****

Applications in Object-oriented Programming and Databases:

Book Management Tool

Professor:

Professor Binswanger, Johannes

johannes.binswanger@unisg.ch

&

Ruben Zürcher

[ruben.zuercher@student.unisg.ch](mailto:ruben.zuercher@student.unisg.ch)

Authors: Bastien Girardet

[bastien.girardet@student.unisg](mailto:bastien.girardet@student.unisg).ch

&

Deborah de Wolff

[Deborah.dewolffdemoorsel@student.unisg.ch](mailto:Deborah.dewolffdemoorsel@student.unisg.ch)

21/05/2018

# Purpose

The purpose of our project, namely the book management tool, is to ease the download and storage of books. This task is done by only providing the name of the author, and the book title, the isbn being optional. The main goal is to offer a unified experience of book management using an easy and efficient tool. In other words, we want to create an instrument that helps the majority of the people. The reasons that make our program useful are the following:

* Only a few websites are known for their offering of free pdfs
* We wanted to create an algorithm which helps the peoples to have access to free pdf version of books and give them the tool that facilitate the management of their book library.
* To do so, the websites Libgen.io and Libgen.pw are used.

We will now explain the overall structure of the project and highlight its specific features which are relevant in connection to the class content.

# Structure

Figure 1: Overall Structure of the Project (Application Overview Diagram)

The entire algorithm is dependent on the initial input from the user. The overall process only starts when a user sends anyone of the HTTP request to perform an action. The user has few general possible actions for each of the objects that can be performed through HTTP routes namely GET,POST,PUT,DELETE (See [API documentation](https://documenter.getpostman.com/collection/view/2471406-227dcc87-8dbe-e51d-264a-c6b929bc3593)) : He can GET access to an entry, POST which add new entry, PUT which edit/new data, or DELETE which delete an entry. All those actions are then shaping the design of the user’s portfolios and overall database content. Those possibilities facilitate the management of a book library. In addition, if the goal of the user is to look for a book, he must perform a POST request on the “/books/search” url and provide as a raw “application/json” the name of the author, the title of the book, and its isbn (as empty if optional). The API creates a direct relation between the action of the user, the model and the web scraper, and it gives back to the user a view under the form of a JSON file containing what the user asked for or an unsuccessful demand. In case of a book search, various steps are performed. First, when the user calls the API through the POST request with the required information, the API checks whether the input from the user (the book wanted) is already present in the database, then if not it will run the webscraping algorithm whose goal is to find the user’s book best match, then look for additional information about the book and its pdf.

As a whole, the user is not only restrained at looking for a book. He also can perform actions according to the API routes documentation.

# Database

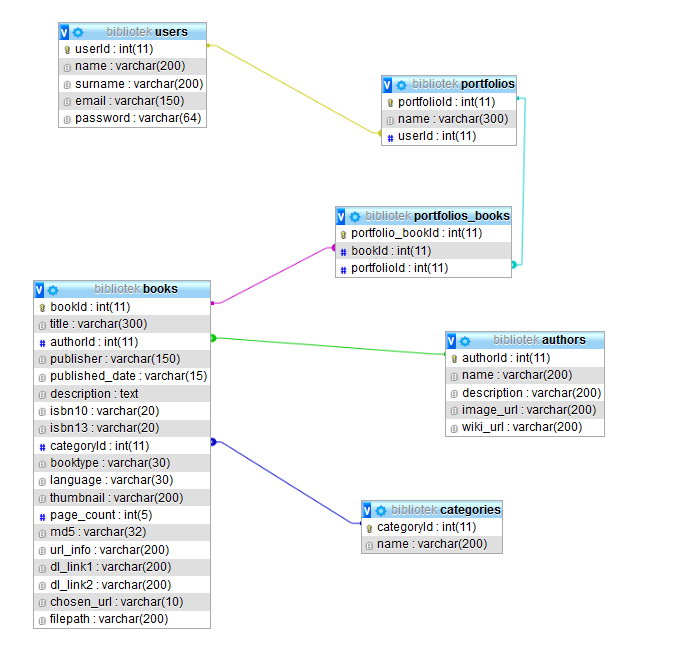


Figure 2: Database ERD

The database is composed of 6 tables. The first table is the “books” table which provides all the information detained about the books, such as the bookid, the title, the publisher, the language, the authorid, the categoryid, etc... Some aspects of this object are mandatory and some aren’t. All the mandatory constituents are assign a “Not Null” characteristic. The table “Books” also receives the information concerning the authors and the categories from two other elements which are called “Authors” and “Category”. Their relations are represented by the term categoryid and authored, present in the “books” table. “Authors” includes all the data concerning the author such as the name, the description, the image, the authorId, and the wiki\_url of each author. However, those attributes are not mandatory, and an entry “Author” can exist without them, expect for the section name and authorid. For “Category”, the only components are the name of the category and the categoryid, which are both mandatory.

A book can then be related to a user portfolio, trough the “Portfolio\_Book” table, an element of the database which include the both foreign keys ‘bookid’ and ‘portfolioid’. This is a relation created to record any book belonging to a portfolio, thus facilitate the management of the book library. In other words, the user has a portfolio that can contain many books, but books are not dependent on the portfolios itself, but on the relation, they have between the portfolio and themselves. Finally, “Portfolio” is linked to the the table “users” with the userid. This table provides all the information of the user, such as the name, surname, email, password, and the userid. To summarize, the database includes six components: “users”, which communicates with “portfolios” which provides information to “portfolio\_book”. Then, “categories” and “authors” which respectively contains information about categories and authors. Moreover, a book is linked to “categories” and “authors” with categoryId and authorId. For more details, please refers to the figure 2 or to the database dump.

However, before downloading those data, the software will have to decide which book’s option found using the API from Google is the most accurate in relation to the initial search. To do so it will do a string comparison between the initial research and all the google results. Then the book best match will be selected by the program. The data of the best match will then be used to perform a http search request on the website Libgen.io. Again the request will result none, one or many books and then, the algorithm will have to select which one is the most relevant for the user a second time using the string comparison algorithm. When the book best match is found, the algorithm will then look whether the author and the category already exist in the database. If it is not the case, our API will then access the Wikipedia library and download information about the author if the author is found. When this step is done, the program will then download the pdf of the book best match found on Libgen.io and then perform a HTTP POST request with all the required details (gathered from Wikipedia API and Google API) to the REST API that will then update the database through SQL Alchemy.

# Documentation

All the code can be found either in the zip file or on the github repo, [here](https://github.com/basgir/bibliotek).

Our API documentation is saved as a collection in Postman. It can be retrieved from [here](https://documenter.getpostman.com/collection/view/2471406-227dcc87-8dbe-e51d-264a-c6b929bc3593) (You will need to register a postman account, it’s free). Or it can be imported in postman with the file : “*Bibliotek.postman\_collection.json”*. It contains all the pre-registered routes of the API. Additionally, all the code is commented with docstrings and normal “#”.

As class diagram, details are available on the [README.md](https://github.com/basgir/bibliotek/blob/master/README.md) on our github. As well as all the rest of the documentation such as the installation, setup, how to run. Moreover, all the credentials required to connect to the database are stated in the appendix. You will need to set up these credentials beforehand using the app.

# Appendix

### Database Credentials

The database is stored on an “AWS” server for an optimal availability. The credentials available here need to be updated in the file **app.py.**

The user is free to change to a local mysql instead. Only he has to change the following credentials.

*# AWS credentials*

DB\_USER = 'bastienroot'

DB\_PASSWORD = 'hGXw%ZYl8MJimhbJ!7eYf58M'

DB\_HOST = 'oopdatabase.cq3xiqh7vtmo.eu-west-3.rds.amazonaws.com'

DB\_NAME = 'bibliotek'