Les dindons

PAINseau

Document d'architecture logicielle

Version 1.2

Historique des révisions

Date	Version	Description	Auteur
2021-02-07	1.0	Rédaction initiale de l'architecture logicielle.	Équipe 107
2021-02-16	1.1	Première révision en équipe de l'architecture logicielle	Équipe 107
2021-02-18	1.2	Version finale pour la réponse à l'appel d'offre	Équipe 107

Table des matières

Document d'architecture logicielle

1. Introduction

Ce document a pour but de présenter l'architecture du logiciel *PainSeau* sous forme de diagrammes conformes au langage UML. Il sera donc d'abord question de décrire les objectifs et les contraintes architecturaux, c'est-à-dire la sécurité, la confidentialité, la portabilité, etc. La prochaine section sera constituée de diagrammes de cas d'utilisation afin d'illustrer les différents cas d'utilisations et scénarios possibles à considérer lors de la conception de l'application. Puis, une vue logique de *PainSeau* sera également présentée avec des diagrammes de paquetages de haut niveau pour démontrer une idée générale de notre architecture. Par la suite, dans la section de la vue des processus, des diagrammes de séquence illustreront le système en termes d'interactions entre les différents processus. Quant à la section de la vue de déploiement, les configurations du matériel physique seront décrites par l'entremise de diagrammes de déploiement et finalement, une description des caractéristiques de taille et de performance sera présentée.

2. Objectifs et contraintes architecturaux

L'objectif ultime de ce projet est de concevoir une application multijoueur qui peut être jouée autant sur un ordinateur (client lourd) qu'une tablette Android (client léger). Donc, de manière plus détaillée, ces deux clients ont comme objectif de communiquer ensemble via un protocole assez solide qui est appuyé par un serveur.

Dans l'ensemble, l'objectif de notre équipe est de rédiger du code (en anglais) maintenable et réutilisable autant pour les clients que le serveur. Bien entendu, nous sommes encadrés par des contraintes temporelles et physiques, ce qui pourraient affecter le développement architectural. En effet, outre nos deux dates de remises officielles (19 février et 19 avril), nous devons établir nos propres dates limites, sans oublier que l'équipe ne se rencontre pratiquement jamais physiquement. Une mauvaise gestion de ces facteurs pourrait nuire à l'ensemble de l'architecture.

Serveur

Sachant que le serveur est au cœur de plusieurs fonctionnalités du jeu, nous avons identifié plusieurs objectifs et contraintes. L'objectif premier du serveur est d'assurer une bonne communication entre les deux clients. Il a également l'objectif de gérer le stockage de diverses données, et ce stockage ne doit pas affecter le protocole de communications entre le client lourd et léger.

Afin de répondre à ces objectifs, il faut tenir en compte des contraintes du serveur qui peuvent affecter l'architecture. Premièrement, le serveur doit supporter de nombreuses parties de jeu simultanément, ce qui correspond à une contrainte au niveau de la performance. Deuxièmement, plusieurs informations confidentielles des utilisateurs seront entreposées par le serveur, ce qui veut dire qu'il faudra procéder par des méthodes de « hashage » afin de préserver la sécurité de l'application. Finalement, comme énoncé dans nos exigences, nous avons posé comme contrainte que la base de données, hébergée par Heroku, doit être disponible 99.9% du temps afin d'assurer une fiabilité constante face à la réception des requêtes.

Client lourd et léger

Pour les deux clients, nous avons comme objectifs d'implémenter toutes nos exigences en termes de fonctionnalités, ainsi que 50% des exigences souhaitées. Évidemment, toutes les fonctionnalités devront fonctionner comme nous l'avons spécifié dans notre liste d'exigences et elles ne doivent pas se briser lors de leur utilisation. D'autre part, les clients lourd et léger ont aussi l'objectif d'offrir une expérience multijoueur convenable et agréable, c'est-à-dire en permettant aux utilisateurs de collaborer en temps réel d'une façon stable.
Comme pour le serveur, les clients doivent respecter la contrainte de sécurité à l'égard des données confidentielles transmises par les usagers. Ainsi, ces données devront être hachées sur chaque client avant d'être envoyées au serveur. Du côté du client lourd, l'implémentation des fonctionnalités liées au dessin est contrainte à une architecture préexistante, puisque nous allons développer à partir d'une application (*Polydessin*) développée d'avance. Enfin, le client lourd sera développé sur Visual Studio Code avec l'utilisation du langage Typescript et du cadriciel Angular, et le client léger sur Android Studio avec Kotlin. Cela est une contrainte qui impacte l'architecture globalement sachant que nous allons construire l'architecture de nos clients selon les normes de ces technologies.

3. Vue des cas d’utilisation

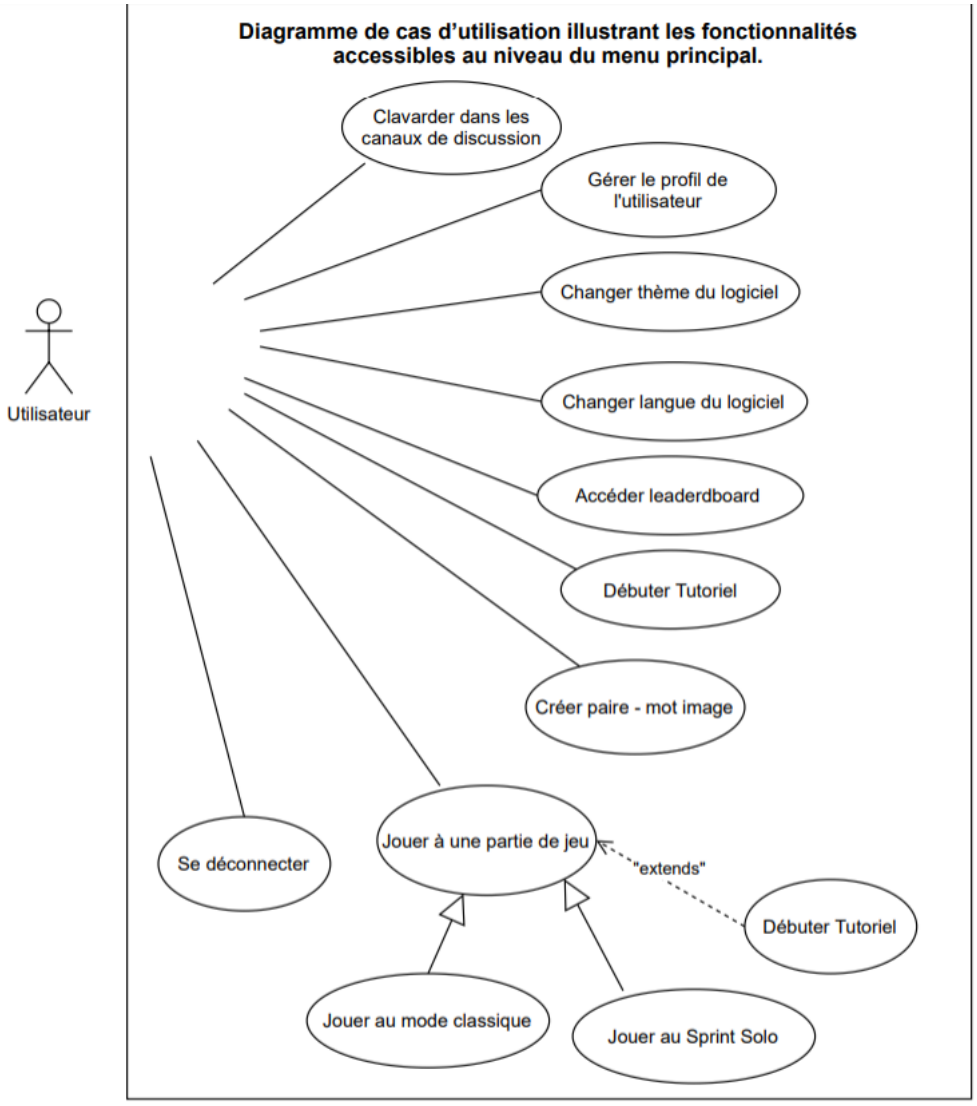

Figure 1. Diagramme de cas d’utilisation illustrant les fonctionnalités accessibles au niveau du menu principal.

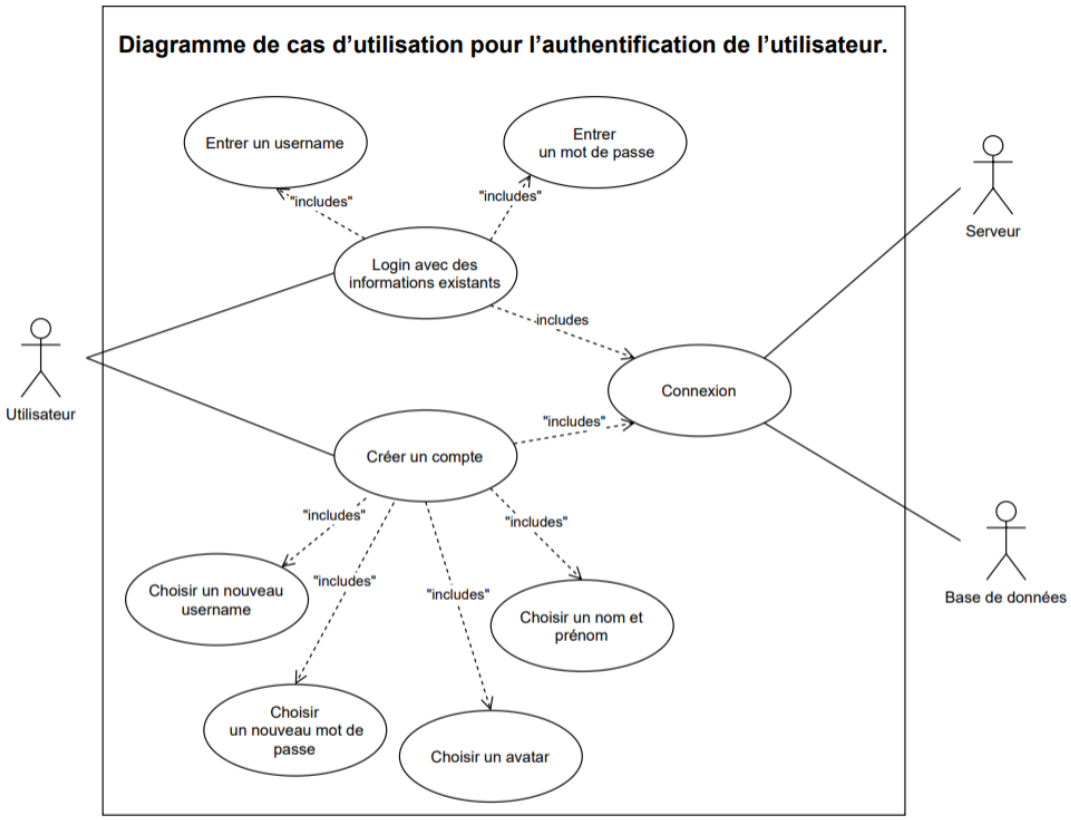

Figure 2. Diagramme de cas d'utilisation pour l'authentification de l'utilisateur.

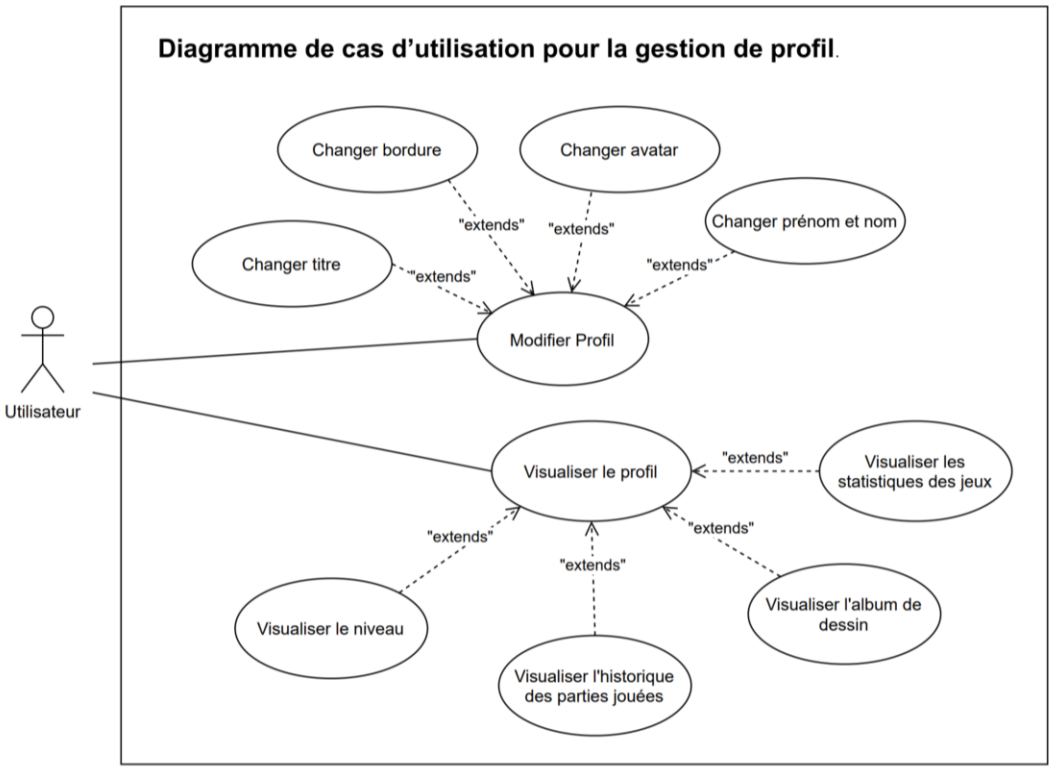

Figure 3. Diagramme de cas d'utilisation pour la gestion de profil.

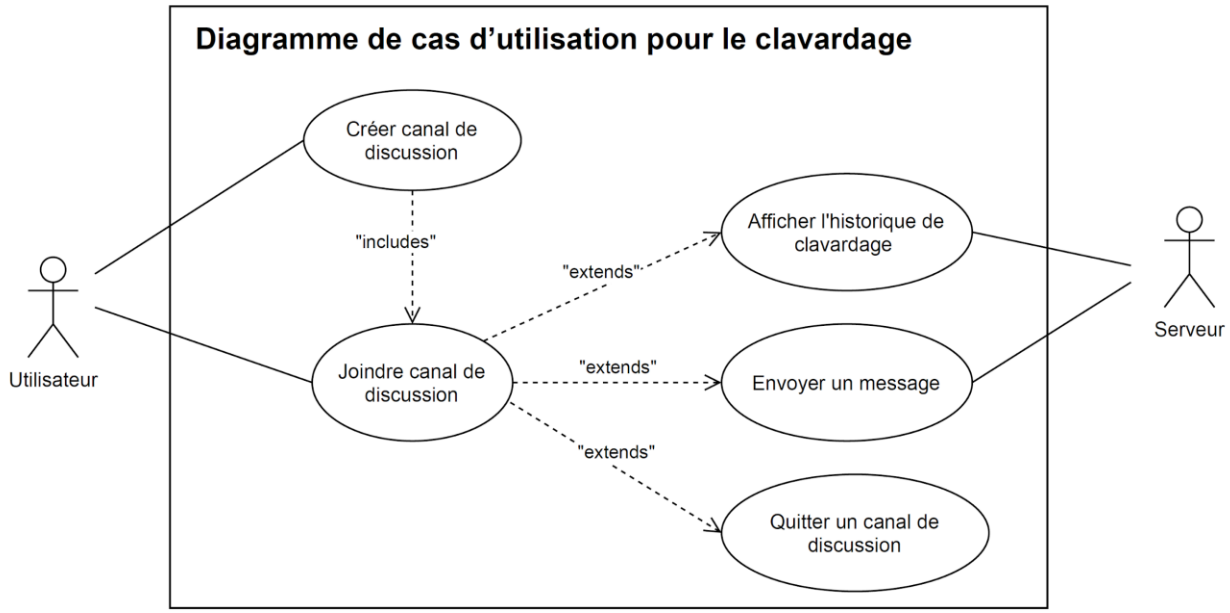

Figure 4. Diagramme de cas d'utilisation pour le clavardage.

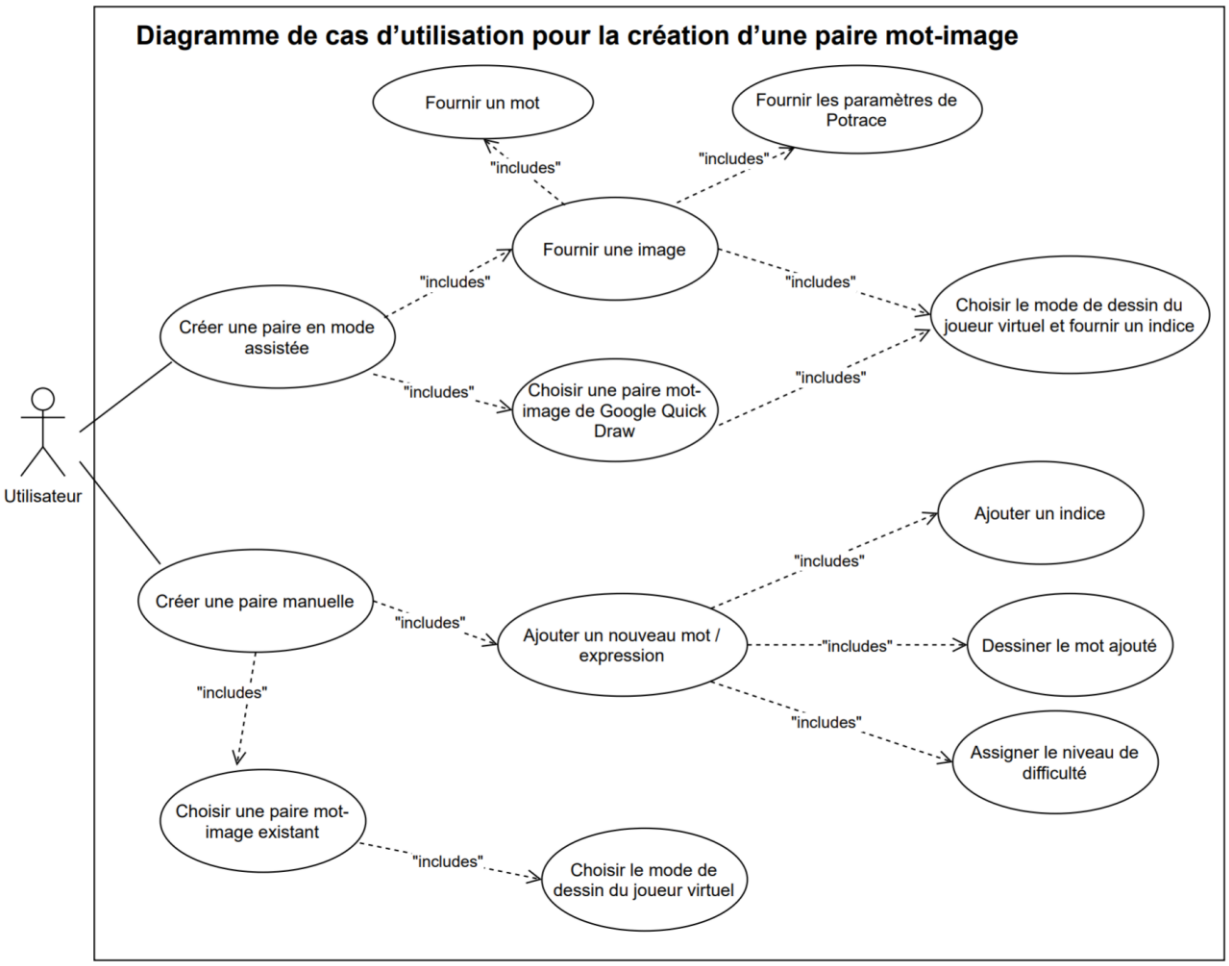

Figure 5. Diagramme de cas d'utilisation pour la création d'une paire mot-image.

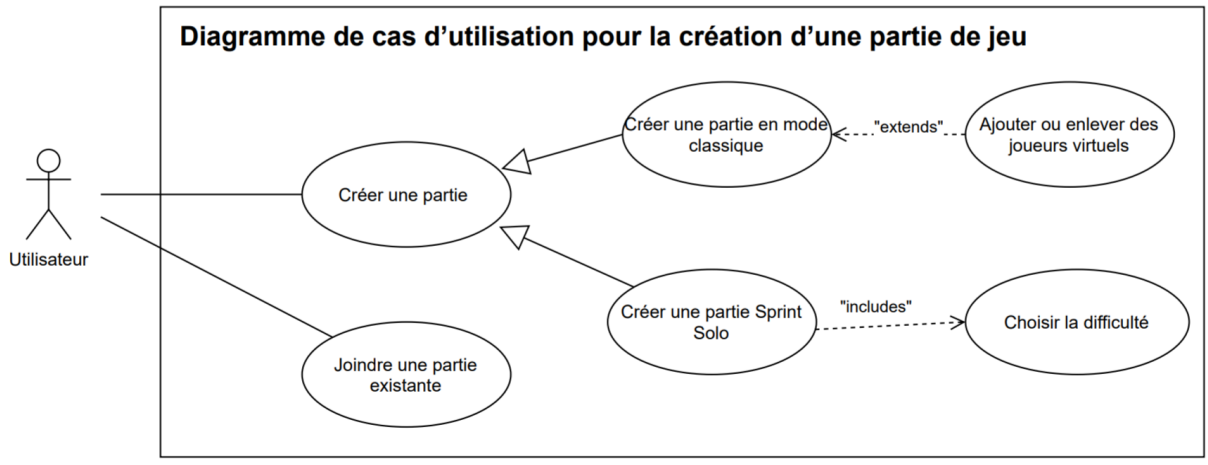

Figure 6. Diagramme de cas d'utilisation pour la création d'une partie de jeu.

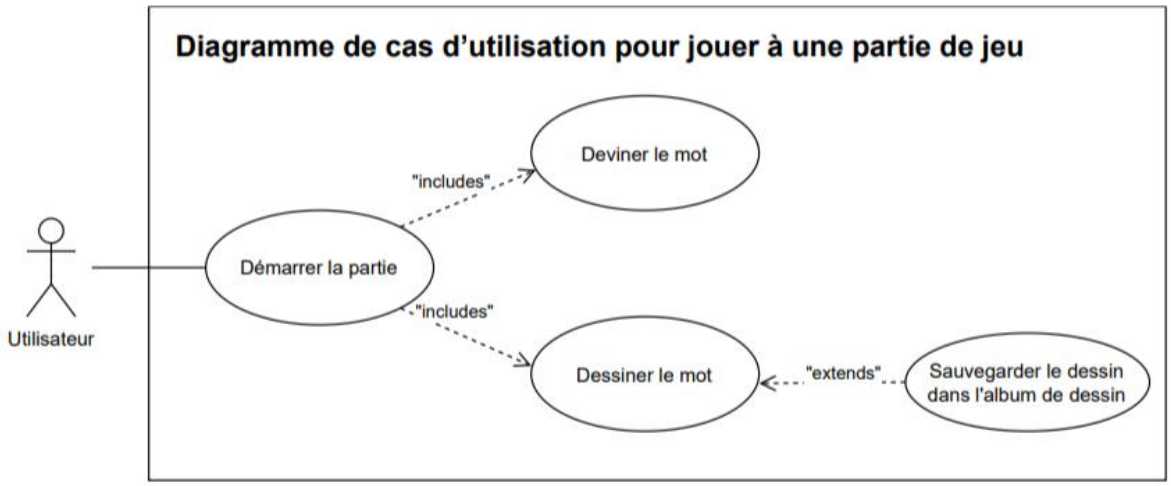

Figure 7. Diagramme de cas d'utilisation pour jouer à une partie de jeu.

4. Vue logique

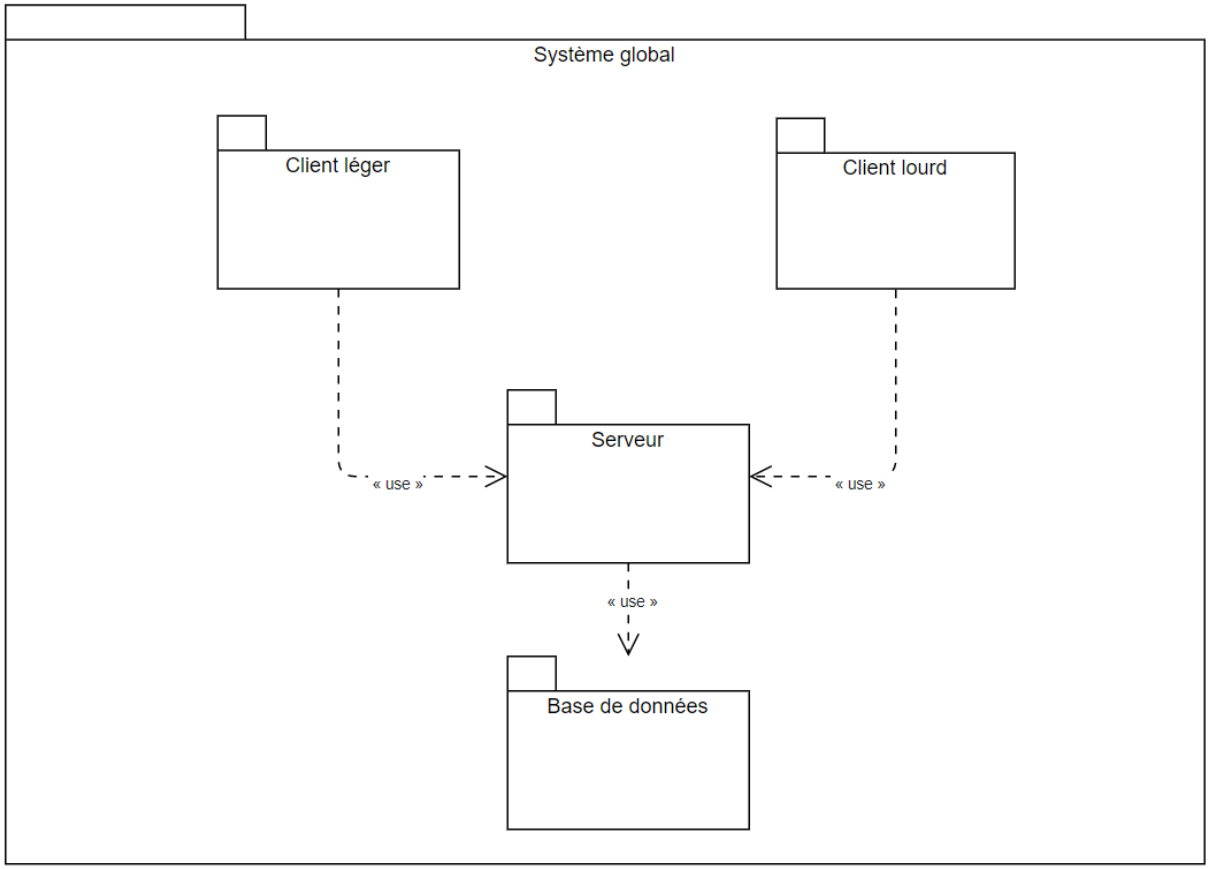

Figure 8. Diagramme de paquetage pour le système global.

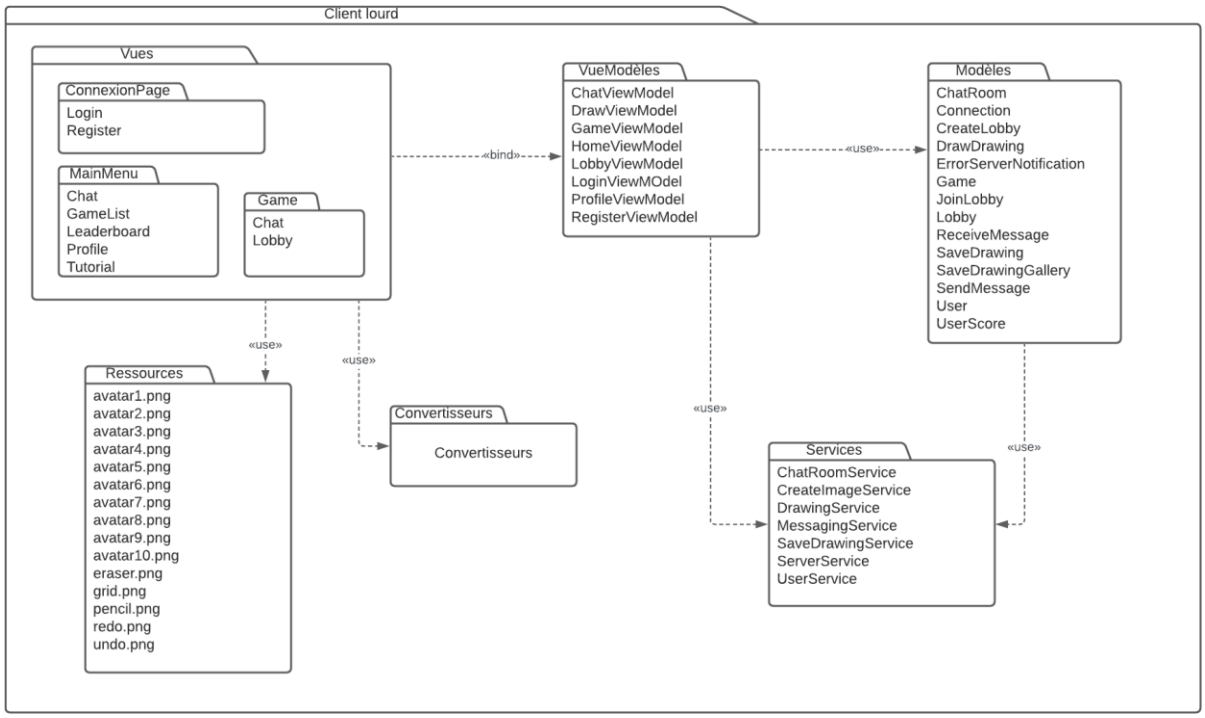

Figure 9. Diagramme de paquetage pour le client lourd.

<Vue>
Ce paquetage regroupe toutes les différentes fenêtres et pages de l'application. Les fichiers *code-behind* s'y retrouvent également (XAML).

<VueModèle>
Ce paquetage regroupe les classes VueModèles de l'architecture MVVM, liant le modèle et la vue ensemble.

<Modèle>
Ce paquetage est composé de toutes les classes s'occupant de la logique de l'application, qui est complètement séparée de la vue.

<Services>
Ce paquetage contient les différents services de notre application, ayant chacun une fonctionnalité.

<Convertisseurs>
Ce paquetage est composé des convertisseurs nécessaires aux liaisons de données.

<Ressources>
Ce paquetage regroupe les différentes images, sons et icônes nécessaires à l'interface de notre application.

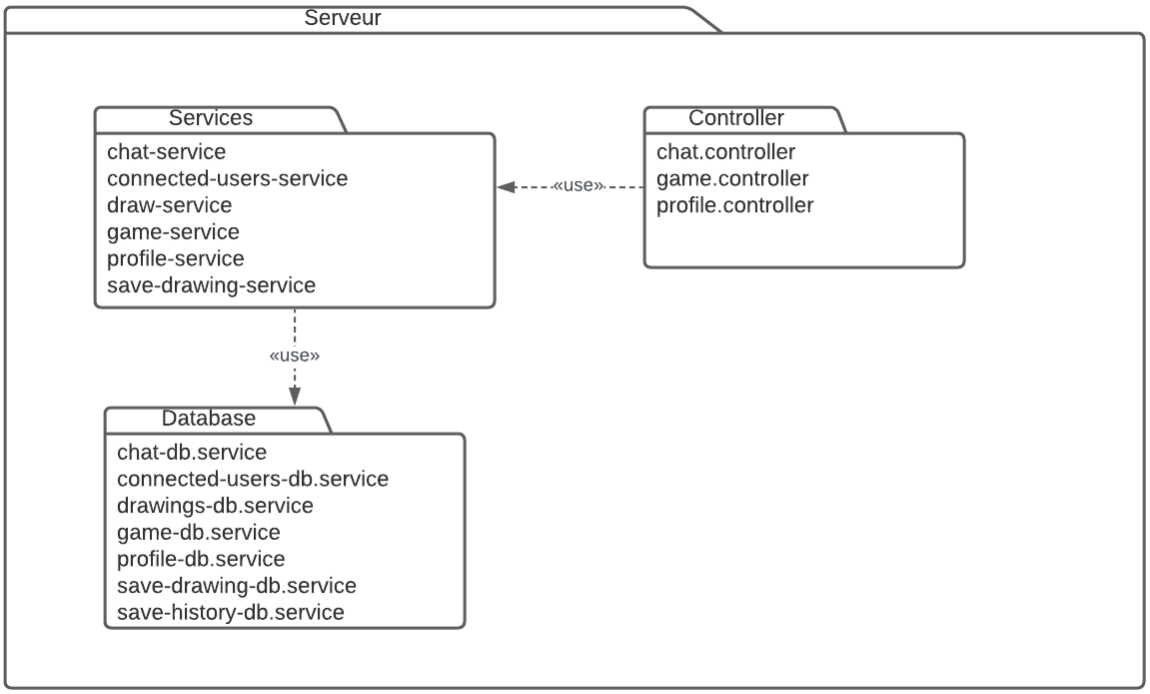

Figure 10. Diagramme de paquetage pour le serveur.

<Database>
Ce paquetage contient les différents services permettant de communiquer avec notre base de données.

<Services>
Ce paquetage regroupe les différents services permettant de gérer les messages, les dessins et les profils des utilisateurs.

<Services>
Ce paquetage contient les différents services de notre application, ayant chacun une fonctionnalité.

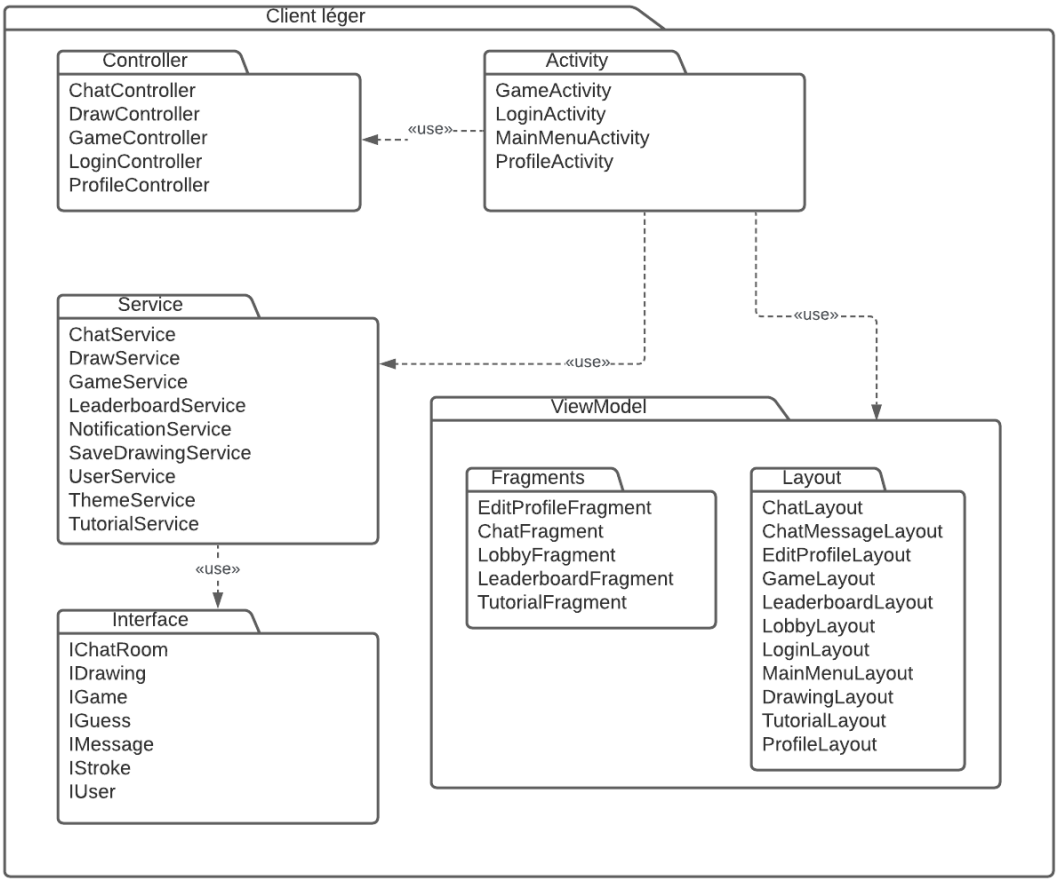

Figure 11. Diagramme de paquetage pour le client léger.

<Controller>
Ce paquetage contient les différents contrôleurs du client léger.

<Activity>
Ce paquetage regroupe toutes les différentes activités que peut entreprendre le client léger.

<ViewModel>
Ce paquetage contient les layouts et les fragments constituant les affichages utilisés par les activités.

<Services>
Ce paquetage contient les différents services formant la partie logique du client léger.

<Interfaces>
Ce paquetage contient les différentes interfaces que contient le client léger.

5. Vue des processus

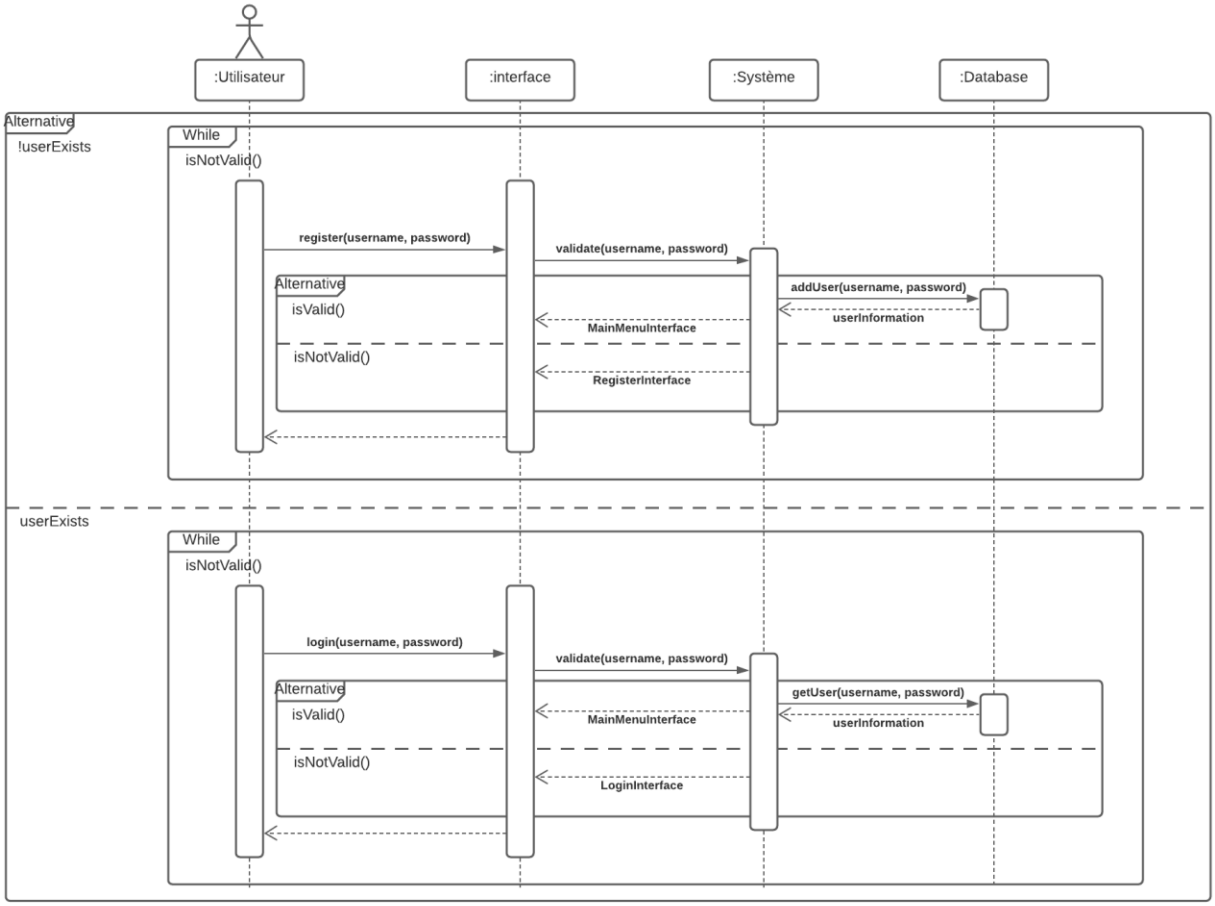

Figure 12. Diagramme de séquence pour l'authentification d'un utilisateur.

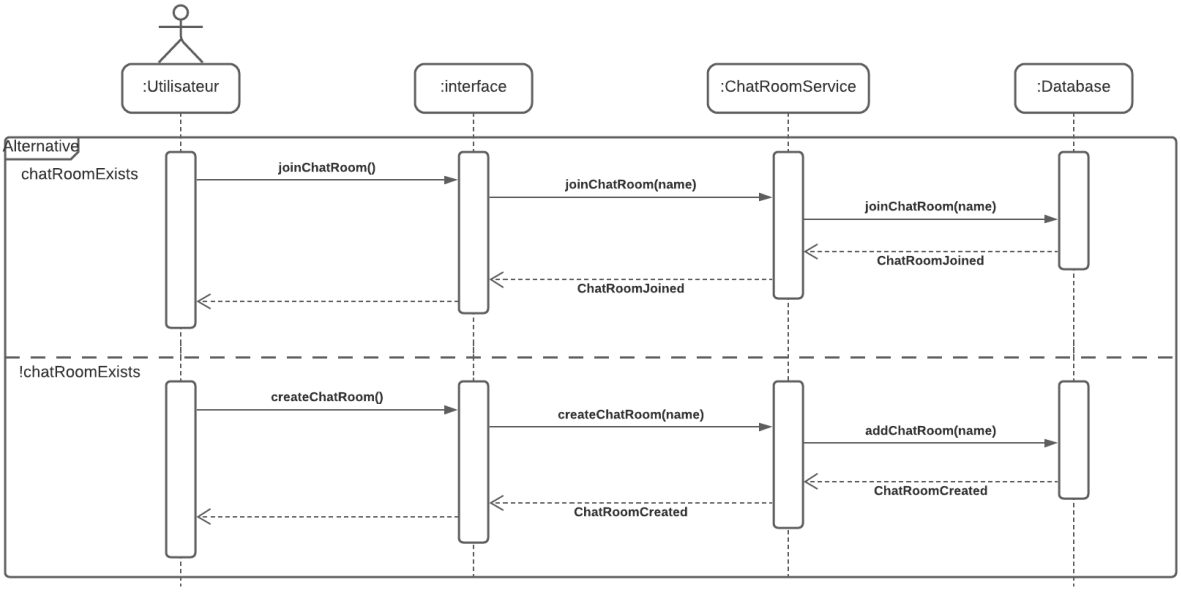

Figure 13. Diagramme de séquence pour la création ou rejoindre un salon de clavardage déjà existant.

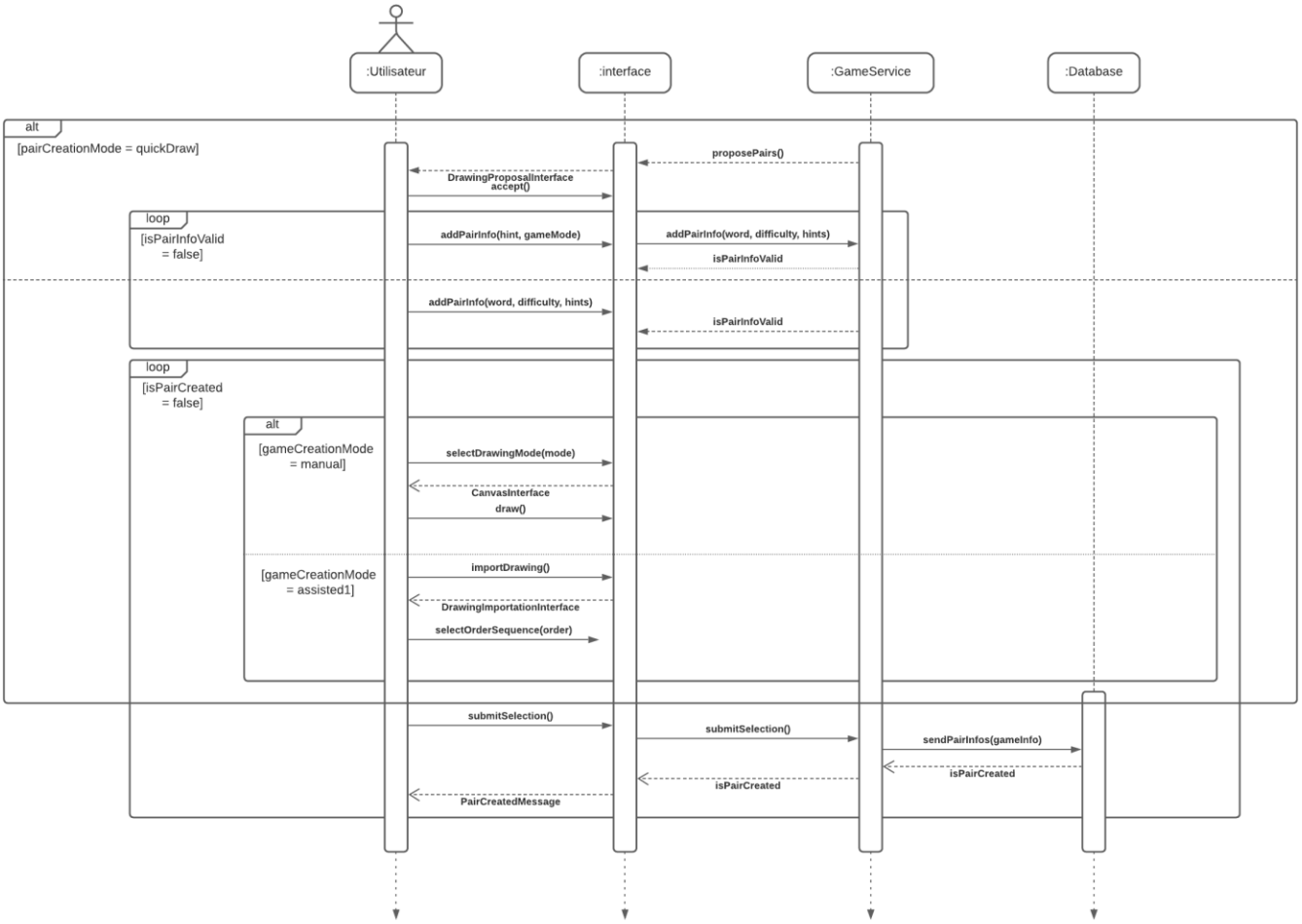

Figure 14. Diagramme de séquence pour la création d'une paire mot-image.

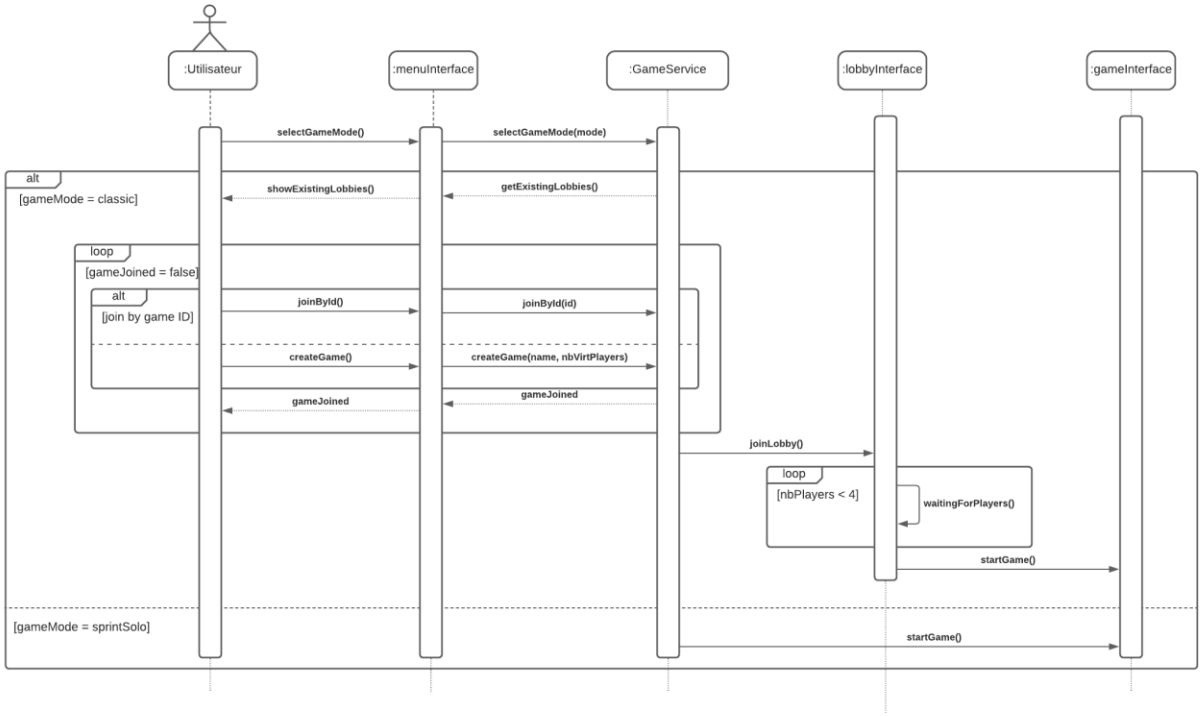

Figure 15. Diagramme de séquence pour le lancement d'une partie.

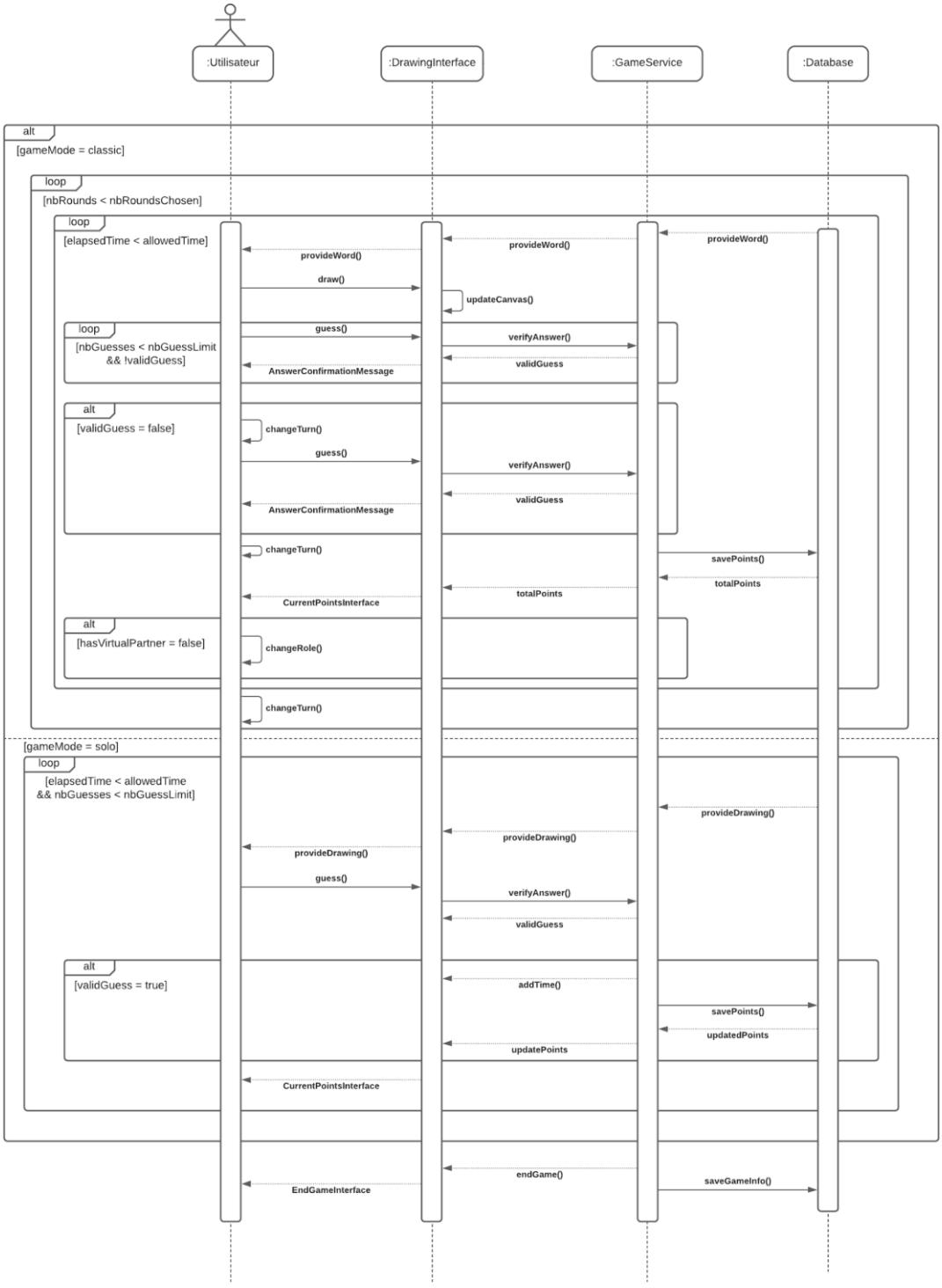

Figure 16. Diagramme de séquence du déroulement d'une partie de jeu.

Note: Lorsqu'une équipe réplique, elle n'a droit qu'à un essai.

6. Vue de déploiement

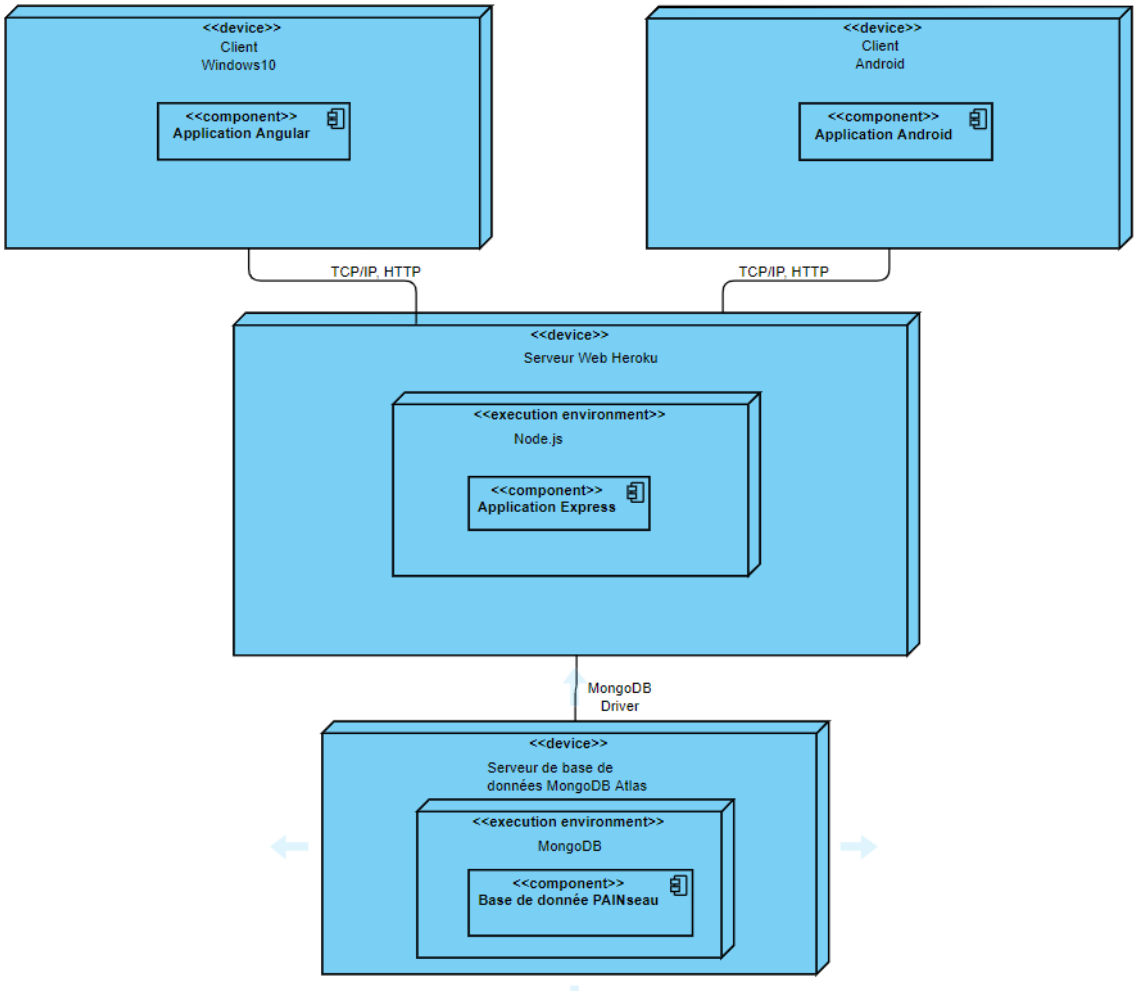

Figure 17. Diagramme de déploiement du système global.

7. Taille et performance

L'architecture logicielle et le design du logiciel doivent respecter les contraintes générales décrites dans les parties 2.3 et 2.4 du SRS. L'architecture du client a été conçue pour respecter les contraintes de la mémoire, l'espace et les périphériques des machines décrites dans le SRS. L'expérience de l'utilisateur doit être fluide dans une partie de dessin autant avec le client léger que le client lourd. Cela veut dire qu'il y a un délai maximum de 100 ms entre le point de contact et l'apparition du trait, et un délai maximum de 500 ms pour la réponse du serveur lors des requêtes. Il ne devrait pas avoir des délais notables durant les parties de jeu ni dans les canaux de discussions. L'interface utilisateur doit donc afficher à tous les participants ce que l'utilisateur envoie dans le canal de discussion dans un délai de 100ms. L'architecture du serveur a été conçue pour pouvoir supporter au plus 10 utilisateurs sans problème de performance durant les communications dans les canaux de discussion ou durant le déroulement d'une partie de jeu. Le logiciel doit utiliser un maximum de 1GB de mémoire vive et doit utiliser moins de 500 MB d'espace pour le client léger. Quant au client lourd, le logiciel doit utiliser un maximum de 2GB de mémoire vive et doit utiliser moins de 1GB d'espace sur le disque dur. Notre architecture client-serveur permet donc de satisfaire pleinement l'utilisateur.