Names: Harsohail Brar (30041921)
Gary Wu (30038110)
Ryan Holt (30038609)
Course Name: Principles of Software Design
Course Code: ENSF 480
Assignment Number: Term Project Design Document
Submission Date and Time: 09/11/2019

Property Rental Management System (PRMS) Requirements

The **property rental management system** is a rental system that will allow people to post their property to let potential renters view and rent it if desired. This system will be hosted on a server that will be connected to a **database** containing information such as the list of users and listings in the system. When a user or client connects to the system, they will be prompted to enter a username and password and if left empty, they will be assumed to be just a **regular renter** who can search, view different listings created by landlords, and email the landlord if desired. If the user chooses to log in, the system will open a menu with different options depending on their account type.

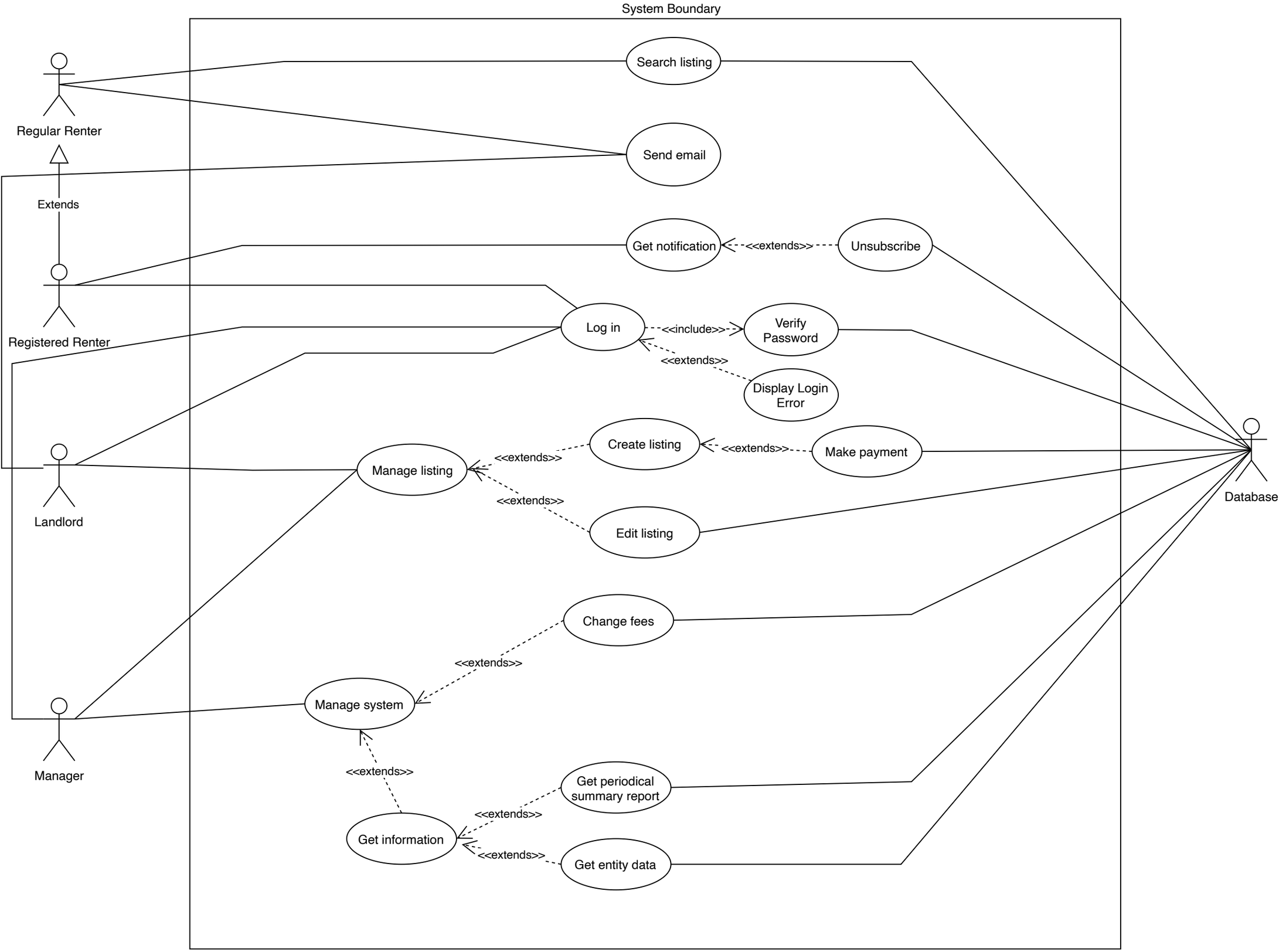
There are 3 types of accounts which are manager, registered renter, and landlord. A **registered renter** can surf the system similar to the regular renter, however, has the ability to receive notifications as new listings are posted matching his/her search history. The **manager** can manage the system though changing the fees or states of different listings in the system. These include the property listing being active, not active, rented, cancelled as well as suspended. They can request the system for a periodical summary report of the system which will have information of the total number of listings, status of these listings, number of houses rented, total number of active listings and the list of houses rented in the desired time period. Finally, the manager can also request information about different users registered to the system and listing stored in the database. Lastly the **landlord** has a menu that will allow them to create, manage, and pay for his/her different listings. The landlord can create different listings but for them to be active, he must pay a certain fee which is set by the manager. He also has the ability to edit these listings such as changing its states which includes active, rented, cancelled, etc.

To implement this property rental management system design, we will need the following system requirements to be met:

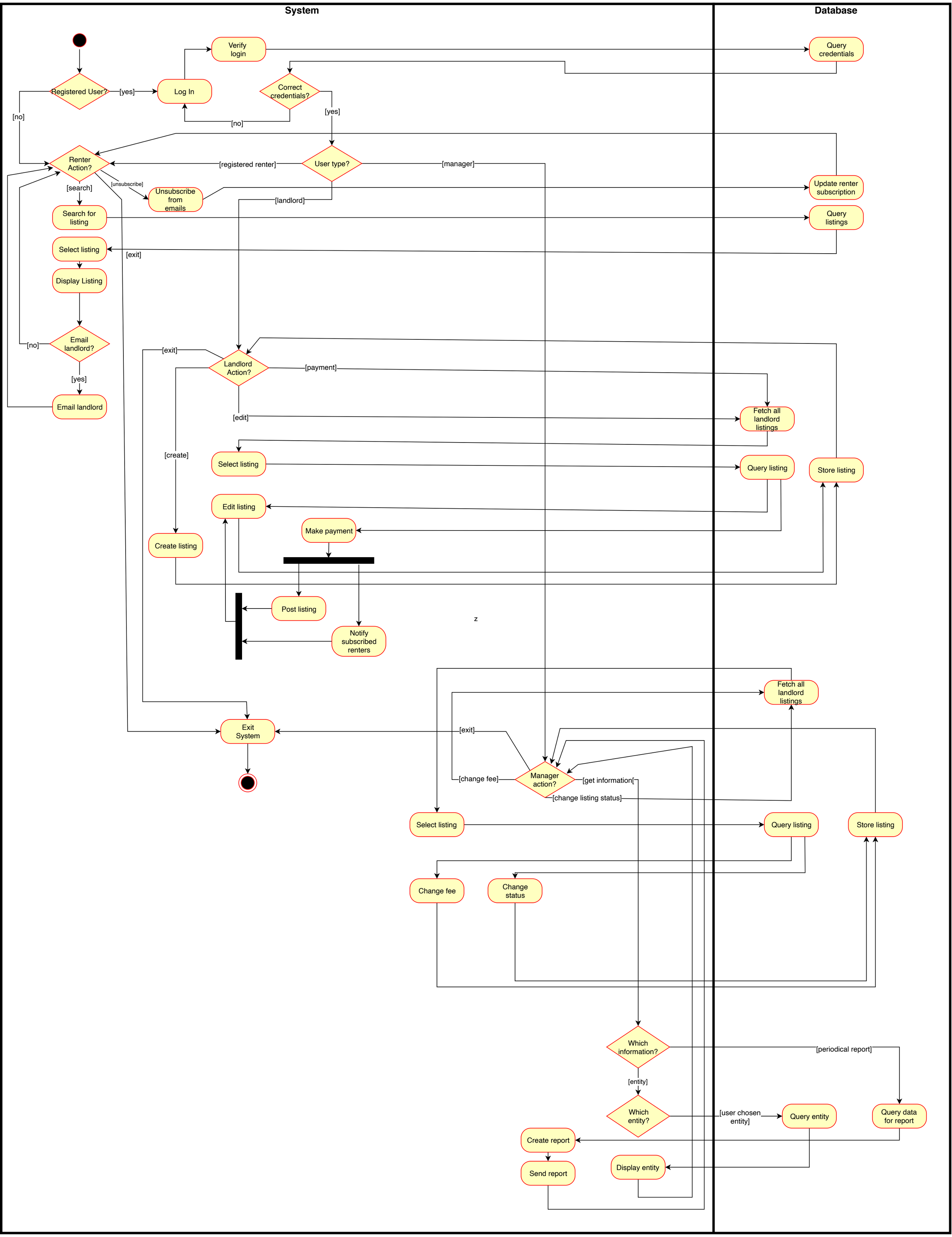
- Users/Client should be able to connect to the system/server
- Users must be able to login with their credentials or as regular renters
- System must be able to connect to a database which store's the system's information
- The following shows the privileges given to each type of user:
    - Regular renter
        - Search for a listing by
            - Apartment, attached/detached house, townhouse, etc.
            - Number of bedrooms
            - Number of bathrooms
            - Furnished/unfurnished
            - City quadrant: SW, NW, NE, SE
        - View listing
        - Email landlord
    - Registered Renter
        - All privileges of regular renter
        - Get notified when new listing matching their searches is posted
    - Landlord
        - Registration of their property to the system

        - Payment of fees to activate listing
        - Edit their listing
    - Manager
        - Change fee by amount or period
        - Get periodical report
        - Get entity data (landlord, renter, or listing)
        - Change listing state (active, rented, cancelled, suspended, etc.)

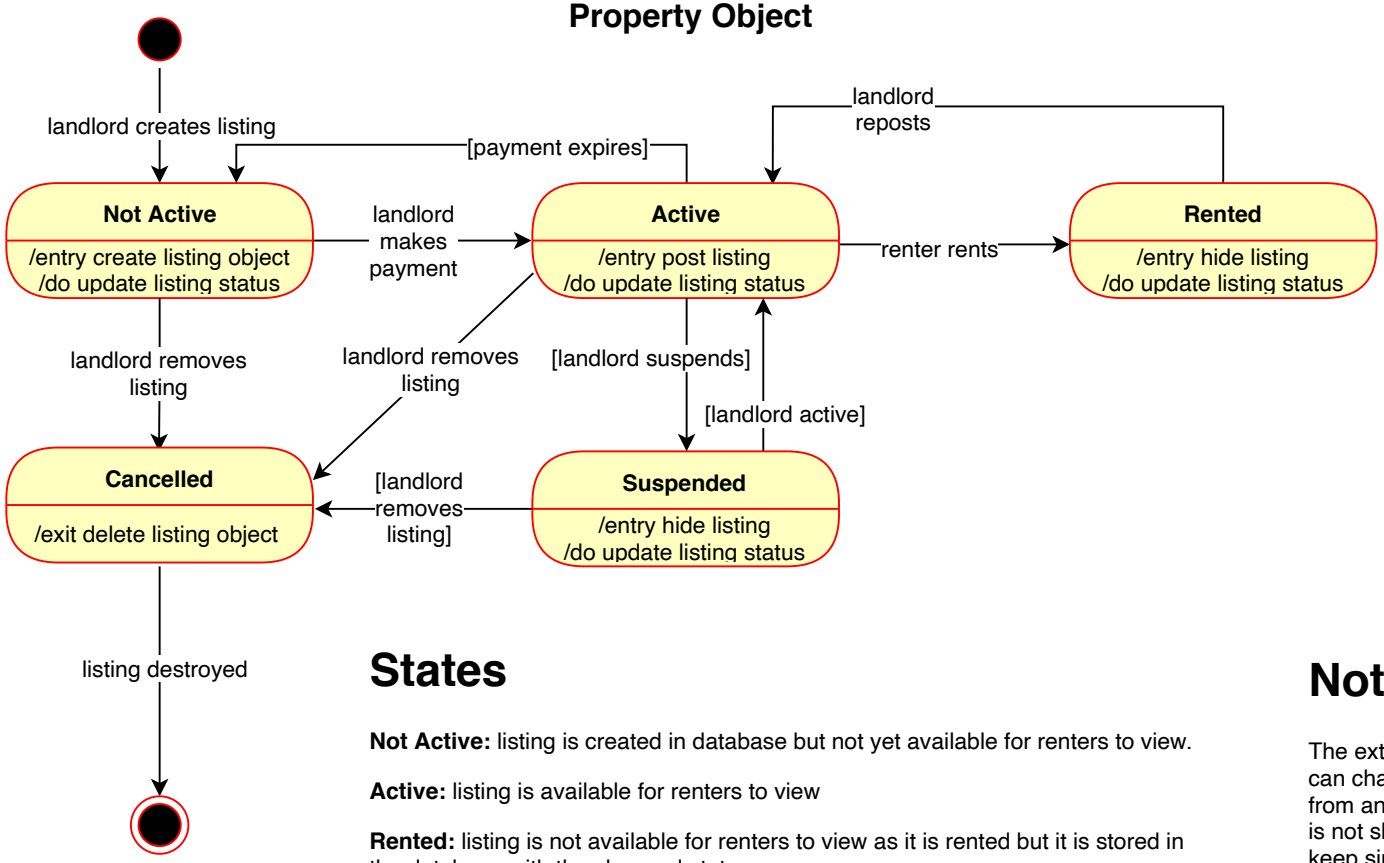
System's Use Case Diagram



System's Activity Diagram



State Transition Diagrams
Property Object



States

Not Active: listing is created in database but not yet available for renters to view.

Active: listing is available for renters to view

Rented: listing is not available for renters to view as it is rented but it is stored in the database with the changed status

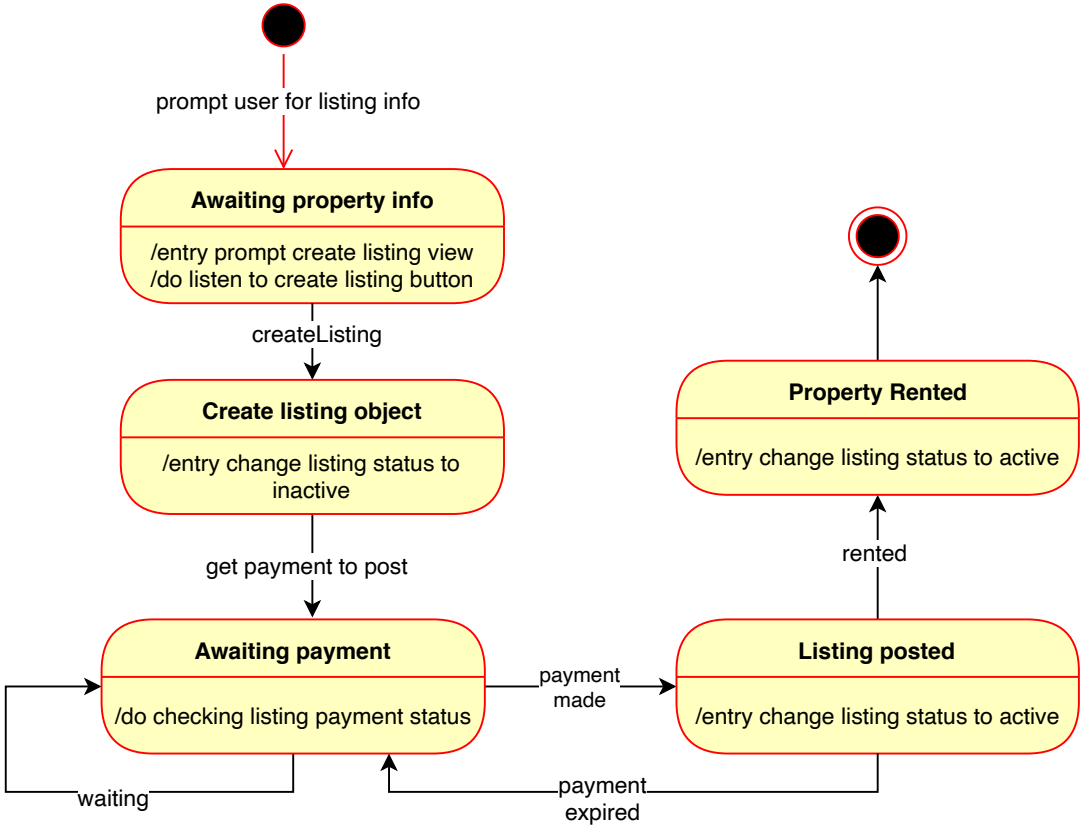
Cancelled: listing is deleted from the system

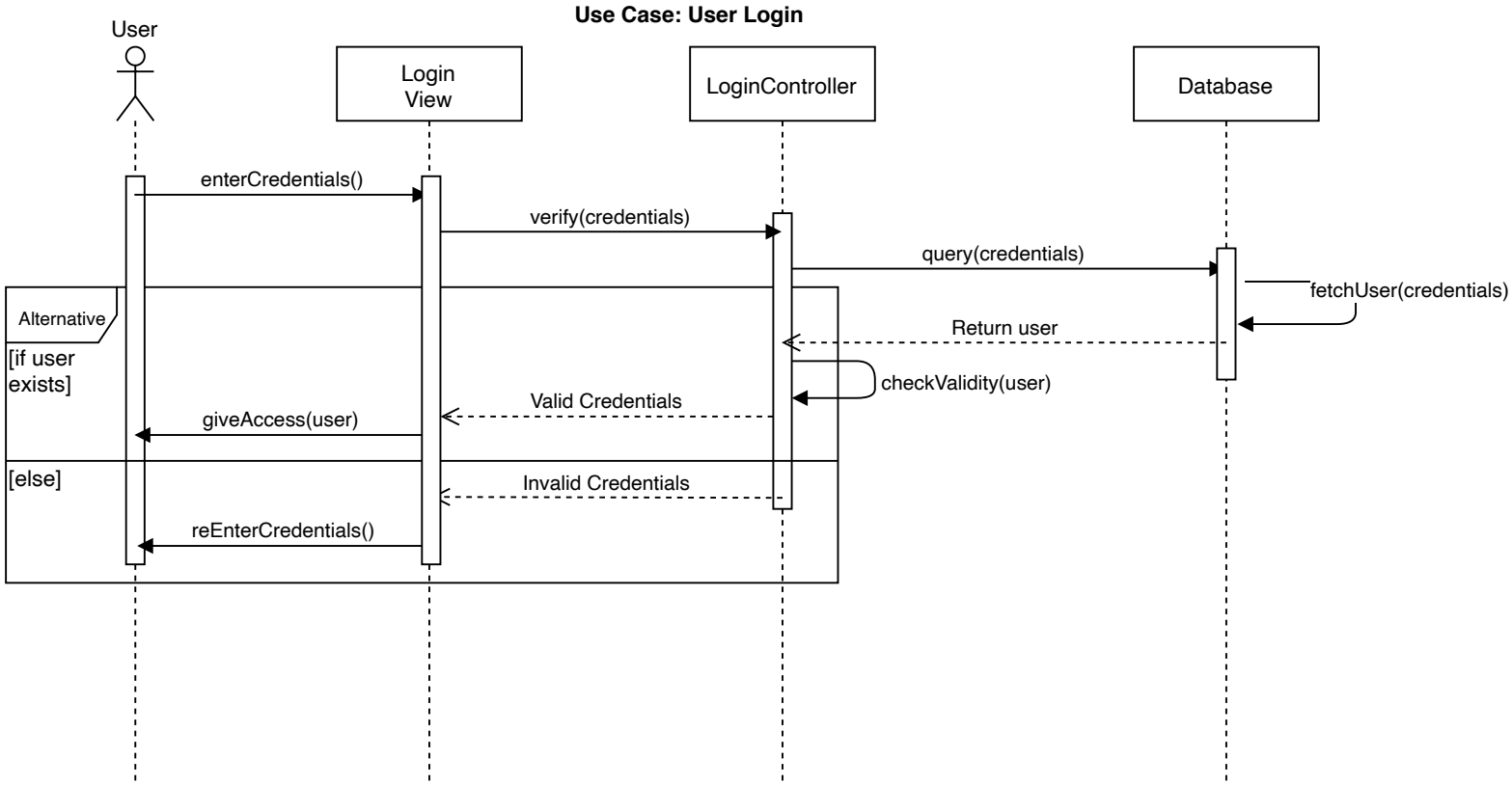
Suspended: landlord suspends the listing temporarly to hide it from renters viewing it. (ex. if landlord is the middle of signing a contract with a potential renter)

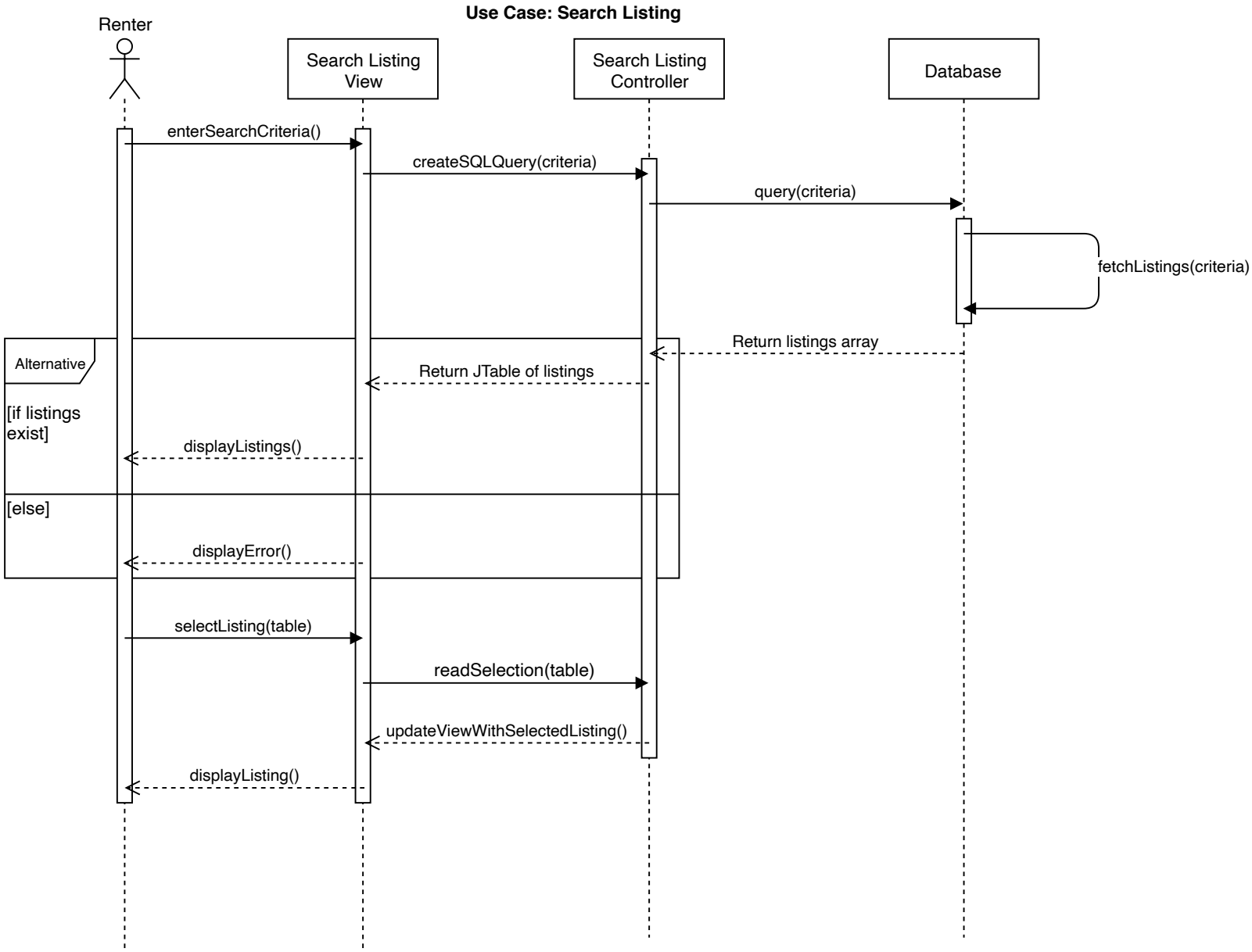
Note

The external stimuli of a manager can change a property obejct from any state to any state which is not shown in the diagram to keep simplicity and clarity

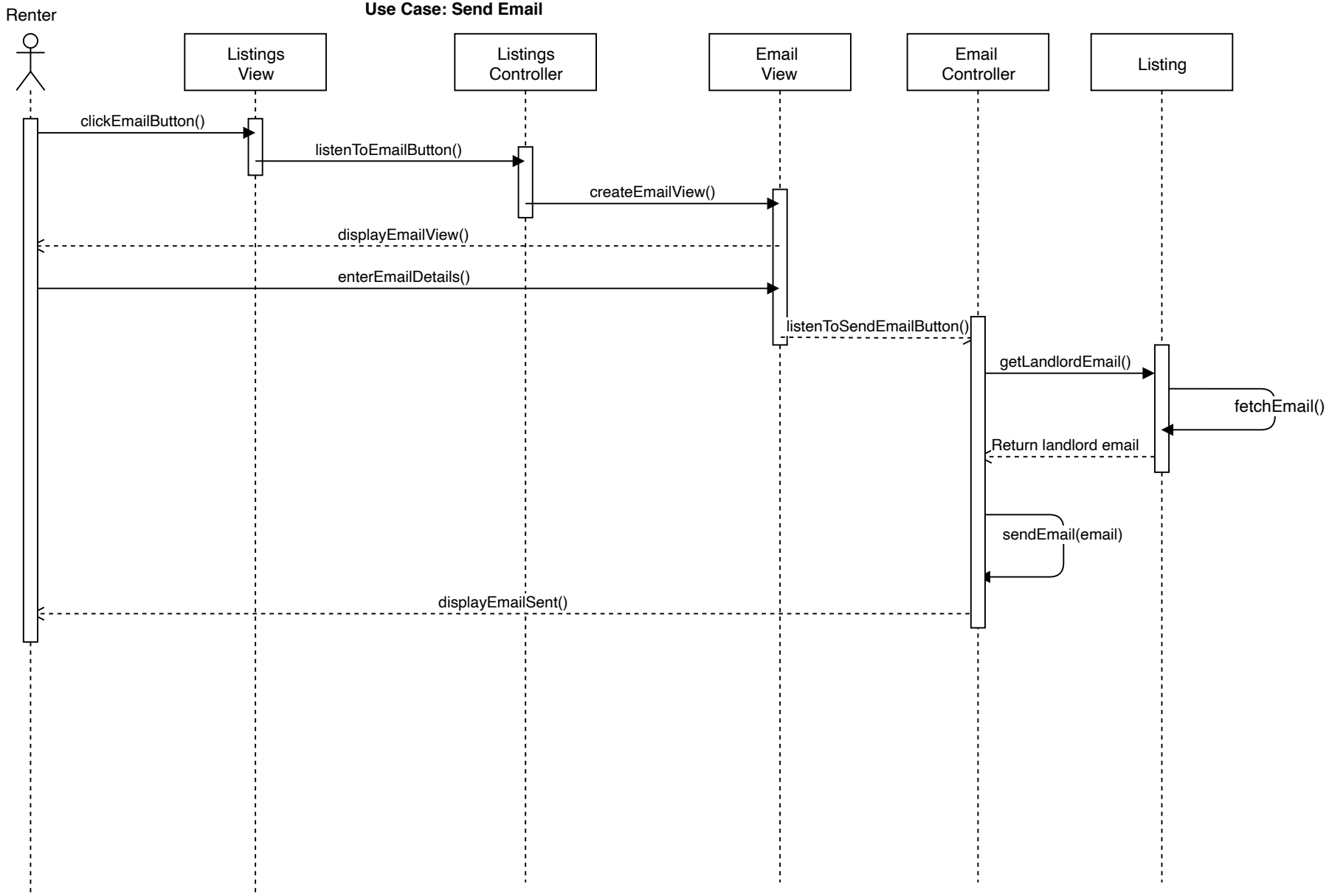
Process that landlord posts his/her property

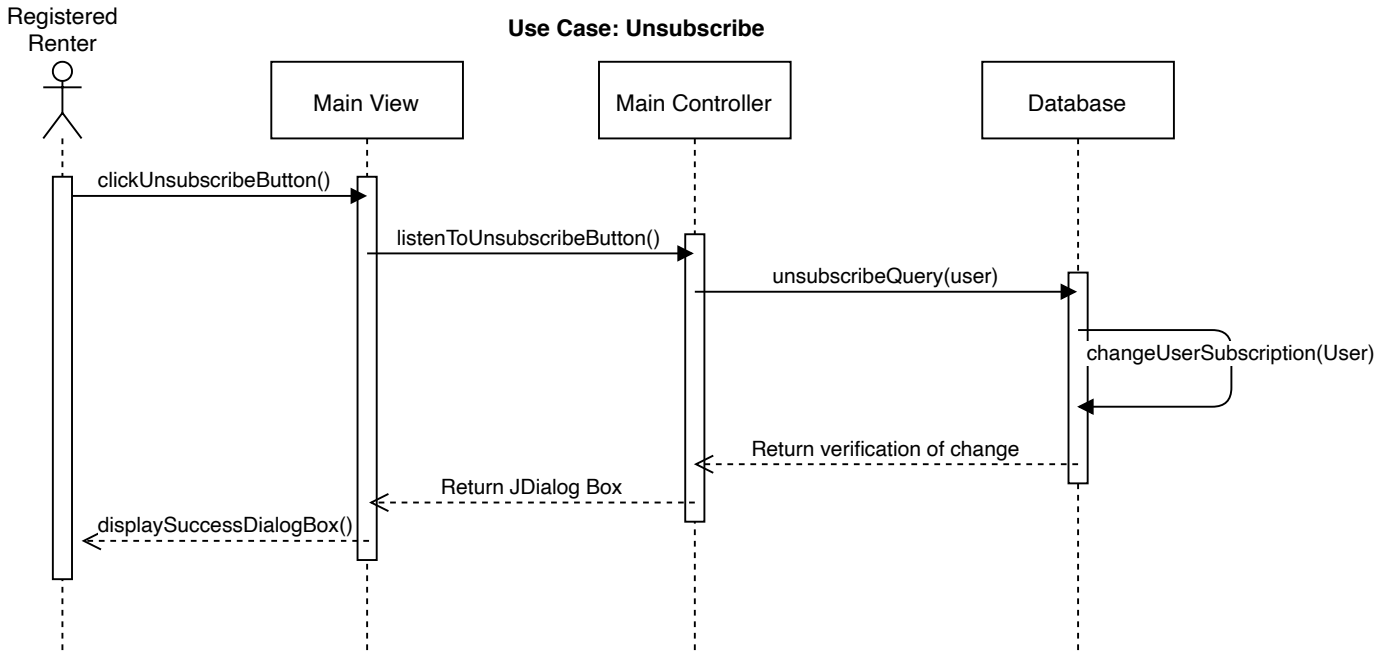


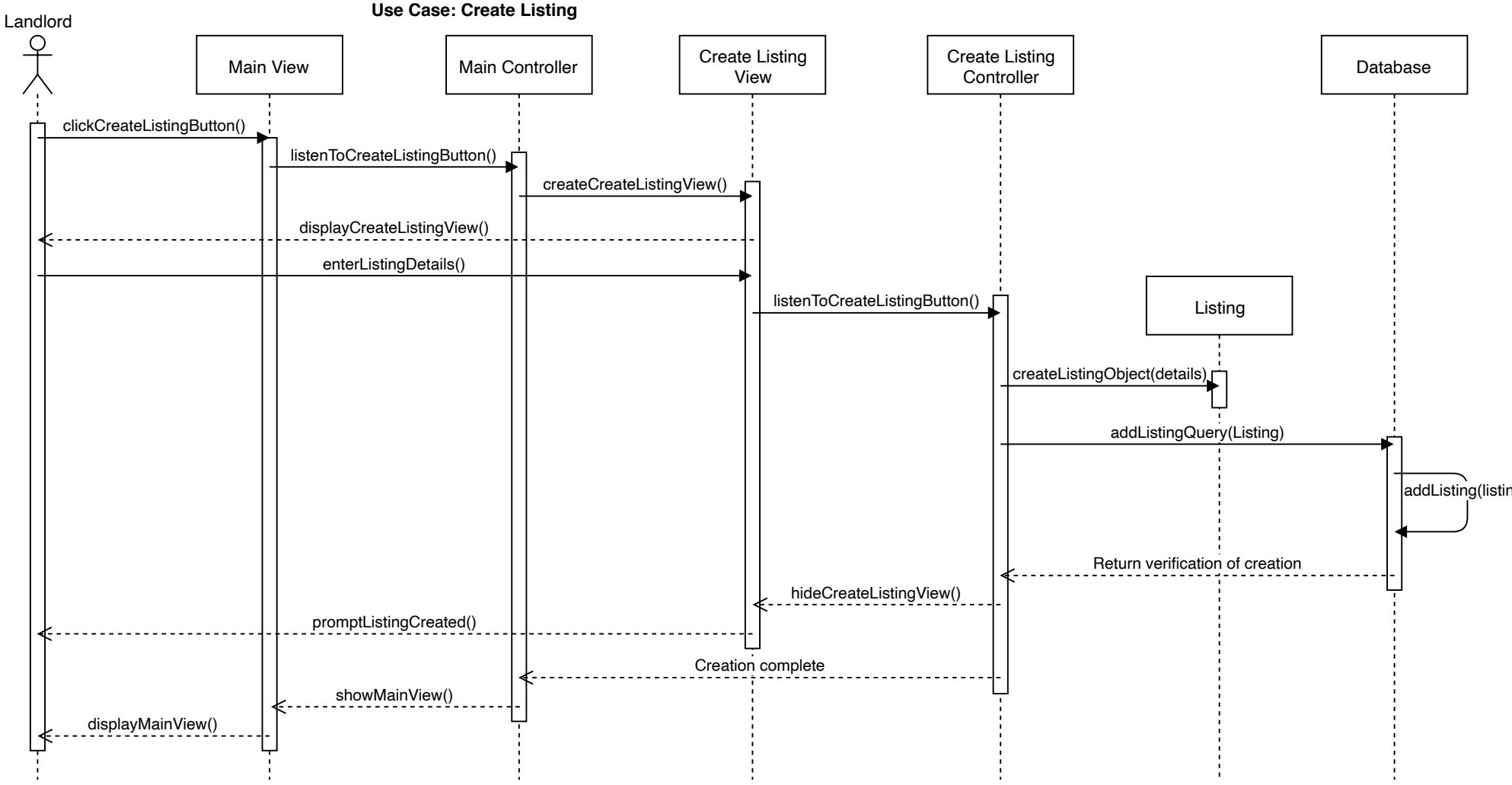

Use Case: User Login
User
Login View
LoginController
Database
enterCredentials()
verify(credentials)
query(credentials)
fetchUser(credentials)
Alternative
[if user exists]
Return user
checkValidity(user)
Valid Credentials
giveAccess(user)
[else]
Invalid Credentials
reEnterCredentials()


Use Case: Search Listing
Renter
Search Listing View
Search Listing Controller
Database
enterSearchCriteria()
createSQLQuery(criteria)
query(criteria)
fetchListings(criteria)
Return listings array
Alternative
Return JTable of listings
[if listings exist]
displayListings()
[else]
displayError()
selectListing(table)
readSelection(table)
updateViewWithSelectedListing()
displayListing()

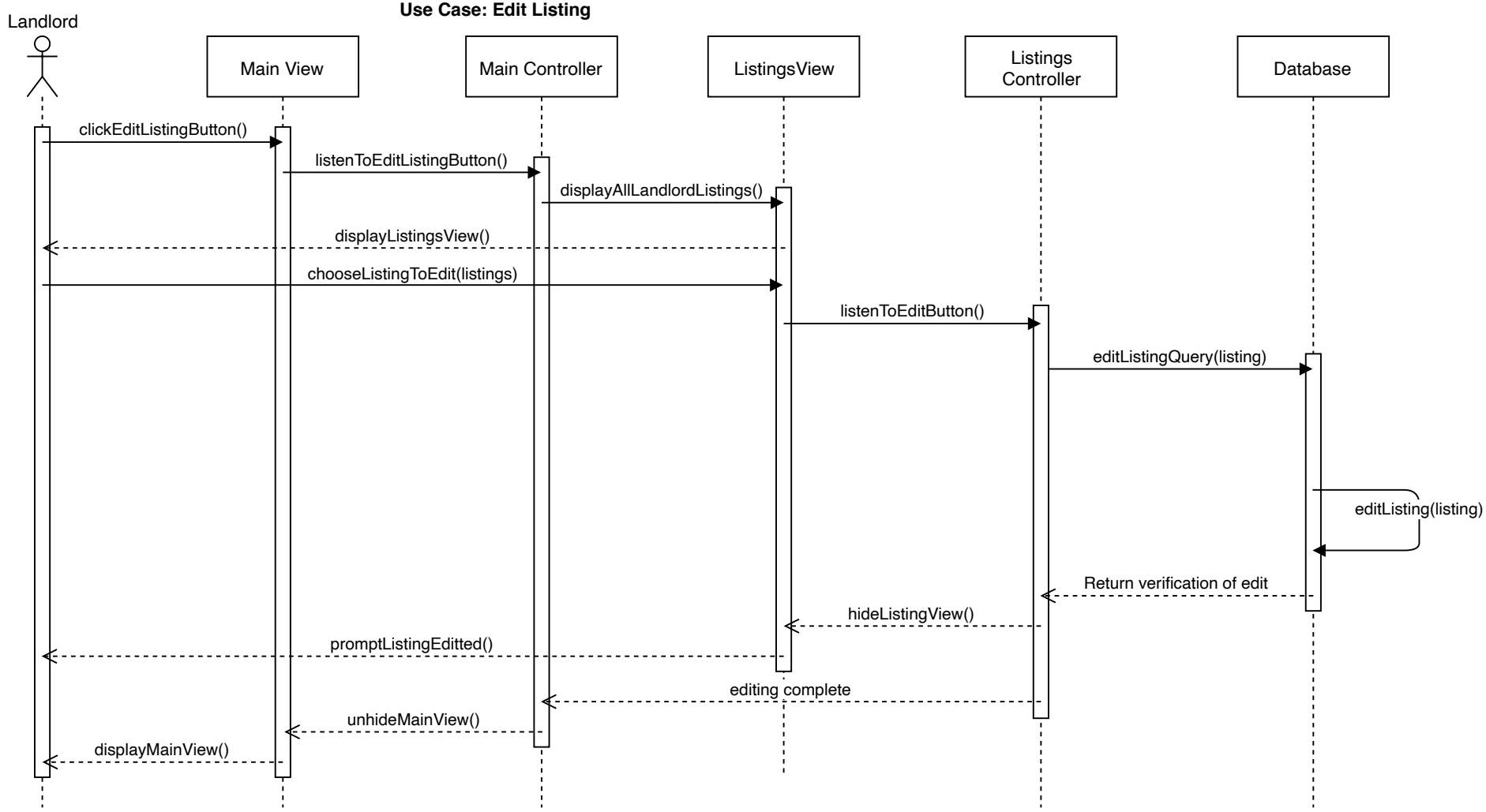
Use Case: Send Email



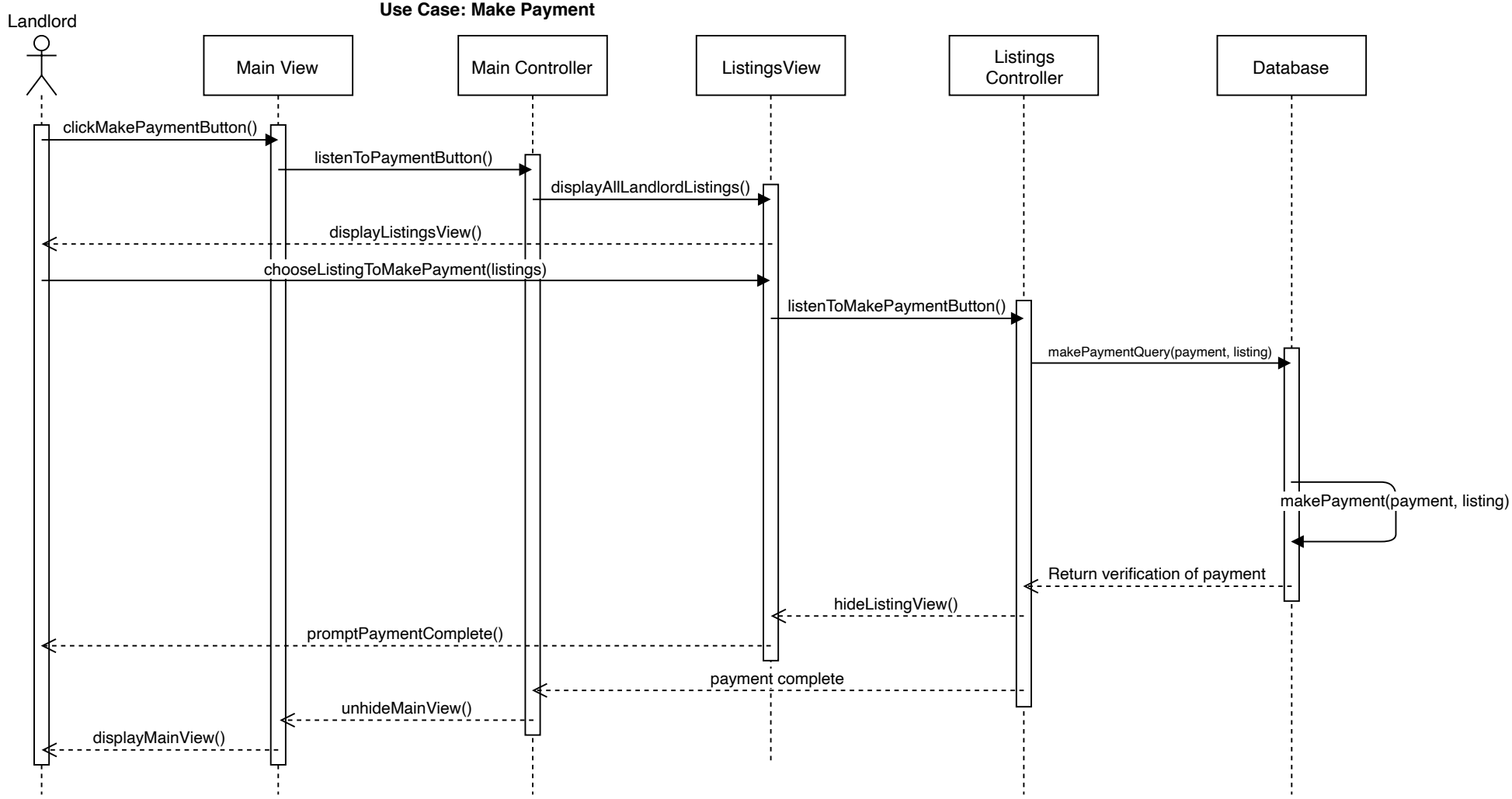

Use Case: Unsubscribe
Registered Renter
Main View
Main Controller
Database
clickUnsubscribeButton()
listenToUnsubscribeButton()
unsubscribeQuery(user)
changeUserSubscription(User)
Return verification of change
Return JDialog Box
displaySuccessDialogBox()


Use Case: Create Listing
Landlord
Main View
Main Controller
Create Listing View
Create Listing Controller
Database
Listing
clickCreateListingButton()
listenToCreateListingButton()
createCreateListingView()
displayCreateListingView()
enterListingDetails()
listenToCreateListingButton()
createListingObject(details)
addListingQuery(Listing)
addListing(listin
Return verification of creation
hideCreateListingView()
promptListingCreated()
Creation complete
showMainView()
displayMainView()

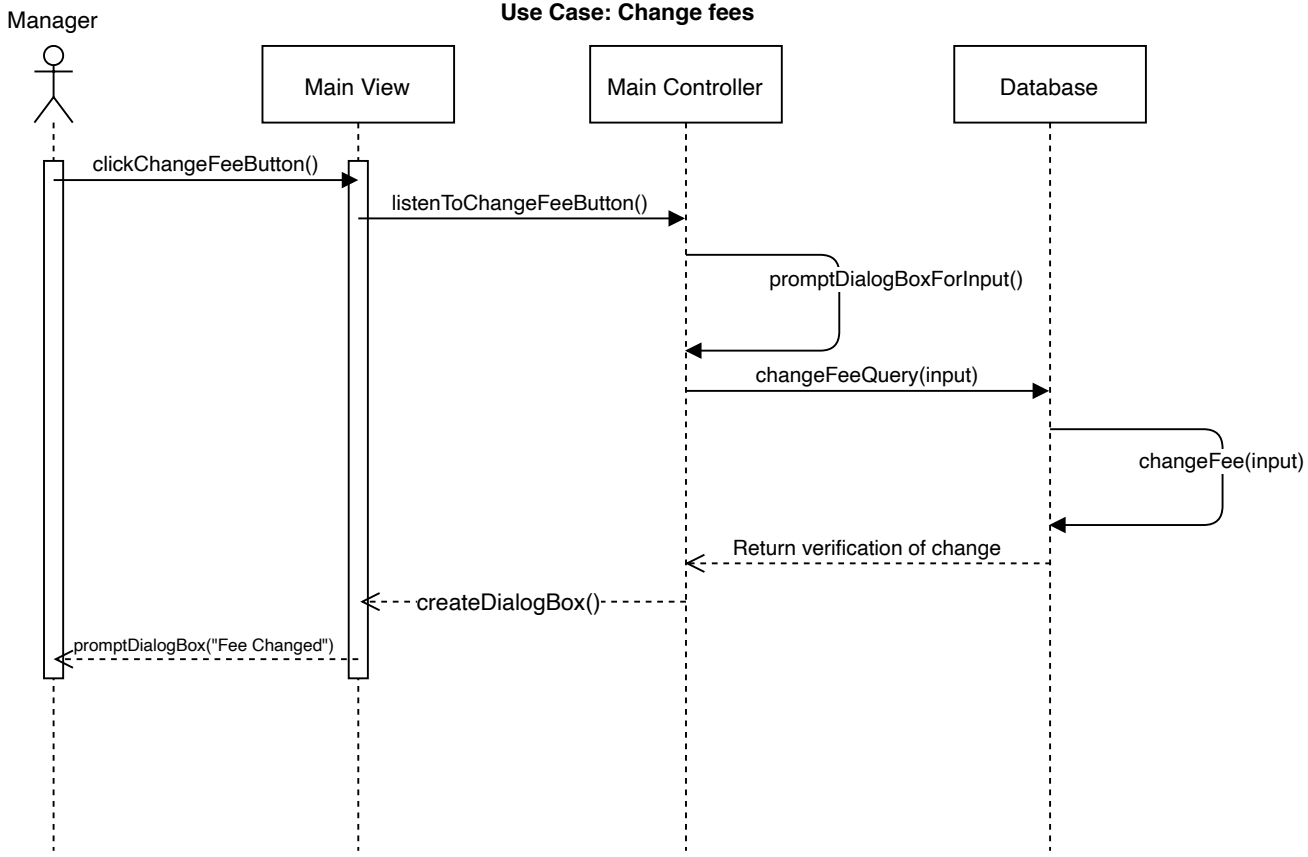
Use Case: Edit Listing

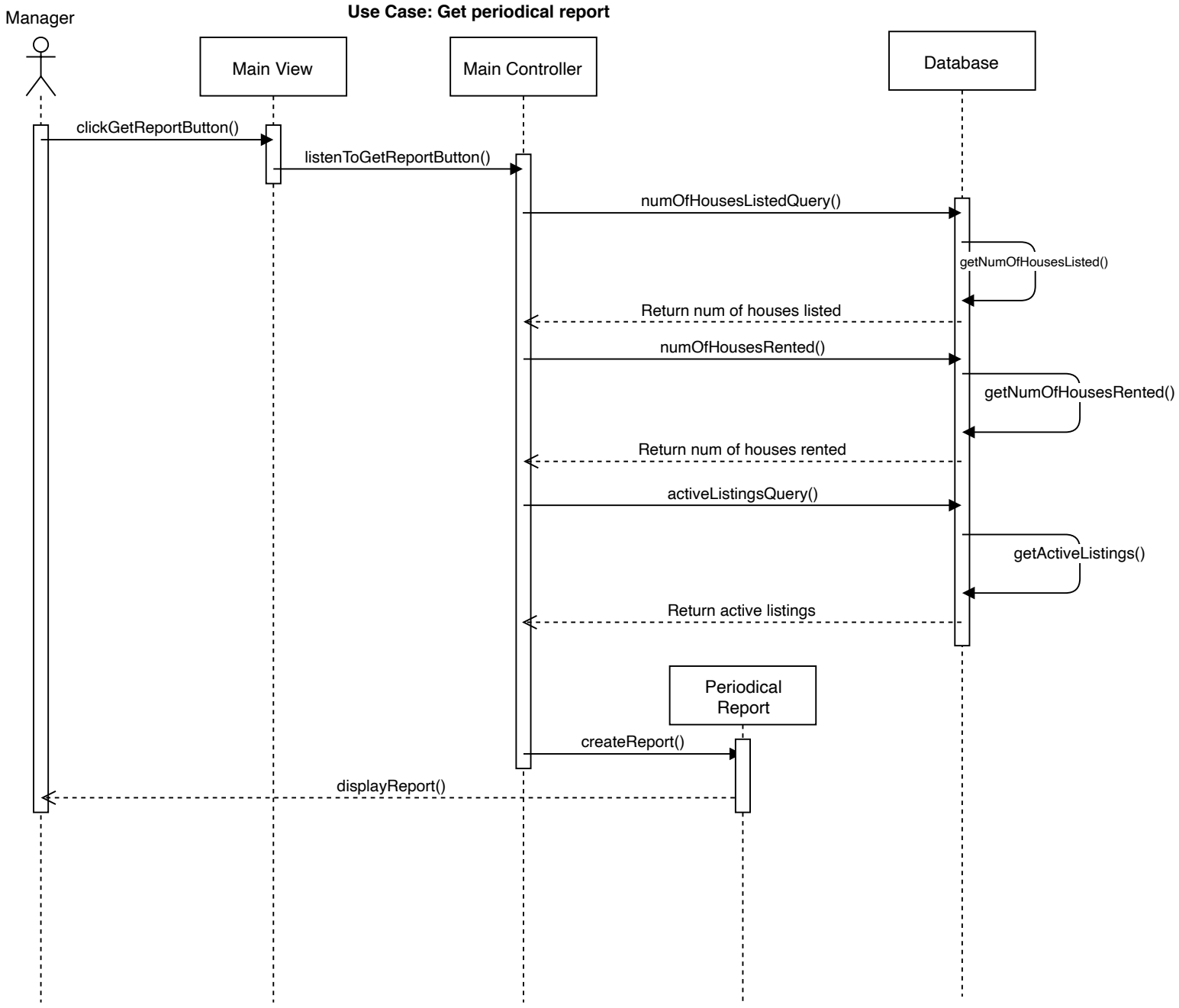


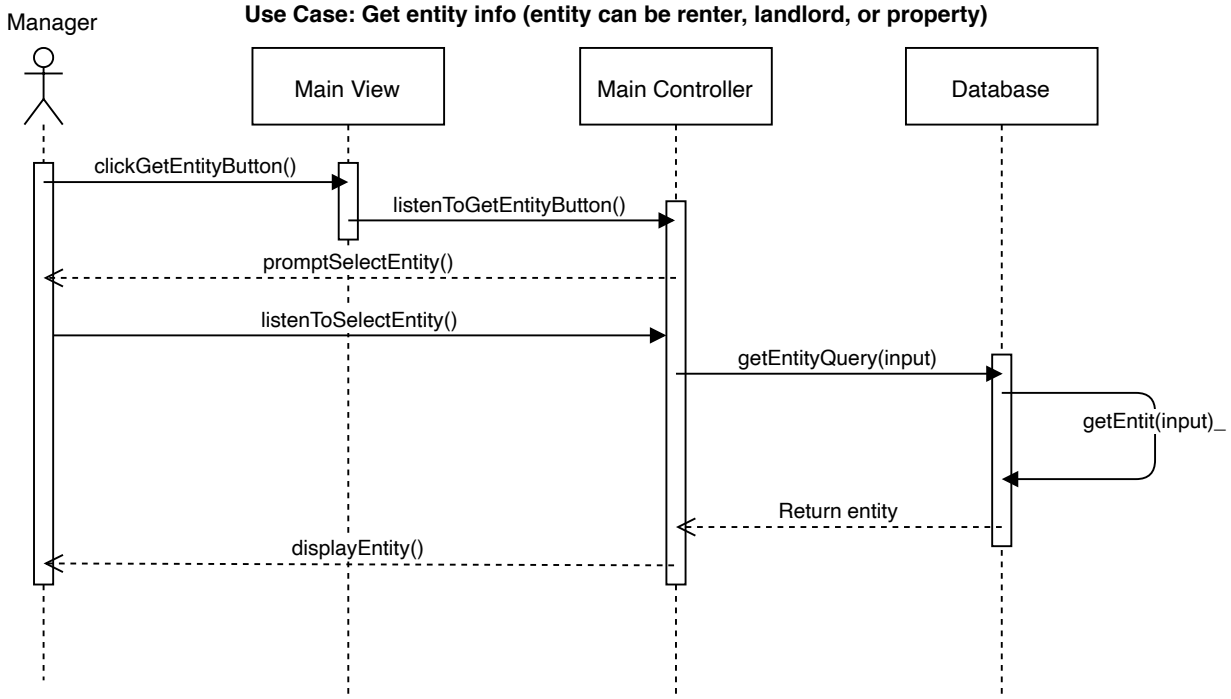
Use Case: Make Payment



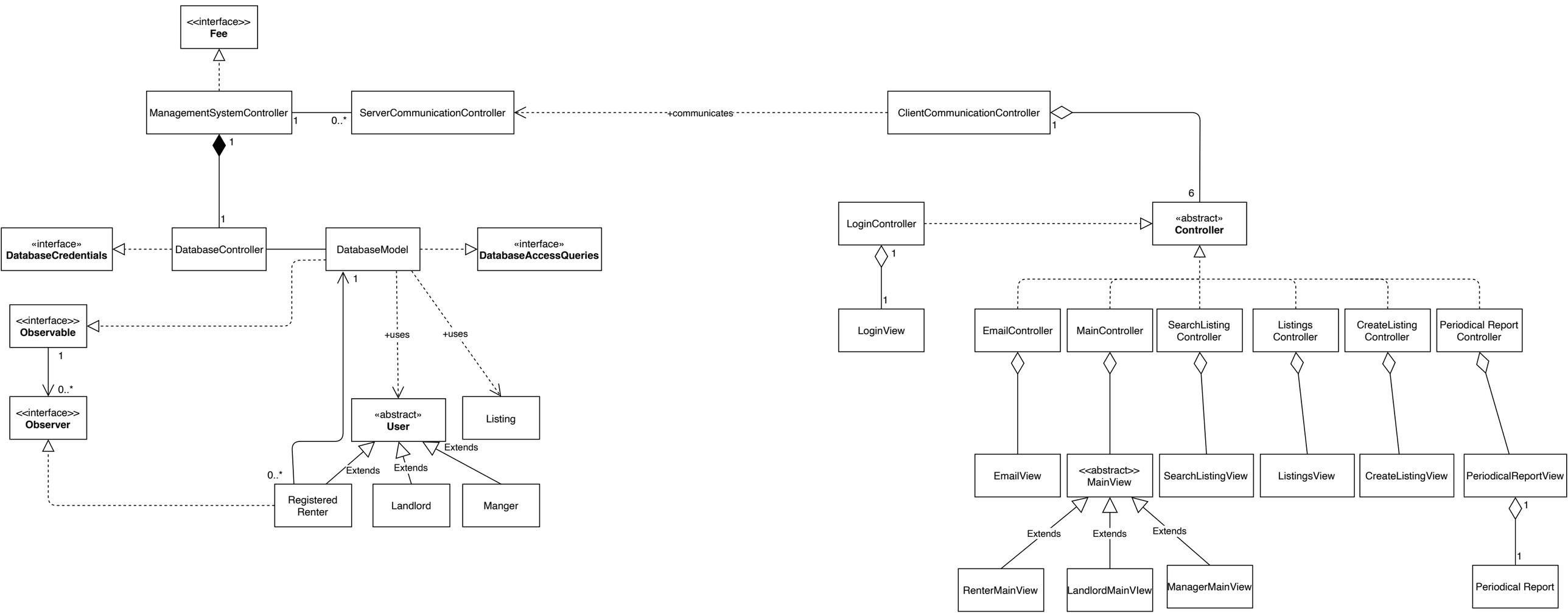
Use Case: Change fees



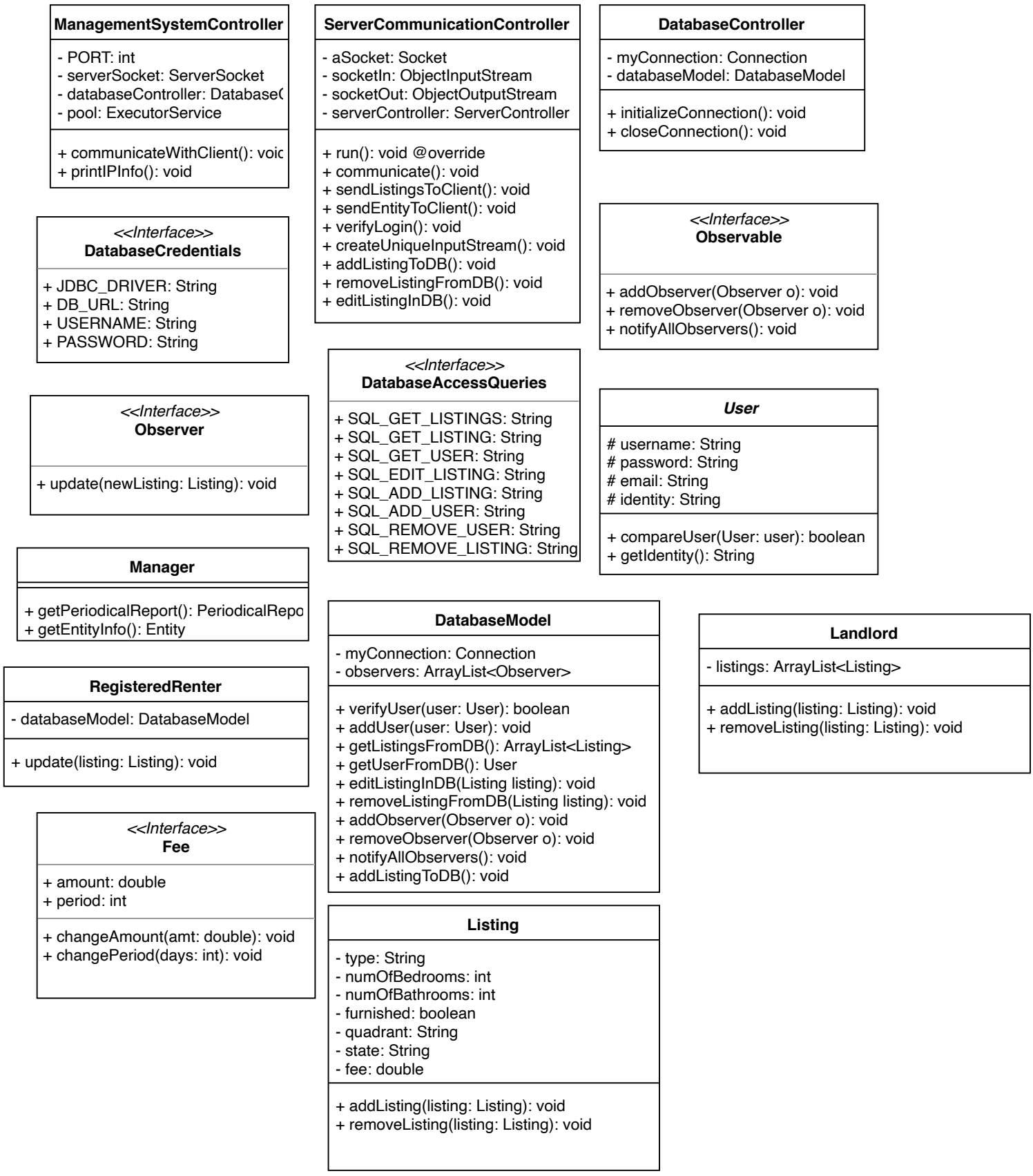

Use Case: Get periodical report
Manager
Main View
Main Controller
Database
clickGetReportButton()
listenToGetReportButton()
numOfHousesListedQuery()
getNumOfHousesListed()
Return num of houses listed
numOfHousesRented()
getNumOfHousesRented()
Return num of houses rented
activeListingsQuery()
getActiveListings()
Return active listings
Periodical Report
createReport()
displayReport()

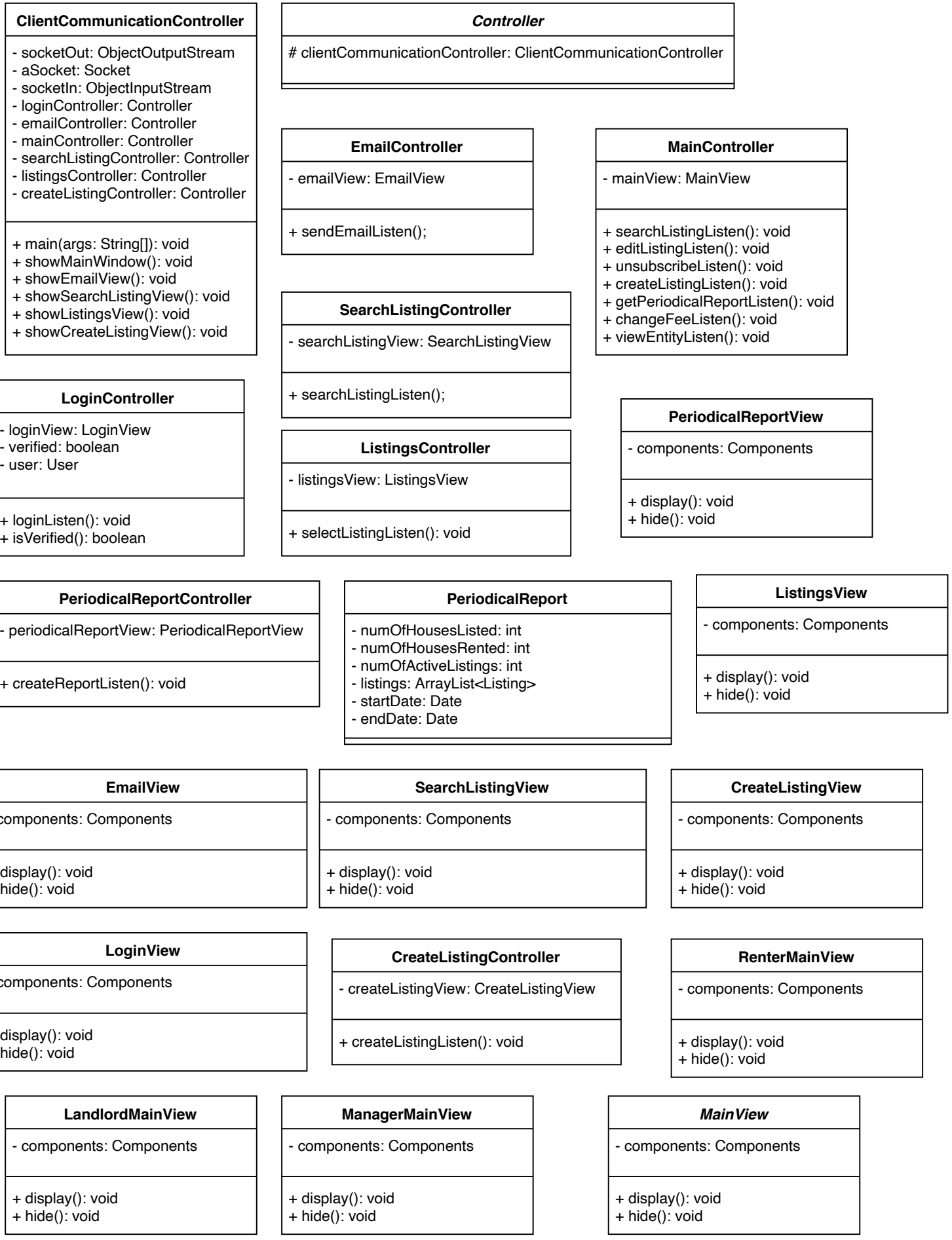

Use Case: Get entity info (entity can be renter, landlord, or property)
Manager
Main View
Main Controller
Database
clickGetEntityButton()
listenToGetEntityButton()
promptSelectEntity()
listenToSelectEntity()
getEntityQuery(input)
getEntit(input)_
Return entity
displayEntity()

Relationships UML

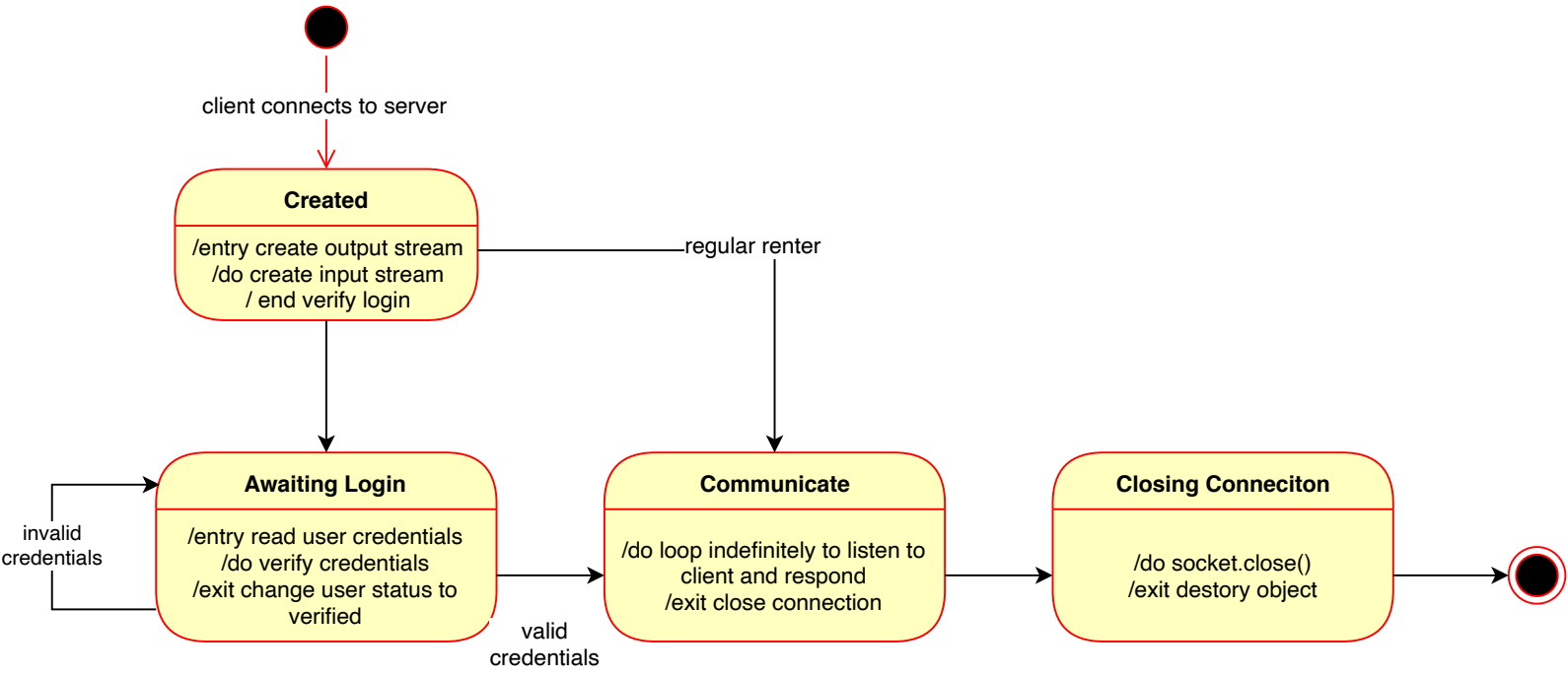


Attributes UML

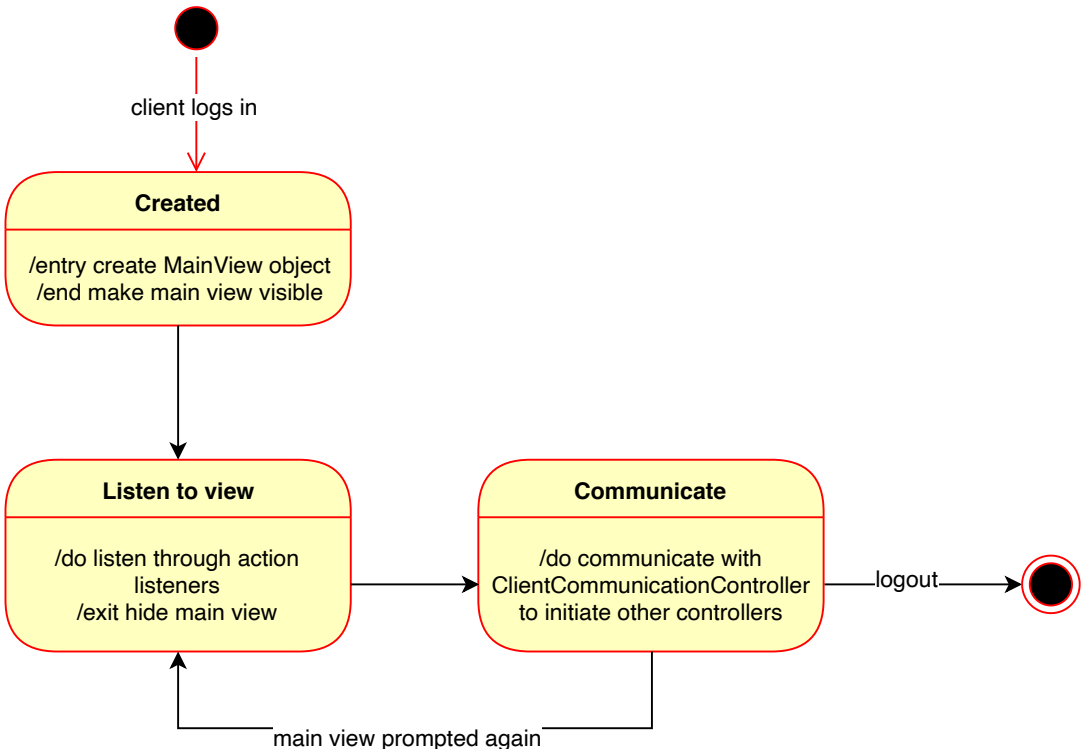



ClientCommunicationController
- socketOut: ObjectOutputStream
- aSocket: Socket
- socketIn: ObjectInputStream
- loginController: Controller
- emailController: Controller
- mainController: Controller
- searchListingController: Controller
- listingsController: Controller
- createListingController: Controller
+ main(args: String[]): void
+ showMainWindow(): void
+ showEmailView(): void
+ showSearchListingView(): void
+ showListingsView(): void
+ showCreateListingView(): void
Controller
clientCommunicationController: ClientCommunicationController
EmailController
- emailView: EmailView
+ sendEmailListen();
MainController
- mainView: MainView
+ searchListingListen(): void
+ editListingListen(): void
+ unsubscribeListen(): void
+ createListingListen(): void
+ getPeriodicalReportListen(): void
+ changeFeeListen(): void
+ viewEntityListen(): void
SearchListingController
- searchListingView: SearchListingView
+ searchListingListen();
LoginController
- loginView: LoginView
- verified: boolean
- user: User
+ loginListen(): void
+ isVerified(): boolean
ListingsController
- listingsView: ListingsView
+ selectListingListen(): void
PeriodicalReportView
- components: Components
+ display(): void
+ hide(): void
PeriodicalReportController
- periodicalReportView: PeriodicalReportView
+ createReportListen(): void
PeriodicalReport
- numOfHousesListed: int
- numOfHousesRented: int
- numOfActiveListings: int
- listings: ArrayList<Listing>
- startDate: Date
- endDate: Date
ListingsView
- components: Components
+ display(): void
+ hide(): void
EmailView
- components: Components
+ display(): void
+ hide(): void
SearchListingView
- components: Components
+ display(): void
+ hide(): void
CreateListingView
- components: Components
+ display(): void
+ hide(): void
LoginView
- components: Components
+ display(): void
+ hide(): void
CreateListingController
- createListingView: CreateListingView
+ createListingListen(): void
RenterMainView
- components: Components
+ display(): void
+ hide(): void
LandlordMainView
- components: Components
+ display(): void
+ hide(): void
ManagerMainView
- components: Components
+ display(): void
+ hide(): void
MainView
- components: Components
+ display(): void
+ hide(): void

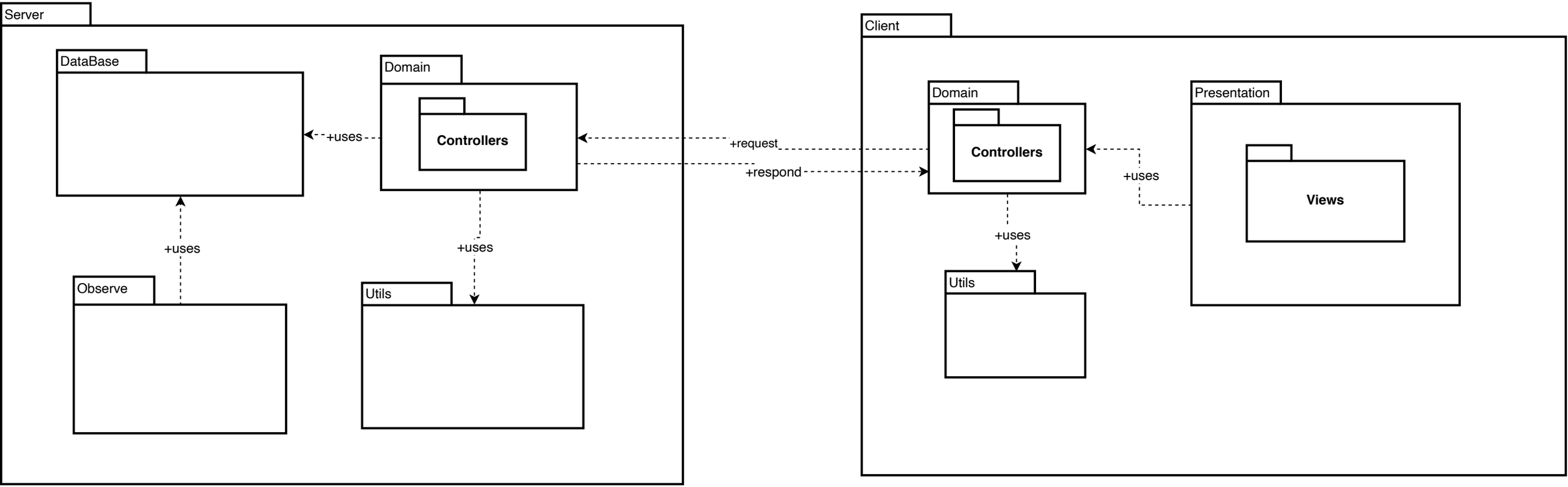
ServerCommunicationController State Diagram



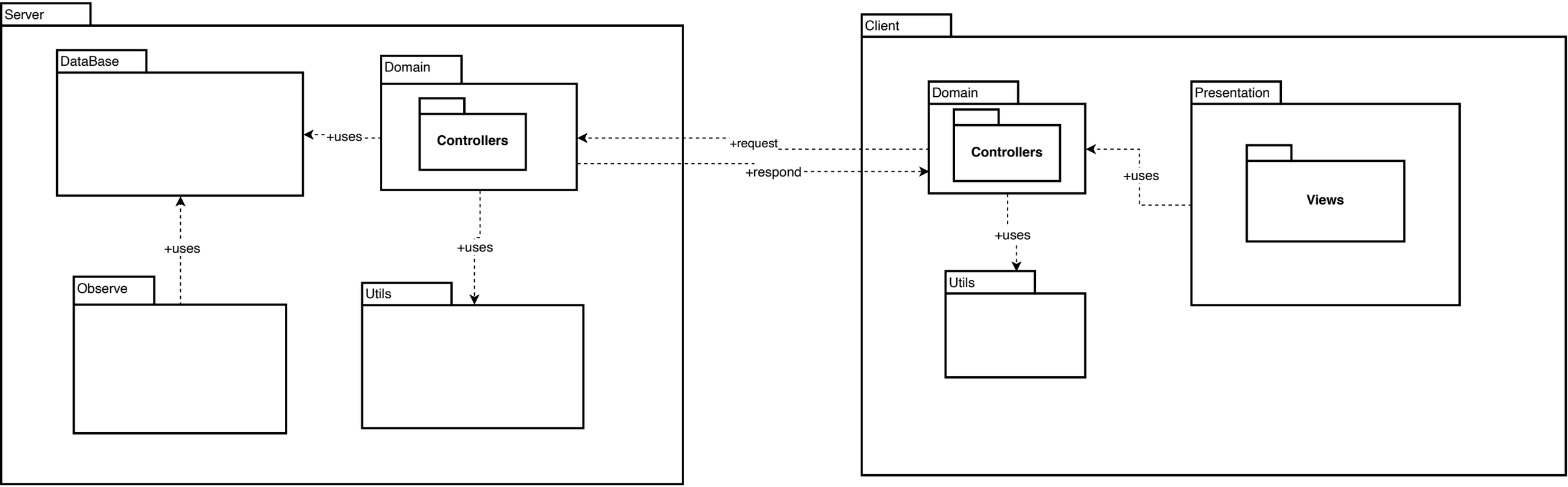
MainController State Diagram



Package Diagram



Package Diagram



Deployement Diagram