

Practice Assignment 4

LIBRARY MANAGEMENT SYSTEM

Following Practice Assignment 3, further develop the new model in the .Net Core framework.

Exercise 1:

Design a database using SQL Server and migrate it to the model in the project with the tables listed below.

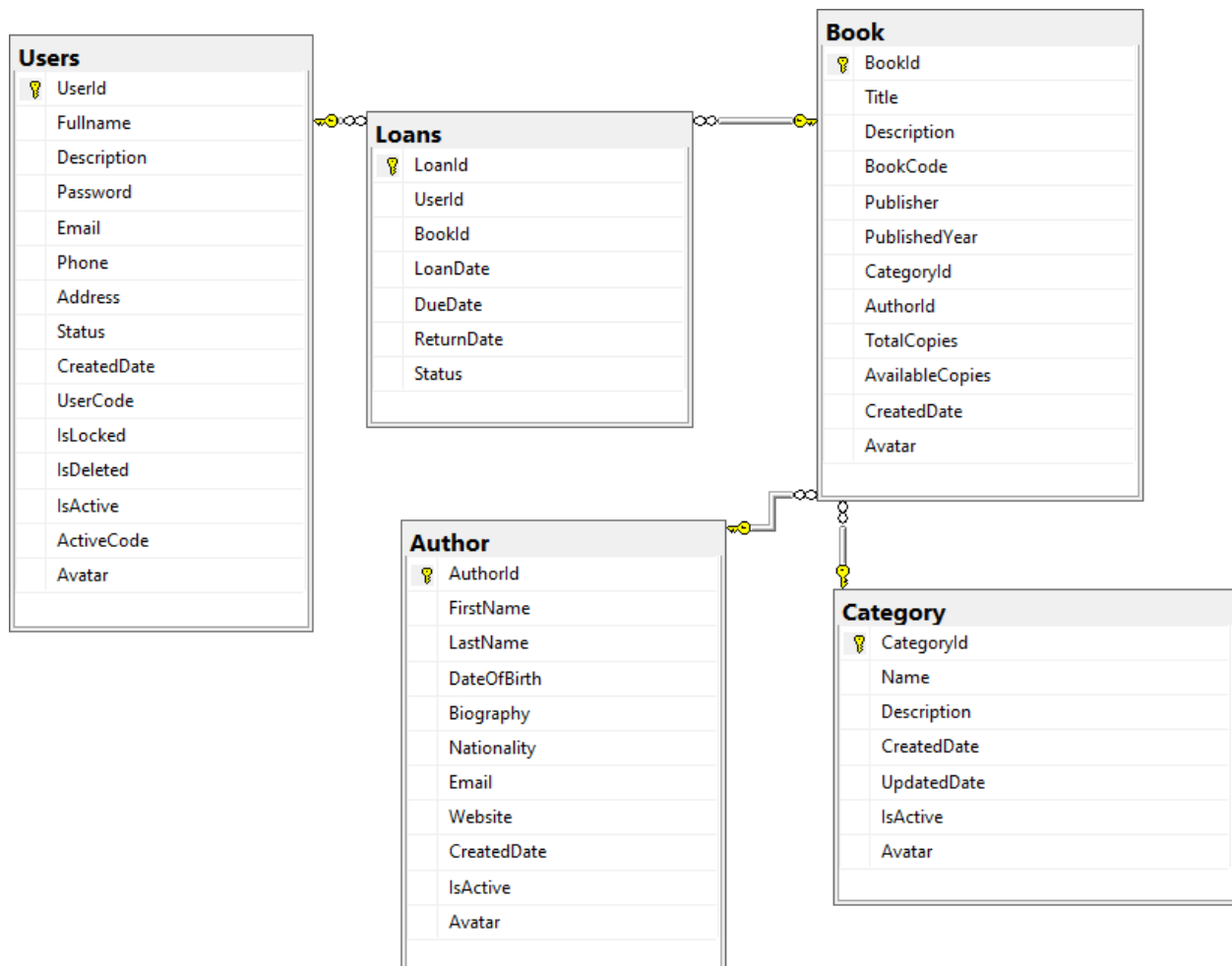


Figure 2: Diagram of the library management database

Users:

User			
#	Col_Name	Data Type	Descriptions
1	UserId	int	Unique identifier for each user, auto-incremented.
2	Fullname	nvarchar(200)	Full name of the user.
3	Description	nvarchar(MAX)	Additional info about the user, like interests or bio.
4	Password	nvarchar(MAX)	Hashed password for secure authentication.
5	Email	nvarchar(100)	User's email address, used for communication and verification.
6	Phone	nvarchar(20)	User's contact phone number.
7	Address	nvarchar(MAX)	User's physical address for mailing or location purposes.
8	Status	int	Represents the user's status.
9	CreatedDate	datetime	Date and time when the user account was created.
10	UserCode	nvarchar(MAX)	Unique code for internal identification of the user.
11	IsLocked	bit	Indicates if the account is locked (1) or active (0).
12	IsDeleted	bit	Marks the account as deleted (1) or active (0) for soft deletion.
13	IsActive	bit	Shows if the account is active (1) or inactive (0).

14	ActiveCode	nvarchar(MAX)	Code for account activation, sent via email during registration.
15	Avatar	nvarchar(MAX)	Avatar's User - Local location in the server to get the picture.

Loans:

Loans			
#	Col_Name	Data Type	Descriptions
1	LoanId	int	(Primary Key, Auto-Increment)
2	UserId	int	References the user who borrowed the book.
3	BookId	int	References the borrowed book.
4	LoanDate	datetime	Date when the book was borrowed.
5	DueDate	datetime	Date when the book is due.
6	ReturnDate	datetime	Date when the book was returned.
7	Status	int	Status of the loan (e.g., "Active", "Returned", "Overdue"). 0: Active , 1: Returned , 2: Overdue

Books:

Books			
#	Col_Name	Data Type	Descriptions
1	BookId	int	(Primary Key, Auto-Increment)
2	Title	nvarchar(200)	Title of the book.
3	Description	nvarchar(MAX)	Description of the book.
4	BookCode	nvarchar(MAX)	Standard Book Number.
5	Publisher	nvarchar(MAX)	

6	PublishedYear	datetime	Year the book was published.
7	CategoryId	int	References the category.
8	AuthorId	int	References the author.
9	TotalCopies	int	Total number of physical copies of the book in the library
10	AvailableCopies	int	Total number of physical copies of the book currently in the library, excluding copies on loan
11	CreatedDate	datetime	Date when the book record was created.
12	Avatar	nvarchar(MAX)	Cover image of the book - Local location in the server to get the picture.
13	Pdf	nvarchar(MAX)	Store the version pdf for reading online

Authors:

Authors			
#	Col_Name	Data Type	Descriptions
1	AuthorId	int	(Primary Key, Auto-Increment). Unique identifier for each author.
2	FirstName	nvarchar(100)	Author's first name.
3	LastName	nvarchar(100)	Author's last name.
4	DateOfBirth	datetime	Author's date of birth.
5	Biography	nvarchar(MAX)	A short biography of the author.
6	Nationality	nvarchar(100)	Nationality of the author.

7	Email	nvarchar(100)	Author's Email.
8	Website	nvarchar(100)	Author's Website.
9	CreatedDate	datetime	Date when the author record was created.
10	IsActive	bit	The IsActive column helps indicate whether a record is currently active and usable within the application.
11	Avatar	nvarchar(MAX)	Authors' Avatar - Local location in the server to get the picture.

Categorys:

Categorys			
#	Col_Name	Data Type	Descriptions
1	CategoryId	int	Unique identifier for each category (Primary Key).
2	Name	Nvarchar (MAX)	Name of the category; must be unique for identification.
3	Description	Nvarchar (MAX)	Additional information about the category.
4	CreatedDate	datetime	Date and time when the category was created.
5	UpdatedDate	datetime	Date and time when the category was last updated.
6	IsActive	bit	Indicates if the category is active (1) or inactive (0).
7	Avatar	Nvarchar (MAX)	Categorys's image - Local location in the server to get the picture.

Note:

Use Data Annotations for Validation:

- Implement data annotations such as [Required], [StringLength], [EmailAddress], etc., to enforce validation rules on model properties.
- Ensure that validation attributes reflect business rules and requirements.

Implement Navigation Properties:

- Define relationships between models using navigation properties (e.g., one-to-many, many-to-many).
- Use collections to represent related entities (e.g., ICollection<Product> in a Category model).

Create a DbContext Class:

- Define a DbContext class that represents the session with the database.
- Include DbSet<T> properties for each model to facilitate data access.

Exercise 2:

Add a new database table named **Carousel** and implement the corresponding model in your backend project to support displaying images or promotional content on the homepage of your Library Management System. The data should be retrieved dynamically from the database through APIs.

Purpose:

- Provide backend support for managing and serving carousel items (such as images and promotional content) displayed on the homepage.

Fields:

- **CarouselId**: int (Primary Key)
- **ImageUrl**: nvarchar(MAX) (Required) - URL of the image.
- **Title**: nvarchar(200) (Required) - Title for the item.
- **Description**: nvarchar(MAX) (Optional) - Additional details.
- **LinkUrl**: nvarchar(MAX) (Optional) - URL linked to the item.
- **Order**: int (Required) - Display order of items.
- **IsActive**: bit - Indicates if the item is active.
- **CreatedDate**: datetime - Timestamp of creation.
- **UpdatedDate**: datetime - Timestamp of last update.

Guide:

To serve image files from the server in your .NET Core 8.0 project, you can configure your application to serve static files from a specific directory on the server. Here's how to set it up:

Step 1: Store Images in a Server Directory

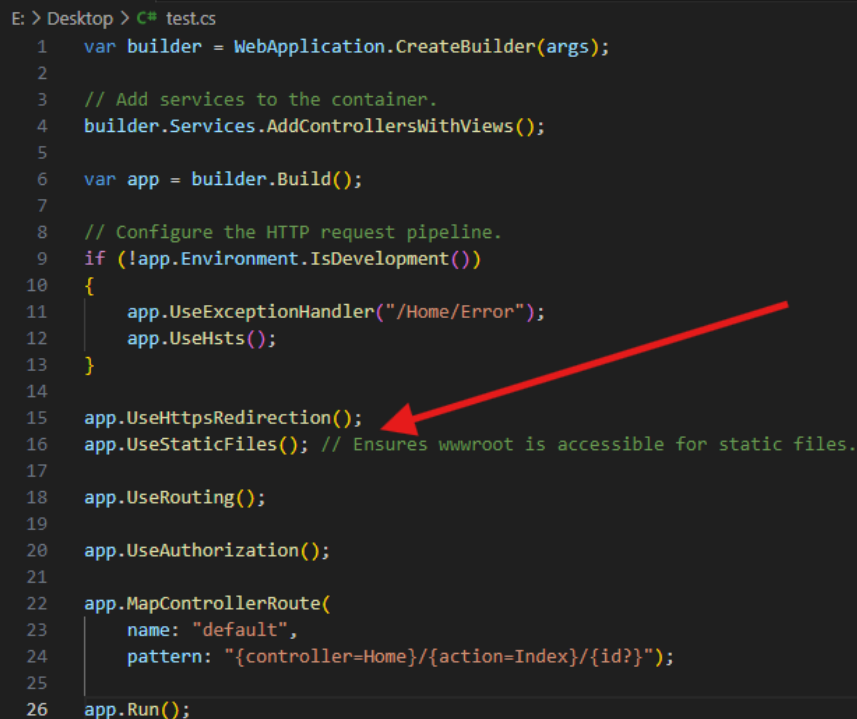
1. Decide on a directory where you'll store the images for the carousel. For example, let's use a folder named `carousel_images` inside the `wwwroot` folder of your project (e.g., `wwwroot/carousel_images`).
2. Place your image files in this directory.

** You can set anywhere in the server if you want to store the image.*

Step 2: Configure Static Files in Startup or Program.cs

In **ASP.NET Core 8.0**, static files are served by default from the `wwwroot` folder. You just need to make sure the `UseStaticFiles` middleware is added to your pipeline, which is typically already included.

In your `Program.cs`, you should have:

A screenshot of a code editor showing the `Program.cs` file. The code is for a .NET Core 8.0 application. A red arrow points from the right side of the image to the `app.UseStaticFiles();` line on line 16. The code is as follows:

```
E: > Desktop > C# test.cs
1  var builder = WebApplication.CreateBuilder(args);
2
3  // Add services to the container.
4  builder.Services.AddControllersWithViews();
5
6  var app = builder.Build();
7
8  // Configure the HTTP request pipeline.
9  if (!app.Environment.IsDevelopment())
10 {
11     app.UseExceptionHandler("/Home/Error");
12     app.UseHsts();
13 }
14
15 app.UseHttpsRedirection();
16 app.UseStaticFiles(); // Ensures wwwroot is accessible for static files.
17
18 app.UseRouting();
19
20 app.UseAuthorization();
21
22 app.MapControllerRoute(
23     name: "default",
24     pattern: "{controller=Home}/{action=Index}/{id?}");
25
26 app.Run();
```

Step 3: Use Relative Paths in the Model

Store relative paths like `/carousel_images/banner1.jpg` in the `ImageUrl` property. This keeps URLs portable and easy to construct on the client side.

Step 4: Create API Endpoints

Build Web API endpoints to support operations such as:

- GET /api/carousel – Fetch all (or only active) carousel items.
- GET /api/carousel/{id}- Get carousel item by Id
- POST /api/carousel – Add a new carousel item.
- PUT /api/carousel/{id} – Update an existing item.
- DELETE /api/carousel/{id} – Remove an item. (Deleted both of database and file in server)

These APIs allow client applications (web/mobile) to retrieve and manage carousel content dynamically.

Step 5: Image Access via URL

When you run your application, ASP.NET Core will serve images directly from the wwwroot directory. So if your ImageUrl is /carousel_images/image1.jpg, it will be resolved to https://yourdomain.com/carousel_images/image1.jpg.

Make sure the **ImageUrl** field correctly maps to this path.

Exercise 3: Implement Full CRUD for Books

Objective: Create complete CRUD API endpoints for the Books table.

Requirements:

- GET: Retrieve a list of books with pagination, search by title, and filtering by CategoryId or AuthorId.
- GET /api/books/{id} – Get book details by ID.
- POST: Add a new book (including uploading a cover image and PDF file).
- PUT: Update existing book information.
- DELETE: Soft delete (by updating **IsActive** or **IsDeleted** instead of removing from database – add this **fields** if dont available in the model and database).

Exercise 4: Author Management

Objective: Manage Authors data.

Requirements:

- GET /api/authors: List all authors with optional search by name.
- GET /api/authors/{id}: Get author details.
- POST, PUT, DELETE: Create, update, and soft delete authors.
- Allow uploading of avatar images and store relative path.

Exercise 5: Loan History Management

Objective: Display and manage book loan records.

Requirements:

- GET /api/loans: Filter by UserId, Status, or loan date range.
- GET /api/loans/{id}: View loan details.
- POST: Create new loan record (only if AvailableCopies > 0).
- PUT: Update return date and automatically set Status to Returned.

Exercise 6: User Registration & Login

Objective: Implement user APIs.

Requirements:

- POST /api/users/register: Register new users (hash password, generate ActiveCode, and simulate sending activation email).
- GET /api/users/activate?code=xxx: Activate account via activation code.(
Optional – add a new method using POST and body json to Activate account via activation code)

Exercise 7: Category Management with Activation

Objective: Manage book categories and enable/disable them.

Requirements:

- GET /api/categories: Filter by IsActive and search by name.
- POST / PUT: Add or update category.

- PATCH /api/categories/{id}/toggle: Toggle the IsActive status of a category.

Exercise 8: Top Borrowed Books Report

Objective: Generate a report of the most borrowed books.

Requirements:

- GET /api/reports/top-borrowed?fromDate=...&toDate=...&top=10

Return list of BookId, Title, and number of times borrowed within the given time range.

Exercise 9: PDF Online Reader

Objective: Allow users to read book PDFs online.

Requirements:

- Create endpoint GET /api/books/{id}/read

Check if the book has a PDF file → return a relative URL. If not return a message is not found.

- Allow the client to embed the PDF using <iframe> or a PDF reader component.

Exercise 10: Soft Delete & Recycle Bin for Books

Objective: Manage soft-deleted books and support recovery.

Requirements:

- GET /api/books/deleted: List books where IsDeleted = true.
- PATCH /api/books/{id}/restore: Restore a previously deleted book.
- DELETE /api/books/{id}/hard: Permanently delete a book and its associated files (PDF and image) from the server.