

**UNIVERSITAS ESA UNGGUL**

**CSF101 ALGORITMA DAN PEMROGRAMAN KJ1001 7174**

**BUILDING CONCEPT OF PROJECT INITIATION – APLIKASI PEMINJAMAN DAN PENGEMBALIAN BUKU**

**TUGAS KELOMPOK 7 PROGRESS 2**

**Dosen Pengampu:**

**7174 - Ir. Sawali Wahyu, S.Kom., M.Kom**

**Kelompok 7:**

|  |  |  |
| --- | --- | --- |
| 1. **Muhamad Akbar Fadilah** | **-** | **20200801269** |
| 1. **Christian Niko Saputra** | **-** | **20240801295** |
| 1. **Denis Prastya Putra** | **-** | **20240801319** |
| 1. **Davina Tri Febriyanti** | **-** | **20240801361** |
| 1. **Arva Raihan Javier** | **-** | **20240801344** |

**PROGRAM STUDI SISTEM INFORMASI**

**FAKULTAS ILMU KOMPUTER**

**UNIVERSITAS ESA UNGGUL**

**TAHUN 2024**

# **DAFTAR ISI**

[**DAFTAR ISI** 2](#_Toc186906081)

[**PROJECT OVERVIEW** 3](#_Toc186906082)

[**1.** **Deskripsi Proyek** 3](#_Toc186906083)

[**2.** **Tujuan Proyek** 3](#_Toc186906084)

[**3.** **Fitur** 3](#_Toc186906085)

[**4.** **Pengguna Sistem** 3](#_Toc186906086)

[**5.** **Teknologi yang Digunakan** 4](#_Toc186906087)

[**FLOWCHART KONSEP APLIKASI** 5](#_Toc186906088)

[**PSEUDOCODE APLIKASI** 6](#_Toc186906089)

[**METODOLOGI PENGEMBANGAN SISTEM** 10](#_Toc186906090)

[**HASIL AKHIR** 11](#_Toc186906091)

[**1.** **Source Code** 11](#_Toc186906092)

[**2.** **Link Github** 34](#_Toc186906093)

[**3.** **Screenshot Aplikasi** 34](#_Toc186906094)

[**4.** **Alur Proses Penggunaan Aplikasi** 34](#_Toc186906095)

# **PROJECT OVERVIEW**

## **Deskripsi Proyek**

Aplikasi Peminjaman dan Pengembalian Buku adalah sistem manajemen perpustakaan digital yang dirancang untuk memudahkan proses administrasi perpustakaan. Sistem ini mengotomatisasi proses peminjaman dan pengembalian buku, serta memberikan visibilitas yang lebih baik terhadap inventaris perpustakaan.

## **Tujuan Proyek**

Dalam era digital yang terus berkembang, perpustakaan tradisional menghadapi tantangan signifikan dalam mengelola operasional mereka secara efisien dan efektif. Dengan mengadopsi teknologi terkini dan metodologi pengembangan yang tepat, sistem ini diharapkan dapat menjadi solusi komprehensif untuk berbagai permasalahan yang dihadapi perpustakaan. Adapun tujuan spesifik dari pengembangan proyek ini adalah:

* Meningkatkan efisiensi pengelolaan perpustakaan
* Meminimalisir kesalahan dalam pencatatan
* Mempermudah pelacakan status buku
* Menghasilkan laporan yang akurat tentang aktivitas perpustakaan

## **Fitur**

* 1. Manajemen Buku
     1. Pendaftaran buku baru
     2. Katalog buku digital
     3. Pencarian buku
     4. Pembaruan status buku
  2. Manajemen Anggota
     1. Pendaftaran anggota baru
     2. Pengelolaan profil anggota
     3. Riwayat peminjaman
     4. Status keanggotaan
  3. Transaksi
     1. Proses peminjaman buku
     2. Proses pengembalian buku
     3. Perpanjangan masa pinjam
     4. Perhitungan denda keterlambatan
  4. Pelaporan
     1. Laporan peminjaman
     2. Statistik buku populer
     3. Laporan keterlambatan

## **Pengguna Sistem**

* 1. Admin Perpustakaan
  2. Petugas Perpustakaan
  3. Supervisor/Kepala Perpustakaan

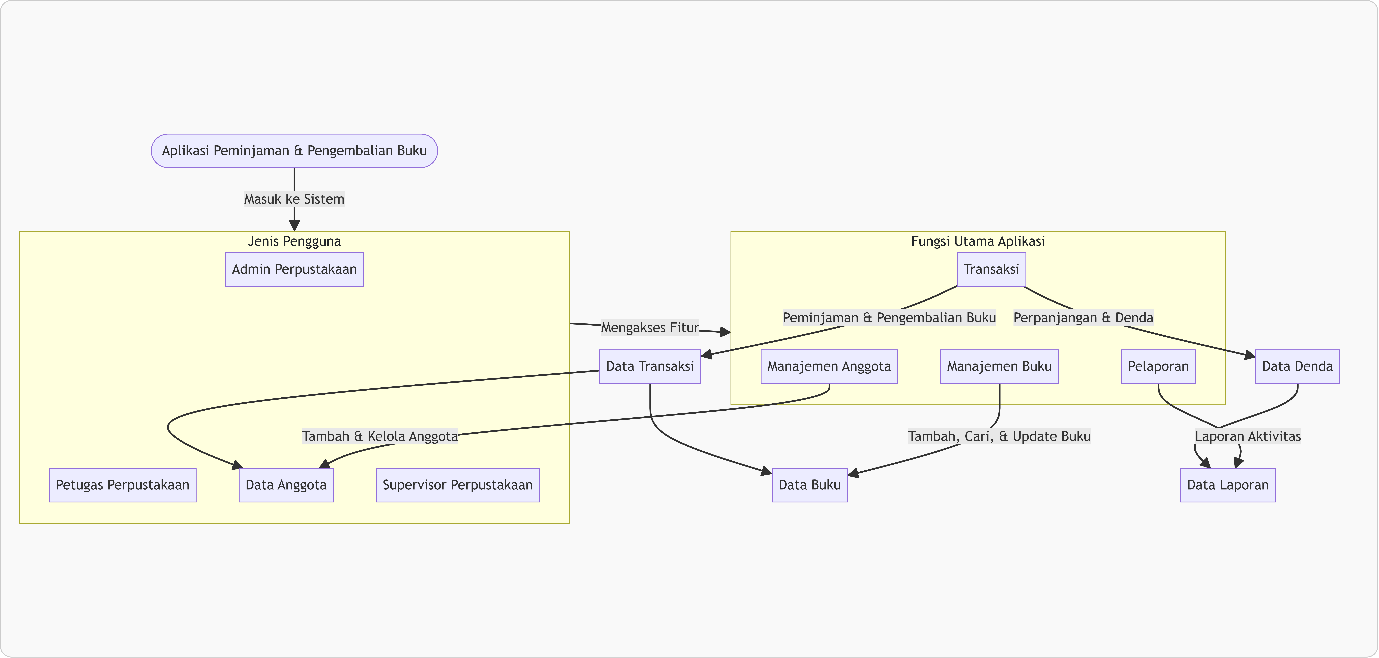
## **Teknologi yang Digunakan**

* 1. Bahasa Pemrograman:
     1. C++
  2. Struktur Data:
     1. Stack

1. **Development Tools**
   1. IDE dan Text Editor:
      1. Dev-C++
   2. Compiler:
      1. GNU G++
2. **Version Control**
   1. Git untuk manajemen versi kode
   2. GitHub untuk repositori dan kolaborasi
3. **Testing Tools**
   1. Unit Testing
      1. Pengujian fungsi-fungsi individual
      2. Validasi operasi stack

# **FLOWCHART KONSEP APLIKASI**

Flowchart ini menggambarkan alur kerja aplikasi Peminjaman & Pengembalian Buku yang mencakup interaksi pengguna, fungsi utama, dan hubungan antar data



***Gambar 1 – Flowchart Konsep Aplikasi***

# **PSEUDOCODE APLIKASI**

START

// Pengguna masuk ke sistem

User = Login()

// Jenis pengguna yang dapat mengakses sistem

IF User is Admin:

    Show Admin Dashboard

    AdminAction = Display Admin Menu

    IF AdminAction is "Manage Books":

        ManageBooks()

    IF AdminAction is "Manage Members":

        ManageMembers()

    IF AdminAction is "Transactions":

        ManageTransactions()

    IF AdminAction is "Reports":

        GenerateReports()

ELSE IF User is Petugas:

    Show Petugas Dashboard

    PetugasAction = Display Petugas Menu

    IF PetugasAction is "Transactions":

        ManageTransactions()

    IF PetugasAction is "Reports":

        GenerateReports()

ELSE IF User is Anggota:

    Show Anggota Dashboard

    AnggotaAction = Display Anggota Menu

    IF AnggotaAction is "Borrow":

        BorrowBook()

    IF AnggotaAction is "Return":

        ReturnBook()

ELSE IF User is Supervisor:

    Show Supervisor Dashboard

    SupervisorAction = Display Supervisor Menu

    IF SupervisorAction is "Reports":

        GenerateReports()

// Fungsi untuk mengelola buku

ManageBooks():

    Display Manage Books Menu

    Action = Select Action (Add, Search, Update Book)

    IF Action is "Add Book":

        AddBook()

    IF Action is "Search Book":

        SearchBook()

    IF Action is "Update Book":

        UpdateBook(

// Fungsi untuk mengelola anggota

ManageMembers():

    Display Manage Members Menu

    Action = Select Action (Add, Manage Member)

    IF Action is "Add Member":

        AddMember()

    IF Action is "Manage Member":

        ManageExistingMember()

// Fungsi untuk mengelola transaksi

ManageTransactions():

    Display Manage Transactions Menu

    Action = Select Action (Borrow, Return, Extend, Fine)

    IF Action is "Borrow Book":

        BorrowBook()

    IF Action is "Return Book":

        ReturnBook()

    IF Action is "Extend":

        ExtendLoan()

    IF Action is "Fine":

        ApplyFine(

// Fungsi untuk membuat laporan

GenerateReports():

    Display Reports Menu

    ReportType = Select Report Type (Activity Report, Transaction Report, etc.)

    IF ReportType is "Activity Report":

        ShowActivityReport()

// Fungsi untuk peminjaman buku

BorrowBook():

    BookID = GetBookID()

    MemberID = GetMemberID()

    CheckAvailability(BookID)

    IF Book is Available:

        CreateTransaction(MemberID, BookID)

// Fungsi untuk pengembalian buku

ReturnBook():

    BookID = GetBookID()

    MemberID = GetMemberID()

    ProcessReturn(MemberID, BookID)

    IF Return is Late:

        CalculateFine(MemberID)

// Fungsi untuk perpanjangan buku

ExtendLoan():

    BookID = GetBookID()

    MemberID = GetMemberID()

    CheckLoanExtension(MemberID, BookID)

    IF Extension is Allowed:

        ExtendBookLoan(MemberID, BookID

// Fungsi untuk menghitung denda

ApplyFine():

    MemberID = GetMemberID()

    FineAmount = CalculateFine(MemberID)

    IF FineAmount > 0:

        Display FineAmount

// Fungsi untuk menampilkan laporan aktivitas

ShowActivityReport():

    Display Activity Report

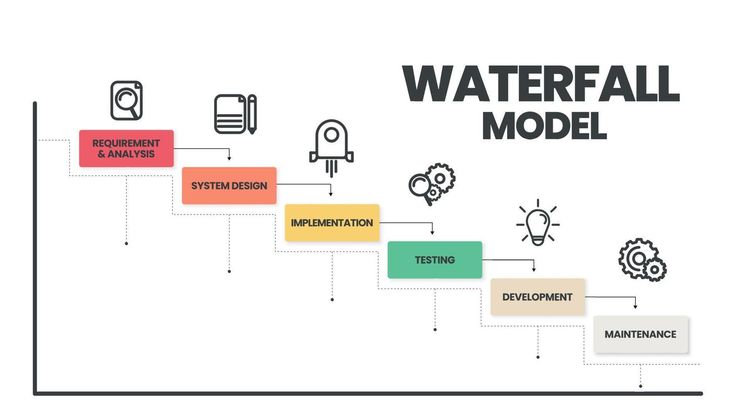
    ShowAllTransactions()

    ShowMemberActivity()

END

# **METODOLOGI PENGEMBANGAN SISTEM**

Metode Waterfall adalah model pengembangan perangkat lunak yang mengikuti pendekatan linier dan berurutan, di mana setiap fase—mulai dari perencanaan, analisis kebutuhan, desain, implementasi, pengujian, hingga pemeliharaan—diselesaikan sepenuhnya sebelum beralih ke fase berikutnya.



***Gambar 2 - Waterfall Model***

***(sumber: vecteezy.com)***

# **HASIL AKHIR**

## **Source Code**

*#include <iostream>*

*#include <stack>*

*#include <vector>*

*#include <string>*

*#include <sstream>*

*#include <iomanip>*

*#include <stdexcept>*

*#include <ctime>*

*using namespace std;*

*struct Book {*

*int id;*

*string title;*

*string author;*

*bool isAvailable;*

*string category;*

*};*

*struct Member {*

*int id;*

*string name;*

*string membershipStatus;*

*stack<int> borrowedBooks;*

*};*

*struct Transaction {*

*int bookId;*

*int memberId;*

*string transactionType; // Type: Borrow, Return, Extend*

*string date;*

*string dueDate;*

*};*

*struct User {*

*string username;*

*string password;*

*string role; // Role: Admin, Petugas, Supervisor*

*};*

*// Global Data*

*vector<Book> books;*

*vector<Member> members;*

*vector<Transaction> transactions;*

*vector<User> users;*

*// Utility functions*

*time\_t parseDate(const string& dateStr) {*

*struct tm tm = {};*

*if (sscanf(dateStr.c\_str(), "%d-%d-%d", &tm.tm\_year, &tm.tm\_mon, &tm.tm\_mday) != 3) {*

*throw std::runtime\_error("Invalid date format: " + dateStr);*

*}*

*tm.tm\_year -= 1900; // Adjust year (tm\_year is years since 1900)*

*tm.tm\_mon -= 1; // Adjust month (tm\_mon is 0-based)*

*return mktime(&tm);*

*}*

*string formatDate(time\_t date) {*

*char buffer[11];*

*strftime(buffer, 11, "%Y-%m-%d", localtime(&date));*

*return string(buffer);*

*}*

*bool login(User& loggedInUser) {*

*string username, password;*

*cout << "Enter username: ";*

*cin >> username;*

*cout << "Enter password: ";*

*cin >> password;*

*for (const auto& user : users) {*

*if (user.username == username && user.password == password) {*

*loggedInUser = user;*

*return true;*

*}*

*}*

*return false;*

*}*

*// Function declarations*

*void registerNewBook();*

*void viewBookCatalog();*

*void searchBook();*

*void updateBookStatus();*

*void deleteBook();*

*void registerNewMember();*

*void viewMember();*

*void searchMember();*

*void updateMemberProfile();*

*void deleteMember();*

*void manageMemberProfile();*

*void viewBorrowHistory();*

*void processBorrowing();*

*void processReturning();*

*void extendBorrowing();*

*void calculateLateFee();*

*void borrowBook();*

*void viewTransactions();*

*void deleteTransaction();*

*void bookManagementMenu() {*

*int choice;*

*do {*

*cout << "\nBook Management\n";*

*cout << "1. Register New Book\n";*

*cout << "2. View Book Catalog\n";*

*cout << "3. Search Book\n";*

*cout << "4. Update Book Information\n";*

*cout << "5. Delete Book\n";*

*cout << "6. Back to Main Menu\n";*

*cout << "Enter your choice: ";*

*cin >> choice;*

*switch (choice) {*

*case 1:*

*registerNewBook();*

*break;*

*case 2:*

*viewBookCatalog();*

*break;*

*case 3:*

*searchBook();*

*break;*

*case 4:*

*updateBookStatus();*

*break;*

*case 5:*

*deleteBook();*

*break;*

*case 6:*

*return;*

*default:*

*cout << "Invalid choice. Please try again.\n";*

*}*

*} while (true);*

*}*

*void registerNewBook() {*

*Book newBook;*

*cout << "Enter book ID: ";*

*cin >> newBook.id;*

*cin.ignore();*

*cout << "Enter book title: ";*

*getline(cin, newBook.title);*

*cout << "Enter author: ";*

*getline(cin, newBook.author);*

*cout << "Enter category: ";*

*getline(cin, newBook.category);*

*newBook.isAvailable = true;*

*books.push\_back(newBook);*

*cout << "Book registered successfully.\n";*

*}*

*void viewBookCatalog() {*

*cout << "\nBook Catalog:\n";*

*cout << "+------------+-----------------------------------------------+-----------------------------------------------+---------------------+----------------------+\n";*

*cout << "| Book ID | Title | Author | Category | Availability |\n";*

*cout << "+------------+-----------------------------------------------+-----------------------------------------------+---------------------+----------------------+\n";*

*bool found = false;*

*for (size\_t i = 0; i < books.size(); ++i) {*

*cout << "| " << setw(10) << left << books[i].id*

*<< " | " << setw(45) << left << books[i].title*

*<< " | " << setw(45) << left << books[i].author*

*<< " | " << setw(19) << left << books[i].category*

*<< " | " << setw(20) << left << (books[i].isAvailable ? "Yes" : "No") << " |\n";*

*found = true;*

*}*

*if (!found) {*

*cout << "| Data books not found.\n";*

*}*

*cout << "+------------+-----------------------------------------------+-----------------------------------------------+---------------------+----------------------+\n";*

*}*

*void searchBook() {*

*string searchQuery;*

*cout << "Enter book title or author to search: ";*

*cin.ignore();*

*getline(cin, searchQuery);*

*bool found = false;*

*cout << "\nSearch Results:\n";*

*cout << "+------------+-----------------------------------------------+-----------------------------------------------+---------------------+----------------------+\n";*

*cout << "| Book ID | Title | Author | Category | Availability |\n";*

*cout << "+------------+-----------------------------------------------+-----------------------------------------------+---------------------+----------------------+\n";*

*for (size\_t i = 0; i < books.size(); ++i) {*

*if (books[i].title.find(searchQuery) != string::npos || books[i].author.find(searchQuery) != string::npos) {*

*found = true;*

*cout << "| " << setw(10) << left << books[i].id*

*<< " | " << setw(45) << left << books[i].title*

*<< " | " << setw(45) << left << books[i].author*

*<< " | " << setw(19) << left << books[i].category*

*<< " | " << setw(20) << left << (books[i].isAvailable ? "Yes" : "No") << " |\n";*

*}*

*}*

*if (!found) {*

*cout << "| No books found matching your search criteria.\n";*

*}*

*cout << "+------------+-----------------------------------------------+-----------------------------------------------+---------------------+----------------------+\n";*

*}*

*void updateBookStatus() {*

*int bookId;*

*cout << "Enter book ID to update: ";*

*cin >> bookId;*

*cin.ignore();*

*for (size\_t i = 0; i < books.size(); ++i) {*

*if (books[i].id == bookId) {*

*cout << "Enter new title (leave blank to keep current): ";*

*string newTitle;*

*getline(cin, newTitle);*

*if (!newTitle.empty()) books[i].title = newTitle;*

*cout << "Enter new author (leave blank to keep current): ";*

*string newAuthor;*

*getline(cin, newAuthor);*

*if (!newAuthor.empty()) books[i].author = newAuthor;*

*cout << "Enter new category (leave blank to keep current): ";*

*string newCategory;*

*getline(cin, newCategory);*

*if (!newCategory.empty()) books[i].category = newCategory;*

*cout << "Is the book available? (1 for Yes, 0 for No): ";*

*cin >> books[i].isAvailable;*

*cout << "Book updated successfully.\n";*

*return;*

*}*

*}*

*cout << "Book with ID " << bookId << " not found.\n";*

*}*

*void deleteBook() {*

*int bookId;*

*cout << "Enter book ID to delete: ";*

*cin >> bookId;*

*for (size\_t i = 0; i < books.size(); ++i) {*

*if (books[i].id == bookId) {*

*books.erase(books.begin() + i);*

*cout << "Book deleted successfully.\n";*

*return;*

*}*

*}*

*cout << "Book with ID " << bookId << " not found.\n";*

*}*

*void memberManagementMenu() {*

*int choice;*

*do {*

*cout << "\nMember Management\n";*

*cout << "1. Register New Member\n";*

*cout << "2. View Member\n";*

*cout << "3. Search Member\n";*

*cout << "4. Update Member Profile\n";*

*cout << "5. Delete Member\n";*

*cout << "6. Back to Main Menu\n";*

*cout << "Enter your choice: ";*

*cin >> choice;*

*switch (choice) {*

*case 1:*

*registerNewMember();*

*break;*

*case 2:*

*viewMember();*

*break;*

*case 3:*

*searchMember();*

*break;*

*case 4:*

*updateMemberProfile();*

*break;*

*case 5:*

*deleteMember();*

*break;*

*case 6:*

*return;*

*default:*

*cout << "Invalid choice. Please try again.\n";*

*}*

*} while (true);*

*}*

*void registerNewMember() {*

*Member newMember;*

*cout << "Enter member ID: ";*

*cin >> newMember.id;*

*cin.ignore();*

*cout << "Enter member name: ";*

*getline(cin, newMember.name);*

*newMember.membershipStatus = "Active";*

*members.push\_back(newMember);*

*cout << "Member registered successfully.\n";*

*}*

*void viewMember() {*

*cout << "\nMember Catalog:\n";*

*cout << "+------------+-----------------------------------------------+---------------------+\n";*

*cout << "| Member ID | Name | Status |\n";*

*cout << "+------------+-----------------------------------------------+---------------------+\n";*

*bool found = false;*

*for (size\_t i = 0; i < members.size(); ++i) {*

*cout << "| " << setw(10) << left << members[i].id*

*<< " | " << setw(45) << left << members[i].name*

*<< " | " << setw(19) << left << members[i].membershipStatus << " |\n";*

*found = true;*

*}*

*if (!found) {*

*cout << "| Data members not found.\n";*

*}*

*cout << "+------------+-----------------------------------------------+---------------------+\n";*

*}*

*void searchMember() {*

*string searchQuery;*

*cout << "Enter member name to search: ";*

*cin.ignore();*

*getline(cin, searchQuery);*

*bool found = false;*

*cout << "\nSearch Results:\n";*

*cout << "+------------+-----------------------------------------------+---------------------+\n";*

*cout << "| Member ID | Name | Status |\n";*

*cout << "+------------+-----------------------------------------------+---------------------+\n";*

*for (size\_t i = 0; i < members.size(); ++i) {*

*if (members[i].name.find(searchQuery) != string::npos) {*

*found = true;*

*cout << "| " << setw(10) << left << members[i].id*

*<< " | " << setw(45) << left << members[i].name*

*<< " | " << setw(19) << left << members[i].membershipStatus << " |\n";*

*}*

*}*

*if (!found) {*

*cout << "| No members found matching your search criteria.\n";*

*}*

*cout << "+------------+-----------------------------------------------+---------------------+\n";*

*}*

*void updateMemberProfile() {*

*int memberId;*

*cout << "Enter member ID to update: ";*

*cin >> memberId;*

*cin.ignore();*

*for (size\_t i = 0; i < members.size(); ++i) {*

*if (members[i].id == memberId) {*

*cout << "Enter new name (leave blank to keep current): ";*

*string newName;*

*getline(cin, newName);*

*if (!newName.empty()) members[i].name = newName;*

*cout << "Enter new membership status (leave blank to keep current): ";*

*string newStatus;*

*getline(cin, newStatus);*

*if (!newStatus.empty()) members[i].membershipStatus = newStatus;*

*cout << "Member updated successfully.\n";*

*return;*

*}*

*}*

*cout << "Member with ID " << memberId << " not found.\n";*

*}*

*void deleteMember() {*

*int memberId;*

*cout << "Enter member ID to delete: ";*

*cin >> memberId;*

*for (size\_t i = 0; i < members.size(); ++i) {*

*if (members[i].id == memberId) {*

*members.erase(members.begin() + i);*

*cout << "Member deleted successfully.\n";*

*return;*

*}*

*}*

*cout << "Member with ID " << memberId << " not found.\n";*

*}*

*void transactionManagementMenu() {*

*int choice;*

*do {*

*cout << "\nTransaction Management\n";*

*cout << "1. Borrow Book\n";*

*cout << "2. Return Book\n";*

*cout << "3. Extend Borrowing\n";*

*cout << "4. Delete Transaction\n";*

*cout << "5. Back to Main Menu\n";*

*cout << "Enter your choice: ";*

*cin >> choice;*

*switch (choice) {*

*case 1:*

*borrowBook();*

*break;*

*case 2:*

*processReturning();*

*break;*

*case 3:*

*extendBorrowing();*

*break;*

*case 4:*

*deleteTransaction();*

*break;*

*case 5:*

*return;*

*default:*

*cout << "Invalid choice. Please try again.\n";*

*}*

*} while (true);*

*}*

*void borrowBook() {*

*int bookId, memberId;*

*cout << "Enter Book ID to borrow: ";*

*cin >> bookId;*

*cout << "Enter Member ID: ";*

*cin >> memberId;*

*for (size\_t i = 0; i < books.size(); ++i) {*

*if (books[i].id == bookId) {*

*if (!books[i].isAvailable) {*

*cout << "Book is currently not available.\n";*

*return;*

*}*

*for (size\_t j = 0; j < members.size(); ++j) {*

*if (members[j].id == memberId) {*

*Transaction newTransaction;*

*newTransaction.bookId = bookId;*

*newTransaction.memberId = memberId;*

*newTransaction.transactionType = "Borrow";*

*time\_t now = time(nullptr);*

*newTransaction.date = formatDate(now);*

*// Set tanggal jatuh tempo (7 hari dari sekarang)*

*time\_t dueDate = now + 7 \* 24 \* 60 \* 60;*

*newTransaction.dueDate = formatDate(dueDate);*

*// Simpan transaksi*

*transactions.push\_back(newTransaction);*

*// Tandai buku sebagai tidak tersedia*

*books[i].isAvailable = false;*

*// Tambahkan buku ke daftar peminjaman anggota*

*members[j].borrowedBooks.push(bookId);*

*cout << "Book borrowed successfully. Due date: " << newTransaction.dueDate << "\n";*

*return;*

*}*

*}*

*cout << "Member not found.\n";*

*return;*

*}*

*}*

*cout << "Book not found.\n";*

*}*

*void processReturning() {*

*int bookId, memberId;*

*cout << "Enter Book ID to return: ";*

*cin >> bookId;*

*cout << "Enter Member ID: ";*

*cin >> memberId;*

*for (auto& transaction : transactions) {*

*if (transaction.bookId == bookId && transaction.memberId == memberId && transaction.transactionType == "Borrow") {*

*transaction.transactionType = "Return";*

*transaction.date = formatDate(time(nullptr));*

*for (auto& book : books) {*

*if (book.id == bookId) {*

*book.isAvailable = true;*

*break;*

*}*

*}*

*for (auto& member : members) {*

*if (member.id == memberId) {*

*if (!member.borrowedBooks.empty() && member.borrowedBooks.top() == bookId) {*

*member.borrowedBooks.pop();*

*}*

*break;*

*}*

*}*

*cout << "Book returned successfully.\n";*

*return;*

*}*

*}*

*cout << "No active borrowing transaction found for the given Book ID and Member ID.\n";*

*}*

*void deleteTransaction() {*

*int bookId, memberId;*

*cout << "Enter Book ID: ";*

*cin >> bookId;*

*cout << "Enter Member ID: ";*

*cin >> memberId;*

*for (size\_t i = 0; i < transactions.size(); ++i) {*

*if (transactions[i].bookId == bookId && transactions[i].memberId == memberId) {*

*transactions.erase(transactions.begin() + i);*

*cout << "Transaction deleted successfully.\n";*

*return;*

*}*

*}*

*cout << "Transaction not found.\n";*

*}*

*void extendBorrowing() {*

*int bookId, memberId;*

*cout << "Enter Book ID: ";*

*cin >> bookId;*

*cout << "Enter Member ID: ";*

*cin >> memberId;*

*for (auto& t : transactions) {*

*if (t.bookId == bookId && t.memberId == memberId && t.transactionType == "Borrow") {*

*time\_t dueDate = parseDate(t.dueDate);*

*dueDate += 7 \* 24 \* 60 \* 60; // Extend by 7 days*

*t.dueDate = formatDate(dueDate);*

*t.transactionType = "Extend";*

*cout << "Borrowing period extended successfully. New due date: " << t.dueDate << "\n";*

*return;*

*}*

*}*

*cout << "Transaction not found or not eligible for extension.\n";*

*}*

*// Reports Menu*

*void reportMenu() {*

*int choice;*

*do {*

*cout << "\nReports\n";*

*cout << "1. Calculate Late Fee\n";*

*cout << "2. View Transactions\n";*

*cout << "3. Back to Main Menu\n";*

*cout << "Enter your choice: ";*

*cin >> choice;*

*switch (choice) {*

*case 1:*

*calculateLateFee();*

*break;*

*case 2:*

*viewTransactions();*

*break;*

*case 3:*

*return;*

*default:*

*cout << "Invalid choice. Please try again.\n";*

*}*

*} while (true);*

*}*

*void viewTransactions() {*

*cout << "\nTransaction History:\n";*

*cout << "+------------+------------+----------------+------------+------------+\n";*

*cout << "| Book ID | Member ID | Transaction | Date | Due Date |\n";*

*cout << "+------------+------------+----------------+------------+------------+\n";*

*for (const auto& t : transactions) {*

*cout << "| " << setw(10) << left << t.bookId*

*<< " | " << setw(10) << left << t.memberId*

*<< " | " << setw(14) << left << t.transactionType*

*<< " | " << setw(10) << left << t.date*

*<< " | " << setw(10) << left << t.dueDate << " |\n";*

*}*

*cout << "+------------+------------+----------------+------------+------------+\n";*

*}*

*void calculateLateFee() {*

*int bookId, memberId;*

*cout << "Enter Book ID: ";*

*cin >> bookId;*

*cout << "Enter Member ID: ";*

*cin >> memberId;*

*for (const auto& t : transactions) {*

*if (t.bookId == bookId && t.memberId == memberId && t.transactionType == "Borrow") {*

*time\_t dueDate = parseDate(t.dueDate);*

*time\_t currentDate = time(nullptr);*

*if (currentDate > dueDate) {*

*int daysLate = (currentDate - dueDate) / (24 \* 60 \* 60);*

*int lateFee = daysLate \* 5000; // Late fee: 5000 per day*

*cout << "Book is late by " << daysLate << " days. Late fee: Rp " << lateFee << "\n";*

*} else {*

*cout << "No late fee. Book is returned on time.\n";*

*}*

*return;*

*}*

*}*

*cout << "Transaction not found or not eligible for late fee calculation.\n";*

*}*

*void showMainMenu(User& loggedInUser) {*

*int choice;*

*do {*

*cout << "\nLibrary Management System\n";*

*if (loggedInUser.role == "Admin") {*

*cout << "1. Book Management\n";*

*cout << "2. Member Management\n";*

*cout << "3. Transactions\n";*

*cout << "4. Reports\n";*

*cout << "5. Exit\n";*

*}*

*else if (loggedInUser.role == "Petugas") {*

*cout << "1. Transactions\n";*

*cout << "2. Exit\n";*

*}*

*else if (loggedInUser.role == "Supervisor") {*

*cout << "1. Book Management\n";*

*cout << "2. Member Management\n";*

*cout << "3. Transactions\n";*

*cout << "4. Reports\n";*

*cout << "5. Exit\n";*

*}*

*cout << "Enter your choice: ";*

*cin >> choice;*

*switch (choice) {*

*case 1:*

*if (loggedInUser.role == "Admin" || loggedInUser.role == "Supervisor") {*

*bookManagementMenu();*

*}*

*else if (loggedInUser.role == "Petugas") {*

*transactionManagementMenu();*

*}*

*break;*

*case 2:*

*if (loggedInUser.role == "Admin" || loggedInUser.role == "Supervisor") {*

*memberManagementMenu();*

*}*

*else if (loggedInUser.role == "Petugas") {*

*transactionManagementMenu();*

*}*

*break;*

*case 3:*

*if (loggedInUser.role == "Admin" || loggedInUser.role == "Supervisor") {*

*transactionManagementMenu();*

*}*

*break;*

*case 4:*

*if (loggedInUser.role == "Admin" || loggedInUser.role == "Supervisor") {*

*reportMenu();*

*}*

*break;*

*case 5:*

*if (loggedInUser.role == "Admin" || loggedInUser.role == "Supervisor") {*

*cout << "Exiting the system.\n";*

*return;*

*}*

*else if (loggedInUser.role == "Petugas") {*

*cout << "Exiting the system.\n";*

*return;*

*}*

*break;*

*default:*

*cout << "Invalid choice. Please try again.\n";*

*}*

*} while (true);*

*}*

*// Main function*

*int main() {*

*users.push\_back({"admin", "admin123", "Admin"});*

*users.push\_back({"petugas", "petugas123", "Petugas"});*

*users.push\_back({"supervisor", "supervisor123", "Supervisor"});*

*User loggedInUser;*

*if (login(loggedInUser)) {*

*cout << "Login successful! Welcome, " << loggedInUser.username << "\n";*

*showMainMenu(loggedInUser);*

*}*

*else {*

*cout << "Login failed! Please check your credentials.\n";*

*}*

*return 0;*

*}*

## **Link Github**

[*https://github.com/akbarfdlh2/Tugas-Kelompok-7*](https://github.com/akbarfdlh2/Tugas-Kelompok-7)

## **Screenshot Aplikasi**

## **Alur Proses Penggunaan Aplikasi**