

Diplomatura en



# FullStack Developer

---

JavaScript  
Consumo de APIs

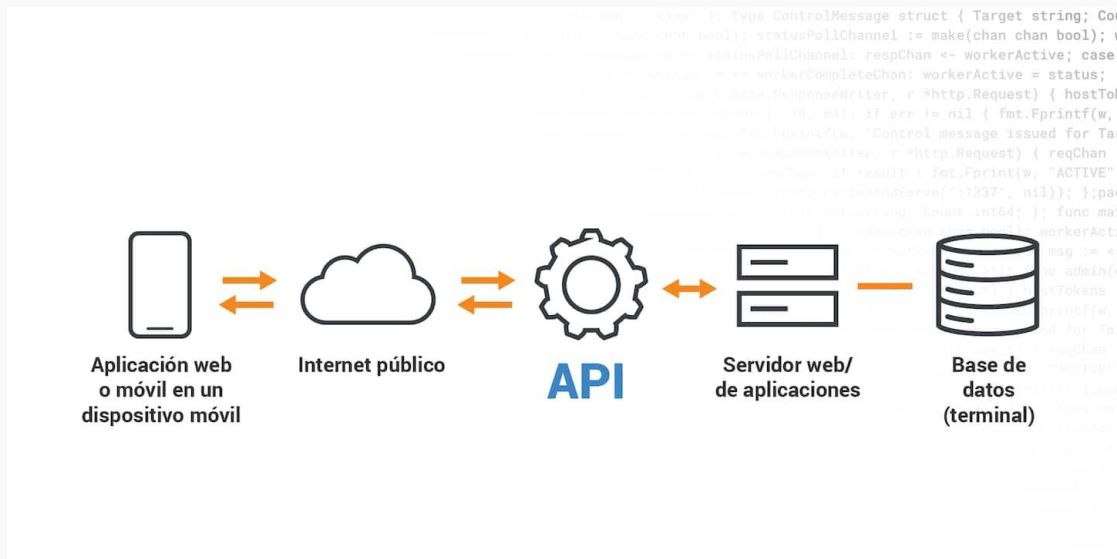
# ¿Qué es una API?

- API son las siglas de Application Programming Interface (Interfaz de Programación de Aplicaciones).
- Una API es un conjunto de reglas y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas. Sirve como puente para que diferentes programas interactúen de manera eficiente sin necesidad de conocer detalles internos del otro software.

## Utilidad de las APIs:

- **Comunicación entre programas:** Permiten que distintos softwares, a menudo desarrollados en diferentes lenguajes de programación o ejecutándose en diferentes plataformas, puedan interactuar.
- **Acceso a servicios:** Facilitan el acceso a funciones o datos que de otro modo no estarían disponibles directamente en un sistema. Por ejemplo, integrar el servicio de mapas de Google en una aplicación propia.

Las APIs son fundamentales en el desarrollo de aplicaciones modernas, permitiendo una mayor modularidad, escalabilidad y reutilización de funciones existentes.



# Métodos HTTP Básicos

## Fundamentos de las solicitudes API

- **GET:**

**Uso:** Solicitar datos desde un servidor. Es el método más común utilizado para obtener información.

**Ejemplo:** Obtener la lista de usuarios de una página, cargar comentarios de un blog.

- **POST:**

**Uso:** Enviar datos al servidor para crear un nuevo recurso.

**Ejemplo:** Crear un nuevo usuario, publicar un comentario en un foro.

- **PUT/PATCH:**

**PUT:** Actualizar completamente un recurso existente.

**PATCH:** Aplicar una actualización parcial a un recurso.

**Ejemplo:** Actualizar la información del perfil de un usuario (PUT podría cambiar toda la información, mientras que PATCH podría modificar solo el nombre).

- **DELETE:**

**Uso:** Eliminar un recurso del servidor.

**Ejemplo:** Borrar un usuario, eliminar un comentario.

**HTTP** (Protocolo de Transferencia de Hipertexto) es el protocolo subyacente que se utiliza en la Web para la transferencia de datos. Es un protocolo de solicitud-respuesta estándar en el modelo cliente-servidor, lo que significa que el cliente envía una solicitud y el servidor responde a esa solicitud.

- Es crucial entender qué método utilizar dependiendo de la acción que se desea realizar con la API.
- Cada método tiene implicaciones en términos de seguridad y rendimiento, que deben considerarse al diseñar una API.

# Herramientas para Consumo de APIs

## Fetch API

•**Descripción:** Una forma nativa de JavaScript para realizar solicitudes HTTP. Disponible en la mayoría de los navegadores modernos sin necesidad de bibliotecas adicionales.

•**Características:**

- Basada en Promesas, lo que facilita el manejo de operaciones asíncronas.
- No soporta cancelación de solicitudes directamente (hasta Fetch API nivel 2).
- No envía cookies ni autentica al usuario por defecto, a menos que se configure explícitamente.

## **Axios:**

•**Descripción:** Una biblioteca JavaScript que se puede utilizar tanto en el navegador como en Node.js. Es popular por su API fácil de usar y por manejar automáticamente la transformación de datos JSON.

### •**Características:**

- Soporta el monitoreo del progreso de las solicitudes.
- Permite cancelar solicitudes y configurar tiempos de espera.
- Maneja automáticamente la transformación de datos JSON.
- Más configurable en términos de interceptores, que permiten manejar o transformar solicitudes y respuestas.

## Comparación:

- Simplicidad:** Fetch es más "bajo nivel" y puede requerir más código para tareas comunes comparado con Axios.
- Soporte y funcionalidad:** Axios proporciona características adicionales que pueden ser cruciales para ciertas aplicaciones, como la cancelación de solicitudes.
- Comunidad y soporte:** Axios tiene una gran comunidad y una amplia documentación, lo que puede ser útil para resolver problemas específicos.



## Introducción a Fetch API:

- Breve repaso: Fetch API permite realizar solicitudes HTTP en JavaScript. Es promesa-basada, lo que facilita el manejo de operaciones asíncronas.

## Ejemplo de Código:

### Solicitud GET

- **Objetivo:** Obtener datos de un servidor externo.

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.error('There was a problem with your
fetch operation:', error));
```

```
async function fetchData() {  
  try {  
    const response = await fetch('https://api.example.com/data');  
    if (!response.ok) {  
      throw new Error('Network response was not ok');  
    }  
    const data = await response.json();  
    console.log(data);  
  } catch (error) {  
    console.error('There was a problem with your fetch operation:', error);  
  }  
}  
  
fetchData();
```

# Manejo de Respuestas y Errores

- **Interpretación de Respuestas:**

- **JSON:** La mayoría de las APIs modernas envían respuestas en formato JSON. Utilizar `response.json()` en Fetch o acceder a `response.data` en Axios para convertir esta respuesta en un objeto JavaScript usable.

- **Datos útiles:** Enseñar a identificar y extraer la información relevante de los objetos de respuesta, como identificadores de estado, mensajes y datos específicos del recurso.

- **Manejo de Errores Comunes:**

- **Errores de Conexión:** Problemas de red o servidor inaccesible. Ejemplo: manejo de excepciones para errores de conexión.

- **Errores HTTP (4xx y 5xx):**

- **4xx:** Errores del lado del cliente, como 404 (No encontrado) o 401 (No autorizado).

- **5xx:** Errores del lado del servidor, como 500 (Error interno del servidor) o 503 (Servicio no disponible).

- **Estrategias de Manejo:**

- Verificar códigos de estado HTTP.

- Utilizar bloques try/catch en funciones async.

- Capturar y manejar rechazos en promesas con `.catch()`.

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.error('There was a problem with your fetch operation:', error));
```

**Leer la documentación:** Antes de empezar a utilizar una API, dedica tiempo a leer su documentación oficial. Esto te ayudará a entender mejor sus capacidades y limitaciones.

# Consumo de API's

Algunas requieren claves API para acceder a ellas, mientras que otras no.

## Importancia de las Claves API

Las claves API son importantes por varias razones:

- **Control de Acceso:** Permiten a los proveedores de la API gestionar quién está utilizando sus servicios.
- **Limitación de Tasa:** Evitan que un usuario sobrecargue el servicio, estableciendo límites en cuántas solicitudes puede hacer un usuario o una aplicación en un período determinado.
- **Monitoreo y Análisis:** Ayudan a los proveedores a monitorear el uso de sus servicios y a analizar cómo y quién los utiliza.

## Api sin key, la famosa PokeAPI

<https://pokeapi.co/api/v2/pokemon?limit=151>

<https://pokeapi.co/docs/v2>

```
// Función para obtener los datos de los primeros 151 Pokémon
function fetchFirstGenPokemon() {
  // URL de la API para obtener los primeros 151 Pokémon
  const url = 'https://pokeapi.co/api/v2/pokemon?limit=151';
  // Hacer la solicitud GET a la API
  fetch(url)
    .then(response => {
      // Verificar si la solicitud fue exitosa
      if (response.ok) {
        return response.json(); // Convertir la respuesta a JSON
      }
      throw new Error('No se pudo obtener los datos de los Pokémon.');
```

```
// Función para obtener los datos de un Pokémon por su nombre
function fetchPokemonData(pokemonName) {
  // URL de la API para un Pokémon específico
  const url = `https://pokeapi.co/api/v2/pokemon/${pokemonName}`;

  // Hacer la solicitud GET a la API
  fetch(url)
    .then(response => {
      // Verificar si la solicitud fue exitosa
      if (response.ok) {
        return response.json(); // Convertir la respuesta a JSON
      }
      throw new Error('No se pudo obtener los datos del Pokémon.');
```

## Api con key

Estas por lo general necesitan que nos registremos para obtener una key y así consumir la API

<http://www.omdbapi.com/>

Completamos el  
formulario y  
presionamos submit

OMDb API

# API Key

**Email Delays!** If your requested key doesn't show up within an hour, please contact me directly.

### Generate API Key

Account Type ☐ Patreon  
☒ FREE! (1,000 daily limit)

Email

Name  First Name  Last Name

Use



A short description of the application or website that will use this API.



Nos aparecerá este mensaje debajo del formulario y debemos ir a la casilla del correo que registramos

**A verification link to activate your key was sent to: aca\_aparece\_tu\_email@mail.com**

The OMDb API <donotreply@omdbapi.com>

  Responder

Para: Usted

Here is your key: mi\_key\_123456

Please append it to all of your API requests,

OMDb API: <http://www.omdbapi.com/?i=tt3896198&apikey=123456>

Click the following URL to activate your key: [http://www.omdbapi.com/?i=tt3896198&ACTIVE\\_KEY=123456](http://www.omdbapi.com/?i=tt3896198&ACTIVE_KEY=123456)

If you did not make this request, please disregard this email.

Recibiremos este email y haremos click sobre el segundo link para activar nuestra key

Your key is now activated!

Solo veremos un HTML con este mensaje, y listo, ya podemos usar la key que recibimos en el correo para integrar a nuestra aplicación

By Search

Parameter	Required	Valid options	Default Value	Description
s	<input checked="" type="checkbox"/>		<empty>	Movie title to search for.
type	<input type="checkbox"/>	movie, series, episode	<empty>	Type of result to return.
y	<input type="checkbox"/>		<empty>	Year of release.
r	<input type="checkbox"/>	json, xml	json	The data type to return.
page <small>New!</small>	<input type="checkbox"/>	1-100	1	Page number to return.
callback	<input type="checkbox"/>		<empty>	JSONP callback name.
v	<input type="checkbox"/>		1	API version (reserved for future use).

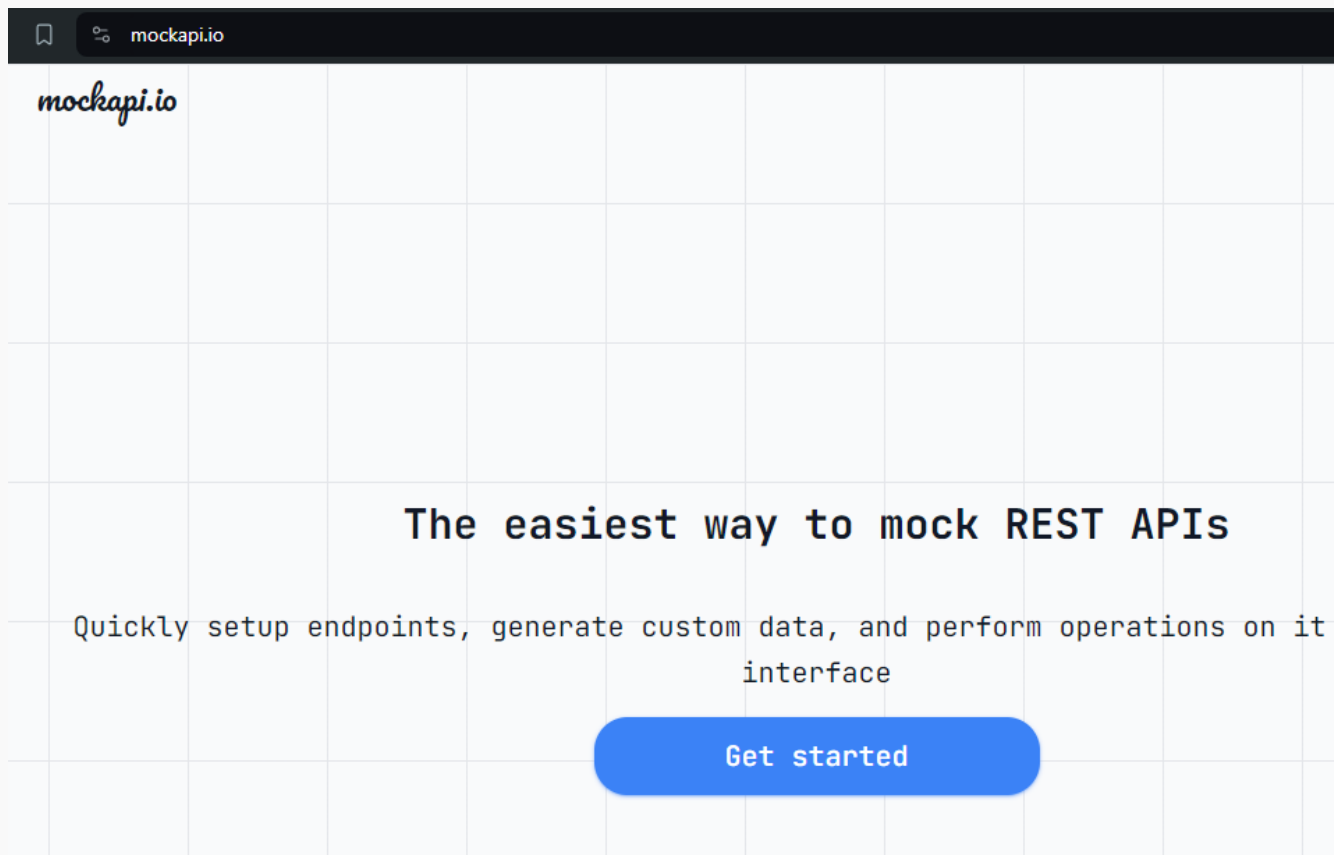
Para la búsqueda de películas usaremos estos parámetros que encontramos en la documentación de la api

Como es gratis, tenemos un uso limitado por día así como una respuesta acotada

Con este código podemos buscar películas por su nombre:

```
async function searchMovies(nameMovie) {  
  try {  
    const response = await fetch(`https://www.omdbapi.com/?apikey=${key}&type=movie&s=${nameMovie}`)  
    console.log(response)  
    if(response.ok){  
      const movies = await response.json();  
      return movies  
    }  
  } catch (error) {  
    throw new Error(error.message)  
  }  
}  
  
searchMovies('Men')  
  .then(data => console.log(data))  
  .catch(error => console.error(error.message));
```

Y como tercer opción vamos a crear una api para consumir



Presionamos en Ger started

Podemos iniciar sesión con github o google

← Back      Sign up

Sign in with Github

Sign in with Google

Name

...

Email

...

Password

...

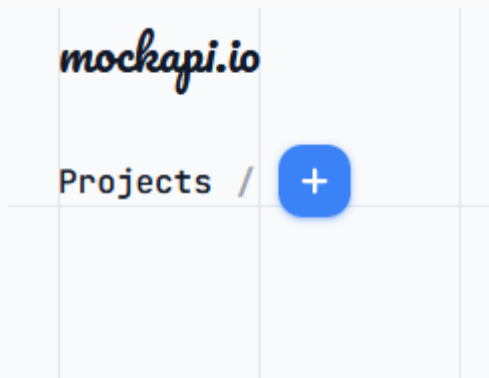
Confirm Password

...

Sign up

Already have an account? Login [here](#)

Presionamos sobre el botón “+”



Ponemos un nombre y si lo deseamos un prefijo, luego presionamos **create**

A screenshot of a 'NEW PROJECT' modal dialog box. The modal has a title bar with 'NEW PROJECT' on the left and a close 'X' button on the right. It contains two input fields. The first field is labeled 'Name' and has a light gray placeholder text 'Example: Todo App, Project X...'. The second field is labeled 'API Prefix' and has a light gray placeholder text 'Example: /api/v1'. Below the input fields, there are two buttons: a 'Cancel' button and a blue 'Create' button. The modal is set against a dark gray background with a grid pattern.

Una vez que eligieron el nombre del proyecto hacemos click sobre el

Tendrán una url similar a esta que usaremos.

Hacemos click en New resource para darle forma a nuestra api

Projects /



Curso JS

API endpoint

<https://00011111.mockapi.io/:endpoint>

Project secret

New resource

Vamos a darle el nombre **users** y dejaremos esos campos predefinidos para trabajar, presionamos en **create** y ya estará lista nuestra api.

NEW RESOURCE ×

**Resource name**  
Enter meaningful resource name, it will be used to generate API endpoints.

Example: users, comments, articles...

**Schema**  
Define Resource schema, it will be used to generate mock data.

id	Object ID	
createdAt	Faker.js	date.recent
name	Faker.js	name.fullName
avatar	Faker.js	image.avatar

+

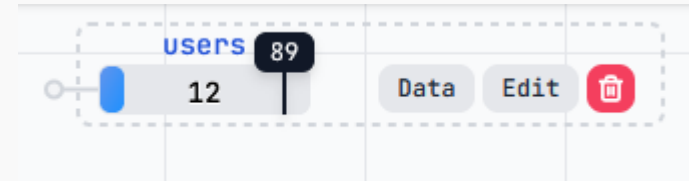
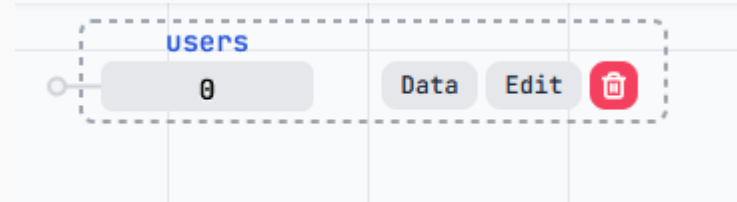
```
{
  "username": "$internet.userName",
  "knownIps": ["$internet.ip", "$internet.ipv6"],
  "profile": {
    "firstName": "$name.firstName",
    "lastName": "$name.lastName",
    "staticData": [100, 200, 300]
  },
}
```

EXAMPLE

Close Create



Tendremos este recuadro donde nos dirá cuantos registros existen, si nos paramos sobre la barra gris, podemos arrastrar el cursor y esto nos generará datos de ejemplo



Estos serán los endpoints que vamos a usar para poder hacer una aplicación CRUD completa.

Create  
Read  
Update  
Delete

On GET /users

\$mockData

On GET /users/:id

\$mockData

On POST /users

\$mockData

On PUT /users/:id

\$mockData

On DELETE /users/:id

\$mockData

# GET

Con esta función traemos todos los usuarios.

```
async function getUsers() {  
  try {  
    const response = await fetch(`https://1110000.mockapi.io/users`)  
    console.log(response)  
    if(response.ok){  
      const users = await response.json();  
      return users  
    }  
  } catch (error) {  
    throw new Error(error.message)  
  }  
}  
  
getUsers()  
  .then(data => console.log(data))  
  .catch(error => console.error(error.message));
```

## GET

Traemos a un usuario por su ID

```
async function getUserById(id) {  
  try {  
    const response = await fetch(`https://1110000.mockapi.io/users/${id}`)  
    console.log(response)  
    if(response.ok){  
      const user = await response.json();  
      return user  
    }  
  } catch (error) {  
    throw new Error(error.message)  
  }  
}
```

## UPDATE:

Pasamos un ID y en data que es un objeto el resto de información que necesitamos para modificar un usuario. Así como también, ahora a fetch pasaremos un argumento más que es un objeto de configuración, donde declaramos el método PUT, en el headers decimos que el contenido es json y en el body convertimos el objeto en formato JSON

```
async function updateById(id, data) {
  try {
    const response = await
    fetch(`https://1110000.mockapi.io/users/${id}`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data)
    })
    console.log(response)
    if(response.ok){
      const user = await response.json();
      return user
    }
  } catch (error) {
    throw new Error(error.message)
  }
}
```

## DELETE:

Ahora solo debemos trabajar con el ID para eliminar a un usuario y además en el objeto de configuración declarar que el método es **DELETE**

```
async function deleteById(id) {  
  try {  
    const response = await fetch(`https://1110000.mockapi.io/users/${id}`, {  
      method: 'DELETE'  
    })  
    console.log(response)  
    if(response.ok){  
      return `User ${id} deleted successfully`  
    }  
  } catch (error) {  
    throw new Error(error.message)  
  }  
}
```

# EJERCICIOS

## CONSUMO DE API'S:

Consumir una de las apis propuestas y mostrar esta información de manera ordenada sobre un documento HTML.

Crear un proyecto en [mockapi.io](https://mockapi.io), queda a opción del lector si desea consumir sus propios datos de esta api.