MOHAMMED VI POLYTECHNIC UNIVERSITY | UM6P-CS SCHOOL OF COMPUTER SCIENCE

Recommendation system

Supervised by:

Prof ASSKLOU Abdelwahed

Prof BERRADA Ismail

Accomplished by:

KABLY Malak

LABZAE kawtar

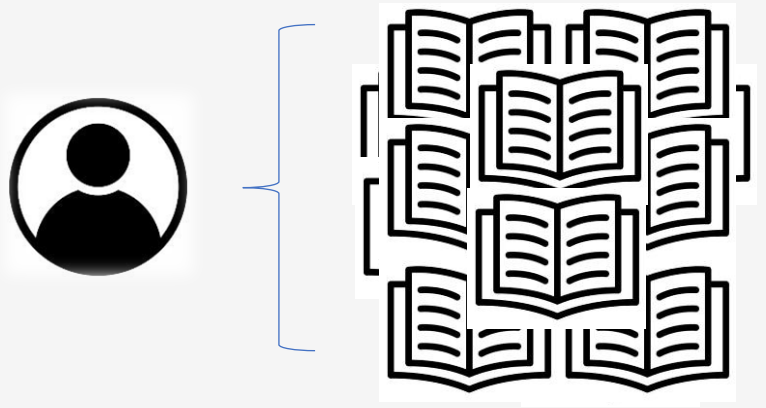
Introducing ReadQuest: Your Personalized Reading Adventure

Embark on a personalized reading adventure with ReadQuest! Unleash your curiosity, discover hidden gems, and immerse yourself in a world of personalized book recommendations. Our cutting-edge system combines content-based and collaborative filtering, analyzing authors, titles, keywords, and more. Explore a treasure trove of books, arranged from oldest to newest by publication year. Join ReadQuest for a transcendent literary odyssey like no other.

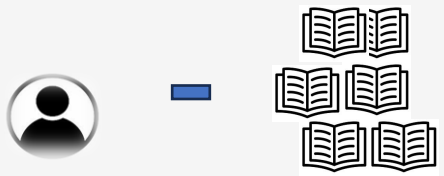
Relationship between the classes

The User class in ReadQuest empowers users to personalize their reading experience by providing a range of features and attributes

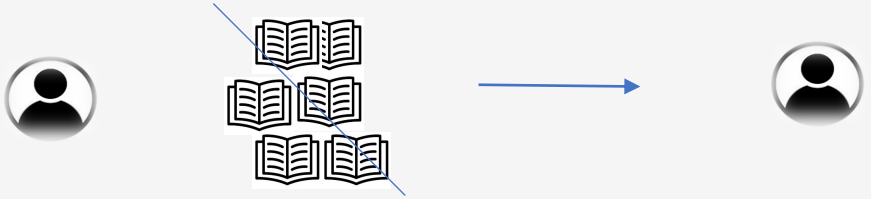
Users can have an unlimited number of books



With methods like liking and disliking books, adding and removing books from their reading list, and filtering search results, users have the flexibility to shape their literary adventure.

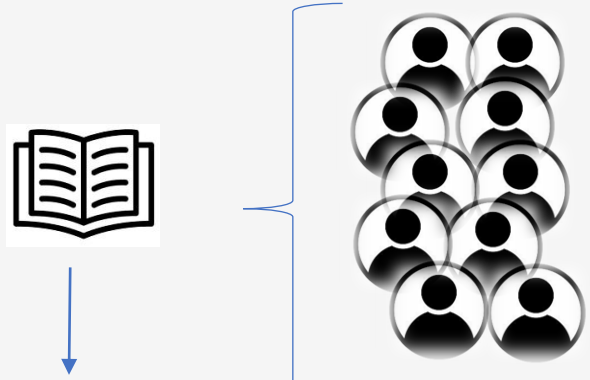


The Reset method allows users to start anew whenever desired

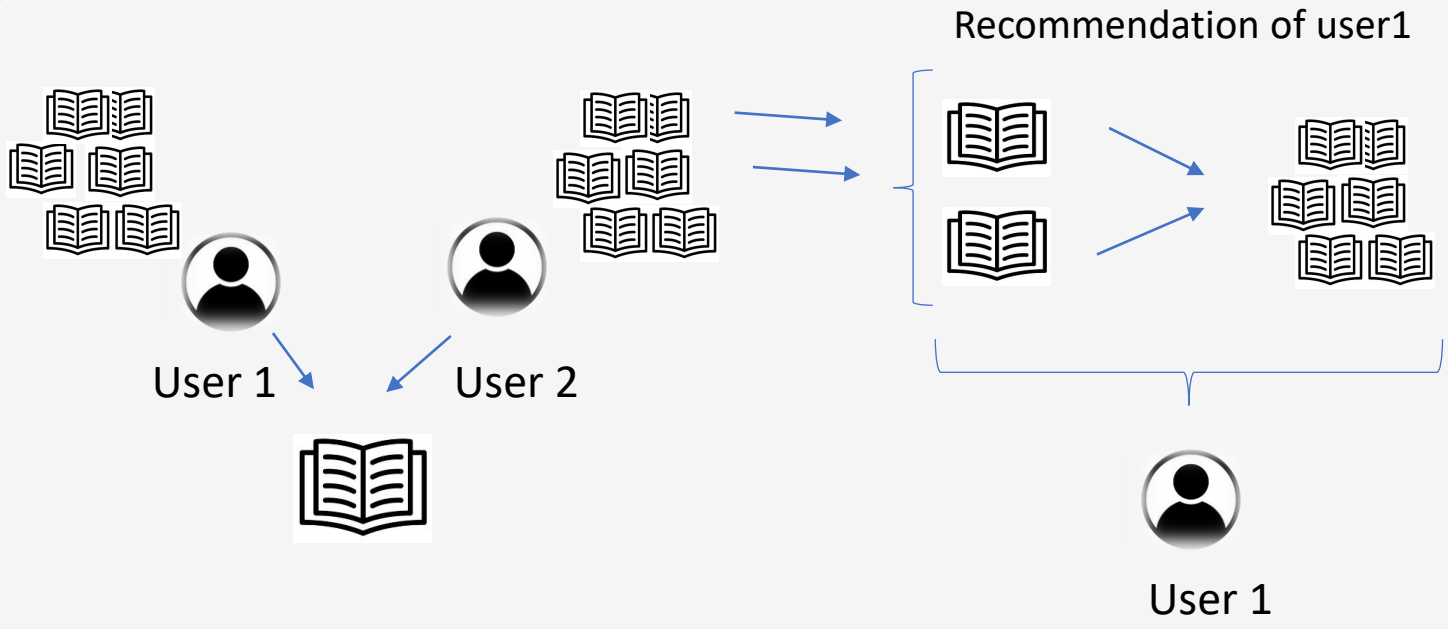


Join ReadQuest to embark on a captivating literary journey, discover hidden gems, and connect with a vibrant community of readers.

In our collaborative filtering approach, we embrace the power of community. Each book within our system can be enjoyed by multiple users who have added it to their reading lists.



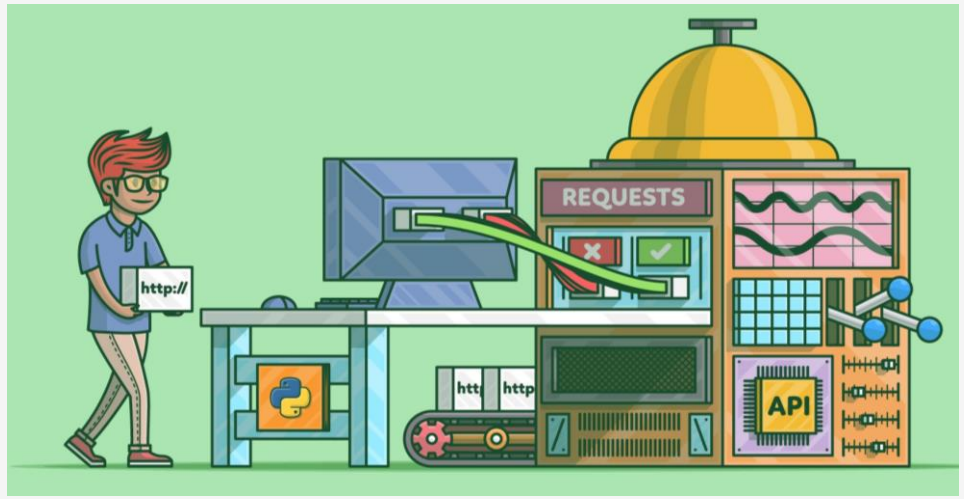
We leverage this collective wisdom to create personalized recommendations for each user. When users add the same book, we recognize their shared interest and use it as a basis for recommending other books that align with their preferences.



By tapping into the diverse tastes of our users, we ensure a rich and dynamic recommendation system that introduces readers to new and captivating literary adventures.

Part 1:

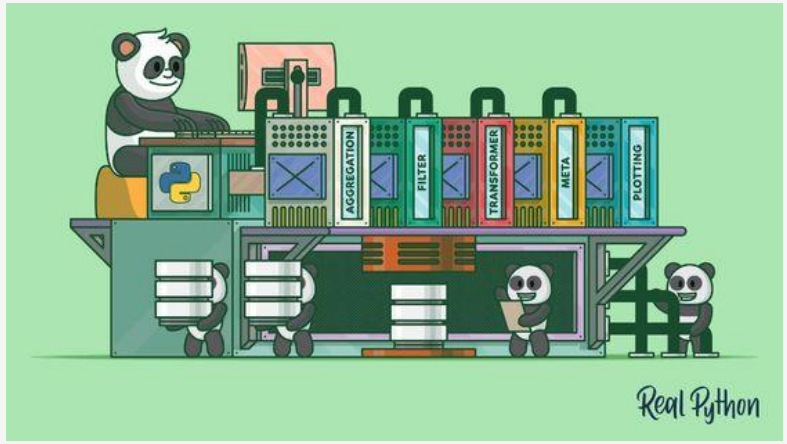
Essential Libraries: random – requests – pandas - IPython.display



The requests library in Python is a powerful tool for making HTTP requests to web servers. It simplifies the process of sending requests and handling responses, making it easier to interact with APIs and retrieve data. In the provided code, the requests library is used to communicate with the Google Books API, enabling the retrieval of book metadata based on search queries. By utilizing the requests library, the code can efficiently access external resources, retrieve information, and incorporate it into its functionality.

In our project, **the random library** plays a vital role in generating various random values. We utilize it to assign unique IDs to books, generate random prices for books within a specified range, and simulate random likes and dislikes for each book. Additionally, the random library aids in the process of recommending books based on specific genres. By leveraging its functionality, we can provide users with diverse and dynamic recommendations, enhancing their reading experience.

The pandas library



- The pandas library is used for data manipulation and analysis.
- It provides a DataFrame object, which is a tabular data structure, to store and organize the extracted book metadata.
- The DataFrame allows for efficient handling and manipulation of structured data.
- Pandas offers methods and functions for filtering, sorting, grouping, and aggregating data.
- It provides statistical analysis capabilities, allowing for data exploration and gaining insights from the extracted book metadata.
- pandas include data visualization tools to create informative plots and charts.
- The library enhances the efficiency and flexibility of working with structured data in our code.
- It enables us to manage and analyze the extracted book metadata more effectively.

The IPython.display library

- The IPython.display library is used for enhanced display and rendering of output in Jupyter notebooks or IPython environments.
- It provides functions and classes to control the display of various types of objects, such as images, HTML content, audio, video, and more.
- IPython.display offers capabilities to customize the representation and formatting of output to enhance readability and visual appeal.
- It allows for the embedding of rich media and interactive content within the notebook environment.
- The library includes functions like display() and clear_output() to control the display of output and manage the notebook interface dynamically.
- IPython.display is used in our code to facilitate the display of book information, search results, and other relevant output to the user.
- It enhances the user experience by providing a more visually appealing and interactive presentation of information.
- By leveraging the capabilities of IPython.display, our code can deliver a more engaging and inform interface for the user.

Part 2:

Data Structures:
Node, Linked List, and Stack implementation

Node:

Represents an individual node in a linked list.

Contains attributes:

- ***data***: Stores the value of the node (all types are permitted).
- ***next***: References the next node in the list.

Linked List:

Manages the structure and operations of a linked list.

Contains methods:

- ***__init__*** : Initializes the linked list with a head attribute set to None (constructor).
- ***append(data)*** : Adds a new node with the given data at the end of the list.
- ***extend(my_list)*** : Incorporates nodes from another linked list, my_list, into the current list.
- ***remove(data)*** : Deletes the first occurrence of a node with the specified data from the list.
- ***is_empty( )*** : Checks if the list is empty.
- ***remove_duplicates():*** Removes duplicate elements from the linked list.

Stack:

Implements a stack using the LinkedList class.

Contains methods:

- ***__init__*** : Initializes the stack with an underlying LinkedList instance (constructor).
- ***append(data)*** : Adds an element to the top of the stack.
- ***remove()*** : Removes and returns the topmost element from the stack.
- ***extend(my_list)*** : Adds elements from another linked list, my_list, to the top of the stack.
- ***is_empty()*** : Checks if the stack is empty.

These implementations provide the necessary components for effectively utilizing linked lists and stacks in our code. They enable efficient storage and manipulation of data, facilitating various operations and algorithms.

Part 3:

Book Class Description

Attributes:

- ***Id(int)***: A unique identifier assigned to each book instance.
- ***title(str)***: The title of the book.
- ***authors(list)***: A list of authors of the book.
- ***summary(str)***: A summary or description of the book.
- ***language(str)***: The language in which the book is written.
- ***publisher(str)***: The publisher of the book.
- ***page_count(str)***: The number of pages in the book.
- ***publication_year(str)***: The year of publication of the book.
- ***price(int)***: The price of the book, randomly generated within a range.
- ***likes(int)***: The number of likes received for the book, randomly generated.
- ***dislikes(int)***: The number of dislikes received for the book, randomly generated.
- ***readers(LinkedList)***: A linked list that stores the readers who have read the book.
- ***Feedbacks(LinkedList)***: A linked list of feedbacks that the users had added to the book.

Constructor:

The **__init__** method initializes a Book object with the provided parameters:

- title, authors, summary, language, publisher, page_count, publication_year.
- Generates a unique id for the book by checking for duplication in the list_id linked list.
- Appends the generated id to the list_id.
- Sets the remaining attributes of the book with provided values and randomly generated values.

Methods:

- ***get_id():*** Returns the unique id of the book.
- ***add_reader(user)*** : Adds a user to the list of readers who have read the book. Also updates the user's read book linked list to include the book.
- ***print_info():*** Prints detailed information about the book, including its attributes such as id, title, authors, summary, language, publisher, page count, publication year, price, likes, and dislikes.

This Book class provides a structured representation of a book and offers methods to manage readership and display book information.

Part 4:

Book Metadata Extraction, Filtering, and Sorting

insertion_sort_publication_year(l) and insert_node(l, node)

The functions `***insertion_sort_publication_year(l)***` and `***insert_node(l, node)***` work together to sort a linked list of `Book` objects based on their publication year in ascending order.

`***insertion_sort_publication_year(l)***` implements the insertion sort algorithm for sorting the linked list.

It creates a new sorted list and iterates through each node in the original linked list, inserting the nodes into the sorted list in the correct order.

‘***insert_node(l, node)***` is a helper function that inserts a node with `Book` object data into the appropriate position in a linked list sorted by publication year.

These functions contribute to the code by enabling the sorting of book metadata based on publication year, which is important for generating recommendations in chronological orde

extract_books_metadata_from_api(query, max_results=10) :

The function serves the purpose of extracting book metadata from an API based on a provided search query.

It facilitates making an HTTP GET request to the Google Books API with the given query and additional parameters.

The function handles the response from the API, checking the status code to determine if the request was successful.

If the response status code is 200, indicating a successful response, the function processes the retrieved data.

It creates a `LinkedList` object called `books` to store the extracted book metadata.

For each book item in the retrieved data, the function extracts relevant information such as title, authors, description, language, publisher, page count, and publication date.

Using this extracted information, it creates a `Book` object and appends it to the `books` linked list.

Finally, the function returns the `books` linked list containing the extracted book metadata.

In case the response status code isn't 200, indicating an error or no data found for the specific query, the function returns an empty `LinkedList`.

filter_existing_books(querry, max_results=10) :

- The ***filter_existing_books*** function is responsible for filtering existing books based on a given query and comparing them with the books in the `our_books` linked list.
- Its purpose is to avoid creating duplicate instances of `Book` objects with different attribute values.
- The function takes two parameters: `query` (the search query) and `max_results` (the maximum number of results to retrieve, the default is 20).
- It first extracts book metadata from the Google Books API by calling the `***extract_books_metadata_from_api***` function with the provided query and maximum results.
- The function then compares and updates the book attributes from the `our_books` list to ensure consistency and avoid duplicates.
- It iterates over each book in the retrieved metadata and checks if there is a matching book title in the `our_books` list.
- If a match is found, the function updates the book's metadata in the retrieved data with the corresponding data from `our_books`.
- Next, the function filters out books with missing or empty attributes by iterating over the book metadata and removing entries that have incomplete information.
- This step ensures that only books with complete and valid attributes are included in the filtered list.
- Finally, the filtered books are sorted in ascending order based on their publication year using the `***insertion_sort_publication_year***` function.
- The function returns a linked list (`book`) containing the filtered and sorted `Book` objects.

Part 5:

User Class Description

Attributes:

- ***Name(str):*** Represents the name of the user.
- ***Readbook(LinkedList)****:* Linked list to store books that the user has read.
- ***likebook(LinkedList)****:* Linked list to store books that the user has liked.
- ***dislikebook(LinkedList):*** Linked list to store books that the user has disliked.
- ***historic(Stack)***: Linked list to store the user's search history.

Constructor:

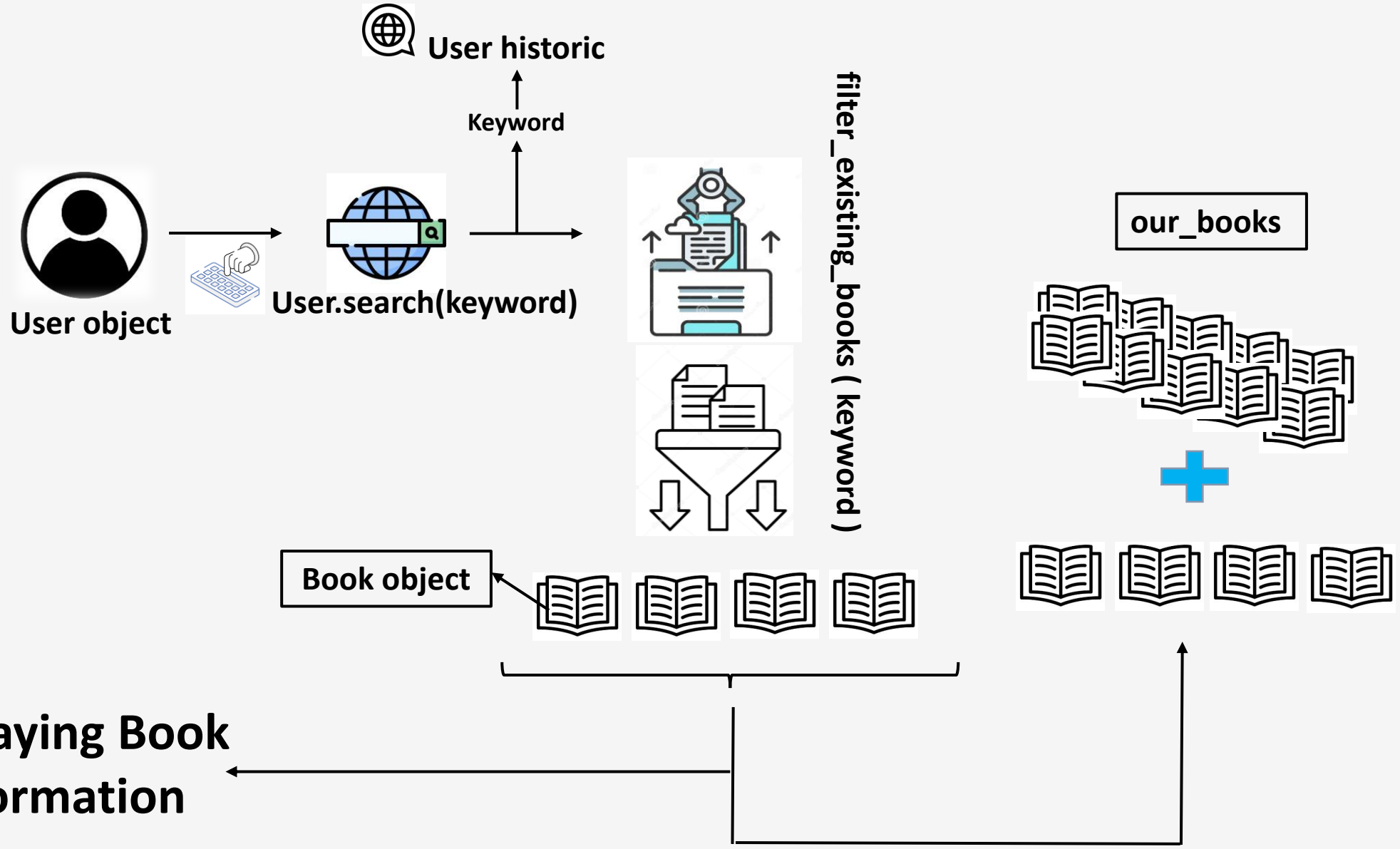
__init__(name): Initializes a user with a given name. Creates empty linked lists for **readbook**, **likebook**, **dislikebook**, and **historic**.

Methods:

- ***set_name(name)***: Sets the name of the user.
- ***like(book_id):*** Increments the like count for a book with the given ID. Appends the book to the **likebook** linked list.
- ***dislike(book_id)***: Increments the dislike count for a book with the given ID. Appends the book to the **dislikebook** linked list.
- ***add_book(book_id)***: Adds a book with the given ID to the **readbook** linked list.
- ***remove*_book(book_id)**: Removes a book with the given ID from the **readbook** linked list.
- ***add_feedback_book(id_book, feed_back):*** Adds feedback to the book with the specified id_book.
- ***reset()***: Resets the user's data by clearing all linked lists(and Stacks).
- ***guide():*** Provides an introduction to the ReadQuest system and describes each method for he user.

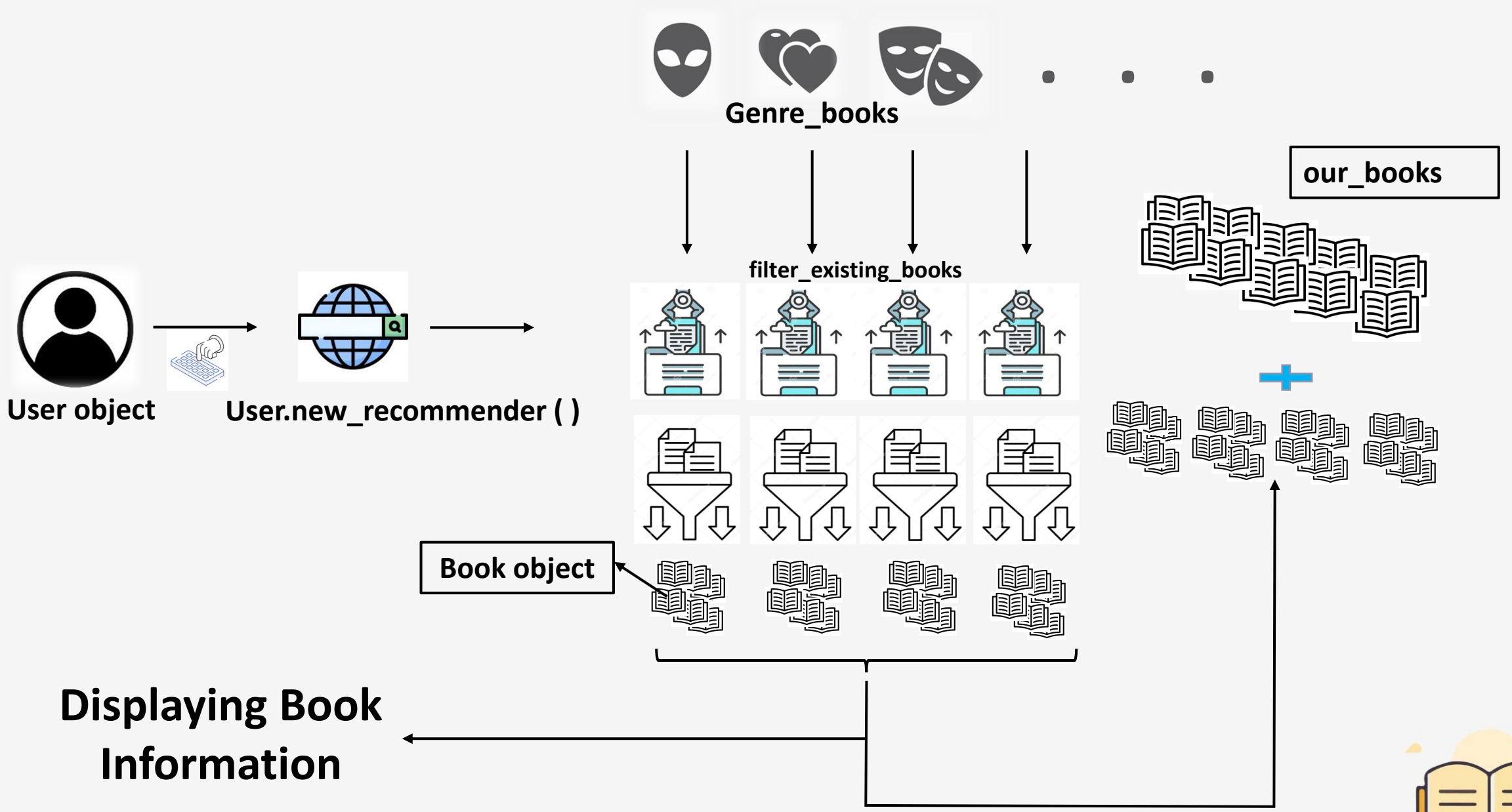
Recommendation-related Methods in the User Class

- ***search(key_word)****:* Searches for books based on the given keyword. Appends the keyword to the **historic** linked list.
- ***new_recommender()****:* Generates recommendations based on the genre of books. Prints the recommended books.
- ***enter_description(text)****:* Filters books based on the given description. Appends the description to the **historic** linked list.
- ***historical_recommender()****:* Generates recommendations based on the user's historical data. Prints the re commended books.
- ***filter_search(keyword, page_count=range(0, 10000000), price=range(80, 1001), likes=20, language="fren", pub_year=0):*** filters books based on various criteria such as page count, price, likes, language, and publication year. It removes books that do not meet the specified criteria and prints detailed information for the remaining filtered books. If no criteria are provided, default parameters are used. Additionally, it appends the keyword used for filtering to the user's search history.

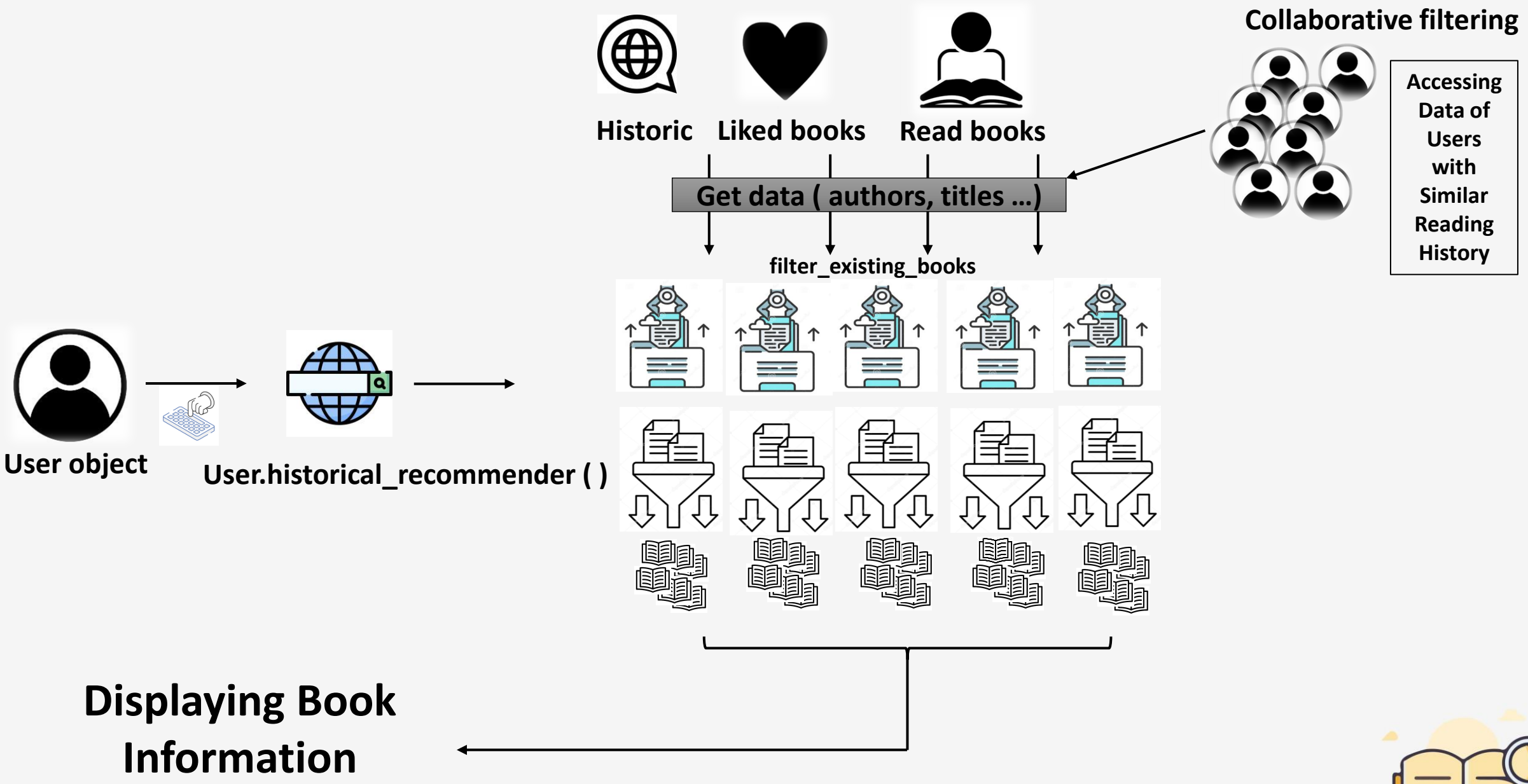


The search method

** Same goes for the enter_description method*



The new_recommender method



The historical_recommender method

Part 6:

Essential Components of Our Book Recommendation Program

Our_books:

- `***Our_books***` is a linked list that serves as a collection of book metadata.
- It stores instances of the `Book` class or objects that encapsulate information about individual books.
- `***Our_books***` is a crucial component of our book recommendation program as it allows us to manage and manipulate book data efficiently.
- With `***Our_books***`, we can add new books, remove existing books, search for specific books, and perform various operations on the book collection.
- It acts as a central repository of book information, facilitating the recommendation process and providing a source for personalized book suggestions.

Genre_books:

- `***Genre_books***` is a linked list that represents a collection of book genres and their associated subgenres.
- It maintains a mapping between each genre and its corresponding subgenres in the form of tuples.
- `***Genre_books***` is a vital component of our book recommendation program as it helps categorize and classify books based on their genres.
- By organizing books into genres and subgenres, `***Genre_books***` enables targeted recommendations based on user preferences.
- It allows users to explore books within specific genres and provides a structured framework for recommending books in a more personalized manner.
- `***Genre_books***` facilitates efficient genre-based filtering, and recommendation generation, and enhances the overall browsing experience for users interested in specific genres.

List_id:

- `***List_id***` represents a list that stores unique identifiers (IDs) of books.
- It is an essential component of our book recommendation program as it allows us to uniquely identify each book in our system.
- By maintaining a list of book IDs, we can efficiently track and retrieve specific books based on their unique identifiers.
- `***List_id***` enables various operations such as adding books, removing books, and accessing book information using their IDs.
- It serves as a fundamental data structure that forms the backbone of our book recommendation program.