<codoacodo/>

Agencia de Aprendizaje a lo largo de la vida

FULL STACK FRONTEND
Clase 21

Javascript 8

BA

Peticiones Fetch

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 20

Asincronismo (Parte II)

- Introducción
- Sincronismo y Asincronismo
- Promesas
- Funciones del constructor y métodos

Clase 21

Peticiones Fetch

- Peticiones APIs Restul: Método GET
- Manipulación de la respuesta recibida.

Clase 22

Base de datos

Peticiones Fetch

El **Fetch API** es una tecnología moderna que permite realizar solicitudes **HTTP** de manera asincrónica desde el navegador. Es parte del estándar de **HTML5** y proporciona una manera poderosa y flexible de interactuar con servidores web. A diferencia **de XMLHttpRequest**, que era la forma anterior de realizar solicitudes, fetch ofrece una forma más sencilla y potente de trabajar con **promesas**, facilitando la escritura de **código asíncrono** que es más fácil de leer y mantener.

Fetch API es una herramienta esencial para los desarrolladores web modernos, proporcionando una interfaz limpia y fácil de usar para realizar solicitudes HTTP. A lo largo de esta clase, aprenderemos a utilizar fetch en diferentes escenarios, manejar respuestas de diversos tipos, y gestionar errores de manera efectiva.

Peticiones APIs RESTful: Método GET

El método GET se utiliza para recuperar información del servidor. No modifica los datos en el servidor; sólo solicita y recibe datos.

En Javascript, existen varias maneras de escribir código para solicitar la información al servidor a través del método GET. Veamos una de ellas con la api : **https://rickandmortyapi.com/api/character**

```
const options = {method: 'GET', headers: {accept: 'application/json'}};

fetch('https://rickandmortyapi.com/api/character', options)
  .then(response => response.json())
  .then(response => console.log(response))
  .catch(err => console.error(err));
```

Se define un objeto **options** que especifica el método HTTP a utilizar (**GET**) y los encabezados de la solicitud. En este caso, el encabezado **accept** se establece en **application/json**, lo que indica que el cliente (nuestro código) espera que la respuesta esté en formato JSON.

Peticiones APIs RESTful: Método GET

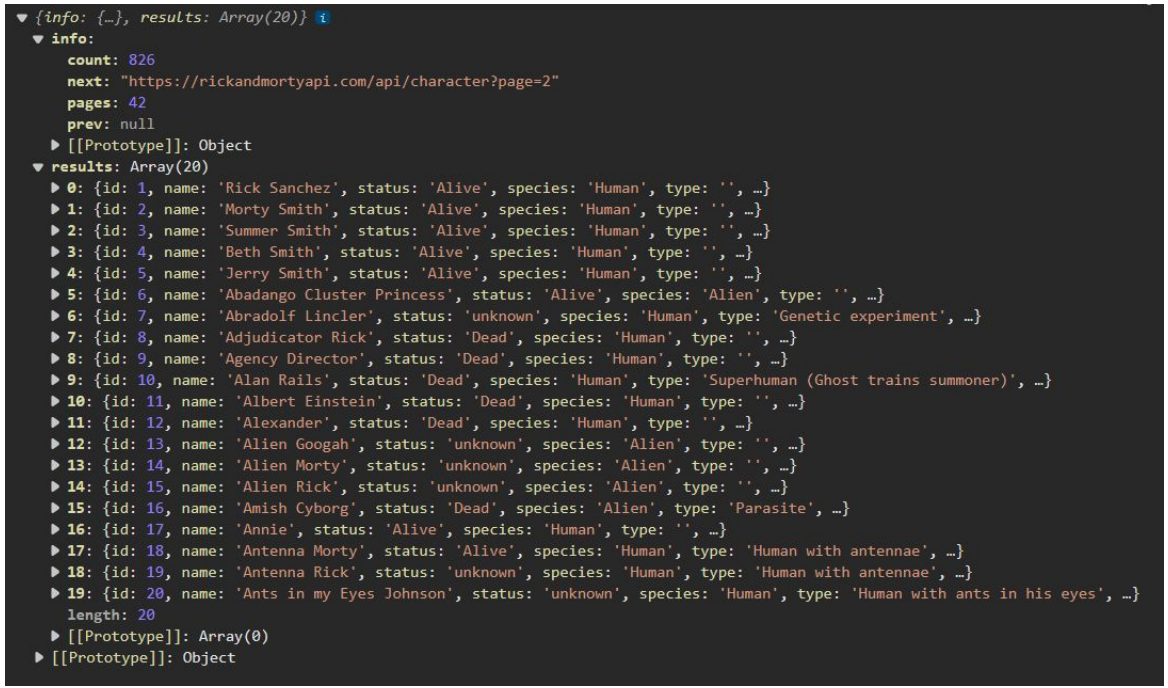
```
const options = {method: 'GET', headers: {accept: 'application/json'}};

fetch('https://rickandmortyapi.com/api/character', options)
  .then(response => response.json())
  .then(response => console.log(response))
  .catch(err => console.error(err));
```

Luego, se llama a la función **fetch** con la URL de la API de Rick and Morty y el objeto **options** como argumentos. **fetch** devuelve una promesa que se resuelve con el objeto **response** que representa la respuesta a la solicitud.

El método **then** se utiliza para especificar lo que debe hacerse una vez que se resuelve la promesa. En este caso, se llama al método **json** del objeto **response**, que también devuelve una promesa. Esta promesa se resuelve con el cuerpo de la respuesta parseado como JSON.

Finalmente, se utiliza otro **then** para manejar el JSON parseado. En este caso, simplemente se imprime en la consola. Si en algún punto del proceso ocurre un error (por ejemplo, si la solicitud falla), se captura con el método **catch** y se imprime en la consola.

Analizando la respuesta recibida



La respuesta **JSON** es usualmente recibida como una cadena de texto. En JavaScript, necesitamos convertir esta cadena de texto en **un objeto JavaScript** para poder manipularlo más fácilmente. Esto se hace utilizando **JSON.parse()**

Una vez que tienes el objeto JavaScript, puedes acceder a sus propiedades y valores para obtener la información específica que necesitas. Dependiendo de la estructura del JSON, estos datos pueden estar en forma de **objetos** anidados o **arrays**.

Validando los datos necesarios

```
const pedirDatos = () => {
    fetch('https://rickandmortyapi.com/api/character', options)
      .then(response => response.json())
      .then(response => {
          response.results.forEach((personaje) => {
              // Imprimir en consola el nombre e imagen de cada personaje
              console.log(personaje.name, personaje.image, personaje.species);
            });

        })
        .catch(err => console.error(err));
    }

pedirDatos();
```

Dentro del método **then**, se accede a la propiedad **results** de la **response**. **response.results** es un array de objetos, donde cada objeto representa a un personaje.

El método **forEach** se utiliza para iterar sobre cada objeto (o personaje) en el array **results**. Para cada personaje, se imprime en la consola el valor de las propiedades **name** e **image**. Es decir, para cada personaje, se muestra su nombre y la ruta donde se encuentra su imagen.

De esta manera, en la consola veremos los datos que vamos a utilizar para construir el **html** de cada personaje. Existen herramientas como **Postman** o **Insomnia** que nos permiten consumir apis. Las veremos más adelante en este mismo curso.

Preparando el HTML

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Rick And Morty Project</title>
</head>
<body>
    <header>
        <h1>Rick And Morty Project</h1>
    </header>
    <main id="personajes">
    </main>
    <footer>
        <p>Curso FullStack Node</p>
    </footer>
    <script src="index.js"></script>
</body>
</html>
```

Dentro de la estructura HTML definimos cual será el espacio para insertar los personajes.
En este caso identificaremos con el **id="personajes"** a la etiqueta HTML que tomaremos desde **Javascript** para agregar los elementos de manera dinámica.

Luego agregamos el archivo de javascript dónde estamos desarrollando el método. En este caso, lo agregamos después porque el index.js ejecutará la función inmediatamente cuando sea llamado.

Definiendo el HTML dinámico

```html
<article class="character">
    <img src="https://rickandmortyapi.com/api/character/avatar/1.jpeg" alt="Rick Sanchez">
    <h2>Rick Sanchez</h2>
    <div>
        <p>Human</p>
    </div>
</article>
```

```javascript
const article = document.createElement('article');
      article.setAttribute('class', 'character');
      article.innerHTML = `
      <img src="${personaje.image}" alt="${personaje.name}">
      <h2>${personaje.name}</h2>
      <div>
      <p>${personaje.species}</p>
      </div>`;

container.appendChild(article);
```

Si pensamos en escribir código HTML para mostrar un personaje, utilizaríamos algo similar al código que se encuentra arriba. Pero necesitamos hacerlo dinámico, para que se complete cada vez con los datos obtenidos desde la **api**. Debemos generar el código en javascript que producirá la misma estructura html pero irá cambiando los valores de acuerdo a los datos del objeto que iteraremos.

Incorporando el código en la iteración del objeto

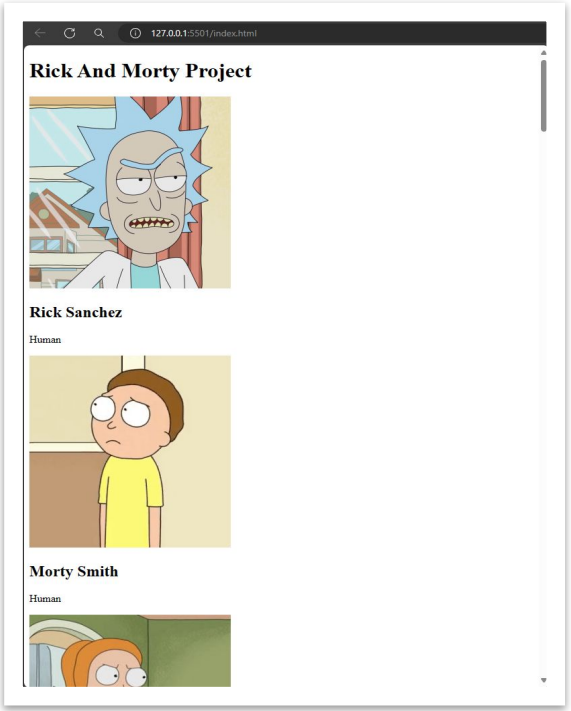
```
const options = {method: 'GET', headers: {accept: 'application/json'}};
const container = document.getElementById('personajes');

const pedirDatos = () => {
    fetch('https://rickandmortyapi.com/api/character', options)
      .then(response => response.json())
      .then(response => {
              response.results.map((personaje) => {
                  const article = document.createElement('article');
                  article.setAttribute('class', 'character');
                  article.innerHTML = `
                  <img src="${personaje.image}" alt="${personaje.name}">
                  <h2>${personaje.name}</h2>
                  <div>
                  <p>${personaje.species}</p>
                  </div>`;
                  container.appendChild(article);
          });

      })
      .catch(err => console.error(err));
}

pedirDatos();
```

Declaramos el contenedor anclándonos en el **id="personajes"** definido en el **HTML**.

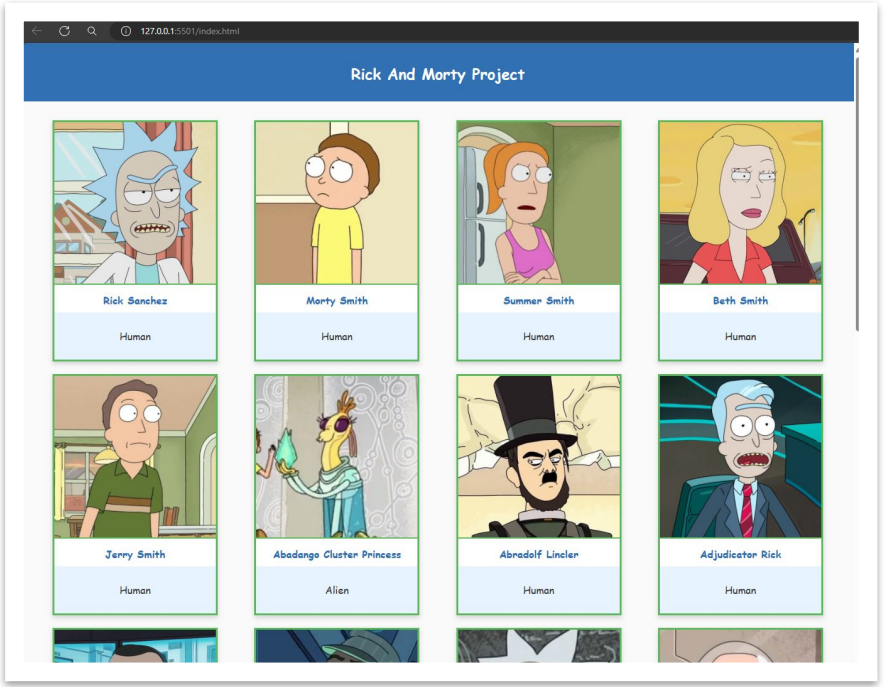
Incorporamos el código que se iterará por cada elemento del array generado con el json recibido.

Llamamos a la función para que se ejecute automáticamente cuando se cargue el archivo. También lo podemos hacer ante un evento determinado.

Resultado final



Aplicamos CSS



Material extra

No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.
- Realizar los Ejercicios obligatorios.

Todo en el Aula Virtual.

Muchas gracias por tu atención.

Nos vemos pronto