Deliverable 2:

Real Estate Database System

Christian Malan

Student ID#: 991720322

SYST17796: Fundamentals of Software Design and Development

CLASS: 1249_83050

Class Schedule: Tuesday 11am – 2pm

SYST 17796 DELIVERABLE 2
DESIGN DOCUMENT TEMPLATE

OVERVIEW

1. Project Background and Description

The trend in finding home and property is always moving and constant. It is crucial for both businesses owners and customers to decide when it comes to looking for the “right property”. The system of the project will provide a solution by providing a computer-based system platform for the business owners to showcase the properties while giving the customers and house owners the chance to book an appointment with their selected agent.

The goal of this project is to develop a computer-based system and create a database for commercial use in the real estate business in accordance with the project deliverables on the Fundamentals of Software Design and Development in the duration of the course. The system has a sign-in or sign-up selection that can make the user log in or register. Only the customers and house owners can register, but admins and agents can sign in. The system will give the business owners, also called admins an access to manage property listings in the interface, add property, edit property, add pending properties added by the house owners, view all users, view all properties, and view all appointments. On the customer side, users will be guided through questions to help them find their ideal home, or they can browse all available property listings and lastly book an appointment with a selected agent. The system will also allow the customers to search for their ideal homes, if there are no exact matches, the system will show recommendations according to the selection of the criteria they preferred. On the house owners, they have the dashboard where they can select to view all the properties and book an appointment with their selected agents. Lastly, for the agent side, users can view all the properties, view all the appointments, and approve appointments that were booked for them. This system will generate revenue by attracting house owners and property sellers to utilize the services provided by the real estate company, facilitating property listings, appointments, and other related transactions within a streamlined digital platform.

The RealEstateApp project, developed in Java, is inspired by a database system example from the course textbook and is structured using object-oriented programming principles. This application models an inventory management system tailored for real estate listings, with each property represented by a property class that includes essential details such as name, address, number of floors, rooms, property type, proximity amenities and schools.

The project follows the MVC (Model-View-Controller) architectural pattern to maintain a clean separation between the data, user interface, and control logic. In this setup, the Model handles the core data and algorithms within the database, utilizing classes for managing user roles—admin, customer, owner, and agent—each with specific permissions and actions accessible from their dashboard accounts. The View presents a user-friendly interface, allowing users to sign in or register, and provides intuitive access to each role’s functions. Agents interact with the system through an AgentPage, where they can view and manage appointments with owners and customers. The Controller is responsible for processing all user requests, updating the Model based on inputs such as property submissions or appointment requests, and then presenting the results in the View to deliver a cohesive, interactive experience for users.

Additionally, the project incorporates a booking system where customers and owners can schedule appointments with agents. Agents can approve or decline these bookings, with approved appointments displayed on the user’s interface. A UserRole enum categorizes users, ensuring that interactions are appropriately handled according to each role. Overall, RealEstateApp simulates a comprehensive and realistic real estate management experience, combining inventory and booking features within a structured, MVC-compliant framework.

2. Design Considerations

The code used in the project system relates to the code used in the textbook used in the class and uses object-oriented principles. In the textbook entitled “Headfirst Object-Oriented Analysis and Design”, it was stated that one of the basic principles of object-oriented programming is the encapsulation, and it helps the flow of program because it makes the variables in the class private and make the system to be delegated into parts using logics (McLaughlin et al, 2006, Chapter 1).

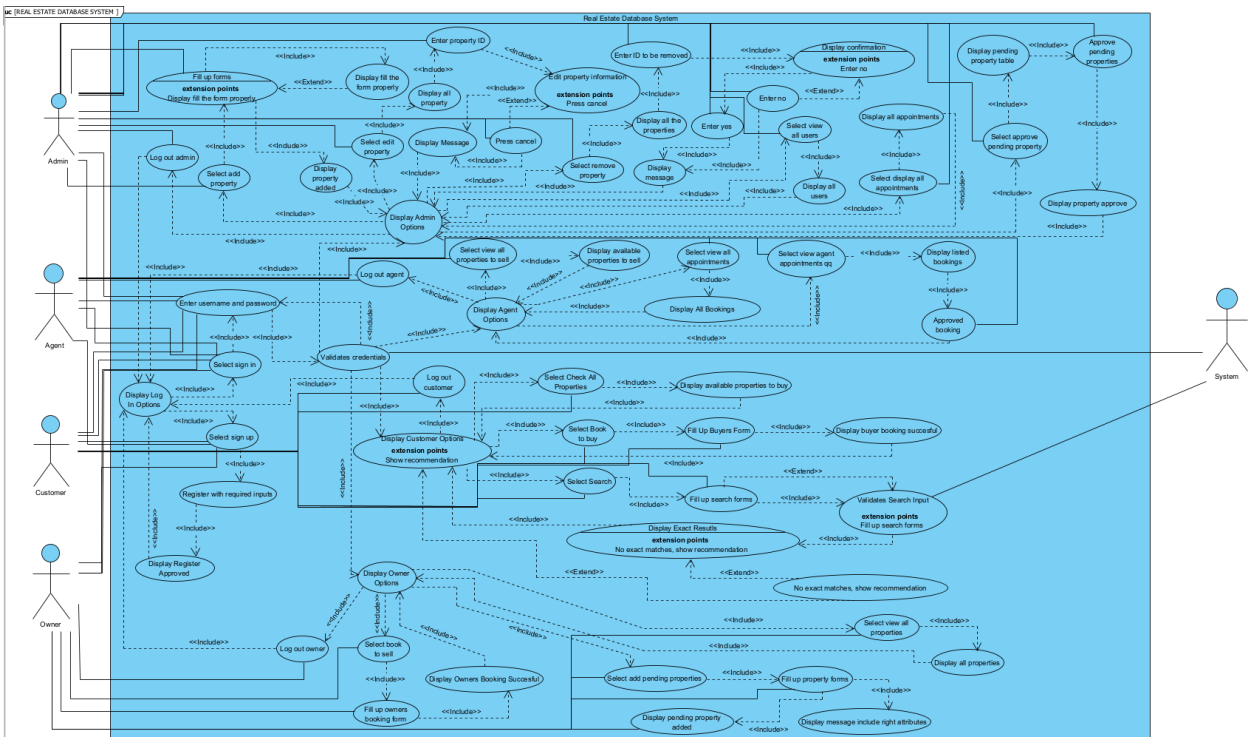

Figure 1. Real Estate System UML Case Diagram

The use case diagram on figure 1 for the real estate application outlines interactions among five main actors: Admin, Agent, Customer, Owner, and the System. Admin manages properties and users by adding, editing, removing, or approving properties and viewing user details. Agent is responsible for approving or declining bookings and viewing their scheduled appointments. Customer searches for properties, views property details, and schedules appointments with agents for property viewings. Owner submits properties for approval, monitors the status of these properties, and can book appointments with agents if necessary. System verifies user credentials during login, and if a user is not registered, it allows them to register with a role such as Customer and Owner. Once registered, System enforces role-based access to ensure each user can only interact with features relevant to their role, such as AdminPage, AgentPage, CustomerPage, or OwnerPage. This diagram visually captures each actor’s goals and interactions, supporting a clear understanding of user flows and access within the application.

The use case diagram and user interactions were described as such in the use case narratives with main and alternate path below.

1. Prompt User to Select Sign in or sign up

1.1 Sign In (Main Path)

1.1.1. Prompt User to enter username and password.

1.1.2. Enter username and password.

1.1.3. Verify credentials.

Successful Login (Main Path):

1.1.3.1. Grant access to the account.

1.1.3.2. Display user dashboard with options based on role (Admin, Agent, Customer, and Owner).

Alternate Path (Incorrect Credentials):

1.1.3.3. System rejects login attempt.

1.1.3.4. Re-prompt User to enter valid credentials until they are correct.

1.2 Sign Up (Alternate Path)

1.2.1. Select sign up option.

1.2.2. System prompts User to enter registration details (e.g., username, password, email).

1.2.3. User enters registration details and submits.

1.2.4. The system confirms registration with a success message.

1.2.5. Proceed to sign in by following steps in the Main Path (1.1).

2. User Dashboard

Once logged in, the User navigates to their specific dashboard based on role:

- 2.1 Admin Dashboard
- 2.2 Agent Dashboard
- 2.3 Customer Dashboard
- 2.4 Owner Dashboard

2.1 Admin Dashboard

2.1.1. Display Admin dashboard with management options: property management, user management, and appointment approvals.

2.1.1.1. Add a New Property

2.1.1.1.1. Select the option to add a new property.

2.1.1.1.2. Display property submission form.

2.1.1.1.3. Enter property details and submit.

2.1.1.1.4. The system confirms that the property has been successfully added.

Alternate Path (Incomplete Form):

2.1.1.1.5. System displays "Please complete all fields."

2.1.1.1.6. Return to submission form to complete.

2.1.1.2. Edit Property

2.1.1.2.1. Select options to edit an existing property.

2.1.1.2.2. Display list of properties.

2.1.1.2.3. Select property to edit.

2.1.1.2.4. Display property details form for editing.

2.1.1.2.5. Enter changes and submit.

2.1.1.2.6. System confirms "Edit successful."

Alternate Path (Cancel Edit):

2.1.1.2.7. Select cancel to abandon changes.

2.1.1.2.8. Return to Admin dashboard.

2.1.1.3. Delete Property

2.1.1.3.1. Select an option to delete a property.

2.1.1.3.2. Display list of properties.

2.1.1.3.3. Select property and confirm deletion.

2.1.1.3.4. The system removes the property.

Alternate Path (Cancel Deletion):

2.1.1.3.5. Select cancel.

2.1.1.3.6. Return to Admin dashboard.

2.1.1.4. View All Properties

2.1.1.4.1. Select options to view all properties.

2.1.1.4.2. The system displays a list of all properties.

2.1.1.4.3. Return to Admin dashboard.

2.1.1.5. Approve or Reject Pending Properties

2.1.1.5.1. View list of pending properties.

2.1.1.5.2. Select property to review.

Main Path (Approve Property):

2.1.1.5.3. Approve property.

2.1.1.5.4. Update status to "Approved" and notify Owner.

Alternate Path (Reject Property):

2.1.1.5.5. Reject property.

2.1.1.5.6. Update status to "Rejected" and notify Owner.

2.1.1.6. Manage Users and Appointments

2.1.1.6.1. View all user accounts.

2.1.1.6.2. Select user to manage permissions or view details.

2.1.1.6.3. View appointments and manage scheduling.

2.2 Agent Dashboard

2.2.1. Display Agent dashboard with options to view appointments, manage bookings, and access assigned properties.

2.2.1.1. View Appointments

2.2.1.1.1. Select options to view appointments.

2.2.1.1.2. The system displays a list of upcoming appointments.

2.2.1.1.3. Return to Agent dashboard.

2.2.1.2. Approve or Reject Bookings

2.2.1.2.1. View pending bookings.

2.2.1.2.2. The agent selects bookings to review.

Main Path (Approve Booking):

2.2.1.2.3. Approved booking.

2.2.1.2.4. Update status to "Approved" and notify Customer.

Alternate Path (Reject Booking):

2.2.1.2.5. Reject booking.

2.2.1.2.6. Update status to "Rejected" and notify Customer.

2.2.1.3. View Assigned Properties

2.2.1.3.1. Select option to view properties assigned to the Agent.

2.2.1.3.2. System displays assigned properties.

2.2.1.3.3. Return to Agent dashboard.

2.3 Customer Dashboard

2.3.1. Display Customer dashboard with options to search for properties, view property details, and book viewings.

2.3.1.1. Search for Properties

2.3.1.1.1. Enter search criteria.

2.3.1.1.2. The system displays properties matching criteria.

2.3.1.1.3. Return to Customer dashboard.

Alternate Path (No Properties Found):

2.3.1.1.4. The system displays "No properties found."

2.3.1.1.5. Return to search form to modify criteria.

2.3.1.2. View Property Details

2.3.1.2.1. Select property from search results.

2.3.1.2.2. The system displays property details.

2.3.1.2.3. Return to Customer dashboard.

2.3.1.3. Book a Property Viewing

2.3.1.3.1. Select property to book a viewing.

2.3.1.3.2. Fill out the booking form with date and time.

2.3.1.3.3. System confirms booking and notifies Customer.

Alternate Path (Booking Error):

2.3.1.3.4. The system displays an error message "Unable to complete booking."

2.3.1.3.5. Return to booking form to retry.

2.4 Owner Dashboard

2.4.1. Display Owner dashboard with options to submit properties, view submitted properties, and book appointments with Agents.

2.4.1.1. Submitting a New Property

2.4.1.1.1. Select an option to submit a property.

2.4.1.1.2. Fill out property submission form.

2.4.1.1.3. System saves property as "Pending Approval."

Alternate Path (Incomplete Form):

2.4.1.1.4. System displays "Please complete all fields."

2.4.1.1.5. Return to submission form to complete.

2.4.1.2. View Submitted Properties

2.4.1.2.1. View list of submitted properties with status.

2.4.1.2.2. Select property to view details.

2.4.1.2.3. Return to Owner dashboard.

2.4.1.3. Book Appointment with Agent

2.4.1.3.1. Select the option to book an appointment with an Agent.

2.4.1.3.2. Fill out appointment details and submit.

2.4.1.3.3. System confirms booking and notifies Owner.

Alternate Path (Booking Error):

2.4.1.3.4. The system displays an error message "Unable to complete booking."

2.4.1.3.5. Return to the appointment form to retry.

The UML class diagram for the real estate application provides a structured view of the system's classes, their attributes, methods, and relationships. It captures the essential components involved in managing properties, bookings, and user roles, including Admin, Agent, Customer, and Owner. Key classes such as Property, Booking, User, and DataStore are depicted, along with specialized classes like AddProperty, ApproveBooking, and Search, which represent actions specific to different user roles. The diagram illustrates associations, dependencies, inheritance, and aggregations between classes, showing how data flows within the application and how each class interacts to fulfill tasks like property management, appointment scheduling, and user verification. Overall, this UML class diagram provides a comprehensive blueprint for understanding the system's architecture, the roles of individual components, and the interconnections that enable core functionalities within the application.

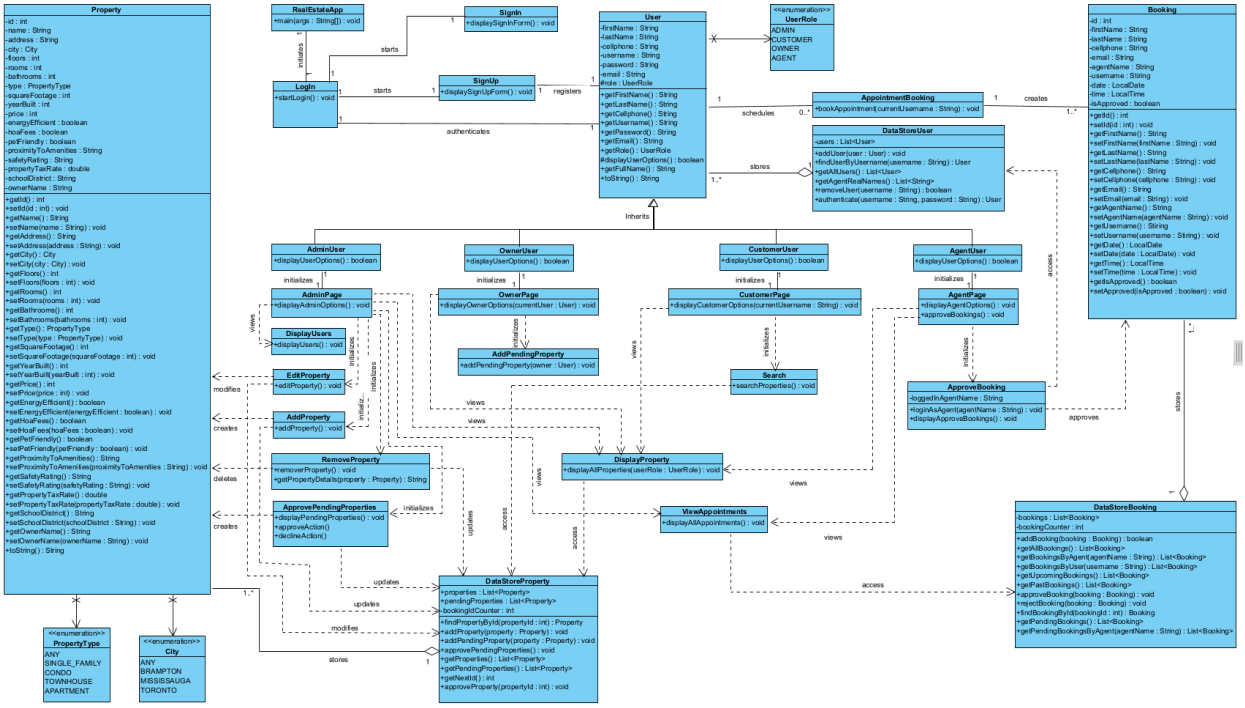

Figure 2. Real Estate System UML Class Diagram

The class diagram in Figure 2 represents the design of the real estate application, showing the relationships, associations, and multiplicities between core classes such as User, Property, Booking, DataStore, and action classes like AddProperty and ApproveBooking. The diagram includes associations and multiplicities to indicate how many instances of one class relate to instances of another, such as a single User being able to have multiple Booking instances (1 to many) and DataStoreUser aggregating multiple User objects (1 to many). Methods, modifiers, and return types are defined within each class, supporting encapsulation and allowing other classes to interact with specific functionalities without direct access to the underlying data.

Key Design Concepts

Encapsulation

Encapsulation is used extensively in Figure 2 to ensure that each class only exposes relevant data and behaviors to other classes. For example, User has private attributes (such as firstName, lastName, and role) with public getter and setter methods, ensuring that other classes can access or modify these properties only through controlled methods. This design choice restricts direct access to class internals, protecting the data and maintaining the integrity of each object.

Delegation

Delegation is implemented through action classes like AddProperty, ApproveBooking, and Search. Rather than having complex logic within UserPage classes (such as AdminPage, AgentPage), these responsibilities are delegated to specialized action classes. For instance, ApproveBooking is responsible for handling the approval of booking requests, while AddProperty manages the addition of properties. This delegation improves modularity and allows specific tasks to be handled by dedicated classes, which simplifies maintenance and makes the codebase more organized.

Cohesion

The design emphasizes high cohesion by grouping related functions within specific classes. Each class has a clear purpose and responsibility, DataStoreProperty stores and manages Property instances, Booking handles booking details, and UserPage classes initialize options specific to user roles. High cohesion ensures that each class is focused on a single aspect of functionality, making the system easier to understand, test, and maintain.

Coupling

The diagram aims for low coupling by minimizing dependencies between classes. For example, UserPage classes (such as AdminPage and AgentPage) interact with DataStore and action classes through well-defined interfaces rather than having direct dependencies on the details of Property or Booking. Where coupling is necessary, it is managed through dependencies and aggregations rather than direct associations, improving flexibility and reducing the impact of changes in one class on other classes.

Inheritance

Inheritance is used to define user roles through a User superclass, which is extended by AdminUser, AgentUser, OwnerUser, and CustomerUser. This approach leverages polymorphism, allowing different user types to have distinct behaviors while sharing common properties and methods defined in User. The UserRole enumeration further supports role-based functionality, associating each User instance with a specific role.

Aggregation

Aggregation is depicted in classes like DataStoreUser and DataStoreProperty, which maintain collections of User and Property instances, respectively. Aggregation is indicated by the open diamond notation, showing that DataStore classes manage but do not own the User or Property objects. This relationship allows DataStore classes to retrieve and store data without creating strong ownership ties, making it easy to manage data without tightly coupling classes.

Composition

Composition is applied to establish a strong relationship between classes where needed. For example, AppointmentBooking might compose Booking objects, as AppointmentBooking is responsible for creating and managing bookings on behalf of User. Since Booking is integral to the appointment functionality, AppointmentBooking could have a composition relationship with Booking, meaning that if AppointmentBooking is destroyed, the associated Booking instances would also be removed.

Flexibility/Maintainability

The design emphasizes flexibility and maintainability by isolating responsibilities through cohesive, loosely coupled classes and using interfaces for action-based interactions. Delegation and encapsulation contribute to flexibility, allowing each class to evolve independently as long as the defined interfaces remain consistent. For example, if additional booking options or property attributes need to be added, these can be introduced without significantly impacting the rest of the system. Furthermore, the use of enumeration (UserRole) for roles makes the system adaptable to adding more roles or modifying role-based behaviors.

Additional Design Elements

Methods and Return Types: Each class includes public methods with specific return types that align with its responsibilities, such as findUserByUsername() in DataStoreUser (returning a User object) and getAllBookings() in DataStoreBooking (returning a list of Booking objects). These methods provide clear, purposeful interactions while maintaining encapsulation.

Enumeration for User Roles: The UserRole enumeration supports role-based functionality and allows each User to be assigned a specific role. This enumeration ensures type safety, reducing the chance of invalid role assignments and allowing for future role expansion without disrupting the existing system structure.

In summary, Figure 2 illustrates a well-structured class diagram for the real estate application, with thoughtful attention to encapsulation, delegation, cohesion, and low coupling. The use of inheritance and aggregation promotes efficient data management, while encapsulation and composition contribute to the integrity and maintainability of the design.

The Real Estate System UML class relationships and multiplicities are described as follows:

1. RealEstateApp ↔ Login
    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "starts"
    - Description: RealEstateApp initializes the Login process, allowing users to authenticate into the system.
2. Login ↔ SignIn
    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "starts"
    - Description: Login uses the SignIn process to authenticate users.
3. Login ↔ SignUp
    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "starts"
    - Description: Login uses the SignUp process to register new users.
4. SignIn ↔ User
    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "authenticates"
    - Description: The SignIn class authenticates a User objects.

5. SignUp ↔ User
    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "registers"
    - Description: The SignUp class registers new User instances.
6. User ↔ UserRole (Enum)
    - Relationship Type: Association
    - Label: "assigned role"
    - Description: Each User has a specific UserRole, which can be ADMIN, CUSTOMER, OWNER, or AGENT.
7. User ↔ AppointmentBooking
    - Relationship Type: Association
    - Multiplicity: * ↔ *
    - Label: "schedules"
    - Description: User can schedule multiple AppointmentBooking entries, and each AppointmentBooking can involve multiple users.
8. DataStoreUser ↔ User
    - Relationship Type: Aggregation
    - Multiplicity: 1 ↔ *
    - Label: "stores"
    - Description: DataStoreUser stores multiple instances of User without owning them, managing user data in the system.
9. AdminUser ↔ AdminPage
    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "initializes"
    - Description: AdminUser interacts with AdminPage to perform admin-related functionalities.
10. AgentUser ↔ AgentPage
    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "initializes"
    - Description: AgentUser interacts with AgentPage to perform agent-related functionalities.
11. CustomerUser ↔ CustomerPage
    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "initializes"
    - Description: CustomerUser interacts with CustomerPage to perform customer-related functionalities.
12. OwnerUser ↔ OwnerPage
    - Relationship Type: Association

- Multiplicity: 1 ↔ 1
- Label: "initializes"
- Description: OwnerUser interacts with OwnerPage to manage owner-specific options like property submission.

13. OwnerUser ↔ AddPendingProperty

    - Relationship Type: Association
    - Multiplicity: 1 ↔ *
    - Label: "creates"
    - Description: OwnerUser uses AddPendingProperty to submit properties for approval.

14. CustomerUser ↔ Search

    - Relationship Type: Association
    - Multiplicity: 1 ↔ *
    - Label: "searches"
    - Description: CustomerUser interacts with Search to find properties within the system.

15. Property ↔ PropertyType (Enum)

    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "classified as"
    - Description: Each Property has a PropertyType defining its type (e.g., SINGLE_FAMILY, CONDO).

16. Property ↔ City (Enum)

    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "located in"
    - Description: Each Property is located in a specific City.

17. DataStoreProperty ↔ Property

    - Relationship Type: Aggregation
    - Multiplicity: 1 ↔ *
    - Label: "stores"
    - Description: DataStoreProperty stores multiple Property objects for management.

18. DisplayProperty ↔ DataStoreProperty

    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "views"
    - Description: DisplayProperty retrieves property data from DataStoreProperty for display purposes.

19. EditProperty ↔ DataStoreProperty

    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "modifies"

- Description: EditProperty allows modifications to properties stored within DataStoreProperty.

20. AddProperty ↔ DataStoreProperty
    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "adds"
    - Description: AddProperty enables adding new properties to DataStoreProperty.
21. RemoveProperty ↔ DataStoreProperty
    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "deletes"
    - Description: RemoveProperty allows properties to be deleted from DataStoreProperty.
22. ApprovePendingProperties ↔ DataStoreProperty
    - Relationship Type: Association
    - Multiplicity: 1 ↔ *
    - Label: "initializes"
    - Description: ApprovePendingProperties processes pending properties submitted for approval.
23. Booking ↔ DataStoreBooking
    - Relationship Type: Aggregation
    - Multiplicity: 1 ↔ *
    - Label: "manages"
    - Description: DataStoreBooking manages multiple instances of Booking.
24. User ↔ Booking
    - Relationship Type: Association
    - Multiplicity: * ↔ *
    - Label: "creates"
    - Description: Users can create multiple bookings, and each booking is linked to a user.
25. AgentUser ↔ ApproveBooking
    - Relationship Type: Association
    - Multiplicity: * ↔ *
    - Label: "approves"
    - Description: AgentUser can approve or reject booking requests through ApproveBooking.
26. DataStoreBooking ↔ ApproveBooking
    - Relationship Type: Association
    - Multiplicity: 1 ↔ *
    - Label: "access"
    - Description: DataStoreBooking provides access to booking data for approval or rejection by agents.
27. AppointmentBooking ↔ User
    - Relationship Type: Association

- Multiplicity: * ↔ *
- Label: "schedules"
- Description: Users can schedule appointments with agents through AppointmentBooking.

28. DataStoreUser ↔ AppointmentBooking

    - Relationship Type: Association
    - Multiplicity: 1 ↔ *
    - Label: "stores"
    - Description: DataStoreUser stores appointment data for various users.

29. Search ↔ DataStoreProperty

    - Relationship Type: Association
    - Multiplicity: 1 ↔ 1
    - Label: "accesses"
    - Description: Search accesses DataStoreProperty to find and filter properties based on user criteria.

Enums

1. UserRole

    - Used to specify roles like ADMIN, CUSTOMER, OWNER, and AGENT for User.
    - Association: User ↔ UserRole
    - Label: "assigned role"

2. PropertyType

    - Defines property types like SINGLE_FAMILY, CONDO, TOWNHOUSE, APARTMENT.
    - Association: Property ↔ PropertyType
    - Label: "classified as"

3. City

    - Lists cities like TORONTO, MISSISSAUGA, BRAMPTON.
    - Association: Property ↔ City
    - Multiplicity: 1 ↔ 1
    - Label: "located in"

REFERENCES

McLaughlin, B., Pollice, G., & West, D. (2006). *Head First Object-Oriented Analysis and Design.* O'Reilly Media, Inc. https://learning.oreilly.com/library/view/head-first-object-oriented/0596008678/?ar