EPSI
114 rue Lucien FAURE
33 300 Bordeaux

Cdiscount
126 Quai de Bacalan
33 300 Bordeaux

Conception et réalisation d'une application web de qualité, un cheminement d'ingénieur logiciel

Payet Emmanuel

Ceuille Arnaud

21 septembre 2015

Taudin Grégory

Remerciements

Je remercie Cdiscount, l'entreprise pour laquelle j'ai eu la chance de travailler pendant 3 ans.

J'ai eu la chance d'avoir deux tuteurs durant cette longue période. Je remercie donc Olivier ABRARD, mon premier tuteur qui a toujours vu en moi un énorme potentiel, et qui m'a soutenu dans mes projets. Je remercie ensuite Grégory TAUDIN, mon second tuteur, qui a toujours eu une grande confiance en mes compétences, en me poussant toujours au maximum de mes capacités.

J'ai rencontré de nombreuses personnes à Cdiscount avec qui il était agréable de travailler. Nombreux d'entre eux sont devenus mes amis. Edouard SOUAN, alternant venant lui aussi de l'EPSI, membre de mon équipe est la personne avec qui j'ai travaillé le plus longtemps. Je le remercie pour sa patience et son entrain.

Je remercie aussi l'EPSI, sans qui toute cette experience n'aurait pas été possible.

Je remercie bien sûr grandement ma famille qui malgré la distance me porte un énorme soutien.

ATTESTATION NON-PLAGIAT
(A inclure dans le mémoire professionnel)

Je soussigné(e)

Nom : Payet Prénom : Emmanuel

Auteur du mémoire professionnel pour le Titre 1 RNCP – Expert informatique et Système d'information

(Titre du mémoire) Conception et réalisation d'un application web de qualité, un cheminement d'ingénieur logiciel

Déclare :

- Que ce mémoire est un document original, fruit d'un travail personnel
- Avoir obtenu les autorisations nécessaires pour la reproduction d'images, d'extraits, de tableaux, figures ou graphiques
- Ne pas avoir contrefait, falsifié, copié tout ou partie de l'œuvre d'autrui afin de la faire passer pour mienne ;

Atteste :

- Que toutes les sources d'information utilisées pour ce travail de réflexion et de rédaction sont référencées de manière exhaustive et claire dans la bibliographie/webographie de mon mémoire professionnel
- Etre informé(e) et conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, et que le plagiat est considéré comme une faute grave et sanctionné par la Loi.

Date et Signature :

Fait le 15/09/2015 A Bordeaux

Signature :

Sommaire

Résumé

Il existe 3 grandes familles d'applications informatiques : **Application de bureau**, **web** et **mobile**. La majeure différence est le support sur lequel elles sont accessibles, et par extension, les technologies à utiliser pour le développement de celle-ci. Les barrières technologiques entre les supports sont de plus en plus fines, et il est désormais aisé d'utiliser une même famille de technologies pour plusieurs supports. Cela permet de réduire grandement les coûts de développement et de maintenance. Le web est désormais la cible de toutes les applications, de par sa simplicité.

Cdiscount, leader de l'e-commerce en France, est de plus en plus présent à l'international. De ce fait, le nombre d'applications à développer est croissant, car chaque pays possède ses spécificités. Elle sous-traite actuellement le développement de ses applications mobiles et songe fortement à réintégrer le développement de celles-ci en interne. Cependant, l'expertise technique de la société est le web, et pas le mobile. N'est-il pas possible pour Cdiscount d'utiliser son expertise pour créer des applications mobiles via les technologies web modernes ?

En parallèle, de nombreux chantiers sont en cours pour rénover le fonctionnement du SI (Système d'Information) de Cdiscount et changer l'architecture interne ainsi que les méthodes de développement utilisées. Elle encourage fortement les nouvelles applications à utiliser les méthodes agiles et à implémenter l'architecture sur laquelle elle travaille (microservices).

Lors de mon cursus d'ingénierie informatique à l'EPSI Bordeaux, j'ai eu l'occasion de travailler en alternance durant 3 ans à Cdiscount (de 2012 à 2015). Mon travail était tout d'abord essentiellement le développement d'applications. Gagnant de l'expérience et de la confiance auprès de mes supérieurs, mon travail de tous les jours a progressivement évolué. En plus du développement, j'ai eu du temps dégagé pour faire de la recherche sur les nouvelles technologies, et de parfaire mes talents de conception.

Je faisais partie de l'équipe Lab (R&D). C'est l'équipe chargée de maintenir l'innovation à Cdiscount. Pour cela, elle est en constante veille technologique et propose de nouvelles idées de technologies au SI. Leur objectif est de maintenir à jour les technologies utilisées, et donc de construire la plateforme de demain. Elle a le champ libre sur le choix des technologies.

Le monde du web évolue très vite et est difficile à suivre. De nombreuses technologies apparaissent régulièrement. Faire une application web moderne, et de qualité n'est pas un travail facile. Trouver la solution idéale et faire les bons choix techniques selon le projet et le style de développement de l'équipe, demande une bonne compétence d'analyse et une veille technologique active.

Quels sont les choix à effectuer et quels sont les étapes à suivre pour la conception et la réalisation d'une application web moderne de qualité ?

Notre équipe s'est proposé pour créer un projet de type POC (Proof Of Concept). L'objectif principal est de démontrer comment faire une application de qualité, mais aussi :

- Utilisation de notre expertise web pour créer une application mobile, et proposer une alternative de réalisation à Cdiscount moins chère, plus efficace et prête à l'emploi (industrialisable)
- Démonstration de l'intérêt de l'utilisation des nouvelles technologies, et donc l'intérêt d'effectuer une veille technologique active
- Proposer une architecture moderne permettant l'isolation de la logique du domaine de l'application, et donc sa réutilisation dans un autre contexte, permettant le changement facile de technologie
- Cette architecture étant celle sur laquelle travaille l'équipe architecture, leur donner la possibilité d'utiliser notre projet comme preuve concrète de l'intérêt de celle-ci
- Intérêt d'avoir des petites équipes polyvalentes (pizzas team) sur des petits projets plutôt que de grosses équipes séparées par domaines techniques sur des gros projets

Mais finalement qu'est-ce qu'un logiciel de qualité ? C'est selon moi un logiciel possédant les caractéristiques suivantes :

- Développement rapide de fonctionnalités (temps = argent)
- Développement de nouvelles fonctionnalités sans impact sur l'existant (non-régression)
- Posséder une architecture évolutive permettant de limiter la dette technique et l'apparition des bugs
- Déploiements fréquents et rapides en production, permettant de satisfaire la demande du client
- S'assurer que l'application fonctionne toujours en production

Pour ce projet, nous nous proposons de reproduire le comportement de l'application Android actuelle, mais en utilisant :

- Technologies web uniquement
- Architecture bien choisie (en correspondance avec l'équipe architecture)
- Méthodologie agile

Nous avons une limite de deux mois, et nous sommes trois développeurs. À la fin des deux mois, nous présenterons le projet réalisé à un jury à la fois technique et non technique.

Créer une bonne application commence par choisir une bonne architecture logicielle. S'il y a bien

une chose à éviter, c'est le monolithe. Une application monolithique est une application mettant plusieurs fonctionnalités dans un seul processus. De nombreuses applications existantes sont monolithiques, car ce sont les plus simples à réaliser. Si cela convient aux petites applications, au fil du temps cette application deviendra de plus en plus complexe lors de son évolution. L'architecture modulaire prévue au début est alors difficile à garder.

« *De l'application simple à l'application à tout faire, il n'y a qu'un pas.* », Julien Dubreuil.

La base de code d'un monolithe est très importante et intimidante. Elle est difficile à comprendre et à modifier. Le moindre changement a un grand impact sur la totalité du code. De plus, il engage à long terme sur une technologie unique.

L'architecture logicielle ne se limite généralement pas à un seul style d'architecture, c'est très souvent une combinaison de plusieurs styles qui formera un système logiciel complet. Les principaux styles d'architecture sont :

Catégorie	Styles d'architectures
Communication	SOA (Service Oriented Architecture), Message Bus
Déploiement	Client/Serveur, N-tiers, 3-tiers
Domaine	DDD (Domain Driven Design)
Structure	Orienté composants, Orienté objet, Architecture en couches

Il existe un autre style, plus récent qui s'inspire grandement des autres, mais avec une approche différente au niveau de l'implémentation :

- SOA : Intègre différentes applications comme un ensemble de services
- Microservices : Architecture chaque application comme un ensemble de services

C'est le style que Cdiscount veut utiliser, et c'est celui que l'on va mettre en place pour notre projet, en plusieurs étapes, car sa mise en place est complexe.

Le DDD (Domain Driven Design) nous a appris qu'il existe en fait deux types de logique dans une application : logique de domaine et logique applicative. Le fait de les séparer permet une meilleure liberté technologique. C'est d'autant plus utile dans le domaine du web car les technologies évoluent très vite.

Pour notre projet, l'application client sera une SPA (Single Page Application) basée sur une architecture orientée composants couplée avec une architecture orientée événements. Le serveur sera un web service RESTful basé sur l'architecture des microservices.

Les choix technologiques sont nombreux, surtout pour le front-end car il existe plusieurs frameworks SPA. Nous avons choisi AngularJS pour sa maturité, combiné avec un bus d'événements fait maison pour respecter nos choix d'architecture (composants et événements). Nous avons aussi utilisé de nombreux outils, nous permettant d'être productif en évitant les répétitions (DRY : Don't Repeat Yourself).

Si Cdiscount utilise historiquement la méthode du cycle en V pour la gestion de projet, elle veut tendre vers des méthodes agiles. C'est pourquoi nous allons utiliser SCRUM combiné avec l'Extreme Programming (XP). Ils allient parfaitement gestion de projets et bonnes pratiques de développement. Ce que nous retenons vraiment des méthodes agiles sont :

- **Testabilité** : Un logiciel testé permet de s'assurer que toute nouvelle fonctionnalité n'affecte pas le reste du système. Le fait d'écrire le test en premier (TDD) force le refactoring constant et assure une bonne conception dès le départ, limitant la dette technique.
- **Code review** : La revue de code par un autre collègue permet de s'assurer de la qualité de celui-ci.
- **Jamais de dette technique** : La dette technique ralentit le développement et renforce la possibilité d'avoir des bugs dans le système.
- **Déploiements rapides, faciles, et fréquents**
- **Utiliser les bons outils**
- **Une bonne communication**
- **Bien-être des membres de l'équipe**

Voici notre méthodologie concrète employée pour ajouter une fonctionnalité au produit.

- Création d'une nouvelle branche sur le gestionnaire de versions (évite le conflit avec les autres nouvelles fonctionnalités implémentées en parallèle)
- Écriture de tests de haut niveau (Intégration ou UI selon le cas)
- Écriture du code source en TDD strict (il est interdit d'ajouter du code source sans la mise en valeur du manque par un test)
- Code review par un autre membre de l'équipe
- Intégration de la nouvelle branche sur la branche principale
- Vérification de la non-régression grâce à l'intégration continue

Ayant appliqué ces principes (architecture, méthodologies, technologies) sur notre projet durant 2 mois avec notre équipe de 3 développeurs, nous sommes arrivés à un résultat satisfaisant. Grâce à la technologie Cordova, nous avons créé une application fonctionnant sur plusieurs terminaux mobiles (iOS, Android) ainsi que sur les navigateurs web. Le résultat est selon nous satisfaisant, avec de bonnes performances, et une interface utilisateur simple et efficace.

Le projet de démonstration est un succès et a atteint ses objectifs. C'est un logiciel de qualité, permettant l'implémentation rapide de nouvelles fonctionnalités sans accumuler de dette technique. Nous avons par la même occasion démontré à Cdiscount qu'il est possible de créer des applications mobiles en utilisant uniquement les technologies web, notre domaine d'expertise. Lors de la présentation, le jury s'en est bien rendu compte et souhaite tester notre solution sur des applications transverses. Il faut avant cela, que nous finalisons l'intégration avec les méthodes actuelles de Cdiscount.

Le travail d'un architecte logiciel est la conception d'applications. Le travail d'un développeur de logiciel est la réalisation d'applications. Nous avons vu qu'il est difficile de séparer la phase de conception d'un projet de sa phase de réalisation. C'est le point faible de la méthodologie du cycle en V, et les méthodes agiles ont pour but de rallier ces deux phases. De nombreuses entreprises séparent ces deux rôles, et la qualité du logiciel est en amoindrie.

Un ingénieur logiciel est capable de concevoir ET réaliser une application. Le meilleur chemin pour avoir un logiciel de qualité est d'avoir une équipe d'ingénieurs logiciels, travaillant ensemble durant toutes les phases du projet (conception, réalisation et mise en oeuvre).

Abstract

There are two kind of software applications: desktop applications and web/mobile applications. The main difference is the device from which it is consumed, and by extension, the choice of the development technologies. Technological barriers between devices become thiner and thiner so the use of one software family for many devices is about to become the common way to do it. It allows to reduce development costs and also maintenance efforts. Because of its simplicity, the web is now the first target choice for every applications.

Cdiscount, the e-commerce french leader, operates on an international scale; hence, the quantity of "to be developed" applications goes increasing. Indeed, every country has its specific culture and so, features. Yet, Cdiscount contracts out its mobile software developments and thinks about getting it back to internal developments. However, the expertise domain of the company is not mobile developments but web developments. Would it not be possible for Cdiscount to use its expertise in order to create mobile applications through recent web technologies?

In the meantime, lots of project are launched to improve the Cdiscount Information System and so, to change both the internal architecture and the development methods. Cdiscount strongly encourages new applications to use agile methods and to implement microservices.

During my time at EPSI Bordeaux, I had the opportunity to get a 3 years co-op work job (from 2012 to 2015). First, my work consisted in software developments. As I became more experienced and as I gained the trust of my supervisors, my everyday job evolved. Besides software developpements, I had free time to make my own researches about new technologies and to improve my conception skills.

I was part of the "Lab team (R&D). This team was in charge of the innovation at Cdiscount. For this purpose, we always spend part of our time in technological monitoring so we would be able to propose new ideas, new ways to built up the tomorrow platform. That's why as a R&D team member, I felt free regarding my technological choices.

The World Wide Web grows strong and quick so it is really difficult to follow. Many technologies appear every single day. Making a web application combining recent technologies with the best technical choices and with a good quality level requires both strong abilities to analyse and an active technological monitoring.

What are the choices to make and the various steps to follow in order to conceive a robust and modern web application?

Our team applied to create a POC (Proof Of Concept) project. The objective was to demonstrate how to proceed to build a high quality application, but also:

- use our expertise in web development to create a mobile application and suggest a more cost effective, more efficient, and ready-to-use realization alternative for Cdiscount (using industrialization concepts)
- show the advantages of new technologies usages, and demonstrate the benefits of an active technological watch
- suggest a modern architecture that isolates contextual logic from the application itself, allowing code reuse within other contexts, allowing easy technological shifts
- This technology, being used by the Architecture team, shows them the concrete proof of the advantages to use our project
- The advantages of having small polyvalent teams (pizza teams) working on small projects, instead of having large teams working on separate technical subsets

But in the end, what is a high quality software? To me, it's a piece of software having the following qualities:

- fast feature development (time is money),
- feature development that does not cause regressions on existing ones,
- has an architecture that allows new features to be built while containing bug count and technical debt growth,
- fast build, frequent and easy production deployments, allowing customer satisfaction
- Make sure the software always works for the end-user

For this project, we suggest to reproduce the android application behaviour, but we will use:

- only web technologies,
- a well-chosen architecture (accordingly to the Architecture team)
- Agile methodology

We have a two months limit and are three developers. By the end of the two months period, we will showcase our project in front of a jury composed by "technical" and "non-technical" people. Create a good application starts by choosing the right software architecture. If there is one thing to avoid at all costs, it is the monolithic application.

A monolithic application is an application putting together lots of features into one single process. Several existing applications are monolithic ones because they are the easiest one in term of realisation. In the early stages, this approach could be relevant for the smallest applications, but those kind of applications are condemned to become really difficult to maintain. The modular

architecture previously designed cannot be maintained anymore.

"*From the simple application to the can-do-everything application, there is only one step.*", Julien Dubreuil.

The basis of a monolithic code is very important and intimidating. It is difficult to understand and to modify. Any single change has a great impact on all the code. Moreover, it is engaging in long-term on a unique technology.

Software architecture is not limited to one type of architecture, it is often a combination of several types that forms a complete software system. The main types of architecture are:

Catégorie	Styles d'architectures
Communication	SOA (Service Oriented Architecture), Message Bus
Deployment	Client/Serveur, N-tiers, 3-tiers
Domain	DDD (Domain Driven Design)
Structure	Component oriented, object oriented, Layered architecture

There is another type of architecture, more recent, which is inspired from others, but with a different approach on implementation:

- SOA: includes different applications like a set of services
- Microservices: Design each application like a set of services

It is the type that Cdiscount wants to use, and it is the one that we are going to put in place for our project, in several phases, because its production is complicated.

The DDD (Domain Driven Design) made us aware that there are two types of logic in an application: domain logic and applicative logic. Separating the two of them allows a better technological freedom. It even more useful in the web field because technologies are growing at a fast pace.

For our project, the client application will be a SPA (Single Page Application), based on a component-oriented architecture with an event-oriented architecture. The server will be a RESTful web service based on the architecture of microservices.

There are many technological choices, especially for front-end development because there are many SPA frameworks. We chose AngularJS because of its maturity, combined with a home-made bus of events to respect our architecture choices (components and events). We also used many tools, allowing us to be productive and avoiding repetitions (DRY: Don't Repeat Yourself).

Cdiscount uses historically the V-cycle methodology of project management and wants to tend to agile methodologies. That's why we are going to use SCRUM combined with Extreme Programming (XP). They are perfectly combining project management and good practices of development. We can sum up agile methodologies by:

- **Testability**: a tested software allows to ensure that any new functionality is not affecting the system. Writing the test first (TDD) reinforces constant refactoring and ensures a good conception from the start, limiting the technological debt.
- **Code review**: allows the assurance of a good quality code
- **No technical debt**: the technical debt slows down the development and reinforces the possibility to have bugs in the system
- **Quick, easy, frequent deployments**
- **The use of good tools**
- **Good communication**
- **Well-being of team members**

Here is our concrete methodology to add functionalities to the product:

- Creation of a new branch on the version manager (avoiding conflict with other new implemented functionalities)
- Writing tests of high level (Integration or UI)
- Writing source code in strict TDD (it is forbidden to add source code without a test)
- Code review by another team member
- Integration of the new branch on the principal one
- Checking of the non-regression thanks to continuous integration

Because we applied those principles (architecture, methodologies, technologies) on our project for two months with a team of three developers, we now have a satisfying result. Thanks to Cordova technology, we created an application that works on different mobile devices (iOS, Android) and on web browsers. The result is, in our opinion, satisfying, with good performances, and with a simple and effective user interface.

The demonstration project is a success and its objectives were achieved. This well-designed software allows a quick implementation of new functionalities with no place for technical debt. In the meantime, we also proved that it is possible to create mobile applications within our expertise field, web applications. Now the jury wants to test our solution on transversal applications. However, beforehand we need to finalize the integration with the Cdiscount current methods.

A software architect job is the conception for applications. A software developer job is the

execution of applications. We saw that it is difficult to separate the conception phase from the execution phase. It is the weak spot of the V-cycle methodology, and agile methodologies exist to bound those two phases together. Many firms are separating these two roles and the software quality diminishes in consequence.

A software engineer is capable of conceive AND execute an application. The best way to have a good quality software is to have a team of software engineers, who work together during all phases of a project (conception, execution and production).

Introduction

Lors de mon cursus d'ingénierie informatique à l'EPSI Bordeaux, j'ai eu l'occasion de travailler en alternance durant 3 ans à Cdiscount (de 2012 à 2015). Leader dans le domaine du e-commerce en France, Cdiscount est une entreprise qui se doit d'être à la pointe de la technologie. Durant ces 3 ans, j'ai pu intégrer plusieurs équipes et participer à de nombreux projets très intéressants, ce qui m'a permis de m'améliorer dans de nombreux domaines et de devenir un expert en technologies web.

Mon travail était tout d'abord essentiellement le développement d'applications. Gagnant de l'expérience et de la confiance auprès de mes supérieurs, mon travail de tous les jours a progressivement évolué. En plus du développement, j'ai eu du temps dégagé pour faire de la recherche sur les nouvelles technologies, et de parfaire mes talents de conception.

Le monde du web évolue très vite et est difficile à suivre. De nombreuses technologies apparaissent régulièrement. Faire une application web moderne, et de qualité n'est pas un travail facile. Trouver la solution idéale et faire les bons choix techniques selon le projet et le style de développement de l'équipe demande une bonne compétence d'analyse et une veille technologique active.

Cdiscount sous-traite actuellement le développement de ses applications mobiles. Le nombre d'applications qu'elle possède ne cesse de grandir et le prix de développement devient de plus en plus conséquent. L'expertise technique de la société étant le web, notre équipe propose de créer un projet de type POC (Proof Of Concept) pour prouver qu'il est possible d'utiliser note expertise pour créer des applications mobiles et limiter les coûts.

En parallèle, de nombreux chantiers sont en cours pour rénover le fonctionnement du SI (Système d'Information) de Cdiscount et changer l'architecture interne ainsi que les méthodes de développement utilisées. Elle encourage fortement les nouvelles applications à utiliser les méthodes agiles et à implémenter l'architecture sur laquelle elle travaille (microservices). Le projet que nous allons réaliser pourra servir d'exemple.

Quels sont les choix à effectuer et quels sont les étapes à suivre pour la conception et la réalisation d'une application web moderne de qualité ?

Ce document définira ce qu'est une application web et ce qu'est un logiciel de qualité. Nous suivrons ensuite le cheminement de création du projet qui servira de support (POC).

Présentation de l'entreprise

1. Une entreprise grandissante

CDiscount est une entreprise d'e-commerce qui gère le site web CDiscount.com. Ses employés ainsi que ses entrepôts sont situés à Bordeaux. Elle emploie environ 1500 personnes et environ 300 personnes sont dans la partie SI. Elle a été créée en 1998 par trois frères et vendait principalement des CD. Elle vend aujourd'hui toutes sortes de marchandises (High tech, loisirs, équipements de maison, etc.).

Elle est aujourd'hui filiale à 99,6% du Groupe Casino qui est un grand groupe de distribution en France et dans le monde. Ceci lui a permis de profiter de différents avantages comme la livraison dans les magasins Casino. En plus de cela, Cdiscount fait partie désormais du grand groupe Cnova, grosse multinationale qui fait passer CDiscount parmis les 1er acteurs mondiaux[10]. Le site s'est alors imposé comme l'un des principaux sites de vente en ligne français avec un chiffre d'affaires de 1,6 milliard d'euros TTC. Elle passe ainsi en France devant Amazon, le leader mondial dans ce domaine. Cdiscount a profité aussi du groupe Casino pour s'implémenter de plus en plus à l'international. Elle s'est récemment implémentée en Colombie, en Côte d'Ivoire, en Thaïlande, puis bientôt au Brésil ou encore au Panama pour profiter du e-commerce fleurissant de ces pays.

Pour s'adapter à ces changements, Cdiscount retravaille sa méthodologie et son architecture. Son objectif est d'accentuer la qualité du code fourni et de miser sur l'innovation technologique. Si historiquement, les technologies principales utilisées chez Cdiscount proviennent du monde Microsoft, elle utilise de plus en plus sur des alternatives open source.

2. Différent types d'équipes

Le SI de Cdiscount est historiquement divisé en équipes par domaine technique.

A. Équipes par domaine

- **Front**

C’est l’équipe qui s’occupe de l’interface du site. La logique est implémentée dans les autres équipes consommés via des *web services*.

- **DBA**

DBA signifie DataBase Administrator (administrateur de base de données). Cette équipe s’assure de la qualité et du bon fonctionnement des bases de données.

- **Delivery Management**

Ils s’occupent de la gestion des différents serveurs à distance et des déploiements d’applications d’un environnement à un autre. Ils s’assurent aussi du maintien à jour et du bon fonctionnement de celles-ci (fichiers de configuration, etc.).

- **Sécurité**

Cette équipe est chargée de s’assurer de la sécurité du site et des différents développements réalisés, ainsi que différentes failles possibles liées au réseau.

- **Architecture**

Ce sont ceux qui font l’architecture globale du SI. Ils font ainsi les choix de technologie et de conception. Ils s’occupent aussi du framework de Cdiscount.

- **Fonctionnel/Métier**

Ils sont l’incarnation du besoin auprès des équipes du SI. En agile (comme le montrera le chapitre à cet effet, il ont souvent le rôle de responsable du produit ou “Product Owner”. Souvent, ce sont aussi eux qui répondent du budget à appliquer aux projets.

- **Big Data**

C’est une équipe dédiée à la récolte ainsi que l’analyse des données récoltées en masse sur le site Cdiscount. Ils utilisent ces données pour réaliser des rapports hebdomadaires/mensuels sur différents sujets. Une autre partie de l’équipe s’occupe de trier, de quantifier et de donner de la valeur aux données, se sont les « data scientists ».

- **Services**

C’est l’équipe qui intègre toutes les fonctionnalités dites « métier » du site sous forme de services web développées généralement pour l’équipe Front mais aussi pour d’autres équipes ayant besoin de fonctionnalités internes (exemple : back-office). L’idée de l’équipe est donc de fournir des briques métiers aux autres équipes de SI. Ces briques régissent les règles métiers mais fournissent également des services plus techniques voir utilitaires.

- **Performances**

Depuis 4 ans, CDiscount a fait un virage a 180° sur le monitoring de la performance. Ce sont absolument tous les indicateurs qui permettent d'améliorer l'expérience client mais aussi le référencement (intimement lié a la performance)

- **Lab : R&D**

C'est l'équipe chargée de maintenir l'innovation à Cdiscount. Pour cela, elle est en constante veille technologique et propose de nouvelles idées de technologies au SI. Leur objectif est de maintenir à jour les technologies utilisées, et donc de construire la plateforme de demain. Elle a le champ libre sur le choix des technologies.

B. Équipes par fonctionnalité

Un nouveau genre d'équipes fait son apparition au sein du SI de Cdiscount : les "feature teams". On les nomme aussi parfois les "pizza teams". Ce sont des équipes polyvalentes ne dépassant pas 8 personnes assignées à une seule fonctionnalité. Les méthodes agiles sont plus faciles à appliquer avec ce genre d'équipes. Il existe une équipe par pays et certaines équipes s'occupent d'une certaine fonctionnalité, comme celles-ci :

- **Market Place**

C'est l'équipe en charge de la place de marché de Cdiscount, c'est-à-dire les autres vendeurs que Cdiscount, qui vendent leurs produits à travers du site cdiscount.com.

- **Moteur de Recherche**

C'est l'équipe en charge du moteur de recherche.

- **Order**

C'est l'équipe en charge de la prise de commande.

3. Implication personnelle dans l'entreprise

Ayant effectué 3 ans en alternance dans cette entreprise, j'ai eu l'occasion d'intégrer plusieurs équipes. J'ai d'abord travaillé dans l'équipe services puis dans l'équipe Lab (R&D). J'ai alors participé à de nombreux projets, me permettant de comprendre le fonctionnement du SI de Cdiscount.

A. Projets

Membre de l'équipe services, les projets étaient en rapport direct avec le site cdiscount.com, mais après avoir intégré l'équipe Lab, les projets étaient beaucoup plus transverses et variés.

1. WSManager

C'est le projet sur lequel j'ai clairement passé le plus de temps. Son objectif est de faciliter l'appel aux différents web service de Cdiscount. L'équipe services est en charge de nombreux web services de types différents sur des environnements différents (production, préproduction, développement, etc.). Les outils existants n'ont pas une prise en main facile et ne proposent pas un listing partagé des services existants à Cdiscount, accessibles à tous les développeurs. WSManager est donc un outil de gestion collaborative de web services.

2. OpenAPI

Cdiscount propose depuis peu une API ouverte donnant un accès au catalogue de Cdiscount via web service. J'ai eu l'occasion de participer à ce projet et suis à l'origine du portail développeur, accessible à l'adresse http ://dev.cdiscount.com. Il contient une documentation interactive de l'openAPI ainsi que la gestion des comptes développeurs.

3. POC

C'est le projet servant de support pour ce document. Il sera décrit par la suite.

B. Veille technologique et formations

En plus des projets cités ci-dessus, mon rôle au sein de l'équipe Lab était d'effectuer une veille technologique active et d'organiser des ateliers de partage de connaissance sur le sujet sous forme de coding dojo. Le projet WSManager me servait de support direct pour appliquer mes recherches. Elles ont été très utiles pour la mise en place du POC.

I

Application Web

Il existe 3 grandes familles d'applications informatique (logiciels) :

- **Application de bureau**

« *Une application de bureau (Desktop application en anglais) est un logiciel applicatif qui affiche son interface graphique dans un environnement de bureau, il est hébergé et exécuté par l'ordinateur de l'usager. Cette technologie est apparue avec les premiers environnements de bureau en 1970* », définition de Wikipédia.

- **Application Web**

« *Une application web (aussi appelée web app, de l'anglais) est une application manipulable grâce à un navigateur web* », définition de Wikipédia.

- **Application mobile**

« *Une application mobile est un logiciel applicatif développé pour un appareil électronique mobile, tel qu'un assistant personnel, un téléphone portable, un « smartphone », un baladeur numérique, une tablette tactile* », définition de Wikipédia.

Ce qui différencie finalement ces 3 types d'applications est le support sur lequel elles sont accessibles. Ceci implique généralement des technologies et langages différents. Les avancées récentes des navigateurs web ont fait des applications web des applications complètes et avancées en utilisant uniquement les technologies web (centrées généralement autour du JavaScript).

La facilité de développement de celles-ci ainsi que la performance des moteurs de navigateurs ont propulsé les applications web à un tout autre niveau, dépassant même les limites de la plateforme. En effet, il est désormais possible de faire des applications natives mobiles ou de bureau en n'utilisant que des technologies web approchant les performances des applications natives. Cela permet d'avoir un coût de développement moindre grâce à la réutilisation de code pour les différentes plateformes. Le web est désormais la cible de toutes les applications.

1. Description

Une application web, contrairement à une application de bureau, est une application uniquement manipulable via un navigateur web. L'application est généralement installée sur un serveur et est accessible via un réseau (internet, réseau local, etc.). On a ici une architecture client/serveur où le navigateur web est le client. À la différence d'un site web statique, une application web est constituée de liens hypertextes provoquant l'envoi de nouvelles requêtes utilisant généralement le protocole HTTP. Voici des exemples d'applications web les plus courantes :

- Moteur de recherche : Application qui recherche des documents (Ex : Google, Bing, Yahoo, etc.)
- WebMail : Application qui permet d'envoyer et de recevoir des courriers électroniques (Ex : Gmail, Outlook, etc.)
- Système de gestion de contenu : Application qui présente des documents
  - Blog : Système de gestion de contenu organisé par date
  - E-commerce : Site marchand (Ex : Amazon, Cdiscount, etc.)
  - Wiki : Création collaborative de documents (Ex : Wikipédia)
- Messagerie instantanée : Application qui permet à plusieurs utilisateurs d'échanger des messages textuels entre eux (Ex : Skype, Facebook messenger, etc.)
- Jeu : Jeu vidéo accessible via le navigateur web
- Réseau social : Application permettant de partager du contenu avec d'autres personnes (Ex : Facebook, Linkedin, etc.)

Il existe de nombreux autres types d'applications, comme par exemple Google Maps qui permet de consulter des cartes géographiques du monde entier.

2. JavaScript, le langage du web

Aujourd'hui, les applications web dépendent énormément (si ce n'est uniquement) du langage de programmation JavaScript, mais cela n'a pas toujours été le cas.

A. Attributs du langage

JavaScript, aussi appelé ECMAScript est un langage de scripts créé en 1995 par Brendan Eich. C'est un langage de programmation dit dynamique ou interprété, c'est-à-dire langage qui exé-

cute au lancement du programme beaucoup de comportements que les langages dits statiques exécutent durant la compilation.

Contrairement à ce que l'on pourrait croire, le JavaScript n'a rien en commun avec le Java. En effet, ces deux langages possèdent des sémantiques et des attributs très différents. Le JavaScript est un langage *orienté prototype* qui utilise des *fonctions de première classe*. Ce mélange de caractéristiques permet au JavaScript d'être un langage multi-paradigme permettant plusieurs styles de programmation : orienté objet, fonctionnel, impératif, etc. Une des caractéristiques phare de ce langage est la possibilité de faire de la programmation dite événementielle ou asynchrone grâce à la notion de callback (fonction de rappel).

On dit souvent que le JavaScript est un langage incompris. Il ressemble beaucoup au Java, mais en est très différent. L'utilisation abusive de de ces fonctions complexifie le code. On parle même de "l'enfer du callback" (*callback hell*).

B. Des alternatives viables mais pas majoritairement adoptées

Pour combler les manques du langage et ses incompréhensions, de nombreux sur-langages sont apparus. Ce sont des langages de plus haut-niveau permettant soit la possibilité d'être directement interprété par le navigateur, soit la possibilité de se transcrire en JavaScript. Les sur-langages les plus utilisés sont les suivants :

- **CoffeeScript** : Créé en 2009, inspiré par le Python, Ruby, et le Haskell
- **TypeScript** : Créé par Microsoft en 2012, ajoute la possibilité d'ajouter des variables fortement typées et d'utiliser des classes pour un orienté objet plus "classique"
- **Dart** : Créé par Google en 2013, ajoute de nombreuses fonctionnalités comme des classes, des interfaces, des classes abstraites, des variables fortement typées, etc. Possède aussi un interpréteur directement intégré au navigateur Google Chrome. Le JavaScript est utilisé pour les autres navigateurs

Ces surcouches ont toutes le même but : ajouter des fonctionnalités, faciliter le développement web, et apporter des évolutions qui apparaîtront plus vite que les nouveaux standards ECMAScript grâce au support de la communauté. Cependant, aucun d'eux n'a réellement pu émerger pour devenir la solution idéale pour créer des applications web. Actuellement, elles ont plus le statut d'alternatives viables mais pas majoritairement adoptées.

C. JavaScript, actuellement la seule solution

Malgré les avantages que proposent ces nouveaux langages, toutes les équipes de développement ne l'adoptent pas forcément. Beaucoup de développeurs connaissent déjà très bien le JavaScript et ne veulent pas s'en défaire pour apprendre un nouveau langage. Le fait qu'il existe de nombreuses surcouches montre bien qu'il est difficile de n'avoir qu'un seul standard de développement satisfaisant tous les développeurs. Il est cependant agréable d'avoir le choix.

Le JavaScript a pris une place trop importante dans le monde du Web pour être remplacé. Voici une citation traduite de Brendan Eich, le créateur du langage JavaScript lors d'une interview le 17 Juin 2015 :

« *On ne casse pas le web, on ne peut pas tout effacer et recommencer. Toute personne essayant échouera* »[14]

Selon lui, il n'est pas possible refaire le web, la seule solution est de l'améliorer ou d'ajouter des alternatives.

D. Une version 6 bien évoluée

On dit souvent que c'est que le JavaScript est à la fois adoré et détesté[13]. C'est un langage très puissant mais souvent mal compris et mal utilisé car très permissif. Malgré une évolution plus lente que les autres alternatives open source, la récente version 6 (on parle d'ECMAScript 6, abrégé ES6, ou encore d'ECMAScript2015) fait de celui-ci un langage de plus en plus viable et comble les différents manques et incompréhensions citées auparavant.

Même si aujourd'hui, cette version n'est pas prise en charge par tous les navigateurs, il est possible d'écrire du code en ES6, et le transformer en ES5 grâce à des transpileurs (transforme un code source en un équivalent dans un autre langage). Le code écrit est alors plus moderne et fonctionne malgré tout sur les différents navigateurs dès aujourd'hui. Le but est d'utiliser au plus tôt les derniers standards et avoir une application plus facile à maintenir et profiter rapidement des dernières évolutions.

E. Une utilisation étendue

Comme vu précédemment, le JavaScript évolue énormément. Ses performances, sa facilite de programmation, et tous les avantages que nous avons vu précédemment ont fait apparaître des projets indépendants des navigateurs web.

Un projet phare est Node.js, projet open source créé en 2009. C'est un environnement multiplateforme orienté vers les applications réseaux. Node.js tire parti de la machine virtuelle V8 et de la programmation événementielle de JavaScript pour créer des serveurs web ultra performants pouvant tenir une charge importante, sans avoir besoin d'un logiciel externe comme Apache ou Nginx. Cette solution est maintenant utilisée par les plus grandes entreprises (Groupon, SAP, LinkedIn, Microsoft, Yahoo !, Walmart, etc) et possède une communauté très active. Paypal est passe d'une solution entièrement écrite en Java à une solution totalement écrite en Node.js.

Le gestionnaire de paquets de Node.js (npm) est celui possédant le plus grand nombre de module (plus de 80 000 en 2014), faisant de Node.js le langage le plus actif, détrônant ainsi Java et son gestionnaire de paquets Maven.

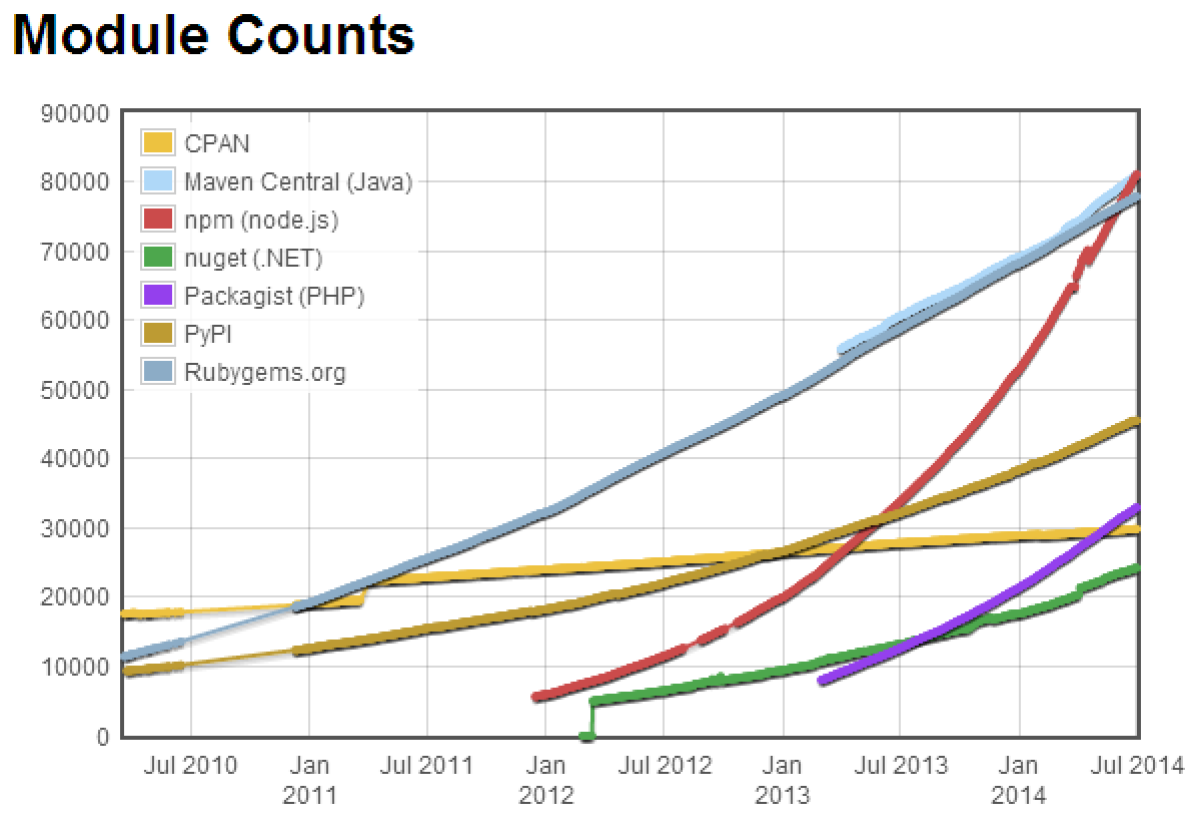

Figure I.1 : Nombre de modules pour chaque langage

La popularité de Node.js et des applications web ont fait grimper en flèche l'utilisation du JavaScript. Il est désormais le langage le plus actif et le plus populaire ces dernières années. Ainsi, en plus de posséder le plus grand nombre de modules, le langage JavaScript contient le plus grand nombre de dépôts actifs sur Github.

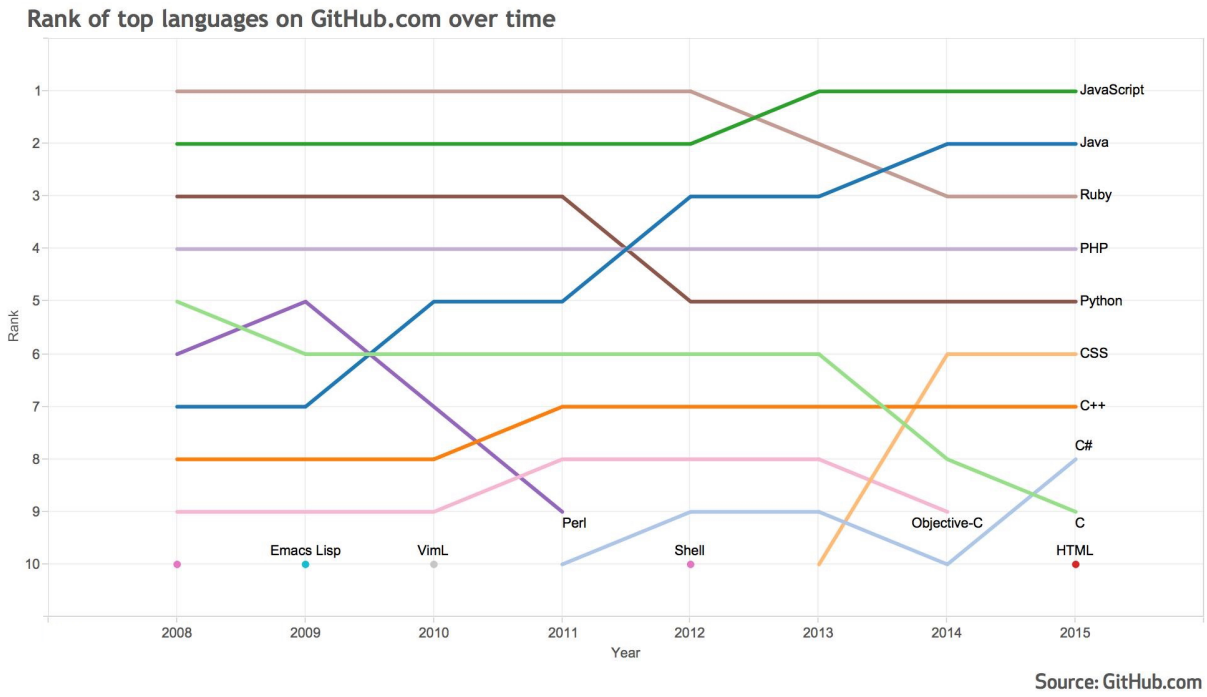

Figure I.2 : Statistiques des langages de programmation sur Github

3. Navigateurs web

Une application Web utilise un ou plusieurs langages de présentation et est interprétée par un navigateur web. Le langage JavaScript, lui aussi interprété par le navigateur, permet de rendre encore plus dynamique l'application en permettant des interactions utilisateurs de plus en plus poussées et rapides. La qualité de l'application repose alors énormément sur la performance et la précision du navigateur.

Un navigateur web est composé d'un moteur de rendu (*layout engine*) et d'un moteur JavaScript (*JavaScript engine*). Le moteur de rendu interprète les langages de présentation comme HTML/CSS, tandis que le moteur JavaScript interprète de langage de programmation JavaScript.

Avant standardisation, les navigateurs avaient tout d'abord leur implémentation individuelle du DOM. Le DOM (Document Object Model) est maintenant un standard du W3C (organisme de normalisation à but non lucratif) qui décrit une interface indépendante de tout langage de programmation et de toute plate-forme, permettant à des scripts d'accéder ou de mettre à jour le contenu, la structure ou le style de documents HTML et XML.

4. Mobile

Si la navigation web a été très longtemps limitée aux ordinateurs personnels, la navigation mobile est de plus en plus forte. En 2014, le nombre d'internautes utilisant un terminal mobile a dépassé celui des ordinateurs personnels.

Il existe en réalité deux types de marché sur le mobile : le mobile natif, et le mobile web. Nous allons voir la différence entre les deux et ce que cela implique dans le développement de celles-ci.

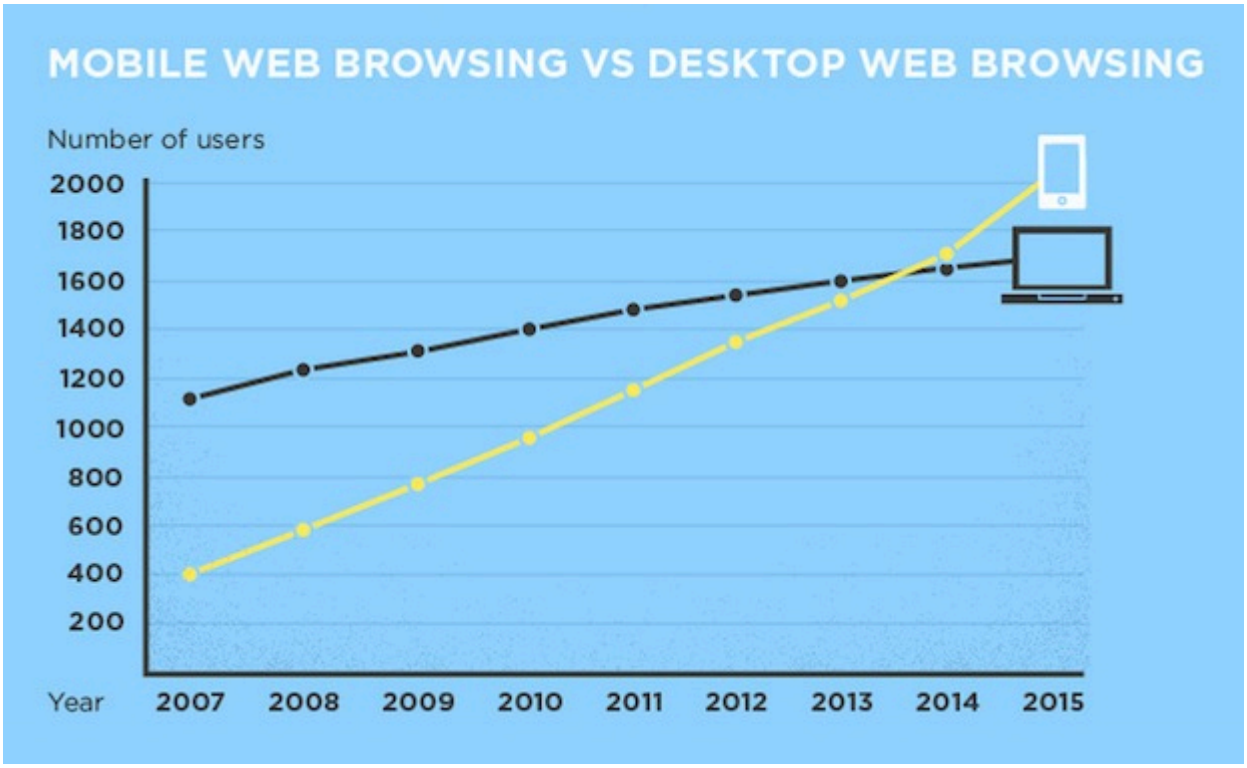

Figure I.3 : Nombre d'internautes mobile et PC

A. Mobile Web

Le mobile web correspond aux applications utilisables uniquement via les navigateurs web installés sur les terminaux mobiles. Les technologies utilisées sont donc les mêmes que les navigateurs de bureau (JavaScript, HTML/CSS, etc.). Elles permettent désormais une interaction et un affichage sur écrans de terminaux mobiles biens adaptés.

De nos jours, il est donc nécessaire qu'un site web soit disponible plusieurs types de navigation (mobile et PC). Le responsive web design facilite le développement et l'adaptation des écrans.

L'application n'étant disponible que via les navigateurs, elle n'est pas installable sur le terminal et l'utilisateur doit être connecté à Internet pour l'utiliser. De plus, l'accès aux ressources du téléphone sont plus limitées.

B. Mobile Natif

Le mobile natif correspond aux applications pour lesquelles il est possible d'installer sur le terminal. L'application a un accès complet aux ressources du téléphones via le système d'exploitation (GPS, appareil photo, notifications, etc.). Les performances sont alors similaires à celles des applications de bureau.

Visuellement, ce qui différencie généralement ces applications, c'est qu'elles utilisent la même charte graphique (*look & feel*) définie par le système d'exploitation. L'utilisateur se retrouve généralement mieux si les contrôles sont similaires entre les applications.

Au niveau du développement, le langage utilisé est défini par le système d'exploitation. Il est donc différent selon le type de terminal utilisé (Java pour Android, et Objective-C pour iOS par exemple). La création d'une application native sur les principaux terminaux du marché possède un coût élevé. Maintenir une application mobile native pour chaque terminal, consiste finalement à maintenir plusieurs applications avec des technologies et chartes graphiques différentes est complexe.

C. Mobile hybride

Le mobile hybride est une approche récente alliant les deux mondes. L'idée est de créer des applications mobiles sur plusieurs terminaux en utilisant uniquement les technologies du web, qu'importe le système d'exploitation. De nombreux outils sont disponibles et les coûts de développement sont grandement diminués.

Cela est possible grâce à la technologie des *webviews*. C'est en fait un navigateur intégré dans une application native. Les principaux systèmes d'exploitation du marché possèdent cette technologie. L'application est donc exactement la même sur tous les terminaux et un seul code est utilisé. Elles peuvent en plus de cela, utiliser les fonctionnalités natives du téléphone comme l'appareil photo, le GPS, etc.

Les seuls inconvénients sont que le *look & feel* sera similaire sur tous les terminaux, utilisant beaucoup moins la charte graphique native du téléphone. Les performances sont aussi plus faibles qu'une application native. Cela ne se ressent que dans certains cas où les performances

sont critiques (jeux vidéos, applications en temps réel, etc.).

C'est finalement une alternative moins coûteuse pour créer des applications mobiles natives. Selon les cas d'utilisation, elle est fortement envisageable.

5. Projet de démonstration : POC

Le projet décrit dans cette partie sera le projet utilisé comme support et démonstration de ce mémoire. C'est un projet réalisé avec mon équipe de type POC (Proof Of Concept).

A. Contexte et motivations

1. Coûts élevés

Cdiscount possède actuellement un site web responsive, une application mobile native sur les deux principaux systèmes d'exploitation du marché (Android et iOS). La société a préféré faire sous-traiter le développement de ces deux dernières par une autre entreprise plutôt que de recruter et former des personnes sachant développer du natif pour chacun d'entre eux.

La société étant présente de plus en plus à l'international, elle possède de plus en plus d'applications et les frais de sous-traitance sont élevés. L'expertise technique de la société est le web. Ne serait-il pas mieux pour elle d'utiliser ces compétences lorsque cela est possible ?

2. Recherche de meilleure méthode et architecture

Cdiscount possède une grande base de code accumulée avec le temps. Ayant gardé la même architecture et la même méthodologie pendant très longtemps, elle a accumulé une dette technique importante. La société se rend compte qu'il lui faut de plus en plus de temps pour créer ajouter des fonctionnalités, et est actuellement en quête de meilleure architecture et méthodologie.

Si Cdiscount utilise historiquement la méthodologie du cycle en V, elle essaye de plus en plus de tendre vers des méthodes agiles. C'est pour cette raison que de plus en plus de pizza teams sont mises en place. Pour les nouvelles applications, elle préfère cette méthode, et en profite pour mettre en place la nouvelle architecture sur laquelle elle travaille.

3. Utilisation de l'expertise de notre équipe

Notre équipe est spécialisée dans la veille technologique et dans la mise en place de nouvelles technologies. Nous avons mis en place plusieurs applications internes en utilisant au maximum les dernières technologies avec une architecture solide. L'équipe architecture travaille actuellement sur la prochaine architecture qu'elle va mettre en place dans le SI, et est à la recherche de nouvelles applications pour l'appliquer. L'avantage est que nous l'appliquions déjà dans nos précédentes applications.

Ainsi, en utilisant notre expertise et en collaboration avec l'équipe architecture, nous avons proposé de mettre en place un projet de type POC pour prouver l'intérêt de cette nouvelle architecture combinée avec l'utilisation des nouvelles technologies.

B. Description

Nous nous proposons de reproduire le comportement de l'application Android actuelle, mais en utilisant :

- Architecture bien choisie (en correspondance avec l'équipe architecture)
- Technologies web uniquement
- Méthodologie agile

Nous avons une limite de deux mois, et nous sommes trois développeurs. À la fin des deux mois, nous présenterons le projet réalisé à un jury à la fois technique et non technique. Même si le but de cette application est plutôt orienté vers la technique, il faudra qu'elle soit agréable à utiliser et que le jury non technique soit convaincu que ce soit une solution viable.

C. Objectifs

1. Principal objectif

Le principal objectif de ce projet est de démontrer comment faire une application de qualité. Mais finalement qu'est-ce qu'un logiciel de qualité ? C'est selon moi un logiciel possédant les caractéristiques suivantes :

- Développement rapide de fonctionnalités (temps = argent)
- Développement de nouvelles fonctionnalités sans impact sur l'existant (non régression)

- Posséder une architecture évolutive permettant de limiter la dette technique et l'apparition des bugs
- Déploiements fréquents et rapides en production, permettant de satisfaire la demande du client
- S'assurer que l'application fonctionne toujours en production

2. Autres objectifs

En plus de vouloir créer un logiciel de qualité, ce projet a aussi pour objectifs :

- Utilisation de notre expertise web pour créer une application mobile, et proposer une alternative de réalisation à Cdiscount moins chère, plus efficace et prête à l'emploi (industrialisable)
- Démonstration de l'intérêt de l'utilisation des nouvelles technologies, et donc l'intérêt d'effectuer une veille technologique active
- Proposer une architecture moderne permettant l'isolation de la logique du domaine de l'application, et donc sa réutilisation dans un autre contexte, permettant le changement facile de technologie
- Cette architecture étant celle sur laquelle travaille l'équipe architecture, leur donner la possibilité d'utiliser notre projet comme preuve concrète de l'intérêt de celle-ci
- Intérêt d'avoir des petites équipes polyvalentes (pizza teams) sur des petits projets plutôt que de grosses équipes séparées par capacité logicielles sur des gros projets

Ce projet possède de nombreux objectifs en temps limité. Si CDiscount valide la présentation du projet, nous pourrons continuer à développer cette solution. Ce projet va dans la même direction de l'évolution du SI de la société et souhaitons qu'il devienne une inspiration pour celui-ci. Nous ne prétendons pas vouloir remplacer les applications mobiles actuelles, mais proposer une nouvelle solution moins chère et peut-être plus adaptée dans certains cas, tout en utilisant notre expertise interne.

II

Choix d'architecture

Créer une bonne application commence par choisir une bonne architecture logicielle. Ceci est même plus important que de choisir une bonne technologie, car il ne faut pas devenir dépendant de celle-ci. En effet, les technologies évoluent rapidement, et il n'est pas rare de vouloir profiter du changement de technologie, surtout dans le monde du web. Ainsi, il est nécessaire d'avoir une bonne architecture permettant cela, en plus de permettre de créer un logiciel de qualité.

1. Le monolithe

Une application monolithique est une application mettant plusieurs fonctionnalités dans un seul processus. De nombreuses applications existantes sont monolithiques, car ce sont les plus simples à réaliser. Si cela convient aux petites applications, au fil du temps cette application deviendra de plus en plus complexe lors de son évolution. L'architecture modulaire prévue au début est alors difficile à garder.[12]

« *De l'application simple à l'application à tout faire, il n'y a qu'un pas.* », Julien Dubreuil. [6]

Les inconvénients du monolithe sont :

- **Développement ralenti** : La large base de code intimide les développeurs, surtout les nouveaux. Il est difficile à comprendre et à modifier. De plus, il alourdit énormément l'environnement de développement (IDE) et est plus lourd à charger
- **Petit changement = Grand impact** : Chaque changement demande une phase de compilation, tests, et déploiement complète. Cela devient un obstacle pour les changements et déploiements fréquents
- **Gestion de l'échec** : Si une partie du monolithe échoue, le monolithe en entier échoue
- **Engagement à long terme sur une technologie** : Le monolithe est réalisé dans une seule technologie. Il est difficile d'en changer, et la réécriture partielle est impossible. Il est intéressant d'exploiter les capacités de différents langages pour des buts spécifiques, comme par exemple R pour faire du calcul statistique ou Node.js pour du temps réel

Ces inconvénients se font de plus en plus ressentir au fil du temps et ont un grand impact sur la productivité des développeurs. Cdiscount possède plusieurs applications monolithiques dans

son SI et s'en rend bien compte. Les choix de styles d'architecture peuvent amener ou éviter le monolithe.

2. Différents styles d'architecture

Il existe de nombreux styles d'architecture. Il n'y a pas vraiment de bon ou mauvais style d'architecture, juste des avantages et des inconvénients. Il faut bien les comprendre et choisir correctement en fonction du besoin. Nous n'allons pas tous les décrire dans ce mémoire, mais juste ceux pris en considération pour le projet (ou pour toute autre application web que l'on voudrait créer actuellement) et aussi ceux utilisés actuellement à Cdiscount.

A. Description

Mais tout d'abord qu'est-ce qu'un style architectural ? Voici une définition de David Garlan et Mary Shaw dans leur livre "An Introduction to Software Architecture" traduite :

« *… famille de systemes en termes d'un modèle d'organisation structurelle. Plus specifiquement, un style d'architecture determine le vocabulaire des composants et connecteurs qui peuvent etre utilises dans des cas de ce style, avec un ensemble de contraintes sur la façon dont ils peuvent être combinés. Ceux-ci peuvent inclure des contraintes topologiques sur descriptions architecturales (par exemple, pas de cycles).* » [9]

Voici un tableau comprenant les principaux styles d'architecture logicielles classés par catégorie. [2]

Catégorie	Styles d'architectures
Communication	SOA (Service Oriented Architecture), Message Bus
Déploiement	Client/Serveur, N-tiers, 3-tiers
Domaine	DDD (Domain Driven Design)
Structure	Orienté composants, Orienté objet, Architecture en couches

Il existe un autre style, plus récent, que l'on décrira en dernier, qui se nomme les Microservices. Comme en architecture traditionnelle, c'est souvent par le mélange d'anciens styles que de nouveaux apparaissent. L'architecture logicielle ne se limite généralement pas à un seul style

d'architecture, c'est très souvent une combinaisons de plusieurs styles qui formeront un système logiciel complet.

C'est aussi une question de granularité. Un ou plusieurs styles d'architecture peuvent être choisis à différents niveaux du système d'information. Cela peut aller du système d'information en lui-même jusqu'au plus petit des composants.

B. MVC

1. Description

Le MVC (Modèle Vue Contrôleur) n'est pas vraiment un style d'architecture mais plus un design pattern. Il est pris en considération ici car il très utilisé dans le monde du web. Conçu pour les logiciels avec interface graphique, il a pour but de bien séparer les données, la présentation et les traitements. Voici le détail des 3 parties extraites :

- **Modèle** : Représente le cœur algorithmique de l'application (traitements des données, interactions avec la base de données, etc.)
- **Vue** : Ce avec quoi l'utilisateur interagit. La vue n'effectue pas de traitement, elle reçoit les actions de l'utilisateur et les transfère au contrôleur
- **Contrôleur** : Prend en charge la gestion des événements de synchronisation pour mettre a jour la vue ou le modèle et les synchronise. Il reçoit tous les événements de la vue et enclenche les actions à effectuer.

Voici donc à quoi ressemble le flux de traitement :

- Chaque action utilisateur est analysée par le contrôleur (clic de souris, etc.)
- Le contrôleur demande au modèle approprié d'effectuer les traitements et notifie à la vue que la requête est traitée
- La vue notifiée fait alors une requête au modèle pour se mettre a jour (par exemple pour afficher le résultat du traitement fait par le modèle)

Il existe de nombreux dérivés à ce modèle (MVP, MVVM, MV*), mais sont finalement très similaires à celui-ci.

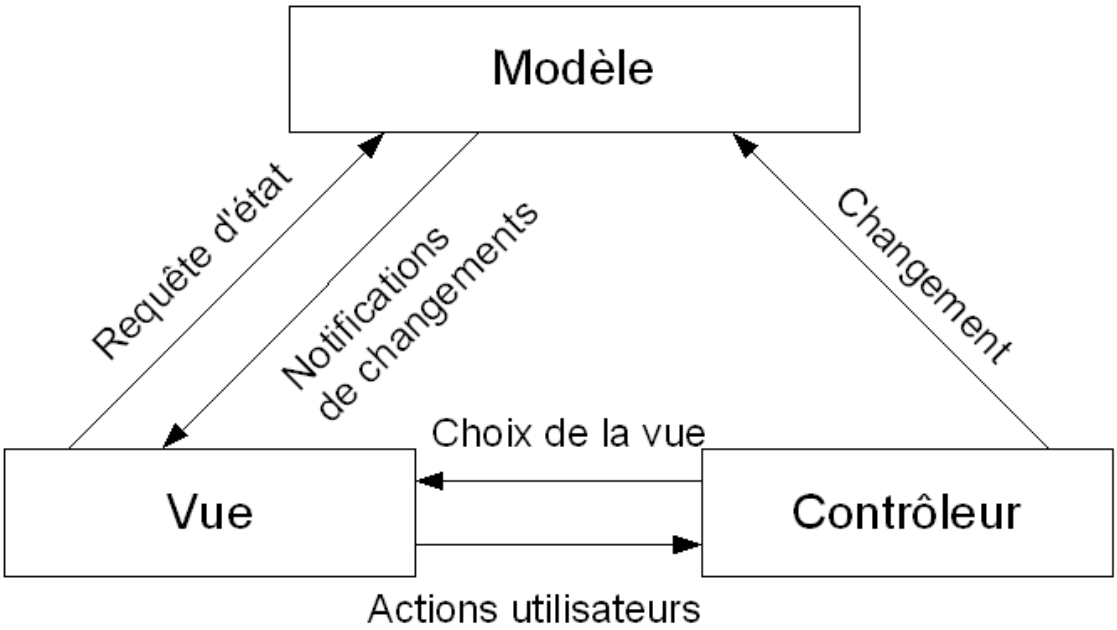

Figure II.1 : Modèle Vue Contrôleur

2. Avantages

- Simplicité, efficacité et clarté : La prise en main avec une telle architecture est rapide est peu coûteuse sur des petites ou moyennes applications
- Séparation des responsabilités minimale, mais efficace
- La modification d'une des parties n'impacte pas ou peu les autres
- Très répandu

3. Inconvénients et critiques

Très souvent dans les implémentations de ce modèle, la logique de l'application ne se trouve pas dans le modèle, mais dans le contrôleur, le modèle ne servant uniquement que de source de données. Ceci n'est pas vraiment un inconvénient, mais un choix. Que la logique soit dans le contrôleur ou le modèle, elle se retrouve finalement fortement couplée à celle-ci. La technologie choisie définit généralement une façon spécifique d'écrire un contrôleur ou un modèle. De cette manière, la logique de l'application se retrouve fortement couplée avec la technologie choisie, ce qui rend le changement de technologie difficile.

Nous verrons avec les autres styles d'architecture qu'il existe finalement deux types de logiques

dans une application. La logique spécifique au domaine de l'application (banque, finances, etc.), et la logique spécifique à l'application (contrôleur, service, etc.) et qu'il est généralement mieux de les séparer.

Ce pattern est à utiliser lorsque l'application est très simple. Il est rapide à mettre en oeuvre et à expliquer son fonctionnement. Pour une plus grosse application avec de nombreuses évolutions, il sera plus compliqué de maintenir un tel programme, devenant fortement couplé à la technologie utilisée. Une application MVC est très souvent monolithique.

C. Architecture en couches

1. Description

L'architecture en couches regroupe les fonctionnalités reliées en couches distinctes empilées verticalement. Les fonctionnalités dans une même couche sont reliées par un rôle ou une responsabilité commune. La communication entre les couches est explicites et faiblement couplée. Chaque couche encapsule les responsabilités de la couche en dessous, et uniquement dans ce sens.

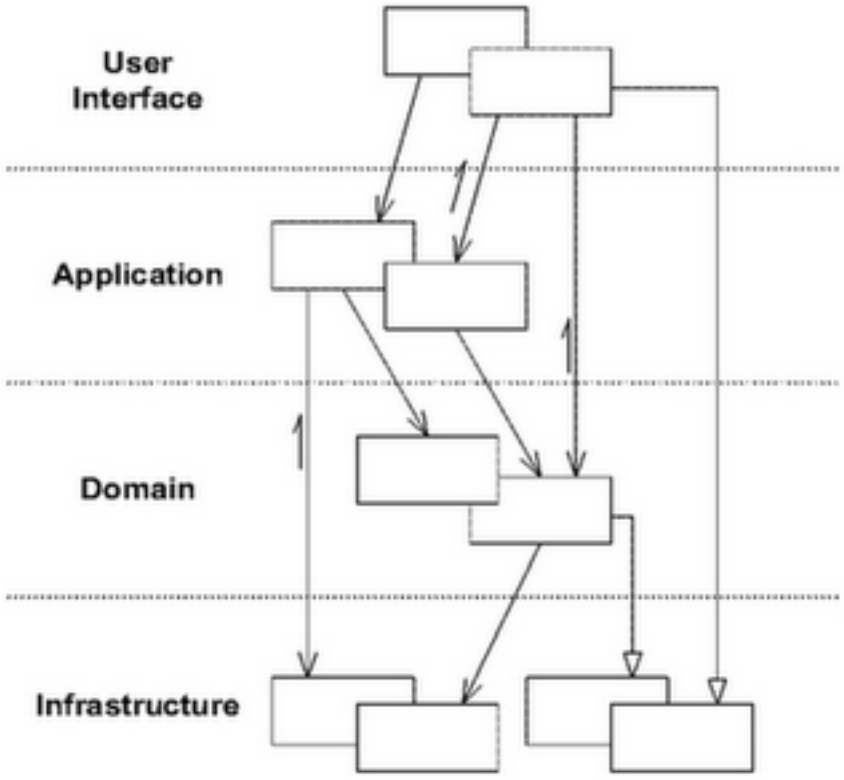

Figure II.2 : Exemple d'architecture en couches

Chaque couche peut être sur un même tiers (même ordinateur physique), ou sur plusieurs. On parle d'architecture multi-tiers dans ce cas là.

2. Avantages :

- **Abstraction** : L'architecture donne une bonne vue d'ensemble du système et il est facile de comprendre le rôle de chaque couche et les relations entre elles. Les couches hautes ont un niveau plus fort d'abstraction
- **Isolation** : Il est possible d'isoler techniquement les couches
- **Séparation des responsabilités** : Les dépendances entre les couches sont claires et facilement gérables
- **Performance** : Séparer les couches en plusieurs tiers (architecture multi-tiers) permet d'augmenter les performances
- **Réutilisable** : Chaque couche est réutilisable une fonctionnalité de la couche supérieure
- **Testabilité** : Les interfaces de chaque couche facilite la testabilité

3. Inconvénients

Si le fait de séparer le système en plusieurs couches est une bonne idée pour ne pas avoir un seul gros système, chaque couche peut devenir un monolithe. Généralement, chaque couche contient plusieurs fonctionnalités et sont dans le même processus. On assimile beaucoup cette architecture à un monolithe.

D. Architecture orienté composants

1. Description

L'architecture orienté composants se concentre sur la décomposition en composants individuels (fonctionnels ou logiques) exposant des interfaces de communication bien définies comprenant méthodes, propriétés et événements. Elle utilise un niveau d'abstraction plus haut que l'orienté objet. Voici une liste de ses principaux attributs :

- **Réutilisable** : Peut être réutilisé dans un autre scénario et dans d'autres applications
- **Remplaçable** : Peut être remplacé par des composants similaires

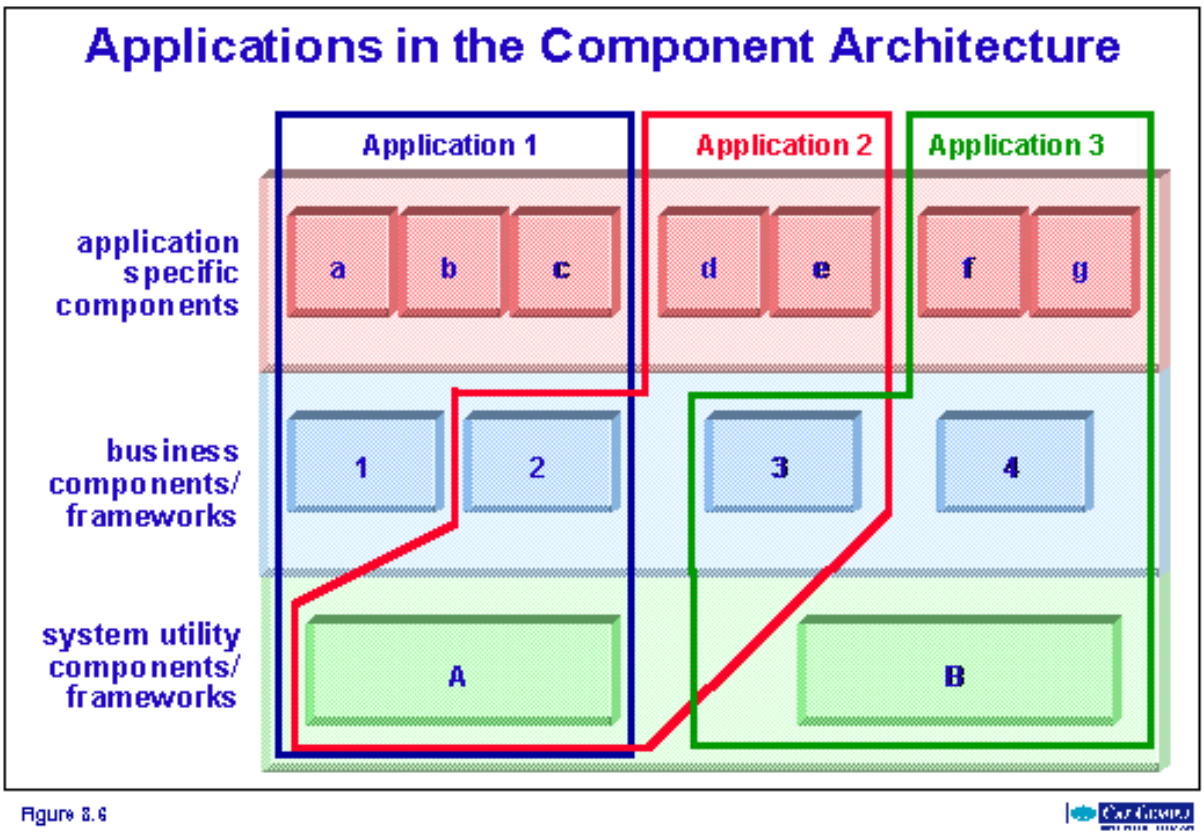

Figure II.3 : Architecture orienté composants

- **Sans contexte spécifique** : Les informations spécifiques a une application comme des données sont passées en variable d'entrée au lieu d'être inclues et gérées par le composant
- **Extensible** : Possibilité d'étendre le comportement d'un composant
- **Encapsulé** : Chaque composant expose des interfaces qui permettent à l'appelant d'utiliser les fonctionnalités du composant sans savoir son fonctionnement interne (état ou variables internes)
- **Indépendant** : Dépend très peu d'autres composants. Peut être déployé dans un autre environnement sans affecter les autres composants

2. Avantages

- **Facilité de déploiement** : Changement de version avec peu d'impact sur le reste du système
- **Coût réduit** : L'utilisation de composants externes permet de réduire le coût de développement et de maintenance
- **Facilité de développement** : Les composants exposent des interfaces définissant une

certaine fonctionnalité permettant de développer sans impact avec le reste du système (états internes plutôt que variables globales)

- **Réutilisable** : Comme nous l'avons vu, les composants peuvent être réutilisés dans un autre scénario ou dans d'autres applications

3. Inconvénients et critiques

Le niveau d'abstraction étant un peu plus élevé que l'orienté objet, ce style peut être un peu plus complexe à utiliser. Ce style ne se concentre pas sur les protocoles de communication, la gestion d'état, etc. Il n'est donc généralement pas utilisé seul. Il est souvent combiné avec d'autres styles pour créer un système complet. Ainsi, pour tirer profit de ce style, il faut bien le comprendre et bien l'utiliser.

E. Architecture orienté événements

1. Description

L'architecture orienté événements (*Message Bus Architecture*), utilise un système permettant de recevoir et d'envoyer des messages en utilisant un ou plusieurs canaux de communication. Il y a généralement un bus de message central, et les communications peuvent se faire de manière asynchrone. Ainsi, l'application peut interagir sans avoir le besoin de connaitre les détails des autres applications du système (couplage faible).

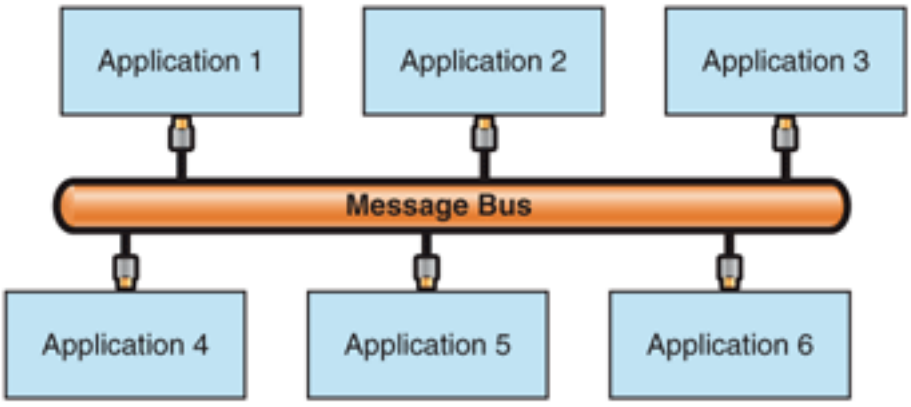

Figure II.4 : Architecture orienté evenements

2. Avantages

- **Extensible** : Les applications peuvent être enlevées ou rajoutées sans avoir d'impact sur les autres applications existantes
- **Faible complexité** : Les applications ont uniquement besoin de savoir comment communiquer avec le bus et non pas avec le reste du système
- **Flexibilité** : La combinaison d'applications formant le système et le pattern de communication peuvent être changés et manipulés à tout moment
- **Faible couplage** : Chaque application n'a qu'une seule dépendance : le bus. Le comportement interne et le langage utilisé sont indépendants. Cela permet une forte diversité technologique
- **Scalabilité** : Il peut y avoir plusieurs instances d'une même application attachés au bus pour accélérer le processus ou gérer plusieurs requêtes à la fois

3. Inconvénients et critiques

Il n'est pas aisé de choisir un pattern de communication. Il ne fait pas confondre "message" et "événement". Un événement est lorsque qu'il s'est passé quelque chose, tandis qu'un message peut être tout autre chose. Par exemple, un message peut contenir une demande d'action à un autre composant, ce qui peut augmenter le couplage entre les applications. Un événement ne doit pas être dépendant d'un application.

L'évolution des événements peut être difficile si un changement s'opère sur les propriétés de celles-ci. La modification ou la suppression d'un champ peut potentiellement avoir des répercussions sur le reste du système. C'est pour cela que choisir un bon pattern de communication permettant les évolutions est important. L'ajout de propriétés sur un événement n'aura pas d'impact sur le reste du système.

La seule dépendance de chaque application est donc le bus central. Il faut bien penser à abstraire la technologie utilisée pour faciliter le changement de technologie du bus (elles sont nombreuses).

F. Architecture orienté domaine (DDD)

1. Description

L'architecture orienté domaine (*Domaine Driven Design*) est une approche orienté objet se concentrant sur le domaine métier de l'application, ses éléments et ses comportements. Le cœur du logiciel est donc la modélisation directe du domaine, on appelle cela le modèle du domaine. Le vocabulaire professionnel du domaine est préféré au jargon technique, et pour cela, la communication avec les spécialistes du métier est primordiale.

Eric Evans, l'auteur du lire Domain Driven Design écrit :

« *Pour créer un logiciel de qualité, il faut connaître le sujet du logiciel. Il n'est pas possible de créer un logiciel de banque si on ne connait pas le domaine de la banque.* » [8]

Les 4 concepts à retenir sur le DDD sont les suivants :

- **Contexte borné** (*Bounded Context*) : Stratégie du DDD pour traiter les gros systèmes en les divisant en différents contextes en étant explicites sur leur relation.
- **Domaine** : Sphère de connaissance, domaine de l'application (exemple : banque, finances, etc.)
- **Modèle** : Système d'abstractions qui décrivent les aspects d'un domaine et qui peuvent être utilisés pour résoudre un problème en lien avec ce domaine
- **Langage omniprésent** (*Ubiquitous Language*) : Langage structuré autour du domaine du modèle utilisé par tous les membres de l'équipe en rapport avec toutes les activités avec le logiciel

Une implémentation possible de cette philosophie est l'architecture en oignon. Le domaine est isolé et encapsulé au centre, et la technique se forme autour. Cela permet de séparer concrètement la logique du domaine et la logique de l'application. Les couches techniques autour du centre sont facilement interchangeables.

2. Avantages

- **Communication** : Toute partie de l'équipe de développement utilise le modèle du domaine et les entités qu'il définit pour communiquer la connaissance du sujet et les demandes du client en utilisant un langage métier commun, sans jargon technique
- **Extensible** : Le domaine du modèle est modulaire et flexible. Les améliorations et les demandes du métier sont simples à implémenter

- **Testable** : Le domaine du modèle étant faiblement couplé, il est facilement testable
- **Couplage faible à la technologie** : Qu'importe la technologie utilisée (web service, interface graphique, etc.), le domaine métier étant central, il est facile de changer de technologies si le besoin se ressent, le cœur même du logiciel restera identique.

3. Inconvénients et critiques

Le langage du domaine n'est pas forcement facile à utiliser, et le jargon doit être compris de toute l'équipe. La communication est vitale et pas forcement facile a maintenir. Un mauvaise communication peut amener une mauvaise implémentation du logiciel.

Cette architecture possède une forte complexité et un coût plus élevé de mise en place. Il conviendra aux applications ayant un domaine complexe, mais sera beaucoup moins utile avec les applications plus simples.

G. Architecture orienté services (SOA)

1. Description

L'architecture orienté services (*Service Orienté Architecture*) a pour principe de proposer les fonctionnalités d'une application comme un ensemble de services. Les services possèdent une interface de communication standard permettant l'invocation, la publication et la découverte des autres services. La communication ne se fait pas via classes internes, mais uniquement via des schémas et des contrats. Elle est généralement distante (réseau local ou global), mais pas obligatoirement. Voici ses principaux attributs :

- **Autonomie** : Chaque service est maintenu, développé, et versionné indépendamment
- **Distribuable** : Peut être n'importe où sur le réseau tant que celui-ci supporte le protocole de communication
- **Faible couplage** : Chaque service est indépendant des autres. Il peut être modifié ou remplacé sans problème tant que l'interface est toujours compatible

2. Avantages

- **Abstraction** : Les services sont autonomes et accédés via un contrat formel (couplage faible)

- **Découvrable** : Les services exposent une description qui permet aux autres applications et services de les localiser et d'automatiquement déterminer les interfaces
- **Interopérabilité** : Les protocoles et formats de données sont basés sur les standards de l'industrie. Le fournisseur et le consommateur peuvent alors être construits et déployés sur des plateformes différentes (utilisation de plusieurs langages possibles)
- **Réutilisation** : Les services sont assez découpés pour fournir une fonctionnalité spécifique, plutôt que dupliquer la fonctionnalités dans plusieurs applications, ce qui supprime la duplication

3. Inconvénients et critiques

À chaque fois qu'un service interagit avec un autre service, la validation de chaque paramètre d'entrée prend place. Ceci augmente le temps de réponse et la charge de la machine, ce qui réduit la performance globale. De plus certains protocoles de communication sont plus lourds que d'autres (exemple : SOAP). L'architecture n'utilise pas forcément SOAP, mai c'est c'est très commun.

Même si un SI est découpé en services, au fil du temps, chaque service peut devenir très large, et effectuer plus de traitements qu'il ne devrait faire.

Cette architecture n'est pas faite pour les applications à interface graphique ou les applications en temps réels. De plus, elle a un certain coût de mise en place. Elle découpe beaucoup le système, et permet un peu plus d'éviter les monolithes. Cependant, chaque service peut devenir un monolithe à lui seul au fil du temps.

H. Microservices

1. Description

Le style d'architecture des microservices est plus récent que les autres et s'inspire grandement de ces derniers. Voici la définition de Martin Fowler dans son article définissant les microservices :

« *Le style d'architecture des microservices prend comme approche de développer chaque application comme un ensemble de petits services, chacun fonctionnant dans son propre processus, et communiquant avec des mécanismes légers (très souvent une ressource HTTP via une API). Ces services sont construits autour d'une fonctionnalité métier et sont déployables indépendamment par un système de déploiement automatisé. Il y a un minimum de gestion centralisée*

de ces services, qui peuvent être écrits dans différents langages de programmation et d'utiliser différentes technologies de stockage de données » [11]

Les microservices se rapprochent beaucoup de l'architecture SOA, mais avec une approche différente au niveau de l'implémentation :

- SOA : Intègre différentes applications comme un ensemble de services
- Microservices : Architecture chaque application comme un ensemble de services

La majeure différence se situe donc au niveau de la taille d'un service. Les microservices se rapprochent plus la philosophie Unix : « *Faire une seule chose, mais la faire bien* » [1]. Plus un service sera petit, moins il aura de chance d'évoluer en un monolithe. Le principe très simple est le suivant : plus un service est petit, moins il sera monolithique au fil du temps. Voici une liste des différents attributs de cette architecture :

- **Indépendant** : Chaque microservice a son propre cycle de gestion de versions et de déploiement. Il y a généralement une équipe de développement aux compétences diverses par microservice (scalabilité des équipes).
- **Faible couplage** : Moyens de communication légers
- **Donnée décentralisée et polyglotte** : Chaque microservice possède sa base de données sur une technologie indépendante

2. Avantages

- **Petit et focus sur une seule fonctionnalité** : Facile à comprendre, temps de chargement de l'IDE court
- **Bonne décomposition des modules** : Limites des modules claires et interface explicite, très bien pour les grandes équipes
- **Déploiements indépendants** : Pas de gros cycles de recettes
- **Diversité technologique** : Chaque microservice peut être implémenté dans des technologies différentes. Permet une architecture évolutive
- **Scalabilité fonctionnelle** : Possibilité de dupliquer les microservices là ou c'est critique (fonctionnalités les plus demandées)
- **Réécriture limitée a un seul service** : Il est parfois bénéfique de réécrire tout ou partie du code existant. Les microservices permettent une réécriture partielle sans grand impact sur la totalité du système

[1] Do one thing and do it well

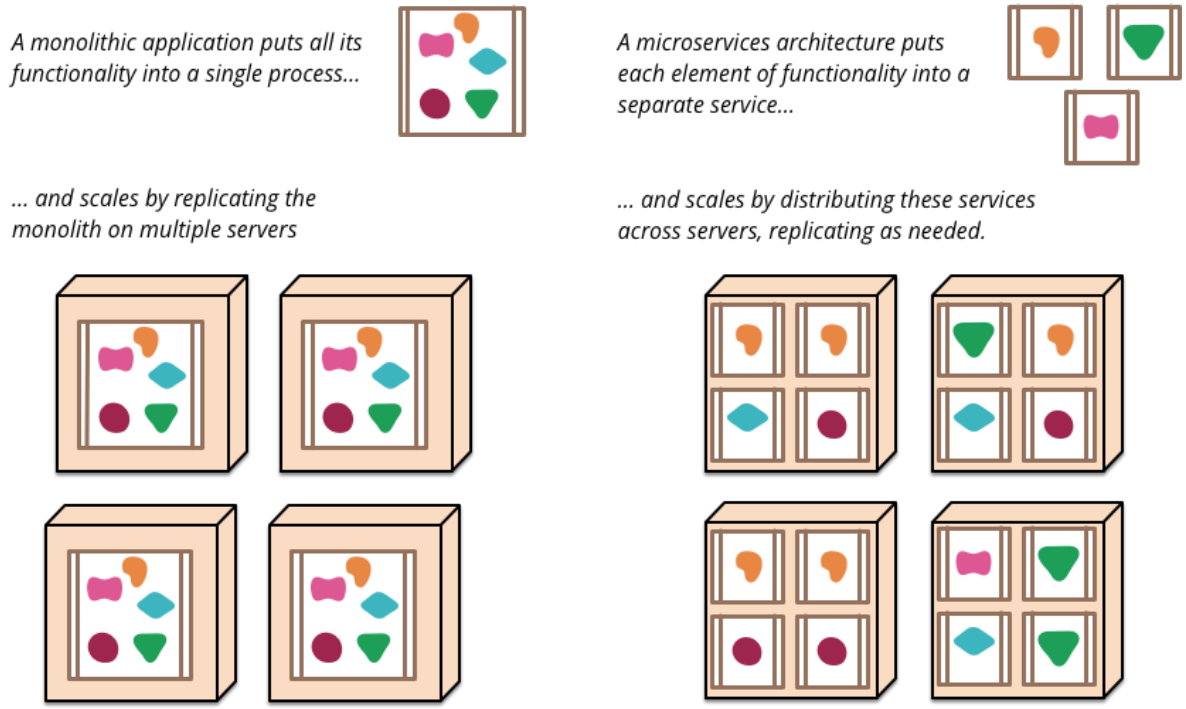

Figure II.5 : Différences entre une architecture monolithique et une architecture microservices

3. Inconvénients et critiques

La décomposition fonctionnelle d'une application en microservices n'est pas tache aisée. Un mauvais découpage peut faire perdre tous les avantages qu'elle apporte et fait se rapprocher du monolithe. Selon Eric Evans[5], l'architecture DDD aiderait à la décomposition fonctionnelle grâce à la notion de contexte borné (*bounded context*).

Cette architecture impose une forte complexité opérationnelle (configuration, déploiement, etc.). Une automatisation des tâches de déploiement (déploiement continu) devient nécessaire. Il faut rajouter à cela une bonne surveillance (monitoring).

Si la diversité technologique est renforcée avec cette architecture, il est demandé aux équipes de développement de maîtriser plusieurs technologies et langages. Cette architecture permet une scalabilité très précise, mais impose le développement distribué. Ainsi il faut savoir gérer :

- Donnée décentralisée
- Communication entre les services
- Gérer l'échec d'un microservice
- Gestion des tests

Cette architecture propose donc de nombreux avantage mais aussi un challenge technique élevé.

Il n'est pas aisé de réussir a implémenter correctement cette architecture du premier coup. Il est préférable de commencer simplement par une architecture monolithique, pour ensuite migrer au fur et a mesure les parties essentielles en microservices.

3. Spécificités du front-end

Une application web moderne se découpe généralement en deux applications (style d'architecture client/serveur) : le client (*front-end*) et le serveur (*back-end*). Nous allons voir les spécificités de chacun et décider des styles d'architecture à appliquer pour notre projet.

A. Rappel sur la gestion d'état

Il y a deux types d'applications, celles qui ont une gestion d'état et celles qui n'en n'ont pas. Une application dites "*stateful*" stocke en mémoire certaines valeurs entre plusieurs appels, tandis qu'une application dîtes "*stateless*" ne gardera rien mémoire et est sans contexte précis.

Une application *stateless* est beaucoup plus facile à gérer car qu'importe le processus depuis lequel il est appelé, le résultat ne sera pas contextualisé à celui-ci. On ne peut pas se permettre d'appeler une application *stateful* depuis 2 processus différents car les valeurs stockées en mémoire par le premier processus sont inaccessibles par l'autre, interdisant à un loadbalancer de passer d'un serveur à l'autre de manière invisible pour le client.

La scalabilité est beaucoup plus facile avec une application *stateless*, car le résultat est indépendant du processus où il est appelé. De manière générale, cela simplifie grandement le développement et le passage aux microservices.

B. Application traditionnelle

1. Description

On parle d'une application web traditionnelle lorsque la formation de la vue (HTML) se produit du cote du serveur (*server-side rendering*). Le principe est le suivant :

- Un navigateur web envoie une première requête lors de l'arrivée sur un site web.
- Le serveur va chercher des données si nécessaire, puis forme la vue entièrement (document HTML statique) et répond ce document au navigateur.

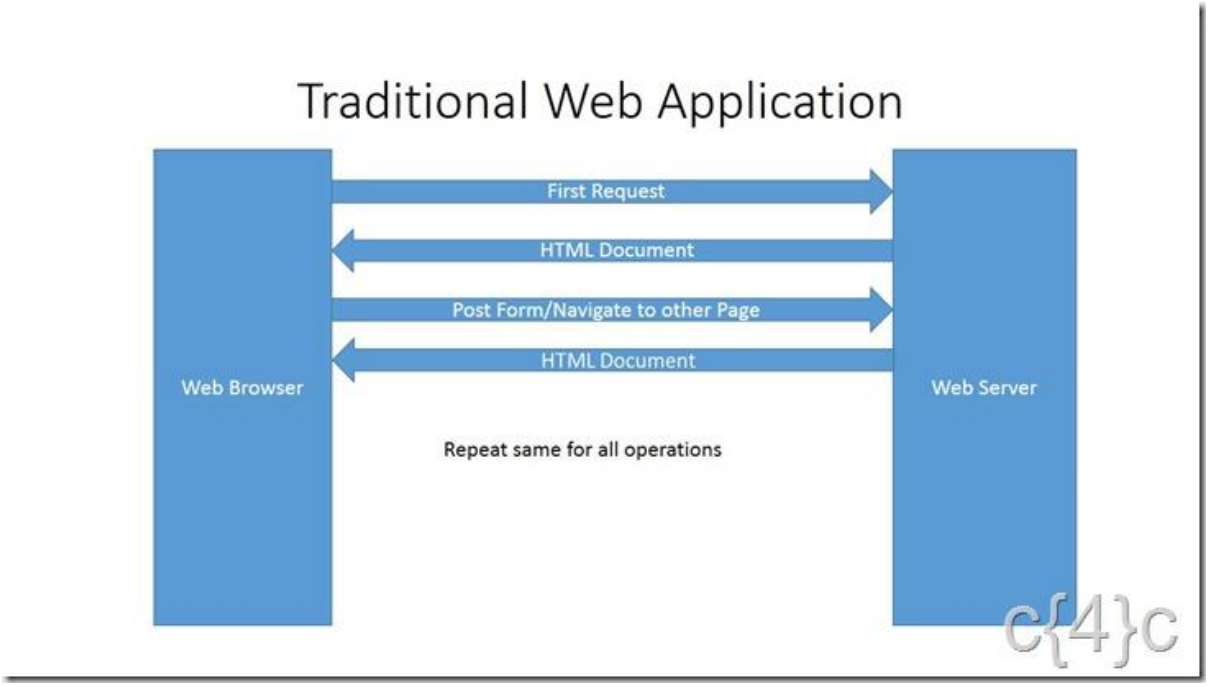

Figure II.6 : Application web traditionnelle

- Le navigateur affiche cette page directement. Si l'internaute remplit un formulaire, ou clique sur un lien pour changer de page, l'opération précédente recommence. Il reçoit la page HTML suivante, etc.

L'HTML étant un document statique, le JavaScript est généralement utilisé pour rendre l'expérience utilisateur plus intéressante en modifiant la structure de l'HTML (via le DOM) et permettre des effets de type animation, etc. Il faut bien comprendre que les fichiers HTML/CSS/Javascript sont rechargés à chaque demande de page, car le serveur doit les régénérer.

2. Critiques

Cette architecture a été utilisée depuis très longtemps et les framework proposés (généralement du MVC) sont très avancés. Nous allons cependant voir les limites de celles-ci.

L'HTML est donc formé du côté du serveur, mais le JavaScript est exécuté du côté du client. C'est donc très perturbant de gérer deux types de code dans une même application, surtout lorsque les interactions avec l'utilisateur sont complexes. L'application devient rapidement monolithique à cause de ce mélange de responsabilités.

Le client doit charger la page HTML à chaque fois qu'il navigue entre les pages ou qu'il envoie un formulaire. Le serveur doit générer la vue à chaque changement de page, ce qui ralentit la

navigation du client et augmente la charge du serveur. L'avantage cependant, c'est que le client possède très peu de charge.

Ce type d'applications est très souvent *stateful*. Il est commun lors du développement d'application web traditionnelle de stocker des variables en session pour chaque utilisateur (panier, etc.). Nous avons vu que cela complexifie beaucoup l'application, surtout si l'on veut ajouter des serveurs.

Ce type d'applications convient très bien pour les applications web simples ayant très peu d'interactions avec le client, et où le fait d'être une application monolithique et *stateful* n'est pas dérangeant. Cela correspond à une poignée de sites web comme un site statique, présentation d'une produit, etc.

C. Passage aux Single Page Application

1. Description

Les évolutions récentes du JavaScript ont changé complètement les méthodes de développement d'applications web grâce aux performances des derniers moteurs JavaScript et à l'interaction facilitée avec un serveur via AJAX[2]. Il existe désormais un nouveau type d'applications web, les *Single Page Applications* (SPA).

L'AJAX permet de lancer une requête HTTP non bloquante (asynchrone) et d'avoir le résultat de la requête sans avoir à recharger la page. Les navigateurs récents et leurs moteurs JavaScript sont capable de gérer eux-mêmes totalement la vue en changeant la structure des pages en fonction des actions du l'utilisateur. L'expérience est alors plus dynamique et non bloquante, et donc plus proche des applications de bureau.

Ce type d'application sépare clairement l'interface graphique (client) et des traitements non visuels (serveur) tels que la gestion des données en deux applications différentes (2 tiers) pouvant utiliser des technologies différentes. Ce dernier apparaît généralement sous la forme de web service, pouvant servir plusieurs types de clients. Les moyens de communication sont généralement légers (JSON, XML, etc.). N'ayant plus à former la vue, la charge des serveurs est aussi grandement diminuée.

[2]Asynchronous JavaScript and XML

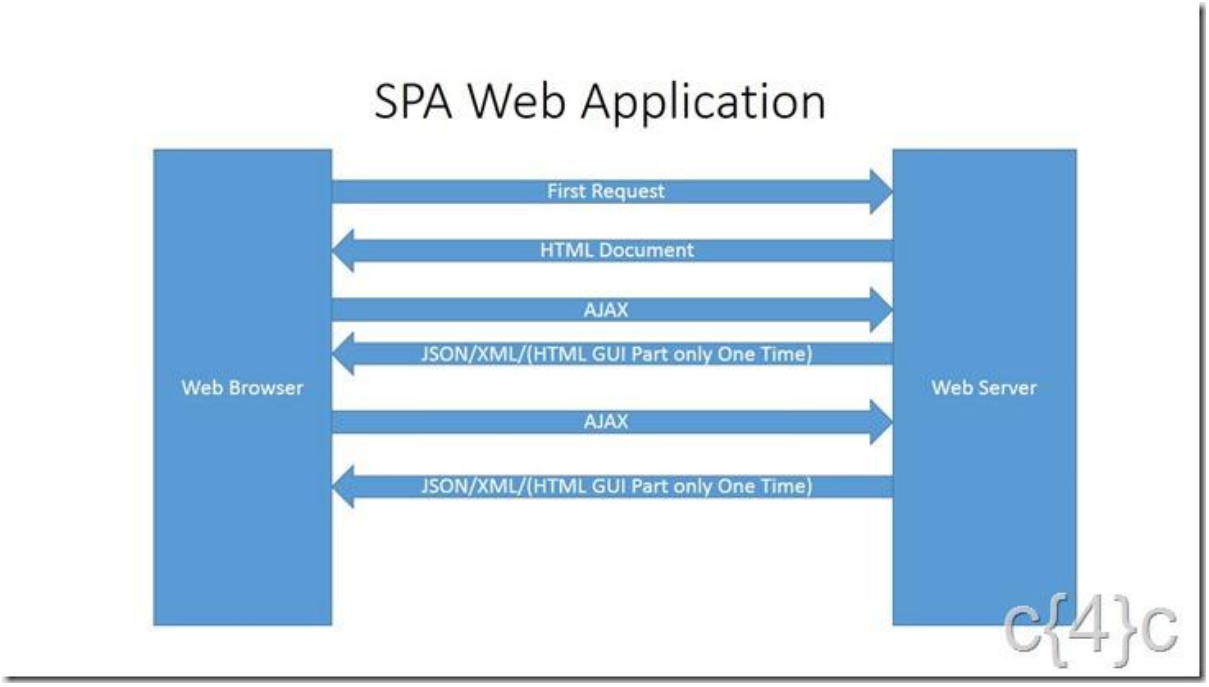

Figure II.7 : Single Page Application

2. Critiques

L'application n'ayant qu'un seul point d'entrée, celle-ci peut être un peu plus longue a charger au début. Ceci vient du fait que le navigateur doit complètement charger les frameworks dont il est dépendant avant de pouvoir débuter l'application. Après ce chargement initial, l'application est cependant très fluide, n'ayant pas le besoin de recharger plus tard. De plus, les frameworks utilisés sont très souvent les mêmes et sont souvent mis en cache.

Ce type d'application repose entièrement sur le JavaScript. Si auparavant il était très peu utilisé, voir facultatif, il est désormais obligatoire pour faire fonctionner ce genre d'applications. On peut cependant considérer qu'aujourd'hui, tous les internautes ont le JavaScript d'activé.

Les moteurs de recherches ou les outils d'analyse sont grandement basés sur le modèle des applications traditionnelles. De nombreux moteurs de recherche n'exécutent pas de JavaScript et ne peuvent indexer les SPA. Les outils d'analyse de page web utilisent généralement le chargement de nouvelles pages comme unité d'analyse. C'est désormais beaucoup moins pertinent avec les applications comprenant un seul point d'entrée.

Si les standards du web ont étés conçus à la base pour les applications traditionnelles, c'est en cours de changement. Les SPA sont de plus en plus communes et évoluent dans ce sens. Par exemple, le moteur de recherche le plus utilisé Google[7] a récemment intégré le JavaScript lors de son processus d'indexation.

Ce type d'applications convient très bien pour les applications web avec beaucoup d'interactions utilisateur avec un meilleur dynamisme. De nombreuses applications très connues utilisent déjà ce modèle comme Gmail (application de gestion de mails), Facebook, Twitter (réseaux sociaux), etc.

D. Choix de styles d'architecture

Au niveau du front-end, le pattern MVC est très répandu, le pattern le plus utilisé pour les interfaces graphiques. Nous avons vu les avantages et les inconvénients de celui-ci. D'autres possibilités sont apparues avec les derniers standards du web.

1. Architecture orienté composants

Cette architecture est de plus en plus utilisé dans le monde du web grâce aux *web components* (composants web). Un composant web est un élément HTML individuel encapsulant certaines fonctionnalités, et réutilisable plusieurs fois dans des contextes différents. Voici une liste des différentes propriétés d'un composant web standard :

- **Éléments et propriétés personnalisés** : Permet la création d'élément HTML autres que ceux standards (p, a, h1, h2, etc.) avec des propriétés propres et personnalisables
- **Shadow DOM et scoped CSS** : Création d'un nouveau noeud isolé du DOM, ayant son propre CSS n'affectant pas le reste de la page
- **Imports HTML** : Permet l'import d'autres composants via le code HTML
- **HTML Templates** : Un composant s'écrit via un fichier HTML

Tous les navigateurs n'implémentent pas ces derniers standards du web, mais il est possible de les utiliser dès aujourd'hui grâce a des polyfills ou des framework définissant leur propre manière de créer des composants web.

2. Architecture orienté événements

Les composants étant indépendants et autonomes, une façon de relier les relier est d'utiliser un bus d'événements central (architecture orienté événements). Il est aisé d'en créer un soi-même en JavaScript, et certains framework le proposent. Cela permet de profiter des avantages de ces deux architectures.

Les possibilités sont nombreuses, et les choix doivent êtres faits selon les équipes et les projets.

4. Spécificités du Back-end

Avec une application web traditionnelle, le back-end possède de nombreuses responsabilités, donc la formation de la vue. Avec les SPA, ce n'est plus nécessaire.

A. Passage aux Web services

Un service web permet la communication et l'échange de données entre deux applications. Le protocole généralement utilisé est l'HTTP. Il existe 2 principaux types de web services : SOAP et REST.

1. SOAP

« SOAP (Simple Object Access Protocol) est un protocole de RPC (Remote Procedure Call) orienté objet bâti sur XML. Il permet la transmission de messages entre objets distants, ce qui veut dire qu'il autorise un objet à invoquer des méthodes d'objets physiquement situés sur un autre serveur. Le protocole SOAP est composé de deux parties :

- *une enveloppe, contenant des informations sur le message lui-même afin de permettre son acheminement et son traitement*
- *un modèle de données, définissant le format du message, c'est-à-dire les informations à transmettre (appelé WSDL[3]).* », définition de Wikipédia.

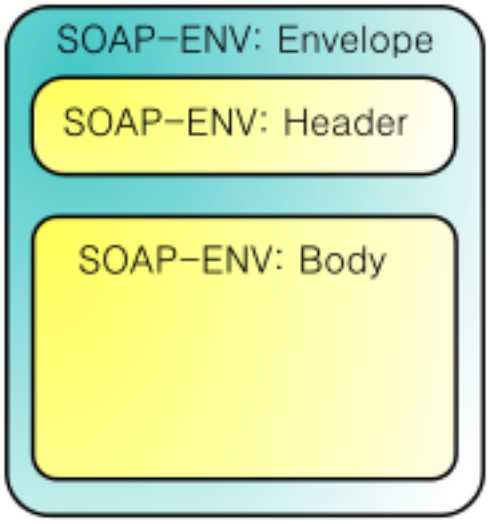

Figure II.8 : Représentation de l'enveloppe SOAP

[3]Web Services Description Language

SOAP décrit la manière dont les applications doivent communiquer entre elles, ce qui peut augmenter le couplage entre le serveur et les clients. Une évolution du coté du serveur demande une mise a jour des clients.

SOAP permet l'utilisation de plusieurs protocoles autres que l'HTTP tel que le SMTP[4], mais en pratique, c'est généralement uniquement l'HTTP qui est utilisé. Le nombre d'informations transitant décrit avec le XML alourdit grandement les échanges.

SOAP a été utilisé pendant très longtemps et de nombreux web services écrits en SOAP existent encore actuellement (de nombreux web services sont écrits en SOAP chez Cdiscount). Cependant, des solutions plus légères sont préférées telle que REST. SOAP est souvent utilisé pour mettre en place une architecture orientée services (SOA).

2. REST

Contrairement à SOAP, qui est un protocole à part entière, REST (*Representational State Transfer*) est un style d'architecture. Les systèmes qui suivent les principes de l'architecture REST sont appelés RESTful. Le protocole HTTP est utilisé, et permet de respecter tous les principes de l'architecture.

REST est une architecture orientée ressource (contrairement à SOAP qui est orienté méthodes). Par exemple, une API web représentera un client comme une ressource, et sera manipulable a cette adresse :`http ://www.foo.com/Clients`. Les verbes HTTP (GET, POST, PUT, DELETE) permettront de récupérer, ajouter, modifier ou supprimer cette ressource.

REST est une architecture dite sans état (*stateless*). Un serveur RESTful peut ainsi répondre à des requêtes venant de plusieurs clients, et il est plus aisé de multiplier les serveurs. REST est généralement utilisé lors d'une mise en place d'une mise en place des microservices grâce a sa légèreté.

B. Choix de styles d'architecture

1. SOA + architecture en couches

Cdiscount utilise actuellement une architecture orientée service et chaque service utilise une architecture en couches. Nous avons vu les avantages et les inconvénients de chacun. Si la séparation fonctionnelle était claire au début, aujourd'hui chaque service s'apparente de plus

[4]Simple Mail Transfer Protocol

en plus à un monolithe. La société est en pleine réflexion pour casser ses monolithes et avoir une architecture plus souple.

2. Microservices

C'est l'architecture vers laquelle Cdiscount veut tendre car elle permettrait de briser ses monolithes qui ralentissent fortement le développement de fonctionnalités. La société étudie actuellement une stratégie de migration. Pour le projet de démonstration, nous souhaitons mettre aussi en place cette architecture. Ce n'est pas tâche aisée et nous allons procéder en plusieurs étapes.

5. Ce qu'il faut retenir

Nous avons vu de nombreux styles d'architecture et les spécificités du monde du web. Les principes clés à retenir selon moi et ce que nous allons appliquer pour notre projet sont les suivants :

A. Le monolithe, à éviter

Les choix d'architecture que nous faisons ont pour but d'éviter le monolithe. Le fait de séparer le client et le serveur sépare déjà le processus en deux, mais ce n'est pas suffisant. Pour faire en sorte que le serveur ne devienne pas lui-même un monolithe, il faut que chaque service soit le plus petit et simple possible (KISS[5]). L'architecture se rapproche le plus de ce concept est l'architecture des microservices.

B. Deux types de logiques distinctes

S'il y a bien un concept principal à retenir du DDD, c'est qu'il existe deux types de logiques : la logique du domaine de l'application et la logique applicative. La logique du domaine est la modélisation la plus proche du métier, et c'est le cœur de l'application. La logique applicative reliera la logique du domaine à une certaine pile technologique (interface graphique, web services, etc.). Le fait de les séparer permet de coller n'importe quelle pile technologique au métier, et d'en changer facilement.

[5]Keep It Simple Stupid

C. Architecture > technologies

La diversité technologique possède de nombreux bénéfices. Un bon choix d'architecture est primordial pour permettre cela. Il faut donc d'abord bien choisir son architecture avant de foncer tête baissée dans une certaine technologie. Certes, la technologie est ce qu'il y a de plus concret pour réaliser une application, mais il ne faut pas perdre de vue qu'elle évolue très vite et qu'elle risque de changer rapidement.

Une architecture permettant une permutation facile de technologies, est une architecture qui durera. Utiliser des nouvelles technologies autant que possible garde les développeurs affûtés et accroît leur créativité.

D. Résumé des choix d'architecture du projet

Pour notre projet, l'application client sera une SPA basée sur une architecture orientée composants couplée avec une architecture orientée événements. Le serveur sera un web service RESTful basé sur l'architecture des microservices. Nous allons implémenter les microservices au fur et à mesure car c'est pour nous une architecture nouvelle et plus complexe à mettre en oeuvre.

III

Choix technologiques

Le monde du web évolue très vite et les technologies sont nombreuses. Il n'est pas aisé d'en choisir, cela dépend beaucoup de l'architecture que l'on veut mettre en place ainsi que le style de développement de l'équipe.

1. Choix d'un framework SPA

C'est surement le choix le plus difficile à effectuer. Il existe de nombreux framework pour créer des SPA. Il est aussi possible de ne pas en utiliser, et d'utiliser une combinaison de librairies. Un framework apporte cependant de la structure à l'application, une architecture flexible et renforce la séparation des responsabilités.

Nous ne présenterons pas tous les framework SPA, car il en existe beaucoup. Ils évoluent très vite et le choix peut être différent d'une année à l'autre. Voici ceux pris en compte pour notre projet :

A. AngularJS

1. Description

AngularJS est un framework créé en 2009 par Google. Il propose :

- Injection de dépendances
- Pattern MVVM utilisé mais avec une notion de "service" qui peut être utilisé pour la logique de domaine
- Orienté composants possible (via directives)
- Testing avancé mis en avant (unitaire et UI (User Interface))
- Découpage fonctionnel sous forme de modules
- Two way data-binding : Lien entre la vue et le contrôleur automatique

Une version 2 est en cours de développement et va apporter de nombreuses choses :

- Shadow DOM
- Meilleurs performances
- Encore plus orienté composant
- Plus simple

2. Avantages

- Stable et largement utilisé en production
- Large communauté : framework le plus utilisé, de nombreux modules open source
- Le Two way data-binding facilite grandement le développement et permet d'éviter la manipulation du DOM
- Séparation des fichiers claire pour chaque composant (HTML, JavaScript, CSS)

3. Inconvénients

- Prise en main difficile
- Version 2 non rétro-compatible (si version utilisée en orienté composants, migration facilitée)
- Performances (le two way data-binding a un impact sur la performance générale de l'application)

B. ReactJS + Flux

1. Description

ReactJS est un framework créé en 2013 par Facebook. Il est par conception orienté composants et utilise le *virtual DOM*, implémentation différente du *Shadow DOM*, avec de très bonnes performances. Par conception, ce framework s'occupe unique de la vue et doit être complété par d'autres librairies.

Facebook a créé une architecture se nommant Flux, complémentaire de ReactJS. C'est une architecture orienté événements prônant le flux de données unidirectionnel (contraire du two way data-binding). Bien utilisé, il est beaucoup plus clair et plus performant. Ce qu'il faut retenir c'est : les données descendent, et les actions remontent, unidirectionnellement.

2. Avantages

- Simple et rapide a mettre en place
- Architecture Flux claire et efficace
- SEO : Possibilité de faire du server-side rendering
- Performances
- Communauté grandissante

3. Inconvénients

- Utilisation du JSX : Mélange l'HTML et le JavaScript dans un même fichier par souci de simplicité. Ce n'est pas un inconvénient en soi, mais il faut approuver ce concept. Cela mène selon moi à un mélange de vue et de logique, même si cela simplifie grandement la création de composants.
- JSX et virtual DOM non standards, contrairement au Shadow DOM. Ils sont cependant très efficaces

C. Polymer

1. Description

Polymer est une librairie crée par Google en 2014. Ce n'est pas vraiment un framework, car il est beaucoup moins complet que les autres. Les web components, tels qu'ils sont définis par le W3C ne sont pas encore supportés par tous les navigateurs. Polymer propose donc un système de polyfills pour créer des composants web se rapprochant plus du standard, comprenant le Shadow DOM, l'import HTML, etc.

2. Avantages

- Les composants créés se rapprochent plus du futur standard que les autres frameworks
- Peut être combiné avec d'autres librairies, combinaison personnalisée

3. Inconvénients

- Très récent, version 1.0 sortie il y a peu de temps

- Peu de modules open source
- Pas un framework, moins complet mais si le choix est de faire une combinaison de librairies, pourra convenir

D. Un choix difficile

Le choix est difficile à effectuer car les technologies présentées sont toutes de bonne qualité, conviennent à nos choix d'architecture (orienté composants et orienté événements) et sont à la pointe des technologies web. Le choix dépendra finalement du projet et du style de développement des équipes.

Pour notre projet, nous avons choisi AngularJS. C'est le framework le plus mature et possédant la plus grande communauté dans sa catégorie et les ressources à disposition sont importantes. La testabilité est fortement mise en avant, ce qui est un point clé pour notre projet. Notre équipe possède déjà de l'expérience sur ce framework et nous sommes efficaces avec celui-ci. Pour un projet durant 2 mois, c'est un facteur important.

Nous aurions pu choisir ReactJS pour ses performances et sa philosophie, mais finalement les performances d'AngularJS sont suffisantes pour notre projet, et nous préférons son style de développement (préférence pour la structure des fichiers proposée par AngularJS par rapport au JSX de ReactJS).

Polymer est selon moi trop jeune, la version 1.0 étant sortie alors que nous avions déjà commencé le projet. Avec une bonne combinaison de librairies, il peut cependant devenir une alternative très intéressante.

Nous ajoutons un bus d'événements central pour profiter de l'architecture orienté événements. N'étant pas fourni avec AngularJS, nous avons créé notre propre bus en JavaScript.

2. Choisir les bons outils

De nombreux outils sont à disposition pour faciliter le développement d'applications web. Certains sont même nécessaires pour le développement mobile ou pour appliquer les méthodes agiles. Les tâches répétitives réduisent grandement la productivité des développeurs (DRY : Don't Repeat Yourself). L'automatisation de celles-ci permettent alors une meilleure productivité et simplifient le déploiement.

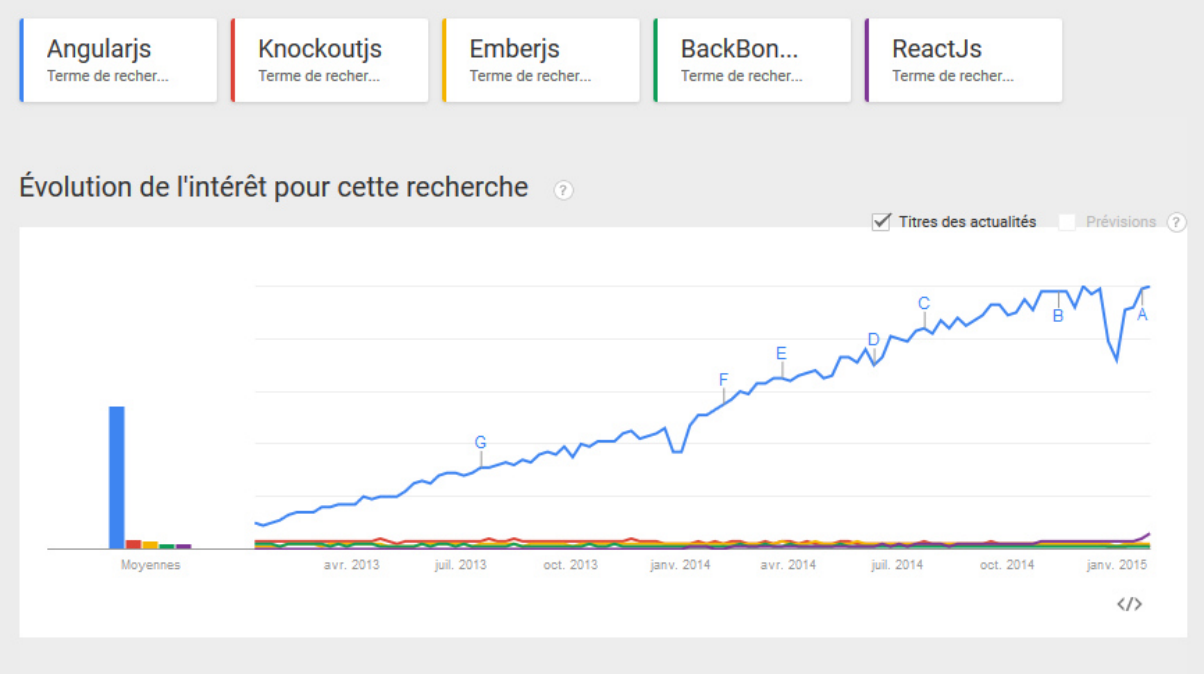

Figure III.1 : Tendance des recherches d'après Google Trends[1]

A. Outils de compilation

1. Gulp

Gulp est un *task runner*. Son but est de limiter les actions répétitives et d'automatiser certaines taches. Il est très comparable à Maven du monde Java. Il en existe plusieurs dans le monde du JavaScript, mais nous avons choisi celui-ci pour sa simplicité. Nos principales tâches de compilation sont :

- "**build :web**" : C'est la tâche principale, elle compile les fichiers sources et prépare le résultat final dans un dossier différent prêt pour la production
- "**build :mobile**" : Effectue l'équivalent pour le mobile
- "**watch**" : Relance automatiquement la compilation des fichiers a chaque sauvegarde d'un des fichiers sources. Recharge aussi le navigateur web
- "**server**" : Lance un serveur de développement lisant les fichiers compilés par la tache de compilation

Comme nous utilisons la dernière version de JavaScript qui n'est pas encore supportée par tous navigateurs, nous devons la transpiler (passage a une version antérieure compatible). D'autres astuces sont aussi utilisées comme la minification des fichiers ainsi que d'autre optimisations. Nous gagnons énormément de temps de développement avec des outils et la mise en production

est facilitée.

a. Cordova Cordova (anciennement PhoneGap) est un framework de développement mobile multiplateforme. Il permet de développement d'application mobiles hybrides en utilisant uniquement les technologies du web avec la possibilité d'utiliser les fonctionnalités natives du téléphone tels que l'appareil photo, le GPS, les notifications, etc.

Nous utilisons CocoonJS combiné avec *webview-plus* pour accélérer l'application (les webviews d'origine sont n'utilisent parfois pas totalement les capacités du téléphone selon le système d'exploitation utilisé.

B. Automate

Dans la même lignée que les *tasks runners*, nous utilisons un automate appelé Yeoman. C'est un outil permettant d'automatiser la création de fichiers redondants. Par exemple avec AngularJS, la création d'un composant est longue parce qu'il y a de nombreux fichiers. Nous avons alors créé notre propre générateur avec cet outil. L'utilisation se fait ainsi : `yo angular-es6-components:component product`.

C. Gestionnaire de versions

Un gestionnaire de versions est fortement recommandé, surtout avec les méthodes agiles. Nous utilisons git.

D. Gestionnaire de paquets

Nous utilisons beaucoup de librairies open source, que ce soit du coté du front-end ou du back-end. Chaque librairie est à utiliser avec une version spécifique et la gestion se complexifie avec de nombreuses librairies externes. De plus, c'est une mauvaise pratique de mélanger les librairie tierces avec le code source sur le gestionnaire de versions.

Un gestionnaire de paquets utilise un fichier descriptif comprenant la liste des librairies ainsi que leurs versions. Il suffit d'une ligne de commande pour toutes les installer. Au niveau du front-end, nous utilisons Bower.

E. Framework Graphique

Il existe plusieurs solution pour faciliter l'utilisation du CSS. Des surcouches comme SASS ou LESS ajoutent des fonctionnalités à ce langage, et de nombreux frameworks existent. Les ajouts à faire en CSS sont beaucoup moindres.

Si Bootstrap est le plus connu, nous avons utilisé un plus récent : Angular Material. Il propose d'utiliser le Material Design de Google adapté pour AngularJS. Il est visuellement très agréable et possède une bonne ergonomie. Il simplifie grandement le développement car il est très adapté à AngularJS et est facile à utiliser.

3. Choix d'un langage serveur

Il existe de nombreuses solutions pour créer des serveurs RESTful. Les plus communes sont :

- Java (avec un framework HTTP comme Jersey)
- C# (avec WCF)
- PHP (pas optimisé à la base pour faire du RESTful, mais possible tout de même)

Nous avons choisi le NodeJS comme solution. Comme vu précédemment, le JavaScript est un langage très adapté pour le back-end, d'autant plus avec la dernière version. NodeJS comporte le plus grand nombres de modules open sources avec son gestionnaire de paquets npm. Utilisant l'architecture des microservices, nous pourront créer certains services dans d'autres technologies si nous trouvons cela est nécessaire. Nous voulons pour l'instant tirer parti du fait que nous utilisons le même langage de développement de chaque côté (client et serveur) sommes très efficaces de cette manière.

4. Containers

La meilleure façon de créer et de gérer des microservices actuellement est de passer via des containers en utilisant Docker. Les containers créés par Docker sont des espaces Linux isolés pouvant interagir avec d'autres containers. Le principe est très similaire à une machine virtuelle, mais en beaucoup plus léger et plus efficace. Il est aisé de créer par exemple plusieurs instances d'une même application. Docker utilise un fichier descriptif (Dockerfile) permettant d'en créer facilement à la volée.

Dans le cas de notre projet, nous avons commencé par creer une seule instance de notre API RESTful dans un seul container (apparant a un monolithe). Il est aisé de creer plusieurs instances de celui-ci si nous le souhaitons grace a du load-balancing.

Si nous souhaitons aller encore plus loin et respecter vraiment l'architecture des microservices, nous devrions créer un microservice par route. Par exemple, les routes /api/products ou /api/-basket serait dans des containeurs isolés avec un nombre d'instance propre (scaling fonctionnel). Il faudra ensuite créér une API interface (API gateway), qui redirigera les requêtes vers les bons microservices.

IV

Méthodologie

Les choix d'architecture et de technologies sont faits, il est maintenant important de choisir une bonne méthodologie. Cdiscount utilise historiquement la méthode du cycle en V. Cependant, elle tend de plus en plus vers les méthodes agiles et la récente création des pizza teams le démontre.

1. Cycle en V

Le cycle en V est un modèle conceptuel de gestion de projet. Il est constitué en plusieurs étapes et sont découpées ainsi :

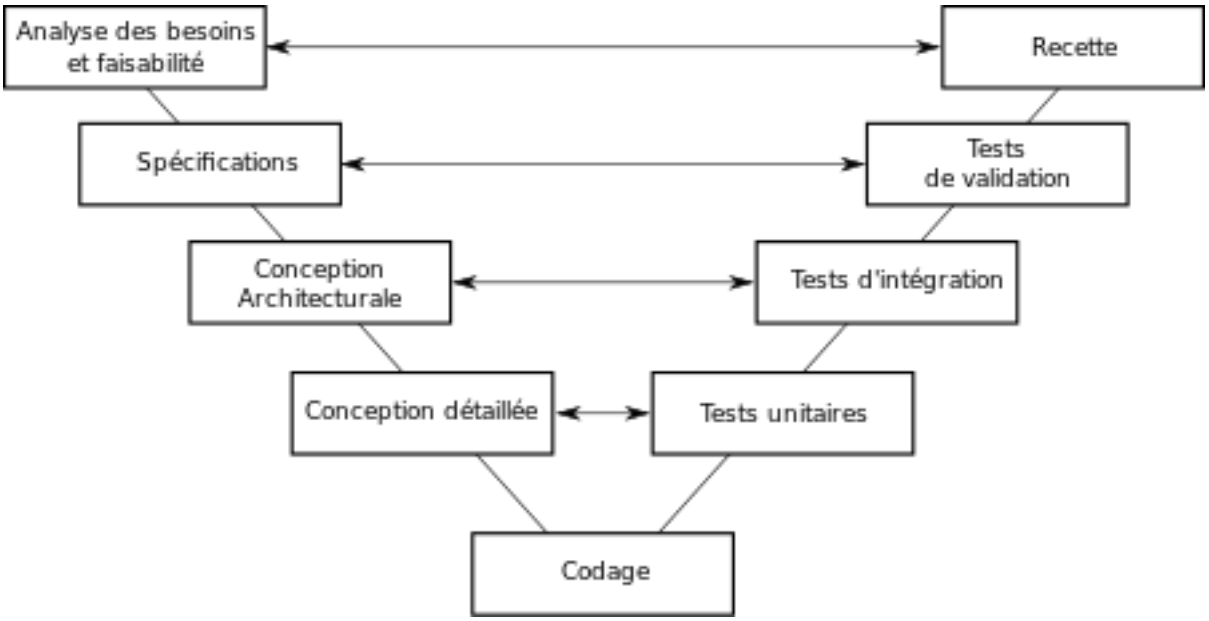

Figure IV.1 : Étapes du cycle en V

Le but de cette méthode est de limiter les retours aux étapes précédentes. En pratique, il est difficile voire impossible de totalement détacher la phase de conception d'un projet de sa phase de réalisation. C'est souvent au cours de l'implémentation qu'on se rend compte que les spécifications initiales étaient incomplètes, fausses, ou irréalisables, sans compter les ajouts de nouvelles fonctionnalités par les clients.

Il est difficile d'appliquer l'architecture des microservices avec cette méthodologie, car cette méthode ne permet pas de mises en production fréquentes. Le cycle en V comporte beaucoup d'étapes impliquant de nombreuses équipes. Une seule équipe indépendante et polyvalente dédiée a chaque microservice serait plus efficace, gérant elle même son cycle de recettes, ayant ses propres choix techniques, etc.

2. Méthode agile : SCRUM

SCRUM est une méthode agile itérative et incrémentale pour gérer le développement d'un produit. Elle a encourage l'équipe a s'auto-organiser en mettant en avant une forte communication par tous les membres de l'équipe, quelque soit sa discipline.

Cette méthode prend en compte le fait que le client change souvent d'avis, et que des imprévus peuvent arriver. Elle a pour cela une approche empirique, c'est-à-dire qu'elle accepte qu'un projet ne peut totalement être compris et bien défini. Pour cela, le fait de modifier souvent le produit permet de répondre rapidement aux satisfactions du client en s'adaptant a ses demandes émergentes. Les trois principaux rôles d'une équipe SCRUM sont :

- **Product owner (PO)** : Représente les parties prenantes et est la voix du client. Son rôle est de s'assurer que l'équipe apporte de la valeur ajoutée au produit.
- **Equipe de développement** : Equipe responsable d'ajouter de la valeur au produit à la fin de chaque itération
- **Scrum Master** : Facilitateur, celui qui s'occupe de ce qui ralentit l'équipe de développement. Il s'assure aussi que les rituels sont bien respectés

Le produit possède un *product backlog*. C'est un artefact contient la liste des *user stories* (fonctionnalités) à ajouter au produit. Le déceloppement de celui-ci se fait en plusieurs itérations (*sprint*). Chaque sprint possède son *sprint backlog* qui contient la liste des fonctionnalités à ajouter pour cette itération.

La méthode est constituée de différents événements et rituels importants à respecter. En effet, les respecter permet d'optimiser l'efficacité de chaque membre de l'équipe grâce a une bonne communication. Le résultat est que le produit est de bonne qualité, respectant les demandes du client. Chaque personne est responsable de son travail et du bon respect des rituels. Voici les principaux événements et rituels :

- **Sprint** : Itération avec une durée spécifique (time-box)
- **Sprint Planning** : Se déroule au début de chaque sprint, le but est d'estimer les tâches à

effectuer et d'assigner les tâches les plus prioritaires (jugé par le PO) au sprint en fonction de la capacité de l'équipe de développement (vélocité)

- **Daily Scrum** : Rencontre journalière de toute l'équipe SCRUM, où chaque membre dit le travail effectué la veille, le travaille qu'il effectuera ce jour et les problèmes rencontrés
- **Sprint review** : Présentation du résultat aux parties prenantes, et revue du travail effectué et non effectué
- **Sprint retrospective** : Réflexion sur le dernier sprint, identification de ce qu'il s'est bien passé, mal passé et comment le processus pourrait être amélioré. Permet aussi d'ajuster la vélocité de l'équipe

3. Méthode agile : Extreme Programming

La méthode SCRUM ne couvre aucune technique d'ingénierie logicielle. Dans le cas d'un développement d'application, il est nécessaire de la compléter avec des pratiques de qualité logicielle.

Extreme Programming (XP) est une méthode agile orientée sur l'aspect réalisation d'une application, sans pour autant négliger l'aspect gestion de projet. XP est adapté aux équipes réduites avec des besoins changeants. XP pousse à l'extrême des principes simples.

Le but principal est de réduire les coûts du changement. Dans les méthodes traditionnelles, les besoins sont définis et souvent fixés au début du projet, ce qui accroît les coûts ultérieurs de modifications. Pour éviter cela, XP définit des principes et des pratiques simple, mais poussés à "l'extrême" :

- puisque la revue de code est une bonne pratique, elle sera faite en permanence (par un binôme)
- puisque les tests sont utiles, ils seront faits systématiquement avant chaque mise en œuvre
- puisque la conception est importante, elle sera faite tout au long du projet (refactoring)
- puisque la simplicité permet d'avancer plus vite, nous choisirons toujours la solution la plus simple
- puisque la compréhension est importante, nous définirons et ferons évoluer ensemble des métaphores
- puisque l'intégration des modifications est cruciale, nous l'effectuerons plusieurs fois par jour
- puisque les besoins évoluent vite, nous ferons des cycles de développement très rapides pour nous adapter au changement

Voici le cycle de développement pour chaque scénario (fonctionnalité a développer) :

- Phase d'exploration déterminant les scénarios qui seront fournis pendant cette itération
- L'équipe transforme les scénarios en tâches à réaliser et en tests fonctionnels
- Chaque développeur s'attribue des tâches et les réalise avec un binôme
- Lorsque tous les tests fonctionnels passent, le produit est livré

XP decrit 12 pratiques, regroupées en 4 categories.

A. Retour d'information (feedback)

- Pair Programming : Programmer avec un partenaire
- Planning Game : Similaire au sprint planning de SCRUM
- TDD (Test Driven Developement) : Coder le test avant la fonctionnalité
- Équipe complète : Le client fait partie de l'équipe et se doit d'être disponible a tout moment

B. Process continu

- Intégration continue : Lancer les tests unitaire à chaque code rajouté par un développeur sur le gestionnaire de versions
- Refactoring : S'assurer que le code est efficace et sans duplication (bonne architecture)
- Petites releases : Mises en production fréquentes

C. Compréhension commune

- Standards de code : Avoir un style de code convenant a toute l'équipe
- Possession commune de code : Le code n'appartient a personne, mais a toute l'équipe
- Architecture simple : La philosophie de l'équipe est "Simple est mieux" (KISS : Keep It Simple Stupid). L'équipe doit toujours se demander "N'y a t il pas un moyen plus simple d'implémenter cette fonctionnalité ?".
- Métaphore du système : Toute l'équipe (client, manager, développeur, etc.) est capable de décrire comment le système fonctionne avec des mots simples.

D. Bien être du programmeur

Le dernier principe est "Rythme tenable". Les programmeurs ne devrait pas faire d'heures supplémentaires pour réaliser le travail. Un programmeur fatigué n'est pas un programmeur efficace et créatif. La qualité du code en sera grandement amoindrie.

4. Appliqué au projet

A. Ce que nous retenons des méthodes agiles

Pour notre équipe et notre projet, nous prenons inspiration de ces méthodes agiles. SCRUM définit de bonnes méthodes de gestion de projet, tandis que l'extreme programming fournit de nombreuses bonnes pratiques logicielles. Il faut faire très attention à ne pas perdre l'intérêt d'une des deux méthodes en ne les appliquant pas totalement.

Le mot "extrême" d'extreme programming peut paraître péjoratif, mais ce n'est finalement qu'une liste simple de bonnes pratiques à respecter pour avoir un logiciel de qualité. Pour notre projet, nous allons donc les utiliser, en plus des rituels et artefacts de SCRUM. Ce que nous retenons vraiment de ces deux méthodes et ce que nous prenons comme philosophie sont les points suivants :

1. Testing

Une des pratiques les plus importantes. Un logiciel testé permet de s'assurer que toute nouvelle fonctionnalité n'affecte pas le reste du système. Le fait d'écrire le test en premier (TDD) force le refactoring constant et assure une bonne conception dès le départ, limitant la dette technique.

2. Code Review

La revue de code par un autre collègue permet de s'assurer de la qualité de celui-ci. Cela donne du recul à celui qui l'a développé et lui permet de s'améliorer et d'affûter son esprit critique. Cela limite par la même occasion la possession du code par une seule personne. Si c'est difficile de faire du pair programming constamment, nous pensons qu'il est nécessaire qu'un code sur la branche principale soit revu au moins par deux personnes.

3. Ne jamais avoir de dette technique

La dette technique ralentit le développement et renforce la possibilité d'avoir des bugs dans le système. Nous faisons tout ce que nous pouvons pour l'éviter. Pour cela, il est important de respecter les bonnes pratiques de l'extreme programming (TDD, code review, etc.). Les développeurs doivent toujours être satisfaits du code sur la branche principale, et ne jamais penser que le problème pourra être réglé plus tard.

Si le travail est effectué sur un code existant de type legacy (code non testé comportant généralement une forte dette technique), nous appliquons la *boy scoot rule*. Celle-ci définit qu'un développeur qui touche a un morceau de code, doit forcement le rendre plus propre qu'il l'a laissé. Cela permet de réduire petit a petit la dette technique lorsque celle-ci est trop élevée.

4. Déploiements rapides, faciles, et fréquents

La procédure de déploiement en production doit être automatisée, et fréquente. Si cette procédure est rapide, les développeurs auront la possibilité d'ajouter des fonctionnalités et corriger les bugs facilement. L'automatisation est un énorme gain de temps. La présence de nombreux tests permet aussi de s'assurer que le déploiement s'est effectué sans problèmes grâce à l'intégration continue.

5. Utiliser les bons outils

Tout outil permettant de gagner du temps doit être utilisé (environnement de développement, etc.). Il en est de même des artefacts de SCRUM tels que la vélocité. Etre capable de mesurer la capacité de travail de l'équipe est très important.

6. Une bonne communication

Une bonne communication tous les jours avec les autres membres de l'équipe permet de ne pas avoir un membre bloqué sur une problématiques pendant plusieurs jours. Une équipe soudée, qui s'occupe rapidement des problèmes est une équipe efficace.

7. Bien être des membres de l'équipe

Une équipe bien dans sa peau, bien reposée, et ayant de l'intérêt pour le travail effectué est une équipe productive. Pour cela, il ne faut pas que les membres aient de nombreuses heures supplémentaires à effectuer. La vélocité de l'équipe est présente pour mesurer sa capacité à travailler aux heures traditionnelles. Elle est plus facile à mesurer sur des intervalles de tempes courts (contrairement au cycle en V possédant de nombreuses étapes). Un système bien testé limite les problèmes en production et les besoins d'effectuer des heures supplémentaires.

Pour les développeurs, avoir du temps dégagé pour effectuer de la veille technologique est très important. Cela permet d'avoir des développeurs toujours affûtés techniquement et renforce leur créativité et leur compétence technique, ainsi que l'intérêt pour le travail effectué. Cela peut paraître sans valeur ajoutée pour le produit, mais c'est en réalité ce qui pourra faire la différence dans la réalisation de celui-ci, la rendant plus originale et moderne. Un peu de veille tous les jours ou même une après-midi par semaine dans les périodes creuses ont de grands bénéfices. Organiser des ateliers de partage de connaissance (exemple : coding dojo) rend la chose encore plus efficace, et permet aussi d'améliorer les talents de communication des membres de l'équipe.

B. Différents types de tests

Nous avons vu que le testing est très important dans la réalisation d'un logiciel. Il existe cependant plusieurs types de tests possédant un intérêt et un coût différents.

1. Pyramide des tests

La pyramide de tests est un concept créé par Mike Cohn, dans son livre "Succeeding with Agile"[4]. Le point clé qu'il essaye de démontrer est qu'un logiciel devrait avoir beaucoup plus de tests unitaires de bas niveau (faciles à écrire), que de tests de haut niveau (difficiles à écrire). Attention, il ne dit pas que les tests de haut niveau sont sans intérêt, il pense juste que l'on devrait écrire plus de tests faciles à écrire que de tests difficiles à écrire (temps perdu). Tout est une question de rapport temps passé / intérêt.

Les tests de haut niveau sont différents selon la partie technique implémentée, dans tous les cas on parle souvent de tests fonctionnels. Ce sont les plus proche du résultat final, représentant une fonctionnalité de l'application. Ce sont généralement les tests que l'on écrit en premier. Dans le cas du front-end, ce sont généralement des tests d'interface graphique, et pour le back-end, ce

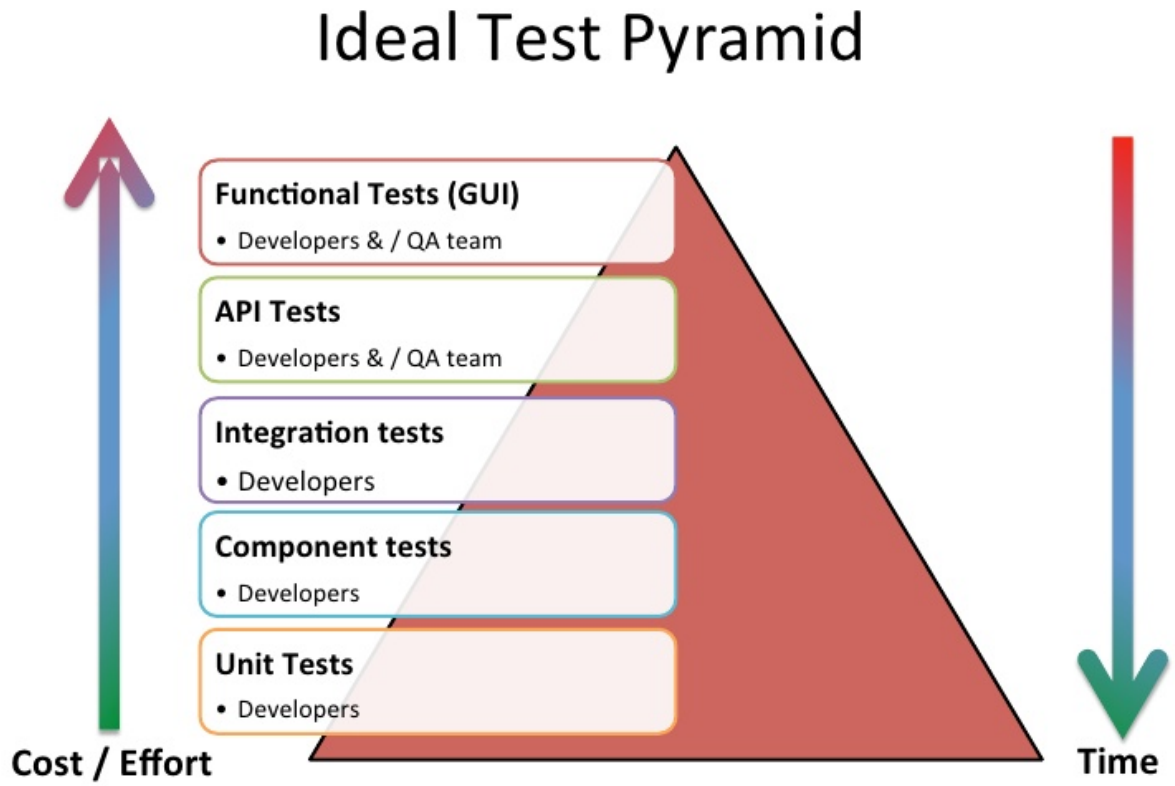

Figure IV.2 : Pyramide des tests

sont des tests d'API ou des tests d'intégration.

Un test unitaire substituera les dépendances (*mock*), ce qui n'est généralement pas le cas des tests de haut niveau. Ils sont beaucoup plus faciles à écrire car grâce aux mocks, la fonction testée retournera toujours le même résultat. Les tests de haut niveau utilise les connections réelles, le résultat est beaucoup moins prévisible, mais plus proche de la réalité.

Il faut retenir ceci : Un test unitaire permet de vérifier qu'une méthode est capable de gérer plusieurs cas bien définis, et un test de haut niveau vérifie qu'une fois le tout relié, l'ensemble fonctionne.

2. TDD (Test Driven Developement)

Le TDD (*Test Driven Development*) ou développement piloté par les tests est une méthode de développement prônant la répétition et les cycles de développement courts. Si au départ, le TDD était une pratique de l'extreme programming, c'est aujourd'hui une pratique plus généralisée.

Le principe est le suivant :

- **Rouge** : Le développeur écrit un test qui va définir l'amélioration ou la création d'une fonction (le test va automatiquement échouer, on dit que le test est rouge)
- **Vert** : Le développeur écrit ensuite le code minimum pour faire réussir le test (le test passe au vert)
- **Refacto** : Il va ensuite réfactorer le code jusqu'à ce qu'il soit satisfaisant (standards acceptables par l'équipe). Il vérifie que ces changement n'ont impacté aucun autre test (pas de régression)

Selon Kent Beck, connu pour avoir défini cette technique dans son livre "Test Driven Development : By Example"[3], le TDD encourage une architecture simple et efficace. Cette technique parait étrange pour beaucoup de développeurs, car ce n'est pas la solution qui est écrite en premier, mais le test la validant. Cela met plus en valeur ce qui manque à la fonction, et le fait d'utiliser la fonction avant de la créer renforce la réflexion sur la bonne conception de celle-ci.

Dans le cas d'une nouvelle fonctionnalité a rajouter, résoudre le test signifie que la fonctionnalité a bien été implémentée. Dans le cas d'un bug, il faut écrire un test mettant en valeur le cas ou la fonction échoue, et faire passer le test signifie que le bug est corrigé. Dans tous les cas, la dernière étape est la plus importante, car c'est celle qui assure qu'en plus du fait que la fonction est bien implémentée, elle est aussi de bonne qualité.

C. Déploiement continu

L'intégration continue vérifie que toute modification du code source ne produit aucune régression au système en lançant une phase tests à chaque nouveau morceau de code ajouté. Le principal but de cette pratique est de détecter les problèmes d'intégration au plus tôt lors du développement. Cette solution est de plus en plus utilisée en entreprise car elle améliore la qualité du code et du produit final.

Le déploiement continu fait partie de l'extreme programming. C'est un niveau au dessus de l'intégration continue car elle effectue en plus de l'intégration, le déploiement automatisé de l'application si tous les tests sont passés.

Il y a généralement un serveur d'intégration qui surveille le gestionnaire de versions. Il existe de nombreuses solutions d'intégration continue, nous avons utilisé Jenkins pour notre projet. Le fait d'utiliser Docker simplifie grandement l'automatisation des déploiements.

5. Démonstration du développement d'une fonctionnalité

Voici notre méthodologie concrète employée pour ajouter une fonctionnalité au produit.

- Création d'une nouvelle branche sur le gestionnaire de versions (évite le conflit avec les autres nouvelles fonctionnalités implémentées en parallèle)
- Écriture de tests de haut niveau (Intégration ou UI[1] selon le cas)
- Écriture du code source en TDD strict (il est interdit d'ajouter du code source sans la mise en valeur du manque par un test)
- Code review par un autre membre de l'équipe
- Intégration de la nouvelle branche sur la branche principale
- Vérification de la non régression grâce à l'intégration continue

[1] User Interface

V

Résultats du projet

Ayant appliqué ces principes (architecture, méthodologies, technologies) sur notre projet durant 2 mois avec notre équipe de 3 développeurs, nous sommes arrivés à un résultat satisfaisant. Nous allons maintenant voir jusqu'où le développement est allé et si les objectifs ont étés atteints.

1. Résultat

Grâce à la technologie Cordova, nous avons créé une application fonctionnant sur plusieurs terminaux mobiles (iOS, Android) ainsi que sur les navigateurs web. En voulant imiter le fonctionnement de l'application Android, nous avons eu le temps d'implémenter les fonctionnalités suivantes :

- **Accueil** : Contient les dernières offres promotionnelles du moment
- **Navigation** : Navigation par catégorie (Informatique, Immobilier, etc.)
- **Fiche produit** : Fiche détaillée d'un produit

Nous n'avons pas eu le temps d'implémenter la gestion du panier, des commentaires, etc. C'étaient les futures fonctionnalités à implémenter sur la liste. Le tout utilise bien évidemment les données réelles de Cdiscount, ayant ainsi un comportement similaire aux applications actuelles.

Le résultat est selon nous satisfaisant, avec de bonnes performances, et une interface utilisateur simple et efficace. Nous avons même tenté d'innover un peu et de faire certaines choses différemment de l'application de base. Au niveau de l'implémentation des microservices, nous en sommes à la première étape.

2. Challenges

Le fonctionnement actuel de Cdiscount n'est pas pensé pour les SPA. Par exemple, la solution de tracking interne s'effectue en se basant sur les applications web traditionnelles. Cela ne nous

a pas dérangé durant le développement du projet, mais c'est un point à prendre en compte si nous souhaitons industrialiser cette solution à Cdiscount. Il faudra alors trouver un moyen d'intégrer l'utilisation du tracking interne avec notre solution.

Nous avons parfois rencontré les soucis de performances engrangés par les webviews et AngularJS. Avec une bonne combinaison de librairies et grâce aux accélérateurs de webview, l'application est maintenant d'une très bonne fluidité.

Le challenge technique était important, surtout au niveau de l'implémentation des microservices. Deux mois de développement c'est finalement très court, mais nous avons pu fournir un travail conséquent grâce aux méthodologies appliquées. C'est cependant pour nous un sujet passionnant et sommes satisfaits du résultat actuel.

3. Objectifs atteints ?

Lors de la présentation finale du projet, le jury non technique était satisfait du résultat et pense qu'après quelques ajouts pour concorder aux méthodes de Cdiscount (tracking, etc.), il sera possible de tester notre solution sur des projets transverses, permettant d'utiliser au mieux notre expertise des technologies du web.

Le jury technique quant-à lui était satisfait de voir que nos méthodes ont porté leurs fruits et que notre logiciel est de qualité au niveau de son implémentation. L'architecture et la méthodologie employées correspondent totalement à l'évolution de Cdiscount.

Conclusion

Le projet de démonstration est un succès et a atteint ses objectifs. C'est un logiciel de qualité, permettant l'implémentation rapide de nouvelles fonctionnalités sans accumuler de dette technique. Nous avons par la même occasion démontré à Cdiscount qu'il est possible de créer des applications mobiles en utilisant uniquement les technologies web, notre domaine d'expertise. Lors de la présentation, le jury s'en est bien rendu compte et souhaite tester notre solution sur des applications transverses. Il faut avant cela, que nous finalisons l'intégration avec les méthodes actuelles de Cdiscount.

Nous retenons que pour créer un logiciel de qualité, il faut choisir une architecture qui permet d'éviter que l'application se transforme en monolithe et le style d'architecture des microservices est la meilleure solution actuellement. Son principe est très simple : plus une fonctionnalité est petite, simple et isolée, moins elle aura de chance de se transformer en monolithe. Une service simple est simple à comprendre et à améliorer. Cdiscount a bien raison de vouloir l'intégrer dans son SI, et notre projet est un bon exemple.

Le DDD (Domain Driven Design) nous a appris qu'il existe en fait deux types de logiques dans une application : logique de domaine et logique applicative. Le fait de les séparer permet une meilleure liberté technologique. C'est d'autant plus utile dans le domaine du web car les technologies évoluent très vite. Il est important d'effectuer une veille technologique active pour pouvoir en profiter.

SCRUM et Extreme Programming (XP) allient parfaitement gestion de projets et bonnes pratiques de développement. XP met beaucoup en avant le bien-être du développeur, et met tout en place pour cela. Nous recommandons fortement leur utilisation lors du développement d'applications, c'est selon moi la clé pour avoir un logiciel de qualité.

Le travail d'un architecte logiciel est la conception d'applications. Le travail d'un développeur de logiciel est la réalisation d'applications. Nous avons vu qu'il est difficile de séparer la phase de conception d'un projet de sa phase de réalisation. C'est le point faible de la méthodologie du cycle en V, et les méthodes agiles ont pour but de rallier ces deux phases. De nombreuses entreprises séparent ces deux rôles, et la qualité du logiciel est en amoindrie.

Un ingénieur logiciel est capable de concevoir ET réaliser une application. Le meilleur chemin pour avoir un logiciel de qualité est d'avoir une équipe d'ingénieurs logiciels, travaillant ensemble durant toutes les phases du projet (conception, réalisation et mise en oeuvre).

Glossaire

API

Une interface de programmation (souvent désignée par le terme API pour Application Programming Interface) est un ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels. Elle est offerte par une bibliothèque logicielle ou un service web, le plus souvent accompagnée d'une description qui spécifie comment des programmes consommateurs peuvent se servir des fonctionnalités du programme fournisseur. (Wikipédia). 5, 37, 55

Coding dojo

Le coding dojo est une rencontre entre plusieurs personnes qui souhaitent travailler sur un défi de programmation de façon collective. Le défi peut être un problème algorithmique à résoudre ou un besoin à implémenter. Chaque coding dojo se concentre sur un sujet particulier, et représente l'objectif de la séance. Ce sujet doit permettre d'apprendre de façon collective sur le plan technique et sur la manière de réussir le défi. L'exercice peut être effectué entre personnes d'une même entreprise, d'une école ou encore venant d'horizons différents. (Wikipédia). 5, 54

CSS

Les feuilles de style en cascade, généralement appelées CSS de l'anglais Cascading Style Sheets, forment un langage informatique qui décrit la présentation des documents HTML et XML. Les standards définissant CSS sont publiés par le World Wide Web Consortium (W3C). (Wikipédia). 12, 32, 35, 41

Design pattern

un patron de conception (plus souvent appelé design pattern) est un arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel. Il décrit une solution standard, utilisable dans la conception de différents logiciels. (Wikipédia). 19

DOM

Standard du W3C qui décrit une interface indépendante de tout langage de programmation et de toute plate-forme, permettant à des programmes informatiques et à des scripts d'accéder ou de mettre à jour le contenu, la structure ou le style de documents HTML et XML. (Wikipédia). 32, 35, 41

Framework

Un framework ou structure logicielle est un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture). (Wikipédia). 3, 32, 35, 40–43, 46

HTML

L'Hypertext Markup Language, généralement abrégé HTML, est le format de données conçu pour représenter les pages web. C'est un langage de balisage permettant d'écrire de l'hypertexte, d'où son nom. HTML permet également de structurer s□□mantiquement et de mettre en forme le contenu des pages, d'inclure des ressources multimédias dont des images, des formulaires de saisie, et des programmes informatiques. (Wikipédia). 12, 31, 32, 35, 41, 42

HTTP

L'HyperText Transfer Protocol, plus connu sous l'abréviation HTTP — littéralement « protocole de transfert hypertexte » — est un protocole de communication client-serveur développé pour le World Wide Web. (Wikipédia). 7, 33, 36, 37, 46

IDE

Un environnement de développement (abrégé EDI en français ou IDE en anglais, pour Integrated Development Environment) est un ensemble d'outils pour augmenter la productivité des développeurs d'application.. 17, 29

JSON

JSON (JavaScript Object Notation) est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple. (Wikipédia). 33

Liens hypertextes

Un hyperlien, ou lien hypertexte, ou lien web, ou simplement lien, est une référence dans un système hypertexte permettant de passer automatiquement d'un document consulté à un document lié. Les hyperliens sont notamment utilisés dans le World Wide Web pour permettre le passage d'une page Web à une autre à l'aide d'un clic. (Wikipédia). 7

Load-balancing

La répartition de charge (en anglais : load balancing) est un ensemble de techniques permettant de distribuer une charge de travail entre différents ordinateurs d'un groupe. Ces techniques permettent à la fois de répondre à une charge trop importante d'un service en la répartissant sur plusieurs serveurs. (Wikipédia). 47

Open source

La désignation open source, ou « code source ouvert », s'applique aux logiciels dont la licence respecte des critères précisément établis par l'Open Source Initiative, c'est-à-dire les possibilités de libre redistribution, d'accès au code source et de création de travaux dérivés. (Wikipédia). 2, 9, 10, 41, 43, 45

Pizza teams

Une équipe polyvalente de taille 8 maximum (nombre de parts d'une pizza).. 4, 14, 16, 48

POC

Proof Of Concept : Veut démontrer un concept sans pour autant être achevé. 1, 14, 15

Polyfills

En programmation web, un polyfill est un ensemble de fonctions, le plus souvent écrites en Javascript ou en Flash, permettant de simuler sur un navigateur web ancien des fonctionnalités qui ne sont pas nativement disponibles. Par exemple, accéder à des fonctions HTML5 sur des navigateurs ne proposant pas ces fonctionnalités. (Wikipédia). 35, 42

Product Owner

Représente les parties prenantes et est la voix du client. Son rôle est de s'assurer que l'équipe apporte de la valeur ajoutée au produit. 3

Responsive web design

Un site web adaptatif (anglais RWD pour responsive web design, conception de sites web adaptatifs selon l'OQLF1) est un site web dont la conception vise, grâce à différents principes et techniques, à offrir une expérience de consultation confortable même pour des supports différents. L'utilisateur peut ainsi consulter le même site web à travers une large gamme d'appareils (moniteurs d'ordinateur, smartphones, tablettes, TV, etc.) .(Wikipédia). 12

SEO

L'optimisation pour les moteurs de recherche (en anglais : Search engine optimization, SEO) est un ensemble de techniques visant à favoriser la compréhension de la thématique et du contenu d'une ou de l'ensemble des pages d'un site Web par les moteurs de recherche. (Wikipédia). 42

SI

Un système d'information (SI) est un ensemble organisé de ressources qui permet de collecter, stocker, traiter et diffuser de l'information (Wikipédia). 1, 3, 16

Web service

Un service web (ou service de la toile) est un programme informatique de la famille des technologies web permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. (Wikipédia). 5

XML

L'Extensible Markup Language (XML, « langage à balise extensible » en français) est un langage informatique de balisage générique. (Wikipédia). 11, 33, 37

Table des figures

Table des matières

Bibliographie

[1] "AngularJS : le framework JavaScript de Google au crible". In : (2015). url : http://www.journaldunet.com/developpeur/outils/angular-js.shtml (cf. p. 44).

[2] "Architectural Patterns and Styles". In : *MSDN : Microsoft Developer Network* (). url : https://msdn.microsoft.com/en-us/library/ee658117.aspx (cf. p. 18).

[3] Kent Beck. *Test Driven Development : By Example* (cf. p. 56).

[4] Mike Cohn. *Succeeding with Agile : Software Development Using Scrum* (cf. p. 54).

[5] "Conference : DDD and Microservices : At last, some boundaries !" In : (2015). url : https://skillsmatter.com/skillscasts/6259-ddd-and-microservices-at-last-some-bounderies (cf. p. 30).

[6] "De l'application monolithique aux architectures microservices". In : (2014). url : http://juliendubreuil.fr/blog/developpement/de-application-monolithique-aux-architectures-microservices-ou-orientees-composants/ (cf. p. 17).

[7] "Desktop Search Engine Market Share". In : (2015). url : https://www.netmarketshare.com/search-engine-market-share.aspx?qprid=4&qpcustomd=0 (cf. p. 34).

[8] Eric Evans. *Domain-Driven Design : Tackling Complexity in the Heart of Software* (cf. p. 26).

[9] David Garlan et Mary Shaw. *An Introduction to software architecture*. 1994 (cf. p. 18).

[10] "La filiale de Casino, Cnova (Cdiscount), a fait une entrée décevante à Wall Street". In : *capital* (2014). url : http://www.capital.fr/bourse/actualites/la-filiale-de-casino-cnova-cdiscount-a-fait-une-entree-decevante-a-wall-street-993220 (cf. p. 2).

[11] "Microservices". In : (2014). url : http://martinfowler.com/articles/microservices.html (cf. p. 29).

[12] "Pattern : Monolithic Architecture". In : (2014). url : http://microservices.io/patterns/monolithic.html (cf. p. 17).

[13] "Why Java Developers Hate JavaScript". In : (2011). url : https://dzone.com/articles/why-java-developers-hate (cf. p. 9).

[14] "Why we Need WebAssembly, An Interview with Brendan Eich". In : (2015). url : https://medium.com/javascript-scene/why-we-need-webassembly-an-interview-with-brendan-eich-7fb2a60b0723 (cf. p. 9).