Software Engineer Challenge

Introduction

Here at MoroTech we are always looking for ways to expand our general knowledge. A way to do this is by reading books! There is a problem though, it's very difficult to know whether a book is good or not before reading it. So, we decided to create a book rating system! As part of this exercise you will create a simple API that will help us rate and review the books we have read. You will be able to use your technologies of choice to achieve this, unless otherwise explicitly stated.

Implementation

The goal of the exercise is to create a web API that will allow consumers to search for books, get details for a specific book and review a book. To achieve this, you will be using a 3rd-party api for book details while the rating system will be implemented from scratch.

Part 1: Searching for books

Gutendex (https://gutendex.com/) is a free web API that provides book details from the collection of Project Gutenberg books. The API itself is very simple and it does not require any authentication or complex requests. In the link of the API there are some instructions and examples for you to use.
For this part, you will need to create an endpoint that will require a search term (a book title) and it will return a response similar to the following based on the result of Gutendex API:

```
{
   "books":[
      {
         "id":84,
         "title":"Frankenstein; Or, The Modern Prometheus",
         "authors":[
            {
               "name":"Shelley, Mary Wollstonecraft",
               "birth_year":1797,
               "death_year":1851
            },
           [...]
         ],
         "languages":[
            "en"
         ],
         "download_count":15779
      },
     [...[]
   ]
}
```

Note: Gutendex API returns more details for each book, but we are only interested in those showing in the screenshot.

Part 2: Reviewing a book

For this part of the exercise, you will have to create an endpoint that we can use to rate and review a specific book. The payload of the request will include the book id as it is returned from part 1, a rating (0-5) and a freetext review. For example, a sample payload may be like this:

```
{
  "bookId": 84,
  "rating": 4,
  "review": "It's been fifty years since I had read Frankenstein, and, now—after a recent second reading—I am pleased to know that the pleasures of that first reading have been revived....."
}
```

The schema of the payload can be changed according to your needs. The ratings and the reviews will be persisted in a database of your choice (though Sqlite is proposed for simplicity). Validations in the request payload are needed in order to keep the database safe and clean!

Part 3: Getting details of a specific book

In this final part, you will combine data from Gutendex and from the local database with the reviews. Given a book id, you will have to create an endpoint that returns book details, the average rating and a list of the reviews submitted for the book. An example response will be similar to the following:

```
{
    "id": 84,
    "title": "Frankenstein; Or, The Modern Prometheus",
    "authors": [
        {
            "name": "Shelley, Mary Wollstonecraft",
            "birth_year": 1797,
            "death_year": 1851
        }
    ],
    "languages": [
        "en"
    ],
    "download_count": 62852,
    "rating": 3.9,
    "reviews": [
      "It's been fifty years since I had read Frankenstein, and, now—after a recent second reading—I am pleased to know that the pleasures of that first reading have been revived....",
      "Another review",
      [...]
    ]
}
```

Technical Details and Deliverables

- You are required to use Python (3+) in order to complete the exercise, however you are free to use any framework/library you see fit.
- Delivering a high quality solution is more important than implementing all the requirements. Focus on code quality, documentation and testing.
- Use python type hints throughout the application.

Acceptance Criteria

- An API consumer can search for books given a title. (Part 1)
- An API consumer can post a review and a rating for a specific book. (Part 2)
- An API consumer can get the details and the average rating of a specific book. (Part 3)
- The services should gracefully handle API errors.
- There is **NO** need to implement authentication or user management.

Deliverables

A link to a git **repo**, or a compressed **file** containing all code necessary to run the app. Users should be able to install & run everything locally using scripts and / or containers, in which case necessary Dockerfile(s), scripts, documentation, installation instructions, mock data etc should be provided as part of the deliverable.

Alternatively, users should be able to access the API on any cloud infrastructure of your choice - in this case the source code alone will suffice, no need to worry about installation.

In addition to the code, a simple document with the thought process or any additional comments regarding the implementation can be provided.

Bonus Objectives (**Optional**)

Since this application was a hit inside the company, we decided to make it public. However, we need to tackle a lot of things before allowing anyone to use it! Here are some bonus objectives that you can choose to implement:

- Handle pagination of the search service. (Part 1)
- Implement a caching mechanism for book details. (Parts 1 & 3)
- Create a service that returns the top N books based on their average rating. (Part 2 extension)
- Create a service that given a book id, returns its average rating per month.
- Provide a high-level system diagram of how you would deploy such a service in a scalable way.