Índice

[**Estudiando el protocolo HTTP, verbos y status** 1](#_Toc98596952)

[**Estructuras de las URLs** 2](#_Toc98596953)

[Recurso 3](#_Toc98596954)

[Colecciones 3](#_Toc98596955)

[URL 3](#_Toc98596956)

[CRUD 3](#_Toc98596957)

[Endpoints 3](#_Toc98596958)

[Entonces, ¿cuál es la buena práctica en este caso? 4](#_Toc98596959)

[Recursos anidados 5](#_Toc98596960)

[Nuestro API 5](#_Toc98596961)

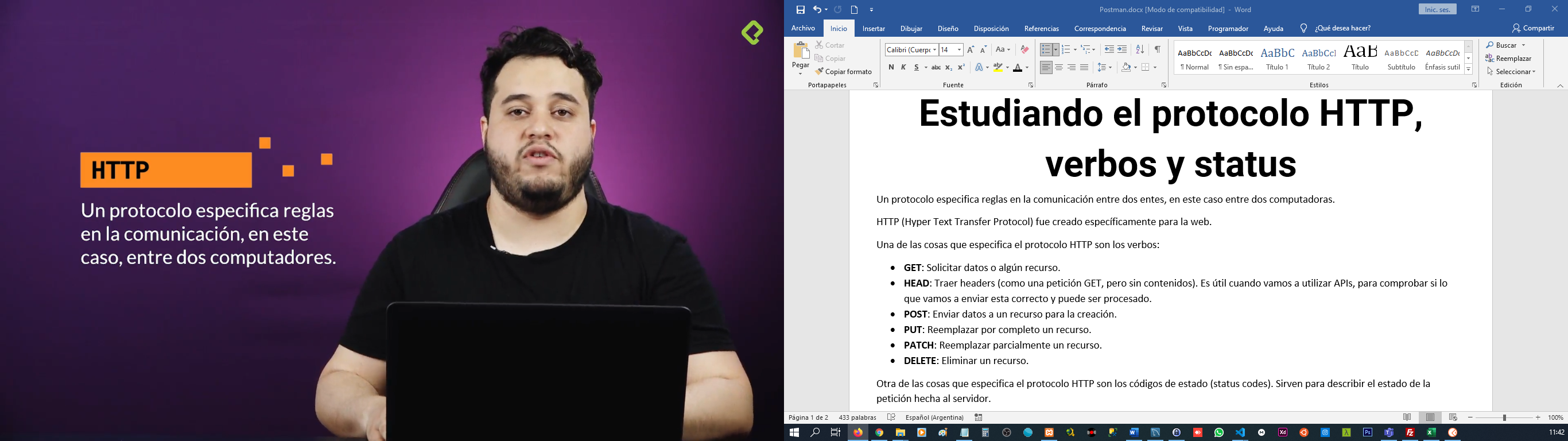
[Los endpoints que tenemos 5](#_Toc98596962)

[**Instalación de Postman** 6](#_Toc98596963)

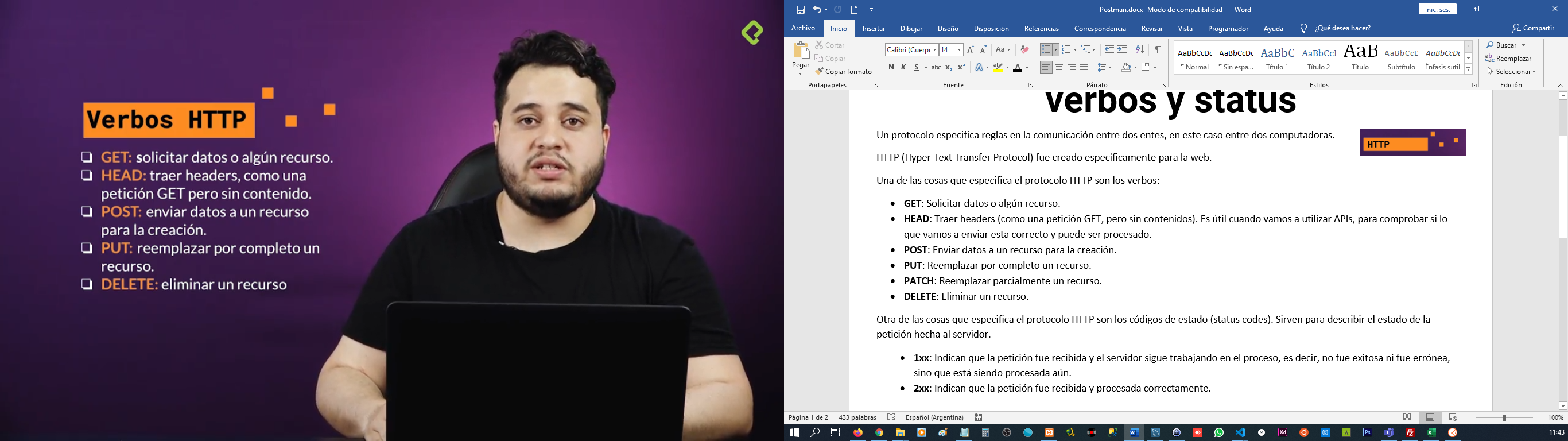
[En windows 7](#_Toc98596964)

[En MacOS 10](#_Toc98596965)

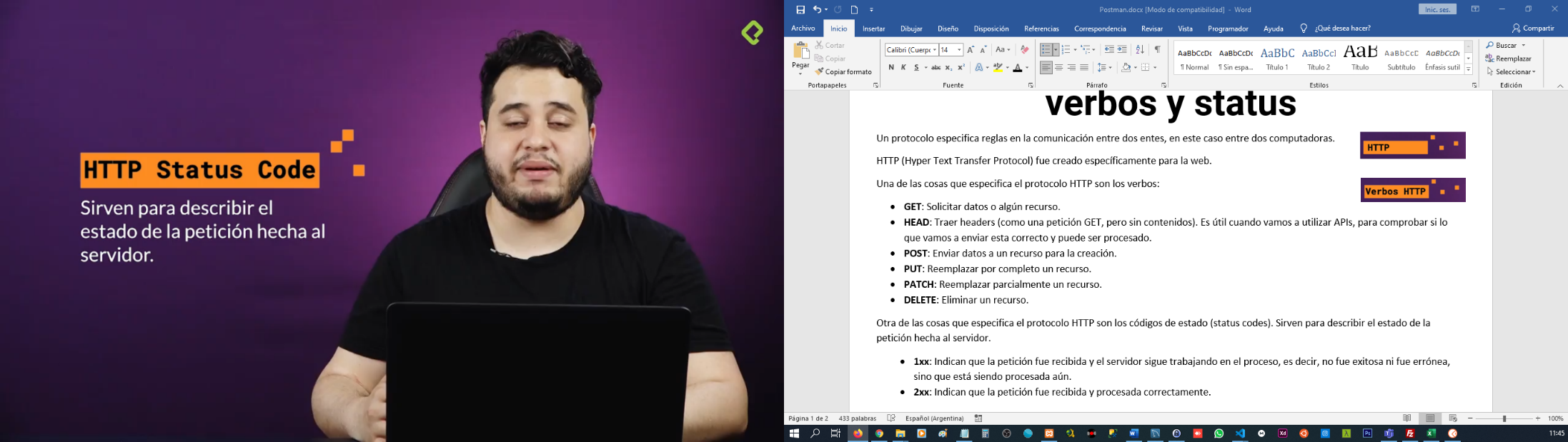
# **Estudiando el protocolo HTTP, verbos y status**

Un protocolo especifica reglas en la comunicación entre dos entes, en este caso entre dos computadoras.

HTTP (Hyper Text Transfer Protocol) fue creado específicamente para la web.

Una de las cosas que especifica el protocolo HTTP son los verbos:

* **GET**: Solicitar datos o algún recurso.
* **HEAD**: Traer headers (como una petición GET, pero sin contenidos). Es útil cuando vamos a utilizar APIs, para comprobar si lo que vamos a enviar esta correcto y puede ser procesado.
* **POST**: Enviar datos a un recurso para la creación.
* **PUT**: Reemplazar por completo un recurso.
* **PATCH**: Reemplazar parcialmente un recurso.
* **DELETE**: Eliminar un recurso.

Otra de las cosas que especifica el protocolo HTTP son los códigos de estado (status codes). Sirven para describir el estado de la petición hecha al servidor.

* **1xx**: Indican que la petición fue recibida y el servidor sigue trabajando en el proceso, es decir, no fue exitosa ni fue errónea, sino que está siendo procesada aún.
* **2xx**: Indican que la petición fue recibida y procesada correctamente.
* **3xx**: Indican que hay que tomar acciones adicionales para completar la solicitud. Por lo general estos códigos indican redirección. Generalmente en los APIs no se usan redirecciones porque no contienen estados, es decir, toda la información está contenida en una solicitud, no se guarda un estado en el servidor con una sesión.
* **4xx**: Indican errores del lado del cliente, por ejemplo: Se hizo mal la solicitud, faltan datos, headers o cualquier otro error que pueda ocurrir.
* **5xx**: Indican errores del servidor. Suelen aparecer cuando existe un fallo en la ejecución en el servidor.

Los códigos más comunes a la hora de interactuar con un API son:

* **200**: Todo está OK.
* **201**: Todo OK cuando se hizo una solicitud POST, el recurso se creó y se guardó correctamente.
* **204**: Indica que la solicitud se completó correctamente pero no devolvió información. Es muy común cuando se hacen peticiones con el verbo DELETE.
* **400**: Bad Request, algo está mal en la petición. Se nos olvidó enviar un dato o algo relacionado. Por lo general la respuesta nos especifica cuales fueron los errores a la hora de hacer la petición.
* **401**: Unauthorized, es decir, no estamos autorizados (autenticados) a realizar la petición.
* **403**: Forbidden, yo no tengo acceso a ese recurso, aunque este autenticado.
* **404**: Not Found, no existe el recurso que se está intentando acceder.
* **500**: Interna Server Error, es un error que retorna el servidor cuando la solicitud no pudo ser procesada. Por lo general, si no tenemos acceso al backend, no tenemos control sobre los errores 500 que retorna un API.

# **Estructuras de las URLs**

En un API es importante tener bien estructuradas las rutas por las cuales se usarán cada uno de los **endpoints** que contiene. Antes de entrar de lleno a explicar el API con el que trabajaremos en este curso veamos unos conceptos muy importantes a la hora de trabajar con APIs.

Recurso

Es la instancia o la representación de un objeto o la representación de algo, tiene datos y operaciones asociadas a él. Por ejemplo: course, material y video son recursos que tenemos disponibles en el API con la que trabajaremos y podemos realizar operaciones sobre ellos: crear, actualizar y eliminar.

Colecciones

Es un conjunto de recursos, por ejemplo: **courses** es una colección de course.

URL

(**U**niform **R**esource **L**ocator) es la ruta en la cual puede ser ubicado un recurso y ejecutar las operaciones sobre él.

CRUD

Siglas que hacen referencia a las operaciones básicas que se pueden ejecutar sobre un recurso:

* **C**: Create (crear)
* **R**: Read (leer)
* **U**: Update (actualizar)
* **D**: Delete (eliminar)

Endpoints

Es el **punto final** de la comunicación con un ente, en este caso, un endpoint está asociado a una URL y a las operaciones que podemos ejecutar. Este término es muy utilizado en las APIs.

Los endpoint definen operaciones que se quieren ejecutar sobre un recurso. Por ejemplo: Si queremos un conjunto de operaciones CRUD sobre **Cursos** podríamos crear los siguientes endpoints:

* **/get-all-courses**: Para obtener una colección de Cursos.
* **/add-new-course**: Para crear un nuevo recurso de Cursos.
* **/delete-all-courses**: Para eliminar todos los Cursos.

Y así sucesivamente. Pero, esto implicaría un problema. El API puede llenarse de endpoints que ejecutan una sola operación, esto no es escalable, significa que si por ejemplo el recurso **Cursos** pasa a llamarse **Clases** habría que actualizar absolutamente todas las URLs y asegurarse de ello puede convertirse en un dolor de cabeza.

Entonces, ¿cuál es la buena práctica en este caso?

Como lo vimos en la clase pasada, el protocolo HTTP especifica una serie de verbos que indican acciones. Lo ideal es que la operación que se ejecute sobre un recurso se obtenga a través del verbo HTTP de la petición y no que esté definido en el endpoint. Por ejemplo:

* **/courses**: Dependiendo del verbo HTTP se ejecutará una operación en particular. Quedaría así:
  + **GET** /courses: Trae la colección de Cursos.
  + **POST** /courses: Crea un nuevo recurso de Cursos.
  + **DELETE** /courses: Elimina todos los cursos.

De esta manera queda mucho más organizado un API. Pero, qué pasa si queremos traer no una colección de cursos como en los casos anteriores, sino un recurso en específico.

Por lo general cada recurso tiene un identificador único, un ID, es con esto que llamaremos a un endpoint para que nos retorne la información del recurso. Por ejemplo:

* **GET** /courses/**2**/: Vemos que acá se le está pasando algo que en los endpoints anteriores no veíamos, es el número 2, ese es el identificador único, de esta manera el API sabe que tiene que buscar el curso con ID 2 y retornarlo.

Así sucesivamente con cada uno de los verbos, por ejemplo: Casi nunca se hace una eliminación en masa en un API, como el ejemplo que tenemos más arriba donde se eliminan todos los cursos. Lo ideal es que se elimine un recurso individualmente y no una colección, igualmente pasa con la actualización.

Recursos anidados

Hay veces en las que un recurso depende de otro recurso, por ejemplo, comentarios depende de materiales; usualmente en los APIs las anidaciones se hacen hasta dos niveles, es decir:

* **materials/1/comments**: Estos son dos niveles
* **materials/1/comments/2/answers/**: Son tres niveles, ahí lo ideal sería entonces separar el endpoint de comentarios y ejecutar comments/2/answers/

Nuestro API

Ya he dado algunos spoilers sobre lo que nuestro API hace, es un clon de lo que Platzi es, una plataforma es donde tenemos Cursos, Materiales, Videos y Comentarios. El API es sencillo y es una prueba que utilizamos en este curso para explorar todas las capacidades que nos ofrece **Postman** para trabajar con ellos.

Una convención que se usa a la hora de documentar APIs es abstraer el endpoint de la URL del sitio al cual vamos a hacer la petición, puesto que esto al final es redundante de escribir, es decir, usualmente escribimos únicamente **/api-token-auth/** en vez de **[http://mistioweb.com/api-token-auth/](http://mistioweb.com/api-token-auth/)**.

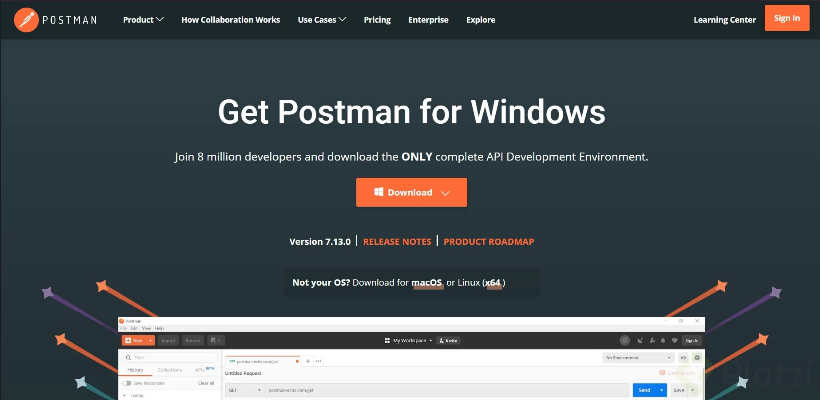
Los endpoints que tenemos

* /api-token-auth/
* /courses
* /courses/:id/
* /courses/:id/upload\_badge/
* /materials/
* /materials/:id/
* /materials/:id/comments/
* /comments/
* /comments/:id/
* /comments/:id/like/
* /comments/:id/dislike/

# **Instalación de Postman**

Postman es multiplataforma, lo que implica que no importa el sistema operativo que uses Postman funcionará.

Es esta dirección <https://www.postman.com/downloads/> puedes encontrar la versión de Postman más reciente y descargarla, por defecto, el sitio sabe desde qué sistema operativo estás accediendo y te muestra como primera opción, si no, más abajo aparecen los demás sistemas operativos que soporta Postman.



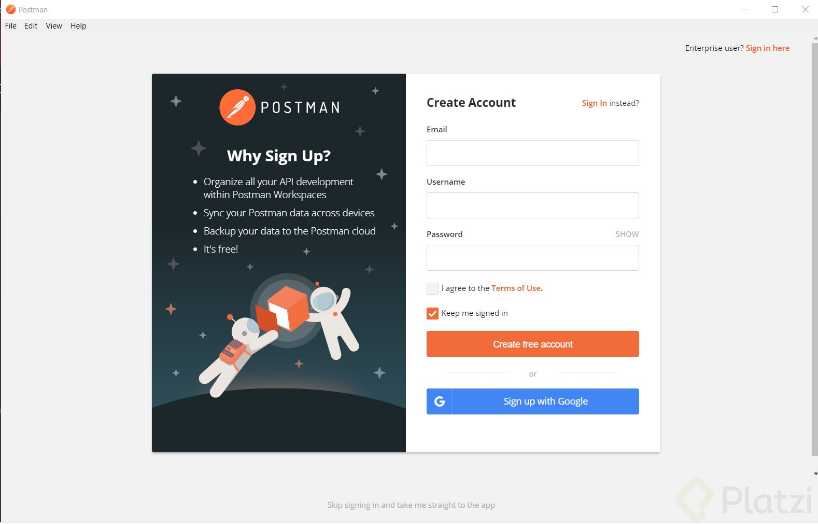
En windows

La instalación de Postman tiende a ser mucho más sencilla que la de una aplicación tradicional; no hay una serie de configuraciones que se deben aceptar o personalizar. Postman se instala en el sistema listo para utilizarse.

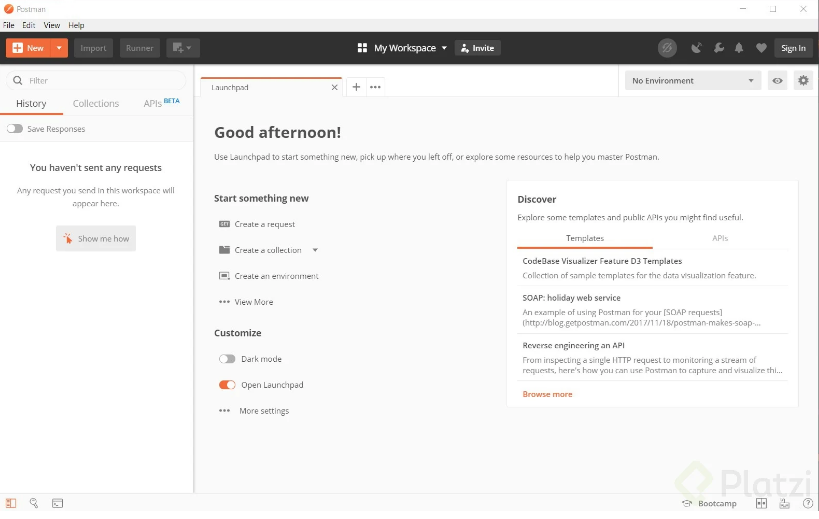
* Descargar Postman.
* Abrir el archivo que se descargó, una vez habrá aparecerá una ventana así:



* Una vez instalado te pedirá que inicies sesión o si es el caso que crees una nueva cuenta.



* Y ya está Postman listo para usarse.



En MacOS

El proceso no es muy diferente a lo que es en Windows. Descargamos la App y luego la movemos a las aplicaciones de nuestro sistema y ya está lista para usarse.