TomeRaider

CFG Degree Group Project

Spring 2023

Maryam Asaria, Mun-Wei Kan, Caitlin Oddy, Sophie Philipps, Sabine Lazuhina, Tania Alessi

Contents:

Introduction:

Welcome to TomeRaider. The report documents our experience creating a program for book readers to find their next read and for parents to get age-appropriate book recommendations for their children and keep track of their reading.

We are a group of 6 who have completed this project as part of the CFG Degree Software Specialisation.

The structure of our program will include an internal API that connects our front-end user interface to an external API and to a database where the user can store the books they have read and the ones they wish to read in the future.

Background:

Problem we're solving:

A common problem for readers is deciding which book to read next. The paradox of choice is a well-documented phenomenon, where it is expected that greater choice would lead to increased happiness, but it actually leaves individuals unable to make a decision and/or unhappy with the one they make.This program aims to combat this issue by providing more specific, and crucially limited, suggestions.

What the program should do:

We are building an application which enables the user to search for books by author, genre, or other relevant information. It can also be used to recommend a random book to the user.

The user will then have the option of adding the books they want to read to a database so they can be reminded later. Once completed these books can be moved to a database of read books, with the option to add a star rating and review. This allows the reader to track and look back on the books they have read.

Specifications and Design:

In terms of functionality the program should be able to:

- Welcome the user and ask what they would like to do first out of the following options:
    - Search for books by specifying different parameters such as lexile measure, genre etc.
    - Get a random book suggestion by only specifying a genre.

    - View the data stored in the database in either their to-read books and/or read books.

- For the search function the programme will:
    - Ask the user how many book recommendations they would like.
    - Ask them if they would like to specify a value for specific search parameters:

        - Author
        - Genre
        - Fiction/Non fiction
        - Lexile range minimum
        - Lexile range maximum

        The only ones of these they are required to give a response for is the genre.

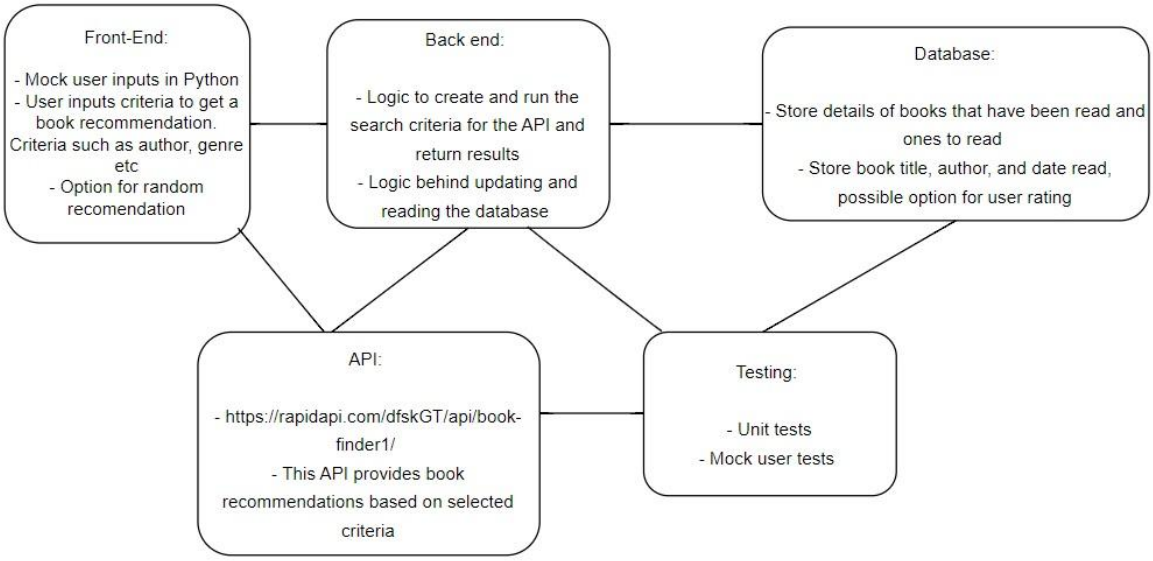
    - The programme will then ask if they would like to add any of the books to their to-read list within the database.

- For the random function the program will only ask the user to specify a genre. After the book is returned the programme will ask if the user would like to add the book to their to-read list within the database.

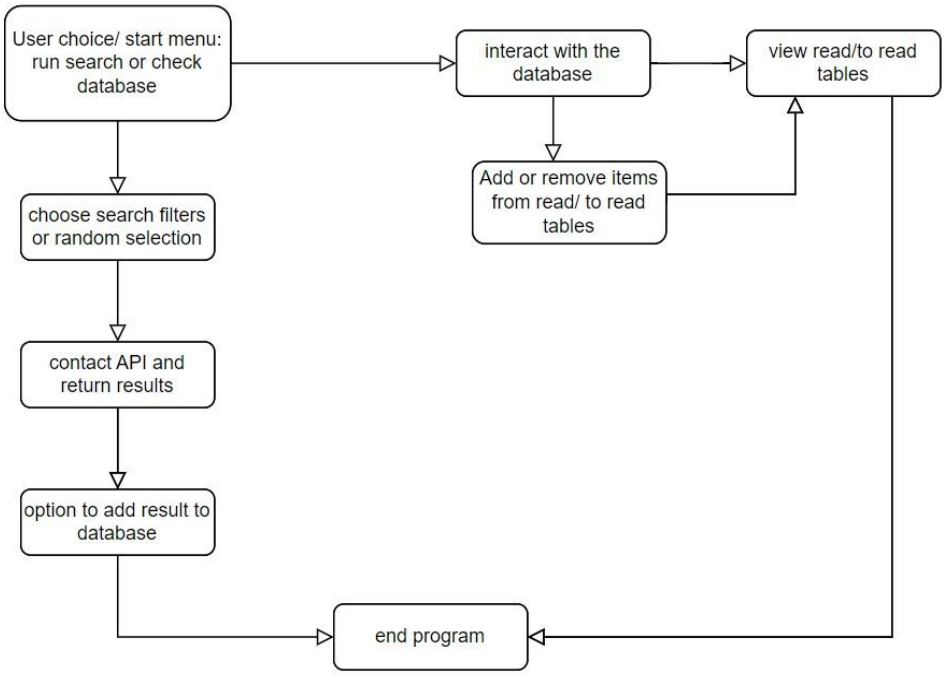
- If the user decides to look at either their to-read list or their read list the latest version will be retrieved from the MySQL database and printed to the python console.

- The to-read and read databases will have the following columns:
    - Title
    - Author
    - Category
    - Review (read books table only)
    - Star Rating (read books table only)

Overall Architecture Diagram:



Simplified Flow Diagram:



Implementation and Execution:

Development approach and team member roles:

We worked in an Agile manner, and adopted agile methodologies such as sprints and stand ups (scrums). Due to the short timescale of the project we have not implemented retrospectives after sprints but this is something we would have liked to do with more time for the project.

To enable good communication we created our own project Slack with separate channels for backend, report and general discussion. As the project progressed, we also created a Trello board as we were accumulating many small jobs and thought a Trello board would provide a clear method to record these and let others know when they were complete to avoid doubling up on work.

Our workload was divided amongst the group based on a SWOT analysis conducted at our first project meeting according to the table below. The tasks for each role were iterated over as we refined our project aims.

Roles	Person(s)	Tasks
Presentation	Caitlin Sophie	- Prepare a presentation to present the project and work behind it. - Lead the product demonstration.
Report writer	Caitlin Sophie	- Prepare a report documenting the project and the processes followed guided by the project document requirements issued by CFG.
Database Management	Tania Caitlin Sophie	- Set up the databases to be used and write code to save, update and view the entries in the database. - Responsible for the integration of SQL in Python and vice versa
Testers	Maryam Mun Wei Sabine	- Create a tests file and design tests for every function and class. - Work closely with the code reviewer to refactor the code as necessary. - Exception handling, test design, automated unit tests, mock user testing expected to be used. - Use of regression tests (end to end process test) if appropriate

Code reviewers	Caitlin Tania Sabine	- Make sure the code runs correctly and works as expected. - Ensure code is efficient and applies the most recent and/or appropriate knowledge learned through the course - Ensures appropriate class, method, and variable naming. - Refactor code as necessary to allow for better testing
Architect / Backend engineers	Maryam Sabine Mun Wei	- Design the python code that will control the program to enable the user to get recommendations. - Making sure the program is as user-friendly as possible as the program will run from the command window. - Researching APIs and implementing the best one to solve our problem. - Work together to try and make sure the code needs limited refactoring.
Project Manager	Caitlin	- Project management and ensuring delivery on time. - Completing the CFG project log. - Ensuring minutes are taken for group meetings. - Organising regular standups. - In the event of absence, appoint someone to be project manager.

<u>Tools and libraries:</u>

Alongside python and MySQL, there were a number of tools and libraries used during this project which are listed below:

- requests
- random
- itertools
- mysql.connector
- tabulate
- responses

Modules such as mysql-connector and requests were essential to enable the key components of our program to function. Whereas others, including itertools and tabulate allowed our code to be streamlined and increased overall efficiency.

Implementation process

Within the teams defined above the different functionalities were allocated to individual group members to write the specific code for each component. These sections were fed back and combined into the larger elements of the program. Separate python files were created for the different aspects of the program to help with the organisation of the project and maintain readability of the code.

As well as using an external API, basic user interface and internal database, it was decided to include an internal API to facilitate communication between the different component parts of the program. This allowed for a modular approach to constructing the program, which made distribution of the tasks more straightforward and improved overall readability. Additionally, using an internal API facilitates the later inclusion of more users and improves the overall scalability of the app. In order to maximise consistency and reduce code replication, classes were created to handle the user interactions, internal API and external API searches. Within these classes, methods were created to handle the different functions required, along with various helper and validation functions to handle exceptions and invalid inputs. During the refactoring process each method and function was streamlined to a single action.

The user interaction and internal API rely heavily on each other, and to ensure consistency across both sections of the code the team members developing these sections utilised paired programming. These sessions allowed real-time adjustments to the code and rapid problem solving when issues were encountered.

As the program is currently accessed within the IDE console it was important to consider how the data from the database would be displayed to the user so that it was clear and understandable. After exploring possible print function options and research into available python modules. It was decided to use the tabulate module to generate tables for the user. Tabulate provides a series of functions to create simple and clear tables within the python console in a variety of styles, mimicking the layout of the database tables themselves, which was ideal for our program.

Implementation challenges

One of the main challenges we encountered when implementing our code was that the external API used by our program is free and is limited to one call per second. We considered implementing artificial time delays after each API call to work around this, however, that would have slowed down the overall response rate since most searches return several pages of results and each page effectively is a new API request. To avoid this issue, it was decided to select only the first page of 100 records, which also means the number of results presented to the user is kept to a manageable size. Using the first page also means that searches that return less than 100 are always returned. These measures do limit the range of returns available to the user but increase the overall efficiency by reducing the amount of data handled internally by the program.

Another issue encountered when searching the external API was that empty dictionary values in the user input raised errors in the search functions. To counter this, a function was added in the internal API which removes empty entries before passing the data to the external API.

While writing functions to communicate with the database went smoothly, creating unit tests for those functions initially proved challenging. The functions output print statements rather than a

return, which complicated the process. However, the functions had to be refactored in order to integrate better with the wider program and during this return statements were built in, which simplified the unit testing as well.

Another implementation challenge we encountered was how to deal with specifying a reading age when searching the external API. This caused some difficulties, as although the API included max and min age as a categories, these were not searchable terms, instead lexile range was available as a search criteria. Unfortunately, the majority of users are unlikely to know what a lexile range represents and so a way to convey this information to the user was required. It was discussed whether the conversion could be implemented within the code, however, it was decided that providing a chart which illustrates the meaning of the various ranges would allow the user to best understand what they were choosing. - check understood the problem

Agile development

Once initial working versions of the key components of the code had been drafted, these were shared with the wider group, through both slack and Github, for review and comments. Short sprints were used within the different sections to meet smaller self determined deadlines prior to review within the wider group. Initial comments and obvious discrepancies were shared in comments and then regular meetings were held to discuss more complex refactoring and how best to integrate all of the component parts. This process was repeated both within the teams working on the different sections, and with the group as a whole, as the project developed.

Due to busy work and life commitments, daily stand ups weren't realistic, however, teams kept up to date using slack channels. The intention had been to implement standups every three days but this mainly led to more in depth discussions and review.

Testing and Evaluation:

Overview

When considering our testing strategy we focused on two areas: unit testing and user testing. We chose to create test files for each of our python files, refactoring our code as necessary to facilitate this. In addition to this, all members of the group conducted testing on the program to identify bugs and confirm the program was working as intended.

Once group user testing was completed and test files created we also aimed to collect some user feedback by demonstrating the program to our friends and family outside CFG. Their feedback enabled us to identify additional features we would wish to add to the program if time permitted, as well as giving general feedback about the usability of the program.

User Testing

Once we had a working prototype of our program all group members worked on user testing to identify bugs and to suggest improvements to the user experience. These were fed back to the rest of the group via a Slack channel, enabling discussion and consensus about different solutions.

For example, during this process it became apparent that simply having a search criteria called 'Category' with no further clarification was too vague and led to invalid search terms being entered. To counter this, the specific terms required by the external API were added to the user display and allocated numbers, which removed any ambiguity.

Another example would be that initially the reviews section was displaying reviews in such a way that the 'read' table was not very readable. This was pointed out and adjustments made to the code to correct this.

As a consequence of user testing we also adjusted several features of the program, for example adding the 'star rating' feature if a user did not wish to write an entire review, but would still be able to set down their thoughts. In addition, with further development users would be able to create a query from the database by star rating to find their highest rated books, should they wish to.

Unit Testing

Owing to the amount of unit testing required for this program, this work was completed by more members of the group than originally planned. Overall, unit testing proved to be an area of development for our group and this project gave us the opportunity to expand our understanding.

An important learning point for us during this project was determining which methods required testing most, given our time constraints. Following the logic within certain files we established the methods that were most likely to break the program and those relied on heavily through the file and made it a priority to test these. With additional time and as our understanding of the topic increases we would wish to return and create comprehensive tests for all methods.

System limitations

The key limitation for this program is the current lack of separate user interface, which is outside the scope of this project. The current interaction within the IDE console is functional for this preliminary stage but would require further development before wider implementation.

During testing it became apparent when using the function to delete entries from either table in the database that the MySQL query was not case sensitive and would delete all entries that matched the criteria, regardless of capitalisation. Overall, this does not pose a significant issue but does affect how the data can be manipulated during any future expansion of the program.

The limited number of results returned from the external API also restricts the eventual scope of the search function.
In the current form books cannot be moved directly from the 'to read' list to the 'read' list, which is a function that would be useful to the user and streamline the user experience.

Conclusion:

Strengths

Overall, we have achieved our aim of producing a program which allows the user to search for book recommendations either using specified search criteria or a random selection. The user is able to navigate successfully around the various functions of the program through the welcome screen and return either search results or the contents of the database. A lot of thought was put into how the user would want to navigate around the program and streamline the process. Additionally, the overall construction of the program means that expansion and scalability will be more straightforward.

Areas for expansion/ improvement

The first area to develop further would be creating a user interface using Flask, ensuring that navigating and interacting with the application becomes even more seamless and enjoyable for users. In order to deploy this program a cloud based hosting service would be needed, as well as moving the currently locally served database to a cloud based database service. Further development would include adding user accounts and authentication so that users can interact with their own book lists created in the database.

Another area for development would be adding additional functionality to the program. For example, adding the option for the users to choose to 'cancel' at any point during the program, and return to the main menu, rather than having to complete the current function before being offered the welcome menu again.

One feature in particular we would have liked to add but were constrained by time is to allow the user to move books directly from the 'to read' list to the 'read' list without having to re-enter the details. This is currently possible by adding and then deleting but to improve the user experience could be streamlined.

Currently, the program does not use the data in the database beyond simply displaying the stored entries for the user. This area could be further developed to allow the user to manipulate and analyse their reading history through functions to sort by rating or most read author for example.

Overall this project has been an excellent opportunity for us to learn and solidify knowledge gained through the CFG Degree and we have found the process very enjoyable and satisfying. Our thanks to our supervisor Arthur Kalule for his support and guidance through this project and to the entire CFG team.