TRƯỜNG ĐẠI HỌC ĐÀ LẠT

**KHOA CÔNG NGHỆ THÔNG TIN**

**--------**

**BÁO CÁO ĐỀ TÀI**

**Xây dựng ứng dụng quản lý đặt phòng cơ bản**

***Giáo viên hướng dẫn*:** Đoàn Minh Khuê

***Lớp*:** CTK45B

***Sinh viên thực hiện*:** 2115234 Nguyễn Thành Long

***Đà Lạt - 10/2025***

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

Đà Lạt, ngày … tháng … năm

Giáo viên hướng dẫn

(Ký tên và ghi rõ họ tên)

**LỜI CẢM ƠN**

Với lòng kính trọng và biết ơn sâu sắc, em xin chân thành cám ơn sâu sắc đến giáo  
viên hướng dẫn thầy Đoàn Minh Khuê, người đã tận tình giúp em hiểu rõ hơn về  
đề tài này, giải đáp những thắc mắc và chỉnh sửa bài báo cáo của em. Nhờ có sự hỗ trợ nhiệt tình của thầy mà bài báo cáo mới đạt được hiệu quả nhất. Một lần nữa cám ơn thầy rất nhiều.

Cuối cùng, em xin gửi lời thân thương đến gia đình và bạn bè những người luôn  
ủng hộ và động viên với những lúc em muốn bỏ cuộc. Nhất là những người bạn trong  
lớp, những người luôn luôn chia sẻ kiến thức và kinh nghiệm hữu ích trong cuộc đời sinh viên.

Dù trải qua biết bao cố gắng nhưng báo cáo đề tài này vẫn còn tồn tại những thiếu sót và hạn chế. Rất mong nhận được sự đóng góp ý kiến nhiệt tình từ thầy cô, bạn bè và những người có mặt tại buổi báo cáo này, để chung tay hoàn thiện hơn.

Em xin trân trọng cám ơn!

Đà Lạt, ngày 29 tháng 10 năm 2025

**MỤC LỤC**

[**LỜI MỞ ĐẦU** 6](#_Toc212289395)

[**CHƯƠNG 1: GIỚI THIỆU** 7](#_Toc212289396)

[1.1. Mục tiêu đề tài 7](#_Toc212289397)

[1.2. Phạm vi của dự án 7](#_Toc212289398)

[**CHƯƠNG 2: NỘI DUNG** 9](#_Toc212289399)

[2.1. Phân tích và Thiết kế hệ thống 9](#_Toc212289400)

[2.1.1. Yêu cầu chức năng 9](#_Toc212289401)

[2.1.2. Lựa chọn công nghệ 10](#_Toc212289402)

[2.2. Các Mẫu thiết kế được áp dụng 10](#_Toc212289403)

[2.2.1. Singleton Pattern 10](#_Toc212289404)

[2.2.2. Decorator Pattern 11](#_Toc212289405)

[2.2.3. Strategy Pattern 13](#_Toc212289406)

[2.2.4. Factory Pattern 15](#_Toc212289407)

[2.2.5. Observer Pattern 16](#_Toc212289408)

[2.3. Kiến trúc Model-View-Controller (MVC) 18](#_Toc212289409)

[2.3.1. Giới thiệu về MVC 18](#_Toc212289410)

[2.3.2. Áp dụng MVC trong dự án 19](#_Toc212289411)

[2.4. Sơ đồ UML 20](#_Toc212289412)

[2.4.1. Sơ đồ Use-Case 20](#_Toc212289413)

[2.4.2. Sơ đồ lớp 20](#_Toc212289414)

[**CHƯƠNG 3: KẾT QUẢ ĐẠT ĐƯỢC** 21](#_Toc212289415)

[3.1. Giao diện xác thực đăng nhập và đăng ký 21](#_Toc212289416)

[3.2. Giao diện Chính 22](#_Toc212289417)

[3.3. Giao diện đặt phòng 23](#_Toc212289418)

[3.4. Giao diện quản lý đặt phòng 24](#_Toc212289419)

[**CHƯƠNG 4: KẾT LUẬT** 25](#_Toc212289420)

[4.1. Kết quả đạt được 25](#_Toc212289421)

[4.2. Hướng phát triển 26](#_Toc212289422)

[**TÀI LIỆU THAM KHẢO** 27](#_Toc212289423)

# **LỜI MỞ ĐẦU**

Trong bối cảnh cuộc cách mạng công nghiệp 4.0, công nghệ thông tin đã trở thành động lực cốt lõi, thúc đẩy sự thay đổi trong mọi ngành nghề. Ngành dịch vụ khách sạn, với sự cạnh tranh ngày càng gay gắt, cũng không nằm ngoài xu thế này. Việc ứng dụng phần mềm vào quản lý không chỉ là một công cụ giúp tự động hóa các nghiệp vụ như đặt phòng, quản lý phòng, mà còn là một chiến lược quan trọng để nâng cao chất lượng dịch vụ và tối ưu hóa hiệu quả kinh doanh.

Tuy nhiên, việc xây dựng một phần mềm không chỉ dừng lại ở việc làm cho nó chạy được. Một thách thức lớn hơn đối với các nhà phát triển là làm thế nào để tạo ra một sản phẩm có cấu trúc vững chắc, dễ dàng bảo trì, có khả năng mở rộng trong tương lai và mang lại trải nghiệm người dùng tốt. Đây chính là lúc tầm quan trọng của kiến trúc và thiết kế phần mềm được thể hiện rõ rệt. Các mẫu thiết kế (Design Patterns) ra đời như những giải pháp đã được kiểm chứng, cung cấp những khuôn mẫu tối ưu để giải quyết các vấn đề thiết kế phổ biến trong lập trình hướng đối tượng. Dự án này tập trung vào việc triển khai kiến trúc Model-View-Controller (MVC) làm nền tảng, kết hợp với việc vận dụng sáng tạo năm mẫu thiết kế kinh điển. Cụ thể:

Singleton được dùng để đảm bảo tính nhất quán của kho dữ liệu.

Decorator được xây dựng nền tảng để cho phép mở rộng dịch vụ trong tương lai.

Strategy và Factory được kết hợp để xử lý linh hoạt các nghiệp vụ tính giá.

Observer được triển khai để tạo ra một trải nghiệm người dùng tương tác cao, tự động cập nhật.

Đề tài này là kết quả của quá trình nghiên cứu và nỗ lực vận dụng lý thuyết thiết kế phần mềm vào một sản phẩm thực tế, có khả năng bảo trì và mở rộng.

# **CHƯƠNG 1: GIỚI THIỆU**

## Mục tiêu đề tài

* Xây dựng ứng dụng hoàn chỉnh: Phát triển một phần mềm chạy trên máy tính cá nhân với giao diện đồ họa (GUI) thân thiện, cho phép thực hiện các nghiệp vụ quản lý đặt phòng cơ bản.
* Áp dụng kiến trúc Model-View-Controller: Tổ chức mã nguồn của dự án theo mô hình MVC một cách rõ ràng, giúp tách biệt giữa logic nghiệp vụ, dữ liệu và giao diện người dùng.
* Triển khai các Mẫu thiết kế: Vận dụng và phân tích năm mẫu thiết kế cụ thể là Singleton, Strategy, Factory, Observer, và Decorator vào các bối cảnh phù hợp để giải quyết các vấn đề.
* Lưu trữ dữ liệu đơn giản: Sử dụng file văn bản làm cơ chế lưu trữ dữ liệu thay vì cơ sở dữ liệu, nhằm mục đích đơn giản hóa việc triển khai và tập trung vào logic xử lý của ứng dụng.
* Xây dựng chức năng xác thực: Triển khai hệ thống đăng nhập, đăng ký và áp dụng mã hóa để bảo mật mật khẩu người dùng.

## Phạm vi của dự án

Các chức năng trong phạm vi:

* Xác thực người dùng:
* Chức năng Đăng nhập.
* Chức năng Đăng ký tài khoản mới.
* Mã hóa mật khẩu khi lưu và khi xác thực.
* Quản lý phòng:
* Quản lý thông tin phòng (đọc từ file cấu hình phong.txt).
* Nghiệp vụ Đặt phòng:
* Chức năng đặt phòng cho khách hàng mới và khách hàng đã tồn tại.
* Áp dụng linh hoạt các chiến lược tính giá (ngày thường, cuối tuần, ngày lễ).
* Quản lý Khách hàng:
* Quản lý thông tin khách hàng (lưu và truy xuất từ khachhang.txt).
* Quản lý Hiển thị:
* Hiển thị danh sách các lượt đặt phòng hiện có.
* Tự động làm mới (Observer) danh sách này ngay khi có lượt đặt mới hoặc trả phòng.
* Chức năng trả phòng (xóa một lượt đặt phòng khỏi datphong.txt).

Các chức năng ngoài phạm vi:

* Không quản lý phân quyền chi tiết.
* Không có các chức năng thanh toán, xuất hóa đơn, báo cáo, thống kê doanh thu.
* Không hỗ trợ hoạt động đa người dùng cùng lúc (client-server).
* Không tích hợp với hệ quản trị cơ sở dữ liệu.

# **CHƯƠNG 2: NỘI DUNG**

## Phân tích và Thiết kế hệ thống

### Yêu cầu chức năng

* Hệ thống phải yêu cầu người dùng Đăng nhập bằng tên đăng nhập và mật khẩu.
* Hệ thống phải cho phép người dùng Đăng ký tài khoản nếu chưa có.
* Hệ thống phải mã hóa mật khẩu bằng mã hóa trước khi lưu vào taikhoan.txt.
* Sau khi đăng nhập, hệ thống phải cho phép người dùng xem danh sách các phòng có sẵn (đọc từ phong.txt).
* Hệ thống phải cung cấp chức năng nhập thông tin khách hàng, bao gồm CMND, họ tên và số điện thoại.
* Hệ thống phải có khả năng kiểm tra xem một khách hàng (dựa trên CMND) đã tồn tại trong khachhang.txt hay chưa.
* Hệ thống phải cho phép người dùng thực hiện một lượt đặt phòng bằng cách chọn phòng, nhập thông tin khách hàng, số đêm ở và loại giá áp dụng (Ngày thường, Cuối tuần, Ngày lễ).
* Hệ thống phải tính toán tổng chi phí cho một lượt đặt phòng dựa trên giá gốc của phòng, số đêm và loại giá được chọn.
* Hệ thống phải lưu thông tin khách hàng mới và thông tin đặt phòng mới vào các file văn bản tương ứng (khachhang.txt, datphong.txt).
* Hệ thống phải hiển thị một danh sách tất cả các lượt đặt phòng đang có trong hệ thống trên một JTable.
* Danh sách đặt phòng phải tự động cập nhật ngay lập tức khi một lượt đặt phòng mới được thêm hoặc bị xóa mà không cần người dùng nhấn nút Làm mới.
* Hệ thống phải cho phép người dùng chọn một lượt đặt phòng từ danh sách và thực hiện thao tác "Trả phòng" (xóa bản ghi đó).

### Lựa chọn công nghệ

* Ngôn ngữ lập trình: Java. Java là một ngôn ngữ lập trình hướng đối tượng mạnh mẽ, đa nền tảng, phù hợp để xây dựng các ứng dụng desktop.
* Thư viện giao diện: Java Swing. Swing là một bộ công cụ có sẵn trong JDK để xây dựng giao diện đồ họa người dùng (GUI).
* Cơ chế lưu trữ: File văn bản (.txt). Để giữ cho dự án đơn giản, hệ thống sử dụng file text để lưu trữ dữ liệu.
* API Bảo mật: java.security.MessageDigest và java.util.Base64 để triển khai mã hóa.

## Các Mẫu thiết kế được áp dụng

### Singleton Pattern

* Lý thuyết: Đảm bảo một lớp chỉ có duy nhất một thực thể (instance) và cung cấp một điểm truy cập toàn cục đến nó.
* Vấn đề trong đề tài: Ứng dụng cần một nơi duy nhất để quản lý tất cả dữ liệu (danh sách phòng, khách hàng, đặt phòng, tài khoản) được đọc lên từ file. Nếu DatPhongController và QuanLyController mỗi cái tự tạo một đối tượng DataRepository riêng, dữ liệu sẽ bị mất đồng bộ.
* Áp dụng (DataRepository): Lớp service.repository.DataRepository được thiết kế theo Singleton. Nó có private constructor để ngăn cản việc tạo mới từ bên ngoài, và một phương thức public static getInstance() để mọi thành phần khác ngoài các Controller đều truy cập cùng một đối tượng dữ liệu duy nhất, đảm bảo tính nhất quán.
* Lợi ích: Đảm bảo tính toàn vẹn và đồng bộ dữ liệu trên toàn ứng dụng. DataRepository trở thành nguồn chân lý duy nhất cho dữ liệu.
* Mã nguồn minh họa:

// File: service/repository/DataRepository.java

public class DataRepository {

private static DataRepository instance;

private DataRepository() {

// ... khởi tạo các danh sách

}

public static DataRepository getInstance() {

if (instance == null) {

instance = new DataRepository();

}

return instance;

}

}

### Decorator Pattern

* Lý thuyết: Cho phép thêm các chức năng mới vào một đối tượng một cách linh hoạt mà không cần thay đổi lớp của đối tượng đó.
* Vấn đề trong đề tài: Đây là vấn đề được dự phòng cho tương lai nếu khách hàng muốn thêm các dịch vụ đi kèm (thêm giường, bữa sáng, spa...), làm thế nào để cộng dồn chi phí. Dùng kế thừa sẽ dẫn đến "bùng nổ lớp".
* Áp dụng (Thêm dịch vụ cho phòng): Dự án đã xây dựng khung sườn cho mẫu Decorator.
* Interface IPhongGia định nghĩa getGia() và getMoTa().
* Lớp PhongGiaCoBan là đối tượng gốc.
* Lớp trừu tượng PhongGiaDecorator "bọc" một đối tượng IPhongGia.
* Lợi ích (Cho tương lai): Cấu trúc này cho phép hệ thống dễ dàng mở rộng. Nếu trong tương lai muốn thêm các dịch vụ phụ trợ (ví dụ: ăn sáng, spa, giường phụ...), nhà phát triển chỉ cần tạo các lớp mới cụ thể kế thừa từ lớp PhongGiaDecorator đã có sẵn. Các lớp mới này sẽ bọc đối tượng PhongGiaCoBan và cộng dồn chi phí dịch vụ của mình vào, mà không cần sửa đổi code gốc của PhongGiaCoBan hay DatPhongController.
* Mã nguồn minh họa:

// File: service/phong/PhongGiaDecorator.java

public abstract class PhongGiaDecorator implements IPhongGia {

protected IPhongGia wrappee;

public PhongGiaDecorator(IPhongGia source) {

this.wrappee = source;

}

@Override

public double getGia() {

return wrappee.getGia();

}

@Override

public String getMoTa() {

return wrappee.getMoTa();

}

}

### Strategy Pattern

* Lý thuyết: Định nghĩa một họ các thuật toán , đóng gói chúng lại và làm cho chúng có thể hoán đổi cho nhau.
* Vấn đề trong đề tài: Giá phòng có thể được tính theo nhiều cách (ngày thường, cuối tuần, ngày lễ). Viết logic này bằng if-else hoặc switch-case trực tiếp trong DatPhongController sẽ làm code bị rối, khó bảo trì và vi phạm Nguyên tắc Đóng/Mở.
* Áp dụng (Tính giá phòng):
* Interface IGiaPhongStrategy định nghĩa phương thức chung tinhGia().
* Các lớp GiaNgayThuongStrategy, GiaCuoiTuanStrategy, GiaNgayLeStrategy implement interface này, mỗi lớp cung cấp một thuật toán tính giá riêng.
* DatPhongController sẽ chọn một Strategy phù hợp dựa trên JComboBox của View để tiêm vào đối tượng PhongGiaCoBan khi tính toán.
* Lợi ích: Hệ thống trở nên linh hoạt. Việc thêm một cách tính giá mới chỉ đơn giản là tạo một lớp Strategy mới mà không cần sửa code của Controller.
* Mã nguồn minh họa:

// File: service/phong/PhongGiaCoBan.java

public class PhongGiaCoBan implements IPhongGia {

protected IGiaPhongStrategy strategy;

protected Phong phong;

protected int soDem;

public PhongGiaCoBan(Phong phong, IGiaPhongStrategy strategy, int soDem) {

this.phong = phong;

this.strategy = strategy; // Chiến lược được "tiêm" vào

this.soDem = soDem;

}

@Override

public double getGia() {

return strategy.tinhGia(phong, soDem);

}

}

### Factory Pattern

* Lý thuyết: Cung cấp một interface để tạo các đối tượng, nhưng để các lớp con quyết định lớp nào sẽ được khởi tạo.
* Áp dụng (GiaPhongStrategyFactory): Sau khi áp dụng Strategy, DatPhongController vẫn phải chịu trách nhiệm tạo ra các đối tượng Strategy cụ thể. Để làm sạch Controller, chúng ta tạo ra một lớp là GiaPhongStrategyFactory. Lớp này có một phương thức tĩnh createStrategy chứa logic switch-case để quyết định new đối tượng Strategy nào.
* Lợi ích: Dọn dẹp code cho DatPhongController. Controller chỉ cần gọi GiaPhongStrategyFactory.createStrategy() mà không cần biết các lớp Strategy cụ thể là gì. Logic khởi tạo được đóng gói tập trung tại một nơi, dễ bảo trì.
* Mã nguồn minh họa:

// File: service/giaphong/GiaPhongStrategyFactory.java

public class GiaPhongStrategyFactory {

public static IGiaPhongStrategy createStrategy(String loaiGia) {

if (loaiGia == null) {

return new GiaNgayThuongStrategy();

}

switch (loaiGia) {

case "Cuối tuần":

return new GiaCuoiTuanStrategy();

case "Ngày lễ":

return new GiaNgayLeStrategy();

case "Ngày thường":

default:

return new GiaNgayThuongStrategy();

}

}

}

### Observer Pattern

* Lý thuyết: Định nghĩa một mối quan hệ một-nhiều. Khi một đối tượng Subject thay đổi, tất cả các đối tượng phụ thuộc Observers sẽ được thông báo và tự động cập nhật.
* Áp dụng (Tự động làm mới): Khi người dùng đặt phòng ở DatPhongPanel, JTable ở QuanLyDatPhongPanel cần được cập nhật.
* Subject: DataRepository (Singleton) trở thành Subject. Nó quản lý một List<RepositoryObserver> và có hàm notifyObservers().
* Observer: QuanLyDatPhongPanel implement interface RepositoryObserver và định nghĩa hàm update() là gọi loadDataToTable().
* Đăng ký: Trong MainFrame, ta gọi DataRepository.getInstance().addObserver(quanLyPanel); để bắt đầu lắng nghe.
* Kích hoạt: Khi DataRepository gọi themDatPhong() hoặc xoaDatPhong(), nó sẽ gọi notifyObservers(), kích hoạt hàm update() của QuanLyDatPhongPanel.
* Lợi ích: Tự động cập nhật giao diện. Giao diện phản hồi tức thì với các thay đổi dữ liệu, nâng cao trải nghiệm người dùng, không cần nhấn Làm mới.
* Mã nguồn minh họa:

// File: service/repository/DataRepository.java

public class DataRepository {

private final List<RepositoryObserver> observers = new ArrayList<>();

public void addObserver(RepositoryObserver observer) {

this.observers.add(observer);

}

private void notifyObservers() {

for (RepositoryObserver observer : observers) {

observer.update();

}

}

public void themDatPhong(DatPhong dp) {

dsDatPhong.add(dp);

luuDsDatPhong();

notifyObservers();

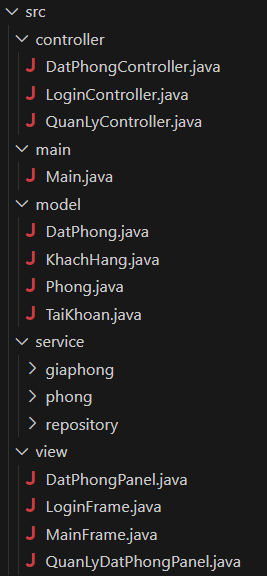
}

}

## Kiến trúc Model-View-Controller (MVC)

### Giới thiệu về MVC

* Model: Quản lý dữ liệu và logic nghiệp vụ. Gồm các lớp Phong, KhachHang, DatPhong, TaiKhoan và lớp quản lý DataRepository.
* View: Hiển thị dữ liệu và nhận tương tác từ người dùng. Gồm các lớp LoginFrame, MainFrame, DatPhongPanel, QuanLyDatPhongPanel.
* Controller: Trung gian xử lý sự kiện từ View và cập nhật Model. Gồm các lớp LoginController, DatPhongController và QuanLyController.

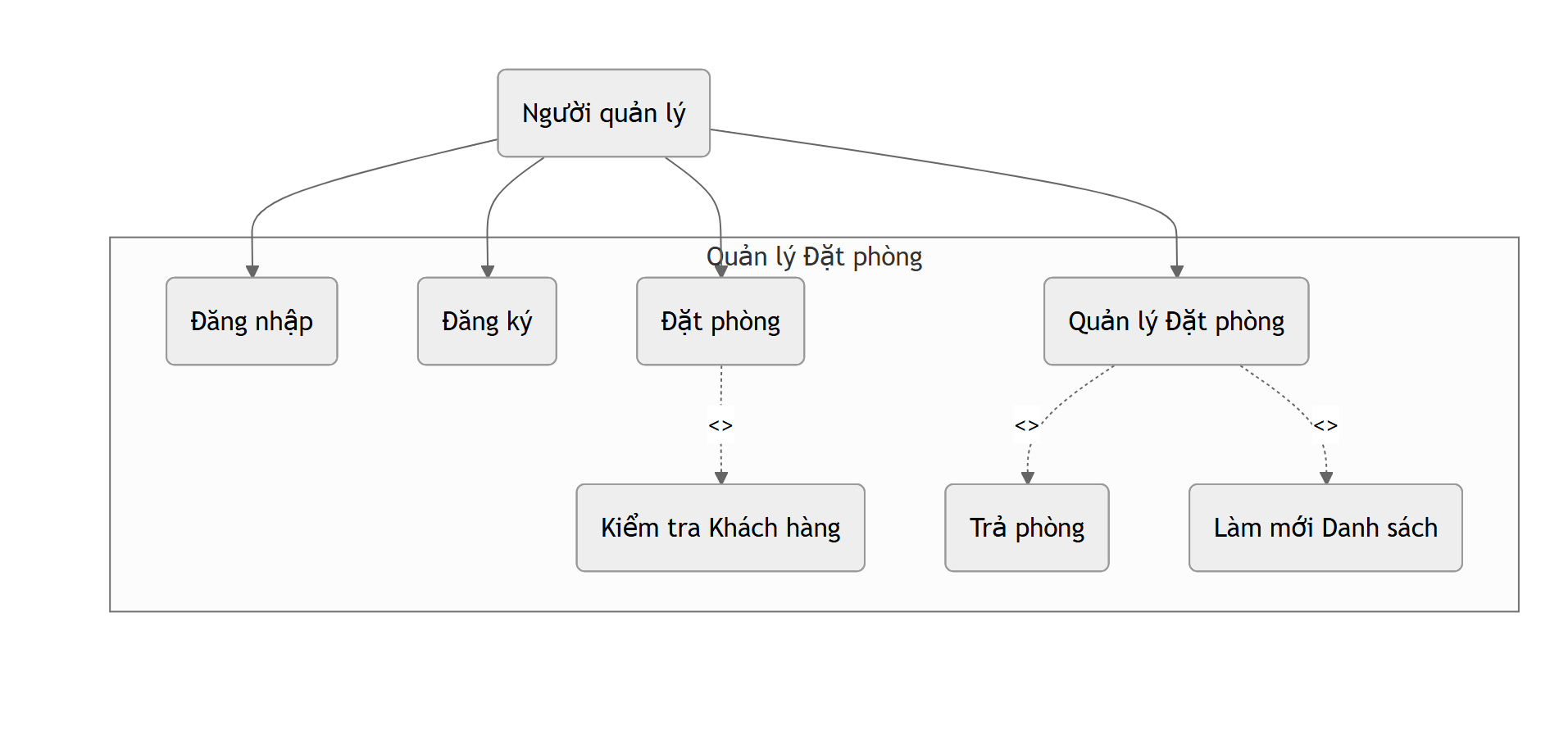


### Áp dụng MVC trong dự án

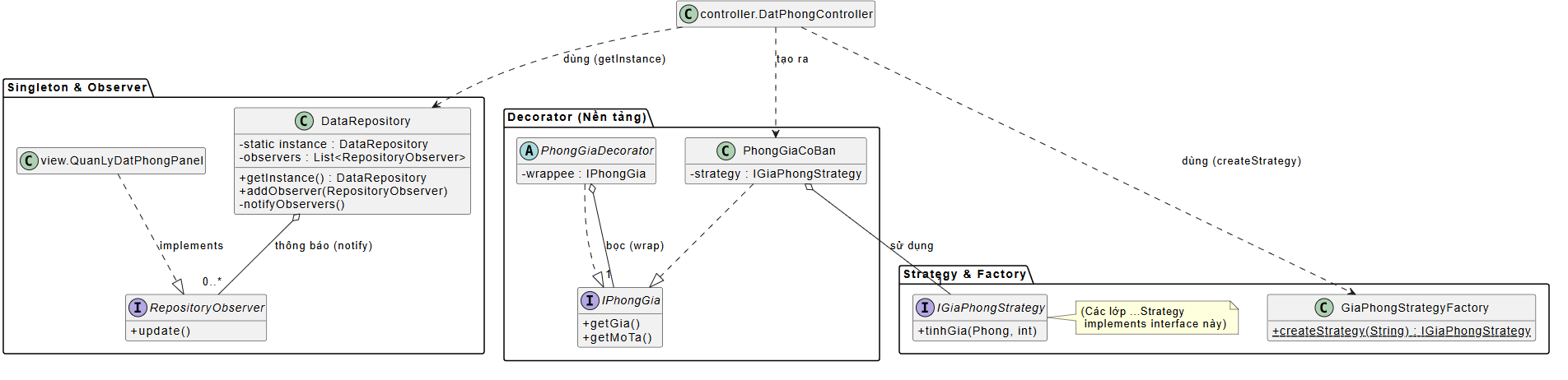
* Luồng Đăng nhập:
* Main.java khởi chạy LoginFrame và LoginController.
* Người dùng tương tác trên LoginFrame.
* LoginController nhận sự kiện (ví dụ: nhấn nút "Đăng nhập").
* Controller gọi DataRepository để kiểm tra dangNhap.
* Nếu thành công, LoginController đóng LoginFrame và mở MainFrame.
* Luồng Đặt phòng:
* Người dùng tương tác trên DatPhongPanel.
* Sự kiện được gửi đến DatPhongController.
* Controller xử lý logic (gọi Factory và Strategy để tính giá) và gọi DataRepository để themDatPhong().
* Luồng Cập nhật:
* Khi DataRepository hoàn tất themDatPhong() hoặc xoaDatPhong(), nó sẽ thông báo cho các người dùng (áp dụng Observer Pattern).
* QuanLyDatPhongPanel đã đăng ký làm người dùng, nó sẽ nhận được thông báo này.
* QuanLyDatPhongPanel tự động gọi hàm loadDataToTable() của chính nó để tải lại dữ liệu từ DataRepository và hiển thị lên JTable.

## Sơ đồ UML

### Sơ đồ Use-Case



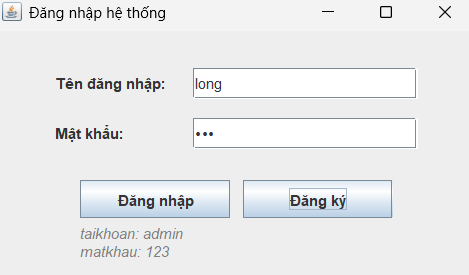
### Sơ đồ lớp



# **CHƯƠNG 3: KẾT QUẢ ĐẠT ĐƯỢC**

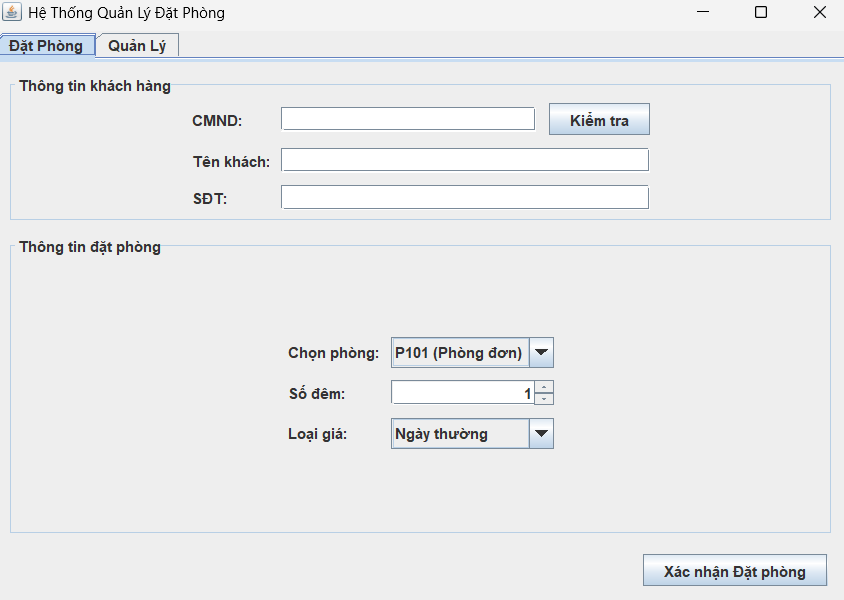
## Giao diện xác thực đăng nhập và đăng ký

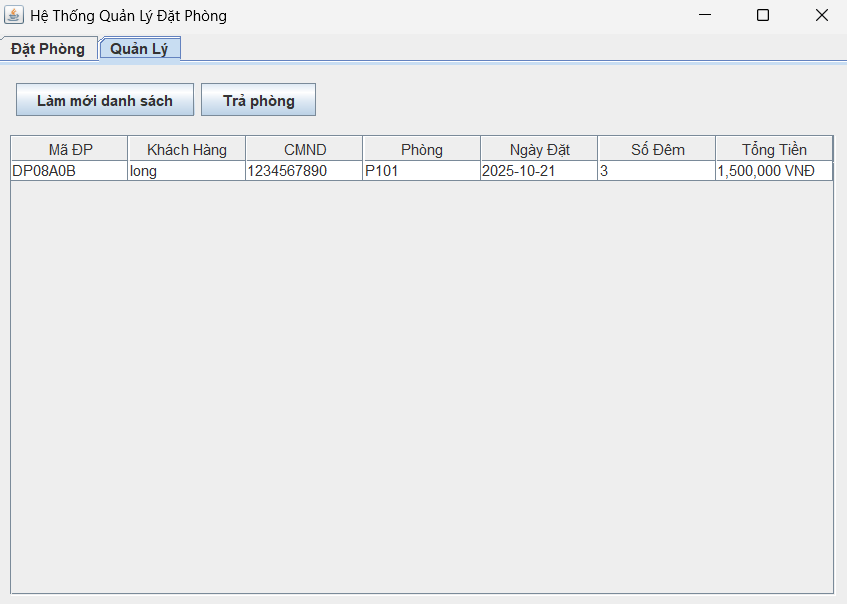
* Đây là giao diện đầu tiên khi khởi động ứng dụng. Giao diện được thiết kế đơn giản gồm tên đăng nhập và mật khẩu cùng hai nút đăng nhập và đăng ký. Khi người dùng nhấn nút, LoginController sẽ được kích hoạt để gọi DataRepository, kiểm tra thông tin đăng nhập ở trong taikhoan.txt hoặc thực hiện đăng ký tài khoản mới.



## Giao diện Chính

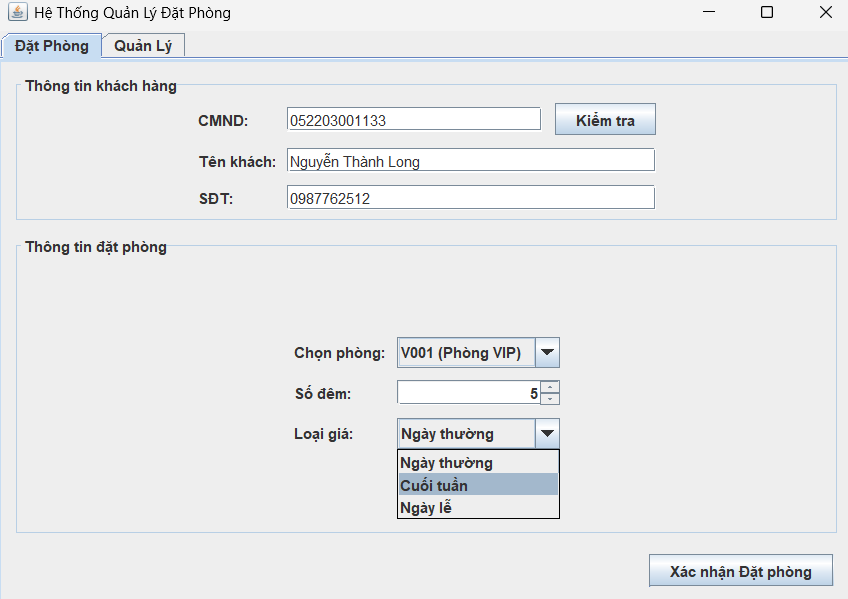
* Sau khi đăng nhập thành công, giao diện chính sẽ xuất hiện. Giao diện này sử dụng thanh tab để phân chia rõ ràng hai khu vực chức năng chính: đặt phòng và quản lý đặt phòng. Cấu trúc tab này giúp giao diện gọn gàng hơn khi mỗi tab là một View độc lập.



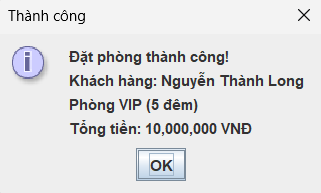


## Giao diện đặt phòng

* Đây là giao diện tại đặt phòng. Nó chứa đầy đủ các thành phần để nhập liệu cho một lượt đặt phòng đã lưu trong phong.txt, để đặt phòng cần cho thông tin khách, cho số đêm muốn ở. Đặc biệt, thông tin về loại giá là nơi áp dụng trực tiếp Strategy & Factory Pattern. Tùy chọn của người dùng (Cuối tuần, Ngày lễ...) sẽ quyết định Strategy tính giá nào được Factory tạo ra để tính tổng tiền.

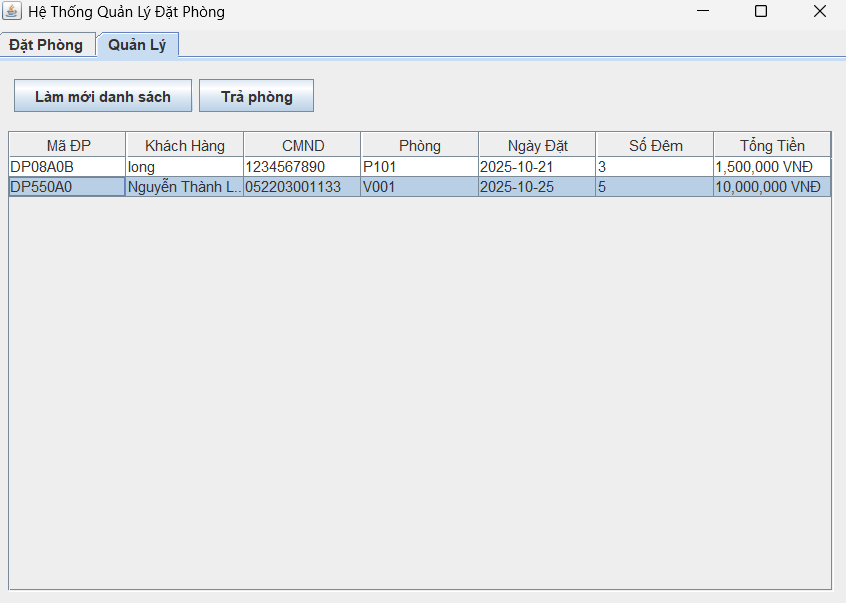


* Sau khi hoàn tất nhập thủ tục theo yêu cầu và nhấn đặt phòng thì sẽ xuất hiện lên bảng chi tiết về thông tin người đặt, loài hình đã đặt và giá tiền cần phải thanh toán. Tất cả sẽ được lưu trong khachhang.txt



## Giao diện quản lý đặt phòng

* Đây là giao diện quản lý. Thành phần trung tâm là JTable hiển thị toàn bộ dữ liệu từ datphong.txt. Giao diện này cũng là một Observer nhờ việc triển khai Observer Pattern, khi người dùng đặt phòng thành công ở giao diện đặt phòng thì ở giao diện quản lý đặt phòng sẽ tự động được làm mới và hiển thị dữ liệu mới ngay lập tức, trường hợp nếu ko thấy thông tin vừa đăng kí thì nhấn nút Làm mới danh sách. Khi muốn trả phòng thì nhấn nút Trả phòng, phòng vừa trả sẽ được xóa khỏi danh sách đươc lưu trong datphong.txt



# **CHƯƠNG 4: KẾT LUẬT**

## Kết quả đạt được

Đề tài đã xây dựng thành công một ứng dụng Quản lý Đặt phòng Khách sạn cơ bản bằng Java Swing, đáp ứng đầy đủ các yêu cầu chức năng đề ra, bao gồm cả chức năng xác thực người dùng (Đăng nhập/Đăng ký) và bảo mật mật khẩu.

Thành công lớn nhất của đề tài không chỉ nằm ở việc hoàn thiện các chức năng, mà ở việc áp dụng thành công năm mẫu thiết kế một cách có chủ đích:

* MVC giúp cấu trúc dự án rõ ràng, tách biệt logic nghiệp vụ, giao diện và dữ liệu.
* Singleton (DataRepository) đảm bảo tính toàn vẹn và nhất quán của dữ liệu trên toàn ứng dụng.
* Strategy (IGiaPhongStrategy) giúp logic tính giá trở nên linh hoạt và dễ mở rộng.
* Factory (GiaPhongStrategyFactory) giúp mã nguồn Controller sạch sẽ, tuân thủ đúng nguyên tắc thiết kế.
* Observer (DataRepository & QuanLyDatPhongPanel) tạo ra trải nghiệm người dùng hiện đại, tự động cập nhật, giảm thiểu thao tác thủ công.
* Decorator (Nền tảng PhongGiaDecorator) chuẩn bị sẵn sàng cho việc mở rộng các dịch vụ phụ trợ trong tương lai.

## Hướng phát triển

Dự án này đã đặt một nền móng vững chắc. Các hướng phát triển tiềm năng trong tương lai bao gồm:

* Hoàn thiện Decorator: Thêm giao diện JCheckBox cho các dịch vụ (ăn sáng, giường phụ...) và tạo các lớp Decorator cụ thể kế thừa từ PhongGiaDecorator để tính giá.
* Nâng cấp lưu trữ: Thay thế file .txt bằng một hệ quản trị cơ sở dữ liệu thực thụ (như SQLite, MySQL) để quản lý dữ liệu hiệu quả, an toàn và hỗ trợ truy vấn phức tạp hơn.
* Mở rộng nghiệp vụ: Thêm các chức năng nâng cao như quản lý hóa đơn, thống kê doanh thu, quản lý trạng thái phòng (trống, đang dọn, đã đặt) thay vì chỉ xóa khi trả phòng.

# **TÀI LIỆU THAM KHẢO**

Sách:

* Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. (Đây là cuốn sách "Gang of Four" (GoF) kinh điển định nghĩa về Mẫu Thiết Kế).
* Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
* Sierra, K., & Bates, B. (2005). *Head First Design Patterns*. O'Reilly Media.

Tài liệu trực tuyến:

* Oracle. *Java™ Platform, Standard Edition 8 API Specification*. Lấy từ: <https://docs.oracle.com/javase/8/docs/api/> truy cập gần nhất vào 18/10/2025
* Oracle. *Creating a GUI With JFC/Swing (The Java™ Tutorials)*. Lấy từ: <https://docs.oracle.com/javase/tutorial/uiswing/> truy cập gần nhất vào 18/10/2025
* Refactoring.Guru. *Design Patterns Catalog*. Lấy từ: <https://refactoring.guru/design-patterns> truy cập gần vào 20/10/2025
* Baeldung. *Design Patterns in Java*. Lấy từ: [https://www.baeldung.com/design-patterns-java](https://www.google.com/search?q=https://www.baeldung.com/design-patterns-java) truy cập gần nhất vào 21/10/2025
* TutorialsPoint. *Java - Design Patterns*. Lấy từ: <https://www.tutorialspoint.com/design_pattern/index.htm> truy cập gần nhất vào 23/10/2025