Waqas Musharaf (U1561634)

CMS3407 – Web and Network Services – Assignment 2 – Report

Contents

[**UML:** 2](#_Toc31401658)

[Architecture Diagram: 2](#_Toc31401659)

[Class Diagram: 2](#_Toc31401660)

[Client Interface Wireframe: 3](#_Toc31401661)

[Sequence Diagram: 3](#_Toc31401662)

[Use Case Diagram: 4](#_Toc31401663)

[**Testing:** 4](#_Toc31401664)

[**Evaluation of Application:** 11](#_Toc31401665)

[**Discussion of Web Services:** 12](#_Toc31401666)

# **UML:**

## Architecture Diagram:

A screenshot of a cell phone

Description automatically generated

## Class Diagram:

A screenshot of a social media post

Description automatically generated

## Client Interface Wireframe:

A screenshot of a cell phone

Description automatically generated

## Sequence Diagram:

A picture containing next, table, sky

Description automatically generated

## Use Case Diagram:

A close up of text on a white background

Description automatically generated

# **Testing:**

Before running the REST client, you must start the REST service. The below screenshot demonstrates this:

A screenshot of a cell phone

Description automatically generated

Following this, the REST client(s) can be started. In a real-world scenario, the service would be permanently live, and many clients could potentially connect to it.

The below screenshot shows a REST client when first run:

A screenshot of a social media post

Description automatically generated

Now that the client is running, a request can be selected from the ‘Request’ dropdown box. To note, the ‘Parameters’ textbox is uneditable as no request is selected:

A screenshot of a cell phone

Description automatically generated

During this stage of testing, ‘Cat Fact’ has been selected as the request. To note, the ‘Parameters’ textbox remains uneditable as no parameters are required for the cat fact request. However, the ‘Submit’ button has become clickable as a valid request has been selected:

A screenshot of a cell phone

Description automatically generated

Following this, clicking the ‘Submit’ button results in a cat fact being displayed in the ‘Response’ textbox:

A screenshot of a cell phone

Description automatically generated

Having returned the cat fact, the client is ready to handle another request. As the request parameters are still valid, the ‘Submit’ button remains clickable. If the request parameters remain unchanged, clicking the ‘Submit’ button returns another cat fact:

A screenshot of a cell phone

Description automatically generated

If ‘Dictionary’ is selected as the request, the ‘Submit’ button becomes unclickable but the ‘Parameters’ textbox becomes editable. This is because the dictionary request requires an additional search parameter to return a definition:

A screenshot of a cell phone

Description automatically generated

If alphabetic input is provided to the ‘Parameters’ textbox, the ‘Submit’ button becomes clickable:

A screenshot of a cell phone

Description automatically generated

The validity of any string input into the ‘Parameters’ textbox is not checked by the client, meaning the input of any string will enable the ‘Submit’ button. However, the dictionary API will perform input validation and will return either a definition or an error depending on the validity of the input provided.

The below screenshot shows the response provided by clicking the ‘Submit’ button with a valid string input in the ‘Parameters’ textbox:

A screenshot of a cell phone

Description automatically generated

The below screenshot shows the response provided by clicking the ‘Submit’ button with an invalid string input in the ‘Parameters’ textbox:

A screenshot of a cell phone

Description automatically generated

If ‘IP Location’ is selected as the request, the ‘Submit’ button becomes clickable and the ‘Parameters’ textbox becomes editable. This is because the IP location request can operate with or without an additional IP parameter.

Additionally, as the ‘Request’ dropdown box has been edited, the ‘Parameters’ textbox has been reset:

A screenshot of a cell phone

Description automatically generated

The below screenshot shows the response provided by clicking the ‘Submit’ button without an additional parameter input:

A screenshot of a social media post

Description automatically generated

The validity of any input into the ‘Parameters’ textbox is not checked by the client. However, the IP location API will perform input validation (if any input is provided) and will return either IP location information or an error, depending on the validity of the input provided.

The below screenshot shows the response provided by clicking the ‘Submit’ button with a valid IP string input in the ‘Parameters’ textbox:

A screenshot of a cell phone

Description automatically generated

The below screenshot shows the response provided by clicking the ‘Submit’ button with an invalid string input in the ‘Parameters’ textbox:

A screenshot of a cell phone

Description automatically generated

# **Evaluation of Application:**

I built my client application as a REST client, written in C#.

REST was my web service communication architecture of choice mainly due to its compatibility with a wide range of data formats. The web APIs I planned to use mostly returned data in JSON format, which is compatible with REST, unlike REST alternatives such as SOAP which only work with XML.

C# was my programming language of choice for the development of the client, as I am more proficient in C# than in any other programming language. Also, I was already familiar with using HTTP web requests in C#, which I used to retrieve data from web APIs.

I created my service application as a REST service, to integrate with my REST client. I wrote the service application in Python using the Flask micro-framework and the Flask-RESTful extension.

Flask is a web framework for Python, categorised as a ‘micro-framework’ as it has no dependencies to external libraries. Flask-RESTful is an extension to Flask which provides support for building REST APIs.

I chose Python with Flask over C# with ASP.NET for building my service application because of Flask’s ease-of-use and minimalism. I believed that this task could be completed in Flask with far fewer lines of code than in ASP.NET, and debugging with Flask was sufficient enough for my needs.

I selected a public cat fact web API as the external web service for my application. The API handled GET requests to the service and required no authentication to access.

The WordNet API required an ID and key to access, both of which were stored within the service application code. The API handled GET requests to the service, retrieving definitions of the provided search terms.

The location service API required a token to access, which was stored within the service application code. The API handled GET requests to the service, retrieving location information based on either the user’s IP address or a provided IP address.

# **Discussion of Web Services:**

A web service is a piece of software hosted by a server on a network. A client can search the network for the server and make a request to invoke the web service. When a web service is invoked, it provides some functionality to the client that invoked it. These requests are called through remote procedure calls.

During a remote procedure call, data is transferred between the client and the server. This data is commonly in XML format, but other formats such as JSON can be transferred instead, depending on the type of web service communication used.

The two main and most common types of web service communication are based on SOAP and REST. Both SOAP and REST allow for the creation of APIs, which facilitate client-server communication through internet protocols.

SOAP is a standardised protocol, developed and maintained by W3C. Conversely, REST is not a protocol, rather an architectural style. REST has loose and flexible guidelines whilst SOAP has strict rules but more advanced features such as enhanced security. SOAP is also more complex and can require more resources, leading to slower load times, whilst RESTful services often have better performance.

SOAP has been the industry standard for many years, whilst REST was created much more recently to address some of the problems with SOAP. Therefore, SOAP’s rigidity is lost in REST, with REST supporting many different messaging formats, as opposed to SOAP’s XML exclusivity.

REST’s better performance and interoperability have resulted in its enormous popularity, allowing it to rival a much longer-standing and widespread competitor in SOAP.