Library Management System

1 Setup Guide

Prerequisites

Ensure the following tools are installed:

- **.NET 8 SDK** – dotnet.microsoft.com/download
- **SQL Server Developer Edition** – microsoft.com/sql-server
- **SQL Server Management Studio (SSMS)** – aka.ms/ssmsfullsetup

Required NuGet Packages

Before running or building the project, ensure the following packages are installed:

- `Microsoft.EntityFrameworkCore`
- `Microsoft.EntityFrameworkCore.SqlServer`
- `Microsoft.EntityFrameworkCore.Tools`
- `Swashbuckle.AspNetCore`

Install them using the .NET CLI:

```
dotnet add package Microsoft.EntityFrameworkCore
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Tools
dotnet add package Swashbuckle.AspNetCore
```

Step-by-Step Setup

1. Clone the Project

```
git clone https://github.com/XApple15/LibraryManagementSystem.git
cd LibraryManagementSystem
```

2. Configure SQL Server Connection

Open `appsettings.json` and set the connection string:

```
"ConnectionStrings": {
  "LibraryDb": "Server=localhost;Database=LibraryDB;Trusted_Connection=
     True;TrustServerCertificate=True"
}
```

Adjust server name and database as needed. No need to create the Database. It is created automatically if it doesn‘t exist.

3. Run the Application

```
dotnet run
```

Visit the API Swagger UI at: `https://localhost:7232/swagger`

Sample API Endpoints

Method	Endpoint	Description
GET	/api/book	List all books
POST	/api/book	Add a new book
PUT	/api/book/{id}	Update an existing book
GET	/api/book/search?title=.....&author=.....&inStock=..	Search books by title, author, or stock
POST	/api/book/lend/{id}	Lend a book
POST	/api/book/return/{id}	Return a book
GET	/api/book/{id}/recommendations	View similar book recommendations

2 Innovative Functionality: AI Book Similarity Recommendation

This system includes a lightweight recommendation engine that provides a list of similar books based on basic string similarity logic applied to titles and authors.

Algorithm Description

Given a selected book identified by its `bookId`, the system performs the following steps:

1. Retrieve the target book from the database.
2. Fetch all books and exclude the target from the comparison set.
3. For each candidate book:
   - Normalize and tokenize the title and author fields.
   - Compare words in the title and author.
   - Compute a score based on the number of intersecting words and an exact author match.
4. Filter out books with a score of 0.
5. Sort the results in descending order of similarity.
6. Return the top 5 most similar books.

API Endpoint Example

```
GET /api/books/{bookId}/recommendations
```

Normalization and Tokenization

For both the title and author fields of the target and candidate books:

- Convert text to lowercase
- Split text into words using whitespace

Similarity Scoring Function

Each pair of books receives a similarity score computed as follows:

- **+5 points** if the author names are an exact match
- **+1 point per shared word** in the book titles (case-insensitive)
- **+1 point per shared word** in the author names

C# Equivalent:

```
if (a.Author == b.Author) score += 5;
score += titleWordsA.Intersect(titleWordsB).Count();
score += authorWordsA.Intersect(authorWordsB).Count();
```

Output

The final output is a ranked list of up to 5 books, sorted by similarity score in descending order. Only books with a score greater than 0 are included.