Code Test

General requirements Books API:

• 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

C# requirements:

• The API should be written in .NET > 5.0
• Feel free to use the API template in Visual Studio 2022

Java requirements:

• The API should be written in Java > 8.
• Utilizing Spring-boot (JavaSE) or Jakarta (Java EE).
• Utilize Maven for building the source-code

Use cases:

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.

Use case examples:

GET https://host:port/api/books returns all unsorted (B1-B13)

GET https://host:port/api/books/id returns all sorted by id (B1-B13)

GET https://host:port/api/books/id/b returns all with id containing 'b' sorted by id (B1-B13) GET

https://host:port/api/books/id/b1 returns all with id containing 'b1' sorted by id (B1, B10-13)

GET https://host:port/api/books/author returns all sorted by author (B1-B13)

GET https://host:port/api/books/author/joe returns all with author containing 'joe' sorted by author (B1)

GET https://host:port/api/books/author/kut returns all with author containing 'kut' sorted by author (B1)

GET https://host:port/api/books/title returns all sorted by title (B1-B13)

GET https://host:port/api/books/title/deploy returns all with title containing 'deploy' sorted by title (B1)

GET https://host:port/api/books/title/jruby returns all with title containing 'jruby' sorted by title (B1)

GET https://host:port/api/books/genre returns all sorted by genre (B1-B13)

GET https://host:port/api/books/genre/com returns all with genre containing 'com' sorted by genre (B1, B10-13)

GET https://host:port/api/books/genre/ter returns all with genre containing 'ter' sorted by genre (B1, B10-13)

GET https://host:port/api/books/price returns all sorted by price (B1-B13)

GET https://host:port/api/books/price/33.0 returns all with price '33.0' (B1)

GET https://host:port/api/books/price/30.0/35.0 returns all with price between '30.0' and '35.0' sorted by price (B1, B11)

GET https://host:port/api/books/published returns all sorted by published_date (B1-B13)

GET https://host:port/api/books/published/2012 returns all from '2012' sorted by published_date (B13, B1)

GET https://host:port/api/books/published/2012/8 returns all from '2012-08' sorted by published_date (B1)

GET https://host:port/api/books/published/2012/8/15 returns all from '2012-08-15' sorted by published_date (B1)

GET https://host:port/api/books/description returns all sorted by description (B1-B13)

GET https://host:port/api/books/description/deploy returns all with description containing 'deploy' sorted by description (B1, B13)

GET https://host:port/api/books/description/applications returns all with description containing 'applications' sorted by description (B1)

POST https://host:port/api/books/{id} edits an existing book

Payload

```
    {
        "author": "TestLastname, TestFirstName",
        "title": "Test Book",
        "genre": "Test genre",
        "price": "38.95",
        "written": "2008-06-01",
        "description": "Test description"
    }
```

PUT https://host:port/api/books Creates a new entry. ID is generated in backend

Payload

```
    {
        "author": "TestLastname, TestFirstName",
        "title": "Test Book",
        "genre": "Test genre",
        "price": "38.95",
        "written": "2008-06-01",
        "description": "Test description"
    }
```

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.

Deliverables:

First! Make sure everything works before handover!

The code should be documented, add comments where you feel the need, try to explain how you were thinking.

Add a brief description of how you were thinking when you created the solution or any additional information how to make it compile/execute if needed.

Once you have finished your solution make sure the code can run, you don't have to deliver the binaries, we are only interested in all the source code.

Make sure everything needed is included in the solution folder and upload it to github and send over the URL. Alternatively, zip and ship it.