Équipe 110

Document d'architecture logicielle

Version 3.0

Historique des révisions

Date	Version	Description	Auteur
2020-02-04	0.1	Ajout des informations de l'équipe sur la page de couverture	François-Xavier Legault
2020-02-04	0.2	Premier jet de la section 1	François-Xavier Legault
2020-02-05	0.3	Premier jet de la section 2	François-Xavier Legault
2020-02-06	0.4	Ajout des diagrammes de séquences	Arthur Garnier
2020-02-06	0.5	Finalisation des partie 1 et 2 et rédaction de la partie 3	François-Xavier Legault
2020-02-07	0.6	Ajout du schéma de déploiement et rédaction de taille et performance	Jacob Dorais
2020-02-07	1.0	Correction et formatage	François-Xavier Legault
2020-04-09	2.0	Correction diagramme de séquence (final)	Arthur Garnier
2020-04-13	3.0	Correction français et diagramme de déploiement	François-Xavier Legault Jacob Dorais
2020-04-13	3.1	Correction diagramme paquetage serveur	Arthur Garnier

Table des matières

1. Introduction **4**
1.1 Objet 4
1.2 Portée 4
1.3 Définitions, acronymes et abréviations 4

2. Objectifs et contraintes architecturales **5**
2.1 Objectifs 5
2.2 Contraintes architecturales 5

3. Vue des cas d'utilisation **5**

4. Vue logique **10**

5. Vue des processus **13**

6. Vue de déploiement **17**

7. Taille et performance **17**
7.1 Client léger 17
7.2 Client lourd 17
7.3 Serveur 18

Document d'architecture logicielle

1. Introduction

1.1 Objet

L'objet de ce document d'architecture logicielle est une justification des choix d'architectures effectués en rapport avec la satisfaction des exigences non fonctionnelles. De plus, l'objet de ce document est aussi de décrire les différents éléments architecturaux de haut niveau de notre logiciel Pïntùré. Pour ce faire, nous nous servirons de diagrammes architecturaux que représentent les vues logicielles.

Ce document d'architecture logicielle présentera dans un premier temps les objectifs et les contraintes architecturales, suivi de la vue des cas d'utilisation, la vue logique, la vue des processus et la vue de déploiement, tous présentés à l'aide de diagrammes. Finalement, une description des caractéristiques de taille et de performance sera présentée.

1.2 Portée

Le document d'architecture logicielle assiste les développeurs afin d'aider à leur compréhension de l'architecture de Pïntùré, lors de la phase d'élaboration du processus logiciel.

1.3 Définitions, acronymes et abréviations

TCP : Protocole de contrôle de transmission

WPF : (Windows Presentation Foundation) un cadriciel d'interface graphique pour concevoir des applications.

Client léger : Application spécifiquement conçue pour être exécutée à partir d'appareils mobiles

(tablette).

Client lourd : Application spécifiquement conçue pour être exécutée sur un PC (windows).

Partie: Durée du jeu à l'issue de laquelle sera désigné le gagnant.

Tour: Durée selon laquelle une seule personne est dessinateur.

Jeu: activité de produire des dessins à tour de rôle et de deviner ce que représente le dessin.

Dessin: ensemble de traits SVG qui forment une image.

Joueur virtuel: simulation d'un joueur fait par un ordinateur.

Joueur humain: un utilisateur utilisant soit client lourd ou client léger.

2. Objectifs et contraintes architecturales

L'objectif de ce projet est de développer une application multijoueur pour le jeu P'ïntùré, et cela, à partir d'une architecture déjà existante. Nous avons donc pu remarquer l'importance qu'une architecture soit documentée et adaptable.

Le fait que nous devions développer le logiciel à partir d'un autre déjà existant constitue la première contrainte architecturale face à laquelle nous nous sommes retrouvés. En effet, l'architecture déjà en place devait être respectée par toutes les modifications que nous lui avons ajoutées, question de garder une continuité en minimisant les impacts architecturaux de ces modifications.

2.1 Client léger

Une contrainte possédant un impact architectural était qu'il nous était imposé de développer le client léger soit en Java ou Kotlin. L'application doit être fonctionnelle sur les tablettes Galaxy Tab A utilisant Android. Notre application doit être résistance lorsque mise vis-à-vis plusieurs scénarios d'utilisations, tout dépendant des exigences prédéterminées.

D'un côté, le client lourd doit permettre aux utilisateurs de dessiner en utilisant différentes fonctionnalités du logiciel, tout en permettant une communication fonctionnant à l'aide de canaux de discussion.

D'un autre côté, nous devons développer une architecture afin de permettre l'échange d'informations entre le serveur et les deux clients (lourd et léger). Quelques points importants à ce niveau sont; la communication asynchrone, la sécurité de la communication et le fait que différentes informations doivent être stockées et accessibles à l'aide de requêtes. Notre application doit donc être adaptée pour la synchronisation afin de permettre la collaboration des utilisateurs en temps réel.

Le patron de conception "modèle vue contrôleur" est utilisé tout au long du développement pour le client léger, parce qu'il permet d'éviter les dépendances des classes et d'isoler les différentes composantes de l'application.

Par ailleurs, le client léger ne doit pas "crash" à des moments imprévus invoqués par l'utilisateur. Celui fonctionnera comme normalement.

2.2 Client lourd

Une contrainte possédant un impact architectural était qu'il nous était imposé de développer le client lourd soit en WPF (.NET C#, voir définition de WPF dans la section 1.3). L'application doit être fonctionnelle sur les ordinateurs personnels utilisant le système d'exploitation Windows. Notre application doit être résistance lorsque mise vis-à-vis plusieurs scénarios d'utilisations, tout dépendant des exigences prédéterminées.

Premièrement, nous devons développer une architecture afin de permettre l'échange d'informations entre le serveur et les deux clients (lourd et léger). La communication entre le serveur et le client lourd doit être stable, ainsi est de la communication avec la base de données hébergée en utilisant Heroku. Quelques points importants à ce niveau sont; la communication asynchrone, la sécurité de la communication et le fait que différentes informations doivent être stockées et accessibles à l'aide de requêtes. Notre application doit donc être adaptée pour la synchronisation afin de permettre la

collaboration des utilisateurs en temps réel.

Deuxièmement, le client lourd doit permettre aux utilisateurs de dessiner en utilisant différentes fonctionnalités du logiciel, tout en permettant une communication fonctionnant à l'aide de canaux de discussion.

Par ailleurs, le client lourd ne doit pas "crash" à des moments imprévus invoqués par l'utilisateur. Celui fonctionnera comme normalement.

2.3 Serveur

Le serveur doit premièrement s'assurer de la qualité de la communication entre le client lourd et client léger. Dans un deuxième temps, il doit s'assurer du stockage des informations, fichiers et images pour que l'application puisse les partager, selon les différentes requêtes des clients. Même s'il y a une panne, la base de données ne devrait pas perdre d'information.

En ce qui concerne la confidentialité des données et la sécurité, le serveur doit crypter les informations sur la base de données afin de les entreposer en sécurité. Toutes les requêtes envoyées au serveur doivent aussi subir une série de vérifications par celui-ci, afin de filtrer toutes les actions et requêtes malveillantes.

Pour la fiabilité, la base de données doit être active et prête pour recevoir des requêtes pour un minimum de 99% du temps. Une assurance de 99.5% est assurée par Heroku que nous utilisons pour l'hébergement de notre base de données.Le système et notre serveur devront être hors d'usage au maximum 1% du temps en minimisant le temps moyen de réparation après une panne.

Par ailleurs, le serveur sera construit avec NodeJS et Express en Typescript. Nous utiliserons Inversify pour les injections de dépendances afin de faciliter le développement ainsi que les tests. Le serveur doit pouvoir rouler sur tous types de systèmes d'exploitations.

3. Vue des cas d'utilisation

Dans cette section, nous présentons les aspects pertinents du modèle de cas d'utilisation à l'aide de diagrammes de cas d'utilisation.

On y retrouve trois acteurs interagissants avec le logiciel Pïntùré:

- l'utilisateur: une personne qui fait usage de l'application Pïntùré.
- serveur: dispositif offrant des services (gestion d'authentification, stockage de données, etc.) à un ou plusieurs clients.
- l'administrateur du réseau: une personne qui s'occupe de la gestion du réseau de Pïntùré

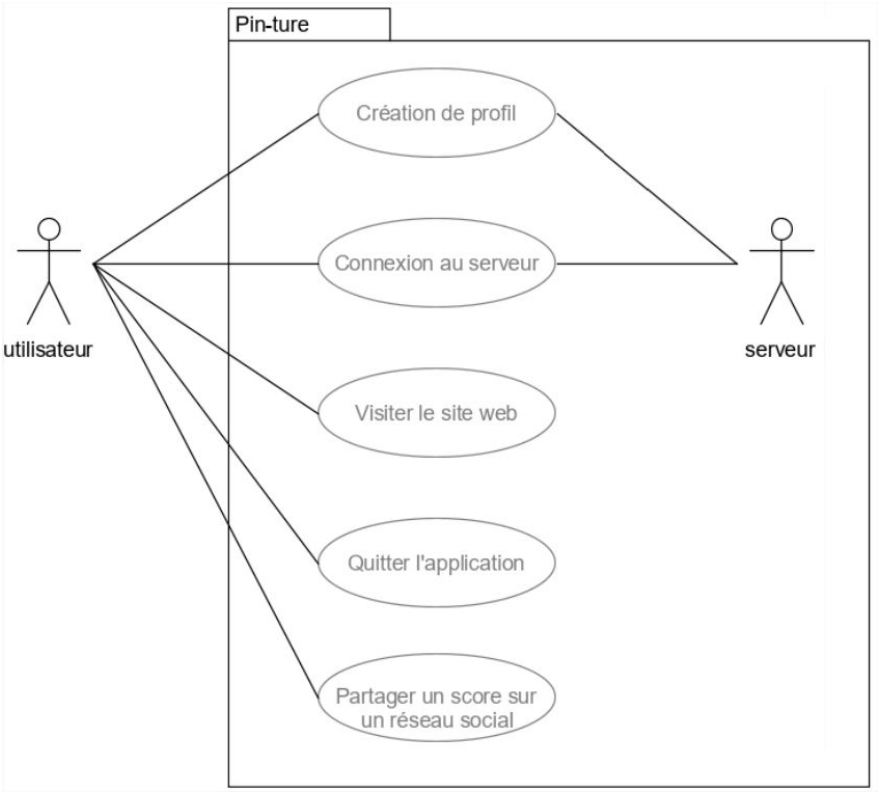

Figure 1: diagramme des cas d'utilisation du client lourd et léger niveau de base.

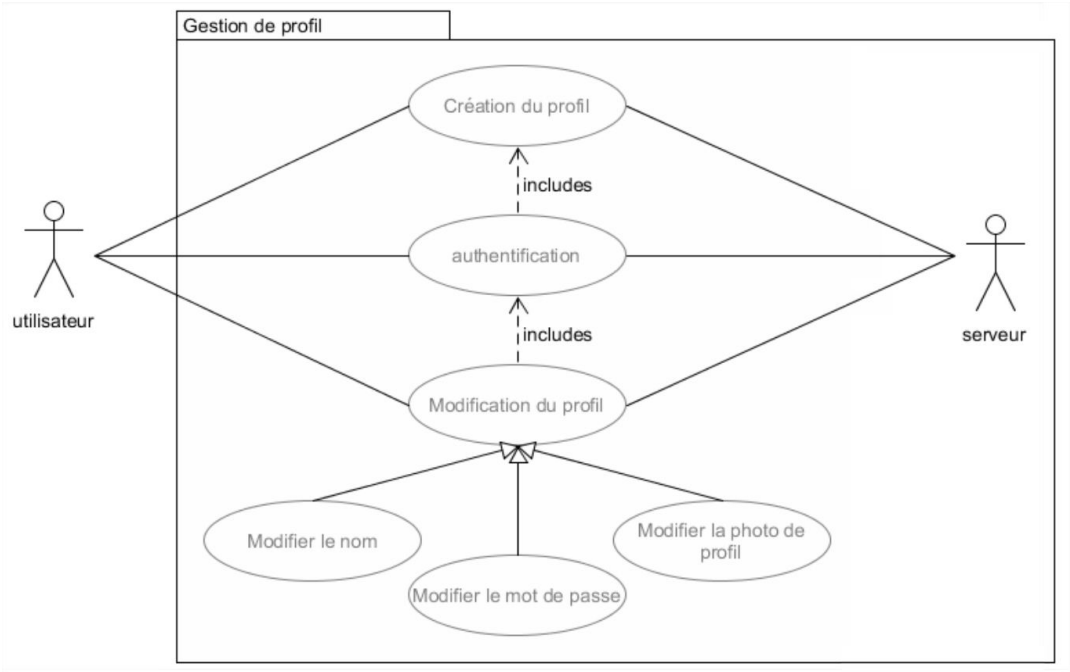

Figure 2: diagramme des cas d'utilisation du client lourd et léger pour la gestion de profil.

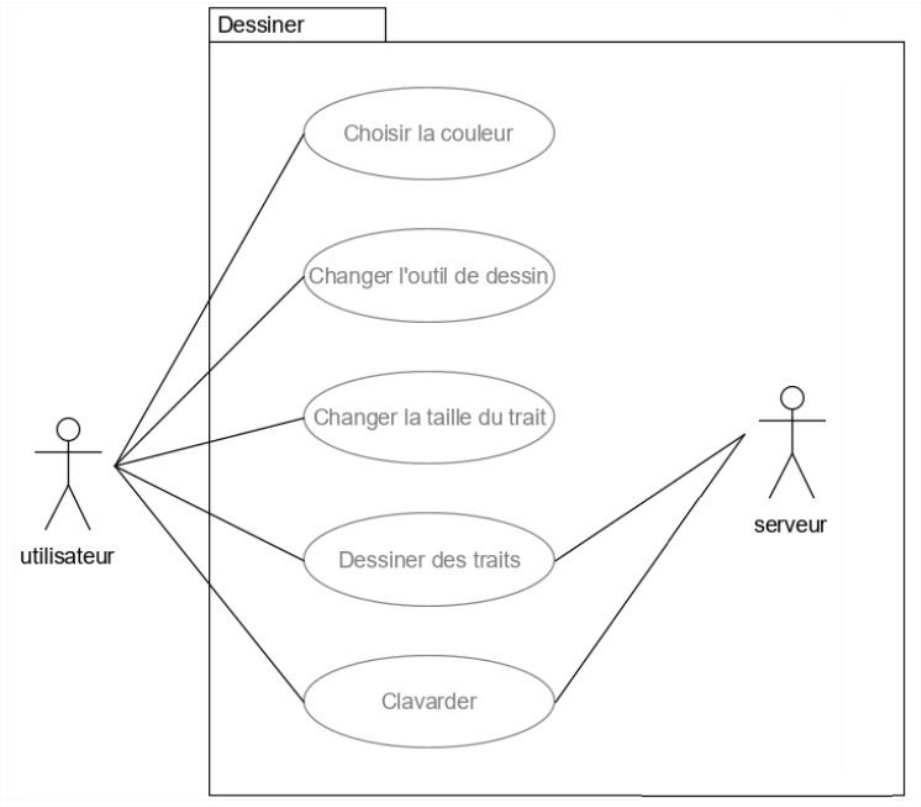

Figure 3: diagramme des cas d'utilisation du client lourd et léger pour dessiner.

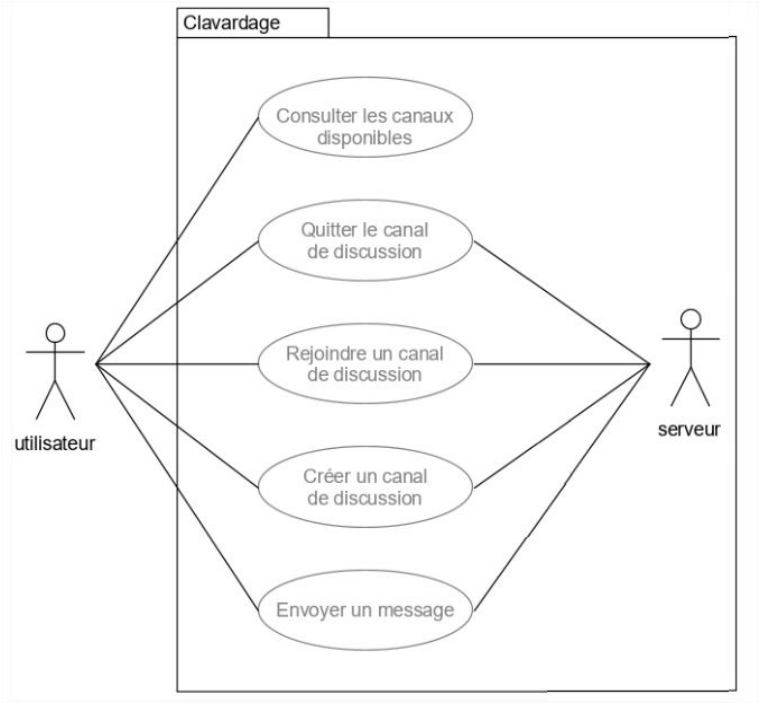

Figure 4: diagramme des cas d'utilisation du client lourd et léger niveau de base.

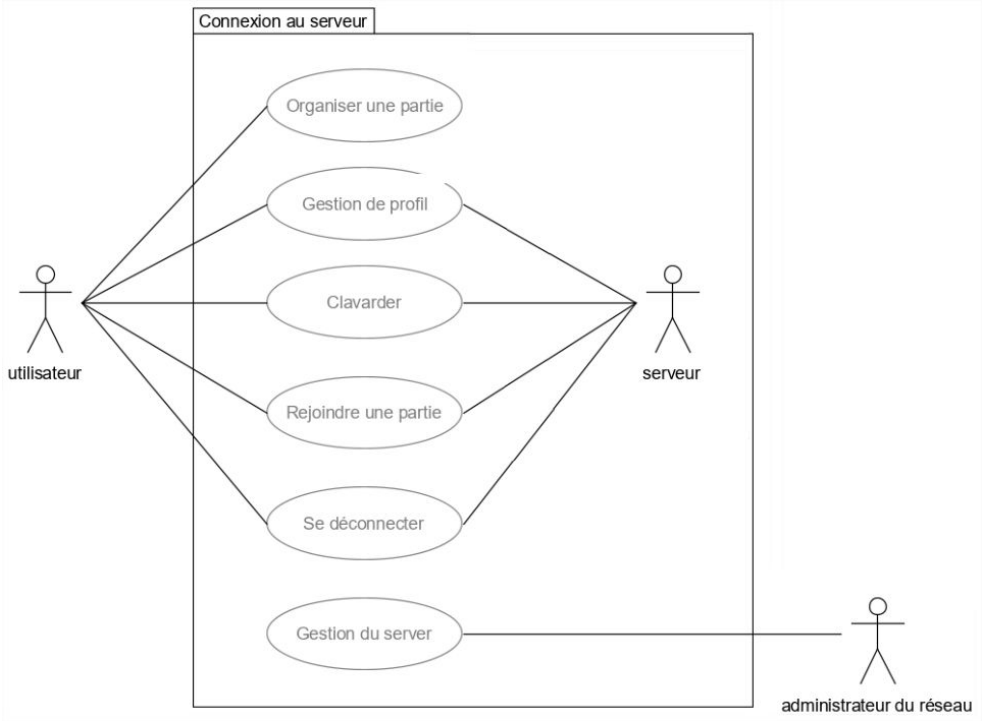

Figure 5: diagramme des cas d'utilisation du client lourd et léger pour la connexion au serveur.

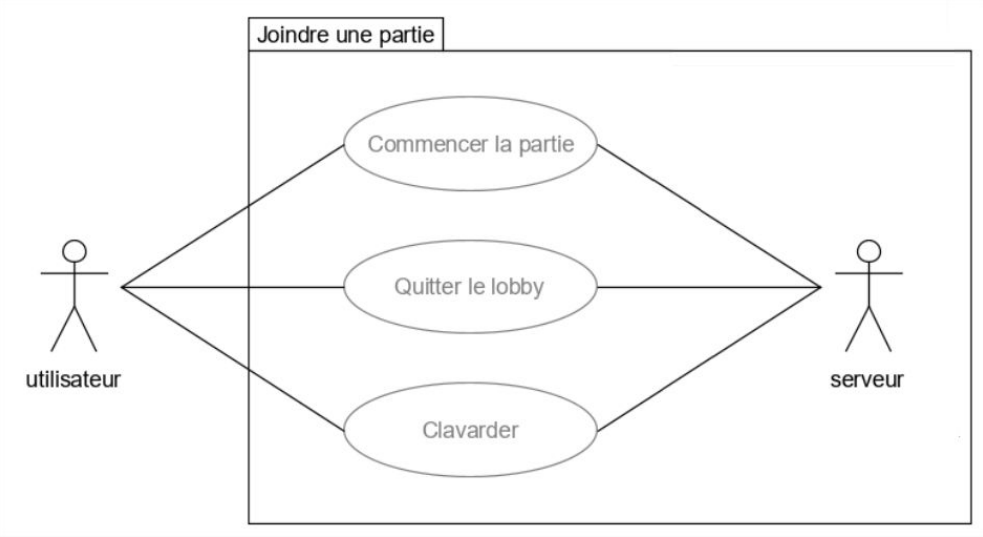

Figure 6: diagramme des cas d'utilisation du client lourd et léger pour rejoindre une partie.

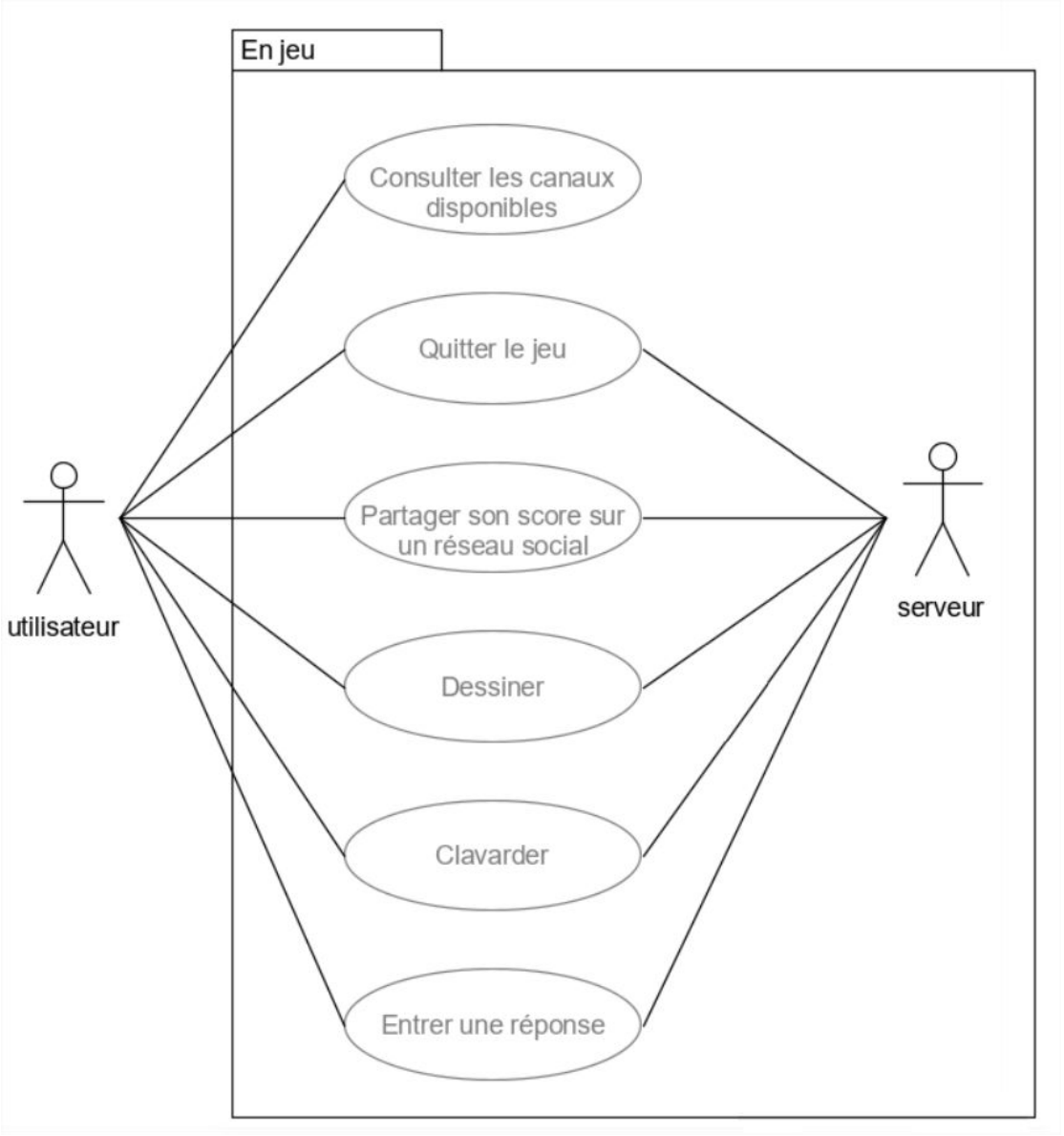

Figure 7: diagramme des cas d'utilisation du client lourd et léger en jeu.

4. Vue logique

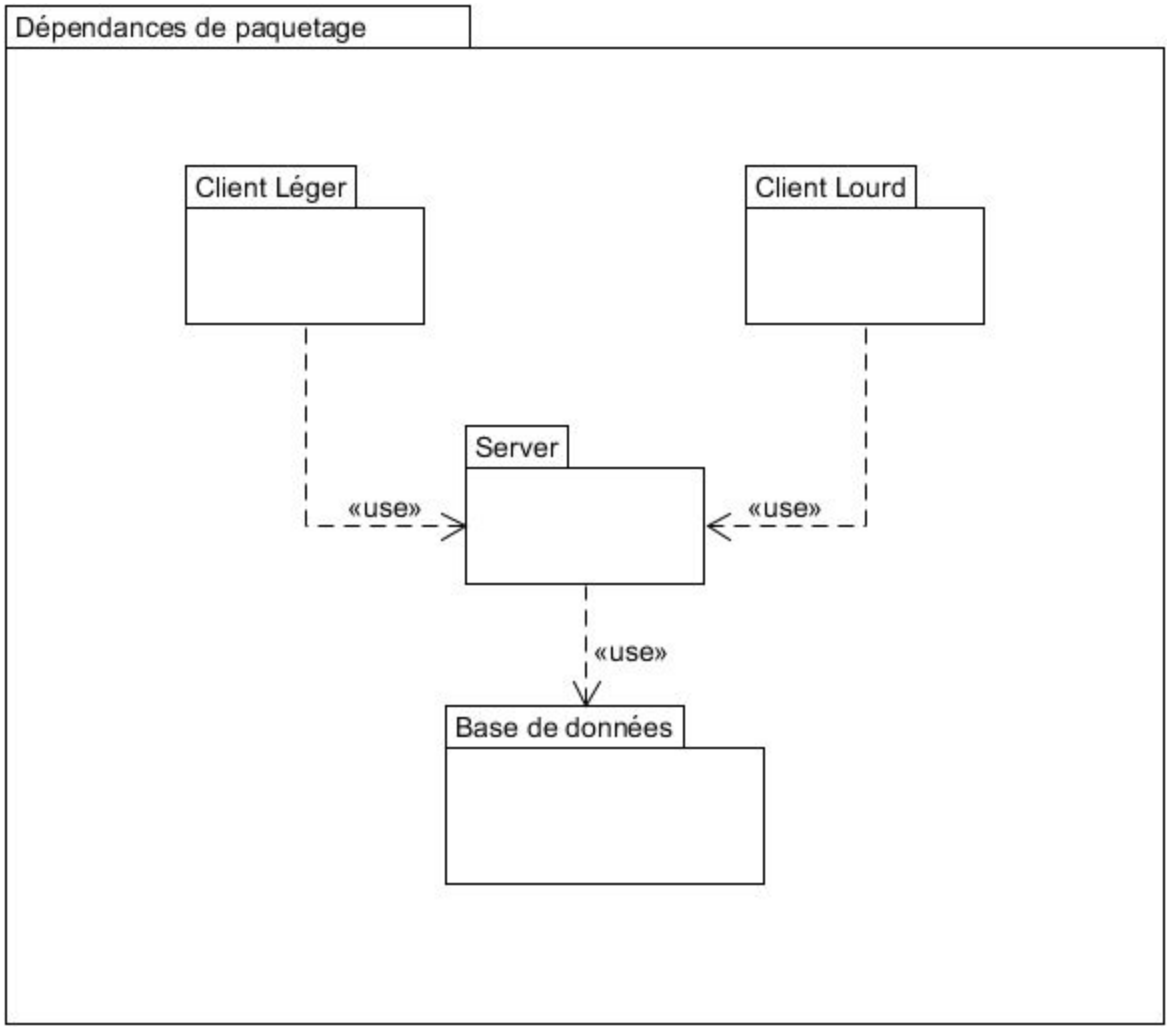

Figure 8: Diagramme de paquetage globale de Pïntùré

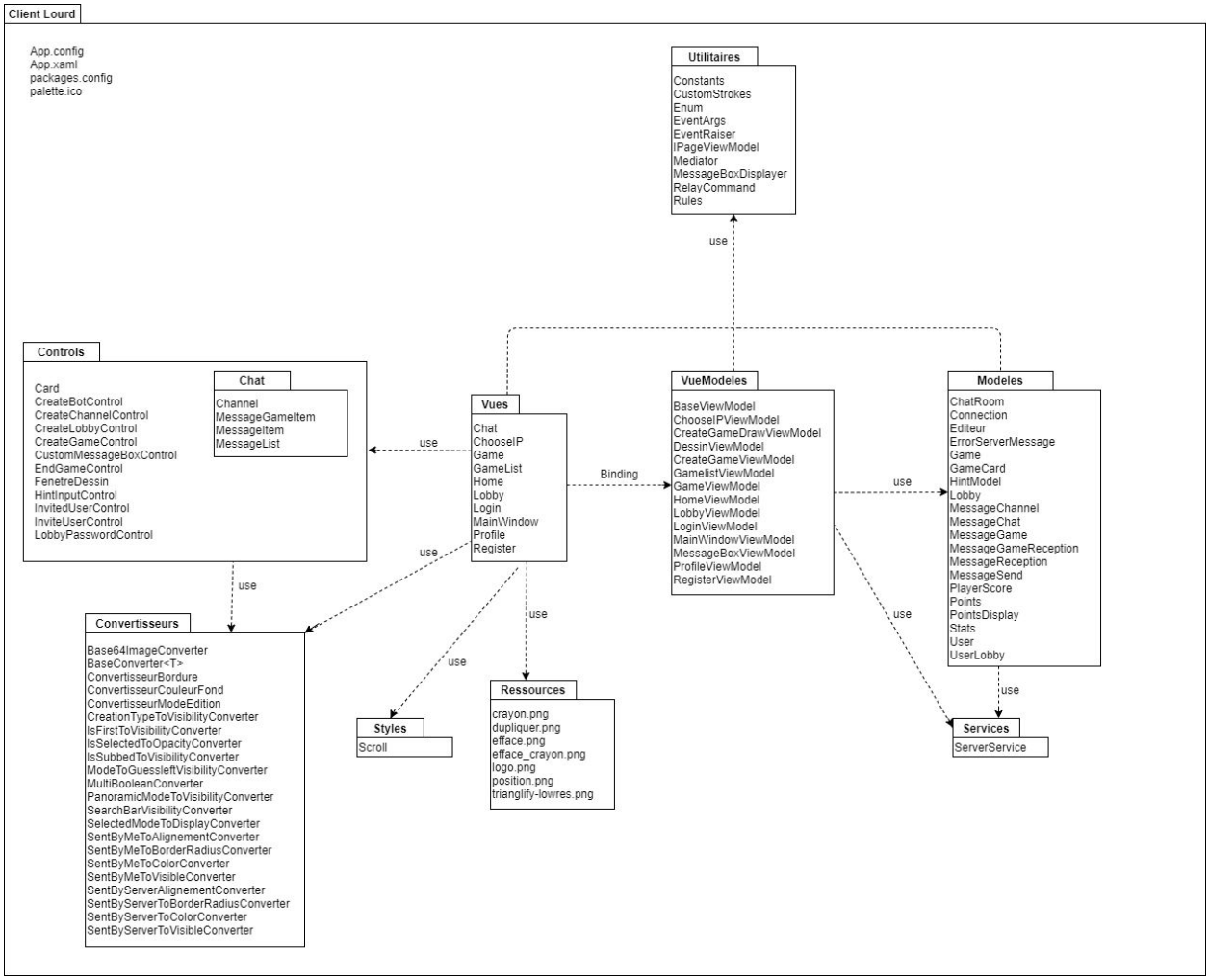

Figure 9: Diagramme de paquetage pour le client lourd

<Vue>
Ce paquetage contient toutes les fenêtres et les pages de l'application. Les fichiers XAML ainsi que leurs fichiers C# correspondants sont inclus dans ce paquetage.

<VueModeles>
Ce paquetage contient les "VueModeles" du modèle d'architecture MVVM associés aux vues.

<Modeles>
Ce paquetage contient les différentes classes contenant la logique de l'application et est complètement séparé des vues.

<Services>
Ce paquetage contient différents services respectant le modèle

d’architecture singleton.

<Ressources>
Ce paquetage contient toutes les images, sons, icones, etc. utilisés dans l’application.

<Convertisseurs>
Ce paquetage contient tous les convertisseurs agissant comme un intermédiaire entre la valeur voulu et la valeur disponible.

<Controles>
Ce paquetage contient les différents vues pouvant être utilisées à plusieurs places dans l’application. Les fichiers XAML ainsi que leurs fichiers C# correspondants sont inclus dans ce paquetage.

<Utilitaitres>
Ce paquetage contient des classes fournissant des fonctions ainsi que des variables communes à toute l’application.

<Styles>
Ce paquetage contient les différents styles modifiant l'apparence de différents composants XAML.

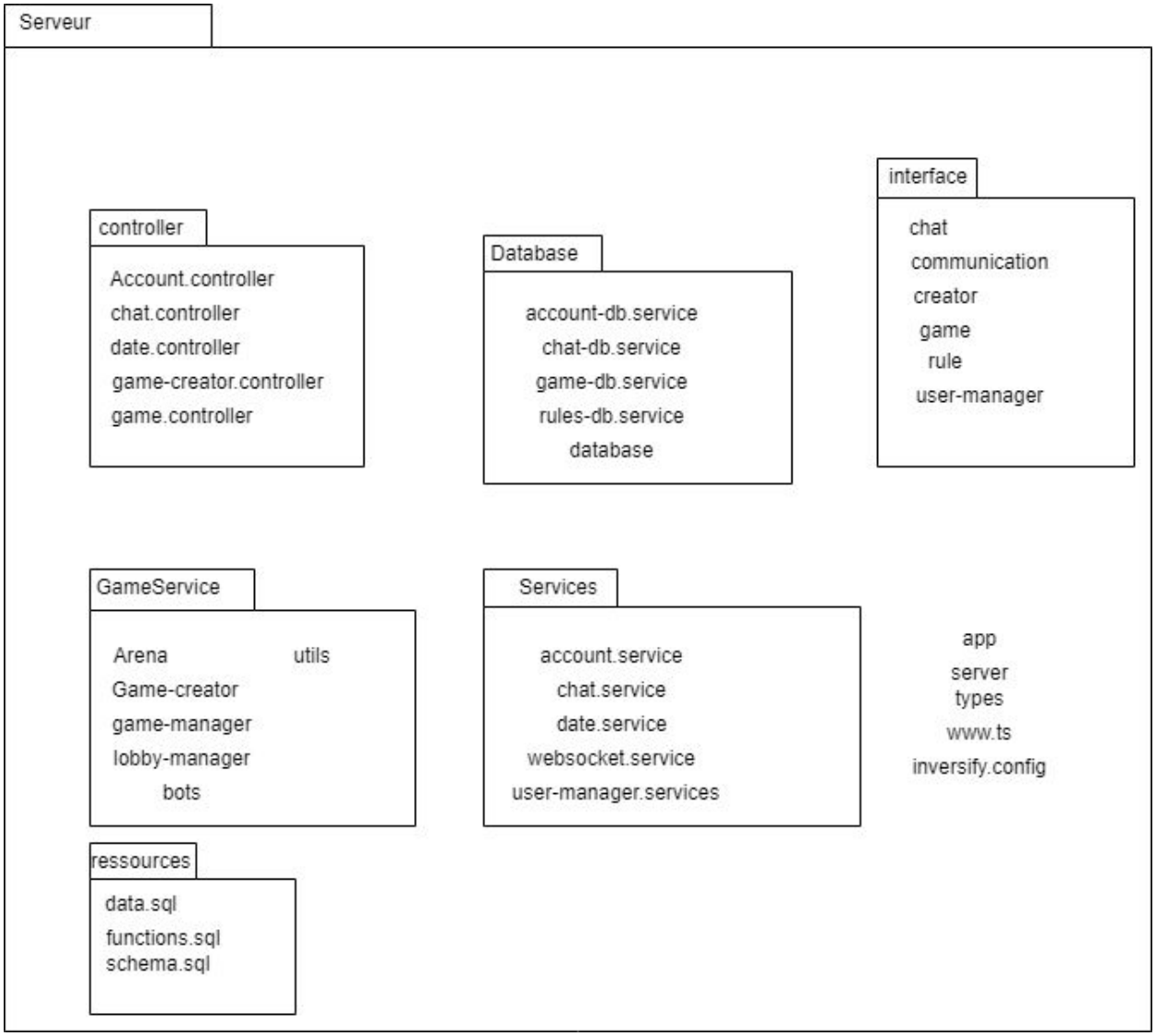

Figure 10: Diagramme de paquetage pour le serveur

<Services>
Ce paquetage contient l'ensemble des services permettant la gestion des profils des utilisateurs ainsi que leur connexion au socket. Permet aussi la gestion du websocket et du chat.

<gameService>
Ce paquetage contient l'ensemble des services permettant la gestion des parties, leur création selon le mode de jeux, le déroulement d'une partie ainsi que le fonctionnement des joueurs virtuels

<interfaces>
Ce paquetage contient l'ensemble des structures de données permettant aux services de communiquer

<controller>
Ce paquetage contient l'ensemble des méthodes pouvant être directement appelées par le client à l'aide d'une requête.

<Database>
Ce paquetage contient l'ensemble des services permettant de communiquer directement avec la base de données

<ressources>
Ce paquetage contient les différentes ressources pouvant être utiles à tous les services du serveur

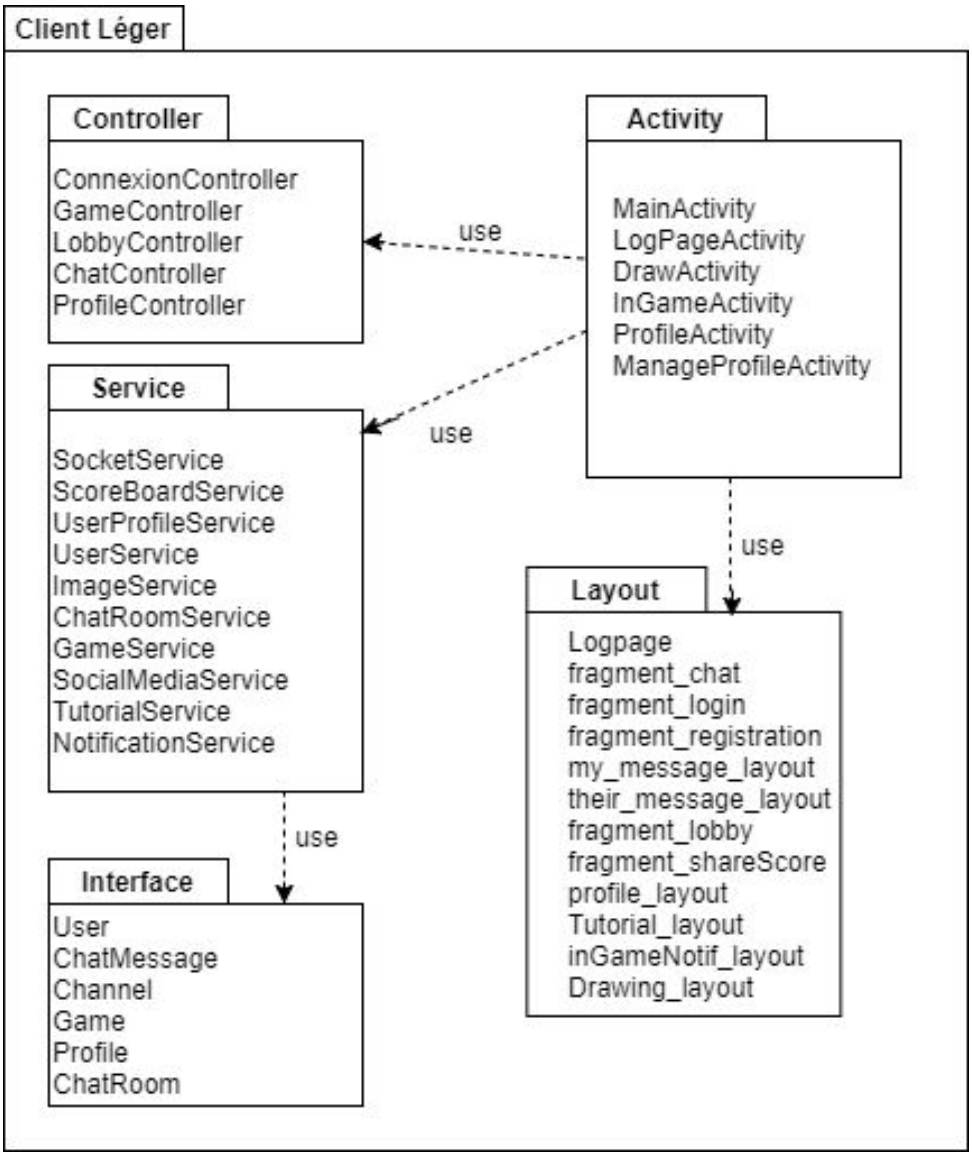

Figure 11: Diagramme de paquetage du client léger

<Controller>
Cet paquetage contient les “endpoints” du client léger.

<Service>
Cet paquetage contient les classes avec les logiques du client léger.

<Interface>
Cet paquetage contient les types d’interfaces pour les communications entre client et serveur.

<Activity>
Cet paquetage contient les activités pour l’ensemble d’interface du client léger.

<Layout>
Cet paquetage contient les affichages réutilisables par des activités.

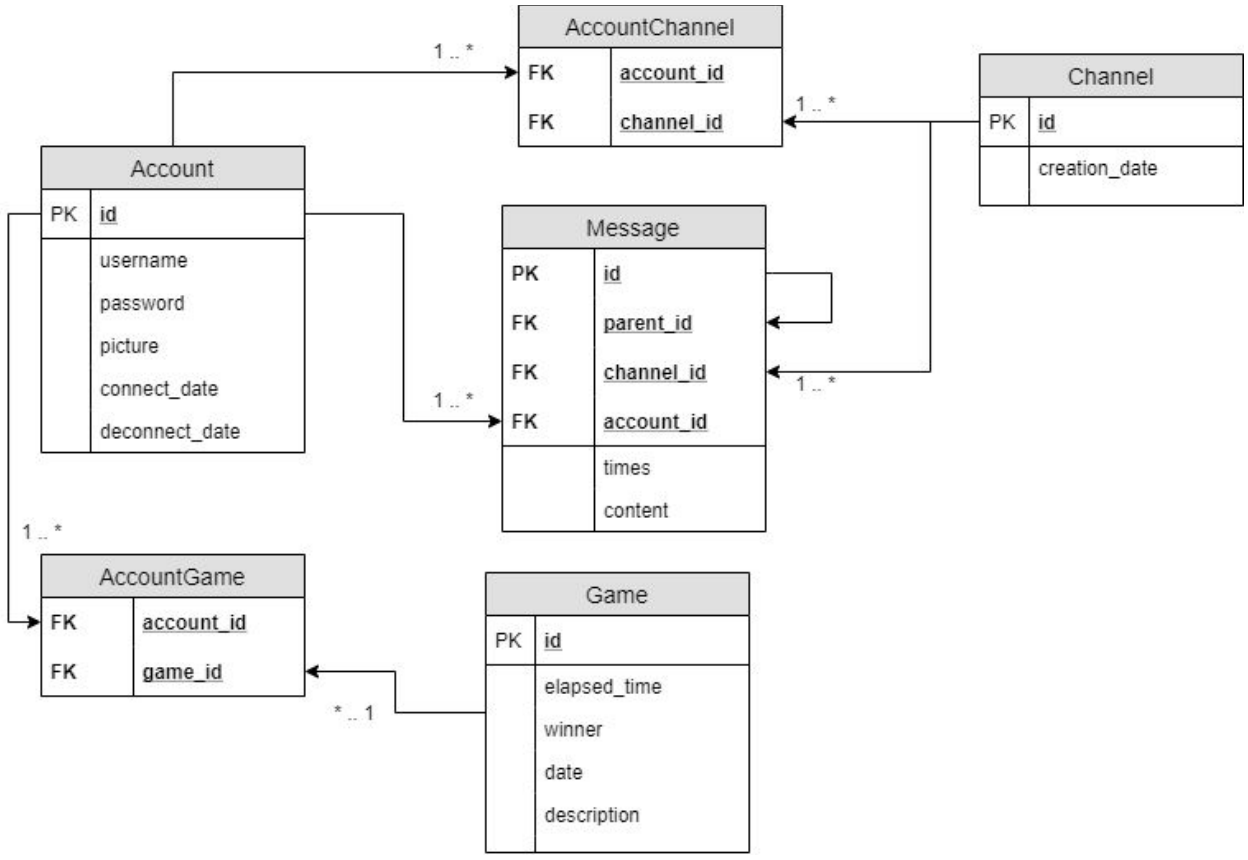

Figure 13: Diagramme des relations et entités de la base de données

Account
Cette table contient l'ensemble des comptes pouvant se connecter sur les clients. Ceux-ci ont aussi des descriptions supplémentaires comme: la photo de profil, la date de connexion et déconnexion.

Channel
Cette table contient l'identification d'une chaîne afin que l'utilisateur peut envoyer un message à un endroit spécifique.

AccountChannel
Cette table contient les clés primaires de "account" et "channel".

Message
Cette table contient l'ensemble des messages envoyés dans tous les chaînes différents. La clé primaire pointe à une clé parent, donc pour

la recherche, il faut chercher récursivement selon l'identification de la chaîne.

Game
Cette table contient toutes les parties qui ont eu lieu depuis le début.

AccountGame
Cette table contient les clés primaires de "account" et "game".

5. Vue des processus

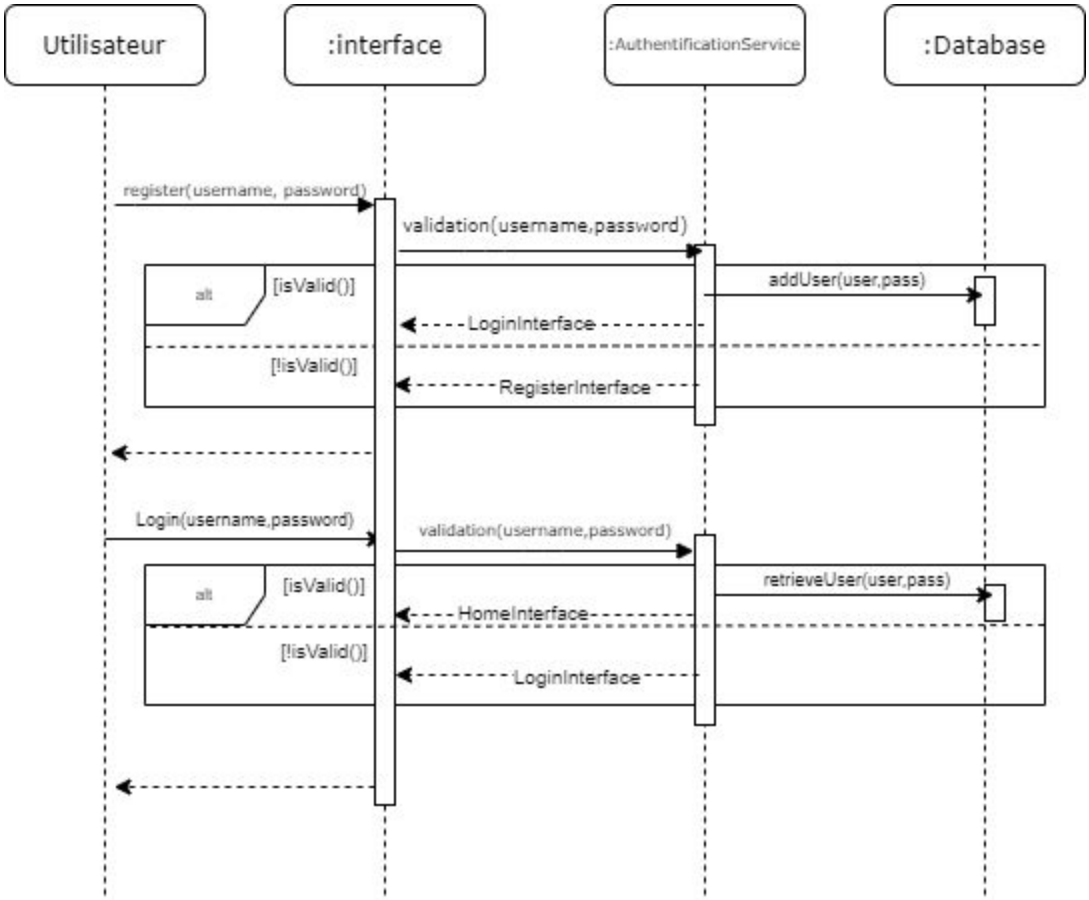

Figure 14: Diagramme de séquence système pour l'authentification

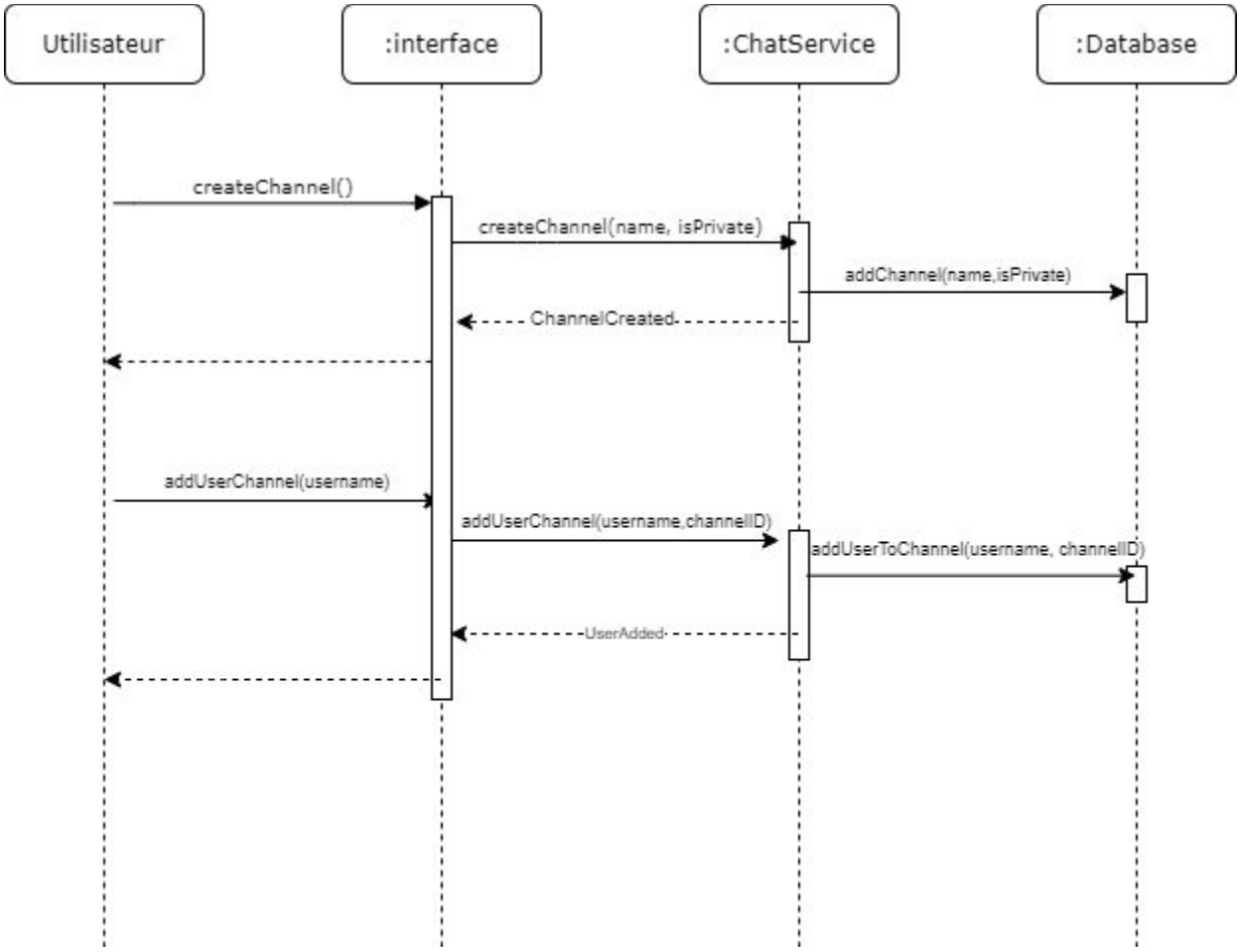

Figure 15: Diagramme de séquence système pour la gestion de canal de clavardage

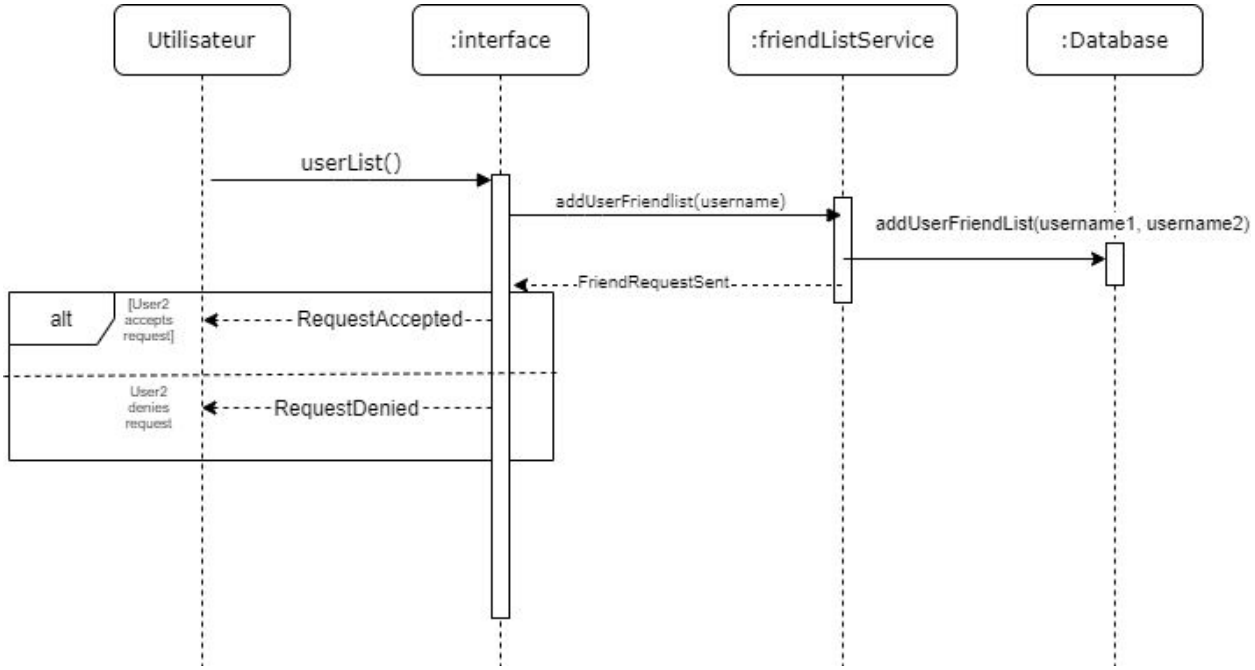

Figure 16: Diagramme de séquence système pour la gestion de la liste d'amis

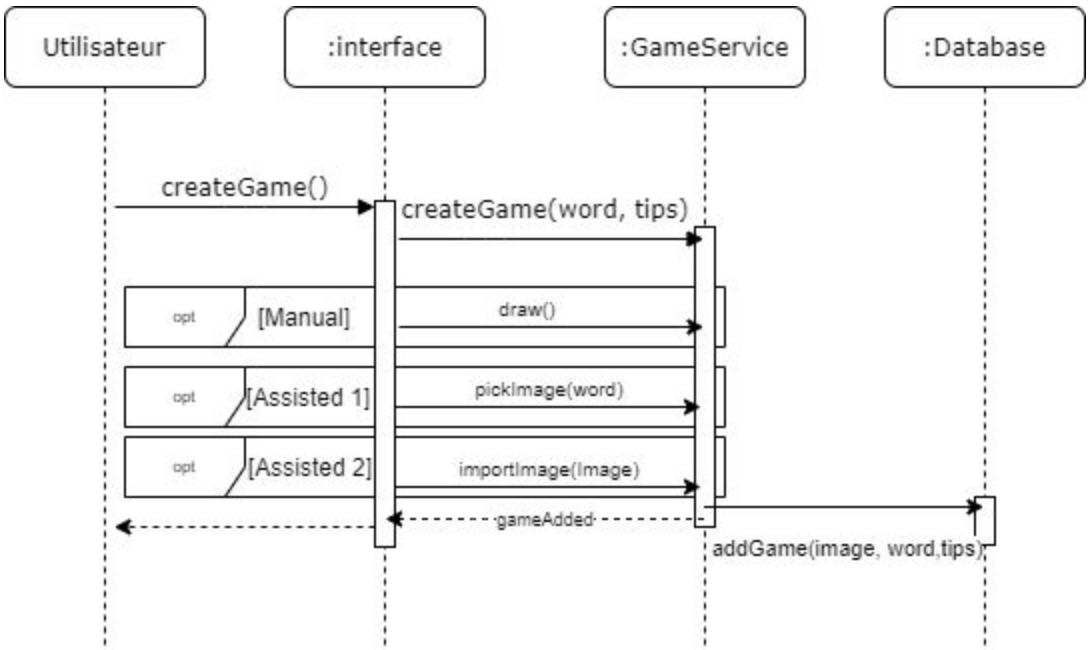

Figure 17: Diagramme de séquence système pour la création d'une partie

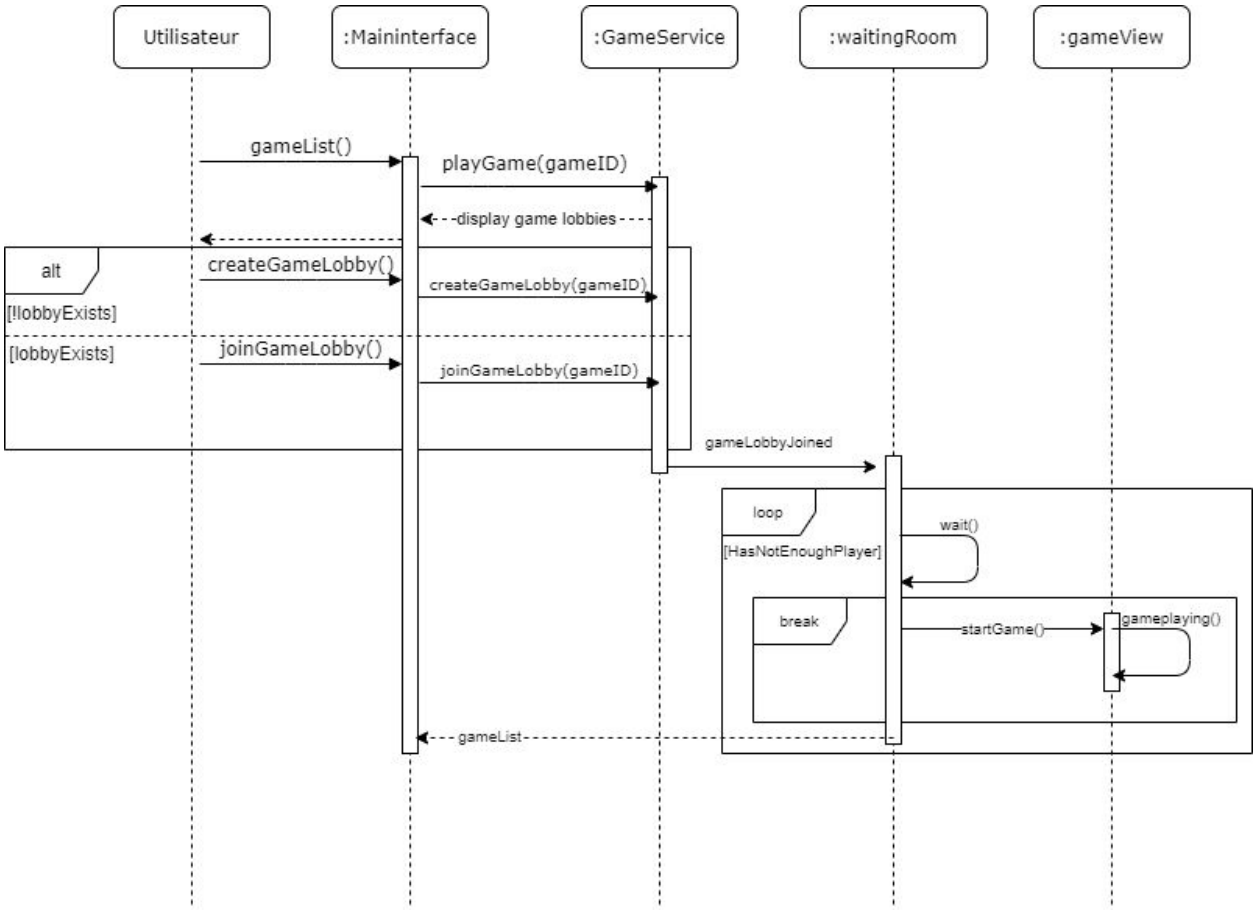

Figure 18: Diagramme de séquence système pour le lancement d'une partie

6. Vue de déploiement

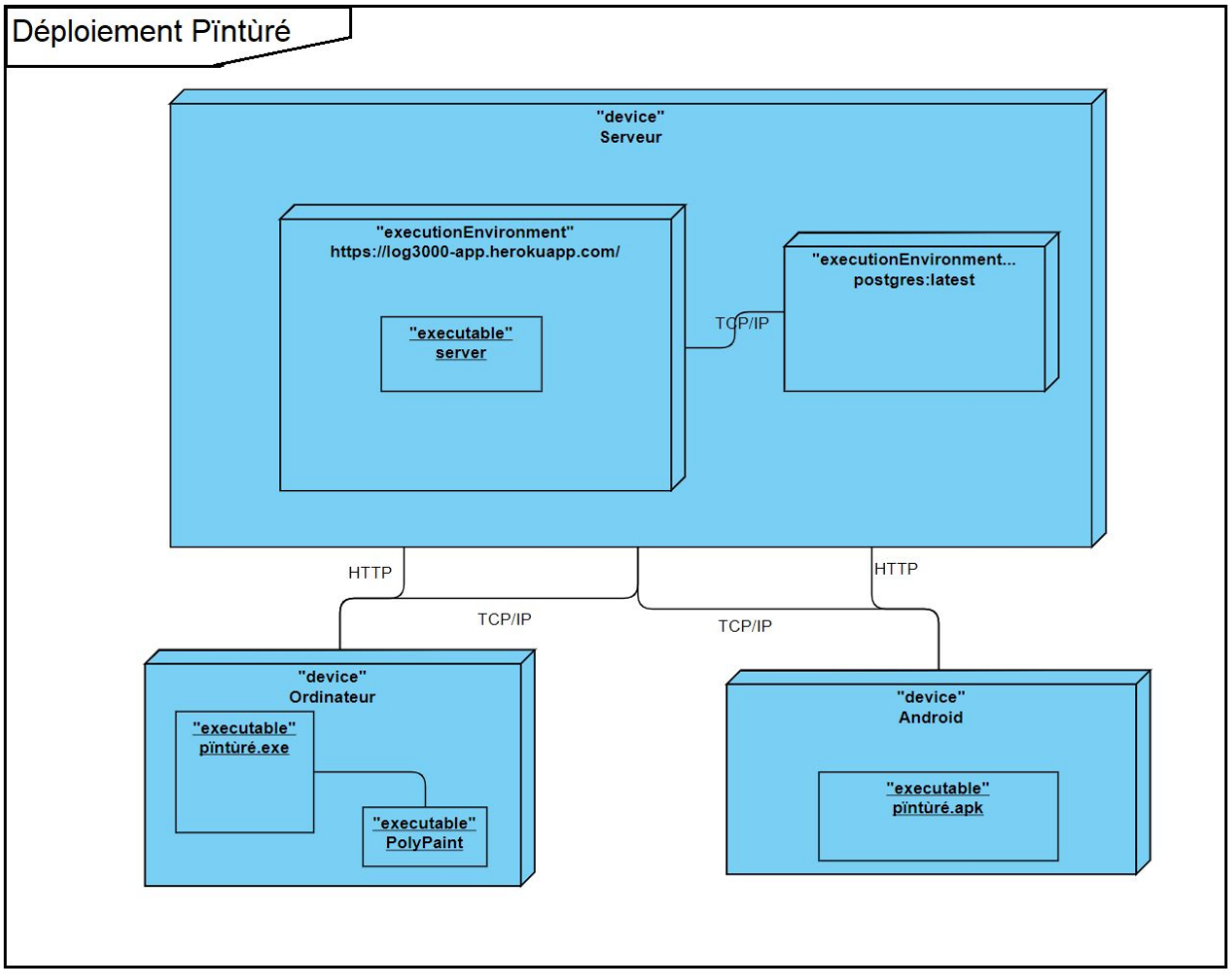

Figure 19: Diagramme de déploiement. (fait sur https://online.visual-paradigm.com/)

7. Taille et performance

7.1 Client léger

Nous développerons le client léger pour la tablette Galaxy Tab dont les caractéristiques sont données sur le site de Samsung[1]. Notre application ne devrait pas dépasser 100 Mo. Un 100 Mo de mémoire vive est nécessaire à l'exécution de l'application.

7.2 Client lourd

Le client lourd sera compatible avec un ordinateur sur Windows 10. Les caractéristiques minimales de l'ordinateur doivent être équivalentes, en terme de mémoire vive et de puissance du processeur, aux caractéristiques de la tablette Samsung Galaxy Tab A. L'application nécessite au moins

[1] https://www.samsung.com/ca/tablets/galaxy-taba-t290/SM-T290NZKAXAC/

200 Mo d’espace sur le disque dûr ainsi que 200 Mo de mémoire vive.

7.3 Serveur

Comme décrit dans la section 2.2.2 du plan de projet, la machine qui héberge le serveur est munie d’un Intel i5 8600K (“stock clock speed”) et possède 16 GB RAM en “Dual Channel” (2400 MHz). La machine se trouve dans un réseau d’une capacité de 30 Mb/s en téléchargement et 10 Mb/s en téléversement. L’application du serveur ne nécessitera pas plus de 200 Mo d’espace sur le disque dur et de 200 Mo de RAM. La base de données sera sur Postgresql sur Heroku. Elle est organisée en SQL.

7.4 Réseau

Le réseau dans lequel nos composantes communiquerons devrait avoir au moins une capacité de 10 Mb/s et un temps de latence d’au maximum 100 ms.

7.5 Base de Donnée

Le temps requis à la base de donnée pour effectuer les requêtes SQL devrait être au maximum d’une seconde.