Development of a tool for automating the collection and analysis of open data GitHub users

Sergei A. Isaev[a], Ilya B. Gosudarev[b] and Irina B. Gotskaya[c]

[a]ITMO University, 49 Kronverksky avenue, Saint-Petersburg, 197101, Russian Federation
[b]ITMO University, 49 Kronverksky avenue, Saint-Petersburg, 197101, Russian Federation
[c]ITMO University, 49 Kronverksky avenue, Saint-Petersburg, 197101, Russian Federation

Abstract

GitHub is incredibly popular among researchers, thanks to its detailed documentation, which helps to develop research tools. That said, most of the research related to the use of the GitHub API does not show the proper implementation of tools. And you can't explore the codebase, which makes it even harder. As a result, it's nearly impossible to repeat the research results and adopt the practice of creating such instruments. All of this imposes some restrictions on the work of researchers. This article takes into account the peculiarities of the GitHub API, and provides you with the necessary tips for creating a software tool that can effectively solve a typical research problem. It also shows the results of a prototyping tool and its effectiveness, as well as takes a look at the most obscure problems and their solutions.

Keywords

GitHub, Data collection, Application architecture, JavaScript, Research tool

1. Introduction

Code repositories like GitHub, GitLab or Bitbucket are not only top software tools, huge repositories and codebase archives, but also promising sources of diverse information.

GitHub is particularly interesting to researchers. As of August 2019, GitHub claimed to have over 100 million repositories and more than 50 million members [1]. The majority of projects available at GitHub are open source repositories that can be explored using dedicated tools. Today it is possible to connect to GitHub using the REST API and GraphQL API. This article will cover only one of these methods - REST API.

Due to the extensive amount of information available on GitHub, it is possible to develop tools for research tasks. Such tools allow the researcher to automate information extraction processes. Same goes for the storage and post-processing process. Having a suitable tool or possessing the necessary knowledge to create one, the researcher will be able to conduct a more complete and qualitative research.

In the works we studied related to the use of the GitHub API [2, 3, 4], the features of the implementation of the tools used to work with this service were poorly reflected. Moreover, the

The Majorov International Conference on Software Engineering and Computer Systems, December 10–11, 2020,Saint-Petersburg, Russia
0994486@gmail.com (S. A. Isaev); goss@itmo.ru (I. B. Gosudarev); iringot@yandex.ru (I. B. Gotskaya)
0000-0002-6479-7587 (S. A. Isaev); 0000-0003-4236-5991 (I. B. Gosudarev); 0000-0003-3074-8936 (I. B. Gotskaya)

codebase was missing.

The purpose of this article is to study the peculiarities of working with the GitHub API. This in turn will help you to develop a convenient and extensible tool that will be able to effectively solve a research problem in an almost automated mode. The creation of a prototype of such a tool helps to study the effectiveness and relevance of the solutions developed, as well as identify unobvious problems. Here is a small console app that was invented as a prototype: https://github.com/rukivbruki/GitHub-API-research-tool.

2. Problem statement and application requirements

The research problem that has to be solved by the application being built is formulated to demonstrate the advantages of automation when operating with the GitHub API.

In order to avoid synthetic approaches, the task description was balanced in accordance with authentic research cases. For instance, there is a study that helps to identify experts in software libraries and frameworks among GitHub users. It describes one of such tasks [4]. Here is a description of the slightly changed task for the prototype under consideration: it is necessary to find out the number of projects that use JavaScript machine learning libraries in their dependencies in a particular region of Russia. In this case TensorFlow.js was chosen as the required library. The projects will look for @ tensorflow / tfjs dependencies (which is a core for TensorFlow.js) and @ tensorflow / tfjs-node - TensorFlow(an extension for Node.js). Being one of the biggest IT sector in Russia, Saint Petersburg was chosen as the place of residency the authors of researched projects [5].

2.1. Recursive resource access

GitHub API's key feature is to get resources using the recursive method. Calling an API is done through sending multiple HTTP requests to api.github.com. The service returns a JSON response and includes a subset of attributes for each requested resource. Every attribute is a URL referring to a subsection within the page. At the top-level of the query tree, the GitHub API offers access to various types of information and services, moderately delving deeper into subsections of each type. This organization of queries should be taken into account when scraping through user pages.

2.2. Optimizing web requests

During its operation, the app can make a huge amount of requests. Let's say that you need to analyze the contents of each user's repository in the area. Even in the most trivial case the number of operations needed would be equal to that for a directed graph to be crossed in depth. And this is without taking into account transitional requests for obtaining a list of users, accessing the database, and requests that require authentication or pagination (it can be ignored) d) [6]. In terms of asymptotic notation, the running time of such an algorithm will be:

$$T(n,k) = O(n+k), \tag{1}$$

where O - time complexity of the algorithm, n - the number of users, k - the number of repositories for each user.

The GitHub API provides a special tool that can reduce the number of requests, which is known as the Search API. With its help you can look for a specific element, as it passes a combination of search qualifiers in the query parameters [7]:

```
q = SEARCH_KEYWORD_1 + SEARCH_KEYWORD_N + QUALIFIER_1 + QUALIFIER_N.
```

Thanks to the use of the Search API, the running time of the algorithm described above becomes linear with the number of users found. In addition to that, you can scan file contents using the mechanism for storing each file in blob format (large binary objects).

Due to the combination of various schemes for constructing a query, it becomes possible to reduce the total time of information extraction, as well as decrease the load on the working system. Moreover, it allows you to diminish the number of errors when executing a query.

2.3. Implementing pagination

Response headers play a big role in building requests using the REST API. GitHub hosts a large number of non-HTTP headers that provide information about the API service. As a matter of fact, it is in such headers that links to the next page of the search results are placed to organize pagination.ers that links to the next page of the search results are placed to set up pagination. In order to create auto-pagination based on the query results, you should take all of these features into consideration.

2.4. Accounting for connection limits

The X-RateLimit response header informs you of the remaining limit for calls to this API. The limit is imposed on the frequency of requests. There is a different limit for different types of requests. In some cases, it can be 30 requests per minute, while in others it can take up to 5,000 requests per hour. It is very important to remember this. If you violate this condition, you will keep seeing 403 Forbidden Error messages. Thus, your app should be able to handle such errors [2].

2.5. Accounting for search results limits

The GitHub Search API has a serious limitation. Regardless of which sections of the service the search query is directed to, there will be a maximum of 1000 first results. You can solve some research problems even with this restriction. However, the task set for the test tool requires you to bypass this restriction. One of the best decisions would be to combine keywords and search qualifiers, thereby narrowing the scope of your search. Instead of one general query returning a large amount of data, you can completely change the parameters for submitting narrower queries with a limit of up to 1000 results per each. Having done so, you will increase the code cohesion and scale your app, but at the same time automate and speed up the research process.

2.6. Data storage and logging

First and foremost you need to make sure that the received data and logs are saved. The very least you should do is write data to a CSV or TSV file for subsequent efficient import into the data analytics system. In case of a more progressive version of the application, it is recommended to provide users with the ability to save information to the database.

Since the application was established for research purposes, one of the more important features is the ability to log information that is not directly related to the collection of data about users of the service. This can be data about the time of requests, the duration of their execution, codes and stack trace errors, as well as any other type of information received in the server response headers. Having collected this data(which you can later use for subsequent analysis), you will be rewarded with valuable information, including how the GitHub service operates. An additional requirement related to logging can be a logging level requirement. There are main levels available in the simplest version:

- Info
- Debug
- Error

2.7. Extensibility and scalability

At the time of writing, the app prototype only has one service that covers one research case. However, if desired, there can be a lot more cases. You will need to create a new service for each research task. Striving for the unification of the research tool leads to the formulation of the problem of developing a modular solution with minimal coupling. The more modules and components will be available for reuse, the easier it will be to implement and test new software functionality, effectively scale the app, and improve its stability.

3. Application architecture

A basic console app has been developed as a prototype, and has all of the necessary functions. JavaScript was chosen as the programming language, while Node.js was selected as the software platform together with Express (a web application server framework). At the same time, Axios works as a JavaScrit library to make HTTP requests. The Pino library has been chosen as the logging tool. Finally, a general purpose distributed database, known as MongoDB, was used as a database.

The application was developed using a component-based approach to meet all of the necessary requirements, including flexibility and scalability. A block diagram of the components [8, 9] in Figure 1 demonstrates the principle of interaction of various application subsystems:

The PredictionModule is shown as an example of a functional extension. It wasn't yet realized in the prototype itself at the time of the writing. Nonetheless, it is easy to imagine a situation when the app will require additional auxiliary modules for processing, analyzing, storing and distributing the collected data.

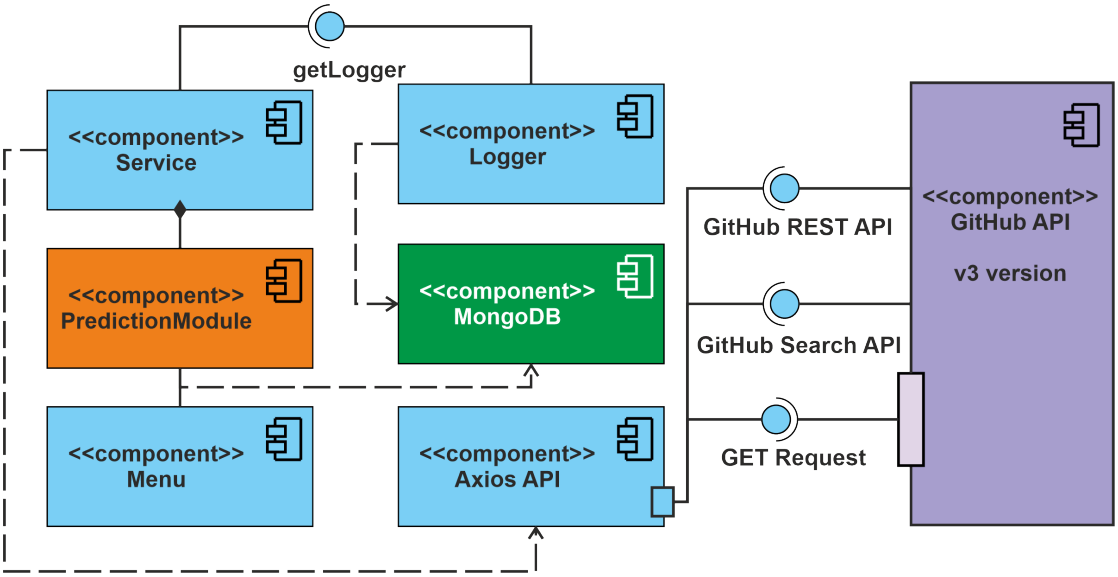


Figure 1: Diagram of application components.

Components called Service, Logger, Menu and Axios API shown in Figure 1 are types of elements that can be deployed asindependent within a microservice architecture or as independent containers. Docker containers are one of the brightest examples. The latter, in turn, can be easily managed using container orchestration tools, like Docker Swarm or Kubernetes [10]. Such an approach ensures the proper execution of all the attributes of the app and helps to manage resource consumption. And scaling by creating additional replicas of the necessary modules provides, among other things, increased reliability of the entire system.

Application use cases can include the need to retrieve a wide variety of information. That being said, there are only a few examples of basic types [9] - Figure 2:

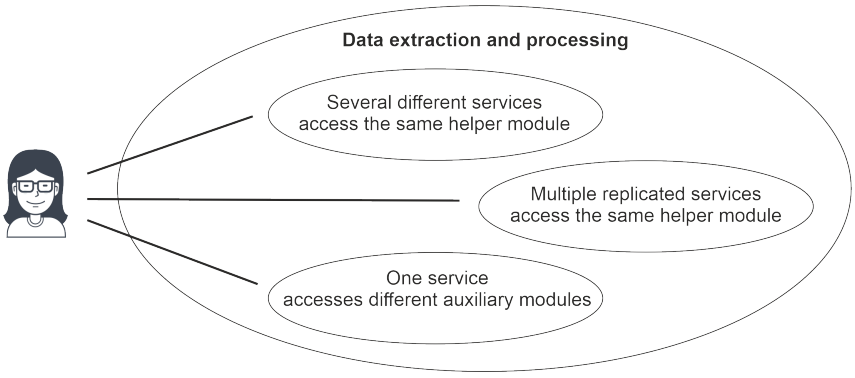


Figure 2: App use case diagram.

- Different services access the same helper module
- Multiple replicated services access the same helper module
- One service accesses different auxiliary modules

4. Features of the application

The general operating principle of the application is depicted in the sequence diagram (see Figure 3).

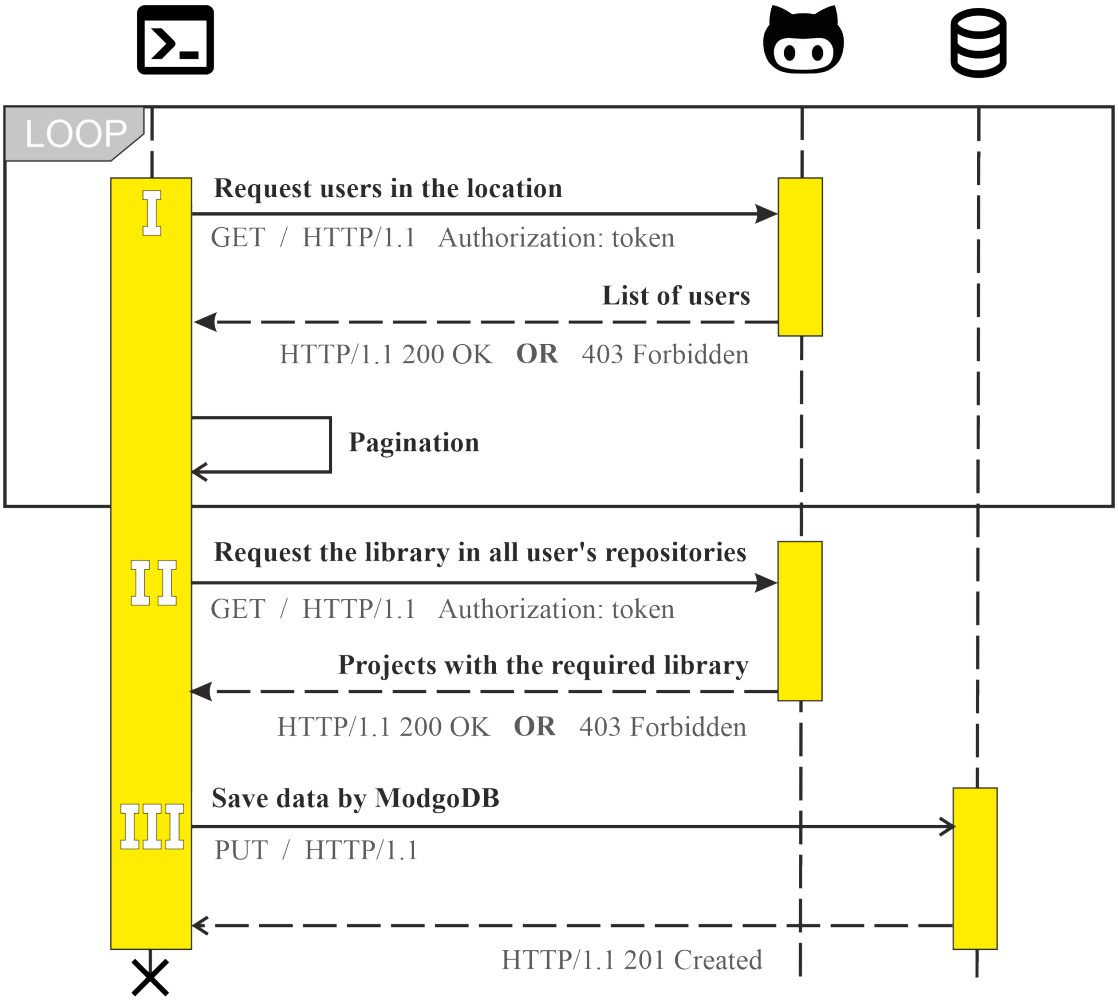


Figure 3: App sequence diagram.

4.1. First stage

The application sends requests to the GitHub API to get user data in the selected region. A pre-generated token is then sent with each request to the Authorization header, which in turn helps to increase the limits of the calls to the service by up to 5,000 requests per hour.

The query results are processed in JSON format, which contains up to 100 desired values (the parameter can be configured) [11], and a link to the next part of the results is passed in the response headers and retrieved by the application using a regular expression.

Asyns generator functions were used to perform pagination and sequential retrieval of resources. They allow you to organize and put a continuous stream of received data in order, as well as transfer it for further processing.

Search qualifiers are used for queries by area, which restricts the number of results given to 1000 results per query. Luckily, there have been a lot of parameters added recently that allow you to get around this limitation. As a result, you can narrow down the field of interest and split one query into several local ones:

```
q=location%3A${place}+language:javascript+created:${date1}..${date2}&per_page=100
```

The "Created" parameter was chosen as the limiter, which sorts users by their date of registration on GitHub. The filter function was taken over by the language parameter. It only selects users whose primary programming language is JavaScript.

Despite the fact that the "Lang" parameter increases the search speed, it also severely decreases its quality. Our app automatically includes this parameter, comparing the most popular libraries being searched with the development languages that are typical for these kinds of libraries.

The optimal settings of the limiter, as well as the degree of influence of the "Lang" parameter on the quality of search results, is beyond the scope of the article and may be examined in the future studies.

4.2. Second stage

At the second stage of the app each identified user is addressed in the body of an asynchronous loop and the attributes for each of its repositories are retrieved. It then searches for the repository files that describe the application's dependencies, such as Package.json or Bower.json, and extracts information about the presence of the required library.

GitHub Search API search limits are then increased from 10 to 30 requests per minute [7] similarly to the first stage.

There is also a mechanism that allows to send requests with a specified frequency and helps minimize authentication errors. This is especially crucial for this stage, considering the small amount of connection limits.

4.3. Third stage

The third stage is associated with the logging procedure and saving the received data. At this stage the app prototype is able to save data to a CSV file, and logs to a simple text file. The GitHub REST API provides data in JSON format, so the extended version of the application uses a document-oriented database such as MongoDB to store it. In case the app is required to

collect a huge amount of data, then your best bet would be columnar databases, like Google BigQuery or Yandex ClickHouse.

5. Application results

As already mentioned, the output of the application can be divided into two main components. The first is the data directly related to the research task, which is shown in Table 1.

Table 1
Application results

Region	Library	Total Number of Users	Active Users	Found projects
Saint Petersburg	React.js	13474	11183	29427
Saint Petersburg	TensorFlow.js	13474	11183	26

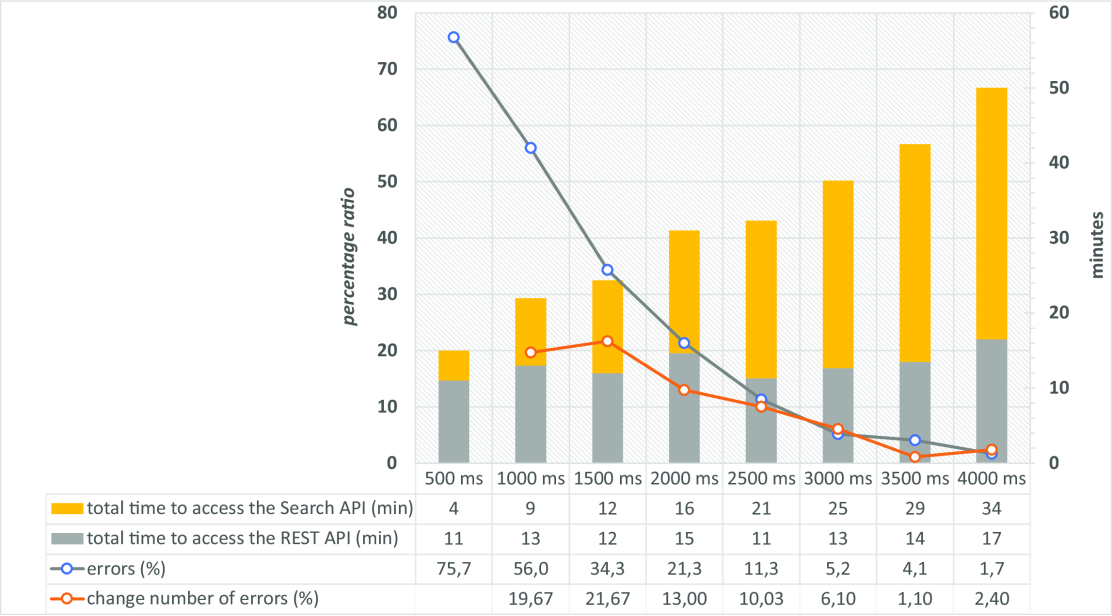
Table 1 shows that there are 13474 users on GitHub who have selected Saint Petersburg as their current residency. Of these users, only 11182 of them have at least one repository. Our app has successfully done its job and found 26 projects with dependencies that indicate the use of the TensorFlow.js library. It took 8 hours and 32 minutes to complete the task in an automated mode.

The interpretation of these results is beyond the scope of this article. However, the overall results can be quite helpful in other studies, including the ones that prefer statistical methods or use machine learning algorithms and artificial neural networks.

The second component is the data that demonstrates aspects of the operation of the GitHub service itself. Ancillary data is just as important as the target information you are looking for. For instance, having analyzed different modes of application operation and the rate of request submission, we managed to obtain the average data on the frequency of "Access Denied Errors"[1]. The settings that made it possible to choose the optimal speed of access to the GitHub API, a sort of threshold, were calculated empirically. Figure 4 shows data on the frequency of occurrence and the rate of change in the number of "Access Denied Errors" as a percentage of all sent requests, depending on their submission rate. The graph along the secondary axis (scale on the right) shows the accumulation of time spent accessing resources through the GitHub REST API and GitHub Search API. The optimal speed of sending requests is equal to 3000 milliseconds. In case you exceed this threshold, there is a big chance that you will keep getting "Access Denied Errors". At the same time, lesser values almost do not give a decrease in the percentage of denials, but increase the time to complete the research task. Despite this, the ability to regulate the request submission rate is an important setting, depending on the research objectives.

It's worth noting that the received results are somewhat different from the official documentation, which recommends a request rate of 30 requests per minute. That is, 1 request every 2000 milliseconds. Pull requests are not optimal either.

[1]The application was launched 10 times for each of the settings for the rate of submission of requests to the API GitHub - from 500 to 4000 ms in increments of 500 ms. Data was collected for the first 500 users GitHub in St. Petersburg and React.js as a search library in user projects.



	500 ms	1000 ms	1500 ms	2000 ms	2500 ms	3000 ms	3500 ms	4000 ms
total time to access the Search API (min)	4	9	12	16	21	25	29	34
total time to access the REST API (min)	11	13	12	15	11	13	14	17
errors (%)	75,7	56,0	34,3	21,3	11,3	5,2	4,1	1,7
change number of errors (%)		19,67	21,67	13,00	10,03	6,10	1,10	2,40

Figure 4: App performance indicators varying from the speed of access to GitHub.

6. Summary

Github is an extremely valuable source of diverse research information that can be later collected, processed, and analyzed through an automated tool. Creation of a prototype helped to define a number of necessary requirements for these types of tools:

- Recursive resource access
- Optimization of web requests
- Pagination implementation
- Accounting for connection limits
- Accounting for search result limits
- Data storage and logging
- Extensibility and scalability

The prototype app was built on Node.js. platform. Thanks to the implementation of the component approach in the design of the application, it became possible to deploy it as part of a microservice architecture. Correspondingly, it allowed to make the app even more trustworthy, extend its functionality and scalability.

We have described the main stages of app development. Taking them into account will help in designing programming tools that are similar in functionality.

With the help of the developed and implemented prototype, we have managed to solve the biggest research problem, which required bypassing 13474 GitHub user accounts from Saint Petersburg. As a result, we were able to identify projects that use one of the machine learning

libraries in JavaScrip - TensorFlow.js. There were 26 such projects in total. It took 8 hours and 32 minutes to complete the assigned search task.

The additional collected data allowed the prototype developers to make recommendations on the optimal speed settings for accessing GitHub API resources to minimize access denial errors. The approximate speed is 3000 milliseconds for each request.

Additionally, the collected data allowed the prototype developers to give recommendations on the optimal settings for the speed of accessing the GitHub API resources to minimize "Access Denied Errors". The approximated speed is 3000 milliseconds for each request. There is also a certain inconsistency when comparing this parameter with the official documentation.

References

[1] Github user search, 2020. URL: https://github.com/search?q=type:user&type=Users.

[2] F. Chatziasimidis, Data collection and analysis of github repositories and users. in: 2015 6th international conference on information, intelligence, systems and applications (iisa), Corfu, Greece, 2013.

[3] P. Dello Vicario, Tortolini, Evaluating a programming topic using github data: what we can learn about machine learning, International Journal of Web Information Systems 17 (2021) 54–64. doi:10.1108/IJWIS-11-2020-0072.

[4] E. João, Identifying experts in software libraries and frameworks among github users. in: 2019 ieee/acm 16th international conference on mining software repositories (msr), Montreal, Canada, 2019.

[5] Petersburg bypassed moscow in the concentration of it companies, 2020. URL: https://www.rbc.ru/spb_sz/31/01/2020/5e33d2b39a7947aa2ac037f9.

[6] T. Rafgarden, Sovershennyy algoritm. Osnovy, Piter, Saint Petersburg, Russia, 2019.

[7] Search github docs, 2017. URL: https://docs.github.com/en/free-pro-team@latest/rest/reference/search.

[8] D. Arlou, I. Neyshtadt, UML 2 i Unifitsirovannyy protsess. Prakticheskiy obyektno-oriyentirovannyy analiz i proyektirovaniye. 2nd edn, SimvolPlyus, Saint Petersburg, Russia, 2007.

[9] M. Fauler, UML. Osnovy. 3nd edn, SimvolPlyus, Saint Petersburg, Russia, 2004.

[10] A. Markelov, Vvedeniye v tekhnologii konteynerov i Kubernetes, DMK Press, Moscow, Russia, 2019.

[11] Resources in the rest api – github docs, 2020. URL: https://docs.github.com/en/free-pro-team@latest/rest/overview/resources-in-the-rest-api.