Application Development of University Smart Assistant

By

Teng Zhi Kwang

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)

JUNE 2025

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Dr Tan Joi San who has given me this bright opportunity to take part in the development of this IT project. It is my first step to establish a career in IC design field. A million thanks to you.

Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

Copyright Statement

This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ABSTRACT

The rapid advancement of mobile technology has significantly impacted various industries, including education. Traditional university resource management systems often suffer from inefficiencies, such as scheduling conflicts, underutilized spaces, and difficulties in navigating large campuses. To address these challenges, this project presents the development of a University Smart Assistant application, which integrates smart scheduling conflict for booking and Augmented Reality (AR) to enhance resource booking and navigation. The Smart Scheduling Assistant module analyses current availability to recommend optimal booking times, reducing scheduling conflicts and improving resource utilization. The AR Navigation module provides clear, arrow directions, and also voice guidance, helping users locate their reserved facilities efficiently. This application aims to streamline resource management, improve the user experience, and set a new standard for booking and navigation in university, allow user to book. The methodology used to develop this mobile application is RAD (Rapid Application Development).

Area of Study: Augmented Reality (AR), Mobile application development

Keywords: AR Indoor Navigation, Smart Booking Conflict Handling, Mobile Application, Unity Application, Voice Guidance

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AR	Augmented Reality
UTAR	Universiti Tunku Abdul Rahman
CRM	Customer Relationship Management
UI	User Interface
IDE	Integrated Development Environment
API	Application Programming Interface
OS	Operating System
IC	Integrated Circuit
Gantt	Generalized Activity Network Task Tracking
ARCore	Augmented Reality Core (Google's AR platform for Android)

CHAPTER 1

1.1 Problem Statement

Managing university resources effectively is a big challenge due to several important issues, such as scheduling conflicts, inefficient utilization of all the resources, and user navigation difficulties. These challenges can significantly impact the university from their normal operation.

1. Challenges in Achieving Optimal Room Booking Times.

Students face significant challenges in booking university rooms at optimal times due to the limited availability of good system for them to book resources, leading to double reservations and scheduling conflicts. This happen especially when university apply paper form booking or verbally booking from their administrator or lecturer for the resources. This situation affects academic programs, causing students to encounter frustration and impacting their overall satisfaction. Large colleges have complicated scheduling, which is made worse by old booking systems. This inefficiency not only wastes valuable resources like time and energy of admin but also imposes a large financial problem on the university to resolve these scheduling issues, giving funds from other critical areas and lowering the overall quality of services and education in the university.

2. Difficulties in locating the booked facilities once a reservation is made.

Students often struggle to locate reserved facilities, particularly in large, complex campuses. This problem is particularly obvious for new students, who might not be familiar with the layout of the campus, which causes delays and anxiety. Students will find it hard to locate the rooms they booked. These problems with booking and navigation make for a poor learning environment that distracts students from academic tasks and negatively affecting their performance. To address these challenges, a more advanced system is needed to optimize both booking and navigation, ultimately enhancing the overall productivity and efficiency of the university.

1.2 Background and Motivation

As stated clearly in [1], recent fast developments in mobile technology have had a significant impact on all industries, with being one of the ones most benefit. These advanced technologies have affected educational systems to improve overall operational efficiency, streamline administrative procedures, and provide better learning experiences. As stated in [2] Universities are adopting more and more intelligent solutions to adapt to the changing needs of their faculty, staffs, and students.

Smart scheduling technologies are very important for improving the efficiency of resource management. They enable automatic detection of any conflict, recommend suitable time slots, and update the real-time booking status, reducing admin or librarian workload and enhancing the use of university's facilities.

Augmented Reality (AR) further help smart scheduling by providing arrow and voice-based navigation assistance for students. AR enhances the real world with virtual elements, offering clear, voice guided, interactive guidance within large and unfamiliar campuses especially for new student. This technology greatly reduces confusion and supports a seamless experience for new and existing students alike.

Motivation exist for this project is as universities like UTAR (University Tunku Abdul Rahman) grow faster and the number of student increases, resources management would become more and more difficult. Booking and managing resources method such as tutorial rooms, laboratories, and other facilities and books, if is using traditional method, often lead to inefficiencies and user dissatisfaction. Traditional booking systems often include manually booking processes or some simple digital tools to manage the booking. For example, most commonly used paper-based systems or basic digital systems using spreadsheets for booking. It is difficult for staff who uses this kind of system because it caused scheduling conflict as well as inefficient utilization of the resources. Scheduling conflicts, underutilized spaces, and the time-consuming are all issues that manual booking processes have and the need for more advanced system [4]. In addition, there is typically a more delay on time between making an appointment request manually or through form and getting it confirmed or approved by admin [11]. Based on research that had done in [13], for a booking system, most of the user prefer online booking system rather than the traditional booking system since they think that

online booking is more efficient and easier for them. A large university needs a smart booking system to manage all the resources it have and must be user friendly [3]. Efficient resource management is very important to avoid scheduling conflicts, optimize utilization, and ensure a smooth academic workflow, which will directly affect the quality of education and enhance student experience in university [6].

In summary, the rapid advancements in mobile technology and smart scheduling are changing and developing the education sector, giving new ways to enhance efficiency and user experience. The development of a University Smart Assistant application exemplifies how these technologies can be harnessed to solve challenges in resource management. Through smart scheduling and AR navigation, the application will confirm to create a more efficient, user-friendly, and technologically advanced academic environment for students. An important advancement in the incorporation of smart scheduling and AR in educational service can be seen in the University Smart Assistant application. The application provides smart campus solutions and improves the overall student experience by solving the problems of resource management and navigation. The use of these technologies ease students and also admin or known as school staff.

.

1.3 Project Objectives

1. Development of a Smart Scheduling System for Optimized Room Booking and Utilization.

To develop a Smart Scheduling System that optimizes room bookings to analyse historical booking data, and current availability, reducing double reservations and scheduling conflicts. This system aims to improve the efficiency of room and book resources utilization, minimize scheduling conflicts, and enhance the overall academic experience by ensuring students have access to optimal timeslots and book for the resources they need. They will be blocked form booking the conflicting resources determined smartly by system.

2. Integration of an Augmented Reality (AR) Navigation Module for Campus Facility Localization.

To create an Augmented Reality (AR) Navigation Module integrated into the booking system, designed to provide real-time, arrow directions for students to locate their reserved facilities within the campus. This module will help new and existing students navigate complex campus layouts with different floor, reduce delays and anxiety, and improve their overall campus experience, ensuring that they arrive at their destinations on time without any issue.

1.4 Project Scope and Direction

The scope of the project is to implement Smart Scheduling and Augmented Reality (AR) system designed to foster resource booking and navigation within a university. The scope of the project also includes all the essential modules necessary for a resource booking mobile application.

I. Smart Scheduling Assistant Module

The Smart Scheduling Assistant leverage smart conflict detection to prevent scheduling issue for booking resources. This module analyses upcoming booking details to suggest optimal time slots. It identifies potential scheduling conflicts and block user from selecting the conflict time for booking whether rooms or books, ensuring efficient use of resources. No matter the booking request is pending or approved, it will be determined and automatically prompt use message and block user to choose another date and time. It will show the booking time it conflicted to user.

II. AR Navigation Module

This module leverages AR technology to provide users with real-time navigation to their booked resources, which are UTAR Block N, ground and first floor and also library . Users can view an interface where user can choose the start location first in a particular block in university. Then user can choose a target location room where user need to navigate to, then an arrow direction will show to guide user to the correct target location, and also voice guidance when turn right/left, also go straight and arriving destination, helping them to easily locate specific rooms or facilities within the university. There is also virtual information

board to show where the rooms are located at. The AR Navigation Module enhances the user experience by overlaying virtual object and information about the environment, making it easier to find the desired location.

III. User Authentication and Profile Management Module

This module allows users to log in with their university credentials and access the system's features based on their role. There is also sign in module for new user to sign in and forget password module which the system will send a link to user's Gmail to change new password. Users can view and update their profiles, which include personal information and booking history. The module ensures that different user roles, such as students, and administrators and librarian, have access to different module and functions.

IV. Resource Management Module

The Resource Management Module organizes and categorizes all available resources, such as rooms and books. It allows administrators and librarians to define resource availability, change the information of rooms like description, availability, name and books' details such as name, title, category like fiction or non-fiction. and save and update in database. Admin can add and update custom details like room capacity. Admin also can add new rooms into the system. Librarian's function are same as admin but it focusses only on books. Admin and librarian can manage bookings made by user, they can approve or reject the booking made by user, and the status of booking will be updated. Users can view detailed information about each resource before making a booking, ensuring they select the most suitable option.

V. Calendar and Booking Module

This module provides users with a comprehensive calendar view where they can choose a date to books for rooms or books. The calendar is shown for user to choose date with the Smart Scheduling Assistant to avoid conflicts, when conflict, the system will automatically disable the booking button for user, prompt message that it conflicts with which booking at what time. Users can also view their past and upcoming bookings, receive notifications for upcoming appointments, and manage their time effectively.

VI. Notification and Reminder Module

To ensure users are informed, this module sends notifications and reminders about upcoming bookings, potential conflicts, overdue bookings and important updates. Notifications can be sent via email for password change and in-app notification panel. This module helps users stay organized and avoid missed appointments or delayed returned the books, user can always view back to the notification panel.

VII. View Booking Details Module

User can view their booked resources, they could view all the details inside for the bookings they made. The booked details and status whether it is approved by admin or librarian for all the rooms and books. For the bookings, user could cancel the pending or approved bookings. The status will be updated and show to user. This module helps user to keep track of his or her bookings details and status.

VIII. Search and Filter Module

The Search and Filter Module enhances the user experience by enabling users to efficiently search for and filter resources according to various criteria, such as availability for rooms and category of books for books. This functionality simplifies the process of finding specific resources, allowing users to quickly locate what they need without having to browse through all lists manually. By providing easy search options, the module helps users save time and

find suitable resources more effectively, ultimately leading to a more streamlined and user-friendly booking experience.

VIV. Check in out module

This module allow user to check in to the rooms and books they reserved for. This module allows users to confirm the use of the resources. When the due time is past, user need to check out. If not, a notification will sent to user to inform him This module allow user to check in for their approved booking and also check out when they are not using or the end time is near.

VV View Overdue Booking Module

This module allows admin or librarian to view all the users booking where they are check in but not check out, meaning that they do not return book or do not leave the room after the end time is up. Through this module, admin/librarian be directed to phone call to inform the user for overdue booking. This module ensure that admin can manage all the students whether they had returned the book reserved or leave the rooms they booked when times out.

VVI View History Module

This module can be enter from check in out module and it allow users to view their completed bookings history, which are all the bookings that is checked out. User can view for the details of the bookings they made for the past.

VVII Statistic Module

This module allows admin/librarian to view a pie chart data analysis, showing from all the completed bookings. It shows the number of bookings made by each rooms or books clearly using different colour in the pie chart. Admin/librarian can know for more details percentage by clicking the label and the details percentage for number of bookings made will shown for each resources. Admin/librarian can filter the data for all, past month or past week. This module allow admin to see which resources commonly booked by user.

VVIII Report and View Issue Module

This module allows user to report for a issue happens in a room and it will be send to administrators. User can choose a room which had issue, choose from the given reason, if not exist user can filled in by themselves. After user submit the issue, Admin can view the issue. Admin here can view all the issue happening, if the issue is resolved admin can confirm the issue resolved by this module.

1.5 Proposed Approach / Study

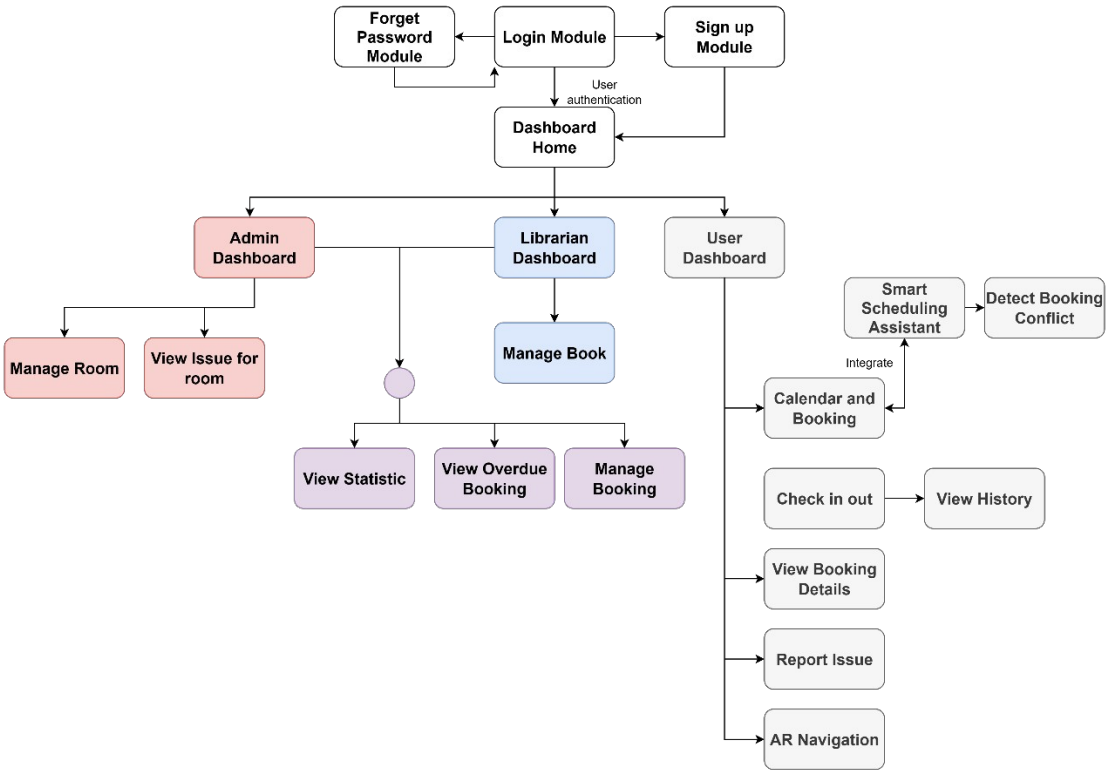


Figure 1.5.1 Flowchart for University Smart Assistant, Infinite Book Application

Above had shown the flowchart for proposed project. It starts from user login module where all user can login with their credentials. If no account, user can sign up for a new account or if forget password user can request email notification for resetting password. After the user is authenticated, they will directed to different dashboards according to their user role, whether is user, admin or librarian. Admin can manage rooms and view all the issues for the rooms. Librarian can manage books. Both admin and librarian can view statistics for overdue

bookings and manage all the bookings whether approve or reject. Moreover, user can made bookings through the booking module and there would be scheduling logic to check for the booking conflict and block user when it meets a conflict with others. User can view their bookings detail whether being approve or rejected by admin/librarian. Furthermore, user can do check in out for the approved bookings and view the history for all the completed bookings. User can report for a issue for rooms and also proceed to AR navigation which will open unity app for rooms or librarian indoor navigation.

1.6 Highlight what had been achieved

An important advancement in the incorporation of smart scheduling and AR in educational services can be seen in the University Smart Assistant application. The application provides smart campus solutions and improves the overall student experience by solving the problems of resource management and navigation. The use of these technologies eases students and also admin or known as school staff.

This project suggests creating a mobile application with AR for room navigation and a smart scheduling assistant to address all of booking and navigation issues. By examining booking details, the Smart Scheduling Assistant makes prevent most of the conflict when making reservations for rooms and books. The system's intelligence will alert users to possible conflicts so they can avoid overlapping reservations and make well-informed decisions. guaranteeing a more equitable and effective use of university resources. By optimizing the entire booking process, this method not only reduces conflicts but also makes it more efficient and user-friendly.

Additionally, the AR Room Navigation feature is designed to guide students to their booked locations with ease. By using augmented reality, this feature provides real-time, clear arrow directions, helping users navigate through the campus efficiently. By reducing the time and effort required to find booked facilities, this navigation helps to enhance the overall user experience and ensures that students reach their destinations without delay on time. The AR feature can also include images with text description when showing the start location, ease for user to find their start location and start for the navigation

1.7 Report Organization

This FYP2 report is mainly separated into 6 chapters, and each chapter description is as below:

Chapter 1 is mainly about the problem statement and objectives of this project, it shows all the work achieve including project scope and flowchart which justifying the overall application looks like and they are created to achieve this project's objective and solve the problem statement.

Chapter 2 is consists of the literature review for a project similar to the University Smart Assistant application. There are also strengths and weaknesses for each of the existing bookings or scheduling application and all of the strengths and weaknesses are also listed and compared by using a table.

Chapter 3 shows all of the use-case diagrams for this system. There is also use-case description for each use case. Furthermore, it covered some explanation on the code developed for smart scheduling and also AR navigation.

Chapter 4 covers all the system methodology of this project and also hardware and software used for this project. Then, timeline which is Gantt Chart of this project, covers from FYP1 to FYP2 clearly displayed at the end of this chapter.

Chapter 5 shows the implementation of this project. Including the user interfaces for the Booking app (Infinite Book) and also AR navigation. Testing is also done at the end of this chapter on some of the functions.

Chapter 6 justify the overall summarization of this project, it states all the project review and for the novelties and contribution. Finally, justifying all the future works can be done for this project.

1.8 Summary

This chapter mainly explains why there exist this project for University Smart Assistant, starting from the problem statement where students and staffs faced when dealing with the booking issues. There would be lots of conflict in the traditional way of booking. There is also an issue that it is hard for students to find the way to the reserved rooms. There exists this project with several module such as smart scheduling and AR navigation etc to cope with these problems. This project would greatly help students and staffs to deals with massive bookings details easily and also help students to manage their time for efficiently without struggling where is the reserved resources. This project will contribute a significant effect on replacing current paper-based booking system which will cause a lot of human error

CHAPTER 2

Literature Review

2.1 Overview

This chapter basically reviewed similar projects related to resources booking. The projects are reviewed on their purpose of having the app and what functionality they must compete with other and related to this project. Strengths and limitations will also be reviewed to compare apps.

2.2 Similar Projects related to Resource Booking

2.2.1 Cozycal

CozyCal is a powerful tool that helps to manage bookings for teams and individual, it is designed to make the scheduling process easier for user to use [20]. CozyCal provides a lot of important features to effectively manage and arrange the team schedules [21]. Admin can share the invitation link or can send emails to team members inviting them to join a shared booking page. Administrators can assign, modify, and check the team's availability and some of the scheduling setting. With modifying availability for various event, best booking can be suggested and prevent conflicts. Admin can also change confirmation messages, or they can create email notifications, they could also gather important data for client which could be improve customer communication. This involves cancellation policies and reminders to increase client satisfaction levels. To synchronize availability and avoid double bookings, administrators can also choose to connect with Google or Outlook calendars.

To manage their own availability and event types within the team booking page, invitees can utilize Cozy Cal’s user-friendly interface. Those who had invited have the choice to create custom event types that are only available to them or they can add their own schedules to shared event types [21]. If invitees would rather keep their services apart from the main team schedule, they can also personalize their own booking pages. They can modify the fields on their own intake forms to include client information, and they can change the email notifications and confirmations linked to their reservations. By leveraging these features,

invitees can maintain control over their scheduling preferences while integrating seamlessly into the broader team management framework provided by CozyCal.

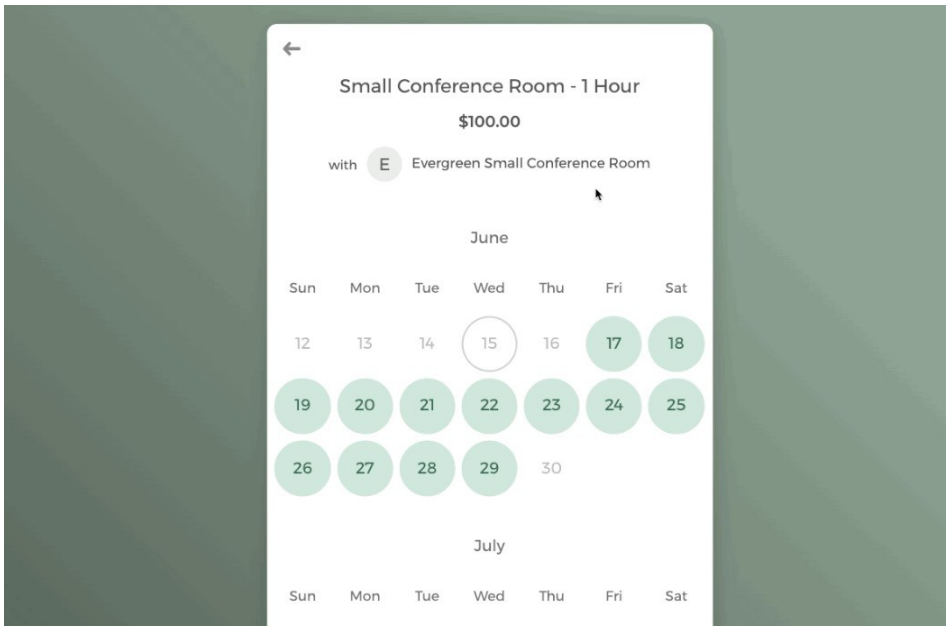


Figure 2.1.1.1 Add availability with flexibility

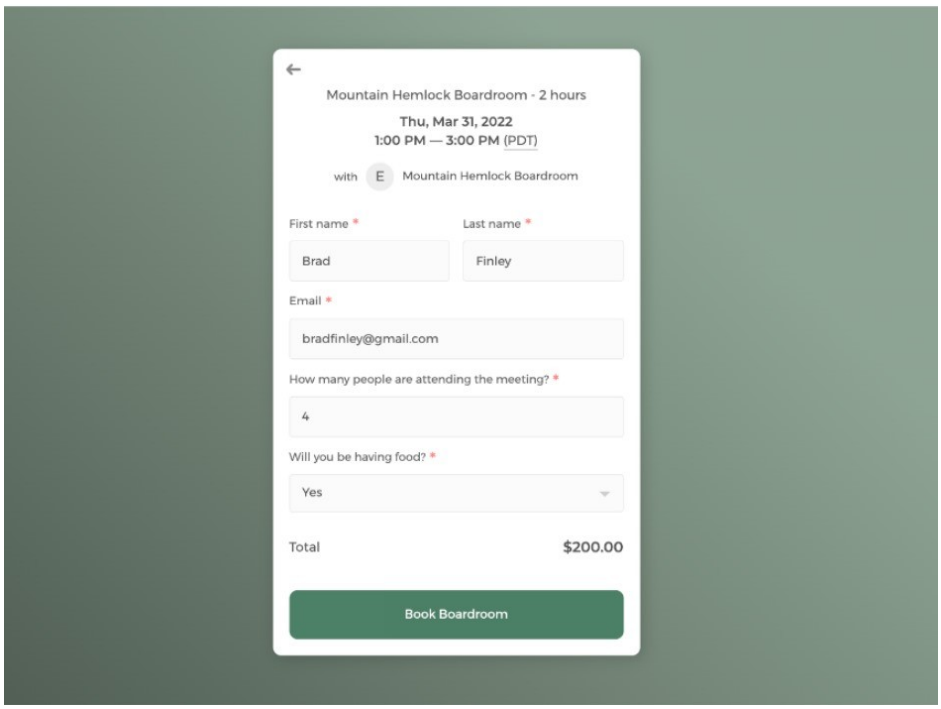


Figure 2.1.1.2 Customized intake form

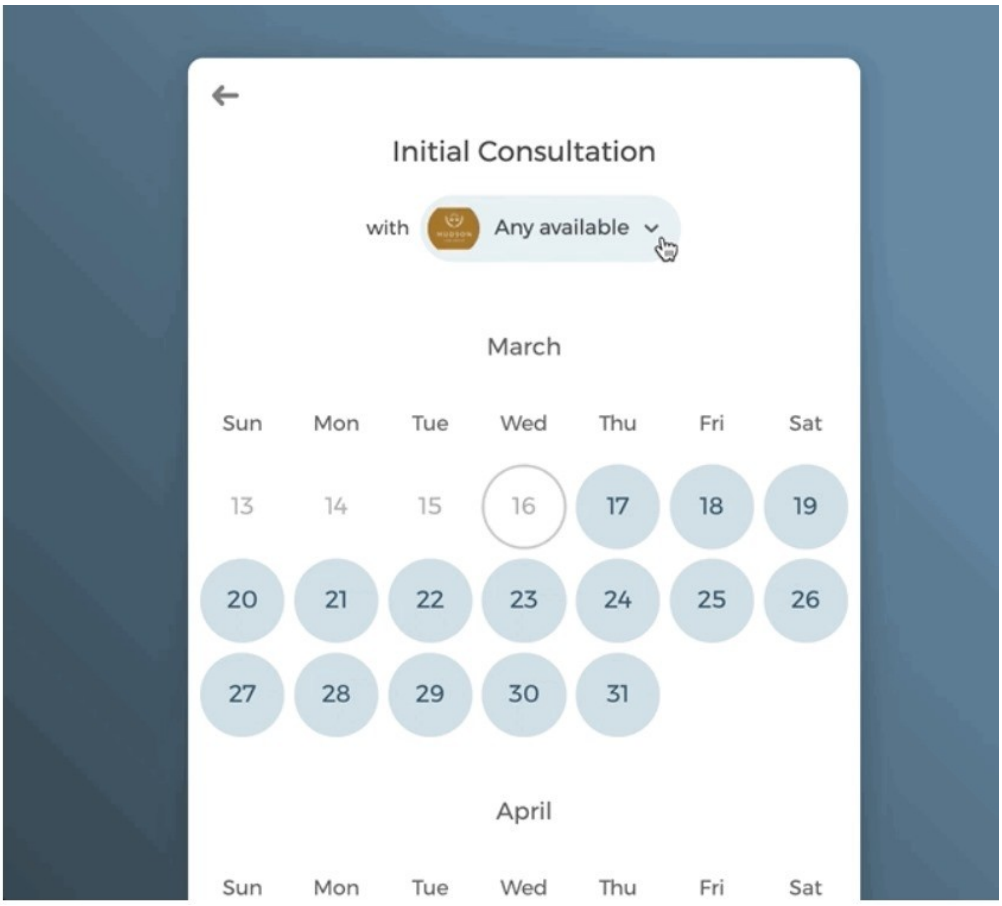


Figure 2.1.1.3 Choose any available to see which date for host is available

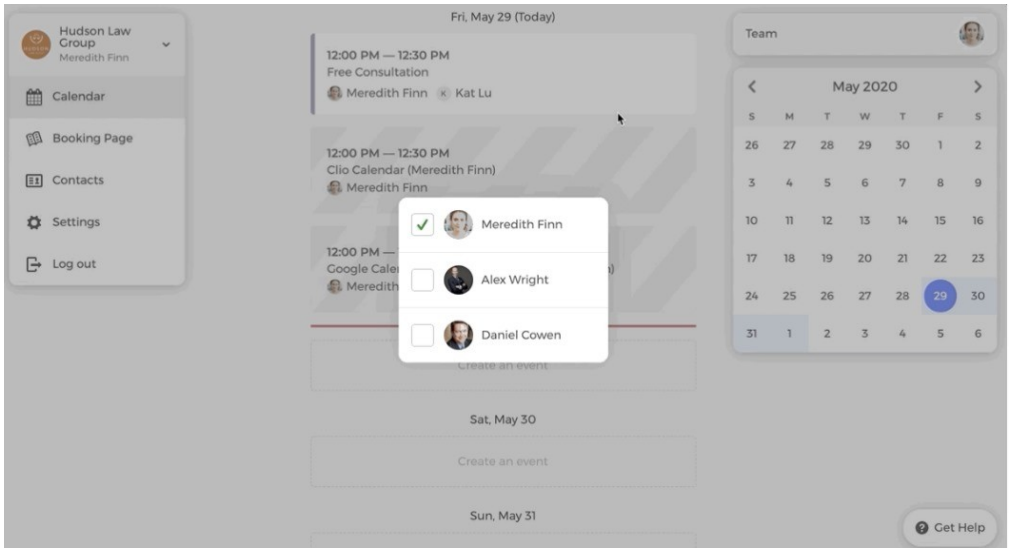


Figure 2.1.1.4 View other Team members' booked events on calendar

Strengths

- Easy to use with a straightforward booking process.
- Supports flexible scheduling with round-robin booking and individual event types.
- Integrates with Stripe for online payments, discounts, and receipts.
- Allows customization of intake forms for detailed client information.
- Sends automatic, customizable email notifications and supports white labeling.
- Provides tools for managing team schedules and individual booking pages.

Weaknesses

- Limited reporting features may not suit larger organizations.
- Primarily designed for small to medium-sized studios.
- Only supports desktop use, lacking dedicated mobile apps.

2.2.2 ZOHO Calendar

Zoho Calendar was created to make resource management in businesses more efficient. Its resource booking function makes it possible to allocate and use resources—like branches, buildings, rooms, and related features—efficiently. The capabilities and execution of Zoho Calendar's Resource Booking feature are the main topics of this review.

As stated in [24], administrators can effectively manage organizational resources with the aid of the Resource Booking feature. The first step is to set up the branches and buildings, including building names and floor plans. Because of the logical resource organization provided by this hierarchical structure, booking is simple and easy. Subsequently, rooms with specifics like location, kind (conference, meeting, etc.), capacity for people, and features (air conditioning, Wi-Fi, projector) can be added to every floor. This classification helps users choose the right rooms. As stated in [25], administrator also can customize their own booking pages with features provided in the app.

Through an interface that shows availability in a configurable grid view by day or hour, users can make reservations for rooms. Users can easily find rooms that fit their schedules by filtering the grid based on working hours and weekends. Establishing a default location and preferred grid view are examples of customization options. Not only that, as stated in [26] user can sync calendar to make sure that the resources that they booked does not crash with their others event.

As in [24] a QR code may be assigned to each room so that quick device scan bookings are possible. Using CSV or TSV files, administrators can import and export branch and room data, enabling bulk updates and backups. Other features include safely exporting data with password protection and printing lists of rooms and branches.

Creating and editing events on a shared calendar, inviting people irrespective of their email provider, and getting email or pop-up reminders are just a few of the collaboration features available on the app. Its usefulness is increased when calendars can be embedded and shared with access to Zoho apps like Planner, CRM, Business, and Meeting.

The Resource Booking function of Zoho Calendar provides an all-inclusive approach to managing organizational resources. Its organized design, adaptable reservation choices, and sophisticated administration tools maximize resource use, boost effectiveness, and raise output in businesses.

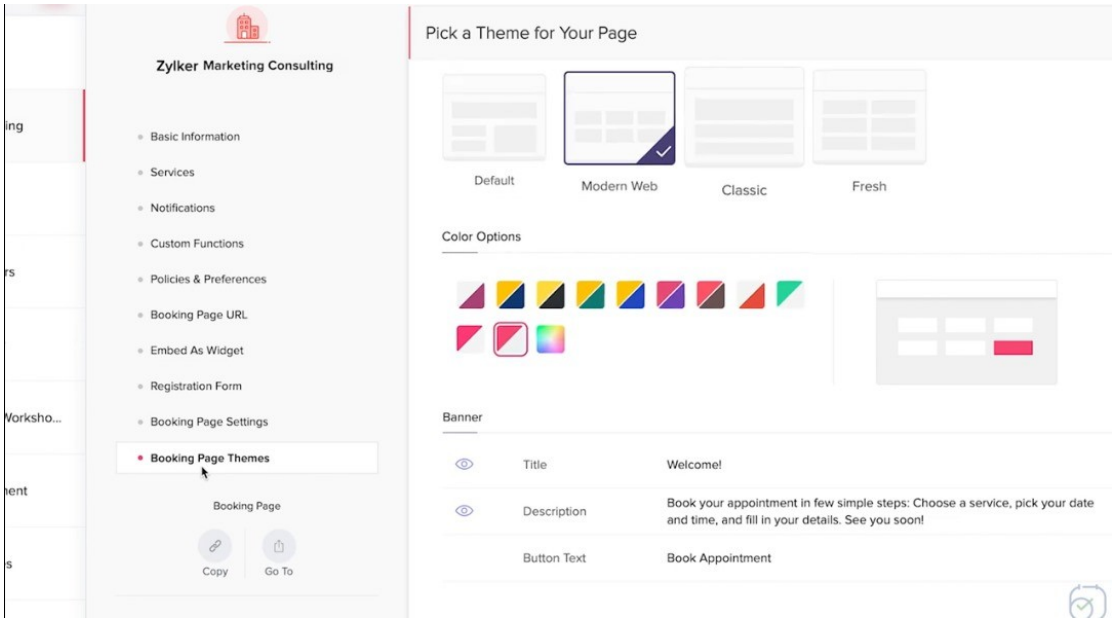


Figure 2.1.2.1 Customize own booking pages

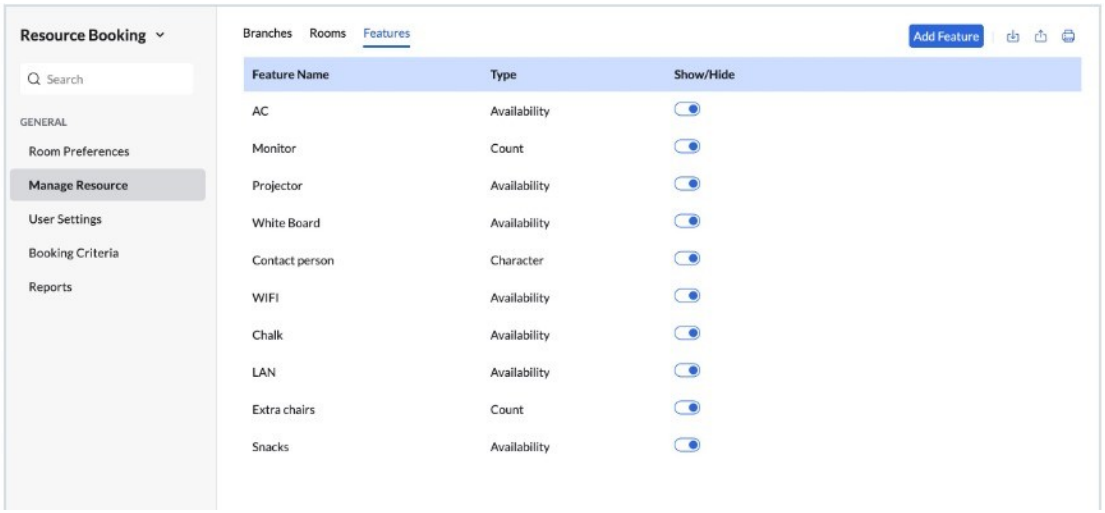


Figure 2.1.2.2 Rooms with specific features can be set

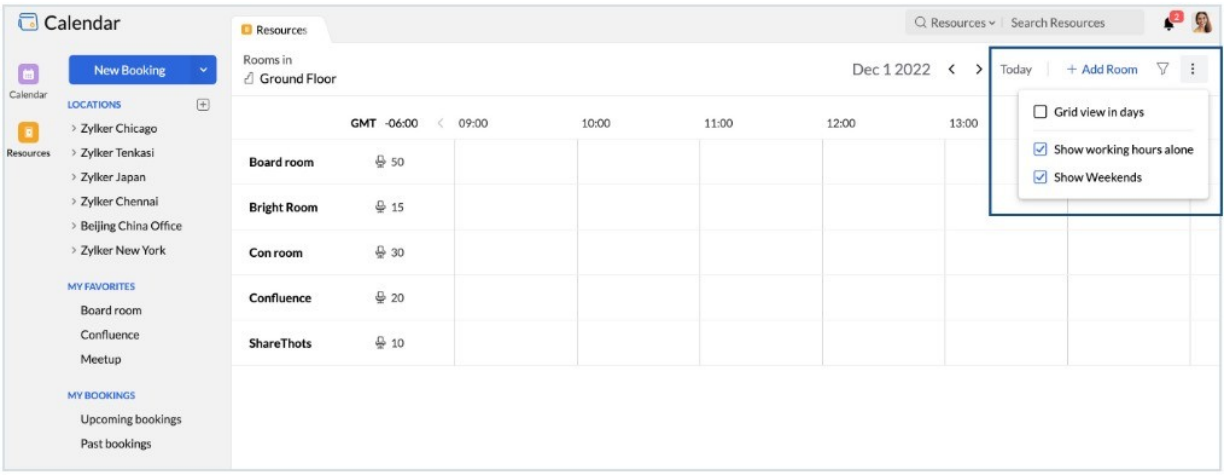


Figure 2.1.2.3 Interface shows availability in a configurable grid view by day or hour

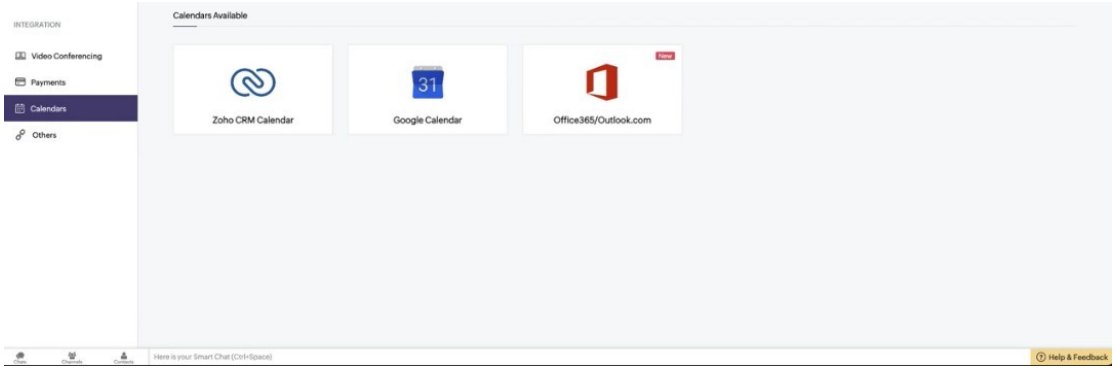


Figure 2.1.2.4 Syncing with calendars

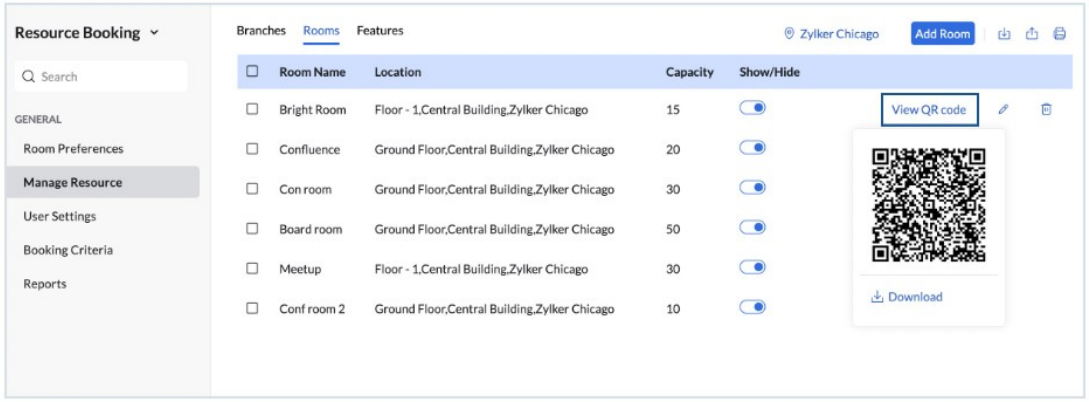


Figure 2.1.2.5 QR code to be assigned to each room

Strengths

- Simple to use and understand
- Setting up meetings and sending invitations could be done with ease.
- Google Calendar and numerous CRMs are easily integrated.
- Functions on PCs and mobile devices (Android and iOS) with the fully integrated calendar view feature
- Free and user friendly
- Supports multiple OS system
- Convenient scheduling at a click

Weakness

- There are moments when Zoho Calendar syncs very slowly with other programs or devices.
- Integration capabilities should be expanded to be fully accessible through an open API for seamless connectivity with third-party services.
- For effectively serve a global user base, it should support additional languages.
- Small buttons in the interface

2.2.3 Ganttic

Ganttic is a online resource scheduling tool that consist of many features that uses Gantt charts to deliver precise and visual planning. Users of Ganttic can quickly identify conflicts and gaps in the resource schedule, as well as who is already booked and available for new work [28]. Whether user creating plans for upcoming projects or reviewing the current schedule, the software enables real-time updates. One of Ganttic's most features is its easy-to-use drag-and-drop interface, which makes task allocation quick. To effectively balance workloads, users can click, drag, and release tasks for new bookings, as well as reallocate them as needed. By guaranteeing optimal resource utilization, this feature lowers frustration. As stated by author in [29] The timeline can be readily adjusted to suit executives' longer-term perspectives as well as short-term scheduling requirements.

There are many resource types are supported by Ganttic, including persons, equipment, vehicles, buildings, and tools. Users can group, filter, and identify the resource for each task by using the custom data fields that facilitate resource organization. This is especially helpful for teams with varying schedules, dispersed locations, or certification requirements. Another essential feature of Ganttic is resource tracking, which provides summary to track the usage of the resources. Tracking scheduled hours and utilization rates allows users to determine who is capable of handling more work. Through early conflict detection, users can prevent future scheduling problems.

In conclusion, Ganttic is a strong and adaptable scheduling tool that guarantees the effective use of resources and improves scheduling procedures. It is a essential tool for companies trying to maximize their resource management because it is perfect for handling a wide range of resources, including workers, meeting spaces, equipment, and cars.

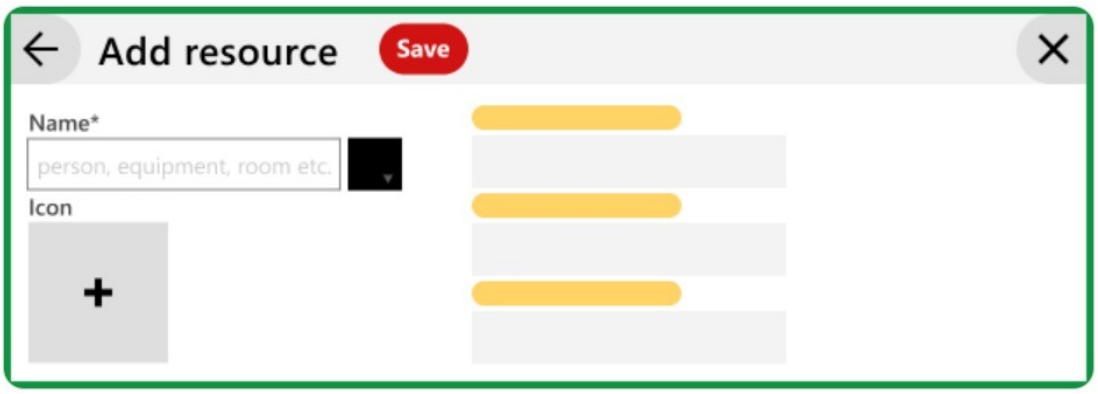


Figure 2.1.3.1 Adding resources for administrator

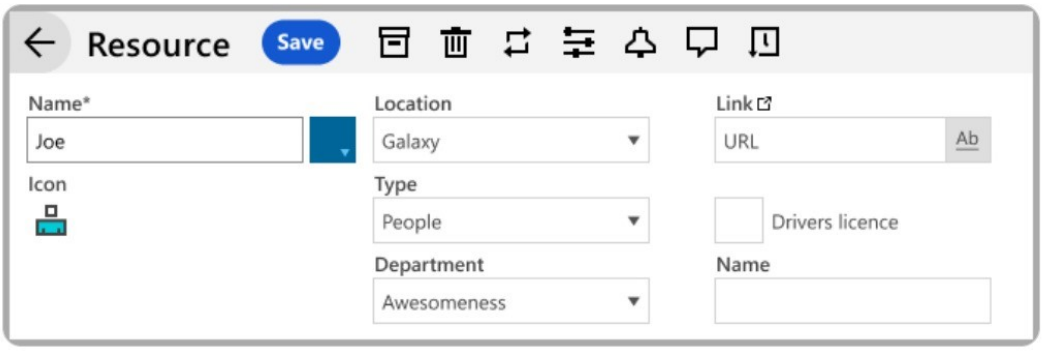


Figure 2.1.3.2 Resource edit dialog for administrator

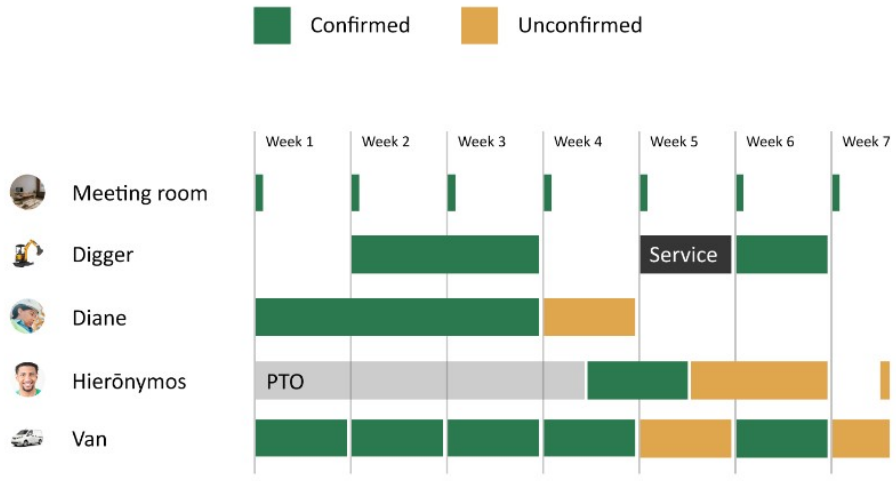


Figure 2.1.3.3 Resources Availability for user

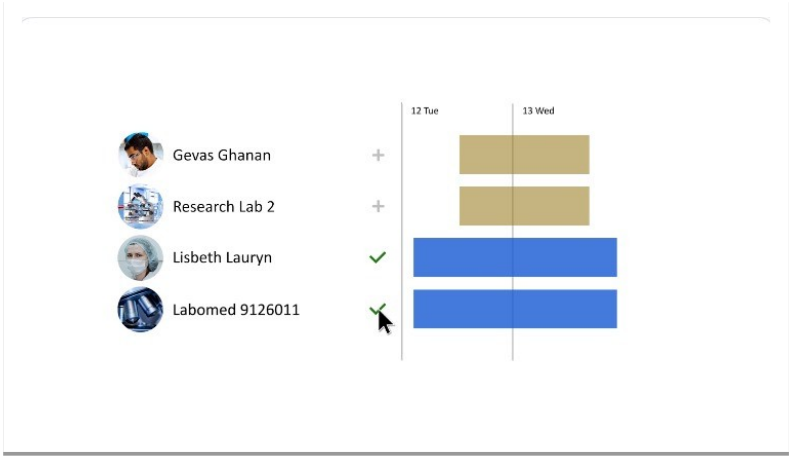


Figure 2.1.3.4 Easy-to-use drag-and-drop interface for user

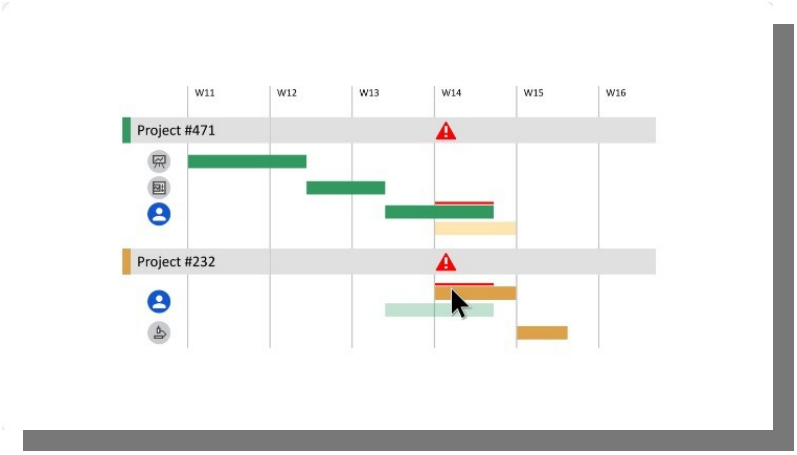


Figure 2.1.3.5 Instance spot fixing conflict with time for user

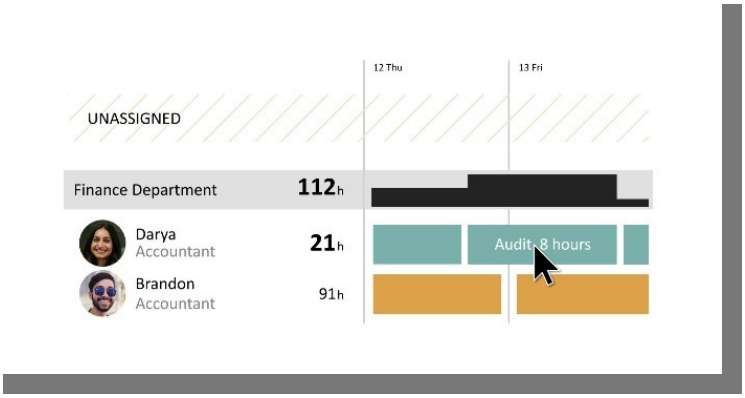


Figure 2.1.3.6 Visual summary of how resources are used over time.

Strength

- Free of up to ten resources
- An excellent substitute for Microsoft Project and spreadsheets that focuses on resources and provides a Gantt chart-style visual summary of everything
- To create the necessary charts and reports, construct various views, groupings, and custom data fields, then filter everything out.
- Utilization planning makes it simple to see who is overbooked by using numbers and colors.
- Facilitates collaborative planning and varying access levels with its multi-user interface support.
- Conditional coloring of taskbars and dynamic resource grouping/filtering enhance visual management.
- Dynamic resource grouping and filtering, along with conditional taskbar coloring, improve visual management.

Weakness

- Lacks an API
- It is intended for resource planning and is not suitable for conventional project planning.
- Subtasks cannot be created.
- An undo button is absent.

2.2.4 Overview for table

This table below provides a comaprison overview between three scheduling and resource management app which are CozyCal, Zoho Calendar, and Ganttic. It states that their strengths in user interface, team and resource management, customization, integrations, and some of the scheduling features.

Table 2.1 Comparison between existing applications

Feature	**CozyCal**	**Zoho Calendar**	**Ganttic**
User Interface	User-friendly interface with drag-and-drop functionality	Simple interface, but small buttons might be an issue	Easy-to-use drag-and-drop interface for task allocation
Team Management	Allows team scheduling, modifying team availability, and custom email notifications	Manages organizational resources, shows room availability in a grid view	Supports scheduling for various resource types, provides resource tracking
Customization Options	Customizable intake forms, cancellation policies, and reminders	Customizable booking pages, assign QR codes to rooms	Custom data fields for resource organization, adaptable timelines
Resource Management	Handles individual and team bookings, round-robin booking, customizable availability	Hierarchical resource organization, import/export branch and room data	Visual resource tracking, conflict detection, suitable for various resource types
Target Audience	Small to medium-sized studios	Businesses needing efficient resource management	Companies needing efficient resource utilization, managing various resources
Customizable Booking Pages	Yes, allows customization of intake forms and email notifications	Yes, allows customization of booking pages	Yes, custom data fields for resource organization

Integration with Other Calendars	Yes, integrates with Google and Outlook calendars	Yes, integrates with Google Calendar and multiple CRMs	Yes, integrates with Google and Outlook Calendar
Payment Integration	Yes, integrates with Stripe for online payments	No	No
Reporting Features	Limited	No	Yes, visual summaries of resource usage and tracking
Scheduling Flexibility	Supports flexible scheduling with round-robin booking and individual event types	Configurable grid view by day or hour	Resource-centric Gantt charts for precise visual planning
Conflict Detection	Not specified	Allows filtering to avoid scheduling conflicts	Yes, conflict detection
Email Notifications	Yes, customizable	Yes, email or pop-up reminders	Yes
QR Code for Room Booking	No	Yes, QR code can be assigned to each room	Not specified
Languages Supported	Multiple language support such as French, Spanish, English, German	Needs improvement to support more languages	Needs improvement to support more languages

CHAPTER 3

System Design

The processes of the project were categorized into different phases in the development, which were project pre-development, data pre-processing, model training architecture building and data training, and prediction on test dataset.

3.1 Overview

This chapter includes 3.2 the use case diagram for University Smart Assistant System, Infinite Book, consists of three main users, user, admin and librarian. 3.3 is the use case description for each use case in this system. 3.4 describes the implementation of mobile app including the code snippet showing how the whole mobile application is developed.

3.2 Use Case

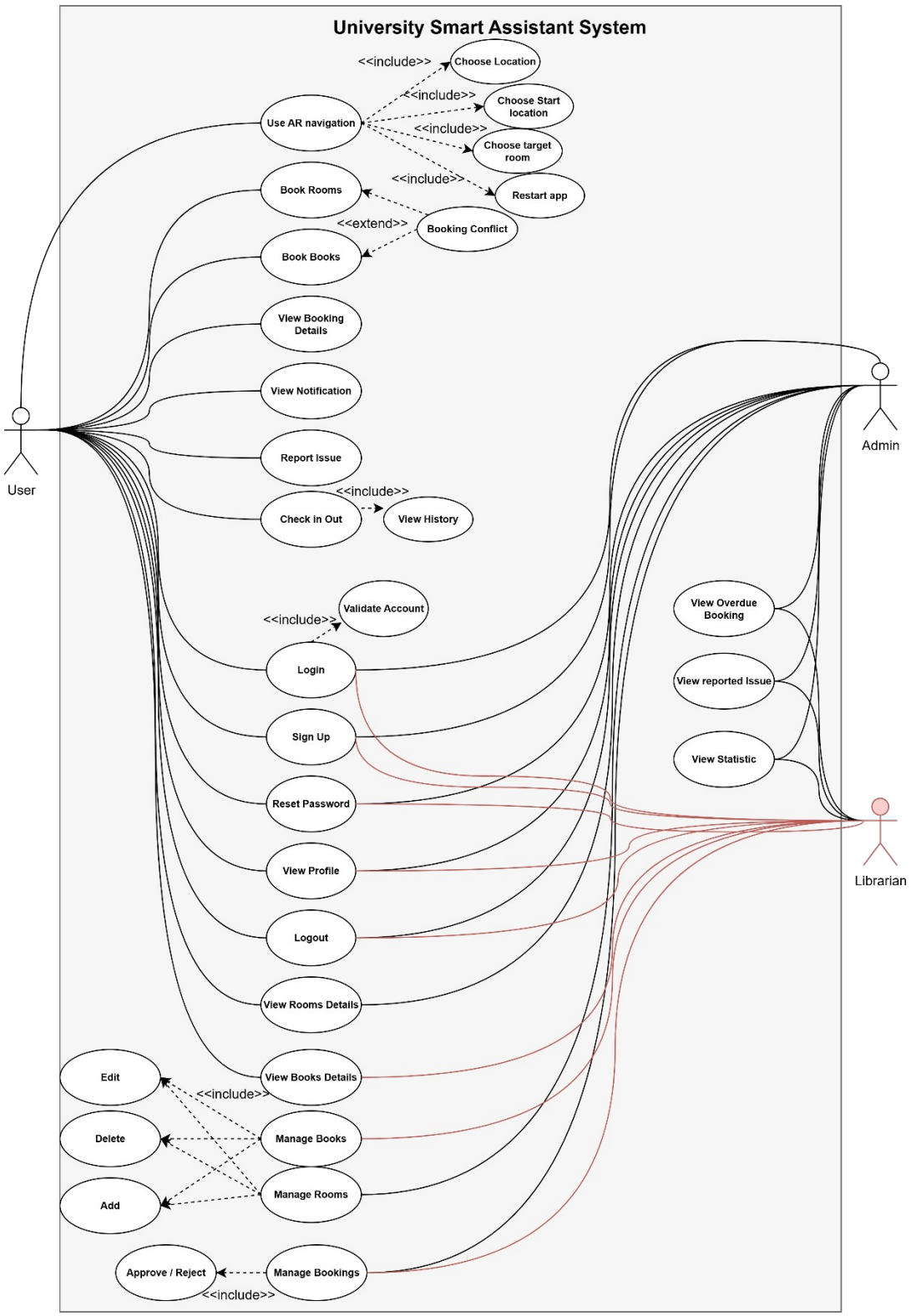


Figure 3.2.1 Use Case Diagram

3.3 Use Case Description

Table 3.3.1 Login Use Case Description

Use Case ID	00001
Use Case Name	Login
Brief Description	Student, administrator is allowed to log into the system using email and password.
Actor	User, Admin, Librarian
Trigger	Go to login page of the mobile app
Precondition	User must had registered for an account in database
Normal flow of events	1. User/Admin/Librarian enters email and password. 2. System validates credentials. 3. System grants access and navigates to main screen.
Sub flows	-
Alternate flows	2a. If the account does not exist, or the combination of email and password is false, error message will prompt to user or admin or librarian, there is a need to re-enter the credentials

Table 3.3.2 Sign up Use Case Description

Use Case ID	00002
Use Case Name	Sign up
Brief Description	User or Admin or Librarian registers a new account by entering the credential information.
Actor	User, Admin, Librarian
Trigger	User/Admin/Librarian initiates sign up process.
Precondition	User/Admin/Librarian is not logged in.
Normal flow of events	1. User/Admin/Librarian enters name, email, password, confirm password, and selects role. 2. System validates inputs. 3. The system creates a user account and stores data. 4. The system navigates users/admins/Librarian to the main screen.
Sub flows	-
Alternate flows	2a. Validation fails ,email does not meet the require email format 2b. System displays error and prompts correction.

Table 3.3.3 Reset Password Use Case Description

Use Case ID	00003
Use Case Name	Reset Password
Brief Description	User or Admin or Librarian requests password reset via email.
Actor	User, Admin, Librarian
Trigger	User/ Admin/Librarian selects forgot password option.

Precondition	User/ Admin/Librarian has a registered email.
Normal flow of events	1. User/ Admin/Librarian requests password reset. 2. System sends password reset email. 3. User/ Admin/Librarian resets password via email link.
Sub flows	-
Alternate flows	2a. Email sending fails. 2b. System displays error message.

Table 3.3.4 View Profile Use Case Description

Use Case ID	00004
Use Case Name	View Profile
Brief Description	User/ Admin/Librarian views their profile information.
Actor	User, Admin,Librarian
Trigger	User/admin/librarian navigates to profile screen.
Precondition	User/admin/librarian is logged in.
Normal flow of events	1. The system retrieves user profile data and booking details. 2. The system displays profile information. 3. The system displays bookings details
Sub flows	-
Alternate flows	1a. Data retrieval fails. 1b. System displays error message.

Table 3.3.5 Booking Conflict Use Case Description

Use Case ID	00005
Use Case Name	Booking Conflict
Brief Description	User can view when booking conflict with others
Actor	User
Trigger	Users select room or book that the time is conflict with others
Precondition	Users select a time and select a room or book
Normal flow of events	1. Users select room/book. 2. Users select time for room/ time range for book. 3. The system checks the booking conflict. 4. System display booking conflict message.
Sub flows	-
Alternate flows	2a. For time range display error message when start time is after end time.

Table 3.3.6 View Bookings Use Case Description

Use Case ID	00006
Use Case Name	View Bookings
Brief Description	User views their current bookings status.
Actor	User
Trigger	User navigates to bookings screen by click the icon view bookings
Precondition	User is logged in and in home page
Normal flow of events	1. The system retrieves bookings for users. 2. The system displays bookings list.

Sub flows	-
Alternate flows	1a. Data retrieval fails. 1b. System displays error message.

Table 3.3.7 Logout Use Case Description

Use Case ID	00007
Use Case Name	Logout
Brief Description	User/Admin/Librarian logs out of the system.
Actor	User, Admin, Librarian
Trigger	User/admin click the logout button
Precondition	User/admin/Librarian is logged in and in user profile or drawer menu
Normal flow of events	1. System ends user session. 2. System navigates to login screen.
Sub flows	-
Alternate flows	-

Table 3.3.8 View Book Details Use Case Description

Use Case ID	00008
Use Case Name	View Book Details

Brief Description	User/Librarian views detailed information about a book.
Actor	User, Librarian
Trigger	User/librarian click the book a book button in home page
Precondition	User/librarian is logged in and in home page
Normal flow of events	1. System retrieves book details. 2. The system displays book information.
Sub flows	-
Alternate flows	1a. Data retrieval fails. 1b. System displays error message.

Table 3.3.9 Book a Book Use Case Description

Use Case ID	00009
Use Case Name	Book a Book
Brief Description	User books an available book.
Actor	User
Trigger	User click the book button on the book he wants to book
Precondition	User is logged in and book is available
Normal flow of events	1. User selects book to book. 2. System checks availability. 3. User select date, time 4. User confirm booking 5. System confirms booking and updates status
Sub flows	-

Alternate flows	2a. The book is not available 3a. If time conflict, not allow user to book and display error message

Table 3.3.10 View Room Details Use Case Description

Use Case ID	00010
Use Case Name	View Room Details
Brief Description	User/Admin views detailed information about a room.
Actor	User, Admin
Trigger	User/admin click the book a room button in home page
Precondition	User/admin is logged in and in home page
Normal flow of events	1. The system retrieves room details. 2. The system displays room information.
Sub flows	-
Alternate flows	1a. Data retrieval fails. 1b. System displays error message.

Table 3.3.11 Book a Room Use Case Description

Use Case ID	00011
Use Case Name	Book a Room
Brief Description	User books an available room.

Actor	User
Trigger	User click the book button on the room he wants to book
Precondition	User is logged in and room is available
Normal flow of events	3. User selects room to book. 4. System checks availability. 5. User select date, time 6. User confirm booking 7. System confirms booking and updates status
Sub flows	-
Alternate flows	2a. The room is not available 3a. If time conflict, not allow user to book and display error message

Table 3.3.12 Manage Rooms Use Case Description

Use Case ID	00012
Use Case Name	Manage Rooms
Brief Description	Admin adds, updates, deletes, and views rooms.
Actor	Admin
Trigger	Admin click the book button on the room he wants to book
Precondition	Admin is logged in and room is available
Normal flow of events	1. Admin logs into the system. 2. Admin navigates to the "Manage Room" module. 3. The system displays a list of existing rooms. 4. Admin selects an action: Add, Edit, or Delete. 5. Admin fills in or updates the necessary details (if applicable).

	6. Admin confirms the action. 7. The system processes the request and updates the room database. 8. A success message is displayed.
Sub flows	4a. Add Room 4a.1 Admin clicks the "Add Room" button. 4a.2 A form appears to input new room details. 4a.3 Admin fills in required fields 4a.4 Admin clicks "Save". 4a.5 System validates and adds the new room to the database. 4a.6 System displays confirmation. 4b. Edit Room 4b.1 Admin selects an existing room from the list. 4b.2 Admin clicks the "Edit Room" button. 4b.3 A form displays with current room details. 4b.4 Admin updates necessary fields. 4b.5 Admin clicks "Save". 4b.6 System validates and updates the room information. 4b.7 System displays confirmation. 4c. Delete Room 4c.1 Admin selects an existing room from the list. 4c.2 Admin clicks the "Delete Room" button. 4c.3 System prompts a confirmation dialog. 4c.4 Admin clicks "Confirm". 4c.5 System deletes the room from the database.
Alternate flows	5a. Invalid or Incomplete Input. System displays validation error message. 7a. Database Update Failure

Table 3.3.13 Manage Books Use Case Description

Use Case ID	00013
Use Case Name	Manage Books
Brief Description	Librarian adds, updates, deletes, and views books.
Actor	Librarian
Trigger	Librarian click the book button on the book he wants to manage
Precondition	Librarian is logged in and book is available
Normal flow of events	1. Librarian logs into the system. 2. Librarian navigates to the "Manage book" module. 3. The system displays a list of existing books. 4. Librarian selects an action: Add, Edit, or Delete. 5. Librarian fills in or updates the necessary details 6. Librarian confirms the action. 7. The system processes the request and updates the room database. 8. A success message is displayed.
Sub flows	4a. Add book 4a.1 Librarian clicks the "Add book" button. 4a.2 A form appears to input new book details. 4a.3 Librarian fills in required fields 4a.4 Librarian clicks "Save". 4a.5 System validates and adds the new book to the database. 4a.6 System displays confirmation. 4b. Edit book 4b.1 Librarian selects an existing book from the list. 4b.2 Librarian clicks the book item for edit. 4b.3 A form displays with current book details. 4b.4 Librarian updates necessary fields.

	4b.5 Librarian clicks "Save". 4b.6 System validates and updates the book information. 4b.7 System displays confirmation. 4c. Delete book 4c.1 Librarian selects an existing book from the list. 4c.2 Librarian clicks the "Delete book" button. 4c.3 System prompts a confirmation dialog. 4c.4 Librarian clicks "Confirm". 4c.5 System deletes the book from the database.
Alternate flows	5a. Invalid or Incomplete Input. System displays validation error message. 7a. Database Update Failure

Table 3.3.14 Ar navigation Use Case Description

Use Case ID	00014
Use Case Name	Select navigation start and target
Brief Description	User can select start location and target location to navigate within the university
Actor	User
Trigger	User click the AR navigation button in home page
Precondition	Users open the unity project
Normal flow of events	1. Users log in into the system 2. Users click the AR navigation button 3. The app displays few buttons for users to choose

Sub flows	3a AR indoor navigation 3a1. Users select floor/block 3a2. Users select start location point 3a3. Users select target location 3a4. System displays navigation arrow and voice guidance to the target 3b Restart app 3b1 The application is restarted.
Alternate flows	-

Table 3.3.15 Manage Bookings Use Case Description

Use Case ID	00016
Use Case Name	Manage Bookings
Brief Description	Admin/Librarian can approve or reject the bookings
Actor	Admin, Librarian
Trigger	Admin/ Librarian click the manage bookings button in the manage module for admin
Precondition	Admin/Librarian is login as admin and click in the manage module
Normal flow of events	1. Admin/ Librarian view all the bookings details. 2. Admin/ Librarian can choose among approve and reject. 3. The status is updated and sent to database as well as user.
Sub flows	-
Alternate flows	3a. Database Update Failure

Table 3.3.16 View Notification Use Case Description

Use Case ID	00016
Use Case Name	View Notification
Brief Description	User can view notification for reserved facilities
Actor	User
Trigger	Users click the notification icon in bottom navigation bar
Precondition	User is login and click in notification
Normal flow of events	1. User click the notification icon below. 2. User view all the notification details. 3. User can choose to clear the notification
Sub flows	-
Alternate flows	3a. Database Update Failure

Table 3.3.17 Check in out Use Case Description

Use Case ID	00017
Use Case Name	Check in out
Brief Description	User can check in or check out for the current approved bookings.
Actor	User
Trigger	Users click the check in out button in dashboard.
Precondition	User is login and click in check in out module

Normal flow of events	1. User click the check in out module. 2. User view all the current approved bookings. 3. User can choose to check in
Sub flows	3a User can choose to check out after checked in.
Alternate flows	3a. Prompt notification if user is checked in but not checked out when the booking is overdue. 3b. Database failure

Table 3.3.18 Report Issue Use Case Description

Use Case ID	00018
Use Case Name	Report Issues
Brief Description	User can report for any issue for the rooms.
Actor	User
Trigger	Users click the report issue button in dashboard.
Precondition	User is login and click report issue module
Normal flow of events	1. User click the report issue module. 2. User fill the room name. 3. User choose the reason for the issue. 4. User can submit the report.
Sub flows	3a User can choose others and fill in the issue not in the choice.
Alternate flows	2a. Database Failure to retrieve room name.

Table 3.3.19 View Reported Issue Use Case Description

Use Case ID	00019
Use Case Name	View Reported Issues
Brief Description	Admin can view and confirm resolution for the issue of the room
Actor	Admin
Trigger	Admin click the view reported issue button in admin dashboard.
Precondition	Admin is login and click view report issue module
Normal flow of events	1. Adminclick the view report issue module. 2. Admin click confirm resolve for the issue. 3. System prompt confirmation dialog. 4. Admin confirm the resolved. 5. Database updates the delete the record of the issues.
Sub flows	-
Alternate flows	5a Database Failure

Table 3.3.20 View Overdue Booking Use Case Description

Use Case ID	00020
Use Case Name	View Overdue booking
Brief Description	Admin/Librarian can view and proceed to phone call for user who checked in but overdue.
Actor	Admin/Librarian
Trigger	Admin/Librarian click the view overdue booking button in admin dashboard.
Precondition	Admin/Librarian is login and click view overdue booking module

Normal flow of events	1. Admin/librarian click the view overdue booking module. 2. Admin view all the overdue booking details. 3. Admin click on the details box. 4. System will open phone with the phone number of the user for admin/librarian to call the user.
Sub flows	-
Alternate flows	-

3.4 Mobile App Development

3.4.1 Database development

The app is implemented using Firebase as server-side, Firebase services is used to provide the best and smooth experience. The google-services.json file as shown in Figure 3.4.1.1 states that the app is connected to Firebase project named "bookingandar," with a real-time database hosted in Asia Southeast. This real-time database serves as database for the app, it is enabling real-time synchronization of data such as user profiles, bookings, rooms, and books which are needed for this project. By using Firebase Realtime Database, the app can reduce latency for updating the information, ensuring that users and admins can interact with the system smoothly even with network connectivity.

Backend-as-a-Service (BaaS) platform used by Firebase gives access to cloud-hosted services. Fundamentally, Firebase Realtime Database is a NoSQL cloud database that synchronizes data in real-time to all connected clients and stores it as JSON. This enables highly responsive and a fast applications by ensuring that all clients listening to the database receive updates almost instantly when data changes in the database. Additionally, Firebase Authentication foster the application's sign-up, login, and role-based access implementation securely.

The app also uses Firebase's real-time database references extensively in its ViewModel and Fragment classes to load, update, and delete data. For example, the AdminViewModel interacts with the database to fetch lists of rooms, books, and bookings, and to perform

CRUD operations on rooms and update booking statuses. Similarly, user-related fragments query the database to display user profiles, bookings, and book details. This tight integration with Firebase's real-time database allows the app to maintain a consistent and up-to-date state across all clients, with changes propagated instantly to all connected users.

Overall, the use of Firebase as the backend enables rapid development and deployment of the app without the need to manage traditional server infrastructure. The google-services.json file configures the app with the necessary credentials and endpoints to securely communicate with Firebase services. This architecture provides scalability, security, and real-time data synchronization, which are critical for the booking and management functionalities offered by the app for both users and administrators.

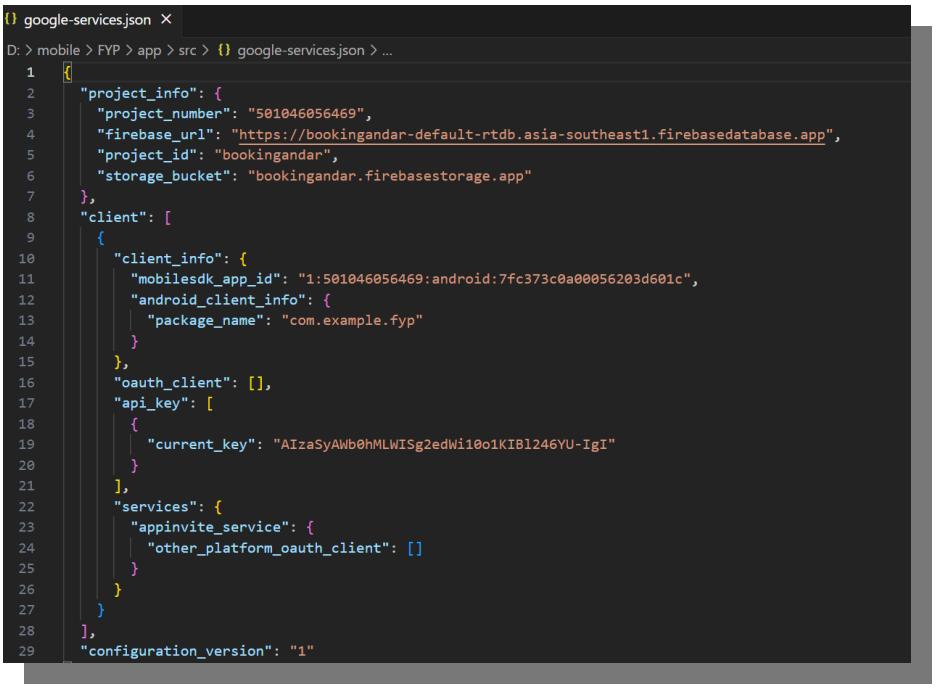
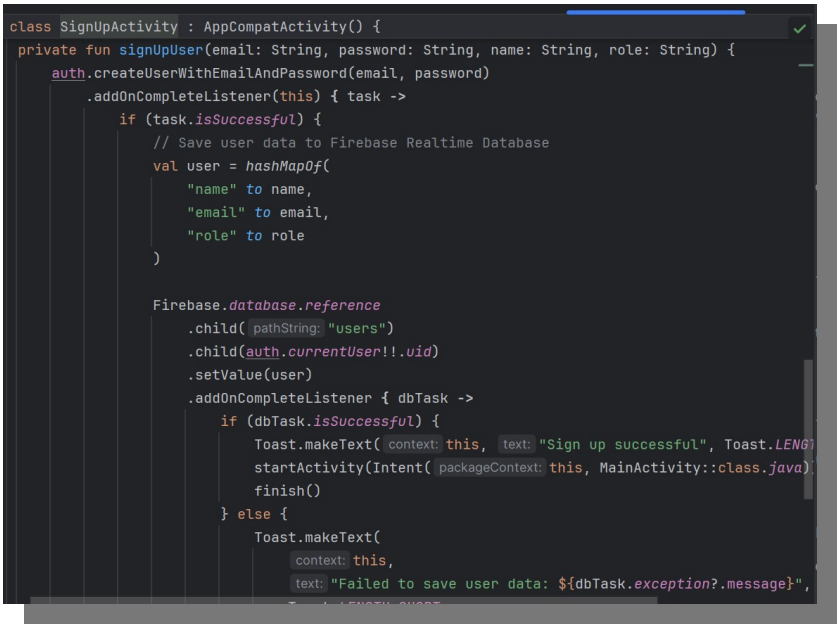


Figure 3.4.1.1 google-services.json

3.4.2 Login Module Development

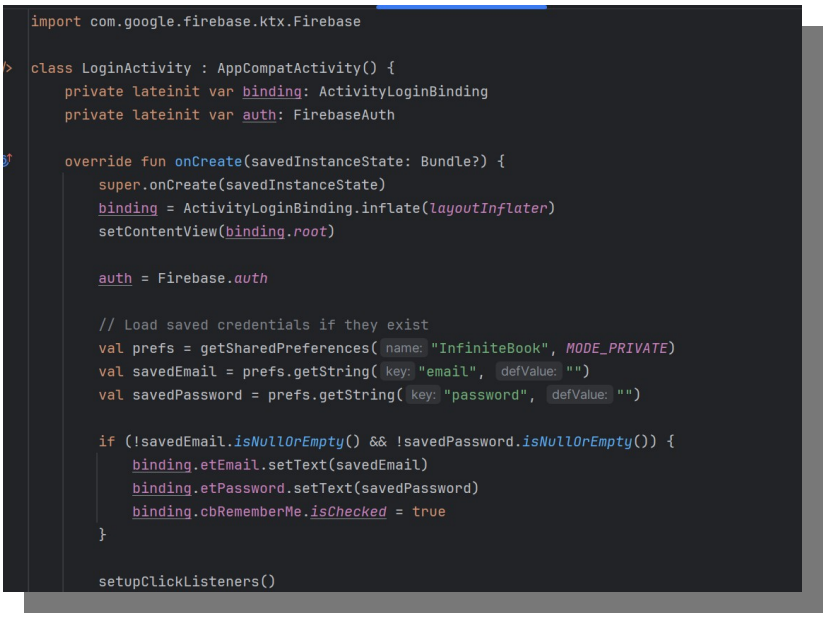
Authentication is used through Firebase Authentication, which could manage user sign-up, login, and password recovery securely. The app uses Firebase Auth SDK to allow users to register with email and password and authenticate themselves in the app and the data is saved to the real time database as shown in figure 3.4.2.1 using Signup Activity Kotlin file. For the roles can be set in database. For this project initially I would set role selection because it eases the testing part. This integration simplifies user management by giving complex

security and session management to Firebase, while also offering some useful features like "remember me" as shown in the code snippet figure 3.4.2.2 using shared preference and password reset via email as shown in code snippet figure 3.4.2.3. The authentication state is used throughout the app to control access to different features and UI components, ensuring that only authorized users can login and only allow admin to do on admin-related module.



```
class SignUpActivity : AppCompatActivity() {
private fun signUpUser(email: String, password: String, name: String, role: String) {
    auth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                // Save user data to Firebase Realtime Database
                val user = hashMapOf(
                    "name" to name,
                    "email" to email,
                    "role" to role
                )

                Firebase.database.reference
                    .child( pathString: "users")
                    .child(auth.currentUser!!.uid)
                    .setValue(user)
                    .addOnCompleteListener { dbTask ->
                        if (dbTask.isSuccessful) {
                            Toast.makeText( context: this,  text: "Sign up successful", Toast.LENG
                            startActivity(Intent( packageContext: this, MainActivity::class.java)
                            finish()
                        } else {
                            Toast.makeText(
                                context: this,
                                text: "Failed to save user data: ${dbTask.exception?.message}",
```

Figure 3.4.2.1 SignUpActivity.kt



```
import com.google.firebase.ktx.Firebase

class LoginActivity : AppCompatActivity() {
    private lateinit var binding: ActivityLoginBinding
    private lateinit var auth: FirebaseAuth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityLoginBinding.inflate(layoutInflater)
        setContentView(binding.root)

        auth = Firebase.auth

        // Load saved credentials if they exist
        val prefs = getSharedPreferences( name: "InfiniteBook", MODE_PRIVATE)
        val savedEmail = prefs.getString( key: "email",  defValue: "")
        val savedPassword = prefs.getString( key: "password",  defValue: "")

        if (!savedEmail.isNullOrEmpty() && !savedPassword.isNullOrEmpty()) {
            binding.etEmail.setText(savedEmail)
            binding.etPassword.setText(savedPassword)
            binding.cbRememberMe.isChecked = true
        }

        setupClickListeners()
```

Figure 3.4.2.2 LoginActivity.kt

```
class ForgotPasswordActivity : AppCompatActivity() {
    private fun setupClickListeners() {
    }

    private fun resetPassword(email: String) {
        auth.sendPasswordResetEmail(email)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    Toast.makeText(
                        context: this,
                        text: "Password reset email sent. Please check your inbox.",
                        Toast.LENGTH_LONG
                    ).show()
                    finish()
                } else {
                    Toast.makeText(
                        context: this,
                        text: "Failed to send reset email: ${task.exception?.message}",
                        Toast.LENGTH_SHORT
                    ).show()
                }
            }
    }
}
```

Figure 3.4.2.3 ForgetPassword.kt

3.4.3 Bookings module Development for User

The bookings module allows both users and admins to use the bookings efficiently. Users can create new bookings through a booking dialog where they select the item whether book or room, date, and time slots. The system automatically checks for conflicts with existing bookings for books and rooms to prevent double-booking, ensuring that users cannot reserve the same item for overlapping times. This conflict management is done by fetching existing bookings in the firebase database for the selected item and date, then comparing the requested time slot against these to detect overlaps. If a conflict is found, the user is notified and prevented from confirming the booking the button would be disabled until a free time slot is chosen. This approach helps maintain the integrity of bookings and provides a smooth user experience.

Models are important for a container to save the information and then update in database there would be three models included in this development as shown in figure 3.4.3.1 to 3.4.3.3, Book, Room and Bookings model, each handle a particular class in firebase database.

Conflict management is one of the most important functions of the bookings module whether in client side or server side The system uses precise time comparisons with ISO 8601 timestamps to detect overlapping bookings. This ensures that no two bookings for the same item can occupy the same time slot, preventing scheduling errors as shown for the function checkAndDisplayConflict in figure 3.4.3.4 for the BookingDialogFragment.kt in this application. There are also findConflictBooking to checks for time overlaps with the selected time slot across all rooms.

hasBookingConflict() to check checking for both same-room and user-specific conflicts, showDetailedRoomConflictMessage to formats and displays detailed conflict messages for room bookings. The UI reflects conflicts clearly by disabling booking confirmation and showing detailed messages about the conflicting bookings, helping users to understand what is the time being occupied and which is not.

Moreover figure 3.4.3.6 saveBooking function, shows that it fetches the current bookings for the same date and check again the time will conflict or not then only save to the database.

Book.kt

```kotlin
package com.example.fyp.model

import android.os.Parcelable
import kotlinx.parcelize.Parcelize

@Parcelize
data class Book(
    var id: String = "",
    val title: String = "",
    val author: String = "",
    val isbn: String = "",
    val status: String = "available",
    val location: String = "",
    val category: String = "", // fiction, non-fiction, textbook, etc.
    val createdAt: String = "", // ISO 8601 format
    val updatedAt: String = "" // ISO 8601 format
) : Parcelable
```

Figure 3.4.3.1 Book.kt model

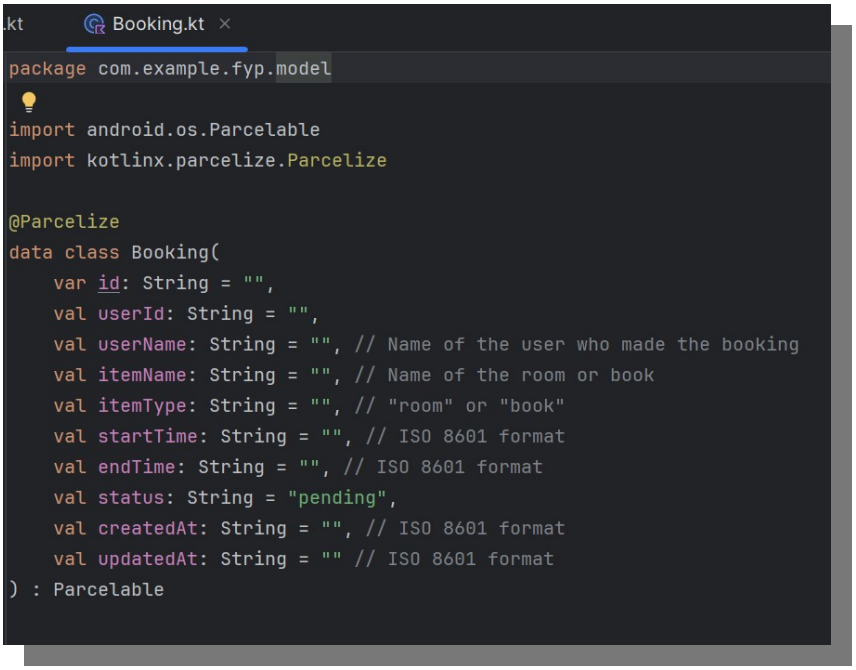


Figure 3.4.3.2 Booking.kt model

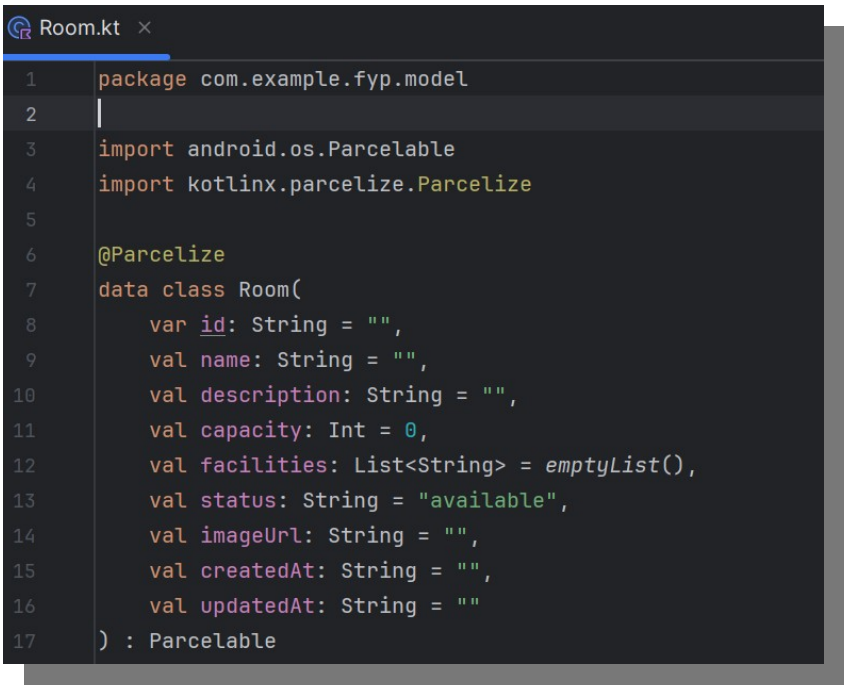
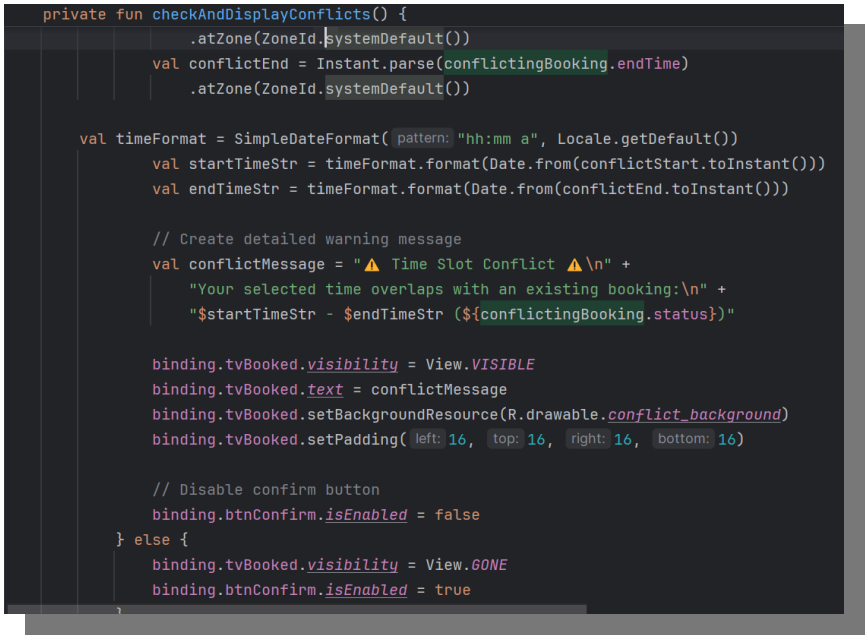
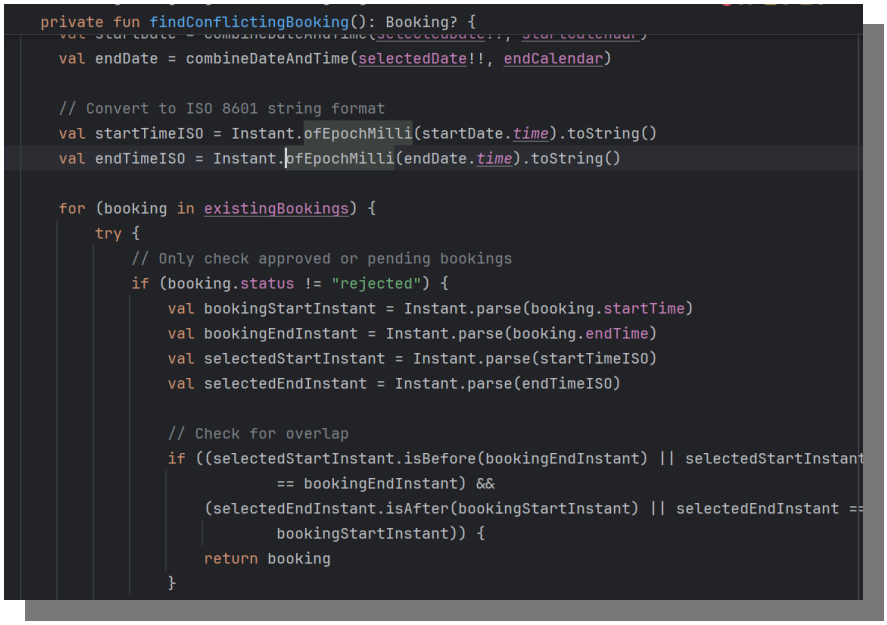


Figure 3.4.3.3 Room.kt model



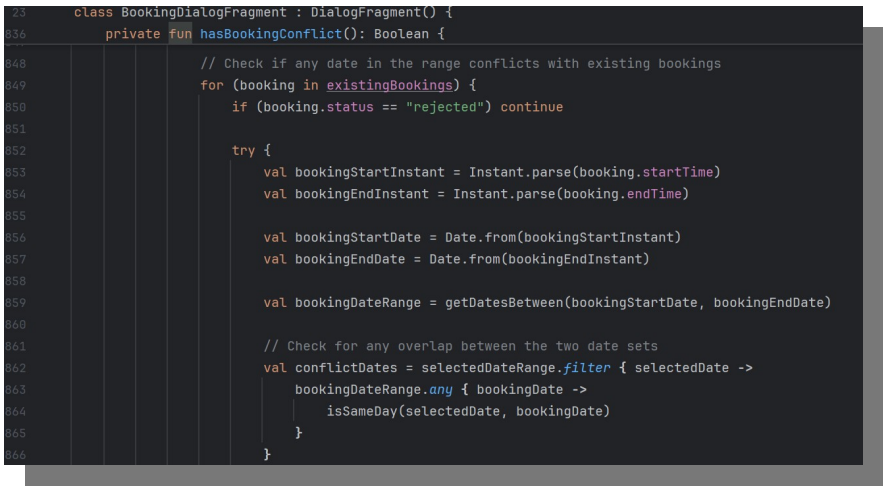
```
private fun checkAndDisplayConflicts() {
                .atZone(ZoneId.systemDefault())
            val conflictEnd = Instant.parse(conflictingBooking.endTime)
                .atZone(ZoneId.systemDefault())

        val timeFormat = SimpleDateFormat( pattern: "hh:mm a", Locale.getDefault())
            val startTimeStr = timeFormat.format(Date.from(conflictStart.toInstant()))
            val endTimeStr = timeFormat.format(Date.from(conflictEnd.toInstant()))

            // Create detailed warning message
            val conflictMessage = "⚠ Time Slot Conflict ⚠\n" +
                "Your selected time overlaps with an existing booking:\n" +
                "$startTimeStr - $endTimeStr (${conflictingBooking.status})"

            binding.tvBooked.visibility = View.VISIBLE
            binding.tvBooked.text = conflictMessage
            binding.tvBooked.setBackgroundResource(R.drawable.conflict_background)
            binding.tvBooked.setPadding( left: 16,  top: 16,  right: 16,  bottom: 16)

            // Disable confirm button
            binding.btnConfirm.isEnabled = false
        } else {
            binding.tvBooked.visibility = View.GONE
            binding.btnConfirm.isEnabled = true
```

Figure 3.4.3.4 checkAndDisplayConflict function in BookingDialogFragment.kt



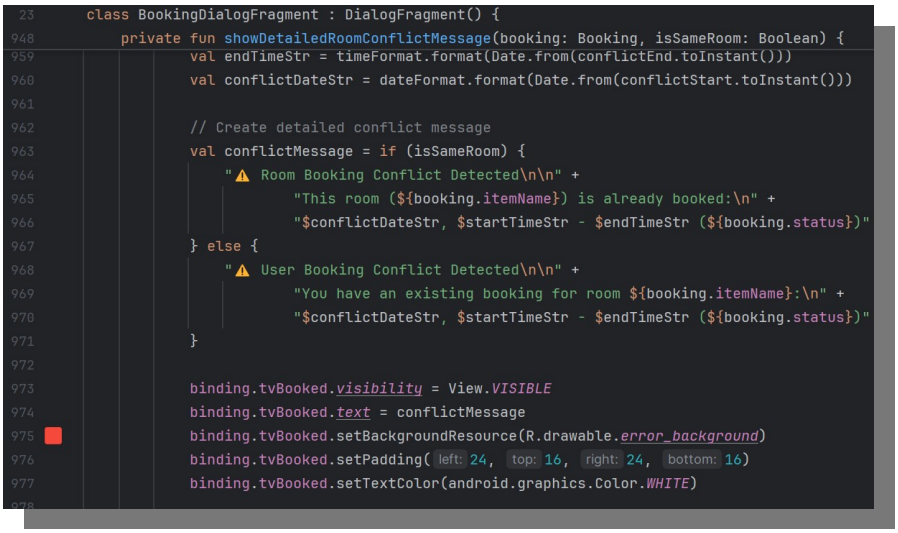
```
private fun findConflictingBooking(): Booking? {
    val endDate = combineDateAndTime(selectedDate!!, endCalendar)

    // Convert to ISO 8601 string format
    val startTimeISO = Instant.ofEpochMilli(startDate.time).toString()
    val endTimeISO = Instant.ofEpochMilli(endDate.time).toString()

    for (booking in existingBookings) {
        try {
            // Only check approved or pending bookings
            if (booking.status != "rejected") {
                val bookingStartInstant = Instant.parse(booking.startTime)
                val bookingEndInstant = Instant.parse(booking.endTime)
                val selectedStartInstant = Instant.parse(startTimeISO)
                val selectedEndInstant = Instant.parse(endTimeISO)

                // Check for overlap
                if ((selectedStartInstant.isBefore(bookingEndInstant) || selectedStartInstant
                        == bookingEndInstant) &&
                    (selectedEndInstant.isAfter(bookingStartInstant) || selectedEndInstant ==
                            bookingStartInstant)) {
                    return booking
                }
```

Figure 3.4.3.5 findConflictBooking function in BookingDialogFragment.kt



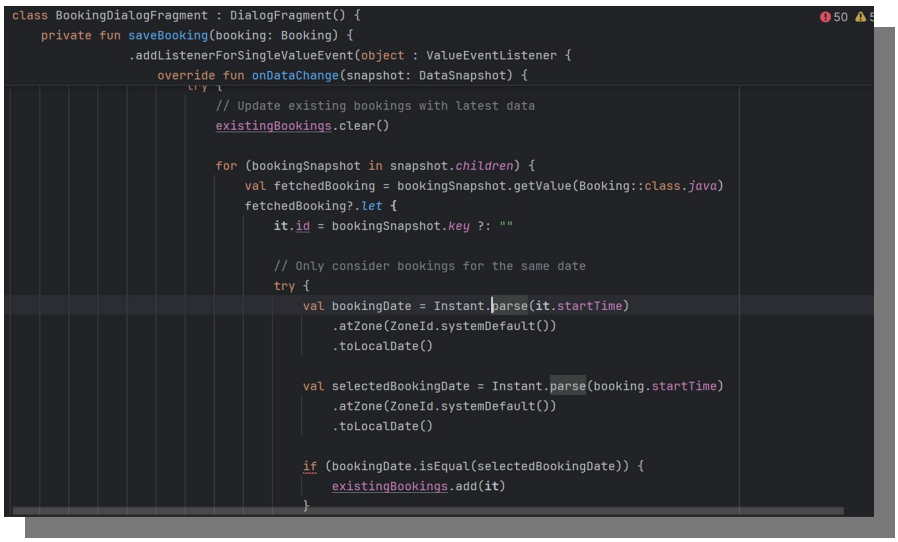
```
class BookingDialogFragment : DialogFragment() {
    private fun hasBookingConflict(): Boolean {
        // Check if any date in the range conflicts with existing bookings
        for (booking in existingBookings) {
            if (booking.status == "rejected") continue

            try {
                val bookingStartInstant = Instant.parse(booking.startTime)
                val bookingEndInstant = Instant.parse(booking.endTime)

                val bookingStartDate = Date.from(bookingStartInstant)
                val bookingEndDate = Date.from(bookingEndInstant)

                val bookingDateRange = getDatesBetween(bookingStartDate, bookingEndDate)

                // Check for any overlap between the two date sets
                val conflictDates = selectedDateRange.filter { selectedDate ->
                    bookingDateRange.any { bookingDate ->
                        isSameDay(selectedDate, bookingDate)
                    }
                }
```

Figure 3.4.3.6 hasBookingConflict() function in BookingDialogFragment.kt



```
class BookingDialogFragment : DialogFragment() {
    private fun showDetailedRoomConflictMessage(booking: Booking, isSameRoom: Boolean) {
        val endTimeStr = timeFormat.format(Date.from(conflictEnd.toInstant()))
        val conflictDateStr = dateFormat.format(Date.from(conflictStart.toInstant()))

        // Create detailed conflict message
        val conflictMessage = if (isSameRoom) {
            "⚠️ Room Booking Conflict Detected\n\n" +
                    "This room (${booking.itemName}) is already booked:\n" +
                    "$conflictDateStr, $startTimeStr - $endTimeStr (${booking.status})"
        } else {
            "⚠️ User Booking Conflict Detected\n\n" +
                    "You have an existing booking for room ${booking.itemName}:\n" +
                    "$conflictDateStr, $startTimeStr - $endTimeStr (${booking.status})"
        }

        binding.tvBooked.visibility = View.VISIBLE
        binding.tvBooked.text = conflictMessage
        binding.tvBooked.setBackgroundResource(R.drawable.error_background)
        binding.tvBooked.setPadding( left: 24,  top: 16,  right: 24,  bottom: 16)
        binding.tvBooked.setTextColor(android.graphics.Color.WHITE)
```

Figure 3.3.3.3.7 showDetailedRoomConflictMessage function in BookingDialogFragment.kt



```
class BookingDialogFragment : DialogFragment() {
    private fun saveBooking(booking: Booking) {
                .addListenerForSingleValueEvent(object : ValueEventListener {
                    override fun onDataChange(snapshot: DataSnapshot) {
                        // Update existing bookings with latest data
                        existingBookings.clear()

                        for (bookingSnapshot in snapshot.children) {
                            val fetchedBooking = bookingSnapshot.getValue(Booking::class.java)
                            fetchedBooking?.let {
                                it.id = bookingSnapshot.key ?: ""

                                // Only consider bookings for the same date
                                try {
                                    val bookingDate = Instant.parse(it.startTime)
                                        .atZone(ZoneId.systemDefault())
                                        .toLocalDate()

                                    val selectedBookingDate = Instant.parse(booking.startTime)
                                        .atZone(ZoneId.systemDefault())
                                        .toLocalDate()

                                    if (bookingDate.isEqual(selectedBookingDate)) {
                                        existingBookings.add(it)
                                    }
```

Figure 3.3.3.3.8 saveBooking function in BookingDialogFragment.kt

3.4.4 Resources Management module For Admin and librarian

The admin management module provides comprehensive tools for managing books, rooms, and bookings. For books, librarians can view the list of all books, add new books, edit existing ones, and delete books that are no longer needed. This is handled in the ManageBooksFragment, which interacts with Firebase Realtime Database to keep the book data real time updated. Some function had shown in figure 3.4.4.1 to 3.4.4.3 for the add/edit and also delete function which is implemented using dialog fragment. A dialog pops out for user to edit, add or delete the current books details. The UI uses adapters and dialogs to facilitate these jobs more easily.

Similarly, room management is handled in the ManageRoomsFragment, where admins can manage the list of rooms available for booking. They can add new rooms with details such as capacity of the rooms, edit room information, and delete rooms. This module is similar to books management module with add, edit and delete dialog fragments which will pop out when admin choose to edit, add or delete the current rooms details and save the information to firebase after the modification.

Booking management for admins and librarians is applied and use in the ManageBookingsFragment, where they can see all the bookings detail by fetching from firebase database as shown in Figure 3.4.4.4 where the loadBooking function is used for

loading all the information for bookings in firebase database, admin also can update their statuses and manage notifications. This module ties together the booking lifecycle from creation to approval or rejection, providing admins with control the bookings made by user or admin himself. Together, these admin and librarian modules form a advanced backend management system that supports the bookings functionality and maintains data integrity across books, rooms, and bookings.

```
class AddEditBookDialogFragment : DialogFragment() {
    private fun saveBook() {
            // For new books
            val bookId = generateBookId()
            val bookRef = database.getReference( path: "books").child(bookId)
            val bookData = Book(
                id = bookId,
                title = title,
                author = author,
                isbn = isbn,
                status = status.lowercase(),
                location = location,
                category = category.lowercase(),
                createdAt = currentTime,
                updatedAt = currentTime
            )

            bookRef.setValue(bookData)
                .addOnSuccessListener {
                    if (isAdded) {
                        Toast.makeText(context, text: "Book added successfully", Toast.LENGTH_SHORT).show()
                        dismissSafely()
                    }
                }
```

Figure 3.4.4.1 saveBook function in AddEditBookDialogFragment.kt for add new book

```
class AddEditBookDialogFragment : DialogFragment() {
    private fun saveBook() {
        try {
            if (book != null) {
                // For existing books
                val bookRef = database.getReference( path: "books").child(book!!.id)
                val bookData = Book(
                    id = book!!.id,
                    title = title,
                    author = author,
                    isbn = isbn,
                    status = status.lowercase(),
                    location = location,
                    category = category.lowercase(),
                    createdAt = book?.createdAt ?: currentTime,
                    updatedAt = currentTime
                )

                bookRef.setValue(bookData)
                    .addOnSuccessListener {
                        if (isAdded) {
                            Toast.makeText(context, text: "Book updated successfully", Toast.LENGTH_SHORT).show()
                            dismissSafely()
                        }
                    }
```

Figure 3.4.4.2 saveBook function in AddEditBookDialogFragment.kt for edit existing book

```kotlin
class DeleteBookDialogFragment : DialogFragment() {
    private fun setupClickListeners() {
        btnDelete.setOnClickListener { deleteBook() }
    }
}

private fun deleteBook() {
    book?.let {
        FirebaseDatabase.getInstance().getReference( path: "books")
            .child(it.id)
            .removeValue()
            .addOnSuccessListener {
                Toast.makeText(context, text: "Book deleted successfully", Toast.LENGTH_SHORT).show()
                dismiss()
            }
            .addOnFailureListener { e ->
                Toast.makeText(context, text: "Error deleting book: ${e.message}", Toast.LENGTH_SHORT).show()
            }
    }
}
```

Figure 3.4.4.3 deleteBook function in DeleteBookDialogFragment.kt for deleting existing Book

```kotlin
ManageBookingsFragment.kt
class ManageBookingsFragment : Fragment() {

private fun loadBookings() {
    binding.progressBar.visibility = View.VISIBLE
    FirebaseDatabase.getInstance().getReference( path: "bookings")
        .addValueEventListener(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                try {
                bookings.clear()
                for (bookingSnapshot in snapshot.children) {
                    val booking = bookingSnapshot.getValue(Booking::class.java)
                    booking?.let {
                        // Set the ID from the snapshot key
                        it.id = bookingSnapshot.key ?: ""
                        bookings.add(it)
                    }
                }
                    bookingsAdapter.submitList(bookings.toList())
                updateEmptyView()
                } catch (e: Exception) {
                    Toast.makeText(context, text: "Error loading bookings: ${e.message}", Toast.LENGTH_SHORT).show()
                } finally {
                binding.progressBar.visibility = View.GONE
                }
            }
```

Figure 3.4.4.4 loadBookings function in ManageBookingsFragment

```kotlin
class ManageBookingsFragment : Fragment() {
    private fun updateBookingStatus(booking: Booking, status: String) {
            return
        }

        binding.progressBar.visibility = View.VISIBLE

        // Create a map with both status and updatedAt fields
        val updates = mapOf(
            "status" to status,
            "updatedAt" to Instant.now().toString()
        )

    val bookingRef = FirebaseDatabase.getInstance().getReference( path: "bookings").child(booking.id)
        bookingRef.updateChildren(updates)
        .addOnSuccessListener {
                try {
                    if (isAdded && context != null) {
            Toast.makeText(context, text: "Booking status updated", Toast.LENGTH_SHORT).show()
                    }
                    // Create notification for the user safely in background
                    createBookingStatusNotification(booking, status)
                } catch (e: Exception) {
                    Log.e( tag: "ManageBookingsFragment", msg: "Error in update success: ${e.message}", e)
                }
```

Figure 3.4.4.5 updateBookingStatus function in ManageBookingsFragment

3.4.5 View reported issue module for admin

View reported issue module retrieve all the data from issue table and display out to admin. It request confirmation from admin If admin confirm that a issue is resolved.As shown in 3.4.5.1,all the issues are load from database. As shown in 3.4.5.2, after admin press the resolved button, system will request confirmation from admin.

```
class AdminIssuesActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
    }

    private fun loadIssues() {
        val dbRef = FirebaseDatabase.getInstance().getReference( path: "issues")
        dbRef.addValueEventListener(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                issues.clear()
                for (child in snapshot.children) {
                    val issue = child.getValue(Issue::class.java)
                    issue?.let { issues.add(it) }
                }
                adapter.notifyDataSetChanged()

                // Show/hide based on issue count
                if (issues.isEmpty()) {
                    tvNoIssues.visibility = View.VISIBLE
                    recyclerView.visibility = View.GONE
                } else {
                    tvNoIssues.visibility = View.GONE
                    recyclerView.visibility = View.VISIBLE
                }
            }

            override fun onCancelled(error: DatabaseError) {}
```

Figure 3.4.5.1 Load all issues from database

```
class IssueAdapter(private val issues: List<Issue>) :                    ⚠6
    inner class IssueViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

fun bind(issue: Issue) {
    tvRoom.text = "Room: ${issue.roomName}"
    tvType.text = "Issue: ${issue.issueType}"
    tvStatus.text = "Status: ${issue.status}"

    btnResolve.setOnClickListener {
        val context = itemView.context
        AlertDialog.Builder(context)
            .setTitle("Confirm Resolution")
            .setMessage("Are you sure you want to mark this issue as resolved? This will remove i
            .setPositiveButton( text: "Yes") { _, _ ->
                val dbRef = FirebaseDatabase.getInstance().getReference( path: "issues")
                issue.id?.let { dbRef.child(it).removeValue() }
            }
            .setNegativeButton( text: "Cancel",  listener: null)
            .show()
    }
}
```

Figure 3.4.5.2 Request Confirmation

3.4.6 View Overdue Booking module for admin and librarian

View Overdue Booking will retrieve all the booking details from database, it will search for those who checked in but not check out and display the details including phone number to admin or librarian, then they can press the details and the system will navigate the admin and librarian to their own phone's call and automatically enter the user's phone number for admin and librarian to contact the user. As shown in figure 3.4.6.1, the system checks for user who are checked in but not check out and exceed the end time of the booking. For figure 3.4.6.2 shows the system open user's phone call and passes the phone number.

```
class AdminCheckBookingsActivity : AppCompatActivity(), AdminBookingAdapter.OnBookingCl ⚠2
    private fun loadCheckedInBookings() {
        dbRef.orderByChild( path: "status").equalTo( value: "checked_in")
            .addValueEventListener(object : ValueEventListener {
                override fun onDataChange(snapshot: DataSnapshot) {
                    bookings.clear()
                    for (child in snapshot.children) {
                        val booking = child.getValue(Booking::class.java)
                        if (booking != null) {
                            userRef.child( pathString: booking.userId ?: "").addListenerForSingleVal
                                override fun onDataChange(userSnap: DataSnapshot) {
                                    val users = userSnap.getValue(Users::class.java)
                                    val phone = users?.phone ?: "Unknown"
                                    bookings.add(Pair(booking, phone))
                                    adapter.notifyDataSetChanged()

                                    // toggle views
                                    binding.tvNoBookings.visibility =
                                        if (bookings.isEmpty()) View.VISIBLE else View.GONE
                                    binding.recyclerAdminBookings.visibility =
                                        if (bookings.isEmpty()) View.GONE else View.VISIBLE
                                }
```

Figure 3.4.6.1 Load overdue user

```
override fun onBookingClick(booking: Booking, phone: String) {
    if (phone != "Unknown") {
        val dialIntent = Intent(Intent.ACTION_DIAL)
        dialIntent.data = Uri.parse( uriString: "tel:$phone")
        startActivity(dialIntent)
    } else {
        Toast.makeText( context: this,  text: "Phone number not available", Toast.LENGTH_SHORT)
    }
}
```

Figure 3.4.6.2 Open call application and pass the user phone number.

3.4.7 Statistic module for admin

Statistic module retrieve all the data from firebase database, from the history table. In this module system will display pie chart data for how many bookings made by user for each resources, arrange by past month, past week or all. Admin can view the pie chart analysis for book and admin can view the pie chart analysis for room. System will check for the user role before displaying the correct details As shown in figure 3.4.7.1, the system check for user role before displaying. For figure 3.4.7.2 shows the pie chart constructor to construct the pie chart for data analysis. For figure 3.4.7.3 shows code snippet that fetch data from database and display to statistic pie chart

```
class StatisticsFragment : Fragment() {                                    4 ^
    private fun checkUserRoleAndSetupUI() {
        usersRef.child(userId).addListenerForSingleValueEvent(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {

                //  Role-based chart visibility
                when (role) {
                    "admin" -> {
                        binding.textBook.visibility= View.GONE
                        binding.textRoom.visibility= View.VISIBLE
                        binding.pieChartRooms.visibility = View.VISIBLE
                        binding.pieChartBooks.visibility = View.GONE
                    }
                    "librarian" -> {
                        binding.textBook.visibility= View.VISIBLE
                        binding.textRoom.visibility= View.GONE
                        binding.pieChartRooms.visibility = View.GONE
                        binding.pieChartBooks.visibility = View.VISIBLE
                    }
                    else -> {
                        binding.pieChartRooms.visibility = View.GONE
                        binding.pieChartBooks.visibility = View.GONE
                    }
                }
```

Figure 3.4.7.1 Check user role for statistics

```
class CustomPieChart @JvmOverloads constructor(                    6 2 ^

    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)
        labelBounds.clear()
        if (data.isEmpty()) {
            canvas.drawText( text: "No data available",  x: 60f,  y: height / 2f, titlePaint)
            return
        }

        val width = width.toFloat()
        val height = height.toFloat()
        val padding = 60f
        val total = data.values.sum().toFloat().takeIf { it > 0 } ?: 1f
        var startAngle = 0f

        // Draw title
        canvas.drawText(title, padding, padding, titlePaint)

        // Setup pie chart bounds
        val radius = min( a: width / 2,  b: height - 2 * padding - 120f) / 2
        rectF.set(
            left: width / 2 - radius,
            top: padding + 60f,
            right: width / 2 + radius,
            bottom: padding + 60f + 2 * radius
```

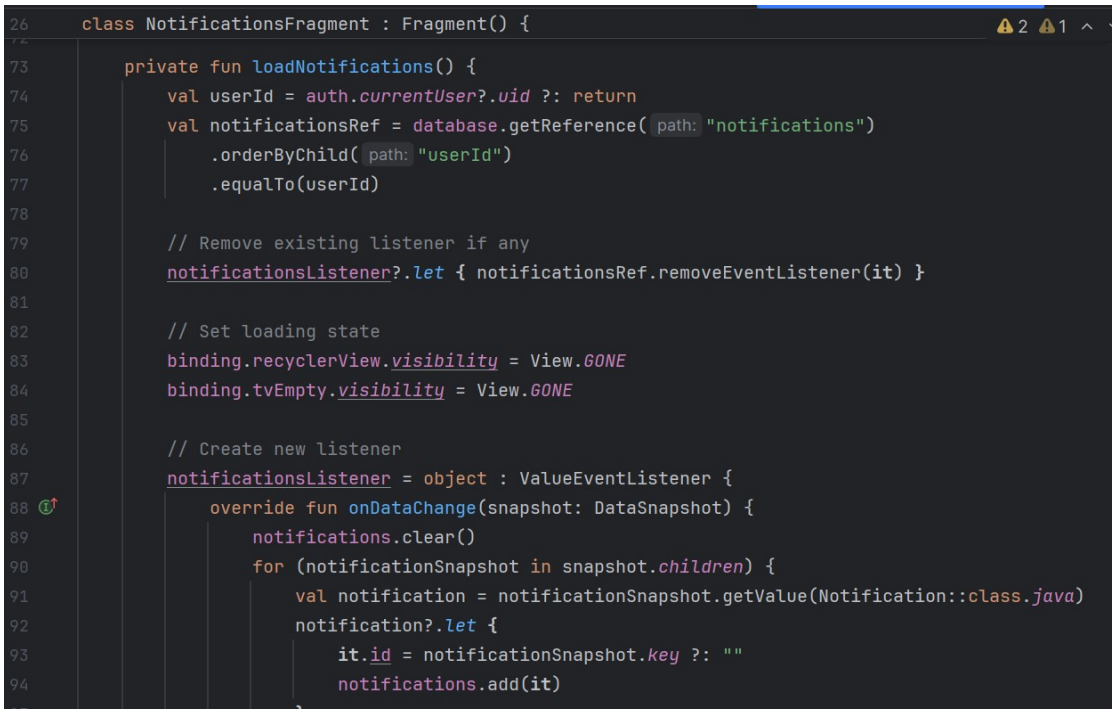
Figure 3.4.7.2 Pie chart constructor

```
class StatisticsFragment : Fragment() {                                    4 ^
    private fun fetchAndDisplayStatistics(filter: String, role: String) {
        roomsRef.addListenerForSingleValueEvent(object : ValueEventListener {
            override fun onDataChange(roomsSnapshot: DataSnapshot) {
                val allRooms = mutableMapOf<String, Int>()
                roomsSnapshot.children.forEach { room ->
                    val roomName = room.child( path: "name").getValue(String::class.java) ?: retur
                    allRooms[roomName] = 0
                }

                booksRef.addListenerForSingleValueEvent(object : ValueEventListener {
                    override fun onDataChange(booksSnapshot: DataSnapshot) {
                        val allBooks = mutableMapOf<String, Int>()
                        booksSnapshot.children.forEach { book ->
                            val bookTitle = book.child( path: "title").getValue(String::class.java
                            allBooks[bookTitle] = 0
                        }

                        historyRef.addListenerForSingleValueEvent(object : ValueEventListener {
                            override fun onDataChange(bookingsSnapshot: DataSnapshot) {
                                val roomTotal = allRooms.toMutableMap()
                                val bookTotal = allBooks.toMutableMap()
                                val calendar = Calendar.getInstance()
                                val currentTime = Calendar.getInstance().apply { time = Date() }
```

Figure 3.4.7.3 Fetch and display statistic

3.4.8 Notification module for user

Notification module will show notification to user when a booking is made or approved/rejected by admin or librarian as shown in 3.4.8.1. Notification also can be send to user as a pop out notification or in app notification dialog to notify user when user is checked in but not checked out in check in out module as shown in figure 3.4.8.2.



```
class NotificationsFragment : Fragment() {
    private fun loadNotifications() {
        val userId = auth.currentUser?.uid ?: return
        val notificationsRef = database.getReference( path: "notifications")
            .orderByChild( path: "userId")
            .equalTo(userId)

        // Remove existing listener if any
        notificationsListener?.let { notificationsRef.removeEventListener(it) }

        // Set loading state
        binding.recyclerView.visibility = View.GONE
        binding.tvEmpty.visibility = View.GONE

        // Create new listener
        notificationsListener = object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                notifications.clear()
                for (notificationSnapshot in snapshot.children) {
                    val notification = notificationSnapshot.getValue(Notification::class.java)
                    notification?.let {
                        it.id = notificationSnapshot.key ?: ""
                        notifications.add(it)
```

Figure 3.4.8.1 Load notification for booked resources and show approve/reject

```
class MainActivity : AppCompatActivity() {

    private fun checkForCheckedInBookings() {
        val userId = auth.currentUser?.uid ?: return
        val dbRef = FirebaseDatabase.getInstance().getReference( path: "bookings")
        dbRef.orderByChild( path: "userId").equalTo(userId)
            .addListenerForSingleValueEvent(object : ValueEventListener {
                override fun onDataChange(snapshot: DataSnapshot) {
                    val checkedInBookings = mutableListOf<String>()
                    for (child in snapshot.children) {
                        val booking = child.getValue(Booking::class.java)
                        if (booking != null && booking.status == "checked_in" && booking.itemNam
                            checkedInBookings.add(booking.itemName)
                        }
                    }
                    if (checkedInBookings.isNotEmpty()) {
                        showCheckedInDialog()
                    }
                }

                override fun onCancelled(error: DatabaseError) {
                    Toast.makeText( context: this@MainActivity,  text: "Failed to check bookings", T
                }
            })
```

Figure 3.4.8.2 check for overdue booking for user

3.4.9 Check in out module for user

Check in out module is also one of the most important module in this application because it allows user to check in and check out after booking. Here user also can view their history for the booking that is completed. As shown in figure 3.4.9.1, the check in out module fetch data from bookings table for those bookings which is approved by admin or librarian then display to user for the check in out functionality. In figure 3.4.9.2 shows code for the history activity which user can view their bookings history here.

```
class CheckInOutActivity : AppCompatActivity(), BookingAdapter.OnBookingActionListener ⚠2
    private fun loadUserBookings() {
        val userId = auth.currentUser?.uid ?: return
        dbRef.orderByChild( path: "userId").equalTo(userId)
            .addListenerForSingleValueEvent(object : ValueEventListener {
                override fun onDataChange(snapshot: DataSnapshot) {
                    bookings.clear()
                    for (child in snapshot.children) {
                        val booking = child.getValue(Booking::class.java)
                        if (booking != null && (booking.status == "approved" || booking.status =
                            bookings.add(booking)
                        }
                    }
                    adapter.notifyDataSetChanged()
                }

                override fun onCancelled(error: DatabaseError) {
                    Toast.makeText( context: this@CheckInOutActivity, text: "Failed to load booking
                }
            })
    }

    override fun onCheckIn(booking: Booking) {
        showConfirmationDialog( message: "Check In to ${booking.itemName}?", action: "Check In") {
            dbRef.child(booking.id).child( pathString: "status").setValue("checked_in")
```

Figure 3.4.9.1 Load approved booking from database

```
class HistoryActivity : AppCompatActivity() {
    }

    private fun loadHistory() {
        val userId = auth.currentUser?.uid ?: return
        historyRef.orderByChild( path: "userId").equalTo(userId)
            .addListenerForSingleValueEvent(object : ValueEventListener {
                override fun onDataChange(snapshot: DataSnapshot) {
                    val tempList = mutableListOf<Booking>()
                    for (child in snapshot.children) {
                        val record = child.getValue(Booking::class.java)
                        if (record != null) {
                            tempList.add(record)
                        }
                    }
                    // ✅ Adapter handles sorting internally
                    adapter.updateData(tempList)
                }

                override fun onCancelled(error: DatabaseError) {
                    Toast.makeText( context: this@HistoryActivity,  text: "Failed to load history",
                }
            })
    }
```

Figure 3.4.9.2 Load all history data in database

3.4.10 Report Issue Module for user

This module consists of a form to let user filled in to choose a room from database, and choose from spinner option a reason for the issue. If the reason not in the list user can choose others and typed in manually. Then when the user submit the form, the form will save into the issue table for the use of admin. Figure 3.4.10.1 shows the code that system request user input for the issue report.

```
class ReportIssueActivity : AppCompatActivity() {
    private fun submitIssue() {
        val roomName = spinnerRooms.selectedItem?.toString() ?: return
        var issueType = spinnerIssues.selectedItem?.toString() ?: return
        val customText = edtOtherIssue.text.toString().trim()

        if (issueType == "Other" && customText.isNotEmpty()) {
            issueType = customText
        }

        val issueId = db.child( pathString: "issues").push().key ?: return
        val userId = auth.currentUser?.uid ?: "unknown"
        val sdf = SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss", Locale.getDefault())

        val issue = Issue(
            id = issueId,
            roomId = roomName,
            roomName = roomName,
            issueType = issueType,
            description = customText,
            status = "unresolved",
            reportedBy = userId,
            reportedAt = sdf.format(Date())
```

Figure 3.4.10.1 Submit issue form

3.5 AR indoor navigation module using Unity for user

3.5.1 Nav mesh surface floor plan

This module allow user to move in a given nav mesh surface so that the navigation works. It starts with a floor plan needs to be found on website representing Block N Utar ground and first floor plan as shown in figure 3. 5.1.1 and 3.5.2.2. Floor plan for library is not founded, extensive testing for measurement needed. Then after creating the 3D model for the environment, nav mesh surface baked is implemented to create a nav mesh surface for user to move in this virtual plane created as shown in figure 3.5.1.3 for the nav mesh plane created for Block N first floor, for the ground floor although the floor plan are similar but there is a need to test for the measurement again for correct size of nav mesh surface created. For floor plan of library in UTAR which is not founded online, a lot of testing needed to test out the measurement of floor plan without a floor plan This module requires extensive amount of testing for all because the measurement for the floor plan is not given, manually testing for the exact location is needed, to ensure that the distance is correct in order for user to navigate to the correct location.

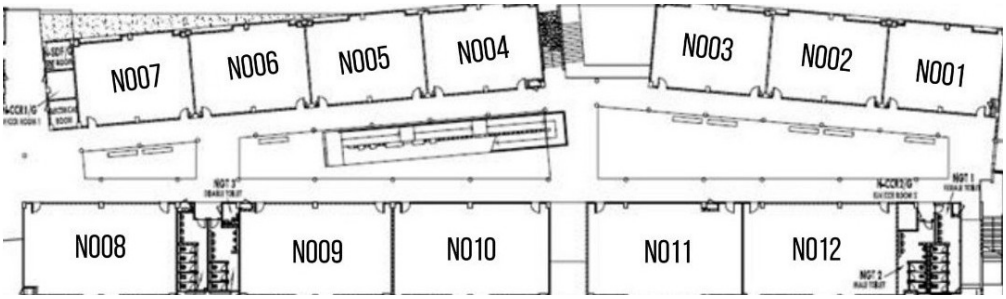


Figure 3.5.1.1 Block N ground floor

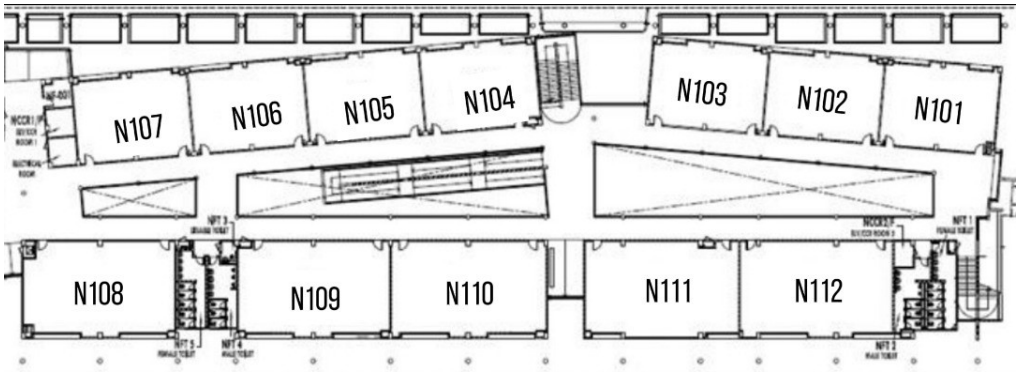


Figure 3.5.1.2 Block N first floor

Figure 3.5.1.3 Block N first floor nav mesh plane

3.5.2 Target and Navigation Module

This module have with additional script for controlling dropdown list for users to choose start location and also target location as well as choosing another floor as shown in figure 3.5.2.1 with a monoscript MultiTarget1st.cs or MultiTargetNavigation.cs to calculate the distance from user current location to target location and place the 3d arrow model along the path for navigation, one of this code snippet is shown in figure 3.5.2.2 and it shows how the code draw path and putting the virtual 3d arrow model along the path. These 2 scripts are basically the same, but the starting point location is different, a repeated 3d arrow path will show after

user chooses a start location and then a target location. For figure 3.5.2.3 shows that how the voice guidance is developed, it will calculate the angle of the turning arrow and play the turning sound.

Figure 3.5.2.1 Canva development for all the dropdown list

```
MultiTargetN1st.cs
Assets > Scripts > MultiTargetN1st.cs
153   #endif
193       void DrawPath()
194       {
195           if (path.corners.Length < 2)
196           {
197               Debug.LogWarning("Path has fewer than 2 corners!");
198               return;
199           }
200
201           for (int i = 0; i < path.corners.Length - 1; i++)
202           {
203               PlaceArrowsAlongSegment(path.corners[i], path.corners[i + 1]);
204           }
205       }
206
207       void PlaceArrowsAlongSegment(Vector3 start, Vector3 end)
208       {
209           Vector3 direction = (end - start).normalized;
210           float distance = Vector3.Distance(start, end);
211           int arrowCount = Mathf.Max(1, Mathf.FloorToInt(distance / arrowSpacing));
212
213           for (int i = 0; i <= arrowCount; i++)
214           {
215               Vector3 position = Vector3.Lerp(start, end, i / (float)arrowCount);
216               Quaternion rotation = Quaternion.LookRotation(direction) * Quaternion.Eu
217               GameObject arrow = Instantiate(arrowPrefab, position, rotation);
218               activeArrows.Add(arrow);
219           }
```

Figure 3.5.2.2 MultiTargetN1st.cs for drawing path and place arrow

```
public class MultiTargetLibrary : MonoBehaviour
    void CheckForUpcomingTurn()
    {
        if (turnPoints.Count == 0) return;

        Vector3 playerPos = ProjectPointToFloor(cameraTransform.position);

        for (int i = 0; i < turnPoints.Count; i++)
        {
            float dist = Vector3.Distance(playerPos, turnPoints[i].position);

            if (dist < turnTriggerDistance)
            {
                if (turnPoints[i].isLeft)
                {
                    PlayTurnSound(turnRightClip);
                }
                else
                {
                    PlayTurnSound(turnLeftClip);
                }
                //  🔹 After turn, play "Go straight" once
            if (!goStraightPlayed && goStraightClip != null)
            {
                StartCoroutine(PlayGoStraightWithDelay(4f)); // 4 second delay
            }
```

Figure 3.5.2.3 Check for upcoming turn and go straight

3.5.3 Restart Module

This module allows user to click on the restart button to restart the unity application. Sometimes the arrow display is not stable so the user can restart the whole application and display the arrow again to prevent unstable arrow. Figure 3.5.3.1 shows how to handle the application restart using the restart button.

```
public class RestartButtonHandler : MonoBehaviour
    void Start()
        }
    }

    1 reference
    void CoordinateRestart()
    {
        // Reset the AR session to clear tracking data
        if (arSession != null)
        {
            arSession.Reset();
            Debug.Log("AR session reset to clear tracking data.");
        }

        // Reload the current scene
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        Debug.Log("Scene reloaded for app restart.");

        // Schedule the position reset for the next frame after scene load
        SceneManager.sceneLoaded += OnSceneLoaded;
    }
```

Figure 3.5.3.1 Restart Button handler

3.5.4 Back to Infinite Booking Module

For this module allow user to back to the Infinite Book android application from this unity project via the back button as shown in Figure 3.5.4.1. This project is then exported as a unity project library and imported into android studio project. A Arnavigation class is developed to navigate to this unity project as shown in Figure 3.5.4.1. Dependency should be added to build gradle in app layer to include the unity imported library, implementation(project(":unityLibrary")). All the folders must be imported into this android studio project including UnityLibrary, Launcher, and a Shared folders for proper working of this unity project.

```
using UnityEngine;
using UnityEngine.UI;

0 references
public class BackButtonHandler : MonoBehaviour
{
    2 references
    public Button backButton; // Assign this in the Unity Inspector

    0 references
    void Start()
    {
        if (backButton != null)
        {
            backButton.onClick.AddListener(() => Application.Quit());
        }
        else
        {
            Debug.LogError("Back Button not assigned in Inspector!");
        }
    }
}
```

Figure 3.5.4.1 Back to Infinite Book android application back button handler

ARNavigationActivity.kt

```kotlin
package com.example.fyp

import android.content.Intent
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.unity3d.player.UnityPlayerGameActivity

class ARNavigationActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Start Unity
        val intent = Intent( packageContext: this, UnityPlayerGameActivity::class.java)
        startActivity(intent)
    }

}
```

Figure 3.5.4.1 ArNavigationActivity.kt for starting unity project

CHAPTER 4

Methodology and Tools

4.1 Overview

This chapter includes 4.1 System Design consists of system diagram and hardware and software used and 4.2 System Methodology to explain about the methodology used which is RAD (Rapid Application Development).

4.2 Methodology

This section details the system methodology used for the development of the University Smart Assistant application. The Rapid Application Development (RAD) methodology was chosen due to its iterative nature for every phases and focus on user involvement to test the syatem. RAD allows for the rapid development of prototypes and emphasizes user feedback throughout the development process. So testing is done thoroughly. The methodology can be broken down into the following key phases.

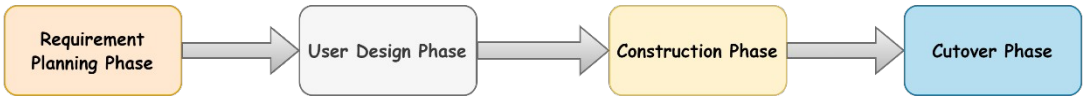


Figure 4.2.1 System methodology

4.2.1 Requirement Planning Phase

The Requirements Planning phase serves as the first important step of the RAD approach, where system requirements are defined. In this phase, literature reviews are done to study on 3 past applications, Ganttic, Cozycal and Zoho Calendar. Then, the scope and objectives of the University Smart Assistant application are discovered and listing out. The main reason of these objectives and scopes are to find out what is the important functionalities that needed for the system, such as the Smart Scheduling Assistant for optimal booking management, AR

Navigation for guiding users to reserved locations, and an intuitive user interface to enhance the overall user experience.

This phase also listing out all the problems had been met, for example like scheduling conflicts, and difficulties in locating reserved rooms. The output of this phase was a clear and detail set of system requirements that used as blueprint to build this project.

4.2.2 User Design Phase

The User Design phase focus on rapid designing and iterative which is continuous user feedback to refine or change the system's design. During this phase, University Smart Assistant icon and logo design were developed using tools such as Canva, allow it to show out the system's user interface and core functionalities. This phase is iterative means that testing is carried out, with frequent cycles of feedback and adjustments to ensure the application design are acceptable.

Smart Scheduling Assistant, AR Navigation module, and other interface elements such as the booking calendar and notification system are needed to design. User feedback must be conduct continuously, user will find out what design is suitable and comfortable for them, allow to reconstruct early to solve the requirement problems.

4.2.3 Construction Phase

The Construction phase uses the design in user design phase to generate out the result of te system. In RAD, this phase is focusing on coding and convert idea and design into working application. The development of the University Smart Assistant was carried out using method such as rapid coding practices and agile methodologies,

Key features, such as the Smart Scheduling Assistant and AR Navigation, were developed using appropriate technologies like Android Studio for mobile application development. User feedback continued to play a very important role during this phase, need to do regular testing sessions that allowed users to interact with working versions of the mobile application. This approach ensured that the development remained align with objectives, resulting in a more effective final product.

4.2.4 Cutover Phase

The Cutover phase is the final stage in the RAD methodology, where the system is already a workable product or application. This phase includes activities such as complete all the documentation, with comprehensive testing, and system deployment. For the University Smart Assistant, extensive testing was conducted to ensure that all system components are functioned well, with focus on the performance of the scheduling conflict and the accuracy of the AR navigation module.

4.3 Tools to use

I. Hardware

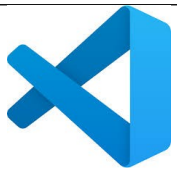
Table 4.3.1 Hardware components and requirements for mobile applications development

Description	**Specifications**
Model	ASUS TUF Gaming F15
Processor	11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz 2.69 GHz
Operating System	Windows 10 Home – 64-bit
Graphic	NVIDIA® GeForce RTX3050
Memory	16GB RAM(random access memory)
Storage	475GB M.2 NVMe™ PCIe® 3.0 SSD

II. Software

Table 4.3.2 Software requirements for mobile applications development

Category	Tools/Platforms	Description
Development Platform	Android Studio	Used as integrated development environment (IDE) for Android app development. It provides tools for designing, coding, testing, and debugging Android applications through Kotlin, and XML
Main Programming Languages	Kotlin and xml	Used with Android Studio for logic and user interface
Augmented Reality (AR)	ARCore (Android)	Platforms for building AR experiences on mobile devices.
Unity with AR Foundation	AR module development integrating ARCore under a unified API.	
IDEs and Editors	Visual Studio Code	A code editor supporting various programming languages and

		frameworks, with plugins for Kotlin and xml
Icon Designing	Canva, Flaticon  	Used for design app logo, icons for app
Database	Firebase  	Used for database for storing user credentials and all of the booking details.

4.4 Timeline

4.4.1 Overview

The timeline of the project is planned follow the methodology of the project. First of all, for requirement planning phase, proposal is written and with problem statement and objective of the project is then be determined, Project scope for all the modules are set to achieve the obejctive. In literature review, a few apps are reviewed. Timeline is generated to ensure that the project is done on time. All the hardware and software tools are chosen for me to works with

For the user design and construction phase, the basic login management module is developed first, which is essential for every app to let user or admin sign up, login and change password. The self-learning is made to learn how to implement firebase as database for this mobile app. Furthermore, the main module for bookings and manage bookings are developed which allow users and administrators to book and view for resources available like books and rooms. For admin, the module for admin to approve, reject the bookings made and also

rooms and books management module is developed. Then a few week is used to develop the unity project and integration unity exported project to android studio booking mobile app. A Final Year Project 1 report is produced.

In later Final Year Project 2, the Indoor AR Navigation module for different block and Check in modules will be then developed and integrated into the mobile app. In addition, a comprehensive testing when the project is done will be carried out to check the project whether it is working fine and also prevent error occur in both Android studio and unity project. Lastly, the Final Year Project 2 Report will be prepared and presented to the supervisor and moderator.

4.4.2 Gantt Chart

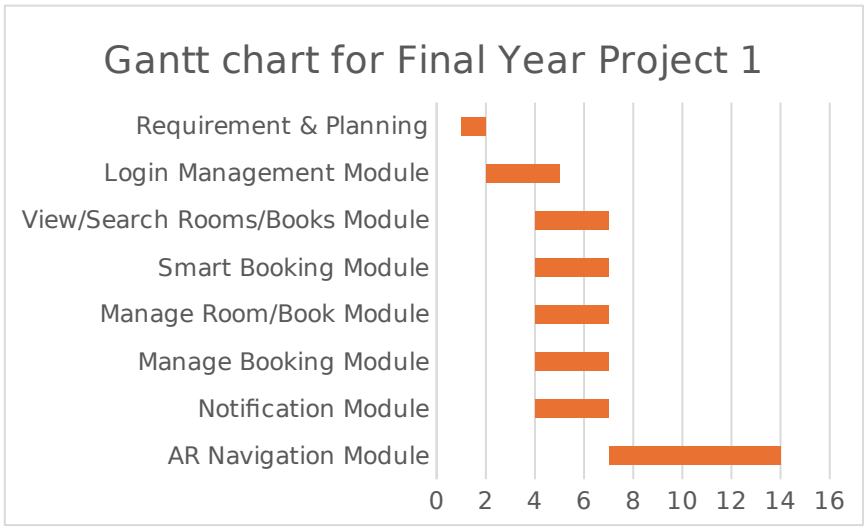


Figure 4.4.2.1 Gantt Chart for fyp1

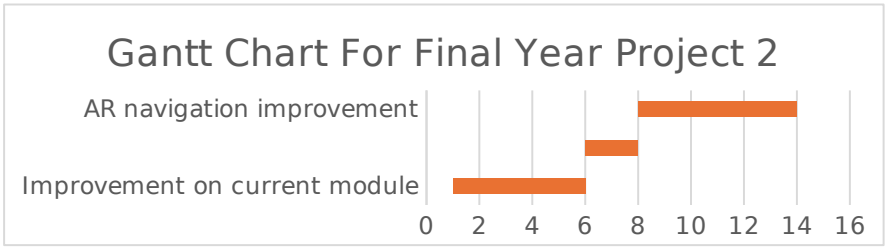


Figure 4.4.2.2 Gantt Chart for fyp2

4.5 Summary

In a short conclusion, RAD methodology had greatly helped in development for this University Smart Assistant application. As shown above there would be a lot of modules and functions, and the displayed is just parts of it, there are a lot more. Hence, it is important to

test at every stage of the development to ensure that each part of the process will not be having any issue. If the testing is done after the development, there will be a lot of additional works to be done. The hardware and software are chosen carefully to be used to develop the backend database and front-end user interface. User case and timeline is drawn to shows all the functionality and the time taken to complete for each functionality. Gantt chart is used for timeline to shows a clearer image on how the development phase is arranged

CHAPTER 5

IMPLEMENTATION AND TESTING

5.1 Overview

For Final Year Project 2, all of the modules are being developed, including User authentication and profile, calendar and booking, notification and reminder, resources management, view booking details, search and filer, smart scheduling, check in out, view overdue, view history, report and view issues and AR navigation. All the expected modules are developed completely The application name for this University Smart Assistant is Infinite Book. It will be a splash screen before login activity. The overall design is mainly shallow colour such as blue, grey, white. There would be a top navigation theme bar which a drawer menu exists for user or admin to access and navigate to some of the module. A bottom navigation bar is also available for user and admin to navigate between home, notification and profile.

5.2 User authentication and Profile management

When user open the application in their mobile phone, a splash screen will show for the app and a login activity page is shown. User or admin will allow to login with their email and password. If new user, an option which is register account can be chosen to register a new account. If user forget about his password and use the forget password function to send a email to user's email. Users need to check his email for the reset password link send by firebase, then the password is reset. A remember me function is used to save the user preference for email and password where user can login without typing the password again. All the user data would be saved to firebase database.

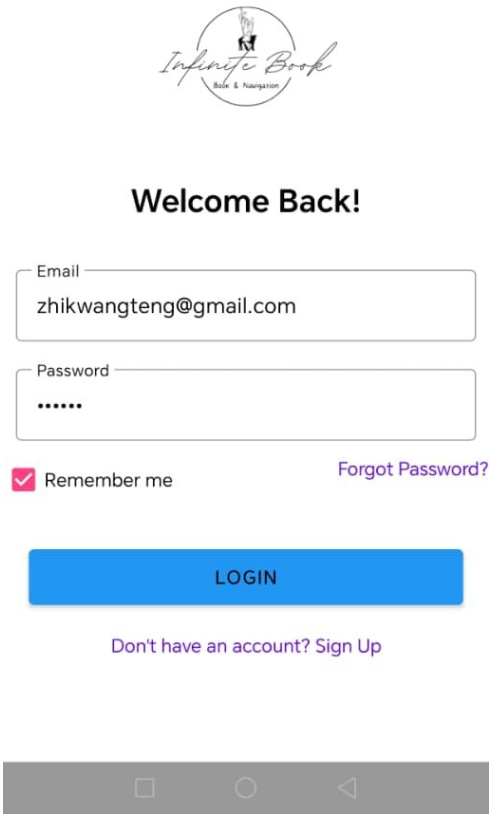


Figure 5.2.1 Login activity

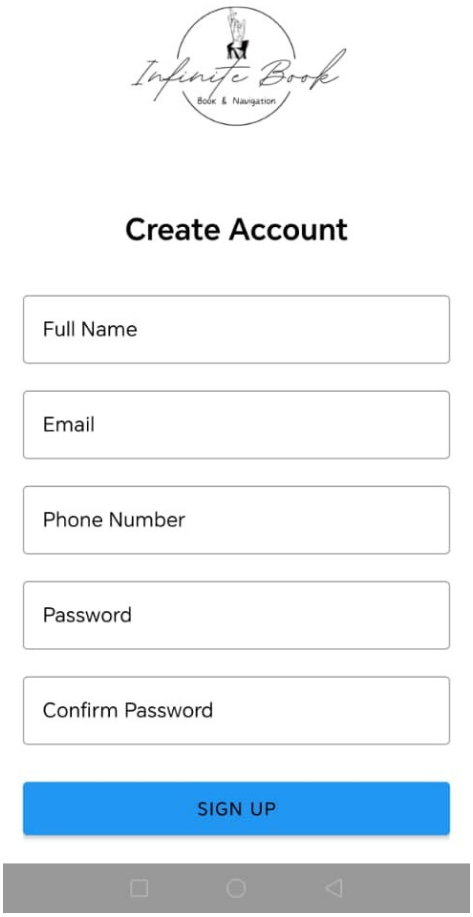


Figure 5.2.2 Sign up activity

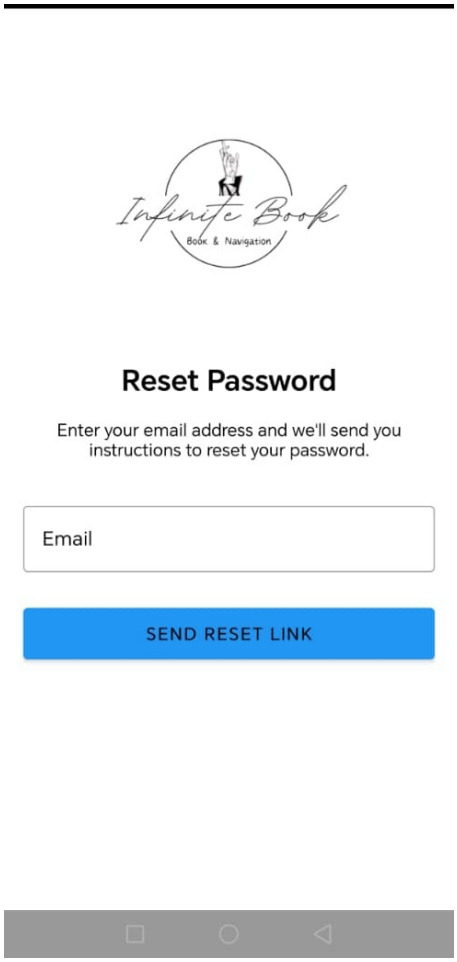


Figure 5.2.3 Reset password page

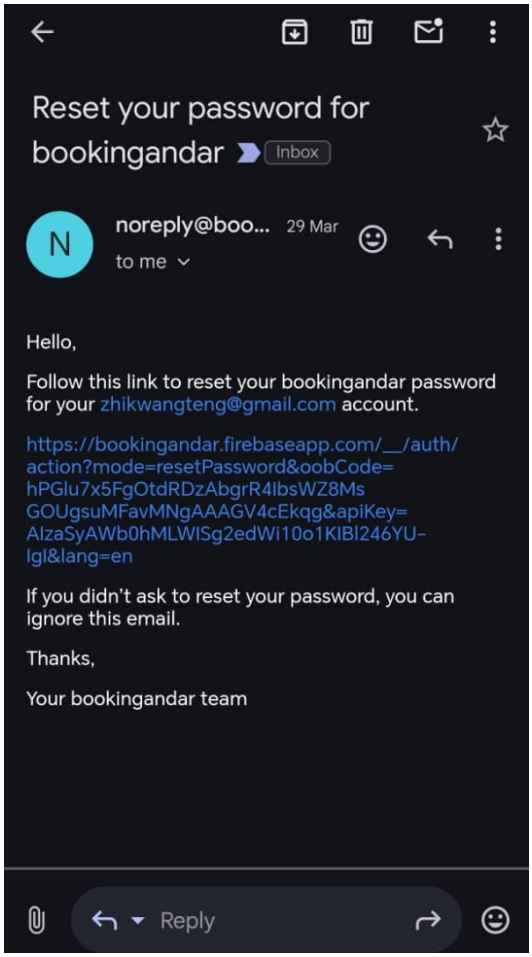


Figure 5.2.4 Reset password email link

5.3 Search and filter

Search and filter module can be entered by click on the book rooms or book books button in the main menu page, there would be a list of rooms or books displayed out for user to book, user can view the details of the rooms such as capacity, name and description and books such as name, author, isbn, category. User can search for a keyword for rooms name or books name to filter out the particular room or book. User can use the filter column to filter out available now rooms in room module, on the other hand user can choose the filter for category of the book, whether user need to find only fiction, non-fiction or textbooks. This search and filter module helps user to easily find the required resources in short period of time effectively.

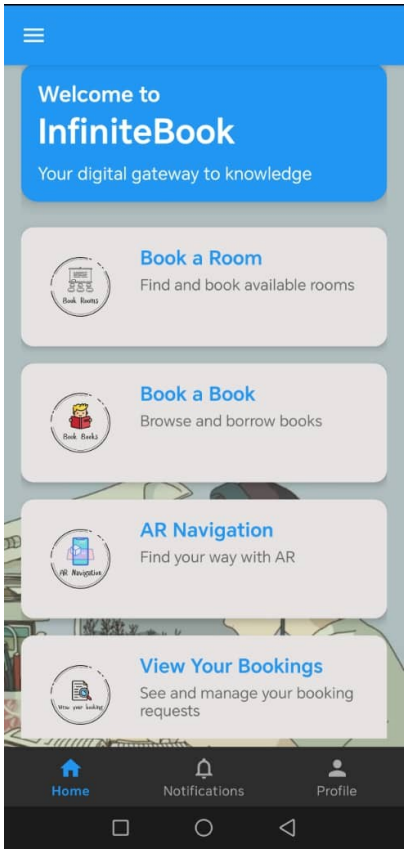


Figure 5.3.1 Home page

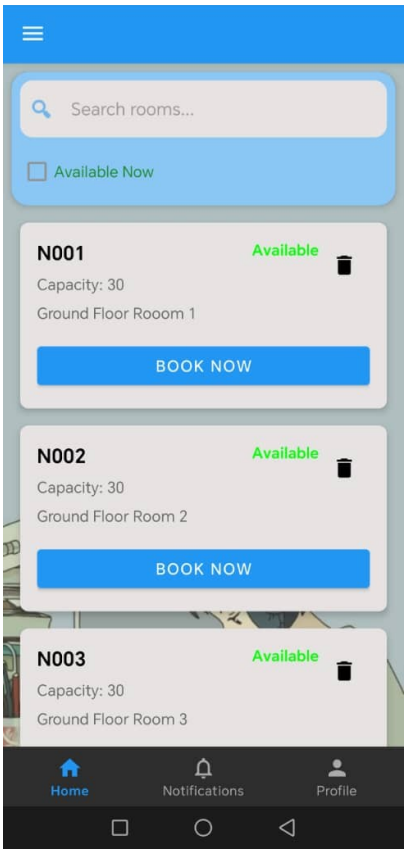


Figure 5.3.2 Search a Room with filter

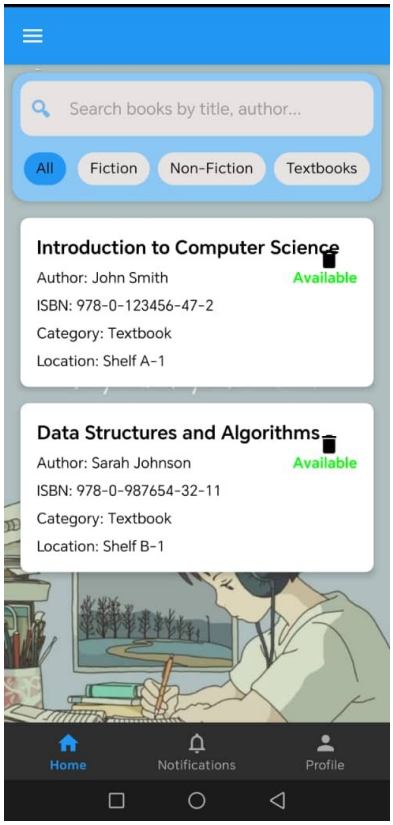


Figure 5.3.3 Search and filter for books

5.4 Calendar and booking

After the user view for rooms and books, when user need to book, user can click on the card view for the room or book and the calendar and booking module will show. Item name and item type which either be books or rooms will show. Different interface is used to request different input for books and rooms. First, for both room and book, calendar will show for user to choose for the start date of booking. For booking rooms, user needs to choose start time and end time for booking the selected room. For Books user needs to select a start and end date for booking. This module is combined with smart scheduling module to determine for booking conflict. If booking conflict occur, user will be not allowed to book for the resources. There is a view booked time slots for user to view for the booked time slots by own or others by the selected date.

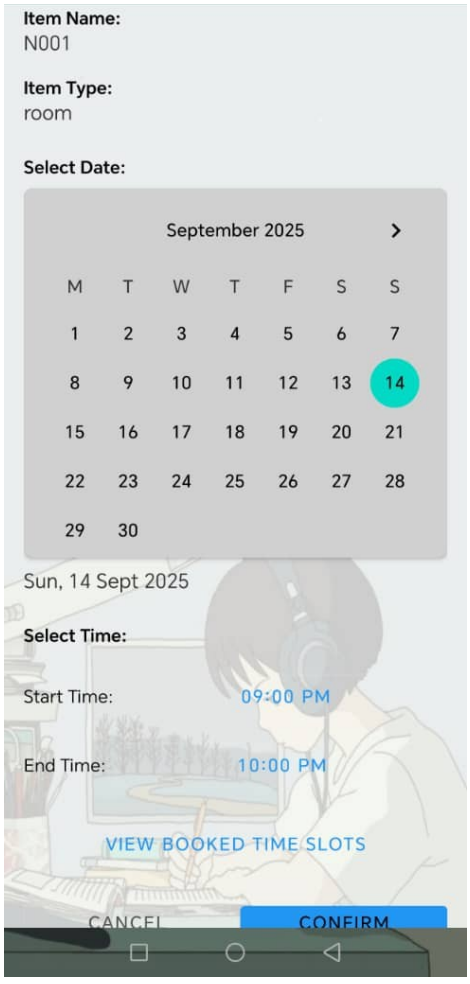


Figure 5.4.1 Scroll view for booking rooms

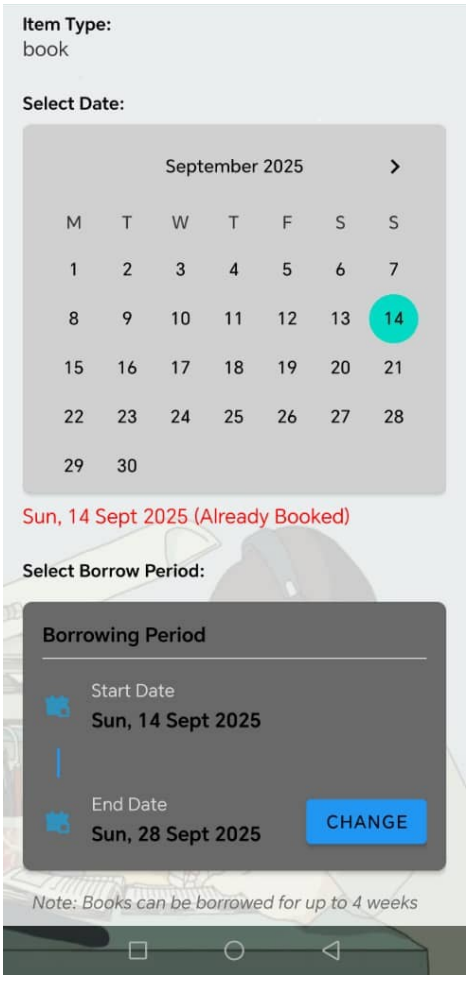


Figure 5.4.2 Scroll view for booking books

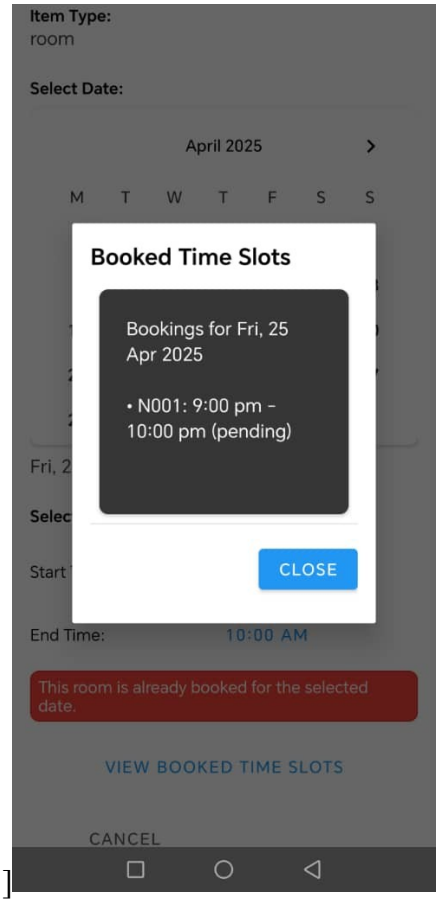


Figure 5.4.3 View Booked time slot

5.5 Smart Scheduling

This module is considering the most complicated module which is contains extensive amount of logic to handle booking conflicts for books and rooms. Since the input for booking details for rooms and books are different, different logic must be applied to handle these resources. For rooms, if the user selects a time slot that overlaps with an existing booking on the chosen date, the app block the user to select the date and prompt a error text message telling user that user cannot choose which is conflict by overlapping others booking as the system compares the selected start and end times with existing bookings for that room which is on the same day. The message is clear and appears instantly to guide the user to pick a different time slot.

For books, the conflict check is slightly different because books are booked for a range of dates. If the user's selected date range overlaps with any existing booking for that book, the app displays a prompt text with a message like, "This book is already booked for the selected dates. " and the date range booked will be shown to user when the system checks if the

chosen start and end dates conflict with any approved or pending bookings in the database. The simple wording helps the user understand they need to adjust their date range to avoid the conflict.

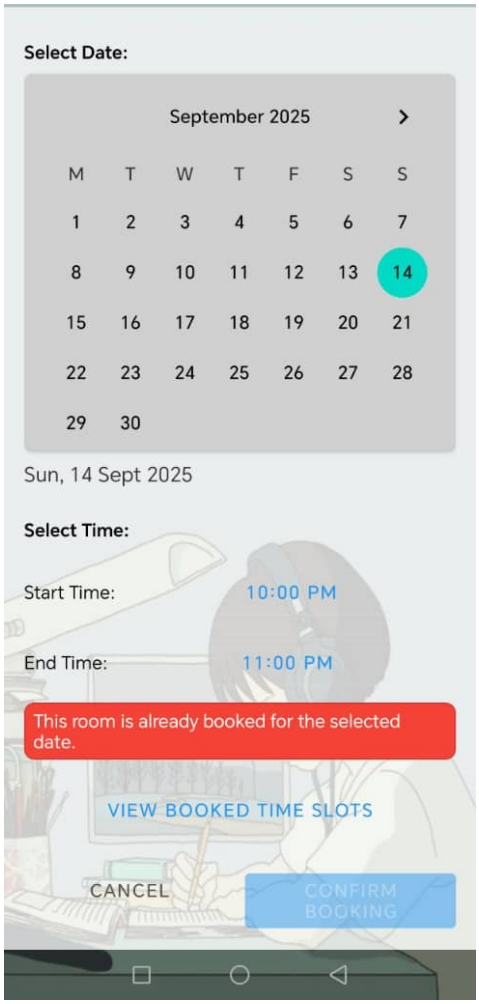


Figure 5.5.1 Same date time conflict by room

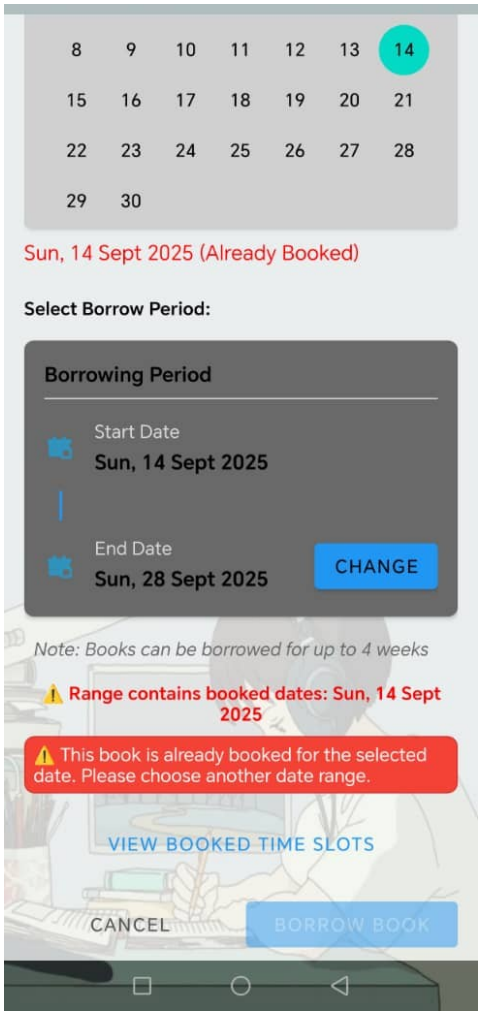


Figure 5.5.3 Chosen date range conflict

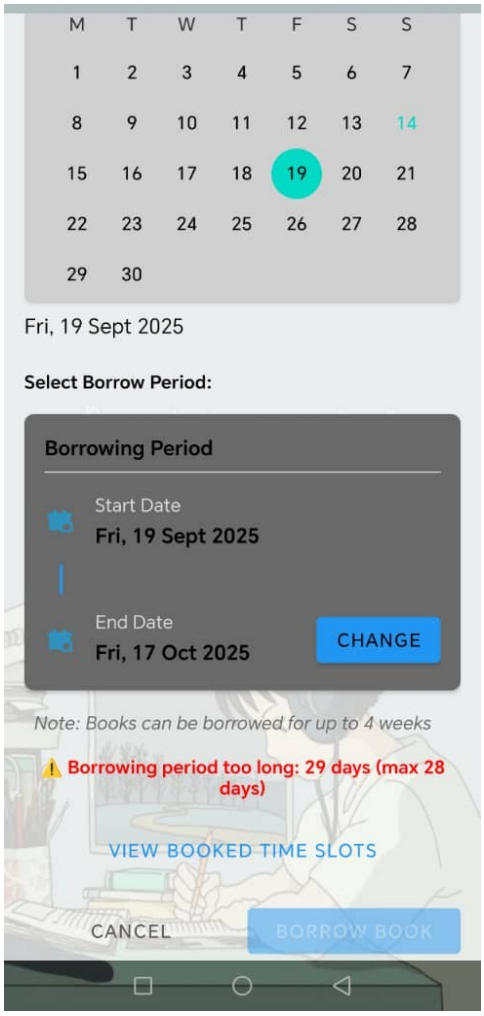


Figure 5.5.4 Booking period too long

5.6 Resources Management

This module allow admin to manage the details for books and rooms as well as bookings. For books and rooms, admin can navigate by clicking the manage button in the admin dashboard, admin will see two buttons which are for manage rooms, books and also booking. For rooms and books, admins are able to change the information or status of rooms or books. Admins could delete the current rooms or books by clicking the delete rubbish bin icon. Admin can also add a new rooms or books by filling the details of the rooms and books. For bookings, admins are able to view all the booking requests and approve or reject on current booking requests made by user.

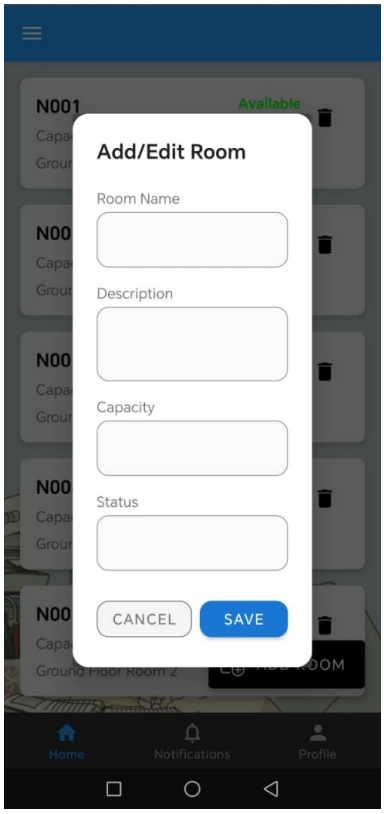


Figure 5.6.1 Add new room

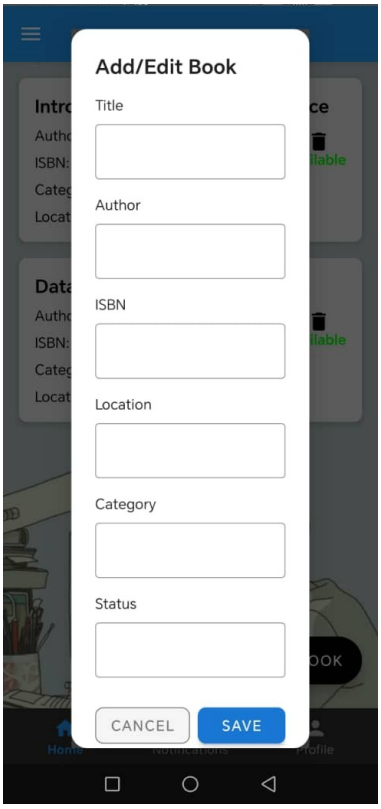


Figure 5.6.2 Add new Book

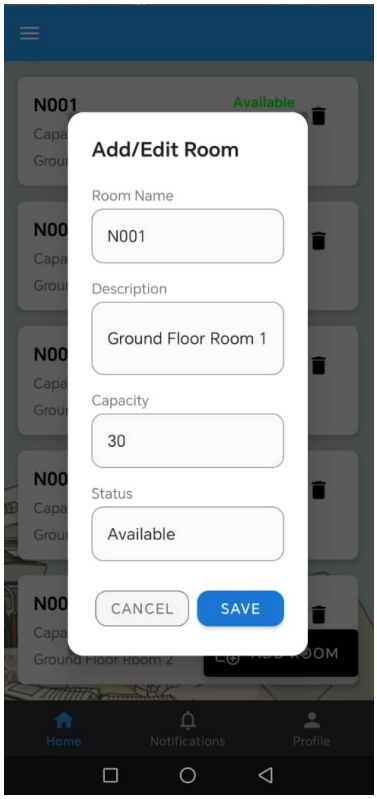


Figure 5.6.3 Edit room

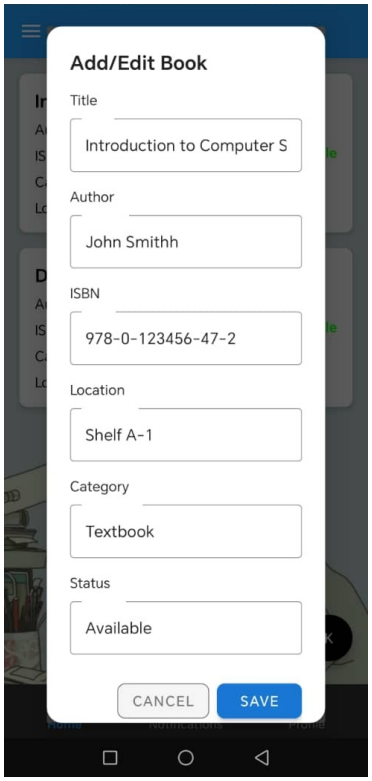


Figure 5.6.4 Edit Book

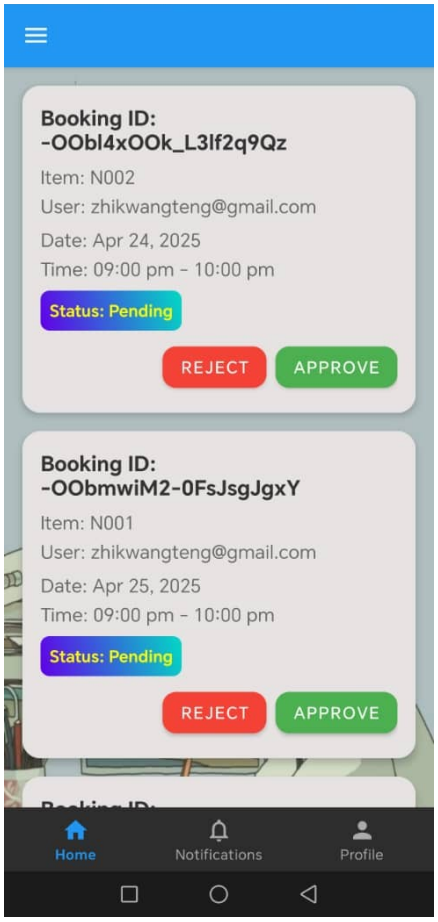


Figure 5.6.5 Manage Booking (Approve /Reject)

5.7 View Booking Details

User can view their own booking details from home page; by clicking on the view bookings button, user can navigate to the view booking details page. Here it fetches all bookings details for the user and display out, showing what user had booked, whether is rooms or books, with the time and date reserved. In this module, user can easily cancel their booking by clicking cancel button. Once, user cancel the booking the status will be updated to be cancelled.

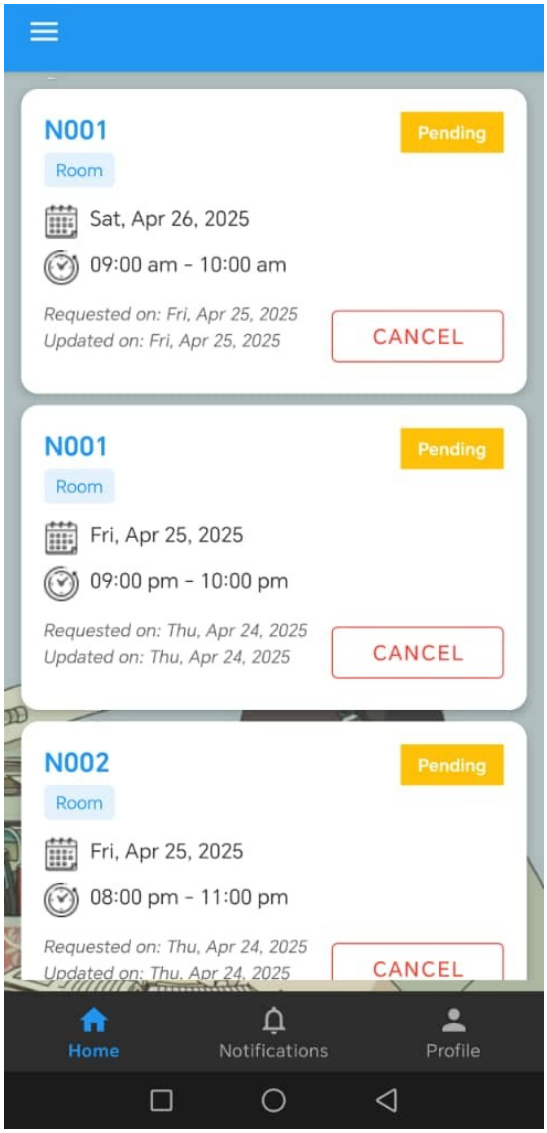


Figure 5.7.1 View booking details

5.8 Notification and reminder

After user booked for the resources, once the admin approved, a notification will send to user, telling him that his booking request is approved, and the start date and time is from which date. User can access to notification page directly from the bottom of the navigation view. After the user check in but not check out the system will send a pop out notification to inform user.

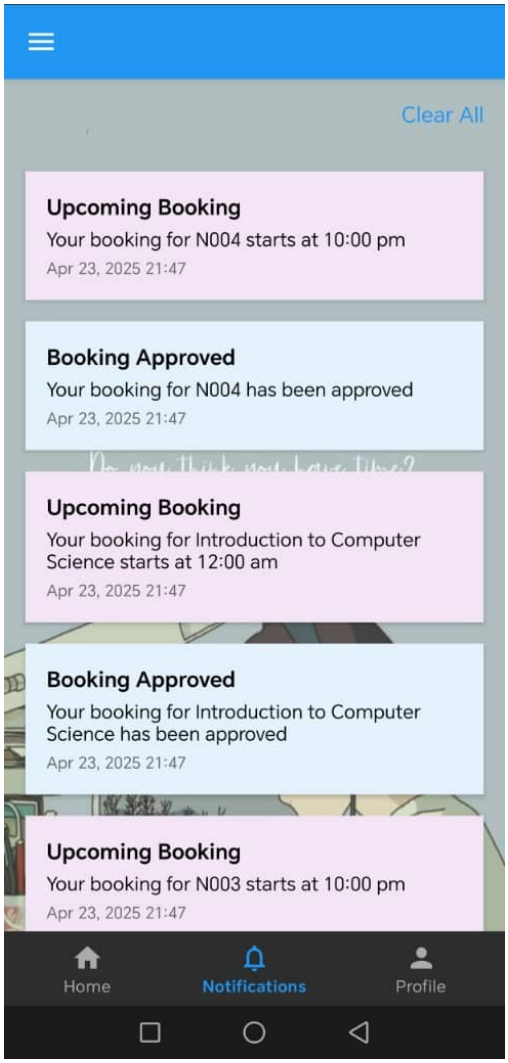


Figure 5.8.1 Notifications

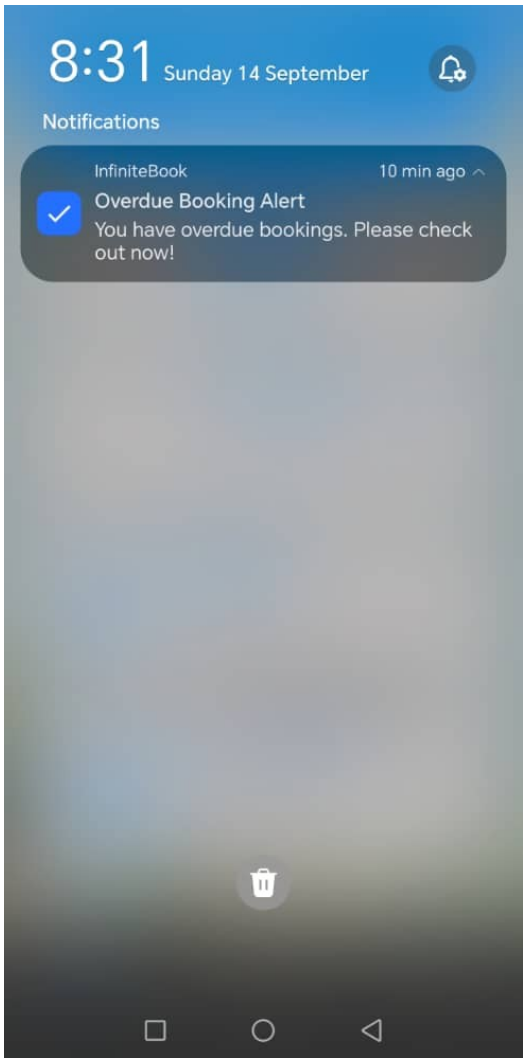


Figure 5.8.2 Overdue Check in

5.9 Statistic Module

In this module all the admin and librarian are able to view the statistic result on the booking details for each room and book. Admin can choose to view for the past month or for the past week for more recent information. Besides, admin can click on the pie chart shown to view thee details on how many percentages each room obtains or more commonly booked by user.

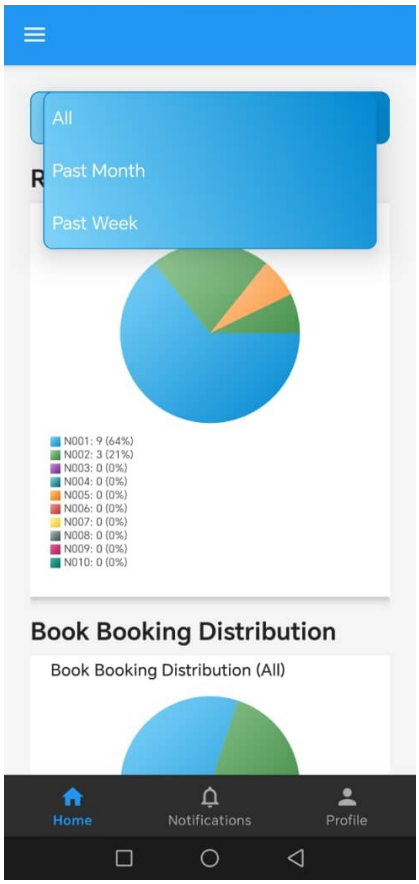


Figure 5.9.1 Statistic and filter

Figure 5.9.2 Details on the booking

5.10 Check in/out Module

This module allows all users to check in and also check out when a bookings is approved by admin. User can enter through dashboard. First users will see all the approved bookings details. When user press the check in or check out button, a confirmation dialog will pop out and request user to confirm the check in or check out. Then the completed booking details can seen on the view history button.

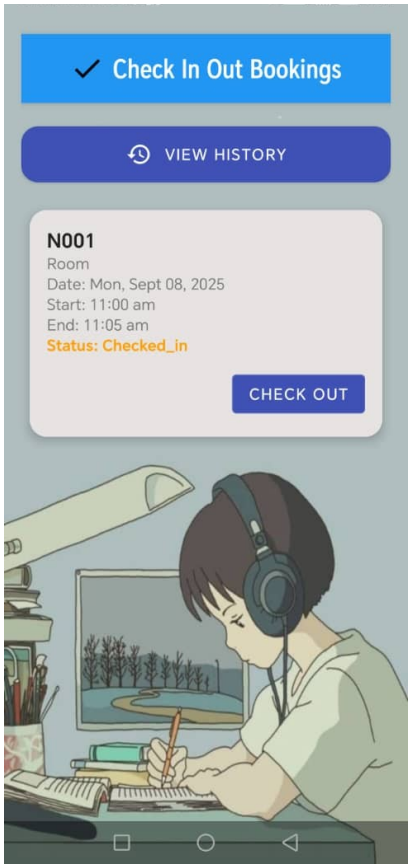


Figure 5.10.1 Check in out

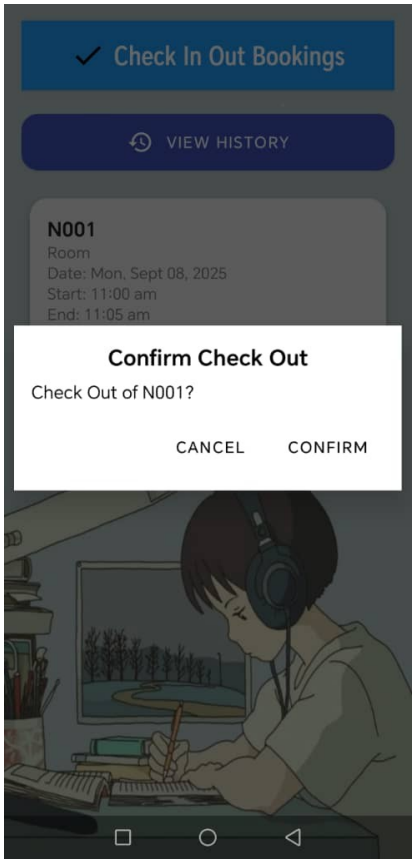


Figure 5.10.2 Confirmation dialog for check in out

5.11 View History Module

This module allow all users to view history for the completed booking. User can see all the completed booking details here. This module is entered from check in out module. User can observe all the history of the booking they made.

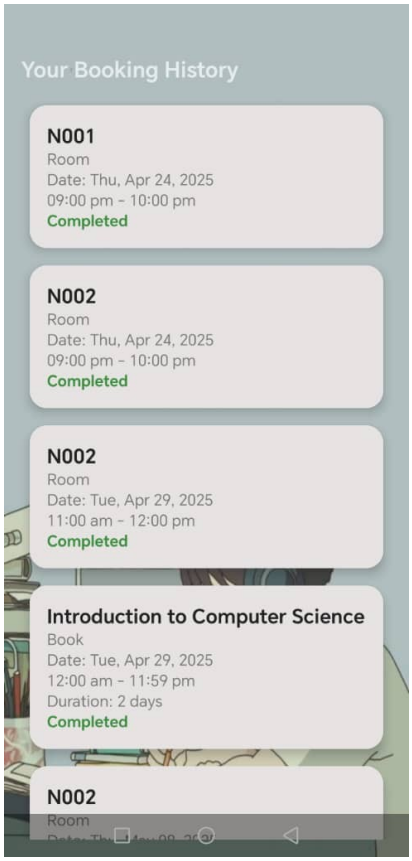


Figure 5.11.1 View Booking History

5.12 Report and View Issue Module

This module allow all users to report for an issue a room has. Users can select rooms first which has issue, then select the reason or the issue, users can submit the form. Admin here can view the issue and confirm the resolve of the issue. A confirmation dialog will show to request the issues is confirm to be resolved.

Figure 5.12.1 Report Issue

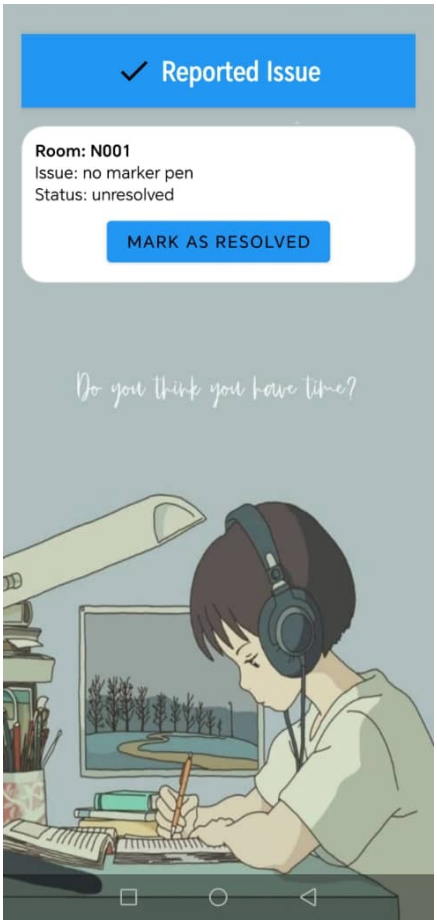


Figure 5.12.2 View Reported Issue

5.13 View Overdue Booking Module

This module allow all admin to view the overdue booking which the user is checked in but not check out, admin can press on the details and system will navigate to phone that allow admin to call the user who had not checked out.

Figure 5.13.1 View Overdue Booking

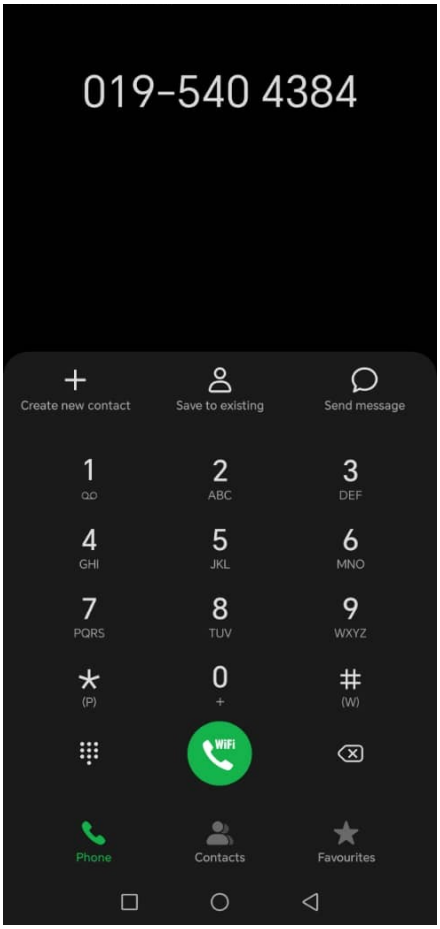


Figure 5.13.2 Phone call direct

5.14 AR navigation

AR navigation module can be accessed by user by clicking the AR navigation button in the main page, user would be able to redirect to the Unity AR indoor navigation app, First user are allowed to choose a location, whether is Block N ground floor or Block N first floor. After choosing the location user must choose a start location, there have 3 start location with provided picture and description for each location. After choosing the start location, user able to start his navigation by choosing the target location which is the room. After choosing the room, user would see an 3d navigation arrow showing the direction to the room that the user booked. Besides that, there would be some information board shows to help user knows where the rooms is located at. There is also a sound guidance when turn at corner, a TURN LEFT or TURN RIGHT sound will play to inform user to turn. Beside, a GO STRAIGHT and DESTINATION ARRIVED sound will also played when after a turn and arrive at destination.

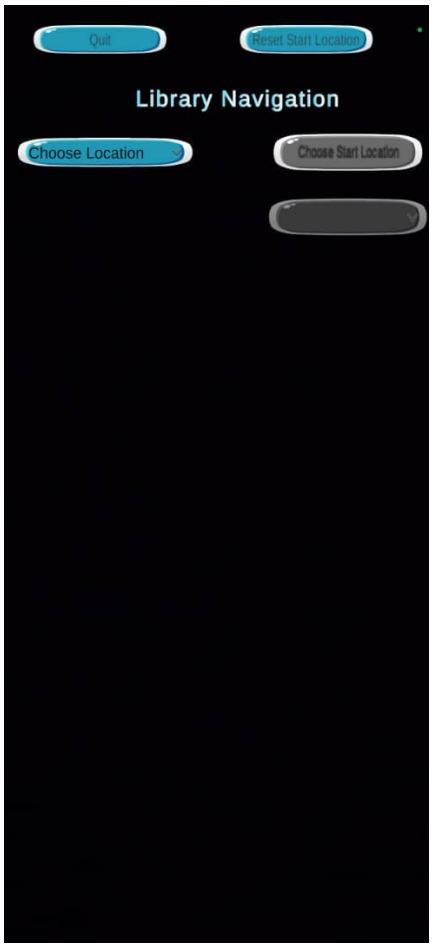


Figure 5.14.1 Main Screen

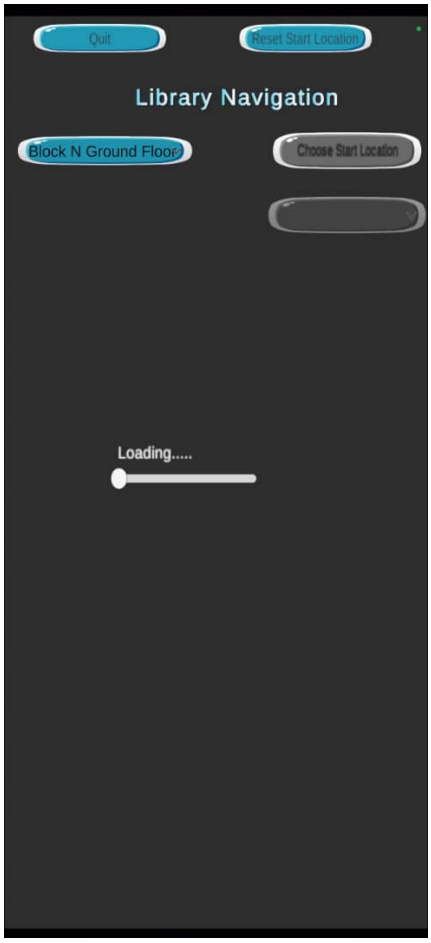


Figure 5.14.2 Loading Screen when change location

Figure 5.14.3 Choose Start Location

Figure 5.14.4 Choosing target location with arrow and info board

5.15 Testing

For testing phase I choose the combination of unit testing, integration testing, system testing as a technique to test all my system component. All of the system features such as authentication, booking with smart scheduling, notifications, resource management, check-in/out, history, issue reporting, and AR navigation was having their own test cases. These test cases are created and under test, there are steps, the expected output, and the actual result are all observed during execution. Testing was performed on Honor Android device connected to the Firebase backend and Unity AR module.

The table below shows the test cases in table test design form for testing phase.

Test Case ID	Module	Test Description	Steps	Expected Result	Actual Result	Pass /Fail

TC-01	User Authentication	Login with valid credentials	Enter registered email & password, press "Login"	User entered to Home screen	User entered to Home screen	Pass
TC-02	User Authentication	Login with invalid credentials	Enter wrong email/ password, press "Login"	Error message "Invalid credentials" appears, no login	Error message "Invalid credentials" appears, no login	Pass
TC-03	User Authentication	Password reset	Click "Forgot Password", enter email	Password reset email sent to user's inbox	Password reset email sent to user's inbox	Pass
TC-04	Profile Management	Remember me function	Tick "Remember Me" during login	Next time app opens, credentials prefilled	Next time app opens, credentials prefilled	Pass
TC-05	Search & Filter	Search rooms	Enter keyword "N" in room search bar	Only rooms containing "N" displayed	Only rooms containing "N" displayed	Pass
TC-06	Search & Filter	Filter available rooms	Select "Available Now" filter	Only currently available rooms displayed	Only currently available rooms displayed	Pass
TC-07	Calendar & Booking	Book a room	Select Room A, date/time, press "Book"	Booking saved, confirmation shown	Booking saved, confirmation shown	Pass

TC-08	Smart Scheduling	Attempt overlapping room booking	Select a slot overlapping existing booking	Error "Time slot already booked" shown, booking blocked	Error "Time slot already booked" shown, booking blocked	Pass
TC-09	Smart Scheduling	Book a book overlapping dates	Select Book B with date range overlapping another booking	Prompt "This book is already booked for selected dates" shown, booking blocked	Prompt "This book is already booked for selected dates" shown, booking blocked	Pass
TC-10	Resource Management (Admin)	Add a new room	Admin enters new room details, press "Save"	Room added and appears in list	Room added and appears in list	Pass
TC-11	Resource Management (Admin)	Delete a book	Admin clicks delete icon on a book	Book removed from list	Book removed from list	Pass
TC-12	View Booking Details	View own bookings	Tap "View Bookings" button	List of user's bookings displayed	List of user's bookings displayed	Pass
TC-13	Notification & Reminder	Receive booking approval notification	Admin approves booking	Notification appears in user's notification fragment	Notification appears in user's notification fragment	Paas
TC-14	Statistics Module (Admin)	View booking statistics	Admin selects "Past Month"	Pie chart of bookings displayed with correct data	Pie chart of bookings displayed with correct data	Pass

TC-15	Check-in/ Check-out	Perform check-in	User selects approved booking, press "Check In"	Status updated to "Checked In"	Status updated to "Checked In"	Pass
TC-16	Check-in/ Check-out	Perform check-out	User selects active booking, press "Check Out"	Status updated to "Completed"	Status updated to "Completed"	Pass
TC-17	View History	View past bookings	Tap "View History"	All completed bookings displayed	All completed bookings displayed	Pass
TC-18	Report & View Issue	Submit room issue	User selects room, fills issue, press "Submit"	Issue saved, confirmation shown	Issue saved, confirmation shown	Pass
TC-19	Report & View Issue (Admin)	View and mark issue resolved	Admin opens "Reported Issues", clicks "Resolve"	Issue marked as resolved, removed from open issues	Issue marked as resolved, removed from open issues	Pass
TC-20	View Overdue Bookings (Admin)	See overdue check-ins	Admin opens "Overdue Bookings"	List of overdue bookings shown with call option	List of overdue bookings shown with call option	Pass
TC-21	AR Navigation	Start AR navigation	User chooses Block N, start location, and destination room	3D arrow appears guiding to the room	3D arrow appears guiding to the room	Pass

TC-22	AR Navigation	Audio guidance at turn	Follow route to a corner	"Turn Left/Right" voice prompt plays at correct moment	"Turn Left/Right" voice prompt plays at correct moment	Pass
TC-23	AR Navigation	Destination reached	Arrive at target room	"Destination Arrived" voice prompt plays, arrow stops	"Destination Arrived" voice prompt plays, arrow stops	Pass

Table 5.15.1 Test Case and Result

5.16 Conclusion

This chapter shows all the full development and testing for the Infinite Book University Smart Assistant application. All modules covering from user authentication and profile management, search and filter, calendar and booking with smart scheduling, resource management, booking details, notifications, statistics, check-in/out with history, issue reporting, overdue booking monitoring, and AR navigation have been successfully developed into one application. The application provides user friendly and consistent interface with easy and clear navigation, allowing users to book resources efficiently while administrators can oversee resources, bookings, and issues. Audio and visual as well as interactive elements such as 3D AR arrows and voice prompts further improve the usability, demonstrating that the system meets its objectives.

.

CHAPTER 6

CONCLUSION

6.1 Project Review, Discussion and Conclusion

A smart booking assistance system is needed especially for a university like UTAR which contain lots amount of student. It is very hard for the staff to control all of the booking details manually. Traditional paper based system should be replaced so that some of the issues such as booking conflict could be avoided. Paper works could also overwhelm the staff and cause human-errors. Another view is that booking system must be accompany with an AR navigation system to guide student to navigate to the target resources room they book, and this is to increase the efficiency and reduce anxiety. So the purpose of this project is to provide a booking conflict free application with AR module that shows the navigation within campus facility which is rooms. According to both objectives which are Development of a Smart Scheduling System for Optimized Room Booking and Utilization and Integration of an Augmented Reality (AR) Navigation Module for Campus Facility Localization.

6.2 Novelties and Contributions

An important advancement in the incorporation of smart scheduling and AR in educational service can be seen in the University Smart Assistant application. The application provides smart campus solutions and improves the overall student experience by solving the problems of resource management and navigation. The use of these technologies ease students and also admin or known as school staff.

This project suggests creating a mobile application with AR for room navigation and a smart scheduling assistant to address all of booking and navigation issues. By examining booking details, the Smart Scheduling Assistant makes prevent most of the conflict when making reservations for rooms and books. The system's intelligence will alert users to possible conflicts so they can avoid overlapping reservations and make well-informed decisions. guaranteeing a more equitable and effective use of university resources. By optimizing the entire booking process, this method not only reduces conflicts but also makes it more efficient and user-friendly.

Additionally, the AR Room Navigation feature is designed to guide students to their booked locations with ease. By using augmented reality, this feature provides real-time, clear arrow directions, helping users navigate through the campus efficiently. By reducing the time and effort required to find booked facilities, this navigation helps to enhance the overall user experience and ensures that students reach their destinations without delay on time. The AR feature can also include images with text description when showing the start location, ease for user to find their start location and start for the navigation

6.3 Future Work

There are still some enhancements can be done to this projects which include a more comprehensive AR navigation. Now the current AR navigation system is only focus on indoor navigation between block N and library. The future enhancement can be made to the whole campus by using a more improved navigation using longitude and latitude.

Besides, more resources can be added to the system for managing. Now the system only manages rooms and books, more types of resources can be added to this system, allowing it to be more comprehensive and more effectively handling of all the university resources. Hopefully it will lessen the administrative burden for university. By adding more resources, this application could be useful for more students who need to book for school resources.

Lastly, for the booking resources module, it can still improve by AI suggestions for a suitable room or other resources. For the current system, the booking module does not give suggestions to the user and it only checks for any booking conflict that happens, so training an AI to give suggestions to the user is considered a good future work that can be added to this application. AI can be trained based on the current data for bookings of users, it will then give suggestions to the user by predicting which room or book or other resources will become peak booking time for the user to book. Hence, it can prevent even more booking conflicts from happening.

REFERENCES

[1] "Possibilities and apprehensions in the landscape of artificial intelligence in education," *IEEE Conference Publication | IEEE Xplore*, Nov. 26, 2021. Available: https://ieeexplore.ieee.org/abstract/document/9697272?casa_token=cnJeRYnqjSEAAAAA:8x26DpoYfrRo7BLqwlcDBLZHQXKO_3JHroe4aq8dm_LwtzVV-joYMNaG-fkTK4ucnjY68BZF

[2] M. L. Owoc, A. Sawicka, and P. Weichbroth, "Artificial intelligence Technologies in Education: Benefits, challenges and Strategies of implementation," in *IFIP advances in information and communication technology*, 2021, pp. 37–58. doi: 10.1007/978-3-030-85001-2_4. Available: https://doi.org/10.1007/978-3-030-85001-2_4

[3] V. A. Dr. V. R. Anand Dr. V. Vijay and K. a. S. Prof. S. T. Swaminathan Prof. R., "Impact of IOT on Campus; Smart Student Information System in the educational sector," *Journal of Informatics Education and Research*, vol. 4, no. 2, May 2024, doi: 10.52783/jier.v4i2.836. Available: https://jier.org/index.php/journal/article/view/836

[4] A. Braaksma, "Timely and efficient planning of treatments through intelligent scheduling," 2015. doi: 10.3990/1.9789036539302. Available: https://research.utwente.nl/en/publications/timely-and-efficient-planning-of-treatments-through-intelligent-s

[5] "Artificial Intelligence meets Augmented Reality," Google Books. Available: https://books.google.com.my/books?hl=en&lr=&id=vb5IEAAAQBAJ&oi=fnd&pg=PT17&dq=This+innovative+application+aims+to+leverage+AI+technologies+and+augmented+reality+(AR)+to+create+a+seamless+and+intuitive+resource+booking+system&ots=h_S0ZmBfW9&sig=L4PsRtC2ngfwBMi-iegYHczwSPE&redir_esc=y#v=onepage&q&f=false

[6] G. Kasalak and M. Dağyar, "University student satisfaction, resource management and metacognitive learning strategies," Teachers and Curriculum, vol. 20, Jul. 2020, doi: 10.15663/tandc.v20i1.343. Available: https://tandc.ac.nz/index.php/tandc/article/view/343

[7] "Augmented reality," Google Books. Available: https://books.google.com.my/books?hl=en&lr=&id=OyGiW2OYI8AC&oi=fnd&pg=PR1&dq=the+incorporation+of+AR+technology+provides+students+with+real-time+navigation+assistance,+guiding+them+to+their+booked+facilities+with+ease+and+precision.&ots=Z0BMX

VMZK4&sig=LxrUAEA4GzGg6GJv_bHoXcYBxPo&redir_esc=y#v=onepage&q&f=false

[8]N. Liu, P. M. Van De Ven, and B. Zhang, “Managing appointment booking under customer choices,” Management Science, vol. 65, no. 9, pp. 4280–4298, Sep. 2019, doi: 10.1287/mnsc.2018.3150. Available: https://doi.org/10.1287/mnsc.2018.3150

[9] “Cloud computing,” *Google Books*. Available: https://books.google.com.my/books?hl=en&lr=&id=mzM53Yp9cpUC&oi=fnd&pg=PT23&dq=By+automating+the+scheduling+process,
+the+application+frees+up+valuable+time+for+both+students+and+administrative+staff&ots=RVSUs0rTEC&sig=PYdwjz5gBAfblDOKEIJjTGyxFuo&redir_esc=y#v=onepage&q&f=false

[10] Mulla, S. Z., Thakur, P. S., and Tambe, T. Z., "Smart Meeting Room Booking System," 2021. Available: https://d1wqtxts1xzle7.cloudfront.net/99827570/smart-meeting-room-booking-system-libre.pdf?1678786277=&response-content-disposition=inline%3B+filename
%3DSmart_Meeting_Room_Booking_System.pdf&Expires=1721628399&Signature=db7UNpRHLE70DpqM4IJU-
28caUrz62jW59qgvHAgx6k0iPcAz1rrll0je9m8CD8qwIOAOh05ZbGxjTIlxTpPs7eawEzr6wR-J7IzPqrKd4styzkQIeM6fOC4-
4lOWm9Tnb7d8f4AcIlxyRaj7uU3F8ZJzt904qfhaN-OjUbg~W-
XiS4cKDqeXlOMk1VUtPhvGH8y4KdJMBMFlZ7FhYesam5kxV4XZAdELp~fXeCtqNKTwYn4Q4uuHruvbFGVtIikaW4FRtFii8jv3k8JPHFxR6rywbCeyL-
7VQ5zAHesRl7P2CS2yMltkk9hDCO30wbodaxsm48H7oreLYzzhI6-hA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

[11] “BOOKAZOR - an online appointment booking system,” *IEEE Conference Publication | IEEE Xplore*, Mar. 01, 2019. Available: https://ieeexplore.ieee.org/abstract/document/8899460

[12] G. Gerstweiler, E. Vonach, and H. Kaufmann, “HyMoTrack: a mobile AR navigation system for complex indoor environments,” *Sensors*, vol. 16, no. 1, p. 17, Dec. 2015, doi: 10.3390/s16010017. Available: https://www.mdpi.com/1424-8220/16/1/17

[13] A. Bogdanovych, H. Berger, S. Simoff, and C. Sierra, “Travel Agents vs. Online Booking: Tackling the Shortcomings of Nowadays Online Tourism Portals,” in Information and Communication Technologies in Tourism 2006, 2007, pp. 418–428.

doi: 10.1007/3-211-32710-x_55. Available: https://doi.org/10.1007/3-211-32710-x_55

[14] R. T. Azuma, "A survey of augmented reality," Presence, vol. 6, no. 4, pp. 355–385, Aug. 1997, doi: 10.1162/pres.1997.6.4.355. Available: https://doi.org/10.1162/pres.1997.6.4.355

[15] J. Carmigniani and B. Furht, "Augmented Reality: An Overview," in Springer eBooks, 2011, pp. 3–46. doi: 10.1007/978-1-4614-0064-6_1. Available: https://doi.org/10.1007/978-1-4614-0064-6_1

[16] C. O. Chung, "Augmented Reality Navigation System on Android - ProQuest." Available: https://www.proquest.com/openview/b81eadc05b1c9ffab5c5b6765da5fa7f/1?pq-origsite=gscholar&cbl=1686344

[17] D. A. Rossit, F. Tohmé, and M. Frutos, "Industry 4.0: Smart scheduling," International Journal of Production Research, vol. 57, no. 12, pp. 3802–3813, Aug. 2018, doi: 10.1080/00207543.2018.1504248. Available: https://doi.org/10.1080/00207543.2018.1504248

[18] "CozyCal Company Profile funding & Investors," *YourStory*. Available: https://yourstory.com/companies/cozycal

[19]"What is CozyCal?" *TrustRadius*. Available: https://www.trustradius.com/products/cozycal/reviews#overview

[20] "How to streamline resource booking?" Available: https://www.cozycal.com/blog/how-to-streamline-resource-booking

[21] "Set up resource booking," Set up Resource Booking. Available: https://support.cozycal.com/set-up-resource-booking

[22] "The 5 best Studio Scheduling Software in 2024." Available: https://www.cozycal.com/blog/the-5-best-studio-scheduling-software

[23] S. Vembu, "Our story | Zoho," *Zoho*. Available: https://www.zoho.com/ourstory.html?ireft=nhome&src=home1-dd

[24] "Resource Booking | Zoho Calendar," *Zoho*. Available: https://www.zoho.com/calendar/help/resource-booking.html

[25] Zoho, "Zoho Bookings: Five out-of-the-box features you'll love in an appointment scheduling software," YouTube. Jun. 22, 2020. Available: https://www.youtube.com/watch?v=xI4d_05yPXY

[26] Clientric, “Zoho Bookings: Syncing Your Calendar [Part 7],” YouTube. Sep. 22, 2022. Available: https://www.youtube.com/watch?v=iYbtCmZoVVo

[27] “What is Zoho Calendar?” Available: https://www.trustradius.com/products/zoho-calendar/reviews#product-details

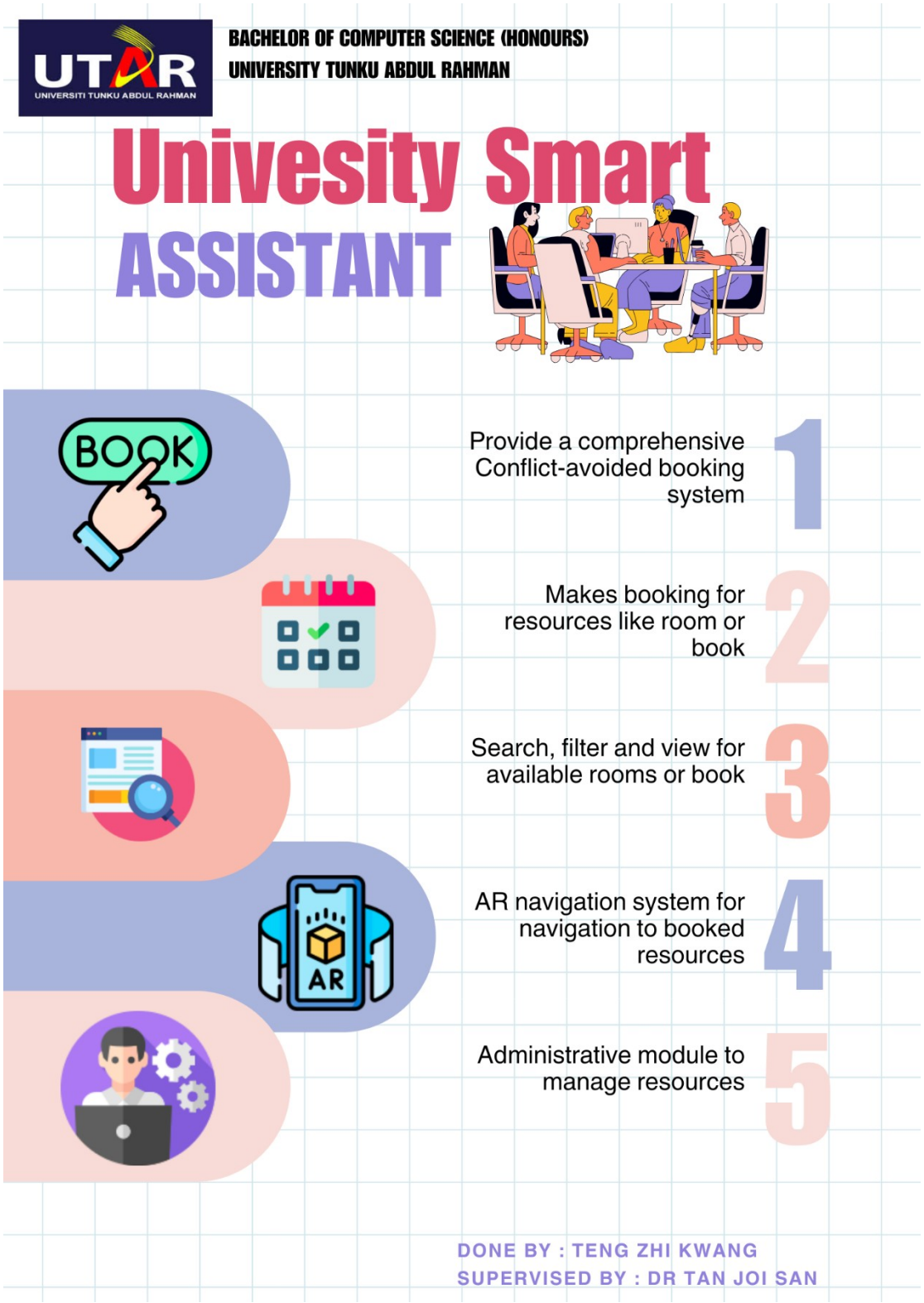
[28] Ganttic, “Plan your resources with maximum efficiency - Ganttic,” *Ganttic*, Jul. 15, 2024. Available: https://www.ganttic.com/

[29] S. Hoban, “Product Review: Ganttic — Sarah M. Hoban,” Sarah M. Hoban, Oct. 10, 2020. Available: https://www.sarahmhoban.com/blog/product-review-ganttic

[30]Skift Meetings, “Ganttic: the modern gantt-chart tool for your events [Review],” Skift Meetings, May 19, 2022. Available: https://meetings.skift.com/reviews/ganttic-modern-gantt-chart-tool-events-review/

[31] A. Haynes, “Ganttic: Overview, Features & Pricing - eLearning Industry,” eLearning Industry, Jul. 13, 2021. Available: https://elearningindustry.com/directory/elearning-software/ganttic/features

Poster


BACHELOR OF COMPUTER SCIENCE (HONOURS)
UNIVERSITY TUNKU ABDUL RAHMAN
Univesity Smart
ASSISTANT
BOOK
Provide a comprehensive Conflict-avoided booking system
1
Makes booking for resources like room or book
2
Search, filter and view for available rooms or book
3
AR
AR navigation system for navigation to booked resources
4
Administrative module to manage resources
5
DONE BY : TENG ZHI KWANG
SUPERVISED BY : DR TAN JOI SAN