

¡Javascript!

guayerd beta hub

Agenda del día



01

APIs

Definición.

Instalar Thunder Client.

02

Ejercitación

Hacer un fetch de una orden de compra.

.

API REST

Definición.

Recursos.

Verbos.

Body.

API FETCH

Definición.

GET

POST

MOCKAPI.



daily

¿Cómo venimos?

¿Algo nos bloquea?

¿Cómo seguimos?



API

Interfaz de Programación de Aplicaciones



SPRINT 3

API



Una API (Interfaz de Programación de Aplicaciones) es una implementación de reglas y protocolos que permite la comunicación entre programas o consumir servicios.

Ejemplo

The screenshot shows the PokeAPI homepage. At the top, there's a red header bar with the "PokeAPI" logo and navigation links for Home, About, API v2, and GraphQL v1beta. Below the header is a dark banner featuring the "PokeAPI" logo and the text "The RESTful Pokémon API" and "Serving over 2.5 billion API calls each month!". The main content area has a light green background. It features a "Try it now!" button, a search bar with the URL "https://pokeapi.co/api/v2/pokemon/ditto", and a "Submit" button. Below the search bar, there's a hint text: "Need a hint? Try pokemon/ditto, pokemon-species/aegislash, type/3, ability/battle-armor, or pokemon?limit=10000&offset=0.". A "Direct link to results" button is also present. Under the "Try it now!" section, there's a heading "Resource for ditto" followed by a JSON snippet:

```
▼ abilities: [] 2 items
  ▼ 0: {} 3 keys
    ▼ ability: {} 2 keys
      name: "limber"
      url: "https://pokeapi.co/api/v2/ability/7/"
```

Instalar Thunder Client en VSCode

The screenshot shows the VS Code Marketplace interface. On the left, a search bar contains the text 'thu'. Below it, a list of extensions is displayed:

- Thunder Client** by Thunder Client: Lightweight Rest API Client for VS Code. Rating 5 stars, 4.603.542 installs. Status: Instalar.
- Tacky The Thumbtack** by FrogRats: Tacky the Thumbtack is the world's wor... Rating 5 stars, 151 installs. Status: Instalar.
- Thunder** by Steve-DevOps: Snippets for lwc (Salesforce). Rating 5 stars, 3K installs. Status: Instalar.
- Thunder** by Ken T Ekeoha: Type quickly. Type freely. Rating 5 stars, 7K installs. Status: Instalar.
- Thunder Theme** by jhonx: Rating 5 stars, 127 installs. Status: Instalar.
- Blue Thunder** by Charles Assuncao: Rating 5 stars, 5K installs. Status: Instalar.

On the right, the product page for 'Thunder Client' is shown:

Thunder Client v2.29.0

Thunder Client | ⚡ 4.603.542 | ★★★★☆(409)

Lightweight Rest API Client for VS Code

Deshabilitar | **Desinstalar** | Actualización automática

DETALLES **CARACTERÍSTICAS** **REGISTRO DE CAMBIOS**

Thunder Client

Thunder Client is a lightweight Rest API Client Extension for VS Code, hand-crafted by [Ranga Vadhineni](#) with a focus on **simplicity, clean design and local storage**.

- Featured on [Product Hunt](#)
- Website - www.thunderclient.com
- Documentation: docs.thunderclient.com
- Support: github.com/rangav/thunder-client-support

Servidores



Clients



API REST

Representational State Transfer





SPECIAL
DELIVERY

API REST

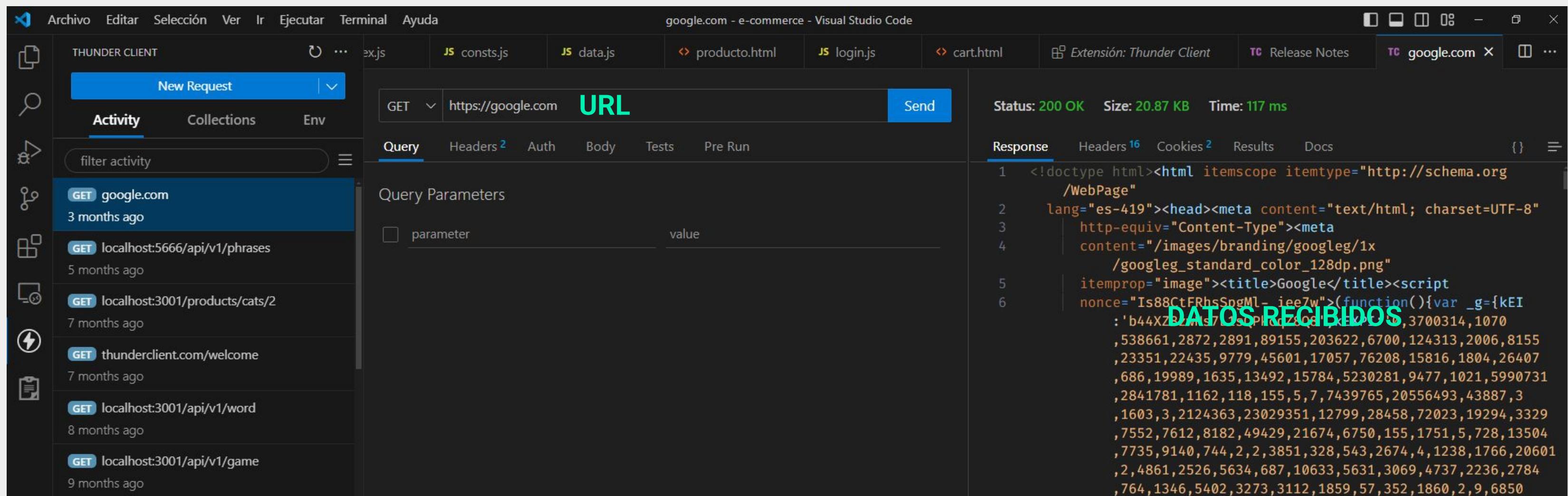
Es una arquitectura para diseñar servicios web que permiten comunicar datos o información entre clientes y servidores.

Está compuesto por:

- Recursos
- Métodos o verbos
- Parámetros
- Headers
- Body

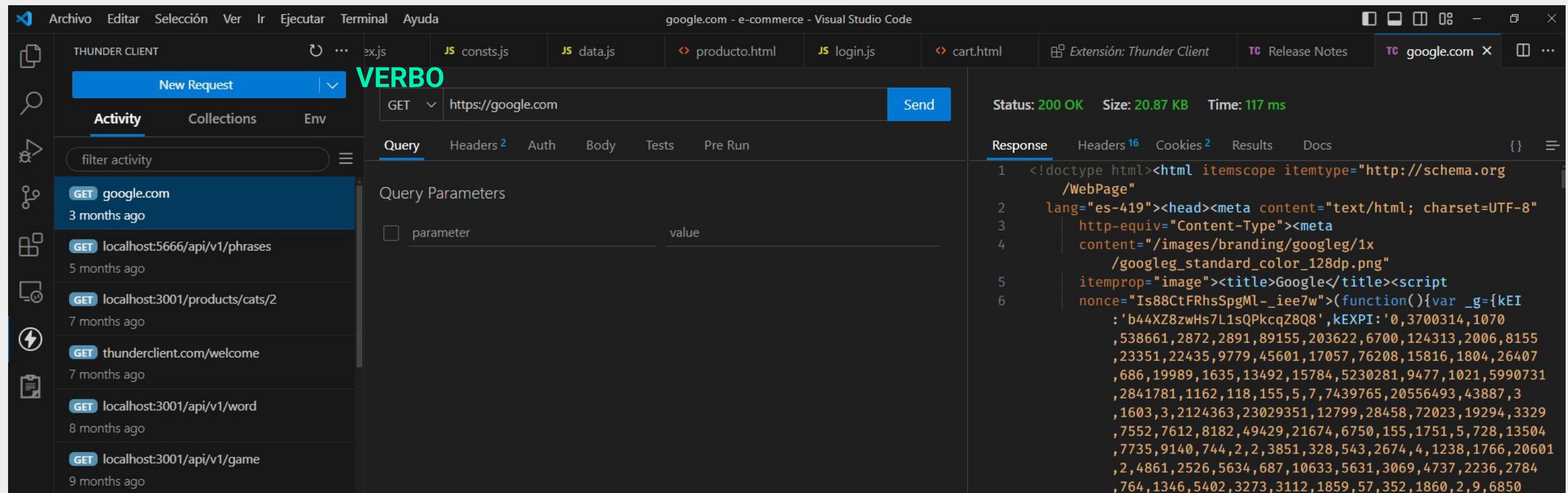
Recursos

Nos comunicamos con una API a través de una URL y obtenemos los recursos representados como datos utilizables.



Métodos o verbos: GET

Permiten comunicar distintas intenciones al servidor. El más usado es GET, que trae directamente un recurso. Lo usamos comúnmente en la barra de direcciones del navegador.



Métodos o verbos: POST

Permite enviar información al servidor para crear un recurso en el servidor. Enviamos la información en formato JSON y usamos el campo body.

The screenshot shows a Postman interface with the following details:

- VERBO:** POST
- URL:** https://64557b55f803f34576439459.mockapi.io/cats
- Body:** JSON (selected tab)
- JSON Content:** BODY
- Format:** JSON
- Request Body:**

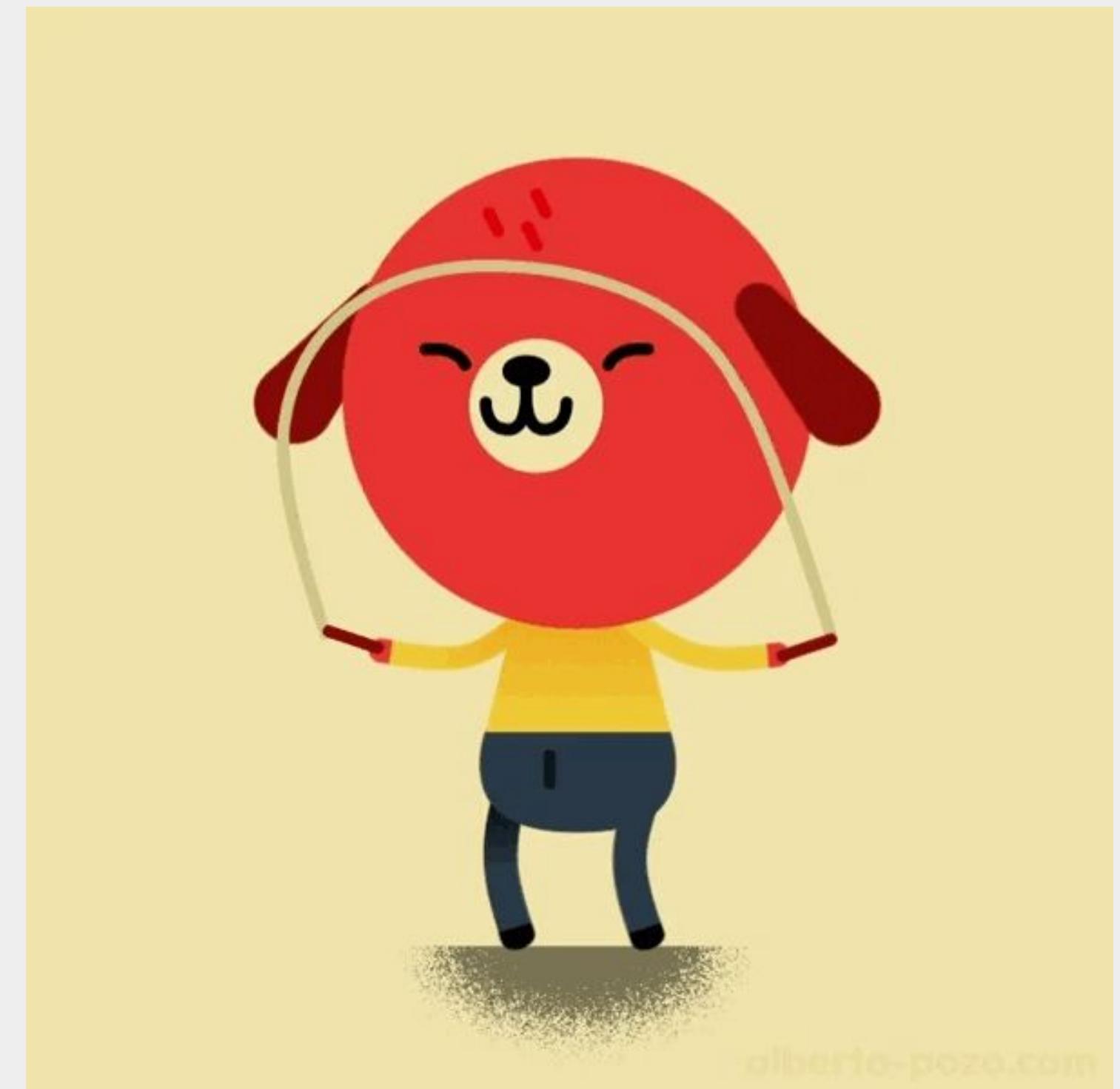
```
1 {  
2   "name": "Wen",  
3   "avatar": "https://ipfs.everipedia.org/ipfs  
/QmTWLHFwyNvzbQBd9LfcbGiPu1rrWpX6rchEv3YiputpUY"  
4 }
```
- Send:** Button
- Status:** 201 Created
- Size:** 140 Bytes
- Time:** 831 ms
- Response:** Headers (13), Cookies, Results, Docs
- Response Content:**

```
1 {  
2   "name": "Wen",  
3   "avatar": "https://ipfs.everipedia.org/ipfs  
/QmTWLHFwyNvzbQBd9LfcbGiPu1rrWpX6rchEv3YiputpUY",  
4   "stock": 94,  
5   "category": {},  
6   "id": "58"  
7 }
```



API FETCH

Navegador





API FETCH

Es una API de navegador que permite comunicarnos con APIS (servidores) desde el navegador.

De este modo integramos nuestras aplicaciones con datos de la **nube**.

Fetch funciona asincrónicamente, es una **promesa**, por lo cual, encadena los métodos: `then`, `catch`, `finally`.

GET

El verbo get del método fetch, es el verbo predeterminado, por eso no se declara. Fetch recibe como primer argumento la URL. En el primer then, transformamos lo que devuelve el servidor en un formato que Javascript puede entender. El segundo then obtiene la data.

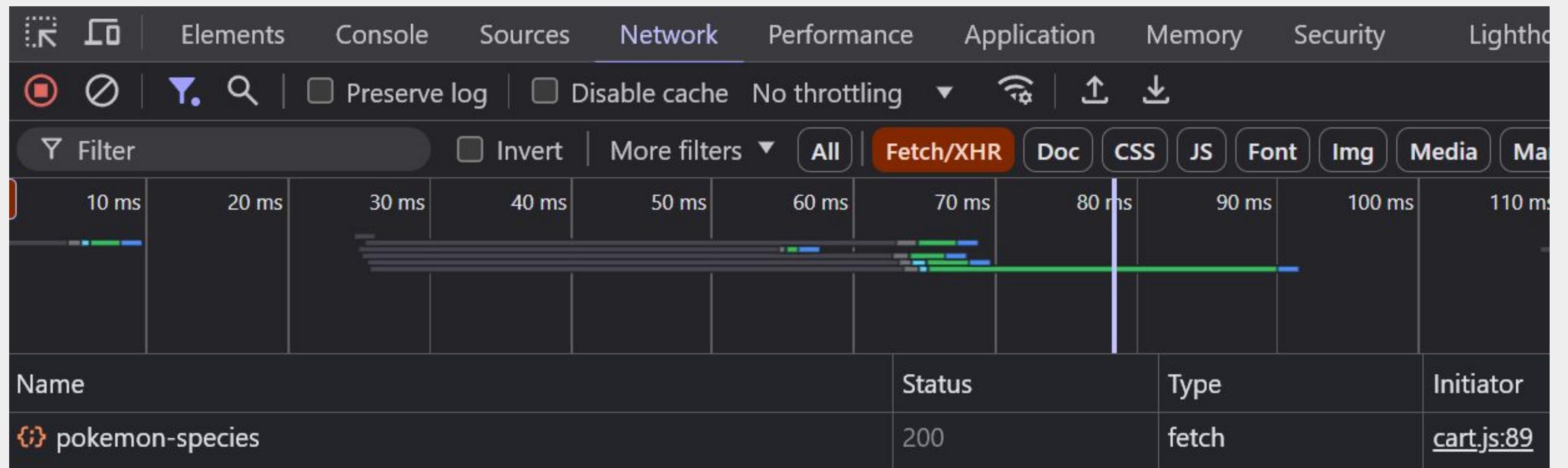


```
fetch("https://pokeapi.co/api/v2/pokemon-species")
  .then(response => response.json())
  .then(data => console.log(data))
```

GET

En Dev Tools, en la pestaña Red (Network) podemos ver la petición.

El status 200 indica que la comunicación funcionó.



GET

Si damos clic en la petición podemos ver información en varias pestañas.

Name	X	Headers	Preview	Response	Initiator	Timing
pokemon-species	X	Headers	Preview	Response	Initiator	Timing
▼ General						
		Request URL:		https://pokeapi.co/api/v2/pokemon-species		
		Request Method:		GET		
		Status Code:		200 OK (from disk cache)		
		Remote Address:		172.67.195.193:443		
		Referrer Policy:		strict-origin-when-cross-origin		
▼ Response Headers						
		Accept-Ranges:		bytes		
		Access-Control-Allow-Origin:		*		
		Alt-Svc:		h3=":443"; ma=86400		
		Cache-Control:		public, max-age=86400, s-maxage=86400		

GET

Si damos clic en la petición podemos ver información en varias pestañas.

Name	X Headers	Preview	Response	Initiator	Timing
pokemon-species			<pre>▼ {count: 1025, next: "https://pokeapi.co/api/v2/pokemon-species?offset=20&limit=20", previous: null,...} count: 1025 next: "https://pokeapi.co/api/v2/pokemon-species?offset=20&limit=20" previous: null ▼ results: [{name: "bulbasaur", url: "https://pokeapi.co/api/v2/pokemon-species/1/"},...] ► 0: {name: "bulbasaur", url: "https://pokeapi.co/api/v2/pokemon-species/1/"} ► 1: {name: "ivysaur", url: "https://pokeapi.co/api/v2/pokemon-species/2/"} ► 2: {name: "venusaur", url: "https://pokeapi.co/api/v2/pokemon-species/3/"} ► 3: {name: "charmander", url: "https://pokeapi.co/api/v2/pokemon-species/4/"} ► 4: {name: "charmeleon", url: "https://pokeapi.co/api/v2/pokemon-species/5/"} ► 5: {name: "charizard", url: "https://pokeapi.co/api/v2/pokemon-species/6/"} ► 6: {name: "squirtle", url: "https://pokeapi.co/api/v2/pokemon-species/7/"} ► 7: {name: "wartortle", url: "https://pokeapi.co/api/v2/pokemon-species/8/"} ► 8: {name: "blastoise", url: "https://pokeapi.co/api/v2/pokemon-species/9/"} ► 9: {name: "caterpie", url: "https://pokeapi.co/api/v2/pokemon-species/10/"} ► 10: {name: "metapod", url: "https://pokeapi.co/api/v2/pokemon-species/11/"}</pre>		
1 / 11 requests 0 B / 1					

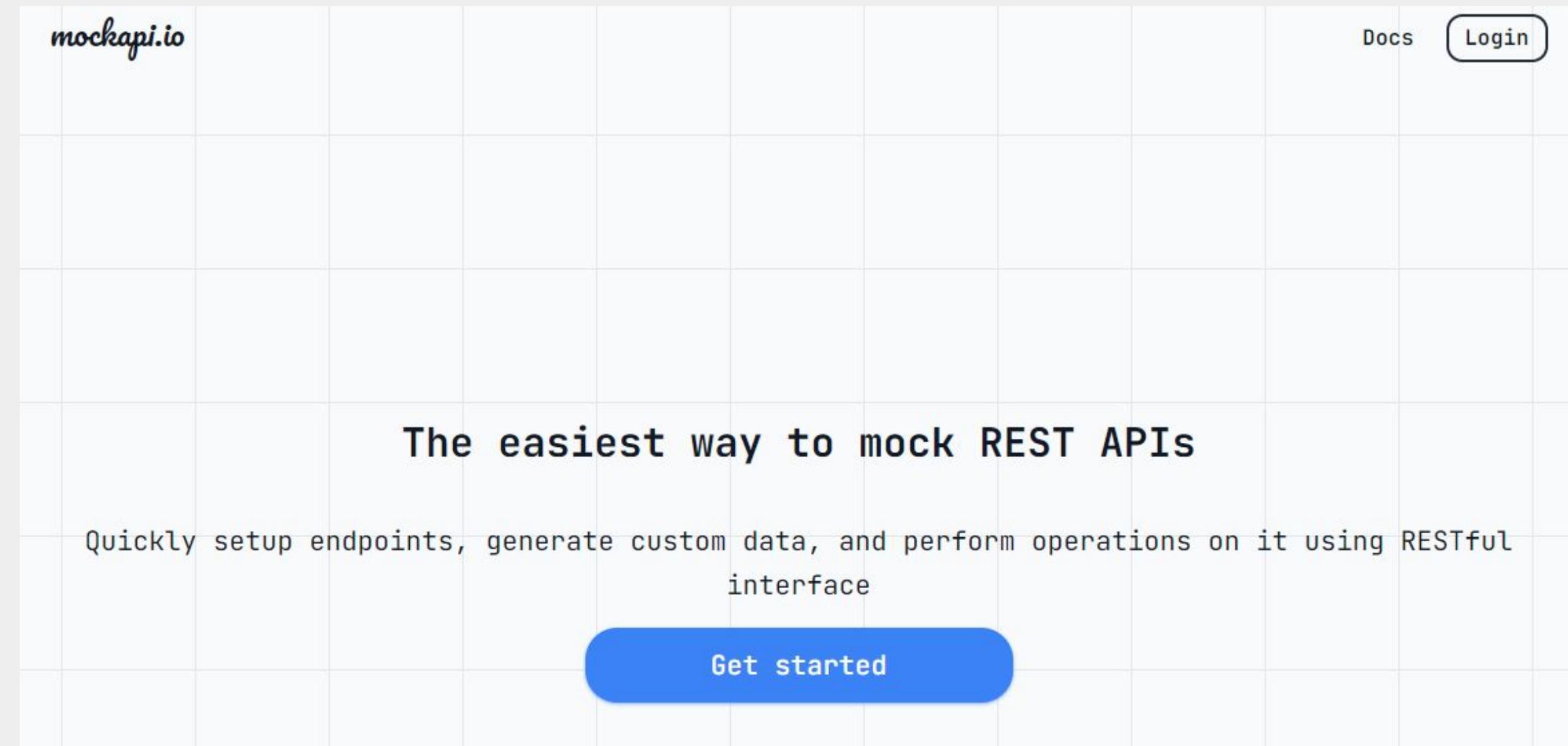
POST

En el segundo argumento del método fetch se incluye un objeto de opciones. El verbo POST lo especificamos con la clave method y el body con la data que queremos enviar stringificada.

```
● ● ●  
const recurso = {  
  name: "Wen",  
  avatar: "https://ipfs.everipedia.org/ipfs/QmTWLHFwyNvvbQBd9LfcbGiPu1rrWpX6rchEv3YiputpUY",  
}  
  
fetch("https://64557b55f803f34576439459.mockapi.io/cats", {  
  method: "POST",  
  body: JSON.stringify(recurso),  
})  
  .then(response => response.json())  
  .then(data => console.log(data))
```

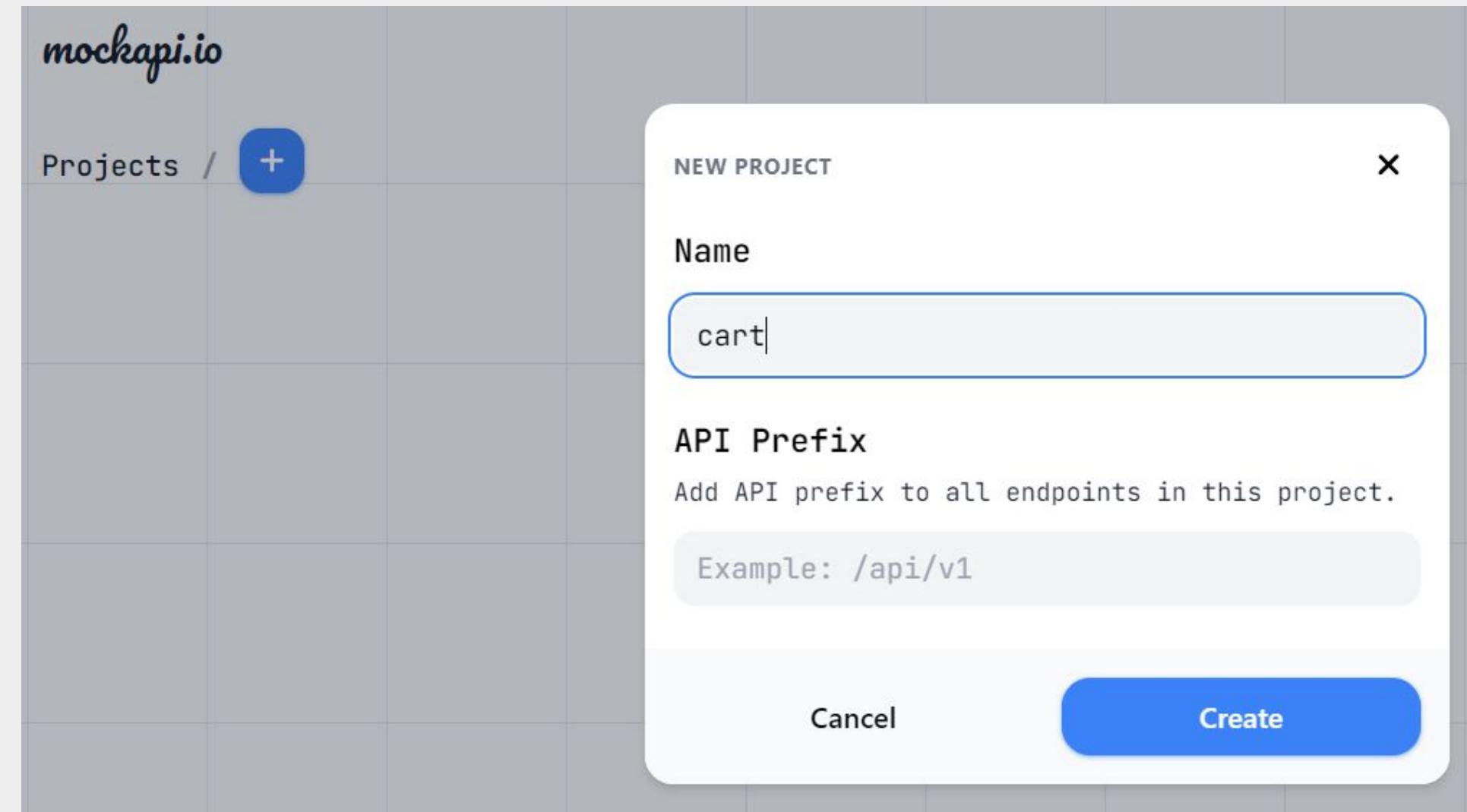
MOCKAPI

Creamos una API de prueba. Ver: <https://mockapi.io/projects>



MOCKAPI

Damos un clic al botón **+** y ponemos en name: cart.



MOCKAPI

Damos un clic a la flecha de la derecha.



MOCKAPI

Agregamos el campo items como Object y el campo user como string.

Schema
Define Resource schema, it will be used to generate mock data.

id Object ID
createdAt Faker.js date.recent
items Object
user Faker.js random.word

Object template

+ Search ...
Faker.js
String

To define more complex schema, click on the "Object template" button. You can reference

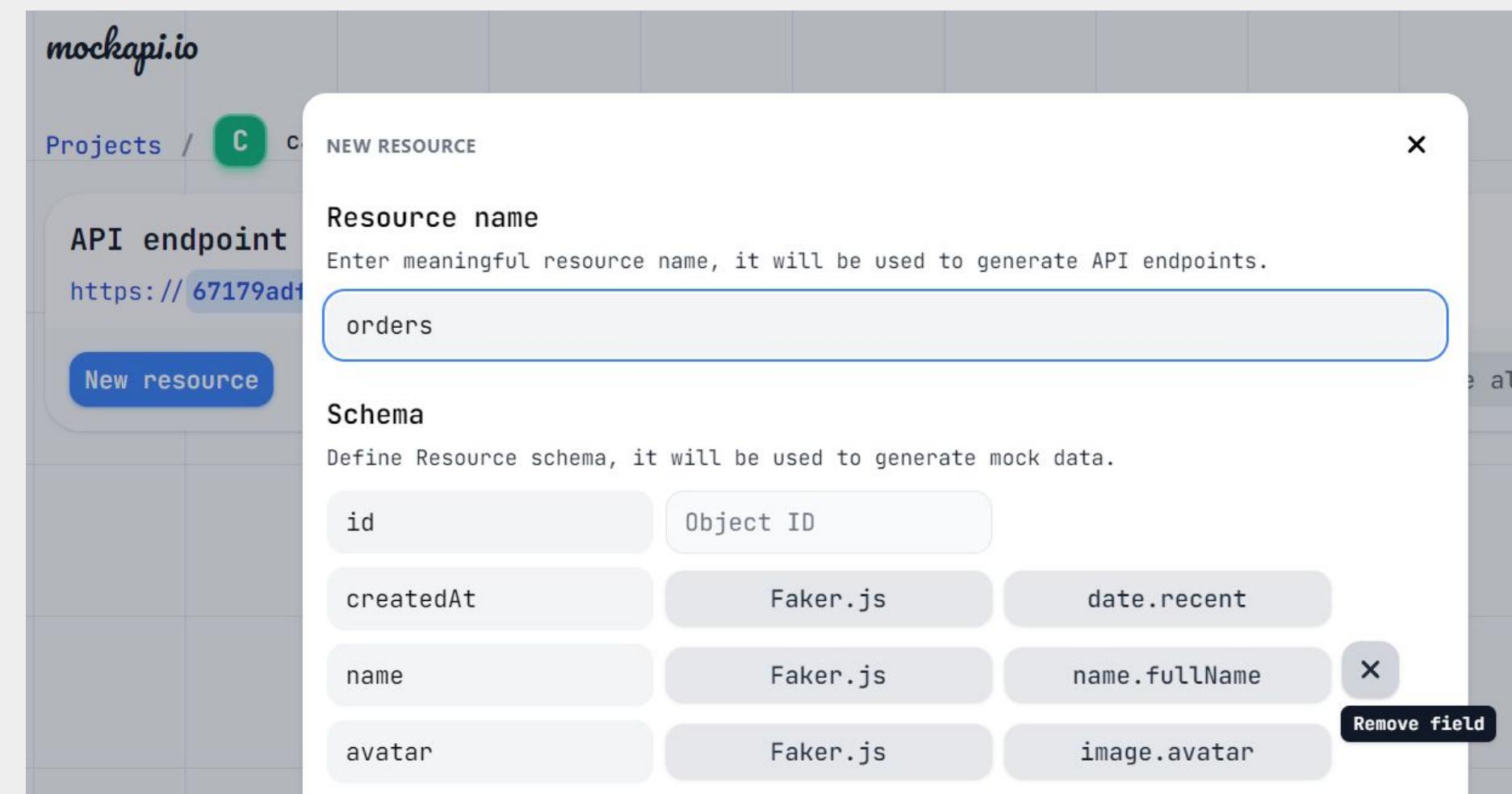
Requires subscription [See plans](#)



MOCKAPI

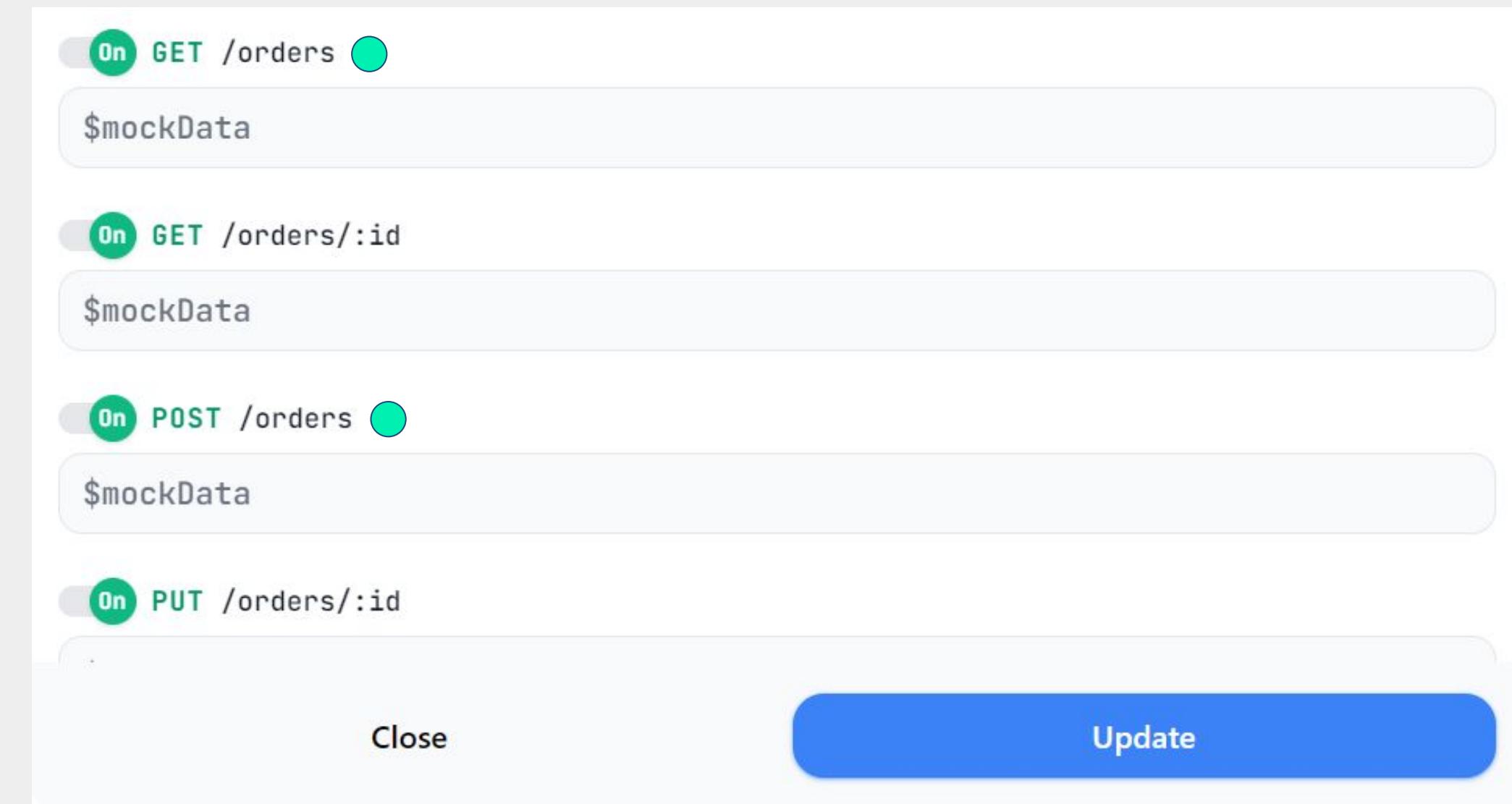
Damos un clic en New Resource.

En Resource name ponemos: orders y removemos el campo name y avatar.



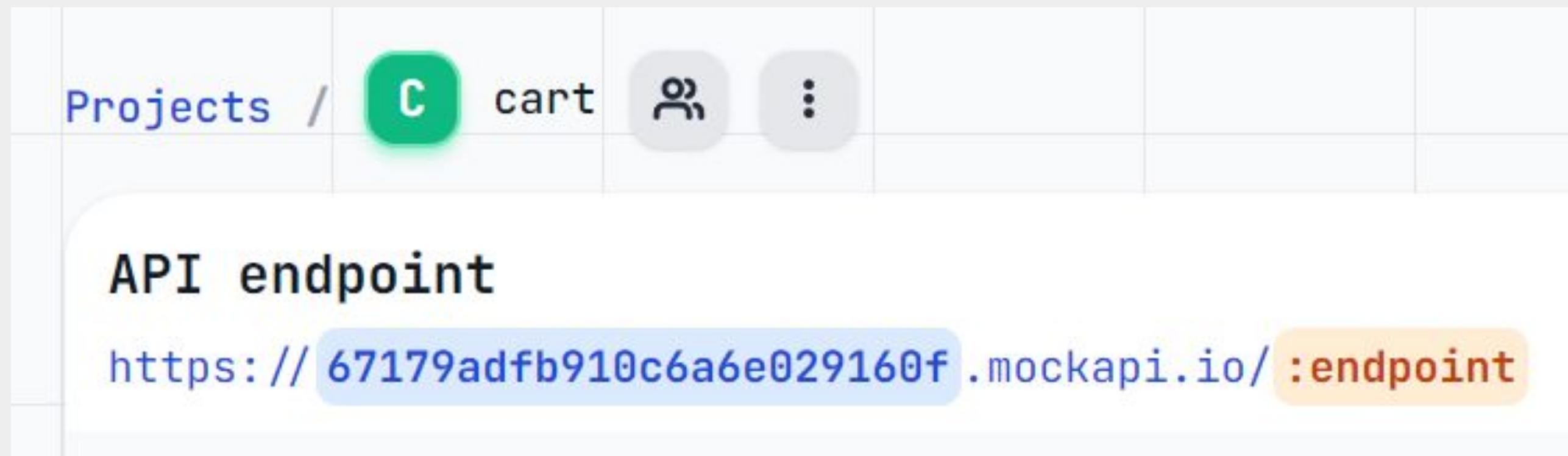
MOCKAPI

Hacemos scroll y vemos los métodos. Finalmente damos clic en Update.



MOCKAPI

Endpoints.



MOCKAPI GET

GET <https://67179adfb910c6a6e029160f.mockapi.io/orders> Send

Status: 200 OK Size: 528 Bytes Time: 678 ms

Query Headers 2 Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 [  
2 {  
3   "createdAt": "2024-10-22T00:41:58.027Z",  
4   "items": [  
5     {  
6       "id": 1,  
7       "title": "Heroes of Telemark, The ",  
8       "detail": "Nam ultrices, libero non mattis pulvinar,  
9         nulla pede ullamcorper augue, a suscipit nulla elit  
10        ac nulla. Sed vel enim sit amet nunc viverra dapibus.  
11        Nulla suscipit ligula in lacus.\n\nCurabitur at ipsum  
12        ac tellus semper interdum. Mauris ullamcorper purus  
13        sit amet nulla. Quisque arcu libero, rutrum ac,  
14        lobortis vel, dapibus at, diam.",  
15       "price": 8,  
16       "stock": 56,  
17       "category": "Moderno",  
18       "quantity": 2  
19     }  
20   ],  
21   "user": "pato@duck.com",  
22   "id": "1"  
23 }  
24 ]
```

MOCKAPI POST

The screenshot shows a POST request to the URL `https://67179adfb910c6a6e029160f.mockapi.io/orders`. The request body is a JSON object representing an order. The response is a 201 Created status with a size of 526 bytes and a time of 707 ms. The response body is also a JSON object containing the created order details.

Request	Response
<p>POST <input type="button" value="▼"/> https://67179adfb910c6a6e029160f.mockapi.io/orders <input type="button" value="Send"/></p> <p>Query Headers ² Auth <input checked="" type="radio"/> Body ¹ Tests Pre Run</p> <p><input checked="" type="radio"/> JSON <input type="radio"/> XML <input type="radio"/> Text <input type="radio"/> Form <input type="radio"/> Form-encode <input type="radio"/> GraphQL <input type="radio"/> Binary</p> <p>JSON Content <input type="button" value="Format"/></p> <pre>1 { 2 "user": "pato@duck.com", 3 "items": [4 { 5 "id": 1, 6 "title": "Heroes of Telemark, The", 7 "detail": "Nam ultrices, libero non mattis pulvinar, nulla pede ullamcorper augue, a suscipit nulla elit ac nulla. Sed vel enim sit amet nunc viverra dapibus. Nulla suscipit ligula in lacus.\n\nCurabitur at ipsum ac tellus semper interdum. Mauris ullamcorper purus sit amet nulla. Quisque arcu libero, rutrum ac, lobortis vel, dapibus at, diam.", 8 "price": 8, 9 "stock": 56, 10 "category": "Moderno", 11 "quantity": 2 12 } 13 }</pre>	<p>Status: 201 Created Size: 526 Bytes Time: 707 ms</p> <p>Response Headers ¹³ Cookies Results Docs { } <input type="button" value="≡"/></p> <pre>1 { 2 "createdAt": "2024-10-22T00:41:58.027Z", 3 "items": [4 { 5 "id": 1, 6 "title": "Heroes of Telemark, The", 7 "detail": "Nam ultrices, libero non mattis pulvinar, nulla pede ullamcorper augue, a suscipit nulla elit ac nulla. Sed vel enim sit amet nunc viverra dapibus. Nulla suscipit ligula in lacus.\n\nCurabitur at ipsum ac tellus semper interdum. Mauris ullamcorper purus sit amet nulla. Quisque arcu libero, rutrum ac, lobortis vel, dapibus at, diam.", 8 "price": 8, 9 "stock": 56, 10 "category": "Moderno", 11 "quantity": 2 12 } 13], 14 "user": "pato@duck.com", 15 "id": "1" 16 }</pre>







¡Manos a la obra!

SPRINT 3

Fetch

Entregable



- Crear una mockapi por grupo.
- Al dar clic en el botón Checkout de la página cart enviar un fetch para crear un recurso.
- Si **funciona**, mostrar un SweetAlert que incluya el email del usuario y el número de orden creada. Vaciar el carrito.
- Si **falla** qué catch muestre un mensaje al usuario con SweetAlert.

🔍 retro

¿Cómo nos fué?

¿Qué cosas no quedaron claras y
necesitamos repasar la próxima?

