**Design Decisions**

* The RAML design was done based on the analysis of the backend web service exposed (webserviceX mock using docker)
* The transformation was performed by extracting the cData message in the response
* The query parameters were determined based on the requirement for each of the GET APIs exposed
* All the exceptions were traced and adequate on error propagate handling was performed to propagate the apt message to the API consumer
* Adequate enumerations were chosen as per the countries supported
* The examples and types were designed after taking into consideration of the requirement for of the APIs exposed

**Challenges**

* Understanding and designing the response fields to be exposed as part of the API. It was able to overcome this challenge through clear analysis of the essentials related to backend service and functionality exposed
* Defining the RAML was a challenge faced – This was overcome through clearly defining the resources, types, examples in the right manner based on the requirement specified
* Defining the exception handling – Clear exception handling was defined post analysis of the types of exceptions and categorization
* Logging framework – Defined loggers at critical processing junctures
* Handling the cData using transformation in the right manner – This was addressed using the transform event processor and the specific function used