BookStore API

BookStore API – ASP.NET Core Coding Assessment

Table of Contents

Table of Contents

Overview

Overview

This submission implements a RESTful BookStore API using ASP.NET Core and Entity Framework Core.

The API supports CRUD operations for both Authors and Books. It uses an in-memory database for simplicity and includes Swagger UI for interactive API documentation and testing.

This project demonstrates RESTful API design, model validation, error handling, and asynchronous calls.

Technology Stack

Technology Stack

- .NET 8.0 Framework
- C# Programming Language
- ASP.NET Core
- Entity Framework Core (In-Memory Provider)
- Swagger UI for API documentation

Setup & Running Instructions

Setup & Running Instructions

1. Restore dependencies:
   dotnet restore

2. Build the project:
   dotnet build -c Release

3. Run the API:
   dotnet run

4. (Optional) Run migrations:
   dotnet ef database update

5. Access Swagger UI at:
   http://localhost:5067/swagger

Data Models & Validations

Data Models & Validations

Author Model:

- Id: int (auto-generated)
- Name: string (required, max length 120 characters)
- Books: ICollection<Book> (navigation property)

Book Model:

- Id: int (auto-generated)
- Title: string (required, max length 200 characters)
- AuthorId: int (required, foreign key to Author)
- Author: Author (navigation property)
- PublicationYear: int (range 0-3000)

API Endpoints & Descriptions

Method	Endpoint	Description
GET	/api/authors	Retrieve all authors
GET	/api/authors/{id}	Retrieve an author by ID
POST	/api/authors	Create a new author
PUT	/api/authors/{id}	Update an existing author by ID
DELETE	/api/authors/{id}	Delete an author by ID
GET	/api/books	Retrieve all books
GET	/api/books/{id}	Retrieve a book by ID
GET	/api/authors/{authorId}/books	Retrieve books by author ID
POST	/api/books	Create a new book
PUT	/api/books/{id}	Update an existing book by ID
DELETE	/api/books/{id}	Delete a book by ID

Sample Requests & Responses

```
Author POST Request Body:
{
  "name": "J. K. Rowling"
}

Book POST Request Body:
{
  "title": "HP 1",
  "authorId": 1,
  "publicationYear": 1997
}
```

Response Codes:

- 201 Created (returns created resource)
- 200 OK (on successful GET, PUT, DELETE)
- 204 No Content (on successful update or delete)
- 400 Bad Request (validation errors)
- 404 Not Found (resource not found)

Error Handling & Status Codes

Scenario	HTTP Status Code
Successful GET/PUT/DELETE	200 / 204
Successful POST	201
Validation errors	400
Resource not found	404

Validation errors return 400 Bad Request with details.

Requests for non-existent authors or books return 404 Not Found.

Testing Guide

Use tools like Postman, Swagger UI, or Fiddler for API testing.

Recommended tests:

1. Create author — capture returned ID.
2. Create book with author ID.
3. Retrieve all authors and books.
4. Retrieve individual author/book by ID.
5. Update author/book entities.
6. Delete author/book and verify deletion.
7. Test with invalid data to check validation error handling.
8. Retrieve books by author ID.

Source Code — Program.cs

```
using BookStoreApi.Data;
using Microsoft.EntityFrameworkCore;
using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseInMemoryDatabase("BookStoreDb"));

builder.Services.AddControllers().AddJsonOptions(options =>
{
    options.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles;
});

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.MapControllers();
app.Run();
```

Source Code — AppDbContext.cs

```
using BookStoreApi.Models;
using Microsoft.EntityFrameworkCore;

namespace BookStoreApi.Data
{
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions options) : base(options) { }

        public DbSet<Author> Authors => Set<Author>();
        public DbSet<Book> Books => Set<Book>();

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<Author>()
                .HasMany(a => a.Books)
                .WithOne(b => b.Author!)
                .HasForeignKey(b => b.AuthorId)
                .OnDelete(DeleteBehavior.Cascade);
        }
    }
}
```

Source Code — Author.cs

```
using System.ComponentModel.DataAnnotations;
using System.Collections.Generic;

namespace BookStoreApi.Models
{
    public class Author
    {
        public int Id { get; set; }

        [Required]
        [StringLength(120)]
        public string Name { get; set; } = string.Empty;

        public ICollection<Book> Books { get; set; } = new List<Book>();
    }
}
```

Source Code — Book.cs

```
using System.ComponentModel.DataAnnotations;

namespace BookStoreApi.Models
{
    public class Book
    {
        public int Id { get; set; }

        [Required]
        [StringLength(200)]
        public string Title { get; set; } = string.Empty;

        [Required]
        public int AuthorId { get; set; }

        public Author? Author { get; set; }

        [Range(0, 3000)]
        public int PublicationYear { get; set; }
    }
}
```

Source Code — AuthorsController.cs

```
using BookStoreApi.Data;
using BookStoreApi.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Threading.Tasks;

namespace BookStoreApi.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class AuthorsController : ControllerBase
    {
        private readonly AppDbContext _dbContext;

        public AuthorsController(AppDbContext dbContext)
        {
            _dbContext = dbContext;
        }

        [HttpGet]
        public async Task<IActionResult> GetAuthors()
        {
            var authors = await _dbContext.Authors.AsNoTracking().ToListAsync();
            return Ok(authors);
        }

        [HttpGet("{id:int}")]
        public async Task<IActionResult> GetAuthor(int id)
        {
                                var   author   =   await   _dbContext.Authors.Include(a   =>
a.Books).FirstOrDefaultAsync(a => a.Id == id);
            if (author == null) return NotFound();
            return Ok(author);
        }

        [HttpPost]
        public async Task<IActionResult> CreateAuthor([FromBody] Author author)
        {
            if (!ModelState.IsValid) return ValidationProblem(ModelState);
            _dbContext.Authors.Add(author);
            await _dbContext.SaveChangesAsync();
            return CreatedAtAction(nameof(GetAuthor), new { id = author.Id }, author);
```

```
        }

        [HttpPut("{id:int}")]
        public async Task<IActionResult> UpdateAuthor(int id, [FromBody] Author author)
        {
            if (id != author.Id) return BadRequest();
            if (!ModelState.IsValid) return ValidationProblem(ModelState);
            if (!await _dbContext.Authors.AnyAsync(a => a.Id == id)) return NotFound();
            _dbContext.Entry(author).State = EntityState.Modified;
            await _dbContext.SaveChangesAsync();
            return NoContent();
        }

        [HttpDelete("{id:int}")]
        public async Task<IActionResult> DeleteAuthor(int id)
        {
            var author = await _dbContext.Authors.FindAsync(id);
            if (author == null) return NotFound();
            _dbContext.Authors.Remove(author);
            await _dbContext.SaveChangesAsync();
            return NoContent();
        }
    }
}
```

Source Code — BooksController.cs

```
using BookStoreApi.Data;
using BookStoreApi.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Threading.Tasks;
using System.Linq;

namespace BookStoreApi.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class BooksController : ControllerBase
    {
        private readonly AppDbContext _dbContext;

        public BooksController(AppDbContext dbContext)
        {
            _dbContext = dbContext;
        }

        [HttpGet]
        public async Task<IActionResult> GetBooks()
        {
                                var books = await _dbContext.Books.Include(b => 
b.Author).AsNoTracking().ToListAsync();
            return Ok(books);
        }

        [HttpGet("{id:int}")]
        public async Task<IActionResult> GetBook(int id)
        {
                                 var book = await _dbContext.Books.Include(b => 
b.Author).FirstOrDefaultAsync(b => b.Id == id);
            if (book == null) return NotFound();
            return Ok(book);
        }

        [HttpGet("/api/authors/{authorId:int}/books")]
        public async Task<IActionResult> GetBooksByAuthor(int authorId)
        {
                 if (!await _dbContext.Authors.AnyAsync(a => a.Id == authorId)) return 
NotFound();
```

```
                    var books = await _dbContext.Books.Where(b => b.AuthorId == authorId).Include(b => b.Author).AsNoTracking().ToListAsync();
            return Ok(books);
        }

        [HttpPost]
        public async Task<IActionResult> CreateBook([FromBody] Book book)
        {
            if (!ModelState.IsValid) return ValidationProblem(ModelState);
             if (!await _dbContext.Authors.AnyAsync(a => a.Id == book.AuthorId)) return NotFound(new { message = "Author not found" });
            _dbContext.Books.Add(book);
            await _dbContext.SaveChangesAsync();
            return CreatedAtAction(nameof(GetBook), new { id = book.Id }, book);
        }

        [HttpPut("{id:int}")]
        public async Task<IActionResult> UpdateBook(int id, [FromBody] Book book)
        {
            if (id != book.Id) return BadRequest();
            if (!ModelState.IsValid) return ValidationProblem(ModelState);
            if (!await _dbContext.Books.AnyAsync(b => b.Id == id)) return NotFound();
             if (!await _dbContext.Authors.AnyAsync(a => a.Id == book.AuthorId)) return NotFound(new { message = "Author not found" });
            _dbContext.Entry(book).State = EntityState.Modified;
            await _dbContext.SaveChangesAsync();
            return NoContent();
        }

        [HttpDelete("{id:int}")]
        public async Task<IActionResult> DeleteBook(int id)
        {
            var book = await _dbContext.Books.FindAsync(id);
            if (book == null) return NotFound();
            _dbContext.Books.Remove(book);
            await _dbContext.SaveChangesAsync();
            return NoContent();
        }
    }
}
```