HOUSE RENTAL MANAGEMENT SYSTEM (DREAMHOUSE)

MUHAMMAD AIMAN IZZAT BIN AZIZAN

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

HOUSE RENTAL MANAGEMENT SYSTEM (DREAMHOUSE)

MUHAMMAD AIMAN IZZAT BIN AZIZAN

This report is submitted in partial fulfillment of the requirements for the Bachelor of Computer Science (Database Management) with Honours.

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TEKNIKAL MALAYSIA MELAKA

2024

DECLARATION

I hereby declare that this project report entitled

HOUSE RENTAL MANAGEMENT SYSTEM (DREAMHHOUSE)

is written by me and is my own effort and that no part has been plagiarized without citations.

STUDENT : _____ _____ Date : _2/9/2024_
(MUHAMMAD AIMAN IZZAT BIN AZIZAN)

I hereby declare that I have read this project report and found this project report is sufficient in term of the scope and quality for the award of Bachelor of [Computer Science (Software Development)] with Honours.

SUPERVISOR : _____ Date : 5/9/2024
(NOOR AZILAH BINTI DRAMAN@MUDA)

DEDICATION

A lot of thank you to especially to my mother, Mrs. Zaiton Binti Teh and my father, Mr. Azizan Bin Abdul Latiff for giving me full support to manage my final year project well.

To my beloved supervisor, Noor Azilah Binti Draman@Muda, thank you for helping by giving ideas on my final year project.

And to the fellow friends of BITD, who gives co-operation and knowledge sharing in completing this project.

Thank you so much.

ACKNOWLEDGEMENTS

I would like to extend my deepest gratitude to all those who provided the expertise, knowledge, and resources that greatly assisted the research and development of the House Rental Management System (DreamHouse).

First and foremost, I express my sincerest appreciation to my supervisor, Noor Azilah Binti Draman@Muda whose guidance, support, and encouragement were invaluable throughout this project. Her mentorship was instrumental in shaping both the direction and success of this endeavor.

Special thanks to my peers and colleagues at Universiti Teknikal Malaysia Melaka (UTeM), especially those in the BITD class, for their thoughtful feedback and camaraderie during the demanding phases of the project. Their perspectives and critiques helped me address key challenges and improve the functionality and user experience of the system.

I am also grateful to the participants of the User Acceptance Testing phase, including tenants, guests, landlords, administrators, and agents, who took the time to engage with the system and provided valuable insights that have significantly enhanced the project's practicality and usability.

Lastly, I would like to thank my family and friends for their understanding and support throughout my study. Their encouragement when it was most needed is greatly appreciated.

This project could not have been accomplished without the contributions of each individual and organization mentioned above, as well as many others who, though not listed, have played valuable roles in the successful completion of this project.

ABSTRACT

The House Rental Management System (DreamHouse) project was developed to streamline the management and operation of rental properties through an integrated software solution. This system provides a comprehensive platform for tenants, guests, landlords, administrators, and agents to manage property listings, contracts, and user interactions efficiently. The primary objective of this project was to enhance the functionality and user experience of property management by leveraging modern web technologies.

The system architecture is built on a robust framework that ensures security, reliability, and scalability, addressing the complex needs of property management. Key features of the system include property listing management, search functionality with advanced filters, user role management, and a dynamic contract system. The system was designed with a user-friendly interface to ensure ease of use for individuals with varying levels of technical skills.

Extensive testing, including unit testing, integration testing, system testing, and user acceptance testing, was conducted to ensure the system met all functional requirements and performance benchmarks. The testing phases highlighted the system's capability to handle multiple user requests simultaneously, maintain data integrity, and protect against potential security threats.

The implementation of the House Rental Management System (DreamHouse) has demonstrated significant improvements in operational efficiency and user satisfaction. Feedback from user acceptance testing has been overwhelmingly positive, with particular commendation for the system's intuitive navigation and fast response times.

This project not only fulfills the academic requirements of Bachelor of Computer Science at Universiti Teknikal Malaysia Melaka (UTeM) but also provides a scalable solution that can be adapted to different markets within the real estate industry. Future enhancements will focus on incorporating artificial intelligence for predictive analytics and further automating property management processes.

ABSTRAK

Projek Sistem Pengurusan Sewaan Rumah (DreamHouse) telah dibangunkan untuk mempermudah pengurusan dan operasi hartanah sewaan melalui penyelesaian perisian yang terintegrasi. Sistem ini menyediakan platform yang menyeluruh bagi penyewa, tetamu, tuan tanah, pentadbir, dan ejen untuk mengurus senarai hartanah, kontrak, dan interaksi pengguna dengan cekap. Objektif utama projek ini adalah untuk meningkatkan fungsi dan pengalaman pengguna dalam pengurusan hartanah dengan memanfaatkan teknologi web moden.

Arkitektur sistem dibina atas kerangka kerja yang kukuh yang menjamin keselamatan, kebolehpercayaan, dan kebolehskalaan, memenuhi keperluan kompleks pengurusan hartanah. Ciri utama sistem ini termasuk pengurusan senarai hartanah, fungsi pencarian dengan penapis lanjutan, pengurusan peranan pengguna, dan sistem kontrak yang dinamik. Sistem ini direka dengan antaramuka yang mesra pengguna untuk memastikan kemudahan penggunaan bagi individu dengan pelbagai tahap kemahiran teknikal.

Pengujian yang luas, termasuk pengujian unit, pengujian integrasi, pengujian sistem, dan pengujian penerimaan pengguna, telah dilakukan untuk memastikan sistem memenuhi semua keperluan fungsi dan penanda aras prestasi. Fasa pengujian menonjolkan keupayaan sistem untuk mengendalikan beberapa permintaan pengguna secara serentak, mengekalkan integriti data, dan melindungi dari ancaman keselamatan yang berpotensi.

Pelaksanaan Sistem Pengurusan Sewaan Rumah (DreamHouse) telah menunjukkan peningkatan ketara dalam efisiensi operasi dan kepuasan pengguna. Maklum balas dari pengujian penerimaan pengguna amat positif, dengan pujian khusus terhadap navigasi intuitif sistem dan masa tindak balas yang cepat.

Projek ini tidak hanya memenuhi keperluan akademik Ijazah Sarjana Muda Sains Komputer di Universiti Teknikal Malaysia Melaka (UTeM) tetapi juga menyediakan penyelesaian yang boleh disesuaikan dengan pasaran yang berbeza dalam industri hartanah. Penambahbaikan masa depan akan memberi tumpuan kepada penggabungan kecerdasan buatan untuk analitik ramalan dan mengautomatikkan proses pengurusan hartanah dengan lebih lanjut.

TABLE OF CONTENTS

PAGE

LIST OF TABLES

PAGE

LIST OF FIGURES

PAGE

LIST OF ABBREVIATIONS

FYP - **Final Year Project**

LIST OF ATTACHMENTS

CHAPTER 1: INTRODUCTION

1.1 Introduction

This project developed for the use of people that need to find house rental because of there need to work outside from their hometown or need to live at new place. Sometime people cannot have opportunity to get house rental because of there have lack of information to get about house rental.

The House Rental Management System (DreamHouse) is one of the platform that manage house and tenant properly. Information associated with house rental will be stored systematically and associated contracts can be produced using the proposed system.

1.2 Problem Statement (s)

Below are the problem statement of House Rental Management System (DreamHouse):

1. Manually, data of rental agreement is recorded on books/papers which may easily get damaged leading to loss of data.

    a. All the information regarding the sport events are recorded manually using form and were placed in a file. Therefore, loss of data may occur. It is also hard for house rental management landlord to update and maintain the data. Using the system developed, it will help the landlord in managing their house rental by using the system which is recording and maintain the data.

2. No proper way in recording important information.

    a. The House Rental Management System (DreamHouse) have no any database in storing all the information associated with information of house detail, dealing the contracts and tenant management. The new system developed important to make sure that all the data are recorded in a proper storage.

3. Inadequate information management cause reports not to be accurately and waste of time.

    a. In the conventional approach to house rental management, inadequate information management poses a significant challenge. The lack of efficient data organization and retrieval systems leads to inaccurate reports and a waste of time. This results in inefficiencies in decision-making processes and hampers the overall effectiveness of rental property management.

1.3 Objective

Below are the objective of House Rental Management System (DreamHouse):

1. Improve data security and accessibility using a website designed for future used.

    a. Implement a secure digital database system to safeguard rental property information from potential loss or damage. Ensure easy accessibility to data for authorized users, reducing the risk of data mishandling or loss.

2. Improve information recording efficiency and more friendly to update, maintenance and advertisement.

a. Establish a systematic method for recording and storing crucial information related to house details, contracts, and tenant management. Streamline data entry processes to reduce manual errors and ensure completeness and accuracy of information.

3. Enable accurate reporting based on record of data existed for monitoring and keep up to date for user about rental.

a. Develop robust information management practices to facilitate the generation of accurate reports on various aspects of rental property management. Provide tools and features for data analysis and visualization to support performance monitoring.

1.4 Scope (the boundary of system)

1. Login Module

a. In this module, the user will use their email and password for login based or their role either staff, agent, landlord, tenant, or guest to ensure the system cannot be accessed or change by an unauthorized user.

b. Allow new users to register themselves by providing necessary information and creating login credentials.

2. Staff Module

a. This module handle user registration for agent and landlord, authentication and profile management including personal information and preferences.

b. Allow staff to manage property registered by landlord, assigned agent to property registered to handle.

c. This module can view and manage user which are agent, landlord, and tenant. Staff also can manage contracts made by the agent.

3. Agent Module

a. This module gives permission for agents to create contract for tenant.

b. Manage their profile except email and integrated card number (I/C).

c. The agent must fill the form of appointment to to set as reminder and KPI tracker for the staff.

4. Landlord Module

a. This module can access information about current tenants, including contract details and payment history made by tenants.

b. It also can track rent payments, due dates, and any outstanding payments.

c. Maintain and manage comprehensive database of each property, including its location, type, size, and specific features.

d. Manage their profile except email and integrated card number (I/C).

5. Tenant Module

a. The tenant module allows tenants to monitor their contract information, including making payment for proceed contracts and monthly payment as rental agreement in contract.

b. Manage their profile except email and integrated card number (I/C).

6. Complaint Module

    a. Tenants can submit complaints or issues through an online form within the system to notify the agent about the problems.

    b. This module can enable communication between tenants and property agents to take action in solving the problem regarding specific complaints and notify the tenants by adding remarks for the complaints.

7. Payment Module

    a. This module handles rental payments, security deposit, and any other financial transactions related to the rental process.

    b. The tenant can add their new payment and have notification to alert them about payment of rental.

8. Contract Module

    a. This module specifies the start and end dates of the lease agreement based on date start and period of the agreement valid.

    b. Outline monthly rent, due date, balance, and the status of the house contract.

9. Appointment Module

    a. Guest/Tenant can set appoitnemnt with agent.

    b. Agent needs to approve/reject and email will sent to the tenant about the appointment confirmation.

1.5 Project significance

Below are the project significance for House Rental Management System (DreamHouse):

a. The House Rental Management System (DreamHouse) represents a significant advancement in the realm of property rental management, particularly for university students seeking off-campus accommodations. By providing a centralized platform for property search, contract management, and communication with landlords or agents, DreamHouse streamlines the often daunting process of finding suitable housing. Students, especially those facing challenges securing university-provided hostel accommodations, benefit from the system's accessibility and transparency, which ensure a smoother rental experience. Moreover, DreamHouse offers landlords and property agents a streamlined approach to property management, enabling efficient listing, contract creation, and tenant management, ultimately enhancing the rental process's overall efficiency and effectiveness.

b. In addition to its practical benefits, DreamHouse signifies a shift towards modern, tech-enabled solutions in the rental property industry. By digitizing traditional rental processes and offering a user-friendly interface, the system reflects the growing importance of technology in optimizing various aspects of daily life, including housing. Its implementation not only addresses immediate housing needs but also sets a precedent for future innovations in property management, benefiting both students and landlords while paving the way for more efficient and transparent rental practices in the future.

1.6 Expected output

Below are the expected outputs for House Rental Management System (DreamHouse) when was done be developed:

1. Secure and Accessible Database

    a. Implementation of a secure digital database system that safeguards rental property information, ensuring easy accessibility for authorized users while minimizing the risk of data loss or mishandling.

2. Efficient Information Recording

    a. Establishment of a systematic method for recording and storing crucial information related to property details, contracts, and tenant management, ensuring completeness, accuracy, and ease of access.

3. Accurate Reporting Tools

    a. Development of robust information management practices that facilitate the generation of accurate reports on various aspects of rental property management, enabling informed decision-making and performance monitoring.

4. User-friendly Interface

    a. Creation of a user-friendly interface that simplifies property search, communication with agents, and contract management for tenants, landlords, and property agents, enhancing overall user experience and satisfaction.

5. Streamlined Operations

    a. Optimization of property management operations through centralized property listings, contract management, and tenant communication, saving time and resources for landlords and property agents while improving visibility and efficiency.

1.7 Conclusion

The House Rental Management System (DreamHouse) represents a significant innovation for a broad audience beyond just university students and landlords. By providing a centralized platform for property search, contract management, and communication, DreamHouse facilitates the discovery and management of off-campus accommodations. Its accessibility and transparency not only benefit those directly involved in the university community but also extend to a wider range of renters and property managers. The system enhances operational efficiency for landlords and agents, establishing a model for technology-driven solutions in the rental market. Key features include a secure database, precise information recording, effective reporting tools, and a user-friendly interface, all of which contribute to a smoother rental process for all participants.

CHAPTER 2: LITERATURE REVIEW AND PROJECT METHODOLOGY

2.1 Introduction

This chapter outlines the approach and planning involved in developing the House Rental Management System (DreamHouse). It covers the methodologies adopted for system development, including requirements gathering, design, implementation, and testing phases.

Additionally, it details the project timeline, resources, and tools utilized to ensure the successful completion of the project. By adhering to a structured methodology, the project aims to deliver an efficient and user-friendly rental management system tailored to the needs of university students and landlords.

2.2 Project Methodology

The development of the House Rental Management System (DreamHouse) follows a structured approach based on the Database Life Cycle (DBLC) phases. The DBLC ensures a systematic process for designing, implementing, and maintaining the database, crucial for the success of the project. Below are the DBLC phases and the associated tasks, along with the plan for performing these tasks.

Table 2.3.1-1 Database Life Cycles Phases

Database Initial Study	The initial study phase involves understanding the requirements and scope of the database to be developed. This phase is critical to identify the objectives and goals of the system and ensure that the database aligns with these goals. During this phase, interviews and surveys were conducted with potential users, including students, landlords, agents, and staff, to gather requirements. These interactions provided valuable insights into the needs and challenges faced by each group. Existing systems were analyzed to identify gaps and areas for improvement, ensuring that the new system addresses all deficiencies. The objectives and goals of the House Rental Management System (DreamHouse) were clearly defined, setting a solid foundation for the project. A feasibility study was conducted to evaluate the practicality, costs, benefits, and potential challenges of the proposed system, ensuring its viability and alignment with user needs. In this phase, the task will included as below: 1. Requirements Specification Document. • Detailed documentation of user requirements and system functionalities. 2. Feasibility Study Report. • Analysis of the project's practicality and benefits, including cost-benefit analysis and risk assessment. 3. Project Charter. • Formal document that outlines the project's objectives, scope, stakeholders, and overall plan.

Database Design	The database design phase translates the requirements gathered in the initial study into a detailed blueprint for the database. This phase involves both logical and physical design activities. An Entity-Relationship Diagram (ERD) was developed to model the database structure, representing entities, relationships, and attributes. A data dictionary was defined, specifying the attributes, data types, and constraints for each entity, ensuring data consistency and integrity. The database was normalized to eliminate redundancy and ensure efficient data storage and retrieval. The physical schema was designed, including indexing strategies and storage requirements, to optimize performance and ensure scalability. In this phase, the task will included as below: 1. Entity-Relationship Diagram (ERD). • Visual representation of the database structure, showing entities, relationships, and key attributes. 2. Data Dictionary. • Detailed description of the database schema, including data types, constraints, and relationships. 3. Normalized Database Schema. • Database design that minimizes redundancy and dependency, ensuring data integrity and efficiency. 4. Physical Database Design Document. • Documentation of the physical aspects of the database, including storage, indexing, and performance considerations.

Implementation & Loading	The implementation phase involves the actual creation of the database using a Database Management System (DBMS). This phase also includes loading the initial data into the database. An appropriate DBMS (e.g., MySQL, PostgreSQL) was selected for the system based on factors like scalability, reliability, and performance. SQL scripts were written to create the database schema based on the design documents. These scripts included the creation of tables, indexes, and constraints. Scripts were also developed to populate the database with initial data, such as property details, user accounts, and contract information, ensuring a smooth transition from existing systems. User interfaces for data entry and management were implemented to facilitate easy interaction with the database. In this phase, the task will included as below: 1. SQL Scripts for Database Creation. • Scripts to create the database schema, including tables, indexes, and constraints. 2. Initial Data Loading Scripts. • Scripts to populate the database with initial data, ensuring data accuracy and completeness. 3. User Interface Prototypes. • Initial designs and implementations of user interfaces for data entry and management.

Testing & Evaluation	The testing and evaluation phase ensures that the database and the associated system functions correctly and meets the specified requirements. This phase involves various types of testing to identify and fix any issues. Test plans and test cases were developed for functional and non-functional requirements, outlining the testing approach and expected outcomes. Unit testing was conducted to verify individual components, ensuring they function as expected. Integration testing was performed to ensure different modules work together seamlessly. User acceptance testing (UAT) was conducted with actual users to gather feedback, ensuring the system meets user expectations and requirements. Identified issues were logged, tracked, and resolved systematically. In this phase, the task will included as below: 1. Test Plans and Test Cases. • Detailed plans and cases for testing various aspects of the system, including functional and non-functional requirements. 2. Test Reports (Unit, Integration, UAT). • Reports documenting the outcomes of testing phases, including identified issues and their resolution status. 3. Bug and Issue Logs. • Logs of identified bugs and issues, including their severity, status, and resolution.

Operation	The operation phase involves deploying the system into a live environment and ensuring its smooth operation. This phase includes training users and providing necessary support. The database and the application were deployed on a production server, following a well-defined deployment plan to ensure minimal disruption. Training sessions were conducted for users to familiarize them with the system, covering key functionalities and best practices. User manuals and documentation were provided to assist users in navigating the system and troubleshooting common issues. System performance was monitored continuously, and necessary adjustments were made to ensure optimal performance. In the phase, the task will included as: 1. Deployment Plan. • Detailed plan for deploying the system into a live environment, ensuring minimal disruption. 2. User Training Materials. • Training materials to help users understand and use the system effectively. 3. User Manuals and Documentation. • Comprehensive documentation to assist users in navigating the system and troubleshooting common issues.
Maintenance & Evaluation	The maintenance phase ensures the long-term success of the database system by addressing any issues that arise and making necessary updates and improvements. System performance and usage were monitored to identify areas for improvement. Regular backups were performed, and data recovery mechanisms were put in place to ensure data integrity and availability. The system was updated to accommodate new requirements or changes in the business process, ensuring it remains relevant and effective. Periodic evaluations were conducted to assess the system's effectiveness and user satisfaction, ensuring continuous improvement and adaptation to changing needs. In this phase, the task included as below: 1. Maintenance Schedule and Logs. • Schedule and logs of maintenance activities, ensuring regular updates and issue resolution. 2. System Update Records.

	• Records of system updates and changes, ensuring transparency and accountability. 3. Evaluation Reports. • Reports assessing the system's effectiveness and user satisfaction, identifying areas for improvement.

2.3 Project Schedule and Milestones

2.3.1 Milestones Timeline

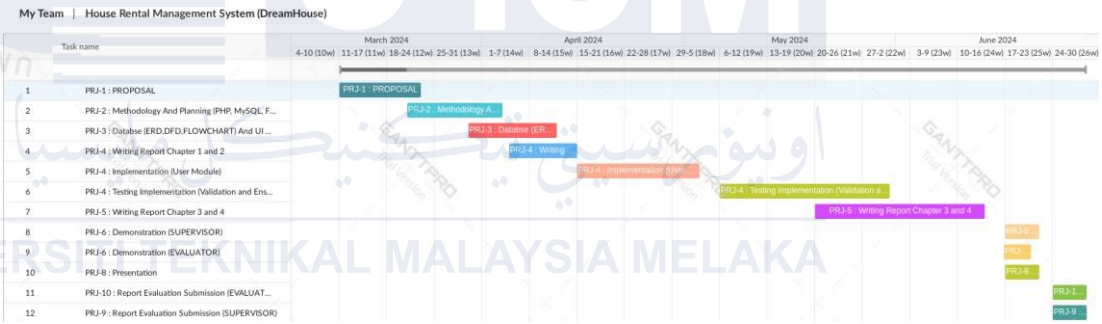


Figure 2.3-1 Gantt Chart

2.3.2 List of Milestones

Table 2.3.2-1 List of Milestones

Start Date	End Date	Duration	Assessment	Description
11 March 2024	22 March 2024	12 days	PRJ-1: PROPOSAL	The initial document used to define an internal or external project. The proposal includes sections such as title, project background, problem statements, objectives, scopes,

				start and end dates, and a descriptor of the proposed solution. Use the form provided that can be downloaded from ULearn. REQUIREMENT: Log Record, Document - Completed proposal form.
25 March 2024	29 March 2024	5 days	PRJ-2: PROJECT PROGRESS 1	Planning and Illustrate design for the system
1 April 2024	12 April 2024	12 days	PRJ-3: REPORT WRITING PROGRESS 1	Design efficiently database before avoid missing important part for display in interface. (Report Chapter 1 completed)
15 April 2024	26 April	12 days	PRJ-4: PROJECT PROGRESS 2	Debugging code for system which interface. (Report Chapter 2 completed)
6 May 2024	14 June 2024	40 days	PRJ-5: REPORT WRITING PROGRESS 2	Completing in styling interface, finalize display of data in display. (Report Chapter 3 and Chapter 4 completed)

17 June 2024	21 June 2024	5 days	PRJ-6: DEMONSTRATION (SUPERVISOR)	Demonstration of the project results to supervisor which is Dr. Noor Azilah.
17 June 2024	21 June 2024	5 days	PRJ-7: DEMONSTRATION (EVALUATOR)	Demonstration of the project results to evaluator and supervisor which is Dr. Norashikin and Dr. Noor Azilah.
17 June 2024	21 June 2024	5 days	PRJ-8: PRESENTATION	Demonstration of the project results to evaluator and supervisor which is Dr. Norashikin and Dr. Noor Azilah.
24 June 2024	28 June 2024	5 days	PRJ-10: REPORT EVALUATION (EVALUATOR)	PSM1 Draft report for evaluation by Evaluator. REQUIREMENT: Log Record, Document - PSM1 Draft Report
24 June 2024	28 June 2024	5 days	PRJ-9: REPORT EVALUATION (SUPERVISOR)	PSM1 Draft report for evaluation by Evaluator. REQUIREMENT: Log Record, Document - PSM1 Draft Report

2.4 Conclusion

In conclusion, the methodological rigor and comprehensive planning detailed in this chapter set a solid foundation for the successful implementation of the DreamHouse system. It positions the project team to effectively manage, deliver, and sustain a system that meets the dynamic needs of its users while adhering to high standards of quality and performance. This strategic approach not only assures project success but also enhances the system's capacity to adapt to future requirements, ensuring long-term relevance and usability.

CHAPTER 3: ANALYSIS

3.1 Introduction

This chapter provides a critical evaluation of the existing House Rental Management System, DreamHouse, identifying its reliance on outdated methodologies that hinder functionality and accessibility for a broader audience. It explores the market demands and specific challenges faced by renters across various demographics, who contend with inefficient systems that often fail to provide competitive rental pricing. Through an extensive assessment involving user feedback, market research, and competitive analysis, this chapter uncovers significant deficiencies and highlights opportunities for substantial enhancements.

The insights derived are instrumental in transforming DreamHouse into a user centric platform, addressing critical issues such as affordability and usability. By focusing on a tailored approach, the system is re-engineered to not only meet but surpass the diverse expectations of all stakeholders, including renters, landlords, and agents. This paves the way for a more practical and efficient rental management solution that can adapt to the evolving needs of the rental market.

3.2 Problem Analysis

The examination of the current house rental management system reveals it relies on outdated methods that diminish accessibility and usability for a wider audience. The system's inefficiencies are particularly problematic due to its failure to provide competitive rental pricing, an aspect critical for individuals constrained by budget considerations. The introduction of the DreamHouse system seeks to address these challenges by offering a modern, intuitive platform that streamlines the process

of finding and managing rental properties. This system also ensures a variety of housing options to accommodate diverse financial capabilities, making it more inclusive and practical. By leveraging advanced technology, DreamHouse aims to enhance accessibility and cost-effectiveness, meeting the essential needs of its users including renters, landlords, and agents, thus revolutionizing the rental management experience.

Figure 3.2-1 Context Diagram of Old System

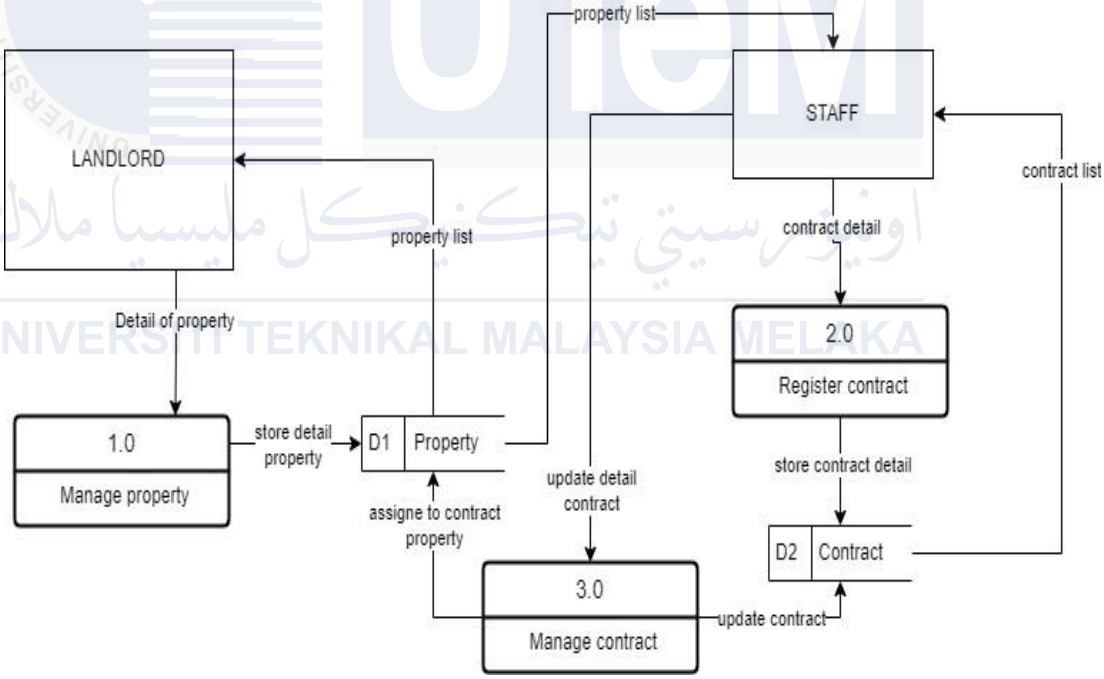


Figure 3.2-2 Data Flow Diagram

Figure 3.2-3 Interface of The Old System

The old system used for house rental management was heavily reliant on manual operations, making it inefficient and cumbersome for both tenants and landlords. This traditional approach typically involved face-to-face interactions for every step of the process, from viewing properties and negotiating terms to signing contracts and making payments. Such systems often used paper-based records for maintaining details about tenants, leases, and payment histories. The lack of digital tools not only made the process time-consuming but also increased the risk of data errors and loss due to mishandling or misplacement of physical documents.

Furthermore, the old system did not cater well to the dynamic needs of university students, who often look for flexible and budget-friendly housing options. Without the ability to easily compare prices or explore different properties online, students were limited to the listings they could physically visit or learn about through word-of-mouth. This system also failed to provide a platform for transparent communication between tenants and agents, which could lead to misunderstandings and disputes over lease terms and property conditions. The need for DreamHouse emerged as a response to these inefficiencies, aiming to introduce a digital solution that enhances accessibility, improves the efficiency of rental processes, and offers a more competitive and transparent marketplace for student housing.

3.3 The proposed improvements/solutions

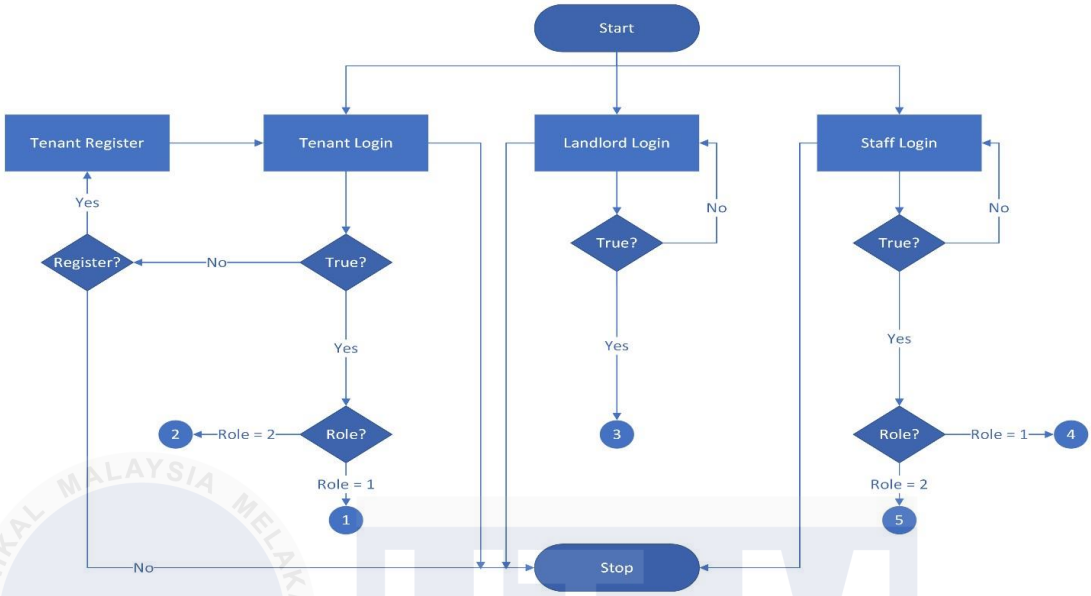


Figure 3.3-1 Initial State

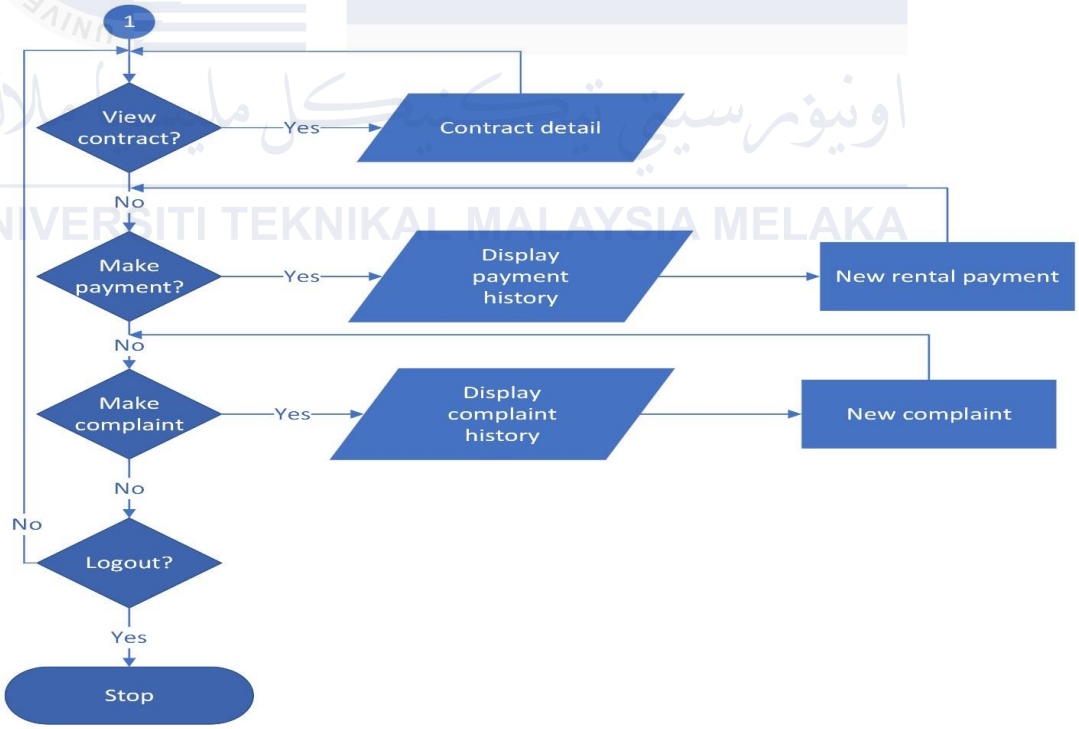


Figure 3.3-2 Tenant Page

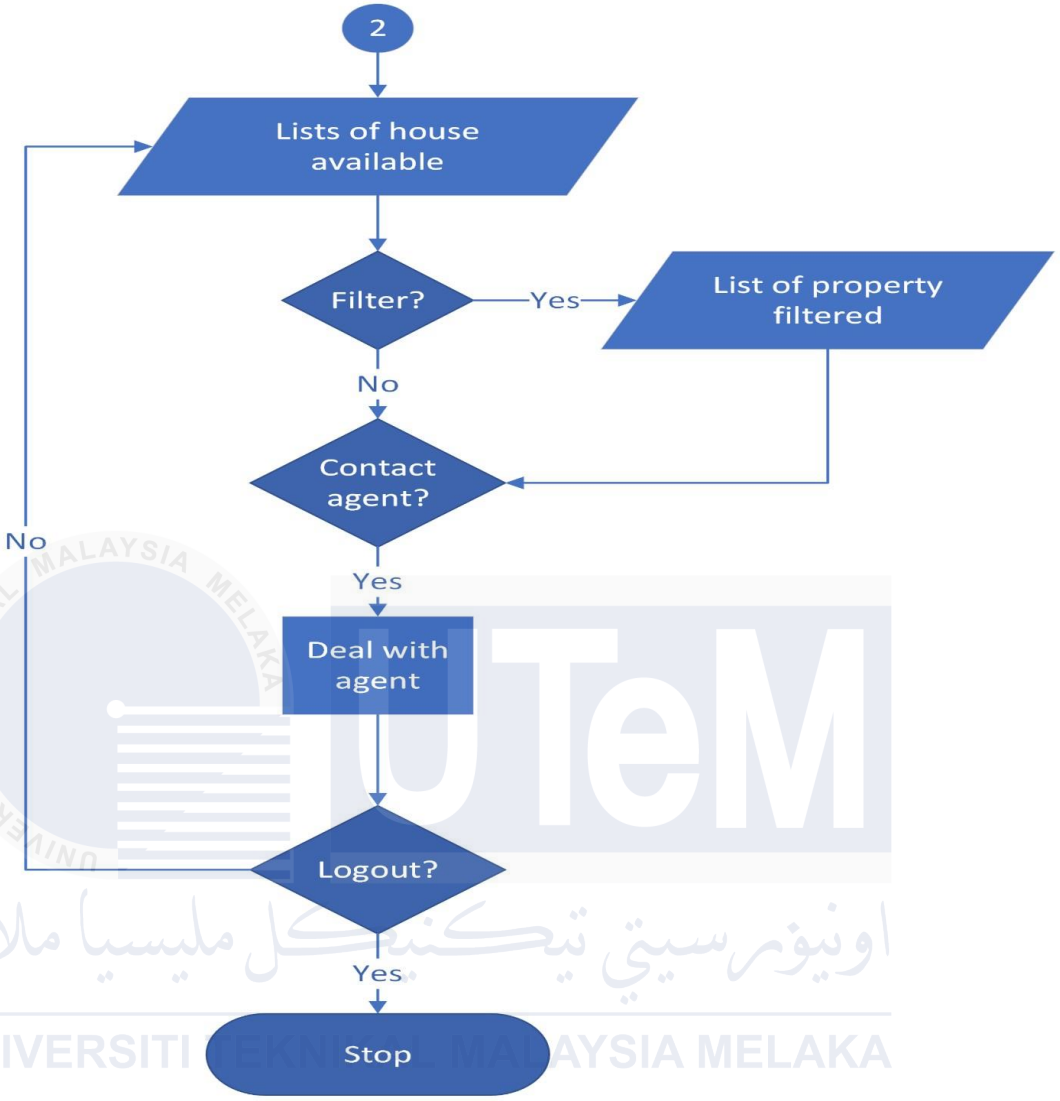


Figure 3.3-3 Guest Page

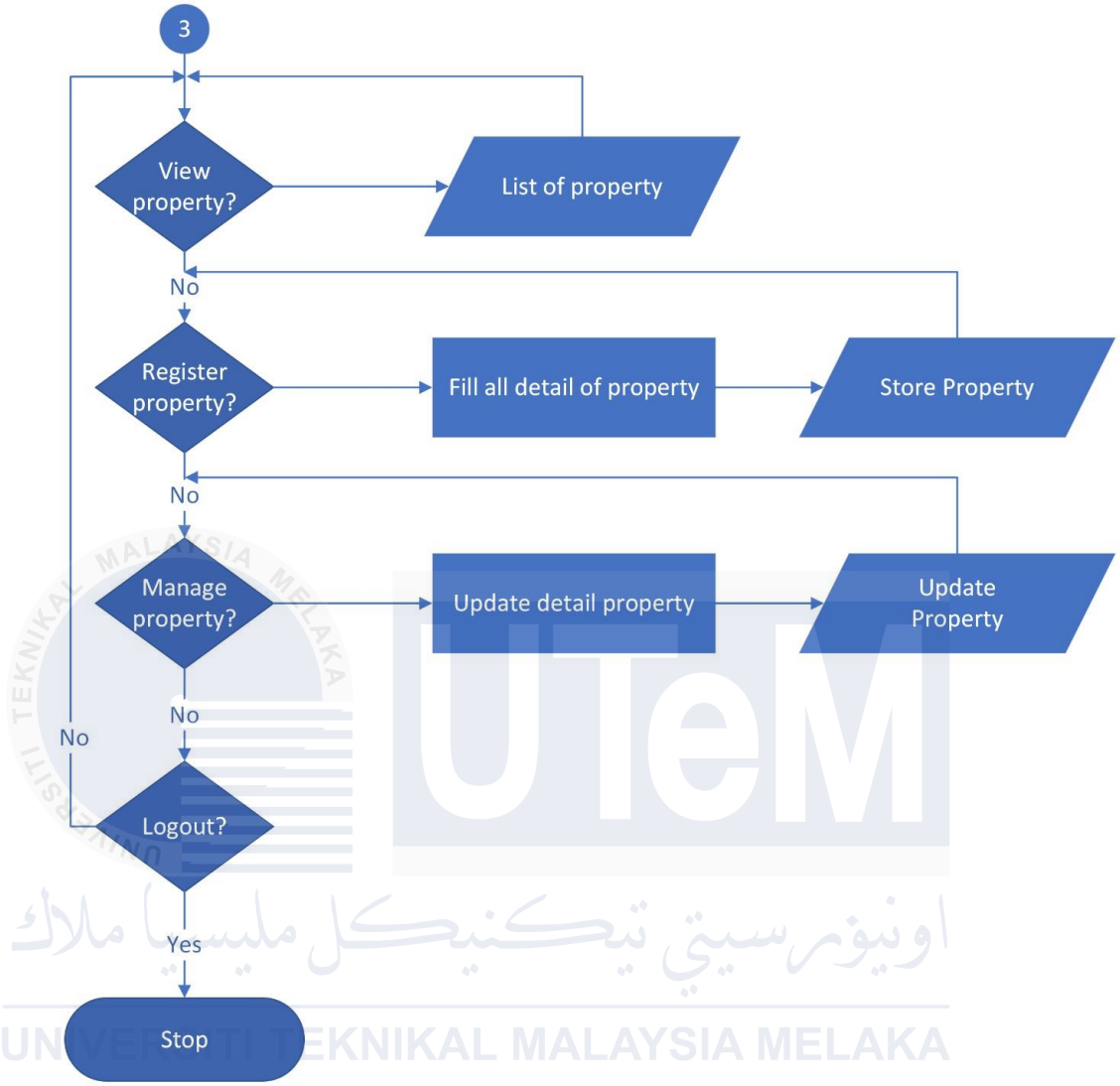


Figure 3.3-4 Landlord Page

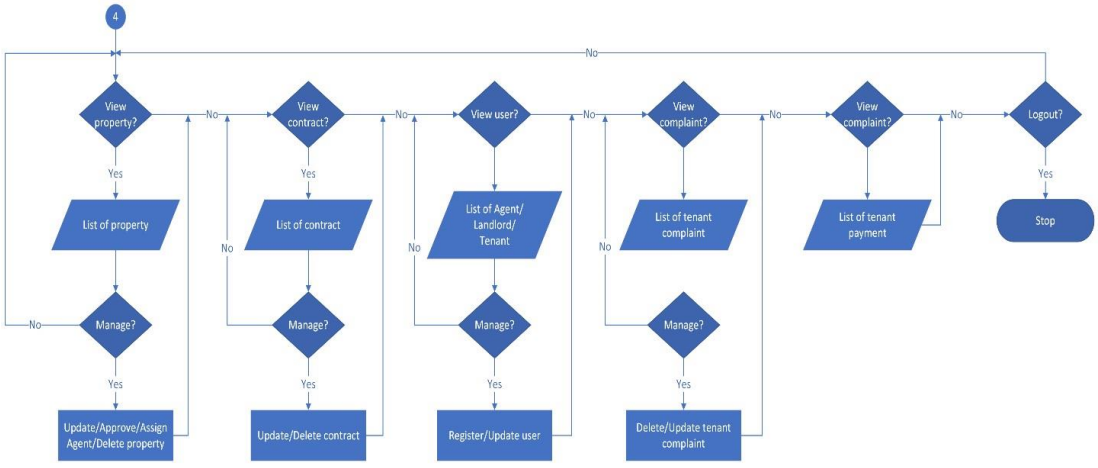


Figure 3.3-5 Staff Page

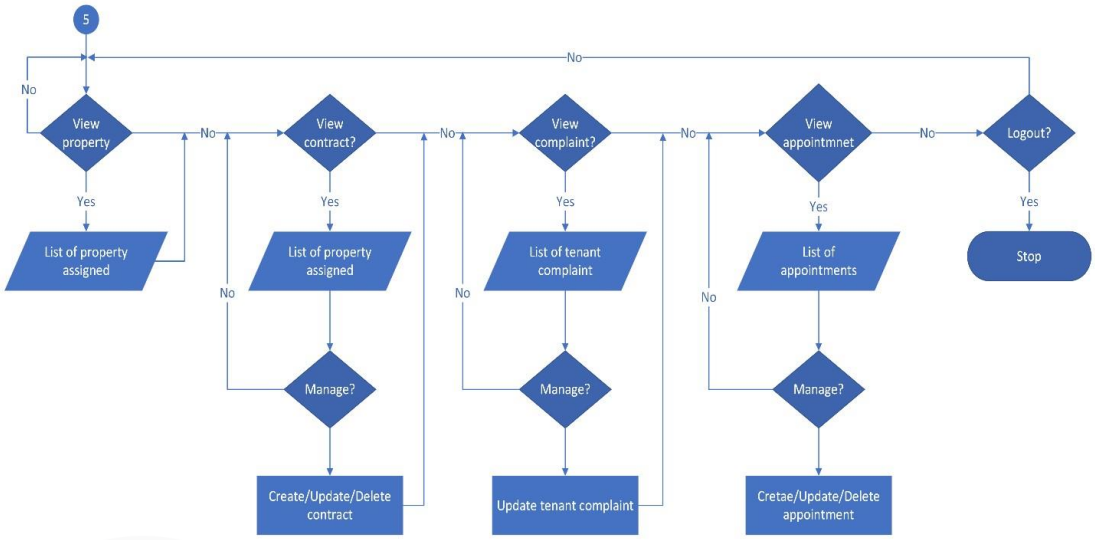


Figure 3.3-6 Agent Page

3.4 Requirements analysis of the to-be-system

3.4.1 Functional Requirement (Process Model)

The functional requirements of the forthcoming DreamHouse system are designed to enhance efficiency and automation within house rental processes, specifically tailored for university students and landlords. This design addresses the shortcomings of the existing manual system by integrating essential functionalities into a process model. These include User Registration and Authentication for secure access, Property Listing Management for up-to-date property details, Rental Application Processing for streamlined applications, Contract Management for digital handling of agreements, a Payment System with secure processing and reminders, Maintenance and Support Requests for operational efficiency, and Reporting and Analytics for strategic oversight. Collectively, these features aim to significantly improve the rental management experience, making it more accessible and user friendly.

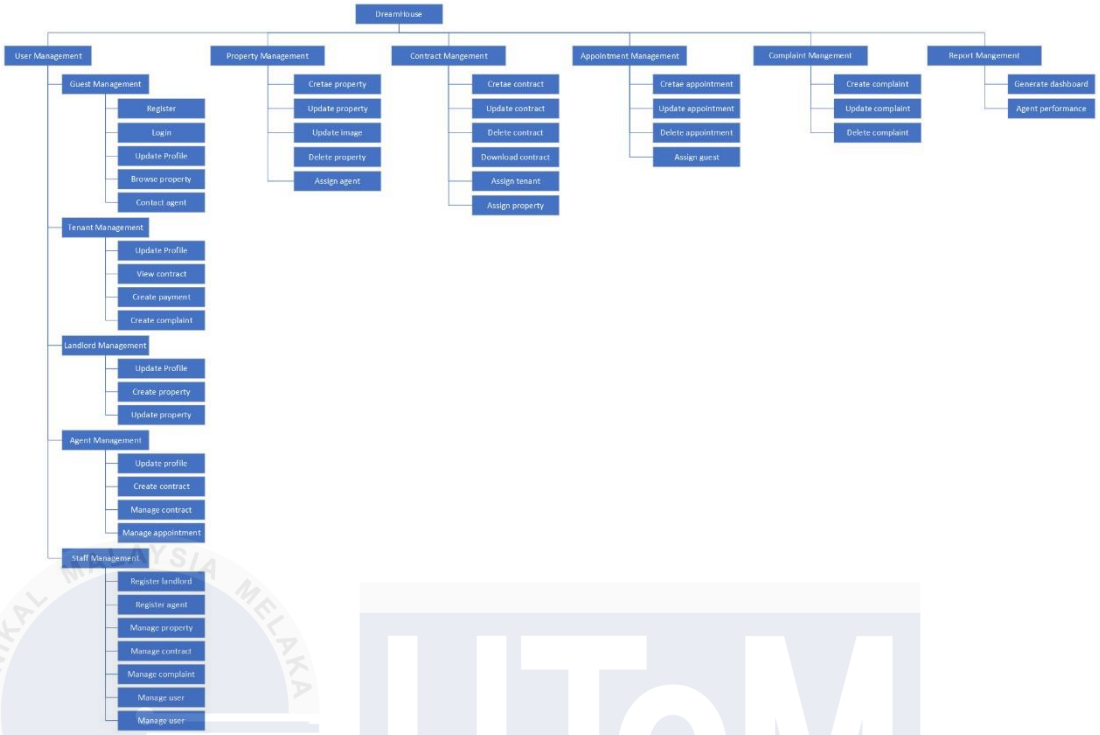


Figure 3.4-1 Structure Chart

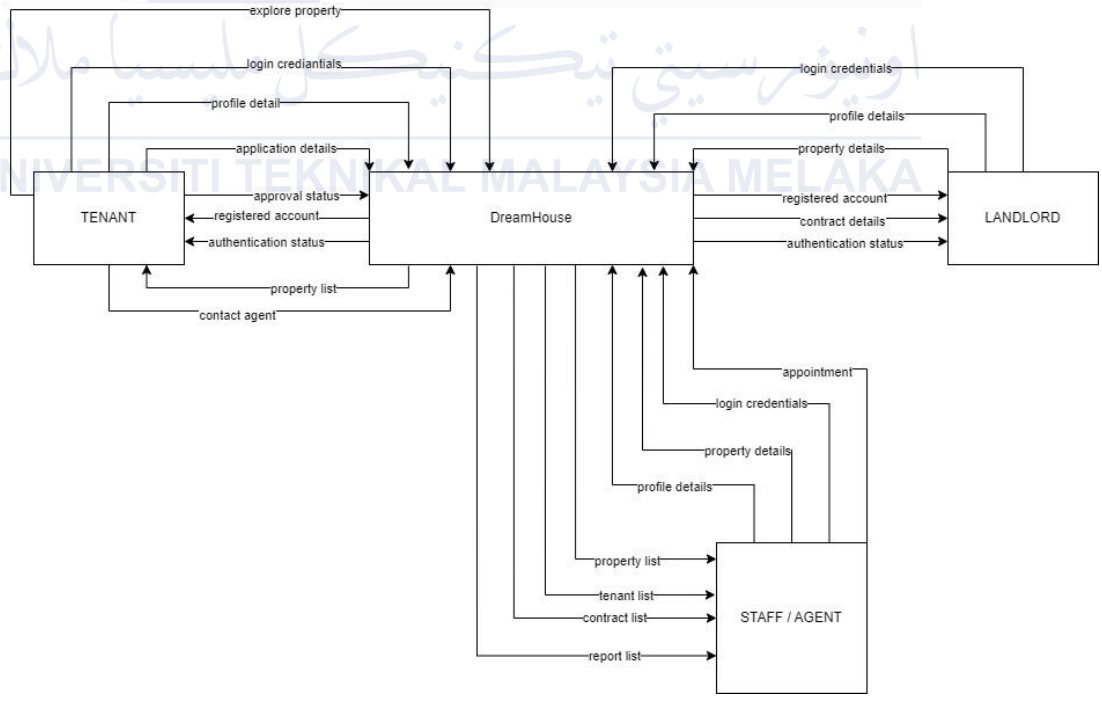


Figure 3.4-2 Context Diagram

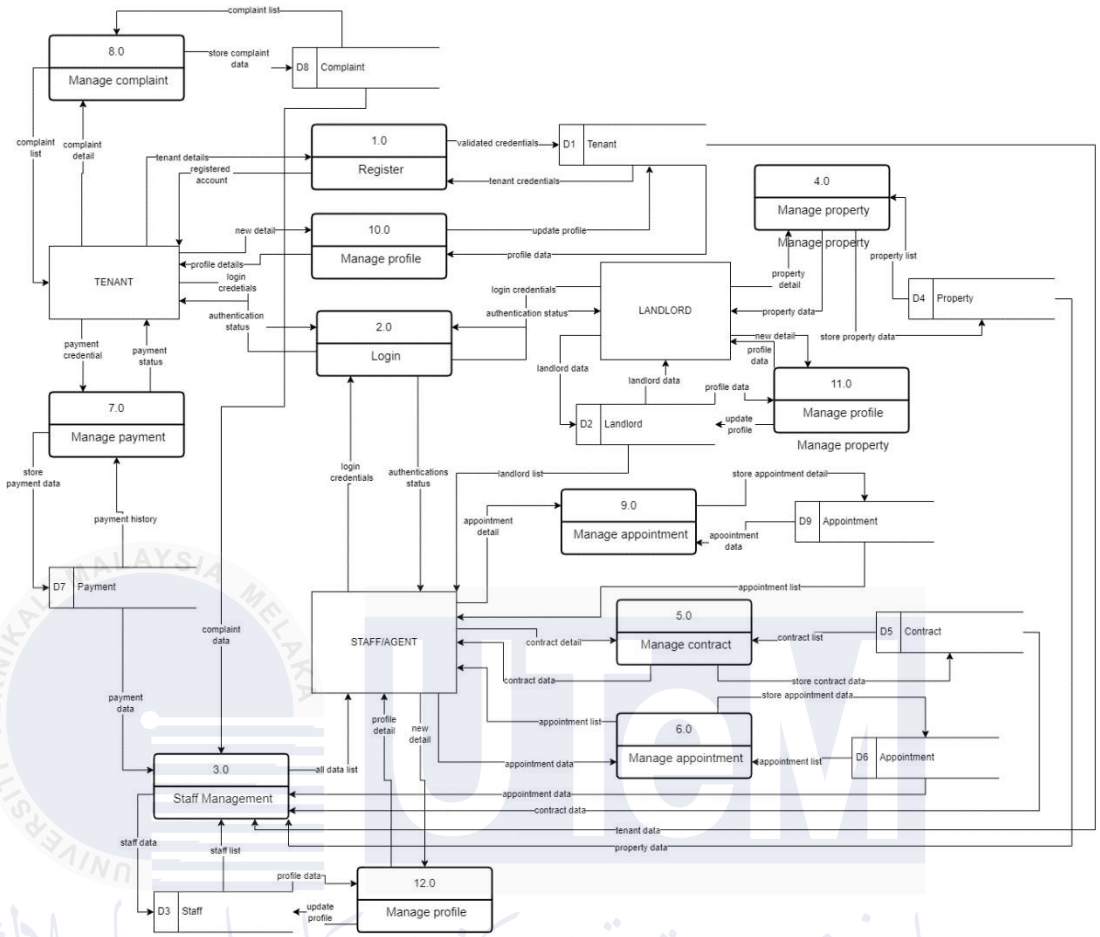


Figure 3.4-3 Data Flow Diagram Level 0

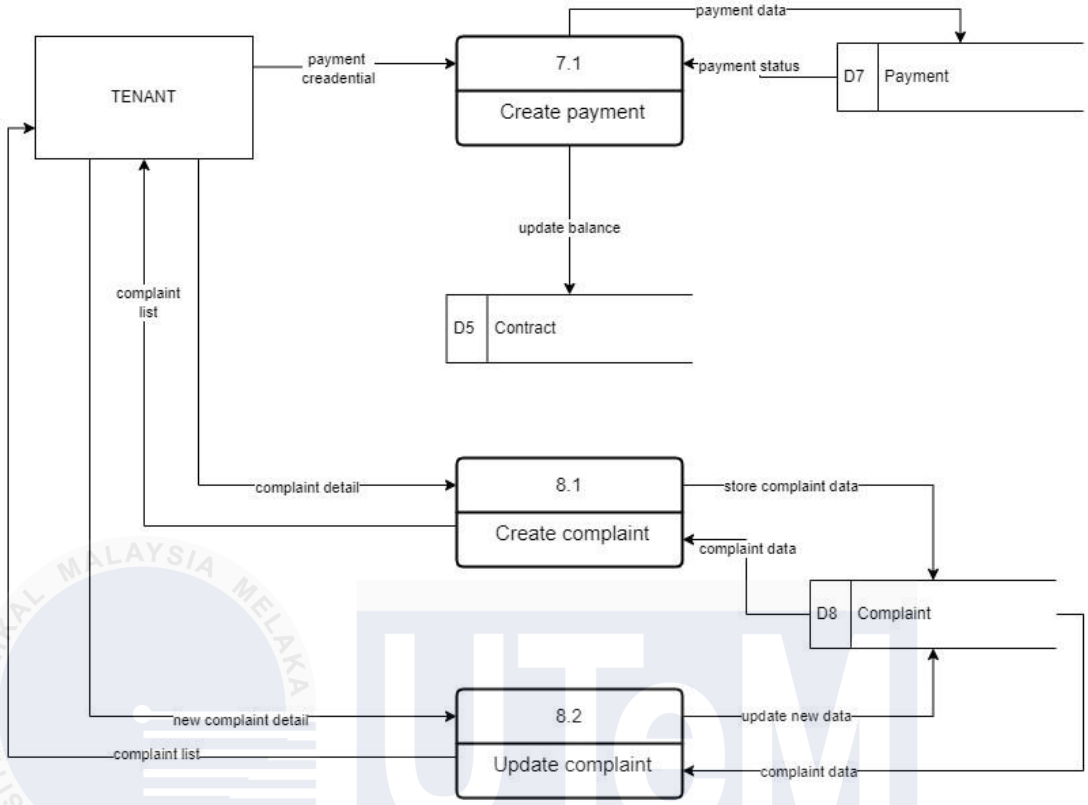


Figure 3.4-4 Data Flow Diagram Level 1 (Tenant - Manage Payment, Manage Complaint)

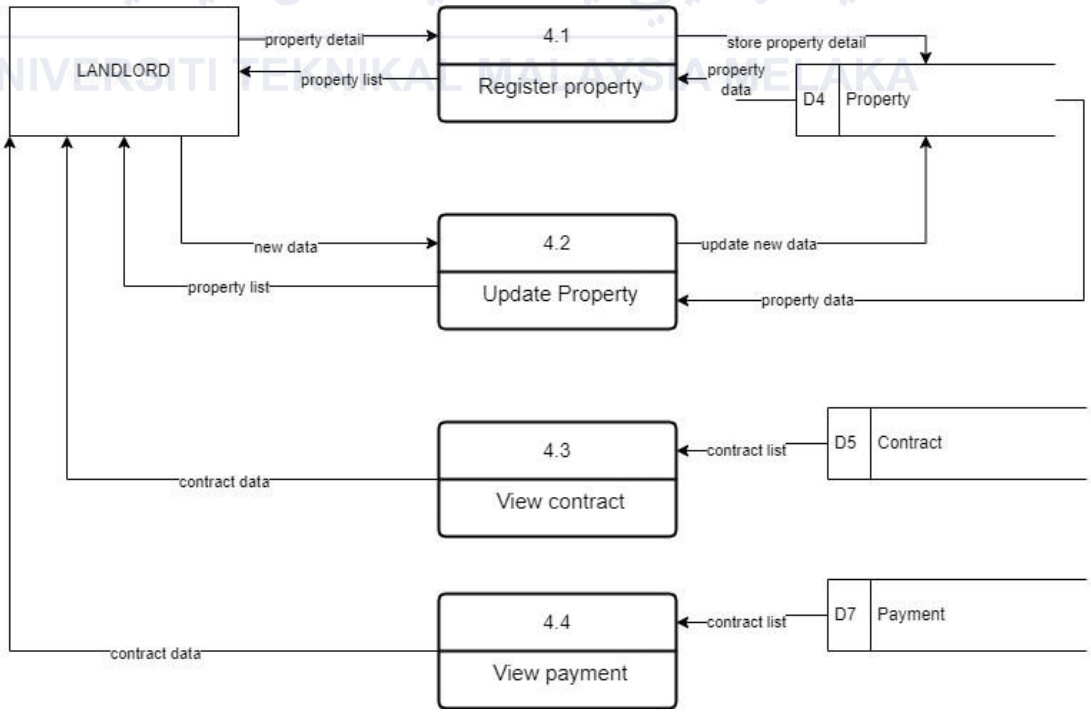


Figure 3.4-5 Data Flow Diagram Level 1 (Landlord - Manage Property)

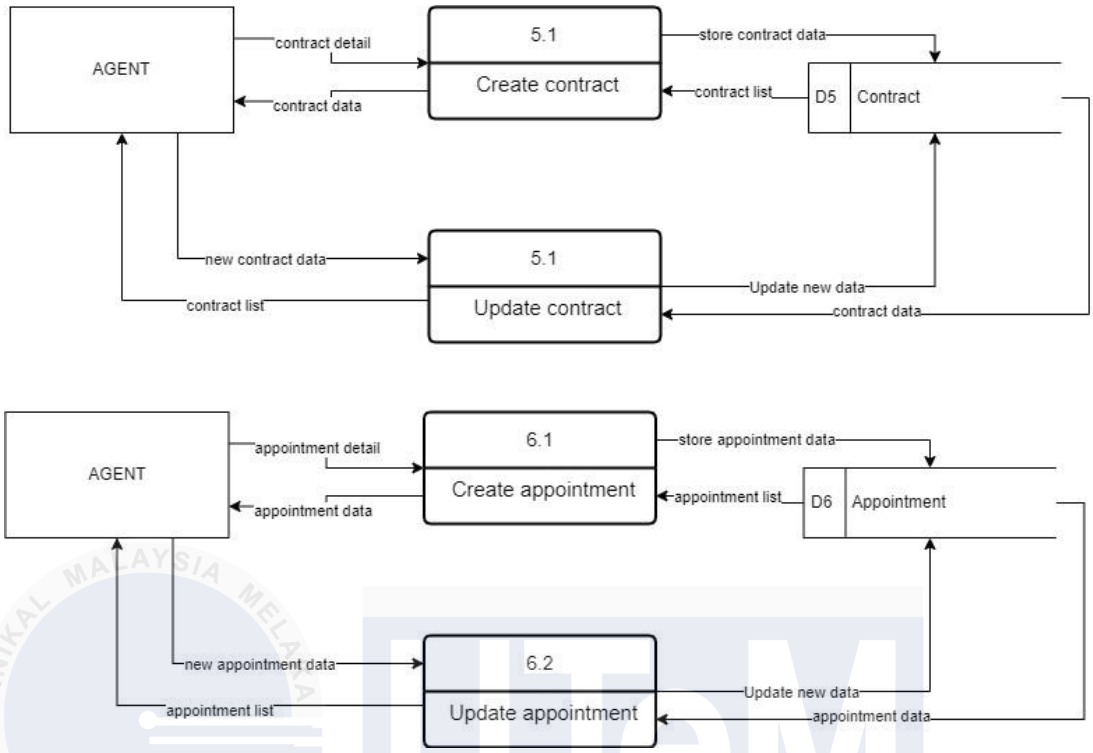


Figure 3.4-6 Data Flow Diagram Level 1 (Agent - Manage Contract, Manage Appointment)

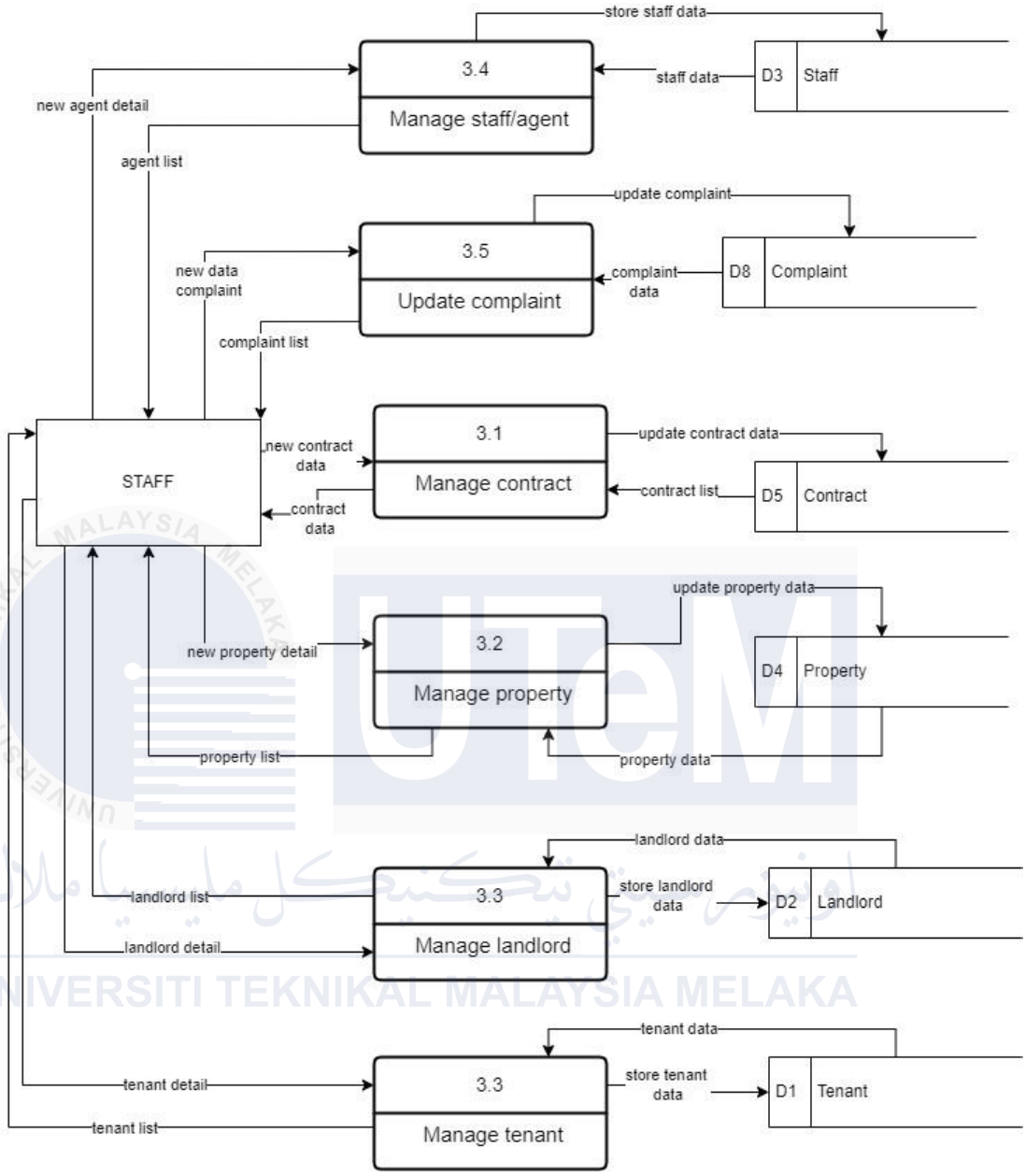


Figure 3.4-7 Data Flow Diagram Level 1 (Admin – Staff Management))

3.4.2 Non-functional Requirement

1. Quality Requirements

- DreamHouse prioritizes accuracy and security to ensure the system is reliable for students and landlords. Accuracy is crucial, as it impacts everything from rental prices to lease agreement details, ensuring that all information is up-to-date and correct, preventing misunderstandings or financial discrepancies. Security measures are vital to protect personal information like contact details, financial data, and personal preferences. As students increasingly rely on digital solutions, DreamHouse is committed to enhancing its security protocols continuously. This includes using modern encryption techniques and conducting regular security assessments to stay ahead of potential cyber threats, ensuring that user data is always protected.

2. Performance Requirements

- Performance in the DreamHouse system is measured by how quickly the system responds to user requests, such as searching for a rental or processing a payment, and its ability to handle many users at once without slowing down. This is particularly important during peak rental periods, such as at the beginning of an academic term when many students are looking for accommodation. The system's design aims for minimal downtime, with maintenance typically scheduled during off peak hours to avoid inconveniencing users. Continued monitoring of system performance helps identify and resolve any issues quickly, maintaining a smooth and efficient user experience.

3. Usability Requirements

- Usability in DreamHouse focuses on making the system easy and intuitive for all users, regardless of their tech savviness. This includes a clean, simple interface that minimizes the steps needed to perform actions like submitting rental applications or signing lease agreements. Usability also involves accessibility, ensuring that the system is usable for students with disabilities, such as those who might need screen readers or other assistive technologies. The goal is to create an inclusive environment where all students can independently manage their housing needs. Feedback from users is regularly solicited to make iterative improvements to the system's design and functionality.

4. Maintainability and Support Requirements

- Maintainability involves keeping the DreamHouse system running smoothly and updating it with new features or fixes without significant downtime. This is crucial for ensuring that the system remains useful and effective long-term, especially as new housing laws or student needs arise. Support is equally important, providing users with help whenever they encounter problems or have questions. Effective support can significantly enhance user satisfaction and trust in the system. Efforts are continuously made to streamline support processes and reduce response times, making support as helpful and accessible as possible.

5. Compliance Requirements

- Compliance is about adhering to laws and regulations related to data protection and privacy, which is crucial for maintaining user trust. DreamHouse commits to complying with relevant data protection laws, such as those that protect personal information from being misused. For students, this means knowing that their

data won't be sold to third parties or used inappropriately. Compliance also involves regular reviews and updates to the system to ensure it meets all legal requirements, adapting to new laws as they come into effect to protect both the users and the system operators legally and ethically.

3.4.3 Others Requirement

1. Software Requirement and Justifications.

   a. Visual Studio Code

Chosen for Visual Studio Code coding due to its versatility and robust support for multiple programming languages, including PHP and JavaScript. Its extensive library of extensions and integration capabilities enhances development efficiency and simplifies code management.

   b. XAMPP

Utilized for its ability to set up a local testing server quickly, facilitating easy development and testing of applications with Apache, MySQL, PHP, and Perl, which is essential for iterative testing before deployment.

   c. MySQL with phpMyAdmin

Selected as the database management system because of its reliability and the widespread use of phpMyAdmin for database

administration. This combination supports complex queries and storage needs while providing a user-friendly interface for database management.

d. Laravel 10

The latest version of Laravel is used for its high scalability and robust features including easy routing, sessions, caching, and authentication which are critical for modern web application development.

e. PHP 8.2

This version is chosen for its improved performance and new language features that enhance the security and efficiency of the application.

f. Bootstrap

Employed for front-end development due to its responsive design templates and extensive component library that speeds up the UI development process and ensures a mobile-friendly interface.

g. Stripe

Integrated as the payment gateway to provide secure and flexible payment options. Stripe is known for its ease of integration, comprehensive security measures, and broad acceptance of different payment methods.

h. Google Maps

Used for integrating dynamic maps, which help users easily locate houses. Its reliable API supports various customizations and real time location tracking, enhancing the user experience.

i. Javascripts, CSS, HTML5

Fundamental technologies for web development, chosen for their stability and support across all web browsers. They ensure the application is interactive, stylistically consistent, and structurally sound.

2. Hardware Requirements and Justifications.

    a. MSI GF63 Thin Laptop with Intel Core i5-12450H, 8GB RAM, and 526GB ROM

    This laptop is chosen for its balance of performance and portability. The Intel Core i5 processor and 8GB RAM are sufficient for development tasks, including running local servers and databases via XAMPP. The 526GB storage offers ample space for all necessary software and project files. Its capabilities ensure that the development environment is fast and responsive, which is crucial for productivity and testing.

3. Others Requirements.

    a. Reliable Internet Connection

    A high-speed internet connection is essential to access cloud-based resources like Stripe for payment processing, Google Maps for location services, and to perform online testing and deployment of the application.

    b. Version Control System

Use of Git for version control, to manage changes to the project codebase efficiently, facilitating collaboration and maintaining the history of project developments.

3.5 Conclusion

In this chapter, we have thoroughly analyzed the requirements and specifications for the DreamHouse system, detailing both the functional and non functional aspects needed to enhance the house rental management experience for university students and landlords. We have explored the integration of advanced software and hardware to ensure the system is robust, user-friendly, and secure. Moving forward, the next steps involve the actual development and implementation of the system. This will include coding the application, setting up the database, and integrating all specified technologies such as Laravel, Spatie for backups, and Stripe for payment processing. Following development, rigorous testing will be conducted to ensure the system meets all specified requirements and functions efficiently under real world conditions.

CHAPTER 4: DESIGN

4.1 Introduction

This chapter presents the design phase of the DreamHouse system, focusing on converting the previously discussed requirements into a detailed system architecture and user interface design. It outlines the structural blueprint and interaction design necessary for the development of the system, integrating essential components such as the backend framework, database schema, and user interface elements.

The designs will demonstrate how the system's architecture supports its scalability, security, and performance goals while ensuring a user-friendly experience through an intuitive interface. This comprehensive approach sets the stage for the subsequent development and implementation phases, ensuring that the DreamHouse system aligns with the functional needs and expectations of its users.

4.2 Introductory preview to this chapter.

In this section of the report, we delve into the architectural framework of the DreamHouse system. The focus is on structuring the system in a way that ensures it is robust, scalable, and responsive to user needs. This chapter outlines the system's architecture using a combination of layered structures, frameworks, and tier-based organization.

Simplified Architecture View of The System

1. Layered Architecture

    - The DreamHouse system uses a layered architecture that includes a presentation layer for user interfaces, a business logic layer for processing data, and a data access layer for database interactions. This structure helps in organizing code and responsibilities clearly, making maintenance easier.

2. Framework-based Design (Using Laravel)

    - By adopting the Laravel framework, which follows the MVC (Model View-Controller) architecture, DreamHouse separates its operations into models (data handling), views (user interface), and controllers (application logic). This helps in efficient management and scaling of the application.

3. Three-tier Architecture

    - The application is structured into three tiers: the client tier for the frontend, the application tier for backend processing, and the data tier for managing the database. This separation enhances performance and security by distributing responsibilities.

4. Object-Oreinted Analysis and Design (OOAD)

    - Using UML diagrams, the system's object-oriented structure is meticulously planned. These diagrams help in visualizing and understanding the relationships and interactions within the application, ensuring a robust design framework.

4.3 Database Design

4.3.1 Conceptual Design

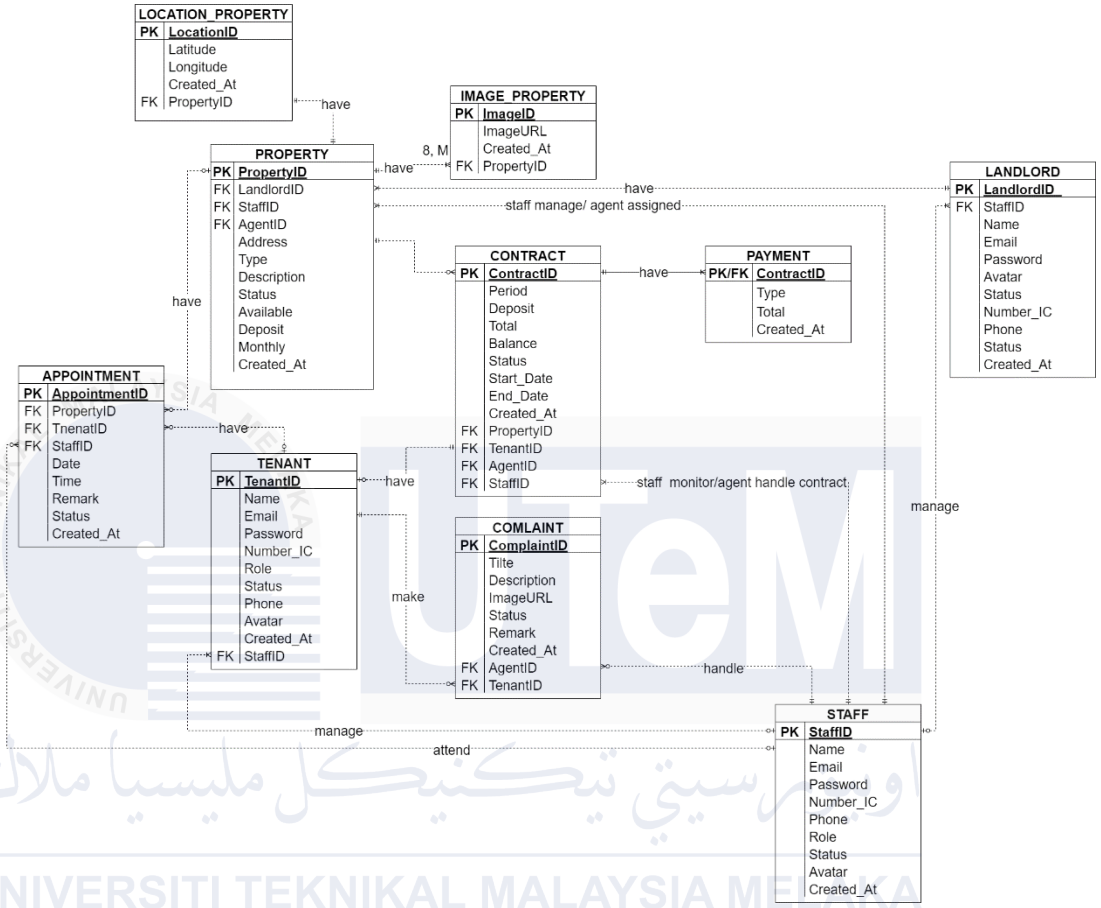


Figure 4.3-1 Entity Relationship Diagram (ERD)

Business Rule:

a) Agent manage one or many tenant and tenant will be managed by only one agent.

b) Staff manage one or many landlord and landlord will be managed by zero or only one staff.

c) Agent will assigned one or many property and property will assigned to only one agent.

d) Agent will have zero or many appointment and appointment will hanle by only one agent.

e) Agent handle one or many contract and each contract will handled by only one agent.

f) Agent can have zero or many complaint and each complaints will handled by only one agent.

g) Landlord can have one or many property and each property will have only one landlord.

h) Guest can have zero or many appointment and each appointment will have only one guest.

i) Tenant can have zero or many complaint and each compalaints will have only one tenant.

j) Tenant can have only one contract and each contract will assigned to only one tenant.

k) Contract can have multiple payments, and each payment will from only one contract.

l) Each property have minimum eight or many image_property and each image_property will assigned to only one property.

m) Each property must have only one location_property and each location_property will assigned to only one property.

n) Each property can have only one contract and each contract will assigned to only one property.

4.3.2 Logical Design

Data Dictionary

Table 4.3.2-1 Data Dictionary (staff)

Column	Data Type	Nullable	Default	Description	Relationships
StaffID	INT	No	AUTO_INCREMENT	Primary key, uniquely identifies a staff member.	Primary Key
Name	VARCHAR (255)	No	None	Staff member's name.	
Email	VARCHAR (255)	No	None	Staff email address.	
Password	VARCHAR (255)	No	None	Password for account access.	
Avatar	VARCHAR (255)	No	None	URL/link to the staff's avatar image.	
Phone	VARCHAR (255)	No	None	Staff phone number.	
Number_IC	VARCHAR (255)	No	None	Staff member's identification number.	
Role	INT	No	2	Representing staff role	
Status	INT	No	1	Status of the staff account.	
Created_At	TIMESTAMP	No	None	When the record was created.	

Table 4.3.2-2 Data Dictionary (landlord)

Column	Data Type	Nullable	Default	Description	Relationships
LandlordID	INT	No	AUTO_INCREMENT	Primary key, uniquely identifies a landlord.	Primary Key
StaffID	INT	Yes	NULL	Reference staff table.	Foreign Key (staff.StaffID) Role = 1
Name	VARCHAR (255)	No	None	Landlord name.	
Email	VARCHAR (255)	No	None	Landlord email address.	
Password	VARCHAR (255)	No	None	Password for account access.	
Avatar	VARCHAR (255)	No	None	URL/link to the landlord avatar image.	
Phone	VARCHAR (255)	No	None	Landlord phone number.	
Number_IC	VARCHAR (255)	No	None	Landlord identification number.	
Status	INT	No	1	Status of the landlord account.	
Created_At	TIMESTAMP	No	None	When the record was created.	

Table 4.3.2-3 Data Dictionary (tenant)

Column	Data Type	Nullable	Default	Descriptio n	Relationships
TenantID	INT	No	AUTO_INCREMENT	Primary key, uniquely identifies a tenant.	Primary Key
StaffID	INT	Yes	NULL	Reference staff table.	Foreign Key (staff.StaffID) Role = 1
Name	VARCHAR (255)	No	None	Tenant name.	
Email	VARCHAR (255)	No	None	Tenant email address.	
Password	VARCHAR (255)	No	None	Password for account access.	
Avatar	VARCHAR (255)	Yes	NULL	URL/link to the tenant avatar image.	
Phone	VARCHAR (255)	Yes	NULL	Tenant phone number.	
Number_IC	VARCHAR (255)	Yes	NULL	Tenant identificati on number.	
Role	INT	No	2	Representi ng tenant role (1tenant,2guest)	
Status	INT	No	1	Status of the tenant account.	
Created_At	TIMESTAMP	No	None	When the record was created.	

Table 4.3.2-4 Data Dictionary (property)

Column	Data Type	Nullable	Default	Description	Relationships
PropertyID	INT	No	AUTO_INCREMENT	Primary key, uniquely identifies a property.	Primary Key
LandlordID	INT	No	None	Reference landlord table.	Foreign Key (landlord.LandlordID)
StaffID	INT	Yes	NULL	Reference staff table.	Foreign Key (staff.StaffID) Role = 1
AgentID	INT	Yes	NULL	Reference staff table.	Foreign Key (staff.StaffID) Role = 2
Address	VARCHAR (255)	No	None	Physical address of the property.	
Type	INT	No	None	Type of property.	
Status	INT	No	3	Status of the property.	
Available	INT	No	2	Availability status.	
Deposit	DECIMAL (8,2)	No	None	Deposit amount required for renting.	
Monthly	DECIMAL (8,2)	No	None	Monthly rental price	

Description	VARCHAR (255	Yes	NULL	Description of the property.	
Created_At	TIMESTAMP	No	None	When record created.	

Table 4.3.2-5 Data Dictionary (image_property)

Column	Data Type	Nullable	Default	Description	Relationships
ImageID	INT	No	AUTO_INCREMENT	Primary key, uniquely identifies a image.	Primary Key
PropertyID	INT	No	None	Reference property table.	Foreign Key (property.PropertyID)
ImageURL	VARCHAR (255)	No	None	URL or link to the image file.	
Created_At	TIMESTAMP	No	None	When record created.	

Table 4.3.2-6 Data Dictionary (contract)

Column	Data Type	Nullable	Default	Description	Relationships
ContractID	INT	No	AUTO_INCREMENT	Primary key, uniquely identifies a contract.	Primary Key

Property ID	INT	No	None	Reference to the property involved in the contract.	Foreign Key (property.PropertyID)
StaffID	INT	Yes	NULL	Reference staff table.	Foreign Key (staff.StaffID) Role = 1
AgentID	INT	No	None	Reference staff table.	Foreign Key (staff.StaffID) Role = 2
TenantID	INT	No	None	Reference tenant table.	Foreign Key (tenant.TenantID)
Period	INT	No	None	Duration of the contract in months.	
Deposit	DECIMAL (8,2)	No	None	Deposit amount for proceed the contract.	
Total	DECIMAL (10,2)	No	None	Total rental payment.	
Balance	DECIMAL (10,2)	No	None	Remaining balance on the contract.	
Status	INT	No	2	Current status of the contract.	

Start_Date	DATE	No	None	Start date of the contract.	
End_Date	DATE	No	None	End date of the contract.	
Created_At	TIMESTAMP	No	None	When record created.	

Table 4.3.2-7 Data Dictionary (payment)

Column	Data Type	Nullable	Default	Description	Relationships
ContractID	INT	No	AUTO_INCREMENT	Reference to the associated contract.	Foreign Key (contract.ContractID)
Type	INT	No	None	Type of payment.	
Total	DECIMAL (10,2)	No	None	Total amount paid.	
Created_At	TIMESTAMP	No	None	When record created.	

Table 4.3.2-8 Data Dictionary (complaint)

Column	Data Type	Nullable	Default	Description	Relationships
ReportID	INT	No	AUTO_INCREMENT	Primary key, uniquely identifies a report.	Primary Key
AgentID	INT	No	None	Reference	Foreign Key (staff.StaffID) Role = 2

				st aff table.	
TenantID	INT	No	None	Referenc e te nant table.	Foreign Key (tenant.Tenant ID)
Title	INT	No	None	Title or brief descripti on of the complain t.	
Descripti on	VARCHA R (255)	No	None	Detailed descripti on of the	
				complain t.	
Remark	VARCHA R (255)	Yes	Null	Addition al remarks on the complain t.	
Status	INT	No	2	Current status of the complain t.	
ImageU RL	VARCHA R (255)	No	None	URL/link to an image related to the complain t.	
Created_ At	TIMESTA MP	No	None	When record created.	

Table 4.3.2-9 Data Dictionary (location_property)

Column	Data Type	Nullable	Default	Description	Relationships
LocationID	INT	No	AUTO_INCREMENT	Primary key, uniquely identifies a location.	Primary Key
PropertyID	INT	No	None	Reference to the property.	Foreign Key (property.PropertyID)
Latitude	Decimal (10,8)	No	None	Geographical latitude of the property.	
Longitude	Decimal (11,8)	No	None	Geographical longitude of the property.	
Created_At	TIMESTAMP	No	None	When record created.	

Table 4.3.2-10 Data Dictionary (appointment)

Column	Data Type	Nullable	Default	Description	Relationships
AppointmentID	INT	No	AUTO_INCREMENT	Primary key, uniquely identifies a appointment.	Primary Key
PropertyID	INT	No	None	Reference to the property involved in the contract.	Foreign Key (property.PropertyID)

AgentID	INT	No	None	Reference staff table.	Foreign Key (staff.StaffID) Role = 2
TenantID	INT	No	None	Reference tenant table.	Foreign Key (tenant.TenantID)
Date	DATE	No	None	Date of the appointment.	
Time	Time	No	None	Time for appointment	
Remark	VARCHAR (255)	Yes	NULL	Optional remark appointment.	
Status	INT	No	2	Current status of appointment.	
Created_At	TIMESTAMP	No	None	When record created.	

4.3.3 Physical Design

1. Selection of DBMS

    a. For the DreamHouse project, MySQL has been selected as the Database Management System (DBMS), accessed via phpMyAdmin, a popular free and open source administration tool for MySQL and MariaDB. This combination is widely used in web development due to its user-friendly web interface, ease of setup, and strong community support. MySQL is ideal for web applications and offers excellent performance, comprehensive feature sets, and flexibility, making it suitable for

handling complex data structures and relationships needed in house rental management applications.

2. Usage of stored procedures, triggers, and other related database objects.

    a. Stored Procedures

```
DELIMITER $$

CREATE PROCEDURE InsertProperty (
    IN landlordId BIGINT,
    IN staffId BIGINT,
    IN agentId BIGINT,
    IN address VARCHAR(255),
    IN type INT,
    IN status INT,
    IN available INT,
    IN deposit DECIMAL(6,2),
    IN monthly DECIMAL(6,2),
    IN description VARCHAR(255)
)
BEGIN
    INSERT INTO properties (landlord_id, staff_id, agent_id, address, type, status, available, deposit, monthly, description, created_at, updated_at)
    VALUES (landlordId, staffId, agentId, address, type, status, available, deposit, monthly, description, NOW(), NOW());

    SELECT LAST_INSERT_ID() AS propertyId;
END$$

DELIMITER ;
```

Figure 4.3-2 SQL Stored Procedure (InsertProperty)

```
DELIMITER $$

CREATE PROCEDURE InsertContract (
    IN propertyId BIGINT,
    IN staffId BIGINT,
    IN agentId BIGINT,
    IN tenantId BIGINT,
    IN period INT,
    IN deposit DECIMAL(6,2),
    IN total DECIMAL(6,2),
    IN balance DECIMAL(10,2),
    IN status INT,
    IN startDate DATE,
    IN endDate DATE
)
BEGIN
    INSERT INTO contracts (property_id, staff_id, agent_id, tenant_id, period, deposit, total, balance, status, start_date, end_date, created_at, updated_at)
    VALUES (propertyId, staffId, agentId, tenantId, period, deposit, total, balance, status, startDate, endDate, NOW(), NOW());

    SELECT LAST_INSERT_ID() AS contractId;
END$$

DELIMITER ;
```

Figure 4.3-3 Stored Procedure (InsertContract)

b. Update Procedure

```
DELIMITER $$

CREATE PROCEDURE UpdateProperty (
    IN propertyId BIGINT,
    IN landlordId BIGINT,
    IN address VARCHAR(255),
    IN type INT,
    IN deposit DECIMAL(6,2),
    IN monthly DECIMAL(6,2),
    IN description VARCHAR(255)
)
BEGIN
    UPDATE properties
    SET landlord_id = landlordId,
        address = address,
        type = type,
        deposit = deposit,
        monthly = monthly,
        description = description,
        updated_at = NOW()
    WHERE id = propertyId;
END$$

DELIMITER ;
```

Figure 4.3-4 Update Procedure (UpdateProperty)

```
DELIMITER $$

CREATE PROCEDURE UpdateContractDetails (
    IN contractId BIGINT,
    IN newPropertyId BIGINT,
    IN agentId BIGINT,
    IN newTenantId BIGINT,
    IN period INT,
    IN balance DECIMAL(10,2),
    IN startDate DATE,
    IN endDate DATE,
    IN newTenantIcNumber VARCHAR(255)
)
BEGIN
    DECLARE oldPropertyId BIGINT;
    DECLARE oldTenantId BIGINT;

    -- Get the old property_id and tenant_id from the contract
    SELECT property_id, tenant_id INTO oldPropertyId, oldTenantId FROM contracts WHERE id = contractId;

    -- Update the old property as available
    UPDATE properties SET available = 1 WHERE id = oldPropertyId;

    -- Update the old tenant's IC number and role
    UPDATE tenants SET number_ic = NULL, role = 2 WHERE id = oldTenantId;

    -- Update the contract with new details
    UPDATE contracts
    SET property_id = newPropertyId,
        agent_id = agentId,
        tenant_id = newTenantId,
        period = period,
        balance = balance,
        start_date = startDate,
        end_date = endDate,
        updated_at = NOW()
    WHERE id = contractId;

    -- Mark the new property as unavailable
    UPDATE properties SET available = 2 WHERE id = newPropertyId;

    -- Update the new tenant's role and IC number
    UPDATE tenants SET number_ic = newTenantIcNumber, role = 1 WHERE id = newTenantId;
END$$

DELIMITER ;
```

Figure 4.3-5 Update Procedure (UpdateContractDetails)

c. Trigger

```
DELIMITER $$

CREATE TRIGGER update_contracts AFTER INSERT ON payments
FOR EACH ROW
BEGIN
    DECLARE deposit_value DECIMAL(10,2);
    DECLARE property_deposit DECIMAL(10,2);

    -- Get the total deposit for the contract
    SELECT deposit INTO deposit_value FROM contracts WHERE id = NEW.contract_id;

    -- Get the property's required deposit
    SELECT deposit INTO property_deposit FROM properties WHERE id = (SELECT property_id FROM contracts WHERE id = NEW.contract_id);

    IF NEW.type = 1 THEN
        -- Update deposit in contracts table
        UPDATE contracts
        SET deposit = deposit_value + NEW.total
        WHERE id = NEW.contract_id;

        -- Check if deposit matches the property's required deposit
        IF (deposit_value + NEW.total) = property_deposit THEN
            -- Set status attribute in contracts table based on contract_id
            UPDATE contracts
            SET status = 1
            WHERE id = NEW.contract_id;
        END IF;

    ELSEIF NEW.type = 2 THEN
        -- Update total and balance in contracts table
        UPDATE contracts
        SET total = total + NEW.total,
            balance = balance - NEW.total
        WHERE id = NEW.contract_id;
    END IF;
END$$

DELIMITER ;
```

Figure 4.3-6 SQL Trigger (update_contracts)

d. Function


```
DELIMITER $$

CREATE FUNCTION CheckDeposit(contractId BIGINT, newTotal DECIMAL(10,2)) RETURNS BOOLEAN
BEGIN
    DECLARE currentDeposit DECIMAL(10,2);
    DECLARE propertyDeposit DECIMAL(10,2);
    DECLARE depositMatches BOOLEAN;

    -- Get the current deposit for the contract
    SELECT deposit INTO currentDeposit FROM contracts WHERE id = contractId;

    -- Get the property's required deposit
    SELECT deposit INTO propertyDeposit FROM properties WHERE id = (SELECT property_id FROM contracts WHERE id = contractId);

    -- Check if the updated deposit matches the property's required deposit
    SET depositMatches = ((currentDeposit + newTotal) = propertyDeposit);

    RETURN depositMatches;
END$$

DELIMITER ;
```

Figure 4.3-7 SQL Function (CheckDeposit)

e. Events

```
CREATE EVENT update_contract_status
ON SCHEDULE EVERY 1 DAY
STARTS (TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY)
DO
UPDATE contracts
SET status = 3
WHERE end_date < CURRENT_DATE AND status = 1;
```

Figure 4.3-8 SQL Event (update_contract_status)

```
CREATE EVENT update_status_for_week_old_contracts
ON SCHEDULE EVERY 1 DAY
STARTS (TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY)
DO
UPDATE contracts
SET status = 4
WHERE created_at <= CURRENT_DATE - INTERVAL 7 DAY AND status = 2;
```

Figure 4.3-9 Event (update_status_for_week_old_contracts)

f. SQL Query (Join, Selection & Projection, Ordering and Limiting)


```
SELECT
    s.id,
    s.name,
    s.number_ic,
    s.phone,
    COUNT(DISTINCT p.id) AS assigned_properties_count,
    COUNT(DISTINCT c.id) AS handled_contracts_count
FROM
    staff s
LEFT JOIN
    properties p ON s.id = p.agent_id AND p.status != 5
LEFT JOIN
    contracts c ON s.id = c.agent_id AND c.status != 4
WHERE
    s.role = 2
GROUP BY
    s.id;
```

id	name	number_ic	phone	assigned_properties_count	handled_contracts_count
1	Azhar Hairi	010604030654	0135267674	2	0
3	Muhammad Aiman	010503020432	01162441617	4	3
4	Aliff Najmi	010705020345	01121652347	3	1
6	Kery Sabrina	010506040234	01162441618	0	0
7	Zarine Andie	010708090807	01980770806	3	3

Figure 4.3-10 SQL Query (LandlordList)

g. SQL Query (Aggregation, Grouping)

```
SELECT
  YEAR(created_at) as SelectedYear,
  SUM(CASE WHEN table_name = 'payments' THEN total ELSE 0 END) AS TotalPayments,
  COUNT(CASE WHEN table_name = 'properties' AND status != 5 THEN 1 END) AS TotalProperties,
  COUNT(CASE WHEN table_name = 'contracts' AND status NOT IN (4, 5) THEN 1 END) AS TotalContracts,
  COUNT(CASE WHEN table_name = 'staff' AND status = 1 THEN 1 END) AS TotalStaffs,
  COUNT(CASE WHEN table_name = 'landlords' AND status = 1 THEN 1 END) AS TotalLandlords,
  COUNT(DISTINCT CASE WHEN table_name = 'contracts' AND status NOT IN (4, 5) THEN tenant_id END) AS TotalTenants,
  SUM(CASE WHEN table_name = 'payments' THEN total ELSE 0 END) AS MonthlyIncome
FROM (
  SELECT 'payments' as table_name, total, created_at, NULL as status, NULL as tenant_id FROM payments
  UNION ALL
  SELECT 'properties', NULL as total, created_at, status, NULL as tenant_id FROM properties
  UNION ALL
  SELECT 'contracts', NULL as total, created_at, status, tenant_id FROM contracts
  UNION ALL
  SELECT 'staff', NULL as total, created_at, status, NULL as tenant_id FROM staff
  UNION ALL
  SELECT 'landlords', NULL as total, created_at, status, NULL as tenant_id FROM landlords
) AS combined
WHERE YEAR(created_at) = 2024
GROUP BY YEAR(created_at), MONTH(created_at);
```

SelectedYear	TotalPayments	TotalProperties	TotalContracts	TotalStaffs	TotalLandlords	TotalTenants	MonthlyIncome
2024	3700.00	10	1	4	5	1	3700.00
2024	48770.00	16	6	21	14	5	48770.00

Figure 4.3-11 SQL Query (Chart-monthly,year)

h. Automatic Backup (Using Spatie and backup database to Drive – Every Minute)

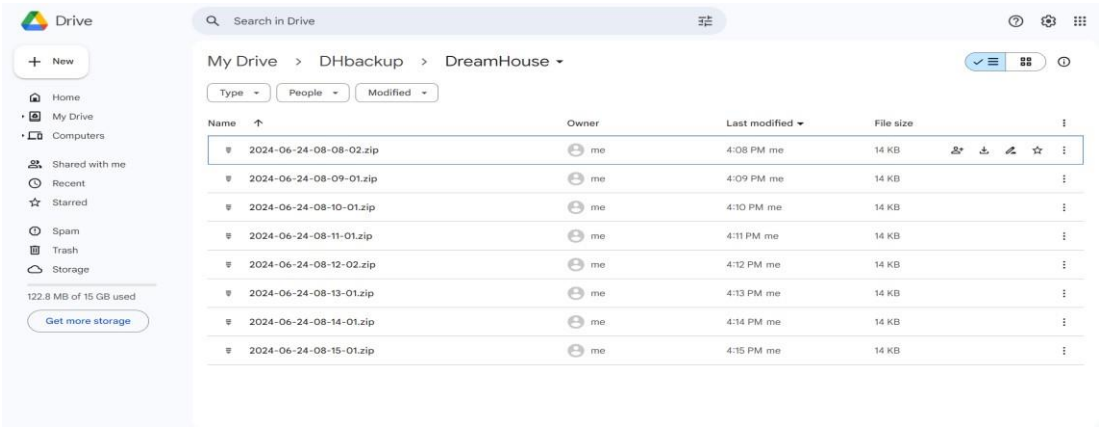


Figure 4.3-12 Automation(BackupDatabase-saved in Drive)

4.4 Graphical User Interface (GUI) Design

1. Guest Module

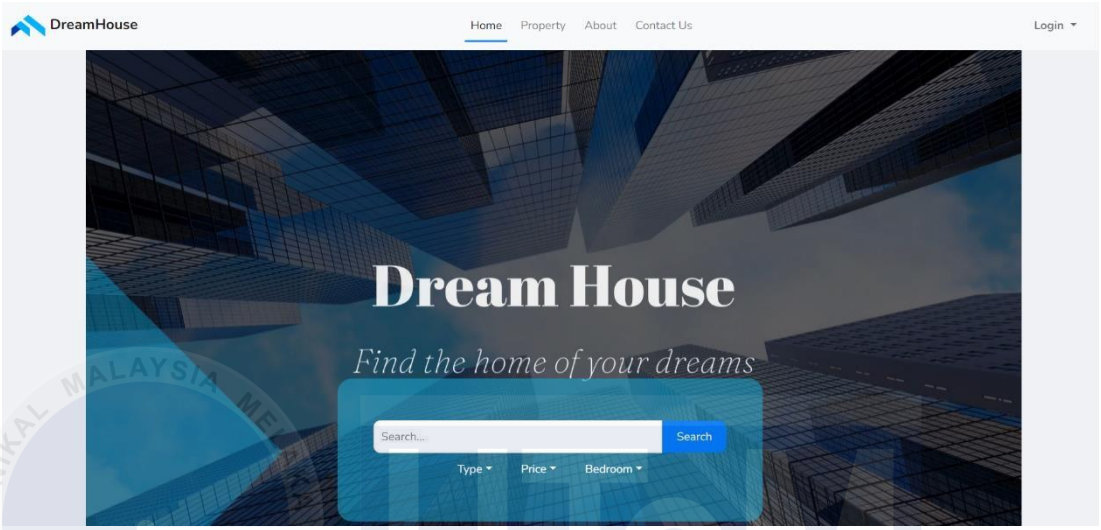


Figure 4.4-1 HomePage (Guest)

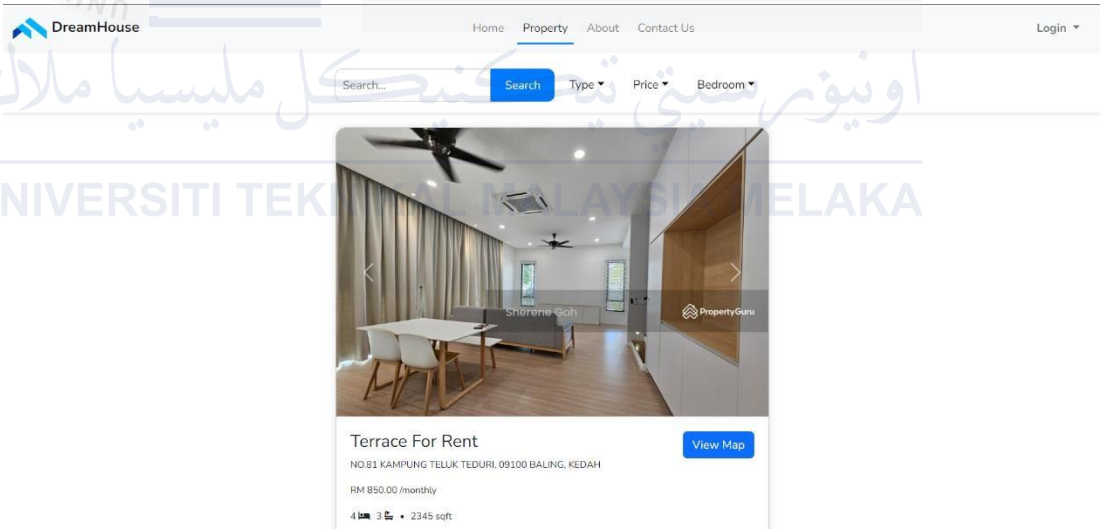


Figure 4.4-2 Property Listing (Guest)

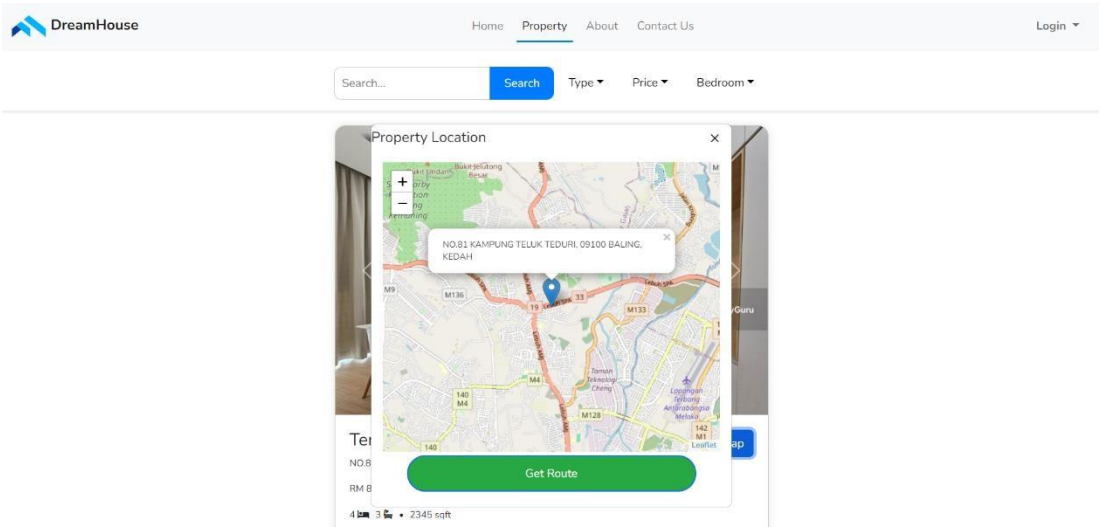


Figure 4.4-3 Location of The Property (Guest)

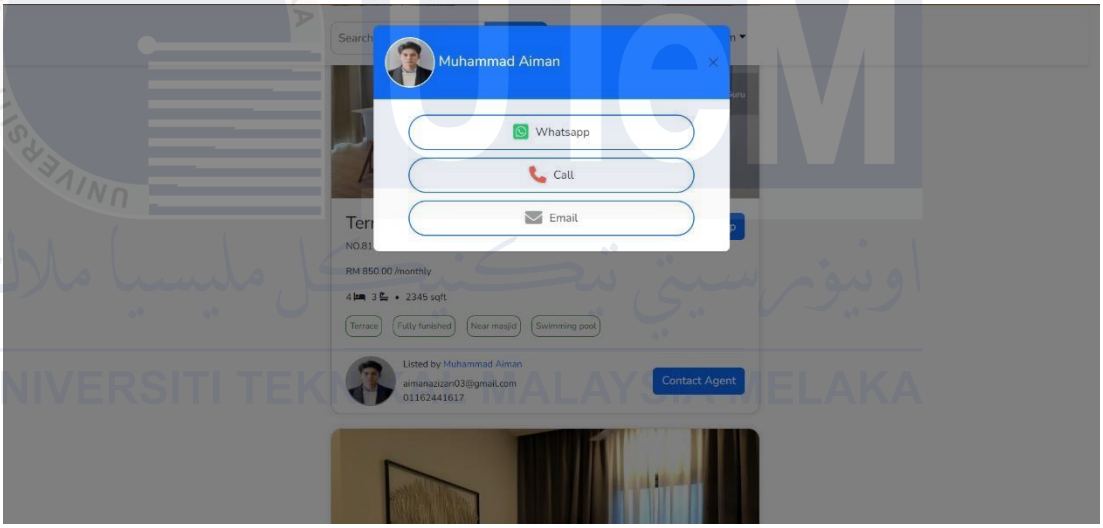


Figure 4.4-4 Contact Agent (Guest)

2. Tenant Module

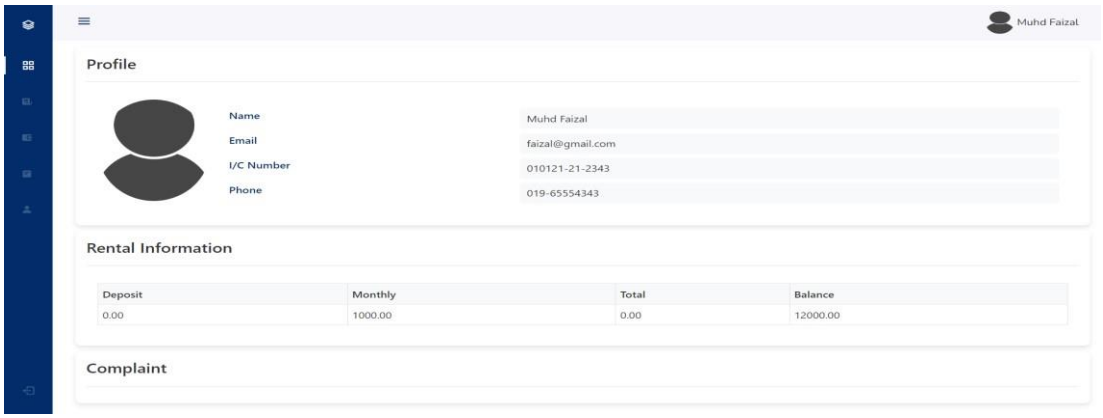


Figure 4.4-5 Dashboard (Tenant)

Figure 4.4-6 Contract Detail (Tenant)

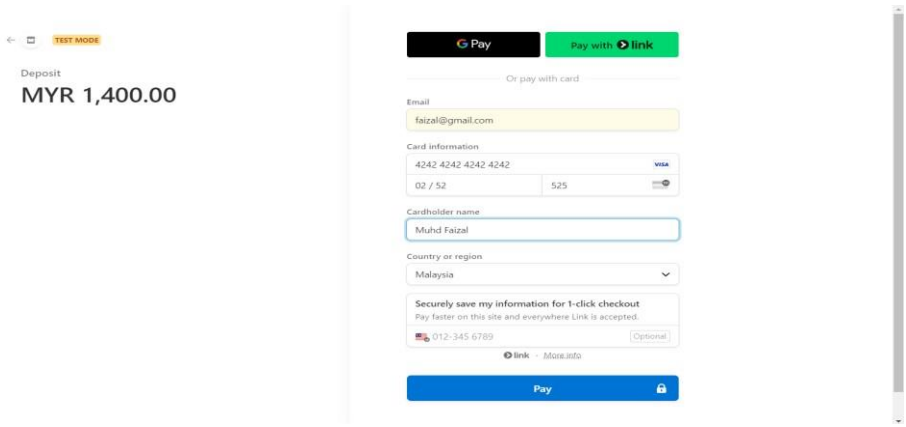


Figure 4.4-7 Payment (Tenant)

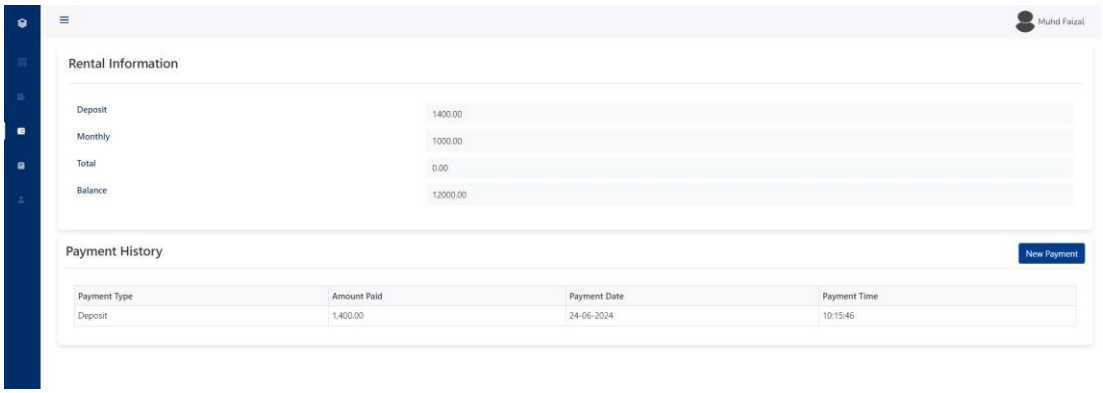


Figure 4.4-8 Payment Detail (Tenant)

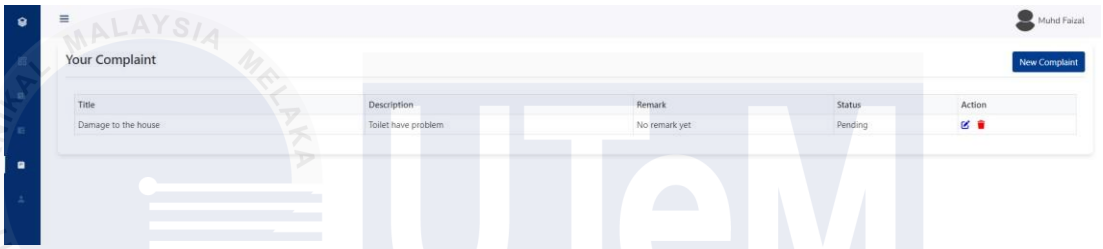


Figure 4.4-9 Complaint Detail (Tenant)

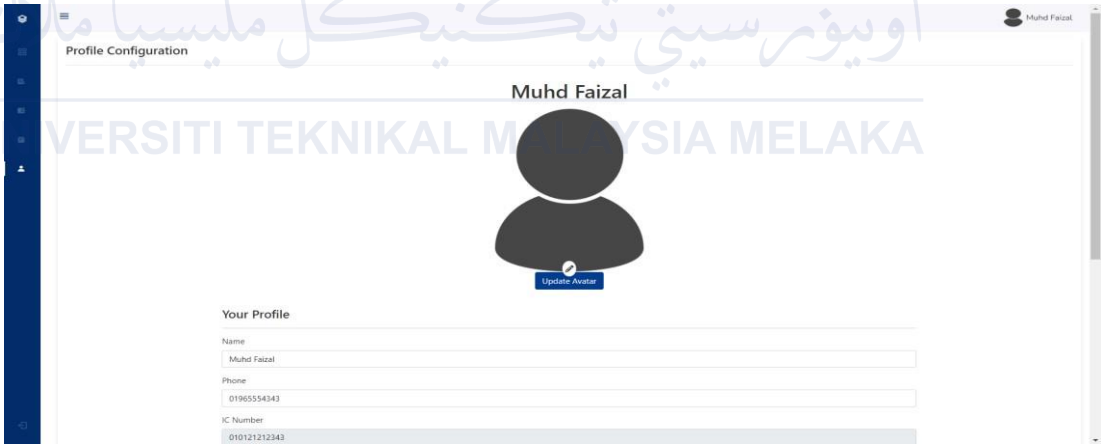


Figure 4.4-10 Profile Configuration (Tenant)

3. Landlord Module

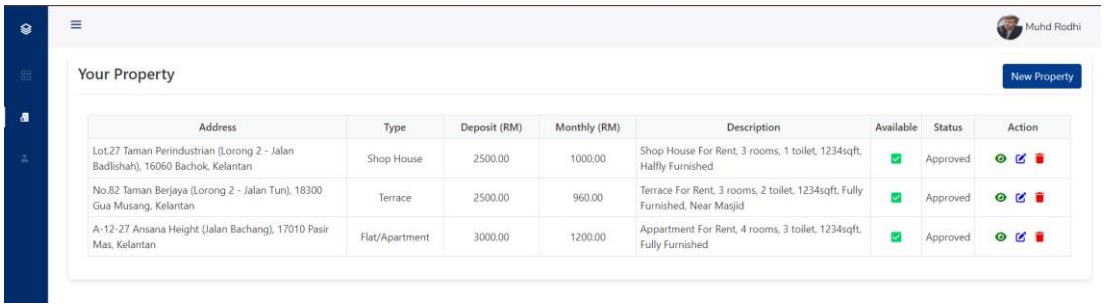


Figure 4.4-11 Property List (Landlord)

Figure 4.4-12 Register Property (Landlord)

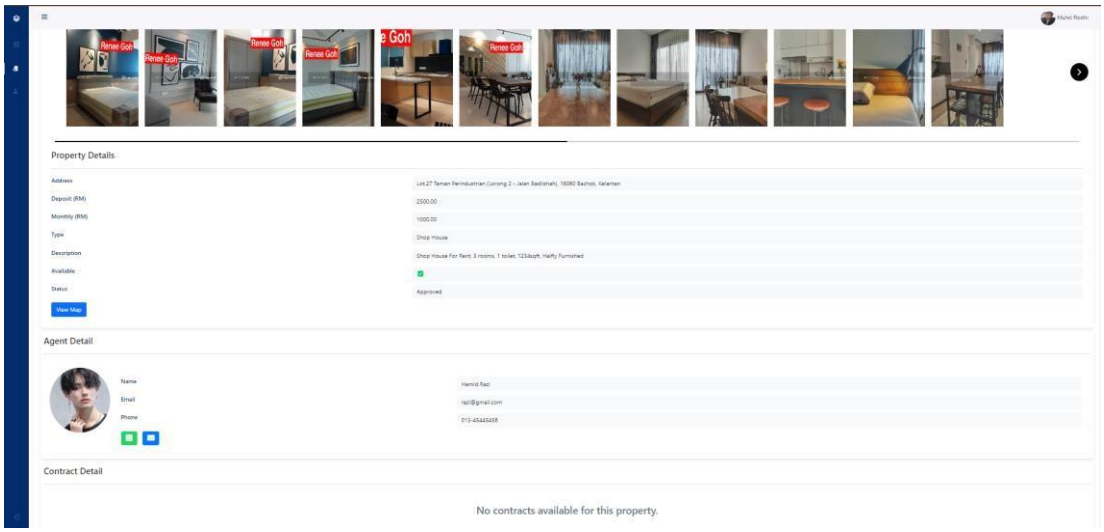


Figure 4.4-13 Property Detail (Landlord)

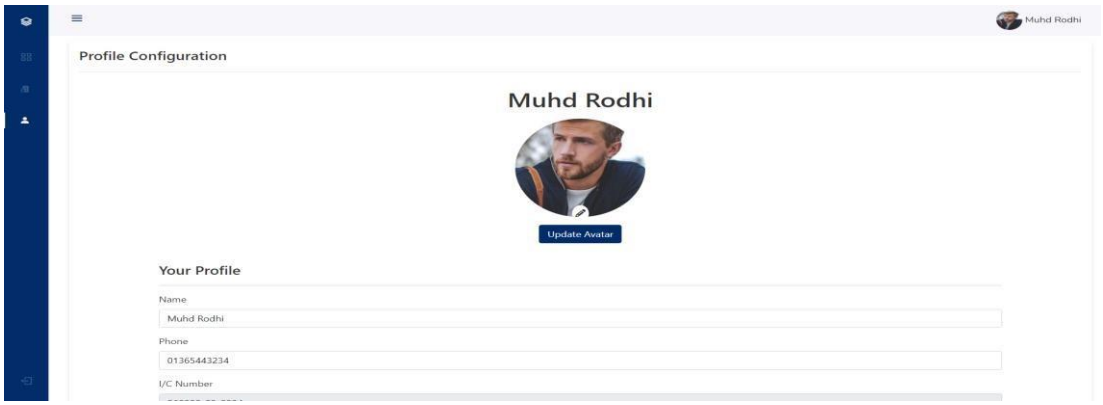


Figure 4.4-14 Profile Comfiguration (Landlord)

4. Agent Module

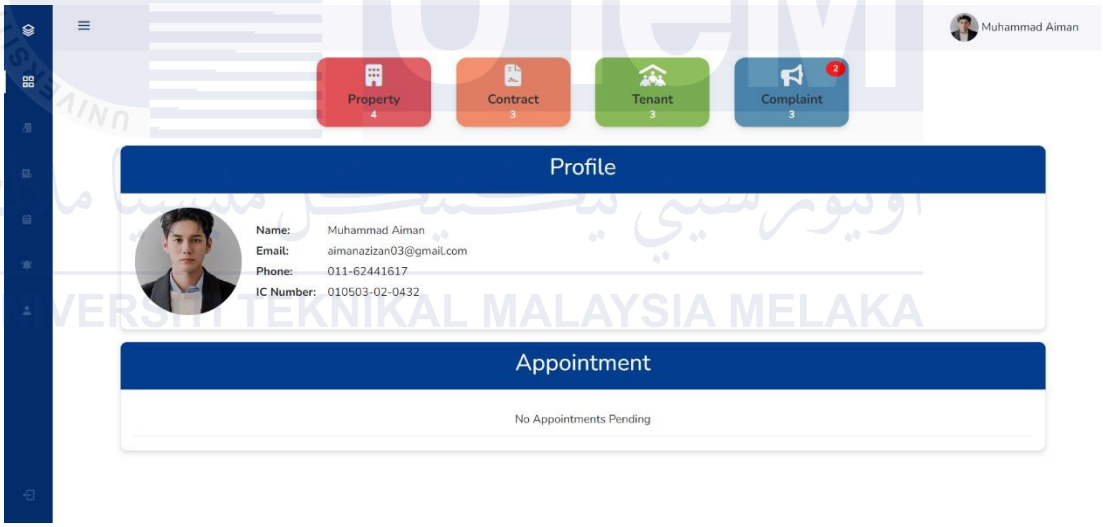


Figure 4.4-15 Dashboard (Agent)

Figure 4.4-16 Contract List (Agent)

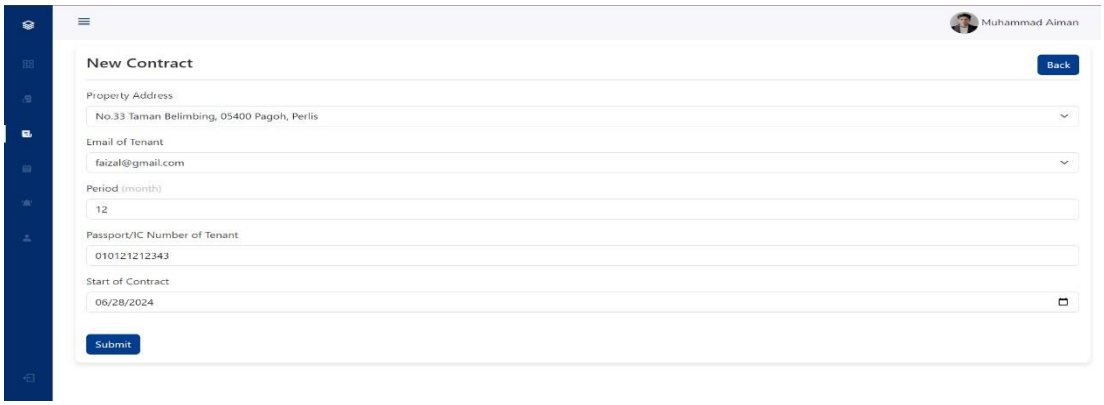


Figure 4.4-17 Create New Contract (Agent)

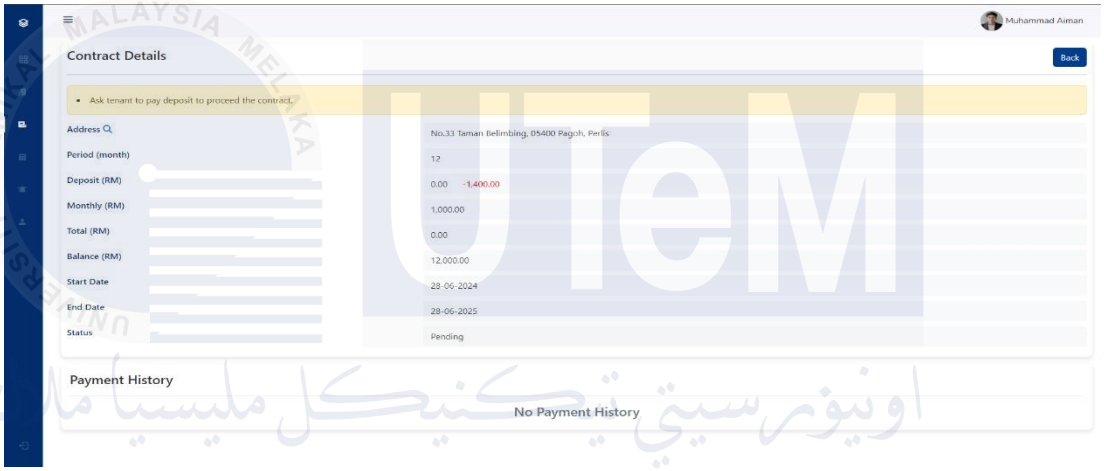


Figure 4.4-18 Contract Detail (Agent)

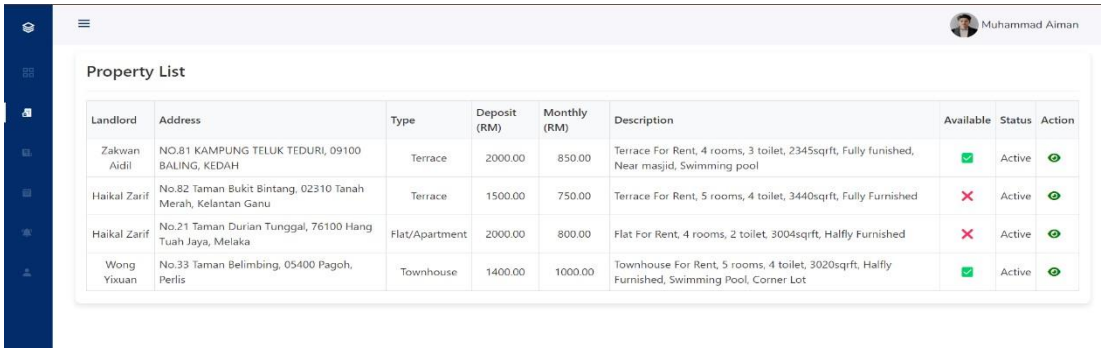


Figure 4.4-19 Property Assigned List (Agent)

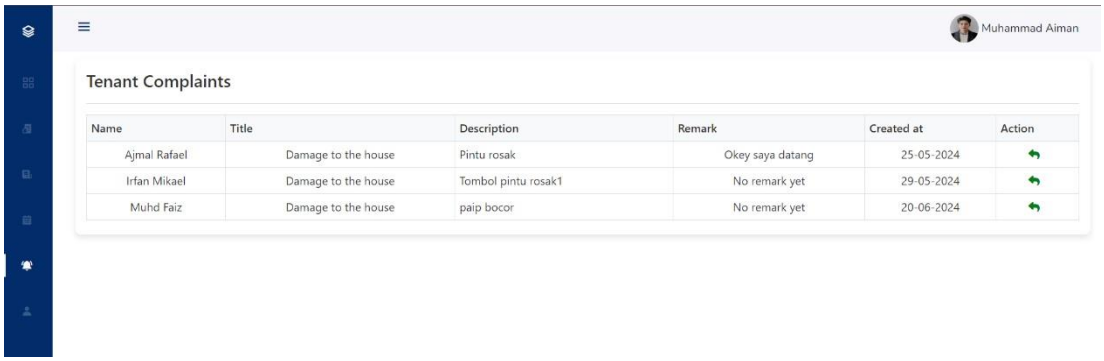


Figure 4.4-20 Tenant Complaint (Agent)

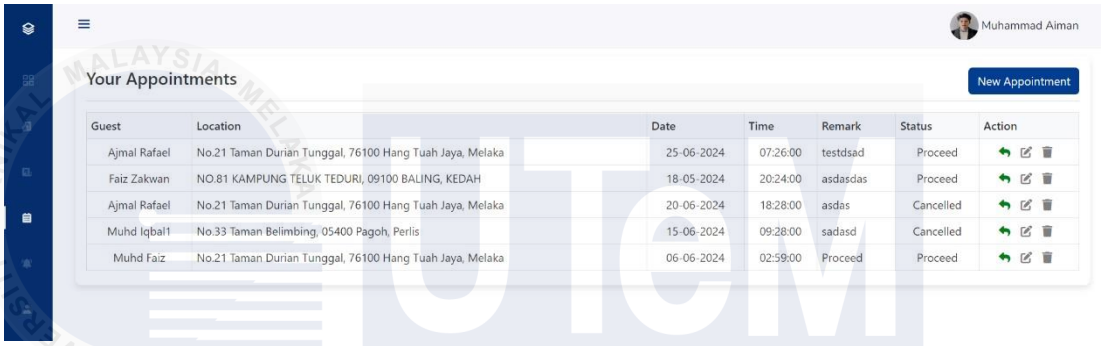


Figure 4.4-21 Appointments List (Agent)

Figure 4.4-22 Profile Configuration

5. Staff Module

Figure 4.4-23 Dashboard (Staff)

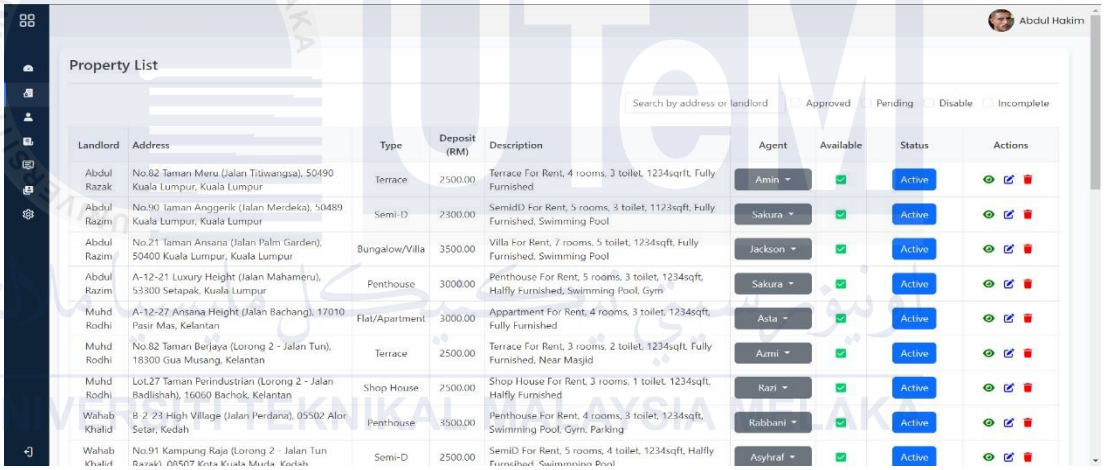


Figure 4.4-24 Property List (Staff)

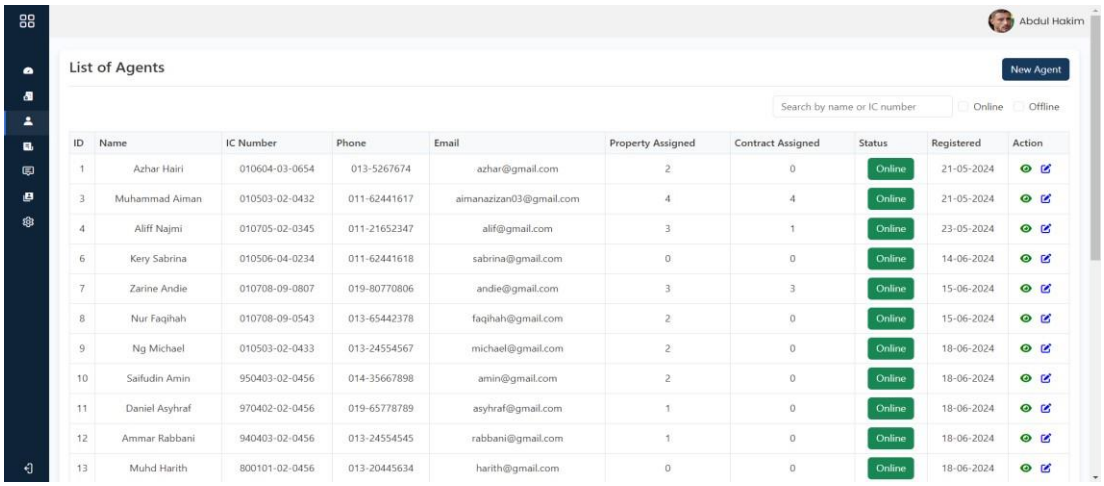


Figure 4.4-25 List of Agents (Staff)

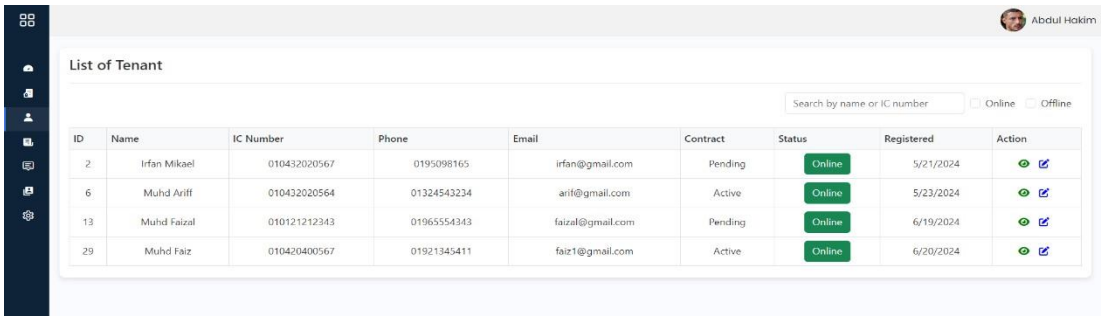


Figure 4.4-26 List Of Tenants

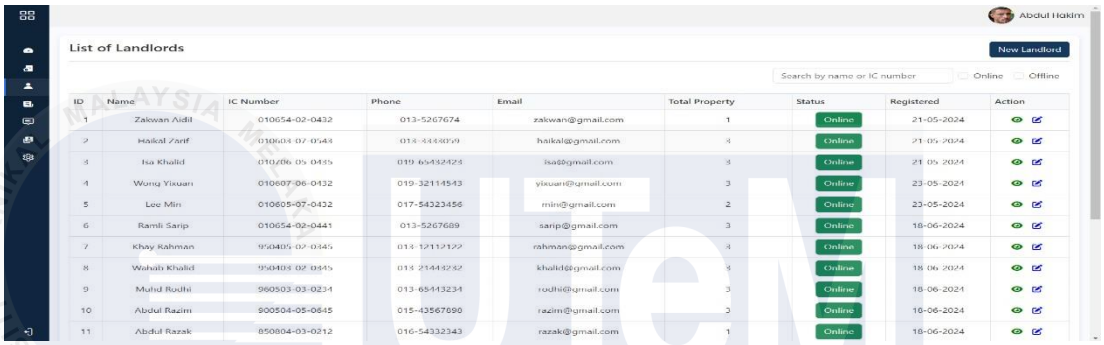


Figure 4.4-27 List of Landlords

Figure 4.4-28 Tenant Complaints (Staff)

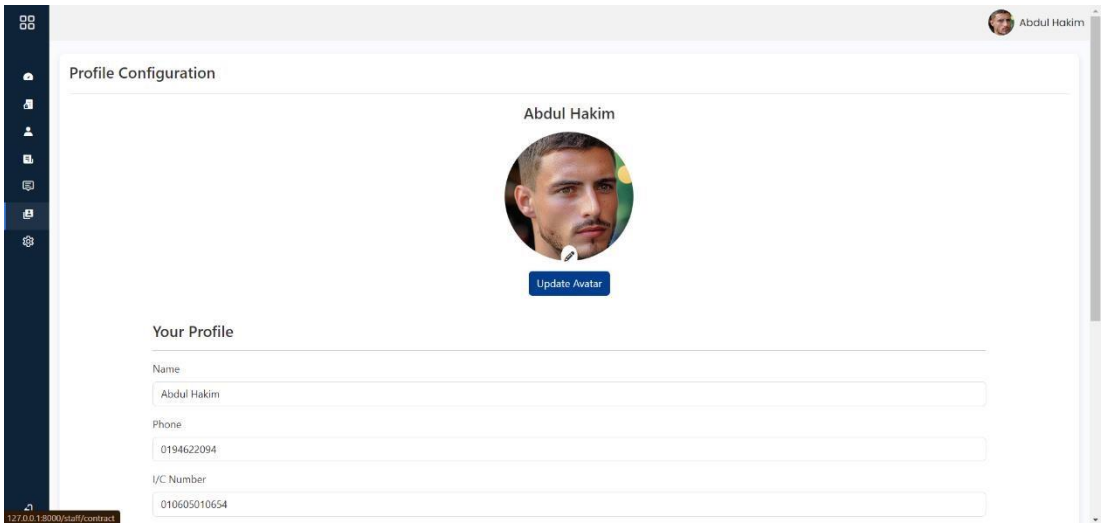


Figure 4.4-29 Profile Configuration (Staff)

4.5 Conclusion

In this chapter, we have systematically explored the architectural and design components crucial for the successful implementation of the DreamHouse rental management system. From the structured presentation of the system's architecture in various models and frameworks to the detailed phases of database design—from conceptual mapping with ERDs and UML diagrams to the intricacies of logical and physical database constructions we have laid a robust foundation for a technologically advanced system. Additionally, the design of the graphical user interface has been tailored to ensure intuitive navigation and optimal user interaction based on the functional and non-functional requirements previously outlined. This comprehensive approach not only anticipates the technical demands of the system but also addresses user convenience and system efficiency, paving the way for a more dynamic and user friendly rental management experience.

CHAPTER 5: IMPLEMENTATION

5.1 Introduction

For my House Rental Management System (DreamHouse), I've selected phpMyAdmin as my database management tool and Laravel as the framework. phpMyAdmin's user-friendly interface makes it easy to manage and visualize database structures, tables, and data. Its intuitive design allows me to efficiently create, modify, and query databases without requiring extensive SQL knowledge.

Laravel, on the other hand, provides a robust MVC (Model-View-Controller) architecture, which promotes code organization and maintainability. By separating concerns into distinct layers, Laravel simplifies development and makes it easier to manage complex applications. Its built-in features and tools, such as Eloquent ORM and Blade templating, further enhance productivity and help me optimize the MVC functions for developing DreamHouse.

5.2 Software Development Environment Setup

System Environment Setup:

1. Installation

    i. Download and install latest version of PHP and MySQL from their respectives website. Choose appropriate installers for my operating system which Windows.

ii. Configure PHP which ensure PHP is configured to use the installed MySQL extension.

iii. Configure MySQL that actually setup MySQL with a secure password and configure any necessary setting.

2. Laravel Installation

i. Use composer, a dependency manager for PHP, to install Laravel.

ii. Using GitBash:

Database Environment Setup

1. Database Creation

i. Install XAMPP as host for server hosting.

ii. Use phpMyAdmin to create a new database and named “dreamhouse”.

2. Database Objects

i. Using SQL queries, to create the staff, landlord, tenant, property, image_property, contract, payment, complaint, location_property, appointment tables to store my data.

Programming Technique

1. MVC Architecture

(MVC) architecture, which helped in separating the application's logic from its user interface. This made the code more organized, maintainable, and scalable. Figure 5.2-1 shows that MVC technique that implemented for manage properties table

```php
class Property extends Model
{
    use HasFactory;

    protected $table = 'properties';

    protected $fillable = [
        'address',
        'type',
        'deposit',
        'monthly',
        'description',
        'landlord_id',
    ];
```

```blade
<table class="details-table">
    <thead>
        <tr>
            <th style="text-align: center;">Address</th>
            <th style="text-align: center; width: 150px;">Type</th>
            <th style="width: 150px; text-align: center;">Deposit (RM)</th>
            <th style="width: 150px; text-align: center;">Monthly (RM)</th>
            <th style="text-align: center;">Description</th>
            <th style="text-align: center;">Available</th>
            <th style="text-align: center;">Status</th>
            <th style="width: 150px; text-align: center;">Action</th>
        </tr>
    </thead>
    <tbody id="propertyTableBody">
        @foreach ($properties as $property)
        <tr>
            <td>{{$property->address}}</td>
            <td style="text-align: center;">
            @if ($property->type == 1)···
            @endif
            </td>
            <td style="text-align: center;">{{$property->deposit}}</td>
            <td style="text-align: center;">{{$property->monthly}}</td>
            <td>{{$property->description}}</td>
            <td style="text-align: center;">
                @if ($property->available == 1)···
                @endif
            </td>
            <td style="text-align: center;">
            @if ($property->status == 3)···
            @endif
```

```php
class PropertyController extends Controller
{

    //Landlord
    public function lview()
    {
        $properties = Property::where('landlord_id', Auth::guard('landlord')->user()->id)
            ->where('status', '!=', 5)
            ->get();
        return view('landlord.LProperty', compact('properties'));
    }
```

Figure 5.2-1 MVC Implementation

2. Blade Templating Engine

Blade, Laravel's templating engine, was used to build the front-end views for the system. Blade's lightweight templates allow for reusability and easy integration of dynamic content, which was particularly useful for creating user-specific interfaces (e.g., guest, tenant, landlord). Figure 5.2-2 shows that interface with layout using blade.

```
<body id="body-pd">
    @include('layouts.landlord-nav')
    <!--Container Main start-->

    <div>
        {{-- Page content here bruh!!--}}
        @yield('content')
    </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js"
        integrity="sha384-ENjdO4Dr2bkBIFxQpeoTz1HIcje39Wm4jDKdf19U8gI4ddQ3GYNS7NTKfAdVQSZe"
        crossorigin="anonymous"></script>

    <script>···
    </script>
    <script>···
```

```
public function lview()
{
    $properties = Property::where('landlord_id', Auth::guard('landlord')->user()->id)
        ->where('status', '!=', 5)
        ->get();
    return view('landlord.LProperty', compact('properties'));
}
```

```
@extends('layouts.landlord-layout')
@section('title', 'DreamHouse • Property')
@section('content')

<style>···
</style>

<div class="notice" style="margin-top: 75px; overflow: hidden;">···
</div>

<div class="details-container">···
</div>

<script>···
</script>

@endsection
```

Figure 5.2-2 Blade Template

3. Middleware

Laravel's middleware was utilized for handling requests and managing user roles and permissions. This helped ensure that different user roles (guest, tenant, landlord, agent, admin) had access to only the functionalities specific to their roles. Figure 5.2-3 shows how middleware are used for authentication.

```php
if(Auth::guard('tenant')->attempt($credentials)){

    $user = Tenant::where('email',$request->input('email'))->first();
    Auth::guard('tenant')->login($user);
    if(Auth::guard('tenant')->user()->role == 1){
        return redirect()->route('tenant_dashboard')->with('success','Login Successful');
    }else{
        return redirect()->route('welcome')->with('success','Login Successful');
    }
}else{
    return redirect()->route('tenant_login')->with('error','Login unsuccessful');
}
```

```php
if (Auth::guard('staff')->attempt($credentials)) {
    $user = Staff::where('email', $request->input('email'))->first();
    Auth::guard('staff')->login($user);

    if (Auth::guard('staff')->user()->role == 1) {
        return redirect()->route('staff_dashboard')->with('success', 'Login Successful');
    } else {
        return redirect()->route('agent_dashboard')->with('success', 'Login Successful');
    }
} else {
    return redirect()->route('staff_login')->with('error', 'Login unsuccessful');
}
```

Figure 5.2-3 Middleware Authentication

4. Routing

Laravel's routing system was employed to manage all user requests and direct them to the appropriate controllers. The routing system provided flexibility and control over the URLs in the system, ensuring that the web application was both SEO-friendly and easy to navigate. Figure 5.2-4 shows that the route for each controller to perform action.

```php
Route::get('agent/profile', [StaffController::class, 'agentprofile']);
Route::post('agent/profile/update/{id}', [StaffController::class, 'agent_update_profile']);
Route::post('agent/profile/avatar/{id}', [StaffController::class, 'agent_update_avatar']);
Route::post('agent/profile/password/{id}', [StaffController::class, 'agent_update_password']);
```

```
class="container text-center mb-3">
<form id="avatar-form" action="{{ url('agent/profile/avatar/'.Auth::guard('staff')->user()->id.'') }}" enctype="multipart/form-data" method="POST">
    @csrf
    <h1 class="mb-3">{{ Auth::guard('staff')->user()->name }}</h1>
    @if(Auth::guard('staff')->check())
    <div class="avatar-container position-relative">
        <label for="avatar-upload" class="edit-icon"><i class="fas fa-pencil-alt"></i></label>
        <img src="{{ asset(Auth::guard('staff')->user()->avatar) }}" alt="Avatar" class="rounded-circle circle-image mb-3" id="avatar-image">
        <input type="file" id="avatar-upload" name="avatar" style="display: none;" onchange="displaySelectedImage(event)">
    </div>
    <div>
        @error('avatar') <span class="text-danger">{{ $message }}</span>@enderror
    </div>
    @endif
    <button type="submit" class="btn" style="background-color: #023d90; color: white;">Update Avatar</button>
</form>
```

Figure 5.2-4 Route of Controller

5. Validation and Form Handling

Laravel's built-in validation and form handling mechanisms were used to ensure data integrity and security. This was especially important for user registration or property submissions. Figure 5.2-5 shows that validation handling for each input that must meet specific criteria.

```
$request->validate([
    'name' => ['required', 'string', 'max:255'],
    'email' => ['required', 'string', 'lowercase', 'email', 'max:255', 'unique:'.Tenant::class],
    'password' => [
        'required',
        'confirmed',
        Password::min(8)
                ->numbers()
                ->symbols()
                ->letters()
                ->mixedCase()
    ],
    'image' => 'nullable|mimes:png,jpg,jpeg',
    'phone' => [
        'nullable',
        'regex:/^\d{10,11}$/',
        'unique:tenants,phone'  // Ensure the phone number is unique
    ],
]);
```

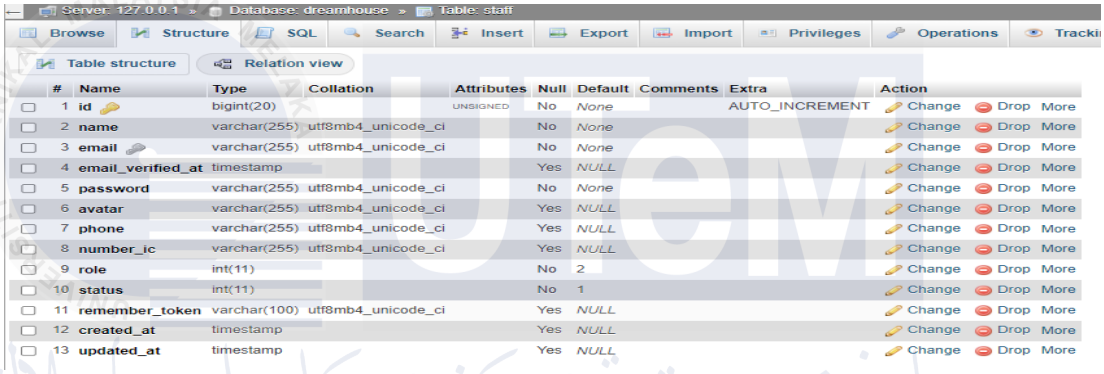
```
$request->validate([
    'state' => 'required',
    'city' => 'required',
    'postcode' => 'required',
    'address' => [
        'required',
        'string',
        'regex:/^[A-Za-z0-9\s\.\(\)\-]+$/',
    ],
    'deposit' => 'required|numeric|max:5250',
    'monthly' => 'required|numeric|max:1500',
    'room' => [
        'required',
        'integer',
        'max:8'
    ],
    'type' => 'required',
    'area' => [
        'required',
        'digits_between:1,4',
        'integer',
        'max:2600'
    ],
    'description' => ['required', function ($attribute, $value, $fail) {
        $formattedDescription = ucwords(strtolower($value));
        $formattedDescription = str_replace(',', ', ', $formattedDescription);
        if (!preg_match('/^[A-Za-z, ]*$/', $formattedDescription)) {
            $fail("The $attribute format is invalid.");
        }
    }],
    'title' => ['required', 'regex:/^[\w\s()]+$/'],
    'toilet' => 'required',
    'google_maps_link' => 'required|url',
    // Additional rule for images count
    'images' => ['required', function ($attribute, $value, $fail) use ($images, $minImageCount) {
        if (count($images) < $minImageCount) {
            $fail("You must upload at least {$minImageCount} images.");
        }
    }],
    'images.*' => 'image|mimes:jpg,jpeg,png,webp', // Not making it required here, because it's handled by the 'images' rule
]);
```

Figure 5.2-5 Validation Input

5.3 Database Implementation

DDL statements (SQL Queries)

```
CREATE TABLE `staff` (
  `StaffID` INT AUTO_INCREMENT PRIMARY KEY,
  `Name` VARCHAR(255) NOT NULL,
  `Email` VARCHAR(255) NOT NULL,
  `Password` VARCHAR(255) NOT NULL,
  `Avatar` VARCHAR(255) NOT NULL,
  `Phone` VARCHAR(255) NOT NULL,
  `Number_IC` VARCHAR(255) NOT NULL,
  `Role` INT NOT NULL DEFAULT 2,
  `Status` INT NOT NULL DEFAULT 1,
  `Created_at` TIMESTAMP NOT NULL
);
```



Server: 127.0.0.1 » Database: dreamhouse » Table: staff

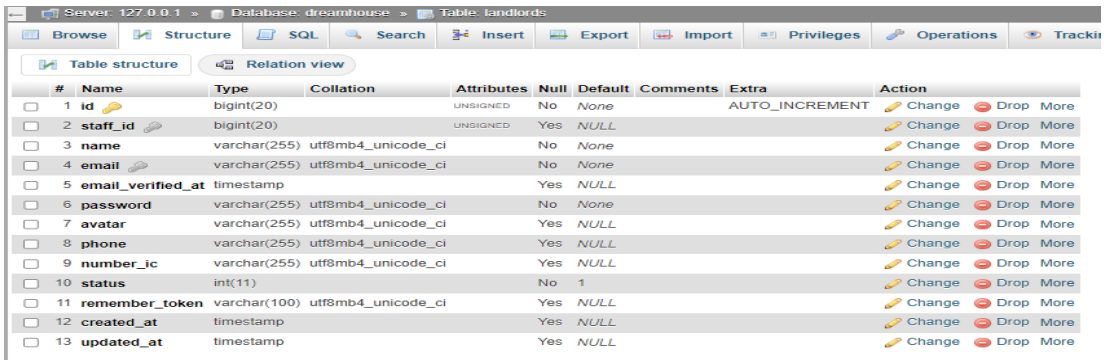
Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracki

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	name	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
3	email	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
4	email_verified_at	timestamp			Yes	NULL			Change Drop More
5	password	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
6	avatar	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
7	phone	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
8	number_ic	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
9	role	int(11)			No	2			Change Drop More
10	status	int(11)			No	1			Change Drop More
11	remember_token	varchar(100)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
12	created_at	timestamp			Yes	NULL			Change Drop More
13	updated_at	timestamp			Yes	NULL			Change Drop More

Figure 5.3-1 Table `staff``


```
CREATE TABLE `landlords` (
  `LandlordID` INT AUTO_INCREMENT PRIMARY KEY,
  `StaffID` INT DEFAULT NULL,
  `Name` VARCHAR(255) NOT NULL,
  `Email` VARCHAR(255) NOT NULL,
  `Password` VARCHAR(255) NOT NULL,
  `Avatar` VARCHAR(255) NOT NULL,
  `Phone` VARCHAR(255) NOT NULL,
  `Number_IC` VARCHAR(255) DEFAULT NULL,
  `Status` INT NOT NULL DEFAULT 1,
  `Created_At` TIMESTAMP NOT NULL,
  FOREIGN KEY (`StaffID`) REFERENCES `staff`(`StaffID`)
);
```



Server: 127.0.0.1 » Database: dreamhouse » Table: landlords

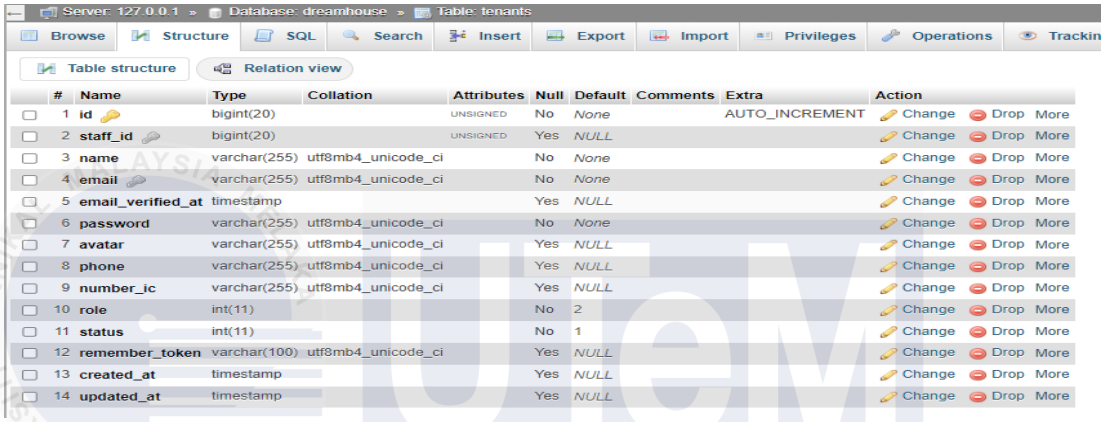
Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Tracki

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	staff_id	bigint(20)		UNSIGNED	Yes	NULL			Change Drop More
3	name	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
4	email	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
5	email_verified_at	timestamp			Yes	NULL			Change Drop More
6	password	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
7	avatar	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
8	phone	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
9	number_ic	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
10	status	int(11)			No	1			Change Drop More
11	remember_token	varchar(100)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
12	created_at	timestamp			Yes	NULL			Change Drop More
13	updated_at	timestamp			Yes	NULL			Change Drop More

Figure 5.3-2 Table `landlords`

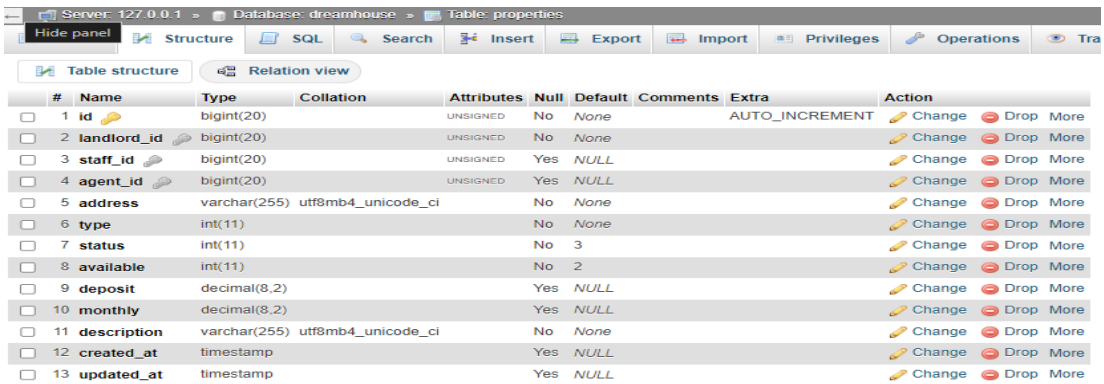
```
CREATE TABLE `tenants` (
  `TenantID` INT AUTO_INCREMENT PRIMARY KEY,
  `StaffID` INT DEFAULT NULL,
  `Name` VARCHAR(255) NOT NULL,
  `Email` VARCHAR(255) NOT NULL,
  `Password` VARCHAR(255) NOT NULL,
  `Avatar` VARCHAR(255) DEFAULT NULL,
  `Phone` VARCHAR(255) DEFAULT NULL,
  `Number_IC` VARCHAR(255) DEFAULT NULL,
  `Role` INT NOT NULL DEFAULT 2,
  `Status` INT NOT NULL DEFAULT 1,
  `Created_At` TIMESTAMP NOT NULL,
  FOREIGN KEY (`StaffID`) REFERENCES `staff`(`StaffID`)
);
```



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	staff_id	bigint(20)		UNSIGNED	Yes	NULL			Change Drop More
3	name	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
4	email	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
5	email_verified_at	timestamp			Yes	NULL			Change Drop More
6	password	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
7	avatar	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
8	phone	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
9	number_ic	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
10	role	int(11)			No	2			Change Drop More
11	status	int(11)			No	1			Change Drop More
12	remember_token	varchar(100)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
13	created_at	timestamp			Yes	NULL			Change Drop More
14	updated_at	timestamp			Yes	NULL			Change Drop More

Figure 5.3-3 Table `tenants`


```
CREATE TABLE `properties` (
  `PropertyID` INT AUTO_INCREMENT PRIMARY KEY,
  `LandlordID` INT NOT NULL,
  `StaffID` INT NOT NULL,
  `AgentID` INT DEFAULT NULL,
  `Address` VARCHAR(255) NOT NULL,
  `Type` INT NOT NULL,
  `Status` INT NOT NULL DEFAULT 3,
  `Available` INT NOT NULL DEFAULT 2,
  `Deposit` DECIMAL(6,2) DEFAULT NULL,
  `Monthly` DECIMAL(6,2) DEFAULT NULL,
  `Description` VARCHAR(255) NOT NULL,
  `Created_At` TIMESTAMP NOT NULL,
  FOREIGN KEY (`LandlordID`) REFERENCES `landlord`(`LandlordID`),
  FOREIGN KEY (`StaffID`) REFERENCES `staff`(`StaffID`),
  FOREIGN KEY (`AgentID`) REFERENCES `staff`(`StaffID`)
);
```



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	landlord_id	bigint(20)		UNSIGNED	No	None			Change Drop More
3	staff_id	bigint(20)		UNSIGNED	Yes	NULL			Change Drop More
4	agent_id	bigint(20)		UNSIGNED	Yes	NULL			Change Drop More
5	address	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
6	type	int(11)			No	None			Change Drop More
7	status	int(11)			No	3			Change Drop More
8	available	int(11)			No	2			Change Drop More
9	deposit	decimal(8,2)			Yes	NULL			Change Drop More
10	monthly	decimal(8,2)			Yes	NULL			Change Drop More
11	description	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
12	created_at	timestamp			Yes	NULL			Change Drop More
13	updated_at	timestamp			Yes	NULL			Change Drop More

Figure 5.3-4 Table `properties`

```
CREATE TABLE `contracts` (
  `ContractID` INT AUTO_INCREMENT PRIMARY KEY,
  `PropertyID` INT NOT NULL,
  `StaffID` INT DEFAULT NULL,
  `AgentID` INT NOT NULL,
  `TenantID` INT NOT NULL,
  `Period` INT NOT NULL,
  `Deposit` DECIMAL(6,2) NOT NULL,
  `Total` DECIMAL(8,2) NOT NULL,
  `Balance` DECIMAL(10,2) NOT NULL,
  `Status` INT NOT NULL DEFAULT 2,
  `Start_Date` DATE NOT NULL,
  `End_Date` DATE NOT NULL,
  `Created_At` TIMESTAMP NOT NULL,
  FOREIGN KEY (`StaffID`) REFERENCES `staff`(`StaffID`),
  FOREIGN KEY (`PropertyID`) REFERENCES `properties`(`PropertyID`),
  FOREIGN KEY (`AgentID`) REFERENCES `staff`(`StaffID`),
  FOREIGN KEY (`TenantID`) REFERENCES `tenant`(`TenantID`)
);
```

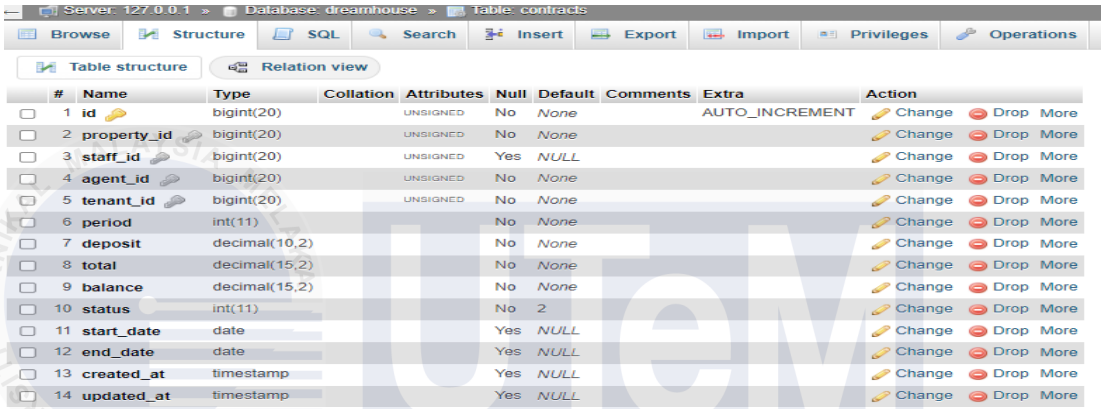


Figure 5.3-5 Table `contracts`


```
CREATE TABLE `payments` (
  `ContractID` INT NOT NULL,
  `Type` INT NOT NULL,
  `Total` DECIMAL(8,2) NOT NULL,
  `Created_At` TIMESTAMP NOT NULL,
  FOREIGN KEY (`ContractID`) REFERENCES `contracts`(`ContractID`)
);
```

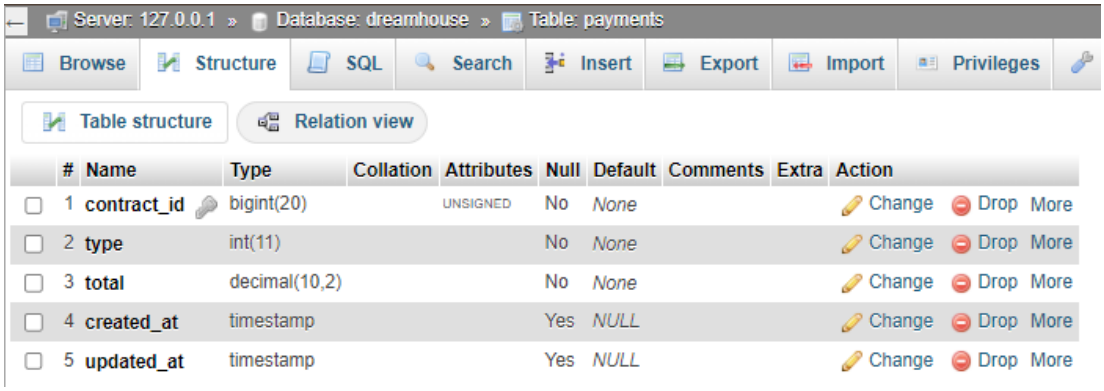


Figure 5.3-6 Table `payments`

```
CREATE TABLE `location_properties` (
  `LocationID` INT AUTO_INCREMENT PRIMARY KEY,
  `PropertyID` INT NOT NULL,
  `Latitude` decimal(10,8) NOT NULL,
  `Longitude` decimal(11,8) NOT NULL,
  `Created_At` timestamp NULL DEFAULT NULL,
  FOREIGN KEY (`PropertyID`) REFERENCES `property`(`PropertyID`)
);
```

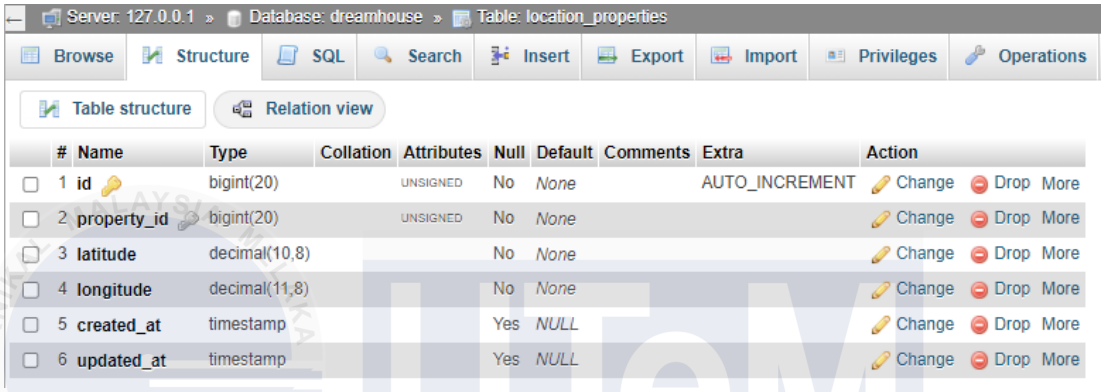


Figure 5.3-7 Table `location_properties`

```
CREATE TABLE `image_properties` (
  `ImageID` INT AUTO_INCREMENT PRIMARY KEY,
  `PropertyID` INT NOT NULL,
  `ImageURL` VARCHAR(255) NOT NULL,
  `Created_At` TIMESTAMP NOT NULL,
  FOREIGN KEY (`PropertyID`) REFERENCES `properties`(`PropertyID`)
);
```

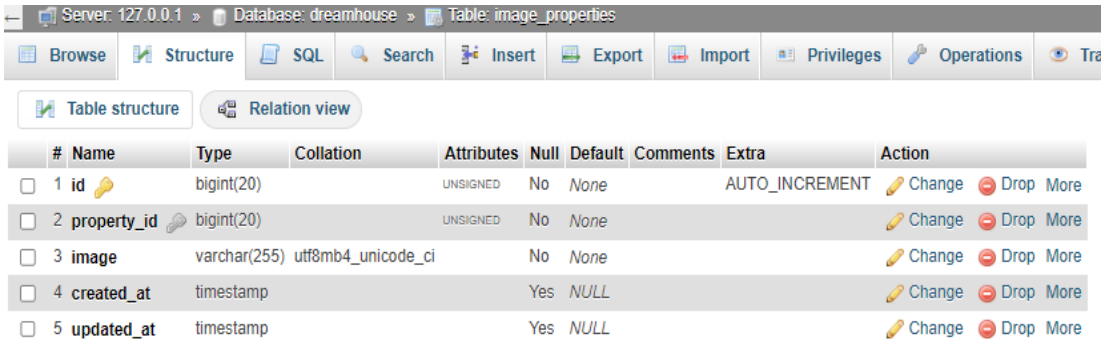
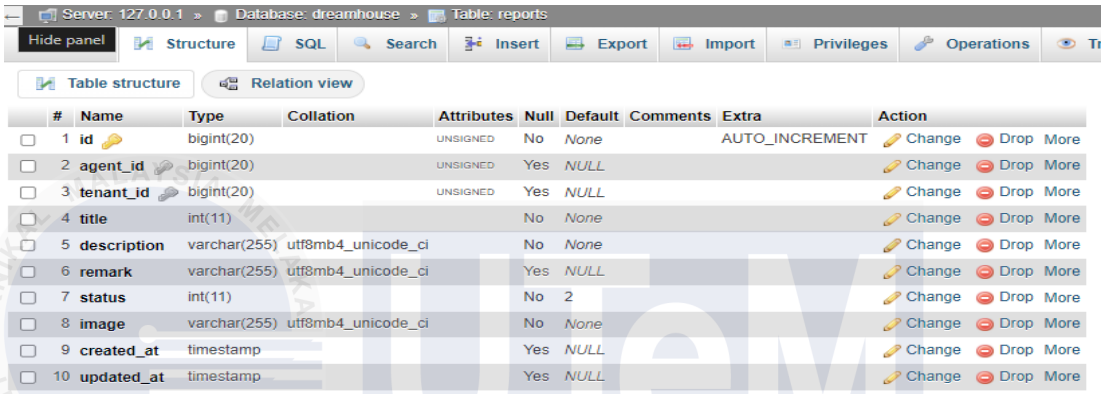


Figure 5.3-8 Table `image_properties`

```
CREATE TABLE `reports` (
  `ReportID` INT AUTO_INCREMENT PRIMARY KEY,
  `AgentID` INT NOT NULL,
  `TenantID` INT NOT NULL,
  `Title` INT NOT NULL,
  `Description` VARCHAR(255) NOT NULL,
  `Remark` VARCHAR(255) DEFAULT NULL,
  `Status` INT NOT NULL DEFAULT 2,
  `ImageURL` VARCHAR(255) NOT NULL,
  `Created_At` TIMESTAMP NULL DEFAULT NULL,
  FOREIGN KEY (`AgentID`) REFERENCES `staff`(`StaffID`),
  FOREIGN KEY (`TenantID`) REFERENCES `tenant`(`TenantID`)
);
```



Server: 127.0.0.1 » Database: dreamhouse » Table: reports

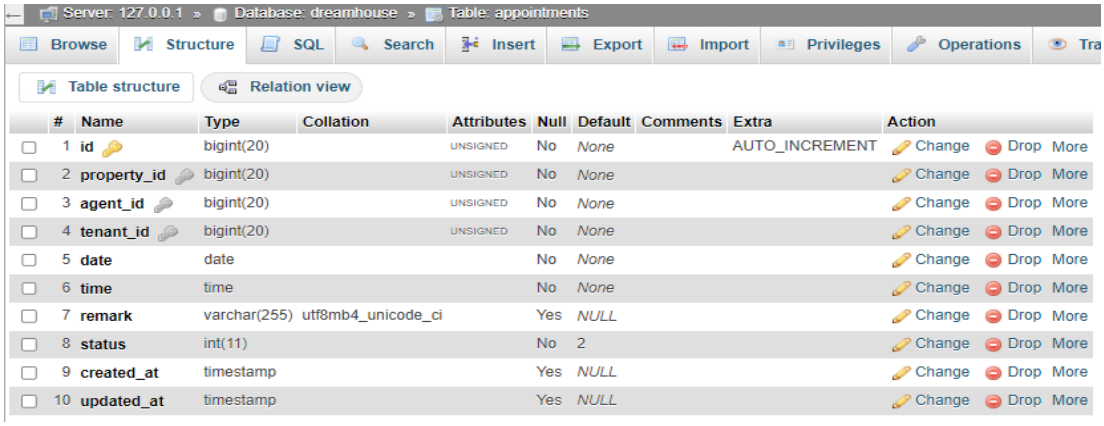
Hide panel | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	agent_id	bigint(20)		UNSIGNED	Yes	NULL			Change Drop More
3	tenant_id	bigint(20)		UNSIGNED	Yes	NULL			Change Drop More
4	title	int(11)			No	None			Change Drop More
5	description	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
6	remark	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
7	status	int(11)			No	2			Change Drop More
8	image	varchar(255)	utf8mb4_unicode_ci		No	None			Change Drop More
9	created_at	timestamp			Yes	NULL			Change Drop More
10	updated_at	timestamp			Yes	NULL			Change Drop More

Figure 5.3-9 Table `reports`


```
CREATE TABLE `appointments` (
  `AppointmentID` INT AUTO_INCREMENT PRIMARY KEY,
  `PropertyID` INT NOT NULL,
  `AgentID` INT NOT NULL,
  `TenantID` INT NOT NULL,
  `Date` date NOT NULL,
  `Time` time NOT NULL,
  `Remark` varchar(255) DEFAULT NULL,
  `Status` int(11) NOT NULL DEFAULT 2,
  `Created_At` timestamp NULL DEFAULT NULL,
  FOREIGN KEY (`PropertyID `) REFERENCES `property`(`PropertyID`),
  FOREIGN KEY (`AgentID `) REFERENCES `staff`(`StaffID`),
  FOREIGN KEY (`TenantID `) REFERENCES `tenant`(`TenantID`)
);
```



Server: 127.0.0.1 » Database: dreamhouse » Table: appointments

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	property_id	bigint(20)		UNSIGNED	No	None			Change Drop More
3	agent_id	bigint(20)		UNSIGNED	No	None			Change Drop More
4	tenant_id	bigint(20)		UNSIGNED	No	None			Change Drop More
5	date	date			No	None			Change Drop More
6	time	time			No	None			Change Drop More
7	remark	varchar(255)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
8	status	int(11)			No	2			Change Drop More
9	created_at	timestamp			Yes	NULL			Change Drop More
10	updated_at	timestamp			Yes	NULL			Change Drop More

Figure 5.3-10 Table `appointments`

5.4 Conclusion

Chapter 5 has provided a comprehensive overview of the implementation phase for the DreamHouse project, encompassing database setup, model creation, controller and view logic, routing, and testing. In Section 5.1, we established the foundation for the application by setting up the MySQL database and configuring Laravel. Section 5.2 focused on building the application's core functionality, defining models to represent database entities, creating controllers to handle user interactions, and designing views to present data. Finally, Section 5.3 addressed the deployment and maintenance aspects of the project, ensuring the application is ready for production and can be effectively supported going forward. By following these steps, we have successfully constructed a functional and scalable House Rental Management System.

CHAPTER 6: TESTING

6.1 About References

The testing phase is crucial to ensuring that the Dreamhouse Rental Management System functions as intended and meets the needs of its users, including guests, tenants, landlords, agents, and administrators. The system facilitates key activities such as property registration by landlords, property assignment by administrators, property browsing by guests, contract creation by agents, and rental management by tenants. To verify that all functionalities work correctly, a comprehensive testing process will be carried out.

The testing strategy adopted for this project primarily involves black-box testing. This method focuses on validating the system's outputs based on specific inputs without examining the internal code structure. This approach is ideal for testing the system's functionality from the perspective of different user roles. By implementing this testing strategy, the project aims to ensure the system's robustness, reliability, and user satisfaction.

6.2 Test Plan

6.2.1 Test Organization

As this is an individual project, all testing activities will be conducted by the developer which me and friend. This includes designing the test cases, setting up the test environment, executing the tests, and analyzing the results. The developer will also act as a user to simulate interactions from different roles such as guest, tenant, landlord, agent, and admin. This approach ensures that the testing covers all aspects of the system from multiple user perspectives.

Table 6.2.1-1 List of Tester

Tester ID	Name	Position	Responsibilities
T01	Aiman Izzat	System Developer & Test Manager	- Prepare test plan - Involve in unit testing and integration testing.
T02	Nuqman Danial	System Tester	- Detect and track system error for validation input and effective button. - Provide feedback to enhance system

6.2.2 Test Environment

The testing will be carried out on a local development environment set up on the developer's machine. The environment will replicate the expected production setup, including the necessary software and configurations. The testing environment will include the following:

i. Hardware

Table 6.2.2-1 Device Specification

Environment Specification	Description
Laptop	MSI GF63 Thin
CPU	12th Gen Intel(R) Core(TM) i5-12450H
Memory (RAM)	8 GB
Storage (ROM)	512 GB

ii. Software

Table 6.2.2-2 Software For Testing

System Specification	Description
Web Server	Apache
Database	MySQL (phpMyAdmin)
Framework	Laravel 10
Back-end	PHP
Front-end	HTML5, CSS, Js
Browser	Chrome

The testing phase for the Dreamhouse Rental Management System will be conducted over two cycles, each spanning one week, and will cover various crucial aspects of the system:

1. Database Testing to ensuring the integrity and performance of database operations.

2. Unit Testing to verifying individual components for correctness in isolation.

3. Integration Testing to checking the interactions between integrated units for any discrepancies.

4. System Testing to assessing the system’s overall behavior and stability.

5. User Acceptance Testing to confirming the system meets the end-users' needs and requirements, based on their feedback.

Table 6.2.2-3 Schedule For Testing

Module	Start Date	End Date	Test Type	Tester
Guest	27/8/2024	28/8/2024	System Testing	T01, T02
Tenant	27/8/2024	28/8/2024	Unit Testing, Integration Testing, System Testing	T01, T02
Landlord	27/8/2024	28/8/2024	Unit Testing, Integration Testing, System Testing	T01, T02
Agent	27/8/2024	28/8/2024	Unit Testing, Integration Testing, System Testing	T01, T02
Admin	27/8/2024	28/8/2024	Unit Testing, Integration Testing, System Testing	T01, T02

6.3 Test Strategy

The testing strategy selected for the Dreamhouse Rental Management System primarily involves black-box testing. This method focuses on validating the system's functionality through various user interactions without examining the internal code structure. By adopting this approach, we ensure that the system behaves as expected from the perspectives of different user roles, including guests, tenants, landlords, agents, and administrators. This comprehensive testing covers the functional aspects of the system, confirming its readiness and reliability for real-world usage.

6.3.1 Classes of tests

The following classes of tests will be conducted to ensure the system meets all requirements:

1. Database Testing

   Database testing primarily aims to ensure that the storage and retrieval of data work as intended and that the database maintains data integrity and security. It also tests the database's schema, data model, and performance under various loads.

2. Unit Testing

   Focuses on testing individual components or modules of the system to ensure that each part functions correctly in isolation. It is usually conducted by the developer during the development phase.

3. Integration Testing

   Integration Testing focuses on verifying the interactions between different modules or components within the “My UTeM Laptop Service” system. Theyr individual modules are tested, they are integrated, and this test checks whether they work together as expected.

4. System Testing

   Evaluates the complete system as a whole to ensure that all components work together seamlessly. It tests the integration of modules and the overall functionality of the system.

5. User Acceptance Testing

   The user acceptance testing involves having testers or end-users review or re-execute the previous test cases to confirm the system's effectiveness.

6.4 Test Design

6.4.1 Test Description

To ensure that the Dreamhouse Rental Management System operates as expected, a series of test cases have been designed and documented for each key functionality. These test cases focus on validating the core operations of the system, including user registration, property management, contract creation, and rental payments. By systematically testing each module, potential issues can be identified and addressed before deployment.

Each test case defines the specific actions to be performed, the expected results, and the criteria for success. These test cases cover a wide range of scenarios, from typical user interactions to edge cases that may challenge the system's robustness. The goal of this testing process is to verify that every module of the system functions correctly and meets the intended user requirements.

1. Database Testing

Table 6.4.1-1 Test Description For Database Testing

Module	Property Management		
Type	Database Testing	Date	28/8/2024
Method	Black Box		
Description	Manage property list (admin)		
Test Case ID	Description	Step	Expected Result
DT_001 Structural Testing	Verify that all database schema components match the specifications.	1. Inspect database schema. 2. Verify data types,	All tables and columns conform to the specified

		relationships, and constraints.	schema without discrepancies.
DT_002 Functional Testing	Test the addition of a new property through a stored procedure.	1. Execute the stored procedure with provided parameters. 2. Query the database to ensure the property is added.	Property is added correctly and all details match the provided parameters.
DT_003 Data Integrity	Ensure that data integrity is maintained when deleting a property with active contract.	1. Attempt to delete a property that has active contract. 2. Check error handling and data integrity.	Deletion is prevented, and an appropriate error message is displayed.
DT_004 Performance Testing	Assess the performance of the database under simulated high user load.	1. Simulate multiple concurrent property searches. 2. Measure response times and system behavior under load.	All queries return results within acceptable time frames, and system remains stable.
DT_005 Security Testing	Check for SQL injection vulnerabilities.	1. Input malicious SQL statements into input fields. 2. Monitor the database for unauthorized actions or errors.	Input is sanitized; no unauthorized SQL execution occurs.

DT_006 Backup and Recovery	Test the system's ability to recover data accurately after a simulated failure.	1. Backup the database. 2. Delete specific data. 3. Restore the database from the backup. 4. Verify the data integrity post-restoration.	All data is accurately restored to its state prior to deletion, and data integrity is maintained.

2. Unit Testing

Table 6.4.1-2 Test Description For Unit Testing (Agent Registration)

Module	Registration Module (Agent Registration)		
Type	Unit Testing	Date	28/8/2024
Method	Black Box		
Description	New agent registration functionalities		
Test Case ID	Description	Step	Expected Result
UT_001-1	All input field are blank.	1. Go to agent registration form page. 2. Click button ‘Register’.	Error message “This field is required” will be displayed for each input field.

UT_001-2	Verify email address have been used.	1. Fill the email input field with any email.	Error message “The email has already been taken.” displayed.
UT_001-3	Password input field are filled with wrong format.	1. Fill the password input field with 4 character of lowercase.	Error message “The password field must be at least 8 characters.” displayed.
UT_001-4	Password input field are filled with 8 character with lowercase.	1. Fill the password input field without have combination of uppercase, number and special character.	Error message “The password must at least 8 character that combination of special character, uppercase character, lowercase character and number..” displayed.
UT_001-5	Confirmation input field are not matched with password input field.	1. Fill confirmation password input field with unmatched password with password input field.	Error message “The password field confirmation does not match.” displayed.

UT_001-6	Phone number input field are filled with invalid format.	1. Filled the phone number input field with 15 character of number.	Error message “The phone field format is invalid.” displayed.
UT_001-7	Verify phone number have been used.	1. Enter existed phone number that been used by other agent.	Error message “The phone has already been taken.” displayed.
UT_001-8	IC number input field are filled with invalid format.	1. Filled the IC number input field with number and other character.	Error message “The IC number field format is invalid.” displayed.
UT_001-9	Verify IC number have been used.	1. Enter existed IC number that been used by other agent.	Error message “The IC number has already been taken.” displayed.
UT_001-10	Fill all input field with correct format.	1. Fill name, email, phone number, IC number, password, confirmation password and avatar field with correct format.	Success message “Agent successfully registered” displayed.

Table 6.4.1-3 Test Description FOr Unit Testing (Property Registration)

Module	Property Module (Property Registration)		
Type	Unit Testing	Date	28/8/2024
Method	Black Box		
Description	New property registration functionalities		
Test Case ID	Description	Step	Expected Result
UT_002-1	All input field are blank.	1. Go to property registration form page. 2. Click button 'Register'.	Error message "This field is required" will be displayed for each input field.
UT_002-2	Title input field are filled with wrong format	1. Fill the title input field with all character.	Error message "The title field format is invalid. Title should contain letters, spaces, and parentheses only." displayed.
UT_002-3	Street name input field are filled with wrong format.	1. Fill the street name input field with any character.	Error message "The address field format is invalid. Address should contain letters, numbers, spaces, and full stops, and

			parentheses only.” displayed.
UT_002-4	Deposit input field are filled with illogical value.	1. Fill the deposit input field with large number of value.	Error message “The deposit field must not be greater than 6000.” displayed.
UT_002-5	Monthly input field are filled with illogical value.	1. Fill the monthly input field with large number of value.	Error message “The monthly field must not be greater than 1600.” displayed.
UT_002-6	Area(sqft) input field are filled with illogical value.	1. Fill area(sqft) with large number of value.	Error message “The area field must not be greater than 2600.” displayed.
UT_002-7	Description input field are filled with invalid format.	1. Fill the description field with any character or number.	Error message “The description format is invalid. Description should contain letters, spaces, and commas only.” displayed.
UT_002-8	Google maps link field are filled with wrong format.	1. Enter google maps link with any	Error message “The google_maps_link must be a valid

		link except google maps link.	Google Maps URL with latitude and longitude.” displayed.
UT_002-9	Image input field are filled with wrong type of file.	1. Filled the image field with any file with pdf format.	Error message “The images field must be an image.” displayed.
UT_002-10	Image input field are filled with 7 file of images.	1. Select 7 images to upload.	Error message “You must upload at least 8 images.” displayed.
UT_002-11	Fill all input field with correct format.	1. Fill Title, State, City, Postcode, Building, Deposit, Monthly, Number of Room, Number of Toilet, Area, Description, Type of Property, Google Maps Link, and 8 Image for Image of Property .	Success message “Property successfully registered” displayed.

3. Integration Testing

Table 6.4.1-4 Test Description For Integration Testin (Contract Management)

Module	Contract Management		
Type	Integration Testing	Date	28/8/2024
Method	Black Box		
Description	New contract registration and view detail contract		
Test Case ID	Description	Step	Expected Result
IT_001-1	Verify function to store registration of contract with filled all required field with correct format to triggered store procedure to store data of contract into database.	1. Agent go to registration contract form. 2. Select property address, tenant email, duration and choose feature date.	Success message will displayed "New contract registered" and role of new guest will be change to tenant automatically.
IT_001-2	Verify function to view detail of contract that have been registered to make sure that data correctly stored.	1. Agent go to contract list page. 2. Click button green "eye" icon to view detail of contract have been registered.	All detail of contract will display. The total and balance will auto generated based on monthly of property and duration of the contract.

IT_001-3	Verify function to update detail of contract for contract status is pending and trigger update procedure to update the detail of contract.	. Go to list contract page. 2. Click button blue “pencil” icon to update detail of contract. 3. Update any field with correct format.	Details of contract have been updated with new detail for the field that have been updated. If uupdated field is tenant email, the old tenant will become guest if they not become tenant before this.

Table 6.4.1-5 Test Description For Integration Testing (Property Management)

Module	Property Management		
Type	Integration Testing	Date	28/8/2024
Method	Black Box		
Description	New property registration and view detail		
Test Case ID	Description	Step	Expected Result
IT_002-1	Verify function to register new property with filled all required field and triggered the procedure to store the data into database.	1. Landlord go to register property form and filled all required field with correct format.	Success message will displayed “New property registered”.

IT_002-2	Verify function to view detail of property that have been registered to make sure all data store correctly.	1. Go to list property page. 2. Click button green ‘eye’ icon to view detail of property.	All details of the property must be correct and same as data in all required field have been filled.
IT_002-3	Verify the function to update detail of property information and triggered update procedure to update data of the property.	1. Go to list property page. 2. Click button blue “pencil” icon to update detail of property. 3. Update any field with correct format.	Details of property have been updated with new detail for the field that have been updated.

4. System Testing

Table 6.4.1-6 Test Description For System Testing (User Credential, Property Listing, Contract Management)

Module	User Credential, Property Listing, Contract Management		
Type	Integration Testing	Date	28/8/2024
Method	Black Box		
Description	New property registration and view detail		

Test Case ID	Description	Step	Expected Result
ST_001	Test login functionality for all user roles.	1. Attempt to login as an admin. 2. Attempt to login as an agent. 3. Attempt to login as a tenant. 4. Attempt to login as a guest.	Each user role should be able to login successfully and be directed to the correct landing page.
ST_002	Verify property listing addition and retrieval.	1. Log in as an landlord 2. Add a new property. 3. Retrieve the added property details.	Property details are correctly added and displayed upon retrieval.
ST_003	Check search functionality with filters.	1. Perform searches using different filters like address, price range, property type and number of rooms.	Search results match the criteria specified in the filters.
ST_004	Validate contract process for a property.	1. Log in as a tenant. 2. Select a property. 3. Complete the contract process.	Contract is successfully processed, and confirmation is received.

ST_005	Assess system performance under high load.	1. Simulate multiple users accessing the system simultaneously. 2. Monitor response times and system behavior.	System remains stable and responsive, with no significant performance degradation.
ST_006	Test system security measures.	1. Attempt common security breaches like SQL injections and XSS attacks. 2. Check for unauthorized data access.	System is secure against attacks, and no unauthorized access is permitted.
ST_007	Evaluate backup and recovery processes.	1. Perform a backup. 2. Simulate a data loss. 3. Execute a recovery from backup.	System recovers all data accurately and within the expected time frame.

5. User Acceptance

Question for user feedback

Table 6.4.1-7 Test Question For User Acceptances

No.	Question	Input Type	Optional Answer
1	Name	Text field	
2	Role	Dropdown	Tenant, Guest, Landlord, Admin, Agent
3	Date of Testing	Date picker	
4	How intuitive did you find the navigation of our system?	Initial Scale	Very difficult, Difficult, Normal, Easy, Very Easy
5	Were you able to complete the tasks without assistance?	Multiple choice	Yes, No
6	Did you encounter any issues while using the system?	Multiple choice	Yes, No
7	Was the property search function up to your expectations?	Multiple choice	Yes, No
8	Was the information about each property clear and sufficient?	Multiple choice	Yes, No
9	Did you find any discrepancies or errors in property details?	Multiple choice	Yes, No

10	How would you rate the speed of the system?	Initial Scale	Very difficult, Difficult, Normal, Easy, Very Easy
11	Did the system perform consistently during your tests?	Multiple choice	Yes, No
12	Were there any unexpected restarts or crashes?	Multiple choice	Yes, No
13	How satisfied are you with the overall functionality of the system?	Initial Scale	Very dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied
14	Do you consent to the use of your responses for the improvement of our system?	Multiple choice	Yes, No

6.4.2 Test Data

The data used for testing the Dreamhouse Rental Management System will be a combination of synthetic data created specifically for testing purposes. Synthetic data allows for the simulation of various real-world scenarios without relying on actual user information or property listings. This approach ensures that the system can be thoroughly tested while maintaining data privacy and security.

The synthetic data will represent different user roles, including guests, tenants, landlords, agents, and administrators. This data will be used to test functionalities such as user registration, property management, contract creation, and rental payments. By carefully crafting the data, the system's ability to handle a wide range of scenarios, including edge cases, will be evaluated. This ensures that the system is robust and ready for deployment.

Database Testing

Table 6.4.2-1 Test Data For Database Testing

Test Data ID	Test Data	Context
DTD_001	N/A	No specific data input needed. Inspection focuses on structural elements like tables, columns, and relationships defined in the database schema.
DTD_002	**Parameters**: {Title: “Terrace For Rent”, State: "Melaka", City: “Durian Tunggal”, Postcode: “76100”, Street Name: “Taman Bukit Tambun”, Room: “4”, Toilet: “3”, Area: “1234”, Type: “Terrace”, Depsoit: “1500”, Monthly: “950”, Description: "Fully Furnished, Swimming Pool”, Google Maps Link: “Any google maps”, Image: “10 file of image .jpeg”.	This test data simulates adding a new property listing to the system. It covers typical data fields that a property listing would include, ensuring the stored procedure can handle standard inputs.
DTD_003	Property ID: 25, Contract IDs: [23, 28]	This scenario uses a property that already has active contract linked to it. The test data is designed to validate foreign key constraints and prevent deletion of data that

		would violate referential integrity.
DTD_004	Simulate 100 concurrent searches for "Type : Terrace, Room: 3, Monthly: 450(min)-1000(max)"	This test involves multiple users performing the same search query to evaluate how well the database handles high traffic and parallel processing.
DTD_005	Input: ' OR 1=1;--	This classic SQL injection test checks if the system properly sanitizes inputs to prevent malicious SQL code execution, which could compromise database security.
DTD_006	Steps include deleting rows from the ‘properties’ table, then restoring.	The test involves intentional deletion of data followed by a restoration process to verify the effectiveness of backup systems and procedures in real-world data loss scenarios.

Unit Testing

Table 6.4.2-2 Data Test For Unit Testing (Agent Registration)

Test Data ID	Input Type	Test Data
TD_001-1	Name	
	Email	
	IC number	
	Phone	
	Password	
	Confirmation Password	
	Avatar	
TD_001-2	Name	Ahmad Ismail
	Email	ismail@gmail.com
	IC number	010503020432
	Phone	01921343309
	Password	Password@1
	Confirmation Password	Password@1
	Avatar	avatar.jpeg
TD_001-3	Name	Ahmad Ismail
	Email	ismail@gmail.com
	IC number	010503020431
	Phone	01121652347
	Password	Password@1
	Confirmation Password	Password@1
	Avatar	avatar.jpeg
TD_001-4	Name	Ahmad Ismail
	Email	ismail@gmail.com
	IC number	010503020431
	Phone	01921343309
	Password	abcd
	Confirmation Password	Password@1

	Avatar	avatar.jpeg
TD_001-5	Name	Ahmad Ismail
	Email	ismail@gmail.com
	IC number	010503020431
	Phone	01921343309
	Password	password123
	Confirmation Password	Password@1
	Avatar	avatar.jpeg
TD_001-6	Name	Ahmad Ismail
	Email	ismail@gmail.com
	IC number	010503020431
	Phone	01921343309
	Password	Password@1
	Confirmation Password	Password@12
	Avatar	avatar.jpeg
TD_001-7	Name	Ahmad Ismail
	Email	ismail01@gmail.com
	IC number	010503020431
	Phone	01921343309
	Password	Password@1
	Confirmation Password	Password@1
	Avatar	avatar.jpeg
TD_001-8	Name	Ahmad Ismail
	Email	ismail@gmail.com
	IC number	010503020431123
	Phone	01921343309
	Password	Password@1
	Confirmation Password	Password@1
	Avatar	avatar.jpeg
TD_001-9	Name	Ahmad Ismail
	Email	ismail@gmail.com
	IC number	010503020431

	Phone	019213433091234
	Password	Password@1
	Confirmation Password	Password@1
	Avatar	avatar.jpeg
TD_001-10	Name	Ahmad Ismail
	Email	ismail@gmail.com
	IC number	010503020431
	Phone	01921343309
	Password	Password@1
	Confirmation Password	Password@1
	Avatar	avatar.jpeg

Table 6.4.2-3 Data Test For Unit Testing (Property Registration)

Test Data ID	Input Type	Test Data
TD_002-1	Title	
	State	
	City	
	Postcode	
	Street Name	
	Deposit	
	Monthly	
	Area (sqft)	
	Description	
	Number of Room	
	Number of Toilet	
	Type of Property	
	Google Maps Link	
	Image of Property	
TD_002-2	Title	Terrace for Rent . @ Complete.
	State	Kedah

	City	Baling
	Postcode	09100
	Street Name	No.21 Kampung Kaseh
	Deposit	1500
	Monthly	500
	Area (sqft)	1234
	Description	Fully Furnished, Swimming Pool
	Number of Room	3
	Number of Toilet	2
	Type of Property	Terrace
	Google Maps Link	Any full google maps link
	Image of Property	image.jpeg (10 file)
TD_002-3	Title	Terrace for Rent
	State	Kedah
	City	Baling
	Postcode	09100
	Street Name	No.21 Kampung Kaseh, (Lorong 2.
	Deposit	1500
	Monthly	500
	Area (sqft)	1234
	Description	Fully Furnished, Swimming Pool
	Number of Room	3
	Number of Toilet	2
	Type of Property	Terrace
	Google Maps Link	Any full google maps link
	Image of Property	image.jpeg (10 file)
TD_002-4	Title	Terrace for Rent
	State	Kedah
	City	Baling
	Postcode	09100
	Street Name	No.21 Kampung Kaseh
	Deposit	1500000

	Monthly	500
	Area (sqft)	1234
	Description	Fully Furnished, Swimming Pool
	Number of Room	3
	Number of Toilet	2
	Type of Property	Terrace
	Google Maps Link	Any full google maps link
	Image of Property	image.jpeg (10 file)
TD_002-5	Title	Terrace for Rent
	State	Kedah
	City	Baling
	Postcode	09100
	Street Name	No.21 Kampung Kaseh
	Deposit	1500
	Monthly	500000
	Area (sqft)	1234
	Description	Fully Furnished, Swimming Pool
	Number of Room	3
	Number of Toilet	2
	Type of Property	Terrace
	Google Maps Link	Any full google maps link
	Image of Property	image.jpeg (10 file)
TD_002-6	Title	Terrace for Rent
	State	Kedah
	City	Baling
	Postcode	09100
	Street Name	No.21 Kampung Kaseh
	Deposit	1500
	Monthly	500
	Area (sqft)	1234
	Description	Fully Furnished, Swimming Pool
	Number of Room	3

	Number of Toilet	2
	Type of Property	Terrace
	Google Maps Link	Any full google maps link
	Image of Property	image.jpeg (10 file)
TD_002-7	Title	Terrace for Rent
	State	Kedah
	City	Baling
	Postcode	09100
	Street Name	No.21 Kampung Kaseh
	Deposit	1500
	Monthly	500
	Area (sqft)	1234
	Description	(Near Masjid) Public Transport @
	Number of Room	3
	Number of Toilet	2
	Type of Property	Terrace
	Google Maps Link	Any full google maps link
	Image of Property	image.jpeg (10 file)
TD_002-8	Title	Terrace for Rent
	State	Kedah
	City	Baling
	Postcode	09100
	Street Name	No.21 Kampung Kaseh
	Deposit	1500
	Monthly	500
	Area (sqft)	1234
	Description	Fully Furnished, Swimming Pool
	Number of Room	3
	Number of Toilet	2
	Type of Property	Terrace
	Google Maps Link	Any full google maps link
	Image of Property	image.pdf (10 file)

TD_002-9	Title	Terrace for Rent
	State	Kedah
	City	Baling
	Postcode	09100
	Street Name	No.21 Kampung Kaseh
	Deposit	1500
	Monthly	500
	Area (sqft)	1234
	Description	Fully Furnished, Swimming Pool
	Number of Room	3
	Number of Toilet	2
	Type of Property	Terrace
	Google Maps Link	www.youtube.com
	Image of Property	image.jpeg (10 file)
TD_002-10	Title	Terrace for Rent
	State	Kedah
	City	Baling
	Postcode	09100
	Street Name	No.21 Kampung Kaseh
	Deposit	1500
	Monthly	500
	Area (sqft)	1234
	Description	Fully Furnished, Swimming Pool
	Number of Room	3
	Number of Toilet	2
	Type of Property	Terrace
	Google Maps Link	Any full google maps link
	Image of Property	image.jpeg (7 file)
TD_002-11	Title	Terrace for Rent
	State	Kedah
	City	Baling
	Postcode	09100

	Street Name	No.21 Kampung Kaseh
	Deposit	1500
	Monthly	500
	Area (sqft)	1234
	Description	Fully Furnished, Swimming Pool
	Number of Room	3
	Number of Toilet	2
	Type of Property	Terrace
	Google Maps Link	Any full google maps link
	Image of Property	image.jpeg (10 file)

Integration Testing

Table 6.4.2-4 Datat Test For Integration Testing (Contract Management)

Test Data ID	Input Type	Test Data
DT_003-1	Property Address	Choose any property address
	Tenant Email	Choose any tenant email
	Period (month)	6
	Number IC	010213043212
	Start of Contract	08/31/2024
DT_003-2	Property Address	Choose any property address
	Tenant Email	Choose any tenant email
	Period (month)	12
	Number IC	010213043212
	Start of Contract	08/31/2024

Table 6.4.2-5 Data Test For Integration (Property Management)

Test Data ID	Input Type	Test Data
TD_004-1	Title	Terrace for Rent
	State	Kedah
	City	Baling
	Postcode	09100
	Street Name	No.21 Kampung Kaseh
	Deposit	1500
	Monthly	500
	Area (sqft)	1234
	Description	Fully Furnished, Swimming Pool
	Number of Room	3
	Number of Toilet	2
	Type of Property	Terrace
	Google Maps Link	www.youtube.com
	Image of Property	image.jpeg (10 file)
TD_004-2	Title	A Terrace For Rent
	State	Melaka
	City	Durian Tunggal
	Postcode	76100
	Street Name	No.21 Kampung Kaseh (Lorong 2)
	Deposit	1200
	Monthly	850
	Area (sqft)	1232
	Description	Fully Furnished, Near UTeM
	Number of Room	4
	Number of Toilet	3
	Type of Property	Terrace
	Google Maps Link	Any full google maps link
	Image of Property	image.jpeg (10 file)

System Testing

Table 6.4.2-6 Data Test For System Testing

Test Data ID	Input Type	Test Data
STD_001	User credentials for admin, agent, tenant, landlord, and guest.	Credentials used to test login functionality for different roles.
STD_002	Property details: {address, type, room, monthly price}	Data used to test the property listing and retrieval functions.
STD_003	Search parameters: address, price range, property type, number of rooms.	Parameters used to filter search results in the system.
STD_004	Contract details: property ID, tenant ID, Period, IC number (if tenant not have been tenant before this), Start of Contract Date	Information necessary to simulate a contract process by a agent
STD_005	Contract details: property ID, tenant ID, Period, IC number (if tenant not have been tenant before this), Start of Contract Date	Simulated user activity to test system performance under stress.

STD_006	Malicious SQL and script inputs	Inputs intended to test the system's resilience against security threats.
STD_007	Backup and simulated data loss	Data used to validate the effectiveness of backup and recovery protocols.

6.5 Test Results and Analysis

Table 6.4.2-1 Result and Analysis For Unit Testing

Test Case ID	Test Data ID	Expected Result	Actual Result	Pass/Fail
UT_001-1	TD_001-1	Agent failed to register. Error message displayed below each input “This field is required”.	Agent failed to register. Error message displayed below each input “This field is required”.	Pass
UT_001-2	TD_001-7	Agent failed to register and error message displayed “The email already been taken”.	Agent failed to register and error message displayed “The email already been taken”.	Pass
UT_001-3	TD_001-4	Agent failed to register and error message displayed “The password must	Agent failed to register and error message displayed “The password must	Pass

		be at least 8 character".	be at least 8 character".	
UT_001-4	TD_001-5	Agent failed to register and error message displayed "The password must be at least 8 character that combination of special character, uppercase character, lowercase character and number.".	Agent failed to register and error message displayed "The password must be at least 8 character that combination of special character, uppercase character, lowercase character and number.".	Pass
UT_001-5	TD_001-6	Agent failed to register and error message displayed "The password confirmation not matched".	Agent failed to register and error message displayed "The password confirmation not matched".	Pass
UT_001-6	TD_001-9	Agent failed to register and error message displayed "The phone format is invalid".	Agent failed to register and error message displayed "The phone format is invalid".	Pass
UT_001-7	TD_001-3	Agent failed to register and error message displayed "The phone has been taken".	Agent failed to register and error message displayed "The phone has been taken".	Pass

UT_001-8	TD_001-8	Agent failed to register and error message displayed “The IC number format is invalid”.	Agent failed to register and error message displayed “The IC number format is invalid”.	Pass
UT_001-9	TD_001-2	Agent failed to register and error message displayed “The IC number has been taken”.	Agent failed to register and error message displayed “The IC number has been taken”.	Pass
UT_001-10	TD_001-10	Agent successfully registered and success message will be displayed “New agent registered”.	Agent successfully registered and success message will be displayed “New agent registered”.	Pass
UT_002-1	TD_002-1	Property failed to register. Error message displayed below each input “This field is required”.	Property failed to register. Error message displayed below each input “This field is required”.	Pass
UT_0012-2	TD_002-2	Property failed to register and error message displayed “The title field format is invalid. Title should contain letters, spaces, and parentheses only.”.	Property failed to register and error message displayed “The title field format is invalid. Title should contain letters, spaces, and parentheses only.”.	Pass

UT_002-3	TD_002-3	Property failed to register and error message displayed “The address field format is invalid. Address should contain letters, numbers, spaces, and full stops, and parentheses only.”.	Property failed to register and error message displayed “The address field format is invalid. Address should contain letters, numbers, spaces, and full stops, and parentheses only.”.	Pass
UT_002-4	TD_002-4	Property failed to register and error message displayed “The deposit field must not be greater than 6000”.	Property failed to register and error message displayed “The deposit field must not be greater than 6000”.	Pass
UT_002-5	TD_002-5	Property failed to register and error message displayed “The monthly field must not be greater than 1500”.	Property failed to register and error message displayed “The monthly field must not be greater than 1500”.	Pass
UT_002-6	TD_002-6	Property failed to register and error message displayed “The area field must not be greater than 2600”.	Property failed to register and error message displayed “The area field must not be greater than 2600”.	Pass

UT_002-7	TD_002-7	Property failed to register and error message displayed “The description format is invalid. Description should contain letters, spaces, and commas only”.	Property failed to register and error message displayed “The description format is invalid. Description should contain letters, spaces, and commas only”.	Pass
UT_002-8	TD_002-9	Property failed to register and error message displayed “The google_maps_link must be a valid Google Maps URL with latitude and longitude.”.	Property failed to register and error message displayed “The google_maps_link must be a valid Google Maps URL with latitude and longitude.”.	Pass
UT_002-9	TD_002-8	Property failed to register and error message displayed “The images field must be an image”.	Property failed to register and error message displayed “The images field must be an image”.	Pass
UT_002-10	TD_002-10	Property failed to register and error message displayed “You must upload at least 8 images”.	Property failed to register and error message displayed “You must upload at least 8 images”.	Pass

UT_002-11	TD_002-11	Property successfully registered and success message “Property successfully registered” displayed.	Property successfully registered and success message “Property successfully registered” displayed.	Pass

Table 6.4.2-2 Result and Analysis For Integration Testing

Test Case ID	Test Data ID	Expected Result	Actual Result	Pass/Fail
IT_001-1	TD_003-1	Success message will displayed “New contract registered”. Contract detail successfully registered, and triggered store procedure to store all detail of contract into database correctly. If the tenant is new guest, their role will change from guest to tenant. Number IC will auto generated for tenant	Success message will displayed “New contract registered”. Contract detail successfully registered, and triggered store procedure to store all detail of contract into database correctly. If the tenant is new guest, their role will change from guest to tenant. Number IC will auto generated for tenant	Pass

		that have been tenant before. The available of property selected will automatically change to not available.	that have been tenant before. The available of property selected will automatically change to not available.	
IT_001-2	TD_003-1	The contract detail same to field that been filled before. The total and balance of contract will auto generated based on monthly of property and period.	The contract detail same to field that been filled before. The total and balance of contract will auto generated based on monthly of property and period.	Pass
IT_001-3	TD_004-2	Success message will displayed “Contract Updated”. The total and balance will updated based on update duration. The available of old property will change to available. The role of tenant before will change to guest if they not have been tenant before this.	Success message will displayed “Contract Updated”. The total and balance will updated based on update duration. The available of old property will change to available. The role of tenant before will change to guest if they not have been tenant before this.	Pass

IT_002-1	TD_004-1	Success message will displayed “New property registered”. Property detail successfully registered, and triggered store procedure to store all detail of property into database correctly.	Success message will displayed “New property registered”. Property detail successfully registered, and triggered store procedure to store all detail of property into database correctly.	Pass
IT_002-2	TD_004-1	The property detail same to field that been filled before.	The property detail same to field that been filled before.	Pass
IT_002-3	TD_004-2	Success message will displayed “Property Updated”. Title of property updated. The address also updated. The deposit and monthly updated. The number of room and toilet updated.	Success message will displayed “Property Updated”. Title of property updated. The address also updated. The deposit and monthly updated. The number of room and toilet updated.	Pass

Table 6.4.2-3 Result and Analysis For Database Testing

Test Case ID	Test Data ID	Expected Result	Actual Result	Pass/Fail
DT_001	DTD_001	All tables and columns conform to the specified schema.	All tables and columns correctly implemented as per schema.	Pass
DT_002	DTD_002	Property is added correctly and can be retrieved.	Property added successfully; details match the input.	Pass
DT_003	DTD_003	Deletion should fail due to dependency constraints.	Deletion failed; integrity constraints upheld.	Pass
DT_004	DTD_004	All queries should return results within 5 seconds.	Queries returned results in under 5 seconds.	Pass
DT_005	DTD_005	No SQL injection is successful; inputs are sanitized.	SQL injection attempt blocked; input sanitized.	Pass
DT_006	DTD_006	Data is fully restored to its state prior to deletion.	Data successfully restored; no records lost.	Pass

Table 6.4.2-4 Result and Analysis For System Testing

Test Case ID	Test Data ID	Expected Result	Actual Result	Pass/Fail
ST_001	STD_001	Each user role logs in and accesses the correct page.	All users logged in successfully; correct redirection.	Pass
ST_002	STD_002	Property added and retrieved correctly.	Property details added and retrieved accurately.	Pass
ST_003	STD_003	Search results accurately reflect the filters applied.	Search filters correctly applied; accurate results returned.	Pass
ST_004	STD_004	Booking process completes successfully; confirmation received.	Booking completed successfully; confirmation received.	Pass
ST_005	STD_005	System remains responsive under high load.	System performance maintained; no degradation noted.	Pass
ST_006	STD_006	No security breaches; system prevents unauthorized access.	Security protocols effective; no breaches occurred.	Pass
ST_007	STD_007	Data fully restored to its original state after recovery.	Data accurately restored following simulated loss.	Pass

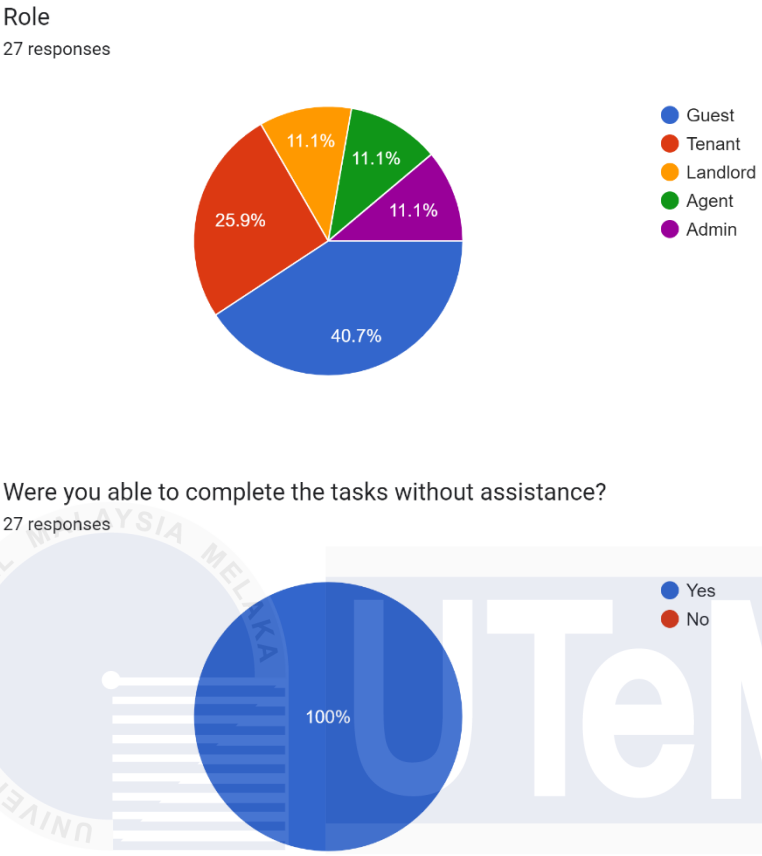


Figure 6.5-1 User Acceptance Test Result and Analysis

i. Role Distribution Pie Chart:

- Guest: 40.7% of the respondents identified as guests.
- Tenant: 25.9% of the respondents identified as tenants.
- Landlord: 11.1% of the respondents identified as landlords.
- Agent: 11.1% of the respondents identified as agents.
- Admin: 11.1% of the respondents identified as administrators. This chart indicates the distribution of different user roles among the respondents, with the largest group being

ii. Task Completion Pie Chart:

- Yes: 100% of the respondents indicated that they were able to complete the tasks without assistance. This chart indicates that all respondents were able to perform their tasks independently, suggesting a high degree of user-friendliness or simplicity in the system's interface or task design.

6.6 Conclusion

In this task, testing procedure is behaviour to guarantee the House Rental Management System (DreamHouse) meets the necessities that are now expressed in the past section. The techniques utilized as a part of this testing is front end and back end testing, which is more concern on the rightness yield by framework. This part have talked about the test arrangement and how the testing has done and the outcomes that has been record. Plus, this part likewise portrayed the testing of the individual modules of the framework to guarantee that meet the client prerequisites.

CHAPTER 7: PROJECT CONCLUSION

7.1 Introduction

This chapter summarizes the overall findings, observations, and conclusions drawn from the development of the House Rental Management System (DreamHouse). It reflects on the strengths and weaknesses of the project, proposes potential improvements, and highlights the contributions the project has made. Additionally, this chapter evaluates whether the project has met the objectives set out at the beginning.

7.2 Observation on Weakness and Strengths

After deploying the DreamHouse House Rental Management System, it's clear that the system significantly enhances how organizations manage property and contract information. Admins and agents benefit immensely, as they can easily handle property registrations, contract management, and appointments, making this method far more efficient compared to the old paper-based logbook records. Users are able to update most of their personal details independently in the system, promoting greater autonomy. However, certain sensitive information such as IC numbers and email addresses require admin intervention for updates, which could hinder user independence and delay necessary changes.

Additionally, the system lacks a feature for direct communication within the platform. Currently, guests and tenants must use external platforms like WhatsApp to contact agents, and agents are unable to directly update landlords through the system. Integrating a live chat feature could greatly improve operational efficiency and user

satisfaction by facilitating direct interactions among all users, thereby enhancing the overall functionality and user experience of the system.

7.3 Propositions for Improvement

While the House Rental Management System (DreamHouse) effectively manages property and contract information, there are a few areas where enhancements could further improve the system's functionality and user experience.

1. Live Chat Feature

    - Adding a live chat feature within the system would allow guests, tenants, and agents to communicate directly without relying on external messaging apps like WhatsApp. This would streamline communication, making it easier for agents to assist potential tenants and collaborate with landlords on property-related matters.

2. Expanded User Management Capabilities

    - Although users can manage most of their information, they are currently unable to change their IC number. Expanding the system to allow users to request updates to this information, with proper validation and approval processes in place, would enhance user control over their personal data.

3. Improved Communication Tools

    - Integrating an internal messaging system that allows tenants, landlords, and agents to communicate within the platform would centralize all communication related to rental agreements, property

management, and contract updates, ensuring that all parties stay informed without needing external tools.

4. Mobile Optimization

- Further optimizing the system for mobile devices would enhance accessibility, allowing users to manage their rental activities conveniently on their phones, ensuring that they can access the system anytime, anywhere.

7.4 Project Contribution

The House Rental Management System (DreamHouse) contributes significantly to both the academic and practical realms. Academically, it showcases the application of modern web development techniques using Laravel, offering a comprehensive solution for managing rental properties. This project can serve as a valuable reference for future students and researchers interested in developing similar management systems or working with Laravel as a framework.

From a practical perspective, DreamHouse streamlines the rental management process for landlords, agents, and tenants, reducing the reliance on manual, paper-based methods. It enhances efficiency by providing a centralized platform where property and contract information can be managed with ease. The system's ability to manage different user roles and automate key tasks such as property assignment and contract creation contributes to the overall improvement of rental operations.

Furthermore, the user manual, included in **Appendix XX**, provides detailed guidance for administrators and users on how to navigate and utilize the system, ensuring that it can be easily adopted by any organization or individual looking to implement a similar solution.

7.5 Conclusion

In conclusion, the House Rental Management System (DreamHouse) has successfully achieved the primary objectives set at the beginning of the project. The system has streamlined property and contract management, making it easier for landlords, agents, and tenants to handle rental processes efficiently. By centralizing various user roles and automating key tasks, DreamHouse has proven to be an effective tool for improving rental operations and reducing the reliance on manual methods.

While the system excels in its core functionalities, there are opportunities for future improvements, such as integrating live chat features and enhancing user management capabilities. These enhancements would further elevate the user experience and system efficiency. Overall, the project demonstrates the successful application of web development techniques using Laravel, providing a strong foundation for future refinement and potential real-world implementation.

REFERENCES

Buradkar, K., Kori, S., Ruikar, S., Galfat, V., Patil, D., & Nasare, R. (2022). Property Rental Management System. *International Journal of Computer Science and Mobile Computing*, *11*(11), 177–179. https://doi.org/10.47760/ijcsmc.2022.v11i11.014

Hamilton, T. (2024b, April 12). What is Integration Testing? (Example). Guru99 https://www.guru99.com/integration-testing.html

Omkar Sheshraoji Kosare, Minal Vinod Sawarkar, Ketaki Dattaraj Sewatkar, & Pratik Ladekar. (2023). Rental Property Management System. *International Journal of Advanced Research in Science, Communication and Technology*, 273–275. https://doi.org/10.48175/ijarsct-9298

Boland, M. (2021). Airbnb property management: Performance Evaluation of a rental property*. *Accounting Perspectives*, *20*(2), 255–263. https://doi.org/10.1111/1911-3838.12249

Aliu, I. R. (2023). Urban Property Markets and Security Risk: Explaining how neighborhood security shapes housing rental prices in Ojo Lagos, Nigeria. *Property Management*, *41*(3), 404–430. https://doi.org/10.1108/pm-09-2022-0070

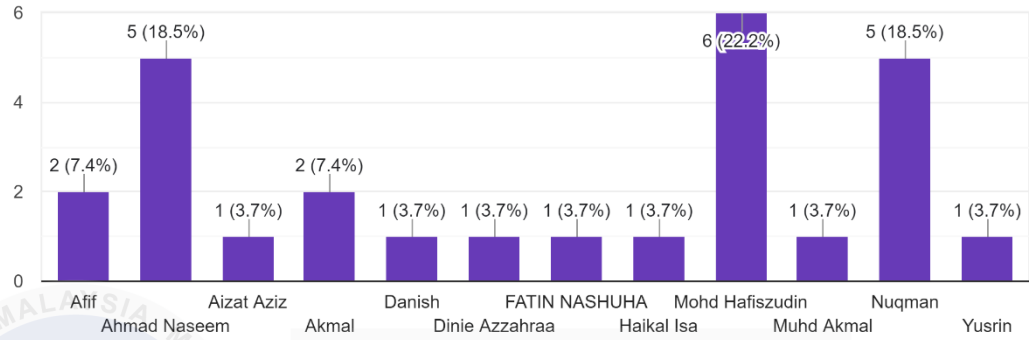
Singapore property, property for sale/rent, singapore real estate - propertyguru singapore. (n.d.). https://www.propertyguru.com.sg/

Property matrix: Property management software. Property Matrix | Property Management Software. (n.d.). https://www.propertymatrix.com/

APPENDIX A

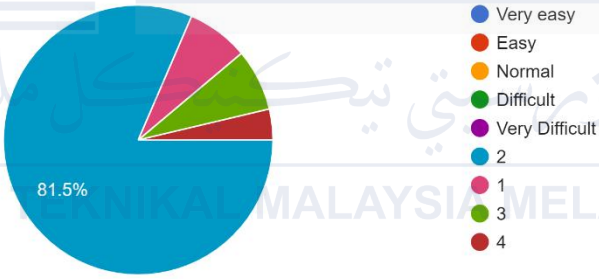
Name

27 responses



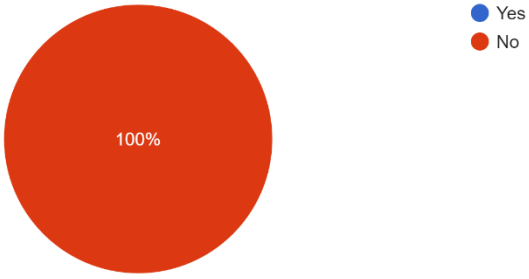
How intuitive did you find the navigation of our system?

27 responses



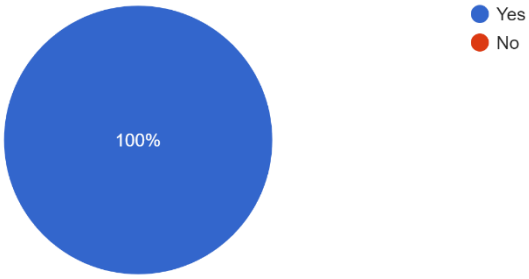
Did you encounter any issues while using the system?

27 responses



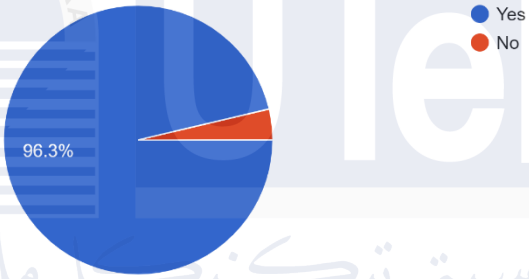
Was the property search function up to your expectations?

27 responses

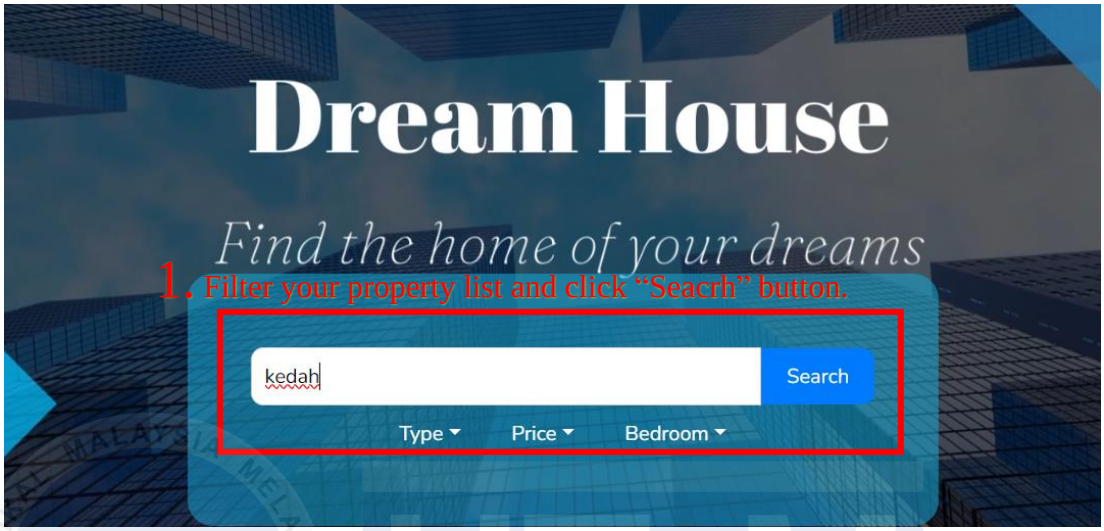


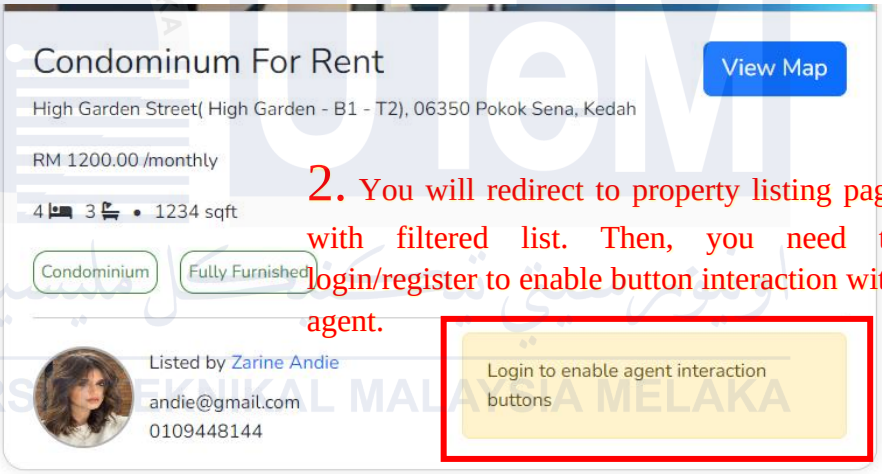
Was the information about each property clear and sufficient?

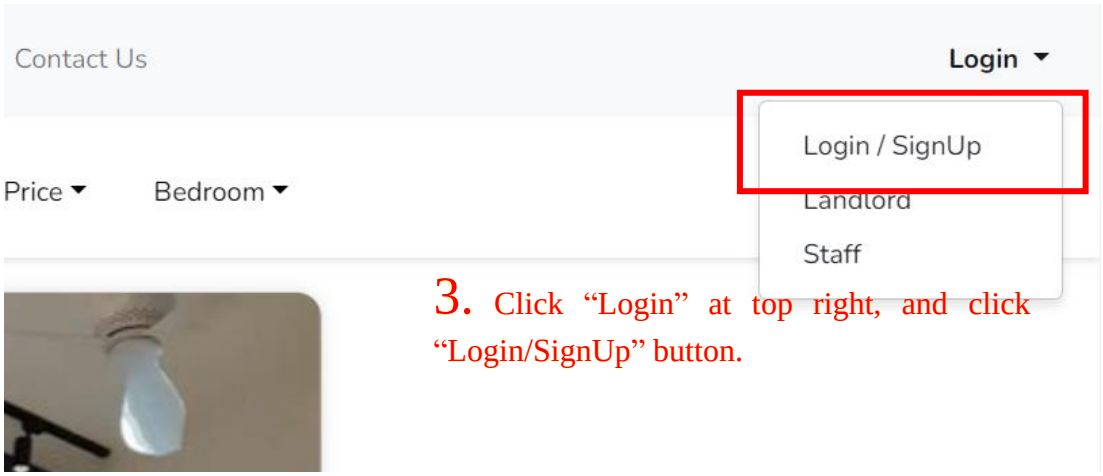
27 responses



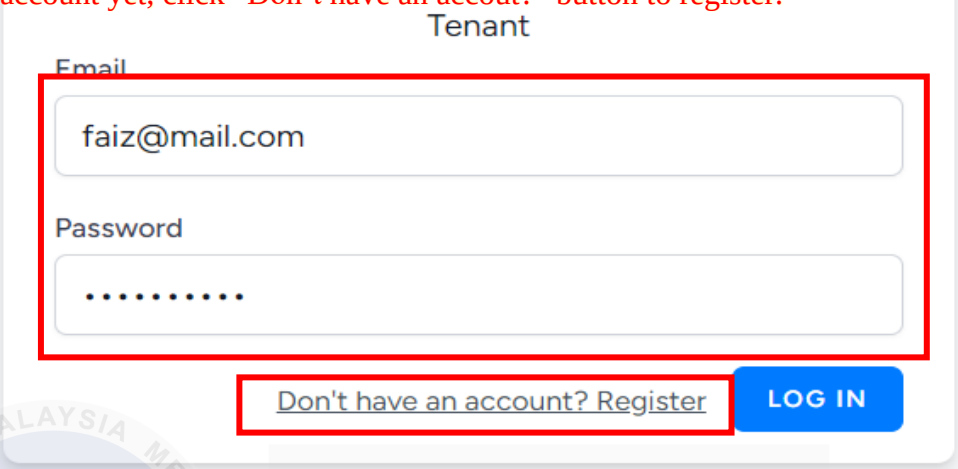
APPENDIX B

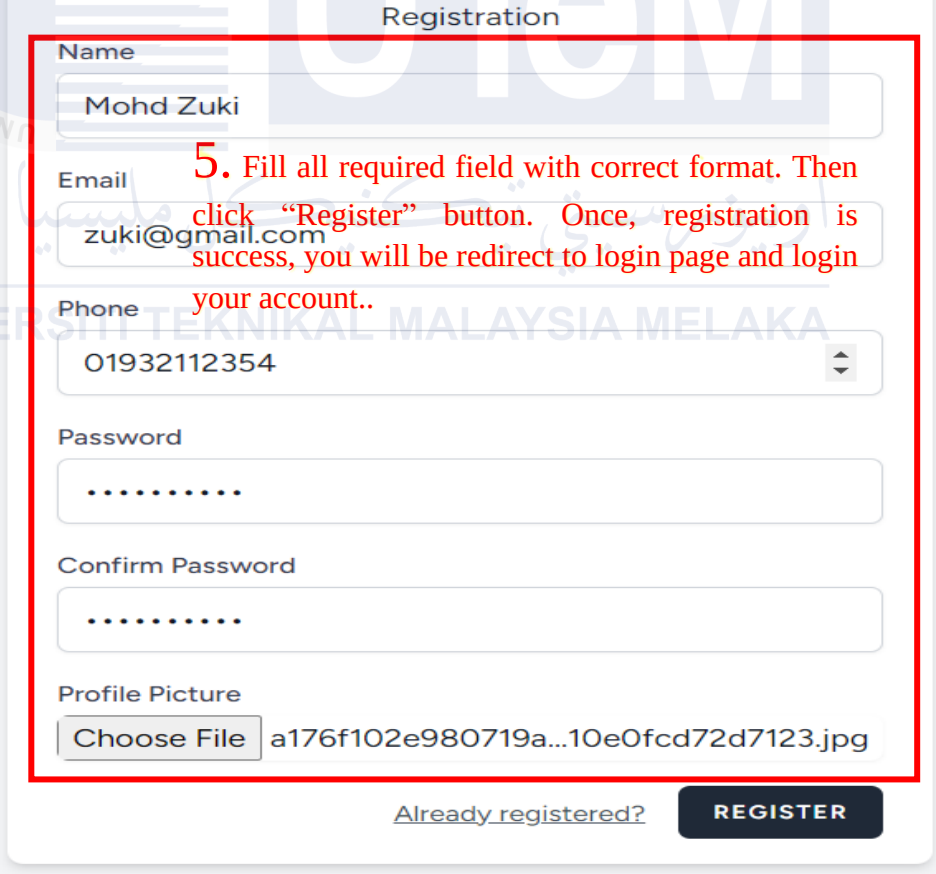




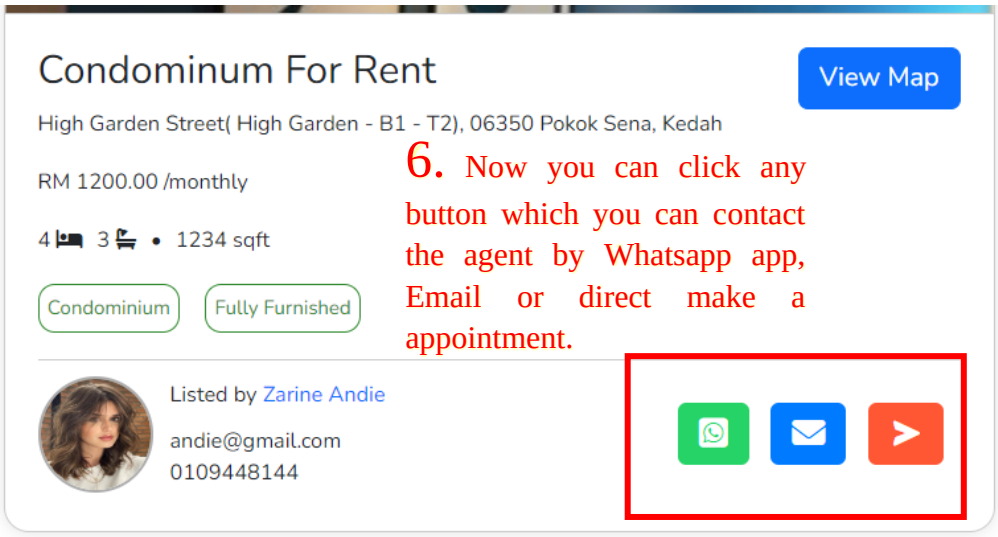


4. Fill the required field to login credential. If you don't have any account yet, click "Don't have an accout?" button to register.

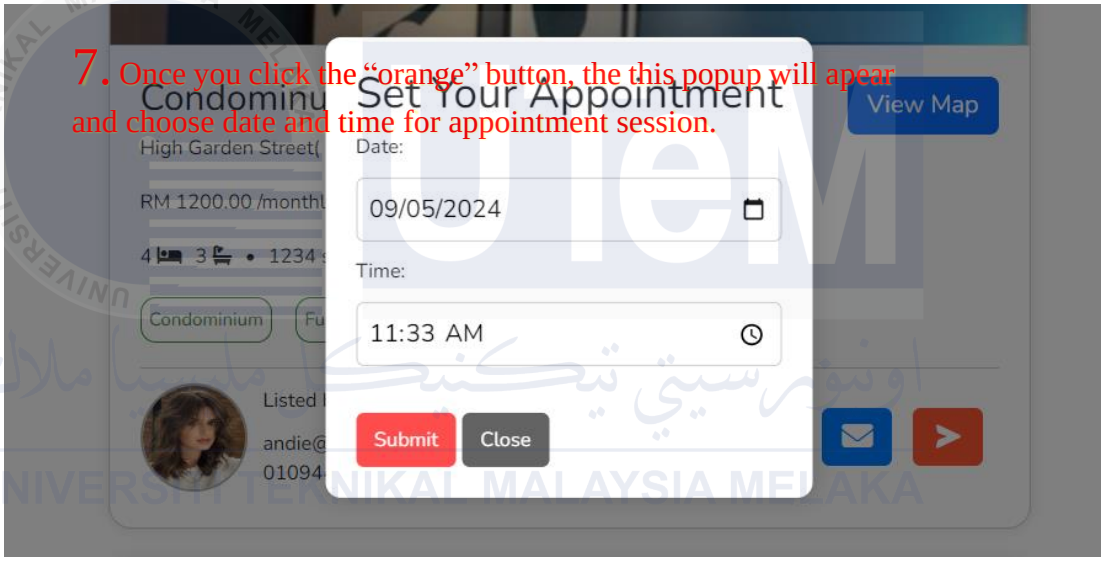




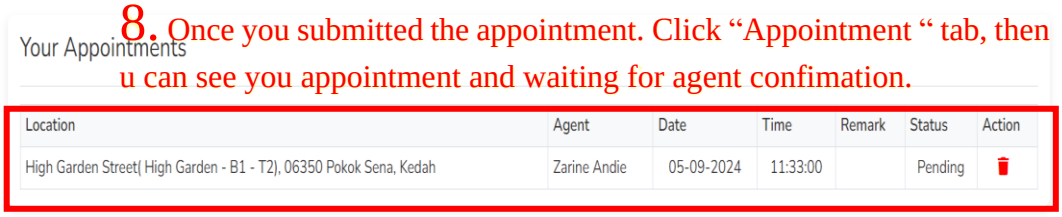
5. Fill all required field with correct format. Then click "Register" button. Once, registration is success, you will be redirect to login page and login your account..



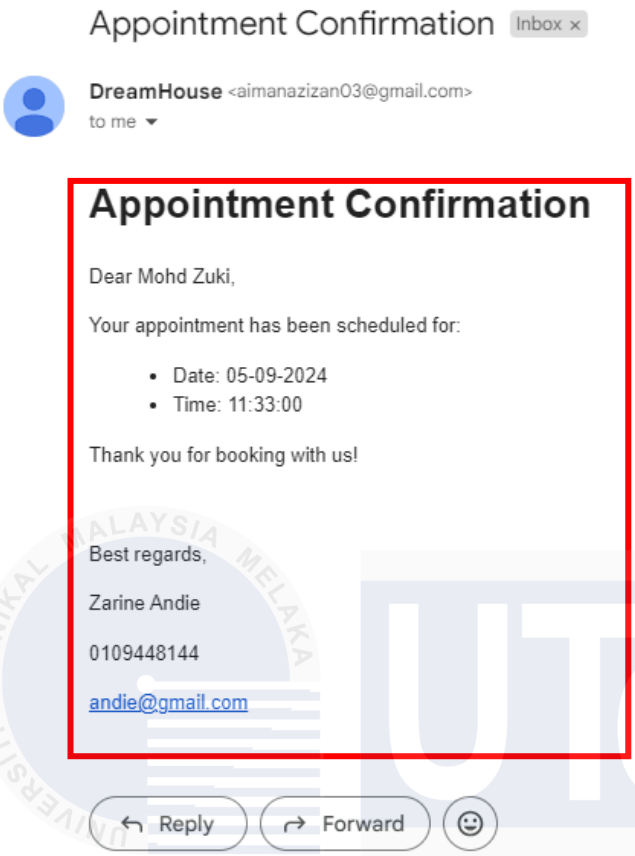
6. Now you can click any button which you can contact the agent by Whatsapp app, Email or direct make a appointment.



7. Once you click the "orange" button, the this popup will apear and choose date and time for appointment session.



8. Once you submitted the appointment. Click "Appointment " tab, then u can see you appointment and waiting for agent confimation.



9. Once agent have approve your appointment, an email will sent to you about your appointment have been submitted.