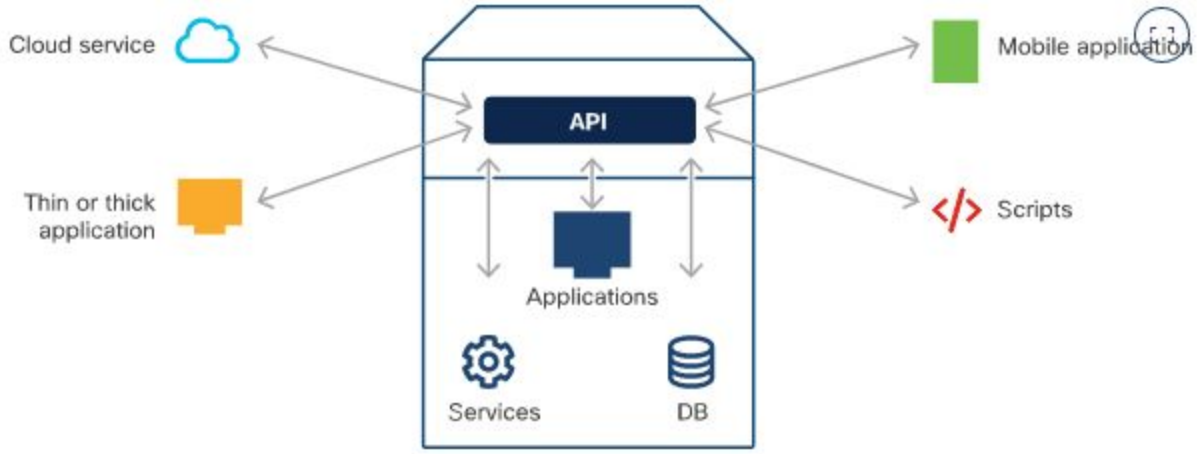
Que son API’s

Una API puede usar interacciones web o protocolos de comunicación, Y también puede usar sus propios protocolos standard.

Como parte de este proceso, la API también determina qué tipo de data, servicio, y funcionalidad la aplicación mostrará a terceras partes; si no está expuesto por la API, No está disponible(asumiendo que la seguridad de esta este bien configurada) Con API’s las aplicaciones pueden controlar lo que exponen de una forma segura.

Ejemplos de diferentes tipos de integraciones API.



Podríamos pensar de un API como un botón en nuestro vehículo. Cuando uno presiona o gira la llave, Lo que vemos es que el carro enciende.A uno no le importa en pensar las conexiones eléctricas que se combinan con las emisiones de gasolina para que los pistones se muevan. Todo lo que sabemos es que queremos que el carro encienda, Cuando presionemos o giremos nuestra llave. Si presionamos un botón distinto como el de la radio por ejemplo, El carro no encendera dado que el carro y su diseño nunca definieron que el botón del radio encienda el motor. Así mismo el motor del carro ejecuta a cabo más labores las cuales no son necesarias para el usuario a saber ni están vinculadas a un botón en el tablero dado que no es algo que el usuario necesite.

Porque usar API's?

API's son construidas usualmente para otras aplicaciones, De ahí viene su definición API(Application Programming Interface). Puede ser usadas por humanos para poder interactuar con aplicaciones de una forma manual.

Aca veremos sus usos más frecuentes:

- **Automatizacion**: Construir un script que lleve a cabo labores cotidianas que son repetitivas. Ejemplo: un gerente que debe tomar información de las horas laboradas por sus empleados para poder contabilizar cuánto se les debe pagar. Si se crea un script que haga uso de la API del portal de empleados y por medio de un API call podamos obtener la información y hacer el conteo de horas.

- **Integración de Data:** Una aplicación puede consumir o reaccionar a data que es proveída por otra aplicación. Ejemplo: muchos sitios web hace uso de sus servicios de pago mediante REST API's. El usuario auténtica con el servicio de pago, La página recibe su pago sin tener intervención directa con la data de la tarjeta de crédito del cliente.

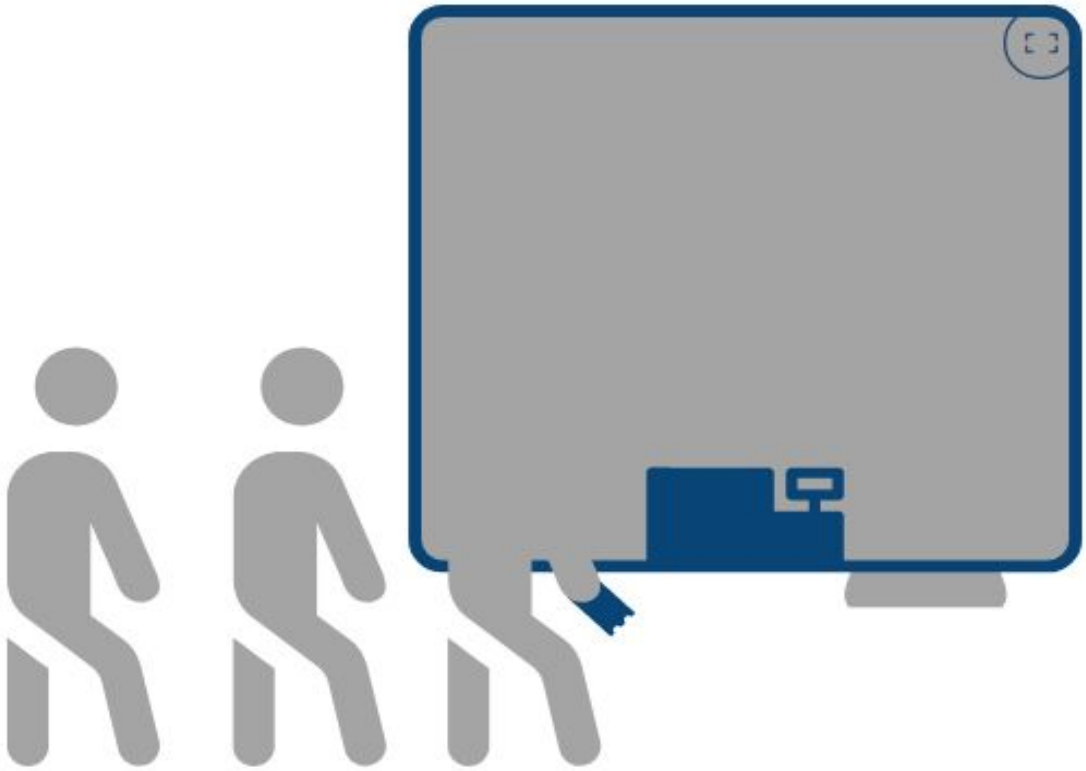
- **Funcionalidad**: Una aplicación puede integrar la funcionalidad de otra aplicación dentro de la suya. Ejemplo: Muchos servicios en línea como lo es Yelp y Uber, Intercambian data con google maps para poder crear y optimizar rutas de viaje. Ellos también incrustan funcionalidades de google maps en su propia aplicación para poder presentar al usuario la ruta en tiempo real.

API se han vuelto más reconocidas hoy en día dado que la industria del desarrollo software ha crecido exponencialmente en los últimos 10 años y con ello la demanda por integración de servicios y aplicaciones en conjunto.

Diseños y estilos de API

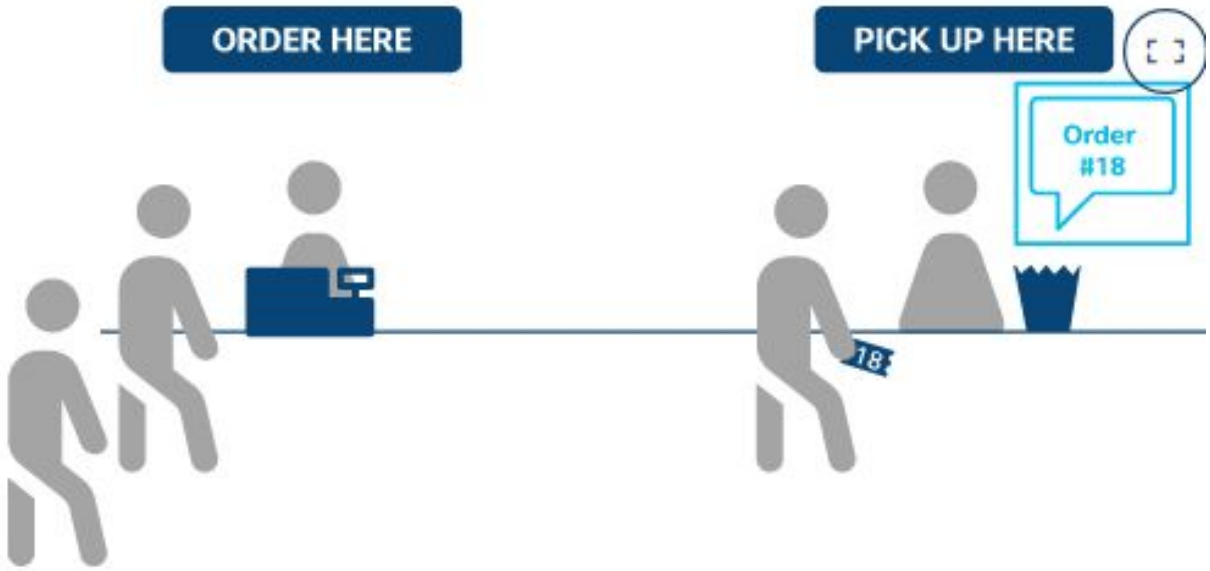
API’s pueden ser entregadas de dos formas: Sincronicas y Asincronicas. Ustedes deberan saber la diferencia entre ellas, Dado que la aplicación que esté consumiendo una API estará manejando su respuesta de forma diferente esto dependiendo del diseño API. Cada diseño tiene su propio propósito, Pero también tiene su propio conjunto de complejidades en su interacción Cliente/Servidor. Un producto conjunto de API’s puede consistir de ambos diseños, Donde cada diseño API es independiente de los demás. Para buenas prácticas la lógica de su diseño debe ser consistente.

API’s sincronicas



Tickets son vendidos y entregados en el orden en que van llegando las personas. Las API’s sincronicas usan esta metodología.

API’s Asincrónica



Con una API asincrónica la respuesta muestra que la petición fue aceptada pero la respuesta no contiene ningún tipo de data aun. El servidor procesa la información que puede llevar tiempo y luego manda una notificación con la Data requerida.

Representational State Transfer

REST es un estilo arquitectónico su autor es Rok Thomas Fielding. Lo dio a conocer en el capítulo 5 de su disertación doctoral. “Architectural Styles and the Design of Network-based Software” En su disertación el establecio 6 puntos aplicados a elementos de esta arquitectura.

1) Cliente - Servidor
2) Stateless
3) Cache
4) Interface Uniforme
5) Sistema de capas
6) Código-en-demanda

Cuanto estos puntos son aplicados a cualquier protocolo, Se convierte en un diseño de arquitectura RESTful.

REST Web API's

Un servicio API(REST API) es una interfaz programable que se comunica a través de el protocolo HTTP mientras se adhiere a los principios de arquitectura REST.

Dado que REST API's se comunican mediante el protocolo HTTP, estos usan los mismos conceptos del protocolo HTTP:

- HTTP requests/responses
- HTTP verbs
- HTTP status codes
- HTTP headers/body

Que son REST API's Requests?

Estos requests son esencialmente peticiones HTTP que siguen los principios de una arquitectura REST. Habilitan a la aplicación(cliente) a pedir a un servidor a hacer una función. Dado que es una API. Sus funciones son predefinidas por el servidor y deben seguir las especificaciones siguientes:

- Uniform Resource Identifier(URI)
- HTTP method/metodo
- Header
- Body

Uniform Resource Identifier(URI)
Se refiere tambien a el termino Uniform Resource Locator(URL). Identifica qué recursos el cliente quiere manipular. Una petición REST debe identificar el recurso deseado.

El URI utiliza el mismo formato que un URL de tu navegador Web. La sintaxis que utiliza es siguiendo en orden los siguientes componentes:

- Scheme
- Authority
- Path
- Query

HTTP Method: REST API's usan métodos HTTP, También conocidos como HTTP verbs, una forma de saber que acción lleva la petición a un determinado recurso es saber su método HTTP.

<u>HTTP Method</u>	<u>Action</u>	<u>Description</u>
POST	Create	Create a new object or resource.
GET	Read	Retrieve resource details from the system.
PUT	Update	Replace or update an existing resource.
PATCH	Partial Update	Update some details from an existing resource.
DELETE	Delete	Remove a resource from the system.

REST API Responses

Respuestas HTTP comunican el resultado de la petición HTTP. La respuesta puede contener la data que fue requerida, El servidor que recibe la petición informa sus resultados, O podria informar un error en la petición de la data.

Respuestas REST API son similares a los request, Estas toman tres factores como factores clave:

- HTTP status
- Header
- Body

HTTP Status

<u>HTTP Status Code</u>	<u>Status Message</u>	<u>Description</u>
200	OK	Request was successfully and typically contains a payload (body)
201	Created	Request was fulfilled and the requested resource was created
202	Accepted	Request has been accepted for processing and is in process
400	Bad Request	Request will not be processed due to an error with the request
401	Unauthorized	Request does not have valid authentication credentials to perform the request
403	Forbidden	Request was understood but has been rejected by the server
404	Not Found	Request cannot be fulfilled because the resource path of the request was not found on the server
500	Internal Server Error	Request cannot be fulfilled due to a server error
503	Service Unavailable	Request cannot be fulfilled because currently the server cannot handle the request

Header

Response header

<u>Key</u>	<u>Example Value</u>	<u>Description</u>
Set-Cookie	JSESSIONID=30A9DN810FQ42BB; Path=/	Used to send cookies from the server
Cache-Control	Cache-Control: max-age=3600, public	Specify directives which MUST be obeyed by all caching mechanisms

Entity headers

<u>Key</u>	<u>Example Value</u>	<u>Description</u>
Content-Type	application/json	Specify the format of the data in the body

Listo! Ya cubrimos las bases fundamentales para poder iniciar nuestra primera API.

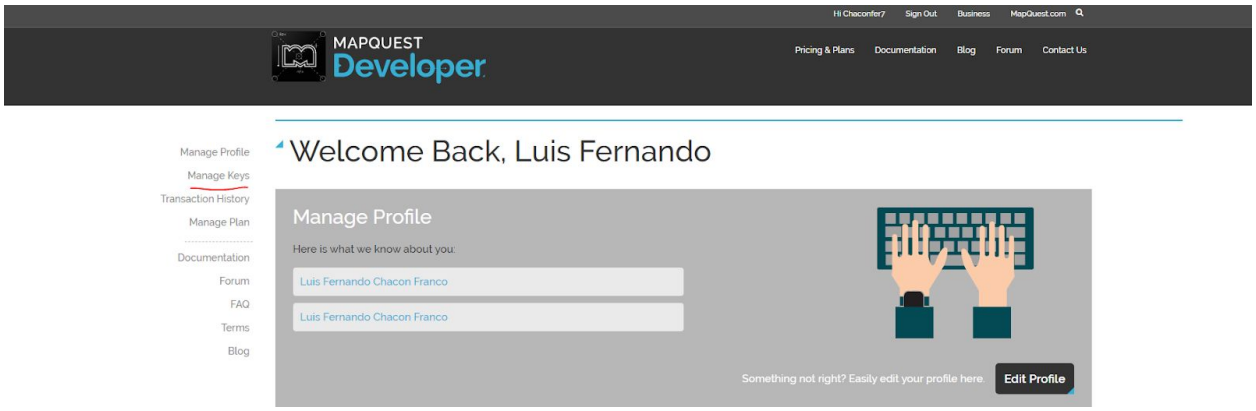
Mi primer API con Mapquest

Los objetivos de este laboratorio son el poder ayudar al estudiante a conocer como hacer API calls y entender cómo usar la documentación API de la aplicación que usaremos. REST API's usan la especificacion openAPI. Definiremos nuestra aplicación API que en base a las direcciones que brindemos o que obtenga podrá hacer el cálculo de distancia de un lugar a otro así mismo nos mostrará un STATUS CODE donde podamos poder si la operación fue exitosa o no.

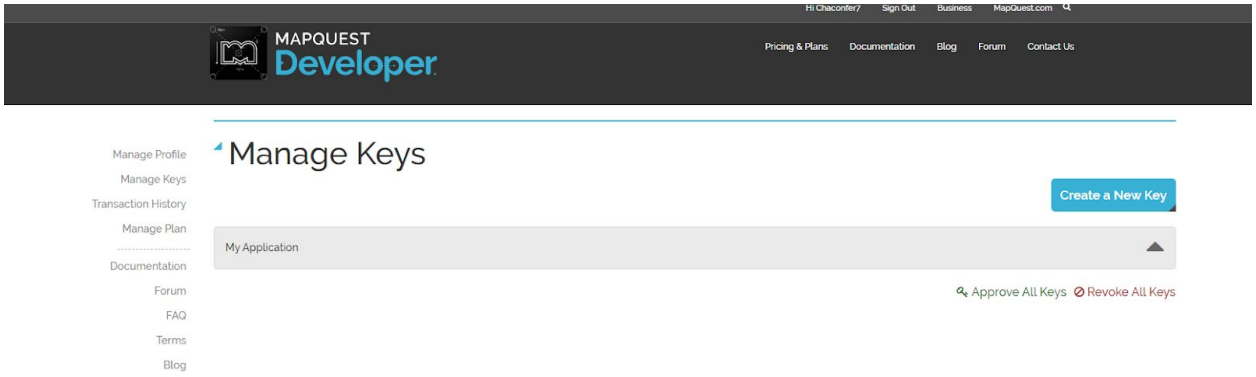
Paso 1: Antes de poder construir nuestra aplicación , deberemos conseguir para nuestra API una llave.

- Vamos a: https://developer.mapquest.com/
- Damos click en la opción **Sign Up** en la página.
- Llenamos la forma y en la opción **Company** ingresamos: **Cisco Networking Academy Student.**

Una vez que damos click a Sign Me up nos redireccionará a la página **Manage Keys**.



Una vez nos mande a esa opción daremos click a MyApplication.



Consumer Key es la información que necesitamos para nuestro laboratorio.

Construir la aplicación de direcciones con Mapquest

En esta parte crearemos un script en python que mande una peticion URL a la API de Mapquest. Luego podrás probar tu API haciendo API calls. Durante este laboratorio construiremos nuestro script en partes, Guardándolo con un nuevo nombre cada vez que ejecutemos un siguiente paso. Esto nos Ayudará a poder tener una serie de pasos y revertir algún cambio si en algún punto tuviéramos problemas.

Paso 2: Abrimos VS Code o nuestro IDLE de python. Luego abrimos la carpeta Repo que se creó para este curso y crearemos un archivo nuevo con nuestro nombre-API.py y luego comenzaremos a crear nuestro script.

Importar módulos para la aplicación

Para poder empezar nuestro script para analizar data JSON, debemos importar dos módulos de la librería de python los cuales son requests y urllib.parse. El módulo request nos ayuda a poder utilizar funciones para obtener data JSON desde una URL y la librería urllib.parse nos da una variedad de funciones que nos habilitan el analizar y manipular la data JSON que recibiremos desde una petición(request) a un URL.

```
1  import urllib.parse
2  import requests
```

Creacion de URL request a MapQuest API.

El primer paso en crear nuestra petición API es construir nuestra URL que nuestro script(aplicación) usará para hacer su API call inicial. La URL es una combinación de las siguientes variables:

- main_api: la URL principal que estamos accediendo
- orig: el parámetro que indica el punto de origen
- dest: el parámetro que indica el punto de destino
- key: la llave API que obtuvimos del sitio del desarrollador de esta app.

```
4  main_api = "https://www.mapquestapi.com/directions/v2/route?"
5  orig = "Washington, D.C."
6  dest = "Baltimore, Md"
7  key = "YOUR_API_KEY"
```

Recordemos que todas las Llaves API serán distintas por lo tanto debemos usar la que nos genero el sitio developer de MapQuest.

Combinaremos las cuatro variables que creamos(main_api, orig, dest, key) para hacer el formato de petición URL. Usaremos el método urlencode para hacer el formato correctamente de la direccion. Esta funcion construye los parametros como parte de la URL y cualquier posible caracter especial en la direccion lo convierte a caracteres aceptados.(Ejemplo: “+” y , convertido a “%2C”).

```
 9   url = main_api + urllib.parse.urlencode({"key":key, "from":orig
10
```

Ahora Crearemos la variable que contenga la respuesta de nuestra petición URL y que imprima en pantalla la data JSON. la variable json_data contiene un diccionario de python en representación de la respuesta en formato JSON de el método GET de el modulo requests(request.get). Así también ingresamos la función print() con el contenido variable json_data para verificar el éxito de la operación.

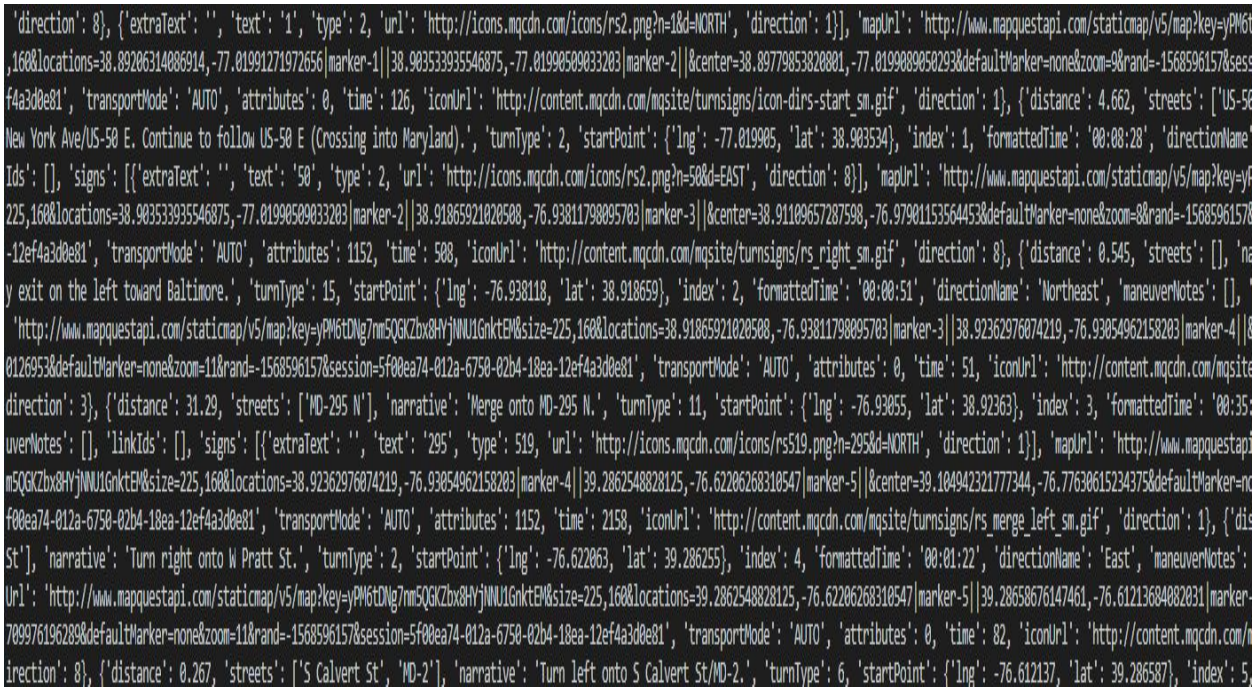
```
11   json_data = requests.get(url).json()
12   print(json_data) #mostrar respuesta de peticion
```

El resultado final se verá de la siguiente manera:

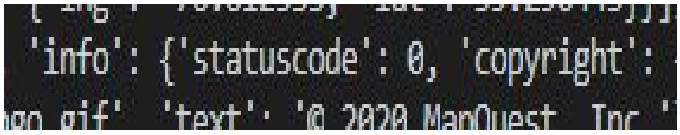
```
1   import urllib.parse
2   import requests
3
4   main_api = "https://www.mapquestapi.com/directions/v2/route?"
5   orig = "Washington, D.C."
6   dest = "Baltimore, Md"
7   key = "yPM6tDNg7nm5QGKZbx8HYjNNU1GnktEM" #USAR TU LLAVE OBTENIDA de el sitio developer de MapQuest
8
9   url = main_api + urllib.parse.urlencode({"key":key, "from":orig, "to":dest})
10
11  json_data = requests.get(url).json()
12  print(json_data) #mostrar respuesta de peticion
```

Probando la petición URL

Guardaremos nuestro script con nuestro nombre nombre-API-respuesta.py y luego lo haremos correr para ver el resultado.



El resultado obtenido debe ser parecido al que veremos arriba o al que actualmente estamos efectuando en el laboratorio, Debemos recordar que esta respuesta es un diccionario de python con dos valores **key/value**.
Uno de los valores que son parte de la respuesta es el Status code de la llamada API.



Haremos uso de esta información de ahora en adelante en nuestro laboratorio.

Mostrar el URL y ver el Status code de nuestra peticion JSON.
Ahora que ya sabemos que la respuesta a nuestra petición funciona y regresa la información deseada vamos a eliminar la función print que contiene dentro la variable json_data, En su lugar personalizamos la data que recibimos de la siguiente manera.

a) Mostraremos la URL para que el usuario pueda ver la petición exacta hecha por la aplicación.

```
11  json_data = requests.get(url).json()
12  print("URL: " + (url))
13  json_status = json_data["info"]["statuscode"]
14
```

b) Crearemos una variable llamada json_status la cual nos ayudará a filtrar el contenido dentro de json_data, Recordemos que ahora el contenido es un diccionario de python por lo tanto sigue las convenciones de este lenguaje.
c) Añadiremos una condicional para poder mostrar el Status code de la API call en vigencia.

```
15 if json_status == 0:
16     print("API Status: " + str(json_status) + " = A successful route call.\n")
```

Más tarde en nuestro laboratorio crearemos una condicional que nos pueda ayudar a ver los diferentes valores Status code.

El resultado si la operación fue exitosa y no tenemos errores de sintaxis se verá de la siguiente manera:

```
URL: https://www.mapquestapi.com/directions/v2/route?key=yPM6tDNg7nm5QGKZbx8HYjNNU1GnktEM&from=Washington%2C+D.C.&to=Baltimo
re%2C+Md
API Status: 0 = A successful route call.
```

Procederemos a personalizar aún más nuestro script y habilitar user input.

```
while True:
    orig = input("Add starting location: " )
    if orig == "quit" or orig == "q":
        break
    dest = input("add your destination: ")
    if dest == "quit" or dest == "q":
        break
    key = "yPM6tDNg7nm5QGKZbx8HYjNNU1GnktEM" #USAR TU LLAVE OBTENIDA de el sitio developer de

    url = main_api + urllib.parse.urlencode({"key":key, "from":orig, "to":dest})

    json_data = requests.get(url).json()
    print("URL: " + (url))
    json_status = json_data["info"]["statuscode"]

    if json_status == 0:
        print("API Status: " + str(json_status) + " = A successful route call.\n")
```

Agregaremos un while True para poder agregar una secuencia indefinida de inputs por parte del usuario. Añadiremos así mismo una condicional dentro de este loop el cual nos da la opción de poder terminar su ejecución si presionamos “q” o “quit”. Ya podemos guardar nuestro script y probar la funcionalidad de este.

```
Add starting location: Washington
add your destination: Baltimore
URL: https://www.mapquestapi.com/directions/v2/route?key=yPM6tDNg7nm5QGKZbx8HYjNNU1GnktEM&from=Washington&to=Baltimore
API Status: 0 = A successful route call.

Add starting location:
```

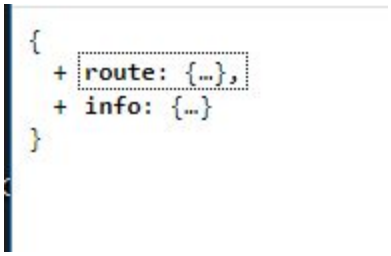
Ahora instalaremos una extensión en Chrome para poder visualizar de mejor manera nuestro formato JSON

Hay muchas otras extensiones que nos ayudan a poder ver esta información pero para los fines de este laboratorio usaremos JSONview.

Luego volveremos a nuestro script en visual code o nuestro python IDLE y copiamos el URL que nos muestra de output cuando ejecutamos nuestro script.

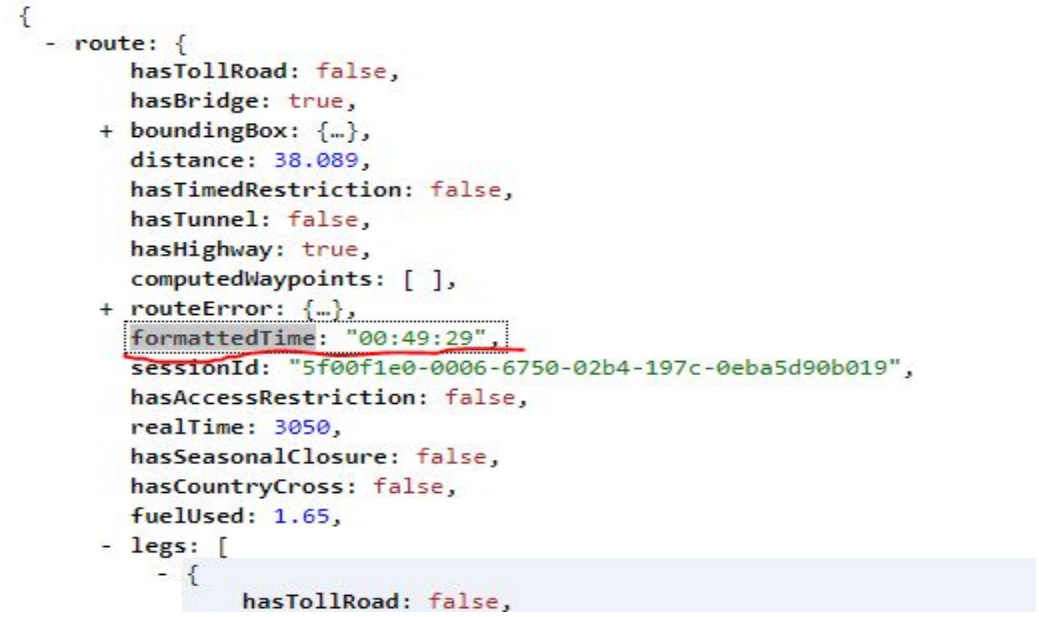
Pegamos este link en nuestro navegador y la extensión JSONview nos ayudará a poder visualizar la información más comprensiblemente.

Podemos dar click en el símbolo de menos para poder contraer la información y veremos que nos mostrará dos diccionarios principales(root dictionaries)



Si damos click en el símbolo “+” veremos Data que muestra valores para el uso de tiempo, gasolina, Túneles, carreteras y demás información que pueda mostrar la aplicación. Esta información puede ser analizada y manipulada en nuestro script. Recordemos que la respuesta a nuestra petición es un diccionario de python y por sintaxis sabemos que un diccionario puede almacenar valores **key/value** como listas o diccionarios incrustados.

Mostrando información de duración, distancia y Gasolina usada.
Regresaremos a nuestro navegador y buscaremos el tiempo que nos tomará llegar de un punto a otro, La llave asignada a este valor se llama **formattedtime** la cual está dentro del diccionario **route**.



Ya sabemos cual es la llave y sabemos su valor, Ahora podemos añadir esta información para ser mostrada por nuestro script.

```
21      if json_status == 0:
22          print("API Status: " + str(json_status) + " = A successful route call.\n")
23          print("Trip Duration: " + (json_data["route"]["formattedTime"]))
24
```

Podremos ver el siguiente resultado si nuestra operación en filtrar valores del diccionario fueron exitosos:

```
Add starting location: Washington
add your destination: Baltimore
URL: https://www.mapquestapi.com/directions/v2/route?key=yPM6tDNg7nm5QGKZbx8HYjNNU1GnktEM&from=Washington&to=Baltimore
API Status: 0 = A successful route call.

Trip Duration: 00:49:29
Add starting location:
```

Podemos ver como ahora nos muestra la información de duración del viaje. Ahora procederemos con la distancia y gasolina a usar.

```
if json_status == 0:
    print("API Status: " + str(json_status) + " = A successful route call.\n")
    print("Trip Duration: " + (json_data["route"]["formattedTime"]))
    print("Miles: " + str(json_data["route"]["distance"]))
    print("Fuel Used (Gal): " + str(json_data["route"]["fuelUsed"]))
```

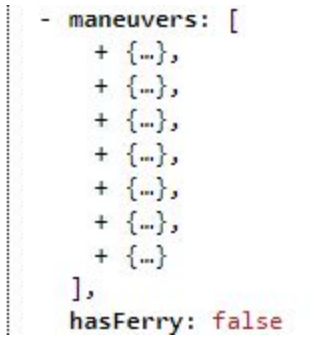
Nuestro aplicación nos mostrara la información siguiente, Tomar en cuenta el resultado varía dependiendo de los parámetros que ingresemos.

```
Add starting location: Washington
add your destination: Utah
URL: https://www.mapquestapi.com/directions/v2/route?key=yPM6tDNg7nm5QGKZbx8HYjNNU1GnktEM&from=Washington&to=Utah
API Status: 0 = A successful route call.

Trip Duration: 31:45:03
Miles: 2163.79
Fuel Used (Gal): 108.4
Add starting location:
```

Personalizamos aún más nuestra aplicación manipulando la información de nuestro diccionario de python

Veremos qué Route es un diccionario y legs es uno de sus contenidos la llave legs contiene una lista incrustada y dentro de esta lista podemos ver un diccionario, Por sintaxis sabemos que las listas son objetos ordenados por un index en este caso legs contiene solamente un index el cual es 0, si desplegamos la información de este diccionario veremos que uno de sus key/value es maneuvers, veremos como el contenido de maneuvers tiene como valor una lista y dentro 7 diccionarios incrustados.



Ahora que entendemos la estructura de la data que tenemos podemos hacer uso de un for loop para poder filtrar la información narrativa la cual nos dará direcciones de el como llegar a nuestro destino.

El resultado final se verá de la siguiente manera:

```
21      if json_status == 0:
22          print("API Status: " + str(json_status) + " = A successful route call.\n")
23          print("Trip Duration: " + (json_data["route"]["formattedTime"]))
24          print("Miles: " + str(json_data["route"]["distance"]))
25          print("Fuel Used (Gal): " + str(json_data["route"]["fuelUsed"]))
26          for each in json_data["route"]["legs"][0]["maneuvers"]:
27              print((each["narrative"]) + " (" + str("{:.2f}".format((each["distance"])) + " km)"))
28
```

La variable each contendra informacion de cada contenido dentro del diccionario route que contiene el key/value legs que así mismo en su lista en index 0 contiene la key/llave maneuvers con su valor narrativa.

Guardaremos nuestra aplicación y luego ejecutaremos nuestro código, el resultado será parecido al siguiente:

```
Miles: 1786.7102
Fuel Used (Gal): 90.41
Start out going east on Nelson Rd toward Shumway Spring Rd. (2.03 km)
Turn right onto N Old Highway 89/US-89 S. Continue to follow US-89 S. (19.93 km)
Turn left onto N Main St/US-89 S. Continue to follow US-89 S. (16.05 km)
Stay straight to go onto S State St/US-50 E. (0.08 km)
Merge onto I-70 E via the ramp on the left toward Green River (Portions toll) (Passing through Colorado, then crossing into Kansas).
Merge onto I-670 E via EXIT 421B on the left (Crossing into Missouri). (4.08 km)
I-670 E becomes I-70 E. (229.33 km)
Merge onto I-270 N via EXIT 232AB toward Chicago (Crossing into Illinois). (30.96 km)
I-270 N becomes I-70 E (Crossing into Indiana). (222.95 km)
Keep right to take I-70 E via EXIT 112A toward Columbus OH (Crossing into Ohio). (169.71 km)
Merge onto I-670 E via EXIT 96 on the left toward Airport. (4.28 km)
Merge onto I-71 N via EXIT 5A/5B toward Cleveland. (31.20 km)
Take the OH-61 exit, EXIT 140, toward Cardington/Mt Gilead. (0.27 km)
Turn left onto State Route 61/OH-61. (1.93 km)
Turn right onto State Route 229/OH-229. Continue to follow OH-229. (0.84 km)
Turn left onto S Main St/OH-229/County Hwy-26. (0.16 km)
Turn right onto E Noble St/OH-229. Continue to follow OH-229. (5.46 km)
Turn left onto State Route 314/OH-314. (1.08 km)
Welcome to OH. (0.00 km)
Add starting location:
```

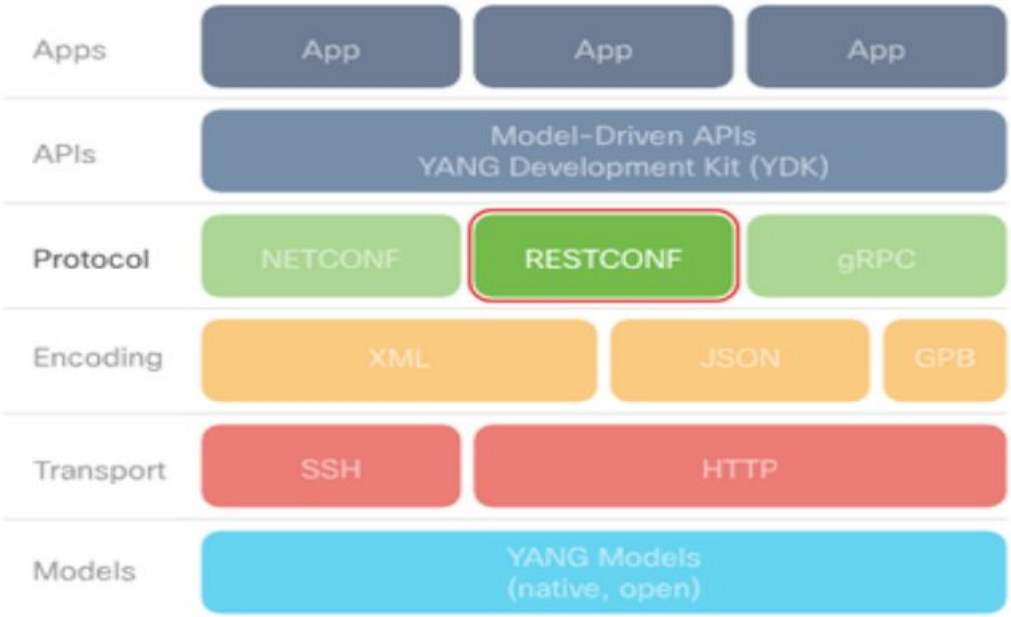
Por último agregaremos la funcionalidad de poder mostrarnos Status code de error en caso nuestro script falle por algún motivo. Si vemos de nuevo en el diccionario route veremos un diccionario dentro llamado routeError el cual contiene errorCode2. Cuando nuestro código se ejecuta dado que nuestra condicional sólo da instrucciones para cuando nuestro output sea 0 deberemos ingresar el siguiente código:

```
15      url = main_api + urllib.parse.urlencode({"key":key, "from":orig, "to":dest})
16
17      json_data = requests.get(url).json()
18      print("URL: " + (url))
19      json_status = json_data["info"]["statuscode"]
20
21      if json_status == 0:
22          print("API Status: " + str(json_status) + " = A successful route call.\n")
23          print("Trip Duration: " + (json_data["route"]["formattedTime"]))
24          print("Miles: " + str(json_data["route"]["distance"]))
25          print("Fuel Used (Gal): " + str(json_data["route"]["fuelUsed"]))
26          for each in json_data["route"]["legs"][0]["maneuvers"]:
27              print((each["narrative"]) + " (" + str("{:.2f}".format((each["distance"])) + " km)"))
28          print("=============================================\n")
29      elif json_status == 402:
30          print("**********************************************")
31          print("Status Code: " + str(json_status) + "; Invalid user inputs for one or both locations.")
32          print("**********************************************\n")
33      elif json_status == 611:
34          print("**********************************************")
35          print("Status Code: " + str(json_status) + "; Missing an entry for one or both locations.")
36          print("**********************************************\n")
37      else:
38          print("************************************************************************")
39          print("For Staus Code: " + str(json_status) + "; Refer to:")
40          print("https://developer.mapquest.com/documentation/directions-api/status-codes")
41          print("************************************************************************\n")
```

Podremos ver como el script nos mostrará la información de error en caso fuese necesario así mismo como cualquier otro Status code de error que podamos tener.

RESTCONF

RESTCONF es un protocolo basado en HTTP, RESTCONF se define en el RFC 8040 este es un protocolo y un mecanismo para configuraciones REST, Similar a NETCONF, este usa una base de modelos y comandos definidos por el protocolo NETCONF, Encapsulando esta información en mensajes HTTP. Como con NETCONF, El lenguaje YANG es usado para definir la sintaxis de la estructura.

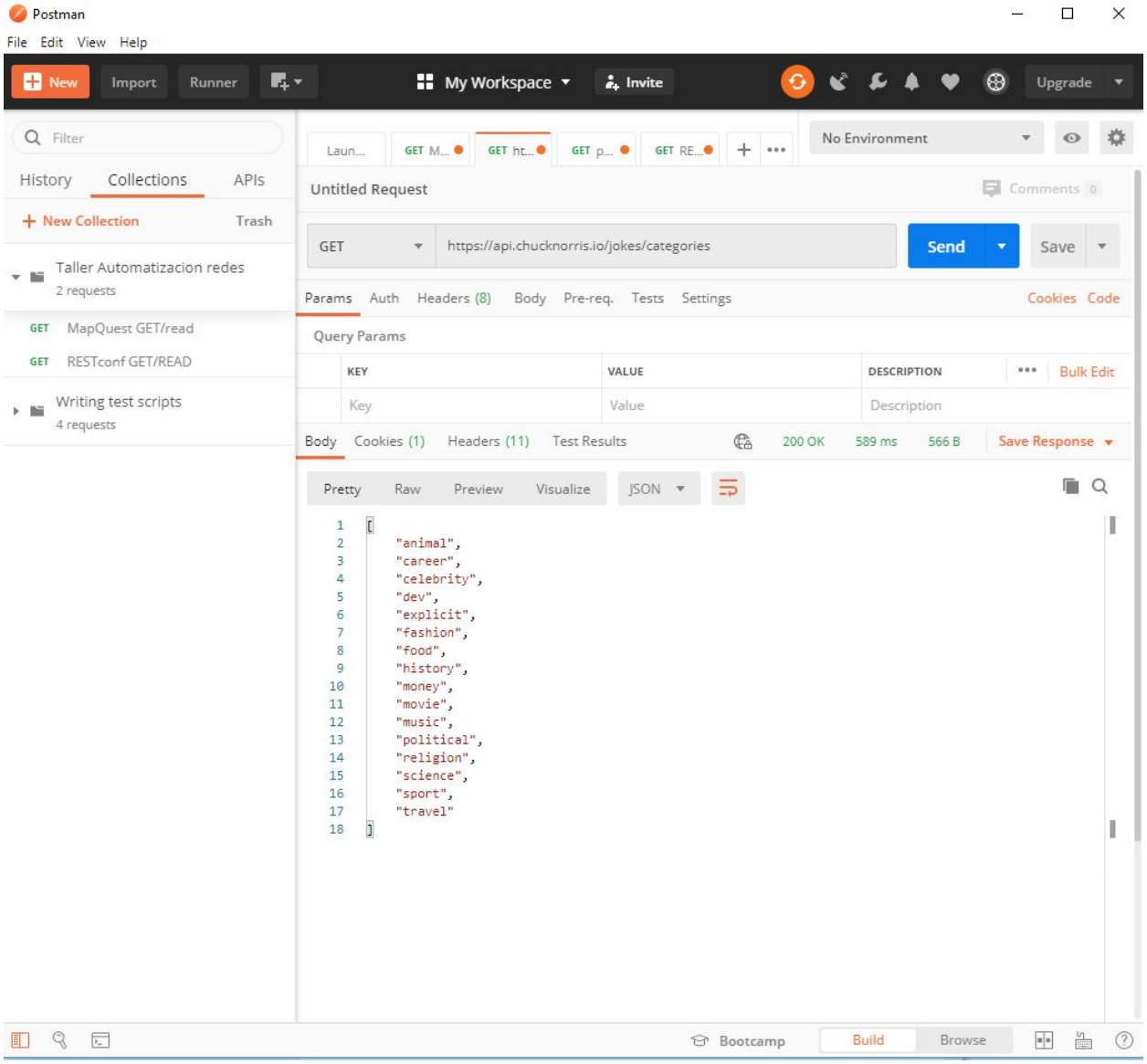


RESTCONF usa data estructurada(XML, JSON) y YANG que provee API's estilo REST, habilitando acceso programable a equipos. Comandos HTTP como GET, POTS,PUT,PATCH y DELETE son redireccionados a una RESTCONF API para acceder a los recursos de data representados por modelos YANG.

Postman

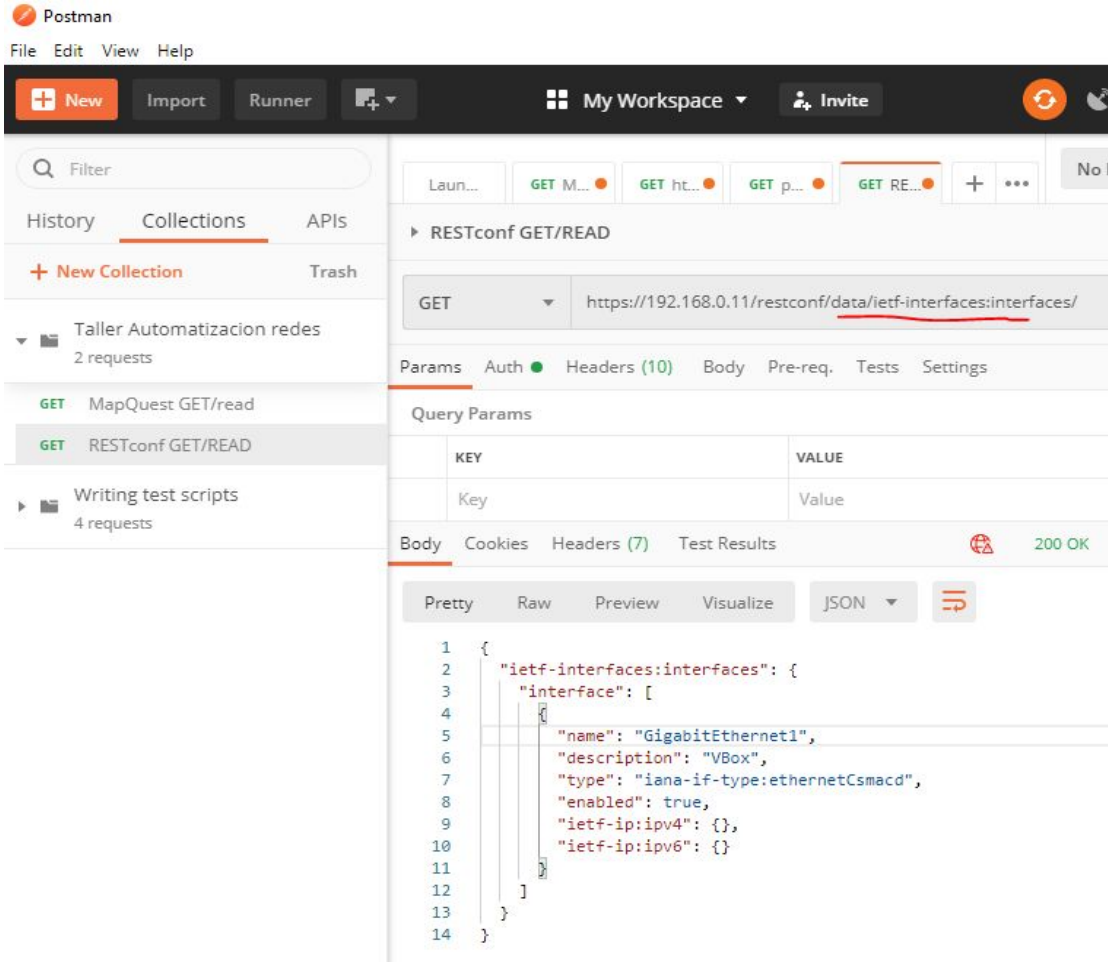
Es una de las herramientas más populares para hacer pruebas eficaces a nuestra API con postman podremos hacer uso de los métodos GET, POST, PUT y PATCH en este laboratorio veremos los métodos GET y PUT.

GET



La operación GET es muy sencilla, Abrimos un tab nuevo y seleccionaremos el método GET y también copiaremos la dirección endpoint API.

A continuación veremos otro ejemplo esta vez usando el protocolo RESTCONF



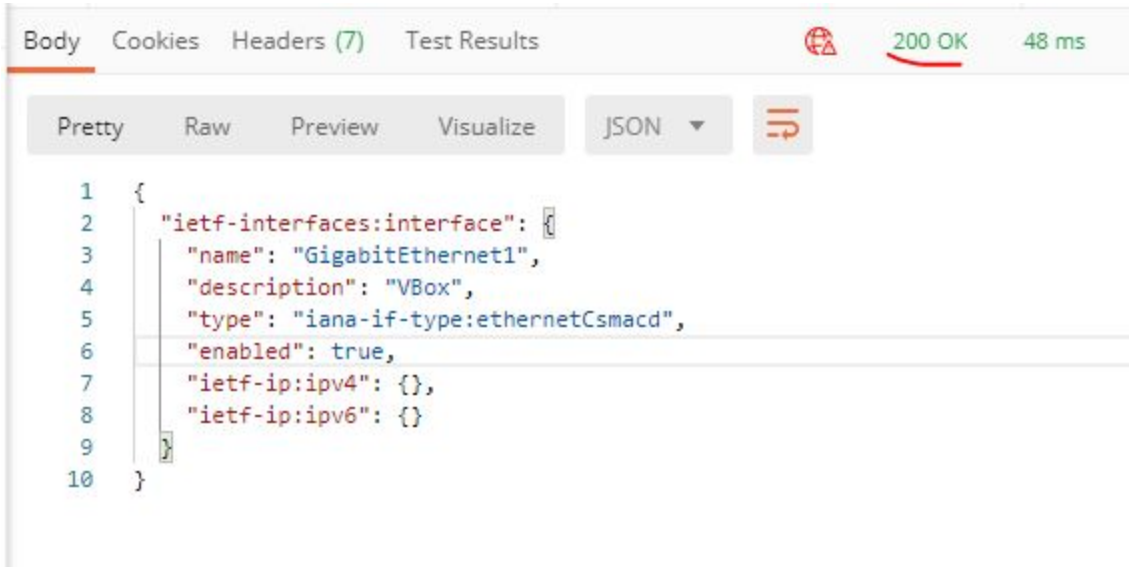
Como podemos ver hacemos uso de modelos de data YANG, estamos haciendo una petición a el módulo iet-interfaces de su contenedor interfaces.

Podemos hacer uso de un leaf del contenedor interfaces llamado interface-type

```
leaf type {
  type identityref {
    base interface-type;
  }
  mandatory true;
```

Aca podemos ver cómo se vería el GET request en postman

La respuesta de esta petición nos mostrará la información requerida por medio del modelo de data YANG.

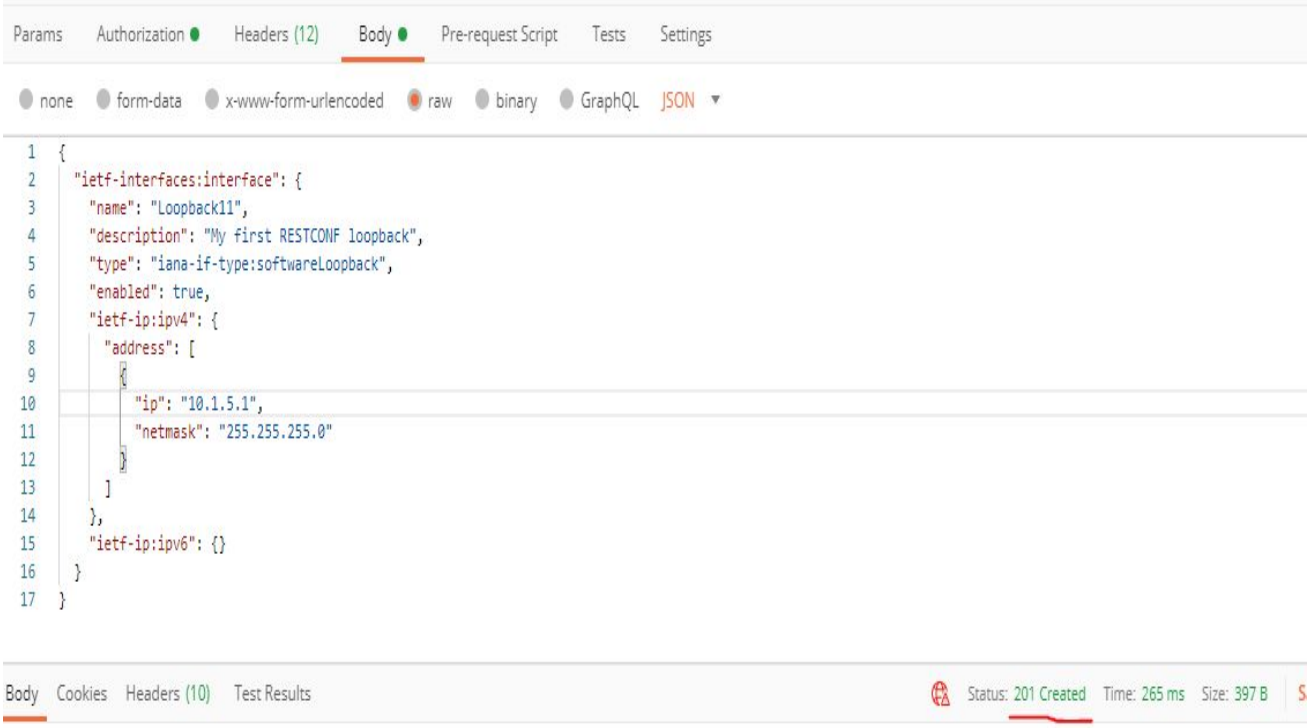


PUT

Ahora configuraremos una interfaz loopback a nuestro CSR 1000 V para esto cambiaremos el método de GET a PUT

La variante será el cambio de método y añadiremos la loopback que vamos a crear en esta caso será Loopback 11, Luego iremos a body y seleccionaremos la opción raw en este espacio ingresamos la información que deseamos ser configurada en nuestro router.

Luego de haber habilitado las opciones requeridas daremos click en **send** y podremos ver lo siguiente.



Tendremos un Status code 201 que nos indica que el recurso fue creado exitosamente.

```
login as: cisco
Keyboard-interactive authentication prompts from server:
Password:
End of keyboard-interactive prompts from server

*                                       *
**                                     **
***                                   ***
***  Cisco Networking Academy         ***
***                                   ***
***  This software is provided for    ***
***   Educational Purposes            ***
***    Only in Networking Academies   ***
***                                   ***
**                                     **
*                                       *


CSR1kv#show ip int brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.0.11    YES DHCP   up                    up
Loopback11             10.1.5.1        YES other  up                    up
CSR1kv#
```

Si deseamos verificar con putty veremos la nueva interfaz.

GET request with RESTCONF

Ahora crearemos un script con python que nos ayudará a poder efectuar peticiones GET a nuestro CSR1kv, podremos usar VS Code o python IDLE para esta operación. Recordemos que RESTCONF hace uso de la arquitectura REST por lo tanto encontraremos mucha similitud en su estructura a una REST API.

1) Importamos los módulos necesarios para deshabilitar SSL certificate Warnings.

```
1   import json
2   import requests
3   requests.packages.urllib3.disable_warnings()
```

El módulo json incluye métodos para convertir data JSON a objetos en python y viceversa. El módulo requests tiene métodos que nos permite mandar peticiones REST a una URL.

2) Crear variables que componen la petición, crearemos la variable api_url y le asignamos la URL que tendrá acceso ala información de interfaces de nuestro CSR1Kv

```
api_url = "https://192.168.0.11/restconf/data/ietf-interfaces:interfaces/"
```

Crearemos así mismo una variable diccionario que se llamara headers que contendrá las llaves **Accept** y **Content-type** y asignaremos a ambos el valor **application/yang-data+json**.

```
7   headers = { "Accept": "application/yang-data+json",
8    "Content-type":"application/yang-data+json"
9    }
```

Para terminar esta parte crearemos un tuple con el nombre basicauth que contendrá dos llaves las cuales necesitamos para autenticar.

```
11   basicauth = ("cisco", "cisco123!")
```

3) Crearemos una variable para mandar la petición y almacenar la respuesta JSON, Usaremos las variables creadas anteriormente como parámetros para el método requests.get() este método enviara una peticion GET(http) al servidor API RESTCONF en nuestro CSR1Kv. Luego asignaremos el resultado a la variable que llamaremos **resp**. La variable contendrá la respuesta JSON de nuestro API. Si la petición fue exitosa, JSON contendrá la respuesta de YANG data.

```
14   resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
15
```

El HTTP status code de respuesta si agregamos la función print(resp)

```
resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
print(resp)
```

Resultado:

```
<Response [200]>
```

Formatear y desplegar data JSON recibida de nuestro CSR1Kv

La respuesta que recibimos y que contiene la variable resp esta en formato JSON y no es compatible con python podemos verificar usando la función type()

```
resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
print(type(resp))
```

El resultado nos mostrará lo siguiente:

```
C:\Users\q669YSCG\Desktop\testpy>C:/
<class 'requests.models.Response'>
```

Crearemos una variable que llamaremos response_json y le asignaremos la variable resp y hacemos uso del método json() para poder la respuesta a una estructura de data en python.

```
14  resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False
15  response_json = resp.json()
16  print(type(response_json))
17
```

Haciendo uso de la función type() veremos que la información es ahora un diccionario y puede ser manipulado y filtrado según la sintaxis que ya conocemos del lenguaje.

```
C:\Users\q669YSCG\De
<class 'dict'>
```

El resultado sin el uso de la función type() será el siguiente:

```
C:\Users\q669YSCG\Desktop\testpy>C:/Python38/Python38-32/python.exe c
{'ietf-interfaces:interfaces': {'interface': [{'name': 'GigabitEtherr
rue, 'ietf-ip:ipv4': {}, 'ietf-ip:ipv6': {}}, {'name': 'Loopback2', '
ack', 'enabled': True, 'ietf-ip:ipv4': {'address': [{'ip': '10.2.10.1
escription': 'My first RESTCONF loopback', 'type': 'iana-if-type:soft
netmask': '255.255.255.0'}]}, 'ietf-ip:ipv6': {}}]}}
```

Para poder ver el resultado en forma similar a cuando lo vimos en postman usamos la función json.dumps() con el parámetro “indent”

```
resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
response_json = resp.json()
print(json.dumps(response_json, indent=4))
```

Resultado:

```
{
    "ietf-interfaces:interfaces": {
        "interface": [
            {
                "name": "GigabitEthernet1",
                "description": "VBox",
                "type": "iana-if-type:ethernetCsmacd",
                "enabled": true,
                "ietf-ip:ipv4": {},
                "ietf-ip:ipv6": {}
            },
            {
                "name": "Loopback2",
                "description": "My second RESTCONF loopback",
                "type": "iana-if-type:softwareLoopback",
                "enabled": true,
                "ietf-ip:ipv4": {
                    "address": [
                        {
                            "ip": "10.2.10.1",
                            "netmask": "255.255.255.0"
                        }
                    ]
                },
                "ietf-ip:ipv6": {}
            },
            {
                "name": "Loopback11",
                "description": "My first RESTCONF loopback",
                "type": "iana-if-type:softwareLoopback",
                "enabled": true,
```


PUT request with RESTCONF

En esta parte, Crearemos un script el cual nos ayude a mandar una petición PUT a nuestro CSR1Kv. Como lo hicimos con postman, crearemos una interfaz nueva.

1) Crearemos un archivo nuevo y importamos los módulos que se necesitan para deshabilitar SSL certificate Warnings.

```
1  import json
2  import requests
3  requests.packages.urllib3.disable_warnings()
4
```

2) Crearemos variables que componen la peticion, crearemos la variable api_url y asignaremos la URL que nos ayude a crear la interfaz Loopback99

```
api_url = "https://192.168.0.11/restconf/data/ietf-interfaces:interfaces/interface=Loopback99"
```

Crearemos así mismo una variable diccionario que se llamara headers que contendrá las llaves **Accept** y **Content-type** y asignaremos a ambos el valor **application/yang-data+json**.

```
headers = { "Accept": "application/yang-data+json",
 "Content-type":"application/yang-data+json"
 }
```

crearemos un tuple con el nombre basicauth que contendrá dos llaves las cuales necesitamos para autenticar.

```
basicauth = ("cisco", "cisco123!")
```

Para terminar esta parte crearemos una variable diccionario **yangConfig** que contendrá la data YANG que necesitaremos para crear loopback99. Podemos usar el mismo diccionario que usamos en postman sin embargo debemos editar el número de interfaz y su dirección, Así mismo recordemos de colocar T mayúscula para el key/value “enable”:True ya que valores booleanos deben ser especificados con T mayúscula para True

```
yangConfig = {
 "ietf-interfaces:interface": {
 "name": "Loopback99",
 "description": "My second RESTCONF loopback",
 "type": "iana-if-type:softwareLoopback",
 "enabled": True,
 "ietf-ip:ipv4": {
 "address": [
 {
 "ip": "10.2.1.1",
 "netmask": "255.255.255.0"
 }
 ]
 },
 "ietf-ip:ipv6": {}
 }
}
```

3) Crearemos una variable para mandar petición y almacenar la respuesta JSON, usaremos las variables previamente creadas como parametros de el método requests.put(), mandaremos una petición HTTP PUT a la API RESTCONT. Asignaremos el resultado de la operación a la variable **resp**. La variable contendrá la respuesta JSON de la API. si la petición fue exitosa, Obtendremos información de vuelta del modelo de data YANG.

```
32  resp = requests.put(api_url, data=json.dumps(yangConfig), auth=basicauth,
33  headers=headers, verify=False)
34
```

Por último agregaremos un par de condicionales para poder filtrar de mejor manera nuestra respuesta, Obtendremos uno de los Status code de If si la operación fue exitosa y obtendremos un código y mensaje de error si se detectó algún error.

```
32  resp = requests.put(api_url, data=json.dumps(yangConfig), auth=basicauth,
33  headers=headers, verify=False)
34
35  if(resp.status_code >= 200 and resp.status_code <= 299):
36      print("STATUS OK: {}".format(resp.status_code))
37  else:
38    print('Error. Status Code: {} \nError message:{}'.format(resp.status_code,resp.json()))
39
```

Si nuestra operación resulta exitosa al ejecutar nuestro script obtendremos el siguiente Status code:

```
C:\Users\q669YSCG\De
STATUS OK: 201
```

201 nos confirma que la operación tuvo éxito y la interfaz fue creada. Si confirmamos con putty veremos como la interfaz se encuentra configurada.

```
Loopback99             5.5.1.1         YES other  up
CSR1kv#
```