20 - 10 - 2023

Axios y Fetch.

Alumno:
Edgar Hernandez Rodríguez.

Grupo:
ISW 7A

Materia:
Programación Concurrente.

Maestro:
Ramon Morales.

Unidad 2.

Peticiones HTTP

Las peticiones HTTP (Hypertext Transfer Protocol) en aplicaciones web se refieren a solicitudes que un cliente web (como un navegador) realiza a un servidor web para obtener información o realizar una acción. Las peticiones HTTP son la base de la comunicación en la World Wide Web y son utilizadas para transmitir datos entre el cliente y el servidor. Las peticiones HTTP en aplicaciones web son:

Métodos HTTP: Las peticiones HTTP se realizan utilizando métodos HTTP estándar, los más comunes son:

- GET: Se utiliza para solicitar recursos, como páginas web o datos, del servidor. No suele tener un cuerpo (payload) y se utiliza para recuperar información.
- POST: Se utiliza para enviar datos al servidor, por ejemplo, cuando se envían datos de formularios o se realiza una acción que modifica datos en el servidor.
- PUT: Se utiliza para actualizar un recurso existente en el servidor.
- DELETE: Se utiliza para eliminar un recurso en el servidor.

URL y Recursos: En una solicitud HTTP, se especifica una URL (Uniform Resource Locator) que apunta a un recurso en el servidor. Por ejemplo, una URL podría ser https://www.ejemplo.com/api/datos para solicitar datos de una API web.

Cabeceras HTTP: Las peticiones HTTP también pueden incluir cabeceras HTTP que proporcionan información adicional sobre la solicitud, como el tipo de contenido aceptado, la información de autenticación, el tipo de compresión, etc.

Cuerpo de la Solicitud: Algunos métodos, como POST y PUT, pueden incluir un cuerpo en la solicitud que contiene los datos que se envían al servidor. Estos datos pueden estar en diferentes formatos, como JSON, XML o formularios HTML.

Respuesta del Servidor: Cuando el servidor recibe una solicitud HTTP, procesa la solicitud y devuelve una respuesta HTTP que incluye un código de estado y opcionalmente datos en el cuerpo de la respuesta. El código de estado indica si la solicitud se ha completado con éxito (por ejemplo, 200 OK) o si ha habido un error (por ejemplo, 404 Not Found o 500 Internal Server Error).

Axios y Fetch

Axios y Fetch son dos formas populares de realizar peticiones HTTP en aplicaciones web, particularmente en el contexto de JavaScript y el desarrollo web. Ambas se utilizan para hacer solicitudes a servidores web y obtener datos, que pueden ser HTML, JSON, XML u otros tipos de datos.

Axios.

- Axios es una biblioteca JavaScript que se utiliza para realizar solicitudes HTTP desde el navegador o desde Nodejs. Es una biblioteca de terceros que simplifica la realización de peticiones HTTP y el manejo de respuestas.
- Proporciona una API simple y fácil de usar para realizar solicitudes GET, POST, PUT, DELETE, y otras solicitudes HTTP.
- Puede manejar automáticamente las respuestas en formato JSON y convertirlas en objetos JavaScript.
- Ofrece un manejo más avanzado de errores y capacidades de intercepción de solicitudes y respuestas.
- Es ampliamente utilizado en aplicaciones web modernas y es especialmente popular en el desarrollo de aplicaciones front-end con frameworks como React y Vue.

Ejemplo de uso de Axios en JavaScript:

```
const axios = require('axios');

axios.get('https://api.example.com/data')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error(error);
  });
```

Fetch.

- Fetch es una API nativa de JavaScript para realizar solicitudes HTTP. Está disponible en la mayoría de los navegadores modernos, lo que significa que no es necesario agregar ninguna biblioteca externa.
- Fetch proporciona una interfaz simple y promesas para realizar solicitudes y manejar respuestas, lo que la hace fácil de usar.
- A diferencia de Axios, Fetch no realiza automáticamente la conversión de datos en objetos JavaScript, por lo que debes manejar manualmente la conversión de la respuesta a JSON u otros formatos.
- Fetch es ampliamente utilizado en aplicaciones web modernas debido a su disponibilidad nativa y su simplicidad.

Ejemplo de uso de Fetch en JavaScript.

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => {
    console.log(data);
  })
  .catch(error => {
    console.error(error);
  });
```

Ambas Axios y Fetch son útiles para realizar solicitudes HTTP en aplicaciones web, y la elección entre ellas depende de tus necesidades y preferencias. Axios proporciona más características y un manejo más avanzado de errores, mientras que Fetch es nativo en la mayoría de los navegadores y es una opción ligera y simple para muchas aplicaciones.

Async/await.

Async/await es una característica de JavaScript que simplifica el manejo de código asíncrono, como las llamadas a API, la lectura/escritura de archivos, o cualquier operación que tome tiempo para completarse. En lugar de utilizar promesas encadenadas o callbacks, puedes usar async/await para escribir código más legible y estructurado.

Definir una función asincrónica (async function): Para utilizar async/await, primero debes definir una función con la palabra clave async. Esto indica que la función contendrá código asincrónico que será manejado con await.

```
async function miFuncionAsincronica() {
  // Código asincrónico aquí
}
```

Utilizar await: Dentro de una función asincrónica, puedes usar la palabra clave await antes de una expresión que devuelva una promesa. El await pausará la ejecución de la función hasta que la promesa se resuelva o se rechace.

Ejemplo de una llamada a una API utilizando async/await:

```
async function obtenerDatosDeAPI() {
  try {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error(error);
  }
}
```

Manejar errores: Puedes usar un bloque try…catch para manejar errores en las operaciones asincrónicas. Cualquier error que se produzca dentro del bloque try se capturará en el bloque catch.

Llamar a la función asincrónica: Finalmente, para ejecutar la función asincrónica, simplemente llámala como lo harías con cualquier otra función.

```
obtenerDatosDeAPI();
```

Async/await es especialmente útil cuando se trabaja con promesas y permite escribir código más limpio y legible en comparación con anidamiento excesivo de promesas o el uso de callbacks. Ten en cuenta que las funciones marcadas como async siempre devuelven promesas, lo que significa que puedes usar await para esperar a que se complete la función y obtener su valor resuelto.

Es importante destacar que el soporte para async/await está disponible en versiones modernas de JavaScript y en navegadores. Asegúrate de que tu entorno de desarrollo sea compatible con estas características.