Technical Specification Document

**Introduction:**

This is the technical specification document for the mule application which basically helps in exposing the API which provides the details of countries and their associated cities as well as weather condition of any selected city. Moreover, this mule application is also implemented with specific error handling scenarios to cater the request of the user/consumer in more efficient way. This document provides the detailed information about the list of inbound and outbound endpoints, type of input payload/request to consume operation specific services.

**Problem Description:**

There is a website which provides the information about weather condition based on the input provided i.e., country and city. This website also exposes its service to provide same information through SOAP webservice. But the response payload of the service is not clear and include junk characters.

**How the problem is affecting users:**

The systems who are developed in such a way that they can only trigger requests in json format, those systems are unable to fetch the weather details from the above-mentioned service exposed by the website developers. Also, those systems who can consume SOAP webservice, must perform extra work to trim the response to remove all the unwanted characters from the XML response.

**Problem Solution:**

This problem can be solved by implementing a centralized communication service bus which is basically based on SOA and implements the API led connectivity. This service bus will serve all types of requests from all the consumers without depending on the format of their requests. This middleware system will consume the SOAP service which is exposed by the website for the weather information and process the response to remove all junk characters and convert it to readable dataset.

**Solution Design:**

To address the problem mentioned above we will be using Mule ESB which will be communicating with service provider (Weather Website) and consumers in a totally decoupled way which is known as “Logical Coupling”. In this design principle, the consumer does not know about the service provider and service provider does not know about the consumer.

To expose the results in the format of JSON to the user/consumer, we will develop the mule flow and publish the API to serve the request in json format using RESTful webservice as requested in the deliverables of the challenge. To implement the API management in the mule application, we have used RAML to design the API for serving the requests and for also propagating the error if the input format is invalid or incorrect.

**Technical Specifications:**

The SOAP webservice URL provided to access the weather information generally do not work. So, for the workaround we have used docker desktop to run docker image, which runs Node application and server in backend exposes the soap service providing same set of output as the mail URL.

|  |  |
| --- | --- |
| **Tools** | **Description** |
| Docker Desktop | To run the docker file provided in the initial codebase. |
| Node Js | To expose the SOAP webservice and handle the request from the local system. |
| SOAP UI | To trigger and check the request and response of the SOAP service. |
| Mule 4.0 | Used for the implementation of mule flows and consuming the service. |
| Mule Anypoint Studio | Used as IDE for the development of mule flows, RAML as well as Datatypes. |
| Postman | Tool to check the request and response from the implemented Rest web service exposed for the consumer. |
| Mule Runtime | To deploy the mule application locally to test the flow and error handling in the mule flows. |

**Endpoint Details:**

|  |  |  |
| --- | --- | --- |
| **Endpoints** | **Type** | **Endpoint Description** |
| <http://localhost:8080/GlobalWeather?WSDL>  (Accessed by running docker or by running node server) | SOAP URL | To access the city and weather information by passing the request parameters exposed by weather website. |
| <http://localhost:8081/api/globalweather/citiesByCountry>  ?country=Australia | GET | To access the Rest API implemented in mule to retrieve list of cities with input country name. |
| <http://localhost:8081/api/globalweather/weather>  ?country=India&city=Melbourne | GET | To access the Rest API implemented in mule to retrieve weather condition based on input country name and city name. |

**Flow Details:**

* First and foremost, we have used docker or node to access the SOAP webservice provided in the initial codebase.
* This soap web service has two operations, in which one is for retrieving the list of cities by providing country name in the input body and another one is for retrieving the weather condition based on input country name and city name.
* As per the requirement mentioned in the deliverables, later we have implemented a mule flow to consume that SOAP webservice with body which includes country names to access city as well as weather related information.
* To implement API led connectivity, we have added all this flow as system API which retrieves and process the data to pass it over to process API and later experience API.
* To implement API management as per the restful design, we have used RAML and Datatypes to define the base URL, input parameters and error handling.
* We have also used error handing in the mule flows by propagating the error during execution and halt the flow.
* Finally, we have implemented MUnit functions to implement the unit testing to validate the final output from the mule flows.