HomeQuest – SDS Report

By: Dominik Turowski, Hamza Madany, and Ziad El Gabry

Member Name	Contribution to Deliverable 1
Dominik *Turowski*	Requirements, Use Case Diagram, State Machines, Client Server Architecture Design
Hamza Madany	Requirements, Component Diagram, Singleton Design Pattern, Presentation Design
Ziad El Gabry	Requirements, Class Diagram, Sequence Diagram, Project "Overseer"

Functional Requirements

Property Search:

- The system shall allow users to search for properties available for purchase or rent.
- The system shall enable users to filter properties based on the following parameters:
    - **For Purchase**:
        - The system shall allow users to filter properties for purchase by location.
        - The system shall allow users to filter properties for purchase by price.
        - The system shall allow users to filter properties for purchase by size.
        - The system shall allow users to filter properties for purchase by the number of rooms.
        - The system shall allow users to filter properties for purchase by type of property.
    - **For Rent**:
        - The system shall allow users to filter properties for rent by location.
        - The system shall allow users to filter properties for rent by price range.
        - The system shall allow users to filter properties for rent by the duration of rent.
        - The system shall allow users to filter properties for rent by size.

    The system shall allow users to filter properties for rent by the number of rooms.

        - The system shall allow users to filter properties for rent by type of property.

- The system shall allow anyone to search for available properties.

Seller Profile:

- The system shall require property owners to register before listing properties.
- The system shall mandate sellers to provide the following information during registration:
    - The system shall require sellers to provide their full name.
    - The system shall require sellers to provide their email address.
    - The system shall require sellers to provide their date of birth.
    - The system shall require sellers to provide their consent to share location information.

- The system shall allow sellers to optionally enhance their profiles by providing:
    - The system shall allow sellers to include a profile photo.
    - The system shall allow sellers to include a phone number.
    - The system shall allow sellers to include a map-based property location.

Property Listing:

- The system shall allow sellers to register their properties with the following details:
    - The system shall require sellers to specify the property's location, which shall be displayed on a map.
    - The system shall require sellers to specify the property's price.
    - The system shall require sellers to specify the property's size.
    - The system shall require sellers to specify the number of rooms in the property.
    - The system shall require sellers to specify the type of property.
    - The system shall require sellers to upload photographs of the property.

Property Verification:

- The system shall allow sellers to optionally verify their properties by uploading official property documents.
- The system shall issue a verification badge to properties that pass verification.
- The system shall allow users to filter property search results by a "Verified Only" option.

Seller Types:

- The system shall categorize sellers into the following types:
    - **Free Users**:
        - The system shall allow free users to list up to two properties for sale or rent.
    - **Gold Users**:
        - The system shall allow gold users to list unlimited properties.
        - The system shall provide different subscription options for gold users.

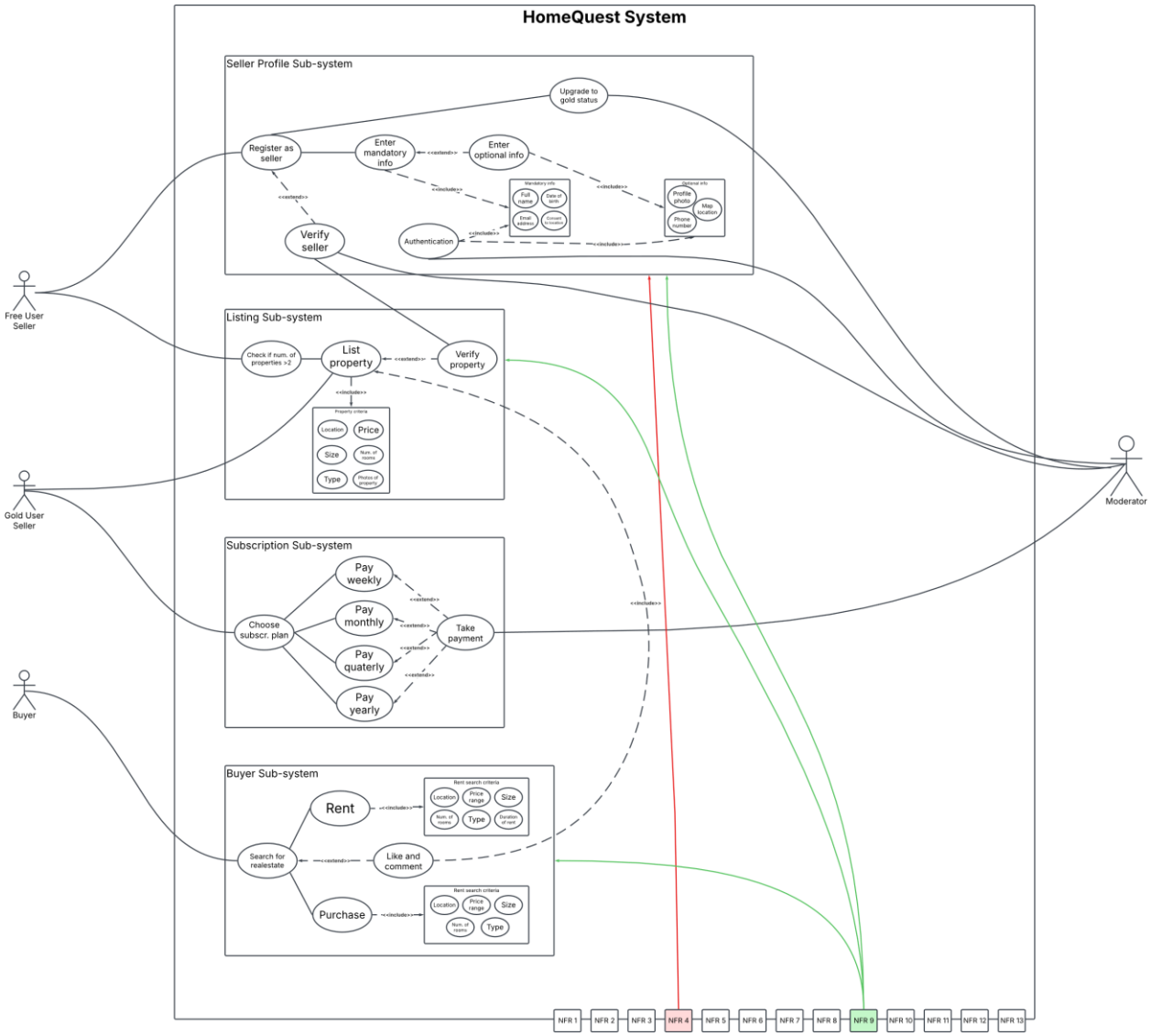
Interactive Features:

- The system shall allow property viewers to like and comment on property listings.

Non-Functional Requirements

1. The system should be able to handle 1,000 simultaneous users without noticeable deterioration in efficiency.
2. The platform should be able to adapt to an increase in features, properties and users.
3. The user data shall always be encrypted.
4. The system shall implement two factor authentication.
5. The interface shall be easy to use and comfortable.
6. The platform shall be optimized for laptops and mobile devices.
7. The system should retain 99.9% uptime over 12 months.
8. The system should recover from failure within 15 minutes.
9. The system shall store property photos and user data efficiently, ensuring quick retrieval within 5 seconds for any search query.
10. The system shall perform automated daily backups to prevent data loss and ensure data recovery in case of failure.
11. The system shall comply with accessibility standards to accommodate users with disabilities.
12. The platform shall support bilingual functionality, allowing seamless toggling between English and German.
13. The system shall comply with GDPR, ensuring users can provide consent for data usage and delete their data upon request.

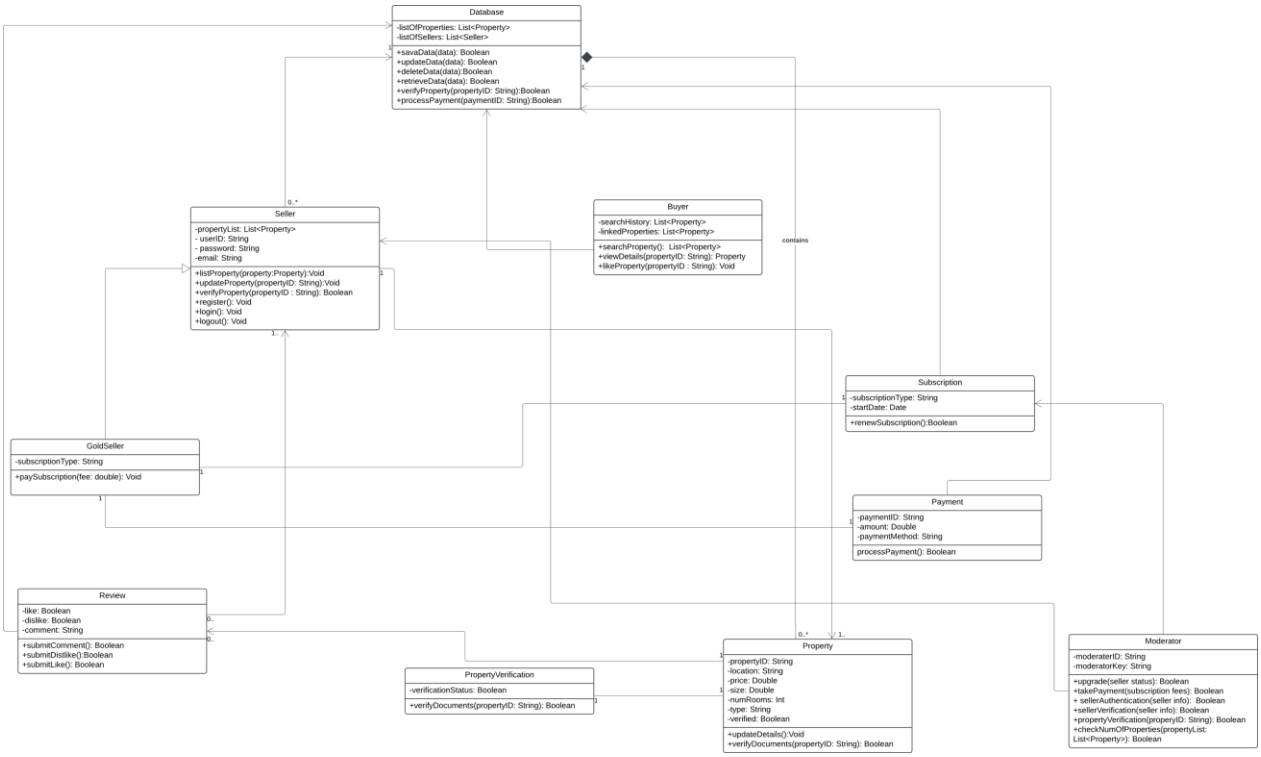
UML Use Case Diagram



Link to chart page:

https://lucid.app/lucidchart/7207ba37-3ff9-4260-9ba7a35e147620c1/edit?viewport_loc=7905%2C1823%2C11316%2C5260%2C0_0&invitationId=inv_d7dd7689-b185-4b95-968e-d56b508535ef

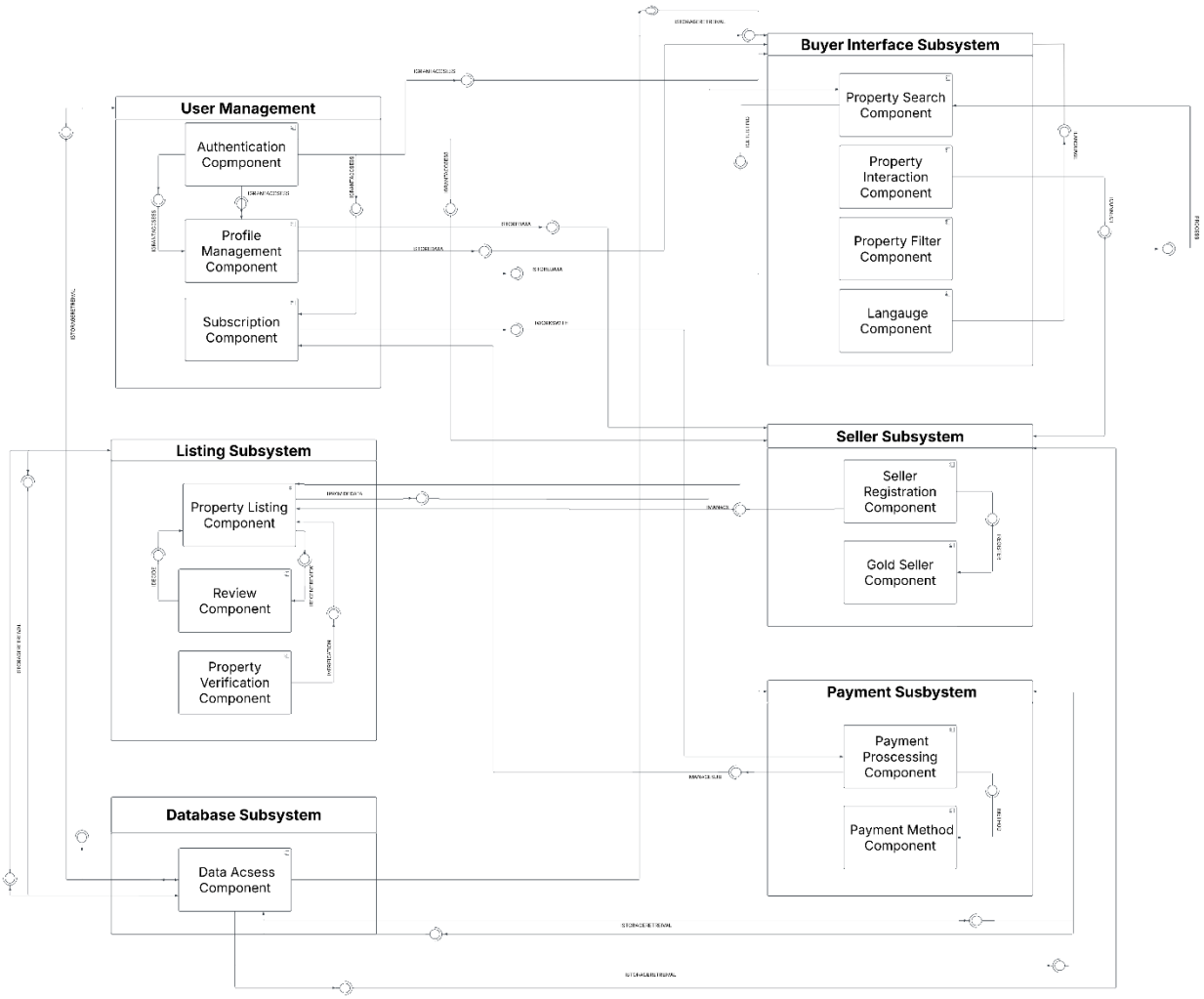
UML Class Diagram



Link to chart page:

https://lucid.app/lucidchart/56e78f18-a878-4985-a709ce680add83bf/edit?viewport_loc=-2637%2C-1608%2C4762%2C2236%2C0_0&invitationId=inv_be71b5b7-73d0-4a78-9e72-1eb123884710

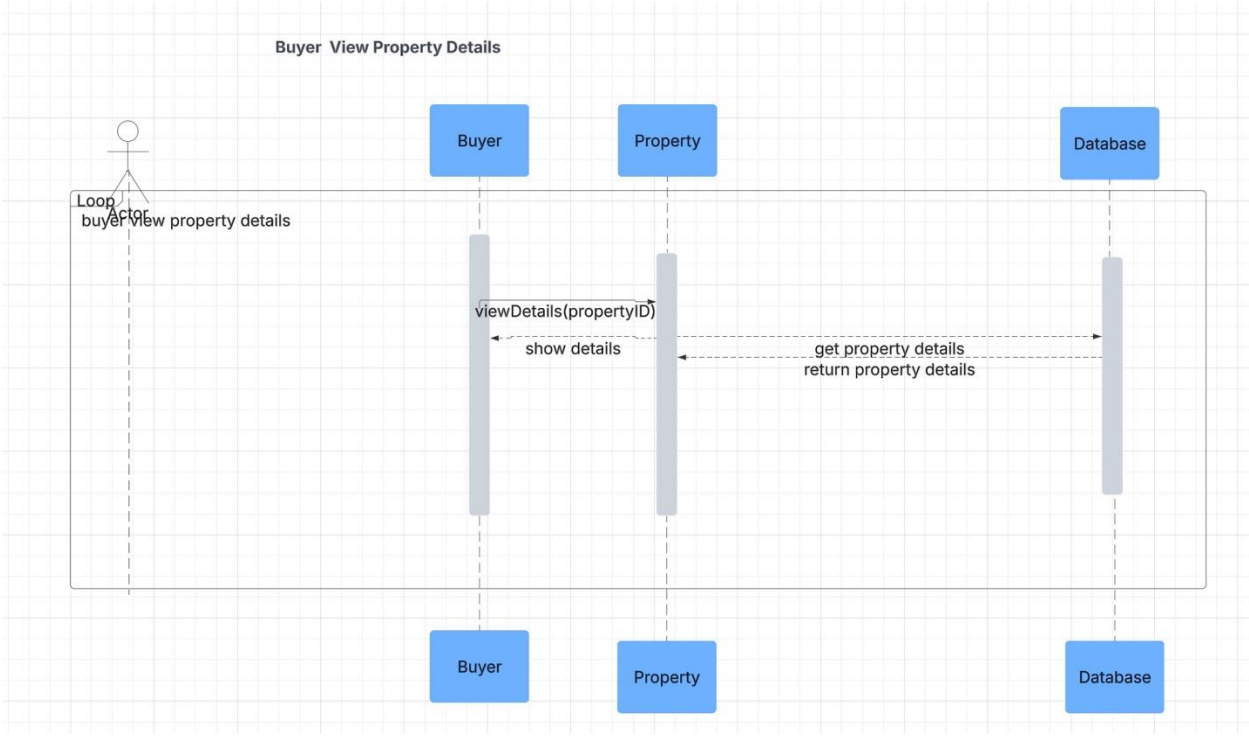
UML Component Diagram

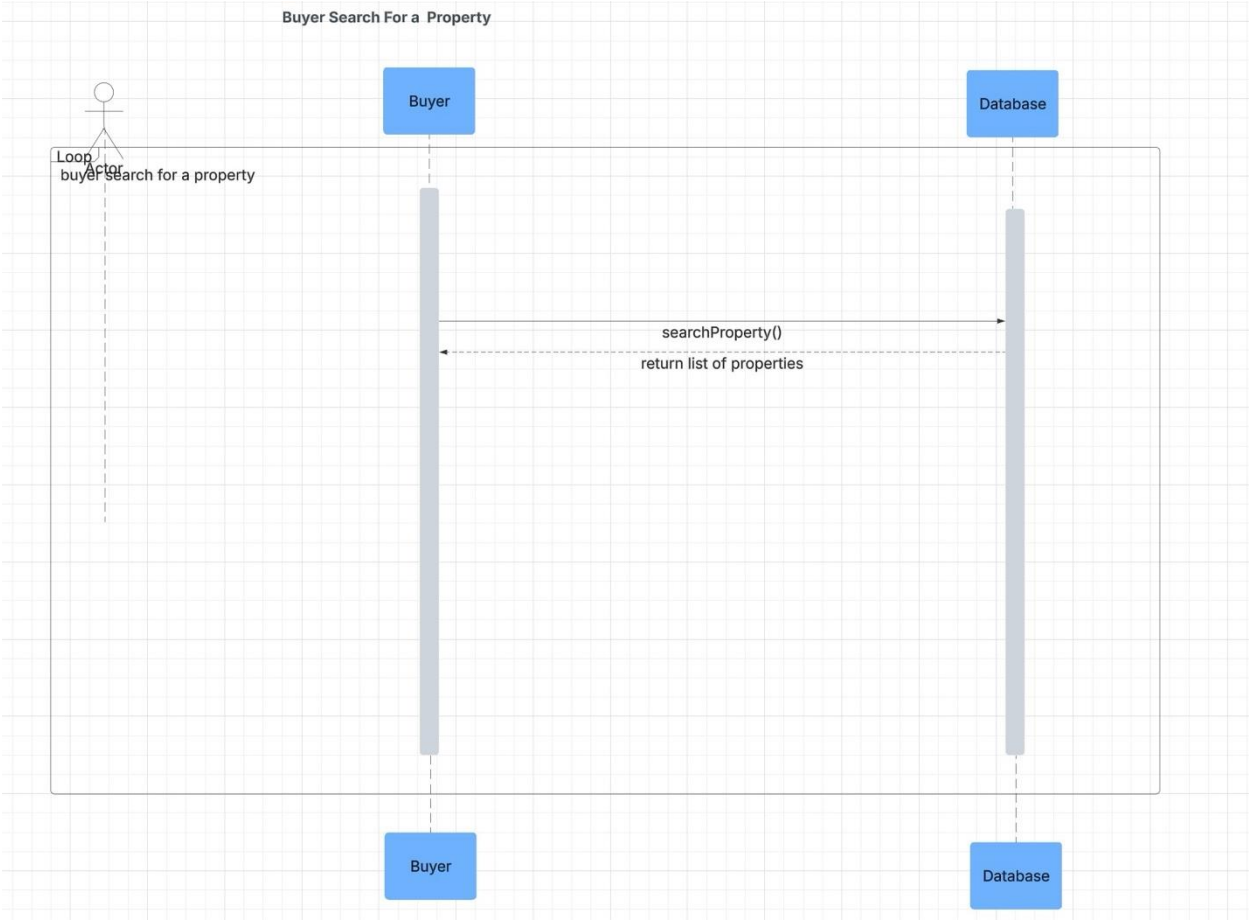


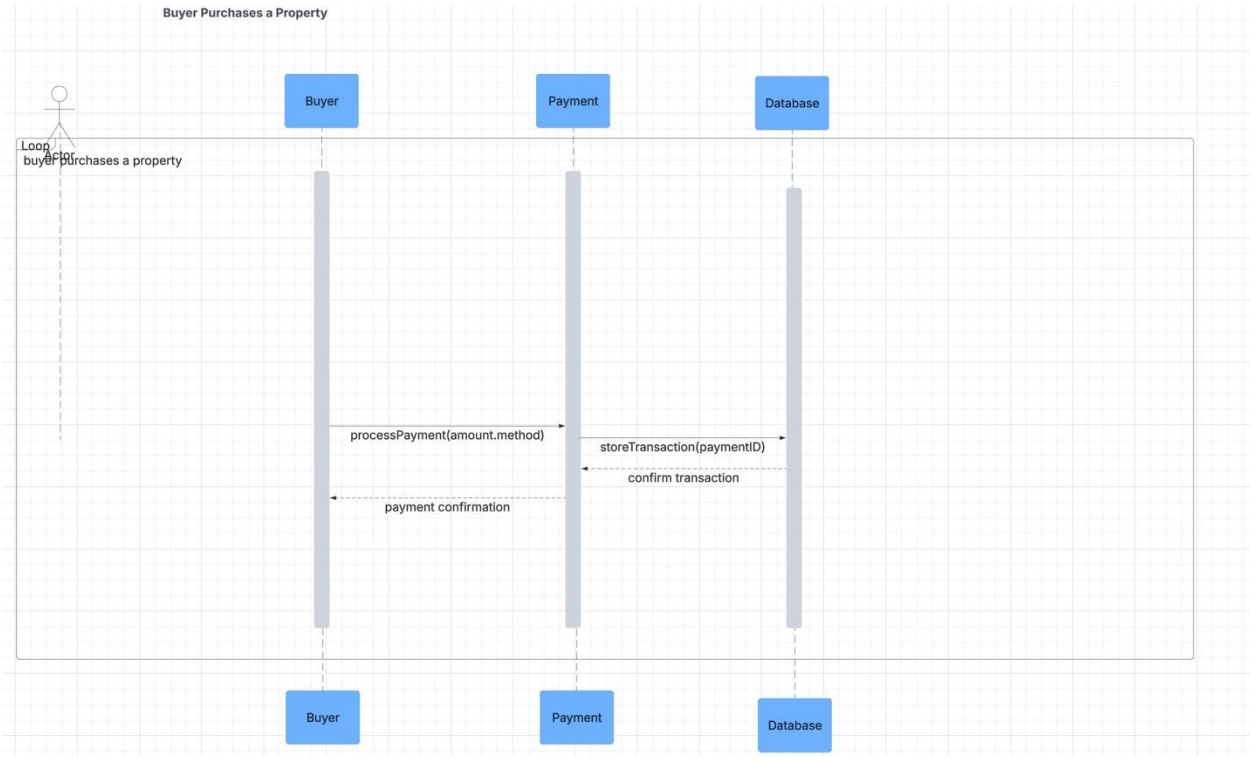
Link to chart page:

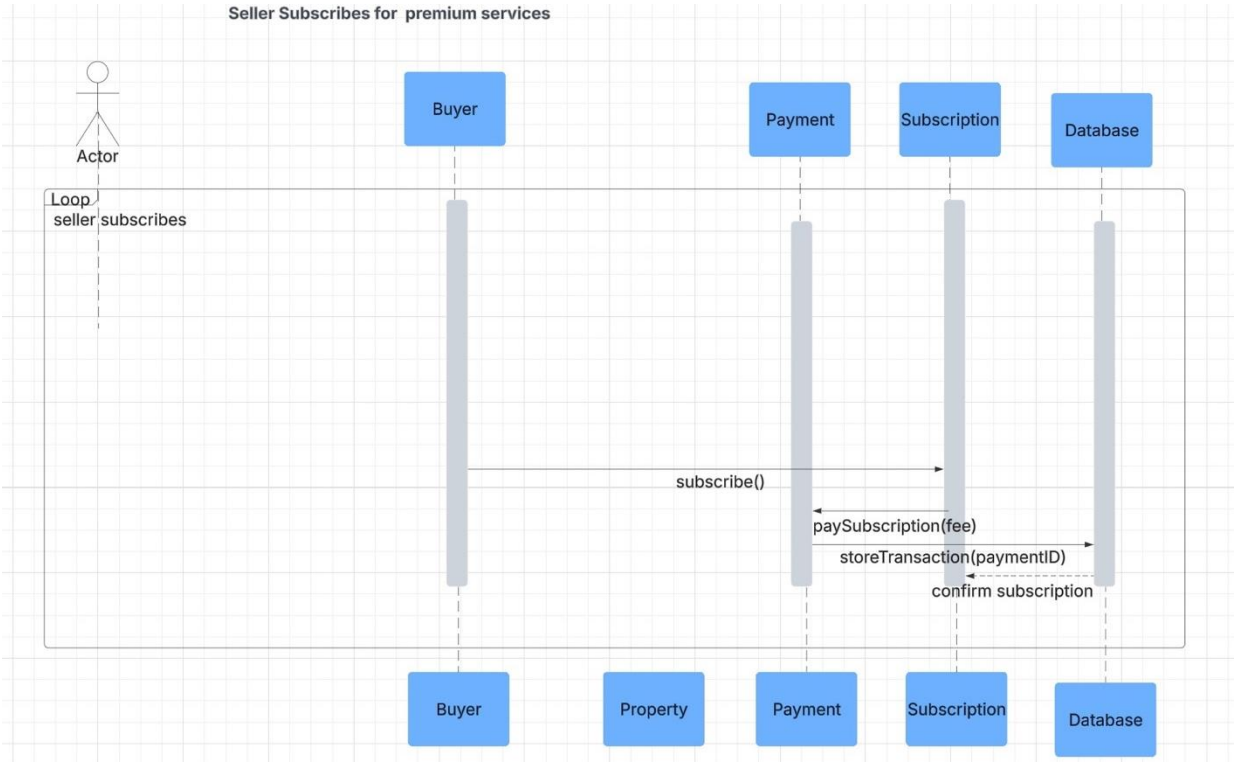
https://lucid.app/lucidchart/6c01eec5-a2cd-41b1-80f9-45c4b3c62aec/edit?viewport_loc=-994%2C-3065%2C8793%2C5069%2C0_0&invitationId=inv_408571ac-747e-4997-9487-07742d6a8d35

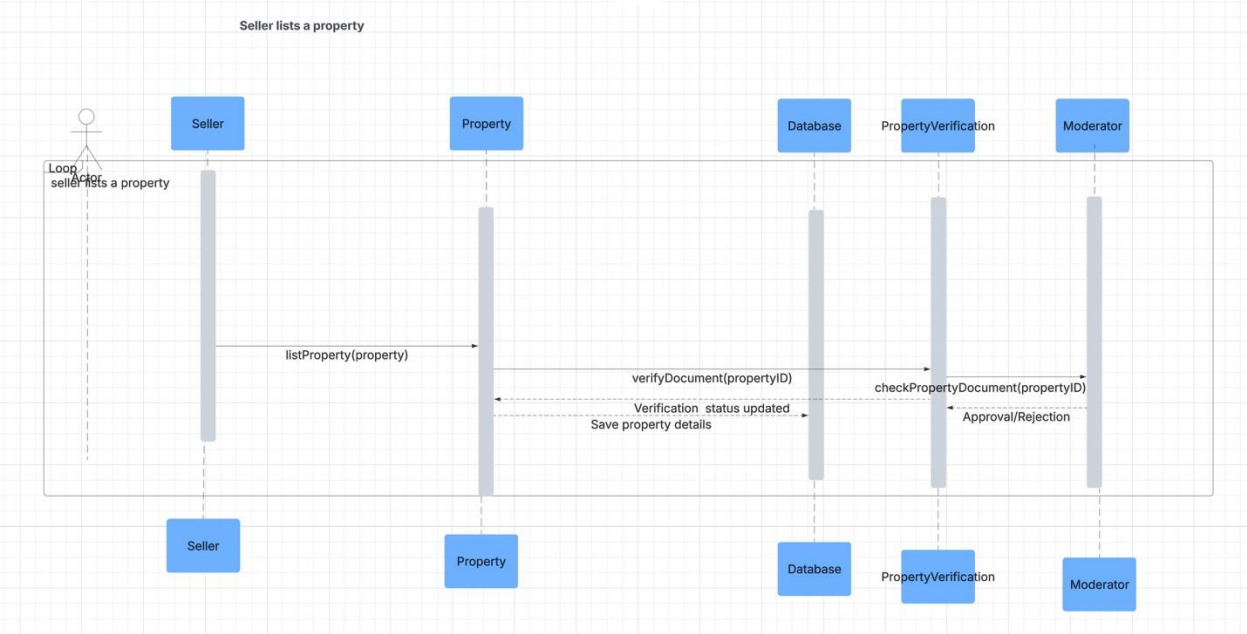
UML Sequence Diagrams





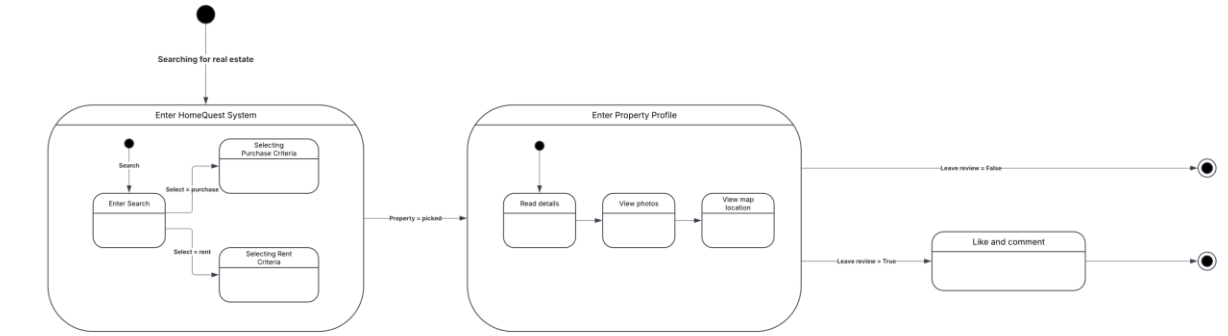

Buyer Purchases a Property
Buyer
Payment
Database
Actor
Loop
buyer purchases a property
processPayment(amount.method)
storeTransaction(paymentID)
confirm transaction
payment confirmation
Buyer
Payment
Database


Seller Subscribes for premium services
Actor
Buyer
Payment
Subscription
Database
Loop
seller subscribes
subscribe()
paySubscription(fee)
storeTransaction(paymentID)
confirm subscription
Buyer
Property
Payment
Subscription
Database

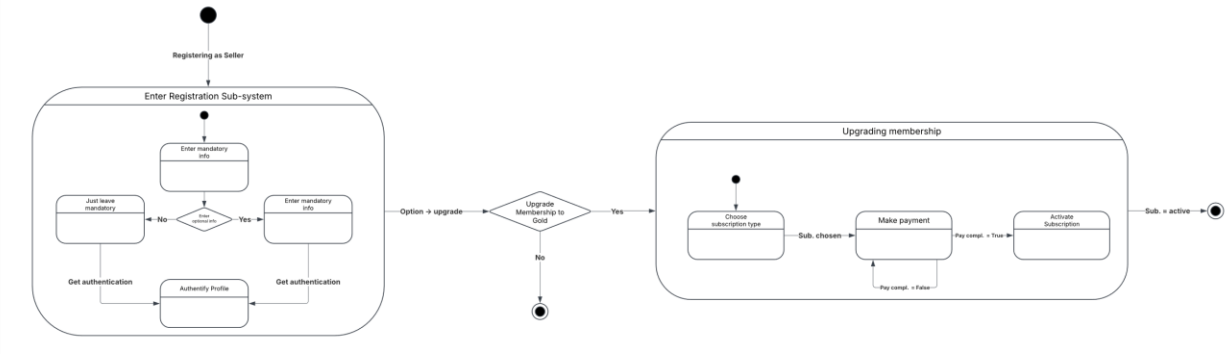

Seller lists a property
Seller
Property
Database
PropertyVerification
Moderator
Loop
seller lists a property
Actor
listProperty(property)
verifyDocument(propertyID)
checkPropertyDocument(propertyID)
Verification status updated
Approval/Rejection
Save property details
Seller
Property
Database
PropertyVerification
Moderator

UML State Machines

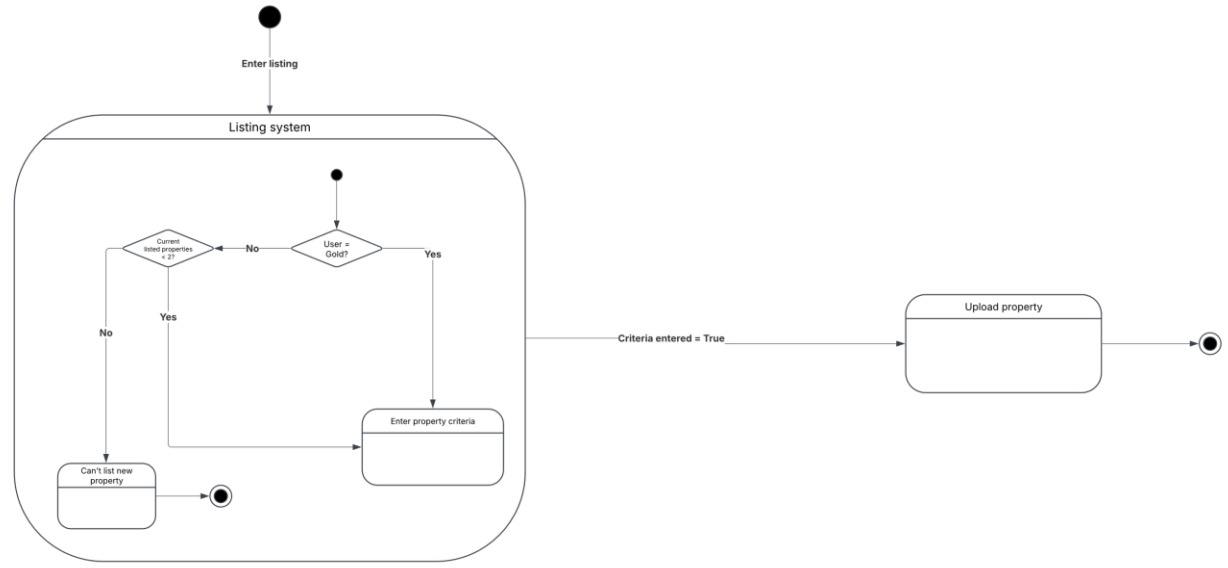
State Machine for Searching Sub-system



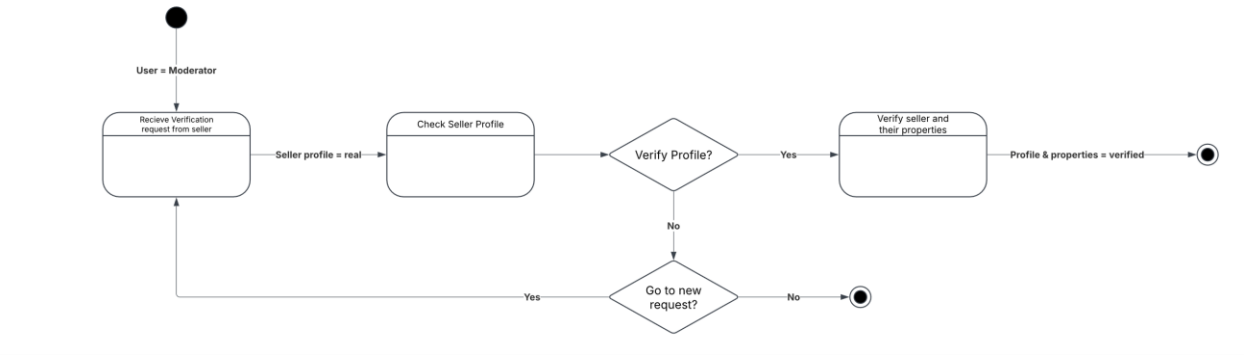
State Machine for Seller Registration and Subscription Sub-system



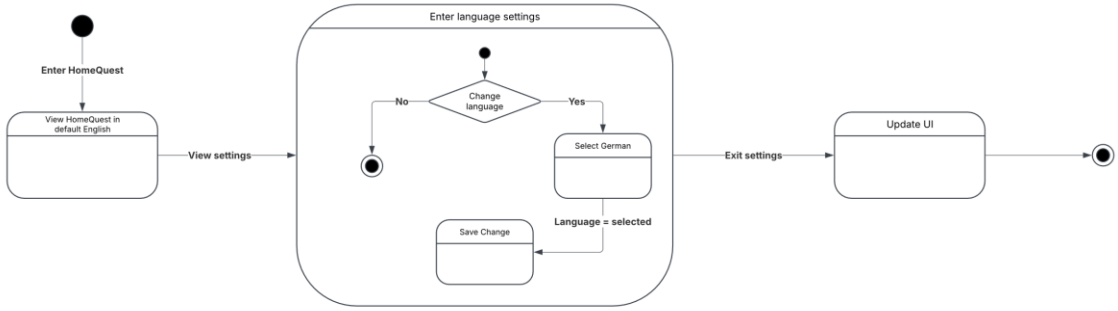
State Machine for Listing Sub-system



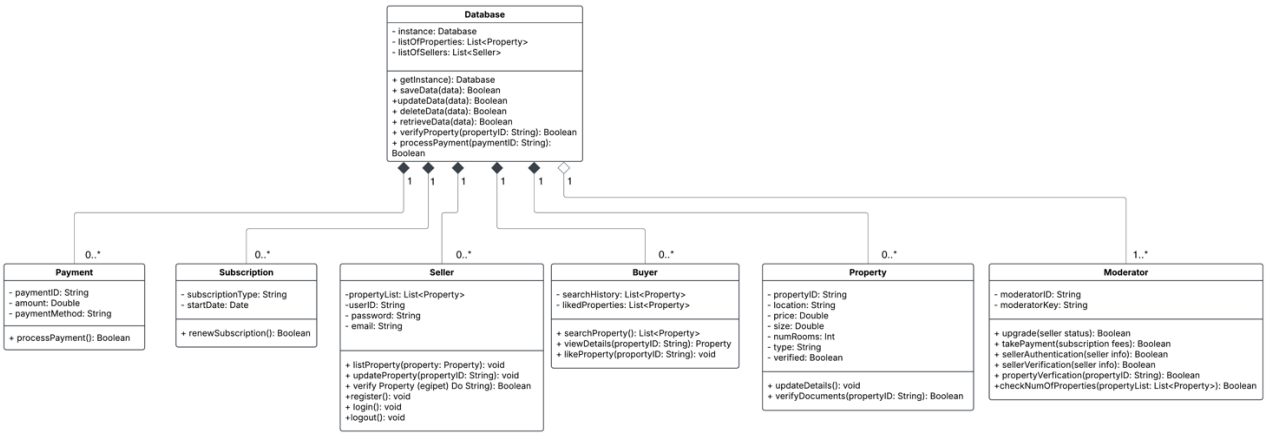
Verification Sub-systerm



Language Selection Sub-system

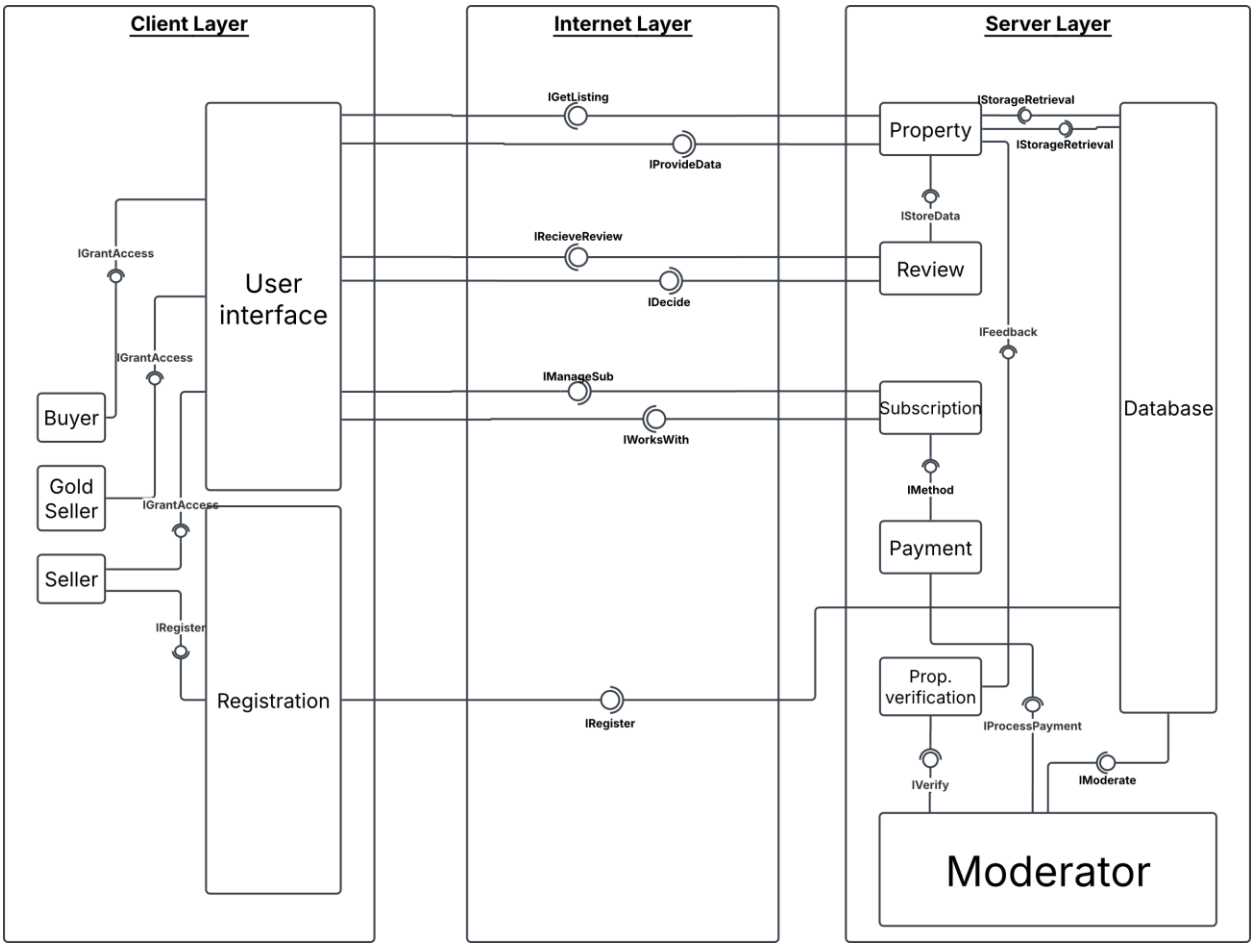


Singleton Design Pattern



The Singleton pattern in the Database class is what keeps everything running smoothly. Since there's only one instance managing all the properties, sellers, buyers, and transactions, it prevents data conflicts and ensures efficiency. This is especially important when handling 1,000 simultaneous users, as it avoids unnecessary duplication and keeps the system fast and responsive. It’s a central hub where all property listings, payments, and verifications are processed in one place. By structuring it this way, the platform stays scalable, secure, and easy to manage, even as more users and properties get added.

Client – Server Architecture Style (with layered features)



The Client – Server architectural style is suitable for this task since it allows the separation of the requesters and the providers. Both in the context of the overall service that HomeQuest gives (the users requesting service, and server providing it), as well as the interactions between the components, highlighted with ball-and-socket connections. In this format, it's easy to navigate and understand which elements of our system the users can interact with and which components can only interact among each other. In addition, aspects of the layered style have been applied to the design to highlight the mentioned interactions, where the internet layer serves solely as a connector between the front-end and back-end functions. This ensures security and ease of database management, as each component is designed to interact with its own layer, or the one directly beside it.