PROPERTY MANAGEMENT SYSTEM

Final Document

Kripa Dixit

Shivanesh Bharathi

Varsha Kampli

Swati Awasthi

Nikhil Mukhedkar

Whitney Adkins

DECEMBER 5, 2017

TEAM 5: PRISM- BLUE LIGHTNING

624-601

Table of Contents

1.0 Business Case

1.1 Background Information

Ms. Jane Mywick is a real estate property owner who rents units to individuals and families; depending upon the capacity of the unit. She currently uses quick books to track expenses involved in her business. However, rents are tracked manually. Additionally, few of her tenants receive government subsidies from Housing Assistant Plan (HAP). Which means, part of their rent comes from HAP which is again subject to change as per tenant's economic condition. This makes tracking rent data even more difficult. She wishes to employ an efficient property management system which allows her to track information about her property units, tenants, rents and their batch processing remotely. Currently, the available property management packages are too detailed and complicated for her use, and she needs a simple dedicated system on her personal device.

1.2 Proposed Solution and scope

After realizing the business value of the property management system, we have decided to develop a system which can be easily hosted and scaled. The initial proposal of property management system implemented a simple UI, dedicated backend and integrated database for Ms. Mywick's system that brought the features for capturing data of units owned by her, tenant's information, payment processing and tracking etc. Additionally, the system offers some functionalities that are vital for her; which includes report generation, reminders for lease and rents.

However, since we have decided to scale the system up, it expands the scope of the system. The new system will be hosted on a cloud and will be offered as a service to the clients. This will make the system light weight which subsequently will allow for faster installation and operation, better performance and flexibility for customization. The service nature of the system will eliminate the need of hosting any data stores or business logic servers by the clients. A simple desktop application – compatible with diverse platforms – is the final outcome of the proposed solution.

Following section enlists the features offered by the system:

- Create a system admin (business side)
- Create a manager (client side)
- Add a unit to the property
- Tracks tenants' and lease information
- Tracks payment information (incorporating HAP subsidies also) and export data to quick books
- Allows rent adjustment features
- Batch processing of rents
- Archiving desired information
- Report Generation – outstanding rent, overdue, rent history, unit availability, income

Benefits

- A simple and customizable system for personal use as opposed to complex systems available in the market
- Massive improvements over current manual tracking process for rent
- Batch processing and manual adjustments of rents
- Report generation for ease of record access
- An extremely light weight and faster system
- No need of hosting the system on personal hardware. Just a device is needed to run the application.
- Cost reduction owing to minimal investment in hardware as well as maintenance

1.2.1 Assumptions

- Payment is only through check
- One property can have multiple managers
- One lease can many tenants
- Archival is performed twice a year
- The only actors are system admin and manager
- Scheduler class which performs backend operations

Outline plan

The team has increased the use cases from 10 to 13 in order to incorporate the business expansion. After careful consideration of the peer feedback and expansion in business requirements, the team has identified improvements in the proposed solution and scope. The team has come up with an overview of the system through a class diagram and better visualization of the system through some mock ups. Team will continue to follow agile practices by taking feedback from clients and peers to incorporate improvements.

1.3 Functional Requirements:

1. Add Property Management Group: The system provides a feature to add a new Property Management Group.

2. Add Property: The system must allow the manager to add Property Management Group to each property managed by him/her.

3. Add Property Unit: The system must allow the manager to add Property unit to a Property Management Group managed by him/her.

4. Add Tenant: The system must allow the managers of the properties to add tenants to the units. The system must now allow access to properties managed by other managers or property groups

5. Enter Lease Details: The system must allow managers to enter lease details and generate lease when a new tenant is added, or the old lease has expired

6. Display Unoccupied Properties: The system must provide manager the ability to generate the occupancy chart and display properties available for rent to new tenants.

7. Outstanding Rent Report: The system must be able to generate outstanding rent report for all the properties managed by the manager.

8. Total Income by Unit: The system must be able to generate Total Income by unit report for all the units managed by the manager.

Test Strategy: For this requirement, the Manager, Property, Unit, Lease and Tenant classes will need to be tested.

9. Archive rent and payment info: The system should archive rent and payment information automatically every 6 months by storing the information in the database.

10. Tenant lease-up: The system should be able to generate lease for a tenant and a unit which is to be valid for specific period of time. This lease engagement is to be permanently recorded in the database and rent payment should be made regularly against it.

11. Rent history by Unit: Rent history by Unit is a report that can be run by the manager/user to see and analyze rent revenue by Unit. The data for this report is fetched from the database and presented to the user in the form of excel report.

12. Record and track payment: The payment made by the payer should be recorded and tracked by the system. The system should track details like check no., payer details, payment source, etc. for each check and store it in the database.

13. Generate batch grid: The system takes all the received checks as input and generates a batch grid of checks to be deposited to the bank.

1.4 Non-Functional Requirements:

14. Cross browser compatibility: User must be able to access the system over multiple browsers.
15. Ability to render on multiple resolution devices: User should be able to access the application over mobile, laptop and tablet devices.
16. Ability to handle 10000 users: The system should support about 1000 property management groups with 10,000 units each
17. Pages must load within 4 seconds: All the pages and reports should load within 4 seconds
18. AES encryption standard for passwords: The password stored in the database should be stored using AES encryption. This will be tested by logging in into the Database and looking at the password column. The password should be encrypted using AES algorithm and should be unreadable.
19. Usability requirement: Tooltips for textboxes, help button, contact support feature in each page

2.0 System Diagrams

2.1 Deployment Diagram

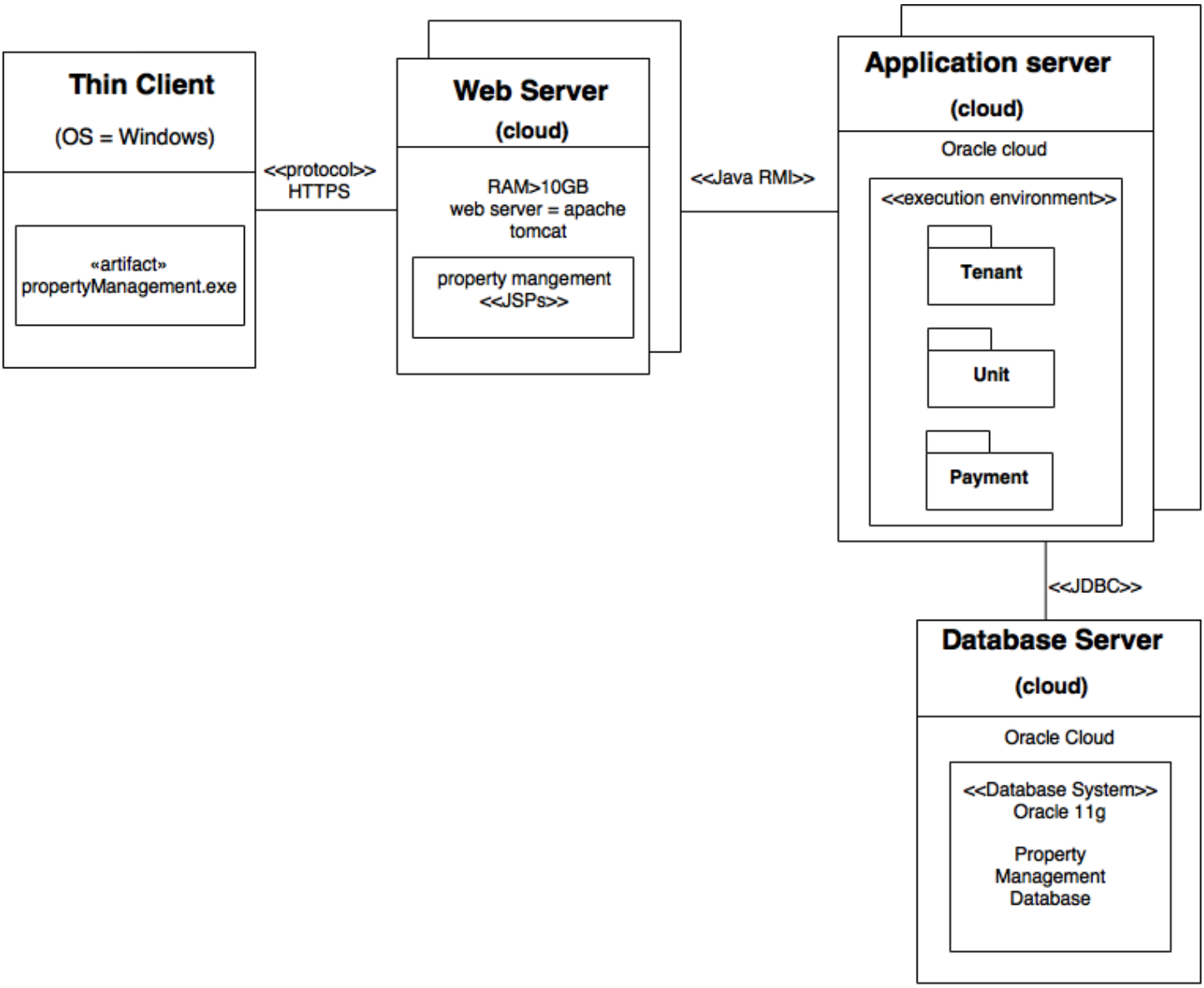


Figure 2.1

The deployment diagram shows the architecture of the property management system. The Application Server contains application packages and artifacts which need to be deployed and it interacts with the Oracle 11g database which stores all the application data. The application server and database server are hosted on the cloud. Thin client interacts with the web server to access the packaged files of the applications.

2.2 Package Diagrams

2.2.0 Application Layer

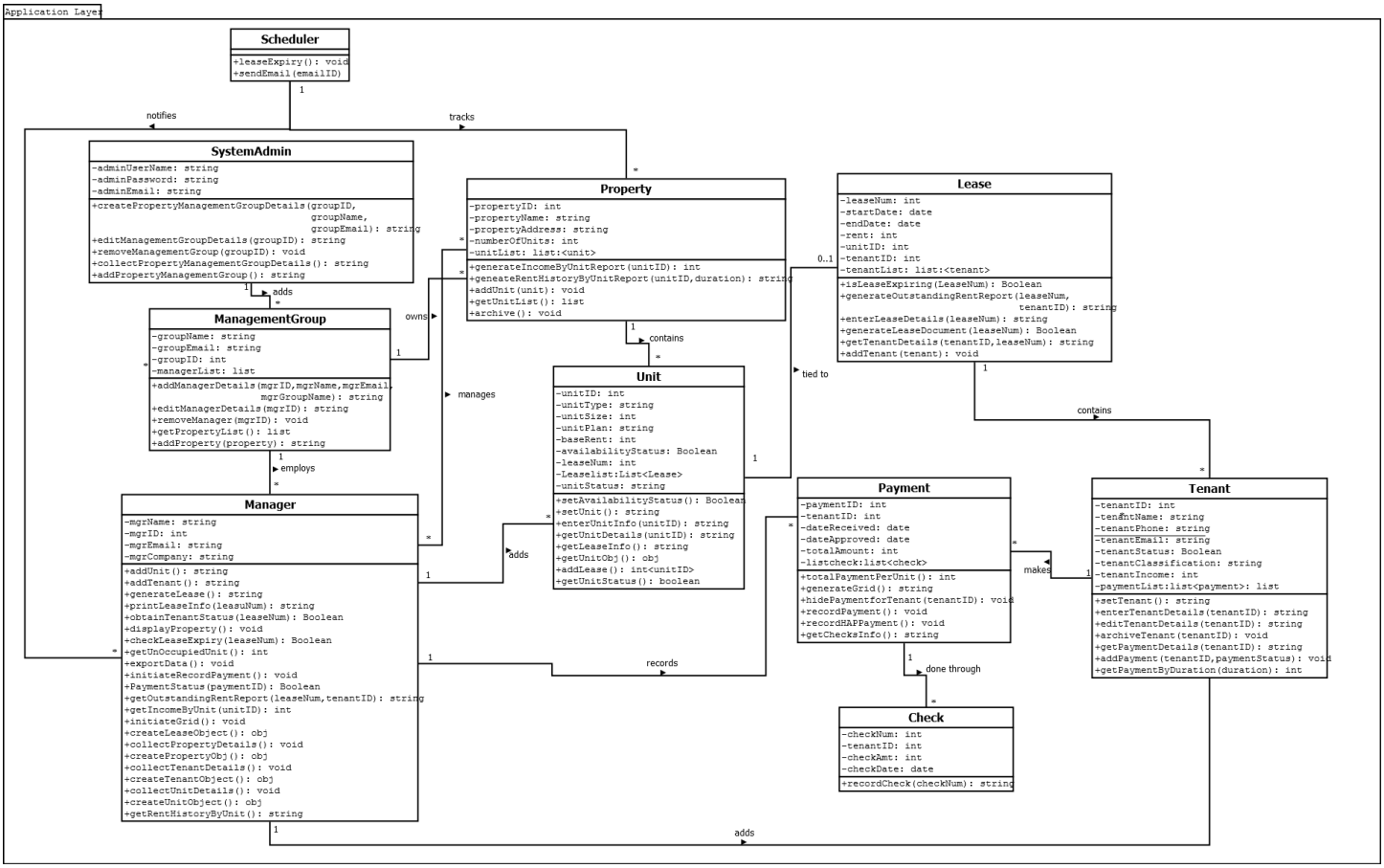


Figure 2.2.0

Figure 2.2.0 is the package diagram for the application layer. It depicts all the classes which are required to capture the business functionality.

2.2.1 Data Layer

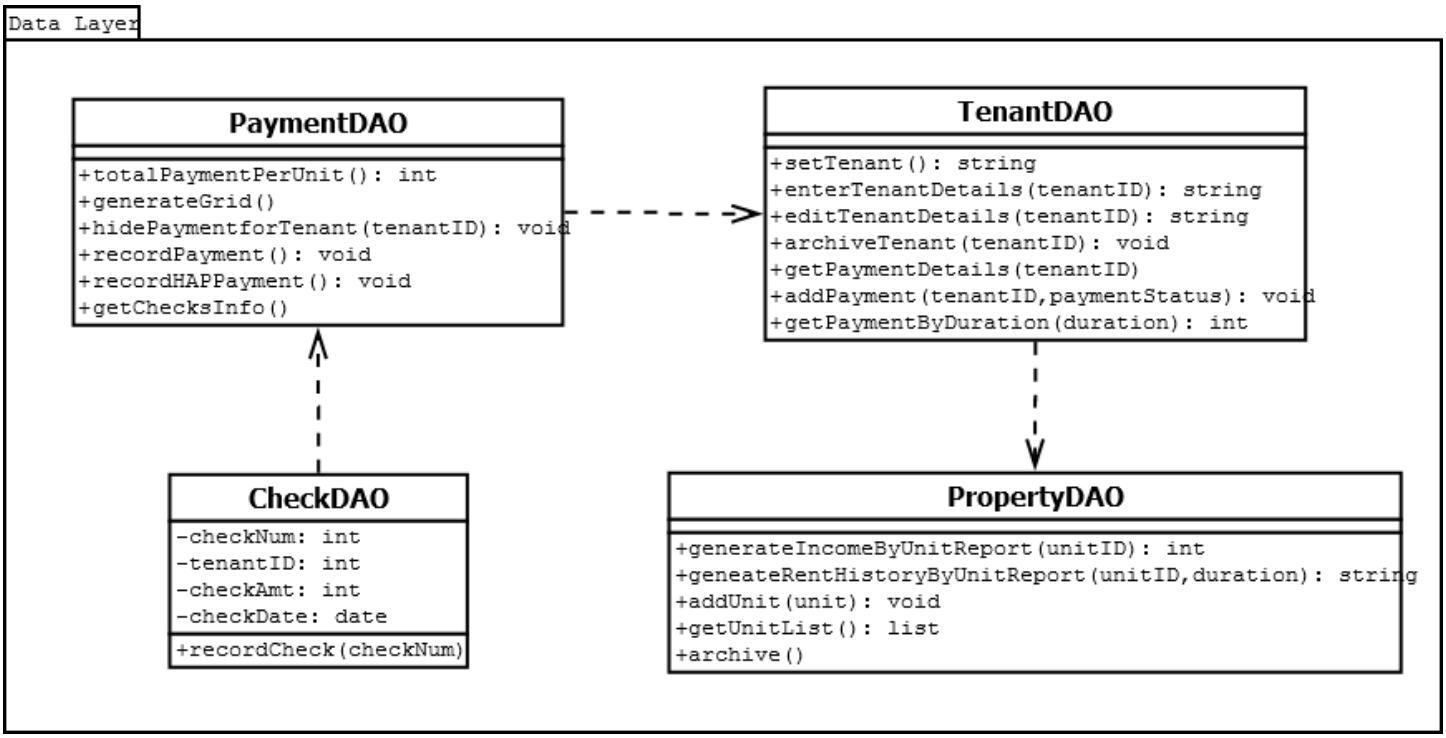


Figure 2.2.1

Figure 2.2.1 represents the data layer. It contains the predominant data access objects required in the system.

2.2.2 UI Layer

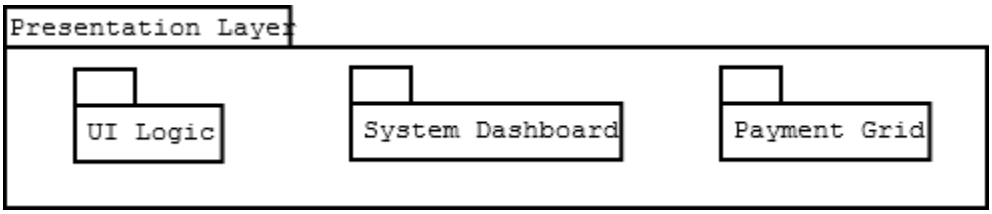


Figure 2.2.2

Figure 2.2.2 is the package diagram for UI of the system. It contains the predominant presentation components such as the dashboard logic, payment grid etc.

2.2.3 Overall Package Diagram

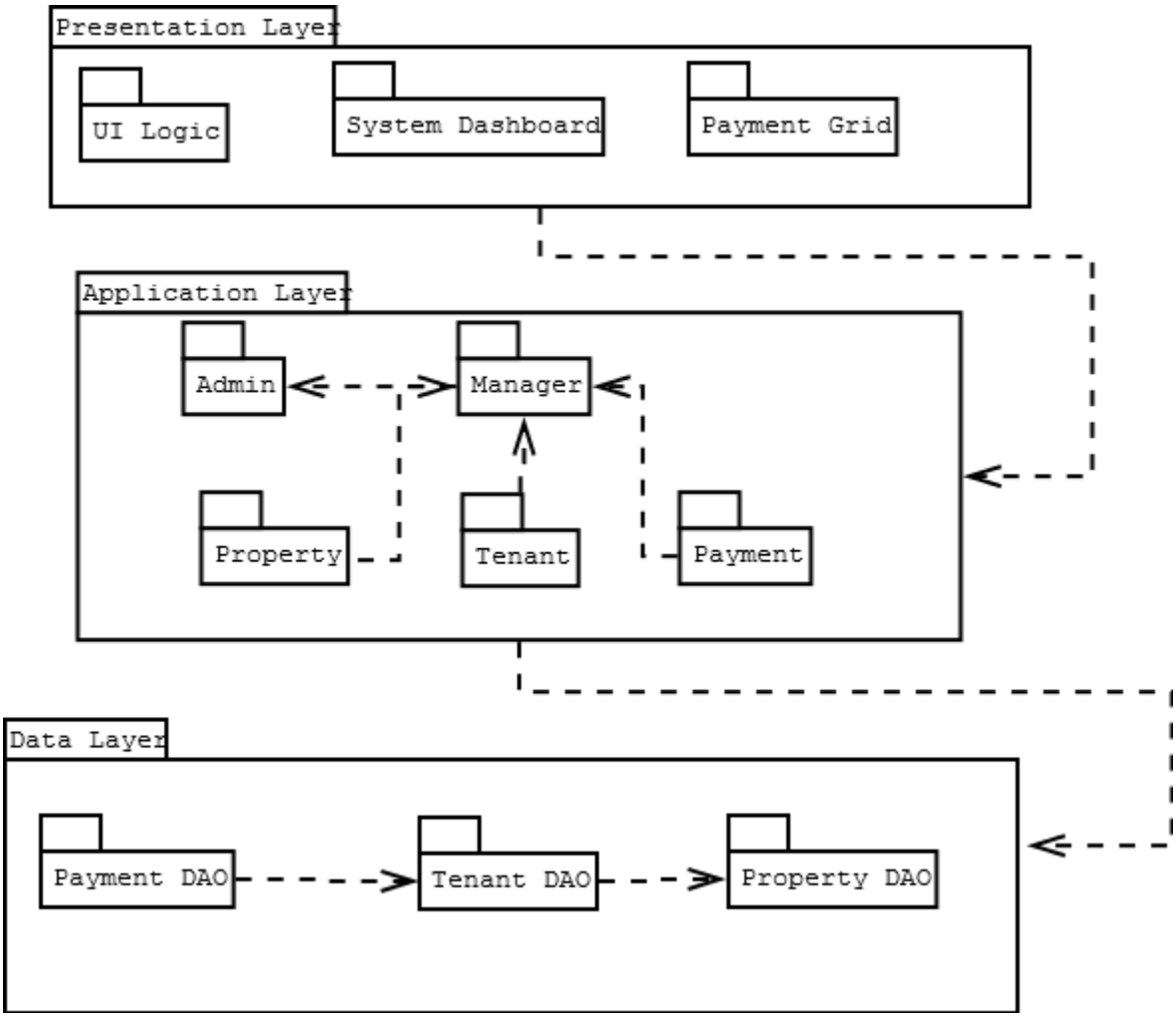
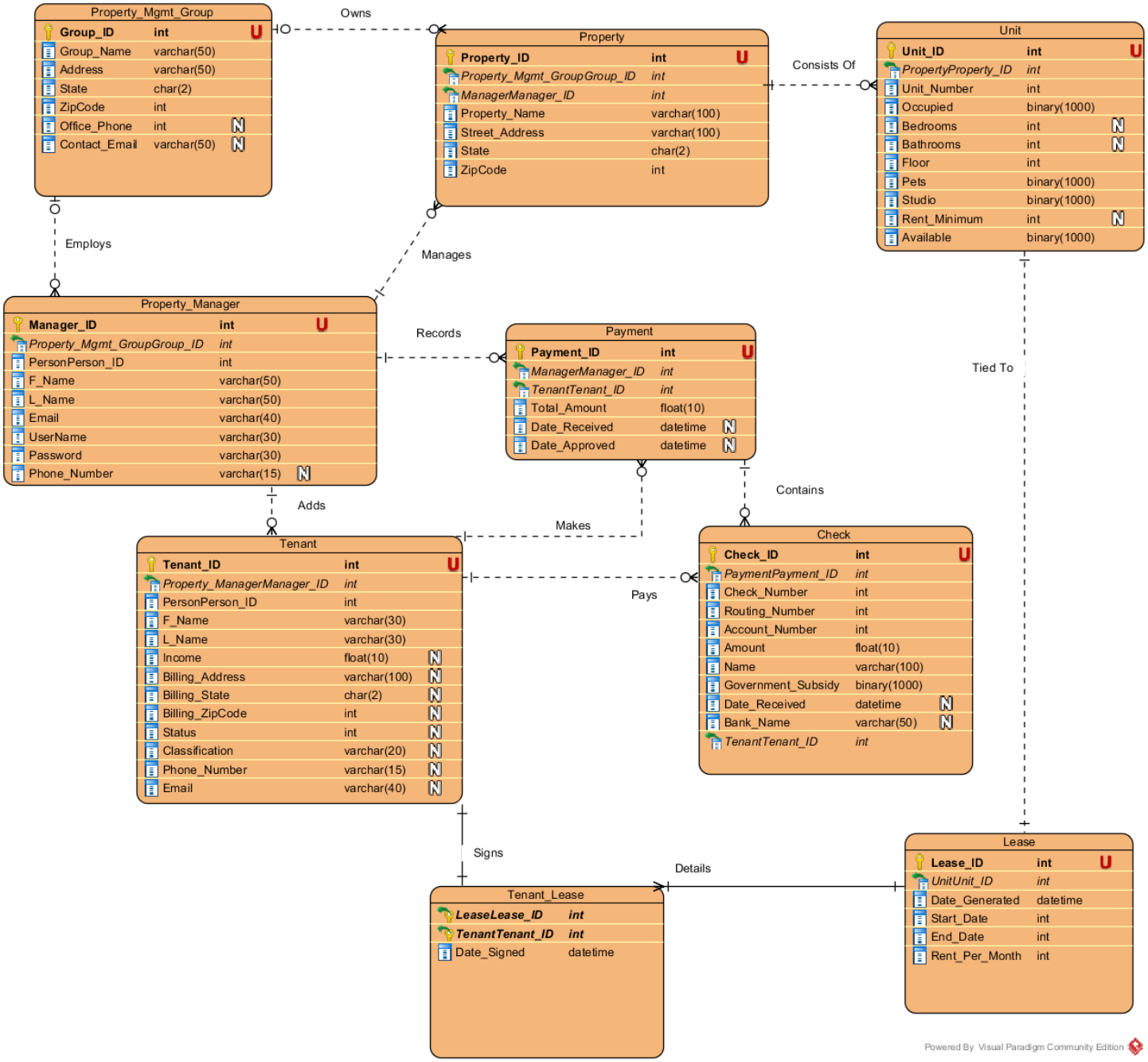


Figure 2.2.3

Figure 2.2.3 is the package diagram for the entire system. It is divided into 3 layers, i.e. UI, Business(application) and data layers. They are interacting with each other as depicted.

2.3 ERD- Physical



LEGEND

- U Unique
- N Nullable
- Primary Key
- Foreign Key
- Standard Column

Figure 2.3

Figure 2.3 is the physical Entity Relationship Diagram for the system.

3.0 Use Cases and Sequence Diagrams

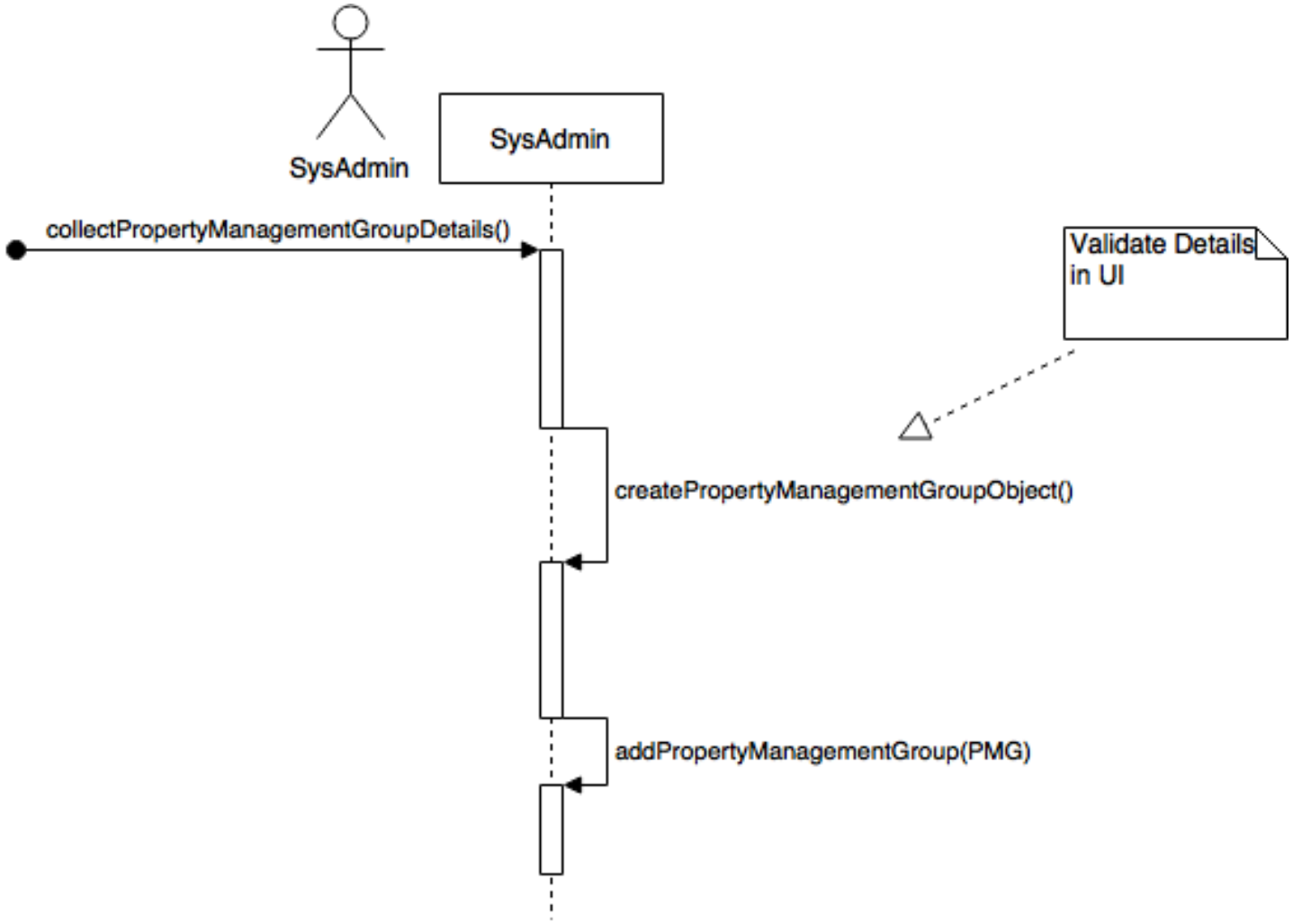
In this section we have listed our use cases, each followed by their respective sequence diagram.

3.1 Add Property Management Group

Use Case Name: Add property management group	UC-1	Priority: Medium
Actor: System Admin		
Description: The system provides a feature to add a new Property Management Group		
Trigger: System Admin initiates the process of adding a new property management group Type ☑ External ☐ Temporal		
Preconditions: The System Admin must have received property management access privileges with valid credentials		
Normal Course: 1. System Admin selects "Add property Management group" feature 2. System Admin enters the name of the property Management group 3. System Admin clicks the confirmation for adding the property management group 4. System displays the message, "New property management group" 5. Exit from the use case		Information for the step ←Property group details
Alternative Course 4a. System Admin enters a existing property management group name 1. System displays the message, "PMG already exists; please input a different group" 2. System Admin inputs a different group name 3. System resumes normal course (step 3)		→System alert ←Property Management group details
Postconditions: The new property management group is added to the system,		
Exceptions: E1: The system is unable to add the property management group (occurs at step 5) 1. The system displays message "Error! <Error code>Unable to add the PMG; please try again" 2. The system terminates the case.		
Summary Inputs Source Outputs Destination		

Property Group Name	System Admin	Success Message Error Message	System Admin

3.1.1 Add Property Group Sequence Diagram



Classes Interacting: SysAdmin

SSD:

- System admin clicks on Add Property Management group button in the UI
- This function collects information entered in the UI like the property management group name, primary contact and creates a new property management object with the entered data.
- The system admin class maintains a list of property management groups

3.2 Add Property

Use Case Name: Add property	UC-2	Priority: Medium
Actor: Manager		
Description: The system provides a feature to add a new property to an existing property management group		
Trigger: Manager initiates the process of adding a new property to the property management group Type ☑ External ☐ Temporal		
Pre-conditions: The Manager must have received property management access privileges with valid credentials		

Normal Course:	Information for the step
1.Manager selects "Add property " feature	
2.Manager selects the Property Management group where the property needs to be added	
4.Manager enters new property details	←Property details
5.Manager clicks the confirmation for adding the new property	
6.System displays the message, "New Property added"	→Success Message
7.Exit from the use case	
Alternative Course	
4a. Manager enters a new property name for the new property 1. System displays the message, "Property already exists; please enter different information"	→System alert
2. Manager inputs a different set of information	←Property details
3. System resumes normal course (step 6)	

Post-conditions:

The new property is added to the property Management group

Exceptions:

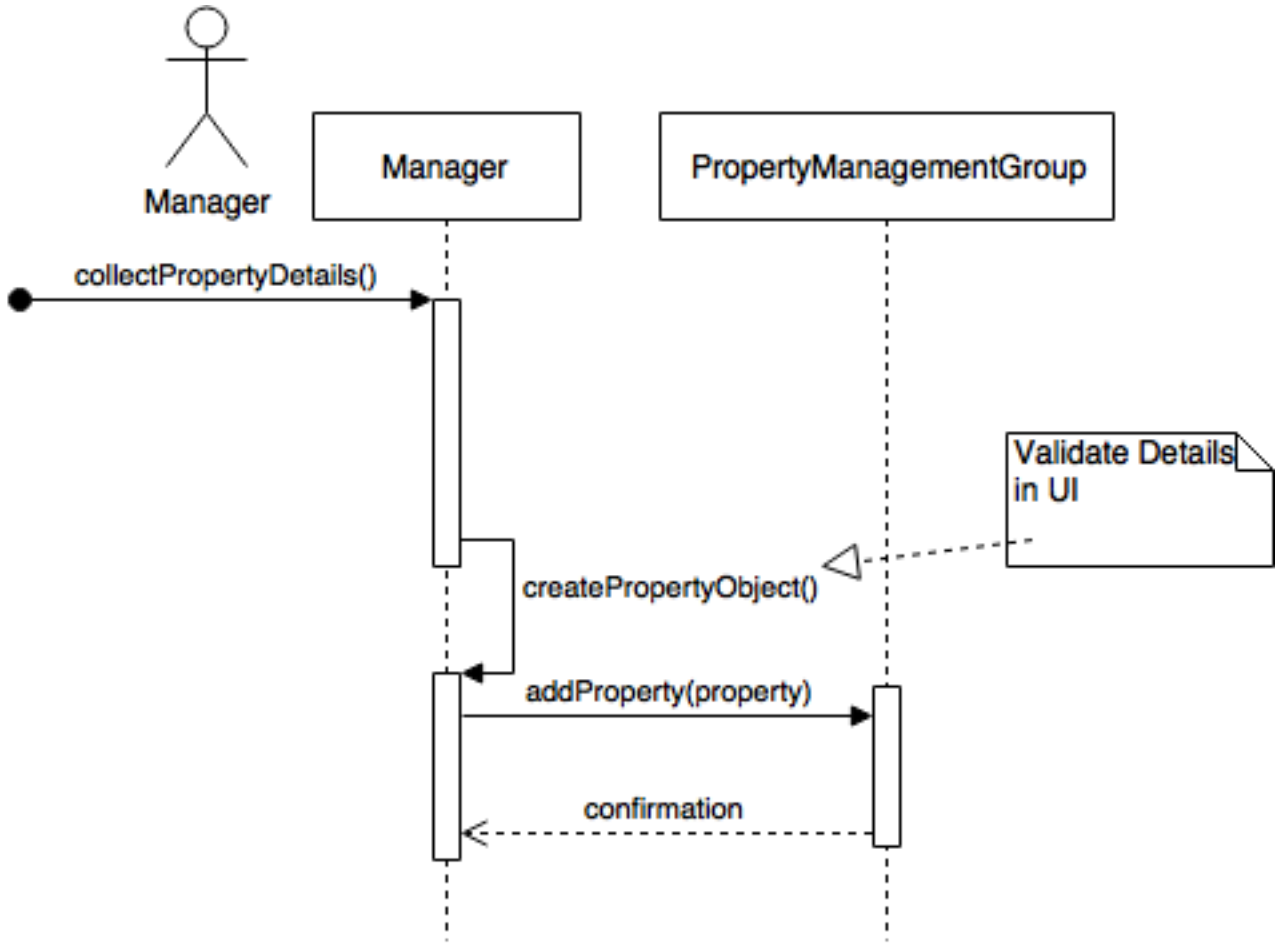
E1: The system is unable to add the property to the group (occurs at step 5)

1. The system displays message "Error! <Error code>Unable to add the property ; please try again"
2. The system terminates the case.

Summary

Inputs	Source	Outputs	Destination
Property Details	Manager	Success Message System Alert	Manager

3.2.1 Add Property Sequence Diagram



Classes Interacting: Manager, Property Management Group

SSD:

- Manager clicks on Add Property button in the User Interface
- This function collects information entered in the UI like the property name, location, address and creates a new property object with the entered data.
- This property object is passed to the PropertyManagementGroup class where the object is added to the list of properties in the property management group

3.3 Add Property Unit

Use Case Name: Add property unit	UC-3	Priority: Medium
Actor: Manager		
Description: The system provides a feature to add a new unit to an existing property		
Trigger: Manager initiates the process of adding a new unit to the property Type ☑ External ☐ Temporal		
Preconditions: The Manager must have received property management access privileges with valid credentials		

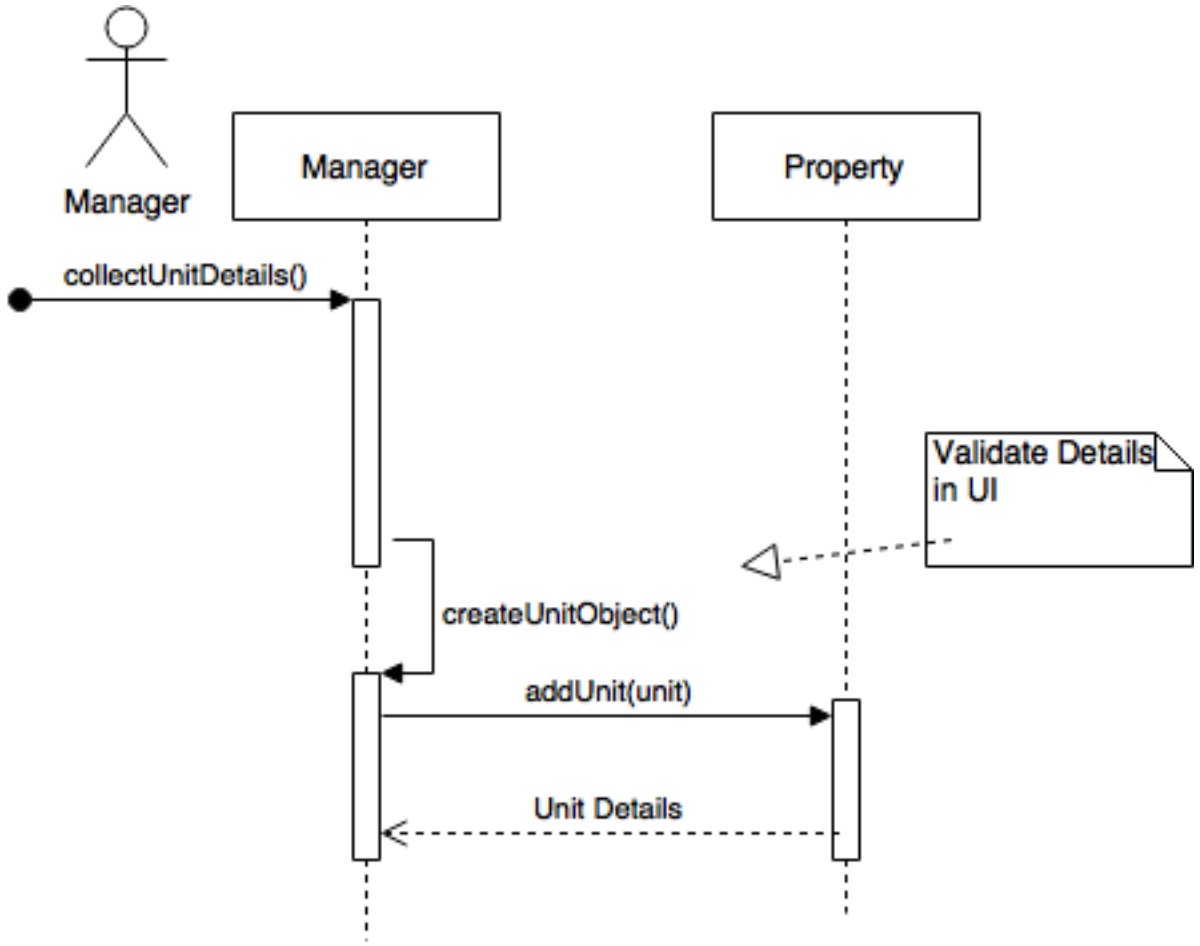
Normal Course:	Information for the step
1. Manager selects "Add unit" feature 2. Manager selects the Property where the unit needs to be added from the property drop down 3. Manager selects the type of unit from the type drop down 4. Manager enters a new number for the new unit 5. Manager clicks the confirmation for adding the unit 6. System displays the message, "New unit added" 7. Exit from the use case	←Property name ←Property type ←Unit Number →Add confirmation
Alternative Course 4a. Manager enters a new number for the new unit 1. System displays the message, "Unit number already exists; please input a different number" 2. Manager inputs a different number 3. System resumes normal course (step 6)	→System alert ←Unit number

Post-conditions:
The new unit is added to the property
Exceptions: E1: The system is unable to add the unit to the property (occurs at step 5) 1. The system displays message "Error! <Error code>Unable to add the unit; please try again" 2. The system terminates the case.

Summary

Inputs	Source	Outputs	Destination
Property Name Unit Type Unit Number	Manager	Success Message Error Message	Manager

3.3.1 Add Property Unit Sequence Diagram



Classes Interacting: Manager and Property

SSD:

- Manager clicks on Add Unit button in the User Interface
- This function collects information entered in the UI like the unit number, size, floor number etc and creates a new unit object with the entered data.
- This unit object is passed to the Property class where the object is added to the list of units in the property

3.4 Add Tenant

Use Case Name: Add Tenant	UC-4	Priority: Medium
Actor: Manager		
Description: The system provides a feature to add a new tenant to the existing system database		
Trigger: Manager initiates the process of adding a new tenant Type ☑ External ☐ Temporal		
Preconditions: The Manager has logged into the system with valid credentials.		

Normal Course:	Information for the step
1. Manager selects "Add Tenant" feature	
2. New tenant form opens up	
3. Manager enters names of 1 or 2 tenants per unit	←Tenant name
4. Manager enters phone numbers for the tenants	←Tenant phone number
5. Manager records the deposit amount paid by the tenant	←Deposit information
6. Manager clicks the confirmation for adding the tenant	
7. System displays the message, "New tenant added"	→Display confirmation
8. Exit from the use case	
Alternative Course 3a. Manager updates a tenant name for the unit	
1. Manager enters new tenant name	←Tenant name
2. Manager enters the new phone numbers for the tenant	←Phone number
3. System resumes normal course (step 5)	

Post-conditions:
The new tenant is added to the property

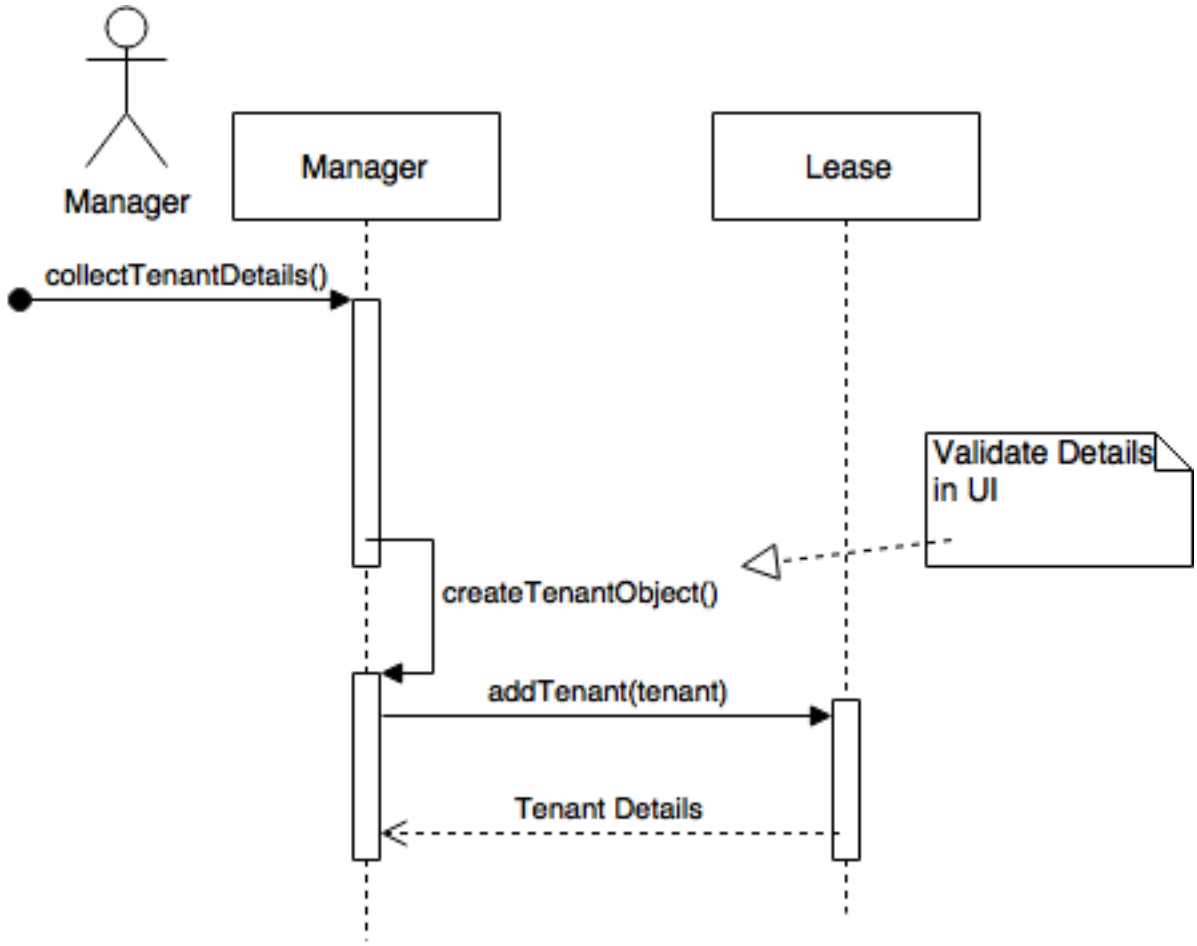
Exceptions:
E1: The Manager enters more than 2 tenant names (occurs at step 3)
2. The system displays error message "Error! <Error code>Maximum two tenant names allowed"
3. The Manager does not add any more tenant names

E1: The Manager enters more than 3 phone numbers per tenant (occurs at step 4)
1. The system displays error message "Error! <Error code>Maximum three phone numbers allowed"
2. The Manager does not add any more phone numbers

Summary

Inputs	Source	Outputs	Destination
Tenant Name Tenant Phone numbers Deposit Amount	Manager	Success Message Error Message	Manager

3.4.1 Add Tenant Sequence Diagram

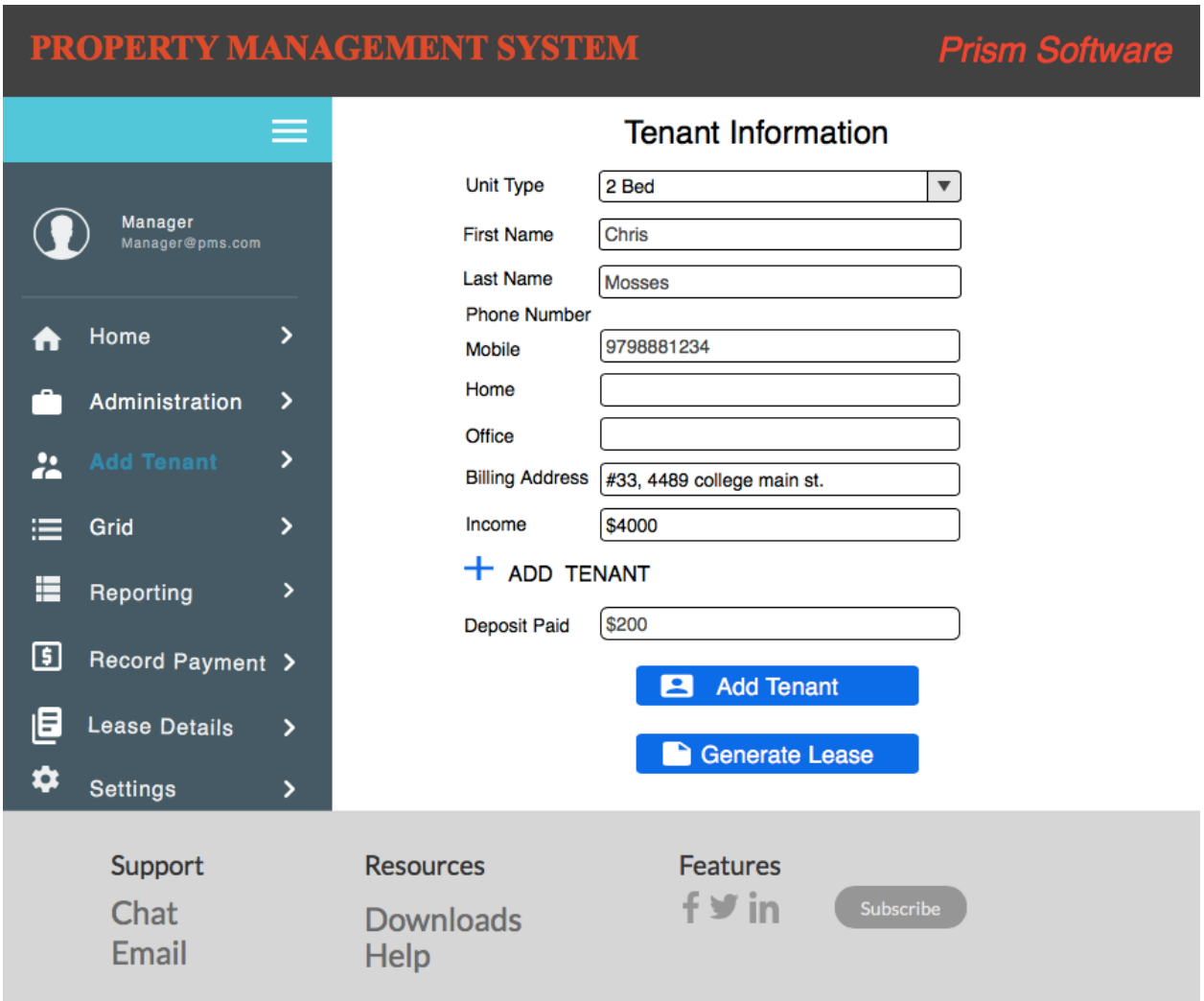


Classes Interacting: Manager and Lease

SSD:

- Manager clicks on Add Tenant Button in the User Interface.
- This function collects information entered in the UI like the tenant name, phone, and creates a new tenant object.
- This tenant object is passed to the Lease class where it is added to the list of tenants associated with the Lease

3.4.2 New Tenant Screen Mockup

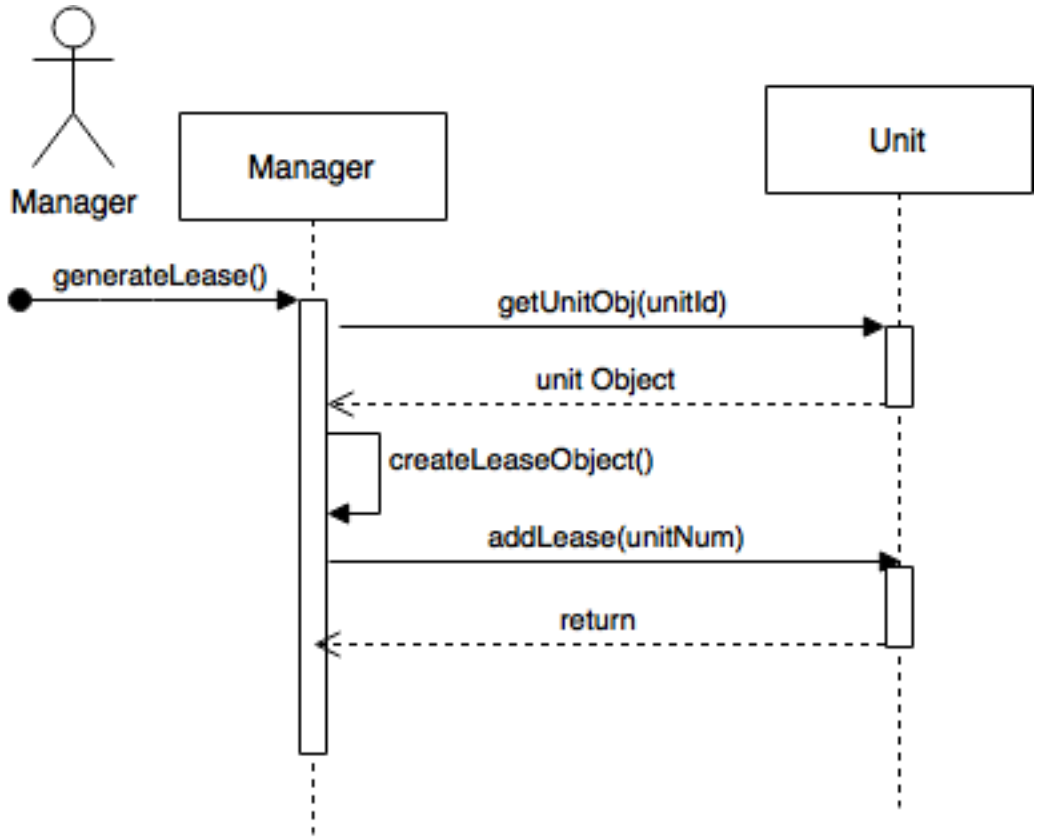


Add Tenant screen. On this screen, the manager adds a new tenant.

3.5 Enter lease details

Use Case Name: Enter Lease Details and generate lease document.	UC-5	Priority: Medium
Actor: Manager		
Description: The system provides a feature to add a new lease for an existing property unit and generate the lease document on demand		
Trigger: None Type ☑ External ☐ Temporal		
Preconditions: The Manager has logged into the system with valid credentials.		
Normal Course: 1. Manager selects "Add Lease" feature 2. Manager enters tenant information 3. Manager enters lease duration 4. Manager enters rent to be paid 5. Manager records deposit to be paid 6. System displays the message, "New lease created" 7. The System generates lease document 8. Exit from use case		Information for the step ←Tenant Info ←Lease Duration ←Rent Amount ←Deposit Amount →Display confirmation
Alternative Course 3a. Replacing a tenant in between lease duration 1. Manager updates tenant info 2. System resumes		←Tenant Info
Post-conditions: New lease document is generated		
Exceptions: E1: The lease document fails to create 1. The system displays error message "Error! <Error code>Unable to create lease document, Please Retry" 2. The system terminates the case		
Summary Inputs Source Outputs Destination		
Tenant Info Lease Duration Rent Amount Deposit Amount	Manager	Error Message Success Message	System

3.5.1 Enter Lease Details Sequence Diagram



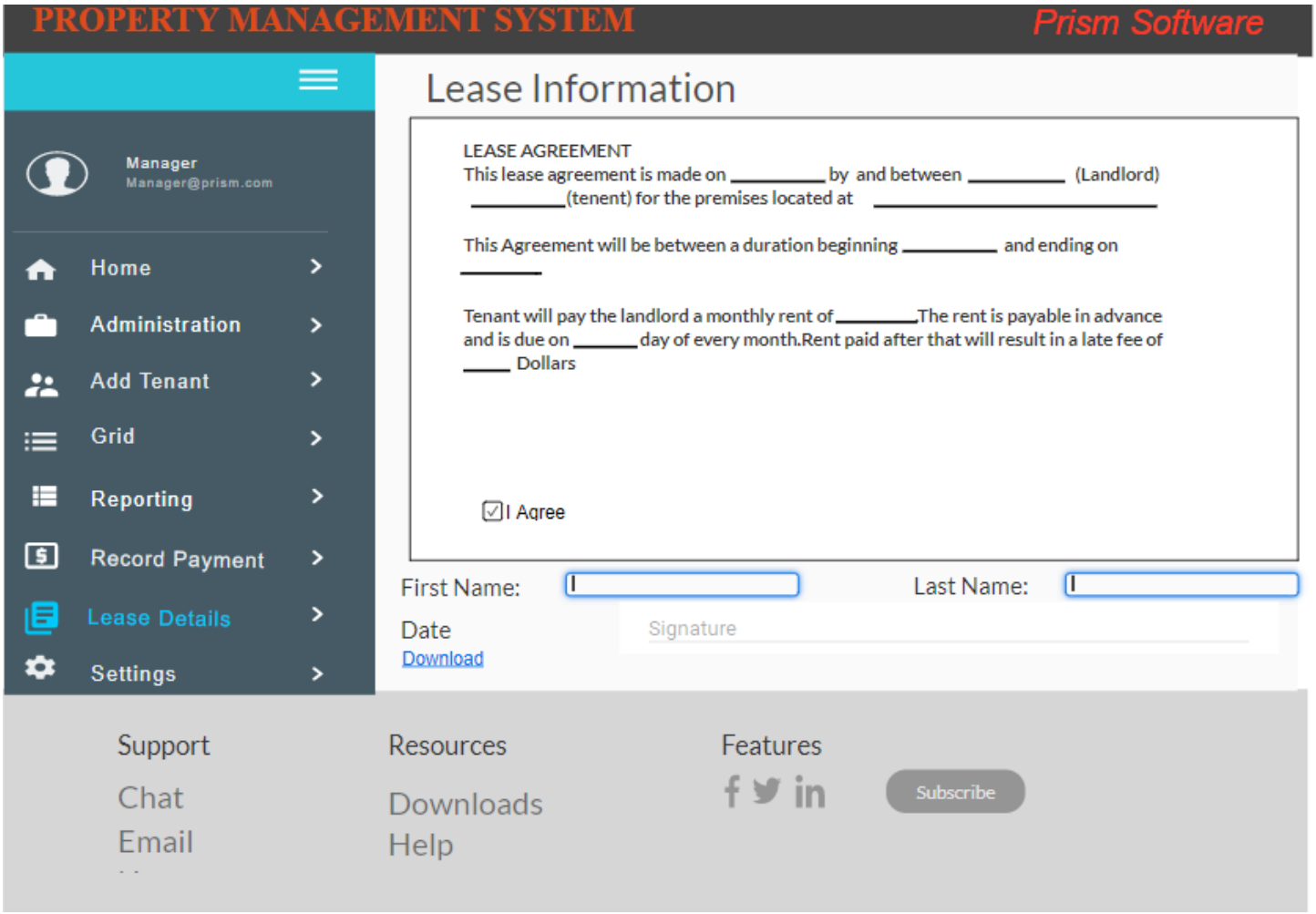
Classes Interacting: Manager and Unit

SSD:

Prerequisite – The tenant and Unit info must be present in the database

- The UI features 2 buttons – Add tenant and Add lease. Once the tenant information is added into the system, the add Lease button is clicked.
- This is going to create a Lease object with the unit number of the tenant and the lease details entered in the UI.
- The lease object is then saved into the Unit. Each unit is associated with one and only one lease.

3.5.2 Generate Lease Screen Mockup



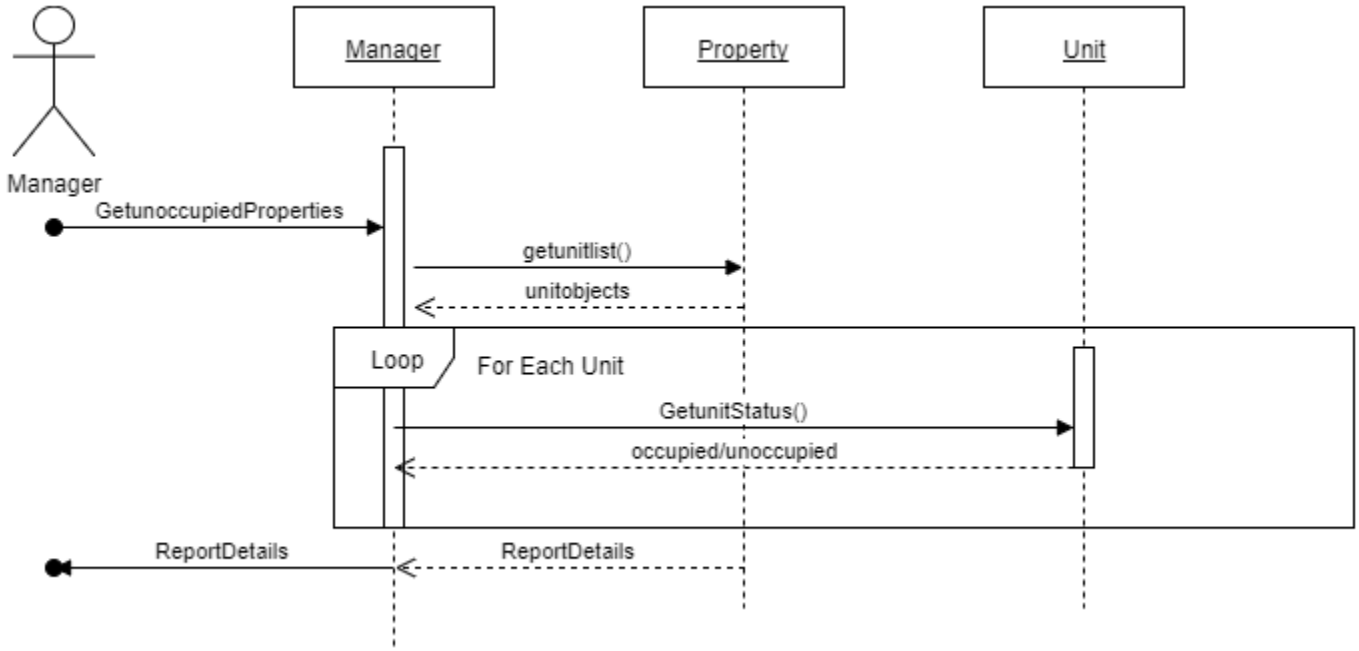
On this screen, the manager generates a lease. This screen is created from the prior screen after the user selects 'generate lease' option.

3.6 Display Unoccupied Properties

Use Case Name: Display Properties	ID: UC-6	Priority: Medium
Actor: Manager		
Description: The use case searches for and displays properties that are not being leased currently to tenants		
Trigger: Owner wants to run the system to find properties that are currently not being leased Type: ☑ External ☐ Temporal		
Preconditions: Manager has valid login credentials		
Normal Course: 1. Manager selects Administration Tab 2. Select trends tab 3. System searches for Properties marked as unoccupied 4. Percentage of unoccupied properties by week is displayed 5. Close System	Information for Steps: →Status Information	
Post-conditions: The Manager now knows what properties are available and can use the information to conduct business		

Major Inputs	Source	Major Outputs	Destination
	System	Status Information	Dashboard

3.6.1 Display Properties Sequence Diagram



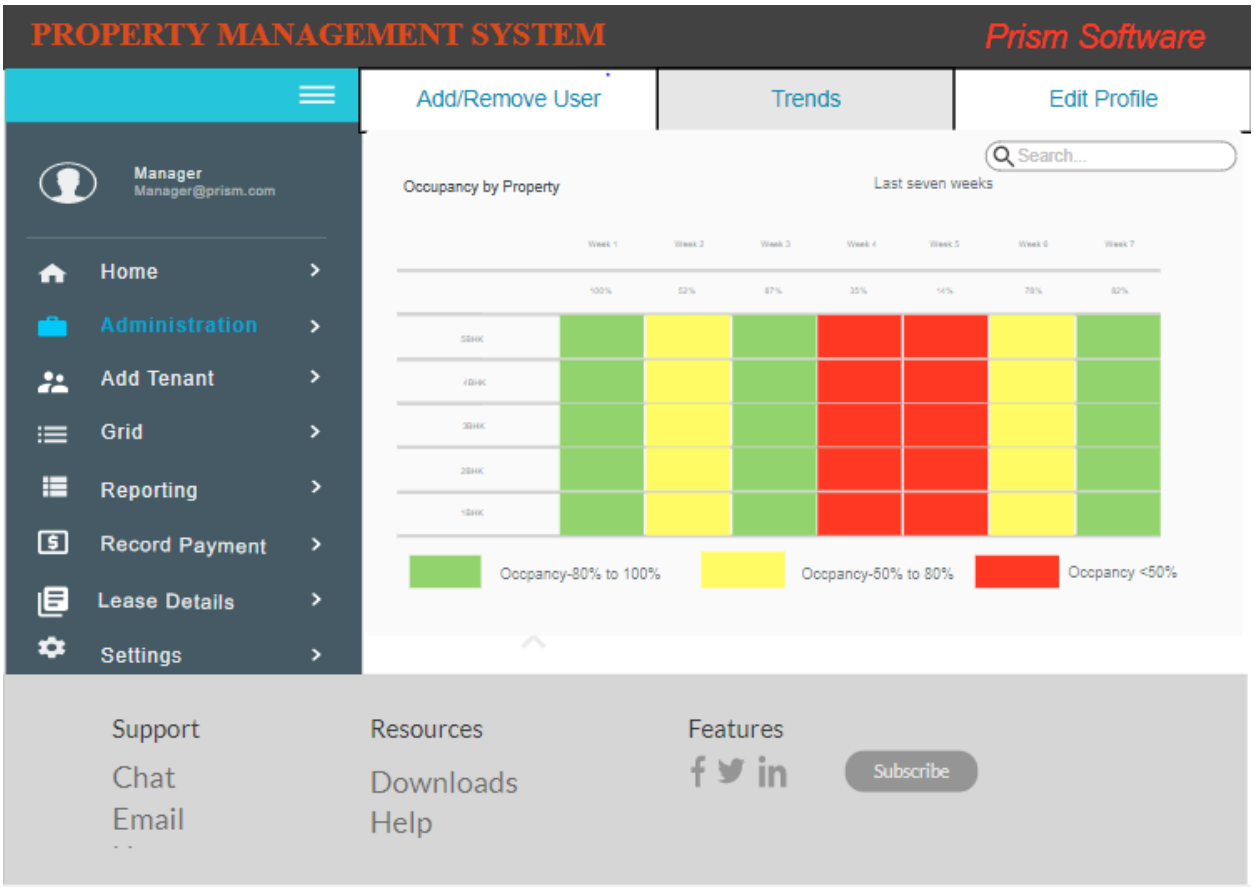
Classes Interacting: Manager, Property and Unit

SSD:

The manager logs into the system and navigates to the Dashboard where he can see the list of properties according to their availability

- Manager object gets the property list from the property object and the units associated with the property.

3.6.2 Dashboard Screen Mockup



Dashboard screen mockup. On this screen the classes interacting are: Admin, Manager, Unit

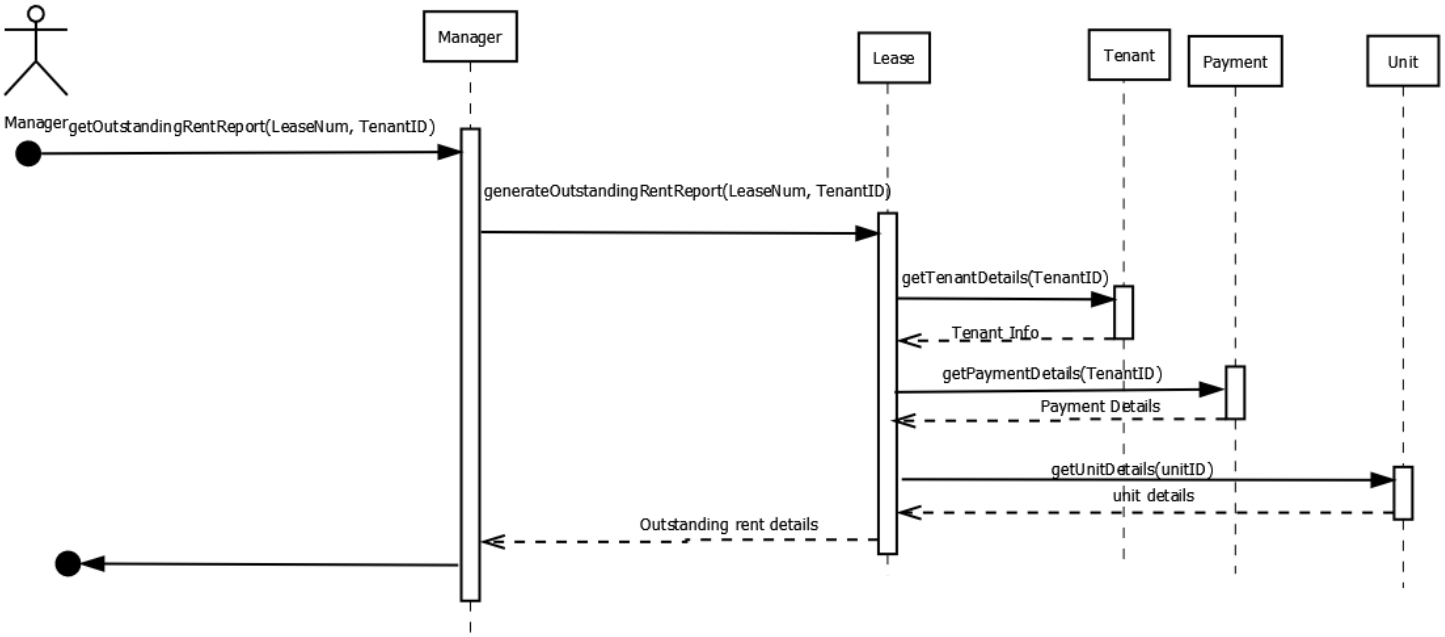
This is post login, while the Manager class decides what information they see on the dashboard. Unit is also at play for the diagram, showing occupancy rates.

3.7 Outstanding Rent Report

Use Case Name: Outstanding Rent Report	UC-7	Priority: High
Actor: Manager		
Description: The system displays a report on outstanding rent by tenants		
Trigger: None Type: ☑ External ☐ Temporal		
Preconditions: The system is up and running and data is up to date		
Normal Course: 1. System displays 'Report' Button 2. Manager clicks on 'Report' button 3. Reports page with list of reports is displayed 4. Manager selects outstanding rent report 5. Selected option is displayed on the page 6. Manager enters Lease Number and Tenant ID 7. System accepts Lease Number and Tenant ID 8. Manager clicks on run and export 9. System displays export options 10. Manager selects one of the export options and clicks on Ok 11. System run the report and exports it in desired format	**Information for Steps:** →Report options list ←Lease Number, Tenant ID ←Excel, Quickbooks →Outstanding Rent Report	
Post-conditions: The outstanding rent by tenants report should be available as per user required format.		
Exceptions: E1: Data connection error: 12. Manager clicks on run and export 13. System displays export options 14. Manager selects one of the export options and clicks on Ok 15. System displays "Data connection error. Please try again later"		

Major Inputs	Source	Major Outputs	Destination
Lease Number Tenant ID Excel Quickbooks	Manager	Report Options List Outstanding Rent Report	

3.7.1 Outstanding Rent Sequence Diagram



Classes Interacting: Manager, Lease, Tenant, Payment and Unit

SSD:

Manager selects the outstanding rent report from the report page

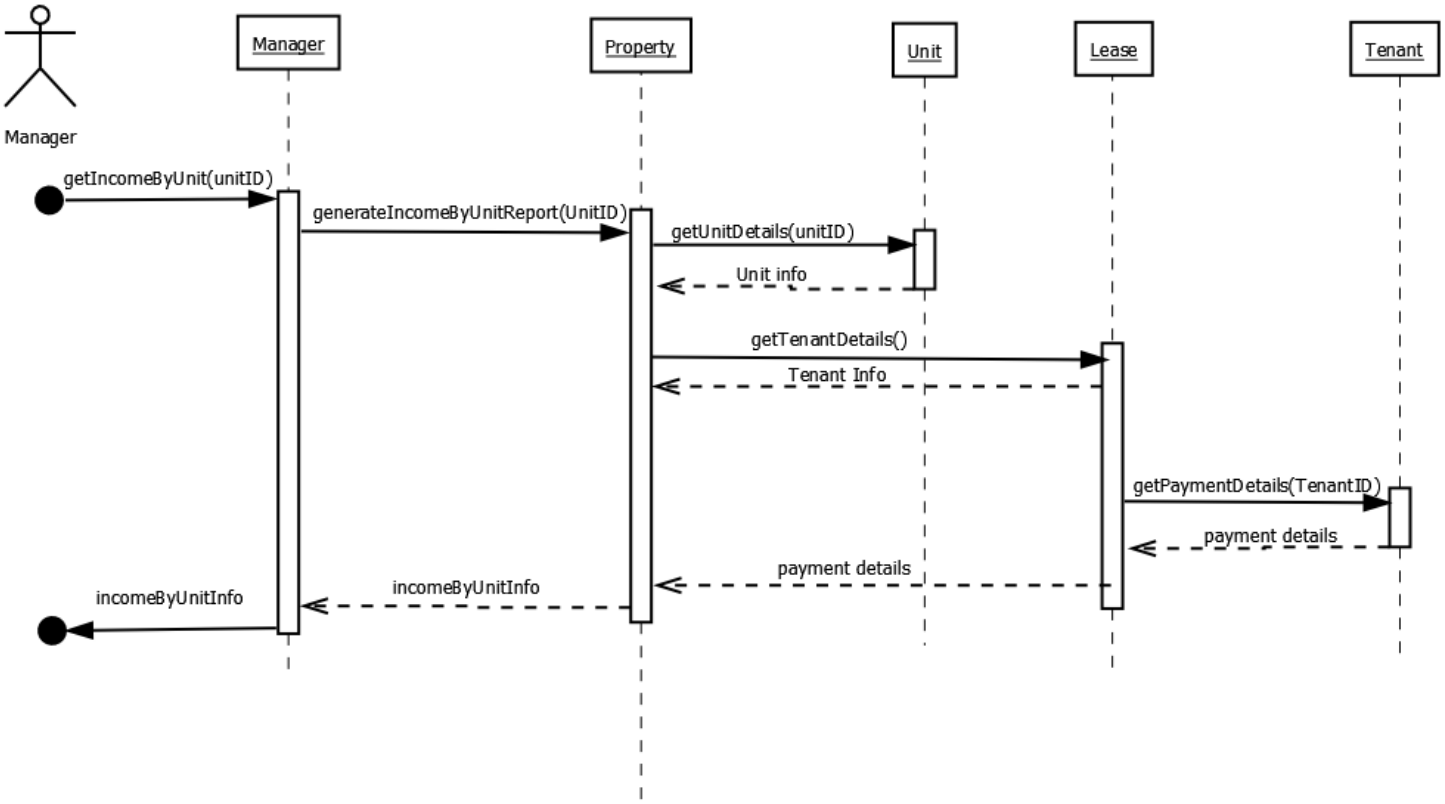
- Manager enters the lease number and the tenant id in the UI
- The function collects information from unit, tenant and payment for that lease and tenant. These details will be displayed in the UI

3.8 Total Income by Unit

Use Case Name: Total Income by Unit	UC-8	Priority: High
Actor: Manager		
Description: The system displays a report on total income by unit type for a selected time period or unit		
Trigger: None Type: ☑ External ☐ Temporal		
Preconditions: The system is up and running and data is up to date		
Normal Course: 1. System displays 'Report' Button 2. Manager clicks on 'Report' button 3. Reports page with list of reports is displayed 4. Manager selects outstanding rent report 5. Selected option is displayed on the page 6. Manager enters Lease Number and Tenant ID 7. System accepts Lease Number and Tenant ID 8. Manager clicks on run and export 9. System displays export options 10. Manager selects one of the export options and clicks on Ok 11. System run the report and exports it in desired format		**Information for Steps:** →Report options list ←Lease Number, Tenant ID ←Excel, Quickbooks →Total Income by Unit report
Post-conditions: The total income by unit report should be available as per user required format.		
Exceptions: E1: Data connection error: 12. Manager clicks on run and export 13. System displays export options 14. Manager selects one of the export options and clicks on Ok 15. System displays "Data connection error. Please try again later"		

Major Inputs	Source	Major Outputs	Destination
Lease Number Tenant ID Excel Quickbooks	Manager	Total Income By Unit Report Report Options List	

3.8.1 Income By Unit Sequence Diagram



Classes Interacting: Manager, Property, Unit, Lease, and Tenant

SSD:

Manager selects income by unit from the report page

- Manager enters the unit ID in the UI
- The function gets the following information:
    - Property in which the unit is located
    - The unit details
    - The tenant associated with the unit
    - The payments made by the tenant
- The UI displays the total income by the unit

3.9 Archive Tenant And Payment Info

Use Case Name: Archive Tenant and Payment Info	UC-9	Priority: Medium
Actor: System		
Description: The system hides the tenant and payment information while displaying the active tenant information. It remains hidden until it is explicitly deleted		
Trigger: None Type ☐ External ☑ Temporal		
Preconditions: The data for the tenant and the lease information must exist in the system		

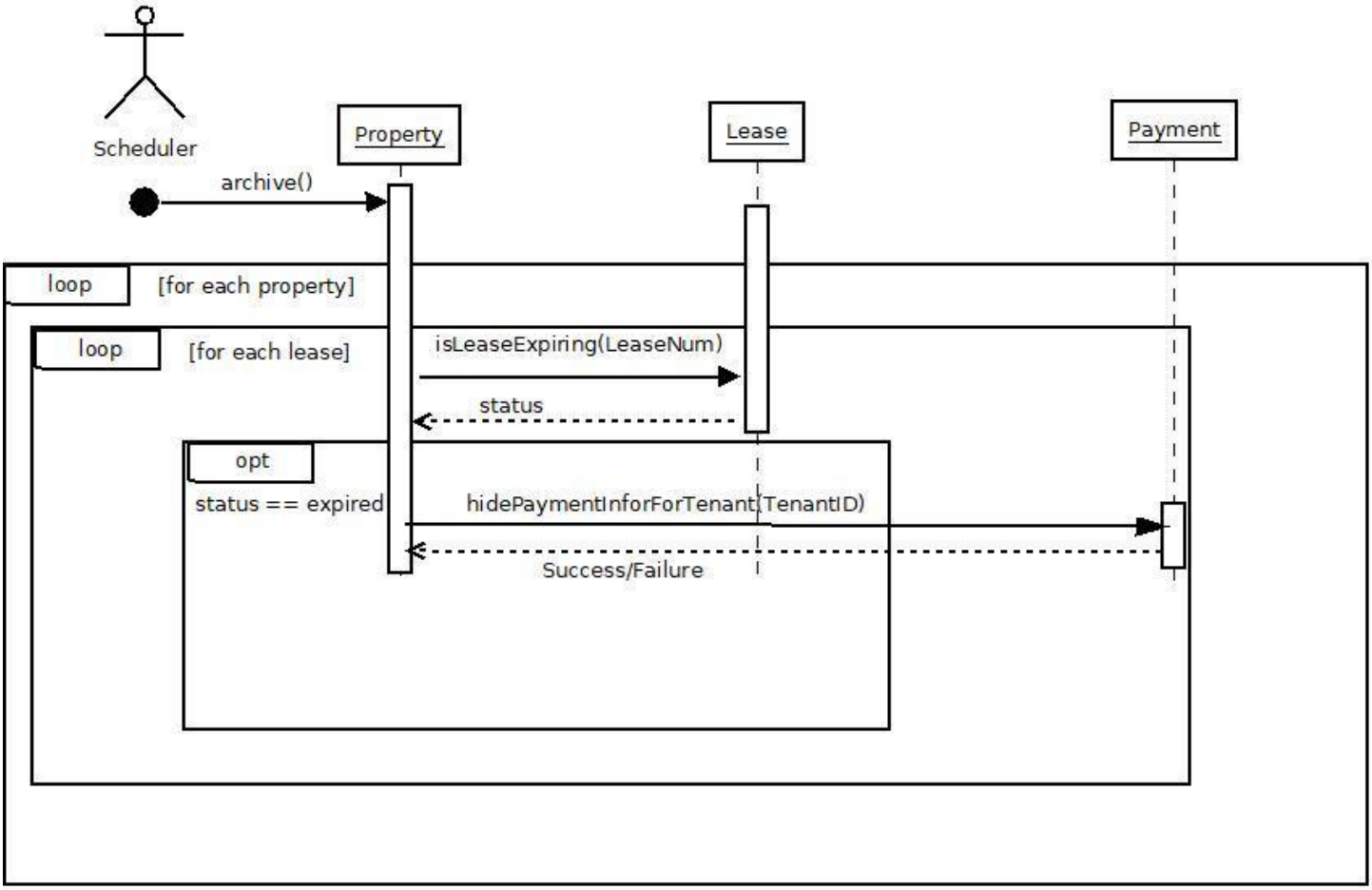
Normal Course:	
1. Lease contract expires 2. Collect the tenant information 3. Collect information of all payments made by the tenant 4. Hide the payments made by the tenant 5. Display Message "Information about Tenant (Name) hidden" 6. Exit from use case	←Lease Information ←Tenant Info ←Payment Info →Archival Data →Success Message
Alternative Course 1a. Tenant lease is not expiring Do nothing	

Post-conditions:
The current data is transformed to archive data
Exceptions: E1: The lease contract expires and the tenant information is not found (occurs at step 2) 1. The system displays message "The tenant info not found" 2. The system asks to verify and exits

Summary

Inputs	Source	Outputs	Destination
Lease info Tenant Info Payment Info	System	Archival Info Success Message Error Message	Database

3.9.1 Archive Tenant & Payment Info Sequence Diagram



Classes Interacting: Property, Lease, and Payment

SSD:

This is a backend activity.

- The scheduler class will invoke the archive() method which handles the archival of data
- This method does the following
    - collects the lease numbers from unitList (All the list of units associated with the property)
    - Checks if the lease is expired
    - If status is expired, then the payment info is hidden from the UI

3.10 Tenant Lease-up

Use Case Name: Tenant Lease-up	UC-10	Priority: Medium
Actor: System		
Description: The system provides timely reminders of tenant lease-up to enable the owners to act on		
Trigger: None Type ☐ External ☑ Temporal		
Preconditions: The lease information is present in the system		

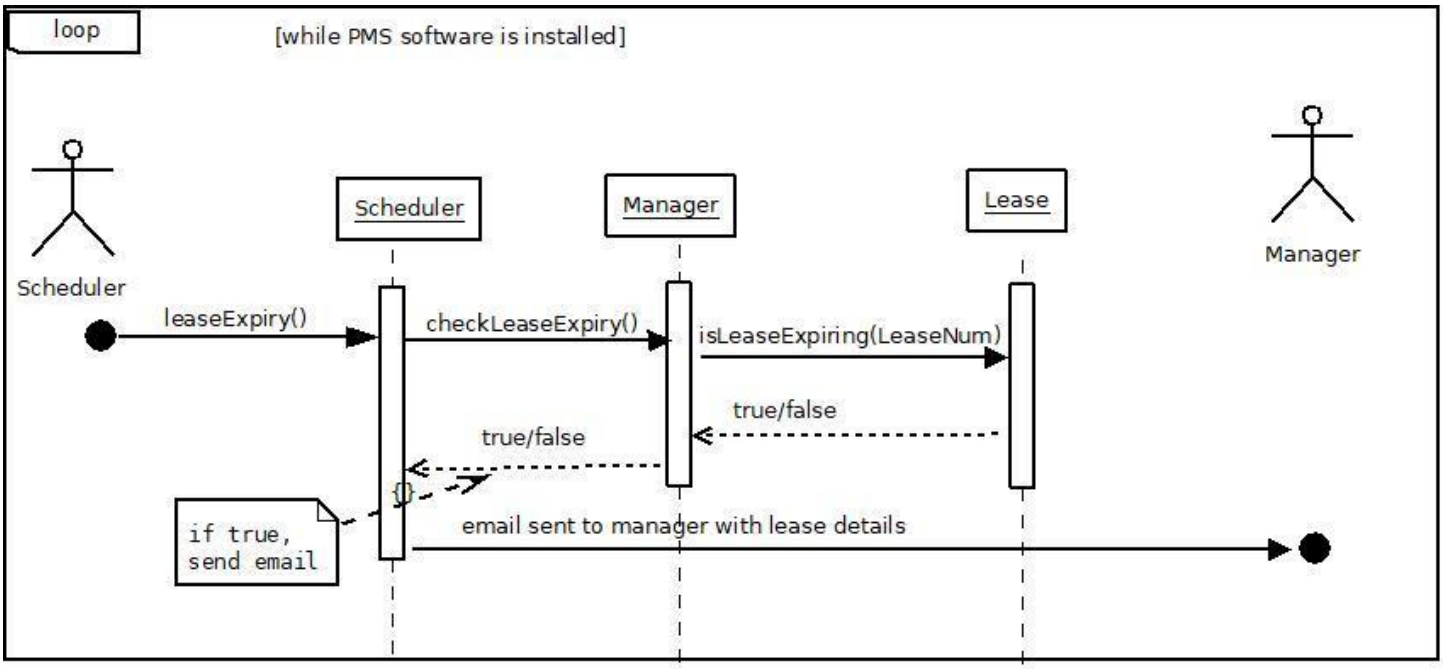
Normal Course:	
1. System periodically checks all the current leases to track for lease expiry	←Lease Information
2. If the lease end period is less than 2 months from the current date	←Current Date
3. Send an email to the email address with the tenant Info and the lease expiry details	→Email with details
4. Exit from use case	
Alternative Course 2a. Lease period is more than 2 months from the current date a. Do nothing	

Post-conditions:
An email and message with the lease expiry details is sent from the system to the owner
Exceptions: E1: The email ID is not found (occurs at step 3) a. The system displays message “The email ID is not found” b. The system prompts to enter the email ID

Summary

Inputs	Source	Outputs	Destination
Lease info Current Date Registered email ID	System	Error Message Email with details	Manager

3.10.1 Tenant Lease-Up Sequence Diagram



Classes Interacting: Scheduler, Manager, and Lease

SSD:

This is a backend activity.

- The scheduler will invoke the leaseExpiry() method which checks if the lease is about to expire and will send a notification accordingly
- This method does the following:
    - Gets the list of units and associated leases and checks if the lease expiry date is less than 2 months.
    - If true, the scheduler sends a notification

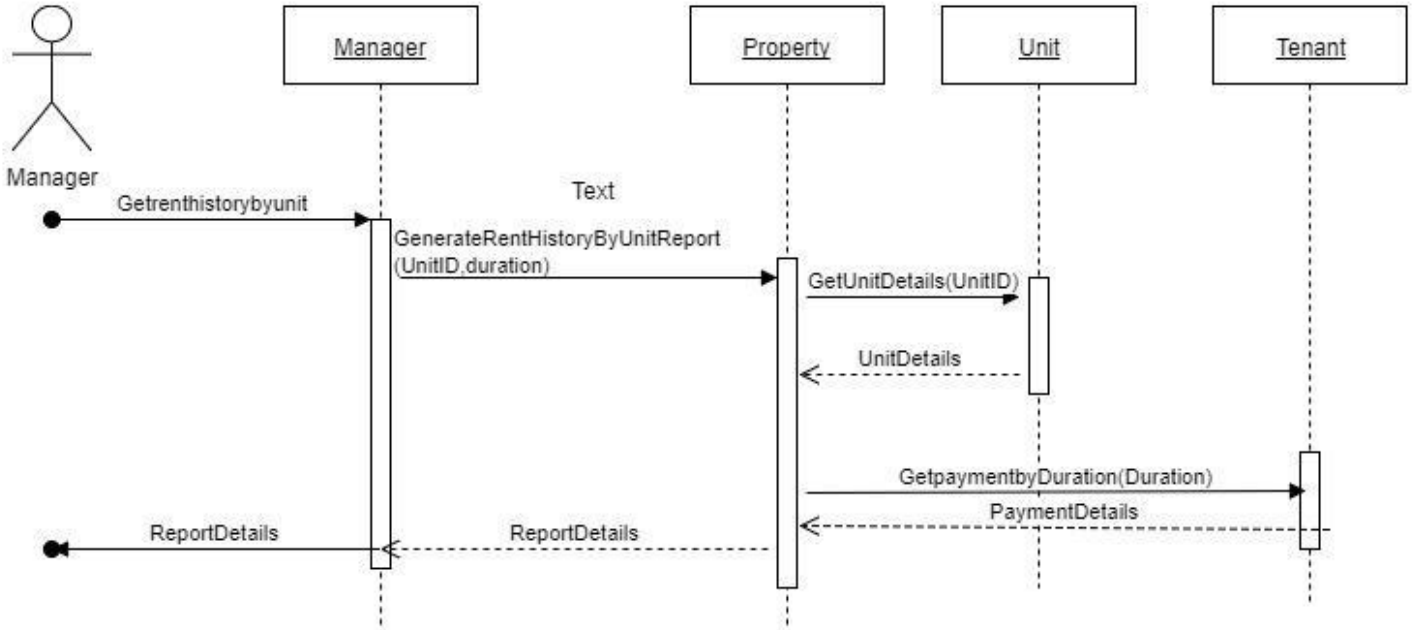
3.11 Rent History By Unit

Use Case Name: Rent History by Unit	**UC-11**	Priority: High
Primary Actor: Manager		
Description: The use case handles the rent history by unit report		
Preconditions**:** The Manager is present in the system and the data is up to date		
Trigger**:** Manager clicks the reports button Type ☑ External ☐ Temporal		

Major Steps Performed	
1.System displays the Report button 2.Manager clicks the Reports button, 3.Reports page with list of reports is displayed 4.Manager selects the Rent history by Unit report 5. The reports page is displayed. 6.Run button is clicked in the report 7.Option to select unit number is displayed 8.Duration of the report is chosen 9. The report is run and the results are displayed. 10.Export button is clicked 11.Report format is chosen and then the report is exported	←Unit Number ←Duration ←Excel, Quickbooks →Rent History by Unit report
Alternative Flow 1. Manager Selects the report and an incorrect unit is chosen a. System displays incorrect unit error	

Post Conditions: The report is exported to the user in the required format			
Inputs Unit Number Duration Excel Quickbooks	Source Manager Input	Outputs Rent History By Unit Report	Destination

3.11.1 Rent History By Unit Sequence Diagram

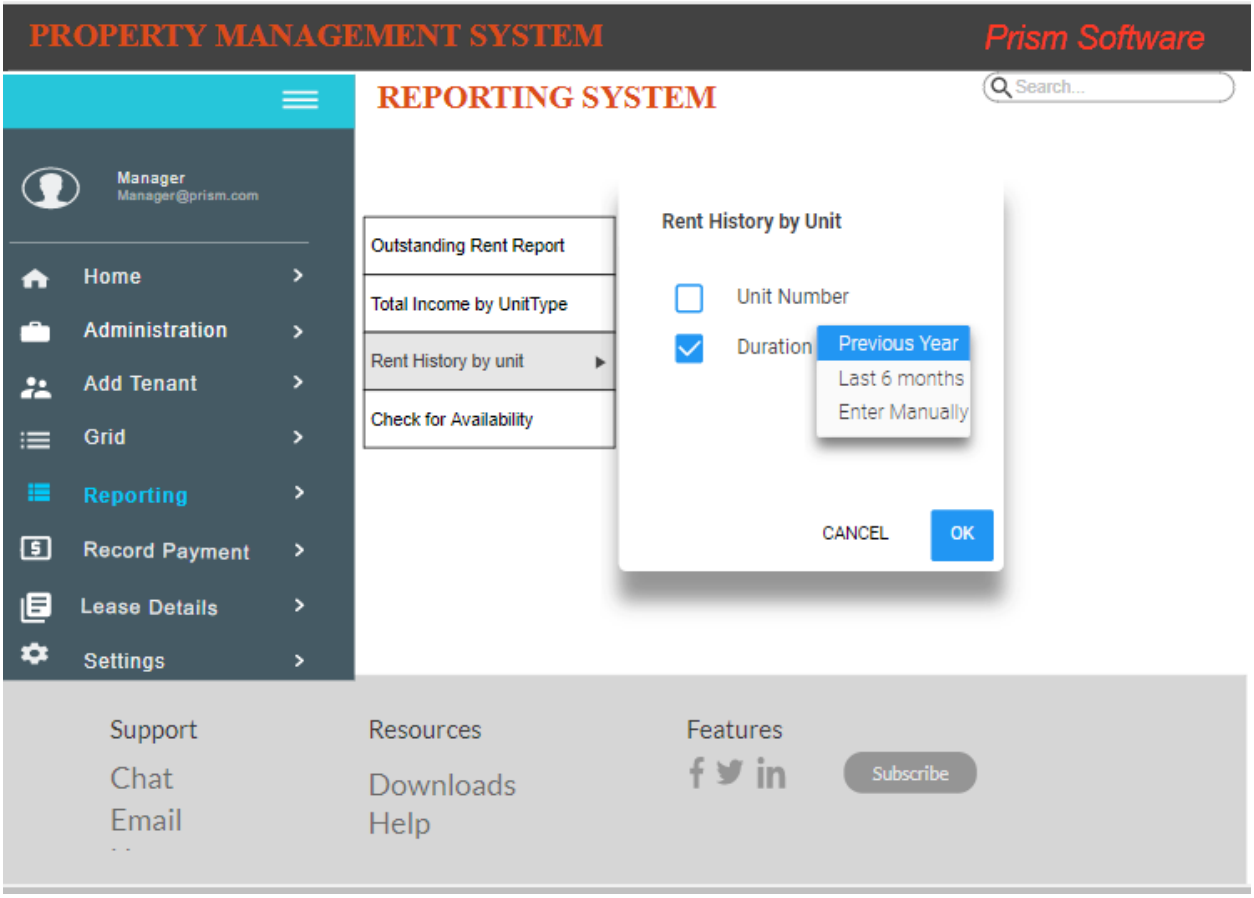


Classes Interacting: Manager, Property, Unit and Tenant

SSD:

- Manager clicks the Getrenthistorybyunit report and that results in Getrenthistorybyunit method being invoked.
- The generaterentHistoryByUnitReport method in the property class is called from which we get the unitdetails(as unit is contained in property) and payment details from tenant(payment contains tenant)

3.11.2 Report Screen Mockup



Report screen. On this screen the manager can run several pre-made reports in the system. The classes interacting on this screen are: Report, OutstandingRent, IncomeByUnit, RentHistoryByUnit, and Unit. The report class is what begins this process. And from here a user can choose to run the reports.

3.11.3 Rent History Screen Mockup

Rent History screen. On this screen, the manager can see archived information about prior rent activity per unit.

3.12 Record and Track Payments

Use Case Name: Record and track Payments	UC-12	Priority: High
Actor: Manager		
Description: The system provides a feature for capturing the data pertaining to payments that are needed to be recorded		
Trigger: Manager initiates the process of recording a payment Type ☑ External ☐ Temporal		
Preconditions: Manager must receive a check from tenant or HAP to proceed for recoding payment		

Normal Course:	
1. Manager confirms tenant's identity and looks up the system for the latest profile and rent due from tenant.	
2. Manager looks up the latest subsidiary records of the tenant.	←Lease Number
3. Manager updates tenant's rent amount if applicable.	←Amount due from the tenant
4. Manager records the check details in the system and updates tenant's profile with latest payment.	←Check information
5a. Manager receives a check from HAP.	←Check details from HAP
5b. Manager updates respective tenant profiles and goes to step 3	
6. Manager receives payment status	→Payment Status
Alternative Course:	
1. Manager doesn't receive any payment from tenant by deadline. 2. Manager checks tenant record.	
3. Manager enters comments manually in the tenant profile.	←Updated tenant profile information

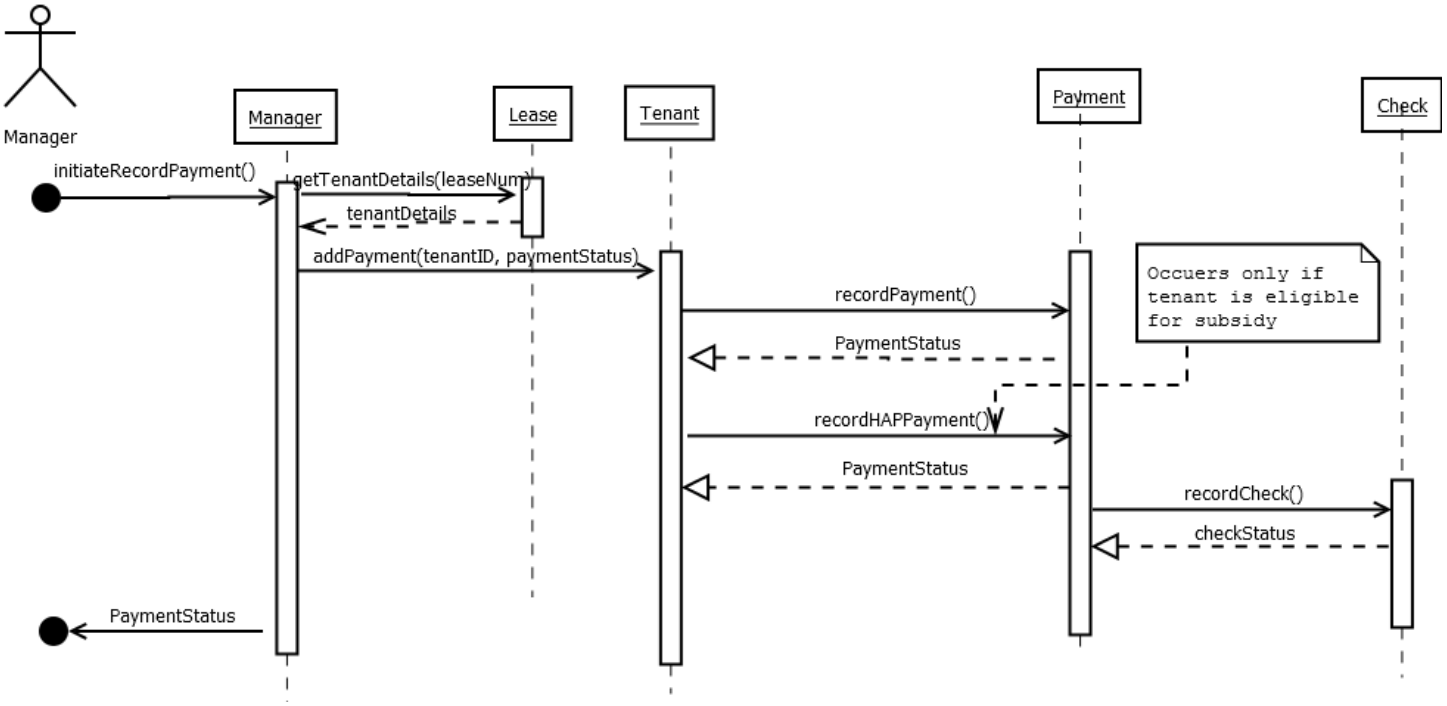
Post-conditions:
The payment has been recorded and the tenant profile is updated with latest payment information.

Exceptions:
E1: The amount on check from Manager doesn't match with the due amount in the system (Occurs at step 1)
1. The Manager checks for updates from HAP subsidy for the tenant.
2. The Manager informs the tenant about updated amount and asks for a different check.

Summary

Inputs	Source	Outputs	Destination
Amount due from Tenant Tenant check information HAP check information	System Manager	Payment Status	Manager

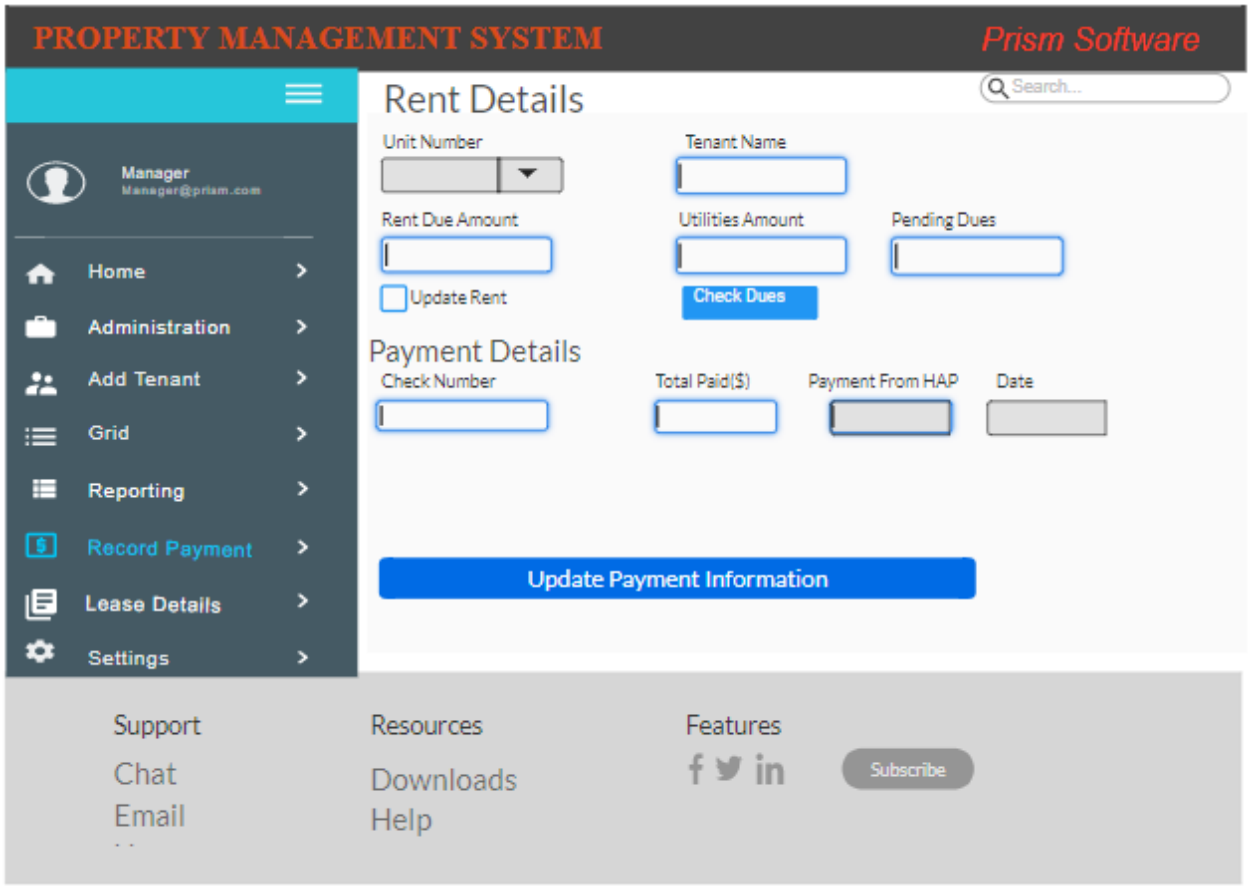
3.12.1 Record And Track Payments Sequence Diagram



Classes Interacting: Manager, Lease, Tenant, Payment and Check

SSD:
The manager clicks the record payment button.
The initiateRecordPayment() method is called. The tenant details are obtained and thereafter payment object is created. After creation of payment object, payment is recorded.

3.12.2 Record And Track Payments Screen Mockup



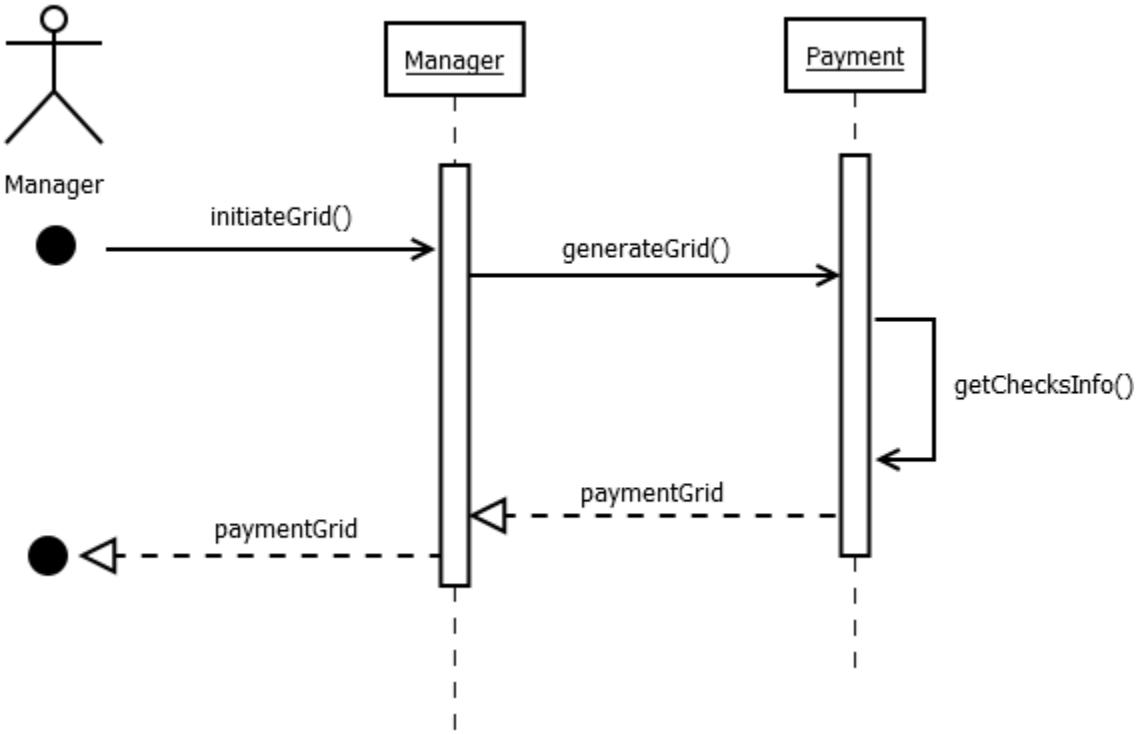
The payment screen mockup shows the payment screen in which the payment details are recorded.

3.13 Generate Batch Grid

Use Case Name: Generate batch grid	UC-13	Priority: High
Actor: Manager		
Description: The Manager takes all received checks as input and generates a batch grid of checks to be deposited in the bank		
Trigger: System initiates the process of creating a batch grid Type ☑ External ☐ Temporal		
Preconditions: At least one check must be recorded in the system by the deadline		
Normal Course: 1. After payment deadline, Manager checks for information about every check received. 2. Manager generates a payment batch grid containing all pre-specified columns.	←Check Information →Batch Grid Document	
Post-conditions: The payment batch grid id generated.		
Exceptions: E1: System doesn't find any check record. (Occurs at step 1) 1. System displays an error message saying, "No payment records found" 2. System terminates the use case.		
Summary		

Inputs	Source	Outputs	Destination
Check Information	Manager	Payment Batch grid	Manager

3.13.1 Generate Batch Sequence Diagram

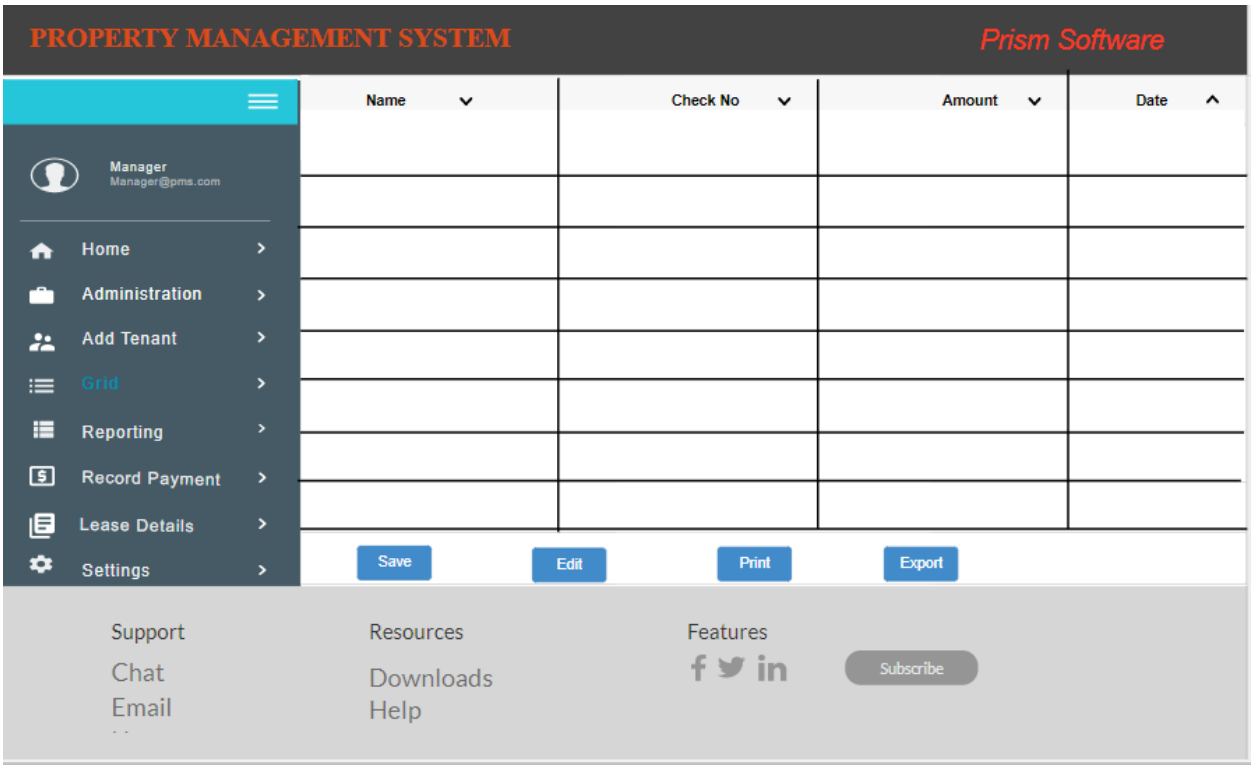


Classes Interacting: Manager and Payment

SSD:

The manager invokes the generateGrid method in payment class which contains check objects. This thereafter generates the check grid.

3.13.2 Check Payment Screen Mockup



Check Payment Screen. On this screen, the manager batch enters payments into the system. Once the details are entered, the manager could use the print option to get the batch of checks to be deposited in the bank.

4.0 Test Plans, Scope, and Levels of Testing

4.1 Test Classification

Defects found during the testing phase will be classified into the following categories

Priority	Description
1 – Critical	The bug is critical enough to crash the system or cause data loss
2 – High	The bug impacts vital functionality
3 – Medium	The bug will degrade the quality of the system
4 – Low	The bug will cause minimum impact
5 – Intermittent/Cosmetic	The bug is intermittent or has no impact on the system usage

4.2 Scope of Testing

4.2.1 Functional Requirements:

1. Add Property Management Group

Description: The system provides a feature to add a new Property Management Group

Test Strategy: For this requirement, the Manager, Property and Unit classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Black box integration testing will also be done each iteration. Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration.

2. Add Property

Description: The system must allow the manager to add Property Management Group to each property managed by him/her.

Test Strategy: For this requirement, the Manager, Property and Unit classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Black box integration testing will also be done each iteration. Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration. We have mentioned detailed test cases for this feature below (page 53).

3. Add Property Unit

Description: The system must allow the manager to add Property unit to a Property Management Group managed by him/her.

Test Strategy: For this requirement, the Manager, Property and Unit classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Black box integration testing will also be done each iteration. Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration. We have mentioned detailed test cases for this feature below (page 54).

4. Add Tenant

Description: The system must allow the managers of the properties to add tenants to the units. The system must now allow access to properties managed by other managers or property groups

Test Strategy: For this requirement, the Manager and Lease classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Blackbox integration testing will also be done each iteration. We have mentioned detailed test cases for this feature below (page 50). Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration.

5. Enter Lease Details

Description: The system must allow managers to enter lease details and generate lease when a new tenant is added, or the old lease has expired

Test Strategy: For this requirement, the Manager and Unit classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Black box integration testing will also be done each iteration. Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration.

6. Display Unoccupied Properties

Description: The system must provide manager the ability to generate the occupancy chart and display properties available for rent to new tenants.

Test Strategy: For this requirement, the Manager, Property and Unit classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Black box integration testing will also be done each iteration. Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration.

7. Outstanding Rent Report

Description: The system must be able to generate outstanding rent report for all the properties managed by the manager.

Test Strategy: For this requirement, the Manager, Lease, Tenant, Payment and Unit classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Black box integration testing will also be done each iteration. Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration.

8. Total Income by Unit

Description: The system must be able to generate Total Income by unit report for the units desired by the manager managed by the manager.

Test Strategy: For this requirement, the Manager, Property, Unit, Lease and Tenant classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Black box integration testing will also be done each iteration. Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration.

9. Archive rent and payment info:

Description: The system should archive rent and payment information automatically every 6 months by storing the information in the database.

Test Strategy: For this requirement, Property, Lease, and Payment classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Black box integration testing will also be done each iteration. Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration.

10. Tenant lease-up:

Description: The system should be able to generate lease for a tenant and a unit which is to be valid for specific period of time. This lease engagement is to be permanently recorded in the database and rent payment should be made regularly against it.

Test Strategy: For this requirement, the Scheduler, Manager and Lease classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Black box integration testing will also be done each iteration. Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration.

11. Rent history by Unit

Description: Rent history by Unit is a report that can be run by the manager/user to see and analyze rent revenue by Unit. The data for this report is fetched from the database and presented to the user in the form of excel report.

Test Strategy: For this requirement, the Manager, Property, Unit and Tenant classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Black box integration testing will also be done each iteration. Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration.

12. Record and track payment

Description: The payment made by the payer should be recorded and tracked by the system. The system should track details like check no., payer details, payment source, etc. for each check and store it in the database.

Test Strategy: For this requirement, the Manager, Lease, Tenant, Payment and Check classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Black box integration testing will also be done each iteration. Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration. We have mentioned detailed test cases for this feature below (page 52).

13. Generate batch grid

Description: The system takes all the received checks as input and generates a batch grid of checks to be deposited to the bank.

Test Strategy: For this requirement, the Manager and Payment classes will need to be tested. Each will be black box tested as each change is committed to Git. White box code review will be completed during each iteration. Black box integration testing will also be done each iteration. Automation test scripts will be first written in HP Unified Functional Test when the functionality is first developed. These scripts will be run during integration testing and during every iteration.

4.2.2 Non-functional requirements:

14. Cross browser compatibility

Description: User must be able to access the system over multiple browsers.

Test Strategy: This requirement will be tested during usability testing and system testing where the application is tested on multiple browsers like Firefox, Chrome, Internet Explorer

15. Ability to render on multiple resolution devices

Description: User should be able to access the application over mobile, laptop and tablet devices.

Test Strategy: This requirement will be tested during usability testing and system testing where the application is tested on various electronic devices like mobile, laptop, desktop and tablet of different aspect ratios.

16. Ability to handle 10000 users

Description: The system should support about 1000 property management groups with 10,000 units each

Test Strategy: This requirement will be tested during the load and stress testing using jmeter tool.

17. Pages must load within 4 seconds

Description: All the pages and reports should load within 4 seconds

Test Strategy: This will be tested with Jmeter tool by running test scripts to check for performance for 1000 Property Management Groups with 10000 units each and recording the average time for each of the page load and/or report request. All the reports and pages must load within 4 seconds.

18. AES encryption standard for passwords

Description: The password stored in the database should be stored using AES encryption. This will be tested by logging in into the Database and looking at the password column. The password should be encrypted using AES algorithm and should be unreadable.

Test Strategy: All the passwords should be encrypted before we store in the database.

19. **Usability requirement**:

Description: Tooltips for textboxes, help button, contact support feature in each page

Test Strategy: All the links should have a tooltip. Help should be available to guide the end user in using the application features. Contact support feature should be available to enable the end user to contact support.

This project follows these testing techniques:

	Purpose	**Tester and technique**	**Timing**
Unit Testing	To determine that each module of the system is working as expected	Developer Blackbox +whitebox	After each module is completed
Integration Testing	To test that the individual components that are developed are working as expected when integrated together.	Testing team with the use of Test Scripts	In each iteration
System Testing	To test if the integrated system is compliant with specified requirements.	Testing team with the use of Test Scripts for smoke, sanity, and regression	In each iteration

Acceptance Testing	To evaluate the systems compliance with business requirements and is tested for its acceptability.	Business users	Meets the exit criteria in exit criteria table below
Load Testing	To test if the system can handle concurrent users up to 10,000 users	Testing team with the use of test scripts like JMeter	End of last iteration
Stress Testing	To tests and find the maximum number of users the system can handle until the performance deteriorates below a specific level	Testing team with the use of test scripts like JMeter	End of last iteration
Performance Testing	To tests and find the maximum number of users the system can handle until the performance deteriorates below a specific level	Testing team with the use of test scripts like JMeter	End of last iteration
Security Testing	To tests application for robustness and reliability	Testing team tests for encryption, security measures, etc.	End of last iteration

Testing Criteria:

Exit Criteria for Testing	Status
100% Test Scripts executed	
90% pass rate of Test Scripts	
All critical and high severity bugs are addressed. Low severity bugs < 30%	
All defects are documented in ALM	
Remaining bugs are re-prioritized and targeted to future releases	
Test completion document signed off the testing team	
All test results uploaded into the SharePoint and stored for future references	

4.3 Testing Tools

Automation Test tool: HP UFT

Defect Tracking Tool: ALM

Version Control: Git

4.3 Test Cases

This is the test case as mentioned above in functional requirement 6 above.

4.4.1 Add Tenant

Module: addTenant() **Version Number: 1**

Tester: -------- **Date Designed: 11/20/17** **Date Conducted: ------**

Results: ------ **Open Items ---------**

Test ID: 1

Requirement: Verify the setTenant() of Lease Class

Objective: Ensure that the setTenant() method is working correctly. Each field is validated independently. This method

Test cases for setTenant():

#	Unit Type	First Name	Last Name	Home/ Office Phone No.	Billing Address	Income	Deposit Paid	Expected output
1	NULL							Error- Invalid input
2	0							Error- Invalid input
3	A							Pass
4		!12ac						Error- Invalid input
5		My09						Error- Invalid input
6		NULL						Error- Invalid input
7		Marvin						Pass
8			!12ac					Error- Invalid input
9			My09					Error- Invalid input
10			NULL					Error- Invalid input
11			Gayn					Pass
12				0000000000				Error- Invalid input
13				Accdddeeff				Error- Invalid input
14				9898887656				Pass
15				NULL				Error- Invalid input
16					3423 Drive			Pass
17					#@#$			Error- Invalid input

18					NULL			Error- Invalid input
19						0		Pass
19						340		Pass
21						NULL		Error- Invalid input
22						ac%		Error- Invalid input
23						acbd		Error- Invalid input
24							340	Pass
25							0	Pass
26							NULL	Error- Invalid input
27							ac%	Error- Invalid input
28							acbd	Error- Invalid input

Expected Results:

All the valid test cases must pass.

Actual Results/Notes:

This is the test case as mentioned above in functional requirement 12 above.

4.4.2 Record Payment

Module: Manager_recordPayment **Version Number: 1**

Tester: -------- **Date Designed: 11/16/17** **Date Conducted: ------**

Results: ------ **Open Items ---------**

Test ID: 1

Requirement: Verify the recordPayment() of Payment Class

Objective: Ensure that the recordPayment() method is working correctly. Each field is validated independently

Test cases for recordPayment

#	Amount	Check#	Unit#	Date Received	Expected Result
1	0				Error - Invalid input. Numbers only
2	9999999				Error Message – Verify amount
3	ABCD				Error - Invalid input. Numbers only
4	1*+231				Error - Invalid input. Numbers only
5	2000				Pass. Proceed to next validation
6	3456.5				Pass. Proceed to next validation
7		12345678			Pass. Proceed to next validation
8		ABCD			Error – Invalid input. Numbers only
9		123ABC			Error – Invalid input. Numbers only
10		1%^&			Error – Invalid input. Numbers only
11			12		Pass. Proceed to next validation
12			4A		Pass. Proceed to next validation
13			AA		Pass. Proceed to next validation
14			3$#@		Error – Invalid input
15				12/12/12	Pass. Proceed to next validation
16				12/13/12	Pass. Proceed to next validation
17				13/12/12	Error – Invalid date
18				00/00/00	Error – Invalid date

Expected Results:

All the valid test cases must pass.

Actual Results/Notes:

This is the test case as mentioned above in functional requirement 2 above.

4.4.3 Set Availability Status

Module: Unit_setAvailabilityStatus **Version Number:** 1

Tester: -------- **Date Designed:** 11/16/17 **Date Conducted:** ------

Results: ------ **Open Items** ---------

Test ID: 1

Requirement: Verify the setAvailabilityStatus() of Unit Class

Objective: Ensure that the setAvailabilityStatus()method is working correctly. Each field is validated independently

Test cases for setAvailabilityStatus()

#	isAvailable	Property Name	Unit#	Expected Result
1	9999999			Error – Invalid input. isAvailable can be only true or false
2	ABCD			Error – Invalid input. isAvailable can be only true or false
3	1*+231			Error – Invalid input. isAvailable can be only true or false
4	true			Pass. Proceed to next validation
5	false			Pass. Proceed to next validation
6		12345678		Pass. Proceed to next validation.
7		ABCD		Pass. Proceed to next validation
8		123ABC		Pass. Proceed to next validation
9		1%^&		Pass. Proceed to next validation
10			12	Pass. Proceed to next validation
11			4A	Pass. Proceed to next validation
13			AA	Pass. Proceed to next validation
14			3$#@	Error – Invalid unit number

Expected Results:

All the valid test cases must pass.

Actual Results/Notes:

This is the test case as mentioned above in functional requirement 3 above.

4.4.4 Set Unit

Module: Manager_setUnit **Version Number:** 1

Tester: -------- **Date Designed:** 11/16/17 **Date Conducted:** ------

Results: ------ **Open Items** ---------

Test ID: 1

Requirement: Verify the setUnit() of Payment Class

Objective: Ensure that the setUnit() method is working correctly. Each field is validated independently. This is part of SetUnit functionality of Unit class

Test cases for setUnit():

#	UnitID	unitType	unitSize	unitPlan	baseRent	Expected Result
1	123					Pass
2	NULL					Error- Invalid Input
3	-					Error- Invalid Input
4		ABC				Pass
5		12				Pass
6		@				Error- Invalid Input
7		NULL				Error- Invalid Input
8			1200			Pass
9			Abc123			Error- Invalid Input
10			Medium			Error- Invalid Input
11			NULL			Error- Invalid Input
12				A		Pass
13				1		Pass
14				NULL		Error- Invalid Input
15					800	Pass
16					Abc	Error- Invalid Input

Expected Results:

All the valid test cases must pass.

Actual Results/Notes: