

JavaScript, un langage web?

BARASCUT Jérémy

3 juin 2013

Résumé

Le résumé en français de mon article.

Résumé

Le résumé (abstract en anglais) de mon article.

Table des matières

1	Introduction	6
1.1	Architecture 3-tiers	8
1.1.1	Présentation	8
1.2	Histoire de JavaScript	9
1.2.1	Présentation	9
1.2.2	Nom de code Mocha	10
1.3	EcmaScript	13
1.3.1	Présentation	13
1.4	JSON	14
1.4.1	Présentation	14
1.5	Ajax	16
1.5.1	Présentation	16
1.6	JQuery	19
1.6.1	Introduction	19
1.6.2	Document Object Model (DOM)	20
1.6.3	Les autres forces de jQuery	22
1.7	Restfull	23
1.7.1	Présenation	23
1.8	MVC	24
1.8.1	Une structure	24
1.8.2	Qu'est ce que MVC ?	24
1.9	Conclusion	26
2	Client	27
2.1	Introduction	27
2.2	Ember	29

2.2.1	Présentation	29
2.3	Backbone.js	31
2.3.1	Introduction	31
2.3.2	Dépendances	31
2.3.3	Composants	32
2.3.4	Les données	32
2.3.5	La vue	32
2.3.6	Asynchronous Module Definition API	32
2.3.7	Plugins	33
2.4	AngularJS	34
2.4.1	Introduction	34
2.4.2	Objectifs de conception du framework	34
2.5	CanJS	36
2.5.1	Présentation	36
2.5.2	Caractéristiques vs Poids	36
2.5.3	Facilité d'utilisation	37
2.5.4	Prévention des fuites mémoire	37
2.5.5	Performance	37
2.5.6	Support de librairie	38
2.6	Conclusion	39
2.6.1	Fonctionnalités	39
2.6.2	Flexibilité	40
2.6.3	La documentation et la courbe d'apprentissage	41
2.6.4	La productivité des développeurs	42
2.6.5	Communauté	43
2.6.6	Ecosystème	43
2.6.7	Poid	43
2.6.8	Performance	44
2.6.9	Maturité	45
2.6.10	Sécurité des fuites mémoire	45
2.6.11	Testabilité	45
2.6.12	Goût personnel	46
2.6.13	Total	46

3	Serveur	48
3.1	Introduction	48
3.2	Node.js	49
3.2.1	Présentation	49
3.2.2	Architecture de Node.js	50
3.2.3	Boucle d'évènement	51
3.2.4	I/O non bloquant	51
3.2.5	Asynchrone	51
3.2.6	Performance	52
3.2.7	Communauté	52
3.2.8	Forces	52
3.2.9	Retour d'expérience Linkedin	53
3.3	NPM	54
3.3.1	Présentation	54
3.4	Socket.IO	55
3.4.1	Présentation	55
3.5	Meteor.js	56
3.5.1	Présentation	56
3.5.2	Principes	56
3.5.3	Live HTML	57
3.5.4	Compensation de la latence	57
3.5.5	Temps réel	57
3.5.6	Rafraichissement automatique	58
3.6	Conclusion	59
3.6.1	Wordsquared.com	59
4	Base de données	60
4.1	Introduction	60
4.2	SGBDR	60
4.3	MongoDB	62
4.3.1	Présentation	62
4.3.2	Schéma libre	62
4.3.3	MongoDB vs SGBDR	63
4.3.4	JavaScript	63

4.3.5	Types de données	63
4.3.6	JSON Document-Oriented	64
4.3.7	Temps-réel	64
4.3.8	Fonctionnalités avancées	64
4.3.9	Connecteur NodeJS	64
4.3.10	Conclusion	64
4.4	Web Storage	65
4.4.1	Présentation	65
4.5	Conclusion	66
5	Client lourd	67
5.1	Windows 8	67
5.1.1	Présentation	67
5.2	Gnome3	68
5.2.1	Présentation	68
5.3	Conclusion	69
6	Smartphone	70
6.1	Introduction	70
6.1.1	La croissance du Web mobiles	71
6.1.2	Nouveaux paradigmes	72
6.2	Phonegap	73
6.2.1	Introduction	73
6.3	SenchaTouch	75
6.3.1	Introduction	75
6.3.2	Un peu d'histoire	76
6.3.3	Ext JS est né	76
6.3.4	Caractéristiques principales	77
6.3.5	Supprt appareils et navigateurs	77
6.3.6	Licence	77
6.4	JQuery Mobile	79
6.4.1	Introduction	79
6.4.2	Plateformes supportés	79
6.4.3	Compatibilité	79

6.4.4	Compatibilités avec les anciennes plateformes mobiles	80
6.4.5	Principales caractéristiques	81
6.5	Conclusion	83
7	Plateforme	84
7.1	Wakanda	84
7.1.1	Présentation	84
8	Futur	86
8.1	Ecmascript6	86
8.1.1	Présentation	86
8.1.2	Compatibilité	86
8.1.3	Nouveautés	88
8.2	Futur de JavaScript	89
8.2.1	Du web 1.0 au web 2 squarred	89
8.2.2	Pourquoi parler du Web Squared?	90
8.2.3	L'essor du temps réel : un cerveau collectif	90
8.2.4	Et JavaScript dans tout ça?	91
9	Conclusion	92

1 Introduction

Le développement JavaScript a nettement changé depuis sa conception. Il est facile d'oublier la mise en œuvre de JavaScript dans le navigateur Netscape, jusqu'au navigateur d'aujourd'hui avec de puissants moteurs, tels que le moteur V8 de Google.

Cela a été un chemin rocailleux impliquant renommage, fusion et normalisation éventuelle comme ECMAScript. Les capacités de JavaScript que nous avons aujourd'hui sont au delà des rêves les plus fous de ses concepteurs.

Malgré son succès et sa popularité, JavaScript est encore largement méconnu. Peu de gens savent que JavaScript est un langage puissant et orienté objet. Ils sont surpris d'en apprendre davantage sur certaines de ses fonctionnalités les plus avancées, telles que l'héritage, les modules et les espaces de noms. Alors, pourquoi JavaScript est-il si mal compris ?

Un premier élément de réponse s'explique par précédentes implémentations de JavaScript boguées, et le second élément de réponse proviens du nom du préfixe Java suggérant que JavaScript est en quelque sorte lié à Java. En réalité la raison de cette incompréhension est totalement différente. La véritable raison est la façon dont la plupart des développeurs sont initiés à ce langage. Avec d'autres langages, tels que Python ou Ruby, les développeurs font généralement un effort concerté pour apprendre le langage avec l'aide de livres, screencasts et tutoriaux. Jusqu'à récemment, les développeurs recevaient des demandes pour ajouter un peu de validation de formulaire, peut être ajouter un album ou une galerie photo avec du code tout prêt ou un calendrier. Ils utilisaient des scripts qu'ils trouvaient sur Internet, appelé avec peu de compréhension du langage derrière le script. Après cette phase, certains développeurs ajoutent JavaScript à leurs CV.

Récemment les moteurs JavaScript et les navigateurs sont devenus si puissant que la construction complète d'applications riches en JavaScript est non seulement faisable mais aussi de plus en plus populaire. Les applications tels que Gmail et Google Maps ont ouvert la voie à une manière complètement différente de penser les applications Web, et les utilisateurs en réclament plus. Les entreprises embauchent pour répondre à la demande des développeurs JavaScript à temps

plein. Ce n'est plus un sous-langage relégué à des scripts simples et un peu de validation de formulaire, il est désormais un langage autonome de son propre droit, avec un sérieux potentiel. Cet afflux de popularité signifie qu'un grand nombre de nouvelles applications JavaScript sont en cours de construction, surtout avec l'émergence de toujours plus de périphériques autonomes comme les smartphones, les tablettes etc

Malheureusement, et peut être en raison de l'histoire de ce langage, beaucoup d'applications JavaScript sont mal conçus. Peu importe la raison, les modèles reconnus et les meilleures pratiques partent à la poubelle. Les développeurs ignorent les modèles architecturaux tels que le Modèle-Vue-Contrôleur (MVC), mêlant à la place un désordre de HTML et JavaScript à leurs applications .

Ce mémoire à pour but de répondre à la problématique suivante :

Est-il possible de faire une application 100% en JavaScript ?

En suivant une architecture trois-tiers ? Et en développant une seule fois ou en réutilisant des parties de l'application ?

Je vais au travers de ce document, présenter des bibliothèques, plateformes et frameworks pour structurer et construire des applications complexe entièrement en JavaScript dans le cas du application trois-tiers. Que se soit la partie serveur, les clients légers comme les navigateurs, les clients lourds avec les applications Windows 8 et Gnome 3, les périphériques mobiles comme les smartphones, tablettes, tv connectés et liseuses ainsi que le stockage des données avec l'utilisation de bases de données. Afin d'être le plus complet possible dans le développement d'une application rigoureuse, ce mémoire inclut également la présentation d'outils utilisé pour créer une application de qualité avec l'inclusion des tests, de la documentation ...

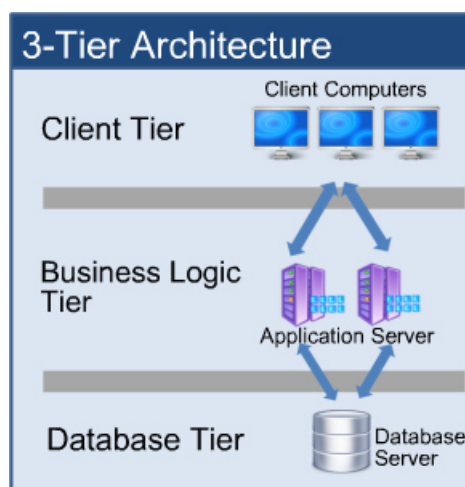
Ce mémoire n'a pas pour but l'apprentissage du langage, pour ceux qui veule en apprendre plus sur le langage JavaScript de nombreux livres sont disponibles pour apprendre à sa syntaxe et sa structure.

1.1 Architecture 3-tiers

1.1.1 Présentation

Dans le monde du développement informatique, un modèle d'architecture s'est imposé dans la réalisation d'application. Le modèle est appelé "Architecture trois tiers". Il s'agit d'un modèle logique d'architecture applicative qui vise à modéliser une application en la divisant en trois couches.

Le but du 3 tiers est de séparer les 3 couches habituelles d'une application : IHM, traitements et données. Une application sera composée de 3 couches indépendantes :



- La couche **présentation** des données : correspondant à la restitution et à l'affichage sur le clients, le dialogue avec l'utilisateur ;
- Le **traitement** métier des données : correspondant à la mise en oeuvre de l'ensemble des règles de gestion et de la logique applicative ;
- et enfin l'**accès aux données** persistantes : correspondant aux données qui sont destinées à être conservées sur la durée, voir de manière définitive

Cette séparation a pour but de rendre indépendantes chacune des couches afin de simplifier la maintenance et les évolutions futures de l'application (par exemple, changement de système de base de données, portage d'un environnement graphique à un autre, ...). Elle assure une sécurité plus importante car l'accès à la base de données n'est autorisé que par la couche Traitements. Elle a également l'avantage d'optimiser le travail en équipe et de réduire les coûts de développement et de maintenance.

1.2 Histoire de JavaScript

1.2.1 Présentation

Javascript (js pour les intimes) est un langage de script créé en 1995 par Brendan Eich pour le compte de Netscape Communications.

A cet époque Netscape Communications domine le marché du web. La firme fournit le navigateur web le plus populaire Netscape Navigator mais aussi des logiciels serveurs.

Netscape Communications lançait un partenariat avec Sun Microsystems pour exploiter leur nouveau langage ainsi que sa VM multi-plateforme, Java. Le but étant d'utiliser Java au sein du navigateur du côté serveur afin de fournir des UI riches portables ainsi qu'un moyen d'accéder à des applications via le web à travers un simple navigateur.

Cependant, Java est perçu par Sun et Netscape comme un langage peu adapté à une utilisation simple applicable à la page web car trop professionnel et contraignant. Java est à l'époque en concurrence avec Visual C++ de Microsoft. Il faut donc un langage plus simple à écrire, plus facile à utiliser pour les développeurs débutant et à prendre en main.

Netscape et Sun décide de créer un nouveau langage répondant à cette demande et en confie la création à Brendan Eich, gourou technique chez Netscape. La seule condition est que le langage doit être “dans le style java” mais en moins puissant et clairement différenciés de java par les potentiels développeurs.

Comme l'a expliqué Brendan Eich dans une interview :

JS devait « ressembler à Java », mais en moins avancé, [il devait] être le petit frère simplet de Java, son partenaire-otage. Et par-dessus le marché, je n'avais que dix jours pour pondre ça, ou on se retrouverait avec un truc pire que JS.

Javascript utilise la même syntaxe que Java (JS 1.0, mots-clés réservés et convention du JDK) mais il a dû s'abstenir d'utiliser la syntaxe orientée-objet du langage, à une époque où la POO (Programmation Orientée Objet) était encore considérée comme un sujet réservé aux professionnels...

Brendan ne voulant pas écrire un langage diminué, il a dû trouver des ruses pour y glisser assez de puissance sans que celle-ci ne soit immédiatement visible aux profanes. Le langage devait rester simple et léger en apparence, tout en ayant assez de sophistication pour que des développeurs avancés soient à même d'en tirer des applications puissantes.

Même si Javascript et Java appartiennent à une famille syntaxique “de type C” (syntaxe des identifiants, accolades, opérateurs principaux, structures de contrôle...), ils ont des sémantiques

extrêmement différentes. Javascript à une philosophie très fortement inspirée de langage objet ou fonctionnels “purs” au premier rang desquels Scheme et Self, mais aussi certains aspects de LISP et SmallTalk.

Java est un langage statique (chaque variable est typée), compilée et dotée d’un typage fort, là ou JavaScript est dynamique, interprété avec un typage plus léger.

Java utilise un modèle d’héritage “classique”, mono-parent, basé sur l’héritage de classes. JavaScript s’appuie également sur les prototypes et autorise plusieurs paradigmes de programmation notamment les types impératif, fonctionnel et orienté-objet.

Les deux langages sont donc extrêmement différents, à un niveau philosophique, fondamental et pratique.

1.2.2 Nom de code Mocha

JavaScript est développé sous le nom de code Mocha. Son nom officiel étant LiveScript. Dans les deux premières versions beta de Netscape Navigator 2.0, en septembre 1995, on trouve en effet LiveScript. A cet époque Microsoft n’avait pas encore collé une connotation négative à “Live”.

Cependant les marketeux ont voulu insister sur le rôle “collaboratif” de ce langage et tenter de récupérer un peu du prestige issu du marketing de Sun autour de Java. Du coup Brendan a du renommer LiveScript en JavaScript.

Peu de temps après (1996-1997), Netscape voulut faire bénéficier à JavaScript d’un processus formel de standardisation. ISO, IETF et le jeune W3C posaient chacun des problèmes distincts dans cette standardisation. L’ECMA (European Computer Manufacturers Association), un organisme de standardisation européen, en récupéra le bébé. Ainsi est sortie la première édition du standard ECMA-262 Ed. 1 : ECMAScript est le nom de la norme officielle, JavaScript étant la plus connue des implémentations. ActionScript 3 est une autre implémentation bien connue de ECMAScript, avec des extensions.

Au fil du temps, il était clair cependant que Microsoft n’avait pas l’intention de coopérer ou de mettre en oeuvre correctement JS dans Internet Explorer même si Microsoft n’avait pas de proposition concurrente et avaient une mise en oeuvre partielle (et divergent à ce point) sur le côté serveur avec .NET.

Ainsi en 2003, le travail de JS2/original-ES4 a été mis en sommeil.

Le prochain événement a eu lieu en 2005. Il s’agit en fait de deux événements majeurs de l’histoire de JavaScript. Tout d’abord Brendan Eich et Mozilla rejoignent Ecma en tant que membres

non-lucratif. Ont commencé alors les travaux sur E4X ECMA-357 en travaillant conjointement avec Macromedia.

Ainsi, avec Macromedia (racheté par Adobe), le travail a redémarré sur ECMAScript 4, dans le but de normaliser ce qui était en AS4 et sa mise en oeuvre dans SpiderMonkey.

Hélas, en 2007, Doug Crockfort puis Yahoo ont uni leurs forces avec Microsoft pour s'opposer à ECMAScript 4, ce qui a conduit à la spécification ECMAScript 3.1 effort.

Pendant que les géant s'affrontaient, la communauté de développeurs open source s'est mise au travail pour révolutionner ce que l'on pouvait faire avec JavaScript. Cet effort collectif a été déclenché en 2005, lorsque Jesse James Garrett a publié un livre blanc dans lequel il inventa le terme Ajax. Dans ce livre blanc, il décrit un ensemble de technologies, donc l'épine dorsale est JavaScript. La technologie est utilisée pour créer des applications web où les données peuvent être chargées en arrière-plan, en évitant la nécessité de recharger la page entière et aboutissant à des applications plus dynamiques. Suite à ce livre blanc, il en a résulté une période de renaissance de l'utilisation de JavaScript dirigée par les bibliothèques open source et les communautés qui se sont formées autour d'elles. Ainsi sont nées des bibliothèques telles que Prototype, JQuery, Dojo, Motools et bien d'autres.

En juillet 2008, les parties conflictuelles se sont réunies à Oslo. Cela a conduit début 2009, à l'accord final de renommer ECMAScript 2.1 à ECMAScript 5 et à conduire le langage vers l'avant avec la future norme connue sous le nom de "Harmony".

La 3ème édition (ES3) consitue le socle de la version la plus utilisée/répandue du langage. La 4ème est morte-née, et la 5ème (ES5), désormais implémentée dans tous les navigateurs modernes, sert de socle aux applications web modernes ainsi qu'à JavaScript côté serveur (avec notamment Node.js).

Tout cela nous amène à aujourd'hui, JavaScript entre dans un cycle complètement nouveau et passionnant dans son évolution, son innovation et sa normalisation, avec de nouveaux développements tels que Node.js, permettant d'utiliser JavaScript coté serveur. L'arrivé de HTML5 dans le monde du web va également transformer l'évolution de JavaScript grâce aux HTML5 APIs, ces dernières vont permettre de controler les navigateurs coté-client, d'utiliser les web-sockets pour toujours plus de communications, obtenir des données sur des fonctionnalités tels que l'accéléromètre, la localisation GPS, et bien plus encore.

Actuellement JavaScript est disponible de base sur davantage de périphériques et de plates-formes que Java.

Même si l'explosion d'Android a fortement relancé le déploiement de Java qui s'enorgueillit de " plusieurs milliards de périphériques installés ", Java n'est pas tellement déployé sur d'autres

plates-formes mobiles, et n'est pas non plus automatiquement présent sur toutes les plates-formes desktop.

Allez trouver un seul desktop, laptop, smartphone, tablette ou liseuse qui n'ait pas une runtime JavaScript installé et qui ne s'en serve pas intensivement !

Dans certains cas, comme webOs, Firefox Mobile, JavaScript est au cœur-même de la plate-forme, constituant sa clé de voûte.

iOS, Android, Windows Phone et Blackberry ont une tendance forte à utiliser des web apps reposant très lourdement sur JavaScript. Petit à petit, les technologies collectivement appelées "HTML5" remplacent ce pourquoi on avait encore recours aux applets ou, plus souvent, à Flash. Ainsi commence la révolution JavaScript.

1.3 EcmaScript

1.3.1 Présentation

ECMAScript est un langage de programmation de type script. Il est standardisé par Ecma International dans le cadre de la spécification ECMA-262. Il s'agit d'un standard, donc les spécifications sont mises en œuvre dans différents langages comme JavaScript ou ActionScript. JavaScript évolue donc en fonction de l'avancement des standard de Ecma-International.

Comme vu précédemment JavaScript a vu le jour en décembre 1995 par Sun et Netscape.

En mars 1996, Netscape implémente le moteur JavaScript dans son navigateur web Netscape Navigator 2.0. Le succès de ce navigateur contribue à l'adoption rapide de JavaScript dans le développement web orienté client. Microsoft qui était à l'époque le seul concurrent a réagit en développant JScript, qu'il inclut ensuite dans Internet Explorer 3.0 en août 1996 pour la sortie de son navigateur.

Netscape soumet alors JavaScript à l'ECMA pour le faire standardiser. Les travaux débutent en novembre 1996, et se terminent en juin 1997 par l'adoption du nouveau standard ECMAScript.

Les spécifications sont rédigées dans le document Standard ECMA-262.

ECMAScript existe à ce jour en 5 versions du standard ECMA-262.

1.4 JSON

1.4.1 Présentation

JSON est un format d'échange de données qui est un sous ensemble de la notation objet littérale en JavaScript. Il a beaucoup gagné en popularité ces derniers temps comme une alternative légère au format XML, en particulier dans les applications AJAX.

Pourquoi ça ?

En raison de la capacité de JavaScript pour l'analyse rapide de l'information à l'aide la fonction `eval()`. JSON ne nécessite pas cependant JavaScript, et il est possible de l'utiliser comme un format d'échange simple pour n'importe quel langage de script.

Voici un exemple de ce que JSON est :

```
1 { 'details' : {  
2   'id' : 1,  
3   'type' : 'memoire',  
4   'auteur' : 'BARASCUT Jeremy',  
5   'titre' : 'JavaScript, un langage web?',  
6   'detail' : {  
7     'pages' : 120,  
8     'extra' : 22,  
9     'price' : {  
10      'eu' : 00.00,  
11      'us' : 00.00  
12    }  
13  }  
14 }}
```

Voici à titre de comparaison sont équivalent en XML

```
<details id='1' type='memoire'>  
  <auteur>BARASCUT Jeremy</auteur>  
  <title>JavaScript, un langage web?</title>  
  <detail>  
    <pages extra='22'>120</pages>  
    <price us='00.00' ca='00.0' />  
  </detail>  
</details>
```

</detail>

</details>

Certains développeurs pensent que JSON est une façon plus élégante d'écrire les données.

D'autres apprécient sa simplicité. D'autres encore soutiennent qu'il est plus léger.

Peut importe, si JSON est si populaire c'est surtout grace à la technologie AJAX.

1.5 Ajax

1.5.1 Présentation

AJAX (Asynchronous JavaScript And XML) est une architecture informatique permettant de construire des sites web dynamiques interactifs et des applications Web en se servant de différentes technologies présentées précédemment.

Ajax combine JavaScript, XML, CSS, le DOM et XMLHttpRequest afin d'améliorer la mania-
bilité et le confort d'utilisation des Applications Internet Riches (RIA).

Le terme Ajax a été introduit par Jesse James Garrett, le 18 février 2005, dans un article sur le site Web Adaptive Path. Ajax a créé une petite révolution dans les navigateurs.

En utilisant Ajax, le dialogue entre le navigateur et le serveur se déroule la plupart du temps de la manière suivante :

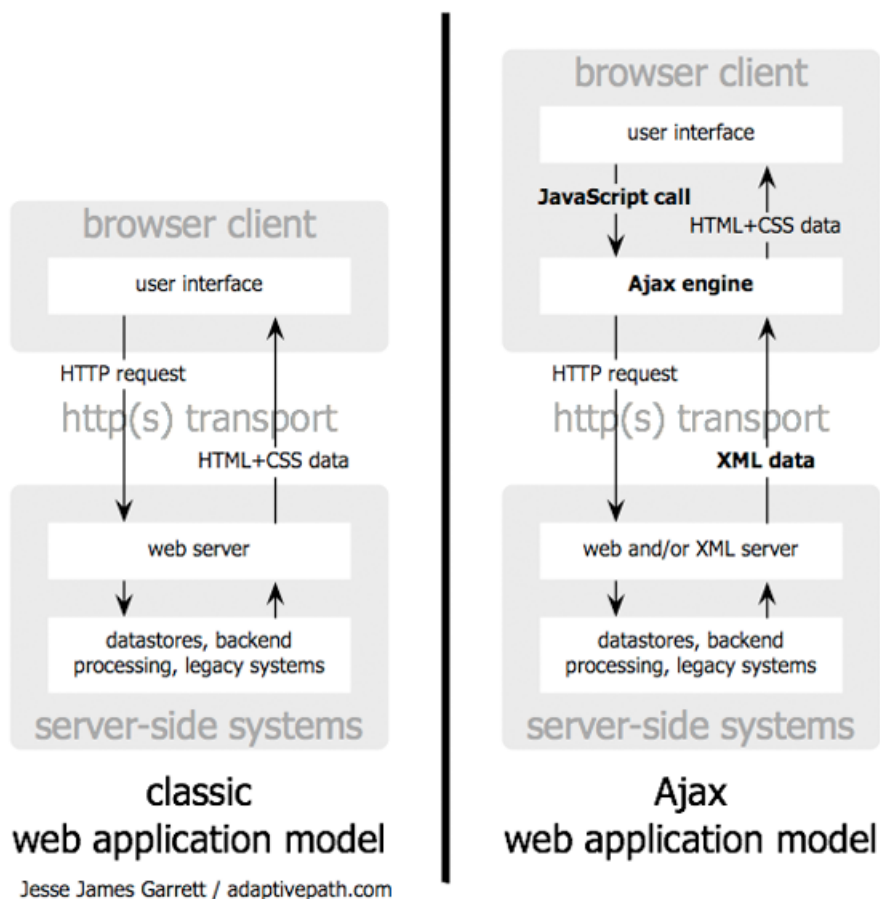


FIGURE 1.1 – Graphique technologie ajax

Les échanges de données entre le client et le serveur peuvent utiliser d'autres formats notamment le format JSON.

Ajax a permis à JavaScript de gagner en popularité et de ne plus être le vilain petit canard des langages de programmation. Ainsi avec AJAX JavaScript n'est plus un langage qui se contente d'afficher des popup intempestives ou de vérifier des formulaires.

Google a marqué les esprits avec Google Maps. Maps n'aurait jamais pu exister sans Ajax. L'utilisation de JavaScript par Google a permis à JavaScript de montrer son potentiel. Depuis JavaScript s'est désinhibé, et de véritables logiciels sont apparus dans nos navigateurs.

Google est un très gros consommateurs de JavaScript notamment avec Drive ou Gmail.

JavaScript est donc passé du petit langage d'agrément pour pages web à un langage de développement d'applications réseau supporté par tout les navigateurs quel que soit le système d'exploitation. Et le navigateur, pour un grand nombre d'utilisateurs, est la porte d'entrée de l'ordinateur et du réseau.

Avant AJAX : la page web est le support pour de petites applications JavaScript, le plus souvent d'agrément et facultatives pour utiliser le contenu.

Avec AJAX : JavaScript devient le cœur du site. Il génère du contenu HTML dont il a la maîtrise. Avec un autre langage coté serveur, JavaScript coté client s'appuie sur le moteur graphique du navigateur pour générer l'interface de l'application, plus pratique que n'importe quel "toolkit".

Cette mutation a pris du temps. Le navigateur est passé d'un logiciel pour consulter des pages web à un logiciel permettant d'exécuter d'autres logiciels (en somme comme un système d'exploitation).

Ajax est donc plébiscité parce qu'il donne naissance à des applications inédites accessibles à travers notre navigateur. Avec AJAX une application comme Google Earth pourrait aussi bien exister de façon autonome.

Malgré toutes ces avancées, JavaScript était encore confronté à des obstacles majeurs qui ont freiné son adoption.

JavaScript étant exécuté par le navigateur, ses performances de l'époque étaient extrêmement faibles. Actuellement la bataille des navigateurs se fait sur de meilleurs supports tels que HTML5/CSS3 mais aussi sur les performances des moteurs JavaScript. L'arrivée de Google Chrome et des énormes besoins de Google en performances JavaScript ont poussé tout les acteurs du marché à améliorer les moteurs pour toujours plus de performances.

Le second problème de JavaScript est l'exécution du script par le moteur. En fonction du navigateur utilisé, le code JavaScript doit être différent, chaque éditeur étant libre d'intégrer JavaScript comme il le souhaite. Ainsi un code JavaScript peut fonctionner correctement sur Firefox ou Chrome mais pas sur Internet Explorer.

Le Javascript est aussi un langage qui nécessite des efforts importants et dont le développement en AJAX pur peut être extrêmement coûteux.

Afin de résoudre ce problème de développement, John Resig a sorti un framework révolutionnant la façon d'écrire du code JavaScript. Le nom de ce framework : jQuery.

1.6 JQuery

1.6.1 Introduction

Au cours des dernières années, JavaScript a subi une transformation remarquable. A partir du moment où JavaScript a pu reléguer son image de langage jouet au second plan, il a alors pu s'affirmer comme l'un des langage de programmation les plus importants dans le monde.

Avec l'importance grandissante du développement basé sur Ajax, la montée des bibliothèques JavaScript augmentent et la stigmatisation entourant JavaScript a pratiquement disparu.

Il faut reconnaître que la bibliothèque la plus populaire et user-friendly, jQuery est en partie responsable de ce progrès.

jQuery est plus qu'un simple choix de débutant, il est en réalité utilisé par certaines des plus grandes organisations dans le monde, ajoutant de l'interactivité à des milliards de pages vues chaque mois. Google, Microsoft, Amazon, IBM, Twitter, NBC, Best Buy et Dell ne sont que quelques une des entreprises utilisant jQuery en production. Avec une très grande utilisation dans le monde, il n'est donc pas surprenant que jQuery évolue à grande vitesse.

jQuery continue de s'épanouir et les développeurs du monde entier contribuent aux corrections des bugs, des plugins et des travaux sur des projets connexes comme jQuery UI et QUnit. Ce regain d'activité assure que jQuery représente une option complète pour tout développeur cherchant à faire du développement JavaScript de catégorie mondiale.

Cela est vrai quelle que soit la philosophie ou la technique de développement utilisée : jQuery est utilisé et ce peu importe le langage coté serveur tel que : Java/Spring, PHP, .NET, Ruby on Rails, Python/Django par exemple.

jQuery est une bibliothèque JavaScript qui porte sur l'interaction entre JavaScript (comprenant AJAX) et HTML, et a pour but de simplifier les commandes communes de JavaScript. JQuery se caractérise par un ensemble de fonctions qui permettent d'offrir une alternative à la programmation JavaScript de façon uniforme sur les navigateurs les plus courants et permet par exemple de manipuler aisément le DOM, de créer des animations etc. . . mais surtout de gagner du temps dans le développement des applications : « write less, do more ».

La librairie est sous licence GPL et MIT, et donc complètement réutilisable sur des travaux professionnels. De plus la librairie à l'avantage d'être compatible avec d'autres librairies JavaScript.

- C'est une bibliothèque puissante. Le système jQuery réalise toutes sortes de tâches impressionnantes pour simplifier l'écriture du code JavaScript.
- Elle est légère. Il faut inclure une référence à votre bibliothèque dans chaque fichier qui

l'utilise. La bibliothèque jQuery fait 26 ko (dans sa version compressée), une taille inférieure à certains fichiers image. Elle n'a donc aucun impact significatif sur la vitesse de chargement.

- Elle prend en charge un mécanisme de sélection flexible. jQuery simplifie et développe le mécanisme `document.getElementById`, essentiel pour la manipulation du DOM.
- Elle dispose d'un excellent support d'animation. Vous pouvez utiliser jQuery pour afficher et masquer, déplacer et glisser des éléments.
- Elle rend les requêtes AJAX évidentes. Vous allez être surpris par la facilité d'utiliser AJAX avec jQuery.
- Elle possède un mécanisme d'évènement amélioré. JavaScript dispose d'un support très limité pour les événements. jQuery offre un outil très puissant pour ajouter un gestionnaire d'évènement à presque tous les éléments.
- Elle fournit un support multiplateforme. La bibliothèque jQuery tente de gérer des questions de compatibilité de navigateur pour vous. Ainsi, vous n'avez pas à vous soucier d'éventuels problèmes de navigateurs.
- Elle prend en charge les composants d'interface utilisateur. jQuery propose une bibliothèque d'interfaces utilisateur puissante qui compte des outils que HTML n'a pas, comme les contrôles glisser-déposer, les sliders et les calendriers.
- Elle est évolutive. jQuery possède une bibliothèque d'extension qui accepte tous types de fonctionnalités optionnelles, y compris de nouveaux composants et outils tels que l'intégration de son, les galeries d'images, les menus, etc.
- Elle introduit de nouvelles idées de programmation. jQuery est l'outil idéal pour découvrir des idées intéressantes comme la programmation fonctionnelle et les objets chaînables.
- Elle est gratuite et open-source. jQuery est disponible en licence open-source, ce qui signifie que son utilisation est gratuite et que vous pouvez la consulter et la modifier si vous le souhaitez.
- Elle est tout de même classique. Si vous décidez d'utiliser une autre bibliothèque AJAX, vous pourrez y exploiter les enseignements acquis dans jQuery.

1.6.2 Document Object Model (DOM)

Quand vous regardez un site web, vous voyez beaucoup d'éléments regroupés et assemblés pour former ce qui est en face de vous. Pour pouvoir accéder à ces éléments pour supprimer, ajouter et manipuler ces éléments, vous avez besoin d'une interface appropriée, d'une représentation des éléments dans une page structurée et qui suit un ensemble de règles sur la manière de les

modéliser. C'est ce qu'on appelle le DOM. Le DOM nous permet aussi de capturer dans le navigateur un événement comme lorsqu'un utilisateur clique sur un lien, soumet un formulaire, ou fait défiler la page.

Dans les premiers jours du web et des navigateurs, les normes en matière de mise en œuvre de JavaScript n'étaient pas efficaces. Cela a conduit les navigateurs à intégrer leur propre mise en œuvre de JavaScript créant ainsi des caractéristiques d'applications différentes et causant des problèmes aux développeurs d'applications. Ainsi pour développer une application JavaScript, il faut la coder pour chaque navigateur, ceux-ci ayant des implémentations différentes principalement entre Netscape et Internet Explorer.

Heureusement, les choses ont progressé, les navigateurs optent actuellement pour les mêmes normes et les choses se sont stabilisées. Toutefois, le niveau auquel les navigateurs supportent le DOM peut encore poser des problèmes aujourd'hui. En particulier avec les versions d'Internet Explorer 6, 7 et 8 qui ne supportent pas le DOM au même niveau que les navigateurs les plus modernes.

C'est une des raisons pour lesquelles jQuery est si précieux : tout ce qu'il offre fonctionne aussi bien dans une version antérieure d'Internet Explorer que dans la dernière version de Google Chrome ou Mozilla Firefox.

Afin d'avoir des bases solides pour la suite, je vais prendre la peine de présenter la façon dont le DOM est mis en œuvre.

Quand une page est chargée, le navigateur génère une représentation de ce qui est sur la page, et pour chaque élément il génère un ou plusieurs nœuds qui le représentent. Lorsque vous travaillez avec JavaScript, le DOM (suivant les implémentations différentes des navigateurs) peut provoquer des problèmes et vous amener à passer beaucoup de temps sur des solutions de contournement, mais l'efficacité d'un framework comme jQuery permet de régler ce souci. Lorsqu'un navigateur forme une représentation de la page en cours avec le DOM, chaque élément est un nœud. Prenons l'exemple d'un paragraphe avec du texte à l'intérieur, tels que :

```
<p>Hello World</p>
```

Ce n'est pas un, mais deux nœuds. Il y'a un nœud de type texte qui contient "Hello World" et un nœud de type élément qui est la paragraphe. Le nœud de type texte est un enfant du nœud de type élément parce qu'il réside en son sein. Dans une page type, il existe beaucoup de nœuds imbriqués. Un div avec deux paragraphes composé de texte à l'intérieur s'articule comme ceci :
div element node – paragraph element node — text node – paragraph element node — text node

Les deux paragraphes de cette instance sont frères parce qu'ils ont le même nœud parent. Les

paragraphes sont enfants de la div, mais les nœuds de type texte ne sont pas des nœuds enfants parce qu'ils ne sont pas descendants directs de l'élément div. Ils sont les nœuds enfants des nœuds de type paragraphe. Il existe trois principaux types de nœuds que vous devez connaître : élément, texte et attribut.

1.6.3 Les autres forces de jQuery

Pour finir avec jQuery j'ajouterai quelques autres détails qui permettront d'expliquer comment jQuery à permis à JavaScript de prendre son envol.

- La documentation officielle est très fournie et de grande qualité ;
- La communauté qui gravite autour de jQuery est en perpétuelle expansion et elle fournit un support de qualité ;
- De nombreux acteurs de premier plan du Web (Microsoft, Google, Amazon, Twitter, Mozilla, etc.) utilisent jQuery ;
- Une multitude de plugins est disponible afin d'augmenter les possibilités de base de jQuery.

1.7 Restfull

1.7.1 Présentation

En 2000, Roy Fielding, l'un des principaux contributeurs aux protocoles HTTP et URI, a codifié l'architecture du Web dans sa thèse de doctorat intitulée "Architectural Styles and the Design of Network-Based Software Architectures."

Dans cette thèse, il a introduit une architecture connue sous le nom "Representational State Transfer (REST)". Ce modèle, en termes abstraits, décrit les fondations du World Wide Web (WWW). Les technologies qui composent ces fondations comprennent le Hypertext Transfer Protocol (HTTP), Uniform Resource Identifier (URI), les langages de balisages tels que HTML et XML et formats adaptés au Web comme JSON.

REST est un style d'architecture pour les applications en réseau. Il se compose de plusieurs contraintes pour assurer la visibilité, la fiabilité, l'évolutivité etc.

Cela rend REST attrayant pour construire des applications client/serveur distribué et décentralisé dans l'infrastructure du Web. Le déploiement de services Web sur cette infrastructure vous permet de profiter d'un large éventail d'infrastructures existantes qui comprennent les serveurs web, les clients, les bibliothèques, les serveurs proxy, les caches, les firewalls, etc.

HTTP est un protocole de niveau applications qui définit des opérations de transfert entre les clients et les serveurs. Dans ce protocole, les méthodes telles que GET, POST, PUT et DELETE sont des opérations génériques agissant sur les ressources. Ce protocole élimine le besoin d'inventer des opérations spécifiques à l'application tels que CreateOrder, getStatus, updateStatus, etc. Les bénéfices que vous pouvez tirer de l'infrastructure HTTP dépendent de comment vous utilisez le protocole HTTP. Cependant, un certain nombre de techniques, y compris SOAP et certains frameworks web Ajax utilisent HTTP comme protocole pour transporter des messages.

1.8 MVC

1.8.1 Une structure

Le secret pour faire de grandes applications JavaScript n'est pas d'en faire une volumineuse. Il s'agit plutôt de dissocier votre application dans une série de composants relativement indépendants. Les développeurs font bien souvent l'erreur de créer des applications avec beaucoup d'interdépendances, avec d'énormes fichiers JavaScript linéaires générant une flopée de balises HTML. Ces applications sont difficiles à maintenir et à étendre, et par conséquent devraient être évitées à tout prix.

Faire un peu attention à la structure de l'application lorsque vous commencez à la construire peut faire une grande différence sur le résultat final. Ignorez toutes les idées préconçues que vous avez sur JavaScript et traitez le comme un langage orienté objet. Utilisez les classes, l'héritage, les objets et les modèles de la même façon que vous le feriez avec un autre langage, comme Python ou Ruby. L'architecture est essentielle du côté serveur alors pourquoi ne pas faire la même chose côté client ? L'architecture retenue dans ce mémoire comme dans la plupart des frameworks présentés est l'architecture MVC, une architecture permettant de maintenir et étendre vos applications. Ce modèle s'applique particulièrement bien aux applications JavaScript.

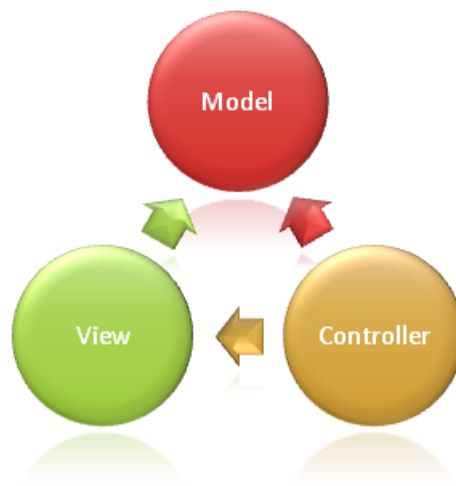
1.8.2 Qu'est ce que MVC ?

MVC est un modèle de conception qui découpe une application en trois parties : les données (Modèle), la couche de présentation (Vue), et la couche d'interaction de l'utilisation (Contrôleur).

En d'autres termes, le flux événementiel est comme ceci :

1. L'utilisateur interagit avec l'application
2. Le gestionnaire d'événements déclenche le Contrôleur
3. Le Contrôleur demande les données du modèle, qui est envoyé à la Vue.
4. La Vue présente les données à l'utilisateur.

Ou en représentation graphique



Le modèle architectural MVC peut même être mis en oeuvre sans bibliothèques ni framework. La clé est de répartir les responsabilités des composants MVC en sections de code définies, en les gardant découpés. Cela permet de rendre indépendant le développement, les tests et la maintenances de chaque composants.

1.9 Conclusion

Afin de pouvoir être lu et compris par un maximum de personne, j'ai décidé au travers de cette introduction de poser les bases d'une application d'entreprises ainsi que de tout les concepts important entourant JavaScript et de ses avancés.

Je vais utilise le modèle trois-tiers pour expliquer comment JavaScript intervient dans chaque couche pour développer l'application avec ce langage. La présentation de l'histoire permet de comprendre pourquoi JavaScript est si peu répandu ailleurs que dans le navigateur, Le fait que JavaScript est un processus de standardisation prouve que ce n'est pas un langage écrit il y'a longtemps et n'évoluant pas. Au contraire c'est un langage avec un processus d'évolution et de standardisation, à tel point que la futur version de JavaScript promet de combler certaines critique fait à son encontre. Il est également important de comprendre l'intêret du JSON comme format d'échange standard . L'architecture d'AJAX à permis à JavaScript de réssucité et de succité également énormément d'intêret. jQuery est arrivé et à combler les problème d'interopérabilité entre les navigateurs, ce qui à permis de fédérer une communauté autour du produit mais également autour de JavaScript. Le Restfull à également été une révolution dans le Web en permettant de faire des échanges entre serveurs et clients grâce à des API public en se basant uniquement sur le protocole simple et reconnu HTTP. J'ai également décidé de présenter le modèle de conception MVC afin de mieux comprendre pourquoi les frameworks inclus ce schéma par défaut dans le développement d'application Node.js ou coté client.

J'espère qu'après la lecture de mon mémoire, vous serez et comprendrez l'enthousiasme entourant actuellement JavaScript et Node.js Et en quoi sont arrivé sur le marché remet en question le fameux modèle LAMP (Linux, Apache, MySQL et PHP) et même la conception d'application traditionnelle.

2 Client

2.1 Introduction

Les années 1990 ont été marquées par les architectures client / serveur. Face à la difficulté de déploiement et de maintenance de ces clients lourds sur chaque poste client, les années 2000 ont vu l'explosion de clients dits « légers », qui déportent une grande partie du code sur des serveurs d'applications et réduisent ainsi considérablement les coûts de déploiement. Ces clients légers reposent sur les technologies HTML, JavaScript et CSS.

Avec l'évolution de la demande sur les interfaces web, les applications veulent toujours plus de fonctionnalité dans les navigateurs. Pour palier à ce problème de nombreux frameworks sont apparus avec un niveau d'abstraction plus élevé que les bibliothèques de manipulation du DOM comme jQuery et Prototype. Ces frameworks de haut niveau offrent généralement la gestion de la vue côté client, le routage, la liaison des données, etc. Ceci est utilisé, car si vous décidez d'écrire une application modérément complexe en JavaScript avec seulement jQuery, vous devrez mettre en œuvre ces fonctionnalités vous-même, ou vous retrouvez avec un désordre d'événements et de rappels.

Pendant longtemps, il n'a eu aucun gagnant clair entre les différents frameworks JavaScript de haut niveau (parfois appelé framework MV*). En 2011 Backbone.js a gagné énormément en popularité. Il a été récemment contesté par Ember.js. Les deux frameworks sont légers, mais ils sont très différents sur les goûts en matière de conception et en philosophie.

Même si Ember et Backbone attirent beaucoup l'attention actuellement, il existe d'autres frameworks avec des approches intéressantes comme Angular et CanJS. Ces frameworks font sensiblement toutes la même chose. Afin de choisir le framework le plus adapté à vos besoins, il me paraît indispensable de voir ensemble leurs différences et de les comparer sur des critères communs et indispensables dans pour un développeur.

Une belle façon de comparer les différents frameworks MV* est TodoMVC. TodoMVC présente l'implémentation d'une application de gestion de liste de tâches dans différents frameworks

MV*. Ainsi il est possible pour un même application de comparer le code source et les différences de plusieurs framework. Le nombre de framework disponible sur TodoMVC étant d'une cinquantaine, j'ai décidé d'en présenter et comparer seulement quatres (Ember, Backbone, Angular et CanJS). Ces quatre framework représente pour moi les framework qui ont un fort potentiel et vont être fortement utilisé en entreprise.

2.2 Ember

2.2.1 Présentation

Ember.js est un framework frontend MVC écrit en JavaScript qui s'exécute dans le navigateur. Il est destiné aux développeurs qui cherchent à construire des applications web avec autant de puissance et de fonctionnalités que les applications natives concurrentes.

Ember.js a été créé à partir de concepts introduits par les applications natives telles que Cocoa. Ember.js vous aide à créer une grande expérience pour l'utilisateur. Il vous aidera à organiser toutes les interactions directes qu'un utilisateur peut effectuer sur votre site/applications.

Quand vous pensez que votre code JavaScript va devenir complexe, lorsque le code devient long et compliqué, lorsque le refactoring du code s'annonce complexe alors Ember.js vous permet d'éviter de vous retrouver dans ces situations. Grâce à sa structure MVC (Modèle-vue-contrôleur) il est facile de faire des modifications ou refactoring de code de n'importe quelle partie de votre code. Il vous permettra également d'adhérer aux principes du DRY (Don't Repeat Yourself). Le modèle associé aux vues et aux contrôleurs exécute du CRUD (Create, Read, Update, Delete) pour l'informer d'un changement d'état. Il peut envoyer également une demande à la vue pour modifier la façon dont la vue représente le modèle actuel. La vue recevra alors des informations du modèle pour créer un rendu graphique.

Ember.js découpe les zones problématiques de votre interface vous permettant de vous concentrer sur une zone à la fois sans le souci d'affecter d'autres parties de votre application. Pour vous donner un exemple de certains domaines de Ember.js, jetez un oeil à la liste suivante :

- Navigation : le routeur d'Ember s'occupe de la navigation de l'application
- Mise à jour automatique des modèles : Les vues d'Ember sont automatiquement mises à jour lorsqu'il y a une modification. Ce qui signifie qu'ember met à jour automatiquement les données sous-jacentes lorsqu'il y a un changement.
- Manipulation des données : Chaque objet créé sera un objet Ember, héritant ainsi de toutes les méthodes Ember.object.
- Comportement asynchrone : Les liaisons et les propriétés utilisées avec Ember aident à gérer l'asynchrone.

Ember.js est plus qu'une bibliothèque. Ember.js prévoit de construire une bonne partie du frontend autour de méthodes et d'une architecture carrées, créant ainsi une solide architecture une fois que vous avez terminé votre application. C'est la principale différence entre Ember et un framework comme Angular.js. Angular s'autorise à être incorporé dans une application existante

alors que Ember doit être utilisé avec sa propre architecture et sa philosophie. Backbone.js serait un autre exemple de bibliothèque pouvant être facilement inséré dans des projets JavaScript existants.

Ember.js est un excellent framework pour la gestion complexe des interactions réalisées par les utilisateurs de votre application. Si vous pensez que Ember.js est un framework difficile à apprendre, c'est totalement faux. La seule difficulté pour les développeurs réside dans la compréhension des concepts que Ember.js cherche à mettre en oeuvre. Ember.js favorise convention plutôt que configuration.

2.3 Backbone.js

2.3.1 Introduction

Backbone.js est un framework JavaScript permettant de structurer une application web non pas comme une suite d'instruction jQuery, mais comme un ensemble de vues autonomes les unes des autres.

Backbone est un bon choix pour les applications dites ‘single page application’.

Single page application (Application Web Monopage) c’est à dire pour simplifier, une page principale avec un nombre important d’interactions utilisateur. Plutôt que d’avoir une navigation par page (le serveur envoie une page à chaque URL), la navigation, l’envoi de formulaire et toutes les actions classiques se gèrent en javascript.

Avant, lorsque l’on souhaitait faire ce genre de choses, on écrivait son propre JavaScript, sans aucune convention, à grand coup d’AJAX et de script JQuery. On se retrouvait vite avec un code complexe à maintenir et à tester.

Backbone.js permet de cadrer cela en définissant modèle, vue et collections afin de structurer notre code. On va pouvoir définir des événements sur des changements de valeurs de nos modèles et ainsi rafraîchir automatiquement nos vues.

Backbone.js possède son propre routeur. On peut faire correspondre des actions à des URL.

Il se veut non-contrainant par rapport à ses rivaux, ce qui lui coûte d’être présenté souvent comme moins complet. Son point fort est d’être facilement utilisable avec d’autres libraires ou frameworks.

Par contre Backbone.js est uniquement une couche client. Il ne gère pas la persistance des données sur un serveur. Heureusement cela se gère très bien en utilisant l’architecture REST. Il suffit d’indiquer à nos modèles une URL et d’utiliser une API JSON Restful pour être capable de réaliser du CRUD très facilement.

2.3.2 Dépendances

Backbone.js est basé sur la blirie Undercore.js. Cette dernière propose des fonctionnalités de manipulation d’objets, de collections et de tableaux assez poussées. Comme toute librairie populaire, elle se veut cross-browser et par héritage, Backbone.js l’est aussi.

Pour tout ce qui est manipulation DOM, plutôt que réinventer la roue, l’équipe en charge de Backbone.JS a décidé de sous traiter cette tâche à jQuery. Il est possible de remplacer jQuery,

par exemple avec Zepto, du moment qu'elle respecte l'api jQuery-compatible.

2.3.3 Composants

Backbone.js fournit des composants logiciels pouvant être utilisés librement, que ce soit avec les autres composants Backbone.js ou avec une autre librairie. Les trois plus importants sont le Model, La Vue et la Collection.

Les composants Router et Sync fournissent des fonctionnalités très intéressantes.

2.3.4 Les données

La classe Backbone.Model est utilisée pour gérer du contenu sous la forme d'un objet JavaScript, en l'encapsulant et en proposant des méthodes accesseurs.

La classe Backbone.Collection permet de manipuler des collections de Model. On retrouve les méthodes habituelles : push, pop, shift, unshift, add, remove, get, sort ou encore length. On retrouve également les fonctions provenant de Underscore.js

Ces deux classes permettent de manipuler des données. Mais Sync permet de le faire plus simplement.

Sync est le composant permettant de synchroniser les objets à travers une API du type RESTful JSON. Pour cela, il suffit de lier les objets Model et Collection à une ressource grâce à l'attribut url.

2.3.5 La vue

Toutes les manipulations DOM se font à travers le plugin jQuery qui est de loin, le mieux armé pour ces opérations.

Chaque objet Backbone.View est lié à un nœud DOM (el) et pourra le générer à nouveau, à n'importe quel moment. Le but est alors de découper le document en une multitude de vues que l'on pourra régénérer à souhait et individuellement.

2.3.6 Asynchronous Module Definition API

Il est possible d'utiliser Backbone.JS avec un AMD Loader comme RequireJS ou Curl. Ni Backbone.JS, ni Underscore ne supportent officiellement AMD. Toutefois, il est possible d'utiliser un fork implémentant de cette API. On trouve sur Github des projets templates (boilerplate) qui facilitent la mise en place d'un environnement couplant RequireJS et Backbone.

2.3.7 Plugins

Backbone.JS se veut léger : il ne fournit que des composants essentiels. Un des gros manquements de Backbone.JS par rapport à ses concurrents est l'absence de la fonctionnalité de DataBinding. Mais qu'à cela ne tienne, de nombreux plugins sont présents en libre accès sur GitHub, pour implémenter des fonctionnalités.

2.4 AngularJS

2.4.1 Introduction

AngularJs est un framework structurel pour les applications web dynamiques. Il est écrit en JavaScript et est sous Licence MIT.

C'est un framework open-source au même titre que MooTools, Prototype, Dojo ou JQuery.

AngularJs est développé par Google et sa communauté.

AngularJS permet d'utiliser le HTML comme langage de template et vous permet d'étendre la syntaxe du HTML pour exprimer les composants de votre application de façon claire et succincte.

Il a pour but de simplifier l'écriture du JavaScript en simplifiant la syntaxe et de combler les faiblesses de ce langage en lui ajoutant de nouvelles fonctionnalités. Le but d'AngularJs est de faciliter la réalisation d'applications web monopages.

AngularJs peut être utilisé avec ou sans jQuery pour la manipulation du DOM.

Le framework adapte et étend le HTML traditionnel pour servir le contenu dynamique de façon améliorée grâce à un data-binding bidirectionnel qui permet la synchronisation automatique des modèles et des vues. En conséquence AngularJS minore l'importance des manipulations DOM et améliore la testabilité du code.

AngularJs essaie d'être une solution complète pour créer une application web. Il est livré avec tout ce dont vous avez besoin pour construire une application CRUD de façon cohérente : liaison des données, directives de bases pour les templates, la validation du formulaire, le routage, le deep-linking, la réutilisation de composants et l'injection des dépendances. AngularJs permet l'écriture de tests unitaire, cas de test, mocks etc.

2.4.2 Objectifs de conception du framework

- Découper les manipulations du DOM de la logique métier. Cela améliore la testabilité du code.
- Guider les développeurs pendant toute la durée de la construction d'une application : de la conception de l'interface utilisateur, en passant par l'écriture de la logique métier, jusqu'au test de l'application.
- Considérer le test d'une application aussi important que l'écriture de l'application elle-même.

La difficulté de la phase de test est considérablement impactée par la façon dont le code est

structuré.

- Découper les côtés client et serveur d'une application. Cela permet au développement logiciel des côtés client et serveur de progresser en parallèle, et permet la réutilisation du code de chaque côtés.
- Rendre les tâches faciles évidentes et les tâches difficiles possibles.

AngularJS est un framework MVC et encourage le couplage faible entre la présentation, les données, et les composants métiers. En utilisant l'injection de dépendances, AngularJS apporte aux applications web coté client les services traditionnellement apportés coté serveur, comme les contrôleurs de vues. En conséquence, une bonne partie du fardeau supporté par le back-end est supprimé, ce qui conduit à des applications web beaucoup plus légères et maintenables.

2.5 CanJS

2.5.1 Présentation

Le framework CanJS est issu du framework Javascript MVC. Il s'agit en fait de l'extraction du noyau et il est décomposé en modules pour correspondre exactement aux besoins de l'application sans embarquer d'importants volumes de fonctionnalités inutiles. La nouvelle version de Javascript MVC se basera sur CanJS pour assurer l'inter-compatibilité des deux frameworks.

Le grand intérêt de CanJS est de fournir des objets « Model » pour stocker nos données, y associer des fonctions de mise à jour coté serveur personnalisables et des événements auxquels attacher des actions. Mieux, il gère automatiquement, grâce à son système de templates, la mise à jour de toutes les vues associées à un modèle lorsque ce dernier est modifié.

CanJS est composé de :

- can.Construct - hérite des fonctions du constructeur
- can.Observe - clé valeur contraignante
- can.Model - permet de connecter une interface JSON-REST
- can.View - chargement du modèle, mise en cache
- can.EJS - modèles contraignant direct
- can.Control - liaisons d'évènements déclaratif
- can.Route - support du bouton précédent et des onglets

Il prend également en charge un riche ensemble d'extensions et de plugins.

2.5.2 Caractéristiques vs Poids

En plus de jQuery, CanJS fait 8.5kb. A titre d'exemple voici quelques autres bibliothèques MVC (compressé avec gzip) :

- 8.97kb pour Backbone (avec Underscore.js)
- 24kb pour AngularJS
- 13kb pour Knockout
- 37kb pour Ember
- 15kb pour Batman

Pour être juste, le poids est trompeur, car chaque bibliothèque possède un ensemble différent de fonctionnalités. Cependant, CanJS fournit tout ce dont vous avez besoin pour créer une application coté client riche, avec la bibliothèque la plus légère parmi celles que nous avons comparées.

Par comparaison Backbone.js est livré le micro template Underscore.js, mais celui-ci ne se compare pas à la puissance de EJS. La plupart des applications réseau de base comprennent également un moteur de templates qui ajoute du poids à la bibliothèque.

2.5.3 Facilité d'utilisation

CanJS bénéficie d'une courbe d'apprentissage plus facile que n'importe quelle autre bibliothèque. Il bénéficie d'une bonne documentation, et il permet de se faire la main avec la page de présentation, puis de plonger plus profondément en lisant chaque méthodes et classes dans la page de documentation. CanJS permet de voir comment les applications sont construites en parcourant les exemples, en lisant les annotations et en regardant les jeux de tests. Un blog permet d'en savoir toujours plus sur CanJS, et il est possible de poser des questions sur le forum, Twitter ou d'obtenir un support premium, de formations ou de conseil.

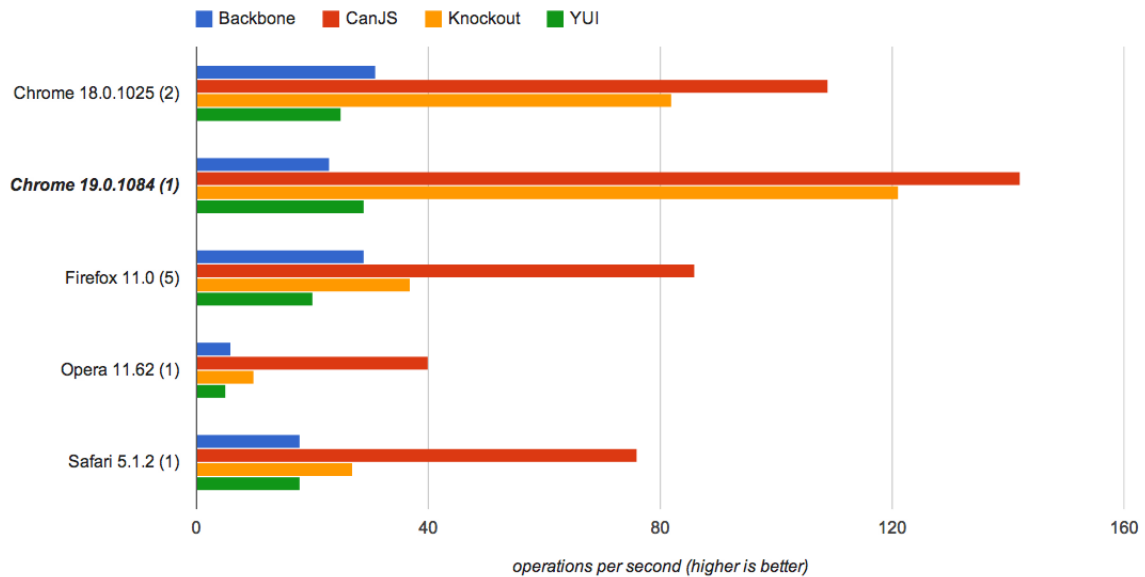
2.5.4 Prévention des fuites mémoire

CanJS empêche les fuites de mémoire qu'un développeur ne connaît probablement pas et a dans son application. Les applications JavaScript ont communément des fuites mémoire de deux sources : les gestionnaires d'événements et les objets inutilisés. Autant dire qu'il s'agit d'un problème critique pour les client MVC. CanJS gère ces fuites automatiquement ce qui rend presque impossible une fuite mémoire dans l'application.

2.5.5 Performance

CanJS est optimisé pour la performance dans des domaines clés. Par exemple la liaison en temps réel est optimisée pour les performances en modifiant directement et exactement ce qui doit être mis à jour, plutôt que le modèle en entier.

Voici un schéma mesurant les performance de CanJS en fonction d'autres frameworks.



2.5.6 Support de librairie

CanJS permet d'intégrer cinq des bibliothèques les plus couramment utilisées pour l'accès au DOM :



Cela donne la possibilité de choisir la bibliothèque préférée ou passer même facilement par une autre bibliothèque sans avoir à réécrire la couche MVC de votre application.

Il y a une intégration profonde avec chaque bibliothèque afin de permettre une utilisation complète et ce peu importe la librairie choisie.

Fonctionnalités	Angular	Backbone	CanJS	Ember
Observables	O	O	O	O
Routage	O	O	O	O
Liaison modèle-vue	O	-	O	O
Liaison modèle-vue bidirectionnelle	O	-	-	O
Vue partielle	O	-	O	O
Vue filtrée	O	-	O	O

TABLE 2.1 – Mon tableau

2.6 Conclusion

Au vue de la présentation des différents framework client, il peut être extrêmement difficile de choisir un framework MVC JavaScript pour développer notre client web. Il y’a tellement de facteurs à considérer et tellement d’options à étudier que la sélection d’un framework peut être hardue. Pour avoir une idée de toutes les alternatives possibles, il y a le site [TodoMVC](#) qui permet de comparer l’utilisation de plusieurs framework codés afin de réaliser une liste de choses à faire (TODO).

Je vous ai présenté 4 frameworks : Angular, BackBone, CanJS et Ember. J’ai donc décidé de réaliser ce mémoire autour de la comparaison de ces derniers afin de faciliter le choix du framework à utiliser. Je vais passer en revue plusieurs facteurs que vous pourriez envisager pour en choisir un.

Pour chaque facteur, j’ai attribué une note comprise entre 1 et 5. Où 1 est pauvre et 5 est riche. J’ai essayé d’être impartial dans mon mémoire, mais bien entendu mon objectivité est fortement compromise parce que mes sources sont uniquement basées sur les exemples du site [TodoMVC](#).

2.6.1 Fonctionnalités

Il y’a des éléments vraiment importants qu’un framework doit fournir pour avoir les bases nécessaires à la réalisation d’applications utiles. Fait-elle une liaison entre le modèle et la vue ? une liaison bidirectionnelle entre le modèle et la vue ? l’utilisation de filtres ? les propriétés sont elles calculées ? Y a-t-il une validation des formulaires ? etc. Cette liste peut être une très longue. Voici une comparaison de ce que je considère comme des caractéristiques vraiment importantes dans un framework MVC.

Observables : Objets qui peuvent être observés lors des changements.

Routage : Ecoute les changements d'URL du navigateur et permet d'agir en conséquence.

Liaison modèle-vue : Utilisation d'objets observables dans les vues, les vues ayant un rafraîchissement automatique lorsque le changement de l'objet est observable.

Liaison bidirectionnelle entre le modèle et la vue : Permet à la vue de pousser la modification de l'objet observation automatiquement, par exemple, un formulaire de saisie.

Vue partielle : Vues qui comprennent d'autres vues

Vue filtrée : Avec des vues qui affichent des objets filtrés par un certain nombre de critères.

Résultat

Sur la base de ces caractéristiques les scores sont les suivants :

Angular	Backbone	CanJS	Ember
5	2	4	5

Il est important de noter que Backbone peut faire la plupart de ces choses avec beaucoup d'écriture de code manuel ou à l'aide de plug-in. Mais je n'ai considéré que les fonctions disponibles de base dans les framework.

2.6.2 Flexibilité

Il existe des centaines de plugin et bibliothèques qui ont actions bien spécifiques. Ils le font habituellement mieux que le framework d'origine. Il est donc important d'être en mesure de pouvoir intégrer ces bibliothèques dans le framework MVC choisi.

Backbone est le framework le plus souple, car il est celui qui a le moins de conventions et de normes. Vous êtes obligés de prendre beaucoup de décisions lors du développement.

CanJS est presque aussi souple que Backbone car il permet d'intégrer facilement d'autres bibliothèques avec un minimum d'effort. Avec CanJS il est possible également d'utiliser un moteur de rendu totalement différent. Par exemple l'utilisation du moteur de rendu "Rivets" ne pose aucun problème. Mais il est vivement recommandé d'utiliser le moteur de rendu du framework originel.

Ember et **Angular** sont aussi des framework souples dans une certaine mesure, mais il se peut que l'utilisation d'autres bibliothèques soit difficile à mettre en place en complément du framework originel.

Résultat

Angular	Backbone	CanJS	Ember
3	5	4	3

2.6.3 La documentation et la courbe d'apprentissage

Angular

Angular est un framework qui lorsqu'on le découvre procure un effet “wow”. Il peut faire des choses étonnantes, comme les liaisons bidirectionnelles, sans avoir à apprendre beaucoup de choses. Il semble assez facile à prendre en main d'un premier abord. Mais une fois les bases acquises, la courbe d'apprentissage se met alors à stagner. Il s'agit d'un framework complexe avec beaucoup de particularités. La lecture de la documentation n'est pas facile car il y'a beaucoup de jargon spécifique à Angular et un manque important d'exemple.

Backbone

Les bases de Backbone sont elles aussi assez faciles à assimiler. Mais encore une fois, une fois dépassé le stade des bases, on s'aperçoit que la documentation ne donne pas assez de conseils sur la façon de structurer son code. Il devient alors obligatoire de regarder ou lire des tutoriels pour apprendre certaines des meilleures pratiques de backbone. Il se peut également que l'apprentissage d'une autre bibliothèque soit obligatoire pour faire avancer votre développement (par exemple Marionette ou Thorax). Au final, l'apprentissage de Backbone n'est pas plus facile.

CanJS

CanJS en comparaison est le plus facile à apprendre. Juste en lisant le site internet du framework (<http://canjs.us>), l'essentiel de ce que l'on a besoin de savoir est présenté pour être productif. Il y'a bien sûr encore beaucoup à apprendre, mais la nécessité de recourir à l'aide survient à de rares occasions (tutoriels, forum, irc).

Ember

Ember a aussi une courbe d'apprentissage abrupte comme Angular, mais l'apprentissage de Ember est un peu plus facile que Angular. Il nécessite par contre un investissement important au début pour acquérir les bases. Angular en revanche permet de faire des choses incroyables sans en avoir trop à apprendre. Ember manque de cet effet “wow”.

Résultat

Angular	Backbone	CanJS	Ember
2	4	5	3

2.6.4 La productivité des développeurs

Après avoir appris correctement le framework ce qui importe le plus est de savoir comment le framework permet d'avoir une productivité performante en mixant habilement conventions, magie et conception la plus rapide possible.

Angular

Une fois que l'on maîtrise Angular correctement, on devient très productif. Il n'y a aucun doute à ce sujet. Il n'a cependant pas le score le plus élevé parce que Ember franchi une étape supplémentaire dans cette catégorie.

Backbone

Backbone oblige à écrire beaucoup de code réutilisable, ce qui est pour certains projet totalement inutile. Certains développeurs pensent qu'il s'agit davantage d'une contrainte importante allant à l'encontre de la productivité des développeurs.

CanJS

CanJS ne brille pas mais ne déçoit pas non plus dans ce domaine. En raison de la faible courbe d'apprentissage vous pouvez devenir très productif très tôt avec ce framework.

Ember

Ember brille particulièrement ici. Parce qu'il est plein de conventions, il fait beaucoup de choses à votre place. Tout ce qu'il reste à faire c'est d'apprendre et d'appliquer ces conventions et Ember s'occupe du reste.

Résultat

Angular	Backbone	CanJS	Ember
4	2	4	5

2.6.5 Communauté

Est-il facile de trouver de l'aide, des tutoriels et des experts ?

La communauté **Backbone** est énorme, c'est indubitable. On trouve des dizaines de tutoriaux sur Backbone et une communauté très active sur StackOverflow et IRC.

Les communautés **Angular** et **Ember** sont également assez grandes. Il y a également de nombreux tutoriel ainsi qu'une forte activité sur StackOverflow et IRC, mais pas autant que sur Backbone.

La communauté **CanJS** est assez faible en comparaison, mais heureusement, très active et utile. La taille de la communauté n'est par conséquent pas un frein important.

Résultat

Angular	Backbone	CanJS	Ember
4	5	3	4

2.6.6 Ecosystème

Y'a t'il un écosystème de plugin et de bibliothèques ?

Là encore **Backbone** surpasse les autres. Il y a un volume très important de plugins disponible.

L'écosystème d'**Angular** devient assez intéressant avec entre autre l'interface utilisateur. L'é-

cosystème d'**Ember** est moins développé mais il devrait s'améliorer en raison de sa popularité importante. **CanJS** en comparaison dispose du plus petit écosystème.

Résultat

Angular	Backbone	CanJS	Ember
4	5	2	4

2.6.7 Poids

Cela peut être un facteur important, surtout pour du développement mobile.

Poids des frameworks seul (sans aucune dépendances)

Angular	Backbone	CanJS	Ember
80ko	61ko	57ko	269ko

Backbone est le plus léger et les développeurs soulignent souvent ce fait. Mais ce n'est pas le cas si l'on compte les dépendances.

Poids des frameworks avec dépendances

Angular est le seul framework du groupe qui ne nécessite pas de bibliothèques supplémentaires pour travailler.

Tout les autres ont besoins de dépendances pour fonctionner.

Backbone a au moins besoin de Underscore et Zepto. Il est possible d'utiliser les mini-templates de Underscore pour le rendu des vues, mais la plupart du temps, l'utilisation d'un véritable moteur de template est appréciable comme Moustache par exemple. Au final le poids est de 61ko.

CanJS a besoin d'au moins Zepto. 57ko.

Ember a besoin de jQuery et de Handlebars. 269ko.

Résultat

Angular	Backbone	CanJS	Ember
4	5	5	2

2.6.8 Performance

La performance n'est pas forcément un facteur déterminant dans le choix d'un framework car ils sont tous assez performant pour la plupart des cas d'utilisation. Mais cela dépend bien sur de l'utilisation du framework. S'il est utilisé pour la création d'un jeu alors les performances deviennent importantes.

Il y a sur internet de nombreux tests de performances avec ces frameworks. Par contre la fiabilité de ces tests n'est pas une valeur sure car il est impossible de vérifier si ces tests ont été réalisés de dans bonnes conditions et avec un code de qualité.

Cependant, au vu des tests, il semblerait que **CanJS** ait un avantage lorsqu'il s'agit de performances, en particulier dans l'affichage de la vue. De plus, **Angular** est le moins performant.

Résultat

Angular	Backbone	CanJS	Ember
3	4	5	4

2.6.9 Maturité

Est-ce un framework mature, est-ce prouvé en production, est-ce que de nombreuses applications web l'utilise ?

Backbone a de nombreux sites construit avec lui. Sa base de code n'a pas eu de grands changements dans les deux dernières années ce qui est une bonne chose du point de vue de la maturité.

Bien que **Ember** ne soit pas nouveau, il a subi de profonds changements lors de sa progression, transformant tout pour atteindre une forme stable dans les trois derniers mois. Par conséquent il ne s'agit pas d'un framework mature.

Angular semble plus stable et éprouvé que Ember mais moins que Backbone.

CanJS est une extraction de JavaScriptMVC, une bibliothèque qui a été lancé en 2008 et possède un important retour d'expériences et d'application construites.

Résultat

Angular	Backbone	CanJS	Ember
4	5	4	3

2.6.10 Sécurité des fuites mémoire

Il s'agit d'une considération importante si l'on construit des applications en single page, qui sont destinées à rester ouvert pendant de longues périodes. Si l'on ne veut pas que l'application soit une fuite mémoire, cela peut devenir un réel problème. Malheureusement les fuites peuvent se faire assez facilement, surtout si l'on crée soit même des écoutes pour les évènements DOM.

Angular, **CanJS** et **Ember** traiteront de façon efficace et aussi longtemps les fuites mémoire du moment que l'on suit les meilleures pratiques. **Backbone** en revanche oblige à faire ce travail manuellement dans une méthode de désallocation mémoire.

Résultat

Angular	Backbone	CanJS	Ember
5	3	5	5

2.6.11 Testabilité

Est-il facile de tester le code ?

Les clés pour avoir un grand code testable sont la modularité (avoir de petits morceaux qui peuvent être testés séparément) et l'injection de dépendances (être capable de changer les dépendances dans les tests).

Il est possible de le faire avec la plupart des frameworks du moment que l'on utilise les bons modèles, mais ce n'est pas simple et cela oblige de changer ses habitudes d'applications.

La modularité et l'injection de dépendances sont des caractéristiques fortes de **Angular**, en décourageant activement de faire les choses d'une autre manière que Angular l'a prévue. Cela conduit généralement à un code plus facile à tester. Pour cette raison, Angular dispose d'un avantage dans ce domaine.

Résultat

Angular	Backbone	CanJS	Ember
5	4	4	4

2.6.12 Goût personnel

C'est probablement l'un des facteurs les plus importants lors du choix d'un framework. Mais il n'est pas possible de noter ce sujet et cela reste à l'entière appréciation du développeur.

2.6.13 Total

En rassemblant les résultats on arrive à un palmarès. Ceci n'est qu'une opinion forgée à l'aide de TodoMVC ainsi qu'à la lecture de différents sites internet. Il ne représente pas un résultat fiable et sert simplement à dégager un point de vue.

Total général	Angular	Backbone	CanJS	Ember
Fonctionnalités	5	2	4	4
Flexibilité	3	5	4	3
La documentation et la courbe d'apprentissage	2	4	5	3
La productivité des développeurs	4	2	4	5
Communauté	4	5	3	4
Écosystème	4	5	2	4
Poid	4	5	5	2
Performance	3	4	5	4
Maturité	4	5	4	3
Sécurité des fuites mémoire	5	3	5	5
Testabilité	5	4	4	4
Total	43	44	45	42

Si le développeur accorde autant de poid à chaque facteur, il s'agit d'un concours serré, il n'y a ni gagnants ni perdants. Donc je suppose que tout revient au goût personnel du développeur ou à l'affectation de poids différents pour chaque catégories.

3 Serveur

3.1 Introduction

Il y a encore quelques années, l'idée de faire du développement en Javascript faisait hérisser les cheveux de la plupart des programmeurs. Les développeurs se seraient volontiers passé de JavaScript si une alternative viable existait. Grâce à l'évolution technologique, l'explosion et la démocratisation des RIA (Rich Internet Applications) et la révolution du marché des navigateurs avec Chrome et Firefox, le JavaScript est devenu un langage de première classe dans le paysage web.

A la base JavaScript est également un langage prévue pour tourner coté serveur. Alors pourquoi ne pas développer un serveur d'application en JavaScript ?

Car malgré ses défauts, JavaScript est un langage possédant une expressivité impressionnante, **orienté objet**, doté de principes hérités des langages fonctionnels (Lisp, etc.) comme la **closure** ou les **lambda-functions**, et intégrant le paradigme de **programmation événementielle**... Bien loin, d'un langage pour faire de l'accès DOM.

Ce chapitre a pour but de présenter le développement coté serveur avec Node.js, une technologie basée sur V8, le moteur JavaScript de Google Chrome, dont elle étend les capacités en ajoutant de nombreuses fonctionnalités d'I/O (Entrées/Sorties).

Node est tellement important et performant qu'un mémoire/livre entier n'arriverait pas à couvrir son ensemble.

Ce chapitre va présenter **Node.js**, son gestionnaire de paquet **NPM** ainsi qu'un framework très prometteur **Meteor** permettant de simplifier le développement d'application serveur avec Node.

3.2 Node.js

3.2.1 Présentation

Node.js est une plateforme open source pour le développement d'application coté serveur.

Il a été créé par Ryan Dahi en février 2009 et est depuis devenu très populaire. L'esprit de Node est similaire à Twisted pour Python et Eventmachine pour Ruby.

Il se compose délibérément d'une bibliothèque de base minimaliste accompagné d'un écosystème riche. Il fonctionne au dessus du moteur V8. V8 est un moteur JavaScript très rapide développé par les ingénieurs de chez Google.

Le moteur JavaScript V8 de Google utilisé dans Chrome fait partie des moteurs JavaScript les plus puissants du marché.

JavaScript est populaire coté client, mais depuis peu il est un langage plébiscité pour le développement coté serveur.

Node permet de développer très simplement des applications scalables. : l'idée est d'utiliser des IO non bloquantes pour gérer toutes les requêtes entrantes, sortantes ainsi que tout le processus lié à la requête.

La page d'accueil de Node.js le décrit ainsi :

“Node.js est une plateforme construire sur le moteur JavaScript de Chrome pour construire facilement des applications réseau rapides et évolutives. Node.js utilise une gestion d'événement basé sur des I/O non bloquantes, qui le rend léger et efficace, idéal pour les applications de données intensives en temps réels qui s'exécutent à travers des dispositifs distribués.
<http://nodejs.org>

Cela peut sembler énigmatique aux nouveaux arrivants, mais il résume succinctement certaines principales forces de Node.js et mérite d'être explorée plus en détail. Les gens sont souvent surpris quand ils entendent que JavaScript est un vrai langage de programmation complet et entier. Pour la communauté des développeurs, JavaScript n'est pas un vrai langage comme C, C++ ou Java.

Depuis ses humbles débuts, JavaScript a évolué et est désormais pris en charge dans tout les principaux navigateurs, y compris ceux sur les appareils mobiles. non seulement il est un langage populaire, mais les outils et frameworks actuellement disponibles sont nombreux et efficace, il est reconnu comme un langage puissant. JavaScript en tant que plateforme de développement coté serveur prend en charge l'intégration continue, le déploiement continu, les connexions aux bases de données relations, les architectures orientés services et à peu près toutes les techniques

disponibles.

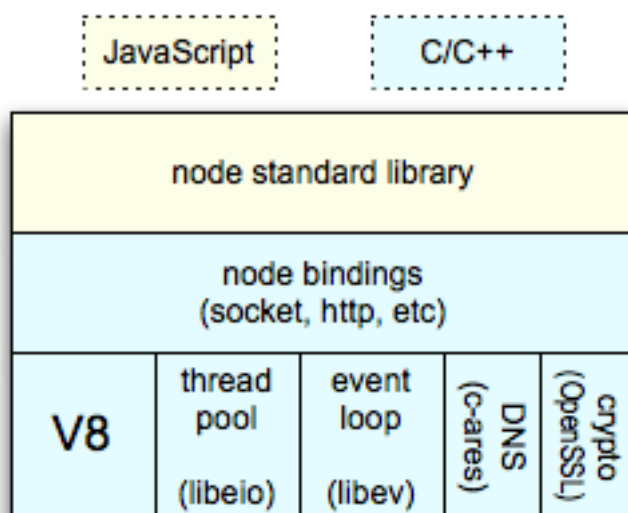
En conjonction avec le moteur JavaScript V8 de Google, il est maintenant extrêmement rapide. En fait il est plusieurs fois plus rapide que les autres langages de script comme Ruby et Python. Contre Python 3, le moteur JavaScript V8 est environ 13 fois plus rapide avec un code similaire. Node.js est plus rapide d'environ 8 fois que Ruby 1.9. Ce sont des points de repère incroyables pour un langage dynamique, et est dû en grande partie à des optimisations du moteur v8.

La description officielle des discussions sur le modèle événementiel à I/O non bloquant est la suivante : Traditionnellement, la programmation se fait de manière synchrone : une ligne de commande est exécutée, le système attend le résultat, le résultat est traité, puis l'exécution reprend.

Parfois, attendre le résultat peut reprendre après un certain temps, par exemple, la lecture d'une base de données ou d'écriture sur un réseau.

Dans des langages tels que Java et C#, une solution consiste à lancer un nouveau thread. Un thread peut être considéré comme un processus léger qui exécute des tâches. Programmer en utilisant des threads peut être difficile parce que plusieurs threads peuvent essayer d'accéder simultanément à la même ressource. Sans disséquer les subtilités de la programmation multi-threading, vous pouvez imaginer que ce serait désastreux pour un thread à incrémenter alors qu'un autre thread décrémente le compteur en même temps.

3.2.2 Architecture de Node.js



Node.js est en grande partie écrit en C/C++. Mais il possède un framework complet en JavaScript pour développer nos applications. C'est ce qui le rend particulièrement intéressant.

3.2.3 Boucle d'évènement

Node est écrit en C++ et utilisant la bibliothèque libev de Marc Lehman, la boucle d'évènement utilise epoll ou kqueue comme mécanisme d'évènement évolutifs.

3.2.4 I/O non bloquant

Node évite de perdre en temps de calcul CPU habituellement rencontré par l'attente d'une réponse d'entrée ou de sortie (base de données, système de fichier, service web, ...) grâce à l'asynchrone complet fourni par la bibliothèque libeio de Marc Lehmann

Ces caractéristiques permettent à Node de gérer une grande quantité de trafic en manipulant le plus rapidement une demande afin de libérer le thread pour la demande suivante. Node.js est un serveur capable de répondre au problème C10K.

Node possède un support intégré de la plupart des protocoles importants comme TCP, DNS et HTTP (celui qui nous intéresse ici). L'objectif de la conception d'une application Node est que toute fonction effectuant une E / S doit utiliser un callback. C'est pourquoi il n'existe pas de méthode de blocage prévu dans l'API de Node.

3.2.5 Asynchrone

Prenons l'exemple du serveur Apache. Chaque requête HTTP entrante se voit allouer un thread. Celui-ci est utilisé pendant toute la durée du traitement de la requête : lecture/écriture en base de données, lecture/écriture dans des fichiers de logs ou autres, exécution de code... Ainsi, si votre code côté serveur fait un accès à la base de données, celui-ci est en attente de la réponse. De plus, durant toute la durée de la requête, le client sera bloqué, en attente de la réponse du serveur.

Node et plus globalement les serveurs dits non bloquants (comme Netty ou Deft pour ceux qui tournent sur JVM) adoptent une autre approche. Node n'utilise qu'un seul thread pour gérer les requêtes entrantes. De plus, Node ne propose pas, dans ses API, de fonctions bloquantes. Ainsi, tout notre code est asynchrone. L'avantage immédiat à cela est la simplification à l'extrême de la gestion de la concurrence dans nos applications qui n'est ici plus gérée par le développeur mais directement par l'OS.

Concrètement, toutes les fonctions fournies par Node prennent en paramètre un callback. Une fois que la fonction termine son traitement, le callback sera appelé avec le résultat en paramètre et une éventuelle erreur. Ainsi, pendant toute la durée du traitement, le thread est relâché et

peut être donné à une autre requête pour effectuer un autre traitement. Nous sommes donc face à un système événementiel. Et Javascript est un très bon choix de langage du fait que l'on peut programmer de manière fonctionnelle avec celui-ci. Les callbacks dans tous les sens sont bien connu avec l'utilisation de bibliothèques comme jQuery qui nous habituent depuis déjà quelques temps à ce type de développement.

3.2.6 Performance

Node.js est rapide. L'une des critiques les plus fréquentes à propos des langages interprétés comme PHP, Python et Ruby est la performance. Jason Hoffman, directeur de la technologie de Joyent, a discuté de la façon dont Node.js est ultra performant et peut être ralenti à cause des systèmes d'exploitation. Un seul core avec moins de 1 Go de RAM est capable de gérer 10 Go de trafic et un million de terminaux connectés. La combinaison de 24 d'entre eux sur une seule machine produit un niveau global de débit qui dépasse les capacités des systèmes d'exploitation et Piles TCP / IP. En d'autres termes, avec une application bien conçue, ce n'est pas Node.js le goulot d'étranglement c'est votre système d'exploitation.

3.2.7 Communauté

Malgré sa jeunesse, la communauté Node grandit de plus en plus rapidement et le dépôt de Node est le 3ème dépôt le plus suivi sur Github.

3.2.8 Forces

- Bien que Node.js date seulement de 2009 :
 - il a déjà un bon système de package : Node Package Manager que la grande majorité de la communauté semble utiliser
 - il a déjà une très forte communauté, de très nombreuses bibliothèques, exemple : plein de bibliothèques classées, déjà plus de 9000 packages sur npm
 - de plus en plus de sociétés semblent l'utiliser comme LinkedIn par exemple.
- tous les développeurs web (front-end) connaissent le langage, il est très populaire. Par conséquent, pour passer à Node.js il n'y a pas le frein d'apprendre un nouveau langage comme ça serait le cas avec l'utilisation d'un framework web basé sur Ruby ou Python. Je pense que dans les mois qui viennent, de nombreux développeurs PHP vont passer à

Node.js... surtout ceux qui regardaient ailleurs mais qui ne souhaitent pas apprendre un nouveau langage.

- la possibilité de partager du code (bibliothèques communes...) entre la partie client et serveur peut être intéressant. La question se pose de plus en plus étant donné qu'on est de plus en plus amené à effectuer beaucoup de traitement côté client (exemple avec Backbone.js).
- Ces derniers jours, la visualisation de screencast sur le framework JavaScript Meteor m'a profondément intéressé.
- Autre exemple, il est possible d'utiliser la même API Canvas côté client et côté serveur (avec node-canvas).
- Node.js semble être très rapide, ça tient très bien la montée en charge.
- La vitesse de l'interpréteur Javascript est en constante progression V8.

3.2.9 Retour d'expérience LinkedIn

3.3 NPM

3.3.1 Présentation

Node Package Manager (NPM) est un gestionnaire de modules pour Node. C'est en quelque sorte un équivalent pour Node de aptitude ou apt-get (les gestionnaires de paquets sous Debian et Ubuntu) avec une interface très similaire.

NPM permet d'ajouter de nombreuses fonctionnalités à Node (exemple : gestion des sockets avec socket.io, utilisation d'un framework avec Express JS, utilisation de template avec Jade. . .) en utilisant une commande `npm` qui s'occupera de télécharger et installer les modules demandés. Par exemple pour télécharger la dernière version d'Express, il suffit d'exécuter la commande *`npm install express`* et de laisser l'utilitaire faire son travail. Si quelques mois plus tard, le désire d'obtenir la dernière version d'express se fait sentir, on exécutera simplement *`npm update express`*.

Cet outil permet également d'installer directement un exécutable du module grâce à l'option `-g` (très pratique pour Coffee-Script par exemple).

Actuellement le nombre de modules Node.js disponible est supérieur à 30 000 modules.

NPM, ainsi que des informations concernant son installation et son utilisation sont disponibles sur github, comme la grande majorité de l'écosystème node.

3.4 Socket.IO

3.4.1 Présentation

Socket.IO est un module Node puissant qui apporte la possibilité de gérer simplement les connexions entre le serveur et le client.

D'après les dire du créateur de Socket.IO, "l'objectif est de fournir pour les applications basées sur les navigateurs, un mécanisme de communication bidirectionnelle entre des serveurs et/ou des clients". **Websocket** est un protocole bidirectionnel moderne qui permet une communication interactive entre le navigateur et le serveur. Son principal inconvénient est que la mise en œuvre est généralement disponible uniquement sur les navigateurs les plus récents. Cependant en utilisant Socket.IO, ce détail de bas niveau n'est pas un problème pour le développeur qui est soulagé par Socket.IO de la nécessité d'écrire du code spécifique en fonction du navigateur. Le côté client utilisera automatiquement et de manière transparente le meilleur type de communication (basé sur les fonctionnalités du navigateur) pour se connecter à un serveur Node. Prenons un exemple Si l'architecture de l'application utilise un client Google Chrome, le navigateur est capable d'utiliser le protocole WebSocket et par conséquent Node choisira ce type de connexion.

Mais que se passerait il si l'on décide de créer une connexion avec le navigateur de Mozilla Firefox où le support des WebSocket est désactiver ? Socket.IO choisira d'utiliser alors le mode de connexion en "long pooling" automatiquement et ce sans besoin de changer quoi que ce soit dans le code du client et du serveur.

3.5 Meteor.js

3.5.1 Présentation

Bien que peu connu, Meteor est un environnement très simple qui permet de développer des applications modernes en quelques heures, alors que ça prendrait plusieurs semaines avec de très bons outils. En effet, les applications développées avec Meteor s'exécutent directement dans le navigateur plutôt que sur un serveur web et récupèrent toutes les données nécessaires dans le cloud. Meteor est un framework open source qui mise beaucoup sur le JavaScript. Avec quelques lignes de JavaScript et quelques commandes, il est possible de développer une application Web relativement simple qui s'actualise en temps réel. Un tel modèle est déjà utilisé par des grandes entreprises telles que Twitter, Facebook et Google mais était jusqu'alors hors de portée pour un développeur moyen. Avec cette levée de fonds conséquente (11,2 millions de dollars), le projet espère devenir plus accessible aussi bien aux débutants qu'aux experts et devenir, pourquoi pas, un standard.

En attendant, Meteor est toujours en cours de développement mais se développe rapidement et des changements majeurs sont au rendez-vous dans l'API à chaque nouvelle version. Ce n'est pas encore une version stable, il faudra pour cela patienter sans doute plusieurs mois. Mais je trouve intéressant de le présenter dans ce mémoire.

3.5.2 Principes

Meteor permet de développer avec le même langage (en Javascript ou dans un langage compilant vers Javascript comme CoffeeScript ou Dart) et avec la même API sur le client et sur le serveur. Ce choix d'architecture permet de déplacer facilement un traitement du serveur vers le client (et réciproquement) voire de le dupliquer par exemple dans le cas de la validation d'un formulaire. Dans cette logique, Meteor inclut un système de gestion de base de données côté client, fonctionnalité originale du framework. Il est ainsi possible d'effectuer des requêtes même en étant déconnecté du serveur. Cela permet notamment à Meteor d'inclure par défaut, des mécanismes de compensation de latence. Par exemple l'envoi d'un message dans un chat sera instantanément ajouté au fil des messages au clic sur le bouton "Envoyer", tandis que la vérification du message se fera en arrière plan côté serveur. Ce mécanisme permet l'utilisation de la programmation réactive coté client.

3.5.3 Live HTML

Si des modifications sont apportées au code HTML, CSS, JavaScript, ou à des données sur le serveur, chaque client montre les changements automatiquement sans actualisation du navigateur. Par exemple, si le développeur change la couleur de fond d'une page en jaune alors chaque navigateur affichera la nouvelle couleur de fond jaune sans aucun rafraîchissement. Ou, si un utilisateur ajoute un nouveau film à une collection de films, alors tous les navigateurs afficheront le nouveau film automatiquement. Avec Live HTML, les utilisateurs n'ont plus besoin d'un bouton d'actualisation. Les modifications apportées à une application se produisent partout automatiquement sans aucun effort. Le framework Meteor gère tous les détails nativement afin de garder tous les clients synchroniser avec le serveur simplement et sans aucun développement spécifique.

3.5.4 Compensation de la latence

Lorsque que quelqu'un réalise une modification des données sur le client, ces modifications apparaissent comme si elles se sont produites sur le serveur, sans aucun retard. Par exemple, si un utilisateur crée un nouveau film, le film apparaît instantanément. Cependant, ce n'est qu'une illusion. Dans le fond, les mises à jour de meteor actualise la base de données avec le nouveau film. Si, pour une raison quelconque, le film ne peut pas être ajouté à la base de données, Meteor enlève le film dans le client automatiquement. La compensation de la latence est extrêmement importante pour la création d'une application Web sensible. Lorsque l'on veut que l'utilisateur soit en mesure d'apporter des modifications instantanées dans le navigateur, le framework va gérer les détails de la mise à jour de la base de données sans ralentir l'utilisateur.

3.5.5 Temps réel

Plutôt que d'envoyer du html à votre client, Meteor propose d'envoyer seulement les données et de laisser le client décider de la façon dont elles doivent être affichées. Là où Meteor est intéressant, c'est pour sa gestion native du temps réel. L'objet affiché dans votre navigateur a-t-il été modifié par un autre utilisateur ? Pas de problème, Meteor va informer tous les clients de cette modification, et affiche la modification en conséquence. Meteor a fait le choix d'utiliser SockJS. À la différence de socket.io, SockJS ne passe pas par les websockets, mais par les requêtes xhr par souci de compatibilité avec tous les navigateurs. Tout se passe par un système de push/subscribe que l'on retrouve de plus en plus. L'intérêt du push/subscribe est que les

clients s'abonnent à des événements qui les intéressent et réagissent en conséquence.

3.5.6 Rafraichissement automatique

Lorsqu'un développeur est en train de développer, la sauvegarde d'un fichier sur son ordinateur va automatiquement rafraîchir le navigateur avec ses modifications. Ce principe de modification à chaud est aussi disponible en production : Meteor annonce qu'une nouvelle version d'une application peut être déployée de façon transparente à ses utilisateurs.

3.6 Conclusion

En peu de temps, c'est à dire 2009, Node.js a suscité l'intérêt de milliers de développeurs expérimentés et possède actuellement un gestionnaire de paquet avec de nombreux modules intéressant. Node.js est également une raison de la création de nombreuse startup à travers le monde.

J'ai essayé au travers de ce chapitre de vous présenter la petite révolution de Node.js dans le développement d'application 3-tiers et de son intérêt.

Node.js a beaucoup de points forts qui peuvent faire de lui une option sérieuse pour le développement d'applications coté serveur. Comme nous l'avons déjà vu, il tourne avec V8, le moteur javascript de chez google, ce qui lui confère de très bonnes performances. Ensuite, il utilise une gestion événementielle des I/O, non-bloquante. EventMachine, Twisted ou encore Tornado font de même en Ruby ou Python, mais Node.js pousse le concept plus loin (les accès aux fichiers sont asynchrones, par exemple), et la syntaxe javascript s'y prête particulièrement bien. C'est vraiment un élément clé pour la scalabilité. Enfin, Node.js utilise maintenant les modules de CommonJS, favorisant ainsi la réutilisation avec les autres frameworks JS.

Bref, Node.js a un gros potentiel, et la prolifération des frameworks (Picard, fab, Express, Django, Vroom, Node-router, Bomber...) et bibliothèques (tpl-node, node-restclient, restclient, node-crypto, node-compress, cookie-node...) laisse penser que Node.js a un brillant avenir devant lui.

3.6.1 Wordsquared.com

Ce dernier point s'adressera particulièrement aux sceptiques, pour lesquels node n'est encore qu'un gadget. Le site wordsquared.com est l'un des quelques sites basés sur node aujourd'hui. Wordsquared est particulier, car il s'agit d'une démonstration technologique et technique d'une rare qualité : il s'agit en fait d'un jeu de scrabble géant, multijoueur, en temps réel. La réalisation de l'application est impressionnante, tout comme sa réactivité. C'est sans aucun doute la meilleure preuve de maturité du serveur node.js/

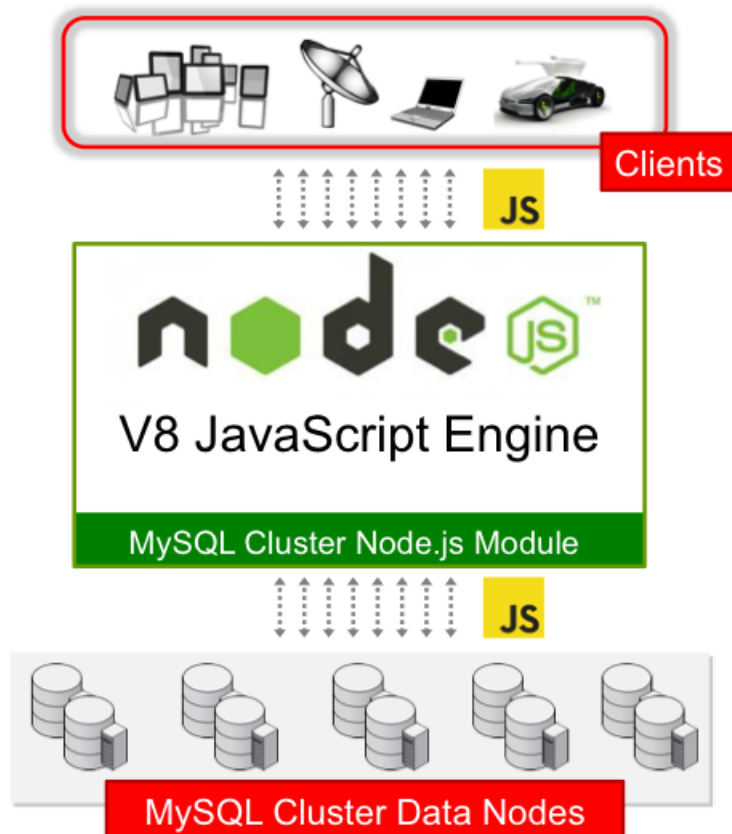
4 Base de données

4.1 Introduction

Une base de données est un conteneur permettant de stocker l'intégralité des informations d'une application sur un serveur. Il existe plusieurs sortes de base de données, mais j'ai décidé de n'en présenter que trois. Ce sont celles utilisées en majorité dans les entreprises. Il s'agit des bases de données relationnelles (MySQL, PostgreSQL, Oracle, SQLserveur...), des bases de données NoSQL (MongoDB...), et enfin récemment de Webstorage.

4.2 SGBDR

Le monde du Système de Gestion de Base de Données Relationnelles est actuellement bouleversé par l'arrivée du NoSQL et de nouveaux serveurs comme Node.js. Mais la plupart des éditeurs de base de données ont compris la nécessité de s'adapter et proposent actuellement une réponse au NoSQL et à Node.js.



Afin d'éviter la fuite des développeurs vers des serveurs NoSQL, Oracle par exemple a développé un connecteur NoSQL dans MySQL Cluster 7.3. Grâce au connecteur NoSQL de MySQL, implanté en tant que module du moteur V8 de Node.js, les utilisateurs peuvent bénéficier de fonctionnalités avancées du SGBDR avec auto-fragmentation pour les applications en plus d'un accès à NoSQL sans transformation en SQL. La nouvelle API NoSQL Node.js intégrée dans MySQL Cluster offre à Node.js une interface JavaScript asynchrone native qui peut être utilisée à la fois à la requête et à la réception des résultats établis directement à partir de MySQL Cluster, sans transformations en SQL. Cela garantit également une faible latence pour les requêtes simples. Ainsi il est possible en natif d'utiliser uniquement JavaScript pour parler avec MySQL Cluster.

Le fait d'utiliser Node.js permet grâce à sa communauté d'accéder à de nombreuses extensions pour utiliser n'importe quelle bases de données relationnelle. Certaines bases de données incluent également l'accès à la base de données par Webservices ce qui peut éviter l'utilisation d'extensions.

JavaScript est donc capable au travers de Node.js d'accéder à n'importe quelle base de données relationnelle en écrivant ou non du SQL. Cela peut avoir un intérêt pour des applications spécifiques ayant besoin de contraintes ACID ou pour éviter aux développeurs et administrateurs de changer leurs habitudes.

Node.js recommande l'utilisation de MongoDB qui est beaucoup plus adapté à sa philosophie que ne le sont les bases de données relationnelles. Ainsi j'ai décidé de parler beaucoup plus en détail de MongoDB et de son intégration parfaite avec Node.js et JavaScript

4.3 MongoDB

4.3.1 Présentation

MongoDB se présente comme une solution évolutive, de haute performance et open source. C'est un type de base de données NoSQL particulier, plus précisément c'est un système de base de données orienté document. MongoDB est écrit en C++.

Pour bien comprendre le concept de base de données orienté document, il faut acquérir une certaine terminologie de base sur MongoDB.

L'écosystème de Node.js possède un excellent support de nombreuses bases de données relationnelles et populaires telles que PostgreSQL et MySQL. Les bases de données relationnelles sont robustes et ont été éprouvées dans le temps, elles sont à même de répondre de manière appropriée aux besoins de nombreuses applications web modernes.

Alors pourquoi utiliser MongoDB ?

MongoDB a été choisi pour répondre à notre problématique car il propose un ensemble de fonctionnalités caractéristique de ce type de bases de données.

4.3.2 Schéma libre

Un schéma libre signifie qu'il n'y a pas de structure prédéfinie. Une façon simple d'envisager MongoDB est de prendre l'exemple d'un tableau géant d'objets JSON qui est à la fois rapide et puissant dans l'insertion de données et dans ses capacités de recherche.

Parce que toutes les informations de la même entité peuvent être stockées de façon dynamique au sein d'un seul document, rejoindre les opérations n'est plus nécessaire dans ce type de base de données.

Une base de données orientée document est une base de données où plusieurs document d'une même collection peuvent avoir une structure tout à fait différente. Grâce à cette orientation, un certain nombre de champs peuvent être ajouté à un document et ce peut importe la longueur de ces champs. De plus cela est valable même si le document a déjà été créé.

Une opération sur des jointures coûte très cher, ils exigent une cohérence forte et un schéma fixe. Les éviter permet de réduire les coûts et d'augmenter les performances.

4.3.3 MongoDB vs SGBDR

Pour un puriste de la base de données relationnelle, la notion de schéma de base libre peut dérouter. Cependant il y a certains cas d'utilisation où MongoDB offre des avantages sur son homologues traditionnel.

Tout d'abord il est bien adapté pour le développement itératif. Par exemple, imaginons que nous avons une table de personne et que nous souhaitons ajouter une liste de films favoris. Dans une base relationnelle, cela nécessiterait une table de films supplémentaire et une table de jonction pour stocker les relations. En MongoDB cela pourrait se faire en ajoutant simplement une série de films dans la table personne en fonction des besoins. Pour une entreprise en démarrage à l'évolution rapide dont les besoins sont à court terme, et qui dispose d'équipes d'ingénieurs à temps plein, l'agilité de MongoDB peut jouer un rôle important. MongoDB est également approprié pour les données non structurées. Dans le monde de la finance qui dispose de milliers de type de données différentes, utiliser une base de données relationnelle poserait un certain nombre de défis mais en utilisant MongoDB c'est aussi simple que représenter les données en JSON et les insérer dans la base de données. En outre MongoDB a été conçu dès le départ pour le big data, qui est soutenu par un mécanisme connu sous le nom de fragmentation permettant d'accélérer la recherche.

D'autres cas d'utilisation de MongoDB sont discutés en détail sur le site officiel.

4.3.4 JavaScript

Les demandes faites à MongoDB sont écrites en JavaScript, ce qui est parfait pour une application 100% JavaScript. En outre, ces demandes utilisent une requête de style SGBDR qui réduit l'écart et permet aux développeurs habitués à un langage traditionnelle de requête structurées de s'adapter.

4.3.5 Types de données

MongoDB utilise BSON pour le stockage des données et comme format de transmission réseau. BSON signifie "Binary JSON", il s'agit d'une sérialisation binaire codé en JSON-like. Selon le bsonspec.org, il a été conçu pour être léger, transversal et efficace.

Son principal avantage sur XML et JSON est l'efficacité en terme d'espace et de temps de traitement.

Les documents BSON peuvent être utilisés pour stocker plusieurs types de données comme

“string, integer, boolean, double, null, array, objet, date, données binaires, expression régulière et code source”.

4.3.6 JSON Document-Oriented

MongoDB utilise le format JSON parsé comme structure de données. Les données envoyées à MongoDB peuvent être stockées immédiatement et sans aucun prétraitement. L’approche clé/valeur d’autres bases de données NoSQL comme Redis n’est pas convenable en cas de stockage de valeur plus important qu’une seule valeur.

4.3.7 Temps-réel

MongoDB est très bon pour insérer en temps réel des données et garder un support de transaction extrêmement simple. Il est rapidement devenu la base de données de facto pour Node.js. Jason Hoffman, fondateur de Joyent, a donné une présentation sur Node.js et MongoDB comme étant LA pile moderne pour le Web temps réel en remplacement de la couche LAMP traditionnelle (Linux, Apache, MySQL, PHP).

4.3.8 Fonctionnalités avancées

MongoDB possède un ensemble de fonctionnalités avancées telles que l’index complet, la réplique, la fragmentation et le map/reduce.

4.3.9 Connecteur NodeJS

MongoDB a un pilote source natif et ouvert écrit par Christian Amor Kvalheim appelé node-mongodb-native. <https://github.com/christkv/node-mongodb-native>

4.3.10 Conclusion

MongoDB et plus généralement le mouvement NoSQL, offre une nouvelle approche sur l’utilisation des bases de données. MongoDB a remplacé le SQL par du JavaScript ce qui aurait pu être un facteur inquiétant à première vue, mais il s’est finalement trouvé rapide et facile à apprendre et très flexible.

Un grand merci à son interpréteur JavaScript qui permet de commencer à pratiquer MongoDB très rapidement. En fait, vous pouvez même commencer sans l’installer en utilisant le site try.mongodb.org qui propose un tutoriel interactif pour apprendre MongoDB.

Comme le nombre important et croissant de pilotes est disponible (Java, Scala, C#, Erlang, C, C++, etc), MongoDB évolue rapidement et à été mis en open source en 2009 par 10gen l'éditeur.

4.4 Web Storage

4.4.1 Présentation

Lorsque les développeurs Web pensent à stocker des données utilisateurs, leur premier réflexe consiste à télécharger ces données sur le serveur. HTML5 vient révolutionner tout cela, puisqu'il existe désormais plusieurs technologies permettant aux applications d'enregistrer des données sur un appareil client. Ces données peuvent également être synchronisées avec le serveur ou ne demeurer que sur le client : la décision revient au développeur.

Le stockage côté client est avantageux pour plusieurs raisons. Tout d'abord, il vous permet de faire fonctionner l'application lorsque l'utilisateur est hors connexion, et éventuellement, de synchroniser les données une fois la connexion au réseau rétablie. Ensuite, il vous permet de booster vos performances. En effet, vous avez la possibilité d'afficher un important corpus de données dès que l'utilisateur clique sur votre site, au lieu de devoir attendre qu'il soit de nouveau téléchargé. Enfin, il s'agit d'un modèle de programmation plus simple, pour lequel aucune structure de serveur n'est nécessaire.

Le principe est vraiment très simple : c'est un stockage sous la forme clé / valeur. Il y en a deux différents :

- stockage uniquement pour la durée de la session (et uniquement pour cette instance du site, un autre onglet sur la même page n'est pas affecté) ;
- stockage permanent.

Pour chacun de ces stockages, on a un objet manipulable, respectivement `sessionStorage` et `localStorage`. Les deux objets exposent la même API.

Comme il s'agit d'une API, JavaScript peut l'utiliser. Ainsi il est facile de créer des applications stockant des données sur le client et n'utilisant pas de base de données. On peut également mélanger webstorage avec un serveur de base de données. Afin de permettre un mode offline d'une application web.

4.5 Conclusion

Actuellement les bases de données sont en évolution car de nouvelles tendances arrivent et bousculent le marché du stockage des données. JavaScript est actuellement capable, au travers d'API dédiées ou de Node.js, de se connecter à n'importe quelle base de données. Que ce soit une base de données relationnelle, une base de données NoSQL, une base de données NewSQL et meme aucun serveur avec l'api Web Storage. Le fait que la plupart des bases de données NoSQL utilisent JSON en font une solution très pertinente pour stocker rapidement et simplement des données sans nécessiter un travail complexe. C'est pourquoi à mon avis il est beaucoup plus pertinent d'utiliser une solution de type NoSQL comme MongoDB plutôt que le traditionnel MySQL ou un autre SGBDR. Mais comme dans ma partie Client léger, il est pour l'instant trop tot pour se prononcer. Le choix reste pour l'instant aux développeurs car aucune solution ne sort actuellement du lot. La bataille se joue actuellement pour imposer le standard de la base de données Node.js. MongoDB se démarque du lot en étant disponible en natif dans Node.js

5 Client lourd

5.1 Windows 8

5.1.1 Présentation

Avec l'arrivée des nouveaux matériels - smartphones, tablettes... - le type d'interface que proposait Microsoft avec Windows depuis de nombreuses années ne remplissait plus totalement sa fonction. Difficile d'utiliser un clic droit générant un petit menu avec les doigts directement sur une surface tactile. Il fallait donc faire évoluer l'interface, ce qui entraînait forcément un changement d'habitudes et d'utilisations. Microsoft a donc pris le pari de lancer une nouvelle version de son système d'exploitation, Windows 8, avec une nouvelle interface, un nouveau concept, et pleins de nouvelles fonctionnalités. Ainsi est apparu Metro.

Pour cette nouvelle version, Microsoft a mis au point une nouvelle API pour pouvoir utiliser son noyau : Windows RunTime. Elle garde en plus toutes les meilleures fonctionnalités du .NET en les améliorant. Elle n'est cependant accessible que par les langages suivants : C++, C#, VB.Net et JavaScript

Microsoft cherche à rassembler un maximum de développeurs de tous horizons et à se moderniser avec les derniers standards. C'est pourquoi il a été intégré au noyau de Windows 8 la possibilité de développer en HTML5 et JavaScript. Il est désormais aussi facile de créer une application qu'un site web ! Les développeurs peuvent donc intégrer rapidement et sans trop de modification leurs applications existantes pour client léger et smartphones. Le fait que Windows 8 supporte JavaScript en natif permet l'intégration de l'ensemble de l'écosystème JavaScript et donc une superbe plateforme de développement.

5.2 Gnome3

5.2.1 Présentation

Le souci de plaire aux développeurs n'est pas nouveau au sein du projet GNOME. Dès son origine, le développeur doit pouvoir écrire une application GNOME dans le langage de son choix. Ce fut un des objectifs de GObject puis de GObject introspection, un des fondements de la plateforme GNOME.

Aujourd'hui les bases techniques sont là mais l'équipe est à la recherche d'une approche globale pour multiplier les applications GNOME. L'idée est donc de fournir une solution officielle et unifiée aux problématiques du développement d'application, sans exclusive.

Une décision importante a été prise par les membres de Gnome : le choix du langage par défaut pour le développement d'application GNOME. Attention, il ne s'agit pas de réécrire les programmes existants dans un nouveau langage. Actuellement, le C est utilisé pour documenter la plateforme, car c'est la langue native des bibliothèques. Il faut présenter un langage de haut niveau pour le développement d'applications.

La majorité des applications sont écrites dans un langage de plus haut niveau : python, vala, javascript, perl, C#, etc. Peu sont écrites en C ou en C++.

Les membres de l'équipe Gnome ont décidé d'harmoniser le développement en mettant en avant un seul langage pour le développement d'application Gnome.

Mettre en avant un langage n'est pas rejeter les autres, au contraire. Le but est justement que l'utilisation première des bibliothèques de la plateforme se fasse à travers des passerelles. Hors GObject Introspection est utilisé pour tous les langages. C'est donc un gain pour tous les langages de haut niveau que d'en avoir un mis en avant.

C'est finalement Javascript qui a obtenu le plus large consensus. En voici quelques raisons :

- Les développements récents en font un langage rapide et indépendant de la plateforme.
- Très répandu, de plus en plus en dehors du web avec Windows 8.
- Déjà fait ses preuves avec GNOME Shell et GNOME Document.

Concrètement, les nouvelles applications GNOME seront en javascript, à l'instar de GNOME Document. Les autres ne devraient pas migrer sauf avis du mainteneur. Ainsi GNOME Contact restera en Vala.

5.3 Conclusion

Windows 8 et le prochain Gnome permettent d'écrire des applications native en JavaScript et ainsi de réutiliser du code afin de développer une fois sans beaucoup de modification entre les plateformes. Je n'en ai pas parlé afin de ne pas alourdir cette partie, mais QT, le célèbre framework C++ libre multiplateforme orienté création d'interface graphique pour client lourd, permet l'utilisation de JavaScript au travers de QML. L'équipe de Gnome et de Microsoft misent sur JavaScript pour le développement des futurs applications de Gnome. Ainsi JavaScript n'a plus un usage uniquement web et commence à être utilisé en dehors d'un navigateur

6 Smartphone

6.1 Introduction

Dans votre poche se trouve un appareil qui a changé la vie de milliards de personnes partout dans le monde. Le troisième écran personnel (après la télévision et l'ordinateur) c'est le plus personnel de tous et la priorité clés des entreprises de cette décennie afin de proposer de nouveaux services. Cependant le développement mobile est une activité plus difficile que le développement sur Pc. Les plateformes mobiles sont très fragmentées et les développeurs doivent travailler avec un minimum de ressources. Heureusement le web-mobile permet de traiter plus facilement cette fragmentation en offrant aux développeurs la possibilité de créer des applications qui s'exécutent sur plus de plateformes qu'en natif.

Les statistiques les plus récentes, au moment de la rédaction de ce mémoire, indiquent que Android est en tête du classement des smartphones avec environ plus de 70% de toutes les ventes au dernier trimestres 2012, iOS est à environ 20% dans la même période. Blackberry un très grand nom dans le monde des smartphone ainsi que Windows Phone, Bada et Symbian avec d'autre plateformes plus ou moins connues, se partagent le reste des pourcentages restant.

Global Smartphone OS Shipments (Millions of Units)	Q4 '11	2011	Q4 '12	2012
Android	80.6	238.9	152.1	479.0
Apple iOS	37.0	93.0	47.8	135.8
Others	39.4	158.6	17.1	85.3
Total	157.0	490.5	217.0	700.1

Global Smartphone OS Marketshare %	Q4 '11	2011	Q4 '12	2012
Android	51.3%	48.7%	70.1%	68.4%
Apple iOS	23.6%	19.0%	22.0%	19.4%
Others	25.1%	32.3%	7.9%	12.2%
Total	100.0%	100.0%	100.0%	100.0%

Total Growth Year-over-Year %	55.9%	63.8%	38.2%	42.7%
--------------------------------------	--------------	--------------	--------------	--------------

Ces chiffres montrent clairement que le marché des smartphones est très différent du marché des PC, il n'y a pas vraiment de gagnant et les sociétés voulant profiter ce nouveau canal de communication ont à faire d'importants investissements afin d'être présents dans autant de poches que possible. Beaucoup d'applications doivent être écrites dans au moins deux ou trois plateformes (généralement iOS, Android et Blackberry) pour atteindre une tranche non négligeable du marché.

Actuellement les smartphones ont conquis le marché des téléphones portables ces dernières années.

Beaucoup de choses ont changé depuis 2007, mais comme pour son homologue de bureau, le Web apparaît comme la solution multiplateforme la plus importante à la disposition des développeurs d'aujourd'hui.

6.1.1 La croissance du Web mobiles

Une des avancées de cette nouvelle génération d'appareils mobiles est la possibilité d'utiliser pleinement un véritable navigateur Web mobile, compatible avec la plupart des normes actuelles telles que HTML5, CSS, JavaScript et de nombreux autres standards.

Beaucoup d'entre nous se souviennent d'avoir regardé Steve Jobs présentant les capacités du navigateur Safari mobile dans le premier iPhone, tout en reconnaissant qu'une nouvelle ère avait commencé précisément ce jour-là.

Les navigateurs mobiles n'étaient pas seulement aussi capable que leurs homologues de bureau, ils étaient meilleurs que les pc , ils étaient rapides et étaient entièrement conformes aux normes. La montée en puissance de l'internet mobile a apporté de nouvelles possibilités, en particulier dans les pays à faible pénétration technologique comme l'Amérique latine ou l'Afrique. Les smartphone apparaissent alors comme un bien beaucoup moins cher pour accéder aux informations et services en ligne.

Exemple, en 2010, plus de 30% des accès web en provenance de l'Afrique était réalisés à partir d'un smartphone.

Dans le monde, on estime que plus de 50% de toutes les requêtes web viendront d'appareils mobile d'ici 2015.

6.1.2 Nouveaux paradigmes

Tout cela représente un énorme changement dans notre façon de développer des logiciels.

Un changement radical indiquant que le web mobile est en train de devenir le nouveau canal principal de la présence du web. L'utilisation de la bande passante à partir d'un pc va être inférieure à celle du mobile.

Mais cette nouvelle perspective pose quelques questions :

- Combien de plateformes dois-je tester pour mon site web ?
- Dois-je me soucier des téléphones mobiles bas de gamme ?
- Quelles bibliothèques puis-je utiliser pour accélérer mon développement ?
- Quel est le niveau de support des standard dans les principaux navigateurs mobiles.

Pour ce faire, nous allons étudier les technologies suivantes, qui sont actuellement très prometteuses et qui ont une feuille de route très intéressantes

- PhoneGap
- Sencha Touch
- JQuery Mobile

Il y a bien sûr d'autres technologies intéressantes mais je vais me limiter à l'étude de ces 3 frameworks.

6.2 Phonegap

6.2.1 Introduction

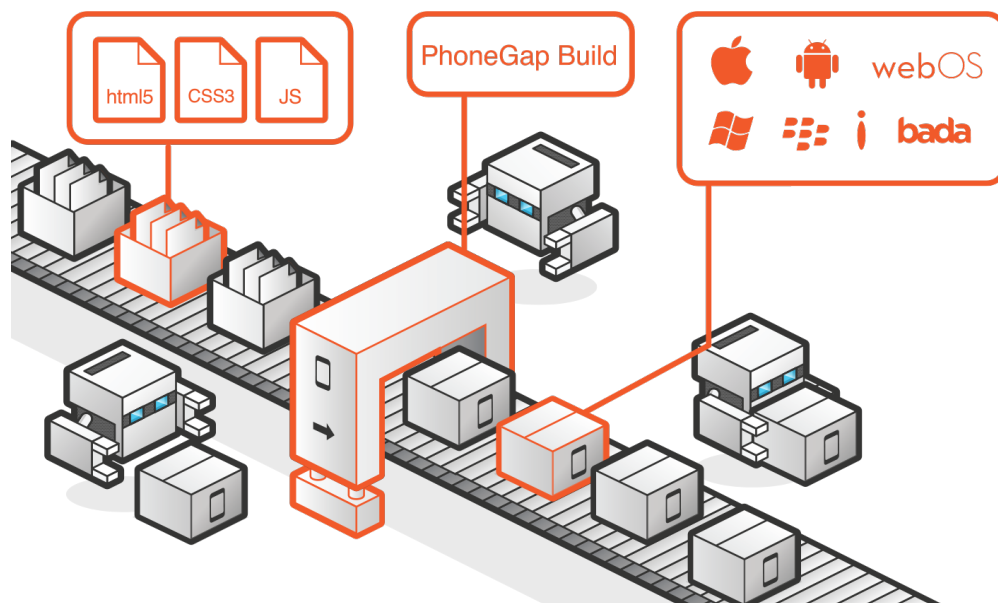
Développer des applications pour les appareils mobiles peut se faire en utilisant différentes approches et langages. La plupart des applications sont développées en native, ce qui signifie qu'ils sont développés en Java, Objective C, ou un autre langage compris par le SDK disponible dans l'appareil.

Alors que le développement natif permet une plus grande flexibilité et de meilleures performances, le problème se pose lorsque vous souhaitez déplacer une application d'une plateforme à l'autre. Tout d'un coup vous devez réécrire l'application presque à partir de zéro. Et si vous voulez passer à une autre plateforme la même chose se produit.

Il doit y avoir une meilleure façon de faire !

Toutes les plateformes mobiles actuelles supportent l'utilisation d'application web. Ce sont des applications codées entièrement en HTML, JavaScript. Pour des applications simples, ou pour des applications qui n'ont pas besoin d'interagir avec les capacités de l'appareil, cela fonctionne très bien. Mais du moment où vous avez besoin d'accéder au système de fichiers, travailler avec la caméra, l'accéléromètre ou d'autre composants de l'appareil, vous commencez à avoir un plus grand accès à l'appareil.

C'est là qu'intervient PhoneGap.



Lorsque vous utilisez l'API de PhoneGap, vous pouvez écrire votre application sans écrire de code natif (Java, Objectif-C, etc). Au lieu de cela, les technologies web sont utilisées, et sont

hébergées en local dans l'application elle même. Et parce que ces API JavaScript sont compatibles sur toutes les plateformes mobiles et construites sur les standards du Web, l'application doit être portable sur les plateformes des appareils avec peu ou pas de modifications.

PhoneGap enveloppe votre HTML et votre JavaScript avec juste assez de code natif pour que votre application web se sente plus à l'aise sur l'appareil. Cette enveloppe est différente pour chaque plateforme, mais elle expose ses capacités commune de manière cohérente. Cela vous permet d'écrire moins de code sur de multiples plateformes.

PhoneGap est disponible sur les plateformes suivantes : iOS, Android, Blackberry, Windows Phone, Palm WebOS, Bada et Symbian.

Depuis que PhoneGap enveloppe votre code HTML et JavaScript dans une application native, vous gagnez la possibilité de soumettre votre application au store de l'appareil. Chose que vous ne pouviez pas faire depuis une application web simple. Gardez à l'esprit, cependant, que la plupart des stores veulent que votre application ait le style d'une application native et que certains store sont plus strict que d'autres.

Voici une liste des composants supporté par PhoneGap.

	iPhone / iPhone 3G	iPhone 3GS and newer	Android	Blackberry OS 5.x	Blackberry OS 6.0+	WebOS	Windows Phone 7 + 8	Symbian	Bada
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓	✓
Compass	X	✓	✓	X	X	✓	✓	X	✓
Contacts	✓	✓	✓	✓	✓	X	✓	✓	✓
File	✓	✓	✓	✓	✓	X	✓	X	X
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓	X	X	X	✓	X	X
Network	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓	✓	X

✓ - supported feature
X - unsupported feature due to hardware or software restrictions

6.3 SenchaTouch

6.3.1 Introduction

Sencha Touch est un framework MVC spécialement conçu pour créer des applications web mobile pour les appareils tactile. Sencha Touch permet aux développeurs de créer des applications pour les plateformes mobiles qui disposent de navigateurs supportant les derniers standards, comme le moteur de rendu Webkit.

Au moment d'écrire ces lignes, la dernière version disponible de Sencha Touch était la version 2.2.0

Sencha Touch est un gros framework, qui peut sembler intimidant pour les développeurs JavaScript habitués à utiliser des petites bibliothèques telles que jQuery ou Prototype.

Sencha Touch est conçu comme un tout, y compris la plupart des fonctions des services offerts par d'autres frameworks, et il peut être facilement étendu de nombreuses et différentes façon pour s'adapter aux besoins des développeurs de différents domaines d'expertises

Vous n'avez généralement pas besoin d'utiliser d'autres bibliothèques que Sencha Touch dans votre projet, si vous avez besoin d'une fonction particulière vous êtes sûr que le framework l'embarque par défaut.

Le choix explicite de Webkit est intéressante, l'équipe de Sencha Touch a pris une décision délibéré de ne pas soutenir d'autres moteurs de navigateurs mobile, tels que Gecko (Firefox), Presto (Opéra), ou Trident (Internet Explorer). Le soutien exclusif des navigateurs modernes permet à Sencha Touch d'utiliser de nombreuses technologies Web les plus avancées.

Ce choix influe également sur l'expérience du développeur, parce que Safari ou Google Chrome peuvent être utilisé pour déboguer les applications Sencha Touch sur un environnement de bureau comme Linux, Windows ou OS X.

L'équipe de Sencha a récemment annoncé le support d'Internet Explorer 10 pour Windows Phone 8.

Quel type d'application peut-on écrire avec Sencha Touch.

Apple, dans l'une des premières version de son guide sur les directives de design pour IOS a énoncé qu'il existe trois grands types d'applications mobiles qui peuvent être créé pour l'iPhone.

- les applications utilitaires, comme la météo ou les informations d'informations de stock.
- Les applications de productions, comme les applications d'affaires ou orientés document.
- Les applications immersives comme les jeux vidéo.

Suite à cette taxonomie simple, Sencha Touch est plus adapté pour des applications délivrant

les deux premiers types. Bien qu'il est certainement possible de créer des jeux ou d'autres types d'applications mettant en vedette des expériences utilisateurs complexe, ce mémoire ne couvre que la problématique des applications des deux premiers types.

6.3.2 Un peu d'histoire

Retour en 2005, le mouvement web 2.0 est en train de transformer radicalement la notion de contenu web. Les sites en AJAX comme Gmail ont montré au public qu'un nouveau type d'interaction était possible, qu'un nouveau type de contenu a pu être proposé dans les pages web classiques sans l'aide d'extension propriétaires. Douglas Crockford expliquait que JavaScript était un grand langage incompris par beaucoup, et les bibliothèques comme Script.aculo.us et Prototype offrait aux développeurs de concrètes et solides raison pour développer des applications cross-plateformes.

Au milieu de toute cette agitation , Yahoo a publié la première version de sa bibliothèque YUI, permettant de développer des applications complexes à la “desktop-like” à travers les systèmes d'exploitations et navigateurs. YUI peut être considéré comme une œuvre précurseur, après quoi plusieurs autres bibliothèques sont apparus au fil des ans.

Pendant ce temps, Jack Slocum a commencé à travailler sur un ensemble d'extensions pour YUI appelé YUI-Ext. Après quelques versions, l'intérêt pour sa bibliothèques a tellement augmenté qu'il à supprimé l'obligation d'utiliser YUI en complément, rendant la bibliothèque en mesure d'utiliser Prototype et/ou YUI pour un niveau de compatibilité cross-plateformes.

6.3.3 Ext JS est né

Pendant des années, Ext JS a établi la norme en terme de compatibilité cross-browser et de conception, permettant aux développeurs de créer des applications de navigation complexe en une fraction de temps, et sans avoir à se soucier des problèmes de compatibilité entre navigateurs. En 2009, la société derrière Ext JS incorporé comme Sencha Inc, dont le siège se trouve à Redwood City, en Californie.

En 2009, la hausse des martphone à écran tactile et, plus tard, 'liPad, a incité l'équipe d' Ext JS à créer une version du framework orienté exclusivement pour ces nouveaux dispositifs : le résultat de leurs efforts est Sencha Touch, sortie en version1.0 à la fin 2010.

La première version de Sencha Touch n'était pas totalement compatible avec le version courante d'Ext JS, et il a aussi été critiqué pour ses relatives faibles performance, en particulier sur les anciens appareils tels que l'iPhone 3G. Pour répondre à ces question, Sencha Touch 2 a été

publié en Mars 2012 offrant un tout nouveau moteur de rendu basé à 100% sur Cascading Style Sheet (CSS), et un nouveau système de classe compatible avec Ext JS 4

6.3.4 Caractéristiques principales

Sencha Touch est plus que juste un framework complet orienté vers la création de services et d'applications orienté productivité, il est en fait un système web complet d'entreprise pour application cross-plateforme, avec les caractéristiques suivantes :

- Nombreux widget disponibles, largement inspiré par iOS, tant dans la conception que dans les fonctionnalités.
- Moteur de rendu rapide basé sur CSS, qui peut être accéléré par le matériel dès les smartphones moderne.
- Une architecture bien définie, Sencha Touch utilise une architecture MVC.
- Des connecteurs intégrés pour les serveurs de transfert de données réseau, telles que les services web REST et le soutiens aux application web offline mobile.
- Un système de construction de ligne de commande, la gestion de la fusion et de la minification du code de l'application, ainsi que la création d'applications natives pour Android et iOS.

Une documentation complète est disponible comme un ensemble de pages HTML dynamiques, y compris la recherche et le filtrage des fonctionnalités sans nécessiter d'infrastructure serveur. Sencha Touch peut être considéré comme un framework “tout en un”, y compris toutes les API et outils nécessaire pour créer vos applications mobiles.

6.3.5 Supprt appareils et navigateurs

Sencha Touch au moment d'écrire ces lignes ne supporte que les plateformes suivantes :

- iOS depuis la version 3
- Android depuis la version 2.3
- BlackBerry OS depuis la version 6 (uniquement pour les appareils équipés de WebKit)
- Windows Phone 8

Sencha Touch est un framework basé à 100% sur le navigateurs, et vous pouvez déployer vos applications Sencha Touch en utilisant n'importe quelle technologie coté serveur, à l'instar de PHP, Java, Ruby on Rails, .Net ou tout autre langage de votre choix.

6.3.6 Licence

Sencha Touch est disponible sous plusieurs licences :

Pour des projets open-source :

- Si vous prévoyez de distribuer votre application en divulguant pleinement le code source, il y'a une version de Sencha Touch distribué sous licence GPLv3
- Si vous ne souhaitez pas utiliser la licence GPLv3, vous pouvez également utiliser la licence FLOSS.

Pour des projets commerciaux :

- Vous pouvez utiliser Sencha Touch gratuitement, sans aucun frais que se soit par applications, par utilisateur ou par développeur
- Pour les applications embarquées, vous pouvez utiliser Sencha Touch gratuitement jusqu'à 5000 installations.

Enfin, une licence OEM commercial est disponible aussi bien pour les entreprises désireuses de distribuer Sencha Touch comme une partie de leurs propres applications ou services commerciales.

6.4 JQuery Mobile

6.4.1 Introduction

jQuery mobile est un framework open source JavaScript UI construit sur la populaire bibliothèque jQuery, créé par John Resig au cours de la dernière décennie.

Le développement de jQuery mobile a commencé mi-2010, et est rapidement devenu l'un des frameworks JavaScript les plus populaires. Aujourd'hui jQuery Mobile est utilisé dans plus d'application web mobile que n'importe quel autres framework.

jQuery Mobile est un projet open source, hébergé sur Github et avec un site internet très complet, énormément de documentation, d'exemples et de références à des applications créées avec ce framework. Au moment d'écrire ces ligne, la version actuelle de jQuery Mobile est la version 1.3.1.

6.4.2 Plateformes supportés

jQuery Mobile fonctionne sur la grande majorité des pc moderne, smartphone et tablette et des plateformes eReader.

En outre, les téléphones et les navigateurs plus anciens sont également supportés en raison d'une évolution progressive. C'est probablement l'une des caractéristiques les plus importes de jQuery Mobile

6.4.3 Compatibilité

Les utilisateurs des navigateurs mobiles les plus avancés peuvent profiter d'une expérience plus complète, avec des transitions de pages animées basées sur Ajax. Au moment d'écrire ces lignes cette liste comprend les systèmes / navigateurs suivant :

- iOS depuis la version 3.2
- Android depuis la version 2.1
- Windows Phone depuis la version 7
- BlackBerry depuis la version 6, notamment Playbook
- Palm WebOS depuis la version 1.4
- Firefox mobile depuis le 10 bêta
- Skyfire depuis la version 4.1
- MeeGo depuis la version 1.2

- Samsung Bada depuis la version 2.0
- Navigateur UC
- Kindle et Kindle Fire
- Nook Color depuis la version 1.4.1

Une liste impressionnante ! Toutes les plateformes importantes de smartphone à écran tactile sont aujourd'hui disponibles et sont représentées et soutenues par jQuery Mobile.

Sur les plateformes de bureau, jQuery Mobile est compatible avec Windows, Linux et Mac OS X sur les versions des navigateurs suivants :

- Firefox depuis la version 4
- Chrome depuis la version 11
- Safari depuis la version 4
- Internet Explorer depuis la version 7
- Opéra depuis la version 10

Un des plus grands avantages en regardant les listes ci-dessus, c'est que jQuery Mobile est l'un des frameworks les plus largement compatibles aujourd'hui. Même mieux, son grand support des navigateurs de bureau permet aux développeurs d'utiliser différentes plateformes pour construire et tester leurs applications.

Étant donné que les versions les plus récentes de ces navigateurs intègrent des outils de développement, elle augmentent aussi sont attirés auprès des développeurs.

6.4.4 Compatibilités avec les anciennes plateformes mobiles

Mais que faire si nos utilisateurs ou les exigences requièrent une compatibilité avec certaines des anciennes plateformes ?

Est-ce que jQuery Mobile peut nous aider dans ce cas ?

Les applications jQuery Mobile sont construites avec une dégradation progressive par défaut des fonctionnalités. Les anciennes plateformes, incapables d'afficher les dernières normes CSS et JavaScript, vont tranquillement afficher par défaut la structure HTML des applications, ce qui pourrait ou non être une solution idéale, mais c'est néanmoins une réponse par défaut. Par exemple, les navigateurs suivants ont une expérience améliorée, à l'exception d'Ajax pour la navigation :

- BlackBerry 5.0
- Opera Mini 5.0 à 6.5
- Nokia Symbian 3

Et quelques autres navigateurs ne peuvent profiter que d'une expérience de base en HTML, non amélioré :

- BlackBerry 4.x
- Windows Mobile 6 et plus.
- Plateformes de smartphone plus anciennes, y compris les téléphones.

6.4.5 Principales caractéristiques

Une liste succincte des principales caractéristiques de jQuery Mobile

- Construit sur la syntaxe de jQuery afin d'avoir une syntaxe familière, cohérente et une courbe d'apprentissage minimale
- Compatible avec toutes les principales plateformes de bureau et mobiles : iOS, Android, BlackBerry, Palm WebOS, Nokia/Symbian, Windows Mobile, Opera Mobile/Mini, Firefox Mobile, et tout les navigateurs de bureau récents.
- Léger (environ 20ko une fois compressé avec toutes les fonctionnalités mobile) et minimal
- Utilisation du HTML5 qui conduit à un développement des pages rapides et à un minimum de scripting requis.
- L'utilisation de l'amélioration progressive apporte un contenu de base et de fonctionnalités à tout mobile, tablette ou navigateur avec une expérience riche.
- Initialisation automatique des widget jQuery en utilisant des attributs HTML dans le code HTML.
- Des fonctions d'accessibilité telles que WAI-ARIA sont également incluses afin de s'assurer que les pages puissent travailler avec les lecteurs d'écran (par exemple, VoiceOver dans iOS) et d'autres technologies d'assistance.
- Prise en charge des événements tactiles et de la souris en rationalisant le processus du support tactile.

La chose la plus importante à savoir sur jQuery Mobile est qu'il s'agit d'une bibliothèque d'interfaces utilisateur, et non pas un plugin de jQuery. C'est une bibliothèque qui aura des balises HTML en entrée et qui utilise des styles prédéfinis en les adaptant aux capacités des navigateurs actuels. Ce n'est pas un framework complet comme .NET, Java, ou encore Sencha Touch qui fournissent des services de niveau inférieur, comme la sérialisation, le stockage ou le réseau.

jQuery mobile s'appuie sur JavaScript et les fonctionnalités du HTML5 prises en charge par le navigateur utilisé.

Cette première caractéristique détermine pourquoi le support des navigateurs mobiles de jQuery Mobile est important comparé aux développeurs Java devant déployer leur propre code pour mettre en œuvre des comportements complexe comme le stockage ou pour interagir avec le matériel exposé par le navigateur de l’hôte (géolocalisation, boussole, etc)

Une autre caractéristique importante de jQuery Mobile est qu’elle n’impose aucune sorte de structure du code JavaScript dans votre application, la principale composante de l’application étant les fichiers HTML qui définissent la sémantique de l’interface utilisateur, mais pas son look. En général, les développeurs s’appliqueront à utiliser le comportement de jQuery en utilisant sa norme, sa syntaxe comme avec n’importe quelle page web ordinaire.

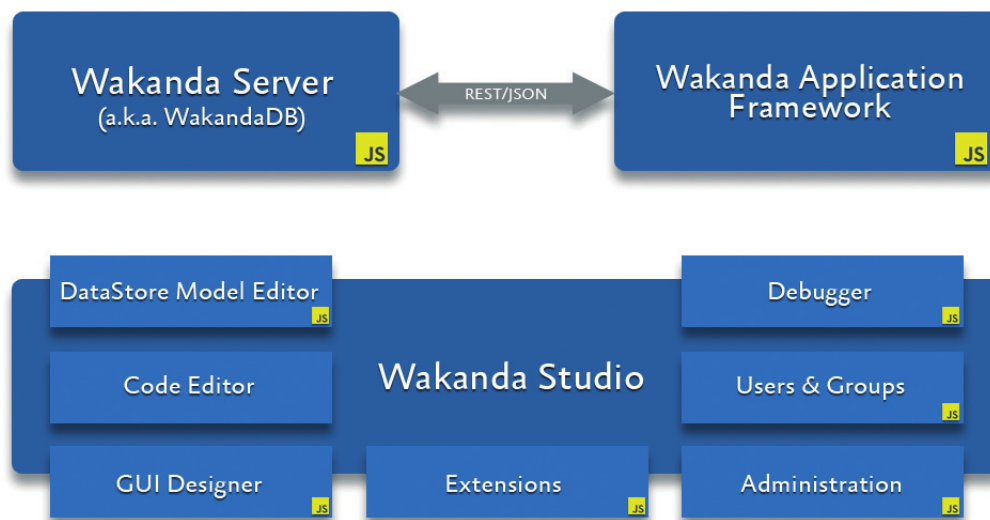
6.5 Conclusion

7 Plateforme

7.1 Wakanda

7.1.1 Présentation

Wakanda est une plateforme Open Source qui propose une dual licence, avec des capacités de développement strictement identique dans les deux cas. La création d'une communauté d'utilisateurs et contributeurs est clairement l'ambition première affichée par l'éditeur de Wakanda, dans le but de garantir la pérennité des projets et de s'adapter au foisonnement d'idées, de frameworks, d'appareils et d'usages qui se déroule sous nos yeux.



Au cœur de la plateforme se niche WakandaDBn un datastore objet NoSQL ultra-performant, qui, joint au moteur d'exécution JavaScript, basé sur Webkit, et à un serveur HTTP communiquant via JSON/REST constitue Wakanda Server, le back-end multi-plateformes (Linux, Mac, Windows) qui permettra d'héberger une solution Wakanda sur un serveur dédié, sur un Cloud privé ou public, en mode SaaS, etc...

Le framework Wakanda est automatiquement déployé vers les clients HTML et embarque un Data Provider qui se comporte comme un proxy des différentes DataClasses générées coté serveur. Ainsi de façon transparente, tous les objets JavaScript générés à l'aide du modèle de données sont exploitables via les différents widgets fournis avec le framework, mais également au travers de requêtes REST pour potentiellement connecter tout type d'interface, par ligne de commande ou à l'aide d'adaptateurs que pourront fournir des tierces parties, la communauté, ou l'éditeur 4D lui-même. Enfin, le studio Wakanda vient compléter la suite de développement en proposant un IDE dédié capable de contrôler, via une application Desktop Mac ou Windows, aussi bien le modèle, les widgets, et l'adaptation cross-device de chaque écran coté client. Le studio fait la part belle à la gestion graphique du projet, et fait en sorte d'économiser à chaque étape l'écriture de code «technique». Le développeur peut dès lors se concentrer sur ses propres scripts et méthodes qui contiendront l'intelligence métier qui fera la valeur de son application. Outre les différents éditeurs (modèles, code, UI, utilisateurs et groupes) le studio propose une fenêtre d'administration des solutions et projets et un débogueur qui permet de tracer le code JavaScript tournant sur le serveur. L'installation, tout comme le déploiement, s'effectuent par drag-and- drop d'un package sans aucune configuration. L'hébergement d'une solution Wakanda peut s'effectuer sur un serveur dédié, sur un VPS (serveur dédié virtuel, ou même sur une plateforme IaaS Cloud comme Amazon EC2 ou Rackspace. Côté mobile, Wakanda permet de réaliser des applications Web ou Hybrides, c'est-à-dire embarquées dans un runtime natif généré par PhoneGap. Il est également possible de créer des applications mobiles natives et d'accéder au Serveur via ses API REST et JSON-RPC. Pour ceux qui sont à la recherche d'une plateforme tout en un, Wakanda est une solution intéressante avec un support compétent.

8 Futur

8.1 EcmaScript6

8.1.1 Présentation

JavaScript a parcouru un long chemin depuis ses humbles débuts il y'a à peu près vingt ans. Aujourd'hui , les développeurs sont en train d'écrire des milliers de ligne de code qui créent des applications JavaScript complexes. Avant de nous plonger dans les caractéristiques de ES6, vous pouvez regarder le tableau d'ensemble qui est défini dans le projet de spécifications, en termes d'exigences, d'objectifs , de moyen et de thèmes bordés.

L'un des objectifs de ES6 est d'être un meilleur langage pour la création :

- applications complexes
- bibliothèques
- générateurs de code

8.1.2 Compatibilité

Le tableau de compatibilité d'ES6 est très utile, car elle nous indique les caractéristiques pris en charge dans le s navigateurs actuel. Elle nous donne également un lien pratique pour les spécifications de chacun des caractéristiques énumérées.



8.1.3 Nouveautés

Voici une liste non exhaustive des fonctionnalités qu'apporte EcmaScript 6

- - Block scoping with let (en utilisant Firefox)
 - Bloc scoping with const (en utilisant Chrome)
 - Classes (en utilisant Traceur)
 - Default function parameters (en utilisant TypeScript)
 - Collections (en utilisant NodeJS)
 - Destructuring (en utilisant Firefox)
 - Rest parameters & Spread operator (en utilisant Grunt plugin Traceur)
 - Iterators (en utilisant Firefox)
 - Array compréhension (en utilisant Firefox)
 - Modules (en utilisant ES6 Module Transpiler)

8.2 Futur de JavaScript

Le futur de JavaScript est extrêmement lié au futur du web et plus généralement à l'évolution de l'informatique.

L'avenir du web est un sujet périlleux, car il est toujours difficile de cerner une tendance (l'année prochaine) et futurologie (dans 10ans).

Pour ce mémoire, je vais essayer de donner mon point de vue sur l'évolution d'internet et pourquoi j'ai décidé de parler de JavaScript et mise sur lui comme un acteur majeur de cette évolution. Comprenez par là que c'est mon avis et qu'à travers les parties que j'ai présenté dans mon mémoire vous êtes libres ou non d'adhérer à mon avis.

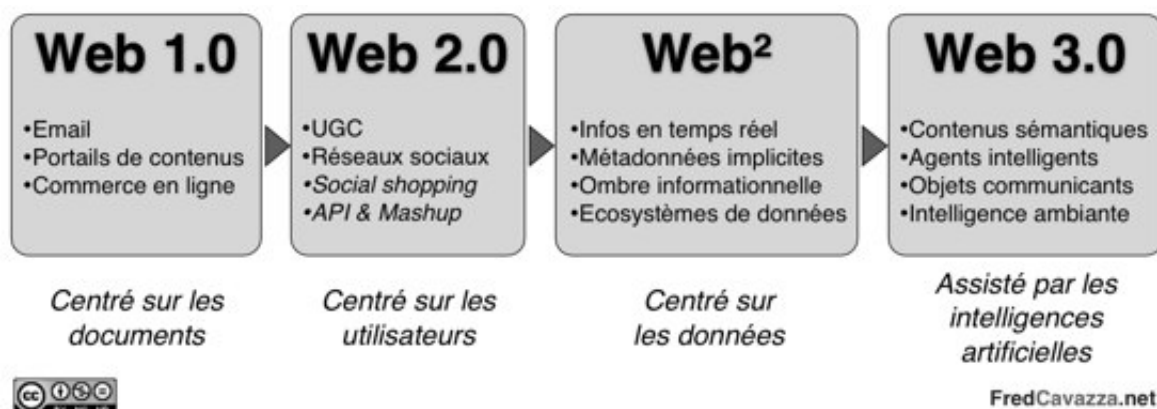
8.2.1 Du web 1.0 au web 2 squarred

JavaScript a actuellement 18ans et le moins que l'on puisse dire, c'est qu'en 18ans, tout à changé même si les fondements sont toujours là. De nouveaux usages se sont développés et se développent encore, mais ils se cumulent avec les anciens, du moins pour les technologies grand public. Depuis les années 2000 et la bulle d'internet, je ne pense pas me tromper en disant que l'histoire du web s'écrit par cycle de 5 ans marqué par la domination d'acteur sur-puissants. Les années 2000 ont été les années des portails (Yahoo, MSN, AOL...) qui concentraient l'audience avec une offre exhaustive de contenus et de services. Les années 2005 sont les années de Google et de son moteur de recherche concentrant plus de 80% du marché. Les années 2010, dans lesquels nous nous trouvons, sont marquées par la domination des réseaux sociaux et notamment Facebook qui comporte quasiment 1 milliard d'utilisateurs.

Puisque nous évoquons l'histoire de l'évolution du web, il me paraît indispensable de parler brièvement des grands stades d'évolution.

- Le web 1.0, centré sur les **documents**, avec la généralisation d'outils comme l'email, des portails de contenus ainsi que le commerce en ligne ;
- Le web 2.0, centré sur les **utilisateurs**, avec la généralisation des réseaux sociaux, de contenus générés par les utilisateurs, les pratiques du social shopping et de nombreuses innovations autour des API et mashups ;
- Le web squarred, centré sur les données, principalement axé sur la circulation temps-réel de l'information. Les données sont agrégées dans des écosystèmes, qui sont rattachés à tous types d'objets (ombres informationnelles) et générées de façon mécanique (métadonnées implicites) ;

- Le web 3.0, centré sur les agents intelligents, mais ceci n'est pour l'instant qu'un concept abstrait.



8.2.2 Pourquoi parler du Web Squared ?

Je suis persuadé que nous sommes à la fin du web 2.0, l'émergence des terminaux mobiles a bousculé le marché et a ouvert la prochaine grosse itération du web : le web en temps réel, la réalité augmentée, les objets communicants, le cloud sont des innovations en cours de maturation qui vont entièrement changer notre façon de consommer, travailler et communiquer.

8.2.3 L'essor du temps réel : un cerveau collectif

Le microblogging est la révolution qui oblige l'informatique à devenir temps réel. Alors que le blogging a ajouté des dizaines de millions de sites à indexer chaque jour et même chaque heure, le microblogging nécessite une mise à jour instantanée. Cette nécessité implique de profond changement à la fois dans l'infrastructure et dans la méthode. La recherche d'un sujet populaire sur Twitter par exemple est confronté au message suivant "Voyez ce qu'il se passe maintenant" suivi, quelques instant plus tard par "42 résultats supplémentaires. Actualisez la page pour les consulter."

La recherche en temps réel encourage la réponse en temps réel. Twitter étant massivement temps réel est devenu en peu de temps, la première source d'informations dans le monde. Avec des services comme les mises à jour de statut des réseaux sociaux, une nouvelle source de données a été ajoutée au Web. Des modifications en temps réel et qui sont devenu indispensable pour la plupart d'entre nous.

L'utilisation du temps réel notamment par Twitter a provoqué de profond changement dans notre société, tant au niveau économique, politique que sociétale.

Le Guatemala et l'Iran ont tous deux ressenti l'effet Twitter, les protestations politiques ayant été lancées et coordonnées sur Twitter.

8.2.4 Et JavaScript dans tout ça ?

JavaScript est par nature un langage événementiel ce qui le rend parfait pour les applications nécessitant le temps réel. Comme je viens de l'expliquer récemment, le temps réel est la prochaine évolution du Web. Le web squarred annonce d'après moi la révolution de JavaScript dans l'informatique et le juste retour du bâtons. Grâce à l'émergence du temps réel, JavaScript va enfin prouver ce dont il est capable. Node.js et Socket.IO en sont les exemple vivant de toute ce bouillonnement autour de l'ère du temps réel. JavaScript est tellement préparé au temps réel que tout les frameworks cotés clients l'ont prévue. Le nouveaux standard de EcmaScript (Harmony) et CommonJS prépare JavaScript à devenir un acteur de choix dans le développement d'application à but temps réel.

9 Conclusion

À travers ce mémoire, nous avons pu voir que JavaScript a un très fort potentiel dans le développement 3-tiers d'application multi-plateformes. Nous avons abordé les principaux composants de l'architecture 3-tiers et de comment JavaScript peut répondre à chaque composant. Ainsi nous avons pu voir que JavaScript a une réponse pour chaque problématique.

Le JavaScript, a de nombreux atouts, notamment son interopérabilité sur l'ensemble des navigateurs récents du marché et sa bataille importante pour toujours plus de performance et de fonctionnalités mené par Google Chrome et Mozilla Firefox.

JavaScript a actuellement un regain d'actualité avec l'explosion du marché des smartphones et la multiplication des navigateurs mobiles et des systèmes d'exploitation. De plus l'arrivée de nouveaux concurrents comme Tizen, Ubuntu Touch et Firefox OS va stimuler la concurrence autour du développement d'application multi-plateformes.

Le JavaScript avec Node.js s'invite du côté du serveur et répond à une forte demande concernant les applications temps réel et à forte demande. Node.js possède un nombre impressionnant de modules et est un des dépôts les plus suivis sur github. La feuille de route de Node.js pour la version 1.0 est impressionnante ce qui reste prometteur pour son avenir.

Les développeurs du libre ont particulièrement bien compris ce nouveau besoin. Si l'on regarde les statistiques de Github, JavaScript représente 21% des projets hébergés. Ce qui montre la tendance du libre à utiliser massivement JavaScript dans l'avenir.

Les systèmes d'exploitations ont également bien saisi cette tendance en misant également sur le JavaScript pour le développement de leur application. Ainsi Microsoft inclus le support de JavaScript dans Windows 8 et a ainsi introduit le premier langage non développé par la firme. Néanmoins tout n'est pas forcément au mieux avec la version actuelle de JavaScript et de nombreuses entreprises proposent des alternatives à du JavaScript brut. CoffeeScript (Dropbox) et TypeScript (Microsoft) sont des surcouches de JavaScript afin de permettre aux développeurs de coder plus efficacement que ne leur permet JavaScript et ainsi répondre aux principales critiques effectuées sur le code JavaScript. Google propose Dart, un concurrent direct de JavaScript, qui

n'a jamais réussi à percer.

Heureusement EcmaScript Harmony, va permettre à JavaScript d'être plus adapté dans la création d'application en JavaScript et la plupart des navigateurs le supporte déjà. Ainsi avec la nouvelle version Harmony orienté application, la question de ce mémoire sur JavaScript, est il uniquement un langage web ne devrait plus être valable officiellement. Pour l'instant au vu des frameworks disponibles la réponse est non mais ce n'est pas une version officielle.

L'avenir de JavaScript dans les 5 prochaines années sera marqué par l'évolution des moteurs JavaScript et aussi par l'activité autour de Node.js coté serveur ce qui en fait un acteur majeur des prochaines années. Mais est-ce une tendance pour les 5 prochaines année à venir comme l'a été Python ou Ruby ou un sérieux changement pour le développement d'application 3-tiers ?

Liste des tableaux

Index

bla, 16