Library Management System

Software Engineering Internship – Assignment

Report

Ishini Madakaladeniya.

Department of Computer Engineering,

Faculty of Engineering, University of Ruhuna.

Table of Contents

1 Backend (C# .NET)

1.1 Setting Up the Backend (C# .NET)

1. **Installation and Tools Used**

   Code Editor – Visual Studio

   .NET SDK – version 8.0

2. **Creating a new project**
   Terminal Commands:
   - dotnet new webapi -n LibraryManagementBackend – created a WEBAPI project named LibraryManagementBackend

3. **Installing Packages to set SQLite database**
   Entity Framework was used as the ORM (Object Relational Mapper) and file database SQLite was used for data storage.
   Packages added from NuGet Package manager:
   - Microsoft.EntityFrameworkCore
   - Microsoft.EntityFrameworkCore.Sqlite
   - Microsoft.EntityFrameworkCore.Design

4. **Creating Models, Controllers and Database related folders.**
   I followed the MVC (Model, View, Controller) architecture and since the application is based on API requests there is no view section.

- Model: Book model which includes, Book Id, Book Title, Book Author and Book Description which is used to create the database .
- Data: LibraryContext.cs consist of the configuration for the application to interact with the SQLite database. And registered the Database context in Program.cs using the following,

```
//comfigure the Sqlite file database for data transaction
builder.Services.AddDbContext<LibraryContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));
```

Added the connection String to appsettings.json,

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=library.db"
  },
  "AllowedHosts": "*"
}
```

- Controller: BooksController.cs consists of the logic related to the CRUD operations.

To get a List of all the existing book records – GET request is created.

```
// Get list of all books
[HttpGet]
public async Task<ActionResult<IEnumerable<Book>>> GetBooks()
{
    try
    {
        return await _context.Books.ToListAsync();
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { message = "An error occurred while retrieving books.", error = ex.Message });
    }
}
```

To get a book record by its Id – GET request is created.

```
// Get a book by ID
[HttpGet("{id}")]
public async Task<ActionResult<Book>> GetBook(int id)
{
    try
    {
        var book = await _context.Books.FindAsync(id);
        if (book == null) return NotFound();
        return book;
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { message = "An error occurred while retrieving the book.", error = ex.Message });
    }
```

To add a new book record to the database – POST request is created. The availability of the book record Id is checked before adding.

```
// Add a new book record
[HttpPost]
public async Task<ActionResult<Book>> CreateBook(Book book)
{
    try
    {
        var existingBook = await _context.Books.FindAsync(book.Id);
        if (existingBook != null)
        {
            return Conflict(new { message = "A book with this ID already exists. Please use a different ID." });
        }

        _context.Books.Add(book);
        await _context.SaveChangesAsync();
        return CreatedAtAction(nameof(GetBook), new { id = book.Id }, book);
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { message = "An error occurred while adding the book.", error = ex.Message });
    }
}
```

To Update a book by its Id – POST request is created.

```
// Update a book record by ID
[HttpPut("{id}")]
public async Task<IActionResult> UpdateBook(int id, Book book)
{
    if (id != book.Id) return BadRequest();

    try
    {
        _context.Entry(book).State = EntityState.Modified;
        await _context.SaveChangesAsync();
        return NoContent();
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { message = "An error occurred while updating the book.", error = ex.Message });
    }
}
```

To Delete a book by its Id – DELETE request is created.

```
// Delete a book by ID
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteBook(int id)
{
    try
    {
        var book = await _context.Books.FindAsync(id);
        if (book == null) return NotFound();

        _context.Books.Remove(book);
        await _context.SaveChangesAsync();
        return NoContent();
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { message = "An error occurred while deleting the book.", error = ex.Message });
    }
}
```

Knowledge Gained: FindAsync, SaveChangesAsync, ToListAsync are used in web/API based applications instead of Find, SaveChanges and ToList because to run the application without interrupting the other process, using Async make sure that when one database transaction is made the other processes are executing simultaneously without working in Sync.

5. **Run Migrations**
   Ran the following commands to create and apply migrations:
   - dotnet ef migrations add InitialCreate – This command creates a migration file inside the Migrations Folder that contains the SQL commands needed to generate the database schema based on the C# model class Book.
   - dotnet ef database update – Reads the commands in the migration file and applies the commands to the database library.db and if database doesn't exist EF creates a new file database with name library.db in the mentioned path in the Data Source in appsettings.json file. In this project it is created at the same location folder where the backend application is saved.
6. **Run the application and Test API endpoints in Swagger**
   - dotnet run
   - Tested all the API endpoints in http://localhost:5103/swagger
7. **Created a new Repository in Git Hub and published the master branch**

1.1.1 Problems Faced

1. Version Incompatibility occurred when trying to clone the application and run in another computer. I initially used the .NET version 9.0 then found out that 9.0 is a preview version and a stable version is 8.0.

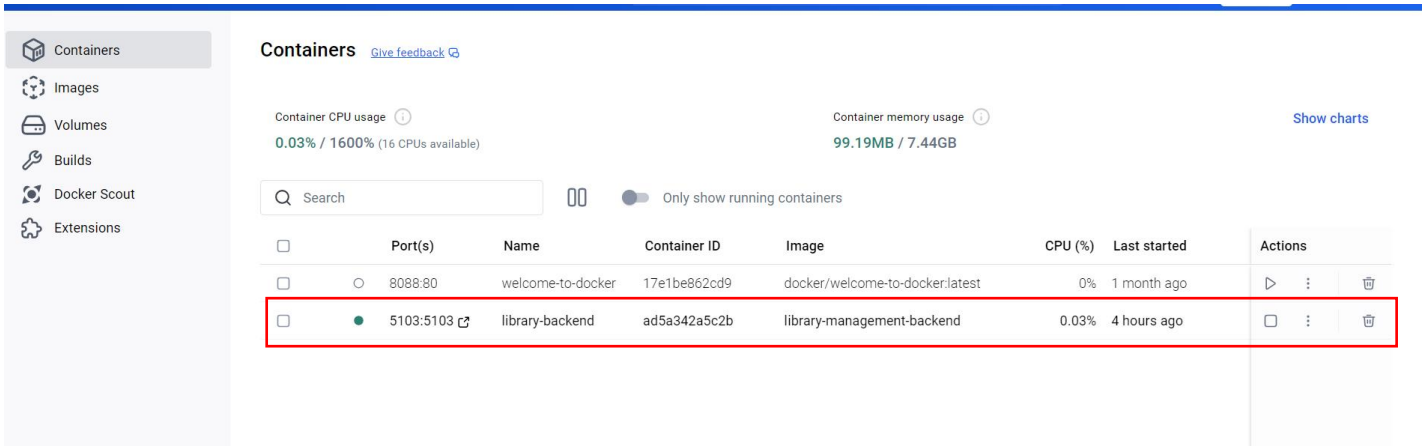
   Solution

Deprecated the version from 9.0 to 8.0 for better stability and to overcome version incompatibility overall decided to dockerize the backend application.

Created a dockerFile and .dockerignore files and wrote the code required to build and run the application with .NET version 8.0.

And created a dockerImage named library-management-backend and started the image by using following commands.

- docker build -t library-management-backend . – created a docker image named library-management-backend
- docker run -d -p 5103:5103 --name library-backend library-management-backend – run the image.



2. CORS errors occurred when receiving
   Solution: configured CORS policy in backend

```
//configuring cors policy
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowFrontend", policy =>
    {
        policy.WithOrigins("http://localhost:5173")
              .AllowAnyHeader()
              .AllowAnyMethod();
    });
});
```

```
app.UseCors("AllowFrontend");
```

1.2 Setting Up the Frontend (React Typescript)

1. **Creating the frontend**
   Created a Vite project with TypeScript and configured Shadcn UI for CSS styling. Followed the Shadcn UI Vite Documentation and configured the project.
   Documentation Link: https://ui.shadcn.com/docs/installation/vite

   Commands:
   - npm create vite@latest library-management-frontend --template react-ts – create vite react project
   - npm install – install node modules
   - npm install -D tailwindcss@3.4.17 postcss autoprefixer– install tailwind
   - npx tailwindcss init -p – initiate ailwing
   - npx shadcn-ui@latest init – install shadcn ui

2. **Creating Components and Services**
   - BookList Component: to show the list of books available with options to edit, delete and add a new book
   - BookForm Component: Create and Update new/existing Book records.
   - Book Model: Defines the structure of a database schema of a book.
   - BookService: Defines all the API endpoints called from the backend to all the CRUD operations. Used **Try Catch** blocks for error handling.
   - AppRoutes – define all the routes in the system.

1.2.1 Problems faced

1. TailwindCSS Installation Issue in Frontend - When trying to install Tailwind for ShadCN UI, I got errors because Tailwind recently **removed tailwind.config.js** in new versions.
   Solution: I installed an **older version of Tailwind** where tailwind.config.js was still supported.
2. Handling Duplicate Book IDs in Backend - Adding a book with an existing ID caused failures.
   Solution: Implemented a 409 Conflict error in createBook() to notify the frontend.

```
export const createBook = async (book: Omit<Book, "id">): Promise<Book> => {
  try {
    const response = await axios.post(API_URL, book);
    toast.success("Book record created successfully");
    return response.data;
  } catch (error: unknown) {
    if (error instanceof AxiosError && error.response) {
      if (error.response.status === 409) {
        toast.error(error.response.data.message);
      } else {
        toast.error("Failed to create book: " + error.response.data.message);
      }
    } else {
      toast.error("An unexpected error occurred.");
    }
    throw error;
  }
};
```

3. Form Not Updating Fields When Updating an existing Book record - When editing a book, the input fields did not update correctly.

   Solution: Used form.reset(book) inside useEffect() to update values dynamically.

1.3 Lessons Learned

- How to set up .NET Web API with Entity Framework Core.
- How to connect React & .NET using REST APIs.
- How to handle database migrations and work with SQLite.
- The importance of error handling in APIs.

1.4 Instructions for running the frontend and backend

1.4.1 Running the Backend

Go to GitHub repository: https://github.com/ishii1218/LibraryManagementBackend

Step 01

Clone the repository: git clone https://github.com/ishii1218/LibraryManagementBackend.git

Step 02

Option 01: Run Docker Image (version compatible)

Commands:

- docker build -t library-management-backend .
- docker run -d -p 5103:5103 --name library-backend library-management-backend

Option 02: Run the backend (Recommended version - .NET 8.0 or higher)

Command:

- dotnet run

1.4.2 Running the Frontend

Go to GitHub repository: https://github.com/ishii1218/LibraryManagementCRUD-app

Step 01

Clone the repository: git clone https://github.com/ishii1218/LibraryManagementCRUD-app.git

Step 02

Run commands:

- cd library-management-frontend
- npm install
- npm run dev
  and follow the link given.