

COMPREHENSIVE TESTING FRAMEWORK FOR MICROSERVICES

Vajiramal Vilochane Vidyaratne

(179355D)

M.Sc. in Computer Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

March 2021

COMPREHENSIVE TESTING FRAMEWORK FOR MICROSERVICES

Vajiramal Viloachane Vidyaratne

(179355D)

This dissertation submitted in partial fulfilment of the requirements for
the Degree of MSc in Computer Science specializing in Software
Architecture

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

March 2021

DECLARATION

I declare that this is my own work and this MSc Thesis Project Report does not incorporate without acknowledgment any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgment is made in the text.

Also, I hereby grant to the University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works.

B.K.V. V.Vidyarathne

Date

I certify that the declaration above by the candidate is true to the best of my knowledge and that this project report is acceptable for evaluation for the MSc Thesis (CS6997).

Dr. Indika Perera

Date

ABSTRACT

An emerging trend has begun with the introduction of microservices architecture, transforming monolithic applications into microservices. These services are focused on smaller independent autonomous services, which encourage development, deployment, scale, and maintain each service independently with improved parallel development among multiple autonomous development teams. Unlike the monolith microservices, architecture introduces additional testing challenges.

Microservices integration testing, a crucial process that ensures the communication paths are correct while interacting with other services. Because of this unique architecture, testing integration in isolation impose a challenge due to many reasons, such as each microservice is developed and maintained by individual autonomous teams, unavailability or instability of service due to parallel continuous rapid development, deployment and difficulty in setting up all the services and seed the necessary data for testing in a local or a remote development environment. This process requires effort to initialize and maintain the resource-intensive environment. Currently, available testing approaches or tools have particular limitations. Such approaches or tools require extra effort to create, initialize and maintain with the rapidly changing unstable services and requirements. Gradually with the introduction of new services as the number of services is greater this process turns in to a development overhead.

The main objective of this research is to overcome the integration testing challenges in REST-based microservices, via a service virtualization solution based on widely adapted Open API specification. The proposed implemented solution was evaluated against the existing services. The solution is proven to mitigate the identified integration testing challenges in microservices and successfully emulates the tested producer services via the virtualization of its Open API specification. The solution facilitates reproducing the edge cases and failures, such is difficult to produce in the real services, and this helps to reduce the development time significantly. The solution can evolve with the services due to the OAS of the respective service reflects the producer services' additions or its changes. Because of this reason, unlike the existing approaches or tools, this solution does not require any maintenance via mocking, simulation by the record and replay requests or as a third-party dependency package or a library to the producer and consumer source code to reflect the service changes or additions. This report concludes by mentioning research contributions, limitations, and future works of the implemented solution.

Keywords: monolithic, microservice, REST, integration, testing, development, architecture, OAS.

ACKNOWLEDGEMENTS

I would first like to express my sincere gratitude to my research supervisor Dr. Indika Perea for the guidance, knowledge, suggestions, and encouragement throughout the research which steered for the successful completion of this research.

Secondly, I would like to thank my family members, friends, and peers for the support and encouragement for the research work.

Finally, I would like to thank my current workplace CMS (PVT) Ltd, and my colleagues for their support and encouragement.

TABLE OF CONTENTS

CHAPTER 1 - INTRODUCTION	1
1.1 BACKGROUND.....	1
1.2 MONOLITHIC ARCHITECTURE.....	2
1.3 MICROSERVICES ARCHITECTURE.....	3
1.4 COLLECTION OF SERVICES WORK TOGETHER AS A SYSTEM	7
1.5 PROBLEM STATEMENT	8
1.6 MOTIVATION.....	9
1.7 OBJECTIVES	10
CHAPTER 2 - LITERATURE REVIEW.....	11
2.1 MICROSERVICES TESTING	11
2.1.1 <i>Unit tests</i>	12
2.1.2 <i>Sociable unit tests</i>	12
2.1.3 <i>Solitary unit tests</i>	12
2.1.4 <i>Integration tests</i>	12
2.1.5 <i>Gateway integration tests</i>	13
2.1.6 <i>Persistence integration tests</i>	13
2.1.7 <i>Component tests</i>	13
2.1.8 <i>Contract tests</i>	13
2.1.9 <i>End to end tests</i>	14
2.1.10 <i>Test Pyramid</i>	14
2.2 MICROSERVICES TESTING CHALLENGES	15
2.2.1 <i>Shared testing environment</i>	15
2.2.2 <i>Test kits</i>	16
2.2.3 <i>Test doubles</i>	16
2.2.4 <i>Service virtualization</i>	16
2.3 CURRENTLY AVAILABLE TOOLS	17
2.3.1 <i>PACT</i>	17
2.3.2 <i>Wire mock</i>	17
2.3.3 <i>Hooverfly</i>	17
CHAPTER 3 - METHODOLOGY	18
3.1 PROPOSED TESTING FRAMEWORK.....	18
3.2 OPEN API SPECIFICATION STRUCTURE.....	19
3.2.1 <i>Open API specification as code annotations</i>	20
3.2.2 <i>Paths</i>	21
3.2.3 <i>Parameters</i>	22
3.2.4 <i>Request body</i>	23
3.2.5 <i>Response</i>	24

3.2.6	<i>Testing isolated microservices using the framework</i>	25
3.2.7	<i>REST API</i>	27
3.2.8	<i>Request handler</i>	27
3.2.9	<i>Response generator</i>	28
3.2.10	<i>Report generator</i>	28
CHAPTER 4 - DESIGN AND IMPLEMENTATION		29
4.1	THE SCOPE OF THE IMPLEMENTATION	29
4.2	TEST FRAMEWORK IMPLEMENTATION	31
4.3	PRODUCER REGISTRATION AND CAPTURE OPEN API SPECIFICATION	36
4.4	PRODUCER SERVICE VIRTUALIZER	37
4.4.1	<i>Request parameter validation</i>	38
4.4.2	<i>Request body validation</i>	41
4.5	RESPONSE GENERATOR	48
4.6	VIEW PRODUCER OPEN API SPECIFICATION	52
4.7	REPORTING	54
CHAPTER 5 - EVALUATION		55
5.1	CTFM VS. EXISTING TOOLS OR TECHNOLOGIES	55
5.2	MINIMUM SERVER REQUIREMENTS AND CONFIGURATIONS	56
5.3	CONFIGURATIONS FOR WRITTEN INTEGRATION TESTS	58
5.4	REAL SERVICES VS. CTFM VIRTUALIZED SERVICES	62
5.4.1	<i>Test results interpretation</i>	71
5.5	PERFORMANCE TEST	81
CHAPTER 6 - CONCLUSION		84
6.1	RESEARCH CONTRIBUTION	85
6.2	RESEARCH LIMITATIONS	86
6.3	FUTURE WORK	87

LIST OF FIGURES

Figure 1.1: Sample monolithic application layout	2
Figure 1.2: Sample microservice internal structure	3
Figure 1.3: Microservice communication between external services and external data stores	5
Figure 1.4: System is formed by collection of microservices	7
Figure 2.1: Test Pyramid.....	14
Figure 3.1The high-level structure of the proposed testing framework.....	18
Figure 3.2: Open API specification main structure.....	19
Figure 3.3: Open API specification as code annotations	20
Figure 3.4: Paths definitions of an Open API Specification	21
Figure 3.5: Parameters of a request.....	22
Figure 3.6: Request body schema	23
Figure 3.7: Sample response segment	24
Figure 3.8: Testing microservices in isolation	25
Figure 3.9: Request activity flow.....	26
Figure 4.1: CTFM abstract architecture	32
Figure 4.2: composer.json	33
Figure 4.3: package.json.....	34
Figure 4.4: Framework folder structure	35
Figure 4.5: Adding a new producer configuration.....	36
Figure 4.6: Producer virtualize class.....	37
Figure 4.7: Open API parameter schema object.....	38
Figure 4.8: Schema analyzer request parameter validator	39
Figure 4.9: Request parameter validation error response	40
Figure 4.10: Request body as a reference	41
Figure 4.11: Request body definition as a object	42
Figure 4.12: Request complex payload structure definition.....	44
Figure 4.13: Formation of validation rules for a given specification	45
Figure 4.14: Request validation rule overridden example	47
Figure 4.15: Open API response specification.....	48
Figure 4.16: CTFM generated response according to response specification	49
Figure 4.17: Expected error response only instructed with the status code.....	50

<i>Figure 4.18 Expected response with the status and overridden response</i>	<i>51</i>
<i>Figure 4.19: Configuration view to access the specification of a producer</i>	<i>52</i>
<i>Figure 4.20: Open API specification view of a producer</i>	<i>53</i>
<i>Figure 4.21 User interface of the reporting module</i>	<i>54</i>
<i>Figure 5.1 Hooverfly producer test with simulation capture mode</i>	<i>58</i>
<i>Figure 5.2 Hooverfly consumer test configured for a specific simulation.....</i>	<i>59</i>
<i>Figure 5.3 Consumer test written to test with CTFM</i>	<i>60</i>
<i>Figure 5.4 Groovy response assertion for /card/\${token}/transaction.....</i>	<i>70</i>
<i>Figure 5.5: Producer services test response times</i>	<i>71</i>
<i>Figure 5.6: Producer services test results complete summary</i>	<i>71</i>
<i>Figure 5.7: CTFM virtualized services test response times.....</i>	<i>72</i>
<i>Figure 5.8: CTFM virtualized producer services test results complete summary</i>	<i>72</i>
<i>Figure 5.9: CTFM virtualized test results summary graphical presentation</i>	<i>73</i>
<i>Figure 5.10: Response assertion failure due to unavailable response definition</i>	<i>74</i>
<i>Figure 5.11: Response assertion failure due to unavailable response definition</i>	<i>75</i>
<i>Figure 5.12: Response assertion failure due to missing response definition.....</i>	<i>76</i>
<i>Figure 5.13: Response assertion failure due wrong response definition</i>	<i>77</i>
<i>Figure 5.14: Tests failure composition</i>	<i>78</i>
<i>Figure 5.15: CTFM results with corrected producer specification</i>	<i>79</i>
<i>Figure 5.16: Average response time for a single user</i>	<i>80</i>
<i>Figure 5.17: Average response time for a hundred users</i>	<i>81</i>
<i>Figure 5.18: Average response time for a ten users</i>	<i>81</i>
<i>Figure 5.19: Performance results summary graph</i>	<i>82</i>
<i>Figure 6.1 composer.json</i>	<i>2</i>
<i>Figure 6.2 Package.json.....</i>	<i>4</i>
<i>Figure 6.3 Dockerfile</i>	<i>5</i>
<i>Figure 6.4 Docker compose file</i>	<i>6</i>
<i>Figure 6.5 Make file</i>	<i>7</i>

LIST OF TABLES

<i>Table 4.1 Header to override validations.....</i>	<i>46</i>
<i>Table 4.2: Expected response request headers with only status code.....</i>	<i>50</i>
<i>Table 4.3: Expected response request headers with json schema.....</i>	<i>51</i>
<i>Table 5.1 CTFM vs existing tools or technologies</i>	<i>55</i>
<i>Table 5.2 Testing framework minimum server requirements</i>	<i>57</i>
<i>Table 5.3: Selected services for testing</i>	<i>70</i>
<i>Table 5.4: CTFM Virtualized test results summary</i>	<i>74</i>
<i>Table 5.5: Testing resources and environment.....</i>	<i>81</i>
<i>Table 5.6: Performance results summary.....</i>	<i>83</i>

APPENDICES

APPENDIX I: FRAMEWORK PACKAGES AND LIBRARIES	1
APPENDIX II: CTFM DOCKER CONFIGURATION	5

LIST OF ABBREVIATIONS AND SYMBOLS

OAS	Open API specification
API	Application programming interface
REST	Representational state transfer
HTTP	Hypertext Transfer Protocol
Groovy	Java syntax compatible object oriented programming language
PHP	Hypertext Preprocessor
CPU	Central Processing Unit
RAM	Random access memory
HDD	Hard disk drive
XML	Extensible Markup Language
JSON	JavaScript Object Notation
NPM	Node package manager
GraphQL	Open source data query and manipulation language
CTFM	Comprehensive testing framework for microservices