BOOK SHELF

MVC Web Application

MVC Design Project

This application is developed by using the **Model-View-Controller (MVC)** architectural pattern. The application involves managing a collection of **books**, allowing users to **add**, **view**, **update**, and **delete** books from the collection while maintaining a clear separation of concerns and modularity through the **MVC** pattern.

In addition to the core requirements, my application integrates **user authentication**, providing a secure and personalized experience for each user. Furthermore, i have extended the scope to include the ability to **add**, **view**, **update**, and **delete genres** from the collection. This feature enables efficient organization and categorization of books, enhancing the overall user experience.

In the following pages i will talk about the implementation of **Model-View-Controller (MVC)** pattern within my application. The **MVC** pattern serves as the architectural foundation for developing a robust and maintainable web application. It facilitates the clear separation of concerns and promotes modularity, which is essential for effective software development.

Throughout this documentation, I will address the responsibilities and interactions of each component - the **Model**, the **View**, and the **Controller**. By discussing the distinct roles played by these components, With using the **code snippets** i will highlight how they contribute to establishing a well-structured and organized codebase.

Database

Program.cs:

```
builder.Services.AddDbContext<DatabaseContext>(options => options.UseSqlServer(builder.Configuration.GetConnectionString("conn")));
```

Registers the **DatabaseContext** class for dependency injection, specifying the **SQL Server** connection string obtained from the application configuration.

20230617103832_20221027055012_mig.Designer.cs:

```
modelBuilder.Entity("BookStoreMvc.Models.Domain.Book", b =>
    {
        b.Property<int>("Id")
            .ValueGeneratedOnAdd()
            .HasColumnType("int");

        SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

        b.Property<string>("BookImage")
            .HasColumnType("nvarchar(max)");

        b.Property<string>("Characters")
            .IsRequired()
            .HasColumnType("nvarchar(max)");

        b.Property<string>("PurchaseYear")
            .HasColumnType("nvarchar(max)");

        b.Property<string>("Title")
            .IsRequired()
            .HasColumnType("nvarchar(max)");

        b.Property<string>("Writer")
            .IsRequired()
            .HasColumnType("nvarchar(max)");

        b.HasKey("Id");

        b.ToTable("Book");
    });
```

```
modelBuilder.Entity("BookStoreMvc.Models.Domain.Genre", b =>
    {
        b.Property<int>("Id")
            .ValueGeneratedOnAdd()
            .HasColumnType("int");

        SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

        b.Property<string>("GenreName")
            .IsRequired()
            .HasColumnType("nvarchar(max)");

        b.HasKey("Id");

        b.ToTable("Genre");
    });
```

The provided code defines the entity models and their relationships within the database schema. The code begins by configuring the **Book** entity, specifying properties such as **Id**, which is set as an automatically generated integer value using an identity column in **SQL Server**. Additional properties such as **BookImage**, **Characters**, **PurchaseYear**, **Title** and **Writer** are defined with their respective data types and constraints. The primary key for the **Book** entity is set to **Id** and the corresponding table name is specified as **Book**. Then proceeds to define the **BookGenre** entity, which represents the relationship between **Book** and **Genre**. It includes properties like **Id**, **BookId** and **GenreId** with their respective data types. The primary key for the **BookGenre** entity is set to **Id** and the table name is defined as **BookGenre**. The **Genre** entity is configured, with properties including **Id** and **GenreName** along with their data types and constraints. The primary key is assigned to **Id** and the table name is specified as **Genre**.

Model

In the model section of this documentation, I will explain the key elements that form the foundation of my application's data management. This includes the implementation of domain data transfer objects (**DTO**s) and **repositories**.

Domain data transfer objects (**DTO**s) are structured containers that hold data related to books and genres. They allow for the efficient transfer of information between different parts of the application, such as the controller and the database. Using DTOs ensures that data is handled consistently and effectively.

Repositories are responsible for managing data storage and retrieval. They act as an intermediary between the application and the underlying data storage, like a database. Repositories handle tasks such as creating, reading, updating, and deleting book and genre data. By separating the data access logic within repositories, we improve the maintainability of our application.

Book.cs

```
namespace BookStoreMvc.Models.Domain
{
    20 references
    public class Book
    {
        12 references
        public int Id { get; set; }
        [Required]
        10 references
        public string? Title { get; set; }
        9 references
        public string? PurchaseYear { get; set; }

        6 references
        public string? BookImage { get; set; }  // stores book image name with extension (eg, book0034.jpg)
        [Required]
        8 references
        public string? Characters { get; set; }
        [Required]
        10 references
        public string? Writer { get; set; }

        [NotMapped]
        8 references
        public IFormFile? ImageFile { get; set; }
        [NotMapped]
        [Required]
        9 references
        public List<int>? Genres { get; set; }
        [NotMapped]
        3 references
        public IEnumerable<SelectListItem>? GenreList { get; set; }
        [NotMapped]
        4 references
        public string? GenreNames { get; set; }

        [NotMapped]
        3 references
        public MultiSelectList? MultiGenreList { get; set; }

    }
}
```

Book.cs is representing a book entity in the book store domain. The class includes properties such as the **book's ID**, **title**, **purchase year**, **book image**, **characters**, and **writer**. Some properties have annotations to enforce certain requirements, such as the **[Required]** attribute for the title, characters, and writer properties, indicating that these fields must have values. Additionally, the class includes some **[NotMapped]** properties that are not directly mapped to the database. These properties are used for additional functionality in the application, such as the **ImageFile** property, which represents an uploaded image file for the book, and the **Genres** property, which is a list of genre IDs associated with the book. Other **[NotMapped]** properties include **GenreList**, which is a collection of genre items used for selection in the user interface, **GenreNames**, which stores the names of genres associated with the book, and **MultiGenreLis**t, which represents a multi-select list of genres.

Genre.cs

```
namespace BookStoreMvc.Models.Domain
{
    20 references
    public class Genre
    {
        6 references
        public int Id { get; set; }
        [Required]
        10 references
        public string? GenreName { get; set; }
    }
}
```

Genre.cs defines a class named **Genre** within the **BookStoreMvc.Models.Domain** namespace. This class represents a genre entity in the book store domain. The **Genre** class has two properties: **Id** and **GenreName**. The **Id** property is an integer type and serves as the unique identifier for each genre. The **GenreName** property represents the name of the genre and is marked with the **[Required]** attribute. This attribute ensures that the genre name must be provided and cannot be null or empty when creating or updating a genre entity. By using this attribute, the code enforces data validation and maintains the integrity of genre records in the application.

BookGenre.cs

```
namespace BookStoreMvc.Models.Domain
{
    3 references
    public class BookGenre
    {
        0 references
        public int Id { get; set; }
        7 references
        public int BookId { get; set; }
        6 references
        public int GenreId { get; set; }
    }
}
```

BookGenre.cs defines a class named **BookGenre** within the **BookStoreMvc.Models.Domain** namespace. This class represents the relationship between books and genres in the book store domain. The **BookGenre** class has three properties: **Id**, **BookId**, and **GenreId**. The **Id** property serves as the unique identifier for each book-genre association, while the **BookId** and **GenreId** properties represent the IDs of the book and genre associated with this entry in the BookGenre table, respectively. This class does not include any explicit data annotations, but depending on the application's (future) requirements, additional annotations or configurations could be added to establish relationships, enforce constraints, or define navigation properties between the **BookGenre** class and other entities in the domain.

DatabaseContext.cs

```
namespace BookStoreMvc.Models.Domain
{
    11 references
    public class DatabaseContext : IdentityDbContext<ApplicationUser>
    {
        0 references
        public DatabaseContext (DbContextOptions<DatabaseContext> options) : base(options)
        {

        }

        6 references
        public DbSet<Genre> Genre { get; set; }
        9 references
        public DbSet<BookGenre> BookGenre { get; set; }
        5 references
        public DbSet<Book> Book { get; set; }
    }
}
```

DatabaseContext.cs defines the **DatabaseContext** class within the **BookStoreMvc.Models.Domain** namespace. This class inherits from **IdentityDbContext<ApplicationUser>**, indicating that it represents the database context for the application with support for user authentication and authorization. The class constructor takes **DbContextOptions<DatabaseContext>** as a parameter and passes it to the base constructor to configure the context. The **DatabaseContext** class includes **DbSet** properties for **Genre**, **BookGenre**, and **Book**, representing the corresponding entities in the database. This context facilitates database operations using Entity Framework Core, including user management through **IdentityDbContext**. The code establishes a single database context for the application, integrating user authentication and authorization functionalities with other entities in the book store domain.

BookListVm.cs

```
namespace BookStoreMvc.Models.DTO
{
    9 references
    public class BookListVm
    {
        3 references
        public IQueryable<Book> BookList { get; set; }
        1 reference
        public int PageSize { get; set; }
        2 references
        public int CurrentPage { get; set; }
        2 references
        public int TotalPages { get; set; }
        2 references
        public string? Term { get; set; }
    }
}
```

The **BookListVm** class within the **BookStoreMvc.Models.DTO** namespace is a view model that represents a list of books in the **Book Store Web Application**. It contains properties as **BookList** for storing the collection of books, **PageSize** to determine the number of books per page, **CurrentPage** to track the current page being viewed, **TotalPages** to store the total number of pages based on the book count and page size, and **Term** for optional search term input. This view model facilitates the presentation and management of book data by enabling pagination, search functionality, and providing the necessary information to render and navigate through the **book list** in the **Book Store MVC Web Application**.

IBookService.cs

```
namespace BookStoreMvc.Repositories.Abstract
{
    6 references
    public interface IBookService
    {
        2 references
        bool Add(Book model);
        2 references
        bool Update(Book model);
        4 references
        Book GetById(int id);
        2 references
        bool Delete(int id);
        3 references
        BookListVm List(string term = "", bool paging = false, int currentPage = 0);
        3 references
        List<int> GetGenreByBookId(int bookId);

    }
}
```

IBookService.cs defines an interface named **IBookService** in the **BookStoreMvc.Repositories.Abstract** namespace. This interface declares methods for **adding**, **updating**, **retrieving**, and **deleting** books from the data store. It also includes a method for **retrieving genre IDs** associated with a specific book. By implementing this interface, classes can define the functionality for book-related operations, ensuring a consistent contract for interacting with book data throughout the application.

IGenreService.cs

```
namespace BookStoreMvc.Repositories.Abstract
{
    6 references
    public interface IGenreService
    {
        2 references
        bool Add(Genre model);
        2 references
        bool Update(Genre model);
        3 references
        Genre GetById(int id);
        2 references
        bool Delete(int id);
        6 references
        IQueryable<Genre> List();

    }
}
```

IGenreService.cs defines an interface named **IGenreService** in the **BookStoreMvc.Repositories.Abstract** namespace. This interface declares methods for performing various operations related to genres. The methods include **Add**, **Update**, **GetById**, **Delete** and **List**. These methods are responsible for adding a genre, updating a genre, retrieving a genre by its ID, deleting a genre, and listing all genres.

BookService.cs

```
namespace BookStoreMvc.Repositories.Implementation
{
    2 references
    public class BookService : IBookService
    {
        private readonly DatabaseContext ctx;
        0 references
        public BookService(DatabaseContext ctx)
        {
            this.ctx = ctx;
        }
        2 references
        public bool Add(Book model)
        {
            try
            {

                ctx.Book.Add(model);
                ctx.SaveChanges();
                foreach (int genreId in model.Genres)
                {
                    var bookGenre = new BookGenre
                    {
                        BookId = model.Id,
                        GenreId = genreId
                    };
                    ctx.BookGenre.Add(bookGenre);
                }
                ctx.SaveChanges();
                return true;
            }
            catch (Exception ex)
            {
                return false;
            }
        }
```

```
2 references
public bool Delete(int id)
{
    try
    {
        var data = this.GetById(id);
        if (data == null)
            return false;
        var bookGenres= ctx.BookGenre.Where(a => a.BookId == data.Id);
        foreach(var bookGenre in bookGenres)
        {
            ctx.BookGenre.Remove(bookGenre);
        }
        ctx.Book.Remove(data);
        ctx.SaveChanges();
        return true;
    }
    catch (Exception ex)
    {
        return false;
    }
}

4 references
public Book GetById(int id)
{
    return ctx.Book.Find(id);
}

3 references
public BookListVm List(string term="",bool paging=false, int currentPage=0)
{
    var data = new BookListVm();

    var list = ctx.Book.ToList();


    if (!string.IsNullOrEmpty(term))
    {
        term = term.ToLower();
        list = list.Where(a => a.Title.ToLower().StartsWith(term)).ToList();
    }

    if (paging)
    {
        // here we will apply paging
        int pageSize = 5;
        int count = list.Count;
        int TotalPages = (int)Math.Ceiling(count / (double)pageSize);
```

```
        int TotalPages = (int)Math.Ceiling(count / (double)pageSize);
        list = list.Skip((currentPage - 1)*pageSize).Take(pageSize).ToList();
        data.PageSize = pageSize;
        data.CurrentPage = currentPage;
        data.TotalPages = TotalPages;
    }

    foreach (var book in list)
    {
        var genres = (from genre in ctx.Genre
                      join mg in ctx.BookGenre
                      on genre.Id equals mg.GenreId
                      where mg.BookId == book.Id
                      select genre.GenreName
                      ).ToList();
        var genreNames = string.Join(',', genres);
        book.GenreNames = genreNames;
    }
    data.BookList = list.AsQueryable();
    return data;
}

2 references
public bool Update(Book model)
{
    try
    {
        // these genreIds are not selected by users and still present is bookGenre table corresponding to
        // this bookId. So these ids should be removed.
        var genresToDeleted = ctx.BookGenre.Where(a => a.BookId == model.Id && !model.Genres.Contains(a.GenreId)).ToList();
        foreach(var mGenre in genresToDeleted)
        {
            ctx.BookGenre.Remove(mGenre);
        }
        foreach (int genId in model.Genres)
        {
            var bookGenre = ctx.BookGenre.FirstOrDefault(a => a.BookId == model.Id && a.GenreId == genId);
            if (bookGenre == null)
            {
                bookGenre = new BookGenre { GenreId = genId, BookId = model.Id };
                ctx.BookGenre.Add(bookGenre);
            }
        }

        ctx.Book.Update(model);
        // we have to add these genre ids in bookGenre table
        ctx.SaveChanges();
        return true;
    }
```

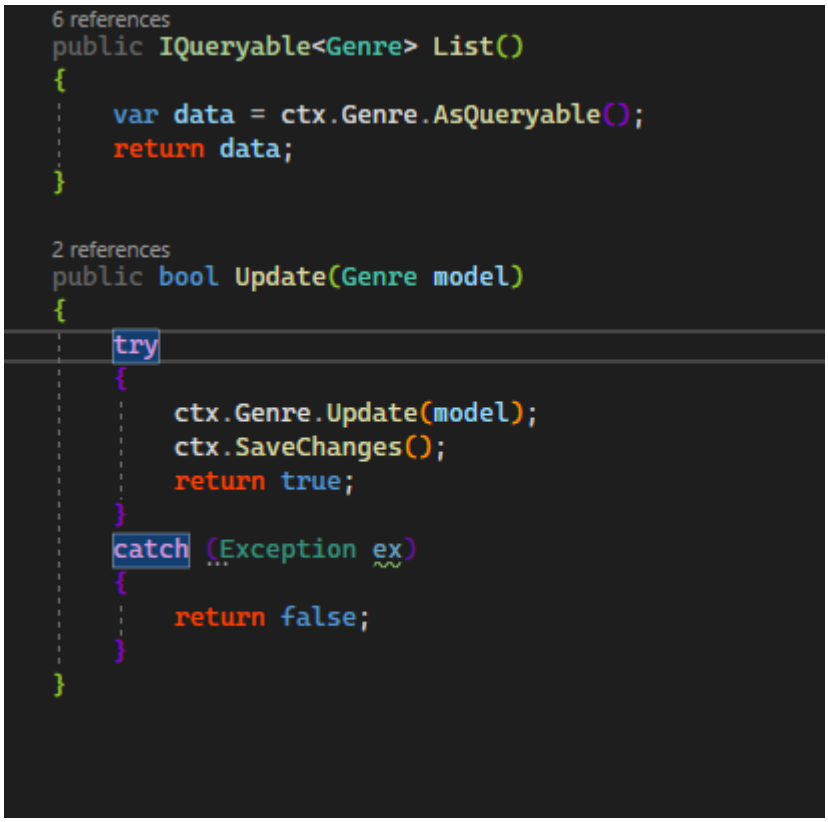
IBookService represents the implementation of the **IBookService** interface in the **BookService** class. This class is responsible for handling operations related to books in the application, such as **adding**, **deleting**, **updating** and **listing** books. It interacts with the **DatabaseContext** to perform these operations. The **Add** method adds a new book to the database, including its associated genres. The **Delete** method removes a book and its associated genre associations. The **GetById** method retrieves a book by its ID. The **List** method retrieves a list of books, optionally filtered and paginated. The **Update** method updates a book and its genre associations. The **GetGenreByBookId** method retrieves the genre IDs associated with a book.

GenreService.cs

```
2 references
public class GenreService : IGenreService
{
    private readonly DatabaseContext ctx;
    0 references
    public GenreService(DatabaseContext ctx)
    {
        this.ctx = ctx;
    }
    2 references
    public bool Add(Genre model)
    {
        try
        {
            ctx.Genre.Add(model);
            ctx.SaveChanges();
            return true;
        }
        catch (Exception ex)
        {
            return false;
        }
    }

    2 references
    public bool Delete(int id)
    {
        try
        {
            var data = this.GetById(id);
            if (data == null)
                return false;
            ctx.Genre.Remove(data);
            ctx.SaveChanges();
            return true;
        }
        catch (Exception ex)
        {
            return false;
        }
    }

    3 references
    public Genre GetById(int id)
    {
        return ctx.Genre.Find(id);
    }
```



```
6 references
public IQueryable<Genre> List()
{
    var data = ctx.Genre.AsQueryable();
    return data;
}

2 references
public bool Update(Genre model)
{
    try
    {
        ctx.Genre.Update(model);
        ctx.SaveChanges();
        return true;
    }
    catch (Exception ex)
    {
        return false;
    }
}
```

The **GenreService** class in the **BookStoreMvc.Repositories.Implementation** namespace implements the **IGenreService** interface and handles database operations related to genres. It has methods for adding, updating, deleting, and retrieving genres from the database. The constructor takes a **DatabaseContext object**, which establishes the database connection. The **Add** method adds a new genre to the database, while the **Delete** method removes a genre. The **GetById** method retrieves a genre based on its ID, and the **List** method returns all genres as an **IQueryable<Genre>**. The **Update** method updates an existing genre. The class facilitates interaction with the database for genre-related operations.

Controller

BookController.cs

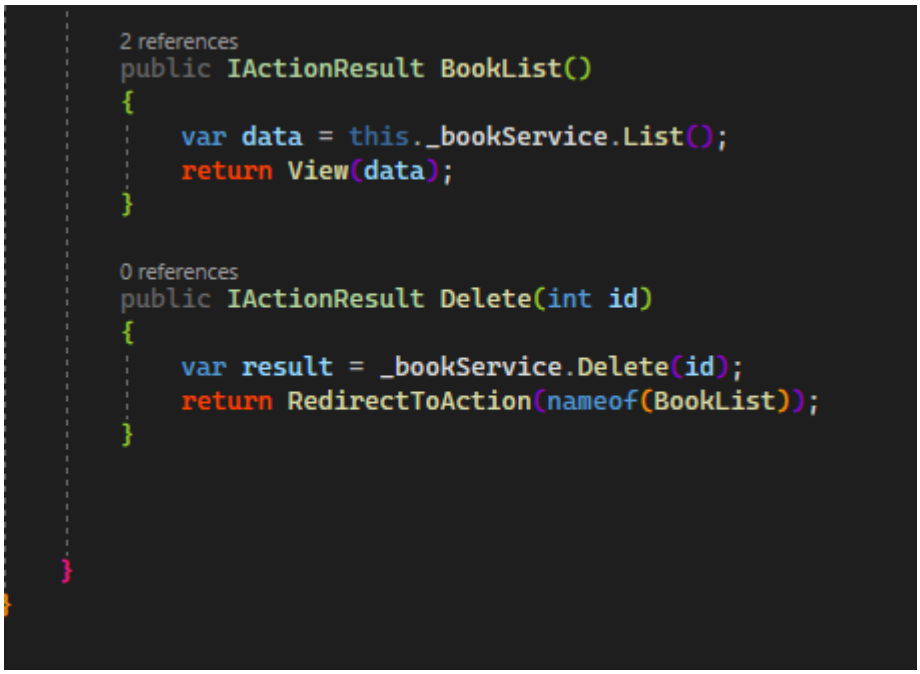
```
namespace BookStoreMvc.Controllers
{
    [Authorize]
    1 reference
    public class BookController : Controller
    {
        private readonly IBookService _bookService;
        private readonly IFileService _fileService;
        private readonly IGenreService _genService;
        0 references
        public BookController(IGenreService genService,IBookService BookService, IFileService fileService)
        {
            _bookService = BookService;
            _fileService = fileService;
            _genService = genService;
        }
        1 reference
        public IActionResult Add()
        {
            var model = new Book();
            model.GenreList = _genService.List().Select(a => new SelectListItem { Text = a.GenreName, Value = a.Id.ToString() });
            return View(model);
        }

        [HttpPost]
        1 reference
        public IActionResult Add(Book model)
        {
            model.GenreList = _genService.List().Select(a => new SelectListItem { Text = a.GenreName, Value = a.Id.ToString() });
            if (!ModelState.IsValid)
                return View(model);
            if (model.ImageFile != null)
            {
                var fileReult = this._fileService.SaveImage(model.ImageFile);
                if (fileReult.Item1 == 0)
                {
                    TempData["msg"] = "File could not saved";
                    return View(model);
                }
                var imageName = fileReult.Item2;
                model.BookImage = imageName;
            }
            var result = _bookService.Add(model);
            if (result)
            {
                TempData["msg"] = "Added Successfully";
                return RedirectToAction(nameof(Add));
            }
            else
```

```
        {
            TempData["msg"] = "Error on server side";
            return View(model);
        }
    }

    0 references
    public IActionResult Edit(int id)
    {
        var model = _bookService.GetById(id);
        var selectedGenres = _bookService.GetGenreByBookId(model.Id);
        MultiSelectList multiGenreList = new MultiSelectList(_genService.List(), "Id", "GenreName", selectedGenres);
        model.MultiGenreList = multiGenreList;
        return View(model);
    }

    [HttpPost]
    0 references
    public IActionResult Edit(Book model)
    {
        var selectedGenres = _bookService.GetGenreByBookId(model.Id);
        MultiSelectList multiGenreList = new MultiSelectList(_genService.List(), "Id", "GenreName", selectedGenres);
        model.MultiGenreList = multiGenreList;
        if (!ModelState.IsValid)
            return View(model);
        if (model.ImageFile != null)
        {
            var fileReult = this._fileService.SaveImage(model.ImageFile);
            if (fileReult.Item1 == 0)
            {
                TempData["msg"] = "File could not saved";
                return View(model);
            }
            var imageName = fileReult.Item2;
            model.BookImage = imageName;
        }
        var result = _bookService.Update(model);
        if (result)
        {
            TempData["msg"] = "Added Successfully";
            return RedirectToAction(nameof(BookList));
        }
        else
        {
            TempData["msg"] = "Error on server side";
            return View(model);
        }
    }
```



```
        2 references
        public IActionResult BookList()
        {
            var data = this._bookService.List();
            return View(data);
        }

        0 references
        public IActionResult Delete(int id)
        {
            var result = _bookService.Delete(id);
            return RedirectToAction(nameof(BookList));
        }


    }
}
```

The **BookController** class in the **BookStoreMvc.Controllers** namespace acts as the controller responsible for managing **book-related HTTP requests** in the BookStore application. It supports actions such as **adding**, **editing**, and **deleting** books, as well as **listing** books. With the **[Authorize]** attribute, the controller ensures that only **authenticated users** can access its actions. It relies on interfaces like **IGenreService**, **IBookService**, and **IFileService** for interacting with the underlying domain model and performing operations such as adding, updating, and deleting books. The controller retrieves necessary data, handles form submissions, performs validation, and renders appropriate views.

GenreController.cs

```
namespace BookStoreMvc.Controllers
{
    [Authorize]
    1 reference
    public class GenreController : Controller
    {
        private readonly IGenreService _genreService;
        0 references
        public GenreController(IGenreService genreService)
        {
            _genreService = genreService;
        }
        1 reference
        public IActionResult Add()
        {
            return View();
        }

        [HttpPost]
        1 reference
        public IActionResult Add(Genre model)
        {
            if (!ModelState.IsValid)
                return View(model);
            var result = _genreService.Add(model);
            if (result)
            {
                TempData["msg"] = "Added Successfully";
                return RedirectToAction(nameof(Add));
            }
            else
            {
                TempData["msg"] = "Error on server side";
                return View(model);
            }
        }

        0 references
        public IActionResult Edit(int id)
        {
            var data = _genreService.GetById(id);
            return View(data);
        }

        [HttpPost]
        0 references
        public IActionResult Update(Genre model)
        {
```

```
    [HttpPost]
    0 references
    public IActionResult Update(Genre model)
    {
        if (!ModelState.IsValid)
            return View(model);
        var result = _genreService.Update(model);
        if (result)
        {
            TempData["msg"] = "Added Successfully";
            return RedirectToAction(nameof(GenreList));
        }
        else
        {
            TempData["msg"] = "Error on server side";
            return View(model);
        }
    }

    2 references
    public IActionResult GenreList()
    {
        var data = this._genreService.List().ToList();
        return View(data);
    }

    0 references
    public IActionResult Delete(int id)
    {
        var result = _genreService.Delete(id);
        return RedirectToAction(nameof(GenreList));
    }


}
```

GenreController class in the **BookStoreMvc.Controllers** namespace serves as the controller responsible for handling **genre-related HTTP requests** in the BookStore application. It is decorated with the **[Authorize]** attribute to ensure that **only authenticated users** can access its actions. The controller depends on the **IGenreService** interface to interact with the underlying domain model and perform genre-related operations. The controller provides actions for **adding**, **editing** and **deleting** genres, as well as **listing** all genres. It handles form submissions, performs validation, and renders appropriate views.

HomeController.cs

```
namespace BookStoreMvc.Controllers
{
    1 reference
    public class HomeController : Controller
    {
        private readonly IBookService _bookService;
        0 references
        public HomeController(IBookService bookService)
        {
            _bookService = bookService;
        }
        0 references
        public IActionResult Index(string term="", int currentPage = 1)
        {
            var books = _bookService.List(term,true,currentPage);
            return View(books);
        }

        0 references
        public IActionResult About()
        {
            return View();
        }

        0 references
        public IActionResult BookDetail(int bookId)
        {
            var book = _bookService.GetById(bookId);
            return View(book);
        }

    }
}
```

HomeController class in the **BookStoreMvc.Controllers** namespace is responsible for managing the actions and views related to the home page and book details in the BookStore application. It relies on the **IBookService** interface to interact with the book-related functionality provided by the repositories. The **Index** action takes optional parameters for the **search term** and **current page number**, retrieves a paginated list of books based on the provided search term, and renders the corresponding view. The **BookDetail** action retrieves the details of a specific book based on the provided **bookId** and renders the corresponding view.

View

Book

Add.cshtml

```
<form class="w-40" asp-action="Add" method="post" enctype="multipart/form-data">
    <h2 class="pb-10">Add Book</h2>

    <div class="input-container">
        <label asp-for="Title" class="w-40">Title</label>
        <input type="text" class="input" asp-for="Title">
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="Title"> </span>
    </div>

    <div class="input-container">
        <label asp-for="Writer" class="w-40">Image</label>
        <input type="file" asp-for="ImageFile">
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="ImageFile"> </span>
    </div>

    <div class="input-container">
        <label asp-for="Genres" class="w-40">Genre</label>
        <select asp-items="Model.GenreList" class="input" asp-for="Genres" multiple>
            <option value="">--select--</option>
        </select>
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="Genres"> </span>
    </div>

    <div class="input-container">
        <label asp-for="PurchaseYear" class="w-40">Purchase Year</label>
        <input type="text" class="input" asp-for="PurchaseYear">
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="PurchaseYear"> </span>
    </div>

    <div class="input-container">
        <label asp-for="Characters" class="w-40">Characters</label>
        <input type="text" class="input" asp-for="Characters">
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="Characters"> </span>
    </div>
```

```
    <div class="input-container">
        <label asp-for="Writer" class="w-40">Writer</label>
        <input type="text" class="input" asp-for="Writer">
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="Writer"> </span>
    </div>

    @if (TempData["msg"] != null)
    {
        <div class="alert mb-1">
            @TempData["msg"]
        </div>
    }

    <div class="input-container">
        <button class="btn btn-default" type="submit">Save</button>
        <a class="btn btn-sec" href="/Book/BookList">All records</a>
    </div>
</form>
```

Add.cshtml represents a **Razor view file** for adding a book in the **BookStoreMvc** application. It starts with the **@model** directive, specifying the model type as **BookStoreMvc.Models.Domain.Book**. Inside the **<form>** tag, there are various input fields for capturing the book details such as **title**, **image**, **genres**, **purchase year**, **characters**, and **writer**. The form is submitted using the **post** method to the **Add** action of the corresponding controller. The view includes **error message boxes** for displaying validation errors, and if there is a **msg** value stored in TempData, it is displayed as an **alert message**. Finally, there are **buttons** for saving the book and navigating to the list of all records.

BookList.cshtml

```
<div class="tbl-container">
    <h2>Books</h2>

    <table>
        <tr>
            <th>Title</th>
            <th>Genre</th>
            <th>PurchaseYear</th>
            <th>Image</th>
            <th>Characters</th>
            <th>Writer</th>
            <th>Action</th>
        </tr>
        @foreach(var item in Model.BookList){
        <tr>
            <td>@item.Title</td>
                <td>@item.GenreNames</td>
                <td>@item.PurchaseYear</td>
            <td><img style="height:60px;width:80px" src="/Uploads/@item.BookImage"/></td>
                <td>@item.Characters</td>
                <td>@item.Writer</td>
                <td>
                <a href="/Book/Edit?id=@item.Id" class="btn btn-default"><i class="fa fa-edit"></i></a>
                <a onclick="return window.confirm('Are you sure?')" href="/Book/Delete?id=@item.Id" class="btn btn-danger"><i class="fa fa-trash"></i></a>
            </td>
        </tr>
        }

    </table>
    <div class="input-container">
        <a href="/Book/Add" class="btn btn-sec">Back</a>
    </div>
</div>
```

BookList.cshtml represents a **Razor view file** for displaying a list of books in the **BookStoreMvc** application. It starts with the **@model** directive, specifying the model type as **BookStoreMvc.Models.DTO.BookListVm**. Inside the ** ** with the class **tbl-container**, there is a table structure for displaying the book data. Each book in the **Model.BookList** collection is iterated through using a **foreach** loop. The book properties such as **title**, **genre**, **purchase year**, **image**, **characters**, and **writer** are displayed in the corresponding table cells. The image is displayed using an **<img>** tag with a **specified height and width**, and the source is set to the appropriate path. The table also includes **action buttons** for editing and deleting each book. At the end, there is an **anchor tag** for navigating back to the book addition page.

Edit.cshtml

```
<form class="w-40" asp-action="Edit" method="post" enctype="multipart/form-data">
    <h2 class="pb-10">Update Book</h2>
        <input type="hidden" class="input" asp-for="Id">
    <input type="hidden" class="input" asp-for="BookImage">

    <div class="input-container">
        <label asp-for="Title" class="w-40">Title</label>
        <input type="text" class="input" asp-for="Title">
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="Title"> </span>
    </div>

    <div class="input-container">
        <label asp-for="Writer" class="w-40">Image</label>
        <input type="file" asp-for="ImageFile">
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="ImageFile"> </span>
    </div>

    <div class="input-container">
        <label asp-for="Genres" class="w-40">Genre</label>
        <select asp-items="Model.MultiGenreList" class="input" asp-for="Genres" multiple>
            <option value="">--select--</option>
        </select>
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="Genres"> </span>
    </div>

    <div class="input-container">
        <label asp-for="PurchaseYear" class="w-40">Purchase Year</label>
        <input type="text" class="input" asp-for="PurchaseYear">
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="PurchaseYear"> </span>
    </div>

    <div class="input-container">
        <label asp-for="Characters" class="w-40">Characters</label>
        <input type="text" class="input" asp-for="Characters">
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="Characters"> </span>
    </div>
```

```
    <div class="input-container">
        <label asp-for="Writer" class="w-40">Writer</label>
        <input type="text" class="input" asp-for="Writer">
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="Writer"> </span>
    </div>

    @if (TempData["msg"] != null)
    {
        <div class="alert mb-1">
            @TempData["msg"]
        </div>
    }

    <div class="input-container">
        <button class="btn btn-default" type="submit">Save</button>
        <a class="btn btn-sec" href="/Book/BookList">All records</a>
    </div>
</form>
```

Edit.cshtml file represents a **Razor view file** for updating a book in the **BookStoreMvc** application. It starts with the **@model** directive, specifying the model type as **BookStoreMvc.Models.Domain.Book**. Inside the **<form>** tag, the form's action is set to **Edit** and the method is set to **post** with the **multipart/form-data** encoding type. There are various **input fields** for updating the book properties such as **title**, **image**, **genres**, **purchase year**, **characters**, and **writer**. The **hidden input fields** for **Id** and **BookImage** are used to store the book's ID and existing image. The form includes **validation error messages** for each input field. If there is a message in **TempData[msg]**, it is displayed in an **alert div**. The form ends with **buttons** to save the changes and navigate back to the book list page.

Genre

Add.cshtml

```
<form class="w-40" asp-action="Add" method="post">
    <h2 class="pb-10">Add Genre</h2>
    <div class="input-container">
        <label asp-for="GenreName" class="w-40">GenreName</label>
        <input type="text" class="input" asp-for="GenreName">
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="GenreName"> </span>
    </div>

    @if (TempData["msg"] != null)
    {
        <div class="alert mb-1">
            @TempData["msg"]
        </div>
    }

    <div class="input-container">
        <button class="btn btn-default" type="submit">Save</button>
        <a class="btn btn-sec" href="/Genre/GenreList">All records</a>
    </div>
</form>
```

Add.cshtml represents a **Razor view file** for adding a genre in the **BookStoreMvc** application. It starts with the **@model** directive, specifying the model type as **BookStoreMvc.Models.Domain.Genre**. Inside the **<form>** tag, the form's action is set to **Add** and the method is set to **post**. There is an **input field** for entering the genre name. The form includes a **validation error message** for the genre name input field. If there is a message in **TempData[msg]**, it is displayed in an **alert div**. The form ends with **buttons** to save the genre and navigate back to the genre list page.

Edit.cshtml

```
<form class="w-40" asp-action="Update" method="post">
    <h2 class="pb-10">Update Genre</h2>
    <input type="hidden" class="input" asp-for="Id">

    <div class="input-container">
        <label asp-for="GenreName" class="w-40">GenreName</label>
        <input type="text" class="input" asp-for="GenreName">
    </div>
    <div class="message-box pd-x-28 error">
        <span asp-validation-for="GenreName"> </span>
    </div>

    @if (TempData["msg"] != null)
    {
        <div class="alert mb-1">
            @TempData["msg"]
        </div>
    }

    <div class="input-container">
        <button class="btn btn-default" type="submit">Save</button>
        <a class="btn btn-sec" href="/Genre/GenreList">All records</a>
    </div>
</form>
```

Edit.cshtml represents a **Razor view file** for updating a genre in the **BookStoreMvc** application. It starts with the **@model** directive, specifying the model type as **BookStoreMvc.Models.Domain.Genre**. Inside the **<form>** tag, the form's action is set to **Update** and the method is set to **post**. There is **a hidden input field** for the genre ID. The form includes an **input field** for updating the genre name. The form also includes a **validation error message** for the genre name input field. If there is a message in **TempData[msg]**, it is displayed in an **alert div**. The form ends with **buttons** to save the updated genre and navigate back to the genre list page.

GenreList.cshtml

```
<div class="tbl-container">
    <h2>Genres</h2>

    <table>
        <tr>
            <th>Name</th>
            <th>Action</th>
        </tr>
        @foreach(var item in Model){
        <tr>
            <td>@item.GenreName</td>
            <td>
                <a href="/Genre/Edit?id=@item.Id" class="btn btn-default"><i class="fa fa-edit"></i></a>
                    <a onclick="return window.confirm('Are you sure?')" href="/Genre/Delete?id=@item.Id" class="btn btn-danger"><i class="fa fa-trash"></i></a>
            </td>
        </tr>
        }

    </table>
    <div class="input-container">
        <a href="/Genre/Add" class="btn btn-sec">Back</a>
    </div>
</div>
```

GenreList.cshtml represents **a Razor view file** that displays a list of genres in a **tabular format** for the BookStoreMvc application. It uses the **IEnumerable<BookStoreMvc.Models.Domain.Genre>** model to **iterate over** a collection of genres and **dynamically generate** table rows with genre names. Each genre is displayed in **a table cell**, and corresponding edit and delete **buttons** are provided for each genre entry. The edit **button** redirects to the genre edit page with the genre's ID as a query parameter, while the delete **button** prompts a confirmation dialog and **redirects** to the genre delete page upon confirmation. Additionally, there is a **link** to the genre add page for adding new genres.

Home

BookDetail.cshtml

```
<div class="book-detail w-80 m-auto">
    <div class="img-container">
        <img alt="image" src="/Uploads/@Model.BookImage">
    </div>

    <div class="detail-container">
        <h3>@Model.Title</h3>
        <h4>@Model.GenreNames</h4>
        <h4>@Model.PurchaseYear</h4>
        <h4>Characters : @Model.Characters</h4>
        <h4>Writer : @Model.Writer</h4>
    </div>
</div>
<div class="w-80 m-auto">
    <a href="/Home/Index" class="btn btn-default">Back</a>
</div>
```

BookDetail.cshtml represents a **Razor view file** that displays the details of a book in the BookStoreMvc application. It uses the **BookStoreMvc.Models.Domain.Book** model to access the properties of the book and **populate** the view. The book details, such as the **title**, **genre names**, **purchase year**, **characters**, and **writer**, are displayed within a designated container. Additionally, an **image** of the book is shown using the **BookImage** property. At the bottom of the view, there is a **Back** button that **redirects** the user to the home page when clicked. The view is structured using **CSS** classes to control the layout and styling of the elements.

Index.cshtml

```
<section class="search-bar">
    <form method="get" asp-action="Index">
        <input type="text" name="term" placeholder="search here">
        <button type="submit"><i class="fa fa-search"></i></button>
        <a class="btn btn-default" href="/Home/Index">All</a>
    </form>
</section>
<section class="books">
    @foreach(var book in Model.BookList){
        <div class="book-card" onclick="window.location.href='/Home/BookDetail?bookId=@book.Id'">
        <div class="book-image">
            <img src="/Uploads/@book.BookImage">
        </div>
        <div class="book-info">
            <h4>@book.Title</h4>
                <h4>@book.GenreNames</h4>
                <h4>@book.PurchaseYear</h4>
                @* <h4>Characters : @book.Characters</h4>
            <h4>Writer : @book.Writer</h4>*@
        </div>
    </div>
    }
</section>

<div class="pagination">
    @for(int i=1;i<=Model.TotalPages;i++){
        if(i==Model.CurrentPage){
            <a href="/Home/Index?currentPage=@i&term=@Model.Term" class="active">@i</a>
        }
        else{
            <a href="/Home/Index?currentPage=@i&term=@Model.Term">@i</a>

        }
    }

</div>
```

Index.cshtml is a Razor view file that represents the **search functionality** and **book listing** feature of the **BookStoreMvc** application. It utilizes the **BookListVm** model to **retrieve** the list of books and **display** them in a visually organized manner. The view includes a **search bar** where users can enter search terms, a list of book cards that showcase book details such as title, genre, and purchase year, and **pagination links** for navigating through the book list. Each book card is **clickable** and **redirects** to a detailed view of the selected book.

Shared

Layout.cshtml

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Book Shelf</title>
    <link rel="stylesheet" href="/css/style.css">
    <link rel="stylesheet" href="/css/forms.css">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@500;600&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel="icon" href="~/Uploads/mbr-121x93.png">
    <link href="https://fonts.googleapis.com/css2?family=Mukta:wght@500&display=swap" rel="stylesheet">
</head>
<body>
    <nav class="nav-bar">
        <div class="logo">

          BookShelf
        </div>
        <ul class="nav-link">
            <li><a class="active" href="/Home/Index">Home</a></li>

            @if (!string.IsNullOrEmpty(username))
            {
                <li>

                    <a href="#">
                        Books
                        <i class="fa fa-caret-down"></i>
                    </a>

                    <ul class="dropdown-nav-link">
                        <li><a href="/Book/Add"> Add </a></li>
                        <li><a href="/Book/BookList"> List</a></li>



                    </ul>
                </li>
            }

            @if (!string.IsNullOrEmpty(username))
            {
                <li>
```

```
            <li>

                <a href="#">
                    Genres
                    <i class="fa fa-caret-down"></i>
                </a>

                <ul class="dropdown-nav-link">

                    <li><a href="/Genre/Add">Add </a></li>
                    <li><a href="/Genre/GenreList">List</a></li>


                </ul>
            </li>
        }

        @if (!string.IsNullOrEmpty(username))
        {
            <li>
                <a href="#"> @username  <i class="fa fa-caret-down"></i></a>

                <ul class="dropdown-nav-link">

                    <li><a href="/UserAuthentication/Logout">Logout</a></li>
                </ul>
            </li>
        }

        @if (string.IsNullOrEmpty(username))
        {
            <li><a href="/UserAuthentication/Login">Login</a></li>
        }
    </ul>
</nav>
<div class="container">
    @RenderBody()
</div>
<footer>
    &#169; BookShelf
</footer>
</body>
</html>
```

Layout.cshtml is a **Razor layout** and the foundation of the **BookStoreMvc** application's user interface. It includes a **navigation bar** with various options based on the **user's authentication status**, such as **adding** and **listing** books and genres. The layout incorporates custom styling through **CSS** and **imports external fonts** for typography. The main content is **dynamically rendered** using the **RenderBody()** method, allowing individual views to populate the **central container**. The **footer** displays a standard **copyright message**.

Error.cshtml

```
<h1 class="text-danger">Error.</h1>
<h2 class="text-danger">An error occurred while processing your request.</h2>
```

Error.cshtml is a **Razor view** represents an error page. The **ViewData[Title]** is set to **Error** to specify the title of the page. The view displays an error message with a heading indicating that an error occurred while processing the request. The **text-danger** class is applied to both the **main heading** and the **subheading** to indicate that they should be displayed in **red** as an indicator of the error. This error page can be shown to users when an unexpected error occurs in the application.