Technical Specification Documentation

for

Weather Rest API

|  |  |
| --- | --- |
| Date: | 17th November, 2019 |
| Version | 1.0.0 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Table of Contents

[Revision History](#_heading=h.3abk1vopay1g)

[Executive Summary](#_heading=h.ut1pt0wdouhq)

[Project Codes and Abbreviations](#_heading=h.3znysh7)

[Deployment Architecture](#_heading=h.fhcn8adlu6te)

[Backend - Weather SOAP Services](#_heading=h.suvtjh8nljx0)

[Executing the REST API](#_heading=h.44jhgbk1wk3q)

[Testing the API](#_heading=h.ulztb68627qa)

[Appendix](#_heading=h.2jxsxqh)

# 

# 

# 

# 

# 

# 

# 

# 

# 

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Authors** | **Revision Details** |
| 1.0 | 17.11.2019 | Firzhan Naqash | Initial Revision |

# Executive Summary

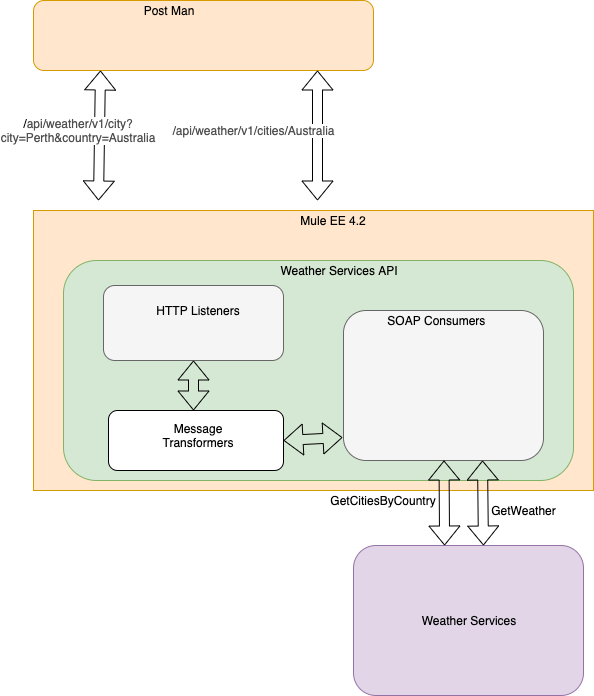
This document details the tasks carried out to implement REST interfaces for the Weather API which are exposed as SOAP Services. This documentation provides detailed information on the exposed REST services and the output obtained under different conditions. The following sections describe the detailed explanatory diagrams and information about these implementations.

### 

## Project Codes and Abbreviations

|  |  |
| --- | --- |
| **Code/Abbreviation** | **Description** |

# **Deployment Architecture**



*Figure 01*: Weather API Integration Architecture

The MULE EE 4.2 runtime serves the the API requests. The main intention of the task is to provide the required REST interface to communicate with a SOAP Based Weather API Services.

The API resources are exposed via HTTP Listeners. The HTTP Listeners consume the incoming API request over the HTTP port. The required information from the requests would be extracted from either URL parameters or query parameters. Eventually, the extracted information would be used to form the SOAP messages. These SOAP messages is used by the Web Consumer Connector to consume services from Weather SOAP Services. Transformation of messages from REST to SOAP and vice versa is done by Message Transformers. The whole operational workflow has been clearly depicted in the *figure 01*.

# 

# **Backend - Weather SOAP Services**

The backend is a docker container made of nodejs application. The results returned had some tricky scenarios which had to be handled specifically. Those scenarios are listed down.

**Scenario #1**

The returned XML messages are encoded. Hence, the messages are decoded before being processed.

**Scenario #2**

The Weather Response Message wraps the information within double CDATA groups.

<GetWeatherResponse xmlns="http://www.webserviceX.NET"><**![CDATA[<![CDATA[**&lt;NewDataSet&gt;

&lt;Location&gt;Melbourne&lt;/Location&gt;

&lt;Time&gt;11 AM&lt;/Time&gt;

&lt;Wind&gt;15 km per hour&lt;/Wind&gt;

&lt;Visibility&gt;10 km&lt;/Visibility&gt;

&lt;SkyConditions&gt;sunny&lt;/SkyConditions&gt;

&lt;Temperature&gt;18&lt;/Temperature&gt;

&lt;DewPoint&gt;2 C&lt;/DewPoint&gt;

&lt;RelativeHumidity&gt;35&lt;/RelativeHumidity&gt;

&lt;Status&gt;Normal&lt;/Status&gt;

&lt;/NewDataSet&gt;]]]]>><![CDATA[]]></GetWeatherResponse>

The above **CDATA** grouping is removed using DataWeave’s replace functionality in the **Set-Payload** operator.

# **Executing the REST API**

The integration configuration could be found on the github link [https://github.com/firzhan/weather-services](https://github.com/firzhan/weather-api).

Prerequisite :- JDK 8

**Executing the Weather SOAP Services**

* The Weather Service is instantiated by executing the [Dockerfile](https://github.com/firzhan/weather-api/blob/master/weatherExcerciseDockerFile/Dockerfile).
* First, we have to build the dockerfile provided for the backend services using the following command. **docker build -t deloitt/weather**
* Next we have to run the docker container on port 8080 using following command. **docker run -p 8080:8080 deloitt/weather:latest**

**Building the deployable JAR**

* First, we have to navigate into the [WeatherServices folder](https://github.com/firzhan/weather-api/tree/master/weather-services) which contains the mule source. At that point, I have to run the command **mvn clean install** to generate the JAR file.
* JAR file under the name **weather-services-1.0.0-mule-application.jar** could be found under the **target** folder of the Weather Services source directory.

**Executing the Mule EE Environment**

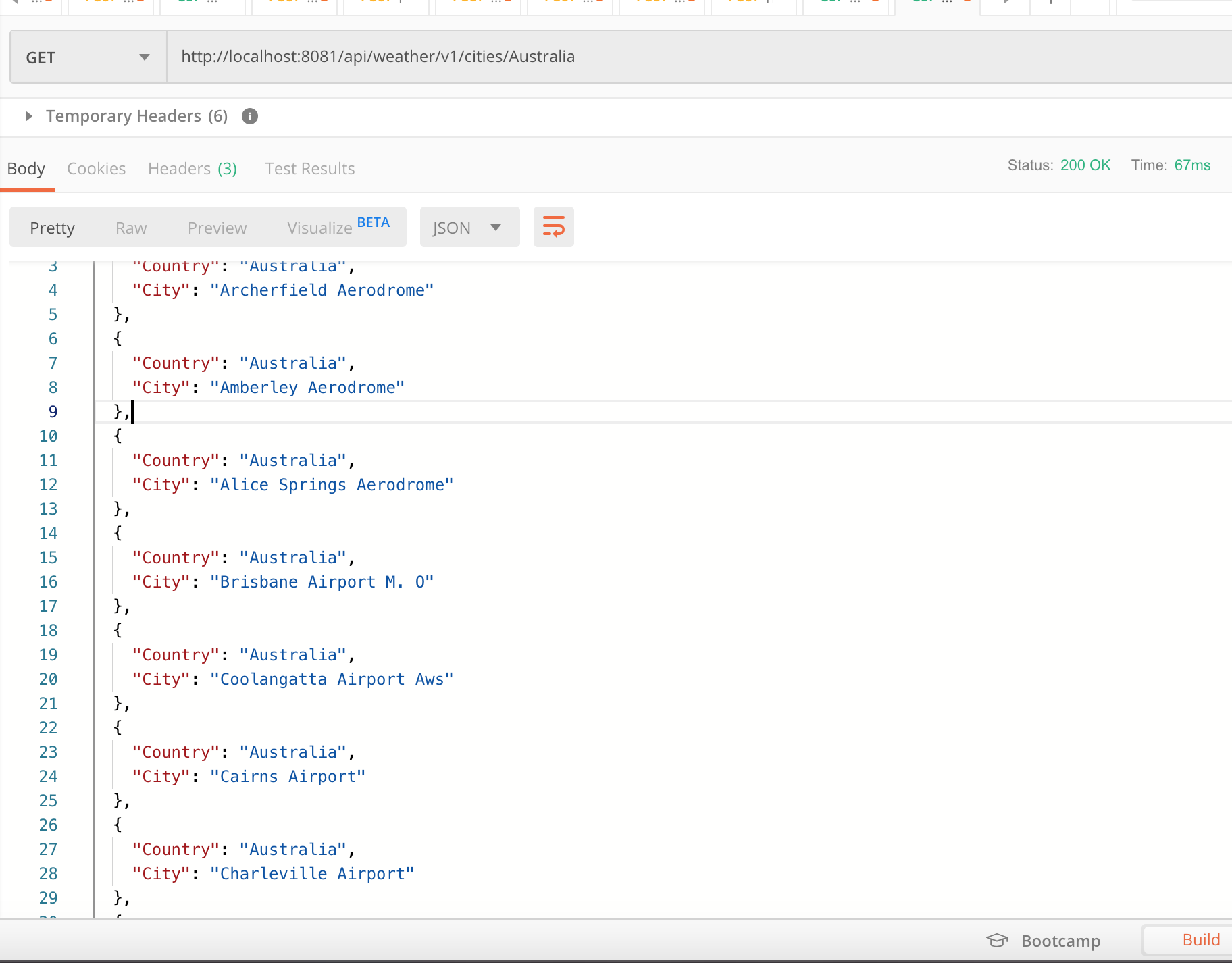
* The latest version of Mule EE 4.2.1 could be downloaded over the following [link](https://www.mulesoft.com/lp/dl/mule-esb-enterprise)**.**
* Unzip the archive and copy the JAR file into the **apps** folder inside the root directory of the Mule EE distribution.

# **Testing the API**

* Since the APIs are only fetching the records, operations with GET verb would suffice.
* Following APIs are defined to fetch the records.

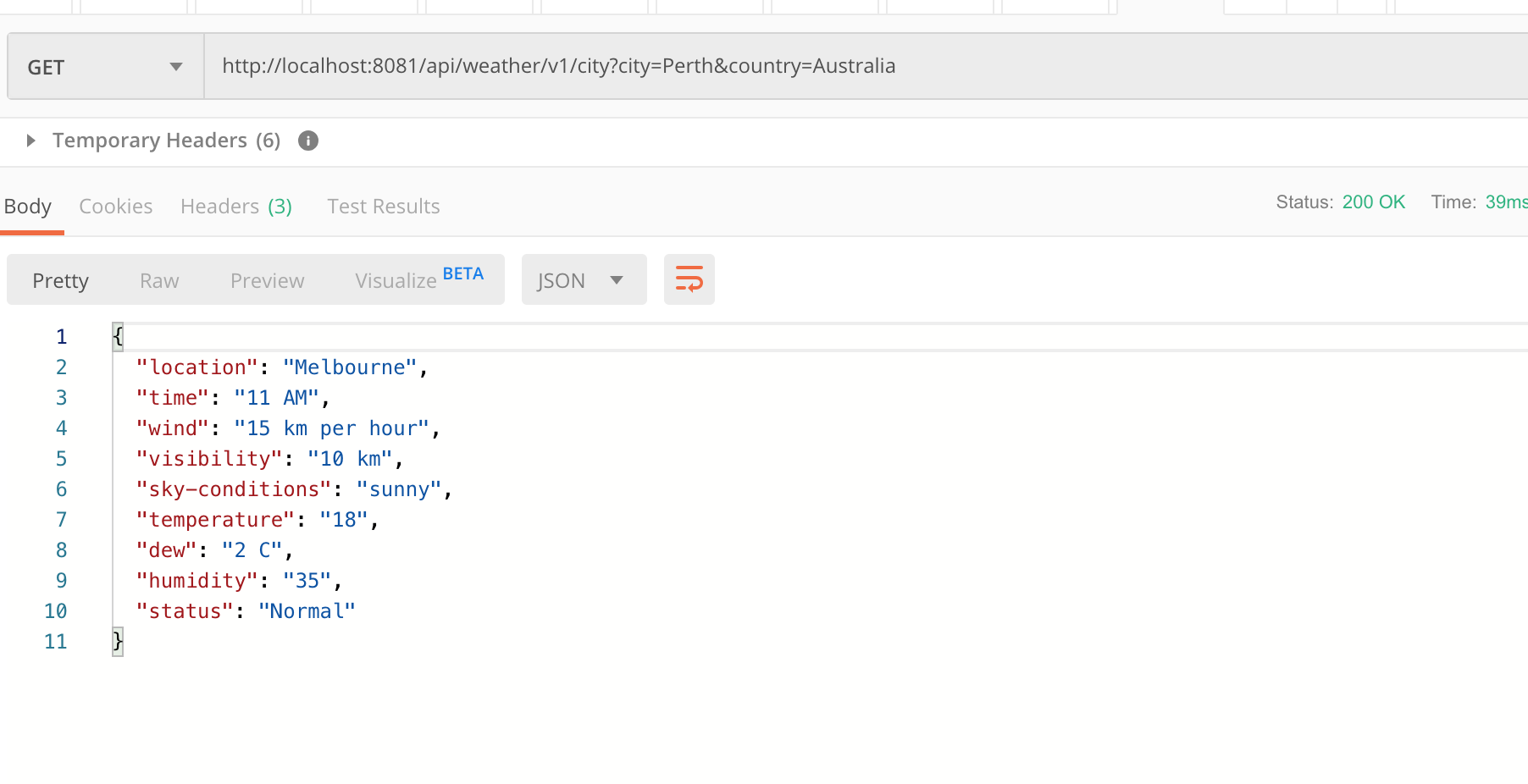
1. Fetching all the cities within a country

[**http://localhost:8081/api/weather/v1/cities/Australia**](http://localhost:8081/api/weather/v1/cities/Australia)

****

1. Fetching the status of weather in a city of a country

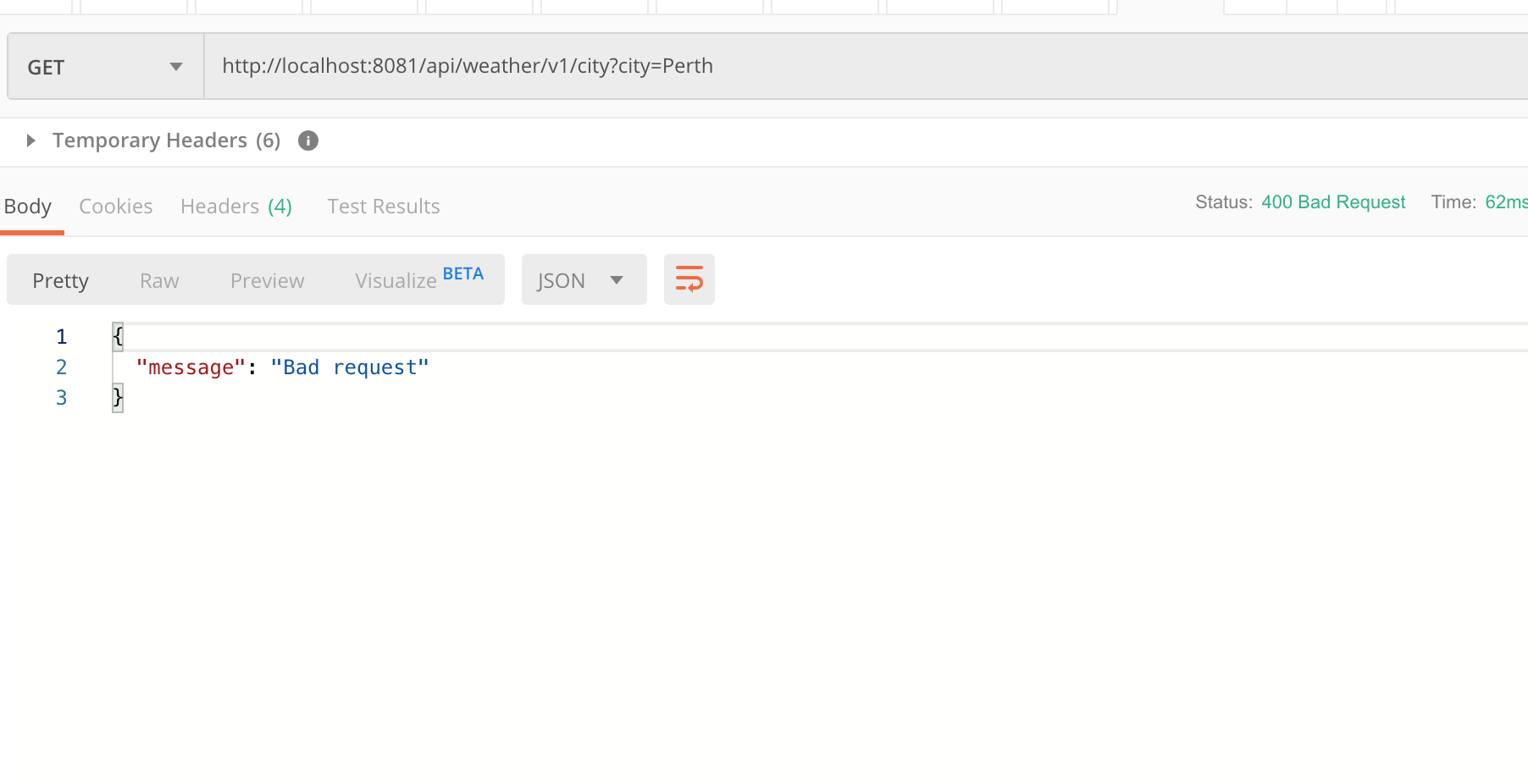
[**http://localhost:8081/api/weather/v1/city?city=Perth&country=Australia**](http://localhost:8081/api/weather/v1/cities/Australia)



**Handling Invalid API Invocations**

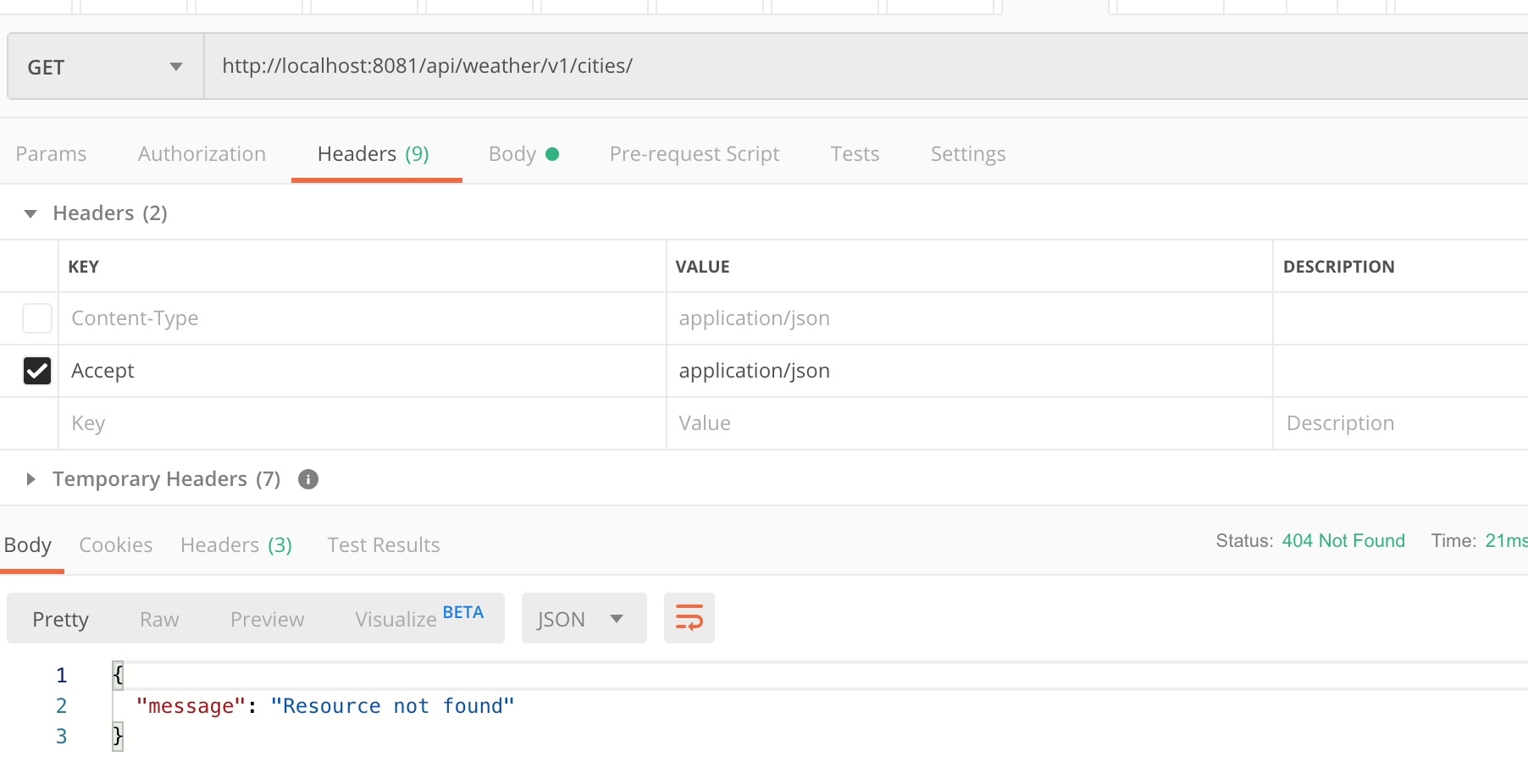
**Scenario #1**

While fetching the weather details, both the country and city names should be passed as query params over the URL. Missing out one of them or both would end up with **Bad Request** response.



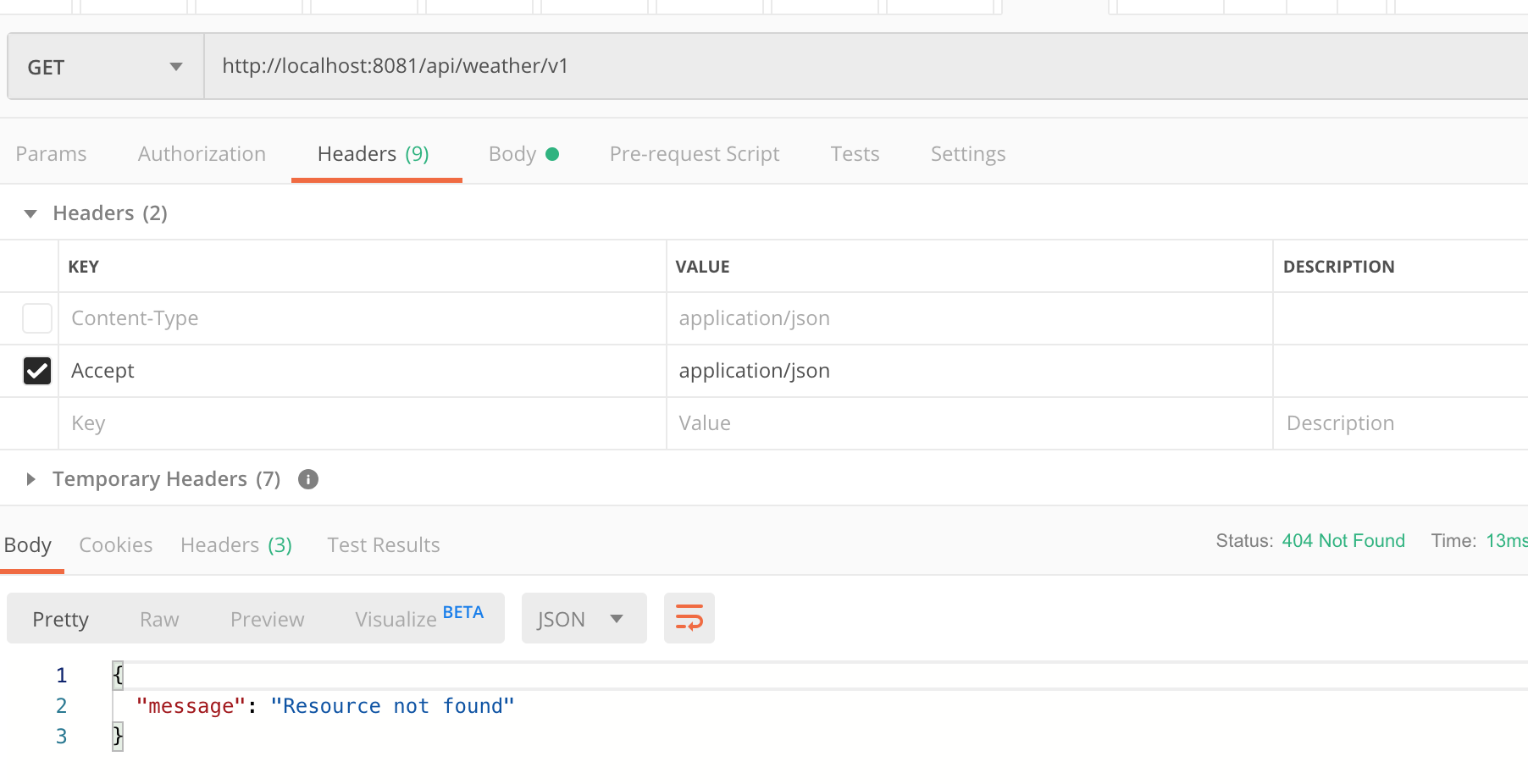
**Scenario #2**

Country name should be passed as URL param while requesting to list down all the cities of a country. If not, it would end up with Resource Not Found error.



**Scenario #3**

Any of the invalid URL invocation, would end up with Resource Not Found error.



# 

# Appendix

[1] <https://developer.mulesoft.com/tutorials/quick-start/developing-your-first-mule-application>

[2] <https://developer.mulesoft.com/tutorials/quick-start/designing-your-first-api>

[3] <https://docs.mulesoft.com/mule-runtime/4.2/dataweave>

[4] <https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9>

[5] <https://restfulapi.net/rest-architectural-constraints/>