Testing RESTful APIs – Use Case: RESTful API for Solving Multidimensional Time–Independent Schrödinger Equation

Davor Dimoski[1,*,†], Bojana Koteska[1,*,†], Ljupco Pejov[2,†] and Anastas Mishev[1,†]

[1]Ss. Cyril and Methodius University, Faculty of Computer Science and Engineering, Skopje, North Macedonia
[2]Ss. Cyril and Methodius University, Faculty of Natural Sciences and Mathematics, Skopje, North Macedonia

Abstract
Many of the enterprise software applications today are developed using a service–oriented architecture (SOA) where web services, especially microservices play a fundamental role. A RESTful web service is a lightweight and scalable web service that provide data over Internet via an API using the HTTP protocol. Testing of a RESTful API is challenging because the user inputs are sequences of HTTP requests to a remote server and the server responses are also in HTTP format. In this paper, we aim to analyse ten different tools for testing RESTful API web services and provide tool comparisons first based on the request support and second based on multiple functionalities – implementation, platform, languages, testing techniques and supported imports. Taking into account this analysis and tools possibilities, we perform testing on RESTful API service for solving multidimensional time–independent Schrödinger equation using Hermite DVR approach. The findings show that not all tests could be run on all testing tools since some of them do not offer ready–made assertions for validating the response time or HTTP body content or there is no option for validating the HTTP body.

Keywords
REST API, RESTful web service, Testing tools, Schrödinger equation

1. Introduction

In the distant past, developers have tried to find a way to provide interoperability between systems and find ways to utilize the reuse of software [1]. Object–oriented technology provided minor success in this field, but it could not fully succeed due to the hard communication between different technologies. To bridge the gaps that object–oriented technology couldn't, web services came into play.

Web services are interfaces that describe a set of operations that can be used to exchange

SQAMIA 2022: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 11—-14, 2022, Novi Sad, Serbia
*Corresponding author.
†These authors contributed equally.

davor.dimoski@students.finki.ukim.mk (D. Dimoski); bojana.koteska@finki.ukim.mk (B. Koteska); ljupcop@pmf.ukim.mk (L. Pejov); anastas.mishev@finki.ukim.mk (A. Mishev)
https://www.finki.ukim.mk/en/content/bojana--koteska--phd (B. Koteska); https://www.finki.ukim.mk/en/staff/anastas--mishev (A. Mishev)
0000-0001-6118-9044 (B. Koteska)

information over the HyperText Transfer Protocol (HTTP) between a client and a server [2]. In other words, they are a standardized medium for communication between applications.

REpresentational State Transfer (REST) architecture was introduced in 2000 by Roy Thomas Fielding and it relies on Uniform Resource Identifiers (URIs) for detection and interaction with resources [3]. A URI usually represents a document that holds the current state of the resource. RESTful web services are services based on the REST architecture and they are lightweight, usually used for basic, ad–hoc integration scenarios [4]. Unlike SOAP, RESTful web services do not require XML definitions and they additionally support JSON, plain text and other formats. These web services use HTTP methods (GET, POST, PUT, DELETE etc.) to manipulate resources.

The goal of testing APIs is to validate the ways an API is expected to behave in terms of functionality, reliability, performance and security [5]. SOAP based testing is very often based on the Web Service Description Language (WSDL), but considering this is not applicable to RESTful APIs, the task becomes more complicated – REST is only a design pattern, so it doesn't follow any unified standards to describe the APIs. Test coverage criteria for RESTful APIs are usually divided in two parts: input criteria – related to the requests, and output criteria – related to the responses of the API [6].

Even though to the best of our knowledge there are not many papers that have examined the differences between several RESTful API testing tools, there are a few that describe the process of developing RESTful API testing tools and automated testing approach. In [7], the author presents a case study of automating tests without using a GUI library. In [8], Venkatraj et al. develop an automation framework based on Groovy for REST API testing. Atlidakis et al. have written a paper on developing REST–ler – an intelligent automatic REST API security testing tool that generates tests by analysing the Swagger specification of the API [9]. Arcuri presented a fully automated white–box RESTful service testing approach, where test cases are automatically generated using an evolutionary algorithm. The author also developed an open–source tool for testing, called EvoMaster [10, 11].

The aim of this paper is to explore different RESTful API testing tools and examine the differences between the functionalities they offer. We will be discussing ten different RESTful API testing tools and we will examine them by testing a RESTful web service for solving multidimensional time–independent Schrödinger equation using Hermite DVR approach.

This paper is organized as follows: in Section 2 we provide an overview on the ten selected RESTful API testing tools that we are going to examine and draw a comparison table to see the differences between them; in Section 4 we describe our case study, testing approach and examine the results; and finally in Section 5 we draw conclusions.

2. Testing RESTful APIs

In this Section we give a short overview of several RESTful API Testing Tools and we provide tool comparison based on different criteria.

2.1. RESTful API Testing Tools

2.1.1. SoapUI

SoapUI [12] is a tool dedicated for testing web services – this includes both SOAP and RESTful APIs. This testing software offers two versions – a free, open source one with basic functionalities like SOAP and REST API testing, mocking, WSDL support and message assertions; and a pro version which contains additional functionalities not available in the free version, like data–driven testing, end–to–end testing support, native CI/CD integration support, etc. SoapUI allows for REST projects to be created either by entering a URI or by importing a WADL file. It supports the basic HTTP requests like POST, GET, PUT, and DELETE, but also HEAD, OPTIONS, TRACE, PATCH, PROPFIND, LOCK, UNLOCK, COPY and PURGE requests as well. Users can create test cases from the requests where assertions can be added. The assertions are divided in several categories, offering a variety of choices from checking a property's content to validating the HTTP status codes and checking security for sensitive information exposure.

2.1.2. Katalon Studio

Katalon Studio [13] is an automation tool that offers testing for SOAP and REST APIs, Web, Desktop and Mobile applications. It supports validation testing, functional testing, UI testing, security testing, load testing, runtime and error detection, penetration testing and fuzz testing. Katalon Studio also supports CI/CD integration and BDD (behavior driven development) with Cucumber files. When it comes to testing REST Web Services, upon a creation of an API/Web Services project, the user can manually create REST requests, or they can import Swagger, WSDL and Postman files or URLs. Katalon offers requests like GET, POST, PUT, DELETE, PATCH, HEAD, CONNECT, OPTIONS and TRACE. The project is divided into several sections, among which are Object Repository – where we can create a RESTful endpoint, Test Cases – stores test scenarios for the test objects, and Test Suites – a place where test cases can be stored and run at once. When executing a Test Suite, a report is automatically generated.

2.1.3. Postman

Postman [14] is a platform that supports all stages of an API lifecycle: development, testing, publishing and documenting. It offers automated testing and integration in the CI/CD (Continuous integration/Continuous delivery) pipeline, as well as exploratory testing. Postman allows creation of tests by manually creating a request or by importing a RAML/WADL/Open API/GraphQL Schema/DHC/cURL/Runscope file. This tool supports the following types of requests: GET, POST, PUT, PATCH, DELETE, COPY, HEAD, OPTIONS, LINK, UNLINK, PURGE, LOCK, UNLOCK, PROPFIND and VIEW. In exploratory testing, users can use scripts to send asynchronous requests, chain requests to create test scenarios and document the findings to share them with others. With automated testing, users can create custom test suites in JavaScript, parametrize requests, run the tests and debug.

2.1.4. Apache JMeter

Apache JMeter [15] is an open source software that allows load testing, functional testing and measuring performance. Although it was originally designed to test Web Applications, it has since expanded and now offers a wider range of functionalities. All tests in Apache JMeter must have a test plan, a thread group and at least one sampler. A test plan is the basis of any test and it describes a sequence of steps that JMeter will execute once it is run. The thread group offers control over the number of threads/users that will be used when executing the tests, the ramp–up period (in what time frame should all threads be executed) and the number of times the test should be executed. Samplers simply send requests to a server and wait for a response. This tool offers various assertions among which the most popular are Response and JSON assertions. The HTTP methods that Apache JMeter supports are: GET, POST, HEAD, PUT, OPTIONS, TRACE, DELETE, PATCH, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK, REPORT, MKCALENDAR and SEARCH. For load testing, there is a CLI Mode which allows loading tests from any Java compatible OS. As mentioned earlier, it supports multithreading and it offers CI support as well.

2.1.5. Rest–Assured

Rest–Assured [16] is a Java library that is used for testing XML–based and JSON–based Web Services. It supports POST, GET, PUT, DELETE, OPTIONS, PATCH and HEAD requests. To use the Rest–Assured library, it needs to be imported in the Maven or Gradle file in a Java project. It supports BDD given/when/then syntax and it also allows data–driven testing.

2.1.6. Assertible

Assertible [17] is a tool for testing and monitoring APIs and its focus is on automation. It supports the following HTTP requests: GET, PUT, POST, PATCH, DELETE, HEAD, OPTIONS. Web services can be manually created or imported with a Swagger file, a Postman collection file or by a curl command. Assertible allows monitoring uptime and performance and continuous integration as well. This tool utilizes the power of jq (command line JSON processor) to modify and manipulate variables before they are used for a test run. Assertible has a feature called encrypted variables which is a safe way to store sensitive data during testing.

2.1.7. Karate

Karate [18] is an open–source testing tool that allows automation, mocks, performance–testing and UI–automation. The most common usage is in a Java IDE, but it can also be used as a stand–alone executable. Karate supports the following requests: GET, POST, PUT, DELETE, PATCH, OPTIONS, HEAD, CONNECT and TRACE. Furthermore, Karate supports multi–threaded parallel execution. It has native support for YAML and CSV files which can be used for data–driven testing. It can easily be integrated in CI/CD.

2.1.8. Swagger Inspector

Swagger Inspector [19] is an open source tool built around the OpenAPI Specification for testing and documenting APIs. It supports importing Swagger/OAS 2, OAS 3 and WSDL files. It is a web–based editor (full support for Firefox and Google Chrome) and it requires installation of an extension in order to work properly. The history of requests is visible in the sidebar and you can pin them and add them to collections. OpenAPI 3.0 supports GET, POST, PUT, PATCH, DELETE, HEAD and OPTIONS requests.

2.1.9. Insomnia

Insomnia [20] is a cross–platform REST API client. It provides an intuitive approach to managing APIs with sending requests, defining environment variables, authentication helpers, etc. It supports GET, POST, DELETE, PUT, PATCH, OPTIONS, HEAD and additionally a custom HTTP method (common examples being LINK, UNLINK, FIND, PURGE). Insomnia supports environment variables, which are most commonly used when the user is using identical values across several different requests.

2.1.10. HttpMaster

HttpMaster [21] is a software development and testing tool for web applications and RESTful web services. HttpMaster offers support for the most common rest methods like GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, PATCH, but also a custom method can be specified. There is a functionality available for global and multi–valued parameters. It also utilizes Command Line Interface to automate execution of HttpMaster projects and it allows OpenAPI (Swagger) imports. Additionally, it supports load testing.

2.2. Tool Analysis and Discussion

Below, we present two comparison tables that show how each tool differs from the others. Table 1 shows the range of requests each tool supports (Fields marked with * represent that the methods are covered through a custom HTTP request), while in Table 2 there is a detailed tools comparison based on the following characteristics:

1. **Implementation** – how the tool is used is presented to the tester for usage:
    - IDE – Integrated Development Editor;
    - Framework;
    - Library;
    - CLI – Command Line Interface;
2. **Platform** – technologies/a framework of services the software relies on for standard operations;
3. **Languages** – languages used for writing test cases;
4. **Testing techniques** – the types of testing the tool supports;
5. **Supported imports** – the types of files that can be used to load already created tests.

Table 1
Comparison table for the support of different requests by each tool.

HTTP method / Testing tool	GET	POST	DELETE	PUT	PATCH	COPY	HEAD	OPTIONS	LINK	UNLINK	PURGE	LOCK	UNLOCK	PROPFIND	PROPPATCH	VIEW	CONNECT	TRACE	MKCOL	MKCALENDAR	REPORT	SEARCH	MOVE
SoapUI	x	x	x	x	x	x	x	x			x	x	x	x				x					
Katalon Studio	x	x	x	x	x		x	x									x	x					
Postman	x	x	x	x	x	x	x	x	x	x	x	x	x	x		x							
Apache JMeter	x	x	x	x	x	x	x	x				x	x	x	x			x	x	x	x	x	x
Rest-Assured	x	x	x	x	x		x	x															
Assertible	x	x	x	x	x																		
Karate	x	x	x	x	x		x	x									x	x					
Swagger Inspector	x	x	x	x	x		x	x															
Insomnia	x	x	x	x	x	*	x	x	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
HttpMaster	x	x	x	x	*	*	x	x	*	*	*	*	*	*	*	*	*	x	*	*	*	*	*

Table 2
Comparison table based on functionalities each testing tool offers.

Testing tool	Implementation	Platform	Languages	Testing techniques	Supported imports
SoapUI	Framework, IDE	Windows, macOS, Linux	Groovy, Javascript	functional, performance, interoperability, regression, mocking, automation, data-driven, security, load	WADL, URI
Katalon Studio	IDE	Windows, macOS, Linux	Groovy	validation, functional, UI, security, load, runtime error detection, penetration, fuzz	Open API, WSDL, Postman
Postman	IDE	Windows, macOS, Linux	Javascript	exploratory, automation	RAML, WADL, Open API , GraphQL Schema, Runscope file
Apache Jmeter	IDE, CLI	JVM	Java	performance, functional, load	/
Rest-Assured	Library	JVM	Java	automation, data-driven	/
Assertible	Web IDE	Web-based	UI-based commands	automation	Open API, Postman collection
Karate	Framework, CLI	Java DSL	Java	automation, mock, performance, UI automation	/
Swagger Inspector	Web IDE	SwaggerHub	/	manual	Open API, WSDL
Insomnia	IDE	Windows, macOS, Linux	/	manual	/
HttpMaster	IDE, CLI	Windows	UI-based commands	load, automation	Open API

The tools that have their own IDE are similar in the way that they are constructed. More or less, working on each of them is a similar experience. However, there is a difference in the fact that in most of the tools that have an IDE, aside from Postman, don't require any programming knowledge to construct basic test cases. For more advanced testing, Katalon, SoapUI and Apache JMeter offer the option to use the languages they support for writing more specific test cases.

Postman uses Javascript for constructing test cases.

One advantage Apache JMeter, SoapUI and Karate have over the other testing tools is the functionality of using concurrent threads – which allows load testing. This allows choosing how many threads/virtual users will make a request to the server.

When it comes to testing response duration, some tools fell short – they did not offer the option to validate the response time. Katalon Studio, SoapUI, Swagger Inspector, Insomnia and HttpMaster do not offer ready–made assertions for the response time. However, Katalon Studio and SoapUI still have the option available with manually scripting assertions.

The libraries that were used as testing tools – Rest–Assured and Karate, proved to be sufficient in conducting tests for our use case. It is important to note that the biggest advantage Karate has over Rest–Assured is that for validating payloads it supports assertions such as "contains", which isn't directly available with Rest–Assured. In order to use such assertion in Rest–Assured you need to use external libraries such as Hamcrest.

The free version of HttpMaster offers obscure validation options: only whether the response status code is 1xx, 2xx, 3xx for a successful call and 4xx or 5xx for an unsuccessful call. The tool did not allow us to run the assertions if the HTTP body of the response contains an opening bracket and the response time for the call.

Two of the tools – Insomnia and Swagger Inspector did not offer options for writing test cases. The only testing that can be done with them is manual – the tester themselves have to manually validate the response of the API call.

3. Use case – testing RESTful API for solving multidimensional time–independent Schrödinger equation using Hermite DVR approach

3.1. API description

To test the abilities of the aforementioned tools, we conducted testing on the Schrödinger API [22] – a RESTful web service for solving multidimensional time–independent Schrödinger equation using Hermite DVR approach. This RESTful web service provides a method for solution of one–dimensional, two–dimensional and three–dimensional time–independent Schrödinger equation based on the Gauss–Hermite Discrete Variable Representation (DVR) approach. Web service source code is based on a python module that solves one–dimensional potentials using a DVR method [23].

The solution of one–dimensional Schrödinger equation is illustrated in the case of following model potentials: **Morse potential; Simple Harmonic Oscillator (SHO) potential; Sombrero potential (Mexican hat); Woods–Saxon potential**.

Solutions of two–dimensional and three–dimensional Schrödinger equations are illustrated for the following two model potentials: multidimensional Morse potential and multidimensional SHO potential.

3.2. Test scenarios

The test scenarios that we will consider for testing this RESTful API are:

- Check the response on default values for parameters (happy path);
- Check the response on valid values for parameters (happy path);
- Check the response on invalid values for parameters (negative testing with invalid input);
- Check the response on large values for parameters (destructive testing).

When validating the response we consider the following response elements:

- **HTTP status code** – Validate the response status code (expected 200 for all scenarios, except when invalid values are entered, in which case we expect a response code of 400);
- **response payload** – Validate whether the body of the response contains an opening bracket "[" – all valid responses contain an opening and a closing bracket;
- **basic performance sanity** – Validate the response time of the request being less than 1000 ms.

The values that were used for each scenario are shown in Table 3.

3.3. Results

Table 4 contains the response information from the tests designed for each test scenario. The information in the table is drawn from the Postman tool.

In the test scenarios for default values, valid and large values we are expecting the response status code to be 200 OK – all of the responses to be valid, to contain an opening bracket "[" and additionally we expect the response time to be less than 1000ms. In the test scenario for invalid values, we are expecting the response status code to be 400 Bad Request and the response time to be less than 1000ms as well.

The majority of tests that failed are the ones in the test scenario for large values. Namely, the requirement for the response HTTP to contain "[" failed on all of methods in the API calls in the last testing scenario. Additionally, the requirement for the time response to be less than 1000ms failed on the methods: 2D SHO, 3D Morse, 3D SHO in the large values scenario as well. The same test for the method 3D SHO with valid values also had execution time larger than 1000ms.

Not all tests could be run in all of the tools. Namely, Katalon Studio, SoapUI and HttpMaster do not offer ready–made assertions for validating the response time. HttpMaster Express also does not have an option for validating whether the body contains a given string, so we could not validate it if the response body contains an opening bracket. None of the tests could be automatically executed in Insomnia and Swagger Inspector because they are tools intended for manual testing. All testing tools where the tests were executed gave the same responses for all test scenarios.

4. Conclusion

The necessity for system interoperabilty and reusing imposes the need for creation of web services. RESTful web services came into play in the early 2000s. Testing of such web services

Table 3
Values used for each test scenario.

Method	Test Scenario			
	Default values	Valid values	Invalid values	Large values

1D Morse
npts – number of points
D – dissociation depth
a – inverse "width" of the potential
x0 – equilibrium bond distance
prec – precision
	npts=10
	D=3
	a=0.5
	x0=0.0
	prec=6
		npts=20
		D=5
		a=0.4
		x0=0.5
		prec=15
			npts=p
			D=128
			a=0.5
			x0=4.2
			prec=5
				npts=500
				D=100
				a=15
				x0=20.5
				prec=30

1D SHO
npts – number of points
k – wavenumber of the SHO potential
x0 – displacement from origin
prec – precision
	npts=5
	k=1.0
	x0=0.0
	prec=8
		npts=20
		k=3
		x0=0.5
		prec=34
			npts=34
			k=12
			x0=12
			prec=s
				npts=500
				k=90
				x0=2000
				prec=70

1D Sombrero
npts – number of points
a – coefficient of the x^2 term
b – coefficient of the x^4 term
prec – precision
	npts=5
	a=−5
	b=1.0
	prec=8
		npts=55
		a=−15
		b=20
		prec=15
			npts=30
			a=.
			b=12
			prec=20
				npts=500
				a=−100
				b=10
				prec=90

1D Woods–Saxons
npts – number of points
V0 – potential depth
z – surface thickness
r0 – rms nuclear radius
A – mass number
prec – precision
	npts=5
	V0=50.0
	z=0.5
	r0=1.2
	A=16
	Prec=8
		npts=60
		V0=120
		z=1
		r0=0.9
		A=15
		prec=4
			npts=55
			V0=0
			z=!
			r0=1.9
			A=20
			prec=5
				npts=500
				V0=2000
				z=1000
				r0=120
				A=70
				prec=40

2D Morse
(same as 1D Morse but with two dimensions)
	npts=10
	D1=3 D2=3
	a1=0.5 a2=0.5
	x0=0 y0=0
	prec=6
		npts=7
		D1=17 D2=50
		a1=0.5 a2=0.1
		x0=15 y0=2.0
		prec=7
			npts=25
			D1=t D2=1
			a1=0 a2=m
			x0=23 y0=3
			prec=30
				npts=500
				D1=120 D2=75
				a1=400 a2=−50.2
				x0=70 y0=75
				prec=30

2D SHO
(same as 1D SHO but with two dimensions)
	npts=5
	k=1.0
	x0=0 y0=0
	prec=8
		npts=18
		k=1.5
		x0=0.2 y0=0.9
		prec=3
			npts=f
			k=10
			x0=23 y0=23
			prec=11
				npts=250
				k=230
				x0=10 y0=14
				prec=10

3D Morse
(same as 1D Morse but with three dimensions)
	npts=10
	D1=3 D2=3
	a1=0.5 a2=0.5 a3=0.5
	x0=0 y0=0 z0=0
	prec=6
		npts=3
		D1=6 D2=3 D3=6
		a1=0.1 a2=1 a3=12
		x0=−4.5 y0=9 z0=3.7
		prec=3
			npts=15
			D1=6.8 D2=3.1 D3=**
			a1=5.7 a2=5 a3=12
			x0=1.3 y0=32 z0=5
			prec=9
				npts=100
				D1=50 D2=−12.5 D3=10
				a1=12.5 a2=13 a3=12
				x0=100 y0=10 z0=−12.5
				prec=15

3D SHO
(same as 1D SHO but with three dimensions)
	npts=5
	k=1.0
	x0=0 y0=0 z0=0
	prec=8
		npts=14
		k=5
		x0=3.5 y0=0.2 z0=2.7
		prec=7
			npts=12
			k=k
			x0=x y0=y z0=z
			prec=5
				npts=100
				k=100
				x0=402 y0=12 z0=43
				prec=20

differed from testing of SOAP services because REST approach did not follow any unified description standards, but it required inputs criteria related to the requests and output criteria related to the API responses. To explore the current testing options for RESTful APIs, we have analyzed the existing testing tools and identified their main characteristics and differences in terms of HTTP request support and functionalities such as implementation, platform, languages, testing techniques and supported imports. As a use case we considered the REST API for solving multidimensional time–independent Schrödinger equation. We defined different testing scenarios and success criteria. Based on the results, the majority of the tests that failed were

Table 4
Responses from tests execution in Postman.

Method	Test Scenario			
	Default values	Valid values	Invalid values	Large values
1D Morse	status code: 200 body: [−2.41671645 −1.39124794 −0.28535681 1.09633735 7.42311473] response time: 449ms	status code: 200 body: [−4.387544466935593e+00 −3.282632704373475e+00 −2.337556625564300e+00 −1.543675048723042e+00 −8.136905006475681e−01 ...] response time: 506ms	status code: 400 body: Error Page Status code: 400 Exception Message: Invalid parameters response time: 137ms	status code: 200 body: response time: 526ms
1D SHO	status code: 200 body: [0.5 1.5 2.5 3.5 4.5] response time: 465ms	status code: 200 body: [0.8660254037859515 2.5980762109657265 4.330127034139766 6.062177340943015 7.794235639993951...] response time: 442ms	status code: 400 body: Error Page Status code: 400 Exception Message: Invalid parameters response time: 157ms	status code: 200 body: response time: 403ms
1D Sombrero	status code: 200 body: [−3.31729755 −3.29056942 −1.70943058 −1.67240256 2.48970011] response time: 396ms	status code: 200 body: [−4.105372703392596e−01 1.692638946849319e+00 6.794837989064262e+00 1.250634592748826e+01 1.908246809900831e+01...] response time: 448ms	status code: 400 body: Error Page Status code: 400 Exception Message: Invalid parameters response time: 94ms	status code: 200 body: response time: 420ms
1D Woods −Saxons	status code: 200 body: [−49.73342002 −49.02383 −47.92816698 −46.25839997 −42.70563227] response time: 446ms	status code: 200 body: [−119.9007 −119.6602 −119.3168 −118.8833 −118.3667...] response time: 486ms	status code: 400 body: Error Page Status code: 400 Exception Message: Invalid parameters response time: 116ms	status code: 200 body: response time: 452ms
2D Morse	status code: 200 body: [−4.83343289 −3.80796439 −3.80796439 −2.78249589 −2.70207325] response time: 466ms	status code: 200 body: [3902255.4066199 3902256.6004603 3902258.1494372 3902260.1866722 3902262.9744909...] response time: 440ms	status code: 400 body: Error Page Status code: 400 Exception Message: Invalid parameters response time: 99ms	status code: 200 body: response time: 451ms
2D SHO	status code: 200 body: [1. 2. 2. 3. 3.] response time: 402ms	status code: 200 body: [1.225 2.449 2.449 3.674 3.674 ...] response time: 505ms	status code: 400 body: Error Page Status code: 400 Exception Message: Invalid parameters response time: 85ms	status code: 200 body: response time: 1472ms
3D Morse	status code: 200 body: [−7.25014934 −6.22468084 −6.22468084 −6.22468084 −5.19921233]	status code: 200 body: [3.784e+26 3.784e+26 3.784e+26] response time: 444ms	status code: 400 body: Error Page Status code: 400 Exception Message: Invalid parameters response time: 87ms	status code: 200 body: response time: 18.30s
3D SHO	status code: 200 body: [1.5 2.5 2.5 2.5 3.5] response time: 474ms	status code: 200 body: [3.3597915 5.5193015 5.5947324 5.5960373 7.5875726 ...] response time: 6.70s	status code: 400 body: Error Page Status code: 400 Exception Message: Invalid parameters response time: 85ms	status code: 200 body: response time: 17.68s

defined as a part of the destructive testing scenario. Also, some tools did not provide ready–made assertions for validating the execution time. This research contributes to the theory by analyzing the options for testing RESTful APIs. It also helped us to think about the possible code optimization in order to satisfy the desired execution time requirements.

Acknowledgments

This work was supported in part by the European Union's Horizon 2020 research and innovation programme, project National Initiatives for Open Science – Europe, NI4OS–Europe, [857645] and by the Faculty of Computer Science and Engineering, Skopje, North Macedonia.

References

[1] B. Lim, H. J. Wen, Web services: An analysis of the technology, its benefits, and implementation difficulties, Information systems management 20 (2003) 49–57.
[2] K. Gottschalk, S. Graham, H. Kreger, J. Snell, Introduction to web services architecture, IBM systems Journal 41 (2002) 170–177.
[3] A. Neumann, N. Laranjeiro, J. Bernardino, An analysis of public rest web service apis, IEEE Transactions on Services Computing (2018).
[4] F. Halili, E. Ramadani, Web services: a comparison of soap and rest services, Modern Applied Science 12 (2018) 175.
[5] H. Ed-Douibi, J. L. C. Izquierdo, J. Cabot, Automatic generation of test cases for rest apis: a specification-based approach, in: 2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC), IEEE, 2018, pp. 181–190.
[6] A. Martin-Lopez, S. Segura, 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.
[7] A. J. Richardson, Automating and Testing a REST API: A Case Study in API testing using: Java, REST Assured, Postman, Tracks, cURL and HTTP Proxies, 2017.
[8] S. Venkatraj, R. Vincent, V. Vijayakumar, K. Vengatesan, M. Rajesh, Development of test automation framework for rest api testing, Journal of Computational and Theoretical Nanoscience 16 (2019) 453–457.
[9] V. Atlidakis, P. Godefroid, M. Polishchuk, Rest-ler: automatic intelligent rest api fuzzing, arXiv preprint arXiv:1806.09739 (2018).
[10] A. Arcuri, Restful api automated test case generation, in: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2017, pp. 9–20.
[11] A. Arcuri, Restful api automated test case generation with evomaster, ACM Transactions on Software Engineering and Methodology (TOSEM) 28 (2019) 1–37.
[12] SmartBear Software, The world's most popular api testing tool | soapui, https://www.soapui.org/, 2006. (Accessed on 05/22/2020).
[13] Katalon LLC, Katalon studio | the #1 codeless automation tool, https://www.katalon.com/katalon-studio/, 2015. (Accessed on 05/22/2020).

[14] Postman HQ, Postman | the collaboration platform for api development, https://www.postman.com/, 2012. (Accessed on 05/22/2020).
[15] Apache Software Foundation, Apache jmeter - apache jmeter™, https://jmeter.apache.org/, 1998. (Accessed on 05/22/2020).
[16] J. Haleby, Rest assured, http://rest-assured.io/, 2010. (Accessed on 05/22/2020).
[17] Assertible, The easiest way to test and monitor your web services : Assertible, https://assertible.com/, 2015. (Accessed on 05/22/2020).
[18] P. Thomas, Karate, https://intuit.github.io/karate/, 2017. (Accessed on 05/22/2020).
[19] SmartBear Software,, Openapi design & documentation tools | swagger, https://swagger.io/tools/swagger-inspector/, 2010. (Accessed on 05/22/2020).
[20] Kong Inc, Insomnia | api design platform and rest client, https://insomnia.rest/, 2019. (Accessed on 05/22/2020).
[21] Borvid, Httpmaster | master http testing and debugging, https://www.httpmaster.net/, 2013. (Accessed on 05/22/2020).
[22] Koteska et al, SchrödingerAPI - Jupyter Notebook, https://schrodinger.chem-api.finki.ukim.mk/, 2021. (Accessed on 06/02/2022).
[23] A. Richie-Halford, richford/dvr_py: Python module that solves one-dimensional potentials using a discrete variable representation method., https://github.com/richford/dvr_py/, 2017. (Accessed on 05/25/2020).