{desafío} latam_

Más allá del GET

Objetivo

- Utilizar los verbos de POST, PUT y DELETE
- Conocer el concepto de negociación de contenido
- Utilizar headers para negociar contenido
- Utilizar el body de un request para enviar contenido
- Crear un nuevo objeto utilizando el método Post

Introducción

Las APIs no solo nos permiten obtener información, también podemos subir información, actualizar información y borrar información. En una API Rest estos pedidos suceden bajo otros verbos diferentes a **GET**

¿Cómo subir información?

En una API Rest hay un estándar definido. El método **Post** se utiliza para subir información.

Routes

All HTTP methods are supported.

GET	/posts
GET	/posts/1
GET	/posts/1/comments
GET	/comments?postId=1
GET	/posts?userId=1
POST	/posts
PUT	/posts/1
PATCH	/posts/1
DELETE	/posts/1

Note: you can view detailed examples here.

Pero esta parte de la documentación no nos dice mucho, pero hay un link a ejemplos, sigamos el link.

Creating a resource

```
// POST adds a random id to the object sent
fetch('https://jsonplaceholder.typicode.com/posts', {
    method: 'POST',
    body: JSON.stringify({
      title: 'foo',
      body: 'bar',
      userId: 1
    }),
    headers: {
      "Content-type": "application/json; charset=UTF-8"
    }
  })
  .then(response => response.json())
  .then(json => console.log(json))

/* will return
{
  id: 101,
  title: 'foo',
  body: 'bar',
  userId: 1
}
*/
```

El ejemplo de la documentación está en Javascript, pero de todas formas podemos leerlo poniendo atención en algunos elementos específicos.

Estudiemos los elementos clave:

- El request indica el método post.
- El request indica además headers donde hay un content-type
- El body está envuelo en un método .stringify que si buscamos en internet aprenderemos que es la operación inversa a .parse o sea que puede transformar un hash en un json.

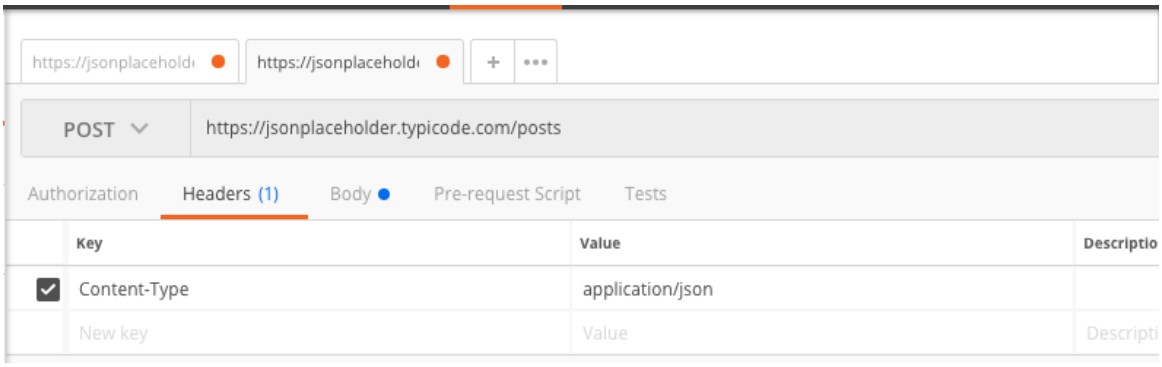
Utilicemos esto para agregar un artículo nuevo a la API

¿Qué son el header y el body?

Un request a una API Rest tiene un header y un body, en el header especificamos información general, por ejemplo codificación o tipo de contenido que vamos a enviar. En el body enviaremos información específica, por ejemplo si queremos subir un artículo aquí agregaremos el título y el contenido.

Subiendo un artículo desde Postman

Para subir un artículo nuevo necesitamos seleccionar el verbo post, e ingresar la URL de la documentación.



Negociación de contenido

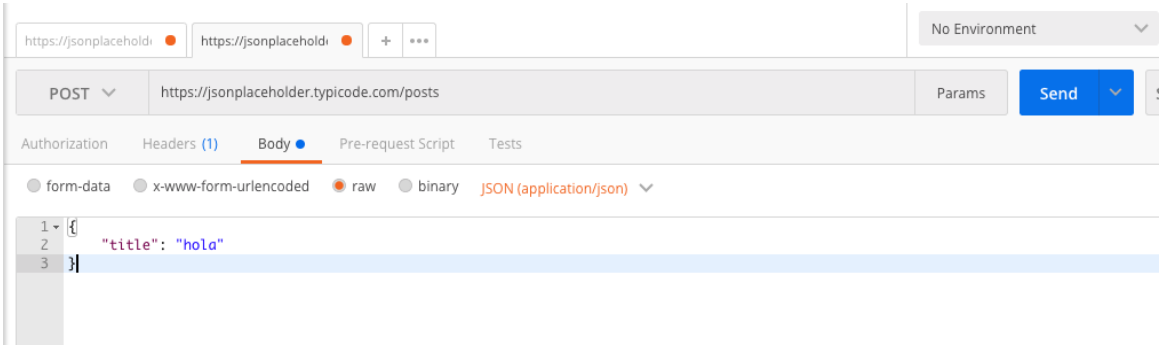
Además necesitaremos agregar el header `Content-type: application/json` Esto es porque algunas APIs pueden recibir distintos tipos de contenido.

Negociar el contenido permite que en un mismo endpoint se pueden resolver pedidos de distintos clientes en distintos formatos. Los tipos de contenido que puede manejar una API se especifican en la documentación, en este caso lo obtuvimos del ejemplo.

Agregando el contenido

Dentro de la API y del ejemplo vimos que un artículo se compone de un título **title**, de un cuerpo **body** y de un dueño **userId**

Como primera prueba subiremos un artículo. Para eso iremos al tab de body y seleccionar la opción **raw** y a la derecha marcaremos la opción JSON (application/json)



Aquí es muy importante indicar que los strings **deben utilizar doble comilla**, utilizar comillas simples alrededor de un string en un JSON dará como resultado en un error.

Tip: Validador de JSON en línea: https://jsonlint.com/

Observando el resultado

En la sección inferior bajo el tab body encontraremos el resultado entregado de la API indicando que se creó el artículo con id 101 y título "hola".

Intentemos guardar un nuevo artículo, ahora con `body` y `userId`


```
{
   "title": "Post 101",
   "body": "Este es nuestro primer post",
   "userId": 1
}
```


Obtendremos como respuesta:


```
{
   "title": "Post 101",
   "body": "Este es nuestro primer post",
   "userId": 1,
   "id": 101
}
```


Esta API en particular no es persistente

Para asegurar el funcionamiento con muchos usuarios distintos, esta API no guarda los cambios realizados, solo dice que los hizo, trabajar con una API que los guarda es exactamente igual.

Subiendo un artículo desde Ruby

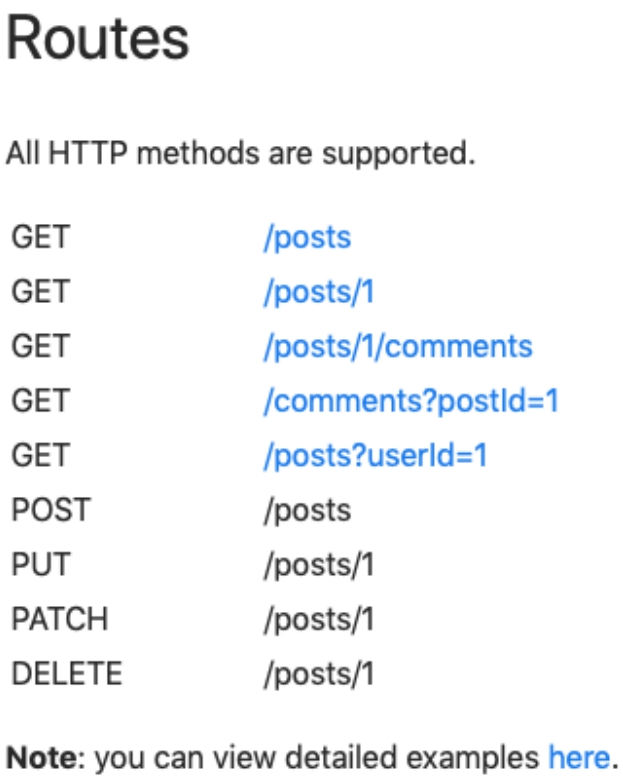
Para lograr esto copiaremos el código que genera Postman

```
require 'uri'
require 'net/http'

url = URI("https://jsonplaceholder.typicode.com/posts")

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true
http.verify_mode = OpenSSL::SSL::VERIFY_NONE

request = Net::HTTP::Post.new(url)
request["content-type"] = 'application/json'
request["cache-control"] = 'no-cache'
request["postman-token"] = 'f230cc3e-dcda-37b6-556d-fd1e9d334108'
request.body = "{\n\t\"title\": \"Post 101\",\n\t\"body\": \"Este es nuestro primer Post\",\n\t\"userId\": 1\n}\n"

response = http.request(request)
puts response.read_body
```

Actualizando un recurso

Los métodos para actualizar un recurso en una API rest son PUT o PATCH, si bien existe una diferencia entre estos la mayoría de las API no hace distinción.

Lo otro que tenemos que saber además del verbo es el endpoint (o dirección). Para esto veremos de nuevo la documentación.



Routes

All HTTP methods are supported.

GET	/posts
GET	/posts/1
GET	/posts/1/comments
GET	/comments?postId=1
GET	/posts?userId=1
POST	/posts
PUT	/posts/1
PATCH	/posts/1
DELETE	/posts/1

Note: you can view detailed examples here.

Actualizando el recurso desde Postman

Vemos en la documentación que a diferencia de subir un artículo que utiliza la ruta `/posts` la actualización utiliza la ruta `/posts/1` donde el número no tiene que ser necesariamente 1, si no que es identificador del recurso que queremos actualizar, para saber cuales son tenemos que listar los recursos (como lo hicimos al momento de aprender a ocupar las APIs)

En este caso los identificadores van desde 1 hasta 100, por lo tanto si queremos cambiar el artículo con id 20 tendríamos que hacer un request con método put a `https://jsonplaceholder.typicode.com/posts/20`

Ademas dentro de body tenemos que agregar los nuevos valores para el artículo.

Actualizando un artículo desde Ruby

Para estudiar como podemos modificar el artículo utilizando Ruby lo que haremos es copiar el código generado por Postman, y veremos que la idea es exactamente la misma.

```ruby
require 'uri'
require 'net/http'

url = URI("https://jsonplaceholder.typicode.com/posts/20")

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true
http.verify_mode = OpenSSL::SSL::VERIFY_NONE

request = Net::HTTP::Put.new(url)
request["content-type"] = 'application/json'
request["accept-charset"] = 'UTF-8'
request["cache-control"] = 'no-cache'
request["postman-token"] = 'e12f50c0-9587-fbde-0960-61de47ba3517'
request.body = "{\n\t\"title\": \"Cambio de post\",\n\t\"body\": \"Actualizando el post\",\n\t\"userId\": 1\n}\n"

response = http.request(request)
puts response.read_body
```

Borrar un artículo

Para borrar un artículo seguiremos la misma idea, veremos de la documentación que el método es delete y que la dirección es `posts/id`

Borrando un artículo desde Ruby

Utilizaremos el código generado por Postman

```ruby
require 'uri'
require 'net/http'

url = URI("https://jsonplaceholder.typicode.com/posts/20")

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true
http.verify_mode = OpenSSL::SSL::VERIFY_NONE

request = Net::HTTP::Delete.new(url)
request["content-type"] = 'application/json'
request["accept-charset"] = 'UTF-8'
request["cache-control"] = 'no-cache'
request["postman-token"] = '9a1b192d-aeb7-4509-688f-2fe0fbf5b4b0'
request.body = "{\n\t\"title\": \"Cambio de post\",\n\t\"body\": \"Actualizando el post\",\n\t\"userId\": 1\n}\n"

response = http.request(request)
puts response.read_body
```

Resumen del capítulo

En este capítulo aprendimos:

- Una API Rest es un estándar para ver, crear, actualizar y borrar recursos.
- Para ver recursos utilizaremos el verbo (o http method) **GET**
- Para agregar un recurso nuevo utilizaremos el verbo **POST**
- Para actualizar un recurso utilizaremos **PUT** o **PATCH**
- Para borrar un recurso utilizaremos **DELETE**