Towards the Definition of Test Coverage Criteria for RESTful Web APIs

Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés

Department of Computer Languages and Systems
Universidad de Sevilla, Spain
{amarlop,sergiosegura,aruiz}@us.es

Abstract. Web APIs following the REST architectural style (so-called RESTful Web APIs) have become the de-facto standard for software integration. As RESTful APIs gain momentum, so does the testing of them. However, there is a lack of mechanisms to assess the adequacy of testing approaches in this context, which makes it difficult to measure and compare the effectiveness of different testing techniques. In this work-in-progress paper, we take a step forward towards a framework for the assessment and comparison of testing approaches for RESTful Web APIs. To that end, we propose a preliminary catalogue of test coverage criteria. These criteria measure the adequacy of test suites based on the degree to which they exercise the different input and output elements of RESTful Web services. To the best of our knowledge, this is the first attempt to measure the adequacy of testing approaches for RESTful Web APIs.

Keywords: REST · testing · web services · coverage criteria.

1 Introduction

REpresentational State Transfer (REST) has become the preferred architectural style for developing Web Application Programming Interfaces (APIs), so-called *RESTful APIs*. Most testing approaches for RESTful Web APIs and tools follow a black-box approach, where test cases are derived from the specification of the Web services that compose the API [1, 3, 5]. However, to the best of our knowledge, there is no standard way to assess the adequacy of these types of black-box approaches, and so it is hard to measure and compare the effectiveness of different test suites and techniques. Previous works have addressed the formalisation of black-box coverage criteria as a means to validate their testing techniques, but none of them have proposed a common framework for the assessment and comparison of multiple approaches.

In this paper, we take a first step towards the definition of a framework for the assessment and comparison of testing approaches for RESTful Web APIs. In particular, we propose a preliminary catalogue of test coverage criteria for RESTful APIs divided into two groups: inputs (i.e. service request) and outputs

(i.e. service response). To this end, we took inspiration from the OpenAPI Specification (OAS) [4] and previous works which have made an effort to somehow evaluate the efficacy of their black-box testing approaches [2, 3]. This catalogue aims to serve as a starting point for the definition of a test coverage model, where the proposed criteria will be arranged into different levels. This will yield a framework for the assessment and comparison of testing approaches for RESTful APIs based on the coverage levels that they can reach.

2 RESTful Web APIs

A RESTful API is composed of several Web services, each of them implementing one or more create, read, update and delete (CRUD) operations over a specific resource. These operations are usually mapped to the HTTP methods POST, GET, PUT and DELETE, respectively. A *resource* is any type of information that can be exposed to the Web (e.g. a photo, a HTML document, information about a book); for instance, a *playlist* is a resource in the Spotify API [6]. Resources are addressable by a unique Uniform Resource Identifier (URI). A *path* (also called *route* or *endpoint*) represents a resource over which operations can be performed. The term *operation* itself refers to the use of one of the four HTTP verbs over a specific path. Below there are some examples of operations on different resources of the Spotify API.

`GET /search?q=rhapsody&type=track` Search for songs with *rhapsody*.

`POST /users/{user_id}/playlists` Create a playlist.

`PUT /playlists/{playlist_id}` Update an existing playlist.

`DELETE /playlists/{playlist_id}/tracks` Remove tracks from a playlist.

Furthermore, operations accept parameters. A *parameter* is a piece of information that can be passed together with the request for several purposes, such as filtering and sorting results (e.g. `q` and `type` in the first operation).

Several expected responses can be specified for every operation (i.e. request). A *response* is identified by the returned status code and can optionally include a body. The *status code* determines the result of the operation (e.g. whether it has been successful or not) and the *response body* includes additional information (e.g. when retrieving a given resource). For instance, "Search for songs" in the previous example could return a 404 status code if no results were found or a 200 together with a set of results in the response body, matching the filters passed in the request.

3 Test coverage criteria

We propose classifying the coverage criteria into two types: input (those addressing API requests) and output (API responses). All criteria described here can be applied to measure test coverage on the whole API or subsets of it. For instance, it is possible to measure how many parameters have been covered on a single operation, on all operations of a single path, or on all paths, that is, the entire API.

3.1 Input coverage criteria

Path coverage. This criterion refers to the coverage of the API's paths. In the case of the Spotify API, it has 53 paths, so at least 53 test cases (i.e. requests) are needed to reach 100% coverage.
Operation coverage. It measures the coverage of the HTTP methods that can be applied on each path. This criterion can be applied to a single path or to the whole API. For instance, to fully cover this criterion on the path `/playlists/{playlist_id}/tracks`, three test cases must be executed, since this path accepts three methods: GET, PUT and DELETE.
Parameter coverage. All operation parameters are liable to be used when testing the API. This criterion measures the percentage of parameters covered by a test suite. Consider, for example, the `GET /search` operation from the Spotify API: it accepts seven parameters, therefore all of them must be submitted at least once in order to fulfill this criterion. Stricter criteria could require certain combinations of parameters to be tested.
Parameter value coverage. It concerns the testing of multiple values for every parameter, mainly for *booleans* and *enums*, that is, those with a finite set of values. Nevertheless, testing as many values as possible with the other types, e.g. *strings*, *integers* and so on, is something worth considering. This criterion can be applied to a single parameter or to the whole API. For the `type` parameter from the previous operation, it accepts four possible values (`album`, `artist`, `playlist` and `track`), so all them should be tested to reach 100% coverage of this criterion on that specific parameter.
Content-type coverage. All allowed input data formats such as JSON and XML must be tested for all operations that accept a request body (e.g. POST operations). This criterion measures the percentage of input data formats covered by a test suite. The Spotify API consumes only JSON, therefore no other data formats need to be tested.

3.2 Output coverage criteria

Status code class coverage. It refers to the coverage of *faulty* (4XX) and *correct* (2XX) status codes. This criterion can be applied to every operation separately. For the `GET /search` operation from the Spotify API, for instance, both a successful response (e.g. one including a set of results) and a faulty one (e.g. due to a missing mandatory parameter) should be obtained.
Status code coverage. It makes reference to obtaining API responses with all the status codes that are defined in the specification. This criterion can also be applied to every operation separately. In Spotify, the `GET /search` operation can return a 200 upon success, or a set of faulty status codes when an error occurs, namely 400, 401, 403, 404 and 429. To achieve 100% coverage of this criterion in this example, all these six status codes should be obtained.
Response body properties coverage. For API responses including a body, it should be checked that it contains all response properties defined in the specification. This criterion can be considered for a single response. The `GET /search`

operation in the Spotify API returns an array containing up to three different types of objects: artists, albums and tracks; to achieve full coverage, a search of all three types of results should be performed.

Content-type coverage. The same way that RESTful Web APIs can accept data in several formats, they can also return it. This criterion measures the percentage of response data formats covered by a test suite. The Spotify API produces only JSON, therefore no other data formats need to be obtained.

4 Future work

We are currently developing a prototype tool to automatically measure the degree of coverage reached by a test suite based on the proposed criteria. Once completed, we will evaluate the feasibility of each criterion, refining them as required. Our next step will focus on the definition of an assessment framework, where the selected criteria will be conveniently arranged into different levels. For example, level 0 could require path coverage, level 1 could require the fulfillment of level 0 plus the coverage of operations, and so on. The final goal is to assess the effectiveness of test suites based on the coverage level they reach, which in turn will depend on the input and output elements covered, as described in Section 3. Hopefully, this will result in a widely-accepted method for the assessment and comparison of testing approaches for RESTful Web APIs.

Acknowledgements

This work has been partially supported by the European Commission (FEDER) and Spanish Government under projects BELI (TIN2015-70560-R) and HORATIO (RTI2018-101204-B-C21), and the FPU scholarship program, granted by the Spanish Ministry of Education and Vocational Training (FPU17/04077).

References

1. Atlidakis, V., Godefroid, P., Polishchuk, M.: REST-ler: Automatic Intelligent REST API Fuzzing. Tech. rep., Microsoft Research (2018)
2. Bartolini, C., Bertolino, A., Marchetti, E., Polini, A.: WS-TAXI: A WSDL-based Testing Tool for Web Services. In: Intern. Conference on Software Testing Verification and Validation. pp. 326–335 (2009)
3. Ed-Douibi, H., Canovas Izquierdo, J., Cabot, J.: Automatic generation of test cases for REST APIs: A specification-based approach. In: Intern. Enterprise Distributed Object Computing Conference. pp. 181–190 (2018)
4. OpenAPI Specification, https://github.com/OAI/OpenAPI-Specification, accessed June 2019
5. Segura, S., Parejo, J., Troya, J., Ruiz-Cortes, A.: Metamorphic testing of RESTful Web APIs. IEEE Transactions on Software Engineering **44**(11), 1083–1099 (2018)
6. Spotify API, https://developer.spotify.com/documentation/web-api/reference/, accessed March 2019