**School of Information Technologies and Engineering, ADA University**

**CSCI3509 – Intro to Software Engineering**

**Fall 2024 – 12/22/2024**

**Assignment 3 by Group#5**

**Complete Class Diagram of the System**

A black and white image of text

Description automatically generated

**Design Rationale**

Our report explains the design decisions for the Real Estate Parallax System's implementation, focusing on why certain choices were made to create a maintainable and flexible system. We tried to guide the design with well-established object-oriented principles and design patterns. The Single Responsibility Principle (SRP) was a key consideration in the design. Each class was designed to have only one responsibility, which makes the system easier to understand and maintain. For instance, the Seller class is dedicated to handling seller-specific tasks, the Property class manages details about properties, and the Bid class focuses on bid-related actions. By this separation of responsibilities, we achieved simplifications on maintenance and debugging, as each class is dedicated to a single responsibility.The Open/Closed Principle (OCP) was also key in our design. Classes were designed to be extendable without modifying existing code, ensuring that the system remains stable as it evolves. For example, the Payment class can be extended to support new payment methods without changing its core functionality. This principle allows for new features to be added with minimal risk of breaking existing functionality, promoting a more adaptable system.The Liskov Substitution Principle (LSP) was applied to ensure that subclasses could be used interchangeably with their base classes without causing errors. This principle ensures consistency and reliability across the system. For example, if will have a VIPSeller class in the future, it could replace a Seller object without causing issues, maintaining the expected behavior.By using the Interface Segregation Principle (ISP), the design avoids forcing classes to implement methods they do not need. Specific boundaries were created for different functionalities, preventing unnecessary dependencies. For example, separate methods for Login and Property ensure that an Admin class does not have to implement methods meant for a Buyer or Seller. This approach reduces unnecessary dependencies and enhances modularity.The Dependency Inversion Principle (DIP) was followed to ensure that high-level modules do not depend on low-level modules. Instead, both are structured to work through a defined contract or shared set of methods. This was achieved through dependency addition, making the system more flexible and easier to test and maintain. For example, a Seller class does not create a Property class directly but operates with a mechanism that allows the specific implementation to be supplied at runtime. This design choice enhances the system's flexibility, testability, and maintainability.Several design patterns were also included to solve common problems and improve the system’s design. As an example of implementation, we can show how the observer pattern was applied to allow an object to notify other objects about changes in its state. This is useful in scenarios such as a Seller class notifies Buyer class by using Message class when a requested property is placed or its status changes. By using some design principles and patterns, we also tried to keep the implementation simple meanwhile. As an example, the singleton pattern was not used because the system requires multiple instances of certain classes like Admin, Agent, Property, and others to represent different entities and their specific roles. Using the singleton pattern would have restricted the creation of more than one instance, which is not suitable for this context where each entity needs to maintain its own state and behavior. Instead, we focused on maintaining modularity and clarity in the implementation to ensure scalability and ease of understanding.

The design is focused on extensibility, allowing future requirements to be accommodated with minimal impact on existing code. The compact design ensures that each class and module is self-contained, promoting reusability and ease of maintenance. The use of established design patterns like observer further enhances the system’s ability to adapt to changing requirements without significant refactoring. The design decisions for the “Parallax Real Estate System” are guided by fundamental object-oriented principles and established design patterns. These decisions ensure that the system is not only functional and efficient but also maintainable and extensible. By focusing on why these principles and patterns are applied, the design rationale provides a solid justification for the architecture of the system, ensuring its longevity and adaptability to future needs.

**Source Code**

***You will find the source code attached:***

[***https://github.com/CSCI3509-2024/parallax-real-estate-10109-group5/tree/main***](https://github.com/CSCI3509-2024/parallax-real-estate-10109-group5/tree/main)

***Manual Blackbox Testing and Traceability Matrix***

Our code structure supports below test cases:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Tester** | **Inputs** | **Expected Output** | **Actual Output** | **Steps to Reproduce** | **Pass/Fail** |
| TestCase1 | Login functionality test | Shahla | Username: "B001", Password: "password" | Login successful, Buyer menu displayed | Login successful, Buyer menu displayed | 1. Launch app 2. Enter valid credentials 3. Click "Enter" | Pass |
| TestCase2 | Invalid login | Laman | Username: "invalidUser", Password: "wrongPass" | "Invalid credentials please try again” | "Invalid credentials please try again” | 1. Launch app 2. Enter invalid credentials 3. Click "Enter" | Pass |
| TestCase3 | Search property by suburb name | Shahla | Suburb: "Brookline" | List of properties in "Brookline" suburb displayed | List of properties displayed | 1. Login as buyer 2. Select "Search Property" 3. Enter suburb name "Brookline" | Pass |
| TestCase4 | Create property (Seller) | Laman | Property Details: Type: "Villa", Suburb: "Greenwood", Price: "150K" | Property created successfully and added to listings | Property created successfully! | 1. Login as seller 2. Select "Create Property" 3. Enter property type | Pass |
| TestCase5 | Edit Property (Seller) | Laman | Property ID: 4  New Suburb: “Greenwood”  New Price: 550000 | Property edited successfully and added to listings | Property edited successfully!  Property ID: 4  Type: “Villa”  Suburb: “Greenwood”  Price: 550000 | 1. Login as seller 2. Select "Edit Property" 3. Enter property type  4. Enter suburb name  5. Enter price | Pass |
| TestCase 6 | Edit Property Price only (seller) | Shahla | Property ID: 4  New Suburb: “Greenwood”  New Price: 550000 | Only property price edited successfully and added to listings | Whole details were edited not only price. | 1. Login as seller 2. Select "Edit Property" 3. Enter property type  4. Enter suburb name  5. Enter price | Fail |
| TestCase7 | View contract between buyer/seller | Laman | Buyer ID: "buyer1", Property ID: "P123" | Message sent to seller for negotiation | Message was sent successfully | 1. Login as buyer 2. Select “Show Interest” and then "View Contract" 3. Enter details | Pass |
| TestCase8 | Process payment | Shahla | Payment Details: Property ID: "P1", Amount: "$150,000" | Payment processed successfully; balance updated | Payment processed successfully | 1. Login as buyer 2. Select "Proceed with Payment" 3. Enter payment details | Pass |
| TestCase 9 | Unarchiving the archived property | Laman | Property ID: "P1" | Property unarchived successfully. | Property was not unarchived successfully. | Login as seller 2. Select Property ID | Fail |
| TestCase10 | Archive property (Seller) | Shahla | Property ID: "P2" | Property archived successfully, no longer visible | Property archived successfully | 1. Login as seller 2. Select "Archive Property" 3. Enter property ID | Pass |

**Traceability Matrix**

|  |  |  |
| --- | --- | --- |
| **Requirement ID** | **Requirement Description** | **Test Case ID(s)** |
| 1 | Login functionality for all users | TestCase1, TestCase2 |
| 2 | Buyers and sellers can search for properties | TestCase3 |
| 3 | Sellers can create/edit property listings | TestCase4, TestCase5, TestCase6 |
| 4 | Buyers and sellers can view contracts | TestCase7 |
| 5 | Payment system for property transactions | TestCase8 |
| 6 | Sellers can archive/unarchive properties | TestCase9, TestCase10 |

***System Demonstration***

***You will find the demonstration videos attached:***

[***https://github.com/CSCI3509-2024/parallax-real-estate-10109-group5/tree/main***](https://github.com/CSCI3509-2024/parallax-real-estate-10109-group5/tree/main)

***Recourses***

* UML Diagrams, “Online Shopping Domain UML Diagram Example,” [Online]. Available: <https://www.uml-diagrams.org/examples/online-shopping-domain-uml-diagram-example.html>. [Accessed: Dec. 6, 2024].
* Design Real Estate Management System by usingJava Language <https://iasj.net/iasj/download/053a4addee4e63cb> [Online]. [Accessed: Dec. 9, 2024].
* Java Real Estate Management System Source Code - Real Estate Management System Project Source Code Using JAVA NetBeans And MySQL Database <https://1bestcsharp.blogspot.com/2020/02/java-real-estate-management-system.html> [Online]. [Accessed: Dec. 11, 2024].
* Black box testing - <https://www.javatpoint.com/black-box-testing> [Online]. [Accessed: Dec. 13, 2024].

**Laman Panakhova16882 and Shahla Azizova11542**

|  |
| --- |
|  |
| **I used AI in the following ways:**  I used <ADD AI tool> (<ADD link if needed>) to <ADD how used> (<ADD number> iterations/drafts). I modified the outputs in <ADD ways>.  <Students to insert very brief explanation of AI tool(s) and how used>  <Students to list the prompts used and provide screenshots of prompts>    **AI was not used.**  Prompt screenshots: |

**Reflection on AI use in the assessment.**

Briefly reflect on what you learned through collaborating with AI tools in the production of this assessment. [100-200 words]

**During the process mainly course materials, professor feedbacks, and consultations were used and referenced to.**

**Group Assessment Academic Integrity Declaration**

We, the undersigned, declare that the work submitted for this assessment is the original work of our group and complies with the university’s academic integrity policy. We understand that:

* All group members have contributed to the development of original ideas, and no part of the submission has been copied or derived from external sources without appropriate acknowledgment.
* Academic misconduct, including but not limited to plagiarism, collusion with other groups, and any form of cheating, is strictly prohibited.
* The ethical use of AI tools and technologies to support idea generation or research is permitted, provided that all AI contributions are properly acknowledged and cited.

We acknowledge that any detected instance of academic misconduct, either by an individual or the group, will result in a score of **0 (zero)** for the entire assessment. In severe or repeated cases, it may lead to **failure** of the course or further disciplinary action as determined by the university.

By signing this declaration, we confirm that we have read, understood, and agree to abide by the terms outlined above.

**Group Members:**

| **Name** | **Student ID** | **Signature** |
| --- | --- | --- |
| Laman Panakhova | 16882 | A close-up of a signature  Description automatically generated |
| Shahla Azizova | 11542 | A close-up of a signature  Description automatically generated |
| Ramil Khaspoladov | 13582 | A signature on a white background  Description automatically generated |
|  |  |  |

**Date: 12/22/2024**