Requêtes HTTP en JS : Le Cœur de la Communication Web

Communiquer avec les Serveurs

Pour les étudiants de 1ère année Supinfo

Au Programme

1. HTTP - Le Protocole de Communication Web
2. JSON - Format de Données Universel
3. Fetch API - Effectuer des Requêtes en JavaScript

HTTP : Introduction et Client-Serveur

QU'EST-CE QUE HTTP ?

HTTP (Hypertext Transfer Protocol) est le **protocole de communication client-serveur** qui est le fondement du transfert de données sur le Web. Il permet aux navigateurs Web de récupérer des informations des serveurs Web.

MODÈLE CLIENT-SERVEUR

- **Client** - Généralement un navigateur Web qui **envoie une requête** au serveur
- **Serveur** - Héberge les données du site Web et **répond à la requête**
- Le client peut aussi être un autre serveur (ex: API calls entre services)

STATELESS ET COOKIES

HTTP est un protocole sans état (stateless), ce qui signifie que chaque requête est **indépendante**. Le serveur ne garde pas de trace des requêtes précédentes du client. Pour pallier ce manque de mémoire, les **cookies** sont utilisés pour stocker des informations sur le client (ex: authentification).

HTTP : Les Verbes (Méthodes)

GET

Récupère des données d'une ressource spécifiée. Ne doit pas affecter l'état de la ressource (idempotente).

Cas d'usage : Charger une page web, récupérer des données d'une API

POST

Envoie des données à un serveur pour créer ou mettre à jour une ressource. Les données sont incluses dans le corps de la requête.

Cas d'usage : Envoyer des identifiants pour se connecter, créer un nouvel utilisateur

PUT

Envoie des données pour créer une nouvelle ressource ou remplacer une représentation existante de la ressource ciblée.

Cas d'usage : Remplacer complètement une ressource existante

PATCH

Applique des modifications partielles à une ressource. Contrairement à PUT, PATCH est utilisée pour faire des mises à jour partielles.

Cas d'usage : Modifier son profil, mettre à jour certains champs seulement

DELETE

Supprime la ressource spécifiée sur le serveur.

Cas d'usage : Effacer son compte, supprimer un article

HTTP : CRUD et Correspondance

CRUD

Create

Créer une nouvelle ressource sur le serveur

Read

Lire ou récupérer une ressource existante

Update

Mettre à jour une ressource existante

Delete

Supprimer une ressource existante

Verbes HTTP

POST

Envoyer des données pour créer une nouvelle ressource

GET

Récupérer les données d'une ressource spécifiée

PUT / PATCH

Remplacer ou modifier une ressource existante

DELETE

Supprimer la ressource spécifiée

💡 **Utilité pour les APIs**

CRUD est un acronyme très utile pour décrire les fonctionnalités d'une API. Pour chaque ressource (ex: un utilisateur), on doit généralement pouvoir lire, créer, modifier et effacer. Tous les utilisateurs ne doivent pas avoir ces droits (par exemple, seuls les administrateurs peuvent avoir la liste des utilisateurs), mais l'API devrait pouvoir les gérer.

HTTP : Requête et Réponse

Structure de la Requête

Une requête HTTP est initiée par le client et comprend généralement les éléments suivants :

- **Méthode**
  Indique l'action que le client veut effectuer (GET, POST, PUT, DELETE, etc.)
- **URL**
  L'adresse de la ressource sur le serveur (Uniform Resource Locator)
- **Version HTTP**
  La version du protocole utilisée (ex: HTTP/1.1, HTTP/2)
- **En-têtes (Headers)**
  Fournissent des informations supplémentaires (Content-Type, Authorization, etc.)
- **Corps (Body)**
  Les données à envoyer au serveur (utilisé avec POST, PUT, PATCH)

Structure de la Réponse

Après avoir traité la requête, le serveur envoie une réponse qui comprend :

- **Statut**
  Code de statut (200, 404, 500) et message de statut (OK, Not Found, etc.)
- **Version HTTP**
  La version du protocole utilisée dans la réponse
- **En-têtes (Headers)**
  Fournissent des informations supplémentaires (Content-Type, Set-Cookie, etc.)
- **Corps (Body)**
  Les données réelles demandées (HTML, JSON, images, etc.)

HTTP : Codes de Statut

SUCCÈS ET REDIRECTION

200 OK

La requête a réussi. Le serveur a traité la requête avec succès et retourne les données demandées.

301 Moved Permanently

L'URI de la ressource demandée a été modifiée de manière permanente. Les futurs liens devraient utiliser la nouvelle URL.

304 Not Modified

La ressource n'a pas été modifiée depuis la dernière demande du client. Utilisé pour la mise en cache.

ERREURS CLIENT ET SERVEUR

400 Bad Request

La requête ne peut pas être traitée par le serveur en raison d'une erreur côté client (syntaxe erronée, etc.).

401 Unauthorized

La requête nécessite une authentification de l'utilisateur.

403 Forbidden

L'accès à la ressource demandée est refusé par le serveur.

404 Not Found

Le serveur n'a pas trouvé la ressource demandée.

500 Internal Server Error

Une erreur générique indiquant que le serveur a rencontré une condition inattendue.

💡 Catégories de Codes

Les codes de statut HTTP sont organisés en 5 catégories : 1xx (Information), 2xx (Succès), 3xx (Redirection), 4xx (Erreur Client), 5xx (Erreur Serveur). Comprendre ces codes est essentiel pour déboguer les problèmes de requêtes HTTP.

JSON : Introduction et Règles

QU'EST-CE QUE JSON ?

JSON (JavaScript Object Notation) est un **format de données léger** apprécié pour la facilité de lecture et écriture par les humains et les machines. Le JSON est construit en deux structures : **objets** (clés-valeurs) et **tableaux**. Il se lit comme un objet en JavaScript.

Objets JSON

Collection de paires clé-valeur entourées d'accolades. Chaque clé est unique et doit être entourée de guillemets.

```
{
"nom": "Alice",
"age": 25,
"email": "alice@example.com",
"actif": true
}
```

Tableaux JSON

Liste ordonnée de valeurs entourées de crochets. Les valeurs peuvent être de différents types (nombres, chaînes, objets, etc.).

```
[
"Alice",
"Bob",
"Charlie"
]

Ou avec objets :
[
{"id": 1, "nom": "Alice"},
{"id": 2, "nom": "Bob"}
]
```

RÈGLES DE SYNTAXE

- Chaque **clé doit être entourée de guillemets** (pas d'apostrophes)
- Les **valeurs chaînes doivent être entourées de guillemets**
- Les **nombres, booléens et null** n'ont pas besoin de guillemets
- Les **clés doivent être uniques** dans un objet

JSON : Conversions en JS

L'OBJET JSON GLOBAL

JavaScript fournit un objet **JSON** global qui permet de convertir entre des chaînes JSON et des objets JavaScript. Cet objet dispose de deux méthodes principales pour les conversions bidirectionnelles.

JSON.parse()

Convertit une **chaîne JSON en objet JavaScript**. Utile pour traiter les données reçues d'une API ou d'un serveur.

```
// Chaîne JSON
const jsonString = '{"nom":"Alice","age":25}';

// Conversion en objet JS
const obj = JSON.parse(jsonString);

console.log(obj.nom); // "Alice"
console.log(obj.age); // 25
```

JSON.stringify()

Convertit un **objet JavaScript en chaîne JSON**. Utile pour envoyer des données à un serveur ou les stocker.

```
// Objet JavaScript
const obj = {
nom: "Bob",
age: 30
};

// Conversion en chaîne JSON
const jsonString = JSON.stringify(obj);

console.log(jsonString);
// '{"nom":"Bob","age":30}'
```

CAS D'USAGE COURANTS

- **Recevoir des données d'une API** - Utiliser JSON.parse() pour convertir la réponse en objet JavaScript manipulable
- **Envoyer des données à un serveur** - Utiliser JSON.stringify() pour convertir l'objet en chaîne JSON avant l'envoi
- **Stocker des données localement** - Utiliser JSON.stringify() pour sauvegarder les données dans localStorage

Fetch API : Introduction et Syntaxe

QU'EST-CE QUE FETCH ?

Fetch est une **fonction JavaScript standard** utilisée pour effectuer des **requêtes réseau (HTTP)**. Elle remplace l'ancienne API XMLHttpRequest et offre une interface plus moderne et plus facile à utiliser. **Fetch retourne une Promesse**, ce qui permet de gérer les réponses asynchrones.

SYNTAXE DE BASE

La forme la plus simple de fetch est **fetch(url)**, où url est l'adresse de la ressource que vous souhaitez récupérer. Par défaut, fetch effectue une requête **GET**.

```
// Requête GET simple
fetch('https://api.example.com/data')
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error('Erreur:', error));
```

CARACTÉRISTIQUES CLÉS

- **Retourne une Promesse** - Permet de gérer les réponses asynchrones avec .then() ou async/await
- **GET par défaut** - Sans options supplémentaires, fetch effectue une requête GET
- **Configurable** - Accepte un deuxième argument (objet options) pour POST, PUT, DELETE, etc.
- **Conversion JSON** - La méthode .json() convertit la réponse en objet JavaScript
- **Gestion d'erreur** - Utilisez .catch() ou try...catch pour gérer les erreurs

Fetch API : Traitement de la Réponse

FLUX DE TRAITEMENT

Lorsqu'on reçoit une réponse de **fetch**, elle n'est pas directement en format JSON. La méthode **.json()** de l'objet de réponse la convertit en un objet JavaScript.

1. **Requête Envoyée**
   fetch() envoie la requête au serveur
2. **Réponse Reçue**
   Le serveur retourne un objet Response
3. **Conversion JSON**
   .json() convertit le corps en objet JavaScript
4. **Données Utilisables**
   Les données peuvent être utilisées dans le code

EXEMPLES DE CODE

Avec .then() et .catch()

```
fetch('/api/data')
.then(r => r.ok ? r.json() : Promise.reject('Erreur'))
.then(d => console.log('Données:', d))
.catch(e => console.error('Erreur:', e));
```

Avec async/await

```
async function getData() {
try {
const r = await fetch('/api/data');
if (!r.ok) throw new Error('Erreur');
const d = await r.json();
console.log('Données:', d);
return d;
} catch (e) {
console.error('Erreur:', e);
}
}
```

Fetch API : Options Clés

OPTIONS COURANTES

method

Définit la méthode HTTP pour la requête (GET, POST, PUT, DELETE, PATCH, etc.). Par défaut : GET.

headers

Un objet représentant les en-têtes HTTP à envoyer avec la requête (ex: 'Content-Type': 'application/json', 'Authorization': 'Bearer token').

body

Le corps de la requête. Utilisé avec des méthodes comme POST, PUT, PATCH pour envoyer des données au serveur. Doit être une chaîne JSON ou FormData.

OPTIONS AVANCÉES

mode

Définit le mode de la requête : 'cors' (cross-origin), 'no-cors', 'same-origin'. Par défaut : 'cors'.

credentials

Contrôle si les cookies et authentifications sont envoyés : 'omit', 'same-origin', 'include'.

cache

Définit la politique de mise en cache : 'default', 'no-store', 'reload', 'no-cache', 'force-cache'.

redirect

Gère le comportement lors d'une redirection : 'follow', 'error', 'manual'. Par défaut : 'follow'.

signal

Permet d'associer un AbortSignal pour annuler la requête à tout moment.

> 💡 **Utilisation Courante**
>
> Pour la plupart des requêtes simples, vous n'aurez besoin que de **method**, **headers** et **body**. Les options avancées sont utiles pour des cas spécifiques comme la gestion des CORS, les redirections, ou l'annulation de requêtes.

Fetch API : Exemple Complet

CODE COMPLET

```
async function createUser(userData) {
try {
// 1. Préparer les options
const options = {
method: 'POST',
headers: {
'Content-Type': 'application/json',
'Authorization': 'Bearer token123'
},
body: JSON.stringify(userData)
};

// 2. Envoyer la requête
const response =
await fetch('/api/users', options);

// 3. Vérifier le statut
if (!response.ok) {
throw new Error(
`Erreur ${response.status}: ${response.statusText}`
);
}

// 4. Convertir en JSON
const data =
await response.json();

// 5. Retourner les données
console.log('Utilisateur créé:', data);
return data;

} catch (error) {
console.error('Erreur lors de la création:', error.message);
// Gérer l'erreur (afficher message, etc.)
throw error;
}
```

EXPLICATIONS

1. Fonction async

La fonction est déclarée avec **async** pour pouvoir utiliser **await** à l'intérieur.

2. Options de requête

Objet contenant **method** (POST), **headers** (Content-Type et Authorization), et **body** (données JSON converties avec JSON.stringify).

3. await fetch()

Envoie la requête et **attend** la réponse du serveur. Retourne un objet Response.

4. Vérifier response.ok

Vérifie que le statut HTTP est 2xx (succès). Si ce n'est pas le cas, lance une erreur.

> 💡 Important : fetch ne rejette pas la promesse pour les codes d'erreur HTTP (4xx, 5xx). Il faut vérifier manuellement response.ok ou response.status.

5. await response.json()

Convertit le corps de la réponse en objet JavaScript. C'est aussi une opération asynchrone.

6. try...catch

Capture toutes les erreurs (réseau, parsing JSON, erreurs lancées manuellement) et les gère gracieusement.

Conclusion

3 CONCEPTS CLÉS À RETENIR

1

HTTP : Le Protocole de Communication Web

2

JSON : Format de Données Universel

3

Fetch API : Effectuer des Requêtes en JS

RESSOURCES POUR APPROFONDIR

MDN Web Docs - Documentation complète sur HTTP, JSON et Fetch API

JavaScript.info - Tutoriels interactifs sur les requêtes réseau

Fetch API Specification - Spécification officielle du W3C

REST API Best Practices - Guide pour concevoir des APIs RESTful

Continuez à pratiquer en construisant des applications web qui communiquent avec des serveurs. Les requêtes HTTP, JSON et Fetch API sont les fondations de la web moderne !