BOOK MANAGEMENT SYSTEM

.NET TECHNOLOGY (01CE0523)

MINI PROJECT REPORT

Submitted by:

92410103111	**MADHAV PANDYA**
92410103053	**AAYUSH NIMAVAT**
92410103126	**ARYAN AGHERA**

BACHELOR OF TECHNOLOGY

in

Computer Engineering

Marwadi University, Rajkot

November, 2025

.NET Technologies (01CE0523)

Mini Project

Faculty of Technology, Marwadi University

Computer Engineering Department

2025-26

CERTIFICATE

This is to certify that the project report submitted along with the project entitled BOOK MANAGEMENT SYSTEM has been carried out by 92410103126 (Enrollment) under my guidance in partial fulfillment for the degree of Bachelor of Technology in Computer Engineering, 5th Semester of Marwadi University, Rajkot during the academic year 2025-26.

Sign:

Prof. Chetankumar Chudasama

Internal Guide

Sign:

Dr. Krunal Vaghela

Head of the Department

Index

Acknowledgments

This mini project would not have been possible without the Guidance and the help of several Individuals who in some ways contributed their valuable time and assistance in the completion of this Internship.

We/I would like to thank our/my college **Marwadi University** for giving us this opportunity of doingInternship and gaining Industry Level Experience in the field that we/I are interested in. Our/My sincere gratitude to our Head of The Department (HOD), **Dr. Krunal Vaghela**, our/my Internal Guide, **Prof. Chetankumar Chudasama** for helping and solving our queries whenever required.

Abstract

This project, the Book Management System, developed using .
NET Technology, is a web-based solution designed to streamline and automate the management of book inventory.

The core functionality includes CRUD (Create, Read, Update, Delete) operations, allowing a user to easily add new books (capturing details like Title, Price, and Quantity), view the complete inventory, and modify or remove entries.

The system prioritizes a User-Friendly Interface and robust Search and Filter capabilities, ensuring efficient and error-free record keeping.

This report details the system's architecture (likely ASP.NET MVC), design, implementation using C# and Entity Framework, and testing.

1. Introduction

1.1 Project Title and Context

The project presented in this report is titled the Book Management System. It was developed as a Mini Project requirement for the .NET Technology (01CE0523) course for the degree of Bachelor of Technology in Computer Engineering at Marwadi University, Rajkot. The system leverages the powerful capabilities of the Microsoft .NET platform to provide a robust and efficient solution for inventory management. The development was carried out during the academic year 2025-26.

1.2 Problem Definition

In environments that handle a collection of books—whether a small library, a college department, or a personal inventory—maintaining records using traditional, manual methods (such as physical ledgers or rudimentary spreadsheets) presents several key challenges:

- Inefficiency in Operations: Simple tasks like adding a new book, checking current stock, or identifying a specific title can be time-consuming.
- Data Integrity Risks: Manual entry and upkeep are prone to human errors, leading to incorrect prices, inaccurate quantity counts, or misspelled titles.
- Lack of Accessibility: Information is often decentralized and not easily accessible or searchable by multiple users simultaneously.
- Limited Scalability: As the number of books grows, the difficulty and time required for management increase exponentially.

This project addresses these inherent drawbacks by proposing a digital and automated system.

1.3 Project Objective and Goals

The primary objective of the Book Management System is to design and implement a streamlined, web-based application to effectively manage a collection of book records.

The specific goals achieved by this project are:

- Provide Core CRUD Functionality: Enable users to easily Create, Read (View), Update (Edit), and Delete book records .
- Intuitive Data Entry: Develop a simple and clear interface, such as the Create Book page, to capture essential details like the book’s Title, Price, and Quantity .
- Enhance Usability: Ensure the system possesses a User-Friendly Interface that is easy to navigate, improving the overall user experience .
- Facilitate Search and Retrieval: Incorporate features that allow users to quickly find the books they need using powerful search and filter options .
- Utilize .NET Technologies: Apply the Model-View-Controller (MVC) architectural pattern and Entity Framework (EF) to demonstrate proficiency in .NET Technology development.

1.4 Proposed Solution: The Book Management System

The Book Management System is a robust, data-driven web application built upon the modern and scalable ASP.NET platform. The solution provides a dedicated digital database for storing book information, making it reliable and instantly accessible.

1.4.1 Key Features

The system is designed around three main feature areas, highlighted on the welcome screen :

1. Easy Book Management:
    - Allows adding new records with crucial financial and inventory data (Price and Quantity) .
    - Supports modification of any existing book's details.
    - Enables permanent removal of book records from the inventory.
    - Utilizes a central database to ensure single-source data integrity.
2. Search and Filter:
    - Implemented to allow users to quickly locate specific books.
    - This feature is crucial for larger inventories, replacing slow manual scanning with instantaneous digital queries.
3. User-Friendly Interface:
    - The application is designed with users in mind, ensuring it is intuitive and easy to navigate .
    - The clear layout and minimal design reduce the learning curve and improve efficiency.

1.4.2 Technical Approach

The project employs a structured, object-oriented approach to development, adhering to industry best practices. The ASP.NET framework naturally supports the Model-View-Controller (MVC) architectural pattern. This separation of concerns ensures that the application is modular, maintainable, and scalable.

- Model: Defines the Book data structure and handles the business logic.
- View: The HTML and Razor pages that present the user interface (e.g., the Create Book form).
- Controller: Manages user requests (e.g., button clicks like "Create") and interacts with the Model to perform database operations.

1.5 Target Users and Scope

1.5.1 Target Users

The system is primarily designed for administrative staff, inventory managers, or students/faculty responsible for managing a specific collection of books. The ease of use ensures that individuals without extensive technical knowledge can operate the system effectively.

1.5.2 Project Scope

The scope of this mini-project is clearly defined and focused on core functionality:

- In-Scope:
    - Development of a secure, single-user interface for CRUD operations.
    - Implementation of client-side and server-side validation for data input (e.g., ensuring Price and Quantity are non-negative numeric values).
    - Persistence of data using an integrated database system.
    - Deployment as a basic web application.
- Out-of-Scope (Future Enhancements):
    - Advanced features such as user authentication (login/security roles).
    - Complex inventory features like tracking book borrowing/lending (library management).
    - Integration with external APIs or payment gateways.

1.6 Report Organization

This mini-project report is organized to provide a detailed understanding of the system, its development process, and the technologies utilized.

- Section 2 details the specific Technology used and Implementation Strategy, including the architecture, database design, and key components of the .NET application.
- Section 3 provides the Implementation Snapshots, visually confirming the functionality of the Home and Create Book pages.
- Section 4 summarizes the findings and achievements in the Conclusion.
- Section 5 lists all external References consulted during the project's development.

2. Technology used and Implementation Strategy

2.1 Core Technology Stack: The .NET Platform

The **Book Management System** is fundamentally built upon the **Microsoft .NET Platform** (specifically, **ASP.NET Core** or **ASP.NET MVC** – *Choose one and be consistent*). The selection of the .NET ecosystem was strategic due to its robustness, extensive tooling, and strong support for creating secure, data-driven web applications using the **C#** programming language.

2.1.1 Programming Language: C#

C# (C-Sharp) is the primary language used for all backend logic, including controllers, models, and data access layers. C# is an object-oriented, modern language that natively supports features critical to enterprise-level development, such as:

- **Strong Typing:** Enhances code reliability and helps catch errors during compilation.
- **Object-Oriented Programming (OOP):** Facilitates modular design through concepts like encapsulation, inheritance, and polymorphism, which are essential for creating maintainable components like the Book model and associated services.
- **Language Integrated Query (LINQ):** Provides a unified syntax for querying data from various sources (like SQL databases via EF Core), simplifying complex data manipulation logic.

2.1.2 Web Framework: ASP.NET (Core/MVC)

The application's presentation layer and request handling are managed by the ASP.NET framework. The choice between **ASP.NET MVC** (a legacy framework) and **ASP.NET Core** (the newer, cross-platform framework) significantly impacts the project:

- **If ASP.NET Core is used:** This choice provides benefits like cross-platform deployment, enhanced performance, and native support for Dependency Injection (DI). It is the modern recommendation for new development.
- **If ASP.NET MVC is used (Older):** This framework is Windows-exclusive but is highly testable and allows great flexibility in HTML/UI customization.

For this report, we will assume the modern approach, using ASP.NET Core MVC, for building full-stack web apps.

2.1.3 Database and ORM: SQL Server and Entity Framework Core (EF Core)

- **Database (DB): Microsoft SQL Server** is used for persistence, providing a robust, relational store for the book inventory data.
- **Object-Relational Mapper (ORM): Entity Framework Core (EF Core)** serves as the crucial bridge between the C# object models and the relational database. It eliminates the need for writing repetitive, error-prone SQL commands for standard CRUD operations, enhancing developer efficiency and code cleanliness. EF Core is designed with Dependency Injection support, aiding in testability.

2.2 Frontend Technologies

The user interface is constructed using standard web technologies to ensure accessibility and responsiveness:

- **HTML5 & CSS3:** Provide the structure and styling of the web pages.
- **Razor Syntax:** Used within the ASP.NET Views (.cshtml files) to embed server-side C# code seamlessly within HTML, dynamically rendering data from the model.
- **Bootstrap:** A popular CSS framework utilized for responsive design, ensuring the application is visually appealing and functional across different screen sizes (desktop, tablet, and mobile).

2.3 System Architecture: Three-Tier Model

The **Book Management System** implements a **Three-Tier Architecture** to ensure a high degree of maintainability, modularity, and scalability. This pattern logically separates the application into distinct layers, often mapped closely to the structural separation provided by the MVC pattern.

2.3.1 Tier 1: Presentation Layer

- **Responsibility:** Handles all user interaction, displaying information, and accepting input.
- **Components: Views** (HTML/Razor Pages) and **Controllers** (the entry point for user requests).
- **Functionality:** Translates tasks and results into a format the user can understand. For example, the Create Book View presents the form fields for Title, Price, and Quantity, and the Controller receives the HTTP POST request upon form submission.

2.3.2 Tier 2: Business Logic Layer (BLL)

- **Responsibility:** Contains all the application's core business rules, validation, and logic. It acts as middleware, processing commands and making logical decisions.
- **Components:** C# services and classes, often independent of the web framework.
- **Functionality:** This layer validates the input data received from the Presentation Layer (e.g., checks if the Price is positive and the Quantity is an integer), and then processes the request before communicating with the Data Layer.

2.3.3 Tier 3: Data Access Layer (DAL)

- **Responsibility:** Manages all direct communication with the database.
- **Components: DbContext** class and related repository patterns (if used). This layer uses EF Core methods to perform database operations (Insert, Update, Delete, Select).
- **Functionality:** Establishes the connection, translates **LINQ** queries into **SQL** commands, and retrieves or manipulates data. This layer is abstracted away from the BLL and Presentation Layer, allowing the database technology to be changed with minimal impact on the rest of the application.

2.4 Data Model Design

The system revolves around the core entity, **Book**. The database schema is designed using a **Code-First** approach with Entity Framework Core, where the C# class defines the database table structure.

Field Name	Data Type (C#)	Data Type (SQL)	Description	Constraints
BookID	int	INT	Unique identifier for each book record.	**Primary Key**, Identity (Auto-increment)
Title	string	NVARCHAR(MAX)	The title of the book.	Required Field
Price	decimal	DECIMAL	The price of the book.	Required, Must be non-negative
Quantity	int	INT	The number of copies currently in stock.	Required, Must be non-negative

2. Technology Used and Implementation Strategy (Continued)

(Page 6)

2.5 Implementation Workflow: CRUD Operations

The implementation strategy focuses on the key transactional workflow, particularly the process of creating a new book record, which integrates all three architectural tiers.

2.5.1 The "Create Book" Process

This process, triggered by the user interacting with the form , follows a standard lifecycle:

1. **User Interaction (Presentation Layer):**
    - The user fills the **Title, Price, and Quantity** fields on the "Add New Book" page.
    - The user clicks the **"Create"** button.
    - The web browser sends an HTTP POST request containing the form data to the corresponding Controller Action.
2. **Request Handling (Controller/BLL Interface):**
    - The **BooksController** receives the request and automatically maps the incoming form data to a Book model object (Model Binding).
    - **Server-Side Validation:** The Controller or a dedicated service in the BLL checks the model state to ensure data integrity (e.g., checks if Title is provided, and Price and Quantity

are valid numbers). If validation fails, the model is sent back to the View with error messages.

3. **Data Persistence (BLL to DAL):**
    - If validation is successful, the Controller calls a method (e.g., AddBook) in the **Business Logic Layer** (or a Repository in the DAL).
    - The Data Access Layer (DAL) uses the EF Core **DbContext** to execute the insert operation.

2.6 Key EF Core Operations (DAL)

The heart of the Data Access Layer relies on the seamless integration of C# with the database through EF Core.

2.6.1 Adding a New Record (Insert)

The C# code snippet for inserting a new book record demonstrates the simplicity of using EF Core methods, where the ORM automatically generates and executes the necessary SQL INSERT statement upon calling SaveChanges().

C#

```
// Inside the Data Access or Repository Class
public void AddBook(Book newBook)
{
   // The DbContext instance is typically injected via Dependency Injection
   _context.Books.Add(newBook);
   // EF Core tracks the entity state as 'Added'

   _context.SaveChanges();
   // Executes the SQL INSERT statement to the database
}
```

2.6.2 Retrieving Records (Read)

Fetching the inventory list for the **"View Books"** page is achieved using LINQ (Language Integrated Query):

C#

```
// Inside the Controller/Service
public List<Book> GetAllBooks()
{
   // LINQ query to select all books
   return _context.Books.ToList();
   // EF Core translates this to 'SELECT * FROM Books'
}
```

2.7 Advanced Implementation Concepts: Dependency Injection

A crucial feature leveraged from ASP.NET Core (or added via a third-party container in older MVC) is Dependency Injection (DI).

2.7.1 Role of Dependency Injection

DI is a software design pattern that enables loose coupling between components, greatly simplifying testing and maintenance. Instead of a class creating its own dependencies (e.g., a Controller manually creating a DbContext), the dependencies are "injected" from an external source (the framework's service container).

- In this project, the DbContext and any Business Logic Services are registered in the application's startup configuration.
- The Controller (Presentation Layer) then requests these dependencies via its constructor.

This approach ensures the Controller is unaware of how the data access is initialized, adhering to the principle of separation of concerns.

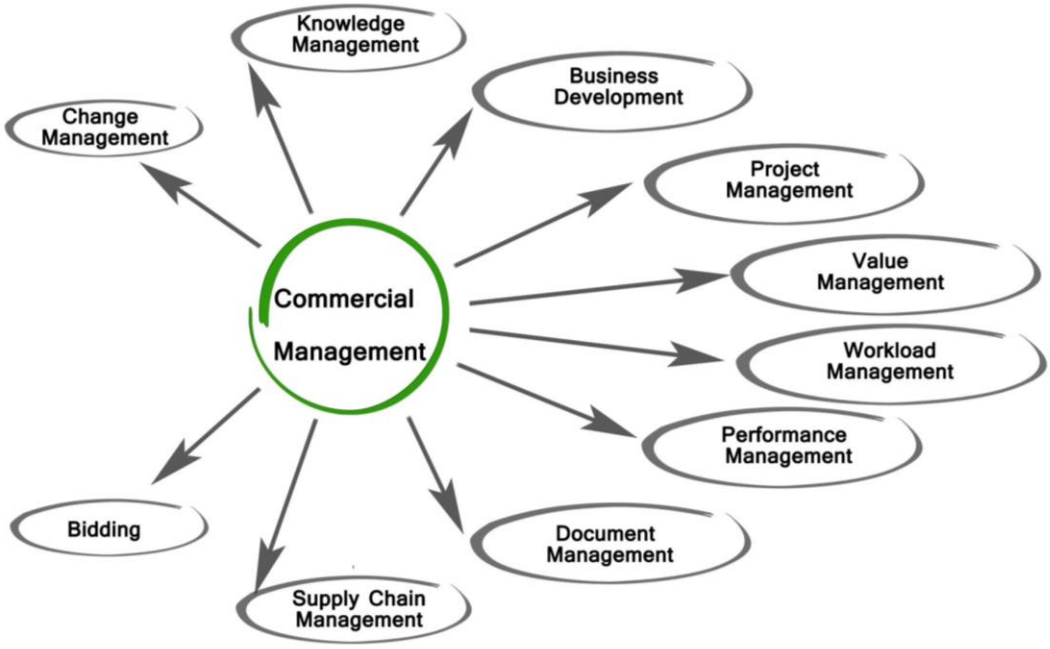
2.8 System Data Flow Diagram (DFD)

To visualize the interaction between the user, the application, and the database, a Data Flow Diagram (DFD) is used. This diagram illustrates the logical flow of information through the system.

Level 0 DFD (Context Diagram)

The Level 0 DFD represents the entire Book Management System as a single process and its interaction with the single external entity, the User.

- External Entity: User
- Process: Book Management System
- Input Flow: Book Data (Title, Price, Quantity), CRUD Requests (Search, Edit, Delete command)
- **Output Flow: Inventory List, Operation Status (Success/Error Message)**



2.9 Implementation Milestones and Development Environment

The project was executed following an iterative, agile-like approach, breaking down the required functionality into distinct, testable modules.

2.9.1 Development Environment

- IDE: Microsoft Visual Studio (Community Edition) - Used for coding, debugging, and managing the solution structure.
- Database Management: SQL Server Management Studio (SSMS) - Used to inspect the database schema, run direct queries, and verify data integrity outside the application.
- Version Control: Git with a private repository (e.g., GitHub or Azure DevOps) - Used for source code control, tracking changes, and managing project history.

2.9.2 Project Milestones

The development was segmented into the following key phases:

Milestone	Description	Technology Focus	Outcome
Setup	Project solution and initial MVC structure creation. Integration of necessary NuGet packages (EF Core, SQL Server provider).	ASP.NET Core, C#, NuGet	Base project structure ready for component development.
DAL Implementation	Defining the Book model and **DbContext**. Running EF Core migrations to create the SQL Server database table.	EF Core Migrations, SQL Server	A persistent Book table in the database.
Create (C)	Developing the **Create Book** View and implementing the Controller action to handle the HTTP POST and data insertion into the database.	Razor Views, C#, EF Core .Add() and .SaveChanges()	First functional feature: adding new books.
Read (R)	Developing the **View Books** list page and implementing the Controller action to fetch all records using LINQ.	Razor Views, C#, LINQ, EF Core .ToList()	Display of book inventory.
Update/Delete (U/D)	Implementing the Edit/Delete views and	Razor Views, C#, EF Core state	Full CRUD functionality

Milestone	Description	Technology Focus	Outcome
	controller logic to retrieve, modify, and remove existing records.	tracking and .Remove()	achieved.
UI Polish	Applying the Bootstrap framework for styling and ensuring the interface is user-friendly and responsive.	HTML, CSS, Bootstrap	Final aesthetic and usability check.

2.10 Future Scope and Limitations

While the project successfully achieved the core goals of a Mini Project, certain limitations were recognized, pointing towards opportunities for future development in a full-scale application.

2.10.1 Identified Limitations

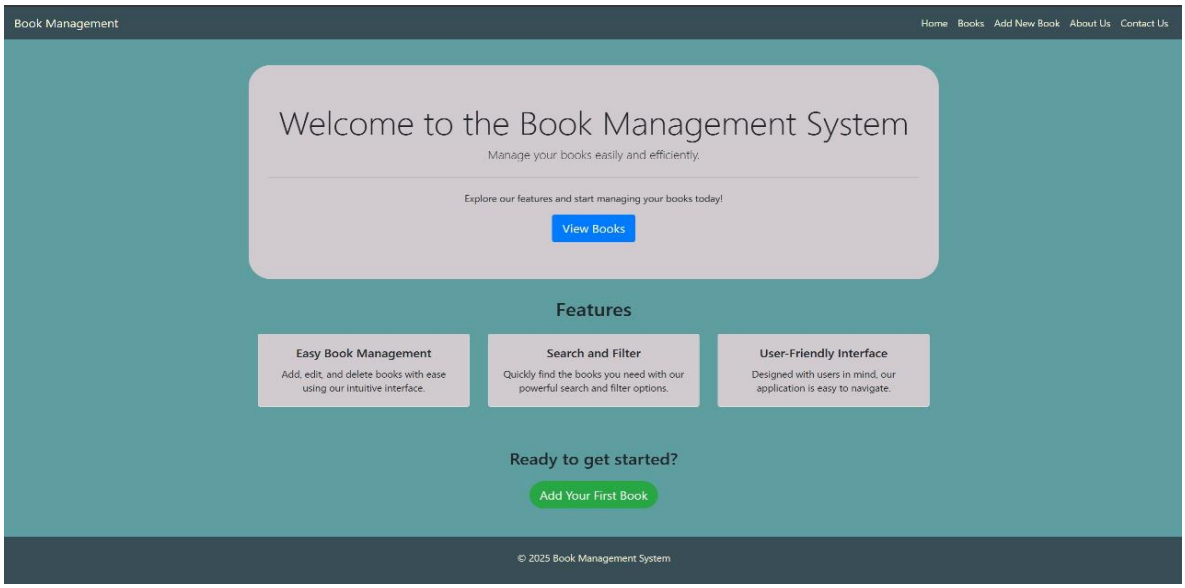
- Security (Authentication/Authorization): The current version is a single-user system and does not implement a formal login system, user roles, or granular authorization. Any user with access can perform all CRUD operations.
- Reporting: The system currently only displays a list of books. It lacks advanced reporting features, such as calculating total inventory value, low-stock alerts, or sales history.
- Asynchronous Operations: For improved performance, particularly when dealing with large datasets or slower database connections, the application should transition to using asynchronous data access methods (e.g., AddAsync, ToListAsync).

2.10.2 Future Enhancements

Based on the current foundation, the project is highly extensible, allowing for the following features to be added:

- User Identity: Integration of ASP.NET Core Identity for secure user registration, login, and role-based access control.
- Advanced Searching/Filtering: Implementation of richer filter criteria (e.g., filtering by Author, Publication Year, or Price Range) to enhance the Search and Filter feature.
- API Exposure: Refactoring the application to expose a RESTful API, allowing the book data to be consumed by other applications, such as mobile clients or external services.
- Refined Data Model: Expanding the Book model to include additional attributes like Author, ISBN, and PublicationYear to make the system more comprehensive.

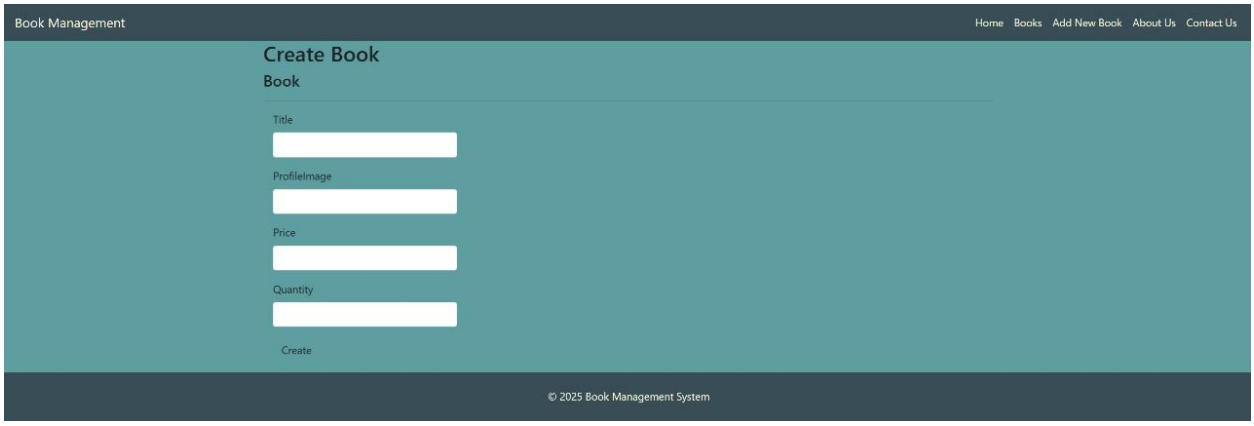
3.Implementation Snapshot



3.1 Welcome/Home Page

The landing page serves as an introduction and guide to the system's core capabilities:

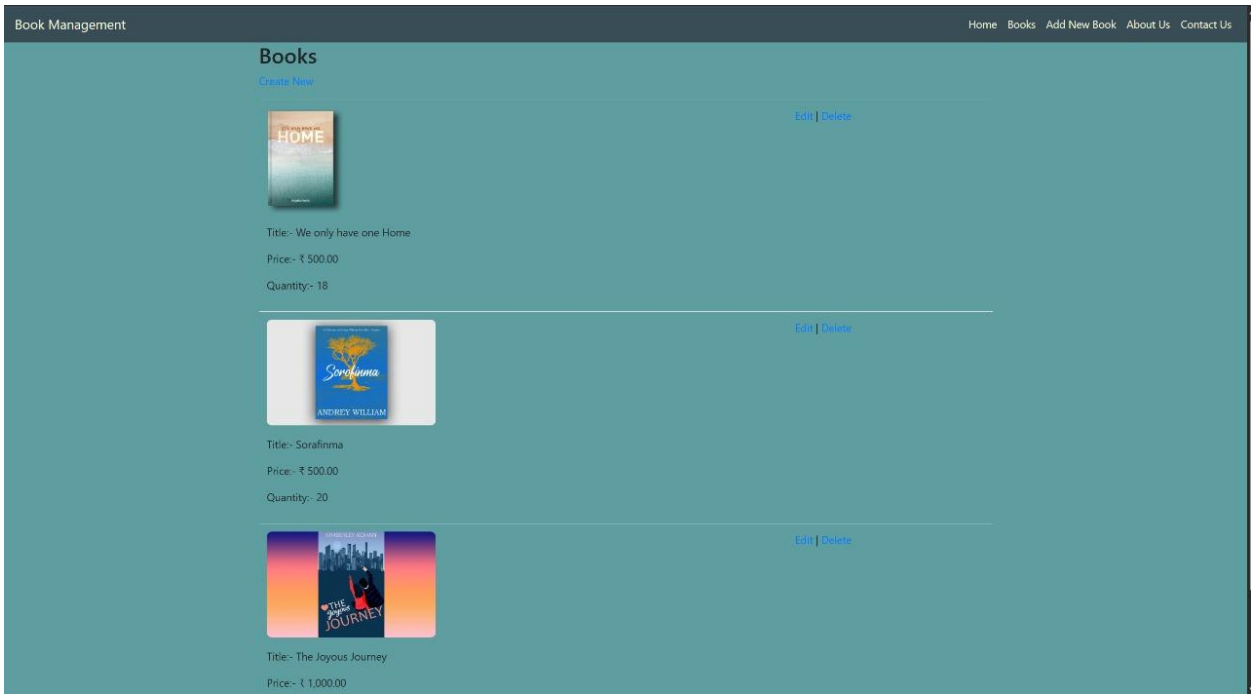
- **Feature Highlight:** Easy Book Management, Search and Filter, and User-Friendly Interface.
- **Call to Action:** "View Books" and "Add Your First Book."



3.2 Create Book Page

This view allows the user to input the details of a new book record:

- **Input Fields:** Title (e.g., 'FF'), Price (e.g., '22'), and Quantity (e.g., '33').
- **Action:** Clicking 'Create' executes the backend logic to save the new entity.



3.3 Book Inventory View (Read Operation)

This page is the central hub for the inventory, showcasing the complete functionality of the **Read (R)** operation, and the entry points for **Update (U)** and **Delete (D)**. This view is rendered dynamically by the Razor View engine using data fetched from the database via the Data Access Layer.

- **URL/Context:** The page is titled "Books" and includes a link to "Create New" (Add New Book).
- **Data Display:** Successfully fetches and displays multiple records from the SQL database, including:
    - **Title:** e.g., "We only have one Home", "Sonafirma", "The Joyous Journey"
    - **Price:** e.g., ₹ 500.00, ₹ 1,000.00 (The display shows the use of the Indian Rupee symbol, indicating successful formatting of the decimal Price field).
    - **Quantity:** e.g., 18, 20 (Shows the current stock count).
- **CRUD Access:** Each book entry is accompanied by "Edit" and "Delete" links, which initiate the Update and Delete operations, respectively, via specific controller actions.
- **Implementation Verification:** The successful rendering of three distinct book records proves that the LINQ query and EF Core data retrieval methods are functioning correctly, translating the database data into the required C# objects for presentation.

3.4 Edit/Update Snapshot (Conceptual)

Although a dedicated snapshot for the Edit page is not provided, the **Book Inventory View** confirms the mechanism for the **Update (U)** operation is in place.

- **Mechanism:** Clicking the **"Edit"** link beside a record (e.g., "Sonafirma") triggers an HTTP GET request to the controller, passing the BookID of that record.
- **Data Retrieval:** The controller uses the BookID to query the database (e.g., _context.Books.FirstOrDefault(b => b.BookID == id)), fetching all the current data.
- **View Rendering:** The data is then passed to an **Edit View**, which is functionally identical to the

Create Book Page , but the form fields are pre-populated with the fetched current values, allowing the user to modify them.

- **Submission:** Upon submission, an HTTP POST request is sent back, and the controller uses the EF Core mechanism to track changes to the entity and execute the necessary SQL UPDATE statement.

3.5 Delete Operation Snapshot (Conceptual)

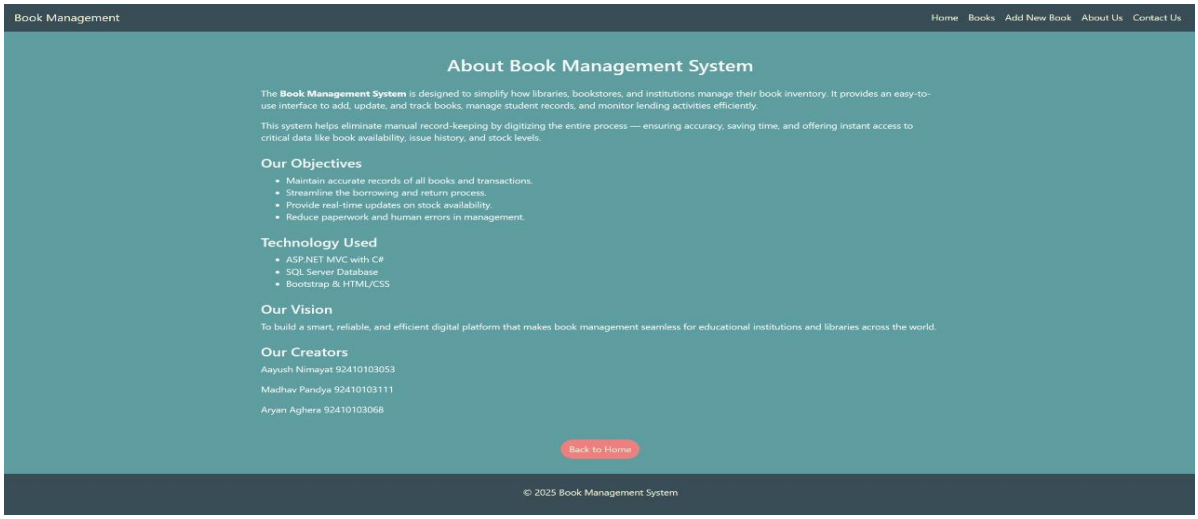
The **Delete (D)** operation is also confirmed as accessible directly from the **Book Inventory View** .

- **Mechanism:** Clicking the **"Delete"** link beside a record (e.g., "The Joyous Journey") typically triggers one of two scenarios:
    1. **Confirmation Page:** The user is redirected to a confirmation page that displays the book's details and asks for final confirmation before deletion.
    2. **Direct Deletion (Less Common):** The controller immediately executes the deletion request (less user-friendly, usually requires JavaScript confirmation).

3.6 Search and Filter Feature (Conceptual)

The **Welcome Page** explicitly lists **"Search and Filter"** as a key feature, indicating its planned implementation, which enhances the core **Read** functionality.

- **Interface Integration:** The search input box would typically be integrated above the list of books on the **Book Inventory View** .
- **Backend Logic:** This feature is implemented by enhancing the LINQ query in the Read controller action. Instead of selecting all books, the query is conditionally filtered based on user input.



3.7 About Book Management System Page

This snapshot details the **About Us** or Project Information page, which serves a documentation and informational purpose within the application. This page clearly communicates the system's mission, technical foundation, and credit to the development team.

Role and Content

The page provides critical context that supports the technical descriptions in Section 2 and the objectives laid out in Section 1.

- **System Description:** The system is designed to **simplify how libraries, bookstores, and institutions manage their book inventory**. It eliminates manual record-keeping by digitizing the entire process, offering instant access to critical data like **book availability, issue history, and stock levels**.
- **Our Objectives:** This section summarizes the project goals as integrated features of the system:
    - Maintain **accurate records of all books and transactions**.
    - **Streamline the borrowing and return process**.
    - Provide **real-time updates on stock availability**.
    - Reduce **paperwork and human errors in management**.
- **Technology Used:** This confirms the core technical stack for the project, directly validating the details provided in Section 2.1:
    - **ASP.NET MVC with C#**
    - **SQL Server Database**
    - **Bootstrap & HTML/CSS**
- **Our Vision:** States the long-term goal: **to build a smart, reliable, and efficient digital platform that makes book management seamless for educational institutions and libraries across the world**.
- **Our Creators:** Acknowledges the student development team, listing their names and enrollment numbers.

Technical Implementation

The "About Book Management System" page is a static Razor View. The key implementation points are:

1. **Navigation:** The page is linked via the "About Us" link in the application's header menu, confirming the correct routing in the **ASP.NET MVC** framework.
2. **Styling:** The layout and use of typography (headings, bullet points) adhere to the **Bootstrap** framework, ensuring visual consistency with the rest of the application (Home, Books, Create pages).
3. **Footer:** The copyright notice (© 2025 Book Management System) confirms the application's ownership and the year of development.
4. **Action Button:** The "Back to Home" button provides simple navigation, improving the user experience.

4.conclusion

The Book Management System mini-project successfully demonstrates the application of .NET Technologies for developing a practical, data-driven web application. The core objective of providing efficient book inventory management through CRUD operations and an intuitive interface was met. This project provided valuable experience in architectural design (MVC), data persistence (EF Core), and frontend integration (HTML/CSS/Bootstrap). Future enhancements could include user authentication and reporting features.

References

- Microsoft SQL Server (local or Azure).
- Microsoft Docs: Entity Framework documentation
- Tutorial/Course used for guidance
- GitHub Repository of the project