End to End Representational State Transfer Application Programming Interface Development

Akilesh N S, Rohini S Hallikar

Electronics and Communication Department, R V College of Engineering, Bangalore, India

***Abstract*— An application programming interface or an API can be interpreted as a black box performing a function. This black box is accessible to anyone permitted to give it an input, or in software terms, a request. A well developed API should return a response, regardless of the request. There has been a trend in the tech industry to break down complex software operations into simpler single step operations, and hence moving from one convoluted end to end software program performing a single convoluted operation, towards creating multiple, independent, API's, which perform a very specific task, and call each other sequentially as each task completes performing. Breaking down this complex software operation into multiple single API's has various benefits. An obvious one, is the reusability of each of these API's, as a step in multiple different complex operations. This paper discusses the overview and development of REST API's and the overarching steps involved in the development of an API, before it is finally deployed.**

***Keywords*— API, REST**

I. INTRODUCTION

An Application Programming Interface is a piece of software performing a function. This API can connect to other API's or other services on completion of execution. The internal working of the API is generally unknown to the user, and the only information known to the user is the parameters expected for the request, and the parameters present in the API response. In this way, an API is a useful tool for companies that want to keep their internal software confidential, while providing a client with the service. Apart from this, an API is usually configured to perform only a single task. In this way, code can be successfully re used.

Figure 1 represents an example of the traditional software development approach. Steps such as Verifying the eligibility of the user, validation of parameters, and returning the response in a user understandable format are used as functions within the program. These steps, however, are common between both applications. It therefore makes sense to externalize these functions to API's. Similarly, the core function performed by Application 1 and 2 can also be externalized to API's.

Figure 2 represents API externalization of functions, and hence effective code use between different applications. Similar to how functions are used in local level applications to perform a certain functionality repeatedly and avoid code redundancy, API's are used to perform a certain common functionality, however, on an application level.

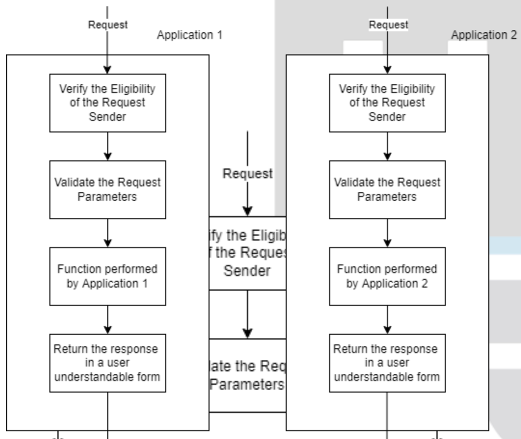

Request
Application 1
Verify the Eligibility of the Request Sender
Validate the Request Parameters
Function performed by Application 1
Return the response in a user understandable form
Response
Request
Application 2
Verify the Eligibility of the Request Sender
Validate the Request Parameters
Function performed by Application 2
Return the response in a user understandable form
Response

Figure 1: Traditional Software Development Approach

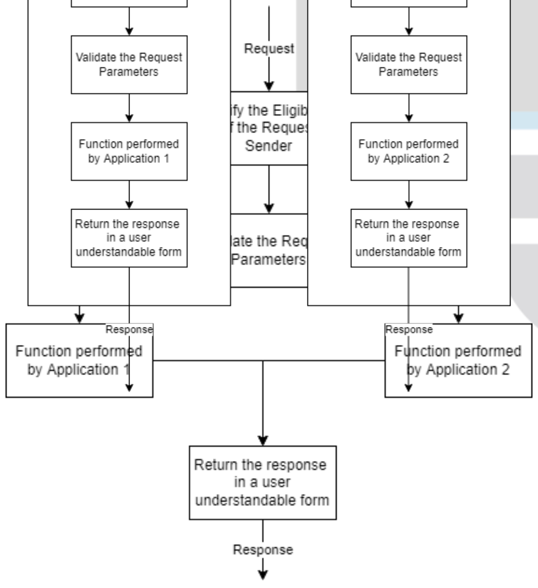

Request
Verify the Eligibility of the Request Sender
Validate the Request Parameters
Function performed by Application 1
Function performed by Application 2
Return the response in a user understandable form
Response

Figure 2: API Based Approach

Each of the API's perform a single function, and call each other sequentially, as they complete performing their functionality.

The development of any API follows a predefined architecture. Two of the most popular architectures are the REST– Representational State Transfer protocol and the SOAP – Simple Object Access Protocol protocol. The REST protocol has multiple benefits over the SOAP protocol – REST has lower bandwidth requirements, supports more file formats.
The rest of the paper is organized as follows. Section II discusses a step by step approach for developing, testing and deploying a REST API. Section III discusses results supporting the efficacy of the REST architecture over SOAP. In section IV, the conclusion and future work is discussed, followed by References in Section V.

II. PROPOSED MODEL

The development of any API, can in general, be broken down into 4 stages:

i. **Definition:** The definition stage involves creating high level architectural models of the API, clearly defining the other services or software interacting with it. It also involves defining the format of the request of the API, with all the expected parameters from the user, and the format they can expect to receive back.

ii. **Development:** Development involves the local development of the software. API's are usually developed in lightweight languages such as JavaScript. Once locally developed, the API is uploaded and tested on cloud services, with sample requests, to evaluate their working.

iii. **Testing:** After Development, the API is tested periodically for Quality Assurance. This is also known as QA testing. A predefined request is sent to the API, and the response returned by the API is compared with the response expected. For every mismatch in response, the report percentage of the code decreases. The API is also performance tested to determine the number of requests it can handle simultaneously.

iv. **Deployment:** An API that has been developed and tested is then deployed. Deployment usually happens on a pipeline that has additional checks such as code quality/code redundancy checks. Deployment allows the software developer to continuously integrate changes to the code. The client or customer can update their version of the code, through the single click of a button.

The steps further involved in each of the aforementioned stages are discussed henceforth.

A. *Definition*

The Definition of an API before development carries various purposes. It helps document the services and other software/API, the API will require for its operation, through the use of a high level architecture diagram. Along with this the required parameters for the request, along with a sample API request is documented. The response returned from the API and the parameters it returns are documented as well.

1. *High Level Definition*

The High Level Definition of an API defines the other software/services required to interface with an API.

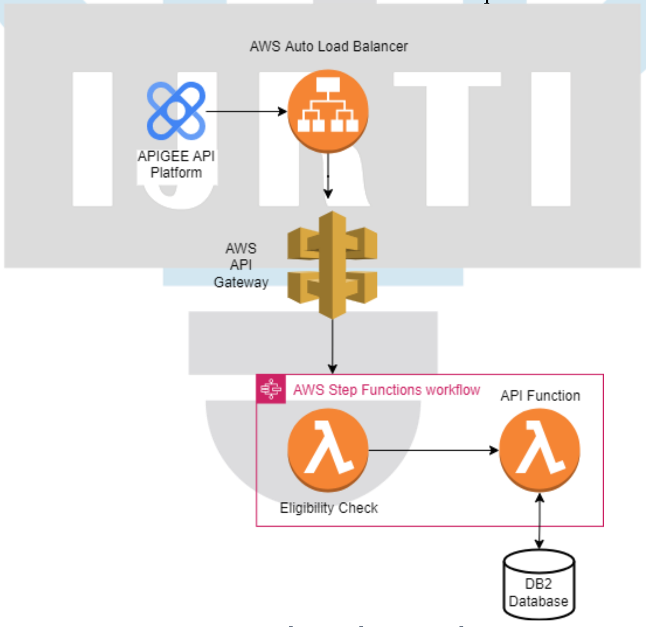


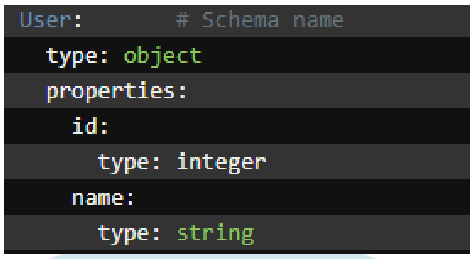
Figure 3: High Level API Architecture

Figure 3 represents a typical example of an API onboarded on to the Amazon Web Services Platform.

The Main API here, is the AWS Lambda tagged as API Function. However, as seen above, to interface with the API, it is necessary to go through multiple services. This level of documentation, is necessary to understand the other services/API that will have to be configured.

2. Request/Response Definition

The parameters required by a request, for an API to work successfully are to be documented. The response parameters returned by the API are documented as well. Services such as Postman, SwaggerHub or Redocly are used for this.
Along with just defining the parameters required, the schema and expected datatype of each parameter is defined as well.



```
User:         # Schema name
  type: object
  properties:
    id:
      type: integer
    name:
      type: string
```

Figure 4: Parameter Definition

An example of parameter definition for an API, is shown in figure 4. The parameter ‘User’ is an object, with the properties ‘id’ of type integer, and ‘name’ of type string.

B. *Development*

The development stage involves actually coding the API. The functionality and requirement of the API is understood and broken down into functions. API’s for web applications are usually developed in JavaScript or TypeScript.The JSON/XML Request from the client, is passed as an input parameter to the main function of the API. Necessary information from the request is parsed, and the functionality of the API is decided based on these input parameters.Once developed locally, the API is tested on the cloud service planned for deployment. Examples of such cloud services are Amazon Web Services, Google Cloud Platform or Microsoft Azure Cloud. API’s developed on the RESTful Architecture are usually deployed to asynchronous code execution environments. ‘Lambdas’ are such a service available on AWS. Lambdas are asynchronously trigerred when an input JSON request is given.It is crucial to test locally developed programs, on the cloud, as the API is ultimately going to reside on the cloud after deployment. Time out errors due to performance constraints are common errors when local code is deployed to the cloud. The runtime environment should be optimized to support code execution.

C. *Testing*

Testing of an API can be broken down into two components:

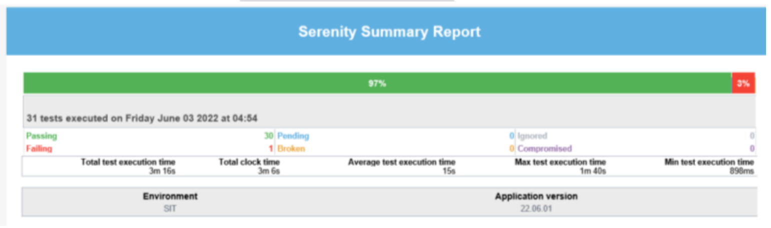
i. Quality Assurance(QA) Testing
ii. Performance Testing

1. Quality Assurance(QA) Testing

Quality Assurance or QA Testing are testcases designed to test all the possible scenarios an API can actually face when it is deployed to clients. An example of a popular QA Automation suite is the Java-Selenium-Cucumber testing suite.Edge cases or rare request types are also to be considered while creating test scenarios.Test Scenarios are commonly broken down into:

i. **Positive Scenarios**: These are scenarios designed with error free requests. That is, there is no issues with the request sent, and the scenario is designed to test if the API provides an accurate response. The response expected from the API, is directly compared with the response received. If the expected response matches the received response, the scenario succeeds.

ii. **Negative Scenarios:** Negative Scenarios test the error handling capabilities of an API. An intentionally erroneous request is sent to the API. The API should successfully identify that the request is erroneous, and accurately return a response, stating why the request is erroneous. This test scenario compares the message expected to be received from the API, with the actual message received. If there is a match, the scenario succeeds.



Serenity Summary Report

97% 3%

31 tests executed on Friday June 03 2022 at 04:54

Passing	30	Pending	0	Ignored	0
Failing	1	Broken	0	Compromised	0

Total test execution time	Total clock time	Average test execution time	Max test execution time	Min test execution time
3m 16s	3m 6s	15s	1m 40s	898ms

Environment	Application version
SIT	22.06.01

Figure 5: Report Generated from a QA test suite

Figure 5 represents a report generated from the Java-Selenium suite. It has both positive and negative scenarios as discussed previously, and passes 31 out of the 32 tests, rendering an Automation Percentage of 97 percent.
Only if all the positive and negative scenarios of a QA Automation test suite pass, a program can be deployed to clients.

2. Performance Testing

Performance testing, unlike QA testing, does not check the functionality of the program. The program is tested with varying workloads for the stability and speed of the software.

Performance bottlenecks are identified, and corrected.LoadNinja is a popular performance testing suite.

D. *Deployment*

A deployment pipeline follows the concept of CI/CD(Continuous Integration/Continuous Deployment). In this principle, new versions of the code are updated by the developer, to the pipeline, and the client can integrate or update the code by simply clicking a button. This pipeline introduces automation into the software delivery stage.

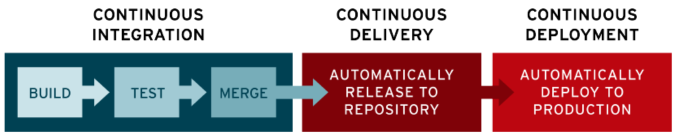


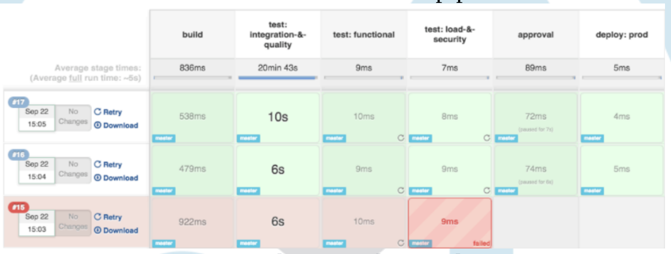
Figure 6: CI/CD Concept

Figure 6 represents the CI/CD approach. The updated version of the code is built, tested and merged. When merged to the repository, the pipeline can be trigerred to automatically update the release, when a change is made to the repository. This deployment is then automatically available to the client.

The pipeline usually has multiple stages, each stage performing a certain task. The developer can decide the amount of quality control they wish to perform on the pipeline.

The pipeline builds the code from scratch on a virtual machine, and runs unit test cases to measure code coverage and code quality, every time it's run. Only if all the stages in a pipeline pass, the newest version of the code is deployed.

Some of the most common tools used for CI/CD are the Jenkins or circleCI pipelines.



	build	test: integration-&-quality	test: functional	test: load-&-security	approval	deploy: prod
Average stage times: (Average full run time: ~5s)	836ms	20min 43s	9ms	7ms	89ms	5ms
#17 Sep 22 15:05 No Changes Retry Download	538ms	10s	10ms	8ms	72ms (paused for 7s)	4ms
#16 Sep 22 15:04 No Changes Retry Download	479ms	6s	9ms	9ms	74ms (paused for 6s)	5ms
#15 Sep 22 15:03 No Changes Retry Download	922ms	6s	10ms	9ms failed		

Figure 7: Jenkins Pipeline and stages

The pipeline represented in Figure 7, has 6 stages required to pass, for successful deployment.

The builds 16,17 have successfully deployed because they passed all stages, however, build 15 fails due to an error in the 'test:load-&-security' stage.

III. RESULTS AND DISCUSSIONS

The method proposed in the previous section provides an overview of how to design an API following the REST Architecture. The REST Architecture has been found to have a multitude of benefits over the SOAP Architecture.

Some of these advantages are:

1. REST has lower bandwidth requirements due to compact request/response
2. REST permits use of different data formats for transmission
3. REST can work on an underlying SOAP protocol, but vice versa is not possible.

A study between REST and SOAP protocol, contrasting the time taken for the upload of an image, in the form of a byte array is further discussed.

Test Setup: Byte Array's of sizes varying from 10kb to 40kb are uploaded onto an Image Upload Webservice, using, REST, as well as SOAP, and the time taken in milliseconds for the operations using both the protocols are compared.

Table 1: Image upload response times

Bytes Array(kb)	Time(Milliseconds)	
	SOAP	REST
10	8760	3950
20	12800	9560
30	19870	15689
40	28050	23900

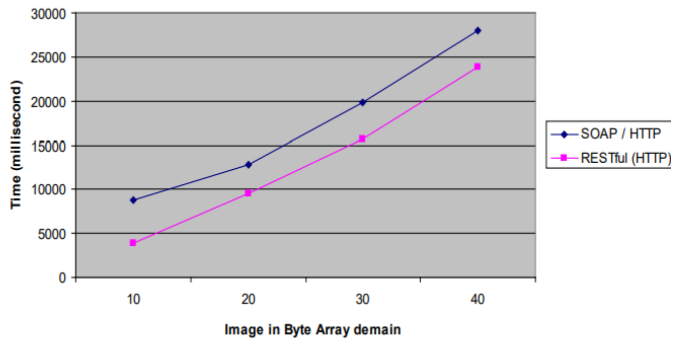


Figure 8: Response Time Comparison Plot

The upload time observed in REST for all the varying sizes of the Bytes Array is found to be lower than that of the SOAP protocol. This reduction can be attributed to the lightweight nature of the REST Protocol.

IV. CONCLUSION AND FUTURE SCOPE

The paper presented hence provides a template to create an API based on the REST Architecture. It walks through the steps and stages usually involved in developing an API, from the code stage till the deployment stage. The objective of this paper is to provide an overview, or an approach for end to end development. Further information of each of these tools can be found on their website documentation. As cited in the results, the REST Architecture, is the ideal choice for quick, lightweight and responsive applications, over applications that utilize the SOAP Protocol.Tech is evolving and so should the methods used to create software. The QA Automation suite, and some of the tech discussed here are based on Java. These are established methods to perform the task required, but nevertheless, slow. New software based on quick, lightweight, languages such as JavaScript or TypeScript are in need. Further, new and improved cloud services and CI/CD protocols are nascent, but growing. The impact these services will make are yet to be observed.

REFERENCES

[1] Rathod, Digvijaysinh. "Performance evaluation of restful web services and soap/wsdl web services." International Journal of Advanced Research in Computer Science 8.7 (2017): 415-420.

[2] Potti, Pavan Kumar, et al. "Comparing performance of web service interaction styles: Soap vs. rest." Proceedings of the conference on information systems applied research issn. Vol. 2167. 2012.

[3] Ofoeda, Joshua, Richard Boateng, and John Effah. "Application programming interface (API) research: A review of the past to inform the future." International Journal of Enterprise Information Systems (IJEIS) 15.3 (2019): 76-95.

[4] Masse, Mark. REST API design rulebook: designing consistent RESTful web service interfaces. " O'Reilly Media, Inc.", 2011.

[5] Doglio, Fernando, Doglio, and Corrigan. REST API Development with Node. js. Vol. 331. Apress, 2018.

[6] Surwase, Vijay. "REST API modeling languages-a developer’s perspective." Int. J. Sci. Technol. Eng 2.10 (2016): 634-637.

[7] Ofoeda, Joshua, Richard Boateng, and John Effah. "Application programming interface (API) research: A review of the past to inform the future." International Journal of Enterprise Information Systems (IJEIS) 15.3 (2019): 76-95.

[8] Arcuri, Andrea. "RESTful API automated test case generation." 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2017.

[9] Soni, Anshu, and Virender Ranga. "API features individualizing of web services: REST and SOAP." International Journal of Innovative Technology and Exploring Engineering 8.9 (2019): 664-671.

[10] Rodríguez, Carlos, et al. "REST APIs: a large-scale analysis of compliance with principles and best practices." International conference on web engineering. Springer, Cham, 2016.