International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 5 | ISSN : 2456-3307

Analysis of REST API Implementation

Chaitanya Mukund Kulkarni, Prof. M. S. Takalikar
Department of Computer Engineering, Pune Institute of Technology, Pune, Maharashtra, India

ABSTRACT

RESTful web services provide an architectural style for developing the web services and way of consuming those apis for client. The apis, developed using http protocol may not be following all the REST constraints. The motivation of this paper is to design the method for api validation. Method checks if the implementation is developed as per the requirements of the specification document of the respective api. This paper also studies the challenges in analysis of the REST api and validation of the api implementation. This mechanism will consider OpenApi Specification document of the RESTful web api implementation.

Keywords: Restful Web Services, Web Interfaces, Web Services, OpenApi specification, swagger documentation

I. INTRODUCTION

The technologies developed for the web based services starting from the basic RPC mechanisms, SOAP based webservices and the architectural style Representational State Transfer (REST) [1]. REST is the term defined by Roy Fielding. REST based micro service oriented architecture is gaining more and more popularity for the realization of web based service design. RESTful web apis works on HTTP protocol with the resource having URIs and MIME types. As REST is developed on already established and efficient internet web technology it provides simplicity along with standard inter operable and availability on all the platforms [1]. REST architecture provides some constraints which stresses on the quality attributes on REST full web services. REST based software systems assure modular, scalable and extendable web service development [2]. As REST services are based on REST architecture, the framework helps to realize REST compliant design to a running web service.

The Fielding's [1] research describes four main constraints for any web service to be REST compliant. Resources, their representations, response messages given by the apis and HATEOAS links provided by the apis for connectedness are the four principles defined for the REST architecture. Most of the constraints are fulfilled using the appropriate, language specific framework.

To provide a way for analysing the implementation of REST services, this paper provides method for analysis with the help of OpenApi Specification document 2.0. Our main goal will be of providing a way to compare the implementation and its documentation and giving the detailed report on the REST api implementation. The report will also have the details about gaps between the implementation and the documentation. We envision that knowing and analysing these data in detail will allow deriving fitting approaches for resolving the identified deficits, helping to improve the state of the art with purposeful solutions. There will be two parts, one for comparison between implementation and the documentations for identifying gap in the

implementation and second one will be for applying the REST api validation criteria.

The rest of the paper is organized as: part II will give the review of the related work done, part III will give the implementation detail of the system, part IV will describe the result analysis information and in finale section the work will be concluded

- **Definitions**

A. OpenApi Specification/Swagger: OpenApi Specification is the standard for the REST api documentation. The document is developed in json or yaml format. The specification document has information about each path. Every path or the endpoint has the details about methods it implements. Http methods gives details of parameters, responses, accept and return types. The solution proposed by our system is based on this specification documentation format. The version used by the system in OpenApi specification 2.0.

B. Resource: REST architecture is a resource oriented architecture. Resources are the entities of the system around which the apis are implemented. RESTful services return resources in some representation format. REST architecture is stateless architecture as client and server does not store.

C. HTTP protocols: REST web apis are developed using HTTP protocol. Hence it must make use of all the HTTP properties such as all the HTTP verbs, HTTP responses, header information etc. The work presented here will consider that the apis are using HTTP protocols and the apis are validated accordingly.

II. REVIEW OF LITERATURE

Web services have been studied and investigated to improve the performance in many ways. One of the way of categorizing the web services on basis of types of the services such as RPC based services, resource oriented like REST, service oriented as SOAP services and hybrid services developed [3].

Paul Adamczyk et. Al [4] focused the study on principles of the REST architecture defined by Fielding's literature [1]. Using these principles, the REST standards are defined and comparison between REST and traditional web services is carried out. The work analyses the services theoretically as well as practically. In theoretical study, the focus is on the principles of the REST architecture which are responsible for efficient design of the web api. Each of the principles (or constraints defined by Fielding's work [1]) is elaborately defined. REST architecture is resource oriented. Hence, resource is core of the REST services, which makes resource first principle of the REST architecture. Each service is related to a particular resource. A resource is an entity or a group of entities which are represented in the forms defined by the servers by considering the client's needs. Thus, the representation of a resource is the second principle of REST. Whenever a resource is requested by client server provides it in a representation such as application/json. For each request to server, server sends some message as a response. The responses given by a server gives the success and failure details of the api endpoint. These responses are third principle of the REST architecture. Forth principle ensures connectedness among the api endpoints. This constraint stresses each endpoint to provide hypermedia links to other resource apis. The work presented by Dominik Renzel et. Al [6] try to define some standards for the RESTful web services. The standards are classified in 17 analysis criteria. Among these criteria 12 are defined as the properties which the apis should follow while remaining 5 standards are defined as the

properties which are not always required for a web service to be RESTful. The standards are defined in such a way that the service must follow all the characteristics provided by Fielding's research [1] and the book RESTful web services [2]. The analysis of this paper is carried out over 25 most popular Web services from www.programmabelweb.com. The analysis starts from availability of formal description of the web service along with HATEOAS link checking, resource analysis, various header parameter checks, HTTP usage checking, and ends with the analysis regarding security requirements for the web api.

There is another approach for analysing the REST apis which is the structural analysis of the api [13] [14] [15]. The work by F. Haupt et. Al [13] targets the api description for checking and analysing the api implementation. It defines and uses canonical model which gives benefits of defining resources, apis, links, root, and number of operations performed on the resources. The REST apis provided by some cloud providers are studied by F. Petrillo et. Al [8] with help of 73 best practices compiled from literature. It checks whether the apis are following REST constraints or not.

By studying the work on REST api architecture there is need for the work in standardization of the web apis developed on REST architecture. This paper will consider the implementation and documentation of the api and try to check if the api is following the REST constraints or not.

III. SYSTEM OVERVIEW

The proposed system will be designed in such a way that the analysis of the REST APIs can be carried out by considering the implementation as well as the documentation of REST web api. Documentation of RESTful API is written in swagger 2.0 format. The springfox framework is used for generation of the documentation from the implementation. Following section will describe the architecture of the system proposed.

- **Design**

There will be two inputs mainly require for our system. First is the manual written document of the system and second is the document generated by springfox framework.

There will be 2 functional modules present in the system. First module will generate a document from the code. Second module will compare the generated document with the hand written one and check for the gaps present.

The rules are defined for the comparison of Swagger document or OpenApi document. This comparison will be used for generating the analysis report of the api. The rules can be manipulated based on priority of the developer.

- **System Architecture**

Figure 1 gives high level overview of different parts of the system. Architecture consists of various modules based on the functionality related to the system. Analyzer and parser are the main components of this system architecture. Parser will work on two specification files used in this system.

One of the Module is developed by using the Springfox framework which is an open source project. This project is used to create the documentation automatically from the implementation of the api in the spring framework. This is the initial module of the project.

The system has 2 swagger documents as inputs. One of the open api specification document is the documentation which was written before actual implementation of the service. This document gives the design details along with all the endpoint information and their responses and parameters. The implementation should take place by following this

document. The second document i.e. swagger document is created from the code using springfox. This document is compared with the initial one by following analysis rules defined to check if the implementation covers all the necessary details of the resource in the api or not.

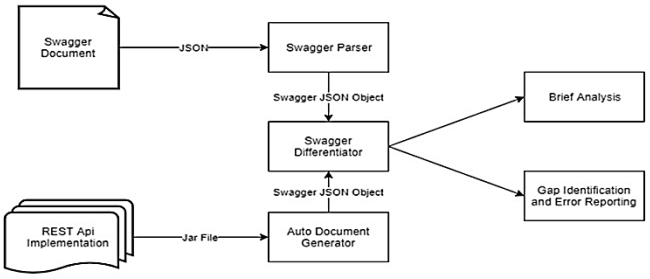

Figure 1. System Architecture

- **Software requirements & specification**

The system which is developed for checking the consistency between implementations of a restful service and it's open api documentation dependents on some of the open source technologies. The JDK on which the project is checked is on JDK 1.8. The dependencies of the software are as follows:

- OpenApi Specification 2.0,
- Spring Boot Framework,
- SpringFox Framework,

As the service is being developed in java language spring boot framework is most popular framework for REST api development.

IV. SYSTEM ANALYSIS

The REST api analysis is carried out by comparing the inputs to the system. The analysis provides details about the differences in the implementation and the expected documentation. The auto generated document represents the implementation of the api.

The reports generated by the system will consider the details about correctly implemented apis, api implementation with error, not implemented apis and implemented but not documented apis. There will be mainly 2 types of analysis the system will generate. First one will give overview of the implemented or not implemented apis. And second analysis report will be regarding every single end point. In this second analysis the end point with a particular verb will be checked for it's uri representation, parameters, responses along with consumes and accept mime types.

Another major contribution of this paper is the finding of the gap in documentation and implementation. If some part of documented api is not implemented the such gap is also reported.

- **Result Analysis**

This section will discuss the two types of analysis results in details along with the gap identification details generated by this technique.

The first analysis result can be viewed as overview of the system analysis while second analysis result is detailed analysis of each end point. Following result values will be generated as first analysis.

Sr. No.	Parameters to Check
1	Number of Resources
2	Number of Endpoints Expected
3	Number of Endpoints Implemented
4	Not Expected but Implemented Endpoints
5	Number of Gaps in System
6	Percentage of System completed

This table gives overview describing that, if the service is implemented according to the specification document requirements or not. Such analysis is presented by values in the result table mentioned above.

The second analysis will consider each of the end point documented and check whether the implemented end point is similar to what documentation design expected. In this analysis each http verb and its details like header info, of a particular end point will be checked with the document. The result of this analysis will contain following properties.

Sr. No.	Details of Api Endpoint
1	Expected parameters and implemented parameters
2	Expected consume mime type and implemented consume mime types
3	Expected produces mime types and implemented produces mime types
4	Expected http responses and implemented http responses

From this result table, the difference in implementation and documentation will be shown so that the apis can be validated.

By using result of first and second analysis the gap between expected system implementation and actual implementation can be shown.

V. CONCLUSION

The system will be helpful in lot of ways to analyse the REST api implementation. Proposed system can be used for complex web service. Once the implementation is completed, the implementation and the documentation provided prior to actual implementation both are checked by this system and result is generated accordingly. If implementation lacks some functionality or property related to any resource it will be prompted to developer. The approach described can also be used to check whether the implementation and document follows REST standards and if it is REST compliant or not.

VI. FUTURE WORK

Current system is being created by considering the version of the open api specification 2.0. However, the system can be upgraded for open api specification 3.0, as this version provides details about multiple servers and much more. Basic design will remain same. New upgrade will consider the details provided by version 3. Right now, the system expects the code of the implementation, however the work can be extended for running api services.

VII. ACKNOWLEDGMENT

This project was sponsored by SAS Research & Development, Pune. I take this opportunity to express my deep sense of gratitude towards my project mentor Rakesh Jadhav and Arvind Jagtap for their valuable guidance and suggestions for the project.

VIII. REFERENCES

[1] R. T. Fielding and R. N. Taylor, Principled design of the moder Web architecture, ACM Trans. Internet Technol. 2, May 2002: 115-150.

[2] Leonard Richardson and Mike Amundsen, RESTful Web APIs", O'Reilly Media, 2013.

[3] M. Maleshkova, C. Pedrinaci, and J. Domingue, Investigating web APIs on the World Wide Web, The 8th IEEE European Conference on Web Services (ECOWS 2010), 1-3 Dec 2010, Ayia Napa, Cyprus.

[4] P. Adamczyk, P.H. Smith, R.E. Johnson, and M. Hafiz, "REST and Web services: In theory and in practice", REST: from Research to Practice, Springer New York, 2011.

[5] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and

Content", RFC 7231, 2014, http://www.ietf.org/rfc/rfc7231.txt.

[6] D. Renzel, P. Schlebusch, and R. Klamma, "Today's top 'RESTful' services and why they are not RESTful", WISE, 2012.

[7] F. Petrillo, P. Merle, N. Moha, and Y.G. Guéhéneuc, "Are REST APIs for Cloud Computing Well-Designed? An Exploratory Study." ICSOC 2016, Springer International Publishing, 2016.

[8] Rodríguez, Carlos, et al. "REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices." International Conference on Web Engineering, Springer, 2016.

[9] M. Fowler, "Richardson maturity model: steps toward the glory of rest", http://martinfowler.com/articles/richardsonMaturityModel.html, 2010.

[10] Swagger, http://swagger.io/

[11] R.T. Fielding and R.N. Taylor, "Principled design of the modern Web architecture", ACM Trans. Internet Technol. 2, May 2002: 115-150

[12] F. Haupt, D. Karastoyanova, F. Leymann, and B. Schroth, "A modeldriven approach for REST compliant services", ICWS, 2014.

[13] F. Haupt, F. Leymann, and C. Pautasso. "A conversation based approach for modeling REST APIs." WICSA 2015 , IEEE, 2015.

[14] K. Vukojevic-Haupt, F. Haupt, F. Leymann, and L. Reinfurt, "Bootstrapping Complex Workflow Middleware Systems into the Cloud." e-Science 2015, IEEE, 2015.

[15] F. Palma, J. Dubois, N. Moha, and Y.G. Guéhéneuc, "Detection of REST patterns and antipatterns: a heuristics-based approach", ICSOC 2014, Springer Berlin Heidelberg, 2014.

[16] F. Palma, J. Gonzalez-Huerta, N. Moha, Y.G. Guéhéneuc, and G.Tremblay, "Are restful apis well-designed? detection of their linguistic (anti) patterns." International Conference on Service Oriented Computing. Springer Berlin Heidelberg, 2015.