Formal Method in Software Engineering

(SE-313)

VDM Specification Document For

“Property & Estate Management System”

Group Members

Izma Shafqat (SE-21003)

Inshara Iqbal (SE-21018)

Table of Contents

1. Scope:

The scope of the "**Property & Estate Management System"** project involves creating a simplified system for managing users, properties, and transactions. Users, categorized as Buyers and Sellers, can be added with unique usernames. The system accommodates two property types, Residential and Commercial, with specific attributes. Buyers can purchase listed properties, and the system calculates the total bill based on the acquired properties The platform provides a menu-driven interface for users to interact, displays user and property information, and includes basic error handling. The project is designed for extensibility and operates in-memory without persistent data storage, offering a console-based user interface. The simplified nature of the project serves educational purposes, and in a real-world scenario, additional features and complexities would be considered for a comprehensive real estate platform.

2. 4+1 View Model:

The 4+1 View Model is a software architecture documentation technique that provides five concurrent views of a system to address different concerns and stakeholders. It was introduced by Philippe Kruchten in 1995. The "4+1" in the model's name refers to the four primary views plus an additional view, which represents a set of scenarios or use cases. These views together help in understanding the architecture from various perspectives. Here are the four primary views and the additional view:

i. Logical View:

- **Focus:** Describes the system in terms of high-level abstractions, such as classes, objects, modules, and their relationships.

- **Purpose:** Provides an insight into the functional requirements of the system and how they are realized in terms of software components.

ii. Process View:

- **Focus:** Captures the dynamic aspects of the system, emphasizing the interactions among components, processes, and the flow of data.

- **Purpose**: Helps understand the system's runtime behavior, concurrency, and synchronization of processes.

iii. Physical View:

- **Focus:** Illustrates the distribution of software components across different physical nodes or hardware elements (e.g., computers, servers).

- **Purpose:** Addresses concerns related to system deployment, scalability, performance, and the physical distribution of components.

iv. Development View:

- **Focus:** Highlights the organization of the software into modules or subsystems and their dependencies.

- **Purpose:** Aids in understanding how the software is structured for development, including the organization of the source code, build processes, and development environments.

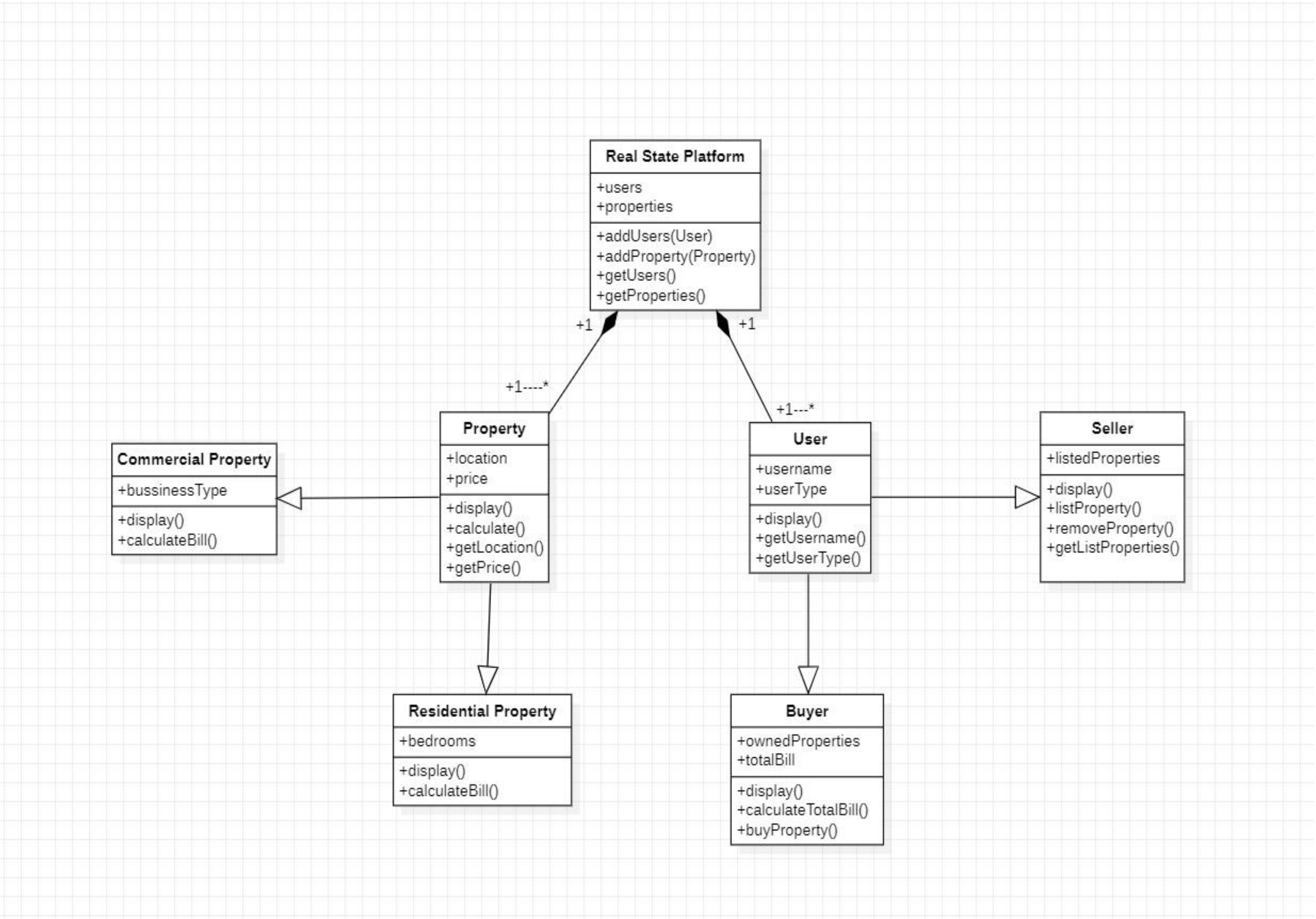
v. Scenario (or Use Case) View:

- **Focus:** Represents a set of use cases or scenarios that describe how the system interacts with its environment and users.

- **Purpose:** Provides a narrative or story-like description of the system's behavior under different conditions, helping to validate and refine the architecture.

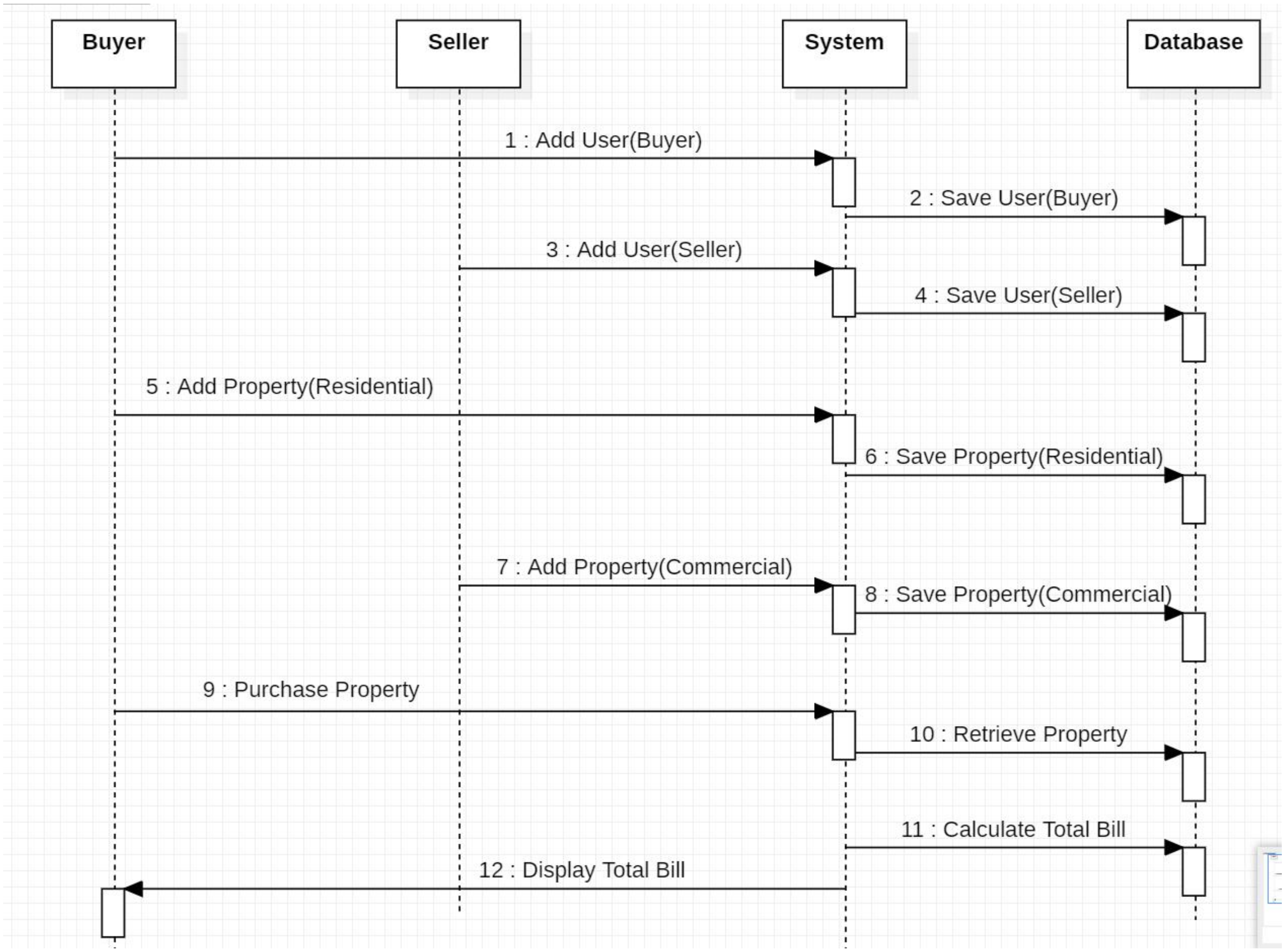
1. Logical View (Class Diagram):

The Logical View in the 4+1 View Model focuses on the high-level functional aspects and logical organization of a software system. It abstracts away implementation details and highlights key entities such as classes, objects, modules, and their relationships. By providing a conceptual representation of the system's structure, the Logical View aids stakeholders, including architects and developers, in understanding the overall architecture and how different components collaborate to achieve the system's functionality. This view is instrumental in communicating and validating functional requirements, facilitating clear discussions, and serving as a foundation for subsequent design and development activities.



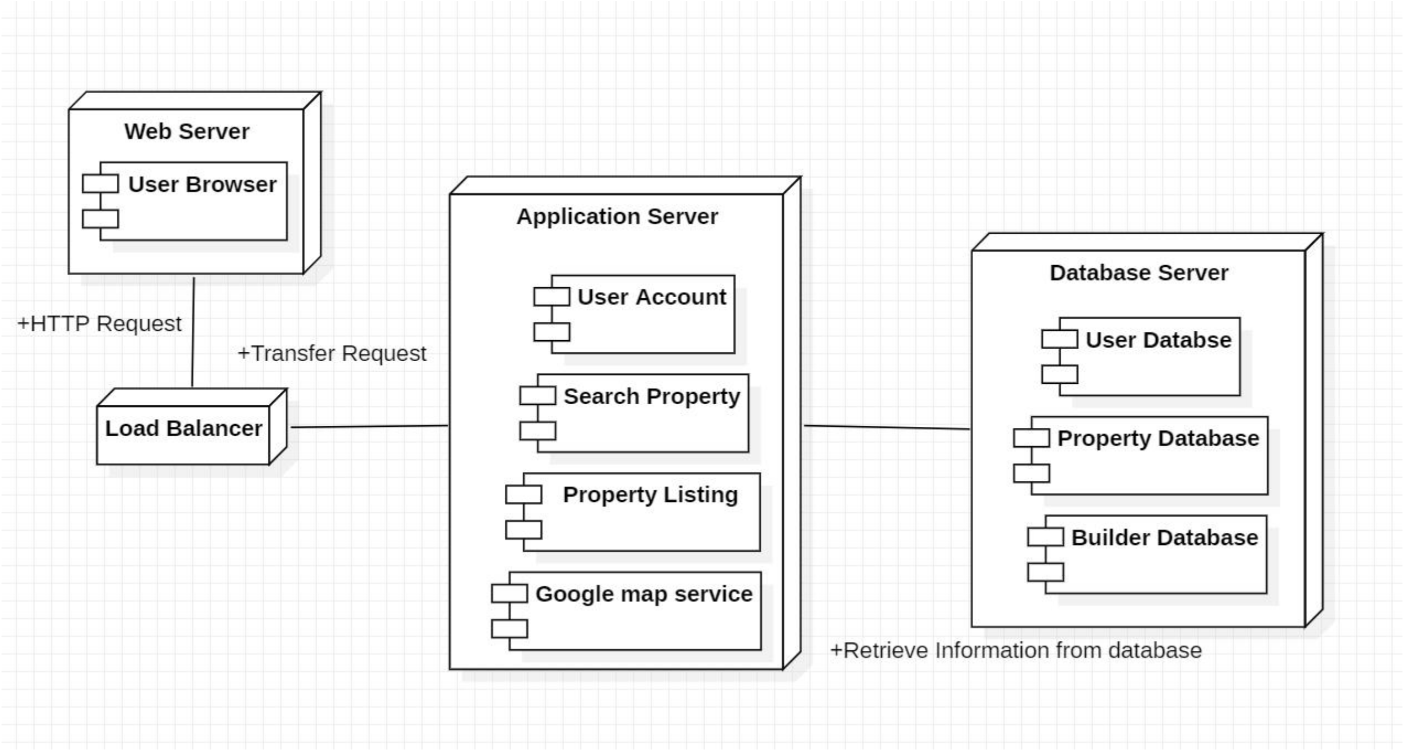
2. Process View (Sequence Diagram):

The Process View in the 4+1 View Model offers a dynamic perspective of a software system, emphasizing the interactions, concurrency, and data flow among various processes or components during runtime. This view provides insights into the system's behavior by illustrating how different elements collaborate and execute concurrently. It captures the runtime aspects of the software, depicting the flow of control and communication between processes, thereby addressing concerns related to performance, synchronization, and system behavior under various scenarios. The Process View is crucial for understanding the system's dynamic characteristics, aiding architects and developers in optimizing and ensuring the efficiency of the software during execution.



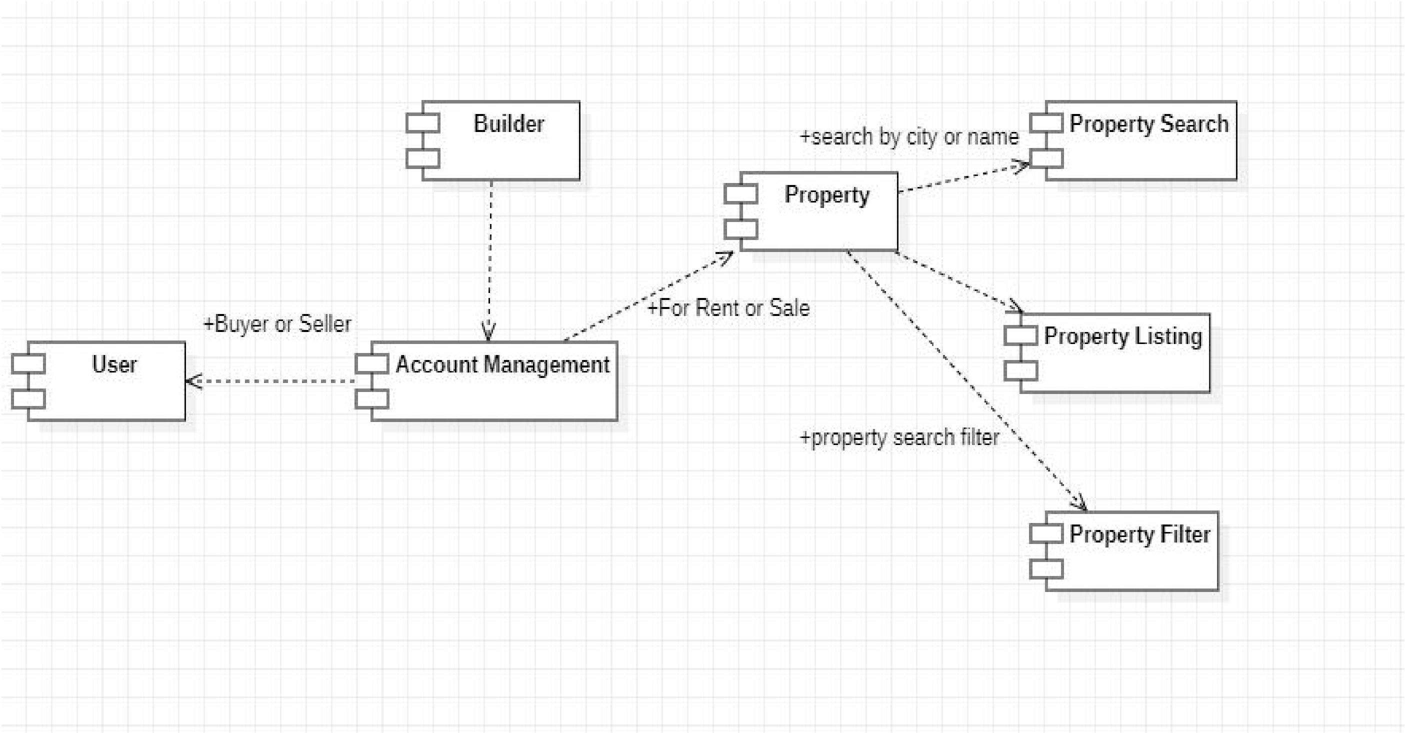
3. Physical View (Deployment Diagram):

The Physical View in the 4+1 View Model presents a spatial perspective of a software system, illustrating the distribution and deployment of software components across hardware nodes or infrastructure elements. This view focuses on how the system's logical elements, such as modules or components, are mapped onto physical entities like servers, computers, or other hardware resources. It addresses concerns related to scalability, performance, and resource allocation by providing insights into the system's physical architecture. The Physical View is essential for understanding the system's deployment configuration and ensuring that the software can effectively utilize available hardware resources while meeting performance and reliability requirements.



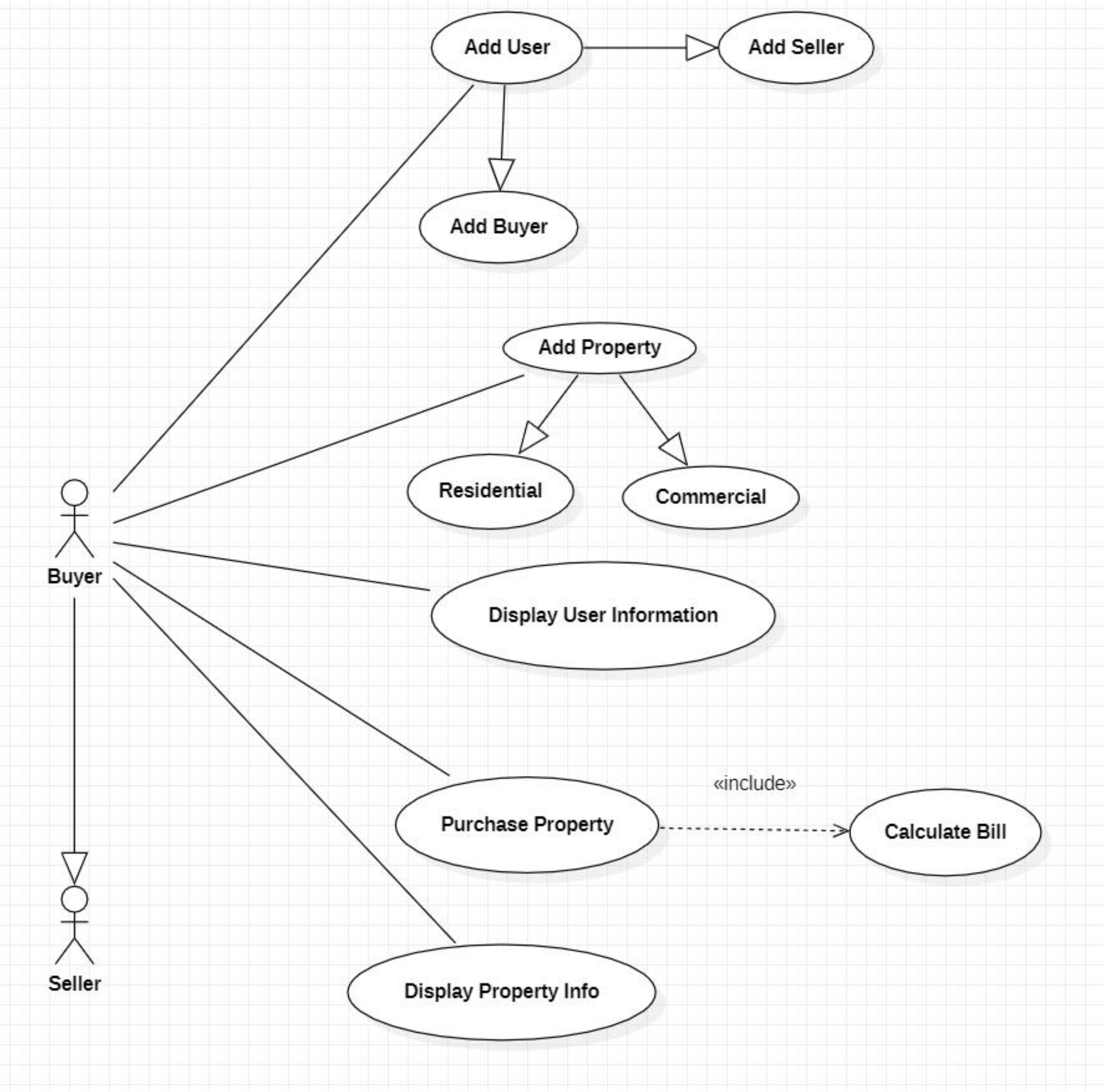
4. Development View (Component Diagram):

The Development View in the 4+1 View Model concentrates on the organization and structuring of software components during the development process. It delineates the modular breakdown of the system, highlighting dependencies, interfaces, and interactions between various software modules or subsystems. This view is instrumental for software developers and engineers as it provides a blueprint of the codebase, aiding in project organization, source code management, and collaboration. By emphasizing the development structure, the Development View contributes to maintainability, code reuse, and effective collaboration among development teams, ensuring a coherent and scalable software architecture throughout the software development life cycle.



5. Scenarios (UseCase Diagram):

The Scenarios View in the 4+1 View Model provides a narrative representation of a software system by describing a set of use cases or scenarios that illustrate how the system interacts with its environment and users. It offers a holistic perspective on the system's behavior under different conditions, helping stakeholders comprehend its functionality in real-world situations. This view serves as a valuable tool for validation, refinement, and communication of requirements by presenting concrete examples of system usage. By focusing on scenarios, the view enables stakeholders to evaluate the system's responsiveness to user actions and ensures that the architecture aligns with the intended user experience and operational context.



VDM++ Specification:

```
class Property
types
public Location = seq of char;
public Price = real;
instance variables
private location: Location;
private price: Price;
operations
- - Constructor for Property class
public Property: Location * Price ==> Property
Property(loc, pr) ==
location := loc;
price := pr;
functions
- - Getter for location
public getLocation: () ==> Location
getLocation() ==
return location;
- - Getter for price
public getPrice: () ==> Price
getPrice() ==
return price;
- - Display details of the property
public virtual display: () ==> ()
display() ==
IO`print("Location: ");
IO`println(location);
IO`print("Price: $");
IO`println(price);
- - Calculate the bill for the property
public virtual calculateBill: () ==> real
calculateBill() ==
return price;
end Property
class ResidentialProperty is subclass of Property
types
public Bedrooms = nat;
instance variables
private bedrooms: Bedrooms;
operations
- - Constructor for ResidentialProperty class
public ResidentialProperty: Location * Price *
Bedrooms ==> ResidentialProperty
ResidentialProperty(loc, pr, beds) ==
Property(loc, pr);
bedrooms := beds;
functions
- - Display details of the residential property
public display: () ==> ()display() ==
super.display();
IO`print("Bedrooms: ");
IO`println(bedrooms);
- - Calculate the bill for the residential property
public calculateBill: () ==> real
calculateBill() ==
return super.calculateBill(); - - For simplicity,
assuming no additional tax for residential
properties
end ResidentialProperty
class CommercialProperty is subclass of Property
types
public BusinessType = seq of char;
instance variables
private businessType: BusinessType;
```

```
operations
- - Constructor for CommercialProperty class
public CommercialProperty: Location * Price *
BusinessType ==> CommercialProperty
CommercialProperty(loc, pr, type) ==
Property(loc, pr);
businessType := type;
functions
- - Display details of the commercial property
public display: () ==> ()
display() ==
super.display();
IO`print("Business Type: ");
IO`println(businessType);
- - Calculate the bill for the commercial property
public calculateBill: () ==> real
calculateBill() ==
return super.calculateBill(); - - For simplicity,
assuming no additional tax for commercial
properties
end CommercialProperty
class User
types
public Username = seq of char;
public UserType = seq of char;
instance variables
private username: Username;
private userType: UserType;
operations
- - Constructor for User class
public User: Username * UserType ==> User
User(uname, type) ==
username := uname;
userType := type;
functions
- - Getter for username
public getUsername: () ==> Username
getUsername() ==
return username;
- - Getter for user type
public getUserType: () ==> UserType
getUserType() ==
return userType;
- - Display details of the user
public virtual display: () ==> ()
display() ==
IO`print("User Type: ");
IO`println(userType);
IO`print("Username: ");
IO`println(username);
end User
class Buyer is subclass of User
types
public TotalBill = real;
instance variables
private ownedProperties: seq of Property;
private totalBill: TotalBill;
operations
-- Constructor for Buyer class
public Buyer: Username ==> Buyer
Buyer(uname) ==
User(uname, "Buyer");
ownedProperties := [];
totalBill := 0.0;
functions
- - Display details of the buyer
public display: () ==> ()
display() ==
super.display();
IO`println("Owned Properties:");
for property in set ownedProperties do
property.display();
end for;
IO`print("Total Bill: $");
IO`println(totalBill);
- - Buy a property and update the total bill
public buyProperty: Property ==> ()
buyProperty(property) ==
ownedProperties := ownedProperties ^ [property];
totalBill := totalBill + property.calculateBill();
end Buyer
class Seller is subclass of User
instance variables
private listedProperties: seq of Property;operations
- - Constructor for Seller class
public Seller: Username ==> Seller
Seller(uname) ==
User(uname, "Seller");
listedProperties := [];
functions
- - Display details of the seller
public display: () ==> ()
display() ==
super.display();
IO`println("Listed Properties:");
for property in set listedProperties do
property.display();
end for;
- - List a property for sale
public listProperty: Property ==> ()
listProperty(property) ==
listedProperties := listedProperties ^ [property];
end Seller
class RealEstatePlatform
instance variables
private users: seq of User;
private properties: seq of Property;
operations
- - Constructor for RealEstatePlatform class
public RealEstatePlatform: () ==>
RealEstatePlatform
RealEstatePlatform() ==
users := [];
properties := [];
functions
- - Add a user to the platform
public addUser: User ==> ()
addUser(user) ==
users := users ^ [user];
- - Add a property to the platform
public addProperty: Property ==> ()
addProperty(property) ==
properties := properties ^ [property];
- - Get the list of users on the platform
public getUsers: () ==> seq of User
getUsers() ==
return users;
- - Get the list of properties on the platform
public getProperties: () ==> seq of Property
getProperties() ==
return properties;

end RealEstatePlatform
```

Development Code:

```
#include <iostream>
#include <vector>
#include <string>
#include <limits> // Include for handling invalid input

class Property {
protected:
    std::string location;
    double price;

public:
    Property(const std::string& loc, double pr) : location(loc), price(pr) {}

    virtual void display() const = 0;
    virtual double calculateBill() const = 0;

    std::string getLocation() const { return location; }
    double getPrice() const { return price; }
};

class ResidentialProperty : public Property {
private:
    int bedrooms;

public:
    ResidentialProperty(const std::string& loc, double pr, int beds)
        : Property(loc, pr), bedrooms(beds) {}

    void display() const override {
        std::cout << "Residential Property: " << location << ", Bedrooms: " << bedrooms << ", Price: $" << price << std::endl;
    }

    double calculateBill() const override {
        // Return the original price without any additional tax
        return price;
    }
};

class CommercialProperty : public Property {
private:
    std::string businessType;

public:
    CommercialProperty(const std::string& loc, double pr, const std::string& type)
        : Property(loc, pr), businessType(type) {}

    void display() const override {
        std::cout << "Commercial Property: " << location << ", Business Type: " << businessType << ", Price: $" << price << std::endl;
    }

    double calculateBill() const override {
        // Return the original price without any additional tax
        return price;
    }
};
```

```
class User {
protected:
    std::string username;
    std::string userType;

public:
    User(const std::string& uname, const std::string&
type) : username(uname), userType(type) {}

    virtual void display() const {
        std::cout << userType << " User: " <<
username << std::endl;
    }

    const std::string& getUsername() const {
        return username;
    }

    const std::string& getUserType() const {
        return userType;
    }
};

class Buyer : public User {
private:
    std::vector<Property*> ownedProperties;
    double totalBill;

public:
    Buyer(const std::string& uname) : User(uname,
"Buyer"), totalBill(0.0) {}

    void display() const override {
        User::display();
        std::cout << "Owned Properties:" << std::endl;
        for (const auto& property : ownedProperties) {
            property->display();
        }
        std::cout << "Total Bill: $" << totalBill <<
std::endl;
    }

    void buyProperty(Property* property) {
        ownedProperties.push_back(property);
        totalBill += property->calculateBill();
    }

    double calculateTotalBill() const {
        return totalBill;
    }
};

class Seller : public User {
private:
    std::vector<Property*> listedProperties;

public:
    Seller(const std::string& uname) : User(uname,
"Seller") {}

    void display() const override {
        User::display();
        std::cout << "Listed Properties:" << std::endl;
        for (const auto& property : listedProperties) {
            property->display();
        }
    }
    void listProperty(Property* property) {
        listedProperties.push_back(property);
    }

    void removeProperty(Property* property) {
        // Implement removal logic
    }

    const std::vector<Property*>&
getListedProperties() const {
        return listedProperties;
    }
};

class RealEstatePlatform {
private:
    std::vector<User*> users;
    std::vector<Property*> properties;

public:
    void addUser(User* user) {
        users.push_back(user);
    }

    void addProperty(Property* property) {
        properties.push_back(property);
    }

    const std::vector<User*>& getUsers() const {
        return users;
    }

    const std::vector<Property*>& getProperties()
const {
        return properties;
    }
};

class RealEstatePlatformTester {
private:
    RealEstatePlatform platform;

public:
    void run() {
        int choice;
        do {
            displayMenu();
            std::cout << "Enter your choice: ";

            try {
                std::cin >> choice;

                if (!std::cin.good()) {
                    std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>:
:max(), '\n');
                    throw std::runtime_error("Invalid input.
Please enter a number.");
                }

                handleChoice(choice);

            } catch (const std::exception& e) {
                std::cerr << "Error: " << e.what() <<
std::endl;
                std::cin.clear();
```

```
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            }

        } while (choice != 8);
    }

private:
    void displayMenu() {
        std::cout << "\n----- Real Estate Platform Menu -----\n";
        std::cout << "1. Add Buyer\n";
        std::cout << "2. Add Seller\n";
        std::cout << "3. Add Residential Property\n";
        std::cout << "4. Add Commercial Property\n";
        std::cout << "5. Display Users\n";
        std::cout << "6. Display Properties\n";
        std::cout << "7. Buy Property\n";
        std::cout << "8. Exit\n";
        std::cout << "------------------------------------\n";
    }

    void handleChoice(int choice) {
        switch (choice) {
            case 1: {
                std::string username;
                std::cout << "Enter buyer's username: ";
                std::cin >> username;
                platform.addUser(new Buyer(username));
                break;
            }
            case 2: {
                std::string username;
                std::cout << "Enter seller's username: ";
                std::cin >> username;
                platform.addUser(new Seller(username));
                break;
            }
            case 3: {
                std::string location;
                double price;
                int bedrooms;
                std::cout << "Enter property location: ";
                std::cin >> location;
                std::cout << "Enter property price: $";
                std::cin >> price;
                std::cout << "Enter number of bedrooms: ";
                std::cin >> bedrooms;
                platform.addProperty(new ResidentialProperty(location, price, bedrooms));
                break;
            }
            case 4: {
                std::string location;
                double price;
                std::string businessType;
                std::cout << "Enter property location: ";
                std::cin >> location;
                std::cout << "Enter property price: $";
                std::cin >> price;
                std::cout << "Enter business type: ";
                std::cin.ignore(); // Ignore newline character
                std::getline(std::cin, businessType);
                platform.addProperty(new CommercialProperty(location, price, businessType));
                break;
            }
            case 5: {
                const auto& users = platform.getUsers();
                std::cout << "Users on the Platform:" <<
std::endl;
                for (const auto& user : users) {
                    std::cout << "Username: " << user-
>getUsername() << ", Type: " << user-
>getUserType() << std::endl;
                }
                break;
            }
            case 6: {
                const auto& properties =
platform.getProperties();
                std::cout << "Properties on the Platform:"
<< std::endl;
                for (const auto& property : properties) {
                    property->display();
                }
                break;
            }
            case 7: {
                int buyerIndex, propertyIndex;
                const auto& users = platform.getUsers();
                const auto& properties =
platform.getProperties();

                std::cout << "Buyers on the Platform:" <<
std::endl;
                for (size_t i = 0; i < users.size(); ++i) {
                    if (users[i]->getUserType() == "Buyer")
{
                        std::cout << i << ". " << users[i]-
>getUsername() << std::endl;
                    }
                }
                std::cout << "Enter the index of the buyer:
";
                std::cin >> buyerIndex;

                std::cout << "Properties on the Platform:"
<< std::endl;
                for (size_t i = 0; i < properties.size(); ++i)
{
                    std::cout << i << ". ";
                    properties[i]->display();
                }
                std::cout << "Enter the index of the
property to buy: ";
                std::cin >> propertyIndex;

                if (buyerIndex >= 0 && buyerIndex <
users.size() &&
                    propertyIndex >= 0 && propertyIndex
< properties.size()) {
                    Buyer* buyer =
dynamic_cast<Buyer*>(users[buyerIndex]);
                    Property* property =
properties[propertyIndex];

                    if (buyer && property) {
                        buyer->buyProperty(property);
                        std::cout << "Property bought
successfully!\n";

                        // Display the buyer's information,
including the total bill
                        buyer->display();
                    } else {
                        std::cout << "Invalid buyer or
property!\n";
                    }
                } else {
                    std::cout << "Invalid indices!\n";
                }
                break;
            }
            case 8:
                std::cout << "Exiting the program.\n";
                break;
            default:
                std::cout << "Invalid choice. Please try
again.\n";
        }
    }
};

int main() {
    RealEstatePlatformTester tester;
    tester.run();
    return 0;
}
```

Testing Class:

```
class RealEstatePlatformTester {
private:
    RealEstatePlatform platform;

public:
    void run() {
        int choice;
        do {
            displayMenu();
            std::cout << "Enter your choice: ";

            try {
                std::cin >> choice;

                if (!std::cin.good()) {
                    std::cin.clear();
                    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                    throw std::runtime_error("Invalid input. Please enter a number.");
                }

                handleChoice(choice);

            } catch (const std::exception& e) {
                std::cerr << "Error: " << e.what() << std::endl;
                std::cin.clear();
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            }

        } while (choice != 8);
    }

private:
    void displayMenu() {
        std::cout << "\n----- Real Estate Platform Menu -----\n";
        std::cout << "1. Add Buyer\n";
        std::cout << "2. Add Seller\n";
        std::cout << "3. Add Residential Property\n";
        std::cout << "4. Add Commercial Property\n";
        std::cout << "5. Display Users\n";
        std::cout << "6. Display Properties\n";
        std::cout << "7. Buy Property\n";
        std::cout << "8. Exit\n";
        std::cout << "-------------------------------------\n";
    }

    void handleChoice(int choice) {
        switch (choice) {
            case 1: {
                std::string username;
                std::cout << "Enter buyer's username: ";
                std::cin >> username;
                platform.addUser(new Buyer(username));
                break;
            }
            case 2: {
                std::string username;
                std::cout << "Enter seller's username: ";
                std::cin >> username;
                platform.addUser(new Seller(username));
                break;
            }
            case 3: {
                std::string location;
                double price;
                int bedrooms;
                std::cout << "Enter property location: ";
                std::cin >> location;
                std::cout << "Enter property price: $";
                std::cin >> price;
                std::cout << "Enter number of bedrooms: ";
                std::cin >> bedrooms;
                platform.addProperty(new ResidentialProperty(location, price, bedrooms));
                break;
            }
            case 4: {
                std::string location;
                double price;
                std::string businessType;
                std::cout << "Enter property location: ";
                std::cin >> location;
                std::cout << "Enter property price: $";
                std::cin >> price;
                std::cout << "Enter business type: ";
                std::cin.ignore(); // Ignore newline character
                std::getline(std::cin, businessType);
```

```
                platform.addProperty(new CommercialProperty(location, price, businessType));
                break;
            }
            case 5: {
                const auto& users = platform.getUsers();
                std::cout << "Users on the Platform:" << std::endl;
                for (const auto& user : users) {
                    std::cout << "Username: " << user->getUsername() << ", Type: " << user->getUserType() <<
std::endl;
                }
                break;
            }
            case 6: {
                const auto& properties = platform.getProperties();
                std::cout << "Properties on the Platform:" << std::endl;
                for (const auto& property : properties) {
                    property->display();
                }
                break;
            }
            case 7: {
                int buyerIndex, propertyIndex;
                const auto& users = platform.getUsers();
                const auto& properties = platform.getProperties();

                std::cout << "Buyers on the Platform:" << std::endl;
                for (size_t i = 0; i < users.size(); ++i) {
                    if (users[i]->getUserType() == "Buyer") {
                        std::cout << i << ". " << users[i]->getUsername() << std::endl;
                    }
                }
                std::cout << "Enter the index of the buyer: ";
                std::cin >> buyerIndex;
            std::cout << "Properties on the Platform:" << std::endl;
            for (size_t i = 0; i < properties.size(); ++i) {
                std::cout << i << ". ";
                properties[i]->display();
            }
            std::cout << "Enter the index of the property to buy: ";
            std::cin >> propertyIndex;

            if (buyerIndex >= 0 && buyerIndex < users.size() &&
                propertyIndex >= 0 && propertyIndex < properties.size()) {
                Buyer* buyer = dynamic_cast<Buyer*>(users[buyerIndex]);
                Property* property = properties[propertyIndex];

                if (buyer && property) {
                    buyer->buyProperty(property);
                    std::cout << "Property bought successfully!\n";

                    // Display the buyer's information, including the total bill
                    buyer->display();
                } else {
                    std::cout << "Invalid buyer or property!\n";
                }
            } else {
                std::cout << "Invalid indices!\n";
            }
            break;
        }
        case 8:
            std::cout << "Exiting the program.\n";
            break;
        default:
            std::cout << "Invalid choice. Please try again.\n";
    }
}
```

```
};
```