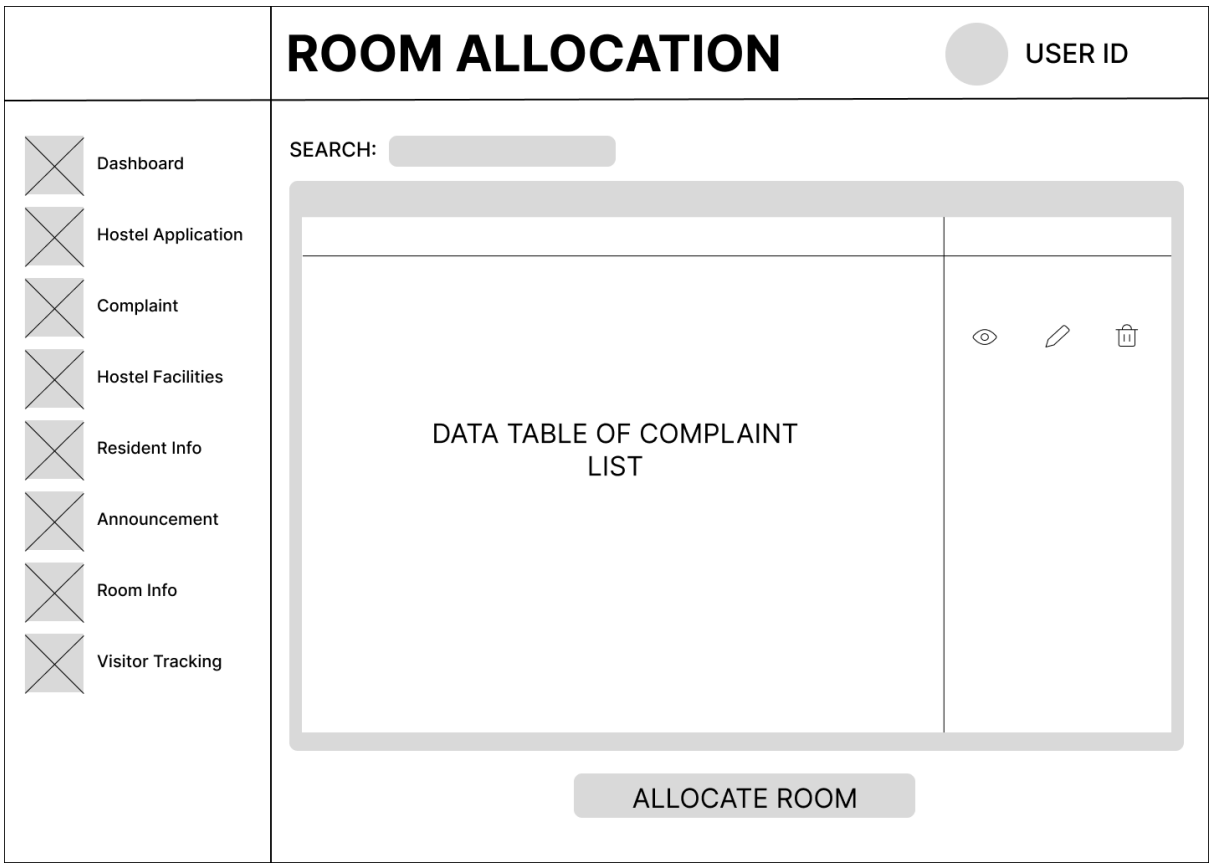


Figure 4.40: Wireframe of allocate room for approved application for admin panel

Figure 3.40 illustrates the wireframe of allocate room for approved application for admin panel. The data table will record the application list that have been approved. Admin allocate the room to each application by clicking the ALLOCATE ROOM button.

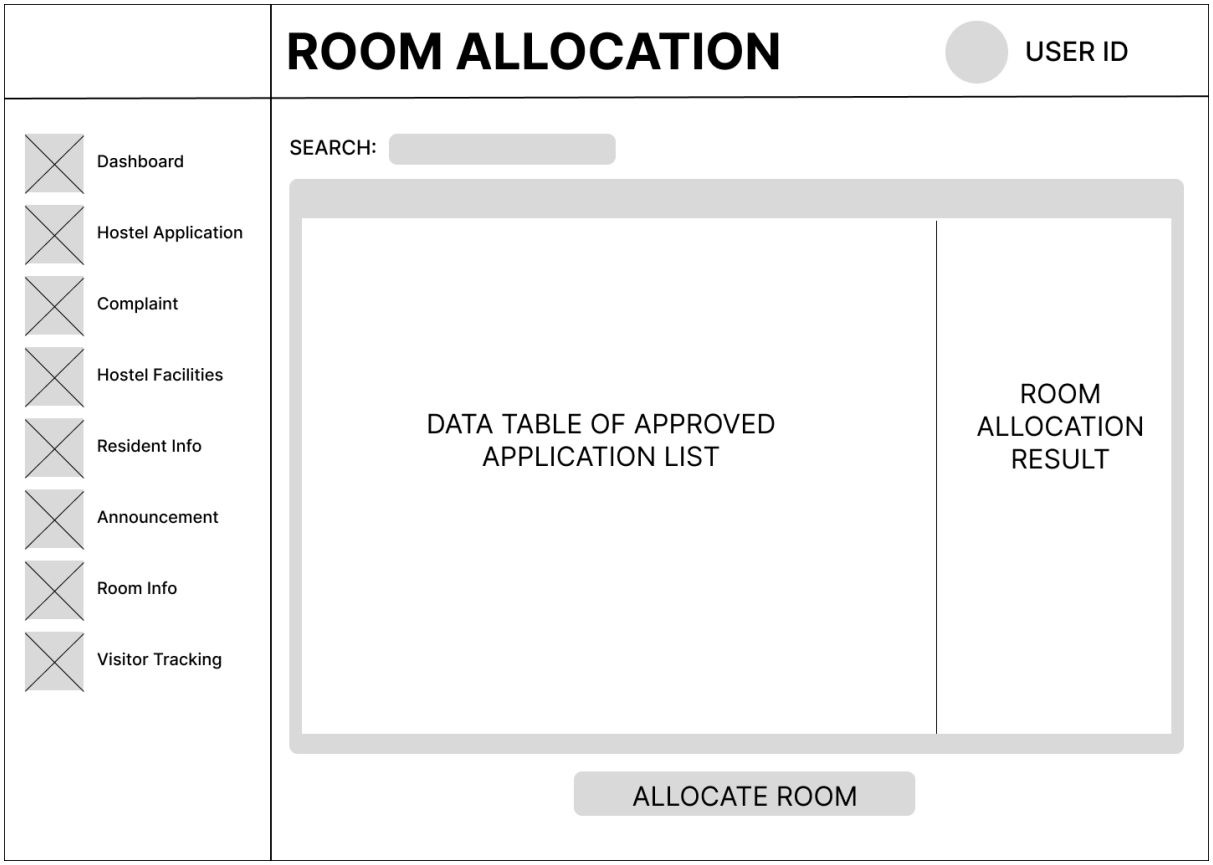


Figure 4.41: Wireframe of displays the result of room allocation process for admin panel

Figure 3.41 illustrates the wireframe of displays the result of room allocation process for admin panel. After clicking the ALLOCATE ROOM button, the allocated room will be displayed on beside each approved application.

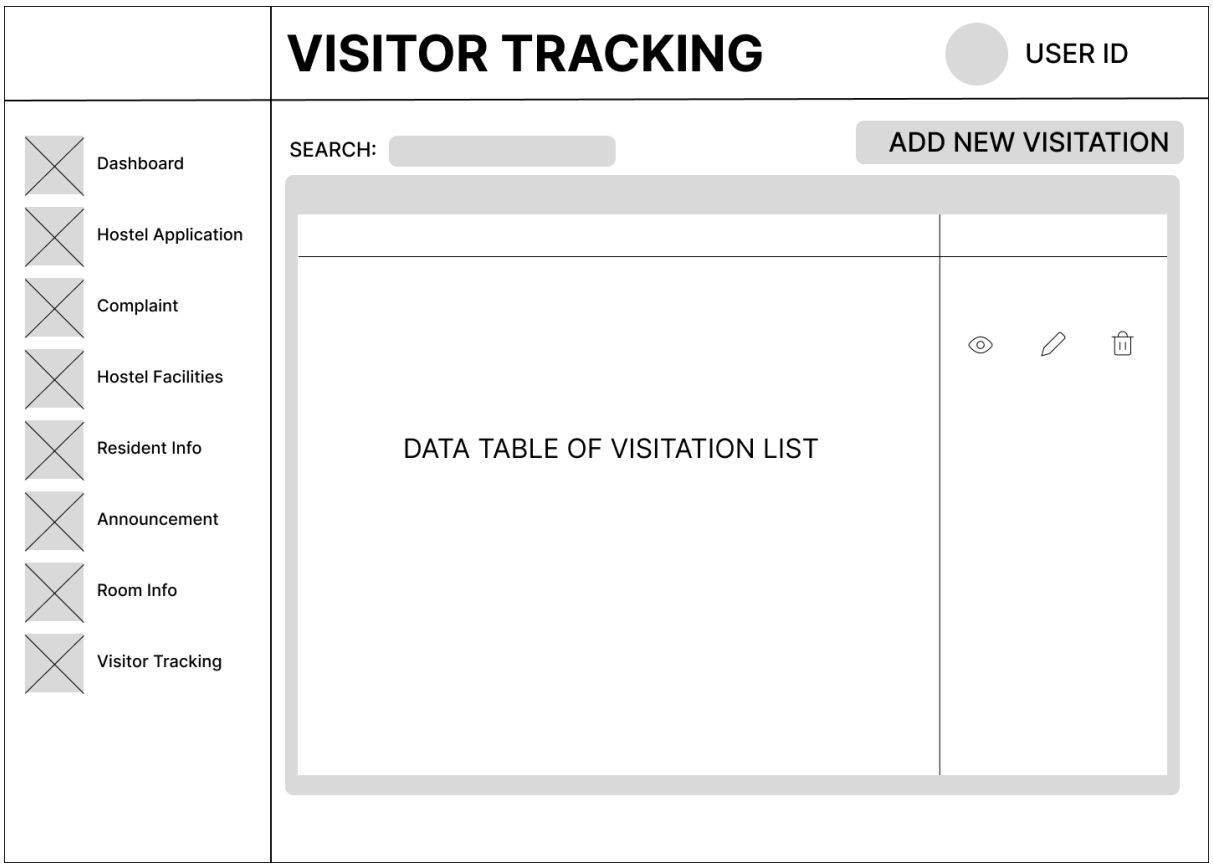


Figure 4.42: Wireframe of visitation record for admin panel

Figure 3.42 illustrates the wireframe of visitation record for admin panel. admin can add, view, edit and delete the visitation by clicking the corresponding button and icon.

Figure 4.43: Wireframe of resident information record for admin panel

Figure 3.43 illustrates the wireframe of resident information record for admin panel. Admin can add, view, edit and delete the resident record by clicking the corresponding button or icon.

3.5 Summary

This chapter provided detailed explanations of the requirements analysis and design. By conducting a survey to gather the necessary requirements, the functions to be integrated into the proposed system are defined, ensuring convenience for the users. Through careful design, we can ensure an optimal user experience and develop a high-performance system that aligns with the requirements.

CHAPTER 4: IMPLEMENTATION

4.1 Introduction

This chapter discusses the implementation of the proposed hostel management system. It outlines the steps required to set up the development environment and configure the required frameworks. The implementation process involves both backend and frontend development, database configuration, and integration of essential features to ensure a fully functional system.

4.2 Environment Setup

Before starting the system development, it is essential to set up the required environment. Laravel will serve as the primary framework for developing the proposed Hostel Management System. Laragon will be used as the local development environment, providing both database and web server functionalities during the development phase. The Genetic Algorithm (GA)-based room allocation process will be implemented as backend logic and executed as a Laravel job**,** allowing it to run asynchronously in the background.

4.2.1 Laravel Setup

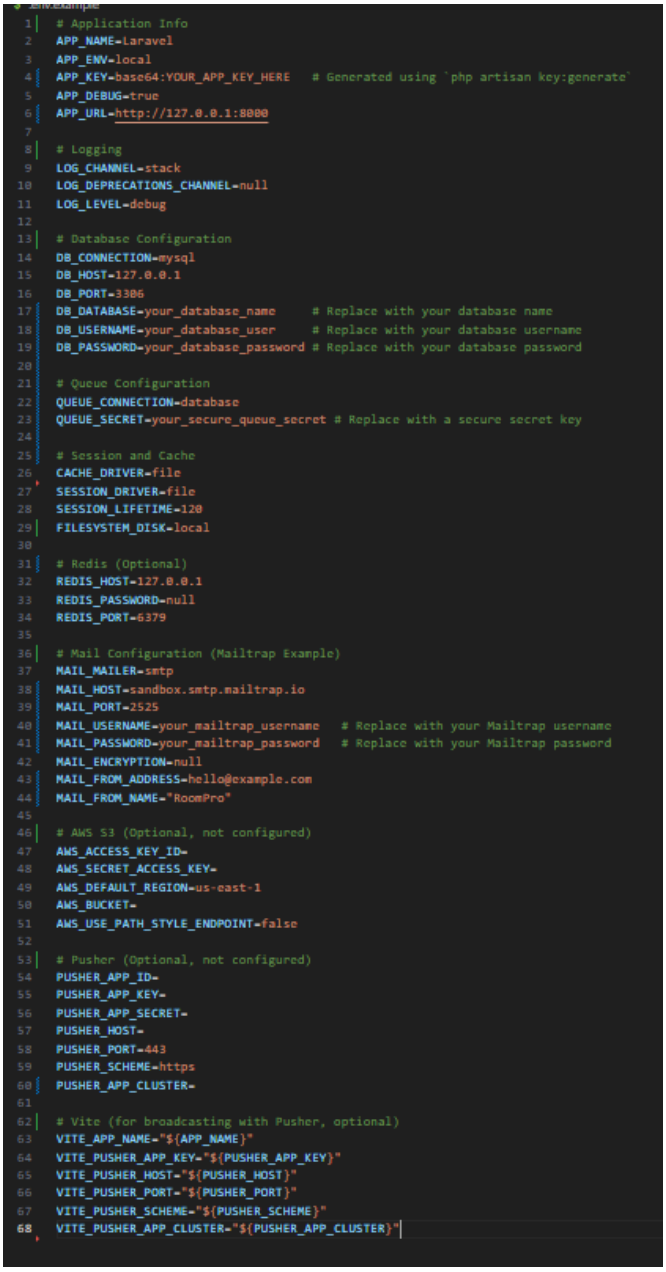
```
C:\laragon\www>composer create-project --prefer-dist laravel/laravel RoomProHMS
```

Figure 5.1: Command to create Laravel project

Figure 4.1 shows the command to create Laravel project. To create a project with Laravel framework, enter the command in the terminal, 'composer create-project --prefer-dist laravel/laravel RoomRroHMS' and a project folder will be created, before this composer need to be installed to support.

4.2.2 Laragon

During the development process, Laragon will be used as the local web server and database manager. Start Laragon by clicking the **"**Start**"** button, which will activate both the web server and MySQL services. To access the database, click the **"**Database**"** button, which opens phpMyAdmin in the browser. In phpMyAdmin, create a new database named hmssystem. After creating the database, configure the Laravel project to connect to it by updating the .env file in the project root directory. Modify the database settings and setup the environment variable as showed in figure 2.



```
# Application Info
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:YOUR_APP_KEY_HERE   # Generated using `php artisan key:generate`
APP_DEBUG=true
APP_URL=http://127.0.0.1:8000

# Logging
LOG_CHANNEL=stack
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

# Database Configuration
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=your_database_name     # Replace with your database name
DB_USERNAME=your_database_user     # Replace with your database username
DB_PASSWORD=your_database_password # Replace with your database password

# Queue Configuration
QUEUE_CONNECTION=database
QUEUE_SECRET=your_secure_queue_secret # Replace with a secure secret key

# Session and Cache
CACHE_DRIVER=file
SESSION_DRIVER=file
SESSION_LIFETIME=120
FILESYSTEM_DISK=local

# Redis (Optional)
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

# Mail Configuration (Mailtrap Example)
MAIL_MAILER=smtp
MAIL_HOST=sandbox.smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=your_mailtrap_username   # Replace with your Mailtrap username
MAIL_PASSWORD=your_mailtrap_password   # Replace with your Mailtrap password
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS=hello@example.com
MAIL_FROM_NAME="RoomPro"

# AWS S3 (Optional, not configured)
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=
AWS_USE_PATH_STYLE_ENDPOINT=false

# Pusher (Optional, not configured)
PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
PUSHER_HOST=
PUSHER_PORT=443
PUSHER_SCHEME=https
PUSHER_APP_CLUSTER=

# Vite (for broadcasting with Pusher, optional)
VITE_APP_NAME="${APP_NAME}"
VITE_PUSHER_APP_KEY="${PUSHER_APP_KEY}"
VITE_PUSHER_HOST="${PUSHER_HOST}"
VITE_PUSHER_PORT="${PUSHER_PORT}"
VITE_PUSHER_SCHEME="${PUSHER_SCHEME}"
VITE_PUSHER_APP_CLUSTER="${PUSHER_APP_CLUSTER}"
```

Figure 5.2: Environment variables setup

Figure 4.2 show the environment variable set up for the system. These variables allow the system to run securely and flexibly across different environments without hardcoding sensitive information.

4.2.3 Room Allocation based on Genetic Algorithm

To efficiently manage the large number of applications, the room allocation logic is implemented using a Genetic Algorithm (GA) within a Laravel background job**.** Running the

allocation in the background prevents users from having to remain on the page and wait for the process to complete, improving the system's responsiveness and user experience.

The core algorithm is defined in the handle() function of the Laravel job. It simulates the process of evolution by generating multiple possible solutions, known as chromosomes, and refining them across several generations to find the best room allocation arrangement. The algorithm uses the following parameters: a population size of 20 chromosomes per generation, a maximum of 5 generations (to avoid long execution times), a mutation rate of 0.05 to introduce variation and avoid local optima, and a stagnation limit of 3 generations. If no improvement is observed after three consecutive generations, the process is stopped early.

Each chromosome represents a potential allocation of applicants to rooms, and the initial population is generated randomly with considerations for gender and preferred room type. A fitness score is then calculated for each chromosome based on how well the assigned rooms match the applicants' preferences. These preferences include room type, block, and floor. Based on survey findings, room type is considered the most important factor, followed by block and then floor. Therefore, the fitness scoring is weighted as follows: a match in room type earns 3 points, block earns 2 points, and floor earns 1 point, giving a maximum score of 6 points per applicant.

The match percentage for each applicant is calculated using the formula (score / 6) * 100. For example, if a room matches the applicant's preferred type and block, but not the floor, the score would be 5, resulting in a match percentage of approximately 83.33%. If all preferences match, the percentage would be 100%. Once all applicants are evaluated, the algorithm calculates the overall match percentage for the entire batch by averaging the match percentages of all successfully assigned applications.

Additionally, the algorithm ensures that room assignments respect room capacity and gender restrictions. Rooms that are already full are excluded from further allocation, and earlier applications are prioritized to ensure fairness in the process. After the final solution is selected, the results are saved to the database, including the room assigned to each applicant and their individual match percentages. The overall match percentage for the batch is recorded, and the user who triggered the allocation is notified upon completion.

The coding for this GA based room allocation can be referred in the appendix 2.

4.3 System Implementation

This section will delve into the system implementation. The proposed system will feature four user panels: one for students who have registered an account but are not yet residents, one for residents, one for staff, and one for administrators.

4.3.1 Welcome, Login, Register, Forgot Password and Force Change Password Page

This section will discuss the authentication features, which include the welcome page, registration page, login page, forgot password page and force change password page.

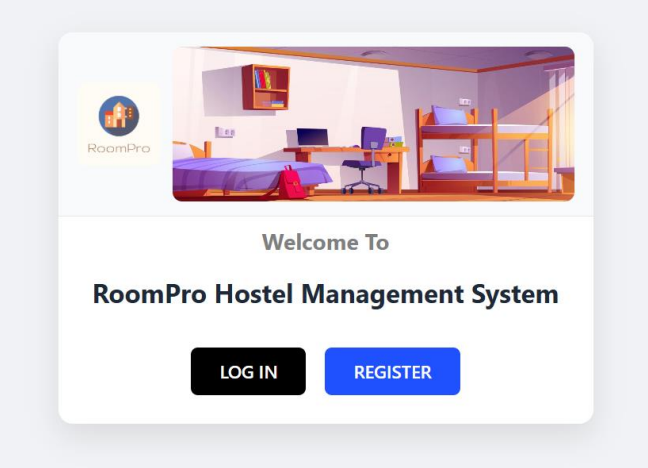


Figure 5.3: Welcome page for RoomPro HMS

Figure 4.3 shows the welcome page of the RoomPro Hostel Management System. Users can click the appropriate button to either log in or register.

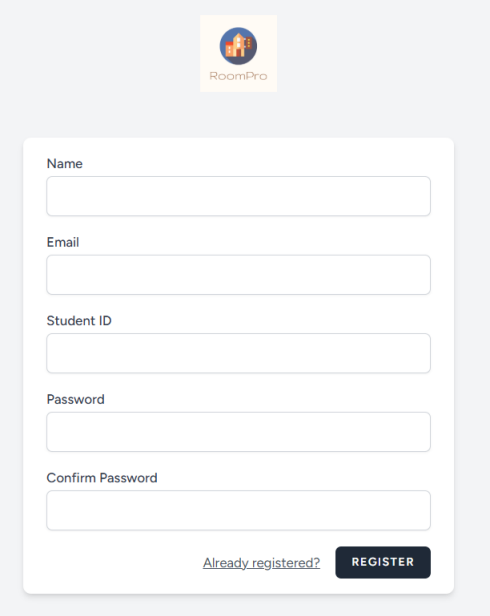


Figure 5.4: Registration Page

Figure 4.4 shows the registration page. To register, users must use their siswa email and create a password following the required format.

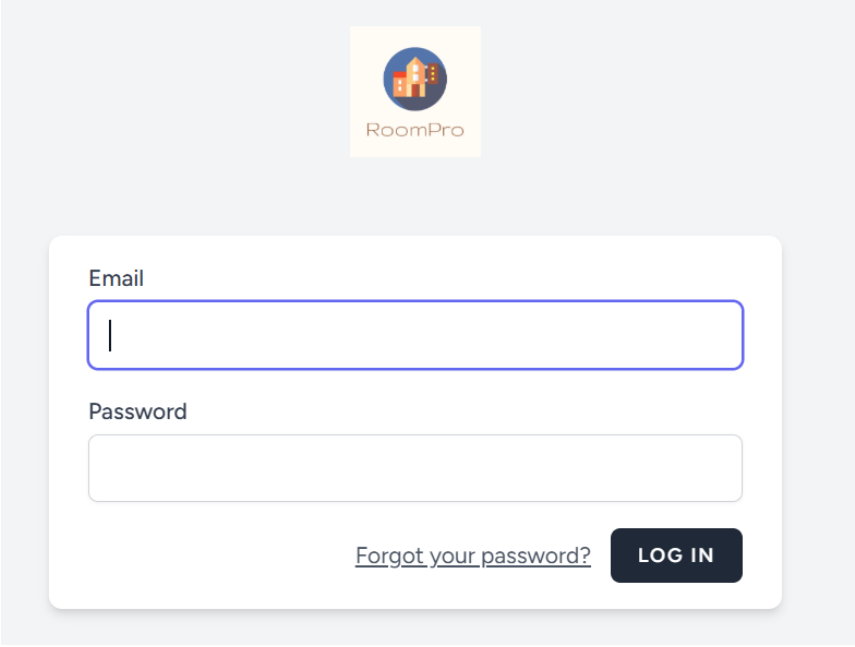


Figure 5.5: Login page

Figure 4.5 shows the login page. If the entered credentials are incorrect, an error message will be displayed to inform the user that the email or password is invalid.

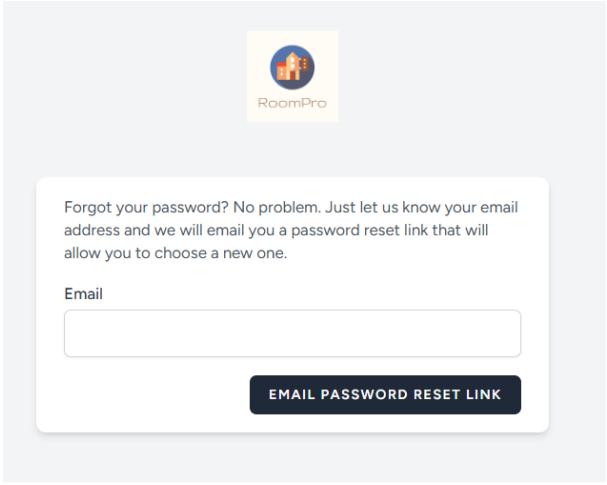


Figure 5.6: Forgot password page

Figure 4.6 shows the forgot password page. If a user forgets their password, they can click the “Forgot Password” option on the login page and enter their registered email address. The system will then send a password reset link to their registered email.

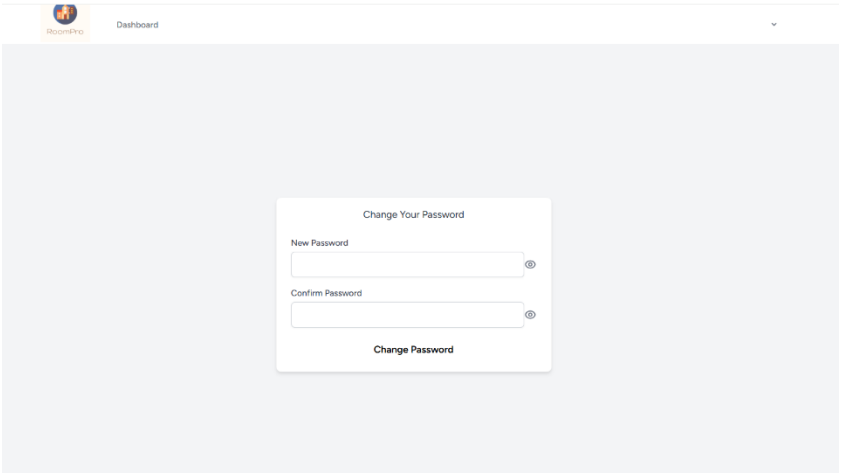


Figure 5.7: Force change password page

Figure 4.7 shows the force change password page. Staff members are required to enter a new password here to complete the password change process during their first login.

4.3.2 User Panel

This section will explain the implementation of the system and available functionalities for users who have registered but are not yet residents. Their user type is defined as “user”.

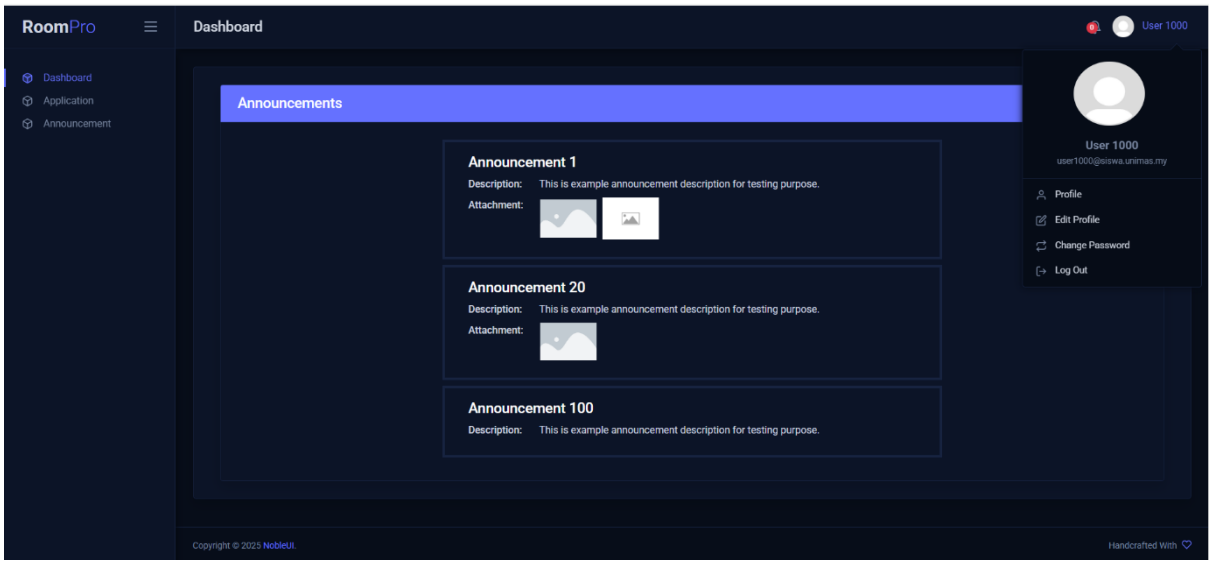


Figure 5.8: User dashboard

Figure 4.8 shows the dashboard for users with the "user" user type. These users have access only to the Announcement and Application sections. The header contains a notification icon and a profile section. By clicking on their profile picture, users can access a dropdown menu with options to view or edit their profile, change their password, and log out. The dashboard also displays the three most recent announcements for quick access. Users can click on an announcement title to be redirected to the full Announcement section.

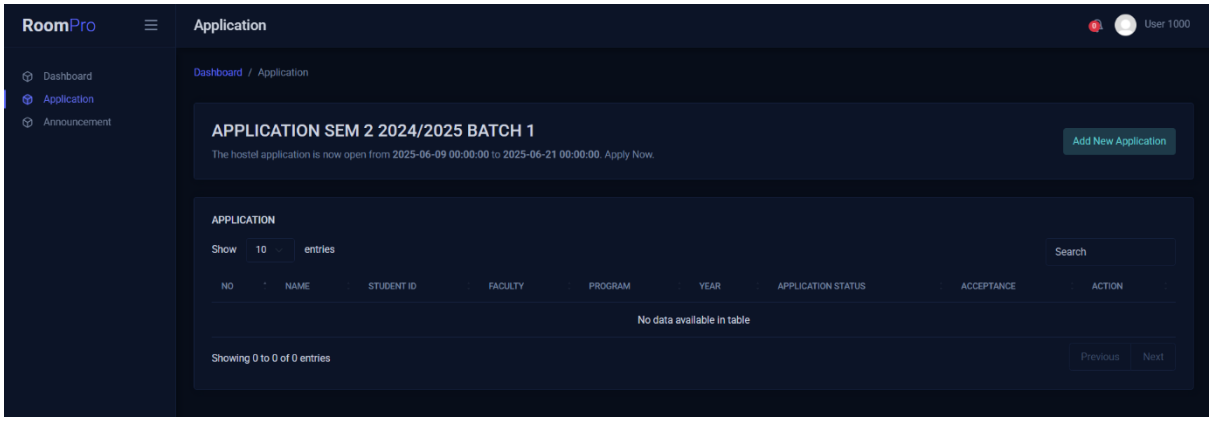


Figure 5.9: Main application page for user panel

Figure 4.9 shows the main application page for user panel, which contains a table listing all applications submitted by the logged in user. Above the table, a status message indicates whether the application period is open or closed. Next to the status message, there is an Add Application button that allows users to submit a new application.

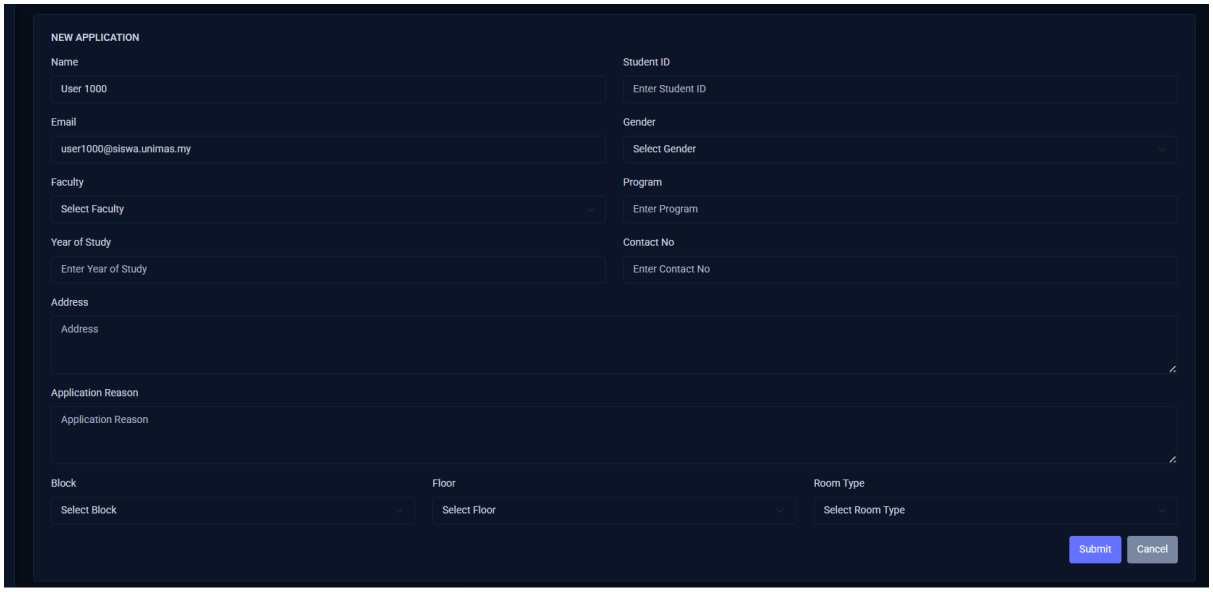


Figure 5.10: Application form

Figure 4.10 shows the application form, where users can fill in the required details and submit the form to apply.

4.3.3 Staff Panel

This section will explain the implementation of the system and available functionalities for hostel staff. Their user type is defined as “staff”.

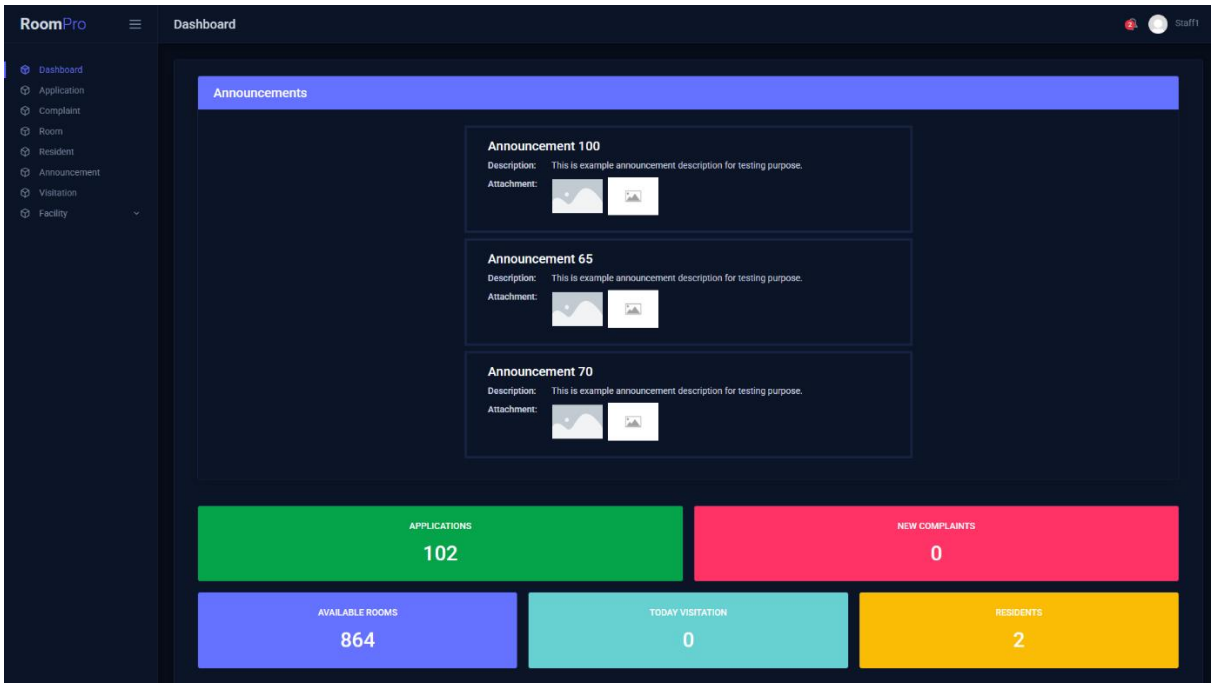


Figure 5.11: Staff dashboard

Figure 4.11 shows the dashboard for staff users. Staff have access to various sections, including Application, Complaint, Room, Resident, Announcement, Visitation, and Facility. In addition to the announcements, the dashboard displays five summary cards for quick access:

- **Application Card:** Shows the total number of submitted applications
- **Complaint Card:** Displays the number of new complaints
- **Room Card:** Shows the number of available rooms
- **Visitation Card:** Indicates the total number of visitations
- **Resident Card:** Displays the total number of residents for the current semester

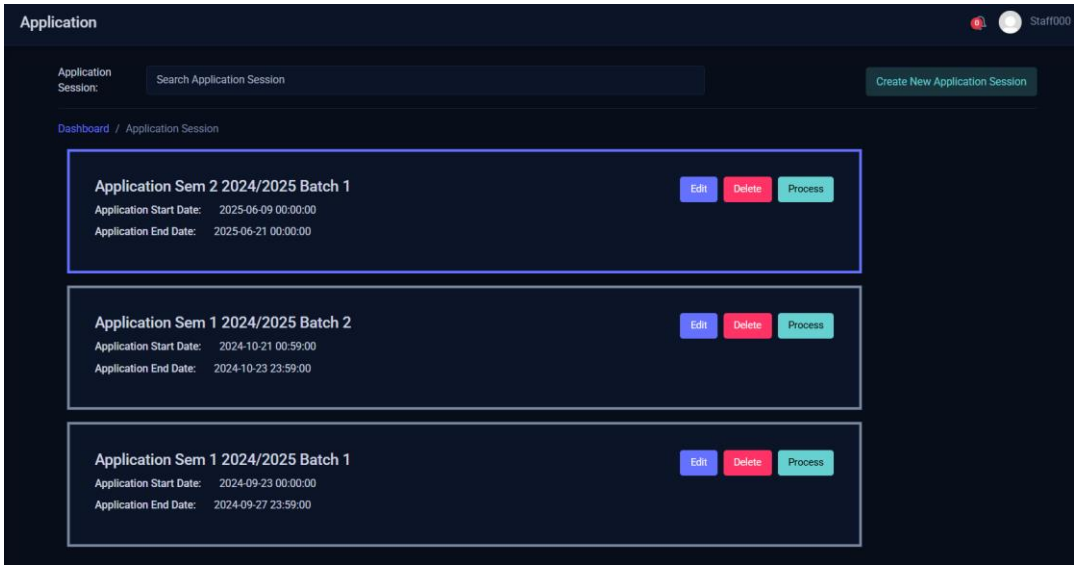


Figure 5.12: Application session

Figure 4.12 the application session management page. This page lists all application sessions, which categorize applications by batch. For example, first and second batch within a semester. Students who missed the first batch may apply during the second. Staff can click the "Process" button to navigate to the application list (Figure 13) and review applications submitted within a session.

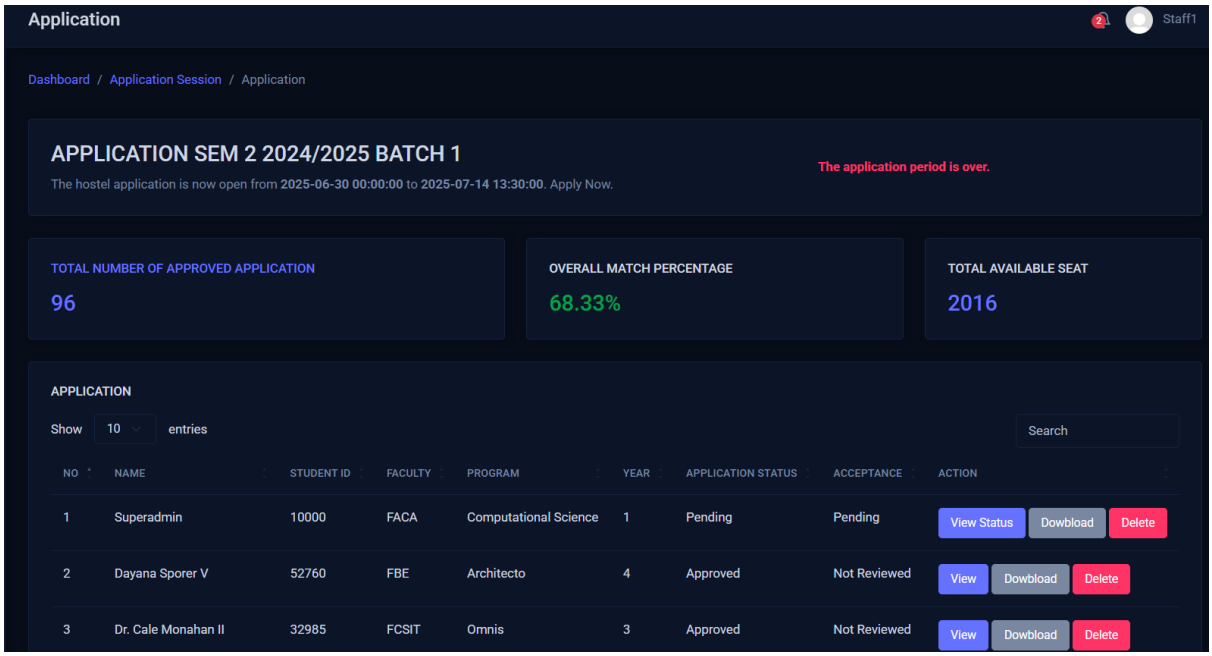


Figure 5.13: Main application page for staff panel

Figure 4.13 shows the main application page for the staff panel. This page includes a message indicating whether the application period is currently open or closed, along with a table listing all submitted applications. Positioned between these two elements are three key summary sections: the total number of approved applications, the overall match percentage of the room allocation process, and the total number of available room seats. The "Total Approved Applications" section not only displays the number of approved applications but also functions as a clickable link that redirects staff to the room allocation section. This is the only way to access the room allocation page, making it a critical navigation feature. Additionally, the "Total Available Room Seats" section helps ensure that the number of approved applications does not exceed the number of available seats.

Figure 5.14: Application approval page

Figure 4.14 show the application approval page. Staff may approve or reject applications. If rejected, a modal prompts them to enter a rejection reason.

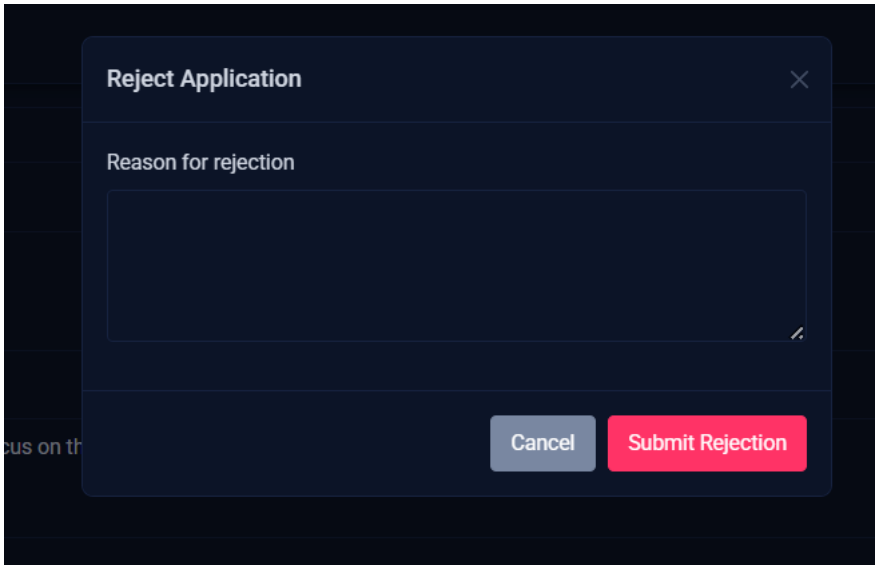


Figure 5.15: Rejection modal of application

Figure 4.15 show the rejection modal of application. If staff reject the application, this modal will pop up to prompt the user to enter the reason.

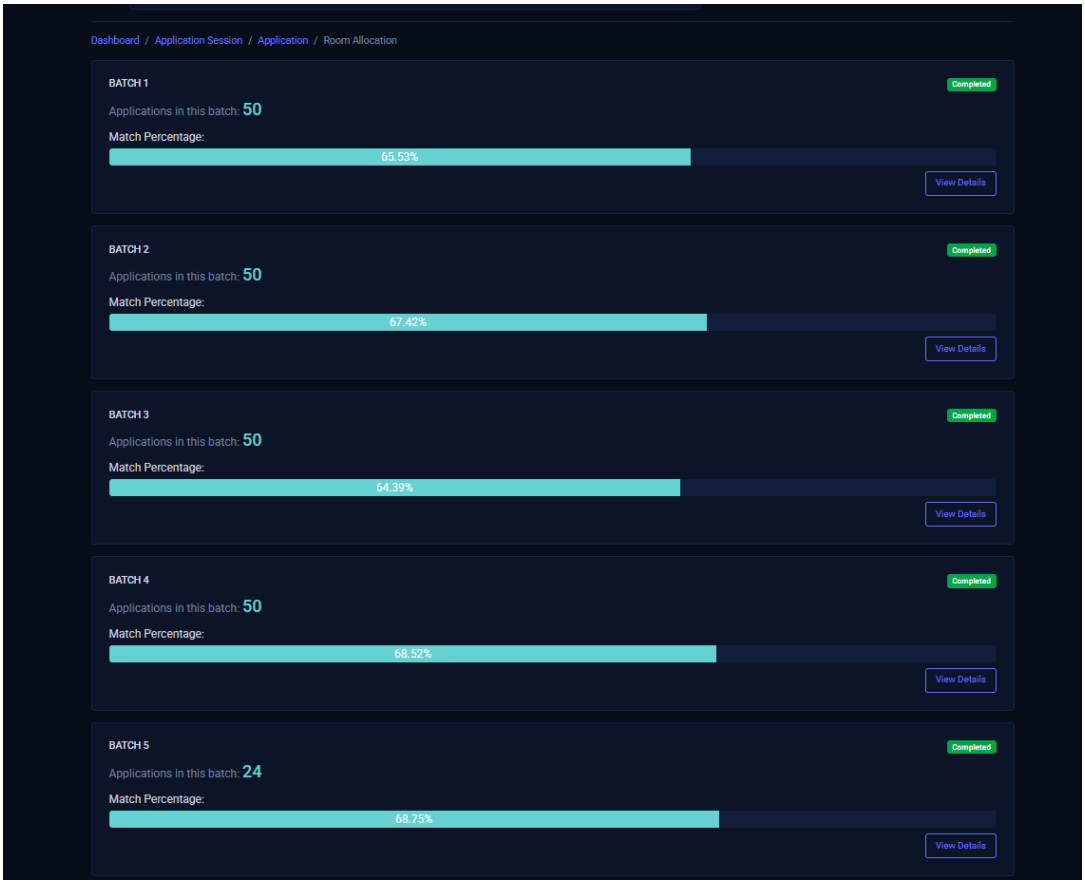


Figure 5.16: Main room allocation page

Figure 4.16 illustrates the main room allocation page. Approved applications are grouped into chunks (or batches), with each batch containing up to 50 applications. Staff can initiate the room allocation process for each batch by clicking the “Staff Allocation” button, which navigates them to the run room allocation page (Figure 98).

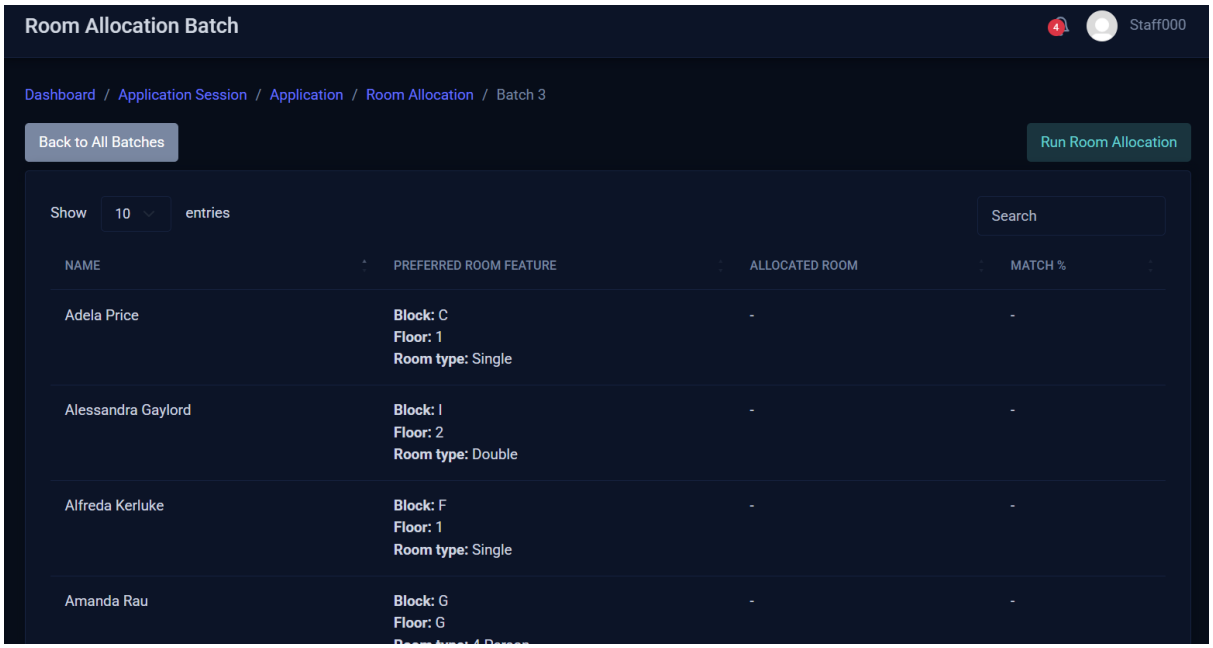


Figure 5.17: Run room allocation page before the allocation process has been run

Figure 4.17 show the run room allocation batch page. Clicking the “Run Room Allocation” button starts the Genetic Algorithm (GA) based room allocation process in the background. During this process, staff may continue navigating to other parts of the system without interrupting the allocation.

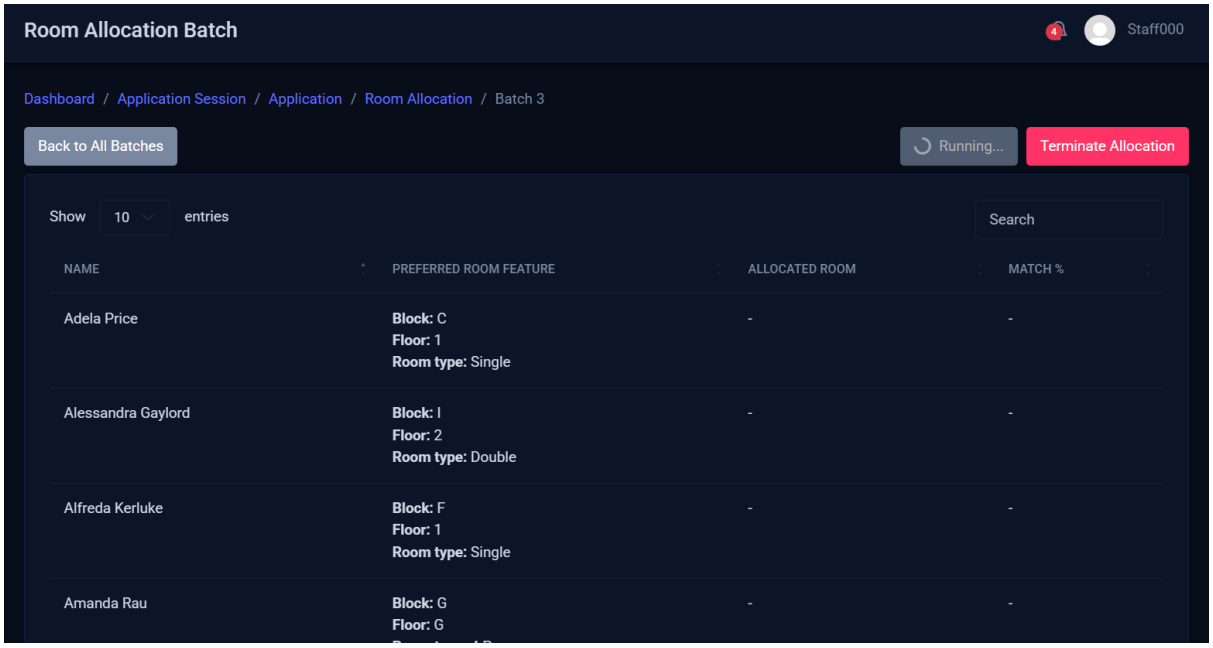


Figure 5.18: Room allocation process running page

Figure 4.18 shows the room allocation process running page. When the process is running, there will have an button "Running…" to indicate the process is currently running. While the process is running, staff may choose to terminate it by clicking the "Terminate Allocation" button.

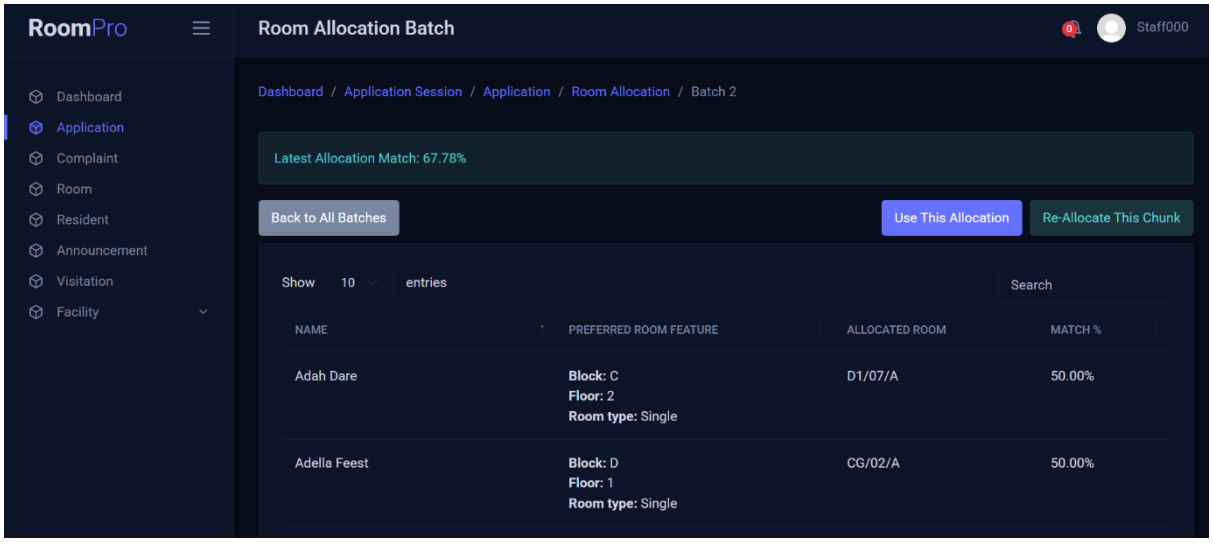


Figure 5.19: Room allocation process completed page

Figure 4.19 shows the page where the complete room allocation process is executed. Once the process is complete, matched rooms are displayed under the Allocated Room column.

Additionally, a column shows the match percentage between the allocated rooms and the applicants' preferred rooms. At the top of the table, the overall match percentage for that batch is displayed. If the results are unsatisfactory, they can click “Re-allocate” to run the process again. If the results are satisfactory, clicking the “Use This Allocation” button finalizes the room assignment.

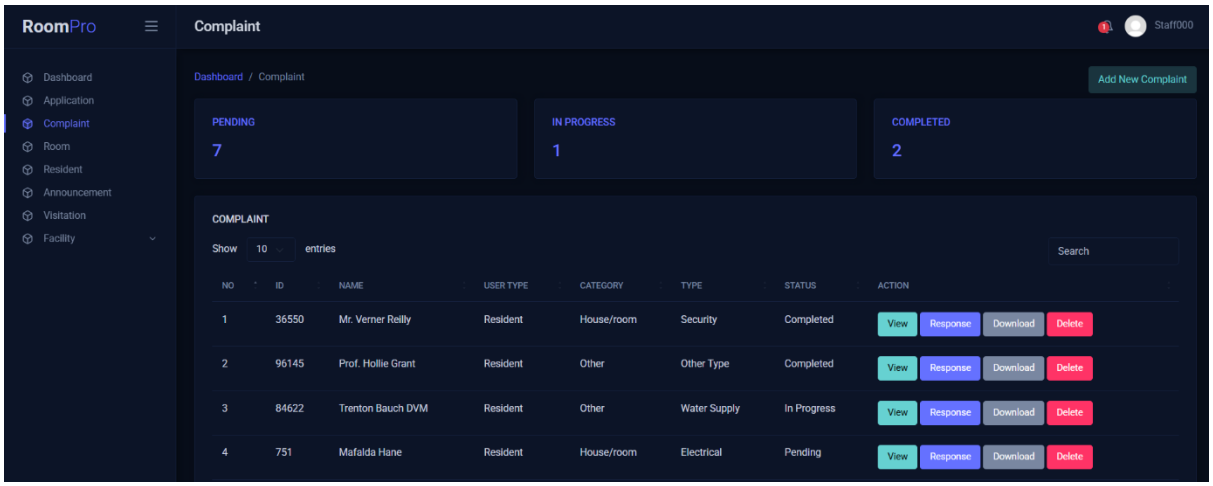


Figure 5.20: Main complaint page for staff panel

Figure 4.20 shows the main complaint management page. At the top, there are three filter sections: Pending, In Progress, and Completed. Below is a table listing all complaints. Staff can filter complaints based on their status by selecting one of the three categories.

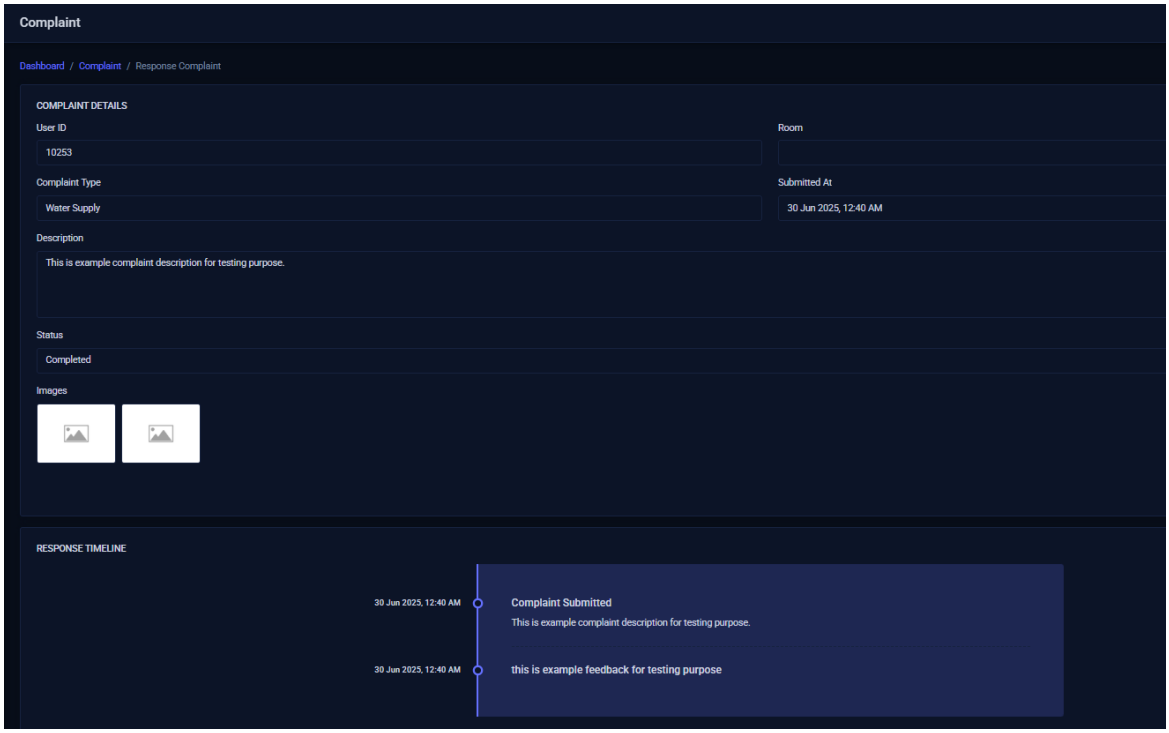


Figure 5.21: Complaint details page for staff panel

Figure 4.21 shows the complaint details page. The top section displays the submitted complaint information, followed by a feedback timeline. To receive a complaint, staff must click the “Receive” button to acknowledge it. For each new response, staff can click the “Response” button to provide feedback, which is then added to the timeline. When a complaint is resolved, staff click the “Complete” button to mark it as resolved.

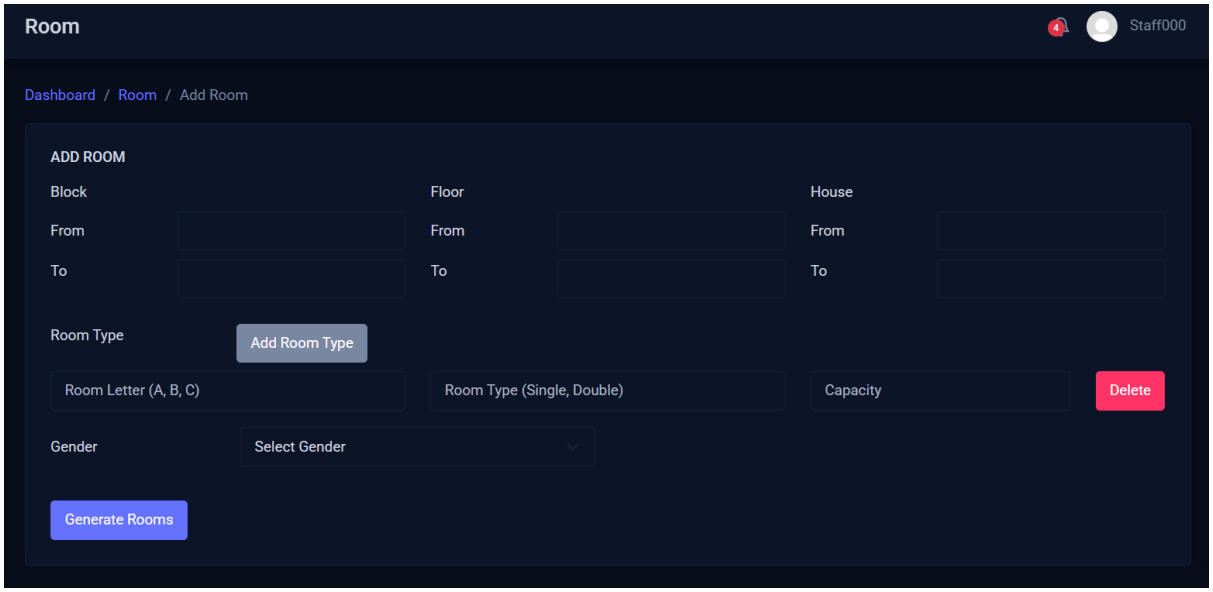


Figure 5.22: Create room page

Figure 4.22 shows the room generation page. Staff can generate a large number of rooms by specifying the range of blocks, floors, and room numbers. They can also assign the generated rooms as either male or female. Staff can define room types by clicking "Add Room Type" and entering details such as the room letter, type name, and capacity.

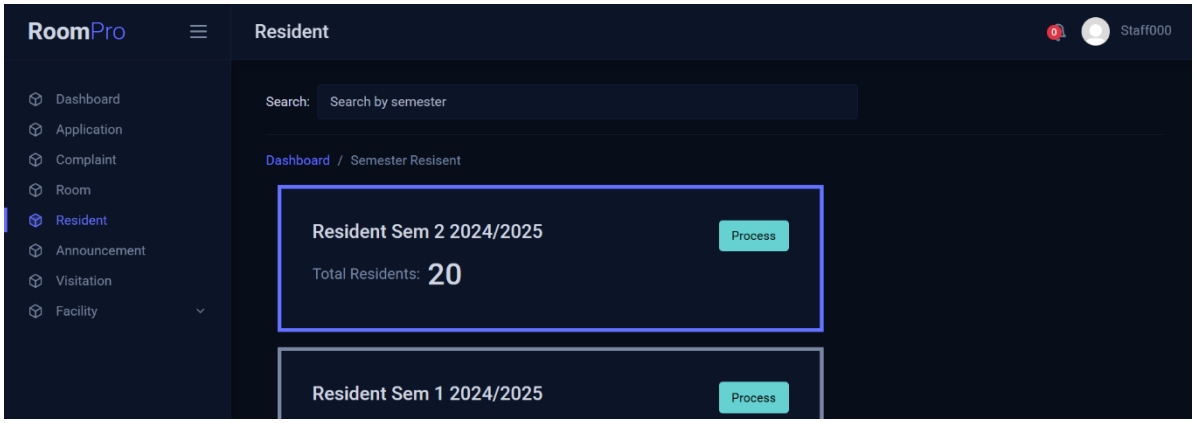


Figure 5.23: Main resident page for staff panel

Figure 4.23 shows the main resident management page. Residents are categorized by semester, and the total number of residents is displayed. Staff can click the *Process* button to view detailed resident information.

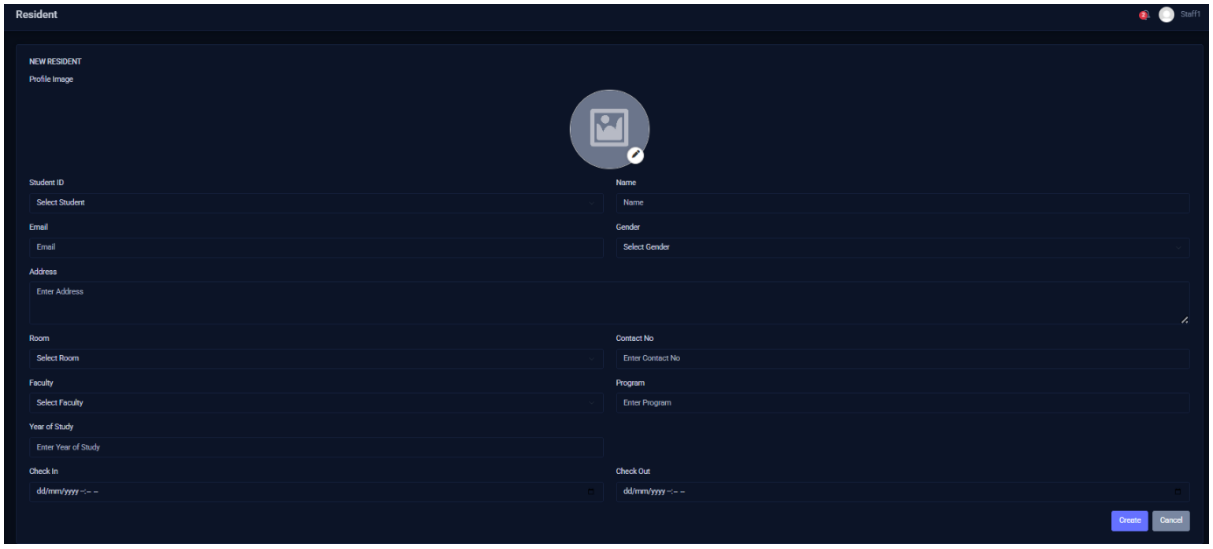


Figure 5.24: Manual add resident page for staff panel

Figure 4.24 shows the manual add resident page for staff panel. To manually add residents who did not apply during the application period, staff can click "Add Resident" and input the student's details and assign a room manually.

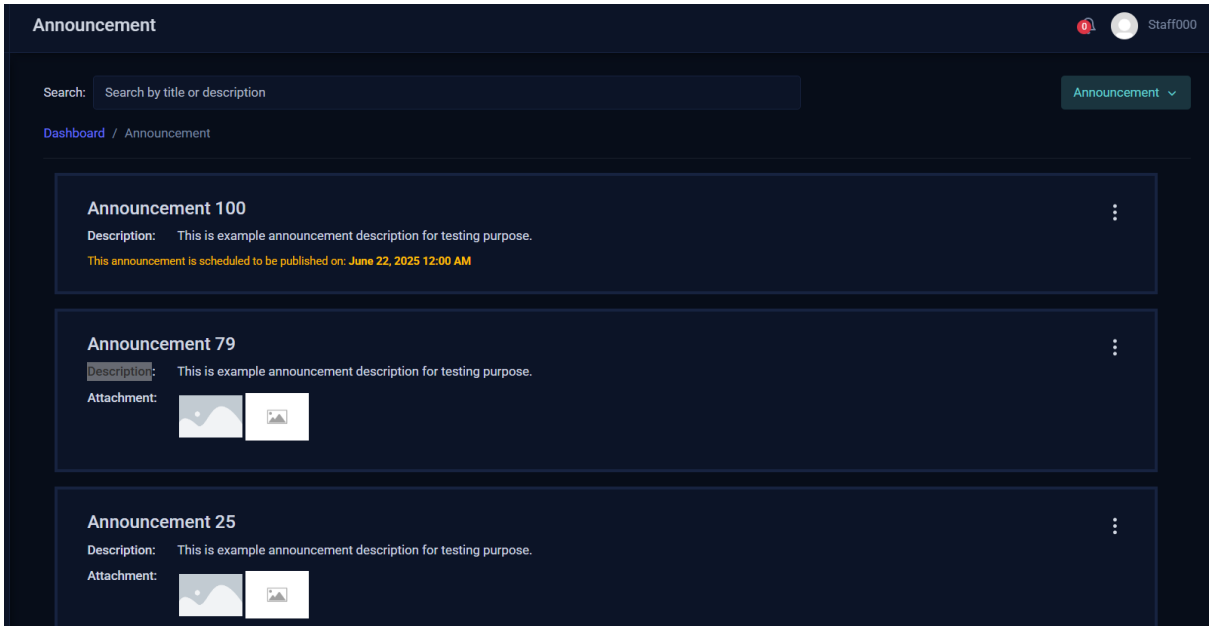


Figure 5.25: Main announcement page for staff panel

Figure 4.25 shows the main announcement page for staff panel. It displays both published and scheduled announcements. Staff can archive or unarchive announcements by clicking the three-dot menu and selecting the desired action.

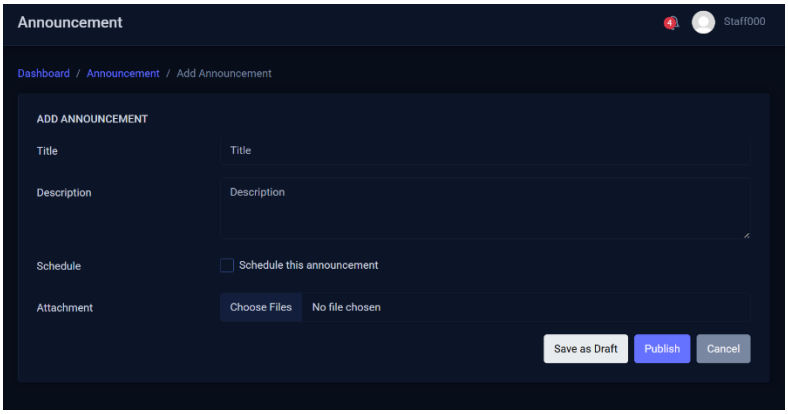


Figure 5.26: Add announcement page for staff panel

Figure 4.26 shows the Add Announcement page, where staff can create announcements, schedule them for later, save them as drafts, or publish them immediately.

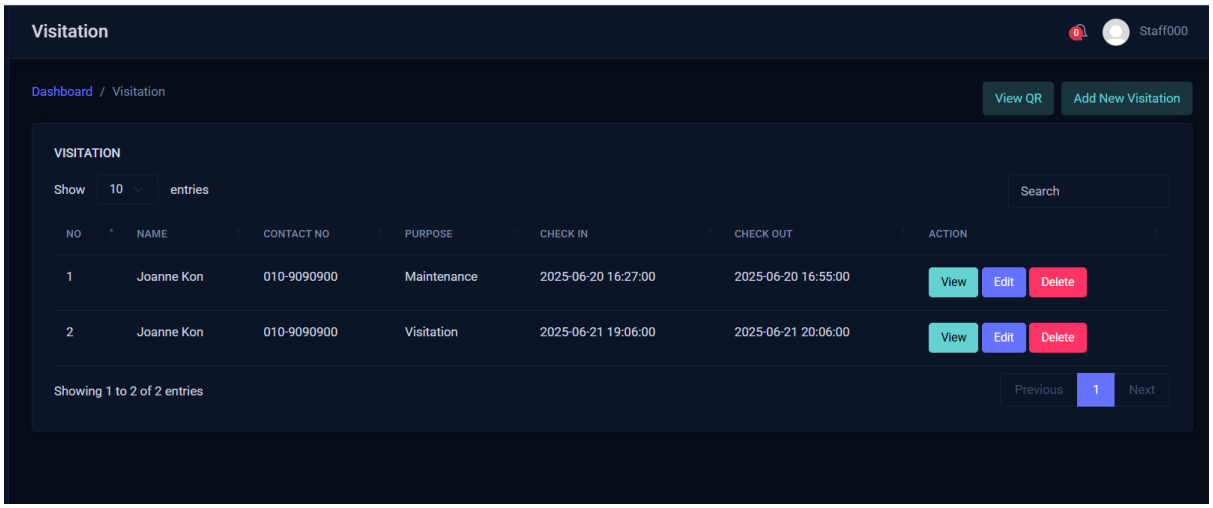


Figure 5.27: Main visitation page for staff panel

Figure 4.27 shows the visitation management page, which lists all recorded visits. Staff can click “View QR” to generate and print a QR code for visitors to scan and fill in their visit details. Staff can also manually add visit records by clicking the “Add New Visitation” button.

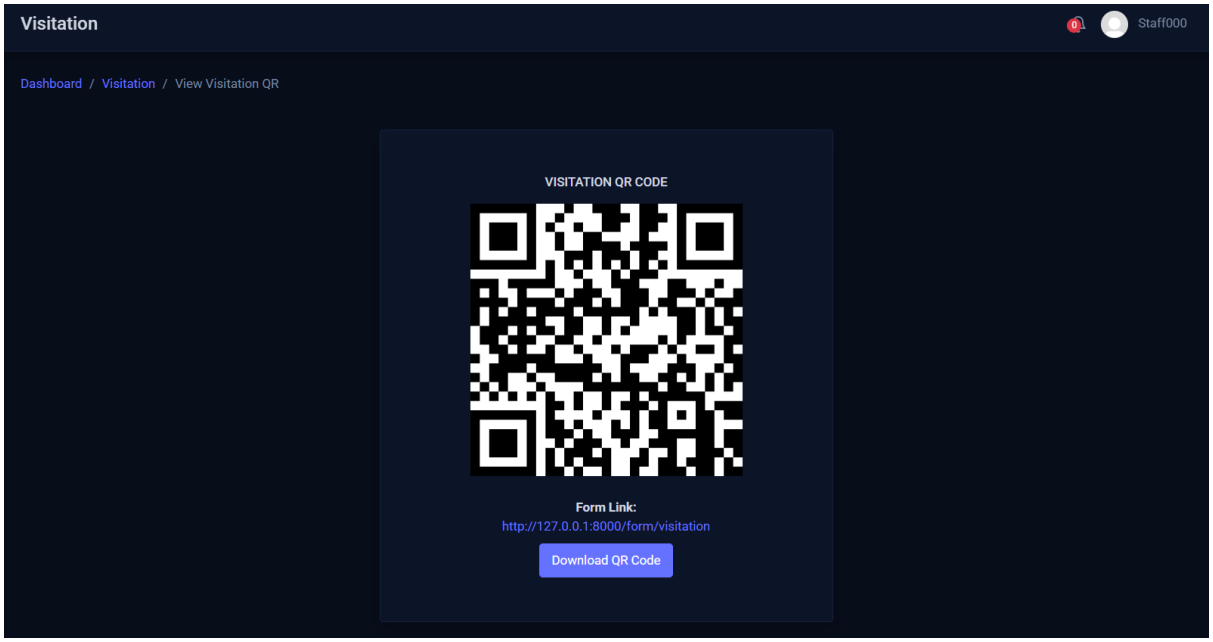


Figure 5.28: QR code for visitation form

Figure 4.28 shows the visitation QR code. Staff can download and print this QR code for external visitors to scan, allowing them to record their visit without logging into the system. Upon scanning the QR code, a form will appear for visitors to fill in their details. Additionally, a URL is displayed above the download button, which also directs users to the visitation form.

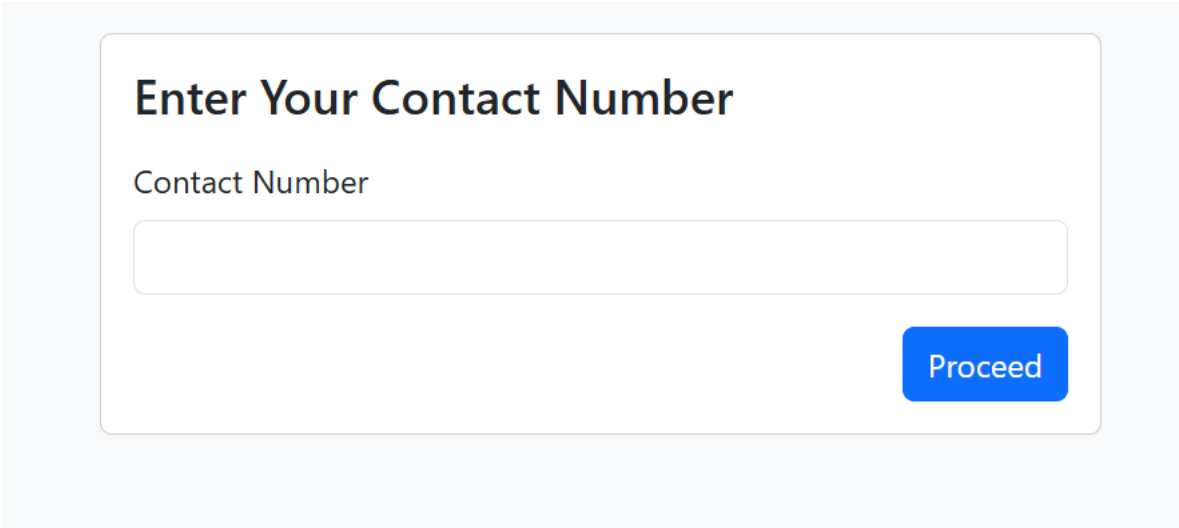


Figure 5.29: Enter contact number page for visitor to fill the visitation form

Figure 4.29 shows the enter contact number page. This page will appear after the visitor scan the QR code. Before filling in the form, the visitor is required to enter their contact number. This step is used to check whether the visitor has already filled in the form on the same day. If no previous submission is found for that day, a blank form will be displayed, and the visitor will be required to complete all fields, including the check-in time. However, if a record already exists for the same contact number on the same date, the previously filled form will be retrieved, allowing the visitor to update their check-out time. If the same contact number is entered but the date is different, the system will consider it a new visit and display a blank form accordingly.

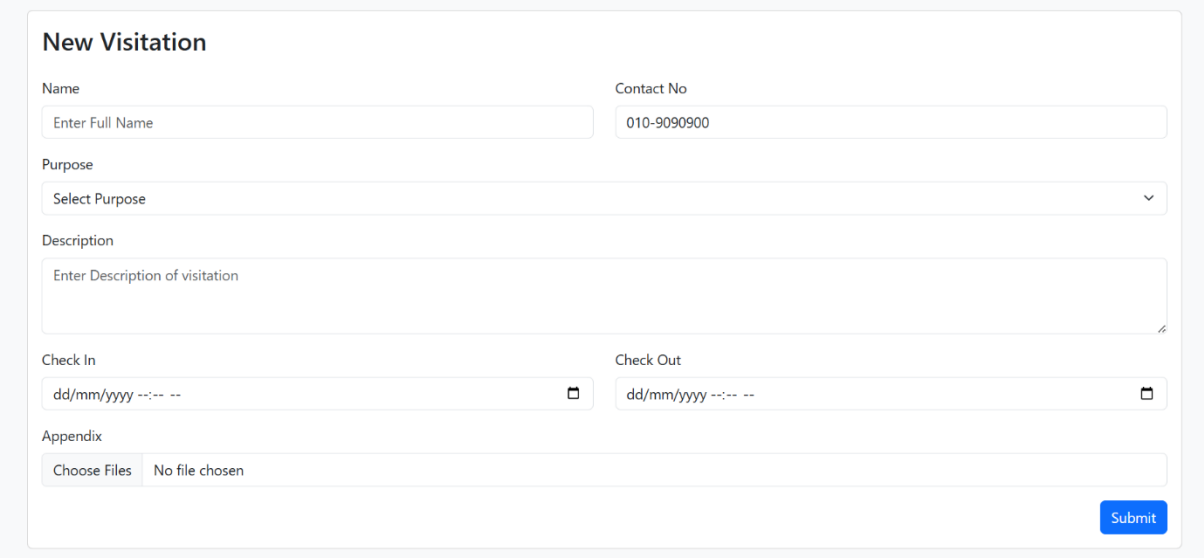


Figure 5.30: Public visitation form

Figure 4.30 shows the public visitation form, which appears after the visitor enter the contact number.

Figure 5.31: Facility page for staff panel

Figure 4.31 shows the main facility management page. Staff can view, edit, or delete facilities.

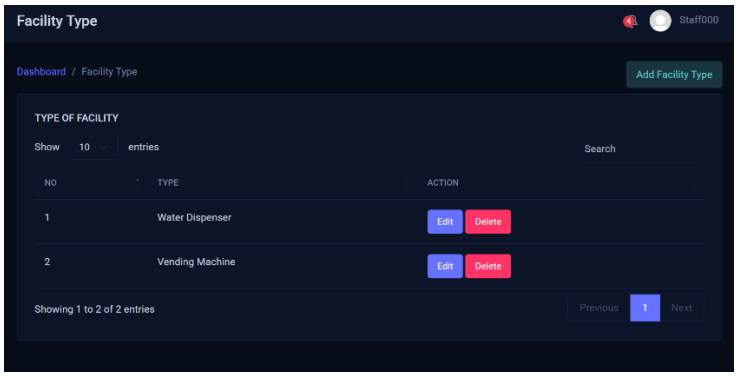


Figure 5.32: Facility type page for staff panel

Figure 4.32 shows the Facility Type page, where staff can add new types of facilities for classification.

4.3.4 Resident Panel

This section will explain the implementation of the system and available functionalities for residents. Their user type is defined as “resident”. The user’ user type will change for “user” to “resident” after they accept the hostel offer.

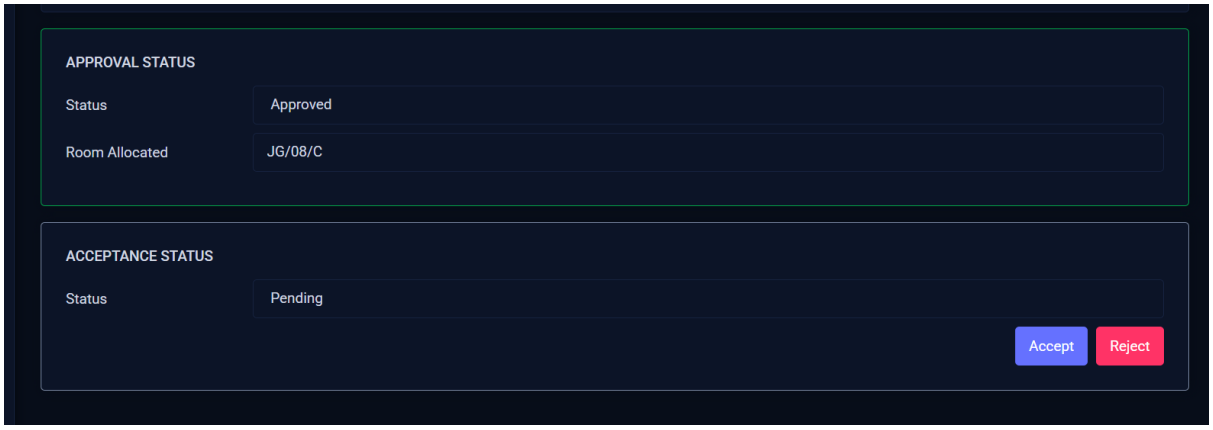


Figure 5.33: Application acceptance page

Figure 4.33 shows the application acceptance page for applicants. When an application's status is updated, a notification (Figure 18) is sent to inform users of their application's status. Users can choose to accept or reject the offer by clicking the respective button. If they reject, a modal will appear prompting them to enter the reason. Once an offer is accepted, the user's role is updated from user to resident, and they are granted access to resident features, including Resident, Complaint, and Facility sections.

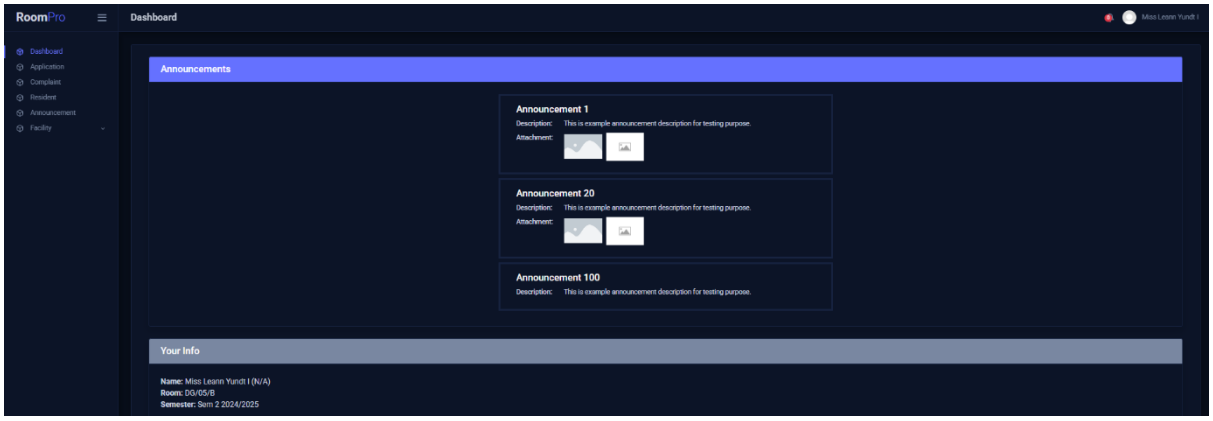


Figure 5.34: Resident dashboard

Figure 4.34 shows the dashboard for residents. Below the announcements section, the resident's information is displayed.

Figure 5.35: Resident information page

Figure 4.35 shows the resident's personal information page. The user's own details are displayed at the top, followed by the details of their housemates. Residents can check in and check out by clicking the respective button.

Figure 5.36: Main complaint page for resident panel

Figure 4.36 shows a table listing complaints submitted by the resident.

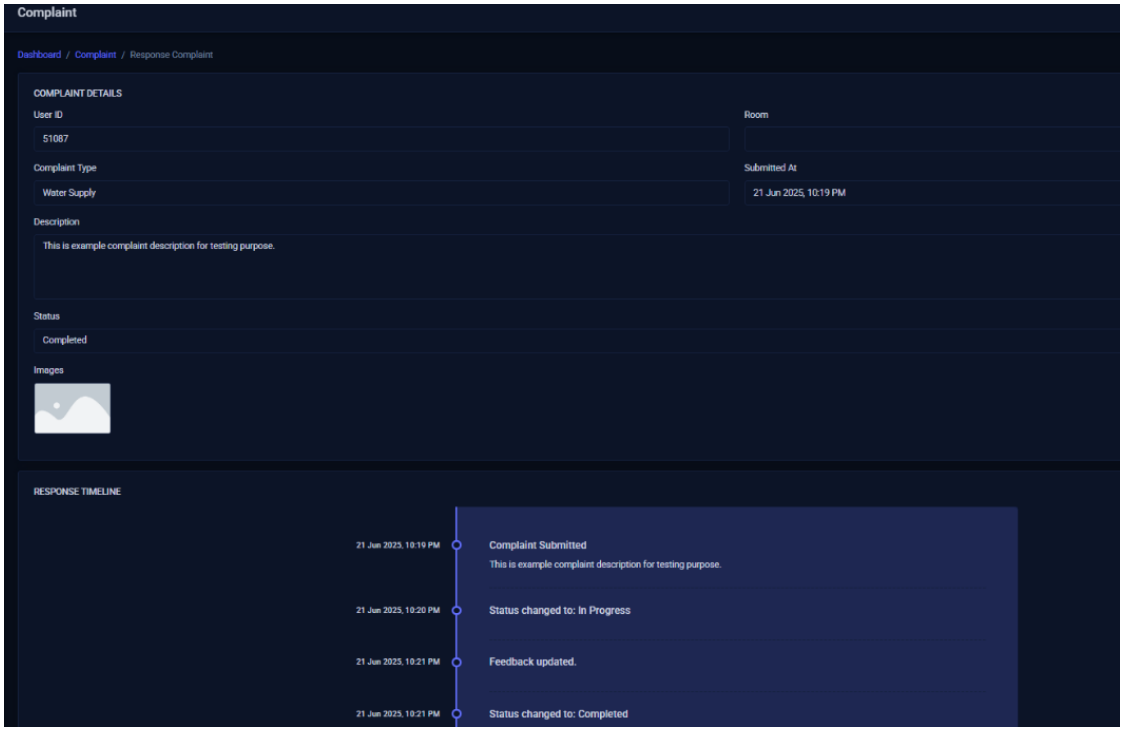


Figure 5.37: Complaint response page for resident panel

Figure 4.37 displays the complaint response page. Residents can report a complaint and view staff responses. Notifications are sent to residents when feedback is provided. They can view these in the Notification section.

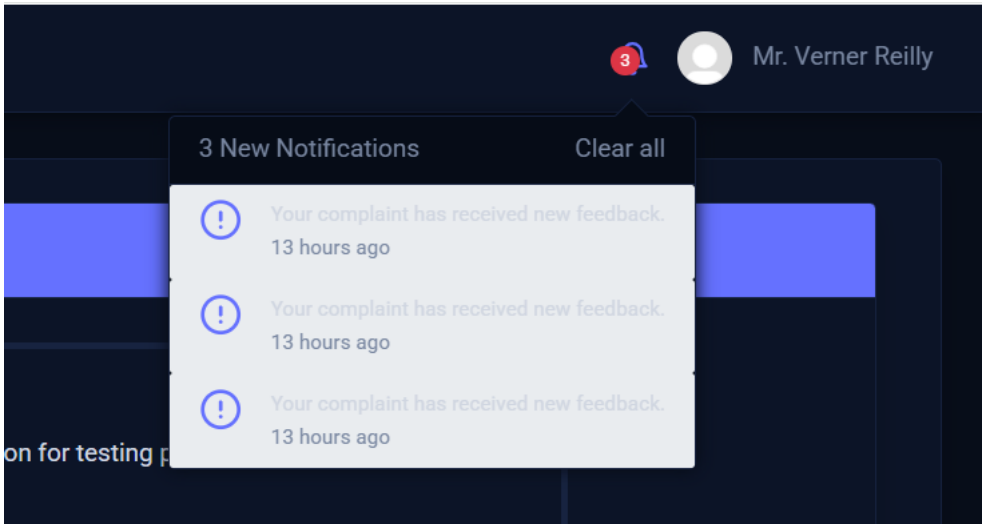


Figure 5.38: Complaint feedback update notification of resident panel

Figure 4.38 shows the complaint notification for the resident panel. Each time feedback is updated, a notification is sent to the complaint submitter.

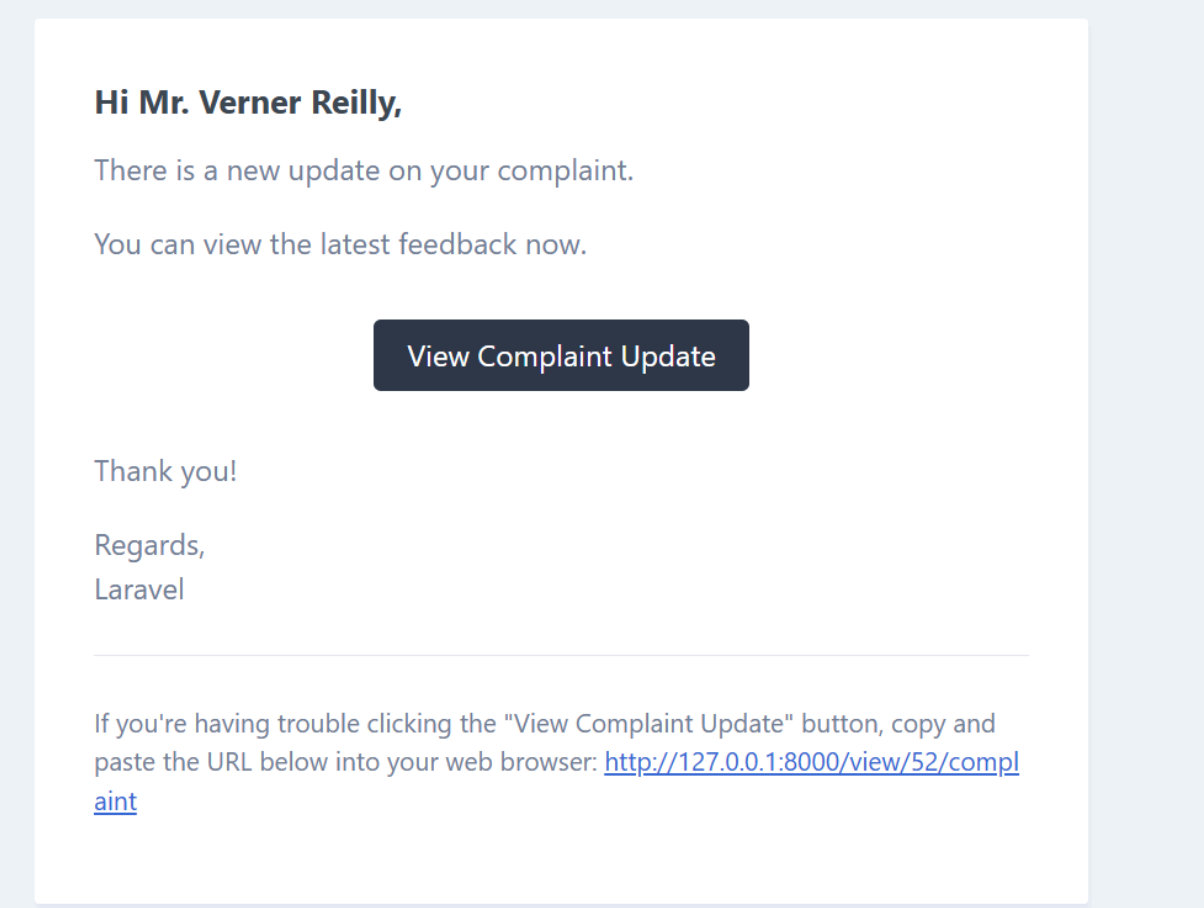


Figure 5.39: Email sent when complaint feedback updated

Figure 4.39 shows the email sent when complaint updated. Email when be sent to the complaint submitter each time the complaint feedback was updated.

4.3.4 Admin Panel

This section explains the implementation of the system and the additional functionalities that are accessible only to system administrators. Users with the role of "admin" have elevated privileges that allow them to manage and control various aspects of the system. The admin user is created during the development phase to facilitate administrative operations and testing.

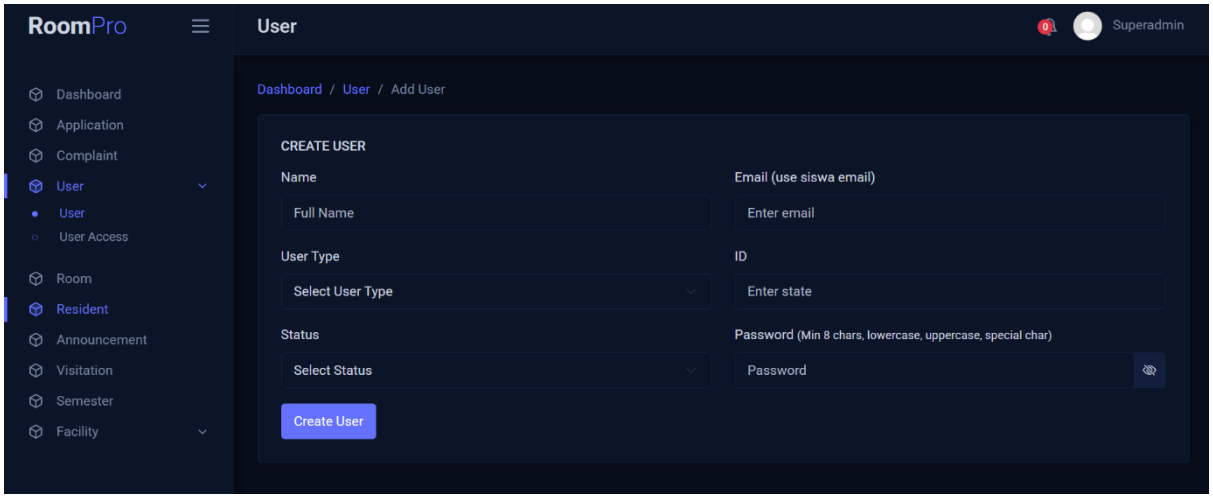


Figure 5.40: Create user page for admin panel

Figure 4.40 shows the create user page in the admin panel. This page is used to create staff accounts for the hostel system. Accounts are created with a predefined password. When staff log in for the first time using these credentials, they will be required to change their password.

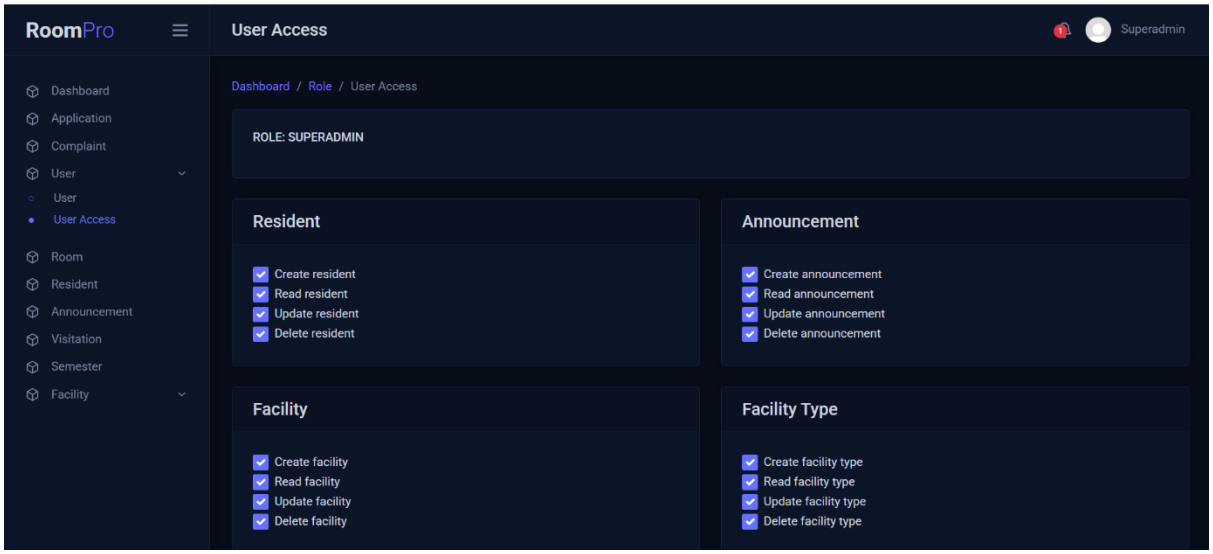


Figure 5.41: Ability management page for admin panel

Figure 4.41 shows the ability management page in the admin panel. On this page, the admin assigns abilities based on user roles.

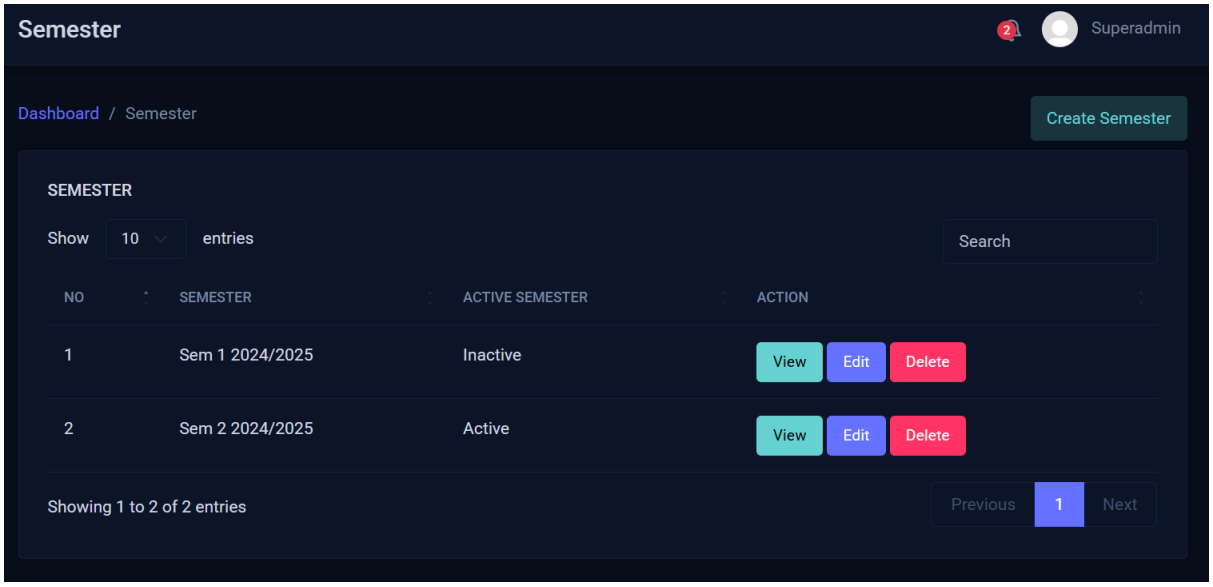


Figure 5.42: Main semester page for admin page

Figure 4.42 shows the main semester page. This section is used to create new semesters. The "Active" label indicates the current semester.

4.4 Summary

In conclusion, this chapter has presented the overall implementation process of the proposed Hostel Management System. The key system functionalities have been clearly demonstrated through the implementation of various pages and user interfaces. All essential features and components have been successfully integrated, ensuring that the system operates as intended.

CHAPTER 5: TESTING

5.1 Introduction

This chapter covers system functionality testing and usability testing to ensure the system runs correctly and to identify any errors for debugging and improvement.

5.2 Functional Testing

Functional testing is carried out to verify that all features of the proposed hostel management system work as intended. The goal is to detect any existing or hidden errors and confirm that the system performs as expected. This is achieved by conducting test cases for the main sections of the system.

Table 6.1: Test case of user registration

Test Case ID: user_registration Test Priority (Low/Medium/High): High Module Name: Registration Test Title: Verify registration process Description: Test the functionality of user registration				Test Designed By: Joanne Test Designed Date: 19/6/2025 Test Executed By: Joanne Test Executed Date: 19/6/2025		
Pre-condition: • User does not have an account of the proposed system.						
Scenario	Test Case	Testing Procedure	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Verify successful registration with valid data	1. Enter valid information. 2. Click the 'REGISTER' button.	Username: Joanne Kon Xin Yue Email:79684@siswa.unimas.my Student ID: 79684 Password: U79684u# Confirm Password: U79684u#	Successfully register an account and redirect to the dashboard with a success alert message displayed.	Successfully register an account and redirect to the dashboard with a success alert message displayed.	Pass
2	Verify unsuccessful registration with invalid data	1. Enter invalid information.	Username: Joanne Kon Xin Yue Email:joanne@email.com	Unsuccessfully register an account and validation	Unsuccessfully register an account and validation	Pass

		2. Click the 'REGISTER' button.	Student ID: 79684 Password: U79684u# Confirm Password: U79684u#	message displayed.	message displayed.	
3	Verify unsuccessfully registration with register	1. Enter with registered email. Click the 'REGISTER' button.	Username: Joanne Kon Xin Yue Email:79684@siswa.unimas.my Student ID: 79684 Password: U79684u# Confirm Password: U79684u#	Unsuccessfully register an account and validation message displayed.	Unsuccessfully register an account and validation message displayed.	Pass
4	Verify unsuccessfully registration with invalid password format	1. Enter invalid password. Click the 'REGISTER' button.	Username: Joanne Kon Xin Yue Email: 79684@siswa.unimas.my Student ID: 79684 Password: 12345678 Confirm Password: 12345678	Unsuccessfully register an account and validation message displayed.	Unsuccessfully register an account and validation message displayed.	Pass
Post-condition: • User successfully registers with valid data. • Registers user data is securely stored in system' database or storage.						

The test case in table 5.1 outlines the user registration feature. It ensures users can register with a valid siswa email and a correctly formatted password. If invalid data is entered, such as an incorrect email format or a weak password, the system should display validation errors.

Table 6.2: Test case of user login

Test Case ID: user_login Test Priority (Low/Medium/High): High Module Name: User Login Test Title: Verify login process Description: Test the functionality of user login				Test Designed By: Joanne Test Designed Date: 19/6/2025 Test Executed By: Joanne Test Executed Date: 19/6/2025		
Pre-condition: • User have registered an account. • User credentials are stored securely in the system's database or storage. • User's status is active.						
Scenario	Test Case	Testing Procedure	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Verify successful login with valid data	1. Enter valid information. 2. Click the 'LOG IN' button.	Email:79684@siswa.unimas.my Password: U79684u#	Successfully login and redirect to the dashboard based on user type.	Successfully login and redirect to the dashboard based on user type.	Pass
2	Verify unsuccessful login with invalid data	3. Enter invalid information. 4. Click the 'LOG IN' button.	Email:79684@ unimas.my Password: U79684u#	Unsuccessfully login and validation message displayed.	Unsuccessfully login and validation message displayed.	Pass
3	Verify unsuccessfully login with not registered email	2. Enter with not registered email. Click the 'LOG IN' button.	Username: Joanne Kon Xin Yue Email:70000@siswa.unimas.my Password: U79684u#	Unsuccessfully login and validation message displayed.	Unsuccessfully login and validation message displayed.	Pass

4	Verify unsuccessfully login with invalid password	3. Enter invalid password. Click the 'LOG IN' button.	Email:79684@siswa.unimas.my Password: U79684u@	Unsuccessfully login and validation message displayed.	Unsuccessfully login and validation message displayed.	Pass
Post-condition: • User successfully login and access to the system.						

The test case in table 5.2 outlines the user login feature. It ensures users can log in with valid credentials. If incorrect credentials are provided, appropriate validation errors should be displayed.

Table 6.3: Test case of application section

Test Case ID: user_make_application Test Priority (Low/Medium/High): High Module Name: Hostel application process Test Title: Verify hostel application Description: Test the functionality of making a hostel application				Test Designed By: Joanne Test Designed Date: 19/6/2025 Test Executed By: Joanne Test Executed Date: 19/6/2025		
Pre-condition: • User is logged into the system. • Semester data have been created. • Room data have been created.						
Scenario	Test Case	Testing Procedure	Input Data	Expected Result	Actual Result	Status (Pass/Fail)

1	Verify successful application session creation	1. Log in as staff. 2. Create a session with valid semester, application batch, and application/acceptance start and end dates. 3. Click the 'Create' button.	Valid semester session, application batch, application and acceptance start and end dates	Application session is successfully created.	Application session is successfully created.	Pass
2	Verify unsuccessful application session creation	1. Log in as staff. 2. Attempt to create a session with invalid application or acceptance dates. 3. Click the 'Create' button.	Invalid semester session, application batch, application and acceptance start and end dates	Application session creation fails with validation error.	Application session creation fails with validation error.	Pass
3	Verify successful application submission	1. Log in as a registered user. 2. Enter valid application data. 3. Click the 'Submit' button.	Valid application data	Application is successfully submitted and an alert message appears.	Application is successfully submitted and an alert message appears.	Pass
4	Verify unsuccessful application submission with invalid data	1. Log in as a registered user. 2. Enter invalid application data. 3. Click the 'Submit' button.	Invalid application data.	Application submission fails with validation errors displayed.	Application submission fails with validation errors displayed.	Pass
5	Verify successful application approval update	1. Log in as staff. 2. Navigate to the application session. 3. Select a pending application. 4. Choose to approve or reject (enter reason if rejecting).	Application ID, status, reason	Application status is updated to approved or rejected, with no room assigned.	Application status is updated to approved or rejected, with no room assigned.	Pass
6	Verify unsuccessful application	1. Log in as staff. 2. Navigate to an already approved or rejected application.	Already processed application ID	Approve or Reject buttons are not visible.	Approve or Reject buttons are not visible.	Pass

	approval update	3. Confirm that the approve/reject buttons are not visible.		Application status cannot be updated again.	Application status cannot be updated again.	
7	Verify successful room allocation for approved applications	1. Log in as staff. 2. Navigate to the room allocation section. 3. Click the 'Process' button for a batch of approved applications. 4. Click the 'Run Room Allocation' button. 5. View allocation results.	Approved application IDs, available room IDs	Rooms are allocated to approved applications. Room name and match percentage are displayed for each application.	Rooms are allocated to approved applications. Room name and match percentage are displayed for each application.	Pass
8	Verify successful offer acceptance update	1. Log in as the user. 2. Navigate to the application session. 3. If the application is approved, accept or reject the offer (enter a reason if rejecting).	Submitted application ID	Acceptance status is updated. If accepted, the user type changes to "resident" and resident access is granted.	Acceptance status is updated. If accepted, the user type changes to "resident" and resident access is granted.	Pass
9	Verify unsuccessful offer acceptance update	1. Log in as user/resident. 2. Navigate to an already processed application. 3. Confirm that the Accept/Reject buttons are not visible.	Already processed application ID.	Accept or reject button are hidden. The acceptance status unchanged.	Accept or reject button are hidden. The acceptance status unchanged.	Pass

Post-condition:

- Applications are submitted and stored in the database.
- Application status and acceptance status are updated and stored.
- Rooms are allocated to each approved application and stored in the database.
- If the offer is accepted, the user's type is updated from "user" to "resident" and resident access is granted.

The test case in table 5.3 outlines the hostel application process, covering key functionalities such as application session creation, application submission, approval or rejection of applications, room allocation using a genetic algorithm, and offer acceptance by users. The test case ensures that both valid and invalid scenarios are handled correctly, and that the system responds appropriately by updating application statuses, allocating rooms, and managing user roles upon acceptance.

Table 6.4: Test case of complaint section

Test Case ID: user_make_complaint Test Priority (Low/Medium/High): High Module Name: Hostel complaint process Test Title: Verify hostel complaint functionality Description: Test the functionality of submitting and updating hostel complaint				Test Designed By: Joanne Test Designed Date: 19/6/2025 Test Executed By: Joanne Test Executed Date: 19/6/2025		
Pre-condition: • User is logged in as resident. • Resident has an approved application and has accepted the hostel offer.						
Scenario	Test Case	Testing Procedure	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Verify successful complaint submission	1. Log in as a resident. 2. Navigate to the complaint section. 3. Enter valid complaint details. 4. Click the 'Submit' button.	Valid complaint data.	Complaint is successfully submitted and an alert message displayed.	Complain is successfully submitted and an alert message displayed.	Pass
2	Verify unsuccessful	1. Log in as a resident. 2. Navigate to the complaint section.	Invalid complaint data.	Complaint submission fails	Complaint submission fails	Pass

	complaint submission	3. Enter invalid complaint details. 4. Click the 'Submit' button.		and validation error messages are displayed.	and validation error messages are displayed.	
3	Verify complaint feedback update	1. Log in as staff. 2. Navigate to the complaint section. 3. Select a complaint where status is 'Pending' or 'In Progress'. 4. Enter feedback for complaint.	Submitted complaint ID, valid feedback data	Complaint feedback is successfully saved and displayed in the timeline. Email and notification sent to the resident.	Complaint feedback is successfully saved and displayed in the timeline. Email and notification sent to the resident.	Pass
Post-condition: • Complaints are submitted and stored in the database. • Complaint feedback is updated and tracked in the system. • Resident are notified via email and in-app notification each time feedback is provided.						

The test case in table 5.4 outlines the hostel complaint process, focusing on the submission of complaints by residents and the updating of complaints feedback by staff. It ensures the system correctly handles both valid and invalid complaints submissions and allows staff to provide feedback. The system is expected to store all complaints, track feedback updates and notify residents accordingly.

Table 6.5: Test case of facility section

Test Case ID: create_facility Test Priority (Low/Medium/High): High Module Name: Facility	Test Designed By: Joanne Test Designed Date: 19/6/2025 Test Executed By: Joanne

Test Title: Verify facility functionality Description: Test the functionality of facility section				Test Executed Date: 19/6/2025		
Pre-condition: • User is logged in as staff. • At least one facility type has been created.						
Scenario	Test Case	Testing Procedure	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Verify successful facility creation	1. Log in as a staff. 2. Navigate to the facility section. 3. Enter valid facility information. 4. Click the 'Create' button.	Valid facility data	Facility is successfully created and an alert message is displayed.	Facility is successfully created and an alert message is displayed.	Pass
2	Verify unsuccessful facility creation	1. Log in as a staff. 2. Navigate to the facility section. 3. Enter invalid facility information. 4. Click the 'Create' button.	Invalid facility data	Facility creation fails and validation errors messages are displayed.	Facility creation fails and validation errors messages are displayed.	Pass
Post-condition: • Facilities are created and stored in the database. • Resident can view facility details for reference.						

The test case in table 5.5 outlines the facility management process, focusing on the creation of facilities by staff. It ensures the system handles both valid and invalid input correctly, allowing successful creation of facility records and displaying appropriate validation errors when necessary.

Table 6.6: Test case of announcement section

Test Case ID: create_announcement Test Priority (Low/Medium/High): High Module Name: Announcement Test Title: Verify announcement functionality Description: Test the functionality of announcement section				Test Designed By: Joanne Test Designed Date: 19/6/2025 Test Executed By: Joanne Test Executed Date: 19/6/2025		
Pre-condition: • User is logged in as staff.						
Scenario	Test Case	Testing Procedure	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Verify successful announcement creation	1. Log in as staff. 2. Navigate to the announcement section. 3. Enter valid announcement information. 4. (Optional) Set a schedule. 5. Click 'Publish' or 'Save as Draft'.	Valid announcement data	Announcement is created successfully and an alert message is shown. If scheduled, it will only be visible to staff until the publish time is reached.	Announcement is created successfully and an alert message is shown. If scheduled, it will only be visible to staff until the publish time is reached.	Pass
2	Verify unsuccessful announcement creation	1. Log in as staff. 2. Navigate to the announcement section. 3. Enter invalid or	Invalid announcement data	Annoucnement Announcement creation fails. Validation error messages are displayed.	Announcement creation fails. Validation error messages are displayed.	Pass

		missing data. 4. Click 'Publish' or 'Save as Draft'.				
3	Verify announcement archive or unarchive	1. Log in as staff. 2. Navigate to the announcement section. 3. Click 'Archive' on a published announcement. 4. Optionally, click 'Unarchive'.	Published announcement ID	Announcement is archived successfully and hidden from users. When unarchived, it becomes visible again based on its original status (published/scheduled/draft).	Announcement is archived successfully and hidden from users. When unarchived, it becomes visible again based on its original status (published/scheduled/draft).	Pass
Post-condition: • Announcements are created and stored in the database. • Published announcement are visible to all users depending on status and schedule. • Archived announcements are hidden from view but retained for future unarchiving.						

The test case in table 5.6 verifies the functionality of the announcement section, including creating, scheduling, publishing, archiving, and unarchiving announcements. It ensures that announcements are properly validated, stored, and displayed to users based on their status and schedule, and that archived announcements are hidden but can be restored as needed.

Table 6.7: Test case of resident section

Test Case ID: manage_resident Test Priority (Low/Medium/High): High Module Name: Resident	Test Designed By: Joanne Test Designed Date: 19/6/2025 Test Executed By: Joanne

Test Title: Verify resident functionality Description: Test the functionality of resident section				Test Executed Date: 19/6/2025		
Pre-condition: • User is logged in as staff or resident. • Resident has approved application and has accepted hostel offer.						
Scenario	Test Case	Testing Procedure	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Verify successful manual resident creation for walk in student	1. Log in as staff. 2. Navigate to the Resident section. 3. Enter valid resident data. 4. Click the 'Create' button.	Valid resident data	Resident is successfully created and an alert message is displayed.	Resident is successfully created and an alert message is displayed.	Pass
2	Verify unsuccessful manual resident creation for walk in section	1. Log in as staff. 2. Navigate to the Resident section. 3. Enter invalid or incomplete resident data. 4. Click 'Create'.	Invalid resident data	Resident creation fails and validation errors messages are displayed.	Resident creation fails and validation errors messages are displayed.	Pass
3	Verify successful view resident and housemate details	1. Log in as a resident. 2. Navigate to the Resident section. 3. View their own resident details and housemate information.	Resident ID, room ID	Resident and housemate information are displayed.	Resident and housemate information are displayed.	Pass
4	Verify successful resident check in and check out	1. Log in as resident. 2. Navigate to resident section. 3. Enter valid check in or check out data.	Valid check in or check out data	Resident is successfully checked in or out and an alert	Resident is successfully checked in or out and an alert	Pass

				message is displayed.	message is displayed.	
5	Verify unsuccessful resident check in and check out	1. Log in as resident. 2. Navigate to resident section. 3. Enter invalid check in or check out data.	Invalid check in or check out data.	Resident fails to check in or check out and validation error message is displayed.	Resident fails to check in or check out and validation error message is displayed.	Pass
Post-condition: • Residents are created and stored in the database. • Resident and housemates details are accessible to the logged in user.						

The test case in table 5.7 verifies the functionality of the resident section, including manual resident creation for walk-in students, viewing of resident and housemate details, and resident check-in and check-out. It ensures that valid data is processed correctly while invalid input is handled with proper validation, maintaining accurate resident records and access control.

Table 6.8: Test case of user section

Test Case ID: create_user_and_manage_user_acess Test Priority (Low/Medium/High): High Module Name: User and user access Test Title: Verify user and user access Description: Test the functionality of user and user access	Test Designed By: Joanne Test Designed Date: 19/6/2025 Test Executed By: Joanne Test Executed Date: 19/6/2025
Pre-condition: • A superadmin account has been created and stored in the database.	

• Abilities are created and store in the database.						
Scenario	Test Case	Testing Procedure	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Verify successful staff creation	1. Log in as superadmin. 2. Navigate to the user section. 3. Enter valid staff information. 4. Click the 'Create' button.	Valid user data	User is successfully created and an alert message is displayed.	User is successfully created and an alert message is displayed.	Pass
2	Verify unsuccessful user creation	1. Log in as superadmin. 2. Navigate to the user section. 3. Enter invalid user information. 4. Click the 'Create' button.	Invalid user data	User creation fails and validation errors messages are displayed.	User creation fails and validation errors messages are displayed.	Pass
3	Verify successful user access update	1. Log in as superadmin. 2. Navigate to the user access section. 3. Update user access.	Role ID, ability ID	User access successfully updated and an alert message is displayed.	User access successfully updated and an alert message is displayed.	Pass
4	Verify successful admin forces staff to change password on login	1. Staff log in use the credentials from admin. 2. Change password.	User ID	Password is successfully changed.	Password is successfully changed.	Pass
Post-condition: • Staff are created and stored in the database. • User roles and access permissions are updated accordingly. • Staff created by admin are required to change their password upon first login.						

The test case in table 5.8 verifies the functionality of the user and user access sections, including staff account creation, user access management and forced password change upon first login. It ensures that the superadmin can create users with valid data, assign roles and abilities and enforce security measures to protect system access.

Table 6.9: Test case of visitation section

Test Case ID: manage_visitation Test Priority (Low/Medium/High): High Module Name: Visitation Test Title: Verify visitation functionality Description: Test the functionality of visitation				Test Designed By: Joanne Test Designed Date: 19/6/2025 Test Executed By: Joanne Test Executed Date: 19/6/2025		
Pre-condition: • A valid QR code is generated and ready to be scanned.						
Scenario	Test Case	Testing Procedure	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Verify successful visitation creation	1. Log in as staff. 2. Navigate to the visitation section. 3. Enter valid visitation information. 4. Click the 'Submit' button.	Valid visitation data	Visitation is successfully created and an alert message is displayed.	Visitation is successfully created and an alert message is displayed.	Pass
2	Verify unsuccessful visitation creation	1. Log in as staff. 2. Navigate to the visitation section. 3. Enter invalid visitation information. 4. Click the 'Submit' button.	Invalid visitation data	Visitation creation fails and validation errors messages are displayed.	Visitation creation fails and validation errors messages are displayed.	Pass

3	Verify successful visitation creation via QR (no login)	1. Scan the QR code. 2. Enter valid visitation information. 3. Click the 'Submit' button.	Valid visitation data	Visitation successfully submitted and stored to the system.	Visitation successfully submitted and stored to the system.	Pass
4	Verify form reloads for existing contact no within the same day (via QR)	1. Scan QR code with the same token on the same day.	Token	Previously entered form is reloaded with existing data.	Previously entered form is reloaded with existing data.	Pass
Post-condition: • Visitation records are created and stored in the database. • QR code opens the correct form for new or existing visitation based on the contact no and time of access.						

The test case in table 5.9 verifies the functionality of the visitation section, including manual visitation entry by staff and QR-based visitation submission by visitors. It ensures that valid and invalid inputs are handled correctly, and that the form reloads appropriately when the same contact number is used within the same day via QR code. The system is expected to store visitation records and handle the form based on the contact number and access time.

Table 6.10: Test case of semester section

Test Case ID: create_semester	Test Designed By: Joanne
Test Priority (Low/Medium/High): High	Test Designed Date: 19/6/2025
Module Name: Semester	Test Executed By: Joanne

Test Title: Verify semester functionality Description: Test the functionality of semester section				Test Executed Date: 19/6/2025		
Pre-condition: • A superadmin is created and stored in the database.						
Scenario	Test Case	Testing Procedure	Input Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Verify successful semester creation	1. Log in as a admin. 2. Navigate to the semester section. 3. Enter valid semester information. 4. Click the 'Create' button.	Valid semester data	Semester is successfully created and an alert message is displayed.	Semester is successfully created and an alert message is displayed.	Pass
2	Verify unsuccessful semester creation	1. Log in as a superadmin. 2. Navigate to the semester section. 3. Enter invalid semester information. 4. Click the 'Create' button.	Invalid semester data	Semester creation fails and validation errors messages are displayed.	Semester creation fails and validation errors messages are displayed.	Pass
Post-condition: • Semesters are created and stored in the database.						

The test case in table 5.10 verifies the functionality of the semester section, focusing on the creation of semesters section, focusing on the creation of semester records by a superadmin. It ensures the system allows valid semester data to be submitted and stored successfully, while invalid input triggers appropriate validation error messages.

5.3 Usability Testing

Usability testing was conducted to evaluate how user-friendly and accessible the proposed system is for users. This testing helps identify areas for improvement to enhance the user experience. The test was carried out through a Google Form survey, which included various questions about users' opinions of the system. The survey received responses from 30 participants.

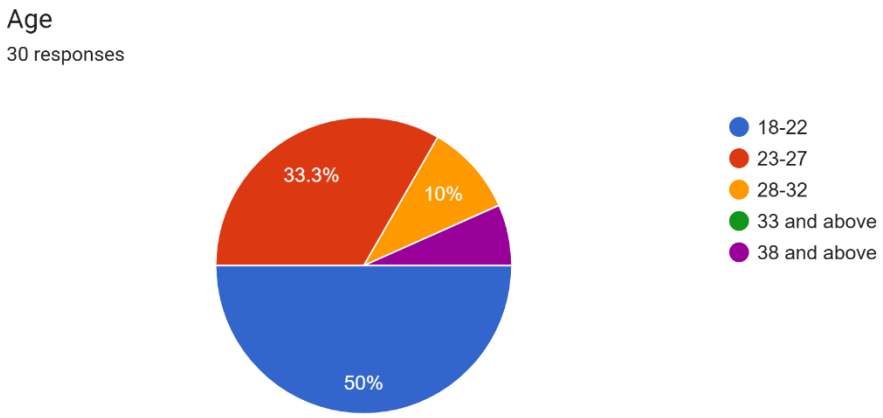


Figure 6.1: Pie chart of respondent age

Figure 5.1 shows the pie chart of the respondent age. Based on the chart in figure 126, 50% of the respondents fall within the 18–22 age group**,** followed by 33.3% in the 23–27 age group, 10% in the 28–32 age group, and the remaining 6.7% are aged 33 and above.

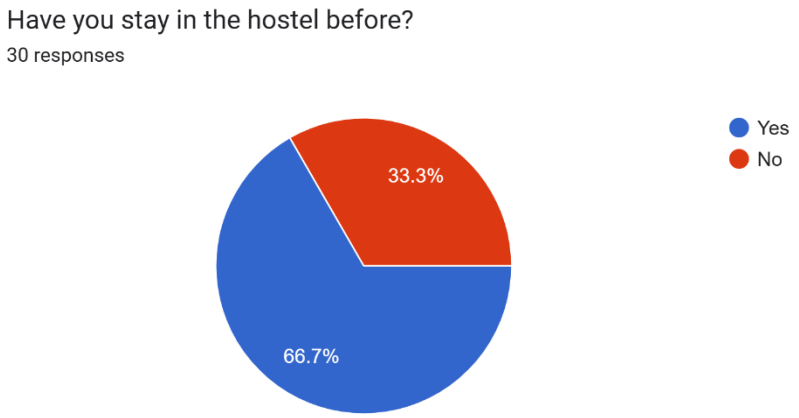


Figure 6.2: Pie chart of have you stay in the hostel before?

Figure 5.2 shows the pie chart of have you stay in the hostel before. Based on the chart in the figure 127, 66.7% of respondents have stayed in a hostel before, while 33.3% have not.

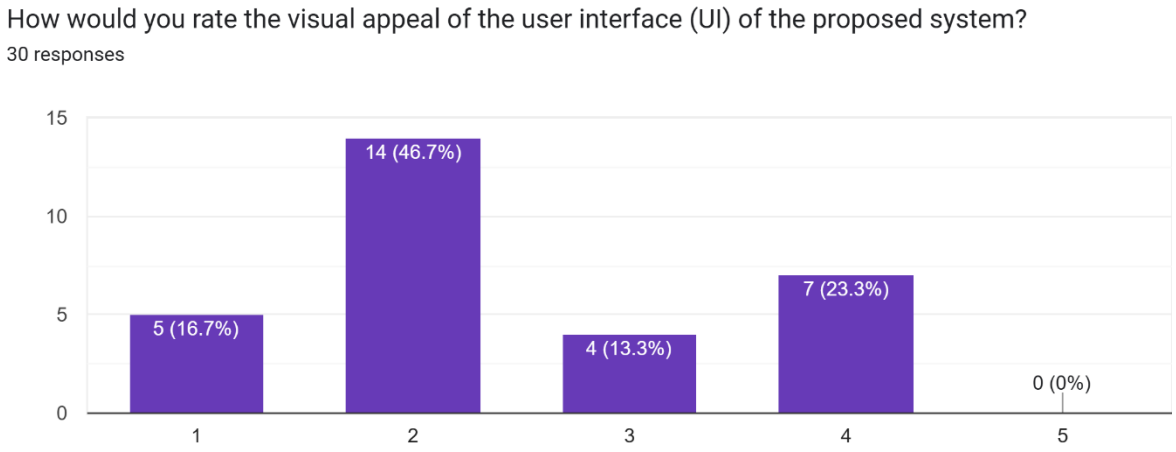


Figure 6.3: Bar chart of the result of how would you rate the visual appear of the user interface (UI) of the proposed system

Figure 5.3 shows the bar chart of the result of how would you rate the visual appear of the user interface (UI) of the proposed system. Based on the chart in the figure 128, the score

of 2 received the highest number of responses (14), indicating that many users found the UI unattractive and believe there is significant room for improvement.

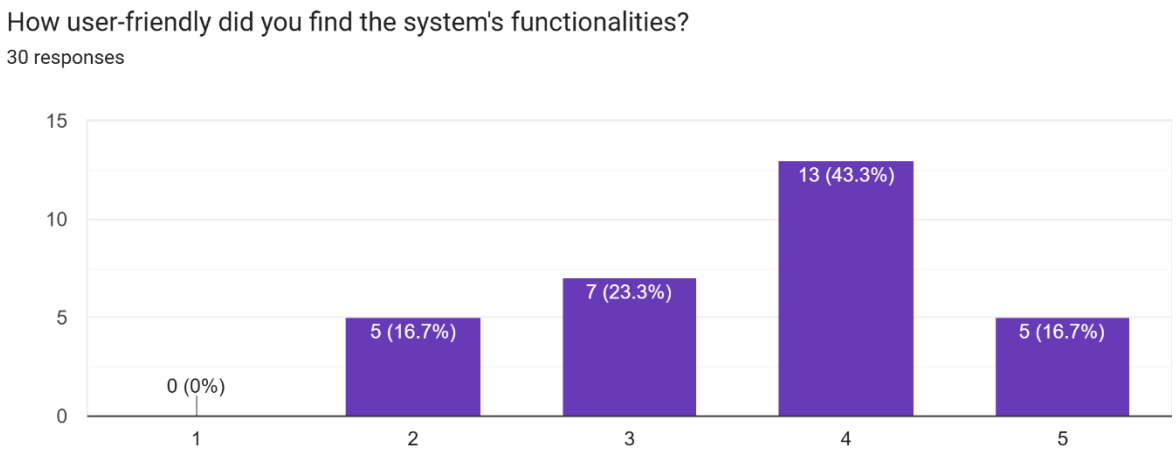


Figure 6.4: Bar chart of the result of how user-friendly did you find the system's functionalities

Figure 5.4 shows the bar chart of the result of how the user-friendly did you find the system's functionalities. Based on the chart in the figure 129, the score of 4 received the highest number of responses (13), followed by score 3 and score 5, showing that most users find the system functionalities user-friendly.

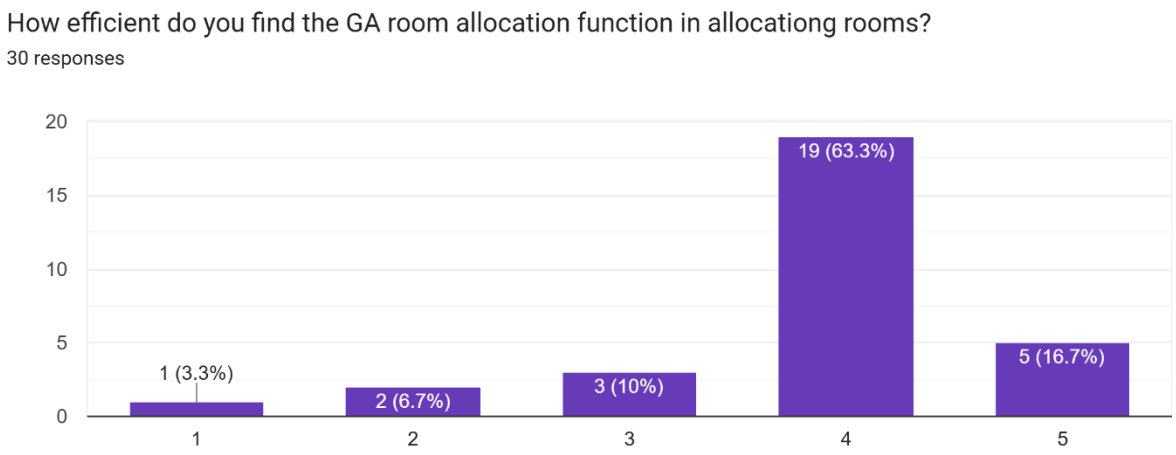


Figure 6.5: Bar chart of the result of how efficient do you find the GA room allocation function in allocating rooms

Figure 5.5 shows the bar chart of the result of how efficient do you find thr GA room allocation function in allocating rooms. Based on the chart in figure 130, the score of 4 dominated the responses with 19, suggesting that users generally find the GA-based room allocation function efficient.

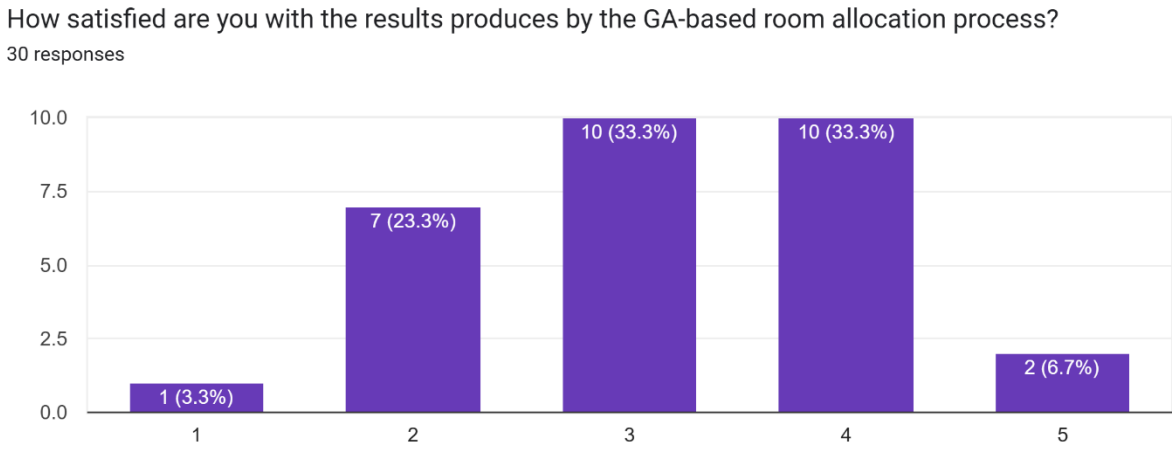


Figure 6.6: Bar chart of the result of how satisfied are you with the results produces by the GA-based room allocation process

Figure 5.6 shows the bar chart of the result of how satisfied are you with the results produces by the GA-based room allocation process. Based on the chart in figure 131, the highest number of responses (10) was for score 3 and score 4, indicating that most users are satisfied with the room allocation results, though there is still room for improvement.

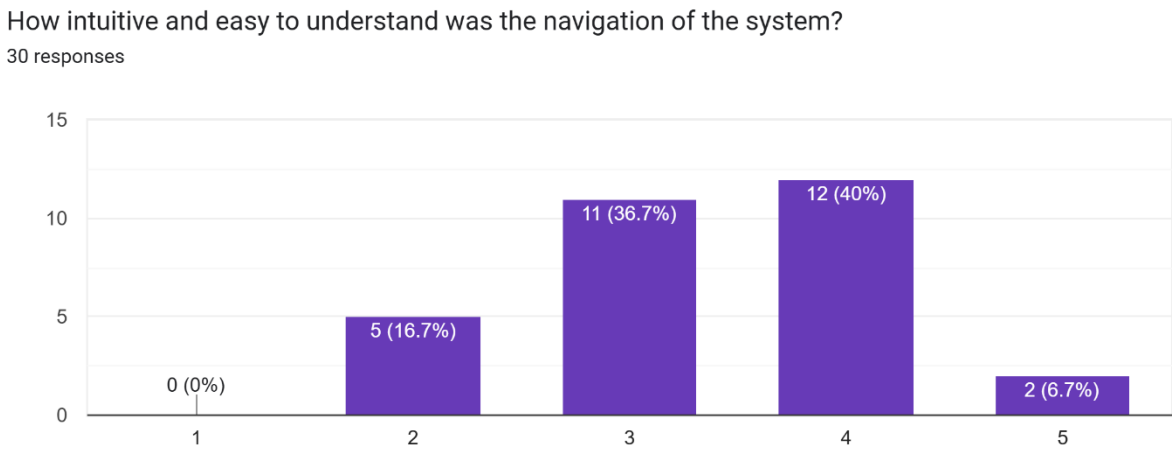


Figure 6.7: Bar chart of the result of how intuitive and easy to understand was the navigation of system

Figure 5.7 shows the bar chart of the result of how intuitive and easy to understand was the navigation system. Based on the chart in figure 132, the score of 4 received the most responses (12), followed closely by score 3 with 11 responses. This indicates that users find the system relatively easy to navigate, though some still find it moderately challenging.

Overall, the results indicate that the user interface (UI) needs the most improvement, particularly in terms of attractiveness. The current UI design is plain and lacks visual appeal, which may reduce user engagement. Enhancing the UI could make the system more visually appealing and increase user interest and adoption. The system's functionalities are generally user-friendly and well-suited for a hostel community. Features such as the facility section help residents locate and check the status of hostel facilities, while the visitation module keeps track

of visitor records. The announcement section allows staff to inform residents of important updates, ensuring they don't miss crucial notices. The application module streamlines the hostel application process, the complaint section allows residents to report issues related to the hostel environment, and the resident module helps manage and view resident information efficiently. The genetic algorithm-based room allocation function is efficient, allowing staff to assign rooms with a single click instead of handling each application manually, a process that is typically time-consuming and error prone. However, user satisfaction with the room allocation results could be further improved to increase acceptance and encourage continued hostel stay in upcoming semesters. In terms of navigation, the system is generally easy to use. The sidebar provides access to all main modules, and the header includes profile settings and notifications. However, improving guidance and layout structure could enhance the user experience, especially for first-time users. In conclusion, the proposed system effectively fulfils its intended purpose by providing a centralised digital platform with essential features to manage hostel operations. While users are generally satisfied, targeted improvements, particularly in UI design and guidance, could significantly enhance overall usability and user satisfaction.

5.4 Summary

This chapter covered the testing process of the proposed system. Through functional and usability testing, errors and areas for improvement were identified and addressed to ensure the system performs well. Testing is a crucial step in system development as it helps uncover hidden issues and ensures that the application functions reliably and efficiently.

CHAPTER 6: CONCLUSION

6.1 Introduction

This chapter presents the evaluation of the RoomPro Hostel Room Allocation and Management System. It highlights the key accomplishments achieved throughout the development process and outlines the limitations encountered. This evaluation provides insights into the effectiveness of the proposed system and identifies potential areas for future improvement.

6.2 Project Achievements

The achievements based on the project's objectives are summarised below:

Objective	Achievement
To design and develop a system with an advanced room allocation feature and hostel management functionalities to streamline hostel operations.	A comprehensive system was successfully designed and developed, integrating various hostel management features such as application handling, complaint reporting, facility, resident, visitation and announcement management and room allocation.
To implement a Genetic Algorithm-based room allocation feature in the proposed system to optimise the room allocation process.	The Genetic Algorithm-based room allocation feature was successfully implemented, enabling automated room assignments that consider user preferences and predefined constraints, reducing manual effort and potential errors.

To evaluate the functionality and usability of the proposed system.	Functional testing and usability evaluation were carried out to confirm that the system is reliable, efficient, and user-friendly.

6.3 Limitations and Constraints

One significant limitation of the proposed system lies in the room allocation process. Due to the large volume of applications, achieving a higher matching percentage requires longer processing time. To maintain reasonable performance, the generation count in the Genetic Algorithm had to be limited. As a result, the system currently achieves a match rate of approximately 60% to 70%. Extending the processing time could potentially improve the match percentage, but at the cost of usability and efficiency.

6.4 Future Work

Several improvements can be made to enhance the system further:

- Improve Match Percentage:

  Enhance the GA algorithm to be more effective or implement a scheduled job that allows it to run for a longer duration, thus increasing the overall match percentage.
- Enhance UI Design:

  Develop a more creative and attractive user interface to improve user experience.
- Integrate Chat Functionality:

  Introduce a built-in chat feature so that users can communicate directly within the system, improving coordination and support.
- Enforce Logical Process Flow:

Implement stricter control over the management sequence to ensure users follow the correct process flow. Preventing actions out of order will help maintain data integrity and ensure smoother operations.

6.5 Conclusion

In conclusion, the project successfully achieved its main objectives. The system functions effectively in managing hostel-related operations and offers a solution to streamline the room allocation process. By addressing its limitations, the system can be further enhanced to provide a more comprehensive and efficient hostel management experience.

REFERENCES

Atlassian, B. (2025, April 24). *Waterfall Methodology: A Comprehensive guide*. Atlassian. https://www.atlassian.com/agile/project-management/waterfall-methodology

Baipai, P., & Kumar, Dr. (2010). Genetic Algorithm - an Approach to Solve Global Optimization Problems. *Indian Journal of Computer Science and Engineering*, *1*(3), 199–206.

Blumer, R. (2018, June 6). My Visual Studio Code Review - Rich Blumer - Medium. *Medium*. https://medium.com/@richblumer/my-visual-studio-code-review-9870c7f7ab93

Documentation. (n.d.). Laragon - Portable, Isolated, Fast & Powerful Universal Development Environment for PHP, Node.js, Python, Java, Go, Ruby. https://laragon.org/docs/

Faulds, J. (n.d.). *Microsoft VS Code review*. TechRadar. https://www.techradar.com/reviews/microsoft-vs-code

Haldurai, L., Madhubala, T., & Rajalakshmi, R. (2016). A Study on Genetic Algorithm and its Application. *International Journal of Computer Science and Engineering*, *4*(10).

Laravel - Overview. (n.d.-b). https://www.tutorialspoint.com/laravel/laravel_overview.htm

Shukla, M. (2023, September 21). Eloquent ORM in Laravel: Simplifying Database Interactions. *Medium*. https://manoj-shu100.medium.com/eloquent-orm-in-laravel-simplifying-database-interactions-b0269942f190

Team, L. (2024, September 11). *Waterfall methodology: Advantages, disadvantages and when to use it?* Lvivity. https://lvivity.com/waterfall-model

Yadav, P. K., & Prajapati, N. L. (2012). An Overview of Genetic Algorithm and Modeling. *International Journal of Scientific and Researh Publications*, *2*(9).

What Is the Purpose and Importance of Literature Reviews in Research? - Pubrica. (n.d.). Pubrica. https://pubrica.com/insights/study-guide/what-is-the-purpose-and-importance-of-literature-reviews-in-research/

APPENDICES

Appendix 1: Survey Google Form

RoomPro: Hostel Room Allocation and Management System

Hello UNIMAS Student!

RoomPro is a hostel room allocation and management system developed specifically for the hostel community. The proposed system aims to benefit the hostel community by simplifying hostel-related tasks.

This survey is designed to gather insights into students' experiences of staying in campus hostels and their preferences regarding the functionalities to be included in the proposed system. This survey consists of three section: Section A - Demographics, Section B - User Experience and Section C - System Preference.

If you have any questions about the survey, feel free to contact me, **Joanne Kon Xin Yue**, at **79684@siswa.unimas.my.**

Thank you for taking the time to respond to this survey. I greatly appreciate your support.

* Indicates required question

Section A: Respondent Demographics

1. Year of Study *

   Mark only one oval.

   - ◯ Year 1
   - ◯ Year 2
   - ◯ Year 3
   - ◯ Year 4
   - ◯ Postgraduate

2. Where are you from? *

Mark only one oval.

- Sarawak
- Sabah
- Selangor
- Perak
- Terengganu
- Kelantan
- Kedah
- Pulau Pinang
- Perlis
- Melaka
- Johor
- Wilayah Persekutuan Kuala Lumpur
- Wilayah Persekutuan Putrajaya
- Wilayah Persekutuan Labuan
- Other (International Student)

3. Where do you currently stay at? *

Mark only one oval.

- Campus Hostel
- Home
- Outside campus rented room

Section B: User Experience

4. Do you prefer to stay in campus hostel during the entire study semester? *

Mark only one oval.

- Yes
- No
- Maybe

5. What is your most preferred option when applying for a hostel room? *

Mark only one oval.

- Room type
- Floor
- Block

6. How do you stay updated with the lates announcements? *

Mark only one oval.

- Notified by housemate
- Social media group chat
- Other: _____

7. Do you find it difficult to stay updated with latest hostel announcements using method mentioned in the previous question? *

Mark only one oval.

- Yes
- No
- Maybe

8. How do you report complaints related to hostel? *

Mark only one oval.

- Fill out the complaint form
- Face to face report to hostel staff
- Private message through chat to hostel staff

9. Do you find it inconvenient to report a complaint using the method mentioned in the previous section? *

   Mark only one oval.

   - ◯ Yes
   - ◯ No
   - ◯ Maybe

10. Do you prefer to handle hostel related tasks at single platform or multiple platforms? *

    Mark only one oval.

    - ◯ Single platform
    - ◯ Multiple platforms

Section C: System Preferences

Please answer each question by selecting one of the option. The range start for 1 (strongly disagree) to 5 (strongly agree).

11. How likely are you to prefer a centralised system to manage most hostel related tasks? *

    Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

12. How likely are you to prefer the proposed system to include a section to display the information of the facilities such as location and condition? *

    Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

13. How likely are you to prefer the proposed system to include a function for hostel application? *

    Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

14. How likely are you to prefer the proposed system to include a function for reporting complaints? *

    Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

15. How likely are you to prefer the proposed system to include a section for displaying hostel announcements notices? *

    Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

End of Survey

Thank you for your cooperation in responding to the survey.

Appendix 2: GA Room Allocation Job Code

```php
<?php

namespace App\Jobs;

use App\Models\Application;
use App\Models\Room;
use App\Models\User;
use App\Models\AllocationStatus;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
use App\Notifications\RoomAllocationCompletedNotification;
use Illuminate\Support\Facades\Log;
use Illuminate\Queue\Middleware\WithoutOverlapping;

class RunRoomAllocationGA implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public $sessionId;
    public $chunkNumber;
    public $triggeredBy;

    public function __construct($sessionId, $chunkNumber = null, User
$triggeredBy)
    {
        Log::info("Constructor called with session ID: {$sessionId}");
        $this->sessionId = $sessionId;
        $this->chunkNumber = $chunkNumber;
        $this->triggeredBy = $triggeredBy;
    }

    public function handle(): void
    {
        $start = microtime(true);
        if ($this->chunkNumber !== null) {
        Log::info("Starting Room Allocation GA for session:
{$this->sessionId}, chunk: {$this->chunkNumber}");
        } else {
            Log::warning("No chunk number provided for session:
{$this->sessionId} (chunk is null)");
        }

        try {
            $perPage = 50;
```

```
            $appQuery = Application::where('application_status', 'approved')
                ->where('session_id', $this->sessionId)
                ->whereDoesntHave('resident')
                ->orderBy('created_at');

            if ($this->chunkNumber !== null) {
                $appQuery->skip(($this->chunkNumber - 1) *
$perPage)->take($perPage);
            }

            $applications = $appQuery->get()
                ->filter(fn($app) =>
is_array(json_decode($app->preferred_room_feature, true)))
                ->values();

            if ($applications->isEmpty()) {
                Log::warning("No applications for session {$this->sessionId},
chunk {$this->chunkNumber}");
                return;
            }

            $rooms = Room::where('status', 'available')->get();

            if ($rooms->isEmpty()) {
                Log::warning("No available rooms for session
{$this->sessionId}");
                return;
            }

            Log::info("Chunk {$this->chunkNumber}: Applications:
{$applications->count()}, Rooms: {$rooms->count()}");

            $preferenceMap = $applications->mapWithKeys(fn($app) => [
                $app->id => json_decode($app->preferred_room_feature, true) ??
[],
            ]);

            $populationSize = 20;
            $maxGenerations = 5;
            $mutationRate = 0.05;
            $stagnationLimit = 2;

            $population = [];
            for ($i = 0; $i < $populationSize; $i++) {
                $chromosome = [];
                foreach ($applications as $app) {
                    $pref = $preferenceMap[$app->id] ?? null;
```

```
                    if (!is_array($pref)) continue;
                    $room = $this->getRandomRoom($rooms, $app->gender,
$pref['room_type'] ?? null);
                    $chromosome[$app->id] = $room?->id;
                }
                $population[] = $chromosome;
            }

            $lastBestFitness = null;
            $stagnantCount = 0;

            for ($g = 0; $g < $maxGenerations; $g++) {
                usort($population, fn($a, $b) =>
                    $this->calculateFitness($b, $applications, $rooms,
$preferenceMap)
                    <=> $this->calculateFitness($a, $applications, $rooms,
$preferenceMap)
                );

                $bestSolution = $population[0];
                $overallMatch = $this->calculateOverallMatch($bestSolution,
$applications, $rooms, $preferenceMap);
                $currentFitness = $this->calculateFitness($bestSolution,
$applications, $rooms, $preferenceMap);

                Log::info("□ Gen $g — Chunk {$this->chunkNumber} — Match:
{$overallMatch}%, Fitness: $currentFitness");

                if ($overallMatch >= 70 || ($lastBestFitness ===
$currentFitness && ++$stagnantCount >= $stagnationLimit)) {
                    break;
                }

                $lastBestFitness = $currentFitness;
                $newPopulation = array_slice($population, 0, $populationSize /
2);
                while (count($newPopulation) < $populationSize) {
                    $p1 = $population[random_int(0, count($population) / 2 -
1)];
                    $p2 = $population[random_int(0, count($population) / 2 -
1)];

                    $child = [];
                    foreach ($applications as $app) {
                        $child[$app->id] = rand(0, 1) ? $p1[$app->id] :
$p2[$app->id];
                    }
```

```
                    if (mt_rand(0, 100) < $mutationRate * 100) {
                        $randApp = $applications[random_int(0,
count($applications) - 1)];
                        $pref = $preferenceMap[$randApp->id] ?? [];
                        $child[$randApp->id] = $this->getRandomRoom($rooms,
$randApp->gender, $pref['room_type'] ?? null)?->id;
                    }

                    $newPopulation[] = $child;
                }

                $population = $newPopulation;
            }

            $bestSolution = $population[0];
            $assignedRooms = [];
            $totalMatch = 0;
            $matched = 0;

            foreach ($applications as $app) {
                $roomId = $bestSolution[$app->id] ?? null;
                if (!$roomId || isset($assignedRooms[$roomId])) continue;

                $room = $rooms->firstWhere('id', $roomId);
                if (!$room || $room->gender !== $app->gender) continue;

                $assignedCount = Application::where('room_id',
$roomId)->count();
                if ($assignedCount >= $room->capacity) {
                    $assignedRooms[$roomId] = true;
                    continue;
                }

                $pref = $preferenceMap[$app->id] ?? null;
                if (!is_array($pref)) continue;

                $score = 0;
                if (strcasecmp($room->type, $pref['room_type']) === 0) $score
+= 3;
                if (strcasecmp($room->block, $pref['block']) === 0) $score +=
2;
                if (strcasecmp($room->floor, $pref['floor']) === 0) $score +=
1;

                $match = round(($score / 6) * 100, 2);
                $totalMatch += $match;
                $matched++;
```

```
                $app->room_id = $roomId;
                $app->allocation_match_percentage = $match;
                $app->save();

                $assignedRooms[$roomId] = true;
            }

            $overall = $matched > 0 ? round($totalMatch / $matched, 2) : 0;
            $status = AllocationStatus::firstOrNew([
                'session_id' => $this->sessionId,
                'chunk_number' => $this->chunkNumber,
            ]);
            $status->chunk_number = $this->chunkNumber;

            $status->is_running = false;
            $status->overall_match_percentage = $overall;
            $status->save();

            Log::info("Saved overall match percentage: {$overall}% for
session: {$this->sessionId}");

            Log::info("Chunk {$this->chunkNumber} completed. Match:
{$overall}%");

            $this->triggeredBy->notify(
                new RoomAllocationCompletedNotification(
                    "Room allocation completed for session {$this->sessionId},
chunk {$this->chunkNumber}.",
                    $this->sessionId,
                    $this->chunkNumber
                )
            );

            Log::info("👎 Notification sent to user:
{$this->triggeredBy->id}");


        } catch (\Throwable $e) {
            Log::error("Job failed for session {$this->sessionId}, chunk
{$this->chunkNumber}: {$e->getMessage()}");
            throw $e;
        } finally {
            $duration = round(microtime(true) - $start, 2);
            Log::info("Chunk {$this->chunkNumber} for session
{$this->sessionId} completed in {$duration} seconds.");
        }
    }
```

```
    private function calculateFitness($chromosome, $applications, $rooms,
$preferenceMap): int
    {
        $score = 0;
        $occupancy = [];

        foreach ($chromosome as $appId => $roomId) {
            if (!$roomId) continue;

            $app = $applications->firstWhere('id', $appId);
            $room = $rooms->firstWhere('id', $roomId);
            if (!$room || $room->gender !== $app->gender) continue;

            $pref = $preferenceMap[$appId];
            if (strcasecmp($room->type, $pref['room_type']) === 0) $score +=
3;
            if (strcasecmp($room->block, $pref['block']) === 0) $score += 2;
            if (strcasecmp($room->floor, $pref['floor']) === 0) $score += 1;

            $occupancy[$roomId] = ($occupancy[$roomId] ?? 0) + 1;
            if ($occupancy[$roomId] > $room->capacity) $score -= 10;
        }

        return (int) $score;
    }

    private function calculateOverallMatch($solution, $applications, $rooms,
$preferenceMap): float
    {
        $totalMatch = 0;
        $matched = 0;

        foreach ($solution as $appId => $roomId) {
            if (!$roomId) continue;

            $app = $applications->firstWhere('id', $appId);
            $room = $rooms->firstWhere('id', $roomId);
            if (!$room || $room->gender !== $app->gender) continue;

            $score = 0;
            $pref = $preferenceMap[$appId];
            if (strcasecmp($room->type, $pref['room_type']) === 0) $score +=
3;
            if (strcasecmp($room->block, $pref['block']) === 0) $score += 2;
            if (strcasecmp($room->floor, $pref['floor']) === 0) $score += 1;

            $totalMatch += round(($score / 6) * 100, 2);
            $matched++;
```

```
        }

        return $matched > 0 ? round($totalMatch / $matched, 2) : 0;
    }

    private function getRandomRoom($rooms, $gender, $preferredType = null)
    {
        $filtered = $rooms->filter(fn($room) =>
            $room->gender === $gender &&
            (!$preferredType || strcasecmp($room->type, $preferredType) === 0)
        );

        return $filtered->isNotEmpty() ? $filtered->random() : null;
    }

    public function failed(\Throwable $exception)
    {
        AllocationStatus::where('session_id', $this->sessionId)->update([
            'is_running' => false,
            'message' => 'Room allocation job failed.',
        ]);
        Log::error("Room allocation job failed for session {$this->sessionId}:
" . $exception->getMessage());
    }

    public function middleware(): array
    {
        $key = "room-allocation-{$this->sessionId}";

        if ($this->chunkNumber !== null) {
            $key .= "-chunk-{$this->chunkNumber}";
        }

        return [
            (new
\Illuminate\Queue\Middleware\WithoutOverlapping($key))->expireAfter(60) //
expire lock after 60 seconds
        ];
    }

}
```

Appendix 3: Usability Testing Form

Usability Testing of RoomPro: A Web- based Hostel Room Allocation using Genetic Algorithm and Management System

Hello!

RoomPro is a hostel room allocation using Genetic Algorithm (GA) and management system developed specifically for the hostel community. The proposed system aims to benefit the hostel community by simplifying hostel-related tasks.

This survey is designed to gather opinions of the usability testing of the proposed system.

If you have any questions about the survey, feel free to contact me, **Joanne Kon Xin Yue**, at **79684@siswa.unimas.my**.

Thank you for taking the time to respond to this survey. I greatly appreciate your support.

* Indicates required question

Section A: Demographics

Age *

Mark only one oval.

- 18-22
- 23-27
- 28-32

◯ 33 and above

Have you stay in the hostel before? *

Mark only one oval.

◯ Yes ◯ No

Section B: Usability Testing

How would you rate the visual appeal of the user interface (UI) of the proposed system? *

Mark only one oval.

	1	2	3	4	5	
Very low	◯	◯	◯	◯	◯	Very high

How user-friendly did you find the system's functionalities? *

Mark only one oval.

	1	2	3	4	5	
Very not user-friendly	◯	◯	◯	◯	◯	Very user-friendly

How efficient do you find the GA room allocation function in allocating rooms? *

Mark only one oval.

	1	2	3	4	5	
Very inefficient	◯	◯	◯	◯	◯	Very efficient

How satisfied are you with the results produces by the GA-based room allocation process? *

Mark only one oval.

	1	2	3	4	5	
Very unsatisfied	◯	◯	◯	◯	◯	Very satisfied

How intuitive and easy to understand was the navigation of the system? *

Mark only one oval.

	1	2	3	4	5	
Very difficult	◯	◯	◯	◯	◯	Very easy

End of Survey

Thank you for your cooperation in responding to the survey.