**M2S11: API (Consumo y exposición vía DOM)**

**Hola, espero te encuentres muy bien, feliz inicio de semana. Antes de iniciar me gustaría que veas un video que te brinda unos tips para estudiar cuando no se tienen muchas ganas, y permitir con ello que tu cuerpo y mente se puedan preparar para recibir nueva información**[**Presiona aquí para que veas el video.**](https://www.youtube.com/watch?v=10WOfvDnFXY)

**¿Cómo te sientes?, espero que muy bien; sigamos adelante. ☺️**

**ÍNDICE**

* [Conceptos de API Rest](https://github.com/U-Camp/BOOT-M2-SEM11#api-rest)
* [Consumo de un API](https://github.com/U-Camp/BOOT-M2-SEM11#consumo-de-una-api)
  + [Repositorios públicos](https://github.com/U-Camp/BOOT-M2-SEM11#repositorios-p%C3%BAblicos)
  + [Autorizaciones](https://github.com/U-Camp/BOOT-M2-SEM11#autorizaciones)
  + [Fetching](https://github.com/U-Camp/BOOT-M2-SEM11#fetching)

**API Rest**

Para comunicar el frontend con el backend (propio o de un servicio de terceros) y establecer un formato estándar de datos que ambas plataformas entiendan, se necesita un protocolo que permita éste flujo de datos, ésta es la funcionalidad principal de API REST, el cual es un protocolo de intercambio y manipulación de datos que se usa en los servicios de internet, es una interfaz entre sistemas que usa HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML o JSON.

Así mismo, una petición a una API REST siempre devuelve una respuesta con información del estado del proceso de los datos de la petición, normalmente se usa el formato JSON para este tipo de comunicación entre cliente / servidor.

En la actualidad, no existe proyecto o aplicación web que no disponga de una API REST para la creación de los servicios profesionales a partir del software, por lo que su uso se vuelve necesario durante la impartición del curso.

Una API REST cuenta con diferentes características que hacen posible la comunicación:

* Protocolo cliente / servidor sin estado: Esto indica que toda petición realizada por medio de HTTP contiene toda la información necesaria para realizarla sin necesidad de que ni el cliente ni el servidor necesiten recordar ningún estado previo para satisfacerla.
* Operaciones: Son aquellas que definen el tipo de petición e indican como debe actuar el servidor dependiendo de ese tipo. Las más importantes son GET (para obtener datos del backend), POST (para crear datos en el backend), PUT (para actualizar datos en el backend) y DELETE (para eliminar datos en el backend).
* URI: Es el identificador único de cada recurso en el sistema REST, implementa una interfaz uniforme sobre los recursos para facilitar el proceso de la información, conocido formalmente como endpoint.
* Hipermedio: Es una extensión del concepto de Hipertexto, en el caso de una API REST, este proporciona la capacidad de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

Una petición a una API REST siempre devuelve una respuesta con información del estado del proceso de los datos de la petición, normalmente se usa el formato JSON para este tipo de comunicación entre cliente/servidor. JSON es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de Javascript, son cadenas que pueden ser convertidos a objetos nativos de Javascript para tener acceso a los datos, los cuales pueden incluir tipos de datos básicos (cadenas, números, arreglos, booleanos y otros literales de objeto), lo que permite construir una jerarquía de datos.

Una API Rest, normalmente es código backend ejecutándose del lado del servidor usando algún lenguaje de programación. Como ya se mencionó, NodeJS es Javascript del lado del servidor, pero para poder cumplir cabalmente con la especificación Rest, tiene que ser RESTFul, esto se logra implementando un módulo llamado ExpressJS. ExpressJS es el framework web más popular de NodeJS, es la librería subyacente para un gran número de otros frameworks de NodeJS populares. Proporciona mecanismos para:

* Escritura de manejadores de peticiones con diferentes verbos HTTP (Métodos de petición y consulta, como GET, POST, PUT y DELETE) con diferentes rutas (URLs), lo que hace que su uso nativo de REST APIs sea fácil de implementar.
* Añadir procesamiento de peticiones middleware adicionales a cualquier punto dentro del pipeline del manejo de peticiones REST.

**Hola, si deseas puedes consultar y descargar una infografía que resumen un poco lo visto sobre API Rest**[**Presiona aquí**](https://github.com/U-Camp/BOOT-M2-SEM11/blob/main/infografias/Semana_11_APIREST.pdf)

**Seguimos con Consumo de una API**

**Consumo de una API**

El proceso para consumir un servicio API está dividido en 3 partes:

1. Identificación de rutas en la documentación del servicio de API.
2. Elegir el método que establecerá cómo vamos a consumir esa ruta.
3. Gestionar la respuesta.

Con esto claro, un servicio API podemos desarrollarlo nosotros mismos, o, buscar organizaciones que publiquen sus propios servicios para que accedamos a ellos y obtengamos los datos necesarios. Vamos a comenzar buscando en repositorios públicos.

**Repositorios públicos**

Es muy posible que quieras practicar o buscar servicios API para una necesidad específica que tengas.

Existen múltiples repositorios que contienen servicios API públicos.

De los más conocidos es Public APIS, el cual incluye diferentes servicios API divididos por temáticas. Da click [aquí](https://github.com/public-apis/public-apis).

En ese espacio, vas a encontrar diferentes servicios APIs, varios gratuitos y otros de paga.

Te recomendamos probar con uno de los más sencillos incluidos ahí: [Dog API](https://dog.ceo/dog-api/)

Accede y lee la documentación. Observa que muchas de las opciones incluidas ahí te establecen una ruta específica:

https://dog.ceo/api/breeds/list/all

Esto significa que, si abres este link en tu navegador, se consumirá un servicio API y te regresará un JSON.

{

"message": {

"affenpinscher": [],

"african": [],

"airedale": [],

"akita": [],

// ...

},

"status": "success"

}

Hemos realizado nuestro primer consumo en un servicio API público.

**Autorizaciones**

Un servicio API puede decidir que tan público o privado pueden exponer sus datos.

Primordialmente, nos enfocaremos en dos tipos de accesos:

* **Acceso público**

Puedes consumir directamente el recurso del servicio API sin ningún tipo de permiso.

Al colocar la URL en el navegador, obtendrás sin problema un archivo .JSON.

Ejemplos:

https://dog.ceo/api/breeds/image/random

{

"message": "https://images.dog.ceo/breeds/redbone/n02090379\_3415.jpg",

"status": "success"

}

* **Acceso con API KEY**

Existen servicios que quieren cerrar algunos datos a menos que te registres y/o te identifiques. Esto por una cuestión de seguridad y conocer también qué personas están consumiendo el API de su servicio.

Por ello, habrá sitios que te pedirán que te registres. Te comparto algunos servicios API con registro:

* [Exchange-Rate - Servicio para consultar divisas](https://www.exchangerate-api.com/)
* [Open Weather Map - Servicio para conocer el clima del mundo](https://openweathermap.org/api)

Una vez registrado, obtendrás un API KEY. Puede ser que lo recibas por correo electrónico, o bien, en el área de tu perfil.

A continuación deberás leer la documentación para saber cómo integrarla. Usualmente va acompañado dentro de la URL donde consumimos el servicio.

Observa el ejemplo siguiente:

https://api.openweathermap.org/data/2.5/weather?q=London&appid={API key}

* Observa que la primera parte se está estableciendo el protocolo (https)
* Posteriormente, vemos la dirección host (api.openweathermap.org)
* Luego entramos a los diferentes parámetros.
* Finalmente, verás un query string el cual inicia con un signo de interrogación. Luego, hay una letra que dice q, significando query que anexa London. Y, el otro query stringes appid, donde se coloca el API KEY que nos dieron al registrarnos.

Con esto, puedes acceder a APIs privados. Si no pasas el API Key, te negaran el acceso.

**Fetching**

Para realizar el consumo de datos, usaremos fetch.

fetch puede ser utilizado únicamente en el Frontend. Es decir, del lado del cliente.

Vamos a situarnos en un archivo js y vamos a consumir los datos del servicio API con este formato:

fetch("https://dog.ceo/api/breeds/image/random")

.then((response) => {

return response.json()

})

.then((data) => {

console.log(data)

return data

})

**¡Muy bien, llegaste al final del contenido de esta semana! recuerda que puedes leer la información cada vez que lo consideres y escribirle a tus coaches en caso de ser necesario.**

**¡Nos vemos luego! 😉**