Università
della
Svizzera
italiana

**Software
Institute**

Software & Data Engineering Master Thesis Proposal

Edoardo Riggio

API Scout: An Information Retrieval System for OpenAPI Specifications

Abstract

The primary objective of this thesis is to create an information retrieval system enabling users to explore a compre hensive collection of OpenAPI specifications. The system will use historical versioned data - scraped from existing github repositories - as well as user-uploaded data.

This platform serves a dual purpose, catering to both academics and developers. In the case of academics, this platf orm can serve as a repository from which to conduct research, facilitating more in-depth studies on API specificatio ns and their evolution. On the other hand, developers can look up several different examples that could help them get started with their own OpenAPI documentation.

The main contributions of this thesis include two crucial aspects: data engineering and software engineering. In the first case, the thesis will delve in the process of refining raw data and finding ways to classify the API specifications. In the second case, the thesis will define the architectural design and the development of the service itself.

Advisor: Prof. Dr. Cesare Pautasso
Co-Advisor: Souhalia Serbout

Contents

List of Figures

1 Introduction

In the ever-evolving landscape of software engineering, APIs play a pivotal role in facilitating the communication and interoperability between software services. Being software systems intrinsically complex, it is good practice to keep track of the interfaces that they expose. Moreover, especially in the cases where these interfaces have to be used by external developers, it is paramount to have good documentation for such interfaces.

In this thesis, we will work with a specific type of API: HTTP APIs. Such APIs are exposed by Web servers and accessible though HTTP requests. To properly document public HTTP APIs, Swagger proposed in 2011 the ”Swagger Specification” (today known as ”OpenAPI Specification”). As Swagger explains on its website, ”[t]he OpenAPI Specification (OAS) defines a standard, language-agnostic interface to HTTP APIs which allows both humans and computers to discover and understand the capabilities of the service [...].” [1]

Every service with public endpoints, should expose – as per Swagger's specifications – a `openapi.json` or `openapi.yaml` file. This file can be one, or it can be the root of several other smaller files – at the discretion of the author. Moreover, each document that specifies the endpoints' structure must be written following a specific set of rules. Such rules are defined and explained on Swagger's ”OpenAPI Specification” website.

1.1 Motivation

From our research, we noticed that there is a void in tools and services that can be helpful in the study of the evolution and current state of HTTP APIs. Although platforms such as SwaggerHub[1] and APIs.guru[2] already exist, they lack features that are essential for research.

For example, they lack a proper search system. In the previously mentioned platforms, indexing of specifications is superficial and most of the time only the title is matched with the query. With API Scout, we want to index API specifications based not only on their title, but based on all the tags containing natural language.

Another feature missing in the aforementioned platforms is a proper visualization system. By using tools to convert specifications into interactive trees, users can better understand the structure of the service's endpoints. Moreover, we can also plot some statistics about the selected API metrics to visually compare it to all the specifications present on our platform.

1.2 Objective

The goal of this thesis is to build a platform that both academics and developers can use to understand and study the intricacies of real-world HTTP APIs.

In the case of the developers, they can search and consult our historical and current collection of APIs to better understand how to structure one. Aided by the graphical tools, a developer can also understand how the endpoints should be structured to obtain a more cohesive structure.

On the academic side, this tool can be used to perform research on the evolution of HTTP

[1]https://app.swaggerhub.com/search
[2]https://apis.guru/

APIs in time. By offering historical data as well as current data, researchers can have a better understanding of how endpoints and data structures change over time as a service grows in complexity. Researchers can also use plots and statistics pre-computed by our platform to understand how an API specification relates to the plethora of specifications present.

2 State of the Art

2.1 OpenAPI Specification

An API is a set of interfaces that can be accessed by other users or developers. To document such APIs, API specifications are used [2]. These specifications document the functioning and expected output of such interfaces. The OpenAPI Standard [1] is a specification language that offers a standardized way of writing API specifications for HTTP-based APIs.

2.2 Document Embedding

To find the most similar documents to a user-defined query, we need a way of comparing documents and defining what do we mean by similar. First of all, we need to represent documents in a more mathematical way [3]. For this task, a document embedding model is employed – we used the Universal Sentence Encoder [4]. This encoder will transform a document into a vector of numbers. These vectors of numbers are then stored somewhere – ideally a database – and assigned a name.

When a user wants to search for a specification, we have to use the document embedding model for the query, and then retrieve the top K documents that are the most similar to the vectorized query. The similarity of two documents is defined by how much the two vectors (the one of the query and the one of the document) are distant from one another. We can use one of the following metrics to define similarity: L2 norm, dot product, cosine, and maximum inner product. When dealing with vectorized textual data, the best metric to use is the cosine similarity metric [5], which is the one we chose for our platform. The more the two vectors are close to each other, the more they are similar. To find out the K nearest documents to the query, we can use the K-Nearest Neighbors (K-NN [6]) algorithm. This algorithm will return the K nearest documents, thus the most similar.

2.3 Evaluation

To evaluate the performance of our solution, we used average precision and recall. In information retrieval systems [7], precision is defined as the inverse of the position in which the document – defined in the ground truth – was found in (Equation 1). Where POS is the position in which the document was found in (1 to 5 if the document is in the top 5, 0 otherwise).

$$P_i = \frac{1}{POS_i} \tag{1}$$

The average precision is the average of the precisions among all queries defined in the ground truth (Equation 2). Where N is the total number of queries in the ground truth.

$$\overline{P} = \frac{1}{N} \cdot \sum_{i=1}^{N} P_i \tag{2}$$

Finally, the recall is defined as the number of documents found in the top 5 – regardless of the position they were found in – over the total number of queries (Equation 3). Where C is the number of documents found in the top 5, and N is the total number of queries in the ground truth.

$$R = \frac{C}{N} \tag{3}$$

2.4 t-SNE Plots

A t-SNE plot is an algorithm used to visualize high-dimensional data in a two- or three-dimensional space [8]. This algorithm is a variation of the Stochastic Neighbor Embedding [9] approach presented in 2002. The t-SNE variation, according to the authors, is ”much easier to optimize, and produces significantly better visualizations by reducing the tendency to crowd points together in the center of the map.”

With respect to the context of this proposal, the t-SNE plot is used to evaluate the output of the implemented information retrieval system [10]. In our case, after retrieving the most similar documents to the query given in the ground truth, we plot these documents. By using t-SNE, we can show the semantic distance between the documents and the relative queries (Subsection 3.4).

3 Preliminary Results - Feasibility Study

With the API Scout platform being a data aggregator for OpenAPI Specifications (section 2.1), we need a way of efficiently indexing and searching through the plethora of crawled documents in our database. The goal of this feasibility study is to have a better understanding of the technologies and algorithms to use during the development of the platform.

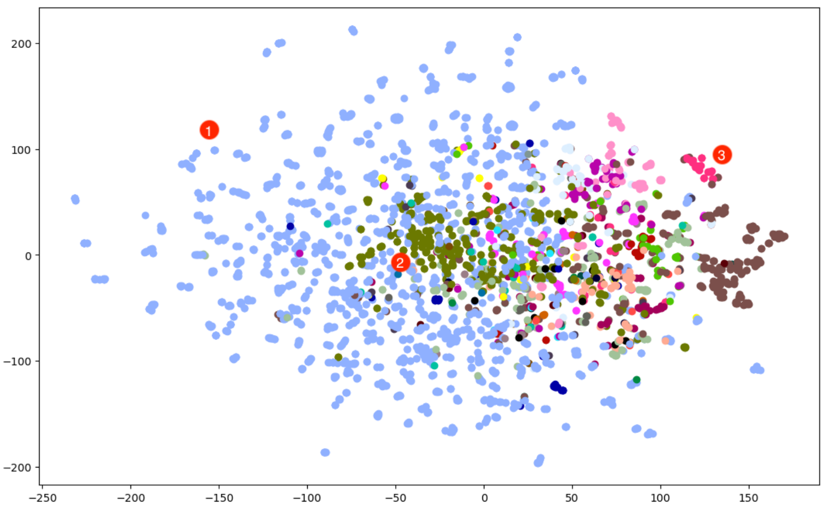
In the following sections, we will see an in-depth explanation on how the indexing and retrieval of documents will be implemented, as well as an evaluation of this solution on a test dataset. The dataset used in this study is composed of ∼3000 specifications taken from APIs.guru. This dataset has been used in several other research papers on APIs [11, 12, 13, 14, 15, 16, 17].

3.1 Data Analysis

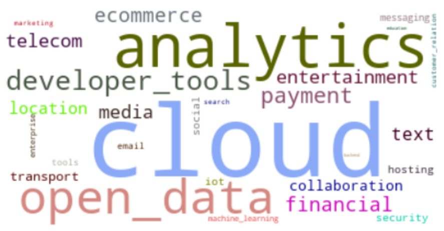
Before delving further into the approach taken for the indexing and retrieval of the documents, we shall analyze the documents contained in the APIs.guru dataset.

During this phase, we had to visualize the semantic relationship of the documents. To do so, we embedded (Subsection 2.2) the documents in 512 dimensions. Once embedded, we performed a dimensionality reduction using t-SNE (Subsection 2.4). At this point we added the third dimension representing the category of the document, and plotted the points. The category of the API specification is taken from the `info.x-apisguru-categories` tag manually assinged by APIs.guru.

The aforementioned steps produce the result depicted in Figure 1. As we can see from both plots, the majority of the documents have been tagged with the ”cloud” (**1**) and ”analytics” (**2**) labels, even though most of them are not tightly clustered together. Upon closer inspection, though, we can see that most – if not all – documents from the ”marketing” (**3**) category are very close to one another.



(a) Documents Color-Coded by Category



(b) Category Word Cloud

Figure 1. APIs.guru Documents Plots

3.2 Approach

When dealing with OpenAPI Specifications, we are working with JSON or YAML files (which are structured). Moreover, these files follow the OpenAPI Specifications; thus we know that the documents will contain a very similar structure.

Since we are dealing with a very high number of documents, it is paramount to have a fast and reliable index that can be searched through. For this reason, we decided to use vector embeddings to represent documents in our database. Embedding models offer a way of transforming documents into vectors of numbers, and – in our case – we used Google's Universal Sentence Encoder [4] to perform the aforementioned task. The embedding is done on only a part of the document's content, namely everything that is contained in the `description`, `name`, `title`, and `summary` tags.

We chose to do so because these tags are the only ones that contain natural language descriptions of the API in general or of its endpoints. After the creation of the embeddings, we saved the vectors – as well as the name, version, and id – of the specifications in an ElasticSearch database. We chose ElasticSearch and the ELK (Elastic-Logstash-Kibana) stack because it is very rapid and scalable. Moreover, ElasticSearch offers the possibility of searching documents based on the K-NN algorithm.

This solution seems to be the most promising for the amount of data and hardware availability we have for this project. We were considering also deep-learning models but discarded them since they are resource intensive and would be overkill for this kind of application.

```
...
aws iot service on secure tunneling
AWS IoT Secure Tunneling

driving license registration service
Transport Department

sentiment analysis service api
Text Analytics & Sentiment Analysis API | api.text2data.com

email mailbox checker
MailboxValidator Free Email Checker
...
```

Figure 2. Ground Truth

3.3 Ground Truth

To evaluate the performance of our information retrieval system, we used a ground truth. Our ground truth consists in a textual file in which we have a query and the expected result. Some examples are depicted in Figure 2. The file is divided into blocks of two lines. The first line represents the query that will be vectorized and passed to the K-NN query. On the other hand, the second line represents the title of the document that needs to be found in the top 5 results of the query. The title of the document on the second line of the block has been chosen manually, this means that we read some of the API specifications and wrote a query that can be connected to that specific document.

However, in some cases, there can be multiple documents that satisfy one query; for example, in the case of the query ”driving license registration service”. In the database, there are several different APIs related to the query mentioned: ”Transport Department, Tamil Nadu” and ”Transport Department, Haryana”. In such cases, we take the first occurrence that starts with the words ”Transport Department”.

3.4 Results

The results obtained with the embedding + K-NN solution are very promising, as we can see in the t-SNE plot (Figure 3, Subsection 2.4). With the ”Y” markers representing the queries, and the dots representing the most similar documents found for that query, we can see that the dots and ”Y” markers of the same colors are fairly close to each other.

Moreover, we performed a validation step with a ground truth. From the validation step, we concluded that the average precision and recall of the document retrieval (Subsection 2.3) are:

$$\overline{P} = 0.6 \qquad R = 1.0$$

This validation was done on 10 different queries. The position in which each ground truth appeared in the searches can be found in Figure 4. As we can see, all ground truths we found in the top 5 results – hence $R = 1.0$. The validation on the ground truth, though, does not paint the full picture of the capabilities this solution offers. For example, Figure 5 represents the top 5 documents returned by the information retrieval system if we try searching using

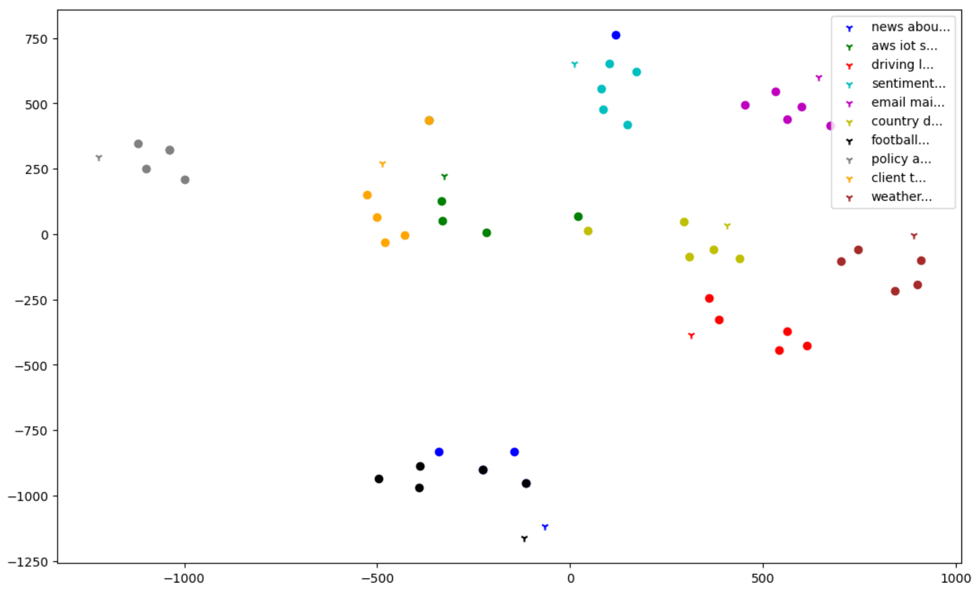


Figure 3. t-SNE Plot

```
Position in which the ground truth was found:

    "NFL v3 Ply-by-Play" was found at position #1
    "AWS IoT Secure Tunneling" was found at position #4
    "Transport Department" was found at position #4
    "Text Analytics & Sentiment Analysis API | api.text2data.com" was
      found at position #4
    "MailboxValidator Free Email Checker" was found at position #1
    "Interzoid Country Data Standardization API" was found at position #1
    "Soccer v3 Projections" was found at position #1
    "PolicyClient" was found at position #1
    "NetworkManagementClient" was found at position #3
    "Interzoid Get Weather City API" was found at position #2
```

Figure 4. Ground Truth Evaluation

```
These are the top 5 results of the query "american sports news":

    1. NFL v3 Play-by-Play     v.1.0   [67%]
    2. News Plugin             v.1     [66%]
    3. Soccer v3 Projections   v.1.0   [64%]
    4. MLB v3 Projections      v.1.0   [64%]
    5. NFL v3 Scores           v.1.0   [64%]
```

Figure 5. Example Output of a Query

the query: ”american sports news”. As we can see in the results above, the engine is able to recognize that both the *NFL* and the *MLB* are professional American sport leagues. The former being the National Football League, and the latter being the Major League Baseball. Moreover, it is also able to understand that soccer is a sport and return it.

As can be seen both in the average precision ($\overline{P} = 0.6$), and the above example, there are still some noisy results that match part or none of the queries, but this is acceptable. The reason is that this is not the only means of searching. To further filter the documents returned by the information retrieval system, we will implement a DSL for the user to specify filters to apply to the NL query.

3.5 Comparison with APIs.guru

After validating our system against a tailored ground truth, we compared the results of API Scout with the ones of APIs.guru. The comparison was performed by taking the queries from the ground truth from Subsection 3.3 and searching for them on the APIs.guru website. The result of this evaluation is depicted in Figure 6. APIs.guru's platform has an average precision ($\overline{P}$) of 0, and a recall (R) of 0.

```
Position in which the ground truth was found:

    "NFL v3 Ply-by-Play" was NOT found
    "AWS IoT Secure Tunneling" was NOT found
    "Transport Department" was NOT found
    "Text Analytics & Sentiment Analysis API | api.text2data.com" was
      NOT found
    "MailboxValidator Free Email Checker" was NOT found
    "Interzoid Country Data Standardization API" was NOT found
    "Soccer v3 Projections" was NOT found
    "PolicyClient" was NOT found
    "NetworkManagementClient" was NOT found
    "Interzoid Get Weather City API" was NOT found
```

Figure 6. APIs.guru Evaluation

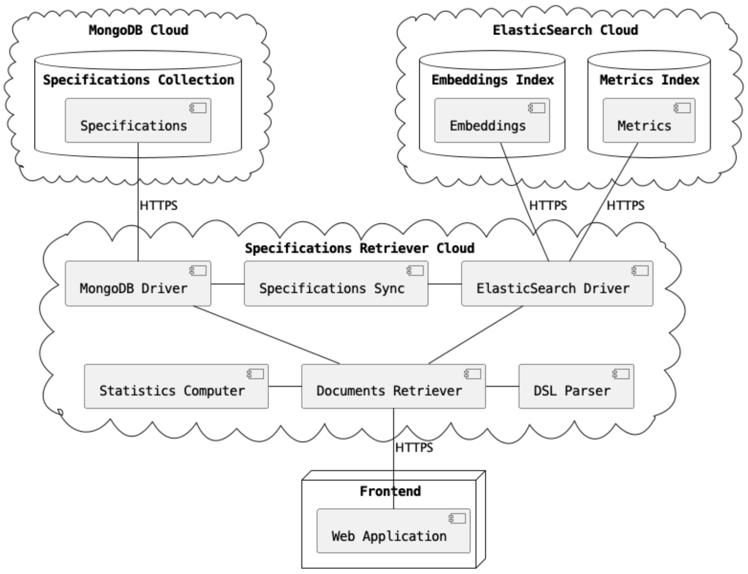


Figure 7. API Scout Architecture

4 Proposal

During the spring semester, we will work on three macro-areas: platform development, platform evaluation, and report writing. A drilled-down list of the aforementioned macro-areas can be found in the next section of this proposal (Section 5).

The architecture of API Scout is defined in Figure 7. The system will be divided into four main parts: the ElasticSearch cloud, the MongoDB cloud, the backend, and the frontend.

ElasticSearch Cloud This part of the system will contain two indices, one for the embeddings of the OpenAPI specifications, and one for the pre-computed metrics of the OpenAPI specifications.

MongoDB Cloud This part will contain one collection containing all the ∼500k scraped OpenAPI specifications.

Backend This part of the system performs all the necessary operations on the databases and exposes an HTTP API to the frontend. More specifically, it will keep the two databases in sync, parse the DSL passed from the frontend, compute on-the-fly statistics given a user-defined query, and finally it retrieves the documents that are most similar to the given query (Subsection 3.2).

Frontend This last part will handle all the user interaction with the collection of OpenAPI specifications. The user will be able to search the collection, as well as visualize the structure of the specification and analyze the evolution of a given API through time.

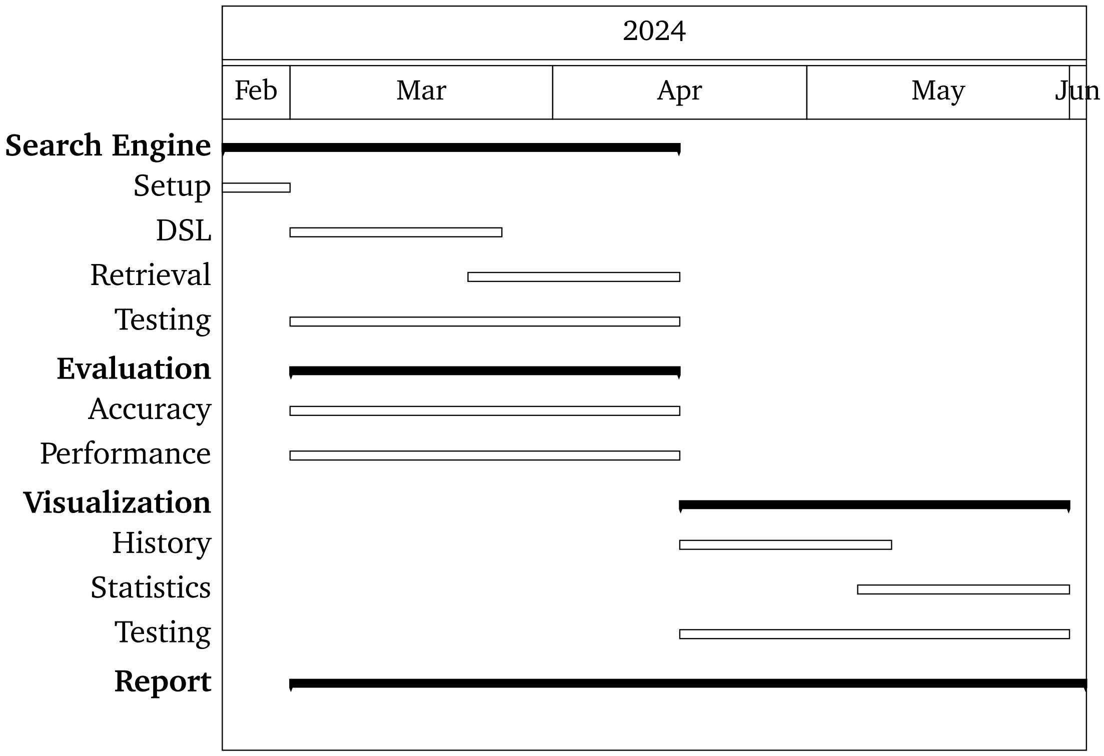
5 Work Plan for the Spring semester

This section describes a high-level view of the work plan for the spring semester.

- Iteration 1: Search Engine Phase
  - Task 1: Project setup and database migration

- Task 2: DSL implementation and parser
- Task 3: Information retrieval implementation
- Task 4: Write tests for the system

- Iteration 2: Evaluation Phase
  - Task 1: Evaluate the accuracy of the results
  - Task 2: Evaluate the performance of the system
- Iteration 3: Visualization Phase
  - Task 1: API history given different versions
  - Task 2: General and query-related statistics
  - Task 3: Write tests for the system
- Iteration 4: Report



References

[1] Swagger. OpenAPI Specification v3.1.0 | Introduction, Definitions, & More, February 2021.

[2] Brajesh De. *API Management: an Architect's Guide to Developing and Managing APIs for Your Organization*. Apress, Berkeley, first edition edition, 2017.

[3] Andrew M. Dai, Christopher Olah, and Quoc V. Le. Document Embedding with Paragraph Vectors, July 2015. arXiv:1507.07998 [cs].

[4] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal Sentence Encoder, April 2018. arXiv:1803.11175 [cs].

[5] Ken Guo. Testing and Validating the Cosine Similarity Measure for Textual Analysis, October 2022.

[6] T. Cover and P. Hart. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967. Conference Name: IEEE Transactions on Information Theory.

[7] W. Frakes and R. Baeza-Yates. Information Retrieval: Data Structures and Algorithms. June 1992.

[8] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[9] Geoffrey E Hinton and Sam Roweis. Stochastic Neighbor Embedding. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2002.

[10] Jaakko Peltonen and Ziyuan Lin. Information retrieval approach to meta-visualization. *Machine Learning*, 99(2):189–229, May 2015.

[11] Shang-Pin Ma, Ming-Jen Hsu, Hsiao-Jung Chen, and Chuan-Jie Lin. RESTful API Analysis, Recommendation, and Client Code Retrieval. *Electronics*, 12(5):1252, January 2023. Number: 5 Publisher: Multidisciplinary Digital Publishing Institute.

[12] Yun Wan Kim, M. Consens, and O. Hartig. An Empirical Analysis of GraphQL API Schemas in Open Code Repositories and Package Registries. 2019.

[13] Chung-Hsuan Tsai, Shi-Chun Tsai, and Shih-Kun Huang. REST API Fuzzing by Coverage Level Guided Blackbox Testing, December 2021. arXiv:2112.15485 [cs].

[14] Jerin Yasmin, Yuan Tian, and Jinqiu Yang. A First Look at the Deprecation of RESTful APIs: An Empirical Study, August 2020. arXiv:2008.12808 [cs].

[15] Sae Young Moon, Gregor Kerr, Fran Silavong, and Sean Moran. API-Miner: an API-to-API Specification Recommendation Engine, July 2023. arXiv:2212.07253 [cs].

[16] Jinqiu Yang, Erik Wittern, Annie T. T. Ying, Julian Dolby, and Lin Tan. Towards extracting web API specifications from documentation. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, pages 454–464, New York, NY, USA, May 2018. Association for Computing Machinery.

[17] Shang-Pin Ma, Ming-Jen Hsu, Hsiao-Jung Chen, and Yu-Sheng Su. API Prober – A Tool for Analyzing Web API Features and Clustering Web APIs. In Kuo-Ming Chao, Lihong Jiang, Omar Khadeer Hussain, Shang-Pin Ma, and Xiang Fei, editors, *Advances in E-Business Engineering for Ubiquitous Computing*, Lecture Notes on Data Engineering and Communications Technologies, pages 81–96, Cham, 2020. Springer International Publishing.