1. System architecture
The Library Management System follows a monolithic, layered architecture, deployed as a single Python application containing GUI using a SQLite database.
Internal Logic has 3 layers:

1.1 Presentation Layer (GUI Layer)
Implemented using Python Tkinter, providing navigation between pages. The pages include: Home Page, Search Books, Manage Loans, Manage Borrowers, Manage Fines. Use a custom theme to style colors, font, and spacing. Implemented convenience functionalities such as: Autofill ISBN when selecting a book, Autofill Card ID when creating a borrower, Select to copy for borrower card IDS, Click to copy ISBN. The presentation layer communicates entirely through library_app.py

1.2 Application Logic Layer
This layer is encapsulated within library_app.py, it mediates between the GUI and the SQlite database. It's responsible
Transaction Management: Logic for checking books in and out, ensuring that database constraints are enforced programmatically before database writes occur.
Algorithm Implementation: specifically the logic for fine calculation, which involves date arithmetic to determine overdue days relative to the current system date.
Data Transformation: formatting raw database rows into dictionaries or tuples suitable for GUI display (example: generating the "No." sequence for search results).

1.3 Data Persistence Layer
Storage**:** Persistent storage of normalized data across tables (BOOK, AUTHORS, BOOK_AUTHORS, BORROWER, BOOK_LOANS, FINES).
Constraint Enforcement: Strict enforcement of primary Keys, foreign Keys, and unique constraints so data integrity is kept
Query Execution**:** queries to retrieve and manipulate data.

2. Design Decisions
Monolithic: simplicity
Separation of Logic: easier debugging and teamwork, separate front-end and back-end
Theme: centralized design, color palette, button styling, fonts, highlighting, tree view
Enhancements: Card ID copied on click, Autofill information

3.1 Database: SQLite
Portability: The database file can be easily moved, backed up, or reset, simplifying the testing and grading process.
Concurrency Needs: As the system is designed for a librarian (single user context), the high concurrency features of Postgres weren't needed.

3.2 Programming Language and GUI Framework: Python & Tkinter
Decision: Python was selected as the core language, utilizing Tkinter for the GUI.

Benefits:
Standard Library Dependency: Tkinter is included with standard Python installations.
This removes the need for third-party dependencies, making sure the application is as
easy as possible to build and run on any machine with Python installed.Integration: Python has
the sqlite3 module, which is able to provide a DBI 2.0 interface
to SQLite.This makes connecting the logic portion and permanent data storage portion
making the application easy.

3.3 Fine Calculation: Batch Processing
Fines are calculated with an update operation instead of using database triggers or a
real time calculation for every read.
Benefits:
Performance: Calculating the fines needs calculations that take up a lot of time and
processing power. Doing these hard calculations every single time that the fines tab is
opened would be inefficient and would likely make the performance of the database
bad. This would get worse as the dataset grows larger. Doing this for every row every
time the Fines tab is opened could degrade performance as the dataset grows.
Business Logic Complexity: The requirement to assess fines for both returned (static
date difference) and non-returned (dynamic date difference) books are complex. A Python
triggered SQL UPSERT (Insert on Conflict Update) allows for an operation that
refreshes the state of the library in one action.

4. Assumptions
A borrower may have max 3 active loans
Fines must be paid only after returned books
All ISBNs are stored as 10 digit

5. Authentication
Authentication in the system is handled through a dedicated USER table that stores unique usernames and passwords required to access the application. Because the original dataset did not include credentials, existing borrowers are automatically assigned login information by using their card_id as the username and their SSN as the password, ensuring backward compatibility without altering the provided CSV files. A sign-up interface allows new users to create accounts by supplying valid identifying information, and the system verifies that usernames remain unique and properly associated with existing borrower records. The authentication process integrates directly with the GUI as the initial entry point, validating credentials before granting access to the main pages.