Project Report on

Library Management System

Submitted in the partial fulfilment of requirement for the award of degree of

Bachelor of Technology

In

Computer Science and Engineering

Batch

(2022-2026)

Submitted To:

Er. Bindu Goyal

Asst. Professor CSE

Submitted By:

Harpreet Saini

12200435

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DAV UNIVERSITY

JALANDHAR PUNJAB-144012

ABSTRACT

The Library Management System (LMS) is a software application designed to automate and streamline the operations of a library. In traditional libraries, managing books, members, and transactions was a manual and time-consuming process, prone to errors and inefficiencies. The LMS addresses these challenges by providing a digital solution that facilitates easy management, tracking, and retrieval of library resources.

This project aims to develop an automated system that can handle all aspects of library management, including book acquisition, cataloging, issuing and returning books, member management, and reporting. The system also incorporates modern features such as barcode scanning, fine calculation, anomaly detection, and machine learning-based recommendations to enhance the efficiency and usability of the library.

The LMS offers user-friendly interfaces for both librarians and members, enabling faster search, easy tracking of book availability, and accurate recording of transactions. It also generates reports and analytics to provide insights into library usage patterns, helping in effective decision-making for procurement and resource allocation.

By replacing manual processes with automation, the Library Management System improves operational efficiency, reduces errors, saves time, and enhances the overall user experience. This project demonstrates how technology can modernize library operations and provide a scalable solution for libraries of any size.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my project guide, Er. Bindu Goyal , for their valuable guidance, constructive feedback, and continuous support throughout the duration of this project. I am also thankful to Dr. Rahul Hans and the Department of Computer Science and Engineering for providing the necessary resources and an encouraging academic environment.

My appreciation extends to all faculty members and classmates whose suggestions and cooperation helped strengthen the quality of this work. Lastly, I am deeply grateful to my family for their constant encouragement and support, which motivated me to complete this project successfully.

DECLARATION

I, Harpreet Saini, hereby declare that the work which is being presented in this project/training titled "LIBRARY MANAGEMENT SYSTEM", in partial fulfilment of the requirements for the award of Bachelor of Technology (B.Tech) Degree in "Computer Science and Engineering" is an authentic record of my own work carried out under the guidance of Er. Bindu Goyal.

To the best of my knowledge, the matter embodied in this report has not been submitted to any other University/ Institute for the award of any degree or diploma.

HARPREET SAINI

12200435

B.Tech CSE

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION

A Library Management System (LMS) is a comprehensive software solution designed to automate the management of libraries. Traditionally, libraries relied on manual processes for cataloging books, tracking borrowing and returning, and maintaining member records. These manual systems were often time-consuming, error-prone, and inefficient, especially in large libraries with thousands of books and multiple users. LMS replaces these traditional methods with a digital approach to streamline library operations.

The primary aim of a Library Management System is to facilitate the efficient management of books and library resources, allowing librarians and users to access, organize, and track information quickly. It integrates all library functions, such as book acquisition, cataloging, circulation, member management, and reporting, into a single unified platform.

1.1 Purpose

The purpose of this project is to develop an Automated Library Management System (LMS) to manage the operations of a library efficiently and accurately. The system aims to:

- Digitally manage books, users, and transactions.
- Reduce manual paperwork and errors.
- Provide real-time availability of books.
- Facilitate book issue, return, and fine calculation automatically.
- Track borrowing trends and flag anomalies.
- Offer recommendations and forecasts for library inventory.

1.2 Scope

The scope of this project includes:

- A desktop-based application with a full-featured GUI.
- Role-based access: Admin and regular users.
- Book management: Adding, updating, searching, and deleting books.
- User management: Adding users, assigning roles, and secure password storage.
- Transaction management: Issuing books, returning books, calculating fines, and tracking overdue books.
- Request handling: Users can request books and admins can approve/deny.
- Optional features: Barcode scanning for ISBN, forecasting, and recommendation system.
- Export/Reporting: Export transaction records in CSV format for analysis.

1.3 Definitions, Acronyms, and Abbreviations

Term / Acronym	Definition
LMS	Library Management System
GUI	Graphical User Interface
ISBN	International Standard Book Number
CSV	Comma-Separated Values
SQL	Structured Query Language
SQLite	Lightweight relational database engine
PIL	Python Imaging Library (used for images)
OpenCV	Open Source Computer Vision Library (optional barcode scanning)

1.4 Need for the System

Traditional libraries rely on manual operations, which are:

- Time-consuming.
- Error-prone.
- Difficult to track real-time availability of books.
- Challenging to maintain borrowing history and fines.

The automated LMS is needed to:

- Simplify library operations.
- Provide accurate and quick information on books and users.
- Automate fine calculation and overdue tracking.
- Forecast demand and suggest popular books.
- Support decision-making for library administration.

CHAPTER 2: LITERATURE SURVEY

2.1 Traditional LMS

- Operates manually or with basic digital record-keeping (Excel/Word).
- Requires librarians to track books manually, calculate fines, and update records.
- High risk of human error and data inconsistency.
- Limited ability to analyze borrowing trends or predict demand.

2.2 Automated LMS

- Modern LMS automates library operations using software.
- Features include digital catalogs, issue/return tracking, and fine calculation.
- Can generate reports, maintain historical records, and allow multi-user access.
- Examples: Koha, Evergreen, OpenBiblio.
- Limitation: Most traditional LMS don’t integrate forecasting or anomaly detection.

2.3 Machine Learning in Library Operations

- Machine learning (ML) can predict demand, detect anomalies, and recommend books.
- ML applications in LMS:
    - Borrowing trend prediction using past transaction data.
    - Recommendation system: Suggest books based on user history or popular subjects.
    - Anomaly detection: Flag unusual borrowing patterns to prevent misuse.
- Integration of ML improves library decision-making, efficiency, and user satisfaction.

CHAPTER 3: PROBLEM STATEMENT

- Manual library systems are inefficient, error-prone, and time-consuming.
- Difficulty in tracking book availability, overdue fines, and borrowing patterns.
- No automated forecasting or recommendation for popular books.
- Library staff spends significant time in data entry, report generation, and manual tracking.
- There is a need for a complete automated LMS that integrates GUI, database, and optional ML features for improved efficiency and decision support.

CHAPTER 4: OBJECTIVES

4.1 What to do

- Develop an Automated LMS to manage books, users, and transactions.
- Implement fine calculation, issue/return management, and reporting.
- Provide role-based access for admins and users.
- Integrate optional ML features for recommendations and forecasting.
- Include optional barcode scanning for ISBN identification.

4.2 Tools and Technologies Used

Component	Tool/Library	Purpose
Programming Language	Python 3.x	Core development
GUI Framework	Tkinter	Fullscreen GUI interface
Database	SQLite	Storing books, users, and transactions
Image Handling	Pillow	Background and cover images
Plotting/Forecast	Matplotlib	Forecast charts and data visualization
Barcode Scanner (Optional)	OpenCV + Pyzbar	Quick ISBN recognition
Data Export	CSV	Export transaction records

4.3 How to do

- Requirement Analysis: Identify system requirements for books, users, and transactions.
- Database Design: Create tables for books, users, transactions, and requests.
- Frontend Development: Build the GUI using Tkinter with responsive frames.
- Backend Development: Implement business logic for book issue/return, fine calculation, and reporting.
- Machine Learning Module (Optional): Implement demand forecasting and recommendations using historical transaction data.
- Testing and Validation: Test all modules for functionality and correctness.
- Deployment: Package the LMS for offline desktop use with configuration options.

4.4 Expected Outcome

- Fully functional LMS with book and user management.
- Automated issue, return, and fine calculation.
- Exportable reports of transactions.
- Optional forecasting and recommendations for better library management.
- User-friendly interface with background customization and navigation.

CHAPTER 5: SYSTEM DESIGN

5.1 Database Schema

1. Books Table

Column	Type	Description
isbn	TEXT (PK)	Unique identifier for the book
title	TEXT	Book title
authors	TEXT	Authors of the book
subject	TEXT	Subject/category
total_copies	INTEGER	Total copies available
issued_copies	INTEGER	Number of copies currently issued
added_on	TEXT	Date book was added

2. Users Table

Column	Type	Description
user_id	TEXT (PK)	Unique identifier for user
name	TEXT	Full name
role	TEXT	Role (admin/user)
password_hash	TEXT	SHA-256 hashed password
created_on	TEXT	Date of user creation

3. Transactions Table

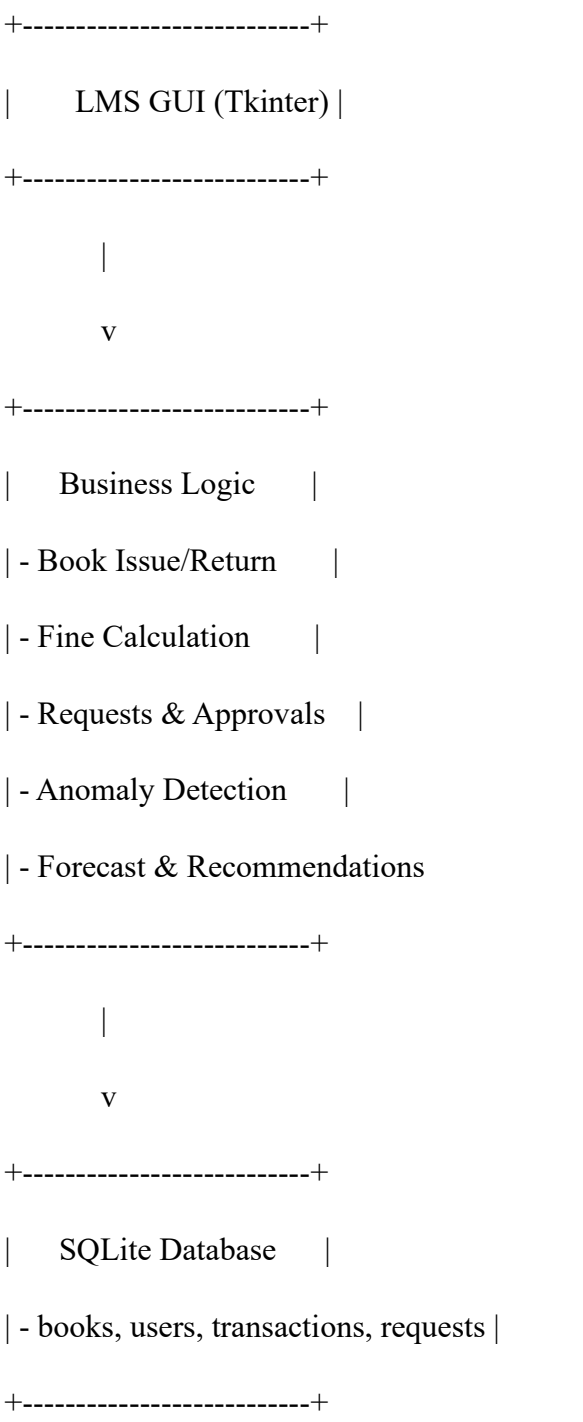
Column	Type	Description
tx_id	INTEGER (PK)	Transaction ID (auto increment)
isbn	TEXT	Book ISBN
user_id	TEXT	User issuing/returning book
issue_date	TEXT	Date of issue
due_date	TEXT	Due date for return
return_date	TEXT	Date of return
fine	REAL	Fine for late return
anomaly_flag	INTEGER	Flag for suspicious activity
copies_issued	INTEGER	Number of copies issued in this transaction

4. Requests Table (For non-admin users requesting books)

Column	Type	Description
request_id	INTEGER (PK)	Request ID (auto increment)
isbn	TEXT	Book requested
user_id	TEXT	User requesting
request_date	TEXT	Date of request
days	INTEGER	Number of days requested
status	TEXT	pending/approved/denied
admin_id	TEXT	Admin handling request
decision_date	TEXT	Date admin processed request

5.2 System Architecture

The LMS follows a client-side desktop architecture:



CHAPTER 6: METHODOLOGY

6.1 Requirement Analysis

- Identify key functional requirements: book management, user management, transactions, requests.
- Identify non-functional requirements: security, scalability, usability.
- Gather inputs from library staff on daily workflows.

6.2 System Design

- Design modular architecture:
    - Frontend GUI module
    - Backend database module
    - Transaction processing module
    - Optional ML module
- Role-based access design: Admin and User.

6.3 Database Development

- Create SQLite database with:
    - Books table (isbn, title, authors, total copies, issued copies).
    - Users table (user_id, name, role, password hash).
    - Transactions table (tx_id, isbn, user_id, issue/return/fine data).
    - Requests table (request_id, isbn, user_id, status).

6.4 Frontend Development

- Develop GUI using Tkinter with:
    - Fullscreen window.
    - Navigation panel with buttons for each module.
    - Dynamic frames for displaying book lists, user lists, and transactions.
    - Forms for issue/return, adding books/users, and exporting data.

6.5 Backend Development

- Implement Python business logic:
    - Issue/return books with availability checks.
    - Calculate fines based on due date and fine per day.
    - Track and update transaction history.
    - Handle book requests (approve/deny by admin).
    - Export data to CSV for reporting.

6.6 Machine Learning Module Implementation

- Implement optional ML features:
    - Demand forecasting: Predict which books will be in demand next month using historical data.
    - Recommendation system: Suggest popular books based on user history and subject.

6.7 Testing and Validation

- Test each module independently:
    - Book addition, issue, return, and request.
    - Fine calculation accuracy.
    - GUI responsiveness and navigation.
    - ML predictions and recommendations (if implemented).
- Validate against expected outcomes.

6.8 Deployment

- Package LMS as a standalone Python application.
- Provide configuration options for:
    - Database path.
    - Background images.
    - Fine per day.
- Deployment supports offline desktop use without internet dependency.

CHAPTER 7: IMPLEMENTATION

CODE :-

```
import os
import sqlite3
import hashlib
import datetime
from collections import defaultdict, deque
import random
import time
import threading
import json
import csv
import tkinter as tk
from tkinter import ttk, messagebox, simpledialog
from tkinter.scrolledtext import ScrolledText
import urllib.request
import urllib.error
try:
    from PIL import Image, ImageTk
except Exception:
    Image = None
try:
    import matplotlib
```

```
    matplotlib.use("TkAgg")
    from matplotlib.figure import Figure
    from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
except Exception:
    Figure = None
try:
    import cv2
    from pyzbar import pyzbar
except Exception:
    cv2 = None
    pyzbar = None
DB_PATH = os.path.join(os.path.dirname(__file__), "lms.db")
CONFIG_PATH = os.path.join(os.path.dirname(__file__), "config.json")
def get_conn():
    conn = sqlite3.connect(DB_PATH, timeout=30, check_same_thread=False)
    conn.row_factory = sqlite3.Row
    try:
        conn.execute("PRAGMA journal_mode=WAL;")
        conn.execute("PRAGMA busy_timeout=30000;")
    except Exception:
        pass
    return conn
def execute_with_retry(cur, sql, params=None, retries=6, delay=0.3):
    for attempt in range(retries):
        try:
```

```
            if params is None:
                cur.execute(sql)
            else:
                cur.execute(sql, params)
            return
        except sqlite3.OperationalError as e:
            if "locked" in str(e).lower() and attempt < retries - 1:
                time.sleep(delay)
                delay *= 1.8
                continue
            raise
def init_db():
    conn = get_conn()
    cur = conn.cursor()
    cur.execute("""
    CREATE TABLE IF NOT EXISTS books(
        isbn TEXT PRIMARY KEY,
        title TEXT,
        authors TEXT,
        subject TEXT,
        total_copies INTEGER,
        issued_copies INTEGER DEFAULT 0,
        added_on TEXT
    )
    """)
```

```
    cur.execute("""
    CREATE TABLE IF NOT EXISTS users(
        user_id TEXT PRIMARY KEY,
        name TEXT,
        role TEXT,
        password_hash TEXT,
        created_on TEXT )
    """)
    cur.execute("""
    CREATE TABLE IF NOT EXISTS transactions(
        tx_id INTEGER PRIMARY KEY AUTOINCREMENT,
        isbn TEXT,
        user_id TEXT,
        issue_date TEXT,
        due_date TEXT,
        return_date TEXT,
        fine REAL DEFAULT 0,
        anomaly_flag INTEGER DEFAULT 0,
        copies_issued INTEGER DEFAULT 0,
        returned INTEGER DEFAULT 0  )
    """)
    cur.execute("""
    CREATE TABLE IF NOT EXISTS requests(
        request_id INTEGER PRIMARY KEY AUTOINCREMENT,
        isbn TEXT,
```

```
        user_id TEXT,
        request_date TEXT,
        days INTEGER,
        status TEXT DEFAULT 'pending',
        admin_id TEXT,
        decision_date TEXT )
    """)
    conn.commit()
    try:
        cur.execute("ALTER TABLE transactions ADD COLUMN copies_issued INTEGER DEFAULT 0")
        conn.commit()
    except Exception:
        pass
    try:
        cur.execute("ALTER TABLE transactions ADD COLUMN returned INTEGER DEFAULT 0")
        conn.commit()
    except Exception:
        pass
    cur.execute("SELECT COUNT(*) FROM users")
    if cur.fetchone()[0] == 0:
        create_user("admin", "Administrator", "admin", "admin")  # default password admin
    conn.close()
def hash_pw(password):
    return hashlib.sha256(password.encode('utf-8')).hexdigest()
def create_user(user_id, name, role, password):
```

```
    conn = get_conn()
    cur = conn.cursor()
    now = datetime.datetime.now().isoformat()
    try:
        execute_with_retry(cur,
            "INSERT OR IGNORE INTO users(user_id,name,role,password_hash,created_on) VALUES(?,?,?,?,?)",
            (user_id, name, role, hash_pw(password), now))
        conn.commit()
    finally:
        conn.close()
def seed_books():
    sample = [
        ("9780131103627", "The C Programming Language", "Kernighan, Ritchie", "Computer Engineering", 5),
        ("9780132350884", "Clean Code", "Robert C. Martin", "Software Engineering", 4),
        ("9780123745149", "Computer Networks", "Andrew S. Tanenbaum", "Computer Engineering", 3),
        ("9780070582014", "Theory of Machines", "R.S. Khurmi", "Mechanical Engineering", 6),
        ("9780070619462", "Mechanics of Materials", "James M. Gere", "Civil Engineering", 3),
       ("9780470901118", "Fundamentals of Electric Circuits", "Alexander and Sadiku", "Electrical Engineering",
4),
        ("9780130449112", "Digital Design", "M. Morris Mano", "Electronics", 2),
        ("9780130606204", "Signals and Systems", "Oppenheim", "Electronics", 3) ]
    conn = get_conn()
    cur = conn.cursor()
    for isbn, title, authors, subject, copies in sample:
        execute_with_retry(cur,
```

```
        "INSERT OR IGNORE INTO books(isbn,title,authors,subject,total_copies,issued_copies,added_on) VALUES(?,?,?,?,?,?,?)",
        (isbn, title, authors, subject, copies, 0, datetime.datetime.now().isoformat()))
    subjects = ["Computer Engineering","Software Engineering","Mechanical Engineering","Civil Engineering","Electrical Engineering","Electronics","Aerospace","Chemical Engineering"]
    for i in range(1, 101):
        gen_isbn = f"GEN{str(i).zfill(6)}"
        title = f"Engineering Reference Volume {i}"
        authors = f"Author {i}"
        subject = subjects[i % len(subjects)]
        copies = random.randint(1, 6)
        execute_with_retry(cur,
            "INSERT OR IGNORE INTO books(isbn,title,authors,subject,total_copies,issued_copies,added_on) VALUES(?,?,?,?,?,?,?)",
            (gen_isbn, title, authors, subject, copies, 0, datetime.datetime.now().isoformat()))
    conn.commit()
    conn.close()
def add_book(isbn, title, authors, subject, copies):
    conn = get_conn()
    cur = conn.cursor()
    try:
        execute_with_retry(cur,
            "INSERT OR REPLACE INTO books(isbn,title,authors,subject,total_copies,issued_copies,added_on) VALUES(?,?,?,?,?,?,?)",
            (isbn, title, authors, subject, copies, 0, datetime.datetime.now().isoformat()))
        conn.commit()
```

```
    finally:
        conn.close()
def get_all_books():
    conn = get_conn()
    cur = conn.cursor()
    cur.execute("SELECT * FROM books")
    rows = cur.fetchall()
    conn.close()
    return rows
def get_book(isbn):
    conn = get_conn()
    cur = conn.cursor()
    cur.execute("SELECT * FROM books WHERE isbn=?", (isbn,))
    row = cur.fetchone()
    conn.close()
    return row
def update_book_copies(isbn, delta_total=0, delta_issued=0):
    conn = get_conn()
    cur = conn.cursor()
    try:
        cur.execute("SELECT total_copies, issued_copies FROM books WHERE isbn=?", (isbn,))
        r = cur.fetchone()
        if not r:
            return False
        total = r[0]
```

```
        issued = r[1]
        total += delta_total
        issued += delta_issued
        if total < 0 or issued < 0 or issued > total:
            return False
        execute_with_retry(cur, "UPDATE books SET total_copies=?, issued_copies=? WHERE isbn=?", (total, issued, isbn))
        conn.commit()
        return True
    finally:
        conn.close()
def issue_book(isbn, user_id, days=14):
    conn = get_conn()
    cur = conn.cursor()
    try:
        cur.execute("SELECT total_copies, issued_copies FROM books WHERE isbn=?", (isbn,))
        r = cur.fetchone()
        if not r:
            return False, "Book not found"
        total = r[0]
        issued = r[1]
        if issued >= total:
            return False, "No copies available"
        issue_date = datetime.date.today()
        due_date = issue_date + datetime.timedelta(days=days)
        new_issued = issued + 1
```

```
        cur.execute("SELECT issued_copies FROM books WHERE isbn=?", (isbn,))
        r = cur.fetchone()
        if r:
            issued = r[0]
            new_issued = max(0, issued - 1)
            execute_with_retry(cur, "UPDATE books SET issued_copies=? WHERE isbn=?", (new_issued, isbn))
        anomaly = detect_anomaly_simple(tx["user_id"])
        if anomaly:
            execute_with_retry(cur, "UPDATE transactions SET anomaly_flag=1 WHERE tx_id=?", (tx_id,))
        conn.commit()
        return True, {"fine": fine, "anomaly": anomaly}
    finally:
        conn.close()
DEFAULT_CONFIG = {
    "fine_per_day": 1.0,
    "seed_count": 100,
    "bg_image": "library_bg.jpg"}
def load_config():
    try:
        with open(CONFIG_PATH, "r", encoding="utf-8") as f:
            cfg = json.load(f)
            for k,v in DEFAULT_CONFIG.items():
                if k not in cfg:
                    cfg[k] = v
            return cfg
```

```
    return count > threshold
def forecast_demand_monthly(isbn, months=3):
    conn = get_conn()
    cur = conn.cursor()
    cur.execute("""
        SELECT issue_date FROM transactions WHERE isbn=? AND issue_date IS NOT NULL
    """, (isbn,))
    rows = cur.fetchall()
    conn.close()
    if not rows:
        return 0
    counts_by_month = defaultdict(int)
    for r in rows:
        d = datetime.date.fromisoformat(r["issue_date"])
        key = (d.year, d.month)
        counts_by_month[key] += 1
    months_sorted = sorted(counts_by_month.items(), reverse=True)
    vals = [v for k, v in months_sorted[:months]]
    if not vals:
        return 0
    return sum(vals) / len(vals)
def recommend_books(subject, limit=5):
    conn = get_conn()
    cur = conn.cursor()
    cur.execute("""
```

```
        SELECT b.isbn, b.title, COUNT(t.tx_id) as borrows
        FROM books b LEFT JOIN transactions t ON b.isbn = t.isbn
        WHERE b.subject = ?
        GROUP BY b.isbn
        ORDER BY borrows DESC
        LIMIT ?
    """, (subject, limit))
    rows = cur.fetchall()
    conn.close()
    return rows
def create_request(isbn, user_id, days=14):
    conn = get_conn()
    cur = conn.cursor()
    try:
        execute_with_retry(cur,
            "INSERT INTO requests(isbn,user_id,request_date,days,status) VALUES(?,?,?,?,?)",
            (isbn, user_id, datetime.date.today().isoformat(), days, "pending"))
        conn.commit()
    finally:
        conn.close()
def get_pending_requests():
    conn = get_conn()
    cur = conn.cursor()
    cur.execute("SELECT * FROM requests WHERE status='pending' ORDER BY request_date")
    rows = cur.fetchall()
```

```
    conn.close()
    return rows
def get_requests_for_user(user_id):
    conn = get_conn()
    cur = conn.cursor()
    cur.execute("SELECT * FROM requests WHERE user_id=? ORDER BY request_date DESC", (user_id,))
    rows = cur.fetchall()
    conn.close()
    return rows
def approve_request(request_id, admin_id):
    conn = get_conn()
    cur = conn.cursor()
    try:
        cur.execute("SELECT * FROM requests WHERE request_id=?", (request_id,))
        req = cur.fetchone()
        if not req:
            return False, "Request not found"
        if req["status"] != "pending":
            return False, "Request already processed"
        ok, msg = issue_book(req["isbn"], req["user_id"], days=int(req["days"] or 14))
        if not ok:
            return False, f"Issue failed: {msg}"
        decision_date = datetime.date.today().isoformat()
            execute_with_retry(cur, "UPDATE requests SET status=?, admin_id=?, decision_date=? WHERE
request_id=?",
            ("approved", admin_id, decision_date, request_id))
```

```
        conn.commit()
        return True, "Approved and issued"
    finally:
        conn.close()
def deny_request(request_id, admin_id):
    conn = get_conn()
    cur = conn.cursor()
    try:
        cur.execute("SELECT * FROM requests WHERE request_id=?", (request_id,))
        req = cur.fetchone()
        if not req:
            return False, "Request not found"
        if req["status"] != "pending":
            return False, "Request already processed"
        decision_date = datetime.date.today().isoformat()
            execute_with_retry(cur, "UPDATE requests SET status=?, admin_id=?, decision_date=? WHERE
request_id=?",
            ("denied", admin_id, decision_date, request_id))
        conn.commit()
        return True, "Denied"
    finally:
        conn.close()
   def _build_ui(self):
        self.nav_frame = tk.Frame(self, width=250, bg="#2c3e50")
        self.nav_frame.pack(side="left", fill="y")
        self.main_frame = tk.Frame(self, bg="#ecf0f1")
```

```
        self.main_frame.pack(side="right", fill="both", expand=True)
        self._set_background()
        btns = [
            ("Login", "show_login"),
            ("Dashboard", "show_dashboard"),
            ("Books", "show_books"),
            ("Requests", "show_requests"),
            ("All Users", "show_all_users"),
            ("Add Book", "show_add_book"),
            ("Create User", "_create_user_prompt"),
            ("Issue Book", "show_issue_book"),
            ("Return Book", "show_return_book"),
            ("Transactions", "show_transactions"),
            ("Export Transactions", "export_transactions"),
            ("Scan Barcode", "scan_barcode_ui"),
            ("Forecast & Recommend", "show_forecast"),
            ("Anomalies", "show_anomalies"),
            ("Settings", "show_settings"),
            ("Logout", "logout"),
            ("Exit", "on_close"),]
        for text, method_name in btns:
            def make_cmd(name):
                return lambda: getattr(self, name)()
            safe_cmd = make_cmd(method_name)
            b = tk.Button(self.nav_frame, text=text, fg="white", bg="#34495e", relief="flat", command=safe_cmd)
```

```
            b.pack(fill="x", padx=10, pady=5)
            self._nav_buttons[text] = b
        self.status_var = tk.StringVar(value="Not logged in")
        status_lbl = tk.Label(self.nav_frame, textvariable=self.status_var, bg="#2c3e50", fg="white")
        status_lbl.pack(side="bottom", pady=10)
        self._refresh_nav()
        self.show_dashboard()
    def clear_main(self):
        for w in list(self.main_frame.winfo_children()):
            if w is self.bg_label:
                continue
            w.destroy()
        self.bg_label = None
        img_name = CONFIG.get("bg_image", "library_bg.jpg")
        img_path = os.path.join(os.path.dirname(__file__), img_name)
        if Image and not os.path.exists(img_path):
            try:
                urllib.request.urlretrieve(DEFAULT_BG_URL, img_path)
            except Exception:
                pass
        if Image and os.path.exists(img_path):
            try:
                self._bg_image_orig = Image.open(img_path).convert("RGBA")
                self.bg_label = tk.Label(self, bd=0)
                self.bg_label.place(x=0, y=0, relwidth=1, relheight=1)
```

```
            self.bg_label.lower()  # move behind all widgets
            self._on_main_resize()
            self.bind("<Configure>", lambda e: self._on_main_resize())
        except Exception:
            self.bg_label = None
    def _on_main_resize(self):
        if not getattr(self, "bg_label", None) or not hasattr(self, "_bg_image_orig"):
            return
        try:
            w = max(1, self.winfo_width())
            h = max(1, self.winfo_height())
            img = self._bg_image_orig.resize((w, h), Image.LANCZOS)
            self._bg_tk = ImageTk.PhotoImage(img)
            self.bg_label.configure(image=self._bg_tk)
            self.bg_label.image = self._bg_tk
            self.bg_label.lower()
        except Exception:
            pass
        cfg = CONFIG.copy()
        def on_save():
            try:
                cfg["fine_per_day"] = float(fine_e.get().strip())
            except Exception:
                messagebox.showerror("Error", "Invalid fine value")
                return
```

```
            cfg["bg_image"] = bg_e.get().strip() or DEFAULT_CONFIG["bg_image"]
            save_config(cfg)
            global CONFIG
            CONFIG.update(cfg)
            self._set_background()
            messagebox.showinfo("Settings", "Saved")
            s.destroy()
        s = tk.Toplevel(self)
        s.title("Settings")
        tk.Label(s, text="Fine per day").grid(row=0, column=0, sticky="e", padx=8, pady=6)
                fine_e = tk.Entry(s); fine_e.grid(row=0, column=1, padx=8, pady=6); fine_e.insert(0,
str(cfg.get("fine_per_day",1.0)))
        tk.Label(s, text="Background image filename").grid(row=1, column=0, sticky="e", padx=8, pady=6)
         bg_e = tk.Entry(s); bg_e.grid(row=1, column=1, padx=8, pady=6); bg_e.insert(0, cfg.get("bg_image",
DEFAULT_CONFIG["bg_image"]))
        tk.Button(s, text="Save", command=on_save).grid(row=2, column=0, columnspan=2, pady=12)
    def logout(self):
        self.current_user = None
        self.status_var.set("Not logged in")
        self._refresh_nav()
        self.show_dashboard()
    def show_login(self):
        self.clear_main()
        frm = self._make_centered_frame(width=560, height=280, bg="white", padx=40, pady=30)
        label_font = ("Arial", 14)
        entry_font = ("Arial", 13)
```

```
        btn_font = ("Arial", 12, "bold")
        tk.Label(frm, text="User ID", font=label_font, bg="white").grid(row=0, column=0, sticky="e", padx=10,
pady=8)
        uid = tk.Entry(frm, font=entry_font, width=28)
        uid.grid(row=0, column=1, padx=10, pady=8)
       tk.Label(frm, text="Password", font=label_font, bg="white").grid(row=1, column=0, sticky="e", padx=10,
pady=8)
        pwd = tk.Entry(frm, show="*", font=entry_font, width=28)
        pwd.grid(row=1, column=1, padx=10, pady=8)
        def do_login():
            user_id = uid.get().strip()
            password = pwd.get().strip()
            conn = get_conn()
            cur = conn.cursor()
            cur.execute("SELECT * FROM users WHERE user_id=?", (user_id,))
            row = cur.fetchone()
            conn.close()
            if row and row["password_hash"] == hash_pw(password):
                self.current_user = dict(row)
                self.status_var.set(f"{self.current_user['name']} ({self.current_user['role']})")
                self._refresh_nav()
                messagebox.showinfo("Login", "Login successful")
                if self.current_user.get("role") == "admin":
                    self.show_admin_home()
                else:
                    self.show_user_home()
```

```
            else:
                messagebox.showerror("Login", "Invalid credentials")
        btn_frame = tk.Frame(frm, bg="white")
        btn_frame.grid(row=2, column=0, columnspan=2, pady=(15,5))
            tk.Button(btn_frame, text="Login", font=btn_font, width=12, command=do_login).pack(side="left",
padx=8)
    def _create_user_prompt(self):
        if not self.current_user or self.current_user.get("role") != "admin":
            messagebox.showerror("Permission", "Only admin can create users")
            return
        uid = simpledialog.askstring("User ID", "Enter user id")
        if not uid:
            return
        name = simpledialog.askstring("Name", "Enter name")
        role = simpledialog.askstring("Role", "admin or user", initialvalue="user")
        pwd = simpledialog.askstring("Password", "Enter password", show='*')
        if uid and name and pwd:
            create_user(uid, name, role, pwd)
            messagebox.showinfo("User", "User created")
            try:
                self._refresh_nav()
            except Exception:
                pass
    def show_dashboard(self):
        self.clear_main()
        frm = self._make_centered_frame(width=None, height=None, bg="white", padx=20, pady=20)
```

```
        tk.Label(frm, text="LIBRARY MANAGEMENT SYSTEM", font=("Arial", 28, "bold"), bg="white").pack(pady=(10,20))
    if not self.current_user:
        btn_frame = tk.Frame(frm, bg="white")
        btn_frame.pack(pady=10)
        tk.Button(btn_frame, text="Admin Login", width=18, height=2, command=self.show_login, bg="#2c3e50", fg="white", font=("Arial", 12, "bold")).pack(side="left", padx=12)
        tk.Button(btn_frame, text="User Login", width=18, height=2, command=self.show_login, bg="#2980b9", fg="white", font=("Arial", 12, "bold")).pack(side="left", padx=12)
        tk.Label(frm, text="Please click Login and enter your credentials.", bg="white").pack(pady=8)
        return
    if self.current_user.get("role") == "admin":
        self.show_admin_home()
    else:
        self.show_user_home()
  def show_admin_home(self):
    self.clear_main()
    frm = self._make_centered_frame(width=None, height=None, bg="white", padx=20, pady=20)
    tk.Label(frm, text="Admin Dashboard", font=("Arial", 20, "bold"), bg="white").pack(pady=(6,12))
    btns = [
        ("All Users", self.show_all_users),
        ("All Transactions", self.show_transactions),
        ("Add Book", self.show_add_book),
        ("View Books", self.show_books)
    ]
    row = tk.Frame(frm, bg="white")
```

```
        row.pack(pady=10)
        for (t, cmd) in btns:
            b = tk.Button(row, text=t, width=18, height=2, command=cmd, font=("Arial", 11))
            b.pack(side="left", padx=8, pady=8)
        self.clear_main()
        frm = self._make_centered_frame(width=820, height=480, bg="white", padx=10, pady=10)
        tk.Label(frm, text="All Users", font=("Arial", 16), bg="white").pack(anchor="w")
        cols = ("user_id", "name", "role", "created_on")
        tree = ttk.Treeview(frm, columns=cols, show="headings", height=18)
        for c in cols:
            tree.heading(c, text=c.capitalize())
        tree.pack(fill="both", expand=True)
        conn = get_conn()
        cur = conn.cursor()
        cur.execute("SELECT user_id, name, role, created_on FROM users ORDER BY created_on DESC")
        for r in cur.fetchall():
            tree.insert("", "end", values=(r["user_id"], r["name"], r["role"], r["created_on"]))
        conn.close()
    def show_books(self):
        self.clear_main()
        frm = self._make_centered_frame(width=980, height=520, padx=10, pady=10)
        tk.Label(frm, text="Books", font=("Arial", 16), bg=frm.cget("bg")).pack(anchor="w")
        sf = tk.Frame(frm, bg=frm.cget("bg"))
        sf.pack(fill="x", pady=(6,6))
        tk.Label(sf, text="Search", bg=frm.cget("bg")).pack(side="left")
```

```
        q_e = tk.Entry(sf, width=40)
        q_e.pack(side="left", padx=6)
        def refresh_tree(q=None):
            for i in tree.get_children():
                tree.delete(i)
            books = get_all_books()
            if q:
                ql = q.lower()
                books = [b for b in books if ql in (b["title"] or "").lower() or ql in (b["authors"] or "").lower() or ql in (b["subject"] or "").lower()]
            for b in books:
                tree.insert("", "end", values=(b["isbn"], b["title"], b["authors"], b["subject"], b["total_copies"], b["issued_copies"]))
        tk.Button(sf, text="Go", command=lambda: refresh_tree(q_e.get().strip())).pack(side="left", padx=4)
        tk.Button(sf, text="Clear", command=lambda: (q_e.delete(0,tk.END), refresh_tree(None))).pack(side="left", padx=4)
        cols = ("isbn","title","authors","subject","total","issued")
        tree = ttk.Treeview(frm, columns=cols, show="headings", height=18)
        for c in cols:
            tree.heading(c, text=c.capitalize())
        tree.pack(fill="both", expand=True)
        refresh_tree(None)
    def show_add_book(self):
        if not self._ensure_admin(): return
        self.clear_main()
        frm = self._make_centered_frame(width=640, height=360, bg="white", padx=20, pady=20)
```

```
        labels = ["ISBN","Title","Authors","Subject","Copies"]
        entries = {}
        for i, l in enumerate(labels):
            tk.Label(frm, text=l, bg="white").grid(row=i, column=0, sticky="e", padx=6, pady=6)
            e = tk.Entry(frm, width=50)
            e.grid(row=i, column=1, padx=6, pady=6)
            entries[l] = e
        def do_add():
            isbn = entries["ISBN"].get().strip()
            title = entries["Title"].get().strip()
            authors = entries["Authors"].get().strip()
            subject = entries["Subject"].get().strip()
            try:
                copies = int(entries["Copies"].get().strip())
            except Exception:
                messagebox.showerror("Error", "Copies must be integer")
                return
            if not isbn or not title:
                messagebox.showerror("Error", "ISBN and Title required")
                return
            add_book(isbn, title, authors, subject, copies)
            messagebox.showinfo("Added", "Book added/updated")
            self.show_books()
        tk.Button(frm, text="Add Book", command=do_add).grid(row=len(labels), column=0, columnspan=2,
pady=10)
    def _ensure_admin(self):
```

```
        if not self.current_user or self.current_user.get("role") != "admin":
            messagebox.showerror("Permission", "Admin access required")
            return False
        return True
        self.clear_main()
        frm = self._make_centered_frame(width=520, height=220, bg="white", padx=20, pady=20)
         tk.Label(frm, text="Transaction ID (tx_id)", bg="white").grid(row=0, column=0, sticky="e", padx=6,
pady=6)
        tx_e = tk.Entry(frm)
        tx_e.grid(row=0, column=1, padx=6, pady=6)
        def do_return():
            try:
                tx_id = int(tx_e.get().strip())
            except Exception:
                messagebox.showerror("Error", "Invalid tx_id")
                return
            ok, info = return_book(tx_id)
            if ok:
                fine = info.get("fine", 0.0)
                anomaly = info.get("anomaly", False)
               messagebox.showinfo("Returned", f"Fine: {fine:.2f}\nAnomaly flagged: {'Yes' if anomaly else 'No'}")
                if self.current_user and self.current_user.get("role") == "admin":
                    self.show_transactions()
                else:
                    self.show_books()
            else:
```

```
            messagebox.showerror("Error", info)
        tk.Button(frm, text="Return", command=do_return).grid(row=1, column=0, columnspan=2, pady=10)
    def show_requests(self):
        if not self.current_user or self.current_user.get("role") != "admin":
            messagebox.showerror("Permission", "Admin access required")
            return
        self.clear_main()
        frm = self._make_centered_frame(width=920, height=520, bg="white", padx=10, pady=10)
        tk.Label(frm, text="Pending Requests", font=("Arial", 16), bg="white").pack(anchor="w")
        cols = ("request_id","isbn","user_id","request_date","days","status")
        tree = ttk.Treeview(frm, columns=cols, show="headings", height=18)
        for c in cols:
            tree.heading(c, text=c.capitalize())
        tree.pack(fill="both", expand=True)
        def refresh():
            for i in tree.get_children():
                tree.delete(i)
            for r in get_pending_requests():
                tree.insert("", "end", values=(r["request_id"], r["isbn"], r["user_id"], r["request_date"], r["days"],
r["status"]))
        def do_approve():
            sel = tree.selection()
            if not sel:
                messagebox.showerror("Select", "Select a request first")
                return
            rid = tree.item(sel[0])["values"][0]
```

```
            ok, msg = approve_request(int(rid), self.current_user["user_id"])
            if ok:
                messagebox.showinfo("Approved", msg)
            else:
                messagebox.showerror("Error", msg)
            refresh()
        def do_deny():
            sel = tree.selection()
            if not sel:
                messagebox.showerror("Select", "Select a request first")
                return
            rid = tree.item(sel[0])["values"][0]
            ok, msg = deny_request(int(rid), self.current_user["user_id"])
            if ok:
                messagebox.showinfo("Denied", msg)
            else:
                messagebox.showerror("Error", msg)
            refresh()
        btns = tk.Frame(frm, bg="white")
        btns.pack(pady=6)
        tk.Button(btns, text="Approve", command=do_approve).pack(side="left", padx=8)
        tk.Button(btns, text="Deny", command=do_deny).pack(side="left", padx=8)
        refresh()
    def show_transactions(self):
        self.clear_main()
```

```
        frm = self._make_centered_frame(width=980, height=520, padx=10, pady=10)
        tk.Label(frm, text="Transactions", font=("Arial", 16), bg=frm.cget("bg")).pack(anchor="w")
        cols = ("tx_id","isbn","user_id","issue_date","due_date","return_date","copies_issued","returned","fine","anomaly")
        tree = ttk.Treeview(frm, columns=cols, show="headings", height=18)
        for c in cols:
            tree.heading(c, text=c.capitalize())
        tree.pack(fill="both", expand=True)
        conn = get_conn()
        cur = conn.cursor()
        cur.execute("SELECT * FROM transactions ORDER BY tx_id DESC LIMIT 200")
        for r in cur.fetchall():
            copies = int(r["copies_issued"]) if "copies_issued" in r.keys() and r["copies_issued"] is not None else 0
            returned_flag = int(r["returned"]) if "returned" in r.keys() and r["returned"] is not None else 0
            returned_text = "Yes" if returned_flag == 1 else "No"
            return_date_display = r["return_date"] if r["return_date"] else "-"
            fine_val = float(r["fine"]) if "fine" in r.keys() and r["fine"] is not None else 0.0
            anomaly_flag = int(r["anomaly_flag"]) if "anomaly_flag" in r.keys() and r["anomaly_flag"] is not None else 0
            anomaly_text = "Yes" if anomaly_flag == 1 else "No"
            tree.insert("", "end", values=(
                r["tx_id"], r["isbn"], r["user_id"],
                r["issue_date"], r["due_date"], return_date_display,
                copies, returned_text, f"{fine_val:.2f}", anomaly_text   ))
        conn.close()
    def scan_barcode_ui(self):
```

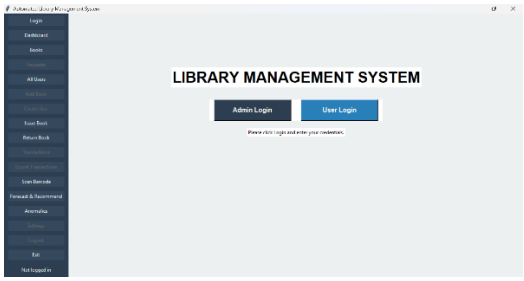
```
        self.clear_main()
        frm = self._make_centered_frame(width=560, height=260, bg="white", padx=20, pady=20)
        tk.Label(frm, text="Barcode Scanner", bg="white").pack()
        txt = tk.StringVar()
        tk.Label(frm, textvariable=txt, font=("Arial", 14), bg="white").pack()
        def do_scan():
            isbn, err = scan_barcode_from_camera()
            if isbn:
                txt.set(f"Detected ISBN: {isbn}")
            else:
                txt.set(f"Scan failed: {err}")
        tk.Button(frm, text="Start Scan", command=do_scan).pack(pady=10)
    def show_forecast(self):
        self.clear_main()
        frm = self._make_centered_frame(width=720, height=420, bg="white", padx=10, pady=10)
        tk.Label(frm, text="Forecast & Recommendations", font=("Arial", 16), bg="white").pack(anchor="w")
        tk.Label(frm, text="ISBN for forecast (leave blank to use first book)", bg="white").pack(anchor="w")
        isbn_e = tk.Entry(frm); isbn_e.pack(anchor="w")
        def do_forecast():
            isbn = isbn_e.get().strip()
            books = get_all_books()
            if not isbn and books:
                isbn = books[0]["isbn"]
            if not isbn:
                messagebox.showerror("Error", "No book selected")
```

```
            return
        f = forecast_demand_monthly(isbn)
        b = get_book(isbn)
        sub = b["subject"] if b else ""
        recs = recommend_books(sub)
        out = f"Forecast monthly demand (avg past months): {f:.2f}\nRecommendations in subject '{sub}':\n"
        for r in recs:
            out += f"- {r['title']} (borrows: {r['borrows']})\n"
        txt = ScrolledText(frm, height=10)
        txt.pack(fill="both", expand=True)
        txt.delete(1.0, tk.END)
        txt.insert(tk.END, out)
    tk.Button(frm, text="Run Forecast", command=do_forecast).pack(pady=5)
def show_anomalies(self):
    self.clear_main()
    frm = self._make_centered_frame(width=880, height=420, padx=10, pady=10)
    tk.Label(frm, text="Anomalies", font=("Arial", 16), bg=frm.cget("bg")).pack(anchor="w")
    cols = ("tx_id","isbn","user_id","issue_date","due_date","return_date","copies_issued","returned","fine")
    tree = ttk.Treeview(frm, columns=cols, show="headings", height=16)
    for h in cols:
        tree.heading(h, text=h.capitalize())
    tree.pack(fill="both", expand=True)
    conn = get_conn()
    cur = conn.cursor()
    cur.execute("SELECT * FROM transactions WHERE anomaly_flag=1 ORDER BY tx_id DESC")
```

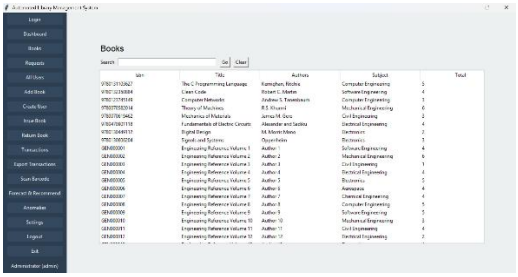
```
        for r in cur.fetchall():
            copies = int(r["copies_issued"]) if "copies_issued" in r.keys() and r["copies_issued"] is not None else 0
            returned_flag = int(r["returned"]) if "returned" in r.keys() and r["returned"] is not None else 0
            returned_text = "Yes" if returned_flag == 1 else "No"
            return_date_display = r["return_date"] if r["return_date"] else "-"
            fine_val = float(r["fine"]) if "fine" in r.keys() and r["fine"] is not None else 0.0
            tree.insert("", "end", values=(
                r["tx_id"], r["isbn"], r["user_id"],
                r["issue_date"], r["due_date"], return_date_display,
                copies, returned_text, f"{fine_val:.2f}"))
        conn.close()
    def on_close(self):
        if messagebox.askokcancel("Quit", "Do you want to exit?"):
            self.destroy()
def main():
    init_db()
    seed_books()
    app = LMSApp()
    app.mainloop()
if __name__ == "__main__":
    main()
```

OUTPUT:-

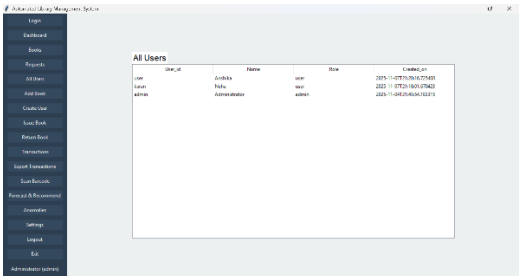
Login page:



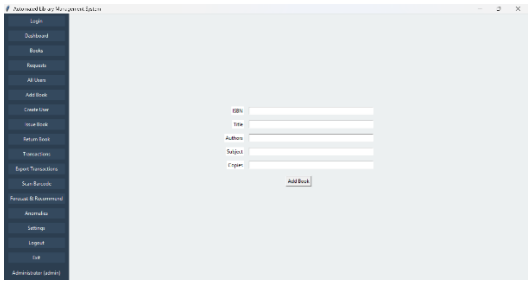
Books available:



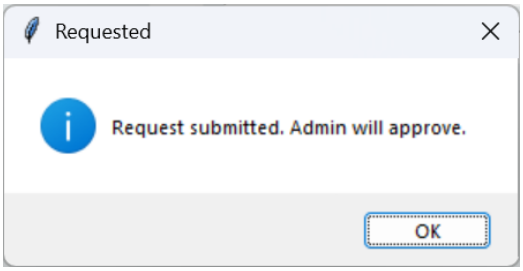
All users:



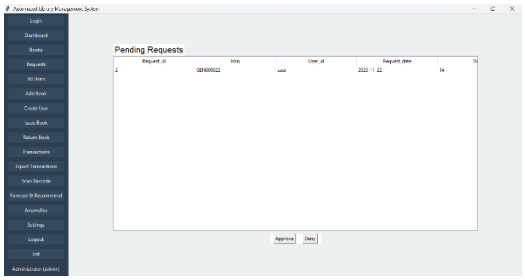
Add books:



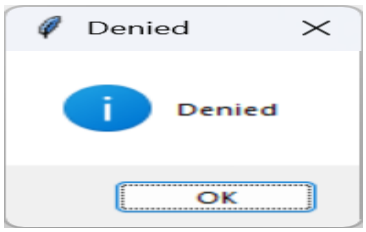
Request sent to issue book:



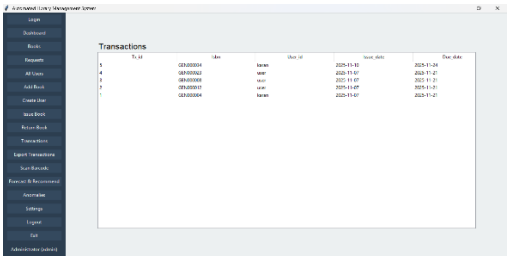
Pending requests:



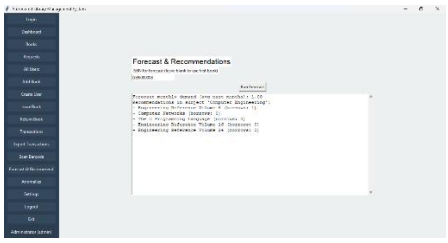
Request Accept/Deny:



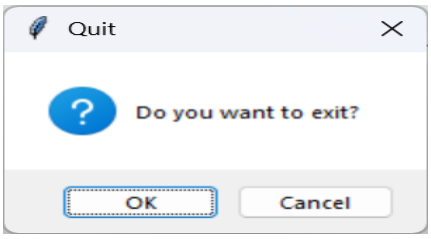
Transactions:



Forcast & Recommendations:



Exit:



CONCLUSION

The Automated Library Management System (LMS):

- Provides an efficient solution for library operations.
- Reduces human error and manual work in tracking books, users, and transactions.
- Offers automation of fines, requests, and reports.
- Optional machine learning features enhance forecasting, recommendations, and anomaly detection.
- The system is modular, scalable, and user-friendly, serving as a comprehensive platform for library management.

REFERENCES

1. Korth, H., & Silberschatz, A. *Database System Concepts*. McGraw-Hill, 2011.
2. Shelly, G. B., & Cashman, T. J. *Systems Analysis and Design*. Cengage, 2018.
3. Kumar, P., & Singh, R. "Design and Implementation of Library Management System using Database." *Int. Journal of Computer Applications*, 2017.
4. Gupta, S., & Arora, A. "Automation in Libraries using Machine Learning." *Int. Journal of Advanced Research in CS*, 2019.
5. GeeksforGeeks. *Library Management System in Python*.

   https://www.geeksforgeeks.org/library-management-system-in-python/
6. TutorialsPoint. *Python Tkinter GUI Programming*. https://www.tutorialspoint.com/python/python_gui_programming.htm