Tishk International University -Sulaimani

Relevant Course - Entrepreneurship IT

Department of Computer Engineering

RealEstate Management System

Prepared by : Haima Hussien Jafr,Hazhan Hiwa

Course Lecturer:

Ms. Yusra mohemmed

Assigned Date:

December 8, 2024

Submission Due:

December 23, 2024

Introduction

The **Real Estate Management Database** is a comprehensive system designed to facilitate the management of real estate properties, clients, agents, transactions, and rentals. This project aims to streamline the operations of a real estate business, providing efficient data storage, retrieval, and analysis. The database is structured to handle various aspects of the real estate market, such as property listings, client preferences, agent information, and transaction records. By automating data management, the system ensures accuracy, transparency, and ease of access for all stakeholders. Real estate businesses, whether large or small, require effective solutions to manage their operations. This project caters to such needs by offering a well-organized database design that aligns with the requirements of real estate firms. From residential and commercial properties to land listings, the database covers a broad spectrum of real estate types, providing a versatile foundation for day-to-day operations and long-term strategy.

Problem Statement

The real estate industry involves managing large volumes of data, including property details, client requirements, agent commissions, and transaction records. Traditionally, this data is maintained manually or through fragmented systems, leading to challenges such as:

1. **Data Inconsistency**: Multiple systems or manual records often result in outdated or conflicting information, which can harm client relationships and operational efficiency.
2. **Inefficiency**: Manually tracking property statuses, client interactions, and financial transactions consumes significant time and effort, reducing overall productivity.
3. **Error Prone**: Manual data entry increases the likelihood of errors, which can have costly implications, such as incorrect financial calculations or missed opportunities for property sales or rentals.

4. **Lack of Insights**: Without an integrated system, deriving actionable insights from data becomes difficult, limiting the ability to make data-driven decisions.

Poor Communication: Fragmented systems can hinder communication between agents, clients, and administrators, leading to delays and misunderstandings.

These challenges highlight the need for a centralized, automated system that can handle the complexities of real estate management while minimizing errors and inefficiencies.

Solution

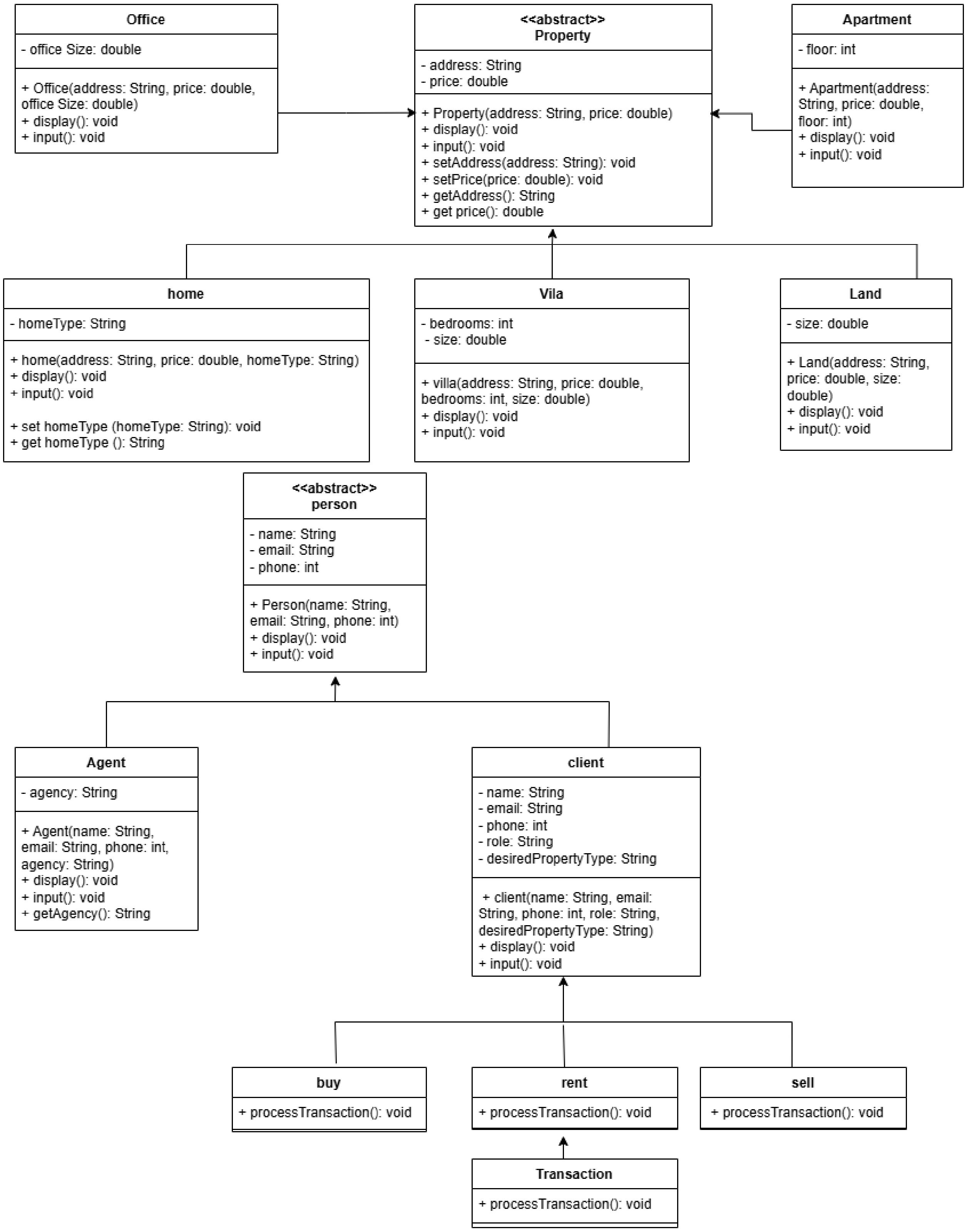
The Real Estate Management Database addresses these challenges by providing a centralized, relational database system. This database offers the following key features:

- **Centralized Property Management**: Comprehensive storage and categorization of property details, including type, price, size, and status. This eliminates data redundancy and ensures that information is always up to date.
- **Client Relationship Management**: The system tracks client preferences, budgets, and interactions, enabling tailored recommendations and services.
- **Agent Coordination**: Information about agents, their specializations, and commission rates is stored, ensuring that the right agents are assigned to the right tasks.
- **Transaction and Rental Records**: Maintaining transaction histories and rental agreements ensures transparency and traceability for all financial dealings.
- **Scalability**: The database structure is designed to accommodate the growing needs of a real estate business, supporting additional features and larger datasets over time.

Validation and Accuracy: Built-in constraints and validation rules ensure data integrity, reducing the risk of errors and inconsistencies.

Methodology:

Uml Diagram:



Office
- office Size: double
+ Office(address: String, price: double, office Size: double)
+ display(): void
+ input(): void
<<abstract>> Property
- address: String
- price: double
+ Property(address: String, price: double)
+ display(): void
+ input(): void
+ setAddress(address: String): void
+ setPrice(price: double): void
+ getAddress(): String
+ get price(): double
Apartment
- floor: int
+ Apartment(address: String, price: double, floor: int)
+ display(): void
+ input(): void
home
- homeType: String
+ home(address: String, price: double, homeType: String)
+ display(): void
+ input(): void
+ set homeType (homeType: String): void
+ get homeType (): String
Vila
- bedrooms: int
- size: double
+ villa(address: String, price: double, bedrooms: int, size: double)
+ display(): void
+ input(): void
Land
- size: double
+ Land(address: String, price: double, size: double)
+ display(): void
+ input(): void
<<abstract>> person
- name: String
- email: String
- phone: int
+ Person(name: String, email: String, phone: int)
+ display(): void
+ input(): void
Agent
- agency: String
+ Agent(name: String, email: String, phone: int, agency: String)
+ display(): void
+ input(): void
+ getAgency(): String
client
- name: String
- email: String
- phone: int
- role: String
- desiredPropertyType: String
+ client(name: String, email: String, phone: int, role: String, desiredPropertyType: String)
+ display(): void
+ input(): void
buy
+ processTransaction(): void
rent
+ processTransaction(): void
sell
+ processTransaction(): void
Transaction
+ processTransaction(): void

Database Design

The database includes the following tables:

- **Clients**: Stores information about clients (e.g., name, email, phone, type).
- **Agents**: Stores agent details (e.g., name, email, agency).
- **Properties**: Tracks property data (e.g., property type, location, price).
- **Transactions**: Logs interactions between clients, agents, and properties.

Java Classes

Key classes include:

1. **Person Class**: Abstract base class with attributes like name, email, and phone.
2. **Client Class**: Derived from Person, includes client-specific details (e.g., type: Buyer, Seller, Renter).
3. **Agent Class**: Derived from Person, includes agency information.
4. **Property Class**: Abstract class extended by:
    - **Villa**: Attributes like bedrooms and size.
    - **Land**: Attributes like land size.
    - **Apartment**: Attributes like floor number.
    - **Office**: Attributes like office size.
    - **Home**: Attributes like home type.

Implementation

Java Code

The system's backend was implemented using Java. Key components include:

- **RealEstateManagementSystem Class**: Handles the main functionality of the system.

- **Property Management**: Allows adding, updating, viewing, and deleting properties.
- **Client/Agent Management**: Facilitates the registration and management of clients and agents.

Coding Language Conversion To Python

class Person:

```
    def __init__(self, name, email, phone):
        self.name = name
        self.email = email
        self.phone = phone

    def display(self):
        print(f"Name: {self.name}")
        print(f"Email: {self.email}")
        print(f"Phone: {self.phone}")
```

class Client(Person):

```
    def __init__(self, name, email, phone, client_type, preference):
        super().__init__(name, email, phone)
        self.client_type = client_type
        self.preference = preference

    def display(self):
        super().display()
        print(f"Client Type: {self.client_type}")
        print(f"Preference: {self.preference}")
```

```
class Agent(Person):
    def __init__(self, name, email, phone, agency):
        super().__init__(name, email, phone)
        self.agency = agency

    def display(self):
        super().display()
        print(f"Agency: {self.agency}")
```

```
class Property:
    def __init__(self, property_id, location, price):
        self.property_id = property_id
        self.location = location
        self.price = price

    def display_property_details(self):
        raise NotImplementedError("Subclasses must implement this method")
```

```
class Villa(Property):
    def __init__(self, property_id, location, price, bedrooms, size):
        super().__init__(property_id, location, price)
        self.bedrooms = bedrooms
        self.size = size
    def display_property_details(self):
        print(f"Villa - ID: {self.property_id}, Location: {self.location}, Price: {self.price}")
        print(f"Bedrooms: {self.bedrooms}, Size: {self.size} sqft")


class Apartment(Property):
    def __init__(self, property_id, location, price, floor):
        super().__init__(property_id, location, price)
        self.floor = floor

    def display_property_details(self):
        print(f"Apartment - ID: {self.property_id}, Location: {self.location}, Price: {self.price}")
        print(f"Floor: {self.floor}")


class Land(Property):
    def __init__(self, property_id, location, price, size):
        super().__init__(property_id, location, price)
        self.size = size

    def display_property_details(self):
        print(f"Land - ID: {self.property_id}, Location: {self.location}, Price: {self.price}")
        print(f"Size: {self.size} acres")
```

```
class Office(Property):
    def __init__(self, property_id, location, price, size):
        super().__init__(property_id, location, price)
        self.size = size

    def display_property_details(self):
        print(f"Office - ID: {self.property_id}, Location: {self.location}, Price: {self.price}")
        print(f"Size: {self.size} sqft")


class Home(Property):
    def __init__(self, property_id, location, price, home_type):
        super().__init__(property_id, location, price)
        self.home_type = home_type

    def display_property_details(self):
        print(f"Home - ID: {self.property_id}, Location: {self.location}, Price: {self.price}")
        print(f"Type: {self.home_type}")


# Main System
class RealEstateManagementSystem:
    def __init__(self):
        self.people = []
        self.properties = []

    def register_client(self, name, email, phone, client_type, preference):
```

```
    client = Client(name, email, phone, client_type, preference)
    self.people.append(client)
    print("Client registered successfully.")

def register_agent(self, name, email, phone, agency):
    agent = Agent(name, email, phone, agency)
    self.people.append(agent)
    print("Agent registered successfully.")

def display_all_clients(self):
    print("Displaying All Clients:")
    for person in self.people:
        if isinstance(person, Client):
            person.display()

def display_all_agents(self):
    print("Displaying All Agents:")
    for person in self.people:
        if isinstance(person, Agent):
            person.display()

def add_property(self, property_type, property_id, location, price, **kwargs):
    if property_type == "Villa":
        property_obj = Villa(property_id, location, price, kwargs['bedrooms'], kwargs['size'])
    elif property_type == "Land":
        property_obj = Land(property_id, location, price, kwargs['size'])
    elif property_type == "Apartment":
            property_obj = Apartment(property_id, location, price, kwargs['floor'])
        elif property_type == "Office":
            property_obj = Office(property_id, location, price, kwargs['size'])
        elif property_type == "Home":
            property_obj = Home(property_id, location, price, kwargs['home_type'])
        else:
            print("Invalid property type.")
            return

        self.properties.append(property_obj)
        print("Property added successfully.")

    def view_all_properties(self):
        print("Viewing all properties...")
        for property_obj in self.properties:
            property_obj.display_property_details()

# Example Usage
if __name__ == "__main__":
    system = RealEstateManagementSystem()

    # Register clients and agents
    system.register_client("Alice", "alice@example.com", 1234567890, "Buyer", "Villa")
    system.register_agent("Bob", "bob@example.com", 9876543210, "RealEstate Co.")

    # Add properties
    system.add_property("Villa", "V001", "Downtown", 500000, bedrooms=4, size=2500)
    system.add_property("Land", "L001", "Uptown", 300000, size=2.5)

    # Display all clients, agents, and properties
    system.display_all_clients()
    system.display_all_agents()
    system.view_all_properties()
```

References

- https://docs.oracle.com/javase/
- https://www.sqltutorial.org/
- https://ieeexplore.ieee.org/
- https://hbr.org/
- https://www.ijcaonline.org/
- https://link.springer.com/
- https://www.forbes.com/sites/