

CLOUD-BASED WEB SERVICES

Adam Vimpany
ACCA6024 1810613

TABLE OF CONTENTS

Introduction.....	2
REST.....	2
SOAP	3
GraphQL.....	3
Discussion	3
Conclusion.....	4
Bibliography.....	5
Appendix 1: Video Link.....	7

INTRODUCTION

Application programming interfaces (APIs) are protocols that communicate with applications and systems, enabling the exchange of data between them. An API is an interface that can standardise communication between web applications by providing frameworks and libraries containing functions, data and features that can be implemented [1]. An application must *call* an API to send or receive data, and to do so it must correctly adhere to the API protocol [2]. They are essential for the functioning of many web applications that are commonly used today [2].

There are three main types of API protocols: REST, SOAP and RPC. RPCs are the oldest type of API and are no longer commonly used in modern applications [2], while REST is a well-established API that is commonly used today. GraphQL is an opensource query language that can be used to create APIs that is increasing in popularity due to its flexibility [2].

This report aims to identify the main features of GraphQL, REST and SOAP APIs as used in web applications, and analyse their relative advantages and constraints.

REST

Representational State Transfer (REST) is an architectural style which introduces constraints on the way data is handled in order to allow different, distributed applications to communicate with each other in a standardised way [3]. By introducing limitations on how applications that use the architecture are developed, it becomes much easier for a developer to maintain and update applications, while maintaining its interoperability [4]. These constraints define how the system should work, but do not limit the technologies that may be used: at the time when Roy Fielding [4] outlined the theory of RESTful systems in his doctoral dissertation, the web was already an existing example of a RESTful system; Fielding took the step of describing explicitly what the requirements of the architecture should be to enable development of interoperable networked applications, he did not dictate the technologies that should be used to implement them [3].

A RESTful system should adhere to six architectural constraints: uniform interface, client-server, stateless, cacheable, layered system and code on demand (the only optional constraint) [3].

The uniform interface constraint simplifies the architecture [3] by requiring a single uniform resource identifier (URI) that is unique to each resource that will be used. [5] Resources can be identified by identifying nouns in the system requirements [3]. URIs should be logical and follow recommended guidelines, such as naming conventions and data formats, for example XML [5]. URIs should be able to be accessed by a client through a generic interface using generic, consistent methods [3]: a common approach is to use **HTTP GET** to request data [5]. An application will interact with a client via a server by providing data through hypermedia: any client with the capability to understand the hypermedia will be able to receive the data, and this therefore decouples the client and server [5]. The server hypermedia responses will control the actions available to the client, in what is known as Hypermedia as the Engine of Application State (HATEOAS), which is an essential component of the uniform interface constraint [6].

The client-server constraint means that client and server applications must be decoupled (as briefly mentioned above) to allow them to be developed independently from one another: this is closely linked to the uniform interface constraint, because clients should only be able to access URIs from the interface, and so the server can be changed as long as the interface remains the same [5].

All interactions between the client and server should be stateless: this means that the server does not store user sessions and all requests are treated as new [2]. Where there is a need for an application to be stateful, for example if a user must log in to use the service, this should be handled client side and every request made by the client should contain all information required for the request each time, such as authentication details [5]. This constraint ensures that the service is scalable.

A RESTful system must also be cacheable to avoid repetitive requests from a client for the same resource [3], thereby improving its performance by reducing the load on the server [5].

The final non-optional requirement is that the system is layered. A layered system can be implemented in a way that is most appropriate for the requirements of the networked application [3],

for example by using different servers for authentication, security policies, or stored data [5], and this flexibility ensures that the system is scalable and does not sacrifice performance [3].

The optional constraint is code on demand. The client may be allowed to request and download code from the server to execute. This is optional as it may not be appropriate for some types of applications [4], but a common use of this constraint is to download JavaScript code to run applets client side [7].

SOAP

Simple Object Access Protocol (SOAP) is a protocol that precedes REST and is a web services API that has been commonly used since the 1990s [2]. It is becoming less commonly used in modern APIs [2], for reasons that will be discussed in the discussion section of this report. SOAP is a protocol that follows Service-Oriented Architecture (SOA) principles: applications are providers of services, and they interact using web services [8]. SOA requires that services must be *self-describable* and be suitable for dynamic detection and retrieval. They use XML to describe interfaces and relationships, and the interactions between services occurs in the transport layer using the HTTP protocol [8].

The SOAP protocol works above the transport layer and creates, sends and receives XML-based messages to enable communication between a client and a server [8]. The protocol is therefore lightweight and simple [9]. The XML message is structured into parts: the envelope, header and body. The envelope is the main message element [9]. The header is optional and may contain additional attributes which are not necessarily agreed between the client and server [9]. SOAP messages are able to mark whether these attributes in the header are mandatory or optional. The body contains the essential information required for the recipient of the message [9].

GraphQL

GraphQL is a data query language that is used for APIs that is becoming more and more common because of the level of control and flexibility it offers to a developer above REST and other APIs [2]. GraphQL can use the request-response model used by SOAP and REST APIs but can also push messages out to clients using a subscription model [10].

GraphQL was designed to be product-centric and prioritises both the requirements of the final front-end applications and the developers who use the language [11]. GraphQL queries are hierarchical, to match the natural structure of applications themselves [11] and the response is shaped like the data itself. Data structures are defined explicitly: strong-typing ensures that the GraphQL tools are able to validate requests and provide only the data requested by the client and no more [11]. This type-system ensures that the response is client-specified, which means that the server sends follows the structure of the client query in its response [11]. Finally, the type-system is introspective because it allows the type system to be queried using the GraphQL language itself [11].

DISCUSSION

SOAP is less commonly used in modern APIs because it is a stricter, more heavyweight protocol [2]. For some applications, this may be desirable as it is therefore very standardised, unlike REST APIs [12]. One study found that many HTTP-based APIs described as REST APIs were not truly RESTful as they did not fully adhere to these architectural constraints and concluded that there was a significant gap between the theory and real-life implementation of REST technologies [13]. Although REST architecture is more flexible than SOA, in large, interconnected systems such as those found in healthcare information systems, the constraints inherent to both SOAP and REST architectures can contribute to increased complexity, which may make maintenance and changes difficult [14]. Mukhiya *et al.* posit that the GraphQL standard offers a solution to these challenges, as it is more flexible, scalable and faster than REST or SOAP [14]. However, at the time of writing, GraphQL based APIs had not yet been adapted for real-world healthcare information systems [14], and therefore further research is required to determine the usefulness of GraphQL in this area. Being a new technology, it is not as well-supported as older technologies such as SOAP and REST [10]. However, there have been some user studies that support the ease of use for GraphQL, even when the user was only experienced with REST and not GraphQL [15].

SOAP and GraphQL APIs, unlike REST APIs, are independent of language, platform and transport: REST APIs are required to use HTTP which may not be suitable for all types of applications [12, 10].

Meanwhile, SOAP and GraphQL can still use HTTP if desired, since it is widely supported by most operating systems.

REST is scalable because every request sent by client contains all required information every time, so the server does not need to manage the state, however the drawback of this is network performance as this can result in overfetching, where more data is supplied than requested [14]. GraphQL is gaining ground because it has been designed to solve this limitation of REST APIs by strong-typing (only sending the data that was requested) [11]. However, the connection must be kept open in GraphQL because it is stateful, unlike REST APIs and so this has a detrimental effect on resources [14].

REST and SOAP APIs often need to make multiple requests of a server, which can also have a detrimental effect on performance [10]. GraphQL eliminates the need to send multiple requests to fetch data by offering subscriptions to push data to clients automatically [10]. This has been particularly important with the ever-increasing use of mobile web applications, because computing power and memory resources are very limited on these devices [13] and was the impetus for Facebook to begin the development of GraphQL [10].

The benefit of a REST API is that it makes caching very easy. Caching has a beneficial effect on performance by mitigating against the need for multiple requests to a server or responses to a client. Requests are sent to a specific URI and the cache can then be linked to this at both client and server level [4]. Caching is more difficult with GraphQL APIs, and so the performance benefits discussed above may be somewhat negated. However, caching may be less desirable for some systems where data kept on the server is constantly in flux, and caches holding out of date data will be a waste of resources [16]. A study by Heredia *et al.* stated that GraphQL performance was greater than REST API with multiple clients with no caching [17].

CONCLUSION

GraphQL is a promising technology, with some proven improvements in performance and flexibility over older technologies such as SOAP and REST APIs. It is more appropriate for web-based mobile applications because it is less resource intensive, but for other systems such as healthcare information systems there is not yet enough evidence to decisively support its use, and a standardised system using SOAP may be more appropriate. It is important to consider the importance of support for these systems before embarking on development: GraphQL has been shown to be easy to use for users who have no experience with it, but REST and SOAP APIs are well-established and better supported.

Consideration should be made before implementing an API as to the importance of caches: where caching is essential to prevent duplicate requests, REST API is recommended, but for some systems where data is constantly changing this may be less of a concern, and SOAP or GraphQL could be used instead.

BIBLIOGRAPHY

- [1] M. P. R. & Y. B. Chhetri, "Recommending reference API documentation," *Empirical software engineering*, vol. 20, no. 6, pp. 1558-1586, 2015.
- [2] J. Davidse, "API Fundamentals," IBM Developer, 16 12 2020. [Online]. Available: <https://developer.ibm.com/articles/api-fundamentals/>. [Accessed 10 11 2022].
- [3] B. M. Balachandar, *RESTful Java Web Services - Third Edition : Master Core REST Concepts and Create RESTful Web Services in Java*, Birmingham, UK: Packt Publishing, Limited, 2017.
- [4] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures: Doctoral dissertation," University of California, Irvine, 2000.
- [5] L. Gupta, "REST Architectural Constraints," REST API Tutorial, 09 03 2022. [Online]. Available: <https://restfulapi.net/rest-architectural-constraints/>. [Accessed 10 11 2022].
- [6] J. Thijssen, "What is HATEOAS and why is it important for my REST API?," *restfulcookbook.com*, [Online]. Available: <https://restcookbook.com/Basics/hateoas/>. [Accessed 11 11 2022].
- [7] J. Thijssen, "What is the code-on-demand constraint, and why is it an optional constraint?," *restcookbook.com*, [Online]. Available: <https://restcookbook.com/Basics/codeondemand/>. [Accessed 11 11 2022].
- [8] M. S. Ivanov, "Architecture of Service-Oriented Applications," *Automatic documentation and mathematical linguistics*, vol. 41, no. 6, pp. 251-254, 2007.
- [9] D. E. G. K. A. L. N. M. H. F. N. S. T. & D. W. Don Box, "Simple Object Access Protocol (SOAP) 1.1," in *W3C Note 08 May 2000*, 2000.
- [10] J. M. J. A. J. H. H. L. & K. T. Riku Ala-Laurinaho, "Comparison of REST and GraphQL Interfaces for OPC UA," *Procedia computer science*, vol. 160, pp. 338-345, 2019.
- [11] GraphQL Contributors, "GraphQL Specification October 2021," 10 2021. [Online]. Available: <http://spec.graphql.org/October2021/>. [Accessed 12 11 2022].
- [12] SoapUI, "SOAP vs REST 101: Understand The Differences," SoapUI, [Online]. Available: <https://www.soapui.org/learn/api/soap-vs-rest-api/>. [Accessed 10 11 2022].
- [13] M. B. F. D. F. C. J. C. T. L. C. & G. P. Carlos Rodriguez, "REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices," *ICWE 2016. Lecture Notes in Computer Science*, vol. 9671, 2016.
- [14] F. R. V. K. I. P. A. R. Y. L. Suresh Kumar Mukhiya, "A GraphQL approach to Healthcare Information Exchange with HL7 FHIR," *Procedia computer science*, vol. 160, pp. 338-345, 2019.

- [15] G. B. & M. T. Valente, "REST vs GraphQL: A Controlled Experiment," *2020 IEEE International Conference on Software Architecture (ICSA)*, 2020.
- [16] A. S. & M. W. Rainer Schiekofner, "REST based OPC UA for the IIoT," *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018.
- [17] E. F.-G. & A. R. S. Jaime Sayago Heredia, "Comparative Analysis Between Standards Oriented to Web Services: SOAP, REST and GraphQL," *Communications in Computer and Information Science*, vol. 1193, 2020.

APPENDIX 1: VIDEO LINK

For the practical section of this assignment, a dynamic webpage, using AJAX and a REST API was created to exchange data with webservices. A demonstration of the website and code explanation can be found in the following link.

https://www.youtube.com/watch?v=p_N5eXthADQ