

資料結構與物件導向設計

HW2  
圖書館管理系統  
REPORT

資工一A 113550193 李品翰

2025.06.08

## 閱前須知：

### 關於 users.json 中多數帳號無法登入的說明

本專案中附有 **users.json** 測試資料檔，其中：

- 大部分帳號無法登入是預期行為
- 這是因為我自行實作的雜湊函式 `sha256()` 是基於 C++ 的 `std::hash<std::string>`，而 `std::hash` 在**不同編譯器／不同作業系統平台**下，其內部實作並不保證一致
- 因此，即使密碼相同，不同平台編譯出的可執行檔所產生的雜湊值也會不同，導致登入比對失敗

這些帳號的**主要用途**，是為後續**統計圖表功能**（Ex. 借閱書籍紀錄）提供多樣測資，非作為通用登入帳號。如果沒有這些資料，測試者就必須手動新增大量使用者及借閱行為，才能產生完整統計資料，較為費時。

本系統內建一組可以跨平台登入的帳號：

- 使用者名稱：`admin`
- 密碼：`admin`

助教如需建立其他帳號進行測試，建議可使用該 `admin` 帳號登入後，透過主選單中的 **1. 建立新使用者** 功能建立。由於新增帳號的雜湊計算會在執行中的程式環境內進行，因此能確保產生與驗證雜湊值一致，可立即登入驗證，不會有編譯器差異問題。

## 閱前須知：

備註：若連 **admin / admin** 都無法登入，請依下列步驟操作

1. 將原始帳號資料檔 **data/users.json** 暫時改名為 **users\_.json**
2. 重新執行程式，系統會自動偵測找不到帳號檔案，並進入**初始化**流程
3. 請依畫面提示輸入帳號與密碼，建立新的 **admin** 帳號（可自訂）
4. 建立成功後，程式會自動儲存成新的 **users.json**
5. 最後，再**將原本的 users\_.json 改回 users.json**，即可繼續使用原有帳號測資資料

此流程不會影響後續功能，包括統計圖表、借閱紀錄等，僅用於初始化過程解鎖登入功能

註：原本想自己實作 SHA-256 雜湊，不過後來在不同電腦測試時，才發現登入密碼驗證可能會出問題。

這部分我在這兩頁補充修正方式，如果還是造成測試上的不便，請助教們見諒，感謝~

# 一、資料結構

## 主資料容器

```
std::vector<Book> books;
```

### 存放所有書籍資訊

- 動態擴充  
可隨新增書籍自動擴大容量，  
適合資料筆數不固定的情境
- 支援隨機存取 ( $O(1)$ )  
能快速取得第 N 本書，  
利於排序與顯示功能
- 記憶體連續  
加快遍歷與複製操作，提升效能

## 書籍 ID 映射表

```
std::unordered_map<int, int> bookIdMap;
```

### 將每本書的 ID 對應到 books vector 中 的索引位置，達成 $O(1)$ 的快速存取

- 編輯書籍時：  
能透過 ID 直接定位到對應書籍
- 刪除或更新時：  
快速查找目標書籍位置，避免線性搜尋

## 反向索引表

```
std::unordered_map<std::string, std::unordered_set<int>> titleIndex;  
std::unordered_map<std::string, std::unordered_set<int>> authorIndex;  
std::unordered_map<std::string, std::unordered_set<int>> categoryIndex;
```

為了支援多條件查詢與布林運算，針對常見欄位建立了關鍵字 → 書籍 ID 的對應表

- 可用於搜尋書名、作者、分類等欄位的關鍵字
- 執行 AND、OR、NOT 等集合運算時僅處理符合條件的 ID 集合，避免全書掃描
- 例如：查詢 (作者 = 王文華) 與 (分類 = 歷史)，可直接對兩組 ID 做交集運算

## 二、排序演算法

## 採用的排序演算法：快速排序 (Quick Sort)

為了在顯示書籍清單時提供依照「標題、作者、出版年份」等欄位排序的功能，採用自行實作的快速排序演算法。

程式碼位置：

```
// include/SortUtil.h + src/SortUtil.cpp
template<typename T, typename Compare>
void sort(std::vector<T>& arr, Compare comp);
```

## 核心設計：三數取中快速排序

Quick Sort 是一種 Divide and Conquer 的排序演算法。本作業採用三數取中方法，可有效減少最壞情況的發生。

運作流程：

1. 選定一個 **pivot**：取開頭、中間、結尾三個值的**中位數**
2. 將資料切分為小於 **pivot** 與大於 **pivot** 兩半。
3. 分別遞迴排序這兩半
4. 合併完成排序

## 時間複雜度分析

平均時間複雜度： $O(n \log n)$

最壞情況： $O(n^2)$  (已透過中位選取減緩)

## 實作特色

- 使用**泛型模板函式 (template)** 實作，支援物件（如 Book）依任意欄位排序
- 排序依據由 Compare 函式物件或 lambda 傳入

```
sort(books, [](const Book& a, const Book& b) {
    return a.getTitle() < b.getTitle();
});
```

## 三、搜尋演算法

本作業提供「簡單搜尋」與「多條件智慧搜尋」兩種模式，分別採用不同演算法

# 1. 簡單搜尋 (Simple Search)

- 使用方式

使用者輸入關鍵字後，系統將遍歷所有書籍，依序比對書名、作者、分類、簡介等欄位，若任一欄位包含關鍵字，則將該書加入結果清單

- 實作邏輯

```
for (auto& book : books) {  
    if (const_cast<Book&>(book).matchesKeyword(keyword))  
        results.push_back(&book);  
}
```

- 核心函式 — `SearchUtil::contains()`

```
bool SearchUtil::contains(const std::string& text, const std::string& pattern) {  
    return indexOf(text, pattern) != -1;  
}  
int SearchUtil::indexOf(const std::string& text, const std::string& pattern) {  
    for (int i = 0; i + pattern.size() <= text.size(); ++i) {  
        int j = 0;  
        while (j < pattern.size() && text[i + j] == pattern[j]) ++j;  
        if (j == pattern.size()) return i;  
    }  
    return -1;  
}
```

- 時間複雜度：

$O(N_{books} \times k_{fields} \times n_{text} \times m_{pattern})$

$N_{books}$ ：書籍總數

$k_{fields}$ ：比對欄位數（如書名、作者等）

$n_{text}$ ：欄位長度

$m_{pattern}$ ：關鍵字長度

- 優點：

實作簡單，容易維護

- 缺點：

需完整掃描所有書籍，效能較差

- 多個欄位比對皆透過此函式執行子字串搜尋
- 實作方式為 **Naive Substring Search**

## 2. 多條件智慧搜尋 (Advanced Boolean Search)

- 使用方式

使用者可輸入多欄位的條件式（例如：title:小說 AND author:王文華），系統會解析並執行布林邏輯運算以找出符合條件的書籍。

- 實作邏輯

- 透過 QueryParser 建立查詢運算樹（包含 AND / OR / NOT 節點）
- 對每個欄位查詢節點，從對應的反向索引表中取出符合條件的書籍 ID 集合

- 時間複雜度：

$O(k)$   $k$ ：符合條件的書籍數量，通常遠小於  $N_{books}$

- 優點

- 查詢效能高，不需全書掃描
- 支援欄位指定與布林邏輯組合 (AND/OR/NOT)
- 適用於大量資料與複雜查詢情境

## 四、基本功能介紹

**新增書籍**

# 1. 使用者輸入 (UI) — Library.cpp

```
// src/Library.cpp
void Library::addBook() {
    if (!userManager.hasPermission(Role::Staff)) {
        ConsoleUtil::printError("權限拒絕：只有館員和管理員可以新增圖書");
        return;
    }

    BookInfo info = getBookInfoFromUser();

    Book book(0, info.title, info.author, info.year, info.copies,
              info.isbn, info.publisher, info.language, info.pageCount, info.synopsis);

    addBookCategories(book);

    if (bookManager.addBook(book)) {
        ConsoleUtil::printSuccess("新增成功，ID: " + std::to_string(book.getId()));
        bookManager.saveToFile(bookFile);
    } else {
        ConsoleUtil::printError("新增失敗");
    }
}
```

錯誤判斷

利用建構子一次設定書籍全部欄位

新增書籍前會檢查權限，  
只有館長或管理員能執行

(本系統含三種職位：館員、管理員和讀者，加分項會提及)

```
BookInfo Library::getBookInfoFromUser() {
    BookInfo info;

    std::cout << "書名: ";
    std::getline(std::cin, info.title);

    std::cout << "作者: ";
    std::getline(std::cin, info.author);

    std::cout << "出版年份: ";
    std::cin >> info.year;

    std::cout << "複本數量: ";
    std::cin >> info.copies;

    // 以下為額外欄位輸入
    info.isbn = getUserInput("ISBN");
    info.publisher = getUserInput("出版社");
    info.language = getUserInput("語言");
    // ...

    return info;
}
```

此函式用來  
輸入書本  
詳細資料

## 2. 加入資料結構與索引 — BookManager.cpp

```
// src/BookManager.cpp
bool BookManager::addBook(Book& book) {
    if (book.getId() == 0)
        book.setId(nextId++);
    else if (getBook(book.getId()) != nullptr)
        return false;

    books.push_back(book);
    bookIdMap[book.getId()] = books.size()-1;
    indexBook(book);
    return true;
}
```

### 防呆機制：

- 如果沒有設定 ID (Ex. 使用者沒填)，由系統自動給一個新 ID (且保證唯一)
- 如果有填 ID，檢查是否已經有相同的書，有的話不新增

### 新增步驟：

- 把書放進 **vector<Book>** books 裡
- 記下這本書的 ID 對應的**索引** (方便未來快速查詢)
- 更新**倒排索引** (關鍵字、年份、作者…快速搜尋用)

**依出版年份  
搜尋書籍**

## 1. 使用者輸入 (UI) — Library.cpp

```
// src/Library.cpp
std::vector<Book*> Library::searchByYear() {
    ConsoleUtil::printInfo("請輸入年份: ");
    int year;
    std::cin >> year;
    clearInputBuffer();

    ConsoleUtil::printInfo("請輸入運算符 (=, >, <, >=, <=): ");
    std::string op;
    std::getline(std::cin, op);

    std::vector<Book*> results = bookManager.filterByYear(year, op);
    SortUtil::sort(results, [](Book* a, Book* b) {
        return a->getTitle() < b->getTitle();
    });

    return results;
}
```

- 使用者輸入年份與比較運算子 (Ex. = 或 >)，並交由 **BookManager** 過濾出符合條件的書籍

## 2. 書籍篩選邏輯 — BookManager.cpp

```
// src/BookManager.cpp
std::vector<Book*> BookManager::filterByYear(int year, const std::string& op) const {
    std::vector<Book*> results;

    for (auto& book : books) {
        if (const_cast<Book&>(book).matchesYear(year, op)) {
            results.push_back(const_cast<Book*>(&book));
        }
    }
    return results;
}
```

- 遍歷整個書籍清單 **books**，針對每本書呼叫 **matchesYear()** 進行判斷，符合者加到 **results**

## 3. 條件判斷邏輯 — Book.cpp

```
// src/Book.cpp
bool Book::matchesYear(int y, const std::string& op) const {
    if (op == "=") return year == y;
    else if (op == ">") return year > y;
    else if (op == "<") return year < y;
    else if (op == ">=") return year >= y;
    else if (op == "<=") return year <= y;
    else return false;
}
```

- 判斷書籍出版年份是否符合使用者輸入的條件

**借閱書籍**

# 1. 使用者輸入 (UI) — Library.cpp

```
// src/Library.cpp
void Library::borrowBook() {
    auto availableBooks = getAvailableBooks();

    if (availableBooks.empty()) {
        ConsoleUtil::printTitle("借閱圖書");
        ConsoleUtil::printWarning("目前沒有可借閱的圖書");
        ConsoleUtil::pauseAndWait();
        return;
    }

    ConsoleUtil::printTitleWithSubtitle("借閱圖書", "可借閱的圖書");
    displayAvailableBooks(availableBooks);

    int bookId = getBookIdChoice("請輸入圖書 ID");
    std::string username = getBorrowerUsername();

    if (loanManager.borrowBook(username, bookId)) {
        ConsoleUtil::printSuccess("圖書借閱成功！");
    } else {
        ConsoleUtil::printError("圖書借閱失敗，請檢查圖書 ID 和使用者名稱");
    }

    ConsoleUtil::pauseAndWait();
}
```

如果沒有書可以借，直接結束流程

要求使用者輸入帳號  
(用於後續提及的加分項：借閱歷史、計算逾期與罰款)

錯誤判斷

## 2. 借書紀錄處理邏輯 — LoanManager.cpp

```
// src/LoanManager.cpp
bool LoanManager::borrowBook(const std::string& username, int bookId, int graceDays) {
    time_t now = time(nullptr);
    time_t dueDate = now + (14 * 24 * 60 * 60); // 用於加分項：計算逾期與罰款

    LoanRecord loan(username, bookId, now, dueDate, graceDays); // 建立借書紀錄物件
    // (包含使用者、書籍ID、時間、寬限期)

    loans.push_back(loan);
    bookLoans[bookId].push_back(&loans.back());
    userLoans[username].push_back(&loans.back()); // 加入資料結構中
    // (順便紀錄借閱紀錄)

    return true;
}
```

用於加分項：計算逾期與罰款  
建立借書紀錄物件  
(包含使用者、書籍ID、時間、寬限期)

加入資料結構中  
(順便紀錄借閱紀錄)

歸還書籍

## 1. 使用者輸入 (U I) — Library.cpp

```
// src/Library.cpp
void Library::returnBook() {
    std::string targetUser = getTargetUserForReturn();
    auto activeLoans = getActiveLoansForUser(targetUser);

    if (activeLoans.empty()) {
        showNoActiveLoansMessage(targetUser);
        return;
    }

    displayActiveLoans(activeLoans, targetUser);
```

查詢此人借過哪些書

```
    int bookId = getBookIdChoice("請輸入要歸還的圖書 ID");

    if (loanManager.returnBook(targetUser, bookId)) {
        ConsoleUtil::printSuccess("圖書歸還成功！");
        showFineIfAny(targetUser, bookId);
    } else {
        ConsoleUtil::printError("圖書歸還失敗");
    }
```

錯誤判斷：  
如果沒有任何可還書，則輸出提示

```
    ConsoleUtil::pauseAndWait();
}
```

顯示借閱中的書籍

顯示逾期歸還罰款（加分項，後續詳細說明）

錯誤判斷

## 2. 歸還紀錄處理邏輯 — LoanManager.cpp

```
// src/LoanManager.cpp
bool LoanManager::returnBook(const std::string& username, int bookId) {
    LoanRecord* loan = nullptr;

    for (auto* record : userLoans[username]) {
        if (record->getBookId() == bookId && !record->isReturned()) {
            loan = record;
            break;
        }
    }

    if (!loan) return false;

    loan->setReturnDate(time(nullptr));
    return true;
}
```

如果使用者沒借這本書，提示歸還失敗

標記這筆借閱紀錄的還書時間  
(加分項：逾期歸還罰款)

遍歷該使用者目前的借書紀錄，  
找出第一筆「書籍 ID 相符」且「尚未歸還」的紀錄，。  
找到後馬上跳出迴圈，只處理一筆紀錄（避免同書多次還書錯誤）

# 書籍列表

## 列出所有書籍 — Library.cpp

```
// src/Library.cpp
void Library::viewBookList() {
    auto allBooks = bookManager.getAllBooks();

    if (allBooks.empty()) {
        ConsoleUtil::printWarning("目前沒有任何圖書");
        return;
    }

    const int booksPerPage = 20;
    int currentPage = 1;
    // 0=NONE, 1=TITLE, 2=AUTHOR, 3=YEAR, 4=PAGES
    int currentSortField = 0;
    // 0=ASC, 1=DESC
    int currentSortOrder = 0;

    while (true) {
        auto sortedBooks = allBooks;
        applySorting(sortedBooks, currentSortField, currentSortOrder);

        int totalPages = (sortedBooks.size() + booksPerPage - 1) / booksPerPage;
        displayBookPage(sortedBooks, currentPage, booksPerPage, currentSortField, currentSortOrder);

        // 互動式選單略 (上一頁、下一頁、排序選項等)
        if (!handleBookListNavigation(...)) break;
    }
}
```

錯誤判斷

設定每頁顯示最多 20 本書  
初始畫面顯示第 1 頁 (加分項)

預設排序欄位：書名  
預設升序 (從 A 到 Z)  
(其餘選項為加分排序項)

UI 介面設計 (加分項)

**退出系統**

## 推出系統 — UIController.cpp

```
// src/UIController.cpp
bool UIController::handleLogoutChoice(int choice, int logoutOption, int exitOption) {
    if (choice == logoutOption) {
        ConsoleUtil::printSuccess("已登出");
        return false;
    } else if (choice == exitOption) {
        ConsoleUtil::printSuccess("資料已儲存，系統即將退出");
        ConsoleUtil::pauseAndWait();
        exit(0);
    }
    return true;
}
```

- 印出結束訊息
- 呼叫 **pauseAndWait()** 停住畫面，等待使用者按 Enter
- 使用 **exit(0)** 強制結束整個程式

## 五、加分功能介紹

# 帳號角色

# 功能概述

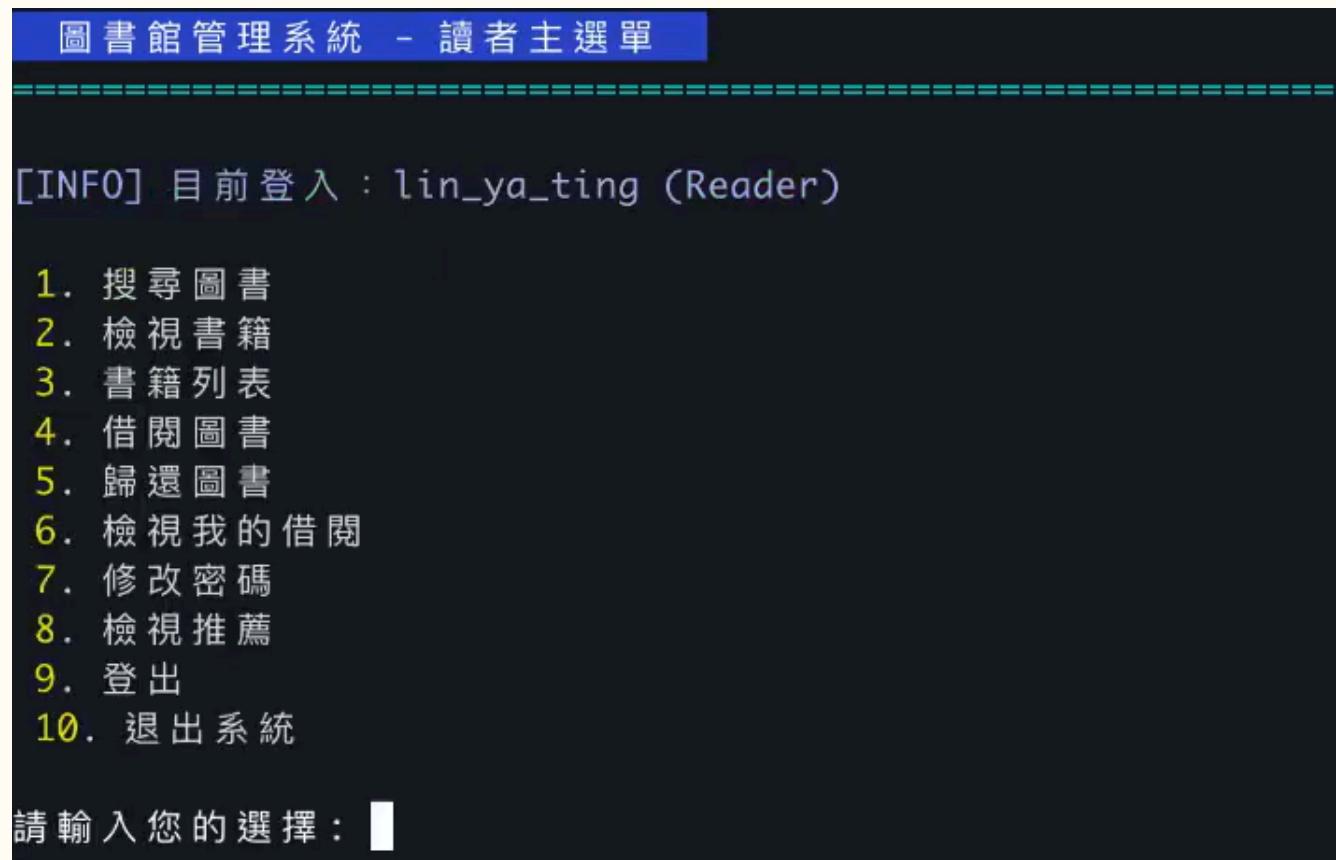
系統預設**三種身份**來分權限使用不同功能：

- **Admin**：館長，可新增帳號、修改罰則、管理書籍。
- **Staff**：管理員，可管理借還書、查詢資料。
- **Reader**：一般讀者，只能借還書、修改個人密碼與查詢資料。

這個分權設計讓使用者進入系統後，會看到不同的主選單項目，避免不必要的功能暴露給一般用戶。舉例來說，若為 Admin，會看到：



若為 Reader：



# 核心程式碼與解釋

## 1. 角色定義 — User.h

```
enum class Role {  
    Admin,  
    Staff,  
    Reader  
};
```

定義了三種角色，  
使用 **enum class** 實作，  
確保型別安全

## 2. 角色定義 — User.cpp

```
std::string roleToString(Role role) {  
    switch (role) {  
        case Role::Admin: return "Admin";  
        case Role::Staff: return "Staff";  
        case Role::Reader: return "Reader";  
        default: return "Unknown";  
    }  
}
```

用於 UI 顯示與 debug 訊息中

Ex. [INFO] 目前登入：admin (Admin)

## 3. 動態主選單 — UIController.cpp

```
void UIController::mainMenu() {  
    switch (currentUser.role) {  
        case Role::Admin:  
            displayAdminMenu();  
            break;  
        case Role::Staff:  
            displayStaffMenu();  
            break;  
        case Role::Reader:  
            displayReaderMenu();  
            break;  
    }  
}
```

每種角色會呼叫不同的  
**displayXMenu()** 來顯示  
對應功能

## 4. 帳號資訊 — users.json

```
{  
    "username": "admin",  
    "passwordHash": "QJ5qVFrj001Hs5oK:7db54fe11276cf82",  
    "role": "Admin"  
}
```

帳號資訊儲存在 **users.json**，帳號資料包含：

- **username**：使用者名稱
- **passwordHash**：加密後密碼
- **role**：對應的角色字串

# 帳號密碼管理

# 功能概述

## 使用者登入

- 開啟系統後，會提示使用者輸入帳號與密碼
- 帳號密碼正確後，才可成功登入系統
- 登入成功會顯示使用者名稱與角色，例如：

使用者名稱: admin

密碼:

[OK] 登入成功，您的角色: Admin

## 修改密碼

- 登入後可選擇主選單中的「修改密碼」功能
- 系統會提示輸入「舊密碼」與「新密碼」
- 若舊密碼驗證成功，則會更新密碼，並顯示成功訊息

當前密碼:

新密碼:

確認新密碼:

[OK] 密碼修改成功

[INFO] 按 Enter 繼續...

## 登出系統

- 使用者可透過主選單中的「14. 登出系統」項目回到登入畫面
- 登出時，系統會清除目前的登入狀態 `currentUser`

註：使用者在輸入密碼時畫面會隱藏輸入內容（使用終端機關閉 echo）

## 功能概述

### 新增使用者

- 僅限 Admin 能進入
- 系統會要求輸入：帳號、密碼、使用者角色  
(帳號不可重複)
- 建立新使用者並儲存至 users.json

```
使用者名稱 : new_user
密碼 :
確認密碼 :
[INFO] 選擇角色 ( 1=館員 , 2=讀者 ) :
2
[OK] 使用者 new_user 成功新增
[INFO] 按 Enter 繼續 ...
```

### 第一次啟動系統 (edge case)

- 若偵測到 users.json 為空，系統會啟動初始化模式，建立第一位使用者帳號
- 第一個帳號會被自動指定為 Admin 角色，用來初始化系統

```
首次設定 - 創建管理員帳號 :
管理員使用者名稱: admin
管理員密碼:
確認密碼:
管理員帳號創建成功 !
請重新啟動程式以登入。
```

## 核心程式碼與解釋

### 1. 登入驗證 — UserManager.cpp

```
bool UserManager::login(const std::string& username, const std::string& password) {
    auto it = users.find(username);
    if (it == users.end()) return false;

    if (it->second.checkPassword(password)) {
        currentUser = &(it->second);
        return true;
    }

    return false;
}
```

- 先從 **users** map 裡查找帳號
- 呼叫 **checkPassword()** 驗證密碼是否正確
- 登入成功會設定 **currentUser** 指向該帳號資料

## 核心程式碼與解釋

### 2. 密碼驗證 — User.cpp & PasswordUtil.cpp

```
// User.cpp
bool User::checkPassword(const std::string& password) const {
    return PasswordUtil::verifyPassword(password, passwordHash);
}

// PasswordUtil.cpp
bool verifyPassword(const std::string& password, const std::string& storedHash) {
    size_t colonPos = storedHash.find(':');
    if (colonPos == std::string::npos) return false;

    std::string salt = storedHash.substr(0, colonPos);
    std::string hash = storedHash.substr(colonPos + 1);

    std::string computedHash = sha256(salt + password);
    return computedHash == hash;
}
```

- 密碼驗證會從 **SALT:HASH** 字串中分離 salt
- 再用相同 salt 對使用者輸入密碼計算 **std::hash** 雜湊值
- 若和原始 hash 相符，即驗證成功

## 核心程式碼與解釋

### 3. 修改密碼 — UserManager.cpp

```
bool UserManager::changePassword(const std::string& username,
                                  const std::string& oldPassword,
                                  const std::string& newPassword) {
    User* user = findUser(username);
    if (!user) return false;

    if (!user->checkPassword(oldPassword)) return false;

    user->setNewPassword(newPassword);
    return true;
}
```

- 使用者輸入「帳號 + 舊密碼 + 新密碼」
- 先透過 **findUser()** 找到對應帳號
- 呼叫 **checkPassword()** 驗證舊密碼是否正確
- 呼叫 **setNewPassword()** 時，  
會使用**雜湊 + 加鹽**方式處理  
(詳細在下頁說明)

### 4. 登出系統 — UIController.cpp

```
void UIController::logout() {
    currentUser = nullptr;
    std::cout << "[INFO] 您已成功登出，返回登入畫面。\\n";
}
```

登出時將 **currentUser** 設為 **nullptr**，  
代表目前無人登入。

# 核心程式碼與解釋

## 5. 密碼雜湊處理 — PasswordUtil.cpp

```
std::string hashPassword(const std::string& password) {
    std::string salt = generateSalt();
    std::string saltedPassword = salt + password;
    std::string hash = sha256(saltedPassword);
    return salt + ":" + hash; // Format: SALT:HASH
}
```

- 使用 **generateSalt()** 產生隨機字串（避免 hash 重複）
- 將 salt 加在密碼前面（形成 saltedPassword）
- 對 saltedPassword 執行 **sha256()** 雜湊
- 回傳字串格式為 salt:hash

```
std::string generateSalt(size_t length = 16) {
    static const char alphanum[] =
        "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(0, sizeof(alphanum) - 2);

    std::string salt;
    for (size_t i = 0; i < length; ++i)
        salt += alphanum[dis(gen)];
}

return salt;
}
```

### generateSalt() — 用亂數字元產生 salt

- 每次雜湊的密碼都會搭配不同 salt →  
即使兩人密碼一樣，hash 值也不同
- 抗 **rainbow table** 攻擊
- 預設長度為 16 字元

```
std::string sha256(const std::string& str) {
    std::hash<std::string> hasher;
    size_t hash = hasher(str);
    std::stringstream ss;
    ss << std::hex << std::setw(16) << std::setfill('0') << hash;
    return ss.str();
}
```

### PasswordUtil.cpp — sha256 實作

- 用 **std::hash<string>** 來模擬 SHA-256 雜湊
- 搭配 salt 一起使用可提升安全性

# 核心程式碼與解釋

## 6. 帳號資料儲存與讀寫 — UserManager.cpp

```
void UserManager::loadFromFile(const std::string& filename) {
    std::ifstream file(filename);
    if (!file) return;

    JSONValue json = SimpleJSON::parse(file);
    for (const auto& [username, info] : json.asObject()) {
        std::string hash = info["passwordHash"].asString();
        std::string role = info["role"].asString();
        users[username] = User(username, hash, stringToRole(role));
    }
}
```

```
void UserManager::saveToFile(const std::string& filename) {
    JSONObject json;
    for (const auto& [username, user] : users) {
        JSONObject info;
        info["passwordHash"] = user.getPasswordHash();
        info["role"] = roleToString(user.getRole());
        json[username] = info;
    }

    std::ofstream file(filename);
    file << JSONValue(json).stringify(true); // pretty print
}
```

### 讀取帳號資料 — loadFromFile()

- 開啟指定的 JSON 檔案（預設為 **users.json**）
- 將其中每一筆帳號資訊轉成 User 物件儲存在 **users map** 中

### 儲存帳號資料 — saveToFile()

- 把所有帳號資料寫入 JSON 檔中
- 每次**新增使用者、修改密碼**後都會呼叫這個函式，更新檔案內容

註：帳號儲存格式 **users.json** 已在前面「帳號角色」加分功能提及，這裡就省略不提

## 核心程式碼與解釋

### 7. 新增使用者 — UserManager.cpp

```
bool UserManager::addUser(const std::string& username,
                           const std::string& password,
                           Role role) {
    if (users.find(username) != users.end()) return false;

    std::string passwordHash = PasswordUtil::hashPassword(password);
    users[username] = User(username, passwordHash, role);
    saveToFile(filename);

    return true;
}
```

- 檢查 **username** 是否已存在（帳號不重複）
- 呼叫 **hashPassword()** 處理密碼
- 建立新 **User** 並加入 **users** map
- 最後呼叫 **saveToFile()** 寫入 JSON

### 8. 第一次啟動系統 — UIController.cpp

```
void UIController::setupAdminAccount() {
    std::cout << "[首次啟動] 請先建立系統管理員帳號\n";
    // 輸入帳號/密碼 (略)
    userManager.addUser(username, password, Role::Admin);
}
```

**setupAdminAccount()**

只會在系統偵測到 **users.json** 為空時觸發

**額外書籍欄位**

# 功能概述

為提升書籍資料完整性，額外加入以下**六個欄位**。

## 核心程式碼與解釋

### 1. ISBN 編號

```
// Book.h
std::string isbn;
std::string getIsbn() const;
void setIsbn(const std::string& isbn);

// BookManager.cpp
if (bookJsonPtr->contains("isbn"))
    book.setIsbn(bookJsonPtr->at("isbn")->getString());
bookJson->set("isbn", book.getIsbn());
```

### 2. 出版社 Publisher

```
// Book.h
std::string publisher;
std::string getPublisher() const;
void setPublisher(const std::string& publisher);

// BookManager.cpp
if (bookJsonPtr->contains("publisher"))
    book.setPublisher(bookJsonPtr->at("publisher")->getString());
bookJson->set("publisher", book.getPublisher());
```

### 3. 書籍語言 Language

```
// Book.h
std::string language;
std::string getLanguage() const;
void setLanguage(const std::string& language);

// BookManager.cpp
if (bookJsonPtr->contains("language"))
    book.setLanguage(bookJsonPtr->at("language")->getString());
bookJson->set("language", book.getLanguage());
```

### 4. 頁數 Page Count

```
// Book.h
int pageCount;
int getPageCount() const;
void setPageCount(int pageCount);

// BookManager.cpp
if (bookJsonPtr->contains("pageCount"))
    book.setPageCount(bookJsonPtr->at("pageCount")->getInt());
bookJson->set("pageCount", book.getPageCount());
```

# 核心程式碼與解釋

# 各項欄位說明

## 5. 書籍簡介 Synopsis

```
// Book.h
std::string synopsis;
std::string getSynopsis() const;
void setSynopsis(const std::string& synopsis);

// BookManager.cpp
if (bookJsonPtr->contains("synopsis"))
    book.setSynopsis(bookJsonPtr->at("synopsis")->getString());
bookJson->set("synopsis", book.getSynopsis());
```

## 6. 書籍分類 Categories

```
// Book.h
std::vector<std::string> categories;
void addCategory(const std::string& category);
const std::vector<std::string>& getCategories() const;

// BookManager.cpp
if (bookJsonPtr->contains("categories")) {
    for (auto& c : bookJsonPtr->at("categories")->getArray())
        book.addCategory(c->getString();
}
auto arr = JSONValue::createArray();
for (auto& c : book.getCategories())
    arr->push_back(c);
bookJson->set("categories", arr);
```

### ISBN

- 作為書籍識別碼
- 方便查詢書籍來源

### Publisher

- 紀錄出版單位名稱
- 可篩選同一出版社出版的書籍

### Language

- 指明書籍的語言（如 zh-TW、en）
- 方便針對語言分類

### Page Count

- 記錄書籍總頁數
- 用於頁數統計或篩選

### Synopsis

- 提供內容概要
- 讓讀者可快速了解書籍內容

### Categories

- 記錄書籍類型（如小說、漫畫、科普）
- 支援單本多分類，方便進行分類瀏覽與搜尋

**編輯書籍資料**

## 功能概述

此功能允許館長或管理員針對既有書籍進行內容修改。

使用者須先輸入欲修改書籍的 ID，可修改的資料包括：

- 書名、作者、出版年
- 頁數、ISBN、出版社、語言
- 書籍簡介、分類（可多選）

使用者按下 Enter 可保留原本欄位，或輸入新值覆蓋。修改保存後系統即時更新檔案。

範例：

1. 書名（目前：台北愛情故事）
2. 作者（目前：王文華）
3. 年份（目前：2023）
4. ISBN（目前：978-986-123-456-7）
5. 出版社（目前：聯經出版）
6. 語言（目前：繁體中文）
7. 頁數（目前：320）
8. 複本數量（目前：5）
9. 摘要
10. 管理分類
11. 保存更改並退出
12. 取消編輯

請輸入您的選擇：9

新摘要：這本書很好看，雖然我沒看過 ■

1. 書名（目前：台北愛情故事）
2. 作者（目前：王文華）
3. 年份（目前：2023）
4. ISBN（目前：978-986-123-456-7）
5. 出版社（目前：聯經出版）
6. 語言（目前：繁體中文）
7. 頁數（目前：320）
8. 複本數量（目前：5）
9. 摘要
10. 管理分類
11. 保存更改並退出
12. 取消編輯

請輸入您的選擇：11

[OK] 圖書資訊更新成功

[INFO] 按 Enter 繼續...

注意：

修改完資料後，

要記得輸入「11. 保存修改並退出」

才會更新檔案

## 核心程式碼與解釋

實作程式碼所在檔案 — **Library.cpp**

**1. 進入編輯模式 — `editBook()`** 約第 1030 行

使用者輸入欲編輯的書籍 ID，若存在即顯示書籍資訊並複製一份資料供後續修改

**2. 進入編輯主選單 — `runEditMenu() + createEditOptions()`** 約第 1055 行、1080 行

系統顯示欄位清單 (1~12)，包含目前值，供使用者選擇欲修改項目

**3. 輸入新資料 — `editBookField()`** 約第 1097 行

選擇任一欄位後輸入新值 (或按 Enter 保留原值)，支援修改書名、作者、頁數等資訊

**4. 修改分類 — `manageBookCategories()`** 約第 1156 行

進入分類管理子選單，可新增或刪除多個分類標籤

**5. 保存結果 — `saveBookChanges()`** 約第 1205 行

選擇「11. 保存並退出」後，會更新原始書籍並寫入 **books.json** 檔案

**刪除書籍**

## 功能概述

此功能允許館長或管理員對現有書籍進行**永久刪除操作**。

使用者須先輸入欲刪除書籍的 **ID**，系統將顯示書籍完整資訊供確認，  
並需經過**兩次確認機制**才會真正執行刪除，刪除後系統會即時更新檔案。

### 範例：

[INFO]

請輸入要刪除的圖書 ID:

101

— 即將刪除的圖書資訊 —

書名：一本書

作者：我

出版年份：2025

ISBN: 123123-12412512-5dawdaw

總複本數：25 本

分類：[原創] [自傳]

[WARNING] ⚠ 警告：刪除操作無法復原！

[INFO] 確認要刪除這本書嗎？(輸入 'DELETE' 確認，其他任意鍵取消):

DELETE

[WARNING] 最後確認：您真的要刪除「一本書」嗎？(y/N):

y

[OK] 圖書「一本書」已成功刪除

[OK] 資料已成功保存

[INFO] 按 Enter 繼續 ...

## 核心程式碼與解釋

實作程式碼所在檔案 — **Library.cpp**

**1. 進入刪除模式 — `deleteBook()`** 約第 465 行

僅限館長和管理員使用，輸入書籍 ID 後檢查是否存在，若不存在無法刪除

**2. 顯示資訊與二次確認** 約第 498 行

顯示該書完整資訊供使用者確認，使用者需輸入 **DELETE** (全大寫) 確認意圖，否則取消刪除流程

**3. 最後確認 — `cin >> finalConfirmation`** 約第 558 行

再次提示輸入 **y** 或 **Y** 確認，提供**雙重防呆**機會

**4. 執行刪除 — `bookManager.deleteBook(bookId)`** 約第 572 行

從 **vector<Book>** 中移除指定書籍，並更新 ID 映射與反向索引結構

**5. 保存結果 — `saveToFile()`** 約第 576-580 行

將變更寫入 **books.json**，確保刪除操作**永久生效**

**分頁顯示書籍**

## 功能概述

在基本功能中，所有書籍會一次性顯示於同一畫面，當書籍數量增加時，畫面過於冗長。  
為了解決這個問題，我們導入分頁顯示系統，將書籍清單切分為多個頁面，  
並提供彈性操作介面，讓使用者可以快速切換與瀏覽。

本功能支援：

- 書籍列表按每頁 20 筆進行分頁顯示
- 顯示目前所在頁數與總頁數（如：Page 2 / 5）
- 提供「上一頁」「下一頁」「跳轉頁數」等導覽選項
- 分頁與排序、搜尋整合，保持一致顯示邏輯

### — 導航選項 —

1. 下一頁
2. 跳到指定頁
3. 重新排列
4. 檢視書籍詳情
5. 回到主選單

ID	書名	作者	年份	頁數	狀態
1	台北愛情故事	王文華	2023	320	可借(5)
2	日治時期的台灣社會	李安琪	2022	450	可借(4)
3	Python程式設計入門	陳志明	2024	280	可借(3)
4	台灣味家常菜	林美惠	2023	200	可借(6)
5	台灣山林攝影集	張育成	2023	350	可借(3)
6	四季台灣	劉詩涵	2022	240	可借(5)
7	台灣新創之路	黃建國	2024	300	可借(4)
8	心靈療癒之道	吳雅芳	2023	220	可借(2)
9	高效學習法	許志偉	2023	260	可借(4)
10	台灣設計美學	蔡佩君	2022	180	可借(2)
11	30分鐘健身計畫	趙明輝	2024	290	可借(1)
12	台灣黑熊的冒險	林雅琪	2023	48	可借(6)
13	永續台灣	鄭文龍	2023	330	可借(2)
14	台灣音樂之美	何佳玲	2022	210	可借(3)
15	台灣民主之路	謝志強	2024	380	可借(1)
16	親子溝通的藝術	楊淑芬	2023	250	可借(2)
17	台股投資入門	羅志華	2024	320	可借(3)
18	島嶼詩篇	曾美玲	2022	160	可借(2)
19	AI在台灣	石建宏	2024	400	可借(1)
20	台灣小鎮漫遊	馬雅雯	2023	280	可借(4)

## 核心程式碼與解釋

實作程式碼所在檔案 — **Library.cpp**

**1. 進入分頁書籍清單 — `viewBookList()`** 約第 2379 行

初始化頁碼、取得所有書籍並根據當前排序方式排序，顯示第 1 頁內容

**2. 顯示單頁內容 — `displayBookPage()`** 約第 2435 行

每頁顯示 20 筆書籍，並標示目前頁碼（如 Page 2 / 5）與每欄欄位標題，  
也負責更新畫面與換頁後的重新繪製

**3. 提供分頁操作選項 — `handleBookListNavigation()`** 約第 2538 行

使用者可選擇 **p = 上一頁**、**n = 下一頁**、**j = 跳轉頁碼**、**r = 重新排序** 等操作  
若使用者翻頁或重新排序，會自動重新顯示當前頁面

**4. 處理頁碼跳轉 — `jumpToPage(int& page, int totalPages)`** 約第 2592 行

提示使用者輸入目標頁數並**檢查合法性**，若有效則修改目前頁碼並返回主流程

# 多種排序列表

## 功能概述

在基本功能中，書籍列表僅支援單一排序方式，排序欄位與順序固定，使用彈性有限。  
本加分功能進行擴充，讓使用者能在瀏覽書籍列表時，自行選擇排序依據與排序順序。  
本功能支援：

- 書籍可依 **書名／作者／出版年份／頁數** 排序
- 點選同一欄位兩次可在 **升序／降序** 間切換
- 排序設定與分頁瀏覽整合，可即時呈現結果並重新跳至第 1 頁
- 排序結果會標記 **「▲ / ▼」** 箭頭，清楚顯示目前的排序狀態

範例：



[INFO] 顯示第 1 - 20 本圖書 (共 101 本)

ID	書名 ▲	作者	年份	頁數	狀態
25	20世紀世界史	劉志明	2022	520	可借(4)
11	30分鐘健身計畫	趙明輝	2024	290	可借(1)
19	AI在台灣	石建宏	2024	400	可借(1)
3	Python程式設計入門	陳志明	2024	280	可借(3)
101	一本書	我	2025	500	可借(25)
78	世界料理大全	馬佳玲	2024	280	可借(2)
75	世界歷史故事	鄭雅玲	2023	200	可借(6)
48	中國古代政治史	林志強	2022	520	可借(1)
57	中醫理論與實務	黃志明	2024	380	可借(2)
22	中醫養生智慧	李美玲	2023	280	可借(2)
68	人力資源管理	王志強	2024	300	可借(4)

## 核心程式碼與解釋

實作程式碼所在檔案 — **Library.cpp**

**1. 顯示排序選單 — `showSortMenu()`** 約第 2671 行

提供「依書名／作者／出版年／頁數排序」，選擇後會更新排序欄位並記錄升／降序狀態

**2. 更新排序設定 — `setSortField()`** 約第 2727 行

若連續兩次選相同欄位會在升序／降序間切換，否則切換欄位並預設升序排序

**3. 執行排序 — `applySorting()`** 約第 2634 行

使用 **lambda + QuickSort** 根據欄位欄位呼叫 **getter** 進行排序

```
SortUtil::sort(books, [](const Book& a, const Book& b) {
    return a.getTitle() < b.getTitle();
});
```

**檢視單本書籍**

## 功能概述

本加分功能新增「檢視單本書籍詳情」，允許使用者輸入書籍 ID，即可顯示該書所有欄位資訊，包含：

- 書名、作者、出版年份、ISBN、出版社、語言、頁數
- 分類標籤
- 書籍簡介（摘要）
- 借閱狀態（總複本數、可借副本數）
- 歷史借閱統計（次數／熱門程度）

此功能亦整合於「刪除書籍」與「分頁列表」中，使用者在刪除前或清單中輸入書籍 ID，即可自動顯示該書完整資訊，方便確認內容、避免誤操作。

範例：

書名：書法藝術入門  
作者：王美華  
出版年份：2023  
出版社：藝術家出版  
頁數：180 頁  
語言：繁體中文  
ISBN：978-986-456-123-4

館藏狀況  
總複本數量：2 本  
可借閱數量：**1** 本  
狀態：**可借閱**  
已借出數量：1 本

分類標籤  
[藝術] [書法]

內容簡介  
從基本筆法到作品賞析，完整的書法學習指南。

借閱統計  
歷史借閱次數：2 次  
熱門程度：**普通受歡迎**  
借閱次數中等，有一定讀者群  
統計指標：排名前 100.0% | 為平均值的 0.8 倍 | 為中位數的 1.0 倍

[INFO] 按 Enter 繼續...

## 核心程式碼與解釋

實作程式碼所在檔案 — **Library.cpp**

### 1. 進入檢視詳情模式 — **viewBookDetailsFromList()** 約第 2607 行

- 提示使用者輸入要查看的書籍 ID
- 若輸入  $\leq 0$  或無對應書籍，則退出

```
int bookId = getBookIdChoice("請輸入要查看詳情的書籍 ID");
if (bookId <= 0) return;
const Book* book = bookManager.getBook(bookId);
if (!book) { ConsoleUtil::printError(...); ConsoleUtil::pauseAndWait(); return; }
```

### 2. 顯示詳細資訊區塊 — 一系列 **display** 函式 約第 2623-2628 行

```
ConsoleUtil::clearScreen();
displayBookDetailsHeader(book);
// ... 顯示該書籍所有欄位資訊
ConsoleUtil::pauseAndWait();
```

### 3. 返回列表

按任意鍵後，函式結束，回到原本的分頁或主選單，頁面依然保持先前的排序／搜尋狀態

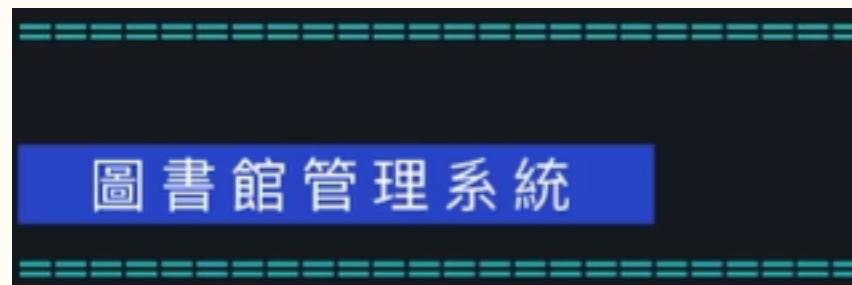
# UI 介面設計

## 功能概述

為了提升使用者體驗，本功能設計了一套簡潔統一的**文字介面（Console-based UI）**，所有畫面均依照**相同的**版面配置與色彩／符號標示，避免出現風格不一致或格式錯亂的狀況。介面包含**標題、選單、錯誤提示、成功訊息、暫停等待等**功能，**並支援多種色彩與分隔線強化視覺層次**。

此外，在**統計圖表**功能中，亦搭配 **VisualizationUtil** 使用 ASCII 輸出簡單圖形，進一步豐富 UI 呈現。

### 範例：



```
[OK] 資料已成功保存  
[WARNING] ! 警告：刪除操作無法復原！  
[INFO] 目前登入：admin (Admin)  
[INFO] 按 Enter 繼續...  
[ERROR] 使用者名稱或密碼無效，請重試
```

## 核心程式碼與解釋

實作程式碼所在檔案 — **ConsoleUtil.cpp**

**1. 清空畫面 — clearScreen()** 約第 17 行

透過轉義序列或系統呼叫**清除終端機螢幕**，確保每次顯示新畫面時不殘留舊內容

**2. 暫停並等待 — pauseAndWait()** 約第 25 行

印出提示（如「按 Enter 繼續…」），等待使用者**按鍵後才繼續主流程**，  
確保畫面內容有足夠時間閱讀

**3. 顯示主標題 — printTitle()** 約第 63 行

在標題前後自動加上分隔線（如 ===== 標題 =====），並以背景色顯示，強化主題識別

**4. 顯示主標題+子標題 — printTitleWithSubtitle()** 約第 73 行

與 printTitle 類似，額外接收副標題參數，適合呈現子頁面

**5. 顯示子標題 — printSubtitle()** 約第 83 行

以「—— 子標題 ——」形式呈現，用亮青色標示，用於區隔內容區塊

## 核心程式碼與解釋

實作程式碼所在檔案 — **ConsoleUtil.cpp**

**6. 顯示成功訊息 — printSuccess()** 約第 89 行

印出 [OK] 訊息，使用亮綠色標示操作成功

**7. 顯示錯誤訊息 — printError()** 約第 94 行

印出 [ERROR] 訊息，使用亮紅色顯示錯誤或無效操作

**8. 顯示警告訊息 — printWarning()** 約第 99 行

印出 [WARNING] 訊息，使用亮黃色提示潛在問題

**9. 顯示資訊訊息 — printInfo()** 約第 104 行

印出 [INFO] 訊息，使用亮藍色說明中立提示

**10. 顯示分隔線 — printSeparator()** 約第 109 行

輸出由特定字元組成的長條（如 =====），用於結構分隔或畫面框架

## 核心程式碼與解釋

實作程式碼所在檔案 — **ConsoleUtil.cpp**

**11. 顯示方框 — `printBox()`** 約第 114 行

用 ASCII 畫出「+————+」框住字串，常見於提醒框。

**12. 顯示完整選單（含標題） — `printMenu()`** 約第 129 行

若有提供標題會呼叫 **printSubtitle()**，再顯示選項內容，常見於主要功能列選單。

**13. 顯示選單選項 — `printMenuOptions()`** 約第 147 行

將字串選項**自動編號**（1. 2. 3.），一行一項顯示，最後提示輸入選擇

**14. 顯示進度條 — `printProgressBar()`** 約第 161 行

顯示 [####-----] 66.7% 風格的文字進度條，計算比例動態更新

**15. 顯示 loading 動畫 — `printLoading()`** 約第 173 行

以 |/-\ 動畫圖案輪播方式顯示「**載入中**」，適合長時間操作提示。

# 錯誤判斷與防呆

## 功能概述

在本作業中，廣泛實作了「錯誤判斷與防呆」機制，以提升系統的穩定性與使用者體驗。

該機制主要包含以下幾個面向：

1. 輸入防呆：防止使用者輸入不合法格式（如非整數選單、空白帳號等），避免程式異常終止
2. 簡化輸入與預設值：部分功能支援「按 Enter 採用預設值」或「保留原本欄位值」，提升容錯率
3. JSON 錯誤偵測：當 JSON 檔案無法正確讀取或開啟失敗時，給予提示並避免執行錯誤流程
4. 資料邏輯檢查：如帳號是否重複、新增書籍時 ID 是否有效、借書是否已借出，皆有明確判斷條件
5. 統一錯誤輸出：透過 ConsoleUtil 中的錯誤／警告／提示訊息介面，使用色彩與標籤標示問題
6. 查詢語法驗證：在進階搜尋中，針對括號、欄位、關鍵字等進行語法分析，偵測並顯示錯誤提示

這些機制貫穿整個系統，從使用者輸入、JSON 處理、帳號管理、借還書邏輯、書籍管理、查詢語法解析等多個模組皆有涵蓋。

範例：

```
11. 修改密碼  
12. 檢視統計資料  
13. 檢視逾期圖書  
14. 登出  
15. 退出系統

請輸入您的選擇：16
[ERROR] 無效的選擇，請重試
[INFO] 按 Enter 繼續...

請輸入您的選擇：ABC
[ERROR] 無效的選擇，請重試
[INFO] 按 Enter 繼續...
```

```
使用者名稱：admin  
密碼：  
確認密碼：  
[INFO] 選擇角色（1=館員， 2=讀者）：  
1  
[ERROR] 新增使用者失敗，使用者名稱可能已存在
[INFO] 按 Enter 繼續...
```

```
[INFO]
請輸入要刪除的圖書 ID:  
101
[ERROR] 找不到 ID 為 101 的書籍
[INFO] 按 Enter 繼續...
```

# 核心程式碼與解釋

## 1. 使用者輸入防呆

舉例：

```
int UIController::getMenuChoice() {
    int choice;
    std::cin >> choice;
    std::cin.clear(); // 清除 std::cin 的錯誤旗標
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // 丟棄這一行中剩下的所有字元（直到換行）
    return choice;
}
```

- 確保使用者在選單輸入非數字或輸入多餘字元時，不會卡在錯誤狀態或死讀取

## 2. 輸入預設值支援

舉例：

```
if (!titleInput.empty()) {
    newBook.setTitle(titleInput);
} else {
    newBook.setTitle(book.getTitle()); // 保留原書名
}
```

- 減少使用者操作負擔，例如「不想修改書籍名就直接按 Enter 保留原值」

# 核心程式碼與解釋

## 3. JSON 錯誤處理

舉例：

```
bool BookManager::loadFromFile(const std::string& filename) {
    std::ifstream file(filename);
    if (!file.is_open()) {
        return false; // 檔案開啟失敗
    }
    try {
        auto j = SimpleJSON::parse(file);
        // ...後續格式／型別檢查...
    } catch (const std::exception& e) {
        std::cerr << "Error loading books: " << e.what() << std::endl;
        return false;
    }
    return true;
}
```

- 每次讀取或儲存 JSON，都先檢查檔案是否成功開啟，並以 **try-catch** 捕捉解析錯誤

## 4. 資料邏輯檢查

舉例：

```
bool UserManager::addUser(const User& u) {
    auto it = SearchUtil::mapFind(users, u.getUsername());
    if (it != users.end()) {
        return false;
    }
    users[u.getUsername()] = u;
    return true;
}
```

- 帳號新增時，先檢查是否已經存在，避免重複註冊

```
bool LoanManager::borrowBook(const std::string& username,
                               int bookId, int graceDays) {
    for (auto& rec : loans) {
        if (rec.getBookId() == bookId && !rec.isReturned())
            return false; // 已有未還紀錄
    }
    // ...新增借閱...
    return true;
}
```

- 確認該書目前沒有未還紀錄，防止重複借出

## 核心程式碼與解釋

### 5. 統一錯誤輸出

此部分在「UI 介面設計」加分功能已詳細介紹過，此處省略

[ERROR] 使用者名稱或密碼無效，請重試

### 6. 查詢語法驗證

舉例：

```
if (token != ")") {  
    std::cerr << "Error: Missing closing parenthesis" << std::endl;  
    return nullptr;  
}
```

- 遇到語法不符即時提示，避免後續錯誤運算

**多條件智慧搜尋**

## 功能概述

本作業實作了「多條件智慧搜尋」，允許使用者輸入一段具邏輯結構的查詢語句，結合欄位篩選與邏輯運算，快速篩出符合條件的書籍資料。

本功能具備以下特性：

1. 多欄位查詢：可針對 **書名**、**作者**、**年份**、**出版社**、**分類** 等欄位進行比對
2. 多種運算子：**=**、**~**（包含）、**>**、**<**、**>=**、**<=**
3. 邏輯組合能力：可使用 **AND**、**OR**、**NOT** 串接多個條件，並使用括號 **()** 控制優先順序
4. 自動辨識書名關鍵字：若未指定欄位，系統會將查詢視為對書名進行的模糊搜尋（子字串比對）
5. 整合倒排索引與欄位比對：系統會先透過 **titleIndex** 快速初篩，再結合欄位邏輯評估

輸入格式：

關鍵字快速搜尋（模糊搜尋）：輸入的文字沒有指定欄位與運算子，將其視為書名搜尋（**title~關鍵字**）

輸入：台灣 **AND** 年份**>=2023 AND** 之

實際查詢：**title~台灣 AND year>=2023 AND title~之**

其他範例查詢語句：

**title~AI AND author~王小明 AND year>=2020**

**台灣 AND (year>=2023 OR author~黃建國)**

**NOT category~漫畫**

**language=English AND pages<=300**

# 功能概述

## 實際介面範例：

### — 多條件智慧搜尋說明 —

[INFO] 支援的運算符：

AND - 交集（同時滿足）

OR - 聯集（滿足其一）

NOT - 差集（排除）

( ) - 括號（優先運算）

[INFO] 支援的欄位查詢：

作者 = "張三" - 精確匹配作者

title ~ "程式" - 書名包含關鍵字

年份 >= 2020 - 年份條件

標籤 ~ "入門" - 標籤包含

[INFO] 運算符說明：

= (等於) ~ (包含) > (大於) < (小於) >= (大於等於) <= (小於等於)

[OK] 範例：程式設計 AND (作者 = "陳鍾誠" OR 年份 >= 2020) NOT 標籤 ~ "入門"

請輸入搜尋條件（支援 AND/OR/NOT、書名/作者/年份/標籤）：台灣 AND 年份 >= 2023 AND "之"

[INFO] 搜尋中...

[OK] 找到了 3 本書：

[7] 台灣新創之路

(作者：黃建國，年份：2024)

[39] 台灣生態之美

(作者：曾志華，年份：2024)

[15] 台灣民主之路

(作者：謝志強，年份：2024)

# 核心程式碼與解釋

## 1. 查詢字串解析器 (QueryParser)

實作程式碼所在檔案 — **QueryParser.h**

```
enum class NodeType { TERM, AND, OR, NOT, FIELD_QUERY, KEYWORD_QUERY };
enum class FieldOperator { EQUALS, CONTAINS, GREATER, LESS, GREATER_EQ, LESS_EQ };

struct QueryNode {
    NodeType type;
    std::string term;
    std::string field;
    FieldOperator fieldOp;
    std::string value;
    std::shared_ptr<QueryNode> left, right;
    explicit QueryNode(NodeType type);
    explicit QueryNode(const std::string& term);
    QueryNode(const std::string& field, FieldOperator op, const std::string& value);
};

class QueryParser {
public:
    std::shared_ptr<QueryNode> parse(const std::string& query);
private:
    void tokenize();
    std::shared_ptr<QueryNode> parseExpression();
    std::shared_ptr<QueryNode> parseTerm();
    std::shared_ptr<QueryNode> parseFactor();
    std::shared_ptr<QueryNode> parseAtom();
    std::shared_ptr<QueryNode> parseFieldQuery(const std::string& field);
    // ...其他 parseXxx()...
};
```

### 功能簡介：

- 接收使用者輸入的查詢字串  
(可包含括號、AND/OR/NOT、欄位條件)
- tokenize → 遞迴下降 Parser → 產生  
**QueryNode** 語法樹

### 重點摘要：

- **NodeType** 定義節點種類：  
關鍵字、欄位查詢、邏輯運算
- parse() 呼叫 tokenize() 後，由  
parseExpression() 依序解析 **OR > AND > NOT**
- **QueryNode** 的左右指標串出一棵完整的查詢樹

# 核心程式碼與解釋

## 2. 遞迴下降 Parser 實作

實作程式碼所在檔案 – **QueryParser.cpp**

```
void QueryParser::tokenize() {
    // 省略空白，自動分割 ()，AND，OR，NOT，:，>，>=，<=...
}

std::shared_ptr<QueryNode> QueryParser::parse(const std::string& q) {
    input = q; pos = 0;
    tokenize();
    return parseExpression();
}

std::shared_ptr<QueryNode> QueryParser::parseExpression() {
    auto node = parseTerm();
    while (pos < tokens.size() && tokens[pos] == "OR") {
        auto root = std::make_shared<QueryNode>(NodeType::OR);
        root->left = node;
        ++pos;
        root->right = parseTerm();
        node = root;
    }
    return node;
}
```

### 功能簡介：

- 實作 `tokenize()` 及遞迴下降法：  
`parseExpression()`、`parseTerm()`、`parseFactor()`

### 重點摘要：

- `tokenize()` 支援各種運算子和符號
- 三層 `parseXxx()` 函式處理括號與邏輯優先序
- `parseExpression()` 建立所有 OR 節點

# 核心程式碼與解釋

## 3. 欄位比對工具

實作程式碼所在檔案 — **QueryMatcher.cpp**

```
std::string QueryMatcher::toLowerCase(const std::string& str) {
    std::string result = str;
    SortUtil::transform(result.begin(), result.end(), result.begin(), ::tolower);
    return result;
}

bool QueryMatcher::matchString(const std::string& text, FieldOperator op,
                               const std::string& value, bool ignoreCase) {
    std::string lhs = ignoreCase ? toLowerCase(text) : text;
    std::string rhs = ignoreCase ? toLowerCase(value) : value;
    switch (op) {
        case FieldOperator::EQUALS:    return lhs == rhs;
        case FieldOperator::CONTAINS: return SearchUtil::contains(lhs, rhs); // 畫面標註
        case FieldOperator::GREATER:   return lhs > rhs;
        // ...其餘比較...
    }
}

bool QueryMatcher::matchNumber(int number, FieldOperator op,
                               const std::string& value) {
    int rhs = std::stoi(value);
    switch (op) {
        case FieldOperator::EQUALS:    return number == rhs;
        case FieldOperator::GREATER:   return number > rhs;
        // ...其餘比較...
    }
}
```

功能簡介：

- 提供文字與數值欄位的條件比對函式，支援等於、包含、大小比較等運算。

重點摘要：

- 用 **SearchUtil::contains()** 手刻 **search** 函式
- **matchString()** 可選擇忽略大小寫
- **matchNumber()** 處理所有數值比較運算

## 核心程式碼與解釋

### 4. 查詢相關宣告

實作程式碼所在檔案 — **BookManager.h**

```
class BookManager {  
public:  
    std::vector<Book*> advancedSearch(const std::string& query) const; // 多條件智慧查詢  
private:  
    std::unordered_set<int> evaluateFieldQuery(const std::shared_ptr<QueryNode>& node) const;  
    bool bookMatchesFieldQuery(const Book& book, const std::shared_ptr<QueryNode>& node) const;  
    std::unordered_set<int> searchInTitle(const std::string& query) const;  
};
```

#### 功能簡介：

- 宣告「多條件智慧查詢」的主入口與輔助函式

#### 重點摘要：

- **advancedSearch()** 完全取代舊版 **searchBooks()**
- **private** 封裝欄位查詢與標題查詢邏輯

# 核心程式碼與解釋

## 5. 查詢主流程與輔助函式

實作程式碼所在檔案 – **BookManager.cpp**

```
std::vector<Book*> BookManager::advancedSearch(const std::string& query) const {
    QueryParser parser;
    auto root = parser.parse(query);
    std::unordered_set<int> allBookIds;
    for (auto& p : bookIdMap) allBookIds.insert(p.first);

    std::function<std::unordered_set<int>(std::shared_ptr<QueryNode>)> eval =
        [&](auto node) {
            switch (node->type) {
                case NodeType::TERM:      return searchInTitle(node->term);
                case NodeType::AND:       { /* 交集 */ }
                case NodeType::OR:        { /* 聯集 */ }
                case NodeType::NOT:       { /* 補集 */ }
                case NodeType::FIELD_QUERY: return evaluateFieldQuery(node);
                default: return std::unordered_set<int>();
            }
        };
    auto ids = eval(root);
    std::vector<Book*> results;
    for (int id : ids) results.push_back(&books[bookIdMap.at(id)]);
    return results;
}

std::unordered_set<int> BookManager::evaluateFieldQuery(const std::shared_ptr<QueryNode>& node) const {
    std::unordered_set<int> res;
    for (auto& bk : books)
        if (bookMatchesFieldQuery(bk, node)) res.insert(bk.getId());
    return res;
}
```

### 功能簡介：

- **advancedSearch()** 呼叫 **QueryParser** 解析語法樹，並遞迴組合所有條件
- **evaluateFieldQuery()**, **bookMatchesFieldQuery()**, **searchInTitle()** 分別處理欄位查詢與關鍵字查詢

### 重點摘要：

- **advancedSearch()** 使用**遞迴 lambda 函式**，把整個布林邏輯處理流程包在一起
- **evaluateFieldQuery()** 會針對所有書籍，逐一檢查是否符合查詢條件
- **searchInTitle()** 的實作**兼顧效率與中文支援**：先對英文做 token 比對，再 fallback 成中文字串查找

# 統計圖表

## 功能概述

本功能透過 **ASCII 字元** 與 **ANSI 色碼**，將統計資料以**視覺化**方式輸出於終端機介面中，協助**館長**快速掌握系統概況。

所有圖表統一使用**水平長條圖**格式呈現，並依據每筆資料的佔比自動套用不同顏色：

0 - 70% (低至中等值) — 綠色

70 - 90% (中高值) — 黃色

90 - 100% (極高值) — 紅色

進入方式：

- 僅限 **Admin** 角色可使用本功能
- 登入後，從主選單選擇 **12. 檢視統計資料** 進入
- 接著會進入統計子選單，包含四種統計類型供使用者瀏覽

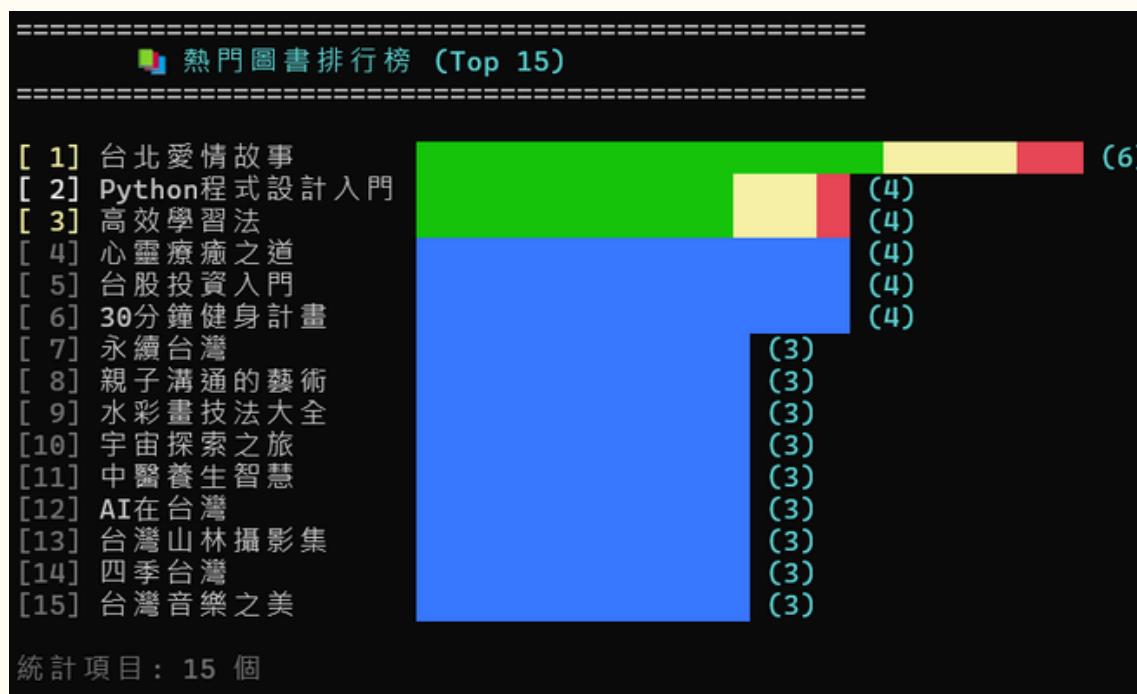
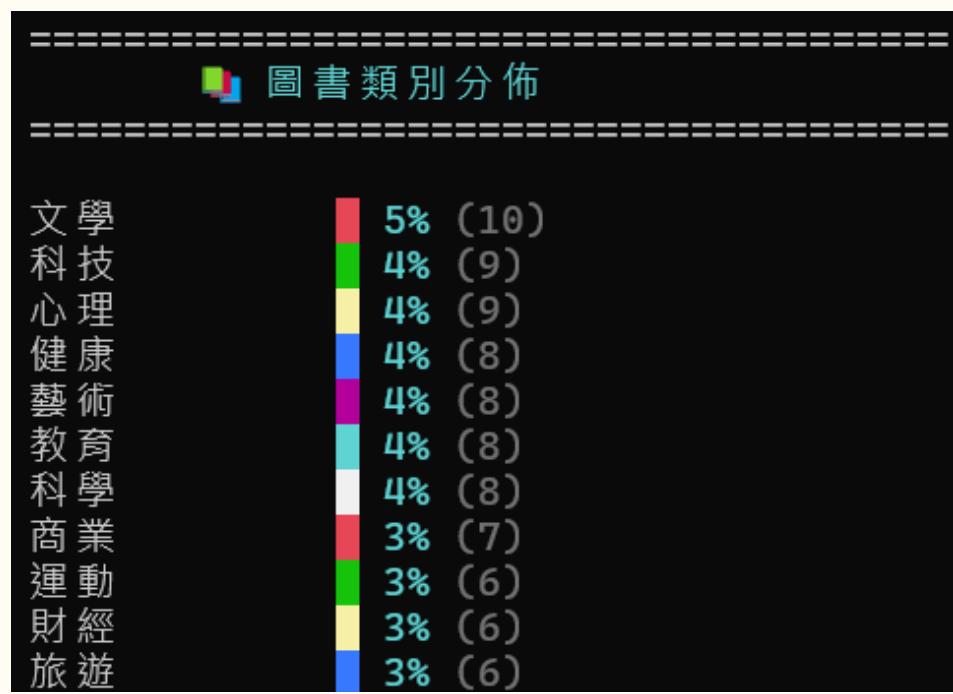
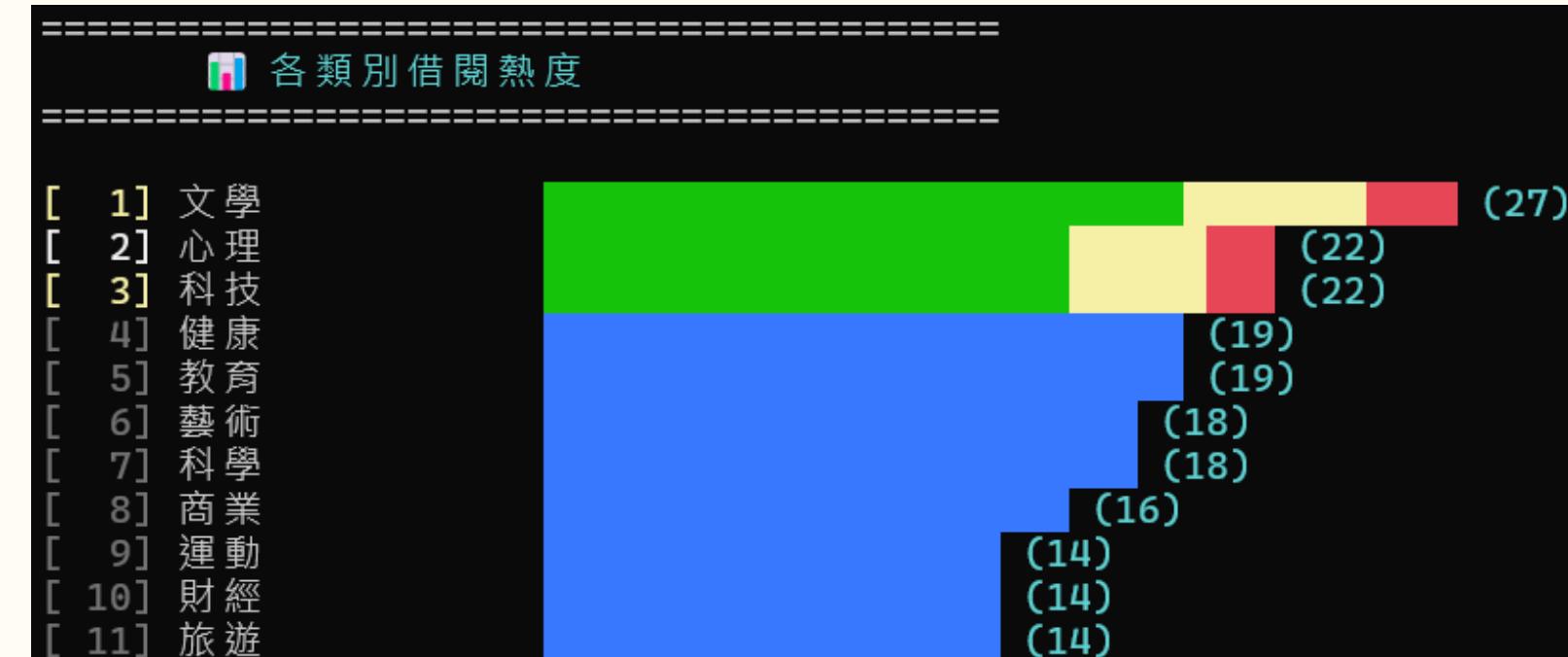
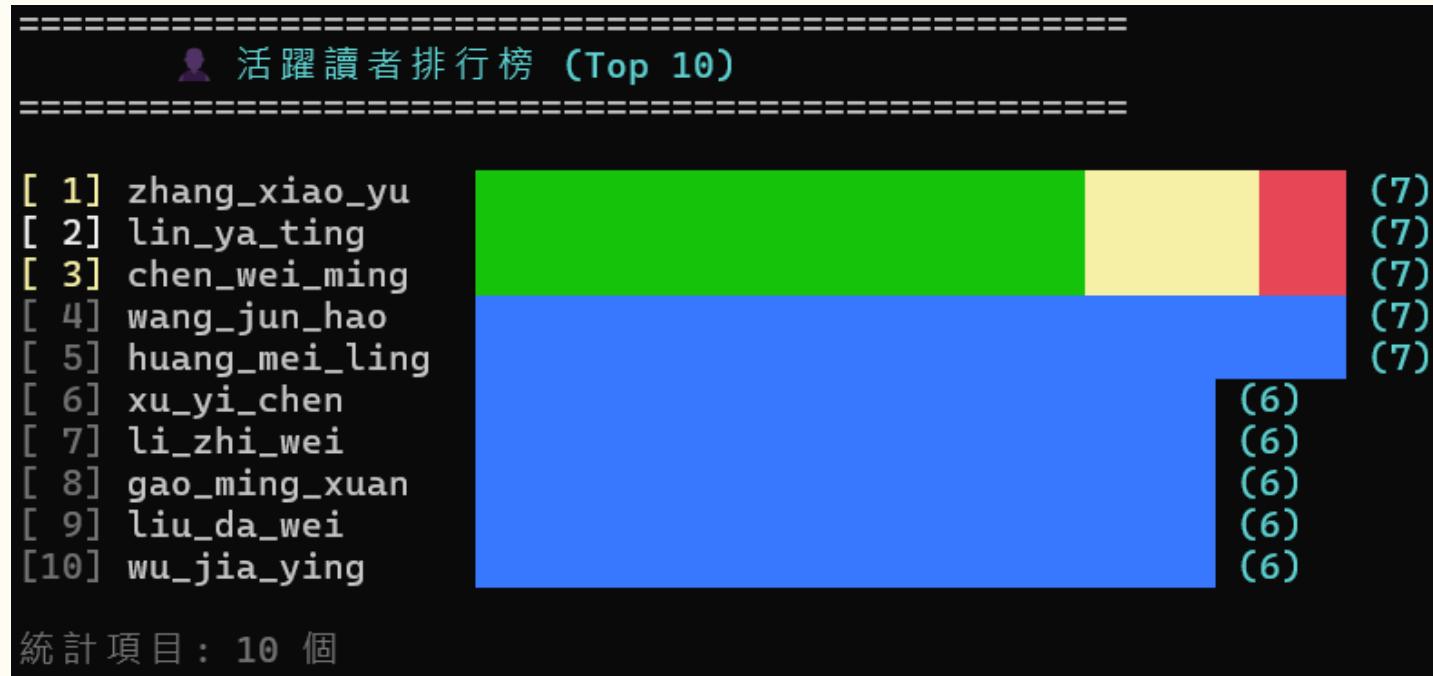


統計圖表內容（下頁有範例圖示）：

- **熱門圖書排行榜**：顯示借閱次數前 15 名圖書
- **活躍讀者排行榜**：顯示借閱次數前 10 名使用者
- **圖書分類分佈**：顯示各分類館藏數量（含百分比）
- **各類別借閱熱度**：顯示每個分類的總借閱次數
- **月度借閱趨勢分析**：顯示近 12 個月的月別借閱數量變化

# 功能概述

## 統計圖表範例：



# 核心程式碼與解釋

## 1. 圖表函式定義

實作程式碼所在檔案 — **VisualizationUtil.h**

### Bar Chart (水平長條圖)

```
void drawBarChart(const std::vector<std::pair<std::string,int>>& data, ...);  
void drawBarChart(const std::unordered_map<std::string,int>& data, ...);  
void drawBarChart(const std::unordered_map<int,int>& data, ...);
```

- 輸入資料：標籤 + 數值
- 功能：依數值長度畫出長條圖
- 顏色依比值自動套用：綠／黃／紅

### Pie Chart (模擬圓餅)

```
void drawPieChart(const std::vector<pair<string,int>>&, ...);  
void drawPieChart(const std::unordered_map<string,int>&, ...);
```

- 使用 ■ 表示占比
- 顯示數量與百分比
- 用於分類分布視覺化

### Line Chart (水平線圖)

```
void drawLineChart(const std::vector<pair<string,int>>&, ...);
```

- 雖名為折線，實際仍為水平線段圖
- 顯示數值變化趨勢
- 用於月度借閱分析

## 核心程式碼與解釋

### 2. 圖表繪製邏輯

實作程式碼所在檔案 — **VisualizationUtil.cpp**

#### 共用工具函式：

`getDisplayWidth()` — 正確計算中英文混排的實際寬度，避免對齊錯亂

`printHeader()` — 輸出統一風格標題框（使用亮藍色與分隔線）

`printSeparator()` — 輸出指定長度的分隔線（預設為 =）

#### Bar Chart 實作重點

```
drawBarChart(vector<pair<string,int>> data, string title, int maxWidth);
```

- 將資料排序後依照最大值轉換長度比例
- 使用 `createBar()` 畫出由 █ 組成的長條
- 前 3 名套用顏色：綠／黃／紅

```
drawBarChart(unordered_map<...> data, ...);
```

- 將 map 轉成 vector，再呼叫上面版本

## 核心程式碼與解釋

### 2. 圖表繪製邏輯

實作程式碼所在檔案 — **VisualizationUtil.cpp**

#### Pie Chart 實作重點

```
drawPieChart(vector<pair<string,int>> data, string title);
```

- 計算總數後，依每項比例以 ■ 重複輸出
- 顯示分類名稱、數量與百分比
- 實際為模擬圓餅的長條格式

#### Line Chart 實作重點

```
drawLineChart(vector<pair<string,int>> data, string title);
```

- 雖稱 Line Chart，實際為長度對應的橫線 —————
- 主要用於月份或時間序列呈現趨勢
- 可自訂最大長度（如 60）

# 核心程式碼與解釋

## 3. 統計功能呼叫

實作程式碼所在檔案 — **Library.cpp**

### **showBorrowStats()** — 借閱排行與摘要

- 呼叫 `loanManager` 拿取借閱資料
- 顯示圖表：
  - Top 15 熱門圖書 → `drawBarChart`
  - Top 10 活躍使用者 → `drawBarChart`
  - 借閱摘要 → `showBorrowingSummary()` (文字表)

### **showCategoryStats()** — 分類分析

- 對全館圖書進行分類統計與借閱量統計
- 顯示圖表：
  - 館藏分類比例 → `drawPieChart`
  - 各分類借閱量 → `drawBarChart`
  - 分類效率 → `showCategoryEfficiency()` (平均每本借閱次數)

### **showMonthlyStats()** — 月度趨勢分析

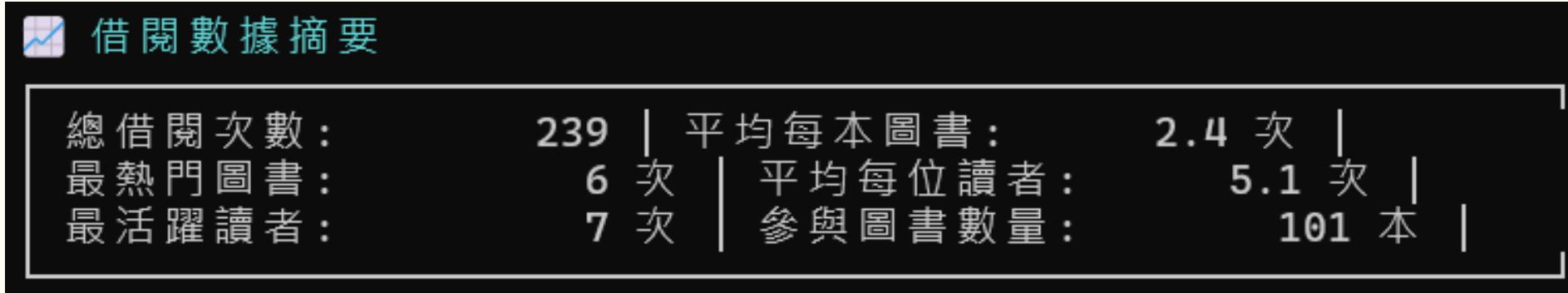
- 分析近 12 個月借閱紀錄
- 製作 key 為 YYYY-MM 的月別索引
- 顯示圖表：
  - 月別趨勢 → `drawLineChart`
  - 附上趨勢摘要文字 → `showMonthlyStatsSummary()`

# 核心程式碼與解釋

## 4. 統計摘要與效率分析

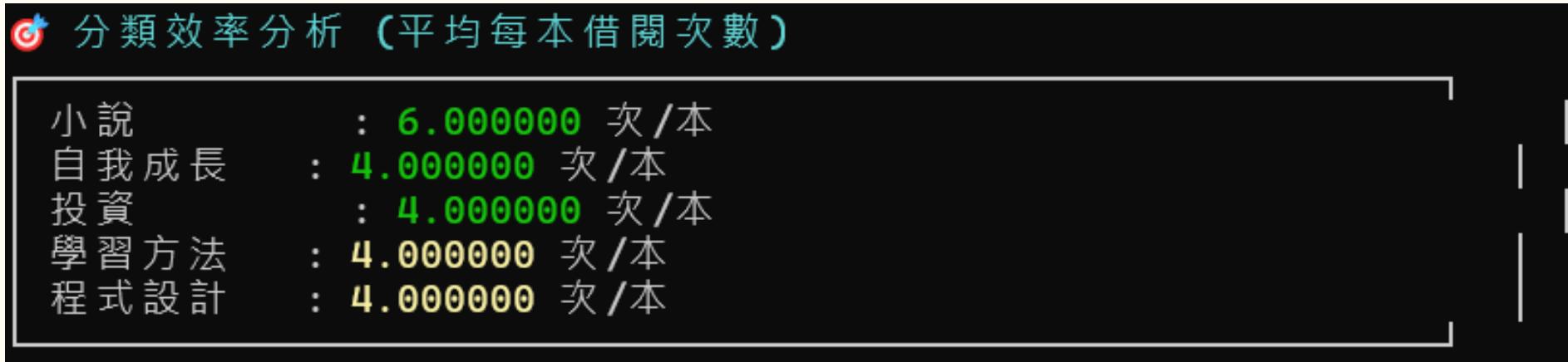
實作程式碼所在檔案 — **Library.cpp**

### showBorrowingSummary()



- 使用 `std::setw()` 對齊欄位
- 框線使用 `---` | `—` 等字元
- 主標題有藍色高亮提示

### showCategoryEfficiency()



- 僅顯示前 5 名分類 (依效率由高到低排序)
- 前 3 名使用亮綠色，後 2 名使用亮黃色
- 自動略過未分類或 0 本書的項目
- 避免除以 0，結果為 0.0

- 計算方式：每類平均借閱次數 = 總借閱數 ÷ 館藏數

# 借閱與歸還圖書

## 功能概述

本功能提供使用者**借閱與歸還圖書**的基本操作，確保館藏流通與紀錄一致性。  
使用者可透過**圖形化介面**選擇書籍進行借出或歸還，並獲得即時成功或錯誤提示。

### 借閱圖書功能：

使用者可瀏覽館藏中**可供借閱**的書籍，選擇欲借書籍後建立借閱紀錄，並設定**歸還期限**（預設14天）  
若書籍無庫存、使用者帳號無效，會顯示錯誤提示

- Reader – 僅可借閱自己的帳號
- Staff / Admin – 可為任意使用者借書

```
[INFO]  
請輸入圖書 ID:  
89  
[OK] 圖書借閱成功！  
[INFO] 按 Enter 繼續...
```

### 歸還圖書功能：

使用者可**查看**目前仍在**借閱中的**書籍，並完成歸還流程。  
系統會自動**記錄**歸還時間，作為後續統計與**延伸功能**（如逾期罰款）的依據。

```
[ID: 79] 音樂理論基礎 [正常]  
[INFO]  
請輸入要歸還的圖書 ID:  
79  
[OK] 圖書歸還成功！  
[INFO] 按 Enter 繼續...
```

# 核心程式碼與解釋

## 1. 借還功能介面

實作程式碼所在檔案 — **LoanManager.h**

```
bool borrowBook(const std::string& username, int bookId, int graceDays = 0);
bool returnBook(const std::string& username, int bookId);
```

- 對外提供借書與還書的主函式介面
- 系統內部以**三層索引管理紀錄**：
- loans (全體紀錄) , bookLoans[bookId] (每本書紀錄) , userLoans[username] (每位使用者的紀錄)

## 2. 核心邏輯實作

實作程式碼所在檔案 — **LoanManager.cpp**

```
bool LoanManager::borrowBook(const std::string& user, int id, int grace) {
    time_t now = time(nullptr);
    time_t due = now + 14*24*60*60; // 預設 14 天
    loans.emplace_back(user, id, now, due, grace);
    bookLoans[id].push_back(&loans.back());
    userLoans[user].push_back(&loans.back());
    return true;
}
```

- 新增一筆借閱紀錄 **LoanRecord**
- 將指標同步加入三個索引表中

```
bool LoanManager::returnBook(const std::string& user, int id) {
    for (auto* rec : userLoans[user])
        if (rec->getBookId() == id && !rec->isReturned()) {
            rec->markReturned(); // 標記已歸還、記錄時間
            return true;
        }
    return false;
}
```

- 找到尚未歸還的該書借閱紀錄並更新狀態
- 回傳是否成功 (失敗代表無符合紀錄)

## 核心程式碼與解釋

### 3. 借還流程整合 實作程式碼所在檔案 — Library.cpp

```
void Library::borrowBook() {
    auto list = getAvailableBooks();
    displayAvailableBooks(list);
    int id = getBookIdChoice("請輸入圖書 ID");
    std::string user = getBorrowerUsername(); // 根據角色取得帳號
    loanManager.borrowBook(user, id)
        ? printSuccess("借閱成功")
        : printError("借閱失敗");
}
```

- 取得可借書目 → 輸入 ID → 輸入帳號  
(Reader 自動帶入) → 執行借閱

```
void Library::returnBook() {
    std::string user = getTargetUserForReturn();
    auto loans = getActiveLoansForUser(user);
    displayActiveLoans(loans);
    int id = getBookIdChoice("請輸入要歸還的圖書 ID");
    loanManager.returnBook(user, id)
        ? printSuccess("歸還成功")
        : printError("歸還失敗");
}
```

- 顯示借閱中清單 → 輸入 ID →  
執行歸還 → 顯示結果

## 核心程式碼與解釋

### 4. 借閱紀錄結構

實作程式碼所在檔案 — **LoanRecord.h**

```
class LoanRecord {
    std::string username;
    int bookId;
    time_t borrowTime, dueTime, returnTime;
    bool returned = false;
public:
    bool isReturned() const { return returned; }
    void markReturned()      { returned = true; returnTime = time(nullptr); }
};
```

- 儲存每筆借閱資訊：借書時間、到期時間、是否歸還、實際歸還時間
- 提供基本操作介面供 **LoanManager** 使用

# 協同過濾 + 內容相似度推薦

## 功能概述

本功能實作了「**協同過濾 + 內容式推薦系統**」，能根據使用者的借閱行為與書籍內容特徵，自動產生**個人化推薦清單**，協助使用者快速找到可能感興趣的圖書。

當使用者有借閱紀錄時，系統會同時計算兩種來源的推薦分數：

1. 協同過濾推薦：根據與你借過相似書的**其他使用者**，找出他們也借過但你尚未借的書籍
2. 內容式推薦：計算書名、作者、分類、簡介等文字向量之間的**餘弦相似度**，找出內容相近的書籍
3. 加權混合推薦：預設以權重 **協同過濾 60% + 內容相似度 40%** 合併兩者結果，產生**最終推薦清單**

若使用者尚無任何借書紀錄，則退回顯示當前熱門書單作為推薦結果

進入方式：登入系統後，從主選單選擇 **檢視推薦** 進入

範例：

```
— ===== 歡迎，test！以下是為您精選推薦 ===== —
1. [44] 經濟學原理          (基於協同過濾推薦指數 : 5.69)
   作者 : 劉建國 | 可借閱 (2 本)
2. [34] 日本深度旅遊          (混合推薦指數 : 0.63)
   作者 : 蔡佩君 | 可借閱 (5 本)
3. [24] 水彩畫技法大全          (基於協同過濾推薦指數 : 4.97)
   作者 : 張雅婷 | 可借閱 (1 本)
4. [39] 台灣生態之美          (基於協同過濾推薦指數 : 3.80)
   作者 : 曾志華 | 可借閱 (6 本)
5. [82] 供應鏈管理          (推薦指數 : 0.41)
   作者 : 張建華 | 可借閱 (3 本)

[INFO] 輸入書號即可借閱，或按 Enter 返回主選單 ...
```

## 核心程式碼與解釋

### 1. 借還功能介面

實作程式碼所在檔案 — **RecommendationEngine.h**

```
class RecommendationEngine {
public:
    void initialize(const BookManager&, const LoanManager&);
    std::vector<std::pair<int,double>> getCollaborativeFilteringRecommendations(const std::string& username, int count) const;
    std::vector<std::pair<int,double>> getContentBasedRecommendations(int bookId, int count) const;
    std::vector<std::pair<int,double>> getHybridRecommendations(const std::string& username, int count) const;
    void displayRecommendations(const std::string& username, const BookManager&, int count) const;
};
```

- 封裝所有推薦邏輯，外部僅需呼叫 **getHybridRecommendations()**
- **initialize()** 在主程式開啟時執行一次，會產生**共現矩陣與 TF-IDF 向量**

## 核心程式碼與解釋

### 2. 演算法核心

實作程式碼所在檔案 — **RecommendationEngine.cpp**

```
void RecommendationEngine::initialize(const BookManager& bm, const LoanManager& lm) {
    buildUserLoanMatrix(lm);
    buildCooccurrenceMatrix();
    buildVocabulary(bm);
    computeIDF(bm);
    computeTFIDFVectors(bm);
}

auto cf = getCollaborativeFilteringRecommendations(user, 25);
auto cb = getContentBasedRecommendations(lastBorrowBookId, 25);
mergeAndWeight(cf, cb, 0.6, 0.4); // 將兩者混合排序
```

- 協同過濾：根據借閱共現次數計算 **confidence × rarityBonus** 分數，並依熱門度正規化
- 內容分析：計算書籍文字資訊的 TF-IDF 向量，透過餘弦相似度比較相似書
- **mergeAndWeight()**：根據預設比重（如 **0.6 vs 0.4**）將兩個推薦結果合併排序

## 核心程式碼與解釋

### 3. 使用者互動流程

實作程式碼所在檔案 — **Library.cpp**

```
void Library::showRecommendations() {
    ConsoleUtil::printTitle("智能推薦系統");
    std::string user = userManager.getCurrentUser()->getUsername();
    if (loanManager.getLoansForUser(user).empty()) {
        showPopularBooksRecommendation(); // 無歷史則顯示熱門書籍
        return;
    }
    showInteractiveRecommendations(user);
}

void Library::showInteractiveRecommendations(const std::string& user) {
    auto hybrid = recommendationEngine.getHybridRecommendations(user, 5);
    displayRecommendationList(hybrid, user);
    handleRecommendationInteraction(hybrid); // 可直接借閱推薦書
}
```

- UI 僅負責呼叫演算法與展示結果；互動介面提供借閱、刷新等操作

**逾期歸還罰款**

## 功能概述

本功能實作了借書逾期自動罰款機制，在歸還與借閱查詢時同步顯示罰款資訊，並提供館長和管理員針對政策的設定與逾期清單檢視

### 罰款觸發條件：

- 歸還圖書時，若實際歸還日 > 到期日 + 寬限天數，即判定逾期
- 預設為 14 天到期、0 天寬限；每延遲一天罰款 \$0

### 讀者端顯示：

- 在「檢視我的借閱」列表中，會標註：逾期天數 & 預估罰款金額
- 在歸還成功後也會顯示本次實際罰款

### 管理端操作：

- 主選單 2. 設置罰款政策 (Admin 專屬)  
可即時修改：寬限天數、每日罰金、遞增因子

- 主選單 12. 檢視逾期圖書 (Staff & Admin)  
列出所有逾期書籍與對應使用者  
顯示逾期天數與累計罰款，並可依使用者分組總覽

```
1. 修改寬限期 (目前: 5 天)
2. 修改固定費率 (目前: $20.00000 每天)
3. 修改遞增因子 (目前: 1.000000)
4. 使用預設政策
5. 儲存並退出
6. 取消設置
```

請輸入您的選擇： 2

```
==== wang_jun_hao (2 本逾期) ====
[15] 台灣民主之路
    逾期: 136 天
    罰款: $2620.00
[93] 芳香療法指南
    逾期: 145 天
    罰款: $2800.00
小計罰款: $5420.00
```

# 核心程式碼與解釋

## 1. 介面定義 實作程式碼所在檔案 — FinePolicy.h

```
class FinePolicy {  
private:  
    int    graceDays      = 0;        // 寬限天數  
    double fixedRate      = 0.0;      // 固定日罰金  
    double incrementalFactor= 1.0;    // 遞增因子  
  
public:  
    FinePolicy() = default;  
    FinePolicy(int g,double r,double f)  
        : graceDays(g), fixedRate(r), incrementalFactor(f) {}  
  
    double calculateFine(int overdueDays) const;  
  
    int    getGraceDays()      const { return graceDays; }  
    double getFixedRate()       const { return fixedRate; }  
    double getIncrementalFactor() const { return incrementalFactor; }  
    void   setGraceDays(int g)      { graceDays = g; }  
    void   setFixedRate(double r)    { fixedRate = r; }  
    void   setIncrementalFactor(double f) { incrementalFactor = f; }  
  
    void   display() const;  
};
```

- **graceDays**：允許的「免罰期」天數  
(預設 0 天)
- **fixedRate**：每天的罰金金額  
(預設 0.0 → 未設定則不收罰款)
- **incrementalFactor**：若 >1.0，則罰金會依因子每日累增；否則固定費率

## 核心程式碼與解釋

### 2. 罰款計算

實作程式碼所在檔案 — **FinePolicy.cpp**

```
double FinePolicy::calculateFine(int overdueDays) const {
    if (overdueDays <= graceDays) return 0.0;

    int chargeDays = overdueDays - graceDays;
    if (incrementalFactor <= 1.0)
        return chargeDays * fixedRate;

    double fine = 0.0;
    for (int i = 0; i < chargeDays; ++i)
        fine += fixedRate * std::pow(incrementalFactor, i);
    return fine;
}
```

- 過期判定：只有 `overdueDays > graceDays` 時才算罰款
- 固定費率：`incrementalFactor ≤ 1.0` 時，罰款 =  $\text{chargeDays} \times \text{fixedRate}$
- 遞增費率：`incrementalFactor > 1.0` 時，罰款依次日累加

## 核心程式碼與解釋

### 3. 歸還時套用罰款

實作程式碼所在檔案 — **LoanManager.cpp**

```
double LoanManager::calculateFine(const LoanRecord& rec) const {
    return finePolicy.calculateFine(rec.getDaysOverdue());
}

bool LoanManager::returnBook(const std::string& user, int id) {
    // ...尋找 rec 並 rec->markReturned()...
    double fine = calculateFine(*rec);
    std::cout << "[INFO] 本次罰款 $" << fine << "\n";
    return true;
}
```

- **calculateFine()**：由 **LoanRecord** 提供逾期天數，呼叫 **FinePolicy** 計算
- **returnBook()**：歸還時印出本次罰款金額，尚未寫入使用者帳戶

# 核心程式碼與解釋

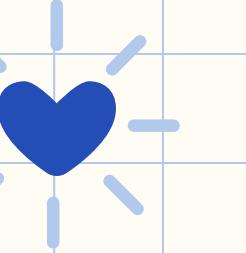
## 4. 管理介面 & 逾期清單

### 實作程式碼所在檔案 — Library.cpp

```
// 2. 設置罰款政策 (Admin 專屬)
void Library::finePolicyMenu() {
    FinePolicy p = loanManager.getFinePolicy();
    while (true) {
        p.display();
        ConsoleUtil::printMenu({
            "1. 修改寬限天數",
            "2. 修改固定日罰金",
            "3. 修改遞增因子",
            "4. 返回"
        });
        int ch = getMenuChoice();
        if      (ch==1) p.setGraceDays(promptInt("新寬限天數"));
        else if (ch==2) p.setFixedRate(promptDouble("新固定日罰金"));
        else if (ch==3) p.setIncrementalFactor(promptDouble("新遞增因子"));
        else break;
    }
    loanManager.setFinePolicy(p);
}
```

```
// 12. 檢視逾期圖書 (Staff & Admin)
void Library::displayOverdueLoans() {
    auto overdue = loanManager.getOverdueLoans();
    double total = 0;
    for (const auto* loan : overdue) {
        double fine = loanManager.calculateFine(*loan);
        total += fine;
        std::cout << "[" << loan->getBookId() << "] "
              << loan->getDaysOverdue() << " 天逾期，預估罰款 $"
              << fine << "\n";
    }
    ConsoleUtil::printWarning("小計罰款: $" + std::to_string((int)total));
}
```

- **finePolicyMenu()**：Admin 可即時調整三項參數，修改後由 LoanManager 採用
- **displayOverdueLoans()**：列出所有逾期紀錄、逾期天數與估算罰款，並計算總額



THANK YOU

報告到此結束，感謝參閱