Programación con JavaScript II

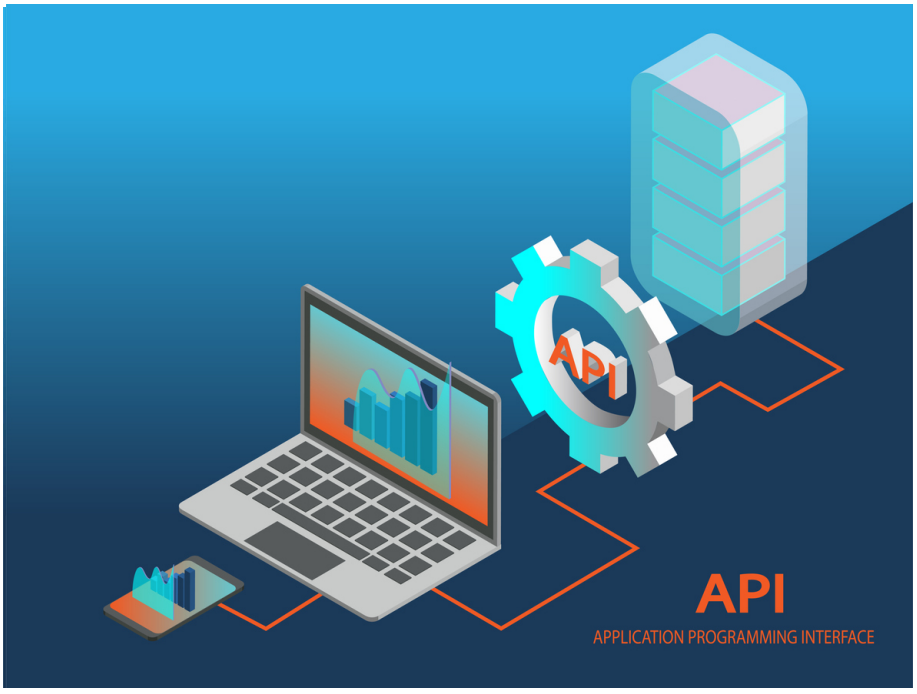
API

Introducción

Los desarrollos web frecuentemente requieren información que debe ser solicitada a bases de datos para que esta se utilice en múltiples procesos. Por ejemplo, si se requiere autenticar a un usuario, se pide a un servidor que verifique si el nombre de usuario y password proporcionados son correctos, esto con el fin de permitir o negar el acceso. Este proceso lo realizan independientemente de la plataforma o el tipo de base de datos que almacena la información a través de una API *(Application Programming Interface)*, es decir, un intermediario que traduce las peticiones del cliente al servidor.

Al hacerlo de esta forma se usa una abstracción de información, ya que no importa la forma en la que se encuentren los datos dentro del servidor, pues se tiene una entidad llamada API que ofrece lo que se busca tal y como se espera. Entre el cliente y el servidor estará la API que conoce de antemano las reglas de comunicación.

En este tema revisarás el proceso de comunicación entre el cliente y el servidor mediante una API. En este caso, la API estará implementada sobre una arquitectura REST y usará un formato llamado JSON (muy parecido a XML). Dicho formato tiene reglas específicas que permiten obtener, crear, borrar o modificar datos desde peticiones a través del protocolo HTTP.



Explicación

JSON

En cualquier organización se toman decisiones diarias basadas en información histórica que por lo regular se encuentra almacenada en diferentes plataformas. De allí surge la imperiosa necesidad de establecer una estructura o formato que sea sencillo de interpretar, sin importar la plataforma en la que se guarda o la que requiere utilizarla. Así es como surgen formatos como XML, CSS y JSON.

JSON es el acrónimo de JavaScript Object Notation, es decir, Notación de Objetos de JavaScript. Su objetivo es generar una estructura de datos ligera, simple de interpretar e independiente de cualquier lenguaje de programación.

Según Microsoft (2022), el uso del formato JSON es muy popular debido a que se trata de documentos más pequeños que los archivos XML y son consumidos por servicios RESTful, o sea, servicios que contienen API que se basan en la arquitectura REST.

Sus ventajas son las siguientes:

- Establecen una estructura jerárquica clara, lo que facilita ordenar sus datos y relacionarlos.
- Es compatible para la mayoría de los lenguajes de programación.
- Es posible convertir un archivo JSON en una estructura de base de datos relacional.
- Es un formato que se utiliza en bases de datos conocidas como “no SQL” como MongoDB, Couchbase y Azure Cosmos DB.

JSON ofrece un formato de texto con dos estructuras:

1. Un conjunto de pares de datos: Nombre + Valor, que puede ser interpretado como un objeto.
2. Una lista de valores, similar a un arreglo.

Sintaxis del JSON

1. El archivo debe iniciar y terminar con llaves: { }.
2. Para declarar arreglos se usan los corchetes: [ ]. Ejemplo [“Rojo”,”Azul”,”Negro”].
3. Tanto los títulos como los valores deben estar encerrados con comillas dobles “ ”, a menos que sean números. No se permiten comillas simples.
4. Se utilizan dos puntos “:” para especificar el nombre de un campo.
5. Los datos booleanos deben tener valores *True* o *False*.
6. Los datos deben estar separados por comas.

Explicación

```
{
  "nombre":"Juan Pérez",
  "profesion":"Contador",
  "edad":30,
  "paqueteria":["Excel","Word","PowerPoint"],
  "disponibilidadParaViajar":true,
  "experiencia": {
     "antigüedad": 12,
     "nivel": "Gerente"
  }
}
```

Ejemplos de estructuras JSON.

Como puedes observar, los siguientes tres ejemplos contienen la misma información, pero tienen una estructura diferente:

- **Versión 1:** la información se encuentra en un objeto de datos llamado “Capitales” y los valores se encuentran en un arreglo por pares con un título: estado y capital.
- **Versión 2:** aquí puedes observar que las capitales están en un objeto con un arreglo de pares de datos, pero no tienen un título.
- **Versión 3:** solo aparecen los pares datos de información sin declarar un objeto ni los títulos de cada par de información.

Explicación

Lo anterior da una idea de la versatilidad del formato JSON.

Capitales_v1.json

```
{
 "Capitales":[{
       "estado":"Aguascalientes",
       "capital":"Aguascalientes"
    },
    {
       "estado":"Baja California",
       "capital":"Mexicali"
    },
    {
       "estado":"Baja California Sur",
       "capital":"La Paz"
    },
    {
       "estado":"Campeche",
       "capital":"San Francisco"
    },
    {
       "estado":"Chiapas",
       "capital":"Tuxtla Gutiérrez"
    },
    {
       "estado":"Chihuahua",
       "capital":"Chihuahua"
    },
    {
       "estado":"Coahuila",
       "capital":"Saltillo"
    }
 ]
}
```

Capitales_v2.json

```
{
   "Capitales":[{
         "Aguascalientes":"Aguascalientes",
         "Baja California":"Mexicali",
         "Baja California Sur":"La Paz",
         "Campeche":"San Francisco",
         "Chiapas":"Tuxtla Gutiérrez",
         "Chihuahua":"Chihuahua",
         "Coahuila":"Saltillo"
      }
   ]
}
```

Capitales_v3.json

```
{
   "Capitales":[{
         "Aguascalientes":"Aguascalientes",
         "Baja California":"Mexicali",
         "Baja California Sur":"La Paz",
         "Campeche":"San Francisco",
         "Chiapas":"Tuxtla Gutiérrez",
         "Chihuahua":"Chihuahua",
         "Coahuila":"Saltillo"
      }
   ]
}
```

Limitaciones de JSON:

Según desarrolloweb (s.f.), el formato JSON tiene las siguientes limitaciones:

1. **No tiene un esquema:** de antemano no conoces la estructura de los datos que permita la corrección de objetos o validación de información.
2. **No contiene tipo de datos:** Como podemos observar el formato JSON no incluye una especificación del tipo de datos. Eso implica que al recibirlos en codigo JavaScript se requiere hacer una conversión para determinar si se trata de datos que serán tratados del tipo “short”, “long” o “float”.
3. **No tiene la capacidad de incluir comentarios.** Al abrir un archivo con format JSON no es posible incluir comentarios que expliquen sus partes.

Explicación

Las siguientes funciones de JavaScript pueden ser de gran utilidad para convertir objetos desde una cadena de caracteres:

- ***PARSE*(cadena):** permite tomar una cadena de caracteres que tenga la estructura JSON para convertirla en un objeto. En el siguiente ejemplo puedes observar cómo se declara el objeto "empleado".

Sintaxis: JSON.**parse**(text[, reviver])

Donde:
Text: es la cadena de texto que se convertirá en formato JSON.
Reviver (opcional): parse() puede convertirse en una función que prescribe cómo serán tratados los valores antes de convertirlos a la cadena JSON. Requiere de dos parámetros *Key* (llave asociada al valor) y Value (valor que convertirá).

```
const empleado = JSON.parse('{"nombre": "juan", "edad": 33, "casado": false}');
```

Al probar el siguiente código se podrán comprobar los valores en la consola del navegador:

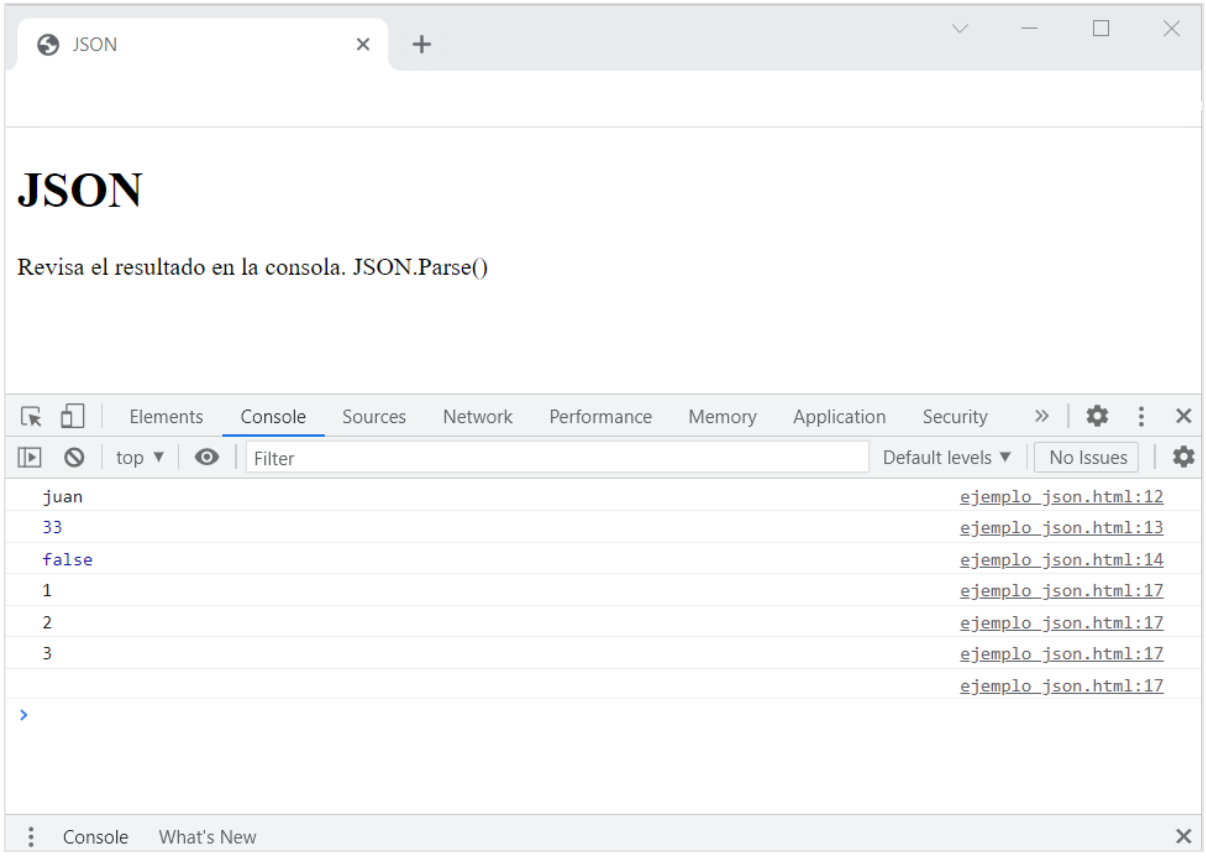
En la primera parte se define la constante obj y su estructura la obtiene de la cadena de caracteres JSON. Más abajo aparece un ejemplo del método parse(), incluyendo las funciones KEY y VALUE.

Explicación

Aquí se observa que los datos se presentan en pares, indicando la llave y su valor correspondiente.

```
<!DOCTYPE html>
<html lang="en-us">
<!DOCTYPE html>
<html lang="en-us">
 <head>
  <title>JSON</title>
 </head>
<!DOCTYPE html>
<html lang="en-us">
 <head>
  <title>JSON</title>
 </head>
 <body>
    <h1>JSON</h1>
    <p>Revisa el resultado en la consola. JSON.Parse()</p>
    <script>
     const obj = JSON.parse('{"nombre": "juan", "edad": 33, "casado": false}');

     console.log(obj.nombre);
     console.log(obj.edad);
     console.log(obj.casado);

     JSON.parse('{"1": 1, "2": 2, "3":3}', (key, value) => {
     console.log(key);
     return value;
     });
    </script>
 </body>
</html>
```

Explicación



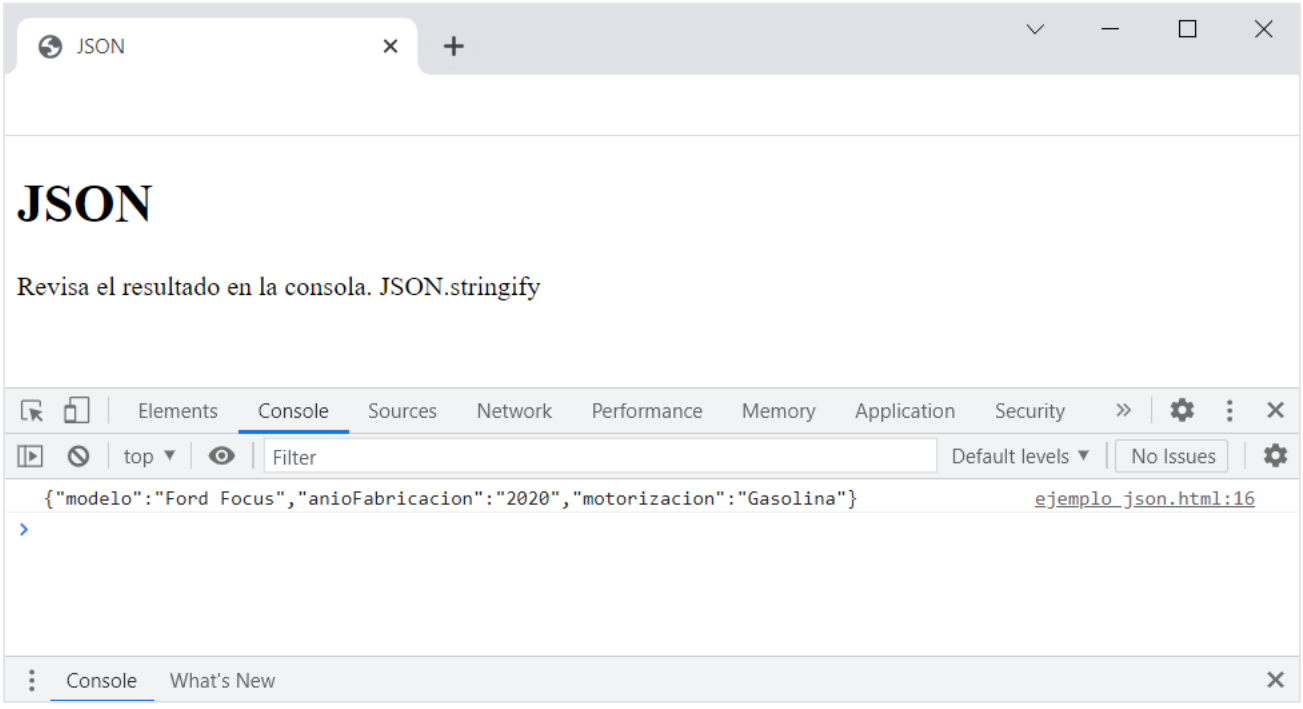
JSON.stringify(): esta función permite realizar la acción contraria a PARSE(), es decir, convertir un objeto a una cadena con formato JSON.

Sintaxis: JSON.**stringify**(value[, replacer[, space]])

Donde:
Value: es el valor por convertir a formato JSON.
Replacer (opcional): permite cambiar el comportamiento de la conversión de la cadena de texto en caso de omitirse las propiedades del objeto incluidas en la cadena JSON.
Space: se utiliza cuando es necesario agregar un espacio en blanco dentro de la cadena

Explicación

```
<script>
      var coche = {
       modelo: "Ford Focus",
       anioFabricacion: "2020",
       motorizacion: 'Gasolina'
       }
       var cadena = JSON.stringify(coche);
       console.log(cadena);
     </script>
```



Explicación

REST

Según un artículo de Amazon (2022), una API (Application Programming Interface) es un mecanismo que permite el intercambio de datos entre aplicaciones de diferentes estructura o configuración. Existen API's para la mayoría de las plataformas y permiten consumir información de diferentes orígenes. Es muy parecido a contar con un traductor cuando visitas un país de un idioma desconocido. Para entablar una comunicación eficiente el traductor será la interfaz que logra que dos personas puedan entenderse.

Las API's deben apegarse a una definición de reglas previamente establecidas entre las plataformas. Estas reglas se encuentran claramente definidas en lo que se conoce como documentación de una API. A la entidad solicitante se le conoce como **CLIENTE,** mientras que a la entidad que responde se le conoce como **SERVIDOR.**

En la actualidad las API REST son las más populares y flexibles (Amazon, 2022). Son confiables, eficientes y escalables (MDN Web Docs, 2022). El cliente realiza peticiones al servidor y el servidor ofrece una respuesta enviando datos mediante el protocolo HTTP.

Cuando un sistema se apega a estas reglas se le llama **RESTful.**

El acrónimo REST significa Transferencia de Estado Representacional. Sus funciones más comunes son POST, GET, PUT, DELETE.

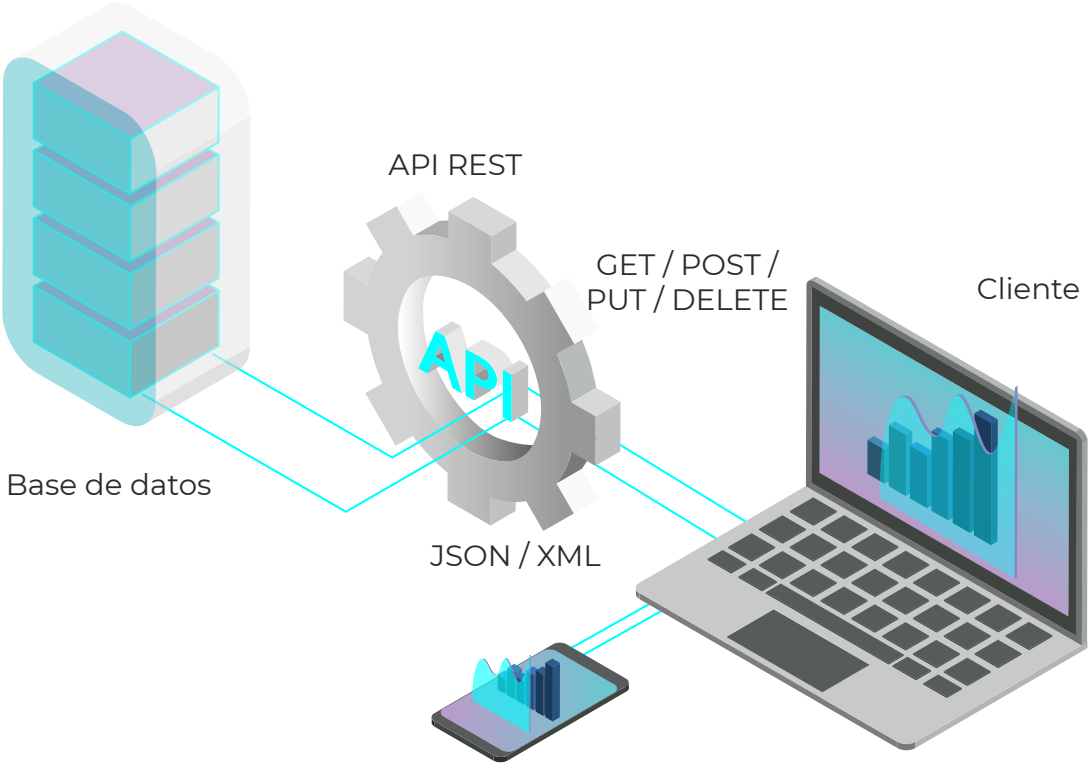
Existen 5 principios fundamentales en este estilo arquitectónico de software:

1. **Arquitectura cliente – servidor:** divide las funciones entre el cliente y el servidor. Cada uno no requiere conocer las funciones de su contraparte.
2. **Ausencia de estado:** el servidor no lleva el registro del estado de las solicitudes realizadas por el cliente, en otras palabras, no se requiere recordar el estado del intercambio de mensajes
3. **Uso de caché:** se requiere el uso de un control de caché que permita identificar si las solicitudes son almacenadas temporalmente "cacheables" antes de que lleguen al servidor, permitiendo que sea más eficiente la comunicación entre el cliente y el servidor cuando se desea recuperar el mismo recurso en varias ocasiones.
4. **Sistema por capas:** el cliente solo necesita conocer la capa de datos a la que está le está hablando.
5. **Interfaz uniforme:** existe una independencia entre la interfaz y las respuestas del servidor. La información se transmite de manera estandarizada.

El funcionamiento de REST consiste en el siguiente proceso: cuando un cliente envía una solicitud de información a través de una API REST, esta entrega la petición al servidor y la respuesta estará en uno de los siguientes formatos: JSON (JavaScript Object Notation), Python, PHP, HTML, XLT o simple texto.

En la siguiente imagen puedes observar que el cliente realiza peticiones a la API y esta, a su vez, solicita la información al servidor. La respuesta del servidor la regresa a la API en formato JSON o XML.

Explicación



Códigos de los estados

Web FX (s.f.) ofrece un glosario de los códigos de los estados HTTP que usa el servidor para notificar el estado de la respuesta. Seguramente el 404 te será muy familiar, ya que es un error frecuente al navegar por Internet. En una petición POST, la respuesta esperada sería el estado 201 (creado).

Código	Descripción
200	OK
201	Creado
302	Encontrado
400	Solicitud inválida
404	No encontrado
500	Error interno en el servidor

Explicación

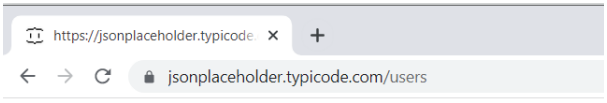
Los siguientes son ejemplos de una petición API REST:

Ejemplo 1

Observa que en el fragmento de código JavaScript se utiliza un objeto XMLHttpRequest que abre una petición al servidor que se encuentra en jsonplaceholder a través del método GET.

```
(() => {
   const xhr = new XMLHttpRequest(),
    $xhr = document.getElementById("xhr"),

   xhr.addEventListener("readystatechange", (e) => {
    if (xhr.readyState !== 4) return;

   //abre una petición AJAX con el método GET hacia la URL de jsonplaceholder
   xhr.open("GET", "https://jsonplaceholder.typicode.com/users");
   xhr.send();
 })();
```

A continuación, observa que el resultado es una estructura JSON con datos de los usuarios.

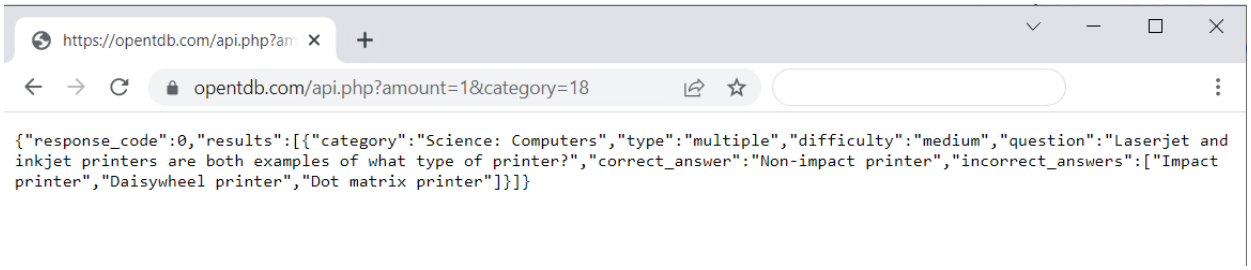



```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
  },
  {
    "id": 2,
    "name": "Ervin Howell",
    "username": "Antonette",
    "email": "Shanna@melissa.tv",
    "address": {
      "street": "Victor Plains",
      "suite": "Suite 879",
      "city": "Wisokyburgh",
      "zipcode": "90566-7771",
      "geo": {
        "lat": "-43.9509",
        "lng": "-34.4618"
      }
    },
```

Explicación

Ejemplo 2

En el ejemplo anterior, la petición se realiza en el archivo llamado api.php, en donde se incluye el método GET para obtener 1 registro de la categoría 18 que, para efectos prácticos de la estructura de datos, corresponde a la categoría ***"Science: Computers".*** Enseguida se observa el resultado de la petición GET.



Métodos HTTP

Ahora se continuará con los métodos disponibles por el protocolo HTTP/HTTPS para realizar peticiones a un servidor en un ambiente RESTful. Estos métodos también se conocen como **verbos HTTP.**

La IETF (2022) explica los métodos HTTP que se pueden utilizar para intercambiar datos. A continuación, se describen los más comunes:

- GET: con este método puedes hacer solicitudes de información hacia un recurso. El servidor responderá hacia la API REST solicitante con datos sin ningún efecto sobre la información enviada, por lo que se considera un método "seguro" para la integridad de los datos en el servidor. En algunos casos, este método es ideal porque puede ser utilizado a través de la dirección URL.

- POST: este método trabaja principalmente con colecciones de recursos. Es un método que permite solicitar la creación de un nuevo registro. Por lo regular, este se usa en foros de Internet, en procesos para suscribirse en listas de correo o para completar alguna compra en línea.

- PUT: con este método se solicita al recurso destino que cree o actualice un registro.

- DELETE: permite que el recurso destino elimine un registro.

Resumen de los Métodos HTTP más usados

Función	Descripción	Descripción
GET	Obtener datos	GET /v1/empleados/1.1
PUT	Actualizar datos	PUT /v1/empleados/1.1
POST	Crear un nuevo recurso	POST /v1/empleados/1.1
DELETE	Eliminar un recurso	DELETE /v1/empleados/1.1

Explicación

Observa que una buena práctica es incluir la versión en la URL y que el recurso es un sustantivo en plural. Por ejemplo, en la siguiente instrucción: GET /v1/empleados/1.1, aparece la versión 1 y el recurso es "empleados". Hay que evitar el uso de guiones medios o guiones bajos o el uso de verbos.

También existen otros métodos más especializados, aunque rara vez se requieren usar:

- ***CONNECT***: establece una conexión segura de comunicación, creando un túnel bajo el protocolo TCP/IP desde el origen hasta el destino, es decir, un mecanismo de encriptación de datos que evita que el mensaje sea interceptado por un intruso en la red pública de Internet. Este método se utiliza frecuentemente en conexiones que requieren confidencialidad de información.
- ***PATCH:*** es un método que solicita modificaciones parciales en un recurso existente.
- ***OPTIONS:*** este método se requiere para describir las opciones de comunicación para el recurso destino.

Ejemplos de métodos HTTP

El siguiente código muestra la solicitud del cliente usando el método GET. En este caso, el código supone la existencia de una API REST que se está ejecutando de forma local, es por ello que se utiliza localhost en la URL.

```
<script>
const request = new XMLHttpRequest();
const url = 'http://localhost:5000/movies';
request.open("GET", url);
request.send();

request.onload = (e) => {
    alert(request.response);
}
```

Explicación

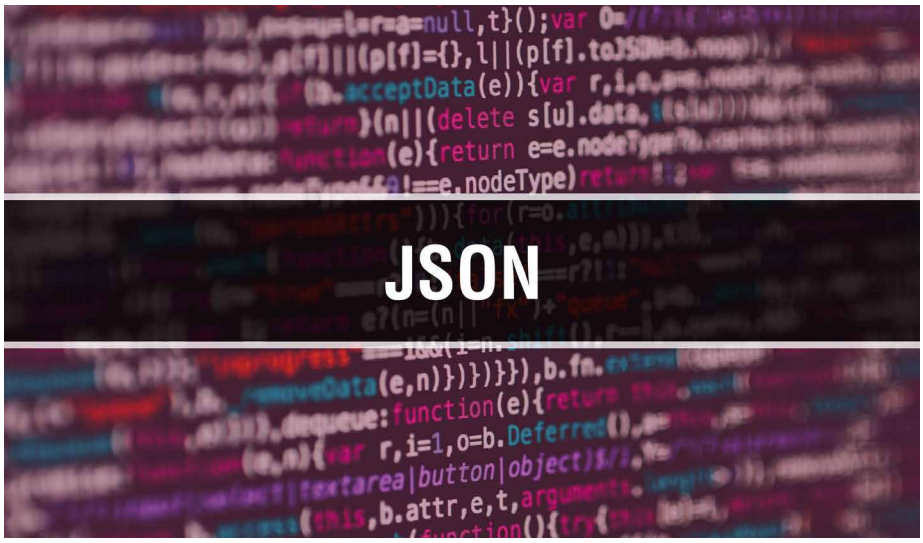
En este ejemplo se observa el método PUT para agregar el título de la película Matrix a la base de datos mediante una estructura JSON.

```
<script>
    const response = new XMLHttpRequest();

    const json = JSON.stringify({
        title: "The Matrix",
        year: "1999"
    });

    response.open("POST", 'http://localhost:5000/');
    response.setRequestHeader('Content-Type', 'application/json');

    response.send(json);

    response.onload = (e) => {
        alert(response.response);
    }
</script>
```

Cierre

Las API son elementos muy necesarios en los desarrollos web, ya que sin ellos sería necesario codificar bajo las reglas que dicte la plataforma sobre la que se encuentre la base de datos de un servidor, lo que limitaría su implementación y escalabilidad.

En este tema se explicó cómo es el mecanismo de comunicación de una API que sea RESTfull, ya sea para solicitar, actualizar o borrar información contenida en una base de datos, basándose en una estructura de datos ligera como lo es JSON. Además, se analizaron los métodos más comunes de intercambio de datos que usan los métodos HTTP.



Referencias bibliográficas

- AWS. (s.f.). *Qué es un API.* Recuperado de https://aws.amazon.com/es/what-is/api/
- Desarrolloweb.com. (s.f.). *JSON.* Recuperado de https://desarrolloweb.com/home/json#track68
- IETF. (2022). *HTTP Semantics.* Recuperado de https://datatracker.ietf.org/doc/html/rfc9110#section-9.3.4
- MDN. (s.f.). *REST*. Recuperado de https://developer.mozilla.org/es/docs/Glossary/REST
- Microsoft. (2022). *Trabajo con archivos CSV y JSON en soluciones de datos.* Recuperado de https://learn.microsoft.com/es-es/azure/architecture/data-guide/scenarios/csv-and-json
- Web FX. (s.f.). *HTTP Status Codes Glosary.* Recuperado de https://www.webfx.com/web-development/glossary/http-status-codes/

Para saber más

Lecturas

Para conocer más acerca de **API**, te sugerimos leer lo siguiente:

- Dos setenta. (s.f.). *¿Qué es una api rest?* Recuperado de https://dossetenta.com/que-es-una-api-rest
- MDN. (s.f.). *Mensajes HTTP*. Recuperado de https://developer.mozilla.org/es/docs/Web/HTTP/Messages
- Red Hat. (2020). *¿Qué es una API de REST?* Recuperado de https://www.redhat.com/es/topics/api/what-is-a-rest-api

Videos

- Carlos Azaustre - Aprende JavaScript. (2022, 19 de mayo). *CÓMO CONSUMIR UN API con JAVASCRIPT desde la web* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=2Xm9P_tXtK8
- Curious Code. (2020, 22 de septiembre). *HTTP Request Methods | GET, POST, PUT, DELETE* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=tkfVQK6UxDI
- Emprinnos. (2022, 14 de junio). *Como usar JSON en JavaScript* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=9he4OewlYFo

Checkpoints

Asegúrate de:

- Identificar la estructura de datos JSON para recuperar y enviar información a un servidor web.
- Comprender las características de una API REST que permite interactuar con los datos de un servidor.
- Utilizar los métodos HTTP para recuperar, modificar y borrar información de un servidor mediante peticiones a una API.

Requerimientos técnicos

- Usar Visual Studio Code.(https://code.visualstudio.com/)
- Acceso a una plataforma con servidor web (Live Server)
- JSON Server para probar API's RESTFull: https://jsonplaceholder.typicode.com/

Prework

- Deberás reconocer las secciones HTML y Javascript en una estructura de código.
- Comprender las estructuras del código de este tema y probarlas.
- Deberás identificar los objetos XMLHttpResponse y Fetch.