



# DESARROLLO *WEB*

## *FULL STACK*

### *NIVEL BÁSICO*

# JSON y Fetch API

# Introducción a JSON y Fetch API

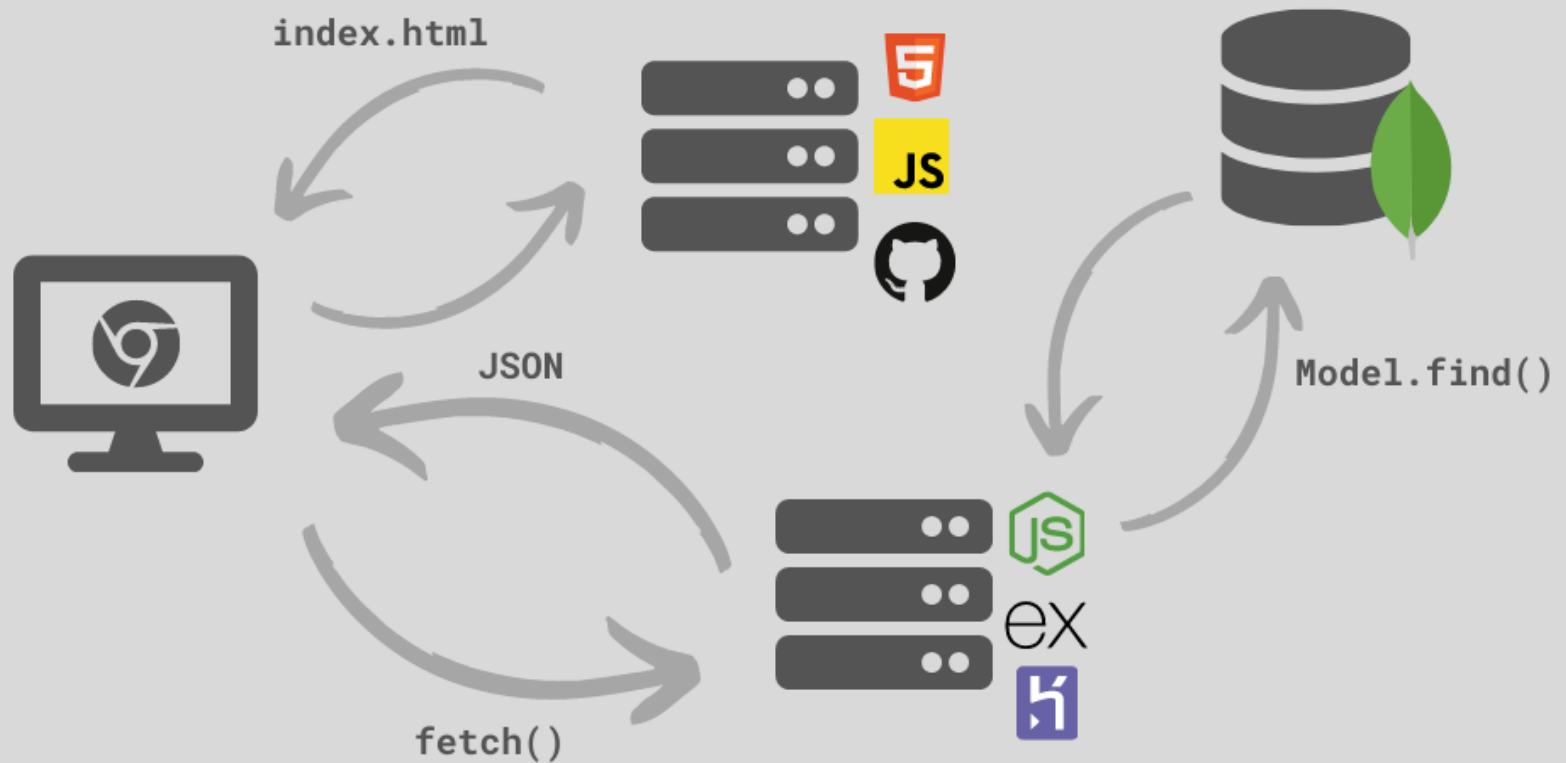
Cuando queremos enviar datos desde JavaScript en el lado del cliente a un backend o a otro lenguaje, nos enfrentamos a un problema:

los lenguajes de programación y los sistemas tienen formatos de datos diferentes.  
Por ejemplo:

- **JavaScript** utiliza objetos y arrays para representar datos estructurados.
- **PHP** utiliza arrays asociativos para representar datos estructurados.
- **Python** utiliza diccionarios para representar datos estructurados.
- **Java** utiliza objetos y colecciones para representar datos estructurados.

Para superar este problema, necesitamos un formato de datos que sea común y estandarizado, que pueda ser leído y escrito por diferentes lenguajes de programación y sistemas. ¡Ese formato es JSON!

# Introducción a JSON y Fetch API



# JSON (JavaScript Object Notation)

## Concepto de JSON

JSON es un formato de texto ligero utilizado para almacenar e intercambiar datos estructurados entre sistemas. Se basa en la sintaxis de objetos de JavaScript, pero es un formato independiente que puede ser utilizado en cualquier lenguaje de programación.

## Propósito de JSON

- ✓ Facilitar el intercambio de datos entre el cliente (frontend) y el servidor (backend).
- ✓ Proporcionar una forma sencilla y legible de representar información estructurada.
- ✓ Ser un estándar utilizado en APIs para enviar y recibir datos.

# JSON (JavaScript Object Notation)

## Sintaxis y estructura de JSON

La estructura de JSON se basa en pares **clave-valor** y sigue reglas muy estrictas para que sea válido.

Los valores pueden ser de varios tipos?

```
json

{
  "nombre": "Carlos",
  "edad": 30,
  "email": "carlos@email.com",
  "activo": true,
  "hobbies": ["leer", "viajar", "futbol"],
  "direccion": {
    "calle": "Av. Siempre Viva",
    "ciudad": "Bogotá",
    "pais": "Colombia"
  }
}
```

A diagram illustrating the structure of a JSON object. It shows a JSON object with several key-value pairs. Red arrows point from the values to the right, indicating that values can be of various types. The keys are in quotes, and the values are in various colors (green, blue, purple, red) to represent different data types: string, number, string, boolean, array, and object. The nested object 'direccion' is also shown with its own key-value pairs.

# Reglas de JSON

Para que un archivo JSON sea válido, debemos seguir estas reglas:

**1** Las claves deben ir entre comillas dobles ("")

✗ Incorrecto: `{ nombre: "Carlos" }`

✓ Correcto: `{ "nombre": "Carlos" }`

**3** No se permiten comentarios en JSON

✗ Incorrecto:

```
json

{
  "nombre": "Carlos" // Este es su nombre
}
```

**2** Los valores pueden ser de varios tipos:

- Strings (texto entre comillas): `"nombre": "Carlos"`
- Números: `"edad": 30`
- Booleanos: `"activo": true`
- Arrays (listas de valores): `"hobbies": ["leer", "viajar"]`
- Objetos anidados: `"direccion": { "calle": "Av. Siempre Viva" }`

# JSON (JavaScript Object Notation)

## 6 Tipos de datos en JSON

JSON admite los siguientes tipos de datos:

Tipo	Ejemplo
String	<code>"nombre": "Carlos"</code>
Number	<code>"edad": 30</code>
Boolean	<code>"activo": true</code>
Array	<code>"hobbies": ["leer", "viajar"]</code>
Object	<code>"direccion": { "calle": "Av. Siempre Viva" }</code>
Null	<code>"apellido": null</code>

# JSON (JavaScript Object Notation)

Javascript  
Array vs Object vs JSON

**[ ]**      **vs**      **{ }**      **vs**      **"{ }"**





# JSON (JavaScript Object Notation)

## 1. JSON.stringify()

Convierte un objeto JavaScript en una cadena de texto en formato JSON.

```
const estudiante = { nombre: "Juan", edad: 21, curso: "Programación" };  
const jsonString = JSON.stringify(estudiante);  
console.log(jsonString);  
// Salida: {"nombre":"Juan","edad":21,"curso":"Programación"}
```

## 2. JSON.parse()

Convierte una cadena JSON en un objeto JavaScript.

```
const jsonString = '{"nombre":"Juan","edad":21,"curso":"Programación"}';  
const estudiante = JSON.parse(jsonString);  
console.log(estudiante.nombre); // Salida: Juan
```

# JSON (JavaScript Object Notation)

## FormData

FormData permite recopilar datos de formularios en JavaScript.

```
<form id="miFormulario">
  <input type="text" name="nombre" placeholder="Nombre">
  <input type="number" name="edad" placeholder="Edad">
  <button type="submit">Enviar</button>
</form>
```

```
const formulario = document.querySelector("form");
formulario.addEventListener("submit", function(event) {
  event.preventDefault(); // Evita el envío tradicional

  const formData = new FormData(formulario);
  const datos = {};

  formData.forEach((valor, clave) => {
    datos[clave] = valor;
  });

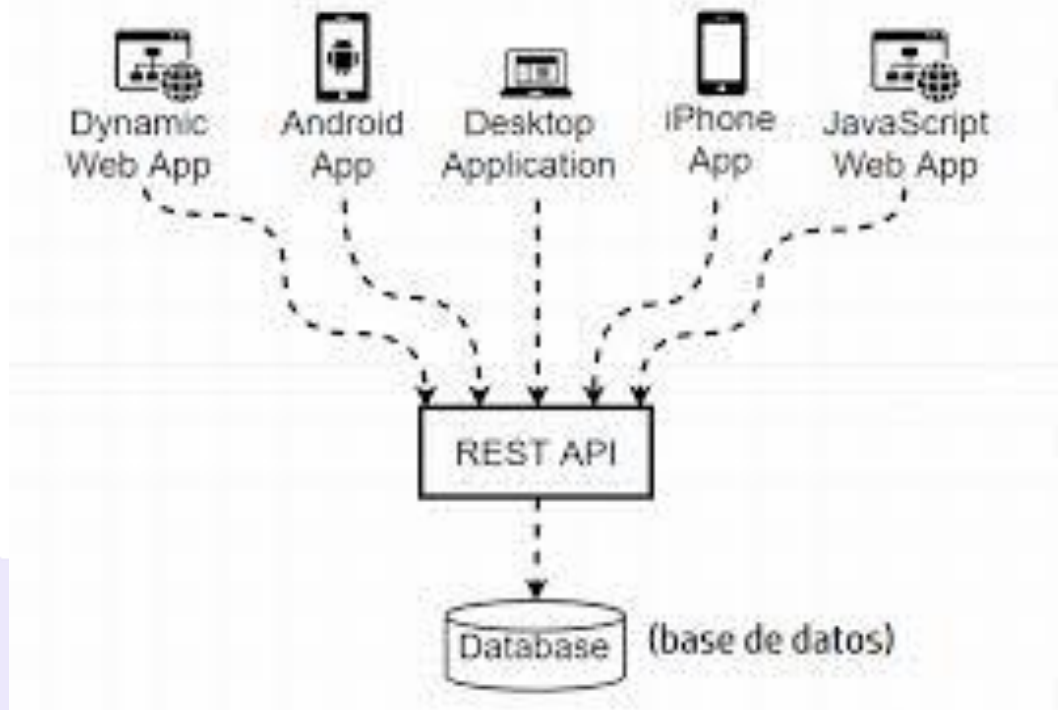
  console.log(JSON.stringify(datos)); // Convierte los datos a JSON
});
```

En JavaScript, **submit** es un evento que ocurre cuando un formulario es enviado. Se usa comúnmente con `addEventListener` para ejecutar código antes de que el formulario se envíe.

# Fetch API: Consumiendo datos con JavaScript

La Fetch API (API de recuperación) es una interfaz de programación de aplicaciones (API) que permite a los desarrolladores web realizar solicitudes HTTP desde JavaScript. Esta API es parte de la especificación de HTML5 y es soportada por la mayoría de los navegadores modernos.

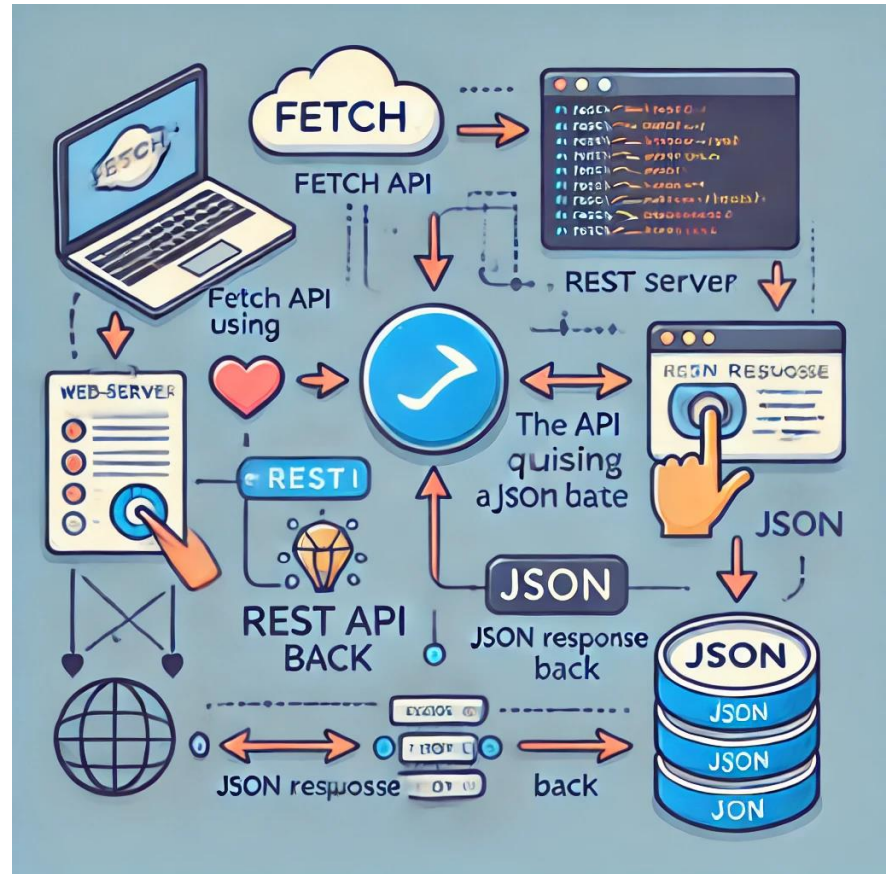
- "Fetch" en inglés significa "recuperar", "obtener" o "traer".
- "API" significa Interfaz de Programación de Aplicaciones.



# ¿Qué es HTTP?

HTTP (Hypertext Transfer Protocol) es un protocolo de comunicación que permite la transferencia de datos entre un cliente (como un navegador web) y un servidor web. Es el protocolo más comúnmente utilizado para acceder a la información en la web.

- 1 El usuario interactúa con el frontend.
- 2 El frontend envía una petición HTTP (`fetch()`).
- 3 El backend recibe la petición y la procesa.
- 4 El backend consulta la base de datos si es necesario.
- 5 El backend responde con datos en JSON.
- 6 El frontend recibe los datos y actualiza la interfaz.



# Fetch API: Consumiendo datos con JavaScript

## ◆ 3. Métodos HTTP en Fetch API

Método	Descripción
GET	Obtiene información del servidor (Ej: listar usuarios).
POST	Envía nuevos datos al servidor (Ej: registrar un usuario).
PUT	Actualiza un recurso existente en el servidor.
PATCH	Modifica parcialmente un recurso existente.
DELETE	Elimina un recurso del servidor.

## ◆ 4. Sintaxis básica de Fetch API

Fetch API sigue la estructura:

javascript

Copy Edit

```
fetch(url, opciones)
  .then(response => response.json()) // Convertir la respuesta en JSON
  .then(data => console.log(data)) // Manejar los datos
  .catch(error => console.error("Error:", error)); // Capturar errores
```

# Fetch API: Consumiendo datos con JavaScript

## GET: Obtener datos de una API

javascript



 Copy  Edit

```
fetch("https://jsonplaceholder.typicode.com/posts")
  .then(response => response.json())
  .then(data => console.log("Publicaciones:", data))
  .catch(error => console.error("Error al obtener datos:", error));
```

## POST: Enviar datos a una API

Para enviar datos al servidor, usamos `POST` y agregamos un cuerpo (body) con `JSON.stringify()`.

javascript

 Copy  Edit

```
const nuevoUsuario = {
  name: "Juan Pérez",
  email: "juan@email.com"
};

fetch("https://jsonplaceholder.typicode.com/users", {
  method: "POST",
  body: JSON.stringify(nuevoUsuario),
  headers: { "Content-Type": "application/json" }
})
  .then(response => response.json())
  .then(data => console.log("Usuario creado:", data))
  .catch(error => console.error("Error al enviar datos:", error));
```

# Fetch API: Consumiendo datos con JavaScript

## PUT: Actualizar un recurso existente

javascript

 Copy  Edit

```
const usuarioActualizado = {
  name: "Juan Pérez Actualizado",
  email: "juanactualizado@email.com"
};

fetch("https://jsonplaceholder.typicode.com/users/1", {
  method: "PUT",
  body: JSON.stringify(usuarioActualizado),
  headers: { "Content-Type": "application/json" }
})
.then(response => response.json())
.then(data => console.log("Usuario actualizado:", data))
.catch(error => console.error("Error al actualizar datos:", error));
```

# Fetch API: Consumiendo datos con JavaScript



## DELETE: Eliminar un recurso

javascript

Copy Edit

```
fetch("https://jsonplaceholder.typicode.com/users/1", {
  method: "DELETE"
})
.then(response => {
  if (response.ok) {
    console.log("Usuario eliminado con éxito");
  } else {
    console.log("Error al eliminar usuario");
  }
})
.catch(error => console.error("Error:", error));
```



# Fetch API: Consumiendo datos con JavaScript

## Autenticación con Fetch API (Headers y Tokens)

Muchas APIs requieren autenticación.

Para enviar un token de autorización, agregamos un header a la petición:

```
fetch("https://api.ejemplo.com/datos", {  
  method: "GET",  
  headers: {  
    "Authorization": "Bearer TU_TOKEN_AQUI"  
  }  
})  
  .then(response => response.json())  
  .then(data => console.log("Datos protegidos:", data))  
  .catch(error => console.error("Error:", error));
```

# Ejercicio Práctico: "Galería de Imágenes de Unsplash"

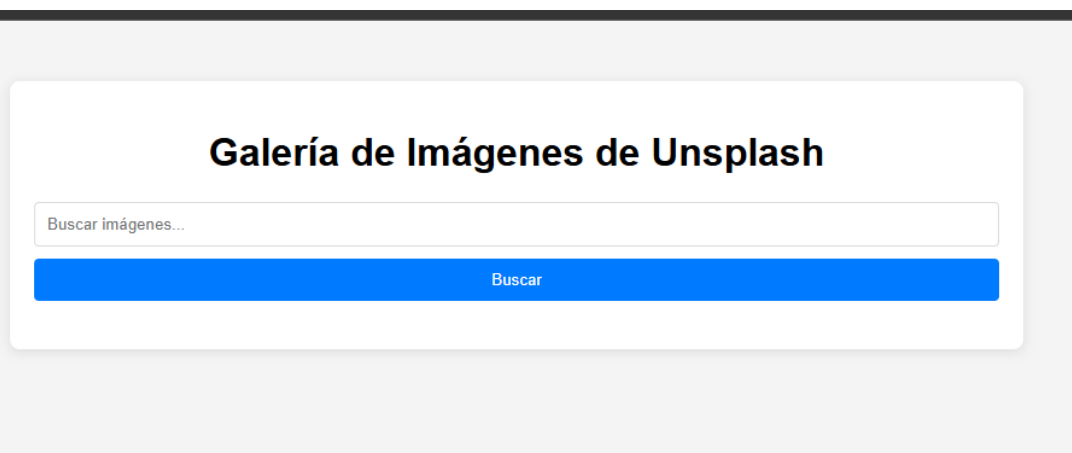
## Descripción del Ejercicio:

En este ejercicio, los estudiantes crearán una interfaz que mediante un buscador cree una galería de imágenes utilizando la API gratuita de Unsplash. Podrán realizar una búsqueda de imágenes y ver los resultados en tiempo real. La aplicación será dinámica e interactiva, mostrando las imágenes obtenidas de la API en un diseño limpio y atractivo.

## Objetivos:

- Aprender a realizar peticiones a una API utilizando JavaScript.
- Manipular el DOM para mostrar los datos obtenidos.
- Estilizar la aplicación utilizando CSS para que sea atractiva.

**Crear un interfaz donde tenga un input un botón y un div:**



The mockup shows a white rectangular interface on a light gray background. At the top, the title "Galería de Imágenes de Unsplash" is centered in a bold, black font. Below the title is a search bar with the placeholder text "Buscar imágenes...". Directly beneath the search bar is a solid blue button with the word "Buscar" centered in white text.

# Actividad # 2

## Objetivo:

Los estudiantes crearán una mini aplicación web que obtenga preguntas de trivia desde la API de **Open Trivia Database** y las muestre en la pantalla. Además, verificarán los datos en **Postman** para entender mejor las respuestas JSON.



## Pasos de la actividad:

### 1 Verificar la API con Postman

Antes de programar, los estudiantes deben probar la API en **Postman** para entender su estructura.

- URL de la API:

```
bash
```

[Copy](#) [Edit](#)

```
https://opentdb.com/api.php?amount=1&type=multiple
```

- Configurar una solicitud **GET** en Postman.
- Observar la estructura del JSON de respuesta.

# Actividad # 2

## 2 Crear el Proyecto Web

### Estructura de archivos:

```
bash

/trivia-app
|— index.html
|— style.css
|— script.js
```



## 6 Prueba en Postman

Para reforzar el concepto de APIs y JSON, los estudiantes pueden:

1. Hacer una solicitud GET en Postman a:

```
bash

https://opentdb.com/api.php?amount=1&type=multiple
```

 Copy  Edit

2. Observar la estructura JSON de la respuesta.
3. Compararla con los datos que se muestran en la app web.