

JavaScript en résumé

(Re)voir les bases du langage

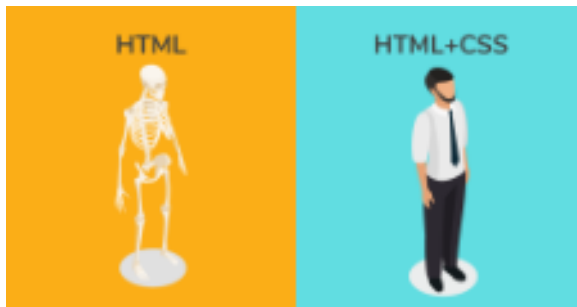
Comprendre le fonctionnement des applications web

Faire ses premières application web en JavaScript



A l'origine il y avait...

- HTML pour le contenu
 - Le fond : l'information
- CSS pour le style
 - La forme : le “**skin**” du site
- Mais il manquait quelque chose...





JavaScript

JavaScript à l'époque



- Né en 1995 (bientôt 30 ans)
- Comme un langage utilitaire par-dessus HTML
- Pour interagir avec le DOM
 - Représentation objet du HTML



JavaScript dans les années 2000



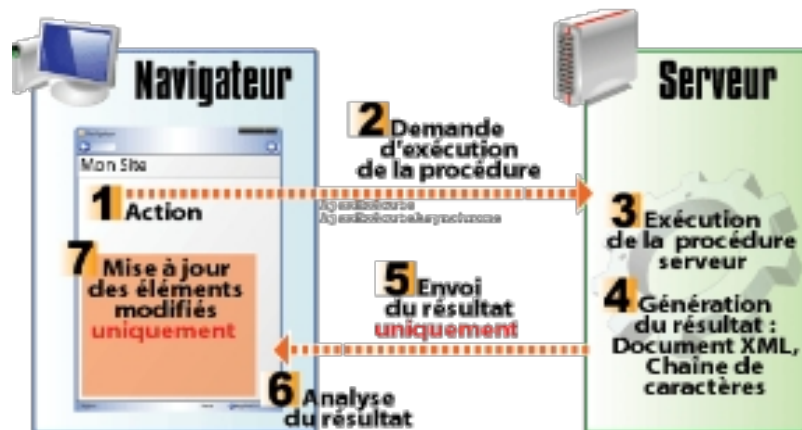
- Langage côté client pour :
 - Dynamiser le contenu d'une page HTML
 - Faire des requêtes AJAX
 - L'âge d'or de jQuery



JavaScript dans les années 2000 - Ajax



- Méthode pour récupérer des données vers/depuis un serveur
- Asynchronous Javascript And XML
 - Envois et récupère des données d'un serveur sans interférer avec l'affichage.
 - Modifier le contenu d'une page web de manière dynamique



JavaScript dans les années 2000 - JQuery



- Bibliothèque Javascript
 - Créer pour faciliter l'écriture de script coté client
 - Modification DOM
 - Gestion d'évènement
 - Utilisation d'AJAX facilité
- Bibliothèque front-end la plus utilisé au monde !
 - Plus de la moitié des sites en ligne aujourd'hui intègre JQuery
 - Réalisé par un étudiant en 2006

JavaScript aujourd'hui



- Langage de programmation ultra répandu
 - Côté client = s'exécute dans les navigateurs
 - Côté serveur = s'exécute sur un serveur (**Node.js**)
- N'a pas grand chose à voir avec Java
 - Sauf le nom parce que c'était à la mode



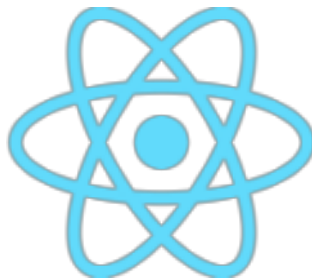
≠



JavaScript aujourd'hui



- XML délaisser au profit du JSON
 - Javascript Object Notation — Format d'échange de données léger et rapide
- XMLHttpRequest délaisser au profit de « Fetch »
 - Requête pour récupérer des données
 - Plus puissante et plus intuitive.
- JQuery délaisser au profit de Vue.js et React



JavaScript aujourd'hui

et c'est pas fini...



Côté serveur

express



METEOR

Moteurs d'exécution



Deno



Bun

Packaging



webpack



Vite

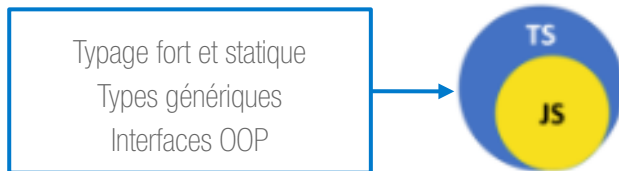
Côté client



Et TypeScript dans tout ça ?



- Un sur-langage à JavaScript développé par Microsoft
 - En gros c'est JavaScript avec des types (entre autres)
 - Chaque variable peut avoir un type à la compilation



- Re compilé en JavaScript
 - Pour pouvoir être exécuté dans les navigateurs

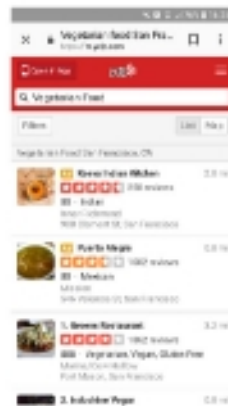


Applications web

- De nos jours, depuis la généralisation de Javascript
 - Site web = application web
 - Application qui s'exécute dans le navigateur



Mobile App

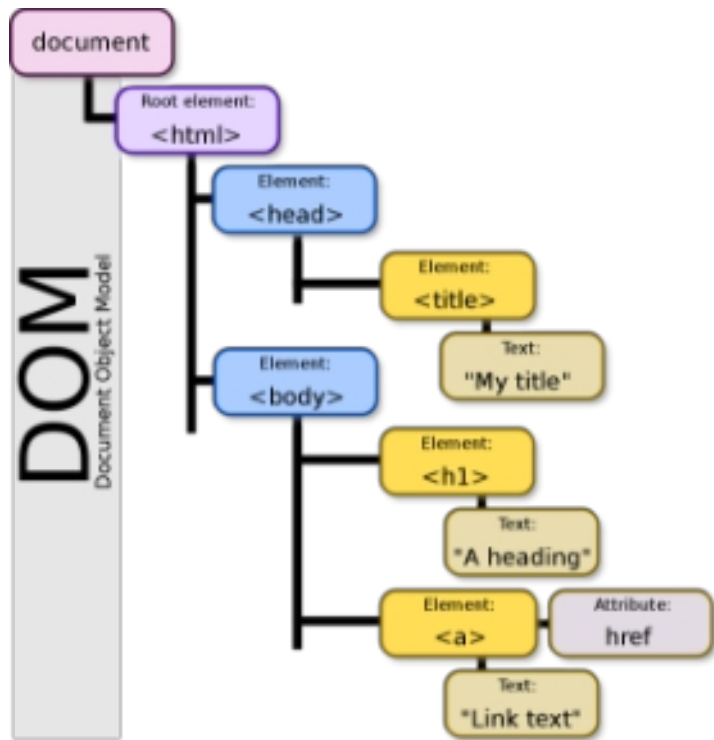


Web App

Ingrédients d'une appli web

- Avec HTML
 - Le contenu : textes, images, liens, ...
- Avec CSS
 - La mise en forme : tailles, disposition (layout), couleurs, ...
 - La gestion du responsive
- Avec **JavaScript**
 - On peut faire tout ce qu'on veut (mais attention)
 - Modifier le contenu et la mise en forme
 - Rajouter de l'interactivité sur les pages → **application**

Le DOM *Document Object Model*



- **Objets** représentant l'arborescence d'un document (XML, HTML, SVG)
- Permet de **manipuler** les noeuds du document avec JavaScript
- On peut créer, modifier, supprimer des noeuds dans notre document et donc le rendre dynamique !

JavaScript : généralités

- Initialement, un langage de script pour les navigateurs
 - Côté client
- De nos jours :
 - Un langage accessible et puissant (ES6)
 - Côté client et côté serveur (Node.js)
 - Paradigmes **objet** et **fonctionnel**
 - Toujours pas de typage des variables ! (voir TypeScript pour ça)
- Avec une syntaxe (trop) simple
 - Proche de celle de Java (d'où le nom)

JavaScript : généralités



- Le langage contient des bugs de conception
 - Ça fait partie du langage maintenant
 - On n'y peut plus rien, il faut développer en les évitant
- Exemples à tester dans la console du navigateur :
 - `0.1 + 0.2`
 - `true + 1`
 - `[] == ''`
- Pour la suite :
 - <https://github.com/denysdovhan/wtfjs/blob/master/README-fr-fr.md>

JavaScript : package manager

- NPM
 - Permet d'installer des librairie
 - `npm install mon_package --save-dev`
- Anecdote
 - How to destroy internet

JavaScript : variables



Attention : absence de typage


- En JavaScript **les variables ne sont pas typées** !
 - Ce sont les valeurs qui sont typées
- Le typage est **faible** et **dynamique**
 - Faible : les variables n'ont pas de type déclaré
 - Dynamique : typage réalisé à l'exécution (pas à la compilation)

Variable typée String
On n'y met **que** des String



Variable non typée
On y met ce qu'on veut

JavaScript : variables

- Déclaration de variables avec les mots-clés **const** et **let**
 - **const** : affectation unique et définitive
 - **let** (remplace var) : portée limitée au bloc englobant
 - Pourquoi ne pas utiliser var ?
 - Portée dans le contexte d'exécution courant :
 - Soit le bloc de la fonction qui contient la déclaration (si déclarée dans une fonction)
 - Soit le contexte global si la variable est déclarée en dehors de toute fonction
 - **Very dangerous !**
-  L'utilisation de var est totalement dépréciée
- Depuis l'apparition de const et let dans ES6

JavaScript : fonctions

- Déclaration de fonction nommée

- `function nomDeLaFonction(params) { /* code... */ }`

- Déclaration de fonction anonyme

- `function(params) { /* code... */ }`

- Affectation de fonction nommée à une variable

- `const maVariable = function nomDeLaFonction(params) { /* code... */ }`

- Affectation de fonction anonyme à une variable

- `const maVariable = function(params) { /* code... */ }`

- Déclaration de fonction fléchée anonyme

- `(params) => { /* code... */ }`

- Affectation de fonction fléchée anonyme à une variable

- `const maVariable = (params) => { /* code... */ }`

JavaScript : modules

- Exporter un module :

- `module.exports = { /* à exporter */ } // on y place les classes, objets, fonctions à exporter`
- `export function ... // ou class, variable ... <-- à préférer`

- Importer un module :

- Pour importer un module, il faut qu'il ait été exporté avant
- Ancienne façon d'importer un module (encore d'actualité dans Node.js)
 - `const Module = require('./module.js');`
- Nouvelle façon d'importer un module (import et from depuis ES6)
 - `import Module from './module.js'; <-- à préférer`
- Le “.js” à la fin du chemin du module est optionnel

Cliquer sur un élément

- Plusieurs façons de faire : onclick en HTML ou .onclick en JS
- Méthode recommandée : **addEventListener(...)**
 - On va attendre un événement (ici le clic)
- Comment faire ?
 - On récupère l'élément du DOM en JS
 - On lui attache une fonction qui écoute un événement
 - Qui se déclenchera chaque fois que...
 - Un événement arrivera ("click" par exemple)

Charger une librairie côté client

- Charger du JS dans une page HTML
- Avec la balise **<script src="...">**
 - Dans la balise head ou body avec **async** ou **defer**
- Par exemple Bootstrap ou anciennement jQuery

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.min.js"
integrity="sha384-mQ93GR66B00ZXjt0YO5KlohRA5SY2XofN4zfuZxLkojlgXtW8ANNce9d5Y3eG5eD"
crossorigin="anonymous"></script>
```

- Avec les frameworks frontend (Angular, React, Vuejs)
 - On fera différemment (npm et Webpack par exemple)

Faire un appel HTTP



- Anciennement appelé AJAX
 - *Asynchronous JavaScript and XML*
- Requête à un serveur, un service web (une API)
- Messages HTTP
 - De requête : envoyés par le client
 - De réponse : envoyés par le serveur
- Verbes HTTP : GET, POST, PUT, PATCH, DELETE, HEAD, ...
- Codes HTTP : 200, 301, 404, 500, ...
- Headers HTTP : Accept, Content-Type, Host, Origin, ...

Les différentes méthodes “AJAX”

- On peut faire des requêtes “AJAX” de différentes façons
- XMLHttpRequest (xhr)
 - A l'ancienne, avec l'API standard
- \$.ajax() avec jQuery
 - Méthode utilitaire qui facilite l'utilisation de XMLHttpRequest
- fetch (ou axios avec les frameworks *front-end*)
 - La nouvelle méthode qui “remplace” XMLHttpRequest

Requête HTTP : à l'ancienne

- L'objet **XMLHttpRequest** (xhr) en JavaScript
- Déployé sur les navigateurs entre 1998 et 2005
- Spécifié par le W3C en 2007
- Requêtes HTTP, directement depuis le navigateur
- Asynchrone avec fonctions de rappel (*callback*)
 - .onreadystatechange, .onload, ...
 - ou en ajoutant des event listeners

Requête HTTP : avec jQuery

- La méthode **`$.ajax`** avec jQuery
- Méthode **utilitaire** qui facilite l'utilisation de XMLHttpRequest
- Permet de simplifier les requêtes AJAX
- jQuery contient beaucoup de méthodes utilitaires :
 - **`$.get`** pour directement faire un GET (`$.post` pour POST)
 - **`$.parseXML`** pour désérialiser une string XML
 - **`$.isXMLDoc`** pour vérifier si un noeud est XML

Voir : <https://api.jquery.com/jquery.ajax/#jQuery-ajax-url-settings>

Voir : <https://www.pierre-giraud.com/jquery-apprendre-cours/creation-requete-ajax/>

Requête HTTP : avec fetch

- La méthode **fetch** de JavaScript
 - Méthode de requêtes HTTP en asynchrone
- Standardisée dans Chrome et Firefox
 - Depuis 2015

```
fetch(url).then(function(response) {  
  // traiter l'objet Response  
});
```

- fetch renvoie une *Promise*, d'où le chaînage avec `.then`
- à l'intérieur de la fonction, on traite l'objet *Response*

fetch : https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

promesses : https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Using_promises

CORS

- *Cross-origin Resource Sharing*
 - <https://developer.mozilla.org/fr/docs/Web/HTTP/CORS>
- Mécanisme de partage de ressources entre domaines
 - Cross-origin = entre différents domaines
 - S'autorise via des en-têtes HTTP (donc côté serveur)
- Par défaut, pour la sécurité
 - les navigateurs interdisent les requêtes cross-origin

Exemple d'extension pour désactiver le blocage dans Chrome :

<https://chrome.google.com/webstore/detail/cors-unblock/lfhmikememgdcahcdlaciloanbhhjino?hl=fr>

CORS

- Exemple de requête à apple.com depuis google.com

○ Dans le navigateur

```
> fetch("https://apple.com")
```

```
< ▶ Promise {<pending>}
```

✖ Access to fetch at 'https://apple.com/' from origin 'https://www.google.fr' 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.

→ Un message d'erreur nous indique qu'il manque le header "Access-Control-Allow-Origin"

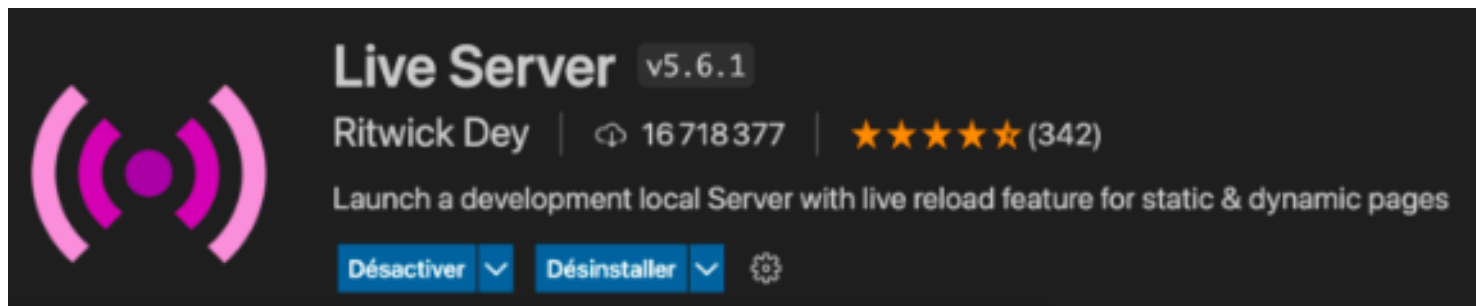
qui pourrait nous autoriser à faire une telle requête, ou bien qu'il faut désactiver CORS.

Exemple d'extension pour désactiver le blocage dans Chrome :

<https://chrome.google.com/webstore/detail/cors-unblock/lfhmikememgdcahcdlaciloanbhhjino?hl=fr>

Tester HTTP sur une page HTML

- Avec l'extension Live Server de VScode



Voir https://developer.mozilla.org/fr/docs/Web/Guide/AJAX/Getting_Started

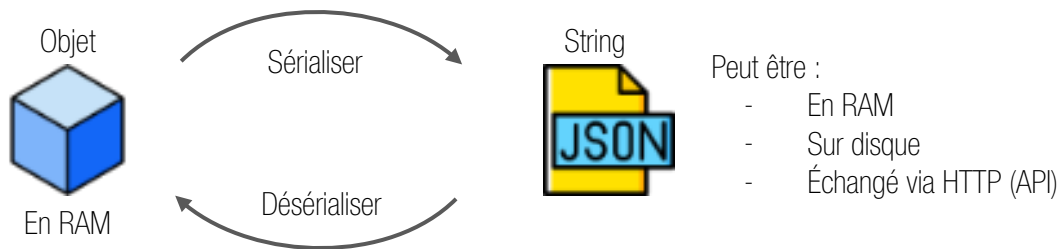
Sérialiser/désérialiser un élément

- On va utiliser le **JSON**
 - On pourrait aussi faire avec XML, YAML, binaire, ou d'autres formats
- Le **JSON** = chaîne de caractères
- Objet JavaScript **≠** JSON
- Des clés-valeurs et des tableaux avec string, number, bool, ...

```
1  [
2    {
3      "title": "Elasticsearch",
4      "description": "Comprendre et développer un moteur de recherche.",
5      "level": "4",
6      "color": "#018FB3",
7      "image": "technologies/elasticsearch.svg"
8    },
9    {
10     "title": "Elastic Stack",
11     "description": "Les outils autour d'Elasticsearch "
```


Sérialiser/désérialiser un élément

- Sérialisation : transformer un objet en chaîne de caractères
- Désérialisation : transformer une chaîne en objet



- Sérialisation : `JSON.stringify(string)`
- Désérialisation : `JSON.parse(objet)`

Lire et écrire dans le localStorage

- Mini base de données locale dans le navigateur
- Écrire : `localStorage.setItem(clé, valeur)`
- Lire : `localStorage.getItem(clé)`
- Effacer un élément : `localStorage.removeItem(clé)`
- Tout effacer : `localStorage.clear()`

Animer un élément



- Méthode **animate** sur un élément du DOM
 - <https://developer.mozilla.org/fr/docs/Web/API/Element/animate>
- Via une librairie JS
 - <https://animejs.com/>
- Via une librairie CSS
 - <https://animate.style/>
- Dans un **<canvas>**
 - https://developer.mozilla.org/fr/docs/Web/API/Canvas_API/Tutorial
 - https://developer.mozilla.org/fr/docs/Games/Tutorials/2D_Breakout_game_pure_JavaScript

Références et ressources

- Documentation de Mozilla sur les technos Web
 - <https://developer.mozilla.org/fr/docs/web>
- En particulier sur JavaScript
 - <https://developer.mozilla.org/fr/docs/Learn/JavaScript>
- Tutos sur différents sites
 - <https://www.w3schools.com/js/default.asp>
 - <https://www.codecademy.com/learn/introduction-to-javascript>
- ...