Documentazione progetto Basi di Dati 2: WorldBookShelf

# Panoramica

**WorldBookShelf** è una web application progettata per gli amanti dei libri, offrendo loro una piattaforma interattiva per esplorare, recensire e organizzare la loro collezione di libri. Con oltre 20.000 titoli disponibili, WorldBookShelf consente agli utenti di lasciare recensioni utilizzando un sistema a stelle da 1 a 5 per valutare i libri presenti sulla piattaforma.

Gli utenti possono anche salvare i titoli che hanno già letto nella libreria "Read" e quelli che desiderano leggere nella libreria "Want to Read". Questa funzionalità consente agli utenti di tenere traccia dei loro progressi di lettura e di gestire facilmente la loro lista di desideri letterari.

Oltre alla gestione delle recensioni e delle librerie personali, WorldBookShelf utilizza un sistema di raccomandazione. Basandosi sui generi dei libri che l'utente ha salvato nelle librerie, la webapp suggerisce altri titoli che potrebbero interessare agli utenti. Questo approccio personalizzato aiuta gli utenti a scoprire nuovi libri che potrebbero corrispondere ai loro gusti e interessi letterari.

# Fase preliminare - dataset

Nella fase preliminare vengono utilizzati diversi script python per estrarre i dati rilevanti da vari dataset, ognuno con colonne diverse, e mapparli alle variabili della classe Book (file book.py). Questo in modo da creare un unico formato di dati che rappresenta i libri, indipendentemente dalle differenze tra i dataset originali.

Dataset utilizzati:

* <https://www.kaggle.com/datasets/bahramjannesarr/goodreads-book-datasets-10m>

Vari dataset contenenti informazioni su rating libri. Questi dataset sono stati utilizzati come base di partenza a cui sono state aggiunte altre informazioni prese dai dataset successivi.

* <https://www.kaggle.com/datasets/dk123891/books-dataset-goodreadsmay-2024>
* <https://www.kaggle.com/datasets/sahilkirpekar/goodreads10k-dataset-cleaned>
* <https://github.com/zygmuntz/goodbooks-10k>
* <https://www.kaggle.com/datasets/ishikajohari/best-books-10k-multi-genre-data>
* <https://www.kaggle.com/datasets/thedevastator/comprehensive-overview-of-52478-goodreads-best-b>

I vari script utilizzati in sequenza sono:

1. **save\_books.py**

In questo file attraverso la funzione *get\_books\_from\_csv(…)*, a cui vengono passati come parametri il percorso e i nomi delle colonne del dataset di nostro interesse, vengono letti i valori e costruito un dizionario di libri che ha come chiave l’id del libro e come valore l’oggetto Book costruito con i valori estratti dal dataset.

Viene costruito un dizionario di libri per ogni dataset utilizzato, e successivamente questi vengono serializzati in modo da avere direttamente i dati estratti per gli script successivi e non dover rileggere e rielaborare i dataset più volte.

1. **books\_to\_load.py**

In questo file al dizionario di libri relativo ai dataset scaricati dal primo link, vengono aggiunte informazioni come il genere, la descrizione e l’url della cover del libro, etc.. prese dai dizionari relativi agli altri dataset.

Da questo dizionario vengono filtrati in un altro dizionario “*books\_to\_load*” solo i libri che hanno i generi, scartando quelli che non li hanno.

Successivamente di quest’ultimi vengono eliminati anche i libri che non hanno descrizione e che non hanno editore (publisher).

Successivamente, essendo che le informazioni sulla lingua vengono prese da più dataset, con la conseguenza che ogni dataset utilizza un modo diverso per rappresentare la lingua, ad esempio un dataset utilizza “en”, un altro “English” etc.. , si è usata la funzione *normalize\_languages(…)* per uniformare i nomi delle lingue, in modo che siano rappresentate in modo coerente.

A questo punto, il dizionario di libri risultante viene serializzato.

1. **export\_books\_db,py**

In questo file viene deserializzato il dizionario serializzato nello script precedente e per ogni libro in esso presente (con numero di pagine > 0) , i generi che erano rappresentati da un’unica stringa vengono trasformati in un array di stringhe, e il numero di stelle viene convertito nell’intero corrispondente. Dopo viene caricato sul database attraverso una richiesta HTTP POST al server locale all’endpoint specificato:

*response = requests.post('http://localhost:4000/api/book/addBook', json=new\_document)*

# WebApplication

1. **backend**

è stato utilizzato **Node.js** con **Express** e **MongoDB** come database. In particolare, è stato utilizzato **Mongoose**, una libreria Node.js per l’interazione con MongoDB.

I vari *schemi* utilizzati sono:

* + ***User*** che rappresenta le informazioni di un utente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente

In questo secondo screen possiamo vedere che viene utilizzato l'approccio del middleware pre di Mongoose per gestire l'hashing della password prima che venga salvata nel database. Questo approccio assicura che la password inserita dall'utente sia criptata utilizzando l'algoritmo bcrypt prima di essere memorizzata nel database.

* + ***Book*** che rappresenta le informazioni di un libro

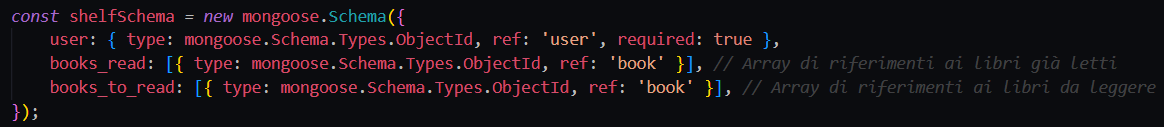
Immagine che contiene testo, schermata, schermo, software

Descrizione generata automaticamente

* + ***Review*** che rappresenta la recensione lasciata da un utente ad un libro

***Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente***

* + ***Shelf*** che rappresenta l’insieme di libri salvati da un utente

*Operazioni:*

* + Login al sito

Funzione **login** in back-end/controllers/authController.js

Immagine che contiene testo, schermata

Descrizione generata automaticamente

* + Registrazione al sito

Funzione **signup** in back-end/controllers/authController.js

Immagine che contiene testo, schermata

Descrizione generata automaticamente

* + Aggiunta libri al database (usata in script python per salvare libri su db)

Funzione **addBook** in back-end/controllers/bookController.js

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

*Scaricamento e conversione dell'immagine*: Nei dati estratti dai file csv abbiamo il link delle varie immagini di copertina dei libri. Nel database vogliamo salvare direttamente queste immagini quindi il codice tenta di scaricare l'immagine dalla rete. Utilizza Axios per fare una richiesta GET al ‘coverUrl’ con ‘responseType’ impostato su ‘arraybuffer’. L'immagine scaricata viene quindi convertita da ‘arraybuffer’ a una stringa in formato ‘base64’ ( il formato Base64 è un modo per rappresentare dati binari in formato testuale). Questa stringa rappresenta i dati dell'immagine convertiti che possono essere inclusi direttamente come parte dei dati del libro salvato nel database. Se il download dell'immagine fallisce per qualche motivo, viene visualizzato un messaggio di errore ma il processo di salvataggio del libro continua senza l'immagine di copertina.

* + Lista dei 10 libri con valutazione media più alta

Funzione **topRatingBooks** in back-end/controllers/bookController.js

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Questa funzione recupera i primi 10 libri con la valutazione media più alta dalla collezione di libri nel database.

La funzione richiama la funzione `getTopRatingBooksAggregationPipeline()` per ottenere la pipeline di aggregazione. Questa pipeline calcola e ordina i libri in base alla loro valutazione media [vedi giù].

Immagine che contiene testo, schermata, linea

Descrizione generata automaticamente Funzione di aggregazione:

Questa funzione getTopRatingBooksAggregationPipeline definisce una serie di stadi di aggregazione per calcolare e recuperare i libri con la valutazione media più alta, ordinati in ordine decrescente di media di valutazione e limitati ai primi 10 risultati.

1. $addFields:
   1. totalReviews: aggiunge un nuovo campo totalReviews al documento. Questo campo è calcolato sommando i valori dei campi number\_stars\_1, number\_stars\_2, number\_stars\_3, number\_stars\_4, e number\_stars\_5. Questi campi rappresentano il numero di recensioni per ogni stella da 1 a 5 di ogni libro.
2. $addFields:
   1. averageRating: aggiunge un campo averageRating al documento.
   2. $cond: operatore condizionale che valuta se totalReviews è uguale a 0.

Se è uguale a 0, allora il libro non ha recensioni e l’averageRating viene impostato a 0.

Altrimenti, calcola la media delle valutazioni:

* Usiamo l'operatore $multiply per moltiplicare il numero di recensioni per stella per il valore della stella (1 \* $number\_stars\_1, 2 \*$number\_stars\_2, etc.…).
* Sommiamo i risultati di tutte le moltiplicazioni usando l'operatore $add.
* Divide la somma totale delle valutazioni per stella per totalReview per ottenere la media effettiva delle valutazioni per quel libro.

1. $project:

Definisce quali campi del documento devono essere inclusi nel risultato, in questo caso: id\_book, title, authors, cover, totalReviews, e averageRating.

1. $sort:

Ordina i documenti in base ad averageRating in ordine decrescente (-1). Quindi i libri con la valutazione media più alta verranno posizionati all'inizio.

1. $limit:

Limita il numero di documenti restituiti a 10. Quindi solo i primi 10 libri con la valutazione media più alta vengano restituiti come risultato.

* + Lista dei 10 libri con valutazione media più alta che abbiano come genere, i generi facenti parte delle librerie dell’utente [se l’utente ha almeno un libro nella libreria]

Funzione **topRatingBooksBasedOnUserShelves** in back-end/controllers/bookController.js

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

const readBooks = await bookModel.find({ \_id: { $in: shelves.books\_read } });

cerca tutti i libri che sono stati contrassegnati come letti dall'utente.

shelves.books\_read contiene un array di ID dei libri che l'utente ha contrassegnato come letti. Quindi la query utilizza l'operatore $in per trovare tutti i documenti nel modello bookModel il cui \_id è presente nell'array shelves.books\_read.

Lo stesso viene fatto per toReadBooks.

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

genres è un array che contiene i generi dei libri presenti nelle librerie dell'utente. Questi generi sono stati ottenuti iterando sui libri letti e da leggere dell'utente e recuperando i generi di ciascun libro.

Viene riutilizzata la funzione getTopRatingBooksAggregationPipeline() definita prima, che restituisce una serie di operazioni di aggregazione per calcolare la media delle valutazioni e ordinare i libri per valutazione media in ordine decrescente, limitandoli ai primi 10.

1. $match:

Filtra i documenti nel modello bookModel in base al campo genres. L'operatore $in è usato per verificare se il campo genres di ciascun libro contiene almeno uno dei generi presenti nell'array genres ottenuto dalle librerie dell'utente.

1. ...topRatingPipeline:

Aggiunge a pipeline, oltre che il match anche le operazioni di aggregazione definite in getTopRatingBooksAggregationPipeline().

* + Ricerca libro attraverso titolo [possibilità di ordinare i libri per titolo, autore, numero di pagine]

Funzione **searchBooks** in back-end/controllers/bookController.js

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

Questa funzione è progettata per gestire la ricerca dei libri, con la possibilità di paginare i risultati per mostrare solo una porzione dei libri per volta sulla pagina web.

1. Parametri di input:
   1. query: È il termine di ricerca inserito dall'utente.

Quando viene eseguita la query bookModel.find({ title: { $regex: query, $options: 'i' } }), MongoDB cercherà tutti i documenti in bookModel in cui il campo title contiene la sequenza di caratteri inserita dall’utente (query), ignorando le differenze tra maiuscole e minuscole (i).

* 1. page: Indica la pagina dei risultati da visualizzare. Di default la prima pagina è 1.
  2. orderBy: Specifica il criterio di ordinamento dei libri. Può essere per titolo, autore o numero di pagine. Se non è specificato, l'ordinamento predefinito è per titolo.

1. skip calcola il numero di documenti da saltare prima di iniziare a restituire i risultati. Questo è calcolato in base al numero della pagina corrente (page) e al limite di libri per pagina (limit).
2. sortCriteria determina come i risultati della ricerca saranno ordinati in base al valore di orderBy. Se orderBy è "title" ad esempio, allora i libri saranno ordinati per titolo.
3. totalBooks conta il numero totale di libri che corrispondono alla query senza limitazione di paginazione. Questo ci serve per calcolare dopo il numero totale di pagine totalPages che l’utente potrà sfogliare, basato sul numero totale di libri trovati e sul limite per pagina.
   * Aggiunta/Rimozione di un libro alla propria libreria

Funzione **addBookToShelf** in back-end/controllers/shelfController.js

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

* + Lista libri presenti nelle librerie dell’utente

Funzione **getBooksByUserShelves** in back-end/controllers/shelfController.js

Immagine che contiene testo, schermata

Descrizione generata automaticamente

* + Aggiunta/Rimozione/Aggiornamento di una valutazione ad un libro

Immagine che contiene testo, schermata

Descrizione generata automaticamenteFunzione **saveReview** in back-end/controllers/reviewController.js

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

\*\*(rating == null) si riferisce al caso in cui ad esempio un utente che aveva inserito 3 stelle ad un libro, riclicca sulle 3 stelle, quindi in questo caso, la review viene rimossa.

*Connessione al database*:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

1. **frontend**

è stato utilizzato **React.js** e altre librerie come **Bootstrap**, **MUI** e **FontAwesome** per realizzare la parte grafica della WebApplication. Le richieste dirette al backend vengono effettuate utilizzando **axios**.

Contenuto pagine:

* Nella *pagina di login* un utente può loggarsi alla piattaforma
* Nella *pagina di registrazione* un utente può registrarsi alla piattaforma
* Nella *home page di un utente non loggato* vengono visualizzati i 10 libri che hanno la valutazione media più alta.
* Nella *home page di un utente loggato* vengono visualizzati i 10 libri consigliati
* Nella *pagina Account* vengono visualizzate le informazioni dell’utente come: username, nome, cognome. Vengono poi visualizzate le due librerie dell’utente: “want to read” e “read”, e i libri a cui l’utente ha aggiunto una review.
* Nella *pagina relativa ad un libro* vengono visualizzate le informazioni del libro, come: titolo, autore, immagine di copertina, descrizione, generi, anno pubblicazione, publisher, lingua, numero di pagine, la media delle valutazioni, numero di stelle. Un utente in questa pagina può lasciare una valutazione al libro e/o salvarlo in una delle sue due librerie.
* Nella *pagina di ricerca* di un libro vengono visualizzati i libri che contengono nel titolo l’input inserito dall’utente.