Automation QA Task: Implement API tests

Introduction

The purpose of this task is primarily to examine your coding and analytical skills. The code must be written and formatted well, using best practices.

Frameworks & tools

- The tests must be written in C# using Visual Studio
- The tests must be implemented with Specflow

The Problem

You have a Library Manager API with several endpoints. Your task is to implement tests for all the endpoints of the service given to you. There are no strict rules what and how to automate. The design and approach are up to you. Any bugs found must be reported accordingly.

The Solution

When ready please send your test solution in ZIP format, excluding binaries, executables, nuget packages. The project should not have any build errors and the tests should be runnable.

Starting the web service

- Start the service by running the LibraryManager.exe.
- The URL for the service is http://localhost:9000
    - **Note**: In case this port doesn't work for you you can change it in the .config file
- All requests and responses should be in JSON format.

Library Manager API Documentation

1. Get all books:

```
GET /api/books?title={title}
```

```
Query string parameters:

    -   title - string, optional - Returns books that contain the input in their title
```

Example

```
Request:

GET /api/books?title=”Test”

Response:

200 OK

[
    {
        "Id": 1,
        "Title": "TestTitle1",
        "Description": "TestDescription1",
        "Author": "TestAuthor1"
    },
    {
        "Id": 2,
        "Title": "TestTitle2",
        "Description": "TestDescription2",
        "Author": "TestAuthor2"
    },
    {
        "Id": 3,
        "Title": "TestTitle3",
        "Description": "TestDescription3",
        "Author": "TestAuthor3"
    }
]
```

2. Get a book:

```
GET /api/books/{id}
```

Example

```
Request:

GET /api/books/6

Response:

200 OK
{
       "Id": 6,
       "Author": "TestAuthor",
       "Title": "TestTitle",
       "Description": "TestDescription"
}
```

3. Add a book:

`POST /api/books`

Body:

- Id - positive integer
- Author - string, not longer than 30 chars
- Title - string, not longer than 100 chars
- Description - string, optional

Example

Request:

```
POST /api/books

{

        "Id": 6,
        "Author": "TestAuthor",
        "Title": "TestTitle",
        "Description": "TestDescription"
}
```

Response:

```
200 OK
{
        "Id": 6,
        "Author": "TestAuthor",
        "Title": "TestTitle",
        "Description": "TestDescription"
}
```

4. Update a book:

`PUT /api/books/{id}`

Body:

- Id - positive number
- Author - string, not longer than 30 chars
- Title - string, not longer than 100 chars
- Description - string, optional

Example

Request:

```
PUT /api/books/6
{

        "Id": 6,
        "Author": "TestAuthor",
        "Title": "TestTitle",
        "Description": "TestDescription"
}
```

```
Response:

200 OK
{
	"Id": 6,
	"Author": "TestAuthor",
	"Title": "TestTitle",
	"Description": "TestDescription"
}
```

5. Delete a book:

```
DELETE /api/books/{id}
```

Example

```
Request:

DELETE /api/books/6

Response:

204 No Content
```

Error responses

All error responses have the following format:

```
{
	"Message": "Book with id 1 not found!"
}
```