JSON et Fetch

1WEBD – Javascript Web Development

Sommaire

1. JSON.
2. HTTP.
3. Fetch.

1. JSON

1. JSON

Introduction

- JSON (Javascript Object Notation): format de données
- apprécié pour la facilité de lecture et écriture par humains et machines

1. JSON

Fondamentaux

- le JSON est construit en 2 structures, objets et tableaux
- il se lit comme un objet en JS, cad une liste de clés-valeurs

1. JSON

Règles

- chaque clef est unique dans son objet et doit être entourée de guillemets

1. JSON

Règles

```
{
  "life":42,
  "life":24
} // objet non valide

{
  "life":42,
  "guide": {
    "life":42
  }
} // objet valide
```

1. JSON

Conversions en JS

- il y a un objet JSON disponible en JavaScript
- cet objet peut parser une chaine de caractères représentant du JSON avec **JSON.parse**
- il peut transformer un objet JSON en chaine de caractères avec **JSON.stringify**

1. JSON

Conversion en JS

```
const jsonString = '{"nom":"Dupont","age":30,"email":"dupont@example.com"}';
// Conversion de l'objet JavaScript en chaîne JSON

// Conversion de la chaîne JSON en objet JavaScript
const utilisateur = JSON.parse(jsonString);

console.log(utilisateur.nom);  // Affiche: Dupont
console.log(utilisateur.age);  // Affiche: 30

const jsonString = JSON.stringify(utilisateur);

console.log(jsonString);  // Affiche: '{"nom":"Dupont","age":30,"email":"dupont@example.com"}'
```

1. JSON

Alternatives

- bien que ce soit le format le plus adopté dernièrement, JSON n'est pas le seul moyen de transférer des données entre applications
- XML
- Protobuffs
- SOAP
- RPC

1. JSON

2. HTTP

2. HTTP

Introduction

- le protocole HTTP (Hypertext Transfer Protocol) est le fondement du transfert de données sur le Web
- il s'agit d'un protocole de communication client-serveur qui permet aux navigateurs Web de récupérer des informations des serveurs Web

2. HTTP

Client Serveur

- dans le cadre de HTTP, le client est généralement un navigateur Web qui envoie une requête au serveur
- le serveur, qui héberge les données du site Web, répond à cette requête
- cependant, le client peut aussi être un autre serveur

2. HTTP

Connexion Stateless

- 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

2. HTTP

Verbes

- Définissent l’action à effectuer sur la ressource identifiée par un URL
- **GET** : Utilisée pour récupérer des données d'une ressource spécifiée. Elle ne doit pas affecter l'état de la ressource (idempotente). Default quand une page est chargée
- **POST** : Utilisée pour envoyer des données à un serveur pour créer ou mettre à jour une ressource. Les données sont incluses dans le corps de la requête.

2. HTTP

Verbes

- **PUT** : Utilisée pour envoyer des données à un serveur pour créer une nouvelle ressource ou remplacer une représentation existante de la ressource ciblée.
- **PATCH** : Utilisée pour appliquer des modifications partielles à une ressource. Contrairement à PUT, PATCH est utilisée pour faire des mises à jour partielles sur une ressource
- **DELETE** : Utilisée pour supprimer la ressource spécifiée.

2. HTTP

Verbes - Exemple

- Charger la page d’un site -> GET
- Envoyer ses identifiants pour se connecter -> POST
- Modifier son profil -> PUT ou PATCH
- Effacer son compte -> DELETE

2. HTTP

CRUD

- **C**reate
- **R**ead
- **U**pdate
- **D**elete

2. HTTP

CRUD

- Utile pour décrire les fonctionnalités d'une API
- Très souvent, pour chaque ressource (ex: un utilisateur), on doit pouvoir lire, créer, modifier et effacer
- Tous les utilisateurs ne doivent pas avoir ses droits (par exemple, seuls les administrateurs peuvent avoir la liste des utilisateurs), mais l'API devrait pouvoir les gérer

2. HTTP

Cookies

- pour pallier le manque de mémoire du protocole HTTP, les cookies sont utilisés
- ils permettent au serveur de stocker des informations sur le client pour une utilisation ultérieure (par exemple, authentifié un utilisateur)

2. HTTP

Requêtes

- une requête HTTP est initiée par le client
- elle comprend généralement:
  - **Méthode**: Indique l'action que le client veut effectuer. Les plus courantes sont GET (pour récupérer des données) et POST (pour soumettre des données).
  - **URL** (Uniform Resource Locator): L'adresse de la ressource sur le serveur.
  - **Version HTTP**: La version du protocole utilisée.
  - **En-têtes (headers)**: Fournissent des informations supplémentaires sur la requête, comme le type de contenu attendu.

2. HTTP

Réponse

- après avoir reçu et traité une requête, le serveur envoie une réponse qui comprend :
  - **Statut**: Un code de statut (comme 200 pour succès, 404 pour non trouvé) et un message de statut
  - **Version HTTP**: La version du protocole utilisée dans la réponse
  - **En-têtes (headers)**: Similaires aux en-têtes de requête, ils fournissent des informations supplémentaires sur la réponse
  - **Corps de la réponse**: Les données réelles demandées, comme le HTML d'une page Web

2. HTTP

Statuts – Fiche

- **200 OK** : La requête a réussi.
- **301 Moved Permanently** : Cette réponse indique que l'URI de la ressource demandée a été modifiée de manière permanente. Les futurs liens devraient utiliser une nouvelle URL.
- **304 Not Modified** : Indique que la ressource n'a pas été modifiée depuis la dernière demande du client. Utilisé pour la mise en cache.
- **400 Bad Request** : La requête ne peut pas être traitée par le serveur en raison d'une erreur apparente côté client (syntaxe erronée, taille trop grande, etc.).

2. HTTP

Statuts – Fiche

- **401 Unauthorized** : Ce statut indique que 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 qui l'a empêché de répondre à la requête.

2. HTTP

CORS

- CORS est une politique de sécurité qui permet aux serveurs qui permet de spécifier qui peut accederers à leurs ressources
- Il s’agit d’une extension de la politique de même origine (Same-Origin Policy)
- CORS utilise des en-têtes HTTP pour permettre ou refuser les requêtes cross-origin
- Ces en-têtes indiquent au navigateur s'il doit ou non bloquer une requête web inter-domaines

2. HTTP

CORS - Preflight

- Pour certaines requêtes, le navigateur envoie d'abord une requête de vérification (preflight) au serveur cible.
- Cette requête utilise la méthode OPTIONS et sert à vérifier si la requête cross-origin est autorisée.
- **Access-Control-Allow-Origin: https://example.com**
- Cet en-tête permet aux requêtes provenant de https://example.com d'accéder à la ressource

2. HTTP

CORS - Réponse

- Le serveur répond avec des en-têtes spécifiques comme **Access-Control-Allow-Origin**.
- Ces en-têtes indiquent les origines autorisées et les méthodes HTTP permises.

2. HTTP

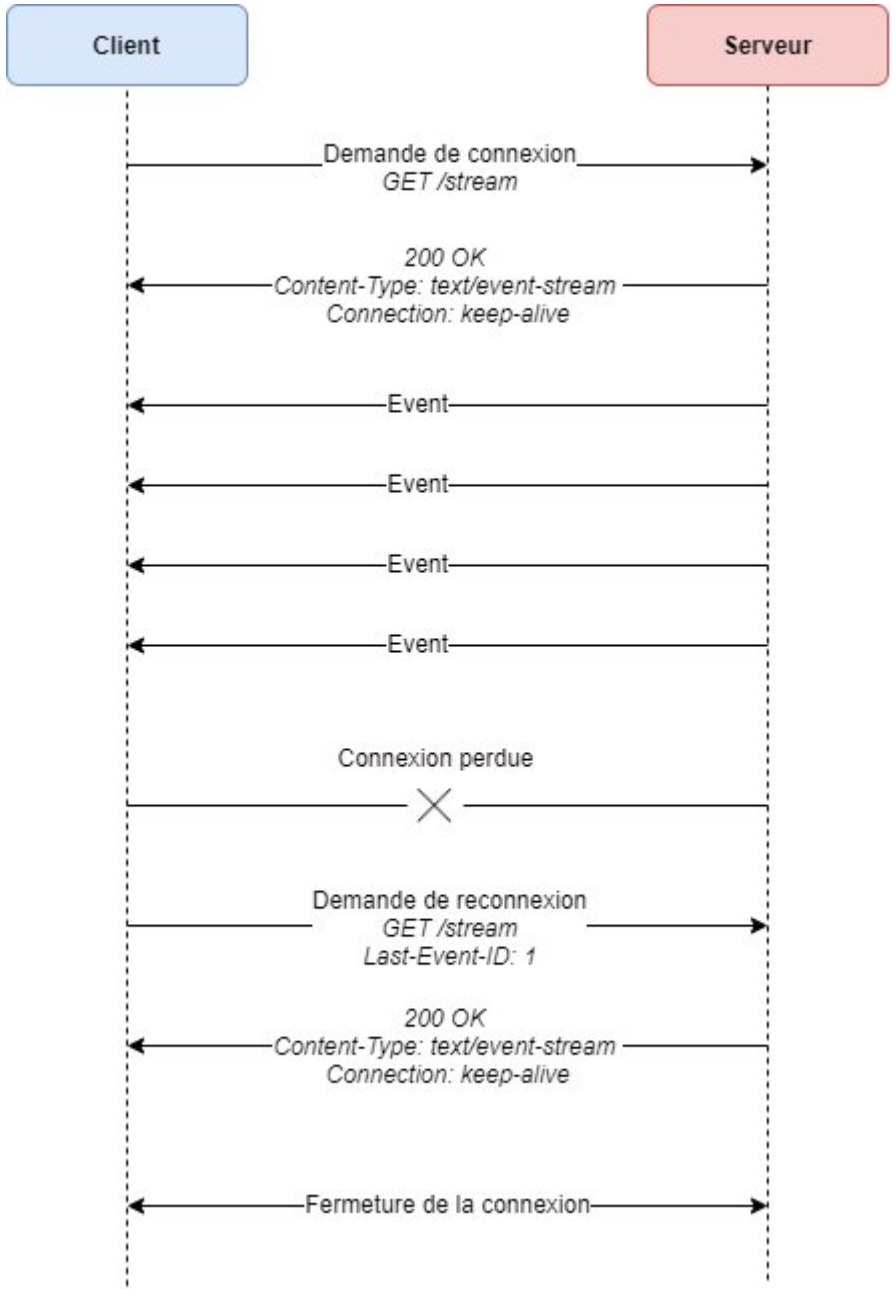
SSE (Server Sent Events)

- Technique pour envoyer des mises à jour du serveur vers le client
- Utilise le protocol HTTP (pas d’outils externes nécessaire)
- Impact léger sur les performances serveur

2. HTTP

SSE (Server Sent Events) - Fonctionnement

- Le header Connection-Type de la réponse doit être **text/event-stream**
- Le header Connection de la réponse doit être **keep-alive**
- La connexion avec le serveur n'est pas fermée après la réponse initiale
- Le serveur continue à envoyer des données



2. HTTP

SSE (Server Sent Events) – Cote client

- Pour JS, on peut utiliser la classe **EventSource**

```
const eventSource = new EventSource('http://localhost:3000/events');
eventSource.onmessage = function(event) {
  console.log("data", event.data)
};
eventSource.onerror = function(err) {
  console.error('EventSource failed:', err);
  eventSource.close();
};
```

2. HTTP

3. Fetch

3. Fetch

Introduction

- API (Application Programming Interface): ensemble de règles qui permettent aux programmes de communiquer entre eux
- dans le contexte web, elles sont utilisées pour communiquer avec des sources externes à l'application (et dans certaines architectures même internes)
- fetch est une fonction JavaScript standard utilisé pour effectuer des requetés réseau

3. Fetch

Syntaxe

- la fonction **fetch** permet de faire des requêtes HTTP. Sa syntaxe la plus simple est **fetch(url)**, où **url** est l'adresse de la ressource que vous souhaitez récupérer
- par défaut, **fetch** effectue une requête GET, mais il peut être configuré pour d'autres types de requêtes (POST, PUT, DELETE, etc.) en passant un deuxième argument sous forme d'objet avec des options supplémentaires
- 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
- il est important de gérer les erreurs potentielles, par exemple en utilisant **.catch()** ou un block **try…catch** pour traiter les cas où la requête échoue

3. Fetch

Syntaxe

```
fetch('https://api.exemple.com/data')
  .then(response => response.json()) // Convertit la réponse en JSON
  .then(data => console.log(data))   // Manipule les données
  .catch(error => console.error('Erreur:', error));
```

```
async function getData() {
  try {
    const response = await fetch('https://api.exemple.com/data')
    const data = await response.json()
    console.log(data)
  } catch (error) {
    console.error("Erreur:", error)
  }
}
```

3. Fetch

Options - Fiche

- **method** : Définit la méthode HTTP pour la requête
- **headers** : Un objet représentant les en-têtes HTTP à envoyer avec la requête.
- **body** : Le corps de la requête. Utilisé avec des méthodes comme **POST** ou **PUT** pour envoyer des données au serveur.
- **mode** : Définit le mode de la requête, tel que **cors**, **no-cors**, ou **same-origin**.
- **credentials** : Contrôle si les cookies et autres informations d'authentification sont envoyés avec la requête.

3. Fetch

Options - Fiche

- **cache** : Définit la politique de mise en cache pour la requête
- **redirect** : Gère le comportement de **fetch** lorsqu'une redirection se produit
- **referrer** : Contrôle l'en-tête **Referer** qui sera envoyé avec la requête
- **referrerPolicy** : Définit la politique à appliquer pour l'en-tête **Referer**
- **integrity** : Permet de vérifier que la ressource récupérée est livrée sans altération, en utilisant un hash SHA.
- **signal** : Permet d'associer un objet **AbortSignal** pour pouvoir annuler la requête.

3. Fetch

Exemple

```html
<body>
    <!-- Section pour afficher les données récupérées -->
    <div id="data-container">
        <h2>Données récupérées de l'API :</h2>
        <ul id="data-list"></ul>
    </div>
```

```
    <script>
        // Fonction pour récupérer des données de l'API
        function fetchData() {
            fetch('https://jsonplaceholder.typicode.com/posts')
                .then(response => {
                    // Vérifie si la réponse est réussie
                    if (!response.ok) {
                        throw new Error('Réseau ou réponse du serveur problématique');
                    }
                    return response.json(); // Transforme la réponse en JSON
                })
                .then(data => {
                    const dataList = document.getElementById('data-list');
                    // Construit des éléments de liste avec les données
                    data.forEach(item => {
                        const listItem = document.createElement('li');
                        listItem.textContent = `${item.title}: ${item.body}`;
                        dataList.appendChild(listItem);
                    });
                })
                .catch(error => {
                    console.error('Erreur lors de la récupération des données:', error);
                });
        }

        // Appelle la fonction fetchData lorsque la page est chargée
        window.onload = fetchData;
    </script>
</body>
```

3. Fetch