

CURL/THUNDER

Contenido

1 Introducción	2
2 Api Rest.....	2
Data fake	3
3 Bash	3
4 Curl	6
Content Type: Codificaciones en http.	7
Ejercicio 1:	7
Usando el método get de curl	8
Ejercicio 2:	9
Ejercicio 3:	10
Enviar datos.....	10
Practica 1:	14
5 Thunder Client.....	17
Referencias:.....	19

1 Introducción

Una URL (Uniform Resource Locator) no es más que una dirección que es dada a un recurso único en la Web. En teoría, cada URL válida apunta a un único recurso.

Curl significa cliente URL y es una herramienta de línea de comandos gratuita para transferir archivos con sintaxis de URL. Curl admite varios protocolos, incluidos certificados HTTP, FTP, SMB y SSL. Hay varios clientes Curl para Windows, Linux, macOS, Android e iOS, y ahora con el cliente ReqBin Online para la web. Los desarrolladores pueden utilizar la biblioteca libcurl para integrar Curl en sus aplicaciones C/C++, Java, PHP y Python. Curl se ha convertido en la herramienta principal para describir llamadas API en la documentación debido a su popularidad y facilidad de uso.

2 Api Rest.

Una API es una interfaz de programación de aplicaciones, es básicamente un conjunto de definiciones de variables, constantes, funciones y protocolos, por otro lado, REST (representational state transfer) deriva de "REpresentational State Transfer", que traducido vendría a ser "transferencia de representación de estado" permite estandarizar comunicaciones web. Básicamente es una interfaz, pero no es una interfaz de usuario, es una interfaz para los programadores de **front end**.

La arquitectura REST se basa en que el cliente envía **peticiones** para recuperar o modificar recursos, y el servidor responde con el resultado, que puede ser con los datos que hemos pedido o el estado de la petición (por ejemplo, un código de error, o un código **ok** indicando que todo fue bien).

Una **petición** REST está formada por:

- Un verbo HTTP que define la operación a realizar.
- Una cabecera o header que incluye información sobre la petición.
- Una ruta o path hacia un recurso.
- El cuerpo del mensaje o body con los datos de la petición.

Existen 4 verbos principales que forman parte de las peticiones HTTP para interactuar con una API REST, estos son:

- GET Para recuperar información sobre un recurso específico.
- POST Para crear un nuevo recurso.
- PUT Actualizar un recurso.
- DELETE Eliminar un recurso.

Cuando termina la operación de la API en el servidor, este nos enviará el resultado junto con una serie de datos para que podamos interpretar el éxito o fracaso de la operación y el tipo de datos que recibimos como respuesta.

Una API REST es una API que se ajusta a los principios de diseño de REST, un estilo de arquitectura también denominado transferencia de estado representacional.

Las API REST se comunican a través de solicitudes HTTP, para realizar funciones estándar de base de datos, como crear, leer, actualizar y suprimir registros (también conocidos como CRUD) dentro de un recurso. Por ejemplo, una API REST utilizará una solicitud GET para recuperar uno o varios registros, una solicitud POST para crearlo, una solicitud PUT para actualizarlo y una solicitud DELETE para suprimirlo. Todos los métodos HTTP se pueden utilizar en llamadas API. Una API REST bien diseñada es similar a un sitio web que se ejecuta en un navegador web con funcionalidad HTTP incorporada.

El estado de un recurso en un instante específico se conoce como la representación de recursos. Esta información se puede entregar a un cliente en varios formatos, entre ellos *JavaScript Object Notation* (JSON), HTML, XML, Python, PHP o un texto sin formato. El formato JSON es popular debido a que es legible tanto por los seres humanos como por las máquinas y debido a que es independiente de un lenguaje de programación.

Las cabeceras y parámetros de solicitud también son importantes en las llamadas de API REST, puesto que incluyen información de identificador importante, como metadatos, autorizaciones, identificadores de recursos uniformes (URI), almacenamiento en memoria caché, cookies y más. Las cabeceras de solicitud y las cabeceras de respuesta, junto con los códigos de estado HTTP convencionales, se utilizan en las API REST bien diseñadas.

La API REST es básicamente el **back end** de nuestras aplicaciones, son funciones y protocolos que nos permiten realizar operaciones: *create, read, update, delete* (CRUD) a un almacén de datos y nos devuelve la información el algún formato, el más usado es JSON.

Data fake

Para realizar pruebas podemos encontrar en Internet varias api's con datos falsos con los que podemos simular nuestro back end.

Pruebe en su navegador las siguientes ligas:

- <https://mdn.github.io/learning-area/javascript/oojs/json/superheroes.json>
- <https://catfact.ninja/docs/api-docs.json>
- <https://randomuser.me/api>
- <https://randomuser.me/api?inc=name,email,picture>
- <https://jsonplaceholder.typicode.com/users>
- <https://reqres.in/api/user>

En cada una, vera en su navegador un conjunto de datos json, este es el funcionamiento de una API REST, usted envía solicitudes (http en este caso) y recibe datos (json en este caso).

3 Bash

GNU Bash o simplemente Bash (Bourne-again shell) es una popular interfaz de usuario de línea de comandos, específicamente un **Shell** de **Unix** (similar a la consola CMD de Windows); así como un lenguaje de scripting de **Git**, permite ejecutar comandos de Linux que son muy útiles durante el desarrollo, por ejemplo, el comando “**curl**”. Bash fue originalmente escrito por Brian Fox para el sistema operativo GNU y está disponible para Windows en varias instalaciones, una de ellas es la de GIT. Bash viene en la instalación de GIT y también en otros paquetes de software como MSys2, puede instalar mySys2 desde:

<https://www.msys2.org/>

Git es un software de control de versiones diseñado por Linus Torvalds, para el mantenimiento de versiones de aplicaciones, particularmente útil cuando estas tienen un gran número de archivos de código fuente. Funciona como un conjunto de comandos ejecutables en una *terminal* (como la de **CMD** o **Power Shell**). Cuando instala GIT, le da la opción de instalar la interfaz BASH (instálela). Puede instalar git desde:

<https://git-scm.com/downloads>

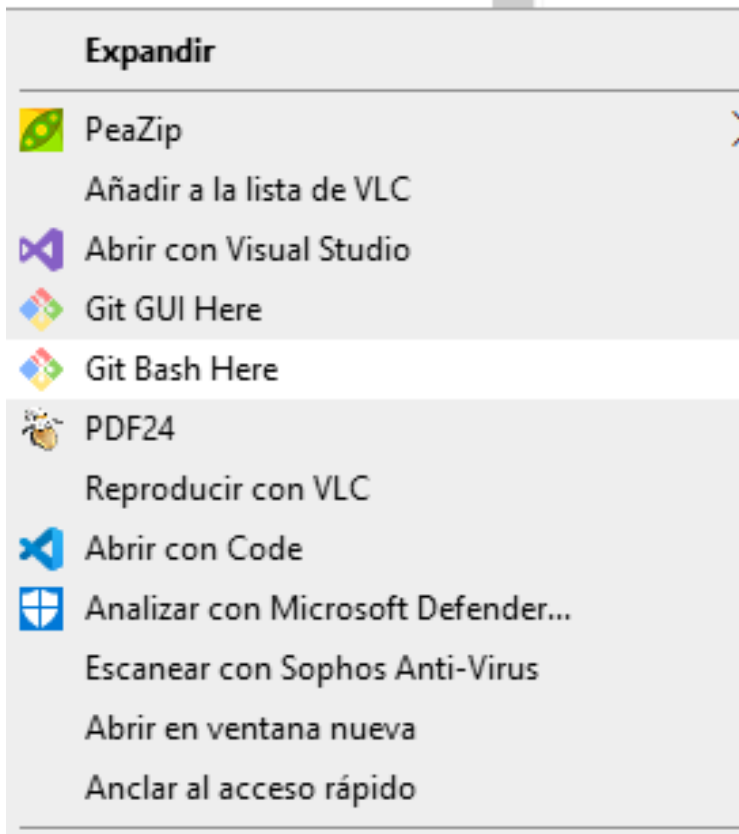
Y en esta liga tiene un video que explica el funcionamiento de git:

https://www.youtube.com/watch?v=HiXLkL42tMU&list=PLExyZ7hD-J5bigogeZq1G_AfD4K4cWPn8&index=1

Una referencia de instalación de Git la encuentra aquí:

<https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Instalaci%C3%B3n-de-Git> .

Si ya tiene instalado Bash, seleccione una carpeta y en el menú contextual (de clic derecho sobre la carpeta) le mostrará la opción:



Git Bash Here entre otras, selecciónela y le abrirá una ventana de Bash:

Se verá más o menos así:

```
Docente@MXL2042C2B MINGW64 /c/trabajo2023/nuevoMaterial/prgAvzITCC/codigos_prgavz (main)
$ |
```

Algunos comandos de la terminal Bash:

Tabla 1 Comandos linux

ls	Muestra los archivos del directorio (carpeta) como dir en msdos
cat archivo	Muestra el contenido de un archivo
cd directorio	Nos lleva al directorio indicado
cd ..	Nos lleva al directorio padre

4 Curl

Para probar un **api** mientras la programamos, o para revisar su funcionamiento mientras no tenemos un **front end** para ella, es bastante útil el comando **curl** de **Linux**, que puede usar desde la consola **Bash** (consola que viene al instalar **git** por ejemplo).

Formato general del comando curl:

```
curl -X POST/GET/PATCH/DELETE [url]
-H "Content-Type: [content type]"
-d '[request data]'
-v/-i
```

Las opciones anteriores tienen sinónimos que son los siguientes:

-X sinonimo --request

-H sinonimo --header

-d sinonimo --data

Los verbos **POST/GET/PATCH/DELETE**, indican el método de conexión al sitio.

La opción -v no muestra las cabeceras, la enviada por el cliente y la que devuelve el servidor.

La opción -i nos da la información resumida. Las opciones -v/-i son opcionales.

Si no se especifica -X se asume que el método a usar será GET.

En tipo de contenido (Content-Type), indica el tipo de codificación en que se envían o reciben los datos, existen varios: json, url-encode, texto plano ...

```
Content-Type: application/json
Content-Type: application/x-www-form-urlencoded
Content-Type: text/html; charset=utf-8
Content-Type: multipart/form-data; boundary=something
```

Al usar -d por default **content type** es: **Content-Type: application/x-www-form-urlencoded**

Si necesita otra codificación que no sea esa, debe especificarla en -H.

Un url se escribe así, ejemplos:

Un ejemplo de url
<code>https://jsonplaceholder.typicode.com/posts</code>
Los datos se escriben así: suponga que quiere enviar un registro con los campos: title y el campo body, en una codificación Content-Type: application/x-www-form-urlencoded , los datos se escribirían así:
<code>-d "title=hola mundo&body=mi primer post"</code>
Los datos se escriben así en una codificación: Content-Type: Application/json , suponga que quiere enviar id, nombre, correo y edad:
<code>-d '{"id":23,"nombre":"ana perez","correo":"ana@mail","edad":32}'</code>
Si usa una codificación Content-Type: Application/json debe especificar -H, así:
<code>-H "Content-Type: Application/json"</code>

Content Type: Codificaciones en http.

La codificación es el formato en que recibimos o enviamos datos a la API y tenemos entre otras, las siguientes opciones.

1	Content-Type: text/html; charset=utf-8
2	Content-Type: Application/json
3	Content-Type: application/x-www-form-urlencoded
4	Content-Type: multipart/form-data; boundary=something

Las que más usaremos son la 2 y la 3, la 2 indica que los datos se envían en formato json, la 3 indica que los datos se envían como una cadena **url** codificada de la siguiente forma: Los valores son codificados en tuplas llave-valor separadas por '&', con un '=' entre la llave y el valor.

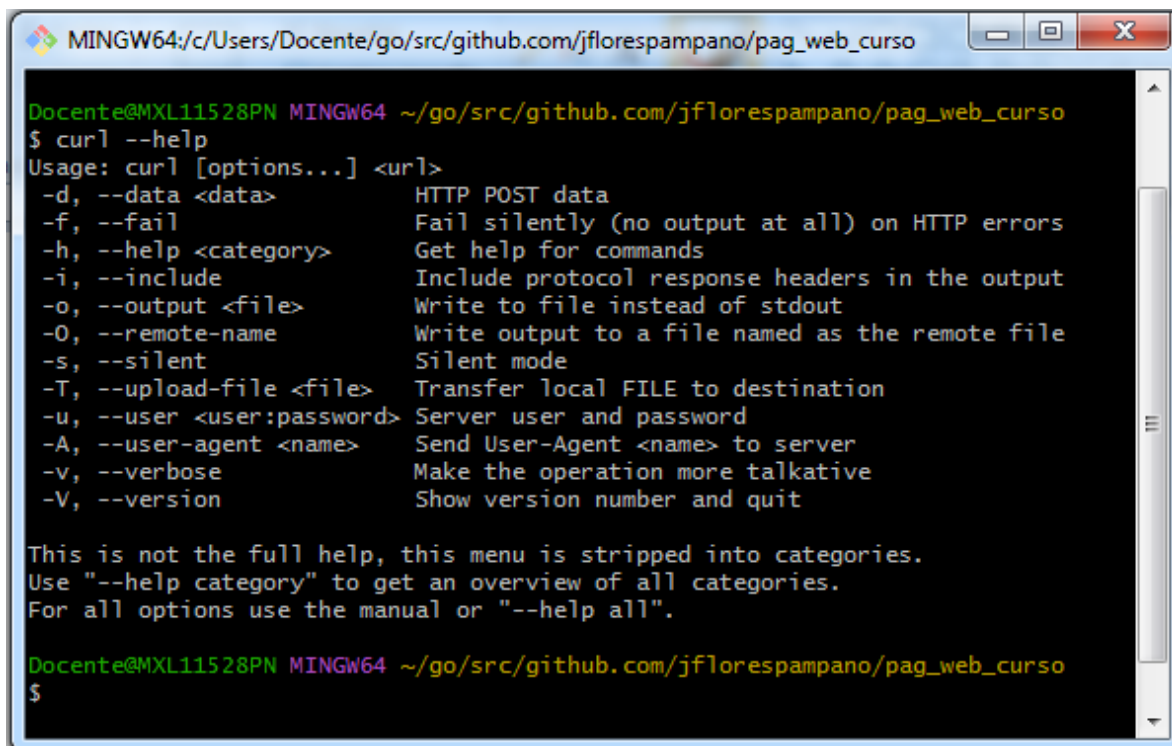
Ejercicio 1:

Abra una ventana de **Bash**.

En la ventana ejecute el comando:

\$curl -help

Nota: no ponga el \$ el símbolo \$ lo pone la consola Bash



```
MINGW64:/c/Users/Docente/go/src/github.com/jflorespampano/pag_web_curso
Docente@MXL11528PN MINGW64 ~/go/src/github.com/jflorespampano/pag_web_curso
$ curl --help
Usage: curl [options...] <url>
  -d, --data <data>           HTTP POST data
  -f, --fail                   Fail silently (no output at all) on HTTP errors
  -h, --help <category>       Get help for commands
  -i, --include                 Include protocol response headers in the output
  -o, --output <file>          Write to file instead of stdout
  -O, --remote-name             Write output to a file named as the remote file
  -s, --silent                  Silent mode
  -T, --upload-file <file>     Transfer local FILE to destination
  -u, --user <user:password>   Server user and password
  -A, --user-agent <name>      Send User-Agent <name> to server
  -v, --verbose                 Make the operation more talkative
  -V, --version                 Show version number and quit

This is not the full help, this menu is stripped into categories.
Use "--help category" to get an overview of all categories.
For all options use the manual or "--help all".

Docente@MXL11528PN MINGW64 ~/go/src/github.com/jflorespampano/pag_web_curso
$
```

Vera una ayuda sobre el uso de curl.

Usando el método get de curl

El sitio <https://catfact.ninja/docs/api-docs.json> tiene un conjunto de datos en formato json, pruebe poniendo esta liga en su navegador. Es una API REST disponible para que usted haga pruebas.

Veamos con acceder a esa API con curl, en la consola bash, ejecute:

```
curl -X GET https://jsonplaceholder.typicode.com/posts
#por default si no se pone -X, se asume que es un get
#por lo tanto el comando anterior lo puede escribir así:
curl https://jsonplaceholder.typicode.com/posts
#como otro ejemplo pruebe los siguientes comandos:
curl https://catfact.ninja/docs/api-docs.json
curl https://reqres.in/api/user
```

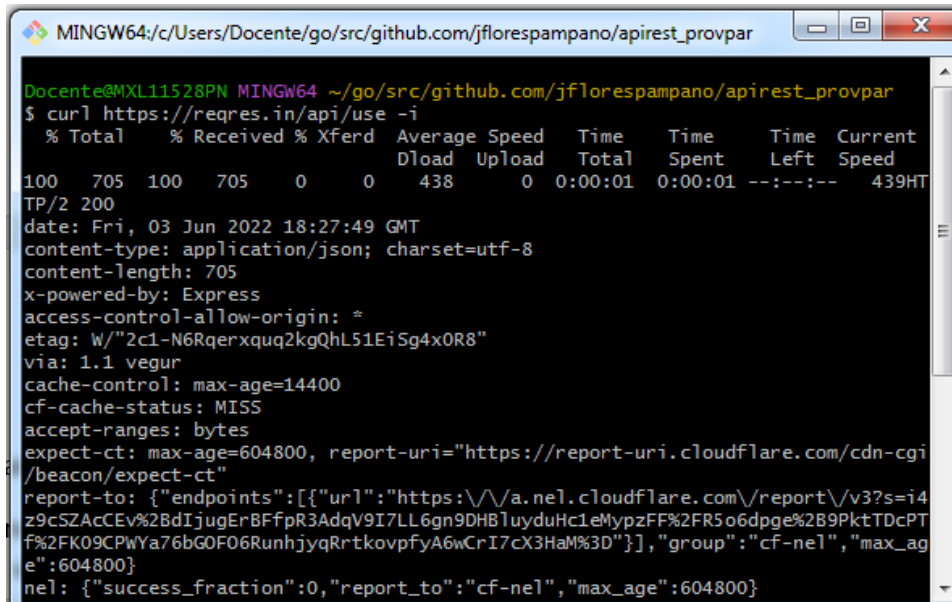
Ahora vera la ventaja de usar curl, escriba el comando:

```
curl https://reqres.in/api/user -v
```

Deberá ver ahora los mismos datos, pero con la cabecera HTTP enviada por el servidor. Si le parece demasiada información, pruebe con el comando siguiente:


```
curl https://reqres.in/api/user -i
```

Ahí podemos ver más claramente algunos datos como el estado de la respuesta:



```
MINGW64/c/Users/Docente/go/src/github.com/jflorespampano/apirest_provpar
Docente@MXL11528PN MINGW64 ~/go/src/github.com/jflorespampano/apirest_provpar
$ curl https://reqres.in/api/user -i
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100  705    100  705    0     0   438      0  0:00:01  0:00:01 --:--:-- 439HT
TP/2 200
date: Fri, 03 Jun 2022 18:27:49 GMT
content-type: application/json; charset=utf-8
content-length: 705
x-powered-by: Express
access-control-allow-origin: *
etag: W/"2c1-N6Rqerxquq2kgQhL51EiSg4x0R8"
via: 1.1 vegur
cache-control: max-age=14400
cf-cache-status: MISS
accept-ranges: bytes
expect-ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi
/beacon/expect-ct"
report-to: {"endpoints":[{"url":"https://a.ne1.cloudflare.com/report/v3?s=i4
z9c5ZAcCEv%2BdIjugErBFfR3AdqV9I7LL6gn9DHB1uyduHc1eMypzFF%2FR5o6dpge%2B9PktTDcPT
f%2FK09CPwYa76bG0F06RunhjyqRrtkovpfyA6wCrI7cX3HaM%3D"}],"group":"cf-ne1","max_ag
e":604800}
ne1: {"success_fraction":0,"report_to":"cf-ne1","max_age":604800}
```

La línea **http/2 200, 200:** indica que no hubo ningún problema. En otra línea, Puede ver que el contenido de la respuesta viene en formato:

content-type: application/json; charset=utf-8.

Ejercicio 2:

El sitio *jsonplaceholder* nos devuelve un conjunto de posts de ejemplo.

```
curl https://jsonplaceholder.typicode.com/posts
```

Sin embargo, ese api, tiene la opción de enviarnos un solo post, agregando en la url el id del post, si deseamos solo el post 1

```
$curl https://jsonplaceholder.typicode.com/posts/1
```

Vera que esta liga nos envía un solo post.

Veamos ahora como solicitar un recurso y almacenarlo en un archivo:

```
curl -o datos.txt https://jsonplaceholder.typicode.com/posts/1
```

También lo puede hacer así:

```
curl https://jsonplaceholder.typicode.com/posts/1 > datos.txt
```

Ambas líneas hacen lo mismo pero la segunda almacena el resultado en el archivo *datos.txt* usando comandos del Shell.

Pruebe el comando:

```
$ curl -I https://jsonplaceholder.typicode.com/posts/1
```

Muestra solo el encabezado del sitio.

Ejercicio 3:

Cuando necesitamos enviar datos al sitio con el método GET, podemos enviarlos en la url de la siguiente forma:

```
curl http://localhost:3000/?id=23&nombre=juan perez&correo=juan@mail&edad=23
```

Donde el sitio al que enviamos los datos es: **http://localhost:3000/**

Y los parámetros son: **?id=23&nombre=juan perez&correo=juan@mail&edad=23**

Hay que tomar en cuenta que en Linux para poner el & hay que poner el carácter de escape \.

Por tanto, debe escribirse así en la consola de Linux:

```
curl http://localhost:3000/?id=23\&nombre=juan perez\&correo=juan@mail\&edad=23
```

En este ejemplo observe que no ponemos -X, recuerde que de no ponerlo por default se están enviando datos por el método GET.

Enviar datos

Cuando necesitamos mandar información a la API y lo podemos hacer en varios formatos como:

```
Content-Type: text/html; charset=utf-8  
Content-Type: Application/json  
Content-Type: application/x-www-form-urlencoded  
Content-Type: multipart/form-data; boundary=something
```

Por ejemplo, cuando pasa parámetros en curl con la opción -d, por default se usa el formato '**Content-Type: application/x-www-form-urlencoded**', si quiere usar otro formato se debe especificar otra con -H

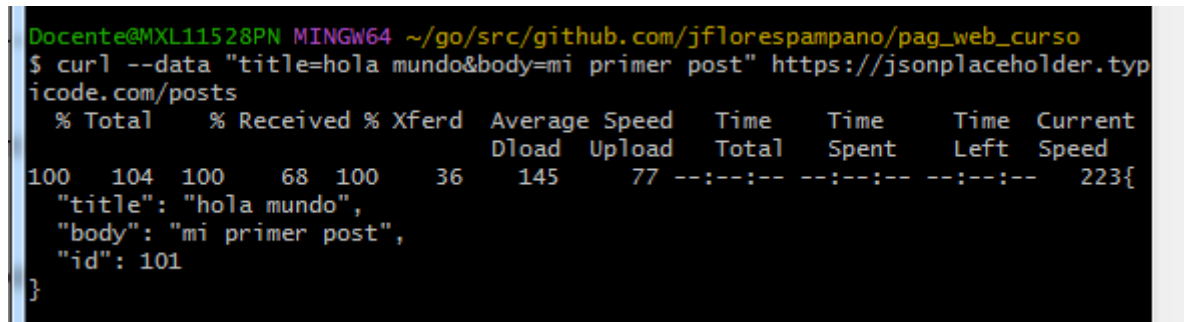
Ejemplo 2:

Suponga que queremos insertar un post en la página: **https://jsonplaceholder.typicode.com/posts**, Para insertar datos: esta página de ejemplo espera que enviemos *title* y *body*, el *userid* y el *id* lo genera la misma página.

Entonces si queremos enviar los datos con el método GET, en curl escribimos:

```
$curl --data "title=hola mundo&body=mi primer post" https://jsonplaceholder.typicode.com/posts
```

Si todo va bien nos devuelve:



```
Docente@MXL11528PN MINGW64 ~/go/src/github.com/jflorespampano/pag_web_curso
$ curl --data "title=hola mundo&body=mi primer post" https://jsonplaceholder.typicode.com/posts
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100  104    100    68  100    36    145    77  --:--:-- --:--:-- --:--:--   223{
  "title": "hola mundo",
  "body": "mi primer post",
  "id": 101
}
```

Nos devuelve el mismo dato enviado que significa que todo estuvo bien.

Este comando se puede escribir también así:

```
$curl -d "title=hola mundo&body=mi primer post" https://jsonplaceholder.typicode.com/posts
```

Nota: Recuerde que cuando omite -X se usa por default el método GET y con -d por default se usa la codificación: **'Content-Type: application/x-www-form-urlencoded'** de no usar esa se debe especificar otra con -H,

Por ejemplo, enviar los mismos datos, pero en formato json:

```
curl -H "Content-Type: Application/json" -d '{"title":"hola mundo","body":"mi primer post"}' https://jsonplaceholder.typicode.com/posts
```

Los datos enviados serán usados por el servidor para crear un registro en esa api, sin embargo, en las aplicaciones modernas se recomienda usar GET solo para recibir datos por lo que a lo más enviaremos un dato con el id del registro que necesitamos de regreso. Si queremos enviar varios datos para una inserción se debe usar el método POST.

```
curl -X POST -H "Content-Type: Application/json" -d '{"title":"hola mundo","body":"mi primer post"}' https://jsonplaceholder.typicode.com/posts
```

O usando la codificación **'Content-Type: application/x-www-form-urlencoded'**:

```
curl -X POST -H "Content-Type: application/x-www-form-urlencoded" -d 'title=hola mundo&body=mi primer post' https://jsonplaceholder.typicode.com/posts
```

Recuerde que si usa la codificación Content-Type: application/x-www-form-urlencoded, -H es opcional y se puede escribir así:

```
curl -X POST -d 'title=hola mundo&body=mi primer post' https://jsonplaceholder.typicode.com/posts
```

Este último comando lo puede escribir también así:

```
curl -X POST -d 'title=hola mundo' -d 'body=mi primer post'  
https://jsonplaceholder.typicode.com/posts
```

En el sitio <https://jsonplaceholder.typicode.com/posts> puede insertar un dato usando GET o POST, porque el sitio esta programado para aceptar la inserción con ambos verbos, cuando usted programe su sitio deberá elegir uno y seguramente será POST.

Por ejemplo, en un sitio programado por usted insertaría así:

```
curl -X POST http://localhost:3000/user -d 'id=23&nombre=ana  
perez&correo=ana@mail&edad=32'  
  
## Post enviando json  
curl -X POST http://localhost:3000/user -H "Content-Type: Application/json" -d  
'{"id": "23", "nombre": "ana perez", "correo": "ana@mail", "edad": 32}'  
  
o usando:  
  
curl --header "Content-Type: Application/json" -d '{"id": "23", "nombre": "ana  
perez", "correo": "ana@mail", "edad": 32}' http://localhost:3000/user  
  
Realmente -H es una abreviación de --header
```

Si tiene una aplicación que espera recibir datos desde un formulario en formato **multipart/form-data**:

```
curl -X POST -F 'name=noviello' -F 'email=noviello@example.com'  
https://example.com/contact.php
```

Cuando se usa la opción -F curl envía los datos usando el tipo de contenido multipart/form-data

Otra forma de realizar una solicitud POST es utilizar la opción -d. Esto hace que los datos se envíen con curl usando la opción: **Content-Type: application/x-www-form-urlencoded**.

```
curl -X POST -d 'name=noviello' -d 'email=noviello@example.com'  
https://example.com/contact.php
```

Una explicación más detallada la puede encontrar en el video:

https://www.youtube.com/watch?v=9u2qDQc6KWg&list=PLExyZ7hD-J5bigogeZq1G_AfD4K4cWPn8&index=17

de donde se tomaron estos ejemplos.

Resumen comandos curl:

```
```sh
#solicitar datos get
curl https://jsonplaceholder.typicode.com/posts
curl -o respuesta.txt https://jsonplaceholder.typicode.com/posts
#enviar datos url-encode con get
curl -d 'title=hola mundo&body=mi primer
post' https://jsonplaceholder.typicode.com/posts
#enviar datos json con get
curl -H "Content-Type: Application/json" -d '{"title":"hola
mundo","body":"mi primer
post"}' https://jsonplaceholder.typicode.com/posts
#enviar datos json con post
curl -X POST -H "Content-Type: Application/json" -d
'{"title":"hola mundo","body":"mi primer
post"}' https://jsonplaceholder.typicode.com/posts
#enviar datos url-encode post forma 1
curl -X POST -H "Content-Type: application/x-www-form-
urlencoded" -d 'title=hola mundo&body=mi primer
post' https://jsonplaceholder.typicode.com/posts
#enviar datos url-encode post forma 2
curl -X POST -d 'title=hola mundo&body=mi primer
post' https://jsonplaceholder.typicode.com/posts
#enviar datos url-encode post forma 3
curl -X POST -d 'title=hola mundo' -d 'body=mi primer
post' https://jsonplaceholder.typicode.com/posts
#enviar a un sitio local
curl -i -d '{"id":56,"nombre":"juan","carrera":"isc"}' -H
"Content-Type: Application/json"
http://localhost/proyectos/alumnosArray/recibeJson.php
```
```

```
```sh
#solicitar datos get
curl https://jsonplaceholder.typicode.com/posts
curl -o respuesta.txt https://jsonplaceholder.typicode.com/posts
#enviar datos url-encode con get
curl -d 'title=hola mundo&body=mi primer post' https://jsonplaceholder.typicode.com/posts
#enviar datos json con get
curl -H "Content-Type: Application/json" -d '{"title":"hola mundo","body":"mi primer
post"}' https://jsonplaceholder.typicode.com/posts
#enviar datos json con post
```

```

curl -X POST -H "Content-Type: Application/json" -d '{"title":"hola mundo","body":"mi primer post"}' https://jsonplaceholder.typicode.com/posts
#enviar datos url-encode post forma 1
curl -X POST -H "Content-Type: application/x-www-form-urlencoded" -d 'title=hola mundo&body=mi primer post' https://jsonplaceholder.typicode.com/posts
#enviar datos url-encode post forma 2
curl -X POST -d 'title=hola mundo&body=mi primer post' https://jsonplaceholder.typicode.com/posts
#enviar datos url-encode post forma 3
curl -X POST -d 'title=hola mundo' -d 'body=mi primer post' https://jsonplaceholder.typicode.com/posts
#enviar a un sitio local
curl -i -d '{"id":56,"nombre":"juan","carrera":"isc"}' -H "Content-Type: Application/json" http://localhost/proyectos/alumnosArray/recibeJson.php
'''

```

## Practica 1:

### Requisitos

- Tener instalado XAMP

### Actividades de la practica:

1 Cree una carpeta de trabajo así: C:\xampp\htdocs\proyectos\alumnosArray

2 Escriba los códigos:

Archivo: recibeGet.php

```

<?php
$vpersonas=[
 0=>["id"=>"01", "nombre"=>"juan p rez", "carrera"=>"isc"],
 1=>["id"=>"02", "nombre"=>"ana a es", "carrera"=>"isc"],
 2=>["id"=>"03", "nombre"=>"luis p ga", "carrera"=>"isc"],
 3=>["id"=>"04", "nombre"=>"pablo d az", "carrera"=>"isc"],
 4=>["id"=>"05", "nombre"=>"rosa ni o", "carrera"=>"isc"]
];

if(!empty($_GET)){
 //se rcbieron datos mediante GET:
 //http://localhost/.../recibeGet.php?id=02
 $id = $_GET['id']??'';
 for ($i=0; $i < sizeof($vpersonas) ; $i++) {
 if($vpersonas[$i]['id']==$id){
 http_response_code(200);
 echo json_encode($vpersonas[$i],JSON_UNESCAPED_UNICODE);
 exit();
 }
 }
}

```



```

2=>["id"=>"03", "nombre"=>"luis púga", "carrera"=>"isc"],
3=>["id"=>"04", "nombre"=>"pablo díaz", "carrera"=>"isc"],
4=>["id"=>"05", "nombre"=>"rosa niño", "carrera"=>"isc"]
];
if(isset($_SERVER["CONTENT_TYPE"])){
 $contentType = trim($_SERVER["CONTENT_TYPE"]);
 if ($contentType != "application/json") {
 header('Content-Type: text/html; charset=utf-8');
 echo "Error, No se recibieron datos json";
 exit();
 }
 $entrada = trim(file_get_contents("php://input"));
 //se enviaron parametros como json
 if(strlen($entrada)>0){
 $datos = json_decode($entrada); //devuelve un object
 $id=$datos->id?? 0;
 $nombre=$datos->nombre??'vacío';
 $carrera=$datos->carrera??'vacío';
 // echo "recibi json:". $id.", ".$nombre.", ".$carrera;
 $len=sizeof($vpersonas);
 $renglon=["id"=>$id, "nombre"=>$nombre, "carrea"=>$carrera];
 $vpersonas[$len]=$renglon;
 http_response_code(201);
 header('Content-Type: application/json');
 echo json_encode($vpersonas,JSON_UNESCAPED_UNICODE);
 }else{
 http_response_code(400);
 header('Content-Type: text/html; charset=utf-8');
 echo "Error: se esperaban datos id, nombre, carrera";
 }
}else{
 http_response_code(400);
 header('Content-Type: text/html; charset=utf-8');
 echo "Error: se esperaban [CONTENT_TYPE] ";
}
?>

```

3 Pruebe los comandos:

curl <http://localhost/proyectos/alumnosArray/recibeGet.php>

curl <http://localhost/proyectos/alumnosArray/recibeGet.php?id=02>

curl <http://localhost/proyectos/alumnosArray/recibePost.php>

curl -i -d "id=55" -d "nombre=juan" -d "carrera=isc"  
<http://localhost/proyectos/alumnosArray/recibePost.php>



```
curl -i -d "id=56&nombre=juan&carrera=isc"
```

<http://localhost/proyectos/alumnosArray/recibePost.php>

```
curl -i -d '{"id":59,"nombre":"juan","carrera":"isc"}' -H "Content-Type: application/json"
```

<http://localhost/proyectos/alumnosArray/recibeJson.php>

## 5 Thunder Client

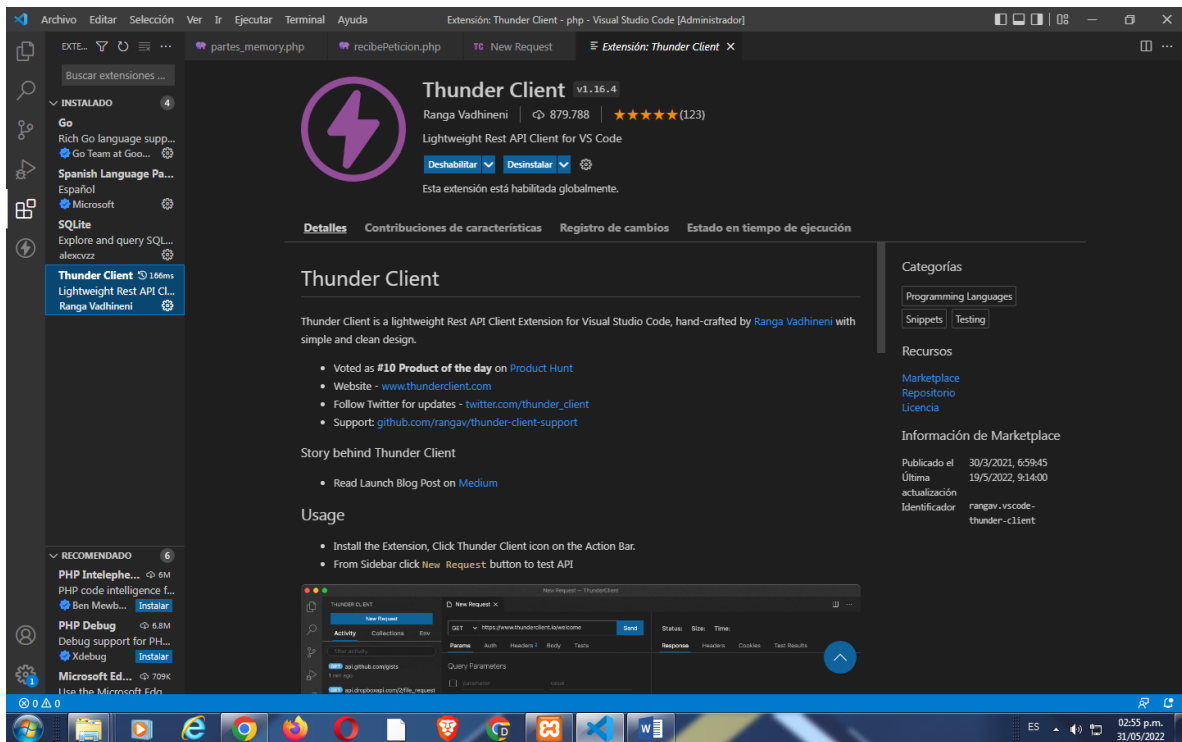
Para la siguiente parte necesitaremos usar Thunder Client, este último es una extensión de code que hace lo mismo que curl, pero en un ambiente de ventanas más amigable, para probarlo, volvamos al código php que probamos cuando vimos curl.

Recordemos el código:

Archivo: recibePeticon.php

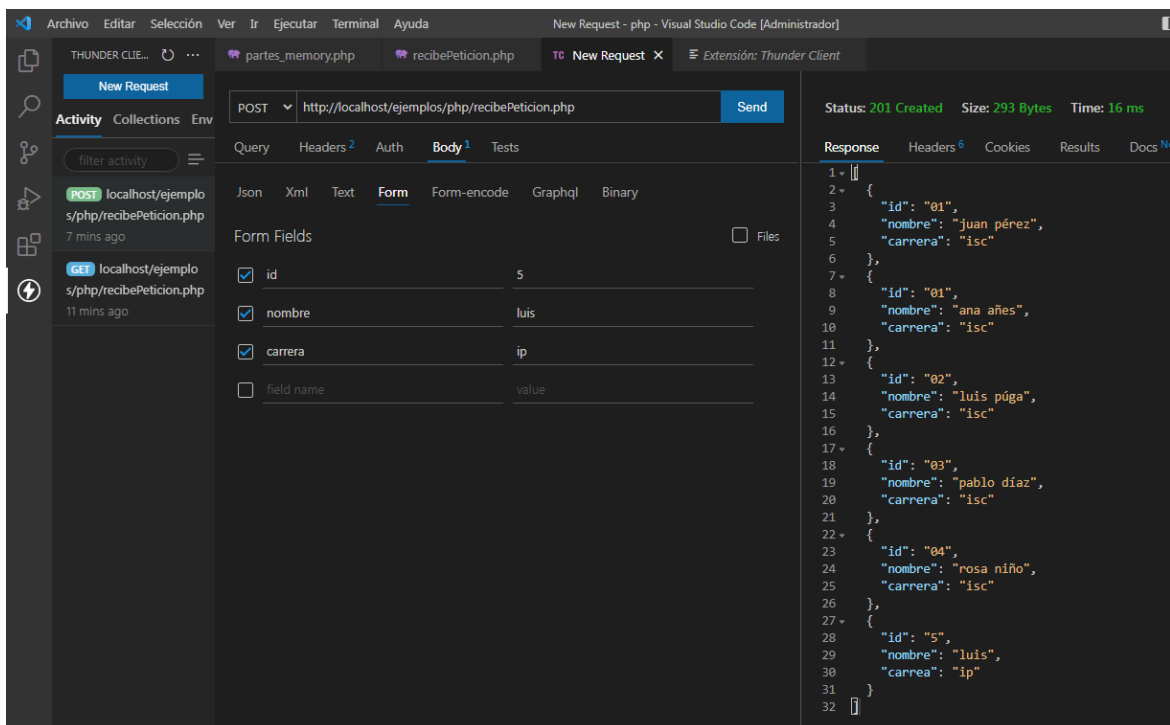
```
<?php
$vpersonas=[
 0=>["id"=>"01", "nombre"=>"juan p  rez", "carrera"=>"isc"],
 1=>["id"=>"01", "nombre"=>"ana a  es", "carrera"=>"isc"],
 2=>["id"=>"02", "nombre"=>"luis p  ga", "carrera"=>"isc"],
 3=>["id"=>"03", "nombre"=>"pablo d  az", "carrera"=>"isc"],
 4=>["id"=>"04", "nombre"=>"rosa ni  o", "carrera"=>"isc"]
];
if (isset($_POST) && !empty($_POST)){
 $id = $_POST['id'];
 $nombre = $_POST['nombre'];
 $carrera = $_POST['carrera'];
 $len=sizeof($vpersonas);
 $renglon=["id"=>$id,"nombre"=>$nombre, "carrea"=>$carrera];
 $vpersonas[$len]=$renglon;
 http_response_code(201);
}else{
 http_response_code(400);
}
header('Content-Type: application/json');
echo json_encode($vpersonas,JSON_UNESCAPED_UNICODE);
?>
```

Agregue el complemento Thunder Client en code:

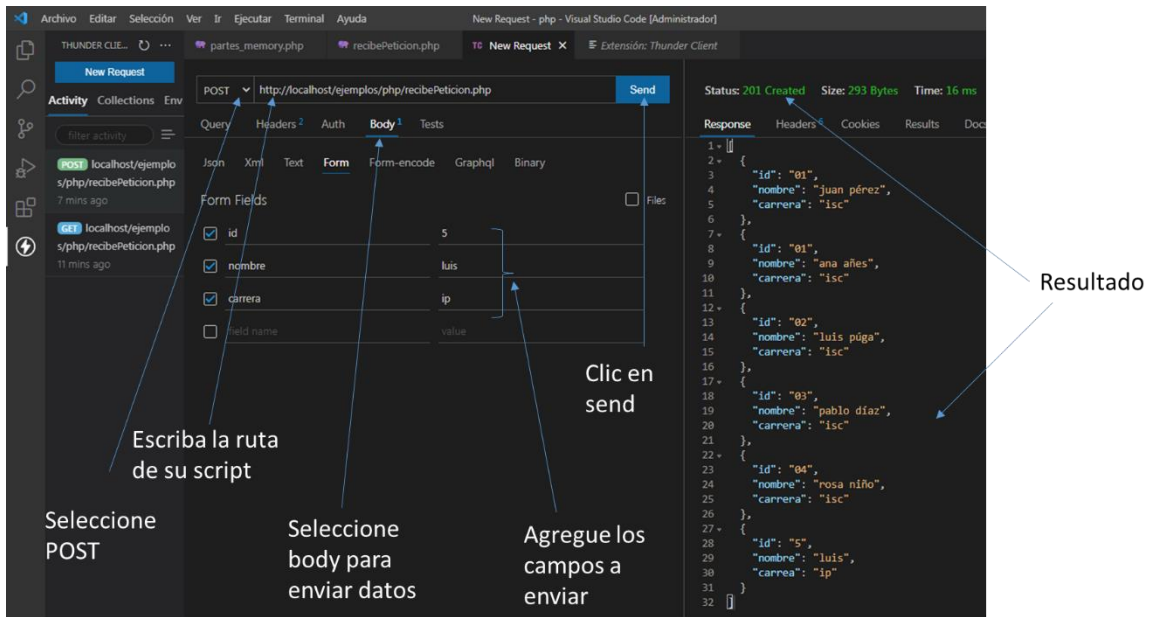


En el ícono del rayito de lado izquierdo de clic.

Aparece la ventana:



Aquí puede enviar peticiones post, get, put etcétera.



## Referencias:

1. <https://steemit.com/cervantes/@orlmicron/estructuras-las-clases-de-go-golang>
2. <https://steemit.com/cervantes/@orlmicron/tipos-de-datos-personalizados-y-metodos-en-go-golang>
3. [https://go.dev/doc/effective\\_go#interface\\_methods](https://go.dev/doc/effective_go#interface_methods) (ver interfaces y métodos)
4. <https://pkg.go.dev/encoding/json#Encoder>
5. <https://developer.mozilla.org/es/docs/Web/HTTP>