

# Les limites et les répercussions de l'industrialisation du développement JavaScript dans le Web



Clever Age  
100% DIGITAL

Fabien Salles



# Sommaire

SOMMAIRE.....	2
REMERCIEMENTS .....	3
PREAMBULE.....	4
INTRODUCTION .....	5
I. LE CONTEXTE .....	6
1.1. HISTORIQUE DU LANGAGE.....	6
1.2. DES BESOINS DE PLUS EN PLUS IMPORTANTS.....	8
1.3. L'APPARITION DE NOUVELLES PROBLEMATIQUES .....	9
II. LES FRAMEWORKS ET COMPILATEURS .....	10
2.1. L'UTILITE DES FRAMEWORKS JS .....	10
2.2. COMMENT CHOISIR SON FRAMEWORK JS ?.....	13
2.3. COMPARATIF DE FRAMEWORKS .....	16
2.4. LES COMPILATEURS.....	33
III. LA REVOLUTION APPARUE AVEC NODE.JS .....	48
3.1. LA REPONSE A DE REELS BESOINS.....	48
3.2. LES FRAMEWORKS COTE SERVEUR.....	63
3.3. LA RECHERCHE DE LA QUALITE .....	66
CONCLUSION .....	77
GLOSSAIRE.....	79
WEBOGRAPHIE .....	82
LE CONTEXTE .....	82
LES FRAMEWORKS ET COMPILATEURS.....	82
LA REVOLUTION APPARUE AVEC NODE.JS .....	85
TABLES DES MATIÈRES .....	89

# Remerciements

Je souhaite remercier dans un premier temps, toute l'équipe pédagogique responsable du master MIAGE à l'université Paris Ouest Nanterre La Défense, pour avoir assuré la partie théorique de celle-ci.

Je remercie également mon tuteur enseignant, M. François Delbot, ainsi que l'ensemble des collaborateurs de la société Clever Age qui m'ont accompagné durant mon Master. Je pense en particulier à mon lead développer pour son expertise, M. Florian Lonqueu-Brochard, ainsi que mon maître d'apprentissage et mon chef de projet, Mme Aude Goetz et M. Jean-Yves Gastaud, pour la confiance et l'autonomie qu'ils m'ont accordées.

Enfin, je tiens à remercier les personnes qui ont participé de près ou de loin à la relecture de ce mémoire :

- M. Boris Schapira : Consultant, formateur JavaScript chez Clever Age Bordeaux et professeur à Ingesup Bordeaux
- M. Matthias Dugué : Consultant et formateur JavaScript chez Clever Age Paris
- M. Guillaume Macaire : Consultant et responsable du pôle expertise chez Clever Age
- M. Jacques Leroux : Chef de projet chez Clever Age
- M. Samuel Bouchet : Ingénieur chez Clever Age
- Mme Aude Goetz
- M. Jean-Yves Gastaud

# Préambule

Dans le cadre de ma dernière année de Master MIAGE, je devais réaliser un mémoire qui rentre à la fois en adéquation avec ma formation et avec mon travail effectué durant mon apprentissage dans l'entreprise Clever Age.

Clever Age est une société par actions simplifiée (SAS) créée en 2001. Elle comporte actuellement 115 salariés et couvre l'intégralité des domaines liés au web. Elle dispose de 6 agences réparties en France et à l'étranger. Il y en a une à Paris, Nantes, Lyon, Bordeaux, Montpellier, en Suisse et une nouvelle qui vient d'ouvrir à Hong Kong.

Je termine actuellement ma 2ème année d'apprentissage chez eux en tant que développeur PHP. Ma principale mission était de maintenir et faire évoluer une plate-forme de gestion et de relation client pour Volkswagen qui utilisait la version 1.4 du Framework Symfony<sup>1</sup>.

Le mémoire de ma première année de Master portait sur l'amélioration continue appliquée à ce projet. Nous entamons actuellement une refonte de l'outil afin de passer sur la version 2.5 du Framework et appliquer des concepts que j'avais pu évoquer comme l'intégration continue, le BDD<sup>2</sup> ou encore le monitoring<sup>3</sup>. Nous allons même encore plus loin en utilisant des approches encore jeunes dans la communauté PHP tels que le DDD<sup>4</sup> (Domain Driven Design) ou encore le CQRS<sup>5</sup> (Command Query Responsibility Segregation).

Ce mémoire sera néanmoins quelque peu différent car il ne reflétera pas mon travail effectué en entreprise mais aura pour vocation de faire un tour d'horizon dans un domaine en perpétuelle évolution qu'est le monde du JavaScript (JS). N'ayant pratiquement pas fait de JavaScript durant mes deux dernières années d'apprentissage, j'ai décidé de profiter de ce mémoire pour approfondir mes connaissances dans ce langage. Durant ma Licence et la réalisation du projet de fin d'année de Master 2, j'ai dû faire face à certaines problématiques qui provenaient de mon manque d'expérience et de connaissance dans le domaine. J'ai donc choisi de parler de l'industrialisation<sup>6</sup> du JavaScript dans le Web et ainsi proposer une aide dans les différents choix qui s'offre à nous pour réaliser un projet Front<sup>7</sup>.

---

<sup>1</sup> **Symfony** : Framework MVC PHP créé par la société française Sensio

<sup>2</sup> **BDD** : Behavior Driven Development. C'est une méthode agile dirigeant le développement par les spécifications.

<sup>3</sup> **Monitoring** : technique de surveillance et de mesure d'un système informatique

<sup>4</sup> **DDD** : Domain-Driven Design. Approche de développement se focalisant sur les contraintes métier (le domaine) plutôt que les contraintes techniques.

<sup>5</sup> **CQRS** : Command Query Responsibility Segregation. Principe reposant sur le design pattern command (voir glossaire) afin d'effectuer une séparation entre les actions de lecture et d'écriture.

<sup>6</sup> **Industrialisation** : dans le Web, l'industrialisation représente différentes techniques permettant de structurer une application afin de la rendre plus stable, et automatiser certaines tâches pour augmenter la productivité des développeurs.

<sup>7</sup> **Front** : représente le client. Un projet Front est donc une application s'exécutant du côté du client plutôt que du serveur.

# Introduction

Le JavaScript est un langage interprété orienté objet à prototype<sup>8</sup>. Permissif et à typage faible<sup>9</sup>, Il est très apprécié par les développeurs avec de faibles connaissances en programmation car il permet d'arriver à un résultat facilement et rapidement. Ces mêmes caractéristiques font que certaines personnes le méprisent, le jugeant trop simple, accessible et trop immature pour concevoir des applications complexes. Il est certes simple de l'utiliser mais il est très dur de le faire de la bonne manière.

Son approche objet par prototype, parfois difficile à appréhender, fait du JavaScript un langage incompris par beaucoup de monde. Il est pourtant devenu incontournable pour améliorer l'expérience utilisateur sur le web. Il a permis l'émergence de concepts tels que la programmation fonctionnelle<sup>10</sup>, événementielle<sup>11</sup> ou encore asynchrone<sup>12</sup> et il fait maintenant partie des enjeux majeurs permettant d'améliorer la visibilité des entreprises sur Internet grâce à la différenciation par l'innovation.

Le JavaScript ne se limite pas à une simple interprétation par les navigateurs. Il est possible dans faire côté serveur, sur des systèmes d'exploitation, sur mobile, sur des circuits imprimés et demain, sur des objets connectés. Il devient donc de plus en plus important d'industrialiser son développement afin de concevoir des applications robustes, maintenables et évolutives.

Quelles sont les situations où nous pouvons utiliser le JavaScript ? Quelles sont les nouvelles problématiques que nous rencontrons et les solutions pour y répondre ? Vers où allons-nous et quel est l'avenir de ce langage ? Autant de questions auxquelles ce mémoire va tenter de répondre en regroupant l'ensemble de ces interrogations en une principale : **“Quelles sont les limites et les répercussions de l'industrialisation du développement JavaScript dans le Web ?”**

Pour répondre à cette problématique, nous situerons dans un premier temps son contexte. Nous parlerons ensuite des différents compilateurs et Frameworks populaires. Enfin, nous finirons par voir l'utilité de node.js permettant d'exécuter du JavaScript côté serveur et l'émergence de nouveaux outils qui deviennent de plus en plus indispensables.

---

<sup>8</sup> **Prototype** : concept objet n'effectuant aucune différence entre les classes et les instances de classes (les objets).

<sup>9</sup> **Typage faible** : La restriction sur les types de données est faible et le langage peut les convertir automatiquement.

<sup>10</sup> **Programmation fonctionnelle** : paradigme renforçant l'utilisation de fonctions et de structures de données immuables (voir glossaire) plutôt que d'objets et de classes.

<sup>11</sup> **Programmation événementielle** : programmation fondé sur les événements.

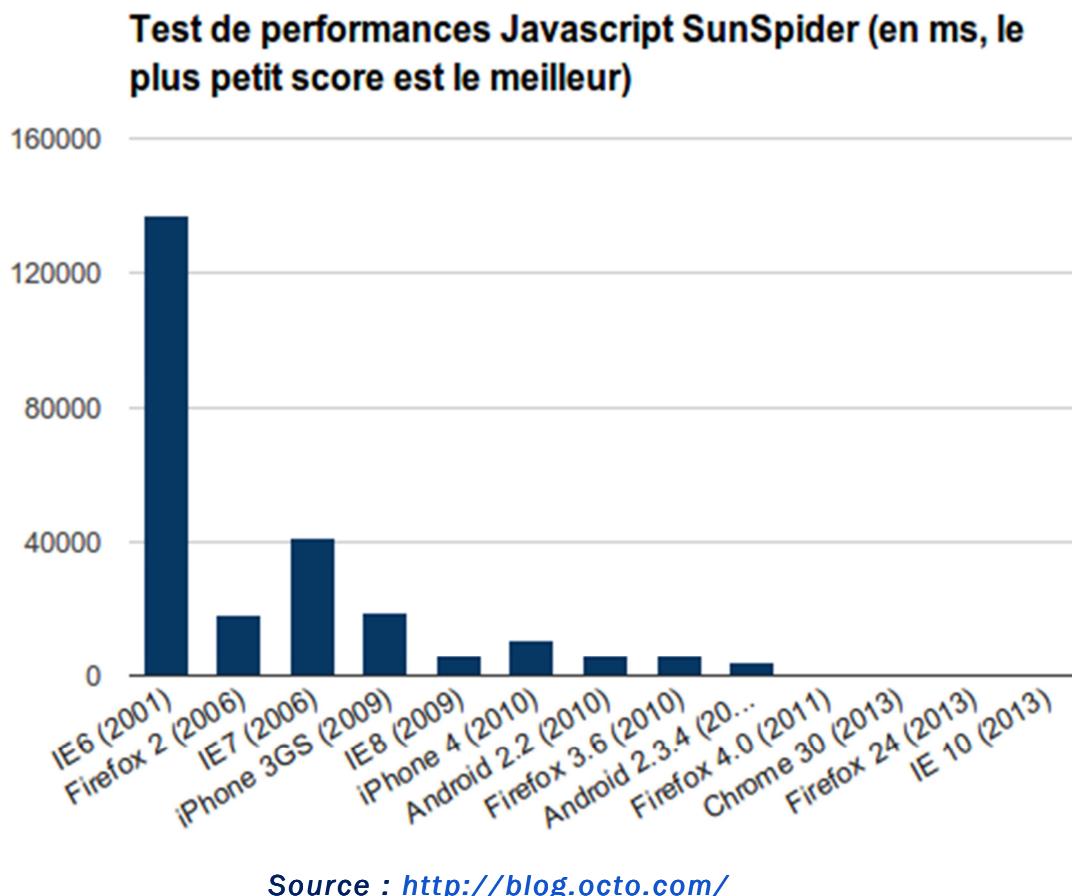
<sup>12</sup> **Programmation asynchrone** : programmation non synchrone (non bloquante pour l'utilisateur).

# I. Le contexte

## 1.1. Historique du langage

JavaScript a été créé en l'espace de 10 jours au mois de mai 1995 par Brendan Eich qui travaillait à l'époque chez Netscape (devenu maintenant Mozilla). L'objectif était de remplacer l'utilisation des applets Java afin de s'ouvrir à plus de monde et en particulier aux designers. Sa popularité grandit et d'autres navigateurs que Netscape Navigator commencèrent à l'utiliser. C'est alors que JavaScript intégra l'European Computer Manufacturers Association (ECMA) afin de proposer des standards à tous les acteurs qui l'implémaient en créant un unique langage (ECMAScript).

Seulement les navigateurs ne se limitèrent pas à ces standards et une "guerre" se créa afin de rester compétitif et gagner des parts de marché. Un des facteurs principaux qui a notamment permis à JavaScript d'évoluer est l'amélioration des capacités d'interprétation et des performances des navigateurs.



En plus des standards de l'ECMA, les navigateurs recherchaient et recherchent toujours des facteurs de différenciation afin d'enrichir les interactions avec les utilisateurs. C'est ainsi qu'apparurent en 2005 des courants technologiques comme l'utilisation d'AJAX<sup>13</sup>.

<sup>13</sup> **AJAX** : Asynchronous JavaScript and XML. Ensemble de technologies regroupant le JavaScript, le DOM et le XMLHttpRequest afin de faciliter l'utilisation des applications riches.

A ce moment-là, des développeurs Java ou encore PHP se mirent à l'utiliser. Des librairies comme Prototype, Mootools ou encore jQuery, ainsi que des outils de développement et de débogage virent le jour afin d'augmenter la productivité et faciliter le développement. Faire du JavaScript était devenu beaucoup plus simple et même une personne néophyte pouvait s'amuser avec. jQuery s'était imposé comme la librairie de référence, augmentant encore plus la popularité et l'utilisation du JavaScript, mais également le nombre de développeurs mécontents.

Beaucoup de personnes faisaient du jQuery avant de faire du JavaScript, ce qui avait pour conséquence d'arriver rapidement à un résultat mais rendait également le code difficilement compréhensible et maintenable par la suite. Il y avait d'un côté les personnes qui utilisaient le JS pour interagir avec le design et de l'autre ceux avec des profils Backend qui interagissaient avec les données. Les développements étaient donc différents et rentraient souvent en conflits.

Des concurrents existaient tels que Flash d'Adobe ou encore SilverLight de Microsoft et ont essayé de détrôner en vain JavaScript. Le fait de ne pas directement être interprété par les navigateurs a eu comme répercussions que ces alternatives n'ont pas tenu le rythme sur les différentes évolutions qui sont survenues. L'évènement déclencheur de cette chute a été en partie dû à l'arrivée d'HTML5<sup>14</sup> et des Framework en 2009.

L'HTML5 apporta de nombreuses API<sup>15</sup> gérant nativement la géolocalisation, le mode hors-ligne, le drag-and-drop, les vidéos, l'audio, les animations 3D ou encore pleins d'autres fonctionnalités permettant d'interagir avec les utilisateurs, les ordinateurs et les mobiles. Les Frameworks JS quant à eux, ont permis de mieux structurer le code et de proposer des bases solides pour des applications ambitieuses. Nous reviendrons plus en détails sur le fonctionnement de ces derniers.

D'autres solutions comme les compilateurs sont encore d'actualité afin de simplifier le développement, la maintenabilité et l'évolutivité du JavaScript. Nous verrons par la suite les problématiques que certains d'entre eux résolvent.

---

<sup>14</sup> **HTML5** : HyperText Markup Language 5. Dernière version majeure d'HTML comprenant de nombreuses API.

<sup>15</sup> **API** : pour Application Programming Interface. Ensemble de classes et méthodes servant de façade afin d'interagir avec des données

## 1.2. Des besoins de plus en plus importants

De la même manière que le JavaScript, les besoins des clients ont évolué au fil du temps ouvrant la porte à de nouvelles professions qui répondent à de nouvelles problématiques.

Le premier point que l'on peut citer est l'expérience utilisateur (ou UX pour User eXperience en anglais). L'UX est le fait de se mettre à la place de l'utilisateur afin de répondre au mieux à ses attentes. Avant, on s'arrêtait à l'ergonomie et à l'accessibilité. Maintenant, on parle d'UX et de design avec de nouveaux postes comme l'UX Designer, le Visual Designer, le Web Designer... On pourrait se dire que ces dénominations sont juste un effet de mode et qu'elles ne sont pas véritablement utiles. Toutefois, elles répondent à un réel besoin qui devient de plus en plus compliqué à prendre en compte. L'expérience utilisateur est devenue un élément primordial pour les entreprises souhaitant s'imposer sur Internet, souhaitant se différencier des concurrents et souhaitant surtout vendre et prospérer dans le Web.

Le rapport avec le JavaScript est que ce dernier contribue énormément à améliorer l'expérience utilisateur. Prenons l'exemple des « applications web » désignant des logiciels hébergés sur Internet, accessibles via un navigateur Web et rivalisant avec des logiciels installés sur un ordinateur. On pourrait citer les Google Apps, avec Google Drive, Google Document, Gmail et bien d'autres encore. Ces applications montrent clairement l'étendue des possibilités que nous offre aujourd'hui le JavaScript et à quel point l'expérience utilisateur a été améliorée ces dernières années.

A côté de ces applications web, on peut également parler de « sites expériences ». Les sites expériences sont encore plus parlants que les applications Web pour se rendre compte de l'importance de l'UX et du JavaScript. Comme on l'a déjà dit le but est de se différencier, de faire rêver et plonger l'utilisateur dans un cadre défini avec un but défini. Par exemple Les sites de commerces permettant de personnaliser les produits achetés sont des sites expériences. Il y a aussi ceux qui vont promouvoir quelque chose, un film ou encore une série. The Hobbit<sup>16</sup> est un bon exemple d'expérience immersive.

On pourrait penser qu'un simple site Internet ne nécessite pas forcément d'UX et de Framework JavaScript mais un simple site avec un design adapté selon le périphérique (ordinateur, tablette, mobile..) nécessite de mettre place certains concepts, méthodologies et outils afin de répondre au mieux au besoin du client et surtout à ceux qui vont utiliser le site.

---

<sup>16</sup> Application disponible ici : <http://middle-earth.thehobbit.com/>

### 1.3. L'apparition de nouvelles problématiques

Maintenant on arrive à comprendre que l'industrialisation du code devient de plus en plus obligatoire et que nous devons adopter des pratiques afin de produire du code de qualité. Il reste cependant un nombre important de points à éclaircir.

Commençons par les Frameworks : lequel devons-nous choisir ? Dans quel cas de figure ? Le JS est un des langages qui subit le plus d'évolutions et « d'effet de mode ». Il faut constamment se tenir au courant des dernières nouveautés, comprendre leurs utilités, prendre du recul et trouver le moyen de les intégrer à des projets d'envergure si celles-ci sont vraiment intéressantes.

L'apparition des Frameworks a eu beaucoup de répercussions sur nos développements. Le premier est que nous effectuons de plus en plus de séparation dans nos développements et qu'il devient maintenant inenvisageable de développer uniquement dans un seul fichier. Nous nous retrouvons donc avec de multiples fichiers et des conséquences sur le chargement et l'exécution des pages web. Comment pouvons-nous améliorer ces performances ?

Le JavaScript étant exécuté par le biais du navigateur, tout le contenu qu'il utilise n'est pas référençable par les moteurs de recherche. Il faut donc également penser à cette problématique et mettre en place, dès le début du développement du projet, des solutions pour les éléments importants qui doivent être référencés.

Un projet un minimum ambitieux qui nécessite beaucoup de JavaScript doit faire face à de nombreuses autres problématiques. Il y a par exemple la gestion des dépendances entre les librairies externes ou encore l'utilisation de compilateurs et d'outils quels qu'ils soient permettant d'industrialiser le développement et de gagner en productivité, maintenabilité et évolutivité.

Il y a donc beaucoup de questions auxquelles ce mémoire (bien que non exhaustif) va essayer de répondre. La communauté du JavaScript est tellement grande que les possibilités qui s'offre à nous deviennent de plus en plus nombreuses et les choix de plus en plus difficiles à faire. Nous allons synthétiser certains points, en approfondir d'autres et passer en revue un grand nombre d'éléments autour de l'industrialisation du développement JavaScript dans le web.

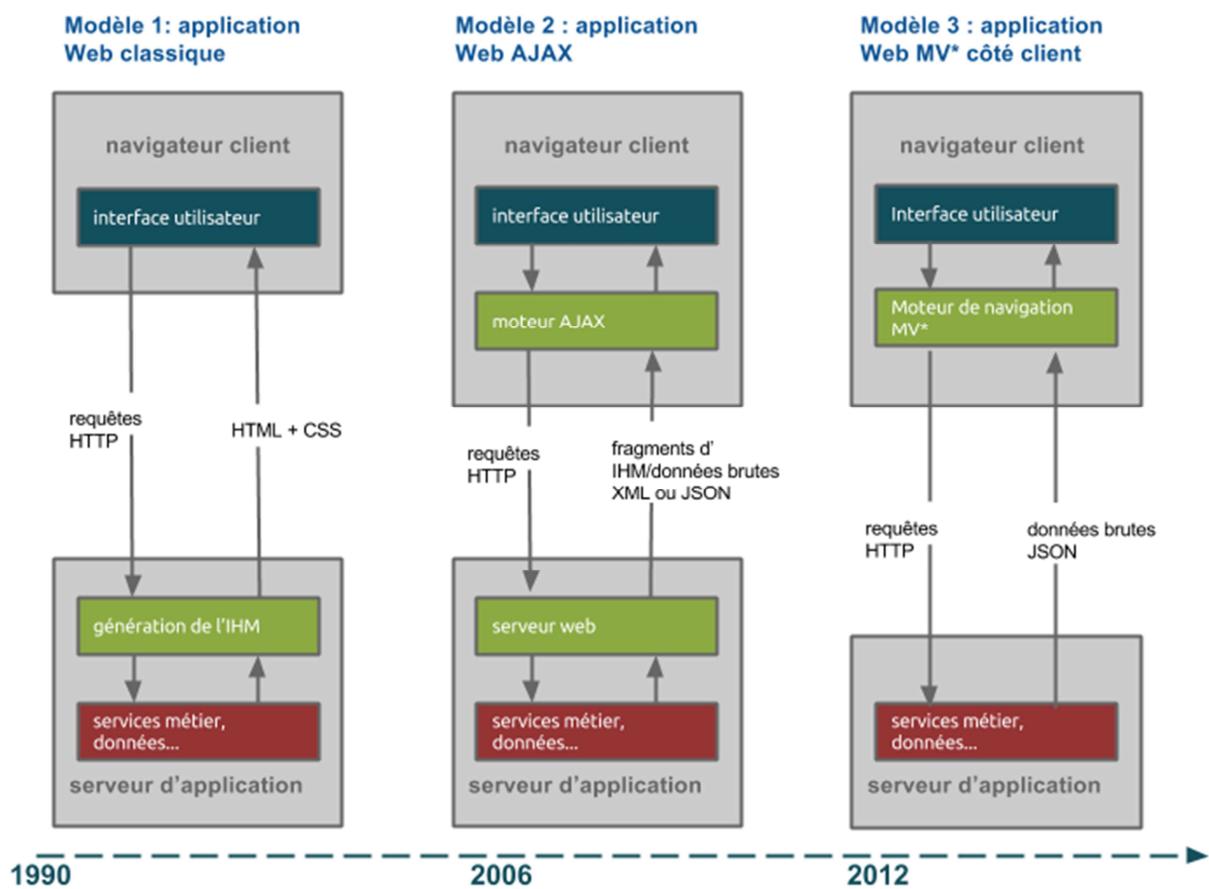
## II. Les Frameworks et Compilateurs

### 2.1. L'utilité des Frameworks JS

Comme nous l'avons vu, le JavaScript devient de plus en plus nécessaire dans les projets web. Les Frameworks sont ainsi logiquement apparus afin d'apporter cette notion de structuration et d'organisation qu'il manquait au développement et vont encore plus loin.

En effet, il n'est pas nécessaire d'intégrer un Framework uniquement pour organiser son développement. Beaucoup d'outils et de méthodes existent actuellement pour pallier à ce besoin. L'avantage du Framework réside dans le fait qu'il vient avec un lot de fonctionnalités permettant de répondre à un ensemble de problématiques que l'on rencontre souvent en cours de projet et auquel on n'a pas forcément pensé.

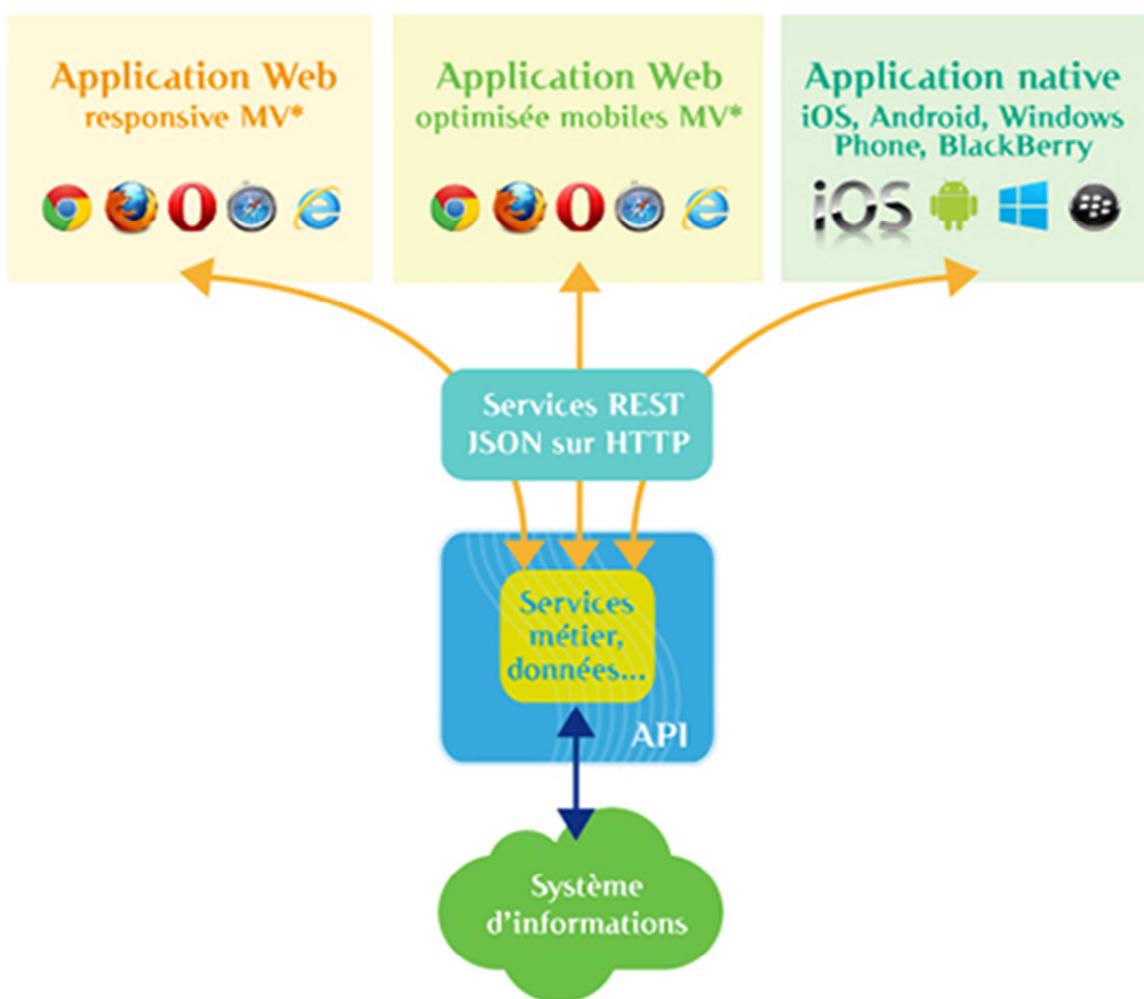
Nous pouvons voir ci-dessous un schéma qui explique les différentes évolutions que nous avons déjà évoquées :



Source : <http://blog.octo.com/>

Le « modèle 3 » illustre les changements apportés par les Frameworks sur l'architecture d'un projet web. La première différence est que la génération des templates<sup>17</sup> HTML, qui était avant effectuée du côté du serveur (Backend), est maintenant remplacée par des moteurs de templates s'exécutant du côté du Frontend. Il n'y a donc plus qu'un échange de données brutes augmentant la séparation entre le client et le serveur. Ce type d'échange a permis encore plus de populariser les APIs s'appuyant sur les principes REST<sup>18</sup> qui, dans le web, permettent d'utiliser pleinement le protocole HTTP<sup>19</sup> afin d'effectuer différentes actions sur une même ressource telles que l'ajout, la modification, l'affichage ou encore la suppression. Grâce à cette séparation et à ce type d'échange, nous pouvons également utiliser différents clients (application mobile, site adaptif, application web) et interagir avec les données et les services métier centralisés sur le serveur.

#### *Architecture globale d'un projet avec différents clients*



Source : <http://blog.octo.com/>

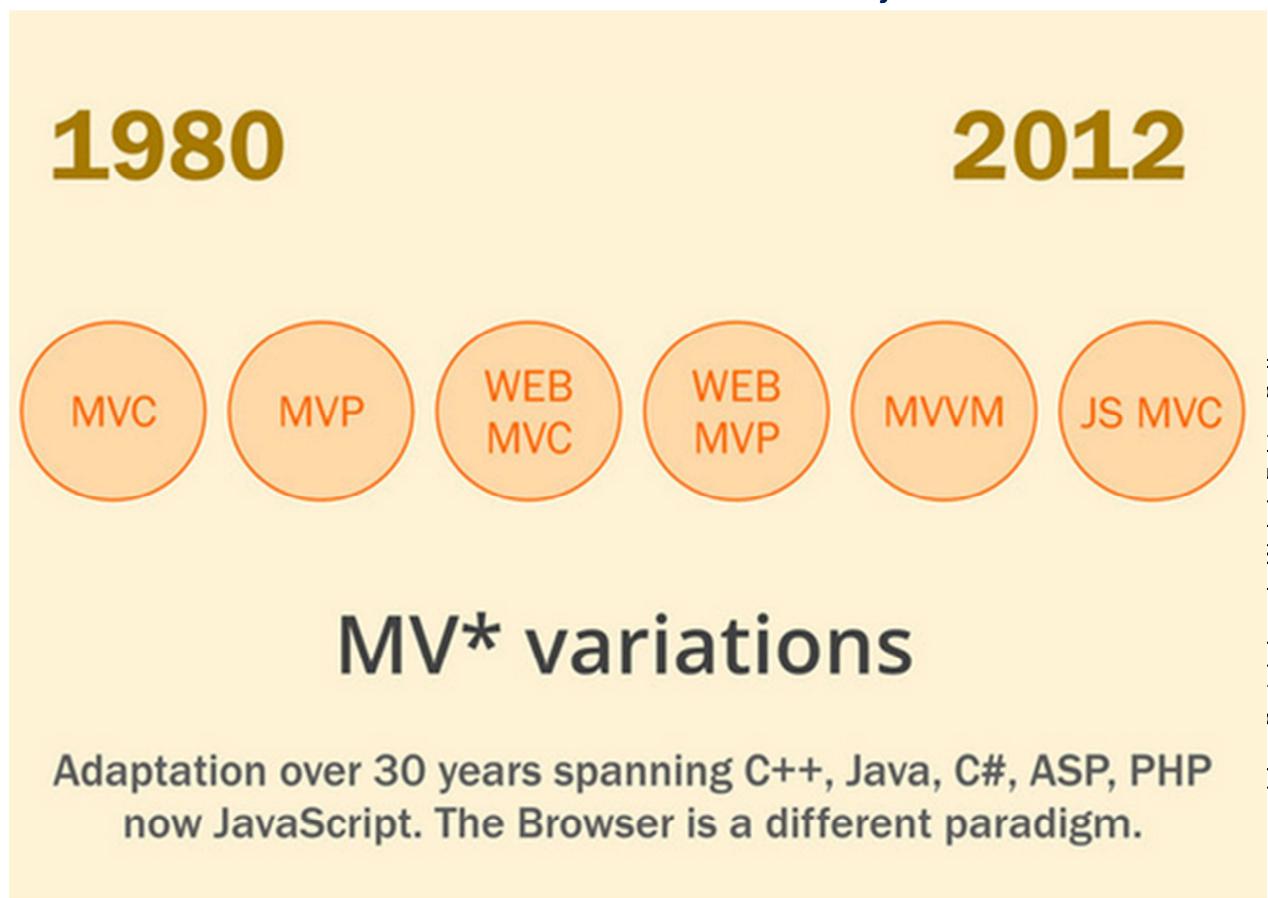
<sup>17</sup> **Template** : structure HTML mettant en forme le contenu de la page.

<sup>18</sup> **REST** : Representational State Transfer. Ensemble de principes créés par Roy Fielding.

<sup>19</sup> **HTTP** : Hypertext Transfer Protocol. Protocole de transfert hypertexte en français permettant de communiquer entre client et serveur sur le Web.

Les 2 illustrations représentent les Frameworks JavaScript par le sigle « MV\* ». Cette abréviation comporte le M pour désigner le modèle (les données) et le V pour la vue (les templates HTML). L'étoile qui suit sert à regrouper un ensemble d'architecture dérivant du design pattern MVC. Certaines personnes n'utilisent pas le terme MV\* mais MVW (W pour Whatever).

#### *Variation du modèle MVC de 1980 à nos jours*



*Source : <http://addyosmani.com/>*

Il y a beaucoup d'abus de langage à la fois sur l'utilisation du pattern MVC et sur la création de nouveaux patterns reflétant le fonctionnement d'un ou plusieurs Framework. Cela est en parti dû au fait qu'il n'y a pas de solution « miracle » répondant à tous les problèmes. Certaines modélisations peuvent mieux convenir dans certains cas de figures et peuvent mieux refléter l'interprétation qu'a le développeur dans le JavaScript et son organisation du développement. L'abréviation MV\* permet donc de regrouper tous les Frameworks en évitant de les catégoriser et de les différencier car au final, ils ont tous pour but de structurer une application web.

## 2.2. Comment choisir son Framework JS ?

Comme nous venons de le voir, le JavaScript possède de nombreux Frameworks apportant de nouvelles approches et de nouvelles manières de développer en JavaScript. Le langage lui-même évolue conjointement aux standards d'ECMAScript et aux nouvelles fonctionnalités des navigateurs qui apparaissent jour après jour. Dans ce contexte, il est très dur, à la fois pour les développeurs et pour les Frameworks, de rester en adéquation avec l'utilisation que l'on fait du JavaScript. Nous nous retrouvons donc dans un phénomène que l'on appelle en anglais « Yet Another Framework Syndrome » (YAFS) où chaque jour de nouvelles librairies et Frameworks apparaissent afin de proposer de nouvelles solutions à de nouveaux problèmes. Cela permet de laisser place à l'innovation mais augmente en même temps notre confusion et notre difficulté à choisir les outils dont nous avons besoins afin de réaliser un projet. Comment pouvons-nous donc procéder afin de sélectionner le Framework qu'il nous faut ? Le mieux est de tout d'abord lister un ensemble de critères afin de faire le tri.

### 2.2.1. Est-il vraiment nécessaire d'utiliser un Framework ?

Nous avons déjà vu pour quelles raisons nous devions utiliser un Framework. Cependant, il n'est pas obligatoire d'en utiliser un et il peut être avantageux selon les cas de figures de s'en dispenser. Si 90% des fonctionnalités du Framework sont inutilisées, nous nous retrouverons avec du code JavaScript inexploité qui cependant sera chargé par le navigateur et contribuera à la diminution des performances. Il peut donc être plus judicieux de recourir à de simples « polyfill<sup>20</sup> » ou à des librairies plus légères qui feront aussi bien le travail.

### 2.2.2. Quelles sont les fonctionnalités dont nous avons besoin ?

La première chose à faire est de repérer l'ensemble des éléments que le Framework devra posséder. Avons-nous besoin de gérer un modèle, des routes, des vues, des composants graphiques ... ?

### 2.2.3. Quelles sont les compatibilités que je dois prendre en compte ?

Cette étape peut être éliminatoire et peut restreindre grandement les choix qui s'offrent à nous. Devons-nous avoir une compatibilité avec IE7, IE8 ? Avons-nous des contraintes techniques entre le Framework JavaScript et les autres technologies (choisies, imposées) sur le projet ?

### 2.2.4. Avons-nous besoin de flexibilité ou de rigidité ?

Il se peut que l'écosystème du projet nécessite un Framework flexible. Cela peut venir d'un manque de spécifications, d'un besoin d'adaptation en fonction des contraintes techniques ou encore des fonctionnalités peu communes nécessitant la liaison de plusieurs

---

<sup>20</sup> **Polyfill** : ensemble de fonction simulant des fonctionnalités que le navigateur ne possède pas

librairies / Frameworks. A contrario, un projet ambitieux peut avoir besoin d'une certaine rigueur autant dans le développement que dans la structure du projet.

### 2.2.5. Le Framework est-il mature ?

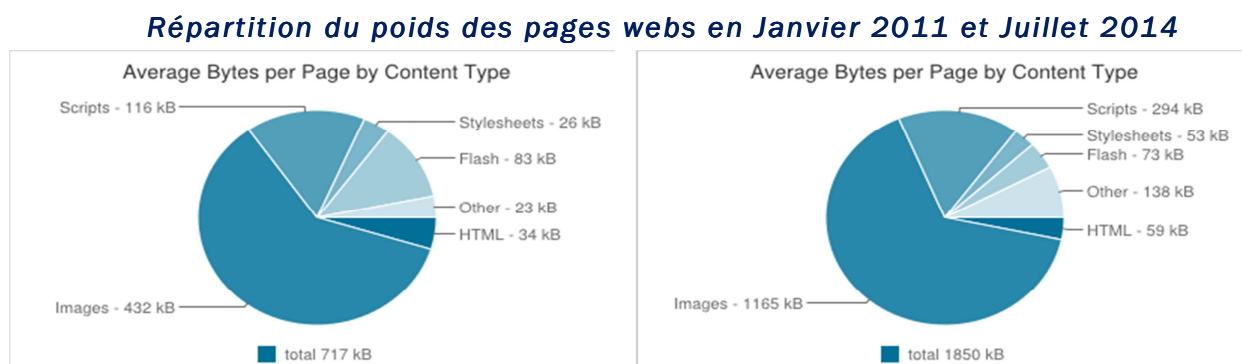
Le JS évoluant rapidement, certaines nouveautés peuvent avoir un effet de « buzz » et disparaître aussi vite qu'elles sont apparues. Il est donc important de prendre un peu de recul, de tester et d'avoir des retours d'expériences avant toute utilisation en production. Un mauvais choix technique peut jouer sur la pérennité d'un projet. Il se peut qu'au bout de quelques semaines ou mois de développement le Framework nous posent des problèmes d'incompatibilités, des limitations, qui imposent de revenir sur les choix effectués et fassent perdre à la fois du temps et de l'argent.

### 2.2.6. La communauté du Framework est-elle importante ?

Dans la même continuité, il est préférable de prendre en compte la communauté qu'il y a autour du Framework. Une bonne communauté permet de s'assurer que le Framework aura plus d'avenir que d'autres, qu'il y aura des corrections, des évolutions, de la documentation et de l'aide que cela soit par le biais de composants déjà développés, ou par le biais de tutoriaux et forums.

### 2.2.7. Avons-nous des contraintes de performances ?

Le poids moyen des pages Web a considérablement augmenté ces dernières années. En utilisant HTTP Archive, on peut arriver à obtenir différentes statistiques sur le web. Les 2 graphiques ci-dessous représentent la taille moyenne découpée selon le type de données chargées en janvier 2011 et en juillet 2014 sur l'ensemble des pages web.



Source : <http://httparchive.org/>

On remarque que les éléments d'une page Web (mise à part Flash) ont évolué proportionnellement au poids total. On se retrouvait donc en 2011 avec une moyenne de 717kB et 116kB de JavaScript alors qu'en juillet 2014, le poids total ainsi que le JavaScript ont subi une augmentation de 260%. Cela s'explique par le fait que les interfaces Web sont devenues plus riches mais également que les compétences des développeurs ont augmentées avec les architectures qu'ils emploient pour répondre aux différents besoins. Il est donc normal que les contraintes de poids et de chargement des pages Web deviennent également plus en plus importantes.

Néanmoins, ce critère peut être dur à prendre en compte. Beaucoup de comparatifs utilisent la taille du Framework comme critère « unique » sur les possibles répercussions lors du chargement des pages web mais il faut également prendre en compte l'utilisation que l'on en fera. Il est certes plus judicieux d'utiliser un Framework léger pour une petite application nécessitant d'être chargée sur mobile et possédant donc des contraintes de bandes passantes mais il peut en être tout autrement si le travail réalisé est important. Il peut par exemple être utile de s'appuyer sur un Framework imposant qui pourrait limiter le code produit par la suite, qui sera certes plus lourd à charger mais également plus rapide à exécuter. Il est également intéressant de vérifier la modularité du Framework. En effet certains proposent de sélectionner et de récupérer uniquement les composants utilisés et d'autres peuvent facilement se décomposer et s'intégrer à des outils de chargement asynchrones.

#### 2.2.8. Quelles sont les compétences techniques de l'équipe de développement ?

Les développeurs ont-ils de l'expérience dans l'utilisation du Framework ? Si ce n'est pas le cas, ont-ils un profil technique orienté Back ou Front ? Cet élément peut sembler futile est pourtant certains Frameworks possèdent une philosophie plus proche des langages Backend avec des concepts et des patterns communs tandis que d'autres peuvent utiliser des méthodes de développement propres au Front. Il faut également penser à la courbe d'apprentissage : Il se peut que l'apprentissage initial soit facile et que très rapidement, on rencontre des limitations. A l'inverse il se peut que le Framework soit dur à appréhender mais possède un gain de productivité beaucoup plus important une fois certaines étapes franchies.

## 2.3. Comparatif de Frameworks

Les éléments précédents permettent de comprendre les difficultés que l'on peut rencontrer dans le choix d'un Framework JavaScript. Nous pourrions définir une grille comparative en fonction de certains critères mais celle-ci ne serait pas réellement utile car, comme nous l'avons vu, le choix d'un Framework dépend plus de l'utilisation que l'on en fait que de ses caractéristiques techniques. En effet, le type de design pattern MVC, MVP, MVVM... n'est juste qu'une modélisation différente pour effectuer une séparation et structurer un projet. La taille du Framework importe beaucoup moins que la manière dont nous l'utilisons. Un Framework léger et flexible peut, par exemple, devenir une meilleure solution pour un gros projet car il peut permettre plus facilement de se coupler à d'autres outils (librairies et Frameworks) répondant mieux à des besoins spécifiques...

Il est difficile d'évaluer toutes les solutions du marché. Celles-ci peuvent rentrer en conflits ou au contraire, être complémentaires. De plus elles sont tellement nombreuses et évoluent tellement vite qu'il devient dur de toutes les connaître et arriver à avoir un retour d'expérience dessus. La meilleure chose à faire est donc de présélectionner un nombre de Frameworks restreint répondant à des critères précis (comme ceux cités précédemment). Le site [todomvc.com](http://todomvc.com) peut également permettre d'aider à effectuer un choix en visualisant le code d'une application conçue avec les Frameworks les plus populaires. J'ai décidé de vous présenter ceux que j'ai estimé être les plus intéressants actuellement. Cependant, ce comparatif est loin d'être exhaustif et il est probable que je sois passé à côté de meilleures solutions.

### 2.3.1. Les Frameworks sélectionnés

Backbone.js



Source <http://backbonejs.org/>

Backbone a été créé en octobre 2010 par Jeremy Ashkenas (créateur de la librairie Underscore et du compilateur CoffeScript, dont nous reparlerons plus loin). Son point fort repose sur sa flexibilité et sa légèreté. Sa syntaxe beaucoup moins déroutante que certains de ses concurrents, possédant une grande ressemblance avec le JavaScript objet « standard », lui a permis d'être très populaire au moment où les Frameworks JavaScript ont commencé à se faire connaître.

## Ember.js



Source <http://www.infragistics.com/>

Ember se décrit lui-même comme un Framework permettant de créer des applications web ambitieuses. Il a été initialisé en 2011 et il lui aura fallu attendre août 2013 afin de sortir la version 1.0. Les deux principaux contributeurs d'Ember sont Yehuda Katz et Tom Dale. Yehuda fait également partie des équipes de développement de Ruby on Rails (Framework Ruby) et de jQuery Core (librairie JS). Tom Dale quant à lui, faisait partie des contributeurs de SproutCore (ancienne version d'Ember.js) et travail chez Apple.

## Angular.js



Source <http://www.developpez.com/>

Angular.js est le plus ancien des Frameworks de ce comparatif. Il a été créé par Miško Hevery et Adam Abrons de Google en 2009. Contrairement aux deux précédents, Angular a eu un peu plus de mal à se populariser durant ses premières années mais à maintenant très largement dépassé Backbone et Ember. Cela est en partie dû à sa philosophie et à sa manière de concevoir des applications Web qui se différencie de ses concurrents.

## Polymer.js



Source <http://connect.adfab.fr/>

Polymer est sorti en fin d'année 2013 par Google qui a pour but d'expérimenter les nouvelles fonctionnalités du Web. Il se compose d'un ensemble de « polyfill » permettant d'utiliser des futurs éléments qui ont pour vocation d'être implémentés nativement sur les navigateurs.



Source <http://moduscreate.com/>

React est également très récent. Il est développé par Facebook, a été ouvert au public en mai 2013 et utilise une nouvelle façon de gérer les vues. Son objectif et de compenser les limitations actuelles du DOM et d'augmenter les performances lorsque le JavaScript change dynamiquement le contenu des templates.

### 2.3.2. Leurs caractéristiques

#### Backbone.js

Backbone est avant tout une librairie minimalistre qui se couple très facilement avec d'autres et qui n'impose aucune manière de l'utiliser. Seul, on ne peut donc pas le considérer comme un Framework en tant que tel mais il peut cependant le devenir lorsqu'il est couplé à des librairies additionnelles telles que Chaplin, Marionette, Aura ou encore Thorax. Étant très léger et souple, il est facile de l'étendre ou de remplacer certains de ses composants tels que le système de template ou de routing<sup>21</sup>. Ces particularités font qu'il est souvent préféré lorsqu'il y a de fortes contraintes architecturales à côté ou encore pour le coupler à d'autres Frameworks répondant à des critères plus précis.

#### Ember.js

Ember.js est tout l'inverse. Il se couple difficilement avec d'autres libraires et accordent une grande importance au respect de ses conventions et de son architecture. Contrairement à Backbone, Ember gère beaucoup de choses et évite ainsi de devoir réfléchir sur la façon de modéliser et développer une fonctionnalité. On appelle cette philosophie en anglais « convention over configuration ».

#### Angular.js

On pourrait situer Angular entre Backbone et Ember. Bien qu'il puisse être flexible, il offre tout de même un cadre et certaines fonctionnalités intéressantes. Sa popularité actuelle vient du fait qu'il a trouvé le moyen d'intégrer des bonnes pratiques provenant des langages Back vers le front. On peut par exemple citer l'injection de dépendance<sup>22</sup> et les services<sup>23</sup>. Ces concepts permettent de limiter les dépendances entre les différents éléments développés et de faire facilement des tests unitaires. Angular utilise également le paradigme « convention over configuration » au niveau de ses vues.

<sup>21</sup> **Routing** : routage en français est le mécanisme déterminant la destination d'une URL

<sup>22</sup> **Injection de dépendance** : Dépendancy Injection (DI) en anglais et un mécanisme permettant de créer de l'inversion de contrôle (voir le glossaire pour l'inversion de contrôle).

<sup>23</sup> **Services** : dans le contexte utilisé, les services représentent des objets instanciés par l'injection de dépendance

## Polymer.js

Son principal but est d'exploiter une nouvelle spécification HTML encore en cours d'implémentation du nom de « Web Components ». L'idée est de créer des composants personnalisés en les séparant de la structure globale de la page. Nous pouvons ainsi beaucoup plus facilement réutiliser ou modifier ces mêmes composants, éviter la duplication de code et rendre les templates beaucoup plus légers et compréhensibles. Polymer dispose de plusieurs composants qui peuvent être pris séparément et intégrés dans d'autres Framework. Ne souhaitant pas proposer de compatibilité avec d'anciennes versions de navigateur, Polymer ne fait que regarder le futur du Web.

## React.js

Au lieu de se préoccuper (comme beaucoup de Framework) du modèle et d'effectuer des liaisons sur des moteurs de templates, React gère directement ses vues. L'avantage est que lorsque des éléments sont modifiés, React ne va pas changer la globalité du template comme beaucoup de système font mais va effectuer littéralement une différence entre le template initial et le template modifié afin d'insérer dans le DOM uniquement cette même différence.

### 2.3.3. Leurs compatibilités et dépendances

Framework	Dépendances
Backbone	Undescore, jQuery ou Zepto
Ember	Handlebars, jQuery, Ember-Data
Angular	aucune mais souvent utilisé avec jQuery
Polymer	aucune
React	aucune

## Leurs compatibilités

Mise à part Polymer, l'ensemble des Frameworks de ce comparatif est compatible avec IE8+. Cependant, comme nous le voyons dans le tableau ci-dessus, certains Frameworks ont des dépendances avec des librairies externes. Les compatibilités des navigateurs peuvent donc changer si par exemple on utilise la version 2.\* de jQuery qui ne supporte plus IE8.

Polymer, quant à lui, ne tient à jour que les navigateurs « evergreen » représentant les navigateurs qui se mettent à jour automatiquement tel que Chrome ou Firefox. Nous pouvons voir dans le tableau ci-dessous que cela inclus également IE à partir de la version 10.

## Prises en charges des Web Components dans les navigateurs

	Polyfill						
	Chrome Android	Chrome	Canary	Firefox	IE 10+	Safari 6+	Mobile Safari
MutationObserver	[1]						
HTML Imports	[1]						
Custom Elements							
Shadow DOM							
MDV							
Pointer Events				[2]			
Pointer Gestures							
Web Animations							
Platform							
Toolkit							

Source <http://slides.com/veeravenkatraja>

### 2.3.4. Leurs architectures

#### Backbone.js

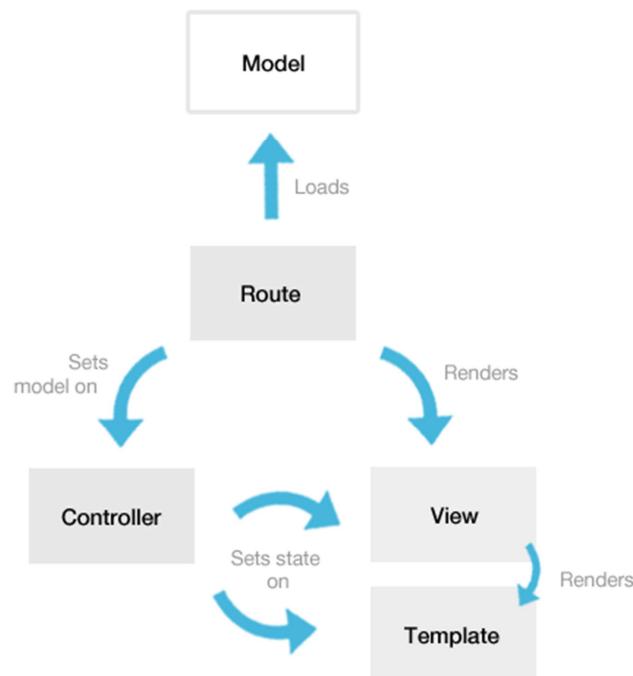
Comme nous l'avons déjà vu, Backbone pris seul n'est pas un Framework mais une librairie. Il correspond à ce qu'on nomme le MVW (Model View Whatever). Backbone possède un système de vue, de modèle, de collection, de route mais il nous laisse choisir quelle architecture adopter afin d'utiliser tous ses composants. C'est à la fois son avantage et inconvénient. Backbone subit de nombreuses critiques sur le fait qu'il demande beaucoup plus de développement et qu'il ne permet pas d'être aussi productif que certains de ses concurrents. Il est vrai qu'il faut en général écrire plus de code, cependant il dispose d'une multitude de librairies afin de pallier aux différents éléments manquants et nous permet ainsi de ne pas « réinventer la roue » tout en utilisant des patterns éprouvés.

Par défaut, il utilise ce que l'on appelle le « One-Way Data Binding ». Le data binding est le fait de lier les données issues de la vue à celles se trouvant dans le modèle. Le One Way signifie que par défaut, Backbone ne possède qu'un data binding unidirectionnel effectué par le modèle. Lorsque l'on modifie les données de la vue, le modèle n'est pas automatiquement modifié. Il nous faut utiliser les évènements afin d'écouter les différentes modifications effectuées et réaliser manuellement des actions sur le modèle. Nous pouvons cependant combler ce manque en utilisant d'autres librairie comme Stickit, Epoxy ou encore Bind'em.

#### Ember.js

Ember utilise le modèle MVC. Sa grande valeur ajoutée comparé aux autres Frameworks réside dans son routeur. Il utilise l'architecture « URL Driven Development » signifiant que toute l'organisation des fichiers repose sur la structure de l'URL. En suivant les conventions de nommage, Ember permet ainsi de gérer de nombreuses choses de manière transparente.

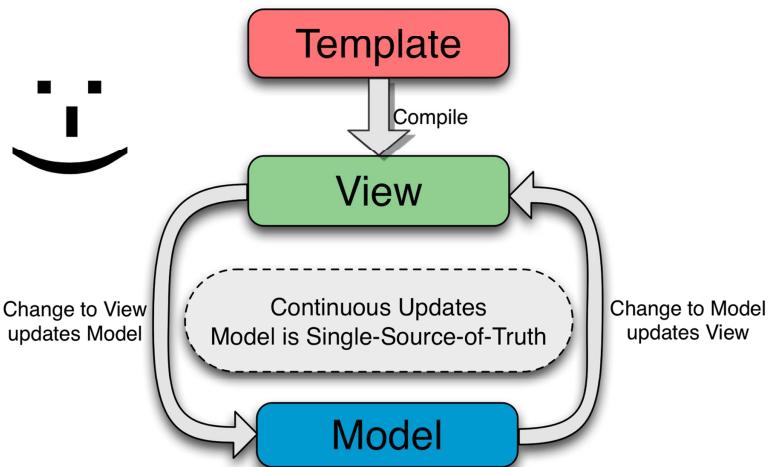
### Modèle MVC d'Ember.js



Source <http://www.smashingmagazine.co>

Contrairement à Backbone, Ember utilise le « Two-Way Data Binding » signifiant que la modification des données issues de la vue modifiera automatiquement celle du modèle et vice-versa.

### Two-Way Data Binding



Source <http://www.codeproject.com>

Ember va également plus loin que les autres car en plus de gérer le modèle et le Data Binding, il dispose maintenant d'un ORM<sup>24</sup> gérant la persistance du côté du client en communiquant avec des API REST.

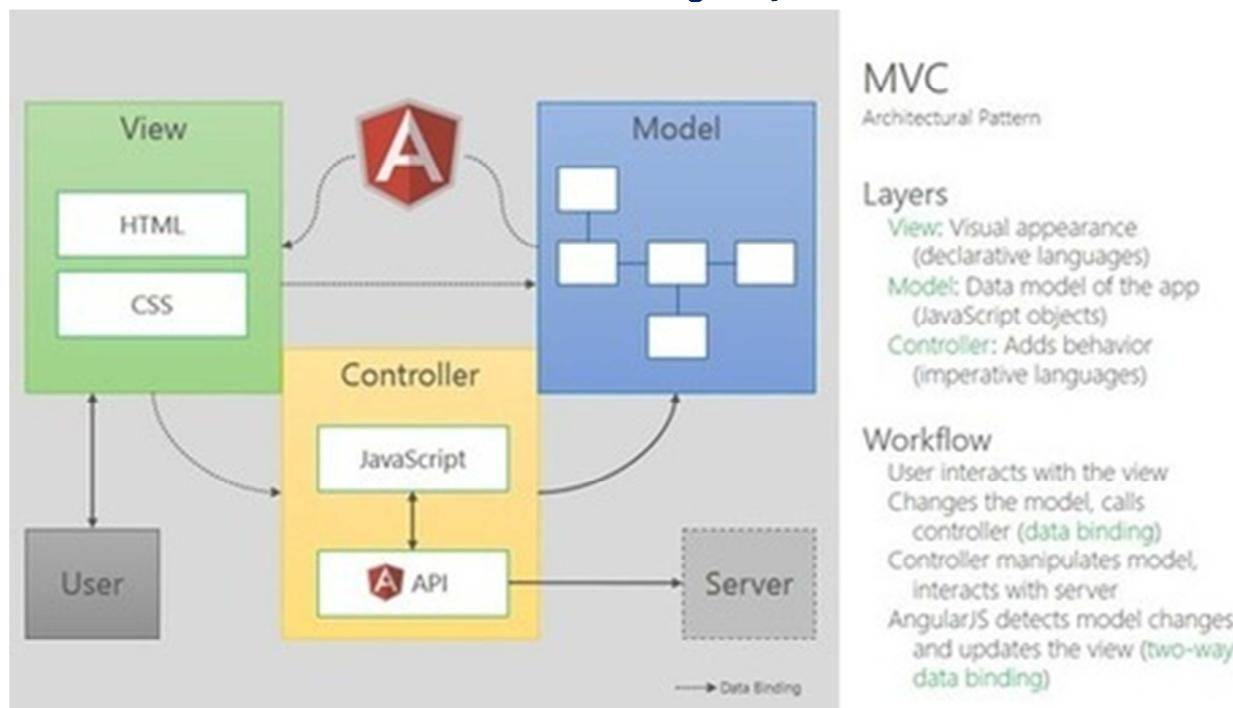
<sup>24</sup> **ORM** : Objet-Relational Mapping. Technique gérant une base de données avec des objets

Cette nouvelle partie s'appelle Ember-Data. Elle est actuellement dissociée d'Ember mais elle a pour vocation d'être intégrée au Framework. Ember-Data permet donc de gérer des méthodes CRUD (Create Read Update Delete) mais possède également un système de cache, un système de transaction et de synchronisation automatique avec validation et merge (fusion). L'objectif est de pouvoir enlever toutes dépendances avec le Backend.

## Angular.js

Angular possède également une architecture MVC qui diffère cependant de celles que l'on peut trouver sur les langages côté serveur. Celle-ci se rapproche plus d'une architecture MVVM (Model View ViewModel). Le contrôleur existe mais il est utilisé uniquement pour initialiser l'état et rajouter des comportements au « \$scope ». Le \$scope est un objet qui utilise le modèle afin d'effectuer des actions sur la vue. La notion de ViewModel n'est donc pas réellement visible mais comporte cet objet qui permet grâce au Two-Way data binding d'animer la vue et d'interagir avec le reste de l'application. Le ViewModel est représenté dans le diagramme ci-dessous par le contrôleur.

Modèle MVC d'Angular.js



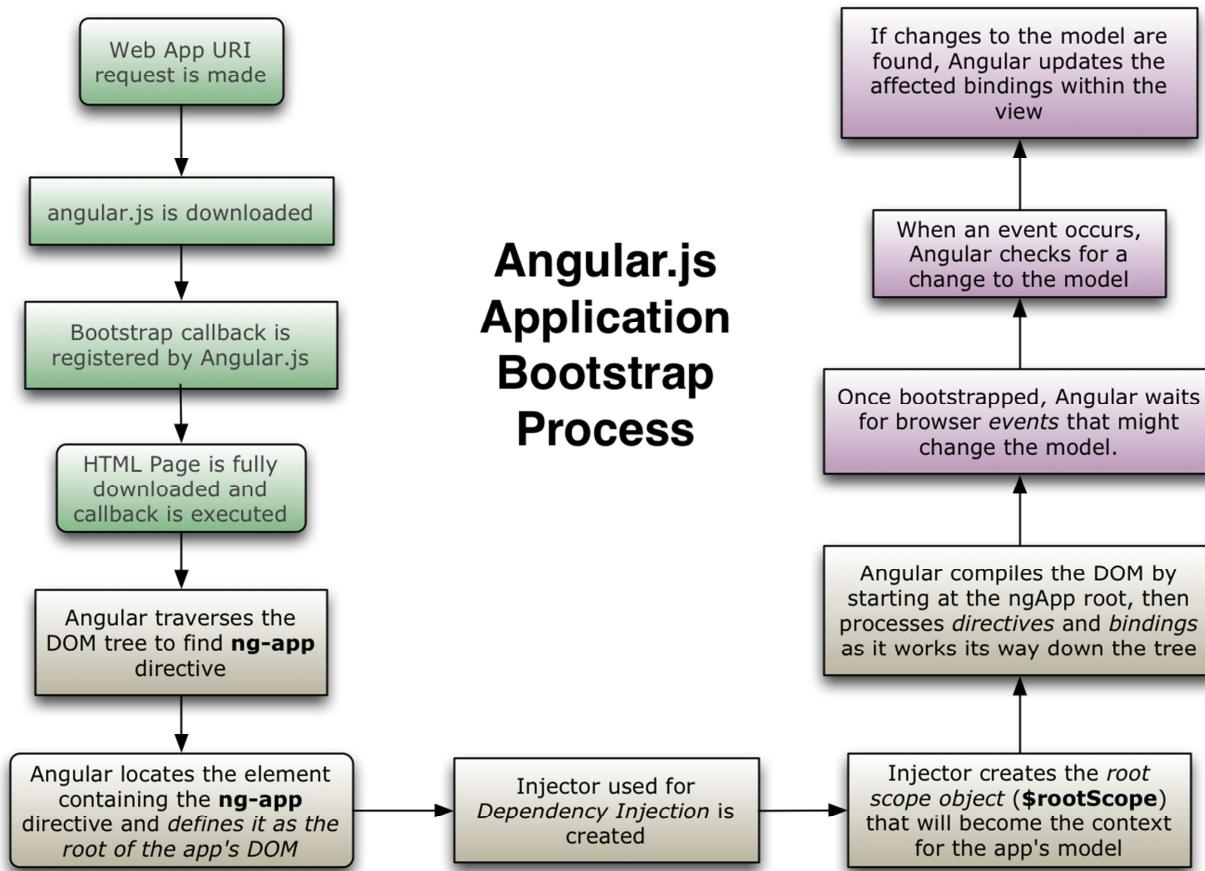
Source <http://www.codeproject.com>

Angular n'est pas aussi flexible que Backbone ni aussi robuste que Ember. Il ne possède pas de couche de persistance ni de système de routing aussi évolué qu'Ember mais a tout de même ses avantages. Angular a été le premier Framework à effectuer une réel séparation entre le JavaScript et la vue tout en enrichissant la syntaxe HTML afin de réaliser certains comportements, effectuer le Data Binding et utiliser la notion de « convention over configuration » dont nous avons déjà parlé. Le deuxième point fort d'Angular est qu'il a été

conçu pour être facilement testé. L'injection de dépendance, l'inversion de contrôle<sup>25</sup> et les autres patterns qu'utilise Angular sont faits pour mettre les tests au centre du développement. Le code devient donc plus maintenable, plus élégant, plus modulaire et plus réutilisable.

L'image qui suit représente le fonctionnement interne d'Angular du chargement du fichier jusqu'à la manipulation du modèle et de la vue. Il fait mention d'un élément dont nous n'avons pas parlé qui s'appelle les directives. Ce sont les directives qui permettent de créer des éléments HTML personnalisés en leurs rajoutant des comportements. Le moteur de template angular est lui-même composé de directives.

### Fonctionnement interne d'Angular.js



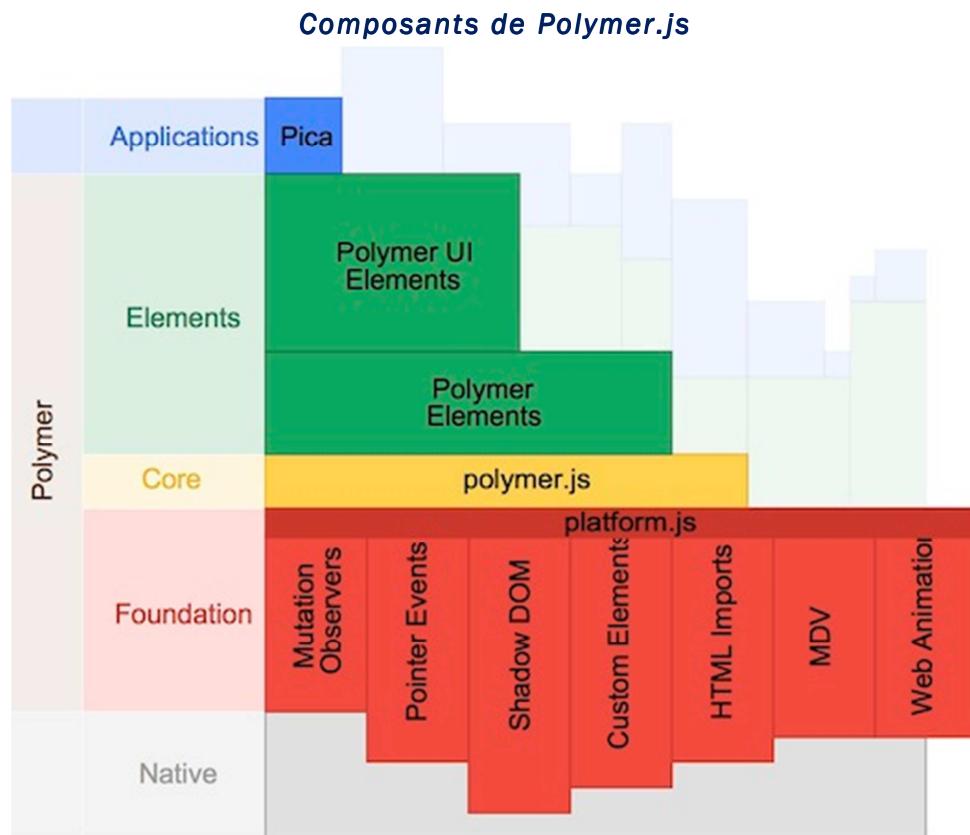
Source <http://www.codeproject.com/>

### Polymer.js

Polymer ne fait pas réellement partie de la catégorie des Frameworks MVW mais je l'ai inclu dans le comparatif car il concurrence les Frameworks sur l'utilisation des vues. Comme nous l'avons déjà dit, c'est un ensemble de « polyfills » permettant de simuler et de créer des fonctionnalités qui ne sont pas encore intégrées dans les navigateurs. Il se focalise sur la spécification des Web Components en utilisant le concept MDV (Model Driven View).

<sup>25</sup> **Inversion de contrôle** : ou IoC est un patron d'architecture donnant le contrôle sur le Framework utilisé plutôt que l'application en elle-même.

Le MDV permet d'utiliser le Data Binding et les templates pour les Web Components d'une manière qui ressemble fortement à celle d'Angular et ses directives. Polymer peut facilement s'intégrer à d'autres architectures mais il est par contre plus dur d'intégrer un autre Framework à l'intérieur d'un Web component de Polymer. Voici l'ensemble des composants que Polymer possède :



Source : <http://www.2ality.com/2013/07>

On retrouve donc dans la partie rouge, les fonctionnalités des Web Components qui seront utilisées par une grande partie des Frameworks dans le futur et qui iront peut-être même jusqu'à remplacer les APIs natives de certains navigateurs. Nous avons parlé de polyfill mais pris séparément, cette partie de Polymer représente plus des « shims ». Un shim intercepte les appels d'une API et effectue des comportements différents sur celle-ci tandis qu'un polyfill est un shim qui va vérifier que le navigateur supporte l'API avant d'en modifier le comportement.

Les autres parties permettent d'utiliser les shims des Web Components en créant un petit socle structurel qui peut être vu comme un petit Framework afin d'utiliser les « Polymer elements ».

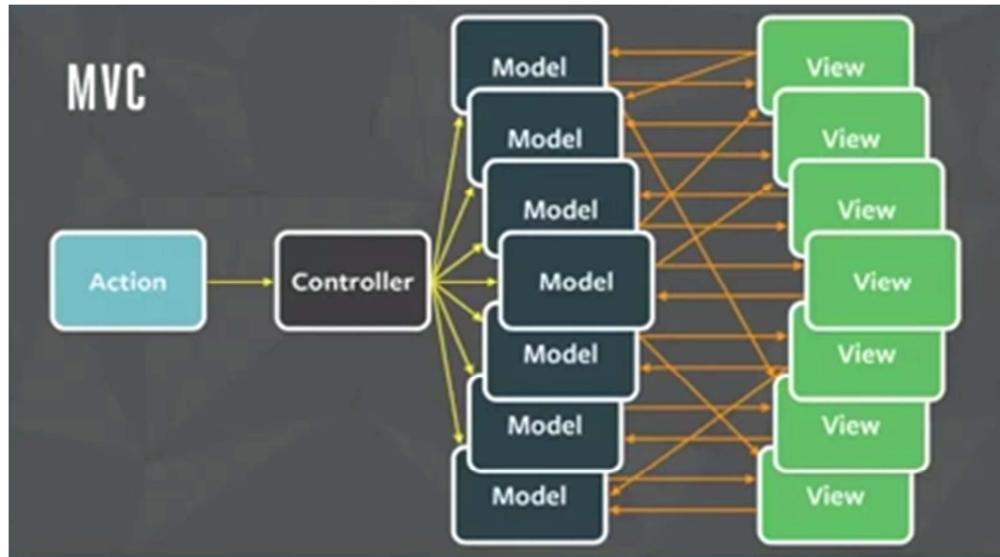
### React.js

React ne fait pas non plus partie des Frameworks MVW mais il propose une nouvelle manière de construire des applications grâce à un nouveau pattern appelé « Flux ». Contrairement aux Frameworks MV\* structurant leurs développements autour de contrôleurs, modèles et Data Binding bidirectionnels, Flux encourage la centralisation des

données entre le modèle et la vue, l'utilisation d'objets immuables<sup>26</sup> et de flux de données à sens unique (One Way Data Binding).

Flux est apparu par le constat que les Frameworks MV\* ne s'adaptaient pas aux besoins de Facebook (créateur de React). De leur point de vue, plus l'application était grande, plus les modèles MVC devenaient rapidement compliqués rendant le code fragile et imprévisible.

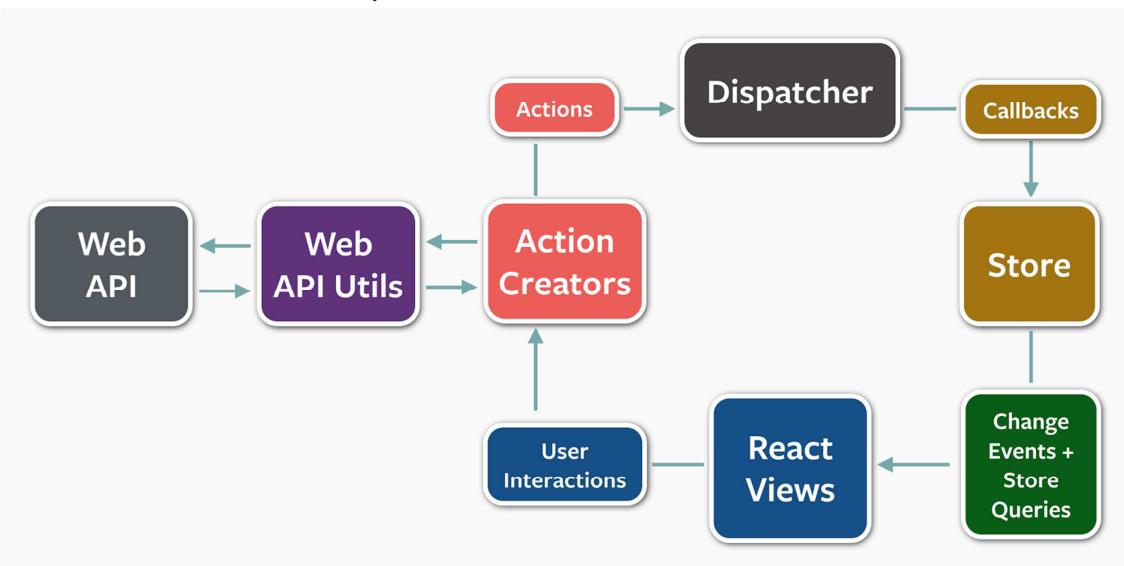
*Problématique du modèle MVC sur un gros projet*



Source <http://www.infoq.com/>

Le pattern Flux ainsi que React sont donc apparus du constat que les applications complexes avaient besoin de se focaliser sur l'interface afin de gagner en clarté, en productivité et en performance.

*Représentation du modèle Flux*



Source <http://facebook.github.io/>

<sup>26</sup> **Objet immuable** : objet dont l'état ne peut pas changer après sa création

Le dispatcher est un singleton<sup>27</sup> qui dirige le flux de données grâce au Data Binding unidirectionnel afin de s'assurer qu'il n'y est pas de mise à jour en cascade et que le résultat d'une interaction sur l'interface devienne facilement prévisible. Le dispatcher s'occupe également des dépendances entre les différents « Store » et enregistre des fonctions de rappels (callbacks) dans un ordre spécifique. Le dispatcher est donc le point vital du pattern par lequel toutes les actions passent.

Les Actions et ActionCreators dans React font référence à une interaction effectuée par une personne ou encore un appel d'une API. Chaque action est représentée par un ensemble de données et envoyée au dispatcher afin que ce dernier puisse déterminer quelle fonction de rappel du « Store » et quelle « View » doit la traiter. On peut donc dire qu'une Action et une représentation du design pattern<sup>28</sup> Command<sup>29</sup> dans React.

Le Store contient l'état et la logique métier de l'application. Son rôle est similaire au modèle dans les Frameworks MVW traditionnels, cependant, il ne représente pas, comme le font ses concurrents, des objets et des collections d'objets provenant d'une API Rest ou d'une base de données. On pourrait dire que le Store est une représentation du domaine dans l'approche DDD. Le DDD (Domain Driven Design) est utilisé en principe pour des applications complexes comportant des fortes interactions.

Les « Views » de React sont automatiquement mises à jour par des événements issus des changements d'état du domaine (le Store). Elles sont, de la même manière que Polymer, des ensembles de composants retrouvant des points communs avec les Web Components afin de pouvoir répondre beaucoup plus facilement aux interfaces utilisateurs complexes. Cependant, les vues de React utilisent un DOM virtuel qui peut s'appuyer pour plus de lisibilité sur le compilateur JSX qui s'occupera de transformer des syntaxes XML en JavaScript.

React subit beaucoup de critiques<sup>30</sup> par rapport à ce choix technique. Certaines personnes adhèrent à cette nouvelle manière de créer des vues et de mettre de la logique métier à l'intérieur ; d'autres trouvent que c'est une régression car il n'y a plus de séparation entre la vue et le JavaScript. La critique principale est que React ne respecte plus le principe de séparation des préoccupations (Separation of concern ou SoC en anglais). La séparation des préoccupations a pour but de réduire le couplage tout en augmentant la cohésion. L'argument de Facebook<sup>31</sup> est que les moteurs de templates classiques encouragent une mauvaise séparation des préoccupations car la technologie utilisée est bien différente, seulement, les préoccupations ne sont pas réellement séparées augmentant un couplage fort entre le JS et les templates.

---

<sup>27</sup> **Singleton** : classe objet possédant qu'une seul instance (un seul objet)

<sup>28</sup> **Design pattern** : patron de conception en français. Représentation de bonnes pratiques reconnues pour répondre à des besoins particuliers

<sup>29</sup> **Command** : le design pattern Command permet d'effectuer une séparation entre une demande et l'action qui en résulte. Il est souvent représenté par un objet "command" permettant de communiquer l'action.

<sup>30</sup> Des avis mitigés pour React : <http://www.infoq.com/fr/news/2013/06/facebook-react>

<sup>31</sup> Conférence vidéo expliquant le choix de Facebook :  
<https://www.youtube.com/watch?v=x7cQ3mrcKaY>

### 2.3.5. Leurs maturités

#### Backbone.js

Backbone est le Framework du comparatif le plus mature. Le site officiel comporte un grand nombre de références. Il a été le premier à être utilisé en production et on peut supposer qu'il aura encore de l'avenir au regard du dynamisme de sa communauté. Cependant, il évolue peu et il risque de passer en phase de déclin sur le cycle de vie d'un produit car depuis son apparition, il a subi beaucoup moins d'évolution comparé aux autres Frameworks. Il faudra donc attendre les futures versions pour connaître les choix qu'il effectuera notamment sur les nouvelles fonctionnalités HTML et JavaScript.

#### Ember.js

Ember est également un Framework très mature. Il possède moins de références que Backbone et Angular, cependant, il est de plus en plus envisagé pour construire des applications robuste. Ember ne possède pas qu'un bon système de routing. Avec l'apparition d'Ember-Data, il se différencie encore plus de ses concurrents et devient un des Frameworks les plus complets du marché. Il ne reste plus qu'à optimiser les performances sur les terminaux mobiles. Point sur lequel Ember a un peu de retard.

#### Angular.js

Angular a subi une évolution fulgurante ces dernières années. Il y a 2 ans, il n'avait que peu de popularité alors que maintenant il est devenu le Framework le plus utilisé. La version 2 d'Angular, comme d'ailleurs beaucoup de Frameworks futurs, va intégrer cette notion de « Web Components ». Il y a de forte chance que cette intégration soit plus simple pour Angular car contrairement aux autres qui, pour la plupart, utilisent des moteurs de templates externes, Angular conservera sûrement la même logique sur son modèle MVVM.

#### Polymer.js

Contrairement aux précédentes réponses, Polymer est très récent et encore en version pré-alpha. Il a surtout pour vocation de montrer comment on construira une application web dans le futur. Il y a encore des manques de documentations et des problèmes<sup>32</sup> de performances sur mobiles ainsi que des conflits de versions ont aussi été remontés. De plus, Polymer, comme nous l'avons dit, est un ensemble de polyfill se basant sur une spécification encore en cours d'implémentation. Cette dernière peut donc à tout moment changer et les polyfills pourront ne plus être utilisés de la même manière. Polymer peut donc être utilisé pour des projets d'études ou des projets à courte durée possédant un périmètre bien défini mais il est néanmoins déconseillé de l'utiliser en Production.

#### React.js

Comme Polymer, React est un Framework récent. Étant donné que la version actuelle du Framework est la 0.11, il est possible que des changements majeurs surviennent jusqu'à avoir une version stable. Il est donc normal qu'il y ait beaucoup de personnes réticentes à l'idée de l'utiliser en production. Cependant, React est déjà présent sur le site de Facebook, Instagram et Github l'utilise pour son éditeur de texte (Atom). Il peut donc être intéressant de profiter de sa philosophie afin d'enrichir les possibilités offertes par les vues et coupler React avec un Framework complémentaire tel que Backbone.

<sup>32</sup> Les problèmes qui apparaissent avec les Web Components de Polymer : <http://developer.telerik.com/featured/web-components-arent-ready-production-yet/>

### 2.3.6. Leurs communautés

Les Frameworks cités sont assez connus et ont une grosse communauté derrière avec, pour certain, de grandes entreprises telles que Facebook ou Google. Cependant, les librairies récentes comme React et Polymer peuvent manquer de documentation et de cas concrets d'exemple. La communauté reste donc un critère important dans le choix d'un Framework. Comme idée de comparaison, j'ai choisi d'utiliser les données issues de Github et Stackoverflow.

Framework	Stars	Forks	Contributors	Releases
Backbone	18 725	4 210	230	21
Ember	10 906	2 340	384	81
Angular	26 814	9 689	918	97
Polymer	5 496	417	17	27
React	8 205	1 084	176	16

*Statistiques de Github le 04/08/2014*

On peut remarquer qu'Angular est loin devant les autres sur tous les aspects et Polymer, étant le projet le plus jeune, possède pas mal d'intérêssante (stars) compte tenu de son âge mais très peu de contributions.

Framework	Questions
Backbone	15 753
Ember	10 938
Angular	47 596
Polymer	709
React	437

*Statistiques de Stackoverflow le 04/08/2014*

On remarque encore une fois qu'Angular est le Framework possédant le plus de question sur stackoverflow. On peut donc en conclure que le gagnant en termes de communautés est Angular.

### 2.3.7. Leurs tailles

Comme nous l'avons vu la taille d'un Framework peut être un critère de sélection. Il faut par contre, penser à prendre en compte le poids des librairies nécessaires et/ou complémentaires afin de compléter les fonctionnalités du Framework. Dans le tableau ci-dessous, nous comparons les poids des librairies (minifiées<sup>33</sup> et compressées) avec les dépendances minimales et celles qui sont en principe utilisées.

Framework	Poids avec dépendances minimales (min + Gzip)	Poids avec dépendances recommandés (min + Gzip)
Backbone	Backbone : 6,5 Underscore : 5 Zepto : 9,1 Total : 20,6kb	Backbone : 6,5 Underscore : 5 jQuery : 29,5 Total : 41kb
Ember	Ember : 90 Handlebars runtime: 1,8 Zepto : 9,1 Total : 100,9kb	Ember : 90 Ember-data : 20,3 Handlebars : 13 jQuery : 29,5 Total : 152,8kb
Angular	Angular : 37kb	Angular : 37 jQuery : 29,5 Total : 66,5kb
Polymer	Polymer : 48kb	Polymer : 48kb
React	React : 26kb	React : 26 Backbone : 6,5 Underscore : 5 Total : 37,5kb

*Poids des différents Frameworks du comparatif*

On remarque qu'Ember fait au minimum 100ko et que la future intégration d'Ember-data n'arrangera guère la situation, ce qui posera encore plus de problèmes concernant les performances mobiles. A côté, on peut comprendre le choix judicieux qu'a effectué Facebook en utilisant à la fois React pour la gestion des vues et Backbone pour l'architecture plus générale du JavaScript sur le site d'Instagram<sup>34</sup>. Backbone et React sont plus légers que Backbone et jQuery. Étant donné que jQuery propose juste d'interagir avec le DOM sans réellement structurer le développement, la combinaison React+Backbone devrait pouvoir se démocratiser dans les mois à venir.

<sup>33</sup> **Minifier** : retirer tous les espaces et retour à la ligne inutile afin de réduire la taille du code

<sup>34</sup> Plus d'information sur le choix effectué par Facebook ;  
<http://fluentconf.com/fluent2014/public/schedule/detail/32395>

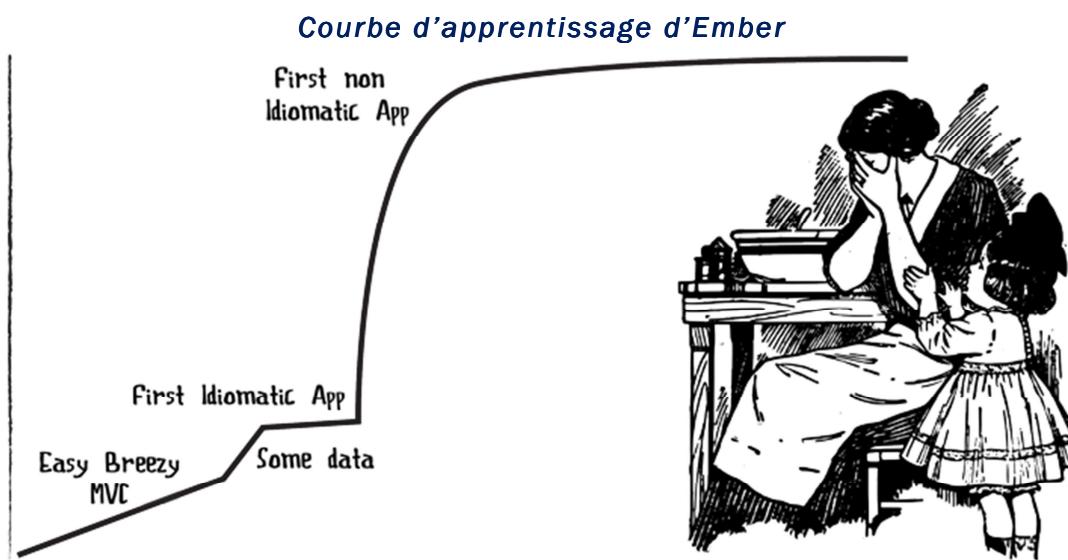
### 2.3.8. Leurs courbes d'apprentissages et de productivité

#### Backbone.js

Backbone est sûrement le Framework le plus facile à apprendre. Cependant, n'imposant aucune architecture précise, la lenteur se fera ressentir dans la recherche et l'application de bonnes pratiques ou dans l'utilisation de surcouches comme Marionnette<sup>35</sup>, Chaplin ou encore Thorax qui permettront d'architecturer correctement un projet web. Sa simplicité permettra d'être rapidement productif au démarrage du projet, néanmoins, on remarquera également très vite que Backbone nous oblige à écrire plus du code comparé à ses concurrents.

#### Ember.js

La grandeur d'Ember fait que l'apprentissage sera beaucoup plus lent que pour Backbone mais il permettra également d'être beaucoup plus productif par la suite. En respectant le principe de « convention over configuration », Ember permet de ne pas « réinventer la roue » et de réduire le code écrit.



Source <http://bigemployee.com/>

Cette courbe montre sur l'axe des abscisses l'évolution des connaissances acquises (l'expertise) et sur l'axe des ordonnées l'effort à fournir lors de l'apprentissage du Framework. On peut voir que la croissance est assez régulière à partir du moment où on s'en tient aux conventions. Par contre, les difficultés se ressentent lorsqu'on doit changer ces mêmes conventions afin d'être en adéquations avec les besoins et les contraintes du projet.

<sup>35</sup> Comparaison entre Marionette et Chaplin : <http://9elements.com/io/index.php/comparison-of-marionette-and-chaplin/>

## Angular.js

Angular possède une courbe d'apprentissage assez particulière. C'est un Framework qui fait également beaucoup de choses pour nous, cependant, il les fait de manière transparente. Il est donc très agréable de voir que certaines résultats voulus sont réalisés d'une façon qui peut paraître « magique » mais également très désagréable d'essayer de comprendre la logique qu'il y a derrière lorsqu'un problème survient. Beaucoup de témoignages<sup>36</sup> font référence à l'image ci-dessous pour qualifier l'apprentissage d'Angular.

*Courbe d'apprentissage d'Angular*



*My Feelings About AngularJS Over Time*

Source <http://www.bennadel.com/>

Contrairement à la courbe d'apprentissage d'Ember, l'axe des ordonnées ne montre pas réellement les difficultés rencontrées mais plutôt le ressenti que l'on peut percevoir au fur et à mesure de notre apprentissage.

<sup>36</sup> Un témoignage sur l'apprentissage d'AngularJS : <http://www.bennadel.com/blog/2439-my-experience-with-angularjs-the-super-heroic-javascript-mvw-framework.htm>

## Polymer.js

La manière dont Polymer est utilisé aura une grande importance sur la courbe d'apprentissage et la productivité. On peut se limiter à utiliser platform.js contenant les « shims » afin de n'utiliser que les APIs des Web Components dont nous avons besoin ou se servir de la librairie entière afin d'utiliser les éléments polymer. La librairie étant jeune, je n'ai pas réussi à trouver des retours d'expériences sur le long terme.

## React.js

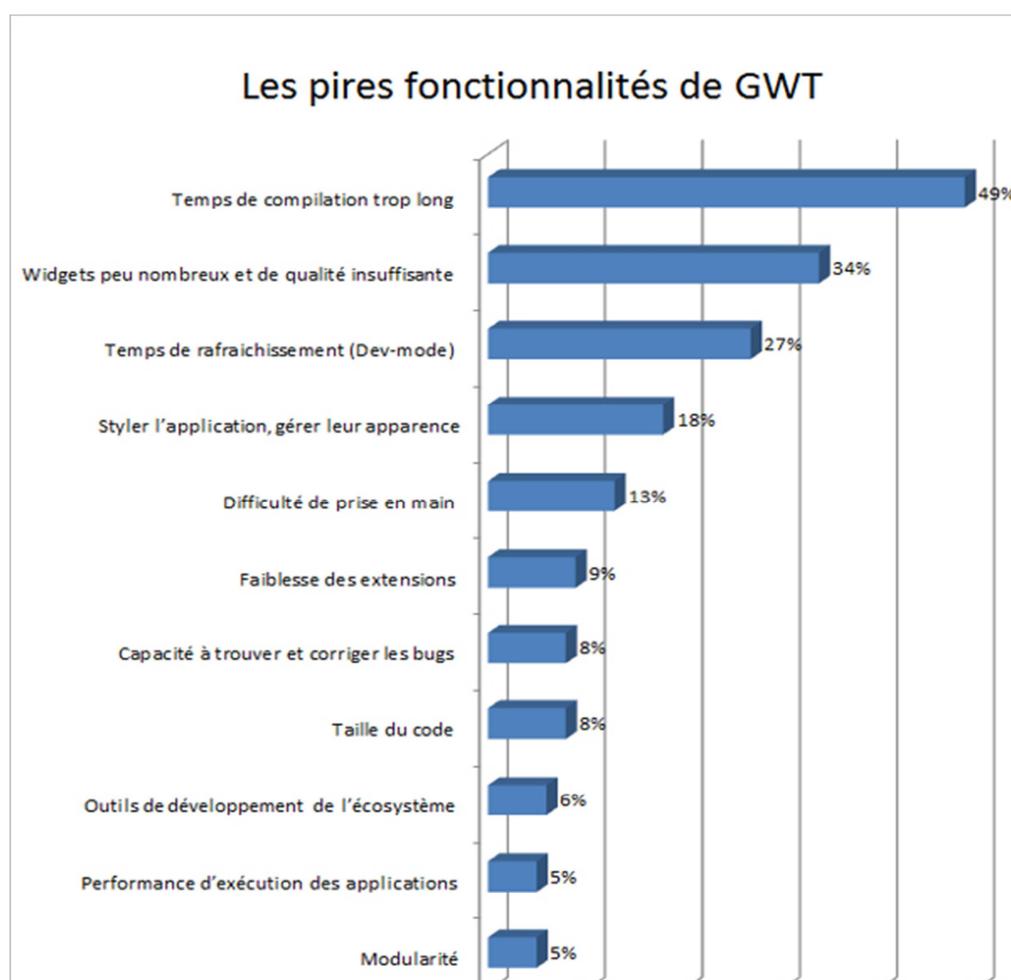
Concernant React, sa philosophie se distingue de ses concurrents car il ne fait pas partie des modèle MVW. Ses principes incitent à se focaliser sur le métier et à créer des objets immuables représentant une forte logique avec le besoin de l'utilisateur et les actions qu'il effectue. Ces méthodes obligent la création d'un nombre de classes plus important qu'un simple modèle MVC. La différence sur ces dernières et qu'elles répondent à un besoin précis et ne sont pas relier à différents endroits dans l'application comme le sont les classes d'un modèle MVC. On peut donc imaginer que la productivité ne se verra pas réellement sur le développement initial mais sur le long terme car grâce au modèle Flux, l'application est censée être beaucoup plus facile à comprendre et à maintenir.

## 2.4. Les Compilateurs

### 2.4.1. L'utilité des compilateurs

Les compilateurs JavaScript sont apparus par le constat que beaucoup de personnes pensent que nous avons de plus en plus besoin du JavaScript mais que son avenir ne se trouve pas dans le langage lui-même ou dans son support actuel par les navigateurs. C'est le seul langage interprété nativement par tous les navigateurs. L'idée des compilateurs n'est donc pas de le remplacer mais de s'appuyer dessus afin de garder uniquement les bons côtés du langage.

Cependant, il y a encore beaucoup de personnes aujourd'hui qui trouvent que les compilateurs n'apportent que des inconvénients. Qu'ils ne sont que des abstractions imparfaites du langage, demandant des connaissances supplémentaires pour ensuite voir apparaître encore plus de limitations que ne contient le JavaScript. Cette position est en partie due à la vision<sup>37</sup> qu'ont les développeurs sur un des premiers compilateurs JavaScript, GWT (Google Web Toolkit).



Source <http://www.journaldunet.com/>

<sup>37</sup> Sondage effectué par le Journal Du Net : <http://www.journaldunet.com/developpeur/outils/google-web-toolkit-l-avis-des-developpeurs/gwt-les-points-faibles.shtml>

Le JavaScript subit pourtant la même évolution qu'a subi à leur époque l'assembleur ou encore le C. Il possède de nombreux avantages qui lui ont permis de gagner en popularité. Cependant, l'utilisation que l'on en fait évolue et les éléments qui étaient vus comme des points forts du langage, deviennent peu à peu des limitations.

### Le JavaScript est trop permissif

La permissivité du langage nous oblige à accroître nos connaissances et à avoir recours à des conventions et des méthodes afin de produire du JavaScript de bonne qualité. Le problème est que ces conventions et méthodes changent d'un développeur à l'autre et d'un Framework à l'autre. Il est donc difficile de se mettre d'accord sur les choix que nous devons faire lorsqu'un projet démarre (choix des outils, librairies, Frameworks, conventions...).

### Problème de classes

Le JavaScript est un langage objet par prototypage. Beaucoup de concepts objets présents dans d'autres langages tels que l'encapsulation ou la spécialisation sont différents dans leur mise en œuvre, et donc moins facile d'accès. L'encapsulation est le fait de protéger certaines informations d'un objet. La spécialisation est le fait de pouvoir hériter d'un objet parent afin de réutiliser des informations (attributs) et des méthodes communes. Nous pouvons néanmoins simuler ces mécanismes mais le fait de concilier les deux est vu en JavaScript comme un anti-pattern parce que l'héritage impose d'avoir un couplage fort entre les objets et rentre en conflit avec l'encapsulation effectuée. Il y a donc une base très fragile sur la création de classes en JavaScript. Le meilleur moyen est donc d'éviter le plus possible l'héritage en JavaScript afin de favoriser la composition.

### Problème d'expression

Le JavaScript demande beaucoup plus de développement pour effectuer une même action comparé à d'autres langages, il y a donc une perte de productivité. L'idée de la programmation objet est de faciliter le développement ainsi que l'ajout de nouveaux comportements métier. Le JavaScript demandant beaucoup plus d'effort pour faire de l'objet, rencontre donc un problème d'expression car il est plus dur d'effectuer des changements sur le développement effectué.

### Problème de scalabilité

La scalabilité représente la capacité à s'adapter à des changements de grandeurs. En JavaScript, on peut donc rencontrer des problèmes de dépendances entre des versions des librairies externes en partie dus au fait qu'elles peuvent toutes interagir ensemble par défaut.

Les compilateurs sont donc là pour combler l'ensemble de ces faiblesses. Tout comme les Frameworks, le nombre de compilateurs présent est très grand<sup>38</sup>. J'ai choisi de vous présenter cinq compilateurs open source qui sont pour moi les plus prometteurs.

<sup>38</sup> Liste des compilateurs JavaScript : <https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS>

## 2.4.2. CoffeeScript



# CoffeeScript

Source <http://coffeescript.org/>

CoffeeScript a été créé en décembre 2009 par l'organisation DocumentCloud et plus particulièrement, Jeremy Ashkenas (créateur de Backbone et Underscore). C'est un petit langage qui s'appuie sur Ruby et Python afin de pouvoir facilement être écrit, lu et maintenu. Son argument principal est d'ailleurs que le JavaScript doit être lu principalement par les humains et exécuté par les ordinateurs. Il est simple de l'intégrer à un environnement JavaScript déjà existant parce qu'il crée du JavaScript.

CoffeeScript est le compilateur de référence pour les développeurs Ruby. Il est présent par défaut dans le Framework Ruby On Rails. De plus, il a eu une grande influence sur l'évolution d'ECMAScript ces dernières années. Son avantage face à certains de ses concurrents est qu'il ne fait que du JavaScript. Il se base sur les standards d'ECMAScript 3 pour la compilation afin de produire du JavaScript compatible avec un grand nombre d'environnement. Cette caractéristique fait que beaucoup d'outils utilisent CoffeeScript pour être développés. Les principaux sont Spine.js (Frameworks MVC inspiré par Backbone), Batman (Framework CoffeeScript), ExpectThat (librairie d'assertion pour CoffeeScript), Zombie.js (Framework de test) et CoffeeScript lui-même. CoffeeScript possède également de belles références<sup>39</sup> comme Trello, Airbnb, ou bien encore la version mobile de Basecamp.

En CoffeeScript, tout est expression. Il n'y a pas de point-virgules, d'accolades, de mots-clés tel que « function » et l'utilisation des parenthèses est beaucoup plus souples. CoffeeScript met également l'indentation au centre de ses intérêts afin de favoriser la lisibilité du code.

### Exemple de condition CoffeeScript et de conversion en JavaScript

```
mood = greatlyImproved if singing

if happy and knowsIt
  clapsHands()
  chaChaCha()
else
  showIt()

date = if friday then sue else jill
```

```
var date, mood;

if (singing) {
  mood = greatlyImproved;
}

if (happy && knowsIt) {
  clapsHands();
  chaChaCha();
} else {
  showIt();
}

date = friday ? sue : jill;
```

Source <http://coffeescript.org/>

<sup>39</sup> Références de CoffeeScript : <https://github.com/jashkenas/coffeescript/wiki/In-The-Wild>

CoffeeScript a de nombreux avantages syntaxiques permettant de gagner en productivité et de se concentrer sur les fonctionnalités à développer plutôt que sur les bonnes pratiques à mettre en œuvre. Il augmente le niveau de notre code, réduit les mauvais côtés du JavaScript et permet de faire beaucoup de choses que le JavaScript peut faire grâce aux nouvelles spécifications d'ECMA.

Cependant, il possède également quelques inconvénients :

- les commentaires simples qui pourraient être utiles lors de la lecture du JavaScript ne sont plus présents après la compilation ;
- certaines erreurs de syntaxe CoffeeScript peuvent produire un code JavaScript totalement différent et être difficile à débugger ;
- le JavaScript produit par CoffeeScript est, en général, plus important et moins lisible que celui que l'on aurait fait nous-même ;
- il n'est pas un standard et peut donc posséder des risques avec les futures fonctionnalités JavaScript.

La communauté autour de CoffeeScript étant importante, il existe des compilateurs alternatifs<sup>40</sup> permettant de rajouter des fonctionnalités au langage ou encore produire du code JavaScript plus réduit.

---

<sup>40</sup> Compilateur de la famille de CoffeeScript : <https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS#coffeescript-family-friends>

### 2.4.3. TypeScript

# TypeScript

JavaScript for tools

Source <http://blog.xebia.fr/>

TypeScript<sup>41</sup> a été ouvert au public en octobre 2012 et la version 1.0 est sortie il y a quelques mois. Il est développé par Microsoft avec Anders Hejlsberg (auteur de Turbo Pascal et architecte de Delphi et C#). Il se définit comme un sur-ensemble de JavaScript avec comme slogan : « TypeScript is JavaScript ». TypeScript fait du JavaScript et peut contenir directement du code JavaScript, mais l'inverse demande tout de même une compilation.

TypeScript intègre une partie des fonctionnalités de la future version d'ECMAScript 6 (classes modules, interface...) et rajoute par-dessus la possibilité d'utiliser un typage statique fort<sup>42</sup>. Le typage statique permet de réaliser des applications robustes en donnant la possibilité aux développeurs de connaître les types à utiliser. En JavaScript, la seule méthode que nous avions pour orienter les développeurs était d'indiquer le type à utiliser dans des commentaires. Avec TypeScript, si un type n'est pas respecté, le compilateur lèvera une erreur lors de la compilation et évitera ainsi toutes anomalies dans le code.

TypeScript ne souhaite pas être un langage à part entière mais un outil de développement pour le JavaScript. Il est d'ailleurs le compilateur qui se rapproche le plus du langage et compte suivre son évolution. L'avantage de TypeScript est donc de pouvoir utiliser les derniers fonctionnalités d'ES6, un typage fort et de le convertir en JavaScript qui sera compatible avec beaucoup de plate-forme. Lors de la compilation on peut choisir une conversion vers ES3 ou ES5 (ES pour ECMAScript) et utiliser des modules CommonJS ou AMD. Les modules permettent d'effectuer une séparation au niveau du code. Les modules CommonJS sont ceux utilisés côté serveur par node.js tandis que les modules AMD sont ceux présents du côté des navigateurs. Leurs syntaxes diffèrent, il est donc dur avec le JavaScript de partager du code client et serveur lorsqu'il y a des différences d'implémentations. TypeScript fait une abstraction sur ces différences et permet donc de partager du code client et serveur. La seule différence se fera au niveau de la compilation suivant les paramètres utilisés.

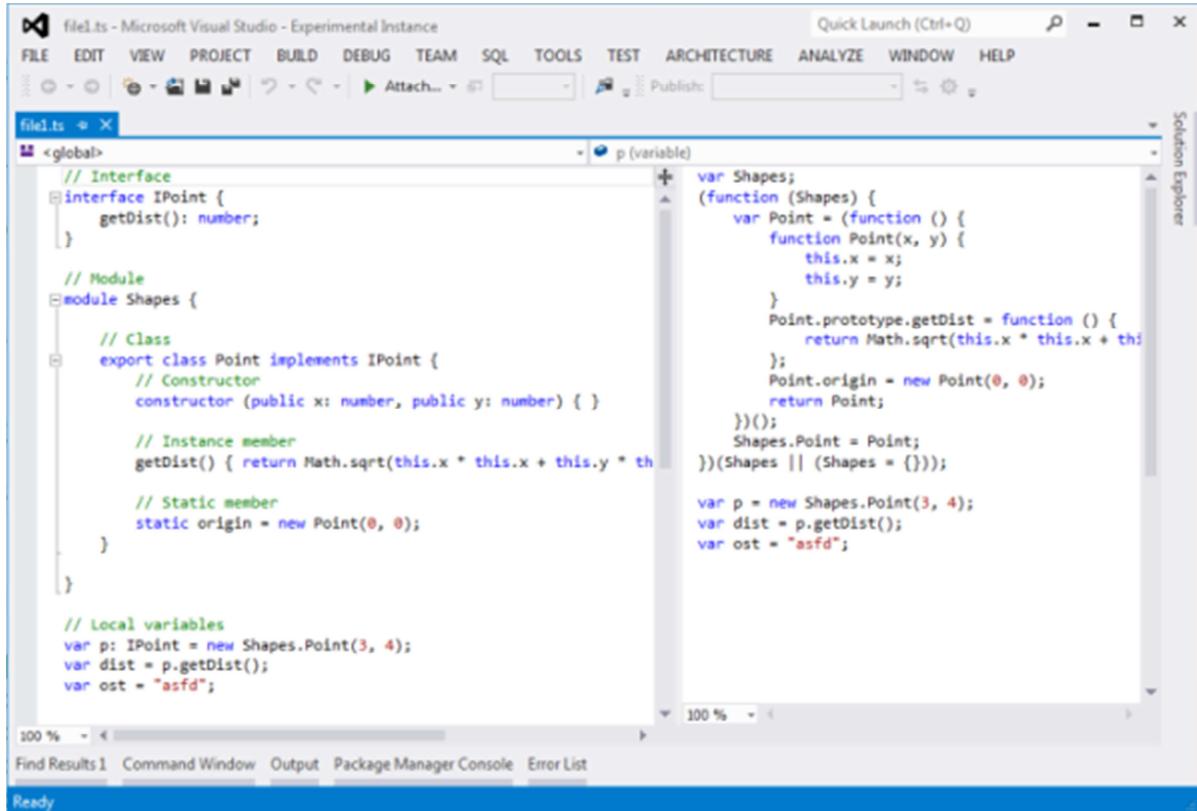
<sup>41</sup> Spécification de TypeScript :

<http://www.typescriptlang.org/Content/TypeScript%20Language%20Specification.pdf>

<sup>42</sup> Explication du typage statique est dynamique : <http://stackoverflow.com/a/1517670>

Le point fort de TypeScript est donc qu'il ne change pas la sémantique actuelle du langage mais améliore la rapidité et la qualité du code produit grâce au typage, à l'abstraction de certains concepts et à l'utilisation des nouvelles fonctionnalités du JavaScript.

### Utilisation de TypeScript dans Visual Studio



```
// Interface
interface IPoint {
    getDist(): number;
}

// Module
module Shapes {

    // Class
    export class Point implements IPoint {
        // Constructor
        constructor (public x: number, public y: number) { }

        // Instance member
        getDist() { return Math.sqrt(this.x * this.x + this.y * this.y); }

        // Static member
        static origin = new Point(0, 0);
    }
}

// Local variables
var p: IPoint = new Shapes.Point(3, 4);
var dist = p.getDist();
var ost = "asfd";
```

Source : <http://habrahabr.ru/>

TypeScript peut s'intégrer avec un grand nombre d'éditeur et d'IDE. Il est présent dans Visual Studio et a permis de réaliser l'éditeur HTML, CSS et JavaScript du nom de Monaco. De la même manière que CoffeeScript, TypeScript s'utilise lui-même pour son développement.

L'inconvénient que l'on peut trouver à TypeScript se trouve au niveau de la définition des types des librairies externes. Les définitions de types reposent sur la communauté et lorsque la communauté de la librairie en question est petite, il est possible que le type n'existe pas. Il est également possible qu'il y ait des problèmes entre la version de la librairie utilisée et la définition de type disponible. Un manager de définition de types est néanmoins disponible : TSD<sup>43</sup>.

<sup>43</sup> TSD : TypeScript Definition manager. Plus d'information ici : <http://definitelytyped.org/tsd/>

#### 2.4.4. Dart



Source : <http://yphiblog.blogspot.fr/>

Dart est un langage fortement typé, inspiré par Java et C++. Il a été rendu public par Google en 2011 et diffère un peu des deux précédents compilateurs car même s'il permet d'être compilé en JavaScript, il a pour vocation d'être exécuter directement dans une VM (machine virtuelle) afin de remplacer le JavaScript. La VM peut être utilisée côté serveur, côté client sur les navigateurs mais également être embarquée dans une application. On garde donc l'aspect multi-plateforme mais on perd l'intérêt d'utiliser des librairies ou Frameworks JavaScript externes.

En effet, Dart interagit avec des librairies JavaScript grâce au composant « dart:js ». Cependant, l'utilisation du JavaScript à l'intérieur de Dart est beaucoup plus fastidieuse que si on codait directement avec. Les objets Dart utilisés avec des librairies JavaScript doivent être obligatoirement convertis. Cela peut se faire automatiquement mais, dans certains cas, il faudra passer par un proxy<sup>44</sup>. Quant aux objets JavaScript, il faut changer le contexte et utiliser la fonction « callMethod » afin d'indiquer qu'on utilise une méthode qui ne provient pas de Dart. On se retrouve donc avec plus de code à écrire et on perd certains des avantages de Dart (comme le typage dynamique qui reste une fois le code compilé, contrairement à TypeScript).

Il est donc préférable de s'appuyer sur des éléments présents dans Dart comme Angular.dart et Polymer.dart. On peut voir Angular.dart comme une transition entre la version 1.x d'Angular et la future version 2. Dans Angular.dart il n'y a plus de directives (qui étaient des éléments posant de plus en plus de problèmes) mais à la place, il y a des « @component » et « @decorator ». Les « decorators » ajoutent des fonctionnalités aux éléments HTML tandis que les « Components » permettent d'en créer d'une manière qui se rapproche étroitement des “Web Components” de la spécification d'HTML.

Avec Dart, beaucoup de choses sont simplifiées. Le typage est très utile pour repérer plus facilement des erreurs. Les évènements, fonctions de rappel et les APIs HTML5 peuvent être utilisés beaucoup plus simplement. On pourrait tout de même se demander pourquoi Google souhaite remplacer le JavaScript par Dart. D'autres ont essayé de le faire et n'y sont pas arrivé.

<sup>44</sup> **Proxy** : composant servant intermédiaire permettant de faciliter et/ou surveiller des échanges

En fait, contrairement aux autres, Dart essaye de changer les problèmes fondamentaux du JavaScript relatifs à la performance. Pour Google, le JavaScript continuera d'évoluer et de combler ses lacunes mais Dart a vu le jour afin de ne pas perdre de temps et concurrencer les applications mobiles. L'HTML5 est apparu et a totalement détrôné les alternatives comme Flash et Silverlight. Cependant, les performances sur mobiles ont bien plus évolué que celles du Web et du JavaScript. L'idée est donc de redevenir des concurrents sérieux grâce à des technologies provenant du Web.

### Résultat de différents Benchmark en JS, Dart et Dart2js

Benchmark	v8	dart	dart2js	dart	dart2js
<a href="#">DeltaBlue</a>	284.86	363.14	186.63	127.48%	65.52%
<a href="#">Richards</a>	400.10	565.72	279.22	141.39%	69.79%
<a href="#">NBody</a>	15945.00	17436.50	10936.50	109.35%	68.59%
<a href="#">BinaryTrees</a>	9.02	9.33	8.24	103.49%	91.38%
<a href="#">Mandelbrot</a>	169.08	167.92	138.36	99.31%	81.83%
<a href="#">Fannkuch</a>	3458.50	4202.00	3172.00	121.50%	91.72%
<a href="#">Meteor</a>	6.68	5.96	2.25	89.29%	33.73%
<a href="#">BubbleSort</a>	25248.51	27160.00	15850.50	107.57%	62.78%
<a href="#">Fibonacci</a>	9156.00	13570.50	9405.50	148.21%	102.72%
<a href="#">Loop</a>	34392.03	35560.00	35302.50	103.40%	102.65%
<a href="#">Permute</a>	11084.50	15921.50	7584.00	143.64%	68.42%
<a href="#">Queens</a>	118474.98	184505.01	103320.00	155.73%	87.21%
<a href="#">QuickSort</a>	17138.49	17723.50	9771.00	103.41%	57.01%
<a href="#">Recurse</a>	13990.50	19994.50	14439.50	142.91%	103.21%
<a href="#">Sieve</a>	102273.54	117566.50	106883.50	114.95%	104.51%
<a href="#">Sum</a>	74409.74	60180.50	75387.00	80.88%	101.31%
<a href="#">Tak</a>	3062.50	4731.50	2487.00	154.50%	81.21%
<a href="#">Takl</a>	8917.50	17101.00	8941.00	191.77%	100.26%
<a href="#">Towers</a>	4915.50	5559.00	3232.50	113.09%	65.76%
<a href="#">TreeSort</a>	7043.50	8672.50	5417.00	123.13%	76.91%
Geo. mean	4009.51	4855.15	3136.27	121.09%	78.22%

Source : <https://www.dartlang.org>

On remarque effectivement sur ces Benchmarks<sup>45</sup>, que Dart est bien plus performant que le JavaScript. Cependant les statistiques de dart2js quand la VM n'est pas présente sont bien moins intéressantes.

<sup>45</sup> Plus d'informations ici : <https://www.dartlang.org/performance/>

Étant donné que Google possède le moteur JavaScript le plus performant (V8), ils sont sûrement les mieux placés pour déterminer jusqu'où peuvent aller les performances du JavaScript à l'intérieur d'un navigateur et jusqu'où il est possible qu'elles aillent à l'avenir. Dart possède tout de même un gros point négatif. Les navigateurs concurrents à Chrome (Internet Explorer, Firefox, Opera et Safari entre autres) ne comptent pas intégrer nativement la machine virtuelle Dart<sup>46</sup>. De plus, la VM étant open source, l'intégration de cette dernière dans les autres navigateurs reposera sur la communauté et il n'y a actuellement aucun plugin ou module permettant d'utiliser la VM dans ces navigateurs. Les performances en JavaScript étant bien moins intéressantes, l'utilisation de Dart pour une application « multi-browser » n'a que peu d'intérêt. Dart peut, par contre, être un choix judicieux pour une application Chrome ou multi-plateforme avec la VM embarquée.

---

<sup>46</sup> Propose de Brendan Eich (créateur de JavaScript) :  
<https://news.ycombinator.com/item?id=2982949>

#### 2.4.5. Sweet.js

Lorsqu'on recherche un compilateur, il arrive fréquemment que l'on regrette le manque d'une ou plusieurs fonctionnalités qui pourraient être intéressantes. Si aucun ne correspond à nos attentes, il nous reste trois cas de figures :

- Créer notre propre compilateur à partir de zéro grâce à des langages comme canopy<sup>47</sup>. Seulement, il y a peu de chance d'arriver à quelque chose qui concurrence les compilateurs existants.
- Utiliser des petits compilateurs répondants à des besoins spécifiques. Par exemple Traceur<sup>48</sup> est un compilateur ECMAScript 6 vers ES5. Il permet d'être compatible avec un maximum de navigateurs et prévoir une migration sur des futures versions de Frameworks dont Angular2 (qui se basent déjà sur ces standards).
- Sinon, il nous reste Sweet.js<sup>49</sup> qui permet de créer des macros.

Les macros transforment du code en définissant des expressions syntaxiques. Ils sont déjà disponibles en C++, Scala et dans des nouveaux langages comme Julia<sup>50</sup> ou encore Rust<sup>51</sup>. Il y a également une proposition<sup>52</sup> d'adoption des macros dans une future version d'ECMAScript (ES8). L'idée est donc d'éviter de devoir utiliser différents compilateurs (ce qui pourrait être source de conflits), mais d'en avoir un seul qui répondrait aux besoins propres d'un projet.

Avec les macros, il y a à la fois le code et le compilateur qui sont modulaires. L'utilisation des macros peut servir de polyfills pour des fonctionnalités d'ECMAScript mais également éviter la redondance sur du code métier. Les macros permettent donc de gagner en productivité grâce à une nouvelle manière de développer. On pourrait croire que la création de macros en JavaScript serait source de conflits par rapport à la portée des variables en JavaScript. Mais avec Sweet, il n'y a pas de problème de porter de variable car il respecte le concept de macro « hygiénique » ajoutant un « scope » (cadre défini) sur les variables utilisées.

<sup>47</sup> Canopy : <http://canopy.jcoglan.com/>

<sup>48</sup> Introduction à Traceur : <https://egghead.io/lessons/intro-to-es6-and-traceur-js>

<sup>49</sup> Projet de Mozilla : <http://sweetjs.org/>

<sup>50</sup> Julia : <http://julialang.org/>

<sup>51</sup> Rust : <http://www.rust-lang.org/>

<sup>52</sup> Plus d'information à partir de cet article : <http://www.2ality.com/2011/09/es6-8.html>

### Macro définissant les classes d'ECMAScript 6

```
// Define the class macro here...
macro class {

    rule {

        $className {
            constructor $cparams $cbody
            $($mname $mparams $mbody) ...
        }

    } => {

        function $className $cparams $cbody

        $($className.prototype.$mname
            = function $mname $mparams $mbody; ) ...

    }

}

// And now classes are in JavaScript!
class Person {
    constructor(name) {
        this.name = name;
    }

    say(msg) {
        console.log(this.name + " says: " + msg);
    }
}
var bob = new Person("Bob");
bob.say("Macros are sweet!");
```

Source : <http://sweetjs.org/>

#### 2.4.6. ClojureScript



Source : <http://code.tutsplus.com/>

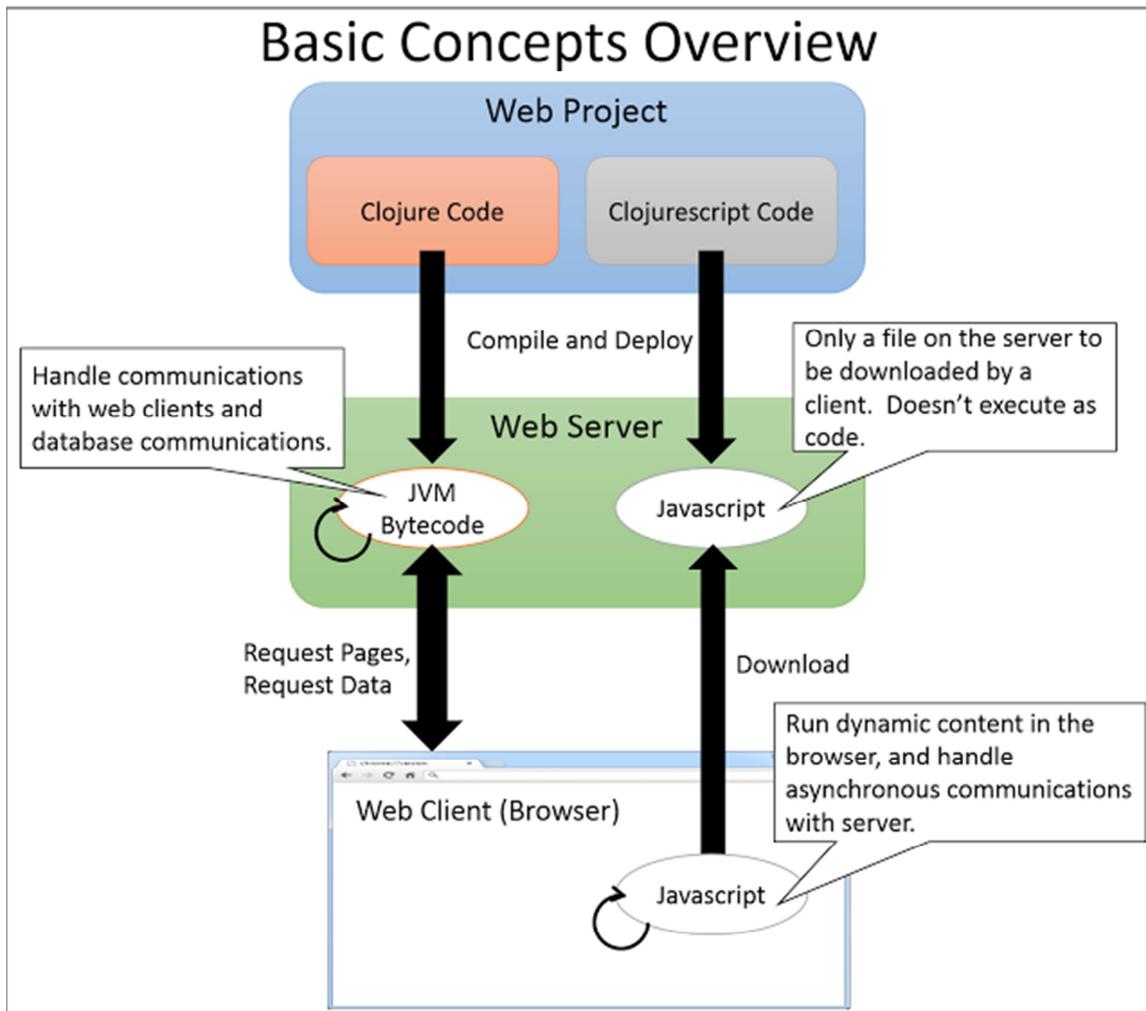
Avant de présenter ce compilateur, un retour en arrière de quelques années s'impose afin de parler de Lisp (le langage précurseur de Clojure et ClojureScript). Lisp a été créé en 1957 par John McCarthy. John McCarthy est également l'inventeur des interpréteurs, de la programmation « haut niveau », fonctionnelle, dynamique et de beaucoup d'autres choses. Il est devenu un des pionniers de l'intelligence artificielle notamment au travers de ce langage. C'est un des premiers langages de programmation à être multi-plateforme et à introduire le concept de REPL<sup>53</sup> (Read-Eval-Print-Loop).

Ce langage est revenu au goût du jour en 2007 avec l'apparition de Clojure qui utilise la syntaxe de Lisp afin de générer un bytecode<sup>54</sup> interprété par les JVM (machine virtuelle de Java). Cette syntaxe permet, entre autres, de programmer différemment en mettant au centre des intérêts la programmation fonctionnelle, les objets immuables ainsi que les macros. Le problème avec les JVM est qu'elles ne peuvent pas s'utiliser dans tous les cas de figures. ClojureScript a donc fait son apparition en 2011 comme compilateur JavaScript avec pour destination le Web et les mobiles.

Contrairement à CoffeeScript qui se rapproche de Ruby, ClojureScript utilise la même syntaxe que Clojure. Tout projet Java reposant sur Clojure a donc un certain intérêt à se tourner vers ClojureScript pour le développement JavaScript afin de rester en cohérence au niveau de l'environnement et des compétences techniques.

<sup>53</sup> **REPL** : Read-Eval-Print-Loop. Terme utilisé pour représenter les langages interactifs haut niveau comme Lisp.

<sup>54</sup> **Bytecode** : code intermédiaire entre les instructions machine et le code source



Source <http://www.clojured.com/>

ClojureScript est jeune. La communauté est encore petite, le langage continue d'évoluer et on peut avoir des difficultés à trouver de la documentation sur Internet. De plus, la syntaxe peut être déroutante pour les personnes qui n'ont jamais utilisé Clojure ou tout autre langage se rapprochant de Lisp. Cependant sa popularité augmente et il possède déjà des outils<sup>55</sup> très intéressants.

### Leiningen

Juste devant ClojureScript, Leiningen est le plus gros projet existant autour de Clojure. C'est un outil possédant un grand lot de fonctionnalités telles que :

- l'initialisation d'un projet Clojure ;
- la gestion des différentes dépendances ;
- la réalisation des compilations et déploiements ;
- le lancement des tests et de tâches personnalisés ;
- il va même jusqu'à publier des librairies sur les dépôts de Clojars<sup>56</sup>

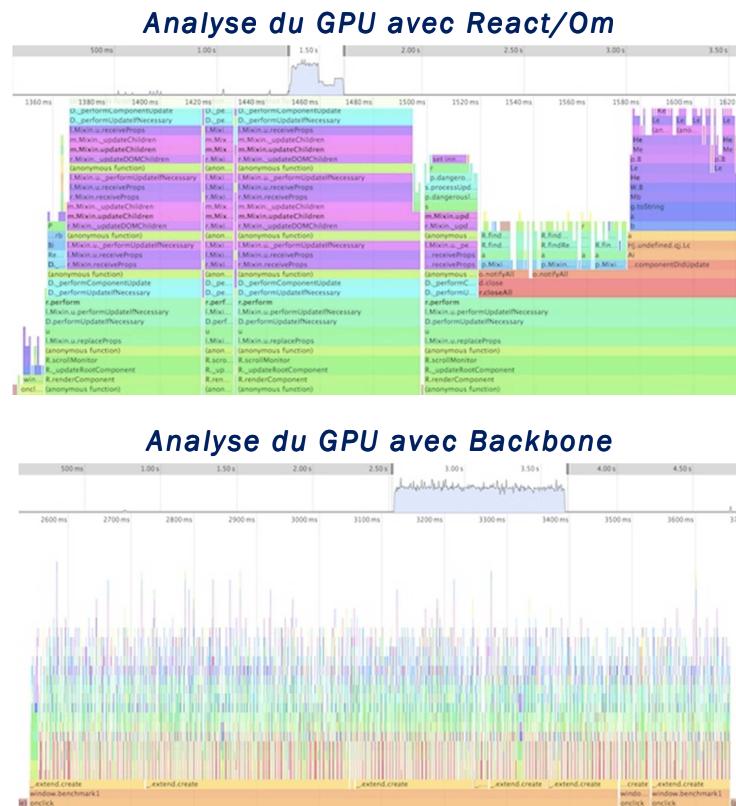
<sup>55</sup> Article décrivant différents outils pour ClojureScript : <http://www.clojured.com/2013/04/writing-clojureclojurescript-web.html>

<sup>56</sup> Centralisation de librairie Clojure accessible ici : <https://clojars.org/>

Leiningen dispose également de plugins. Le principal par rapport à ClojureScript est lein-cljsbuild. Ce dernier permet de compiler le JavaScript de manière automatique à chaque modification du code, rendant le développement beaucoup plus fluide et rapide.

## Om

Il y a beaucoup de ressemblances entre la philosophie de ClojureScript et React. Les deux outils utilisent la programmation fonctionnelle, les structures de données immuables, la composition, les états... Om a donc naturellement été créé afin de pouvoir utiliser React avec ClojureScript. Ces deux outils ensemble annoncent des grands gains de performances sur la gestion des vues allant jusqu'à être deux fois plus rapide que React seul.



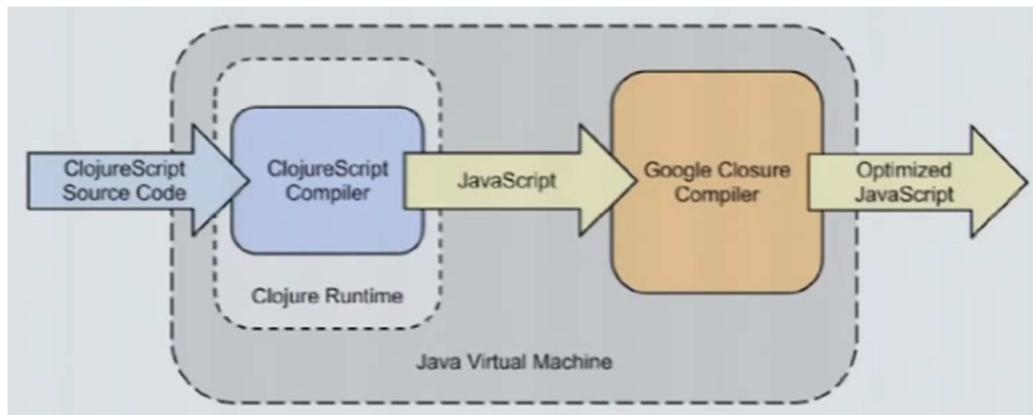
Source : <http://swannodette.github.io/>

Les deux graphiques ci-dessus montrent un benchmark<sup>57</sup> sur l'utilisation GPU du navigateur suivant les différentes actions effectuées en JavaScript pour l'application « TodoMVC ». Ils sont issus de l'outil de développement Chrome. On peut voir que React/Om effectuent des modifications beaucoup plus optimisées sur le DOM et cela est dû à sa gestion des templates « atypique » par rapport à ses concurrents.

<sup>57</sup> Plus d'information sur le test effectué ici : <http://swannodette.github.io/2013/12/17/the-future-of-javascript-mvcs/>

## Google Closure

Google Closure est un deuxième compilateur que Clojure utilise. Il récupère le JavaScript compilé par ClojureScript afin de le retravailler. L'architecture ci-dessous montre le fonctionnement de ces deux compilateurs :



Source: <https://www.youtube.com/>

Google Closure n'a de son côté aucune dépendance avec Clojure mais ce dernier l'utilise parce qu'il est le compilateur le plus performant. Il contient trois modes de compilation :

- le mode « white space only » : effectue des petites optimisations comme la suppression de commentaires d'espaces et de ponctuations inutiles ;
- le mode « simple optimizations » : va aller plus loin en renommant les variables et fonctions ;
- le mode « advanced optimizations » : retravaille entièrement le code afin d'être à la fois plus léger et plus performant. Il peut par exemple supprimer du code mort et remplacer des fonctions par le résultat d'exécution.

### Exemple de changement effectué par le mode avancé de google closure

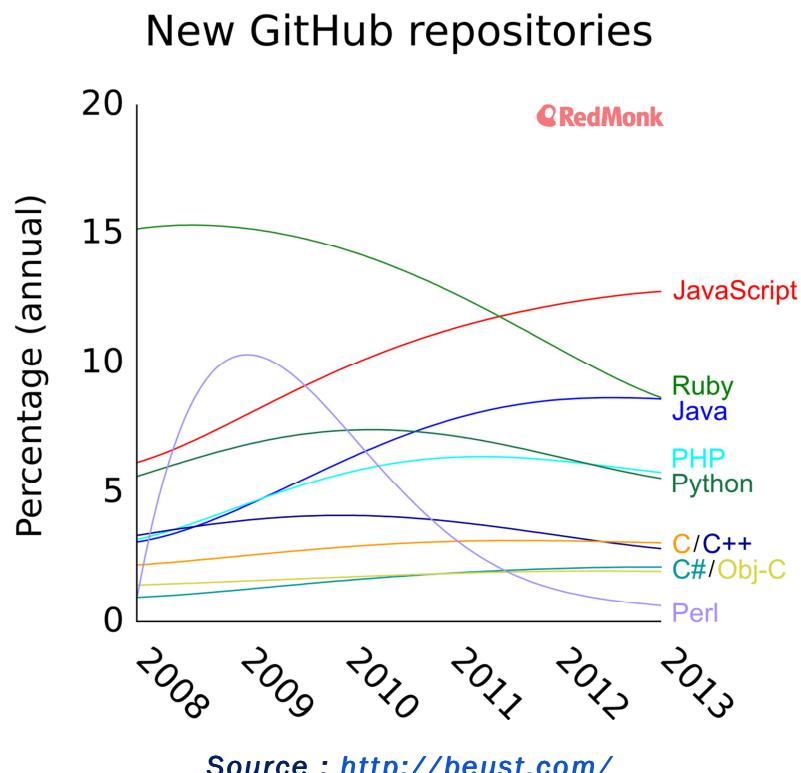
```
Original source:  
  
function doStuff(alpha, beta) {  
    var sum = alpha + beta;  
    alert("The sum of " + alpha + " and " +  
        beta + " is " + sum);  
}  
  
doStuff(7, 5);  
  
Advanced optimization mode:  
  
alert("The sum of 7 and 5 is 12");
```

Source : <https://www.youtube.com/>

### III. La révolution apparue avec node.js

#### 3.1. La réponse à de réels besoins

Nous venons d'évoquer beaucoup de points négatifs sur le JavaScript et des possibles solutions qu'il existe actuellement pour les combler. Néanmoins, ce langage contient plus d'atouts que de défauts, lui accordant de loin la première place dans les langages les plus actifs dans la communauté open source.



Node.js est apparu en 2009 et a été créé par Ryan Lienhart Dahl. Ce n'est ni un langage, ni un Framework, mais une plate-forme bas niveau permettant d'interagir côté serveur avec des fichiers, des bases de données et qui va même jusqu'à contenir un serveur HTTP intégré. Il utilise le moteur JavaScript de Google V8 et permet donc de s'appuyer sur les standards d'ECMAScript 5 sans se préoccuper des problèmes de compatibilité que l'on peut retrouver du côté des navigateurs. Il possède les mêmes forces et faiblesses que le JavaScript. Node.js n'est pas fait pour construire des grosses applications serveur demandant des calculs intensifs mais il est un choix judicieux pour les applications à fort trafic effectuant beaucoup de requêtes côté serveur.

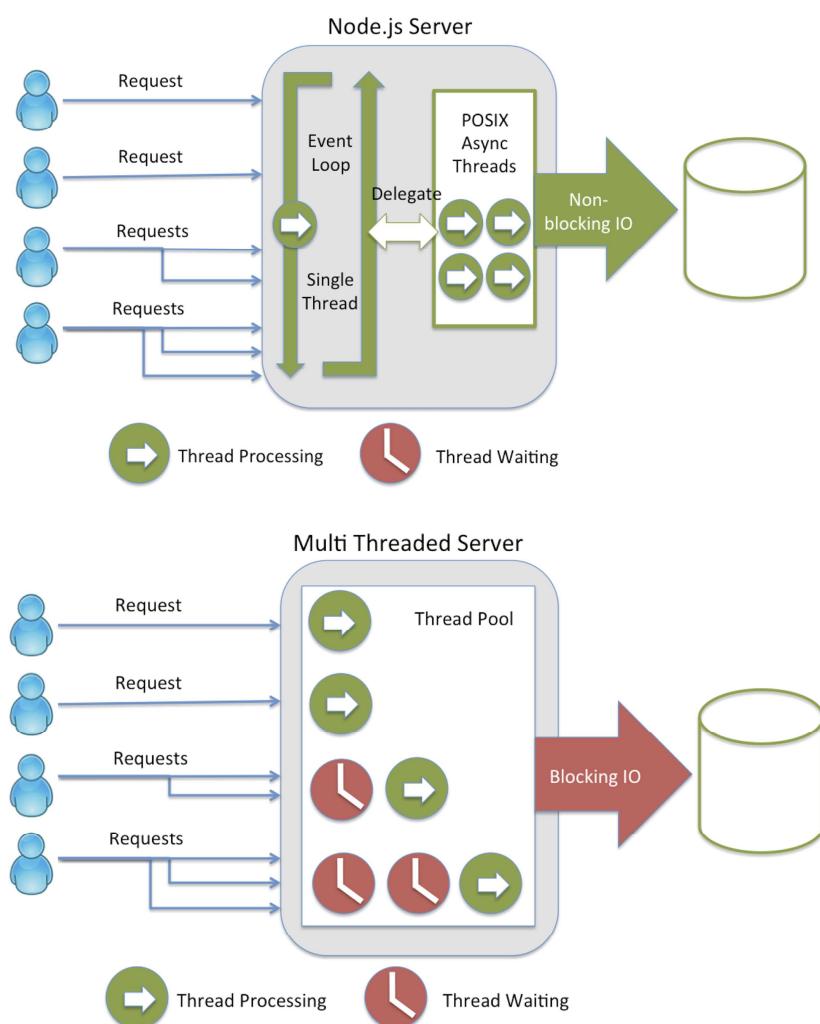
Node.js peut permettre de créer rapidement des API s'appuyant sur les principes REST. Des Frameworks existent avec pour but premier de répondre à ce besoin mais également des services<sup>58</sup> comme Strongloop ou d'autres un peu plus spécialisés dans des domaines particuliers comme Prismatic pour la gestion de contenus et Moltin orienté sur l'Ecommerce.

<sup>58</sup> Liste non exhaustif de service que propose des modules node.js : <https://github.com/joyent/node/wiki/modules#web-frameworks-full>

### 3.1.1. Le temps réel

Le besoin principal auquel répond node.js est le temps réel<sup>59</sup>. Le JavaScript a permis d'augmenter considérablement les interactions utilisateurs et les requêtes envoyées aux serveurs. Bien que le Java ou encore le PHP possède des méthodes pour réceptionner ces requêtes, ils ne sont pas adaptés pour y répondre sur le Web quand leur volume augmente. Pas besoin de recourir à la création d'une application multi-thread (processus multiples) pour prendre en charge en même temps des utilisateurs différents. Cette technique souvent effectuée en Java demande beaucoup de ressources serveur qui augmente au fur et à mesure en fonction du nombre de requêtes.

Avec node.js, il n'y a qu'un seul thread. Il embarque les avantages de la programmation événementielle et asynchrone qui sont des particularités très connues du JavaScript. Celles-ci permettent de proposer un temps de réponses équivalent quel que soit le trafic de l'application car contrairement à ses concurrents, chaque requête est non bloquante.

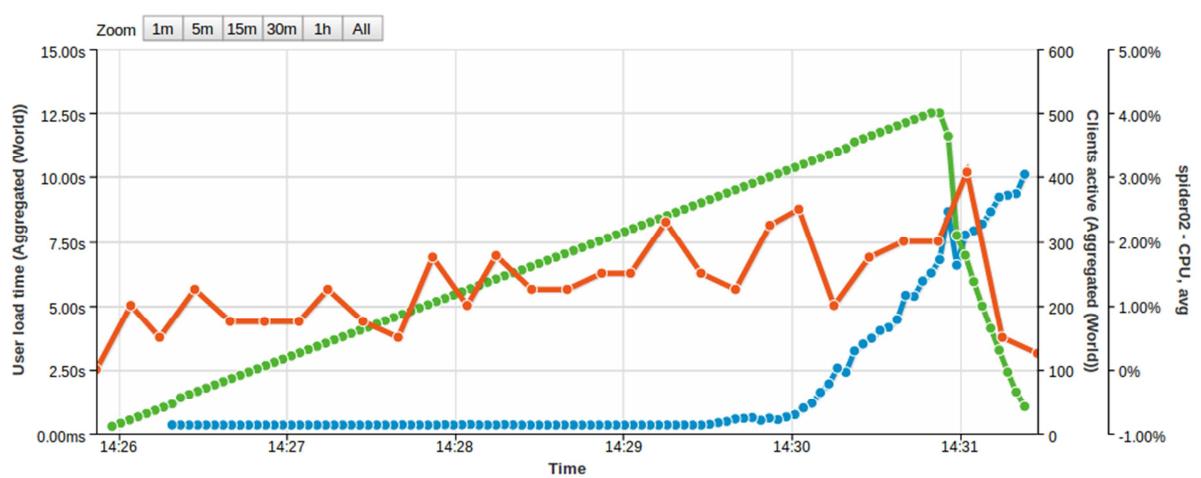


Source <http://strongloop.com/>

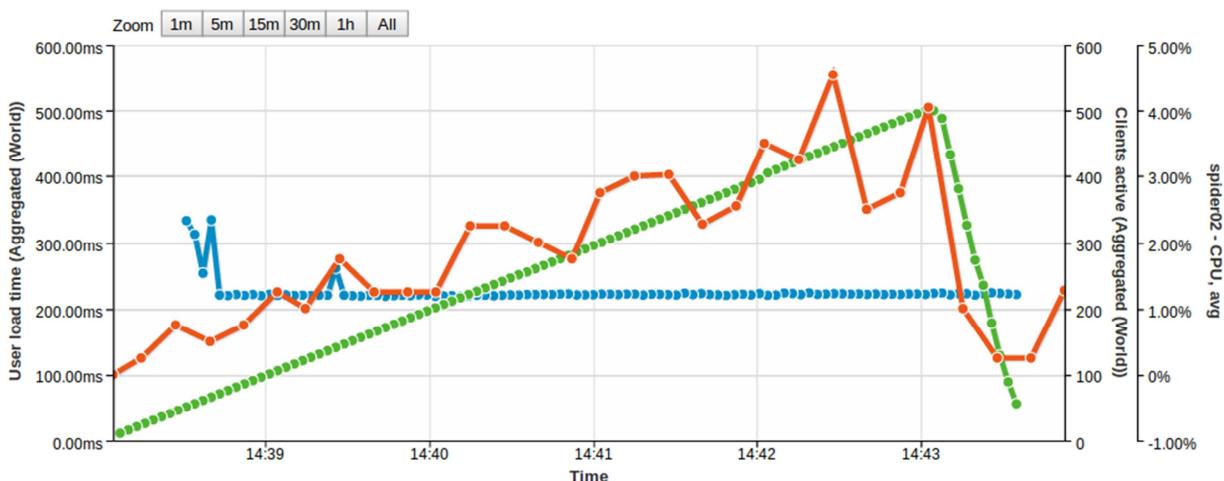
<sup>59</sup> **Temps réel** : système avec des contraintes temporelles sur les temps de réponses

Bien qu'il soit encore jeune, node.js possède déjà beaucoup de références telles que Facebook, Paypal, Groupon, Yahoo et bien d'autres encore. Grâce à son ratio performance / taux d'utilisation, il devient la solution idéale pour répondre à un fort trafic comme les chats ou les moteurs de recherches. Les graphiques ci-dessous montrent des tests de performances effectués en PHP et en node.js. La ligne verte représente le nombre d'utilisateurs simultanés, la ligne bleue représente le temps de réponse; et la ligne rouge représente l'utilisation CPU du serveur.

### Test de performance réalisé en PHP



### Test de performance en node.js



Source : <http://blog.watopa.sh/>

Node.js ne s'arrête pas là, des retours d'expériences indiquent que le passage à cette plate-forme a permis d'avoir des gains de productivité importants dus au fait qu'il y a une cohérence entre le langage utilisé côté Front et Back, le JavaScript. Les mêmes développeurs peuvent ainsi gérer l'application entière et réutiliser du code client côté serveur, avec les nuances déjà évoquées sur les différences de compilation entre ces deux environnements.

Pour toutes les raisons déjà évoquées, il est également possible d'utiliser des compilateurs JavaScript. Plus le développement Backend sera important plus la nécessité d'en utiliser et/ou de recourir à un Framework node.js se fera ressentir. Node.js est également étroitement lié aux bases de données NoSQL<sup>60</sup>. Que cela soit pour gérer des bases temporaires grâce à Redis ou des gros volumes de données avec MongoDB, node.js reste une solution de choix pour une application avec des objectifs de temps de réponses et de trafic.

Il existe des solutions Saas<sup>61</sup> (logiciel en tant que service) utilisant node.js. Il y a par exemple Firebase<sup>62</sup>, Pubnub<sup>63</sup> ou encore Pusher<sup>64</sup> qui proposent des solutions Backend par le Cloud pour des applications à temps réel. Le temps réel devient un enjeu majeur de nos jours et node.js se trouve être le mieux placé pour y répondre. On l'utilise par exemple pour les jeux vidéo, les notifications, les messageries instantanées, les éditeurs en lignes mais également les outils analytiques allant de Google Analytics jusqu'à des plate-formes CRM<sup>65</sup> comme Woopra<sup>66</sup>,

---

<sup>60</sup> **NoSql** : Not only SQL. Base de données non relationnelles

<sup>61</sup> **SaaS** : Software as service ou logiciel en tant que service en français

<sup>62</sup> Site officiel de Firebase : [www.firebaseio.com/](http://www.firebaseio.com/)

<sup>63</sup> Site officiel de Pubnub : <http://www.pubnub.com/>

<sup>64</sup> Site officiel de Pusher : <http://pusher.com/>

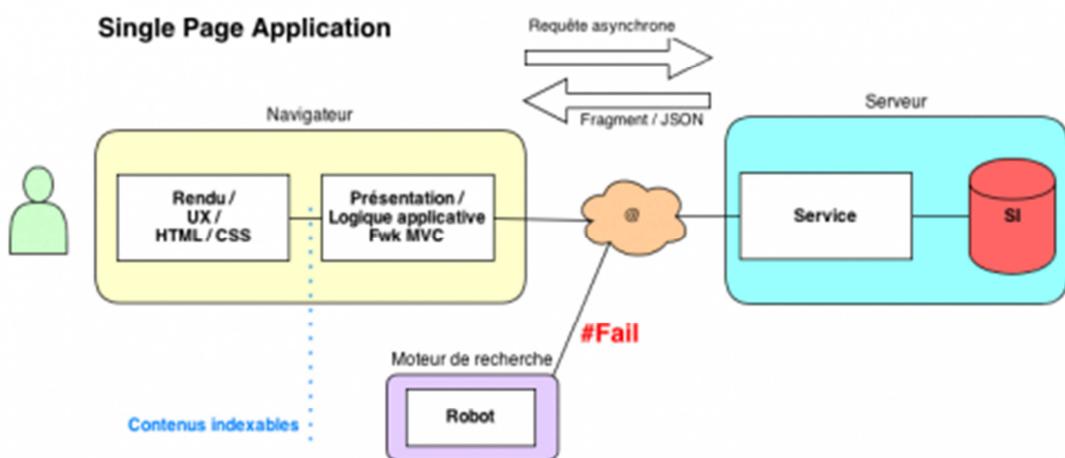
<sup>65</sup> **CRM** : Gestion des relations client

<sup>66</sup> **Woopra** : outil CRM en temps réel

### 3.1.2. Le référencement

La problématique du référencement et du JavaScript a longtemps été d'actualité et continue de l'être pour certains moteurs de recherche. La démocratisation des Frameworks JavaScript et des modèles côté client ont eu pour effet de devoir générer le contenu HTML grâce au JavaScript, rendant des pages vides pour les robots d'indexations des moteurs de recherches. Depuis mai 2014, Google peut prendre en compte le JavaScript<sup>67</sup> mais d'autres moteurs de recherche comme Yahoo et Bing ont encore un train de retard. Ce problème est donc toujours d'actualité et il faut y penser dès le commencement d'un projet sous peine de devoir changer considérablement l'architecture de l'application.

#### **Problématique du référencement et des moteurs de recherche**



Source : <http://blog.octo.com/>

Nous n'avons pas encore parlé de « Simple Page Application » (SPA) qui est pourtant un terme répandu maintenant. Il fait référence à un site Internet disposant d'une unique page dont le contenu change dynamiquement grâce à un Framework JavaScript côté Front.

#### Le cloaking

Le cloaking représente le fait de montrer un contenu différent selon le type de visiteur. Le mot a vu le jour suite à des abus qui sont apparus pour référencer des sites dont les contenus différaient totalement entre les utilisateurs et les robots d'indexation des moteurs de recherches. Comme beaucoup de choses dans le SEO (Search Engine Optimization), les frontières entre les bonnes et les mauvaises pratiques pour référencer un site ne sont pas très claires. Dans l'image qui suit, on peut voir les niveaux de Cloaking allant du « white hat » (stratégies d'indexation autorisées par Google) au « black hat » (stratégies interdites et sanctionnées par Google). Ce tableau n'est pas officiel et est le résultat d'une analyse effectuée en 2008 par Rand Fishkin.

<sup>67</sup> Plus d'information ici : <http://www.business2community.com/seo/googles-crawler-now-understands-javascript-mean-0898263#!bLJU1r>

## Search Engine Cloaking Scale

Pearly White	<b>TACTICS PERMITTED:</b> Cookie Detection, Javascript <b>GOALS/INTENT:</b> Landing Page Optimization / Content Based on User Login/Actions
Near White	<b>TACTICS PERMITTED:</b> Above + User Agent <b>GOALS/INTENT:</b> Geolocation / Browser-Type Targeting / Minimizing Bot Bandwidth Usage
Light Gray	<b>TACTICS PERMITTED:</b> Above + User Agent/IP Lookup <b>GOALS/INTENT:</b> Redirecting Link Juice Appropriately / Showing Behind-the-Wall Content
Dark Gray	<b>TACTICS PERMITTED:</b> Any & Everything <b>GOALS/INTENT:</b> Showing Less SEO'd Versions of Content / Misdirecting Link Juice
Solid Black	<b>TACTICS PERMITTED:</b> Any & Everything <b>GOALS/INTENT:</b> Misleading Users to Content That Doesn't Fit Their Search Query

Source : <http://moz.com/>

Le problème dans le référencement est qu'il y a beaucoup de techniques qui fonctionnent que Google ne met pas en avant. On pourrait supposer que la raison est, en partie, que les algorithmes d'indexation restent secret et évoluent tous les jours. Il est donc très difficile de savoir si on a dépassé la frontière et que l'on risque des sanctions par Google ou non. Il existe même des professionnels du SEO qui utilisent des techniques « Black Hat » afin d'améliorer le référencement de leurs clients.

Pour le Cloaking un article<sup>68</sup> sur un blog Australien de Google indiquait accepter l'utilisation de navigateur « headless » que nous verrons par la suite. Matt Cutts quant à lui (qui travaille chez Google est fourni des informations officielles) a fait une vidéo<sup>69</sup> fin 2011 indiquant que le Cloaking représentait le fait de différencier le rendu entre les utilisateurs et les robots d'indexation. Pour lui, un contenu différent pour des utilisateurs (géolocalisation, device...) n'est pas du cloaking car il ne concerne pas spécifiquement les moteurs de recherche.

<sup>68</sup> Lien de l'article : <http://moz.com/blog/white-hat-cloaking-it-exists-its-permitted-its-useful>

<sup>69</sup> Vidéo de Matt Cutts : <https://support.google.com/webmasters/answer/66355>

Cependant la frontière est faible quand on est dans l'obligation de servir d'une manière différente le même contenu pour des moteurs de recherche ne pouvant pas interpréter la page. Il existe des sites connus<sup>70</sup> qui se rapprochent de cette frontière et utilisent des techniques que l'on pourrait considérer comme du Black Hat. Certains vont enlever les pubs, d'autres vont accepter les robots et obliger les utilisateurs à s'enregistrer sur leurs sites afin de pouvoir visualiser le même contenu. Ces astuces peuvent fonctionner maintenant et être sanctionnées par Google dans un avenir plus ou moins proche.

### La balise <noscript>

Cette balise permet d'insérer des informations pour les personnes n'ayant pas de JavaScript activé sur leur navigateur. Autrement dit un pourcentage se rapprochant de 0 maintenant que les dernières versions des navigateurs ont enlevé cette option de désactivation. Toutefois, grâce à des paramètres et/ou des extensions, il reste possible de désactiver le JavaScript et pour rappel Bing et Yahoo ne l'interprète toujours pas. Il y a aussi de nouveaux périphériques qui peuvent ne pas avoir de moteur JavaScript (objets connectés, TV...). Une solution sûre est donc d'insérer le contenu utilisé par les templates JavaScript dans une ou plusieurs balises « noscript ».

Le rapport avec node.js ? node.js peut réutiliser le JavaScript du Front afin de générer le contenu des templates du côté du serveur. Il suffit ensuite d'insérer ce contenu dans la balise <noscript> afin de pouvoir répondre à tous les utilisateurs ne disposant pas de JavaScript.

### Les « hashbangs »

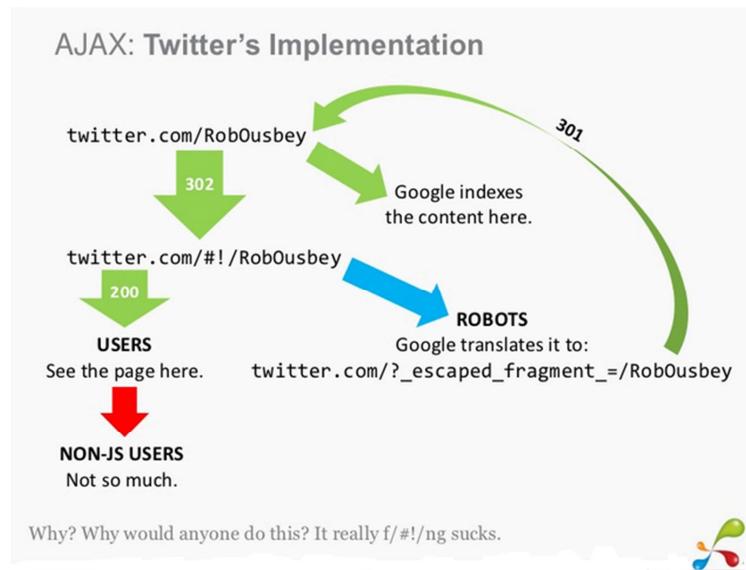
Les hashbangs ou encore « escape fragment » font encore parties des recommandations de Google et pourtant, ils sont déconseillés par un grand nombre d'acteurs<sup>71</sup> les aillant utilisés. Les raisons sont nombreuses :

- solution temporaire jusqu'à l'adoption d'HTML5 ;
- temps de chargement des pages plus longue car le contenu est chargé par le JavaScript pour l'utilisateur et par une redirection pour le robot ;
- perte d'intérêt de l'url sur le référencement car il y a des redirections qui sont effectuées ;
- une fois les URLs référencés, il est difficile de changer de méthode sous peine de perdre le SEO réalisé.

<sup>70</sup> Liste de sites qui effectués du Coaking en 2008 : <http://moz.com/blog/white-hat-cloaking-it-exists-its-permitted-its-useful>

<sup>71</sup> Différents retours d'expériences sur les Hashbangs :

- <http://danwebb.net/2011/5/28/it-is-about-the-hashbangs> ;
- <http://fr.slideshare.net/RobOusbey/new-technologies-11724391> ;
- <http://www.adequatelygood.com/Thoughts-on-the-Hashbang.html> ;
- <https://josephscott.org/archives/2012/05/twitter-backing-away-from-hashbang-urls/>

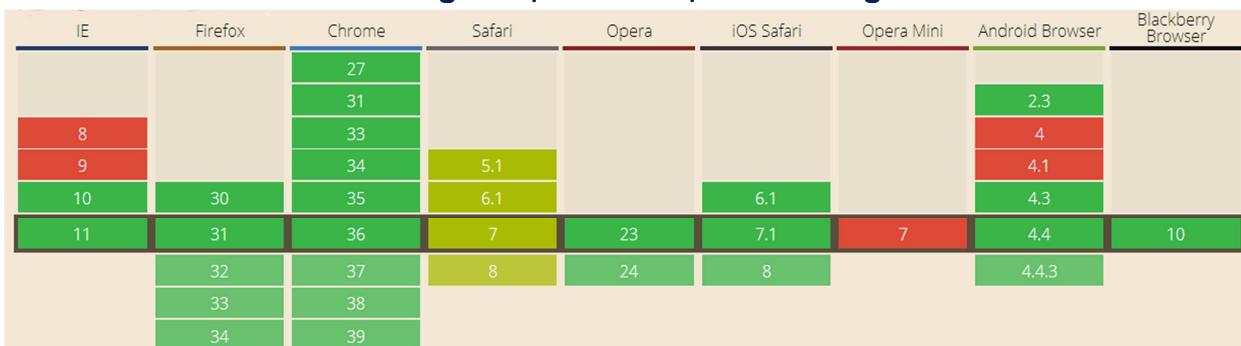


Source: <http://fr.slideshare.net/RobOusbey/new-technologies-11724391/22>

### La fonctionnalité HTML5 « pushState »

« pushState » est une méthode de l'API « history » qui change l'historique de navigation grâce au JavaScript. L'intérêt est de modifier l'url lorsque le contenu de la page change dynamiquement.

**Prise en charge de pushstate par les navigateurs**



Source : <http://caniuse.com/>

On remarque que la majorité des navigateurs prennent en charge cette méthode et un polyfill existe<sup>72</sup> afin de prendre en compte les navigateurs compatibles avec HTML4 (IE6 compris). Les avantages de cette solution sont :

- La possibilité d'utiliser l'historique du navigateur afin de revenir en arrière (ce qui est impossible si le contenu est chargé dynamiquement en JavaScript et que l'url reste la même),
- La possibilité d'utiliser la nouvelle url pour générer un contenu côté serveur dans le cas où le JavaScript serait désactivé.

<sup>72</sup> Polyfill History : <https://github.com/browserstate/history.js>

Il suffirait que le lien soit présent à un endroit afin qu'un robot d'indexation puisse y accéder. Une technique consiste à générer un fichier sitemap.xml permettant de référencer l'ensemble des liens d'un site. Ce fichier sert pour l'indexation des différentes pages dans le cas où le site aurait des difficultés à être référencé. Cependant, il existe des inconvénients<sup>73</sup> :

- si le robot rencontre un quelconque problème lors de l'indexation, le sitemap ne sera pas pris en compte ;
- Google n'utilise pas les liens figurant dans ce fichier pour le « PageRank ». Le pagerank est un indicateur parmi d'autres donnant de l'importance à un lien afin qu'il remonte plus haut dans les résultats de recherche.

## PhantomJS

PhantomJS est la technique la plus répandue afin de générer le contenu Front du côté du Back. C'est un navigateur headless<sup>74</sup> qui permet de parcourir les pages Web directement sur le serveur. Il est utilisé principalement pour lancer des tests fonctionnels mais il peut tout aussi bien servir à retourner une page statique pour les robots. Bien que le contenu soit identique, la méthode diffère entre les utilisateurs et les robots. On se retrouve donc dans la catégorie « Dark Grey » de notre tableau de cloaking. Il y a peu de change que l'on obtienne des sanctions de Google actuellement étant donné que cette technique a fait ses preuves, mais il n'est pas impossible que dans le futur, Google se mettent à l'interdire. Une des raisons que l'on pourrait trouver est que l'utilisateur n'a pas la même expérience que le robot. L'utilisateur pourra avoir des temps de chargement beaucoup plus long. On peut donc apercevoir du cloaking au niveau de l'expérience utilisateur.

## JsDom

JsDom est une interprétation JavaScript du DOM côté serveur. Il permet comme PhantomJs de rendre un contenu statique et par le biais de node.js, de rediriger le robot afin de référencer ce même contenu. L'avantage par rapport à PhantomJS est qu'il n'y a pas de « binaire » à installer et pas de port à ouvrir. Encore une fois, cette technique ainsi que les suivantes fonctionnent mais peuvent comporter des risques.

## Prerender

Prerender.io est un outil open source. Il contient un serveur node.js et utilise PhantomJS afin de rendre facilement des pages statiques. Il peut être utilisé dans une application node.js mais également avec d'autres langages comme Java, PHP, Groovy, Ruby... Son avantage est qu'il se charge de gérer la génération de la page pour les moteurs de recherche. Il peut néanmoins être payant si le nombre de pages générées est supérieur à 250. Il possède comme inconvénient de générer la page à la volé suite à une redirection, ce qui peut provoquer une augmentation du temps de chargement lors de l'indexation de la page. Il y a également un temps d'attente présent afin que les requêtes AJAX puissent s'exécuter. Il faut donc penser à générer la page en cache pour que les robots puissent l'utiliser sans avoir ce temps de chargement contraignant.

Il existe également d'autres services plus ou moins payants comme Brombone (s'appuyant sur le fichier sitemap.xml) ou encore seo4ajax et rankjs.com (utilisant un **crawler**).

<sup>73</sup> Plus d'information sur les problèmes liés au fichier sitemap.xml :

<http://webmasters.stackexchange.com/questions/4803/the-sitemap-paradox>

<sup>74</sup> **Headless browser** : navigateur sans tête en français. Il permet d'interagir avec le DOM sans interface graphique

### 3.1.3. Les gestionnaires de paquets

#### npm

En regardant tous les services qui existent en node.js, répondant à des besoins très variés, on pourrait croire que la plate-forme est imposante. Ce serait entièrement faux, node.js seul, ne fait pas grand-chose. Sa force réside surtout dans sa modularité grâce au gestionnaire de paquets du nom de npm (pour Node.js Package Manager). Npm est l'utilitaire qui permet d'installer des applications node.js mais également des modules de tous types qui enrichissent un projet avec des fonctionnalités déjà développées.

**Évolution du nombre de paquets et du nombre de téléchargements**



Source : <http://blog.npmjs.org/>

Comme on peut le voir dans le graphique ci-dessus, npm subit une évolution fulgurante tant au niveau du nombre de paquets qu'au niveau de leurs téléchargements. Cela est dû à la technologie node.js mais également au fait qu'il est très simple d'installer ou de partager un paquet. Il se base sur un fichier de configuration du nom de package.json. Ce dernier permet entre autre d'identifier le paquet à installer, de gérer ses dépendances et va même jusqu'à lancer des commandes et scripts. Il utilise des dépôts SVN ou GIT publics mais il est tout à fait possible de référencer des dépôts privés.

Npm est actuellement en train de changer son architecture et de déplacer petit à petit les dépôts publics vers npm Entreprise (npmE<sup>75</sup>). L'idée d'npmE est de pouvoir installer son propre gestionnaire de paquets se basant sur npm. npmE permettra de publier des dépôts privés et de créer des systèmes sécurisés avec des restrictions sur les paquets disponibles.

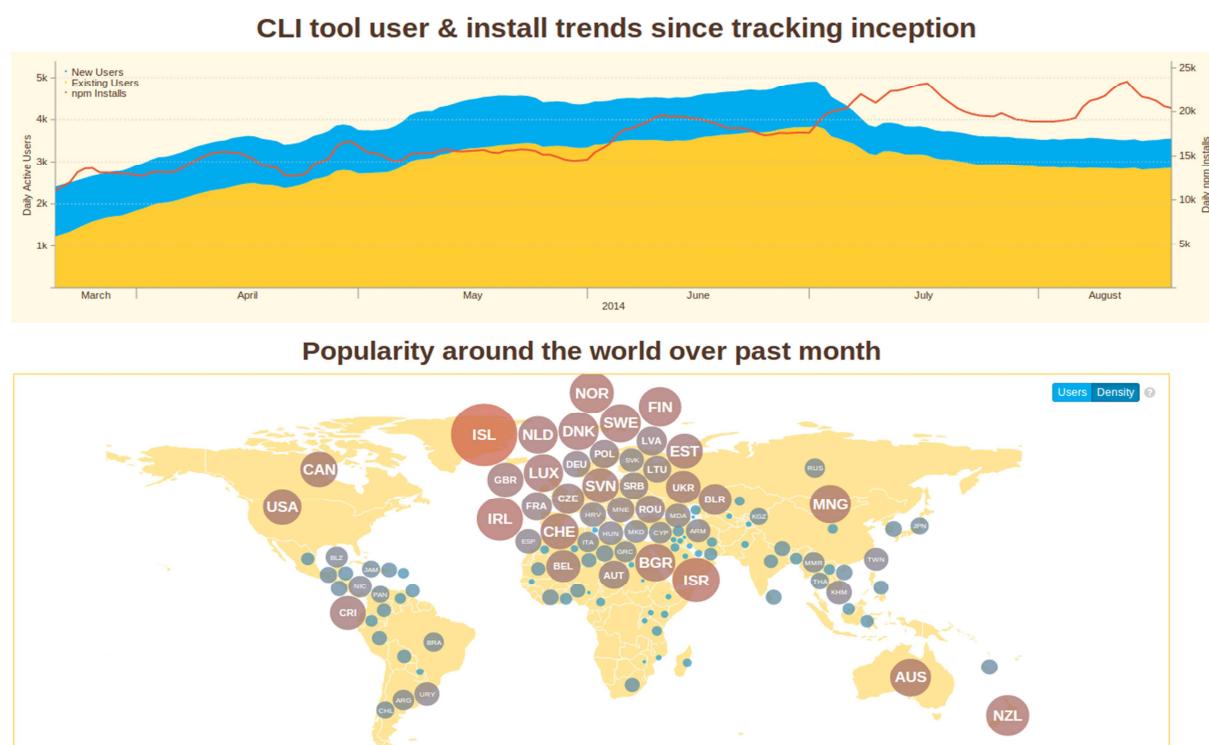
<sup>75</sup> Feuille de route de npmE : <http://blog.npmjs.org/post/93509138505/npm-enterprise-roadmap>

## Bower

Bower est un gestionnaire de paquets pour les librairies Front. Il a été développé par Twitter et utilise npm pour l'installation. Bower ne se limite pas au JavaScript, il contient plus de 18 000 paquets qui comprennent également des librairies CSS. L'utilité de Bower est de prendre en charge la gestion des dépendances de l'ensemble des éléments externes utilisés sur un projet. Plus besoins de vérifier qu'il n'y ait aucun conflit lors de l'ajout d'une librairie externe ou de faire les mises à jours des versions manuellement, Bower s'en charge pour nous.

Contrairement à npm, Bower ne va pas charger directement l'arbre de dépendance de chaque librairie<sup>76</sup>. Il est optimisé pour le Web et fait en sorte de ne garder qu'une version compatible avec tout l'environnement. Si un conflit survient, Bower lèvera une erreur. Par contre, de la même manière qu'npm, il utilise un simple fichier de configuration (bower.json), les dépôts Git, SVN mais également des simples URLs ou des chemins locaux. La publication de composant "package" est encore plus simple que pour npm, une seule commande suffit.

Bower et npm sont complémentaire. Ils peuvent cohabiter ensemble et peuvent posséder des paquets ainsi que des dépendances en communs. Le choix de l'outil se fera donc en fonction de la structure du projet et de l'utilisation que l'on fera du paquet à installer. Le site officiel de Bower contient des statistiques montrant les différentes évolutions du gestionnaire de paquets.



Source : <http://bower.io/>

<sup>76</sup> Plus d'information sur Bower : <http://blog.soat.fr/2014/08/bower-un-gestionnaire-de-dependances-pour-le-web/>

### 3.1.4. Les outils d'automatisation

Les projets Front possédant un nombre d'outils grandissant, ont complexifié le développement et les actions effectuées. Ils ont eu pour répercussions de devoir lancer manuellement différentes tâches (minifications, compilations...). Certains outils comme Compass (Framework CSS), peuvent faciliter le développement en proposant par défaut des commandes utiles. D'autres comme par exemple LESS (compilateur CSS), demandait à l'origine la création de scripts Shell<sup>77</sup> ou MakeFile<sup>78</sup> afin d'automatiser un minimum la compilation des fichiers. Le problème était que chaque outil avait son propre « petit » système d'automatisation ne résolvant qu'à moitié le problème de ces tâches fastidieuses. De plus, l'utilisation de différents Frameworks imposait à chaque initialisation de projet de devoir repenser et recréer une architecture afin que l'ensemble des éléments puissent communiquer correctement entre eux. Cela demandait parfois beaucoup de temps de réflexion. Des outils d'automatisations ont ainsi vu le jour afin de régler ces problèmes.

#### Grunt



Source : <http://luismatute.me/>

Grunt est un lanceur de tâches (task runner) créé par Ben Alman (contributeur jQuery). Il fait partie des nombreux lanceurs de tâches disponibles en node.js<sup>79</sup>. Ces outils permettent d'automatiser certaines actions (minifications, compilations, tests unitaires, fonctionnels, ...). Grunt, quant à lui, est le lanceur de tâches possédant la plus grande communauté. Il s'appuie sur npm afin de référencer ses plugins (plus de 3400 actuellement). Son utilisation peut même aller jusqu'à relancer un serveur node.js à chaque modification ou encore insérer automatiquement dans les templates HTML, les librairies récupérées par Bower. Grunt se sert d'un fichier JavaScript afin de référencer les différentes tâches utilisées. Il utilise l'approche “convention over code” en s'appuyant sur des tableaux JSON. Ce choix permet à des personnes qui n'ont pas forcément de profil technique (des designers par exemples), de facilement utiliser l'outil sans véritablement faire du JavaScript.

<sup>77</sup> **Shell** : Interface utilisateur d'un système d'exploitation

<sup>78</sup> **MakeFile** : Fichier de configuration de l'exécutable make

<sup>79</sup> Liste de lanceur de tâche en node.js : <https://gist.github.com/callumacrae/9231589>

Grunt possède tout de même quelques défauts. Il a été conçu à la base pour des petits projets. Plus on référence de tâches, plus Grunt sera lent car à chaque exécution d'une tâche, Grunt chargera de manière synchrone<sup>80</sup> l'ensemble des plugins utilisés ainsi que leurs dépendances. Il y a également la contrainte du nombre de fichier car à chaque altération demandée, Grunt va effectuer des actions de lectures/écritures. Il existe cependant des astuces afin de réduire ces lenteurs :

- charger les plugins uniquement pour les tâches qui les utilisent ;
- diviser les tâches Grunt dans différents fichiers ;
- exécuter des tâches en parallèle avec le plugin : grunt-concurrent ;
- se préoccuper uniquement des fichiers qui ont changés et pas d'un ensemble de fichier.

## Gulp



Source : <http://gulpjs.com/>

Gulp est également un lanceur de tâche mais avec une philosophie différente. Avec Gulp, on ne se limite plus à configurer des plugins mais on code réellement du JavaScript (code over configuration). Il utilise une approche différente lui permettant d'être par défaut beaucoup plus rapide que Grunt, les « streams ». Les streams permettent d'englober les fichiers utilisés par une tâche et de les manipuler dans des tubes à l'aide de l'API fournie par Gulp. L'avantage est que l'on ne va pas rechercher les mêmes ressources à chaque action contrairement à Grunt.

### Fonctionnement de Gulp



Source : <http://mo.phlow.de/>

<sup>80</sup> **Synchrone** : bloquant l'exécution du script

Gulp fonctionne comme un flux. Le rassemblement de tâches s'effectue en chaînant les différents résultats de ces dernières. Gulp a suscité beaucoup d'intérêt lors de son apparition. Il possède néanmoins une communauté moins grande que Grunt et contient beaucoup moins de plugins (737 actuellement). L'autre point négatif que l'on peut lui trouver est qu'il est moins accessible aux designers car il impose le fait d'avoir des notions de JavaScript.

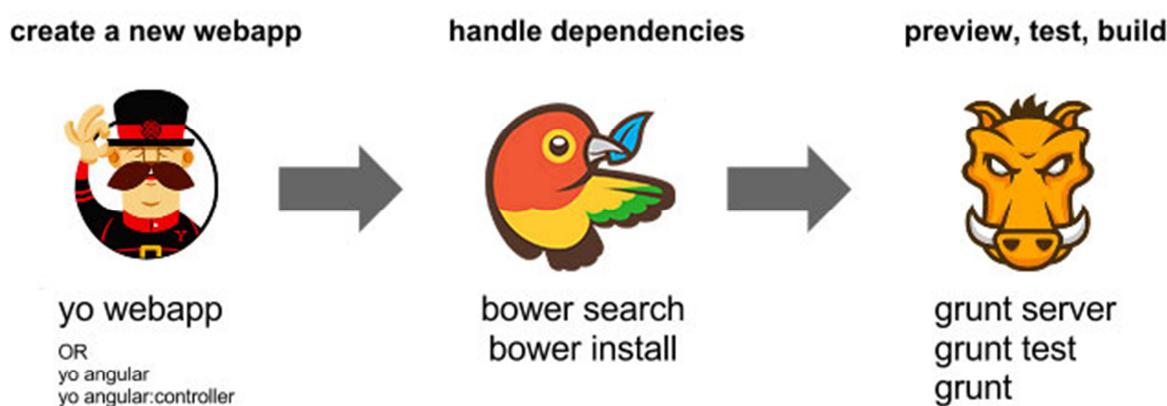
## Yeoman



## YEOMAN

Source : <https://www.openshift.com>

Yeoman est un outil de « scaffolding ». Il permet de construire une application et nous enlève la contrainte de devoir partir de zéro, architecturer un projet et initialiser les dépendances Bower ainsi que les tâches Grunt. Il contient par défaut un exécutable, « yo », qui interagit avec Bower et Grunt.



Source : <http://javamind-fr.blogspot.fr/>

Yeoman est composé de générateurs. Il contient actuellement une vingtaine de générateurs maintenus officiellement par Yeoman et plus de 900 autres provenant de la communauté. Il est très simple de créer son propre générateur, il existe même un générateur prévu pour en créer d'autres.

Le problème que l'on pouvait rencontrer lors de l'utilisation de plusieurs outils côté Front était de trouver le moyen de les rassembler. Yeoman résout cette problématique et nous permet d'avoir une architecture par défaut d'un projet qui fonctionne. Il peut par exemple construire une plate-forme MEAN (MongoDB<sup>81</sup>, Express, Angular, node.js). Mais il peut aussi avoir d'autres utilités :

- créer un diaporama avec Reveal.js
- initialiser un projet Scala, Wordpress, Jekyll ...
- initialiser un wiki sur Github
- et bien d'autres choses...

Yeoman peut remplacer Grunt par Gulp. La limitation que l'on pourrait trouver et qu'il est fortement déconseillé de générer plusieurs architectures sur un même projet car celles-ci ont des risques de rentrer en conflits à partir du moment où elles utilisent des technologies communes. Il faut donc trouver le générateur répondant le mieux aux besoins du projet ou construire son propre générateur afin de gagner du temps pour les projets futurs.

---

<sup>81</sup> **MongoDB** : Base de données NoSQL la plus populaire

## 3.2. Les Frameworks côté Serveur

### 3.2.1. Express.js

Express est le Framework node.js le plus populaire. Il est léger et flexible, ce qui lui accorde une certaine polyvalence lui permettant de répondre à des projets de toutes tailles. Il possède un moteur de templates par défaut du nom de « jade » mais il est très simple de le remplacer. Il peut servir à répondre aux problématiques Backend des SPA (Simple Page Application) mais également aux projets « hybrides » pouvant contenir plusieurs pages et même d'autres technologies Back. On peut facilement lui rajouter des fonctionnalités comme par exemple PassportJS proposant un service d'authentification.

Express en est à sa version 4. Il possède une grande communauté lui accordant une certaine notoriété face aux autres. Au niveau des références, on peut citer MySpace, Storify, Gost (plate-forme de Blogging) et Countly (plate-forme d'analyse d'application mobile). La force d'Express réside surtout dans le fait qu'il est utilisé comme socle pour de nombreux autres Frameworks node.js.

#### Express.io

C'est un Framework MVC rassemblant Express.js et Socket.io. Socket.io permet d'utiliser les « Web Sockets » côté serveur. Les Web Sockets font partie des spécifications d'HTML5. Ils remplacent des pratiques liées à AJAX et offrent une nouvelle manière de communiquer de manière asynchrone. L'intérêt est donc de régler les problématiques de temps réel. Contrairement à l'AJAX, les Web Sockets ne créent pas de connexion à chaque envoi et n'attendent pas une réponse automatiquement. Une fois qu'une connexion est ouverte, elle est utilisée afin de communiquer de manière bidirectionnelle. Il y a donc un grand gain de performance. Socket.io peut facilement se coupler avec Express.js. L'avantage d'Express.io est donc uniquement de proposer les deux composants par défaut.

#### Hapi

Hapi ajoute des fonctionnalités intéressantes à Express comme l'authentification, le cache, les logs<sup>82</sup>, la validation de formulaire ou encore la gestion de plugin. Il possède lui aussi des belles références comme Yahoo, BeatsMusic, Walmart, Mozilla, Paypal et npm. Il permet de réaliser des applications riches mais également des services.

#### Krakenjs

Krakenjs est la suite d'outil de Paypal se basant sur Express et pouvant s'utiliser indépendamment :

- Lusca : offre des fonctionnalités pour sécuriser une application node.js ;
- Kappa : est un proxy du gestionnaire de dépendance de node.js permettant de créer des dépôts privés afin de ne pas dépendre de ceux qui sont publics. C'est en fait un plugin Hapi ;
- Makara : est un module pour Dust.js fait avec Express permettant de gérer l'internationalisation des templates. Dust.js étant un moteur de template node.js ;
- Adaro : est encore un module pour Dust permettant de manipuler les vues.

---

<sup>82</sup> **Logs** : historique d'évènement sauvegardé

## Sails.js

Sails est un Framework regroupant Express et Socket.io, il est donc principalement orienté sur le temps réel. Il utilise le pattern MVC et embarque des commandes CLI<sup>83</sup> facilitant la création d'API REST performantes. Sails peut également utiliser les Web Sockets de façon transparente.

## Locomotive

C'est un des Framework les plus puissants contenant beaucoup de « convention over configuration ». Il repose sur le pattern MVC et sur des routes RESTful<sup>84</sup>. Il peut permettre de se connecter à n'importe quelle base de données et peut se coupler à n'importe quel moteur de template.

### 3.2.2. Meteor.js



# Meteor.js



Source : <https://www.linkedin.com>

Meteor est un Framework un peu particulier qui aurait trouvé sa place dans le comparatif des Framework Front. Il fait partie des Frameworks « full-stack » comme Derby ou Mojito de Yahoo qui s'utilisent côté Front et Back. Il a été créé en décembre 2011 et a levé 11,2 millions d'euros peu de temps après (juillet 2012).

L'avantage de ce type de Framework est qu'il répond plus globalement aux problématiques des applications web. Il permet de communiquer de manière bidirectionnelle en proposant une gestion avancée des Web Sockets. Grâce à l'API de Meteor, la persistance est gérée sur le serveur et les données sont également conservées sur le Front. Par défaut le protocole HTTP est « stateless » : il ne conserve aucun état, envoie des requêtes et reçoit des réponses. Meteor met en place du cache et des sessions créant un état « statefull ». L'avantage est de ne plus dépendre de la disponibilité du serveur et de pouvoir facilement répondre aux contraintes hors lignes. En effet, une application web se doit de pouvoir continuer de fonctionner dans un navigateur même s'il n'y a plus de connexion Internet.

<sup>83</sup> CLI : Command-Line Interface. Interface de ligne de commande en Français

<sup>84</sup> RESTful : qui respecte les principes REST

Meteor n'a pas encore atteint la version 1. Il possède tout de même un écosystème qui grandit vite et est déjà prêt pour la production<sup>85</sup>. Meteor a son propre gestionnaire de paquet du nom de Meteorite ainsi qu'un moteur de recherche : Atmosphere<sup>86</sup>. Meteor possède déjà plus de 1700 paquets référencés. Certains d'entre-eux permettent de facilement coupler Meteor à d'autres Framework Front comme Angular, Ember, Backbone, React mais également Famo.us dont nous n'avons pas encore parlé.

### Famono

Famo.us est un nouveau Framework qui concurrence les applications mobiles natives. Il utilise une nouvelle approche<sup>87</sup> et raisonne en termes de "surface". L'idée et d'animer les pages Web en utilisant des transformations sans passer par le CSS. Il peut facilement se coupler à Meteor grâce au paquet Famono<sup>88</sup>. Cette association fait parler d'elle en ce moment afin de répondre aux applications mobiles en temps réel. On peut par exemple citer Facebook Paper<sup>89</sup> qui a été réalisé avec Famo.us.

---

<sup>85</sup> Conférence de Ritik Malhotra :

<https://www.youtube.com/watch?v=gfFGjmiKfnA&list=UU3fBiJrFFMhKlsWM46AsAYw>

<sup>86</sup> Atmosphere : <http://atmospherejs.com/>

<sup>87</sup> Explication de Famo.us : <http://www.forbes.com/sites/anthonykosner/2014/05/27/famo-us-part-i-new-concepts-will-increase-the-flow-of-highly-dynamic-web-3-0-apps/>

<sup>88</sup> Famono : <https://github.com/raix/Meteor-famono>

<sup>89</sup> Une démonstration de l'application est disponible ici : <http://vimeo.com/86664858>

## 3.3. La recherche de la qualité

### 3.3.1. les tests

Les tests deviennent encore plus importants au fur et à mesure qu'une application grandit. Ils permettent d'effectuer des développements robustes mais également de mieux les architecturer, effectuer des séparations et les rendre plus facilement maintenable et évolutive. Comme dans tout autre langage, il est possible de faire des tests en JavaScript afin de produire du code de qualité.

#### Mocha.js



Source : <http://dailyjs.com/>

Parmi les outils de tests unitaires, Mocha est la librairie de référence. La majorité des générateurs de Yeoman installent par défaut Mocha. L'avantage de cet outil est qu'il est modulaire et flexible. Il se couple très facilement à des librairies d'assertions (comme Should.js<sup>90</sup> ou Chai.js<sup>91</sup>) et de mocks (comme Sinon.js<sup>92</sup>). Avec Mocha, il est possible de choisir sa méthode de développement. Il s'intègre facilement à des outils d'intégration continue et possède des plugins permettant d'enrichir ses fonctionnalités.

#### Jasmine.js



Source : <http://jasmine.github.io/>

Jasmine est un Framework de tests fonctionnels. Contrairement à Mocha qui est flexible mais nécessite des librairies additionnelles, Jasmine est entièrement autonome. Il possède son propre système d'assertions et de mocks. Bien qu'il ne soit pas prévu pour cela, il possède également une version alternative (jasmine-node) afin de faire des tests unitaires pour node.js. Jasmine est surtout populaire pour tester le JavaScript du Front exécuté par un navigateur. Contrairement à QUnit (Framework de tests de jQuery) qui s'appuie sur le TDD<sup>93</sup>, Jasmine utilise le BDD (Behavior Driven Development comme approche). Cette méthode permet de mieux représenter les spécifications des clients plutôt que le résultat attendu par le JavaScript.

<sup>90</sup> Should.js : <https://github.com/visionmedia/should.js/>

<sup>91</sup> Chai.js : <http://chaijs.com/>

<sup>92</sup> Sinon.js : <http://sinonjs.org/>

<sup>93</sup> TDD : Test Driven Development : méthode de développement dirigée par les tests

## Cucumber.js



Source : <http://npmawesome.com/>

Cucumber.js est également un Framework de tests fonctionnels, basé sur Cucumber de Ruby. La différence avec Jasmine est qu'il utilise par défaut un DSL (Domain Specific Language). Un DSL permet de ne plus coder en JavaScript mais de réellement écrire des phrases reflétant les spécifications du projet. Elles sont ensuite converties afin d'être interprétées en JavaScript. En soit, CoffeeScript est une sorte de DSL. La différence avec Cucumber est que ce dernier est fortement orienté sur les tests et les assertions. L'intérêt est de permettre à des personnes qui n'ont pas de profils techniques de pouvoir écrire les scénarios de tests.

### Exemple de tests avec Cucumber.js

```
Feature: Example feature
As a user of cucumber.js
I want to have documentation on cucumber
So that I can concentrate on building awesome applications

Scenario: Reading documentation
Given I am on the Cucumber.js GitHub repository
When I go to the README file
Then I should see "Usage" as the page title
```

Source : <https://github.com/cucumber/cucumber-js>

## CasperJS

Bien que Jasmine s'exécute sur un navigateur, il n'interagit pas directement avec le DOM. CasperJS, quant à lui, fait des tests fonctionnels comme peut le faire Selenium en Java. Il peut naviguer dans le DOM, utiliser des formulaires, cliquer sur des liens et va même jusqu'à faire des imprime-écrans « screenshots ». Ces derniers peuvent permettre de repérer des différences sur l'interface et remonter des problèmes graphiques.

## Istanbul

Istanbul est un outil de couverture de code pour le JavaScript. Il permet d'identifier le taux de couverture et d'ainsi, repérer les parties du code qui ne sont pas testées. Il peut se coupler à d'autres outils comme Coveralls permettant de visualiser la couverture via le Web. Coveralls dispose d'une apparence plus soignée et fait partie des nombreux outils qui peuvent se coupler à GitHub.

## Karma



Source : <http://karma-runner.github.io/>

De la même manière que Mocha, karma est un outil très souvent présent dans les générateurs Yeoman. C'est un « test runner », il permet de lancer plus facilement les tests en proposant une abstraction sur ces derniers. Il peut se coupler à un « task runner » comme Grunt et regrouper le lancement des tests unitaires, fonctionnels et la couverture de code en une seule commande. Il permet aussi de facilement remplacer les Frameworks et les navigateurs utilisés pour les tests.

### 3.3.2. Les outils de monitoring

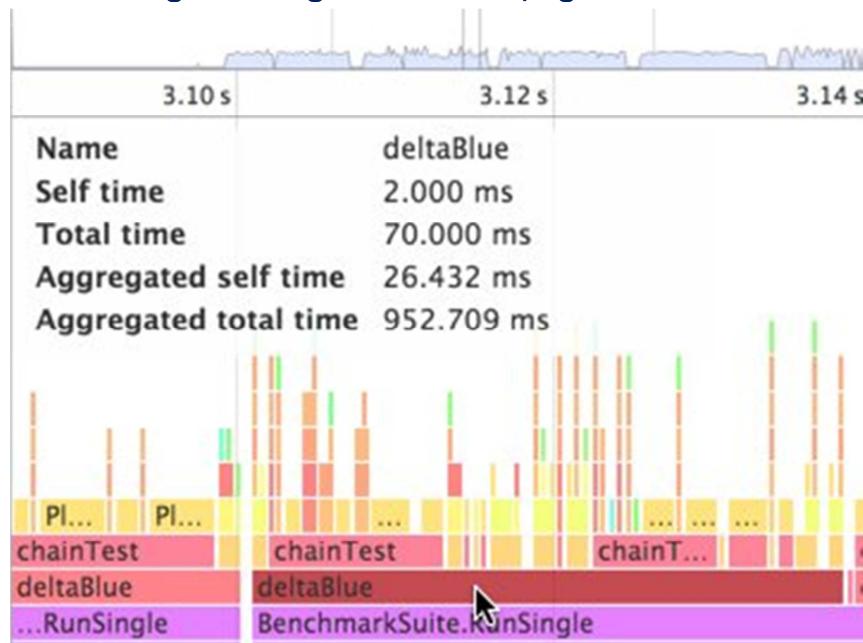
Le JavaScript Front étant exécuté sur les navigateurs, il devient difficile de repérer les erreurs et les zones non optimisées. Différents outils existent afin de répondre à ce besoin.

#### Les outils de débogage

Les outils de débogage des navigateurs (IE>10, Firebug pour Firefox et Chrome DevTools) sont des bons exemples d'outils de débogage. Ils permettent de repérer les erreurs JavaScript qui sont survenues dans une page Web mais contiennent également d'autres fonctionnalités très intéressantes comme le fait de :

- remonter des « warning » lors de l'utilisation de fonctions JavaScript dépréciées ;
- gérer des points d'arrêt (breakpoint) afin de visualiser l'état de l'exécution du JavaScript à un moment donné ;
- possibilité d'écrire du JavaScript depuis la console du navigateur ;
- visualiser des détails sur le chargement de la page ou l'utilisation du GPU ;
- gérer et manipuler des cookies, sessions et bases de données du côté des clients.

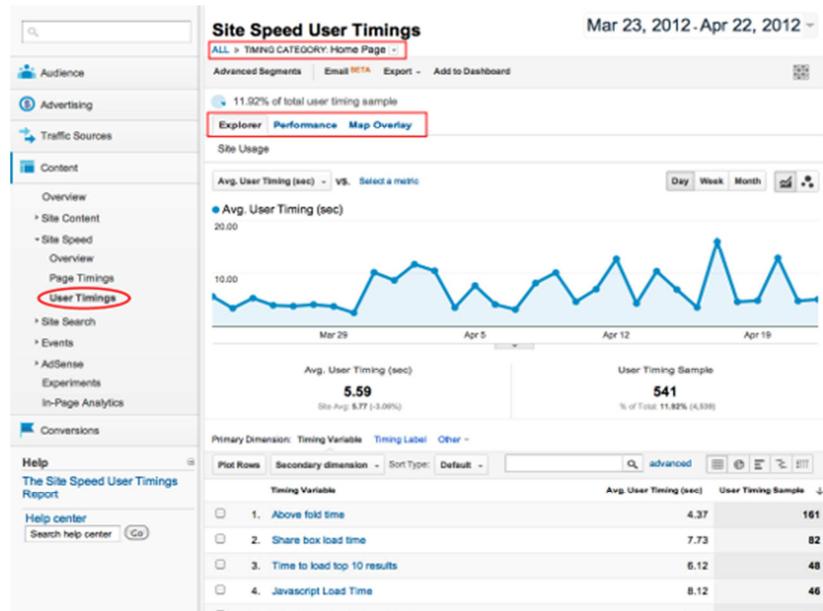
#### *Exemple de monitoring du chargement d'une page à l'aide de Chrome DevTools*



Source : <https://developer.chrome.com/>

#### Google Analytics

Google Analytics est une application de Google permettant de suivre les actions des visiteurs, de savoir d'où ils viennent et ce qu'ils font. Il peut également aller plus loin en proposant une API récupérant des informations sur les temps d'exécution. Nous pouvons ainsi vérifier que les performances sont toujours bonnes selon la géolocalisation ou l'appareil utilisé (portable, tablette, ordinateur...) du visiteur.

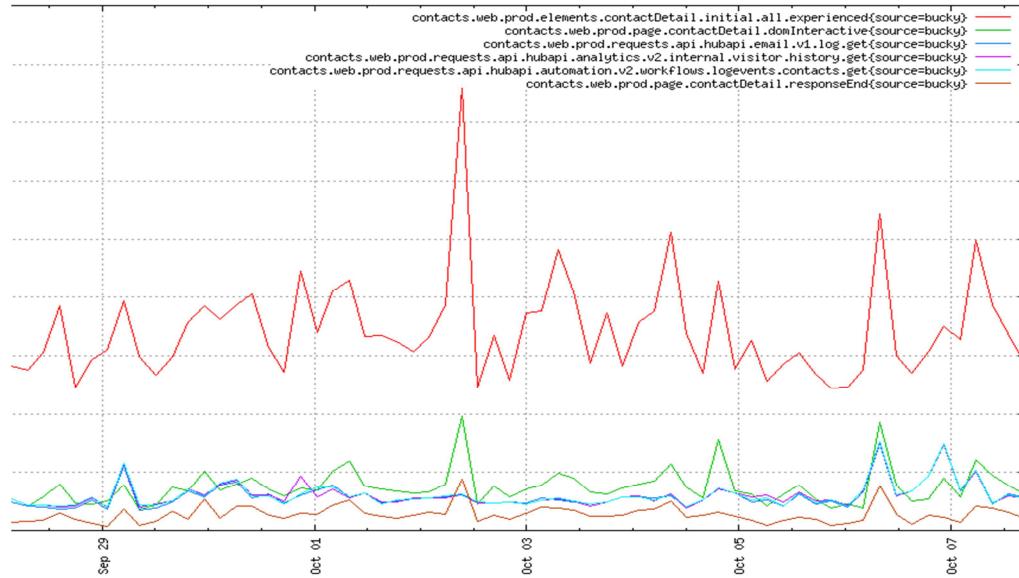


Source : <http://dev.hubspot.com/>

## Bucky

Bucky est le seul service gratuit que j'ai trouvé permettant d'analyser une application node.js. Il se couple avec StatsD<sup>94</sup> ou OpenTSDB<sup>95</sup> qui permettent de stocker et lire les données envoyées à l'aide du paquet Bucky.

### Exemple de graphique Bucky



Source : <http://dev.hubspot.com/>

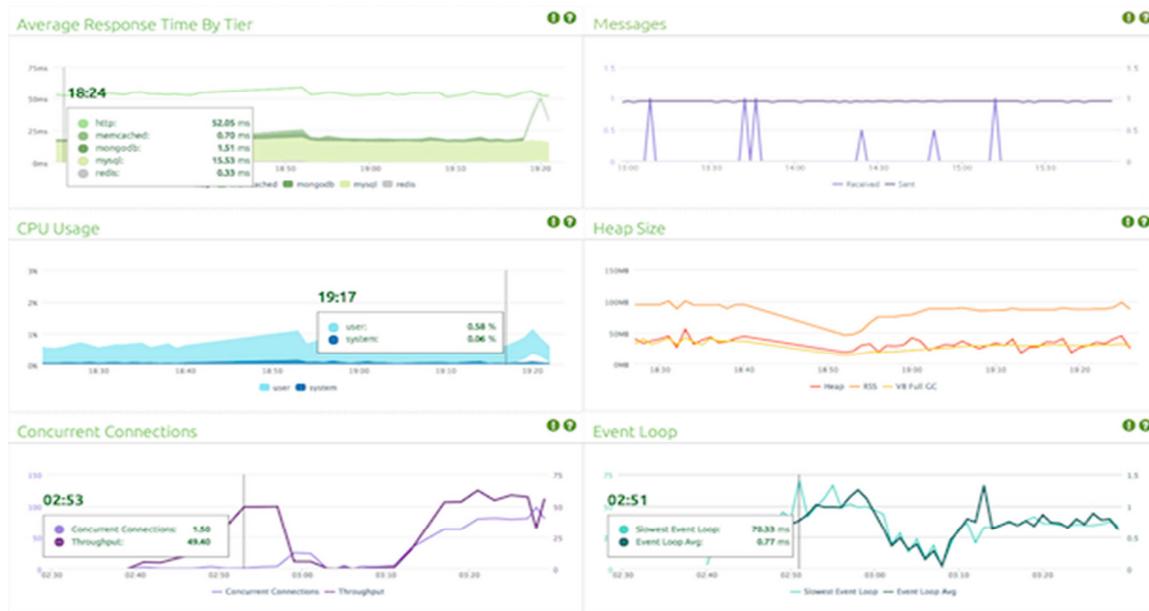
<sup>94</sup> **StatsD** : simple processus qui agrège des statistiques

<sup>95</sup> **OpenTSDB** : Time Series Daemon (TSD) représentant différents processus permettant également d'agrégner des données

## StrongLoop Agent

StrongLoop Agent ou StrongOps est l'outil développé par StrongLoop afin de surveiller une application JavaScript. Il se positionne comme la seule solution à être une ALM (Application Lifecycle Management) pour node.js. Une ALM permet de suivre le cycle de vie d'une application (du développement à l'utilisation en production) afin de faciliter la maintenance et garantir un niveau de qualité.

*Exemple de graphiques remontés par StrongOps*



Source : <http://strongloop.com/>

L'outil est sorti en avril dernier et contient de nombreuses fonctionnalités comme :

- surveillance des clusters (processus node.js) ;
- surveillance de l'utilisation du CPU sur le serveur et du GPU sur les navigateurs clients ;
- signalement des erreurs JavaScript ;
- facilitation des déploiements ;
- et bien d'autres choses<sup>96</sup>...

## New Relic

New Relic est également une ALM mais qui ne se limite pas à node.js. Il peut aussi s'utiliser avec PHP, Java, Ruby, .net et Python. Il possède six applications différentes en SaaS (Software as a service) :

- New Relic APM (Application Performance Management) : analyse les performances de l'application
- New Relic Mobile : analyse les applications mobiles sur IOS et Android
- New Relic Insights : enregistre les données des applications en temps-réel
- New Relic Servers : surveille les serveurs
- New Relic Browser : analyse l'expérience utilisateur sur les navigateurs
- New Relic Platform : offre une plate-forme personnalisée afin de montrer une application entière (client, serveur, temps-réel ...).

<sup>96</sup> Plus d'information sur StrongOps : <http://strongloop.com/node-js/controller/>

New Relic répond donc à beaucoup de besoins. Il est l'un des plus complets du marché et évite de reposer sur plusieurs outils et technologies afin d'analyser différentes parties d'une application.

## Fasterize

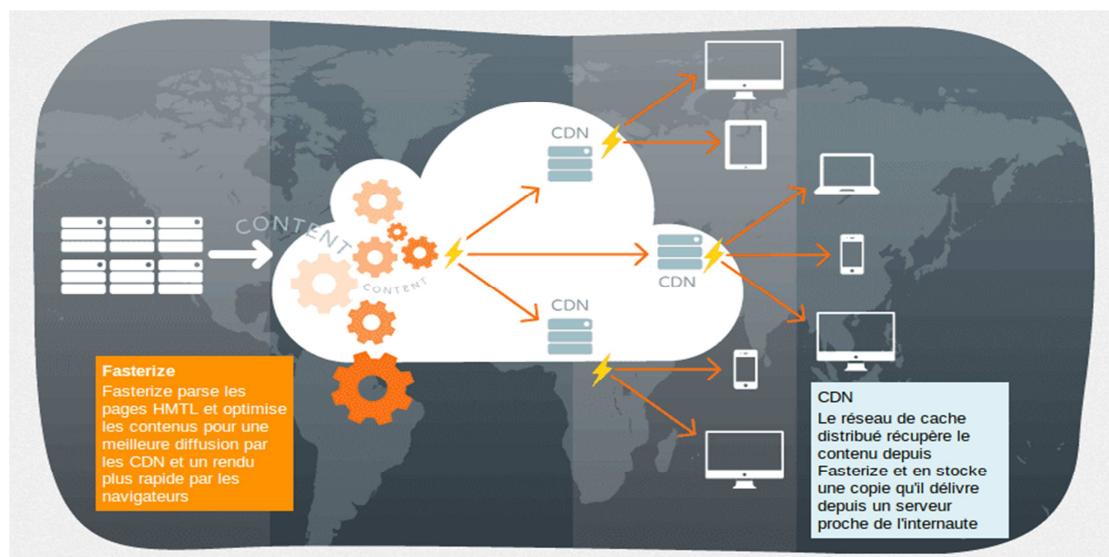
Fasterize est une start-up française se positionnant sur l'accélération des sites Web (FEO pour Front End Optimization en anglais). Il réagit de la même manière qu'un proxy afin d'intercepter les requêtes effectuées par les visiteurs, optimiser les pages selon un ensemble de bonnes pratiques et les rendre en temps réel.

Fasterize peut être utilisé afin de répondre aux différentes problématiques que nous avons déjà évoquées (minifications, concaténations, compression...). L'avantage est qu'il gère toute ces problématiques de son côté et ne modifie ni le code ni l'architecture se trouvant sur le serveur. Il s'appuie également beaucoup sur le chargement asynchrone (lazy-loading) et sur l'utilisation de CDN afin d'optimiser au maximum le temps de chargement.

Le lazy-loading permet de charger des éléments (JavaScript, images, ou du contenu supplémentaire) de manière différée après un premier chargement de la page afin que l'utilisateur puisse avoir accès aux éléments importants sans attendre que l'intégralité de la page web soit chargée. Il existe beaucoup de librairies<sup>97</sup> permettant de remplir ce rôle mais il peut être difficile de les coupler à l'architecture d'un projet.

Les CDN (Content Delivery Network) sont utilisés en JavaScript afin de rendre un contenu depuis une adresse unique, quel que soit le site qui l'utilise. De ce fait, statistiquement, la ressource a de plus grandes chances d'être dans le cache du navigateur de l'internaute. Le but initial est de s'appuyer dessus pour le chargement de librairies externes utilisées par d'autres sites web. Lorsque l'utilisateur charge une fois la librairie provenant du CDN, il la met en cache pour tous les sites qui l'utilisent.

### Fonctionnement de Fasterize et des CDN



Source : <http://www.fasterize.com/>

<sup>97</sup> Liste de librairies de chargement asynchrone : [http://mathrobin.github.io/js\\_loaders/#/](http://mathrobin.github.io/js_loaders/#/)

J'ai inclue Fasterize dans ce comparatif d'outils de monitoring parce qu'il ne s'arrête pas à l'optimisation des pages. Il offre également une interface analytique permettant de visualiser les performances des différentes pages.

Fasterize n'est pas le seul à se positionner dans le secteur des FEO. On pourrait également citer CloudFlare<sup>98</sup>, Torbit<sup>99</sup> ou encore Blaze<sup>100</sup>.

---

<sup>98</sup> CloudFlare : <https://www.cloudflare.com/>

<sup>99</sup> Torbit : <http://torbit.com/>

<sup>100</sup> Blaze : <http://www.akamai.com/blaze>

### 3.3.3. Le respect des bonnes pratiques

Nous venons de voir une solution permettant d'optimiser le chargement de pages selon un ensemble de bonnes pratiques mais il n'est pas interdit de s'appuyer également sur des standards durant le développement. La qualité d'une application se trouve aussi dans l'uniformisation du développement et le respect de certaines règles. Il existe des ressources permettant de regrouper des conventions de codages et des bonnes pratiques. Par exemple, il y a celles de Yahoo<sup>101</sup>, Google<sup>102</sup>, Mozilla<sup>103</sup>, mais également des contenus contribuables par tous comme Web Plateform<sup>104</sup>, Idiomatic JavaScript<sup>105</sup>, Frontend-dev Bookmark<sup>106</sup> et OpQuast que nous verrons un peu plus loin.

#### Le mode strict

Le mode strict en JavaScript est une fonctionnalité apparue avec ECMAScript 5 dépréciant certaines fonctionnalités et levant des erreurs dans les navigateurs clients. Son utilisation repose sur la simple mention de la chaîne de caractères « ‘use strict’; ». Il permet ainsi de ne pas poser problème pour les navigateurs ne supportant pas la fonctionnalité. Il est tout de même préférable de mentionner cette valeur dans une portée définie (scope) car son utilisation peut poser problème lors de la concaténation de fichier<sup>107</sup>.

Le mode strict permet de remédier à certaines imperfections du JavaScript :

- il devient impossible de déclarer une variable globale ;
- une exception est levée lors de la redéfinition de mot clé (NaN, arguments, interface, implements, package, private, protected, public, static, ...) ;
- il devient impossible d'avoir des doublons au niveau des clés des tableaux et des noms des variables dans la signature des fonctions ;
- les variables insérées à l'aide de la fonction “eval” ont une portée limitée ;
- et bien d'autres choses<sup>108</sup> ...

#### JSLint

JSLint est un outil de qualité de code créé par Douglas Crockford, créateur de JSON et auteur du livre « JavaScript : The Good Parts ». Il est complémentaire au mode strict, rajoute différents contrôles et va jusqu'à remonter des erreurs lorsque le code ne respectent pas des conventions de codage précises. JSLint possède un validateur en ligne mais aussi un module node.js ainsi que des plugins pour des IDEs et éditeurs de textes. Il est également utilisé par des compilateurs avec par exemple CoffeeScript qui s'appuie dessus pour la compilation et TypeScript qui peut l'utiliser grâce au paquet npm TSLint.

---

<sup>101</sup> Yahoo Developer : <https://developer.yahoo.com/>

<sup>102</sup> Bonne pratiques de Google : <https://developers.google.com/web/fundamentals/index?hl=FR>

<sup>103</sup> Mozilla Developer : <https://developer.mozilla.org/fr/>

<sup>104</sup> Web Plateform : <http://www.webplatform.org/>

<sup>105</sup> Dépôt GitHub de Rick Waldron : <https://github.com/rwaldron/idiomatic.js/>

<sup>106</sup> Dépôt GitHub de Dimitriy Navrotsky : <https://github.com/dypsiLON>

<sup>107</sup> Réponse sur StackOverflow : <http://stackoverflow.com/a/3889829>

<sup>108</sup> Explication du mode strict par Microsoft : [http://msdn.microsoft.com/en-us/library/ie/br230269\(v=vs.94\).aspx](http://msdn.microsoft.com/en-us/library/ie/br230269(v=vs.94).aspx)

## JSHint

JSHint est un autre outil de qualité s'appuyant sur JSLint. Il a été créé par Anton Kovalyov et Paul Irish (développeur de Chrome et contributeur de la librairie Modernizr et Yeoman). JSHint est apparu afin de proposer une alternative à JSLint qui a rajouté, au fur et à mesure de son évolution, des contrôles<sup>109</sup> jugés trop restrictifs. Il est vrai que ces conventions sont toutes des bonnes pratiques mais l'outil ne proposait aucun moyen de les désactiver<sup>110</sup>. Il existe maintenant des annotations<sup>111</sup> afin de contourner certaines règles. JSHint est tout de même plus flexible en offrant un plus grand nombre d'options de configuration. Il permet de choisir les conventions utilisées et ne repose pas sur les règles d'une seule personne mais sur une communauté entière<sup>112</sup>. Il peut ainsi plus facilement s'adapter aux contraintes des projets.

## Code Climate

Code Climate est une solution en ligne analysant la qualité en JavaScript et Ruby. Il commence également à supporter PHP. L'avantage de cet outil est qu'il se couple très facilement à GitHub pour des projets open source. Il permet de visualiser à chaque modification l'évolution du code, la couverture des tests et affiche sur différents « dashboard » s'il y a du code complexe, dupliqué ou mort. Il peut aussi remonter les erreurs de JSHint en se basant sur son fichier de configuration. Code Climate va encore plus loin pour ceux qui veulent payer en proposant des alertes par email, des comparaisons par branche Git et des analyses sur la sûreté du code.

## GTMetrix

D'une manière plus globale, GTMetric est un service permettant d'évaluer un site internet. Il se base à la fois sur l'outil d'analyse de Google (Page Speed<sup>113</sup>) et celui de Yahoo (YSlow<sup>114</sup>) afin d'identifier un ensemble de recommandations optimisant les pages web. Il ne se limite pas qu'au langage JavaScript et peut aller jusqu'à spécifier des bonnes pratiques HTML, CSS ou encore d'autres relatives au cache et à la compression.

Il existe également des outils concurrents comme WebPageTest<sup>115</sup> et d'autres qui s'installent sur un serveur grâce à npm : SiteSpeed<sup>116</sup> et PSI<sup>117</sup> (version node.js de PageSpeed). L'avantage de ces deux derniers est qu'ils peuvent ressortir le résultat des tests dans un format compatible avec des outils d'intégration continue.

---

<sup>109</sup> JSLint vs JSHint : <http://www.scottlogic.com/blog/2011/03/28/jslint-vs-jshint.html>

<sup>110</sup> Argument du créateur de JSHint : <http://anton.kovalyov.net/p/why-jshint/>

<sup>111</sup> Liste des options de configuration disponibles sur JSLint et JSHint : <https://gist.github.com/bretdavidson/3189814>

<sup>112</sup> JSHint Community-Driven Fork : <http://badassjs.com/post/3364925033/jshint-an-community-driven-fork-of-jshint>

<sup>113</sup> Page Speed est disponible ici : <https://developers.google.com/speed/pagespeed/insights/>

<sup>114</sup> YSlow : Plugin disponible à cette adresse : <https://developer.yahoo.com/yslow/>

<sup>115</sup> WebPageTest : <http://www.webpagetest.org/>

<sup>116</sup> SiteSpeed : <http://www.sitespeed.io/>

<sup>117</sup> PSI : <https://github.com/addyosmani/psi/>

## OpQuast

OpQuast est un écosystème améliorant la qualité du Web. Il a été créé par la société Temesis et comprend un ensemble de bonnes pratiques, des outils d'analyse et une aide par des professionnels de la qualité dans le but d'obtenir la certification OpQuast. Contrairement à GTMetrix, il vise également l'accessibilité des sites afin de permettre à toute personne de pouvoir naviguer de manière optimale sur Internet.

# Conclusion

Comme on peut arriver à le comprendre suite à la lecture de ce mémoire, le JavaScript n'est pas un langage aussi trivial que certaines personnes le pensent. Ses nombreuses caractéristiques qui lui permettaient de s'ouvrir à un plus large public, le rendent maintenant difficilement utilisable pour répondre à des applications complexes. Le JavaScript est le langage qui m'aura posé (et me pose encore) le plus de difficultés lors de mon expérience scolaire et professionnelle.

Sa permissivité a permis de faire émerger différents courants de pensée tant au niveau de son utilisation que de son avenir. Chaque jour, le nombre d'outils améliorant nos développements augmente. Par la même occasion, notre confusion face aux différents choix qui s'offrent à nous augmente également.

Ce mémoire a énoncé une approche permettant de choisir un Framework. Il est cependant impossible de répondre à tous les cas de figures. La sélection d'un Framework doit se faire en pensant à la fois au besoin et à son environnement. On peut néanmoins retenir certains d'entre eux qui proposent des solutions pour des problèmes très précis :

- Om/React pour rechercher des performances sur des interfaces complexes ;
- Famo.us afin de concurrencer les applications natives sur les interactions effectuées via les mobiles et les tablettes ;
- Meteor avec son abstraction entre le client et le serveur répondant aux problématiques de SEO, mode hors ligne et sauvegarde côté client.

Le choix d'un compilateur peut également dépendre de l'environnement. Personnellement, j'ai quelques réticences avec Dart étant donné qu'il y a de forte chance qu'il ne soit jamais intégré dans les navigateurs concurrents à Chrome<sup>118</sup>.

Je vois l'avenir du JavaScript dans un compilateur rassemblant à la fois les caractéristiques de TypeScript et ClojureScript. Un compilateur qui :

- garderait la même syntaxe afin de suivre l'évolution du JavaScript sans entrer en conflit ;
- posséderait un typage statique ainsi que la possibilité de rajouter des fonctionnalités à l'aide de macros ;
- serait couplé à un outil de qualité comme JSHint et serait aussi performant que Google Closure pour la compilation.

Nous nous retrouvons maintenant à concevoir des applications complexes avec des technologies provenant du Web. Les outils d'automatisation, de tests et monitoring n'ont plus rien à envier à ceux qui n'existaient autrefois que du côté Back. Le JavaScript ainsi que ses APIs permettent d'interagir de plus en plus avec l'environnement client et d'être entièrement autonome vis-à-vis du serveur. Il devient présent partout et continue d'évoluer.

---

<sup>118</sup> Propos de Brendan Eich (créateur de JavaScript) : <https://news.ycombinator.com/item?id=2982949>

Nous avons parcouru de nouvelles fonctionnalités telles que les « Web Components » et les « Web Sockets ». Les Web Components vont permettre de répondre de manière beaucoup plus optimisée aux interfaces complexes tandis que les Web Sockets répondent aux problématiques du temps réel. Le Web et les APIs JavaScript évoluent à tel point qu'il y en a déjà une nouvelle répondant à d'autres cas de figure sur le temps réel : le WebRTC<sup>119</sup>.

Le WebRTC signifie Web Real-Time Communication. Grâce à cette nouvelle API on ne sera plus obligé de communiquer avec le serveur. On pourra dialoguer directement entre applications. L'apparition de nouveaux protocoles permettront de faire du Peer-To-Peer<sup>120</sup>, d'envoyer des signaux audio, vidéos et de ne plus se limiter aux contraintes du protocole HTTP. Il est donc certain que le JavaScript a un grand avenir devant lui et que son utilisation a des chances de continuer à dépasser nos attentes dans le futur.

---

<sup>119</sup> Vidéo de Google I/O 2013 : <https://www.youtube.com/watch?v=p2HzZkd2A40>

<sup>120</sup> Peer-To-Peer : ou pair à pair en français est un modèle de réseau informatique permettant de dialoguer entre client (le client étant lui-mêmeB serveur)

# Glossaire

<b>AJAX</b>	Asynchronous JavaScript and XML. Ensemble de technologies regroupant le JavaScript, le DOM et le XMLHttpRequest afin de faciliter l'utilisation des applications riches.
<b>API</b>	Pour Application Programming Interface. Ensemble de classes et méthodes servant de façade afin d'interagir avec des données
<b>Bytecode</b>	Code intermédiaire entre les instructions machine et le code source
<b>CLI</b>	Command-Line Interface. Interface de ligne de commande en Français
<b>Command</b>	Le design pattern Command permet d'effectuer une séparation entre une demande et l'action qui en résulte. Il est souvent représenté par un objet « command » permettant de communiquer l'action.
<b>CQRS</b>	Command Query Responsibility Segregation. Principe reposant sur le design pattern command afin d'effectuer une séparation entre les actions de lecture et d'écriture
<b>CRM</b>	Gestion des relations client
<b>Design pattern</b>	Patron de conception en français. Représentation de bonnes pratiques reconnues pour répondre à des besoins particuliers
<b>Front</b>	Représente le client. Un projet Front est donc une application s'exécutant du côté du client plutôt que du serveur.
<b>HTTP</b>	Hypertext Transfer Protocol. Protocole de transfert hypertexte en français permettant de communiquer entre client et serveur sur le Web.
<b>HTML5</b>	HyperText Markup Language 5. Dernière version majeure d'HTML comprenant de nombreuses
<b>Headless browser</b>	Navigateur sans tête en français. Il permet d'interagir avec le DOM sans interface graphique
<b>Industrialisation</b>	Dans le web, l'industrialisation représente différentes techniques permettant de structurer une application afin de la rendre plus stable, et automatiser certaines tâches pour augmenter la productivité des développeurs
<b>Injection de dépendance</b>	Dependency Injection (DI) en anglais et un mécanisme permettant de créer de l'inversion de contrôle
<b>Inversion de contrôle</b>	Ou IoC est un patron d'architecture donnant le contrôle sur le Framework utilisé plutôt que l'application en elle-même
<b>Logs</b>	Historique d'évènement sauvegardé
<b>MakeFile</b>	Fichier de configuration de l'exécutable make

<b>Minifier</b>	Retirer tous les espaces et retour à la ligne inutile afin de réduire la taille du code
<b>MongoDB</b>	Base de données NoSQL la plus populaire
<b>NoSql</b>	Not only SQL. Base de données non relationnelles
<b>Objet immuable</b>	Objet dont l'état ne peut pas changer après sa création
<b>OpenTSDB</b>	Time Series Daemon (TSD) représentant différent processus permettant d'agréger des données
<b>ORM</b>	Objet-Relational Mapping. Technique gérant une base de données avec des objets
<b>Peer-To-Peer</b>	Ou pair à pair en français est un modèle de réseau informatique permettant de dialoguer entre client (le client étant lui-même serveur)
<b>Polyfill</b>	Ensemble de fonction simulant des fonctionnalités que le navigateur ne possède pas
<b>Programmation fonctionnelle</b>	Paradigme renforçant l'utilisation de fonctions et de structures de données immuables (voir glossaire) plutôt que d'objets et de classes
<b>Programmation événementielle</b>	Programmation fondé sur les événements
<b>Programmation asynchrone</b>	Programmation non synchrone (non bloquante pour l'utilisateur)
<b>Prototype</b>	Concept objet n'effectuant aucune différence entre les classes et les instances de classes (les objets)
<b>Proxy</b>	Composant servant intermédiaire permettant de faciliter et/ou surveiller des échanges
<b>REPL</b>	Read-Eval-Print-Loop. Terme utilisé pour représenter les langages interactifs haut niveau comme Lisp.
<b>REST</b>	Representational State Transfer. Ensemble de principes créés par Roy Fielding
<b>RESTful</b>	Qui respecte les principes REST
<b>Routing</b>	Routage en français est le mécanisme déterminant la destination d'une URL
<b>SaaS</b>	Software as service ou logiciel en tant que service en français
<b>SEO</b>	Search Engine Optimization. Optimisation pour les moteurs de recherche (référencement)
<b>Services</b>	Dans le contexte utilisé, les services représentent des objets instanciés par l'injection de dépendance

<b>Shell</b>	Interface utilisateur d'un système d'exploitation
<b>Singleton</b>	Classe objet possédant qu'une seule instance (un seul objet)
<b>SPA</b>	Simple Page Application. Site web entièrement basé sur le JavaScript reposant sur une unique page avec un contenu changé dynamiquement grâce au JavaScript
<b>StatsD</b>	Simple processus agrégant des statistiques
<b>Template</b>	Structure HTML mettant en forme le contenu de la page
<b>Temps réel</b>	Système avec des contraintes temporelles sur les temps de réponses
<b>TDD</b>	Test Driven Development : méthode de développement dirigée par les tests
<b>TSD</b>	TypeScript Definition manager. Gestionnaire de paquets de TypeScript.
<b>Prototype</b>	Concept objet n'effectuant aucune différence entre les classes et les instances de classes (les objets)
<b>Woopra</b>	Outil CRM en temps réel

# Webographie

## Le contexte

- Arnaud Tanielian "Expérimetons le web" publié le 13/12/2013 :  
<http://www.24joursdeweb.fr/2013/experimentons-le-web/>
- David Rousset "Qu'importe le langage pourvu qu'on ait l'ivresse" publié le 04/09/2013 :  
<http://blogs.msdn.com/b/davrous/archive/2013/09/04/qu-importe-le-langage-pourvu-qu-on-ait-l-ivresse.aspx>
- Josh Haberman "The future of JavaScript MVC Frameworks" publié le 22/02/2014 :  
<http://blog.reverberate.org/2014/02/on-future-of-javascript-mvc-frameworks.html>
- Ian Murphy "The State of the Componentised Web" publié le 07/08/2014 :  
<http://www.futureinsights.com/home/the-state-of-the-componentised-web.html>
- Maxime Thirouin "Automatisez votre workflow front-end" publié le 09/12/2013 :  
<http://www.24joursdeweb.fr/2013/automatisez-votre-workflow-front-end/>
- Noëlle Andrieu "Dix ans de pratique JavaScript à Paris" publié le 07/01/2014 :  
<http://letrainde13h37.fr/47/dix-ans-pratique-javascript-paris/>

## Les Frameworks et Compilateurs

### L'utilité des Frameworks JS

- François Petitit :
  - "Les nouvelles architectures front Web et leurs impact sur les DSI - Partie 1" publié le 29/10/2013 :  
<http://blog.octo.com/les-nouvelles-architectures-front-web-et-leur-impact-sur-les-dsi-partie-1/>
  - "Les nouvelles architectures front Web et leur impact sur la DSI - Partie 2" publié le 12/12/2013 :  
<http://blog.octo.com/les-nouvelles-architectures-front-web-et-leur-impact-sur-la-dsi-partie-2/>
- Spike Brehm "Isomorphic JavaScript: The Future of Web Apps" publié le 11/11/2013 :  
<http://nerds.airbnb.com/isomorphic-javascript-future-web-apps/>

### Comment choisir son Framework JS

- Addy Osmani « Journey Through The JavaScript MVC Jungle » publié le 27/07/2012 :  
<http://www.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle/>
- Craig McKeachi « Choosing a JavaScript MVC Framework » publié le 19/09/2013 :  
<http://www.funnyant.com/choosing-javascript-mvc-framework/>

### Comparatifs de Frameworks

- Alex Mingoia "Here's the difference between Polymer and Angular" publié le 26/06/2014 :  
<http://www.binpress.com/blog/2014/06/26/polymer-vs-angular/>

- Axel Rauschmayer “Hello Polymer: Q&A with Google’s Polymer Team” publié le 16/07/2013 :  
<http://www.2ality.com/2013/07/hello-polymer.html>
- Cédric Nisio “La bibliothèque d’interfaces utilisateur React de Facebook reçoit des critiques mitigées” publié le 26/06/2013 :  
<http://www.infoq.com/fr/news/2013/06/facebook-react>
- Cody Lindley “Part 1: Backbone.js Deconstructed” publié le 24/07/2013 :  
<http://tech.pro/tutorial/1367/part-1-backbonejs-deconstructed>
- Eric Bidelman “What is the difference between Polymer elements and AngularJS directives” publié le 07/08/2013 :  
<http://stackoverflow.com/a/18092637>
- Florent Jaby “Ember.js; Framework challenger pour les Single Page application” publié le 01/07/2014 :  
<http://blog.octo.com/ember-js-framework-challenger-pour-les-single-page-applications/>
- Lauren Orsini “Angular, Ember, And Backbone : Which JavaScript Framework is Right For You” publié le 06/02/2014 :  
<http://readwrite.com/2014/02/06/angular-backbone-ember-best-javascript-framework-for-you>
- Matthias Dugué “Framework MV\*: Kikifékoï” publiée le 18/09/2013 :  
[http://www.dailymotion.com/video/x14v5zk\\_caen-js-2013-3-8-frameworks-mv-kikifeko\\_tech](http://www.dailymotion.com/video/x14v5zk_caen-js-2013-3-8-frameworks-mv-kikifeko_tech)
- Matthias Schäfer “JavaScript MVC Frameworks : A comparison of Marionette and Chaplin” publié le 08/04/2013 :  
<http://9elements.com/io/index.php/comparison-of-marionette-and-chaplin/>
- Max Lynch “Why AngularJS Will Be Huge” publié le 04/02/2014 :  
<http://ionicframework.com/blog/angularjs-will-be-huge/>
- Pete Hunt :
  - “React: Rethinking best practices” publiée le 30/10/2013 :  
<https://www.youtube.com/watch?v=x7cQ3mrckAY>
  - “The Secrets of React’s Virtual DOM” publié le 13/03/2014 :  
<http://fluentconf.com/fluent2014/public/schedule/detail/32395>
- Paul Shan “Why AngularJS is generally better in Angular vs Ember vs Backbone” publié le 12/11/2013 :  
<http://voidcanvas.com/why-angularjs-is-generally-better-than-emberjs-and-backbonejs/>
- Rafal Pocztarski “Pros and Cons of Facebook’s React vs Web Components (Polymer)” publié le 03/05/2014 :  
<http://programmers.stackexchange.com/a/237762>
- Sebastian Porto “A Comparison of Angular, Backbone, CanJS and Ember” publié le 12/08/2013 :  
<http://sporto.github.io/blog/2013/04/12/comparison-angular-backbone-can-ember/>
- Seth Ladd “Angular and Polymer Data Binding, Together!” publié le 05/02/2014 :  
<http://blog.sethladd.com/2014/02/angular-and-polymer-data-binding.html>
- TJ VanToll :

- “Why Web Components Aren’t Ready for Production... Yet” publié le 17/06/2014 :  
<http://developer.telerik.com/featured/web-components-arent-ready-production-yet/>
  - “Why Web Components Are Ready For Production” publié le 07/08/2014 :  
<http://developer.telerik.com/featured/web-components-ready-production/>
- Yves Amsellem et Pierre Bayallet “Backbone, deux ans après” publié le 22/05/2013 :  
<http://blog.xebia.fr/2013/05/22/backbone-deux-ans-apres/>

## Les Compilateurs

- Anders Hejlsberg et Lars Bak “TypeScript, JavaScript and Dart” publiée le 09/01/2013 :  
[https://www.youtube.com/watch?v=QTj6Q\\_zV1yg](https://www.youtube.com/watch?v=QTj6Q_zV1yg)
- Antoine Crochet “Le temps de compilation : le grant défaut de GWT” publié le 23/01/2013 :  
<http://www.journaldunet.com/developpeur/outils/google-web-toolkit-l-avis-des-developpeurs/gwt-les-points-faibles.shtml>
- David Nolen :
  - “The Future of JavaScript MVC Frameworks” publié le 17/12/2013 :  
<http://swannodette.github.io/2013/12/17/the-future-of-javascript-mvcs/>
  - “ClosureScript : Lisp’s Revenge” publiée le 08/04/2014 :  
<https://www.youtube.com/watch?v=MTawgp3SKy8>
- Florent Garin “GWT est-il toujours pertinent” publié le 23/02/2012 :  
<http://www.docdoku.com/blog/2012/02/23/gwt-est-il-toujours-pertinent/>
- Florian Loitsch “JavaScript as a compilation target Making it fast” publié le 06/10/2012 :  
<https://www.dartlang.org/slides/2012/10/jsconfeu/javascript-as-compilation-target-florian-loitsch.pdf>
- Hadrien Lanneau “AngularDart” publié le 26/04/2014 :  
<http://blog.hadrien.eu/2014/05/26/angulardart/>
- James Long “Stop Writing JavaScript Compilers! Make Macros Instead” publié le 07/01/2014 :  
<http://jlongster.com/Stop-Writing-JavaScript-Compilers--Make-Macros-Instead>
- Jeff Walker :
  - “The JavaScript Minefield” publié le 20/02/2014 :  
<http://www.walkercoderanger.com/blog/2014/02/javascript-minefield/>
  - “Why TypeScript isn’t the Answer” publié le 27/02/2014 :  
<http://www.walkercoderanger.com/blog/2014/02/typescript-isnt-the-answer/>
  - “Why Dart Isn’t the Answer” publié le 08/03/2014 :  
<http://www.walkercoderanger.com/blog/2014/03/dart-isnt-the-answer/>
  - “Why CoffeeScript isn’t the Answer” publié le 28/03/2014 :  
<http://www.walkercoderanger.com/blog/2014/03/coffeescript-isnt-the-answer/>
  - “Are JavaScript Linters the Answer?” publié le 03/04/2014 :  
<http://www.walkercoderanger.com/blog/2014/04/are-javascript-linters-the-answer/>
  - “What I Think CoffeeScript Should Have Been” publié le 16/04/2014 :  
<http://www.walkercoderanger.com/blog/2014/04/what-coffeescript-should-have-been/>
- Jeremy Ashkenas “CoffeeScript as a JS/Next” publié le 28/01/2013 :  
[https://www.youtube.com/watch?v=QTj6Q\\_zV1yg](https://www.youtube.com/watch?v=QTj6Q_zV1yg)

- Johannes Rieken “Building a Large Scale JavaScript Application in TypeScript” publiée le 19/05/2014 : <https://www.youtube.com/watch?v=3Jrg8hfNXmU>
- Kevin Lynagh “Extending libraries from ClojureScript” publiée le 07/01/2013 : <https://www.youtube.com/watch?v=XfzXFWTT-z0>
- Matthieu Lux “Développements de Google Dart : Polymer remplace Web UI” publié le 20/08/2013 : <http://www.infoq.com/fr/news/2013/08/dart-polymer-web-ui>
- Matthias Dugué “CoffeeScript, le JS++” publié le 22/01/2013 : <http://fr.clever-age.com/veille/blog/du-javascript-avec-coffeescript.html>
- Murilo Pereira “The Case for React.js and ClojureScript” publié le 04/05/2014 : <http://fr.slideshare.net/murilasso/the-case-for-reactjs-and-clojurescript>
- Omer Sensoy “CoffeeScript Presentation - Code Like You Talk” publiée le 01/02/2014 : [https://www.youtube.com/watch?v=-\\_arPK51bUo](https://www.youtube.com/watch?v=-_arPK51bUo)
- Paul Oliver “CoffeeScript vs TypeScript” publié le 18/08/2013 : <http://prezi.com/zkhsz49ownaw/coffeescript-vs-typescript/>
- Seth Ladd “I ported a JavaScript app to Dart. Here's what I learned” publié le 09/05/2014 : <http://blog.sethladd.com/2014/05/i-ported-javascript-app-to-dart-heres.html>

## La révolution apparue avec node.js

### La réponse à de réels besoins

- Ben Cherry “Thoughts on the Hashbang” publié le 11/02/2011 : <http://www.adequatelygood.com/Thoughts-on-the-Hashbang.html>
- Ben Coe “npm Enterprise Roadmap” publié le 05/08/2014 : <http://blog.npmjs.org/post/93509138505/npm-enterprise-roadmap>
- Callum Macrae “Building With Gulp” publié le 11/06/2014 : <http://www.smashingmagazine.com/2014/06/11/building-with-gulp/>
- Cédric Beust “Language popularity on Github” publié le 03/05/2014 : <http://beust.com/weblog/2014/05/03/language-popularity-on-github/>
- Chris Coyier “Grunt for People Who Think Things like Grunt are Weird and Hard” paru le 11/12/2013 : <http://24ways.org/2013/grunt-is-not-weird-and-hard/>
- Dan Tello “Gulp + Browserify : The Everything Post” publié le 08/04/2014 : <http://viget.com/extend/gulp-browserify-starter-faq>
- Dan Webb “It's About The Hashbangs” publié le 28/05/2011 : <http://danwebb.net/2011/5/28/it-is-about-the-hashbangs>
- Erik Hendriks et Michael Xu “Understanding web pages better” publié le 23/05/2014 : <http://googlewebmastercentral.blogspot.fr/2014/05/understanding-web-pages-better.html>

- Frédéric Notet “Node.js vs PHP” publié le 19/03/2014 : <http://blog.watopa.sh/node-js-vs-php/>
- Guillaume Ehret “Yeoman où comment être productif quand on crée une application web moderne” publié le 06/05/2013 : <http://javamind-fr.blogspot.fr/2013/05/yeoman-ou-comment-etre-productif-quand.html>
- Isaac Z. Schlueter “npm Progress : May” publié le 23/05/2014 : <http://blog.npmjs.org/post/86636834990/npm-progress-may>
- Issac Roth “What Makes Node.js Faster Than Java” publié le 30/01/2014 : <http://strongloop.com/strongblog/node-js-is-faster-than-java/>
- Jeff Atwood “The Sitemap Paradox” publié le 1/11/2010 : <http://webmasters.stackexchange.com/questions/4803/the-sitemap-paradox>
- Josph Scott “Twitter Backing Away From Hashbang URLs” publié le 30/05/2012 : <https://josephscott.org/archives/2012/05/twitter-backing-away-from-hashbang-urls/>
- Hadrien Lanneau “Gulp” publié le 11/02/2014 : <http://blog.hadrien.eu/2014/02/11/gulp/>
- Mathieu Nebra “Des applications ultra-rapides avec Node.js” mis à jour le 08/08/2014 : <http://fr.openclassrooms.com/informatique/cours/des-applications-ultra-rapides-avec-nodejs/nodejs-mais-a-quoi-ca-sert>
- Nicolas Froidure “Gulp remplacera-t-il Grunt” publié le 23/12/2013 : [http://www.insertafter.com/articles-gulp\\_vs\\_grunt.html](http://www.insertafter.com/articles-gulp_vs_grunt.html)
- Nourdine Falola “Bower : un gestionnaire de dépendances pour le web” publié le 19/08/2014 : <http://blog.soat.fr/2014/08/bower-un-gestionnaire-de-dependances-pour-le-web/>
- Olivier Penhoat “Référencer une application web Single Page - AngularJS, NodeJS, PhantomJS et jsDom” publié le 18/07/2013 : <http://blog.octo.com/seo-spa-angular/>
- Orun Bhuiyan “Goolge’s Crawler Now Understands JavaScript : What Does This Mean For You” publié le 30/05/2014 : <http://www.business2community.com/seo/googles-crawler-now-understands-javascript-mean-0898263#!bLJU1r>
- Paul Bakaus “Supercharging your Gruntfile” paru le 13/02/2014 : <http://www.html5rocks.com/en/tutorials/tooling/supercharging-your-gruntfile/>
- Rand Fishkin “White Hat Cloaking : It Exists. It’s Permitted. It’s Useful” publié le 30/06/2008 : <http://moz.com/blog/white-hat-cloaking-it-exists-its-permitted-its-useful>
- Raphael Goetter “Bower pour les nuls” publié le 28/12/2013 : <http://www.alsacreations.com/tuto/lire/1609-bower-pour-les-nuls.html>
- Rey Bango “Managing Your Build Tasks With Gulp.js” publié le 10/02/2014 : <http://code.tutsplus.com/tutorials/managing-your-build-tasks-with-gulpjs--net-36910>
- Rob Ousbey :
  - “Create Crawlable, Link-Friendly AJAX Websites Using pushState()” publié le 26/02/2012 : <http://moz.com/blog/create-crawlable-link-friendly-ajax-websites-using-pushstate>

- “Future Technologies” publié le 23/02/2012 : <http://fr.slideshare.net/RobOusbey/new-technologies-11724391/>
- Romain Maton “Node.js partie 1 - Tout ce que vous devez savoir sur node.js” publié le 15/02/2011 : <http://web-tambouille.fr/2011/02/15/node-js-partie-1-tout-ce-que-vous-devez-savoir-sur-node-js.html>
- Rudy Rigot “How Google is redefining web development (And Backbone.js is getting obsolete)” publié le 01/06/2014 : <http://rudyonweb.net/google-redefines-web-development-backbone-obsolete/>
- Sébastien Lucas “Node.js : la tendance JavaScript côté serveur” publié le 14/01/2014 : <http://www.journaldunet.com/developpeur/expert/56253/node-js---la-tendance-javascript-cote-serveur.shtml>
- Shekhar Gulati “Day 24 : Yeoman Ember -- The Missing Tutorial” paru le 21/11/2013 : <https://www.openshift.com/blogs/day-24-yeoman-ember-the-missing-tutorial>
- Travis Maynard “No Need to Grunt, Take A Gulp of Fresh Air” publié le 13/12/2013 : <http://travismaynard.com/writing/no-need-to-grunt-take-a-gulp-of-fresh-air>

## Les Frameworks côté serveur

- Alex Ivanovs “7 Minimal Node.js Web Frameworks for 2014 and Beyond” paru le 04/02/2014 : <http://codecondo.com/7-minimal-node-js-web-frameworks/>
- Antony Wing Kosner “Famo.us Part I” paru le 27/05/2014 : <http://www.forbes.com/sites/anthonykosner/2014/05/27/famo-us-part-i-new-concepts-will-increase-the-flow-of-highly-dynamic-web-3-0-apps/>
- Jeff Harrell “Open Sourcing Kraken.js” paru le 03/03/2014 : <https://wwwpaypal-engineering.com/2014/03/03/open-sourcing-kraken-js/>
- Oliver Avital “The new Meteor rendering engine” Conférence vidéo publiée le 18/10/2013 : <https://www.youtube.com/watch?v=WSSJ-L9a7NY&index=105&list=UU3fBiJrFFMhKlsWM46AsAYw>
- Ritik Malhotra “Building a Production-Reader Meteor App” Conférence vidéo publiée le 05/11/2013 : <https://www.youtube.com/watch?v=gfFGjmiKfnA&list=UU3fBiJrFFMhKlsWM46AsAYw>
- Vikas “10 Best Node.js MVC Frameworks for JavaScript Developers” paru le 23/02/2014 : <http://designzum.com/2014/02/23/10-best-node-js-mvc-frameworks-for-javascript-developers/>

## La recherche de la qualité

- Addy Osmani “Automating Web Performance Measurement” paru le 06/06/2014 : <http://updates.html5rocks.com/2014/06/Automating-Web-Performance-Measurement>
- Andrew Childs “Client-side Testing with Jasmine, CasperJS and JsTestDriver” paru le 05/12/2012 : <https://speakerdeck.com/andrewchilds/client-side-testing-with-jasmine-casperjs-and-jstestdriver>

- Azami Novak “DevOps Roundup: Gene Kim’s Future Talk, StrongOps 2.0, and More” paru le 18/04/2014 : <http://blog.newrelic.com/2014/04/18/devops-roundup-4-18-14/>
- Jan Stenberg “BDD et JavaScript avec CucumberJS” paru le 18/02/2014 : <http://www.infoq.com/fr/news/2014/02/bdd-cucumberjs>
- Jeff Walden “In ECMAScript5, what’s the scope of “use strict”?” paru le 03/12/2010 : <http://stackoverflow.com/questions/2343608/in-ecmascript5-whats-the-scope-of-use-strict/3889829#3889829>
- Kevin Groat “Which JavaScript test library should you use ? QUnit vs Jasmine vs Mocha” paru le 04/04/2014 : <http://www.techtalkdc.com/which-javascript-test-library-should-you-use-qunit-vs-jasmine-vs-mocha/>
- Lara Swanson “Designing for Performance” paru le 21/02/2014 : <https://speakerdeck.com/lara/design-for-performance>
- Luke Page “JSLint vs JSHint” paru le 28/03/2011 : <http://www.scottlogic.com/blog/2011/03/28/jslint-vs-jshint.html>
- Mathieu Robin “En une minute : Sitespeed.io, analysez la performance de vos sites” paru le 14/01/2014 : <http://www.mathieuRobin.com/2014/02/en-une-minute-sitespeed-io-analysez-la-performance-de-vos-sites/>
- Mathieu Robin “En une minute : Sitespeed.io, analysez la performance de vos sites “ paru le 04/02/2014 : <http://www.mathieuRobin.com/2014/02/en-une-minute-sitespeed-io-analysez-la-performance-de-vos-sites/>
- Matt Wynne “TDD vs BDD” paru le 20/11/2012 : <http://blog.mattwynne.net/2012/11/20/tdd-vs-bdd/>
- Microsoft “Strict mode (JavaScript)” : [http://msdn.microsoft.com/en-us/library/ie/br230269\(v=vs.94\).aspx](http://msdn.microsoft.com/en-us/library/ie/br230269(v=vs.94).aspx)
- Pierre Col “Fasterize, une start-up française qui accélère les sites web” paru le 29/12/2013 : <http://www.zdnet.fr/actualites/fasterize-une-start-up-francaise-qui-accelere-les-sites-web-39796620.htm>
- Zack Bloom “Client-side Performance Monitoring” paru le 10/10/2013 : <http://dev.hubspot.com/blog/client-side-performance-monitoring>

# Tables des matières

SOMMAIRE.....	2
REMERCIEMENTS .....	3
PREAMBULE.....	4
INTRODUCTION .....	5
I. LE CONTEXTE .....	6
1.1. HISTORIQUE DU LANGAGE.....	6
1.2. DES BESOINS DE PLUS EN PLUS IMPORTANTS.....	8
1.3. L'APPARITION DE NOUVELLES PROBLEMATIQUES .....	9
II. LES FRAMEWORKS ET COMPILATEURS .....	10
2.1. L'UTILITE DES FRAMEWORKS JS .....	10
2.2. COMMENT CHOISIR SON FRAMEWORK JS ?.....	13
2.2.1. EST-IL VRAIMENT NECESSAIRE D'UTILISER UN FRAMEWORK ? .....	13
2.2.2. QUELLES SONT LES FONCTIONNALITES DONT NOUS AVONS BESOIN ? .....	13
2.2.3. QUELLES SONT LES COMPATIBILITES QUE JE DOIS PRENDRE EN COMPTE ? ....	13
2.2.4. AVONS-NOUS BESOIN DE FLEXIBILITE OU DE RIGIDITE ? .....	13
2.2.5. LE FRAMEWORK EST-IL MATURE ?.....	14
2.2.6. LA COMMUNAUTE DU FRAMEWORK EST-ELLE IMPORTANTE ?.....	14
2.2.7. AVONS-NOUS DES CONTRAINTES DE PERFORMANCES ?.....	14
2.2.8. QUELLES SONT LES COMPETENCES TECHNIQUES DE L'EQUIPE DE DEVELOPPEMENT ?	15
2.3. COMPARATIF DE FRAMEWORKS .....	16
2.3.1. LES FRAMEWORKS SELECTIONNES .....	16
BACKBONE.JS .....	16
EMBER.JS .....	17
ANGULAR.JS .....	17
POLYMER.JS .....	17
REACT.JS.....	18
2.3.2. LEURS CARACTERISTIQUES .....	18
BACKBONE.JS .....	18
EMBER.JS .....	18
ANGULAR.JS .....	18
POLYMER.JS .....	19
REACT.JS.....	19
2.3.3. LEURS COMPATIBILITES ET DEPENDANCES.....	19
LEURS COMPATIBILITES.....	19
2.3.4. LEURS ARCHITECTURES .....	20
BACKBONE.JS .....	20
EMBER.JS .....	20
ANGULAR.JS .....	22
POLYMER.JS .....	23
REACT.JS.....	24
2.3.5. LEURS MATURITES .....	27
BACKBONE.JS .....	27

EMBER.JS .....	27
ANGULAR.JS .....	27
POLYMER.JS .....	27
REACT.JS.....	27
2.3.6. LEURS COMMUNAUTES .....	28
2.3.7. LEURS TAILLES .....	29
2.3.8. LEURS COURBES D'APPRENTISSAGES ET DE PRODUCTIVITE .....	30
BACKBONE.JS .....	30
EMBER.JS .....	30
ANGULAR.JS .....	31
POLYMER.JS .....	32
REACT.JS.....	32
2.4. LES COMPILATEURS.....	33
2.4.1. L'UTILITE DES COMPILATEURS.....	33
LE JAVASCRIPT EST TROP PERMISSIF .....	34
PROBLEME DE CLASSES.....	34
PROBLEME D'EXPRESSION .....	34
PROBLEME DE SCALABILITE .....	34
2.4.2. COFFEESCRIPT .....	35
2.4.3. TYPESCRIPT.....	37
2.4.4. DART .....	39
2.4.5. SWEET.JS.....	42
2.4.6. CLOJURESCRIPT.....	44
LEININGEN .....	45
OM .....	46
GOOGLE CLOSURE .....	47
III. LA REVOLUTION APPARUE AVEC NODE.JS .....	48
3.1. LA REPONSE A DE REELS BESOINS.....	48
3.1.1. LE TEMPS REEL.....	49
3.1.2. LE REFERENCEMENT .....	52
LE CLOAKING.....	52
LA BALISE <NOSCRIPT>.....	54
LES « HASHBANGS ».....	54
LA FONCTIONNALITE HTML5 « PUSHSTATE ».....	55
PHANTOMJS .....	56
JsDOM .....	56
PRERENDER.....	56
3.1.3. LES GESTIONNAIRES DE PAQUETS.....	57
NPM .....	57
BOWER.....	58
3.1.4. LES OUTILS D'AUTOMATISATION .....	59
GRUNT .....	59
GULP.....	60
YEOMAN .....	61
3.2. LES FRAMEWORKS COTE SERVEUR.....	63
3.2.1. EXPRESS.JS .....	63
EXPRESS.IO .....	63

HAPI .....	63
KRAKENJS .....	63
SAILS.JS .....	64
LOCOMOTIVE .....	64
3.2.2. METEOR.JS .....	64
FAMONO .....	65
3.3. LA RECHERCHE DE LA QUALITE .....	66
3.3.1. LES TESTS .....	66
MOCHA.JS .....	66
JASMINE.JS .....	66
CUCUMBER.JS .....	67
CASPERJS .....	67
ISTANBUL .....	67
KARMA .....	68
3.3.2. LES OUTILS DE MONITORING .....	69
LES OUTILS DE DEBOGAGE .....	69
GOOGLE ANALYTICS .....	69
BUCKY .....	70
STRONGLOOP AGENT .....	71
NEW RELIC .....	71
FASTERIZE .....	72
3.3.3. LE RESPECT DES BONNES PRATIQUES .....	74
LE MODE STRICT .....	74
JSLINT .....	74
JSHINT .....	75
CODE CLIMATE .....	75
GTMETRIX .....	75
OPQUAST .....	76
CONCLUSION .....	77
GLOSSAIRE .....	79
WEBOGRAPHIE .....	82
LE CONTEXTE .....	82
LES FRAMEWORKS ET COMPILEURS .....	82
L'UTILITE DES FRAMEWORKS JS .....	82
COMMENT CHOISIR SON FRAMEWORK JS .....	82
COMPARATIFS DE FRAMEWOKS .....	82
LES COMPILEURS .....	84
LA REVOLUTION APPARUE AVEC NODE.JS .....	85
LA REPONSE A DE REELS BESOINS .....	85
LES FRAMEWORKS COTE SERVEUR .....	87
LA RECHERCHE DE LA QUALITE .....	87
TABLES DES MATIERES .....	89