An Empirical Analysis of GraphQL API Schemas in Open Code Repositories and Package Registries

Yun Wan Kim[1], Mariano P. Consens[1], and Olaf Hartig[2]

[1] University of Toronto, Canada
timyun.kim@mail.utoronto.ca consens@mie.utoronto.ca
[2] Linköping University, Sweden
olaf.hartig@liu.se

Abstract. GraphQL is a query language for APIs that has been increasingly adopted by Web developers since its specification was open sourced in 2015. The GraphQL framework lets API clients tailor data requests by using queries that return JSON objects described using GraphQL Schema. We present initial results of an exploratory empirical study with the goal of characterizing GraphQL Schemas in open code repositories and package registries. Our first approach identifies over 20 thousand GraphQL-related projects in publicly accessible repositories hosted by GitHub. Our second, and complementary, approach uses package registries to find over 37 thousand dependent packages and repositories. In addition, over 2 thousand schema files were loaded into the GraphQL-JS reference implementation to conduct a detailed analysis of the schema information. Our study provides insights into the usage of different schema constructs, the number of distinct types and the most popular types in schemas, as well as the presence of cycles in schemas.

1 Motivation and Approach

The schema of a GraphQL API describes the data and the types of queries supported by the API. An empirical study of the GraphQL schemas used by open source projects, therefore, provides useful information about the characteristics of data interfaces. Currently, there is no comprehensive collection of such schemas or a tool that helps gather schemas from GraphQL APIs. The goal of the work presented in this paper is i) to establish a method to extract schemas into a single collection for analyses and ii) to conduct an empirical analysis of the schemas.

1.1 Data Collection Method

APIs-guru has the most comprehensive list of public GraphQL APIs with links to endpoints and their documentation. By using APIs-guru, combined with manual effort through keyword searching, we collected 67 schemas of distinct APIs. Authentication requirements for most publicly available APIs hindered the efficiency and possible automation of schema extraction. Hence, we decided to take a different approach by extracting schemas from open source repositories from GitHub and used three sources to identify GraphQL repositories.

GitHub API As of June, 2018, there were more than 20,000 repositories on GitHub matching the keyword "graphql" and 2,000 repositories matching the keywords "graphql api".

Libraries.io API Decan et al. [1] explored security vulnerabilities of NPM packages that were dependent on vulnerable packages. Following a similar method, we identified over 37,000 repositories dependent on GraphQL reference implementations.

GHTorrent Archived data of GHTorrent is hosted on Google's Big Query platform. We identified over 5,000 repositories matching the keyword "graphql".

Table 1. Summary of GraphQL repositories identified.

Method	NumRepositories
GitHub API	20,635
Libraries.io API	37,588
GHTorrent	5,188

Table 2. Number of dependent repositories for the most popular implementations.

Package	language	Count
NPM/graphql	JavaScript	12,700
Pypi/graphene	Python	310
Rubygems/graphql	Ruby	470

By using string search for schema for every repository file's full file-path, it was possible to identify exact path of potential schema files and their repository data. Our assumption is that this method returns a considerable portion of actual schemas available such that this portion is representative for the entire population of GraphQL schemas publicly availables. We found that schema files are most often named schema.json, schema.js, and schema.graphql for single-file schemas. For modular schemas, the files are most often separated by types, queries, mutations, and subscriptions but are contained in directories with the name schema or schemas.

After downloading all potential schema files, we tried to load each of them via GraphQL-JS. A successful attempt indicated a valid schema and a failure indicated an invalid schema or an irrelevant file. We identified duplicates through several methods including Levenshtein distance and cosine similarity.

2 Analysis Results

We identified a total of 2,777 valid but non-distinct schemas using the proposed method. 1,880 files were unique JSON-formatted schemas. We also conducted an exhaustive search excluding the "schema" keyword on all GraphQL-related repositories to collect a larger list of 3,949 schemas. The union of the two methods resulted in 4,095 schemas and, by using cosine similarity to filter duplicates, 2,081 schemas were unique. Figure 1 illustrates the number of schemas per source and the overlap of sources. This illustration shows that the different approaches to collect GraphQL schemas are non-redundant.

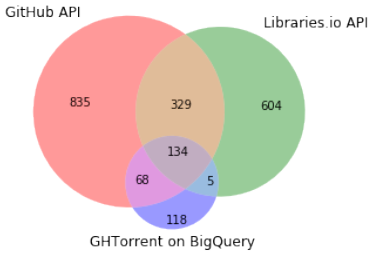

Fig. 1. Number of schemas by sources.

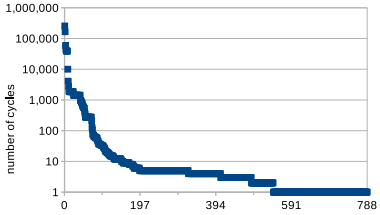

Fig. 2. Number of cycles per schema.

To estimate our recall, we downloaded all .json and .graphql files from all repositories found with the keyword "graphql". By using the 3,949 valid schema counts, the estimated recall of our method is ca. 70% and the precision is 1.8%.

There are five major components of GraphQL schemas that describes the supported operations: Query, Object, Mutation, Subscription, and Directive. While every GraphQL server needs to support queries, which fetch information about data objects, other operations are not necessarily required. Only about 20% of the schemas have the Subscription type that can push information, while about 70% have the Mutation type via which the stored data can be changed.

Table 3. Number of non-empty components in the 2,081 schemas.

Schema components	Frequency
object types	2,079
query type	2,079
directives	2,059
mutation type	1,440
subscription type	414

Table 4. The ten most common object types.

Object type	Frequency
Node	1,009
PageInfo	922
User	879
UserConnection	336
UserEdge	307
BatchPayload	220
Viewer	215
UserPreviousValues	190
Post	182

Object types dictate what information is exchanged between the users and the servers. We find that even after excluding scalar types and type definitions such as Query and Mutation, the most common types are generic types affiliated with reference implementations as shown in Table 4. Node is a reserved interface type for reference implementations such as Apollo and Relay with an identifier field and is the most common.

We traversed each schema in its JSON format recursively to identify their levels of nesting. We find that the median number of levels is 9 and the median number of levels only considering object types is 6. Excluding introspection and scalar type definitions, most schemas have only one level of nesting.

3 Cycles in GraphQL Schemas

Another interesting question is whether the relationships between the types in the schemas form directed cycles, because only if such cycles exist, the data

exposed via a GraphQL API may contain directed cycles and these, in turn, may cause an undesired overhead during query processing [2].

Hence, we analyze *GraphQL schemas as directed graphs*. The vertices in such a graph for a given schema correspond to the object types, the interface types, and the union types in the schema. For every field definition whose value type is based on one such type, the graph contains an edge from the vertex that represents the type in which the field definition appears to the vertex that represents the value type of the field definition. Additionally, there are edges from interface types to their implementing object types and, similarly, from union types to their participating object types. In this paper we focus only on *simple cycles*; that is directed cycles in which repetition of vertices is not allowed.

For the analysis we use a program[3] that loads a schema, generates the corresponding graph representation of this schema, and then enumerates the simple cycles in the generated graph. For the latter step, the program applies a combination of Johnson's algorithm [3] to enumerate the cycles and Tarjan's algorithm [4] to first divide the graph into its strongly connected components, which is a prerequisite of Johnson's algorithm. To run the program for each of the 2,094 schemas we use an ordinary desktop computer with 8 GB of RAM.

We find that 832 of the 2,094 schemas (39.7%) contain at least one simple cycle. For a more detailed analysis of these cycles we can, unfortunately, focus only on 788 of the 832 schemas; the other 44 schemas contain so many simple cycles (at least 10M in each of them) that enumerating these cycles causes the program to crash with an out-of-memory exception.

The distribution of the number of cycles in the remaining 788 schemas is illustrated in Figure 2. As can be observed, the distribution resembles a power law. In more detail, 2 schemas contain more than 100K cycles (that is 0.3% of the 788 schemas), where the maximum is 256,348 cycles; 9 schemas contain more than 10K cycles (that is 1.1%); 41 schemas contain more than 1K cycles (5.2%); 73 contain more than 100 cycles (9.3%); 152 contain at least 10 cycles (19.3%), and 543 contain more than one cycle (68.9%). Hence, 31.1% contain exactly one cycle only.

Moreover, the average length of all cycles within each schema ranges from 2.0 to 20.5, but there is no correlation between this average length and the number of cycles. Similarly, we do not find a correlation between the number of cycles and the number of vertices or edges.

4 Concluding Remarks

This preliminary report describes our approach to collect and analyze thousands of GraphQL schemas from open project repositories. Initial descriptive and structural properties of the collected schemas were presented. The collection has also enabled additional analysis (not included in this contribution) such as temporal characteristics of repository commits and co-committer relationships.

[3] https://github.com/LiUGraphQL/graphql-schema-cycles

Acknowledgements. The authors thank Jonas Lind and Kieron Soames who, as part of their thesis project at Linköping University, have developed the cycle enumeration program and applied it to our collection of schemas. Olaf Hartig's work on this paper has been funded by the CENIIT program at Linköping University (project no. 17.05).

References

1. Decan, A., Mens, T., Constantinou, E.: On the impact of security vulnerabilities in the npm package dependency network. In: Proceedings of the 15th International Conference on Mining Software Repositories. pp. 181–191. MSR '18 (2018), http://doi.acm.org/10.1145/3196398.3196401
2. Hartig, O., Pérez, J.: Semantics and Complexity of GraphQL. In: Proceedings of the 2018 World Wide Web Conference. pp. 1155–1164. WWW '18, Republic and Canton of Geneva, Switzerland (2018), https://doi.org/10.1145/3178876.3186014
3. Johnson, D.B.: Finding all the elementary circuits of a directed graph. SIAM J. Comput. **4**(1), 77–84 (1975). https://doi.org/10.1137/0204007, https://doi.org/10.1137/0204007
4. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972), https://doi.org/10.1137/0201010