ISTM 624

ADVANCED SYSTEM ANALYSIS AND DESIGN

FALL 2017

SUBMITTED BY

TEAM - ALPHA OMEGA

TANUSHREE KOTHARI

NUPUR SINHA

PREKSHA BEOHAR

TSAI-MEI CHEN

VIJAY CHACKO

CONTENT

1. Business Case

1.1 Business Objective

Tracking rents may not seem tough at first. But add in multiple rent payment options such as HAP, cash, check and credit card it tends to get a bit complicated to record manually. Now go further and imagine you have to keep track of 1000 different properties each having multiple units that are leased out. The sheer volume of cash flow given the different channels it can come from makes it extremely difficult to track manually. This is clearly the issue our product tends to address. Our objective is to build a robust cloud based distributed product that multiple real estate agencies can buy as a service. We intend to enable the user to have effective control by keeping track of rental income along with providing great insights into the cash flows occurring throughout. Aggregate value data in form of multiple reports is also a feature that provides the user with a clear perspective and understanding of rent transactions each month and over a period of time. We intend to distinguish ourselves from the other competitors by ensuring simplicity and clarity of operations. With our user experience rich features in every screen we ensure minimum effort in accomplishing tasks. Hence our overall objective is to make tracking rental payments a piece of cake.

1.2 Scope and Impact

The product will be used by several individual landlords and corporate real estate agents to track rental income from one or more properties (properties consists of multiple units) they own or manage. The customer for the information system can either be the individual owner of the property or the employees of the real estate management company. Our application doesn't distinguish between an individual owner and a real estate employee as they both are referred as the "Customer". The tenants are the users added by the customer for a specific unit they have leased out. Tenants have limited functionalities related to rent payment of their unit. Rental amounts received can be recorded in the system against each unit and tenant. The system can also be used to manage multiple units to figure out which are vacant, and which have rent outstanding for the month. Reports can be generated to get aggregate data such as rent per unit, rent per property, rent outstanding per tenant and event data such as average pay date of tenants. The system will be able to accept online payments from tenants. Tenants will have their own portal where they can see the history of their payments, upcoming rent due dates, outstanding amounts and also a message board where the realtor can post specific messages related to their property or unit. Payment disputes can be lodged through the system by either the realtors or the tenants, but payment resolution will not be handled by the system.

The primary impact of the system will be to make rental payments clear for both the beneficiary and the payer. Lot of querying power can be given to the user to get specific information in form of reports. The system can track rent from multiple sources (HAP, Self) and user can record payments of several types (Cash, Check, credit card).

1.3 Benefits and Value added

The Product will add the following benefits:

1.3.1 **Process Efficiency in Rental Tracking:** There are several processes involved in rental payment operations such as payment management, fulfillment, invoicing (receipt generation) and financial consolidation. These are made easy to do through our product

1.3.2 **Dramatically Improved Visibility:** Real – time visibility is highly important in making decisions and to have a better perspective on the processes involved. Our application intends to provide clarity in the rent processing operations to all parties involved.

1.3.3 **Significant Time and Cost Savings:** Our product makes the rent processing operation as simple as possible shortening the process time by 34 % compared to all other products currently in the market. Also, it is an all-encompassing software that comes with support and cloud hosting avoiding the procurement, installation and maintenance of any additional IT infrastructure.

2. Requirements Definition

2.1 Functional Requirements

2.1.1 Property and units

2.1.1.1 The system will allow the customer to add new properties to the database.
2.1.1.2 The system will allow the customer to add new units to the database.
2.1.1.3 The system will allow the customer to make changes to unit or property information stored in the database.

2.1.2 Tenants and Lease

2.1.2.1 The system will allow the customer to enter details about the tenant- name, phone number (up to 3) and deposit amount paid.
2.1.2.2 The system will allow the customer to make changes to the lease.
2.1.2.3 The system will notify the customer to refund deposit paid by a tenant when they leave.
2.1.2.4 The system allows the tenant to view their rent payment history and make payments.

2.1.3 Rent Tracking Management

2.1.3.1 The system should display a form where the customer enters the amount of each check from a tenant.
2.1.3.2 The system will allow the customer to record checks from tenants in a batch mode at various times over the course of a month.
2.1.3.3 The system will allow the customer to run the batch process multiple times per month, as checks arrive.
2.1.3.4 The system will record the last amount received via subsidy, and will allow the customer to change that during any monthly keying of subsidy payments.
2.1.3.5 The system will provide for multiple sources of payments for rent:
  2.1.3.5.1 The system will allow the customer to record checks from tenants.
  2.1.3.5.2 The system will allow the customer to record payments from the federal government Housing Assistance Program (HAP). The system will then cycle through the tenants that have subsidy payments due and will pre-populate subsidy amount fields where possible.
  2.1.3.5.3 The system will allow the customer to record payments from tenant's parents.
  2.1.3.5.4 The system will allow the customer to record payments in parts from tenants.
2.1.3.6 The system will display a list of all the tenants with the actual rent amount received (adding all sources for payment) for a particular month and the expected rent to be paid by the tenant for the month.

2.1.4 Unpaid Rent Management

2.1.4.1 The system will display outstanding amounts due on the first of the month, as well as overdue rent from a previous month.
2.1.4.2 The system will allow customer to write off some amount of unpaid rent for a tenant if their most recent payments are complete and timely.
2.1.4.3 The system will allow customer to write off some amount of unpaid rent for a tenant if they did any work on the unit in exchange for rent.
2.1.4.4 The system will allow customer to enter the reason for the adjustment of rent in a comment field.

2.1.5 Report Management

2.1.5.1 The system will allow customer to view payment history by unit, by period, or by tenant showing rent due, payments made, and balance due.
2.1.5.2 The system will allow customer to view past due amount by unit, by period, or by tenant.
2.1.5.3 The system will allow customer to view total income by unit, by period, or by tenant.
2.1.5.4 The system will allow customer to view the list of tenants by unit; tenants who are behind, the dollar amount behind, and the date of last payment, and the list of all tenants whose lease is up.
2.1.5.5 The system will allow customer to view the list of current vacancies of units.
2.1.5.6 The system will allow the customer to view the report that lists all units for which lease term was broken for more than two times in the past.
2.1.5.7 The system will allow the customer to generate a report for total amount being deposited by the tenant for a unit.

2.2 Nonfunctional Requirements

2.2.1 Operational

2.2.1.1 The system should be a distributed system which will be able to handle up to 1000 different property management groups, each with 1 – 10000 units

2.2.2 Performance

2.2.2.1 Allow fast navigation and quick exit from particular screen within 3 seconds
2.2.2.2 Retrieve search results from the database within 5 seconds
2.2.2.3 Be available at all times i.e. 24 hours a day and 7 days a week
2.2.2.4 Allow any updates to be saved within 3 seconds
2.2.2.5 Store a minimum of 1000 tenants per unit in the database
2.2.2.6 Process selected report information quickly and accurately within 5 seconds
2.2.2.7 Response time for every submit should be 3 seconds

2.2.3 Security

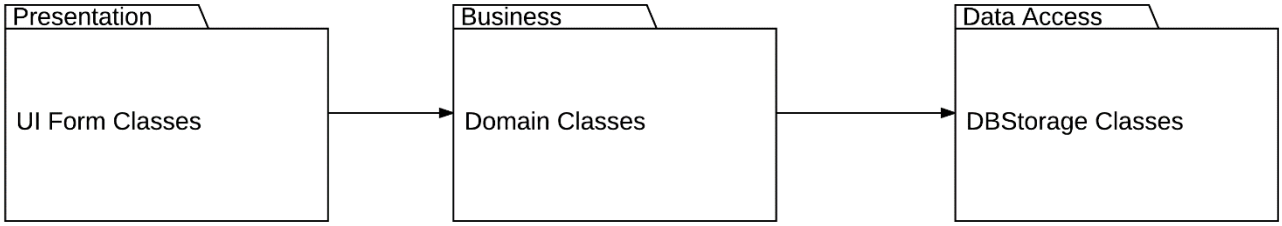
2.2.3.1 Only the authorized user should be able to access the system
2.2.3.2 Secure login features are provided like password must consist a combination of number and characters

2.2.4 Cultural and Political

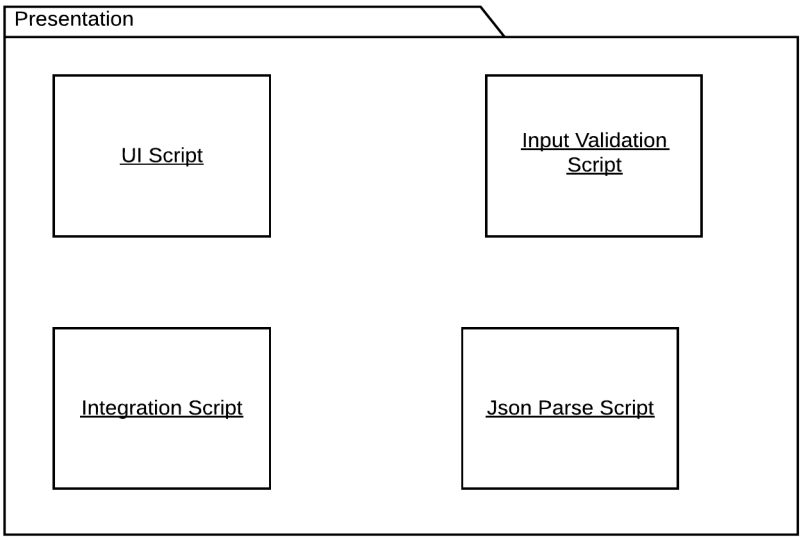
2.2.4.1 System will store the rent amount in US Dollars

3. Package and class diagram

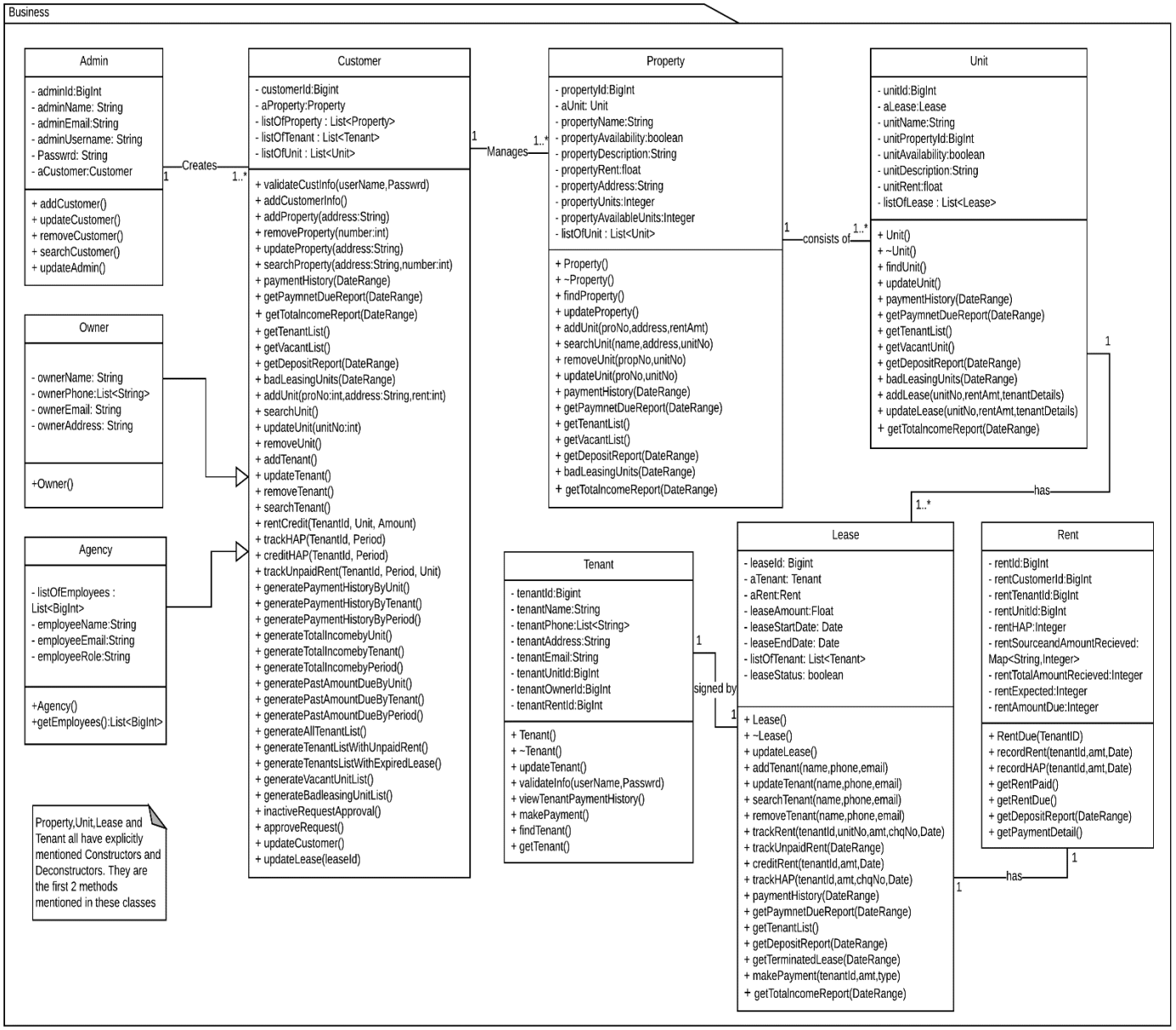
*UML Sketch of Property Management System**
**Getter and setter methods are implied for all the classes in the business layer.*



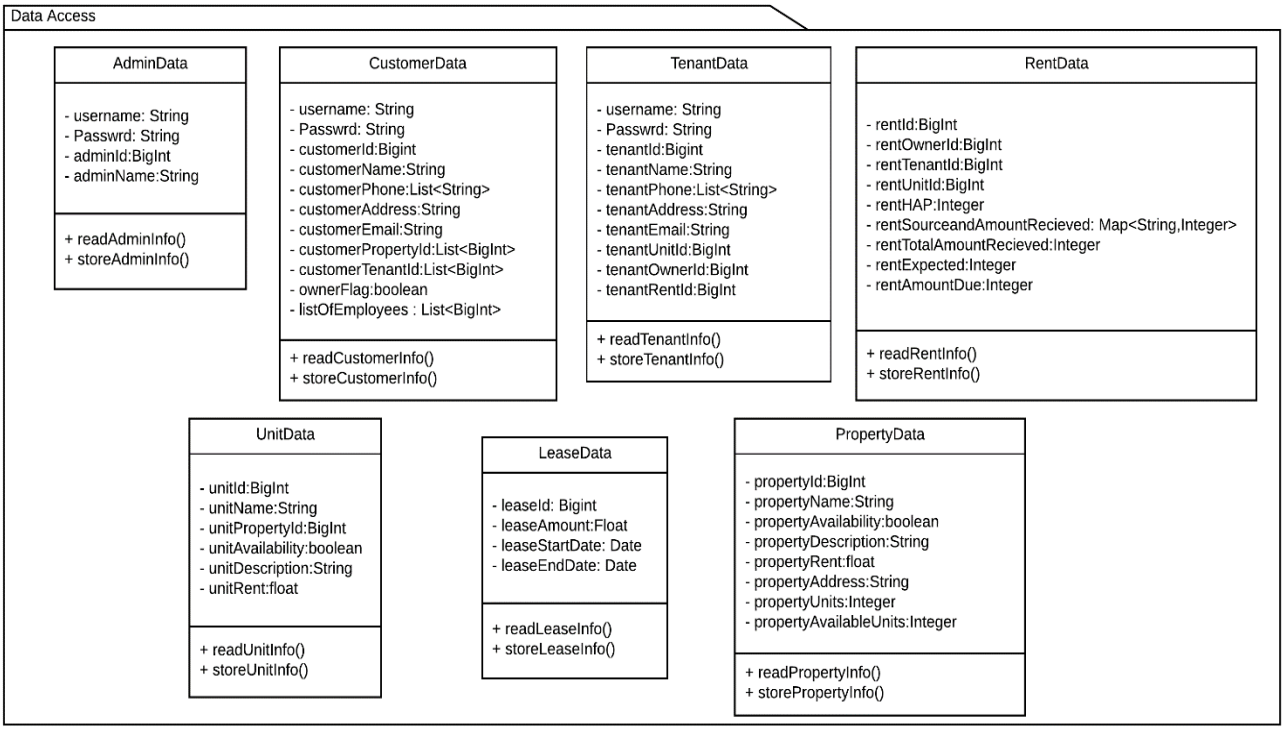
Presentation Layer



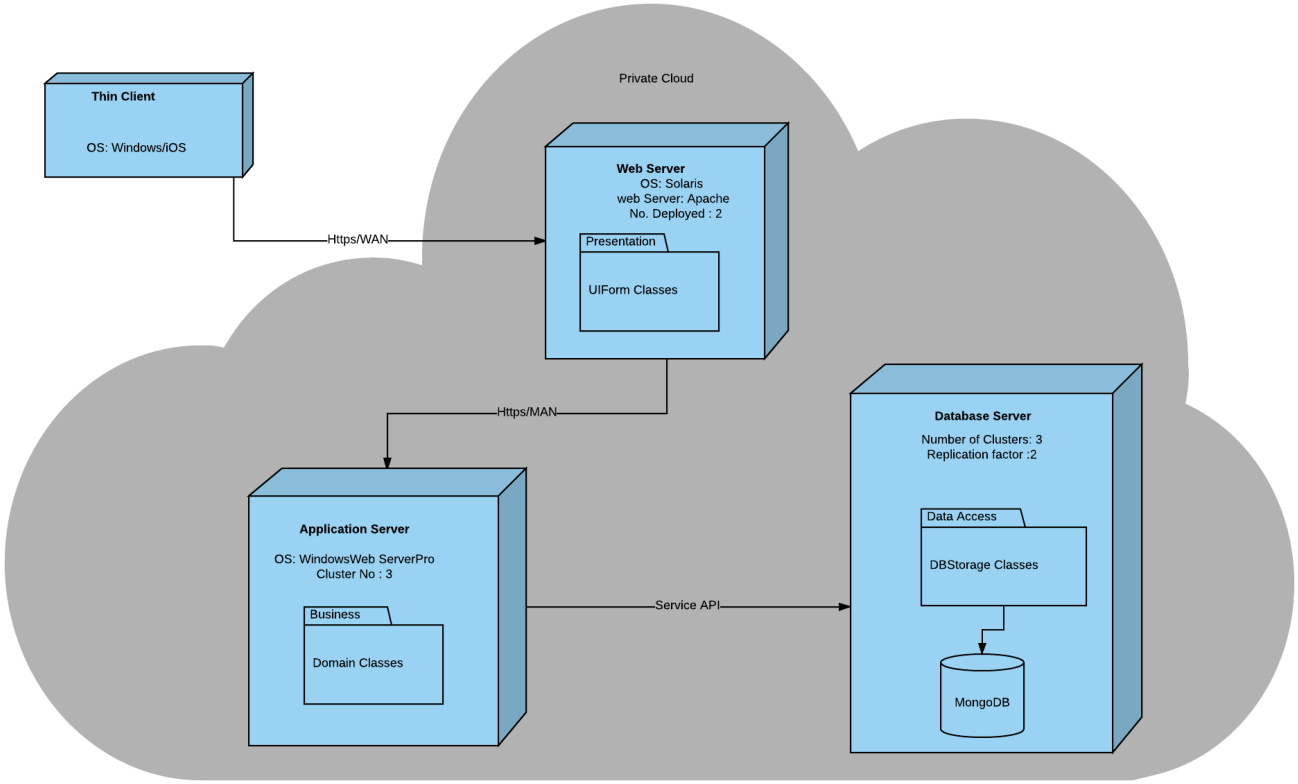
Business Layer



Data Access

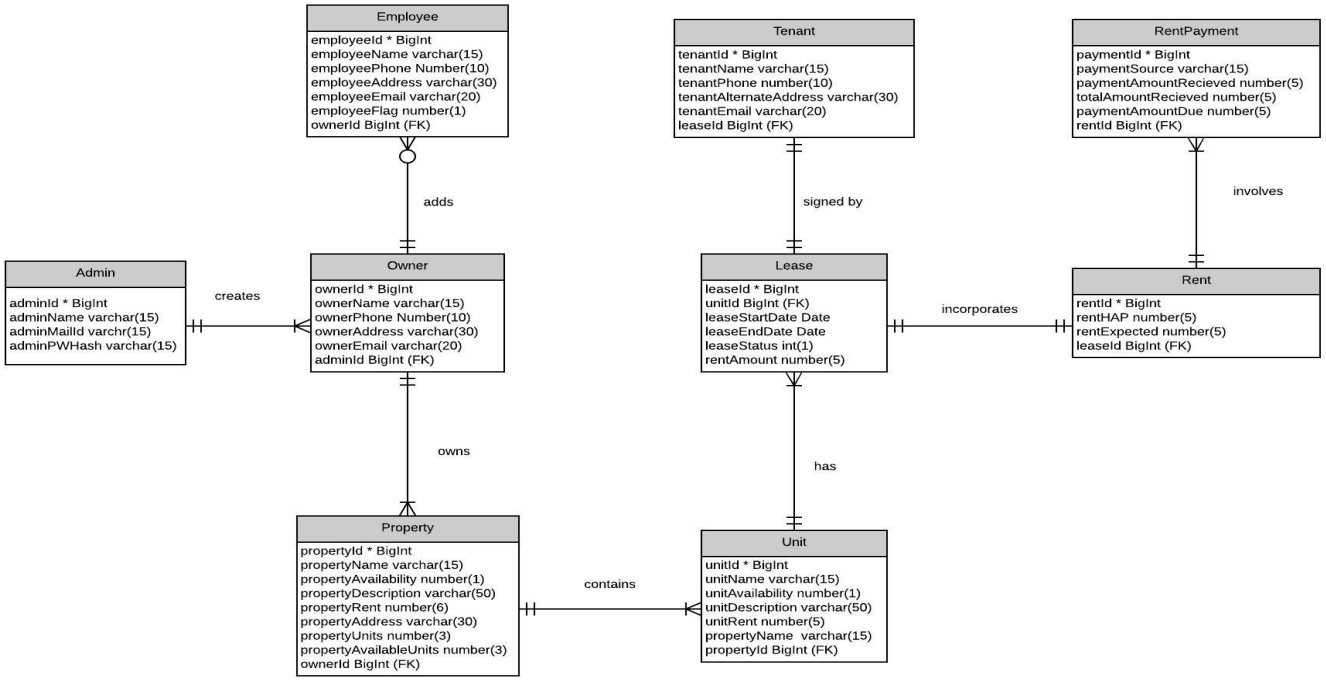

Data Access
AdminData
- username: String
- Passwrd: String
- adminId:BigInt
- adminName:String
+ readAdminInfo()
+ storeAdminInfo()
CustomerData
- username: String
- Passwrd: String
- customerId:Bigint
- customerName:String
- customerPhone:List<String>
- customerAddress:String
- customerEmail:String
- customerPropertyId:List<BigInt>
- customerTenantId:List<BigInt>
- ownerFlag:boolean
- listOfEmployees : List<BigInt>
+ readCustomerInfo()
+ storeCustomerInfo()
TenantData
- username: String
- Passwrd: String
- tenantId:Bigint
- tenantName:String
- tenantPhone:List<String>
- tenantAddress:String
- tenantEmail:String
- tenantUnitId:BigInt
- tenantOwnerId:BigInt
- tenantRentId:BigInt
+ readTenantInfo()
+ storeTenantInfo()
RentData
- rentId:BigInt
- rentOwnerId:BigInt
- rentTenantId:BigInt
- rentUnitId:BigInt
- rentHAP:Integer
- rentSourceandAmountRecieved: Map<String,Integer>
- rentTotalAmountRecieved:Integer
- rentExpected:Integer
- rentAmountDue:Integer
+ readRentInfo()
+ storeRentInfo()
UnitData
- unitId:BigInt
- unitName:String
- unitPropertyId:BigInt
- unitAvailability:boolean
- unitDescription:String
- unitRent:float
+ readUnitInfo()
+ storeUnitInfo()
LeaseData
- leaseId: Bigint
- leaseAmount:Float
- leaseStartDate: Date
- leaseEndDate: Date
+ readLeaseInfo()
+ storeLeaseInfo()
PropertyData
- propertyId:BigInt
- propertyName:String
- propertyAvailability:boolean
- propertyDescription:String
- propertyRent:float
- propertyAddress:String
- propertyUnits:Integer
- propertyAvailableUnits:Integer
+ readPropertyInfo()
+ storePropertyInfo()

4. Deployment Diagram



Deployment Diagram of Property Management System

5. Database Schema



Physical entity relationship diagram of Property Management System

6. Add Customer

6.1 Use case- Add Customer

Use Case Name: Add Customer	**ID**: UC-1	**Priority:** High
Actor: Admin		
Description: This use case allows system admin to create an account for the customer to login and become a registered user.		
Trigger: Admin wants to create new customer account who can access the system **Type: ☑ External ☐ Temporal**		
Preconditions: 1. System should be up and running 2. Admin's identity is authenticated		
Normal Course: 1.0 Create new customer for the system 1. Admin clicks on the 'Create New Account' button displayed on the screen 2. System displays a new screen and asks the admin to select for 'Username' and 'Password' 3. Admin enters the 'Username' and 'Password' and clicks on 'submit' button 4. System validates the data and displays a new screen asking the admin to fill details 'Name', 'Address', 'Contact Number', 'Title', 'Email Address' 5. Admin enters the information and clicks on 'Submit' 6. System saves the data for the Customer Account and notifies the admin that the account has been created		**Information for Steps:** ← Username, Password ← New customer Name, Address, Contact Number, Email address, Title → Account creation Notification
Alternative Course: 1.1 Admin cancels request 1. Admin chooses to cancel the request and selects 'exit' 2. System does not save the data and notifies the admin 'Account creation request is cancelled' 1.2 Admin enters Invalid Customer Account Information		 ← Cancellation Request → Cancellation Notification ← Invalid Data

1. System describes which entered data was invalid and presents the admin with suggestions to enter valid data 2. System prompts the admin to re-enter the valid information 3. Admin re-enters the details 4. System re-validates the information 5. If information is valid a. Customer Account Information is stored b. Admin is notified 6. If information is Invalid a. It goes through the Alternative course 1.2 again until it enters valid information or chooses to cancel	→ Re-enter data request ← Customer information ← Validated information → Account Creation Notification

Postconditions:

1. The Customer account is created successfully
2. The Customer information is stored in the Customer account and Confirmation message is sent
3. No Account is created if admin chooses 'Cancel request' option or enters invalid information

Exceptions:

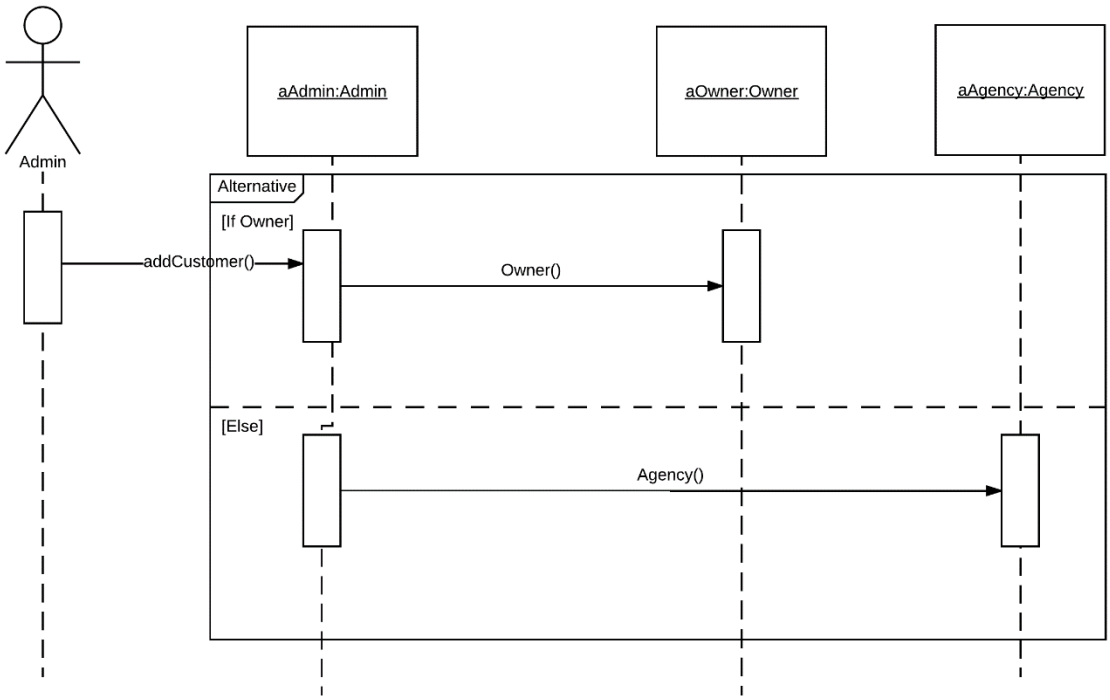
E1: If unexpected error occurs

1. The system displays a message "Unexpected Error found"
2. The system displays the home page

Summary

Inputs	Source	Outputs	Destination
Username, Password	Admin	Account creation Notification Cancellation	Customer Datastore
New username, Address, Contact Number, Email address, Title	Admin	Notification Invalid Data	System
Cancellation Request	Admin	Validated information	Customer Datastore
Customer Information	Admin	Re-enter data request	System

6.2 Sequence Diagram- Add Customer



7. Add Property

7.1 Use case- Add property

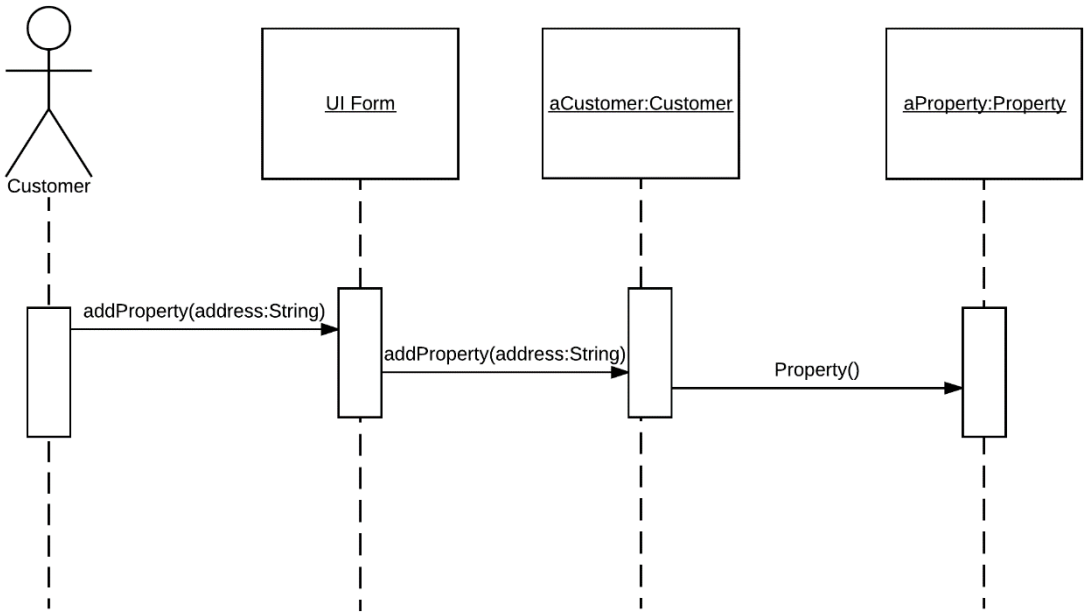
Use Case Name: Add Property	**ID:** UC-2	**Priority:** High
Actor: Customer		
Description: The customer wants to create a new property. The customer will insert the information of the property into the system. The system will save the information after confirmation.		
Trigger: The customer clicks on “Add Property” option on Home Screen and system displays Add Property screen. **Type: ☑ External ☐ Temporal**		
Preconditions: 1. System should be up and running. 2. The customer’s identity is authenticated.		
Normal Course:		**Information for Steps:**

1.0 Create a new tenant to the system 1. The customer enters the detailed information of the property (address, property number) and clicks on “Add New Property” 2. System displays a confirmation screen with message 3. System stores the Add New Property request in the datastore	←Property information ←Request Confirmation →Add New Property request
Alternative Course: N/A	
Postconditions: **1.** The property information is stored in the system	
Exceptions: N/A	

Summary

Inputs	**Source**	**Outputs**	**Destination**
Property information Request Confirmation	System System	Add New Property request	Property datastore

7.2 Sequence Diagram- Add property



7.3 Screen Mock-up- Add property

8. Update Property

8.1 Use case - Update Property

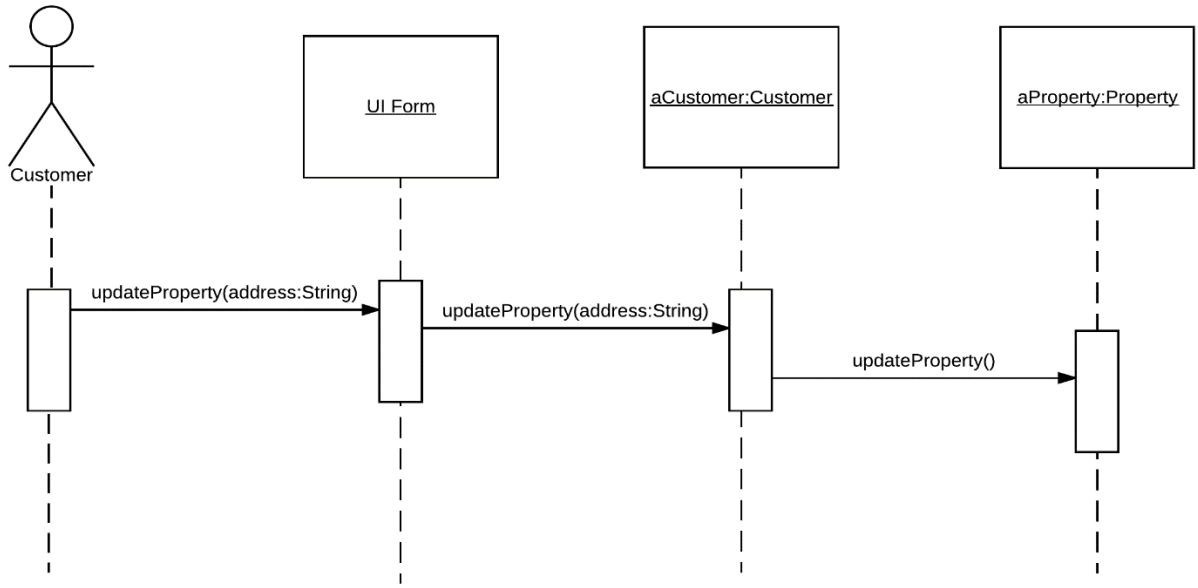
<table>
<tr><td>Use Case Name: Update Property</td><td>ID: UC-3</td><td>Priority: High</td></tr>
<tr><td colspan="3">Actor: Customer</td></tr>
<tr><td colspan="3">Description: The customer wants to update the property’s information. The customer will insert the property address and property number. The system will display the property information. The customer then revises the detailed information of the property. The system will save the information after confirmation.</td></tr>
<tr><td colspan="3">Trigger: The customer clicks on “Update Property” option on Home Screen and system displays Update Property screen.
Type: ☑ External ☐ Temporal</td></tr>
<tr><td colspan="3">Preconditions: 1. System should be up and running.
2. The customer’s identity is authenticated.</td></tr>
<tr><td colspan="2">Normal Course:
1.0 Update property’s information
1. The customer enters the property address and property number and clicks on “Update Property”
2. System displays the property information
3. The customer select the target property, revise the information, and clicks on “Update Property”
4. System displays a confirmation screen with message
5. System stores the Update Property request in the datastore</td><td>Information for Steps:

←Property address/number
→Property information
←Property information

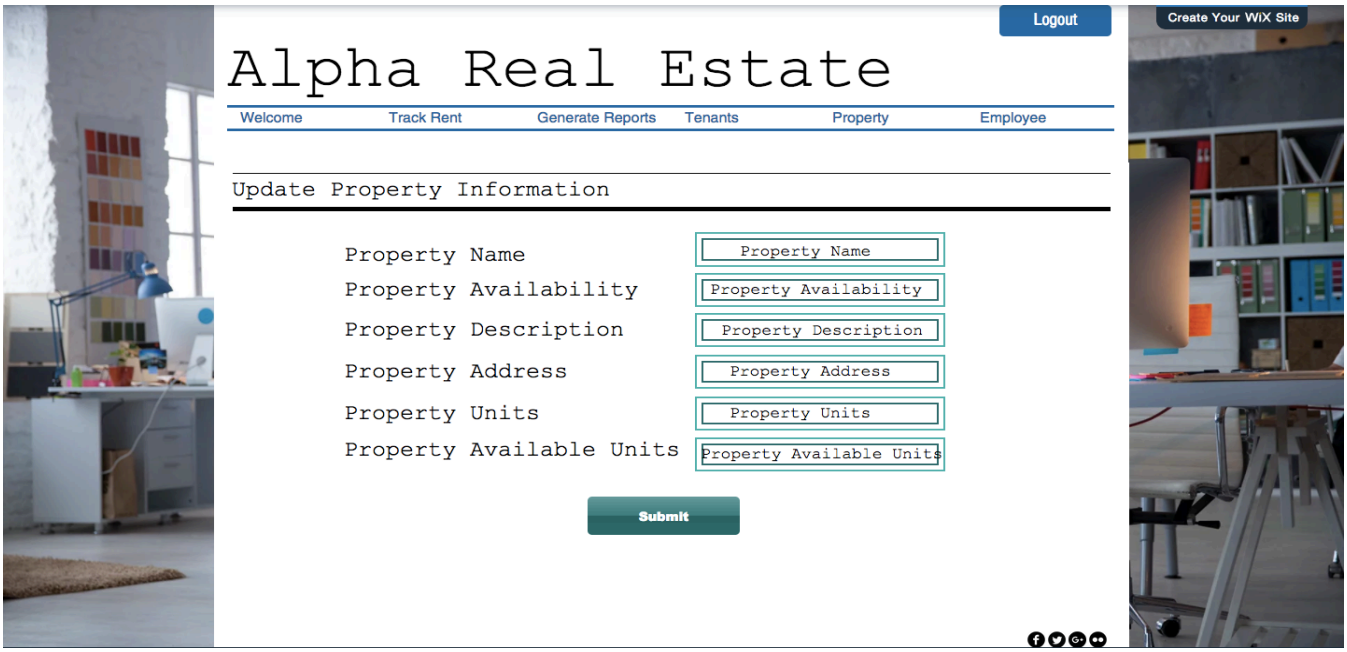
←Request Confirmation
→Update Property request</td></tr>
<tr><td colspan="2">Alternative Course: N/A</td><td></td></tr>
<tr><td colspan="3">Postconditions:
1. The updated property information is stored in the system</td></tr>
<tr><td colspan="3">Exceptions:
E1: System can’t find the property
1. The system displays message. “Sorry, we can’t find the property.”
2. The system asks the customer if he or she wants to type another property
3a. The customer types another property
4a. The system starts Normal Course again
3b. The customer asks to exit</td></tr>
</table>

4b. The system terminates the use case			
Summary			
Inputs	**Source**	**Outputs**	**Destination**
Property address/number	System	Property information	Property datastore
Property information	System	Update Property request	Property datastore
Request Confirmation	System	Request confirmation notification	Notification window

8.2 Sequence Diagram - Update Property



8.3 Screen Mock-up- Update Property



9. Search Property

9.1 Use case- Search Property

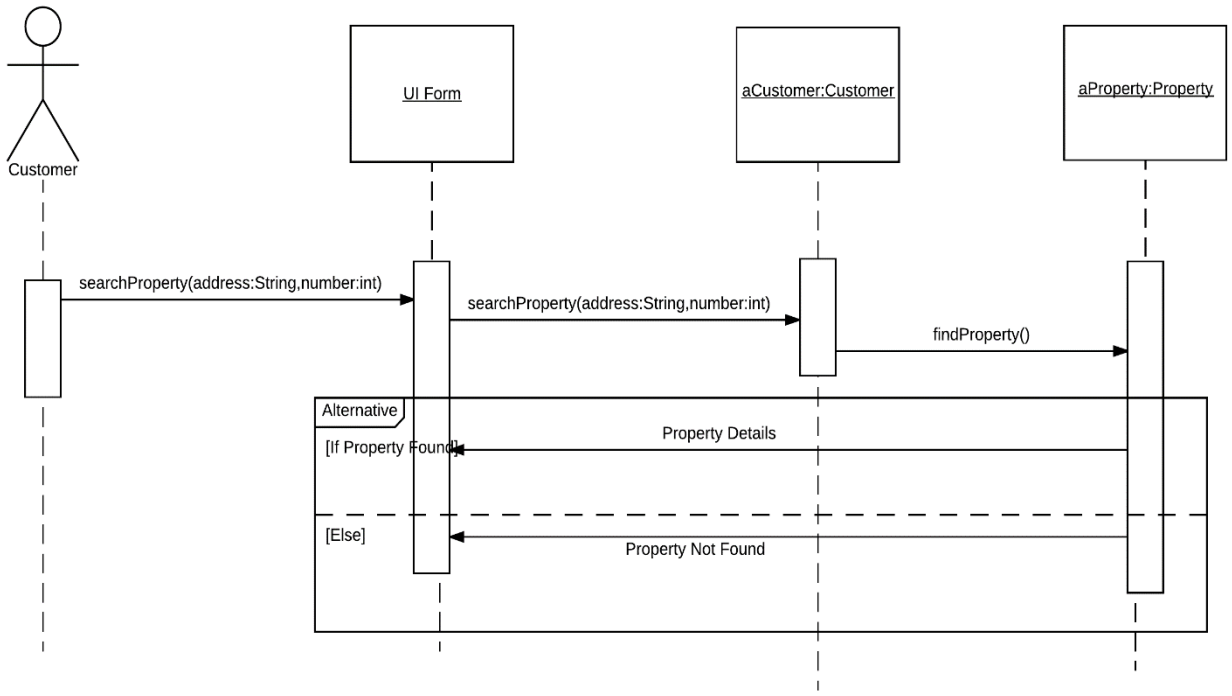
Use Case Name: Search Property	**ID**: UC-4	**Priority:** High
Actor: Customer		
Description: This use case describes how the customer is able to search properties		
Trigger: The customer clicks on "Search Property" **Type:** ☑ **External** ☐ **Temporal**		
Preconditions: 1. System should be up and running 2. The customer's identity is authenticated		
Normal Course: 1.0 Search property from the system 1. The customer clicks on the 'Search Property' button displayed on the screen 2. System asks the customer to enter the address or number of the property in the editable textbox displayed on the screen 3. The customer enters the address or number and clicks on 'submit' button		**Information for Steps:** ← Property address or number

4. System displays the information of the property	→ Property information
Alternative Course: N/A	
Postconditions: The customer can search properties and tenants on the basis of name successfully	
Exceptions: E1: Property address or number does not exist 1. The system displays a message “No records found for the desired property” 2. The system asks the customer if he or she wants to search with another address/number or to exit 3. The customer searches with another name 4. The system starts with Normal Course again 5. The customer asks to exit 6. The system terminates the use case	

Summary

Inputs	Source	Outputs	Destination
Property Name	Customer	Property information	Property Datastore

9.2 Sequence Diagram- Search Property



9.3 Screen Mock-up- Search Property

10. Remove Property

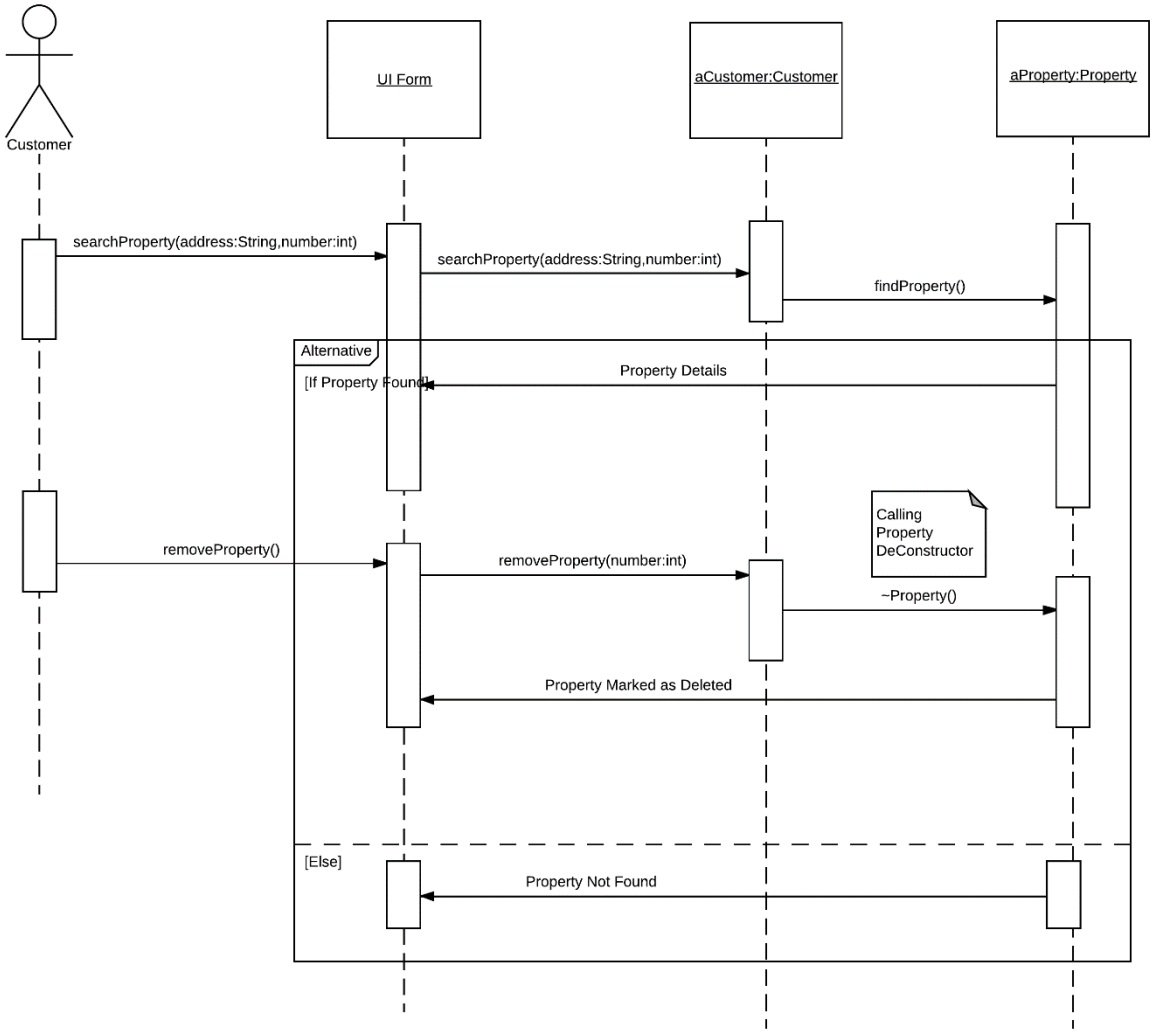
10.1 Use case- Remove Property

Use Case Name: Remove Property	**ID**: UC-5	**Priority:** Medium
Actor: Customer		
Description: The customer wants to remove a property. The customer will provide the system with the property identifier. The system will fetch the corresponding property details from the database and then on confirming removal it will mark the property as removed in the database.		
Trigger: **Type: ☑ External ☐ Temporal**		
Preconditions: 1.The customer should be logged into the system		
Normal Course: 1.0 Customer removes property from system 1.The customer clicks on property drop down and selects 'Remove property'		**Information for Steps:**

2. The screen displays remove property screen 3. Customer provides identifying information of property to be removed 4. System will fetch the corresponding property information 5. customer will click on 'Remove property' link 6. Confirmation dialogue box will pop up and System will display message 'Are you sure you want to remove this property 7. If customer clicks on Yes. The system will mark property as removed in the database.	←property number →property Information
Alternative Course: **1.1 Identifying information does not match any record in Database (branch at step 3)** 1. The system will display a message 'No records found. Please check information added or do you want to add a new property?' 1a. If the customer clicks on 'Add a new property' the system redirects to the 'Add property' page 2a. The customer can enter the identifying information of property once again **1.2 Customer chooses not to remove property (branch step 6)** 1. The customer can choose option 'No' in confirmation dialogue box 2. The dialogue box will automatically close and the 'Remove property' will be shown again.	

Post conditions: The corresponding property information will be marked as delete in the database.			
Exceptions: N/A			
Summary			
Inputs	**Source**	**Outputs**	**Destination**
Identifying information	Customer	Update database with records marked as removed	Property datastore

10.2 Sequence Diagram- Remove property



10.3 Screen Mock-up- Remove Property

11. Add Unit

11.1 Use case- Add unit

Use Case Name: Add Unit	**ID:** UC-6	**Priority:** High
Actor: Customer		
Description: The customer wants to create a new unit. The customer will insert the information of the unit into the system. The system will save the information after confirmation.		
Trigger: The customer clicks on "Add Unit" option on Home Screen and system displays Add Unit screen. **Type:** ☑ **External** ☐ **Temporal**		
Preconditions: 1. System should be up and running. 2. The customer's identity is authenticated.		
Normal Course: 1.0 Create a new tenant to the system		**Information for Steps:** ←Unit's information

1. The customer enters the detailed information of the unit (address, unit number, and rent per month) and clicks on “Create New Unit” 2. System displays a confirmation screen with message 3. System stores the Add New Unit request in the datastore	←Request Confirmation →Create New Unit request
Alternative Course: N/A	

Postconditions:

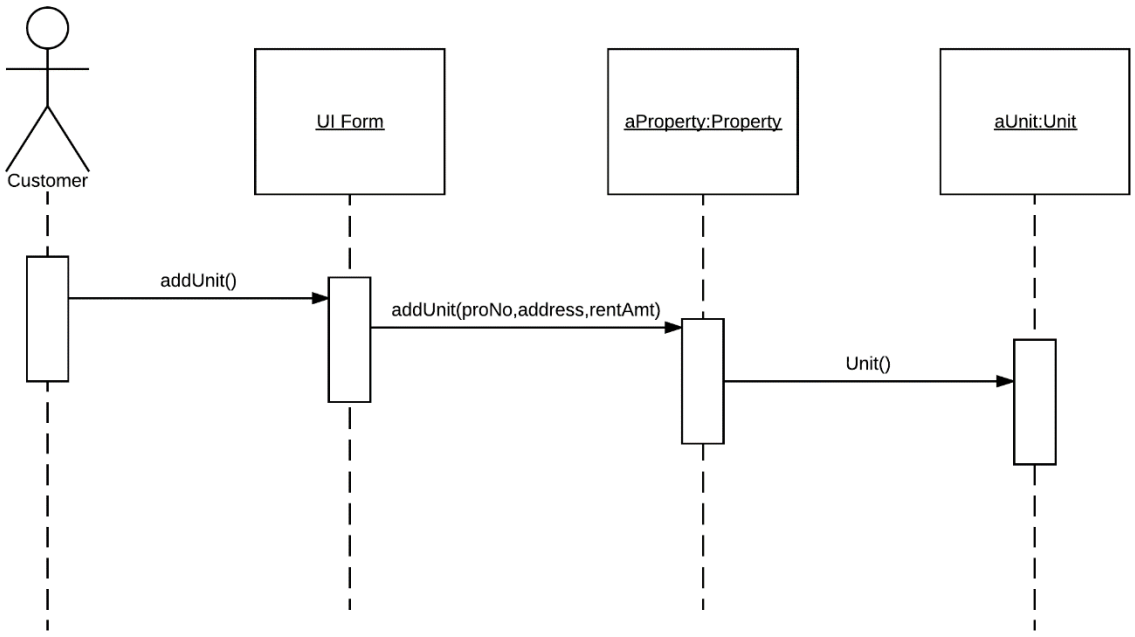
1. The unit information is stored in the system

Exceptions: N/A

Summary

Inputs	**Source**	**Outputs**	**Destination**
Unit’s information Request Confirmation	System System	Create New Unit request	Unit datastore

11.2 Sequence diagram- Add unit



11.3 Screen Mock-up- Add unit

12. Update Unit

12.1 Use case- Update Unit

<table>
<tr><td>Use Case Name: Update Unit</td><td>ID: UC-7</td><td>Priority: High</td></tr>
<tr><td colspan="3">Actor: Customer</td></tr>
<tr><td colspan="3">Description: The customer wants to update the unit’s information. The customer will insert the unit address and unit number. The system will display the unit’s information. The customer then revises the detailed information of the unit. The system will save the information after confirmation.</td></tr>
<tr><td colspan="3">Trigger: The customer clicks on “Update Unit” option on Home Screen and system displays Update Unit screen.
Type: ☑ External ☐ Temporal</td></tr>
<tr><td colspan="3">Preconditions: 1. System should be up and running.
2. The customer’s identity is authenticated.</td></tr>
<tr><td colspan="2">Normal Course:
1.0 Update unit’s information
1. The customer enters the unit address and unit number and clicks on “Update Unit”
2. System displays the unit’s information
3. The customer select the target unit, revise the information, and clicks on “Update Unit”
4. System displays a confirmation screen with message
5. System stores the Update Unit request in the datastore</td><td>Information for Steps:

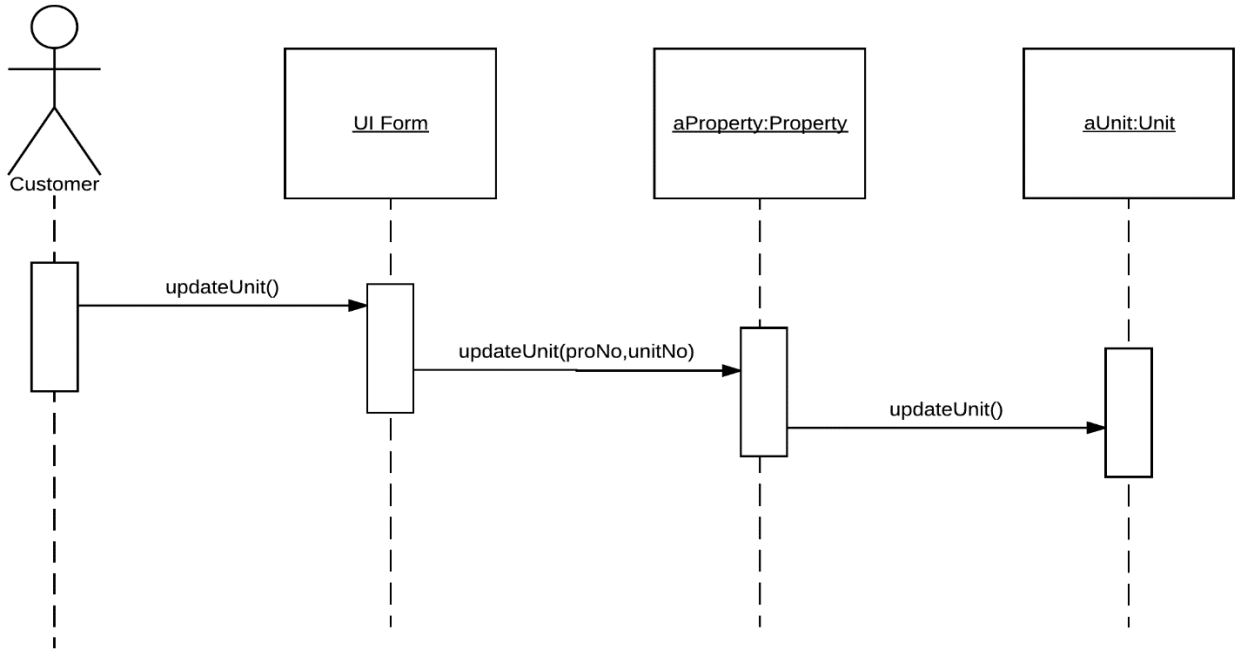
←Unit address/number

→List of units
←Unit’s information

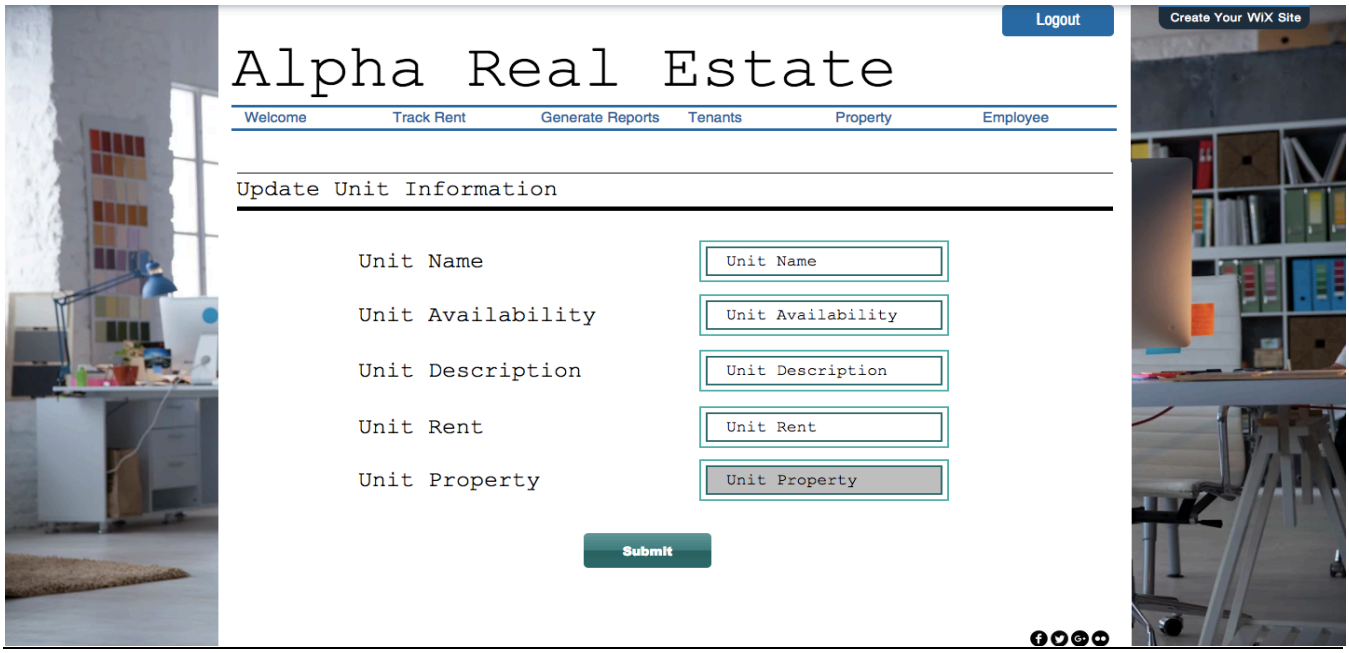
←Request Confirmation
→Update Unit request</td></tr>
<tr><td colspan="2">Alternative Course: N/A</td><td></td></tr>
<tr><td colspan="3">Postconditions:
1. The updated unit information is stored in the system</td></tr>
<tr><td colspan="3">Exceptions:
E1: System can’t find the unit
1. The system displays message. “Sorry, we can’t find the unit.”
2. The system asks the customer if he or she wants to type another unit
3a. The customer asks to type another unit
4a. The system starts Normal Course again
3b. The customer asks to exit
4b. The system terminates the use case</td></tr>
<tr><td colspan="3">Summary</td></tr>
</table>

Inputs	Source	Outputs	Destination
Unit address/number Unit's information Request Confirmation	System System System	List of tenants Update Unit request Request confirmation window	Unit datastore Unit datastore Notification window

12.2 Sequence diagram- Update unit



12.3 Screen Mock-up- Update unit



13. Search Unit

13.1 Use case- Search unit

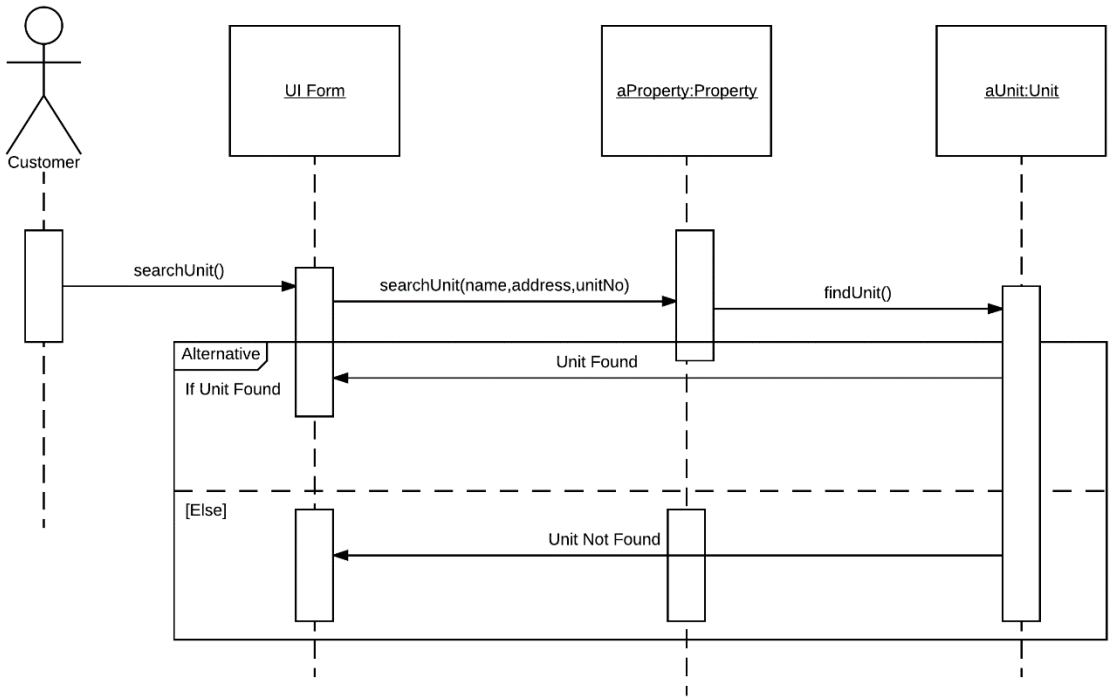
Use Case Name: Search Unit	**ID**: UC-8	**Priority:** High
Actor: Customer		
Description: This use case describes how the customer can search units.		
Trigger: Customer clicks on "Search Unit" **Type:** ☑ **External** ☐ **Temporal**		
Preconditions: 1. System should be up and running 2. Customer's identity is authenticated		
Normal Course: 1.0 Search unit from the system 1. Customer clicks on the 'Search Unit' button displayed on the screen 2. System asks the customer to enter the 'name' of the unit in the editable textbox displayed on the screen 3. Customer enters the 'name' and clicks on 'submit' button 4. System displays the name of the unit with 'Area located' and 'Availability status'		**Information for Steps:** ← Unit name → Unit name, Area located and Unit's Availability Status

Alternative Course: N/A	
Postconditions: Customer can search tenants on the basis of name successfully	
Exceptions: E1: Unit name does not exist 1. The system displays a message “No records found for the desired unit” 2. The system asks the customer if she wants to search with another name or to exit 3. The customer searches with another name 4. The system starts with Normal Course again 5. The customer asks to exit 6. The system terminates the use case	

Summary

Inputs	Source	Outputs	Destination
Unit Name	Customer	Unit name, Area located and Unit’s Availability Status	Unit Datastore

13.2 Sequence diagram - Search unit



13.3 Screen Mock-up- Search unit

14. Remove Unit

14.1 Use case- Remove Unit

<table>
<tr><td colspan="2">Use Case Name: Remove Unit</td><td>ID: UC-9</td><td>Priority: Medium</td></tr>
<tr><td colspan="4">Actor: Customer</td></tr>
<tr><td colspan="4">Description: The customer wants to remove a unit. The customer will provide the system with the unit identifier. The system will fetch the corresponding unit details from the database and then on confirming removal it will mark the unit as removed in the database.</td></tr>
<tr><td colspan="4">Trigger:
Type: ☑ External ☐ Temporal</td></tr>
<tr><td colspan="4">Preconditions:
1.The customer should be logged into the system</td></tr>
<tr><td colspan="3">Normal Course:
1.0 The customer removes unit from system
1.The customer clicks on Units drop down and selects ‘Remove Unit’
2. The screen displays remove unit screen
3. The customer provides identifying information of unit to be removed
4. System will fetch the corresponding unit information
5. The customer will click on ‘Remove Unit’ link
6. Confirmation dialogue box will pop up and System will display message ‘Are you sure you want to remove this unit
7. If customer clicks on Yes. The system will mark unit as removed in the database.</td><td>Information for Steps:

←Unit number

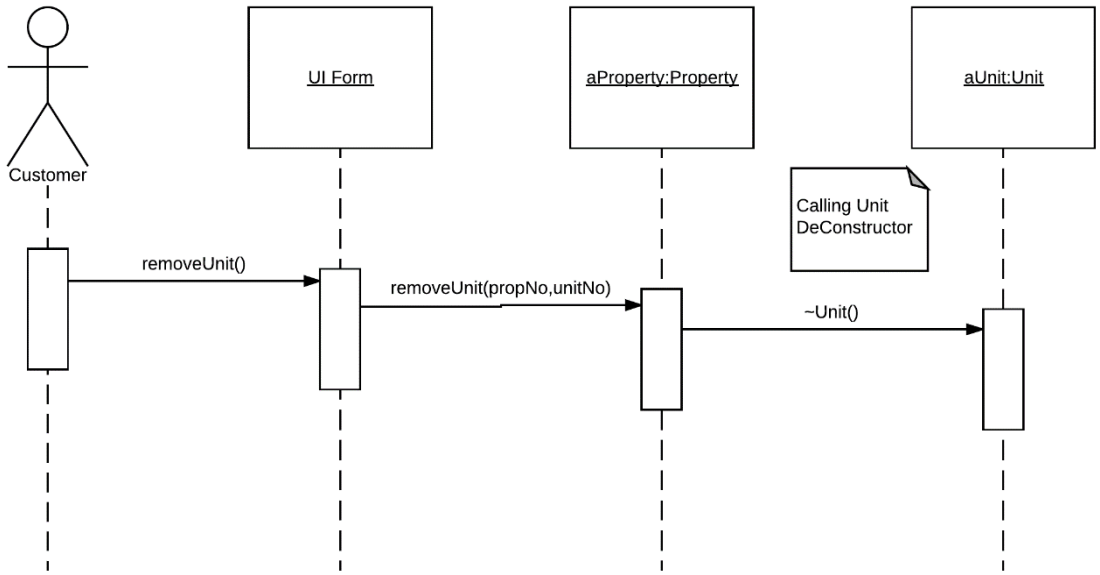
→Unit Information</td></tr>
<tr><td colspan="3">Alternative Course:
1.1 Identifying information does not match any record in Database (branch at step 3)
1. The system will display a message ‘No records found. Please check information added or do you want to add a new unit?’
1a. If customer clicks on ‘Add a new Unit’ the system redirects to the ‘Add Unit’ page
2a. The customer can enter the identifying information of unit once again
1.2 The customer chooses not to remove unit (branch step 6)</td><td></td></tr>
</table>

1. The customer can choose option 'No' in confirmation dialogue box 2. The dialogue box will automatically close and the 'Remove Unit' will be shown again.	
Post conditions: The corresponding unit information will be marked as delete in the database.	
Exceptions: N/A	

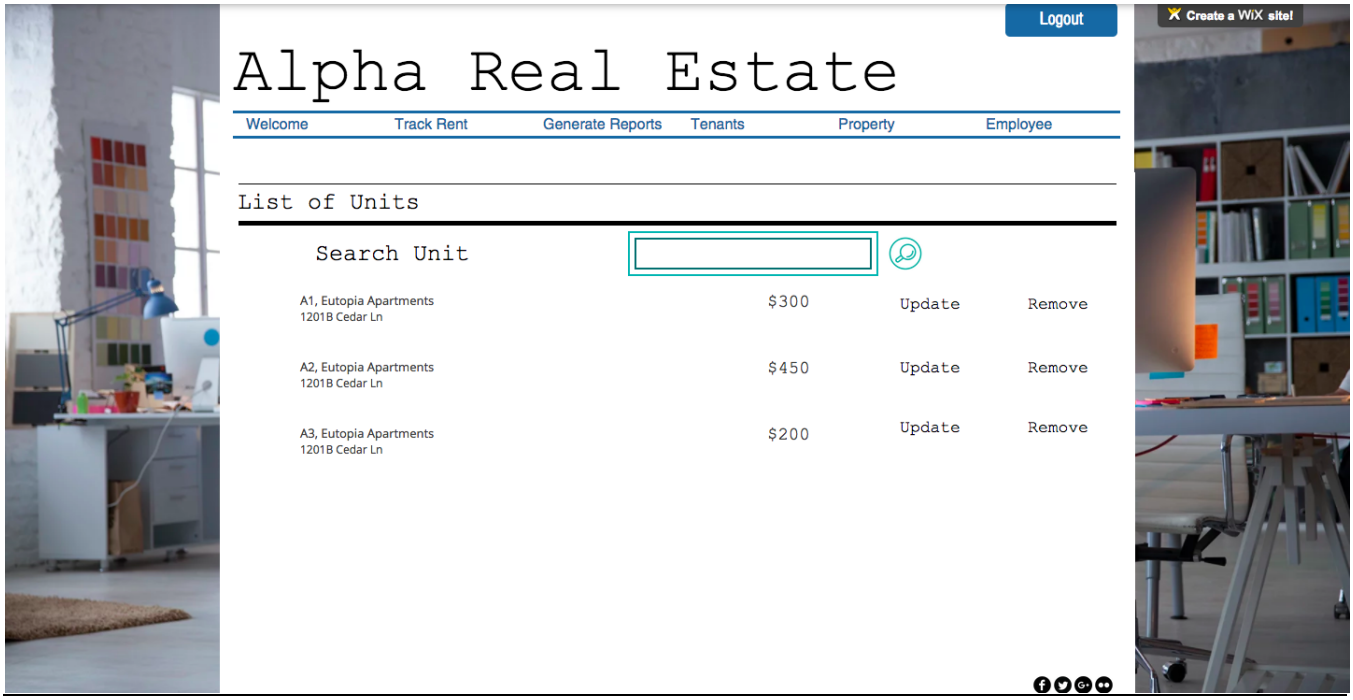
Summary

Inputs	**Source**	**Outputs**	**Destination**
Identifying information	Customer	Update database with records marked as removed	Unit datastore, Property datastore

14.2 Sequence diagram- Remove unit



14.3 Screen Mock-up- Remove Unit



15. Add Tenant

15.1 Use case- Add Tenant

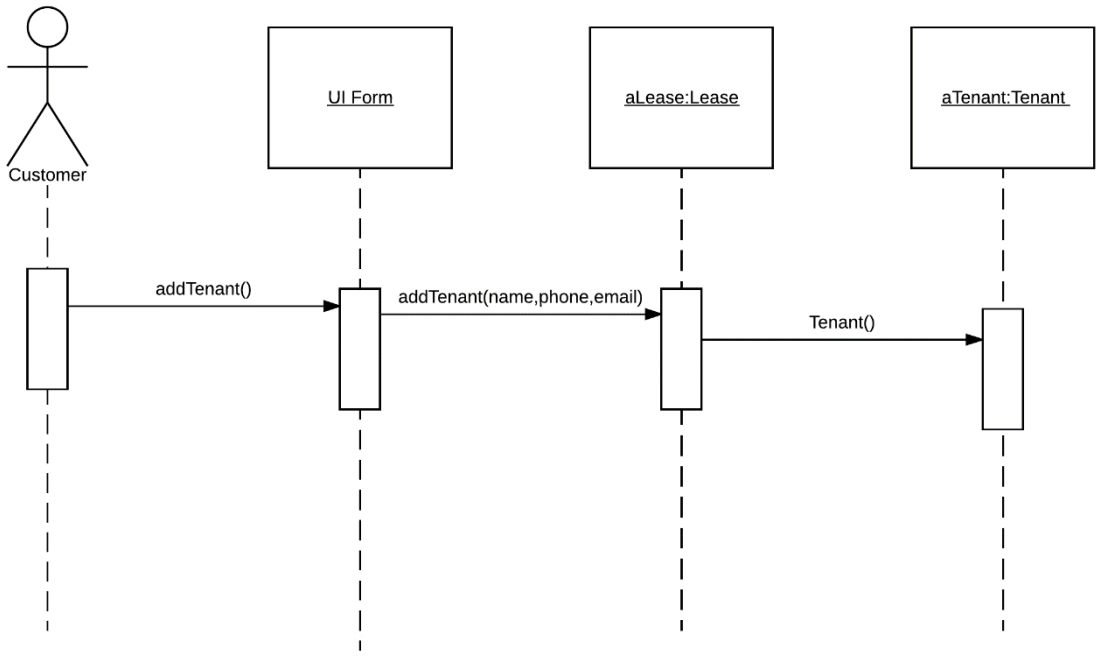
Use Case Name: Add Tenant	**ID**: UC-10	**Priority:** High
Actor: Customer		
Description: The customer wants to create a new tenant. The customer will insert the information of the tenant into the system. The system will save the information after confirmation.		
Trigger: The customer clicks on "Create New Tenant" option on Home Screen and system displays Create New Tenant screen. **Type: ☑ External ☐ Temporal**		
Preconditions: 1. System should be up and running. 2. The customer's identity is authenticated.		
Normal Course: 1.0 Create a new tenant to the system 1. Customer enters the detailed information of the tenant (first		**Information for Steps:** ←Tenant information

name, last name, phone numbers, deposit amount paid) and clicks on “Create New Tenant” 2. System displays a confirmation screen with message 3. System stores the Create New Tenant request in the datastore	←Request Confirmation →Create New Tenant request
Alternative Course: N/A	
Postconditions: 1. The new tenant information is stored in the system	
Exceptions: N/A	

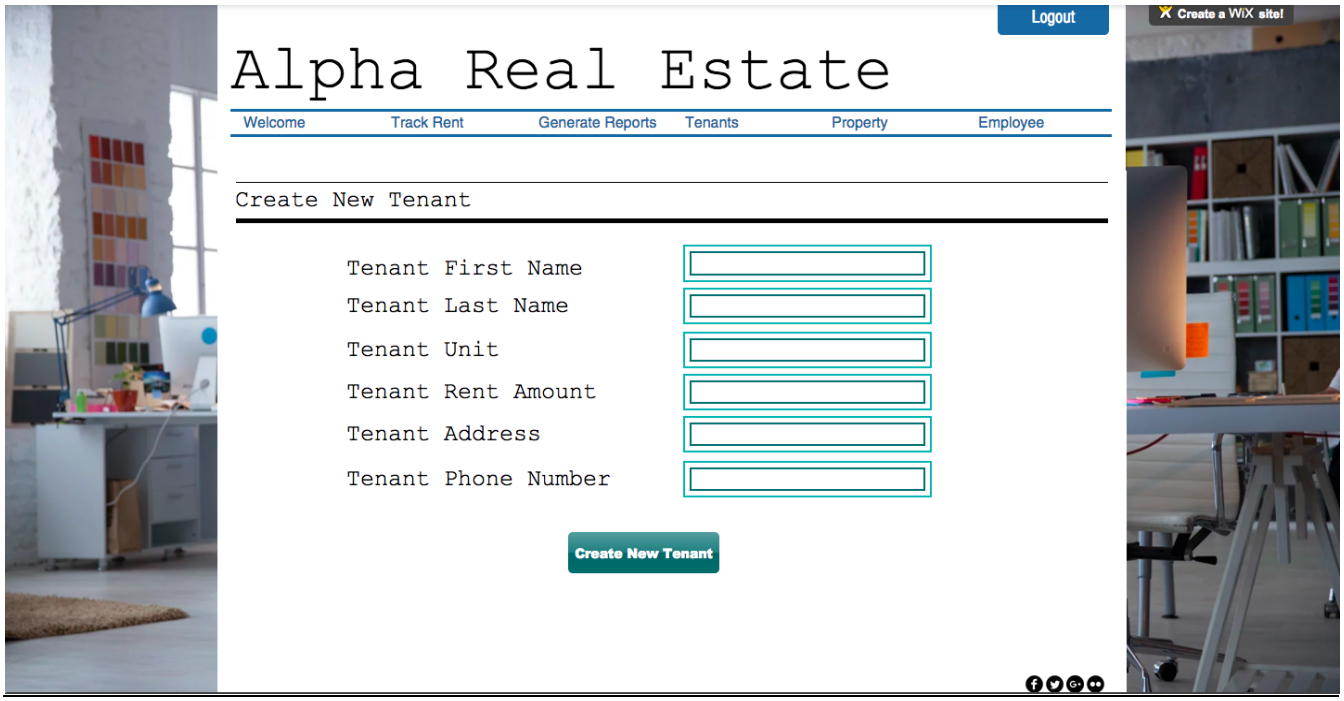
Summary

Inputs	**Source**	**Outputs**	**Destination**
Tenant information	System	Create New Tenant request	Tenant datastore
Request Confirmation	System	Request confirmation notification	Notification window

15.2 Sequence diagram- Add Tenant



15.3 Screen Mock-up- Add Tenant



16. Update Tenant

16.1 Use case- Update Tenant

Use Case Name: Update Tenant	**ID**: UC-11	**Priority:** High
Actor: Customer		
Description: The customer wants to update the tenant's information. The customer will insert the tenant's name. The system will display the tenant's information and the unit number. The customer then revises the detailed information of the tenant. The system will save the information after confirmation.		
Trigger: The customer clicks on "Update Tenant" option on Home Screen and system displays Update Tenant screen. **Type: ☑ External ☐ Temporal**		

Preconditions: 1. System should be up and running.
2. The customer's identity is authenticated.

Normal Course:	**Information for Steps:**
1.0 Update tenant's information 1. The customer enters the tenant's name and clicks on "Update Tenant" 2. System displays the tenant's name, unit number, and his or her information 3. The customer select the target tenant, revise the information, and clicks on "Update Tenant" 4. System displays a confirmation screen with message 5. System stores the Update Tenant request in the datastore.	←Tenant's name →List of tenants ←Tenant's information ←Request Confirmation →Update Tenant request
Alternative Course: N/A	

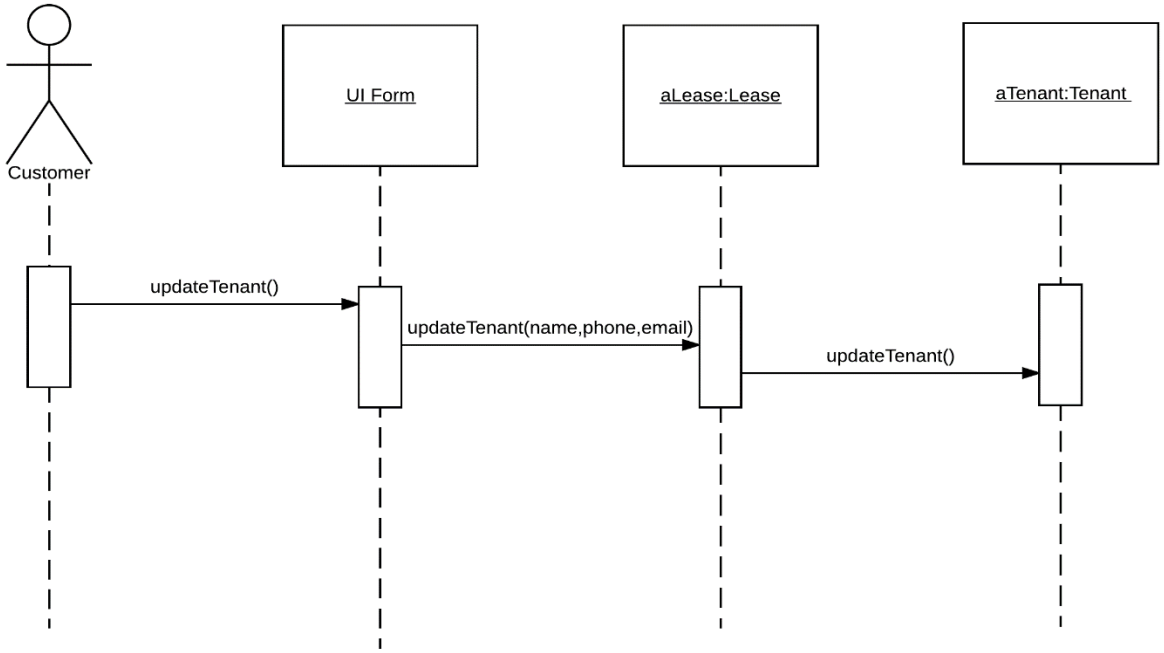
Postconditions:
1. The updated tenant information is stored in the system

Exceptions:
E1: System can't find the tenant
1. The system displays message. "Sorry, we can't find the tenant."
2. The system asks the customer if he or she wants to type another name or exit
3a. The customer asks to type another name
4a. The system starts Normal Course again
3b. The customer asks to exit
4b. The system terminates the use case

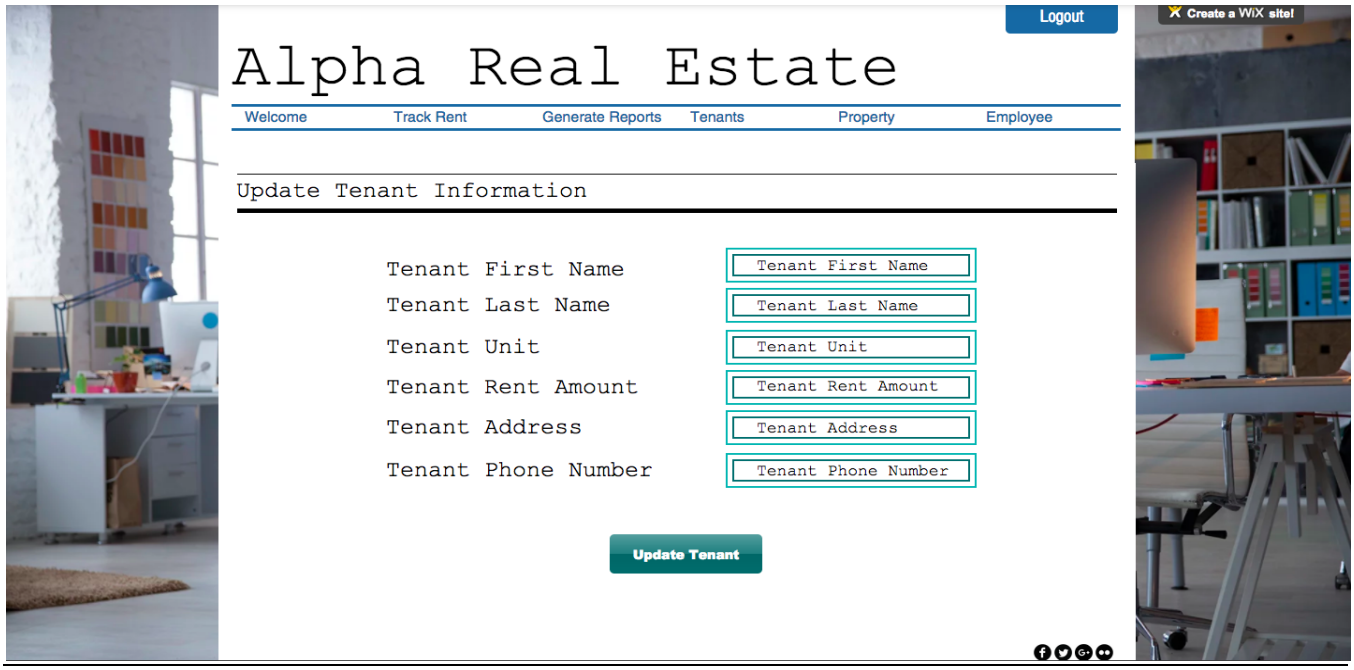
Summary

Inputs	**Source**	**Outputs**	**Destination**
Tenant's name Tenant's information Request Confirmation	System System System	List of tenants Update Tenant request	Display Screen Tenant datastore

16.2 Sequence diagram- Update Tenant



16.3 Screen Mock-up- Update Tenant



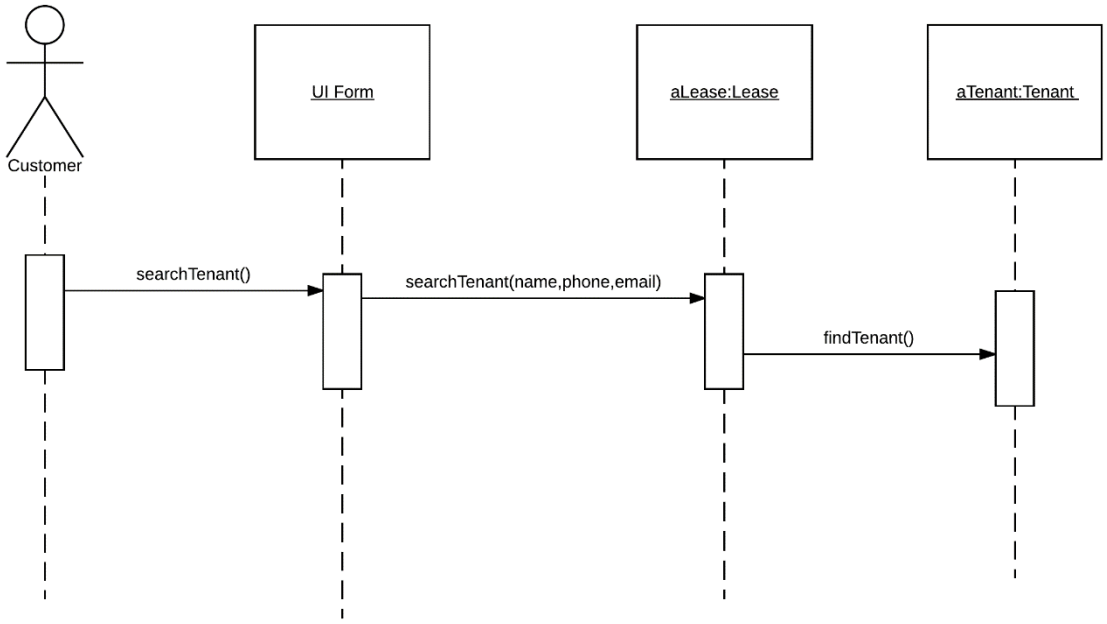
17. Search Tenant

17.1 Use case- Search Tenant

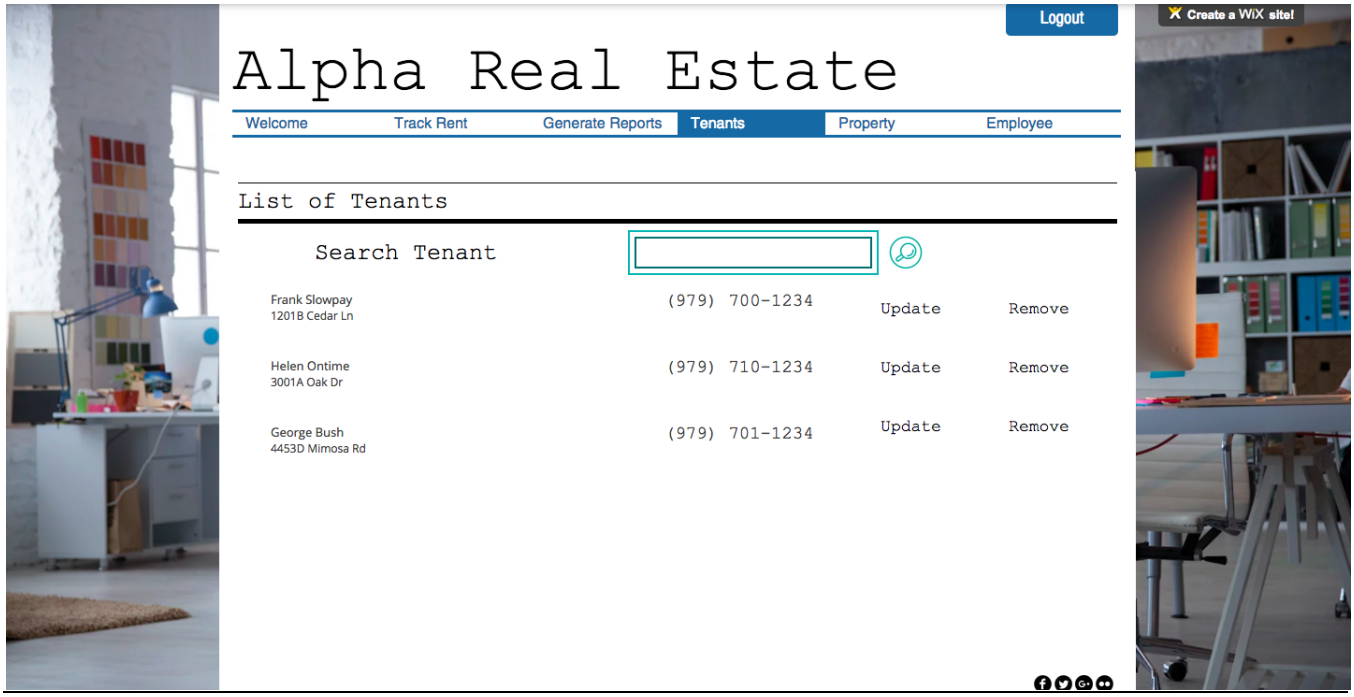
<table>
<tr><td colspan="2">Use Case Name: Search tenant</td><td>ID: 12</td><td>Priority: High</td></tr>
<tr><td colspan="4">Actor: Customer</td></tr>
<tr><td colspan="4">Description: This use case describes how the customer is able to search tenant on the basis of their name</td></tr>
<tr><td colspan="4">Trigger: Customer clicks on “Search tenant”
Type: ☑ External ☐ Temporal</td></tr>
<tr><td colspan="4">Preconditions: 1. System should be up and running
2. Customer’s identity is authenticated</td></tr>
<tr><td colspan="3">Normal Course:
1.0 Search tenants from the system
1. Customer clicks on the ‘Search Tenant’ button displayed on the screen
2. System asks the customer to enter the ‘name’ of the tenant in the editable textbox displayed on the screen
3. Customer enters the ‘name’ and clicks on ‘submit’ button
4. System displays a list of Tenants with ‘Name’, ‘Unit’ and ‘Contact Number’</td><td>Information for Steps:

← Tenant name
→ Tenant name, Unit and Contact Number</td></tr>
<tr><td colspan="4">Postconditions: Customer is able to search tenants on the basis of name successfully</td></tr>
<tr><td colspan="4">Exceptions:
E1: Tenant name does not exist
1. The system displays a message “No records found for the desired tenant”
2. The system asks the customer if she wants to search with another name or to exit
3. The customer searches with another name
4. The system starts with Normal Course again
5. The customer asks to exit
6. The system terminates the use case</td></tr>
<tr><td colspan="4">Summary</td></tr>
<tr><td>Inputs</td><td>Source</td><td>Outputs</td><td>Destination</td></tr>
<tr><td>Tenant Name</td><td>Customer</td><td>Tenant name, Unit and Contact</td><td>Tenant Datastore</td></tr>
</table>

17.2 Sequence diagram- Search Tenant



17.3 Screen Mock-up- Search Tenant



18. Remove Tenant

18.1 Use case- Remove Tenant

<table>
<tr><td colspan="2">Use Case Name: Remove Tenant</td><td>ID: UC-13</td><td>Priority: Medium</td></tr>
<tr><td colspan="4">Actor: Customer</td></tr>
<tr><td colspan="4">Description: The customer wants to remove a tenant. The customer will provide the system with the tenant identifier such as phone number or email. The system will fetch the corresponding tenant details from the database and then on confirming removal it will mark the tenant as removed in the database.</td></tr>
<tr><td colspan="4">Trigger:
Type: ☑ External ☐ Temporal</td></tr>
<tr><td colspan="4">Preconditions:
1.The customer should be logged into the system</td></tr>
<tr><td colspan="3">Normal Course:
1.0 customer removes tenant from system
1.The customer clicks on Tenants dropdown and selects ‘Remove tenant’
2. The screen displays the remove tenant screen
3. The customer provides identifying information such as phone number /email of tenant to be removed
4. System will fetch the corresponding tenant information
5. The customer will click on ‘Remove Tenant’ link
6. Confirmation dialogue box will pop up and System will display message ‘Are you sure you want to remove tenant and refund $### amount?” (This is the security deposit)
7. If customer clicks on Yes. The system will mark tenant as removed in the database.</td><td>Information for Steps:

←Tenant Phone No./Email

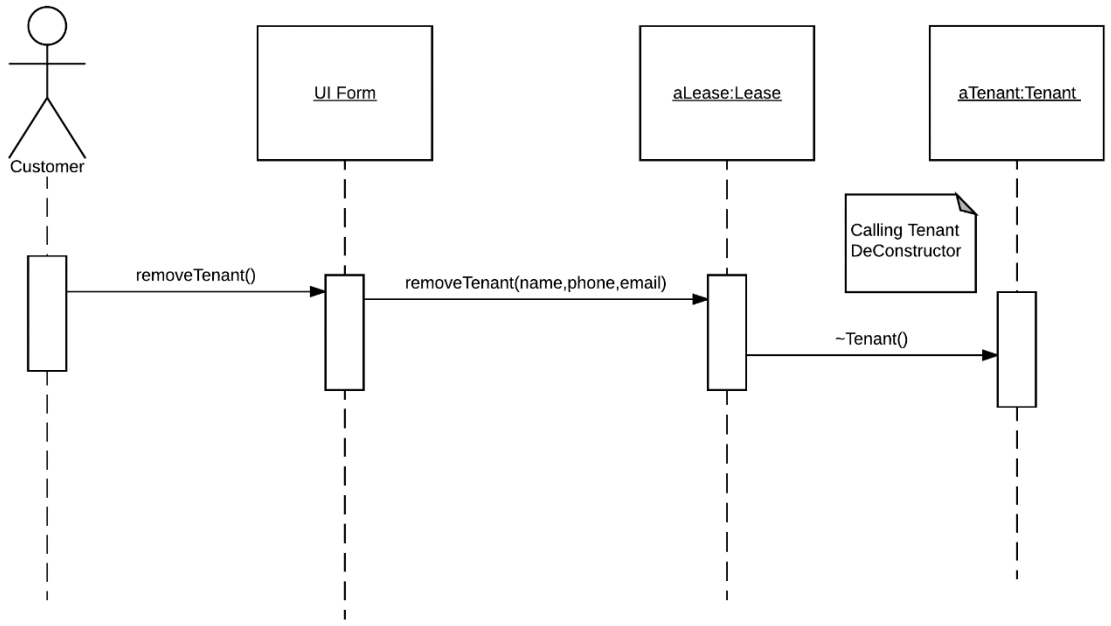
→Tenant Information</td></tr>
<tr><td colspan="3">Alternative Course:
1.1 Identifying information does not match any record in Database (branch at step 3)
1. The system will display a message ‘No records found. Please check information added or do you want to add a new tenant?’
1a. If customer clicks on ‘Add a new Tenant’ the system redirects to the ‘Add Tenant’ page
2a. The customer can enter the identifying information of tenant once again</td><td></td></tr>
</table>

1.2 Customer chooses not to remove tenant (branch step 6) 1. The customer can choose option 'No' in confirmation dialogue box. 2. The dialogue box will automatically close and the 'Remove Tenant' will be shown again.	
Post conditions: The corresponding tenant information will be marked as delete in the database and the corresponding unit will be marked as vacant in the database.	
Exceptions: N/A	

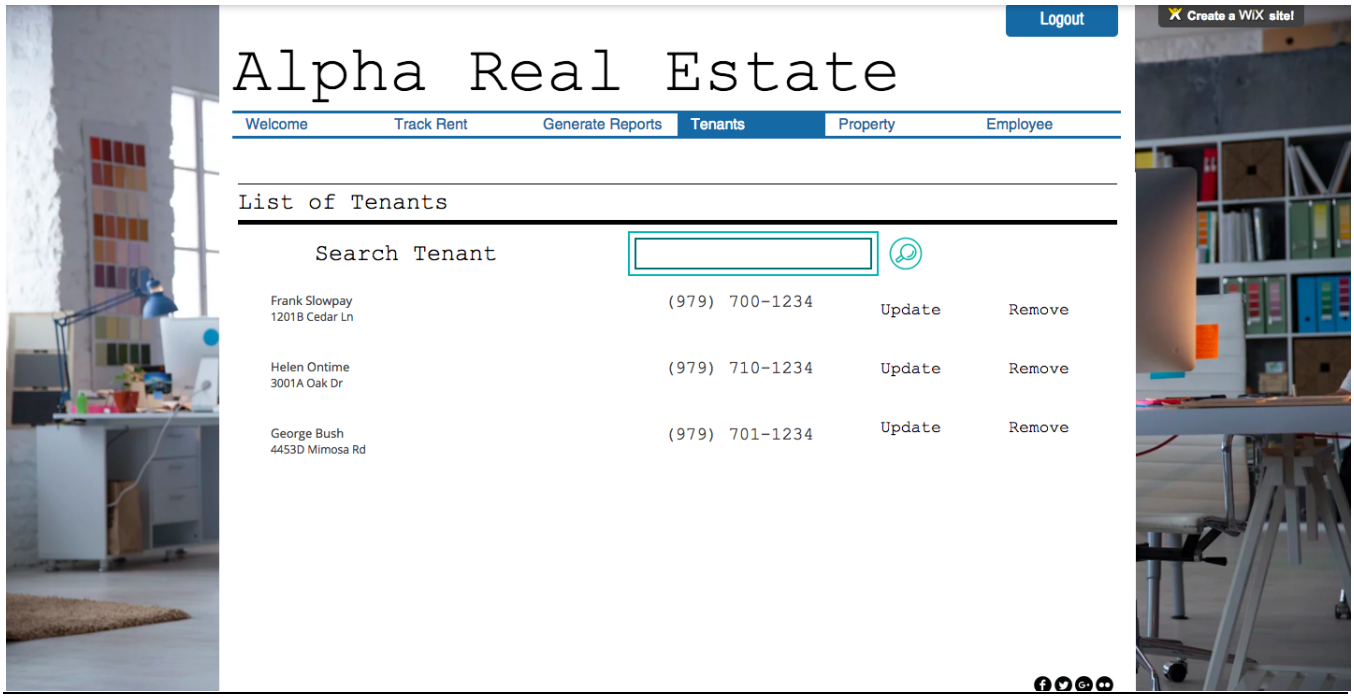
Summary

Inputs	**Source**	**Outputs**	**Destination**
Identifying information	Customer	Update database with records marked as removed and vacant	Tenant datastore, Unit datastore

18.2 Sequence diagram- Remove Tenant



18.3 Screen Mock-up- Remove Tenant



19. Track Rental payment

19.1 Use case- Track rental payment

Use Case Name: Track rental payment	**ID**: UC-14	**Priority:** High
Actor: Customer		
Description: This use case describes how the customer tracks rental payment.		
Trigger: Customer clicks on "Track Rent" **Type:** ☑ **External** ☐ **Temporal**		
Preconditions: **1.** The system is up and available.		
Normal Course: 1.0 The system displays a form where the customer enters the amount of each check from a tenant. 1. The customer clicks on 'Record Rent' option. 2. System displays record rent Screen		**Information for Steps:**

3. The customer enters Tenant Id, Unit No, amount of rent paid, Check no. if payment by check and then clicks on record rent 4. The System saves the rent paid record and generates a payment ID	←Tenant name and Unit number, amount → Payment ID
Alternative Course: N/A	

Post conditions:
The customer has the information of rental payments of all tenants.

Exceptions:
E1. The customer enters an invalid number for a rent amount.

1. The system displays a message “Please enter a valid number field.”
2. The customer enters a valid number field.
3. The system starts Normal Course again
4. The customer asks to exit
5. The system terminates the use case

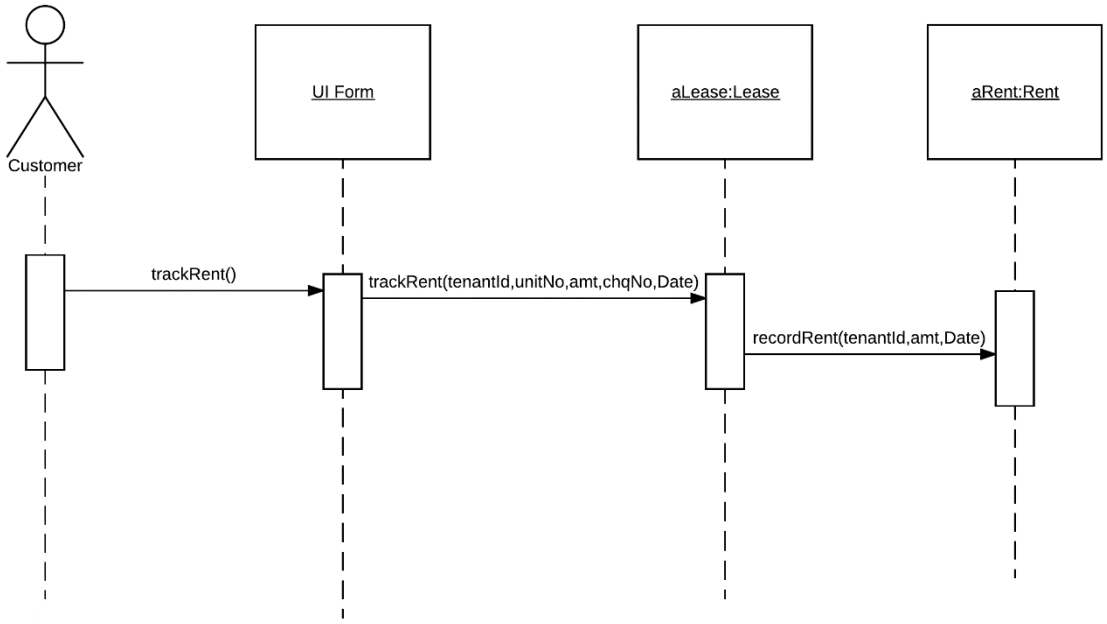
E2. The customer does not enter all the required fields of different amounts on the page.

1. The system displays a message “Please enter all required fields.”
2. The customer enters all the missing fields.
3. The system starts Normal Course again.
4. The customer asks to exit.
5. The system terminates the use case.

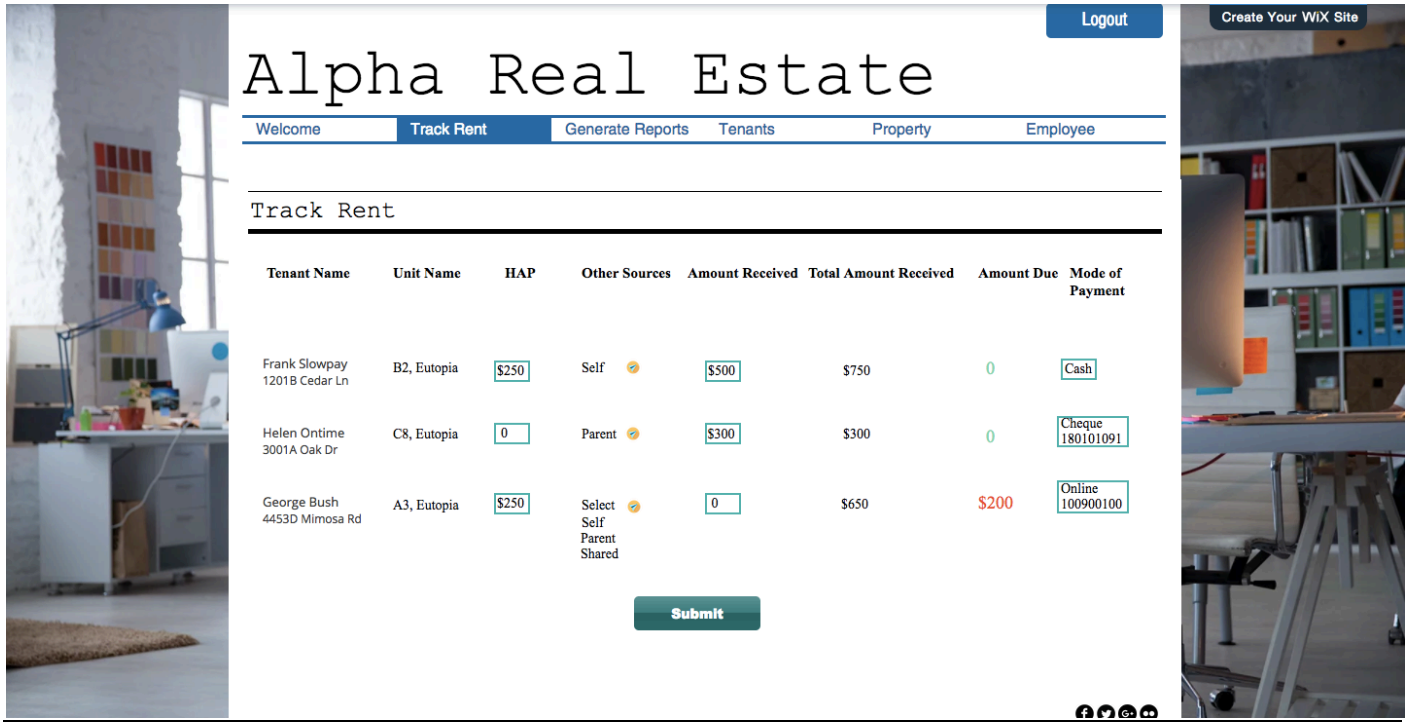
Summary

Inputs	Source	Outputs	Destination
Rent amount	Customer	Rent details	Rent Datastore
Tenant Information	Customer	Tenant details	Tenant Datastore
Unit number	Customer	Unit details	Property/ Unit Datastore

19.2 Sequence diagram- Track rental payment



19.3 Screen Mock-up- Track rental payment



20. Track Unpaid rent

20.1 Use case- Track unpaid rent

<table>
<tr><td>Use Case Name: Track unpaid rent</td><td>ID: UC-15</td><td>Priority: High</td></tr>
<tr><td colspan="3">Actor: Customer</td></tr>
<tr><td colspan="3">Description: This use case describes how the customer tracks unpaid rent.</td></tr>
<tr><td colspan="3">Trigger: Customer clicks on “Track Rent”
Type: ☑ External ☐ Temporal</td></tr>
<tr><td colspan="3">Preconditions:
1. The system is up and available.
2. The tenant and rent information is available.</td></tr>
<tr><td colspan="2">Normal Course:
1.0 The system displays a list of tenants with their outstanding amounts and overdue rent for each tenant.
1. The system displays a list of all the tenants with the actual rent amount paid for a particular month and the expected rent to be paid by the tenant where the actual rent falls short of the expected rent.
2. The customer writes off some amount of unpaid rent for a tenant if their most recent payments are complete and timely.
3. The customer writes off some amount of unpaid rent for a tenant if they did any work on the unit in exchange for rent.
4. The customer enters the reason for the adjustment of rent in the comment field.</td><td>Information for Steps:

→ Tenants List

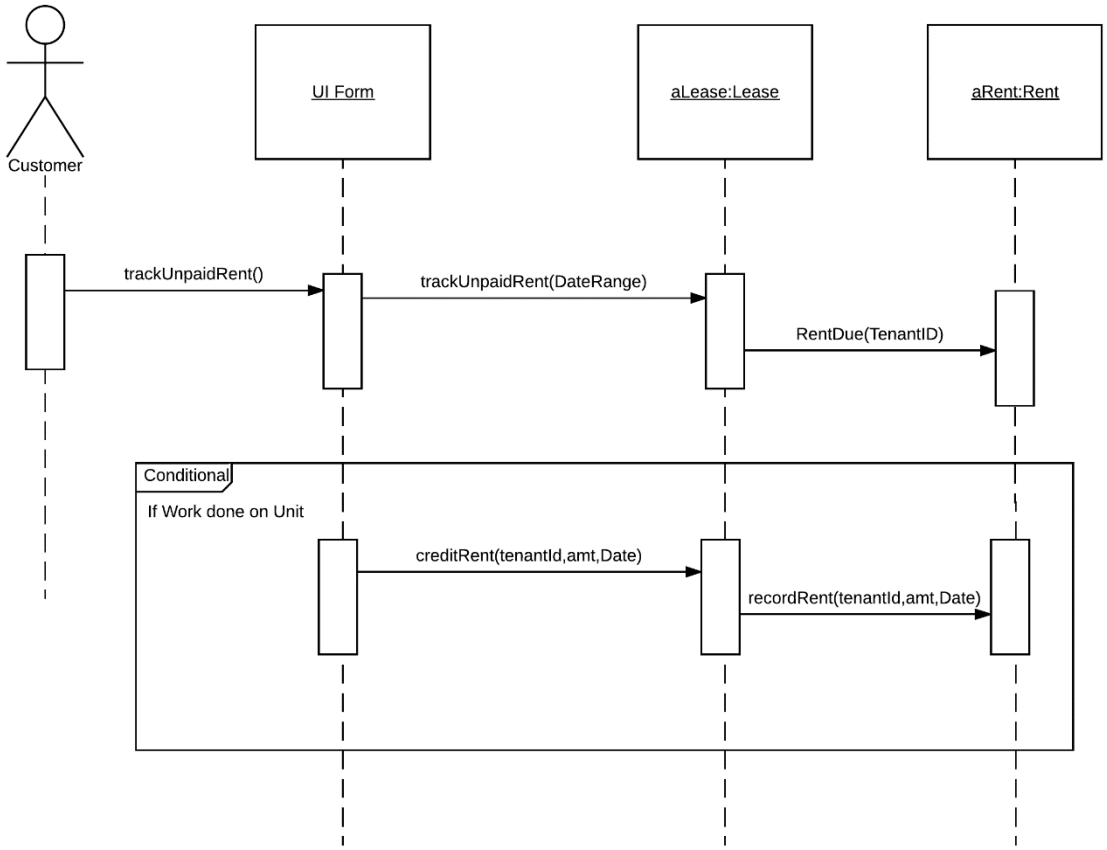
←Adjusted Rent Amount

←Adjusted Rent Amount

← Reason for rent adjustment</td></tr>
<tr><td colspan="2">Alternative Course: N/A</td><td></td></tr>
<tr><td colspan="3">Post conditions:
1. The Customer has a list of all unpaid rentals.</td></tr>
<tr><td colspan="3">Exceptions: N/A</td></tr>
<tr><td colspan="3">Summary</td></tr>
</table>

Inputs	Source	Outputs	Destination
Adjusted Rent amount and Reason for rent adjustment	Customer	List of tenants with unpaid rentals	Rent datastore

20.2 Sequence diagram- Track unpaid rent



21. Track HAP

21.1 Use case- Track HAP

Use Case Name: Track HAP	**ID:** UC-16	**Priority:** High
Actor: Customer		

Description: HAP subsidy details for all relevant customers should be visible to customer

Trigger: Customer wants to check HAP subsidy

Type: ☑ External ☐ Temporal

Preconditions: 1. System is up and running
2. Customer is authenticated

Normal Course:	**Information for Steps:**
1. Customer clicks on Generate Reports 2. Customer selects HAP overview tab 3. Customer inputs month for which she wants to see the report 4. Populated Subsidy amount field for relevant customers is displayed 5. Total rent due is displayed 6. Customer can input subsidy payment against relevant customers 7. Customer can update existing subsidy amount fields	← Month → Updated total rent due → Updated subsidy amount
Alternative Courses: N/A	

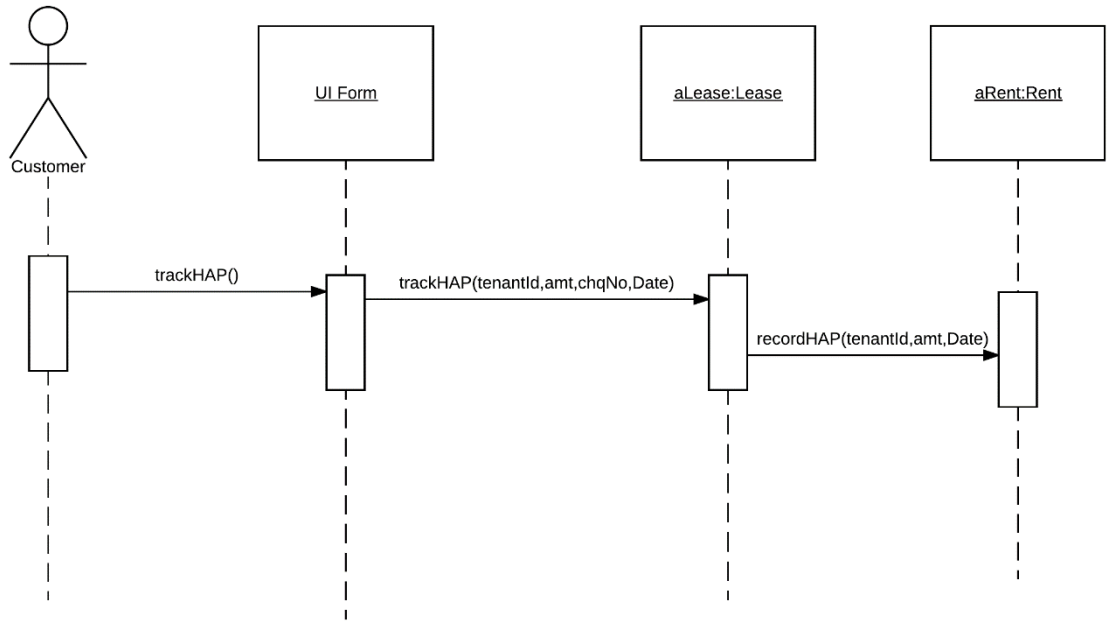
Postconditions: Subsidy rates for customers displayed on screen

Exceptions:
1. If an unforeseen error occurs:
(a.) System displays "Oops. We encountered a problem."
(b.) System prompts the customer to report the issue and exit.

Summary

Inputs	**Source**	**Outputs**	**Destination**
Month	HAP sends subsidy checks to customer	Updated total rent due	Rent datastore

21.2 Sequence Diagram-Track HAP



22. Allow adjustments

22.1 Use case- Allow adjustments

Use Case Name: Allow adjustments	**ID**: UC-17	**Priority**: High
Actor: Customer		
Description: Customer can adjust rents of tenants based on certain conditions		
Trigger: Customer wants to adjust rent **Type:** ☑ **External** ☐ **Temporal**		
Preconditions: 1. System is up and running 2. Customer is authenticated		

<table>
<tr><td>Normal Course:
1. Search tenant’s past due statuses
2. Customer checks if tenant has paid dues 9 out of 10 times
3. If not, customer does not waive off the rent
4. Customer checks if tenant has paid the complete latest due on time
5. If not, customer does not waive off the rent
6. Update rent amount due for tenant</td><td>Information for Steps:

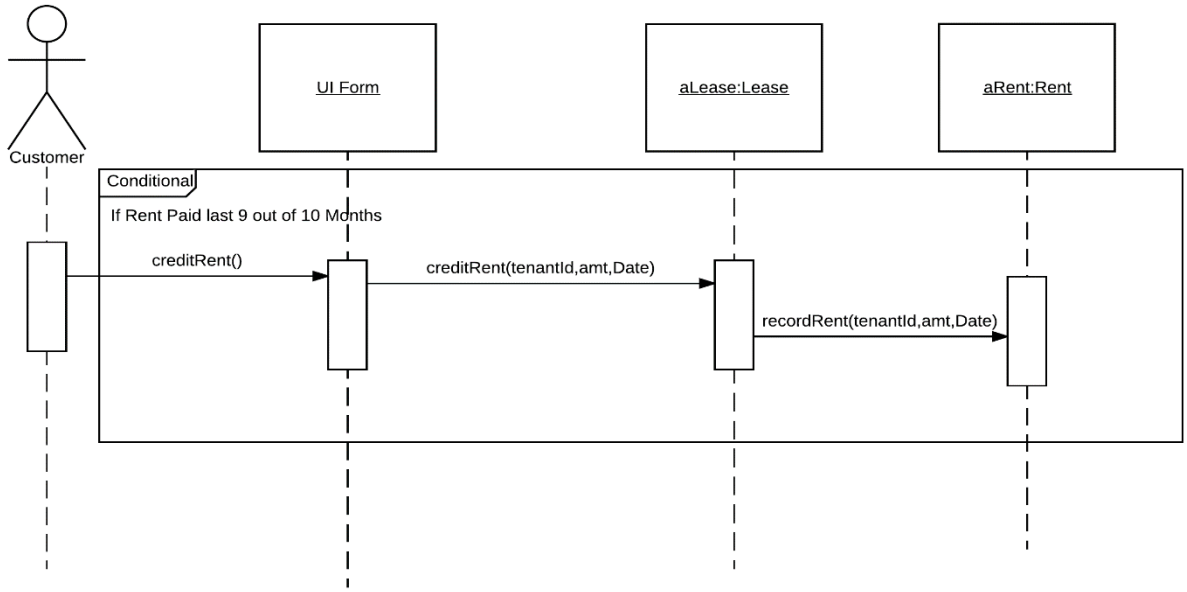
← Tenant Name

← Month

→ Updated rent amount, updated rent status</td></tr>
<tr><td>Alternative Courses:
1. Customer checks if tenant has worked on a unit
2. If not, customer does not waive partial or complete rent</td><td></td></tr>
<tr><td colspan="2">Postconditions:
Tenant’s rent is waived off and status is updated in the system</td></tr>
<tr><td colspan="2">Exceptions:
1. If an unforeseen error occurs:
(a.) System displays “Oops. We encountered a problem.”
(b.) System prompts the customer to report the issue and exits</td></tr>
<tr><td colspan="2">Summary</td></tr>
</table>

Inputs	**Source**	**Outputs**	**Destination**
Tenant name, Month	Rent datastore	Updated rent amount, Updated rent status	Rent datastore

22.2 Sequence Diagram- Allow adjustments



23. Run batch jobs

23.1 Use case- Run batch jobs

Use Case Name: Run Batch Jobs	**ID**: UC-18	**Priority:** High
Actor: System		
Description: System will run regular batch jobs throughout the month providing notifications on 'Rent Due' and amount receivable from each tenant.		
Trigger: **Type: ☐ External ☑ Temporal**		
Preconditions: 1.The System is up and available		
Normal Course: 1.0 System runs usual batch jobs 1.The regularity of batch jobs can be set by customer in the 'Settings' menu 2. As required when the time comes every month the system will generate 'Rent Due' and 'Amount receivable' from each tenant		**Information for Steps:** → Rent Due and Amount receivable from each tenant

3. This information will pop up as a notification in the notification window	
Alternative Course: N/A	

Post conditions: The Rent Due and Amount receivable from each tenant will show up as a notification on the notification window

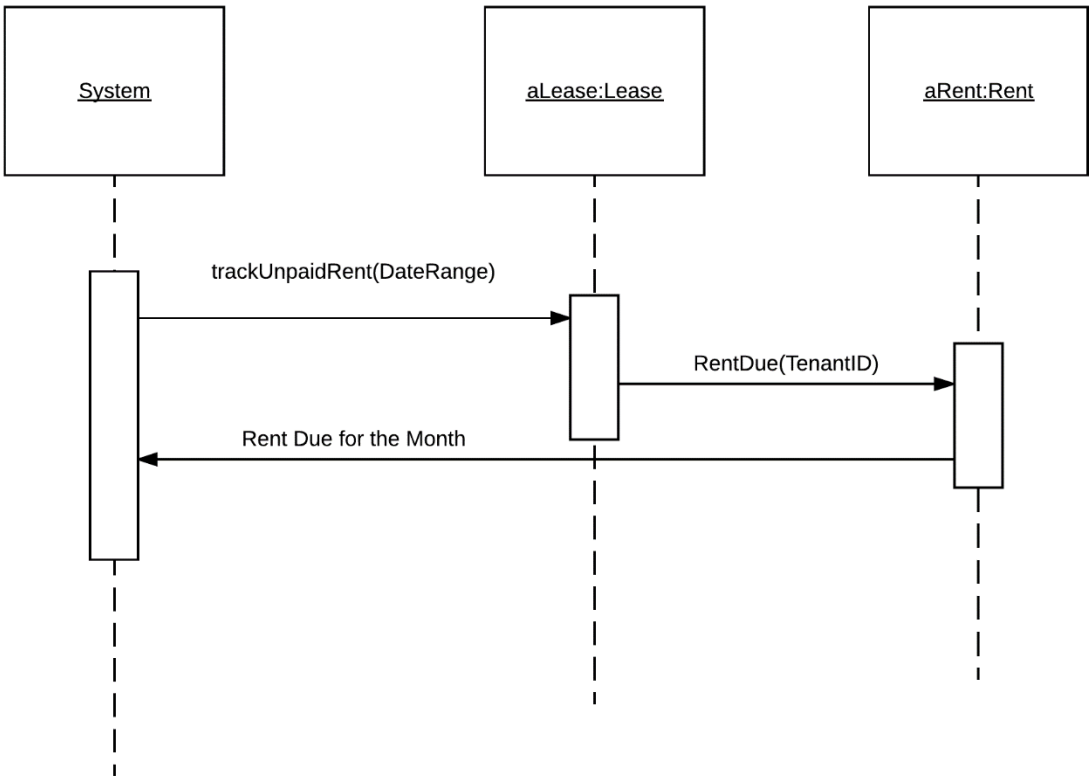
Exceptions: N/A

Summary

Inputs	**Source**	**Outputs**	**Destination**
Rent Information	Rent datastore	Notification of Rent amount due	Notification window
Amount receivable from each tenant	Rent datastore	Amount receivable from each tenant	Notification window

Frequency of Use: Default: Monday of every week. But this can be overwritten by customer in 'Settings' Menu. Then it becomes as set by the customer.

23.2 Sequence diagram- Run batch jobs



24. Generate Payment History Report

24.1 Use case- Generate Payment History Report

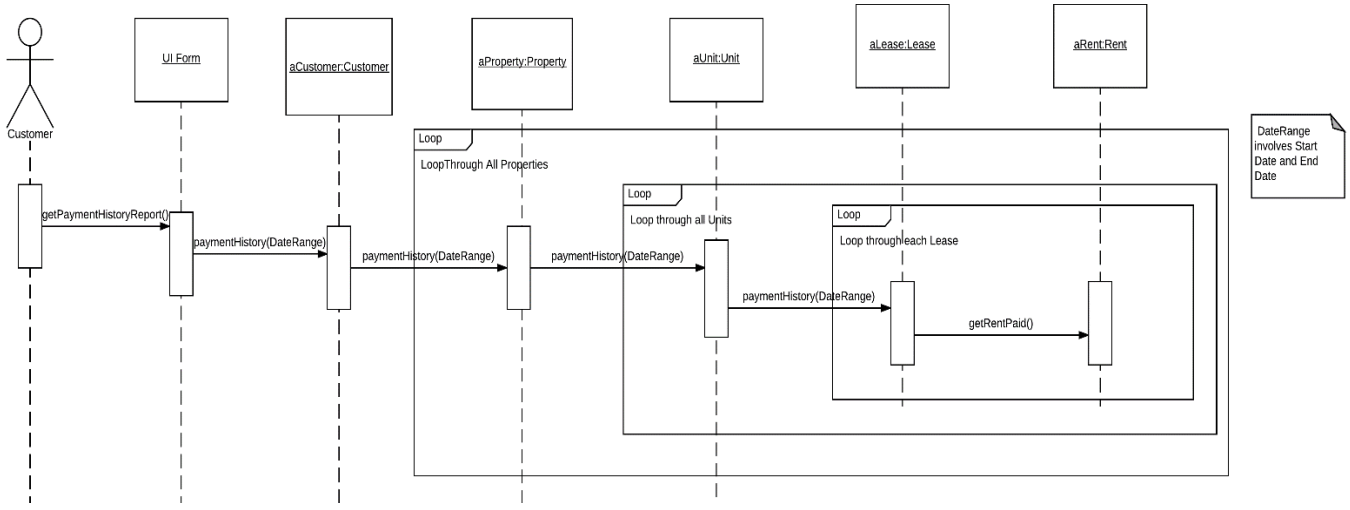
Use Case Name: Generate Payment History Report	**ID**: UC-19	**Priority:** High
Actor: Customer		
Description: This use case describes how the customer generates payment history report by unit, by tenant or by period		
Trigger: Customer clicks on “Generate Payment History Report” **Type:** ☐ **External** ☑ **Temporal**		
Preconditions: **1.** The system is up and available.		
Normal Course: 1.0 The customer clicks on “Generate Payment History Reports”		**Information for Steps:**

1. The system displays a list of various options (Payment History Report by unit, by tenant or by period) to choose from. 2. The customer clicks on “Payment History by unit” Report. 3. The system displays payment history by unit showing rent due, payments made, and balance due. 4. The customer clicks on “Payment History by tenant” Report. 5. The system displays payment history by tenant showing rent due, payments made, and balance due. 6. The customer clicks on “Payment History by period” Report and specifies the period to be a month or a duration of time. 7. The system displays payment history by period (period defined by the customer) showing rent due, payments made, and balance due.	→ Payment History Report by unit → Payment History Report by tenant → Payment History Report by period
Alternative Course: N/A	
Post conditions: The customer has access to payment history report by unit, by tenant or by period.	
Exceptions: N/A	

Summary

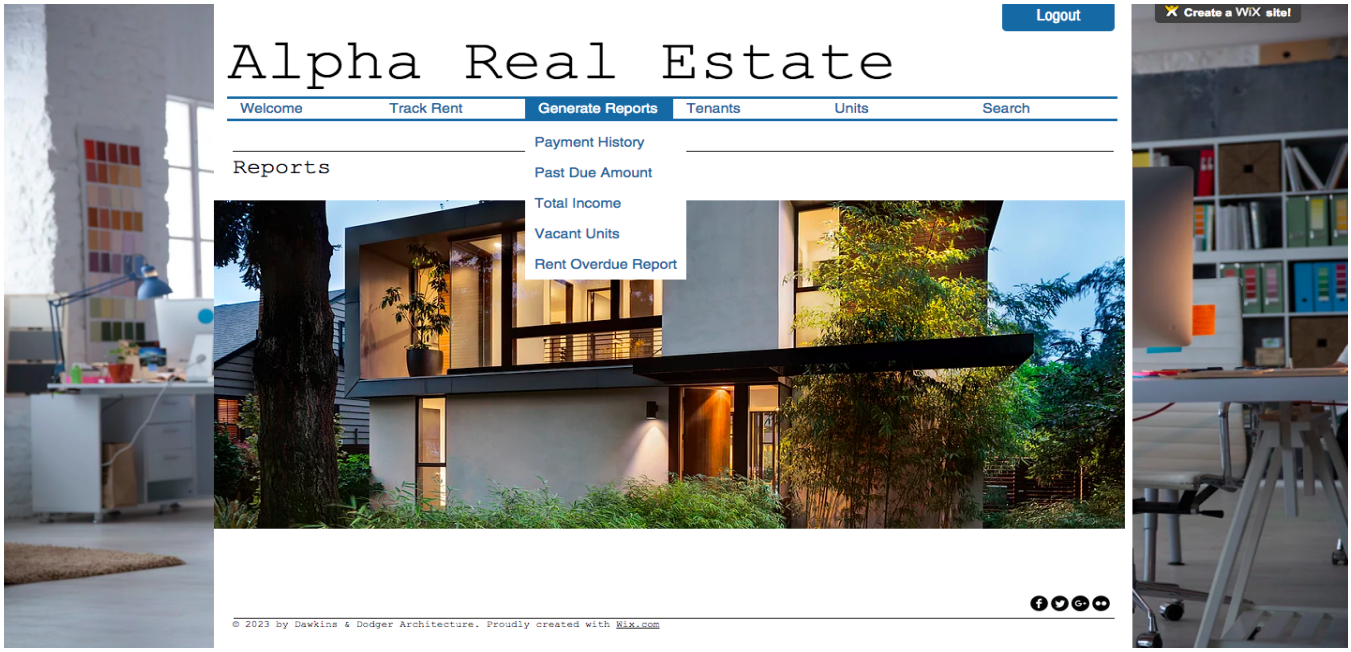
Inputs	**Source**	**Outputs**	**Destination**
Payment History report by unit	Rent Datastore	Payment History by unit	Report Screen
Payment History report by tenant	Rent Datastore	Payment History by tenant	Report Screen
Payment History report by period	Rent Datastore	Payment History by period	Report Screen

24.2 Sequence diagram- Generate Payment History Report



24.3 Screen Mock-up- Generate Payment History Report

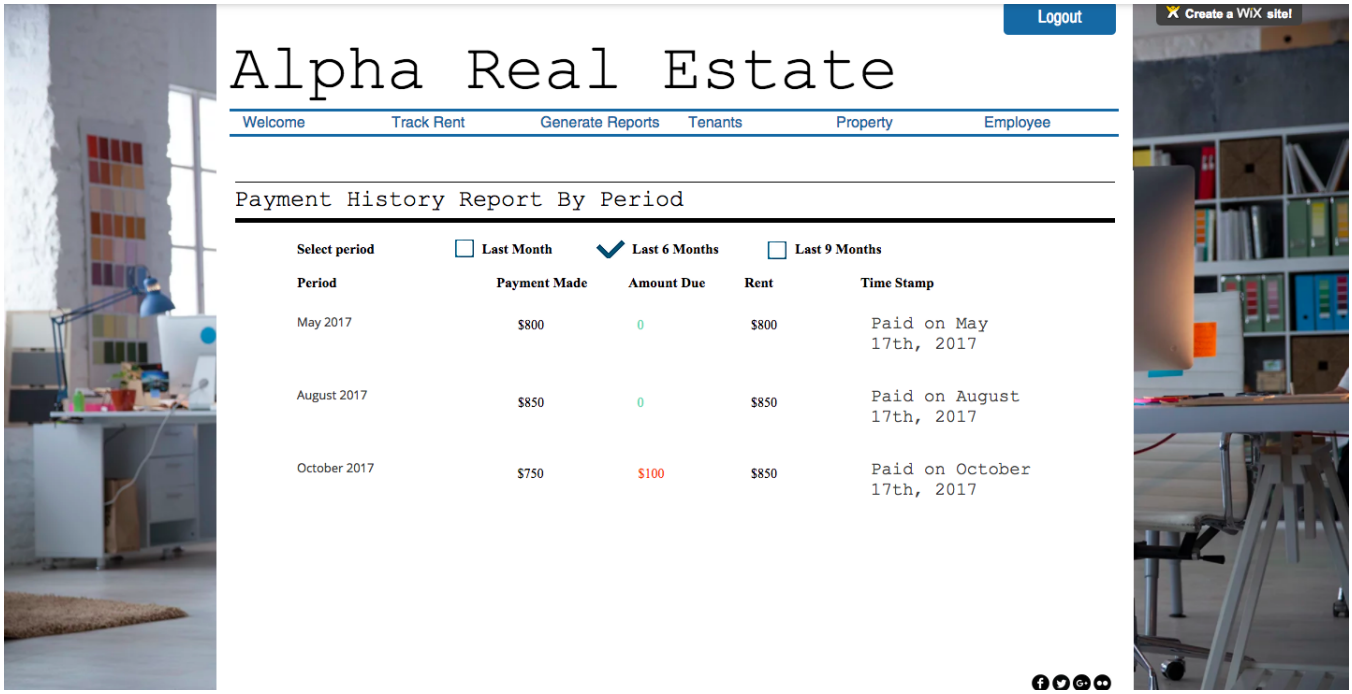
- **Customer clicks on “Generate Reports”**



- **Payment History Report by Unit**

- **Payment History Report by Tenant**

- **Payment History Report by Period**



25. Generate Total Income Report

25.1 Use case- Generate Total Income Report

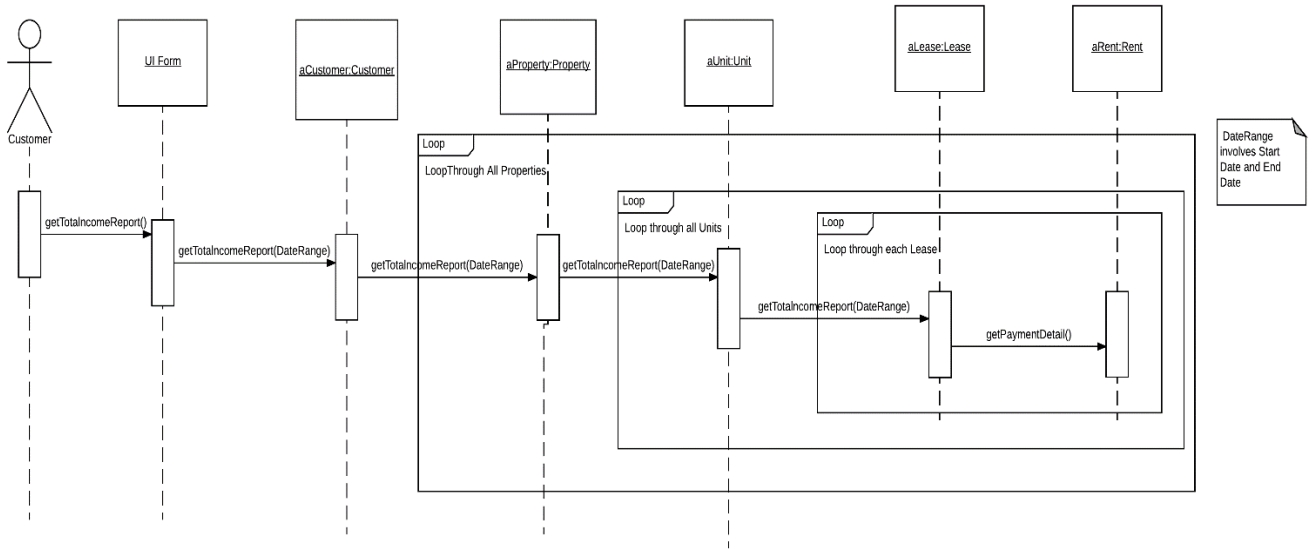
<table>
<tr><td>Use Case Name: Generate Total Income Report</td><td>ID: UC-20</td><td>Priority: High</td></tr>
<tr><td colspan="3">Actor: Customer</td></tr>
<tr><td colspan="3">Description: This use case describes how the customer generates total income report by unit, by tenant or by period</td></tr>
<tr><td colspan="3">Trigger: Customer clicks on “Generate Total Income Report”
Type: ☐ External ☑ Temporal</td></tr>
<tr><td colspan="3">Preconditions:
1. The system is up and available.</td></tr>
<tr><td colspan="2">Normal Course:
1.0 The customer clicks on “Generate Total Income Report”
1. The system displays a list of various options (Total Income Report by unit, by tenant or by period) to choose from.
2. The customer clicks on “Total Income by unit” Report.
3. The system displays Total Income by unit showing rent earned for each unit and the total amount.
4. The customer clicks on “Total Income by tenant” Report.
5. The system displays Total Income by tenant showing rent earned from each tenant and the total amount.
6. The customer clicks on “Total Income by period” Report and specifies the period to be a month or a duration of time.
7. The system displays Total Income by period (period defined by the customer) showing the total rent earned over the specified period of time.</td><td>Information for Steps:

→ Total Income Report by unit

→ Total Income Report by tenant

→ Total Income Report by period</td></tr>
<tr><td colspan="2">Alternative Course: N/A</td><td></td></tr>
<tr><td colspan="3">Post conditions:
The customer has access to total income report by unit, by tenant or by period.</td></tr>
<tr><td colspan="3">Exceptions: N/A</td></tr>
<tr><td colspan="3">Summary</td></tr>
<tr><td>Inputs</td><td>Source</td><td>Outputs</td><td>Destination</td></tr>
<tr><td>Total Income Report by unit</td><td>Rent Datastore</td><td>Total Income by unit</td><td>Report Screen</td></tr>
<tr><td>Total Income Report by tenant</td><td>Rent Datastore</td><td>Total Income by tenant</td><td>Report Screen</td></tr>
<tr><td>Total Income Report by period</td><td>Rent Datastore</td><td>Total Income by period</td><td>Report Screen</td></tr>
</table>

25.2 Sequence diagram- Generate Total Income Report



---

26. Generate Past Due Amount Report

26.1 Use case- Generate Past Due Amount Report

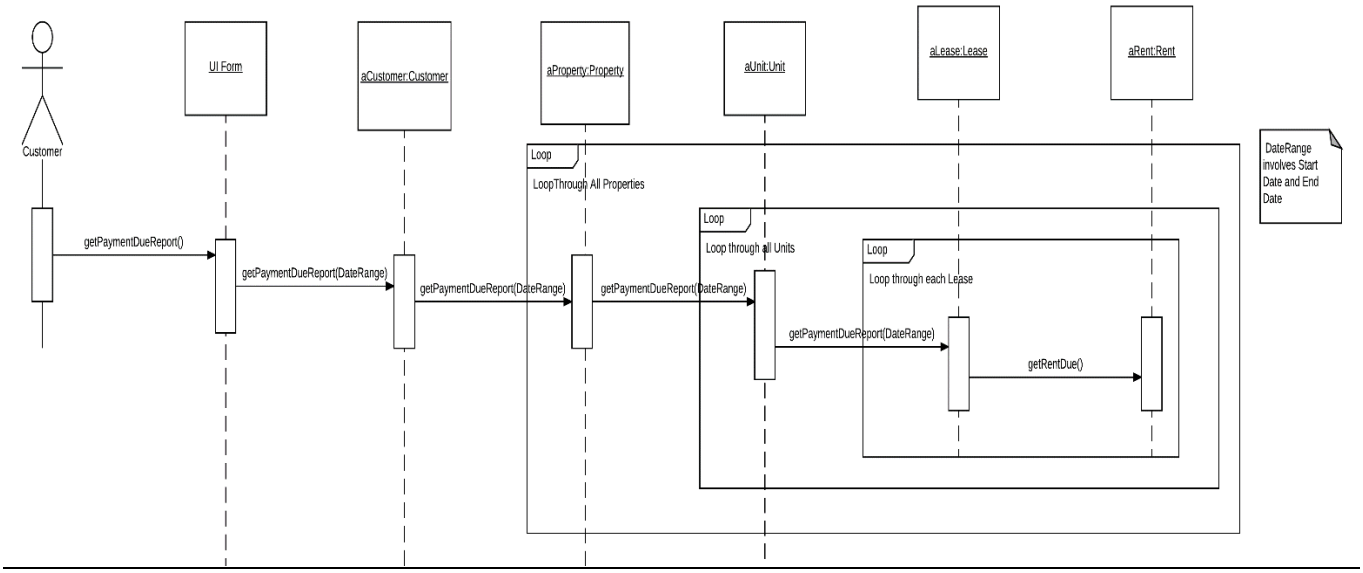
Use Case Name: Generate Past Due Amount Report	**ID**: UC-21	**Priority:** High
Actor: Customer		
Description: This use case describes how the customer generates past due amount report by unit, by tenant or by period		
Trigger: Customer clicks on "Generate Past Due Amount Report" **Type: ☐External ☑ Temporal**		
Preconditions: **1.** The system is up and available.		
Normal Course: 1.0 The customer clicks on "Generate Past Due Amount Report" 1. The system displays a list of various options (Past Due Amount Report by unit, by tenant or by period) to choose from. 2. The customer clicks on "Past Due Amount by unit" Report. 3. The system displays Past Due Amount by unit showing rent due for each unit and the total amount due for all the units. 4. The customer clicks on "Past Due Amount by tenant" Report.		**Information for Steps:** → Past Due Amount Report by unit

5. The system displays Past Due Amount by tenant showing rent due for each tenant and the total amount due for all the tenants. 6. The customer clicks on “Past Due Amount by period” Report and specifies the period to be a month or a duration of time. 7. The system displays Past Due Amount by period (period defined by the customer) showing the total rent due for the specified period.	→ Past Due Amount Report by tenant → Past Due Amount Report by period
Alternative Course: N/A	
Post conditions: The customer has access to Past Due Amount report by unit, by tenant or by period.	
Exceptions: N/A	

Summary

Inputs	**Source**	**Outputs**	**Destination**
Past Due Amount Report by unit	Rent Datastore	Past Due Amount by unit	Report Screen
Past Due Amount Report by tenant	Rent Datastore	Past Due Amount by tenant	Report Screen
Past Due Amount Report by period	Rent Datastore	Past Due Amount by period	Report Screen

26.2 Sequence diagram- Generate Past Due Amount Report



27. Generate Tenant lists report

27.1 Use case- Generate Tenant List Reports

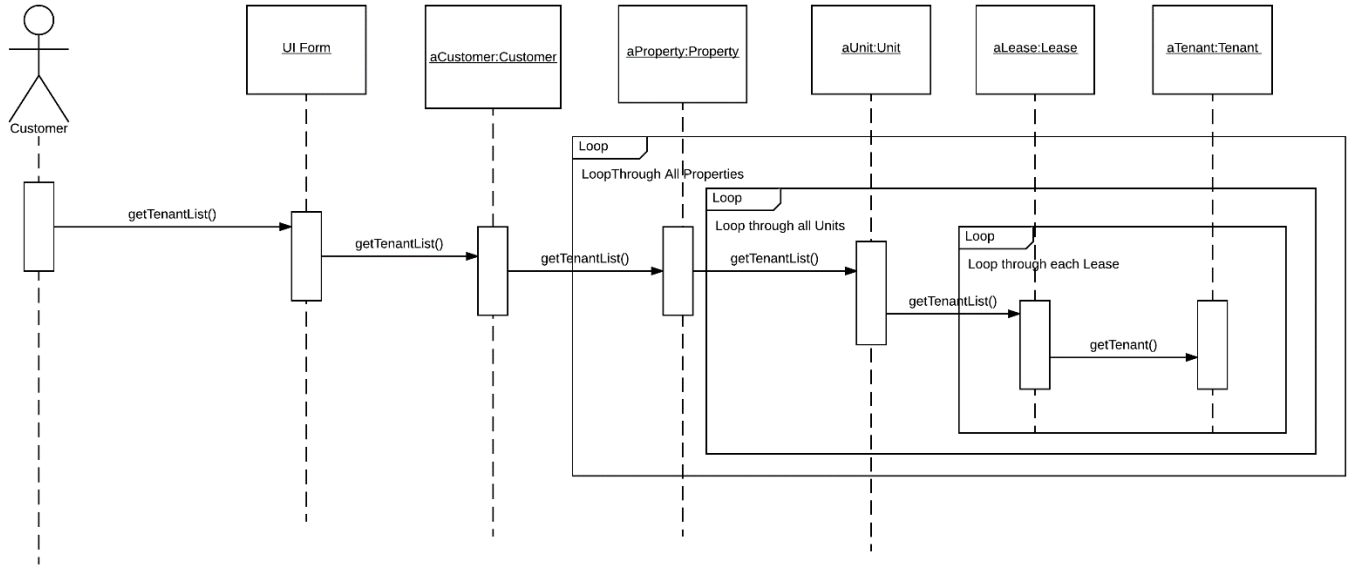
<table>
<tr><td colspan="2">Use Case Name: Generate Tenant List Reports</td><td>ID: UC-22</td><td>Priority: High</td></tr>
<tr><td colspan="4">Actor: Customer</td></tr>
<tr><td colspan="4">Description: This use case describes how the customer generate different tenant lists based on their requirements</td></tr>
<tr><td colspan="4">Trigger: Customer clicks on “Generate Tenant List Reports”
Type: ☐ External ☑ Temporal</td></tr>
<tr><td colspan="4">Preconditions:
1. The system is up and available.</td></tr>
<tr><td colspan="3">Normal Course:
1.0 The customer clicks on “Generate Tenant List Report”
1. The system displays a list of various options to generate a tenant list to choose from.
2. The customer selects the report that lists all the tenants by unit.
3. The customer selects the report that lists all the tenants by unit who are behind, the dollar amount behind, and the date of last payment.
4. The customer selects the report that lists all tenants whose lease is up.</td><td>Information for Steps:

→ Tenants List

→ Tenants List with unpaid rents

→ Tenants List with expired lease</td></tr>
<tr><td colspan="3">Alternative Course: N/A</td><td></td></tr>
<tr><td colspan="4">Post conditions:
The customer has access to track tenants with unpaid rents or expired leases.</td></tr>
<tr><td colspan="4">Exceptions: N/A</td></tr>
<tr><td colspan="4">Summary</td></tr>
<tr><td>Inputs</td><td>Source</td><td>Outputs</td><td>Destination</td></tr>
<tr><td>Tenant list report</td><td>Tenant Datastore</td><td>Tenant list</td><td>Report Screen</td></tr>
<tr><td>Tenant list with unpaid rents report</td><td>Tenant Datastore</td><td>Tenant list with unpaid rents</td><td>Report Screen</td></tr>
<tr><td>Tenant list report with expired lease</td><td>Tenant Datastore</td><td>Tenant list whose lease is up</td><td>Report Screen</td></tr>
</table>

27.2 Sequence Diagram- Generate Tenant List Reports



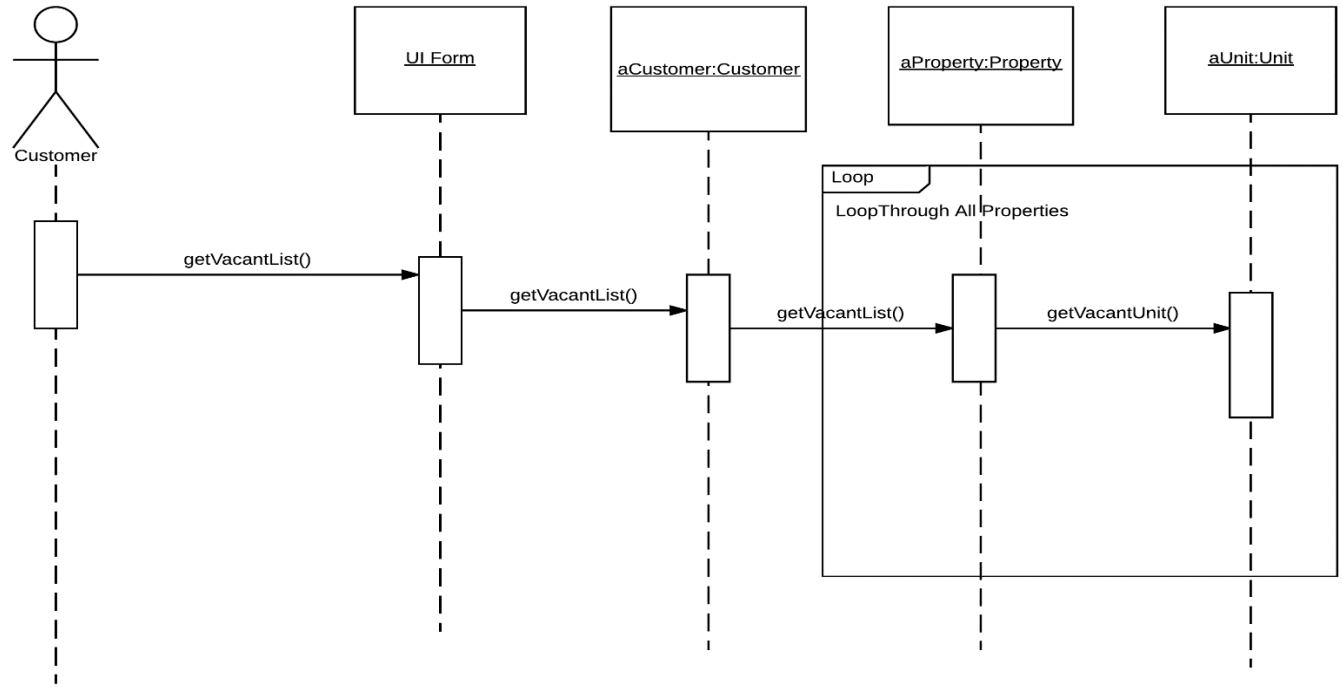
28. Generate Vacant Unit list report

28.1 Use case- Generate Vacant Unit List Reports

Use Case Name: Generate Vacant Unit List Reports	**ID**: UC-23	**Priority:** High
Actor: Customer		
Description: This use case describes how the customer tracks vacant units		
Trigger: Customer clicks on “Generate Vacant Unit List Report” **Type: ☐ External ☑ Temporal**		
Preconditions: **1.** The system is up and available.		
Normal Course: 1.0 The customer clicks on “Generate Vacant Unit List Report” 1.The customer selects the report that lists the current vacancies of all units. 2. The selected report with a list of all vacant units is displayed.		**Information for Steps:** → Vacant Unit List
Alternative Course: N/A		
Post conditions: The customer has access to track all vacant units.		
Exceptions: N/A		

Summary Inputs	Source	Outputs	Destination
Vacant Unit List report	Property/Unit Datastore	Vacant Unit List	Report Screen

28.2 Sequence diagrams- Generate Vacant Unit List Reports



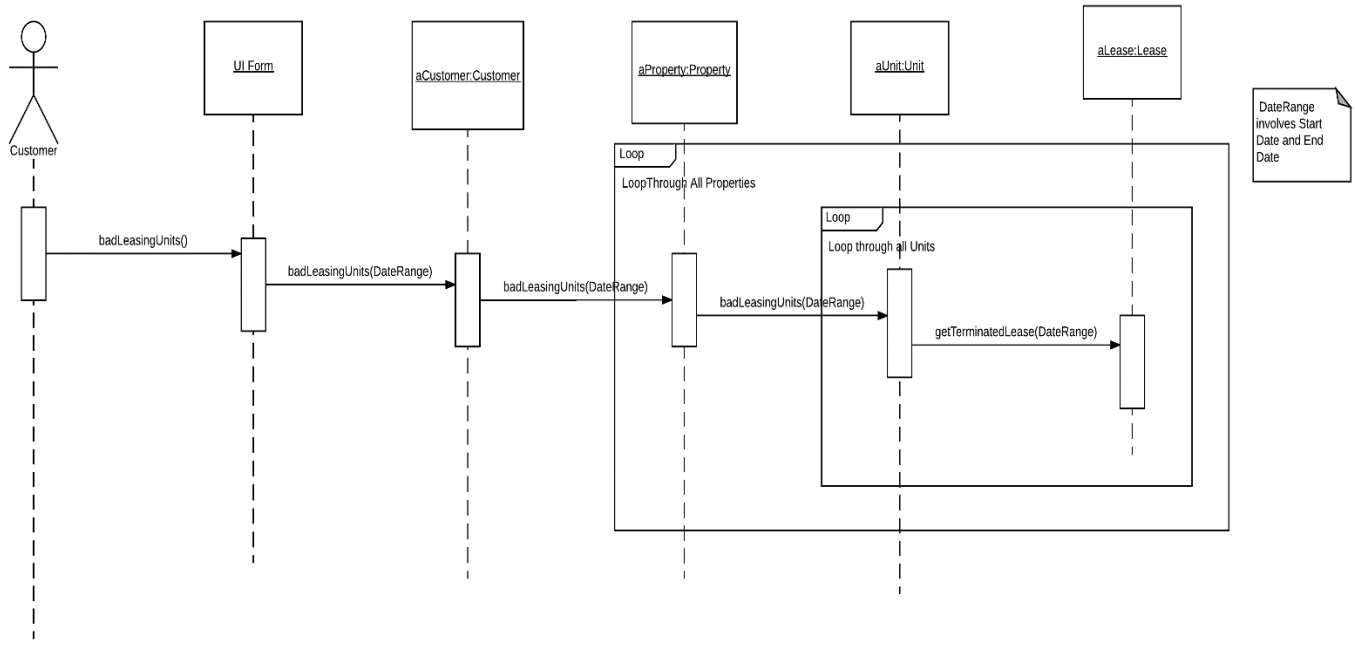
29. Generate Bad Leasing Units Report

29.1 Use case- Generate Bad Leasing Units Report

Use Case Name: Generate Bad Leasing Units Report	**ID**: UC-24	**Priority:** High
Actor: Customer		
Description: This use case describes how the customer tracks units with frequent broken leases		
Trigger: Customer clicks on “Generate Report for bad leasing units” **Type:** ☐ **External** ☑ **Temporal**		

Preconditions: **2.** The system is up and available.			
Normal Course: 1.0 The customer clicks on “Generate Report for bad leasing units” 1.The customer selects the report that lists all units for which lease term was broken (default term for a lease is 1 year) for more than two times in the past. 2. The selected report with a list of all bad leasing units is displayed.			**Information for Steps:** → Bad Leasing Units List
Alternative Course: N/A			
Post conditions: The customer has access to track all units with a history of frequent broken leases.			
Exceptions: N/A			
Summary **Inputs**	**Source**	**Outputs**	**Destination**
Bad Leasing Units List report	Property/Unit Datastore	Bad Leasing Units List	Report Screen

29.2 Sequence diagram- Generate Bad Leasing Units Report



30. Generate Rent Deposit Report

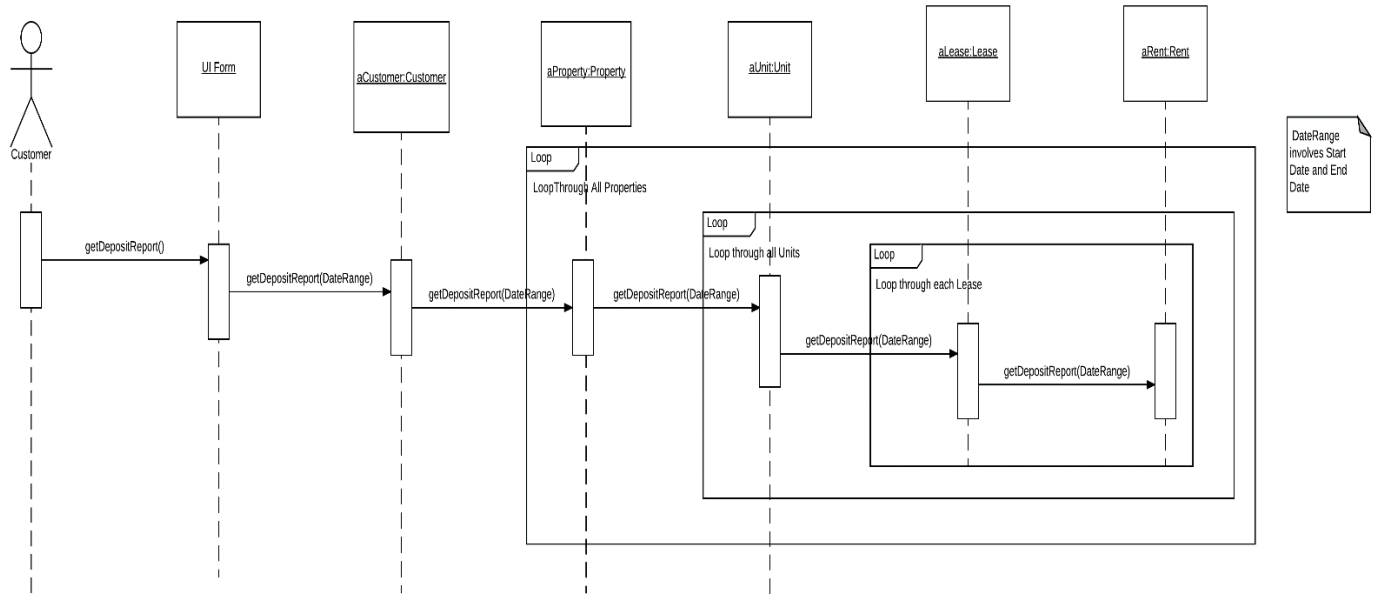
30.1 Use case- Generate Rent Deposit Report

<table>
<tr><td>Use Case Name:
Generate Rent Deposit Report</td><td>ID: UC-25</td><td>Priority: High</td></tr>
<tr><td colspan="3">Actor: Customer</td></tr>
<tr><td colspan="3">Description: This use case describes how the customer generates total rent deposited report.</td></tr>
<tr><td colspan="3">Trigger: Customer clicks on “Generate Rent Deposit Report”
Type: ☑ External ☐ Temporal</td></tr>
<tr><td colspan="3">Preconditions:
1. The system is up and available.
2. Customer is authenticated</td></tr>
<tr><td colspan="2">Normal Course:
1.0 The customer clicks on “Generate Rent Deposit Report”
1. The system displays a screen with tenant details like tenant name, check number, check amount, or cash amount, date of rent received.
2. The system prompts the customer to print the Rent Deposit Report with details showing the total value of deposited amount with tenant name and amount and the date it was deposited.
3. The customers prints the report.</td><td>Information for Steps:

→ Total rent amount deposited report</td></tr>
<tr><td colspan="2">Alternative Course: N/A</td><td></td></tr>
<tr><td colspan="3">Post conditions:
The customer can generate the total amount of rent deposited</td></tr>
</table>

Exceptions: N/A			
Summary			
Inputs	**Source**	**Outputs**	**Destination**
Rent Deposit report	Tenant Datastore, Rent Datastore	Total rent amount Deposited	Report Screen

30.2 Sequence Diagram- Generate Rent Deposit Report



31. View Tenant Payment History

31.1 Use case- View Tenant Payment History

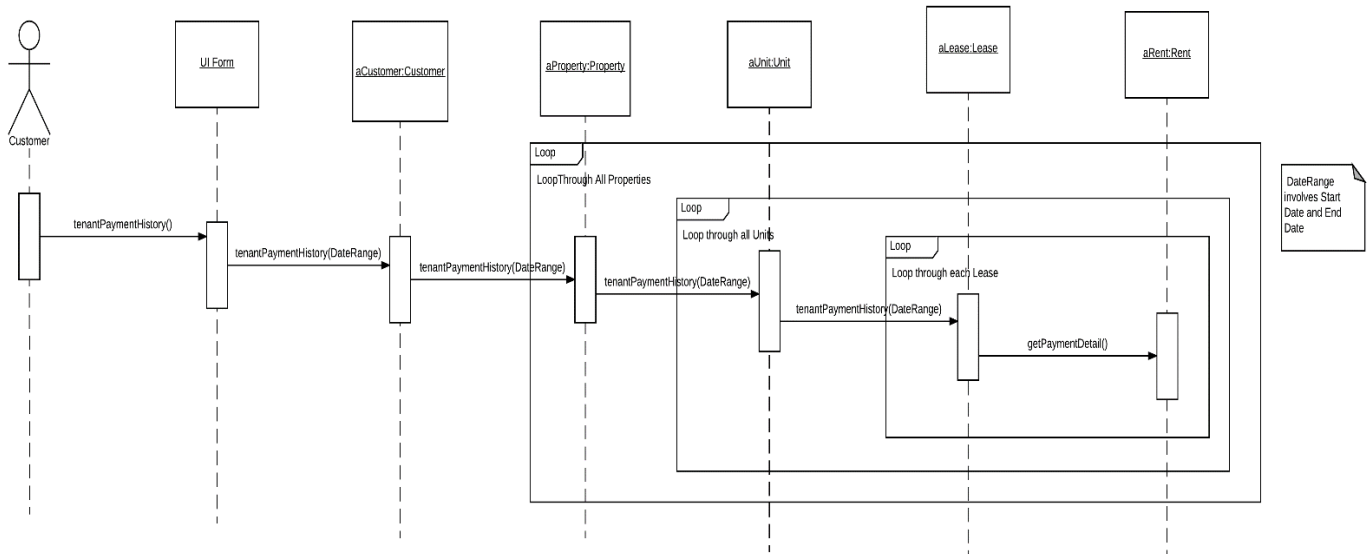
Use Case Name:	Tenant payment History Details	**ID:** UC-26	**Priority:**	High

Actor: Tenant	
Description: Tenant can see the details of all the payments made by him for the unit	
Trigger: Tenant wants to check payment history **Type**: ☑ External ☐ **Temporal**	
Preconditions: 1. System is up and running 2. Tenant is authenticated	
Normal Course: 1. Tenant clicks on View Payment History tab 2. Tenant can select the range for which he wants the records 3. Details of all the payments made displayed on screen	**Information for Steps:** ← Date Range → Tenant Payment Details
Alternative Courses: N/A	
Postconditions: Payment details provided to tenant.	
Exceptions: 1. If an unforeseen error occurs: (a.) System displays "Oops. We encountered a problem." (b.) System prompts the tenant to report the issue and exits.	

Summary

Inputs	Source	Outputs	Destination
Date Range	Tenant datastore	Tenant payment details	Tenant Payment History Screen

31.2 Sequence diagram- View Tenant Payment History



32. Make Payment by Tenant

32.1 Use case- Make Payment

Use Case Name: Make payment	**ID:** UC-27	**Priority:** High
Actor: Tenant		
Description: This use case describes how the tenant will make the payment.		
Trigger: Tenant wants to make a payment. **Type**: ☑ **External** ☐ **Temporal**		
Preconditions: 1. System is up and running. 2. Tenant is logged in and authenticated.		
Normal Course: 1.0 The tenant logs in to make a payment for the current month or previous dues. 1. Tenant clicks on the make payment button 2. The application redirects to a web-form where the tenant enters his last name, phone number and apartment number.	**Information for Steps:** ←Tenant name, phone and apartment number	

3. The application pulls up the tenant's latest rental statement which lists his total rent. 4. The tenant verifies the statement and can pay his dues using the 'Pay with PayPal' button at the bottom of the screen. 5. The rent history page lists all the tenant's previous rent transactions and allows him to download pdf rent statements.	→ Latest rental statement
Alternative Courses: N/A	

Postconditions:

1. The tenant has successfully made the rent payment and has received a confirmation.
2. The payment has been recorded into the database.

Exceptions:

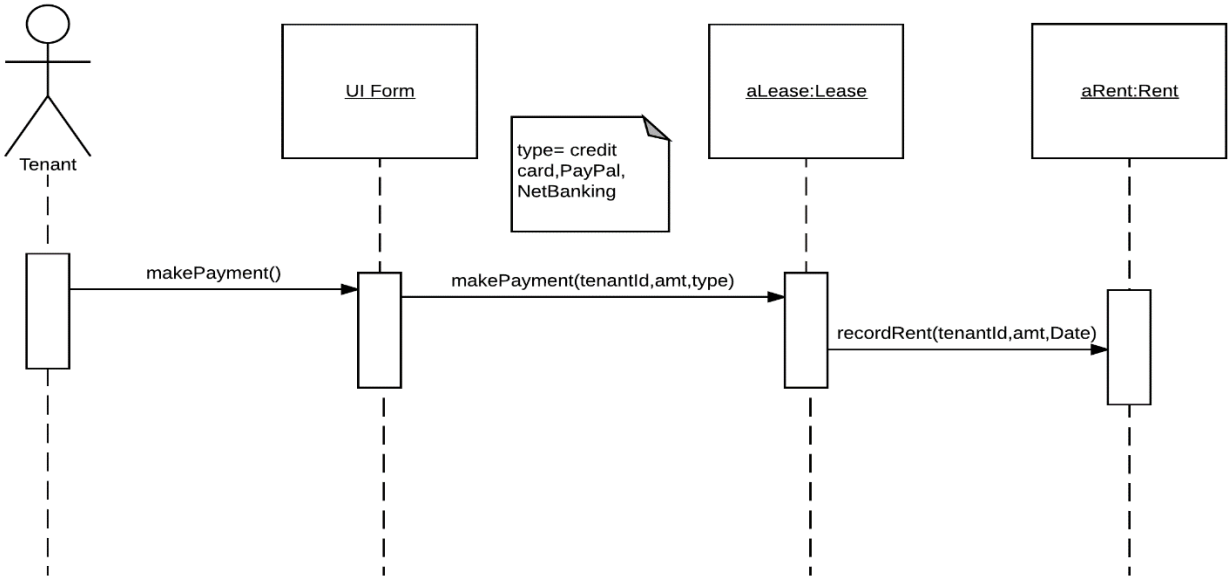
E1. If the payment method fails:

1. System displays "The transaction has failed."
2. System asks the tenant
3. System prompts the tenant to report the issue and exit.

Summary

Inputs	Source	Outputs	Destination
Tenant name, phone and apartment number	Tenant- Payer	Latest rental statement	Rent and Tenant datastore

32.2 Sequence Diagram- Make Payment



33. Add Lease

33.1 Use case- Add Lease

Use Case Name: Add Lease	**ID**: UC-28	**Priority:** High
Actor: Customer		
Description: This use case describes how the customer adds a lease when the tenant rents a unit from the customer		
Trigger: Customer clicks on “Add Lease” **Type: ☑ External ☐ Temporal**		
Preconditions: 1. The system is up and available. 2. Customer is authenticated.		
Normal Course: 1.0 The system displays Add Lease. 1. The customer clicks on Add New Lease for a new tenant. 2. The system displays a lease template form for “New Lease”.		**Information for Steps:**

3. The system prompts the customer to enter a lease start date and end date, new tenant details which can be more than one, to add to the lease- tenant name(s), leasing unit, phone number, email, information of two guarantors, rent and deposit amount. 4. The system prompts the customer to make any new modifications to the existing lease template if required. 5. The customer clicks on “Create New Lease”. 6. The system saves the entered information from the form into the relevant tables- Lease, Tenant and Customer with a LeaseID and returns the LeaseID to the customer. 7. The system displays an option to either print or email the lease to the tenant for their signature.	← Lease start date, end date Tenant name(s), leasing unit, phone number, email, guarantor’s information, rent, and deposit amount → LeaseID
Alternative Course: N/A	

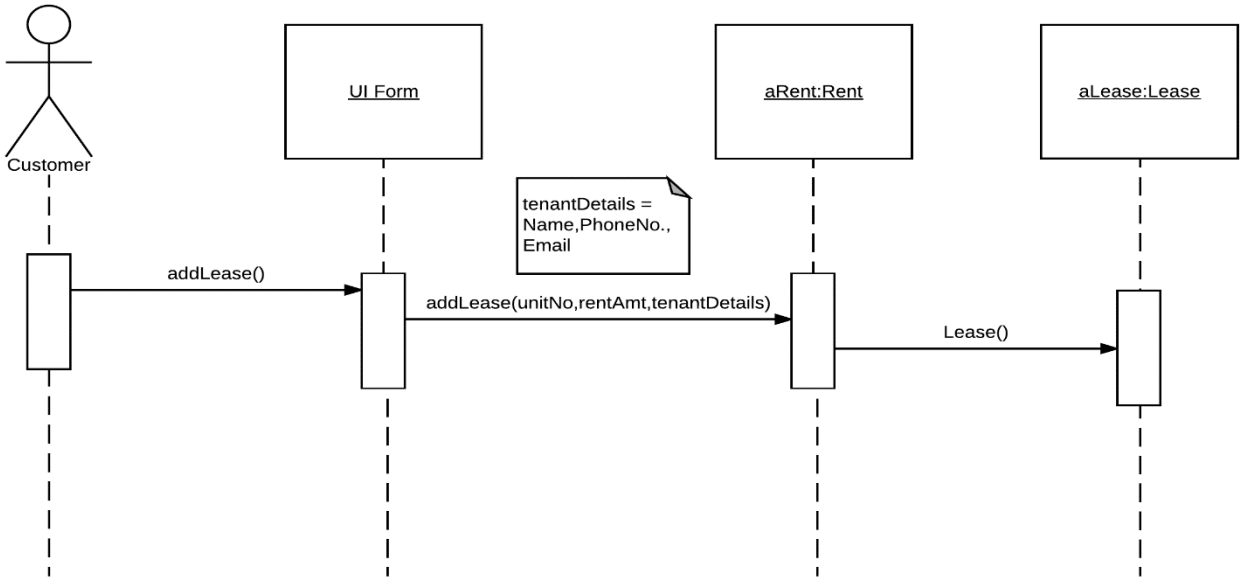
Post conditions:
The customer has created a new lease for a tenant.

Exceptions:
N/A

Summary

Inputs	Source	Outputs	Destination
Lease start date, end date, Tenant name(s), leasing unit, phone number, email, guarantor’s information, rent, and deposit amount	Customer	LeaseID	Lease Datastore Tenant Datastore

33.2 Sequence Diagram- Add Lease



34. Update Lease

34.1 Use case- Update Lease

Use Case Name: Update Lease	**ID**: UC-29	**Priority:** High
Actor: Customer		
Description: This use case describes how the customer updates a lease as needed.		
Trigger: Customer clicks on “Update Lease” **Type: ☑ External ☐ Temporal**		
Preconditions: 3. The system is up and available. 4. Customer is authenticated.		
Normal Course: The system displays Update Lease. 1. The customer clicks on Update Lease for an existing tenant. 2. The system displays a lease template form for “Update Lease”.		**Information for Steps:**

3. The system prompts the customer to enter the LeaseID for the lease that is to be modified or updated. 4. The system prompts the customer to update either the tenant details, add a new tenant, remove an existing tenant, update rent amount, change lease start date and end date, rent or deposit amount. 5. The customer updates fields as required. 6. The system saves the updated information from the form into the relevant tables- Lease, Tenant and Customer. 7. The system displays an option to either print or email the lease to the tenant for their signature	← LeaseID ← Lease start date, end date Tenant name(s), leasing unit, phone number, email, guarantor's information, rent, or deposit amount
Alternative Course: N/A	

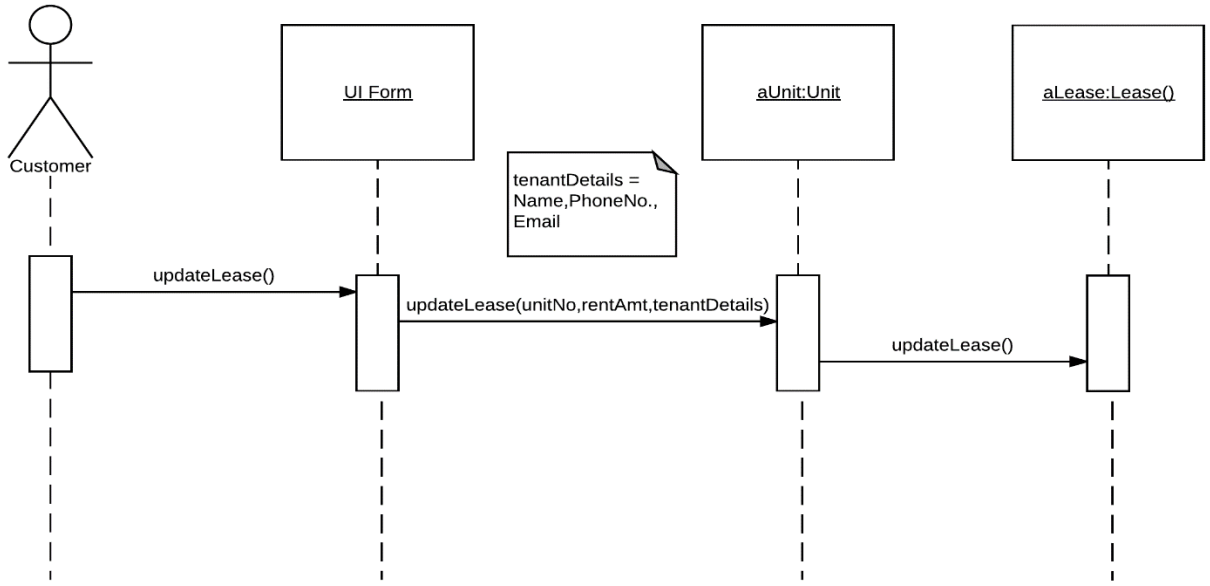
Post conditions:
The customer has made changes to the lease as needed.

Exceptions:
N/A

Summary

Inputs	Source	Outputs	Destination
Lease start date, end date, Tenant name(s), leasing unit, phone number, email, guarantor's information, rent, and deposit amount	Customer	LeaseID	Lease Datastore Tenant Datastore

34.2 Sequence Diagram - Update Lease



35. Testing Plan

35.1 Test plan for Functional Requirements

35.1.1 Property and units

Requirement 35.1.1.1 The system will allow the customer to add new properties to the database.

- **Black box testing** will be done by providing both correct and erroneous input data for property such as the property name, address of the property and the unit number available within the property, and the output will be evaluated.
- **Integration testing** will be done with respect to Customer and Property class to see if the property has been added successfully.
- **White box testing** will be done by looking into the logic implemented to add the property to the database.
- **Frequency of testing**: Monthly

Requirement 35.1.1.2 The system will allow the customer to add new units to the database

- **Black box testing** will be done by providing both correct and erroneous input data for unit such as the unit name, address of the unit, and the output will be evaluated.

- **Integration testing** will be done with respect to Property and Unit class to see if the unit within the property has been added successfully.
- **White box testing** will be done by looking into the logic implemented to add the unit to the database.
- **Frequency of testing**: Monthly

Requirement 35.1.1.3 The system will allow the customer to make changes to unit or property information stored in the database.

- **Black box testing** will be done by providing both correct and erroneous input data for modifying property and units within the property such as the available units within the property, and the output will be evaluated.
- **Integration testing** will be done with respect to Customer and Property class to see if the changes to the property has been updated successfully. Integration testing will be done with respect to Property and Unit class to see if the changes to the unit within the property has been updated successfully.
- **White box testing** will be done by looking into the logic implemented to update the property or unit to the database.
- **Frequency of testing**: Monthly

35.1.2 Tenants and Lease

Requirement 35.1.2.1 The system will allow the customer to enter details about the tenant-name, phone number (up to 3) and deposit amount paid.

- **Black box testing** will be done by providing both correct and erroneous input data for tenant such as tenant name, phone number and deposit amount paid, and the output will be evaluated.
- **Integration testing** will be done with respect to Lease and Tenant class to see if tenant has been added successfully.
- **White box testing** will be done by looking into the logic implemented to add the tenant details to the database.
- **Frequency of testing**: Monthly

Requirement 35.1.2.2 The system will allow the customer to make changes to the lease.

- **Black box testing** will be done by providing both correct and erroneous data for the lease such as lease start date, end date, unit leased out, and the output will be evaluated.
- **Integration testing** will be done with respect to Lease and Rent class while adding a new lease; and Unit and Lease class for updating an existing lease, to validate successful addition or changes made to a lease.
- **White box testing** will be done by looking into the logic implemented to add or update a lease in the database.
- **Frequency of testing**: Monthly

Requirement 35.1.2.3 The system will notify the customer to refund deposit paid by a tenant when they leave.

- **Black box testing** will be done by providing both correct and erroneous data for refunding the deposit amount while removing a tenant and the output will be evaluated to see if the right refund amount is produced.
- **Integration testing** will be done with respect to Lease, and Tenant class to validate while removing a tenant when they leave.
- **White box testing** will be done by looking into the logic implemented to remove a tenant and refund the deposit amount paid.
- **Frequency of testing**: Monthly

Requirement 35.1.2.4 The system allows the tenant to view their rent payment history and make payments.

- **Black box testing** will be done by choosing to view the rent payment history by entering both correct and erroneous data for the period filter and making payments, and the output will be evaluated to validate whether the data being retrieved is correct or not.
- **Integration testing** will be done with respect to Customer, Property, Unit, Lease and Rent class to validate successful retrieval of rent payment history**.** Integration testing will be done with respect to Lease and Rent class to validate successful payment of rent by tenant.
- **White box testing** will be done by looking into the logic implemented to retrieve rent payment history and make rent payments.
- **Frequency of testing**: Monthly

35.1.3 Rent Tracking Management

Requirement 35.1.3.1 The system should display a form where the customer enters the amount of each check from a tenant.

- **Black box testing** will be done by providing both correct and erroneous data by entering each check amount paid by a tenant for a unit for a specific month.
- **Integration testing** will be done with respect to Lease and Rent class to see if the rent paid by the tenant has been recorded successfully.
- **White box testing** will be done by looking into the logic implemented to store and update the rent payment information.
- **Frequency of testing**: Monthly

Requirement 35.1.3.2 The system will allow the customer to record checks from tenants in a batch mode at various times over the course of a month.

- **Integration testing** will be done with respect to Lease and Rent class to see if the rent paid by the tenant has been recorded successfully in a batch mode.
- **White box testing** will be done by looking into the logic implemented to run a batch job to record all the rents paid by a tenant over the month and store and update the rent payment information in the database.

- **Frequency of testing**: Weekly

Requirement 35.1.3.3 The system will allow the customer to run the batch process multiple times per month, as checks arrive.

- **Integration testing** will be done with respect to Lease and Rent class to see if the rent paid by the tenant has been recorded successfully in a batch mode run over multiple times in a month.
- **White box testing** will be done by looking into the logic implemented to run a batch job multiple times in a month to record all the rents paid by a tenant and store and update the rent payment information in the database.
- **Frequency of testing**: Weekly

Requirement 35.1.3.4 The system will record the last amount received via subsidy, and will allow the customer to change that during any monthly keying of subsidy payments.

- **Black box testing** will be done by providing both correct and erroneous data by entering the rent amount received via subsidy and making changes to it over the month, and the output will be evaluated.
- Integration testing will be done with respect to Lease and Rent class to ensure successful changes in the database.
- White box testing will be done by looking into the logic implemented to store and update the subsidy rent payments.
- **Frequency of testing**: Monthly

Requirement 35.1.3.5 The system will provide for multiple sources of payments for rent:
Requirement 35.1.3.5.1 The system will allow the customer to record checks from tenants.

- **Black box testing** will be done by providing both correct and erroneous data by entering each check amount paid by a tenant for a unit for a specific month.
- **Integration testing** will be done with respect to Lease and Rent class to see if the rent paid by the tenant has been recorded successfully.
- **White box testing** will be done by looking into the logic implemented to record all the rents paid by a tenant over the month and store and update the rent payment information in the database.
- **Frequency of testing**: Monthly

Requirement 35.1.3.5.2 The system will allow the customer to record payments from the federal government Housing Assistance Program (HAP). The system will then cycle through the tenants that have subsidy payments due and will pre-populate subsidy amount fields where possible.

- **Black box testing** will be done by providing both correct and erroneous data for recording the payment covered by HAP for each tenant.
- **Integration testing** will be done with respect to Lease and Rent class to see if the rent payment covered by HAP has been recorded successfully.
- **White box testing** will be done by looking into the logic implemented to record the rent amount covered by HAP for a tenant and store and update the rent payment information in the database.

- **Frequency of testing**: Monthly

Requirement 35.1.3.5.3 The system will allow the customer to record payments from tenant's parents.

- **Black box testing** will be done by providing both correct and erroneous data by entering each check amount paid by a tenant's parent for a unit for a specific month.
- **Integration testing** will be done with respect to Lease and Rent class to see if the rent paid by the tenant's parent has been recorded successfully.
- **White box testing** will be done by looking into the logic implemented to record all the rents paid by a tenant's parent over the month and store and update the rent payment information in the database.
- **Frequency of testing**: Monthly

Requirement 35.1.3.5.4 The system will allow the customer to record payments in parts from tenants.

- **Black box testing** will be done by providing both correct and erroneous data by entering each check amount paid by a tenant in parts for a unit for a specific month.
- **Integration testing** will be done with respect to Lease and Rent class to see if the rent paid by a tenant in parts has been recorded successfully.
- **White box testing** will be done by looking into the logic implemented to record all the rents paid by a tenant in parts over a month and store and update the rent payment information in the database.
- **Frequency of testing**: Monthly

Requirement 35.1.3.6 The system will display a list of all the tenants with the actual rent amount received (adding all sources for payment) for a particular month and the expected rent to be paid by the tenant for the month.

- **Black box testing** will be done by validating the actual rent received and expected rent to be received, and the output will be evaluated.
- **Integration testing** will be done with respect to Lease and Rent class to ensure correct values of rent is being retrieved.
- **White box testing** will be done by looking into the logic implemented to retrieve the actual rent received and the expected rent to be received for a month from a tenant for a unit.
- **Frequency of testing**: Monthly

35.1.4 Unpaid Rent Management

Requirement 35.1.4.1 The system will display outstanding amounts due on the first of the month, as well as overdue rent from a previous month.

- **Black box testing** will be done by choosing to view the rent amount due for a unit, to validate whether the outstanding amount is being calculated and retrieved successfully.

- **Integration testing** will be done with respect to Lease and Rent class to see if the rent is retrieved successfully for a specific period and the overdue amount is being calculated correctly.
- **White box testing** will be done by looking into the logic implemented to retrieve the rent for a specific period.
- **Frequency of testing:** Weekly

Requirement 35.1.4.2 The system will allow customer to write off some amount of unpaid rent for a tenant if their most recent payments are complete and timely.

- **Black box testing** will be done by providing both correct and erroneous data while writing off unpaid rent provided timely and complete payments by a tenant.
- **Integration testing** will be done with respect to Lease and Rent class to see if write off is done only when specific conditions are met.
- **White box testing** will be done by looking into the logic implemented to write off unpaid rent only if recent payments by a tenant are timely and complete.
- **Frequency of testing:** Monthly

Requirement 35.1.4.3 The system will allow customer to write off some amount of unpaid rent for a tenant if they did any work on the unit in exchange for rent.

- **Black box testing** will be done by providing both correct and erroneous data while writing off unpaid rent provided tenant performed some work on the unit.
- **Integration testing** will be done with respect to Lease and Rent class to see if write off is done only when specific conditions are met.
- **White box testing** will be done by looking into the logic implemented to write off unpaid rent only if tenant performed some work on the unit.
- **Frequency of testing:** Monthly

Requirement 35.1.4.4 The system will allow customer to enter the reason for the adjustment of rent in a comment field.

- **Black box testing** will be done by providing both correct and erroneous data for the reason for adjustment of rent field, and saving it into the database.
- **Integration testing** will be done with respect to Lease and Rent class to see if the reason for rent adjustment is being updated successfully.
- **White box testing** will be done by looking into the logic implemented to adjust the rent by providing a specific reason.
- **Frequency of testing:** Monthly

35.1.5 Report Management

Requirement 35.1.5.1 The system will allow customer to view payment history by unit, by period, or by tenant showing rent due, payments made, and balance due.

- **Black box testing** will be done by choosing different variations of the report- report with payment history by unit, report with payment history by period, and report with payment

history by tenant, to validate whether all the reports are retrieved successfully with correct data.

- **Integration testing** will be done with respect to Customer, Property, Unit, Lease, and Rent class to see if all the variations of the payment history report are retrieved successfully with valid data.
- **White box testing** will be done by validating the different variations of report by looking into the logic implemented for a specific report view.
- **Frequency of testing:** Weekly

Requirement 35.1.5.2 The system will allow customer to view past due amount by unit, by period, or by tenant.

- **Black box testing** will be done by choosing different variations of the report- report with past due amount by unit, report with past due amount by period, and report with past due amount by tenant, to validate whether all the reports are retrieved successfully with correct data.
- **Integration testing** will be done with respect to Customer, Property, Unit, Lease, and Rent class to see if all the variations of the past due amount report are retrieved successfully with valid data.
- **White box testing** will be done by validating the different variations of report by looking into the logic implemented for a specific report view.
- **Frequency of testing:** Weekly

Requirement 35.1.5.3 The system will allow customer to view total income by unit, by period, or by tenant.

- **Black box testing** will be done by choosing different variations of the report- report with total income by unit, report with total income by period, and report with total income by tenant, to validate whether all the reports are retrieved successfully with correct data.
- **Integration testing** will be done with respect to Customer, Property, Unit, Lease, and Rent to see if all the variations of the total income report are retrieved successfully with valid data.
- **White box testing** will be done by validating the different variations of report by looking into the logic implemented for a specific report view.
- **Frequency of testing:** Weekly

Requirement 35.1.5.4 The system will allow customer to view the list of tenants by unit; tenants who are behind, the dollar amount behind, and the date of last payment, and the list of all tenants whose lease is up.

- **Black box testing** will be done by choosing different variations of the report- report with a list of all tenants, report with list of tenants with unpaid rents, and report with list of tenants with expired lease, to validate whether all the tenant list reports are retrieved successfully with correct information.

- **Integration testing** will be done with respect to Customer, Property, Unit, Lease and Tenant class to see if all the variations of the tenant list are retrieved successfully with valid data.
- **White box testing** will be done by validating the different variations of report by looking into the logic implemented for a specific report view.
- **Frequency of testing:** Weekly

Requirement 35.1.5.5 The system will allow customer to view the list of current vacancies of units.

- **Black box testing** will be done by choosing the Vacant Unit Report submenu option from the Generate Reports menu option- to validate whether the report is being retrieved successfully with correct information of all units within each property with their current vacancies.
- **Integration testing** will be done with respect to Customer, Property and Unit class to see if the current vacancy of units is being retrieved successfully with valid data.
- **White box testing** will be done by validating the report by looking into the logic implemented to retrieve the current vacancy of a unit.
- **Frequency of testing:** Weekly

Requirement 35.1.5.6 The system will allow the customer to view the report that lists all units for which lease term was broken for more than two times in the past.

- **Black box testing** will be done by choosing the Bad Leasing Units Report submenu option from the Generate Reports menu option- to validate whether the report is being retrieved successfully with correct information of all units within each property for which lease term was broken for more than two times in the past.
- **Integration testing** will be done with respect to Customer, Property, Unit and Lease class to see if the all the units with frequent broken leases (more than two times in the past) are being retrieved successfully with valid data.
- **White box testing** will be done by validating the report by looking into the logic implemented to retrieve the units with frequent broken leases.
- **Frequency of testing:** Weekly

Requirement 35.1.5.7 The system will allow the customer to generate a report for total amount being deposited by the tenant for a unit.

- **Black box testing** will be done by choosing the Generate Deposit Report submenu option from the Generate Reports menu option- to validate whether the report is being retrieved successfully with correct information of the total amount of rent being deposited by a tenant for a unit.
- **Integration testing** will be done with respect to Customer, Property, Unit, Lease, and Rent class to see if the total rent amount deposited by a tenant for a unit is being retrieved successfully with valid data.
- **White box testing** will be done by validating the report by looking into the logic implemented to retrieve the total rent amount deposited by a tenant for a unit.

- **Frequency of testing:** Weekly

35.2 Test plan for Nonfunctional Requirements

35.2.1 Operational

Requirement 35.2.1.1 The system should be a distributed system which will be able to handle up to 1000 different property management groups, each with 1 – 10000 units

- **Operational Testing** will be done by generating an automated script to insert dummy data containing 1000 different properties and 10000 units inside a single property so in total 10000000 units will be created to check whether the database is able to store all that data.
- **Frequency of testing:** Weekly

35.2.2 Performance

Requirement 35.2.2.1 Allow fast navigation and quick exit from particular screen within 3 seconds

- **Performance testing** will be done by navigating and checking whether the screens are loading in 3 seconds and also try exiting from any screen.
- **Frequency of testing:** Weekly

Requirement 35.2.2.2 Retrieve search results from the database within 5 seconds

- **Performance testing** will be done by retrieving data through all the search criteria and verifying results are displayed within 5 seconds
- **Frequency of testing:** Weekly

Requirement 35.2.2.3 Be available at all times i.e. 24 hours a day and 7 days a week

- **Operational Testing** will be done by logging into the system at any time of the day and any day of the week and checking whether the system is up and running.
- **Frequency of testing:** Weekly

Requirement 35.2.2.4 Allow any updates to be saved within 3 seconds

- **Performance testing** will be done by editing the data and checking whether the data is stored in the system within 3 seconds
- **Frequency of testing:** Weekly

Requirement 35.2.2.5 Store a minimum of 1000 tenants per unit in the database

- **Operational Testing** will be done by generating an automated script and inserting dummy data containing more than 1000 tenants per unit and checking whether all the tenant information is stored in the database.
- **Frequency of testing:** Weekly

Requirement 35.2.2.6 Process selected report information quickly and accurately within 5 seconds

- **Performance Testing** will be done by generating a report and checking whether the report is processed with all the accurate information within 5 seconds.
- **Frequency of testing:** Weekly

35.2.3 Security

Requirement 35.2.3.1 Only the authorized user should be able to access the system.

- **Black Box Testing** will be done by providing both correct and erroneous login credentials and the output will be evaluated.
- **Frequency of testing:** Weekly

Requirement 35.2.3.2 Secure login features are provided like password must consist a combination of number and characters

- **Black Box testing** will be done providing both correct and erroneous password data and the output will be evaluated for password as possible combinations of number and characters
- **Frequency of testing:** Weekly

35.2.4 Cultural and Political

Requirement 35.2.4.1 System will store the rent amount in US Dollars

- **Black Box testing** will be done by providing amounts as numbers and verifying that the data is stored in US Dollars
- **Frequency of testing:** Weekly

35.3 Test Cases

35.3.1 Add Tenant

Test Case ID	TC_001
Requirement	2.1.2.1 The system will allow the customer to enter details about the tenant- name, phone number (up to 3) and deposit amount paid.
Unit Testing	Classes - Tenant
Reviewed By	Tsai Mei
Tester's Name	Vijay
Date Tested	28-Nov-2017
Version	1.0
Test Precondition	1. Database is up and running 2. Customer is authenticated

Test No.	Tenant First Name	Tenant Last Name	Tenant Phone	Tenant Email	Tenant Address	Deposit Amount	Valid/invalid	Reason
1	Blank	Chen	979-123-456	abc@gmail.com	603 George Bush Drive W	$100	Invalid	Tenant First Name is blank. It is a required field.
2	Jerry	Blank	979-456-789	def@tamu.edu	1201 Texas Avenue	$150	Invalid	Tenant Last Name is blank. It is a required field.
3	Chris	Evans	Blank	ghi@gmail.com	23 University Drive	$200	Invalid	Tenant Phone is blank. It is a required field.
4	Zooey	Deschanel	979-789-123	Blank	603 George Bush Drive W	$250	Invalid	Tenant Email is blank. It is a required field.
5	Tom	Ford	886-123-456	jkl@yahoo.com	Blank	$300	Invalid	Tenant Address is blank. It is a required field.
6	Yuriko	Yoshitaka	886-345-678	mno@hotmail.com	603 George Bush Drive W	Blank	Invalid	Deposit Amount is blank. It is a required field.
7	Andy	Murray	886-456-789	pqr@gmail.com	1201 Texas Avenue	$350	Valid	All the inputs are valid

35.3.2 Add Unit

Test Case ID	TC_002
Requirement	2.1.1.2 The system will allow the customer to add new units to the database
Unit Testing	Classes - Property and Unit
Reviewed By	Tsai Mei
Tester's Name	Vijay
Date Tested	28-Nov-2017
Version	1.0
Test Precondition	1. Database is up and running 2. Customer is authenticated 3. Property in which unit is to be added is available

Test No.	Unit name	Unit Availability	Unit Description	Unit Rent	Unit Property	Valid/invalid	Reason
1	#306	Yes	Blank	$300	Cedar Creek Condos	Invalid	Description is blank. It is a required field.
2	Blank	Yes	2B2B	$600	Park West	Invalid	Unit name is blank. It is a required field.
3	#2301	Blank	2B1.5B	$450	Sundance Apartment	Invalid	Unit availability is blank. It is a required field.
4	#101	Yes	1B1B	Blank	The Enclave	Invalid	Unit Rent is blank. It is a required field.
5	#29	No	3B3B	$350	The Gardens	Valid	All the inputs are valid
6	#77	ABC	1B1B	ABC	College Main	Invalid	Unit availability should be Yes or No. Unit Rent should be numbers.

36. References

- Requirement Definition – *Requirements Determination, System Analysis and Design, Fifth Edition. By Alan Dennis, Barbara Wixom, and Roberta Roth*
- Package and Class Diagram – *Class Diagrams: The Essentials, and Package Diagrams, UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition. By Fowler, Martin*
- Deployment Diagram - Deployment Diagrams, *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition. By Fowler, Martin*
- Database Schema – *Data Modeling, System Analysis and Design, Fifth Edition. By Alan Dennis, Barbara Wixom, and Roberta Roth*
- Testing Plan Template –https://www.guru99.com/download-sample-test-case-template-with-explanation-of-important-fields.html
- Fully Dressed Use Case Template – *Use Case Analysis, System Analysis and Design, Fifth Edition. By Alan Dennis, Barbara Wixom, and Roberta Roth*
- Sequence Diagram – Sequence Diagrams, *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition. By Fowler, Martin*
- Screen mock ups created using https://www.wix.com/
- Case Study: An IS Capstone Project- The Mywick Property Management System- Journal of Information Systems Education, Vol. 14(3) - Martha Myers