**НП ”Обучение за ИТ умения и кариера”**

**Модул 7: Разработка на софтуер**

**КУРСОВ ПРОЕКТ**

на тема:

**Библиотека**

Изготвили:

Михаил Тенев

Група 08

гр. Хасково

2025 г.

1. **Цели**

Разработване на мулти-интерфейсно приложение, което управлява база данни, съдържаща информацията за книги, техните автори, членовете на библиотеката и историята на заемани книги.

1. **Разпределение на ролите**

Михаил Тенев - работя сам затова всичкото планиране и програмен код са изработени от мен. Отговарям за: структура на проекта, планиране на таблици и връзки в базата данни, разработка на Business логика, разработка на Data структура (Models и DbContext), разработка на конзолен интерфейс и неговия Display слой, както и допълнителния помощен клас UIHelper, ASP.NET контролери и views razor станици и тестове за бизнес логиката.

1. **Основни етапи в реализирането на проекта**
2. Планиране на структура

Планиране на структурата на базата данни и разпределението на структурата на проекта. Избор на помощни библиотеки за разработка и избиране на GitHub за контролер на версиите.

1. Разработка на Data слоя  
     
   Добавих Entity Framework Core и допълнителни библиотеки нужни за разработката към проекта Data. Създадох DbContext в класът LibraryDbContext и модели на таблиците в Data/Models. Създадох Enums папка и добавих клас Genre, описващ enum genre - съдържащ информация за жанровете на книгите. Описах connection string и добавих опция за инжектиране на connection string от ASP.NET. Описах връзките на моделите за по-точно създаване на базата данни. Добавяме миграции и обновяваме базата данни чрез PM конзолата.
2. Разработка на Business слоя  
     
   Добавих нужните библиотеки и референция към проекта Data. За всеки модел създадох business клас, в който се описват CRUD операциите към моделите. И добавих Dispose метод от IDisposable интерфейса за премахване на контекста.
3. Разработка на конзолно приложение

Добавих референции към Business и Data слоя, както и нужните библиотеки за всички приложения. Разработих Presentation слой, който ще управлява интерфейса на конзолното приложение. Той се състои от един главен клас Display, класове Sub-displays и един помощен клас UIHelper. При разработката на слоя установих, че ще е много обемен ако цялата логика е само в Display класа, затова го разделихме на Sub-displays, като всяка таблица има свой sub-display. При разработката също създадох помощния клас UIHelper, който има за цел да помага с входа на данни и да намали обема на кода, като в него се съдържат валидациите за входните данни при въвеждане в интерфейса на конзолата. Display класът събира логиката на sub-display класовете, в тях се съдържат менюта и методите които извършват, те използват методите от Business слоя.

1. Разработка на Тестове за Business логиката

Създадох тестове за да съм сигурен дали валидно се изпълняват методите в Business логиката на всяка таблица. При разработката пробвах да използвам Moq библиотеката, но тя не подържа async методи затова смених на In Memory Database библиотеката към EF, която не само направи процеса по-лесен, а и намали обема код и ускори времето за изпълнение на тестовете.

1. Разработка на ASP.NET (MVC) уеб приложение

При създаването на ASP.NET уеб приложението реших сам да напиша controller и view логиката на всеки клас вместо да използвам scaffolding, за да може да имам повече контрол над методите на контролерите и дизайна на страниците. В дизайна също използвам JavaScript за някои елементи, като търсачки и филтри. Също и в ASP.NET приложението ми се съдържа и Seeding слоя, който има за цел, в тестова среда, когато базата данни е празна, да се попълни с тестови данни за по-лесно презентиране и тества на проекта.

1. **Реализация**

**Използвани технологии:**

Език и платформи:

1. C#
2. JavaScript
3. ASP.NET Core MVC

ORM и база данни:

1. Microsoft.EntityFrameworkCore – ORM за работа с бази данни
2. Microsoft.EntityFrameworkCore.SqlServer – SQL Server провайдър
3. Microsoft.EntityFrameworkCore.Tools – инструменти за миграции и работа с базата
4. Microsoft.EntityFrameworkCore.InMemory – In-Memory база за тестване
5. Microsoft.VisualStudio.Web.CodeGeneration.EntityFrameworkCore – за scaffold и code generation

Тестване и покритие

1. Microsoft.NET.Test.Sdk – инфраструктура за тестове
2. NUnit – framework за юнит тестове
3. NUnit.Analyzers – анализи и правила за NUnit
4. NUnit3TestAdapter – интеграция на NUnit с Visual Studio
5. coverlet.collector – за измерване на покритие на тестовете (code coverage)

Инструменти и други

1. GitHub – контрол на версиите
2. Visual Studio – среда за разработка
3. Package Manager Console – използвана за миграции и обновяване
4. Fine Code Coverage - Отбелязва покритието на unit тестове

**Описание на приложението:**  
  
Разработеното приложение представлява библиотечна система за управление на книги, автори, читатели и история на заеманите книги. Системата използва многослойна архитектура, състояща се от:

* Data слой – съдържа модели на таблиците и контекста на базата данни чрез Entity Framework Core;
* Business слой – реализира бизнес логиката и CRUD операциите за всеки модел;
* Конзолен интерфейс – позволява базово управление чрез конзолно меню, разделено по функционалност за всяка таблица;
* ASP.NET Core MVC интерфейс – уеб базиран потребителски интерфейс, позволяващ визуално управление и търсене на записи;
* Тестов слой – включва автоматизирани тестове за валидация на бизнес логиката с помощта на In-Memory база и NUnit.

Приложението е подходящо за малки библиотеки или учебни цели. То позволява:

* Добавяне, редакция и изтриване на книги, автори и читатели;
* Проследяване на история на заетите книги;
* Филтриране и търсене по различни критерии;
* Автоматично зареждане на тестови данни в празна база (Seeding).

**Програмен код на по-важните методи:**

Model пример (Book.cs):

|  |
| --- |
| using Data.Enums;  using System.ComponentModel.DataAnnotations;  using System.ComponentModel.DataAnnotations.Schema;    namespace Data.Models  {  public class Book  {  [Key]  public int Id { get; set; }  [Required]  [MaxLength(55)]  public string Title { get; set; }  [Required]  public Genre Genre { get; set; }  [StringLength(13, MinimumLength = 10, ErrorMessage = "ISBN must be between 10 and 13 characters.")]  public string ISBN { get; set; }  [Required]  [DisplayFormat(DataFormatString = "{0:dd-MM-yyyy}", ApplyFormatInEditMode = true)]  public DateTime PublicationDate { get; set; }  [Required]  public int AuthorID { get; set; }    [ForeignKey("AuthorID")]  public Author? Author { get; set; }    public ICollection<BorrowedBook>? BorrowedBooks { get; set; }  }  } |

LibraryDbContext.cs:

|  |
| --- |
| using Data.Models;  using Microsoft.EntityFrameworkCore;    namespace Data  {  public class LibraryDbContext : DbContext  {  public virtual DbSet<Author> Authors { get; set; }  public DbSet<Book> Books { get; set; }  public DbSet<Member> Members { get; set; }  public DbSet<BorrowedBook> BorrowedBooks { get; set; }    // Constructor that accepts DbContextOptions  public LibraryDbContext(DbContextOptions<LibraryDbContext> options) : base(options)  {  }    public LibraryDbContext()  {  }    // Default constructor for configuring directly (in case not using DI)  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  {  // Check if options were already configured  if (!optionsBuilder.IsConfigured)  {  optionsBuilder.UseSqlServer(@"Server=.\SQLEXPRESS;Database=LibraryDb;Integrated Security=True;TrustServerCertificate=True;");  }  }    protected override void OnModelCreating(ModelBuilder modelBuilder)  {  base.OnModelCreating(modelBuilder);    // Composite primary key for BorrowedBook  modelBuilder.Entity<BorrowedBook>()  .HasKey(bb => new { bb.BookID, bb.MemberID });    // Unique constraint on ISBN  modelBuilder.Entity<Book>()  .HasIndex(b => b.ISBN)  .IsUnique();    // Explicit relationship mapping    /// Book - Author  modelBuilder.Entity<Book>()  .HasOne(b => b.Author)  .WithMany(a => a.Books)  .HasForeignKey(b => b.AuthorID)  .OnDelete(DeleteBehavior.Cascade);    /// BorrowedBook - Book  modelBuilder.Entity<BorrowedBook>()  .HasOne(bb => bb.Book)  .WithMany(b => b.BorrowedBooks)  .HasForeignKey(bb => bb.BookID)  .OnDelete(DeleteBehavior.Cascade);    /// BorrowedBook - Member  modelBuilder.Entity<BorrowedBook>()  .HasOne(bb => bb.Member)  .WithMany(m => m.BorrowedBooks)  .HasForeignKey(bb => bb.MemberID)  .OnDelete(DeleteBehavior.Cascade);    // ENUMS stored as strings  modelBuilder.Entity<Book>()  .Property(b => b.Genre)  .HasConversion<string>();  } |

Business пример (BookBusiness):

|  |
| --- |
| using Data;  using Data.Models;  using Microsoft.EntityFrameworkCore;    namespace Business  {  public class BookBusiness : IDisposable  {  private readonly LibraryDbContext \_context;  private readonly bool \_contextOwned;    // Constructor for ASP.NET Core (DI provides the context)  public BookBusiness(LibraryDbContext context)  {  \_context = context;  \_contextOwned = false;  }    // Constructor for Console App (creates its own context)  public BookBusiness()  {  \_context = new LibraryDbContext();  \_contextOwned = true;  }    public async Task<List<Book>> GetAllAsync()  {  return await \_context.Books.ToListAsync();  }    public async Task<List<Book>> GetAllWithIncludesAsync()  {  return await \_context.Books  .Include(b => b.Author)  .Include(b => b.BorrowedBooks)  .ToListAsync();  }    public async Task<List<Book>> GetAllByGenreAsync(string genre)  {  return await \_context.Books  .Where(b => b.Genre.ToString().ToLower() == genre.ToLower())  .ToListAsync();  }    public async Task<List<Book>> GetAllByGenreWithIncludesAsync(string genre)  {  return await \_context.Books  .Include(b => b.Author)  .Include(b => b.BorrowedBooks)  .Where(b => b.Genre.ToString().ToLower() == genre.ToLower())  .ToListAsync();  }    public async Task<Book> GetAsync(int id)  {  return await \_context.Books.FindAsync(id);  }  public async Task<Book> GetWithIncludesAsync(int id)  {  return await \_context.Books  .Include(b => b.Author)  .Include(b => b.BorrowedBooks)  .FirstOrDefaultAsync(b => b.Id == id);  }    public async Task<Book> GetByISBNAsync(string ISBN)  {  return await \_context.Books.FirstAsync(b => b.ISBN == ISBN);  }    public async Task<Book> GetByISBNWithIncludesAsync(string ISBN)  {  return await \_context.Books  .Include(b => b.Author)  .Include(b => b.BorrowedBooks)  .FirstOrDefaultAsync(b => b.ISBN == ISBN);  }    public async Task AddAsync(Book book)  {  await \_context.Books.AddAsync(book);  await \_context.SaveChangesAsync();  }    public async Task UpdateAsync(Book book)  {  var item = await \_context.Books.FindAsync(book.Id);  if (item != null)  {  \_context.Entry(item).CurrentValues.SetValues(book);  await \_context.SaveChangesAsync();  }  }    public async Task DeleteAsync(int id)  {  var book = await \_context.Books.FindAsync(id);  if (book != null)  {  \_context.Books.Remove(book);  await \_context.SaveChangesAsync();  }  }    // Make sure you clean up if we created the context ourselves  public void Dispose()  {  if (\_contextOwned)  {  \_context.Dispose();  }  }  }  } |

Sub-Display пример (BookDisplay.cs):

|  |
| --- |
| using Business;  using Data.Enums;  using Data.Models;    namespace ConsoleApp.Presentation.SubDisplays  {  internal class BookDisplay  {  // Business layer objects to interact with data models  private readonly AuthorBusiness authorBusiness = new AuthorBusiness();  private readonly BookBusiness bookBusiness = new BookBusiness();  private readonly MemberBusiness memberBusiness = new MemberBusiness();  private readonly BorrowedBookBusiness borrowedBookBusiness = new BorrowedBookBusiness();    // UI helper to assist with common input/output operations  private readonly UIHelper uiHelper = new UIHelper();    /// <summary>  /// Main function to manage book-related operations.  /// Provides a menu for the user to choose actions like borrowing, browsing, adding, updating, or deleting books.  /// </summary>  public async Task BookManager()  {  ShowBookMenu(); // Display the book management menu  var operation = uiHelper.ReadIntInput("Please select an option:");    // Execute the selected operation based on user's input  switch (operation)  {  case 1:  await Borrow(); // Borrow a book  break;  case 2:  await Browse(); // Browse available books  break;  case 3:  await AddBook(); // Add a new book to the system  break;  case 4:  await UpdateBook(); // Update book details  break;  case 5:  await DeleteBook(); // Delete a book from the system  break;  default:  Console.WriteLine("Invalid option."); // Invalid choice handling  break;  }  }    // Menus for book management and browsing  /// <summary>  /// Displays the main menu for book management.  /// </summary>  private void ShowBookMenu()  {  uiHelper.ShowHeader("Book Management");  Console.WriteLine("1. Borrow a Book");  Console.WriteLine("2. Browse Books");  Console.WriteLine("3. Add books");  Console.WriteLine("4. Update books");  Console.WriteLine("5. Delete books");  }    /// <summary>  /// Displays the menu for browsing books by different criteria.  /// </summary>  private void ShowBrowseBooks()  {  uiHelper.ShowHeader("Browse Books");  Console.WriteLine("1. All Books");  Console.WriteLine("2. By Genre");  Console.WriteLine("3. Fetch by ID");  Console.WriteLine("4. Fetch by ISBN");  }    // Main functions to perform actions on books  /// <summary>  /// Allows the user to borrow a book. First prompts the user to browse books.  /// </summary>  private async Task Borrow()  {  ShowBrowseBooks(); // Show browsing options  var operation = uiHelper.ReadIntInput("Please select an option:");    // Handle different borrow scenarios based on user's choice  switch (operation)  {  case 1:  await PrintAllBooks(); // Display all books  await BorrowById(); // Borrow by ID  break;  case 2:  await PrintByGenre(); // Display books by genre  await BorrowById(); // Borrow by ID  break;  case 3:  await BorrowById(); // Directly borrow by ID  break;  case 4:  await BorrowByISBN(); // Borrow by ISBN  break;  default:  Console.WriteLine("Invalid option."); // Invalid choice handling  break;  }  }    /// <summary>  /// Allows the user to browse books. Provides different browsing criteria.  /// </summary>  private async Task Browse()  {  ShowBrowseBooks(); // Show browsing options  var operation = uiHelper.ReadIntInput("Please select an option:");    // Handle different browsing scenarios based on user's choice  switch (operation)  {  case 1:  await PrintAllBooks(); // Display all books  break;  case 2:  await PrintByGenre(); // Display books by genre  break;  case 3:  await PrintById(); // Display books by ID  break;  case 4:  await PrintByISBN(); // Display books by ISBN  break;  default:  Console.WriteLine("Invalid option."); // Invalid choice handling  break;  }  }    /// <summary>  /// Allows the user to add a new book to the system.  /// Prompts the user for book details and validates the genre.  /// </summary>  private async Task AddBook()  {  var book = new Book();  book.Title = uiHelper.ReadStringInput("Please enter the book title:");  string genre = uiHelper.ReadStringInput("Please enter the book genre:").ToLower();    // Validate and assign the genre based on user's input  switch (genre)  {  case "fiction":  book.Genre = Genre.Fiction;  break;  case "non - fiction":  book.Genre = Genre.NonFiction;  break;  case "fantasy":  book.Genre = Genre.Fantasy;  break;  case "mystery":  book.Genre = Genre.Mystery;  break;  case "romance":  book.Genre = Genre.Romance;  break;  case "science fiction":  book.Genre = Genre.ScienceFiction;  break;  case "biography":  book.Genre = Genre.Biography;  break;  case "thriller":  book.Genre = Genre.Thriller;  break;  case "horror":  book.Genre = Genre.Horror;  break;  case "historical fiction":  book.Genre = Genre.HistoricalFiction;  break;  case "health and wellness":  book.Genre = Genre.HealthAndWellness;  break;  case "travel":  book.Genre = Genre.Travel;  break;  case "children literature":  book.Genre = Genre.ChildrenLiterature;  break;  default:  Console.WriteLine($"Genre - {genre} doesn't exist");  return; // Return if the genre is invalid  }    // Input and assign other book details  book.PublicationDate = uiHelper.ReadDateInput("Please enter the book published year:");  book.ISBN = uiHelper.ReadStringInput("Please enter the book ISBN:");  var authorId = uiHelper.ReadIntInput("Please enter the author ID:");  var author = await authorBusiness.GetAsync(authorId);    if (author != null)  {  book.AuthorID = author.Id; // Assign the author to the book  await bookBusiness.AddAsync(book); // Add the book to the system  Console.WriteLine($"Book {book.Title} added successfully.");  }  else  {  Console.WriteLine("Author not found.");  }  }    /// <summary>  /// Allows the user to update an existing book's details.  /// Prompts the user for new values and validates them.  /// </summary>  private async Task UpdateBook()  {  var bookId = uiHelper.ReadIntInput("Please enter the book ID:");  var book = await bookBusiness.GetAsync(bookId);    if (book != null)  {  book.Title = uiHelper.ReadStringInput("Please enter the new book title:");  string genre = uiHelper.ReadStringInput("Please enter the new book genre:").ToLower();    // Validate and assign the new genre  switch (genre)  {  case "fiction":  book.Genre = Genre.Fiction;  break;  case "non - fiction":  book.Genre = Genre.NonFiction;  break;  // Other genre cases...  default:  Console.WriteLine($"Genre - {genre} doesn't exist");  return;  }    // Input new ISBN and author details  book.PublicationDate = uiHelper.ReadDateInput("Please enter the book published year:");  book.ISBN = uiHelper.ReadStringInput("Please enter the new book ISBN:");  var authorId = uiHelper.ReadIntInput("Please enter the new author ID:");  var author = await authorBusiness.GetAsync(authorId);    if (author != null)  {  book.AuthorID = author.Id; // Assign new author to book  await bookBusiness.AddAsync(book); // Update book details in the system  Console.WriteLine($"Book {book.Title} updated successfully.");  }  else  {  Console.WriteLine("Author not found.");  }  }  else  {  Console.WriteLine($"Book with ID - {bookId} doesn't exist");  }  }    /// <summary>  /// Allows the user to delete a book from the system.  /// Prompts the user for the book ID and removes the book if found.  /// </summary>  private async Task DeleteBook()  {  var bookId = uiHelper.ReadIntInput("Please enter the book ID:");  var book = await bookBusiness.GetAsync(bookId);    if (book != null)  {  await bookBusiness.DeleteAsync(book.Id); // Delete the book from the system  Console.WriteLine($"Book {book.Title} deleted successfully.");  }  else  {  Console.WriteLine($"Book with ID - {bookId} doesn't exist");  }  }    // Print functions to display books in various formats  /// <summary>  /// Prints all books in the system.  /// </summary>  private async Task PrintAllBooks()  {  var books = await bookBusiness.GetAllWithIncludesAsync();  foreach (var book in books)  {  Console.WriteLine($"ID: {book.Id}, Title: {book.Title}, Genre: {book.Genre}, Author: {book.Author.FirstName} {book.Author.LastName}, Release Date: {book.PublicationDate:yyyy-MM-dd}, ISBN: {book.ISBN}");  }  }    /// <summary>  /// Prints books filtered by genre.  /// </summary>  private async Task PrintByGenre()  {  var genre = uiHelper.ReadStringInput("Please enter the genre:");  var filteredBooks = await bookBusiness.GetAllByGenreWithIncludesAsync(genre);    if (filteredBooks != null && filteredBooks.Count > 0)  {  foreach (var book in filteredBooks)  {  Console.WriteLine($"ID: {book.Id}, Title: {book.Title}, Author: {book.Author.FirstName} {book.Author.LastName}, Release Date: {book.PublicationDate:yyyy-MM-dd}, ISBN: {book.ISBN}");  }  }  else  {  Console.WriteLine($"Genre - {genre} doesn't exist or no books found.");  }  }    /// <summary>  /// Prints a book found by its ID.  /// </summary>  private async Task PrintById()  {  var bookId = uiHelper.ReadIntInput("Please enter the book ID:");  var book = await bookBusiness.GetWithIncludesAsync(bookId);    if (book != null)  {  Console.WriteLine($"ID: {book.Id}, Title: {book.Title}, Genre: {book.Genre}, Author: {book.Author.FirstName} {book.Author.LastName}, Release Date: {book.PublicationDate:yyyy-MM-dd}, ISBN: {book.ISBN}");  }  else  {  Console.WriteLine($"Book with ID - {bookId} doesn't exist");  }  }    /// <summary>  /// Prints a book found by its ISBN.  /// </summary>  private async Task PrintByISBN()  {  var bookISBN = uiHelper.ReadStringInput("Please enter the book ISBN:");  var book = await bookBusiness.GetByISBNWithIncludesAsync(bookISBN);    if (book != null)  {  Console.WriteLine($"ID: {book.Id}, Title: {book.Title}, Genre: {book.Genre}, Author: {book.Author.FirstName} {book.Author.LastName}, Release Date: {book.PublicationDate:yyyy-MM-dd}, ISBN: {book.ISBN}");  }  else  {  Console.WriteLine($"Book with ISBN - {bookISBN} doesn't exist");  }  }    // Borrow functions for borrowing books by different criteria  /// <summary>  /// Allows the user to borrow a book by its ID.  /// </summary>  private async Task BorrowById()  {  var bookId = uiHelper.ReadIntInput("Please select a book by ID:");  var book = await bookBusiness.GetWithIncludesAsync(bookId);    if (book != null)  {  Console.WriteLine($"You selected: {book.Title}");  var memberId = uiHelper.ReadIntInput("Please enter your member ID:");  var member = await memberBusiness.GetAsync(memberId);    if (member != null)  {  if (book.BorrowedBooks.Any(bb => bb.ReturnDate == null))  {  Console.WriteLine($"Book {book.Title} with ID: {book.Id} is already borrowed.");  return; // Book is already borrowed, exit the method  }  // Create a borrowed book record  var borrowedBook = new BorrowedBook  {  BookID = book.Id,  MemberID = member.Id,  BorrowDate = DateTime.Now.Date, // Current date  DueDate = DateTime.Now.AddMonths(2).Date // Assuming a 2-month borrowing period  };  await borrowedBookBusiness.AddAsync(borrowedBook); // Save the borrowed book  Console.WriteLine($"You have successfully borrowed {book.Title}.");  }  else  {  Console.WriteLine("Member not found.");  }  }  else  {  Console.WriteLine("Book not found.");  }  }    /// <summary>  /// Allows the user to borrow a book by its ISBN.  /// </summary>  private async Task BorrowByISBN()  {  string bookISBN = uiHelper.ReadStringInput("Please enter the book ISBN:");  var book = await bookBusiness.GetByISBNWithIncludesAsync(bookISBN);    if (book != null)  {  Console.WriteLine($"You selected: {book.Title}");  var memberId = uiHelper.ReadIntInput("Please enter your member ID:");  var member = await memberBusiness.GetAsync(memberId);    if (member != null)  {  if (book.BorrowedBooks.Any(bb => bb.ReturnDate == null))  {  Console.WriteLine($"Book {book.Title} with ID: {book.Id} is already borrowed.");  return; // Book is already borrowed, exit the method  }  var borrowedBook = new BorrowedBook  {  BookID = book.Id,  MemberID = member.Id,  BorrowDate = DateTime.Now.Date  };  await borrowedBookBusiness.AddAsync(borrowedBook); // Add the borrowed book record  Console.WriteLine($"You have successfully borrowed {book.Title}.");  }  else  {  Console.WriteLine("Member not found.");  }  }  else  {  Console.WriteLine("Book not found.");  }  }  }  } |

Display.cs:

|  |
| --- |
| using ConsoleApp.Presentation.SubDisplays;      namespace ConsoleApp.Presentation  {  internal class Display  {  // Creating instances of sub-displays for each domain (Book, Member, Author, Borrowed Book)  private readonly AuthorDisplay authorDisplay = new AuthorDisplay();  private readonly BookDisplay bookDisplay = new BookDisplay();  private readonly BorrowedBookDisplay borrowedBookDisplay = new BorrowedBookDisplay();  private readonly MemberDisplay memberDisplay = new MemberDisplay();    // UI helper to assist with common input/output operations  private readonly UIHelper uiHelper = new UIHelper();    /// <summary>  /// The entry point for starting the display interaction.  /// </summary>  public static async Task OnStart()  {  var display = new Display();  await display.Input();  }    // Private constructor to prevent instantiation outside the class  private Display()  {  }    /// <summary>  /// Displays the main menu options for the library management system.  /// </summary>  public void ShowMenu()  {  uiHelper.ShowHeader("Library Management System");  Console.WriteLine("1. Books");  Console.WriteLine("2. Members");  Console.WriteLine("3. Authors");  Console.WriteLine("4. Borrowed Books History");  Console.WriteLine("5. Exit");  }    /// <summary>  /// Handles the user input and navigates to the appropriate display based on selection.  /// </summary>  private async Task Input()  {  var operation = -1;  do  {  ShowMenu(); // Show the menu options  operation = uiHelper.ReadIntInput("Please select an option:"); // Read user input    // Switch case for different menu options  switch (operation)  {  case 1:  await bookDisplay.BookManager(); // Navigate to book manager  break;  case 2:  await memberDisplay.MemberManager(); // Navigate to member manager  break;  case 3:  await authorDisplay.AuthorManager(); // Navigate to author manager  break;  case 4:  await borrowedBookDisplay.BorrowedBookManager(); // Navigate to borrowed book manager  break;  case 5:  Console.WriteLine("Exiting..."); // Inform the user that the program is exiting  break;  default:  Console.WriteLine("Invalid option selected, please try again."); // Handle invalid options  break;  }  } while (operation != 5); // Continue the loop until the user chooses to exit  }  }  } |

UIHelper.cs:

|  |
| --- |
| namespace ConsoleApp.Presentation  {  internal class UIHelper  {  /// <summary>  /// Displays a centered header with the given title.  /// </summary>  /// <param name="title">The title to display in the header.</param>  public void ShowHeader(string title)  {  int totalWidth = 40; // Set the header width    Console.WriteLine(new string('-', totalWidth)); // Top border  Console.WriteLine(title.PadLeft((totalWidth + title.Length) / 2).PadRight(totalWidth)); // Centered title  Console.WriteLine(new string('-', totalWidth)); // Bottom border  }    /// <summary>  /// Reads and returns a valid integer input from the user.  /// Keeps prompting until a valid integer is entered.  /// </summary>  /// <param name="prompt">The message to display to the user.</param>  public int ReadIntInput(string prompt)  {  int result;  while (true)  {  Console.WriteLine(prompt);  if (int.TryParse(Console.ReadLine(), out result) && result >= 0)  {  return result;  }  Console.WriteLine("Invalid input. Please enter a number.");  }  }    /// <summary>  /// Reads and returns a non-empty string input from the user.  /// Keeps prompting until a valid string is entered.  /// </summary>  /// <param name="prompt">The message to display to the user.</param>  public string ReadStringInput(string prompt)  {  string input;  while (true)  {  Console.WriteLine(prompt);  input = Console.ReadLine();    if (string.IsNullOrEmpty(input))  {  Console.WriteLine("Input cannot be empty. Please enter a valid string.");  continue;  }    return input;  }  }    /// <summary>  /// Reads and returns a valid DateTime input from the user in dd-MM-yyyy format.  /// Keeps prompting until a properly formatted and valid date is entered.  /// </summary>  /// <param name="prompt">The message to display to the user.</param>  /// <returns>A DateTime value representing the user's input.</returns>  public DateTime ReadDateInput(string prompt)  {  DateTime dateInput;  while (true)  {  Console.WriteLine($"{prompt} (dd-MM-yyyy)");  string input = Console.ReadLine();    if (string.IsNullOrEmpty(input))  {  Console.WriteLine("Input cannot be empty. Please enter a valid date.");  continue;  }    if (DateTime.TryParseExact(input, "dd-MM-yyyy",  System.Globalization.CultureInfo.InvariantCulture,  System.Globalization.DateTimeStyles.None, out dateInput))  {  return dateInput;  }  else  {  Console.WriteLine("Invalid date format. Please enter the date in dd-MM-yyyy format.");  }  }  }  }  } |

Test пример (BookBusinessTest):

|  |
| --- |
| using Business;  using Data;  using Data.Enums;  using Data.Models;  using Microsoft.EntityFrameworkCore;    namespace Tests  {  public class BookBusinessTests  {  [Test]  public async Task GetAllTest()  {  var options = new DbContextOptionsBuilder<LibraryDbContext>()  .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())  .Options;    // Insert seed data into the database using one instance of the context    using (var context = new LibraryDbContext(options))  {  context.Books.Add(new Book { Id = 1, Title = "Book 1", AuthorID = 1, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-1), ISBN = "0900000001" });  context.Books.Add(new Book { Id = 2, Title = "Book 2", AuthorID = 2, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-2), ISBN = "0900000002" });    context.SaveChanges();    BookBusiness bookBusiness = new BookBusiness(context);  List<Book> books = await bookBusiness.GetAllAsync();    Assert.That(books.Count, Is.EqualTo(2));  }  }    [Test]  public async Task GetAllWithIncludesTest()  {  var options = new DbContextOptionsBuilder<LibraryDbContext>()  .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())  .Options;    // Insert seed data into the database using one instance of the context    using (var context = new LibraryDbContext(options))  {  context.Authors.Add(new Author { Id = 1, FirstName = "Author", LastName = "1", Biography = "abcdefg", DateOfBirth = DateTime.Now.AddYears(-1), ImageUrl = "randomUrl" });  context.Authors.Add(new Author { Id = 2, FirstName = "Author", LastName = "2", Biography = "gfedcba", DateOfBirth = DateTime.Now.AddYears(-2), ImageUrl = "randomUrl" });    context.Books.Add(new Book { Id = 1, Title = "Book 1", AuthorID = 1, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-1), ISBN = "0900000001" });  context.Books.Add(new Book { Id = 2, Title = "Book 2", AuthorID = 2, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-2), ISBN = "0900000002" });    context.SaveChanges();  BookBusiness bookBusiness = new BookBusiness(context);  List<Book> books = await bookBusiness.GetAllWithIncludesAsync();  Assert.That(books.Count, Is.EqualTo(2));  Assert.That(books[0].Author, Is.Not.Null);  Assert.That(books[0].BorrowedBooks, Is.Not.Null);  }  }    [Test]  public async Task GetAllByGenreTest()  {  var options = new DbContextOptionsBuilder<LibraryDbContext>()  .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())  .Options;    // Insert seed data into the database using one instance of the context    using (var context = new LibraryDbContext(options))  {  context.Books.Add(new Book { Id = 1, Title = "Book 1", AuthorID = 1, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-1), ISBN = "0900000001" });  context.Books.Add(new Book { Id = 2, Title = "Book 2", AuthorID = 2, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-2), ISBN = "0900000002" });  context.Books.Add(new Book { Id = 3, Title = "Book 3", AuthorID = 3, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-3), ISBN = "0900000003" });    context.SaveChanges();    BookBusiness bookBusiness = new BookBusiness(context);  List<Book> booksFiction = await bookBusiness.GetAllByGenreAsync("Fiction");  List<Book> booksNonFiction = await bookBusiness.GetAllByGenreAsync("NonFiction");    Assert.That(booksFiction.Count, Is.EqualTo(1));  Assert.That(booksFiction[0].Genre, Is.EqualTo(Genre.Fiction));    Assert.That(booksNonFiction.Count, Is.EqualTo(2));  Assert.That(booksNonFiction[0].Genre, Is.EqualTo(Genre.NonFiction));  }  }    [Test]  public async Task GetAllByGenreWithIncludesTest()  {  var options = new DbContextOptionsBuilder<LibraryDbContext>()  .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())  .Options;    // Insert seed data into the database using one instance of the context    using (var context = new LibraryDbContext(options))  {  context.Authors.Add(new Author { Id = 1, FirstName = "Author", LastName = "1", Biography = "abcdefg", DateOfBirth = DateTime.Now.AddYears(-1), ImageUrl = "randomUrl" });  context.Authors.Add(new Author { Id = 2, FirstName = "Author", LastName = "2", Biography = "gfedcba", DateOfBirth = DateTime.Now.AddYears(-2), ImageUrl = "randomUrl" });  context.Authors.Add(new Author { Id = 3, FirstName = "Author", LastName = "3", Biography = "gfedcba", DateOfBirth = DateTime.Now.AddYears(-3), ImageUrl = "randomUrl" });    context.Books.Add(new Book { Id = 1, Title = "Book 1", AuthorID = 1, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-1), ISBN = "0900000001" });  context.Books.Add(new Book { Id = 2, Title = "Book 2", AuthorID = 2, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-2), ISBN = "0900000002" });  context.Books.Add(new Book { Id = 3, Title = "Book 3", AuthorID = 3, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-3), ISBN = "0900000003" });    context.SaveChanges();    BookBusiness bookBusiness = new BookBusiness(context);  List<Book> booksFiction = await bookBusiness.GetAllByGenreWithIncludesAsync("Fiction");  List<Book> booksNonFiction = await bookBusiness.GetAllByGenreWithIncludesAsync("NonFiction");    Assert.That(booksFiction.Count, Is.EqualTo(1));  Assert.That(booksFiction[0].Genre, Is.EqualTo(Genre.Fiction));    Assert.That(booksNonFiction.Count, Is.EqualTo(2));  Assert.That(booksNonFiction[0].Genre, Is.EqualTo(Genre.NonFiction));    Assert.That(booksFiction[0].Author, Is.Not.Null);  Assert.That(booksNonFiction[0].Author, Is.Not.Null);  Assert.That(booksNonFiction[1].Author, Is.Not.Null);  }  }    [Test]  public async Task GetTest()  {  var options = new DbContextOptionsBuilder<LibraryDbContext>()  .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())  .Options;    // Insert seed data into the database using one instance of the context    using (var context = new LibraryDbContext(options))  {  context.Books.Add(new Book { Id = 1, Title = "Book 1", AuthorID = 1, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-1), ISBN = "0900000001" });  context.Books.Add(new Book { Id = 2, Title = "Book 2", AuthorID = 2, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-2), ISBN = "0900000002" });    context.SaveChanges();    BookBusiness bookBusiness = new BookBusiness(context);  Book book = await bookBusiness.GetAsync(1);    Assert.That(book, Is.Not.Null);  Assert.That(book.Id, Is.EqualTo(1));  Assert.That(book, Is.EqualTo(context.Books.Find(1)));  }  }    [Test]  public async Task GetWithIncludesTest()  {  var options = new DbContextOptionsBuilder<LibraryDbContext>()  .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())  .Options;    // Insert seed data into the database using one instance of the context    using (var context = new LibraryDbContext(options))  {  context.Authors.Add(new Author { Id = 1, FirstName = "Author", LastName = "1", Biography = "abcdefg", DateOfBirth = DateTime.Now.AddYears(-1), ImageUrl = "randomUrl" });  context.Authors.Add(new Author { Id = 2, FirstName = "Author", LastName = "2", Biography = "gfedcba", DateOfBirth = DateTime.Now.AddYears(-2), ImageUrl = "randomUrl" });    context.Books.Add(new Book { Id = 1, Title = "Book 1", AuthorID = 1, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-1), ISBN = "0900000001" });  context.Books.Add(new Book { Id = 2, Title = "Book 2", AuthorID = 2, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-2), ISBN = "0900000002" });    context.SaveChanges();    BookBusiness bookBusiness = new BookBusiness(context);  Book book = await bookBusiness.GetWithIncludesAsync(1);    Assert.That(book, Is.Not.Null);  Assert.That(book.Id, Is.EqualTo(1));  Assert.That(book, Is.EqualTo(context.Books.Find(1)));    Assert.That(book.Author, Is.Not.Null);  Assert.That(book.BorrowedBooks, Is.Not.Null);    Assert.That(book.Author, Is.EqualTo(context.Books.Include(b => b.Author).Include(b => b.BorrowedBooks).First(x => x.Id == 1).Author));  }  }    [Test]  public async Task GetByISBNTest()  {  var options = new DbContextOptionsBuilder<LibraryDbContext>()  .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())  .Options;    // Insert seed data into the database using one instance of the context    using (var context = new LibraryDbContext(options))  {  context.Books.Add(new Book { Id = 1, Title = "Book 1", AuthorID = 1, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-1), ISBN = "0900000001" });  context.Books.Add(new Book { Id = 2, Title = "Book 2", AuthorID = 2, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-2), ISBN = "0900000002" });    context.SaveChanges();    BookBusiness bookBusiness = new BookBusiness(context);  Book book = await bookBusiness.GetByISBNAsync("0900000001");    Assert.That(book, Is.Not.Null);  Assert.That(book.Id, Is.EqualTo(1));  Assert.That(book.ISBN, Is.EqualTo("0900000001"));  Assert.That(book, Is.EqualTo(context.Books.First(x => x.ISBN == "0900000001")));  }  }    [Test]  public async Task GetByISBNWithIncludesTest()  {  var options = new DbContextOptionsBuilder<LibraryDbContext>()  .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())  .Options;    // Insert seed data into the database using one instance of the context    using (var context = new LibraryDbContext(options))  {  context.Authors.Add(new Author { Id = 1, FirstName = "Author", LastName = "1", Biography = "abcdefg", DateOfBirth = DateTime.Now.AddYears(-1), ImageUrl = "randomUrl" });  context.Authors.Add(new Author { Id = 2, FirstName = "Author", LastName = "2", Biography = "gfedcba", DateOfBirth = DateTime.Now.AddYears(-2), ImageUrl = "randomUrl" });    context.Books.Add(new Book { Id = 1, Title = "Book 1", AuthorID = 1, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-1), ISBN = "0900000001" });  context.Books.Add(new Book { Id = 2, Title = "Book 2", AuthorID = 2, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-2), ISBN = "0900000002" });    context.SaveChanges();    BookBusiness bookBusiness = new BookBusiness(context);  Book book = await bookBusiness.GetByISBNWithIncludesAsync("0900000001");    Assert.That(book, Is.Not.Null);  Assert.That(book.Id, Is.EqualTo(1));  Assert.That(book.ISBN, Is.EqualTo("0900000001"));  Assert.That(book, Is.EqualTo(context.Books.First(x => x.ISBN == "0900000001")));    Assert.That(book.Author, Is.Not.Null);  Assert.That(book.BorrowedBooks, Is.Not.Null);    Assert.That(book.Author, Is.EqualTo(context.Books.Include(b => b.Author).Include(b => b.BorrowedBooks).First(x => x.ISBN == "0900000001").Author));  }  }    [Test]  public async Task AddTest()  {  var options = new DbContextOptionsBuilder<LibraryDbContext>()  .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())  .Options;    // Insert seed data into the database using one instance of the context    using (var context = new LibraryDbContext(options))  {  context.Books.Add(new Book { Id = 1, Title = "Book 1", AuthorID = 1, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-1), ISBN = "0900000001" });  context.Books.Add(new Book { Id = 2, Title = "Book 2", AuthorID = 2, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-2), ISBN = "0900000002" });    context.SaveChanges();    BookBusiness bookBusiness = new BookBusiness(context);  Book book = new Book { Id = 3, Title = "Book 3", AuthorID = 3, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-3), ISBN = "0900000003" };  await bookBusiness.AddAsync(book);    Assert.That(context.Books, Has.Exactly(1).EqualTo(book));  }  }    [Test]  public async Task UpdateTest()  {  var options = new DbContextOptionsBuilder<LibraryDbContext>()  .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())  .Options;    // Insert seed data into the database using one instance of the context    using (var context = new LibraryDbContext(options))  {  context.Books.Add(new Book { Id = 1, Title = "Book 1", AuthorID = 1, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-1), ISBN = "0900000001" });  context.Books.Add(new Book { Id = 2, Title = "Book 2", AuthorID = 2, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-2), ISBN = "0900000002" });    context.SaveChanges();    BookBusiness bookBusiness = new BookBusiness(context);  Book oldBook = new Book { Id = 2, Title = "Book 2", AuthorID = 2, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-2), ISBN = "0900000002" };  Book book = new Book { Id = 2, Title = "Book 2.1", AuthorID = 1, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-3), ISBN = "0900000011" };  await bookBusiness.UpdateAsync(book);    Assert.That(context.Books, Has.Exactly(0).EqualTo(oldBook));    Assert.That(context.Books.Count(), Is.EqualTo(2));    Assert.That(context.Books.First(x => x.Id == 2).Title, Is.EqualTo("Book 2.1"));  Assert.That(context.Books.First(x => x.Id == 2).PublicationDate, Is.EqualTo(context.Books.Find(2).PublicationDate));  Assert.That(context.Books.First(x => x.Id == 2).ISBN, Is.EqualTo("0900000011"));  Assert.That(context.Books.First(x => x.Id == 2).Genre, Is.EqualTo(Genre.Fiction));  }  }    [Test]  public async Task DeleteTest()  {  var options = new DbContextOptionsBuilder<LibraryDbContext>()  .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())  .Options;    // Insert seed data into the database using one instance of the context    using (var context = new LibraryDbContext(options))  {  context.Books.Add(new Book { Id = 1, Title = "Book 1", AuthorID = 1, Genre = Genre.Fiction, PublicationDate = DateTime.Now.AddYears(-1), ISBN = "0900000001" });  context.Books.Add(new Book { Id = 2, Title = "Book 2", AuthorID = 2, Genre = Genre.NonFiction, PublicationDate = DateTime.Now.AddYears(-2), ISBN = "0900000002" });    context.SaveChanges();    BookBusiness bookBusiness = new BookBusiness(context);  await bookBusiness.DeleteAsync(2);    Assert.That(context.Books, Has.Exactly(0).EqualTo(context.Books.Find(2)));    Assert.That(context.Books.Count(), Is.EqualTo(1));  }  }  }  } |

Controller пример (BookController):

|  |
| --- |
| using Business;  using Data;  using Data.Models;  using Microsoft.AspNetCore.Mvc;  using Microsoft.AspNetCore.Mvc.Rendering;    namespace WebApp.Controllers  {  public class BooksController : Controller  {  private readonly BookBusiness \_bookBusiness;  private readonly AuthorBusiness \_authorBusiness;    public BooksController(LibraryDbContext context)  {  \_authorBusiness = new AuthorBusiness(context);  \_bookBusiness = new BookBusiness(context);  }    // GET: /Books  public async Task<IActionResult> Index()  {  var books = await \_bookBusiness.GetAllWithIncludesAsync();  return View(books);  }    // GET: /Books/Details/5  public async Task<IActionResult> Details(int id)  {  var book = await \_bookBusiness.GetWithIncludesAsync(id);  if (book == null)  return NotFound();    return View(book);  }    // GET: /Books/Create  public async Task<IActionResult> Create()  {  ViewData["AuthorId"] = new SelectList(  \_authorBusiness.GetAllAsync().Result.Select(a => new SelectListItem  {  Value = a.Id.ToString(),  Text = $"{a.FirstName} {a.LastName}" // Concatenating FirstName and LastName  }),  "Value", "Text");    return View();  }      // POST: /Books/Create  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Create(Book book)  {  if (ModelState.IsValid)  {  await \_bookBusiness.AddAsync(book);  return RedirectToAction(nameof(Index));  }  return View(book);  }    // GET: /Books/Edit/5  public async Task<IActionResult> Edit(int id)  {  var book = await \_bookBusiness.GetWithIncludesAsync(id);  if (book == null)  return NotFound();    ViewData["AuthorId"] = new SelectList(  \_authorBusiness.GetAllAsync().Result.Select(a => new SelectListItem  {  Value = a.Id.ToString(),  Text = $"{a.FirstName} {a.LastName}" // Concatenating FirstName and LastName  }),  "Value", "Text");    return View(book);  }    // POST: /Books/Edit/5  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Edit(Book book)  {  if (ModelState.IsValid)  {  await \_bookBusiness.UpdateAsync(book);  return RedirectToAction(nameof(Index));  }  return View(book);  }    // GET: /Books/Delete/5  public async Task<IActionResult> Delete(int id)  {  var book = await \_bookBusiness.GetWithIncludesAsync(id);  if (book == null)  return NotFound();    return View(book);  }    // POST: /Books/Delete/5  [HttpPost, ActionName("Delete")]  [ValidateAntiForgeryToken]  public async Task<IActionResult> DeleteConfirmed(int id)  {  await \_bookBusiness.DeleteAsync(id);  return RedirectToAction(nameof(Index));  }  }  } |

DbSeeder.cs:

|  |
| --- |
| using Data; using Data.Enums; using Data.Models; using Microsoft.EntityFrameworkCore;  namespace WebApp.Seeders { public class DbSeeder { private readonly LibraryDbContext \_context; public DbSeeder(LibraryDbContext context) { \_context = context; }  public async Task Seed()  {  await \_context.Database.EnsureDeletedAsync();  await \_context.Database.EnsureCreatedAsync();   // Create Lists for seed data  var authors = new List<Author>  {  new Author { FirstName = "George", LastName = "Orwell", DateOfBirth = new DateTime(1903, 6, 25), Biography = "British novelist and essayist", ImageUrl = "<https://upload.wikimedia.org/wikipedia/commons/thumb/7/7e/George_Orwell_press_photo.jpg/1024px-George_Orwell_press_photo.jpg>"},  new Author { FirstName = "Jane", LastName = "Austen", DateOfBirth = new DateTime(1775, 12, 16), Biography = "Renowned English novelist", ImageUrl = "<https://upload.wikimedia.org/wikipedia/commons/thumb/c/cc/CassandraAusten-JaneAusten%28c.1810%29_hires.jpg/1024px-CassandraAusten-JaneAusten%28c.1810%29_hires.jpg>" },  new Author { FirstName = "J.K.", LastName = "Rowling", DateOfBirth = new DateTime(1965, 7, 31), Biography = "Author of Harry Potter series", ImageUrl = "<https://upload.wikimedia.org/wikipedia/commons/thumb/5/5d/J._K._Rowling_2010.jpg/250px-J._K._Rowling_2010.jpg>"},  new Author { FirstName = "Stephen", LastName = "King", DateOfBirth = new DateTime(1947, 9, 21), Biography = "King of horror and supernatural fiction", ImageUrl = "<https://upload.wikimedia.org/wikipedia/commons/thumb/2/24/Stephen_King_at_the_2024_Toronto_International_Film_Festival_2_%28cropped%29.jpg/250px-Stephen_King_at_the_2024_Toronto_International_Film_Festival_2_%28cropped%29.jpg>" },  new Author { FirstName = "Mark", LastName = "Twain", DateOfBirth = new DateTime(1835, 11, 30), Biography = "Famous American humorist and writer", ImageUrl = "<https://upload.wikimedia.org/wikipedia/commons/0/0c/Mark_Twain_by_AF_Bradley.jpg>"}  };   var members = new List<Member>  {  new Member { FirstName = "Alice", LastName = "Johnson", MembershipExpireDate = new DateTime(2026, 5, 1), PhoneNumber = "555-12345678" },  new Member { FirstName = "Bob", LastName = "Smith", MembershipExpireDate = new DateTime(2026, 4, 15), PhoneNumber = "555-87654321" },  new Member { FirstName = "Carol", LastName = "Martinez", MembershipExpireDate = new DateTime(2026, 7, 30), PhoneNumber = "555-56789012" },  new Member { FirstName = "Dave", LastName = "Brown", MembershipExpireDate = new DateTime(2024, 3, 10), PhoneNumber = "555-43210987" }, // EXPIRED  new Member { FirstName = "Eve", LastName = "Davis", MembershipExpireDate = new DateTime(2026, 9, 20), PhoneNumber = "555-34567890" }  };   // Insert Authors and save changes  await \_context.Authors.AddRangeAsync(authors);  await \_context.SaveChangesAsync(); // Save authors to generate IDs   // Retrieve authors with their IDs  var authorList = await \_context.Authors.ToListAsync();   // Now that we have the author IDs, create books and link them to the correct author IDs  var books = new List<Book>  {  new Book { Title = "1984", Genre = Genre.ScienceFiction, ISBN = "9780451524935", PublicationDate = new DateTime(1949, 6, 8), AuthorID = 1},  new Book { Title = "Animal Farm", Genre = Genre.Fiction, ISBN = "9780451526342", PublicationDate = new DateTime(1945, 8, 17), AuthorID = 1},  new Book { Title = "Pride and Prejudice", Genre = Genre.Romance, ISBN = "9780679783268", PublicationDate = new DateTime(1813, 1, 28), AuthorID = 2},  new Book { Title = "Sense and Sensibility", Genre = Genre.Romance, ISBN = "9780141439662", PublicationDate = new DateTime(1811, 10, 30), AuthorID = 2},  new Book { Title = "Harry Potter and the Sorcerer's Stone", Genre = Genre.Fantasy, ISBN = "9780439554930", PublicationDate = new DateTime(1997, 6, 26), AuthorID = 3},  new Book { Title = "Harry Potter and the Chamber of Secrets", Genre = Genre.Fantasy, ISBN = "9780439064873", PublicationDate = new DateTime(1998, 7, 2), AuthorID = 3},  new Book { Title = "The Shining", Genre = Genre.Horror, ISBN = "9780385121675", PublicationDate = new DateTime(1977, 1, 28), AuthorID = 4},  new Book { Title = "It", Genre = Genre.Horror, ISBN = "9780451169518", PublicationDate = new DateTime(1986, 9, 15), AuthorID = 4},  new Book { Title = "Adventures of Huckleberry Finn", Genre = Genre.HistoricalFiction, ISBN = "9780486280615", PublicationDate = new DateTime(1884, 12, 10), AuthorID = 4},  new Book { Title = "The Adventures of Tom Sawyer", Genre = Genre.HistoricalFiction, ISBN = "9780486400778", PublicationDate = new DateTime(1876, 6, 1), AuthorID = 4}  };   // Insert Members and save changes  await \_context.Members.AddRangeAsync(members);  await \_context.SaveChangesAsync(); // Save members to generate IDs   // Insert Books and save changes  await \_context.Books.AddRangeAsync(books);  await \_context.SaveChangesAsync(); // Save books to generate BookIDs   // Create and insert Borrowed Books  var borrowedBooks = new List<BorrowedBook>  {  new BorrowedBook { BookID = books[0].Id, MemberID = members[0].Id, BorrowDate = new DateTime(2025, 4, 1), DueDate = new DateTime(2025, 4, 15), ReturnDate = new DateTime(2025, 4, 10) },  new BorrowedBook { BookID = books[1].Id, MemberID = members[0].Id, BorrowDate = new DateTime(2025, 4, 20), DueDate = new DateTime(2025, 5, 4), ReturnDate = null },  new BorrowedBook { BookID = books[2].Id, MemberID = members[1].Id, BorrowDate = new DateTime(2025, 3, 15), DueDate = new DateTime(2025, 3, 29), ReturnDate = new DateTime(2025, 3, 28) },  new BorrowedBook { BookID = books[3].Id, MemberID = members[1].Id, BorrowDate = new DateTime(2025, 4, 5), DueDate = new DateTime(2025, 4, 19), ReturnDate = null },  new BorrowedBook { BookID = books[4].Id, MemberID = members[2].Id, BorrowDate = new DateTime(2025, 2, 10), DueDate = new DateTime(2025, 2, 24), ReturnDate = new DateTime(2025, 2, 25) },  new BorrowedBook { BookID = books[5].Id, MemberID = members[2].Id, BorrowDate = new DateTime(2025, 4, 10), DueDate = new DateTime(2025, 4, 24), ReturnDate = null },  new BorrowedBook { BookID = books[6].Id, MemberID = members[3].Id, BorrowDate = new DateTime(2025, 3, 1), DueDate = new DateTime(2025, 3, 15), ReturnDate = new DateTime(2025, 3, 12) },  new BorrowedBook { BookID = books[7].Id, MemberID = members[4].Id, BorrowDate = new DateTime(2025, 3, 20), DueDate = new DateTime(2025, 4, 3), ReturnDate = new DateTime(2025, 4, 1) },  new BorrowedBook { BookID = books[8].Id, MemberID = members[4].Id, BorrowDate = new DateTime(2025, 4, 18), DueDate = new DateTime(2025, 5, 2), ReturnDate = null }  };   // Insert Borrowed Books and save changes  await \_context.BorrowedBooks.AddRangeAsync(borrowedBooks);  await \_context.SaveChangesAsync(); // Save borrowed books  } } |

Покритие на тестовете върху business слоя

1. **Развитие и нововъведения**

Към проекта може да се добави Windows Form интерфейс, както и да се надгради с профили в системата, като има различни нива потребители. Така приложенията ще могат да се ползват и от потребителите онлайн за да заемат книги, проверяват наличности и др. данни от дома си.

1. **Заключение**

В хода на разработката на проекта успях да приложа на практика знанията си от Модул 7, свързани с разработката на многослойни приложения, бази данни и уеб технологии. Чрез реализирането на библиотечната система изградих пълно функционално приложение с няколко интерфейса (конзолен и уеб), валидирана бизнес логика и автоматизирани тестове. Работата по проекта ми помогна да усъвършенствам уменията си в работата с Entity Framework Core, ASP.NET MVC, тестове с NUnit, както и в изграждането на добре структурирана и мащабируема архитектура. Смятам, че проектът изпълнява поставените цели и демонстрира ключови практики в реалната софтуерна разработка.

1. **Използвана литература**
2. Microsoft Docs – [Entity Framework Core Documentation](https://learn.microsoft.com/en-us/ef/core/)
3. Microsoft Docs – [ASP.NET Core MVC Documentation](https://learn.microsoft.com/en-us/aspnet/core/mvc/)
4. Microsoft Docs – [Razor Pages Documentation](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/)
5. Microsoft Docs – [Unit Testing in .NET](https://learn.microsoft.com/en-us/dotnet/core/testing/)
6. NUnit Documentation – [https://nunit.org](https://nunit.org/)