

Assignment 1 Report 1909705

by Sophie Michelle Slaughter

Submission date: 10-Dec-2021 12:09PM (UTC+0000)

Submission ID: 165741318

File name: 30198_Sophie_Michelle_Slaughter_Assignment_1_Report_1909705_382340_1761435770.docx
(1.44M)

Word count: 2825

Character count: 18556



Image: [1] - Web Services

BSC Applied Computing
Cloud-Based Web Services
ACCA6024

Assignment 1 - Report
Sophie Slaughter
1909705

Table of Contents

Introduction	2
Image of Dynamic Webpage	3
SOAP	4
Benefits	4
Drawbacks	4
REST	4
Benefits	5
Drawbacks	5
GraphQL	5
Benefits	5
Drawbacks	6
Compare & Contrast	6
Reference List	17

Introduction (67 words)

Within this report, a detailed final image of the dynamic webpage will be produced, the dynamic webpage is to include AJAX and a REST API to exchange data with web services. The code that was used to create the webpage will be provided at the bottom of the report along with a video, demonstrating what code does.

The report side of this assignment is to provide an in-depth explanation which will explain the comparisons and contrasts of SOAP, REST and GraphQL, and as well as discovering the benefits and drawbacks of them all.

Image of Dynamic Webpage

Netflix Members!



Add New Members

Member ID:

Title:

Forename:

Surname:

Date of Birth: 

Email Address:

Contact Number:

Add Member

Waiting ..

Delete an Existing Member

Member ID:

Delete Member

Waiting ..

Read Information of an Existing Member

Member ID:

Get Member Information

Waiting ..

SOAP (143 -200)

As explained on Tutorials Point [2], SOAP is a shortened “acronym for Simple Object Access Protocol”, which is an “XML-based messaging protocol for exchanging information among computers”. It is a way for computer devices to “access web services over HTTP” and “how web services talk to each other or talk to client applications that invoke them”, where it has “some specification which could be used across all applications”, as explained on Guru99 [3]. As found on Tutorials Eye [4], it is explained that the SOAP “XML based protocol consists of three parts”, which are “an envelope”, “a set of encoding rules”, and “a convention”. The envelope “defines what is in the message and how to process it”, the set of encoding rules are “used for expressing instances of application-defined data types”, and lastly the convention is “used for representing procedure calls and responses”.

Benefits(99 -150)

As found techtutorial [5], it is explained that SOAP “uses standard internet HTTP protocol”, which is a benefit as it “eliminates firewall problems” and when the “protocol is used as the protocol binding, an RPC call will be automatically assigned to an HTTP request”, which the response then would be assigned to an HTTP reply. As mentioned within another resource [6], SOAP has been described as “platform independent” and “language independent”, these are benefits because there will be easier communication between the protocol and the internet, and as well as being able to interact with other “programming language applications” [7].

Drawbacks(94 -150)

SOAP uses “a detailed XML format” which is a disadvantage as it creates the protocol “significantly slower than other types of middleware standards”, as explained within the following source [5]. In addition to the previous disadvantage as explained on another source[8], SOAP had been described as “slow”, since “SOAP used the XML format which needs to be parsed and is lengthier too which makes SOAP slower than COBRA, RMI or IIOP”. The second disadvantage mentioned is that SOAP “depends on WSDL and does not have any standardized mechanism for dynamic discovery of the services”.

Debate (54)

In comparison to REST and GraphQL, SOAP “has a number of security extensions” which REST and GraphQL do not, where SOAP “provides privacy and integrity inside transactions while allowing for encryption on the message level”. SOAP has the more preferred “language- and platform-agnostic”, where web-based services “allows SOAP to communications and make responses” independently[9].

REST(164 -200)

In contrast to SOAP, “REST is an acronym for REpresentational State Transfer” and is an “architectural style for distributed hypermedia systems” as described with the following source by REST API Tutorial [10]. Unlike SOAP, “REST does not enforce any rule regarding how it should be implemented at the lower level”, there are just “high-level design guidelines” in which developers are left to create their own implementations [11]. In contrast to GraphQL, it is explained on the following source [12] that “GraphQL requires less effort to implement remote service queries when compared to REST”, which is estimated around 9 vs 6 minutes in comparison. As shown in the table provided by the following article [13], a table had been produced to compare the response time between REST and GraphQL, where “50 requests were run to get their response times for each API architecture”;

TABLE I: Average response time (ms): REST & GraphQL.

Query	REST	GraphQL
Query 1	171.16	176.96
Query 2	627.00	606.34
Query 3	225.44	144.88
Query 4	338.16	388.46

From the results of the table, this “summarizes the average response time of running the same query on GraphQL and REST APIs”.

Benefits(93 -150)

According to the following resource [14], “systems that follow the REST paradigm are stateless”, meaning that it is a benefit, where “the server does not need to know anything about what state the client is in and vice versa”, where both the “server and the client can understand any message received, even without seeing the previous message”. REST web services “permits different data formats” for example “Plain, Text, HTML, XML, and JSON”, and “RESTful Web Services are fast because there is no strict specification like SOAP. It consumes less bandwidth and resource” [7].

Drawbacks(106 -150)

As found on Krify [15], two disadvantages have been detected from using the REST protocol. REST has “lack of state”, meaning most “web applications require stateful mechanisms”, for example suppose a website has an option to purchase items, it must know the required number of items before any purchases are made, but REST cannot which makes this “burden of maintaining the state lies on the client, which makes the client application heavy and difficult to maintain”. REST has “lack of security”, where it does not “impose security”. It is “appropriate for public URLs, but it is not good for confidential data passage between client and server”.

Debate (60)

In comparison to SOAP and GraphQL, REST can “support multiple formats for storing and exchanging data” and is the reason why REST is currently a “prevailing choice for building public APIs”. “Discoverability” is another reason why REST tops SOAP and GraphQL, as “communication between the client and server describes everything so that no external documentation is required to understand” [9].

GraphQL(140 -200)

As found on the following article [16], it is explained that GraphQL is short for “Graph Query Language” using the format JSON and it is “a framework for developing Web Services, which works on the domain model level rather than the functions”. “GraphQL provides a complete and understandable description of the data in you API”, which provides the client with the “power to ask for exactly what they need and nothing more”, as presented on GraphQL[17]”. It is explained when put to test against REST API while undergoing “the display operation under heavy load conditions and while downloading small portions of data”, that GraphQL had the better performance than REST[16] . On GraphQL [18], it is said that “GraphQL isn’t tied to any specific database or storage engine” but is backed up by the existing code and data already present.

Benefits(93 -150)

As mentioned on Java T Point [19], an advantage mentioned is that “GraphQL is way faster than other communication APIs” as it requests for the client to cut down the “request query by choosing only the specific fields” that needs to be queried. Another benefit is the GraphQL is the “best for complex systems and microservices”, where it “unifies them and hides their complexity”. When

GraphQL queries are requested within a server, “the server returns the response in a simple, secure, and predicable shape”, which creates GraphQL “really easy to learn and use”.

Drawbacks(115 -150)

As found within the following resource [19], GraphQL has been described with three disadvantages and these are; “GraphQL Query Complexity”, “GraphQL Caching”, and “GraphQL Rate Limiting”. The query complexity is where clients try for a more complex query but as “it is just a simple query language” and not a “replacement for server-side databases”, the query will have to “access multiple fields in one query” but should have a “maximum query depth” to “stop inefficient requests”. GraphQL makes it “more complicated to implement a simplified cache” as “each query can be different”. Lastly, GraphQL does not specify rate-limiting requests, where “you can simply specify that we allow only this amount of requests in one day”.

Debate (82)

In comparison to REST, GraphQL has “many benefits over REST APIs” and one of these benefits is that the “clients have the ability to dictate exactly what the need” [20], as mentioned above as well. In comparison to REST, GraphQL is more reliable when requesting responses as “REST responses contain too much data or sometimes not enough data, which creates the need for another request”, whereas “GraphQL solves this problem by fetching only the exact and specific data in a single request”[19].

Compare & Contrast (295 - 300)

As found within the following article [21], it compares SOAP and REST web services on the overall performance of both, where it is explained that both “approaches handle service interactions differently”. SOAP is a “standardized for constructing and processing messages independently of the technological capabilities of the receiver”, where it can also be used to “work on top of a variety of application layer protocols such as RPC, HTTP, or SMTP”, whereas for REST, it is a “set of principles for designing Web applications”.

As found within another article [22], an example provided on the article is shown where the “comparative analysis between SOAP, REST and GraphQL web services aims to assess the effectiveness of data transfer capabilities”, and the “methods used were a systematic mapping to define the metrics to use for comparison such as response time and performance”. As the results of the comparison was calculated, it was “that GraphQL has a faster response time and better performance than web services based on SOAP and REST”. This comparison is what will “help developers choose the optimal and appropriate technology based on performance of web services and other metrics that influence an essential aspect such as software quality”.

A contrast between REST and GraphQL was found within the following article [23], where the aim was to “identify the differences, in terms of load time, amount of data traffic, and resources consumption, between REST and GraphQL applications”. Two API prototypes had been implemented to achieve the required results by using the REST design in one and GraphQL in the other, to be used as a “mechanism to respond to queries”. The results of the test had shown that “GraphQL performed better in the most evaluated scenarios” that were created to obtain the most accurate results.

Appendices

1909705java.js

function AJAX()

```
{  
  
    $.ajaxSetup({cache:false});  
  
    $("#post-button").click(function(e)  
    {  
        SendPost();  
        e.preventDefault();  
        return false;  
    });  
  
    $("#delete-button").click(function(e)  
    {  
        DeletePost();  
        e.preventDefault();  
        return false;  
    });  
  
    $("#getid-button").click(function(e)  
    {  
        GetPost();  
        e.preventDefault();  
        return false;  
    });  
};
```

function SendPost()

```
{  
  
    let data = {
```

```

        member_id: parseInt($("#member-id-text").val()),
        title: $("#title-text").val(),
        forename: $("#forename-text").val(),
        surname: $("#surname-text").val(),
        dob: $("#dob-text").val(),
        email_address: $("#email-text").val(),
        contact_number: $("#contact-number-text").val()
    };

    $.ajax({
        cache: false,
        type: "POST",
        url: "https://jsonplaceholder.typicode.com/posts",
        data: JSON.stringify(data),
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        success: GotPostResults,
        error: (req, status, error) => window.alert("AJAX Error")
    });
}

function GotPostResults(json, status, req)
{
    let html = "";

    html += "Member Id:" + json.memberid + "<br/>"
        + "Title:" + json.title + "<br/>"
        + "Forename:" + json.forename + "<br/>"
        + "Surname:" + json.surname + "<br/>"
        + "Date of Birth:" + json.dob + "<br/>"
        + "Email Address:" + json.emailaddress + "<br/>"

```

```
        + "Contact Number:" + json.contactnumber + "<br/>";

        $("#postresults").html(html);
    }

    function DeletePost()
    {
        let postId = parseInt($("#deleteid-text").val());

        $.ajax({
            cache:false,
            type:"DELETE",
            url:`https://jsonplaceholder.typicode.com/posts/${postId}`,
            data:'',
            contentType:"application/json; charset=utf-8",
            dataType:"json",
            success:(json, status, req)=>$("#deleteresults").html("<p>Success</p>"),
            error:(req, status, error)=>window.alert("AJAX error")

        });
    }

    function GetPost()
    {
        let postId = parseInt($("#getid-text").val());

        $.ajax({
            cache: false,
            type: "GET",
            url: `https://jsonplaceholder.typicode.com/posts/${postId}`,
            data: "",
            contentType: "application/json; charset=utf-8",
```

```

        dataType: "json",
        success: GotGetResultById,
        error: (req, status, error) => window.alert("AJAX error")
    });
}

function GotGetResultById(json, status, req)
{
    let html="";

    html+="Member Id:" + json.memberid + "<br/>"
        + "Title:" + json.title + "<br/>"
        + "Forename:" + json.forename + "<br/>"
        + "Surname:" + json.surname + "<br/>"
        + "Date of Birth:" + json.dob + "<br/>"
        + "Email Address:" + json.emailaddress + "<br/>"
        + "Contact Number:" + json.contactnumber + "<br/>";

    $("#postresults").html(html);
}

```

[1909705html.html](#)

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>Netflix Members</title>
```

```
</head>
```

```
<style>
```

```
body {  
  background-color: red;  
  text-align: center;  
  font-family: sans-serif;  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
}
```

```
#unsplashImage {  
  max-height: 500px;  
  border: 20px solid black;  
  transform: rotate(3deg);  
}
```

```
h1 {  
  font-size: 56px;  
}
```

```
.imageDetails {  
  background-color: black;  
  margin-top: 50px;  
  padding: 10px;  
  color: white;  
  text-decoration: none;  
  font-weight: bold;  
  width: fit-content;  
}
```

```
.imageDetails a {  
  color: #FFF;  
}
```

```
.imageDetails a:hover {  
  color: red;  
}
```

```
</style>
```

```
<body>
```

```
<h1>Netflix Members!</h1>
```

```
<div class="imageContainer">
```

```
<a id="imageLink" href="#">
```

```
<img alt="" id="unsplashImage" />
```

```
</a>
```

```
</div>
```

```
<form id="postform">
```

```
<h3>Add New Members</h3>
```

```
<label for="memberid">Member ID:</label> <!-- -->
```

```
<input id="memberid" type="number"/><br>
```

```
<label for="title">Title:</label>
```

```
<input id="title" type="text"/><br>
```

```
<label for="forename">Forename:</label>
```

```
<input id="forename" type="text"/></br>
```

```
<label for="surname">Surname:</label>
```

<input id="surname" type="text"/></br>

<label for="dob">Date of Birth:</label>

<input id="dob" type="date"/></br>

<label for="email">Email Address:</label>

<input id="email" type="email"/></br>

<label for="contactnumber">Contact Number:</label>

<input id="contactnumber" type="number"/></br>

<input id="post-button" type="submit" value="Add Member"/>

<div id="postresults">Waiting...</div>

</form>

<form id="deleteform">

<h3>Delete an Existing Member</h3>

<label for="deleteid">Member ID:</label>

<input id="deleteid" type="number"/>

<input id="delete-button" type="submit" value="Delete Member"/>

<div id="deleterresults">Waiting...</div>

</form>

<form id="getidform">

<h3>Read Information of an Existing Member</h3>

<label for="getid">Member ID:</label>

<input id="getid" type="number"/>

<input id="get-button" type="submit" value="Get Member
Information"/>


```

        <div id="getresults">Waiting...</div>

</form>

<script src="1909705java.js"></script>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<script>
    let clientID = "wrddskzj1Kkyl6Vg3loSbuQFgjk9E413fXhzmXHnH7M"; <!-- A client id is
required in order to gain access to the data provided by the API -->

    let endpoint = `https://api.unsplash.com/photos/random/?client_id=${clientID}`; <!--
- Where the client id is then used to create the link between the webpage and the API -->

    let imageElement = document.querySelector("#unsplashImage"); <!-- The image
element is being displayed within a variable here-->

    let imageLink = document.querySelector("#imagelink") <!-- This will help set the url
for the image itself -->

    fetch(endpoint)
        .then(function (response) {
            return response.json();
        })
        .then(function (jsonData) {
            imageElement.src = jsonData.urls.regular; <!-- This enables the
image to change after refreshing the webpage, where it shows that the API has been used correctly--
>

        })
        <!-- As using a promise, error handling is required as to capture any errors
that arise-->

        .catch(function (error) {
            console.log("Error: " + error);
        });

</script>

```

```

<script>

    let postform = document.getElementById('postform')    <!-- To get the refernce of
the forms-->

    postform.addEventListener('submit', function(e) { <!-- An event handler is used for
when the form submits, the function will be executed -->

        e.preventDefault() <!-- This will prevent auto submission of the form-->

        let memberid = document.getElementById('memberid').value
        let title = document.getElementById('title').value
        let forename = document.getElementById('forename').value
        let surname = document.getElementById('surname').value
        let dob = document.getElementById('dob').value
        let email = document.getElementById('email').value
        let contactnumber = document.getElementById('contactnumber').value

        $.post("https://jsonplaceholder.typicode.com/posts",
{memberid:memberid,title:title,forename:forename,surname:surname,dob:dob,email:email,contact
number:contactnumber}, function(data){

            console.log(data)

        });

    })
</script>

<script>

    fetch('https://jsonplaceholder.typicode.com/posts/1',{
        method:'DELETE'
    }).then(response=>{
        return response.json()
    }).then(data=>

```

```

        // this is the data we get after putting our data,
        console.log(data)
    );

</script>

<script>

    let getidform = document.getElementById('getidform') <!-- Refernce of the forms-->

    getidform.addEventListener('submit', function(e) { <!-- The function will be executed
-->

        e.preventDefault() <!-- Prevent auto submission of the form-->

        let getid = document.getElementById('getid').value

        $.get('https://jsonplaceholder.typicode.com/posts/${getid}', {getid:getid},
function(data){

            console.log(data)

        });

    })

</script>

</body>

</html>

```

Video Link

➔ <https://www.loom.com/share/0c650badc2844b84b9364aa8fecb68c5>

Reference List

- [1] "Introduction to SOAP-Based Web Services - DZone Integration." <https://dzone.com/articles/step-step-web-service-guide> (accessed Oct. 26, 2021).
- [2] "What is SOAP?" https://www.tutorialspoint.com/soap/what_is_soap.htm (accessed Oct. 29, 2021).
- [3] "SOAP Web Services Tutorial: What is SOAP Protocol? EXAMPLE." <https://www.guru99.com/soap-simple-object-access-protocol.html> (accessed Oct. 29, 2021).
- [4] "Introduction of SOAP | Tutorials Eye." <https://tutorialseye.com/introduction-of-soap.html> (accessed Nov. 12, 2021).
- [5] "advantages of SOAP - API Testing tutorial | SOAP testing |." <https://techtutorialz.com/advantages-and-disadvantages-of-soap/> (accessed Nov. 19, 2021).
- [6] "Simple Object Access Protocol (SOAP) Tutorial." <https://searchapparchitecture.techtarget.com/tutorial/Simple-Object-Access-Protocol-SOAP-Tutorial> (accessed Nov. 19, 2021).
- [7] "25 Mulesoft Interview Questions And Answers For Experienced." <https://codingcompiler.com/mulesoft-interview-questions-answers/> (accessed Nov. 19, 2021).
- [8] "What is SOAP? Why is SOAP required? Advantages of SOAP." <https://www.careerride.com/SOAP-What-is-SOAP.aspx> (accessed Nov. 23, 2021).
- [9] "Comparing SOAP vs REST vs GraphQL vs RPC API | AltexSoft." <https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/> (accessed Nov. 25, 2021).
- [10] "What is REST." <https://restfulapi.net/> (accessed Nov. 12, 2021).
- [11] "REST Architectural Constraints." <https://restfulapi.net/rest-architectural-constraints/> (accessed Nov. 12, 2021).
- [12] G. Brito and M. T. Valente, "REST vs GraphQL: A Controlled Experiment," *undefined*, pp. 81–91, Mar. 2020, doi: 10.1109/ICSA47634.2020.00016.
- [13] L. Vadlamani, B. Emdon, J. Arts, and O. Baysal, "Can GraphQL Replace REST? A Study of Their Efficiency and Viability", Accessed: Nov. 12, 2021. [Online]. Available: <https://github.com/Sri-Vadlamani/>
- [14] "What is REST? | Codecademy." <https://www.codecademy.com/articles/what-is-rest> (accessed Nov. 19, 2021).
- [15] "Advantages and Disadvantages of REST API." <https://krify.co/advantages-and-disadvantages-of-rest-api/> (accessed Nov. 25, 2021).
- [16] P. Stünkel, O. von Bargaen, A. Rutle, and Y. Lamo, "Journal of Object Technology GraphQL Federation: A Model-Based Approach", doi: 10.5381/jot.2020.19.2.a18.
- [17] "GraphQL | A query language for your API." <https://graphql.org/> (accessed Nov. 23, 2021).
- [18] "Introduction to GraphQL | GraphQL." <https://graphql.org/learn/> (accessed Nov. 23, 2021).

- [19] "GraphQL Advantages and Disadvantages - javatpoint." <https://www.javatpoint.com/graphql-advantages-and-disadvantages> (accessed Nov. 23, 2021).
- [20] "Advantages and Disadvantages of GraphQL | Stable Kernel." <https://stablekernel.com/article/advantages-and-disadvantages-of-graphql/> (accessed Nov. 23, 2021).
- [21] P. K. Potti, S. Ahuja, K. Umapathy, and Z. Prodanoff, "Comparing Performance of Web Service Interaction Styles: SOAP vs. REST", Accessed: Nov. 10, 2021. [Online]. Available: www.aitp-edsig.org
- [22] J. Sayago Heredia, E. Flores-García, and A. R. Solano, "Comparative Analysis Between Standards Oriented to Web Services: SOAP, REST and GRAPHQL," *Communications in Computer and Information Science*, vol. 1193 CCIS, pp. 286–300, Dec. 2019, doi: 10.1007/978-3-030-42517-3_22.
- [23] M. Diego, C. França, and E. da Silva, "Performance Evaluation of REST and GraphQL APIs Searching Nested Objects."

Assignment 1 Report 1909705

GRADEMARK REPORT

FINAL GRADE

87 / 100

GENERAL COMMENTS

Instructor

Ajax & REST (45): 37

Written Research (45): 40

Demo (10): 10

Feedback:
=====

Good report, could have identified appropriate characteristics for comparison of service types.

Slightly simplistic website but does demo the web service handling and includes a method for getting an image.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18
