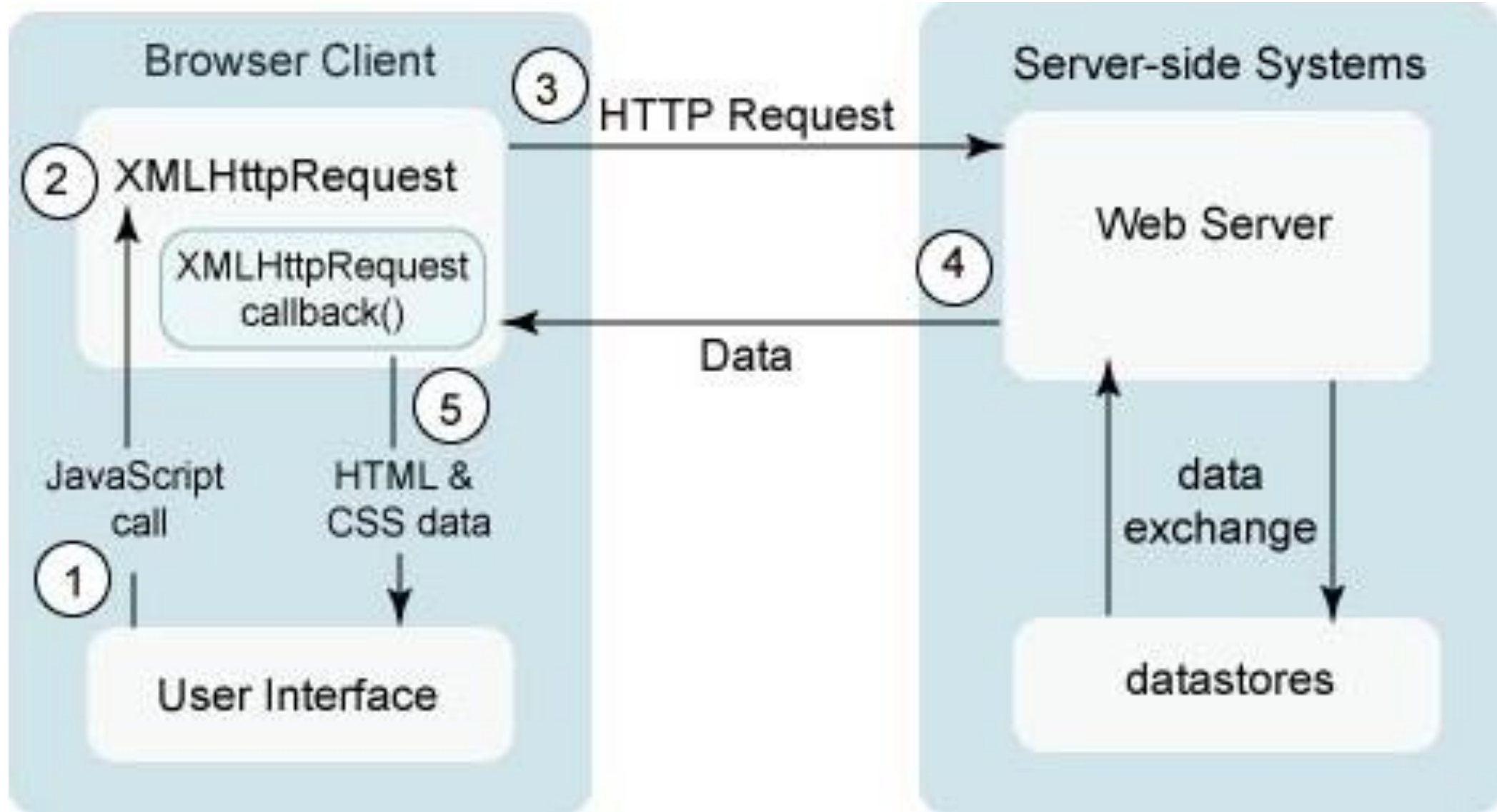


AJAX

- Ajax es acrónimo de **A**synchronous **J**avaScript **A**nd **X**ML
- No se trata de un lenguaje ni de un protocolo concreto sino más bien de una tecnología utilizada en el desarrollo web.
- El objeto central utilizado por AJAX es **XMLHttpRequest (XHR)**.
- La petición y respuesta al servidor se realiza mediante HTTP.
- Gracias a AJAX es posible enviar y recibir información al servidor sin recargar la página.
- El formato de los datos enviados y recibidos puede ser: XML, HTML, JSON, texto plano, etc...
- AJAX es por naturaleza asíncrono: el cliente no tiene porqué esperar de manera síncrona la respuesta del servidor.

AJAX

- Fases en la comunicación cliente-servidor con AJAX:



AJAX. Ejemplo1.html

- Ajax permite enviar una petición y obtener una respuesta sin recargar la página.
- El ejemplo más sencillo para cargar un fichero de texto plano:

```
<div class="container"></div>
<input type="button" id="btn" value="Leer">
<script>
  const xhr = new XMLHttpRequest();
  xhr.addEventListener('load', function () {
    if (xhr.status === 200) {
      document.querySelector('.container').innerHTML = xhr.response;
    }
  });
  document.querySelector('#btn').addEventListener('click', () => {
    xhr.open('get', 'http://localhost:5500/assets/datos.txt');
    xhr.send();
  });
</script>
```

En la propiedad "response" obtenemos la respuesta del servidor (puede ser un texto, un blob, un document, etc...).

Pasos a seguir:

- 1.- Crear el objeto **XMLHttpRequest**.
- 2.- Capturar el evento **load** que se produce cuando se devuelven los datos.
- 3.- Configurar la petición con el método **open()**.
- 4.- Enviar la petición con el método **send()**.

El Status HTTP 200 significa que todo es correcto.

AJAX. Ejemplo2.html

- También es posible leer mediante AJAX el contenido de un fichero y cargarlo y renderizarlo en nuestra propia página web. Ejemplo:

```
<div class="container"></div>
<input type="button" id="btn" value="Leer">
<script>
  const xhr = new XMLHttpRequest();
  xhr.addEventListener('load', function () {
    if (xhr.status === 200) {
      document.querySelector('.container').innerHTML = xhr.response;
    }
  });
  document.querySelector('#btn').addEventListener('click', () => {
    xhr.open('get', 'http://localhost:5500/assets/datos.html');
    xhr.send();
  });
</script>
```

Datos.html:

```
<ul>
  <li>Azul</li>
  <li>Rojo</li>
  <li>Negro</li>
</ul>
```

Fichero renderizado:

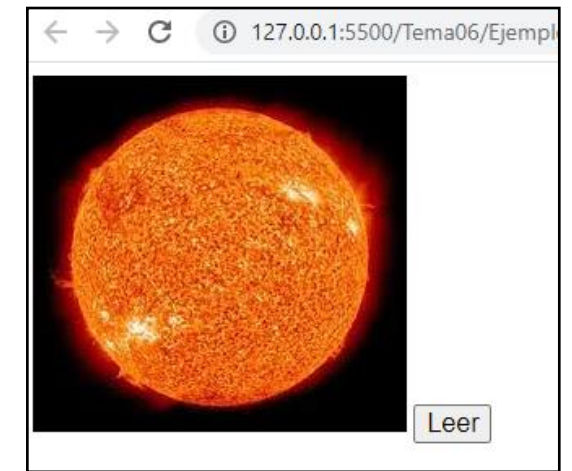
- Azul
- Rojo
- Negro

Leer

AJAX. Ejemplo3.html

- Descargar un fichero del servidor.

```
<img id="img">
<input type="button" id="btn" value="Leer">
<script>
  const xhr = new XMLHttpRequest();
  xhr.addEventListener('load', function () {
    if (xhr.status === 200) {
      const img=document.querySelector('#img');
      img.src=URL.createObjectURL(xhr.response);
    }
  });
  document.querySelector('#btn').addEventListener('click', () => {
    xhr.open('get', 'http://localhost:5500/assets/sol.jpg');
    xhr.responseType="blob";
    xhr.send();
  });
</script>
```



A partir de la respuesta se construye una imagen con el método "URL.createObjectURL()"

No se puede utilizar "responseText", usamos "response" porque los datos son binarios.

En este caso especificamos mediante la propiedad "responseType" el tipo de datos que deseamos como respuesta del servidor.

AJAX. Ejemplo4.html

- Leer datos JSON:

```
const xhr1 = new XMLHttpRequest();
xhr1.addEventListener('load', function () {
  if (xhr1.status === 200) {
    console.log(JSON.parse(xhr1.response));
  }
});
xhr1.open('get', 'http://localhost:5500/assets/datos.json');
xhr1.send();

const xhr2 = new XMLHttpRequest();
xhr2.addEventListener('load', function () {
  if (xhr1.status === 200) {
    console.log(xhr2.response);
  }
});
xhr2.open('get', 'http://localhost:5500/assets/datos.json');
xhr2.responseType = 'json';
xhr2.send();
```

```
[
  {
    "titulo": "Sol",
    "nombreImagen": "sol.jpg"
  },
  {
    "titulo": "Luna",
    "nombreImagen": "luna.jpg"
  }
]
```

Los dos ejemplos de código son iguales pero en el primer caso se obtienen los datos en modo texto y se parsean con `JSON.parse()`. En el segundo se indica mediante la propiedad `responseType` que los datos descargados son de tipo JSON.

```
▼ (2) [{...}, {...}] ⓘ
  ▶ 0: {titulo: 'Sol', nombreImagen: 'sol.jpg'}
  ▶ 1: {titulo: 'Luna', nombreImagen: 'luna.jpg'}
  length: 2
  ▶ [[Prototype]]: Array(0)

▼ (2) [{...}, {...}] ⓘ
  ▶ 0: {titulo: 'Sol', nombreImagen: 'sol.jpg'}
  ▶ 1: {titulo: 'Luna', nombreImagen: 'luna.jpg'}
  length: 2
  ▶ [[Prototype]]: Array(0)
```

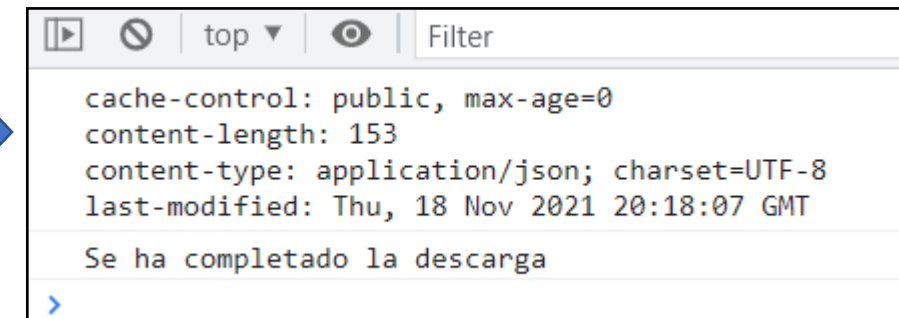
Eventos load y readystatechange. Ejemplo5.html

- El evento **load** se produce cuando la petición se ha completado correctamente.
- Si se desea realizar un control más fino (controlar estados intermedios o posibles errores) es posible utilizar el evento **readystatechange** y la propiedad **readyState**.

```
const xhr = new XMLHttpRequest();
xhr.addEventListener('readystatechange', function () {
  switch (xhr.readyState) {
    case XMLHttpRequest.HEADERS_RECEIVED: // 2
      console.log(this.getAllResponseHeaders());
      break;
    case XMLHttpRequest.DONE: // 4
      console.log('Se ha completado la descarga');
      break;
    default:
      break;
  }
});
xhr.open('get', 'http://localhost:5500/assets/datos.json');
xhr.send();
```

Posibles estados por los que pasa XMLHttpRequest:

- UNSENT = 0; // initial state
- OPENED = 1; // open called
- HEADERS_RECEIVED = 2; // response headers received
- LOADING = 3; // response is loading (a data packet is received)
- DONE = 4; // request complete



The screenshot shows a browser's developer console with the following content:

- cache-control: public, max-age=0
- content-length: 153
- content-type: application/json; charset=UTF-8
- last-modified: Thu, 18 Nov 2021 20:18:07 GMT
- Se ha completado la descarga

A blue arrow points from the code block on the left to this console output.

status y statusText. Ejemplo6.html

- **status** y **statusText** proporcionan información sobre el estado **HTTP**

```
const xhr1 = new XMLHttpRequest();
xhr1.addEventListener('readystatechange', function () {
  if (xhr1.readyState === 4 && xhr1.status === 200) {
    console.log(xhr1.statusText); // OK
  }
});
xhr1.open('get', 'http://localhost:5500/assets/datos.json');
xhr1.send();

const xhr2 = new XMLHttpRequest();
xhr2.addEventListener('load', function () {
  if (xhr2.status === 200) {
    console.log(xhr2.response);
  } else {
    console.log(xhr2.status, xhr2.statusText); // 404 'Not Found'
  }
});
xhr2.open('get', 'http://localhost:5500/assets/datos1.json');
xhr2.send();
```

Algunos de los códigos de respuesta HTTP son:

200 ok
201 created
204 no response
400 bad request
404 not found

Con xhr1 se utiliza readyState y status para determinar la lectura correcta (es el método antiguo).

Con xhr2 se utiliza el evento 'load' que se produce cuando se ha completado la lectura (método moderno). En este caso la lectura no es correcta (404) porque el fichero no existe.

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

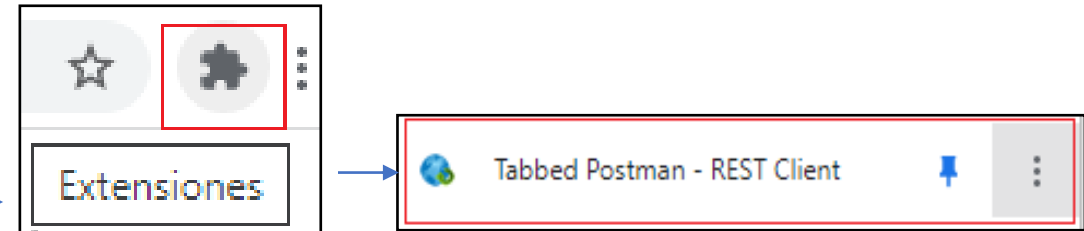
Datos de ejemplo API REST

- 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.

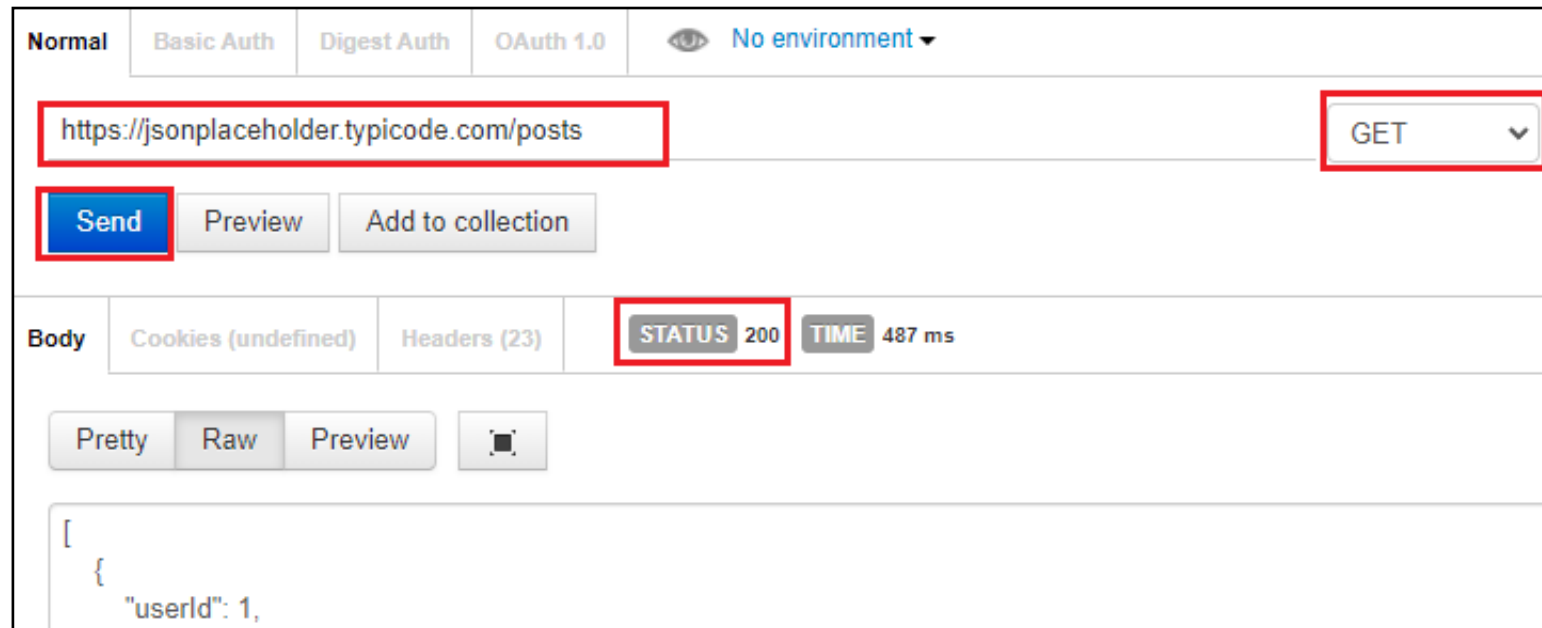
Cientes Rest. Tabbed Postman.

- 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:

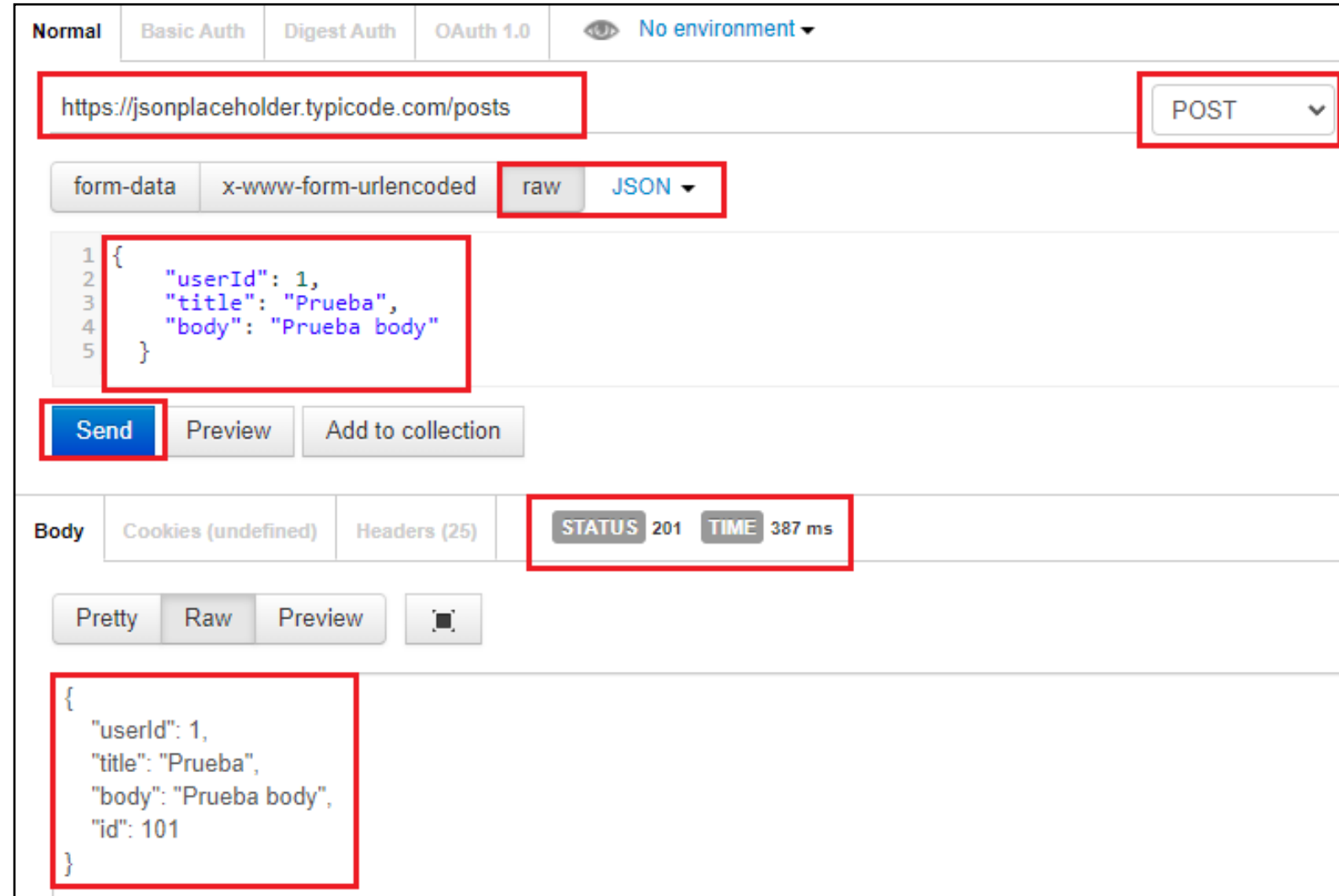


Cientes Rest. Tabbed Postman.

- 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

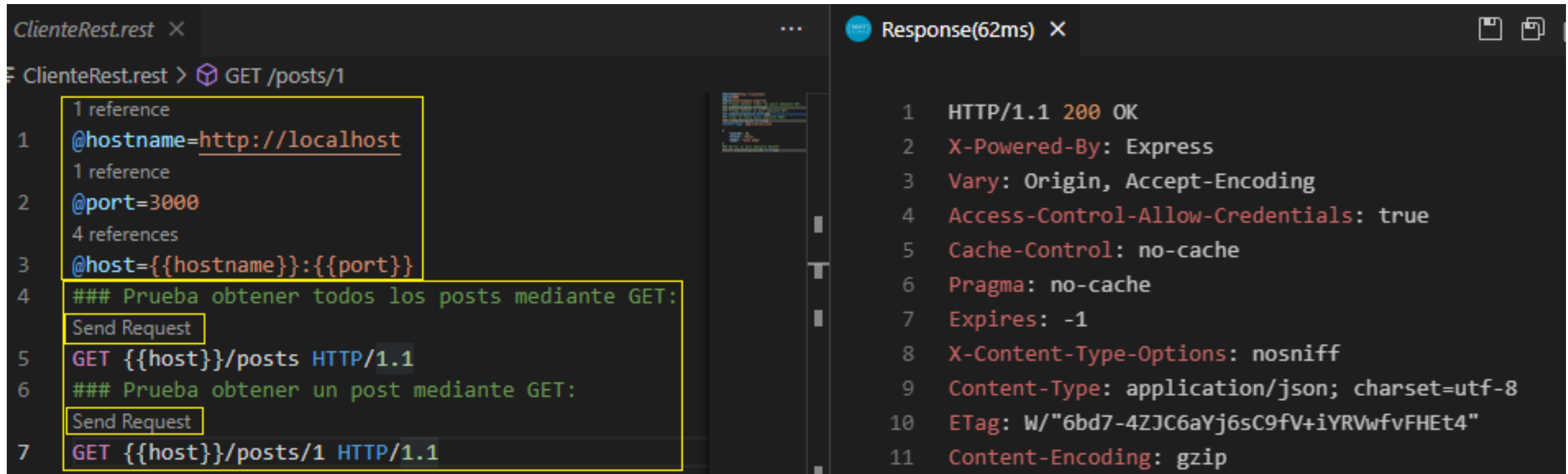
Type s + enter at any time to create a snapshot of the database
Watching...

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, showing 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 GET requests. The first request is 'GET {{host}}/posts HTTP/1.1' and the second is 'GET {{host}}/posts/1 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 first request. The response is a 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

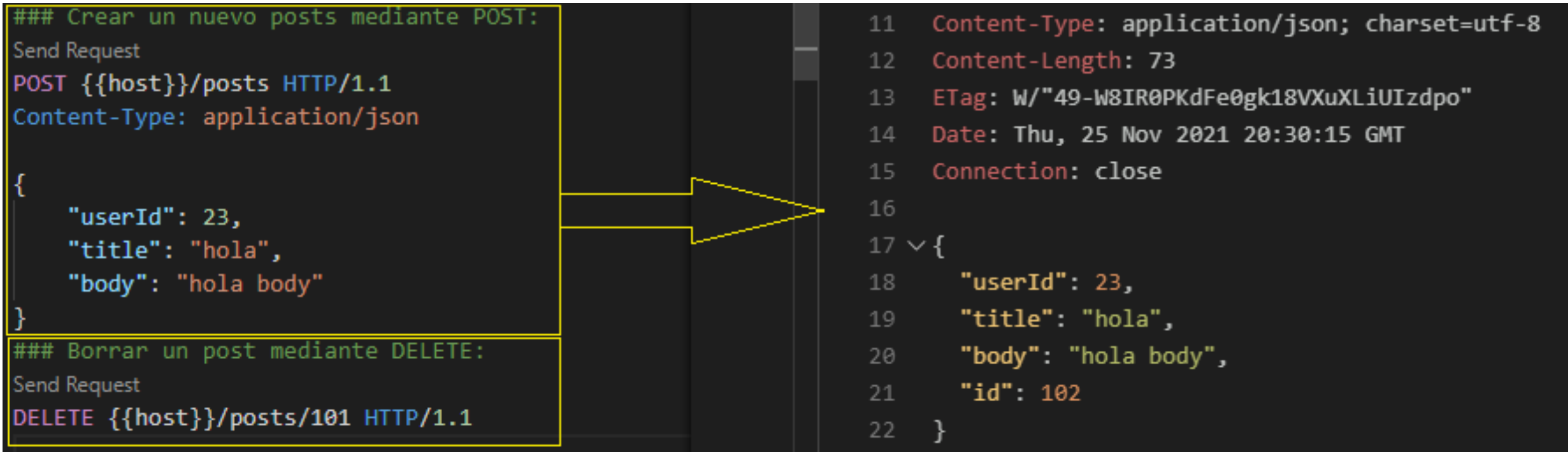
1 reference
1 @hostname=http://localhost
1 reference
2 @port=3000
4 references
3 @host={{hostname}}:{{port}}
4 ### Prueba obtener todos los posts mediante GET:
Send Request
5 GET {{host}}/posts HTTP/1.1
6 ### Prueba obtener un post mediante GET:
Send Request
7 GET {{host}}/posts/1 HTTP/1.1

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 a 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:

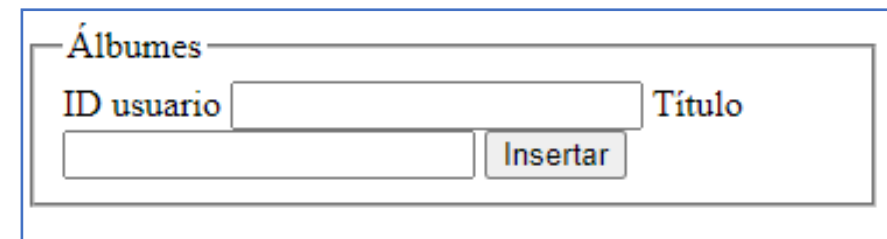


Añadir un elemento. Ejemplo10.html

- A partir de los datos de un formulario, crear un objeto, convertirlo a JSON con `JSON.stringify()` y enviarlo mediante POST:

```
const xhr = new XMLHttpRequest();
xhr.addEventListener('load', function () {
  if (xhr.status === 201) {
    alert('Album insertado');
  } else {
    alert(`Error: ${xhr.status} ${xhr.statusText}`);
  }
});
document.getElementById('album')
  .addEventListener('submit', function (e) {
    e.preventDefault();
    const album = {
      userID: document.forms.album.userID.value,
      title: document.forms.album.title.value
    };
    xhr.open('POST', 'http://localhost:3000/albums');
    xhr.setRequestHeader('Content-type', 'application/json');
    xhr.send(JSON.stringify(album));
  });
```

```
<form id="album">
  <fieldset>
    <legend>Álbumes</legend>
    <label for="userID">ID usuario</label>
    <input type="text" name="userID" id="userID">
    <label for="title">Título</label>
    <input type="text" name="title" id="title">
    <input type="submit" value="Insertar">
  </fieldset>
</form>
```



Álbumes

ID usuario

Título

Es obligatorio utilizar la cabecera "Content-type" como "application/json".

Borrar un elemento. Ejemplo11.html

- Para borrar basta con añadir el id del elemento a borrar a la url y realizar la solicitud con el método “DELETE”.

```
const xhr = new XMLHttpRequest();
xhr.addEventListener('load', function () {
  if (xhr.status === 200) {
    alert('Album borrado');
  } else {
    alert(`Error: ${xhr.status} ${xhr.statusText}`);
  }
});
document.getElementById('album')
  .addEventListener('submit', function (e) {
    e.preventDefault();
    const Id = document.forms.album.Id.value;
    const url = 'http://localhost:3000/albums/' + Id;
    xhr.open('DELETE', url);
    xhr.setRequestHeader('Content-type', 'application/json');
    xhr.send();
  });
```

```
<form id="album">
  <fieldset>
    <legend>Álbumes</legend>
    <label for="Id">ID del album a borrar:</label>
    <input type="text" name="Id" id="Id">
    <input type="submit" value="Borrar">
  </fieldset>
</form>
<script>
```

Álbumes	
ID del album a borrar:	<input type="text"/>
<input type="submit" value="Borrar"/>	

Modificar un elemento. Ejemplo12.html

- Para modificar es obligatorio especificar el id de la entidad en la URL:

```
const xhr = new XMLHttpRequest();
xhr.addEventListener('load', function () {
  if (xhr.status === 201) {
    alert('Usuario modificado');
  } else {
    alert(`Error: ${xhr.status} ${xhr.statusText}`);
  }
});

document.getElementById('user')
  .addEventListener('submit', function (e) {
    e.preventDefault();
    const user = {
      id: document.forms.user.id.value,
      name: document.forms.user.name.value,
      username: document.forms.user.username.value,
      email: document.forms.user.email.value
    };
    xhr.open('PATCH', 'http://localhost:3000/users/' + user.id);
    xhr.setRequestHeader('Content-type', 'application/json');
    xhr.send(JSON.stringify(user));
  });
```

```
<form id="user">
  <fieldset>
    <legend>Usuario a modificar</legend>
    <label>Id:<input type="text" name="id"></label><br>
    <label>Nombre:<input type="text" name="name"></label><br>
    <label>Nombre de usuario:<input type="text" name="username"></label><br>
    <label>Email:<input type="text" name="email"></label><br>
    <input type="submit" value="Modificar">
  </fieldset>
</form>
```

Usuario a modificar

Id:

Nombre:

Nombre de usuario:

Email:

Se puede utilizar **PUT** para modificar la entidad completa o **PATCH** para modificar solo alguno de los campos.

Ejercicio2. Maestro-Detalle simple.

- Crear una página web que permita visualizar la lista de usuarios.
- En la parte de la izquierda se visualiza una tabla con las columnas: Id, Nombre, Nombre de usuario y email.
- Cuando se haga “click” sobre una fila se visualizan los datos del detalle del usuario (Id, Nombre, Nombre de usuario, email y Ciudad) en el formulario que aparece a la derecha.

ID	Nombre	Nombre de usuario	Email	Detalle de usuario	
1	Leanne Graham	Bret	Sincere@april.biz	ID	<input type="text" value="Identificador"/>
2	Ervin Howell	Antonette	Shanna@melissa.tv	Nombre	<input type="text" value="Nombre"/>
3	Clementine Bauch	Samantha	Nathan@yesenia.net	Nombre de usuario	<input type="text" value="Nombre de usuario"/>
4	Patricia Lebsack	Karianne	Julianne.OConner@kory.org	Email	<input type="text" value="Email"/>
5	Chelsey Dietrich	Kamren	Lucio_Hettinger@annie.ca	Ciudad	<input type="text" value="Ciudad"/>
6	Mrs. Dennis Schulist	Leopoldo_Corkery	Karley_Dach@jasper.info		
7	Kurtis Weissnat	Elwyn.Skiles	Telly.Hoeger@billy.biz		
8	Nicholas	Maxime_Nienow	Sherwood@rosamond.me		

Simplificar el código

- Es posible utilizar una función que encapsule el uso del objeto XMLHttpRequest:

```
function ajax (options) {  
  const { url, method, success, error, data } = options;  
  const xhr = new XMLHttpRequest();  
  xhr.addEventListener('load', e => {  
    if (xhr.status >= 200 && xhr.status < 300) {  
      success(JSON.parse(xhr.response));  
    } else {  
      error(`Error: ${xhr.status} ${xhr.statusText}`);  
    }  
  });  
  xhr.open(method || 'GET', url);  
  xhr.setRequestHeader('Content-type', 'application/json; charset=utf-8');  
  xhr.send(JSON.stringify(data));  
}
```

Después se puede llamar a la función proporcionando solo los parámetros necesarios y simplificando el código.

```
ajax({  
  url: 'http://localhost:3000/posts',  
  success: posts => posts.forEach(p => console.log(p)),  
  error: () => console.log('Error en la lectura')  
});
```

```
ajax({  
  url: 'http://localhost:3000/posts',  
  method: 'post',  
  success: resultado => console.log('Post creado:', resultado),  
  error: () => console.log('Se ha producido un error escritura'),  
  data: { userID: 10, title: 'Prueba' }  
});
```

Ejercicio3. Mejorar Maestro-Detalle.

- Crear una página web igual a la del ejercicio 2 pero utilizando la función Ajax para simplificar el código.
- Añadir otras dos funciones: “leerUsuarios” y “leerDetalle”.

ID	Nombre	Nombre de usuario	Email	Detalle de usuario	
1	Leanne Graham	Bret	Sincere@april.biz	ID	<input type="text" value="Identificador"/>
2	Ervin Howell	Antonette	Shanna@melissa.tv	Nombre	<input type="text" value="Nombre"/>
3	Clementine Bauch	Samantha	Nathan@yesenia.net	Nombre de usuario	<input type="text" value="Nombre de usuario"/>
4	Patricia Lebsack	Karianne	Julianne.OConner@kory.org	Email	<input type="text" value="Email"/>
5	Chelsey Dietrich	Kamren	Lucio_Hettinger@annie.ca	Ciudad	<input type="text" value="Ciudad"/>
6	Mrs. Dennis Schulist	Leopoldo_Corkery	Karley_Dach@jasper.info		
7	Kurtis Weissnat	Elwyn.Skiles	Telly.Hoeger@billy.biz		
8	Nicholas	Maxime_Nienow	Sherwood@rosamond.me		