Técnico en DESARROLLO DE SOFTWARE

Programación Avanzada

Programación Avanzada

Semana IV

1. ¿Qué son los JSON?

Los JSON (JavaScript Object Notation) o Notación de Objetos de JavaScript, son la forma en la que se representan los objetos en el lenguaje de JS.

Más que eso ahora los JSON son usados como un estándar de formato de información usado en las transferencias de datos en la red.

A pesar de tener JavaScript en el nombre no es estrictamente usado en JS, de hecho es una forma de representar objetos que muchos lenguajes saben leer y escribir, por lo que es posible encontrarlo en lenguajes ajenos a JS como Python o PHP.

Los JSON tienen una estructura bastante simple:

- Están conformados por pares clave-valor
- Las claves van a ser cadenas de caracteres o Strings
- Todos los pares clave-valor están separados por coma
- Los valores pueden ser tipos de datos numéricos, strings o incluso pueden ser otros JSON.
- Un JSON está contenido dentro de un par de llaves.
- Todos los valores de cadenas de caracteres, incluidas las claves de un JSON tienen que estar dentro de comillas dobles (" ") no simples (' ')

Entonces un JSON se puede ver de esta forma

```
{
        "clave1": 10,
        "clave2": "valor2",
        "clave3": {
```

```
                "sub_clave1": "valor3"
        }
}
```

Los tipos de datos de cadena de caracteres se colocan dentro de comillas, los demás tipos de datos como: números o booleanos no.
Entonces, supongamos que estamos desarrollando una aplicación donde tenemos que guardar información de un usuario, incluyendo nombre, email y número de teléfono.
El JSON correspondiente al objeto que representa el usuario se vería así:

```
{
        "nombre": "Nombre Completo",
        "email": "correo@servidor.com",
        "telefono": "(502) 1234 - 5678"
}
```

Los JSON también pueden contener arreglos, estos se representan con [ ] y los elementos también se separan por coma, en el caso de los arreglos no se colocan claves. Los arreglos se usan de la misma manera que en otros lenguajes, accediendo a los elementos por medio de índices.

Por ejemplo, si tenemos una lista de usuarios a los que les interesa un tema en particular, y quieren que se les informe cuando se realice una publicación nueva, podríamos tener un JSON parecido a este:

```
{
        "tema": "Computación",
        "usuarios_suscritos": [
                {
                        "nombre": "Nombre Completo",
```

```
                "email": "correo@servidor.com",
                "telefono": "(502) 1234 - 5678"
            },
            {
                "nombre": "Nombre Completo",
                "email": "correo@servidor.com",
                "telefono": "(502) 1234 - 5678"
            }
        ]
}
```

Puede ser problemático escribir o formatear un JSON, por eso existen herramientas que podemos usar como JSONLint que nos dicen si la estructura de nuestros JSON es correcta.

2. Manipulando un JSON

En JS existen 2 funciones para manipular JSON.
La primera para convertir un objeto JSON a su representación en String
La segunda que realiza el proceso inverso, convertir de un String a un JSON, siempre y cuando el String contiene un JSON válido

La primera función se llama stringify, es una función estática de la clase JSON, es decir no es necesario crear una instancia de JSON para utilizar esta función

Se llama sobre un JSON de esta forma

```
JSON.stringify(myJSON)
```

La segunda función se llama parse, lo que hace es obtener el valor JSON de un string

Parse se utiliza de esta forma:

```
JSON.parse(stringJSON)
```

Además de parse existe otra función que nos permite obtener un JSON, la función, apropiadamente, se llama json, y el resultado es una promesa. Veremos cómo usar esta función más adelante.

Uno de los usos más prominentes de los JSON, como se mencionó al principio de la sección es que son usados como estándar en la transferencia de datos. Si usamos un servicio web, es muy posible que la información que solicitamos nos la envíen en formato de JSON.

Recorriendo los JSON

A un objeto JSON no se le accede por su índice, se accede por el nombre de la propiedad

Para acceder al contenido de un arreglo se hace de esta forma:

```
arreglo[1]
```

Eso nos devuelve el valor que tiene el arreglo en la posición 1,

Un JSON no tiene información en posiciones numéricas, recordemos que es la representación de un objeto. Entonces accedemos a la información de la siguiente forma:

```
json['nombre']
```

Es importante que coloquemos el nombre del campo dentro de comillas.

Por ejemplo, para el siguiente JSON de un estudiante:

```
{
        "nombre": "Juan",
        "carnet": "1001A",
        "edad": 30
}
```

Para obtener el nombre, no lo haríamos así:

```
estudiante[0]
```

Sino así:

```
estudiante['nombre']
```

También existe la llamada notación punto, es la forma de acceder a las propiedades de un objeto por medio de un operador punto. Esta notación se ve así:

```
estudiante.nombre
```

Ambas notaciones son válidas, sin embargo, una diferencia importante es que con la notación de arreglo podemos acceder a propiedades por medio de variables, por ejemplo lo siguiente es válido solamente con la notación de arreglo:

```
var estudiante = {
  "nombre": "Juan",
  "id": 101
}
var propiedad = 'nombre'
console.log(estudiante[propiedad])
```

Más sobre JSON

https://www.json.org/json-en.html

APIs

Los APIs (Application Programming Interface) son Interfaces que nos permiten comunicarnos con un servicio y obtener información, muchas veces serán usados en servicios de red a través del internet, por lo que hacer uso de las llamadas HTTP es importante para el uso de las APIs.

Los APIs nos permiten comunicarnos con servicios haciendo uso de una interfaz, de manera que podemos crear código modular y eliminar dependencias en los programas que queremos realizar.

También es posible hacer uso de APIs de terceros, obteniendo información sobre servicios que otros desarrolladores ofrecen, por ejemplo un servicio como Google Maps, tiene un API que nos permite obtener información sobre una ubicación en particular.

Los APIs son el punto de acceso a un servidor o recurso, por medio de métodos. Es similar a como realizamos la definición de los métodos CRUD para acceder a una base de datos.

Una API va a tener una colección de métodos y comportamientos que nos permite obtener información de un servicio, y dicha información generalmente va a hacer uso de un formato de intercambio de información, existen dos ampliamente utilizados, el XML y el JSON.

La mayoría de los APIs retornan la información en formato JSON

Más información sobre los APIs
https://www.redhat.com/es/topics/api/what-is-a-rest-api
https://www.youtube.com/watch?v=u2Ms34GE14U

Más sobre XML

https://www.mundolinux.info/que-es-xml.htm

https://www.youtube.com/watch?v=cQnwAoW80r0

https://www.w3schools.com/js/js_json_xml.asp

Peticiones de red.

¿Cómo nos podemos comunicar con un API? Muchas veces las APIs con las que tratamos no son parte de nuestro proyecto, son parte de un servicio, que puede ser de desarrollo interno, o un servicio de terceros. Entonces no podemos acceder al API con simplemente declarar la ubicación del archivo y obtener la información como hemos realizado con bases de datos anteriormente.

Hacemos uso de las API por medio de peticiones de red, una peticion de red es una manera de obtener información de un servicio remoto. Cuando hablamos de peticiones a traves de internet se hace uso de un protocolo llamado HTTP

¿Suena conocido? Pues debería, el protocolo HTTP es la forma en la que un navegador obtiene la información de una página web para luego mostrarnos el contenido que queremos. Seguramente han tratado con una dirección web que empieza con http:// o https:// esto se refiere a que la dirección se va a acceder por medio del protocolo http.
HTTP es el protocolo más conocido, HTTPS es una evolución del protocolo HTTP, es idéntico a HTTP con la excepción de que HTTPS usa un canal seguro para transmitir información.

La petición HTTP tiene un parámetro que todos conocemos: el URL que estamos solicitando. El URL de la petición también puede incluir parametros en la petición, y puede tener varios tipos que cambian el comportamiento de la peticion.

Existen varios tipos de peticiones por medio de HTTP, pero las dos más utilizadas son GET y POST:

- GET: Es posiblemente el tipo de petición más común, se trata de una solicitud para consultar un servicio o información, este es un tipo de petición que usan los navegadores web para obtener información de una página web. Una petición de tipo GET tiene parametros adicionales dentro del URL, por ejemplo

  ```
  http://sitioweb.com/busqueda?tema=computacion&publicado=2020
  ```

- POST. Este tipo de petición se utiliza cuando queremos enviar información en la petición HTTP, un uso de caso común es en el envío de formularios cuando queremos guardar información en la base de datos de nuestro servicio. Las peticiones de tipo POST no envían los parámetros en el URL de la petición, sino lo envían como datos adicionales que acompañan a la petición HTTP.

No entraremos a ver a profundidad todo lo que son los protocolos de transferencia de datos ni todos los detalles de lo que es HTTP. Basta con saber que es un protocolo de transferencia de datos que nos permite hacer solicitudes, enviar datos, recibir respuestas y comunicarnos con servidores y otros computadores a través de la red.

¿Cómo realizamos una petición hacia un API?

Primero vamos a obtener el URL del recurso que queremos, luego tenemos que hacer una petición GET de HTTP a través de la red, la petición es hacia uno de los métodos de la interfaz, este método va a retornar cierta información en un formato determinado, usualmente en formato JSON

Esto lo vamos a realizar haciendo uso de una herramienta de JS que nos facilita las peticiones HTTP, el método fetch.

Más sobre HTTP

https://www.youtube.com/watch?v=dExb007D4TY

https://www.youtube.com/watch?v=iYM2zFP3Zn0

https://developer.mozilla.org/es/docs/Web/HTTP/Methods

https://profesores.virtual.uniandes.edu.co/~isis3710/dokuwiki/lib/exe/fetch.php?media=temas:http-guia.pdf

https://web.stanford.edu/~ouster/cgi-bin/cs142-fall10/lecture.php?topic=http

Peticiones de red con el método fetch

fetch es un método del paquete de red de JS, lo que hace es manejar las conexiones y las peticiones HTTP por nosotros, es posible definir las diversas propiedades de la petición que queremos hacer, pero en muchas ocasiones basta con colocar el URL que queremos obtener.
fetch nos va a entregar la respuesta del servicio o API en una promesa. Gracias al comportamiento asincrónico de las promesas, nos permite reaccionar a la respuesta cuando esta lista.

Para poner manos a la obra, vamos a utilizar un API y el método fetch para poder explicar todo lo que hemos visto

la forma de usar fetch es así:

```
fetch( URL )
```

fetch nos devuelve una promesa, entonces lo tenemos que trabajar en la parte then de la ejecución.

Un dato importante es que en el then no estamos trabajando con un JSON, en el then estamos trabajando con el resultado de la petición, puede que no sea un json,

entonces tenemos que convertirlo a json, usando la función que habíamos mencionado antes: json()
Veamos cómo funciona el proceso de las peticiones por medio de un ejemplo con un API.

El api que vamos a utilizar se llama 7timer, es un API de clima, lo pueden encontrar en la siguiente dirección: http://www.7timer.info/doc.php?lang=en

El url que vamos a usar para realizar peticiones a este API es el siguiente: http://www.7timer.info/bin/api.pl?lon=113.17&lat=23.09&product=civillight&output=json

Si colocamos este URL en nuestro navegador web podemos ver la respuesta del API
Esto lo colocamos de parámetro en la función fetch:

```
fetch('http://www.7timer.info/bin/api.pl?lon=113.17&lat=23.09&product=civillight&output=json')
```

Y luego podemos usar la función then para obtener el resultado:

```
fetch('http://www.7timer.info/bin/api.pl?lon=113.17&lat=23.09&product=civillight&output=json')
        .then(res =>{
        })
```

Este resultado no es un JSON ni un String, por lo que no podemos manejarlo haciendo uso de las funciones JSON.parse y JSON.stringify, lo que tenemos que hacer es hacer uso de la función json() de esta forma:

```
function realizarPeticion(){
fetch('http://www.7timer.info/bin/api.pl?lon=113.17&lat=23.09&product=civillight&output=json')
```

```
    .then(res =>{
        res.json()
            .then(resJSON=>{
            })
    })
```

json() devuelve una promesa, entonces podemos también usar then sobre él.

Ahora vamos a colocar esto en una aplicación, inicializamos una aplicación de electron, recordemos incluir la dependencia de electron con:

```
npm i --save-dev electron
```

Y luego vamos a colocar los siguientes archivos:
El main script

```
const {app, BrowserWindow} = require('electron')

function createWindow(){
    const ventana = new BrowserWindow({
        width : 500,
        height : 500,
    })
    ventana.loadFile('index.html')
}

app.whenReady().then(createWindow)
```

Recordemos cambiar el nombre del main script en el package.json.
El index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>Clima</title>

   </head>
   <body>
      <table id="tablaResultados" class="table">
         <tbody id="contenido">

         </tbody>
       </table>
      <input type="button" value="Realizar Peticion" id="peticion">
            <script type="text/javascript" src="index.js"></script>
   </body>
</html>
```

El renderer del index.

```
var con = document.getElementById('contenido')
var bot = document.getElementById('peticion')

bot.addEventListener('click', reaizarPeticion)

function realizarPeticion(){

fetch('http://www.7timer.info/bin/api.pl?lon=113.17&lat=23.09&product=civillight&o
utput=json')
      .then(res =>{
         res.json()
```

```
            .then(resJSON=>{
                var data = resJSON['dataseries']
            })
        })
}
```

En el renderer lo que estamos haciendo es obtener el boton y el div del contenido, le colocamos el eventListener al botón y lo que vamos a hacer es realizar la petición http por medio de fetch cuando hagamos clic en él.
Convertimos el resultado de fetch a un json y luego obtenemos la información del clima, que en el API de 7timer se encuentra en un elemento llamado 'dataseries'

Ahora con esa data podemos mostrar la información en nuestra aplicación de esta forma:

```
...
var data = resJSON['dataseries']
for(var i = 0; i < data.length; i++){
    var dia = data[i]
    con.innerHTML += dia['date']+" "+dia['weather']
                +" "+dia['temp2m']['max']+"<br>"
}
...
```

Como la dataseries es un arreglo, lo podemos recorrer por medio de índices. Cada elemento del arreglo es otro JSON por lo que podemos acceder a los atributos de este por medio de la notación de arreglo o la notación punto.

¿Por qué no podemos simplemente mostrar la información del JSON? Si tratamos de colocar el contenido de día directamente en el HTML nos va a mostrar de contenido

[object Object] esto es porque es un objeto, no un String. Lo podríamos mostrar de esta forma con.innerHTML += JSON.stringify(dia).

Sin embargo, luego no podríamos manipular la información ni mostrarla de manera más legible o atractiva.

El index.js completo se vería de esta forma:

```
var con = document.getElementById('contenido')
var bot = document.getElementById('peticion')

bot.addEventListener('click', realizarPeticion)

function realizarPeticion(){

fetch('http://www.7timer.info/bin/api.pl?lon=113.17&lat=23.09&product=civillight&output=json')
        .then(res =>{
            res.json()
                .then(resJSON=>{
                var data = resJSON['dataseries']
                for(var i = 0; i < data.length; i++){
                    var dia = data[i]
                    con.innerHTML += dia['date']+" "+dia['weather']
                                +" "+dia['temp2m']['max']+"<br>"
                }
                })
        })
}
```

Más sobre fetch

https://developer.mozilla.org/es/docs/Web/API/Fetch_API

https://davidwalsh.name/fetch

Un primer vistazo a Bootstrap

Ahora, si queremos que nuestra aplicación sea más atractiva lo que tenemos que hacer es crear una hoja de estilos CSS. Sin embargo, no es la única manera de realizar esto, hay hojas de estilos y herramientas disponibles, que podemos utilizar para dar estilo a nuestras aplicaciones. Bootstrap es un framework de diseño de front-end que contiene muchas opciones ya prediseñadas para que podamos aplicar estilos modernos a nuestras aplicaciones.

Podemos descargar las hojas de estilo de Bootstrap en el siguiente enlace https://getbootstrap.com/docs/4.0/getting-started/download/

O si no queremos descargar e instalar Bootstrap podemos colocar lo siguiente en nuestro archivo HTML:

```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JX
m" crossorigin="anonymous">
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
crossorigin="anonymous"></script>
```

Vamos a cambiar un poco los archivos de html y el renderer para mostrar ahora la información en una tabla.

El HTML se vería así:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Clima</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet"                                                integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
  </head>
  <body>
    <table class="table">
      <thead>
        <tr>
          <th scope="col">#</th>
          <th scope="col">Date</th>
          <th scope="col">Weather</th>
          <th scope="col">Max</th>
        </tr>
      </thead>
      <tbody id="contenido">

      </tbody>
    </table>
    <input type="button" value="Realizar Peticion" id="peticion">
      <script type="text/javascript" src="index.js"></script>
```

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.j
s"                                                    integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>

   </body>
</html>
```

Al fondo podemos ver la hoja de estilo de bootstrap. Luego colocamos un elemento table, algunos encabezados y el contenido de la tabla. Cambiamos el id de "contenido" hacia el cuerpo de la tabla ya que es ahí donde se colocará la información. Y colocamos algunos encabezados para mostrar que significa toda la información.
La clase table, es una clase que pertenece a la hoja de estilos de Bootstrap, mientras que la propiedad scope indica cual es el elemento header de una fila o columna. El renderer se vería así:

```
var con = document.getElementById('contenido')
var bot = document.getElementById('peticion')

bot.addEventListener('click', realizarPeticion)

function realizarPeticion(){

fetch('http://www.7timer.info/bin/api.pl?lon=113.17&lat=23.09&product=civillight&o
utput=json')
      .then(res =>{
         res.json()
            .then(resJSON=>{
```

```
                var data = resJSON['dataseries']
                for(var i = 0; i < data.length; i++){
                    var dia = data[i]

                    con.innerHTML += "<tr><th scope=\"row\">"+ i+1+ "</th>"+
                                "<td>"+dia.date+"</td>"+
                                "<td>"+dia.weather+"</td>"+
                                "<td>"+dia.temp2m.max+"</td>"+
                                "</tr>"
                }
            })
        })
}
```

Lo que estamos haciendo es, en lugar de simplemente acceder a la información del JSON y mostrarla en la aplicación, formatear la información en forma de tabla.
Aquí tenemos también un header, que es el número de fila de la tabla y luego tenemos los otros elementos como celdas o datos de la tabla.

Algo muy importante que notar acá es cómo funcionan las promesas, podemos obtener el resultado de una promesa con then. Eso también aplica a promesas dentro de otro then como en el caso de la función json, entonces podemos reescribir el código que maneja esa promesa de la siguiente manera

```
fetch('http://www.7timer.info/bin/api.pl?lon=113.17&lat=23.09&product=civillight&output=json')
        .then(res =>res.json())
        .then(resJSON=>{
                var data = resJSON['dataseries']
                for(var i = 0; i < data.length; i++){
                var dia = data[i]

                con.innerHTML += "<tr><th scope=\"row\">"+ i+1+ "</th>"+
                                "<td>"+dia.date+"</td>"+
                                "<td>"+dia.weather+"</td>"+
                                "<td>"+dia.temp2m.max+"</td>"+
                                "</tr>"
                }
        })
```

Eso nos ayuda a que sea un poco más legible el código.

Más sobre las tablas de HTML

- https://www.w3schools.com/html/html_tables.asp
- https://www.w3schools.com/tags/att_th_scope.asp

Más sobre bootstrap

- https://getbootstrap.com/docs/4.0/getting-started/introduction/
- https://codingpotions.com/desarrollo-web-boostrap

Descargo de responsabilidad

La información contenida en este documento descargable en formato PDF o PPT es un reflejo del material virtual presentado en la versión online del curso. Por lo tanto, su contenido, gráficos, links de consulta, acotaciones y comentarios son responsabilidad exclusiva de su(s) respectivo(s) autor(es) por lo que su contenido no compromete al área de e-Learning del Departamento GES o al programa académico al que pertenece.

El área de e-Learning no asume ninguna responsabilidad por la actualidad, exactitud, obligaciones de derechos de autor, integridad o calidad de los contenidos proporcionados y se aclara que la utilización de este descargable se encuentra limitada de manera expresa para los propósitos educacionales del curso.

Galileo UNIVERSIDAD La Revolución en La Educación

GES Galileo Educational System