

Cloud Based Web Services (ACCA6024)

Hywel Jones 1903723

15/11/2022

Table of Contents

Cloud Based Web Services (ACCA6024)	1
Table of Contents.....	2
Web Services.....	3
SOAP	3
Advantages	3
Disadvantages	3
REST	3
Advantages	4
Disadvantages	4
GraphQL.....	4
Advantages	4
Disadvantages	5
Dynamic Ajax website.....	5
Bibliography.....	6

Web Services

There have been several different types of protocols established for web services over the years as technology has advanced. These have often been developed to try to standardise how data is transferred between different endpoints across the industry.

SOAP

SOAP, formerly known as Simple Object Access Protocol, was one of the first protocols developed in the industry. Developed by Microsoft and initially released as XML-RPC in 1998 it was not officially part of the web standard until 2003, as version 1.2, when it became a W3C recommendation [1].

SOAP is a way of sending data, in the XML format, over the HTTP protocol to remotely call procedures on other machines.

Advantages

One of the principal advantages of SOAP is that it can be easily used to remotely call procedures on other machines. These machines could be using different operating systems as well as be coded in different languages, yet SOAP allows interoperability and ease of use. This is because by using XML it became easy to specify the structure of the remotely available function calls and the structure of the data types to be exchanged [7].

Another advantage of the protocol is that it can run across different message protocols, that is it can run over HTTP, SMTP, UDP and others as the message body is what contains the details of the procedure call [4]. Due to this, new procedures (or methods) can be added easily and the only change needed to call these new procedures is to send the correct body in the message.

Disadvantages

Due to its origin as a way to remotely call procedures, SOAP does not necessarily follow a web-oriented approach and can often time be closer to a programming interface than a web-based protocol. For example, most SOAP services only have one URI, rather than having a URI defined per resource, and then proceed to define a series of actions that can be called on that URI, similar to remotely calling methods on a class. [4]

SOAP is also limited in that it only uses the POST HTTP verbs, this means that it is not possible to determine the effect of a SOAP call by examining the request, the code itself would need to be examined. For example, a HTTP POST message sent to a SOAP URI <http://localhost:1080> would be calling an action defined by XML in the message body. If that action was poorly named even extracting the action name from the body might still not be enough to determine the effect of the message without looking through the code on the API. [3]

Another drawback is that the protocol is based on XML and cannot be changed freely to another format such as JSON, meaning the message body is often rather large compared to the data it actually contains. Having the action defined in the body also means that testing a SOAP API is not very easy as the XML body needs to be regenerated for each request, which is a non-trivial operation for most testing software, such as Postman.

REST

REST, which stands for Representational state transfer, is an architectural style that defines how an interface behaves and its state during processing requests. Even though REST was being actively developed as a concept from the early days of the web it was only released as a style definitively in 2000 in a PhD dissertation by Roy Fielding [2].

REST leverages and builds on the architecture of the web in that design concepts such as getting, putting, posting and deleting data utilize the corresponding HTTP verbs available on the web infrastructure. It also specifies that resources should have their own URI to uniquely identify a resource. Combining a well-defined resource URI with HTTP verbs allows for some very natural and intuitive concepts such as setting a GET HTTP request to a URI <http://localhost:1080/books> would return a list of books held on the endpoint [3].

Advantages

As mentioned above that REST leverages URIs for resources and HTTP verbs for actions against those resources allows REST services to have a certain level of simplicity. It should be relatively easy for new developers to know what a certain RESTful call does and the expected result from that call.

The definition of REST means that services are stateless, this means that the context around an API call should not change the result. Combined with the above point about simplicity and RESTs leverage of URIs and HTTP verbs this can make REST APIs very easy to test using tools such as Postman. These tests are also easily repeatable as they should be stateless and thus can make using full unit test suites easy to integrate [7].

The REST protocol makes no hard choices on the format of the data transferred either, this allows for XML or JSON to be used which is an advantage over some of the older protocols.

Disadvantages

One of the stated advantages of REST is that it is stateless, but this can also be a disadvantage when certain API calls require complex contextual information to execute. This is an additional layer that would need to be built in either through increasing the chattiness of the application or straying from the strict RESTful rules.

Some of the other disadvantages of REST are more theoretical in nature, due to its reliance on HTTP verbs some more advanced scenarios could create some problems. As web applications embrace increasingly more P2P style applications, HTTP verbs could be insufficient to describe some of the intra machine API calls leading to some ambiguity on what those calls accomplish. Similarly, the only encryption available for REST services is the HTTP encryption, SSL/TLS is end to end encryption which detracts from some of the simplicity advantages stated above.

GraphQL

GraphQL was developed in 2012 by Facebook and released in 2015 as a data query language for APIs to help mitigate some of the disadvantages or limitations of RESTful services. In 2018 it was moved to the GraphQL Foundation which is hosted by the Linux Foundation [5].

GraphQL has two components, one is the schema and the other is the resolver. The schema is a definition of the model or object that can be queried whilst the resolver parses the query and accesses the underlying data store. It allows for the caller of the API to specify a query and what the expected result set should be [6].

Advantages

One of the design features for GraphQL is that it can support very complex queries where other protocols might have needed several API calls to resolve the same query. The model can be made as complex or as simple as required, this can be done regardless of the underlying data store which can allow front end queries to be significantly simpler than the complexity required on the backend data store.

The caller of the GraphQL API can define their own result set in the request, this lets the API cut down on chattiness by responding with only the data requested and can also cut down on the size of the data sent over the network.

The data model created is strongly typed, this also allows queries to be validated and makes consolidation between front-end and backend easier during the development phase. [8]

Disadvantages

The drawbacks of GraphQL are linked to its benefits, the complexity of setting up a schema and a resolver means that a significant amount of development might be needed to establish the API. In complex systems with several object models that have complicated object graphs the GraphQL resolver can be very complicated and the APIs can be difficult to maintain and document adequately.

Although having said the above, using a tool called GraphiQL can allow developers to browse the API schema and thus mitigate the complexity and uncertainty of how to use the API. As discussed by Gleison Brito the availability to browse a statically typed schema can lead to GraphQL actually being easier and faster to develop against than REST services [9].

Similarly to SOAP it also pushes the contextual information about what the API call does into the body of the request, making testing and reading simplicity of GraphQL endpoints difficult without extra knowledge of the endpoint itself.

Another disadvantage due to pushing the query into the body of the request is that queries cannot be cached easily, another layer of development would be needed (such as assigning globally unique numbers to entities) to allow efficient caching [8].

Dynamic Ajax website

The youtube video demonstrating the working app and explaining the code can be found here:
<https://youtu.be/VqL9jz8zz2E>

Bibliography

- [1] Wikipedia, "SOAP", [Online] Available: <https://en.wikipedia.org/wiki/SOAP> [Accessed 16/11/2022].
- [2] Wikipedia, "Representational state transfer", [Online] Available: https://en.wikipedia.org/wiki/Representational_state_transfer [Accessed 16/11/2022].
- [3] J. Flanders, RESTful.NET, 1st edition. Sebastopol, California: O'Reilly Media, 2009.
- [4] F. Curbera et al, "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," IEEE Internet Comput., vol. 6, (2), pp. 86-93, 2002. Available: <https://www.proquest.com/scholarly-journals/unraveling-web-services-introduction-soap-wsdl/docview/197336114/se-2>. DOI: <https://doi.org/10.1109/4236.991449>.
- [5] Wikipedia, "GraphQL", [Online] Available: <https://en.wikipedia.org/wiki/GraphQL> [Accessed 16/11/2022].
- [6] D. Chaves-Fraga, F. Priyatna, A. Alobaid, and O. Corcho, "Exploiting Declarative Mapping Rules for Generating GraphQL Servers with Morph-GraphQL," International journal of software engineering and knowledge engineering, vol. 30, no. 6, pp. 785–803, 2020, doi: 10.1142/S0218194020400070.
- [7] M. zur Muehlen, J. V. Nickerson, and K. D. Swenson, "Developing web services choreography standards—the case of REST vs. SOAP," Decision Support Systems, vol. 40, no. 1, pp. 9–29, 2005, doi: 10.1016/j.dss.2004.04.008.
- [8] R. Ala-Laurinaho et al, "Comparison of REST and GraphQL Interfaces for OPC UA," Computers, vol. 11, (5), pp. 65, 2022. Available: <https://www-proquest-com.ezproxy.uwtsd.ac.uk/scholarly-journals/comparison-rest-graphql-interfaces-opc-ua/docview/2670132497/se-2>. DOI: <https://doi-org.ezproxy.uwtsd.ac.uk/10.3390/computers11050065>.
- [9] G. Brito and M. T. Valente, "REST vs GraphQL: A Controlled Experiment," ArXiv.Org, 2020. Available: <https://www-proquest-com.ezproxy.uwtsd.ac.uk/working-papers/rest-vs-graphql-controlled-experiment/docview/2376046899/se-2>.