An OpenAPI-based Testing Framework to Monitor Non-Functional Properties of REST APIs*

Steven Bucaille[1][0000−0003−1997−3753], Javier Luis Cánovas Izquierdo[2][0000−0002−2326−1700], Hamza Ed-douibi[2][0000−0003−4342−4818], Jordi Cabot[2,3][0000−0003−2418−2489]

[1] Katholieke Universiteit Leuven, Belgium
steven.bucaille@student.kuleuven.be
[2] UOC. Barcelona, Spain
{jcanovasi,hed-douibi}@uoc.edu
[3] ICREA. Barcelona, Spain
jordi.cabot@icrea.cat

Abstract. REST APIs have become key assets for any company willing to have online presence and provide access to its services. Several approaches have been proposed to describe this kind of APIs, being OpenAPI the dominant proposal in the last years. OpenAPI allows any consumer to understand the operations and data elements of a REST API. However, it does not cover any kind of non-functional properties, such as performance and availability. In this paper we present Gadolinium, a framework that leverages the OpenAPI specification to test non-functional properties of REST APIs. Gadolinium automatically tests performance and availability in different geographical locations by means of a master/slave architecture. The results of the test can eventually be injected in the original OpenAPI definition of the REST API.
Demo: http://hdl.handle.net/20.500.12004/1/C/ICWE/2020/001

1 Introduction

The Web has become the main source of information and services for both developers and big companies. Nowadays the most popular way to access this information is via REST APIs. REST APIs have been usually documented in natural language only, which hampers its understanding and use. In the last years a number of specifications have appeared to formalize the definition of APIs and solve this problem. OpenAPI is now the *de facto* standard for this.

OpenAPI provides a specification language to describe the operations and data structures of REST APIs. OpenAPI covers the functional and actionable elements of a REST API, however, it does not support Non-Functional Properties (NFPs) like performance or availability, which are crucial to help developers choose and integrate the most suitable API for their applications.

* Work supported by the Spanish government (TIN2016-75944-R project)

In this demo paper we present GADOLINIUM, a framework that relies on OpenAPI to automatically test NFPs of REST APIs. The framework provides data schemas to describe NFPs and the required testing process, which relies on a master/slave architecture. The results of the test can eventually be stored in the OpenAPI description to enrich API information and make sure it is even more helpful for future developers evaluating its adoption. Our current implementation covers the test of performance and availability NFPs, and supports the deployment of clients in the Google Cloud platform.

To the best of our knowledge, ours is the first general approach to automatically test NFPs in OpenAPI. While some works have explored the definition of NFPs in Web development (e.g., [2, 4, 3]) and others have studied how to benchmark quality aspects in Web APIs (e.g., [1]), none of them mix the study and testing of NFPs in OpenAPI. Only some commercial tools (e.g., SOAP UI[4]) propose NFP testing for REST APIs but mostly focusing on load testing.

2 Our Proposal

We propose a framework called GADOLINIUM that relies on the OpenAPI description of REST APIs to test NFPs of their operations.

Our proposal currently supports two NFPs: performance and availability. Others can be added following a similar approach to the one explained herein. Performance is measured by calculating the latency or time interval between a request and the response. Availability is measured via the API uptime (i.e., percentage of time the API is ready to receive requests). We use random values for mandatory parameters of the requests and omits values for optional ones.

Both properties should be evaluated considering that APIs can be transparently replicated in different locations and therefore users can access them from diverse geographical places. As such, NFPs values can change on a per geographical basis. To deal with this, GADOLINUM follows a master/slave architecture where slaves are geographically distributed and deployed in different locations to ensure a good coverage of the test.

Figure 1 illustrates our proposal. As can be seen, GADOLINIUM takes as input the OpenAPI description to be tested and monitored. Once the OpenAPI description is loaded, the user configures the testing process. At that point, the testing process launches several slaves to test the NFPs and report back the results. The master element of this architecture controls the slaves, monitors the sequence of events and displays a dashboard to the user summarizing the status of the testing process and its results. The user can then review and analyze the results, which can also be exported into the OpenAPI description provided initially using the standard extension mechanism of the OpenAPI specification.

Figure 2 shows an example of using GADOLINIUM. Figure 2a shows the importation dialog, where the user provides the OpenAPI description and configure the process. The configuration involves (1) setting the number of times the API

[4] https://www.soapui.org/

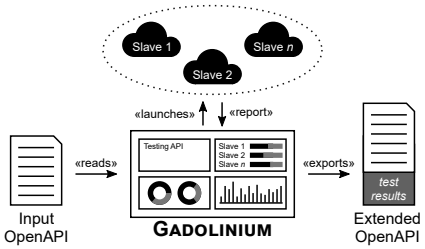


Fig. 1: General overview of GADOLINIUM.

will be tested, (2) the time between tests and (3) the geographical zones to deploy the slaves for each NFP. Figure 2b shows an example of the results page. On top, it shows the importation and slaves execution data, including the progress until reaching the final stage. At the bottom, it shows the results of the uptime (on the left) as a pie chart and latency (on the right) as a bar chart that can be filtered according to either operations or geographical zones.

3 Architecture

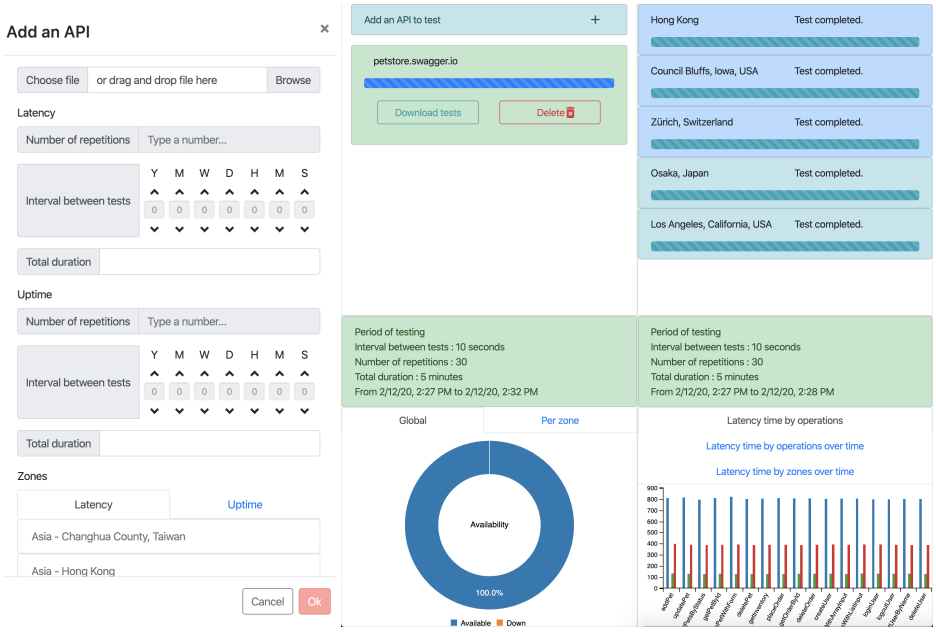
This section provides some more details on the architecture and implementation of GADOLINIUM. As we described above, the two key components are the master and the slaves. While the master can be deployed anywhere, slaves must be physically distributed and deployed in different locations of the world to ensure a good coverage for the NFPs tested. Next we describe the implementation of both master and slaves.

Master The Master is the central piece of GADOLINIUM and provides a dashboard to import OpenAPI files, monitor the APIs being tested and download results. The backend has been developed in NODEJS, providing an HTTP server for the frontend and a communication channel via SOCKETIO for slaves. The frontend has been developed in ANGULAR, allowing the user to provide an OpenAPI description and configure the testing process.

Slaves A slave is created to test a specific non-functional property of a REST API from a location. The lifecycle of a slave includes its deployment, configuration, connection to the Master to get the instructions (i.e., NFP and API to test), test execution and send back the results. Slaves have been developed as independent NODEJS applications running on Google's data centers.

4 Conclusion

We have presented GADOLINIUM, a framework to test and monitor NFPs of REST APIs by leveraging the OpenAPI specification. The approach currently supports testing and monitoring latency and uptime NFPs and provides a dashboard view to control the complete lifecycle of the testing process. GADOLINIUM



(a) (b)

Fig. 2: Example of dashboard in GADOLINIUM. (a) Adding an API and configuring the NFP metrics. (b) Results of the testing process.

has been made available on GitHub[5], where additional information about its inner workings can be found.

As further work, we plan to support additional NFPs (e.g., throughput and reliability) as well as other cloud platforms to improve the geographical coverage. We are also interested in exploring new visualization techniques to help developers study how structural properties (e.g., the size or structure of the payload of the operations) may affect the NFPs.

References

1. Bermbach, D., Wittern, E.: Benchmarking Web API Quality. In: Int. Conf. in Web Engineering. pp. 188–206 (2016)
2. Galster, M., Bucherer, E.: A Taxonomy for Identifying and Non-Functional Requirements in Service-Oriented Development. In: IEEE Congress on Services. pp. 345–352 (2008)
3. Junghans, M., Agarwal, S.: Web Service Discovery Based on Unified View on Functional and Non-functional Properties. In: Int. Conf. on Semantic Computing. pp. 224–227 (2010)
4. Ortiz, G., Núñez, J.H., Clemente, P.J.: How to Deal with Non-functional Properties in Web Service Development. In: Int. Conf. in Web Engineering. pp. 98–103 (2005)

[5] http://hdl.handle.net/20.500.12004/1/A/GADOLINIUM/001