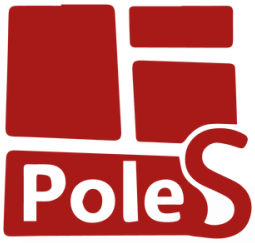

Requêtes POST et en-têtes HTTP

Le PoleS
2024

Focus sur les requêtes POST

Contrairement aux requêtes GET, les requêtes POST permettent d'envoyer des données au serveur pour effectuer une action, comme :

- Sauvegarder des informations dans une base de données.
- Créer un nouvel utilisateur.
- Envoyer un fichier.

1.Syntaxe de base pour une requête POST

Syntaxe Fetch API avec POST

Pour effectuer une requête POST, il faut configurer la méthode et souvent ajouter des données dans le corps de la requête.

2.Envoyer des données avec Fetch

a) Envoyer des données simples (texte ou clés-valeurs)

- On peut envoyer une chaîne de caractères, comme name=Thierry&email=thierry@example.com.
- Cette méthode est similaire à un formulaire HTML standard.

Pourquoi configurer les en-têtes ?

- Les en-têtes indiquent au serveur comment interpréter les données.
- Dans cet exemple, Content-Type: application/x-www-form-urlencoded signifie que les données sont envoyées sous forme de chaîne encodée comme dans un formulaire HTML.

b) Envoyer des données complexes : objets JSON

- Les objets JavaScript doivent être convertis en chaîne JSON avec JSON.stringify.
- Nécessite d'ajouter l'en-tête Content-Type: application/json.

```
fetch("server.php", {
    method: "POST",
    headers: {
        "Content-Type": "application/json", // Indique que le corps contient du JSON
    },
    body: JSON.stringify({ name: "Thierry", email: "thierry@email.com" }), // Conversion en
    JSON
});
```

Pourquoi JSON ?

- JSON est le format le plus utilisé pour l'échange de données entre client et serveur.
- Il est compact, lisible et facile à manipuler.

3.Envoyer des fichiers ou données mixtes : FormData

- FormData est une interface permettant de créer des paires clé-valeur représentant des données (texte, fichiers, etc.).
- Utilisé pour les formulaires complexes contenant des fichiers ou des données diverses.

```
const formData = new FormData();
formData.append("name", "Thierry");
formData.append("profilePhoto", document.getElementById("file").files[0]);

fetch("upload.php", {
    method: "POST",
    body: formData,
});
```

Pourquoi FormData ?

- Automatiquement géré par Fetch API sans besoin de configurer les en-têtes.
- Idéal pour l'envoi de fichiers (images, documents).

4.Gestion des en-têtes

Qu'est-ce qu'un en-tête HTTP ?

Un en-tête HTTP transmet des informations supplémentaires avec une requête ou une réponse.

Exemple :

- Content-Type : Indique le type de données envoyées (JSON, texte brut, etc.).
- Authorization : Utilisé pour l'authentification.

Pourquoi configurer des en-têtes dans Fetch API ?

- Certains types de données (comme JSON) nécessitent un en-tête pour que le serveur puisse les interpréter correctement.
- Renforce la sécurité et permet au serveur de refuser des requêtes mal formatées.

5.Gestion asynchrone avec async/await

a) Pourquoi async/await ?

- Les Promises rendent le code plus lisible, mais peuvent devenir complexes avec plusieurs .then().
- async/await simplifie la gestion des requêtes asynchrones.

b) Syntaxe avec async/await :

```
async function sendData() {
    try {
        const response = await fetch("server.php", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({ name: "Thierry", email: "thierry@email.com" }),
        });

        const result = await response.text(); // Lire la réponse en texte
        console.log(result);
    } catch (error) {
        console.error("Erreur :", error);
    }
}

sendData();
```

Conclusion : Ce que vous devez retenir

1. Les requêtes POST sont utilisées pour envoyer des données au serveur.
2. Fetch API est une solution moderne et flexible pour effectuer des requêtes HTTP.
3. Vous devez choisir le bon format (texte brut, JSON, FormData) en fonction du type de données à envoyer.
4. Configurer les en-têtes est essentiel pour garantir que le serveur interprète correctement les données.
5. Utilisez async/await pour un code asynchrone plus lisible.