A Public Benchmark of REST APIs

Alix Decrop
NADI, University of Namur
Namur, Belgium
alix.decrop@unamur.be

Sara Eraso
University of Valle
Cali, Valle del Cauca, Colombia
sara.eraso@correounivalle.edu.co

Xavier Devroey
NADI, University of Namur
Namur, Belgium
xavier.devroey@unamur.be

Gilles Perrouin
NADI, University of Namur
Namur, Belgium
gilles.perrouin@unamur.be

***Abstract*—In software engineering, benchmarks are widely used to evaluate and compare the performance, functionality, and reliability of analysis tools. Despite the prevalence of benchmarks in areas such as databases, machine learning, and programming languages, there is a notable absence of publicly available benchmarks for REST APIs, a cornerstone of modern web-based systems. While existing research papers occasionally employ similar REST APIs in their evaluations, opportunistic API selection hampers comparison. Moreover, these studies often rely on API documentation and structural characteristics. Without a reliable benchmark, API data used in evaluations may be outdated or inaccurate, compromising reliability and reproducibility.**

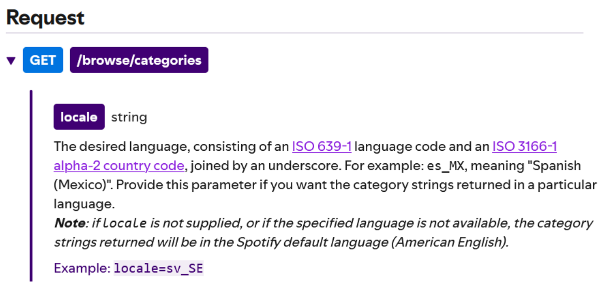
Hence, this paper addresses a gap in the literature by providing a comprehensive and *Public REST API Benchmark* (*PRAB*), to be utilized by researchers in their evaluations. The benchmark contains documentation and structural characteristics of 60 publicly available REST APIs. First, we conduct a systematic mapping study to discover the available and public REST APIs that are utilized in the academic literature. Then, by analyzing the resulting APIs, we report their structural characteristics (*e.g.*, routes, query parameters, HTTP methods, authentication). Finally, we provide their documentation (*i.e.*, OpenAPI Specification, Postman Collection) in a publicly available GitHub repository, to help with future evaluations of REST API studies.

***Index Terms*—REST API, Benchmark, Documentation, OpenAPI Specification, Testing, Systematic mapping study.**

I. Introduction

REST APIs are very popular for supporting web services, such as providing country data, currency exchange statuses, weather information, movie databases, etc. Such APIs adhere to the REpresentational State Transfer (REST) [1] architectural style, hence the name. REST is characterized by a set of design principles; In particular, REST APIs utilize the HTTP protocol to send requests and receive responses containing usage-related data.

REST APIs utilize documentation for two key reasons. First, documentation is required for users to understand the inner workings of a REST API. Indeed, since such APIs utilize routes, parameters, and base URLs, the user must understand how to employ them to form relevant requests. To this end, any form of documentation is viable, as long as it is human-readable and contains natural language descriptions/examples. Thus, many REST APIs have a publicly available website, on which a documentation web page is often available. Figure 1 presents a human-readable documentation excerpt for the Spotify API [2], serving solely as an illustration example.


Request

GET /browse/categories

locale string

The desired language, consisting of an ISO 639-1 language code and an ISO 3166-1 alpha-2 country code, joined by an underscore. For example: es_MX, meaning "Spanish (Mexico)". Provide this parameter if you want the category strings returned in a particular language.
***Note**: if Locale is not supplied, or if the specified language is not available, the category strings returned will be in the Spotify default language (American English).*

Example: locale=sv_SE

Fig. 1. Human-readable documentation excerpt for the Spotify API.

Second, testing tools for REST APIs require machine-readable API documentation to drive test generation. Such testing tools can detect bugs in REST APIs, by uncovering server errors (*e.g.*, `5xx` status codes returned by the API server). Most tools employ the OpenAPI Specification (OAS) [3], which uses the YAML or JSON formats. One can also produce human-readable documentation from an OAS specification via, *e.g.*, the Swagger Editor [4]. Another popular form of machine-readable specification is a Postman Collection [5].

REST API testing is an active research field (*e.g.*, [6], [7], [8], [9]) offering diversified tools such as fuzzers and static analyzers. To evaluate their techniques, researchers select APIs of interest depending on characteristics such as size, OAS availability, and access to the source code that matches the techniques' hypotheses. Some REST APIs are used in different research papers, such as the Features Service API [10] (microservice for managing products feature models), enabling comparison among tools. However, there is a notable absence of publicly available benchmarks for REST APIs. This hinders (1) comparison as researchers use convenience sampling when selecting APIs, and (2) reliability as evaluation data (*e.g.*, API documentation) can be outdated and/or incorrect. This paper aims to fill this gap by gathering a set of 60 public REST APIs based on a systematic mapping study and analyzing their structural characteristics/documentation. Overall, this paper offers the following key contributions:

1) A *Public REST API Benchmark (PRAB)*, containing doc-

This research was funded by the CyberExcellence by DigitalWallonia project (No. 2110186), funded by the Public Service of Wallonia (SPW Recherche); Sara Eraso received support from ERASMUS; Gilles Perrouin is an FNRS Research Associate.

978-1-5386-5541-2/18/$31.00 ©2025 IEEE. This is the authors' version. The final version is published in the *Proceedings of the 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, 28-29 April 2025, Ottawa, Canada.

umentation and structural characteristics of 60 public and distinct REST APIs.
2) A systematic mapping study to discover and analyze the REST APIs that are utilized in the evaluations of relevant research papers.
3) A publicly available GitHub repository [11], containing our benchmark and evaluation data.

We provide the background and related work in Section II. Then, we describe our approach in Section III. Section IV presents our evaluation, while Section V offers an additional discussion. Section VI describes threats to validity. Finally, Section VII wraps up with the conclusion and future work.

II. Background and Related Work

A. *REST APIs*

REpresentational State Transfer (REST) [1] is an architectural style offering several principles for web-based application design. These principles include stateless communication on top of the HTTP protocol, using requests to perform various *CRUD* (*create, read, update, delete*) operations on data resources identified by URIs. In a HTTP request, *routes* (*e.g.*, `/users`) and *query parameters* (*e.g.*, `name=john`) describe what the API should do. APIs implementing or extending REST are termed *RESTful* [12].

HTTP status code interpretations are API-dependent. For the REST Countries API [13], a response for a request with an invalid route contains a `404 - Not Found` status code, which is the standard for non-existing resources. However, for the Bored API [14], it would contain a `200 - OK` status code with the following JSON data: {`"error":"Endpoint not found"`}. Both APIs adhere to HTTP and utilize it to indicate an invalid route, yet not in the same manner.

Regarding API terminology, we use the term "REST API" for an API adhering to the REST architectural style defined by Fielding [1]. In this paper, the term "API" is used as a shorthand for a REST API.

B. *REST API Documentation*

One can rely on documentation to better understand API usage. The *OpenAPI Specification* (OAS) [3] - previously known as the *Swagger Specification* - is a widely adopted format for describing REST APIs. OAS is machine-readable and also human-readable, as some fields can contain natural language descriptions. Moreover, editing tools such as the *Swagger Editor* [4] can convert OAS into human-readable documents. Figure 2 presents an illustration example of an OAS excerpt for An API of Ice and Fire [15].

Another way to document REST APIs is by using *Postman* [5]. Postman is largely used for developing, testing, and managing REST APIs. It provides tools for crafting and sending requests, managing environments, and automating tests. REST APIs can be organized in a *Collection* with Postman, containing descriptions regarding their routes, HTTP methods, parameters, etc. Similarly to OAS, one can export a Postman Collection in the JSON format.

```yaml
openapi: 3.1.0
info:
  title: An API of Ice and Fire
  description: OAS for An API of Ice and Fire.
  version: v1
servers:
  - url: 'https://anapioficeandfire.com/api'
    description: Production Server of the API.
paths:
  /characters:
    get:
      description: Lists all characters.
      parameters:
        - name: name
          description: Filters characters by name.
          in: query
          required: false
          schema:
            type: string
          examples:
            1:
              value: 'Jon+Snow'
            2:
              value: 'Eddard+Stark'
...
```

Fig. 2. Example of an OpenAPI Specification excerpt, in the YAML format.

Documenting is important, allowing developers to understand, reuse, and test APIs; Sohan et al. [16] underlined the effectiveness of documenting API usage examples. However, documenting is time-consuming and error-prone. Neglecting this task results in unavailable, incomplete, or non-machine-readable documentation. As a result, automated documentation generation has been explored in the literature. Some existing methods require a prior form of documentation [17], [18], [19], an HTTP proxy server [20], crawling the API user interface [21], [22], using API call examples [23], or utilizing white-box static analysis [24].

C. *Black-Box REST API Testing*

REST API testing is an active research field, as witnessed by several surveys [25], [26], [27]. State-of-the-art automated testing tools commonly use a black-box approach, requiring an OAS of the API under test [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [7], [40], [41], [42], [43], [9]. REST API testing consists of generating random HTTP requests based on a specification given as input, and analyzing the HTTP responses returned by the API server based on an oracle. Testing is important to mention in this work, as an objective of our benchmark is to assist testing practices by providing public documentation (OAS) for REST APIs.

D. *REST API Benchmarks*

As explained in Section I, there exists no public benchmark of REST APIs. However, REST APIs are sometimes included in larger benchmarks. Arcuri et al. [44] propose EvoMaster Benchmark (EMB), a publicly available GitHub repository containing a set of web applications for experimentation in automated system testing, mainly for the tool EvoMaster [45]. Di Meglio et al. [46] present a performance benchmark of frameworks and execution environments, to be utilized when starting a new REST API project. Neumann et al. [47] analyze 500 REST APIs in terms of compliance and adherence to best practices. Bermbach et al. [48] present a revisited benchmark of web API quality. Finally, Amoroso et al. [49] provide a microservice dataset including projects for REST APIs.

Research studies for REST APIs often construct their own benchmarks. Indeed, for evaluation purposes, a set of REST APIs is required to demonstrate the effectiveness and efficiency of the new approaches. Unfortunately, the evaluation benchmarks are never the same in research papers. To illustrate, we found 2 different research papers [9], [24] with REST APIs in their evaluations. Certain APIs can be found in the evaluation of both research papers: the OCVN, Ohsome, and REST Countries APIs. Nonetheless, other APIs do vary (*e.g.*, the second paper uses the DigDag, EnviroCar, and Catwatch APIs), which is worrisome for comparison purposes. For this reason, we aim to synthesize the REST APIs used in the evaluations of various research papers into a comprehensive and public benchmark.

Websites such as APIs.guru [50] and Public APIs [51] offer extensive listings of REST APIs. However, they lack reliability in terms of evolution (multiple APIs are outdated or do not exist anymore), relevance (a lot of APIs are not used in related research, or not found on the web), and completeness (the Public APIs repository only lists API websites).

III. Approach

To build our benchmark of REST APIs, we conducted a systematic mapping study, following a process provided by Petersen et al. [52]. We explain the protocol in the following subsections.

A. Research Questions

To begin, we formulate the following research questions:

RQ.1: Which research papers in the REST API literature contain an evaluation with a set/benchmark of REST APIs? RQ.1 aims to filter the papers in the REST API literature that contain a set (or a benchmark) of REST APIs in their evaluation. Indeed, most REST API studies utilize a set of APIs to evaluate their approach. This forms the starting point of our building process.

RQ.2: Which REST APIs are used in the relevant scientific literature? RQ.2 aims to present the REST APIs employed in the scientific literature (*i.e.*, in the research papers of RQ.1), as the main objective of this paper is to provide a comprehensive benchmark of public REST APIs. In consequence, RQ.2 will identify the candidate APIs for our benchmark.

RQ.3: Which REST APIs obtained in RQ.2 are actionable for research purposes? In addition to their identification, we must verify if the APIs found in RQ.2 can be used in the benchmark. Indeed, if an API mentioned in a paper is not public, is anonymized, has no documentation, or does not exist anymore, there is no purpose in adding it to the benchmark as it is not actionable for research purposes. The aim of RQ.3 is thus to filter out undesirable APIs, based on defined criteria.

RQ.4: What are the structural characteristics of the REST APIs? RQ.4 aims to detail the structural characteristics of the REST APIs selected in RQ.3. In the context of this paper, the structural characteristics comprise the number of routes and query parameters of the API, the distribution of HTTP methods employed in the API, the rate limit, the authentication method, etc. These characteristics allow for a more informed selection, enabling researchers to filter APIs that match specific experimental needs.

RQ.5: How to obtain the OpenAPI Specifications of the REST APIs? As mentioned in Section II, OAS is a widely adopted format used by many REST API testing tools. In consequence, this documentation format is a crucial aspect of our benchmark, for usability purposes. In consequence, RQ.5 aims to find the OpenAPI Specifications of the REST APIs and explains the inference process. The specifications found throughout RQ.5 are provided in our GitHub repository [11].

B. Search Strategy

Next, we formulate a search strategy. To identify relevant papers, we used the following list of online databases:

- *ACM Digital Library*
- *Google Scholar*
- *IEEE Xplore*
- *MIT Libraries*
- *Semantic Scholar*

We craft a search string focusing on REST APIs and evaluations to find relevant studies. We also include testing and documentation keywords, which reflect our motivation detailed in Section I and Section II. The search string is:

```
SEARCH STRING = "(REST OR RESTful) API (test
OR testing OR documentation OR specification
OR OpenAPI OR Swagger) (evaluation OR
benchmark OR set)"
```

We used a snowballing technique to ensure that no relevant articles were overlooked. This allows us to expand the initial corpus of papers by looking at referenced papers, alleviating search engine limitations and broadening the search scope. Consequently, we applied our research string to the five different databases. We synthesized our findings, by combining the papers found in a single file and removing the duplicates (*i.e.*, the same research paper appearing in different databases). The results of this process (including snowballing) yielded 100 distinct research papers for consideration.

C. Relevant Papers

To filter relevant papers, we define inclusion and exclusion criteria in Table I. Inclusion criteria n°2 states that "the work of the paper on REST APIs is related to one of the following areas: testing or documentation". Indeed, as presented in Section I and Section II, the two main topics in the REST API field are testing and documentation. These additional criteria provide a supplementary guard against including papers referring to REST APIs but not solving a problem related to the REST

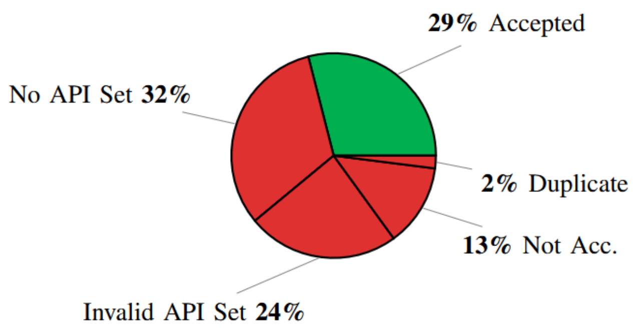


Fig. 3. Overview of the research paper acceptance, based on our inclusion and exclusion criteria. Red slices are rejected papers, of different categories. Not Acc. = Not Accessible.

API research field. Exclusion criteria n°3 states that "the set of REST APIs used in the evaluation of the paper is not explicit or private". This is when the research paper only mentions the number of APIs used from a repository or refers to an unidentified internal (or private) API. For instance, if a paper does not *explicitly* present the set of REST APIs used in the evaluation, the paper will be rejected. We did not exclude papers that are not peer-reviewed (*e.g.*, from arXiv, books, or Master's Theses), as they may list interesting APIs as well. Similarly, we did not exclude surveys/literature reviews and cited their API occurrences.

Based on these inclusion and exclusion criteria, we accepted 29% of papers and rejected the remaining 71%. Figure 3 presents the categories of accepted and rejected papers along with their respective percentages. 2% of the papers were rejected as very similar papers were already accepted. In this case, we compare the two similar papers and only accept the most recent one. Moreover, 24% of papers were rejected because they contained an invalid API set. Such papers did contain an evaluation with a set of REST APIs, but did not satisfy our inclusion criteria. An invalid API set is when:

- The research paper does not explicitly state the REST APIs used in the evaluation. For instance, if a paper mentions "200 APIs were used" but fails to list the APIs explicitly, it will be rejected.
- The research paper utilizes private REST APIs.

We also rejected papers that did not contain an API set in their evaluation (32%) and papers that were not accessible (13%).

D. Keywording

In this step, we list the most frequent keywords found in the surveyed studies to understand their context and ensure it is in line with our search string. This approach allows us to develop a high-level understanding of the nature and contribution of the research papers. We also classify the research papers according to different facets and/or categories. To do so, we analyze the research papers which were filtered and selected in Section III-C.

The keywords indicated in the selected research papers are extracted. When a keyword is composed of multiple words (*e.g.*, REST API, black-box testing), it is decomposed so that each word can be analyzed individually (*e.g.*, REST, API, black-box, testing). The keywords are then documented when they occur at least three distinct times. The number of occurrences for similar keywords such as test/testing, REST/RESTful, and API/APIs are merged. Table II presents the resulting set of keywords. As displayed, the "test/testing" keyword occurs the most (38 occurrences), followed by "REST/RESTful" (23 occurrences), and by "API/APIs" (17 occurrences). This is in line with the scope of our search string defined in Section III-B. Other keywords such as "web", "generation", "software", "OpenAPI", and "black-box" also appear in different research papers.

E. Data Extraction and Analysis

Table III shows the information extracted from the 29 studies that were selected. It maps the role of extracted data for each research question and gathers it into categories.

Another important information to extract from the selected research papers is the venue in which they were published. To do so, the publication's acronym was extracted from each research paper. For research papers published in conferences, their rank was found on the ICORE Conference Portal (CORE2023 database) [53]. Table IV presents the publications, sorted by occurrences. As depicted, most selected references are published in conferences and journals. Eight distinct papers are published in the renowned conferences ICSE (A* rank) and ISSTA (A rank). Some other research papers are published in FSE, ASE (A* rank), ICSME, ICSOC, and ISSRE (A rank). Three selected research papers are published in the TOSEM journal. Additionally, only two unpublished papers were extracted from arXiv, and only two references were Master's Theses.

IV. Evaluation

A. RQ.1 - Research Papers with REST API Evaluations

Our first research question aims to find the research papers in the REST API literature that contain an evaluation. The evaluation should contain a set of REST APIs, that we can leverage to construct the benchmark. Section III already details the process used to find the relevant research papers.

RQ.1 Summary: By searching with a specific search string and in five different databases, we could extract 100 distinct research papers. Next, by applying our inclusion and exclusion criteria, we could filter and accept 29 relevant research papers (*i.e.*, containing a valid set of REST APIs in their evaluations).

B. RQ.2 - REST APIs Used in the Literature

The second research question aims to find the REST APIs that will constitute our benchmark. RQ.1 and Section III presented the approach (*i.e.*, systematic mapping study) that was used to find the relevant papers in the literature. The REST APIs used in the evaluations of these papers were identified,

TABLE I
INCLUSION AND EXCLUSION CRITERIA USED TO FILTER RELEVANT RESEARCH PAPERS.

Inclusion Criteria
(1) The main subject of the paper is about REST (or RESTful) APIs. (2) The work of the paper on REST APIs is related to one of the following areas: testing or documentation.
Exclusion Criteria
(1) The paper mentions REST APIs, but they are not the primary contribution (*e.g.*, implementing a REST API for a tool instead of addressing a fundamental REST API problem). (2) The paper does not contain a set of REST APIs in its evaluation. (3) The set of REST APIs used in the evaluation of the paper is not explicit or private. (4) The paper closely resembles/duplicates a previously accepted paper (we only retain the most recent version of the work). (5) The paper is not in English. (6) The complete reference (research paper, book, thesis, etc.) is not accessible.

TABLE II
SUBSET OF KEYWORDS (WITH AT LEAST THREE OCCURRENCES) EXTRACTED FROM THE SELECTED RESEARCH PAPERS.

Keyword	**No. Occurrences**
Test / Testing	38
REST / RESTful	23
API / APIs	17
Web	9
Generation	8
Software	8
OpenAPI	7
Black-box	5
Case	5
Service	4
Analysis	3
Coverage	3
Language	3
Specification	3

extracted, and reported. API occurrences are also documented (*i.e.*, appearing in multiple papers). A total of 80 APIs were found, which are cited in our public GitHub repository [11].

RQ.2 Summary: From the 29 selected research papers (*i.e.*, satisfying our inclusion and exclusion criteria), we could extract 80 distinct REST APIs. The number of occurrences in research papers for each API ranges from a minimum of 1 to a maximum of 12. The three APIs appearing the most are Features Service [10] (12 occurrences), LanguageTool [54] (10 occurrences), and ProxyPrint [55] (9 occurrences).

C. RQ.3 - Actionable REST APIs for the Benchmark

Before creating the benchmark of REST APIs, an additional filtering process needs to be done. As the primary goal of our benchmark is to provide structural characteristics and documentation (*i.e.*, OAS), some REST APIs that were found might not be used. Indeed, if an API referenced in a research paper is not publicly available, is anonymized, does not exist anymore, or does not contain any form of documentation, it should be excluded from the benchmark as it cannot be leveraged by testing tools and API users.

In consequence, we filtered out certain APIs based on additional exclusion criteria. Table V presents the exclusion criteria with the number of APIs excluded from the benchmark. For instance, the Ind0 API was excluded from our benchmark, as the API was anonymized with the name "Industrial API n°0" (hence Ind0). Another example is the Bored API, as its URL leads to a "Heroku Application Error" page, signifying that the API server is no longer online. Moreover, 9 APIs were excluded as no reference was found online/in the research papers. One of the API documentation was not in English, and thus it was rejected as we could not understand it. As a result, the benchmark contains a total of 60 valid APIs, out of the initial 80 APIs found in the research papers.

Due to size limitations, Table VI presents a subset of our benchmark, limited to the REST APIs appearing in at least 2 distinct research papers. This results in an excerpt of 28 APIs, out of the 60. Nevertheless, the full benchmark is available in our GitHub repository [11]. For each API, the **Application Domain** is detailed. Moreover, the **Used By** column contains the citations for the research papers that utilize the API. Finally, the **No. Occurrences** provides the total number of citations for the API.

RQ.3 Summary: To filter the REST APIs that are actionable for the benchmark, we excluded 20 APIs out of the initial 80 APIs that were found in RQ.2. Indeed, certain APIs were excluded for different reasons (*e.g.*, could not be found, did not exist anymore, were anonymized, etc.), hampering the benchmark quality. In consequence, this translates to a benchmark of 60 public and actionable REST APIs.

D. RQ.4 - Structural Characteristics of the REST APIs

After selecting REST APIs for the benchmark, the next step consists of identifying their structural characteristics. Indeed, the structure of a REST API can drastically change from one API to another API. For instance, an API could contain a single GET route with various usable query parameters (*e.g.*, a `/users` route with the query parameters `id`, `name`, `age`). On the other hand, an API could contain a lot of different

TABLE III
DATA EXTRACTION PROPERTIES MAPPED TO THE RESEARCH QUESTIONS.

Property	Type of Data Extracted	Research Question
Global Information	Authors, title, publication year, summary	RQ.1, RQ.2, RQ.3, RQ.4
Research Problem	Research questions related to REST APIs	RQ.1, RQ.2, RQ.3, RQ.4
Sources Implemented	Set of APIs used, and frequency of use	RQ.1, RQ.2
Documentation	Available documentation (OpenAPI/Postman formats)	RQ.3, RQ.4, RQ.5
Structural Characteristics	HTTP methods, number of routes, parameters, authentication method, etc.	RQ.4

TABLE IV
CONFERENCES, JOURNALS, AND ARCHIVES IN WHICH THE SELECTED RESEARCH PAPERS WERE PUBLISHED, SORTED BY OCCURRENCE.

Published In	Category (Rank)	No. Occurrences
ICSE	Conference (A*)	4
ISSTA	Conference (A)	4
TOSEM	Journal	3
arXiv	Archive	2
FSE	Conference (A*)	2
University Library	Master's Thesis	2
ASC	Journal	1
ASE	Conference (A*)	1
A-TEST	Workshop	1
COORDINATION	Conference (C)	1
DEEPTEST	Workshop	1
ECMFA	Conference (New)	1
ICSME	Conference (A)	1
ICSOC	Conference (A)	1
ISSRE	Conference (A)	1
SCAM	Conference (C)	1
SENSORS	Journal	1
SQJ	Journal	1

TABLE V
NO. OF REST APIS EXCLUDED FROM THE BENCHMARK WITH CORRESPONDING EXCLUSION CRITERIA.

Exclusion Criteria	No. Excluded APIs
The API cannot be found	9
The API does not exist anymore	3
The API does not have a documentation	2
The API installation fails	2
The API documentation is not in English	1
The API has an invalid OAS/Postman file	1
The API is anonymized	1
The API is not REST	1

specialized GET routes without any query parameters (*e.g.*, the routes `/user/{id}`, `/user/{name}`, `/user/{age}`).

Moreover, REST APIs utilize different HTTP methods depending on their application domain. For instance, APIs specialized in retrieving data (*e.g.*, a weather API to retrieve the current weather, precipitation index, or weekly forecast) would contain various routes with the GET HTTP method. Other APIs specialized in sending data (*e.g.*, a booking API to book a hotel room, order food, or send a payment) would contain various routes with the POST HTTP method. Similarly, update-heavy APIs would utilize the PATCH (modify parts of a resource) and PUT (create or replace a resource) HTTP methods.

Structural characteristics related to API usage are also important. For instance, if a user wants to leverage a REST API, the user needs to have a direct overview of the API authentication method, pricing, and rate limits/quotas. The user can then choose APIs in an informed and transparent way. Therefore, we identified structural characteristics that are important for API usage and testing. We consider the following characteristics:

Ref. The reference (*e.g.*, official website, GitHub repository, etc.) of the REST API, containing usage-related information. This reference is notably used when verifying structural characteristics such as authentication, pricing, and rate limits. We did not include API source code in our benchmark. Instead, our references lead to the respective download pages (or websites) of the REST APIs.

Availability. The availability of the API:

- **LOCAL:** The API has to be deployed locally.
- **ONLINE:** The API is hosted on a web server online.
- **BOTH:** The API can be deployed locally and is also hosted online.

Auth. The authentication method to access the API:

- **KEY:** The API requires an access key for user authentication.
- **NO:** The API does not require authentication.

Pricing. The pricing required to use the API:

- **YES:** The API has a mandatory pricing plan.
- **NO:** The API has no pricing plan and is entirely free.
- **OPT:** The API has an optional pricing plan, yet can still be used freely.

Limits. The request rate limits and/or quotas when using the API:

- **YES:** The API has a rate limit and/or a quota.
- **NO:** The API has neither a rate limit nor a quota.

No. Routes. The total number of routes contained in the API. As a single route can handle multiple HTTP methods at once (*e.g.*, `/cart` route with GET to see items in the cart and POST to add items to the cart), it is plausible that an API contains fewer routes than HTTP methods.

No. Param. The total number of distinct query parameters from all routes contained in the API. For example, if a fictive API contains a `/getPet` route accepting the parameters `[id, name, species]` and a `/getStore` route accepting the parameters `[id, location]`, the resulting set of query parameters for the API consists of `[id, name, species, location]`.

TABLE VI
SUBSET OF REST APIS FROM THE BENCHMARK, OCCURRING IN AT LEAST 2 DISTINCT RESEARCH PAPERS.

API Name	Application Domain	Used By	No. Occurrences
Amadeus Hotel	Hotel Booking	[41], [56], [57]	3
Catwatch	GitHub Statistics	[58], [59], [60], [24], [61], [62], [63]	7
CWA Verification Server	Verification Server in CWA App	[64], [60], [24]	3
FDIC	Federal Deposit Insurance Corporation	[41], [9]	2
Features Service	Management Products	[62], [58], [64], [36], [65], [59], [60], [24], [61], [66], [67], [63]	12
Foursquare	Global POI Data	[41], [68], [19]	3
Genome Nexus	Genome Data	[64], [9]	2
Gestao Hospital	Hospital Data	[60], [67]	2
GitHub	GitHub Integration and Automation	[68], [19], [56], [57], [69]	5
LanguageTool	Text Review	[64], [36], [41], [60], [9], [66], [67], [70], [69], [71]	10
Marvel	Marvel Comics Data	[41], [56]	2
OCVN	Importing Vietnam Public Procurement Data	[64], [60], [24], [9], [66], [63]	6
Ohsome	OpenStreetMap History Data	[41], [24], [9], [70], [71]	5
OMDb	Movies Data	[41], [9], [23], [72], [56], [57]	6
PetClinic	Mock Pet Clinic	[72], [67], [8]	3
PetStore	Mock Pet Store	[36], [23], [67], [6], [73], [74]	6
ProxyPrint	Printshops and Consumers	[58], [64], [59], [60], [24], [61], [66], [62], [63]	9
Realworld App	Platform Build Display	[60], [8]	2
REST Countries	Country Data	[64], [41], [60], [24], [9], [72], [67], [70], [71]	9
REST NCS	Numerical Case Study	[64], [59], [60], [61], [66], [75], [62], [63]	8
REST News	News Sources and Blogs	[58], [64], [59], [60], [61], [66], [75], [62]	8
REST SCS	String Case Study	[64], [59], [60], [61], [66], [75], [62], [63]	8
Scout API	Amazon Products Data	[58], [64], [65], [59], [60], [61], [66], [62], [63]	9
Spotify	Spotify Music Streaming Service Interaction	[41], [9], [56], [57]	4
Stripe	Financial Services	[38], [41], [68], [19], [69]	5
Tumblr	Microblogging and Social Networking	[68], [19]	2
Yelp	Crowd-sourced Reviews about Businesses	[38], [41], [68], [19], [56], [57], [69]	7
YouTube	YouTube Features	[38], [41], [68], [9], [56], [57], [69], [71]	8

GET / POST / PUT / DELETE / PATCH. The total number of routes of the specified HTTP method (GET, POST, PUT, DELETE, or PATCH) that the API supports.

The structural characteristics being explained, Table VII presents the results for the 28 APIs of our benchmark subset.

In addition, we present different statistics and results for the whole benchmark of 60 APIs. Figure 4 presents usage-related characteristics in various polar area charts. For the **Availability** structural characteristic, we can see that there are slightly more online APIs (58%) compared to local APIs (42%). Moreover, the **Authentication** and **Limits** characteristics display very similar results, seemingly correlating as an API key is notably used to track user request limits. We also mention that no local API requires a per-request pricing and/or rate limit, as the server is hosted locally.

Next, general data regarding the routes, query parameters, and HTTP methods is important to analyze. Thus, we present (1) the route to query parameter ratio and (2) the HTTP method distribution for each REST API of the benchmark, in two distinct stacked column charts (in percentages, rounded to the nearest integer). As there are 60 APIs, each one is represented by a number in the charts (ranging from 1 to 60). The API names corresponding to the numbers can be found in our GitHub repository [11]. Figure 5 presents the ratios, and Figure 6 presents the HTTP method distribution. On average, APIs usually have a slightly higher percentage of routes (60%) than query parameters (40%). Regarding the HTTP method distribution, GET is the most prominent method on average (67%), followed by POST (21%), DELETE (6%), PUT (5%), and finally PATCH (1%). Accordingly, we observe a larger proportion of GET HTTP methods, suggesting that REST APIs are more inclined towards data retrieval rather than server-side data modifications. The application domains may suggest this inclination.

RQ.4 Summary: Various structural characteristics for the REST APIs of the benchmark were extracted from API websites, repositories, OpenAPI Specifications, and Postman Collections. These characteristics include important usage-related factors, such as local/online availability, authentication method, pricing, and rate limits. Characteristics such as routes, query parameters, and HTTP methods were also quantified. Additional insight regarding the structural characteristics is provided in Section V.

E. RQ.5 - OpenAPI Specifications of the REST APIs

The last research question assesses the public documentation (OAS) provided with the REST APIs of our benchmark. As mentioned previously in the paper, REST API testing tools often require an OpenAPI Specification of the API under test. Additionally, API users can leverage the OAS for understandability purposes. Consequently, we provide a publicly available OpenAPI Specification for all of the APIs considered in the benchmark, which can be found in our GitHub repository [11]. To do so, we meticulously analyzed API resources online (*e.g.*, official API websites, related repositories, Postman collections, etc.).

Occasionally, the API websites contained an available OAS file (JSON or YAML format) to download, or a reference URL to an OAS file. However, some API websites contained outdated documentation. For this reason, we verified different websites related to the REST APIs. Whenever we found

TABLE VII
STRUCTURAL CHARACTERISTICS FOR A SUBSET OF REST APIS FROM THE BENCHMARK.

API Name	Ref.	Availability	Auth.	Pricing	Limits	No. Routes	No. Param.	GET	POST	PUT	DELETE	PATCH
Amadeus Hotel	[76]	ONLINE	KEY	OPT	YES	48	47	37	13	0	1	0
Catwatch	[77]	LOCAL	NO	NO	NO	6	8	6	0	0	0	0
CWA Verification Server	[78]	LOCAL	NO	NO	NO	5	0	0	5	0	0	0
FDIC	[79]	LOCAL	NO	NO	NO	8	15	8	0	0	0	0
Features Service	[10]	LOCAL	NO	NO	NO	11	0	6	6	1	5	0
Foursquare	[80]	ONLINE	KEY	OPT	YES	22	22	16	6	0	0	0
Genome Nexus	[81]	BOTH	NO	NO	NO	22	6	13	10	0	0	0
Gestao Hospital	[82]	LOCAL	NO	NO	NO	13	4	10	10	3	2	0
GitHub	[83]	ONLINE	KEY	OPT	YES	638	76	510	154	91	154	56
LanguageTool	[54]	ONLINE	KEY	OPT	YES	5	0	2	3	0	0	0
Marvel	[84]	ONLINE	KEY	NO	YES	39	38	39	0	0	0	0
OCVN	[85]	LOCAL	NO	NO	NO	96	26	96	96	0	0	0
Ohsome	[86]	ONLINE	NO	NO	NO	61	18	61	61	0	0	0
OMDb	[87]	ONLINE	KEY	OPT	YES	1	11	1	0	0	0	0
PetClinic	[88]	LOCAL	NO	NO	NO	17	1	14	9	7	6	0
PetStore	[89]	LOCAL	NO	NO	NO	13	6	8	6	2	3	0
ProxyPrint	[55]	LOCAL	NO	NO	NO	79	0	48	30	14	10	4
Realworld App	[90]	LOCAL	NO	NO	NO	12	3	7	6	2	4	0
REST Countries	[13]	ONLINE	NO	NO	NO	22	16	22	0	0	0	0
REST NCS	[91]	LOCAL	NO	NO	NO	6	0	6	0	0	0	0
REST News	[92]	LOCAL	NO	NO	NO	6	2	3	1	3	0	0
REST SCS	[93]	LOCAL	NO	NO	NO	11	0	11	0	0	0	0
Scout API	[94]	LOCAL	NO	NO	NO	21	25	21	10	7	9	2
Spotify	[2]	ONLINE	KEY	NO	YES	68	63	59	5	17	8	0
Stripe	[95]	ONLINE	KEY	NO	YES	388	138	257	271	0	32	0
Tumblr	[96]	ONLINE	KEY	NO	YES	16	19	10	6	0	0	0
Yelp	[97]	ONLINE	KEY	OPT	YES	6	3	6	0	0	0	0
YouTube	[98]	ONLINE	KEY	NO	YES	46	71	20	11	0	15	0

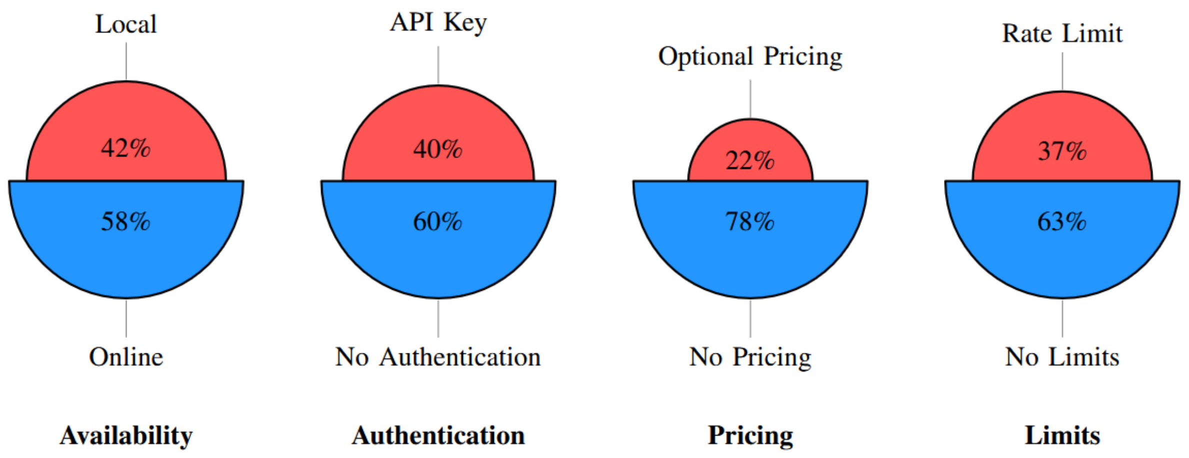


Fig. 4. Polar area charts of different usage-related structural characteristic, for all REST APIs of the benchmark. The percentages are rounded to the nearest integer.

different OpenAPI Specification files, we attempted to provide the most up-to-date version. For instance, the official Spotify API documentation has not been updated in years, according to a community post [99]. While searching the web, we found that a GitHub repository was created to provide up-to-date documentation for the Spotify API [100]. Consequently, we leveraged the GitHub repository data to extract the OpenAPI Specification for the Spotify API. Furthermore, we also utilized tools to transform Postman Collections into OpenAPI Specifications, which we further discuss in Section V.

RQ.5 Summary: For all of the REST APIs in our benchmark, we were able to provide a publicly available OpenAPI Specification for (1) testing and (2) user understandability purposes. We cross-checked multiple sources related to the APIs (*e.g.*, official API websites, Postman Collections, Swagger UIs, and GitHub repositories) to extract the most up-to-date documentation. The OpenAPI Specifications can be found in our GitHub repository [11]. For APIs comprising a Postman Collection, it is also included in a JSON file.

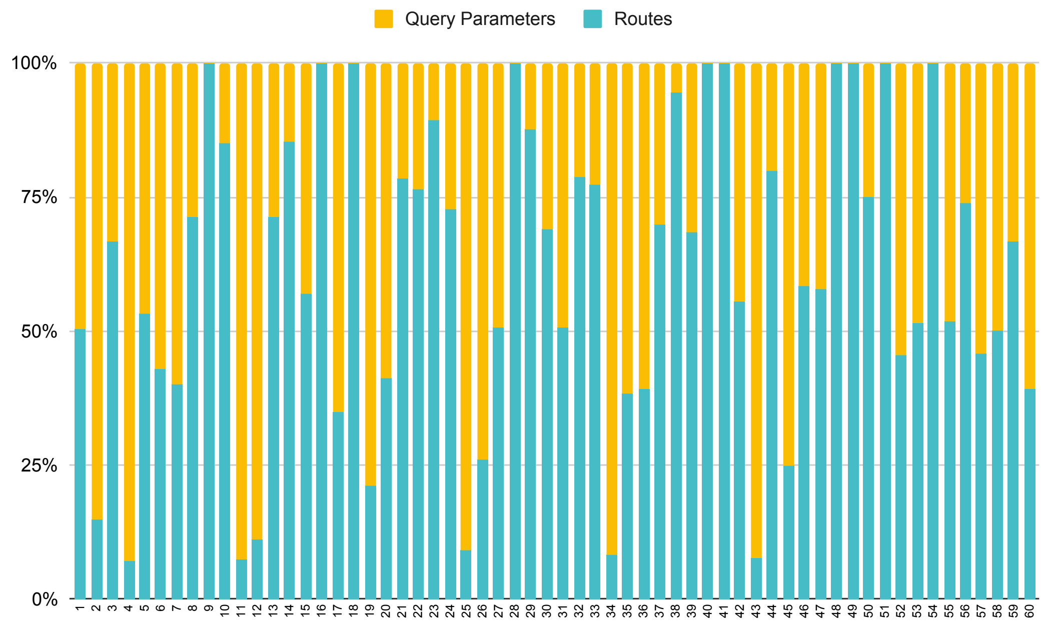


Fig. 5. Stacked column chart of the route to query parameter ratio (in percentages), for each REST API of the benchmark.

V. Discussion

Actionable Insights and Guidance on Using the Benchmark. The main outcome of this research is our benchmark backed by a systematic mapping study. Our goal was not to evaluate the design quality of REST APIs but to provide an unbiased and comprehensive benchmark containing documentation and structural characteristics of 60 public REST APIs. The benchmark is aimed to be used by users, testers, and researchers in their evaluations. We cannot assert that a certain API is *better* compared to another API in the benchmark. Indeed, as REST is not a standard but rather an architectural style, there are multiple ways to implement a REST API (c.f. Section II and Section IV-D). Therefore, we leave the selection of APIs up to the benchmark users but provide concrete metrics to support evidence-based API selection. For instance, a tool for REST API security testing might require the evaluation of APIs that allow data to be sent to the server (*i.e.*, with the POST HTTP method). The evaluation of such a tool would utilize a subset of our benchmark, comprising APIs with a high distribution of POST methods. In consequence, the structural characteristics provided in our benchmark enable researchers to filter the APIs they find relevant (*i.e.*, that match specific experimental needs), without the necessity to utilize all 60 APIs.

For research implying source code analysis, we include URLs for downloading (local) or using (online) the REST APIs. Our quality criteria excluded non-deployable APIs (cf. Table V). Fuzzing research also cares about request rate limits, pricing, and authentication mechanisms, which we also include. Finally, we provide up-to-date documentation in both OpenAPI/Postman formats, which testing tools can leverage. We did not include API source code in our benchmark but rather reference it via respective URLs (GitHub repositories, website downloads, etc.).

On Structural Characteristics. Section IV-D presented various structural characteristics for the REST APIs of the benchmark, such as the HTTP method distribution and the route to query parameter ratio. These characteristics seldom provide insights into API quality on their own, but depend on the application domain and/or context of the API (*e.g.*, GET for retrieving weather data, and POST for sending restaurant orders). However, the presence of certain methods may expose APIs to specific issues (*e.g.*, code injection for POST methods) and be the target of certain quality assurance techniques. For this reason, we allow benchmark users to filter the APIs based on their needs and/or specific criteria. Moreover, results in RQ.4 offer a basis for understanding common variations in API structural design, potentially serving as a reference for future implementations or for improving existing APIs.

Benchmark Documentation. Section IV-D displayed a high variability in the surveyed APIs, inducing diverse granularity levels. Thus, documentation content/elements provided in the benchmark differ depending on the API. However, typical documentation includes API descriptions, notably for routes and parameters. The OpenAPI Specification [3] and Postman Collection [5] describe their standard structures.

Moreover, we need to ensure that other researchers can utilize the benchmark properly in their evaluations, with the provided documentation. To do so, we provided an OpenAPI Specification file in the JSON format for each API. We verified the syntactical validity of such files using a JSON parser. We also verified if the OAS provided in the files are valid, by inserting them into the online Swagger Editor [4]. A similar process is applied for APIs comprising a Postman Collection.

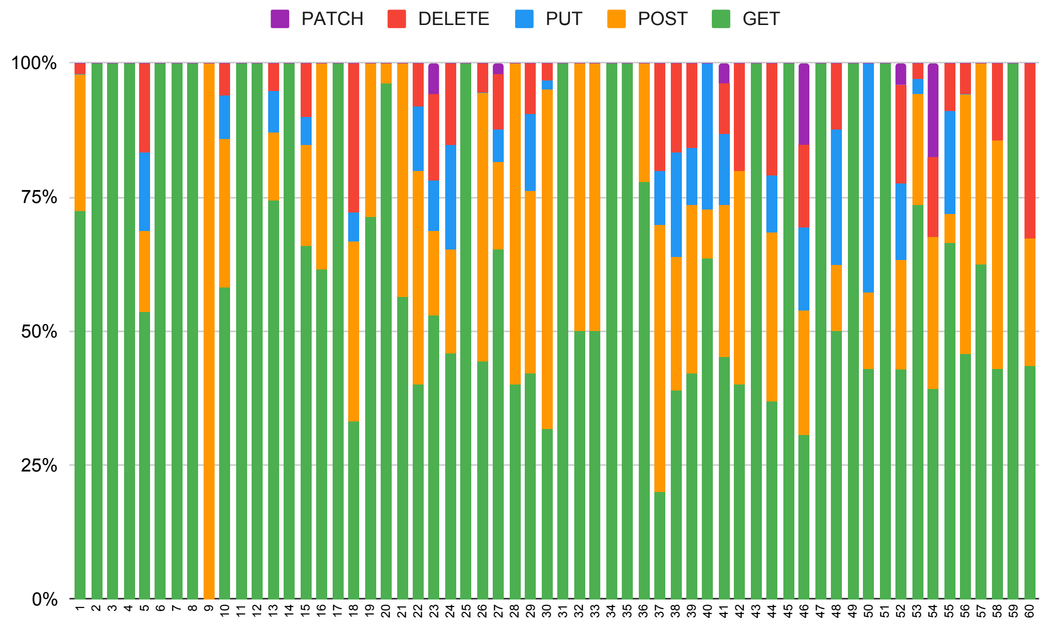


Fig. 6. Stacked column chart of the HTTP method distribution (in percentages), for each REST API of the benchmark.

Benchmark Evolution. In order to properly maintain and keep the benchmark up-to-date, we will routinely keep track of REST APIs updates (*e.g.*, new version/documentation). We also envision a submission mechanism based on pull requests and an automated verification of quality criteria. Moreover, the repository contains a *Issues* page, for users to share their insights/questions/issues regarding the benchmark.

OpenAPI Translation. An issue we encountered in RQ.5 is that we could not directly find all OpenAPI Specifications for the APIs, in the JSON format. Indeed, some APIs provided machine-readable documentation, but not in an adequate format. First, some APIs provided an OAS in the YAML format. To translate this into JSON, we used a tool [101] capable of transforming a YAML file given as input into a JSON file as output. Second, some API specifications were found on Postman, which has its own documentation format. In this case, we used *P2O* [102], a tool capable of converting a Postman Collection into an OpenAPI Specification. We also used a Postman collection to generate an OpenAPI schema from a given Postman collection ID [103]. We used these two tools to cross-check that the generated OpenAPI Specifications are complete and equivalent in both cases. Finally, certain APIs provided documentation through a Swagger interface (*i.e.*, a human-readable webpage equivalent for an OAS). For such APIs, we had to explore/mine the API website by changing routes and/or query parameters to find the machine-readable equivalent version.

API Diversity and Structural Characteristics. Our work focused on carefully selecting REST APIs, based on the research papers filtered in our systematic mapping study. We also carefully reported the API application domains and various structural characteristics. This is an important discussion element, as this process is preferred for external validity [104], [105] compared to convenience sampling that is usually applied.

Other HTTP Methods. We did not include four HTTP methods in our evaluation (HEAD, CONNECT, OPTIONS, and TRACE) due to their infrequent use in REST APIs. HEAD is primarily used for retrieving headers without a response body, CONNECT is mainly utilized for tunneling in proxy connections, OPTIONS is used for discovering supported HTTP methods, and TRACE serves diagnostic purposes. Nonetheless, our benchmark still supports the retrieval of such methods. Should these HTTP methods play a role in the future APIs of our benchmark, we can add them easily as one of the columns of Table VII.

VI. Threats to Validity

Internal Validity. (1) The numbers associated with the structural characteristics of the benchmark APIs (*e.g.* the number of routes and query parameters) might not be accurately reported in our characteristics data. To mitigate this threat, we explored the API websites multiple times to find the related structural characteristics. We also developed a small script to automatically report structural characteristics from a given OpenAPI Specification (*e.g.*, the number of routes, query parameters, and HTTP methods of the API). (2) Our selection of papers for the systematic mapping study might not reflect the whole spectrum of available papers matching our inclusion criteria. To mitigate this threat, we experimented with different search strings and kept the one that yielded the most relevant papers w.r.t. REST APIs. Then, we utilized five different databases, compared the findings, and kept the papers matching our criteria. We also used a snowballing approach with saturation (no new studies were found in the references). The whole process is further detailed in Section III. (3) The OpenAPI Specifications and Postman Collections provided in the GitHub repository are prone to deprecation (as the REST APIs of the benchmark are subject to change in the future), mistakes, or inaccuracies. To mitigate this threat, we meticulously analyzed the online specifications, from different sources, and kept the most complete ones. Our public GitHub repository will be maintained and updated. (4) The tools leveraged to translate a Postman documentation file into an OAS file might contain bugs, leading to an invalid translation containing missing elements and/or inaccuracies. To mitigate such threat, we also provided the Postman Collection file for the APIs that utilized such process, as a backup file to support the generated OAS file.

External Validity. The REST APIs chosen for the benchmark might not accurately reflect all public REST API types. To mitigate this threat, we analyzed REST APIs in various research papers (following a systematic mapping study), related to REST APIs. We filtered the APIs found based on various criteria, notably public availability, then computed various structural characteristics and finally provided their OpenAPI Specifications. Moreover, Table VI presents the wide range of application domains for a subset of APIs found in our benchmark. However, since our search queries explicitly looked for testing and documentation research, this may have influenced the selection of APIs in our benchmark.

Construct Validity. Construct validity concerns the metrics we chose to establish our conclusions. We focused on structural characteristics that serve as proxies for API complexity (routes/number of parameters) and extended them with information useful for black-box/white-box testing such as rate limit (*e.g.*, for fuzzing tools). These characteristics are often reported in the REST API literature (*e.g.*, [8], [19], [27]). We may consider additional metrics in the future.

VII. Conclusion and Future Work

In this paper, we presented *PRAB*, an acronym for *Public REST API Benchmark*. The main motivation of our research was to fill a gap in the REST API literature, as no publicly available benchmark of REST APIs exists, impelling researchers to construct their own in each new study. Indeed, API resources (*e.g.*, websites and online documentation) often lack direct access to (1) important structural characteristics of the APIs (*e.g.*, authentication method, rate limiting, number of routes/parameters, HTTP methods) and (2) machine-readable OpenAPI Specifications (in JSON/YAML format). Such resources are important for testing practices, and user understanding purposes. Consequently, we conducted a systematic mapping study to find the APIs that are used in the evaluations of research papers/studies related to REST APIs, mainly in the testing and documentation fields. Based on the actionable APIs that were found, we constructed a comprehensive benchmark containing documentation (OpenAPI/Postman formats) and various structural characteristics (routes, query parameters, authentication, HTTP methods, etc.) of 60 public REST APIs. The benchmark is publicly available in our GitHub repository [11].

In the future, we would like to keep PRAB up-to-date. To do so, we will keep track of updates related to the REST APIs in the benchmark, and update the GitHub repository accordingly. Next, we would like to add more REST APIs to the benchmark, which could further help API testers and users. Finally, we will add more data formats for each API of the benchmark, such as a YAML version of the OpenAPI Specification (which is currently in JSON), a Postman Collection for all APIs, and additional structural characteristics that could assist REST API researchers, developers, testers, and users alike.

References

[1] R. T. Fielding, *Architectural styles and the design of network-based software architectures.* Irvine, USA: University of California, Irvine, 2000.

[2] Spotify AB, “Spotify api,” https://developer.spotify.com, 2024, [Online; Last Accessed 28-October-2024].

[3] L. Foundation, “Openapi specification,” https://www.openapis.org, 2022, [Online; Last Accessed 5-December-2024].

[4] S. Software, “Swagger editor,” https://editor.swagger.io, 2023, [Online; Last Accessed 5-December-2024].

[5] Postman, “Create api documentation with postman,” https://www.postman.com/api-documentation-tool, 2023, [Online; Last Accessed 19-January-2025].

[6] D. Corradini, A. Zampieri, M. Pasqua, and M. Ceccato, “Restats: A test coverage tool for restful apis,” in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 2021, pp. 594–598.

[7] E. Viglianisi, M. Dallago, and M. Ceccato, “Resttestgen: Automated black-box testing of restful apis,” in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST).* New York, USA: IEEE, 2020, pp. 142–152.

[8] D. Corradini, A. Zampieri, M. Pasqua, and M. Ceccato, “Empirical comparison of black-box test case generation tools for restful apis,” in *2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM).* IEEE, 2021, pp. 226–236.

[9] M. Kim, T. Stennett, D. Shah, S. Sinha, and A. Orso, “Leveraging large language models to improve rest api testing,” in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, 2024, pp. 37–41.

[10] J. Ferrara, “Features service api,” https://github.com/JavierMF/features-service, 2016, [Online; Last Accessed 22-October-2024].

[11] A. Decrop and S. Eraso, “Public rest api benchmark (prab),” https://github.com/alixdecr/PRAB, 2025, [Online; Last Accessed 03-February-2025].

[12] L. Richardson, M. Amundsen, and S. Ruby, *RESTful Web APIs: Services for a Changing World.* Sebastopol, California: ”O’Reilly Media, Inc.”, 2013.

[13] F. Florez and A. Matos, “Rest countries api,” https://restcountries.com, 2024, [Online; Last Accessed 28-October-2024].

[14] D. Thoennes, “Bored api,” https://github.com/drewthoennes/Bored-API, 2020, [Online; Last Accessed 03-February-2025].

[15] J. Skoog, “An api of ice and fire,” https://anapioficeandfire.com, 2023, [Online; Last Accessed 5-December-2024].

[16] S. Sohan, F. Maurer, C. Anslow, and M. P. Robillard, “A study of the effectiveness of usage examples in rest api documentation,” in *2017 IEEE symposium on visual languages and human-centric computing (VL/HCC).* New York, USA: IEEE, 2017, pp. 53–61.

[17] H. Cao, J.-R. Falleri, and X. Blanc, “Automated generation of rest api specification from plain html documentation,” in *Service-Oriented Computing: 15th International Conference, ICSOC 2017, Malaga, Spain, November 13–16, 2017, Proceedings.* Springer-Verlag GmbH, Heidelberg: Springer, 2017, pp. 453–461.

[18] C. González-Mora, C. Barros, I. Garrigós, J. Zubcoff, E. Lloret, and J.-N. Mazón, “Improving open data web api documentation through interactivity and natural language generation,” *Computer Standards & Interfaces*, vol. 83, p. 103657, 2023.

[19] M. Kim, D. Corradini, S. Sinha, A. Orso, M. Pasqua, R. Tzoref-Brill, and M. Ceccato, "Enhancing rest api testing with nlp techniques," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. New York, NY: ACM, 2023, pp. 1232–1243.

[20] S. M. Sohan, C. Anslow, and F. Maurer, "Spyrest: Automated restful api documentation using an http proxy server (n)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. New York, NY: IEEE, 2015, pp. 271–276.

[21] R. Yandrapally, S. Sinha, R. Tzoref-Brill, and A. Mesbah, "Carving ui tests to generate api tests and api specification," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. New York, USA: IEEE Press, 2023, p. 1971–1982. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00167

[22] J. Yang, E. Wittern, A. T. Ying, J. Dolby, and L. Tan, "Towards extracting web api specifications from documentation," in *Proceedings of the 15th International Conference on Mining Software Repositories*. New York, USA: ACM, 2018, pp. 454–464.

[23] H. Ed-Douibi, J. L. Cánovas Izquierdo, and J. Cabot, "Example-driven web api specification discovery," in *European Conference on Modelling Foundations and Applications*. Springer-Verlag GmbH, Heidelberg: Springer, 2017, pp. 267–284.

[24] R. Huang, M. Motwani, I. Martinez, and A. Orso, "Generating rest api specifications through static analysis," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[25] A. Ehsan, M. A. M. Abuhaliqa, C. Catal, and D. Mishra, "Restful api testing methodologies: Rationale, challenges, and solution directions," *Applied Sciences*, vol. 12, no. 9, p. 4369, 2022.

[26] A. Sharma, M. Revathi *et al.*, "Automated api testing," in *2018 3rd International Conference on Inventive Computation Technologies (ICICT)*. New York, USA: IEEE, 2018, pp. 788–791.

[27] A. Golmohammadi, M. Zhang, and A. Arcuri, "Testing restful apis: A survey," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 1, nov 2023. [Online]. Available: https://doi.org/10.1145/3617175

[28] J. C. Alonso, A. Martin-Lopez, S. Segura, J. M. Garcia, and A. Ruiz-Cortes, "Arte: Automated generation of realistic test inputs for web apis," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 348–363, 2022.

[29] V. Atlidakis, P. Godefroid, and M. Polishchuk, "Restler: Stateful rest api fuzzing," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. New York, USA: IEEE, 2019, pp. 748–758.

[30] ——, "Checking security properties of cloud service rest apis," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. New York, USA: IEEE, 2020, pp. 387–397.

[31] D. Corradini, M. Pasqua, and M. Ceccato, "Automated black-box testing of mass assignment vulnerabilities in restful apis," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. New York, USA: IEEE Press, 2023, p. 2553–2564. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00213

[32] P. Godefroid, B.-Y. Huang, and M. Polishchuk, "Intelligent rest api data fuzzing," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, USA: ACM, 2020, pp. 725–736.

[33] P. Godefroid, D. Lehmann, and M. Polishchuk, "Differential regression testing for rest apis," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. New York, USA: ACM, 2020, pp. 312–323.

[34] Z. Hatfield-Dodds and D. Dygalo, "Deriving semantics-aware fuzzers from web api schemas," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*. New York, USA: ACM/IEEE, 2022, pp. 345–346.

[35] S. Karlsson, A. Čaušević, and D. Sundmark, "Quickrest: Property-based test generation of openapi-described restful apis," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. New York, USA: IEEE, 2020, pp. 131–141.

[36] Y. Liu, Y. Li, G. Deng, Y. Liu, R. Wan, R. Wu, D. Ji, S. Xu, and M. Bao, "Morest: model-based restful api testing with execution feedback," in *Proceedings of the 44th International Conference on Software Engineering*. New York, USA: ACM/IEEE, 2022, pp. 1406–1417.

[37] R. Mahmood, J. Pennington, D. Tsang, T. Tran, and A. Bogle, "A framework for automated api fuzzing at enterprise scale," in *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. New York, USA: IEEE, 2022, pp. 377–388.

[38] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, "Restest: automated black-box testing of restful web apis," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. New York, USA: ACM, 2021, pp. 682–685.

[39] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Cortés, "Metamorphic testing of restful web apis," in *Proceedings of the 40th International Conference on Software Engineering*. New York, USA: IEEE/ACM, 2018, pp. 882–882.

[40] H. Wu, L. Xu, X. Niu, and C. Nie, "Combinatorial testing of restful apis," in *Proceedings of the 44th International Conference on Software Engineering*. New York, USA: ACM, 2022, pp. 426–437.

[41] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, "Online testing of restful apis: Promises and challenges," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, USA: ACM, 2022, pp. 408–420.

[42] O. Banias, D. Florea, R. Gyalai, and D.-I. Curiac, "Automated specification-based testing of rest apis," *Sensors*, vol. 21, no. 16, p. 5375, 2021.

[43] S. Ahmed and A. Hamdy, "Artificial bee colony for automated black-box testing of restful api," in *International Conference on Frontiers of Intelligent Computing: Theory and Applications*. Springer-Verlag GmbH, Heidelberg: Springer, 2023, pp. 1–17.

[44] arcuri82, "Emb," https://github.com/WebFuzzing/EMB, 2024, [Online; Last Accessed 10-October-2024].

[45] A. Arcuri, "Evomaster," https://github.com/WebFuzzing/EvoMaster, 2025, [Online; Last Accessed 03-February-2025].

[46] S. Di Meglio, L. L. L. Starace, and S. Di Martino, "Starting a new rest api project? a performance benchmark of frameworks and execution environments." in *IWSM-Mensura*, 2023.

[47] A. Neumann, N. Laranjeiro, and J. Bernardino, "An analysis of public rest web service apis," *IEEE Transactions on Services Computing*, vol. 14, no. 4, pp. 957–970, 2018.

[48] D. Bermbach and E. Wittern, "Benchmarking web api quality-revisited," *Journal of Web Engineering*, vol. 19, no. 5-6, pp. 603–646, 2020.

[49] D. Amoroso d'Aragona, A. Bakhtin, X. Li, R. Su, L. Adams, E. Aponte, F. Boyle, P. Boyle, R. Koerner, J. Lee *et al.*, "A dataset of microservices-based open-source projects," in *Proceedings of the 21st International Conference on Mining Software Repositories*, 2024, pp. 504–509.

[50] APIs.guru, "Apis.guru," https://apis.guru, 2025, [Online; Last Accessed 03-February-2025].

[51] public-apis, "Public apis," https://github.com/public-apis/public-apis, 2025, [Online; Last Accessed 03-February-2025].

[52] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *12th international conference on evaluation and assessment in software engineering (EASE)*. BCS Learning & Development, 2008.

[53] C. R. . Education, "Icore conference portal," https://portal.core.edu.au/conf-ranks, 2023, [Online; Last Accessed 18-October-2024].

[54] LanguageTool, "Languagetool api," https://languagetool.org/http-api, 2024, [Online; Last Accessed 24-October-2024].

[55] F. Sousa, M. Goncalves, and D. Caldas, "Proxyprint kitchen api," https://github.com/ProxyPrint/proxyprint-kitchen, 2016, [Online; Last Accessed 28-October-2024].

[56] J. C. Alonso, S. Segura, and A. Ruiz-Cortés, "Agora: automated generation of test oracles for rest apis," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 1018–1030.

[57] J. C. Alonso, "Automated generation of test oracles for restful apis," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 1808–1810.

[58] A. Arcuri, "Restful api automated test case generation with evomaster," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 28, no. 1, pp. 1–37, 2019.

[59] B. Marculescu, M. Zhang, and A. Arcuri, "On the faults found in rest apis by automated test generation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 3, pp. 1–43, 2022.

[60] M. Zhang and A. Arcuri, “Open problems in fuzzing restful apis: A comparison of tools,” *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 6, pp. 1–45, 2023.

[61] O. Sahin and B. Akay, “A discrete dynamic artificial bee colony with hyper-scout for restful web service api test suite generation,” *Applied Soft Computing*, vol. 104, p. 107246, 2021.

[62] M. Zhang, B. Marculescu, and A. Arcuri, “Resource and dependency based test case generation for restful web services,” *Empirical Software Engineering*, vol. 26, no. 4, p. 76, 2021.

[63] D. Stallenberg, M. Olsthoorn, and A. Panichella, “Improving test case generation for rest apis through hierarchical clustering,” in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 117–128.

[64] M. Kim, Q. Xin, S. Sinha, and A. Orso, “Automated test generation for rest apis: No time to rest yet,” in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 289–301.

[65] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, “Test coverage criteria for restful web apis,” in *Proceedings of the 10th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, 2019, pp. 15–21.

[66] S. Karlsson, R. Jongeling, A. Čaušević, and D. Sundmark, “Exploring behaviours of restful apis in an industrial setting,” *Software Quality Journal*, vol. 32, no. 3, pp. 1287–1324, 2024.

[67] C. B. Burlò, A. Francalanza, A. Scalas, and E. Tuosto, “Cots: Connected openapi test synthesis for restful applications,” in *International Conference on Coordination Models and Languages*. Springer, 2024, pp. 75–92.

[68] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, “Restest: Black-box constraint-based testing of restful web apis,” in *Service-Oriented Computing: 18th International Conference, ICSOC 2020, Dubai, United Arab Emirates, December 14–17, 2020, Proceedings 18*. Springer, 2020, pp. 459–475.

[69] A. G. Mirabella, A. Martin-Lopez, S. Segura, L. Valencia-Cabrera, and A. Ruiz-Cortés, “Deep learning-based prediction of test input validity for restful apis,” in *2021 IEEE/ACM Third International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest)*. IEEE, 2021, pp. 9–16.

[70] A. Martin-Lopez, A. Arcuri, S. Segura, and A. Ruiz-Cortés, “Black-box and white-box test case generation for restful apis: Enemies or allies?” in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 231–241.

[71] ——, “Black-box and white-box test case generation for restful apis: Enemies or allies?” in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 231–241.

[72] A. Decrop, G. Perrouin, M. Papadakis, X. Devroey, and P.-Y. Schobbens, “You can rest now: Automated specification inference and black-box testing of restful apis with large language models,” *arXiv preprint arXiv:2402.05102*, 2024.

[73] A. Tokos, “Evaluating fuzzing tools for automated testing of rest apis using openapi specification,” 2023.

[74] A. Karlsson, “Automatic test generation of rest apis,” 2020.

[75] A. Arcuri, M. Zhang, and J. Galeotti, “Advanced white-box heuristics for search-based fuzzing of rest apis,” *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 6, pp. 1–36, 2024.

[76] Amadeus, “Amadeus hotel api,” https://developers.amadeus.com/self-service/category/hotels, 2024, [Online; Last Accessed 22-October-2024].

[77] A. Kops, “Catwatch api,” https://github.com/zalando-incubator/catwatch, 2017, [Online; Last Accessed 22-October-2024].

[78] Deutsche Telekom AG, “Cwa verification server api,” https://github.com/corona-warn-app/cwa-verification-server, 2023, [Online; Last Accessed 22-October-2024].

[79] D. Bernier, J. Nimety, and T. Reznick, “Fdic api,” https://github.com/ContinuityControl/fdic, 2020, [Online; Last Accessed 22-October-2024].

[80] Foursquare, “Foursquare api,” https://docs.foursquare.com/developer, 2024, [Online; Last Accessed 22-October-2024].

[81] Memorial Sloan Kettering Cancer Center, “Genome nexus api,” https://docs.genomenexus.org/api, 2024, [Online; Last Accessed 22-October-2024].

[82] V. Padilha de Vargas, D. Negrisoli Batista, and L. Wlach, “Gestao hospital api,” https://github.com/ValchanOficial/GestaoHospital, 2019, [Online; Last Accessed 22-October-2024].

[83] GitHub, “Github api,” https://docs.github.com/rest, 2024, [Online; Last Accessed 22-October-2024].

[84] Marvel, “Marvel api,” https://developer.marvel.com, 2024, [Online; Last Accessed 24-October-2024].

[85] S. Alexei and P. Mihai, “Open contracting vietnam (ocvn) api,” https://github.com/devgateway/ocvn, 2017, [Online; Last Accessed 24-October-2024].

[86] HeiGIT gGmbH, “Ohsome api,” https://docs.ohsome.org/ohsome-api, 2024, [Online; Last Accessed 25-October-2024].

[87] B. Fritz, “Omdb api,” https://www.omdbapi.com, 2019, [Online; Last Accessed 28-October-2024].

[88] A. Rey, V. Fedoriv, A. Touret, and et al., “Spring petclinic rest api,” https://github.com/spring-petclinic/spring-petclinic-rest, 2024, [Online; Last Accessed 28-October-2024].

[89] F. Tumanischvili, M. Rychel, and et al., “Swagger petstore sample api,” https://github.com/swagger-api/swagger-petstore, 2024, [Online; Last Accessed 28-October-2024].

[90] L. Jakob and et al., “Realworld example app api,” https://github.com/lujakob/nestjs-realworld-example-app, 2021, [Online; Last Accessed 28-October-2024].

[91] A. Arcuri, “Rest ncs api,” https://github.com/WebFuzzing/EMB/tree/master/jdk_8_maven/cs/rest/artificial/ncs, 2024, [Online; Last Accessed 28-October-2024].

[92] ——, “Rest news api,” https://github.com/WebFuzzing/EMB/tree/master/jdk_8_maven/cs/rest/artificial/news, 2024, [Online; Last Accessed 28-October-2024].

[93] ——, “Rest scs api,” https://github.com/WebFuzzing/EMB/tree/master/jdk_8_maven/cs/rest/artificial/scs, 2024, [Online; Last Accessed 28-October-2024].

[94] ——, “Scout api,” https://github.com/WebFuzzing/EMB/tree/master/jdk_8_maven/cs/rest/original/scout-api, 2021, [Online; Last Accessed 28-October-2024].

[95] Stripe, “Stripe api,” https://docs.stripe.com/api, 2024, [Online; Last Accessed 28-October-2024].

[96] Tumblr Inc., “Tumblr api,” https://www.tumblr.com/docs/en/api/v2, 2024, [Online; Last Accessed 28-October-2024].

[97] Yelp Inc., “Yelp api,” https://docs.developer.yelp.com/, 2024, [Online; Last Accessed 28-October-2024].

[98] Google LLC, “Youtube api,” https://developers.google.com/youtube, 2024, [Online; Last Accessed 28-October-2024].

[99] sonallux, “Openapi/swagger description for the web api,” https://community.spotify.com/t5/Spotify-for-Developers/OpenApi-Swagger-description-for-the-Web-API/td-p/5196705, 2021, [Online; Last Accessed 28-October-2024].

[100] ——, “Spotify web api tools,” https://github.com/sonallux/spotify-web-api, 2021, [Online; Last Accessed 28-October-2024].

[101] Browserling, “Transform yaml into json,” https://onlineyamltools.com/convert-yaml-to-json, 2024, [Online; Last Accessed 05-November-2024].

[102] A. Goyal, “Postman to openapi,” https://p2o.defcon007.com, 2023, [Online; Last Accessed 05-November-2024].

[103] Postman, “Generate an openapi schema,” https://www.postman.com/postman/405e0480-49cf-463b-8052-6c0d05a8e8f3/request/l9w7uk2/generate-an-openapi-schema, 2024, [Online; Last Accessed 06-November-2024].

[104] K. Petersen and C. Gencel, “Worldviews, research methods, and their relationship to validity in empirical software engineering research,” in *2013 joint conference of the 23rd international workshop on software measurement and the 8th international conference on software process and product measurement*. IEEE, 2013, pp. 81–89.

[105] A. Arcuri and L. Briand, “A hitchhiker’s guide to statistical tests for assessing randomized algorithms in software engineering,” *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.