Projet Architecture Logicielle

***Document d’Architecture Technique***

# Contexte : Objectif du Document

## Contexte Général

Dans le cadre de la création d’une entreprise de commerce, cette dernière a émis le souhait de se doter d’une application Web. L’application devra permettre de développer le e-commerce de l’entreprise ainsi que la fidélisation et l’instauration d’une communauté de client. Ce document détaillera l’ensemble de l’architecture technique et logicielle de l’application, grâce entre autres à un langage de modélisation : UML (Unified Modeling Langage) permettant la confection de différents diagrammes représentant l’architecture.

## Détail de l’application

Comme sus-cité, l’application web aura comme fonctionnalité principale de mettre à disposition de l’entreprise et des clients une plateforme de e-commerce, mettant a disposition des produits, la possibilité de commander et d’être livré via un transporteur tier, un système de paiement sécurisé avec possibilité de facilité de paiement, l’édition de facture et de bon de commande… Parallèlement l’entreprise veut mettre a disposition de ses client un forum de discussion et d’entraide ainsi que du contenu personnalisé.

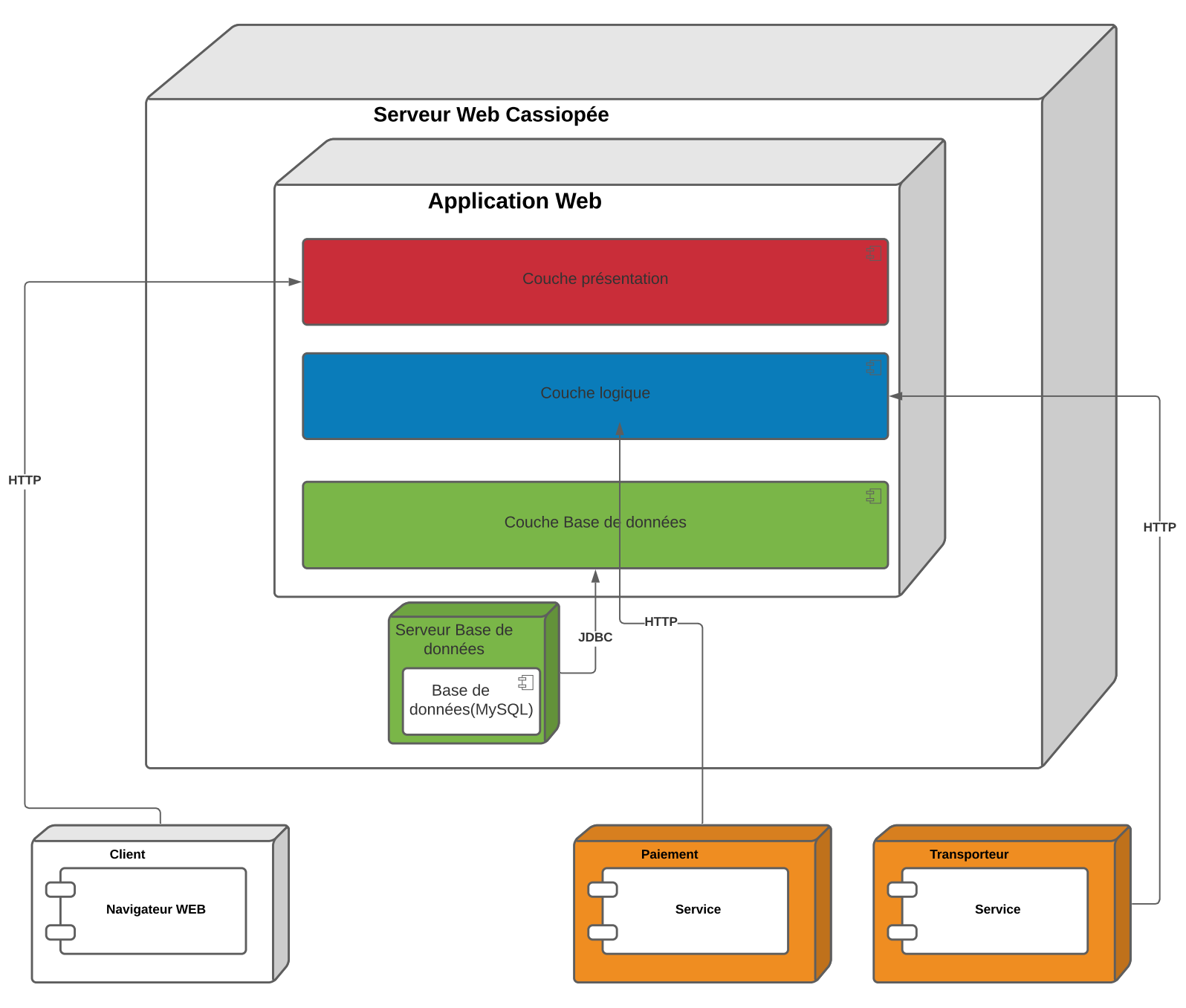
# Architecture matérielle

Pour l’utilisation ainsi que le déploiement de l’application celle-ci aura besoin :

* D’un Serveur Web : Ce dernier hébergera l’application web et la mettra à disposition des clients. A la différence d’un hébergeur ici le serveur sera détenu par l’entreprise ou son prestataire lui offrant plus de contrôle.

Ou

* Un Hébergeur tier mettant à disposition tout les outils nécessaire (OVK, Infomaniak..)
* Base de données : SQL (MySQL, Oracle, PostGre) ou NoSQL (MongoDB) : La base de données sera chargée de contenir les données basées sur les modèles de l’application (Users, Products, Category, Order…)
* Serveur de base de données : Hébergeant la base de données (Il est possible que le serveur web fournisse directement ce service). A la manière d’un serveur Web, il met à disposition la base de données pour l’application Web.
* D’un Nom de Domaine : Nécessaire pour lié l’application à une adresse compréhensible pour les utilisateurs et relié à l’entreprise.
* Client : Navigateur Web
* Environnement de développement (IDE, Librairies, Packages…)
* Librairie de test (JUnit / HttpUnit)



# Architecture Logicielle

Pour ce type d’application deux architectures s’offrent à nous possédant chacune leur spécificité :

## L’architecture en couche

Il s’agit d’une des architectures les plus communes et les plus « intuitives » à déployer.

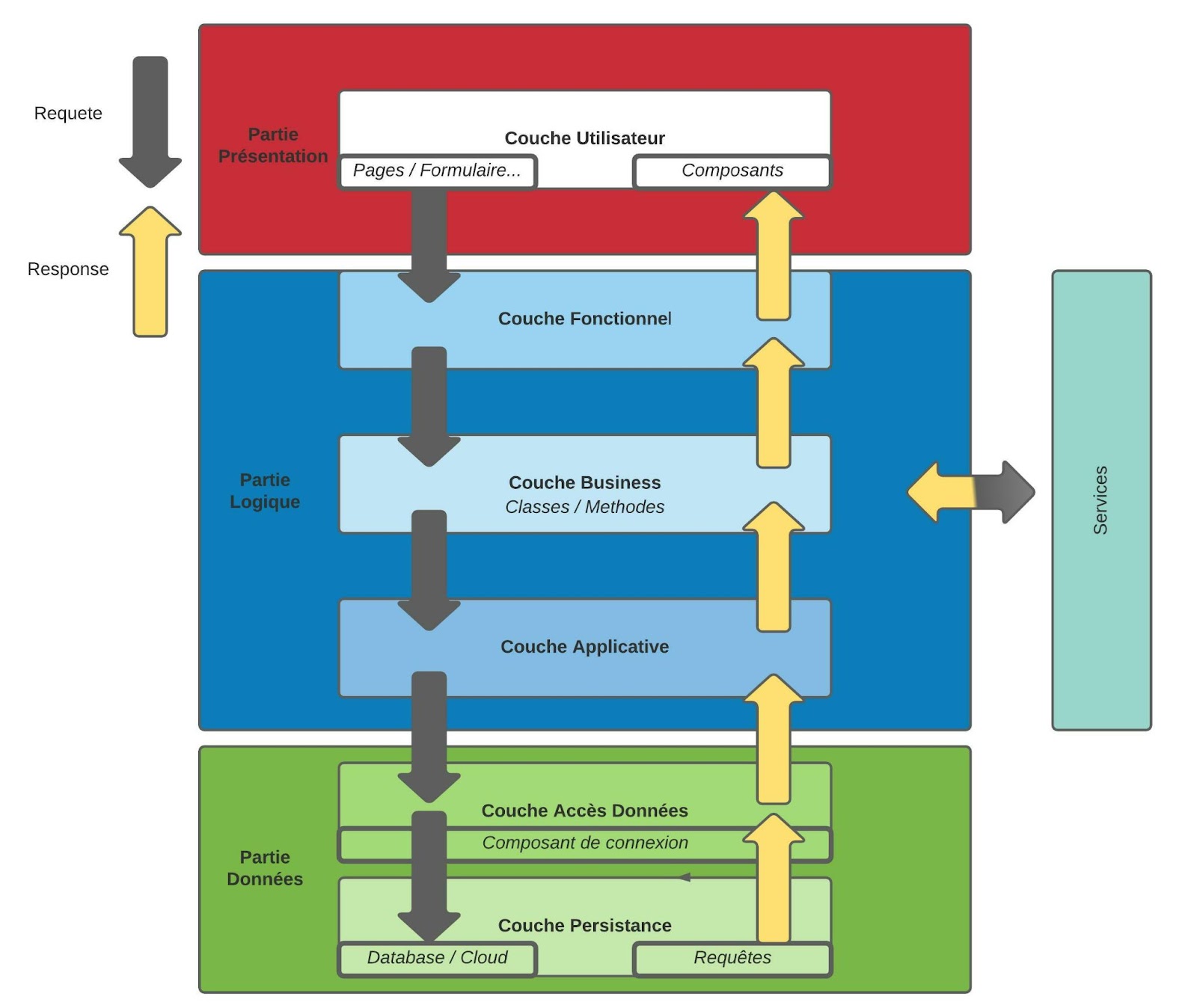
L’application est découpée en plusieurs couches superposées ayant chacune leur rôle à jouer et communicant en cascade (de haut en bas). Cette dernière offre plusieurs avantages :

* Facilité de développement :
  + Peu Complexe
  + Développement en équipe
  + Développement des couches indépendamment des autres
* Testabilité :
  + Test des couches une par une
* Déploiement Rapide

Mais aussi des limites :

* Maintenabilité : Le changement dans une couche nécessite l’arrêt de l’ensemble de l’application
* Lenteur de l’application : par son architecture, cette dernière ralentit les performances sur des gros flux de données

Design Pattern : Le MVC peut être implémenté ici.



## L’architecture en appel retour

L’architecture en appel-retour est aussi une architecture bien connue, qu’il s’agisse d’API ou de micro-services. Un composant appel un autre composant qui lui retourne ce qu’il demande. Dans notre contexte il s’agit d’une application Web divisée en micro-services (fonctionnalités) qui communiquent entre eux via des appels/retours selon leur besoin. Par exemple on peut avoir un micro-services Utilisateur qui communique avec le micro-services s’occupant des commandes. En résumé il s’agit d’une agrégation de composant indépendant des autres.

Avantages :

* Fonctionnalités séparées
* Indépendance des micro-services : Si un micro-services s’arrête les autres continues de fonctionner en dehors du « scope » de celui arrêté
* Code mieux compartimenté
* Meilleure répartition de la charge de travail donc meilleure stabilité

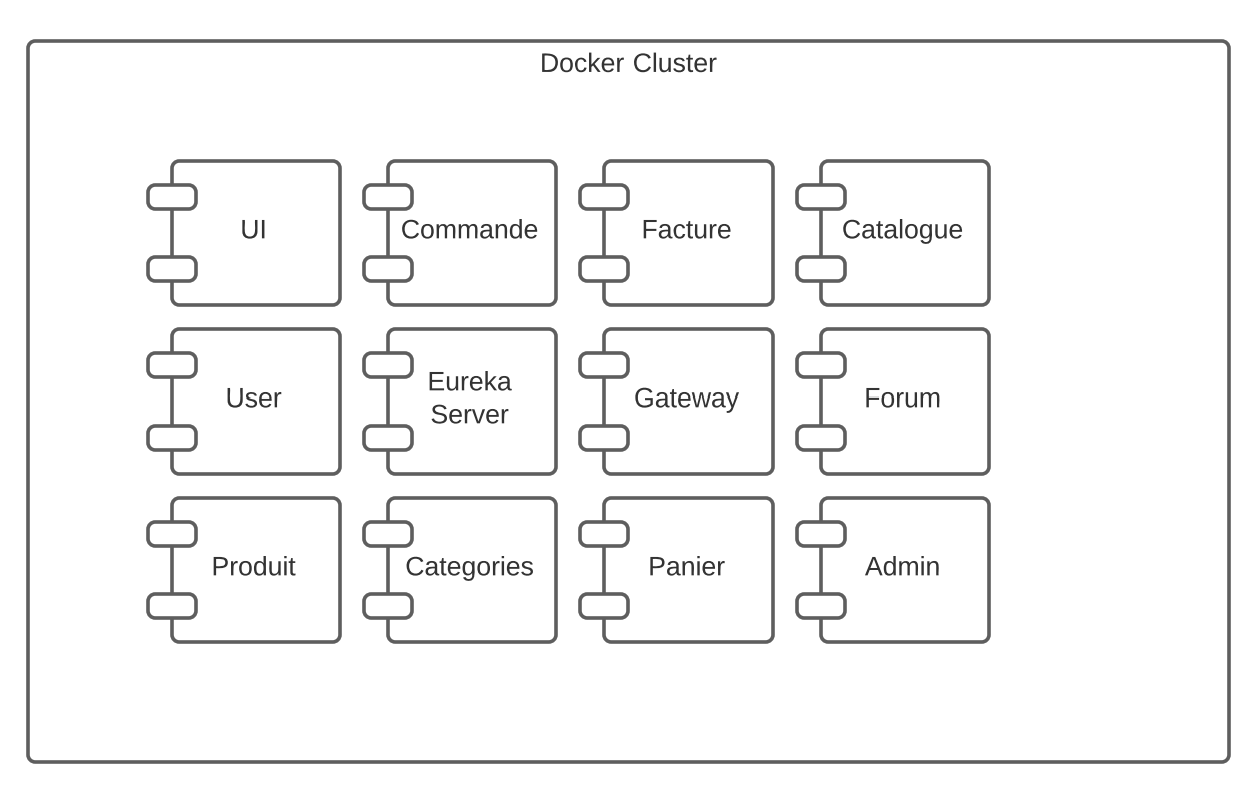
Désavantages :

* Développement plus complexe et plus long : par la fragmentation des différents micro-services
* Equilibrage de charge : Une caractéristique à double tranchant, un micro-services avec un mauvais équilibrage peut ralentir l’application
* Compliqué avec un grand nombre de micro-services
* Utilisation de mémoire élevée : Augmentation des caractéristiques de l’architecture technique

## Conclusion

Chacune de ces architectures possèdent donc des caractéristiques correspondant à notre projet, cependant il convient de choisir la mieux adaptée pour des performances, des coûts et une efficacité de développement optimisés.

De notre point de vue il convient ici d’adopter une architecture basée sur les micro-services afin de d’assurer le « zéro down time » et la haute disponibilité de l’application.



Voici les différents principaux micro-services de l’application qui seront ensuite redécoupés en sous services.

Au vu des nombreuses fonctionnalités nécessaire, cette architecture permettra de maintenir le site même si une fonctionnalité annexe cesse de fonctionner, par exemple si le forum est « down » l’utilisateur pourra toujours commander.

Le développement de l’application prendra plus de temps au début mais sera par la suite bien plus stable et maintenable que la version en couche classique.