VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

PHAN HUYNH DANG KHOA – 523H0042
NGUYEN HUYNH HAI DANG – 523H0010
LE MINH KHA – 523H0036

ROOM RENTAL MANAGEMENT SOFTWARE

FINAL REPORT

SOFTWARE ENGINEERING

HO CHI MINH CITY, 2025

VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

PHAN HUYNH DANG KHOA – 523H0042
NGUYEN HUYNH HAI DANG – 523H0010
LE MINH KHA – 523H0036

ROOM RENTAL MANAGEMENT SOFTWARE

FINAL REPORT

SOFTWARE ENGINEERING

Advised by
M.S. Nguyen Ngoc Phien

HO CHI MINH CITY, 2025

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to Ton Duc Thang University for providing an environment that fosters learning and growth. Deepest thanks to the Faculty of Information Technology and the Software Engineering Department for creating a challenging and stimulating curriculum that has prepared us for the future. Lastly, we would like to express our gratitude to M.S. Nguyen Ngoc Phien, your passion for teaching and unwavering support have left a lasting impact on us. Thank you for your patience, for sharing your knowledge, and for always encouraging us to push our boundaries. Your mentorship has been a cornerstone of our success.

Ho Chi Minh city, 10th May 2025.

Author

(Signature and full name)

Khoa

Phan Huynh Dang Khoa

Dang

Nguyen Huynh Hai Dang

Kha

Le Minh Kha

DECLARATION OF AUTHORSHIP

We hereby declare that this is our project and is guided by M.S. Nguyen Ngoc Phien; The content research and results contained herein are central and have not been published in any form before. The data in the tables for analysis, comments and evaluation are collected by the main author from different sources, which are clearly stated in the reference section.

In addition, the project also uses some comments, assessments as well as data of other authors, other organizations with citations and annotated sources.

If something wrong happens, we'll take full responsibility for the content of our project. Ton Duc Thang University is not related to the infringing rights, the copyrights that i give during the implementation process (if any).

Ho Chi Minh city, 10th May 2025

Author

(Signature and full name)

Khoa

Phan Huynh Dang Khoa

Dang

Nguyen Huynh Hai Dang

Kha

Le Minh Kha

TABLE OF CONTENT

LIST OF TABLES

LIST OF IMAGES

CHAPTER 1. INTRODUCTION

1.1 Purpose and Scope

This document serves as a detailed introduction to the Room Rental Management Software. This tool is engineered to make managing room rentals easier for landlords and administrators by simplifying key operations. Specifically, it focuses on making tenant management, payment tracking, rental arrangement, and property upkeep scheduling more efficient. This report details what the software intends to achieve, what it can do, and its technical details. The goal is to help everyone involved – from stakeholders to developers and users – understand its benefits and how to use it.

The software is designed to manage several rental properties, address tenant questions, automatically send payment reminders, create financial summaries, and offer an easy-to-use platform for both property owners and renters. While it's built for small to medium-sized rental businesses, it can adapt to larger ones. However, the software's focus is strictly on room rental management and does not include tasks like real estate sales or construction project management.

1.2 Product Overview

This software for managing room rentals is equipped with a wide range of robust features aimed at boosting efficiency and improving organization :

- **Tenant Records:** Effortlessly maintain all essential tenant data, such as contact information, financial details, and past rental activity.
- **Availability and Reservations:** Keep a close eye on which rooms are free and efficiently handle the booking process.
- **Integrated Payments:** Seamlessly process rent payments through connection with widely used online payment systems
- **Repair Coordination:** Allow tenants to easily report maintenance issues and streamline the scheduling of necessary repairs.

- **Insightful Reports**: Gain valuable understanding of your business through generated reports on income, occupancy levels, and expenditures, enabling better decisions.
- **Tenant Self-Service**: Empower renters with an online portal to access their lease information, request maintenance, and submit payments independently.
- **Centralized Property Control**: Oversee numerous properties from one central system, with the flexibility to adjust settings for each individual location.

The product can be used in the following scenarios :

- **Small Landlords :** A landlord with a few rental units can benefit from the software's ability to track payments, manage renting and schedule maintenance for their limited number of units.
- **Co-living Spaces** : Co-living operators can streamline short-term rentals and tenant turnover using the integrated booking features.
- **Student Housing** : Student housing providers (universities or private entities) can manage room assignments, automate billing, and handle maintenance for dorms and off-campus housing with this software.

1.3 Structure of the Document

This document is organized into the following chapters:

- **Chapter 1 : Introduction** - Provides an overview of the software, its purpose, scope, and key terms
- **Chapter 2 : Project Management Plan** - Outlines the project’s objectives, timeline, resources, and risk management strategies.
- **Chapter 3 : Requirement Specifications** - Details the functional and non-functional requirements of the software.
- **Chapter 4 : Architecture** - Describes the system architecture

- **Chapter 5 : Design** - Explains the database structure, and other design elements.
- **Chapter 6 : Test Plan** - Outlines the testing methodologies and procedures to ensure software reliability.
- **Chapter 7 : Demo** - Presents a demonstration of the software's features and functionality.
- **References** - Lists all sources and materials referenced in the document.

1.4 Terms, Acronyms, and Abbreviations

- **RRMS**: Room Rental Management Software
- **Tenant**: An individual or entity renting a room or property.
- **Landlord**: The property owner or manager responsible for renting rooms.
- **UI**: User Interface
- **FR** : Functional Requirement
- **NFR** : Non-functional Requirements

CHAPTER 2. PROJECT MANAGEMENT PLAN

2.1 Project Organization

The Room Rental Management Software (RRMS) project is developed by a team of three students as part of a final exam for a software engineering course. Below are the roles that we share to ensure effective collaboration:

- **Nguyễn Huỳnh Hải Đăng :** Backend Development, Testing.
- **Lê Minh Kha :** Backend Development & Database
- **Phan Huỳnh Đăng Khoa :** Documentation, Frontend & UI/UX Design

The project is supervised by the course instructor. The team conducts weekly meetings to discuss progress, resolve issues, and plan ahead. Online tools facilitate collaboration among the student team due to their academic schedules.

2.2 Lifecycle Model Used

An adapted Agile Scrum methodology is being used for this student project, which has a fixed deadline. Development occurs in one-week sprints, each resulting in a small, functional software feature. The main practices include:

- **Sprint Planning**: At the beginning of each week-long development cycle, the team defines the objectives and the specific tasks to be accomplished.
- **Daily Check-ins**: Brief, regular team meetings are held to share progress updates and identify any obstacles hindering work.
- **Sprint Reviews**: Completed software increments are presented to the team and the instructor to gather feedback and ensure alignment.
- **Retrospectives**: Following each week's development sprint, the team reflects on what worked effectively and areas where improvements can be made.

By following this method, our team can incorporate feedback and learn through each iteration, ensuring timely completion within the academic schedule.

2.3 Risk Analysis

Potential risks and mitigation strategies include:

- **Time Constraints**: The team faces limited time due to other academic commitments. **Mitigation**: To address this, core functionalities like booking will be prioritized, and a rigid project timeline will be established and adhered to.
- **Technical Difficulties**: The team anticipates potential challenges when learning new technologies such as Git or Tailwind CSS,... **Mitigation**: To overcome these hurdles, the team will utilize online learning resources and proactively seek guidance from the instructor.
- **Team Coordination**: Late submission of work by team members poses a risk to project timelines. **Mitigation**: To ensure timely progress, team members will be strongly encouraged to meet deadlines. Mutual support will be offered, and regular meetings will be held to track progress and address any potential delays.

The team uses a shared document to track potential risks, and this log is reviewed during weekly meetings.

2.4 Hardware and Software Resource Requirements

Hardware :

- For development and testing purposes, each member will utilize their personal laptop, which meets the minimum requirements of 8GB RAM, a dual-core processor, and 256GB storage.
- The mobile-responsive tenant portal will be tested on the personal smartphones (both iOS and Android) of the team members.

Software :

- Development Tools: Visual Studio Code (IDE), Git (version control), PHP (backend), Tailwind CSS (frontend), and MySQL(database).
- Collaboration Tools: Zalo and Discord for communication.

2.5 Deliverables and Schedule

To fit within the academic semester, the project timeline is set for one month, from April to May 2025 :

- **Phase 1: Planning (April 2025)** - Deliverables include the chosen project and initial requirements. Duration: 1 week.
- **Phase 2: Design (April 2025)** - This phase will produce UI wireframes, the database schema, and a basic architecture diagram. Duration: 1 week.
- **Phase 3: Development (April - May 2025)** - The primary output will be the core software modules (tenant management, booking, payment tracking) along with the complete source code. Duration: 3 weeks.
- **Phase 4: Testing (May 2025)** - This phase will generate test cases, bug reports, and a summary of the test results. Duration: 2 days.
- **Phase 5: Final Submission (May 2025)** - The final deliverables will be the fully functional software, the comprehensive final report. Duration: 2 days.

2.6 Monitoring, Reporting, and Controlling Mechanisms

- **Monitoring**: Weekly team meetings will take place. Code is reviewed by peers to catch errors early.
- **Reporting**: Progress reports once a week, checking the percentage of completion of assigned tasks
- **Controlling**: Project manager resolves issues between members and assists if necessary

2.7 Professional Standards

The team's approach is grounded in academic and basic software development standards. Specifically:

- The ACM Code of Ethics will be followed to ensure honesty, fairness, and respect in all teamwork and documentation.
- Basic JavaScript style guides will be used to maintain consistency throughout the codebase.
- Academic Integrity will be upheld by ensuring all work is original and properly citing any tutorials or libraries used.

2.8 Evidence All Artifacts Have Been Placed Under Configuration Management

All project artifacts (e.g., code, documentation, designs) are stored in a private Git repository on GitLab :

- **Version Control**: Commits are labeled clearly.
- **Branching**: A main branch for stable code and feature branches for new work.
- **Access Control**: All team members have access, with pull requests for code merges.
- **Backup**: Weekly exports of the repository are saved to Google Drive.

2.9 Impact of the Project on Individuals and Organizations

Individuals :

- This project offers students the opportunity to develop practical skills in software creation, team collaboration, and project oversight, which will enhance their resumes and provide valuable preparation for their future careers.
- Hypothetically, small-scale landlords would find the software useful for simplifying tasks like payment tracking, while tenants would

appreciate the ease of submitting maintenance requests, ultimately saving time for both parties.

Organizations :

- This software is envisioned as an affordable solution for small rental businesses (Hypothetical), such as local landlords or student housing providers, to manage their properties more efficiently.

Society :

- The project provides students with real-world software development experience, fostering tech industry skills. By simulating a rental management tool, it showcases technology's ability to improve housing accessibility and efficiency, particularly for small local rental operations.

CHAPTER 3. REQUIREMENT SPECIFICATIONS

3.1 Stakeholders for the System

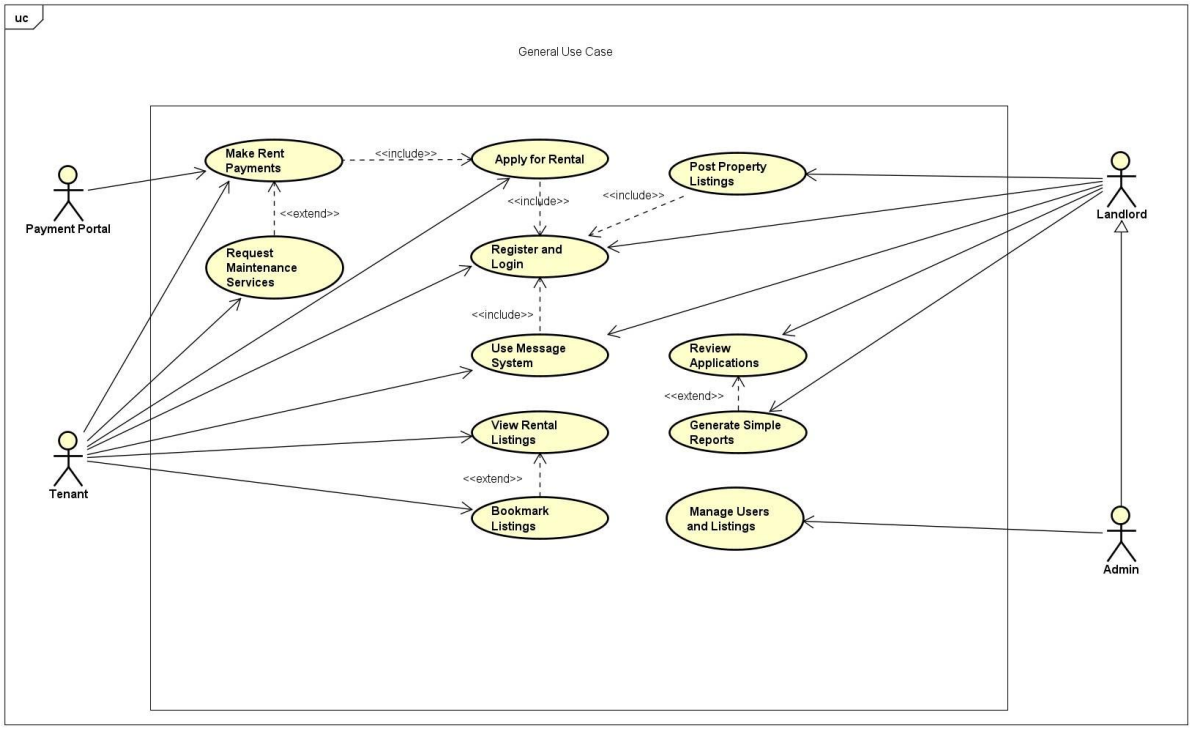
The RRMS serves the following stakeholders:

- Landlords : Primary users who manage room listings, track payments, and handle maintenance through the system.
- Tenants: Secondary users who use a self-service portal to view leases, pay rent, and submit maintenance requests.
- Admin: Manages user accounts, maintains system performance, and ensures platform reliability.
- Payment Portal : A secure online platform that allows tenants to make rent payments and view payment history.

3.2 Use Case Model

3.2.1 Graphical Use Case Model

The following general use case diagram outlines the key ways in which individuals like Landlords, Tenants, and Admin interact with the RRMS :



3.2.2 Textual Description for Each Use Case

3.2.2.1 Register and Login

Use Case Name:	Register and Login
Scenario:	User want to use the website for renting, Admin login to manage.
Triggering Event:	When user choose to login or when choose to register.
Brief Description:	The user can sign in with their existing credentials. The system authenticates the user during the sign-in process or creates a new account during the sign-up process.
Actors:	Tenant, Landlord, Admin
Related Use Cases:	**Include:** Make Rent Payments, Post Property Listings
Stakeholders:	Payment Portal
Preconditions:	The user must have valid credentials for sign-in (username and password) or provide correct registration information during sign-up. The user must not already be logged in. The user's credentials must match what is stored in the database if signing in.
Postconditions:	Sign In: The user gains access to their account and is redirected to the home screen.

	Sign Up: A new user account is created, and the user is redirected to the home screen.	
Flow of Events:	**Actor**	**System**
	1. User enters their credentials (username and password) or opts for sign-up.	1.1 System accepts the entered credentials.
		2. System validates the credentials (for sign-in) or creates a new account (for sign-up). 2.1 For sign-in, the system checks if the credentials match the database records. For sign-up, it checks if the provided details are valid.
	3. If sign-in is successful, the user is granted access. If sign-up is successful, a new account is created.	3.1 System grants or denies access based on authentication results.
	4. The system redirects the user	4.1 The system shows a success

	to the appropriate page (home or profile).	message and redirects the user as appropriate.
Exception:	**Sign in:** If the credentials do not match any records, the system notifies the user with an error message. **Sign up:** If the email is already registered, the system prompts the user to log in instead. If the user inputs invalid or incomplete details, the system prompts the user to fix the issue.	

Table 1. Register and Login Description

3.2.2.2 Make Rent Payments

Use Case Name:	Make Rent Payments
Scenario:	A Tenant wants to make a rent payment online for their selected rental property.
Triggering Event:	The Tenant presses the "Make Payment" button after selecting a rental.
Brief Description:	After logging in and selecting a rental property, the Tenant proceeds to make an online rent payment via an integrated Payment Portal. The system verifies credentials, processes the payment, and notifies the Landlord and Admin upon successful transaction.

Actors:	Tenant, Payment Portal	
Related Use Cases:	**Include:** Apply for rental **Extend**: Request Maintainance Services	
Stakeholders:	Admin, Landlord	
Preconditions:	Tenant must be registered or signed in. Tenant must have selected a rental listing. A valid payment method must be available and active. Internet connection must be stable.	
Postconditions:	Payment is processed, and a confirmation receipt is generated. Landlord and Admin are notified of the successful transaction. Tenant receives a confirmation email or notification in-app.	
Flow of Events:	**Actor**	**System**
	1. Tenant selects a rental to pay for.	1.1 System displays payment options.
	2. Tenant clicks "Make Rent Payment".	2.1 System opens the payment interface.
	3. Tenant selects a payment method and enters details.	3.1 System validates the information.

	4. Tenant confirms the payment.	4.1 System processes payment via Payment Portal.
		5. System displays confirmation and receipt. 5.1 System notifies Landlord and Admin of successful payment.
	6. Tenant receives a message/receipt.	6.1 System logs the payment in the database.
Exception:	3.2 If payment method is declined, system notifies Tenant. 4.2 If payment portal is down, system shows a retry message.	

Table 2. Make Rent Payments Description

3.2.2.3 Use Message System

Use Case Name:	Use Message System
Scenario:	A Tenant wants to send a message to the Landlord to ask about a rental issue.
Triggering Event:	The Tenant clicks the “Send Message” button after typing the content.
Brief Description:	After logging in and selecting a rental, the Tenant composes a message and sends it to the Landlord using the

	system’s built-in messaging feature. The system saves the message, notifies the Landlord, and logs it in both users' message histories.	
Actors:	Tenant, Landlord, Admin	
Related Use Cases:	**Include:** Register and Login **Extend:** Request Maintenance Services	
Stakeholders:	Admin, Landlord	
Preconditions:	User must be logged in. Tenant can only send message to Landlord and Admin. User has access to the messaging system. Internet connection must be active.	
Postconditions:	Message is delivered to the receiver. System saves the message to the database. Users can view the message in their message history.	
Flow of Events:	**Actor**	**System**
	1. User selects the rental property.	1.1 System loads rental details with messaging option.
	2. User clicks "Send Message".	2.1 System opens message composition interface.
	3. User types the message content.	3.1 System waits for submission.

	4. User presses “Send”.	4.1 System validates input and sender/receiver relationship. 4.2 System saves the message to database. 4.3 System notifies receiver. 4.4 System logs the message for future retrieval.
	5. User is shown confirmation.	5.1 System updates message history UI.
Exception:	4.1.1 If message content is empty, system shows error: "Message cannot be empty." 4.3.1 If notification service fails, system retries or marks message as "Sent, no notification".	

Table 3. Use Message System Description

3.2.2.4 View Rental Listings

Use Case Name:	View Rental Listings
Scenario:	A tenant searches for rental listings by using search filters and views detailed information of the listings
Triggering Event:	A tenant wants to find a rental property that fits specific needs
Brief Description:	A tenant accesses the rental listing system, enters search filters such as location and price, and views the list of matching rental properties. From there, the tenant can view the detailed information of each listing

Actors:	Tenant	
Related Use Cases:	Include: Search Listings, View Listing Details	
Stakeholders:	Tenants seeking rental homes Landlords listing their properties Admin managing listing data	
Preconditions:	The tenant is logged into the system The rental listing database is updated	
Postconditions:	The system displays a filtered list of rental listings Tenant can view more details or proceed to further actions like bookmarking or applying	
Flow of Events:	**Actor**	**System**
	1. Tenant accesses the rental listing feature	
	2. Tenant provides search filters	
	3. Tenant submits search	3.1 Retrieve matching listings from the database 3.2 Display filtered list
	4. Tenant selects a listing	4.1 Show detailed information of the selected listing
Exception Conditions:	3.1 If no listings match the filters, the system informs the user and suggests adjusting the criteria 3.2 If a system/database error occurs, an error message is shown	

Table 4. View Rental Listings Descriptions

3.2.2.5 Bookmark Listings

Use Case Name:	Bookmark Listings
Scenario:	A tenant wants to save interesting rental listings for future reference
Triggering Event:	A tenant finds a listing and wants to save it or manage their saved listings

<table>
<tr><td>Brief Description:</td><td colspan="2">A tenant browses rental listings and uses the bookmarking feature to save listings of interest. The tenant can later view or remove saved listings from the bookmark list</td></tr>
<tr><td>Actors:</td><td colspan="2">Tenant</td></tr>
<tr><td>Related Use Cases:</td><td colspan="2">Include: Add Listing to Bookmarks, View Bookmarked Listings
Extend: Remove Listing from Bookmarks</td></tr>
<tr><td>Stakeholders:</td><td colspan="2">Tenants looking to manage preferred listings
System admin ensuring data consistency</td></tr>
<tr><td>Preconditions:</td><td colspan="2">The tenant is logged into the system
There are rental listings available</td></tr>
<tr><td>Postconditions:</td><td colspan="2">Listings are added to or removed from the tenant's personal bookmark list
Tenant can access the list of saved listings anytime</td></tr>
<tr><td rowspan="5">Flow of Events:</td><td>Actor</td><td>System</td></tr>
<tr><td>1. Tenant views a rental listing</td><td></td></tr>
<tr><td>2. Tenant selects "Bookmark"</td><td>2.1 Add listing to tenant's bookmark list</td></tr>
<tr><td>3. Tenant accesses bookmark section</td><td>3.1 Display all bookmarked listings</td></tr>
<tr><td>4. Tenant removes a listing from bookmarks</td><td>4.1 Update the bookmark list and reflect changes</td></tr>
<tr><td>Exception Conditions:</td><td colspan="2">2.1 If the listing is already bookmarked, the system notifies the user
3.1 If no listings are bookmarked, the system shows an empty state message
4.1 If the listing is not found or already removed, an appropriate message is displayed</td></tr>
</table>

Table 5. Bookmark Listings Description

3.2.2.6 Apply for Rental

Use Case Name:	Apply for Rental	
Scenario:	A tenant decides to apply for a rental listing they are interested in	
Triggering Event:	A tenant selects a rental listing and chooses to initiate the application process	
Brief Description:	A tenant fills out a rental application form, submits it for review, and may cancel the application if needed before approval	
Actors:	Tenant	
Related Use Cases:	Include: Fill Rental Application Form, Submit Application Extend: Cancel Application	
Stakeholders:	Tenants wishing to secure a rental property Property managers reviewing applications	
Preconditions:	The tenant must be logged in A valid rental listing must be selected The listing must be open for applications	
Postconditions:	Application is submitted and recorded in the system Tenant receives feedback or status updates	
Flow of Events:	**Actor**	**System**
	1. Tenant selects a rental listing	
	2. Tenant starts the application process	2.1 System loads rental application form
	3. Tenant fills out the form	3.1 System captures and validates form inputs
	4. Tenant submits the application	4.1 System stores the application and notifies relevant parties
	5. Tenant cancels the application	5.1 System removes or flags the application as canceled

Exception Conditions:	3.1 If any required fields are missing or invalid, the system prompts the tenant to correct them 4.1 If submission fails due to a system or database error, an error message is displayed 5.1 If the cancellation deadline has passed, the tenant is notified that cancellation is no longer allowed

Table 6. Apply for Rental Description

3.2.2.7 Request Maintenance Services

Use Case Name:	Request Maintenance Services	
Scenario:	A tenant encounters an issue in their rental unit and wants to request maintenance	
Triggering Event:	A tenant selects the option to request maintenance via the system	
Brief Description:	A tenant creates a maintenance request detailing the issue, submits the request to the property manager or maintenance team, and may cancel the request if needed	
Actors:	Tenant	
Related Use Cases:	Include: Create Maintenance Request, Submit Request Extend: Cancel Request	
Stakeholders:	Tenants needing prompt issue resolution Maintenance staff who manage service tasks	
Preconditions:	Tenant is logged into the system Tenant has an active rental property assigned	
Postconditions:	A maintenance request is created and sent to the appropriate personnel Tenant may view status or cancel if allowed	
Flow of Events:	**Actor**	**System**
	1. Tenant accesses maintenance service page	
	2. Tenant creates a maintenance request by filling in issue details	2.1 System validates request form

	3. Tenant submits the request	3.1 System stores request and sends notification to maintenance team
	4. Tenant cancels the request	4.1 System marks request as canceled
Exception Conditions:	2.1 If any required fields are missing or invalid, the system prompts the user to correct them 3.1 If submission fails due to a system/database error, an error message is displayed 4.1 If the cancellation window has passed, cancellation is denied and tenant is notified	

Table 7. Request Maintenance Services Description

3.2.2.8 Post Property Listings

Use Case Name:	Post Property Listings
Scenario:	A landlord decides to post a property listing to make it available for tenants to view and apply for.
Triggering Event:	A landlord selects the option to create a new property listing and initiates the posting process.
Brief Description:	A landlord fills out a property listing form with details such as location, price, and description, submits it for posting, and may edit or remove the listing if needed before it is approved or viewed by tenants.
Actors:	Landlord
Related Use Cases:	Include: Register and Login Extend: Manage Users and Listings
Stakeholders:	Landlords wishing to list their properties for rent. Tenants who will view the listings. Admin overseeing the platform.
Preconditions:	The landlord must be logged in. The landlord must have the authority to post a listing.
Postconditions:	The listing must be successfully posted and visible to tenants in the system. Landlord receives confirmation of the posted listing.

Flow of Events:	**Actor**	**System**
	1. Landlord selects the option to post a new property listing.	1.1 System display form
	2. Landlord fills out the property listing form with details (e.g., location, price, description).	2.1 System loads the property listing form.
	3. Landlord submits the listing.	3.1 System validates the form inputs.
		3.2 System stores the listing and makes it visible to tenants.
		3.3 System notifies the landlord of successful posting.
Exception Conditions:	2.1 If any required fields are missing or invalid, the system prompts the landlord to correct them. 3.1 If submission fails due to a system or database error, an error message is displayed.	

Table 8. Post Property Listings Description

3.2.2.9 Review Applications

Use Case Name:	Review Applications
Scenario:	A landlord decides to review applications submitted by tenants for their posted property listings.
Triggering Event:	A landlord receives a notification of a new application and selects the option to review it.
Brief Description:	A landlord reviews tenant applications, evaluates the details provided, and decides to approve, reject, or request additional information, with the ability to cancel the review process if needed. Optionally, the landlord may generate a simple report on the applications reviewed.
Actors:	Landlord
Related Use Cases:	Include: Register and Login

	Extend: Manage Users and Listings, Generate Simple Reports	
Stakeholders:	Landlords reviewing tenant applications Tenants whose applications are under review Admin overseeing the application process	
Preconditions:	The landlord must be logged in. At least one valid application must be submitted for a property listing posted by the landlord.	
Postconditions:	The application status must be updated (e.g., approved, rejected, or pending) in the system. The tenant receives feedback or status updates from the landlord If a report is generated, it is saved and accessible to the landlord.	
Flow of Events:	**Actor**	**System**
	1. Landlord selects the applications to review.	
	2. Landlord views the application details	2.1 System loads the application details.
	3.1 Landlord evaluates and decides on the application (e.g., approve, reject, or request more info).	3.1 System validates the landlord's decision.
		3.2 System updates the application status.
		3.3 System notifies the tenant of the decision.
Exception Conditions:	2.1 If application details are unavailable or incomplete, the system prompts the landlord to request additional information. 3.1 If the decision submission fails due to a system or database error, an error message is displayed.	

Table 9. Review Applications Description

3.2.2.10 Generate Simple Reports

Use Case Name:	Generate Simple Reports	
Scenario:	A landlord decides to generate a simple report based on available data such as applications or listings.	
Triggering Event:	A landlord selects the option to generate a report and initiates the process.	
Brief Description:	A landlord selects criteria for a report (e.g., application status, listing details), generates it for review, and may save or cancel the process if needed.	
Actors:	Landlord	
Related Use Cases:	Extend: Review Applications	
Stakeholders:	Landlords needing reports on applications or listings Tenants indirectly affected by report outcomes (e.g., application decisions)	
Preconditions:	The landlord must be logged in. Relevant data (e.g., applications or listings) must be available in the system.	
Postconditions:	The report is generated and made available to the landlord. The report may be saved in the system for future reference if requested.	
Flow of Events:	**Actor**	**System**
	1. Landlord selects the option to generate a report.	
	2. Landlord specifies report criteria (e.g., application status, listing details).	2.1 System loads the report generation interface.
	3. Landlord generates the report.	3.1 System processes the criteria and generates the report.
		3.2 System displays the report to the landlord.

Exception Conditions:	2.1 If the report interface fails to load due to a system error, an error message is displayed. 3.1 If report generation fails due to insufficient data or a system error, an error message is displayed.

Table 10. Generate Simple Reports Description

3.2.2.11 Manage Users and Listings

Use Case Name:	Manage Users and Listings	
Scenario:	An admin decides to manage user accounts or property listings within the system.	
Triggering Event:	An admin selects the option to manage users or listings to perform actions such as editing, deleting, or updating records.	
Brief Description:	An admin accesses user accounts or property listings, performs management tasks such as updating, deleting, or reviewing records, and may cancel the process if needed.	
Actors:	Admin	
Related Use Cases:		
Stakeholders:	Admins overseeing the platform Landlords whose listings are managed Tenants whose accounts or applications may be affected	
Preconditions:	The admin must be logged in. Relevant data (e.g., user accounts or property listings) must exist in the system.	
Postconditions:	The user accounts or listings are updated, deleted, or reviewed as requested. Affected users (e.g., landlords or tenants) are notified of changes if applicable.	
Flow of Events:	**Actor**	**System**
	1. Admin selects the option to manage users or listings.	

	2. Admin chooses to view, edit, or delete a user account or listing.	2.1 System loads the relevant user or listing data.
	3. Admin performs the management action (e.g., update, delete).	3.1 System validates the action.
		3.2 System updates or removes the user/listing record.
		3.3 System notifies affected users (e.g., landlord or tenant) of the change if applicable.
Exception Conditions:	2.1 If the requested data fails to load due to a system error, an error message is displayed. 3.1 If the management action fails due to a system or database error, an error message is displayed.	

Table 11. Manage Users and Listings Description

3.2.3 Decomposition Use Cases

3.2.3.1 Register and Login

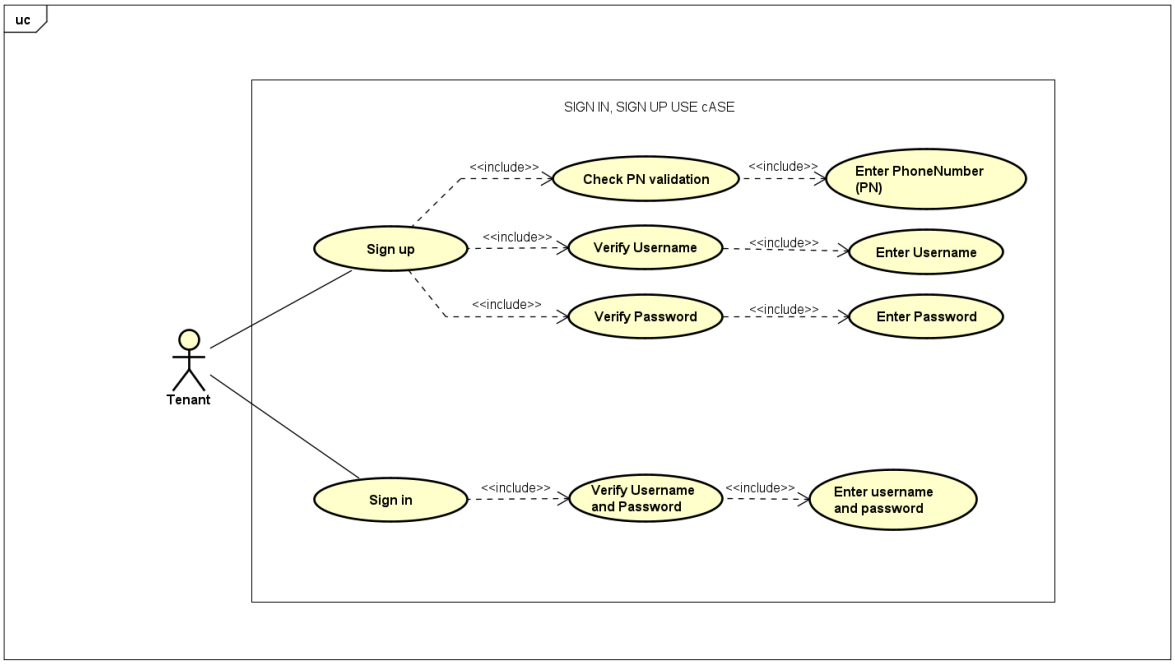


Figure 1. Register and Login Decompositon Use Case

3.2.3.2 Make Rent Payments

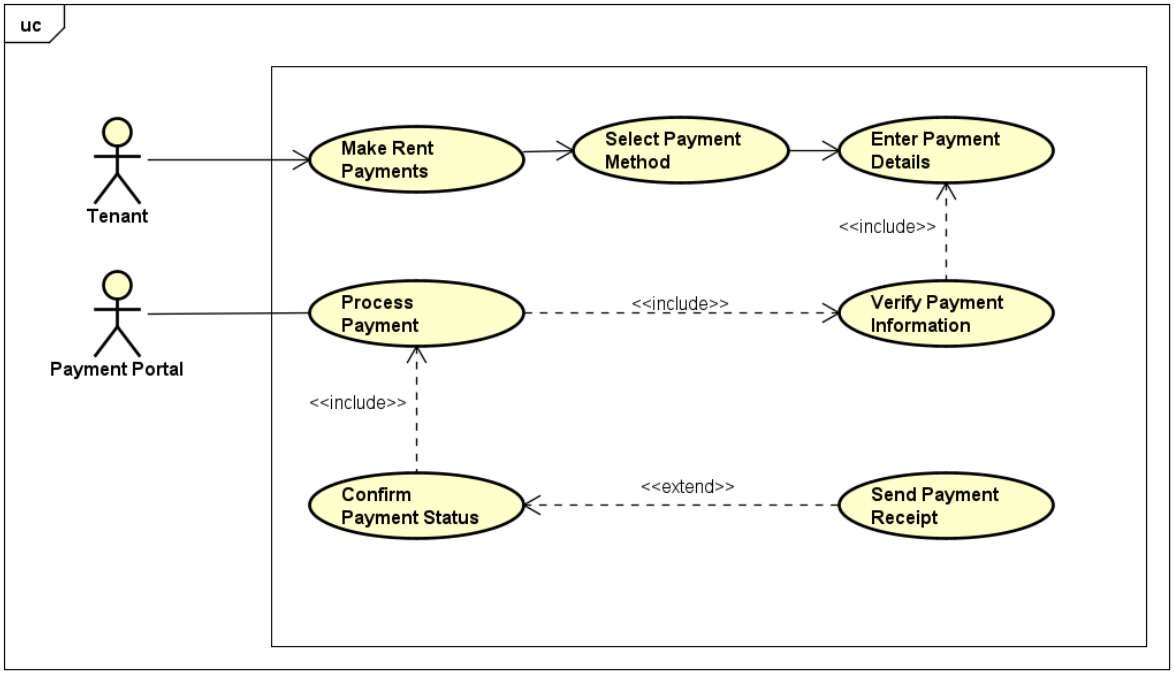


Figure 2. Make Rent Payments Decompositon Use Case

3.2.3.3 Use Message System

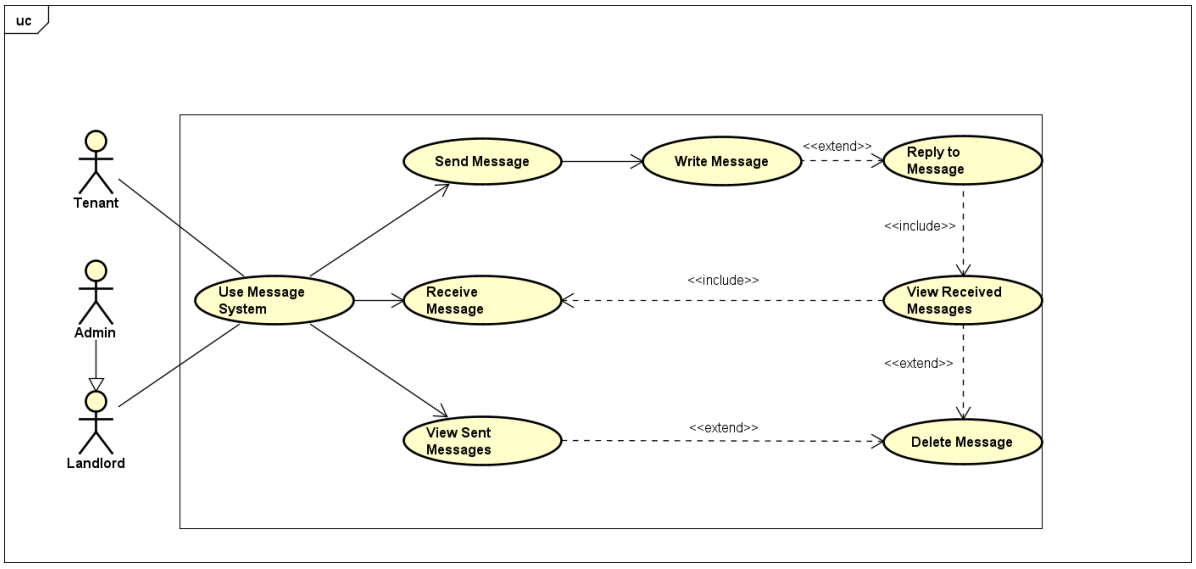


Figure 3. Use Message System Decompositon Use Case

3.2.3.4 View Rental Listings

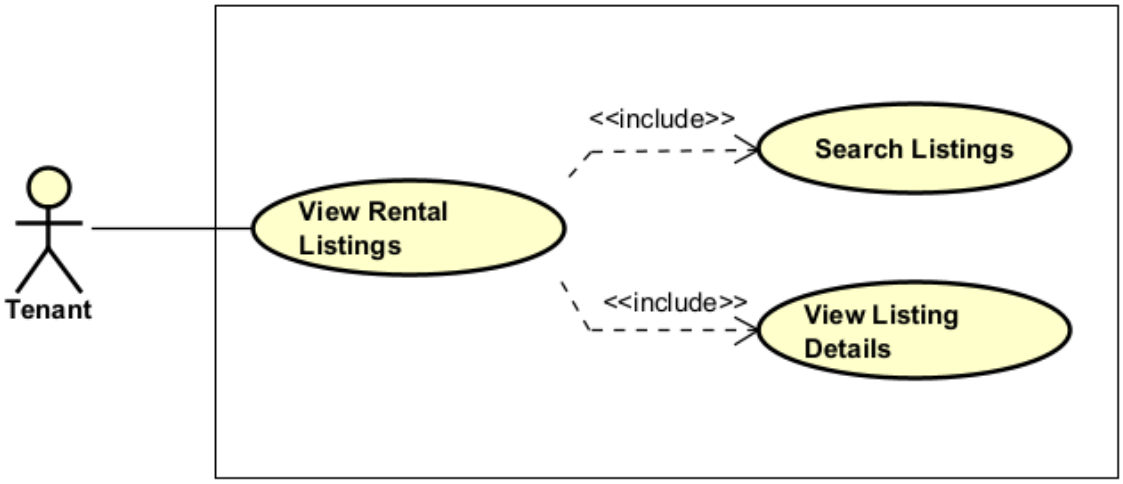


Figure 4. View Rental Listings Decompositon Use Case

3.2.3.5 Bookmark Listings

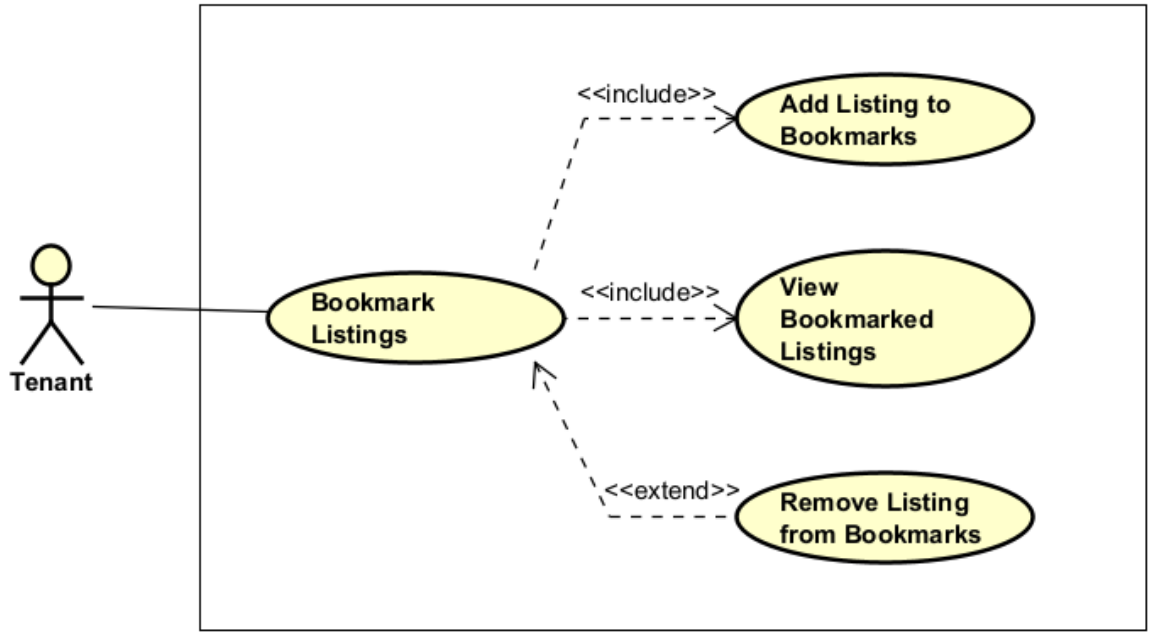


Figure 5. Bookmark Listings Decompositon Use Case

3.2.3.6 Apply for Rental

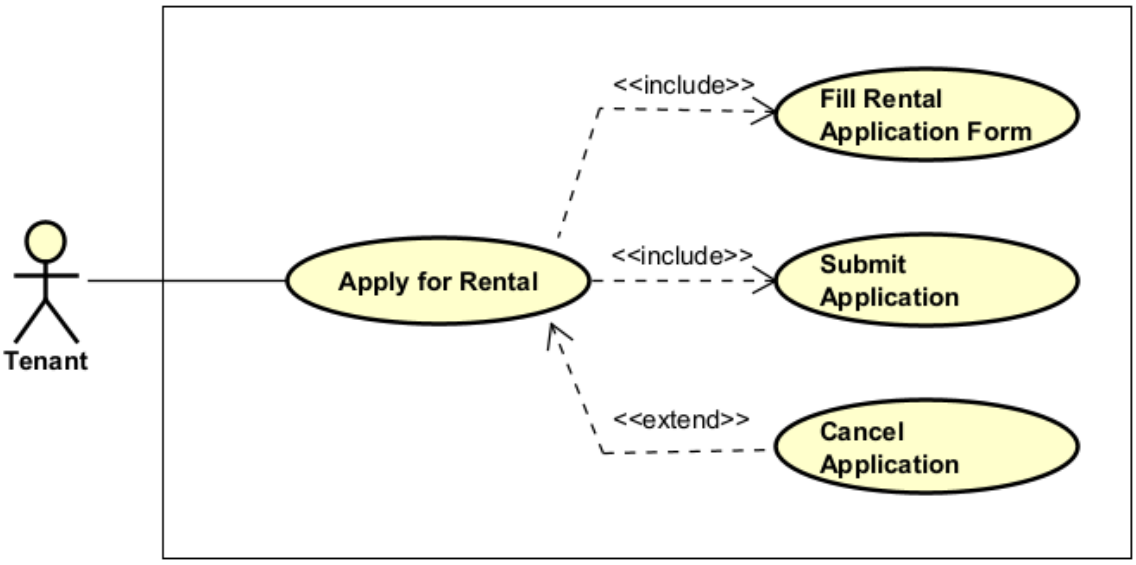


Figure 6. Apply for Rental Decompositon Use Case

3.2.3.7 Request Maintenance Services

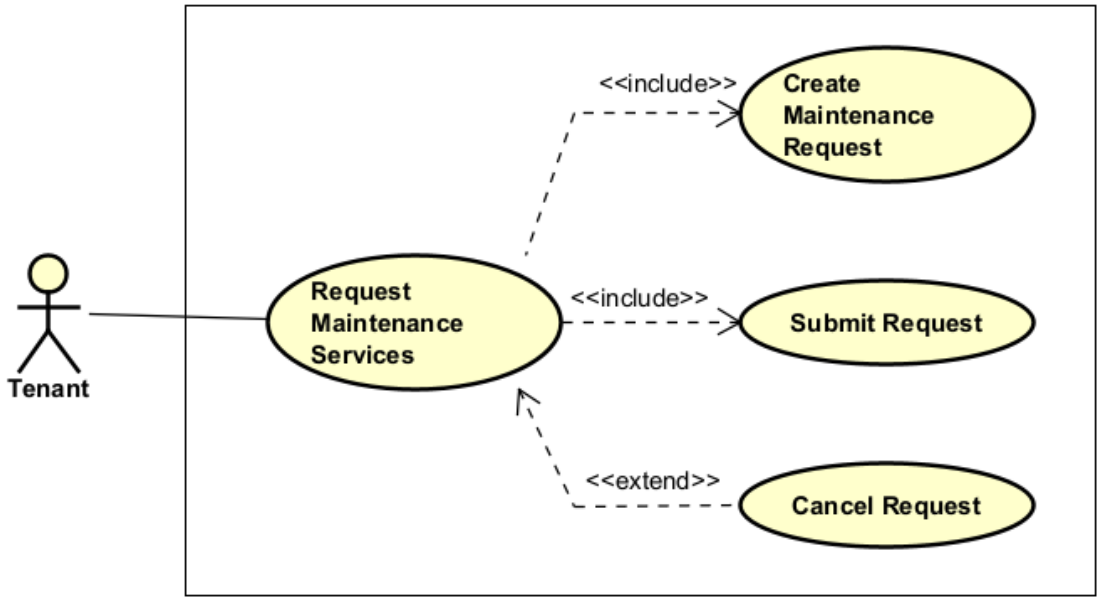


Figure 7. Request Maintenance Services Decompositon Use Case

3.2.3.8 Post Property Listings

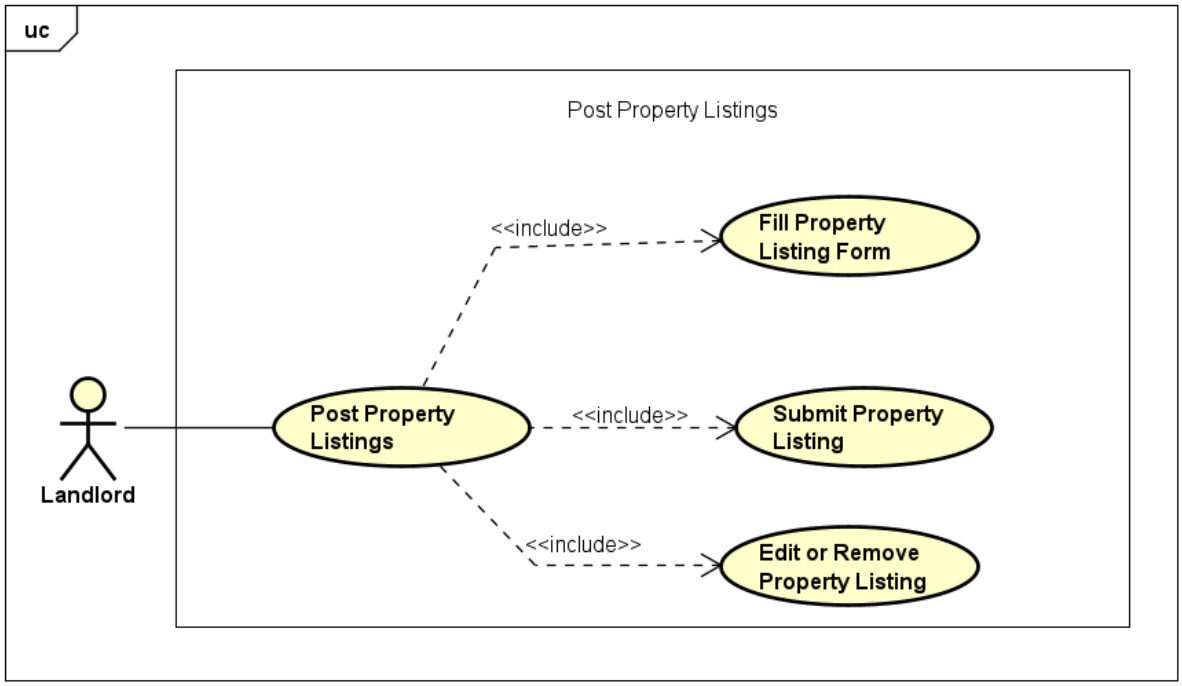


Figure 8. Post Property Listings Decompositon Use Case

3.2.3.9 Review Applications

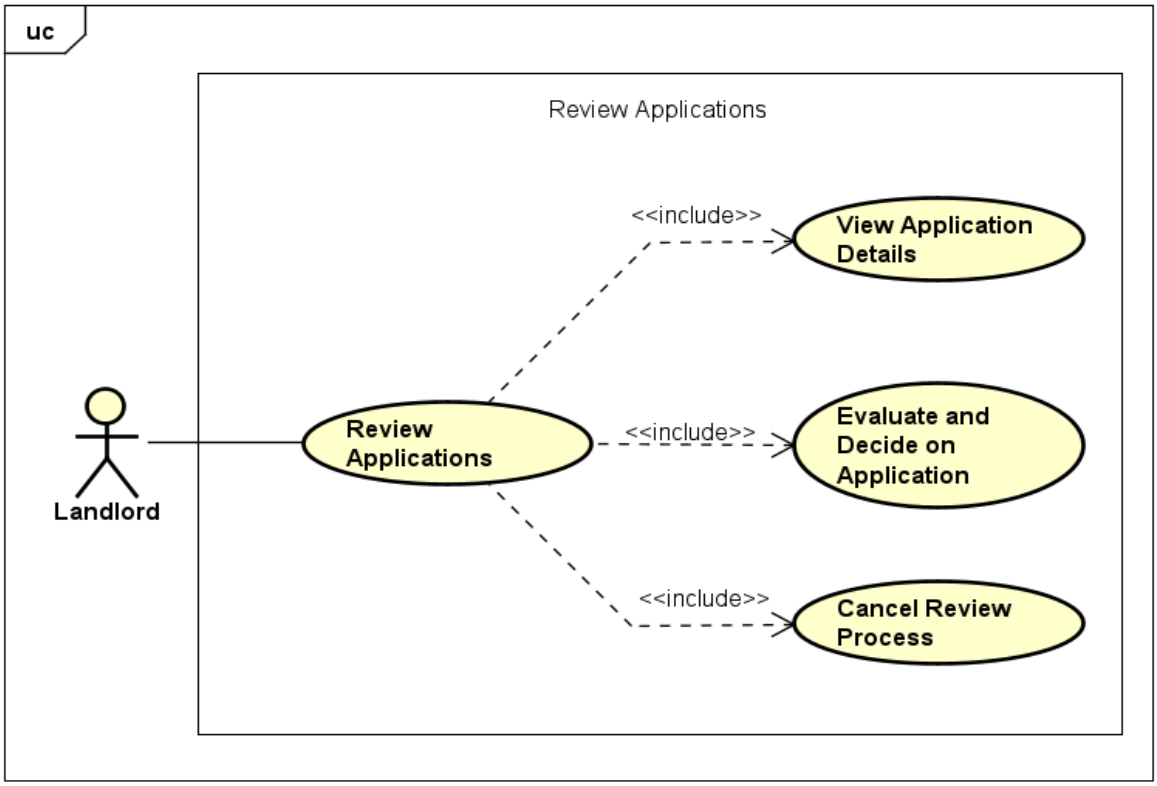


Figure 9. Review Applications Decompositon Use Case

3.2.3.10 Generate Simple Reports

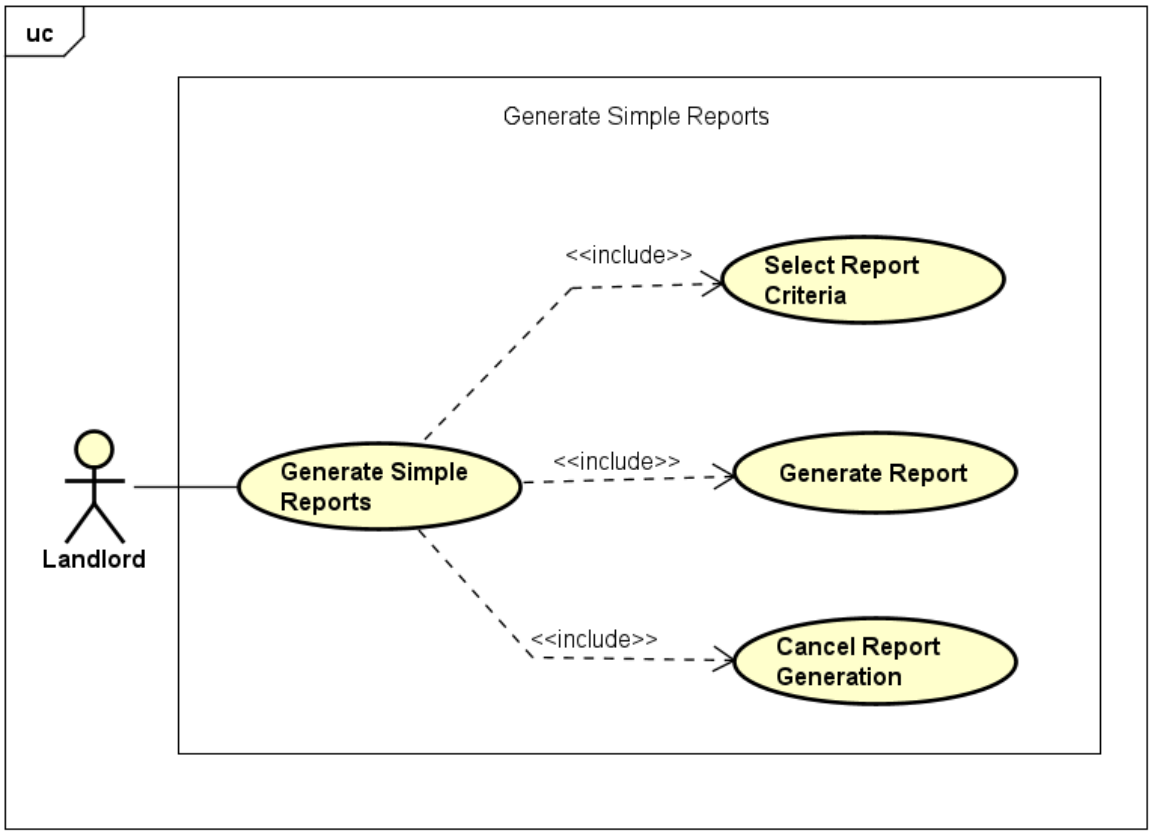


Figure 10. Generate Simple Reports Decompositon Use Case

3.2.3.11 Manage Users and Listings

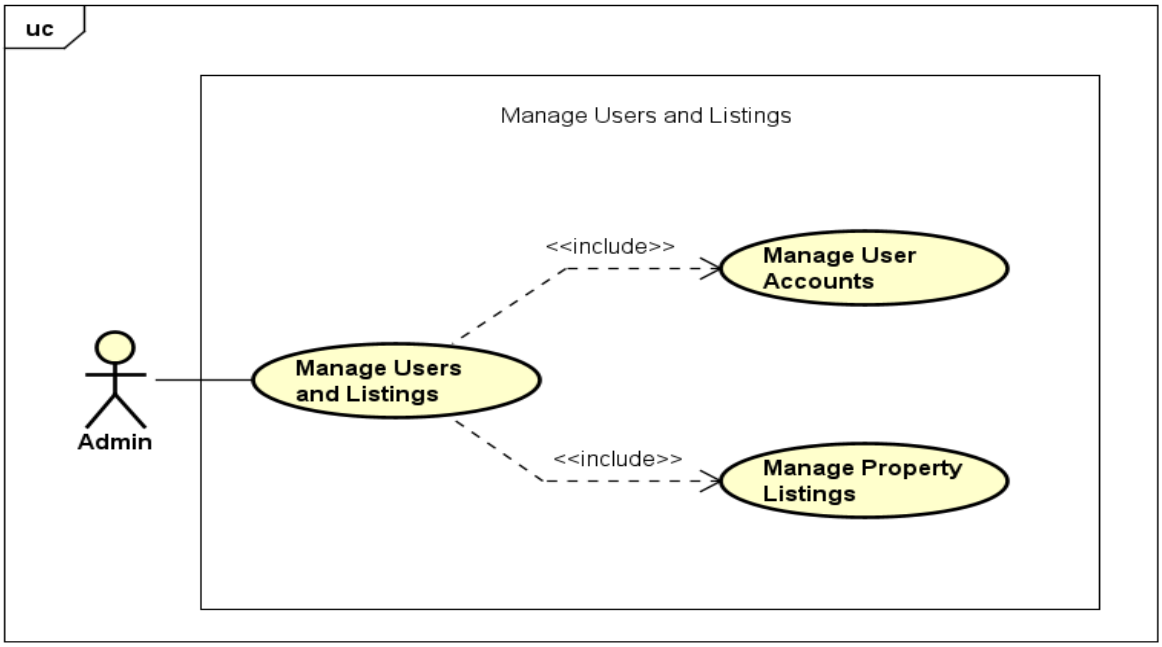


Figure 11. Manage Users and Listings Decompositon Use Case

3.2.4 Activity Diagram

3.2.4.1 Register and Login

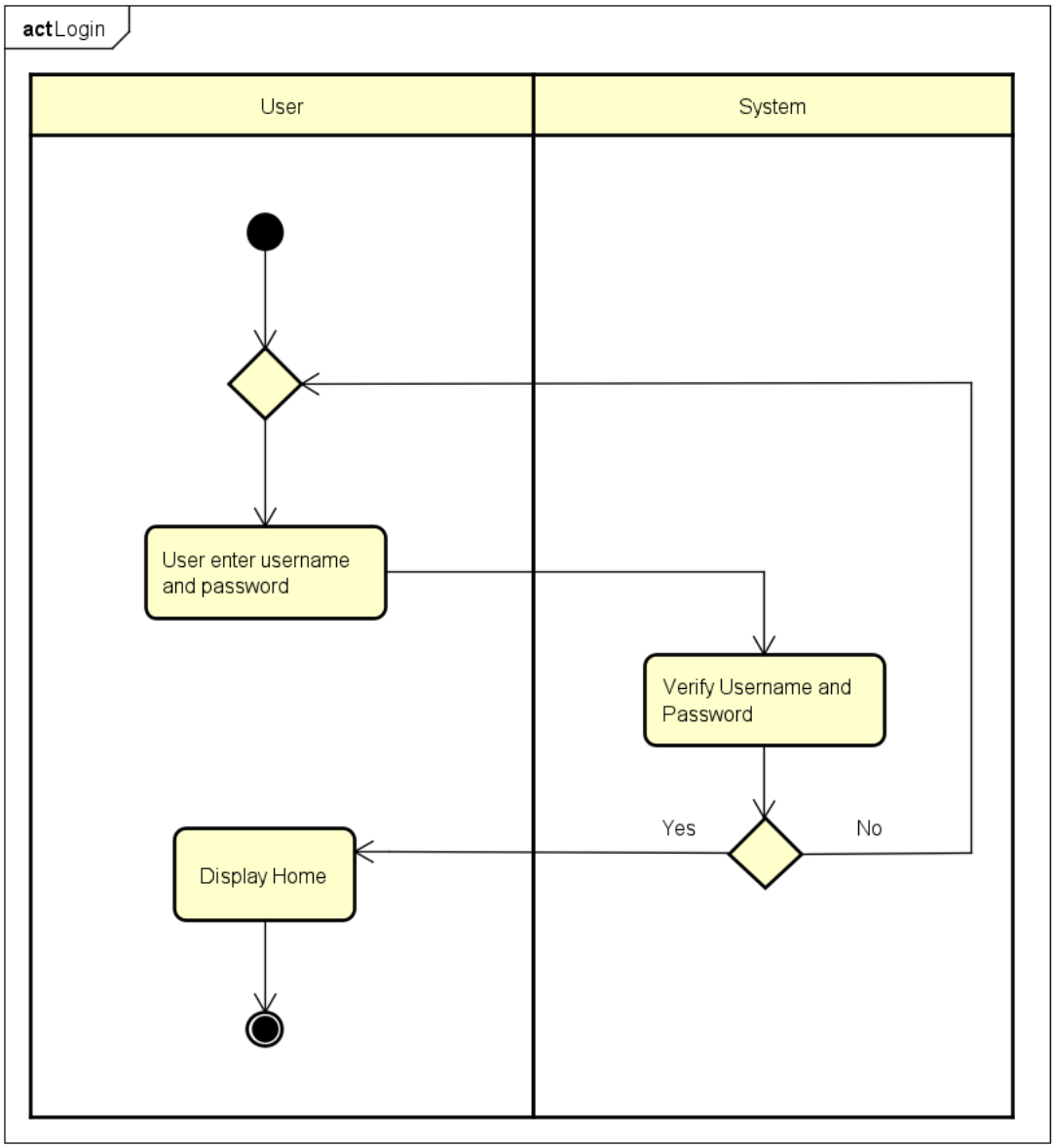


Figure 12. Login Activity Diagram

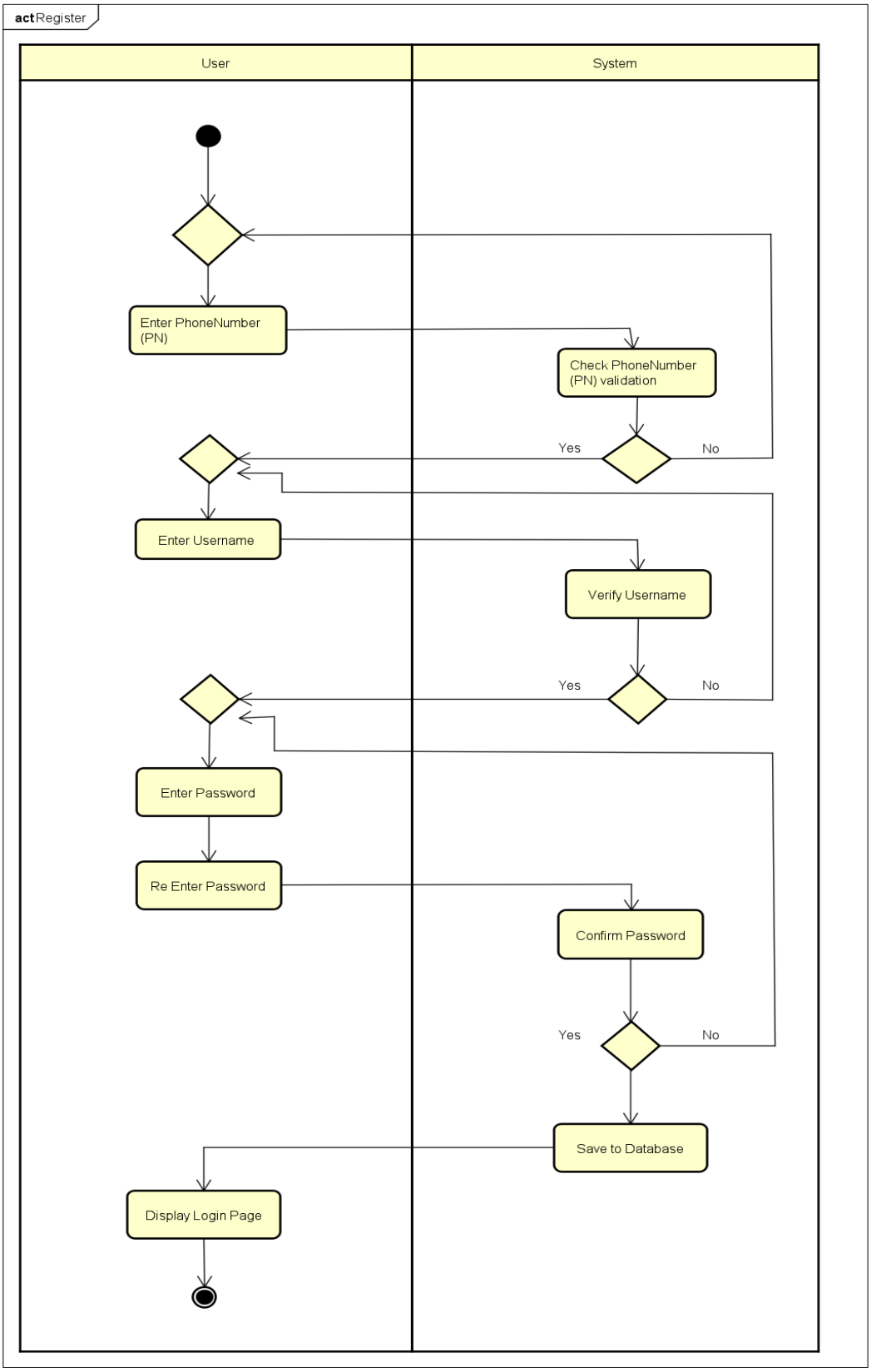


Figure 13. Register Activity Diagram

3.2.4.2 Make Rent Payments

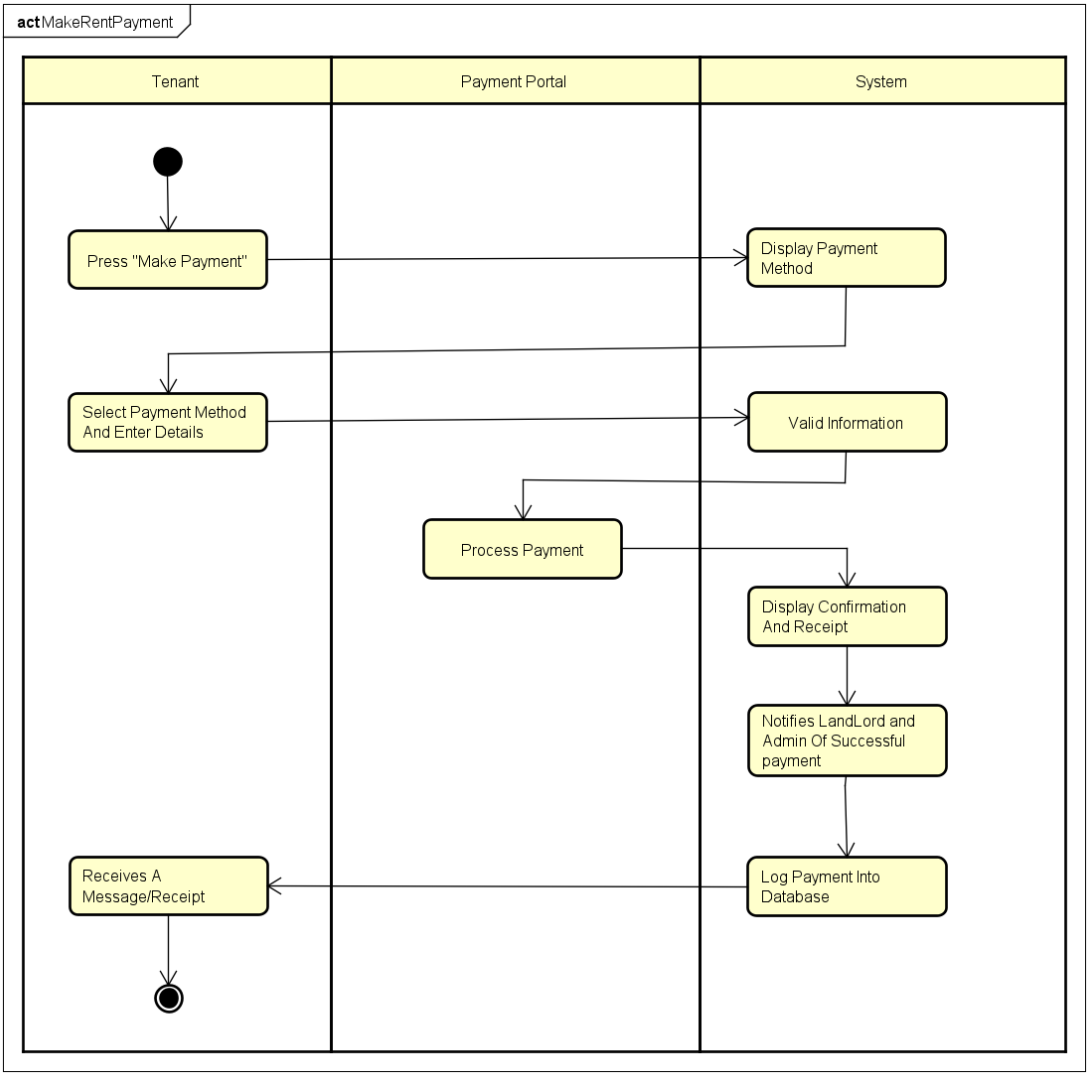


Figure 14. Make Rent Payments Activity Diagram

3.2.4.3 Use Message System

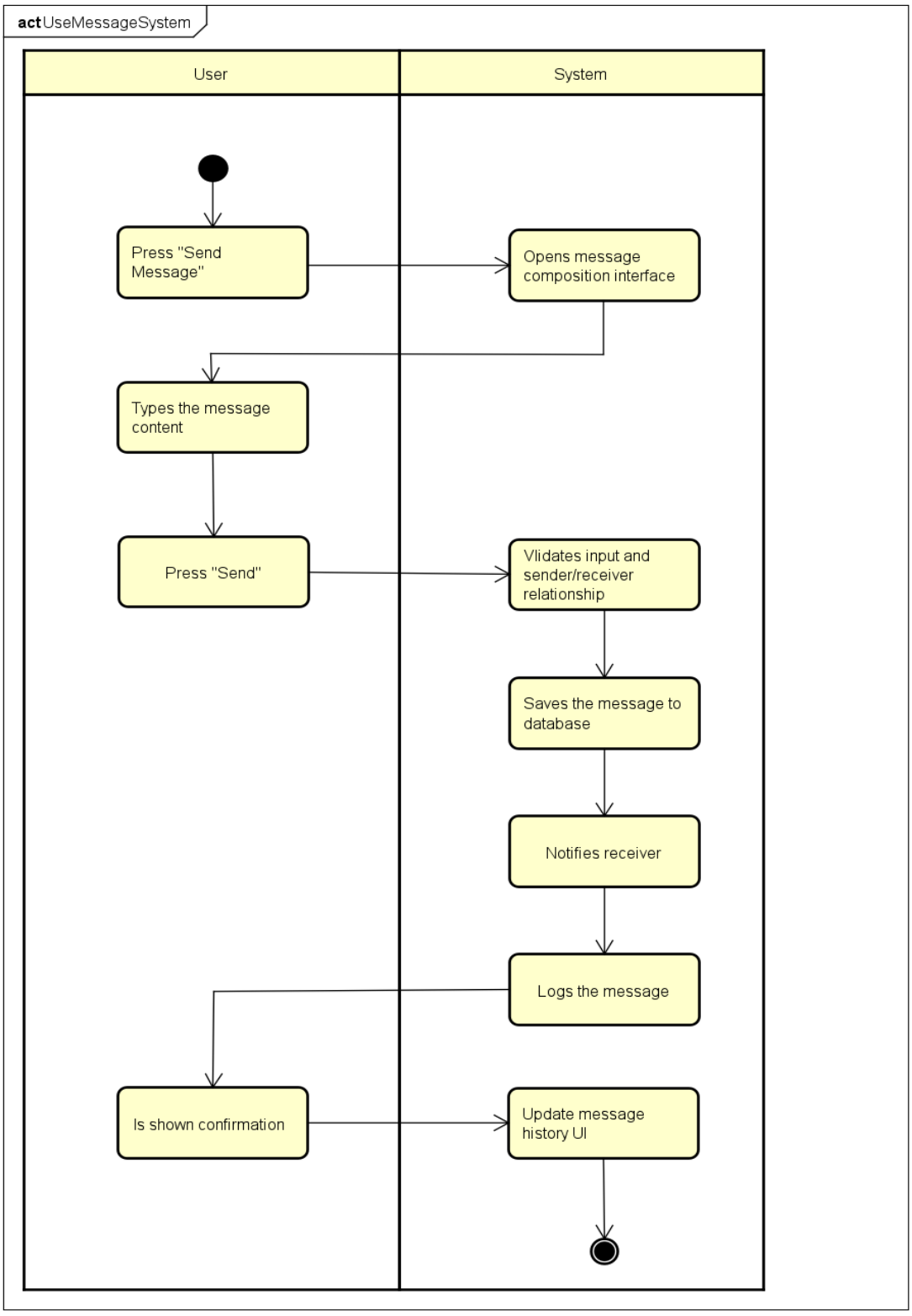


Figure 15. Use Message System Activity Diagram

3.2.4.4 View Rental Listings

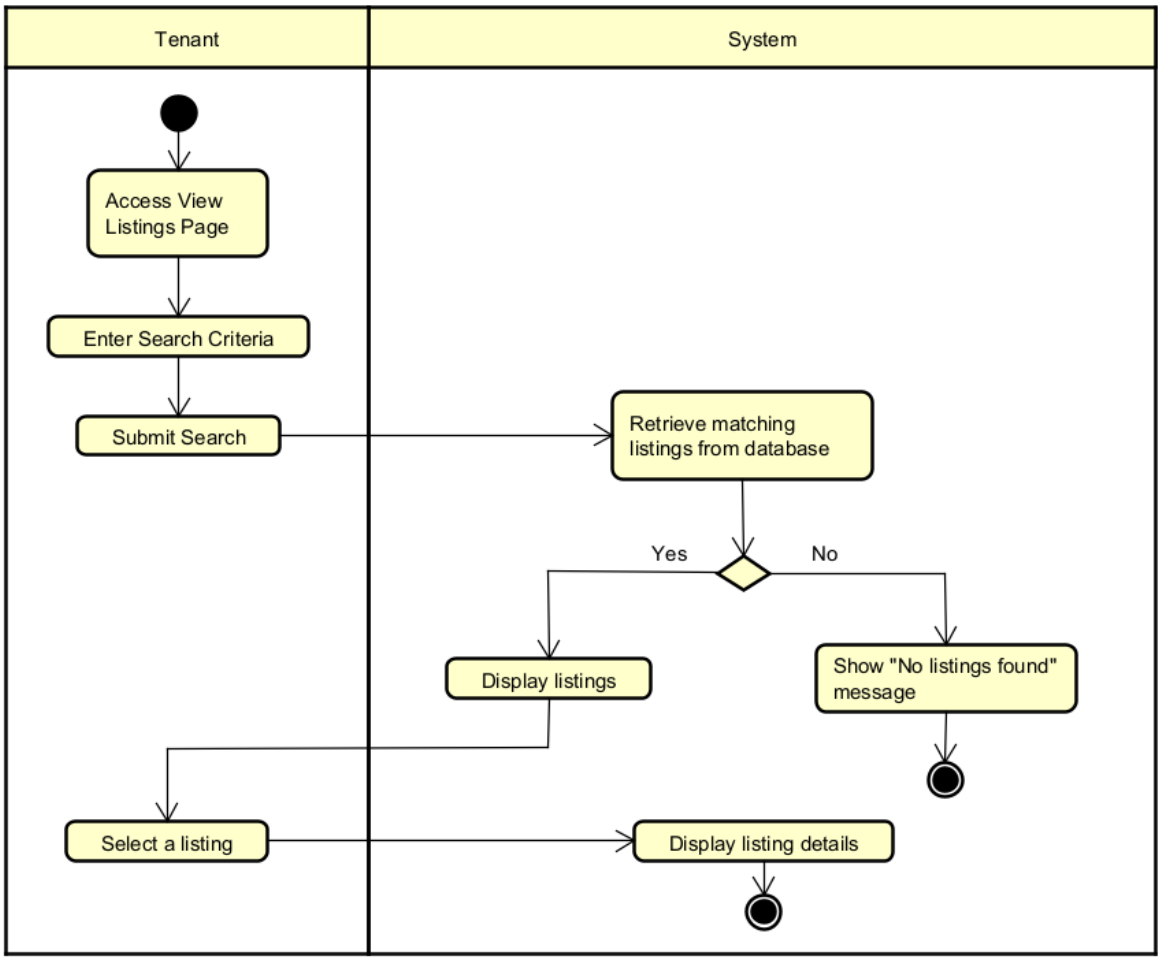


Figure 16. View Rental Listings Activity Diagram

3.2.4.5 Bookmark Listings

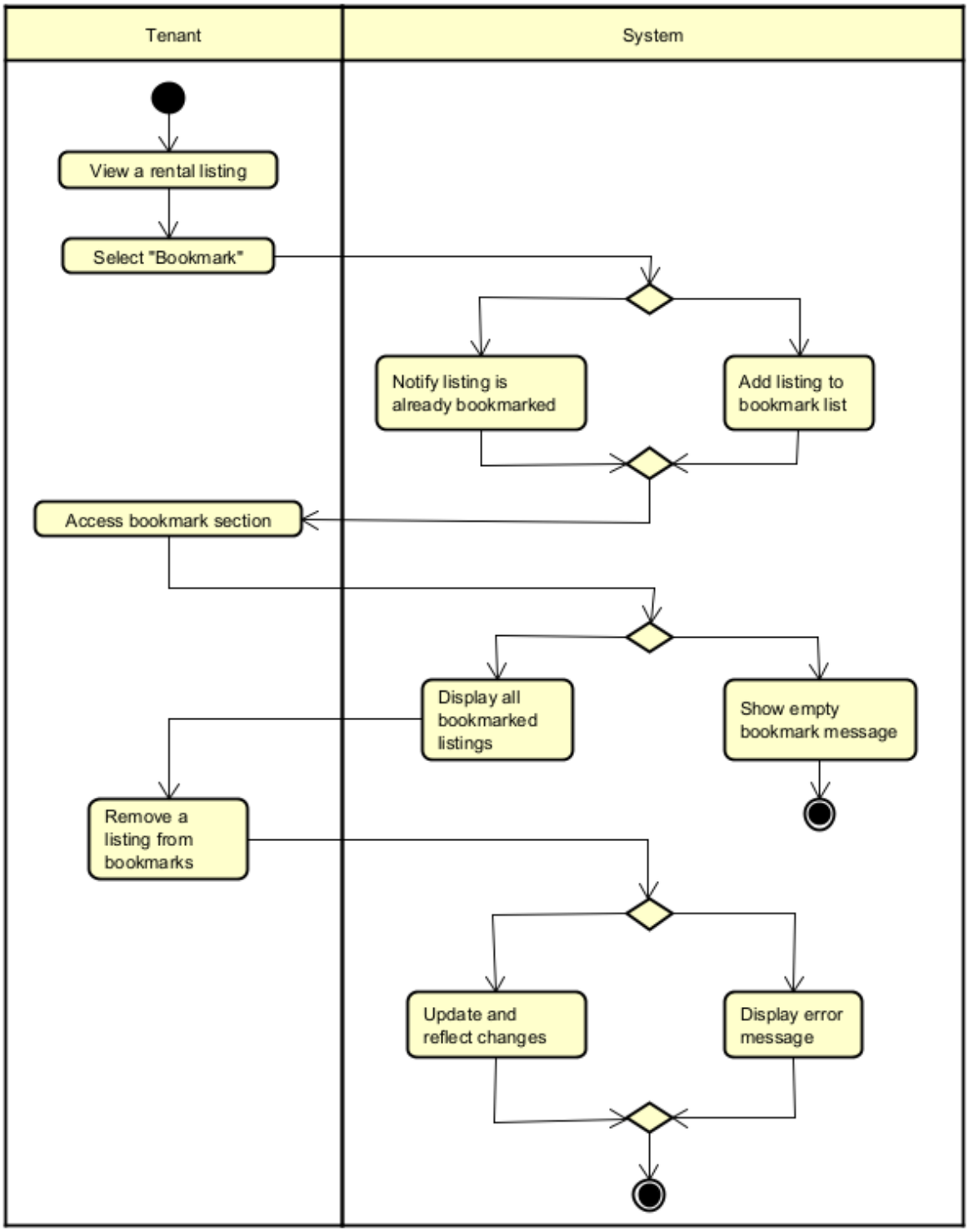


Figure 17. Bookmark Listings Activity Diagram

3.2.4.6 Apply for Rental

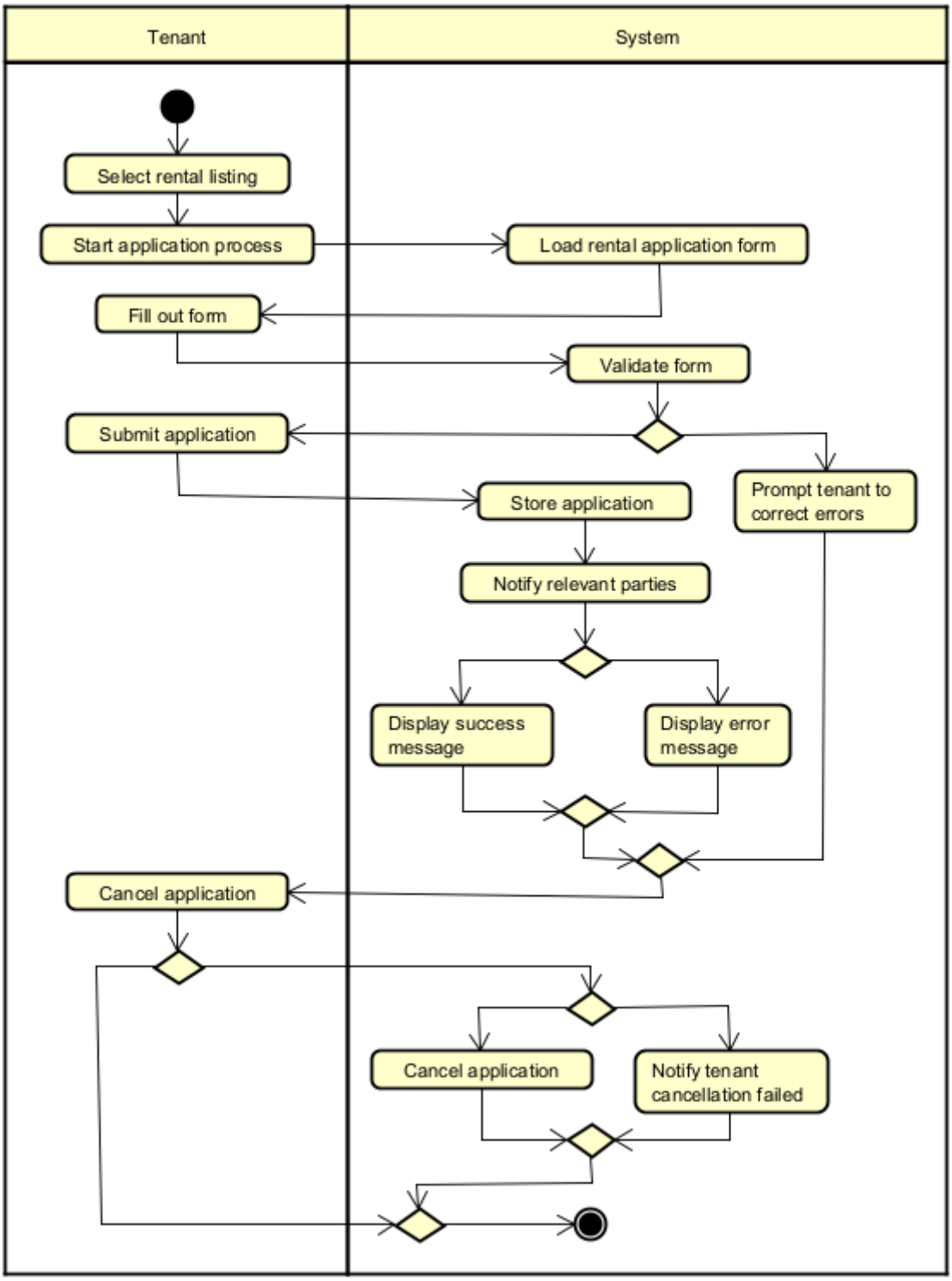


Figure 18. Apply for Rental Activity Diagram

3.2.4.7 Request Maintenance Services

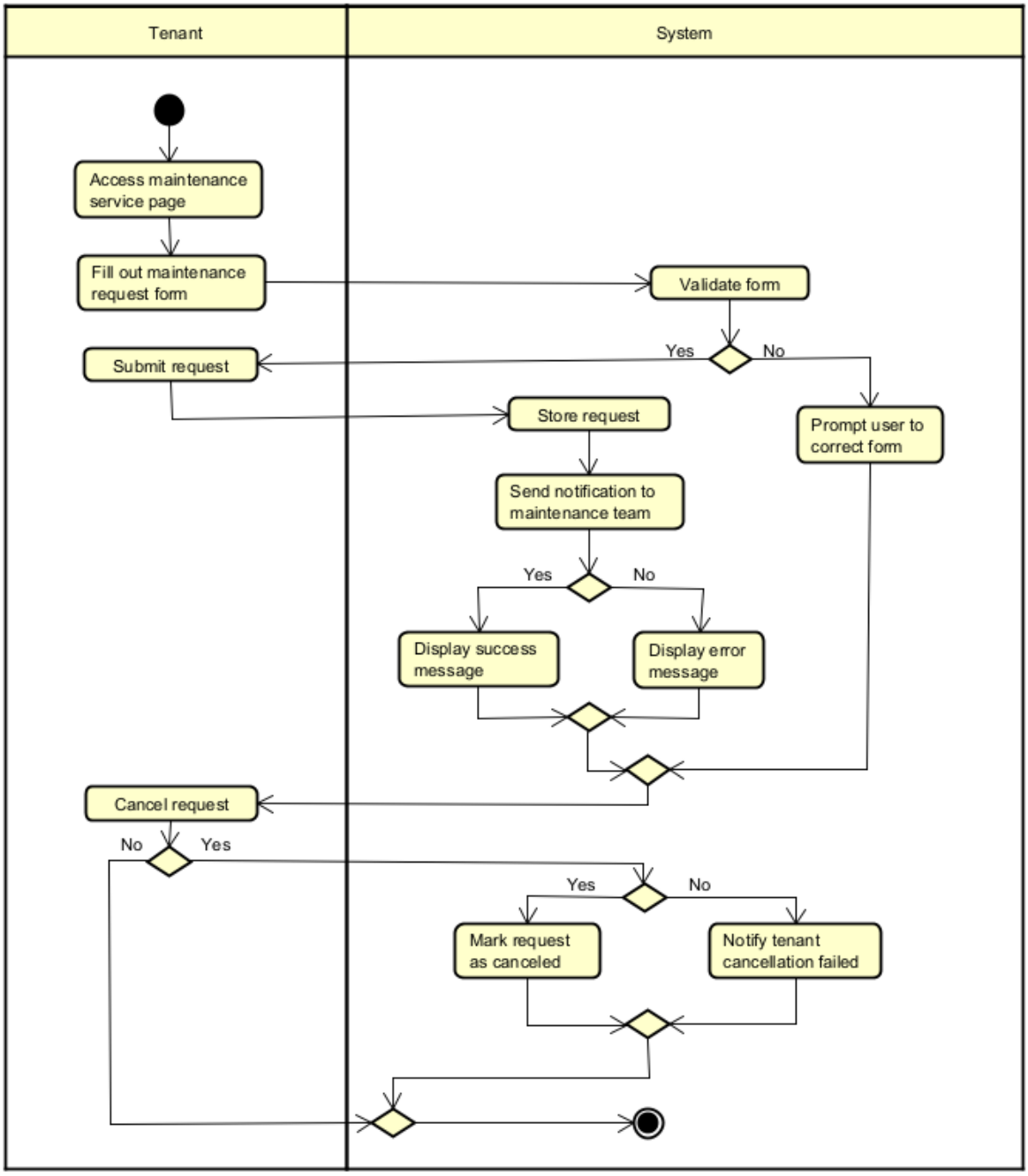


Figure 19. Request Maintenance Services Activity Diagram

3.2.4.8 Post Property Listings

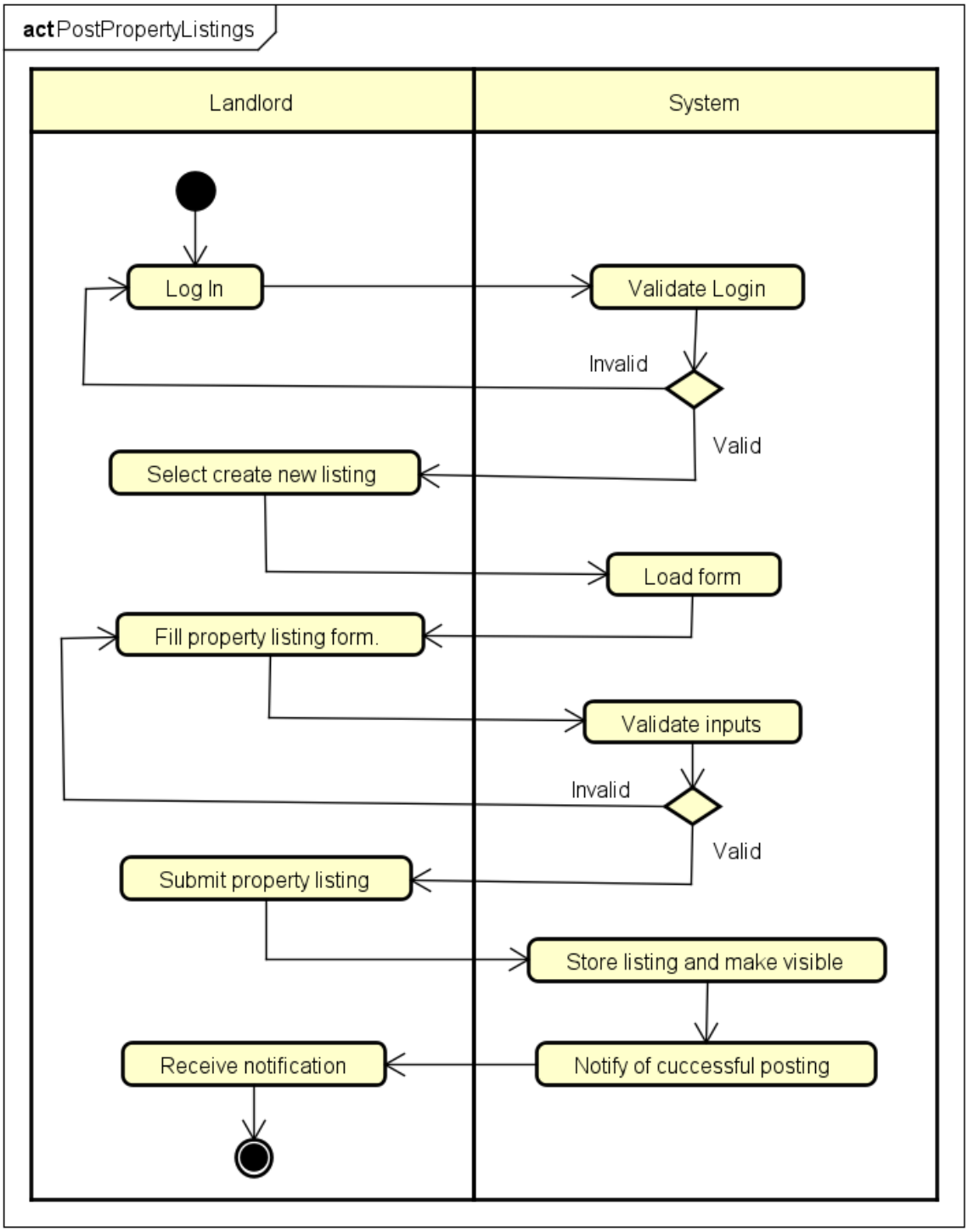


Figure 20. Post Property Listings Activity Diagram

3.2.4.9 Review Applications

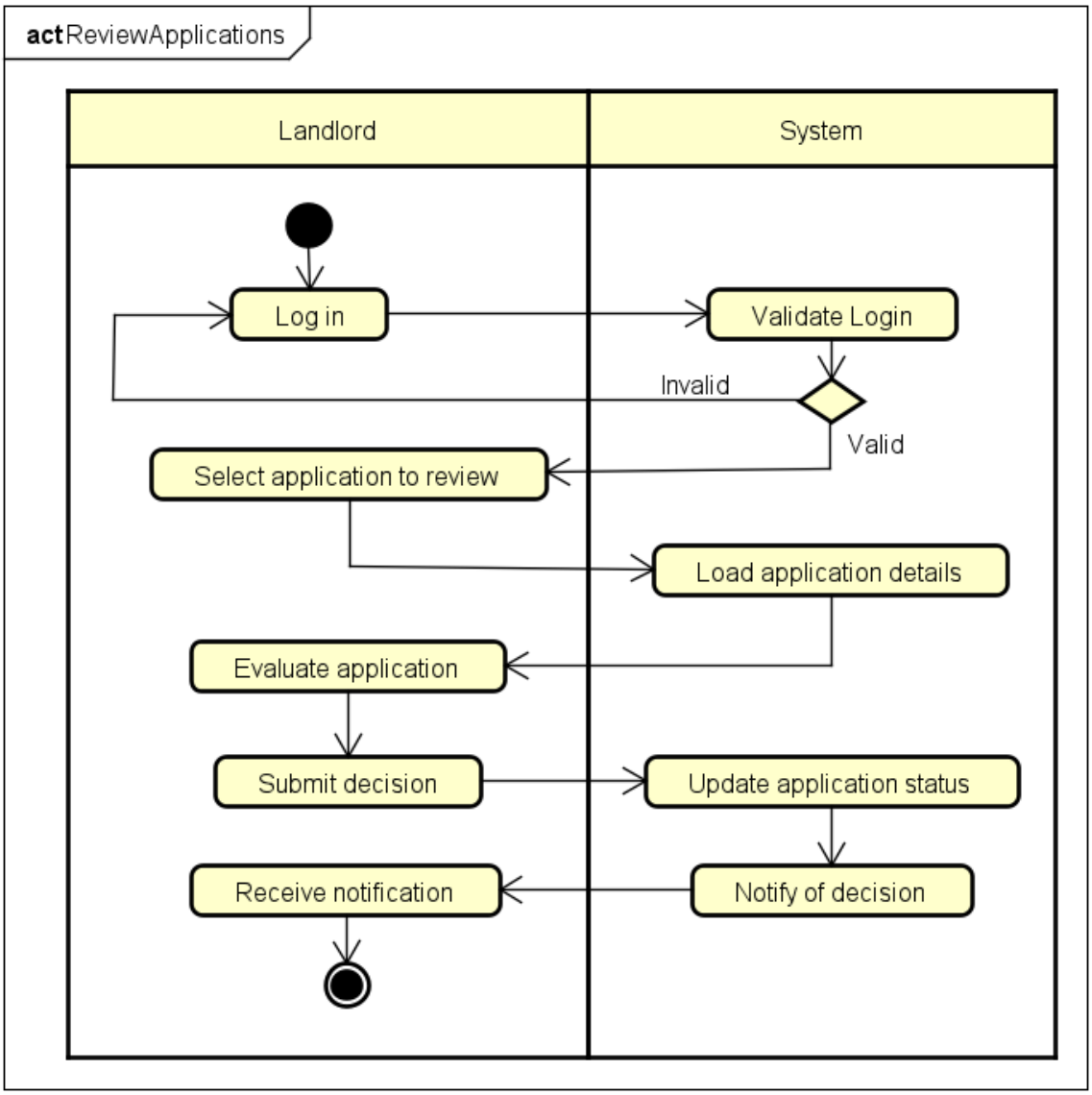


Figure 21. Review Applications Activity Diagram

3.2.4.10 Generate Simple Reports

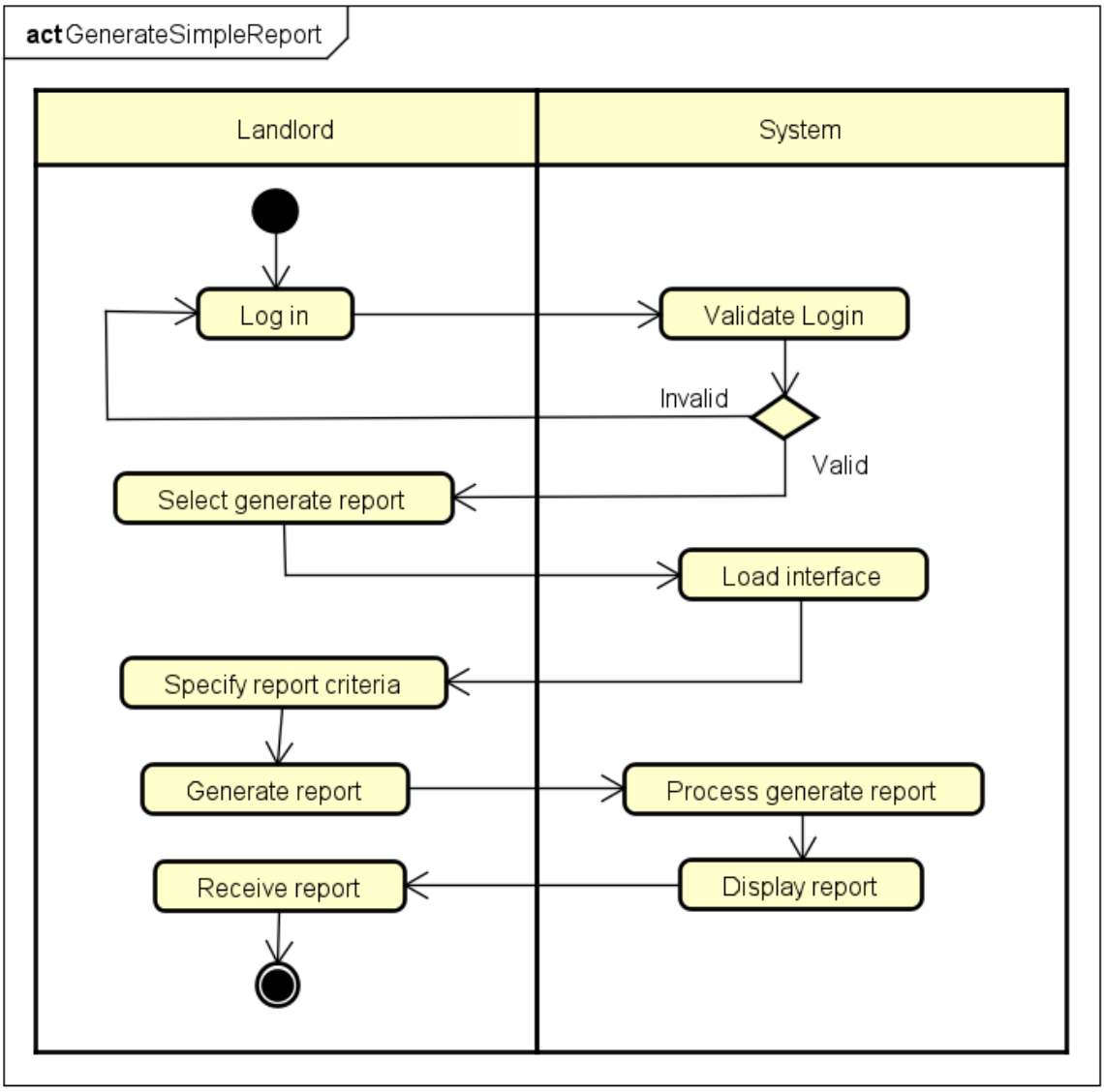


Figure 22. Generate Simple Reports Activity Diagram

3.2.4.11 Manage Users and Listings

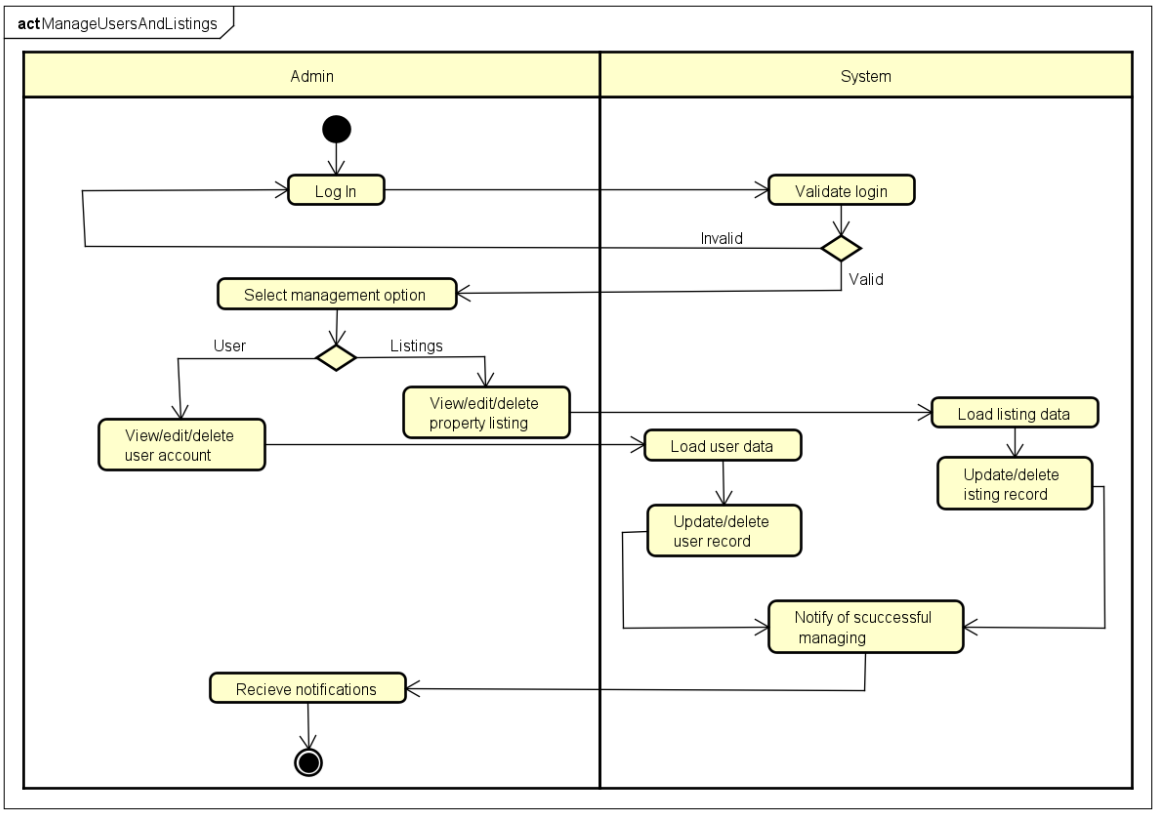


Figure 23. Manage Users and Listings Activity Diagram

3.3 Functional requirements

FR1 : Tenant Portal

- The system shall provide tenants with a portal to view lease details, make payments, and submit maintenance requests.

FR2 : User Authentication

- The system shall authenticate users (landlords, tenants, administrators) with phone number and password.

FR3 : User Account Management

- The system shall allow administrators to create, edit, or delete user accounts.

FR4 : Data Backup

- The system shall enable administrators to back up the database manually.

FR5 : Reporting

- The system shall generate reports on revenue, occupancy rates, and outstanding payments for a specified period.

FR6 : Maintenance Scheduling

- The system shall allow landlords to log and schedule maintenance tasks based on tenant requests.

FR7 : Payment Processing

- The system shall enable landlords to track tenant payments and send reminders for overdue payments.

3.4 Non-Functional requirements

NFR1 : Performance

- The system shall load pages (e.g., tenant dashboard) within 3 seconds under normal conditions.

NFR2 : Scalability

- The system shall support up to 100 concurrent users without significant performance degradation.

NFR3 : Security

- The system shall encrypt sensitive data (e.g., payment details) and use HTTPS for all communications.

NFR4 : Usability

- The system shall have an intuitive interface, requiring no more than 10 minutes for new users to learn basic functions.

NFR5 : Availability

- The system shall be available 99% of the time during the project evaluation period, excluding scheduled maintenance.

NFR6 : Compatibility

- The system shall be compatible with modern browsers (Chrome, Firefox) and mobile devices (iOS, Android).

NFR7 : Maintainability

- The system shall allow developers to apply updates or fix bugs within 2 hours of identification.

NFR8 : Portability

- The system shall run on any platform supporting PHP and SQLite, ensuring easy deployment for academic evaluation.

CHAPTER 4. ARCHITECTURE

4.1 Architectural style(s) used

The Room Rental Management Software (RRMS) leverages two architectural styles to meet its design goals:

- Client-Server :
  - The system is structured with clients (web browsers) requesting services from a centralized server.
  - This style enables remote access, allowing landlords and tenants to use the system from any device with an internet connection, a key requirement for distributed users.
- Layered (Three-Tier) :
  - Presentation Tier: A HTML5/Tailwind CSS front-end that provides a responsive, utility-first user interface.
  - Application Tier: PHP (MVC-style) handling session/authentication and business logic.
  - Data Tier: A MySQL database for storing user accounts and file metadata with proper foreign-key constraints.
- This separation enhances maintainability by isolating user interface, logic, and data concerns, while also supporting scalability for future enhancements.

Together, the Client-Server style governs external interactions, while the Three-Tier style structures the internal architecture, ensuring a robust and flexible system.

4.2 Architectural Model

The 4+1 View Model is adopted to provide a comprehensive architectural description of RRMS, addressing multiple perspectives:

- **Logical View** :

    - Outlines the functional components, such as classes or modules for managing tenants, payments, and rooms (e.g., Tenant, Payment,…).
- **Process View**:
    - Describes runtime behavior and concurrency, utilizing Node.js's asynchronous, event-driven model to manage simultaneous user requests (e.g., multiple tenants viewing payment details).
- **Physical View**:
    - Details the deployment setup, with the application and database hosted on Heroku, a cloud platform that ensures accessibility and simplifies deployment.
- **Development View**:
    - Shows the codebase structure, following the Model-View-Controller (MVC) pattern :
        - **Model**: PHP classes under model/ that define data schemas and encapsulate all database logic.
        - **View**: Plain HTML5/Tailwind templates under views/ responsible for rendering the UI and binding dynamic data passed from controllers.
        - **Controller**: PHP controller classes under controller/ that handle incoming HTTP requests (routes), invoke model methods, and load the appropriate view.
- **Scenarios (Use Cases)**:
    - Illustrates key interactions, such as "Make Rent Payments" integrating all views to demonstrate system functionality.

This model ensures a well-rounded representation of the architecture, covering functionality, performance, deployment, and development aspects.

4.3 Technology, Software, and Hardware used

Technology :

- **Front-end** : HTML5, Tailwind CSS, JavaScript
- **Back-end** : PHP
- **Database** : MySQL

Software :

- **Development Environment**: Visual Studio Code, a versatile IDE for coding and debugging.
- **Version Control**: Git, hosted on GitLab, for collaborative development and version tracking.
- **Testing Tools**: PHPUnit (or Pest) for unit and integration tests, Cypress (or Playwright) for browser-based end-to-end tests, Postman for API testing.

Hardware :

- **Development**: Personal laptops with minimum specs of 8GB RAM, dual-core processor, and 256GB storage.
- **Deployment :** XAMPP, a free and open-source cross-platform web server solution stack. It provides an integrated environment with Apache, MySQL, and PHP, allowing for easy development, testing, and demonstration without the need for internet-based hosting.

4.4 Rationale for Architectural Style and Model

Client-Server Style :

- **Remote Access Simulation:** Although deployed locally via XAMPP, the architecture simulates a real-world client–server model where users (landlords and tenants) interact with the system through a browser, while the server (Apache + PHP) handles logic and database operations.

- **Centralized Data:** All user data and file information are stored in a centralized MySQL database, ensuring consistency, easier backup, and simplified access control.

Layered (Three-Tier) Style :

- **Separation of Concerns**: Divides responsibilities across presentation, application, and data tiers, making development, testing, and maintenance easier.
- **Maintainability & Scalability:** Each layer can be improved or extended independently—such as upgrading the UI or switching to a more powerful database engine.

4+1 View Model :

- **Holistic Perspective**: Captures multiple dimensions of the architecture, ensuring all stakeholders (e.g., developers, instructors) understand the system.
- **Educational Value**: Aligns with academic standards, providing a structured approach to document and analyze the design.

Technology Choices :

- **Practicality** : PHP and MySQL are foundational technologies in web development, widely supported and still used in many real-world applications. HTML5, Tailwind CSS, and JavaScript provide a modern, responsive, and interactive front-end experience.
- **Affordability** : All tools used—PHP, MySQL, XAMPP, JavaScript, Tailwind CSS—are completely free and open-source, making them ideal for student projects with zero licensing or hosting costs.
- **Simplicity** : XAMPP offers an all-in-one local server environment, allowing for quick setup and testing without requiring cloud deployment. This setup streamlines development and keeps the project scope manageable within limited academic timeframes.

CHAPTER 5. DESIGN

5.1 Database Design

The RRMS uses a relational database to manage data about tenants, rooms, payments, and maintenance requests. Below are the key tables and their fields:

- **Account**

```
CREATE TABLE `account` (
  `id` int(11) NOT NULL,
  `username` varchar(127) NOT NULL,
  `password` varchar(255) NOT NULL,
  `phone` varchar(20) DEFAULT NULL,
  `role` enum('tenant','landlord','admin') DEFAULT 'tenant'
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Figure 24. Account table

- **List**

```
CREATE TABLE `list` (
  `id` int(11) NOT NULL,
  `title` varchar(255) NOT NULL,
  `description` text DEFAULT NULL,
  `address` text DEFAULT NULL,
  `price` float DEFAULT NULL,
  `area` float DEFAULT NULL,
  `availableFrom` date DEFAULT curdate(),
  `image` text DEFAULT NULL,
  `status` enum('available','rented') DEFAULT 'available',
  `landlordID` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Figure 25. List table

- **Message**

```
CREATE TABLE `message` (
  `id` int(11) NOT NULL,
  `sender` int(11) DEFAULT NULL,
  `receiver` int(11) DEFAULT NULL,
  `content` text DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Figure 26. Message table

- **Payment**

```
CREATE TABLE `payment` (
  `id` int(11) NOT NULL,
  `amount` float NOT NULL,
  `date` date DEFAULT curdate(),
  `method` enum('PayPal','Visa') DEFAULT 'PayPal'
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Figure 27. Payment table

- **Report**

```
CREATE TABLE `report` (
  `id` int(11) NOT NULL,
  `name` varchar(255) NOT NULL,
  `portAt` date DEFAULT curdate(),
  `content` text DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Figure 28. Report table

5.2 Static Model – Class Diagrams

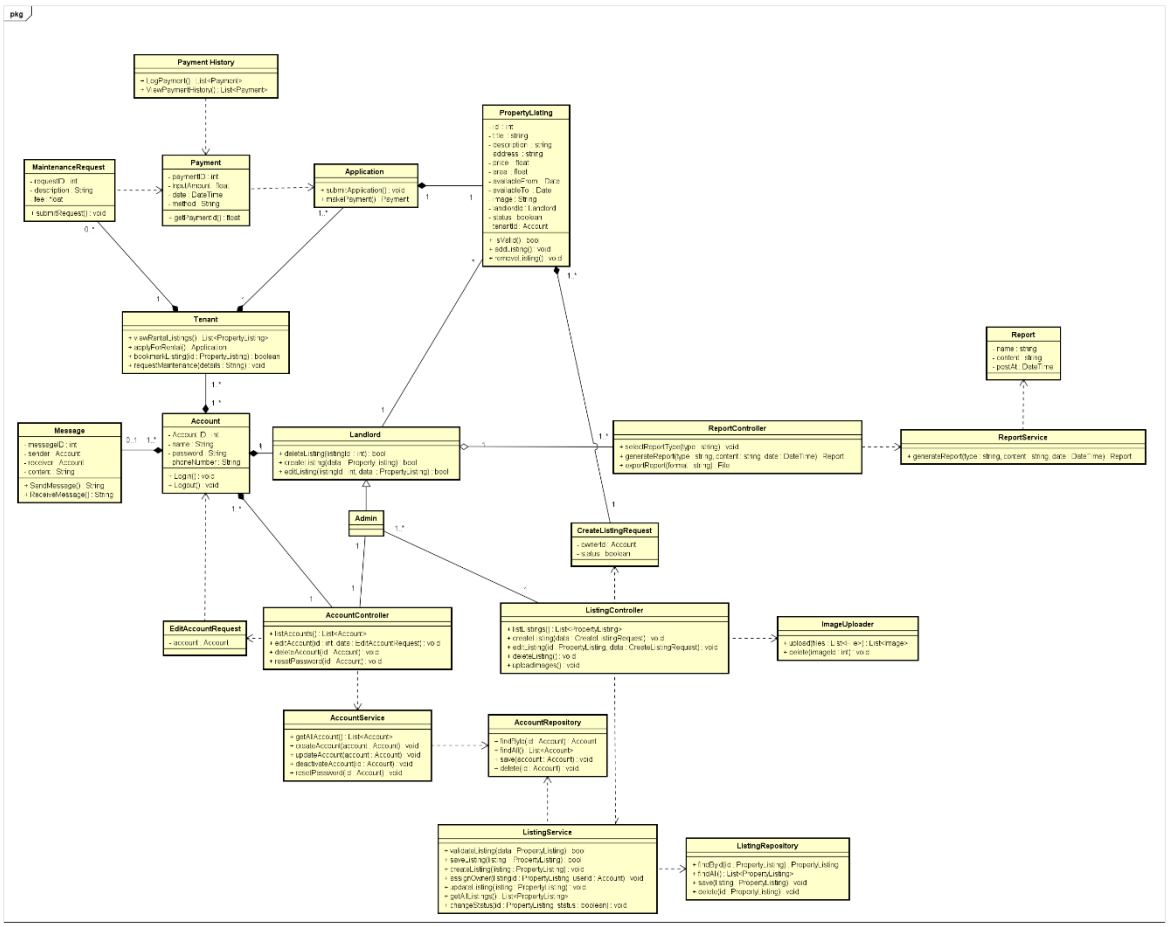


Figure 29. Class Diagram

These classes separate data management (models), business logic (controllers), and user interface (views), enhancing maintainability.

5.3 Dynamic Model – Sequence Diagrams

5.3.1 Register and Login

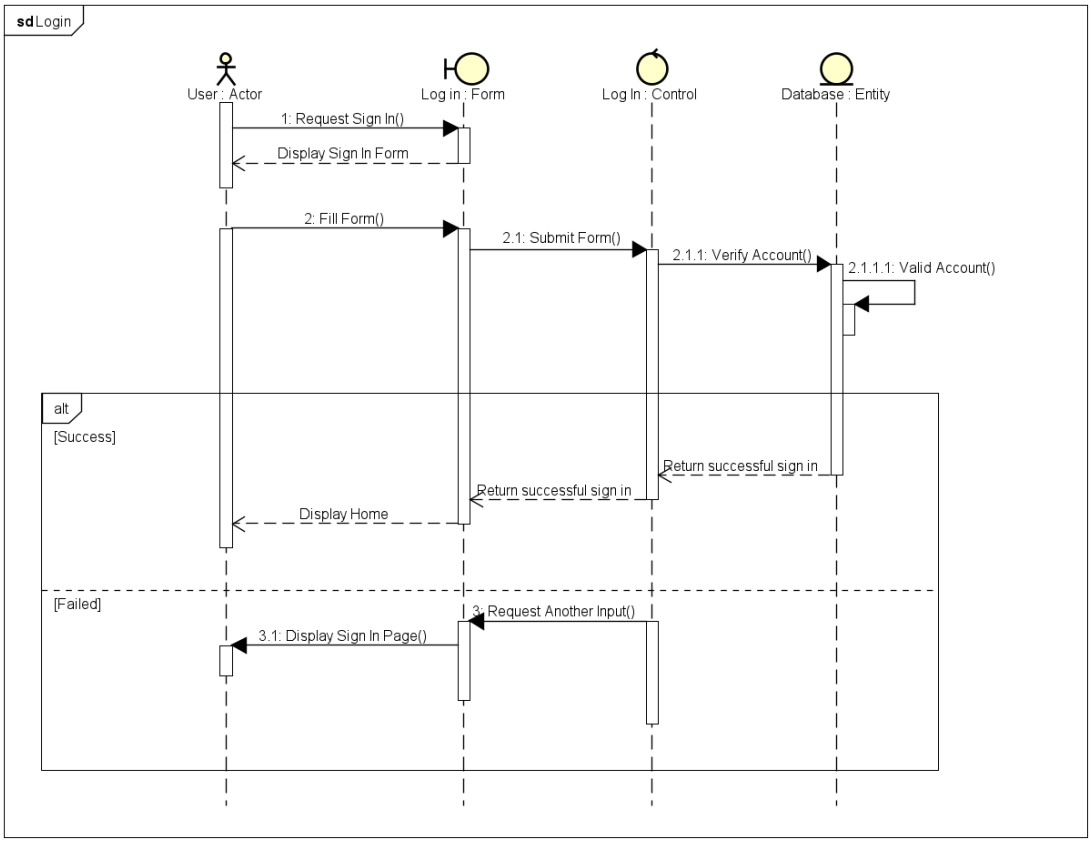


Figure 30. Log In Sequence Diagram

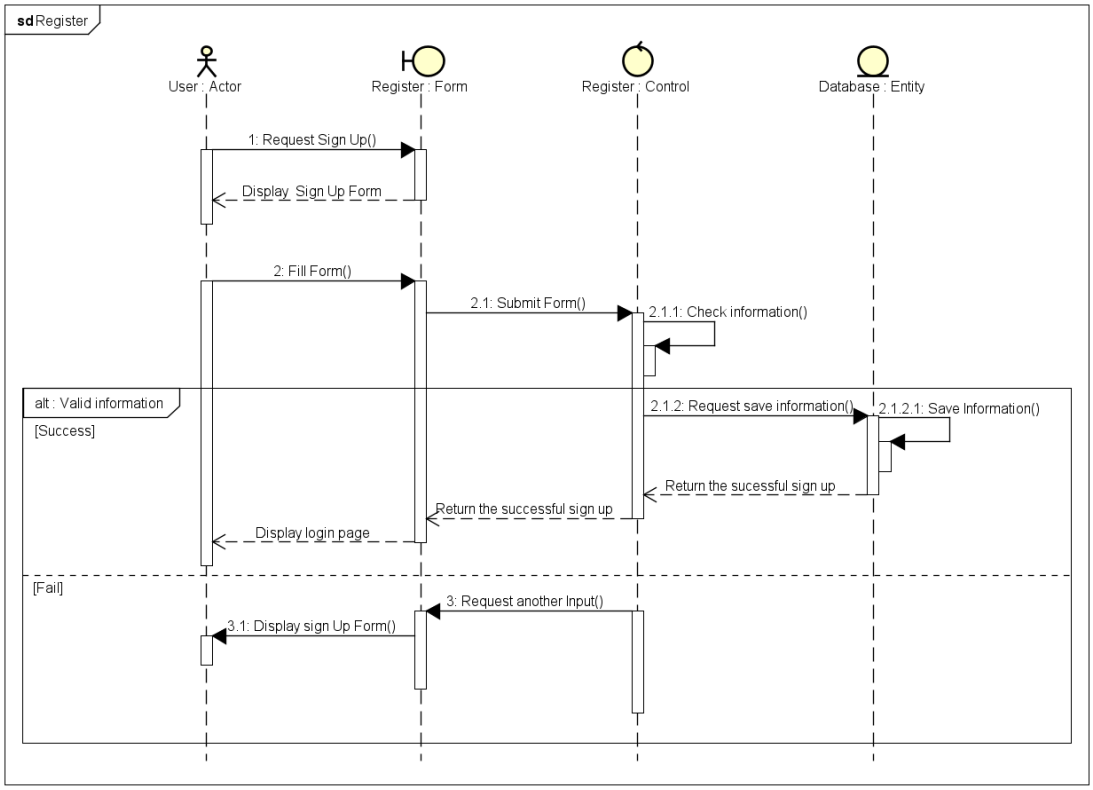


Figure 31. Register Sequence Diagram

5.3.2 Make Rent Payment

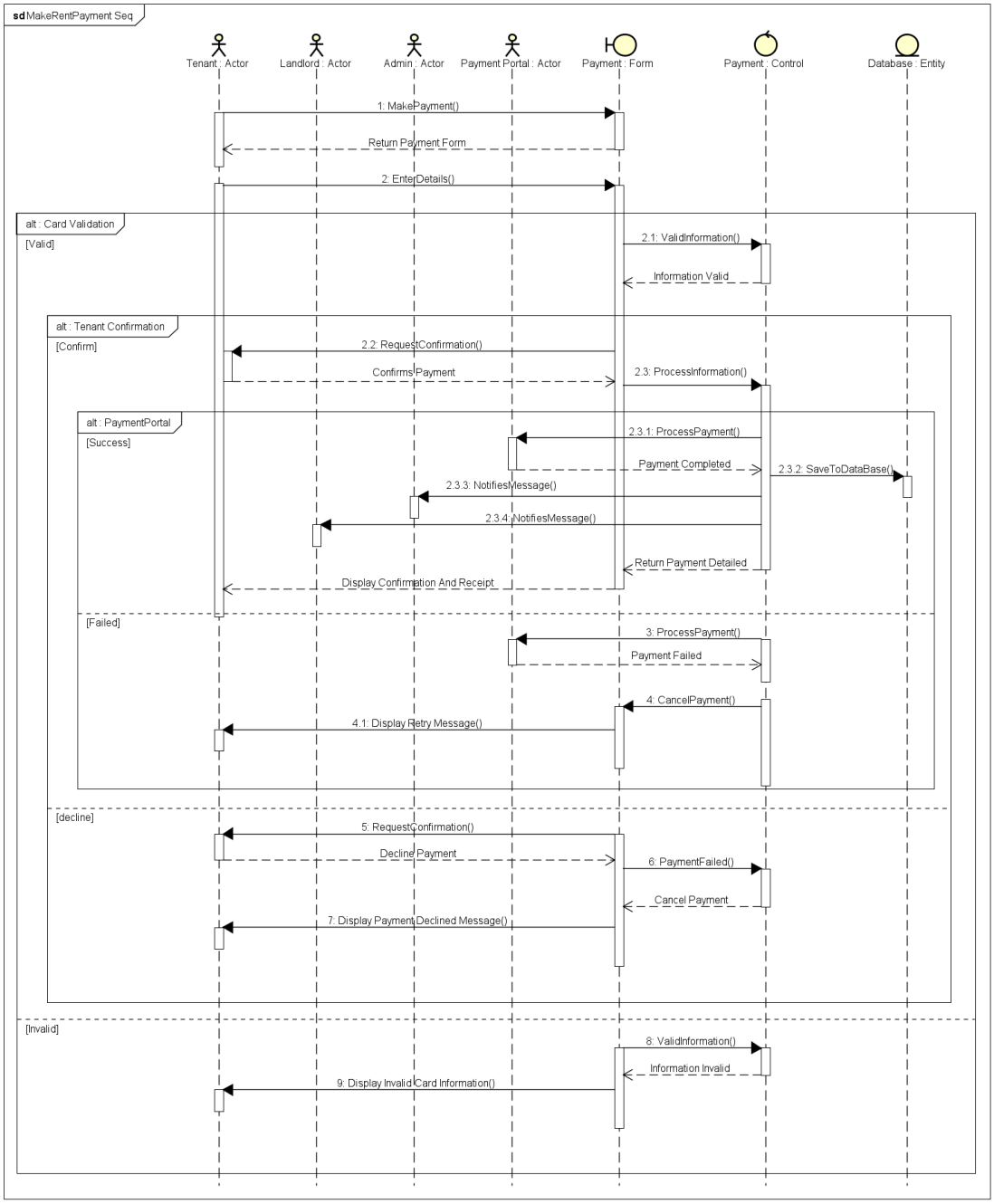


Figure 32. Make Rent Payments Sequence Diagram

5.3.3 Use Message System

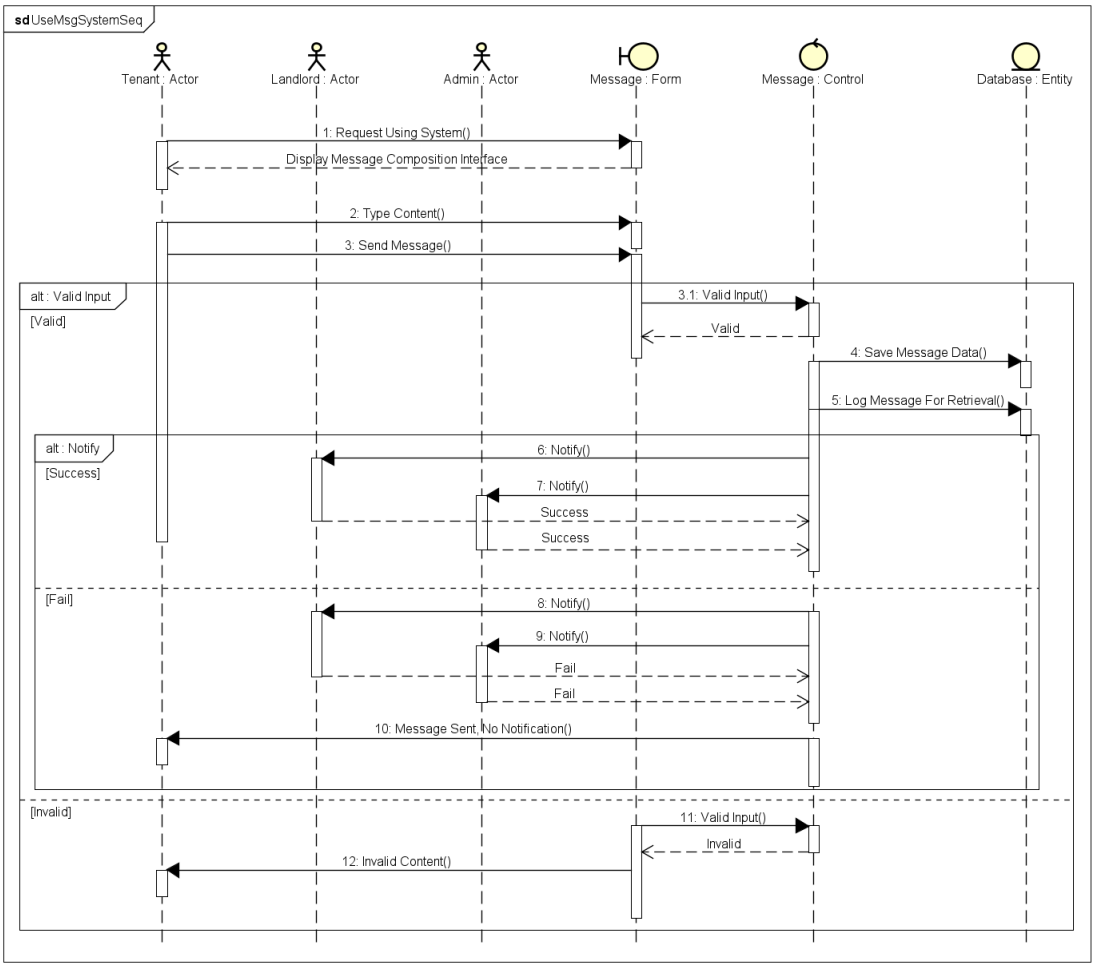


Figure 33. Use Message System Sequence Diagram

5.3.4 View Rental Listings

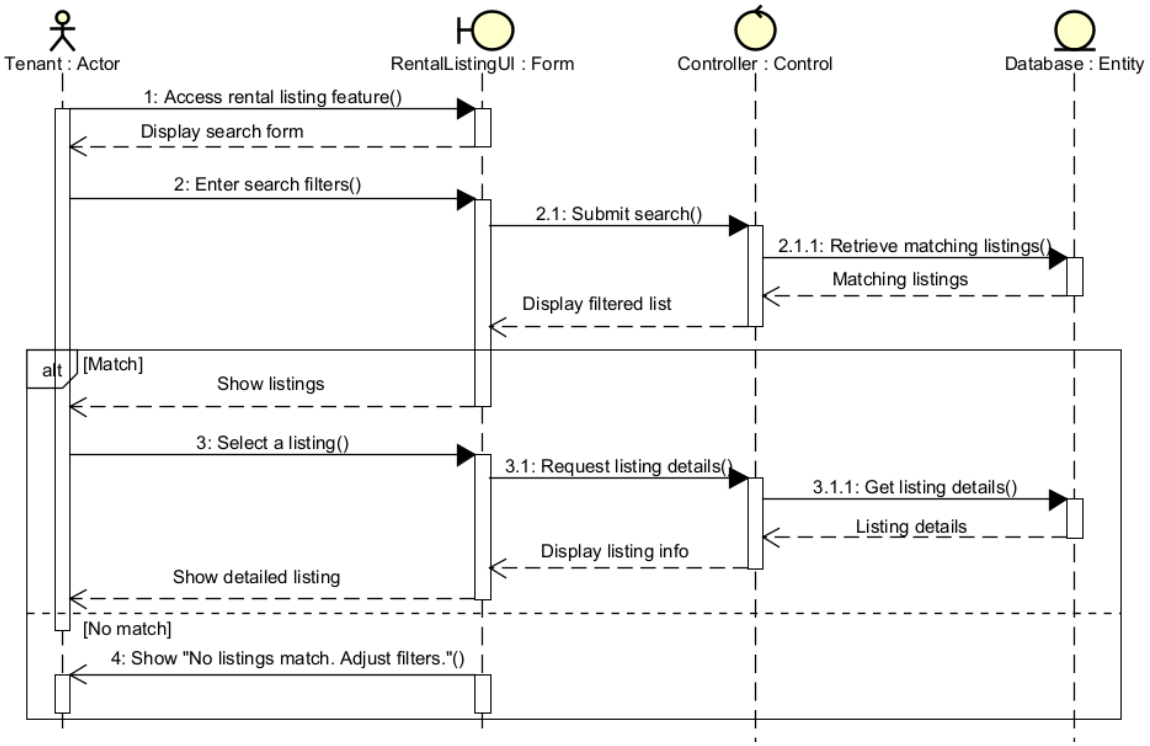


Figure 34. View Rental Listings Sequence Diagram

5.3.5 Bookmark Listings

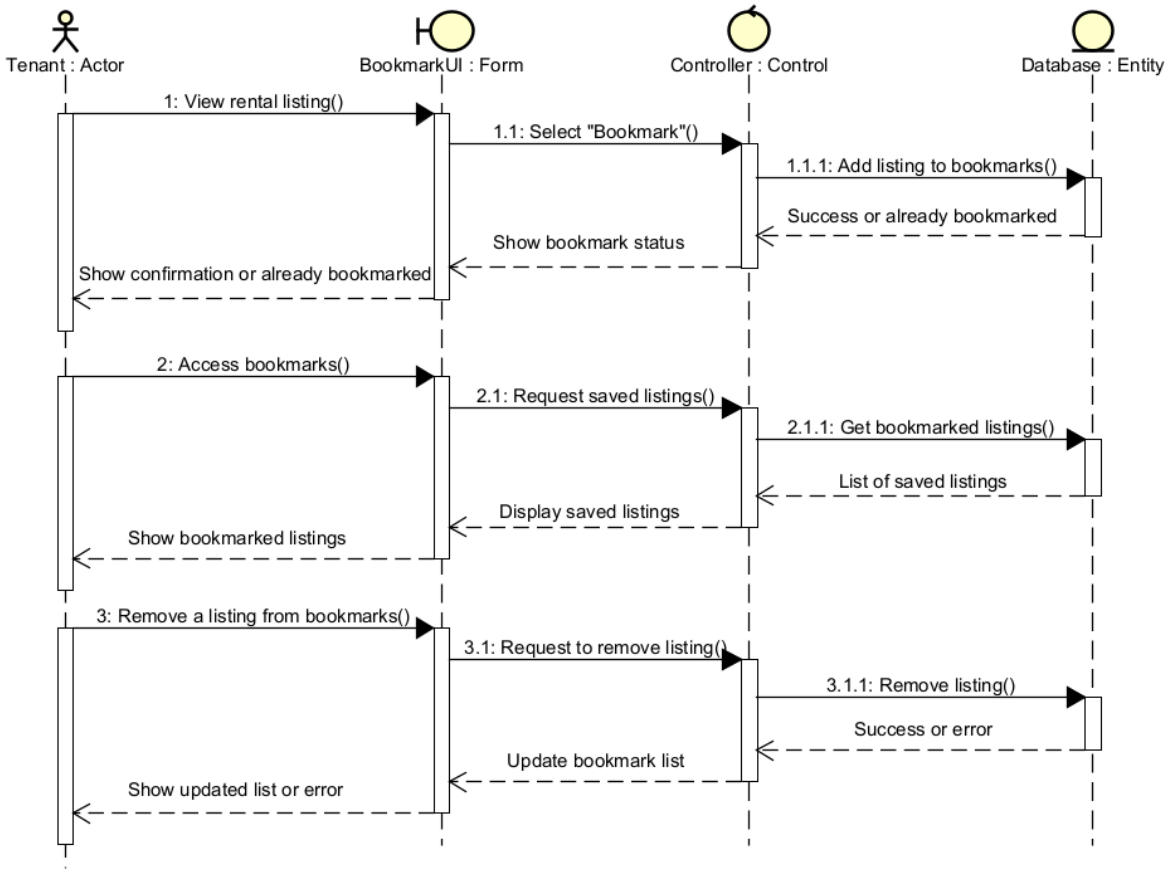


Figure 35. Bookmark Listings Sequence Diagram

5.3.6 Apply for Rental

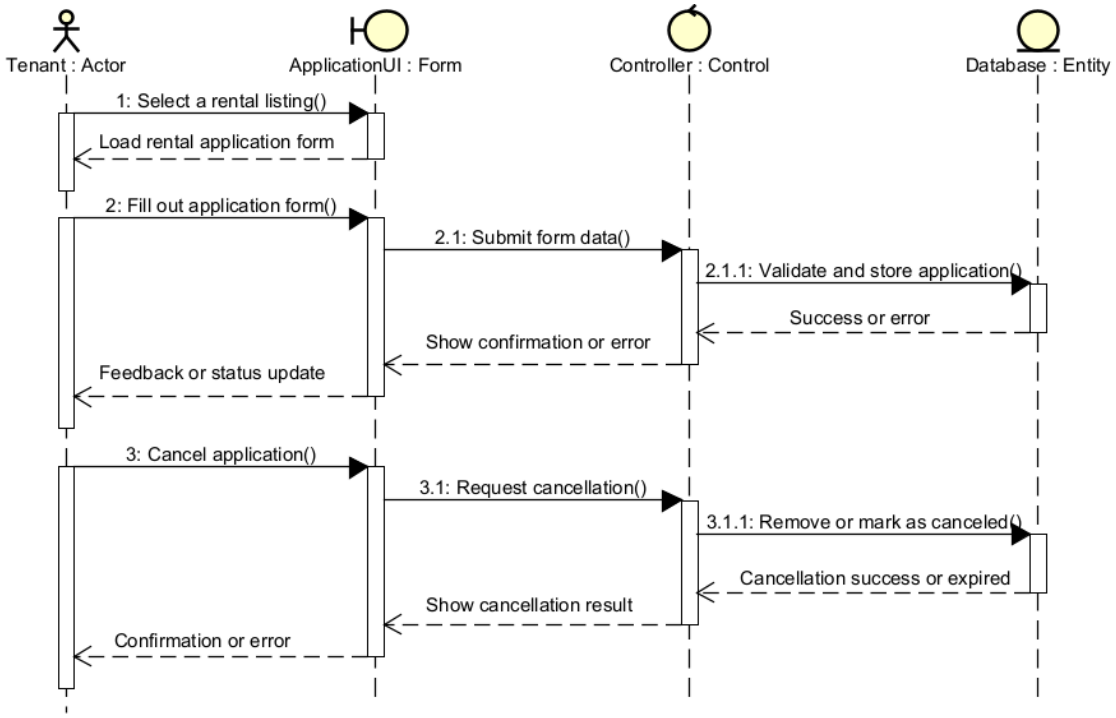


Figure 36. Apply for Rental Sequence Diagram

5.3.7 Request Maintenance Services

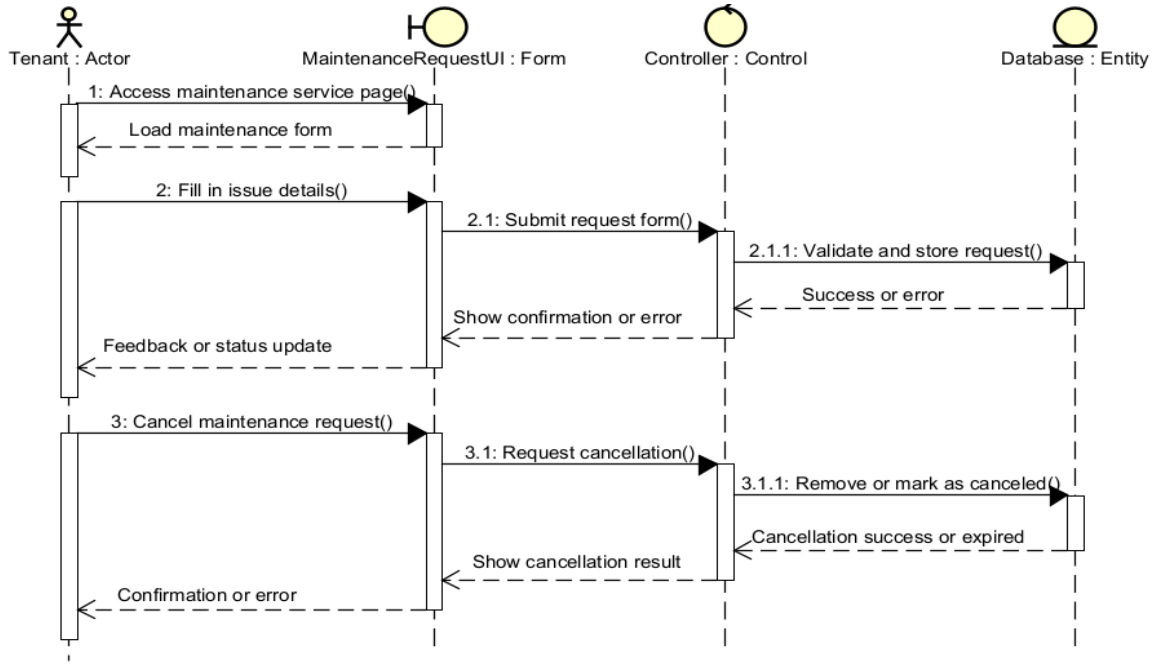


Figure 37. Request Maintenance Services Sequence Diagram

5.3.8 Post Property Listings

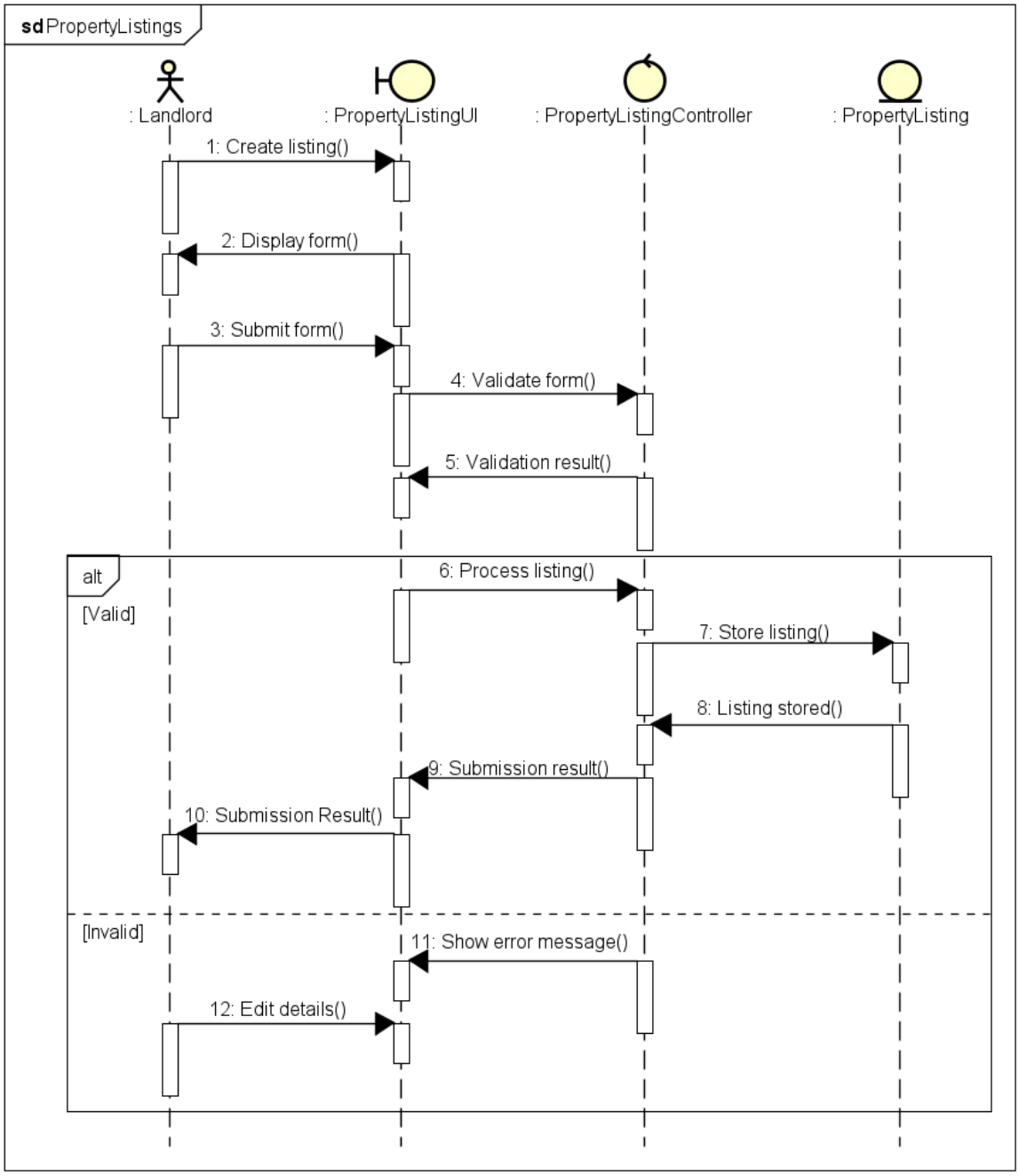


Figure 38. Post Property Listings Sequence Diagram

5.3.9 Review Applications

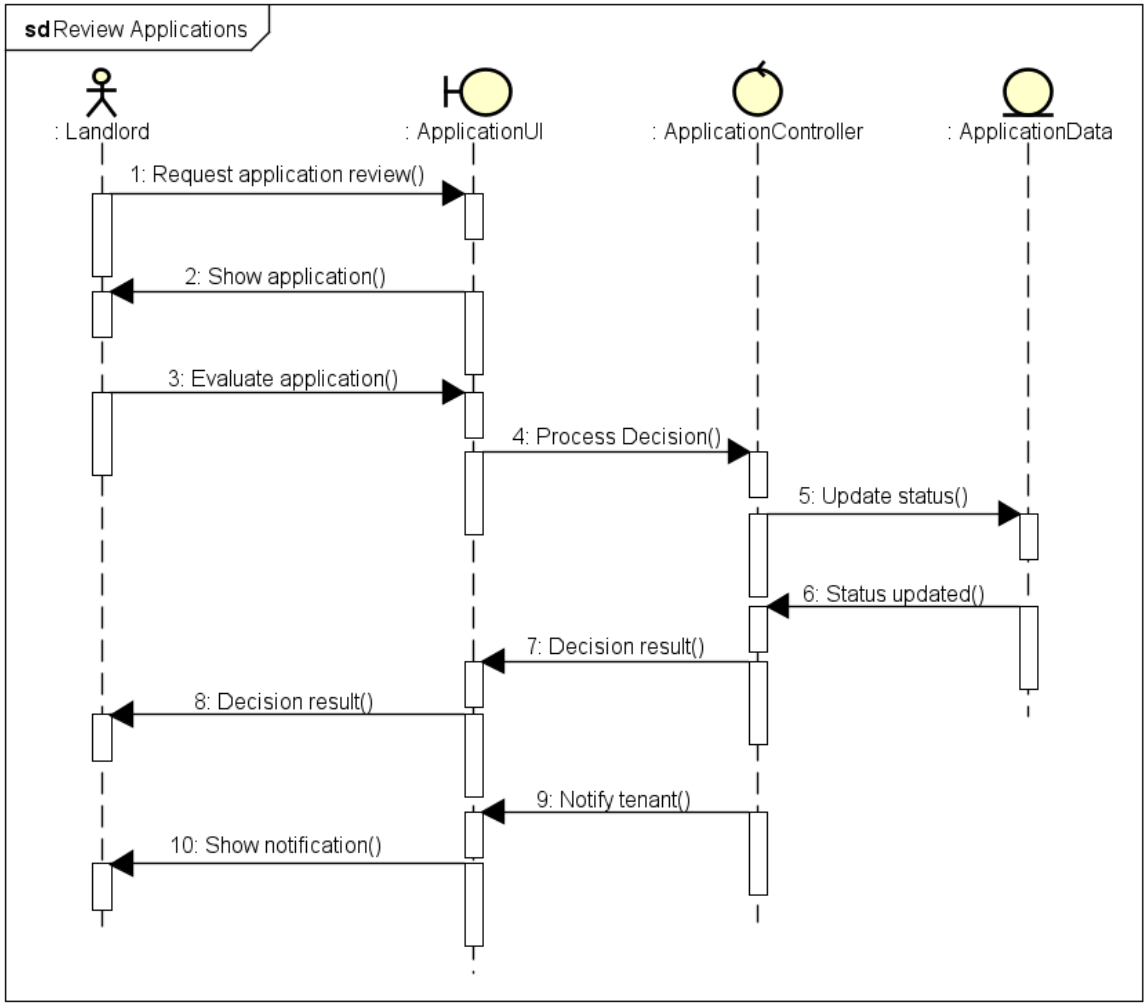


Figure 39. Review Applications Sequence Diagram

5.3.10 Generate Simple Reports

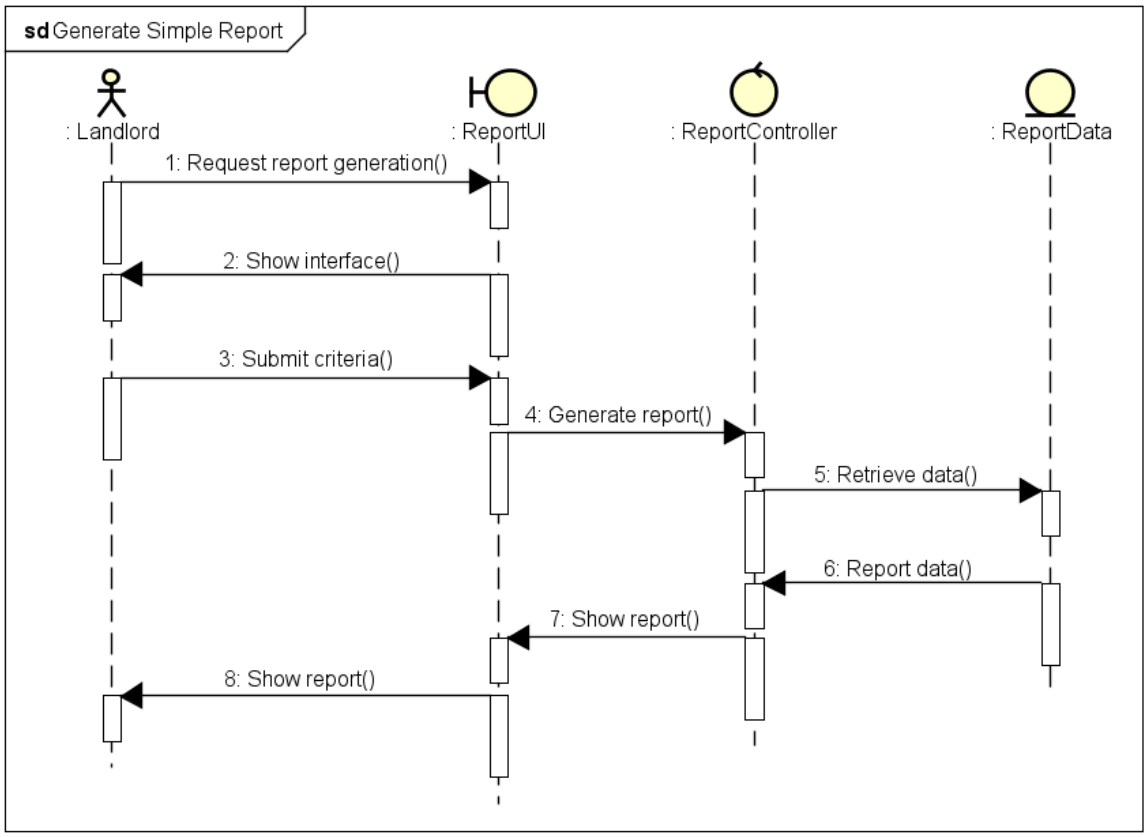


Figure 40. Generate Simple Reports Sequence Diagram

5.3.11 Manage Users and Listings

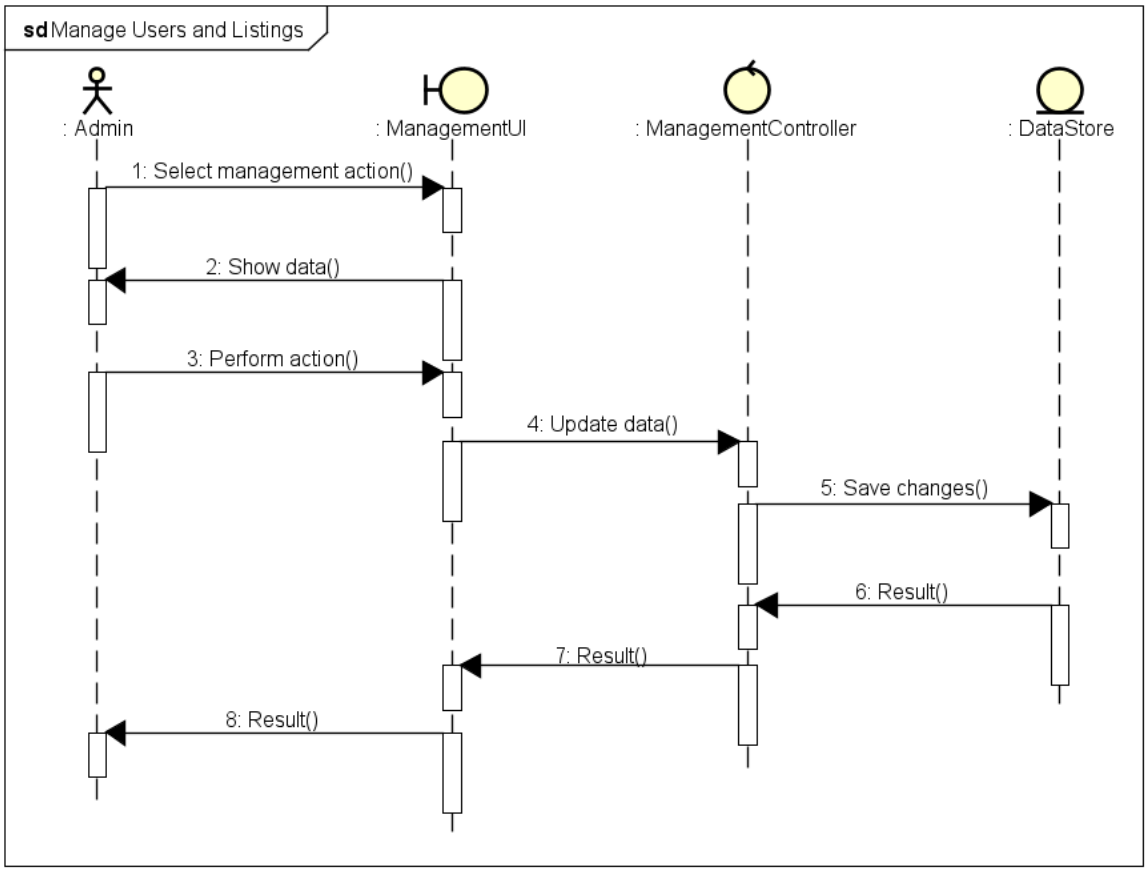


Figure 41. Manage Users and Listings Sequence Diagram

5.4 Rationale for Your Detailed Design Model

Design Justification:

- **Relational Database (MySQL):** Chosen for its robustness and ability to handle structured data with defined relationships, such as linking tenants to rooms. It ensures data integrity and supports complex queries efficiently.
- **MVC Pattern:** Adopted to separate logic (PHP), presentation (HTML5/Tailwind CSS), and data (MySQL). This improves code organization, simplifies maintenance, and supports scalability.

- **HTML5 & Tailwind CSS (Front-end):** Used to build a clean, responsive user interface with utility-first styling. This approach speeds up UI development and ensures consistency across pages.
- **JavaScript (Client-Side Interactivity):** Enhances user experience by enabling features like real-time validation and modal interactions without full page reloads.
- **PHP (Back-end):** Handles server-side logic, form submissions, and communication with the database. PHP is well-supported and integrates seamlessly with MySQL, making it a reliable choice for web applications.

5.5 Traceability from Requirements to Detailed Design Model

Requirement	Design Component
Manage account	Acount model, AcountController, AcountService
Track room availability	PropertyListing model, ListingController, ListingService
Process payments	Payment model, PaymentHistory
Schedule maintenance requests	MaintenanceRequest model
Generate reports	Report model, Report Service, ReportController

Table 12. Traceability from Requirements

CHAPTER 6. TEST PLAN

6.1 Requirements/specifications-based system level test cases

The test cases are derived from the functional and non-functional requirements specified in Chapter 3. Below are selected system-level test cases, focusing on key functionalities and quality attributes:

Test Case 1: Tenant Portal – Apply for Rental

- **Requirement**: FR1 (Tenant Portal)
- **Objective**: Verify that a tenant can apply for rental through the portal.
- **Preconditions**: Tenant is logged in; exists room for rent.
- **Input**: Tenant fill the rental form and submit.
- **Expected Output**:Show nofitication if tenant apply for rental succesfully; page loads within 3 seconds (per NFR1).
- **Test Type**: Positive

Test Case 2: User Authentication - Invalid Credentials

- **Requirement**: FR2 (User Authentication)
- **Objective**: Ensure the system rejects invalid login attempts.
- **Preconditions**: User account exists.
- **Input**: Phone number and password.
- **Expected Output**: Error message ("Invalid username or password") displayed; access denied.
- **Test Type**: Positive

Test Case 3: User Account Management - Create Account

- **Requirement**: FR3 (User Account Management)
- **Objective**: Verify that an admin can create a new user account.
- **Preconditions**: Admin is logged in.
- **Input**: User details (username : "katzfan", phone number: "123456789", password: "katzfan123", role: "tenant").
- **Expected Output**: User account created in the database.

- **Test Type**: Positive

Test Case 4: Generate Report

- **Requirement**: FR5 (Reporting)
- **Objective**: Verify that the system generates a revenue report for a specified period.
- **Preconditions**: Landlord is logged in.
- **Input**: Report contents, report type.
- **Expected Output**: Report is generated.
- **Test Type**: Positive

Test Case 5: Payment Processing – Send Reminder

- **Requirement**: FR7 (Payment Processing)
- **Objective**: Confirm that a landlord can send a reminder for an overdue payment.
- **Preconditions**: Landlord is logged in; an overdue payment exists.
- **Input**: Tenant name, message to remind
- **Expected Output**: Remind message is sent to tenant via message portal
- **Test Type**: Positive

Test Case 6: Usability - Learning Curve

- **Requirement**: NFR4 (Usability)
- **Objective**: Verify that a new user can learn basic functions within 10 minutes.
- **Preconditions**: Users first time using the system
- **Input**: Perform tasks (make payment,…).
- **Expected Output**: Tasks completed within 10 minutes; positive feedback on ease of use.
- **Test Type**: Non-functional

6.2 Traceability of Test Cases to Use Cases

The test cases are mapped to the use cases defined in Chapter 3, ensuring all key functionalities are verified. The use cases referenced are assumed to align with the provided requirements:

Use Case	Test Case	Requirement
Apply for Rental	Test Case 1	FR1
Make Rent Payments	Additional test case	FR1
Use Message System	Test Case 5	FR7
Generate Simple Reports	Test Case 4	FR5
Manage Users and Listings	Test Case 3	FR3
Request Maintenance Service	Additional Test Case	FR1

Table 13. Traceability of Test Cases

This traceability matrix ensures each use case is tested, covering both positive and negative scenarios where applicable.

6.3 Techniques Used for Test Generation

The following techniques were used to generate test cases:

Requirements-Based Testing:

- Test cases were derived directly from functional (FR1–FR7) and non-functional (NFR1–NFR8) requirements.

Use Case-Based Testing:

- Each use case's basic flow and exceptions (from Section 3.2.2) were analyzed to create test cases.

Boundary Testing:

- Applied to inputs like phone numbers, passwords, and date ranges to test edge cases (e.g., invalid phone number formats, empty fields).

Manual Test Design:

- Due to the student project's scope and time constraints, test cases were manually designed, focusing on critical functionalities and common error conditions.

Automated testing with tools like PHPUnit (for PHP unit tests) was considered but prioritized minimally, with manual testing being the primary approach to ensure feasibility within the project timeline.

6.4 Assessment of the Goodness of Your Test Suite

The test suite's quality was evaluated using the following metrics:

Test Coverage:

- **Metric**: Percentage of requirements covered by at least one test case.
- **Result**: Almost functional requirements and key non-functional requirements are covered, ensuring comprehensive validation of critical features.

Defect Detection Rate:

- **Metric**: Number of defects identified during testing divided by total test cases executed.
- **Result**: Initial testing identified UI issues and minor logic, indicating the suite's ability to uncover defects.

Test Case Reusability:

- **Metric**: Ability to reuse test cases across development iterations.
- **Result**: Test cases are modular (e.g., authentication tests apply to multiple user roles), supporting regression testing as the system evolves.

Execution Feasibility:

- **Metric**: Time and resources required to execute the test suite.
- **Result**: The suite can be executed manually within 2–3 hours by one team member, fitting the student project's resource constraints.

The test suite is effective because it covers all critical requirements, detects defects, and is practical for a small team. Limitations include limited automated testing and minimal stress testing due to resource constraints, but these are acceptable given the academic context.

CHAPTER 7. DEMO

7.1 Database

The RRMS database is implemented using **MySQL**, a powerful and reliable relational database management system well-suited for handling structured data with complex relationships. MySQL supports efficient querying, data integrity, and scalability, making it appropriate for managing the core entities of the system such as users, rooms, and rental records. Below is an overview of the database structure along with a sample dataset used for demonstration purposes.

The database consists of the tables that designed in Chapter 5, include : Account, List, Payment, Report, Message.

7.2 Source Code

The RRMS source code is written in **PHP** for server-side logic, with **HTML5** and **Tailwind CSS** for the front-end presentation, and **MySQL** for data storage. The codebase follows the **Model-View-Controller (MVC)** pattern, as described in Chapter 5, to promote modularity, maintainability, and separation of concerns. Below is an overview of the code structure along with sample snippets used in the demonstration.

Code Structure :

- /models/: Contains PHP classes for data management.
- /controllers/: Contains PHP scripts for business logic.
- /views/: Contains HTML/PHP templates for the UI.
- /config/: Database connection settings (db.php).

Sample code snippet :

Below is a simplified PHP snippet from HomeController.php

```php
<?php
require_once 'Controller.php';
class HomeController extends Controller {
    public function index() {
        // Check if user is logged in
        if (!isset($_SESSION['user'])) {
            // Redirect to login if not logged in
            header('Location: index.php?controller=account&action=login');
            exit;
        }
        $role = $_SESSION['role'];

        // If logged in, render home view
        require_once 'view/home.php';
    }
}
```

Figure 42. HomeController.php

The code is hosted in a GitLab repository, with version control ensuring traceability (per Chapter 2). For the demo, the system is deployed on XAMPP accessible via a browser to meet NFR6 (compatibility).

7.3 Testing

The testing section demonstrates the results of the test plan outlined in Chapter 6, verifying that RRMS meets its requirements. The demo focuses on executing key test cases to show the system's functionality and quality.

The following test cases from Chapter 6 are executed during the demo:

- **Test Case 1: Tenant Portal – Apply for Rental**
  - **Action :** Tenant log in and choose "Apply for Rent".
  - **Result :** Room booked successfully.
  - **Status** : Pass
- **Test Case 2: User Authentication - Invalid Credentials**
  - **Action :** Attempt login with incorrect password**.**
  - **Result :** Error message ("Invalid username or password") displayed; access denied.

    - **Status** : Pass
- **Test Case 3: User Account Management - Create Account**
    - **Action :** Administrator creates a new tenant account
    - **Result :** Account added to the database; visible in user list.
    - **Status** : Pass
- **Test Case 4: Generate Report**
    - **Action :** Landlord generates a revenue pdf report.
    - **Result :** Report generated successfully.
    - **Status** : Pass
- **Test Case 5: Payment Processing – Send Reminder**
    - **Action :** Landlord sends a reminder for an overdue payment.
    - **Result :** Reminder message logged in the portal.
    - **Status** : Pass
- **Test Case 6: Usability - Learning Curve**
    - **Action :** A new user performs tasks within 10 minutes.
    - **Result :** Tasks completed in 8 minutes; user reports intuitive interface.
    - **Status** : Pass

Test results are logged in a simple report (e.g., a table showing pass/fail status), demonstrating that the system meets its requirements. Any defects found during testing (e.g., minor UI alignment issues) were fixed prior to the demo, ensuring a stable presentation.

REFERENCES

Pressman, R. S., & Maxim, B. R. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education.

Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson Education.

Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3rd ed.). Prentice Hall.

IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications.