applied sciences

MDPI

Review

RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions

Adeel Ehsan [1], Mohammed Ahmad M. E. Abuhaliqa [1], Cagatay Catal [1] and Deepti Mishra [2,*]

[1] Department of Computer Science & Engineering, Qatar University, Doha 2713, Qatar; ae2100558@student.qu.edu.qa (A.E.); ma2000142@student.qu.edu.qa (M.A.M.E.A.); ccatal@qu.edu.qa (C.C.)
[2] Software, Data and Digital Ecosystem Group, Educational Technology Laboratory, Department of Computer Science, Norwegian University of Science and Technology, 2815 Gjøvik, Norway
* Correspondence: deepti.mishra@ntnu.no

Abstract: Service-oriented architecture has evolved to be the backbone for large-scale integration between different applications and platforms. This concept has led to today's reality of cloud services. Many of the major business platforms are providing their services to end-users and other companies as well. Companies are crafting ways to allow other businesses fast service integration and to get on board quickly in the market. REST (representational state transfer) has emerged as the standard protocol for implementing and consuming these services, which are called RESTful application programming interfaces (APIs). As the internal details of the RESTful APIs are not completely available during consumption, thorough testing has been a major challenge. Any unprecedented change in the APIs can cause the major failure of service operations, which can cause an organization to face both financial and trust losses. Research efforts have been made to alleviate testing challenges by introducing different frameworks and auto-generating unit test approaches. However, there is still a lack of an overview of the state-of-the-art in RESTful API testing. As such, the objective of this article is to identify, analyze, and synthesize the studies that have been performed related to RESTful APIs' testing methodologies and unit test generation. With this perspective, a systematic literature review (SLR) study was conducted. In total, 16 papers were retrieved and included based on study selection criteria for in-depth analysis. This SLR discusses and categorizes different problems and solutions related to RESTful APIs' testing and unit test generation.

Keywords: auto-test case generation; cloud services; JSON base services; micro services; RESTful APIs; testing frameworks

Citation: Ehsan, A.; Abuhaliqa, M.A.M.E.; Catal, C.; Mishra, D. RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions. *Appl. Sci.* **2022**, *12*, 4369. https://doi.org/10.3390/app12094369

Academic Editors: Zhenyu Chen, Chunrong Fang and Song Huang

Received: 20 February 2022
Accepted: 22 April 2022
Published: 26 April 2022

1. Introduction

REST (representational state transfer) is the architecture used for designing services consumed across different platforms and environments to support interoperability and WWW (World Wide Web) [1]. Statelessness and cross-platform consumption readiness are two major attributes of this architecture. It has become a widely followed standardized way of publishing services over the Internet [2]. REST application programming interfaces (APIs) are widely part of the design of micro-services [3]. Research efforts were made to extend the REST architecture to support distributed systems [4].

RESTful APIs, widely known as web APIs, consist of endpoints. Every endpoint is a concrete implemented functionality of a business process. These APIs are generally accessible over HTTP (hypertext transfer protocol) by including defined standard verbs like GET, POST, PUT, and DELETE. RESTful APIs are invoked with the help of an address that is known as a URI (uniform resource identifier).

One of the challenges was to define a standard format of messaging (i.e., request and response). Initially, informal text was used to describe REST APIs [5]. JSON (JavaScript Object Notation) documents evolved later as a standard. The format is pure text and easily identifiable and processable by machines across networks and platforms. The challenge

of coming up with a standardized way for the description of REST services, however, still remains. OpenAPI specification [6] is emerging as one of the solutions to that challenge.

The loosely coupled and adaptable access to RESTful services opens a wide possibility for wrong input to be sent with the request. This could potentially invoke service faults that might not have been caught during static analysis and unit testing. The additional layer of third-party libraries provided by other vendors used in RESTful APIs' development adds more complexity to testing. In this scenario, it becomes even more important to detect and resolve possible bugs in the service for stability. This is particularly severe when mission-critical activities or businesses are the end clients of these services. No matter what is sent from the client as a request, the service should be able to respond in a graceful manner by taking care of any possible runtime fault.

Different research efforts have been made to alleviate RESTful API testing challenges by introducing different frameworks and auto-generating unit test approaches. However, there is still a lack of an overview of the state-of-the-art in RESTful API testing. Therefore, the objective of this article is to identify, analyze, and synthesize the studies that have been performed related to RESTful APIs' testing methodologies and unit test generation.

The objective of this SLR study is to research existing work done related to RESTful API testing. In the light of comprehensive analysis, we aimed to look at the challenges involved in testing, especially the impact on the code coverage. Additionally, this study aims to enlist and discuss available solutions for RESTful API testing and their ability to test authentication-based APIs. The study fulfills the final part of the objective, which is to classify the primary studies appropriately based on the technique used.

A well-known SLR protocol has been adopted to answer the research questions defined in this research and different databases have been searched for finding relevant papers. Study selection criteria were defined for the inclusion and exclusion of retrieved papers, and a quality assessment was performed on the selected papers. Papers that did not satisfy the quality criteria were removed from the analysis and the extraction was performed on high-quality papers. The contributions of this study are three-fold:

1. To the best of our knowledge, this is the first SLR paper on RESTful API testing techniques. An up-to-date overview of the testing methodologies for RESTful APIs is presented.
2. Available methodologies are categorized based on tools, approaches, and frameworks.
3. The main obstacle in achieving a high level of code coverage is identified and discussed.

The remainder of the paper is organized as follows. Background and related work are presented in Section 2. This section is followed by the research methodology. The first sub-section of Section 3 (Research Methodology) is the review protocol (Section 3.1). Afterward, the next sub-section (Section 3.2) outlines the research questions laid out for this article. The next sub-section (Section 3.3) is dedicated to the search strategy followed to retrieve related articles from the electronic databases. Section 3.4 explains the exclusion criteria, Section 3.5 discusses quality assessment, and Section 3.6 presents data synthesis that describes the data extraction and result formulation strategy. Based on synthesized data, the results are presented in the next section (Section 4). Section 5 presents the comprehensive discussion related to research questions and answers. Validity threats are also discussed in the Discussion section, which is followed by the Conclusion section.

2. Background and Related Work

2.1. Background

HTTP has its own methods, which are also known as verbs, that facilitate client–server communication. Originally only GET, HEAD, and POST verbs were defined in the HTTP/1.0. By HTTP/1.1, however, the verbs were expanded to include PUT, DELETE, CONNECT, OPTIONS, and TRACE and, as recent as 2010, PATCH was proposed in RFC 5789.

Representational state transfer (REST) was proposed by Roy Fielding in 2000 in his Ph.D. dissertation [7]. The aim of this style was to improve performance, scalability,

simplicity, modifiability, visibility of the communication, portability, and reliability. To achieve such properties, Fielding defined six controls as follows: architecture artifacts related to client and server, statelessness, the ability to cache data, multi-tier system, standard interface, and optionally on-demand code. As such, out of the HTTP verbs, only the following ones are used for RESTful API: DELETE, GET, POST, and PUT [8].

Whenever a REST API endpoint is invoked, it can be done using one of the verbs described above. For example, if there is an endpoint that returns a list of products from an online shopping website, it can be invoked using the GET verb, which means that there are some data that we want to get as a list of instances or a single instance. The request is called a GET request, and it may have some parameters. Similarly, data can be inserted using a POST request, deleted using a DELETE request, and edited using a PUT request. Accessing any REST API endpoint involves preparing a URL with address, parameters if any, and marking it appropriately with one of the verbs. POST verb is used for sending data with the request as part of the request body instead of embedding it in the URL. This makes it invisible, which is better for sensitive data like credentials. For example, when a user logs in, the username and password are sent to a REST API endpoint as a POST request.

2.2. Related Work

During our research, we did not come across any SLR study that covers RESTful APIs' testing methodologies as a whole and discusses the challenges and solutions available. Therefore, to the best of our knowledge, this is considered to be the first SLR paper that addresses these issues. Since the topic is crucial and relevant for modern software development methodologies, we believe that similar studies will be performed soon and our study will be extended with more published papers published until that date. This paper is beneficial not only for researchers, but also for practitioners in the software industry.

During our search, we came across an SLR study that focuses on web service testing techniques [9]. Ghani et al. [9] stated that they included both SOAP and RESTful web service testing papers in their review study and evaluated 20 papers published between 2010 and 2019. They reported that only eight papers in their study focused on RESTful web services testing; however, we noticed that these papers do not focus on RESTful web services. For example, one of these eight papers belongs to Nabil [10], who focused on specifications that are used to generate test cases for web services testing. Nabil [10] stated that WSDL is the most used specification according to his systematic review and specified that only a few papers had an empirical evaluation to check the effectiveness of the proposed approach. In this paper by Nabil [10], we did not encounter any reference to the RESTful web service testing, though Ghani et al. [9] marked this paper as an example of the RESTful web service testing. Another paper specified by Ghani et al. [9] is the paper of Tosi and Morasca [11]; however, their actual focus is not the testing of web services. Instead, they systematically reviewed the studies that semantically annotate web services. Another paper specified as a RESTful web service testing paper by Ghani et al. [9] is the paper of Rusli et al. [12]; however, there is no specific information on RESTful web services in this paper. Rusli et al. [12] performed a systematic mapping study on web services' composition testing studies such as mutation testing and regression testing. The remaining five papers [13–17] mentioned as RESTful web service testing papers in the paper of Ghani et al. [9] were not related to the testing of RESTful web services; as such, we do not present them in this section. For example, Eliya et al. [15] presented a tool for interoperability testing of web services; however, there was no information on RESTful web services in their paper.

Barbir et al. [18] presented challenges of testing web services in SOA implementations and mainly focused on security aspects. They have listed the following web services' common threats: message alteration, confidentiality, falsified messages, a man in the middle, principal spoofing, forged claims, a replay of message parts, replay, and denial of service. They also listed the following requirements for end-to-end security for web services: mutual authentication, authorization to access resources, data integrity and confidentiality,

end-to-end integrity and confidentiality of messages, the integrity of transactions and communications, audit records and mechanisms, and distributed enforcement of security policy. However, their focus was not the testing of RESTful web services.

3. Research Methodology

The following subsections explain the research methodology used in this research. Figure 1 shows the main steps of the research methodology.

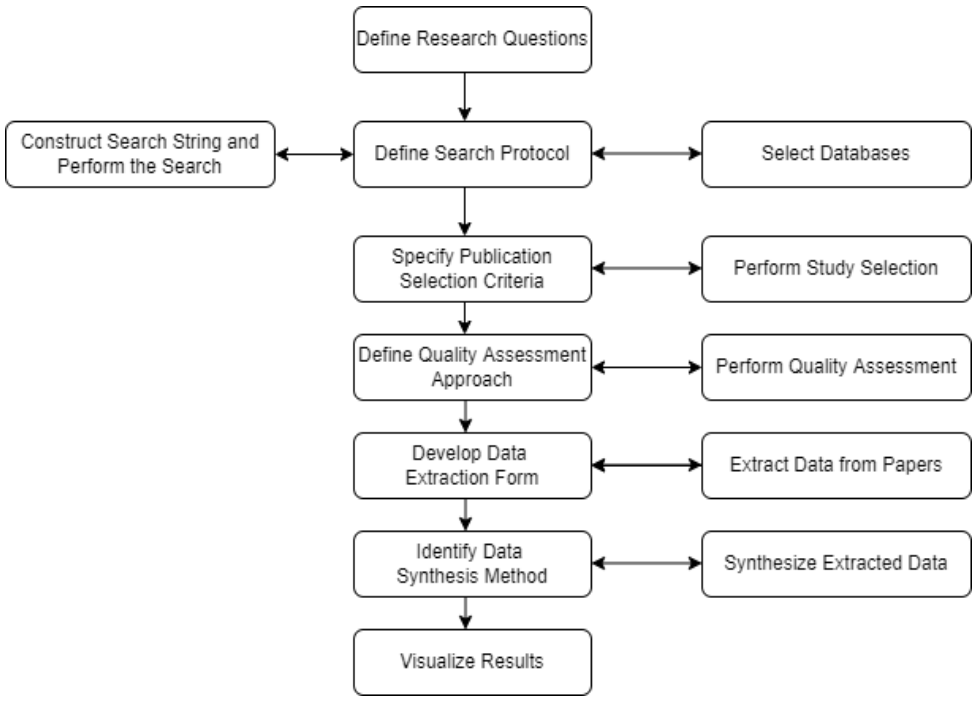

Figure 1. Flowchart of the methodology.

3.1. Review Protocol

Review protocol definition is the first and foremost important step before conducting the SLR research. This was carried out by incorporating the suggestions and guidelines defined by Kitchenham et al. [19]. The first step is to define the research questions. Once the research questions are ready, scientific databases are searched to get the relevant research papers. In our study, we used the following databases:

- IEEE Xplore;
- Science Direct;
- Scopus;
- Web of Science;
- Springer Link;
- Wiley;
- Google Scholar (for the purpose of forward snowballing).

A unique aspect related to our research is that it is a thriving topic. Hence, not many studies have been performed up to now. Databases that were used returned similar research papers. We included studies that address the scope of this research in our review. Later, required data addressing the relevant research questions were extracted together with bibliometric data. In brief, our approach has three main phases: review planning, review execution, and review reporting.

The first phase begins with identifying and laying out the research questions, followed by protocol development. This leads to the database selection, whose input would be search criteria (a.k.a., search strings). Next comes defining the selection criteria for the

publications. In the end, the protocol is refined one last time to check the suitability of the review protocol.

The second phase is the actual review conduction. Conducting the review involved the selection of publications from the relevant scientific databases. Once the publications were selected, data were extracted that resulted in information related to authors, publication year, and type (a.k.a., bibliometric data), and especially information related to the research questions. This was followed by a synthesis process to discuss the relevancy of the papers according to the context and create different categories to answer the research questions. The final phase included the documentation of results and responses to research questions.

3.2. Research Questions

In this SLR, in-depth details are inferred from the studies that have been published related to the field of RESTful APIs' testing methodologies and unit test generation. We laid out the following four research questions (RQs):

- RQ-1: What are the main challenges in generating unit tests for RESTful APIs?
- RQ-2: What are the code coverage concerns when it comes to testing RESTful APIs?
- RQ-3: What solutions are currently available to meet testing and unit test generation challenges?
- RQ-4: What support do solutions provide for authentication-enabled RESTful APIs' testing and unit test generation?

3.3. Search Strategy

Searching for relevant studies was focused on testing methodologies and unit test generation for RESTful APIs. There are several studies published related to RESTful APIs in general. Those papers describe other aspects like efficient development of RESTful APIs, integration between applications and RESTful APIs, and so on. However, a limited number of studies have been published on the testing part of RESTful APIs, as this is one of the thriving topics these days. Based on this, several studies were out of the scope of this research, and thus were left out. The search started using the keyword "RESTful API Testing". The resulting articles were gathered. The abstract and conclusion were read to assess the relevance of the studies. This was achieved by applying study selection criteria. Later, more detailed and complex search keywords were used to make sure that no related study was missed. Ultimately, the used search keyword combination was as follows:

((“RESTful APIs Testing” OR “RESTful APIs Unit Testing”) AND “RESTful APIs” AND (“Web API Testing”) AND (“RESTful Web Services” OR “REST APIs Blackbox Testing”)).

The search resulted in 16 papers, as shown in Table 1. Most of the papers were retrieved from IEEE and Scopus databases. As Scopus is a meta-database that indexes several databases, different papers from several publishers were covered.

Table 1. Number of papers retrieved per database.

Database	Numberof Papers
IEEE Xplore	7
Scopus	6
Google Scholar	3

3.4. Exclusion Criteria

The exclusion criteria were applied to the identified studies to exclude the irrelevant papers that are not required for this SLR. These criteria are presented as follows:

1. Publications that are not directly related to the RESTful API testing;
2. Non-English publications;
3. Duplicate publications;
4. Publications with only abstract;
5. Review or survey paper (i.e., secondary studies);

6. Publication published earlier than 2011.

As this is an emerging topic, we found a limited number of studies that focused on RESTful API testing. As shown in Figure 2, the number of papers has been increasing recently, thus research in this field is considered timely.

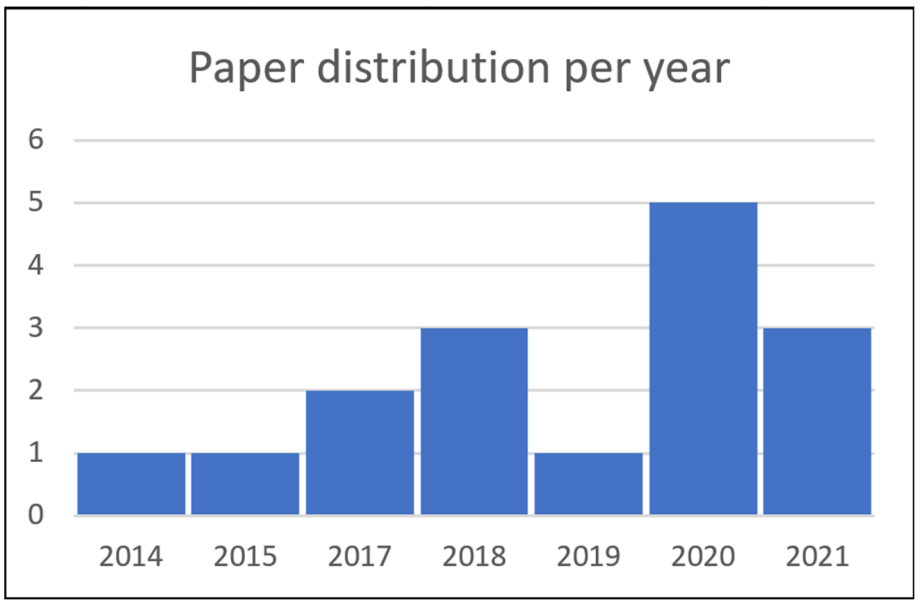

Figure 2. Distribution of the selected publications per year.

To respond to the four research questions, required data were extracted from papers and synthesized in a way that the questions are addressed adequately. While the data were being extracted from each article, a two-part focus was in place. First, we made sure that the selected studies are aligned with the criteria defined, and second, that data should be able to respond to research questions. The extracted and synthesized data were used to formulate answers to the research questions, and the results are laid out and discussed in the forthcoming sections.

3.5. Quality Score

Once the papers were retrieved, they were passed through a quality assessment process to make sure that only high-quality papers were included in this review. The quality assessment is based on the quality assessment questions inferred from a study by Kitchenham et al. [19].

A publication was scored based on the answer to each question as per the following rules:

- 1: if the answer is yes
- 0: if the answer is no
- 0.5: if the answer is somewhat

The following questions were included in the assessment:

1. Have the aims been clearly stated in the study?
2. Has the study scope been clearly defined?
3. Are the study variables dependable?
4. Does the study documentation cover the process of research sufficiently?
5. Have the defined questions been answered effectively?
6. Does the study present negative findings as well?
7. Are the major outcomes related to soundness and dependability listed?
8. Does the conclusion coincide with the study aims?

Figure 3 shows the paper distribution based on quality score. All the papers had a score more than or equal to 4, which is the threshold value to include the papers. As such, all the papers were considered high-quality and none of them were excluded in this review.

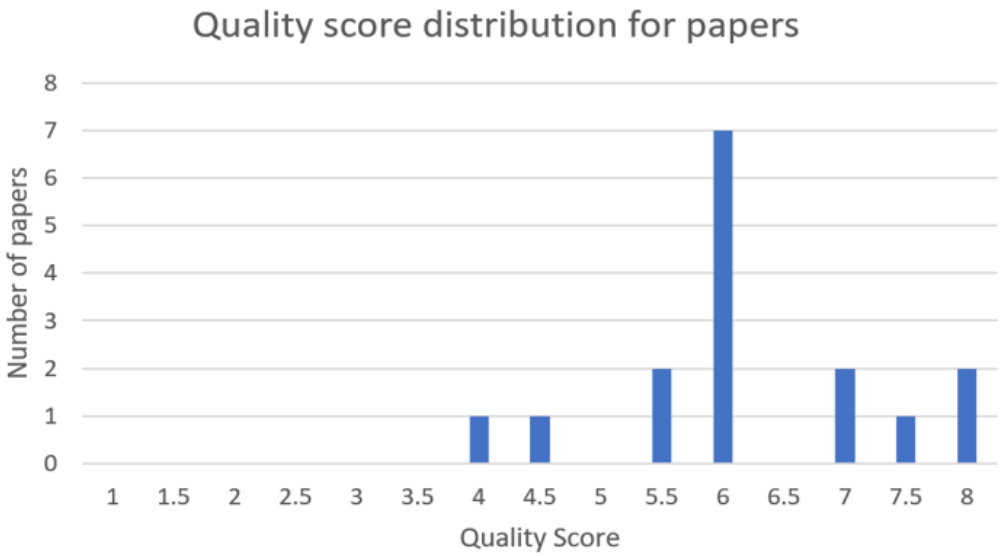

Figure 3. Paper distribution based on quality score.

3.6. Data Synthesis

Data synthesis performs aggregation on the gathered data to effectively come up with the answers to the research questions. Some research questions in this article require us to produce categorical outputs by following a quantitative data analysis such as available solutions. However, some questions such as the main challenges being faced while testing RESTful APIs require using qualitative data synthesis. For identification of the main challenges, textual descriptions and content analysis was required. As such, we performed qualitative data synthesis in this case. We also studied the methodologies, tools, and frameworks being used to solve the issues being faced. Later, we used the extracted data to categorize these solutions into different categories using quantitative data synthesis. All supportive facts were gathered from the selected studies and used to craft the answers to research questions.

4. Results

This section presents the responses to the research questions in each sub-section. Table 2 shows the selected publications along with the title, year, and approach used for testing RESTful APIs.

Table 2. Selected papers.

Ref.	Title	Testing Approach	Year
[20]	Property-based Testing of JSON based Web Services	Behavioral testing of RESTful APIs	2014
[21]	Model-driven Testing of RESTful APIs	Abstract model-based testing	2015
[22]	RESTful API Automated Test Case Generation	White box testing using EVOMASTER	2017
[23]	Study on REST API Test Model Supporting Web Service Integration	Expressive description language-based testing	2017

Table 2. *Cont.*

Ref.	Title	Testing Approach	Year
[24]	Automatic Generation of Test Cases for REST APIs: a Specification-Based Approach	Automated testing using specification-based generated test	2018
[25]	Metamorphic Testing of RESTful Web APIs	Metamorphic relations-based testing	2018
[26]	Automated API Testing	Testing based on parallel API execution and sequenced API calls	2018
[27]	RESTler: Stateful REST API Fuzzing	Producer–consumer dependencies and dynamic response feedback-based testing	2019
[28]	QuickREST: Property-based Test Generation of OpenAPI-Described RESTful APIs	Automatic property-based testing based on OpenAPI documents	2020
[29]	AI-Driven Web API Testing	AI-based testing for RESTful APIs	2020
[30]	A simple, lightweight framework for testing RESTful services with TTCN-3	Communication technology-independent testing using test specification language (TTCN-3)	2020
[31]	Differential Regression Testing for REST APIs	Regression testing targeting contract and service changes	2020
[32]	RESTTESTGEN: Automated Black-Box Testing of RESTful APIs	Blackbox testing using a generated unit test based on RESTful API's standard document (OpenAPI)	2020
[33]	Deep Learning-Based Prediction of Test Input Validity for RESTful APIs	Deep learning-based testing approach by predicting inter-parameter dependencies and validity of input request	2021
[34]	RESTest: Automated Black-Box Testing of RESTfulWeb APIs	Testing using constraint-based and fuzzing oriented generated input	2021
[35]	A Black Box Tool for Robustness Testing of REST Services	Robustness testing based on minimal information exposed for RESTful APIs using bBOXRT	2021

RESTful API testing has been regarded as a thriving topic. There have not been many studies published and this means that there is a lot of room for research. The main reason is that it is the flexibility of the methodology being used for developing, deploying, and consuming RESTful APIs. The lack of following one standard for describing the RESTful APIs is one of the hurdles. Hence, many studies are based on a few assumptions to start with.

To fulfill the objective of answering the inferred research questions, the primary studies were analyzed deeply and relevant data were extracted. The main focus was on the methodology, tools, and approach used for testing RESTful APIs.

The following were the parameters in line with the data of the study:

- The top-down approach is followed by each study.
- The methodology is followed for RESTful testing.
- Tools or frameworks are developed for RESTful APIs' testing.
- Any supportive facts are required to answer the research questions.

Based on the studied facts and analysis, the following sections discuss responses to research questions.

4.1. RQ1—What Are the Main Challenges in Generating Unit Tests for RESTful APIs?

Going through all primary studies and analyses yielded the fact that there are several challenges in generating unit tests for RESTful APIs. In both [25] and [27], the authors argue that effective and concise testing is the most challenging, whereas in [25], the authors argue that the lack of a test oracle is the biggest hamper to generating an effective unit test, as they are not always available. As for [26], the authors argue that the time taken is a major issue, and the authors of [33] argue that learning whether the input is valid or not before executing the REST call is crucial. The authors of [34] report that the lack of source code is the biggest issue. Looking at the challenges in a more consolidated approach for the sake of categorization, the following issues emerged as the most common ones:

a. Security plays a vital role in keeping the APIs safe and stable. However, at the same time, this poses a big challenge in generating unit tests and testing in general. The usual approach of unit test generation depends on an API description that is usually done using OpenAPI, or by using any other compatible descriptive language. This methodology is used by black-box testing because access to the source code is not available. The descriptive document does not contain any security information. Because of such a scenario, the unit test generation process is either limited to fewer unit tests or ends up with invalid unit tests because the required security information was not incorporated into the generation process.
b. Non-compliance to description document standards is another challenge in unit test generation. The most important catalyst for auto-generating unit tests for RESTful APIs is the description document. Different organizations are maintaining this document in different ways. XML, plain JSON, and OpenAPI are some of the document standards. RESTful APIs are quite flexible in this regard, which is another cause of the emergence of this challenge. Because there is no compulsion for maintaining a standard descriptive document for RESTful APIs to work, not much importance is given to this aspect. As a result, the standard tools and frameworks are sometimes unable to even proceed or end up with faulted unit tests.
c. An inconsistent descriptive document contributes to the list of challenges as well. Studies showed that, many times, organizations are not keeping the descriptive document for RESTful APIs up to date. As a result, the APIs that are in production are different than the descriptive document. Rapid changes and pressure to make modified APIs available as early as possible in production pave the way to the descriptive document being inconsistent. Hence, the unit test generation produces incomplete or inconsistent tests.
d. Complex input types are yet another challenge with which to deal. Modern-day RESTful APIs enable communication between hybrid networks, backends, and all possible kinds of devices. Data sent over could be of possibly any time, and ranging from simple integers and strings to a whole file or group of files. One of the core parts of any unit test generation framework is to infer test input values, which is very hard when complex types are involved. For this reason, several studies, such as [32], exclude the scope of complex types when generating unit tests.

4.2. RQ2—What Are the Code Coverage Concerns When It Comes to Testing RESTful APIs?

Code coverage is one of the main objectives of unit testing. It is an important metric to measure the effectiveness of unit tests. The maximum amount of code executed by the unit tests is always desirable. For achieving high code coverage, there should be a sufficient number of unit tests with a wider scale to reach all possible parts of the code. Failing to reach the recommended level of code coverage is a sign of either the fewer number of unit tests or the limited scale of the unit tests in terms of code execution. In RESTful APIs' case, achieving high code coverage could be a difficult task to achieve. One of the major reasons is reaching out to protected APIs. As explained in RQ1, owing to the nature of the RESTful APIs' document description, it becomes very difficult to generate unit tests for the protected endpoint. Therefore, the major part of the RESTful APIs goes without having unit tests

generated for them automatically. Manual testing and unit test generation are required for that, which always introduces the chance to miss some code, hence resulting in lower code coverage. In [25,27,28], the authors argued that code coverage is a good metric to measure the effectiveness of test cases and, as such, integrated it as a metric against which to measure their tools. Moreover, the authors of [11,12] mentioned the same issue. The authors of [25] noted in their testing, however, that the code coverage was relatively low compared with manual tests owing to the high string constraints. As for [27], the authors faced the challenge of a unique checksum and ID to archive high code coverage. The same challenge was faced by the authors of [28] when attempting to improve the code coverage with UUID as input. The authors of [32] completely exclude authenticated APIs' testing and unit test generation owing to the complexity explained earlier, hence they possibly ended up with lower code coverage.

4.3. RQ3—What Are the Existing Solutions Available to Meet Testing and Unit Test Generation Challenges?

Different categorical solutions have been proposed and used in the selected primary studies. Those fall in the following categories:

- Tools;
- Technique/method;
- Framework/model.

Figure 4 shows the contribution classification of the primary studies. According to Figure 4, most of the papers are proposing a methodology/approach and the least number of contributions is related to the framework/model. This indicates that researchers mostly prefer proposing a methodology/approach instead of a model/framework in this field. However, the development of frameworks/models is as important as approaches/methodologies, and more research is needed to develop novel models/frameworks.

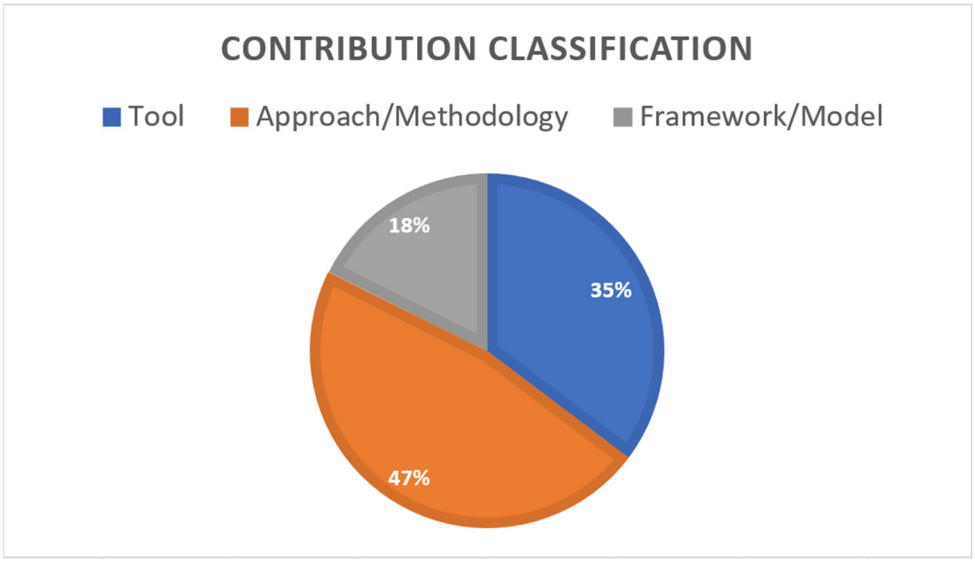

Figure 4. Contribution classification.

Table 3 describes where each primary study falls in terms of the proposed solution.

Table 3. Primary studies in terms of the proposed solution.

Ref.	Title	Tool	Approach/ Methodology	Framework/ Model
[20]	Property-based Testing of JSON based Web Services	☐	☒	☐
[21]	Model-driven Testing of RESTful APIs	☐	☒	☐
[22]	RESTful API Automated Test Case Generation	☒	☐	☐
[23]	Study on REST API Test Model Supporting Web Service Integration	☐	☐	☒
[24]	Automatic Generation of Test Cases for REST APIs: a Specification-Based Approach	☐	☒	☐
[25]	Metamorphic Testing of RESTful Web APIs	☐	☒	☐
[26]	Automated API Testing	☒	☐	☐
[27]	RESTler: Stateful REST API Fuzzing	☒	☐	☐
[28]	QuickREST: Property-based Test Generation of OpenAPI-Described RESTful APIs	☒	☒	☐
[29]	AI-Driven Web API Testing	☐	☒	☐
[30]	A simple, lightweight framework for testing RESTful services with TTCN-3	☐	☐	☒
[31]	Differential Regression Testing for REST APIs	☐	☒	☐
[32]	RESTTESTGEN: Automated Black-Box Testing of RESTful APIs	☒	☐	☐
[33]	Deep Learning-Based Prediction of Test Input Validity for RESTful APIs	☐	☒	☐
[34]	RESTest: Automated Black-Box Testing of RESTfulWeb APIs	☐	☐	☒
[35]	A Black Box Tool for Robustness Testing of REST Services	☒	☐	☐

4.4. RQ4—What Support Do Solutions Provide for Authentication-Enabled RESTful APIs' Testing and Unit Test Generation?

Developing authentication for RESTful APIs has several methodologies to follow. Those RESTful APIs are known as protected ones, which cannot be accessed by an anonymous user. The login process must be completed successfully first in order to get the identification, which can be then used to access the protected part of the RESTful APIs' endpoint collection. The variance is quite high when it comes to actual implementation. Based on the analysis of primary studies, it can be inferred that it is extremely difficult to come up with a solution that takes care of that aspect. Several studies excluded unit test generation for protected (i.e., authentication enabled) RESTful APIs' endpoints. Some of the common authentication implementation methods for RESTful APIs are described as follows:

- Token-based authentication: After successful authentication (i.e., log in using username and password), a long identification string is assigned to the user session, which is called a token. This is usually encrypted and sent back and forth manually between request–response sessions. The creation and validating mechanisms of tokens totally depend on the developer or organization; therefore, it can be done in many different ways and, as such, there is no strict standard. This poses a huge problem when trying to incorporate protected RESTful APIs into test generation. Every token usually has a validity time limit, after which it expires and a new token is required.
- Cookie-based authentication: This mechanism depends on generating a piece of authentication information called a cookie, which is sent back and forth between

request–response sessions like a token-based approach. However, the creation of the cookie depends on the framework being used for RESTful API development. A cookie is created after successful login using credentials and has an expiry time. There are many options available to choose the framework.

- API keys-based authentication: API keys-based authentication does not require the use of credentials (i.e., username and password) to be authenticated. Instead, we need to use an already issued secret API key from the host of RESTful APIs to be able to use its services. The key is issued either freely or by using paid membership, for example, using map services or social media services. As this is a secret key and is usually not shared, it would be impossible to create unit tests for such services and test them appropriately.

During our analysis, we did not come across any study that proposes a solution that takes care of the protected RESTful APIs when it comes to unit test generation, which implies that a major part of the RESTful APIs would have to be manually tested using manually generated unit tests. This also helps to infer that developing an authentication-enabled testing framework for RESTful APIs is an area with a huge scope of research.

5. Discussion and Threats to Validity

This section presents the general discussion, and the threats to validity are explained.

5.1. General Discussion

In this SLR, we systematically reviewed the available literature on RESTful APIs' testing methodologies. This study resulted in providing the challenges, problems, and available solutions and approaches associated with the testing domain for RESTful APIs. The presented results intend to classify and categorize the available solutions. The classification is based on the approach being used. This laid forth the path for further work on those approaches and the implementation of further tools based on the presented models and frameworks.

One of the core aspects of this review is to find out solutions addressing authentication when it comes to RESTful API testing. This is because most of the RESTful API endpoints are protected and, if not covered by auto-generation of unit tests, code coverage can be affected. This would result in insufficient and unreliable testing and code coverage. We came across no complete solution that falls into the autogenerate unit test-based testing category, which covers authentication, which means that there is a lot of dependence on manual testing, which always has a probability of low code coverage.

During the analysis, we found that the selected studies use a common approach to look into the description document of the REST APIs for unit test generation. The open API (a.k.a., swagger) format is one of the most commonly followed formats for describing REST APIs. Because of this, and the fact that authenticated REST APIs endpoints are not covered by any of the approaches, the studies are understood to have the same level of performance for code coverage. Fault detection, on the other hand, is best achieved by RESTTESTGEN: Automated Black-Box Testing of RESTful APIs [32]. The study has tested some of the well-known publicly available REST APIs. The statistics are shown to draw the conclusion clearly.

There are many situations in which REST APIs are hosted on-premise and in the cloud, but these are not accessible publicly because they are used for internal systems. On the other hand, several instances of REST APIs are publicly available in terms of use. In either case, access to the source code is usually not available. This is one of the common reasons that black-box testing becomes a viable option. The advantage of black-box testing is that the proposed solution can easily generate and run unit tests by just analyzing the description of REST APIs. The disadvantage is that the exact location or reason for the fault would not be easy to find out, especially when the implementation hides the actual cause of the error in the output. White-box testing has a clear advantage in terms of finding the actual cause and location of the error or bug. However, access to the source code is a must for this kind of

testing. The code coverage aspect remains the same in both testing methodologies. One of the main obstacles to achieving a high level of code coverage is to test authenticated REST APIs. A proposed solution can achieve a high level of code coverage if it is able to invoke and call all or max REST APIs' endpoints, using either black-box or white-box testing.

There are also additional techniques that can improve search-based test case generation for testing RESTful web services and aim at fully automating the evaluation of testing approaches. Martin-Lopez et al. [36] proposed a catalog of 10 test coverage criteria for RESTful APIs and a framework for the testing approach evaluation. They used two real-world APIs for the evaluation of their proposal and demonstrated that the coverage levels correlate with measurements of fault detection. Vosta [37] investigated how t-wise combinatorial testing works for REST APIs and reported that 1-wise, 2-wise, and 3-wise interaction testing techniques of REST APIs detect the same faults when mutation testing is executed on the test suite. Mutation testing was used to inject faults into the system under test. Zhang et al. [38] developed a search-based method, which is integrated into the Many Independent Objectives (MIO) evolutionary search algorithm. They evaluated the approach on seven open-source RESTful web services and demonstrated that resource-based sampling strategies achieve much higher performance compared with the baseline MIO.

5.2. Threats to Validity

Construct validity: There might be some publications available that were missed owing to the missed synonyms that could be used for the search. As some terms were used slightly differently in some papers, some observations might have been affected.

Internal validity: Paper selection bias was minimized because all authors were involved during this procedure and a consensus was reached while evaluating the papers.

External validity: One major factor that can obstruct the validity of this SLR is the number of studies. As the topic is still an emerging topic, there are not many published studies available. There could be some frameworks, tools, and models in practice in the industry that are not formally published as research papers.

Conclusion validity: To minimize the conclusion validity threat, authors drew conclusions based on discussions and common understanding; therefore, the individual basis on the interpretation of results was reduced in this research.

6. REST APIs' Open Issues, Challenges, and Research Directions

REST APIs are very diverse, both in terms of development and testing. Various methodologies, frameworks, and approaches have been put in practice by various organizations. Because of this diversity and flexibility, organizations do not have to follow a specific framework or methodology. This has led to challenges related specifically to testing. One of the major challenges that we found was authentication support in generating unit tests for REST APIs. As there are many different authentication techniques used these days, this makes it difficult to automate unit test generation. Moreover, organizations do not usually share the way in which authentication has been implemented, and it is continuously evolving; therefore, it has become even harder to come up with a generic unit test generation framework that covers authenticated REST APIs. Some recent papers have proposed this for future research. As authenticated or protected REST APIs' endpoints are not included in test case generation, the achieved code coverage is low—this is the second most prevailing challenge. This has set a future research direction focused on studying the possible common and baseline standard being followed when authentication is implemented. Based on this study, the authentication support for REST APIs' testing is desired to be designed as a pluggable module in the proposed solution.

7. Conclusions

In this paper, we investigated literature that covers different aspects of RESTful API testing from test generation with and without an oracle to black-box testing, where there is

no access to the source code; to input generation, be it using fuzzing or an oracle; and to output prediction using machine learning to test and feedback automation.

As covered in the results section, there have been several challenges faced by the researchers to come up with an effective framework for unit test generation and testing in general. We can infer from the discussion that following universal standards while developing and describing RESTful APIs will make even existing frameworks and methodologies more effective. This is because of the fact that testing and unit test generation strongly depends on the standard document description of the RESTful API (e.g., XML, plain JSON, or OpenAPI (a.k.a., swagger)). As a first step, this standardization will alleviate the first level of challenge. Secondly, this review also shows that keeping the descriptive document consistent is equally important. Thirdly, missing comprehensive support for authentication-enabled RESTful APIs' testing is still an open area to perform research.

We believe that this article will help pave the way for further RESTful API testing-based research. Machine learning is being used to predict the output of a given test case input. Using machine learning to predict the code coverage of a given test suit would be an interesting and important research topic. It can help determine which test suites to keep and which to update or discard owing to its code coverage, which is paramount for the test suite's maintainability, along with security-focused testing, as this paper already identified. There is a serious lack of security-friendly RESTful API test suites, which has the added sensitivity that such an API calls would manage sensitive or secured parts of the service and, as such, unexpected behavior of such an API is a security threat in itself.

Further work in the future would be to enhance the outcomes of this SLR and focus on the role of machine learning being applied to improve the testing solutions, along with authentication enablement for RESTful APIs.

Author Contributions: Conceptualization: A.E., M.A.M.E.A. and C.C.; data curation: A.E. and M.A.M.E.A.; formal analysis: A.E., M.A.M.E.A., C.C. and D.M.; investigation: A.E., M.A.M.E.A., C.C. and D.M.; methodology: A.E., M.A.M.E.A., C.C. and D.M.; project administration: C.C.; resources: A.E., M.A.M.E.A., C.C. and D.M.; supervision: C.C.; validation: A.E., M.A.M.E.A., C.C. and D.M.; writing—original draft: A.E., M.A.M.E.A., C.C. and D.M.; writing—review and editing: A.E., M.A.M.E.A., C.C. and D.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by the Norwegian University of Science and Technology, Norway for the support of Open Access fund.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors thank their universities for scientific database subscriptions and infrastructure support that enabled this collaborative research.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

Acronym	Definition
REST	Representational state transfer
API	Application programming interfaces
SLR	Systematic literature review
WWW	World Wide Web
HTTP	Hypertext transfer protocol
URI	Uniform resource identifier
JSON	JavaScript object notation

References

1. Li, L.; Chou, W.; Zhou, W.; Luo, M. Design Patterns and Extensibility of REST API for Networking Applications. *IEEE Trans. Netw. Serv. Manag.* **2016**, *13*, 154–167. [CrossRef]
2. Neumann, A.; Laranjeiro, N.; Bernardino, J. An Analysis of Public REST Web Service APIs. *IEEE Trans. Serv. Comput.* **2018**, *14*, 957–970. [CrossRef]
3. Pahl, C.; Jamshidi, P. Microservices: A Systematic Mapping Study. In Proceedings of the 6th International Conference on Cloud Computing and Services Science—Volume 1 and 2, Rome, Italy, 23–25 April; pp. 137–146. [CrossRef]
4. Khare, R.; Taylor, R. Extending the Representational State Transfer (REST) architectural style for decentralized systems. In Proceedings of the 26th International Conference on Software Engineering, Edinburgh, UK, 28 May 2004; pp. 428–437. [CrossRef]
5. Pautasso, C.; Zimmermann, O.; Leymann, F. Restful web services vs. "big" web services: Making the Right Architectural Decision. In Proceedings of the 17th International Conference on World Wide Web, Beijing, China, 21–25 April 2008; pp. 805–814. [CrossRef]
6. Ed-Douibi, H.; Izquierdo, J.L.C.; Cabot, J. OpenAPItoUML: A tool to generate UML models from OpenAPI definitions. In Proceedings of the International Conference on Web Engineering, Cáceres, Spain, 5–8 June 2018; Springer: Cham, Switzerland, 2018; pp. 487–491.
7. Fielding, R.T.; Taylor, R.N. *Architectural Styles and the Design of Network-Based Software Architectures*; University of California: Irvine, CA, USA, 2008.
8. Mogul, J.C.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. Hypertext Transfer Protocol–HTTP/1.1. Terminology 1.3. 1999. Available online: https://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf (accessed on 20 November 2021).
9. Ghani, I.; Wan-Kadir WM, N.; Mustafa, A. Web service testing techniques: A systematic literature review. *Int. J. Adv. Comput. Sci. Appl.* **2019**, *10*, 443–458. [CrossRef]
10. Nabil, E.I. Specifications for web services testing: A systematic review. In Proceedings of the 2015 IEEE World Congress on Services, Washington, DC, USA, 27 June–2 July 2015; pp. 152–159.
11. Tosi, D.; Morasca, S. Supporting the semi-automatic semantic annotation of web services: A systematic literature review. *Inf. Softw. Technol.* **2015**, *61*, 16–32. [CrossRef]
12. Rusli, H.M.; Ibrahim, S.; Puteh, M. Testing Web Services Composition: A Mapping Study. *Commun. IBIMA* **2011**, *2011*, 1–12. [CrossRef]
13. Chen, Z.; Shen, L.; Li, F. Exploiting Web service geographical neighborhood for collaborative QoS prediction. *Futur. Gener. Comput. Syst.* **2017**, *68*, 248–259. [CrossRef]
14. Silic, M.; Delac, G.; Krka, I.; Srbljic, S. Scalable and Accurate Prediction of Availability of Atomic Web Services. *IEEE Trans. Serv. Comput.* **2013**, *7*, 252–264. [CrossRef]
15. Elia, I.A.; Laranjeiro, N.; Vieira, M. ITWS: An extensible tool for interoperability testing of web services. In Proceedings of the 2014 IEEE International Conference on Web Services, Anchorage, AK, USA, 27 June–2 July 2014; pp. 409–416.
16. Nacer, A.A.; Bessai, K.; Youcef, S.; Godart, C. A Multi-criteria Based Approach for Web Service Selection Using Quality of Service (QoS). In Proceedings of the IEEE International Conference on Services Computing, New York City, NY, USA, 27 June–2 July 2015; pp. 570–577. [CrossRef]
17. Selvam, R.; Senthilkumar, A. Webservice based vulnerability testing framework. In Proceedings of the 2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE), Coimbatore, India, 6–8 March 2014; pp. 1–6.
18. Barbir, A.; Hobbs, C.; Bertino, E.; Hirsch, F.; Martino, L. Challenges of testing web services and security in SOA implementations. In *Test and Analysis of Web Services*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 395–440.
19. Kitchenham, B.; Charters, S. Guidelines for performing Systematic Literature Reviews in Software. *Engineer. Techn. Rep.* **2007**, *5*, 1–57.
20. Fredlund, L.A.; Earle, C.B.; Herranz, A.; Marino, J. Property-Based Testing of JSON Based Web Services. In Proceedings of the IEEE International Conference on Web Services, Anchorage, AK, USA, 27 June–2 July 2014; pp. 704–707. [CrossRef]
21. Fertig, T.; Braun, P. Model-driven Testing of RESTful APIs. In Proceedings of the 24th International Conference on World Wide Web, Florence, Italy, 18–22 May 2015; pp. 1497–1502. [CrossRef]
22. Arcuri, A. RESTful API Automated Test Case Generation. In Proceedings of the IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, Czech Republic, 25–29 July 2017; pp. 9–20. [CrossRef]
23. Wenhui, H.; Yu, H.; Xueyang, L.; Chen, X. Study on REST API Test Model Supporting Web Service Integration. In Proceedings of the IEEE 3rd International Conference on Big Data Security on Cloud (Bigdatasecurity), IEEE International Conference on High Performance and Smart Computing (Hpsc), and IEEE International Conference on Intelligent Data and Security (ids), Beijing, China, 26–28 May 2017; pp. 133–138. [CrossRef]
24. Ed-Douibi, H.; Izquierdo, J.L.C.; Cabot, J. Automatic Generation of Test Cases for REST APIs: A Specification-Based Approach. In Proceedings of the IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC), Stockholm, Sweden, 16–19 October 2018; pp. 181–190. [CrossRef]
25. Segura, S.; Parejo, J.A.; Troya, J.; Ruiz-Cortes, A. Metamorphic Testing of RESTful Web APIs. *IEEE Trans. Softw. Eng.* **2017**, *44*, 1083–1099. [CrossRef]
26. Isha, A.; Sharma, A.; Revathi, M. Automated API Testing. In Proceedings of the 3rd International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, 15–16 November 2018; pp. 788–791. [CrossRef]

27. Atlidakis, V.; Godefroid, P.; Polishchuk, M. RESTler: Stateful REST API Fuzzing. In Proceedings of the 41st International Conference on Software Engineering, Montreal, QC, Canada, 25–29 May 2019; pp. 748–758. [CrossRef]
28. Karlsson, S.; Causevic, A.; Sundmark, D. QuickREST: Property-based Test Generation of OpenAPI-Described RESTful APIs. In Proceedings of the IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), Porto, Portugal, 24–28 October 2020; pp. 131–141. [CrossRef]
29. Martin-Lopez, A. AI-Driven Web API Testing. In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Seoul, Korea, 5–11 October 2020; pp. 202–205. Available online: https://doi.ieeecomputersociety.org/ (accessed on 20 November 2021).
30. Vassiliou-Gioles, T. A simple, lightweight framework for testing RESTful services with TTCN-2020. In Proceedings of the IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Vilnius, Lithuania, 11–14 December 2020; pp. 498–505. [CrossRef]
31. Godefroid, P.; Lehmann, D.; Polishchuk, M. Differential regression testing for REST APIs. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, New York, NY, USA, 18–22 July 2020; pp. 312–323. [CrossRef]
32. Viglianisi, E.; Dallago, M.; Ceccato, M. RESTTESTGEN: Automated Black-Box Testing of RESTful APIs. In Proceedings of the IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), Porto, Portugal, 24–28 October 2020; pp. 142–152. [CrossRef]
33. Mirabella, A.G.; Martin-Lopez, A.; Segura, S.; Valencia-Cabrera, L.; Ruiz-Cortes, A. Deep Learning-Based Prediction of Test Input Validity for RESTful APIs. In Proceedings of the IEEE/ACM Third International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest), Madrid, Spain, 1 June 2021; pp. 9–16. [CrossRef]
34. Martin-Lopez, A.; Segura, S.; Ruiz-Cortés, A. 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, NY, USA, 11–17 July 2021; pp. 682–685. [CrossRef]
35. Laranjeiro, N.; Agnelo, J.; Bernardino, J. A Black Box Tool for Robustness Testing of REST Services. *IEEE Access* **2021**, *9*, 24738–24754. [CrossRef]
36. Martin-Lopez, A.; Segura, S.; Ruiz-Cortés, A. Test coverage criteria for RESTful web APIs. In Proceedings of the 10th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation, New York, NY, USA, 26–27 August 2019; pp. 15–21.
37. Vosta, D. Evaluation of the T-Wise Approach for Testing REST APIs. Master's Thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2020.
38. Zhang, M.; Marculescu, B.; Arcuri, A. Resource-based test case generation for restful web services. In Proceedings of the genetic and evolutionary computation conference, Prage, Czech Republic, 13–17 July 2019; pp. 1426–1434.