Rental Property Management System

CECS 343

Group 2

Jean Dieb, Al-Muntaser Al Mata'ni

& Manav Dillon

June 29th 2021

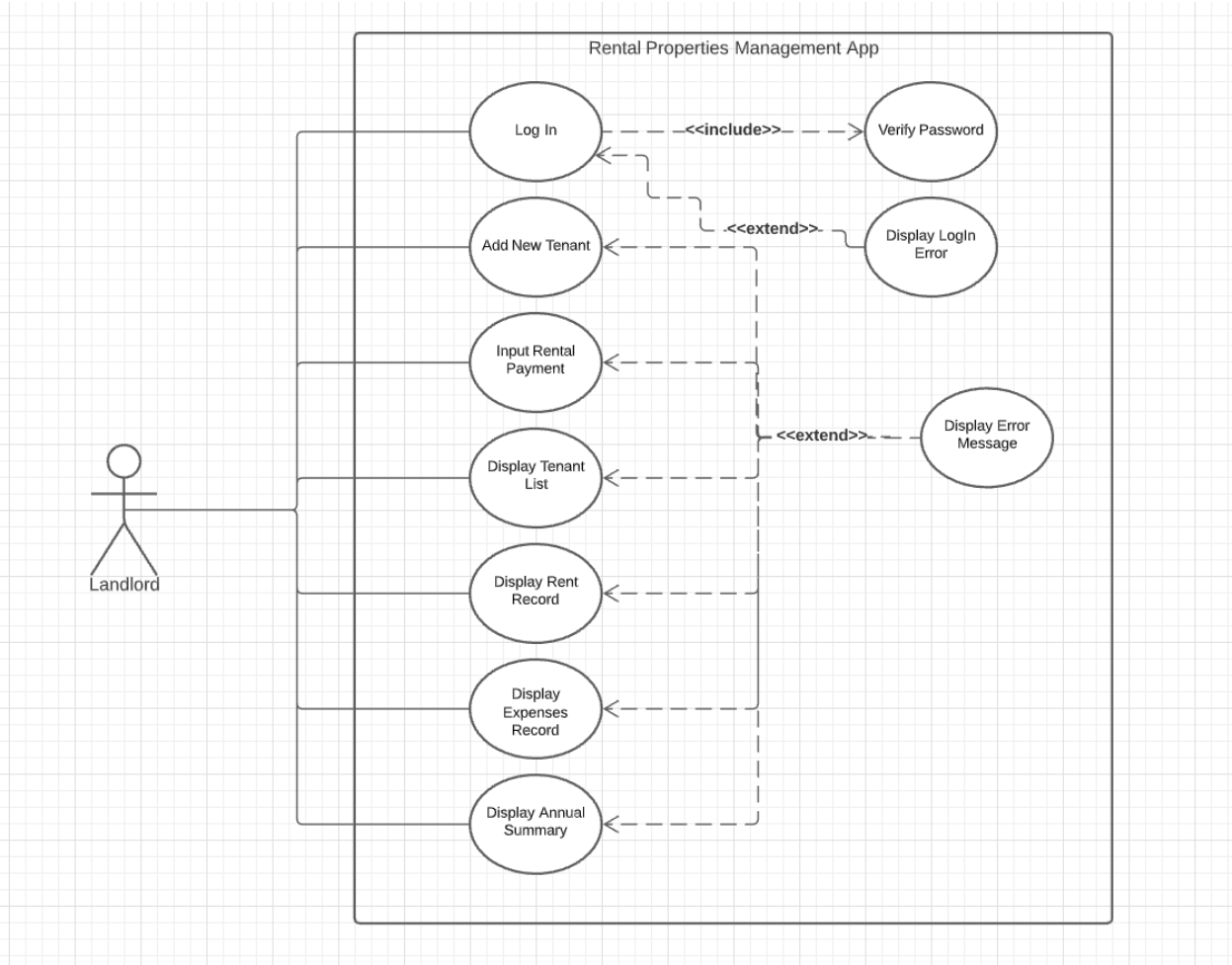
Contents

Problem Statement

The problem is that John Nguyen is using old school method to log his business's financial records. He is using ledger books to write down by hand all the information about his apartments, tenants, rent received, and expenses paid. This is a problem because he might make an untraceable mistake when he's writing down information, it is time consuming to make copies for back-ups and update all of them when new information is added, and it is also inconvenient to update and delete data. A successful solution would be a program that John can use to input, edit, and delete data about his apartments and tenants, that also keeps track of recent update. Give him reports in a nice format whenever needed, and also back his data regularly.

For John Nguyen who is small-time landlord who uses inefficient method to log his important data, our Rental Property Management System is used to log data efficiently, carry out all the complicated calculations accurately and provide real time reports upon request. Unlike the outdated method that the user currently uses that is prone to many mistakes that could lead to having inaccurate data, our product will allow users to log their data accurately and easily.

Use Case and Scenarios



Use case number: 1

Use case name: Log In

Summary: prompt the user to enter a username and a password to log into the system and start using it.

Actor: Landlord

Precondition: the computer must be on the user needs to run the program before he can log in.

Scenario:

1) the user enters a user name and a wrong password

2) the system notifies the user that either the username or password are incorrect

3) the system prompts the user to enter his credentials again

4) the user enters his information correctly

5) the system verifies the user's identity and allow him to precede and use the following functions of the program.

Exception: There is no exceptions, the user can try as many times as needed to log in.

Postcondition: The landlord can use his digital logger after being authenticated by the system

NonFunctional: The system should immediately log the user in when he enters the correct username and password. Authentication should not take more that 2 seconds.

Use case number: 2

Use case name: Start the Program

Summary: Allows the user to decide what action will occur next

Actor: The landlord

Precondition: The landlord needs to be logged into the system

Scenario:

1) The program prompts the user to pick a task he wants done

2) The landlord pics from the choices: Add New Tenant, Input Rental Payment, Display Tenant List, Display Rent Record, Display Expenses Record, Display Annual Summary

Postcondition: The user leaves the main menu and is sent to whatever task was chosen.

Nonfunctional: speed and accuracy; the system needs to get to the next step quickly.

Use case number: 3

Use case name: add new tenant

Summary: allows the user to add a tenant information and the appartement number they occupy.

Actor: The landlord

Precondition: The landlord needs to be logged into the system

Scenario:

1) The program prompts the user to enter the tenant information

(Can one apartment have multiple tenants?)

2) the program prompts the user to enter the apartment number.

3) the program check whether the apartment is already rented out and notify the user about the result

4) the system adds the new data to the appropriate lists

Exception: If the entered apartment has already been rented out (is in the list) the program notifies the user and prompt them to enter a different appartement number.

Postcondition: The user can go back to the main menu and the system add the new tenant to the appropriate list and add it in the right position using the apartment number as an index.

NonFunctional: speed and accuracy; the system needs to add the data instantaneously and exactly as the user typed it.

Use case number: 4

Use case name: input rental payment

Summary: Enter the tenant's name, the month the rental is for, and the amount of rent received. The rental information is inserted in the Rent Record.

Actor: Landlord

Precondition: The landlord needs to be logged into the system

Scenario:

1) the system prompts the landlord to enter a tenant's name

2) The landlord input a name

3) The system verifies if the name entered belongs to one of tenants in the list

4) if not, the system displays an error message and ask the landlord to try again

5) if the name is in the list, it prompts the landlord to input the month and amount received (also validate input)

6) The system adds the data entered to the Rent Record.

Exceptions: If the tenant entered is not in the tenants' list, error message will be displayed.

If month > 12 or month < 0, error message will be displayed

If rent received < 0, error message will be displayed

Postcondition: The user can go back to the main menu and the system adds the new rental payment to the appropriate list.

NonFunctional: accuracy and efficiency

Use case number: 5

Use case name: Display tenant list

Summary: display the names of the tenants and the apartment each one occupies.

Actor: Landlord

Precondition: The landlord must be logged into the system.

Scenario:

1) The landlord chooses this option from a menu

2) The system displays the required results if there are tenants in the list

3) if not, an empty list message is displayed

Exception: if there are no tenants in the list, an error message is displayed

Postcondition: The user can go back to the main menu after viewing the data he requested.

NonFunctional: speed and accuracy

Use case number: 6

Use case name: Display rent record

Summary: display a list of apartments rented and the amount of rent paid each month for each apartment

Actor: Landlord

Precondition: Landlord must be logged in the system

Scenario:

1) The landlord chooses this option from a menu

2) The system prints the required results and place a zero for the months he did not receive rent for yet.

Exception: if the list of apartments is empty, an error message will be displayed

Postcondition: The landlord can go back to the main menu after viewing the requested data.

NonFunctional: speed and accuracy

Use case number: 7

Use case name: Display expenses record

Summary: display the month, day, payee, amount paid and budget category for each payment the landlord has made.

Actor: Landlord

Precondition: the landlord must be logged into the system.

Scenario:

1) The landlord chooses this option from a menu

2) The system format and display the required results if there are any expenses paid so far.

3) if not, the system displays a “no expenses recorded” message.

Exception: If the required list is empty, the system displays a warning message.

Postcondition: The landlord can go back to the main menu after viewing the requested data

NonFunctional: speed and accuracy

Use case number: 8

Use case name: Display the annual summary

Summary: Display the total rent paid to the landlord for the year to date, a list of the total expenses for each budget category, and the resulting balance for the year to date (revenues - expenses)

Actor: Landlord

Precondition: The landlord must be logged into the system

Scenario:

1) the landlord chooses this option from a menu

2) The system displays the required report

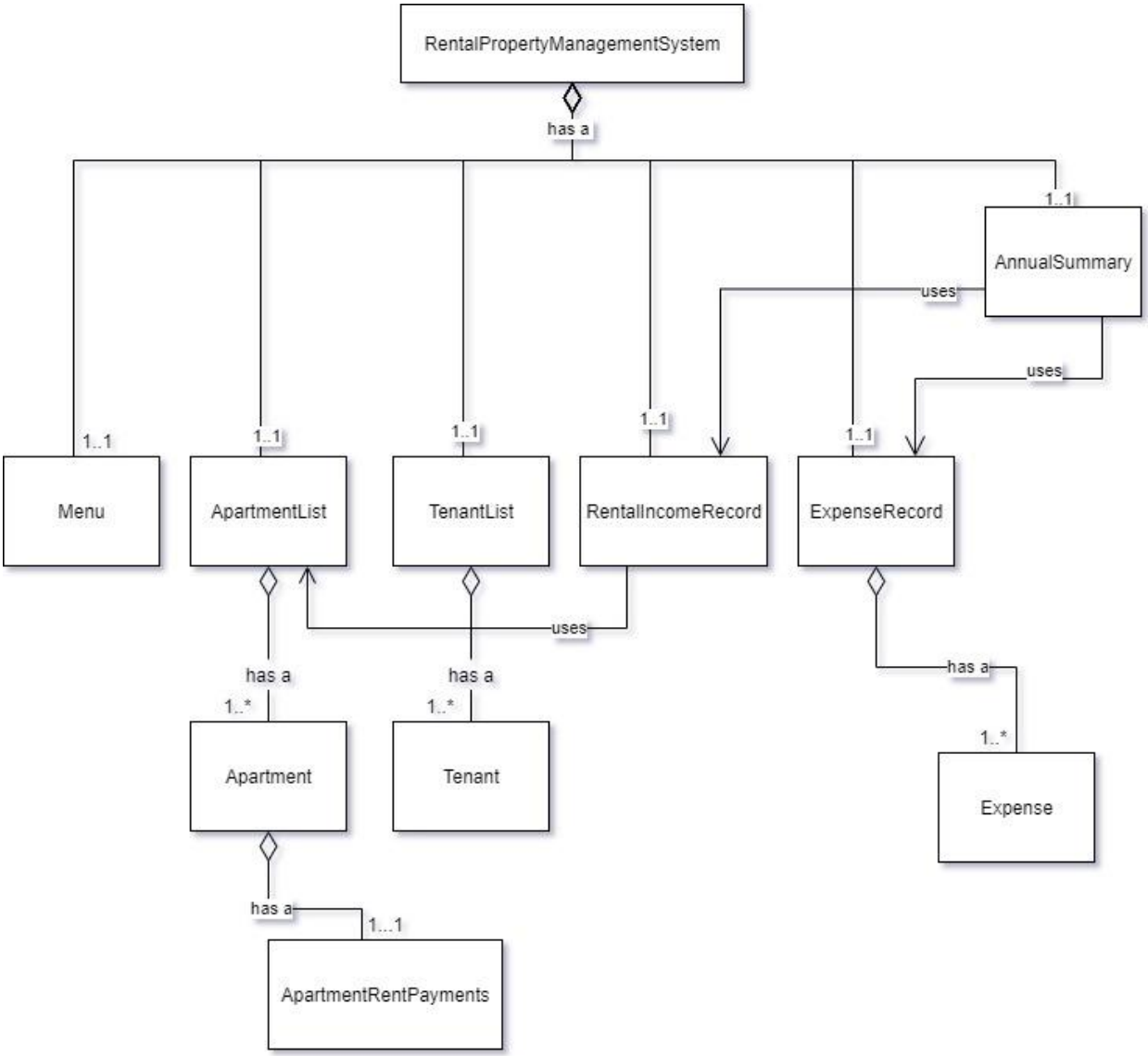
Exception: if any of the lists is empty, an appropriate message will be displayed

Postcondition: the user can go back to the main menu after viewing the requested data

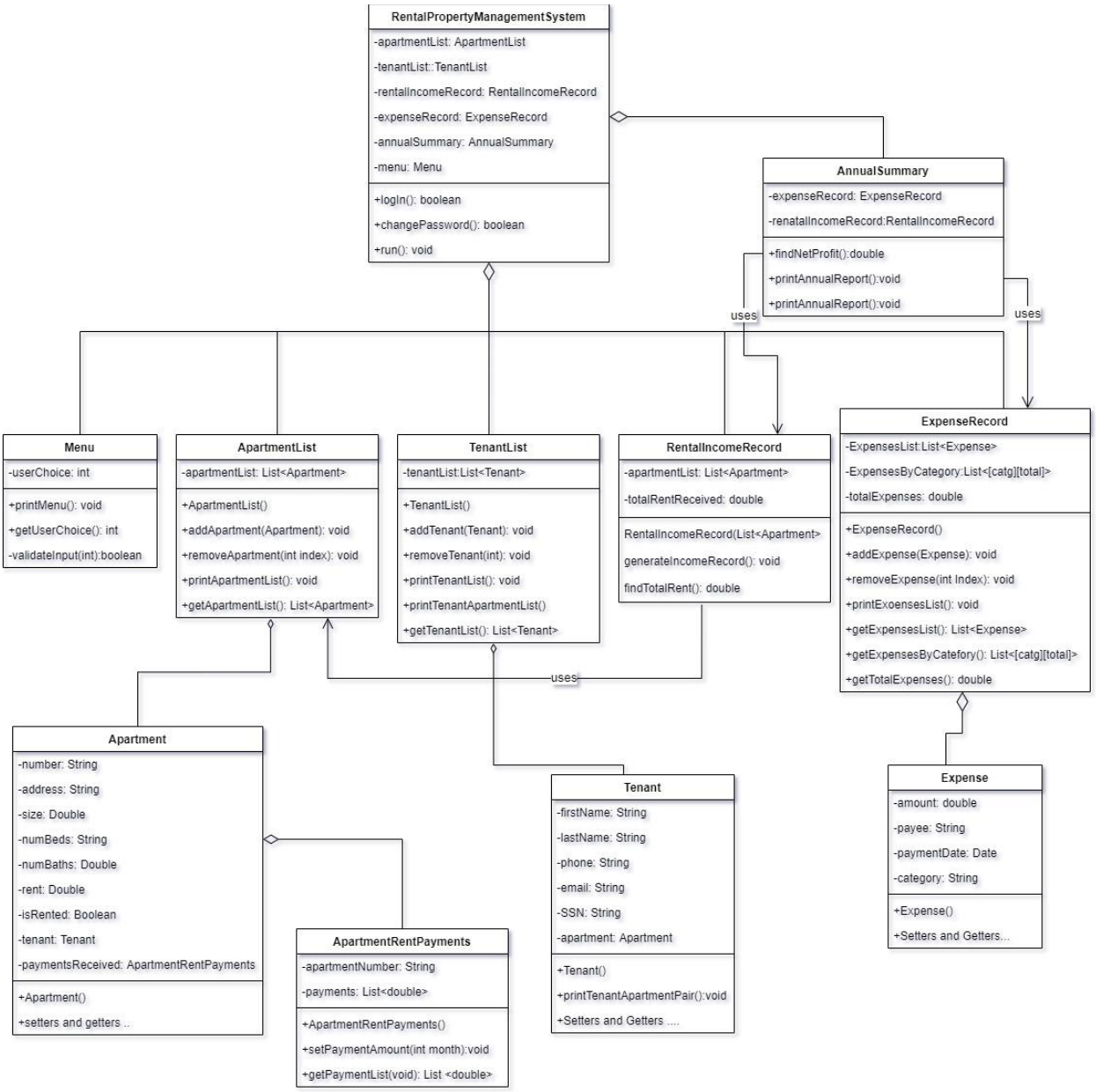
NonFunctional: accuracy, efficiency and speed.

Requirement Modeling

a. **Class Diagram**



b. **Detailed Class Diagram**



c. **CRC**

Tenant	
• store tenants information; their name, phone number, email, apartment, and social security	• TenantList

Apartment	
• Store apartment info • number, address, size, bed, bath, rent, and if its rented or not, and the tenant if it is rented	• Tenant • ApartmentRentPayments

ApartmentList	
• store all the apartments that the landlord have	• Apartment • RentalIncomeRecord

TenantList	
• Store the tenants and the apartments they are occupying	• Tenant

RentalIncomeRecord	
• stores payment received for each apartment • contains 13 columns; one for the apartment number and 12 for the months.	• ApartmentList

Expense	
• stores the date, the payee, the amount being paid, and the budget categor	• ExpenseRecord

ExpenseRecord	
• used to store all the expenses the landlord pays • find total Expenses by category • Calculate total Expenses in the record	• Expense

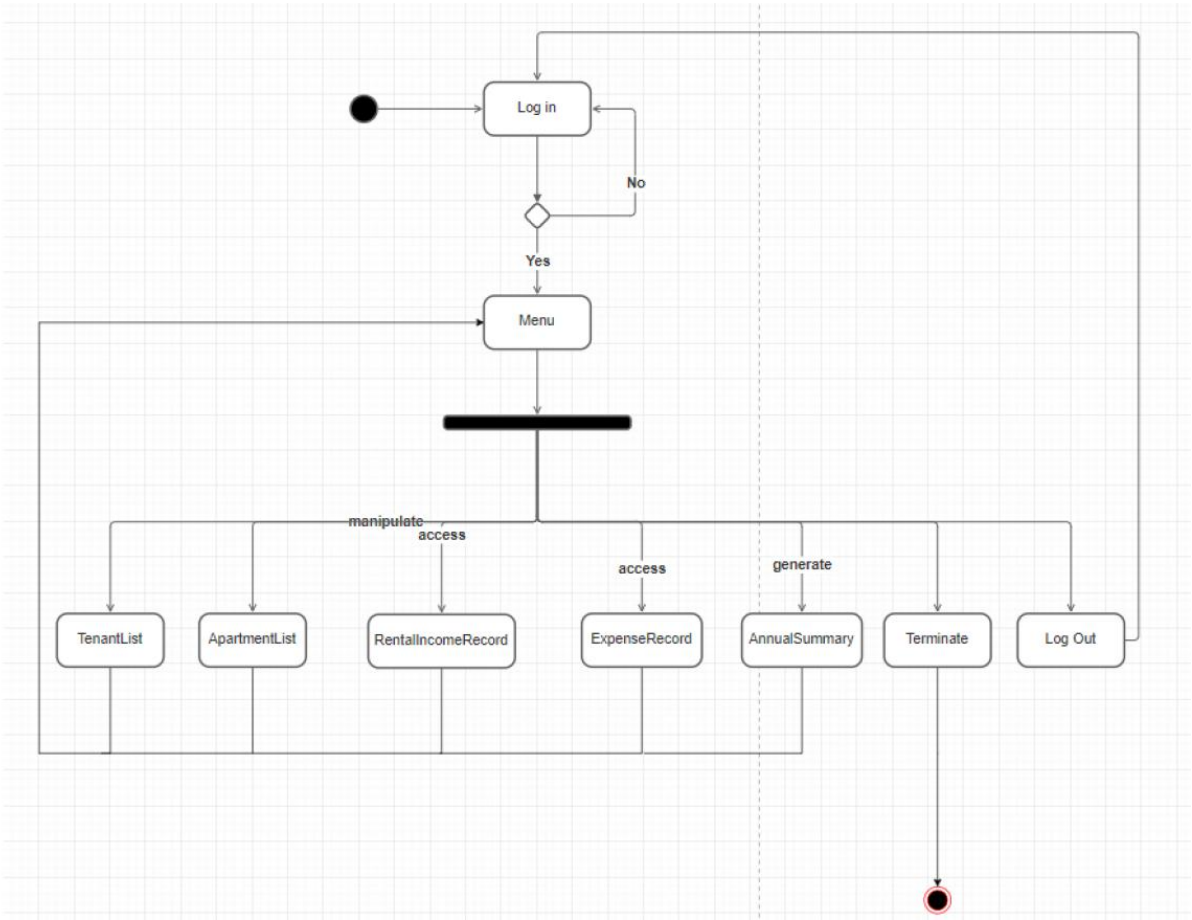
AnnualSummary	
• sum up all the data from the other classes to generate useful analytics. • calculate all the rental income • calculate the net profit	• RentalIncome • ExpenseRecord

Menu	
• print a list of the function the program offers • take user's input and validate it	

RentalPropertyManagementSystem	
• main class in the program • responsible of managing and calling all the other classes	• all classes except for Expense, Tenant, ApartmentRentPayments, and apartment

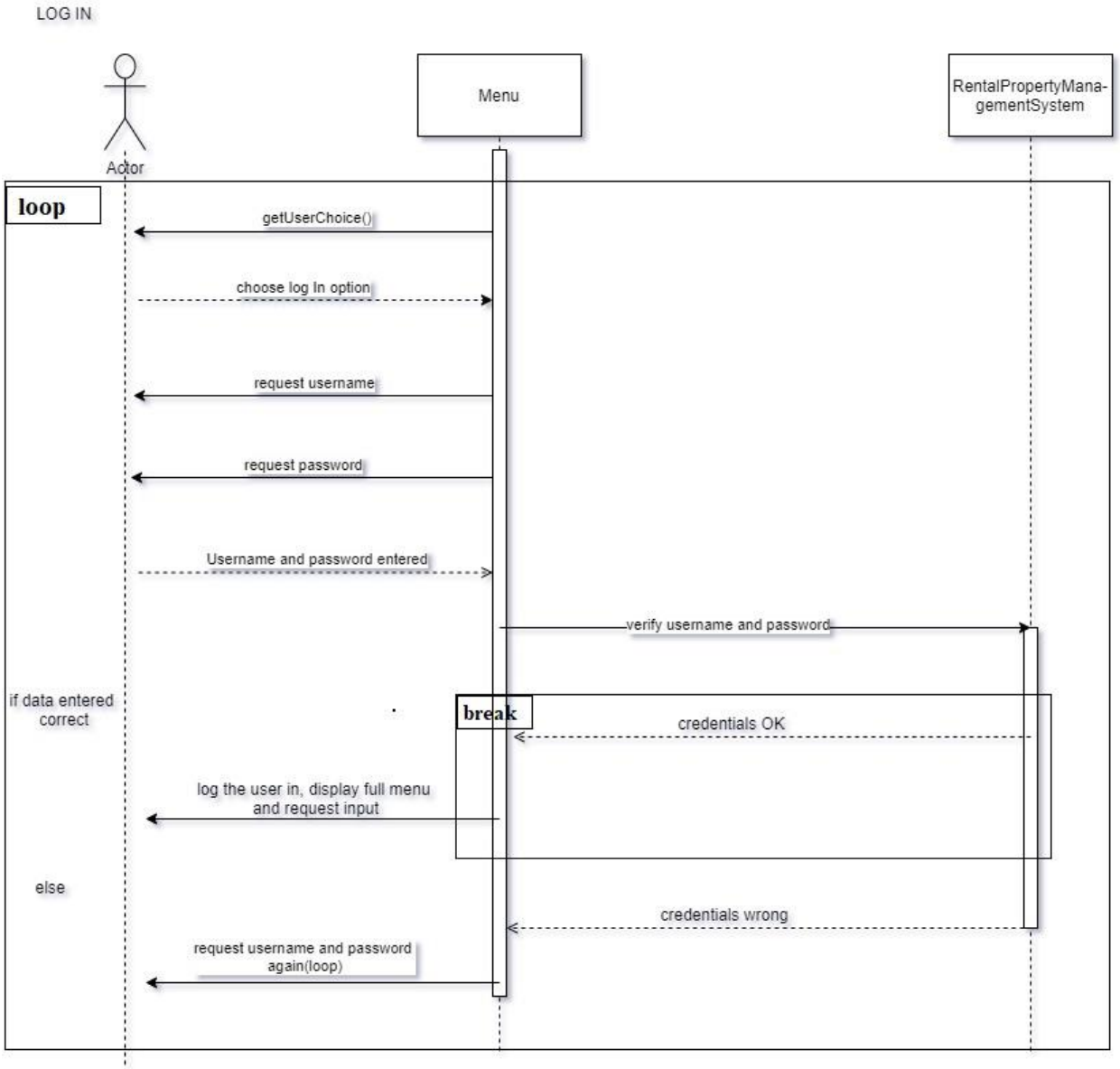
ApartmentRentPayments	
• store an apartment number with the monthly payments received for that apartment	• Apartment

d. **Activity Diagram**

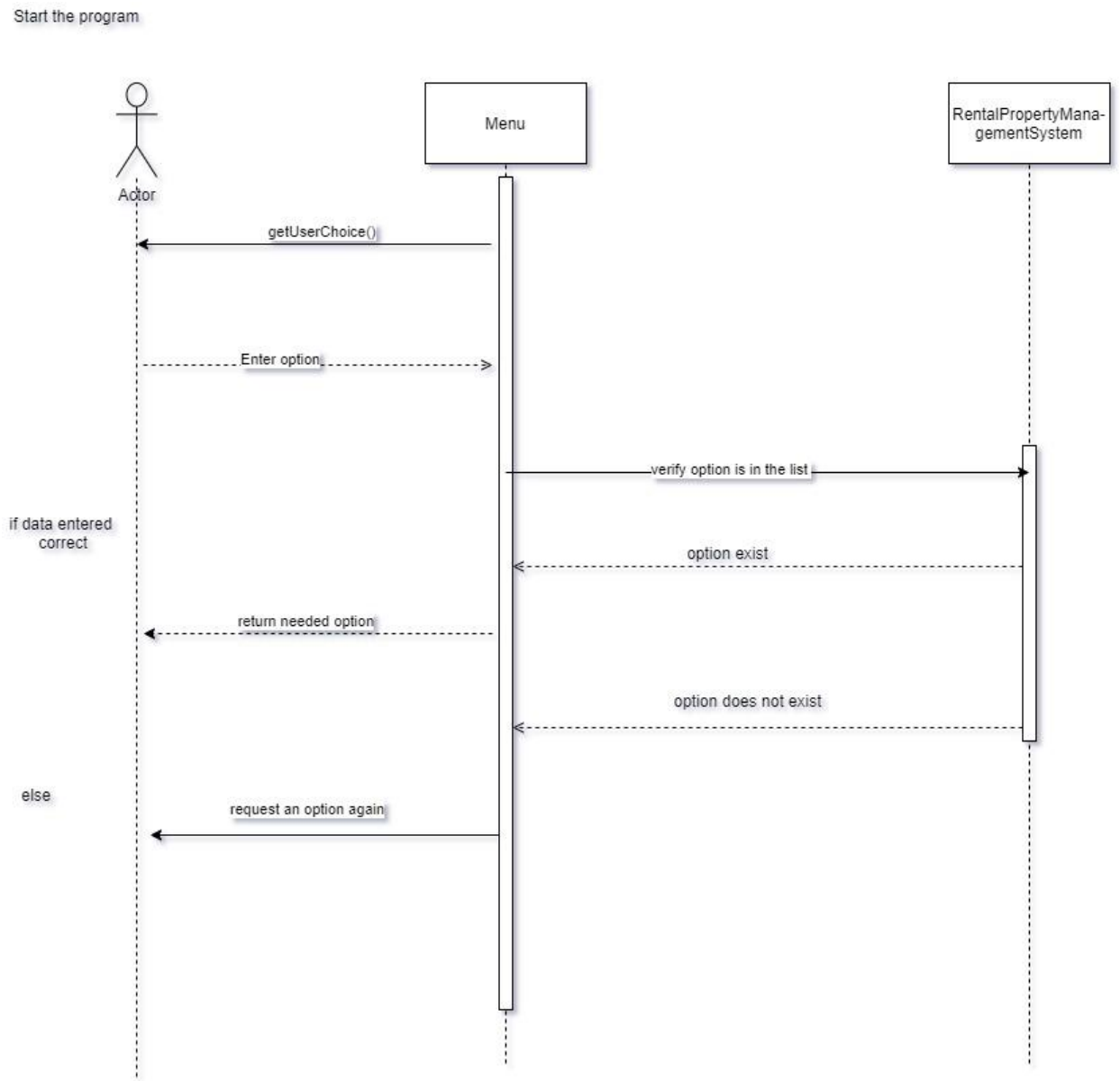


e. **Sequence Diagram**

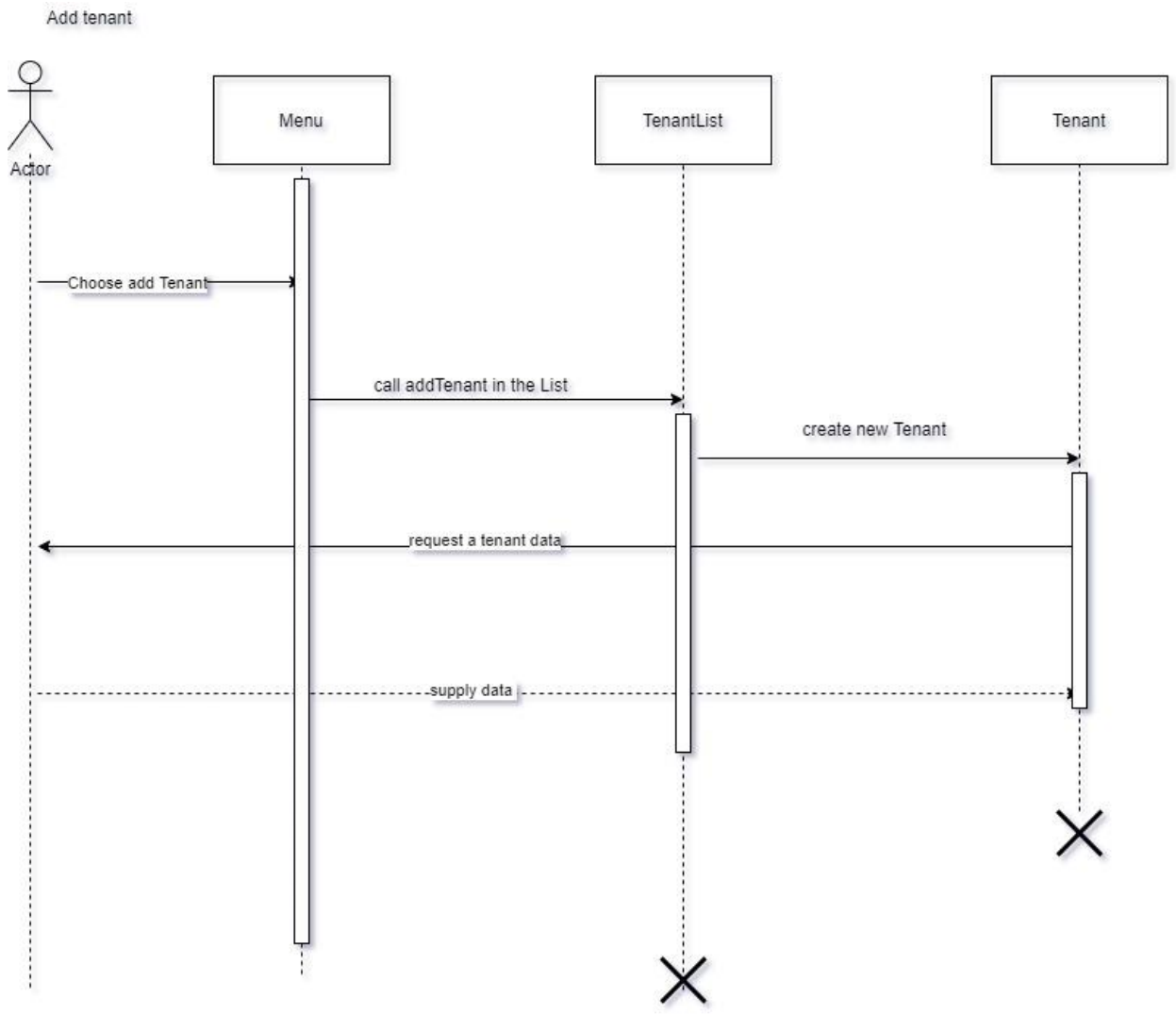
Use case



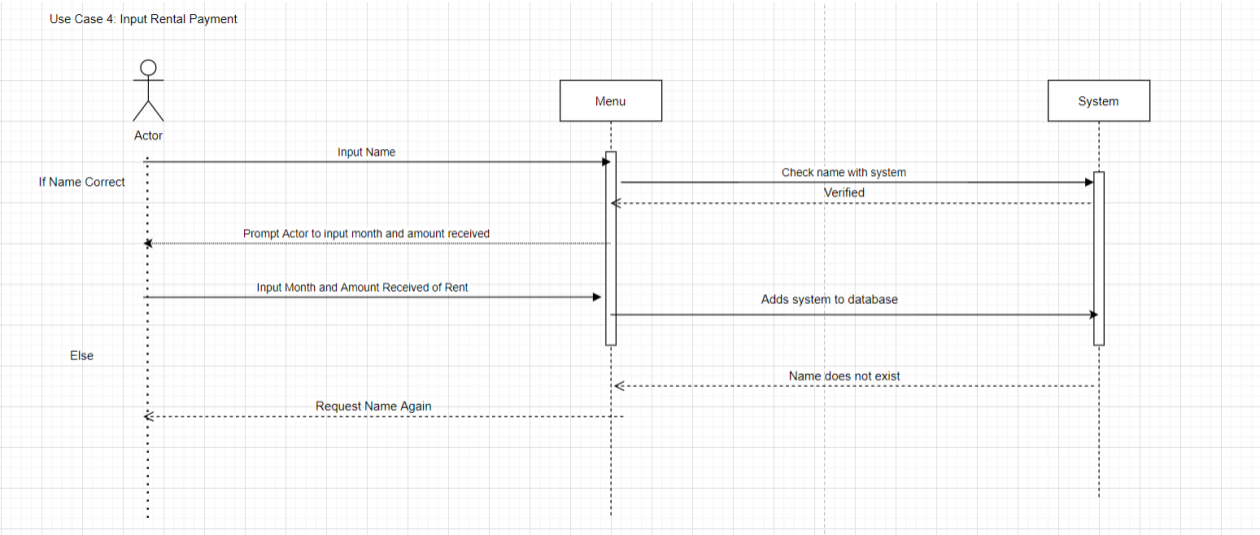
Use case 2:


Start the program
Actor
Menu
RentalPropertyMana-
gementSystem
getUserChoice()
Enter option
verify option is in the list
if data entered
correct
option exist
return needed option
option does not exist
else
request an option again

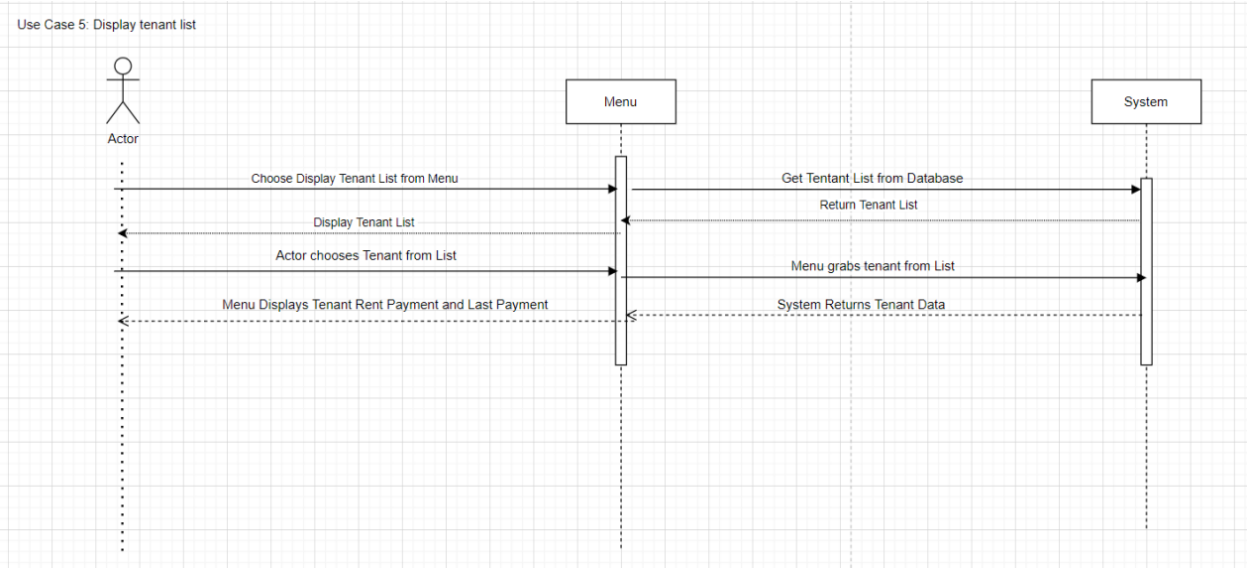
Use case 3:


Add tenant
Actor
Menu
TenantList
Tenant
Choose add Tenant
call addTenant in the List
create new Tenant
request a tenant data
supply data

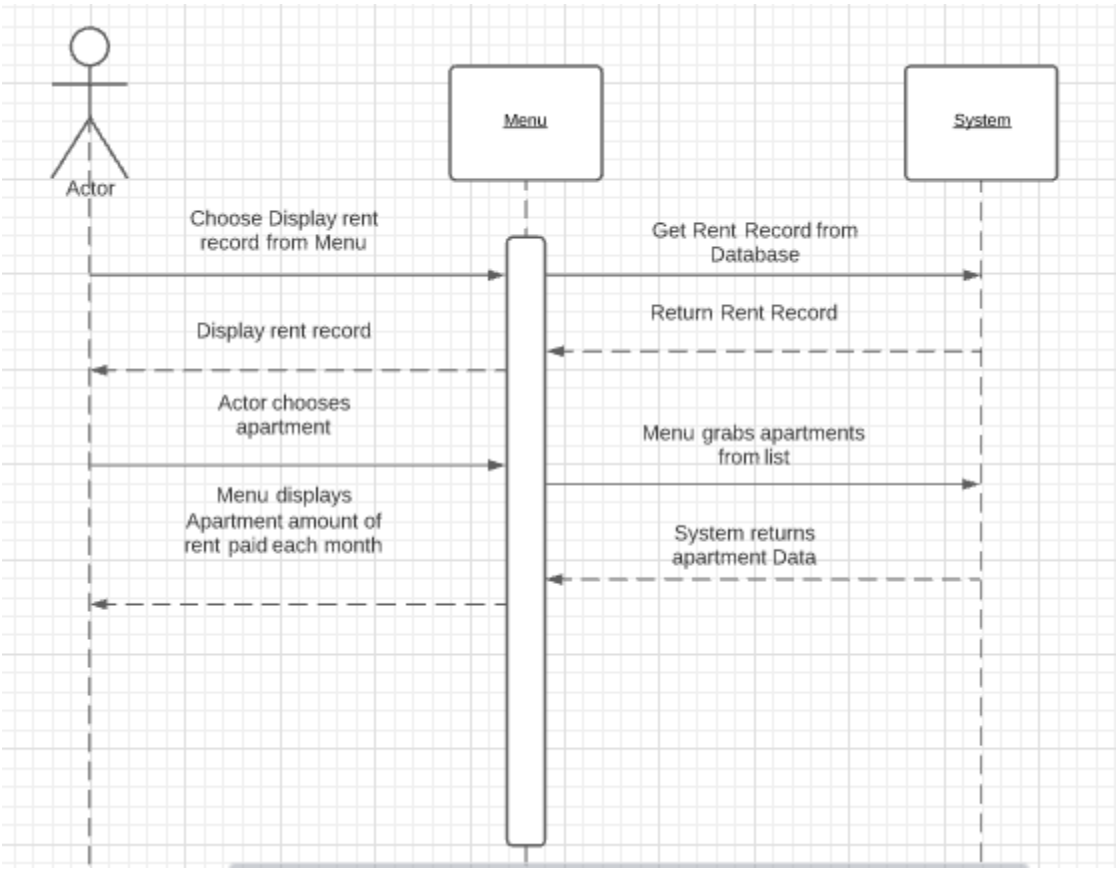
Use case 4:



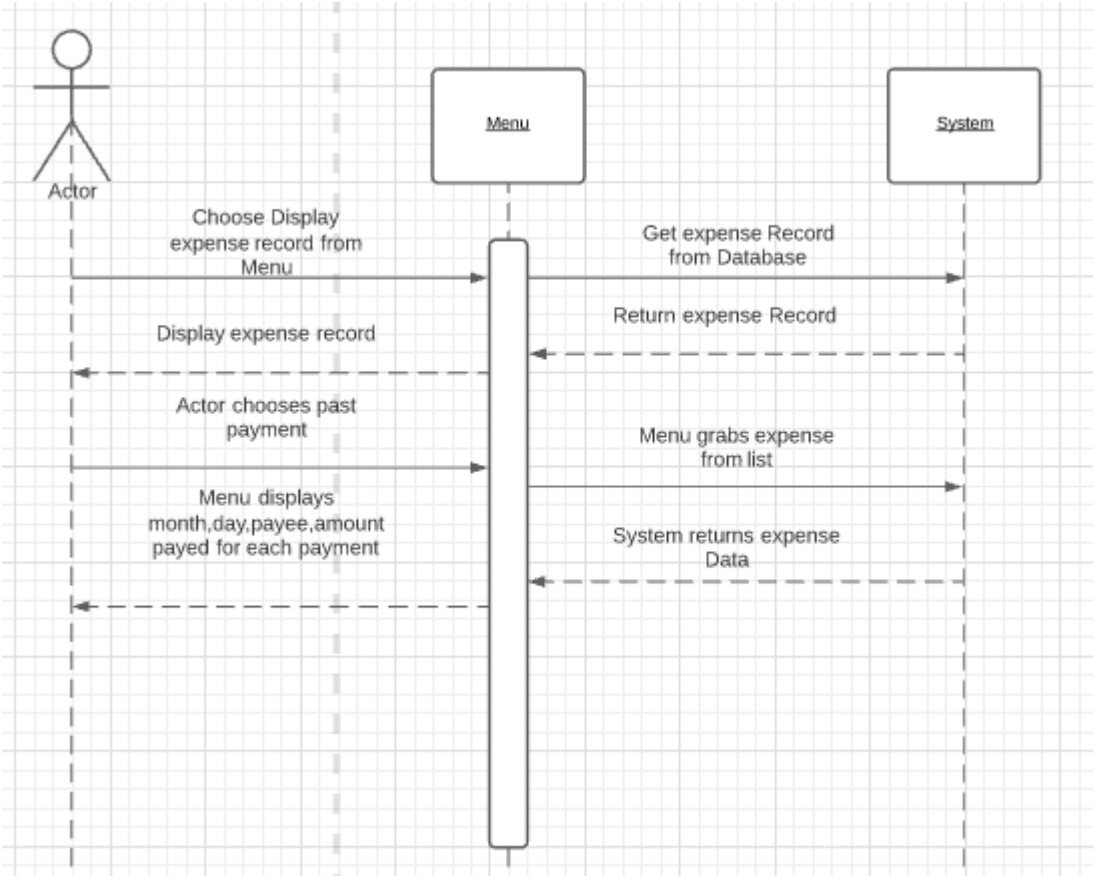
Use case 5:



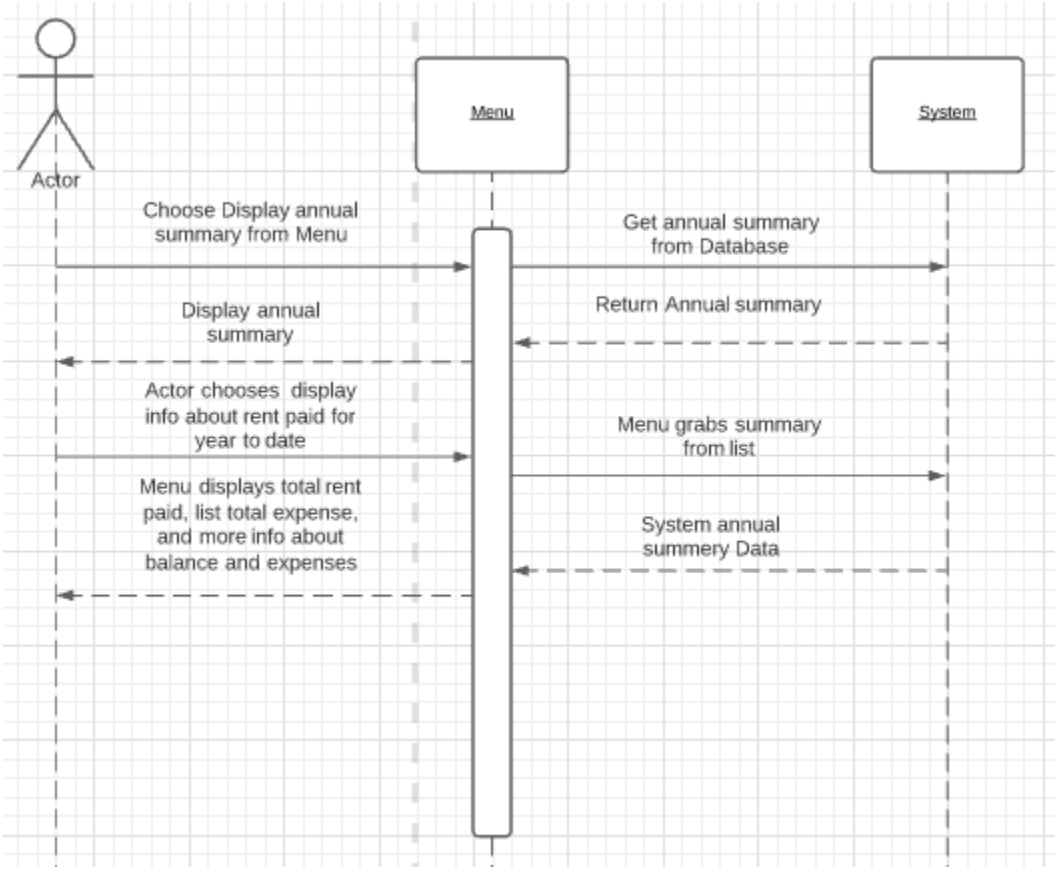
Use case 6:


Actor
Menu
System
Choose Display rent record from Menu
Get Rent Record from Database
Return Rent Record
Display rent record
Actor chooses apartment
Menu grabs apartments from list
Menu displays Apartment amount of rent paid each month
System returns apartment Data

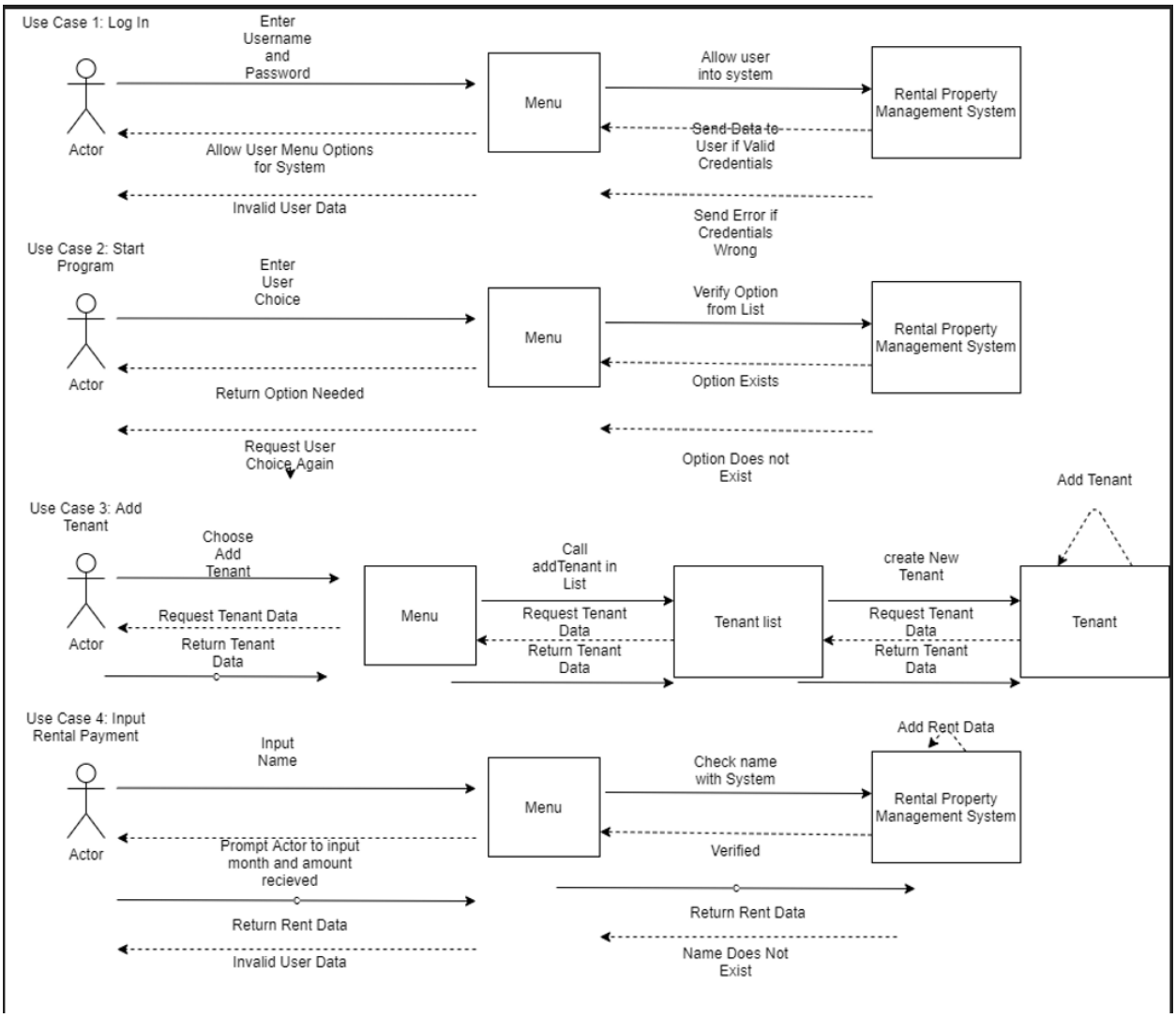
Use case 7:

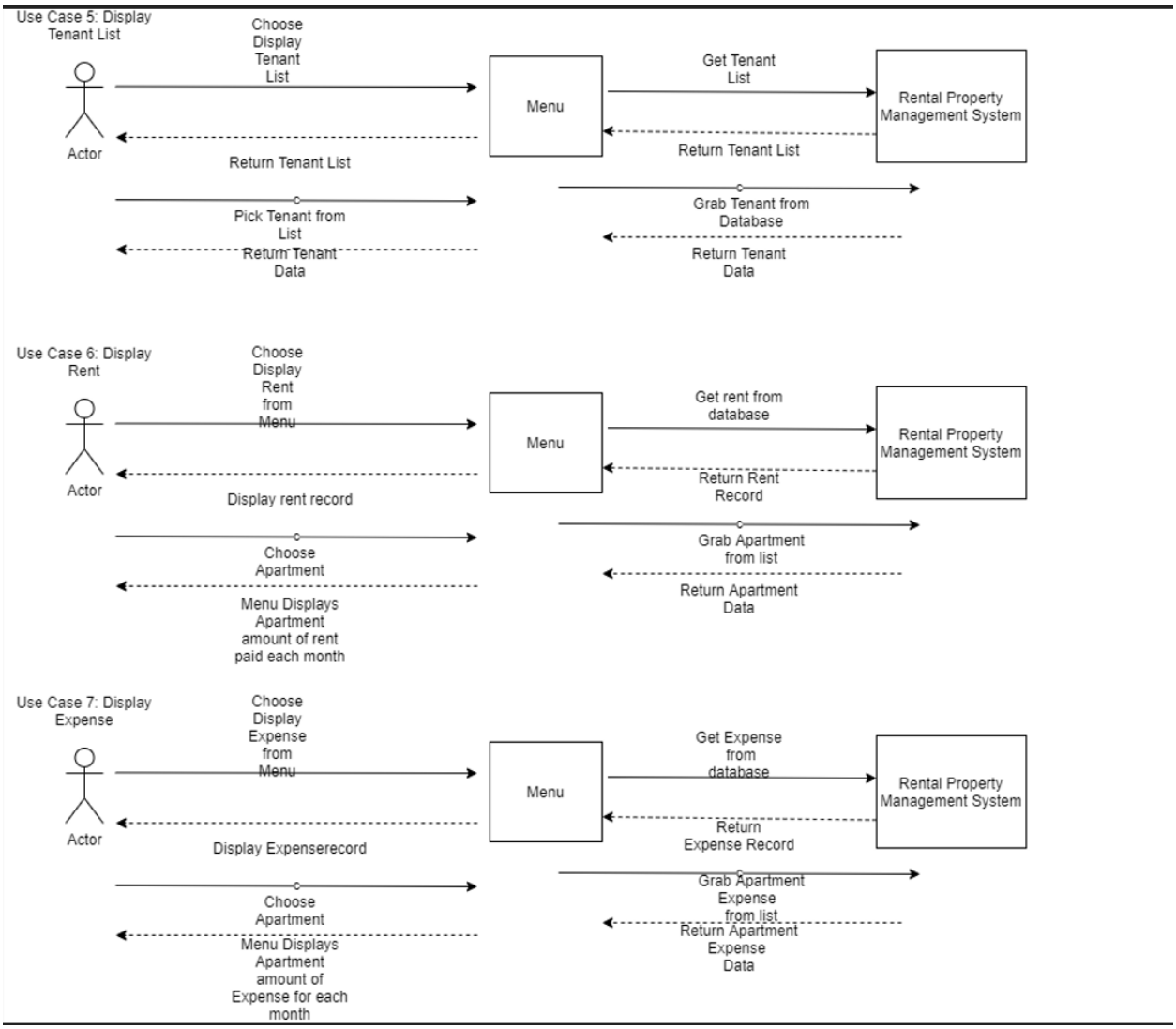

Actor
Menu
System
Choose Display expense record from Menu
Get expense Record from Database
Return expense Record
Display expense record
Actor chooses past payment
Menu grabs expense from list
Menu displays month,day,payee,amount payed for each payment
System returns expense Data

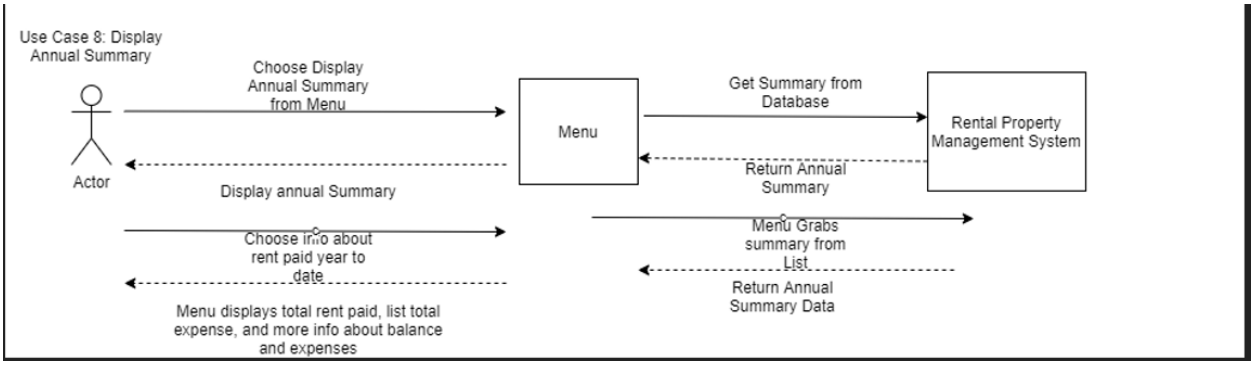
Use case 8:


Actor
Menu
System
Choose Display annual summary from Menu
Get annual summary from Database
Return Annual summary
Display annual summary
Actor chooses display info about rent paid for year to date
Menu grabs summary from list
Menu displays total rent paid, list total expense, and more info about balance and expenses
System annual summery Data

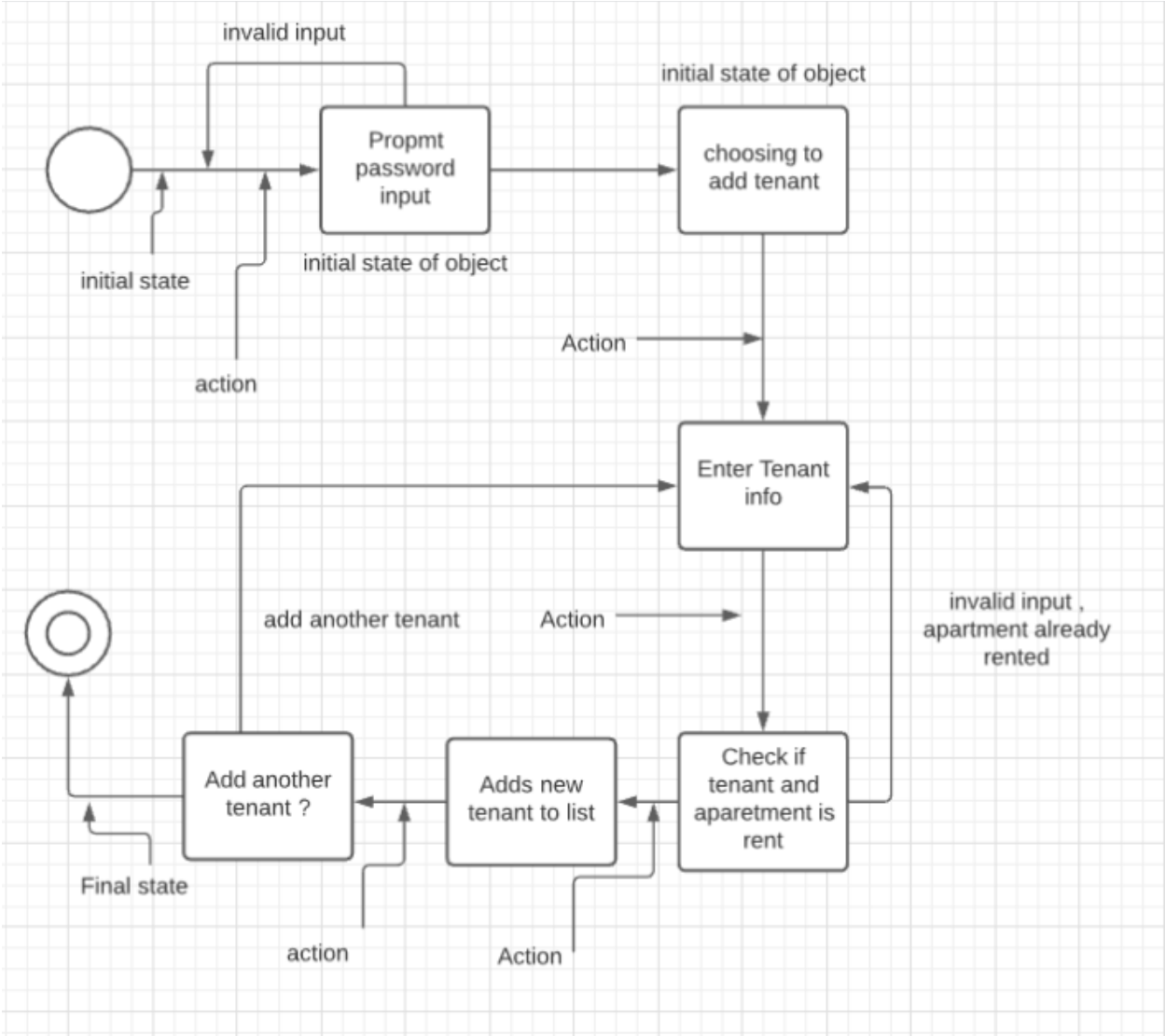
f. **Collaboration Diagram**



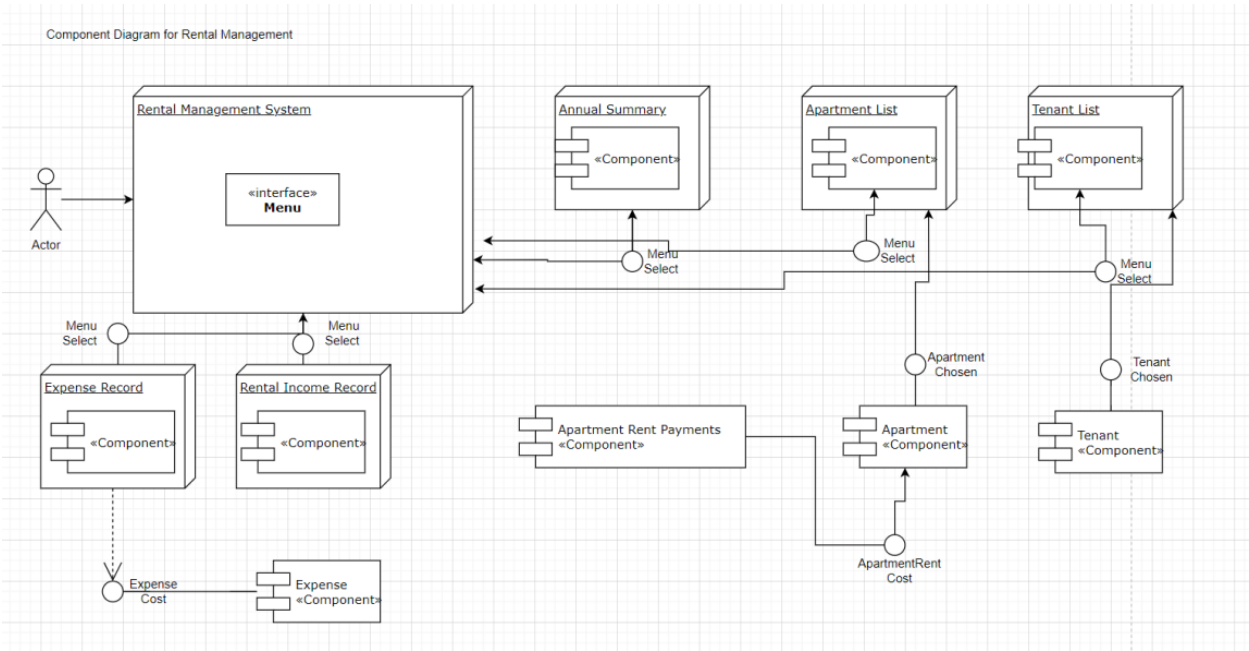

Use Case 5: Display Tenant List
Choose Display Tenant List
Actor
Menu
Get Tenant List
Rental Property Management System
Return Tenant List
Return Tenant List
Pick Tenant from List
Grab Tenant from Database
Return Tenant Data
Return Tenant Data
Use Case 6: Display Rent
Choose Display Rent from Menu
Actor
Menu
Get rent from database
Rental Property Management System
Return Rent Record
Display rent record
Choose Apartment
Grab Apartment from list
Return Apartment Data
Menu Displays Apartment amount of rent paid each month
Use Case 7: Display Expense
Choose Display Expense from Menu
Actor
Menu
Get Expense from database
Rental Property Management System
Return Expense Record
Display Expenserecord
Choose Apartment
Grab Apartment Expense from list
Return Apartment Expense Data
Menu Displays Apartment amount of Expense for each month


Use Case 8: Display Annual Summary
Choose Display Annual Summary from Menu
Actor
Menu
Get Summary from Database
Rental Property Management System
Return Annual Summary
Display annual Summary
Choose info about rent paid year to date
Menu Grabs summary from List
Return Annual Summary Data
Menu displays total rent paid, list total expense, and more info about balance and expenses

g. **State Diagram**



Component Diagram



User Interface Design

The program runs in the terminal and the user interact with the program by typing into the terminal

And the output is formatted and displayed as a text.

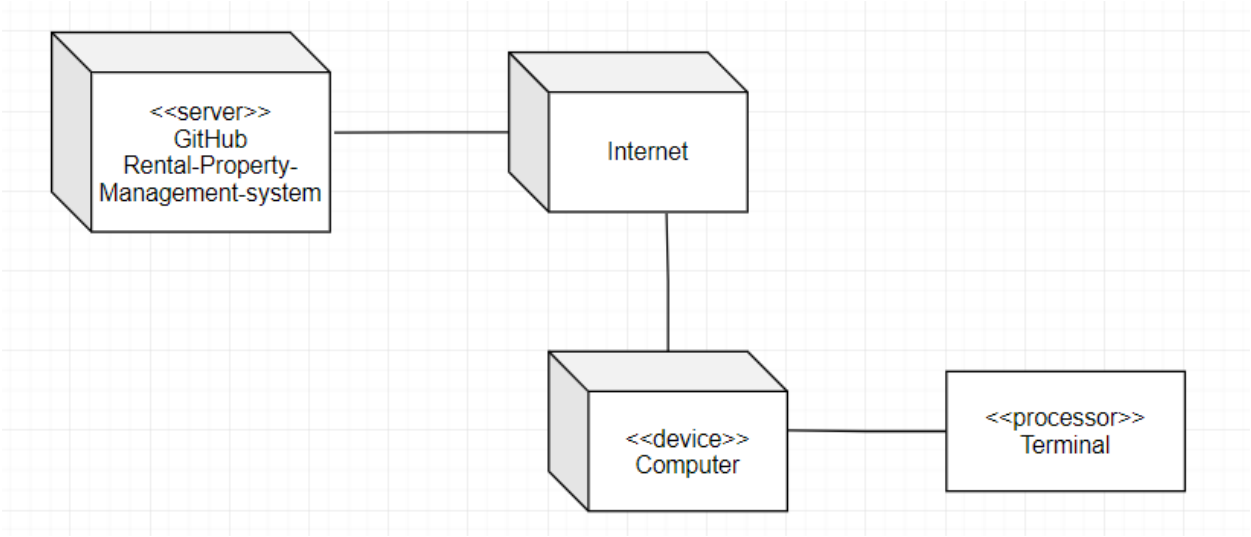
Installation and Deployment

To download the program from GitHub, click here

There is no need to install anything, the program can be used by downloading the ZIP folder, extracting it and double click Run.exe

***Note**: Windows defender might see the executable as a threat. Ignore the warning, it is safe.*

*In case you would not want to run the executable, you can run the program using Run.py, you need to have python **installed**.*



Coding

AnnualSummary.py

```
from ExpenseRecord import ExpenseRecord

from RentalIncomeRecord import RentalIncomeRecord


class AnnualSummary:

    def __init__(self, expense_rec, rental_rec):

        self.__expense_record = expense_rec

        self.__rental_income_record = rental_rec

        #self.__net_profit = 0


    def print_annual_report(self):

        self.net_profit = 0

        total_rent = self.__rental_income_record.find_total_rent()

        total_expense = self.__expense_record.get_total_expenses()

        self.net_profit = total_rent - total_expense

        print('Annual Summary:\n------------------')

        print('Income:\nRent: {}'.format(total_rent))

        print('Expenses:\n{}'.format(self.__expense_record.get_expenses_by_categ()))

        print('Total Expenses:\n{}'.format(self.__expense_record.get_total_expenses()))
```

```
    print('Balance: {}'.format(self.net_profit))
```

Apartment.py

```
from ApartmentRentPayments import ApartmentRentPayments


class Apartment:
    def __init__(self):
        #left empty because initialization will be either from user or db and each has its own method...
        pass


    def db_init(self, number, address, size, num_beds, num_baths, rent, rental_status):
        self.__number = number
        self.__address = address
        self.__size = size
        self.__num_beds = num_beds
        self.__num_baths = num_baths
        self.__rent = rent
        self.__is_rented = rental_status
        self.__apartment_rent_payments = ApartmentRentPayments(self.get_number())


    def user_init(self):
        self.set_number()
        self.set_address()
        self.set_size()
        self.set_num_beds()
        self.set_num_baths()
        self.set_rent()
        self.set__rental_status()
```

```
    self.__apartment_rent_payments = ApartmentRentPayments(self.get_number())




def set_number(self):
    self.__number = input('Enter apartment number: ')

def get_number(self):
    return self.__number

def set_address(self):
    self.__address = input('Enter apartment address: ')

def get_address(self):
    return self.__address

def set_size(self):
    try:
        self.__size = float(input('Enter the apartment size: '))
    except ValueError:
        print('size can only be numbers... try again\n')
        self.set_size()

def get_size(self):
    return self.__size

def set_num_beds(self):
    try:
        self.__num_beds = int(input('Enter number of bedrooms: '))
```

```
    except ValueError:
        print('Number of bedrooms must be an integer... try again\n')
        self.set_num_beds()

def get_num_beds(self):
    return self.__num_beds

def set_num_baths(self):
    try:
        self.__num_baths = float(input('Enter number of bathrooms: '))
    except ValueError:
        print('Number of bathrooms must be a number ... try again\n')
        self.set_num_baths()

def get_num_baths(self):
    return self.__num_baths

def set_rent(self):
    try:
        self.__rent = float(input('Enter the rent for this apartment: $'))
    except ValueError:
        print('rent can only be numbers (e.g: 12.56)... try again\n')
        self.set_rent()

def get_rent(self):
    return self.__rent

def set__rental_status(self):
    user_choice  =  input('Is this apartment rented? (y/n)')
```

```
    if(user_choice != 'y' and user_choice != 'n'):
        print('answer with either y or n ... try again\n')
        self.set__rental_status()

    else:
        if(user_choice == 'y'): self.__is_rented = True
        elif(user_choice == 'n'): self.__is_rented = False

def get_rental_status(self):
    return self.__is_rented

def set_tenant(self, tenant):
    self.__tenant = tenant
    if(tenant == None):
        self.__is_rented = False
    else:
        self.__is_rented = True

def get_tenant(self):
    return self.__tenant

def set_payment_received(self):
    self.__apartment_rent_payments.set_payment_amount()

def get_payments_received(self):
    return self.__apartment_rent_payments.get_payments_list()

def get_apartment_rent_received_summed(self):
    return self.__apartment_rent_payments.get_apartment_rent_sum()
```

```
    def print_appartment(self):
        return('Apartment number: {} \nAddress: {} \nSize: {} \n# of beds: {}\n# of baths: {} \nRent: {}\nRented? {}'
        .format(self.get_number(), self.get_address(), self.get_size(), self.get_num_beds(), self.get_num_baths(), self.get_rent(), self.get_rental_status()))
```

ApartmentList.py

```
from Apartment import Apartment
from sqlite_methods import get_apartments_form_db, save_apartment, remove_apartment


class ApartmentList:
    def __init__(self):
        self.__apartment_list = [] #list that will hold the apartments...


    def load_apartments_from_db(self):
        apartment_list = get_apartments_form_db()
        for apartment in apartment_list:
            to_be_added = Apartment()
            to_be_added.db_init(apartment[0], apartment[1], apartment[2], apartment[3], apartment[4], apartment[5], apartment[6])
            self.__apartment_list.append(to_be_added)
            to_be_added.print_appartment()
```

```
def menu(self):
    user_input = -1
    while(user_input != 0):
        while(user_input == -1):
            data = input('1) Add Apartment\n'+
                    '2) Remove Apartment\n'+
                    '3) Update Existing Apartment\n'+
                    '4) Print Apartment List\n'+
                    '0) Go Back to Main Menu\n'+
                    'Enter your choice: ')
            if(data.isdigit):
                data = int(data)
                if (data >= 0 and data <= 4):
                    user_input = data
            if(user_input == -1):
                print('Enter a number between 0 and 4 ... please try again\n')

        if(user_input == 1):
            self.add_apartment()

        elif(user_input == 2):
            self.remove_apartment()

        elif(user_input == 3):
            self.update_apartment()

        elif(user_input == 4):
            self.print_apartment_list()
```

```
        elif(user_input == 0):
            print('loading ...')
            break

    def add_apartment(self):
        apartment = Apartment()
        apartment.user_init()
        self.__apartment_list.append(apartment)
        save_apartment(apartment)

    def remove_apartment(self):
        user_choice = -1
        self.print_apartment_list()
        while(user_choice == -1):
            data = input("which apartment would you like to remove? ")
            if(data.isdigit):
                data = int(data)
                if(data > 0 and data <= len(self.__apartment_list)):
                    user_choice = data
            if (user_choice == -1):
                print('Enter a number between 1 and {}... please try
again\n'.format(len(self.__apartment_list)))
        remove_apartment(self.__apartment_list[user_choice-1])
        self.__apartment_list.pop(user_choice-1)



    def update_apartment(self):
        user_choice = -1
```

```
        self.print_apartment_list()
        while(user_choice == -1):
            data = input("which apartment would you like to edit? ")
            if(data.isdigit):
                data = int(data)
                if(data > 0 and data <= len(self.__apartment_list)):
                    user_choice = data
            if (user_choice == -1):
                print('Enter a number between 1 and {}... please try
again\n'.format(len(self.__apartment_list)))

        apartment = self.__apartment_list[user_choice-1]
        remove_apartment(apartment)
        user_choice = self.get_what_to_update()
        if(user_choice == 1):
            apartment.set_number()
        elif(user_choice == 2):
            apartment.set_address()
        elif(user_choice == 3):
            apartment.set_size()
        elif(user_choice == 4):
            apartment.set_num_beds()
        elif(user_choice == 5):
            apartment.set_num_baths()
        elif(user_choice == 6):
            apartment.set_rent()
        elif(user_choice == 7):
            apartment.set__rental_status()
        elif(user_choice == 8):
```

```
        apartment.set_tenant()
    elif(user_choice == 9):
        apartment.set_payment_received()
    save_apartment(apartment)
    print('done')



def get_what_to_update(self):
    choice = -1
    while(choice == -1):
        data = input('1) Update number\n' +
                '2) Update address\n'+
                '3) Update size\n' +
                '4) Update number of beds\n' +
                '5) Update number of baths\n' +
                '6) Update rent amount\n' +
                '7) Update rental status\n' +
                '8) Update tenant\n' +
                '9) Update payments record\n'+
                'Enter your choice: ')
        if(data.isdigit):
            data = int(data)
            if(data > 0 and data <= 9):
                choice = data
        if (choice == -1):
            print('Enter a number between 1 and 9... please try again\n')

    return choice
```

```
    def print_apartment_list(self):
        if(len(self.__apartment_list) == 0):
            print('Apartment list is empty...\n')
        index = 0
        for apartment in self.__apartment_list:
            index = index + 1
            print('{}) {}'.format(index, apartment.print_appartment()))

    def get_apartment_list(self):
        return self.__apartment_list
```

ApartmentRentPayments.py

```
from Apartment import Apartment
from sqlite_methods import get_apartments_form_db, save_apartment, remove_apartment


class ApartmentList:
    def __init__(self):
        self.__apartment_list = [] #list that will hold the apartments...


    def load_apartments_from_db(self):
        apartment_list = get_apartments_form_db()
        for apartment in apartment_list:
            to_be_added = Apartment()
```

```
        to_be_added.db_init(apartment[0], apartment[1], apartment[2], apartment[3], apartment[4],
apartment[5], apartment[6])
        self.__apartment_list.append(to_be_added)
        to_be_added.print_appartment()


    def menu(self):
        user_input = -1
        while(user_input != 0):
            while(user_input == -1):
                data = input('1) Add Apartment\n'+
                             '2) Remove Apartment\n'+
                             '3) Update Existing Apartment\n'+
                             '4) Print Apartment List\n'+
                             '0) Go Back to Main Menu\n'+
                             'Enter your choice: ')
                if(data.isdigit):
                    data = int(data)
                    if (data >= 0 and data <= 4):
                        user_input = data
                if(user_input == -1):
                    print('Enter a number between 0 and 4 ... please try again\n')


            if(user_input == 1):
                self.add_apartment()


            elif(user_input == 2):
                self.remove_apartment()


            elif(user_input == 3):
```

```
            self.update_apartment()

        elif(user_input == 4):
            self.print_apartment_list()

        elif(user_input == 0):
            print('loading ...')
            break

    def add_apartment(self):
        apartment = Apartment()
        apartment.user_init()
        self.__apartment_list.append(apartment)
        save_apartment(apartment)

    def remove_apartment(self):
        user_choice = -1
        self.print_apartment_list()
        while(user_choice == -1):
            data = input("which apartment would you like to remove? ")
            if(data.isdigit):
                data = int(data)
                if(data > 0 and data <= len(self.__apartment_list)):
                    user_choice = data
            if (user_choice == -1):
                print('Enter a number between 1 and {}... please try
again\n'.format(len(self.__apartment_list)))
        remove_apartment(self.__apartment_list[user_choice-1])
        self.__apartment_list.pop(user_choice-1)
```

```
    def update_apartment(self):
        user_choice = -1
        self.print_apartment_list()
        while(user_choice == -1):
            data = input("which apartment would you like to edit? ")
            if(data.isdigit):
                data = int(data)
                if(data > 0 and data <= len(self.__apartment_list)):
                    user_choice = data
            if (user_choice == -1):
                print('Enter a number between 1 and {}... please try
again\n'.format(len(self.__apartment_list)))

        apartment = self.__apartment_list[user_choice-1]
        remove_apartment(apartment)
        user_choice = self.get_what_to_update()
        if(user_choice == 1):
            apartment.set_number()
        elif(user_choice == 2):
            apartment.set_address()
        elif(user_choice == 3):
            apartment.set_size()
        elif(user_choice == 4):
            apartment.set_num_beds()
        elif(user_choice == 5):
            apartment.set_num_baths()
        elif(user_choice == 6):
```

```
        apartment.set_rent()
    elif(user_choice == 7):
        apartment.set__rental_status()
    elif(user_choice == 8):
        apartment.set_tenant()
    elif(user_choice == 9):
        apartment.set_payment_received()
    save_apartment(apartment)
    print('done')



def get_what_to_update(self):
    choice = -1
    while(choice == -1):
        data = input('1) Update number\n' +
                '2) Update address\n'+
                '3) Update size\n' +
                '4) Update number of beds\n' +
                '5) Update number of baths\n' +
                '6) Update rent amount\n' +
                '7) Update rental status\n' +
                '8) Update tenant\n' +
                '9) Update payments record\n'+
                'Enter your choice: ')
        if(data.isdigit):
            data = int(data)
            if(data > 0 and data <= 9):
                choice = data
        if (choice == -1):
```

```
            print('Enter a number between 1 and 9... please try again\n')

        return choice

    def print_apartment_list(self):
        if(len(self.__apartment_list) == 0):
            print('Apartment list is empty...\n')
        index = 0
        for apartment in self.__apartment_list:
            index = index + 1
            print('{}) {}'.format(index, apartment.print_appartment()))

    def get_apartment_list(self):
        return self.__apartment_list
```

Expense.py

```
from datetime import datetime

class Expense:
    def __init__(self):
       #left empty because initialization will be either from user or db and each has its own method...
        pass


    def db_init(self, amount, payee, payment_date, category):
        self.__amount = amount
        self.__payee = payee
```

```
    self.__payment_date = datetime.strptime(payment_date, '%Y-%m-%d')
    self.__category = category

def user_init(self):
    self.set_amount()
    self.set_payee()
    self.set_payment_date()
    self.set_category()

def set_amount(self):
    try:
        self.__amount = abs(float(input('Enter the amount paid (positive numebr only): $')))
    except ValueError:
        print('amount can only be numbers (e.g: 12.56)... try again\n')
        self.set_amount()

def get_amount(self):
    return self.__amount

def set_payee(self):
    self.__payee = input('Ente the payee name: ')

def get_payee(self):
    return self.__payee

def set_payment_date(self):
    user_date = input('Enter payment date(mm/dd/yyyy): ')
    try:
        self.__payment_date = datetime.strptime(user_date, '%m/%d/%Y')
```

```
        except ValueError:
            print('Incorrect format... try again\n')
            self.set_payment_date()

    def get_payment_date(self):
        return self.__payment_date.date()

    def set_category(self):
        self.__category = input('Enter payment category: ')

    def get_category(self):
        return self.__category

    def print_expense(self):
        return('Amount: ${}\nPayee: {}\nPayment date: {}\nCategory: {}'.format(
            self.__amount, self.__payee, self.get_payment_date(), self.__category))
```

ExpenseRecord.py

```
from Expense import Expense
from sqlite_methods import get_expenses_from_db, remove_expense, save_expense

class ExpenseRecord:
    def __init__(self):
        self.__expense_list = [] #list that will hold all the expenses
        self.__expense_by_categ = {}
        self.__total_expenses = 0
```

```
def load_expenses_from_db(self):
    expenses_list = get_expenses_from_db()
    for expense in expenses_list:
        to_be_added = Expense()
        to_be_added.db_init(expense[0], expense[1], expense[2], expense[3])
        self.__expense_list.append(to_be_added)

def menu(self):
    user_input = -1
    while(user_input != 0):
        user_input = self.get_choice()

        if(user_input == 1):
            self.add_expense()

        elif(user_input == 2):
            self.remove_expense()

        elif(user_input == 3):
            self.print_expense_list()

        elif(user_input == 0):
            print('loading ...')
            break


def get_choice(self):
    user_input = -1
```

```
    while(user_input == -1):
        data = input('1) Add Expense\n'+
                '2) Remove Expense\n'+
                '3) Print Expense List\n'+
                '0) Go Back to Main Menu\n'+
                'Enter your choice: ')
        if(data.isdigit()):
            data = int(data)
            if (data >= 0 and data <= 3):
                user_input = data
        if(user_input == -1):
            print('Enter a number between 0 and 3 ... please try again\n')

    return user_input

def add_expense(self):
    expense = Expense()
    expense.user_init()
    self.__expense_list.append(expense)
    self.get_total_expenses() #update total
    save_expense(expense)
    #self.calc_expenses_by_categ() #update categories

def remove_expense(self):
    self.print_expense_list()
    user_choice = -1
    while(user_choice == -1):
        data = input("which expense would you like to remove? ")
        if(data.isdigit):
```

```
            data = int(data)
            if(data > 0 and data <= len(self.__expense_list)):
                user_choice = data
        if (user_choice == -1):
            print('Enter a number between 1 and {}... please try again\n'.format(len(self.__expense_list)))

    self.__total_expenses = self.__total_expenses - self.__expense_list[user_choice-
1].get_amount()#udpate total
    remove_expense(self.__expense_list[user_choice-1])
    self.__expense_list.pop(user_choice-1)
    self.get_total_expenses() #update total
    # self.calc_expenses_by_categ()#update categories



  def print_expense_list(self):
    if(len(self.__expense_list) == 0):
        print('Expense list is empty...\n')
    i = 0
    for expense in self.__expense_list:
        i = i+1
        print('{}) {}'.format(i, expense.print_expense()))



  def calc_expenses_by_categ(self):
    self.get_expenses_by_categ = {} #start over, does not update exisiting dict
    for expense in self.__expense_list:
        categ = expense.get_category()
        if(categ in self.__expense_by_categ):
            self.__expense_by_categ[categ] = self.__expense_by_categ[categ] + expense.get_amount()
```

```
        else:
            self.__expense_by_categ.update({categ: expense.get_amount()})



    def get_expense_list(self):
        return self.__expense_list

    #ERROR: a second call to get_expenses_by_categ breaks the program "TypeError: 'dict' object is not
callable..."
    def get_expenses_by_categ(self):
        self.calc_expenses_by_categ()
        return self.__expense_by_categ

    def get_total_expenses(self):
        self.__total_expenses = 0
        for expense in self.__expense_list:
            self.__total_expenses = self.__total_expenses  + expense.get_amount()
        return self.__total_expenses
```

Menu.py

```
from TenantList import TenantList
from RentalIncomeRecord import RentalIncomeRecord
from AnnualSummary import AnnualSummary
from ExpenseRecord import ExpenseRecord

class Menu:
    def __init__(self):
```

```
    self.__tenant_list = TenantList()
    self.__tenant_list.load_tenants_from_db()
    self.__rental_rec = RentalIncomeRecord()
    self.__expense_rec = ExpenseRecord()
    self.__expense_rec.load_expenses_from_db()

def print_menu(self):
    user_choice = -1
    while(user_choice != 0):
        while(user_choice == -1):
            data = input('1) Access Tenant List\n' +
                         '2) Access Rental Income Record\n' +
                         '3) Access Expenses Record\n'+
                         '4) Print Annual Summary\n'+
                         '0) Log Out\n'+
                         'Make a selection: ')
            if (data.isdigit):
                data = int(data)
                if(data >= 0 and data <= 4):
                    user_choice = data

            if(user_choice == -1):
                print('Invalid input... Enter a number between 0 and 4..\n')

        if(user_choice == 1):
            user_choice = -1
            self.__tenant_list.menu()

        elif(user_choice == 2):
```

```
            user_choice = -1
            self.__rental_rec.menu()

        elif(user_choice == 3):
            user_choice = -1
            self.__expense_rec.menu()

        elif(user_choice == 4):
            user_choice = -1
            self.__annual_summary = AnnualSummary(self.__expense_rec, self.__rental_rec)
            self.__annual_summary.print_annual_report()

        elif(user_choice == 0):
            print('Logged Out!')
            break
```

RentalIncomeRecord.py

```
from ApartmentList import ApartmentList


class RentalIncomeRecord:
    def __init__(self):
        self.__apartment_list = ApartmentList()
        self.__apartment_list.load_apartments_from_db()
        self.__total_rent_received = 0.0



    def menu(self):
```

```
    user_input = -1
    while(user_input != 0):
        user_input = self.get_user_input()
        if(user_input == 1):
            self.__apartment_list.add_apartment()

        if(user_input == 2):
            self.__apartment_list.remove_apartment()

        if(user_input == 3):
            self.__apartment_list.update_apartment()

        if(user_input == 4):
            self.__apartment_list.print_apartment_list()

        if(user_input == 5):
            self.generate_income_record()

        if(user_input == 0):
            print('loading...\n')

def get_user_input(self):
    choice = -1
    while(choice == -1):
        data = input('1) Add Apartment\n' +
                '2) Remove Apartment\n'+
                '3) Update Existing Apartment\n'+
                '4) Print All Available Apartments\n' +
                '5) Generate and Print Income Record\n' +
```

```
            '0) Done\n'+
            'Enter your choice: ')
        if(data.isdigit):
            data = int(data)
            if(data >= 0 and data <= 5):
                choice = data
        if (choice == -1):
            print('Enter a number between 0 and 5... please try again\n')

    return choice

  def generate_income_record(self):
    print('Month Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec')
    for apartment in self.__apartment_list.get_apartment_list():
        print(str(apartment.get_number()) + ': ' + str(apartment.get_payments_received()))

  def find_total_rent(self):
    for apartment in self.__apartment_list.get_apartment_list():
        self.__total_rent_received = self.__total_rent_received  +
apartment.get_apartment_rent_received_summed()
    return self.__total_rent_received
```

RentalPropertyManagementSystem.py

```
from Menu import Menu
from sqlite_methods import close_db
```

```
class RentalPropertyManagementSystem:
    def __init__(self):
        self.__user_name = 'user'
        self.__password = '0'
        self.__main_menu = Menu()
        self.log_in_menu()

    def log_in_menu(self):
        print('\n\n\n-------------------------\n' +
              'Rental Management System\n'+
              '-------------------------')
        user_choice = -1
        while(user_choice != 0):
            print('1) Log In\n' +
                '2) Change User Name\n' +
                '3) Change Password\n'+
                '0) Quit\n')
            user_choice = self.get_choice()
            if(user_choice == 1):
                if(self.log_in()):
                    print('Logged in..')
                    self.__main_menu.print_menu()

            elif(user_choice == 2):
                if(self.log_in()):
                    self.__user_name = input('Enter your new user name: ')

            elif(user_choice == 3):
                if(self.log_in()):
```

```
            self.__password = input('Enter your new password: ')

        elif(user_choice == 0):
            close_db()
            print('Good Bye!')

def get_choice(self):
    choice = -1
    while(choice == -1):
        data = input('Enter your choice: ')
        if(data.isdigit()):
            data = int(data)
            if(data >= 0 and data <= 3):
                choice = data
        if (choice == -1):
            print('Enter a number between 0 and 3... please try again\n')

    return choice


def log_in(self):
    username = ' '
    password = ' '
    username = input('Enter your username: ')
    password = input('Enter your password: ')
    if(username == self.__user_name and password == self.__password):
        return True
    else:
        print('Invalid user name or password...\n')
```

```
        return False
```

Run.py

```
from RentalPropertyManagementSystem import RentalPropertyManagementSystem


#Run the program
#----------------------------------------
RentalPropertyManagementSystem()
```

sqlite_methods.py

```
import sqlite3
import os
from Expense import Expense
from Tenant import Tenant

conn = sqlite3.connect('rental_management.db')
#conn = sqlite3.connect(':memory:')

c = conn.cursor()

#Run this only the first time running the program
if(os.stat("rental_management.db").st_size == 0):
#if(True):
    #create expenses table
```

```
c.execute("""CREATE TABLE expenses(
    amount REAL,
    payee TEXT,
    date TEXT,
    category TEXT
) """)

#create tenants table
c.execute("""CREATE TABLE tenants(
    first_name TEXT,
    last_name TEXT,
    phone TEXT,
    email TEXT,
    SSN TEXT
) """)

#create apartments table
c.execute("""CREATE TABLE apartments(
    number TEXT,
    address TEXT,
    size REAL,
    num_beds INTEGER,
    num_baths REAL,
    rent REAL,
    rental_status TEXT
) """)

#create apartments rent payments table
c.execute("""CREATE TABLE apartments_rent_payments(
```

```
        apartment_num TEXT,
        jan REAL,
        feb REAL,
        mar REAL,
        apr REAL,
        may REAL,
        jun REAL,
        jul REAL,
        aug REAL,
        sep REAL,
        oct REAL,
        nov REAL,
        dec REAL
    ) """)

    c.execute("""INSERT INTO expenses VALUES (8000, 'Bank', '2020-01-15', 'Mortgage'),
                                (3500, 'City', '2021-12-30', 'Utilities'),
                                (2000, 'Lemonade', '2021-7-19', 'Insurance'),
                                (500, 'Geico', '1919-01-01', 'Insurance')""")

    c.execute("""INSERT INTO tenants VALUES ('Jean', 'Dieb', '818-123-4567', 'jeand@abc.com', 'XXX-XX-
XXXX'),
                                ('Manav', 'Dillon', '818-098-7654', 'manavd@abc.com', 'XXX-XX-XXXX'),
                                ('Al-Muntaser', 'Al-Matani', '818-765-0957', 'almuntasera@abc.com', 'XXX-XX-
XXXX'),
                                ('Phuong', 'Nguyen', '000-000-0000', 'phuongn@abc.com', 'XXX-XX-XXXX')""")

    c.execute("""INSERT INTO apartments VALUES ('141', 'Warren Street, Colton, CA', 1000, 2, 2, 2300,
'False'),
                                  ('551', 'Halifax Drive, Carson, CA', 850, 2, 1, 1900, 'False'),
```

```
                                ('9280', 'Ivy Road Wilmington, CA', 1200, 3, 2.5, 3000, 'False')""")


    c.execute("""INSERT INTO apartments_rent_payments VALUES ('141' ,1000, 1500, 1500, 1000, 800,
1200, 1000, 1500, 1500, 1000, 800, 1200 ),
                                        ('551', 1900, 1900, 1900, 1900, 1900, 1900, 1900, 1900, 1900, 1900,
1900, 1900),
                                        ('9280', 3000, 3000, 3000, 3000, 3000, 3000, 3000, 3000, 3000, 3000,
3000, 3000)""")
    conn.commit()









def get_expenses_from_db():
    c.execute("SELECT * FROM expenses")
    return c.fetchall()


def save_expense(expense):
    with conn:
        c.execute("INSERT INTO expenses VALUES (:amount, :payee, :date, :category)",
        {'amount': expense.get_amount(), 'payee': expense.get_payee(), 'date':
expense.get_payment_date(), 'category': expense.get_category()})


def remove_expense(expense):
    with conn:
        c.execute("DELETE FROM expenses WHERE amount = :amount AND payee = :payee AND date =
:date",
```

```
        {'amount': expense.get_amount(), 'payee': expense.get_payee(),
'date':expense.get_payment_date()})



def get_tenants_from_db():
    c.execute("SELECT * FROM tenants")
    return c.fetchall()



def save_tenant(tenant):
    with conn:
        c.execute("INSERT INTO tenants VALUES (:first_name, :last_name, :phone, :email, :SSN)",
        {'first_name': tenant.get_first_name(), 'last_name': tenant.get_last_name(), 'phone':
tenant.get_phone(),
        'email': tenant.get_email(), 'SSN': tenant.get_SSN()})


def remove_tenant(tenant):
    with conn:
        c.execute("DELETE FROM tenants WHERE last_name = :last_name AND SSN = :SSN",
        {'last_name': tenant.get_last_name(), 'SSN': tenant.get_SSN()})



def get_apartments_form_db():
    c.execute("SELECT * FROM apartments")
    return(c.fetchall())


def save_apartment(apartment):
    with conn:
        c.execute("INSERT INTO apartments VALUES (:number, :address, :size, :num_beds, :num_baths,
:rent, :rental_status)",
```

```
    {'number': apartment.get_number(), 'address': apartment.get_address(), 'size':
apartment.get_size(),
    'num_beds': apartment.get_num_beds(), 'num_baths': apartment.get_num_baths(),
'rent':apartment.get_rent(), 'rental_status':apartment.get_rental_status()})


def remove_apartment(apartment):
  with conn:
    c.execute("DELETE FROM apartments WHERE number = :number AND address = :address AND rent
= :rent",
    {'number': apartment.get_number(), 'address': apartment.get_address(), 'rent':
apartment.get_rent()})


def get_payments_record_from_db(apartment_num):
  c.execute("SELECT * FROM apartments_rent_payments WHERE apartment_num = :apartment_num",
{'apartment_num': apartment_num})
  return (c.fetchone())


def close_db():
  print('db closed')
  conn.close()
```

Tenant.py

```
class Tenant:
  def __init__(self):
    #left empty because initialization will be either from user or db and each has its own method...
    pass

  def db_init(self, first_name, last_name, phone, email, ssn):
```

```
    self.__first_name = first_name
    self.__last_name = last_name
    self.__phone = phone
    self.__email = email
    self.__SSN = ssn

def user_init(self):
    self.set_first_name()
    self.set_last_name()
    self.set_phone()
    self.set_email()
    self.set_SSN()

def set_first_name(self):
    self.__first_name = input('Enter tenant\'s first name: ')

def get_first_name(self):
    return self.__first_name

def set_last_name(self):
    self.__last_name =  input('Enter tenant\'s last name: ')

def get_last_name(self):
    return self.__last_name

def set_phone(self):
    self.__phone =  input('Enter tenant\'s phone number: ')

def get_phone(self):
```

```
        return self.__phone

    def set_email(self):
        self.__email =  input('Enter tenant\'s email address: ')

    def get_email(self):
        return self.__email

    def set_SSN(self):
        self.__SSN =  input('Enter tenant\'s Social Security Number: ')

    def get_SSN(self):
        return self.__SSN

    def set_apartment(self, apartment):
        self.__apartment = apartment
        apartment.set_tenant(self)

    def get_apartment(self):
        return self.__apartment

    def print_tenant_apartment_pair(self):
        print(str(self.get_first_name()) + ' ' + str(self.get_last_name()) + '    ' +
str(self.__apartment.get_number()))

    def print_tenant(self):
        return('First name: {} \nLast name: {}\nPhone: {}\nEmail: {}\nSocial Security Number: {}'.format(
            self.__first_name, self.__last_name, self.__phone, self.__email, self.__SSN))
```

TenantList.py

```
from Tenant import Tenant
from sqlite_methods import get_tenants_from_db, save_tenant, remove_tenant


class TenantList:
    def __init__(self):
        self.__tenant_list = [] #list that will hold the tenants...



    def load_tenants_from_db(self):
        tenant_list = get_tenants_from_db()
        for tenant in tenant_list:
            to_be_added = Tenant()
            to_be_added.db_init(tenant[0], tenant[1], tenant[2], tenant[3], tenant[4])
            self.__tenant_list.append(to_be_added)

    def menu(self):
        user_input = -1
        while(user_input != 0):
            user_input = self.get_choice()
            if(user_input == 1):
                self.add_tenant()

            elif(user_input == 2):
                self.remove_tenant()

            elif(user_input == 3):
```

```
            self.update_tenant()

        elif(user_input == 4):
            self.print_tenant_list()

        elif(user_input == 0):
            print('loading ...')
            return


def get_choice(self):
    user_input = -1
    while(user_input == -1):
        data = input('1) Add Tenant\n'+
                     '2) Remove Tenant\n'+
                     '3) Update Existing Tenant\n'+
                     '4) Print Tenant List\n'+
                     '0) Go Back to Main Menu\n'+
                     'Enter your choice: ')
        if(data.isdigit()):
            data = int(data)
            if (data >= 0 and data <= 4):
                user_input = data
        if(user_input == -1):
            print('Enter a number between 0 and 4 ... please try again\n')

    return user_input
```

```
def add_tenant(self): ##fix here
    tenant = Tenant()
    tenant.user_init()
    self.__tenant_list.append(tenant)
    save_tenant(tenant)

def remove_tenant(self):
    user_choice = -1
    self.print_tenant_list()
    while(user_choice == -1):
        data = input("which tenant would you like to remove? ")
        if(data.isdigit()):
            data = int(data)
            if(data > 0 and data <= len(self.__tenant_list)):
                user_choice = data
        if (user_choice == -1):
            print('Enter a number between 1 and {}... please try again\n'.format(len(self.__tenant_list)))
    remove_tenant(self.__tenant_list[user_choice-1])
    self.__tenant_list.pop(user_choice-1)


def update_tenant(self):
    user_choice = -1
    self.print_tenant_list()
    while(user_choice == -1):
        data = input("which tenant would you like to edit? ")
        if(data.isdigit()):
            data = int(data)
            if(data > 0 and data <= len(self.__tenant_list)):
```

```
            user_choice = data
        if (user_choice == -1):
            print('Enter a number between 1 and {}... please try again\n'.format(len(self.__tenant_list)))

    tenant = self.__tenant_list[user_choice-1]
    remove_tenant(tenant)#remove the tenant and then re add it after getting updated
    user_choice = self.get_what_to_update()
    if(user_choice == 1):
        tenant.set_first_name()
    elif(user_choice == 2):
        tenant.set_last_name()
    elif(user_choice == 3):
        tenant.set_phone()
    elif(user_choice == 4):
        tenant.set_email()
    elif(user_choice == 5):
        tenant.set_SSN()
    elif(user_choice == 6):
        None#tenant.set_apartment() needs to take apartment from the user..
    save_tenant(tenant)
    print('done')


def get_what_to_update(self):
    choice = -1
    while(choice == -1):
        data = input('1) Update first name\n' +
            '2) Update last name\n'+
            '3) Update phone\n' +
```

```
            '4) Update email\n' +
            '5) Update SSN\n') # +
            #'6) Update rent amount\n')

        if(data.isdigit()):
            data = int(data)
            if(data > 0 and data <= 6):
                choice = data
        if (choice == -1):
            print('Enter a number between 1 and 6... please try again\n')

    return choice

def print_tenant_list(self):
    if(len(self.__tenant_list) == 0):
        print('Tenant list is empty...\n')
    index = 0
    for tenant in self.__tenant_list:
        index = index + 1
        print('{}) {}'.format(index, tenant.print_tenant()))

def get_tenant_list(self):
    return self.__tenant_list
```

Unit Testing

The following unit test checks whether the inputs the Landlord puts in are correct and actually get stored within the system. This was done by importing the Tenant.py and TenantList.py which contain the definitions/methods that associate with adding a tenant. As can be seen at the terminal in the proof pycharm indicates that everything is “OK” meaning all tests passed.

```
class testtenant(unittest.TestCase):
    def test_add(self):
        q = Tenant.Tenant()
        result1 = q.set_first_name()
        result1 = q.get_first_name()
        self.assertEqual(result1, "AlMuntaser")
        result2 = q.set_last_name()
        result2 = q.get_last_name()
        self.assertEqual(result2, "AlMatani")
        result3 = q.set_phone()
        result3 = q.get_phone()
        self.assertEqual(result3, "000-000-0000")
        result4 = q.set_email()
        result4 = q.get_email()
        self.assertEqual(result4, "ALMuntaser.AlMatani@gmail.com")
        result5 = q.set_SSN()
        result5 = q.get_SSN()
        self.assertEqual(result5, "XXX-69-XXX")



if __name__ == '__main__':
    unittest.main()
```

```
C:\Users\monti\AppData\Local\Programs\Python\Python39\python.exe C:/Users/monti/Downloads/RentalPropertyManagementSystem/Unit_Test.py
Enter tenant's first name: AlMuntaser
Enter tenant's last name: AlMatani
Enter tenant's phone number: 000-000-0000
Enter tenant's email address: AlMuntaser.AlMatani@gmail.com
Enter tenant's Social Security Number: XXX-69-XXX
.
----------------------------------------------------------------------
Ran 1 test in 28.563s

OK

Process finished with exit code 0
```

Code for the Unit Test

```
import unittest
import Tenant
import TenantList

class testtenant(unittest.TestCase):
    def test_add(self):
        q = Tenant.Tenant()
        result1 = q.set_first_name()
        result1 = q.get_first_name()
        self.assertEqual(result1, "AlMuntaser")
        result2 = q.set_last_name()
        result2 = q.get_last_name()
        self.assertEqual(result2, "AlMatani")
        result3 = q.set_phone()
        result3 = q.get_phone()
        self.assertEqual(result3, "000-000-0000")
        result4 = q.set_email()
        result4 = q.get_email()
        self.assertEqual(result4, "ALMuntaser.AlMatani@gmail.com")
        result5 = q.set_SSN()
        result5 = q.get_SSN()
        self.assertEqual(result5, "XXX-69-XXX")


if __name__ == '__main__':
    unittest.main()
```

Runtime Output

```
-------------------------
Rental Management System
-------------------------
1) Log In
2) Change User Name
3) Change Password
0) Quit

Enter your choice: 1
Enter your username: user
Enter your password: 0
Logged in..
1) Access Tenant List
2) Access Rental Income Record
3) Access Expenses Record
4) Print Annual Summary
0) Log Out
Make a selection: 1
1) Add Tenant
2) Remove Tenant
3) Update Existing Tenant
4) Print Tenant List
0) Go Back to Main Menu
Enter your choice: 4
1) First name: Jean
Last name: Dieb
Phone: 818-123-4567
Email: jeand@abc.com
Social Security Number: XXX-XX-XXXX
2) First name: Manav
Last name: Dillon
Phone: 818-098-7654
Email: manavd@abc.com
Social Security Number: XXX-XX-XXXX
```

```
3) First name: Al-Muntaser
Last name: Al-Matani
Phone: 818-765-0957
Email: almuntasera@abc.com
Social Security Number: XXX-XX-XXXX
4) First name: Phuong
Last name: Nguyen
Phone: 000-000-0000
Email: phuongn@abc.com
Social Security Number: XXX-XX-XXXX
1) Add Tenant
2) Remove Tenant
3) Update Existing Tenant
4) Print Tenant List
0) Go Back to Main Menu
Enter your choice: 0
loading ...
1) Access Tenant List
2) Access Rental Income Record
3) Access Expenses Record
4) Print Annual Summary
0) Log Out
Make a selection: _
```

```
Make a selection: 2
1) Add Apartment
2) Remove Apartment
3) Update Existing Apartment
4) Print All Available Apartments
5) Generate and Print Income Record
0) Done
Enter your choice: 4
1) Apartment number: 141
Address: Warren Street, Colton, CA
Size: 1000.0
# of beds: 2
# of baths: 2.0
Rent: 2300.0
Rented? False
2) Apartment number: 551
Address: Halifax Drive, Carson, CA
Size: 850.0
# of beds: 2
# of baths: 1.0
Rent: 1900.0
Rented? False
3) Apartment number: 9280
Address: Ivy Road Wilmington, CA
Size: 1200.0
# of beds: 3
# of baths: 2.5
Rent: 3000.0
Rented? False
1) Add Apartment
2) Remove Apartment
3) Update Existing Apartment
4) Print All Available Apartments
5) Generate and Print Income Record
0) Done
Enter your choice:
```

```
Enter your choice: 5
141: [1000.0, 1500.0, 1500.0, 1000.0, 800.0, 1200.0, 1000.0, 1500.0, 1500.0, 1000.0, 800.0, 1200.0]
551: [1900.0, 1900.0, 1900.0, 1900.0, 1900.0, 1900.0, 1900.0, 1900.0, 1900.0, 1900.0, 1900.0, 1900.0]
9280: [3000.0, 3000.0, 3000.0, 3000.0, 3000.0, 3000.0, 3000.0, 3000.0, 3000.0, 3000.0, 3000.0, 3000.0]
1) Add Apartment
2) Remove Apartment
3) Update Existing Apartment
4) Print All Available Apartments
5) Generate and Print Income Record
0) Done
Enter your choice:
```

```
1) Access Tenant List
2) Access Rental Income Record
3) Access Expenses Record
4) Print Annual Summary
0) Log Out
Make a selection: 3
1) Add Expense
2) Remove Expense
3) Print Expense List
0) Go Back to Main Menu
Enter your choice: 3
1) Amount: $8000.0
Payee: Bank
Payment date: 2020-01-15
Category: Mortgage
2) Amount: $3500.0
Payee: City
Payment date: 2021-12-30
Category: Utilities
3) Amount: $2000.0
Payee: Lemonade
Payment date: 2021-07-19
Category: Insurance
4) Amount: $500.0
Payee: Geico
Payment date: 1919-01-01
Category: Insurance
1) Add Expense
2) Remove Expense
3) Print Expense List
0) Go Back to Main Menu
Enter your choice: _
```

```
Make a selection: 4
Annual Summary:
------------------
Income:
Rent: 72800.0
Expenses:
{'Mortgage': 8000.0, 'Utilities': 3500.0, 'Insurance': 2500.0}
Balance: 58800.0
1) Access Tenant List
2) Access Rental Income Record
3) Access Expenses Record
4) Print Annual Summary
0) Log Out
Make a selection: _
```

```
Make a selection: 0
Logged Out!
1) Log In
2) Change User Name
3) Change Password
0) Quit

Enter your choice: 0
db closed
Good Bye!
```

Summary

Rental property management is a tool that allows a landlord to keep track of his apartments, tenants, income and expenses.

This program is meant to be used by landlord to log their rental properties data. Landlords can use it to save data about their apartments, tenants, rent received and expenses. The program also provides useful reports upon request to help landlords track their financials and net profit The program is written in Python and backed by a local SQLite database to store and load data.