

Large-Scale and Multi-Structure Database

Reviook Application Report

Application developed by  
Arancio Febbo Salvatore, Di Donato Mattia, Giorgi Matteo

Summary

Introduction

To implement our project, we use Java and Maven with the support of Scene Builder to manage fxml file for the interface of the application.

Data have been stored into MongoDb and Neo4J database. MongoDb was used for store and retrieve information about Users, Authors and Books while the task of storing and retrieving social information that associate Users and Books was assigned to Neo4J.

This application was developed to provide authors the possibility to share their titles and receive from readers the feedback on their works, for this reason the name of the application came from the union of Review and Book. More in detail a user/author can register/login, can search for other users/authors profiles and view statistics of them profile, view suggested users and books, view a book, and leave a review about it. Users and authors have on their profile a useful section for the ranking analytics.

An additional actor of the application is the administrator that have the possibility to search and view authors/users/books profile as the other actor, but has also the capacity to delete reviews, books and users/authors account.

[The application repository is on GitHub](https://github.com/mattiadido95/Reviook)

Functional and Non-Functional requirements

This paragraph describes all the functional and not functional requirements that the Reviook application satisfy.

The application manages three kinds of actors: User, Author, Admin.

* A user has the access to basic function of the application
* An author can also add and delete his/her books
* An admin has all the function of di application except the social ones and can also decide to delete users/authors’s reviews and their account.

Functional Requirements

* An unregister user can only sign up to become a registered user.

A **user** is allowed to:

* Login/Register
* Logout
* View its homepage
* Delete or modify its account
* View Ranking statistics
* Browse information about:
  + Users
  + Authors
  + Books
* View users/authors profile
* Follow or unfollow another user/author
* View book profile
* Insert a book into read or to\_read list
* Review a book
* Like a review
* Modify delete its own review
* View suggested:
  + Users
  + Authors
  + Books

An **author** is a **user** who is also allowed to:

* Add new books
* Delete its books

An **admin** is allowed to:

* Browse information about:
  + - Users
    - Authors
* Browse reviews and books reported
* Delete a book
* Delete a review
* Delete user/author account

Non-Functional requirements

* The application should be simple, intuitive and fast in response.
* The password must be protected and stored in an encrypted form and must be between 8-20 characters.
* The user’s and author’s username must be unique.
* The application should store locally information to reduce network traffic and DBs load.
* The code must be easy to maintain.
* Data may not be updated for up to 10 minutes

Dataset

We collect information about books, reviews, authors and users from two different sources : <https://snap.stanford.edu/data/amazon-meta.html> <https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home> .

The first one is a pain text dataset of Amazon product. From it we extract information about book purchased and reviewed by users. We wrote a program ([GitHub](https://github.com/mattiadido95/JSONconverter)) to reorganize data in a json format and at the same time scrape some missing information. The second is a collection of data from goodreads site, it is already formatted in json and from it we extract similar information of the first one.

The reviews of two dataset were merged in case we found the same ISBN. The two datasets were quite big, so we decided to take from goodreads the books from 2013 to 2017 that have reviews. Some information about authors and users were generated randomly using [java-faker](https://github.com/DiUS/java-faker). We also used the UUID in order to create a different unique identifier for new entity that can been created during the execution of the application.