

Ejemplo típico de una API REST

- Ajax generalmente se usa junto con una API REST que nos permite consumir servicios alojados en un servidor
- Como ejemplo típico de los verbos (o acciones) que se pueden utilizar en una API REST tenemos:

API	Descripción	Cuerpo de la solicitud	Cuerpo de la respuesta
GET /api/Personas	Obtener todas las Personas	Ninguna	Matriz de tareas Personas
GET /api/Personas/{id}	Obtener un elemento por identificador	None	Persona
POST /api/Personas	Añadir de un nuevo elemento	Persona	Persona
PUT /api/Personas/{id}	Actualizar un elemento existente	Persona	None
DELETE /api/Personas/{id}	Eliminar un elemento	None	None

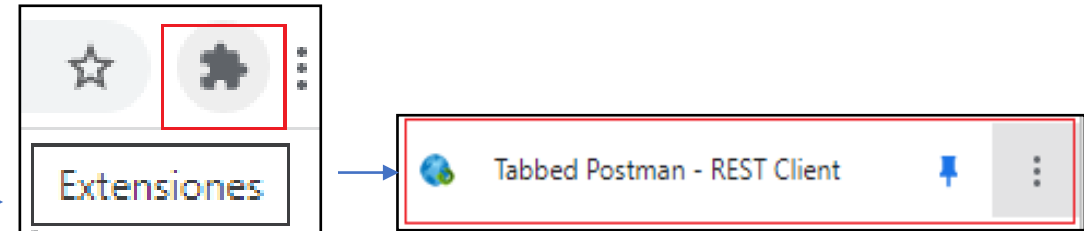
Datos de ejemplo API REST

- En <https://jsonplaceholder.typicode.com/> tenemos una API REST de prueba que permite obtener diferentes objetos como: **posts, comments, albums, photos, todos y users**.
- Ejemplos de lectura:
 - GET [/posts](#) -> Obtiene todos los **posts**
 - GET [/posts/1](#) -> Obtiene el primer **post**
 - GET [/posts/1/comments](#) -> Obtiene los **comments** cuyo **postId** es 1
 - GET [/comments?postId=1](#) -> Igual a la anterior
 - GET [/posts?userId=1](#) -> Obtiene los **posts** del **user 1**
- También se puede utilizar POST para inserciones, PUT o PATCH para modificaciones y DELETE para borrados.

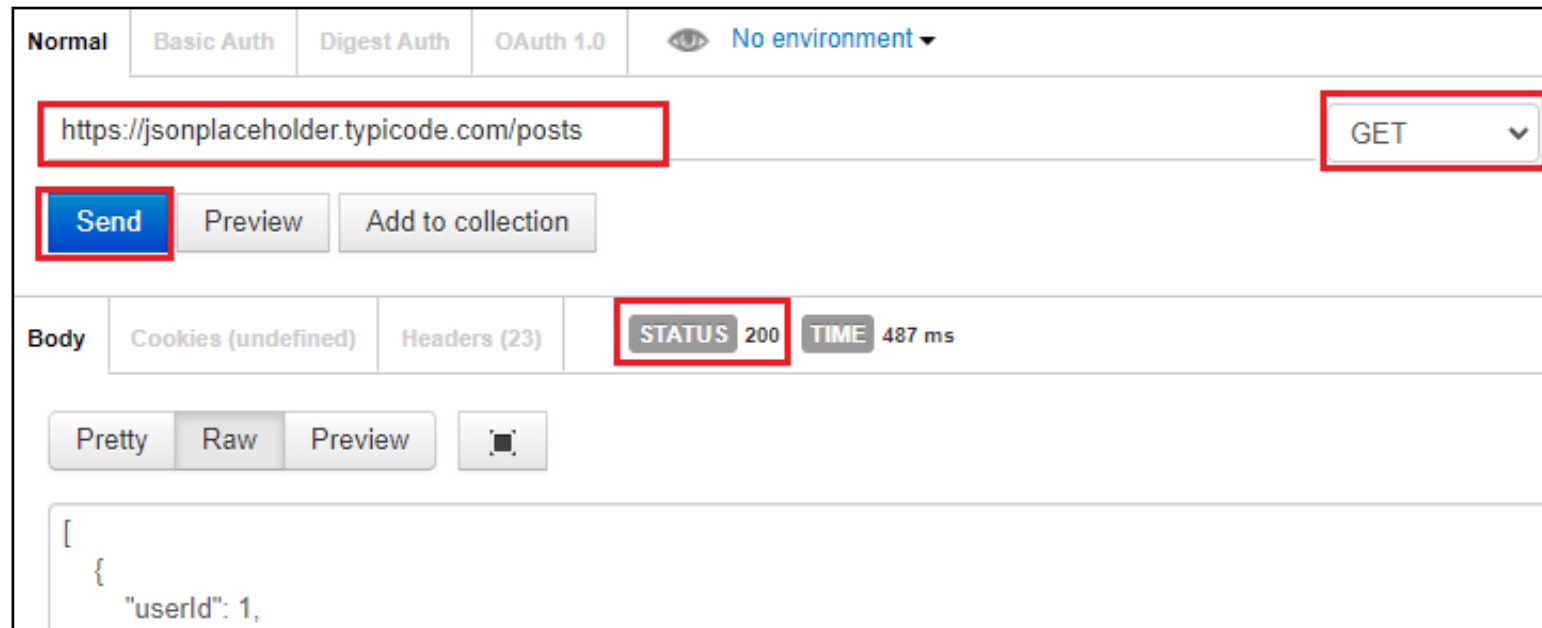
Cientes Rest. Tabbed Postman.

- Hay muchos clientes Rest para consultar una API. Ej: PostMan, Insomnia, etc...
- Una posibilidad sencilla es utilizar la extensión de Chrome “Tabbed Postman”.
- Buscar “Tabbed Postman” en Google y aparecerá en Chrome web store. Seleccionar y “Añadir a Chrome”.

- En el icono “Extensiones” de la parte superior izquierda de Chrome seleccionar “Tabbed Postman – REST Client”:



- En la ventana que aparece ya podemos hacer una consulta GET ejemplo:

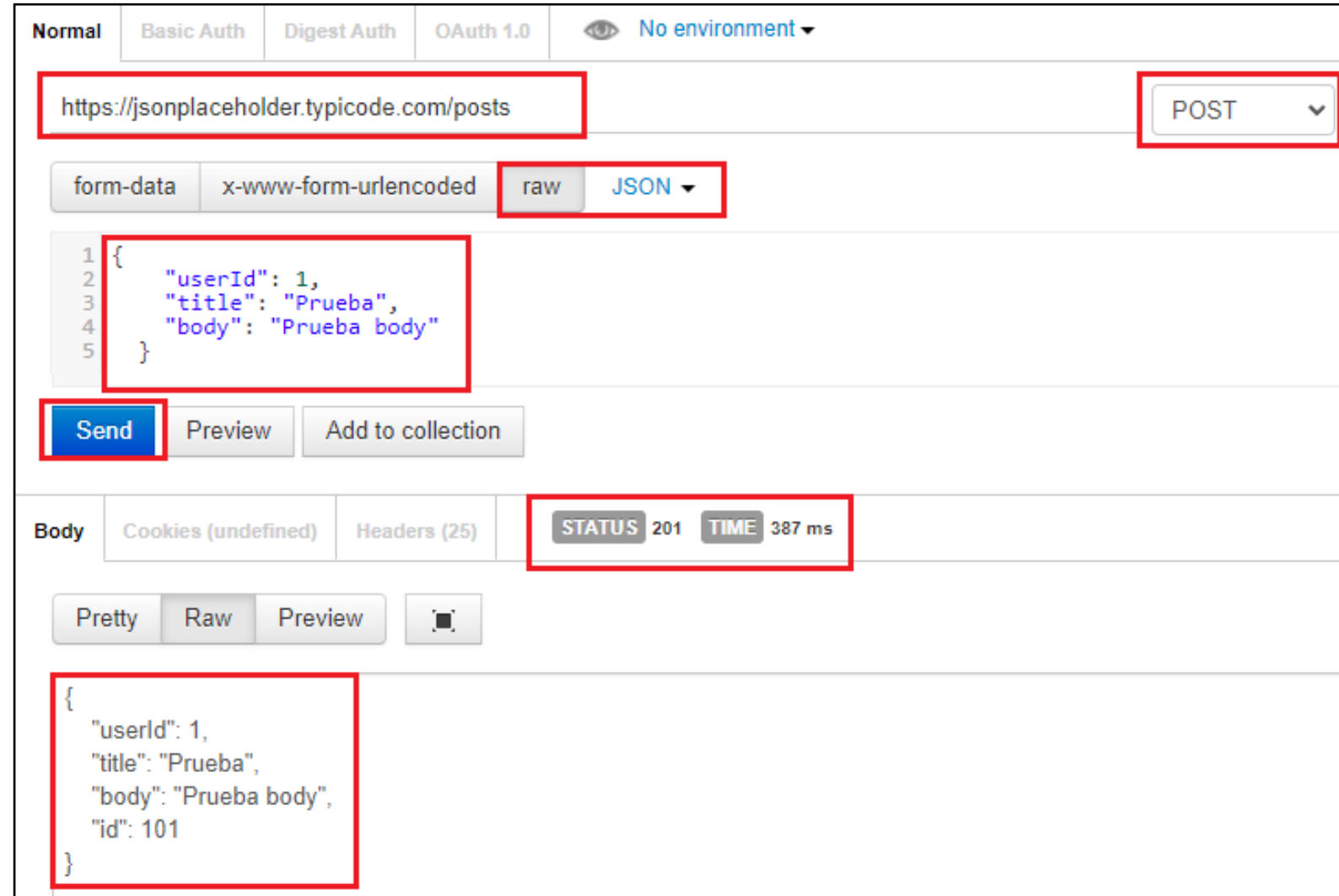


Cientes Rest. Tabbed Postman.

- Para crear un nuevo “posts” deberemos mandar el json con los campos (sin incluir el id) siguiendo los siguientes pasos:

- 1.- Introducir la URL
- 2.- Seleccionar “POST” como método de envío
- 3.- Seleccionar “raw” y JSON para el cuerpo.
- 4.- Escribir el JSON con los datos (sin el “id”)
- 5.- Pulsar “Send”
- 6.- Observar el status y la respuesta.

Realmente se simula la operación porque el servidor no admite modificaciones.



Servidor JSON con una API REST

- Fuente e instrucciones: <https://github.com/typicode/json-server>
- Instalar json-server con el comando: **npm install -g json-server**
- Abrir <https://jsonplaceholder.typicode.com/db> y guardar en **db_original.json**
- Copiar el fichero **db_original.json** al directorio de la web actual como **db.json**

```
G:\_jquery\EjerciciosJQuery>json-server --watch db.json

\{^_^}/ hi!

Loading db.json
Done

Resources
http://localhost:3000/posts
http://localhost:3000/comments
http://localhost:3000/albums
http://localhost:3000/photos
http://localhost:3000/users
http://localhost:3000/todos

Home
http://localhost:3000

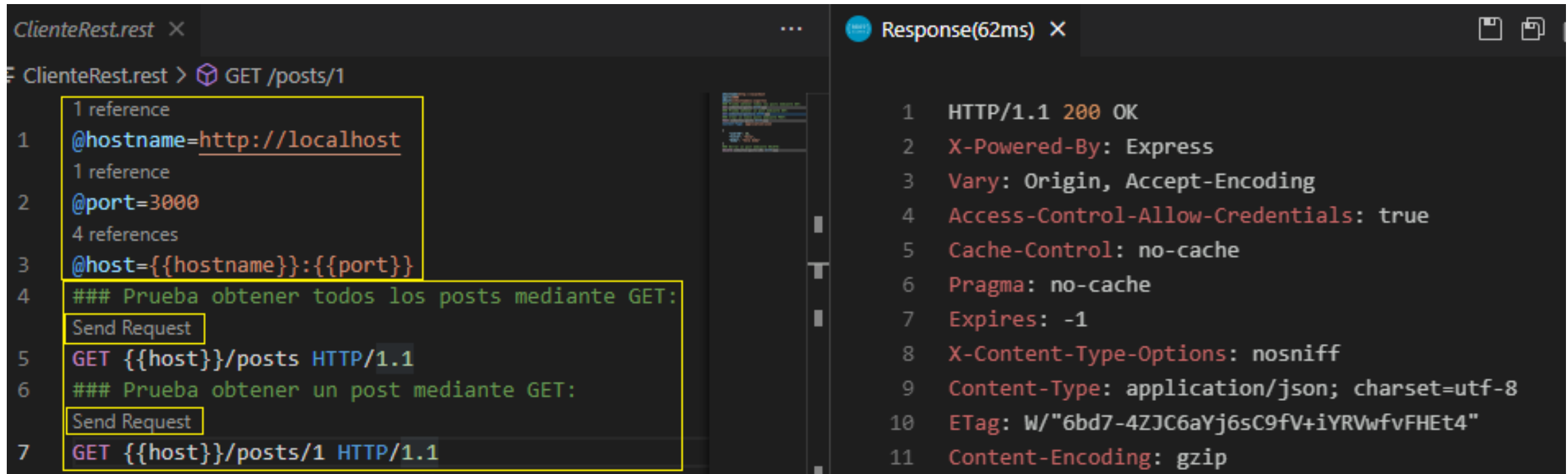
Type s + enter at any time to create a snapshot of the database
Watching...

GET /posts 200 17.793 ms - -
```

- Lanzar el servidor con el comando: **json-server --watch db.json**
- Una vez arrancado el servidor se pueden enviar peticiones para leer y modificar.
- Es mejor utilizar el servidor local porque en Internet mediante <https://jsonplaceholder.typicode.com/> suele haber retardos.

Extensión REST Client

- REST Client es una extensión para VSCode muy útil para probar APIs REST.
- Basta con instalar la extensión y crear un nuevo fichero con extensión “.rest” (ej: ClienteRest.rest) y podemos comenzar a escribir las consultas a realizar.



The screenshot shows the REST Client extension interface in VS Code. On the left, a file named 'ClienteRest.rest' is open, displaying a REST client configuration and two GET requests. The configuration section defines variables: '@hostname=http://localhost', '@port=3000', and '@host={{hostname}}:{{port}}'. Below the configuration, there are two requests. The first request is a GET request to '/posts' with a status of 'HTTP/1.1'. The second request is a GET request to '/posts/1' with a status of 'HTTP/1.1'. Both requests have a 'Send Request' button next to them. On the right, a 'Response(62ms)' window is open, showing the response for the second request. The response is an HTTP 200 OK status with various headers: 'X-Powered-By: Express', 'Vary: Origin, Accept-Encoding', 'Access-Control-Allow-Credentials: true', 'Cache-Control: no-cache', 'Pragma: no-cache', 'Expires: -1', 'X-Content-Type-Options: nosniff', 'Content-Type: application/json; charset=utf-8', 'ETag: W/"6bd7-4ZJC6aYj6sC9fV+iYRVwfvFHet4"', and 'Content-Encoding: gzip'.

```
ClienteRest.rest > GET /posts/1

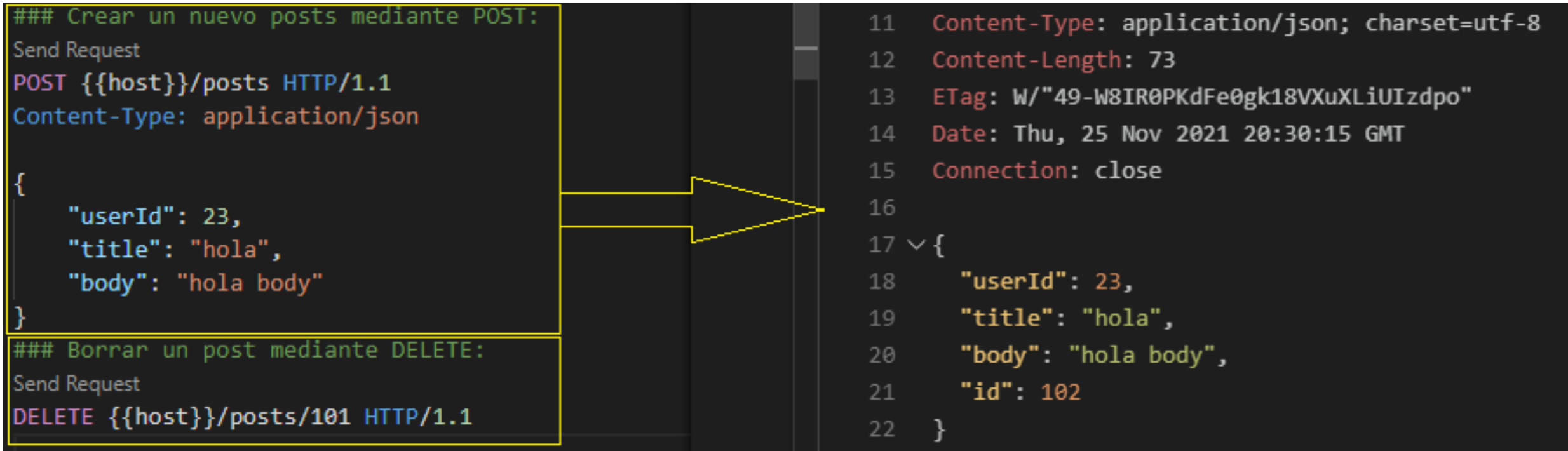
1 reference
1 @hostname=http://localhost
1 reference
2 @port=3000
4 references
3 @host={{hostname}}:{{port}}
4 ### Prueba obtener todos los posts mediante GET:
Send Request
5 GET {{host}}/posts HTTP/1.1
6 ### Prueba obtener un post mediante GET:
Send Request
7 GET {{host}}/posts/1 HTTP/1.1
```

```
Response(62ms) X
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Vary: Origin, Accept-Encoding
4 Access-Control-Allow-Credentials: true
5 Cache-Control: no-cache
6 Pragma: no-cache
7 Expires: -1
8 X-Content-Type-Options: nosniff
9 Content-Type: application/json; charset=utf-8
10 ETag: W/"6bd7-4ZJC6aYj6sC9fV+iYRVwfvFHet4"
11 Content-Encoding: gzip
```

- En la parte superior podemos definir variables (ej: @hostname).
- Debajo definimos las consultas. Deben ir precedidas de “###” y un comentario opcional.
- Cuando la sintaxis de la consulta es correcta aparece un vínculo “Send Request” para ejecutar.
- El resultado de la petición se visualizará en una nueva ventana en la que aparecen las cabeceras y los datos.

Extensión REST Client

- También es posible insertar nuevos elementos con POST, borrar con DELETE, etc:



The screenshot displays the VS Code REST Client interface. On the left, a POST request is defined with a placeholder host and a JSON body. On the right, the corresponding HTTP response is shown, including headers and the JSON body. A yellow arrow points from the request body to the response body, indicating the mapping of the request to the response.

```
### Crear un nuevo posts mediante POST:
Send Request
POST {{host}}/posts HTTP/1.1
Content-Type: application/json

{
  "userId": 23,
  "title": "hola",
  "body": "hola body"
}

### Borrar un post mediante DELETE:
Send Request
DELETE {{host}}/posts/101 HTTP/1.1
```

```
11 Content-Type: application/json; charset=utf-8
12 Content-Length: 73
13 ETag: W/"49-W8IR0PKdFe0gk18VXuXLiUIzdpo"
14 Date: Thu, 25 Nov 2021 20:30:15 GMT
15 Connection: close
16
17 {
18   "userId": 23,
19   "title": "hola",
20   "body": "hola body",
21   "id": 102
22 }
```

- Rest Cliente permite diferentes tipos de autenticación, cabeceras, variables, etc. Mas información en: <https://github.com/Huachao/vscode-restclient>
- Pulsando con el botón derecho sobre una consulta aparecen las opciones “Generate Code Snippet” y “Copy Request As cURL”.

Generate Code Snippet Ctrl+Alt+C
Copy Request As cURL

Consulta API REST desde AJAX. Ejemplo7.html

```
const xhr1 = new XMLHttpRequest();
xhr1.addEventListener('load', function () {
  if (xhr1.status === 200) {
    const result1 = JSON.parse(xhr1.response);
    console.log(result1);
  }
});
xhr1.open('GET', 'http://localhost:3000/users');
xhr1.send();

const xhr2 = new XMLHttpRequest();
xhr2.addEventListener('load', function () {
  if (xhr2.status === 200) {
    const result2 = xhr2.response;
    console.log(result2);
  }
});
xhr2.open('GET', 'http://localhost:3000/users');
xhr2.responseType = 'json';
xhr2.send();
```

- El ejemplo más sencillo: obtenemos todos los usuarios utilizando GET.
- En la versión xhr1 obtenemos los datos en texto plano y los parseamos con **JSON.parse()**.
- En la versión xhr2 establecemos la propiedad **responseType='json'** y con ello no es necesario parsear.
- El resultado es el mismo en ambos casos.

Consulta de varios elementos. Ejemplo8.html

```
<ul id="lista"></ul>
<script>
  const xhr = new XMLHttpRequest();
  xhr.addEventListener('load', function () {
    if (xhr.status === 200) {
      const lista = document.querySelector('#lista');
      xhr.response.forEach(t => {
        const li = document.createElement('li');
        li.innerHTML = `Usuario: ${t.userId} Título: ${t.title} `;
        if (t.completed) {
          li.innerHTML += '<b>Completado</b>';
        } else {
          li.innerHTML += '<b>No Completado</b>';
        }
        lista.appendChild(li);
      });
    }
  });
  xhr.open('GET', 'http://localhost:3000/todos');
  xhr.responseType = 'json';
  xhr.send();
</script>
```

- Se consulta la lista de los “todos” (trabajos por realizar) y se visualiza el resultado en una lista desordenada ul:

- Usuario: 1 Título: delectus aut autem **No Completado**
- Usuario: 1 Título: quis ut nam facilis et officia qui **No Completado**
- Usuario: 1 Título: fugiat veniam minus **No Completado**
- Usuario: 1 Título: et porro tempora **Completado**
- Usuario: 1 Título: laboriosam mollitia et enim quasi adipisci quia provident illum **No Completado**
- Usuario: 1 Título: qui ullam ratione quibusdam voluptatem quia omnis **No Completado**

Consulta de un elemento. Ejemplo9.html

```
<button id="btn-leer">Leer datos</button>
<form id="usuario">
  <input type="text" name="name" id="name">
  <input type="text" name="email" id="email">
  <input type="text" name="city" id="city">
</form>
<script>
  const xhr = new XMLHttpRequest();
  xhr.addEventListener('load', function () {
    console.log(xhr.status);
    if (xhr.status === 200) {
      const usuario = xhr.response;
      document.forms.usuario.name.value = usuario.name;
      document.forms.usuario.email.value = usuario.email;
      document.forms.usuario.city.value = usuario.address.city;
    }
  });
  document.querySelector('#btn-leer').addEventListener('click', () => {
    xhr.open('GET', 'http://localhost:3000/users/1');
    xhr.responseType = 'json';
    xhr.send();
  });
</script>
```

- Leer un usuario y almacenar los campos name, email y city en los controles del formulario.

Leer datos
Leanne Graham
Sincere@april.biz
Gwenborough

Ejercicio 1. Banderas.

- <https://restcountries.com/> es una API Rest completa que devuelve información sobre los países del mundo. Proporciona un JSON con diferentes datos (nombre, capital, moneda, población, URL con la bandera...) Ej:
 - <https://restcountries.com/v3.1/all> información de todos los países del mundo.
 - <https://restcountries.com/v3.1/name/spain> información sobre España.
- Crear una página web que visualice las banderas del mundo utilizando AJAX y la API Rest referenciada para producir un resultado como el siguiente:

