# Online Book Store RESTful API Documentation

## Introduction

This is a RESTful API built in ASP.NET Core that allows users to manage books and authors. The API provides CRUD operations for books and authors and follows RESTful best practices.

## Technology Stack

• Backend: ASP.NET Core Web API  
• Database: SQL Server (can be configured for SQLite or PostgreSQL)  
• ORM: Entity Framework Core  
• Tools: Swagger, Postman, Fiddler

## API Endpoints

All API requests should be prefixed with `/api`.

### Books API

• GET `/api/books` - Retrieve all books  
• GET `/api/books/{id}` - Retrieve a book by ID  
• POST `/api/books` - Create a new book  
• PUT `/api/books/{id}` - Update a book  
• DELETE `/api/books/{id}` - Delete a book

### Example: Create a Book (POST /api/books)

Request Body:

{  
 "title": "The Great Gatsby",  
 "authorId": 1,  
 "publicationYear": 1925  
}

Response:

{  
 "id": 10,  
 "title": "The Great Gatsby",  
 "authorId": 1,  
 "publicationYear": 1925  
}

### Authors API

• 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 author  
• DELETE `/api/authors/{id}` - Delete an author

### Example: Create an Author (POST /api/authors)

Request Body:

{  
 "name": "J.K. Rowling"  
}

Response:

{  
 "id": 1,  
 "name": "J.K. Rowling"  
}

### Books by Author API

• GET `/api/authors/{authorId}/books` - Retrieve all books by an author

### Example: Get Books by Author (GET /api/authors/1/books)

Response:

[  
 {  
 "id": 1,  
 "title": "Harry Potter and the Sorcerer's Stone",  
 "authorId": 1,  
 "publicationYear": 1997  
 },  
 {  
 "id": 2,  
 "title": "Harry Potter and the Chamber of Secrets",  
 "authorId": 1,  
 "publicationYear": 1998  
 }  
]

## Database Models

### Book Model

public class Book {  
 public int Id { get; set; }  
 public string Title { get; set; }  
 public int AuthorId { get; set; }  
 public int PublicationYear { get; set; }  
 public Author Author { get; set; }  
}

### Author Model

public class Author {  
 public int Id { get; set; }  
 public string Name { get; set; }  
 public ICollection<Book> Books { get; set; }  
}

## Testing the API

### Using Postman

1. Import the API endpoints in Postman.  
2. Create sample authors first, then add books.  
3. Test all CRUD operations.

### Using Fiddler

1. Monitor API requests to check for performance bottlenecks.  
2. Debug request headers, response times, and status codes.

## Error Handling

• Invalid Book ID → 404 Not Found `{"error": "Book not found"}`  
• Invalid Author ID → 404 Not Found `{"error": "Author not found"}`  
• Missing Required Fields → 400 Bad Request `{"error": "Validation failed"}`

## Setup & Run

### Clone the Repository

```bash  
git clone https://github.com/your-repo/bookstore-api.git  
cd bookstore-api  
```

### Configure Database

1. Update appsettings.json with your database connection string:  
```json  
"ConnectionStrings": {  
 "MyCon": "Server=YOUR\_SERVER;Database=BookStore;Trusted\_Connection=True;"  
}  
```  
2. Run migrations:  
```bash  
dotnet ef migrations add InitialCreate  
dotnet ef database update  
```

### Run the Application

```bash  
dotnet run  
```  
The API will be available at `https://localhost:5001/api`.

### Test in Swagger

Visit: `https://localhost:5001/swagger` to test API endpoints.

## Future Enhancements

✅ Implement JWT Authentication for secured access  
✅ Add pagination for book listings  
✅ Implement search functionality for books by title

## Contributors

👨‍💻 \*\*Ashish Kumar Mishra\*\*

## Conclusion

This API enables CRUD operations on books and authors, following RESTful principles. Use Postman or Fiddler for debugging, and Swagger for interactive API testing.