Freie Universität Berlin

Institute for Computer Science of Freie Universität Berlin

QIFTool - Documentation

ver1.0 - 21.10.2020

Robert Selack

robfu@zedat.fu-berlin.de

QIFTool or Query Issue Finder-Tool is a project created as a bachelor's thesis. It aims to help in the quality research field of technical debt by provoding relevant discussions regarding these debts. The discussion are presented in form of issues from github. The tool uses keywords and additional metrics to find potentially interesting issues. Although it is meant for the field of techincal debt the tool can be also be used to return all different kind of issues' topics.

Contents

1 Workflow **3**

2 How to run the program **3**

3 Configuration File - config.ini **4**

4 Interactive Mode **6**

4.1 Functions 7

5 SQLite - Database **9**

5.1 Repositories 10

5.2 Issues 12

1 Workflow

QIFTool is written in Python3.8 and uses Google's Custom Search JSON API in conjunction with Google's Search Engine to filter issues directly on Github. Keywords will be read out of a configuration file to determine which issues should be prefiltered. All prefiltered issues found by the engine will be inserted for research and caching reasons into a SQLite database. afterwards it uses the official Github API (PyGithub) to look through the available metainformation inside each found issue and compares it to the other metrics set in the configuration file to only show the issues that match the requirements.

2 How to run the program

1. Download the 'qiftool.py' and 'requirements.txt' files from the repository
2. Place both files in the desired location
3. Open the terminal and navigate to the files' location
4. Install all dependencies by running 'pip3 install -r /path/to/requirements.txt' or just 'pip install -r /path/to/requirements.txt' depending on your python version
5. Run the program by using 'python3 qiftool.py'
6. By running it for the very first time the tool should have created a 'config.ini' file inside the tool's folder. Fill out the necessary parameters following the instructions in 3
7. With the 'config.file' filled out run the programm again just like in step 5
8. The tool should now operate properly and an interactive mode will be seen. Follow 4 for further instructions

3 Configuration File - config.ini

This file is created by running the program for the very first time. It is used to give the user a space to use their own parameters used by the tool. The file contains three sections for the user to fill out.

```
[DEFAULT]
path_of_database = current
path_of_download = current

[credentials]
github_api_key = c2c1d13983cbb8c1d9ce7845c20d7937ba7c25a0
google_api_key = AIzaSyCWoAxVmPxqD9KZmD0_7YWx7dEkX28nwGs
google_cse_id = de940461e6efec5b9

[metrics]
keywords = technical debt   refactor    rewrite
issue_comments = 5
repo_contributors = 50
```

1. `[DEFAULT]`

   - this section contains the path for the database and downloaded repositories to be stored in. The user is able to create their own path with the location of the ’qiftool.py”s as a pivot. These can be changed by providiung a valid path on your machine.

2. `[credentials]`

   - this section contains the corresponding credentials necessary to run the used APIs

   (a) `github_api_key`

      i. register on github

      ii. use this link and click on ’generate new token’ to create a new key

iii. paste the key as a parameter

(b) `google_api_key`

i. register on google

ii. use this link and click on 'Get a Key' to create a new key

iii. either choose a project or create a new one

iv. follow the instructions and paste the key as a parameter

(c) `google_cse_id`

i. login to the google account created in the prior step

ii. use this link and click on the project you used to create the google key with

iii. look for the 'Search engine ID' and paste the ID as a parameter

3.

4. `[metrics]`

These contain the metrics used for the google search. For further details for understanding each metric please refer to the tables in 5.

(a) `keywords`

- string of characters with each element separated by a tabulator. Note that the keywords will be used to find patterns that exactly match. So 'refactor' will find 'refactoring' but not vice versa. In addition the keywords are connected with a logical and.

(b) `issue_comments`

- an integer over 0. It will show issues that have at least the amount of comments set in this metric. So 5 will yield issues with 5 or more comments.

(c) `repo_contributors`

- an integer over 0. It will show issues that have at least the amount of contributors working on the corresponding repository. So 5 will yield issues with more 5 or more contributors working on its repository.

4 Interactive Mode

Once you successfully configered the configuration file in 3 an interactive mode will be seen on the console after running it. In this mode the program will wait for the user

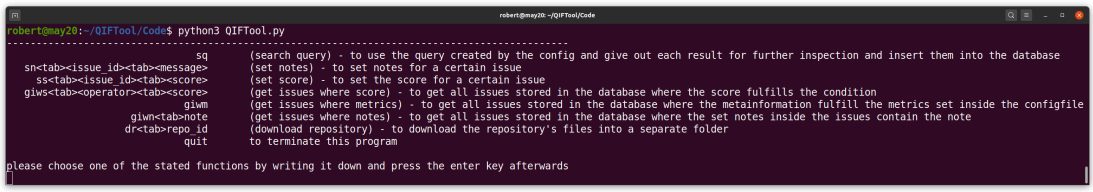

Figure 1: default interactive mode

to simply type a desired function into the console and confirming it by pressing enter. After being done with a function the program goes back to displaying the interactive mode as it loops itself around it.

4.1 Functions

function	description
sq	(search query) - start the google search. The metrics set in the configuration file will be used to determine what results will be found and shown.
sn<tab><issue_id><tab><message>	(set notes) - sets a note for a certain issue inside the database **issue_id** - a string of numbers. Found within the *issue_id* field in either the ouput or database of the issue. **message** - a string of characters that will be inserted into the *notes* field inside the database.
ss<tab><issue_id><tab><score>	sets a score for a certain issue inside the database. **issue_id** - a string of numbers. Found within the *issue_id* field in either the ouput or database of the issue.**score** - a number chosen by the user to represent its relevance.
giws<tab><operator><tab><score>	(get issues where score) - displays all issues stored in the database where the score fulfills the condition set by the user. **operator** - all comparison operators allowed by the SQL. <, >, =, <=, >= **score** - a number chosen by the user to represent its relevance and compare the issues inside the database to.

function	**description**
giwm	(get issues where metrics) - displays all issues stored in the database where the pieces of metainformation fulfill the metrics set inside the configfile. This function yields the same functionality as the 'sn'-function but with the database being the source.
giwn<tab>note	(get issues where notes) - displays all issues stored in the database where their notes contain the note set by the user with this very function **note** - a string of characters. This can be used in conjunction with SQL-syntax like providing a " or % around the note.
dr<tab>repo_id	(download repository) - downloads the repository's files into a separate folder. This folder's location is set by the configuration file. The strucutre of the downloaded files also is identical to that of its respository.
quit	terminates this program.

5 SQLite - Database

This tool uses the SQLite version 3.33.0 (2020-8-14) library. The database created with this tool consists of two tables with one table refering to the other in a 1:n relation.

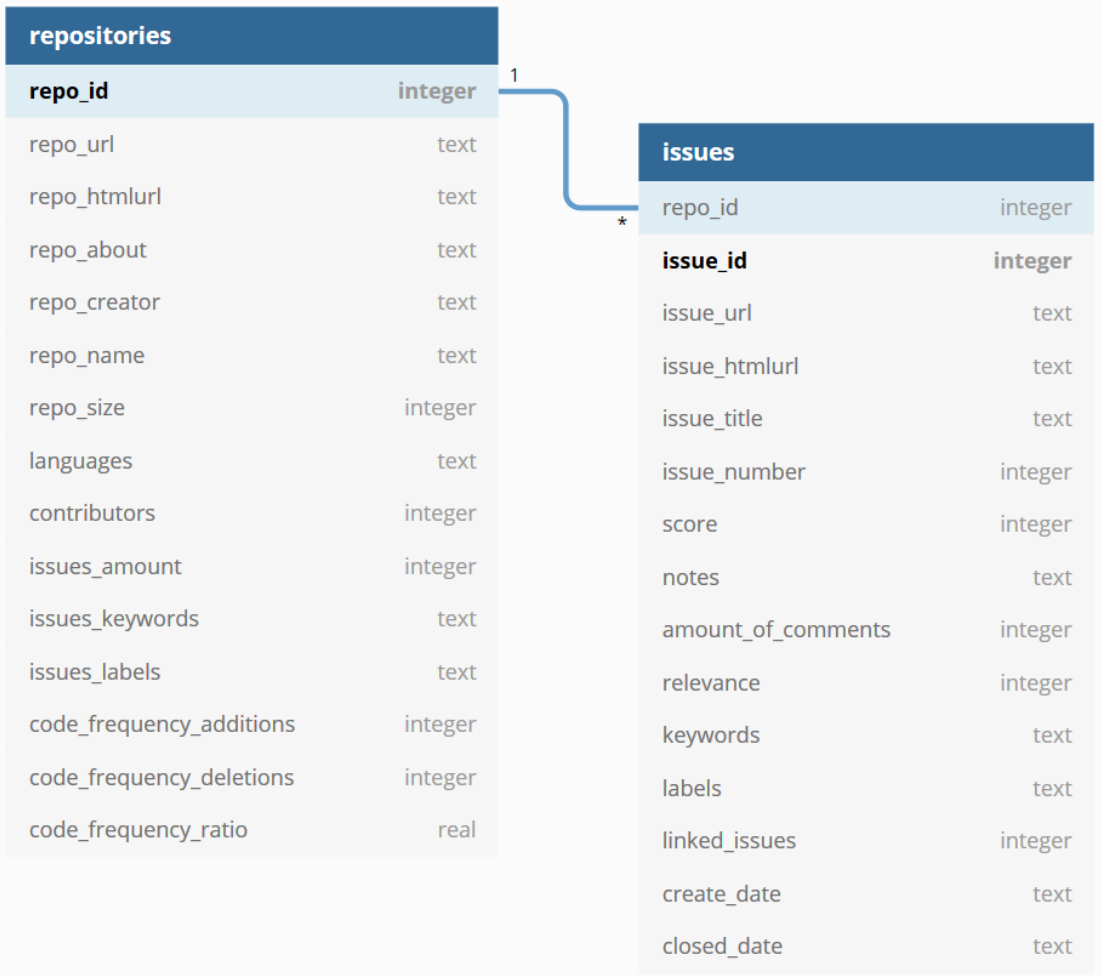

Figure 2: Database schema

5.1 Repositories

attribute	datatype	description
repo_id (primary key)	integer	indentifier for a repository
repo_url	text	url for the JSON file of this repository
repo_htmlurl	text	url that refers to the web based github repository
repo_about	text	contains the 'about' of the repository found on github. It's a brief description of the repository
repo_creator	text	name of the creator of this repository or fork
repo_name	text	name of the repository
repo_size	integer	size of this repository in MB
languages	text	list of programming languages used in this repository
contributors	integer	amount of contributors of this repository
issues_amount	integer	amount of issues of this repository

attribute	datatype	description
issues_keywords	text	a list of all keywords that where used to find issues related to this repository
issues_labels	text	a list of labels that were used for the issues found
code_frequency_additions	integer	overall amount of lines of code added to this repository
code_frequency_deletions	integer	overall amount of lines of code deleted from this repository
code_frequency_ratio	real	quotient of the lines of code added and deleted. A value between 0 and 1 with 1 meaning all code that was added got deleted and 0 meaning all code that was added none got deleted.

5.2 Issues

attribute	datatype	description
repo_id (foreign key)	integer	identifier for a repository
issue_id	integer	identifier for an issue
issue_url	text	url for the JSON file of this issue
issue_htmlurl	text	url that refers to the web based github issue
issue_title	text	title of the issue
issue_number	integer	relative number of this issue created within its repository
score	integer	value to set by the user. Used for the user's own usage of a rank system. Makes it possible to rank found issues relative to each other in order to find more valuable issues easier later on
notes	text	string to set by the user. Used for the user's own organisation. Makes it possible to note interesting attributes about a certain discussion or topic sorting. It's possible to look for certain patterns inside the set notes
amount_of_comments	integer	amount of comments that the issue has

attribute	datatype	description
relevance	integer	value corresponding to the relative order of issues found with a query. The higher up (earlier) the issue has been found the higher its relevance. Third hit on page two equals a relevance of 23. These relevances change to show an all time best relevance every time the issue has been found.
keywords	text	list of all keywords that has been used to find this issue over all query iterations
labels	text	list of all labels that are used with this issue
linked_issues	text	list of all issues that are linked to within this issue. *This attribute has yet to be implemented*

attribute	datatype	description
create_date	text	date of the creation of this issue. Although the datatype is a 'text', SQLlite still recognizes the string as a date due its formatting
closed_at	text	date of when the issue was closed. Although the datatype is a 'text', SQLlite still recognizes the string as a date due its formatting. If the issue has not been closed yet this field is set to 'NA'