API Topic Issues Indexing, Exploration and Discovery for API Community Knowledge

George Ajam
College of Information Technology
University of Babylon
Hilla, Babylon, Iraq
george@itnet.uobabylon.edu.iq

Carlos Rodríguez
Departamento de Electrónica e Informática
Universidad Católica "Nuestra Señora de la Asunción"
Asunción, Paraguay
carlos.rodriguez@uc.edu.py

Boualem Benatallah
School of Computer Science and Engineering
UNSW Sydney
Sydney, NSW, Australia
b.benatallah@unsw.edu.au

***Abstract*—Application Programming Interface (API) is a core technology that facilitates developers' productivity by enabling the reuse of software components. Understanding APIs and gaining knowledge about their usage are therefore fundamental needs for developers that impact a wide range of software development activities. This paper presents an approach to enable API users to explore, discover and learn about APIs through API topic issues discussed in Stack Overflow (SO), a widely used programming, community question-answering (CQA) site. Our work proposes an integrated API Knowledge Base (KB) and indexing technique that combines both SO API-related posts as well as other API learning resources collected from the Web (e.g., API video-tutorials from Youtube). The resulting indexed and enriched API community knowledge can be queried in a API-topic-issue driven manner using a simple yet powerful domain-specific language (DSL). We demonstrate the feasibility of our approach through *Scout-bot*, our tool for exploration and discovery of API topic issues.**

***Index Terms*—API community knowledge, API topic issues, API Indexing, API query bot**

I. Introduction

Developers rely on several APIs for their day-to-day software development tasks [1]. They also resort to several resources in order to learn about the usage of APIs. Examples of such resources include CQAs such as SO[1], API descriptions [2], code examples [3], among other resources. Yet, the problems of searching and finding API-related knowledge that satisfies the needs of developers are still open and they are far from being completely solved. In this context, there are several studies in the literature on the analysis [4], extraction [5] and recommendation [3] of API-related knowledge such as documentation and usage examples.

One such line of studies includes the improvement of API learning resources. Works along this dimension focus on the problems of lack of API usage examples [6], missing descriptions in API documentation [7] and insufficiency of usage patterns [8]. Other studies include the factors affecting the usability of APIs [9] and code snippets in software-related documentation (including APIs) [2]. Further topics of interest in the literature include the production [10]–[14], problems/issues [15]–[18] and enrichment [7], [8], [19]–[21] of API learning resources.

Despite all the studies and proposed approaches in this context (we elaborate more on this in Section 2), it is still challenging to explore and navigate API knowledge and their related issues [22]. This is partly due to the heterogeneous and fragmented information found across different sources, making it difficult for querying, exploration and discovery purposes. In this paper, we address this problem by organizing API community knowledge into API topics issues to enable discovery and exploration. We focus on the latter, as opposed to traditional keyword-based search, because the it is more appropriate for exploration and learning purposes, particularly, in unfamiliar spaces [23]. In concrete, in this paper, we propose:

- An API-topic-centric data model for building the foundation for a KB to represent and store relevant API knowledge and API topic issues.
- An API knowledge indexing technique to support API topic issues exploration that enables the various API resources to be accessed conveniently in an API-topic-driven manner.
- An enrichment technique for API topics terminology that leverages word embeddings [24]. The proposed approach helps provide developers with more flexibility for expressing their API topic issues queries.
- A query bot that helps querying API topic issues through a simple yet powerful domain-specific query language. We showcase how such query bot can be implemented and seamlessly integrated into existing productivity tools.

We organize this paper as follows: Section 2 introduces background and related work. Section 3 presents our API topic issues model and our approach to indexing and enrichment of

[1]https://stackoverflow.com

G. Ajam, C. Rodriguez, B. Benatallah. "API Topic Issues Indexing, Exploration and Discovery for API Community Knowledge", The XLVI Latin American Computing Conference - CLEI 2020, Loja, Ecuador

API topic issues. Section 4 presents our DSL for querying API topic issues. In this same section, we also propose *Scoutbot*, a bot for querying the API topic issues enriched index, including its implementation details and evaluation. We close this paper with Section 5 presenting the final discussions and future directions for this work.

II. Background and Related Work

APIs are vital in software evolution [11], data exchange and provision of functionalities between software components [25] and services [26]. API learning resources (e.g., API reference documentation) are typically the starting point to learn about APIs. However, oftentimes they are insufficient when it comes to learning about how APIs work [15], forcing developers to resort to various alternative resources [27] (e.g., CQA sites like SO). In this section, we provide background and related work for management of API learning resources and the issues faced in this context.

One of the main topics of interest in the literature is that of *API learning resources problems*. Works in this context explore issues found in such resources including problems in content and presentation, changes in APIs and their evolution, undocumented exceptions and their implications, among other issues. For instance, Uddin and Robillard [15] reported on problems/failures in API learning resources and categorized them into two main sub themes: Content and presentation. Problems with *content* include unclear description of an API element, descriptions of API elements in unexpected areas, examples without explanations, descriptions with references to old API versions, inconsistency and incorrectness. Problems with *presentation* include excessive details, bloat descriptions, and structural information issues (e.g., fragmented details). Sohan et al. [16] studied changes in API documentation, communication and versioning to understand how Web APIs evolve. While Kechagia et al. [17] investigated the undocumented exceptions in traces of Android applications crashes and found that 19% of them are the result of undocumented exceptions. Overall, these works show that issues in API learning resources are pervasive, pushing programmers to alternative resources such programming CQAs [18].

The *production of API learning resources* has also been the target of a number of studies. Works under this category explore the production of learning resources to support the usage of APIs. Studies here focus on the management of contributions (e.g., [10]), navigation and search for locating proper information to support API documentation production tasks (e.g., [28]), and tools for actually producing API documentation (e.g., [12]). Under management of contributions, Thominet and Luke [10] studied how author support guides are constructed and improved to produce API documentation in Wikis. Watson [13] studied practices for writing API documentation to enable researchers and technical writers to produce effective documentation. While Pawlik et al. [14] explained how the crowd was managed toward producing the documentation of Numpy[2] Python library.

Navigation and search are considered key tasks in finding relevant information such as code examples and summaries that can be used to support the production of API documentation. Here, Gvero et al. [29] proposed a free syntax form queries to automatically generate code fragments, while in [30] researchers focused on finding relevant reusable classes and methods using search engines. Similarly, Kim et al. [28] proposed a data mining approach to simplify the task of finding code examples and recommending them with summaries of API usage.

Works in the context of actual API documentation production includes Scribble [31], which was proposed to generate documentation for a programming language called PLT scheme's[3], [32] proposing an authoring tool for creating help pages, and, ScribeCrowd [33], which manages the process of composing technical documents and engages experts in writing tasks.

Finally, the literature also features extensive work in the context of *API learning resources improvement*. Works along this dimension aim to address problems such as the lack of usage examples in API documentation (e.g., [6]), insufficiency of usage patterns (e.g., [8]), and the lack of descriptions in API documentation (e.g., [7]). In particular, we observe that researchers put emphasis on the importance of usage examples to improve API documentation. Here, API users and developers are not only after a short code snippet, but they also need additional resources that help with API descriptions and explanations. For instance, McMillan et al. [19] developed a system called Exemplar, which finds software projects of high relevance so that they can be considered as examples. Usage patterns are equally important as examples. Studies in this area investigated techniques to find usage patterns through various data mining approaches. For instance, Nguyen et al. [8] proposed a JavaScript (JS) web application for mining inter-procedural usage patterns. Additionally, in the context of lack of API descriptions, Dasgupta et al. [20] proposed an automated approach called ENTRANCER, which takes source code, requirement documents and other artifacts to extend the client's API calls with related information. While Inozemtseva et al. [21] proposed a tool called NEWTON, which automatically links references (e.g., Github repositories, JavaDoc, code reviews) to source code. NEWTON use references from the Web to enable code searchability.

The wealth of studies found in the literature regarding the management and issues related to API learning resources is a patent indication of the pervasive problems and challenges faced in making APIs accessible and understandable to end users of these APIs. The work presented in this paper aims to contribute to the literature through an API-topic-centric approach that allows API users to explore and discover API community knowledge. We do this through an enriched API KB, handy domain-specific query language and query bot that

[2]https://numpy.org/
[3]https://plt-scheme.org/

helps facilitate such tasks. We elaborate more on this in the following sections.

III. API Topic Issues Modeling and Indexing

Several resources can be found nowadays on the Web that can help with API learning. Among these resources, SO is one of the most widely used and indispensable resources for facilitating the usage and understanding of APIs [34]. It features a vast repository of programming knowledge including 16 million questions, 25 million answers, 68 million comments and 53 thousand tags as of August 2018. Study [35] has shown that API-related posts in SO can be categorized into topic issues such as API security, API usage, API Debugging, among other topics issues. In this section, we propose a data model to represent, enrich and index API-related posts in SO based on such topic issues, in order to support exploration and discovery of API-related knowledge. We discuss next the data model as well as the extraction/curation and indexing techniques we use to represent and index such resources.

A. Data Model

Given the relevance of SO in the programming community, we developed a SO- and API-topic-centric data model to capture knowledge about APIs. Leveraging on our experience and results from previous research [35], we introduce the model in Figure 1. Here, *API* is a software released in a current version represented by *API version*. An API version is typically accompanied by an *API documentation*, where the API documentation can be a reference documentation, getting-started guide or an open API specification (e.g., a Swagger API Documentation[4]). An *API version* contains several API methods represented by the *Method* entity of our model. *Insights*, in turn, are obtained from our information sources and include metrics about posts and developers using the API.

Next, an *API Topic* represents a topic issue, as identified in [35]. *SO Code Example* represents code examples that contain a method from a specific API version. These code examples can be either obtained automatically or from a human-curated list of examples. An API, may contain several other learning resources such as getting-started videos from Youtube[5], code examples from Github[6], among other types of resources. Finally, and, most importantly, our data model includes *Stack Overflow posts*. This entity represents a question, answer or a Wiki[7] posted on SO.

Notice that, in addition to including entities and attributes from SO's dataset in our model (see Figure 1), a key characteristic lies in the API-topic-centric approach we followed in the model. Such modelling decision is key to building a solution that enables the indexing, exploration and discovery of API-related knowledge based on API topic issues as we will be discussing in the next sections.

[4]https://swagger.io

[5]https://youtube.com

[6]https://github.com

[7]https://meta.stackexchange.com/questions/11740/what-are-community-wiki-posts

B. API Community Knowledge Extraction and Curation

We leverage on the API community knowledge in SO and the data model introduced in the previous section in order to build our API KB. The first step toward extracting such API-related knowledge consists in identifying SO resources (e.g., posts) that relate to known APIs. Curated lists of known APIs can be obtained from existing directories of APIs such as ProgrammableWeb[8]. By leveraging SO's search APIs[9] we can construct keyword-based queries that can help us retrieve posts that relate to specific APIs. For instance, in order to search for posts related to Windows API[10] (or WinAPI for short), we can build a query containing the keyword "WinAPI" as tag, as shown in Figure 2. More complex queries can be built using SO's APIs, e.g., to include posts that contain specific keywords in the body of posts (see footnote 9 for more details).

In SO, not all posts are properly tagged and curators from the community may add tags over time as needed. This can lead to missing the extraction of relevant content simply because posts were not tagged properly with the corresponding API name. We therefore resort to queries that retrieve posts based on APIs mentioned not only within tags but also title and body of posts. As explained before, such list of mentions can be obtained from existing list of APIs available on the Web (see footnote 8). Additionally, in order to assign topics to each post extracted from SO, we use a list API-topic-issues seed keywords, e.g., based on the topics listed in [35]. Such list of seed keywords are expanded with alternative mentions obtained from a pre-trained word embedding based on SO [36]. This way, whenever we identify mentions from our list of API-topic-issues in a post, we associate the post to the corresponding topic issue.

In addition to SO posts, we further curate our API KB by enriching it with additional resources such insights, code examples, API documentation, among other resources (see our model in Figure 1). Such enrichments can be sourced from other sites such as Github (e.g., for code examples) and Youtube (e.g., for video-tutorials). Github and Youtube also provide search APIs[11] [12] that can help retrieve API-related resources, which we leverage in our work.

C. Indexing

When an API consumer, programmer or practitioner is interested, e.g., in learning about "security" issues related to "Facebook Graph API", they typically need to make several assumptions and keyword-based guesses in order to find relevant information. This, in turn, may not bring all posts that tackle the target API issues, limiting search results only to exact keyword-based matches. While this type of searches is popular in many platforms (e.g., SO) to look for information that help them solve issues related to specific programming tasks, it is

[8]https://www.programmableweb.com

[9]https://api.stackexchange.com/docs/search

[10]https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list

[11]https://developer.github.com/v3/search

[12]https://developers.google.com/youtube/v3/docs/search

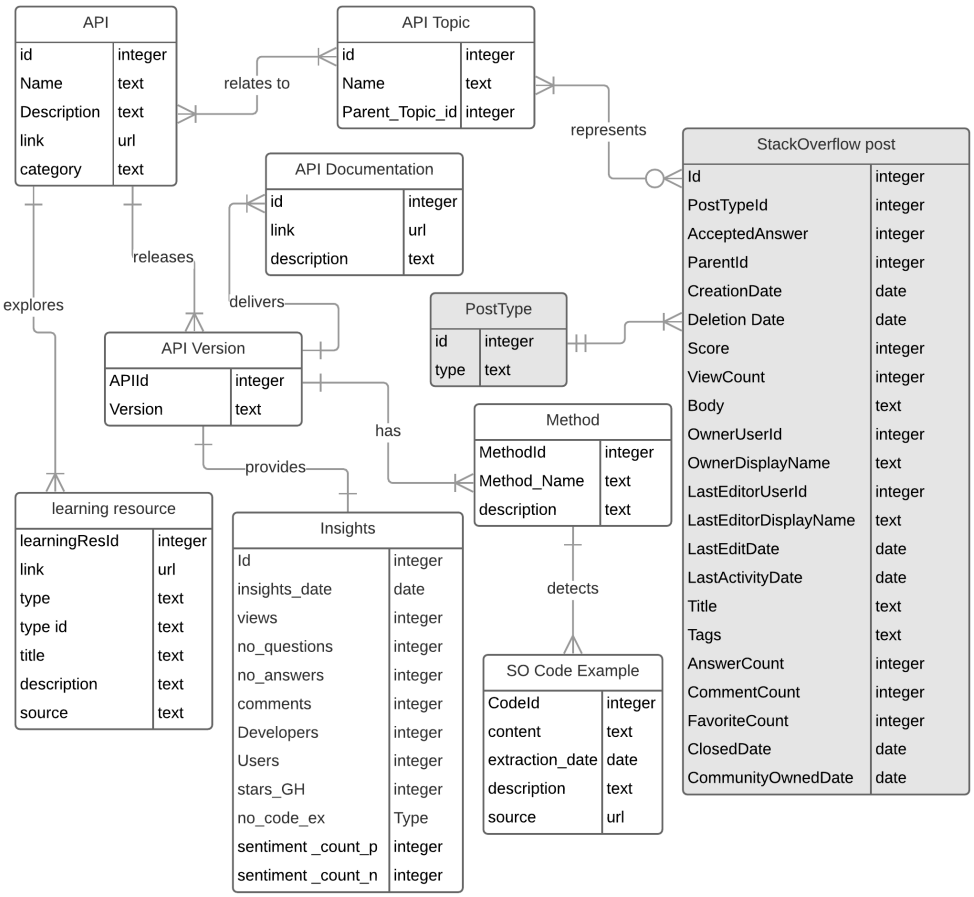

Fig. 1. Data model used for the representation of learning resources.

```
GET /2.2/search?pagesize=100&order=desc&sort=activity&tagged=
    winapi&site=stackoverflow
```

Fig. 2. Example of SO's search API to query for posts related to WinAPI.

not suitable for a topic-driven search of posts related to APIs. In this section, we report on how we leverage Elasticsearch[13] and the data model introduced before for indexing APIs on a topic-centric basis.

Elasticsearch is an open-source search engine that provides near real-time search services. It is built upon an open-source information retrieval and indexing library: Apache Lucene[14]. Elasticsearch provides REST APIs and supports the distributed aspects that Apache Lucene lacks. In Elasticsearch's terminology, an index is a container that stores JSON documents (e.g., posts in SO). The schema of an index is defined through so-called mappings[15]. Elasticsearch uses Apache Lucene to generate an inverted list [37] of keywords where each keyword is associated to documents that contain such keyword.

We leverage on Elasticsearch indexing and searching capabilities to support exploration and discovery of API topic issues. In concrete, we index SO posts related to APIs and use the resulting index as a gateway to get access to our KB represented by the model in Figure 1. The index follows the mapping (i.e., schema) shown in Figure 3. In this mapping, the *title*, *body*, *tags* and *question_id* correspond to standard attributes from SO's dataset. Whereas the attributes *api* and *topic* are extended attributes that characterize each post based on the API (e.g., "WinAPI") and topic (e.g., "API Security") the post relates to.

APIs may be referred to using different mentions across different posts. For instance, "Windows API" can be referred to also as "WinAPI" and "Windows 32 API". The same rationale applies to API topic issues. In order to be able to account for different mentions of APIs and API topic issues,

[13]https://www.elastic.co

[14]http://lucene.apache.org

[15]https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html

```
{"mappings":
  {"_stackoverflow":
    {"properties":
      {
        "title":{"type":"text"},
        "body": {"type":"text"},
        "tags": {"type":"keyword"},
        "api": {"type":"keyword"},
        "topic": {"type":"keyword"},
        "question_id":{"type": "integer"}
      }
    }
  }
}
```

Fig. 3. Elasticsearch mapping for indexing SO posts along with the *api* and *topic* mentions they relate to.

we propose to extend the values within the attributes *api* and *topic* (see Figure 3) to include different mentions thereof. We do such enrichment using word embeddings techniques [24], where words (from a given corpus) are mapped to a vector space. A key property here is that words that are semantically similar appear close to each other in the vector space. Several pre-trained models exist today [24], [36], [38], including general-purpose embeddings (e.g., based on Wikipedia corpus) and domain-specific ones (e.g., programming corpora). We focus on the latter and leverage on a word embedding pre-trained model based on SO [36]. Furthermore, we leverage on previous work on building software engineering thesaurus [39] and API recognition in SO [40] in order to identify API-relevant terms from SO posts. Table I shows examples of SO posts enriched with additional mentions for APIs and API topic issues after the application of the aforementioned techniques.

The resulting index can thus be used to perform searches based on APIs and their associated topic issues. The corresponding queries can be written with terminology flexibility thanks to the enrichments discussed before (e.g., users can write queries related to Windows API by using the terms "WinAPI", "Windows API", etc). In the next section, we elaborate more on how to leverage this index to build a query bot that allows for querying API knowledge based on API topic issues and using our own domain-specific query language.

IV. Scout-bot: API Topics Discovery and Exploration Bot

In this section we introduce *Scout-bot*, a query bot system that enables users to explore and discover APIs through API topic issues. With Scout-bot (see Figure 4), a user writes a query based on a DSL (we will present our API-Topics DSL next) and the bot returns the corresponding results containing related API resources.

In order to do so, Scout-bot first parses the user expression to identify query elements such as API names, API Topics or both, and the boolean operators that may be present in the expression. Then, results from the parser are passed on to a query generator to translate queries written in our API-Topics DSL into Elasticsearch's own DSL. Finally, the translated query is sent to Elasticsearch's search services and the results are presented back to the user. The overall query bot system is depicted in Figure 4. Next, we describe the API-topics DSL we use for writing queries in Scout-bot.

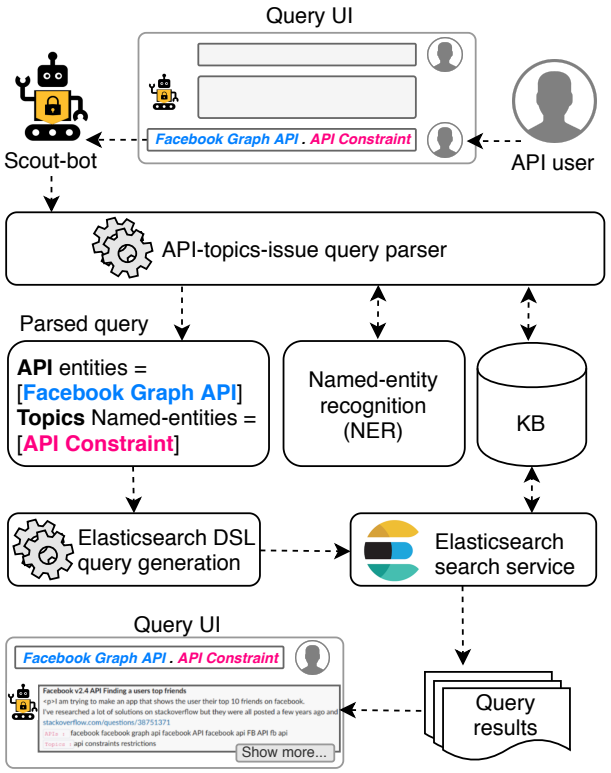

Fig. 4. Overall architecture of Scout-bot.

A. API-Topics DSL: A Domain-specific Query Language for APIs

We provide a simple yet powerful API-topics DSL that allows users to write queries based on the key entities involved in our approach: API names and API topics. We show next the queries allowed by our API-Topics DSL:

API-Name.Topic-Name This query allows users to explore a topic in the context of a given API. It suffices to provide an API and topic name separated by a "dot" (.) operator. For example, *"WinAPI"."API Security"* allows users to explore the topic of API Security in the context of Windows API.

API-Name.Topic-Name contains 'keywords' In this case, the query allows users to explore APIs and topics based on a set of keywords. For example, a user that wants to explore authentication mechanisms in the context of API Security

TABLE I
EXAMPLES OF ELASTICSEARCH'S INDEX ENTRIES INCLUDING SO QUESTIONS (IDENTIFIED BY QID) ALONG WITH THE *api* AND *topic* MENTIONS THEY RELATE TO.

SO QID	title	body	tags	api	topic
9484998	Simple FTP Client authentication?	I have written a simple FTP Client-Server code using WinSocs ...	c++, windows, winapi, authentication, ftp	winapi {win-api, win32-api, windows-api}	authentication {oauth, authorization, auth}
25885369	Windows API function CredUIPromptForWindows-Credentials also returns an error of 31	When I use the function CredUIPromptForWindowsCredentials to display a windows ...	c++, winapi	winapi {win-api, win32-api, windows-api}	security {safety, policies}, authentication {oauth, authorization, auth}
7725464	Windows security center API	I would like to access the Windows Security Center...	c++, windows, security, winapi	winapi {win-api, win32-api, windows-api}	security {safety, policies}

topic for Windows API can write a query like *"WinAPI"."API Security" contains 'Authentication'*.

API-Name.Topic-Name AND API-Name.Topic-Name This expression allows users to explore APIs using a conjunctive query. For example, if a user wants to inquire about overlapping issues of security and debugging under WinAPI, the following query can be used: *"WinAPI"."API Security" AND "WinAPI"."Debugging"* (notice that a post can be categorized under more than one topic issue).

API-Name.Topic-Name OR API-Name.Topic-Name A user can also write disjunctive queries to explore APIs and topic issues. For instance, a user exploring security issues in the context of Github API or GitLab API can write the query *"Github-API"."API Security" OR "GitLab-API"."API Security"*.

Thus, the possibility to express simple queries in terms of API names and topics (with the option to refine queries using the *contains* operator) as well as the possibility of combining them using disjunctive and conjunctive Boolean operators results in a simple query language that enables the construction of arbitrarily complex queries to support exploration and discovery of API knowledge.

B. Implementing and Evaluating Scout-bot

We implemented Scout-bot as a Slack[16] app. Slack is a cloud-based collaboration platform that enables users and developers to interact and communicate with each other as well as with third-party apps. We implemented both the DSL-query parser and the API Topics DSL described previously as microservices using Python Flask[17]. This serves as the webhook that connects our query bot app to our query parser and API-Topics DSL. The query bot interface app and the microservices are deployed on an Ubuntu virtual machine hosted on Google Cloud Compute[18] service. Figure 5 shows a screenshot of Scout-bot in action where, for illustration purposes, we show only SO's main attributes (in practice, Scout-bot can show other elements from our KB model presented in Figure 1 including insights, learning resources, API documentation, among other resources).

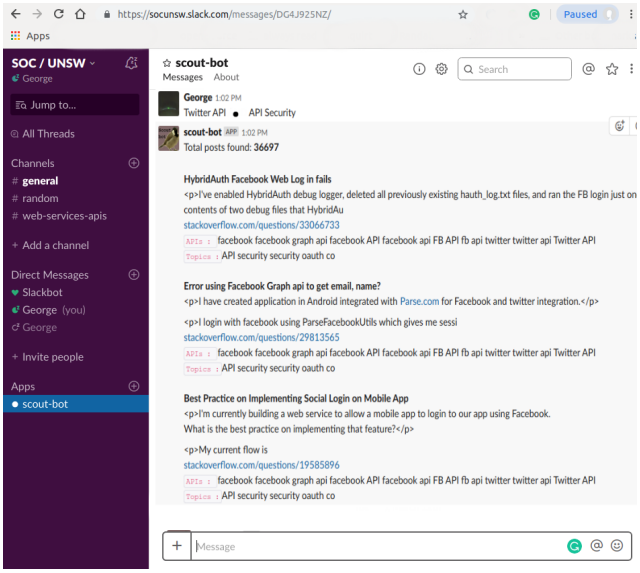

Fig. 5. Implementation of Scout-bot in Slack.

In order to evaluate Scout-bot, we measured its ability to retrieve relevant posts when given a query using our API-Topic DSL. To do so, we extracted, enriched and indexed a sample of approximately 62K API-related posts collected from SO. The rationale of our evaluation consists in sampling posts from these initial 62K posts, building queries that capture the essence of the API and topic being discussed in the post, and

[16]https://slack.com
[17]https://flask.pocoo.org
[18]https://cloud.google.com/compute

manually inspecting the returned results. The recommended minimum number of posts to be sampled was 96, considering a margin of error of 10% and a 95% confidence interval [41]. The actual number of posts sampled were 109, which allowed us to cover a number of API topic issues for at least three different APIs (Windows, Facebook and Youtube APIs). In order to build queries based on our API-Topic DSL we considered the posts' title and question body. Table II shows examples of queries built based on the sampled posts.

TABLE II
API-TOPIC DSL QUERIES BASED ON ACTUAL QUESTIONS (QID) FROM SO.

SO QID	Formulated API-topic DSL query
13115608	facebook.usage contains 'authentication services'
2478391	youtube.usage contains 'get video title jquery'
21329250	facebook.debugging contains 'FB og image pulling'
3566018	winapi.debugging contains 'compile fatal error'
23417356	facebook.usage contains 'permanent login page access token info'
6218325	winapi.usage contains 'check if directory windows exists'
1037595	winapi.usage contains 'user interaction time'
167414	winapi.usage contains 'POSIX system file rename'
940707	winapi.usage contains 'get Win32 native APIs version'

We evaluate the performance of Scout-bot in terms of precision [37]. We focus on how the system performs at returning relevant documents to the query in the exploration search. In this case, we follow the common information retrieval evaluation metrics Precision@K (or P@K) and R-Precision [37]. P@K helps measure the returned results at the top K number of documents. When considering the possibility of exploring more results (beyond the top K ones), it is also useful to report on the R-precision metric, which is calculated as $R\text{-}Precision = r/|rel|$. This metric measures the number of relevant documents returned by the system (r) out of the total number relevant documents ($|rel|$).

After computing the values of P@1, P@3, P@10 and R-Precision for each of the 109 formulated queries (see examples in Table II), we proceeded with computing the mean values for these metrics across all 109 queries. The results show a performance of $\overline{P@1} = 0.99$, $\overline{P@3} = 0.96$, $\overline{P@10} = 0.52$ and $\overline{R-Precision} = 0.98$. The value 0.99 for $\overline{P@1}$ is important because users interacting with Scout-bot can have the most relevant result right in the first returned result in almost all cases. Both $\overline{P@3}$ and $\overline{R-Precision}$ also report relatively high values. In terms of $\overline{P@10}$, however, Scout-bot's performance is rather low. This shortcoming is inherent to the metric P@K when the number of relevant results is (much) lower than K [37]. For example, when the number of relevant results is 1, then the maximum value for $P@10$ is 0.1 (this is why R-Precision is also computed to provide a more comprehensive view on performance).

V. DISCUSSION AND FUTURE WORK

This paper presented an approach to index, explore and discovery API community knowledge in a topic-driven manner. Our approach stems from the needs of the programming community to learn and understand APIs in an increasingly interconnected technology landscape, where APIs are considered first-class citizens. We combine our integrated KB, enriched index and simple yet powerful domain-specific query language into a useful tool, *Scout-bot*, that showcases how our solution can be seamlessly integrated into popular productivity tools such as Slack. Besides the technical advantage of integrating Scout-bot into such productivity tool, our aim was also to showcase that API community knowledge can be brought close to the toolset programmers rely on in their daily tasks, doing it in a manner that facilitates access to such knowledge without the need of switching context (e.g., opening a browser to look for online API documentations or SO Q&As). This helps bring the power of and community knowledge about APIs right into the working environment of API consumers, programmers and practitioners.

This work has its own limitations. Firstly, the API community knowledge used to build our KB is based (and mainly driven by) SO. Yet, the rationale we followed is that of relying on a reputable CQA site (SO) that converges common issues developers face when programming, in general, and, using API, in particular. Thus, by leveraging on SO, we are also leveraging both the wisdom of the crowd and the resulting curated API knowledge. Secondly, the evaluation presented in this paper is focused only on the performance of our approach in terms of (information retrieval) precision. Further and deeper studies are needed where end-users are involved to better understand the implications of our solution in learning and using APIs.

Future directions include leveraging embedding techniques for representing elements of our KB in a vector space (e.g., topics, APIs, Q&As) and the development of novel indexing and querying techniques on top of such embeddings to support semantically richer exploration and discovery. We also plan to validate our approach through studies that involve API users in realistic, production environments.

Acknowledgement. This research was done in the context of the first author's Ph.D. thesis [42] as part of the supervision of his co-authors.

REFERENCES

[1] C. Rodríguez, M. Baez, F. Daniel, F. Casati, J. C. Trabucco, L. Canali, and G. Percannella, "REST APIs: A large-scale analysis of compliance with principles and best practices," in *ICWE'16*. Springer, 2016, pp. 21–39.

[2] P. Chatterjee, M. A. Nishi, K. Damevski, V. Augustine, L. Pollock, and N. A. Kraft, "What information about code snippets is available in different software-related documents? An exploratory study," in *SANER 2017*. IEEE, 2017, pp. 382–386.

[3] J. E. Montandon, H. Borges, D. Felix, and M. T. Valente, "Documenting APIs with examples: Lessons learned with the APIMiner platform," in *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 2013, pp. 401–408.

[4] A. Ahmad, C. Feng, S. Ge, and A. Yousif, "A survey on mining stack overflow: question and answering (Q&A) community," *Data Technologies and Applications*, 2018.

[5] C. Chen and K. Zhang, "Who asked what: Integrating crowdsourced FAQs into API documentation," in *ICSE'14*, 2014, pp. 456–459.

[6] M. A. Saied, O. Benomar, H. Abdeen, and H. Sahraoui, "Mining multi-level API usage patterns," in *2015 IEEE 22nd international conference on software analysis, evolution, and reengineering (SANER)*. IEEE, 2015, pp. 23–32.

[7] R. B. Watson, "Development and application of a heuristic to assess trends in API documentation," in *SIGDOC 2012*, 2012, pp. 295–302.

[8] H. V. Nguyen, H. A. Nguyen, A. T. Nguyen, and T. N. Nguyen, "Mining interprocedural, data-oriented usage patterns in JavaScript web applications," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 791–802.

[9] M. F. Zibran, F. Z. Eishita, and C. K. Roy, "Useful, but usable? Factors affecting the usability of APIs," in *WCRE 2011*, 2011, pp. 151–155.

[10] L. Thominet, "Building foundations for the crowd: Minimalist author support guides for crowdsourced documentation wikis," in *Proceedings of the 33rd Annual International Conference on the Design of Communication*, 2015, pp. 1–10.

[11] M. Kim, D. Cai, and S. Kim, "An empirical investigation into the role of API-level refactorings during software evolution," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 151–160.

[12] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the 22nd International Conference on Program Comprehension*, 2014, pp. 279–290.

[13] R. Watson, "Developing best practices for API reference documentation: Creating a platform to study how programmers learn new APIs," in *2012 IEEE International Professional Communication Conference*. IEEE, 2012, pp. 1–9.

[14] A. Pawlik, J. Segal, H. Sharp, and M. Petre, "Crowdsourcing scientific software documentation: A case study of the NumPy documentation project," *Computing in Science & Engineering*, vol. 17, no. 1, pp. 28–36, 2014.

[15] G. Uddin and M. P. Robillard, "How API documentation fails," *IEEE Software*, vol. 32, no. 4, pp. 68–75, 2015.

[16] S. Sohan, C. Anslow, and F. Maurer, "A case study of web API evolution," in *2015 IEEE World Congress on Services*. IEEE, 2015, pp. 245–252.

[17] M. Kechagia and D. Spinellis, "Undocumented and unchecked: Exceptions that spell trouble," in *MSR'14*, 2014, pp. 312–315.

[18] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming Q&A in Stack Overflow," in *ICSM 2012*, 2012, pp. 25–34.

[19] C. McMillan, M. Grechanik, D. Poshyvanyk, C. Fu, and Q. Xie, "Exemplar: A source code search engine for finding highly relevant applications," *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1069–1087, 2011.

[20] T. Dasgupta, M. Grechanik, E. Moritz, B. Dit, and D. Poshyvanyk, "Enhancing software traceability by automatically expanding corpora with relevant documentation," in *ICSME 2013*. IEEE, 2013, pp. 320–329.

[21] L. Inozemtseva, S. Subramanian, and R. Holmes, "Integrating software project resources using source code identifiers," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 400–403.

[22] A. L. Santos and B. A. Myers, "Design annotations to improve API discoverability," *Journal of Systems and Software*, vol. 126, pp. 17–33, 2017.

[23] C. E. Grant, C. P. George, V. Kanjilal, S. Nirkhiwale, J. N. Wilson, and D. Z. Wang, "A topic-based search, visualization, and exploration system," in *The Twenty-Eighth International Flairs Conference*, 2015.

[24] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[25] D. Orenstein, "Application Programming Interface," *ComputerWorld. https://www.computerworld.com/article/2593623/application-programming-interface.html*, 2000.

[26] C. Rodríguez, D. Schleicher, F. Daniel, F. Casati, F. Leymann, and S. Wagner, "SOA-enabled compliance management: Instrumenting, assessing, and analyzing service-based business processes," *SOCA*, vol. 7, no. 4, pp. 275–292, 2013.

[27] S. Radevski, H. Hata, and K. Matsumoto, "Towards building API usage example metrics," in *SANER 2016*, vol. 1, 2016, pp. 619–623.

[28] J. Kim, S. Lee, S.-W. Hwang, and S. Kim, "Enriching documents with examples: A corpus mining approach," *ACM TOIS*, vol. 31, no. 1, pp. 1–27, 2013.

[29] T. Gvero and V. Kuncak, "Interactive synthesis using free-form queries," in *ICSE 2015*, vol. 2. IEEE, 2015, pp. 689–692.

[30] S. Thummalapenta and T. Xie, "Spotweb: Detecting framework hotspots and coldspots via mining open source code on the web," in *ASE 2008*, 2008, pp. 327–336.

[31] M. Flatt, E. Barzilay, and R. B. Findler, "Scribble: Closing the book on ad hoc documentation tools," *ACM Sigplan Notices*, vol. 44, no. 9, pp. 109–120, 2009.

[32] F. B. Manolache, Y. Li, and O. Rusu, "Automated user documentation system for Android," in *RoEduNet-NER 2015)*. IEEE, 2015, pp. 63–67.

[33] M. Vukovic, V. Salapura, and S. Rajagopal, "Crowd-enabled technical writing services," in *SCC 2013*. IEEE, 2013, pp. 635–642.

[34] F. M. Delfim, K. V. Paixão, D. Cassou, and M. de Almeida Maia, "Redocumenting APIs with crowd knowledge: A coverage analysis based on question types," *Journal of the Brazilian Computer Society*, vol. 22, no. 1, p. 9, 2016.

[35] G. Ajam, C. Rodriguez, and B. Benatallah, "API Topics Issues in Stack Overflow Q&As Posts: An Empirical Study," in *The XLVI Latin American Computing Conference (CLEI2020)*, 2020.

[36] V. Efstathiou, C. Chatzilenas, and D. Spinellis, "Word embeddings for the software engineering domain," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 38–41.

[37] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press, 2008.

[38] S. Mumtaz, C. Rodriguez, B. Benatallah, M. Al-Banna, and S. Zamanirad, "Learning Word Representation for the Cyber Security Vulnerability Domain," in *The 2020 International Joint Conference on Neural Networks (IJCNN 2020)*, 2020.

[39] C. Chen, Z. Xing, and X. Wang, "Unsupervised software-specific morphological forms inference from informal discussions," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 450–461.

[40] M. Seshadri, S. Jayakymar, A. Bansal, and M. G. Gurno, "API Recognition in Stack Overflow Posts," http://madhavanseshadri.com/static/Stackoverflow.pdf.

[41] G. D. Israel, "Determining sample size," University of Florida Cooperative Extension Service, Institute of Food and Agricultural Sciences Extension. https://www.tarleton.edu/academicassessment/documents/Samplesize.pdf, 1992.

[42] G. Ajam, "Quality of Application Programming Interfaces Documentation and Topics Issues Exploration." Ph.D. Thesis, School of Computer Science and Engineering, Faculty of Engineering, UNSW Sydney, 2019.