BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC GIA ĐỊNH**

**KHOA: CÔNG NGHỆ THÔNG TIN**

Logo, company name

Description automatically generated

**TIỂU LUẬN**

**QUẢN LÝ DANH SÁCH CÁC GIAO DỊCH NHÀ ĐẤT**

**MÔN: MẪU THIẾT KẾ CHO PHẦN MỀM**

Ngành: **KỸ THUẬT PHẦN MỀM**

Giảng viên hướng dẫn: **LÊ HUỲNH PHƯỚC**

Sinh viên thực hiện: **NGUYỄN VĂN KHÁNH**

MSSV: **22150129**

Lớp: **221402**

TP. Hồ Chí Minh, tháng 07 năm 2024

**Khoa: Công nghệ thông tin**

**NHẬN XÉT VÀ CHẤM ĐIỂM CỦA GIẢNG VIÊN**

**TIỂU LUẬN MÔN: Mẫu thiết kế cho phần mềm**

1. **Họ và tên sinh viên: Nguyễn Văn Khánh**
2. **Tên đề tài: Quản lý danh sách các giao dịch nhà đất**
3. **Nhận xét:**

***Những kết quả đạt được:***

……………………………………………………………………………………..

……………………………………………………………………………………..

……………………………………………………………………………………..

……………………………………………………………………………………..

***Những hạn chế:***

……………………………………………………………………………………..

……………………………………………………………………………………..

……………………………………………………………………………………..

……………………………………………………………………………………..

1. **Điểm đánh giá** *(theo thang điểm 10, làm tròn đến 0.5):*

Sinh viên: ……………………………………………………………...…………..

Điểm số:……………… Điểm chữ: …...…………..…...…….………...…………..

*TP. HCM, .… tháng 07 năm 2024*

**Giảng viên chấm thi 01 Giảng viên chấm thi 02**

*(Ký và ghi rõ họ tên) (Ký và ghi rõ họ tên)*

**LỜI CAM ĐOAN**

Em xin cam đoan đề tài tiểu luận: “ Quản lý danh sách các giao dịch nhà đất ” do Nguyễn Văn Khánh tìm hiểu và thực hiện.

Em đã kiểm tra dữ liệu theo quy định hiện hành.

Kết quả bài làm của đề tài “ Quản lý danh sách các giao dịch nhà đất ” là trung thực và không sao chép từ bất kì bài tập của các cá nhân khác.

*TP. HCM, …. tháng 07 năm 2024*

**Sinh viên cam đoan**

(Ký và ghi rõ họ tên)

**LỜI CẢM ƠN**

Lời đầu tiên, em xin được gửi lời cảm ơn chân thành nhất đến thầy Lê Huỳnh Phước. Trong quá trình học tập và tìm hiểu môn “ Mẫu thiết kế cho phần mềm ”, em đã nhận được rất nhiều sự quan tâm, giúp đỡ, hướng dẫn tâm huyết và tận tình của thầy. Thầy đã giúp em tích lũy thêm nhiều kiến thức về môn học này để có thể hoàn thành được bài tiểu luận về đề tài “ Quản lý danh sách các giao dịch nhà đất ”.

Trong quá trình làm bài chắc chắn khó tránh khỏi những thiếu sót. Do đó, em kính mong nhận được những lời góp ý của thầy để bài tiểu luận của em ngày càng hoàn thiện hơn.

Em xin chân thành cảm ơn!

**MỤC LỤC**

[**CHƯƠNG 1. CƠ SỞ LÝ THUYẾT** 1](#_Toc173108626)

[**1.1.** **Mẫu thiết kế** 1](#_Toc173108627)

[1.1.1. Định nghĩa mẫu thiết kế 1](#_Toc173108628)

[1.1.2. Các loại mẫu thiết kế 1](#_Toc173108629)

[1.1.3. Lợi ích của mẫu thiết kế 1](#_Toc173108630)

[**1.2.** **Mẫu thiết kế ECB** 2](#_Toc173108631)

[1.2.1. Định nghĩa ECB 2](#_Toc173108632)

[1.2.2. Lợi ích 2](#_Toc173108633)

[1.2.3. Bất lợi 2](#_Toc173108634)

[**1.3.** **Mẫu thiết kế MVC** 3](#_Toc173108635)

[1.3.1. Định nghĩa MVC 3](#_Toc173108636)

[1.3.2. Lợi ích 3](#_Toc173108637)

[1.3.3. Bất lợi 4](#_Toc173108638)

[**1.4.** **Mẫu thiết kế Observer** 4](#_Toc173108639)

[1.4.1. Định nghĩa Observer 4](#_Toc173108640)

[1.4.2. Lợi ích 5](#_Toc173108641)

[1.4.3. Bất lợi 5](#_Toc173108642)

[**1.5.** **Mẫu thiết kế Singleton** 6](#_Toc173108643)

[1.5.1. Định nghĩa Singleton 6](#_Toc173108644)

[1.5.2. Lợi ích 6](#_Toc173108645)

[1.5.3. Bất lợi 7](#_Toc173108646)

[**1.6.** **Mẫu thiết kế CommandProcessor** 7](#_Toc173108647)

[1.6.1. Định nghĩa CommandProcessor 7](#_Toc173108648)

[1.6.2. Lợi ích 8](#_Toc173108649)

[1.6.3. Bất lợi 9](#_Toc173108650)

[**1.7.** **Mẫu thiết kế Three Layer** 9](#_Toc173108651)

[1.7.1. Định nghĩa Three Layer 9](#_Toc173108652)

[1.7.2. Lợi ích 10](#_Toc173108653)

[1.7.3. Bất lợi 11](#_Toc173108654)

[**CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ CHƯƠNG TRÌNH** 12](#_Toc173108655)

[**2.1. Phân tích chương trình** 12](#_Toc173108656)

[2.1.1. Mô tả đề tài 12](#_Toc173108657)

[2.1.2. Yêu cầu đề tài 12](#_Toc173108658)

[**2.2. Biểu đồ Use Case tổng quan** 13](#_Toc173108659)

[**2.3. Đặc tả Use Case** 14](#_Toc173108660)

[2.3.1. Thêm giao dịch 14](#_Toc173108661)

[2.3.2. Sửa giao dịch 15](#_Toc173108663)

[2.3.3. Xóa giao dịch 16](#_Toc173108665)

[2.3.4. Tìm kiếm theo mã giao dịch 16](#_Toc173108667)

[2.3.5. Tính tổng số lượng giao dịch loại nhà/đất 17](#_Toc173108669)

[2.3.6. Tính trung bình thành tiền của giao dịch nhà/đất 18](#_Toc173108671)

[2.3.7. Tính thành tiền của giao dịch 19](#_Toc173108673)

[2.3.8. Xuất ra các giao dịch của tháng 19](#_Toc173108675)

[**2.4. Sơ đồ thiết kế** 21](#_Toc173108677)

[2.4.1. Tầng Presentation 21](#_Toc173108678)

[2.4.2. Tầng Domain 21](#_Toc173108679)

[2.4.3. Tầng Persistence 21](#_Toc173108680)

[**2.5. Mối quan hệ giữa các tầng** 22](#_Toc173108681)

[2.5.1. Tầng Presentation 22](#_Toc173108682)

[2.5.2. Tầng Domain 22](#_Toc173108683)

[2.4.3. Tầng Persistence 23](#_Toc173108684)

[2.5.4. Mối quan hệ giữa các tầng 23](#_Toc173108685)

[**CHƯƠNG 3. CÀI ĐẶT THỰC NGHIỆM VÀ KẾT QUẢ CHƯƠNG TRÌNH** 25](#_Toc173108686)

[**3.1. Cài đặt chương trình** 25](#_Toc173108687)

[3.1.1. Phiên bản sử dụng 25](#_Toc173108688)

[3.1.2. Tạo bảng transaction trong cơ sở dữ liệu 25](#_Toc173108689)

[3.1.3. Thêm dữ liệu vào bảng transaction trong cơ sở dữ liệu 25](#_Toc173108691)

[**3.2. Kết quả chương trình** 26](#_Toc173108693)

[**CHƯƠNG 4. KẾT LUẬN VÀ ĐÁNH GIÁ** 33](#_Toc173108694)

[**4.1. Kết quả** 33](#_Toc173108695)

[**4.2. Kiến nghị phát triển trong tương lai** 33](#_Toc173108696)

[**TÀI LIỆU THAM KHẢO** 34](#_Toc173108697)

**DANH MỤC HÌNH ẢNH**

[Hình 1.3.1.1. Mẫu MVC 3](#_Toc173108781)

[Hình 1.4.1.1. Mẫu Observer 5](#_Toc173108782)

[Hình 1.5.1.1. Mẫu Singleton 6](#_Toc173108783)

[Hình 1.6.1.1. Mẫu Command Processor 8](#_Toc173108784)

[Hình 1.7.1.1. Mẫu Three Layer 10](#_Toc173108785)

[Hình 2.2.1. Biểu đồ Use Case tổng quan 13](#_Toc173108786)

[Hình 2.4.1.1. Tầng Presentation 21](#_Toc173108787)

[Hình 2.4.1.2. Tầng Domain 21](#_Toc173108788)

[Hình 2.4.1.2. Tầng Persistence 22](#_Toc173108789)

[Hình 3.2.1. Giao diện chính của chương trình 26](#_Toc173108790)

[Hình 3.2.2. Thêm giao dịch mới với mã giao dịch là 13 27](#_Toc173108791)

[Hình 3.2.3. Sửa đơn giá của mã giao dịch 13 từ 3000 thành 2500 27](#_Toc173108792)

[Hình 3.2.4. Đã xóa mã giao dịch 13 28](#_Toc173108793)

[Hình 3.2.5. Tìm mã giao dịch 11 28](#_Toc173108794)

[Hình 3.2.6. Tính thành tiền mã giao dịch 11 29](#_Toc173108795)

[Hình 3.2.7. Tính tổng số lượng loại nhà thường 29](#_Toc173108796)

[Hình 3.2.8. Tính tổng số lượng loại đất A 29](#_Toc173108797)

[Hình 3.2.9. Tính trung bình thành tiền nhà (Toàn bộ các tháng) 30](#_Toc173108798)

[Hình 3.2.10. Tính trung bình thành tiền đất (Toàn bộ các tháng) 30](#_Toc173108799)

[Hình 3.2.11. Xuất các giao dịch tháng 1 31](#_Toc173108800)

[Hình 3.2.12. Tính trung bình thành tiền nhà (Tháng 1) 31](#_Toc173108801)

[Hình 3.2.13. Tính trung bình thành tiền đất (Tháng 1) 32](#_Toc173108802)

**DANH MỤC BẢNG**

[Bảng 2.3.1.1. Thêm giao dịch 15](#_Toc173108865)

[Bảng 2.3.2.1. Sửa giao dịch 15](#_Toc173108866)

[Bảng 2.3.3.1. Xoá giao dịch 16](#_Toc173108867)

[Bảng 2.3.4.1. Tìm kiếm giao dịch 17](#_Toc173108868)

[Bảng 2.3.5.1. Tính tổng số lượng giao dịch loại nhà/đất 18](#_Toc173108869)

[Bảng 2.3.6.1. Tính trung bình thành tiền của giao dịch nhà/đất 18](#_Toc173108870)

[Bảng 2.3.7.1. Tính thành tiền của giao dịch 19](#_Toc173108871)

[Bảng 2.3.8.1. Xuất ra các giao dịch của tháng 20](#_Toc173108872)

[Bảng 3.1.2. Bảng transaction 25](#_Toc173108873)

[Bảng 3.1.3.1. Dữ liệu 26](#_Toc173108874)

# **CHƯƠNG 1.** **CƠ SỞ LÝ THUYẾT**

## **Mẫu thiết kế**

* + 1. Định nghĩa mẫu thiết kế

Mẫu thiết kế là một mô tả hoặc khuôn mẫu cho cách giải quyết một vấn đề cụ thể trong một ngữ cảnh nhất định. Nó không phải là một đoạn mã cụ thể mà là một khung lý thuyết hoặc kế hoạch hướng dẫn các nhà phát triển phần mềm trong việc giải quyết vấn đề đó. Mẫu thiết kế có thể được sử dụng trong nhiều ngữ cảnh khác nhau và không bị ràng buộc bởi một ngôn ngữ lập trình cụ thể.

* + 1. Các loại mẫu thiết kế

Mẫu thiết kế thường được chia thành ba loại chính:

- Mẫu thiết kế tạo dựng (Creational Patterns): Tập trung vào việc tạo đối tượng, ẩn đi chi tiết về cách thức khởi tạo và quản lý chúng. Ví dụ: Singleton, Factory Method, Abstract Factory.

- Mẫu thiết kế cấu trúc (Structural Patterns): Tập trung vào việc cấu trúc các đối tượng và lớp để tạo ra những hệ thống phần mềm lớn và linh hoạt. Ví dụ: Adapter, Composite, Proxy.

- Mẫu thiết kế hành vi (Behavioral Patterns): Tập trung vào việc quản lý các thuật toán, luồng điều khiển và sự tương tác giữa các đối tượng. Ví dụ: Observer, Strategy, Command.

* + 1. Lợi ích của mẫu thiết kế

- Tăng tính nhất quán và chuẩn hóa: Mẫu thiết kế cung cấp các giải pháp chuẩn hóa giúp cho mã nguồn dễ hiểu và dễ bảo trì hơn.

- Tái sử dụng giải pháp: Các giải pháp được cung cấp bởi mẫu thiết kế có thể được tái sử dụng trong nhiều dự án khác nhau, tiết kiệm thời gian và công sức phát triển.

- Giảm độ phức tạp: Mẫu thiết kế giúp giảm độ phức tạp của các vấn đề thiết kế, cung cấp các giải pháp đã được thử nghiệm và tối ưu.

- Tăng cường khả năng giao tiếp: Sử dụng mẫu thiết kế giúp các nhà phát triển có thể dễ dàng giao tiếp và hiểu lẫn nhau thông qua các thuật ngữ và giải pháp chung.

## **Mẫu thiết kế ECB**

* + 1. Định nghĩa ECB

Mẫu thiết kế ECB là một mô hình kiến trúc phần mềm giúp tách biệt các trách nhiệm trong một ứng dụng thành ba phần chính:

- Entity (Thực thể): Đại diện cho các đối tượng trong miền vấn đề (domain), chứa dữ liệu và các quy tắc nghiệp vụ liên quan. Ví dụ: Khách hàng, sản phẩm, đơn hàng trong một hệ thống quản lý bán hàng.

- Control (Điều khiển): Quản lý luồng điều khiển của ứng dụng, nhận yêu cầu từ Boundary, xử lý yêu cầu đó và trả kết quả về cho Boundary. Ví dụ: Lớp xử lý đăng nhập, lớp xử lý đơn hàng.

- Boundary (Biên giới): Đóng vai trò là giao diện giữa hệ thống và thế giới bên ngoài, có thể là giao diện người dùng (GUI), giao diện với các hệ thống khác. Ví dụ: Các form trong một ứng dụng Windows, các trang web trong một ứng dụng web.

* + 1. Lợi ích

- Tăng tính dễ hiểu: Mỗi lớp có một trách nhiệm rõ ràng, giúp code dễ đọc, dễ bảo trì và dễ hiểu hơn.

- Tăng tính tái sử dụng: Các lớp Entity có thể được sử dụng lại trong nhiều phần khác nhau của ứng dụng.

- Dễ dàng kiểm thử: Việc kiểm thử các lớp trở nên đơn giản hơn vì mỗi lớp có một chức năng cụ thể.

- Cải thiện khả năng mở rộng: Việc thêm các tính năng mới vào hệ thống trở nên dễ dàng hơn.

* + 1. Bất lợi

- Độ phức tạp ban đầu: Việc thiết kế và triển khai một hệ thống theo mẫu ECB có thể phức tạp hơn so với các cách tiếp cận khác, đặc biệt đối với các ứng dụng nhỏ.  
- Không phù hợp với mọi loại ứng dụng: Mẫu ECB không phù hợp với tất cả các loại ứng dụng, đặc biệt là các ứng dụng có yêu cầu về hiệu suất rất cao.

## **Mẫu thiết kế MVC**

* + 1. Định nghĩa MVC

Mẫu thiết kế MVC (Model-View-Controller) là một phương pháp phân chia hệ thống phần mềm thành ba phần chính:

- Model (Mô hình): Chứa dữ liệu và logic nghiệp vụ của ứng dụng. Model xử lý mọi dữ liệu liên quan đến thông tin nghiệp vụ và phản ánh trạng thái của ứng dụng.

- View (Hiển thị): Là giao diện người dùng. View hiển thị dữ liệu từ Model cho người dùng và gửi các yêu cầu của người dùng đến Controller.

- Controller (Điều khiển): Đóng vai trò trung gian giữa Model và View. Controller nhận các yêu cầu từ View, xử lý dữ liệu (thông qua Model nếu cần), và quyết định phản hồi nào sẽ được gửi lại cho View.

Đây là một kiến trúc phần mềm phổ biến, đặc biệt trong phát triển ứng dụng web.

A diagram of a model

Description automatically generated

Hình 1.3.1.1. Mẫu MVC

* + 1. Lợi ích

- Phân tách rõ ràng giữa các phần: MVC giúp tách biệt các phần khác nhau của ứng dụng, làm cho mã nguồn dễ hiểu và dễ bảo trì hơn.

- Dễ dàng phát triển và mở rộng: Việc tách rời Model, View và Controller cho phép các nhà phát triển làm việc trên các phần khác nhau của ứng dụng mà không ảnh hưởng đến các phần khác.

- Tái sử dụng mã nguồn: Mã nguồn có thể được tái sử dụng và chia sẻ giữa các phần khác nhau của ứng dụng hoặc thậm chí giữa các dự án khác nhau.

- Khả năng kiểm tra và bảo trì cao: Do sự phân tách rõ ràng giữa các phần, việc kiểm thử và bảo trì ứng dụng trở nên dễ dàng hơn, giúp phát hiện và sửa lỗi nhanh chóng.

* + 1. Bất lợi

- Độ phức tạp ban đầu: MVC có thể phức tạp đối với các dự án nhỏ hoặc đối với những người mới bắt đầu, do yêu cầu sự hiểu biết rõ ràng về cách thức hoạt động của từng thành phần.

- Tốn thời gian phát triển: Việc phân chia rõ ràng các phần có thể làm tăng thời gian phát triển ban đầu vì cần phải thiết lập và duy trì ba thành phần riêng biệt.

- Khả năng quá tải dữ liệu giữa các lớp: Nếu không được thiết kế đúng cách, các thành phần trong MVC có thể trở nên quá tải với việc truyền dữ liệu giữa Model, View và Controller.

- Yêu cầu sự phối hợp giữa các thành viên nhóm: MVC yêu cầu sự phối hợp chặt chẽ giữa các nhà phát triển làm việc trên các phần khác nhau, điều này có thể trở nên khó khăn trong các nhóm lớn hoặc các nhóm phân tán.

## **Mẫu thiết kế Observer**

* + 1. Định nghĩa Observer

Mẫu thiết kế Observer (Observer Design Pattern) là một mẫu thiết kế hành vi trong lập trình phần mềm. Nó cho phép một đối tượng (gọi là Subject) tự động thông báo cho các đối tượng khác (gọi là Observers) về những thay đổi trạng thái của mình.  
Mẫu thiết kế Observer hoạt động dựa trên các thành phần sau:

- Subject (Chủ đề/Đối tượng bị quan sát): Đây là đối tượng giữ trạng thái và có thể có nhiều Observer đăng ký để theo dõi các thay đổi của nó. Khi trạng thái của Subject thay đổi, nó sẽ thông báo cho tất cả các Observer đã đăng ký.

- Observer (Người quan sát): Đây là các đối tượng đăng ký với Subject để nhận thông báo khi trạng thái của Subject thay đổi. Mỗi Observer có một phương thức cập nhật để thực hiện hành động khi nhận được thông báo.

A screenshot of a computer screen

Description automatically generated

Hình 1.4.1.1. Mẫu Observer

* + 1. Lợi ích

- Tách biệt các mối quan tâm (Separation of Concerns): Mỗi thành phần có một nhiệm vụ rõ ràng, giúp code dễ đọc, dễ bảo trì và dễ mở rộng.

- Tái sử dụng code: Các View có thể được sử dụng lại với các Model khác nhau, và các Controller có thể xử lý các yêu cầu khác nhau đối với cùng một Model.

- Phát triển đồng thời: Các nhà phát triển có thể làm việc độc lập trên các thành phần khác nhau của ứng dụng.

- Dễ dàng kiểm thử: Mỗi thành phần có thể được kiểm thử một cách độc lập.

- Cải thiện hiệu suất: Bằng cách tách biệt logic nghiệp vụ và giao diện người dùng, MVC có thể giúp cải thiện hiệu suất của ứng dụng.

- Hỗ trợ nhiều nền tảng: MVC có thể được sử dụng để phát triển các ứng dụng web, ứng dụng di động và các ứng dụng desktop.

* + 1. Bất lợi

- Độ phức tạp: MVC có thể làm cho ứng dụng phức tạp hơn, đặc biệt đối với các ứng dụng nhỏ.

- Đường cong học tập: Để làm việc hiệu quả với MVC, các nhà phát triển cần phải hiểu rõ các nguyên tắc của nó.

- Khối lượng code: MVC thường yêu cầu viết nhiều code hơn so với các kiến trúc khác.

- Không phù hợp cho tất cả các loại ứng dụng: MVC có thể không phù hợp cho các ứng dụng đơn giản hoặc các ứng dụng có giao diện người dùng rất phức tạp.

## **Mẫu thiết kế Singleton**

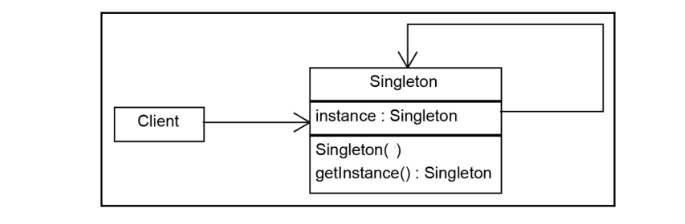
* + 1. Định nghĩa Singleton

Mẫu thiết kế Singleton (Singleton Design Pattern) là một mẫu thiết kế tạo dựng (creational design pattern) trong lập trình phần mềm. Mẫu này đảm bảo rằng một lớp chỉ có một thể hiện duy nhất (instance) và cung cấp một điểm truy cập toàn cục (global access point) đến thể hiện đó.

Mẫu thiết kế Singleton bao gồm các yếu tố chính sau:

- Duy nhất một thể hiện: Singleton đảm bảo rằng một lớp chỉ có một thể hiện duy nhất được tạo ra trong suốt thời gian chạy của ứng dụng.

- Điểm truy cập toàn cục: Singleton cung cấp một điểm truy cập duy nhất cho thể hiện của lớp, thường thông qua một phương thức tĩnh (static method).



Hình 1.5.1.1. Mẫu Singleton

* + 1. Lợi ích

- Quản lý tài nguyên: Singleton hữu ích trong các trường hợp cần quản lý tài nguyên hệ thống như kết nối cơ sở dữ liệu, tập tin cấu hình, hoặc thiết bị ngoại vi. Việc đảm bảo chỉ có một thể hiện giúp tránh xung đột và sử dụng tài nguyên hiệu quả.

- Kiểm soát truy cập: Singleton cung cấp một điểm truy cập toàn cục đến thể hiện của lớp, giúp dễ dàng kiểm soát và quản lý truy cập đến các tài nguyên hoặc dịch vụ mà lớp đó đại diện.

- Tiết kiệm bộ nhớ: Việc sử dụng một thể hiện duy nhất giúp tiết kiệm bộ nhớ, đặc biệt trong các ứng dụng mà chi phí khởi tạo đối tượng lớn.

- Dễ dàng tích hợp: Singleton giúp dễ dàng tích hợp với các phần khác của hệ thống mà không cần thay đổi nhiều mã nguồn.

* + 1. Bất lợi

- Khó kiểm thử: Singleton có thể làm khó khăn trong việc kiểm thử đơn vị (unit testing) vì sự phụ thuộc toàn cục và việc khởi tạo chỉ một lần. Điều này có thể làm phức tạp việc mô phỏng (mock) hoặc kiểm tra các tình huống khác nhau.

- Khả năng mở rộng hạn chế: Singleton có thể hạn chế khả năng mở rộng, vì nó ngăn cản việc tạo nhiều thể hiện của lớp. Điều này có thể gây khó khăn nếu sau này cần phải mở rộng hoặc thay đổi hành vi của lớp.

- Tạo ra sự phụ thuộc ngầm: Singleton có thể tạo ra các phụ thuộc ngầm trong mã nguồn, khiến cho việc duy trì và quản lý mã trở nên phức tạp, đặc biệt trong các hệ thống lớn.

- Vấn đề với đa luồng: Trong môi trường đa luồng, việc đảm bảo chỉ có một thể hiện Singleton cần phải được xử lý cẩn thận để tránh các vấn đề về đồng bộ hóa, điều này có thể dẫn đến lỗi khi nhiều luồng cố gắng truy cập và tạo thể hiện cùng một lúc.

## **Mẫu thiết kế CommandProcessor**

* + 1. Định nghĩa CommandProcessor

Mẫu thiết kế Command Processor (hay còn gọi là Command Pattern) là một mẫu thiết kế hành vi trong lập trình phần mềm. Mẫu này tách biệt việc phát sinh một yêu cầu từ thực thi của nó bằng cách đóng gói các yêu cầu thành các đối tượng, gọi là Commands (Lệnh). Điều này cho phép linh hoạt trong việc quản lý các yêu cầu, chẳng hạn như log các hoạt động, undo (hoàn tác) các thay đổi, hoặc tái thực hiện các yêu cầu.

Mẫu thiết kế Command Processor bao gồm các thành phần chính sau:

- Command (Lệnh): Đóng gói các chi tiết của một yêu cầu, bao gồm tất cả các thông tin cần thiết để thực thi lệnh đó. Mỗi Command thường có một phương thức execute() để thực hiện lệnh.

- Invoker (Bộ kích hoạt): Đối tượng này chịu trách nhiệm thực hiện các lệnh. Invoker không biết chi tiết về cách thức các lệnh được thực thi; nó chỉ cần gọi phương thức execute() của Command.

- Receiver (Bộ nhận): Đối tượng thực tế thực hiện hành động cần thiết khi một lệnh được thực hiện. Command sẽ gọi các phương thức của Receiver để hoàn thành công việc.

- Client (Khách hàng): Tạo ra các Command và thiết lập các liên kết giữa Invoker và các Command cụ thể.

A diagram of a computer program

Description automatically generated

Hình 1.6.1.1. Mẫu Command Processor

* + 1. Lợi ích

- Phân tách trách nhiệm: Command Processor giúp tách biệt logic phát sinh yêu cầu và logic thực thi yêu cầu, giúp mã nguồn dễ hiểu và dễ bảo trì hơn.

- Hỗ trợ undo/redo: Bằng cách lưu trữ các đối tượng Command, hệ thống có thể hỗ trợ các thao tác undo và redo một cách dễ dàng.

- Dễ dàng mở rộng: Dễ dàng thêm các Command mới mà không ảnh hưởng đến mã nguồn hiện tại, chỉ cần tạo lớp Command mới và kết nối với Invoker.

- Linh hoạt trong xử lý yêu cầu: Có thể lưu trữ, log, hoặc xếp hàng các Command để xử lý sau, giúp hệ thống linh hoạt trong việc xử lý các yêu cầu.

* + 1. Bất lợi

- Tăng độ phức tạp: Mẫu thiết kế này có thể làm tăng độ phức tạp của hệ thống do việc tạo ra nhiều lớp Command và quản lý chúng.

- Tăng số lượng lớp: Cần phải tạo ra một lớp Command cho mỗi hành động, điều này có thể làm tăng số lượng lớp trong hệ thống và gây khó khăn trong việc quản lý.

- Tiêu tốn bộ nhớ: Việc lưu trữ các đối tượng Command, đặc biệt khi hệ thống cần hỗ trợ các thao tác undo/redo, có thể tiêu tốn nhiều bộ nhớ.

- Khả năng thực thi: Việc truyền các đối tượng Command có thể làm chậm hệ thống nếu không được tối ưu hóa, đặc biệt là khi các Command chứa nhiều dữ liệu hoặc được xử lý một cách không hiệu quả.

## **Mẫu thiết kế Three Layer**

* + 1. Định nghĩa Three Layer

Mẫu thiết kế Three-Layer (hay còn gọi là Three-Tier Architecture) là một mẫu kiến trúc phần mềm phổ biến, đặc biệt trong phát triển ứng dụng doanh nghiệp và web. Mẫu này phân chia ứng dụng thành ba lớp chính: Presentation Layer (Lớp Giao diện), Domain Layer (Lớp Miền), và Persistence Layer (Lớp Lưu trữ). Mỗi lớp có nhiệm vụ và trách nhiệm riêng, giúp phân tách rõ ràng các phần khác nhau của ứng dụng.

- Mẫu thiết kế gồm 3 lớp:

+ Presentation Layer (Lớp Giao diện): Đây là lớp tương tác với người dùng cuối, bao gồm giao diện người dùng (UI) và các thành phần hiển thị. Lớp này chịu trách nhiệm hiển thị thông tin cho người dùng và thu thập đầu vào từ họ. Các công nghệ phổ biến cho lớp này bao gồm HTML, CSS, JavaScript, hoặc các framework UI như React, Angular.

+ Domain Layer (Lớp Miền): Lớp này chứa logic nghiệp vụ của ứng dụng. Nó xử lý các quy tắc, quy trình và dữ liệu nghiệp vụ. Domain Layer tập trung vào việc quản lý trạng thái của ứng dụng và logic nghiệp vụ mà không liên quan đến các chi tiết cụ thể về giao diện người dùng hay lưu trữ dữ liệu. Đây là nơi các mô hình nghiệp vụ và quy tắc được xác định và thực thi.

+ Persistence Layer (Lớp Lưu trữ): Lớp này chịu trách nhiệm quản lý việc lưu trữ và truy xuất dữ liệu. Nó tương tác với cơ sở dữ liệu hoặc các hệ thống lưu trữ khác để lưu trữ dữ liệu một cách bền vững. Các công nghệ và công cụ phổ biến cho lớp này bao gồm SQL, NoSQL, ORM (Object-Relational Mapping) như Entity Framework, Hibernate.

A diagram of a diagram

Description automatically generated

Hình 1.7.1.1. Mẫu Three Layer

* + 1. Lợi ích

- Phân tách rõ ràng các trách nhiệm: Ba lớp tách biệt giúp mã nguồn dễ hiểu và dễ bảo trì hơn. Mỗi lớp chỉ chịu trách nhiệm về một phần cụ thể của ứng dụng, giảm thiểu sự phụ thuộc giữa các phần.

- Tăng tính linh hoạt và khả năng mở rộng: Ứng dụng có thể dễ dàng mở rộng hoặc thay đổi từng lớp mà không ảnh hưởng đến các lớp khác. Ví dụ, thay đổi giao diện người dùng không cần thay đổi logic nghiệp vụ.

- Dễ dàng kiểm thử: Phân tách các lớp giúp dễ dàng kiểm thử từng phần riêng biệt, đặc biệt là khi kiểm thử đơn vị (unit testing).

- Tái sử dụng mã nguồn: Các thành phần trong Domain Layer và Persistence Layer có thể được tái sử dụng trong các ứng dụng khác nhau.

* + 1. Bất lợi

- Tăng độ phức tạp: Việc thiết kế và duy trì ba lớp tách biệt có thể làm tăng độ phức tạp của hệ thống, đặc biệt là trong các ứng dụng nhỏ hoặc đơn giản.

- Hiệu suất: Việc phân chia ứng dụng thành ba lớp có thể dẫn đến hiệu suất giảm, do các lớp phải tương tác với nhau thông qua các lớp trung gian.

- Chi phí phát triển ban đầu cao: Việc thiết lập một kiến trúc ba lớp yêu cầu thời gian và công sức đáng kể để xây dựng cơ sở hạ tầng và định nghĩa các giao diện giữa các lớp.

- Khó khăn trong việc duy trì sự nhất quán: Đôi khi việc giữ cho các lớp đồng bộ và nhất quán có thể là thách thức, đặc biệt trong các nhóm phát triển lớn.

# **CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ CHƯƠNG TRÌNH**

## **2.1. Phân tích chương trình**

2.1.1. Mô tả đề tài

Để hiểu rõ hơn về phạm vi và cấu trúc của chương trình quản lý giao dịch nhà đất, nhóm đã phân tích chi tiết dựa trên mô tả của đề tài và chia chương trình thành hai phần chính như sau:

Giao dịch đất:

- Thông tin cần quản lý: Mã giao dịch, ngày giao dịch (ngày, tháng, năm), đơn giá, loại đất (A, B, C), diện tích

- Tính toán thành tiền:

+Đất loại B, C: Thành tiền = Diện tích x Đơn giá

+ Đất loại A: Thành tiền = Diện tích x Đơn giá x 1.5

Giao dịch nhà:

- Thông tin cần quản lý: Mã giao dịch, Ngày giao dịch (ngày, tháng, năm), Đơn giá, Loại nhà (cao cấp, thường), Địa chỉ, Diện tích

- Tính toán thành tiền:

+ Nhà cao cấp: Thành tiền = Diện tích x Đơn giá

+ Nhà loại thường: Thành tiền = Diện tích x Đơn giá x 90%

2.1.2. Yêu cầu đề tài

Để đảm bảo chương trình quản lý giao dịch nhà đất hoạt động hiệu quả và đáp ứng đầy đủ các nhu cầu của người dùng, đề tài đặt ra các yêu cầu cụ thể sau:

- Thêm, xóa, sửa thông tin giao dịch:

+ Thêm: Cho phép thêm giao dịch mới vào danh sách.

+ Xóa: Xóa giao dịch đã tồn tại khỏi danh sách.

+ Sửa: Cập nhật thông tin giao dịch đã tồn tại.

- Tìm kiếm giao dịch: Cho phép tìm kiếm thông tin giao dịch dựa trên các tiêu chí như mã giao dịch, ngày giao dịch, loại đất/nhà, v.v.

- Tính tổng số lượng giao dịch cho từng loại: Tính tổng số lượng giao dịch đất và giao dịch nhà, hoặc theo từng loại đất (A, B, C) và loại nhà (cao cấp, thường).

- Tính trung bình thành tiền của giao dịch: Tính giá trị trung bình của thành tiền từ các giao dịch đã lưu trữ, có thể phân loại theo loại đất/nhà hoặc theo các tiêu chí khác.

- Xuất ra các giao dịch của tháng nào đó: Lọc và xuất danh sách giao dịch diễn ra trong một tháng cụ thể.

## **2.2. Biểu đồ Use Case tổng quan**

A diagram of a person with lines and dots

Description automatically generated

Hình 2.2.1. Biểu đồ Use Case tổng quan

## **2.3. Đặc tả Use Case**

2.3.1. Thêm giao dịch

|  |  |  |
| --- | --- | --- |
| UCID | UC01 | |
| Name | Thêm giao dịch | |
| Summary | Cho phép người dùng tạo một bản ghi giao dịch nhà đất mới trong hệ thống | |
| Postconditions | Một bản ghi giao dịch mới được tạo và lưu trữ trong cơ sở dữ liệu | |
| Primary Actor(s) | Người dùng có quyền thêm giao dịch | |
| Secondary Actor(s) | Hệ thống | |
| Trigger | Người dùng nhấn nút "Thêm" | |
| Main Scenario | Step 1 | Người dùng nhập mã giao dịch |
| Step 2 | Người dùng chọn ngày giao dịch |
| Step 3 | Người dùng nhập đơn giá |
| Step 4 | Người dùng chọn loại bất động sản (đất hoặc nhà) |
| Step 5 | Người dùng nhập địa chỉ |
| Step 6 | Người dùng nhập diện tích |
| Step 7 | Người dùng nhấn nút "Thêm" |
| Step 8 | Hệ thống kiểm tra tính hợp lệ của dữ liệu nhập vào |
| Step 9 | Nếu dữ liệu hợp lệ, hệ thống lưu giao dịch vào cơ sở dữ liệu |
| Step 10 | Hệ thống hiển thị thông báo giao dịch đã được thêm thành công |
| Extensions | Step 9a | Nếu dữ liệu không hợp lệ, hệ thống hiển thị thông báo lỗi và yêu cầu người dùng nhập lại |

Bảng 2.3.1.1. Thêm giao dịch

2.3.2. Sửa giao dịch

|  |  |  |
| --- | --- | --- |
| UCID | UC02 | |
| Name | Sửa giao dịch | |
| Summary | Cho phép người dùng cập nhật thông tin của một bản ghi giao dịch nhà đất đã tồn tại trong hệ thống | |
| Postconditions | Thông tin của bản ghi giao dịch được cập nhật và lưu trữ trong cơ sở dữ liệu | |
| Primary Actor(s) | Người dùng có quyền sửa giao dịch | |
| Secondary Actor(s) | Hệ thống | |
| Trigger | Người dùng nhấn nút "Sửa" | |
| Main Scenario | Step 1 | Người dùng chọn giao dịch muốn sửa hoặc nhập mã giao dịch để tìm kiếm |
| Step 2 | Hệ thống hiển thị chi tiết thông tin của giao dịch được chọn |
| Step 3 | Người dùng chỉnh sửa các trường thông tin cần thay đổi |
| Step 4 | Người dùng nhấn nút "Sửa" |
| Step 5 | Hệ thống kiểm tra tính hợp lệ của dữ liệu được sửa |
| Step 6 | Nếu dữ liệu hợp lệ, hệ thống cập nhật thông tin giao dịch trong cơ sở dữ liệu |
| Step 7 | Hệ thống hiển thị thông báo giao dịch đã được sửa thành công |

Bảng 2.3.2.1. Sửa giao dịch

2.3.3. Xóa giao dịch

|  |  |  |
| --- | --- | --- |
| UCID | UC03 | |
| Name | Xóa giao dịch | |
| Summary | Cho phép người dùng xóa một bản ghi giao dịch nhà đất đã tồn tại khỏi hệ thống | |
| Preconditions | Người dùng đã đăng nhập vào hệ thống với quyền xóa giao dịch, giao dịch cần xóa phải tồn tại trong hệ thống | |
| Postconditions | Bản ghi giao dịch được chọn sẽ bị xóa khỏi danh sách giao dịch và cơ sở dữ liệu | |
| Primary Actor(s) | Người dùng có quyền xóa giao dịch | |
| Secondary Actor(s) | Hệ thống | |
| Trigger | Người dùng nhấn nút "Xóa" | |
| Main Scenario | Step 1 | Người dùng chọn giao dịch muốn xóa |
| Step 2 | Người dùng nhấn nút "Xóa" |
| Step 3 | Hệ thống hiển thị thông báo xác nhận việc xóa |
| Step 4 | Nếu người dùng xác nhận, hệ thống xóa giao dịch khỏi cơ sở dữ liệu |
| Step 5 | Hệ thống cập nhật lại danh sách giao dịch |

Bảng 2.3.3.1. Xoá giao dịch

2.3.4. Tìm kiếm theo mã giao dịch

|  |  |  |
| --- | --- | --- |
| UCID | UC04 | |
| Name | Tìm kiếm giao dịch | |
| Summary | Cho phép người dùng tìm kiếm một bản ghi giao dịch nhà đất cụ thể dựa trên mã giao dịch | |
| Postconditions | Hệ thống hiển thị thông tin chi tiết của giao dịch tìm được (nếu có) | |
| Primary Actor(s) | Người dùng | |
| Secondary Actor(s) | Hệ thống | |
| Trigger | Người dùng nhập mã giao dịch và nhấn nút "Tìm" | |
| Main Scenario | Step 1 | Người dùng nhập mã giao dịch vào ô tìm kiếm |
| Step 2 | Người dùng nhấn nút "Tìm theo ID" |
| Step 3 | Hệ thống tìm kiếm giao dịch có mã giao dịch trùng khớp trong cơ sở dữ liệu |
| Step 4 | Nếu tìm thấy, hệ thống hiển thị chi tiết thông tin giao dịch |

Bảng 2.3.4.1. Tìm kiếm giao dịch

2.3.5. Tính tổng số lượng giao dịch loại nhà/đất

|  |  |  |
| --- | --- | --- |
| UCID | UC05 | |
| Name | Tính số lượng giao dịch từng loại nhà/đất | |
| Summary | Cho phép người dùng thống kê số lượng giao dịch cho từng loại nhà/đất đã được giao dịch | |
| Postconditions | Hệ thống hiển thị kết quả thống kê số lượng giao dịch cho loại nhà được chọn | |
| Primary Actor(s) | Người dùng | |
| Secondary Actor(s) | Hệ thống | |
| Trigger | Người dùng chọn loại nhà và nhấn nút "Tổng loại nhà/đất" | |
| Main Scenario | Step 1 | Hệ thống hiển thị danh sách các loại nhà/đất có sẵn |
| Step 2 | Người dùng chọn một loại nhà/đất từ danh sách |
| Step 3 | Người dùng nhấn nút "Tổng loại nhà/đất" |
| Step 4 | Hệ thống truy vấn cơ sở dữ liệu để đếm số lượng giao dịch có loại nhà/đất trùng khớp |
| Step 5 | Hệ thống hiển thị kết quả thống kê |

Bảng 2.3.5.1. Tính tổng số lượng giao dịch loại nhà/đất

2.3.6. Tính trung bình thành tiền của giao dịch nhà/đất

|  |  |  |
| --- | --- | --- |
| UCID | UC06 | |
| Name | Tính số lượng giao dịch từng loại nhà/đất | |
| Summary | Cho phép người dùng thống kê số lượng giao dịch cho từng loại nhà/đất đã được giao dịch | |
| Postconditions | Hệ thống hiển thị kết quả thống kê số lượng giao dịch cho loại nhà/đất được chọn | |
| Primary Actor(s) | Người dùng | |
| Secondary Actor(s) | Hệ thống | |
| Trigger | Người dùng chọn loại đất và nhấn nút "Trung bình TT nhà/đất" | |
| Main Scenario | Step 1 | Hệ thống hiển thị danh sách các loại nhà/đất có sẵn |
| Step 2 | Người dùng nhấn nút "Trung bình TT nhà/đất" |
| Step 3 | Hệ thống tính trung bình thành tiền của nhà/đất có trong hệ thống |
| Step 5 | Hệ thống hiển thị kết quả thống kê |

Bảng 2.3.6.1. Tính trung bình thành tiền của giao dịch nhà/đất

2.3.7. Tính thành tiền của giao dịch

|  |  |  |
| --- | --- | --- |
| UCID | UC07 | |
| Name | Tính thành tiền của giao dịch | |
| Summary | Cho phép người dùng tính toán tổng giá trị của một giao dịch bất động sản | |
| Postconditions | Hệ thống hiển thị tổng giá trị của giao dịch | |
| Primary Actor(s) | Người dùng | |
| Secondary Actor(s) | Hệ thống | |
| Trigger | Người dùng nhấn nút "Thành tiền" | |
| Main Scenario | Step 1 | Người dùng chọn giao dịch cần tính tiền |
| Step 2 | Người dùng nhấn nút "Thành tiền" |
| Step 3 | Hệ thống lấy thông tin diện tích, đơn giá, loại bất động sản của giao dịch từ cơ sở dữ liệu |
| Step 4 | Hệ thống thực hiện phép tính |
| Step 5 | Hệ thống hiển thị tổng giá trị của giao dịch |

Bảng 2.3.7.1. Tính thành tiền của giao dịch

2.3.8. Xuất ra các giao dịch của tháng

|  |  |  |
| --- | --- | --- |
| UCID | UC08 | |
| Name | Xuất ra các giao dịch của tháng | |
| Summary | Cho phép người dùng xem danh sách tất cả các giao dịch diễn ra trong một tháng cụ thể | |
| Preconditions | Người dùng đã đăng nhập vào hệ thống | |
| Postconditions | Hệ thống hiển thị danh sách các giao dịch trong tháng được chọn | |
| Primary Actor(s) | Người dùng | |
| Secondary Actor(s) | Hệ thống | |
| Trigger | Người dùng chọn tháng và nhấn nút "Xuất giao dịch" | |
| Main Scenario | Step 1 | Hệ thống hiển thị một danh sách các tháng (ví dụ: tháng 1, tháng 2,...) |
| Step 2 | Người dùng chọn một tháng từ danh sách |
| Step 3 | Người dùng nhấn nút "Xuất giao dịch" |
| Step 4 | Hệ thống truy vấn cơ sở dữ liệu để lấy danh sách các giao dịch có ngày giao dịch thuộc tháng đã chọn |
| Step 5 | Hệ thống hiển thị danh sách các giao dịch tìm được (bao gồm các thông tin như: mã giao dịch, ngày giao dịch, loại bất động sản,...) |

Bảng 2.3.8.1. Xuất ra các giao dịch của tháng

## **2.4. Sơ đồ thiết kế**

2.4.1. Tầng Presentation

A screenshot of a computer

Description automatically generated

Hình 2.4.1.1. Tầng Presentation

2.4.2. Tầng Domain

A screenshot of a computer screen

Description automatically generated

Hình 2.4.1.2. Tầng Domain

2.4.3. Tầng Persistence

A screenshot of a computer

Description automatically generated

Hình 2.4.1.2. Tầng Persistence

## **2.5. Mối quan hệ giữa các tầng**

2.5.1. Tầng Presentation

#### 2.5.1.1. Các lớp trong tầng

- Lớp TransactionUI: Phụ trách hiển thị giao diện người dùng.

- Lớp CommandProcessor: Phụ trách nhiệm vụ thực thi Command đã được đóng gói.

- Lớp trừu tượng Command: Phụ trách đóng gói một hành động.

#### 2.5.1.2. Mối quan hệ giữa các lớp

- Lớp TransactionUI có mối quan hệ Composition với các lớp JMenuBar, DefaultTableModel, JTable, JButton, JTextField, JLabel, JComboBox, JXDatePicker.

- Lớp TransactionUI kế thừa (Extends) lớp JFrame và kế thừa (Implements) ActionListener, ListSelectionListener.

- Lớp TransactionUI có mối quan hệ Association với lớp CommandProcessor.

- Lớp CommandProcessor có mối quan hệ Dependency với lớp Command.

- Các lớp AddCommand, DeleteCommand, UpdateCommand, FindByIDCommand, AvgHouseCommand, AvgLandCommand, ExportCommand… kế thừa lớp trừu tượng Command.

2.5.2. Tầng Domain

#### 2.5.2.1. Các lớp trong tầng

- Interface TransactionService: Lớp khuôn mẫu dịch vụ

- Lớp TransactionServiceImpl: Chịu trách nhiệm xử lý các quy tắc và logic nghiệp vụ của ứng dụng.

- Lớp Publisher: Lớp tạo ra các thông điệp. Khi trạng thái thay đổi hoặc có sự kiện sẽ gửi thông điệp tới các Subscriber đã đăng ký.

- Interface Subscriber: Lớp khuôn mẫu tiếp nhận thông điệp từ lớp Publisher

- Lớp trừu tượng Transaction: Model mặc định của giao dịch.

- Lớp House: Model của giao dịch Nhà.

- Lớp Land: Model của giao dịch Đất.

#### 2.5.2.2. Mối quan hệ giữa các lớp

- Lớp TransactionServiceImpl kế thừa (Implements) với lớp TransactionService.

- Lớp TransactionService có mối quan hệ Dependency với lớp trừu tượng Transaction.

- Lớp House, Land kế thừa (Extends) lớp trừu tượng Transaction.

- Lớp TransactionServiceImpl kế thừa (Extends) lớp Publisher.

- Lớp Publisher có mối quan hệ Aggregation với lớp Subsciber.

2.4.3. Tầng Persistence

#### 2.5.3.1. Các lớp trong tầng

- Interface TransactionPersistenceService: Lớp khuôn mẫu về các nhiệm vụ liên quan tới việc thao tác đến dữ liệu Database.

- Lớp TransactionPersistenceServiceImpl: Đảm nhiệm việc gửi các yêu cầu đến lớp Database để thực hiện các thao tác liên quan đến dữ liệu

#### 2.5.3.2. Mối quan hệ giữa các lớp

- Lớp TransactionPersistenceServiceImpl kế thừa (Implements) interface TransactionPersistenceService.

- Lớp TransactionPersistenceServiceImpl có mối quan hệ Composition với lớp Connection.

2.5.4. Mối quan hệ giữa các tầng

- Lớp TransactionUI có mối quan hệ Association với TransactionService.

- Lớp TransactionUI kế thừa (Implements) với lớp Subscriber.

- Lớp Command có mối quan hệ Association với TransactionService.

- Lớp TransactionServiceImpl có mối quan hệ Association TransactionPersistenceService.

# **CHƯƠNG 3. CÀI ĐẶT THỰC NGHIỆM VÀ KẾT QUẢ CHƯƠNG TRÌNH**

## **3.1. Cài đặt chương trình**

3.1.1. Phiên bản sử dụng

**+ Windows 11 22H2 (x64)  
+ MySQL Community 8.0.38**

**+ OpenJDK 21.0.2**

**+ Visual Studio Code 1.89.1**

3.1.2. Tạo bảng transaction trong cơ sở dữ liệu

|  |  |
| --- | --- |
| **Bảng transaction** | |
| **Tên** | **Kiểu** |
| maGiaoDich (Khóa chính) | int(11) |
| ngayGiaoDich | date |
| donGia | int(11) |
| loaiDat | varchar(255) |
| loaiNha | varchar(255) |
| diaChi | varchar(255) |
| dienTich | double |

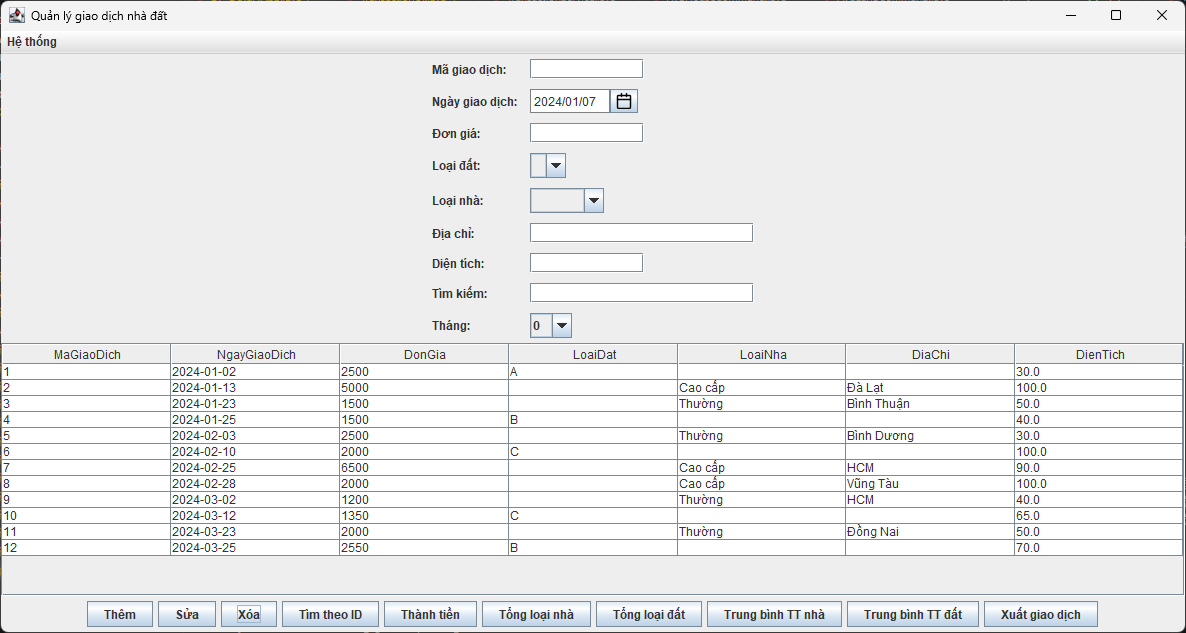
**Bảng 3.1.2. Bảng transaction**

3.1.3. Thêm dữ liệu vào bảng transaction trong cơ sở dữ liệu

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Bảng transaction** | | | | | | |
| **maGiaoDich** | **ngayGiaoDich** | **donGia** | **loaiDat** | **loaiNha** | **diaChi** | **dienTich** |
| 1 | 2024/01/02 | 2500 | A |  |  | 30 |
| 2 | 2024/01/13 | 5000 |  | Cao cấp | Đà Lạt | 100 |
| 3 | 2024/01/23 | 1500 |  | Thường | Bình Thuận | 50 |
| 4 | 2024/01/25 | 1500 | B |  |  | 40 |
| 5 | 2024/02/03 | 2500 |  | Thường | Bình Dương | 30 |
| 6 | 2024/02/10 | 2000 | C |  |  | 100 |
| 7 | 2024/02/25 | 6500 |  | Cao cấp | HCM | 90 |
| 8 | 2024/02/28 | 2000 |  | Cao cấp | Vũng Tàu | 100 |
| 9 | 2024/03/02 | 1200 |  | Thường | HCM | 40 |
| 10 | 2024/03/12 | 1350 | C |  |  | 65 |
| 11 | 2024/03/23 | 2000 |  | Thường | Đồng Nai | 50 |
| 12 | 2024/03/25 | 2550 | B |  |  | 70 |

Bảng 3.1.3.1. Dữ liệu

## **3.2. Kết quả chương trình**



Hình 3.2.1. Giao diện chính của chương trình

A screenshot of a computer

Description automatically generated

Hình 3.2.2. Thêm giao dịch mới với mã giao dịch là 13

**A screenshot of a computer

Description automatically generated**

Hình 3.2.3. Sửa đơn giá của mã giao dịch 13 từ 3000 thành 2500

**A screenshot of a computer

Description automatically generated**

Hình 3.2.4. Đã xóa mã giao dịch 13

*A screenshot of a computer

Description automatically generated*

Hình 3.2.5. Tìm mã giao dịch 11

*A screenshot of a computer

Description automatically generated*

Hình 3.2.6. Tính thành tiền mã giao dịch 11

*A screenshot of a computer

Description automatically generated*

Hình 3.2.7. Tính tổng số lượng loại nhà thường

**A screenshot of a computer

Description automatically generated**

Hình 3.2.8. Tính tổng số lượng loại đất A

*A screenshot of a computer

Description automatically generated*

Hình 3.2.9. Tính trung bình thành tiền nhà (Toàn bộ các tháng)

*A screenshot of a computer

Description automatically generated*

Hình 3.2.10. Tính trung bình thành tiền đất (Toàn bộ các tháng)

*A screenshot of a computer

Description automatically generated*

Hình 3.2.11. Xuất các giao dịch tháng 1

**A screenshot of a computer

Description automatically generated**

Hình 3.2.12. Tính trung bình thành tiền nhà (Tháng 1)

**A screenshot of a computer

Description automatically generated**

Hình 3.2.13. Tính trung bình thành tiền đất (Tháng 1)

# **CHƯƠNG 4. KẾT LUẬN VÀ ĐÁNH GIÁ**

## **4.1. Kết quả**

- Đã làm được:

+ Sau thời gian thực hiện đồ án, nhóm chúng em đã hiểu sâu hơn về môn học này và đạt được nhiều tiến bộ trong việc tìm hiểu, nghiên cứu lý thuyết cũng như nâng cao khả năng lập trình.

+ Chúng em đã tự tìm hiểu và học được nhiều kiến thức về MySQL, sử dụng thành thạo các công cụ hổ trợ lập trình như Visual Studio Code,…

+ Nhóm đã phân tích và thiết kế dự án một cách rõ ràng và theo trình tự nhất định.

+ Nhóm đã hiểu hơn về các mối quan hệ, các mẫu thiết kế.

- Chưa làm được:

+ Tính tổng số nhà/đất theo từng tháng.

+ Chưa có tính năng đăng nhập / phân quyền quản lý.

+ Chương trình còn gặp nhiều lỗi.

## **4.2. Kiến nghị phát triển trong tương lai**

- Tiếp tục phát triển và bổ sung những chức năng còn thiếu để chương trình đạt được sự hoàn chỉnh tối ưu.

- Tiến hành xử lý và khắc phục triệt để những lỗi hiện có trong chương trình để đảm bảo hoạt động mượt mà và ổn định.

- Tìm hiểu và mở rộng các tính năng của chương trình để có thể giải quyết và xử lý nhiều vấn đề phức tạp hơn, đáp ứng nhu cầu đa dạng của người dùng.

- Thiết kế lại chương trình và giao diện, làm cho nó trở nên đẹp mắt, dễ dàng sử dụng và thân thiện hơn với người dùng, đồng thời nâng cao tính thực tế và hiệu quả trong ứng dụng.

# **TÀI LIỆU THAM KHẢO**

1. NIIT HA NOI. Năm tài liệu được tạo ra 30/09/2019. *Mẫu thiết kế MVC*. Truy cập ngày 27/07/2024.
2. GP CODER. Năm tài liệu được tạo ra 28/12/2018. *Mẫu thiết kế Observer*. Truy cập ngày 27/07/2024.
3. CHƯƠNG. Năm tài liệu được tạo ra 28/09/2017. *Mẫu thiết kế Singleton*. Truy cập ngày 27/07/2024.
4. HOÀNG ĐINH. Năm tài liệu được tạo ra 21/11/2021. *Mẫu thiết kế CommandProcessor*. Truy cập ngày 27/07/2024.
5. LÊ NGỌC SƠN. Năm tài liệu được tạo ra 26/07/2020. *Mẫu thiết kế Three Layer.* Truy cập ngày 27/07/2024.