



## PETICIÓN FTECH

**Duración estimada: 150 min.**

**Objetivos:**

- **Conectar a un servicio JSON**
- **Manejar objetos JSON**
- **Manejar FETCH**

**Introducción**

### 1. AJAX

"Ajax es un conjunto de métodos y técnicas que permiten intercambiar datos con un servidor y actualizar partes de páginas web sin necesidad de recargar la página completamente".

### 2. FTECH

La API Fetch proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, como peticiones y respuestas. También provee un método global `fetch()` que proporciona una forma fácil y lógica de obtener recursos de forma asíncrona por la red.

Este tipo de funcionalidad se conseguía previamente haciendo uso de `XMLHttpRequest`. Fetch proporciona una mejor alternativa que puede ser empleada fácilmente por otras tecnologías

Una petición básica de fetch es realmente simple de realizar. Eche un vistazo al siguiente código:

```
fetch('http://example.com/movies.json')  
  .then(function(response) {  
    return response.json();  
  })  
  .then(function(myJson) {  
    console.log(myJson);  
  });
```

Aquí estamos recuperando un archivo JSON a través de red e imprimiéndola en la consola. El uso de `fetch()` más simple toma un argumento (la ruta del recurso que quieres obtener) y devuelve un objeto Promise conteniendo la respuesta, un objeto Response.

FTECH→ Permite hacer una petición ajax que recibe una **url** que a su vez es una promesa.

**.then (data => data.json())** → Función de callback llamada data que recibe ese parámetro data y lo va a convertir a json.



Volvemos a ejecutar

```
.then (data => {
```

```
usuarios=data; // Vamos a recoger los datos y se almacenan en la variable usuario.
```

### **Comprobando que la petición es satisfactoria**

Una petición promise fetch() será rechazada con TypeError cuando se encuentre un error de red, aunque esto normalmente significa problemas de permisos o similares — por ejemplo, un 404 no constituye un error de red.

### **El método THEN**

El método then() retorna un Promise. Recibe dos argumentos: funciones callback para los casos de éxito y fallo de Promise.

## **3. JSONPlaceholder o Regres.in**

JSONPlaceholder es una API REST gratuita en línea que puede usar siempre que necesite algunos datos JSON. Servicio que recibe un JSON o una información y nos devuelve un JSON

```
fetch('https://jsonplaceholder.typicode.com/todos/1')  
  .then(response => response.json())  
  .then(json => console.log(json))
```

Existe otra API REST que es Regres.in

## **4. Función map:**

Cuando tenemos que trabajar con Arreglos en JavaScript, hay varias opciones disponibles con los cuales podemos iterar, transformar o manipular nuestros arreglos. En esta ocasión vamos a hablar de cómo funciona "Array.prototype.map()" este método que podemos usar en los arreglos en JavaScript.

El método **map()** nos permite devolver un nuevo arreglo de datos partiendo de un arreglo, dicho así, no mutamos los datos del arreglo original, ahora tenemos un nuevo arreglo con los valores resultantes.

**Ejemplo: Tenemos una lista de productos que compra un usuario y queremos obtener solamente los nombres de los productos a comprar.**

**map()** recibe como parámetro una función la cual recibe 3 parámetros, el elemento actual, índice del elemento actual y el arreglo original.

```
arreglo.map(function(elementoActual, indice, arregloOriginal) { ... código });
```

```
const products = [
```



```
{ id: "1", name: "shirt", category: "clothing" },
{ id: "2", name: "Sports Tennis", category: "accessories" },
{ id: "3", name: "Casual shoes", category: "footwear" },
{ id: "4", name: "skirt", category: "clothing" },
{ id: "5", name: "tie", category: "clothing" }
]

let nameOfProducts = products.map((product, index, array) => {
  // Cómo solo queremos los nombres, retornamos "name".
  return product.name;
})

console.log(nameOfProducts2); // ["shirt", "Sports Tennis", "Casual shoes", "skirt", "tie"]
```

## Secuencia/Desarrollo:

**1.Ejemplo: Copia el siguiente ejemplo y ejecútalo. A continuación modifica la página web con un buen diseño para mostrar los datos de los usuarios**

### fetch.js

```
'use strict'
var div_usuarios=document.querySelector('#usuarios');

// .Ofrece una API para acceder a recursos tipicos como usuarios, mensajes de un foro y fotos
var usuarios=[];
//fetch('https://jsonplaceholder.typicode.com/users')
fetch('https://reqres.in/api/users')
  .then (data=>data.json())
  .then (users=> {
    usuarios=users.data;

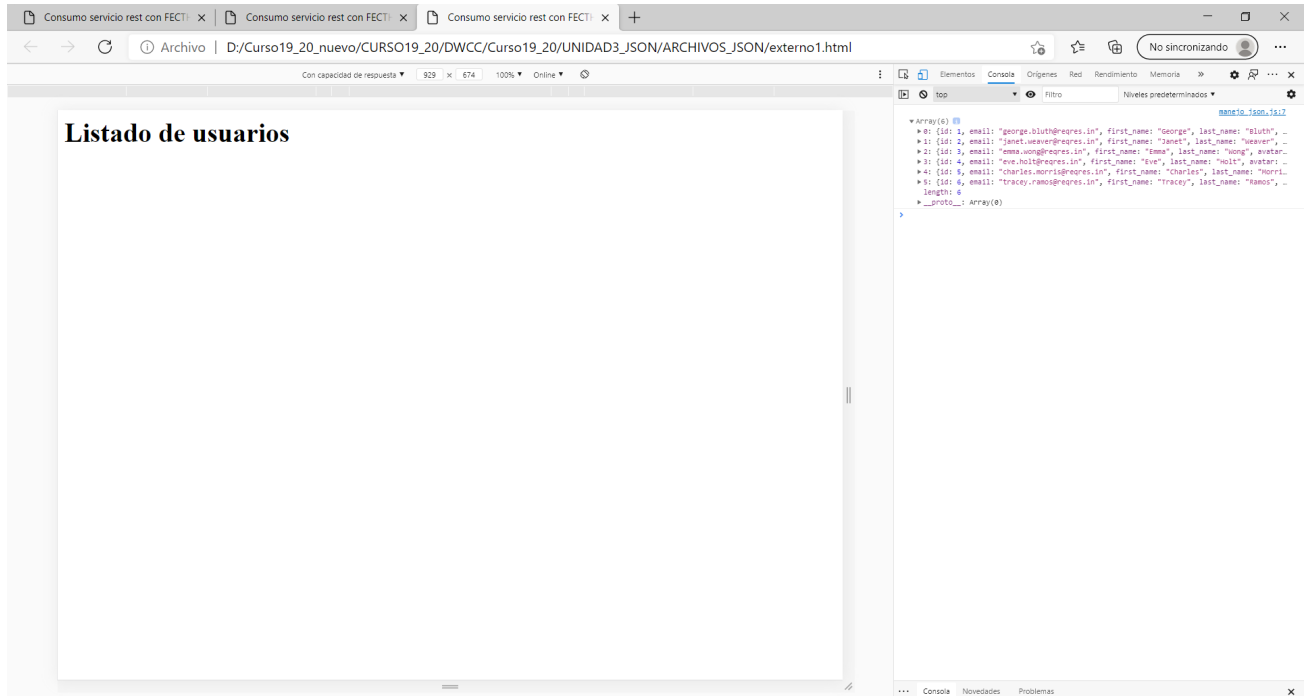
    console.log(usuarios);
    //Crea un nuevo array con los resultadosde la función
    usuarios.map((user,i)=> {
      let nombre=document.createElement("h2");
      nombre.innerHTML=i+ " " +user.first_name;
      div_usuarios.appendChild(nombre);
    });
  });

html lang="es">
<head>
  <title>Consumo servicio rest con FECTH</title>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

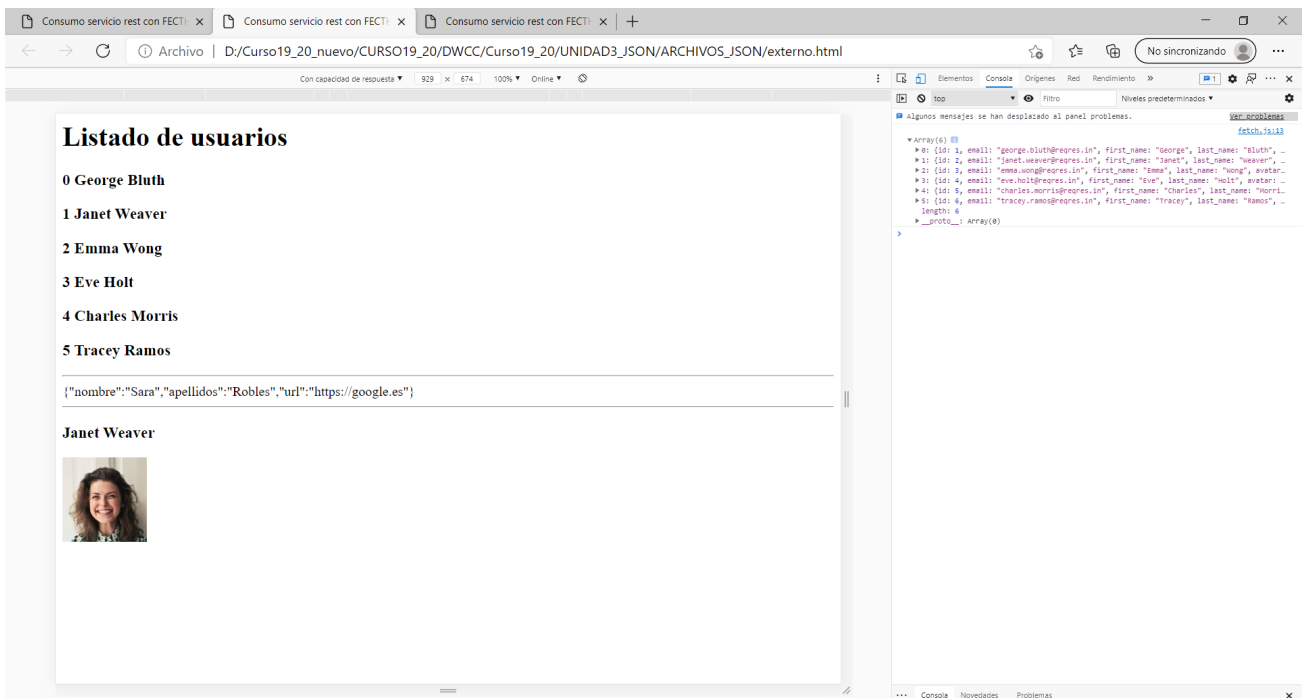
</head>
<body>
  <h1> Listado de usuarios </h2>

  <div id="usuarios">
    <h2> </h2>
```

```
</div>
<script type="text/javascript" src="js/fetch.js" > </script>
</body>
</html>
```



**Modificar el ejemplo anterior y mostrar los datos con el siguiente formato:**



**2.Ejercicio 2:** Realizar una web que nos permita realizar el mantenimiento básico de una tabla (Clientes), los cuales van a relacionarse con otra tabla (Provincias),



**Pasos:**

- Obtener los usuarios a través de la interfaz fetch. Estos datos se corresponden con los Clientes
- Crear un promise desde cero la cual debe tener los siguientes campos:
  - Provincia
  - Id (identificador del cliente)
- Debes diseñar un formulario que inserte los datos de las provincias, los cuales se almacenan en el LocalStorage.
- Añadir un botón mostrar, que muestra los datos del cliente y la provincia con la que está relacionada. Dichos datos se muestran con formato Tabla.
- Dar funcionalidad click a cada fila de la tabla que permita eliminar dichos datos de la interfaz y a su vez del localStorage.