École Polytechnique de Montréal
Département de Génie Informatique et Génie Logiciel

LOG2410

Conception logicielle

Hiver 2017

Travail personnel : Sujet #1

Architecture micro-services

Remis par :

Matricule	Prénom & Nom
1748125	David Tremblay

À : François Guibault

Le 18 avril 2017

Table des matières

Liste des figures

Liste des tableaux

1 Introduction

Planifier un projet d'envergure en informatique est loin d'être une tâche simple. En effet, la conception du produit correspond en moyenne à près de la moitié des efforts mis dans un projet (entre 40 et 45%). Plusieurs outils ont été inventés pour faciliter ce travail qui devient incontournable avec l'ascension du domaine informatique dans notre quotidien. Un de ces outils est la planification de l'architecture logicielle qui représente les relations et interactions entre les divers composants du système d'un projet pour nous orienter sur la bonne voie pour la suite. Plusieurs modèles d'architecture logicielle existent en génie informatique et logiciel ayant ces avantages et désavantages. Citons par exemple l'architecture orientée objet, l'architecture en flot de données ou l'architecture orientée agent. Dans le monde du travail, un des modèles très utilisés traditionnellement est l'architecture N-tier aussi nommé architecture en couches ou par niveaux. Par contre, depuis environ les cinq dernières années, on constate le gain en popularité d'une nouvelle architecture nommée l'architecture micro-services qui est une spécialisation de l'architecture orientée service. Quelle est la philosophie de ces deux modèles de conception et quels sont les avantages et les inconvénients des deux architectures qui pourraient expliquer le choix d'un ou l'autre lors d'un projet informatique? La suite de ce travail sera consacrée à tenter de répondre à cette question très pertinente en lien au processus de conception logicielle. Le développement en silo représente une méthode de travail ou plusieurs petites équipes travaillent chacune sur une partie d'un projet en parallèle et pour finir regroupent le tout pour aboutir au but. À l'opposé, le travail collaboratif est une méthode de travail ou une grande équipe travaille ensemble sur le même objectif afin de profiter pleinement des compétences de chaque membre. Le travail expliquera aussi ce que l'approche basée sur des micro-services a comme effet sur le développement en silo lors d'un projet.

2 Architecture par couche

L'architecture par couche est une architecture client-serveur où la présentation, le processus de l'application et les données sont séparés. La philosophie d'une architecture client-serveur est de partitionner les tâches entre le client et le serveur qui représente le fournisseur du service. Le client, souvent étant l'ordinateur de l'utilisateur, envoie des requêtes et les serveurs reçoivent ces requêtes et y répondent de manière appropriée. L'architecture par couche est aussi nommée architecture N-tier où N représente le nombre de tiers en lesquelles l'application a été répartie. La plus utilisée est de loin l'architecture 3-tier. Elle est utilisée dans la plupart des grands sites de cybercommerces tels que ebay ou amazon. Le tier du haut représente l'interface qui est présentée à l'utilisateur lorsqu'il utilise l'application. On peut faire un lien avec le cours LOG2420: Analyse et conception d'interfaces utilisateurs. Cette couche sert en effet à présenter une interface qui affiche les tâches à l'utilisateur de manière à faciliter son utilisation et augmenter le taux de succès. La deuxième couche, la couche logique, sert à effectuer les opérations arithmétiques et logiques de l'application et à coordonner la communication entre les données et la présentation en servant d'intermédiaire. Finalement, la dernière couche est pour tout ce qui a trait au stockage de données. Les bases de données sont l'élément principal de cette couche. Il est pertinent de mentionner que ces couches sont logiques et non physiques. L'application n'a donc pas à être séparée en trois entités physiques distinctes. C'est par contre ce que la plupart des applications font en ayant par exemple, une entité physique de serveurs qui servent de bases de données séparées de la logique de l'application. Une question légitime à poser lorsqu'on regarde le schéma de l'architecture 3-tier est qu'est-ce qui la différencie du fameux modèle Modèle-Vue-Contrôleur (MVC). En effet, on pourrait penser que les deux désignent le même modèle puisque les deux possèdent trois couches qui fonctionnent de la même façon. Cependant, le modèle MVC ne concerne que la façon de présenter des informations et son modèle est triangulaire. Il peut aussi bien s'appliquer à n'importe quelle architecture N-tier. Une architecture N-tier, quant à elle, est pyramidale et n'est pas seulement pour la présentation. La couche logique doit absolument servir d'intermédiaire entre les deux autres tiers et est donc moins

permissive. La figure 1 ci-dessous résume ce qui a été dit et représente le schéma d'une architecture en couche (elle est en anglais mais tout aussi pertinente).

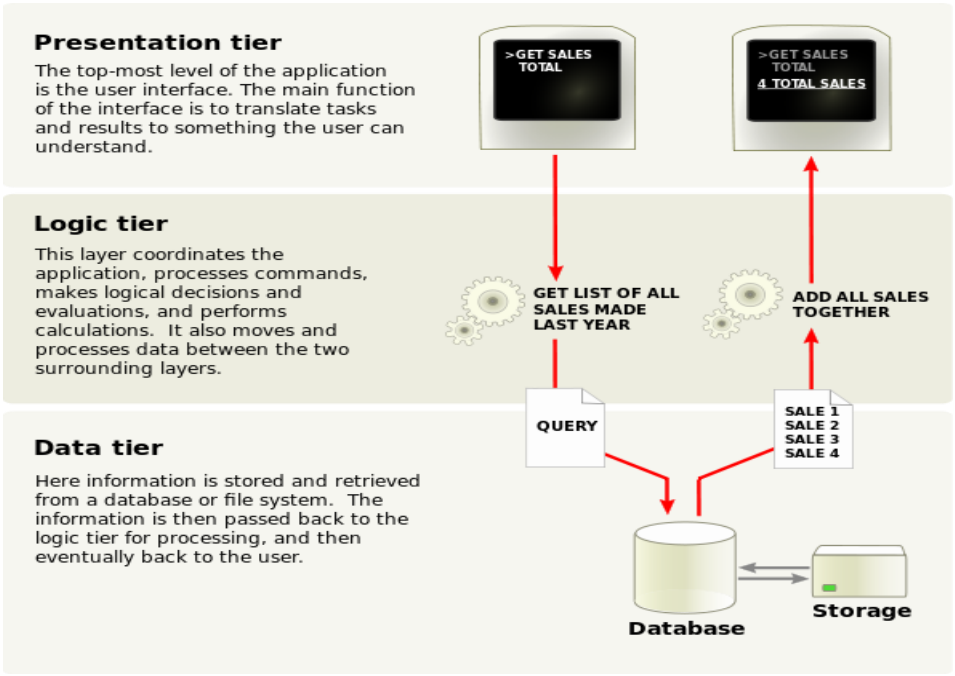

Figure 1: Schéma de l'architecture N-tier

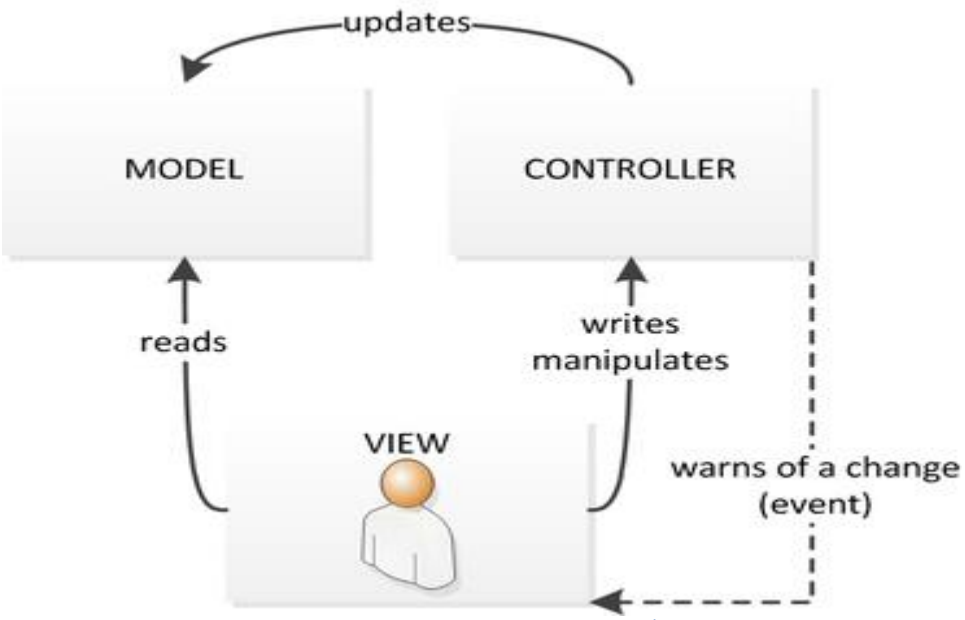

Figure 2: Modèle-Vue-Contrôleur[1]

[1] Les figures 1 et 2 proviennent des références 5 et 6 respectivement

3 Architecture micro-services

L'architecture micro-services suit la philosophie de l'architecture orientée services (souvent appelée SOA pour service oriented architecture) qui est apparue au début du millénaire. Cette philosophie est souvent utilisée dans les sites de cybercommerces qui ont connu un gain de popularité fulgurant depuis les dernières années. Pour une application, le client ne s'intéresse seulement qu'au résultat (acheter un article en ligne par exemple) et c'est au fournisseur de fournir le service qui permettra de réaliser la tâche sans que le client ait besoin de sa voir son fonctionnement. Dans l'architecture micro-service, l'application est séparée en plusieurs services simples ayant chacun un rôle bien précis qui représente une fonctionnalité élémentaire. Le couplage interne est donc élevé et le couplage externe est faible. Chacun de ses composants communique entre eux à travers un réseau ou un API à l'aide de protocoles tels que le HTTP. Les fonctionnalités étant développées séparément, l'architecture micro-services prône le déploiement continu et le réusinage de code tout au long du projet. Les nouveaux membres du projet n'ont pas à apprendre le fonctionnement de tout le projet mais seulement la partie sur laquelle ils vont travailler pour être productif plus rapidement dans un environnement où le roulement de personnel est non négligeable. La philosophie de l'architecture micro-services est souvent comparée à la philosophie du système d'exploitation UNIX qui est : « Do one thing and do it well ». On concentre donc chaque composant sur une tâche ciblée. Il faut parfois faire attention de ne pas faire trop de tâches simples. Une telle erreur peut mener dans ce que certains appellent une architecture nano-services qui rend l'application moins efficace à cause du trop grand nombre de composants. Chaque composant dispose de son propre espace de stockage et d'un processus qui lui est propre. On peut voir à l'aide de la figure 3 ci-jointe que les fonctionnalités de l'architecture monolithique ont été séparées en des composants distincts chacun ayant son propre espace de stockage et qui représente l'architecture micro-services.

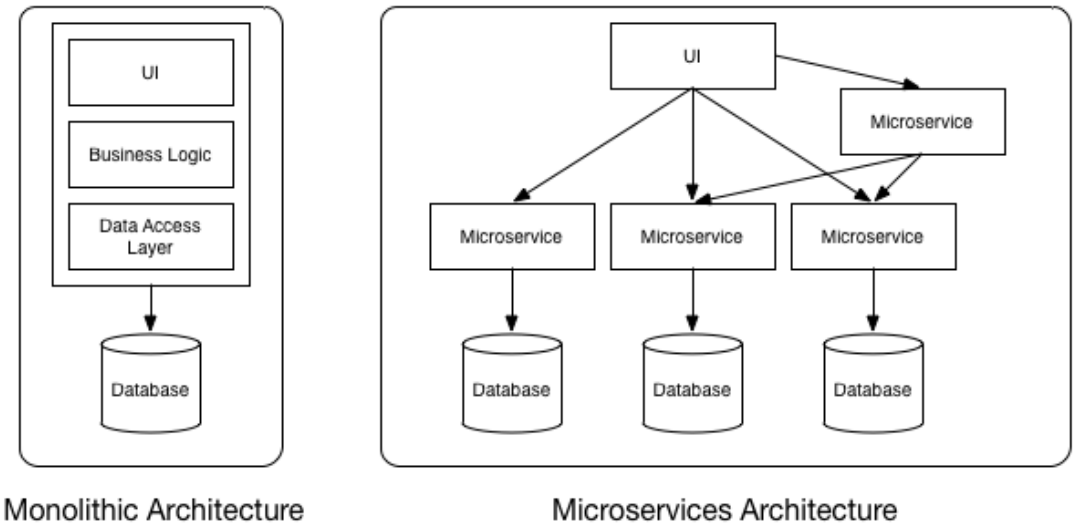

Figure 3: Schéma d'une architecture micro-services[2]

4 Comparaison architecture en couche / architecture micro-services

Maintenant que les deux architectures ont été bien expliquées, nous pouvons les comparer et détailler les avantages et inconvénients de chacune de celles-ci. Premièrement, puisque des protocoles sont utilisés pour réaliser le service demandé par le client et que le couplage est faible entre les composants, la réutilisabilité des services est très grande et il est donc très facile de réutiliser des morceaux de l'application ailleurs comme dans une autre application par exemple. Un autre avantage qui découle du faible couplage externe est qu'il est facile de remplacer un service par un autre. Ceci est donc très utile si on veut utiliser une nouvelle technologie pour un service en particulier ou si on veut faire évoluer le service vers cette technologie. Puisque chaque composant peut être réalisé sur une technologie différente, on peut utiliser le meilleur outil pour chacune des tâches et tirer avantage du plein potentiel de celles-ci. On pourrait par exemple utiliser du C++ pour tirer plein avantage de l'héritage dans un composant et utiliser le FORTRAN, Java ou tout autre type de codage pour une autre fonctionnalité ou ces types de codage apporteraient de meilleurs avantages. Les fonctionnalités étant toutes séparées, il est facile d'ajouter une nouvelle fonctionnalité

[2] La figure 3 provient de la référence 8

plus loin lors du projet. On a qu'à assigner une nouvelle équipe à cette fonctionnalité et elle travaillera en parallèle du projet à son développement. Ensuite, la méthode de travail entre ces deux architectures est très différente. Puisque le micro-service sépare l'application en de multiples composants, le travail en silo est utilisé pour ce mode de développement. De plus, dû au faible couplage entre les composants, les petites équipes de travail peuvent travailler en parallèle. Chaque composant ayant ses propres processus et espaces de stockage, les tests de chaque composant peuvent être réalisés indépendamment des autres. C'est tout à fait différent dans l'architecture en couche. Effectivement, dans cette architecture, l'application est un composant monolithique et les équipes de travaux sont beaucoup plus grosses. Les tests ne peuvent donc ne pas être isolés et il est plus difficile de travailler en parallèle. Il n'y a pas que des torts à cette architecture par contre. En effet, chaque composant communiquant à l'aide d'un protocole dans l'architecture micro-services entraîne une gestion du réseau beaucoup plus coûteuse et difficile à gérer que l'architecture par niveaux. Une critique courante de l'architecture micro-service est qu'elle ne fait que séparer la complexité de l'application dans différents composants sans l'améliorer. Il faut aussi faire attention de ne pas faire trop de fonctionnalités primaires et de tomber dans le nano-service tel qu'expliquer ci-haut. L'architecture en couche est plus facile à tester dans son ensemble tandis que l'architecture micro-services est plus facile à tester pour chaque composant. C'est aussi plus simple dans l'architecture par couches au niveau de la conception puisqu'on n’a pas à identifier toutes les fonctionnalités primaires qui seront développées et les protocoles. Pour tester plusieurs composants dans une architecture micro-service, il faut démarrer chaque composant requis pour un service et cela rend la tâche beaucoup plus fastidieuse. Une architecture monolithique est aussi plus facile à déployer puisque l'application est toute dans un même bloc. L'architecture micro-service demande beaucoup plus de travail et un grand niveau d'automatisation afin de déployer sur plusieurs serveurs. Dans une architecture micro-service, on peut mettre à jour un composant de l'application mais dans une architecture N-tier, toute l'application doit être mis à jour (puisque c'est un bloc). Il peut être aussi plus difficile de comprendre le

code et de faire sa maintenance s'il n'est pas séparé en composant et l'application est plus longue à démarrer puisque toute l'application doit être démarrée. Une défaillance d'un des composants d'une application monolithique entraîne une défaillance de l'application au complet tandis que si le projet est réalisé selon les principes de l'architecture micro-services, seulement le composant défectueux est affecté et est à corriger. On peut créer des systèmes automatiques pour caque composant en cas de panne qui permet facilement à l'application de continuer à fonctionner avec les autres composants et donc d'offrir un service de meilleure qualité au client. Lorsqu'on dispose d'un bloc monolithique représentant notre application, il faut s'assurer que l'application dispose du CPU nécessaire à son bon fonctionnement et il faut donc fournir le maximum demandé par l'application même si plusieurs parties de celle-ci requièrent beaucoup moins à l'ordinateur au niveau du CPU. Par contre, pour une application micro-service, on peut fournir le CPU nécessaire à chacun des composants selon leurs besoins. La réutilisabilité engendrée par l'architecture micro-services est aussi plus adaptée au monde technologique qui est continuellement en changement tandis que l'architecture en couche a plus de difficulté pour adapter les applications. L'architecture micro-service étant un phénomène relativement nouveau comparativement à l'architecture par niveaux, les outils *open source* pour faciliter ce type de projet sont aussi beaucoup moins nombreux et entraîne un travail supplémentaire aux développeurs qui doivent les faire eux-mêmes.

Les principaux avantages et désavantages des deux architectures mentionnées ci-haut sont regroupés dans le tableau ci-dessous afin de résumer et illustrer de manière plus concrète ce qui fera qu'un projet sera réalisé à l'aide d'une architecture par niveaux ou micro-services. À noter toutefois que le texte ci-haut est plus exhaustif dans sa comparaison des deux architectures.

Tableau 1: Avantages et désavantages des architectures

	Architecture par niveaux	Architecture micro-services
Réutilisabilité	Faible	Grande puisque le couplage externe est faible
Mode de travail	Grandes équipes qui travaillent ensemble	Beaucoup de petites équipes (en silo) et travail parallèle
Tests	Facile de tester le programme au complet mais difficile de tester un composant en particulier	Facile de tester un composant indépendant mais difficile de tester plusieurs composants simultanément
Gestion du réseau	Pas un problème	Beaucoup de protocoles à gérer. Très complexe et utilise des ressources considérables
Complexité	On voit la complexité réelle du code	Complexité cachée à travers le code mais toujours présente. Trop faire de composants peut aussi entraîner une architecture nano-services qui augmente la complexité au final
Conception	Architecture beaucoup plus simple à planifier	Architecture complexe où chaque composant doit être correctement identifié
Déploiement	On déploie le bloc monolithique d'un coup sur le serveur	Plus complexe et demande un grand niveau d'automatisation pour déployer
Mis à jour	Tout le programme doit être mis à jour	On peut mettre seulement un composant à jour
Maintenance	Plus difficile de comprendre le code quand tout est dans un bloc	Fonctionnalités séparées et le code est donc plus facile à

		maintenir
Adaptation	Faible réutilisabilité et donc difficile de s'adapter pour une nouvelle technologie	Facilement apte aux changements
Outils disponibles	Beaucoup de documentation et d'outils pour faciliter le travail	Relativement nouveau donc peu d'aide disponible

5 Architecture micro-services et développement en silo

Le développement en silo est une méthode de travail qui consiste à séparer un problème en sous-problèmes pour ensuite les résoudre séparément en petites équipes et finalement trouver une solution au problème en regroupant les solutions. En lisant cette définition de ce mode de développement, on constate tout de suite qu'il s'applique très bien à l'architecture micro-services. En effet, les sous-problèmes dans cette architecture représentent les composants de l'application à la base de la philosophie micro-services. On peut donc dire que le développement en silo n'est pas évité lors d'un projet suivant cette architecture et est au contraire fortement favorisé par la séparation en composant majoritairement indépendants encourageant le développement en parallèle. Par contre, il est intéressant de développer sur la façon dont le développement en silo est effectué dans cette nouvelle architecture logicielle. La méthode de développement en silo classique dans les entreprises est de séparer les équipes en niveau de technologie ou en département. On a donc des équipes telles que l'équipe d'assurance qualité, développement, etc. Par contre, les équipes de travail dans l'architecture micro-services sont différentes. On a besoin de membres de chacun des niveaux pour faire des composants de qualité et les équipes de travail sont donc maintenant composées de membres de chacun de ces niveaux. On a donc des membres avec différentes forces qui contribuent ensemble à répondre è tous les besoins du composant. La figure ci-dessous illustre cette différence dans les équipes de travail. On a à gauche les équipes de travail traditionnelles et à droite le nouveau modèle de développement en silo.

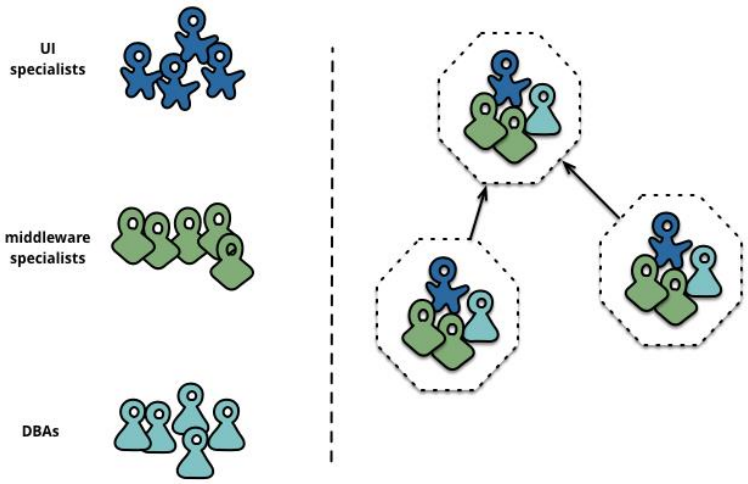

Figure 4: Développement en silo dans l'architecture micro-services[3]

6 Conclusion

En conclusion, on peut constater que chacune des architectures à des avantages et des inconvénients par rapport à l'autre. Une architecture en couche semble plus appropriée pour des applications de petite ou moyenne taille et une architecture micro-services pour les applications plus complexes. L'architecture en couche semble plus appropriée si le projet doit démarrer rapidement due à la simplicité de la conception comparativement à l'architecture micro-services. La plupart des développeurs semblent suggérer de commencer à développer de manière monolithique et ensuite passer à une architecture micro-service lorsque l'application devient trop grosse et difficile à maintenir. Le travail en silo et la réutilisabilité des composants de l'architecture micro-services semblent tout à fait appropriés au monde du développement logiciel qui doit faire face à une technologie évoluant à une vitesse phénoménale. On peut aussi répondre à la deuxième question que tentait de répondre ce travail et affirmer que le développement de fonctionnalités primaires à faible couplage externe et le travail en petites équipes et en parallèle au lieu de grandes équipes qui travaille sur une application monolithique n'évitent pas le développement en silo mais au contraire le

[3] La figure 4 provient de la référence 13

favorise. Par contre, les équipes sont séparées différemment de la manière usuelle et regroupent des membres ayant chacune des compétences différentes. Ce travail a été fort intéressant à réaliser et à piquer ma curiosité sur le développement d'application suivant une architecture micro-services sur le marché du travail. Si on croit ce que certains affirment sur cette architecture SOA qui affirme qu'elle deviendra une des architectures les plus utilisées, je travaillerai certainement de cette façon dans ma carrière d'ingénieur logiciel à au moins un moment. Personnellement, je trouve que le travail en grandes équipes et le travail en silo ont tous les deux des avantages. Si je me fît au projet intégrateur 1 du cours INF1995, le travail collaboratif et le travail en silo ont tout deux des avantages et des désavantages et la meilleure solution se trouve quelque part entre les deux à mon avis (prendre le temps de savoir comment les autres équipes vont implémenter leur partie et être mis au courant des modifications au fur et à mesure). J'éprouve donc de la difficulté à trancher sur laquelle me semble la meilleure et je crois que le contexte du projet est ce sur quoi il faut se baser pour faire un choix éclairé.

7 Références

1- Kharenko, Anton (2015), Monolithic vs. Microservices Architecture, Consulté le 1 avril 2017, tiré de https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59

2- Mulesoft, Microservices vs Monolithic Architecture, Consulté le 4 avril 2017, tiré de https://www.mulesoft.com/resources/api/microservices-vs-monolithic

3- Annett, Robert (2014), Where is the complexity?, Consulté le 4 avril 2017, tiré de http://www.codingthearchitecture.com/2014/05/01/where_is_the_complexity.html

4- CCM, SOA - Architecture Orienté Service, Consulté le 8 avril 2017, tiré de http://www.commentcamarche.net/contents/1241-soa-architecture-orientee-service

5- LINUX journal, Three-Tier Architecture, Consulté le 8 avril 2017, tiré de http://www.linuxjournal.com/article/3508

6- Developpez.com, MVC vs 3-tier, Consulté le 9 avril 2017, tiré de https://www.developpez.net/forums/d1002600/general-developpement/alm/mvc-vs-3-tier/

7- Guibault, François (2006), Modèle architectural, Consulté le 9 avril 2017, tiré de https://moodle.polymtl.ca/pluginfile.php/200820/mod_resource/content/2/05-ModeleArchitectural.pdf

8- Heithem, Abbes (2016), Achitecture n-tiers, Consulté le 9 avril 2017, tiré de https://fr.slideshare.net/HeithemAbbes1/architectures-ntiers

9- Sahnine, Kadda (2015), l'émergence des architectures orientées microservices, Consulté le 9 avril 2017, tiré de http://blog.inovia-conseil.fr/?p=155

10- behnam (2016), Microservice, Consulté le 10 avril 2017, tiré de http://www.imotif.net/index.php/2016/10/06/microservice/

11- Crochet, Antoine (2008), L'architecture en silo, Consulté le 10 avril 2017, tiré de http://www.journaldunet.com/solutions/dsi/enquete/l-histoire-de-l-urbanisation-informatique-en-images/l-architecture-en-silos.shtml

12- Microservices, Pattern: Microservice Architecture, Consulté le 11 avril 2017, tiré de http://microservices.io/patterns/microservices.html

13- Fowler, Martin (2014), Microservices a definiton of this new architectural term, Consulté le 17 avril 2017, tiré de https://martinfowler.com/articles/microservices.html