**Requirements**

- The API can be written in either Java or C#
- The API should always return a collection of books in json format.
- By default, it should return all books without any specific sorting.
- If a specific field is searched it should return the result sorted on that field.
- It should be possible to search on any field.
- When creating a new book, use the same ID standard as the rest of the books
  - Didn't manage to make ID-increment on the format "B1", "B2". Found the attribute **[DatabaseGenerated(DatabaseGeneratedOption.Identity)]** , but didnt find a way to alter this further and follow the desired pattern.
- The API should be written in .NET > 5.0
- Whatever field I ask for, it should return the result sorted by that field.
- I should be able to ask for an author, a title, a genre, or a description. It should perform the search "case insensitive" and with partial strings. So, if I ask for "/api/books/author/kim" it should return only the book by "Ralls, Kim".
- I should be able to ask for a price range or a specific price.
- I should be able to ask for published_date or part of it, that means all books, books from a certain year, books from a certain year-month or books from a certain year-month-day.
- I should be able to edit any field for any book using the book ID as a search parameter.
- I should be able to create a new book.

Hints

- Divide the solution into what you think is architectural suitable parts.
- Refactor the code to make it as easy as possible to read and maintain.
- Use coding techniques like well-known patterns and practices and naming conventions.
- Make use of abstractions, dependency injections, extensions, expressions, attributes etc.
- Make use of different routes in the controller.
- If you are familiar with testing, we would really like to see some unit tests :)
- Don't worry if you think it seems like a tough test! Just do your best. If the requirements and use cases are covered, you are safe. The rest is only to judge the maturity of your skills.
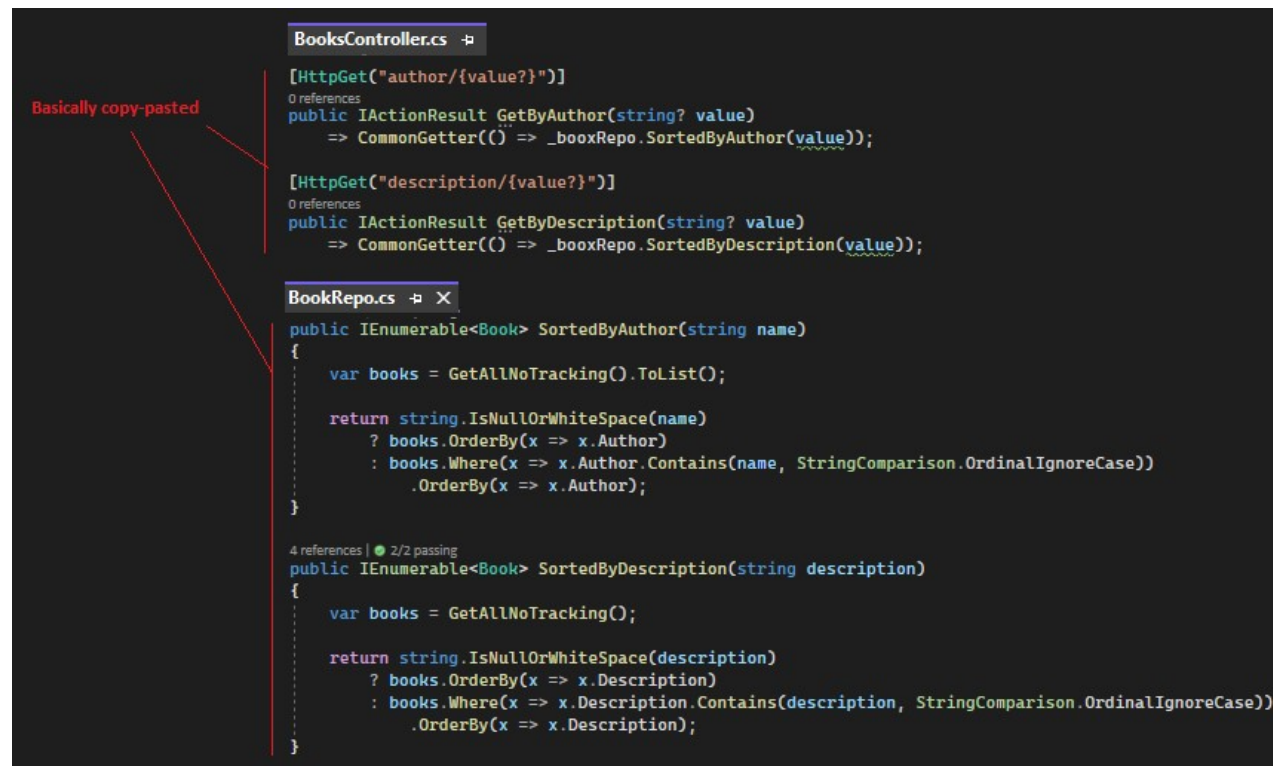
Comments

- Project Structure
  - Of course, a sa
  - *Applying CLEAN-architecture*
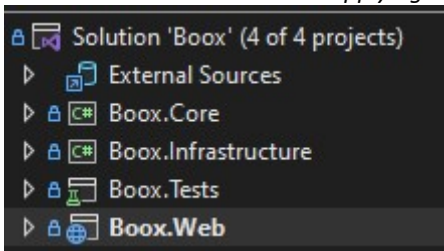
- **Core** *contains classes in common of the solution such as Entities, Dto's, Interfaces, Exceptions and domain services related to the domain.*
- **Infrastructure** *contains direct external dependencis such as Repositories and DbContext-classes. The Core-project should strive towards zero depoendency on external needs/infrastructure and the Infrastructure-library contains these dependencies instead.*
- **Tests** *contains unittests.*
- **Web** *contains of course the Api-definition with controllers, utilizing services, repos and dbcontext from Core and Infrastructure-libraries and contains*

- Many of the get-methods are basically copy-pasted (see picture below), which is indeed never is a good thing since it, or as Scott Allen (Pluralsight) puts it – copypasted code always comes back to haunt you. In order to avoid this my guess is that **Reflection** is needed, but I could of course be incorrect. I Had a look into reflection and made an attempt but didn't manage to get a generalization of these methods. I just want to point out that I'm well aware of the fallacy. It also spreads all the way down to the tests; the code becomes quite verbose too when a more general solution isn't found.


```csharp
BooksController.cs

[HttpGet("author/{value?}")]
0 references
public IActionResult GetByAuthor(string? value)
    => CommonGetter(() => _booxRepo.SortedByAuthor(value));

[HttpGet("description/{value?}")]
0 references
public IActionResult GetByDescription(string? value)
    => CommonGetter(() => _booxRepo.SortedByDescription(value));
```

```csharp
BookRepo.cs

public IEnumerable<Book> SortedByAuthor(string name)
{
    var books = GetAllNoTracking().ToList();

    return string.IsNullOrWhiteSpace(name)
        ? books.OrderBy(x => x.Author)
        : books.Where(x => x.Author.Contains(name, StringComparison.OrdinalIgnoreCase))
            .OrderBy(x => x.Author);
}

4 references | 2/2 passing
public IEnumerable<Book> SortedByDescription(string description)
{
    var books = GetAllNoTracking();

    return string.IsNullOrWhiteSpace(description)
        ? books.OrderBy(x => x.Description)
        : books.Where(x => x.Description.Contains(description, StringComparison.OrdinalIgnoreCase))
            .OrderBy(x => x.Description);
}
```

Basically copy-pasted