Bachelor CDWFS
Applications web dynamiques et interactives
2025-2026

Javascript

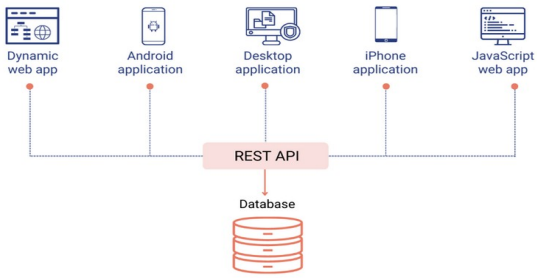
Fetch – 1ère partie

Pré-requis JS : syntaxe de base, tableaux, fonctions, fonctions fléchées, objets littéraux.

Contexte

Requêtes d'API

Fetch est utilisé pour lancer des requêtes http en javascript vers des API.



copyright : datasciencedojo.com

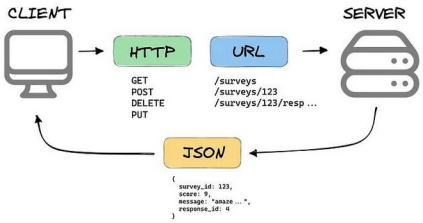
Transmission de données

Les données uniquement

Les réponses aux requêtes d'API contiennent généralement uniquement des données, et non des pages web.

Les données transitent sous forme de chaîne de caractères.

- Les données scalaires (chaines de caractères et valeurs numériques) peuvent transiter telle qu'elle sur le réseau.
- Les données « structurées » (tableaux, objets) doivent être convertis en chaîne de caractères. Ces chaines sont généralement au format json ou xml. Nous allons voir plus en détail le format json.
- Les flux de données et les fichiers transitent sous d'autres formes.



Les données au format JSON

Format Json

Une chaîne JSON peut être stockée dans son propre fichier qui se présente simplement sous la forme d'un fichier texte avec l'extension .json et le MIME type application/json.

Structure :

Exemple 1

```
{
    "id": 1,                                // 1
    "title": "Ajax",                        // 2
    "published": false,                     // 3
    "categories": ["dev", "web"],           // 4
    "status": null                          // 5
}
```

Les accolades **{...}** représentent un objet. Ce JSON est un **objet.** Il contient 4 couples **clefs** – **valeurs.**
Les couples sont séparés par une **virgule** (Il n'y a pas de virgule après le dernier élément).
Les clefs sont entre **double-quote**.
Les valeurs peuvent être :

- des valeurs **numériques** (1)
- des **chaînes** de caractères (2)
- des **booléens** (3)
- des **tableaux** (4). Les crochets **[...]** représentent un **tableau**. Ses éléments sont séparés par une **virgule**.
- des **objets**
- la valeur **null** (5)

Documentation : https://www.json.org/json-en.html

Exemple 2

```
[
    {
      "id": 1,
      "name": "Argelac",
      "latitude": 47.926956,
      "longitude": -4.569858
    },
    {
      "id": 2,
      "name": "Beauvigne",
      "latitude": 44.720258,
      "longitude": -1.953618
    }
]
```

Dans cet exemple, ce JSON est **un tableau** qui contient lui-même deux **objets**.

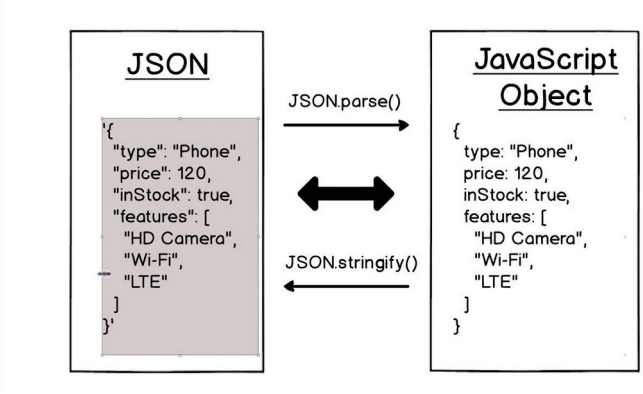
Le JSON est une **structure arborescente**. Elle peut se ramifier sur **plusieurs niveaux** : les objets et les tableaux peuvent contenir d'autres objets et/ou d'autres tableaux.

Conversion Json – Js

Il est nécessaire de convertir le JSON, sous forme d'objet Javascript pour l'utiliser dans les programmes.

➡ Pour plus d'information sur les objets Javascript, consulter le document pdf dédié.

Convertir une chaîne de caractères en un objet littéral Javascript se nomme **désérialisation** (analyse syntaxique ou parsage) tandis que le contraire se nomme **sérialisation** (linéarisation ou stringification).



Copyright : medium.com

Exemple 1, converti en JS

```
{
```

```
    id:1,
    title:"Ajax",
    published:false,
    categories:["dev", "web"],
    status: null
}
```

Exemple 2, converti en JS

```
[
    {
      id: 1,
      name: "Argelac",
      latitude: 47.926956,
      longitude: -4.569858
    },
    {
      id: 2,
      name: "Beauvigne",
      latitude: 44.720258,
      longitude: -1.953618
    }
]
```

Exercice

A quoi ressemblerait ce code Json, après sa conversion en JS

```
{
  "Coktails":
    [
      {
        "id": 1,
        "nom": "Mojito",
        "long drink": true
      },
      {
        "id": 2,
        "nom": "Piña Colada",
        "ingredients": ["Rhum blanc", "jus d'ananas", "Crème de coco"]
      }
  ]
}
```

Introduction à fetch

Syntaxe

La méthode fetch() procure un moyen facile de récupérer des ressources à travers le réseau.

Exemple de requête :

Syntaxe 1 :

```
const url = "https://jsonplaceholder.typicode.com/users"     // 1
fetch(url)                                                    // 2
  .then(function(response){                                   // 3
    return response.json()                                    // 4
  })
  .then(function(data){                                      // 5
    // Votre code ici pour traiter le contenu de la variable data
    console.log(data)                                         // 6
  });
```

Explications :

1. url de la ressource (l'adresse du fichier json)
2. instruction **fetch** avec l'url de la ressource en paramètre
3. l'instruction **then** prend un paramètre une fonction de rappel anonyme qui indique les instructions à exécuter lorsque la **réponse** sera reçue
4. la réponse reçue est un flux de données. La méthode **json()** parse le json en **un objet JS**. L'instruction **return** met fin à la fonction en retournant cet objet.
   https://developer.mozilla.org/en-US/docs/Web/API/Response/json
5. Le paramètre **data** contient l'objet JS (le JSON désérialisé précédement).
6. L'instruction **then** prend en paramètre une fonction de rappel anonyme qui indique les instructions à exécuter. C'est ici que l'on peut exploiter **data** dans un code personnalisé.

Syntaxe 2 :

```
fetch(url)
    .then(response => response.json())
    .then(data => {
        // Votre code ici pour traiter le contenu de la variable data
        console.log(data)
```

```
    });
```

Explications :

Dans cet exemple, les fonctions anonymes sont remplacées par des fonctions fléchées, plus concises.

Syntaxe 3 :

```
async function getData() {                                                  // 1
    const data = await fetch(url).then(response => response.json());        // 2
    // Votre code ici pour traiter le contenu de la variable data
    console.log(data);                                                      // 3
}
getData();                                                                  // 4
```

Explications :

1. Préalable : l'instruction **await** (ligne 2)n'est utilisable que dans une fonction asynchrone. Il faut donc ajouter le mot-clef **async** avant la fonction.
2. Grâce à **await**, le code attend la réponse du serveur et retourne les données dans la variable **data**.
3. Le code JS peut exploiter les données contenues dans **data**
4. Appel de la fonction **getData()**

https://fr.javascript.info/async-await

Précisions sur le premier paramètre de la méthode fetch()

Le premier paramètre de la méthode fetch() est :

soit un chemin relatif

```
const path = './data.json';
fetch(path)
```

Soit une url :

```
const url = 'http://localhost:8080/data.json';
fetch(url)
```

Cette url est :

- Soit une chaine de caractère (cf ci-dessus)
- Soit un objet url : https://developer.mozilla.org/fr/docs/Web/API/URL

```
const url = new URL('http://localhost:8080/data.json');
```

```
fetch(url)
```

- Soit un objet request : https://developer.mozilla.org/fr/docs/Web/API/URL

```
const request = new Request('http://localhost:8080/data.json');
fetch(request)
```

Avant de passer à la pratique

Le protocole HTTP

Le code javascript qui execute fetch doit être situé sur une adresse accessible en http. Comment exécuter son code en http ?

Avec un petit serveur de développement comme « Live server »

- via l'extension Live server de VSCode
- via un package node
  https://classic.yarnpkg.com/en/package/live-server
  https://www.npmjs.com/package/live-server

Avec un serveur web :

- NodeJs
- Apache
- Nginx

Utiliser les outils de développement du navigateur :

Les requêtes fetch sont visibles dans les outils de développement des navigateurs (firefox, chrome, ...)
Click sur Onglet Réseau ou Network > choisir **Fetch** / **XHR** : Cela affiche la liste des requêtes. Pour chaque requête, il est possible de visualiser le détail de la requête et la réponse.

Problèmes de CORS ?

Il arrive que les requêtes fetch echouent. En ouvrant la console (outils de developpement du navigateur), on peut voir ce type de message :

```
Access to fetch at 'https://joke-api-strict-cors.appspot.com/r localhost/:1
andom joke' from origin 'http://localhost:3000' has been blocked by CORS
policy: No 'Access-Control-Allow-Origin' header is present on the requested
resource. If an opaque response serves your needs, set the request's mode
to 'no-cors' to fetch the resource with CORS disabled.
```

Cela signifie que le serveur n'accepte pas la requête, car **le domaine, le port ou le protocole** utilisé n'est pas autorisé. Commencer par vérifier si vous utilisez bien le protocole https par exemple.

Documentation

https://developer.mozilla.org/fr/docs/Web/HTTP/Guides/CORS/Errors

https://php.watch/articles/PHP-Samesite-cookies

Exercices

Pour ces exercices, les pré-requis suivants sont nécessaires :

- connaissance des objets litteraux. ➡ Voir le PDF.
- savoir manipuler les tableaux et les boucles (while, for, forEach, for...in), ➡Voir le PDF.
- savoir créer des élements html de façon programmatique. ➡ Voir le PDF.

Exercice 1 : récupération de données locales: liste de films

Pré-requis : objets litteraux, tableaux

Récupérer le fichier films.json. Il contient une liste de films. Copier ce fichier dans un répertoire de travail.

Créer un fichier javascript et réaliser le code suivant :

- lancer une requête fetch et afficher la liste des films en console

Exercice 2 : récupération de données distantes: yes/no API

Pré-requis : créer des éléments html en js

Interroger l'api https://yesno.wtf/api récupérer la réponse, puis :

- afficher la réponse du serveur dans la console.
- afficher le message et l'image sur une page web

Exercice 3 : données arborescentes sur 3 niveaux de profondeur

Pré-requis : objets litteraux, tableaux

1. Interroger cette url : https://mdn.github.io/learning-area/javascript/oojs/json/superheroes.json
2. Afficher dans la console :
    - la propriété squadName
    - le nombre de membres
    - le nom de chaque membre
    - la liste des pouvoir du premier membre

- la liste des pouvoirs de Molécule Man (utiliser find())
- Le nom des membres ayant le pouvoir "Radiation resistance" (utiliser filter())
- le nom et l'âge du plus jeune (utiliser reduce())

Exercice 4 : codes postaux

Créer une page web où l'utilisateur peut saisir un code postal dans un champ de formulaire.
Lorsque l'utilisateur a saisi un code postal (saisie de 5 caractères), lancer une requête fetch sur cette API : https://www.data.gouv.fr/dataservices/api-carto-module-codes-postaux/
Afficher dans une balise select la ou les communes correspondantes.

Exercice 5 : requêtes d'API REST et mise en place d'une navigation sur 2 pages

Objectif : En utilisant les données d'une API fournissant une liste d'utilisateurs et les détails de chacun, créer une page contenant la liste d'utilisateurs et des liens de navigation vers les pages dédiées à chacun des utilisateurs.

Etape 1 :
Tester cette url dans un navigateur : https://jsonplaceholder.typicode.com/users. Elle retourne des données au format json.
Créer une page list.html qui affiche les propriétés « id » et « username » issue de cette API :

Etape 2 :
L'url https://jsonplaceholder.typicode.com/users/1 retourne l'utilisateur ayant l'id 1. Essayer avec d'autres valeurs.
Créer une page detail.html?id={id} qui affiche le nom et le numéro de téléphone de l'utilisateur dont l'id est passé en paramètre.

Comment récupérer l'id situé dans l'url ?

Utiliser la classe URLSearchParams
https://developer.mozilla.org/fr/docs/Web/API/URLSearchParams

```
const id = new URLSearchParams(window.location.search).get("id");
```

Etape 3 :
Créer une navigation entre les deux pages.

Exercice 6 : Pagination

Objectif : mettre en place une pagination sur l'API pokemon https://pokeapi.co/docs/v2

Etape 1

Tester cette url dans un navigateur. https://pokeapi.co/api/v2/pokemon. Elle retourne :

- le nombre de pokemon (1302)
- les informations sur les 20 premiers
- le lien vers les 20 suivants

Tester l'url suivante : https://pokeapi.co/api/v2/pokemon?limit=100&offset=0 . Que constatez-vous

Etape 2

Créer une page html qui utilisera cette API pour afficher la liste des 100 premiers pokemon.

Etape 3

Créer des liens de navigation « précédent » / « suivant ».

Exercice 7 : construire une requête fetch à travers une IHM : **Mots de passe**

Partie 1

Objectif : Interroger l'api motdepasse.xyz pour générer des mots de passe. https://www.motdepasse.xyz/api/

1. Consulter la documentation qui précise comment utiliser cette API.
2. Choisissez les paramètres souhaités (longueur du mot de passe, caractères inclus dans le mot de passe, etc.) et le nombre de mot de passe.
3. Lancer une requête fetch en précisant les paramètres de requête dans l'url.
4. Afficher le résultat en console.

Exemple de requête : https://api.motdepasse.xyz/create/?include_lowercase&password_length=12&quantity=10

Partie 2

Objectif : créer une interface permettant à l'utilisateur de générer des mots de passe. A travers un formulaire, Il pourra choisir le nombre de mot de passe, leur longueur et les caractères inclus dans le mot de passe.
Pré-requis : formulaires html, gestion d'événements

Conseils

Pour récupérer les valeurs de formulaire, il est possible d'utiliser la classe **FormData**
https://developer.mozilla.org/fr/docs/Web/API/FormData/FormData

Pour construire une URL, en y ajoutant des paramètres, il est possible d'utiliser les classes URL et **URLSearchParams**
https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams

Exercice 8 : construire une requête fetch à travers une IHM : **API - Bornes de recharge VE**

Objectif : Créer une interface permettant de trouver les bornes RVE autour de soi.
Utiliser l'API opendatasoft : https://odre.opendatasoft.com/explore/dataset/bornes-irve/information
Documentation :
https://help.opendatasoft.com/apis/ods-explore-v2/#section/Opendatasoft-Query-Language-(ODSQL)

PARTIE 1
Liste des bornes d'une région / d'un département

Exemples de requêtes :
Filtres de recherche par région
https://odre.opendatasoft.com/api/explore/v2.1/catalog/datasets/bornes-irve/records?where=region=%22Occitanie%22

Par département
https://odre.opendatasoft.com/api/explore/v2.1/catalog/datasets/bornes-irve/records?where=departement=%22Tarn%22

NB : pour obtenir la liste des départments / régions
- liste des régions : https://geo.api.gouv.fr/regions
- liste des départements : https://geo.api.gouv.fr/decoupage-administratif/departements

PARTIE 2
Liste des bornes dans un rayon variable

Modifier l'interface de façon à ce que l'utilisateur puisse renseigner des coordonnées de latitude et de longitude ainsi qu'un rayon.

Exemple de requête : dans un rayon de 1km autour de Castres (Latitude : 43.601962, Longitude : 2.248757)
url non-encodée :
https://odre.opendatasoft.com/api/explore/v2.1/catalog/datasets/bornes-irve/records?where=within_distance(geo_point_borne,geom'POINT(2.248757%2043.601962)',1km)

url encodée :

https://odre.opendatasoft.com/api/explore/v2.1/catalog/datasets/bornes-irve/records?where=within_distance(geo_point_borne,geom%27POINT(2.248757%2043.601962)%27,1km)