# JS en la Web: CRUD con JavaScript asíncrono

## Conclusión 1: Comunicación asíncrona

* Hacer requisiciones utilizando xmlhttprequest.
* Lidias con las promises.
* Hacer una requisición http utilizando xmlhttprequest para buscar todos los clientes del servidor.
* Refactorizar el código utilizando promises mejorando su lectura.
* Utilizar template literals para crear un template html.

## Conclusión 2: Fetch API

* Entender el orden de ejecución del código JavaScript.
* Jalar los datos del servidor utilizando la función fetch api al invés de xmlhttprequest para modernizar el código.
* Refactorizar las responsabilidades del código pensando en la manutención de la aplicación en el futuro.

## Conclusión 3: Crear y borrar clientes

* Usar el método closest para encontrar el elemento del DOM más próximo al cual queremos remover.
* Crear un cliente utilizando el verbo http POST.
* Remover un elemento del dom con el método remove().
* Borrar un cliente utilizando el verbo http DELETE.

## Conclusión 4: Editar datos

* Hacer una query string utilizando la propiedad serachParams.get() para encontrar un id.
* Crear una IIFE o función autoejecutable.
* Editar los datos utilizando el verbo http PUT.

## Conclusión 5: Async/Await

* Utilizar async/await para indicar funciones asíncronas.
* Tratar los errores con try/catch.

## Anexos:

* <https://www.npmjs.com/package/json-server>
* <https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfpapjjmafapmmgkkhgoa/related?hl=es---------->
* <http://localhost:3000/perfil>
* <https://www.cdmon.com/es/blog/la-ejecucion-de-scripts-esta-deshabilitada-en-este-sistema-te-contamos-como-actuar#:~:text=Para%20poder%20ejecutar%20los%20scripts,scripts%20maliciosos%20en%20el%20sistema>.
* <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Promise>
* <https://app.aluracursos.com/course/js-web-crud-javascript-asincrono/task/77095>
* <https://www.aluracursos.com/blog/empezando-con-fetch-en-javascript?utm_source=gnarus&utm_medium=timeline>
* <http://latentflip.com/loupe/?code=JC5vbignYnV0dG9uJywgJ2NsaWNrJywgZnVuY3Rpb24gb25DbGljaygpIHsKICAgIHNldFRpbWVvdXQoZnVuY3Rpb24gdGltZXIoKSB7CiAgICAgICAgY29uc29sZS5sb2coJ1lvdSBjbGlja2VkIHRoZSBidXR0b24hJyk7ICAgIAogICAgfSwgMjAwMCk7Cn0pOwoKY29uc29sZS5sb2coIkhpISIpOwoKc2V0VGltZW91dChmdW5jdGlvbiB0aW1lb3V0KCkgewogICAgY29uc29sZS5sb2coIkNsaWNrIHRoZSBidXR0b24hIik7Cn0sIDUwMDApOwoKY29uc29sZS5sb2coIldlbGNvbWUgdG8gbG91cGUuIik7!!!PGJ1dHRvbj5DbGljayBtZSE8L2J1dHRvbj4%3D>
* <https://cdnjs.com/libraries/uuid/8.3.2>
* <https://www.aluracursos.com/blog/diferencias-entre-get-y-post>
* <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/async_function>
* <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch>

## Código:

## Vocabulario:

* AJAX: JS asíncrono y xml}
* CRUD: Create Read Update Delete (métotodos http)
  + Create – POST
  + Read – GET
  + Update – PUT/PATCH
  + Delete – DELETE
* Backticks
* Callback hell
* Asincronia
* Entorno JS
* Pila de ejecucion: Ejecuta el codigo
* Pila de memoria: Almacena el nombre d elas variables, funciones…
* Motor JS: Toma el código js y lo transforma a codifo que pueda entender la maquina
* Web API: Se compone por una fila de funciones de JS
  + Fetch API
  + Almacenamiento
  + Audio
* Event Loop: Event loop lo que va a hacer es verificar qué parte del código pertenece a JavaScript nativo, y cuál pertenece a la fila de funciones de la web de API, como te comentaba. Pueden ser funciones fetch, parte del almacenamiento del audio, etcétera.
* Fila de promesas: También, si recuerdas, hemos estado trabajando con promesas, entonces también vamos a tener una fila específica para promesas. ¿Qué es lo que va a ser entonces Event loop? Lo que va a hacer es leer el código JavaScript de arriba hacia abajo. Lo que va a hacer es leer cierta porción de código, lo va a poner en la pila de ejecución, va a verificar si es código puramente JavaScript.

Si es así, lo va a ejecutar. En caso en que no sea JavaScript nativo y pertenezca a web API, lo va a tomar y lo va a agregar a nuestra fila de funciones de la web API. También lo mismo pasa con las promesas. En caso en que haga referencia a una promesa, lo que va a ser es mandarlo a la fila de promesas.

* Final del proceso: Una vez que Event loop termina de cargar todo lo que ven en la parte de pila la ejecución, lo que va a hacer es, ya verifiqué que no existe nada en esta pila de ejecución, voy entonces a empezar a tomar los archivos que vienen en las diferentes otras filas, como la fila de funciones, la de promesas y las va a ir poniendo dentro de nuestra pila de ejecución, y así sucesivamente hasta que nuestra pila de ejecución se quede vacía.

Esto indicaría que nuestro código se ha ejecutado completamente. Esto es más o menos una parte de la teoría. Vamos ahora a una página muy interesante, en la cual vamos a poder ver de una manera más visual todo esto que te acabo de comentar. Vamos entonces.

## Capturas

|  |
| --- |
|  |