CREAR PACKAGE.JSON

pwd: Saber donde estoy ubicado.  
mkdir: Creación de carpeta  
cd: Moverse a carpetas  
git init: Inicias Git  
npm init: Le da nombre, versión, entre otras cosas al proyecto  
code . : Inicia el editor de códigos

RECURSO

**Platzi Fake Store API**

Fake store rest API for your e-commerce or shopping website prototype.

<https://fakeapi.platzi.com/>

exercise.js

1

2

3

4

export·function·execCallback(callback)·{

··window.setTimeout(callback,·2000);

}



Intentos

Correr Pruebas

GuíaVistaPruebas

**Should execute the callback**

**Should execute the callback after 2s**

¡Felicidades, todas las pruebas pasaron!

# XMLHTTPRequest

8/26

**𝗖𝗹𝗮𝘀𝗲 #𝟳: 𝗫𝗠𝗟𝗛𝗧𝗧𝗣𝗥𝗲𝗾𝘂𝗲𝘀𝘁𝟳/𝟮𝟭** 📤  
.  
📲 **XMLHttpRequest** es un objeto de JS que permite hacer peticiones hacia servicios en la nube(URLs o APIs).  
.  
📪 **Existen 5 estados en un llamado XMLHttpRequest:**  
.

* **0** → Se ha inicializado.
* **1** → Loading (cargando).
* **2** → Se ha cargado.
* **3** → Procesamiento si existe alguna descarga.
* **4** → Completado.  
  .

📫 **Métodos y propiedades:**  
.  
**xmlhttp.open()** → Prepara la petición para ser enviada tomando tres parámetros: prótocolo, url, asíncrono (true).  
**xmlhttp.readyState** → Retorna el estado de la petición.  
**xmlhttp.onreadystatechange** → Un eventHandler que es llamado cuando la propiedad readyState cambia.  
**xmlhttp.status** → Retorna el estado de la respuesta de la petición. (200,400,500)  
**xmlhttp.send()** → Envía la petición.  
.  
📬 **Características del protocolo http:**  
.  
**Verbos**: Los verbos indican acciones que están asociadas a peticiones y recursos, es decir, sirven para la manipulación de recursos cliente/servidor. Los Verbos http son:

* **GET** → Solicita un recurso.
* **HEAD** → Solicita un recurso pero sin retornar información, la estructura de esta petición es igual que get tanto en su headers como estatus. Es útil cuando vamos a utilizar API, para comprobar si lo que vamos a enviar esta correcto y puede ser procesado.
* **POST** → Sirve para la creación de recursos en el servidor.
* **PUT** → Actualiza por completo un recurso, reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
* **PATCH** → Actualiza parcialmente un recurso.
* **DELETE** → Elimina un recurso.  
  .

📭 **Los códigos de estados del servidor:**  
.  
El código de estado (status codes) sirve para describir el estado de la petición hecha al servidor.

* **1xx** → Indican que la petición fue recibida por el servidor, pero está siendo procesada por el servidor.
* **2xx** → Indican que la petición fue recibida, aceptada y procesada correctamente.
* **3xx** → Indican que hay que tomar acciones adicionales para completar la solicitud.
* **4xx** → Indican errores del lado del cliente que hizo mal una solicitud.
* **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 una API son:**  
.

* **200** → OK → Indica que todo está correcto.
* **201** → Created → Todo está correcto cuando se hizo una solicitud POST, el recurso se creó y se guardó correctamente.
* **204** → No Content → Indica que la solicitud se completó correctamente pero no devolvió información. Este es común cuando se hacen peticiones con el verbo DELETE.
* **400** → Bad Request → Indica que algo está mal en la petición (no encontró algo).
* **401** → Unauthorized → Significa que antes de hacer una solicitud al servidor nos debemos autenticar.
* **403** → Forbidden → Indica que no tenemos acceso a ese recurso aunque se esté autenticado.
* **404** → Not Found → Indica que no existe el recurso que se está intentando acceder.
* **500** → Internal Server Error → Indica que algo falló, es un error que retorna el servidor cuando la solicitud no pudo ser procesada.  
  .  
  Fuente: [aquí](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)  
  .

🖍️ **Ejemplo en VSC:**  
.

1. Ir a la consola y ubicarnos en la carpeta del proyecto y escribir el comando para instalar el paquete **XMLHttpRequest**:  
   **npm i xmlhttprequest**
2. Ir al VSC y crear un archivo llamado **challenge.js** en la ruta **src/callback**. El archivo queda:

**const** XMLHttppRequest = requiere('xmlhttprquest'); //llamado al XmlHttpRequest

**const** API = 'https://api.escuelajs.co/api/v1'; //API en mayúscula porque es una referencia que no va a cambiar

**function** **fetchData**(urlApi, callback){ //urlApi: no confundir y colocar API

**let** xhttp = **new** XMLHttppRequest(); //referencia a new XMLHttpRequest

xhttp.open('GET', urlApi, **true**); //petición "obtener" con true para habilitarlo

xhttp.onreadystatechange = **function**(event) { //escucha diferentes estados de la solicitud y conocer cuando está disponible la información

**if**(xhttp.readyState === 4) { //si el estado ha sido completada la llamada

**if**(xhttp.status === 200 ){ //el servido responde de forma correcta

callback(**null**, JSON.parse(xhttp.responseText)); //dentro de xhttp.responseTex recibimos lo que entrega el servidor en texto y se hace la transformación en JSON

}

} **else** {

**const** error = **new** Error('Error' + urlApi);

**return** callback(error,**null**); //es null porque no se está regresando ningún dato

}

}

xhttp.send();

}

La nueva forma de hacer peticiones a una API es el [fetch](https://developer.mozilla.org/es/docs/Web/API/Fetch_API" \t "_blank).