# Library Management System Documentation

# Table of contents

# Project Overview

## Project Name: Maids.cc Library Management System

## Description

The Maids.cc Library Management System is a robust API built using Spring Boot. It facilitates efficient management of books, patrons, and borrowing records in a library setting. Librarians can easily handle book additions, updates, and removals, as well as patron management and borrowing transactions.

## Entities

Book: Represents information about each book in the library, including attributes such as ID, title, author, publication year, ISBN, etc. Patron: Contains details about library patrons, including ID, name, contact information, etc. Borrowing Record: Tracks the association between books and patrons, including borrowing and return dates.

## API Endpoints

### Book Management Endpoints:

1. GET /api/books: Retrieve a list of all books.
2. GET /api/books/: Retrieve details of a specific book by ID.
3. POST /api/books: Add a new book to the library.
4. PUT /api/books/: Update an existing book's information.
5. DELETE /api/books/: Remove a book from the library.

### Patron Management Endpoints:

1. GET /api/patrons: Retrieve a list of all patrons.
2. GET /api/patrons/: Retrieve details of a specific patron by ID.
3. POST /api/patrons: Add a new patron to the system.

4. PUT /api/patrons/: Update an existing patron's information.

5. DELETE /api/patrons/: Remove a patron from the system.

## Borrowing Endpoints:

1. POST /api/borrow//patron/: Allow a patron to borrow a book.

2. PUT /api/return//patron/: Record the return of a borrowed book by a patron.

# How To Use

To interact with the API endpoints, you can use tools like Postman, cURL, or any HTTP client library in your preferred programming language. Here's how you can use each endpoint:

**Book Management Endpoints**:

Send GET requests to `/api/books` to retrieve a list of all books or `/api/books/{id}` to retrieve details of a specific book by its ID.

Use POST requests to `/api/books` to add a new book to the library. Provide the book details in the request body.

Use PUT requests to `/api/books/{id}` to update an existing book's information. Provide the book ID in the URL and the updated book details in the request body.

Send DELETE requests to `/api/books/{id}` to remove a book from the library by its ID.

**Patron Management Endpoints:**

Similar to book management, use GET, POST, PUT, and DELETE requests to interact with patron management endpoints. Replace `/api/books` with `/api/patrons` in the endpoint URLs.

**Borrowing Endpoints:**

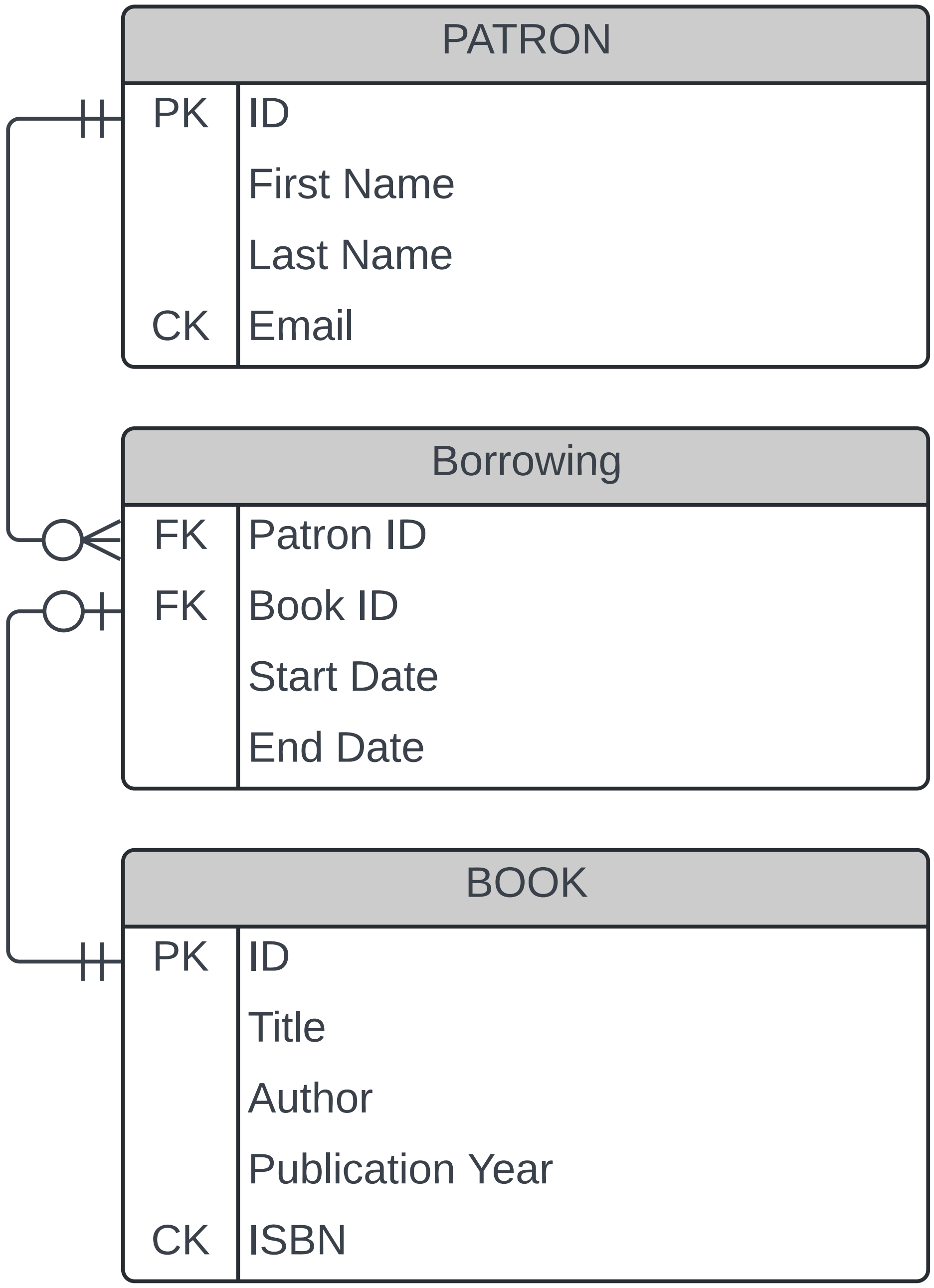
To allow a patron to borrow a book, send a POST request to `/api/borrow/{bookId}/patron/{patronId}`. Replace `{bookId}` and `{patronId}` with the IDs of the book and patron, respectively.

To record the return of a borrowed book, send a PUT request to `/api/return/{bookId}/patron/{patronId}` with the book and patron IDs in the URL.

# Database

The library management system is designed to streamline the organization and retrieval of library resources. At the heart of this system lies a robust database schema, implemented using Microsoft SQL Server, which ensures efficient data management and integrity. The schema is composed of three primary entities: PATRON, BOOK, and BORROWING. The PATRON entity captures essential information about library members, including a unique ID, personal details, and contact information. The BOOK entity records the details of the library's book inventory, such as title, author, and publication year, each identified by a unique ID. The BORROWING entity serves as the linchpin that connects patrons to the books they borrow, tracking the borrowing period with start and end dates. This relational schema facilitates a one-to-many relationship between patrons and books, allowing for multiple borrowings while maintaining a clear record of transactions. By leveraging the capabilities of SQL Server, the system provides a scalable and reliable foundation for managing the library's operations.

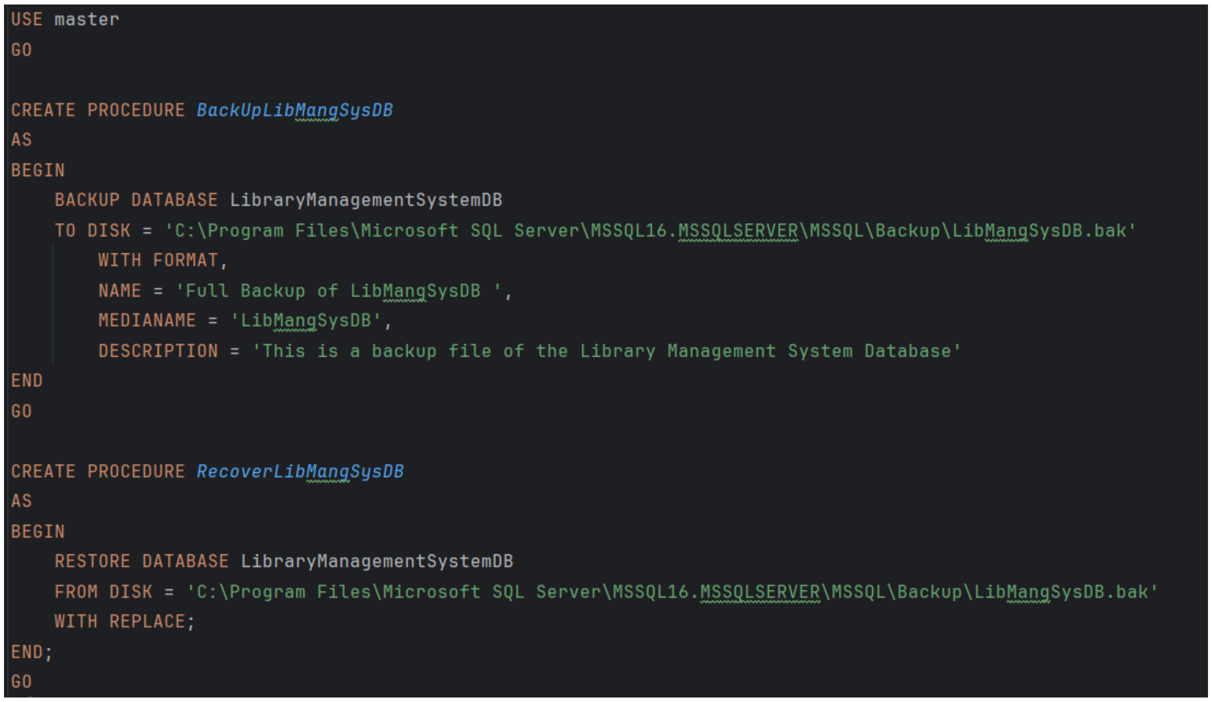
## ERD



ERD.svg

## Backup And Recovery

The backup and recovery aspect of the DB was implemented to allow the DBA the opportunity to backup the current state of the DB and copy it to any external secondary storage medium.

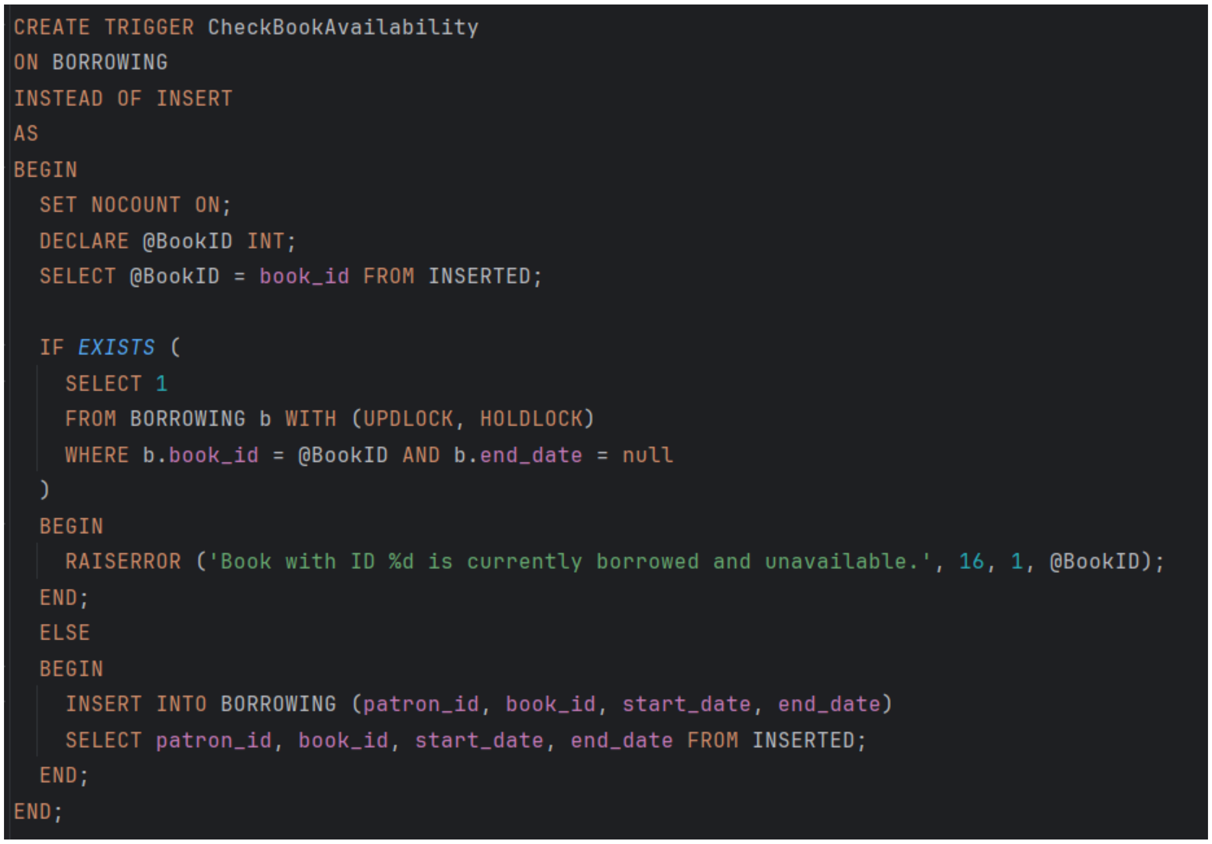


```sql
USE master
GO

CREATE PROCEDURE BackUpLibMangSysDB
AS
BEGIN
    BACKUP DATABASE LibraryManagementSystemDB
    TO DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\Backup\LibMangSysDB.bak'
        WITH FORMAT,
        NAME = 'Full Backup of LibMangSysDB ',
        MEDIANAME = 'LibMangSysDB',
        DESCRIPTION = 'This is a backup file of the Library Management System Database'
END
GO

CREATE PROCEDURE RecoverLibMangSysDB
AS
BEGIN
    RESTORE DATABASE LibraryManagementSystemDB
    FROM DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\Backup\LibMangSysDB.bak'
    WITH REPLACE;
END;
GO
```

backup_recovery.png

## Borrowing Trigger

A Trigger has been developed to insure the DB consistency when adding a borrow record, that is, we need to ensure the book being borrowed is not currently borrowed by someone else.

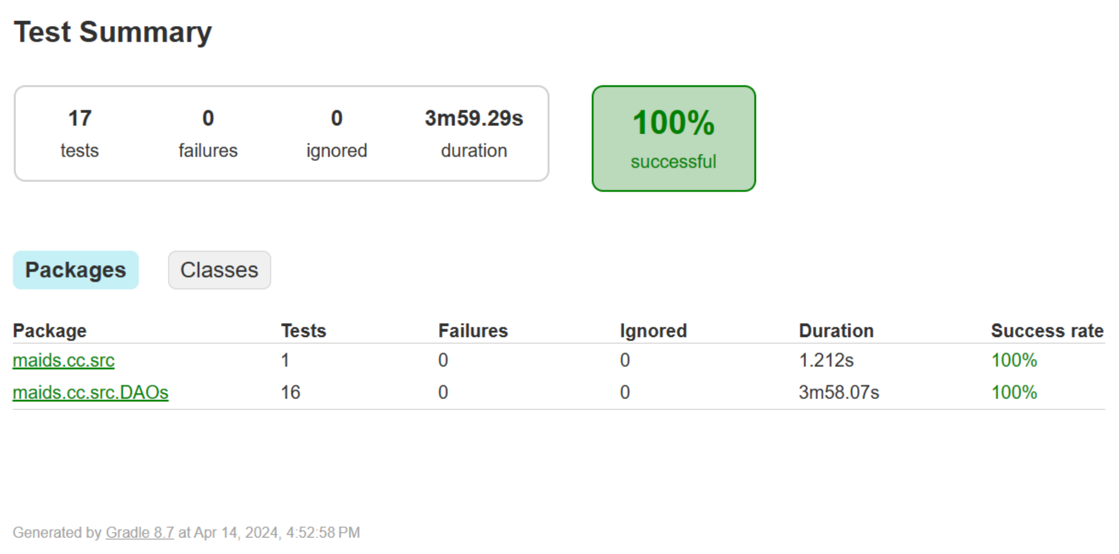


```sql
CREATE TRIGGER CheckBookAvailability
ON BORROWING
INSTEAD OF INSERT
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @BookID INT;
  SELECT @BookID = book_id FROM INSERTED;

  IF EXISTS (
    SELECT 1
    FROM BORROWING b WITH (UPDLOCK, HOLDLOCK)
    WHERE b.book_id = @BookID AND b.end_date = null
  )
  BEGIN
    RAISERROR ('Book with ID %d is currently borrowed and unavailable.', 16, 1, @BookID);
  END;
  ELSE
  BEGIN
    INSERT INTO BORROWING (patron_id, book_id, start_date, end_date)
    SELECT patron_id, book_id, start_date, end_date FROM INSERTED;
  END;
END;
```

borrow_trigger.png

# Testing

Due to the short time window given, unfortunately I was not able to cover multiple classes in the testing process. However, I was able to cover only 2 of the 3 DAOs implemented, and no Endpoints were tested professionally.



Test Summary

| 17 | 0 | 0 | 3m59.29s | 100% |
|---|---|---|---|---|
| tests | failures | ignored | duration | successful |

Packages Classes

| Package | Tests | Failures | Ignored | Duration | Success rate |
|---|---|---|---|---|---|
| maids.cc.src | 1 | 0 | 0 | 1.212s | 100% |
| maids.cc.src.DAOs | 16 | 0 | 0 | 3m58.07s | 100% |

Generated by Gradle 8.7 at Apr 14, 2024, 4:52:58 PM

testing.png

## Backend Server Architecture

The Maids.cc Library Management System adopts a layered architecture design to ensure modularity, flexibility, and maintainability. This architectural style separates the system into distinct layers, each with its own responsibility and abstraction level. The architecture can be decomposed into 3 layers. Listing them from the top most to the down most layer: Endpoints Layers, Business Layer, DAOs Layer, and finally the real DBMS layer. The business logic layer, encapsulating the application's rules and workflows, ensuring independence from specific endpoints logic. Supporting these layers is the data access and modelling layer, responsible for data storage, retrieval, and manipulation, abstracting away the complexities of data management. This layered approach promotes separation of concerns, enabling easier testing, scalability, and future extensibility of the system's components.

The class diagram can be found here (https://github.com/GeorgeBeshay/library-management-system/blob/enhancements/Design/Class%20Diagram.pdf).

## Future Work

Unfortunately, this software has been designed, implemented, tested and documented in less than 24 hrs. So there are few things that the author was not able to achieve during the short time window given. In this section I'll be mentioning few things that will probably be added in the future to this software.

1. Further Testing for the BorrowingDAO, and the application Endpoints.
2. Extending the current documentation to include more examples and discuss some of the decisions that were made plus the intuition behind them.

# Helpful Resources

This section of the software documentation, I'll be attaching few online free resources that I've found helpful in some point during the software development.

1. Logging With AOP in Spring (https://www.baeldung.com/spring-aspect-oriented-programming-logging)
2. Implementing Logging Using Aspect-Oriented Programming (AOP) - Spring: .. (https://youtu.be/fdRKqzEWJqw?si=fdQthyFrDDmzvbPf)