Método PUT

Introducción

En la anterior sección vimos como guardar datos en el servidor con el método `POST`, en esta sección veremos como actualizarlos. Para ello usaremos el método `PUT`.

¿Qué es el método PUT?

El método `PUT` es un método de la especificación HTTP que se usa para actualizar datos en el servidor.

En la siguiente sección usaremos el método `PATCH`, pero se diferencia del método `PUT` en que el método `PATCH` se usa para actualizar una parte de los datos, mientras que el método `PUT` se usa para actualizar todos los datos.

Veremos dos formas de hacerlo, la primera usaremos la librería de terceros `axios` y la segunda usaremos la librería nativa de JavaScript `fetch`.

Axios

Esta librería la usamos en los videos anteriores (GET, POST) . Axios nos permite actualizar datos en el servidor de una forma muy sencilla, para ello usaremos el método `put` de la librería.

Cómo instalar Axios

Para instalar *axios* tenemos que ejecutar el siguiente comando:

```
npm install axios --save
```

Cómo usar Axios

- Lo primero que tenemos que hacer es importar axios en nuestro proyecto.

```
import axios from "axios";
```

- Una vez importado, podemos usar el método `put` de la librería para actualizar datos en el servidor.

Partimos del ejemplo de la sección anterior, donde creamos una entrada nueva en el servidor. Y ahora queremos actualizarla, vamos a cambiar la descripción y añadir una imagen nueva.

```

```

```
const peliculaActualizada = {
  id: "28",
  title: "Spider-Man",
  year: 2002,
  director: "Sam Raimi",
-  description:
-    "Shy student Peter Parker acquires special abilities after being bitten by a genetically modified spider. Under the identity of Spider-Man, he fights crime in New York while dealing with his personal problems.",
  actors: ["Tobey Maguire", "Willem Dafoe", "Kirsten Dunst"],
-  coverUrl: "https://example.com/spiderman.jpg",
+  description:
+    "Peter Parker, a shy high school student, is often bullied by people. His life changes when he is bitten by a genetically altered spider and gains superpowers. He uses his powers to save the city from evil forces.",
+  coverUrl: "https://example.com/spiderman2.jpg",
};
```

Con el método `PUT` vamos a sustituir la entrada que creamos en la sección anterior por la que acabamos de crear.

Creamos una función `actualizarPelicula`, que en este caso va a tener:

- El método *put* de la librería `axios`.
- La URL de la API.
- El objeto que queremos actualizar.
- Un parámetro `id` que es el identificador de la entrada que queremos actualizar.
- Esto es una promesa, por lo que tenemos que usar *then* para obtener la respuesta de la API y *catch* para capturar los errores.

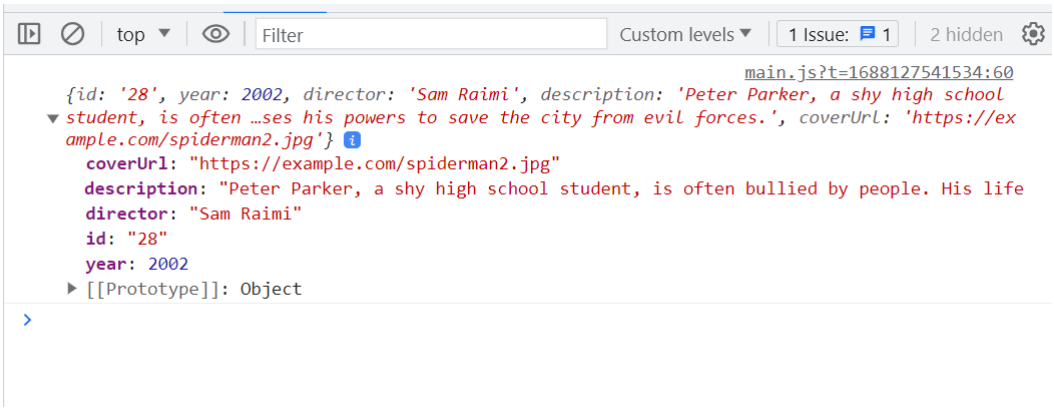
```
const actualizaPelicula = (pelicula, id) => {
  axios
    .put(`http://localhost:3000/movies/${id}`, pelicula)
    .then((response) => {
      console.log(response.data);
    })
    .catch((error) => {
      console.log(error);
    });
};
```

> Ojo aquí hemos harcodeado la URL completa (*http://localhost:3000/movies*), pero lo ideal es que la URL base (*http://localhost:3000*) la guardemos en una variable y la usemos en el resto del código, de esta manera si cambia la URL de la API, solo tenemos que cambiarla en un sitio (suele pasar cuando está desarrollando que tiras de *localhost* y cuando llevas a producción tienes que cambiar la URL).

- Por último, llamamos a la función `actualizarPelicula` y le pasamos como parámetros el objeto que queremos actualizar y el identificador de la entrada que queremos actualizar.

```
actualizaPelicula(peliculaActualizada, "28");
```

Al ejecutar el código, podemos ver que en la consola nos aparece la respuesta de la API, en este caso nos devuelve un objeto con la información de la entrada que acabamos de actualizar.



Cómo vemos en la imagen, en la respuesta nos aparece un objeto con la información de la entrada que acabamos de actualizar.

Axios con async/await

También podemos usar `async/await` para actualizar datos en el servidor, para ello tenemos que hacer lo siguiente:

```
const actualizaPelicula = async (pelicula, id) => {
  try {
    const response = await axios.put(
      `http://localhost:3000/movies/${id}`,
      pelicula
    );
    console.log(response.data);
  } catch (error) {
    console.log(error);
  }
};
```

Fetch

También podemos actualizar datos en el servidor usando la librería nativa de JavaScript `fetch`.

Partimos del ejemplo anterior, donde creamos una entrada nueva en el servidor. Y ahora queremos actualizarla, vamos a cambiar la descripción y añadir una imagen nueva.

```
const peliculaActualizada = {
  id: "26",
```

```
  year: 2002,
  director: "Sam Raimi",
  description:
    "Peter Parker, a shy high school student, is often bullied by people. His life
changes when he is bitten by a genetically altered spider and gains superpowers.
He uses his powers to save the city from evil forces.",
  actors: ["Tobey Maguire", "Willem Dafoe", "Kirsten Dunst"],
  coverUrl: "https://example.com/spiderman2.jpg",
};
```

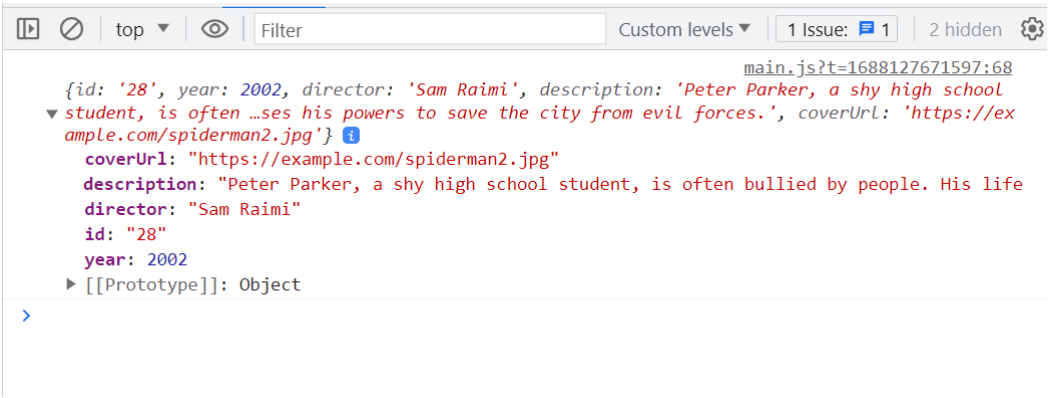
- Creamos una función *actualizarPelicula*, que va a tener:
  - El método *put* de la librería `fetch`.
  - La URL de la API.
  - El objeto que queremos actualizar.
  - Un parámetro `id` que es el identificador de la entrada que queremos actualizar.
  - En el body usamos el método *JSON.stringify()* para serializar el objeto con la información de la película.
  - En los headers le pasamos el *Content-Type* para indicar que el contenido que vamos a enviar es de tipo JSON.
  - Esto es una promesa, por lo que tenemos que usar *then* para obtener la respuesta de la API y *catch* para capturar los errores.

```
const actualizaPelicula = (pelicula, id) => {
  fetch(`http://localhost:3000/movies/${id}`, {
    method: "PUT",
    body: JSON.stringify(pelicula),
    headers: {
      "Content-Type": "application/json",
    },
  })
    .then((response) => {
      return response.json();
    })
    .then((data) => {
      console.log(data);
    })
    .catch((error) => {
      console.log(error);
    });
};
```

- Por último, llamamos a la función `actualizarPelicula` y le pasamos como parámetros el objeto que queremos actualizar y el identificador de la entrada que queremos actualizar.

```
actualizarPelicula(peliculaActualizada, "28");
```

Al ejecutar el código, podemos ver que en la consola nos aparece la respuesta de la API, en este caso nos devuelve un objeto con la información de la entrada que acabamos de actualizar y podemos ver *statusText* nos aparece *OK*, lo que significa que se ha actualizado correctamente.



Fetch con async/await

También podemos usar `async/await` para actualizar datos en el servidor, para ello tenemos que hacer lo siguiente:

```
const actualizaPelicula = async (pelicula, id) => {
  try {
    const response = await fetch(`http://localhost:3000/movies/${id}`, {
      method: "PUT",
      body: JSON.stringify(pelicula),
      headers: {
        "Content-Type": "application/json",
      },
    });
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.log(error);
  }
};
```

Te toca

En este ejercicio vamos a crear una función que nos permita actualizar un actor en nuestro servidor.

Para ello vamos a seguir los siguientes pasos:

- Creamos una función `actualizaActor` que va a tener como parámetros:
  - Un objeto con la información del actor que queremos actualizar.
  - La *id* del actor que queremos actualizar.
  - Llamamos a la función `actualizaActor`.

Esta sería la firma de la función:

```
const actualizaActor = (actor, id) => {};
```

Pistas:

- La URL de la API es *http://localhost:3000/actors*.
- El método que tenemos que usar es *PUT*.
- El objeto que tenemos que pasarle a la API tiene que tener la siguiente estructura:

```
const actorActualizado = {
  name: "Tom Holland",
  movies: [
    "Captain America: Civil War",
    "Spider-Man: Homecoming",
    "Avengers: Infinity War",
    "Spider-Man: Far From Home",
    "Spider-Man: No Way Home",
  ],
-  bio: "Thomas Stanley Holland is an English actor. A graduate of the BRIT School
in London, he began his acting career on stage in the title role of Billy Elliot
the Musical in London's West End from 2008 to 2010.",
-  image: "https://example.com/tom-holland.jpg",
+ bio: "Thomas Stanley Holland is an English actor. A graduate of the BRIT School
in London, he began his acting career on stage in the title role of Billy Elliot
the Musical in London's West End from 2008 to 2010. He gained further recognition
for his starring role in the disaster film The Impossible (2012), receiving the
London Film Critics Circle Award for Young British Performer of the Year.",
+ image: "https://example.com/tom-holland2.jpg",
};
```

- Para actualizar un actor, tenemos que pasarle a la API el objeto con la información del actor que queremos actualizar y el identificador del actor que queremos actualizar.
- Invocamos a la función `actualizarActor` y le pasamos como parámetros el objeto con la información del actor que queremos actualizar y la *id* del actor que queremos actualizar.

Vamos a pausar el video y lo intentamos. Si no has dado con la solución, no desesperes, la vemos a continuación.

Esta sería la solución al ejercicio:

Solución usando axios:

```
import axios from "axios";

const actualizaActor = (actor, id) => {
  axios
    .put(`http://localhost:3000/actors/${id}`, actualizarActor)
    .then((response) => {
      console.log(response.data);
```

```
    })
    .catch((error) => {
      console.log(error);
    });
};
```

Solución usando fetch:

```
const actualizaActor = (actor, id) => {
  fetch(`http://localhost:3000/actors/${id}`, {
    method: "PUT",
    body: JSON.stringify(actor),
    headers: {
      "Content-Type": "application/json",
    },
  })
    .then((response) => {
      return response.json();
    })
    .then((data) => {
      console.log(data);
    })
    .catch((error) => {
      console.log(error);
    });
};
```

Control de errores

En el caso de que el servidor nos devuelva un error, podemos controlarlo de la siguiente manera:

```
const actualizaActor = (actor, id) => {
  fetch(`http://localhost:3000/actors/${id}`, {
    method: "PUT",
    body: JSON.stringify(actualizarActor),
    headers: {
      "Content-Type": "application/json",
    },
  })
    .then((response) => {
      return response.json();
    })
    .then((data) => {
      console.log(data);
    })
    .catch((error) => {
      console.log(error);
    });
};
```

En este caso, si la respuesta del servidor es correcta, devolvemos los datos en formato JSON, en caso contrario lanzamos un error.

Si simulamos el error, para ellos cambiamos la URl de la Api, en vez de *http://localhost:3000/actores* ponemos *http://localhost:3000/noexiste*.

```
const actualizaActor = (actor, id) => {
-  fetch(`http://localhost:3000/actors/${id}`, {
+  fetch(`http://localhost:3000/noexiste/${id}`, {
    method: "PUT",
    body: JSON.stringify(actualizarActor),
    headers: {
      "Content-Type": "application/json",
    },
  })
    .then((response) => {
+      if (response.ok) {
        return response.json();
+      } else {
+        throw new Error("Error en la llamada a la API");
+      }
    })
    .then((data) => {
      console.log(data);
    })
    .catch((error) => {
      console.log(error);
    });
};
```

Al ejecutar el código, podemos ver que en la consola nos aparece el error que hemos lanzado.

```
try {
  actualizaActor(actorActualizado, "1");
} catch (error) {
  console.log(error);
}
```