Markdown Autogeneration from a GraphQL Schema

Author:
Alessandro Buonerba

Supervisor:
Dr Konstantin Kapinchev

A journal log to help me build the
references and discussions for my dissertation
BSc (Hons) Computer Science (Cybersecurity)

Department of Computing & Mathematical Sciences
Liberal Arts & Sciences

University of Greenwich
London, United Kingdom

October 2021

1

AN EMPIRICAL STUDY OF GRAPHQL SCHEMAS

1.1 SYNOPSIS

GraphQL is a query language and thereupon-based paradigm for implementing web Application Programming Interfaces (APIs) for client-server interactions. Using GraphQL, clients define precise, nested data-requirements in typed queries, which are resolved by servers against (possibly multiple) backend systems, like databases, object storages, or other APIs. Clients receive only the data they care about, in a single request. However, providers of existing REST(-like) APIs need to implement additional GraphQL interfaces to enable these advantages. We here assess the feasibility of automatically generating GraphQL wrappers for existing REST(-like) APIs. A wrapper, upon receiving GraphQL queries, translates them to requests against the target API. We discuss the challenges for creating such wrappers, including dealing with data sanitation, authentication, or handling nested queries. We furthermore present a prototypical implementation of OASGraph. OASGraph takes as input an OpenAPI Specification (OAS) describing an existing REST(-like) web API and generates a GraphQL wrapper for it. We evaluate OASGraph by running it, as well as an existing open source alternative, against 959 publicly available OAS. This experiment shows that OASGraph outperforms the existing alternative and is able to create a GraphQL wrapper for $$89.5\%$$of the APIs — however, with limitations in many cases. A subsequent analysis of errors and warnings produced by OASGraph shows that missing or ambiguous information in the assessed OAS hinders creating complete wrappers. Finally, we present a use case of the IBM Watson Language Translator API that shows that small changes to an OAS allow OASGraph to generate more idiomatic and more expressive GraphQL wrappers.

1.2 USEFUL QUOTES

Clients frequently receive unneeded data, or have to chain multiple requests to obtain desired results (Wittern, Cha, and Laredo, 2018).

Another issue is that REST(-like) APIs amass endpoints when providers add new capabilities to an API to preserve compatibility — for example to react to new client requirement without breaking compatibility with existing clients (Wittern, Cha, and Laredo, 2018).

A GraphQL client introspects a server's schema, i.e., queries it with GraphQL queries, to learn about exposed data types and possible operations. The GraphiQL5 online-IDE uses introspection to allow developers to familiarize with GraphQL schemas (Wittern, Cha, and Laredo, 2018).

The Open API Specification (OAS), formerly known as Swagger, is "a standard, programming language-agnostic interface description for REST APIs" [10]. OAS is a format used by providers to describe and document APIs in an organized and predictable manner (Wittern, Cha, and Laredo, 2018).

1.3 PERSONAL REFLECTION

This paper will come very handy for when I have to explain the principal reasons of why users would like to create or refactor their REST-like APIs to GraphQL. The paper also point out that a GraphQL client can introspects a server schema but fails to state that this must be enabled and that usually is disabled to prevent anyone to see and fully inspect all the resources available in the current API schema. With an introspection it is possible to see queries, types, fields and all the directives supported by the API. Another good point and a reason why my proposed title and project is even more valuable, is that since introspection can be dangerous, it is very often (almost by default) disabled. Meaning that is not possible to utilise GraphQL Voyagers or swagger alternative that supports GraphQL. In other words, there is no way to see what's the schema is like or any description that comes with it. My project will most likely solve and help teams or individual that do not want to enable introspection.

1.4 QUESTION RAISED

What is the main reason why users want to create or refactor their REST-like APIs to GraphQL? Is it important to keep introspection on? What will the users lose if they disable it? How would you document a GraphQL API since it doesn't have Swagger?

1.5 NOTES

Important things to take are introspections, why do developers want to disable it and how does it make a documentation tooling a necessity for a team.

2

GENERATING GRAPHQL-WRAPPERS FOR REST(-LIKE) APIS

2.1 SYNOPSIS

GraphQL is a query language for APIs and a runtime to execute queries. Using GraphQL queries, clients define precisely what data they wish to retrieve or mutate on a server, leading to fewer round trips and reduced response sizes. Although interest in GraphQL is on the rise, with increasing adoption at major organizations, little is known about what GraphQL interfaces look like in practice. This lack of knowledge makes it hard for providers to understand what practices promote idiomatic, easy-to-use APIs, and what pitfalls to avoid.

To address this gap, we study the design of GraphQL interfaces in practice by analyzing their schemas — the descriptions of their exposed data types and the possible operations on the underlying data. We base our study on two novel corpuses of GraphQL schemas, one of 16 commercial GraphQL schemas and the other of 8,399 GraphQL schemas mined from GitHub projects. We make available to other researchers those schemas mined from GitHub whose licenses permit redistribution. We also make available the scripts to mine the whole corpus. Using the two corpuses, we characterize the size of schemas and their use of GraphQL features and assess the use of both prescribed and organic naming conventions. We also report that a majority of APIs are susceptible to denial of service through complex queries, posing real security risks previously discussed only in theory. We also assess ways in which GraphQL APIs attempt to address these concerns.

2.2 USEFUL QUOTES

Clients send queries that precisely define the data they wish to retrieve or mutate (Wittern et al., 2019).

GraphQL prescribes a statically typed interface, which drives developer tooling like GraphiQL, an online IDE helping developers explore schemas and write and validate queries, or type-based data mocking for testing services (Wittern et al., 2019).

2.3 PERSONAL REFLECTION

This paper, differently from the previous one goes more into depth with the schema and the language definition. Since to generate markdown from a schema, we must firstly analyse the schema, it will be a useful paper for when I have to explain the references for data types.

2.4 QUESTION RAISED

2.5 NOTES

The article conclusion point out that there are concerns about denial of service with large APIs that could have nested queries. This could be easily solved with a solution for the N+1 problem and limiting the rate and depth limiting at the edge. New modern CDN can also wrap the APIs and provide an extremely useful layer of protection, caching and lower response rate for the API.

3

THE EXPLOITATION OF OPENAPI DOCUMENTATION FOR THE GENERATION OF WEB FRONTENDS

3.1 SYNOPSIS

New Internet-enabled devices and Web services are introduced on a daily basis. Documentation formats are available that describe their functionalities in terms of API endpoints and parameters. In particular, the OpenAPI specification has gained considerable influence over the last years. Web-based solutions exist that generate interactive OpenAPI documentation with HTML5 & JavaScript. They allow developers to quickly get an understanding what the services and devices do and how they work. However, the generated user interfaces are far from real-world practices of designers and end users. We present an approach to overcome this gap, by using a model-driven methodology resulting in state-of-the-art responsive Web user interfaces. To this end, we use the Interaction Flow Modeling Language (IFML) as intermediary model specification to bring together APIs and frontends. Our implementation is based on open standards like Web Components and SVG. A screencast of our tool is available at https: //youtu.be/KFOPmPShak4

3.2 USEFUL QUOTES

Swagger UI is an open source software that automatically generates a HTML5-based visualization and interaction frontend from an OpenAPI file (Koren and Klamma, 2018).

Web Components are a recent manifestation of the increased adoption of component-based software engineering in the frontend Web. Similar approaches on a JavaScript framework level are Ember, React and Vue.js (Koren and Klamma, 2018).

We strongly believe that automation is one of the keys to tackle challenges in the creation of situational applications for the long tail (Koren and Klamma, 2018).

3.3 PERSONAL REFLECTION

This paper will be very useful to have a metric of comparison between OpenAPI and Swagger and the fact that GraphQL doesn't support it. Especially, as previously stated, when introspection is disabled then nothing would be available to document the schema. Another point is that even if GraphQL would support Swagger UI, it would still be locked behind proprietary code without any flexibility or power over the presentation and what to show or hide. This will be one of the strong point of my project, since the files generated will be markdown that can be parsed and manipulated as per developer liking.

3.4 QUESTION RAISED

3.5 NOTES

Make this one of the strong points.

4

FRAMEWORK STUDY FOR AGILE SOFTWARE DEVELOPMENT VIA SCRUM AND KANBAN

4.1 SYNOPSIS

This paper provides a systematic comparison between two well-known Agile methodologies: Scrum, which is a framework of doing projects by allocating tasks into small stages called sprints, and Kanban, which is a scheduling system to manage the flow of work by means of visual signals. In this regard, both methodologies were reviewed to explore similarities and differences between them. Then, a focus group survey was performed to specify the preferable methodology for product development according to various parameters in the project environment including project complexity, level of uncertainty, and work size with consideration of output factors like quality, productivity, and delivery. Results show the flexibility of both methodologies in approaching Agile objectives, where Scrum emphasizes on the corporation of the customer and development teams with a focus on particular skills such as planning, organization, presentation, and reviewing which makes it ideal for new and complex projects where a regular involvement of the customer is required, whereas Kanban is more operative in continuous-flow environments with a steady approach toward a system improvement.

4.2 USEFUL QUOTES

Scrum is a project management approach that works according to iterative and incremental developments (Zayat and Senvar, 2020).

The stages of Scrum model start with product backlog creation as the first stage (Zayat and Senvar, 2020).

Nowadays, Scrum and Kanban methodologies are the most adopted methods by system devel opment organizations around the world (Zayat and Senvar, 2020).

Scrum is defined as a framework in which people can deal with complex problems, while maintaining high productivity in delivering high-quality products (Zayat and Senvar, 2020).

Kanban, which means visual signal, was firstrst used by workers in Toyota to track processes on their manufacturing system (Zayat and Senvar, 2020).

The main focus of Kanban is the flow of work along with the elimination of unnecessary activities which leads to shorter feedback loops (Zayat and Senvar, 2020).

4.3 PERSONAL REFLECTION

While working on my project, I will need a good agile framework to split the work in epics and evaluate what needs to be researched through spikes and how many stories are needed to complete each epic. Even though Scrum could be a nice addition to the workflow, it is most likely aimed to a group of at least two. I will have to decide if I want to simulate being in a small group or use Kanban instead avoiding unnecessary cerimonies that Scrum would lock me in.

4.4 QUESTION RAISED

How would I do standups? Would I need to do retros fortnightly? Who would refine my tickets and point my stories?

4.5 NOTES

I have to think about which Agile framework to use. The questions that are previously made will probably lead me to use Kanban to work more efficiently, but this paper is nice to evaluate and choose the methodology that will follow me during this project.

5

JOURNAL 5

5.1 SYNOPSIS

In modern environment, delivering innovative idea in a fast and reliable manner is extremely significant for any organizations. In the existing scenario, Insurance industry need to better respond to dynamic market requirements, faster time to market for new initiatives and services, and support innovative ways of customer interaction. In past few years, the transition to cloud platforms has given benefits such as agility, scalability, and lower capital costs but the application lifecycle management practices are slow with this disruptive change. DevOps culture extends the agile methodology to rapidly create applications and deliver them across environment in automated manner to improve performance and quality assurance. Continuous Integration (CI) and Continuous delivery (CD) has emerged as a boon for traditional application development and release management practices to provide the capability to release quality artifacts continuously to customers with continuously integrated feedback. The objective of the paper is to create a proof of concept for designing an effective framework for continuous integration, continuous testing, and continuous delivery to automate the source code compilation, code analysis, test execution, packaging, infrastructure provisioning, deployment, and notifications using build pipeline concept.

5.2 USEFUL QUOTES

Application development phases include coding, building, integrating, testing, troubleshooting, infrastructure provisioning, configuration management, setting up runtime environment, and deploying applications in different environments (Soni, 2015).

Continuous Delivery is a software development discipline where you build a deployment pipeline in such a way that the software can be released to production at any time (Soni, 2015).

Quality gates helps to manage orchestration across build pipeline. Ideally, if one step fails to execute successfully then continuous delivery or next step must not take place (Soni, 2015).

Every commit into source code repository by a developer is verified by automated build process using continuous integration server (Soni, 2015).

Rapid delivery in Dev, Test, Pre-production and Production environments: Elimination of manual and repeatable processes based deployment (Soni, 2015).

5.3 PERSONAL REFLECTION

This is an important step for my project. Since one of the requirements is about evaluating if the final product is successfull or not it will require end to end testings and a good pipeline in a ci/cd workflow.

5.4 QUESTION RAISED

How many envoirments do I want in my pipeline? How many steps? Should I consider pre-hooks with husky for linting and more? E2E and Unit? Should I use Vercel for CD?

5.5 NOTES

Need to come up with a good architecture and a plan to avoid messy pipelines.

BIBLIOGRAPHY

Koren, István and Ralf Klamma (2018). "The Exploitation of OpenAPI Documentation for the Generation of Web Frontends". In: *Companion Proceedings of the the Web Conference 2018*. International World Wide Web Conferences Steering Committee. ISBN: 978-1-4503-5640-4. DOI: 10.1145/3184558.3188740. URL: https://doi.org/10.1145/3184558.3188740.

Soni, Mitesh (2015). *End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery*. ISBN: 978-1-4673-8566-4. URL: https://ieeexplore.ieee.org/document/7436936 (visited on 10/06/2021).

Wittern, Erik, Alan Cha, James C. Davis, Guillaume Baudart, and Louis Mandel (2019). "An Empirical Study of GraphQL Schemas". In: *Service-Oriented Computing*. Ed. by Sami Yangui, Ismael Bouassida Rodriguez, Khalil Drira, and Zahir Tari. Cham: Springer International Publishing, pp. 3–19. ISBN: 978-3-030-33702-5. URL: https://link.springer.com/chapter/10.1007/978-3-319-91662-0_5.

Wittern, Erik, Alan Cha, and Jim A. Laredo (2018). "Generating GraphQL-Wrappers for REST(-like) APIs". In: *Web Engineering*. Ed. by Tommi Mikkonen, Ralf Klamma, and Juan Hernández. Cham: Springer International Publishing, pp. 65–83. ISBN: 978-3-319-91662-0.

Zayat, Wael and Ozlem Senvar (June 2020). "Framework Study for Agile Software Development Via Scrum and Kanban". In: *International Journal of Innovation and Technology Management* 17.04, p. 2030002. ISSN: 0219-8770. DOI: 10.1142/S0219877020300025. URL: https://www.worldscientific.com/doi/epdf/10.1142/S0219877020300025 (visited on 10/06/2021).