



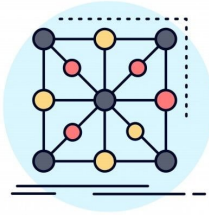
APIs y Promesas

Application Programming Interface

Su principal propósito es abstraer la implementación y solo exponer la información que los desarrolladores necesitan, de esta forma es sencillo reutilizar en distintos proyectos



Principales Usos



Librerías y Frameworks

Compartir código entre
diferentes softwares



Sistemas Operativos

Comunicación entre
software y SO

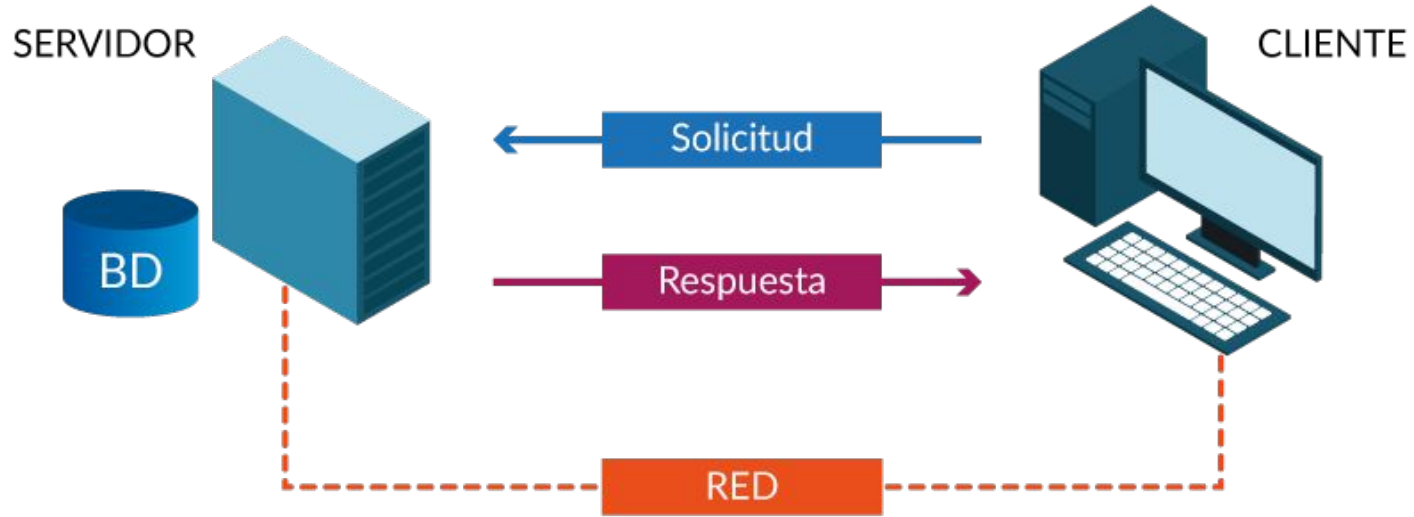


Web APIs

Comunicación entre
webs y servidores

1.

CLIENTE \Leftrightarrow SERVIDOR



Petición de servicios de un cliente a un servidor para transferir de datos

Cliente

Cualquier entidad que solicite un servicio, app, web o servidor

Servidor

Responde a la petición a través de internet o intranet

Protocolos

Archivos

File Transfer Protocol Secure (FTP - FTPS - SFTP)

Web

Hypertext Transfer Protocol Secure (HTTP - HTTPS)

Correo Electrónico

Post Office Protocol (POP3) - Simple Mail Transfer Protocol (SMTP)
Internet Message Transfer Protocol (IMAP)

Protocolo HTTP

Es un protocolo de comunicación para la transferencia de datos a través de la red. Define una semántica y sintaxis que deben utilizar el cliente y el servidor para comunicarse.

Cabeceras - Headers

Metadatos que proporcionan información extra sobre la *request*

Cuerpo - Body

Datos que envía el cliente al servidor

Método

Como se va a pedir la información

URL

Dirección dónde se encuentra el recurso

Métodos

GET - Obtener datos del recurso

POST - Envío de datos a un recurso, crear una entidad

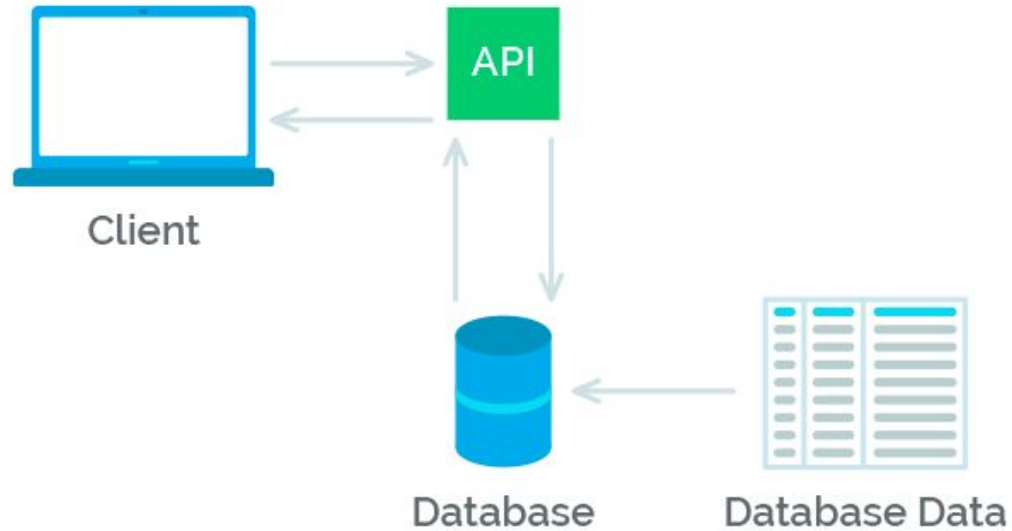
PUT - Actualiza totalmente una entidad, debe enviar un identificador

PATCH - Actualiza parcialmente, mejor performance

DELETE - Elimina un recurso identificado en la *request*

OPTIONS - Lanzado automáticamente por el navegador antes de ejecutar un método específico, da información de que opciones o requisitos están disponibles para el recurso dado

Flujo de Comunicación



HTTP Status Codes

1XX

Informational
Responses

2XX

Successful
Responses

3XX

Redirects

4XX

Client Errors



5XX

Server Errors

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

Representational State Transfer

Estándar de comunicación cliente - servidor. El servidor brinda acceso a recursos a través de URIs que el cliente utiliza mediante APIs.

Reglas de URIs		
Utilizar plurales en recursos	/movies/1/	/movie/1/
No implicar una acción	/movies/2/	/movies/2/edit/
No identificar formato	/movies/3/	/movies/3.pdf
Mantener lógica de jerarquía	/movies/4/actors	/actors/12/movies/
Utilizar query params	/movies/?year=1980	/movies/year/1980

Respuesta

El formato de respuesta va a variar dependiendo de la API.

XML - Extensible Markup Language

Similar a HTML pero con etiquetas definidas por el usuario.

Fue diseñado para almacenar y transportar información, que sea legible para humano y máquinas.

JSON - JavaScript Object Notation

Parece un objeto de JS, se creó con la misma intención que XML pero que sea más liviano. A pesar de llevar el nombre JavaScript es independiente del código, muchos lenguajes lo interpretan.

2. AJAX

Asynchronous JavaScript and XML



Sincronía vs Asincronía

Hasta ahora nuestro código siempre se ejecutaba línea tras línea.

```
setTimeout(ocultarMensaje, 2000);  
setInterval(aumentarTiempo, 1000);
```

Callback

Función que se pasa como parámetro a otra función y será ejecutada en algún momento. Nos ayuda a anidar funciones agregando comportamientos, pero tiene el problema del **callback hell**.



Una Promesa es un objeto que representa el eventual éxito o fallo de una operación asíncrona y su resultado.

Estados de una Promesa

Pending

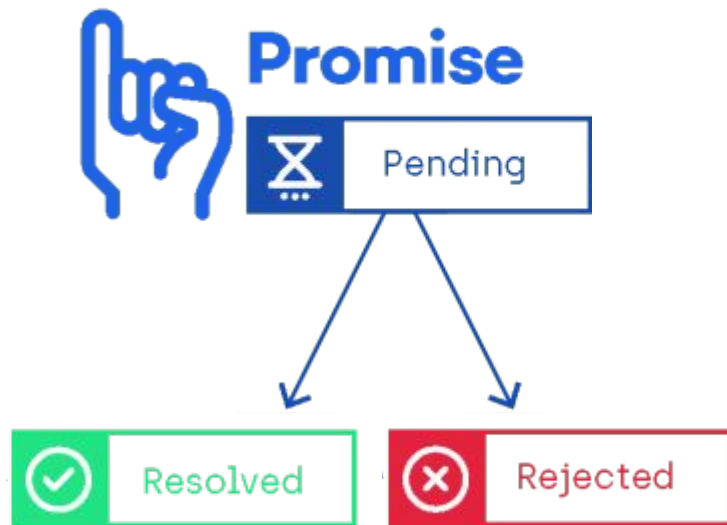
Estado inicial, todavía no sabemos si será resuelta o rechazada

Fulfilled

La operación terminó con éxito

Rejected

Falló la operación, hubo algún error



Crear una Promesa

```
let miPromesa = new Promise(  
  function (resolve, reject) {  
    ...  
    Acciones asíncronas  
    ...  
  
    resolve(info) // resuelva la promesa  
    reject(info)  // rechaza la promesa  
  }  
);
```

Encadenar Promesas

.then()

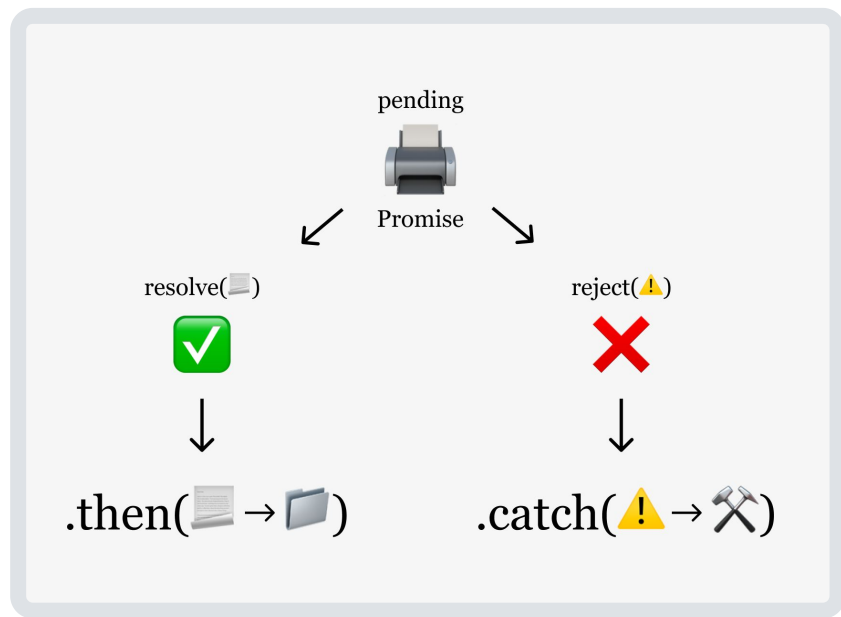
Trabaja con el resultado de la promesa exitosa, pueden encadenarse uno detrás de otro.

.catch()

Maneja los errores, tanto de la promesa como en los `then()`

.finally()

Se ejecutará siempre luego de los `then()` o los `catch()`



Encadenar Promesas

```
miPromesa
  .then(function (respuesta) {
    // trabajo con la respuesta
  })
  .catch(function (error) {
    // reaccionar al error ocurrido
  })
  .finally(function () {
    // acciones indistintamente del resultado
  });
```

Métodos de Promesas

Promise.all

Espera a que todas las promesas sean resueltas o **una rechazada**

Promise.allSettled

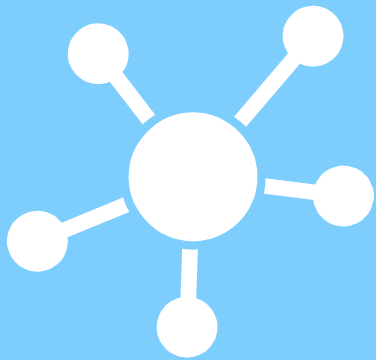
Espera a que todas las promesas sean completadas

Promise.any

Devuelve la primer promesa que finalice correctamente

Promise.race

Devuelve la primer promesa que sea resuelta o rechazada



FETCH API

Nos permite realizar consultas HTTP y cargar archivos de forma asíncrona mediante el uso de promesas

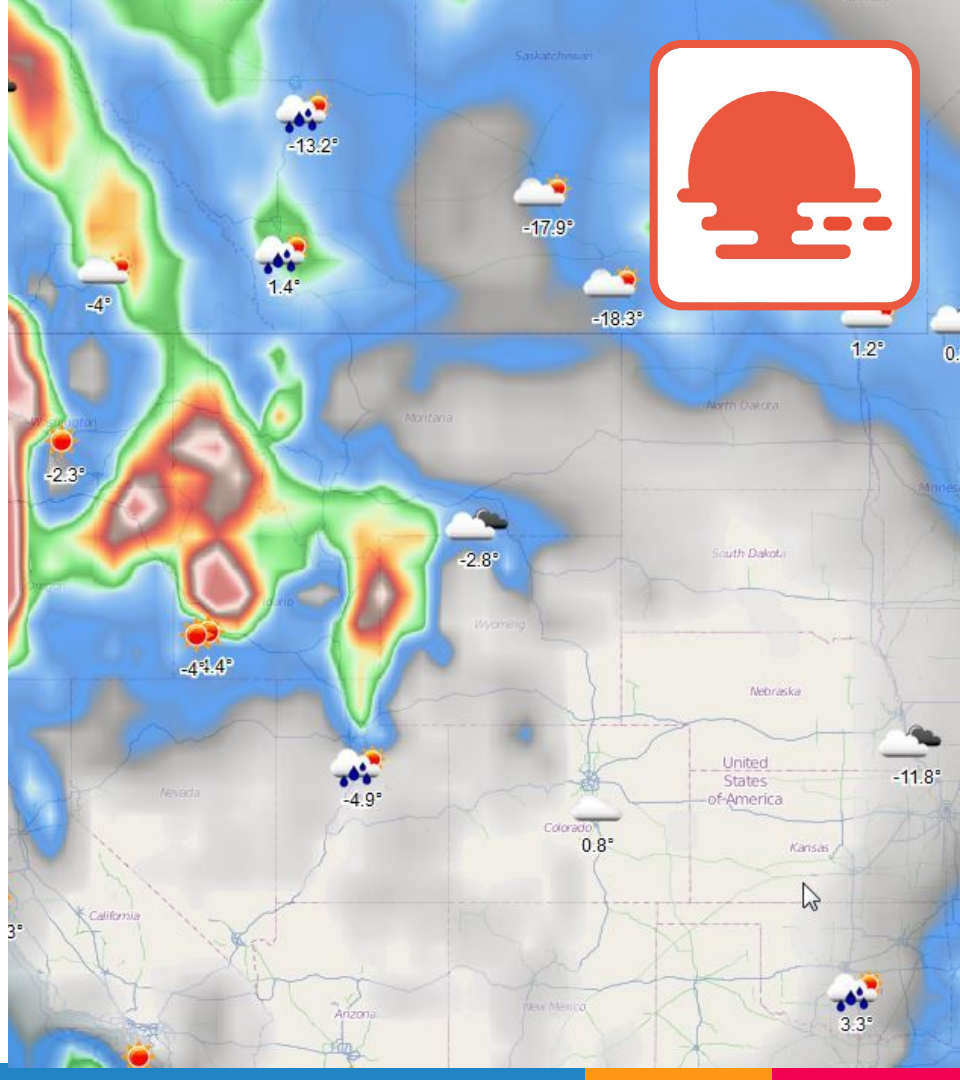
Consultar API

```
fetch("url", { opciones })  
  .then(function (respuesta) {  
    // trabajo con la respuesta  
  })  
  .catch(function (error) {  
    // reaccionar al error ocurrido  
  })  
  .finally(function () {  
    // acciones indistintamente del resultado  
  });
```

Open Weather Map

API gratuita que nos permite conocer el estado del tiempo consultando por ciudad.

<https://openweathermap.org/>



Obtener Respuesta

```
fetch("url", { opciones })  
  .then(function (respuestaAPI) {  
    return respuestaAPI.JSON();  
  })  
  .then(function (respuesta) {  
    // trabajar con la respuesta de la api  
  })  
  .catch(function (error) {  
    // reaccionar al error ocurrido  
  });
```


Opciones Fetch

```
{  
  method: "POST", // Por defecto es GET  
  body: {  
    // Las peticiones GET no llevan body  
    JSON.stringify(data);  
  },  
  headers: {  
    'Content-type': "application/json"  
  }  
}
```