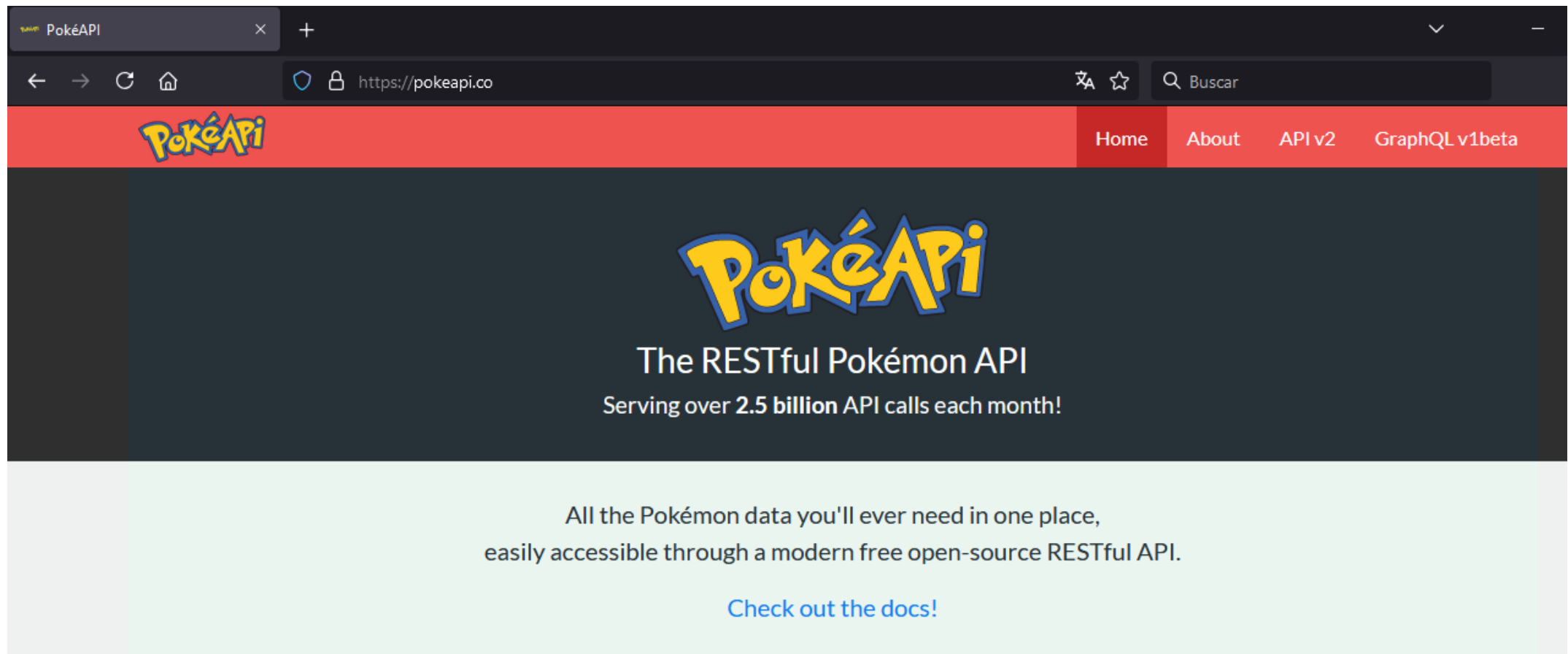


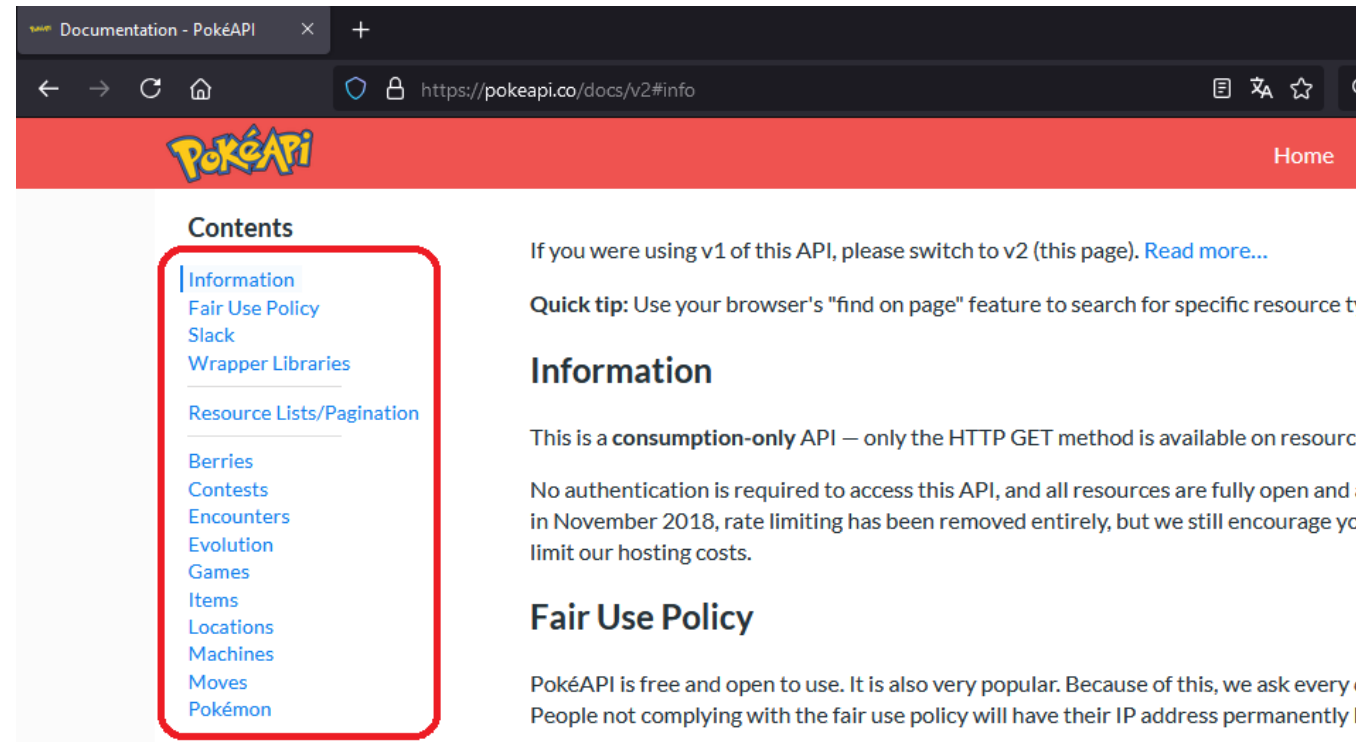
JavaScript

Consumiendo una API con JS

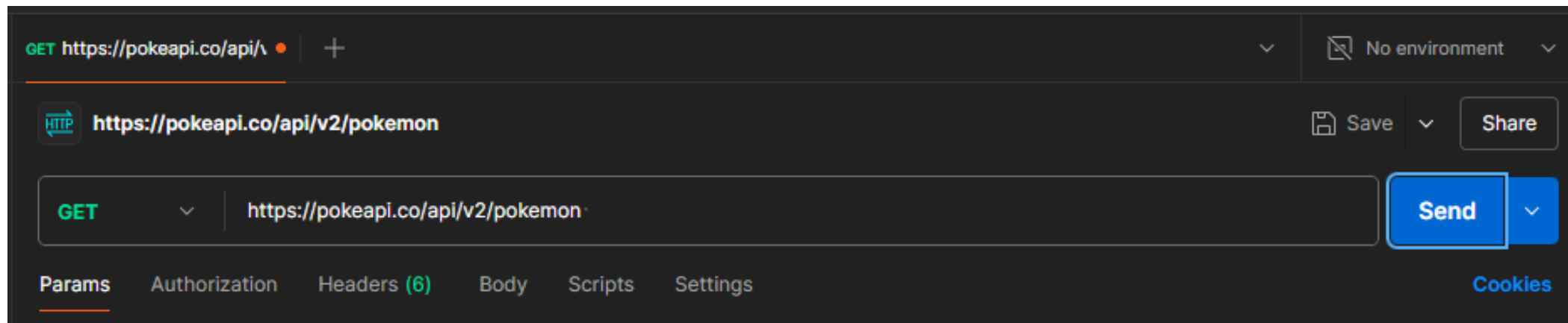
En este ejemplo vamos a consumir la API pública que está disponible en *https://pokeapi.co*, más conocida como PokeAPI



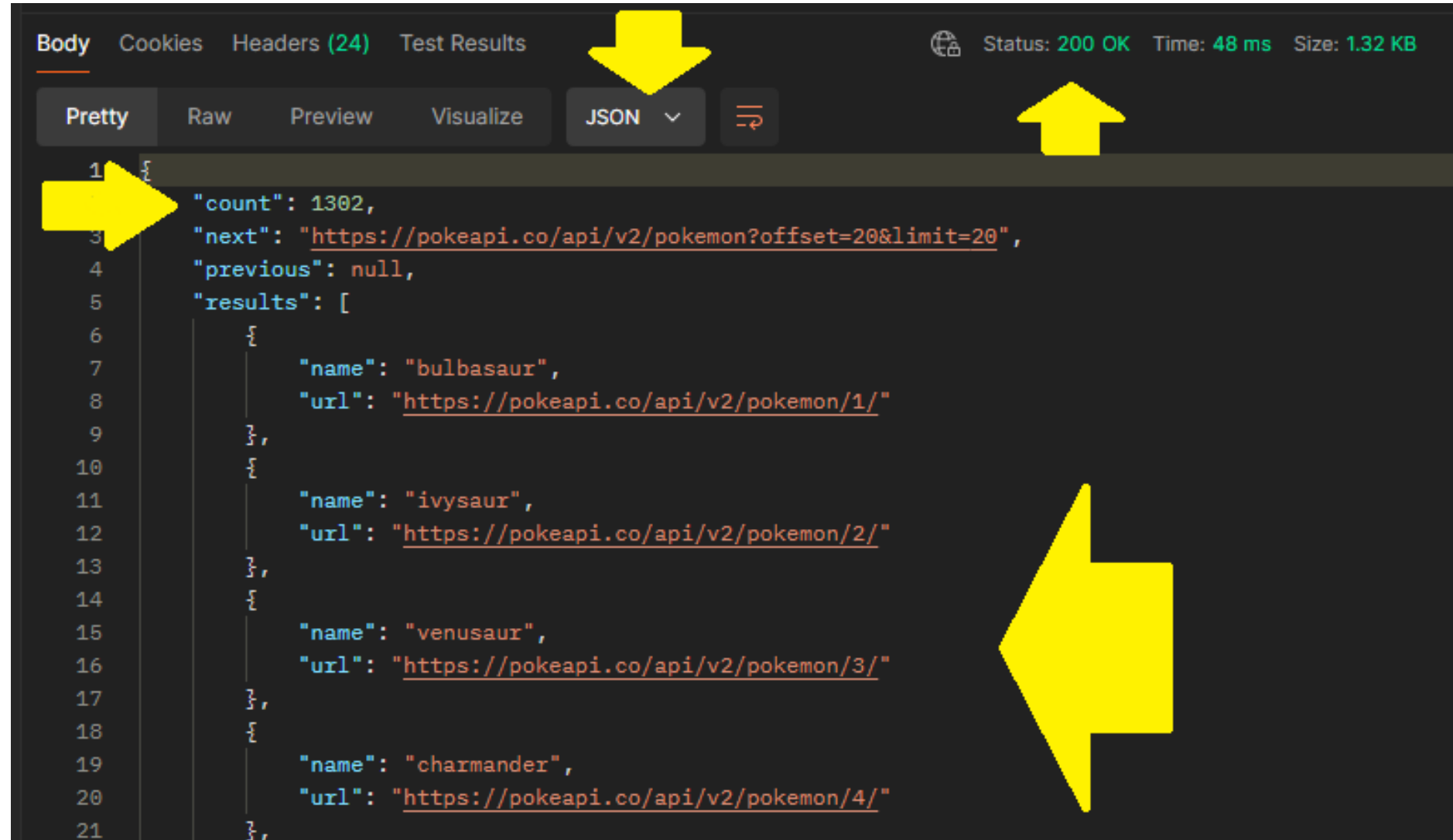
Lo primero que hay que hacer es leer la documentación de la API para ver las solicitudes HTTP que podemos realizar, para esta API en particular, en la documentación nos presenta muchas opciones para poder consultar.



En este primer ejemplo lo que nos interesa es obtener un listado de los pokemones disponibles, para ello analizando la documentación vemos que la URL a la que debemos consultar es ***https://pokeapi.co/api/v2/pokemon*** y el método HTTP para utilizar tiene que ser GET. Pues probemos realizar esa solicitud desde la herramienta *Postman* para ver que nos responde.



Aquí la
respuesta
obtenida:



```
Body Cookies Headers (24) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "count": 1302,
3   "next": "https://pokeapi.co/api/v2/pokemon?offset=20&limit=20",
4   "previous": null,
5   "results": [
6     {
7       "name": "bulbasaur",
8       "url": "https://pokeapi.co/api/v2/pokemon/1/"
9     },
10    {
11      "name": "ivysaur",
12      "url": "https://pokeapi.co/api/v2/pokemon/2/"
13    },
14    {
15      "name": "venusaur",
16      "url": "https://pokeapi.co/api/v2/pokemon/3/"
17    },
18    {
19      "name": "charmander",
20      "url": "https://pokeapi.co/api/v2/pokemon/4/"
21    },
22  ]
23 }
```

Analizando la respuesta lo primero que vemos es que nos responde con un código de estado 200 OK, por lo cual la URL que consultamos mediante el método HTTP GET es correcta.

También vemos que el formato de la respuesta es un objeto JSON.

Dicho objeto tiene mucha información, entre ella nos dice que el resultado total es de 1302 objetos que podríamos llamarle Pokemones, pero si prestamos atención vemos que solo nos trae los primeros 20, y los siguientes 20 están en la URL:

```
"count": 1302,  
"next": "https://pokeapi.co/api/v2/pokemon?offset=20&limit=20",
```

Posteriormente en un arreglo de objetos de nombre “**results**” nos muestra el nombre y la URL de cada Pokemon presente en la respuesta.

```
"results": [  
  {  
    "name": "bulbasaur",  
    "url": "https://pokeapi.co/api/v2/pokemon/1/"  
  },  
  {  
    "name": "ivysaur",  
    "url": "https://pokeapi.co/api/v2/pokemon/2/"  
  },  
]
```

Por lo tanto, si quisiéramos listar el nombre de los primeros 20 pokemones y su URL, tendríamos que acceder a este arreglo para iterarlo y mostrar a través de las propiedades “*name*” y “*url*” los datos que allí están presentes.

Para este primer ejemplo con esos datos es suficiente, veamos como listar esos primeros 20 pokemones en una tabla, mostrando su nombre y la url con los datos de cada pokemon, vamos a utilizar lo que ya hemos visto: fetchAPI de Javascript.

Lo primero que deberíamos hacer es construir en el frontend una tabla para presentar los datos.

Construimos una tabla aplicando algunas clases de Bootstrap. Dejamos libre el <tbody> de la tabla porque la idea es que los datos de la tabla se carguen desde JS, al obtener la respuesta de la API.

Como vamos a precisar acceder a esa parte de la tabla HTML es que le definimos un identificador, id="tabla", para poder capturarlo desde JS.

```
<main class="container-fluid">
  <section class="row">
    <section class="col-10 mx-auto mb-5">
      <table class="table table-striped text-center">
        <thead class="table-dark">
          <tr>
            <th>Pokemon</th>
            <th>URL</th>
          </tr>
        </thead>
        <tbody id="tabla"></tbody>
      </table>
    </section>
  </section>
</main>
```

Desde JS tenemos que capturar el elemento <tbody> para completarlo posteriormente.
La URL de la API que vamos a consultar la podemos guardar en una constante para utilizar luego.
Es recomendable escuchar el evento cuando se cargue el DOM HTML para así lanzar las funciones necesarias para consultar a la API.
Lo que acabamos de mencionar es lo siguiente:

```
const URL = "https://pokeapi.co/api/v2/pokemon";  
let tabla = document.querySelector("#tabla");  
  
document.addEventListener("DOMContentLoaded", iniciar);  
  
function iniciar(){  
  obtenerPokemones();  
}
```

Cuando el DOM HTML haya cargado se dispara la función `obtenerPokemones()`, esta función es la encargada de realizar la consulta a la API y esperar su respuesta, una vez que la respuesta sea obtenida, tendremos que presentar los datos en nuestro frontend, o sea, completar la tabla que hemos definido anteriormente.

```
async function obtenerPokemones(){
  try{
    const respuesta = await fetch(`${URL}`);
    const datos = await respuesta.json();

    //Tenemos el resultado en un array de objetos llamado "results" dentro de un Object
    console.log(datos);

    //Presentar los datos en el frontend
    frontend(datos.results);
  }catch(error){
    console.log(error);
  }
}
```

La función anterior realiza la solicitud HTTP GET a la API y queda a la espera de su respuesta.

Por el análisis que hicimos anteriormente con Postman, sabemos que los datos que queremos mostrar en el frontend vienen en un arreglo llamado “results” dentro de un objeto JSON, por lo tanto, definimos una nueva función llamada “frontend” que reciba un arreglo con los datos para presentar en nuestra tabla, el arreglo que debemos pasar a dicha función es “**datos.results**”, siendo “datos” la respuesta obtenida desde la API.

```
//Presentar los datos en el frontend  
frontend(datos.results);
```

Recordar que **datos.results** es un arreglo de objetos en JS, o sea:

[{...} , {...} , {...} ,]

La función frontend(arreglo), lo que debe hacer es:

- Mostrar los datos recibidos en la consola, para asegurarnos que recibimos los datos correctos
- Limpiar el cuerpo de la tabla HTML, así queda pronta para presentar nuevos datos
- Iterar el arreglo recibido, cada iteración es un nuevo Pokemon con su url
- En cada iteración debemos crear una nueva fila <tr>en la tabla
- Completar los datos de ambas columnas <td>, nombre y url

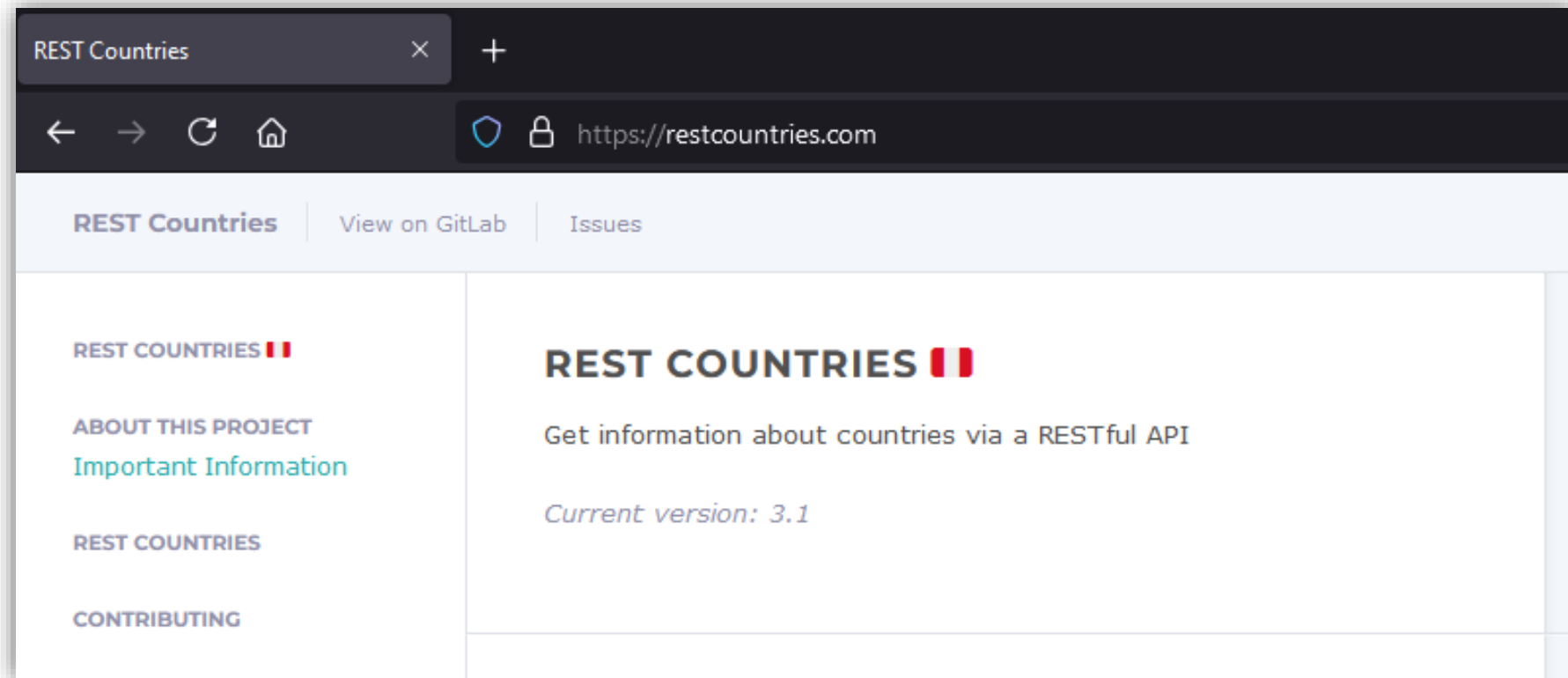
```
function frontend(arreglo){  
    //Los datos de cada objeto del arreglo son:  
    // nombre del pokemon  
    // url con los datos del pokemon  
    console.log(arreglo);  
  
    //Limpiamos el contenido de <tbody> de nuestra tabla  
    tabla.innerHTML = "";  
    //Vamos a mostrar esos datos en el frontend  
    for(let i=0; i<arreglo.length; i++){  
        tabla.innerHTML += `  
            <tr>  
                <td>${arreglo[i].name}</td>  
                <td>  
                    <a href="${arreglo[i].url}" target="_blank">${arreglo[i].url}</a>  
                </td>  
            </tr>  
        `;  
    }  
}
```

Con lo implementado anteriormente quedaría pronta nuestra tabla, la consulta a la API y la presentación de los datos obtenidos, si accedemos a nuestra página HTML veremos lo siguiente:

Consumiendo la **PokeAPI** con **JS**

Pokemon	URL
bulbasaur	https://pokeapi.co/api/v2/pokemon/1/
ivysaur	https://pokeapi.co/api/v2/pokemon/2/
venusaur	https://pokeapi.co/api/v2/pokemon/3/
charmander	https://pokeapi.co/api/v2/pokemon/4/
charmeleon	https://pokeapi.co/api/v2/pokemon/5/
charizard	https://pokeapi.co/api/v2/pokemon/6/
squirtle	https://pokeapi.co/api/v2/pokemon/7/

Veamos un nuevo ejemplo, vamos a consumir la API pública que está disponible en *https://restcountries.com*, para cargar un elemento `<select>` con los nombres de los países y su bandera.



Como es recomendable comenzamos leyendo la documentación de la API y realizando algunas pruebas con Postman para probar las URL y las respuestas.

REST COUNTRIES

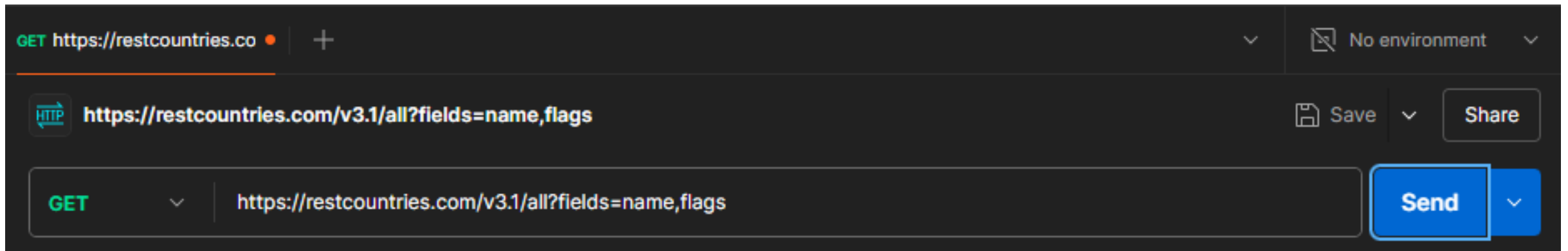
You can access API through <https://restcountries.com/v3.1/all> but in order to get a faster response, you should filter the results by the fields you need. Like

```
https://restcountries.com/v3.1/all?fields=name,flags`
```


Según la documentación podemos realizar una petición HTTP GET a la URL ***https://restcountries.com/v3.1/all*** para obtener todos los datos de los países, o filtrar por algunos datos, por ejemplo, nombre y bandera consultando la URL de la forma ***https://restcountries.com/v3.1/all?fields=name,flags***

Si consultamos por todos los datos podemos ver que la API nos retorna mucha información por cada país, y para nuestro ejemplo no es necesario, vamos a optar por realizar una petición HTTP GET filtrando por nombre y bandera.

Realizamos la petición HTTP GET, filtrando por nombre y bandera:



Y a continuación veremos la respuesta obtenida desde la API.

Body Cookies Headers (7) Test Results Status: 200 OK Time: 484 ms Size: 20.82 KB

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "flags": {
4       "png": "https://flagcdn.com/w320/gs.png",
5       "svg": "https://flagcdn.com/gs.svg",
6       "alt": ""
7     },
8     "name": {
9       "common": "South Georgia",
10      "official": "South Georgia and the South Sandwich Islands",
11      "nativeName": {
12        "eng": {
13          "official": "South Georgia and the South Sandwich Islands",
14          "common": "South Georgia"
15        }
16      }
17    }
18  },
```

Analizando la respuesta vemos que es en formato JSON, y nos retorna en un array de objetos todos los países que tiene.

De cada país nos informa la bandera y su nombre, entre otros datos en sus propiedades “flags” y “name”, que también contienen otras propiedades.

Esos datos son los que usaremos para cargar un elemento <select> en nuestro frontend HTML, pasemos a implementar la parte de nuestro HTML.

Para este ejemplo alcanza con implementar una etiqueta `<select>` y una etiqueta `` para mostrar la bandera.

```
<h1 class="text-center mt-3 display-4">
  <span class="fw-bold text-danger">Países</span> y
  <span class="text-success fw-bold">banderas</span> del
  <span class="text-primary fw-bold">mundo</span>
</h1>

<main class="container-fluid">
  <section class="row">
    <section class="col-4 mt-5 mx-auto">
      <select class="form-control" id="combo">
        <option value="vacio">Seleccionar país</option>
      </select>
    </section>
    <section class="col-4 mt-5 mx-auto">
      <img class="img-fluid" id="bandera" src="">
    </section>
  </section>
</main>
```

Hasta el momento así se ve nuestro frontend, el `` está vacío dado que no hemos seleccionado ningún país.

Pasemos a implementar lo necesario para consultar a la API con Javascript y completar el combo `<select>`

Países y banderas del mundo

Seleccionar país

Lo primero que tenemos que hacer en JS es guardar la URL de la API en una constante.

Luego podemos obtener los elementos `<select>` e `` del HTML con JS, que serán necesarios para cargar esas etiquetas con los datos que obtengamos de la API.

Posteriormente escucharemos el evento de carga del DOM HTML, para que cuando esté lista la página llamar a la función que se encargara de realizar la petición HTTP GET a la API y obtener los datos.

Una vez que obtengamos los datos, los enviaremos a una función para que cargue los datos en nuestro `<select>` del frontend HTML.

```
//Definimos una constante para guardar la URL de la API
const URL = "https://restcountries.com/v3.1/all?fields=name,flags";

//Seleccionamos la etiqueta <select> y la etiqueta <img>
let select = document.querySelector("#combo");
let bandera = document.querySelector("#bandera");

//Escuchamos el evento de carga del DOM HTML
document.addEventListener("DOMContentLoaded", iniciar);

//Definimos la función de inicio que llamara a la función
//para consultar a la API y obtener el listado de países
function iniciar(){
  |   consultarAPI();
}
}
```


Si analizamos 1 objeto con la app Postman en la respuesta vemos que, para cada objeto del arreglo, tenemos dos propiedades, ***flags y name***, que a su vez contienen otro objeto cada uno de ellos, de los cuales para nuestro ejemplo nos interesan las propiedades ***flags.png y name.common***

```
[
  {
    "flags": {
      "png": "https://flagcdn.com/w320/gs.png",
      "svg": "https://flagcdn.com/gs.svg",
      "alt": ""
    },
    "name": {
      "common": "South Georgia",
      "official": "South Georgia and the South Sandwich Islands",
      "nativeName": {
        "eng": {
          "official": "South Georgia and the South Sandwich Islands",
          "common": "South Georgia"
        }
      }
    }
  },
]
```

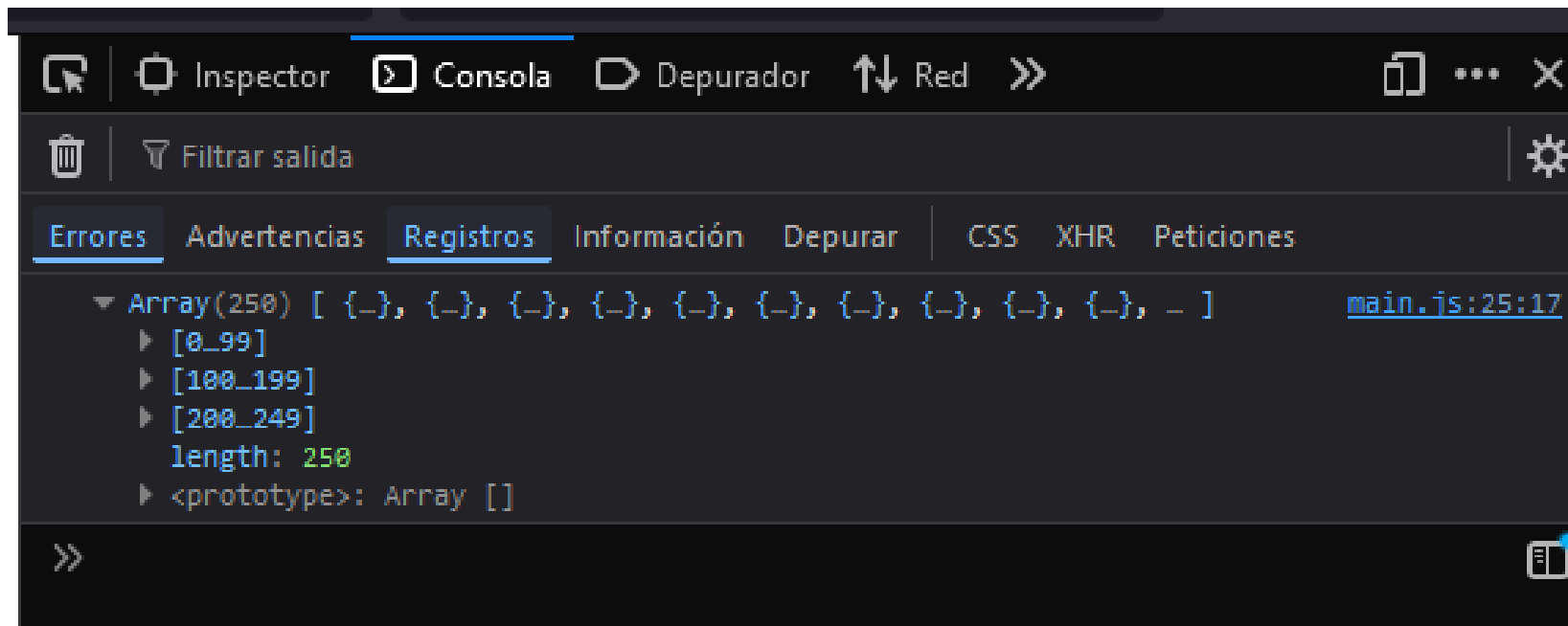
La función consultarAPI() quedaría definida de la siguiente manera:

```
//Implementamos la función para consultar a la API con fetchAPI
async function consultarAPI(){
  try{
    const respuesta = await fetch(URL);
    const datos = await respuesta.json();

    //Mostramos en consola los datos obtenidos de la API
    console.log(datos);

    //Procesamos los datos hacia el frontend
    frontend(datos);
  }catch(error){
    alert(error);
  }
}
```

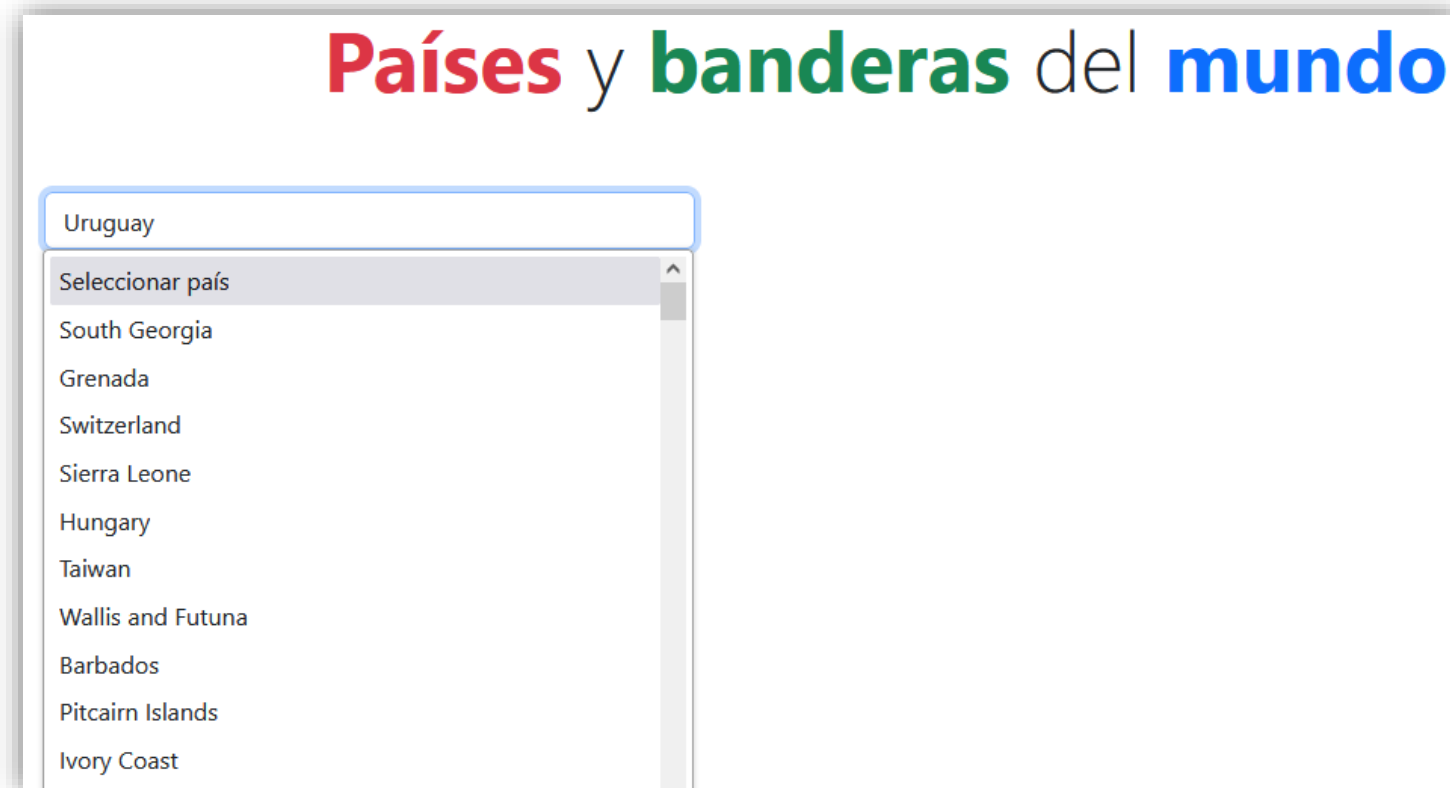
Realizado lo anterior, pasemos a ver nuestra página HTML e inspeccionar la consola del navegador. Al ver en la consola vemos que obtuvimos la respuesta de la API, la cual cuenta con un arreglo de 250 objetos. Ese arreglo es el que debemos iterar para cargar el elemento `<select>`.



A partir de las propiedades que nos interesan pasamos a completar la función que a su vez va a completar nuestro elemento `<select>`. En el `<option>` del `<select>` cargaremos la URL de las banderas, para posteriormente obtener ese dato y presentar la imagen en nuestra etiqueta ``, y en el medio de ambos `<option>` colocaremos el nombre del país, que es el que verá el usuario en nuestro frontend.

```
function frontend(arreglo){  
  //Iteramos el arreglo de objetos para cargar el select  
  for(let i=0; i<arreglo.length; i++){  
    select.innerHTML += `  
      <option value='${arreglo[i].flags.png}'>  
        ${arreglo[i].name.common}  
      </option>  
    `;  
  }  
}
```

Hasta el momento nuestro frontend ya tiene el `<select>` cargado con los países, pero no muestra las banderas. Lo que debemos hacer ahora es que al cambiar cada opción en el `<select>` se nos muestre la bandera correspondiente a cada país.



Para poder actualizar nuestro elemento `` cada vez que se cambie de país en el `<select>`, debemos escuchar el evento “***change***” para tomar una acción en concreto debido al cambio, la acción será actualizar el atributo “***src***” de la etiqueta ``, que es la que nos posibilita ver la bandera de cada país.

Recordemos que tenemos cargada la URL de cada bandera en formato .png en la propiedad “***value***” de cada ***<option>***, pues leyendo ese valor es que podremos ir actualizando nuestra etiqueta ``.

Pasemos a ello...

En nuestra función de inicio definimos que vamos a escuchar el evento “change” de la etiqueta <select> y le asociamos la función “***actualizarBandera()***” para que se ejecute con cada cambio del elemento.

```
//Definimos la función de inicio que llamara a la función
//para consultar a la API y obtener el listado de países
function iniciar(){
    consultarAPI();

    //Escuchar el evento CHANGE del <select>
    select.addEventListener("change", actualizarBandera);
}
```

La función para actualizar la etiqueta queda implementada de la siguiente manera:

```
function actualizarBandera(){
    //Obtenemos el valor del <select>
    let valor = select.value;

    //Si el valor no es "vacio"
    //actualizamos el atributo SRC de la etiqueta <img>
    if (valor !== "vacio"){
        bandera.setAttribute("src", valor);
    }else{
        bandera.setAttribute("src", "");
    }
}
```


Cuando cambiemos de país en el <select>, la etiqueta se actualizará automáticamente.

Países y banderas del mundo

Uruguay



Ejercicio 1

Teniendo en cuenta la implementación realizada anteriormente, ahora queremos crear un nuevo proyecto HTML/CSS/JS para mostrar los datos de 10 Pokemones.

Para ello recomendamos analizar la documentación de la API para ver cómo obtener los datos de 1 Pokemon en particular, para posteriormente analizar cómo obtener datos de 10 pokemones.

De los datos de cada Pokemon mostrar en el frontend nombre e imagen de frente.
Si lo desea puede analizar en el sitio de Bootstrap como generar una Card para presentar los datos.

Ejercicio 2

Utilizando la API pública ***<https://randomuser.me>***, generar 8 tarjetas de presentación con los datos de 8 personas aleatorias. De cada persona mostrar su nombre, apellido, dirección, país, ciudad, email, teléfono y foto.

Es importante ver la documentación de la API para saber cómo realizar una consulta HTTP y es recomendable utilizar Postman para hacer unas pruebas de verificación previas.

Si lo desea puede ver en el sitio de Bootstrap como generar Cards o una Tabla para presentar la información de los usuarios obtenidos.

FIN.