Design Decision and Challenges

Table of Contents

[Design Decisions 3](#_Toc63114494)

[Challenges 5](#_Toc63114495)

# Design Decisions

* The development of this application is intended to demonstrate the development life cycle of RESTful APIs.  
  Development first began with RAML design in the Design Center on Anypoint Platform, where the behavior was first tested out with mock responses.

Graphical user interface, application

Description automatically generated

The RAML was then used to generate the project skeleton to work on.

When future changes arise, it shall follow the development lifecycle like below

Feeback/change request -> RAML change in Design Center -> Publish to Exchange for users to test on mock response -> Feeback -> Update design -> Implement changes -> Test -> Release -> Feeback/change request -> …

* The backend GlobalWeather API performs two functions, one for getting cities based on country, another for getting weather info based on country and city. As these operations are quite clear, data massage and integration from various sources are not needed. The Weather API application thus tries to resemble these functions with two APIs as well.
* On considering the parameter constraints, since the GlobalWeather web service returns mock responses, it is a bit unclear on how the web service validates the request parameters. As per the WSDL file, all parameters are optional, but this remains in doubt as we could not test on the actual web service endpoint.

It would not be efficient for the web service to return the information of all cities for all countries, as there are numerous cities in the world; Similarly, it is not sensible that the web service returns the weather information of all cities.

Based on the information in hand, it is assumed that the actual web service requires parameters to be passed. Our Weather API corresponds to this assumption and requires request parameters to be passed as well.

And even if the actual GlobalWeather web service turns out to be accepting "wildcard" requests, it makes sense for our Weather API to enforce filtering with parameters on our end for performance consideration.

* Since this application is expected to be used for demonstrating the general use of various Mule components, it was decided that fine-grained setup and control on the API not to be implemented. For example, the application had not been deployed to CloudHub and no policy had been implemented to ensure proprietary access.

# Challenges

* The mock responses provided has the target data wrapped in CDATA, which means they are not translated as XML when SOAP response is received (it is unclear if the same kind of response would be received if the actual GlobalWeather web service endpoint is called). As result the response payload body cannot be converted to XML directly.

This proved to be quite tricky and some effort was required to study on how to handle such scenarios. The approach adopted here was as follow

* Convert to string first, change CDATA symbols back to the original ones
* Read the updated string as XML
* Output as JSON

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

* Originally thought was to demonstrate the end-to-end flow from compile (which should execute the Munit test cases) to deployment on runtime (just like what you would do with a development pipeline) on CloudHub.

However, it was discovered that Munit test cases could be run as part maven build, where error was thrown when trying to compile with the Munit test phase.

Text

Description automatically generated

Text

Description automatically generated

It was found that it was a common issue where it requires enterprise license credentials to be present in the Maven settings file (Refer to <https://help.mulesoft.com/s/question/0D52T00004mXXkv/unable-to-run-munits-after-upgrading-mule-runtime-to-420-from-413-details-of-the-error-is-prov>).

As this issue should only be limited to casual users only, and that Munit tests can be run separately from within Anypoint Studio as well, it was decided to skip the Munit test phase in the build.

In addition, since a node.js backend is used to simulate the GlobalWeather web service response, it has no use deploying the Weather API application to CloudHub alone since it won’t be able to connect to the web service anyway.

Considerations have been made on whether or not to deploy the provided node.js application somewhere accessible with a URL, but based on the expected purpose of this project, it was decided to keep the whole setup as a local-run application.