ThePower business school

Fetch JS☎🤖

`fetch` es una función de JavaScript que se utiliza para realizar solicitudes HTTP. Es una forma moderna y más sencilla de realizar solicitudes HTTP en comparación con la función `XMLHttpRequest` tradicional.

La sintaxis básica de `fetch` es la siguiente:

```
fetch(url, options)
  .then((response) => {
    // procesar la respuesta del servidor
  })
  .catch((error) => {
    // manejar cualquier error
  });
```

`fetch` toma dos argumentos:

- `url`: es la URL de la solicitud HTTP.
- `options`: es un objeto opcional que puede contener información adicional sobre la solicitud, como el método HTTP (por ejemplo, `GET`, `POST`, `PUT`, etc.), los encabezados y el cuerpo de la solicitud.

`fetch` devuelve una promesa que se cumple con un objeto `Response` que representa la respuesta del servidor. Luego, se puede utilizar el método `then` para procesar la respuesta y el método `catch` para manejar cualquier error.

Aquí hay un ejemplo de cómo se puede utilizar `fetch` para realizar una solicitud HTTP `GET`:

```
fetch('https://example.com/api/endpoint')
  .then((response) => {
```

```
    // procesar la respuesta del servidor
  })
  .catch((error) => {
    // manejar cualquier error
  });
```

Aquí hay un ejemplo de cómo se puede utilizar `fetch` para realizar una solicitud HTTP `POST` con un cuerpo de solicitud:

```
const data = { username: 'john', password: 'doe' };

fetch('https://example.com/api/login', {
  method: 'POST',
  body: JSON.stringify(data),
  headers: {
    'Content-Type': 'application/json'
  }
})
  .then((response) => {
    // procesar la respuesta del servidor
  })
  .catch((error) => {
    // manejar cualquier error
  });
```

RESTful verbs Fetch

Petición GET: Este tipo de petición (hay otros tipos de peticiones como PUT, POST, DELETE…) se utiliza fundamentalmente para obtener datos de la fuente especificada (habitualmente un servicio web).

No obstante, si tuviésemos que enviar información delicada, por ejemplo para hacer un login, usaríamos POST en lugar de GET.

```
fetch('https://swapi.co/api/people/1/')
.then(res => res.json())
.then(res => console.log(res));
.catch( err => console.error(err));
```

1. Hemos llamado a fetch() con la URL a la que queremos acceder como parámetro. Esta llamada nos devuelve una promesa.
2. El método *then()* de esa promesa nos entrega un objeto response.A partir de este objeto llamamos al método *json()* para obtener el cuerpo de la respuesta. Este método *json()* nos devuelve otra promesa que se resolverá cuando se haya obtenido el contenido.
3. El método *then()* de esta segunda promesa recibe el cuerpo devuelto por la promesa anterior y hace un log de ella.
4. Hemos incluido un catch() por si se produce algún error

Petición POST: La utilizaremos para insertar nuevos registros.

```
fetch('http://localhost:3009/users', {
    method: 'POST',
    headers: {
        "Content-Type": "application/json",
    },
    body: JSON.stringify({ mail: 'pp@pp.com', password: '123' })
})
.then(res => res.json())
.then(res=> {
      console.log(res);
});
```

Petición PUT: La utilizaremos para actualizar información existente. Enviaremos la información que queremos actualizar en el cuerpo de la petición y la id del elemento que queremos actualizar en la url de la petición.

```
fetch('http://localhost:3005/users/1', {
        method: 'PUT',
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify({ mail: 'ñññññ', password: 'ñññññ' })
 })
```

```
.then(res => res.json())
.then(res=> {
      console.log(res);
});
```

Petición Delete: La utilizaremos para eliminar información existente.

```
fetch('http://localhost:3005/users/2', {
      method: 'DELETE',
})
.then(res => res.json())
.then(res=> {
      console.log(res);
});
```

Fetch - Pokeapi

Aquí hay un ejemplo de cómo se puede utilizar `fetch` para realizar una solicitud HTTP `GET` a la PokeAPI para obtener información sobre un Pokémon específico:

```
const pokemonName = 'pikachu';

fetch(`https://pokeapi.co/api/v2/pokemon/${pokemonName}`)
  .then((response) => {
    // la respuesta del servidor es una promesa, así que tenemos que devolver otra promesa
    // para poder procesar el cuerpo de la respuesta
    return response.json();
  })
  .then((data) => {
    console.log(data); // imprime información sobre el Pokémon Pikachu
  })
  .catch((error) => {
    console.error(error);
  });
```

En este ejemplo, primero hacemos una solicitud HTTP `GET` a la URL de la PokeAPI para el Pokémon específico. Luego, utilizamos el método `json` del objeto `Response` para procesar el cuerpo de la respuesta y obtener los datos en formato JSON. Finalmente,

utilizamos el método `then` para acceder a los datos y el método `catch` para manejar cualquier error que pueda ocurrir durante la solicitud.