



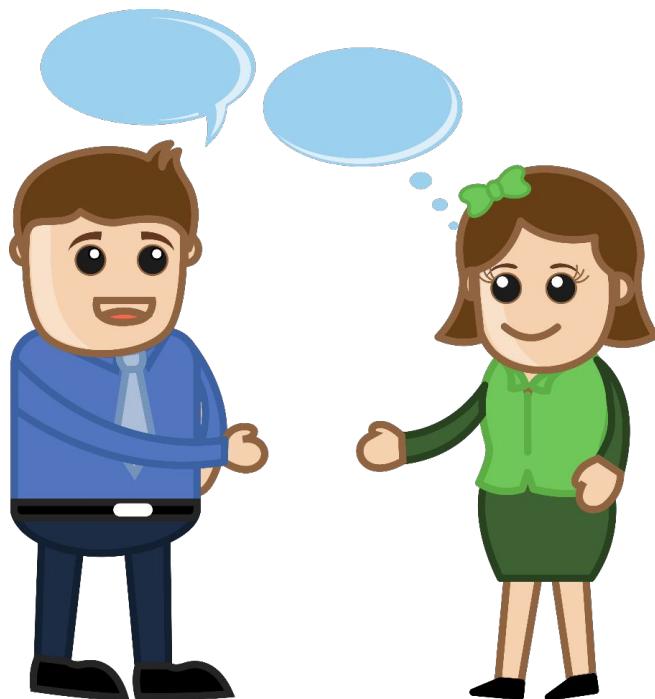
Nicolas Amini-Lamy

**WEBFORCE**  
BE THE CHANGE 

## Présentons-nous

---

- Nom
- Prénom
- Âge
- Objectif personnel
- Passions, détail insolite.. :)



Nicolas AMINI-LAMY

*Ingénieur en architecture des systèmes d'information  
5 années d'expérience en SSII  
Formateur - Entrepreneur depuis 2018*

Pour toute question post-formation :

[pro@naminilamy.fr](mailto:pro@naminilamy.fr)



3

**WEBFORCE**  
BE THE CHANGE  
Nicolas Amini-Lamy

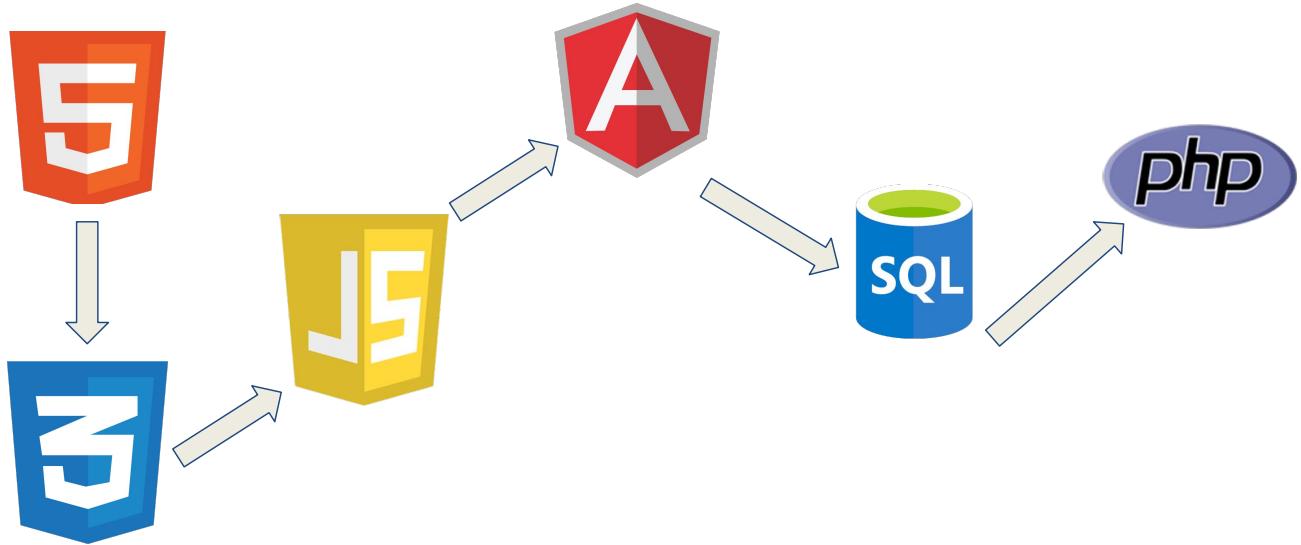
## Programme de la semaine

---

- Point parcours & architectures web
- Compléments JavaScript
- Présentation du langage TypeScript
- Programmation asynchrone
- Présentation Angular & TP Projet

## Point parcours

---



5

## Point parcours

---

Votre parcours est constitué de 2 blocs : front & back.

Vous adressez en premier lieu 3 langages (HTML / CSS / JavaScript) qui vous permettent de développer ce que l'on qualifie de ressources statiques.

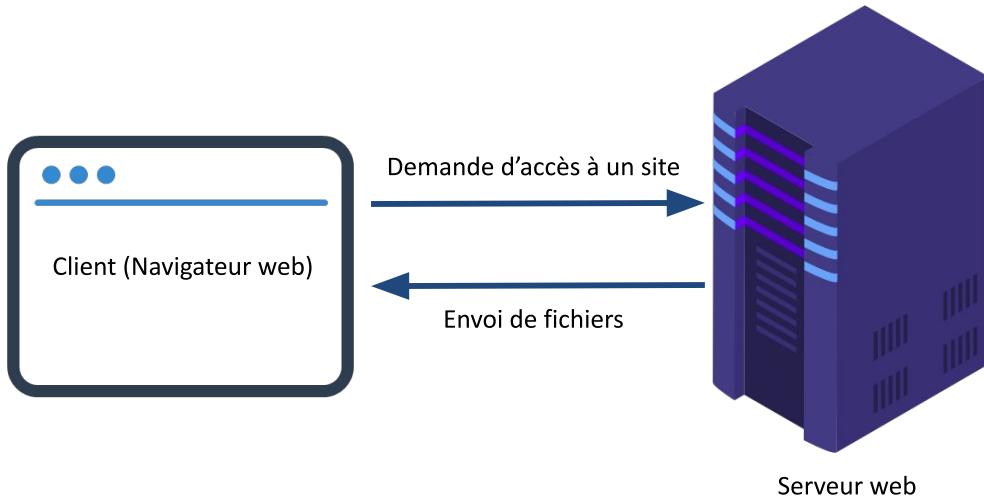
Une ressource statique est une ressource dont le contenu ne changera pas en fonction de la personne qui les récupère. Ce sont des fichiers destinés à être hébergés sur un serveur web.

Un serveur Web est un serveur sur lequel on installe une application dont le rôle est de mettre à disposition des ressources statiques.

Typiquement, un site web vitrine est constitué uniquement de ressources statiques.

Les 3 grandes applications du marché utilisées sont : Microsoft IIS , Apache HTTP Server, Nginx

6



Architecture client / serveur classique.

Un navigateur web interroge un serveur web afin de récupérer des ressources statiques.

7

## Point parcours

---

Dans le cas de figure du “site web”, ou des “ressources statiques”, il n’y a pas :

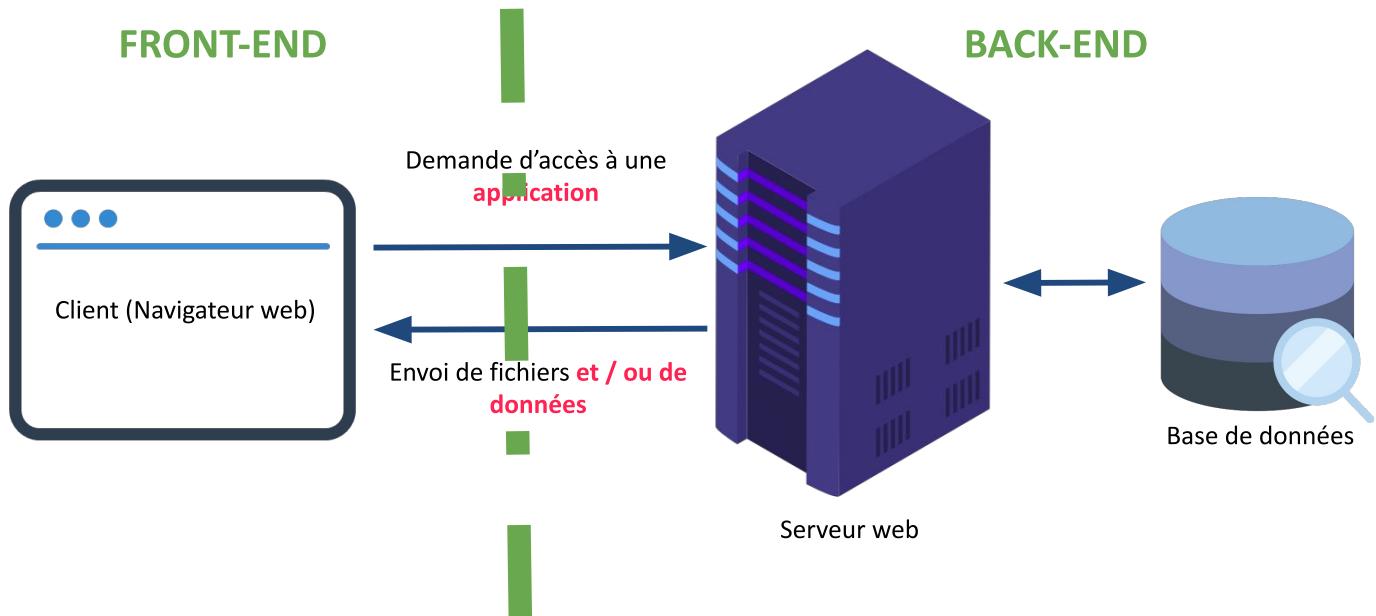
1. De sauvegarde de données en base de données
2. D'affichage différencié en fonction des utilisateurs
3. D'exécution de logiques métier spécifiques, en dehors de logiques d'affichage.

On est dans une logique **informative**.

A partir de maintenant, nous allons développer des **applications web**.

Là où un site web est informatif, une application web sera **interactive**.

Une application web interagit de manière directe ou indirecte avec une base de données.



9

## Point parcours

Dans le cas d'une application web, chaque utilisateur pourra disposer de sa propre interface.

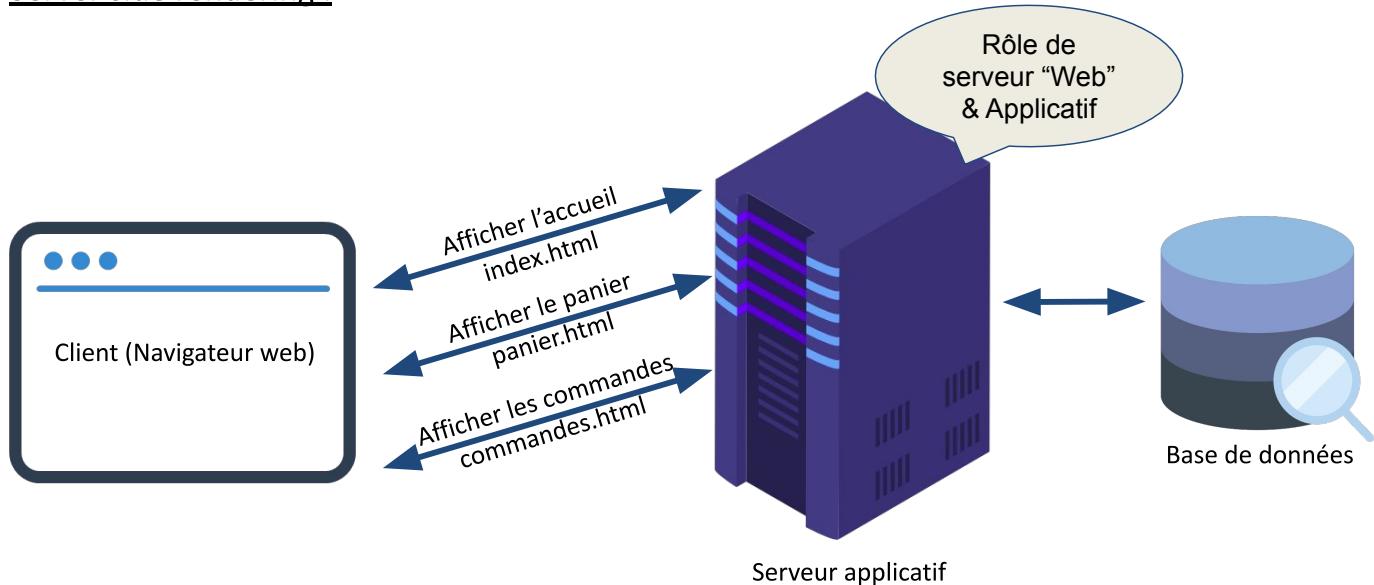
Par exemple, sur un site e-commerce :

1. Nous n'avons pas tous les mêmes suggestions de produits
2. Nous pouvons retrouver l'historique de nos commandes
3. Nous pouvons ajouter des produits dans un panier
4. Nous pouvons passer une commande et payer

Il existe deux grandes manières de développer une application web : le **server side rendering** ou le **client side rendering**.

# Point parcours

## Server side rendering :



11

## Point parcours

Dans le cas du server side rendering, nous allons utiliser un langage (par exemple, le PHP) pour générer des pages HTML au besoin en fonction des demandes des clients.

Nous allons donc générer des fichiers HTML **dynamiquement**, par opposition aux ressources statiques, dont le contenu est toujours le même.

Il existera toujours des ressources statiques, comme des images, du CSS, des polices de caractère... mais plus de fichier HTML "en dur".

Nos fichiers HTML seront générés à l'exécution sur notre serveur.

En terme de sémantique, on introduira la notion de serveur applicatif.

En effet, notre serveur aura désormais 2 rôles :

1. Héberger les ressources statiques (rôle du serveur web)
2. Exécuter le code permettant de générer les fichiers HTML au runtime (ie. à l'exécution) lorsque les navigateurs feront des demandes d'accès à certaines pages. On parle de serveur applicatif.

12

## Point parcours

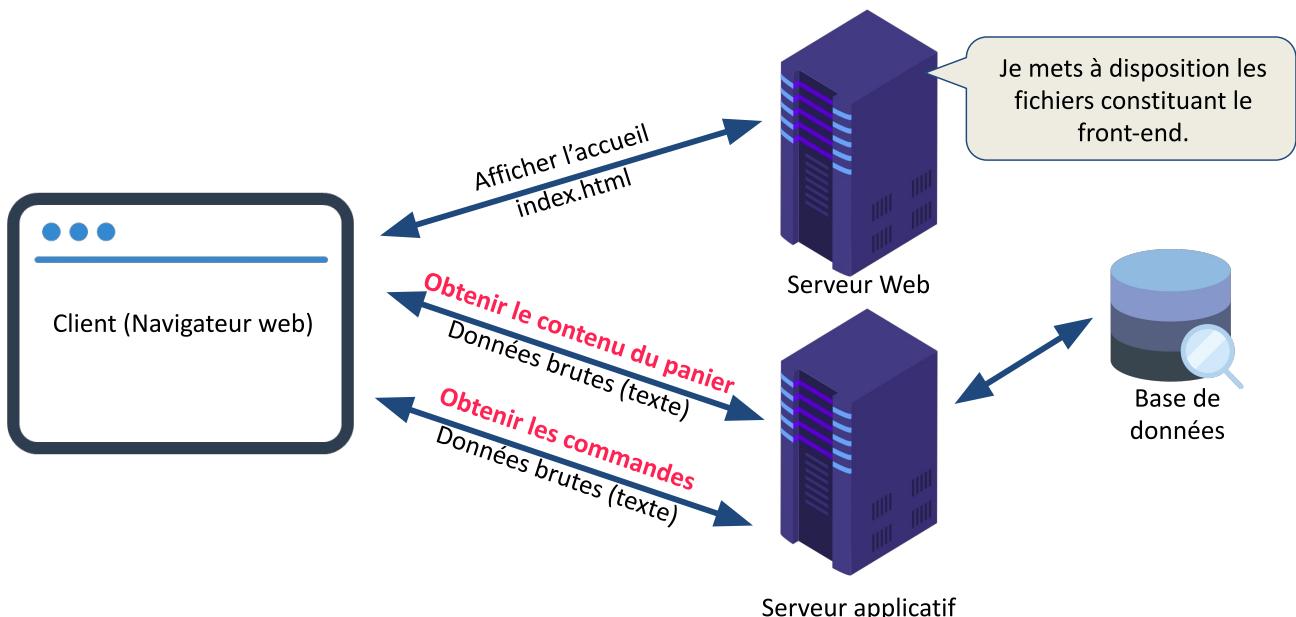
Dans le cas du server side rendering, notre application suit l'architecture “**multi-page application**” (MPA).

Cela signifie que nous avons développé le code nécessaire (en PHP ou dans un autre langage) pour générer dynamiquement des fichiers HTML en fonction des demandes des clients, et que chaque action menée par l'utilisateur conduira à recharger une nouvelle page.

13

## Point parcours

Client side rendering :



14

## Point parcours

---

Dans le cas du client side rendering, notre application suit l'architecture “**single-page application**” (SPA).

En effet le client ne télécharge qu'un seul fichier html : index.html

Ce fichier est autonome, et lorsque l'on navigue dans une SPA, on récupère des données brutes de la part du serveur.

C'est à dire qu'au lieu de nous envoyer de nouveaux fichiers HTML, le serveur web nous envoie uniquement des données non formatées.

C'est à la responsabilité du code JavaScript de récupérer ces données et de les traiter pour les afficher dans le DOM.

Le fichier index.html téléchargé est **autonome**.

15

## Point parcours

---

A l'issue de votre formation, vous serez initiés aux SPA et aux MPA :)

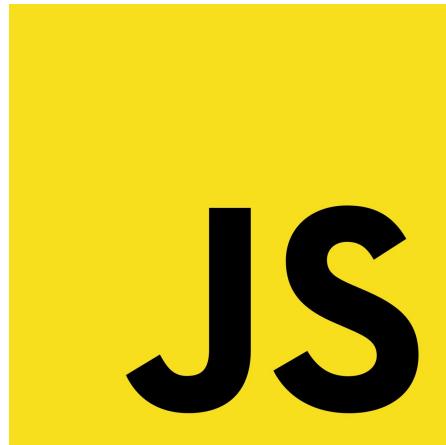
Lorsque l'on développe une SPA, on a une isolation forte entre le front et le back.

C'est moins vrai avec une MPA.

Chaque architecture a ses avantages et inconvénients.

Cette semaine, nous allons mettre en place une SPA.

16



17

## Introduction

---

Le JavaScript (JS) est un langage de script interprété.

Il suit le standard ECMAScript.

Deux grands cas d'utilisation :

1. Dans un navigateur web (client / front-end) : apporte du dynamisme aux pages
2. En dehors d'un navigateur :
  - a. Pour de l'exécution de scripts (comme un script cmd ou shell)
  - b. Côté serveur (back-end)

Ne pas confondre JavaScript avec JAVA !



JAVA est un langage différent de JavaScript à tout point de vue : son usage, sa sémantique...

D'ailleurs, il n'est pas interprétable nativement par un navigateur web !

18

# Introduction - ECMA

---

ECMA, historiquement “European Computer Manufacturers Association”

Nouveau nom :

“Ecma International - European association for standardizing information and communication systems”

ECMAScript est un standard ayant pour objet de spécifier le comportement de langages de scripts. Nom technique des spécifications : ECMA-262 et ECMA-402

Historiquement implémenté par ActionScript (langage utilisé par Flash), mais surtout par JS

9 versions majeures : ES1 > ES9

ES.Next est un nom de version dynamique qui fait toujours référence à la prochaine version d'ECMAScript.

19

**WEBFORCE**  
BE THE CHANGE  
Nicolas Amini-Lamy

## Introduction - ECMA

---

ECMAScript n'est pas la seule standardisation d'ECMA

ECMA-408 : Spécification du langage DART.

Initialement le DART avait pour vocation d'être une alternative au JS.

Finalement, dans un contexte web il est traduit en JS.



Flutter, le framework montant de Google,  
basé sur Dart

20

**WEBFORCE**  
BE THE CHANGE  
Nicolas Amini-Lamy

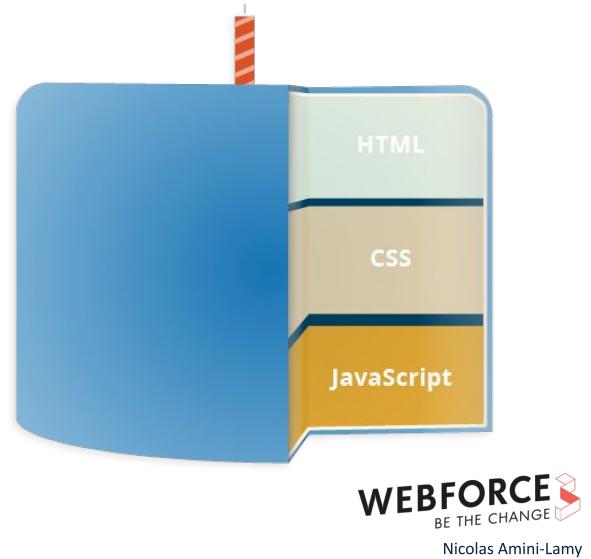
# Introduction - JavaScript dans un navigateur

Premier cas d'usage de JavaScript : dans un navigateur Web.

Permet d'ajouter du dynamisme à nos pages web. Dès lors qu'une page fait plus que simplement afficher du contenu statique (contenu identique / fixe quelque soit la personne qui se connecte ou les actions effectuées) JavaScript a de fortes chances d'être impliqué.

Exemples : un site e-commerce avec la possibilité d'ajouter des éléments dans un panier  
une carte interactive type maps  
des animations 2D/3D

JavaScript est souvent vu comme la 3ème couche après HTML (structure des pages) et CSS (mise en forme).



21

## Introduction - JavaScript dans un navigateur

Les navigateurs principaux (Edge, Firefox, Chrome, Opéra, Safari) supportent tous ES6

Chrome 58	Edge 14
Jan 2017	Aug 2016

Firefox 54	Safari 10	Opera 55
Mar 2017	Jul 2016	Aug 2018

22

# Introduction - JavaScript standalone

Autre cas d'usage de JavaScript : en dehors d'un navigateur.

Pour du scripting simple, par exemple l'envoi d'un mail à un instant T.

Pour des réalisations plus complexes, comme un back-end applicatif.

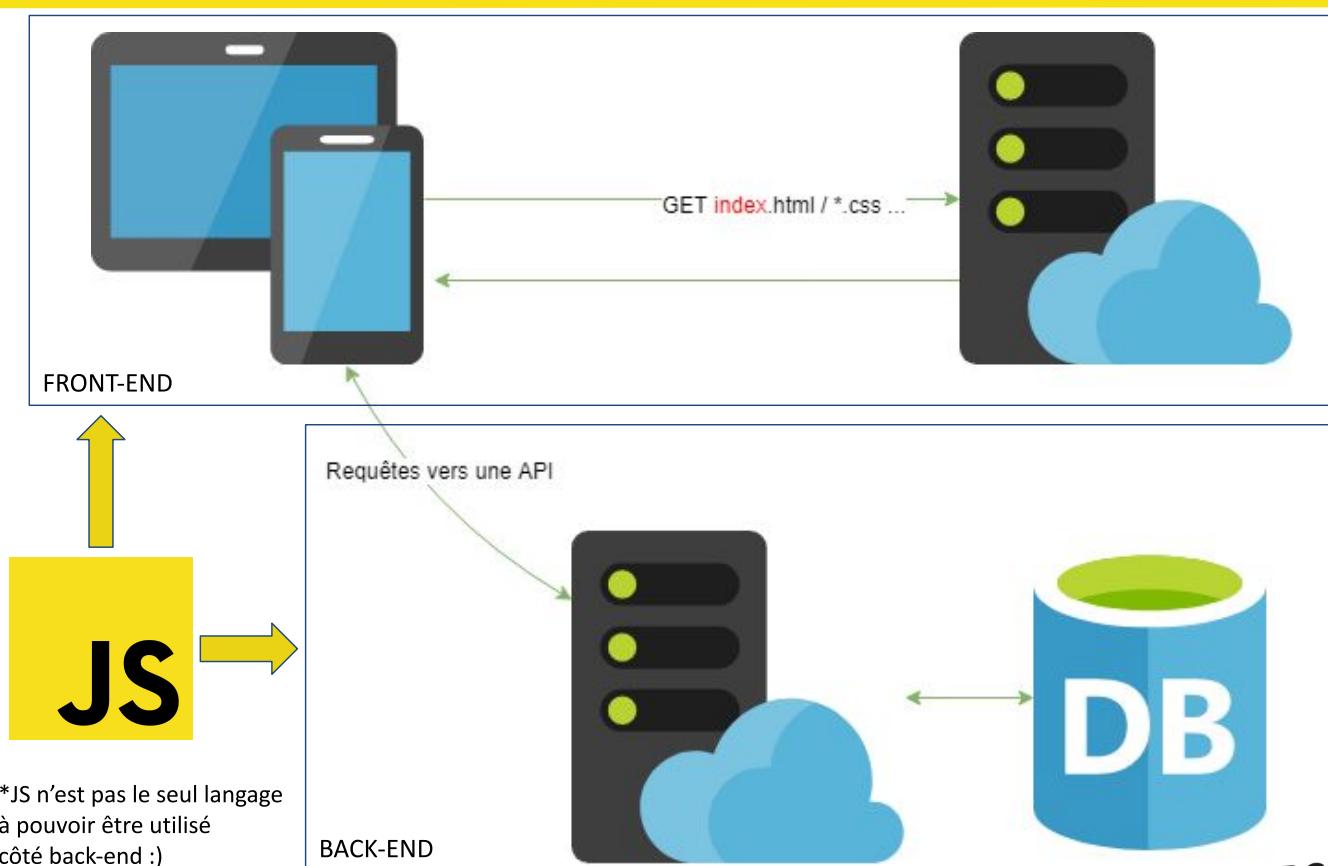


Node.JS, un programme permettant d'interpréter du JavaScript

23

WEBFORCE  
BE THE CHANGE  
Nicolas Amini-Lamy

## Introduction - Exemple d'architecture (SPA)



24

WEBFORCE  
BE THE CHANGE  
Nicolas Amini-Lamy

## Introduction - Installation Node.JS

---

Installez node.js : <https://nodejs.org/en/>

Créez un programme simple permettant d'afficher une liste d'étudiants d'une promotion dans la console, à partir d'un tableau.

Exécutez le via Node.JS avec la commande suivante : node <mon\_fichier.js>

25



## Les nouveautés ES6

---

{ ES6 JS }

26



# Nouveautés ES6 - Template Strings

En fonction des cas, utiliser l'opérateur + pour concaténer des chaînes peut être fastidieux...

Exemple :

```
1 let prenom = "Pierre";
2 let nom = "Dupont";
3 let nbMessages = 5;
4
5 let message = "Bonjour " + prenom + " " + nom + ". Vous avez " + nbMessages + " messages non lus.";
6
7 console.log(message);
```

Run >

> "Bonjour Pierre Dupont. Vous avez 5 messages non lus."

Reset

27



Nicolas Amini-Lamy

# Nouveautés ES6 - Template Strings

Depuis ES6, la notion de template string nous permet d'en finir avec l'enfer des + (surtout quand on doit juste ajouter un espace !) :

```
1 let prenom = "Pierre";
2 let nom = "Dupont";
3 let nbMessages = 5;
4
5 let message = `Bonjour ${prenom} ${nom}. Vous avez ${nbMessages} messages non lus.`;
6
7 console.log(message);
```

Run >

> "Bonjour Pierre Dupont. Vous avez 5 messages non lus."

Reset

28



Nicolas Amini-Lamy

# Nouveautés ES6 - Template Strings

Les template string permettent aussi de passer à la ligne :

```
1 let prenom = "Pierre";
2 let nom = "Dupont";
3 let nbMessages = 5;
4
5 let message = `Bonjour ${prenom} ${nom}.
6 Vous avez ${nbMessages} messages non lus.`;
7
8 console.log(message);
```

Run >

> "Bonjour Pierre Dupont.

Reset

Vous avez 5 messages non lus."

29

**WEBFORCE**  
BE THE CHANGE

Nicolas Amini-Lamy

# Nouveautés ES6 - Template Strings

Au delà de la simple substitution de variable, on peut également injecter des instructions JavaScript simples :

```
1 let prenom = "Pierre";
2 let nom = "Dupont";
3 let nbMessages = 5;
4
5 let message = `Bonjour ${prenom} ${nom}.
6 Vous avez ${++nbMessages} messages non lus.`;
7
8 console.log(message);
```

Run >

> "Bonjour Pierre Dupont.

Reset

Vous avez 6 messages non lus."

30

**WEBFORCE**  
BE THE CHANGE

Nicolas Amini-Lamy

## Nouveautés ES6 - Template Strings

On peut même utiliser une ternaire et une template string dans une template string (évitez cependant de créer trop de complexité) :

```
1 let prenom = "Pierre";
2 let nom = "Dupont";
3 let nbMessages = 5;
4
5 let message = `Bonjour ${prenom} ${nom}.
6 ${nbMessages != 0 ? `Vous avez ${nbMessages} messages non lus.` : "Vous n'avez pas de nouveaux messages."}`;
7
8 console.log(message);
```

Run >

Reset

> "Bonjour Pierre Dupont.  
Vous avez 5 messages non lus."

31



## Nouveautés ES6 - Classes

JavaScript est un langage qui permet de développer selon le paradigme de la programmation orientée objet.

Un paradigme est en quelque sorte une “manière” de programmer.

Vous avez déjà rencontré d’autres paradigmes :

1. Le paradigme impératif
2. Le paradigme déclaratif

32



# Paradigme impératif

Le paradigme de programmation impératif est l'un des paradigmes principaux que l'on rencontre en informatique.

Le principe est d'écrire des instructions les unes à la suite des autres.

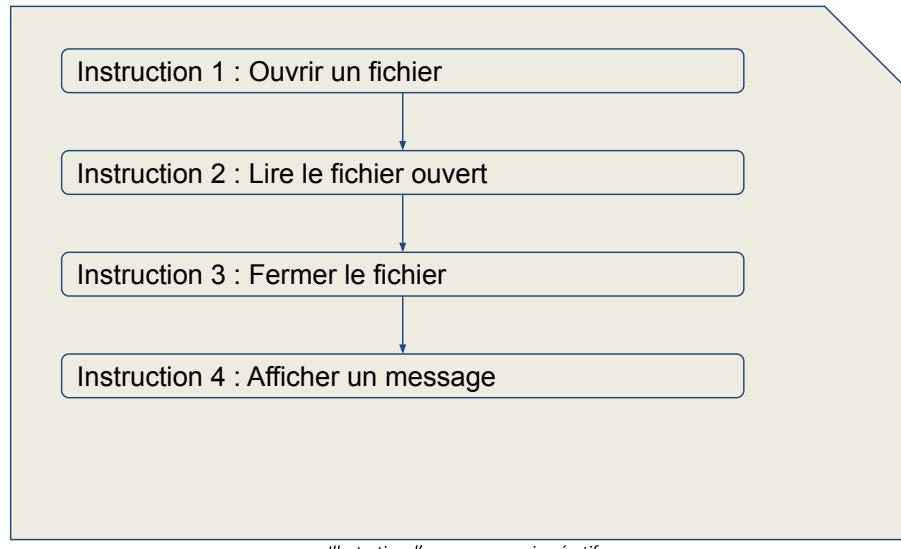


Illustration d'un programme impératif

33

## Paradigme déclaratif

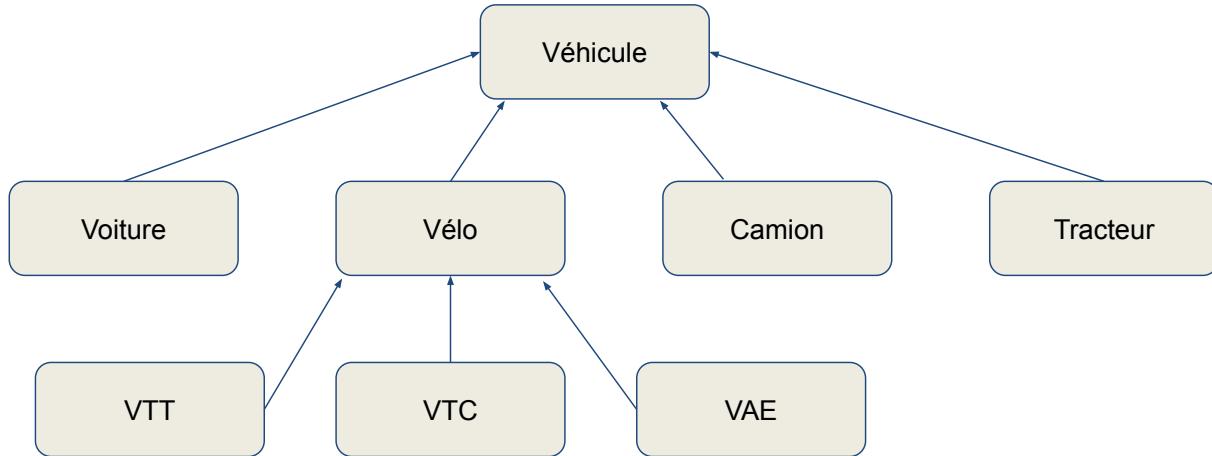
The screenshot shows the LaTeXiT application interface. At the top, there's a text input field containing a LaTeX integral expression:  $\int_1^{\infty} \frac{1}{x^2} dx = \left[ -\frac{1}{x} \right]_1^{\infty} = 1$ . Below the input field are several buttons: 'Auto', 'Align' (which is selected), 'Display', and 'Inline'. To the left of the input field is a font size selector set to '36,00 pt' and a color selector set to dark red. On the right side of the interface, there is a large block of HTML and CSS code, which is the generated output for the LaTeX expression.

```
<div *ngIf="!isUserInPatientDetailsView" class="containerFiltres d-flex al...<br><div class="btn-group filteringBlock" role="group"><button #displayUnlockedPatientsButton type="button" class="btn btn-seco...<button #displayLockedPatientsButton type="button" class="btn btn-seconda...</div><br><div class="input-group pl-2 filteringBlock"><div class="input-group-prepend"><span class="input-group-text" id="inputRecherche"><i class="fa fa-s...</div><input type="text" class="form-control" placeholder="Rechercher..." aria-labe...<aria-describedby="inputRecherche"/></div></div><div *ngIf="!isUserInPatientDetailsView" [ngClass]="{disabledContent : isP...<div class="card patientsList"><ngx-datatable class='material' [reorderable]="reorderable" [rowHeight]="getRowHeight()" [headerHeight]="50" ...</div>
```

HTML et LaTeX ; deux langages déclaratifs.

On décrit le “quoi” et non le “comment”.

34



35

## Nouveautés ES6 - Classes

Le principe de la POO est de modéliser notre application en “classes”.

Ces classes représentent des objets (pas forcément au sens physique) que l’on souhaite représenter dans notre application.

Lorsque l’on définit une classe, on détermine la structure et le comportement de tous les objets de cette classe.

Par exemple, un chien a 4 pattes. C'est une de ses caractéristiques.

=> ses 4 pattes font donc partie de sa **structure**.

Un chien peut aboyer : on lui ajoutera une méthode permettant de le faire.

=> cette méthode fera partie de son **comportement**.

36

# Nouveautés ES6 - Classes

## Vocabulaire élémentaire en POO :

Classe : Créer une classe, c'est créer un nouveau type de données. C'est une entité regroupant des variables (attributs) et des fonctions (comportements) que l'on souhaite associer car ils ont du sens entre eux.

Par exemple, tous les rectangles ont une longueur, une largeur. On peut également calculer leur aire. On peut donc créer une classe Rectangle.

Instance / Objet : une instance d'une classe est un objet construit à l'aide de la classe.

On peut avoir 10 rectangles, ils auront tous une longueur / largeur et on peut calculer l'aire. On va donc utiliser notre classe Rectangle.

Propriété / Attribut : Element de structure. Exemple : une personne peut avoir un nom, prénom, un sexe, une couleur de cheveux...

Les propriétés d'un rectangle sont : sa longueur, sa largeur.

37



# Nouveautés ES6 - Classes

Méthode : action applicable à un objet. Une méthode permet d'implémenter un comportement.

On peut créer une méthode permettant de calculer l'aire d'un rectangle.  
Le calcul sera identique quelque soit le rectangle.

Héritage : Une classe peut hériter d'une autre classe. On dit qu'elle la **spécialise**. Par exemple, une classe Chat peut hériter de la classe Animal.

Lorsqu'une classe hérite d'une autre classe, elle hérite des caractéristiques et du comportement de la classe héritée. On parle de classes mères / filles.

Parfois, on utilise le verbe "étendre" en lieu et place d'hériter".

Constructeur : Le constructeur est une méthode particulière, obligatoire dans toute classe, qui permet d'initialiser les attributs d'une instance d'une classe.

38



## Nouveautés ES6 - Classes

```
1 class Animal {
2   constructor(categorie) {
3     this.categorie = categorie;
4   }
5   direBonjour() {
6     console.log("Bonjour, je suis un animal et un " + this.categorie);
7   }
8 }
9
10 class Chat extends Animal {
11   constructor(nom) {
12     super("Vertébré");
13     this.nom = nom;
14   }
15 }
16
17 let monChat = new Chat("Felix");
18 console.log(monChat.nom);
19 console.log(monChat.categorie);
20 monChat.direBonjour();
```

39



## Nouveautés ES6 - Classes

La notion de Classe existe en JavaScript via ES6.

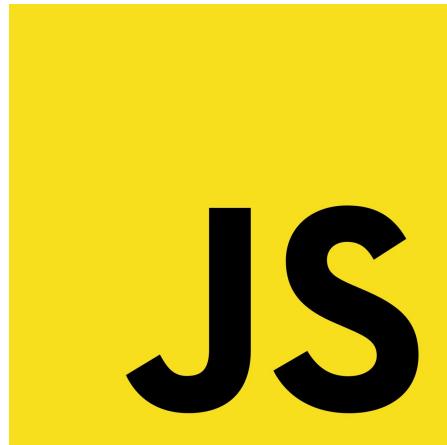
Il s'agit d'un syntaxic sugar pour écrire de la POO en JavaScript sans passer par son fonctionnement natif à base de prototypes.

Pour instancier une classe, il faut utiliser le mot clef **new**.

Ce mot clef new est un type d'appel particulier sur les fonctions.

40





Retour sur les fonctions

41

## Les fonctions - Déclaration

---

Le principe d'une fonction est le même quelque soit le langage : regrouper des instructions qui ont du sens ensemble et qui sont réutilisables, afin de gagner en lisibilité et en maintenabilité.

En JavaScript, une fonction peut se créer de différentes manières.

La syntaxe “classique” est la suivante :

```
1 function additionnerDeuxNombres(nombre1, nombre2) {  
2     return nombre1 + nombre2;  
3 }  
4  
5 console.log(additionnerDeuxNombres(10, 20));
```

42

# Les fonctions - Portées de variables

Dans une fonction il est possible d'appeler des variables déclarées précédemment.

```
1 var nbDecimales = 2;
2
3 function additionnerDeuxNombres(nombre1, nombre2) {
4   return (nombre1 + nombre2).toFixed(nbDecimales);
5 }
6
7 console.log(additionnerDeuxNombres(10.34, 20.45643));
```

43

**WEBFORCE**  
BE THE CHANGE  
Nicolas Amini-Lamy

# Les fonctions - Portées de variables

L'accès aux variables “parentes” est opérationnel même si la fonction parente a terminé son exécution :

```
1 function uneFonction() {
2   let variableFonctionParente = "Démonstration";
3
4   function fonctionEnfant() {
5     console.log(variableFonctionParente + " des closures");
6   }
7
8   setTimeout(fonctionEnfant, 3000); →
9 }
10
11 uneFonction();
```

Run >

Reset

> "Démonstration des closures"

fonctionEnfant sera exécutée “dans” 3 secondes, alors que l'exécution de “uneFonction” sera terminée depuis plusieurs secondes.

Pour autant, le scope de variable est conservé et la fonction enfant peut toujours accéder à la variable du scope parent.

C'est le principe des closures (“fermetures”).

44

**WEBFORCE**  
BE THE CHANGE  
Nicolas Amini-Lamy

# Les fonctions - Invocations

Il existe plusieurs manières d'invoquer une fonction en JavaScript :

- ◆ comme une fonction (oui... )
- ◆ comme une méthode
- ◆ comme un constructeur
- ◆ avec apply() ou call()

Les fonctions sont un concept clef du langage et certaines spécificités sont souvent mal comprises lorsque l'on vient d'autres langages (comme JAVA !).

Lorsque l'on appelle une fonction (quelque soit la méthode) ; le moteur JavaScript ajoute automatiquement (et implicitement) 2 paramètres : **this** et **arguments**.

45

## Les fonctions - Invocations

Appeler une fonction... “comme une fonction”.

C'est le premier type d'appels, le plus simple.

The screenshot shows a code editor on the left with the following JavaScript code:

```
1 function exemple(param) {  
2   console.log(this, arguments);  
3 }  
4  
5 exemple("test");
```

Two yellow arrows point from the text "Contexte" de la fonction and "Paramètres de la fonction" to the "this" and "arguments" parameters in the code respectively. To the right of the code is a browser developer tools interface showing a "Run >" button and a log output: > [object Window] Object { 0: "test" }

Lorsque l'on appelle une fonction de cette manière, la variable **this** correspond à l'objet **window** (dans un contexte web) ou **global** (contexte node).

46

# Les fonctions - Invocations

Seconde manière d'appeler une fonction : comme une méthode.

L'idée est d'attacher une fonction à un objet ; et d'appeler cette fonction sur l'objet.

C'est de cette manière que l'on peut faire de la programmation orientée objet en JavaScript.

```
1 var robot = {  
2   prenom: "Henri"  
3 };  
4  
5 robot.direBonjour = function() {  
6   console.log(this);  
7   console.log("Bonjour");  
8 }  
9  
10 robot.direBonjour();
```

Run >

Reset

```
> Object { prenom: "Henri", direBonjour: function() {  
    console.log(this);  
    console.log("Bonjour");  
} }  
> "Bonjour"
```

47

Dans ce cas, la variable `this` correspond à l'objet "robot".

On retrouve le sens classique de la variable "this" tel qu'on peut le rencontrer dans des langages comme JAVA.

# Les fonctions - Invocations

Troisième manière d'appeler une fonction : comme un constructeur avec le mot clef `new`.

```
1 function exemple(param) {  
2   console.log(this, arguments);  
3 }  
4  
5 exemple("test");  
6  
7 new exemple("test");
```

Run >

Reset

```
> [object Window] Object { 0: "test" }  
> [object Object] Object { 0: "test" }
```

Lorsque l'on utilise `new` :

- ◆ un nouvel objet (vide) est créé
- ◆ cet objet est passé au constructeur et correspond au `this`
- ◆ en l'absence d'un return dans le constructeur, ce nouvel objet est renvoyé implicitement.

48

# Les fonctions - Invocations

L'invocation par constructeur est pratique pour définir des objets en POO.

Sans utiliser cette méthode, pour créer deux objets de la même classe, on devrait écrire :

```
1 function direBonjour() {  
2   console.log("Bonjour");  
3 };  
4  
5 var robot = {  
6   prenom: "Henri",  
7   direBonjour : direBonjour  
8 };  
9  
10 var robot2 = {  
11   prenom: "Jacques",  
12   direBonjour : direBonjour  
13 };
```

```
1 function Robot(prenom) {  
2   this.prenom = prenom;  
3   this.direBonjour = function() {  
4     console.log("Bonjour");  
5   };  
6 }  
7  
8 var henri = new Robot("Henri");  
9 var jacques = new Robot("Jacques");  
10  
11 henri.direBonjour();  
12 jacques.direBonjour();
```

Run >

Reset

> "Bonjour"

> "Bonjour"

49

**WEBFORCE**  
BE THE CHANGE

Nicolas Amini-Lamy

## Les fonctions - Invocations

Une fonction dispose de ses propres méthodes. En effet, toute fonction en JavaScript est un objet Function. Cet objet permet d'utiliser certaines méthodes dont apply() et call().

La dernière manière d'invoquer une fonction est de faire appel à apply() et call().

Apply comme call permettent d'appeler une fonction tout en surchargeant son contexte, c'est à dire la valeur de this.

La différence résulte dans la signature de la fonction au niveau du passage des arguments (via un tableau ou non).

```
1 function direBonjour(nomInterlocuteur) {  
2   console.log(this);  
3   console.log("Bonjour " + nomInterlocuteur);  
4 }  
5  
6 direBonjour.apply({}, ["Jean"]);  
7 direBonjour.call({}, "Jean");
```

Run >

Reset

```
> Object { }  
> "Bonjour Jean"  
> Object { }  
> "Bonjour Jean"
```

50

**WEBFORCE**  
BE THE CHANGE

Nicolas Amini-Lamy