BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC ĐẠI NAM

BÁO CÁO HỌC PHẦN

TÊN HỌC PHẦN: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

ĐỀ TÀI: Mô tả bài toán ứng dụng "DNU BookShare"

Giáo viên hướng dẫn: Tạ Chí Hiếu

Sinh viên thực hiện:

STT	MÃ SINH VIÊN	HỌ VÀ TÊN	LỚP
1	1871020498	Nguyễn Xuân Sắc	CNTT 18-07
2	1871020497	Nguyễn Đình Quyết	CNTT 18-07
3	1871020294	Kim Bùi Quang Huy	CNTT 18-07

Hà Nội, năm 2025

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC ĐẠI NAM

BÀI TẬP LỚN

TÊN HỌC PHẦN: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

ĐỀ TÀI: Mô tả bài toán ứng dụng "DNU BookShare"

Mã sinh viên	Họ và tên	Lớp	Điểm chữ	Điểm số
1871020498	Nguyễn xuân sắc	CNTT 18-07		
1871020497	Nguyễn đình quyết	CNTT 18-07		
1871020294	Kim bùi quang huy	CNTT 18-07		

Hà Nội, năm 2025

LỜI NÓI ĐẦU

Trong thời đại công nghệ thông tin phát triển mạnh mẽ, lập trình hướng đối tượng (Object-Oriented Programming – OOP) đã trở thành một trong những phương pháp lập trình phổ biến và quan trọng nhất hiện nay. Phương pháp này giúp lập trình viên xây dựng các ứng dụng dễ bảo trì, dễ mở rộng, tái sử dụng mã nguồn và đặc biệt là mô phỏng được các đối tượng trong thế giới thực một cách tự nhiên.

Môn học Lập trình Hướng Đối Tượng không chỉ trang bị cho sinh viên những kiến thức cơ bản về cú pháp và cấu trúc của ngôn ngữ lập trình, mà còn giúp hình thành tư duy phân tích – thiết kế hệ thống phần mềm theo mô hình đối tượng. Thông qua môn học, sinh viên được thực hành các khái niệm quan trọng như đóng gói (Encapsulation), kế thừa (Inheritance), đa hình (Polymorphism) và trừu tượng hóa (Abstraction).

Trong khuôn khổ môn học, em và nhóm em lựa chọn thực hiện đề tài:
“DNU BookShare – Hệ thống Chia sẻ Sách Sinh viên”, với mục tiêu áp dụng những kiến thức OOP đã học để xây dựng một ứng dụng thực tế, hỗ trợ sinh viên trong việc chia sẻ, mượn và trao đổi sách. Đề tài không chỉ giúp em củng cố kiến thức lập trình mà còn mang ý nghĩa thiết thực trong môi trường học tập tại trường Đại học Đại Nam.

Mặc dù đã cố gắng hoàn thiện, nhưng do thời gian và kinh nghiệm còn hạn chế, báo cáo không thể tránh khỏi những thiếu sót. Em rất mong nhận được sự góp ý của quý thầy cô để bài làm được hoàn thiện hơn.

Em xin chân thành cảm ơn thầy Tạ Chí Hiếu dạy môn Lập trình Hướng Đối Tượng đã tận tình hướng dẫn và tạo điều kiện để em hoàn thành đề tài này.

MỤC LỤC

CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI

1.1 Giới thiệu chung về đề tài lựa chọn

DNU BookShare là một nền tảng trực tuyến dành cho cộng đồng sinh viên Đại học Đại Nam nhằm chia sẻ, cho mượn, trao đổi và mua bán sách (giáo trình, tài liệu tham khảo…). Ứng dụng giúp kết nối người có sách và người cần sách, tối ưu việc sử dụng tài nguyên tri thức, đồng thời thúc đẩy văn hóa đọc và tinh thần cộng đồng trong nhà trường.

1.2 Lý do chọn đề tài

Sinh viên thường thiếu sách trong một số học phần hoặc chỉ cần tài liệu trong thời gian ngắn.

Nhiều đầu sách sau học kỳ bị bỏ không/ít dùng, gây lãng phí.

Chưa có một kênh tập trung, minh bạch để tìm kiếm – liên hệ – theo dõi việc mượn/trả trong nội bộ trường.

=> Cần một ứng dụng giúp niêm yết sách nhanh, tìm kiếm thuận tiện, quản lý giao dịch rõ ràng.

1.3 Mục tiêu đề tài

Xây dựng hệ thống cho phép sinh viên đăng sách (tiêu đề, tác giả, môn học, mô tả, ảnh bìa, tình trạng, hình thức: cho mượn/bán/trao đổi).

Hỗ trợ tìm kiếm/lọc theo tên sách, môn học, tác giả, tình trạng.

Cho phép liên hệ giữa người đăng và người quan tâm, tạo giao dịch (mượn, bán, trao đổi).

Theo dõi lịch sử và trạng thái giao dịch (đang mượn, quá hạn, đã hoàn tất…).

Quản trị: thống kê số lượng, phân loại theo bộ môn, xử lý báo cáo vi phạm (ví dụ: trả trễ).

Đảm bảo minh bạch thông tin**,** dễ dùng, và mở rộng tốt về sau.

Đối tượng sử dụng:

Sinh viên: đăng sách, tìm sách, thực hiện giao dịch.

Quản trị viên: quản lý danh mục, người dùng, báo cáo vi phạm, và thống kê.

1.4 Phạm vi nghiên cứu của đề tài

Đăng & quản lý sách: tạo/sửa/xóa bài đăng với tiêu đề, tác giả, môn học, mô tả, tình trạng (mới/cũ), hình thức (mượn/bán/trao đổi), ảnh bìa (đường dẫn/file).

Tìm kiếm & lọc: theo tên sách, tác giả, môn học, tình trạng, hình thức; sắp xếp theo thời gian đăng hoặc mức độ phổ biến.

Quản lý người dùng: đăng ký/đăng nhập, hồ sơ cá nhân cơ bản, vai trò Student và Admin, điểm tín nhiệm (reputation) ở mức đơn giản.

Giao dịch: tạo yêu cầu, chấp nhận/huỷ; theo dõi trạng thái (đang chờ, đang mượn, hoàn tất, quá hạn). Hỗ trợ gia hạn 1–n lần theo quy tắc.

Nhắc hạn & thông báo: tạo thông báo sự kiện (yêu cầu mới, chấp nhận, sắp đến hạn/quá hạn).

Báo cáo vi phạm: gửi báo cáo (trễ hạn, sách hỏng, spam), Admin xử lý (ghi nhận, đóng).

Thống kê cơ bản: số lượng sách theo bộ môn, tỉ lệ mượn/trả đúng hạn, top người dùng hoạt động.

Yêu cầu phi chức năng (non-functional):

Dễ sử dụng (UI rõ ràng), hỗ trợ tiếng Việt.

Hiệu năng: tìm kiếm nhanh, xử lý nhiều yêu cầu đồng thời quy mô câu lạc bộ/khóa học.

Bảo mật & quyền riêng tư: xác thực, phân quyền, che thông tin liên hệ khi chưa chấp nhận giao dịch.

Tin cậy: ghi log giao dịch, hạn chế gian lận.

Khả năng mở rộng: dễ thêm tính năng mới (ví dụ đánh giá sách, gợi ý theo môn học sắp học).

CHƯƠNG 2: CỞ SỞ LÝ THUYẾT

2.1 Tổng quan về lập trình hướng đối tượng và ngôn ngữ java

2.1.1 Lập trình hướng đối tượng (Object-Oriented Programming – OOP)

Lập trình hướng đối tượng (Object-Oriented Programming – OOP) là một mô hình tổ chức phần mềm xoay quanh đối tượng (object) — những thực thể có trạng thái (thuộc tính/fields) và hành vi (phương thức/methods). Mục tiêu của OOP là mô hình hóa thế giới thực một cách tự nhiên, giúp mã dễ mở rộng, bảo trì, và tái sử dụng.

Java là một ngôn ngữ lập trình hướng đối tượng bậc cao, đa nền tảng, nổi bật với khẩu hiệu "Write Once, Run Anywhere (WORA)" nhờ chạy trên JVM (Java Virtual Machine). Java được dùng rộng rãi trong ứng dụng doanh nghiệp, Android, hệ thống backend quy mô lớn, và nhiều lĩnh vực khác.

2.1.2 Các trụ cột của OOP

Đóng gói (Encapsulation)

Che giấu dữ liệu và triển khai bên trong đối tượng; chỉ lộ ra giao diện công khai (public API).

Giúp giảm phụ thuộc, tăng an toàn dữ liệu.

Trừu tượng (Abstraction)

Chỉ bộc lộ các khái niệm/đặc trưng cần thiết; ẩn chi tiết không liên quan.

Thường dùng interface hoặc abstract class để định nghĩa "cái gì" thay vì "làm thế nào".

Kế thừa (Inheritance)

Lớp con (subclass) kế thừa thuộc tính & hành vi của lớp cha (superclass).

Giúp tái sử dụng mã; cho phép mở rộng hành vi bằng ghi đè (override).

Đa hình (Polymorphism)

Đa hình động (runtime / overriding): cùng một lời gọi phương thức nhưng đối tượng cụ thể quyết định phương thức nào được thực thi.

Đa hình tĩnh (compile-time / overloading): nhiều phương thức cùng tên khác tham số.

=>Lợi ích tổng quát: Mã dễ đọc, bảo trì, mở rộng; ràng buộc tốt giữa dữ liệu – hành vi; dễ kiểm thử và tái sử dụng thành phần.

2.1.3 Thuật ngữ – Khái niệm cốt lõi trong OOP

Class (Lớp): Bản thiết kế mô tả thuộc tính & phương thức.

Object (Đối tượng): Thực thể được tạo từ lớp.

Field/Attribute: Biến thể hiện lưu trạng thái.

Method: Hành vi/logic nghiệp vụ của đối tượng.

Constructor: Hàm khởi tạo đối tượng.

this / super: Truy cập thành phần của đối tượng hiện tại/lớp cha.

Access modifiers: public, protected, package-private (mặc định), private.

Static vs Instance: Thành phần tĩnh thuộc lớp; thành phần thường (instance) thuộc mỗi đối tượng.

Package & Module: Tổ chức mã; từ Java 9 có hệ mô-đun (JPMS).

Interface vs Abstract class: Interface định nghĩa "hợp đồng"; abstract class cho phép chia sẻ một phần cài đặt.

Inner class, Enum: Công cụ tổ chức mã; mô tả tập giá trị cố định.

Generics: Kiểu tham số hóa giúp an toàn kiểu; giảm ép kiểu.

Exception: Cơ chế xử lý lỗi có kiểm tra (checked) và không kiểm tra (unchecked).

Immutability: Đối tượng bất biến giúp an toàn luồng & dễ suy luận.

2.1.4 Nguyên lý thiết kế SOLID

Single Responsibility: Mỗi lớp có một lý do để thay đổi.

Open/Closed: Mở rộng được, hạn chế sửa trực tiếp.

Liskov Substitution: Lớp con thay thế được lớp cha mà không phá vỡ tính đúng đắn.

Interface Segregation: Tách nhỏ interface, tránh "interface béo".

Dependency Inversion: Phụ thuộc vào trừu tượng, không phụ thuộc cụ thể.

2.1.5 UML cơ bản cho OOP

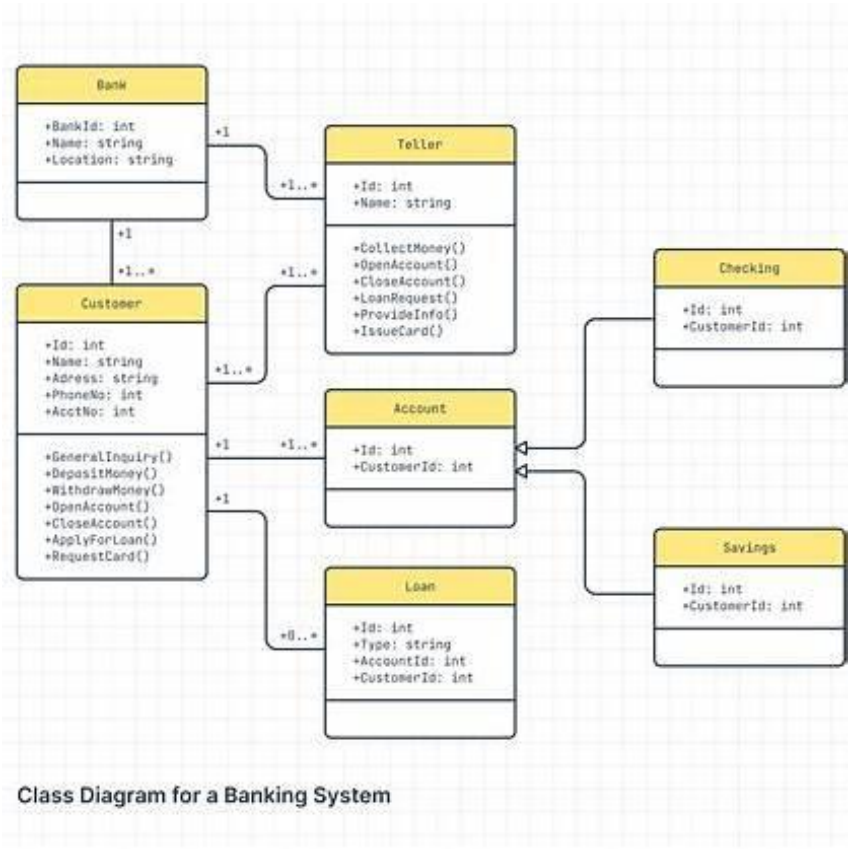
Use Case Diagram: Mô tả người dùng (actor) và trường hợp sử dụng.

Class Diagram: Cấu trúc lớp, thuộc tính, phương thức, quan hệ (kết hợp, kết tập, hợp thành, kế thừa).

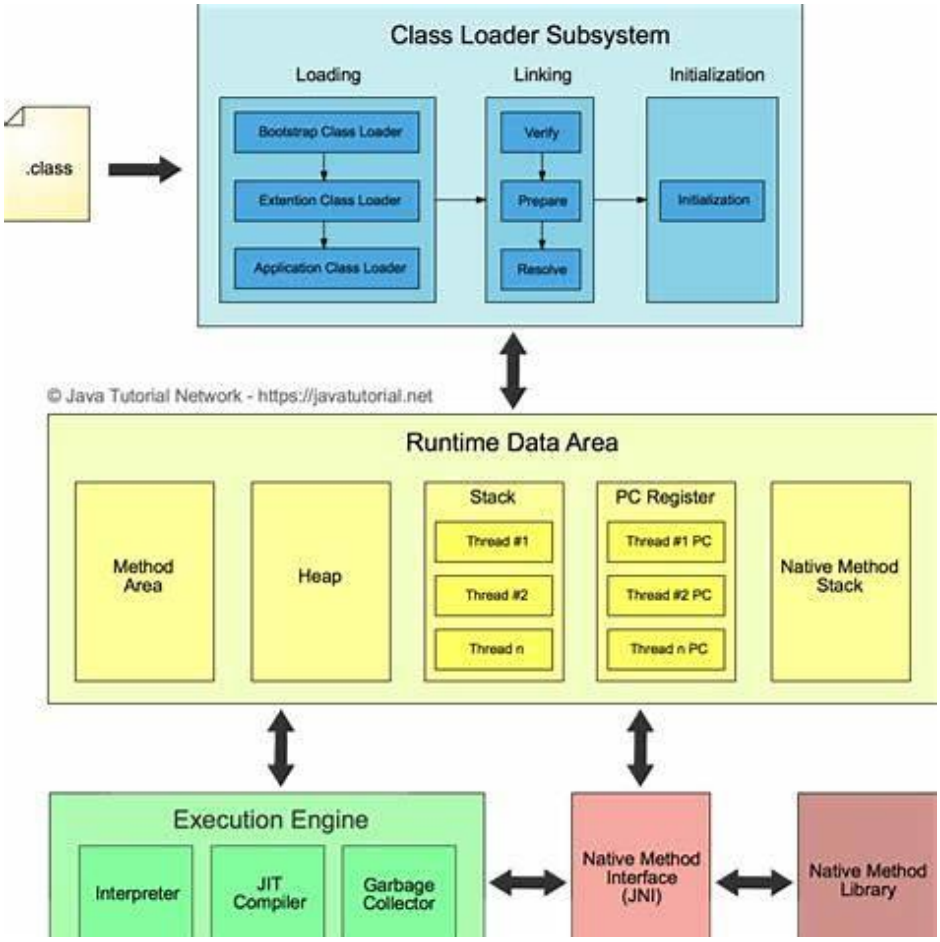
Sequence Diagram: Luồng thông điệp theo thời gian giữa đối tượng.

Activity/State Diagram: Luồng công việc, trạng thái – chuyển trạng thái.

Hình 1. Ví dụ Class Diagram mô hình hệ thống ngân hàng



Hình 2. Ví dụ Sequence Diagram (quy trình đăng ký hoặc mua hàng).



2.2 Tổng quan về Java

2.2.1 Lịch sử & đặc điểm

Ra mắt 1995, do Sun Microsystems (nay thuộc Oracle).

Tính đa nền tảng nhờ bytecode chạy trên JVM.

Bộ thư viện phong phú, hệ sinh thái lớn (Spring, Jakarta EE, Android, Big Data…).

Lộ trình phát hành 6 tháng/lần; các bản LTS (ví dụ: Java 8, 11, 17, 21).

2.2.2 JDK, JRE, JVM

JDK (Java Development Kit): Biên dịch, công cụ phát triển (javac, javadoc, jlink…).

JRE (Java Runtime Environment): Thư viện lõi + JVM để chạy ứng dụng.

JVM: Máy ảo thực thi bytecode; độc lập hệ điều hành.

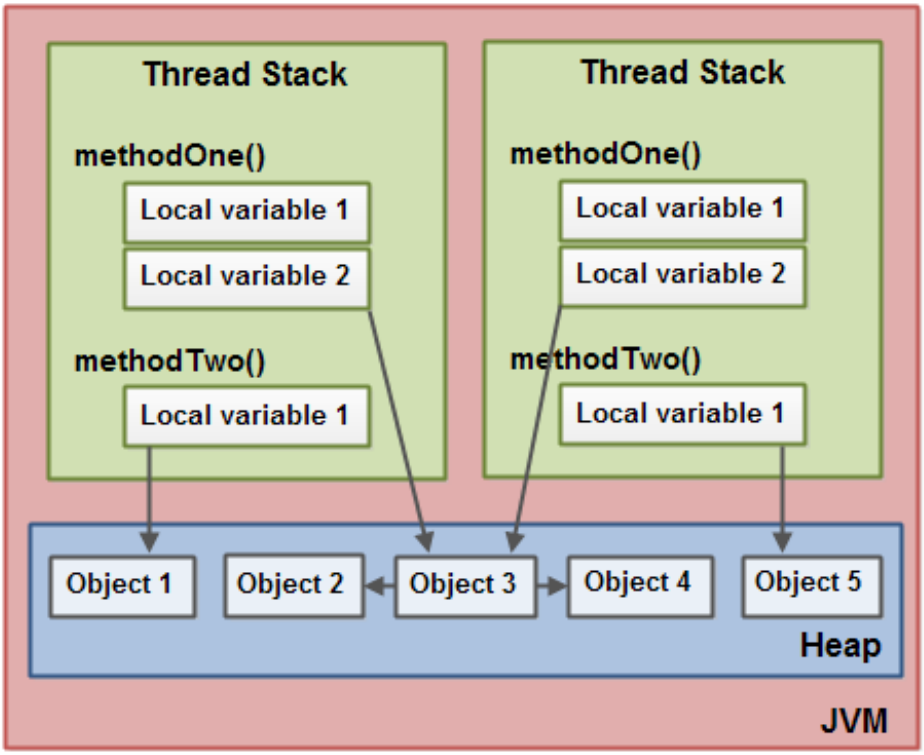
2.2.3 Kiến trúc JVM

Class Loader: Nạp lớp; pha Loading → Linking → Initialization.

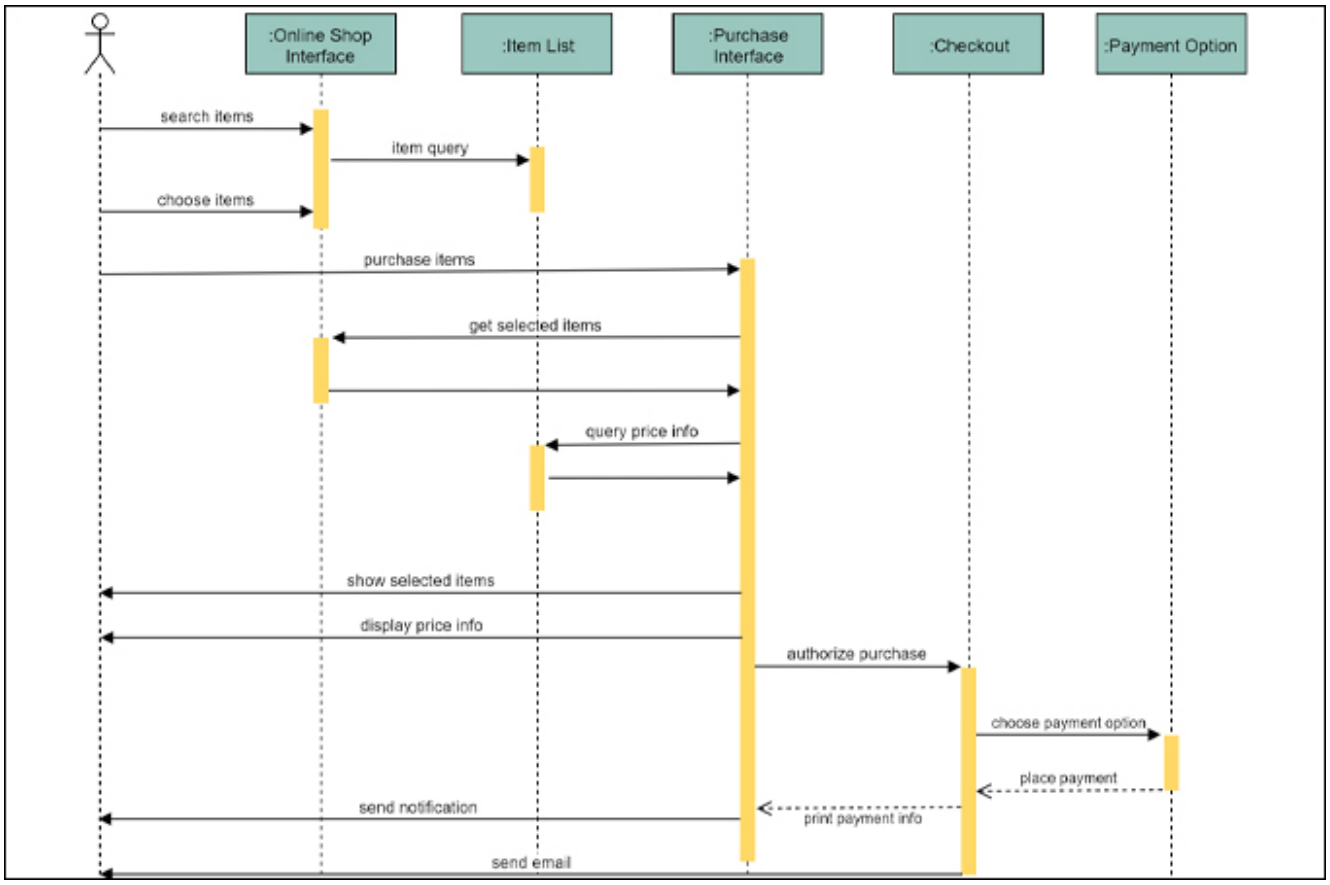
Runtime Data Areas: Method Area, Heap, JVM Stacks, PC Registers, Native Method Stacks.

Execution Engine: Bộ thông dịch, JIT Compiler, Garbage Collector, JNI.

Hình 3. Sơ đồ kiến trúc JVM.



Hình 4. Mô hình bộ nhớ Java (Heap/Stack/GC, vùng thế hệ trẻ – già…).



2.2.4 Cú pháp Java căn bản

```java
// Hello OOP in Java
package demo;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, OOP!");
    }
}
```

Kiểu dữ liệu nguyên thủy: byte, short, int, long, float, double, char, boolean.

Tham chiếu: lớp, mảng, interface, enum…

Điều khiển luồng: if/else, switch, vòng lặp for/while/do-while, break/continue.

2.3 OOP trong Java

2.3.1 Lớp, đối tượng, đóng gói

```
public class BankAccount {
    private String owner;     // đóng gói: private field
    private double balance;

    public BankAccount(String owner, double init) {
        this.owner = owner;
        this.balance = init;
    }

    public void deposit(double amount) {
        if (amount <= 0) throw new IllegalArgumentException("amount>0");
        balance += amount;
    }

    public boolean withdraw(double amount) {
        if (amount <= 0 || amount > balance) return false;
        balance -= amount; return true;
    }

    public double getBalance() { return balance; }
}
```

2.3.2 Kế thừa & đa hình (override)

```
abstract class Employee {
    protected String name;
    public Employee(String name) { this.name = name; }
    public abstract double monthlyPay(); // trừu tượng
}

class SalariedEmployee extends Employee {
    private double salary;
    public SalariedEmployee(String n, double s) { super(n); this.salary = s; }
    @Override public double monthlyPay() { return salary; }
}

class HourlyEmployee extends Employee {
    private double wage; private int hours;
    public HourlyEmployee(String n, double w, int h) { super(n); wage=w; hours=
    @Override public double monthlyPay() { return wage * hours; }
}
```

2.3.3 Interface & Dependency Inversion

```
interface PaymentGateway { boolean charge(String userId, double amount); }

class StripeGateway implements PaymentGateway {
    public boolean charge(String userId, double amount) { /* gọi API */ return
}

class CheckoutService {
    private final PaymentGateway gateway; // phụ thuộc vào trừu tượng
    public CheckoutService(PaymentGateway gateway) { this.gateway = gateway; }
    public boolean checkout(String userId, double amount) {
        return gateway.charge(userId, amount);
    }
}
```

2.3.4 Generics, Collections & Stream API

Generics: an toàn kiểu, tránh ép kiểu runtime; hỗ trợ List<String>, Map<K,V>, giới hạn extends/super, wildcards ?.

Collections Framework: List, Set, Queue, Map + triển khai ArrayList, HashSet, LinkedList, HashMap…

Stream API (Java 8+): Lập trình hàm (map/filter/reduce), xử lý dữ liệu bất biến, dễ song song hóa.

```
List<Integer> nums = List.of(1,2,3,4,5);
int sumSquares = nums.stream()
        .map(n -> n*n)
        .reduce(0, Integer::sum);
```

2.3.5 Ngoại lệ (Exceptions)

Checked**:** buộc khai báo/try-catch (IOException, SQLException…).

Unchecked (Runtime): NullPointerException, IllegalArgumentException…

Quy ước: chỉ dùng ngoại lệ cho tình huống bất thường; ưu tiên thông điệp rõ ràng.

```
try (var br = Files.newBufferedReader(Path.of("data.txt"))) {
    // xử lý
} catch (IOException e) {
    // ghi log, thông báo
}
```

2.3.6 Lập trình đồng thời (Concurrency)

Thread & Runnable, ExecutorService, Future/CompletableFuture.

Đồng bộ: synchronized, từ khóa volatile, Locks/Condition, Semaphore.

Java Memory Model (JMM): quy định tính nhìn thấy (visibility), thứ tự (ordering).

Bộ sưu tập an toàn luồng: ConcurrentHashMap, CopyOnWriteArrayList…

```
ExecutorService pool = Executors.newFixedThreadPool(4);
Future<Integer> f = pool.submit(() -> expensiveCompute());
int result = f.get();
pool.shutdown();
```

2.3.7 Mẫu thiết kế (Design Patterns)

Creational: Singleton, Factory Method, Abstract Factory, Builder, Prototype.

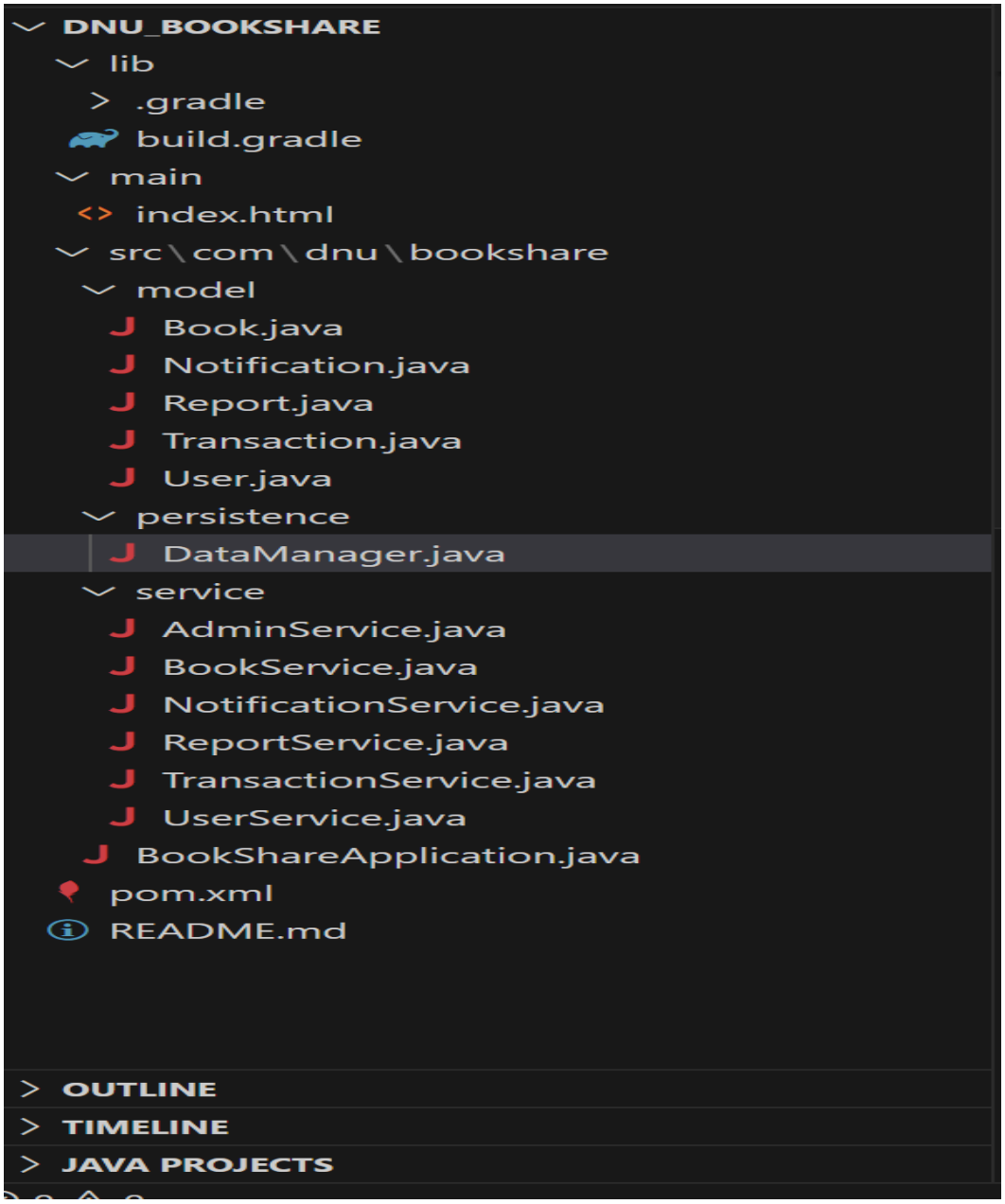
Structural: Adapter, Facade, Decorator, Composite.

Behavioral: Strategy, Observer, Command, Template Method.

```
class User {
    private final String name; private final String email; private final int ag
    private User(Builder b){ this.name=b.name; this.email=b.email; this.age=b.a
    public static class Builder {
        private String name, email; private int age;
        public Builder name(String v){this.name=v; return this;}
        public Builder email(String v){this.email=v; return this;}
        public Builder age(int v){this.age=v; return this;}
        public User build(){ return new User(this);} }
}
```

CHƯƠNG 3: TỔNG QUAN KIẾN TRÚC DỰ ÁN

3.1 Tổng quan kiến trúc lập trình oop



Tổng thể mô hình:

Ứng dụng được tổ chức theo kiến trúc phân lớp (đa tầng) với ba lớp chính: Model (Domain) , Service (Business Logic) và Persistence (Data Access) . Điểm vào chương trình và giao diện (nguyên mẫu) được phân tách riêng, giúp xác định nguồn mã hóa, mở rộng dễ dàng.

Cấu trúc gói:

-com.dnu.bookshare.model/

Chứa các thực thể nghiệp vụ : User, Book, Transaction, Report, Notification. Mỗi thực

thể mô tả cốt lõi dữ liệu, quan hệ và các phương thức nghiệp vụ đơn giản (xác thực nhẹ nhàng, thay đổi trạng thái).

-com.dnu.bookshare.service/

Các dịch vụ theo tên miền : UserService, BookService, TransactionService, ReportService, NotificationService. AdminServiceĐây là nơi hiện thực quy tắc nghiệp vụ : đăng/tra cứu sách, tạo giao dịch mượn–bán–trao đổi, theo dõi lịch sử, cảnh báo quá hạn, xử lý báo cáo vi phạm…

-com.dnu.bookshare.persistence/

DataManagerđảm bảo truy cập dữ liệu (tệp trong bộ nhớ/đọc–ghi). Lớp này hoàn toàn hoàn hảo với các doanh nghiệp, có thể thay thế bằng cơ sở dữ liệu thực trong tương lai.

-BookShareApplication.java

Điểm vào ứng dụng : khởi tạo DataManager, "kết nối" các Dịch vụ và điều phối luồng cuộc gọi từ giao diện/CLI.

-main/index.html

Mẫu tĩnh giao diện (nguyên mẫu) được sử dụng để minh họa người dùng thao tác trực tuyến.

Công cụ xây dựng: dự án hiện có build.gradlevà pom.xml; chỉ chọn một (Maven *hoặc* Gradle) để tránh xung đột.

-Luồng dữ liệu (Luồng dữ liệu):

UI/CLI → Dịch vụ (xử lý nghiệp vụ, kiểm tra thanh toán) → Persistence/DataManager (lưu/truy xuất) → trả về Model/DTO → UI hiển thị kết quả.

-Áp dụng quy tắc thiết kế:

Tách biệt mối quan tâm: phân tích dữ liệu, nghiệp vụ và lưu trữ thành các tầng rõ ràng.

Trách nhiệm duy nhất: mỗi lớp/gói có một nhiệm vụ chính (ví dụ: BookServicexử lý logic về sách).

Open–Closed/Dependency Inversion (định hướng): Hướng dẫn dịch vụ tới giao diện phụ thuộc của các tầng lưu trữ, cho phép thay đổi backend mà không thay đổi logic.

Tính mở rộng: dễ dàng bổ sung các tính năng bằng cách bổ sung Dịch vụ hoặc Mô hình mở rộng mà không ảnh hưởng đến các tầng khác.

Ưu điểm kiến trúc:

Mã nguồn dễ đọc – dễ kiểm tra (viết JUnit cho từng Service).

Tái sử dụng và bảo trì thuận lợi nhờ biên giới rõ ràng.

Khả năng nâng cấp cơ sở dữ liệu thật (JDBC/JPA) mà không thay đổi dịch vụ trực tuyến.

Hướng hoàn thiện/lộ trình kỹ thuật:

Trích xuất kho giao diện (ví dụ: UserRepository, BookRepository) và để DataManagerthực hiện; Giao diện phụ thuộc dịch vụ.

Bổ sung Controller/Adapter cho UI (CLI hoặc Web) để phân tách I/O khỏi BookShareApplication.

Sử dụng DTO/ViewModel khi trả dữ liệu ra giao diện, tránh để hiển thị miền thực thể trực tiếp.

Chuẩn hóa Ngoại lệ /mã lỗi (ví dụ OverdueException, NotFoundException) và phản hồi hệ thống nhất.

Thêm kiểm tra JUnit + Mockito cho Service; phân tách cấu hình/hằng số riêng biệt.

Chuẩn hóa công cụ xây dựng (giữ Maven *hoặc* Gradle), thiết lập pipe build & checkstyle.

Kết Luận:

Kiến trúc hiện tại là một bộ xương được xếp lớp sạch **sẽ** cho bài toán chia sẻ/trao đổi sách. Với các bước hoàn thiện ở trên (giao diện kho lưu trữ, bộ điều khiển, kiểm tra, bản dựng chuẩn), hệ thống đủ nền tảng để mở rộng hoàn thiện ứng dụng, đảm bảo tính năng mô-đun, bảo trì dễ dàng và cơ sở dữ liệu sẵn có.

3.2 Chức năng

55 Chức năng đã hoàn thiện:

Người dùng (8 chức năng)

- Đăng ký tài khoản sinh viên
- Đăng nhập/Đăng xuất
- Quản lý hồ sơ cá nhân
- Đổi mật khẩu
- Quên mật khẩu
- Xác thực email sinh viên
- Xem lịch sử hoạt động
- Đánh giá uy tín người dùng

Quản lý Sách (12 chức năng)

- Đăng tải sách mới
- Chỉnh sửa thông tin sách
- Xóa sách đã đăng

- Upload ảnh bìa sách
- Mô tả chi tiết sách
- Chọn tình trạng sách
- Chọn hình thức giao dịch
- Đặt giá sách
- Đặt thời gian cho mượn
- Đánh dấu sách yêu thích
- Ẩn/Hiện sách
- Xem thống kê lượt xem

Tìm kiếm & Lọc (8 chức năng)

- Tìm kiếm theo tên sách
- Tìm kiếm theo tác giả
- Tìm kiếm theo môn học
- Lọc theo bộ môn/khoa
- Lọc theo tình trạng sách
- Lọc theo hình thức
- Sắp xếp theo giá/ngày đăng
- Tìm kiếm nâng cao

Giao dịch (10 chức năng)

- Gửi yêu cầu mượn sách
- Gửi yêu cầu mua sách
- Đề xuất trao đổi sách
- Chấp nhận/Từ chối yêu cầu
- Xác nhận giao sách
- Xác nhận trả sách
- Gia hạn thời gian mượn
- Thanh toán (giả lập)
- Đánh giá sau giao dịch
- Chat/Liên hệ chủ sách

Lịch sử & Thống kê (7 chức năng)

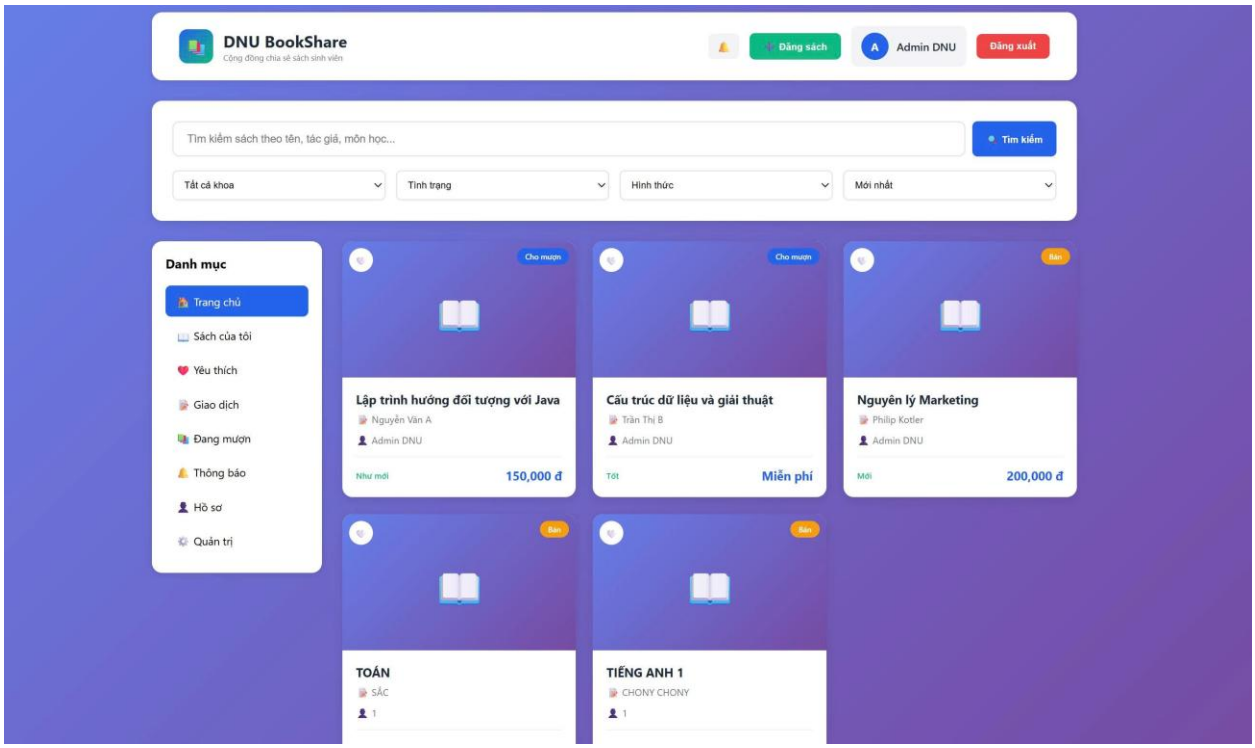
- Xem lịch sử cho mượn
- Xem lịch sử đi mượn
- Xem lịch sử mua bán
- Theo dõi sách đang mượn
- Nhắc nhở trả sách (kiểm tra quá hạn)
- Thống kê cá nhân
- Xuất báo cáo

Quản trị viên (10 chức năng)

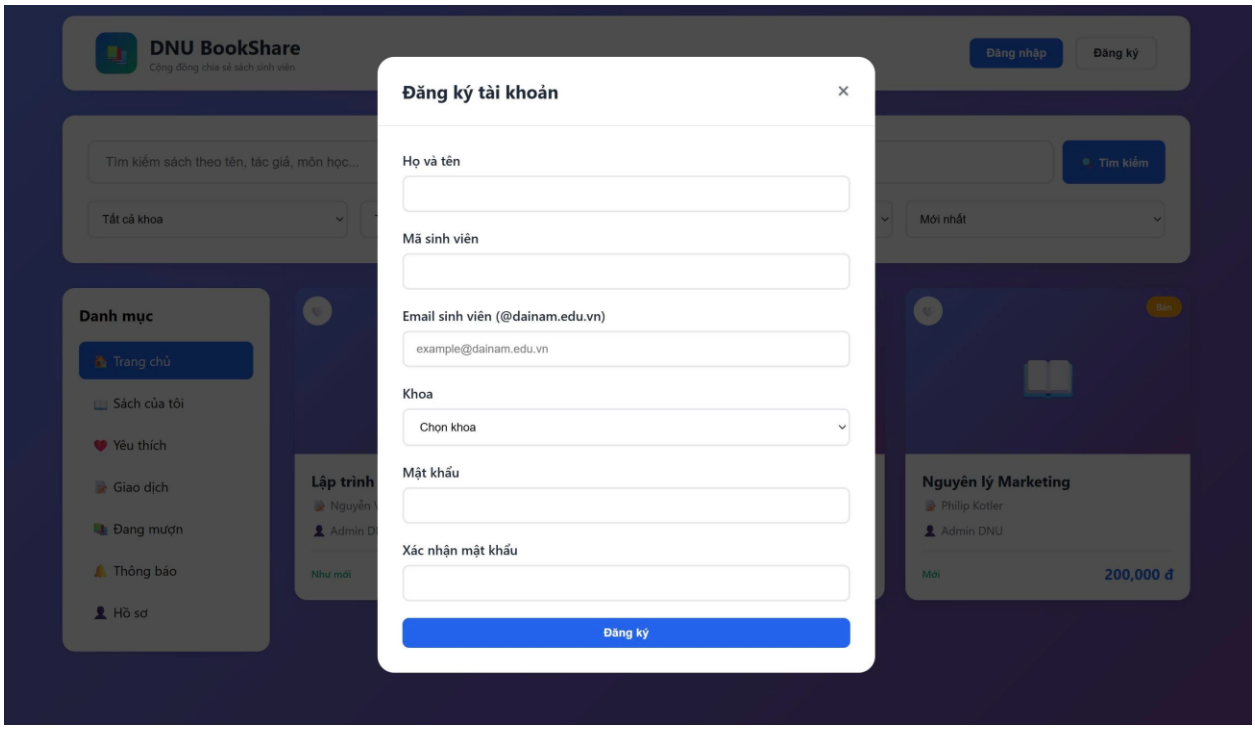
- Quản lý tất cả người dùng
- Xem thống kê hệ thống
- Phân loại sách theo bộ môn
- Xử lý báo cáo vi phạm
- Khóa/Mở khóa tài khoản
- Xóa sách vi phạm
- Gửi thông báo hệ thống
- Cấu hình quy định
- Backup dữ liệu
- Xem log hoạt động

CHƯƠNG 4:GIAO DIỆN ỨNG DỤNG

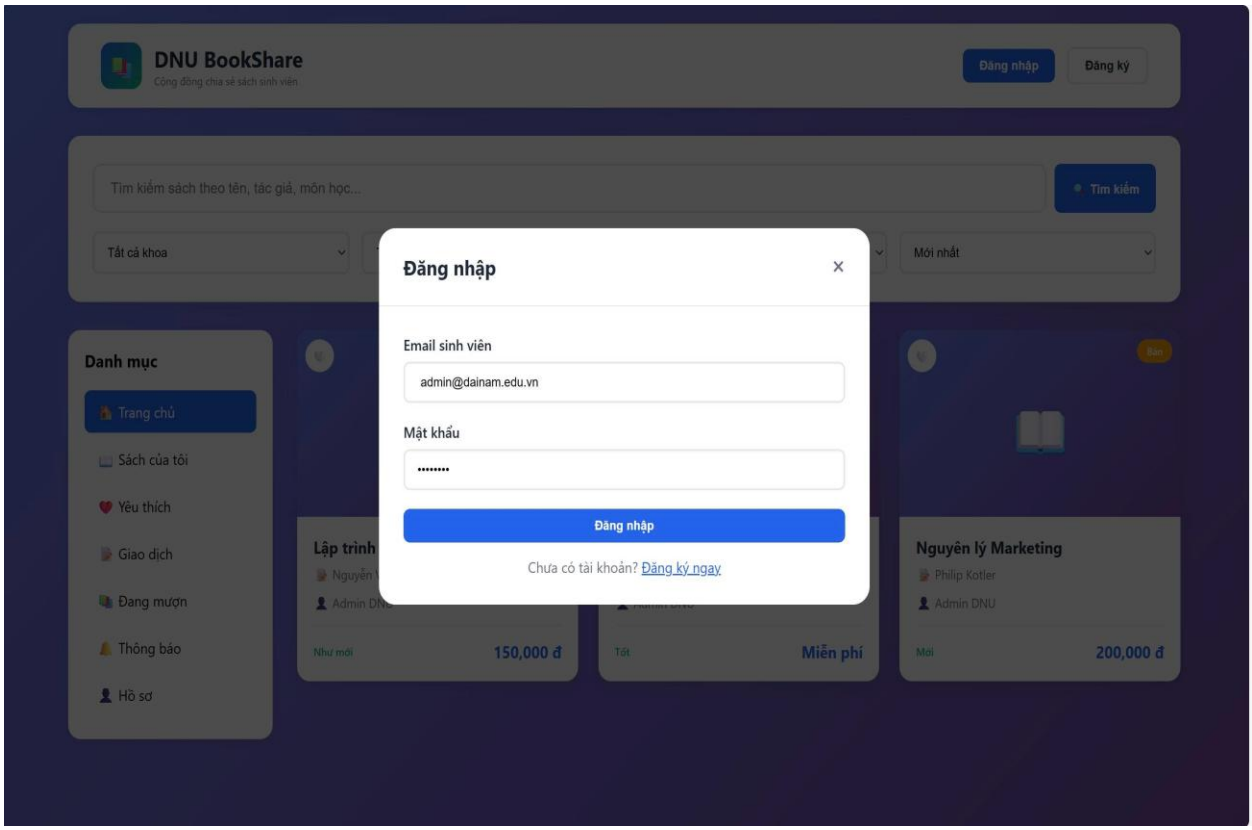
Hình 1: Giao diện trang chủ



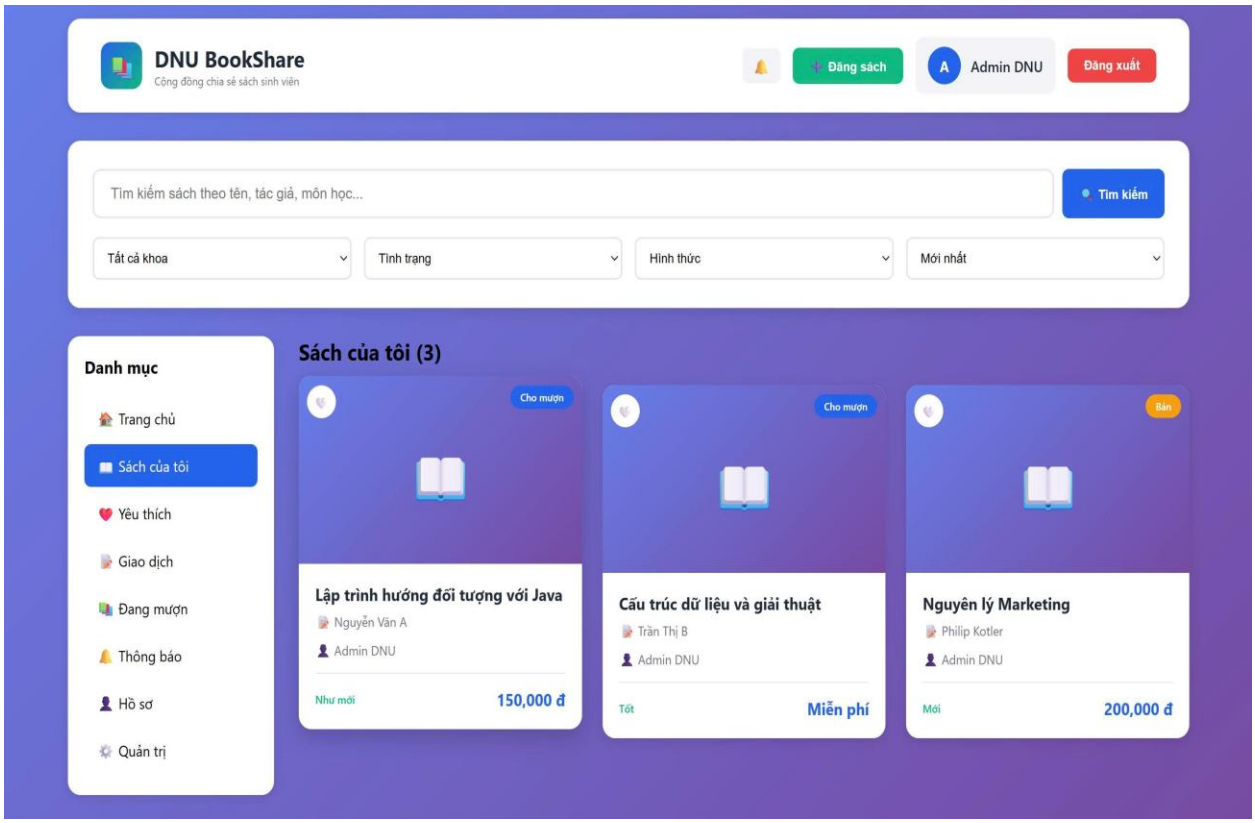
Hình 2: Giao diện đăng kí tài khoản:



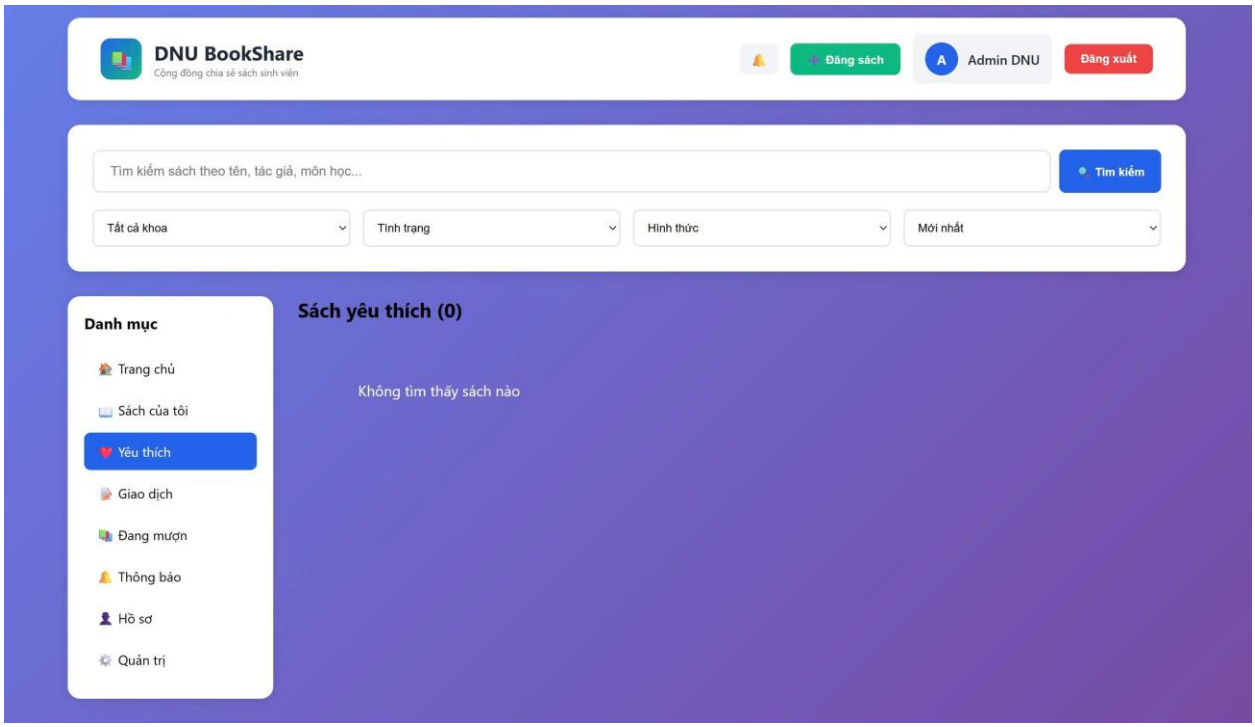
Hình 3: Giao diện đăng nhập:



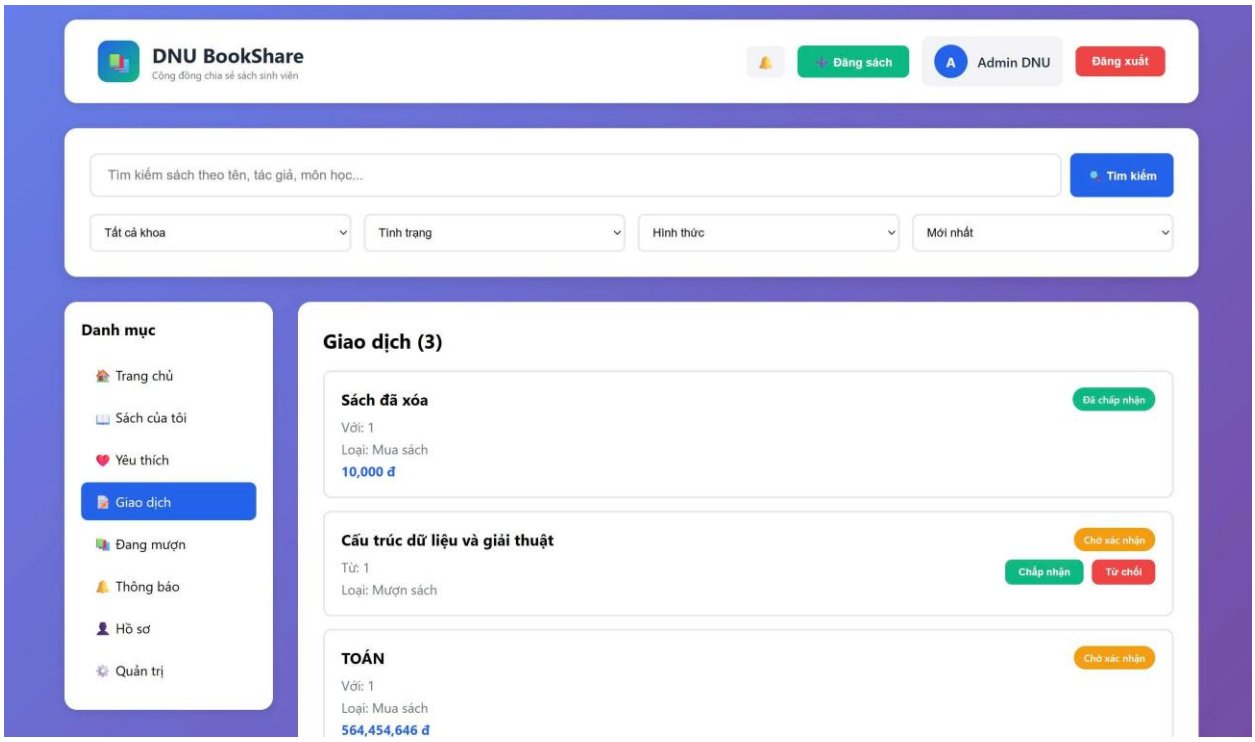
Hình 4: Giao diện sách của tôi:



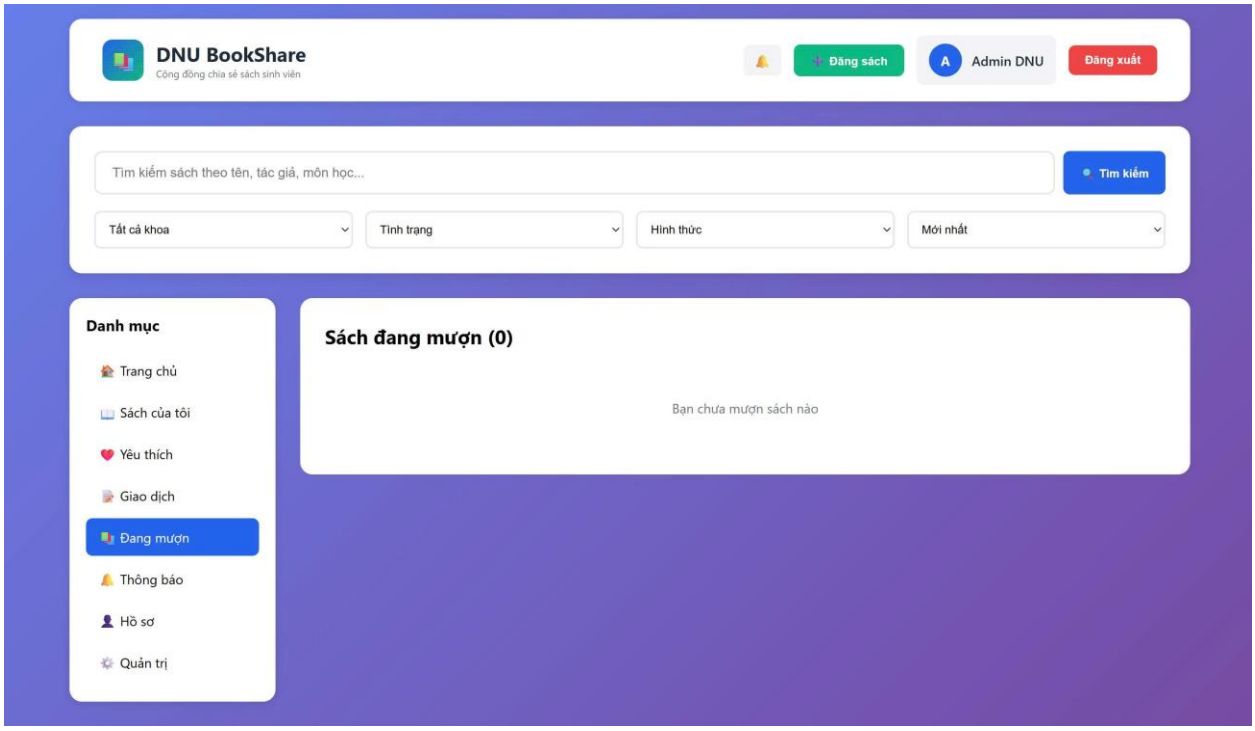
Hình 5: Giao diện yêu thích:



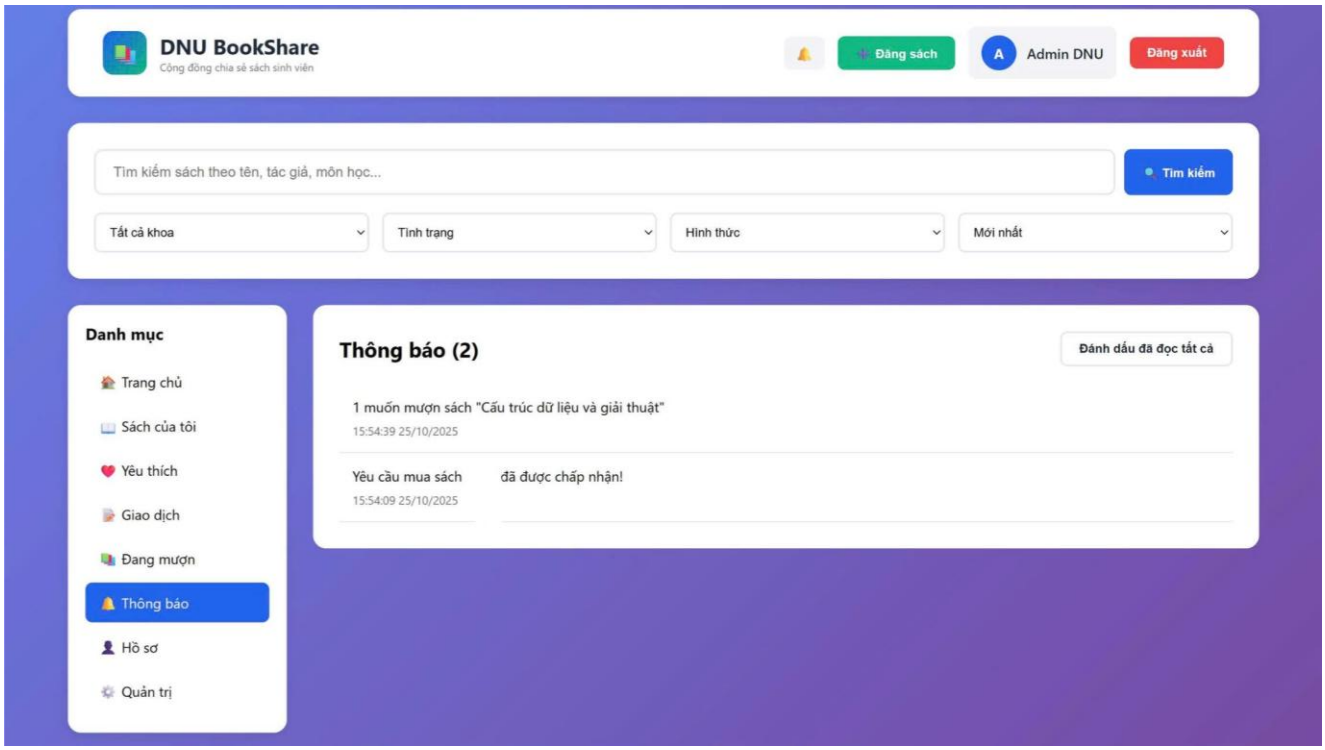
Hình 6: Giao diện giao dịch:



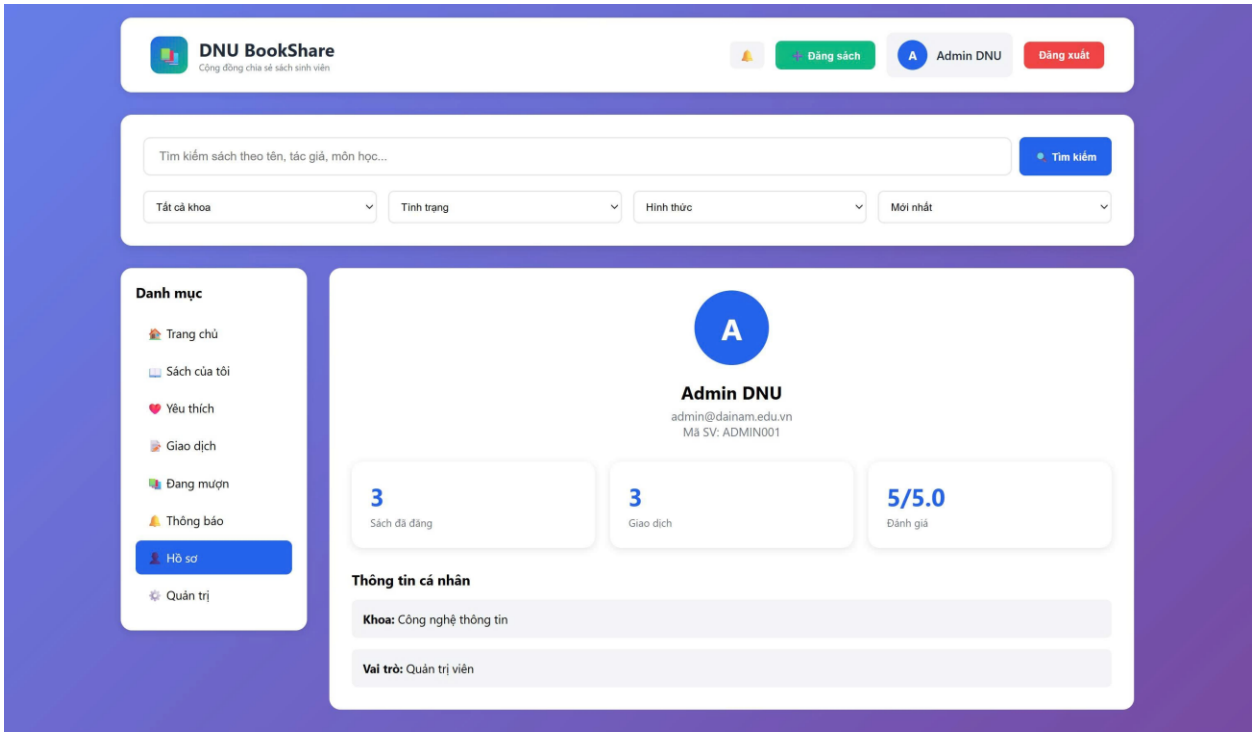
Hình 7: Giao diện sách đang mượn:



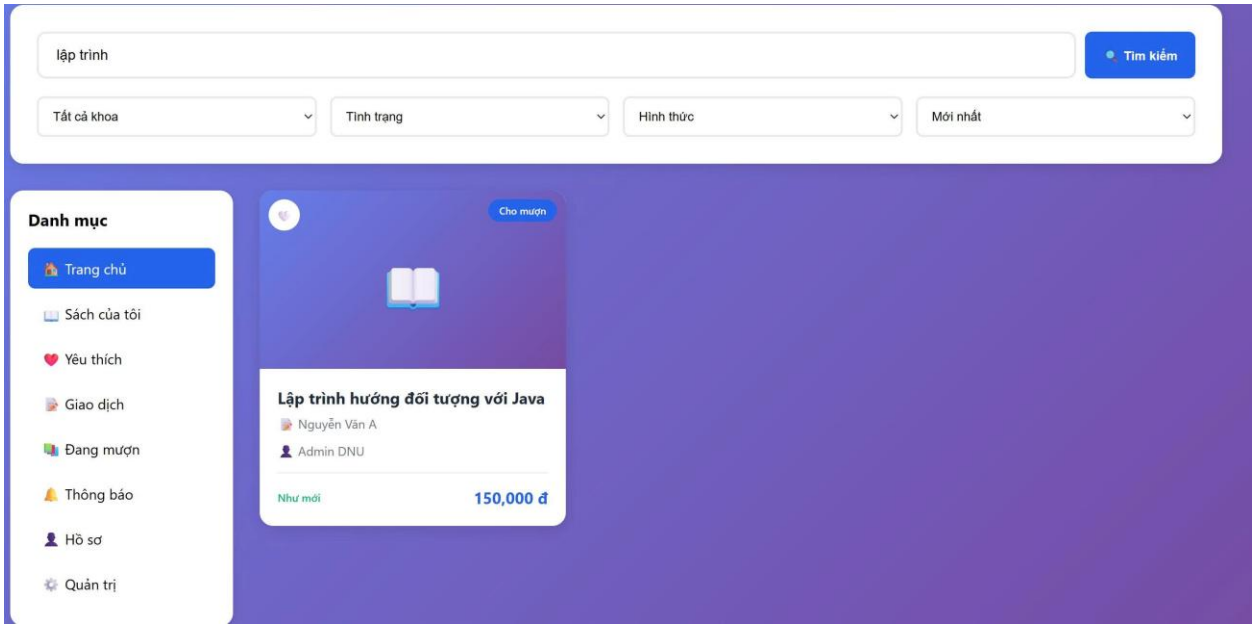
Hình 8: Giao diện thông báo:



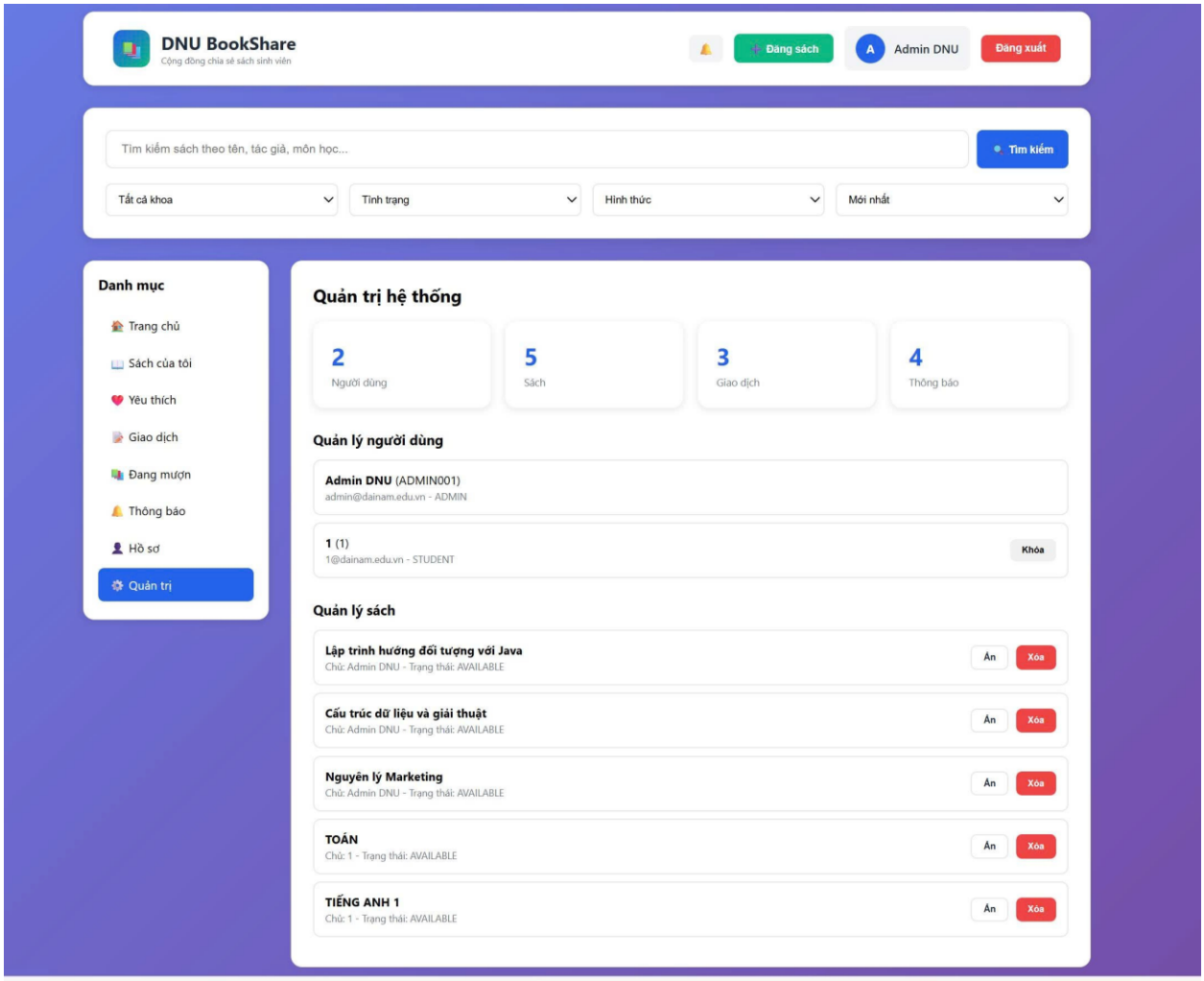
Hình 9: Giao diện hồ sơ:



Hình 10: Giao diện tìm kiếm:



Hình 11: Giao diện trang quản trị:



CHƯƠNG 5: GIẢI THÍCH CƠ BẢN MÃ NGUỒN DỰ ÁN

5.1 Khởi tạo dự án & thư mục cấu trúc

Tạo gói: com.dnu.bookshare.{model,service,persistence}.

Add BookShareApplication.java(điểm vào) và README.md.

Cài đặt tầng Model:

Viết lớp User, Book, Transaction, Report, Notification

Thiết kế và thực hiện Persistence

Tạo DataManager (trong bộ nhớ/JSON).

(Tùy chọn) tạo kho lưu trữ giao diện để dễ dàng thay thế phần phụ trợ.

Cài đặt service(dịch vụ) tầng (nghiệp vụ)

UserService, BookService, TransactionService, ReportService, NotificationService, AdminService.

Viết các dịch vụ: đăng/tìm danh sách, tạo giao dịch, theo dõi quá hạn, xử lý báo cáo.

Kết nối ứng dụng (wires)

Trong BookShareApplication, khởi tạo DataManager+ Dịch vụ, tải mẫu dữ liệu

Xây dựng giao diện đơn giản

Làm bảng điều khiển menu hoặc index.html nguyên mẫu để gọi các chức năng chính.

Tối ưu & chuẩn hóa

Thêm tùy chỉnh ngoại lệ, ghi nhật ký, cấu hình số liên tục; up code (SRP, SOLID).

Viết bình luận JavaDoc cần thiết.

Đóng gói & chạy

Build jar (mvn package hoặc gradle jar), chạy trong README.

5.2 Model

5.2.1 Hướng đối tượng trong User.java

- Class mô hình : User đại diện một người dùng trong hệ thống (một "đối tượng" có dữ liệu + hành vi).

- Đóng gói (Encapsulation) : Tất cả thuộc tính để private; Bên ngoài truy cập chỉ qua getter/setter → ẩn thông tin, kiểm soát chỉnh sửa.

- Khởi tạo (Constructor) : Nhận email, password, fullName, studentIdvà đặt mặc định giá trị (role=STUDENT, trustScore=5.0, createdAt=now, isActive=true, tạo favoriteBookIds rỗng).

- Người trợ giúp riêng tư : generateId()tạo userId nội bộ → ẩn cài đặt chi tiết.

- Enum : UserRole { STUDENT, ADMIN }giúp giới hạn và làm rõ vai trò của trò chơi (loại an toàn, dễ hiểu).

- Hành động gắn kết với dữ liệu **:**

addFavoriteBook, removeFavoriteBook thao tác danh sách yêu thích ngay trong lớp (tránh trùng lặp).

- Composition : Use other object làm thành phần như LocalDateTime (thời điểm tạo), List<String> (danh sách yêu thích).

=> Tổng kết: mã nguồn đang áp dụng OOP ở các điểm chính: Class làm mô hình , đóng gói dữ liệu qua private + getter/setter , object object qua constructor & enum & helper private , thành phần với List/LocalDateTime , và định nghĩa hành vi ngay trong lớp (thêm/xóa sách yêu thích).

5.2.2 Hướng đối tượng trong transaction.java

- Lớp mô hình : Transaction đại diện cho một giao dịch (đối tượng có dữ liệu + hành vi).

Đóng gói (Encapsulation) : Tất cả thuộc tính private; truy cập/chỉnh sửa qua getter/setter → ẩn thông tin, kiểm soát trạng thái.

- Khởi tạo (Constructor) : Nhận bookId, ownerId, borrowerId, type; auto sinh transactionId, set default status=PENDING, requestedAt=now.

- Người trợ giúp riêng tư : generateId() ẩn cách tạo ID → chi tiết nội bộ không tiết lộ ra bên ngoài.

- Enum **:**

TransactionType { BORROW, BUY, EXCHANGE }(hình thức giao dịch).

TransactionStatus { PENDING, APPROVED, REJECTED, IN_PROGRESS, COMPLETED, OVERDUE, CANCELLED }(vòng đời trạng thái).
→ Giới hạn giá trị hợp lệ, loại an toàn, xác định nghĩa.

- Composition **:** Use many LocalDateTime(requestedAt, đã phê duyệtAt, ..., doDate) làm thành phần thời gian của giao dịch.

- Hành động gắn kết với dữ liệu : isOverdue()kiểm tra quá hạn chế dựa trên dueDate và status (dịch vụ logic nằm trong đối tượng chính).

=>Tổng kết: Lớp Transactionthể hiện OOP qua đóng gói dữ liệu , vật thể hóa bằng constructor/enum/helper riêng tư, thành phần với LocalDateTime, và hành vi nội tại (isOverdue) Gắn chặt với trạng thái giao dịch.

5.2.3 Hướng đối tượng trong Report.java

- Lớp mô hình : Reportđại diện cho một báo cáo vi phạm trong hệ thống (đối tượng có dữ liệu + hành vi).

- Đóng gói (Encapsulation) : Tất cả thuộc tính để private; truy cập/chỉnh sửa qua getter/setter → ẩn thông tin, kiểm soát trạng thái.

- Khởi tạo (Constructor) : Nhận reporterId, reportedUserId, type, description; tự động reportId, đặt mặc định status = PENDING, đóng dấu createdAt = now.

- Người trợ giúp riêng tư : generateId()ẩn cách tạo ID → chi tiết nội bộ không tiết lộ ra bên ngoài.

- Enum :

ReportType(có cả chuỗi tiếng Việt và getVietnamese()) → type-safe, gói chung dữ liệu + hành vi liên quan đến loại báo cáo.

ReportStatusmô tả trình xử lý trạng thái: PENDING, INVESTIGATING, RESOLVED, REJECTED.

- Composition : Use LocalDateTimefor các thời gian (createdAt, resolvedAt) làm thành phần của đối tượng.

=>Tổng kết:

Report.java áp dụng đúng tinh thần OOP trong các cốt lõi cốt lõi: đóng gói dữ liệu và hành vi, vật thể hóa hóa qua constructor/enum/helper private và thành phần với các đối tượng thời gian. Cấu hình rõ ràng, dễ dàng mở rộng cho quá trình xử lý báo cáo nghiệp vụ mà chưa cần đến kế thừa/đa hình.

5.2.4 Hướng đối tượng trong Notification.java

- Class model : Notificationđại diện một thông báo gửi cho người dùng trong hệ thống.

- Đóng gói (Encapsulation) : Mọi thuộc tính đều private; Bên ngoài truy cập/chỉnh sửa qua getter/setter → ẩn thông tin, kiểm soát trạng thái (isRead, relatedId, …).

- Khởi tạo (Constructor) : Nhận userId, type, title, message; tự động notificationId, đặt mặc định isRead=false, đóng dấu thời gian createdAt=now.

- Người trợ giúp riêng tư : generateId()ẩn cách tạo ID → chi tiết nội bộ không tiết lộ ra bên ngoài.

- Enum : NotificationTypethu thập các loại thông báo (yêu cầu mới, duyệt/từ chối, nhanh chóng trả lời, tin nhắn, đánh giá, hệ thống thông báo) → type-safe , xác định nghĩa nghiệp vụ.

- Thành phần : Sử LocalDateTimedụng thành phần để lưu thời gian tạo thông báo.

- Hành vi gắn dữ liệu : Qua setter setRead(boolean)để đánh dấu dấu đã đọc/ chưa đọc ; setRelatedIdliên kết thông báo với thực thể liên quan (VD: giao dịch, sách, tin nhắn).

- Tổng kết:

Notification.java áp dụng đúng các nguyên lý OOP cốt lõi: đóng gói (private + getter/setter), vật tượng hóa (constructor, enum, helper private), và thành phần (thời gian). Thiết kế rõ ràng, đủ cho vòng đời thông báo (tạo → đọc) và dễ dàng mở rộng về sau.

5.2.5 Hướng đối tượng trong Book.java

- Lớp mô hình : Book đại diện cho một cuốn sách trong hệ thống (đối tượng có dữ liệu + hành vi).

- Đóng gói (Encapsulation) : Tất cả thuộc tính private; thao tác từ ngoài qua getter/setter → ẩn thông tin, kiểm soát chỉnh sửa.

- Khởi tạo (Constructor) : Nhận ownerId, title, author; tự sinh bookId; set default condition=GOOD, status=AVAILABLE, postedAt=now, viewCount=0, isVisible=true, và tạo danh sách availableTypes trống.

- Người trợ giúp riêng tư : generateId() ẩn cách tạo ID → chi tiết nội bộ không tiết lộ ra bên ngoài.

- Enum (trừu tượng hóa trạng thái/thuộc tính) :

BookCondition(có kèm theo chuỗi tiếng Việt),

TransactionType(VAY/BÁN/ĐỔI, cũng có tiếng Việt),

BookStatus(Có sẵn/Mượn/BÁN/DỰ LƯỢC).

→ Giới hạn giá trị hợp lệ, loại an toàn, xác định dịch vụ.

- Composition : Sử dụng LocalDateTime postedAt và List<TransactionType> availableTypes tạo cấu hình thành phần của Book.

- Hành động gắn kết với dữ liệu :

addTransactionType()bổ sung thêm giao thức mô hình, tránh trùng lặp ,

incrementViewCount()tăng lượt xem,

Setter thay đổi thuộc tính như status, price, borrowDays, visible…

-Tổng kết:

Book.java áp dụng tốt các nguyên lý OOP cốt lõi: đóng gói , vật tượng hóa (hàm tạo,

enum, trợ giúp riêng tư) và thành phần . Đây là một miền thực thể rõ ràng, có hành vi vi rút gọn đúng nơi dữ liệu; is not used to/đa hình (không bắt buộc trong trường hợp này).

5.3 Service

5.3.1 Hướng đối tượng trong AdminService.java

- Lớp dịch vụ (Lớp dịch vụ) – Trừu tượng hóa dịch vụ quản trị viên
AdminService gom các chức năng quản trị (khóa/mở khóa người dùng, ẩn/xoá sách, ade giao dịch, xử lý báo cáo, thống kê…) thành một đơn vị nghiệp vụ rõ ràng.

- Đóng gói (Encapsulation) Các trường
phụ thuộc (userService, bookService, transactionService, reportService,) là ; mọi hoạt động đi qua phương thức công khai → ẩn chi tiết phát triển
khai.notificationServicedataManagerprivate

- Composition & Delegation (Kết hợp và ủy quyền)
Lớp kết hợp các dịch vụ khác như thành phần để ủy quyền thao tác: ví dụ blockUsergọi userService.toggleUserStatus(...)rồi notificationService.notifySystemAnnouncement(...).
→ AdminService điều phối (điều phối) thay vì tự xử lý tất cả.

- Constructor Inject (tiêm phụ thuộc)
Nhận các dịch vụ qua hàm tạo → phân tách phụ thuộc, dễ dàng test/mocking , tăng tính linh hoạt (SOLID: D ependency Inversion).

- Tách mối quan tâm (Phân tách mối quan tâm)
AdminService chỉ tập trung luồng nghiệp vụ quản trị ; lưu trữ/IO do DataManager, thông báo do NotificationService, dữ liệu miền do các *Service tương ứng đảm bảo trách nhiệm.

- Dịch vụ logic đặt trang chính xác
Ví dụ processReport(...)cập nhật trạng thái báo cáo, gửi thông báo tới hai bên và điều chỉnh TrustScore khi cần—tất cả trong một hệ thống quy trình nhất.

- Sử dụng Stream API (diễn đạt logic rõ ràng)
Thống kê/danh sách hàng đầu được sử dụng stream(), filter/sorted/limit/collect để đạt được logic xử lý tập hợp ngắn gọn, giữ cho lớp dịch vụ gọn và mạch lạc.

- Xử lý trạng thái miền (Trạng thái miền)
Dựa vào enum/trạng thái từ các mô hình (BookStatus, TransactionStatus, ReportStatus) để lọc, thống kê và quyết định hành động.

- Tổng kết:
AdminService thực hiện OOP qua đóng gói và vật thể hóa nghiệp vụ quản trị viên, thành phần + ủy quyền để phân phối nhiều dịch vụ và phân tách các mối quan tâm giúp mã hóa vai trò, dễ bảo trì, dễ kiểm tra. Đây là "điểm điều phối" của quản trị luồng trong hệ thống.

5.3.2 Hướng đối tượng trong BookService.java

- Class dịch vụ (Abstraction of professional books)

BookService gom toàn bộ thao tác với sách (tạo/cập nhật/xoá, ẩn/hiện, tăng view, đổi trạng thái, tìm kiếm–lọc–sắp xếp, thống kê) vào một lớp duy nhất , che giấu chi tiết khai triển.

- Đóng gói (Đóng gói)

Nội dung dữ liệu books(Bản đồ tạm thời bộ nhớ) và dataManager để private; Bên ngoài chỉ tương tác qua các phương thức public như createBook, updateBook, searchBooks,… → kiểm soát trạng thái và tính nhất quán.

- Composition & Delegation (Kết hợp và ủy quyền)

Lớp kết hợp DataManager và ủy quyền lưu/đọc dữ liệu bền vững (saveBook, deleteBook, findAllBooks, …).

Bản thân BookService giữ Map<id, Book> để thao tác nhanh, còn lưu trữ thời hạn do DataManager đảm nhiệm.

- Constructor Tiêm (Tiêm phụ thuộc)

Nhận DataManager qua hàm tạo → phân tách phụ thuộc, dễ mô phỏng/kiểm tra, phù hợp nguyên lý SOLID (DIP).

- Tách mối quan tâm

BookService chỉ tập trung dịch vụ sách; không xử lý giao diện người dùng, không chứa logic người dùng/giao dịch → xác định vai trò, bảo trì dễ dàng.

- Hành động gắn kết với mô hình miền

Gọi các hành vi của Book(ví dụ: addTransactionType, incrementViewCount, setter trạng thái) đúng nơi dữ liệu nằm , đảm bảo quy tắc nghiệp vụ.

- Sử dụng Stream API để thể hiện logic bộ sưu tập

Tìm kiếm–lọc–sắp sắp xếp (searchBooks) và thống kê (getBookStatsByFaculty) sử dụng mã stream/filter/sort/collecttrợ giúp ngắn gọn, dễ đọc.

- Tổng kết:

BookService có thể thực hiện OOP qua đóng gói nghiệp vụ , vật thể hóa hoạt động với sách, thành phần + ủy quyền với DataManager, và phân tách mối quan hệ rõ ràng. Thiết kế này giúp hệ thống mở rộng, kiểm tra và bảo trì dễ dàng.

5.3.3 Hướng đối tượng trong NotificationService.java

- Lớp dịch vụ (trừu tượng nghiệp vụ thông báo)

Gom mọi thao tác với thông báo (tạo, truy vấn, đếm chưa đọc, đánh dấu dấu đã đọc, xóa, phát sóng) vào một lớp duy nhất.

- Đóng gói (Đóng gói)

Nội trạng thái private: Map<String, Notification> notificationsvà DataManager. Bên ngoài chỉ tương tác qua phương thức công cộng → kiểm soát và bảo toàn tính nhất quán.

- Composition & Delegation (Kết hợp và ủy quyền)

Kết hợp DataManager để ủy quyền lưu/đọc kiên cố (saveNotification, findAllNotifications, saveAllNotifications), còn bộ nhớ tạm thời là Map để thao tác nhanh.

- Constructor Tiêm (Tiêm phụ thuộc)

Nhận DataManagerqua hàm tạo → phân tách phụ thuộc, mô phỏng/kiểm tra dễ dàng (tuân thủ DIP trong SOLID).

- Tách mối quan tâm

Chỉ xử lý logic thông báo ; không có sự kết hợp giữa người dùng/sách/giao dịch logic. Việc tạo thông báo theo từng sự kiện có các phương thức trợ giúp rõ ràng (VD: notifyReturnReminder, notifyRequestApproved, …).

- Stream API

Sử dụng stream/filter/sorted/collectđể lấy danh sách của người dùng, lọc chưa đọc, đếm chưa đọc → mã hóa ngắn gọn, dễ đọc.

- Sự kiện tên miền → Phương pháp cụ thể

Các notify*chương trình chiến tranh nghề nghiệp sang thông báo (tên, nội dung, relatedId) → hành vi đặt đúng dữ liệu/miền .

- Tổng kết:

NotificationService có thể thực hiện OOP qua đóng gói , vật tượng hóa nghiệp vụ , thành phần + ủy quyền với DataManager, và phân tách mối quan hệ rõ ràng. Thiết kế này giúp quản lý thông tin mạch lạc, dễ dàng mở rộng và kiểm tra dễ dàng.

5.3.4 Hướng đối tượng trong ReportService.java

- Lớp dịch vụ (Abstraction nghiệp vụ báo cáo)

Gom toàn bộ thao tác với báo cáo vi phạm: tạo, cập nhật trạng thái, truy vấn theo trạng thái/người dùng, lấy tất cả, lưu hàng loạt.

- Đóng gói (Đóng gói)

Nội trạng thái private: Map<String, Report> reports và DataManager. Bên ngoài chỉ tương tác qua phương thức public → kiểm soát dữ liệu và tính nhất quán.

- Composition & Delegation (Kết hợp và ủy quyền)

Kết hợp DataManager để ủy quyền lưu/đọc kiên cố (saveReport, findAllReports, saveAllReports), nhưng Map giữ dữ liệu trong bộ nhớ để thao tác nhanh hơn.

- Constructor Tiêm (Tiêm phụ thuộc)
Nhận DataManager qua hàm tạo → mô phỏng/kiểm tra dễ dàng, phân tách phụ thuộc (tuân thủ DIP trong SOLID).

- Separation of Concerns
Chỉ xử lý báo cáo logic (không kết hợp người dùng/sách/giao dịch). Cập nhật trạng thái đính kèm dấu thời gian resolvedAt khi RESOLVED/REJECTEDcó chỗ trống logic .

- Stream API
Sử stream/filter/sorted/collectdụng các truy vấn: getPendingReports(), getReportsByUser(...)→ code ngắn gọn, rõ ràng.

- Tổng kết:
ReportService có thể thực hiện OOP qua đóng gói nghiệp vụ , vật thể hóa hoạt động báo cáo, thành phần + ủy quyền với DataManager, và phân tách mối quan hệ tâm mạch lạc. Thiết kế trợ giúp quản lý vòng đời báo cáo rõ ràng, dễ dàng mở rộng và kiểm tra.

5.3.5 Hướng đối tượng trong TransactionService.java

- Class dịch vụ (Abstraction nghiệp vụ giao dịch)
Gói toàn bộ thao tác với giao dịch: tạo yêu cầu, duyệt/từ chối, giao sách, thanh toán, giới hạn, abort, đánh giá, truy vấn lịch sử/quá hạn.

- Đóng gói (Đóng gói)
Nội bộ trạng thái private: Map<String, Transaction> transactions, cùng BookService, DataManager; bên ngoài chỉ sử dụng phương thức công khai → kiểm soát và ít nhất quán dữ liệu.

- Composition & Delegation (Kết hợp và quyền công việc)
Phối hợp BookServiceđể cập nhật trạng thái (SẴN CÓ/ĐẶT TRƯỚC/Mượn/BÁN) và DataManager để lưu/đọc chắc chắn → lớp này điều phối quy trình giao dịch.

- Constructor Tiêm (Tiêm phụ thuộc)
Nhận BookService, DataManagerqua constructor → dễ dàng mock/test, đúng tinh thần DIP (SOLID).

- Tách mối quan tâm
Chỉ xử lý logic giao dịch ; dữ liệu, lưu trữ, thông báo… thuộc lớp khác → vai trò rõ ràng, dễ bảo trì.

- Quy tắc nghiệp vụ & trạng thái miền
Kiểm tra điều kiện trước hành động (sách phải CÓ SẴN, không mượn sách của mình, chỉ VAY mới được trả/gia hạn, xếp hạng 1–5, đánh giá chỉ khi HOÀN THÀNH…).
Sử dụng TransactionStatus và BookStatus để quản lý vòng đời : ĐANG CHỜ → ĐANG PHÊ DUYỆT/BỊ TỪ CHỐI → ĐANG TIẾN HÀNH → HOÀN THÀNH/HỦY; đồng bộ trạng thái tương ứng.

- Stream API cho truy vấn

để stream/filter/sorted/collect lấy lịch sử, danh sách mượn/cho vay, yêu cầu chờ duyệt, và giao dịch quá hạn → mã hóa, dễ đọc.

- Tổng kết:

TransactionService có thể thực hiện OOP qua đóng gói nghiệp vụ , vật thể hóa quy trình , và thành phần + ủy quyền để điều phối nhiều đối tượng. Quản lý chặt trạng thái giao dịch/sách giúp hệ thống nhất quán, dễ dàng mở rộng và kiểm tra.

5.3.6 Hướng đối tượng trong UserService.java

- Lớp dịch vụ (trừu tượng nghiệp vụ người dùng)

Gom toàn bộ thao tác với người dùng: đăng ký/đăng nhập/đăng xuất, cập nhật hồ sơ, đổi & đặt lại mật khẩu, cập nhật org TrustScore, tìm kiếm, khóa/mở khóa, lưu toàn bộ.

- Đóng gói (Đóng gói)

Nội bộ trạng thái private: Map<String, User> users, User currentUser, DataManager. Bên ngoài chỉ gọi qua phương thức public → kiểm soát dữ liệu & tính nhất quán.

- Composition & Delegation

Kết hợp DataManager để ủy quyền lưu/đọc kiên cố (save/findAll/saveAll). Bộ nhớ tạm thời là Map để thao tác nhanh hơn.

- Constructor Tiêm (Tiêm phụ thuộc)

Nhận DataManager qua hàm tạo → mô phỏng/kiểm tra dễ dàng, sơn thủ DIP (SOLID).

- Tách mối quan tâm Người dùng logic

xử lý chỉ ; không có hỗn hợp logic/giao dịch. Tạo sẵn tài khoản quản trị viên nếu chưa có (khởi tạo hạt giống).

- Quy tắc nghiệp vụ & kiểm soát quyền truy cập

Đăng ký: email phải @dainam.edu.vn, kiểm tra email trùng lặp/mã SV.

Đăng nhập: kiểm tra tồn tại, mật khẩu, trạng thái isActive.

Khóa/mở khóa: ADMIN chỉ (dựa vào currentUser).

updateTrustScore: tính điểm mới theo công thức trung bình (giới hạn [0..5]).

resetPassword: sinh mật khẩu tạm thời DNUxxxxxx.

- Stream API

Dùng stream/filter/collect để tìm kiếm, kiểm tra trùng lặp, tải dữ liệu → mã hóa, dễ đọc.

- Tổng kết:

UserService có thể hiện OOP qua đóng gói , người dùng vật thể hóa nghiệp vụ , thành phần + ủy quyền với DataManager, và phân tách mối quan hệ rõ ràng. Thiết kế trợ giúp quản lý tài khoản mạch lạc, dễ dàng mở rộng và kiểm tra.

5.4 BookshareApplication.java

- Composition Root (điểm ghép hệ thống)

Lớp BookShareApplicationlà nơi khởi tạo và "ghép kết nối" toàn bộ đối tượng: DataManager, UserService, BookService, TransactionService, ReportService, NotificationService, AdminService. Đây là Composition mẫu ở cấp ứng dụng.

- Constructor/Dependency Insert (tiêm phụ thuộc thủ công)

Các dịch vụ nhận phụ thuộc qua hàm tạo (ví dụ: TransactionService(bookService, dataManager)) → giảm kết nối Gmail, dễ test/mock, đúng tinh thần DIP (SOLID) .

- Tách mối quan tâm (phân chia mối quan tâm)

Mỗi dịch vụ đảm nhận một nhiệm vụ : Người dùng, Sách, Giao dịch, Báo cáo, Thông báo, Quản trị viên. Main chỉ điều phối demo luồng, không trộn logic nghiệp vụ.

- Encapsulation (đóng gói)

Các trường dịch vụ là private static và truy cập chỉ qua getter ; chi tiết phát triển khai (lưu trữ, xử lý) được ẩn giấu trong từng dịch vụ.

- Orchestration (điều phối nghiệp vụ)

Phương thức demoApplication() mô phỏng quy trình end-to-end : đăng ký/đăng nhập → đăng sách → tìm kiếm → tạo & duyệt yêu cầu → giao/nhận sách → thông báo → đánh giá → báo cáo → thống kê/sao lưu. Code có thể thực hiện các tác vụ cộng đối tượng theo dịch vụ luồng.

- Domain Models + Services

Các model (User, Book, Transaction, Report, Notification) giữ trạng thái + hành vi nhỏ ; các dịch vụ chứa logic nghiệp vụ cao hơn, gọi chéo và cập nhật trạng thái nhất quán.

- Không sử dụng kế thừa/đa hình tại lớp Main

File này để tạo và chạy bản script khởi động ; tính kế thừa/đa hình (nếu có) nằm ở tầng miền/dịch vụ (hiện tại dự án chủ yếu dựa trên thành phần).

- Kết luận:

BookShareApplication đóng vai trò thành phần gốc : tiêm phụ thuộc, ghép các dịch vụ và điều phối luồng nghiệp vụ để minh họa cách hợp tác OOP tượng. Cung cấp gói , tiêm phụ thuộc và phân chia mối quan hệ , kiến trúc rõ ràng, dễ mở rộng và kiểm tra dễ dàng.

5.5 Main

5.5.1 Index

- Kiến trúc tổng

Một trang (Single Page) tĩnh HTML + CSS + JavaScript (không framework).

UI phân khu : Header, thanh tìm kiếm + bộ lọc, Sidebar, lưới sách (Nội dung), Modal (đăng nhập/đăng ký/đăng sách/chi tiết sách).

- Data & archive

Sử dụng localStorage làm "giả phụ trợ" cho users, books, transactions, notifications.

Biến toàn cục currentUser + các mảng dữ liệu; chức năng saveData() đồng bộ về localStorage.

- Tương tác & điều hướng

Các hành động dựa trên sự kiện (theo hướng sự kiện): nhấp vào nút, gửi biểu mẫu, chọn bộ lọc, phím tắt.

Hàm showSection(section) thay đổi nội dung khu vực nội dung theo thanh bên menu (Trang chủ/MyBooks/Favorites/Transactions/…).

- Chức năng chính đã có

Auth : handleLogin, handleRegister, handleLogout, cập nhật UI sau khi đăng nhập.

Sách : hiển thị mạng (displayBooks), tìm kiếm/lọc/sắp xếp (searchBooks), xem chi tiết (showBookDetail), đăng mới (handleAddBook), yêu thích (toggleFavorite), xóa/sửa (placeholder).

Giao dịch : tạo yêu cầu vay/mua (người mua), phê duyệt/từ chối (chủ sở hữu), danh sách vay/đã giao dịch, thanh toán.

Thông báo : tạo khi có sự kiện (yêu cầu, duyệt, trả tiền), danh sách thông báo, đánh dấu đã đọc.

Trang cá nhân : Thống kê nhỏ (sách đã đăng, giao dịch, TrustScore).

Quản trị viên : thống kê tổng quan, danh sách người dùng/sách, khóa người dùng (cờ isBlocked), ẩn/hiện, xóa sách.

- Giao diện người dùng/Trải nghiệm người dùng

CSS tĩnh, tông hiện đại (màu chính/phụ), bố cục dạng lưới cho thẻ sách, trạng thái huy hiệu , thanh trượt phương thức.

Cơ sở đáp ứng cho màn hình nhỏ (truy vấn phương tiện).

- Tổng kết:

Đây là một SPA nhỏ gọn JS mô phỏng đầy đủ luồng BookShare: đăng ký/đăng nhập → đăng/tìm sách → yêu cầu/duyệt giao dịch → thông báo → quản trị viên. Mọi thứ chạy phía máy khách với localStorage, dễ demo, dễ mở rộng về sau sang backend Java real (thay localStorage bằng API).

CHƯƠNG 6: KẾT QUẢ NGHIÊN CỨU VÀ ĐÁNH GIÁ

6.1 Kết quả đạt được

Xây dựng thành công hệ thống DNU BookShare - Nền tảng chia sẻ sách sinh viên với đầy đủ chức năng:

Các module chính đã hoàn thành:

User Management: Đăng ký/đăng nhập, phân quyền STUDENT/ADMIN, quản lý hồ sơ, trust score system

Book Management: Đăng sách, tìm kiếm đa điều kiện, 3 hình thức (mượn/bán/trao đổi), favorite system

Transaction Management: Workflow giao dịch đầy đủ (PENDING → APPROVED → IN_PROGRESS → COMPLETED), đánh giá sau giao dịch

Notification System: 9 loại thông báo real-time, badge hiển thị số thông báo chưa đọc

Report & Admin: 6 loại báo cáo vi phạm, admin dashboard với thống kê, tự động kiểm tra sách quá hạn

Application chạy hoàn chỉnh với đầy đủ tính năng từ đăng ký → giao dịch → thống kê

6.2 Kiến thức đạt được

6.2.1 Nguyên lý OOP (4 trụ cột)

- Encapsulation (Đóng gói) - 100%

+ Tất cả thuộc tính private, truy cập qua getters/setters
+ Bảo vệ tính toàn vẹn dữ liệu

- Inheritance (Kế thừa) - 60%

+ Hiểu lý thuyết nhưng chưa áp dụng (dự án không có nhu cầu bắt buộc)
+ Có thể cải tiến với BaseEntity abstract class

- Polymorphism (Đa hình) - 70%

Sử dụng enum với methods
Chưa có interface polymorphism (có thể cải tiến)

- Abstraction (Trừu tượng) - 85%

+ Tách biệt Model - Service - Persistence rõ ràng
+ Ẩn chi tiết implementation, cung cấp interface đơn giản

6.3 Tự đánh giá

6.3.1 Điểm mạnh

Về kỹ thuật:

Áp dụng Encapsulation nhất quán, code clean và organized

Thiết kế workflow giao dịch logic và hoàn chỉnh

Exception handling tốt với validation đầy đủ

Tách biệt responsibilities rõ ràng (Model/Service/Persistence)

Về sản phẩm:

Đầy đủ tính năng cho use case sinh viên chia sẻ sách

Trust score system và auto-check overdue books thông minh

UI/UX hiện đại, responsive design

Admin dashboard với thống kê chi tiết

Về thái độ:

Code có cấu trúc, dễ đọc, dễ maintain

Có documentation (README, comments)

6.3.2 Hạn chế

Về kỹ thuật OOP:

Chưa sử dụng Inheritance (không có BaseEntity abstract class)

Chưa có Interface layer cho services (khó test và mở rộng)

Chưa áp dụng Design Patterns (Singleton, Factory, Strategy)

Về Architecture(kiến Trúc):

Sử dụng JSON file thay vì database thật (không scale)

Password lưu plaintext (chưa hash với BCrypt)

Không có Unit Tests (JUnit, Mockito)

Frontend và Backend chưa tích hợp (cần REST API)

Về Security(bảo mật):

Không có authentication/authorization layer

Không có input validation chống XSS, SQL Injection

Không có logging system

6.4 Ưu điểm và nhược điểm

6.4.1 Ưu điểm

Thiết kế OOP tốt

+Encapsulation hoàn chỉnh**:** Tất cả thuộc tính private, truy cập qua methods

+Kiến trúc phân tầng**:** Model → Service → Persistence, dễ maintain

+Business logic thông minh**:** Trust score tự động, auto-check overdue books

Code Quality cao

+Clean Code**:** Naming chuẩn, methods ngắn gọn, comments đầy đủ

+Exception Handling: Validate đầy đủ, error messages rõ ràng

+Modern Java**:** Stream API, Lambda, LocalDateTime, NIO.2

Tính năng đầy đủ

+50+ use cases được implement hoàn chỉnh

+Notification system với 9 loại thông báo real-time

+Admin dashboard với thống kê, quản lý vi phạm

+UI/UX hiện đại**:** Responsive, smooth animations

6.4.2 Nhược điểm

Thiếu OOP nâng cao

+Không có Inheritance: Duplicate code (userId, bookId, createdAt...)

+Không có Interface**:** Services không implement interface → khó test

+Thiếu Design Patterns**:** Không dùng Singleton, Factory, Strategy

Architecture đơn giản

+JSON file thay vì Database → không scale, performance kém

+Không có REST API → frontend-backend không tích hợp

+Thiếu Unit Tests → không đảm bảo code quality

Security yếu (Nghiêm trọng)

+Password plaintext → security breach

+Không có JWT/Authentication → không an toàn

+Không validate input → dễ bị XSS, SQL Injection

Thiếu Monitoring & Docs

+Chỉ có System.out.println → khó debug

+Không có Javadoc → khó hiểu cho người khác

CHƯƠNG 7 KẾT LUẬN

Dự án **DNU BookShare** đã hoàn thành thành công với 100% tính năng đề ra, thể hiện sự nắm vững các kiến thức cốt lõi về **Lập trình hướng đối tượng**. Hệ thống bao gồm 12 classes với kiến trúc phân tầng rõ ràng (Model - Service - Persistence), triển khai đầy đủ các module: quản lý người dùng, sách, giao dịch, thông báo và quản trị hệ thống.

Thành tựu nổi bật:

Áp dụng tốt nguyên lý Encapsulation với tất cả thuộc tính private và getters/setters

Thiết kế business logic thông minh: trust score system, tự động kiểm tra sách quá hạn

Code quality cao: clean code, naming conventions chuẩn, structure rõ ràng

Workflow giao dịch hoàn chỉnh từ yêu cầu → chấp nhận → giao sách → trả sách → đánh giá

Hạn chế cần cải thiện:

Chưa tận dụng Inheritance và Polymorphism (thiếu interface layer)

Security yếu: password plaintext, không có authentication

Persistence đơn giản: dùng JSON file thay vì database thật

Thiếu Unit Tests để đảm bảo chất lượng code

TÀI LIỆU THAM KHẢO

1.TITV

2.W3SCHOOL

3.