s UNIVERSITY OF LEEDS	School of Computing University of Leeds **Coursework 2 - Report**	**Module Code** **COMP3211**

Web Services Composition
Submission Deadline Date: 17/11/2021

Student Name		**Username**	**ID**
1	Mark Muinde	sc19msmm	201374599
2	Ishmam Ahmed	sc19ia	201377974

Composition of Originality (10 marks)
The integrated client, external web service and first internal web service are in one folder, and the second web service in the other.

The integrated client takes user input (a city name) via a webpage. Web Service 1 (WS1) uses OpenWeatherMap RESTful API to return the current weather conditions of that city which are extracted and saved in an array and passed it to Web Service 2 (WS2). WS2 uses the weather conditions from the array to give out advice to prepare for the weather conditions and then appends the advice and integer flags that represent the same advice onto the array.

The integrated client then calls Web Service 3 from the second folder, '*apptwo'*, that takes as input the appended array and uses the integer flags from the array to perform a google search using python's googlesearch package, appends these links to the array and returns to the client the complete array. The client then passes these variables into a html page that displays all this information to the user at once.

Provide details in the table below

Web Service	**Own or External**	**Input**	**Output**	**Output Parsing / Extract something of interest**
1	**External**	**City name**	**Weather conditions array**	**Weather conditions**
2	**Own**	**Weather conditions array**	**Array appended with advice and integer flags that represent the same advice**	**Integer flags that represent the advice**
3	**Own**	**Integer flags representing the advice**	**Array appended with links for further advice**	**Links for further advice**

1st Web Service (20 marks)

Name of student in charge: *MARK MUINDE*

1st Web service	*Fill in this table*
Name of service	*advice.py*
SOAP-based or RESTful	*RESTful*
Brief description	*use the weather conditions gotten from the external WS to give out advice to prepare for them and then appends the advice and integer flags that represent the same advice onto an array that's passed onto the client*

Server design

The server takes an array from the external WS and parses the data to get the weather conditions from the array to give out advice to prepare for the weather conditions and then appends the advice and integer flags that represent the same advice onto the array.

Server implementation

Calls the primary function from the external WS as input and returns the appended array as output to the client.

Explain how the service is invoked. You may include relevant snippet of source code

Uses the weather conditions from an array to give out advice to prepare for the weather conditions and then appends the advice and integer flags that represent the same advice onto the array.

For example, If the user put in London the external web service would have returned London's weather conditions and saved them onto the array which would have been used by *advice.py* to construct some advice and create integer flags that represent the same and append the advice and flags onto the array.

On the next page are snippets of the code.

Include evidence of its execution through a client, e.g. screen shot

Snippets of code:

```
#import necessary packages
from appone import appone
from .views import *
import linecache
import pandas as pd

#advice to survive city's weather

def advices():

    #read the array from the external API
    dataArray = weather()


    #read the variables from the array
    temperatureAdvice = float(dataArray[1])
    humidity = int(dataArray[4])
    windSpeed = float(dataArray[5])


    #new variables for second web service
    #placeholder for weather advice string variable
    advice = ""

    #placeholder for temperature integer variable
    tempID = 0

    #placeholder for humidity integer variable
    humidityID = 0

    #give advice based off of weather data; namely temperature, wind and humidity
    print("\n\n----------------------------------------------------------------------------------------------------------------------------------------------------\n\n")
    print("All advice is based off of scales from Thinkmetric(temperature), the Beaufort Wind Scale(wind speed) and AirThings(humidity) ")
    #temperature advice
    if temperatureAdvice >= 15.00:
        advice += "Your destination will have high to very high temperatures. Put on SPF 40+ sunscreen, dress lightly, go see the city and, where possible, go to the beach.\n"
        tempID = 1
    elif 10.00 <= temperatureAdvice < 15.00:
        advice += "Your destination will have average to high temperatures. Put on SPF 30+ sunscreen, dress normally (but do carry a light jacket just incase) and go see the city.\n"
        tempID = 2
    elif 5.00 <= temperatureAdvice < 10.00:
        advice += "Your destination will have cold to average temperatures. You should have a jacket and some warm clothes. You can go touring if you enjoy the cold but carry the warm clothes with you.\n"
        tempID = 3
    elif 0.00 <= temperatureAdvice < 5.00:
        advice += "Your destination will have cold to very cold temperatures. We recommend that you stay indoors, cover up, have a warm meal and enjoy a lazy day.\n"
        tempID = 4
    elif temperatureAdvice < 0.00:
        advice += "Your destination will have below freezing temperatures. Winter is coming! We recommend you stay indoors. Nevertheless, have some winter clothes handy if you must leave your accomodation.\n"
        tempID = 4
    #wind advice
    if windSpeed >= 27.50:
        advice += "There will also be storms and possibly hurricanes. Do not leave your accomodation until it is safe to do so.\n"
    elif 16.50 <= windSpeed <27.50:
        advice += "There will also be gale winds. Bring a coat, preferrably a windbreaker if you have or can buy one.\n"
    elif 8.00 <= windSpeed <16.50:
        advice += "There will also be breeze winds. Not much to be concerned about, but bring a coat just incase.\n"
    elif 0.00 <= windSpeed <8.00:
        advice += "There will also be gentle winds-a little bit windy, but nothing alarming.\n"

    #humidity advice
    if humidity >= 50:
        advice += "Your destination also has high humidity. Buy a portable A/C and/or a dehumidifier and enjoy some cold drinks.\n"
        humidityID = 1
    elif 30 <= humidity < 50:
        advice += "Your destination also has moderate humidity. Buy a portable A/C or a dehumidifier if you find this too uncomfortable for you and enjoy some cold drinks.\n"
        humidityID = 2
    elif 30 <= humidity < 50:
        advice += "Your destination also has low humidity. Buy a humidifier if you find this too uncomfortable for you.\n"
        humidityID = 3


    #append the array for the second internal web service
    dataArray.append(tempID)
    dataArray.append(humidityID)
    dataArray.append(advice)

    #print advice
    print(advice)
    print("\n\n----------------------------------------------------------------------------------------------------------------------------------------------------\n\n")

    #return the array
    return dataArray
```

Measure the service invocation time. You are expected to run the experiments n times (e.g. n = 5). A statistical analysis (average, standard deviation) is expected.

Run No.	Service Invocation time
1	0.0953s
2	0.0966s
3	0.0091s
4	0.1443s
5	0.0990s
Average	0.0889s
Standard Deviation	0.0439

Explain how you have obtained these measurements

I used the time module in python and a calculator for the analysis.

2nd Web Service (20 marks)

Name of student in charge: ISHMAM AHMED

2nd Web service	*Fill in this table*
Name of service	*weather.py*
SOAP-based or RESTful	*RESTful*
Brief description	*use integer flags from the array to perform a google search using python's googlesearch package and append to the array the first 15 links that are gotten from the search then return it to the client.*

Server design
Takes as input the appended array and uses the integer flags from the array to perform a google search using python's googlesearch package, appends these links to the array and returns to the client the complete array.

Server implementation
The primary function is called from the client, takes an array as input, return as output an appended array with hyperlinks.

Explain how the service is invoked. You may include relevant snippet of source code

Using integer flags from the array created by the external web service and appended by *advice.py* (the first internal web service), *weatherarray.py* performs a google search using python's googlesearch package and returns the first 15 links that are gotten from the search in an appended array.

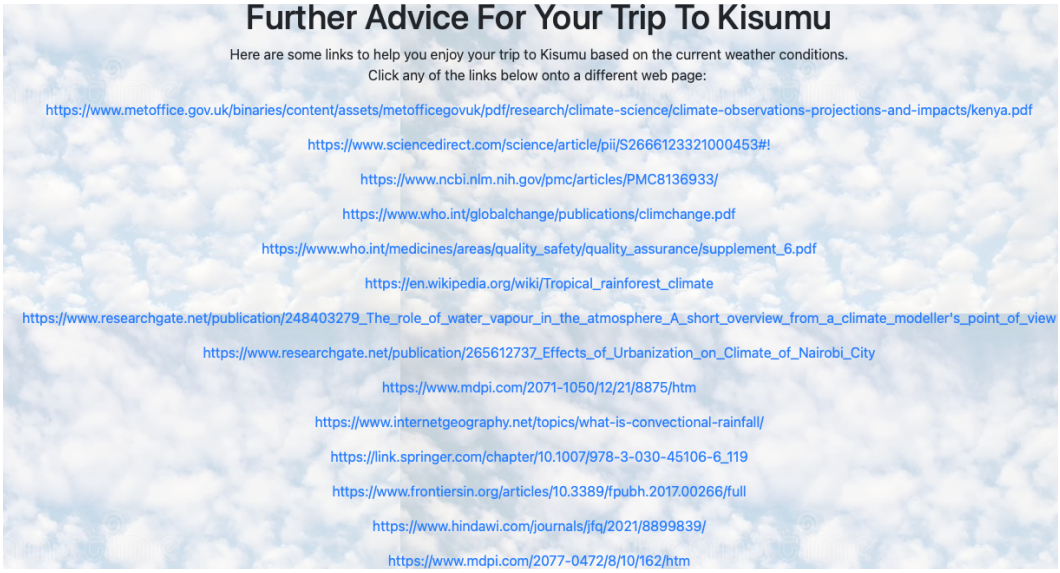
For example, If the user put in London the external web service would have returned London's weather conditions and saved them into the array which would have been used by *advice.py* to construct some advice and create integer flags that represent the same and append the advice and flags onto the array.

These integer flags will then be used to perform a google search as to what to do in London when under those weather conditions.

A snippet of the code is available of the next page:

```python
#second web service
#get array from client as argument and perform google search
def secondWebService(array):

    #read the variables from the array
    cityname = array[0]
    temperatureCurrent = float(array[1])
    temperatureMaximum = array[2]
    temperatureMinimum = array[3]
    humidity = int(array[4])
    windSpeed = float(array[5])
    windDirection = array[6]
    clouds = array[7]
    temperatureAlertLevel = array[8]
    humidityAlertLevel = array [9]
    advice = array [10]

    #save into new array for second web service
    weatherArray = [cityname, temperatureCurrent, temperatureMaximum, temperatureMinimum, humidity,
                    windSpeed, windDirection, clouds,
                    temperatureAlertLevel, humidityAlertLevel, advice]


    #parse array for specific values
    cityname = weatherArray [0]
    temperatureAlertLevel = int(weatherArray [8] )
    humidityAlertLevel = int(weatherArray[9])

    #placeholder for search query string variable
    query = ""

    #use the integer flags from the client to develop a query
    #that will perform a google search based on what they represent
    if temperatureAlertLevel == 1 and humidityAlertLevel == 1:
        query = ("https://www.google.com?q=Activities-in-" +
                 str(cityname) + "-when-very-hot-and-humidity-is-high")
    elif temperatureAlertLevel == 1 and humidityAlertLevel == 2:
        query = ("https://www.google.com?q=Activities-in-" +
                 str(cityname) + "-when-very-hot-and-humid")
    elif temperatureAlertLevel == 1 and humidityAlertLevel == 3:
        query = ("https://www.google.com?q=Activities-in-" +
                 str(cityname) + "-when-very-hot-and-humidity-is-low")
    elif temperatureAlertLevel == 2 and humidityAlertLevel == 1:
        query = ("https://www.google.com?q=Activities-in-" +
                 str(cityname) + "-when-hot-and-humidity-is-high")
```

Include evidence of its execution through a client, e.g. screen shot



Measure the service invocation time. You are expected to run the experiments n times (e.g. n = 5). A statistical analysis (average, standard deviation) is expected.

Run No.	Service Invocation time
1	3.9401s
2	3.8258s
3	6.7733s
4	6.6170s
5	3.6932s
Average	4.9699s
Standard Deviation	1.4112

Explain how you have obtained these measurements

I used the time module in python and a calculator for the analysis.

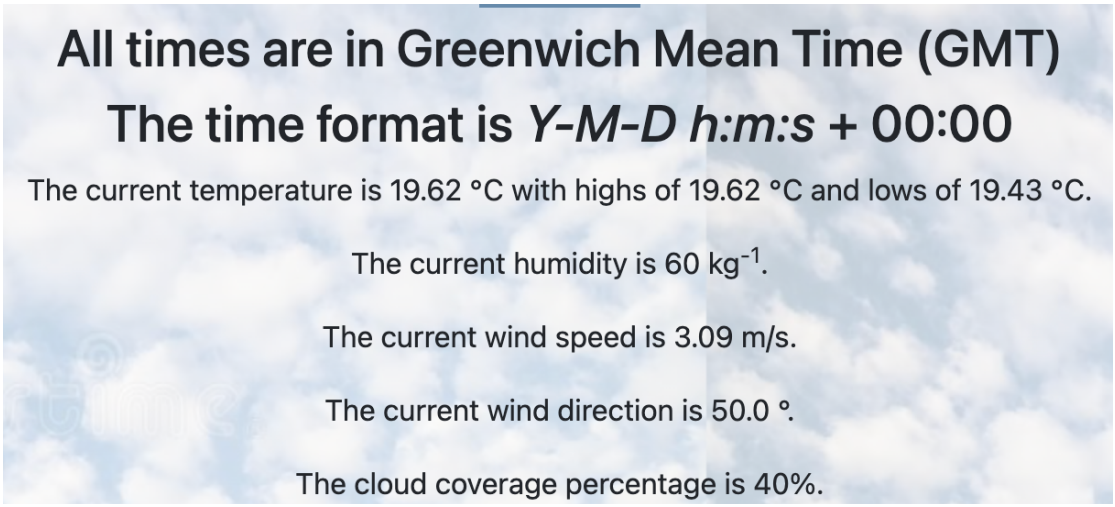
3rd Web Service - External Service (15 marks)

3rd Web service	*Fill in this table*
Name	*views.py*
SOAP-based or RESTful	*RESTful*
Name of publisher, e.g. Google, Twitter …	*Python, OpenWeatherMap*
Brief description	*Takes user input (a city name) and uses OpenWeatherMap RESTful API to return the current weather conditions of that city and then saves them onto a text file on the root folder*
URL	*https://openweathermap.org*

Explain how it is invoked. You may include relevant snippet of source code

It uses a webpage to take user input (a city name) and uses OpenWeatherMap RESTful API to return the current weather conditions of that city which *views.py* extracts and passes into an array for the first internal WS.

Include evidence of its execution through a client, e.g. screen shot:



Measure the service invocation time. You are expected to run the experiments n times (e.g. n = 5). A statistical analysis (average, standard deviation) is expected.

Run No.	**Service Invocation time**
1	0.0900s
2	0.0977s
3	0.0956s
4	0.0985s
5	0.1008s
Average	0.0965s
Standard Deviation	0.0036

Explain how you have obtained these measurements
I used the time module in python and a calculator for the analysis.

Web Services Integration (10 marks)

Provide details of the Web services integration
The integrated client, external web service and first internal web service are in one folder (*'appone'*), and the second web service in the other ('*apptwo'*).

The user inputs a city name via a web interface.

The external API (openweathermap.org) uses an API key for a free account to returns the weather conditions for that city to the client and passes an array representing the current weather conditions of that city to the first internal web service (WS1)

WS1 reads the array to get relevant weather conditions and returns relevant advice as to how to deal with the weather, and relevant integer flags that represent the level of alertness the client should have. It appends the advice and the integer flags in an array and passes it to the client.

The client calls in a function from *apptwo* that takes the appended array as input.

apptwo takes in the appended array and reads the integer flags, translates them into weather conditions and performs a google search as to what to do in the city when it has those weather conditions and returns the first 15 links that have been gotten. *apptwo* then appends these links to the array and send it back to the client.

The client then opens the array and returns to the user all this information on one comprehensive html page.

Mark developed the *advice.py* file to unpack an array of weather data, parse through it for important info, generate and append data to it and return to the client. Ishmam developed the *weatherarray.py* file to take an array as an argument and unpack it, parse through it for important info, generate and append data to it and return to the client. We tested the code outputs by hardcoding arrays at first and then printing the results on terminal.

We then developed the *views.py* file (and its corresponding template *city.html*) as the external WS from OpenWeatherMap API and Python OpenWeatherMap documentation that takes a city as input from the user and returns the weather conditions via an array which is passed to *advice.py*.

Integration was relatively easy as we agreed to have the order of inputting array elements done in a way that the two internal WSs would have no problem parsing them. We tested the outputs via printing on the terminal and once satisfied we developed the HTML & CSS templates for the client output so that a user has transparency when getting the desired output.

Web User Interface (10 marks) – if attempted

Provide details of your Web-based application (Servlets/JSP/Other Frameworks)
The application uses the Flask framework.
The user input is taken via a FlaskForm.
The web services will run on the client file and then return the data to the user via a html page.

This can be done via the published website:
https://markandishmam.pythonanywhere.com

Alternatively, it can be deployed via a localhost using the instructions on the next page:

Installation and usage

For localhost usage: Use the Flask framework At the root directory is a file-*requirements.txt* that has all the dependencies.

1. Unzip the zip file code submission
2. Open a terminal and cd into the folder named 'flask'
3. Run this command:

```
. bin/activate
```

This should activate the virtual environment

4. With the virtual environmet running, cd back to the root directory and run this command:

```
pip install -r requirements.txt
```

This will unpack all the dependencies.

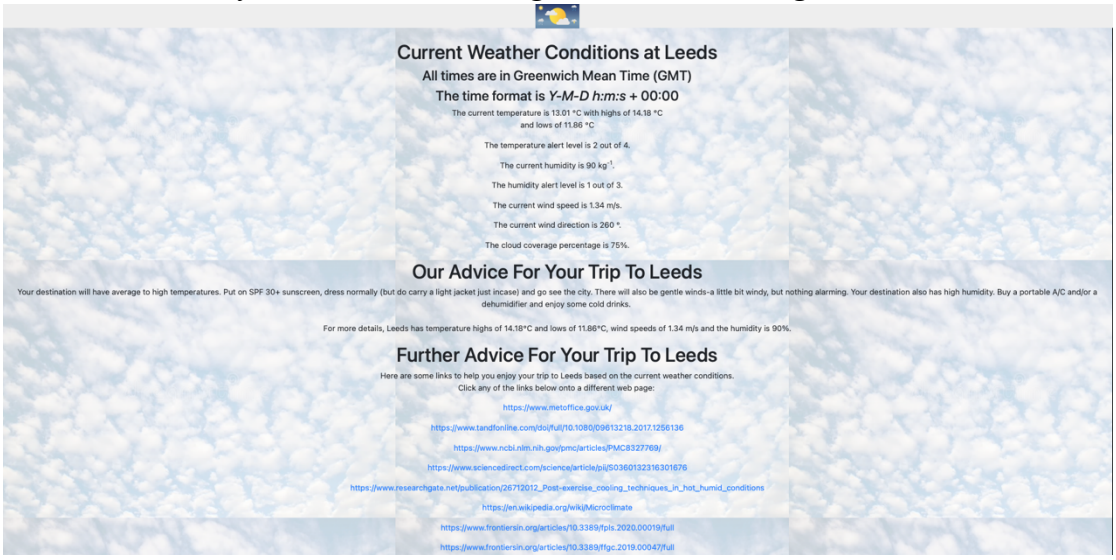
5. Run these commands next:

```
export FLASK_APP=run.py
export FLASK_ENV=development
flask run
```

6. Open a browser and go to the localhost url: http://127.0.0.1:5000/ You will receive output on both the webpage and your terminal once a city name is entered.

Successful Execution (10 marks)

Include evidence of the Web services integration execution, e.g. screen shot



Code Submission

The assumption is that your code has been submitted on Minerva.
If you have made your code available on Git instead, please provide details here:

Video Submission

The assumption is that your video demonstrating your integrated client has been submitted on Minerva.
If you have made your video available elsewhere, e.g. on YouTube, please provide details here: