Queensland Fire and Emergency Services

University of Sunshine Coast

ICT 705

Data and System Integration

TASK 2

Semester 1, 2020

Submitted by: Gagandeep Kaur

Student number: 1121869

Executive summary

In this report SOA architecture is discussed for Queensland Fire and Emergency Services. They realized they need more interactive website to display fire station location in various regions. SOA architecture allowed to use web services such as RESTful web services which are developed in programming and google map public API.

HTML, JavaScript and ajax are used to design the web page, ajax provide dynamic updating of data without needing to refresh whole page. Json format is used to communicate data from server to be displayed on website. Python is used to create web services that connects with csv file to fetch data and connects to local host.

File has definition for various concepts such as SOA, data merging, mashup etc. this project has used various API and webservices to mashup.

Introduction

Queensland fire and emergency services are improving their system by providing more interactive website to provide information on location of fire station in various regions in Brisbane. To provide current information on fire stations mashup is used to utilize google api and web services. Use of SOA as allowed to reuse the module.

Service Oriented Architecture (SOA)

By the definition of open group organization, Service Oriented Architecture is architecture style that supports service-orientation. Service-orientation is way of providing service-oriented solutions and service-based development.

According to IBM, SOA helps in creating enterprise IT architecture that exploits principles of service-orientation to make tighter integration between business and information systems that support the business.

It results in providing better and flexible business process, develops relationship between enterprise architecture and business.

SOA defines methods to develop software solution which are reusable through service interface, these services has communication standard that can be used in new application without needing deep integration each time. Hence, code is reusable, and services are loosely coupled that means it does not require deeper knowledge of integration. Services are utilized through service calls over the internet using standard network protocols lie SOAP/HTTP or JSON/HTTP. Services are designed so that its can be found quickly and used in other applications. (Portier Beryand)

Features of SOA

- Modular: services should be abstract, reusable. It should be loosely coupled which minimizes interaction and dependence on other services, a self-contained unit.
- Use of Standard protocol: SOAP, HTTP, XML
- Evolvable so that it can be integrated into various process easily.
- Scalable to bee able to work with large systems and across organizations.

Benefits of SOA

- Reduce cost and faster development of new systems: reduces integration cost, flexible modules, uses existing assets and long term interoperability.
- Agility- built for change: allows outsourcing, incremental implementation approach, application evolve over time.
- Platform independent
- Available
- Reusable
- Scalable
- Easy to maintain
- Help increase employee productivity as they have to build on existing sills and can combine duplicate functions.
- Built partnerships: standard based, relationships are expressed via service interactions. Integration is not dependent on what is technically possible rather than what is needed.

Disadvantage of SOA

- High initial investment.
- Complex service management, as services interact via messages to perform tasks when number of messages increases it is difficult to keep track.
- High overhead: validation is required each time of inputs when services interact this decreases performance because load and response time increases.

When it is good idea to implement SOA: if an organisation is large, internet is required to provide services, utilise abstraction, reuse business process(use cases of same process)

When it is not good idea to use SOA: when real time performance is required, heterogeneous IT process, tight coupling and thins are rigid i.e. not changing.

QFED should be benefited using SOA because they want to provide services through internet, they have repeated use cases, all the business process are not dependent on other and they not require much of real time performance. (Arpit Sud)

Data Merging and Cleaning

Data cleaning is task of removing inconsistent data from the database. Data that is present today in real world is very unorganized, it contains a lot of missing values, incomplete data and undefined data that has to be cleaned before file is used further. A cleaning process results in complete records that means missing values were handled, data should be accurate, valid and consistent. results should be as accurate as possible

Data merging is process of combing various datasets which have some common features. When two or more datasets are merged one simple solution is to concatenate them and remove duplicates. (Bitton and DeWitt,1983). However, heterogeneous data that is if schema is not same or same real-world entity are represented different then we have to carefully change data to make them consistent with each other. (Hernández and Stolfo,1998)

Challenges faced while merging stations.csv and Queensland regions was that RegionID column names were different. Satation.csv and location.xml had station name common but had to modify it a little.

RESTful Web Service

REST is Representational State Transfer. Architecture is based on Virtual hierarchical folder structure, it depends on stateless, client-server, cacheable communication protocol i.e. HTTP protocol. It is used to design network applications. REST relies on URL rater than XML to make request hence, it is easier to use and flexible. Messages exchanged between server and client are prepared in HTTP standard response and request. RESTful services provide simple lightweight interactions. It provide interoperability as it provide methods in HTTP.

In my demo data that is showed in the website is obtained from URL request to the localhost server which returns data in Json format, this data is utilized by

XMLHttpRequest to receive and we use JSON.parse to read the data.

Data is coming from CSV file, if that file is modified, data received will also be modified.

Mashups

A mashup is lightweight web application that combines information from more than one existing source to deliver new function. It possible via use of web service or public API, it tries to make more interactive websites. It also uses HTTP to read and write data.

Ajax is asynchronous JavaScript and XML; it is used for service calls. It doesn't require browser to refers or load another page. Ajax request is an HTTP request. We can build mashups with Ajax and RESTful services.

In my demo code I used goggle API and RESTful webservice to show markers in google maps. ajax is used to load markers without the need of refreshing the page.

Demo Running Instructions

Steps to run the demo:

- Extract the zip file by right click on it and then choose "extract here" option, this will unzip your file
- Install python from https://www.python.org/downloads/windows/, keep pressing next and wait for installation. Leave the default options as it is.
- Install bottle and petl libraries:
    - Open 'Command Prompt', type the path to Python Folder in C drive.
    - Use 'cd' command.
    - Install bottle by typing 'pip install bottle' and then 'pip install petl' to install petl.
- Move the files folder to the python location (where python is installed) in C Drive.
- After installation, open the 'Python IDEL', click on 'files 'then 'open' a window will show up from where you can select files to open ten chose 'Data_integration.py' file
- Click on 'Run' then click on 'Run Module'
- Open the 'Data_service.py' file the choose 'Run' then choose 'Run Module', this will activate local host 8080 to perform web services call from websites. It connects website with python that fetch data from 'Fire_Station_Solution.csv' file.
- Open the folder and click in on 'Stations_map.html' this will open up the browser and website will appear.
- Select the region in drop down box

- It will return markers on the map of various fire station location in the region.

Conclusion

In conclusion using RESTful and mashup technologies make system more lightweight and it allow reusability of code. It had made the website dynamic i.e. data is loaded without refreshing the page and its always current.

References

- The Open Group. SOA Reference Architecture Technical Standard. https://www.opengroup.org/soa/source-book/soa_refarch/index.htm
- Arpit Sud. Service Oriented Architecture. http://www.cs.colorado.edu/~kena/classes/5828/s10/presentations/soa.pdf

- Portier Beryand. Service, architecture, governance, and business terms https://www.ibm.com/developerworks/library/ws-soa-term1/
- Hernández, Mauricio & Stolfo, Salvatore. (1998). Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem https://www.researchgate.net/publication/220451890_Real-world_Data_is_Dirty_Data_Cleansing_and_The_MergePurge_Problem
- Bitton, D. and DeWitt, D. J. Duplicate Record Elimination in Large Data Files. ACM Transactions on DatabaseSystems, 8(2):255–265, June 1983
- Oracal SOA Tutorial https://mindmajix.com/oracle-soa-tutorial

Appendix

Appendix 1 – Data_integration.py

```
import petl as etl

import re
```

```
t2=etl.fromcsv("Stations.csv")


t2
=etl.capture(t2,'Address','(\d{4}$)',['Postcode'],include_orig
inal=True)


def substitute(table, field, pattern, repl, count=0,flags=0):
       prog = re.compile(pattern, flags)
       conv = lambda v: prog.sub(repl, v, count=count)
       return etl.convert(table, field, conv)


t2=substitute(t2,"Address",'\d{4}\Z',"")


t2=etl.addfield(t2,"State","Qld")


t2=etl.addfield(t2,'email', lambda rec:rec['Station
Name'].lower())
t2=substitute(t2,'email','[\s]','')
t2=substitute(t2,'email','firestation','')
t2=etl.addfield(t2,'E-mail', lambda rec:
'enquire@{}.qfes.gov.au'.format(rec['email']))
t2=etl.cutout(t2,"email")


t2 = etl.rename(t2, 'Phone', 'Phone Number')
t2 = etl.rename(t2, 'Fax', 'Fax Number')
t2 = etl.rename(t2, 'Address', 'Street Address')
t2 = etl.rename(t2, 'Stn Number', 'Station Number')
t2 = etl.rename(t2, 'Stn Type', 'Station Type')
t2=etl.cutout(t2,'Alternate Address')
t2 = etl.rename(t2, 'Region', 'Region Code')
print(t2)


t3=etl.fromcsv("Queensland_Regions.csv")
print(t3)


t4
```

```
=etl.fromxml("Station_Locations.xml","STATION",{"NAME":"NAME",
"LONG":"LONG","LAT":"LAT"})

t4 = etl.rename(t4, 'LAT', 'Latitude')

t4 = etl.rename(t4, 'LONG', 'Longitutde')

t4=etl.convert(t4,"NAME",'title')

t4=etl.addfield(t4,'Station Name', lambda rec: '{} Fire
Station'.format(rec['NAME']))

t4=etl.cutout(t4,'NAME')

print(t4)


mer_1=etl.join(t3,t2,key='Region Code')

print(mer_1)

mer_2=etl.join(mer_1,t4,key='Station Name')

mer_2=etl.cutout(mer_2,'Region Name')

mer_2=etl.cutout(mer_2,'Region Code')

mer_2=etl.movefield(mer_2,'Station Number',1)

mer_2 = etl.sort(mer_2, key=['RegionID','Station Number'])

print(mer_2)


etl.tocsv(mer_2,"Fire_Station_Locations.csv")
```

Appendix 2 – Data_Services.py

```
from bottle import route, run, request, response

import petl as etl

import json as js


# create service to respond to "getoffices" request

@route('/getregions')

def get_region():

    print("Received a request for getregion")


    #open farmers.csv

    t1 = etl.fromcsv("Queensland_Regions.csv")
    print(t1)

    #prepare response
    response.headers['Access-Control-Allow-Origin'] = '*'
    response.headers['Content-type'] = 'application/json'

    #return farmers data in JSON format
    return js.JSONEncoder().encode(list(etl.dicts(t1)))

@route('/getstations')
def get_station():
    regionid=request.query.regionid
    print("Receive request for region:"+regionid)
    t2 = etl.fromcsv("Fire_Station_Locations (Solution).csv")
    if regionid == '0':
       result_two=etl.cut(t2,"Station Number","Station
Name","Street Address","State","Postcode","Phone Number","E-
Mail","Latitude","Longitude")
       print(result_two)
    else:

result_one=etl.select(t2,"{RegionID}=='"+str(regionid)+"'")
       result_two=etl.cut(result_one,"Station Number","Station
Name","Street Address","State","Postcode","Phone Number","E-
Mail","Latitude","Longitude")
       print(result_two)

    #prepare response
    response.headers['Access-Control-Allow-Origin'] = '*'
    response.headers['Content-type'] = 'application/json'

    #return farmers data in JSON format
    return
js.JSONEncoder().encode(list(etl.dicts(result_two)))
run(host='localhost', port=8080, debug=True)
```

Appendix 3 – Stations_map.html

```
<html>
  <head>
    <meta http-equiv="Access-Control-Allow-Origin"
content="*"/>
    <title>Queensland Fire Stations</title>
     <style>
     html, body {
        height: 100%;
        margin: 20;
        padding: 0;
      }
      #map {
       margin:20;
      height: 100%;
      }
    </style>
  </head>

  <body>
  <div>
    <select id="region-dropdown" >
     </select>
     <button id="submit" onclick="url_1
=getselectvalue(),serverRequest(url_1, processResponse)">
     Display Stations</button>
     </div>

     <div id="map"></div>
<script type="text/javascript">
//dynamically loading regions in dropdown box
let dropdown = document.getElementById('region-
dropdown');
dropdown.length = 0;

let defaultOption = document.createElement('option');
defaultOption.text = 'All Regions';
defaultOption.value= 0;

dropdown.add(defaultOption);
dropdown.selectedIndex = 0;

const url = 'http://localhost:8081/getregions';

const request = new XMLHttpRequest();
request.open('GET', url, true);

request.onload = function() {
  if (request.status === 200) {
    const data = JSON.parse(request.responseText);
    let option;
    for (let i = 0; i < data.length; i++) {
      option = document.createElement('option');
      option.text = data[i]["Region Name"];
      option.value = data[i].RegionID;
      dropdown.add(option);
    }
   } else {
    // Reached the server, but it returned an error
  }
}
request.onerror = function() {
  console.error('An error occurred fetching the JSON from
' + url);
```

```
};

request.send();

//getin the regiion value to be passed in server request
function getselectvalue()
{
var a=document.getElementById("region-dropdown").value;
return 'http://localhost:8081/getstations?regionid='+a;
}

function serverRequest(url, returnProcess) {
        var xhttp;
        if (window.XMLHttpRequest) {
        // code for modern browsers
          xhttp = new XMLHttpRequest();
        }
        else {
        // code for IE6, IE5
          xhttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
        xhttp.onreadystatechange = function() {
          if (this.readyState == 4 && this.status == 200)
{
            returnProcess(this);
          }
        };
        xhttp.open("GET", url, true);
        xhttp.send();
        };

      function processResponse(xhttp) {
```

```
      const jdata = JSON.parse(xhttp.responseText);
       for (var i=0;i< jdata.length;i++){
       var marker= new google.maps.Marker({
       position: {lat: jdata[i].lat, lng: jdatat[i].lng},
       title: jdata[i].placeName,
       map:map
       });
       }
       map.setCenter(marker.getPosition());
       };
          var map;
          function initMap() {
            map = new
google.maps.Map(document.getElementById('map'), {
               center: {lat: -27.470125, lng:
     153.021072},
               zoom: 12
            });
          }
</script>
    <script src="https://maps.googleapis.com/maps/api/js?
key=AIzaSyCO3hE4bE7WV7mGuXL4kn9caoWI44tD8Ic&callback=init
Map"
        async defer></script>
     </body>
```