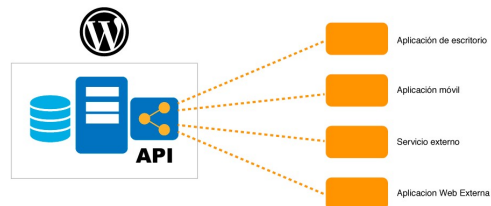


API REST

Denis Pacheco

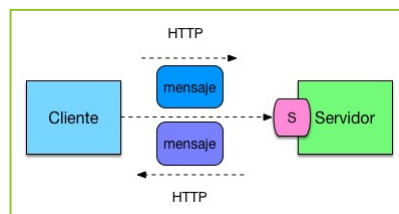
API

- ▶ Una API (**Application programming interface**) es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones.
- ▶ Permite la comunicación entre dos aplicaciones de software a través de un conjunto de reglas.



REST

- ▶ REST (**Representational State Transfer**) es un estilo de arquitectura basada en principios de diseño que facilitan la transmisión cliente-servidor a través de principios estandarizados:
- ▶ Funciona a través del protocolo HTTP
- ▶ Separa el cliente del servidor (los independiza)
- ▶ Sin estado, en otras palabras, el servidor no necesita saber nada acerca de los datos guardados en el cliente.



{ REST }

RESTful

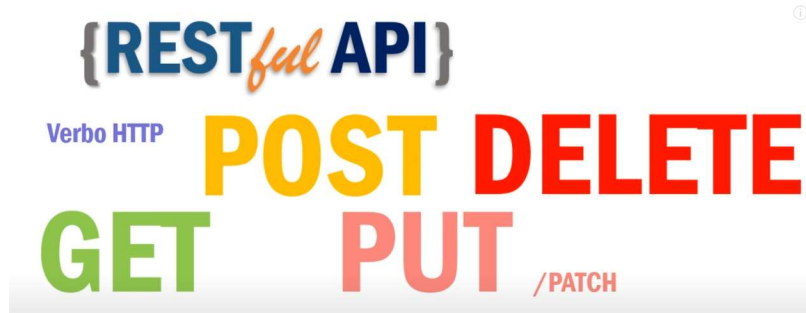
- ▶ Un servicio **RESTful** es una API implementada usando la arquitectura REST.
- ▶ Los servicios **RESTful** se basan en recursos, los cuales se almacenan en el servidor.
- ▶ Características:
 - ▶ Tiene 4 operaciones básicas: crear, leer, actualizar y eliminar
 - ▶ Cada operación necesita 3 cosas: el URI, el verbo HTTP y los datos
 - ▶ El URI es un sustantivo que contiene el nombre del recurso consultado
 - ▶ El HTTP es un verbo, que define la operación
 - ▶ Los datos pueden estar en distintos formatos

{ REST }

RESTful
Web Services

RESTful

- Los verbos que indican las operaciones a realizar son:



{REST}
RESTful
Web Services

RESTful

- Las URIs son las direcciones que indican el servidor y el recurso que se quiere consultar/modificar.
- Por ejemplo (de la Api <https://pokeapi.co/>):

► <https://pokeapi.co/api/v2/pokemon/1>

Dirección del servidor

Nombre del recurso+ID

{REST}
RESTful
Web Services

RESTful

Operación	Método HTTP	URI	Parámetros	Resultado
Crear	POST	/[recurso]	Dentro del cuerpo en el POST	Se crea un nuevo recurso
Leer	GET	/[recurso]/[recurso_id]	No aplica	Recurso en función al id
Actualizar	PATCH/PUT	/[recurso]/[recurso_id]	Se pasan usando una cadena de consulta	Se actualiza/reemplaza el recurso
Borrar	DELETE	/[recurso]/[recurso_id]	No aplica	Se elimina el recurso en función al id

{REST}
RESTful
Web Services

RESTful

- ▶ Lo importante, es que al cliente no le interesa como está construido el servidor, y viceversa.
- ▶ Ambos extremos no se ven, no importa en que lenguajes/framework está construido uno o el otro, simplemente se comunican a través de los canales establecidos.
- ▶ El formato de comunicación de datos puede ser JSON, XML, texto plano o binario.

{REST}
RESTful
Web Services

Json

- ▶ Existen varios formatos para comunicar datos entre servicios, sin embargo, el mas utilizado actualmente es JSON
- ▶ El formato JSON (JavaScript Object Notation) es un formato abierto utilizado como alternativa al XML para la transferencia de datos estructurados entre un servidor de Web y una aplicación Web.
- ▶ Su lógica de organización tiene puntos de semejanza con el XML, pero posee una notación diferente.

{JSON}
JavaScript Object Notation

Json

- ▶ Los archivos JSON también trabajan con pares de atributos y valores, y en vez de marcadores, como en el XML, utilizan delimitadores en cadenas: {}, [], y "". Un típico archivo JSON queda estructurado de la siguiente manera:

```
{
  "localidade 1": {
    "Continente": "África",
    "País": "Angola",
    "Capital": "Luanda"
  },
  "localidade 2": {
    "Continente": "América do Norte",
    "País": "Estados Unidos",
    "Capital": "Washington DC"
  },
  "localidade 3": {
    "Continente": "América Central",
    "País": "México",
    "Capital": "Cidade do México"
  },
  "localidade 4": {
    "Continente": "América do Sul",
    "País": "Brasil",
    "Capital": "Brasília"
  },
  "localidade 5": {
    "Continente": "Europa",
    "País": "Espanha",
    "Capital": "Madri"
  },
  "localidade 6": {
    "Continente": "Europa",
    "País": "Alemanha",
    "Capital": "Berlín"
  },
}
```

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",

```

{JSON}
JavaScript Object Notation

Json

- El delimitador `{` marca el inicio de una sección, y el `}` marca su finalización. Los pares de valor y atributo están separados por `:` y sus valores, cuando se trata de texto, se expresan entre comillas (los números, por ejemplo, no reciben comillas). En el ejemplo de abajo, la "localidad 6" es un atributo que recibe una serie de valores (Continente, País y Capital):

```
##"localidad 6": {
  "Continente": "Europa",
  "País": "Alemanha",
  "Capital": "Berlin"
},
```

- Obsérvese que el valor de la "localidad 6" es un nuevo conjunto de pares atributo-valor. Ese nuevo conjunto se inicia a partir del delimitador `{` y finaliza con `}`. La lógica de encadenar conjuntos de pares puede ser repetida innumerables veces, creando así diversos niveles para la estructura de datos deseada.

{JSON}
JavaScript Object Notation

Json

- Si se carga este objeto en un programa, convertido (parseado) en una variable llamada `superHeroes` por ejemplo, se podría acceder a los datos que contiene utilizando la misma notación de punto/corcheque que se revisó en el artículo JavaScript object basics. Por ejemplo:

```
superHeroes.homeTown
superHeroes['active']
```

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",

```

{JSON}
JavaScript Object Notation

Json

- Para acceder a los datos que se encuentran más abajo en la jerarquía, simplemente se debe concatenar los nombres de las propiedades y los índices de arreglo requeridos. Por ejemplo, para acceder al tercer superpoder del segundo héroe registrado en la lista de miembros, se debería hacer esto:

```
superHeroes['members'][1]['powers'][2]
```

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno"
      ]
    }
  ]
}
```

{JSON}
JavaScript Object Notation

Implementación

- Para crear una “request” del cliente hacia el servidor, generalmente se debe incluir :
- El verbo HTTP (GET,POST,PUT,DELETE)
- Un **header** que permite al cliente pasar información acerca de la petición
- El **path** o ruta hacia el recurso (URI)
- En **body** (opcional) que contiene los datos

{REST}
RESTful
Web Services

Headers

- ▶ En el encabezado del request el cliente envía el tipo de contenido que puede recibir del servidor.
- ▶ Por ejemplo:
- ▶ Text/html
- ▶ Text/css
- ▶ Text/plain
- ▶ Image/png
- ▶ application/json
- ▶ application/xml

```
GET /articles/23
Accept: text/html, application/xhtml
```



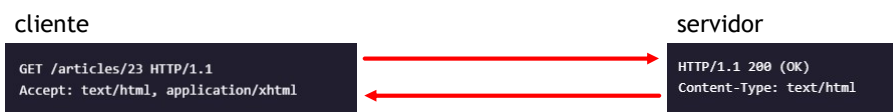
paths

- ▶ Las peticiones deben contener un “path” o ruta a un recurso, las cuales deben ser diseñadas para que el cliente sepa donde se está metiendo
- ▶ Por convención, la primera parte es generalmente el plural del recurso.
- ▶ Por ejemplo:
- ▶ De tienda.com/customers/223/orders/12
- ▶ Se puede desprender que se busca el cliente 223, para luego de ese cliente obtener la orden n° 12
- ▶ Esta ruta varía dependiendo del verbo. Por ejemplo, para un GET no es necesario indicar un ID



Respuestas

- ▶ Cuando el servidor responde a una consulta, la respuesta (data payload) puede venir en distintos formatos.
- ▶ Por esto , el servidor debe incluir la definición del “content-type” en un campo del header
- ▶ Este campo indica el formato de los datos enviados en el body.
- ▶ Ej:



{REST}
RESTful
Web Services

Respuestas

- ▶ Además, el servidor debe contener un código “**status code**” para indicar al cliente si la operación fue exitosa o no.
- ▶ Algunos de los códigos más comunes son:
- ▶ **400** (BAD REQUEST) : error del cliente (generalmente sintaxis)
- ▶ **403** (FORBIDDEN) : permiso
- ▶ **404** (NOT FOUND) : no se encontró el recurso
- ▶ **500** (INTERNAL SERVER ERROR): error en el servidor
- ▶ <https://www.restapitutorial.com/httpstatuscodes.html>

{REST}
RESTful
Web Services

Respuestas

- ▶ Cuando las operaciones son exitosas, dependiendo del verbo que se utilice, el server debe enviar lo siguiente:
- ▶ **GET:** 200(OK)
- ▶ **POST:** 201(CREATED)
- ▶ **PUT:** 200(OK)
- ▶ **DELETE:** 204(NO CONTENT)



Request

- ▶ Ejemplo de request:

HTTP Request Example:

```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

Diagram labels for the request structure:

- Request Line: Points to the first line of the request.
- Request Headers: Points to the header lines.
- Request Message Header: Points to the entire header section.
- A blank line separates header & body: Points to the blank line between the header and body.
- Request Message Body: Points to the body of the request.



Response

- Ejemplo de Response:

HTTP Response Example:

```

HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>

```

Diagram labels for the HTTP response structure:

- Status Line: `HTTP/1.1 200 OK`
- Response Headers: `Date: Sun, 08 Feb xxxx 01:11:12 GMT`, `Server: Apache/1.3.29 (Win32)`, `Last-Modified: Sat, 07 Feb xxxx`, `ETag: "0-23-4024c3a5"`, `Accept-Ranges: bytes`, `Content-Length: 35`, `Connection: close`, `Content-Type: text/html`
- Response Message Header: The entire header section (Status Line and Response Headers).
- A blank line separates header & body
- Response Message Body: `<h1>My Home page</h1>`

{REST}
RESTful
Web Services

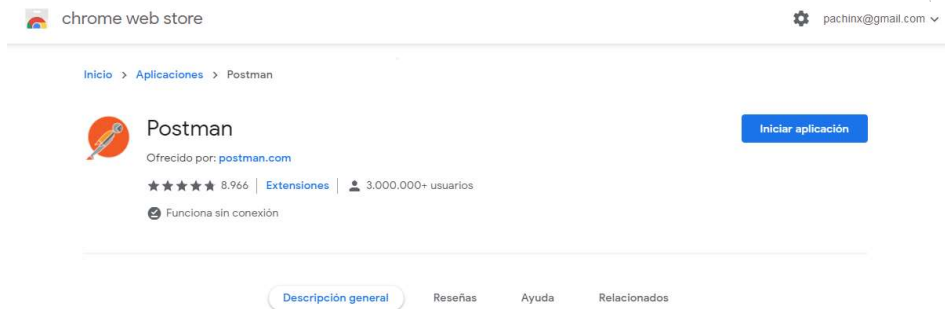
REST

- Ya vimos como funcionaba REST, ahora lo vamos aprobar con un par de programas para hacer request y ver como son las consultas a distintas APIs y sus respuestas
- Para esto necesitaremos instalar Postman, que en principio es una extensión de Chrome, pero también tiene un instalador para Windows.
- Usaremos postman con distintas APIs que se encuentran disponibles en forma gratuita: reqRes, pokeApi, JSONplaceholder, etc.



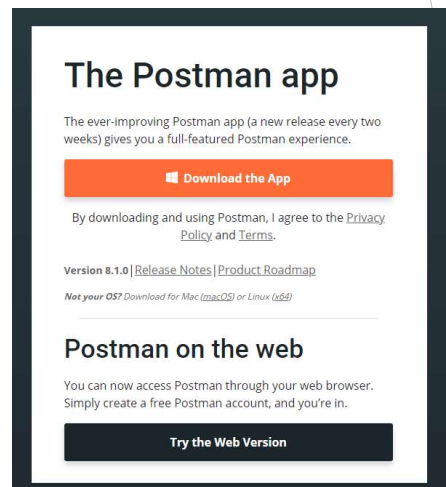
Instalando postman

- Pueden buscar postman en Chrome:



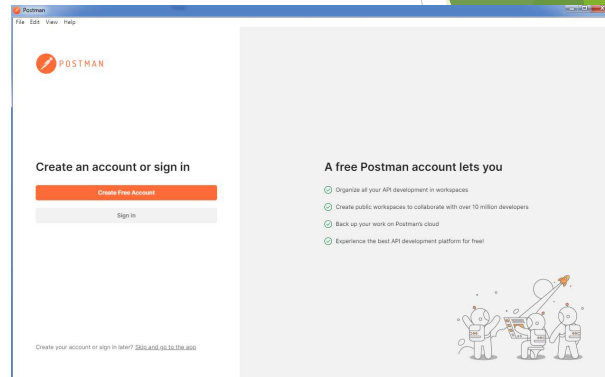
Instalando postman

- También lo pueden descargar de :
- <https://www.postman.com/downloads/>



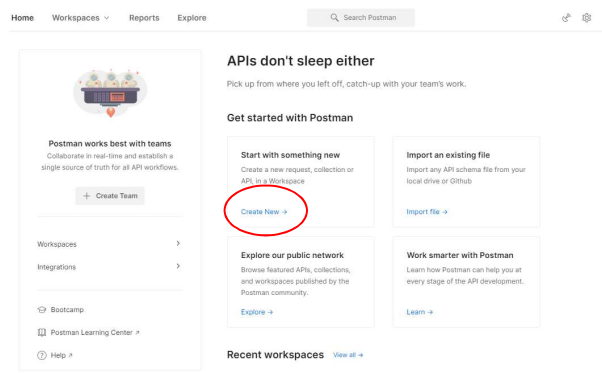
Usando postman

- ▶ Al abrir, postman les pedirá crear una cuenta
- ▶ Pueden usar su correo de Gmail para crear una automáticamente.



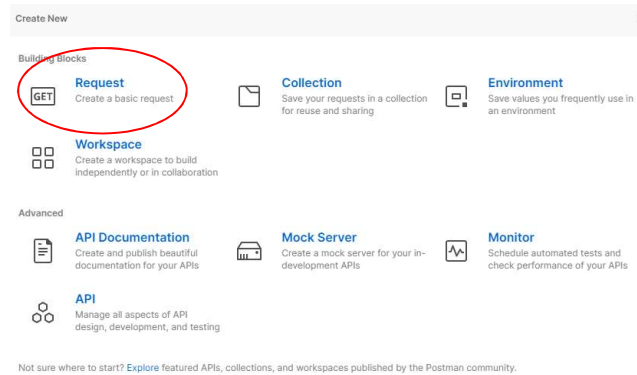
Usando postman

- ▶ Para iniciar una nueva request, podemos crear hacerlo haciendo click en “create new”



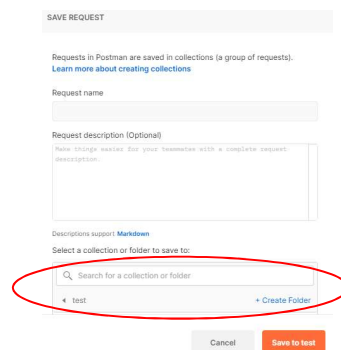
Usando postman

- Aquí postman nos preguntará que queremos hacer. Le damos a request y aceptar



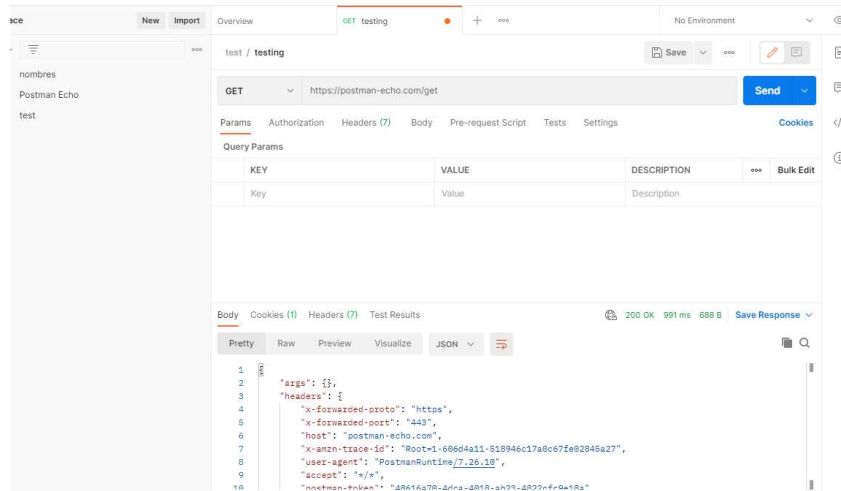
Usando postman

- Luego nos preguntará donde queremos guardar nuestra “request”, en este caso, creamos una nueva carpeta y lo guardamos ahí:



Usando postman

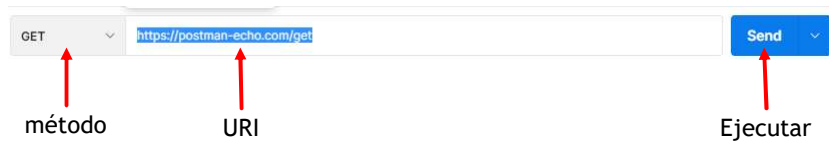
- Una vez listo, nos abrirá la pantalla de request:



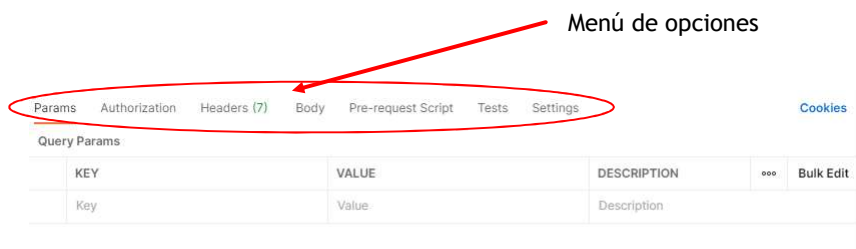
Usando postman

- En esta pantalla podemos elegir el tipo de request, poner datos en el header y body, enviar la petición y recibir una respuesta.
- Para el primer ejemplo, usaremos la dirección de testing de postman:
- <https://postman-echo.com/get>
- Elegiremos el método GET y presionaremos enviar. Esto nos debe traer una respuesta en formato **json** (que está configurado por defecto)

Usando postman



Usando postman

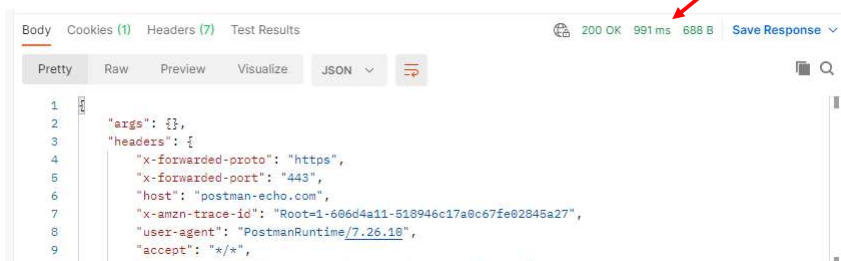


- ▶ En este menú podemos introducir datos que complementan el request
- ▶ Datos del body, especificación del header, parámetros, etc.



Usando postman

- ▶ Respuesta:
- ▶ Aquí podemos elegir el formato para el orden y visualización de la respuesta que nos llega
- ▶ En la parte superior derecha nos da la info de la respuesta (código, texto, etc..)



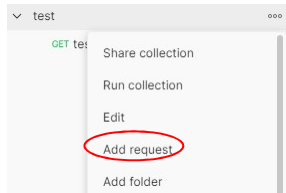
Usando postman

- ▶ Haciendo consultas: vamos a hacer un par de consultas a distintas APIs, la primera a pokeAPI.co
- ▶ La url base para consultar en esta api es:
- ▶ <https://pokeapi.co/api/v2/>
- ▶ A esto podemos agregar los recursos que queremos visualizar.
- ▶ Por ejemplo consultemos todos los pokemons, por lo que a nuestra url base debemos agregarle el recurso /pokemon
- ▶ Quedaría:
- ▶ <https://pokeapi.co/api/v2/pokemon>



Usando postman

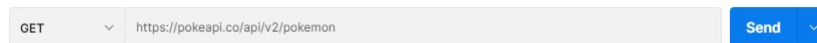
- Nos vamos a **postman** y agregamos un nuevo **request** (o usamos el mismo si quieren)



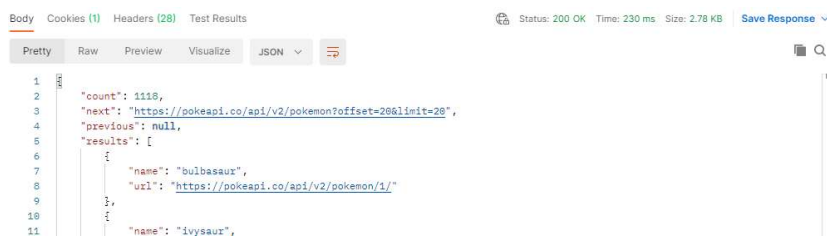
PokéAPI

Usando postman

- Ahora escribimos nuestra URI en la barra del request:



- Ejecutamos el send y nos debiera llegar la respuesta:



PokéAPI

Usando postman

- Aquí podemos analizar la respuesta:
- Por ejemplo el código y texto del código, además del tiempo de ejecución y el tamaño de la respuesta:

Status: 200 OK Time: 230 ms Size: 2.78 KB Save Response

- Podemos ver también los headers enviados por la API:

Body	Cookies (1)	Headers (28)	Test Results
Status: 200 OK Time: 230 ms Size: 2.78 KB			
CF-Cache-Status		HIT	
Age		32481	
cf-request-id		094c941b1d000056b9c0207000000001	
Expect-CT		max-age=604800, report-uri="https://report-uri.cloudflare.com/	
Report-To		({"max_age":604800,"group":"cf-nel","endpoints":[{"url":"https://	
NEL		({"max_age":604800,"report_to":"cf-nel"})	
Server		cloudflare	
CF-RAY		63c122d82be350b9-LIM	
Content-Encoding		br	



Usando postman

- Y por supuesto, podemos ver también los datos que vienen en el **body**, en formato **json**

Body Cookies (1) Headers (27) Test Results

Pretty Raw Preview Visualize JSON

```

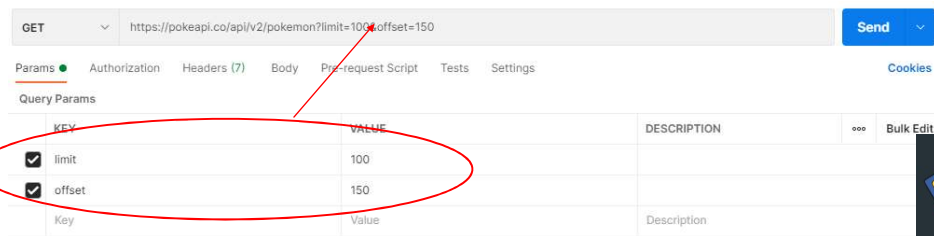
1  {
2    "count": 1118,
3    "next": "https://pokeapi.co/api/v2/pokemon?offset=20&limit=20",
4    "previous": null,
5    "results": [
6      {
7        "name": "bulbasaur",
8        "url": "https://pokeapi.co/api/v2/pokemon/1/"
9      },
10     {
11       "name": "ivysaur",
12       "url": "https://pokeapi.co/api/v2/pokemon/2/"
13     },
14     {
15       "name": "venusaur",

```



Usando postman

- ▶ Podemos usar también parámetros para realizar otras búsquedas.
- ▶ Por ejemplo, si queremos los primeros 100 pokemons, podemos poner el parámetro limit=100 y offset=0
- ▶ O si queremos los segundos 100 pokemons, limit=100 y offset=100
- ▶ Esto lo podemos agregar en la sección params, y automáticamente se agregarán a la URI



Ejercicios

- ▶ Hacer las consultas para , buscar el pokemon 25
- ▶ Buscar el pokemon que se llama bibarel
- ▶ Buscar los pokemon entre el 490 y el 500
- ▶ Buscar un pokemon que no exista y analizar la respuesta
- ▶ Toda la info para crear las request la pueden encontrar en:
- ▶ <https://pokeapi.co/docs/v2>



Usando postman

- ▶ También podemos hacer consultas a otras apis, por ejemplo json place holder:
- ▶ Su url base es:
<https://jsonplaceholder.typicode.com/>
- ▶ Y las secciones que contiene son:
- ▶ Posts
- ▶ Comments
- ▶ Albums
- ▶ Photos
- ▶ Todos
- ▶ users

{JSON} Placeholder

Usando postman

- ▶ Podríamos por ejemplo hacer una consulta a los posts:
- ▶ <https://jsonplaceholder.typicode.com/posts>

GET <https://jsonplaceholder.typicode.com/posts> Send

- ▶ O si quisiéramos un comentario en particular, por ejemplo el 200:

GET <https://jsonplaceholder.typicode.com/comments/200> Send

- ▶ También nos permite rutas anidadas, por ejemplo si quisiéramos el comentario del post numero 1:

GET <https://jsonplaceholder.typicode.com/posts/1/comments> Send

{JSON} Placeholder

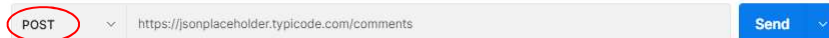
Usando postman

- ▶ Como siempre, necesitan conocer lo que se puede o no hacer con la API, la info para esto la pueden encontrar en:
- ▶ <https://jsonplaceholder.typicode.com/guide/>

{JSON} Placeholder

Usando postman

- ▶ Vamos a hacer una petición POST, para crear un recurso, en este caso, un comentario (comment):
- ▶ Para esto necesitamos la dirección URI:
- ▶ <https://jsonplaceholder.typicode.com/comments>
- ▶ Cambiar el request a POST



POST https://jsonplaceholder.typicode.com/comments Send

- ▶ Y especificar los datos que vamos a ingresar (en un json): siguiente diapositiva

{JSON} Placeholder

Usando postman

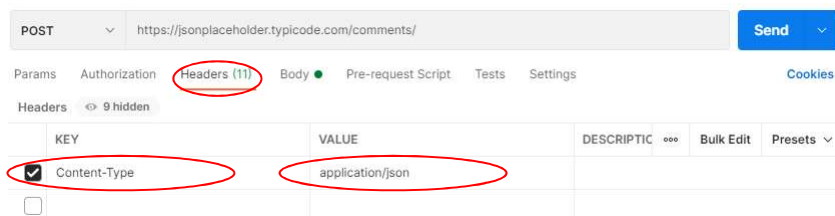
- ▶ Para ingresar los datos, lo hacemos en el **body**.
- ▶ Seleccionado la pestaña **body** y la opción **raw**:



{JSON} Placeholder

Usando postman

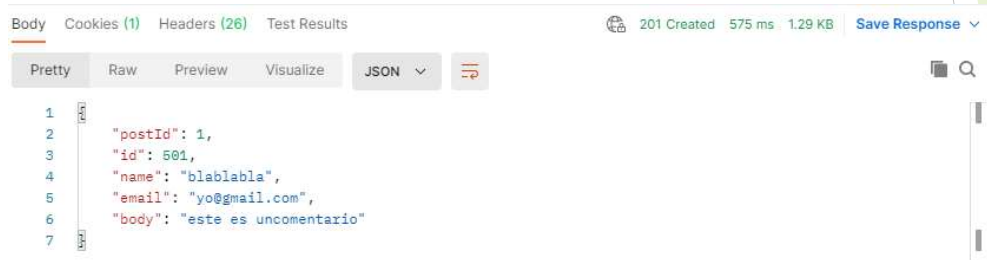
- ▶ También se debería ingresar los **headers**, que en este caso indican que la información enviada va en formato **json**:



{JSON} Placeholder

Usando postman

► La respuesta:



{JSON} Placeholder

Usando postman

- Ejercicios:
- Obtener todos los post donde el id de usuario sea 10 (usar filtros)
- Obtener todas las imágenes del álbum numero 10
- Eliminar el post número 5
- Intentar actualizar un comentario que no exista
- Cargar la lista completa de usuarios

{JSON} Placeholder

Usando postman

- ▶ REQUES es otra API que funciona de manera similar
- ▶ <https://reqres.in/>
- ▶ Su url base es: <https://reqres.in/api/>
- ▶ Y en general tiene para consultar usuarios (también hay una versión de pago)

REQ | RES

Usando postman

- ▶ Por ejemplo, podemos consultar los usuarios de la página número 2 (los resultados están paginados), usando parámetros:

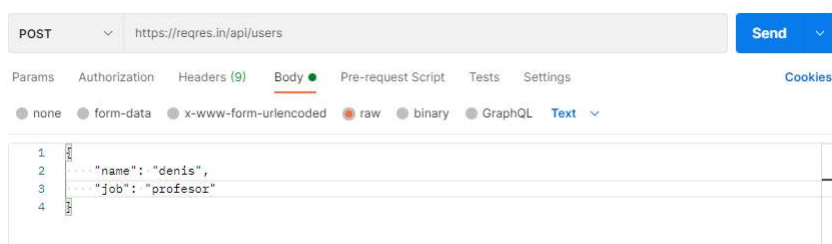
The screenshot shows the Postman interface for a GET request. The URL bar contains 'https://reqres.in/api/users?page=2'. Below the URL bar, the 'Params' tab is selected, showing a table of query parameters. The table has columns for KEY, VALUE, and DESCRIPTION. A single parameter is listed: 'page' with a value of '2'. There are checkboxes for 'Key' and 'Value' in the first row, and a 'Bulk Edit' button.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> page	2	
<input type="checkbox"/> Key	<input type="text" value="Value"/>	<input type="text" value="Description"/>

REQ | RES

Usando postman

- Podemos crear un usuario:

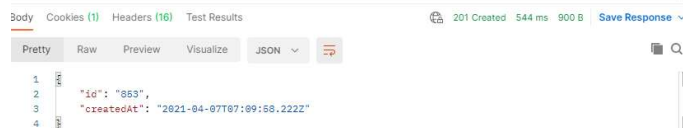
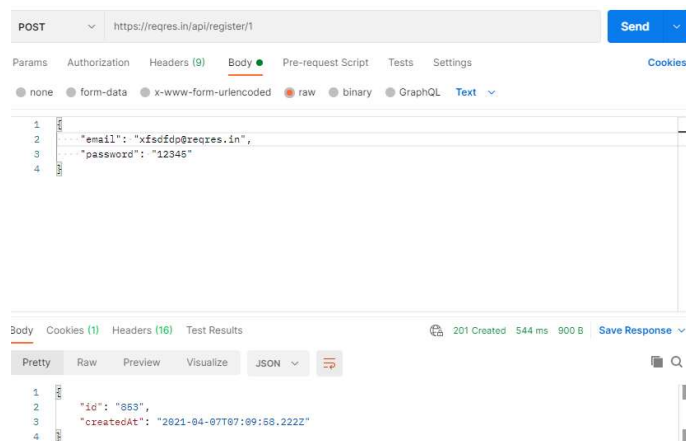


- Respuesta:



Usando postman

- Podemos crear un registro:



Usando postman

- ▶ Ejercicios
- ▶ Loguear a un usuario (usen cualquier nombre), pero sin adjuntar el password en los datos
- ▶ Actualizar el usuario numero 100
- ▶ Eliminar el usuario número 200

REQ | RES