TEST QA FALABELLA FINANCIERO

El siguiente test medirá los conocimientos teóricos y prácticos del participante para el cargo de Analista QA Automation en la gerencia de integración. Consiste en primer lugar de evaluar un caso hipotético de una API rest y generar un set de pruebas que asegure el correcto funcionamiento del servicio. En segundo lugar, se plantea realizar pruebas a una api publica y en tercer lugar se plantea una serie de conceptos que podrá describir con sus propias palabras, por ultimo una pequeña validación de sus conocimientos en js.

Favor enviar el test en un repositorio Github para validarlo

1.- Dado el siguiente escenario:

*Mockear ambiente y endpoint para esta prueba.*

*Dentro de los servicios digitales de banco falabella, se encuentra la revisión del estado de cuenta del cliente, los productos que tiene asignado, puntos acumulados, etc. De estos servicios se necesita realizar pruebas de contrato a la API que conecta el sistema de información de datos con la interfaz que maneja el cliente. La API arroja el siguiente archivo JSON:*

*{*

*Dato\_cliente: {*

*Nombre: string,*

*Apellido: string,*

*Direccion: string,*

*Email: string,*

*Telefono: number,*

*RUT: number*

*},*

*Dato\_perfil: {*

*Cuenta\_ppal: number,*

*Tipo\_cliente: string,*

*Tarjeta\_debito: number*

*},*

*Dato\_Sucursal: {*

*Sucursal\_nro: number,*

*Ejecutivo: string,*

*Telefono\_ejec: number,*

*Direccion\_suc: string*

*},*

*Dato\_productos: {*

*Tiene\_prod: boolean,*

*Tipo\_producto: [*

*Consumo,*

*Hipotecario,*

*Linea\_emergencia,*

*Linea\_sobregiro,*

*],*

*Puntos: number,*

*Tarjeta\_credito: number*

*},*

*Datos\_seguros: {*

*Tiene\_prod: boolean,*

*SOAP: string,*

*Seguro\_vehiculo: string,*

*Seguro\_fraude: string,*

*Seguro\_hogar: string,*

*Seguro\_vida: string*

*}*

*Datos\_fondos: {*

*Fondo\_mutual: string,*

*Ahorro\_previsional: number,*

*ctaRentaDiaria: string*

*}*

*Datos\_penalizaciones: {*

*InteresesMora\_Sobregiro: number,*

*InteresMora\_TarjetaCredito: number,*

*InteresMora\_Consumo: number,*

*InteresMora\_Hipotecario: number*

*}*

*}*

Realizar las siguientes validaciones y agregar las que considere necesaria:

* Validar la estructura del Json.
* Validar respuesta 200.
* Validar que los tipos de datos sean correctos.
* Validar Mensaje OK.
* Validar que el formato sea JSON.
* Validar que RUT sea un número.

2.- Desde la siguiente URL <https://restcountries.eu/#api-endpoints-name>, tomar el endpoint <https://restcountries.eu/rest/v2/name/>{name} y realizar las validaciones que se consideren necesarias, se deben escribir mínimo 6 tests.

A continuación, explique con sus propias palabras los siguientes conceptos:

1. Pruebas de Contrato
2. API Rest.
3. Integración Continua.
4. Despliegue Continuo.
5. Mencione algunas herramientas para realizar pruebas a API y Web Services.

3.- Esta sección es para validar sus conocimientos sobre javascript:

function Sumar(a,b) {

return a + b;

}

function Producto(a,b) {

var c = 20;

return a\*b\*c;

}

Obtener en consola los resultados de las funciones de arriba.

Persona = {

nombre: ‘Juan’,

apellido: ‘Perez’,

telefono: 900199703,

email: ‘juan.perez@gmail.com’

}

Realizar una función que obtenga las propiedades del Objeto Persona.

4.- Dado el siguiente Wsdl http://216.10.245.166:8080/axis2/services/EmployeeManagementService?wsdl

Agregar TestSuite con las validaciones que se consideren necesarias, agregar los assertions que se requieran según el request y el response obtenido.

Que validaciones realizaría al web service?

Cualquier duda comunicarse por correo afrancoh@falabella.cl

Favor enviar el test en un repositorio Github para validarlo

**Buena Suerte!!!**

**RESULTADOS:**

* Validar la estructura del Json.

La estructura del Json es la incorrecta:

Se valida lo siguiente:

* Las variables deben tener “”. Ej: “Dato\_clientes”: {

“rut”: 251800642

}

* El tipo de dato si es string debe tener “”. Ej: “Dato\_clientes”: {

“Nombre”: “Isabel”

}

* No se agraga el tipo de dato, si no el dato como tal y sin “”: Ej: “Dato\_clientes”: {

“rut”: 251800642

}

Otro ejemplo:

"Datos\_seguros": {

"Tiene\_prod": true

}

* Cada estructura Json va separada por una coma (,) Ej:

"Datos\_fondos": {

"Fondo\_mutual": "Fondo",

"Ahorro\_previsional": 123456,

"ctaRentaDiaria": "cuenta"

},

"Datos\_penalizaciones": {

"InteresesMora\_Sobregiro": 7869,

"InteresMora\_TarjetaCredito": 9842,

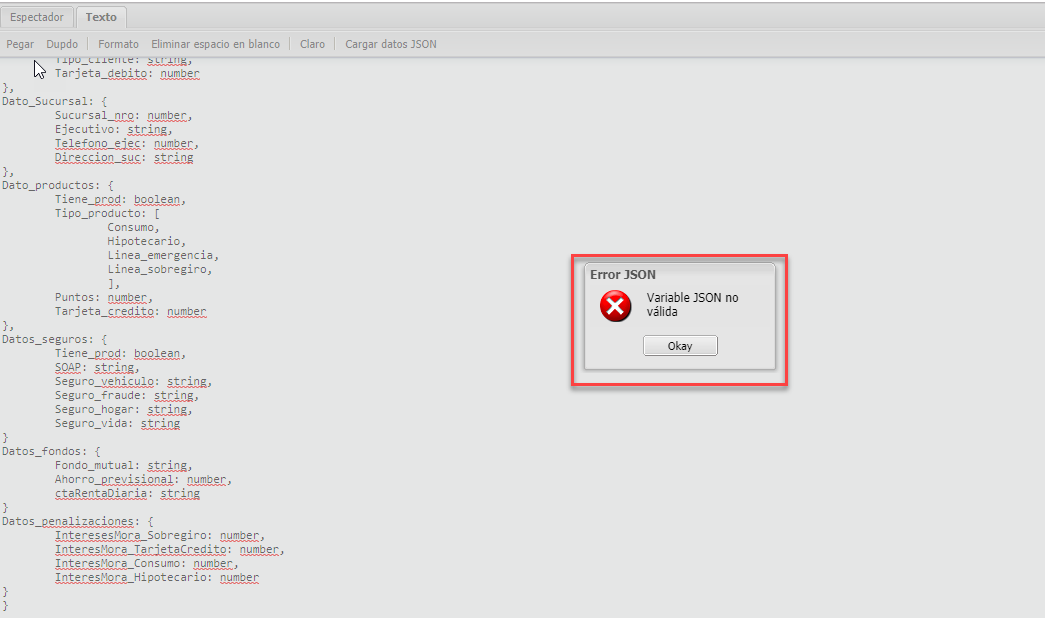
"InteresMora\_Consumo": 6722,

"InteresMora\_Hipotecario": 45367

} 🡪 al finalizar el último no tiene coma (,)

}

Evidencia de que el código no es el correcto:



La estructura correcta es la siguiente:

{

"Dato\_cliente": {

"Nombre": "Isabel",

"Apellido": "Miralles",

"Direccion": "Santiago",

"Email": "isa.miralles1986@gmail.com",

"Telefono": 951338844,

"RUT": 251800642

},

"Dato\_perfil": {

"Cuenta\_ppal": 999999,

"Tipo\_cliente": "Principal",

"Tarjeta\_debito": 4567879876563452

},

"Dato\_Sucursal": {

"Sucursal\_nro": 234,

"Ejecutivo": "Marcela",

"Telefono\_ejec": 2345432,

"Direccion\_suc": "Ahumada"

},

"Dato\_productos": {

"Tiene\_prod": true,

"Tipo\_producto": ["Consumo","Hipotecario","Linea\_emergencia","Linea\_sobregiro"],

"Puntos": 2390,

"Tarjeta\_credito": 4966478976564352

},

"Datos\_seguros": {

"Tiene\_prod": true,

"SOAP": "SOAP",

"Seguro\_vehiculo": "Falabella",

"Seguro\_fraude2": "Falabella",

"Seguro\_hogar": "Falabella",

"Seguro\_vida": "Falabella"

},

"Datos\_fondos": {

"Fondo\_mutual": "Fondo",

"Ahorro\_previsional": 123456,

"ctaRentaDiaria": "cuenta"

},

"Datos\_penalizaciones": {

"InteresesMora\_Sobregiro": 7869,

"InteresMora\_TarjetaCredito": 9842,

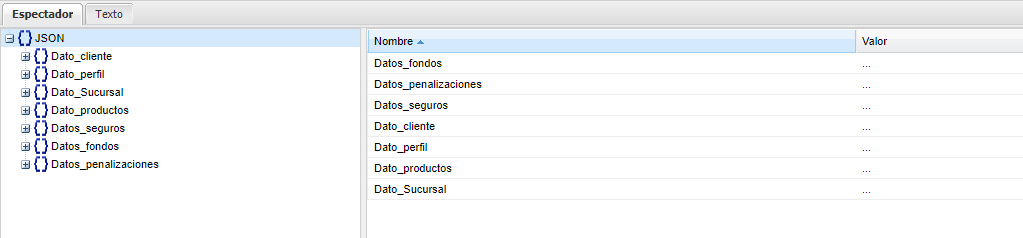
"InteresMora\_Consumo": 6722,

"InteresMora\_Hipotecario": 45367

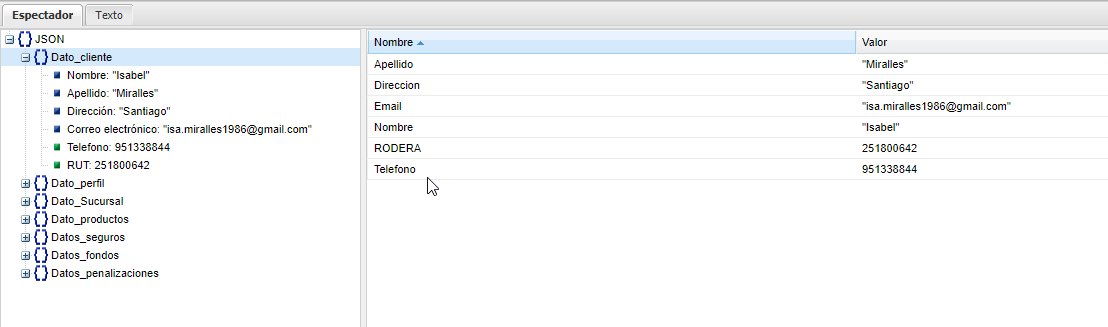
}

}

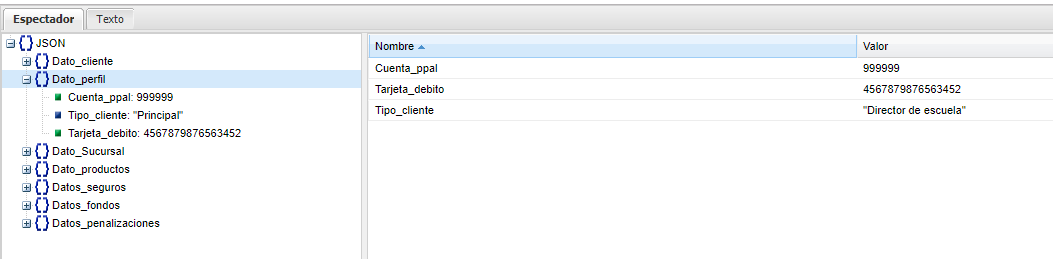
Evidencia de que la estructura mencionada es la correcta:



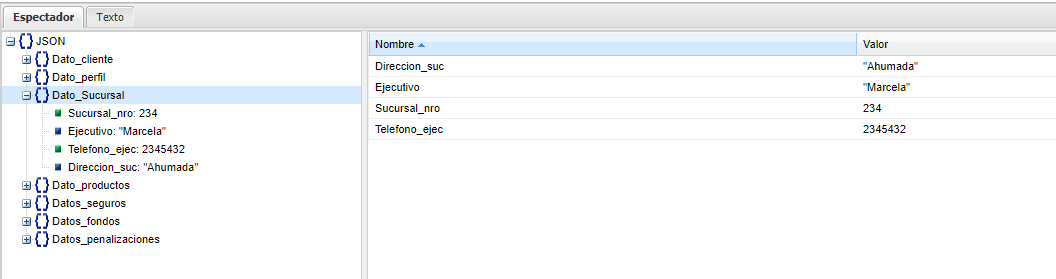
Dato Cliente:



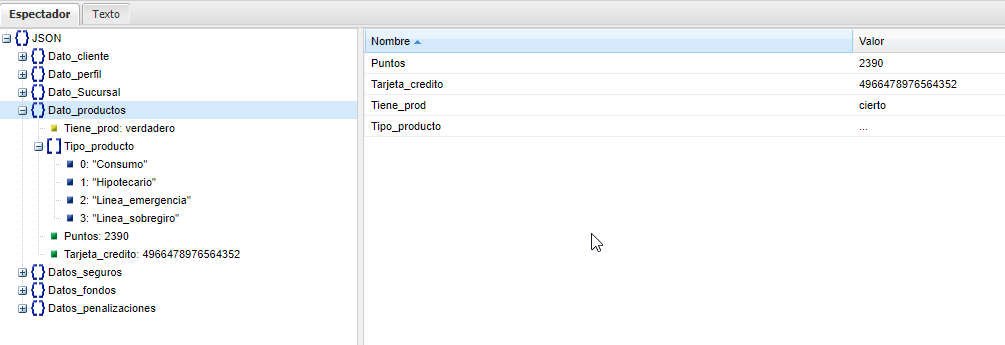
Dato Perfil:



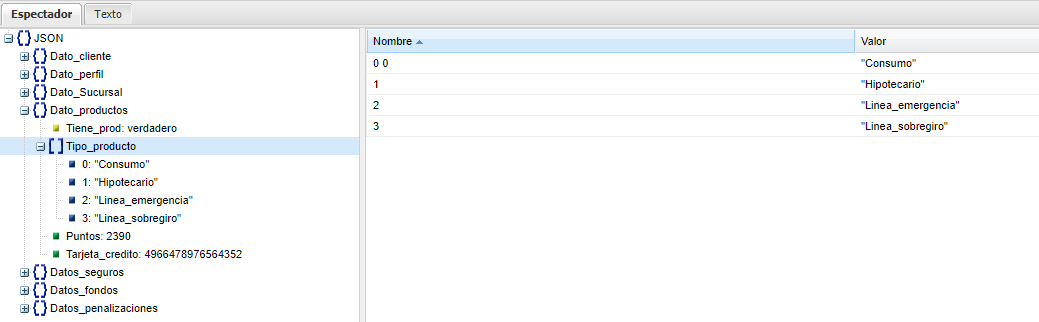
Dato Sucursal:



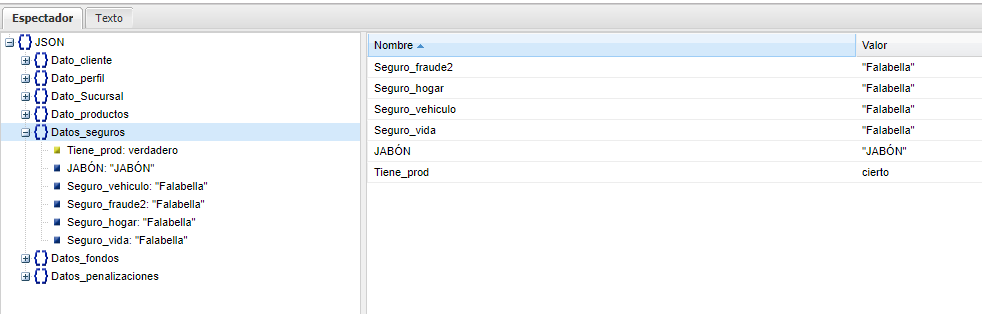
Dato Producto:



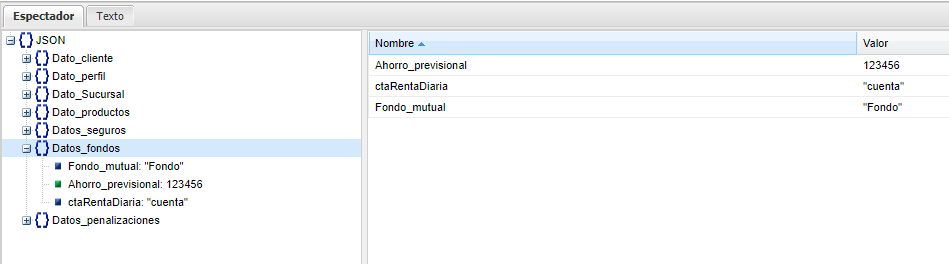
Tipo Producto (arreglo)



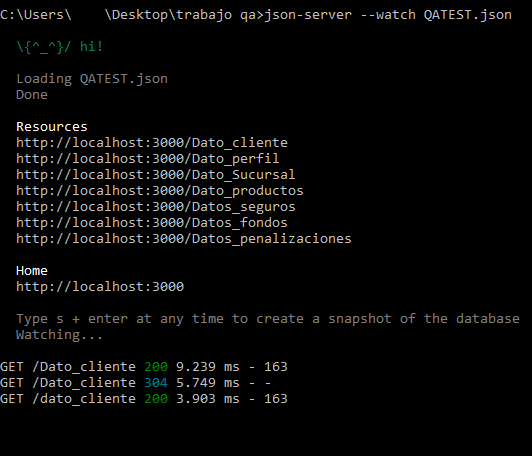
Datos seguros:

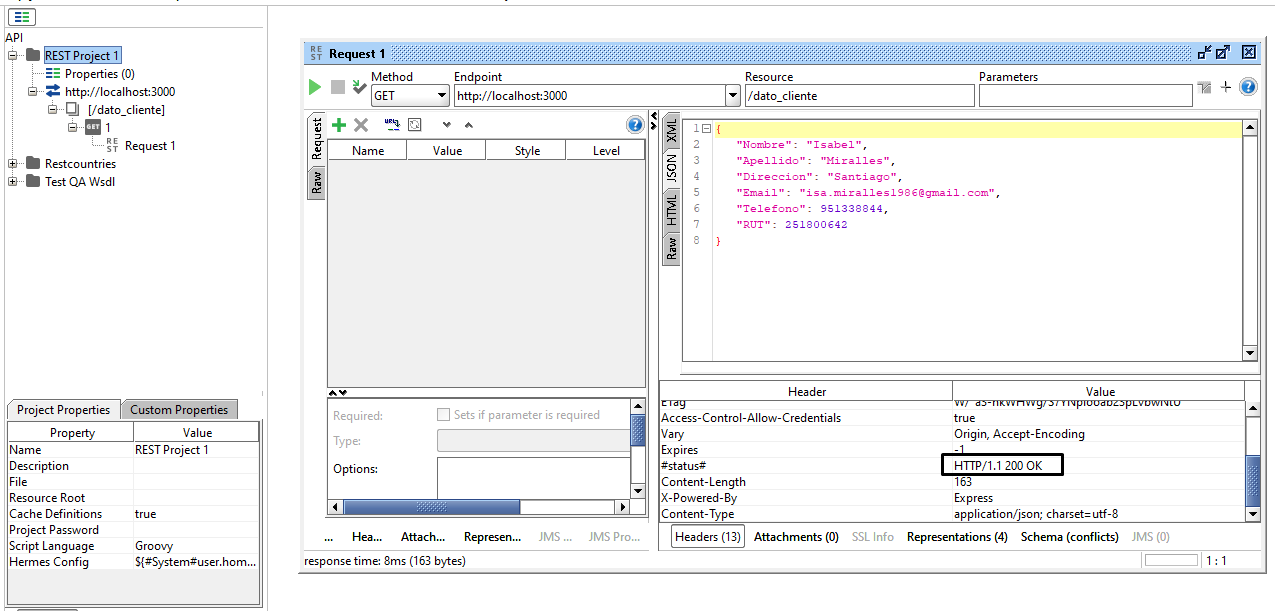


Datos Fondos:



* Validar respuesta 200 - Validar Mensaje OK.





* Validar que los tipos de datos sean correctos.

Loa tipos de datos no son los correctos:

* El tipo de dato si es string debe tener “” y copiar el dato, no el tipo de dato. Ej: “Dato\_clientes”: {

“Nombre”: “Isabel”

}

* No se agrega el tipo de dato, si no el dato como tal y sin “” cuando es numérico, boolean, float, double: Ej: “Dato\_clientes”: {

“rut”: 251800642

}

Otro ejemplo:

"Datos\_seguros": {

"Tiene\_prod": true

}

* Validar que el formato sea JSON.

Si tiene formato JSON, sin embargo, faltaron comas (,)

{

"Dato\_cliente": {

"Nombre": "Isabel",

"Apellido": "Miralles",

"Direccion": "Santiago",

"Email": "isa.miralles1986@gmail.com",

"Telefono": 951338844,

"RUT": 251800642

},

"Dato\_perfil": {

"Cuenta\_ppal": 999999,

"Tipo\_cliente": "Principal",

"Tarjeta\_debito": 4567879876563452

},

"Dato\_Sucursal": {

"Sucursal\_nro": 234,

"Ejecutivo": "Marcela",

"Telefono\_ejec": 2345432,

"Direccion\_suc": "Ahumada"

},

"Dato\_productos": {

"Tiene\_prod": true,

"Tipo\_producto": ["Consumo","Hipotecario","Linea\_emergencia","Linea\_sobregiro"],

"Puntos": 2390,

"Tarjeta\_credito": 4966478976564352

},

"Datos\_seguros": {

"Tiene\_prod": true,

"SOAP": "SOAP",

"Seguro\_vehiculo": "Falabella",

"Seguro\_fraude2": "Falabella",

"Seguro\_hogar": "Falabella",

"Seguro\_vida": "Falabella"

}, 🡪 acá va coma.

"Datos\_fondos": {

"Fondo\_mutual": "Fondo",

"Ahorro\_previsional": 123456,

"ctaRentaDiaria": "cuenta"

}, 🡪 acá va coma.

"Datos\_penalizaciones": {

"InteresesMora\_Sobregiro": 7869,

"InteresMora\_TarjetaCredito": 9842,

"InteresMora\_Consumo": 6722,

"InteresMora\_Hipotecario": 45367

}

}

* Validar que RUT sea un número.

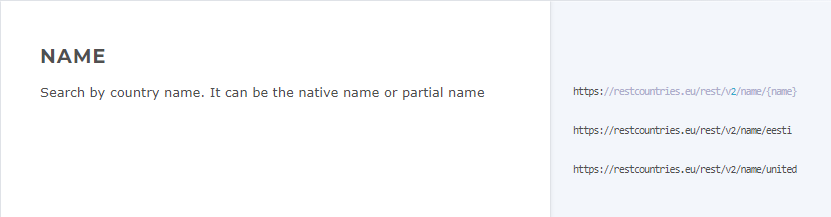
No se agrega el tipo de dato, si no el dato como tal y sin “” cuando es numérico. Ej: “Dato\_clientes”: {

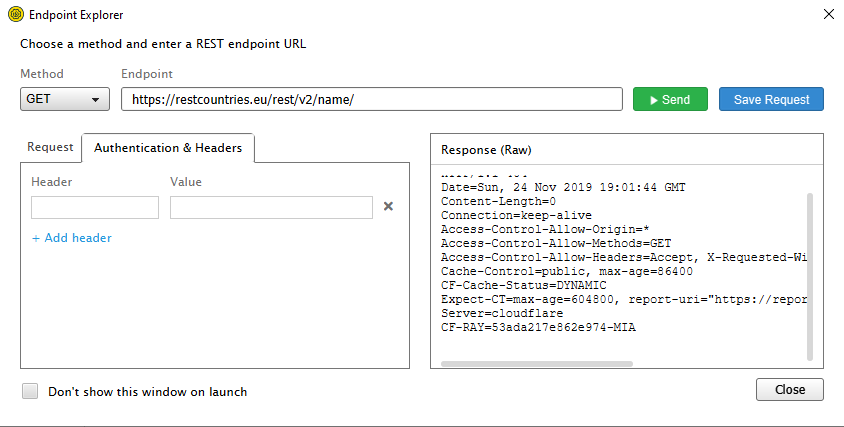
“rut”: 251800642

}

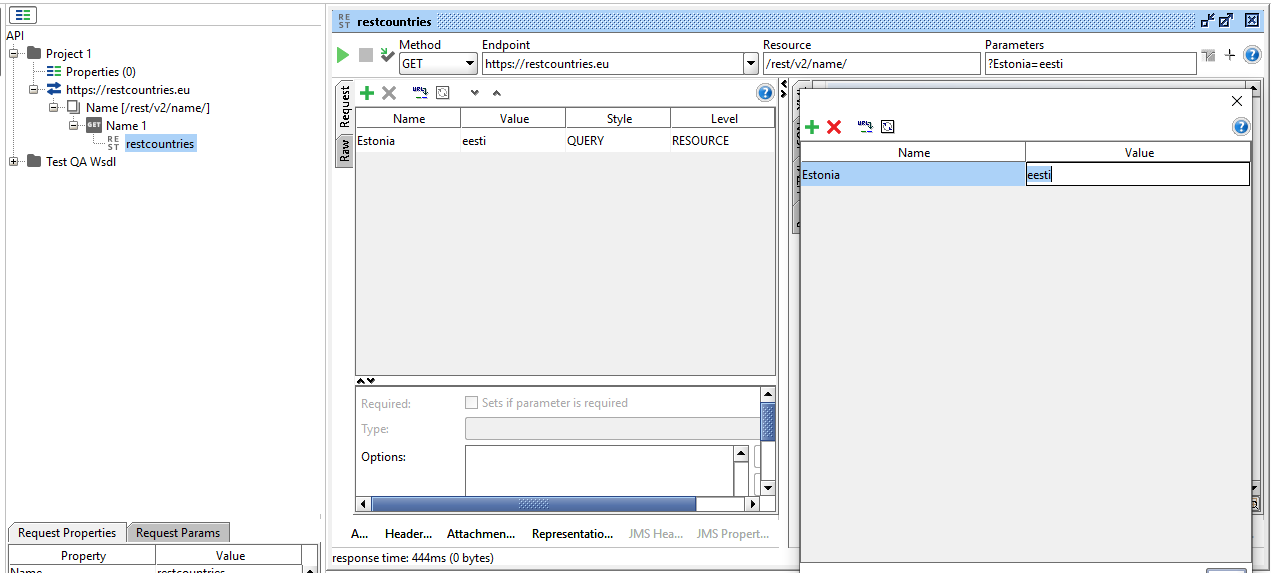
2.- Desde la siguiente URL <https://restcountries.eu/#api-endpoints-name>, tomar el endpoint <https://restcountries.eu/rest/v2/name/>{name} y realizar las validaciones que se consideren necesarias, se deben escribir mínimo 6 tests.

Agregamos el Endpoint en Soapui:

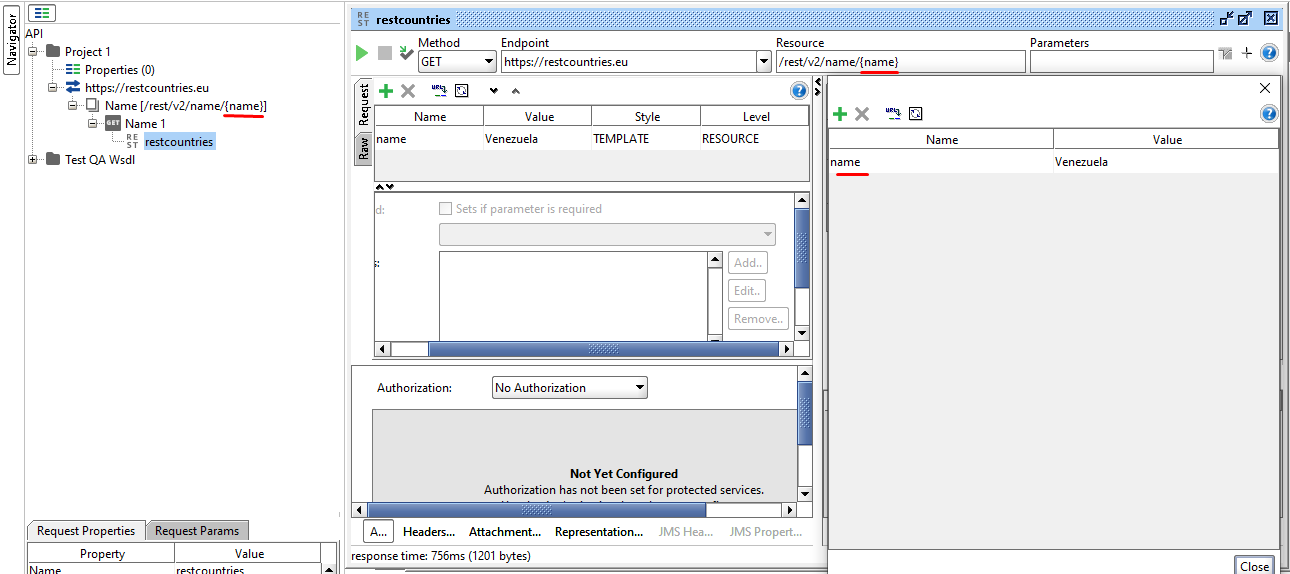




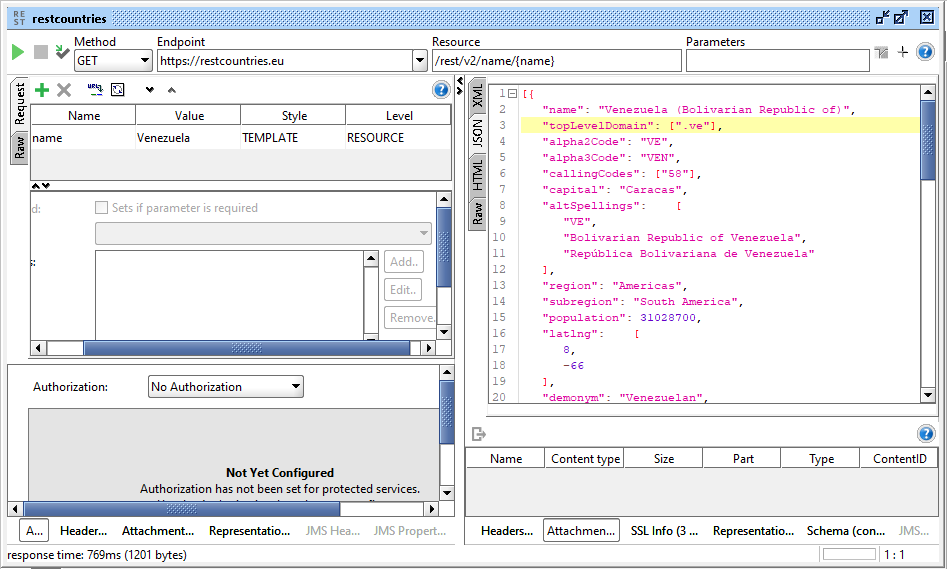
Creamos nuevo proyecto para el endpoint,



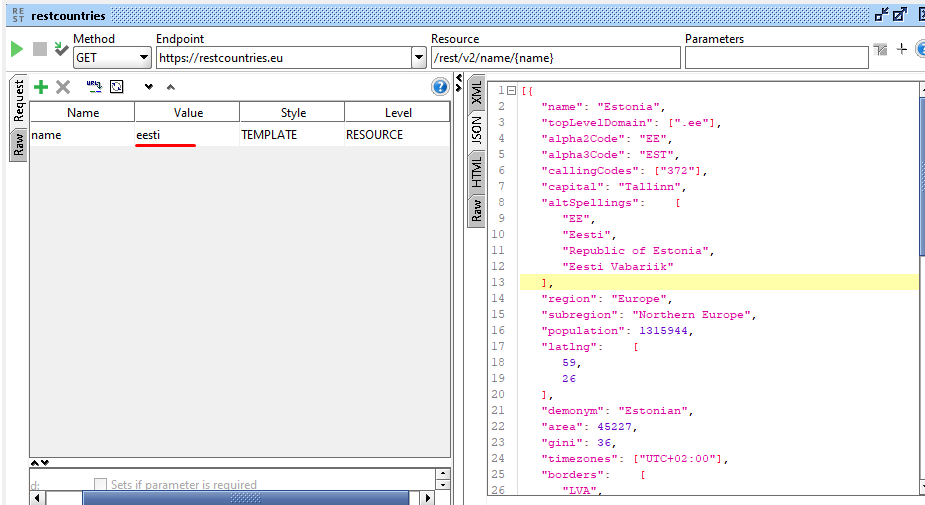
Le agregamos parámetro de entrada tenemos:



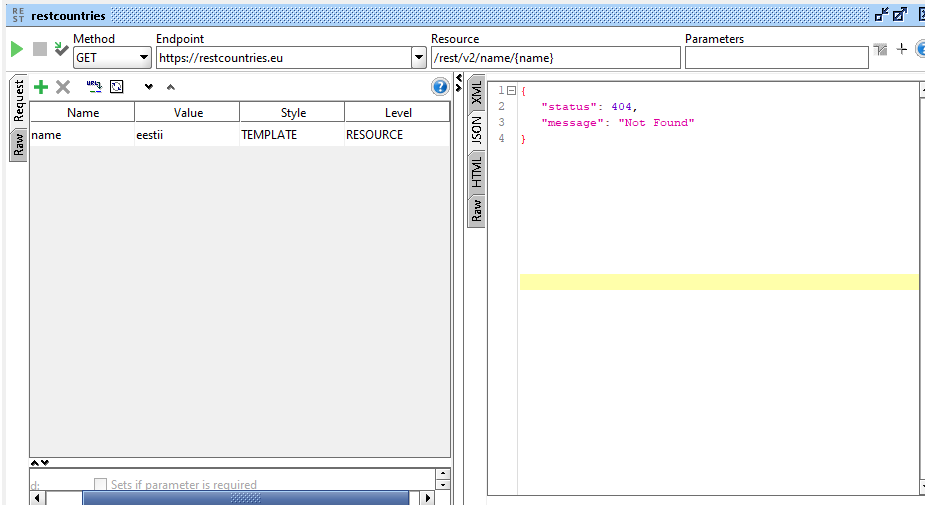
Y ejecutamos



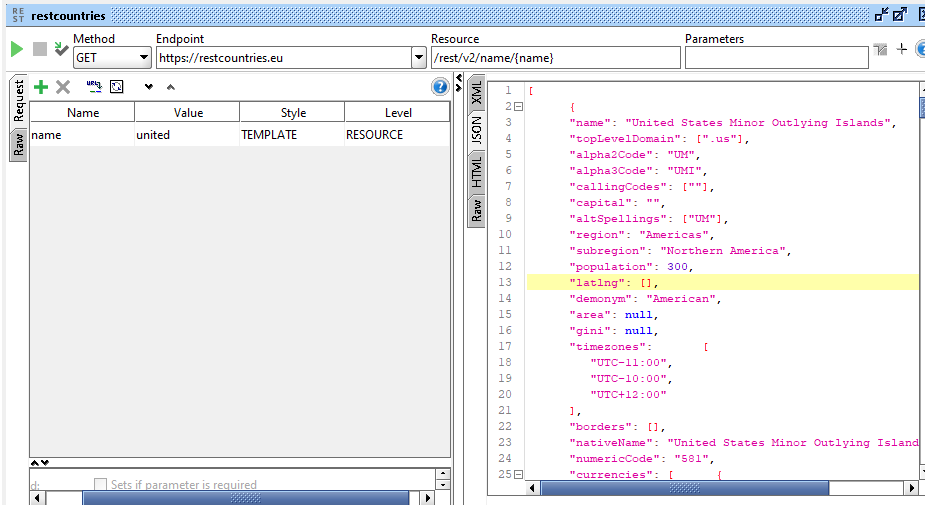
Validar endpoint nombre del ejemplo de página, empezando con estonia:



Verificar endpoint con el país Estonia, agregándole otro carácter, por ejemplo: estonii

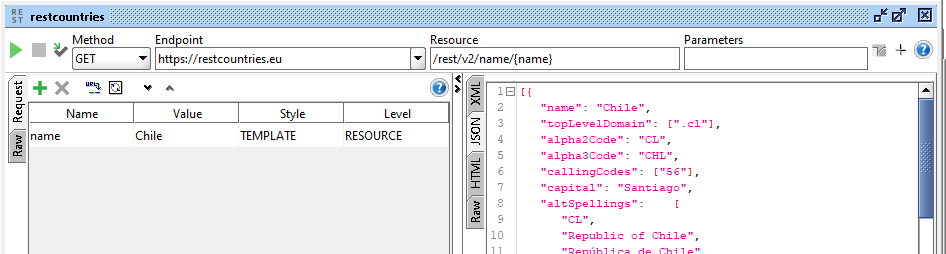


Verificar endpoint agregándole el valor United:

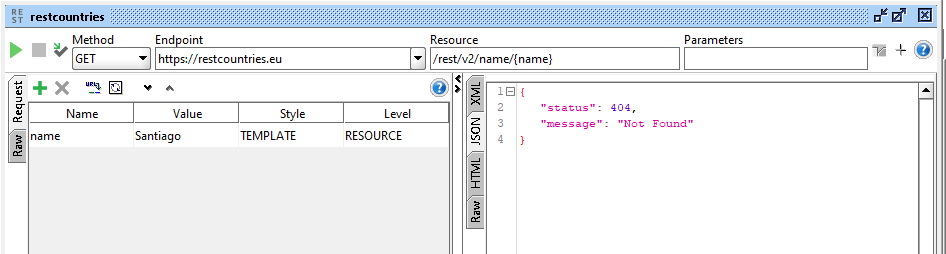


Validar agregando en el campo name, la capital de chile: Santiago.

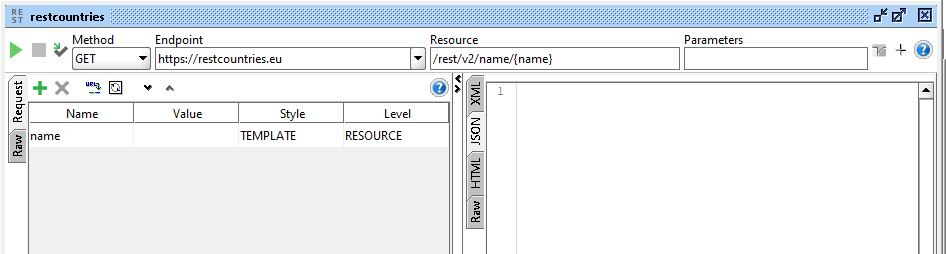
Primero verificado el país chile:



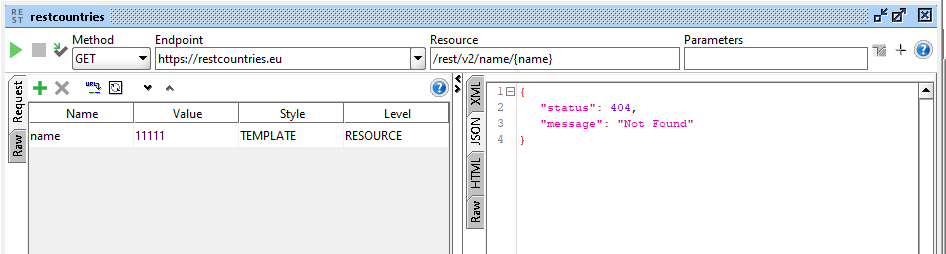
Luego consultamos su capital que es Santiago:



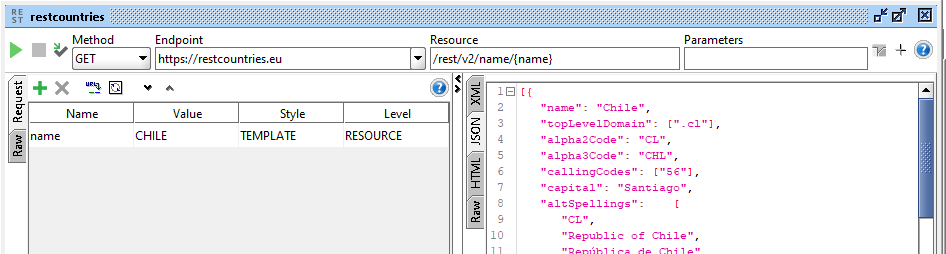
Verificar que el Campo name su valor sea igual a null:

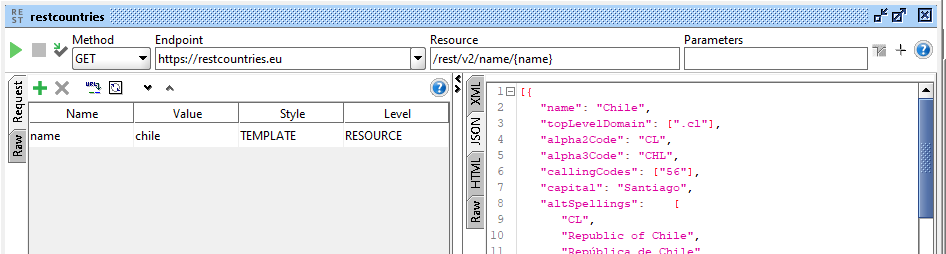


Verificar campo name agregando solo números:

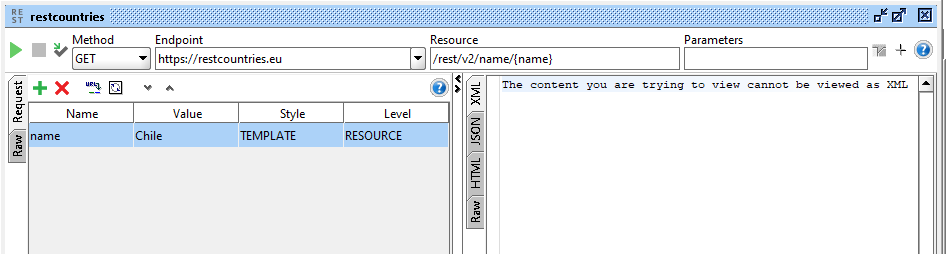


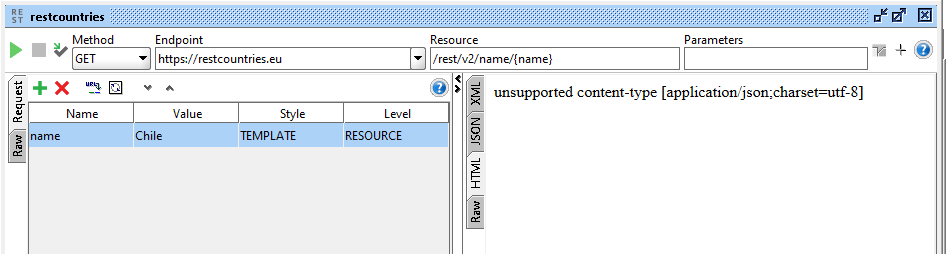
Corroborar que el campo name, acepta mayúsculas y minúsculas

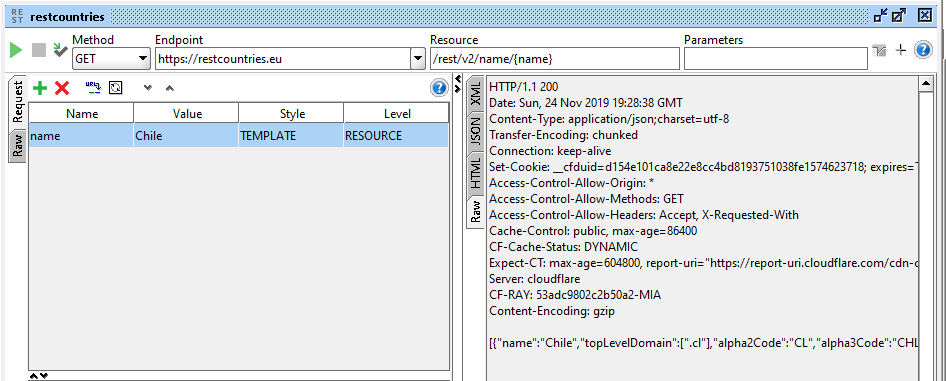




Validar que la salida sea solo en formato Jason:







Se muestra formato Json generado:

[{

"name": "Chile",

"topLevelDomain": [".cl"],

"alpha2Code": "CL",

"alpha3Code": "CHL",

"callingCodes": ["56"],

"capital": "Santiago",

"altSpellings": [

"CL",

"Republic of Chile",

"República de Chile"

],

"region": "Americas",

"subregion": "South America",

"population": 18191900,

"latlng": [

-30,

-71

],

"demonym": "Chilean",

"area": 756102,

"gini": 52.1,

"timezones": [

"UTC-06:00",

"UTC-04:00"

],

"borders": [

"ARG",

"BOL",

"PER"

],

"nativeName": "Chile",

"numericCode": "152",

"currencies": [ {

"code": "CLP",

"name": "Chilean peso",

"symbol": "$"

}],

"languages": [ {

"iso639\_1": "es",

"iso639\_2": "spa",

"name": "Spanish",

"nativeName": "Español"

}],

"translations": {

"de": "Chile",

"es": "Chile",

"fr": "Chili",

"ja": "チリ",

"it": "Cile",

"br": "Chile",

"pt": "Chile",

"nl": "Chili",

"hr": "Čile",

"fa": "شیلی"

},

"flag": "https://restcountries.eu/data/chl.svg",

"regionalBlocs": [

{

"acronym": "PA",

"name": "Pacific Alliance",

"otherAcronyms": [],

"otherNames": ["Alianza del Pacífico"]

},

{

"acronym": "USAN",

"name": "Union of South American Nations",

"otherAcronyms": [

"UNASUR",

"UNASUL",

"UZAN"

],

"otherNames": [

"Unión de Naciones Suramericanas",

"União de Nações Sul-Americanas",

"Unie van Zuid-Amerikaanse Naties",

"South American Union"

]

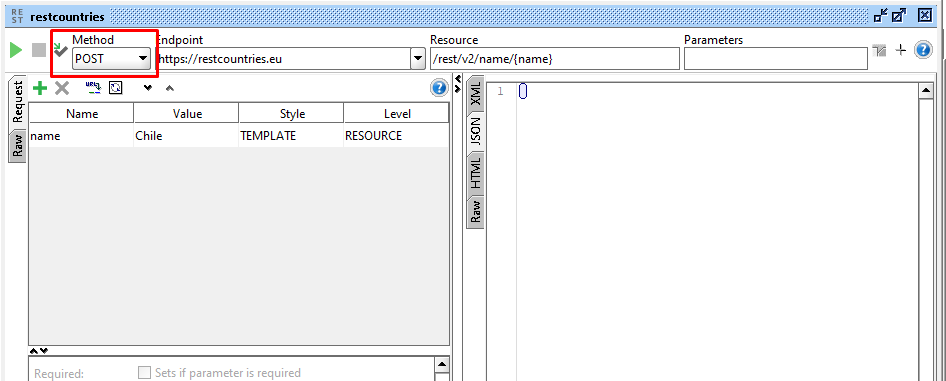
}

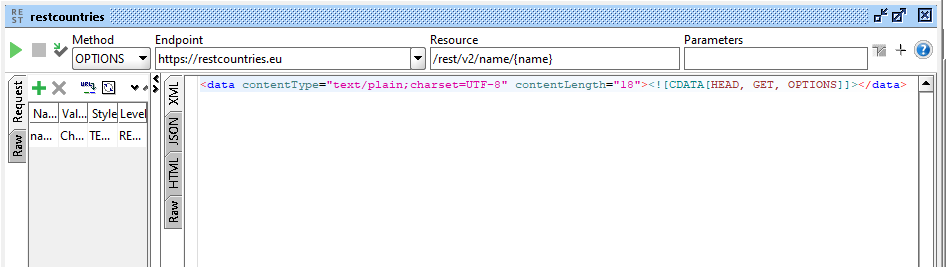
],

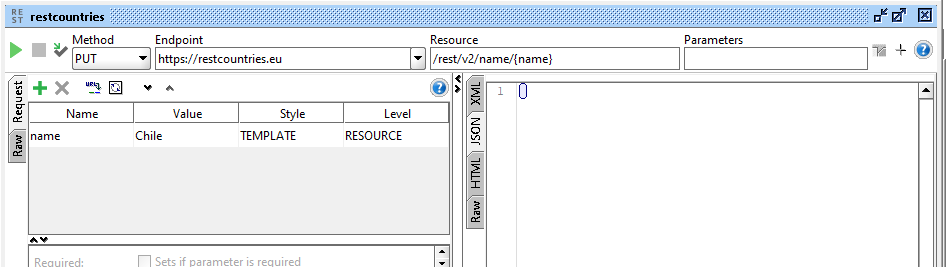
"cioc": "CHI"

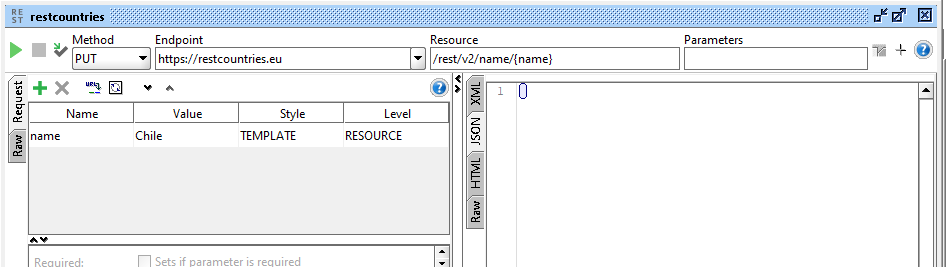
}]

Validar comportamiento al cambiar el método Get por PUT/ DELETE / POST entre otros:



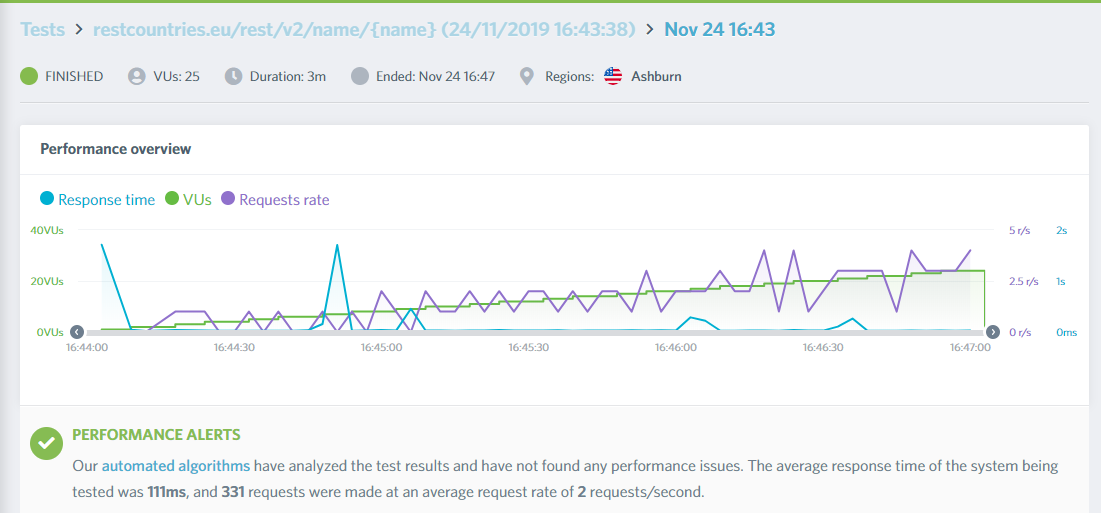


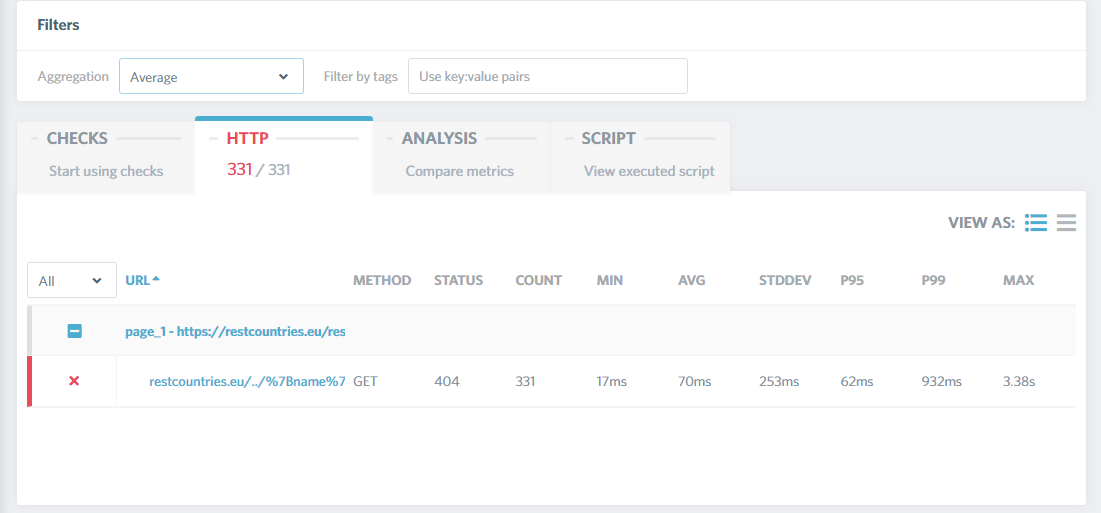


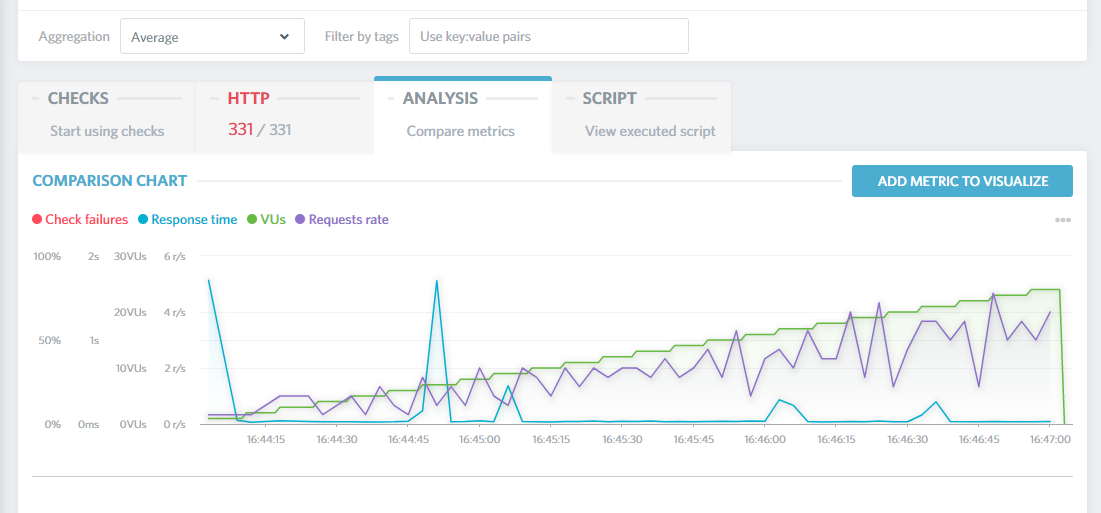


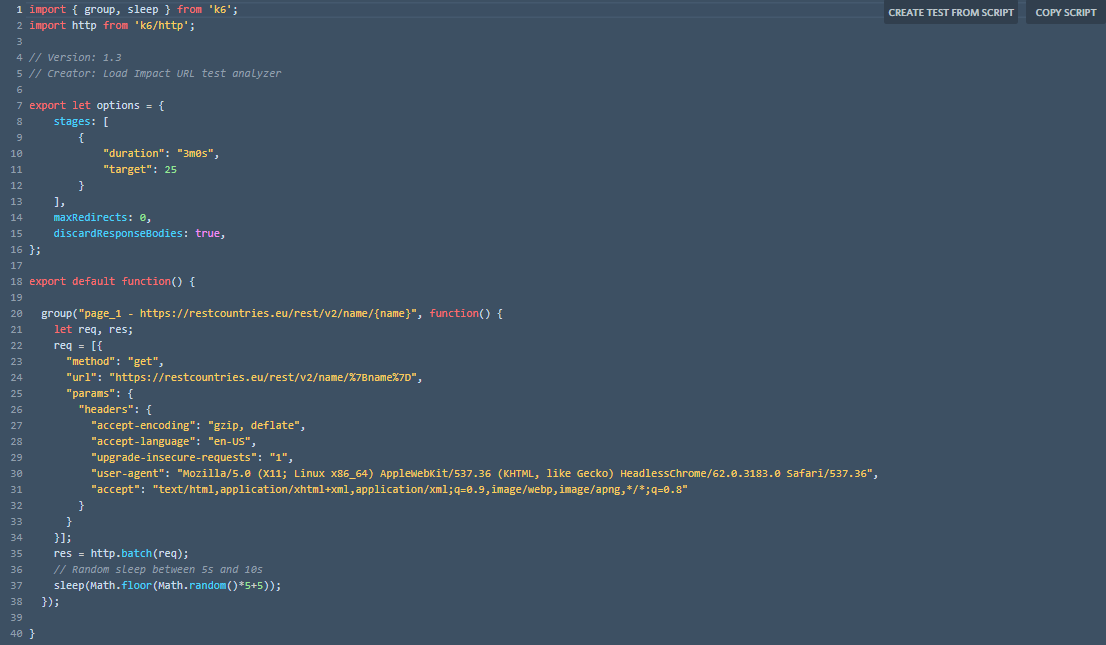
Realizar pruebas de stress, al Endpoint y ver su comportamiento:

Utilizando la <https://loadimpact.com/>, ingresamos la url, y tenemos el siguiente comportamiento:









A continuación, explique con sus propias palabras los siguientes conceptos:

1. Pruebas de Contrato: son las definiciones ideas y objetivos de lo que queremos probar un producto o incremento y a raíz de ahí elaboramos las posibles pruebas, usando mapas mentales u otra herramienta que nos ayuda representar de forma intuitiva nuestro plan de pruebas. esto se usa mucho en las pruebas exploratorias.
2. API Rest: Es una aplicación web que se encuentra en el lado del backend, en sí es un servicio web.
3. Integración Continua: Integrar de forma continua cambios lo más rápido posible con el menor fallos posibles en nuestro repositorio. Ejemplo cuando un desarrollador quiere añadir un cambio/arreglar un bug, se crea un branch de una rama, realiza las modificaciones, realiza test en local para ver que todo está bien, luego sube estos cambios.
4. Despliegue Continuo: Una vez se verifica pasando una cantidad de pruebas el producto o incremento se pasa a producción, en ningún momento la persona interviene para que pase los cambios al ambiente deseado, sea Test, Producción.
5. Mencione algunas herramientas para realizar pruebas a API y Web Services:

* Postman.
* Soapui.
* Jmeter.
* Wizdler.
* Runscope.

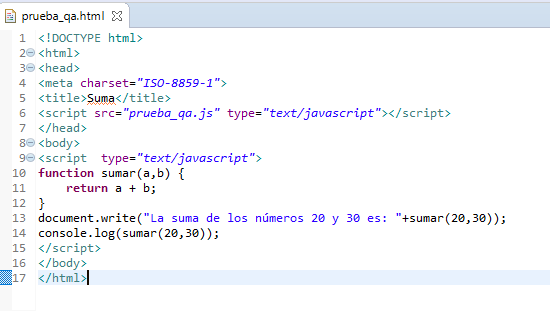
3.- Esta sección es para validar sus conocimientos sobre javascript:

function Sumar(a,b) {

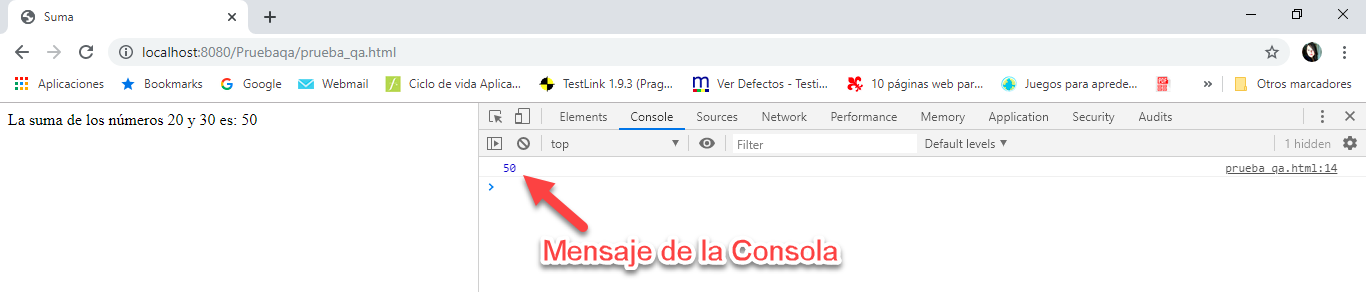
return a + b;

}

Código:



Ejecución:



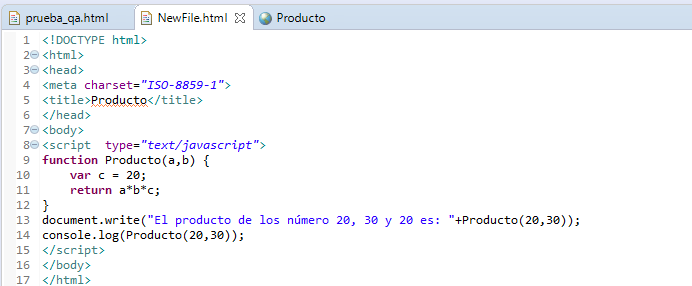
function Producto(a,b) {

var c = 20;

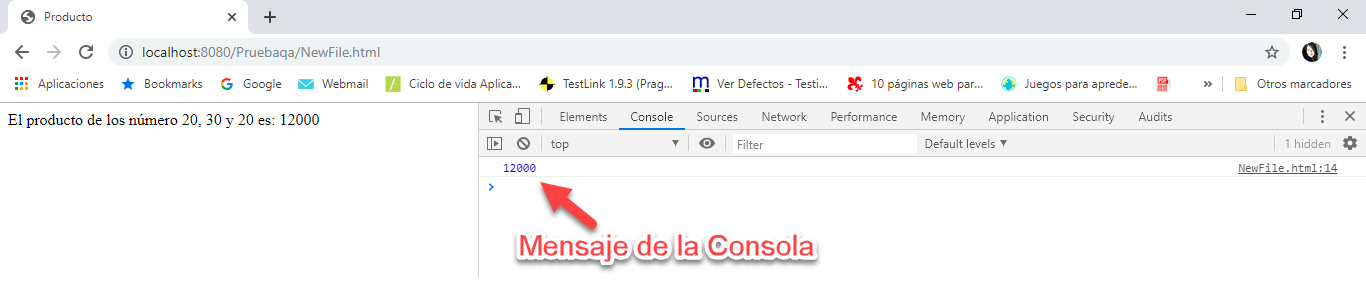
return a\*b\*c;

}

Código:



Ejecución:



Persona = {

nombre: ‘Juan’,

apellido: ‘Perez’,

telefono: 900199703,

email: ‘juan.perez@gmail.com’

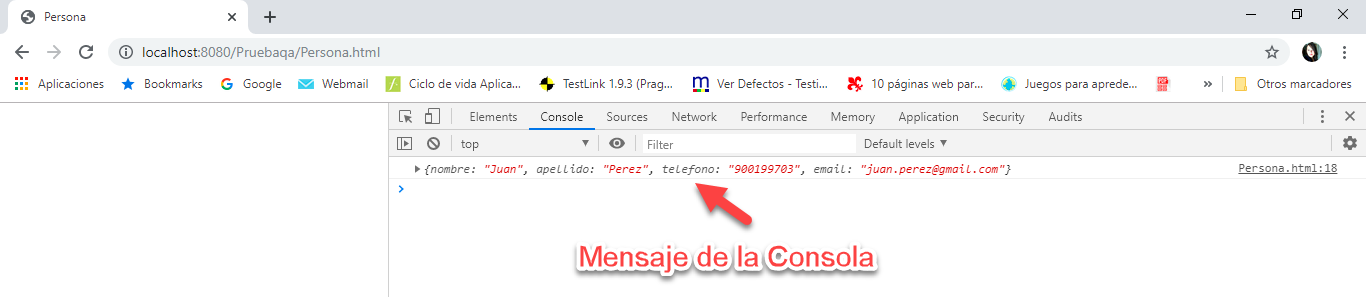
}

Realizar una función que obtenga las propiedades del Objeto Persona.

Código:



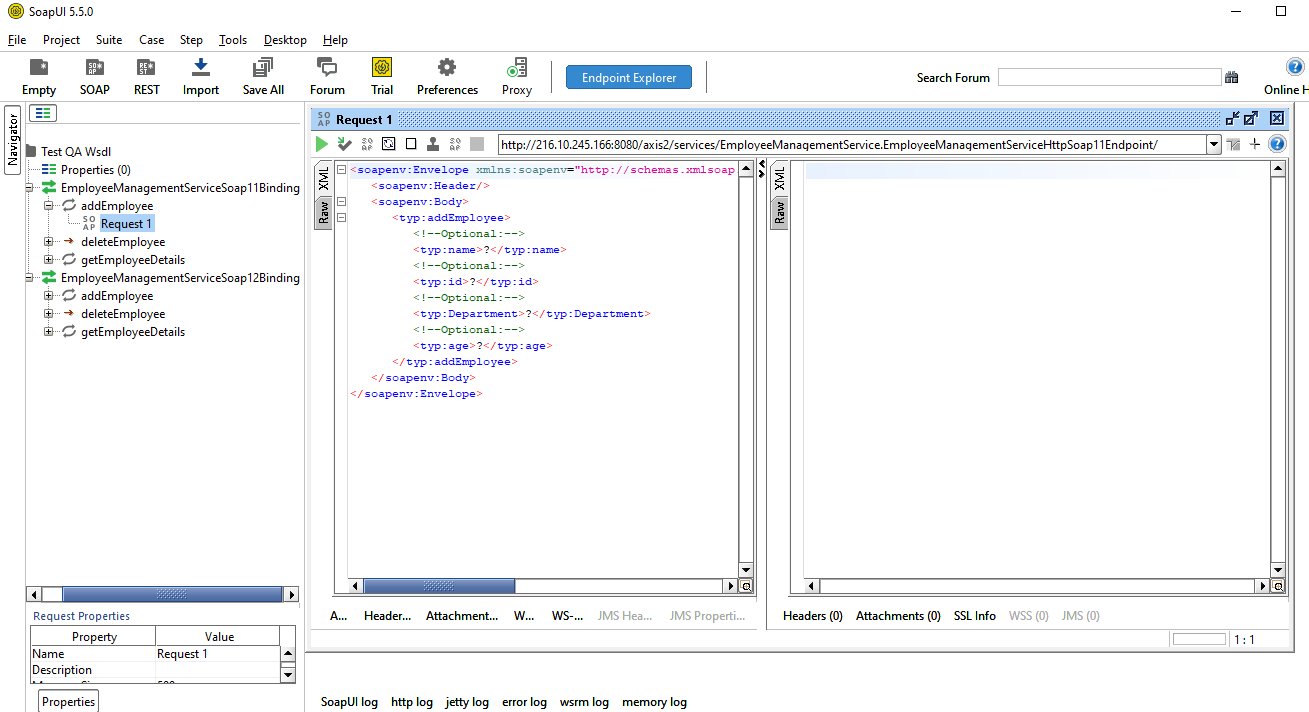
Ejecución:



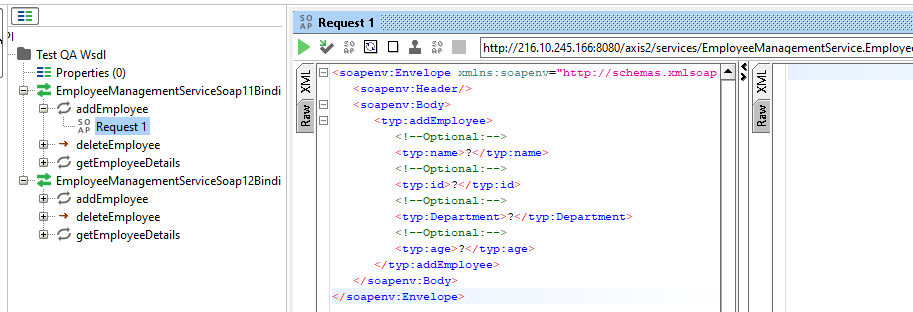
4.- Dado el siguiente Wsdl http://216.10.245.166:8080/axis2/services/EmployeeManagementService?wsdl

Agregar TestSuite con las validaciones que se consideren necesarias, agregar los assertions que se requieran según el request y el response obtenido.

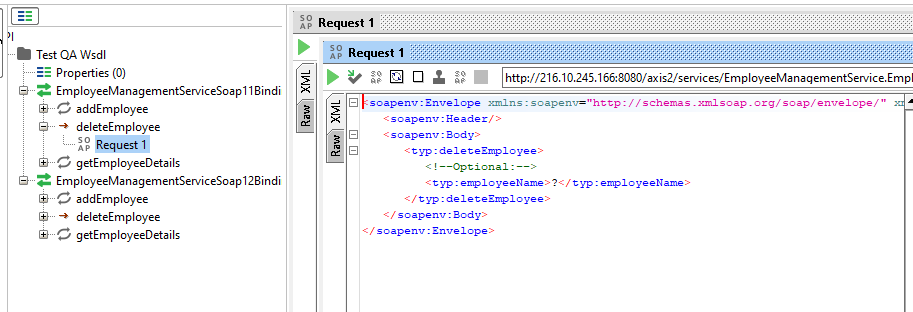
A continuación, muestro estructura del Web Services:



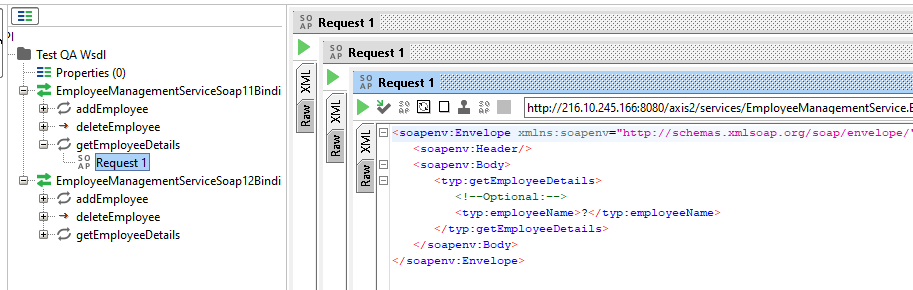
AddEmployee



DeleteEmployee



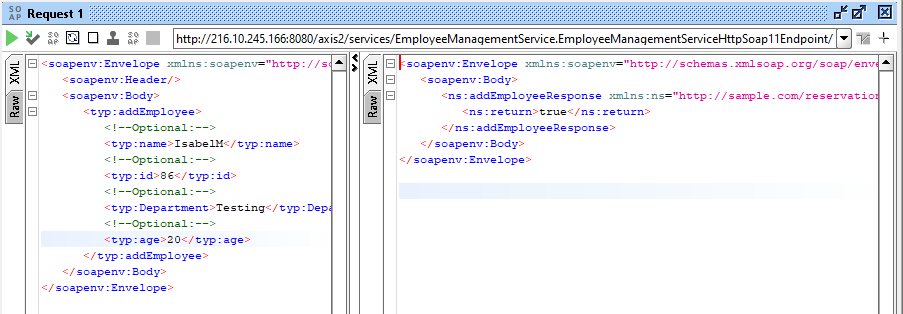
getEmployeeDetails



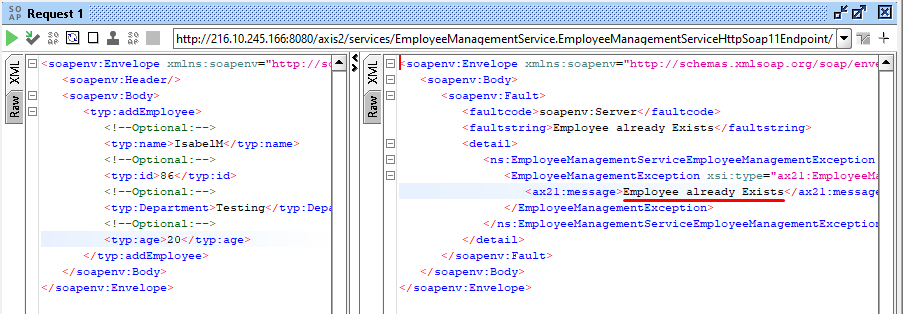
¿Que validaciones realizaría al web service?

Algunas de las validaciones que recomiendo son las siguiente:

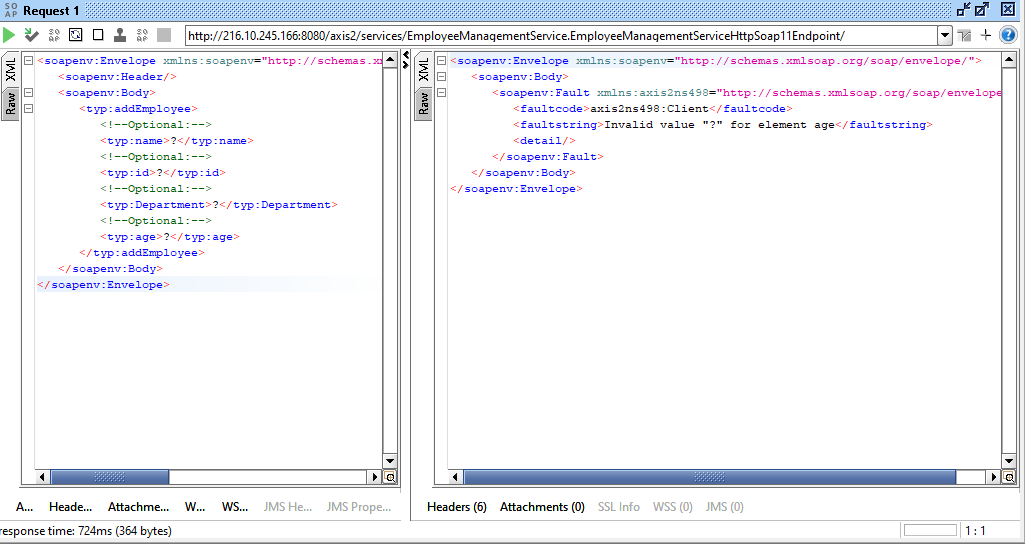
* Verificar añadir empleados Nuevo.



* Verificar añadir empleados existentes.

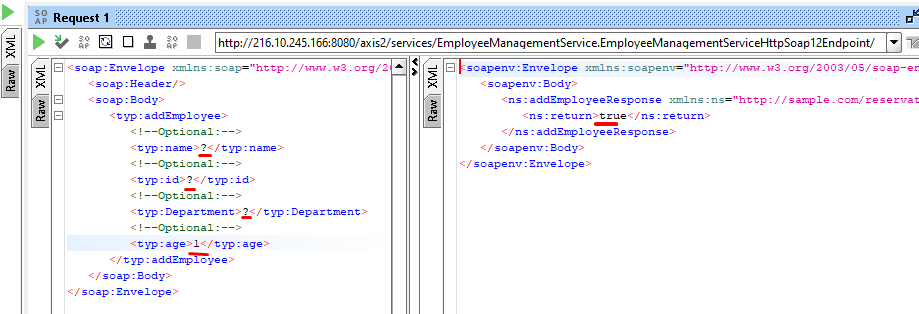


* Revisar añadir empleados sin data. Campos obligatorios

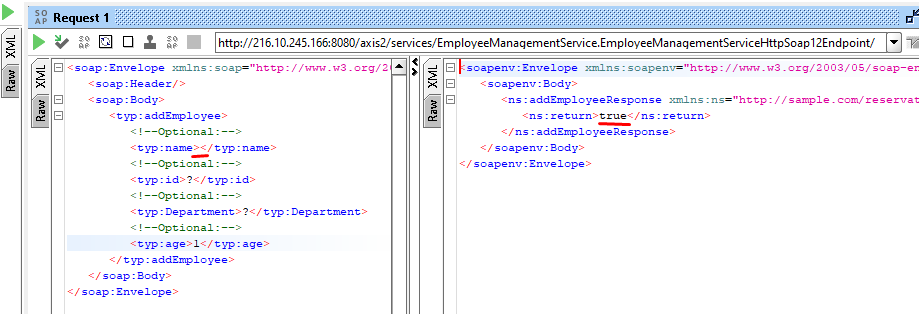


* Verificar agregar empleado solo insertando edad

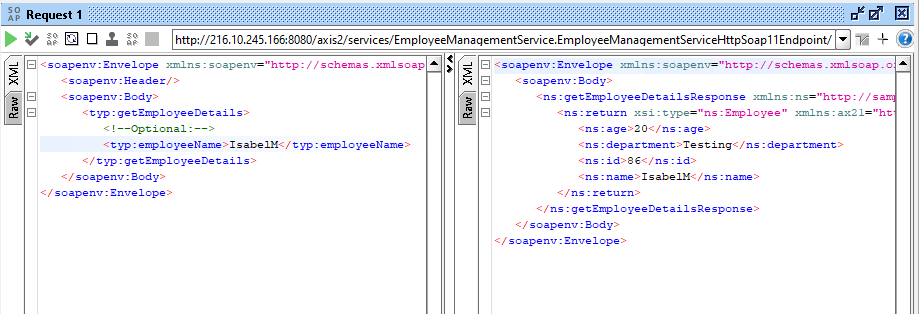
Permite ingresar empleado, con los campos name, id y Department con signo de interrogación



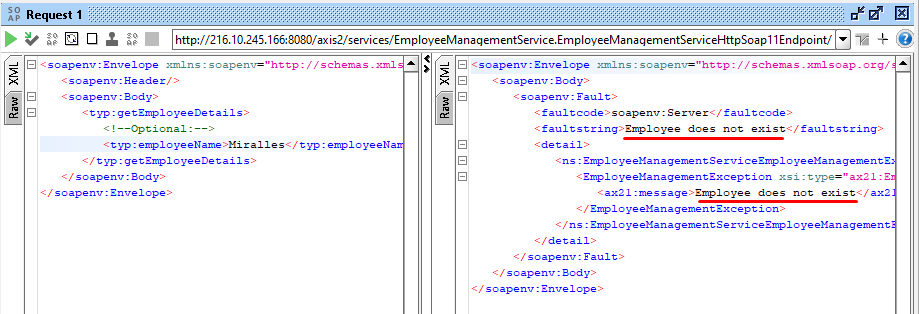
Permite ingresar empleado, sin ingresar data en el campo name



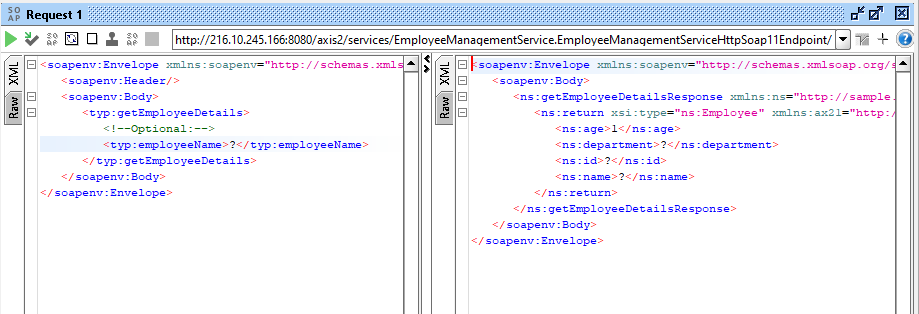
* Validar Consultar empleado existente.



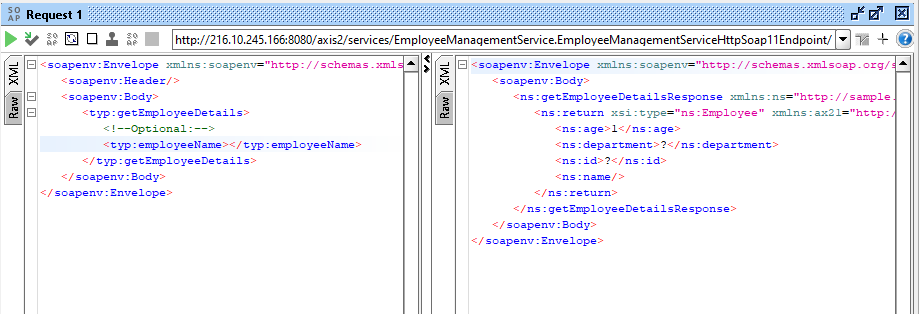
* Revisar Consultar empleado no existente.



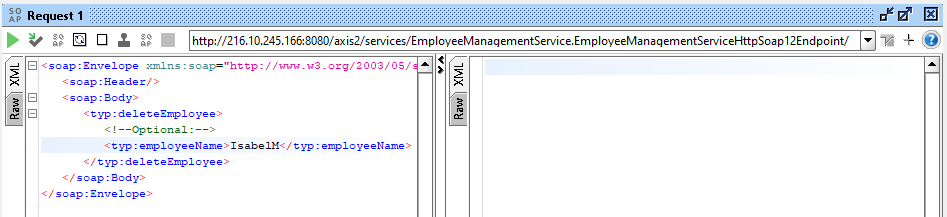
* Revisar Consultar empleado agregado con signo de interrogación en el campo name



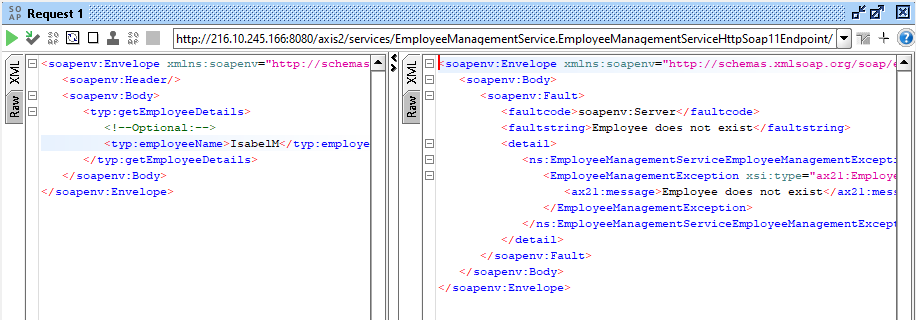
* Revisar Consultar empleado agregado sin data en el name



* Validar eliminar empleado existente.

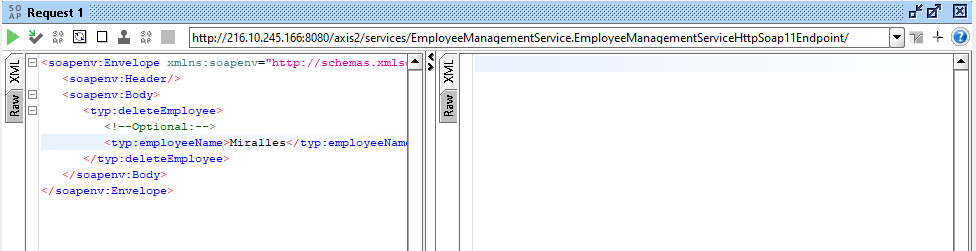


El servicio no muestra mensaje de salida, pero al usar la consulta Empleados, demuestra que ha sido eliminado el empleado.



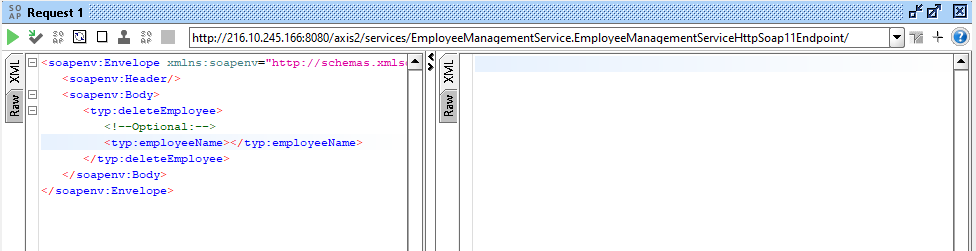
* Validar eliminar empleado no existente.

En la prueba anterior demostramos que el empleado Miralles no existe.



No muestra información de salida. Para validar el comportamiento

* Revisar eliminar empleado sin data.



No muestra información de salida, para validar el comportamiento.