UT09.3: API Fetch

Contenido

1. Introducción 2
2. Objeto `Response` 3
   2.1. Propiedades 3
   2.2. Métodos 3
3. Métodos de la petición 5
   3.1. Método GET 5
   3.2. Método POST 6

1. Introducción

El API Fetch (https://developer.mozilla.org/es/docs/Web/API/Fetch_API) proporciona un interfaz que sustituye al objeto `XMLHTTPRequest` para la obtención de recursos, ofreciendo una serie de funcionalidades más avanza que el anterior interfaz.

El API habilita dos objetos, `Request` y `Response` que podemos utilizar en nuestra aplicación.

El API utiliza el método global `fetch()` el cual toma como argumento una URL y devuelve un objeto `Promise` para resolver con el objeto `Response` la petición. Como argumento opcional recibe un objeto literal para configurar la petición. Podemos encontrar más información sobre las propiedades para personalizar la petición en https://developer.mozilla.org/es/docs/Web/API/fetch.

```
// url (required), options (optional)
fetch(url, {
  method: 'get'
}).then(function(response) {
  ...
}).catch(function(err) {
  ...
});
```

Si no se especifica el argumento de opciones se realizará una petición de tipo GET.

La especificación `fetch` difiere de `jQuery.ajax()` en los siguientes aspectos:

- Utiliza un objeto `Promise` en lugar de una propiedad `status`, incluso si la respuesta tiene códigos HTTP que no sean 200.
- No enviará cookies a otros orígenes, a no ser que se utilicen credenciales como opción de configuración.

2. Objeto `Response`

Veamos el objeto que representa la respuesta a la petición.

2.1. Propiedades

Este objeto representa la respuesta a la petición realiza. Algunas de las propiedades que dispone este objeto son:

- `url`: ofrece la URL de la petición.
- `status`: código de respuesta HTTP.
- `statusText`: Mesaje de respuesta HTTP.
- `headers`: Cabeceras de la respuesta.

```
fetch('https://api.github.com/users/torvals', {
  method: 'get',
}).then((response) => {
  console.log(response.url);
  console.log(response.status);
  console.log(response.statusText);
  for (const header of response.headers.entries()) {
    console.log(`${header[0]}: ${header[1]}`);
  }
}).catch((err) => {
  console.log(err);
});
```

O lo podemos envolver en una función asíncrona.

```
async function f() {
  try {
    const response = await fetch('https://api.github.com/users/torvals');
    console.log(response.url);
    console.log(response.status);
    console.log(response.statusText);
    for (const header of response.headers.entries()) {
      console.log(`${header[0]}: ${header[1]}`);
    }
  } catch (error) {
    console.log(error);
  }
}
f();
```

2.2. Métodos

El objeto `Response` ofrece los datos de la respuesta empaquetados en un objeto `Promise`. El interfaz ofrece una serie de métodos que nos permitirá acceder

- `text()`: devuelve los datos en formato de texto.
- `json()`: devuelve los datos en formato JSON.
- `blob()`: devuelve los datos en formato binario.
- `formData()`: devuelve la respuesta como un objeto `FormData`.

En el siguiente ejemplo, el método `fetch()` devuelve el objeto `Promise` para que se procesado a través del método `then()`, empaquetando los datos en formato de `String` para procesar la respuesta en la siguiente promesa.

```
fetch('https://api.github.com/users/torvals', {
  method: 'get',
})
  .then((response) => response.text())
  .then((data) => console.log(data))
  .catch((err) => {
    console.log(err);
  });
```

O podemos transformar la respuesta en un JSON y procesar el objeto literal recuperado.

```
fetch('https://api.github.com/users/torvals', {
  method: 'get',
})
  .then((response) => response.json())
  .then((data) => {
    console.log(data.login);
    console.log(data.id);
  })
  .catch((err) => {
    console.log(err);
  });
```

Por último, recuperamos un binario y generamos una imagen con él.

```
fetch('img/user.jpg').then((response) => response.blob()).then((image) =>
{
  const myImage = new Image();
  const imageURL = URL.createObjectURL(image);
  myImage.src = imageURL;
  document.body.append(myImage);
});
```

Si queremos trabajar con una función asíncrona necesitamos una promesa para la petición, y una segunda promesa para procesar la respuesta, por lo que lo hacemos en dos sentencias `await`.

```
async function f() {
  try {
    const response = await fetch('https://api.github.com/users/torvals');
    const data = await response.json();
    console.log(data.login);
    console.log(data.id);
  } catch (error) {
    console.log(error);
  }
}
f();
```

O para el ejemplo de la imagen.

```
async function f() {
  try {
    const response = await fetch('img/user.jpg');
    const image = await response.blob();
    const myImage = new Image();
    const imageURL = URL.createObjectURL(image);
    myImage.src = imageURL;
    document.body.append(myImage);
  } catch (error) {
    console.log(error);
  }
}
f();
```

3. Métodos de la petición

Veamos cómo podemos utilizar los métodos GET y POST con el API Fetch.

3.1. Método GET

Vamos a implementar el mismo ejemplo de los usuarios aleatorios, pero utilizando el API Fetch. El servicio sabemos que está implementado para atender peticiones GET.

Utilizamos un objeto `URL` que utilizaremos para realizar la petición al servicio.

```
let url = new URL('https://randomuser.me/api/');
//let params = {results: 4, gender: 'female'}
let params = [['results', 4], ['gender', 'female']]
url.search = new URLSearchParams(params).toString();
```

Una vez construida la URL invocamos el método `fetch()` pasándola como argumento.

```
const url = new URL('https://randomuser.me/api/');
// let params = {results: 4, gender: 'female'}
const params = [['results', 4], ['gender', 'female']];
url.search = new URLSearchParams(params).toString();

fetch(url, {
  method: 'get',
}).then((response) => response.json()).then((data) => {
  data.results.forEach((user) => {
    console.log('##############');
    console.log(`${user.name.title} ${user.name.first}
${user.name.last}`);
    console.log(user.phone);
    console.log(user.email);
    console.log(user.picture.large);
  });
});
```

3.2. Método POST

Utilizaremos este método cuando el peso de nuestros parámetros sea demasiado grande para utilizar la URL, ya que en este método la información se almacena en el cuerpo de la petición. También podremos enviar ficheros.

Vamos a crear una petición que envíe los mismos parámetros que en el apartado anterior, pero además añadiremos un parámetro con un objeto JSON, un objeto de tipo `Blob` y un fichero, para el cual crearemos un pequeño formulario para lanzar la petición. En este caso el servicio será un script PHP propio, diseñado para devolver en un objeto JSON la información con los parámetros que recibe. Creamos el código HTML para recoger el fichero.

```
window.addEventListener('load', (event) => {
  const main = document.querySelector('main');
  main.replaceChildren();
  const container = document.createElement('div');
  container.classList.add('container');
  container.insertAdjacentHTML('afterbegin', `<div class="row"><div
class="form-group">
  <div class="custom-file">
    <input type="file" id="vfPostFile" name="vfPostFile">
  </div>
  <button class="btn btn-primary" type="button"
id="bPost">Enviar</button>
</div></div>`);
  main.append(container);
});
```

Asignamos un manejador al evento `click` del botón.

```
document.getElementById('bPost').addEventListener('click', (event)=> {
});
```

En el cuerpo del manejador invocaremos al método `fetch()`. Primero construimos la URL sobre el servicio **httpbin.org**.

```
const vfPostFile = document.getElementById('vfPostFile');
const url = new URL('https://httpbin.org/post');
```

Creamos el objeto `FormData` y añadimos los parámetros con el método `append()`.

```
let formData = new FormData();
formData.append('results', '8');
formData.append('gender', 'female');
formData.append('webmasterfile', vfPostFile.files[0]);
let product = {
  id: 123,
  name: 'PC',
  brand: 'HP',
  model: 'EliteBook'
}
let blob = new Blob([JSON.stringify(product)], { type: "text/xml"});
formData.append("blobField", blob);
formData.append("product", JSON.stringify(product));
```

Para parametrizar la petición, tenemos que indicar que la petición es de tipo POST, y en la propiedad `body` asignaremos el objeto `FormData` para que sean añadidos al cuerpo de la petición.

```
fetch(url, {
  method: 'post',
  body: formData
}).then(function(response) {
  return response.json();
}).then(function(data) {
  console.dir(data);
}).catch(function(err) {
  console.log('No se ha recibido respuesta.');
});
```