

# Développement front

MMI 2 – CM#2 S3



Danielo **JEAN-LOUIS**

# Javascript - Rappels

- Langage permettant de créer des interactions sur un site web
- **Seul** langage de programmation disponible côté navigateur
- Fichier lu de haut en bas
  - Attention à l'ordre de vos instructions

# Javascript - Rappels

- Utilisation de la balise `<script>` pour exécuter le code
- Programmation événementielle
  - Le code réagit en fonction des actions utilisateurs : clic, survol...
- Extension de fichier : `.js`

# Javascript - Rappels

- Script peut être exécuté :
  - Dans un attribut HTML (**à éviter absolument**)
  - Dans une balise `<script>` (à éviter)
  - Dans un fichier externe (à préférer)
- L'exécution est synchrone
  - La navigateur attend la fin d'une instruction pour passer à la suivante
  - On parle de programmation "synchrone"

# Programmation synchrone

- Paradigme de programmation
- Programme exécuté ligne par ligne
  - Le programme attend le retour d'une instruction avant de passer à la ligne suivante

# Programmation synchrone

```
const maFonction = (paramA, paramB) => {  
  const addition = paramA + paramB;  
  const mul = addition * paramA;  
  
  return mul;  
}
```

Tant que l'addition n'a pas été réalisée, le navigateur ne peut pas passer à la ligne suivante

**Que se passe-t-il si une instruction prend du temps ?**



# Le programme est bloqué, vous ne pouvez plus interagir avec la page\*...

\* Voir exemple ressources/synchrone-vs-asynchrone

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fintegration-web-s3%2Fcours-magistraux%2Fnumero-2%2Fressources>

**C'est là qu'entre en jeu la  
programmation asynchrone**

# Programmation asynchrone

- Paradigme de programmation
- Existe dans quasiment tous les langages
- Exécute les instructions “ailleurs” et renvoie la réponse quand c’est terminé

## Source(s) :

- <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Asynchronous/Introducing>
- <https://waytolearnx.com/2019/01/difference-entre-une-execution-synchrone-et-asynchrone.html>

# Programmation asynchrone

- Permet d'exécuter plusieurs instructions en même temps
- Évite de bloquer l'application
  - Améliore l'expérience utilisateur

## Source(s) :

- <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Asynchronous/Introducing>
- <https://waytolearnx.com/2019/01/difference-entre-une-execution-synchrone-et-asynchrone.html>

# Programmation asynchrone

- Paradigme au cœur des expériences web modernes
  - Ex : Commenter une vidéo sans recharger la page
- Repose sur un système de promesses en javascript

## Source(s) :

- <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Asynchronous/Introducing>
- <https://waytolearnx.com/2019/01/difference-entre-une-execution-synchrone-et-asynchrone.html>

# Promesses

- Classe javascript permettant de réaliser des actions asynchrones
- Trois états possibles :
  - pending (en attente) : état initial
  - fulfilled (tenue) : l'opération a réussi
  - rejected (rompue) : l'opération a échoué

Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Promise)

# Promesses

- Attente de la réponse réalisée grâce au mot-clé “await”
- La méthode “fetch” est une promesse
- Promise en anglais

## Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Promise)

# API Fetch

- API native
- Gérée par tous les navigateurs modernes
- Permet d'effectuer des requêtes serveur asynchrones
  - Charger un fichier ou appeler un serveur
  - Évite le rechargement de la page

Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)
- <https://grafikart.fr/tutoriels/javascript-promise-2068>



# API Fetch

- Version moderne de XMLHttpRequest
  - Évitez d'utiliser XMLHttpRequest de nos jours
- Retourne sur une promesse

## Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)
- <https://grafikart.fr/tutoriels/javascript-promise-2068>

# API Fetch

- Indispensable pour l'utilisation des frameworks javascript (S4)
- Appelé également AJAX
  - Asynchronous JavaScript and XML

## Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)
- <https://grafikart.fr/tutoriels/javascript-promise-2068>

# API Fetch - Exemple

```
const chercherDonnees = async () => {  
  const requete = await fetch("URL");  
  const resultat = await requete.json();  
  
  console.log(resultat);  
}
```

Ce code effectue une requête asynchrone vers un serveur

**Source(s) :**

- [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)
- <https://grafikart.fr/tutoriels/javascript-promise-2068>

# API Fetch - Exemple

```
const chercherDonnees = async () => {  
  const requete = await fetch("URL");  
  const resultat = await requete.json();  
  
  console.log(resultat);  
}
```

On définit notre fonction **asynchrone** grâce au mot-clé “async”

Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)
- <https://grafikart.fr/tutoriels/javascript-promise-2068>

# Mot-clé “async”

- Permet de définir une fonction asynchrone
- Fonctionne de pair avec le mot-clé “await”
  - “await” permet d’attendre l’exécution d’une action
  - **“await” ne peut pas fonctionner sans “async”**

## Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/async\\_function](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/async_function)
- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/await>

# API Fetch - Exemple

```
const chercherDonnees = async () => {  
  const requete = await fetch("URL");  
  const resultat = await requete.json();  
  
  console.log(resultat);  
}
```

On effectue et attend la réponse de la requête HTTP.  
“URL” est un chemin vers une API

Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)
- <https://grafikart.fr/tutoriels/javascript-promise-2068>

# Requête HTTP

- Action à effectuer par un serveur
- Représenté par neuf méthodes
  - **Chaque méthode a un rôle distinct**
- La fonction fetch prend en deuxième paramètre un objet pour préciser le type de méthode

Source(s) :

- [https://fr.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol#M%C3%A9thodes](https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol#M%C3%A9thodes)

# Requête HTTP - GET

- Permet de récupérer une ressource
  - Ex : Afficher une page web
- Peut être rejoué à l'envie
- Méthode par défaut avec fetch
- Peut être exécuté par le navigateur

Source(s) :

- [https://fr.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol#M%C3%A9thodes](https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol#M%C3%A9thodes)



# Requête HTTP - POST

- Permet de créer une ressource
  - Ex : Créer un utilisateur
- Peut être exécuté par le navigateur

```
const creerUtilisateur = async () => {  
  const requete = await fetch("http://www.example.com/api", {  
    method: "POST",  
  });  
  const resultat = await requete.json();  
  
  console.log(resultat);  
};
```

En passant un deuxième paramètre, il est possible de changer le type de requête.  
Ici "POST".

# Requête HTTP - PUT

- Permet de mettre à jour une ressource
  - Ex : Modifier un message
- Ne peut pas être exécuté par le navigateur

## Source(s) :

- [https://fr.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol#M%C3%A9thodes](https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol#M%C3%A9thodes)

# Requête HTTP - DELETE

- Permet de supprimer une ressource
- Ne peut pas être exécuté par le navigateur

## Source(s) :

- [https://fr.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol#M%C3%A9thodes](https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol#M%C3%A9thodes)

# API Fetch - Exemple

```
const chercherDonnees = async () => {  
  const requete = await fetch("URL");  
  const resultat = await requete.json();  
  
  console.log(resultat);  
}
```

On transforme la réponse de la requête au format JSON.  
**Étape indispensable**, sinon vous n'aurez aucune données exploitables.

Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)

# Format JSON

- Syntaxe inspirée des objets JavaScript
  - Mais indépendant du javascript
  - Système de clé-valeur
- Ne pas confondre avec les objets JavaScript

Source(s) :

- <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JSON>


# Format JSON

- Format standard dans la communication client-serveur
  - Géré par quasiment tous les langages
- Chaque clé d'un objet devrait être unique
  - Il n'y aura pas crash si cette règle n'est pas respectée

Source(s) :

- <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JSON>

# Format JSON



```
{  
  "formation": "MMI",  
  "etablissement": "IUT Sarcelles",  
  "tp": "3"  
}
```

Un exemple d'objet JSON

Source(s) :

- <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JSON>

# Format JSON

JS objet-json.js

```
const jsonObj = {  
  "formation": "MMI",  
  etablissement: "IUT Sarcelles",  
  tp: "3",  
}
```

{ } fichier.json

```
{  
  "formation": "MMI",  
  "etablissement": "IUT Sarcelles",  
  "tp": "3"  
}
```

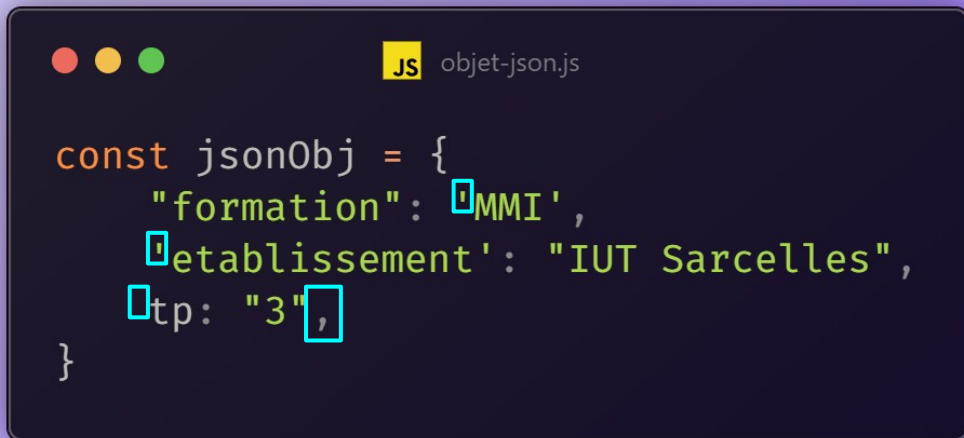
Différence entre un objet JS (gauche) et un objet JSON (droite)  
Quelles sont-elles ?

Source(s) :

- <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JSON>



# Format JSON



```
JS objet-json.js

const jsonObj = {
  "formation": 'MMI',
  'etablissement': "IUT Sarcelles",
  'tp': "3",
}
```



```
{ } fichier.json

{
  "formation": "MMI",
  "etablissement": "IUT Sarcelles",
  "tp": "3"
}
```

- La virgule finale est autorisée
- Les clés n'ont pas besoin de guillemets
- L'utilisation de guillemets simples est autorisée pour les clés et valeurs

Source(s) :

- <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JSON>

# Format JSON


- Conteneur de données :
  - Ne peut pas contenir des fonctions
  - Données hiérarchisées (possibilité d'imbriquer)
- Chaque clé et valeurs doivent être entre guillemets. Sauf :
  - Nombre, sous objets et tableau
- Extension de fichier .json

## Source(s) :

- <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JSON>
- [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global\\_Objects/JSON](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/JSON)

# Format JSON

- Valeurs accessibles via le nom de la clé associée
  - Nécessite que le JSON soit dans une variable



```
const monJSON = {  
  "formation": "MMI",  
  "etablissement": "IUT Sarcelles",  
  "tp": "3"  
};  
// Affichera "MMI"  
console.log(monJSON.formation);
```

Source(s) :

- <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JSON>

# Format JSON

- Possibilité d'accéder aux valeurs soit par la notation pointée soit par la notation crochets



```
// Les deux afficheront "MMI"  
console.log(monJSON.formation);  
console.log(monJSON["formation"]);
```

La notation crochets a l'avantage de pouvoir accepter une variable

Source(s) :

- <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JSON>
- <https://www.freecodecamp.org/news/dot-notation-vs-square-brackets-javascript/>

# API Fetch - Exemple

```
const chercherDonnees = async () => {  
  const requete = await fetch("URL");  
  const resultat = await requete.json();  
  
  console.log(resultat);  
}
```

Enfin, on affiche la réponse du serveur grâce à `console.log()`

Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)

# API Fetch - Exemple

```
const chercherDonnees = async () => {  
  const requete = await fetch("URL");  
  const resultat = await requete.json();  
  
  console.log(resultat);  
}
```

Ce code effectue une requête asynchrone vers un serveur et affiche le résultat dans la console du navigateur

Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)

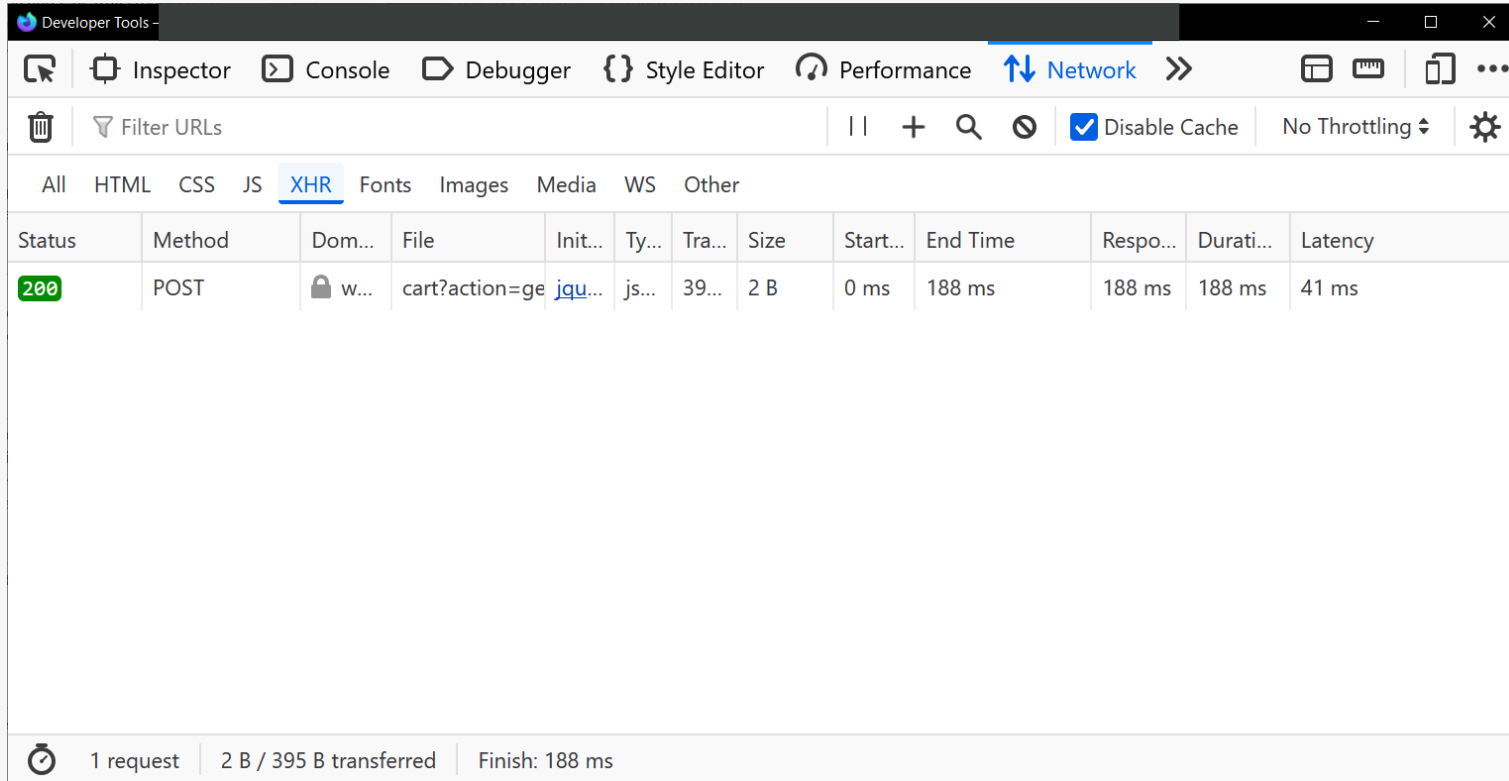
# API Fetch - Debuggage

- Utilisation de la console du navigateur. Onglet “Network” ou “Réseau”. Menu “Fetch/XHR”
  - Permet de voir les requêtes asynchrones exécutées
    - Ce que le navigateur envoie
    - Ce que le navigateur reçoit

## Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)

# API Fetch - Debuggage



La console du navigateur permet de voir les requêtes exécutées

Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)




# Promesses – Gestion des erreurs

- Utilisation du bloc “try/catch”
  - Permet de capturer les erreurs pour éviter un crash de l'application
- Erreurs peuvent être de tout type :
  - Variable inconnue
  - Erreur serveur (erreur 404, pas d'internet...)
  - ...

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/try...catch>

# try/catch – Gestion des erreurs



```
try {  
    // code...  
} catch (err) {  
    // Gestion des erreurs  
}
```

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/try...catch>

# try/catch – Gestion des erreurs

- Le code précédent fonctionne ainsi :
  - Le navigateur essaye d'exécuter une des instructions dans le try {}
  - Si une des instructions échoue, le code entre dans le catch (e) {}
- "try" ne peut pas exister sans "catch" et vice-versa

Source(s) :

- [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Promise)

# Pratiquons ! - Découvrons fetch

Pré-requis :

- Avoir la ressource ressources/fetch

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fintegration-web-s3%2Fcours-magistraux%2Fnumero-2%2Fressources>

# Appeler automatiquement une fonction

- Utilisation d'une IIFE
  - Immediately Invoked Function Expression
- Permet d'appeler une fonction dès qu'elle est définie

Source(s) :

- <https://developer.mozilla.org/fr/docs/Glossary/IIFE>

# Appeler automatiquement une fonction

- Fonctionne avec les fonctions asynchrones et synchrones
- Définit son propre contexte :
  - Toute variable ou fonction définie à l'intérieur est inaccessible à l'extérieur

Source(s) :

- <https://developer.mozilla.org/fr/docs/Glossary/IIFE>

# Appeler automatiquement une fonction



```
(async () => {  
    await fetch("URL");  
})();
```

Notre méthode fetch sera appelée automatiquement car elle est définie dans une IIFE

Source(s) :

- <https://developer.mozilla.org/fr/docs/Glossary/IIFE>

# API Fetch – Annuler la requête

- Utilisation des classes AbortSignal et AbortController

## Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/AbortSignal>



# API Fetch – Annuler la requête

```
const controleur = new AbortController();
const signal = controleur.signal;

const recupererDonnes = async () => {
  const requete = await fetch(url, { signal: signal });
  /* [...] */
};

document.querySelector("[btn-stop-requete]").addEventListener("click", () => {
  controleur.abort();
  console.log("Téléchargement interrompu");
});
```

Lors du clic sur le bouton, la requête sera arrêtée si elle est en cours

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/API/AbortSignal>

**Questions ?**

