Équipe 111 (Stack Overwork)

Projet d’évolution logiciel
Document d'architecture logicielle

Version 2.3

Historique des révisions

Date	Version	Description	Auteur
2021-02-16	1.0	Ajout de l'introduction, les objectifs et contraintes, la vue des cas d'utilisation et la vue des processus	Raphael Lasalle Hugo Palisson Theo Turell
2021-02-16	1.1	Ajout des descriptions des paquetages	Jaafar Kaoussarani Raphael Lasalle
2021-02-16	1.2	Ajout de diagrammes de paquetages et de classes (A1-A4, A11) à l'annexe	Jaafar Kaoussarani Raphael Lasalle David Fraval
2021-02-16	1.3	Ajout de diagrammes de classes (A5, A6, A7) et révision de l'annexe	Jaafar Kaoussarani
2021-02-17	1.4	Ajout de diagrammes de classes (A8) et quelques changements visuels aux autres diagrammes de l'annexe	Jaafar Kaoussatani Raphael Lasalle
2021-02-17	1.5	Ajout de diagrammes de classe (A9, A10) et ajouts à certains des diagrammes de l'annexe (A1, A2, A5)	Jaafar Kaoussarani
2021-02-18	1.6	Ajouts à la section 7	Jaafar Kaoussarani
2021-02-18	1.7	Ajouts à la section 7 et révision du diagramme A10	Jaafar Kaoussarani
2021-04-14	2.0	Ajustements au diagramme de déploiement	Jaafar Kaoussarani
2021-04-14	2.1	Ajustements aux sections 1 et 2	Jaafar Kaoussarani
2021-04-14	2.2	Ajustements à la section 7 pou mettre plus de chiffres et parler plus du serveur et la base de données.	Jaafar Kaoussarani
2021-04-14	2.3	Ajustement des diagrammes A1 à A14, ainsi que leurs déscription	Jaafar Kaoussarani

Table des matières

1. Introduction **4**

2. Objectifs et contraintes architecturaux **4**

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

4. Vue logique **5**

5. Vue des processus **8**

6. Vue de déploiement **9**

7. Taille et performance **9**

Annexe **11**

Références **21**

Document d'architecture logicielle

1. Introduction

Ce document présente les l'architecture de l'application *Fais-moi un dessin*. Pour se faire, nous présenterons les objectifs et contraintes architecturaux avant de passer à divers diagrammes. Cela inclue un diagramme de cas d'utilisation, un diagramme de paquetage, plusieurs diagrammes de classes, un diagramme de séquence et un diagramme de déploiement. Nous terminerons par une présenterons des caractéristiques de taille et de performance pouvant avoir un impact sur l'architecture et le design de l'application.

2. Objectifs et contraintes architecturaux

L'utilisabilité est la contrainte la plus importante qui affecte la manière dont les menus et interfaces sont structurés. Pour que cette contrainte soit respectée, les clients doivent posséder des menus et interfaces cohérents qui doivent être réutilisés autant que possible et qui doivent avoir une certain cohérence visuelle et fonctionnelle. Par exemple, l'interface de dessin est commune aux parties et à la création d'une paire mot-image. Cela est aussi vrai au niveau des menus communs aux deux clients, qui doivent être relativement similaires. De plus, il est important de considérer une architecture favorisant l'implémentation de techniques d'optimisation. L'application peut se retrouver à faire des opérations lourdes ou à afficher plusieurs composantes visuels complexes en même temps, ce qui pourrait très facilement nuire à son utilisabilité. Par la suite, le fait que les deux applications, qui utilisent des langages de programmation différents (TypeScript pour le client lourd, Kotlin pour le client léger), doivent communiquer avec un serveur centralisé fait en sorte que l'architecture des deux clients et du serveur doit être conçus de manière à favoriser une communication simple et fluide (Par exemple, en utilisant des librairies JSON sur le client léger afin d'assurer que les données transmises au serveur et recueillies de ce dernier aient une structure uniforme, ce qui a nécessairement un impact sur l'architecture de tout les clients et du serveur lui-même).

3. Vue des cas d'utilisation

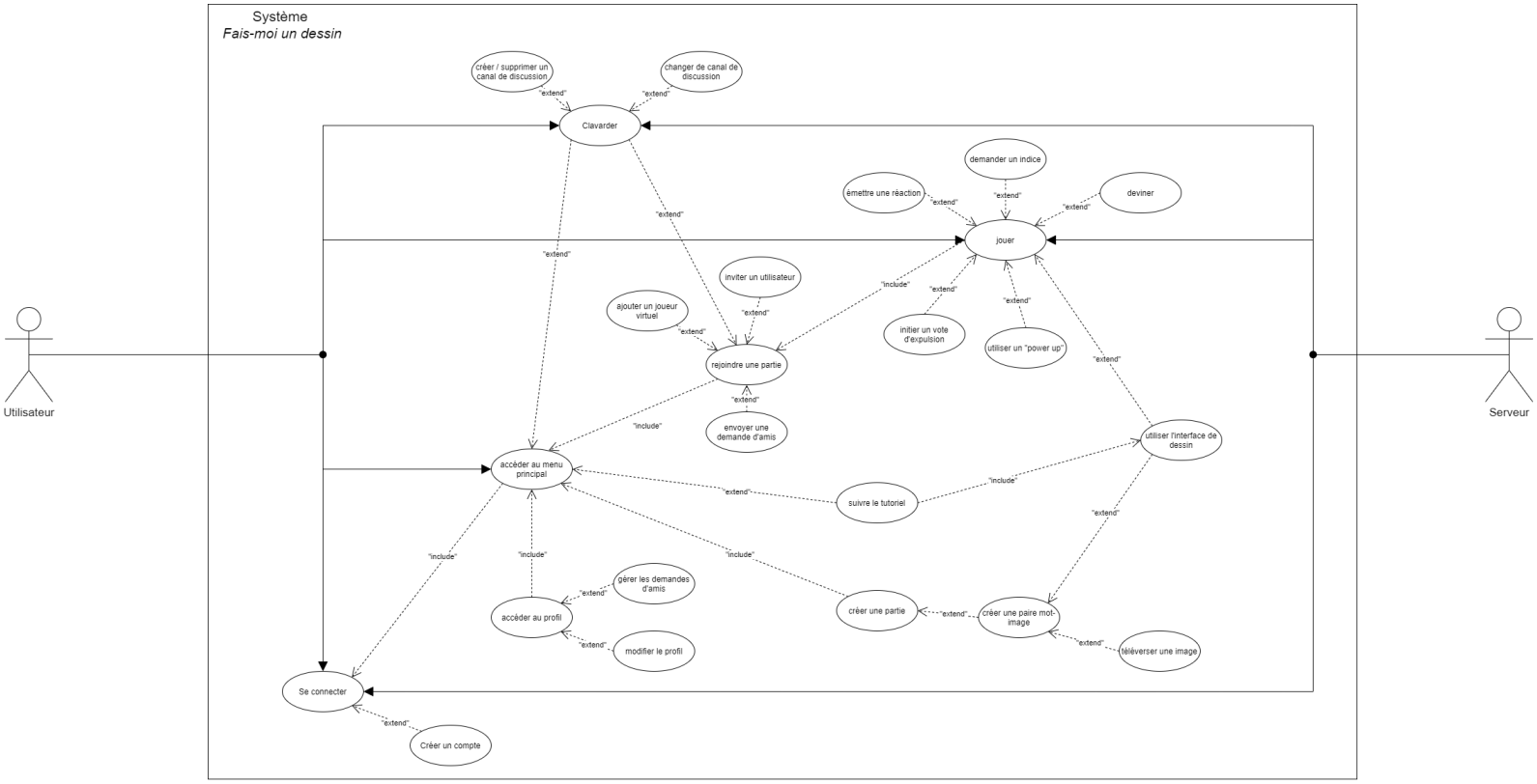

Figure 1 : Diagramme de cas d'utilisation de l'application *Fais-moi un dessin*

4. Vue logique

Voir l'annexe A (A.1 pour le client lourd, A.2 pour le client léger) pour une vue d'ensemble des paquetages du système, ainsi que leurs interactions. Dans la section qui suit, nous décrirons les responsabilités des paquetages et présenterons un diagramme pour chacun de ces dernier.

IU (Client Lourd)
Il s'agit de l'interface utilisateur du client lourd. Ce paquetage est responsable de la présentation visuelle des données reçues et des champs de saisie. L'utilisateur interagit avec le reste de l'application en utilisant ce paquetage. Voir l'annexe (A.3) pour le diagramme de classe.

IU (Client Léger)
Il s'agit de l'interface utilisateur du client léger. Ce paquetage est responsable de la présentation visuelle des données reçues et des champs de saisie. L'utilisateur interagit avec le reste de l'application en utilisant ce paquetage. Voir l'annexe (A.4) pour le diagramme de classe.

Gestionnaire social

Il s'agit des classes et services permettant de gérer le système d'amis. Ce paquetage est responsable de la transmission des requêtes d'amis, de l'acceptation de ces requêtes, de la liste d'amis, du statut de connection d'un utilisateur donné et de ses préférences. Il est intermédiaire entre l'interface utilisateur et la base de données.

Voir l'annexe (A.5) pour le diagramme de classe.

Gestionnaire de chat

Il s'agit des classes et services permettant de gérer le système de chat. Ce paquetage est responsable de la transmission des messages vers les bonnes parties et les bons canaux, de gérer les utilisateurs ayant accès à un canal donné et de la réception de messages. Il est intermédiaire entre l'interface utilisateur et le serveur.

Voir l'annexe (A.6) pour le diagramme de classe.

Créateur de paires mot-image

Il s'agit des classes et services permettant à l'utilisateur de créer les paires mot-image qui seront utilisées par les joueurs virtuels. Ce paquetage utilise l'interface de dessin pour les cas où le dessin de la paire est réalisée manuellement, et permet également de générer une image à partir d'un fichier téléversé par l'utilisateur.

Voir l'annexe (A.7) pour le diagramme de classe.

Gestionnaire d'authentification

Il s'agit des classes et services gérant les connexions des utilisateurs à l'application. Il vérifie que le nom d'utilisateur et le mot de passe entrés sont valides et s'assure qu'un compte peut seulement être connecté sur un client à la fois. il communique avec la base de données pour la validation (nom d'utilisateur et mot de passe) et avec le collecteur de statistiques pour que ce dernier enregistre les connexions / déconnexions.

Voir l'annexe (A.8) pour le diagramme de classe.

Collecteur de statistiques

Il s'agit des classes et services permettant de collecter des statistiques sur les joueurs au cours de leur utilisation de l'application. Il permet notamment de collecter le temps passé à utiliser l'application, le pourcentage de parties gagnées, un historique des connections / déconnections, le temps moyen d'une partie ainsi qu'un historique détaillé des parties jouées. Il accède au gestionnaire de partie, au gestionnaire social en temps réel et au service d'authentification pour collecter des statistiques, et envoie ces dernières à la base de données.

Voir l'annexe (Figure A.9) pour le diagramme de classe.

Gestionnaire de partie

Il s'agit des classes et services permettant de gérer une partie. Ce paquetage est responsable du séquencement d'un match, de l'organisation des joueurs (tant réels que virtuels) et de la transmission des dessins et des réponses. Il est intermédiaire entre l'interface de dessin et le serveur.

Voir l'annexe (A.10) pour le diagramme de classe.

Serveur

Ce paquetage est responsable de la transmission de données entre les client, la gestion de l'état de connection et la sélection des formats des objects contenus dans les paquets de communication. Il est aussi intermédiaire avec la base de données.

Voir l'annexe (A.11) pour le diagramme de classe.

Base de données

Comme son nom l'indique, ce paquetage contient la base de données qui renferme les données des profils utilisateurs, des joueurs virtuels, des parties jouées et des paires mot-image créées par les utilisateurs.

Voir l'annexe (A.12) pour le diagramme de classe.

Interface de dessin (Client Lourd)

Il s'agit des classes et services responsables des fonctionnalités de dessin sur le client lourd, c.-à-d. le crayon, l'efface, la fonctionnalité annuler-refaire, la grille et la sélection de couleurs. Ce paquetage est utilisé par le créateur de paires mot-image et le gestionnaire de parties.

Voir l'annexe (A.13) pour le diagramme de classe.

Interface de dessin (Client Léger)

Il s'agit des classes et services responsables des fonctionnalités de dessin du client léger, c.-à-d. le crayon, l'efface, la fonctionnalité annuler-refaire, la grille et la sélection de couleurs. Ce paquetage est utilisé par le gestionnaire de parties.

Voir l'annexe (A.14) pour le diagramme de classe.

5. Vue des processus

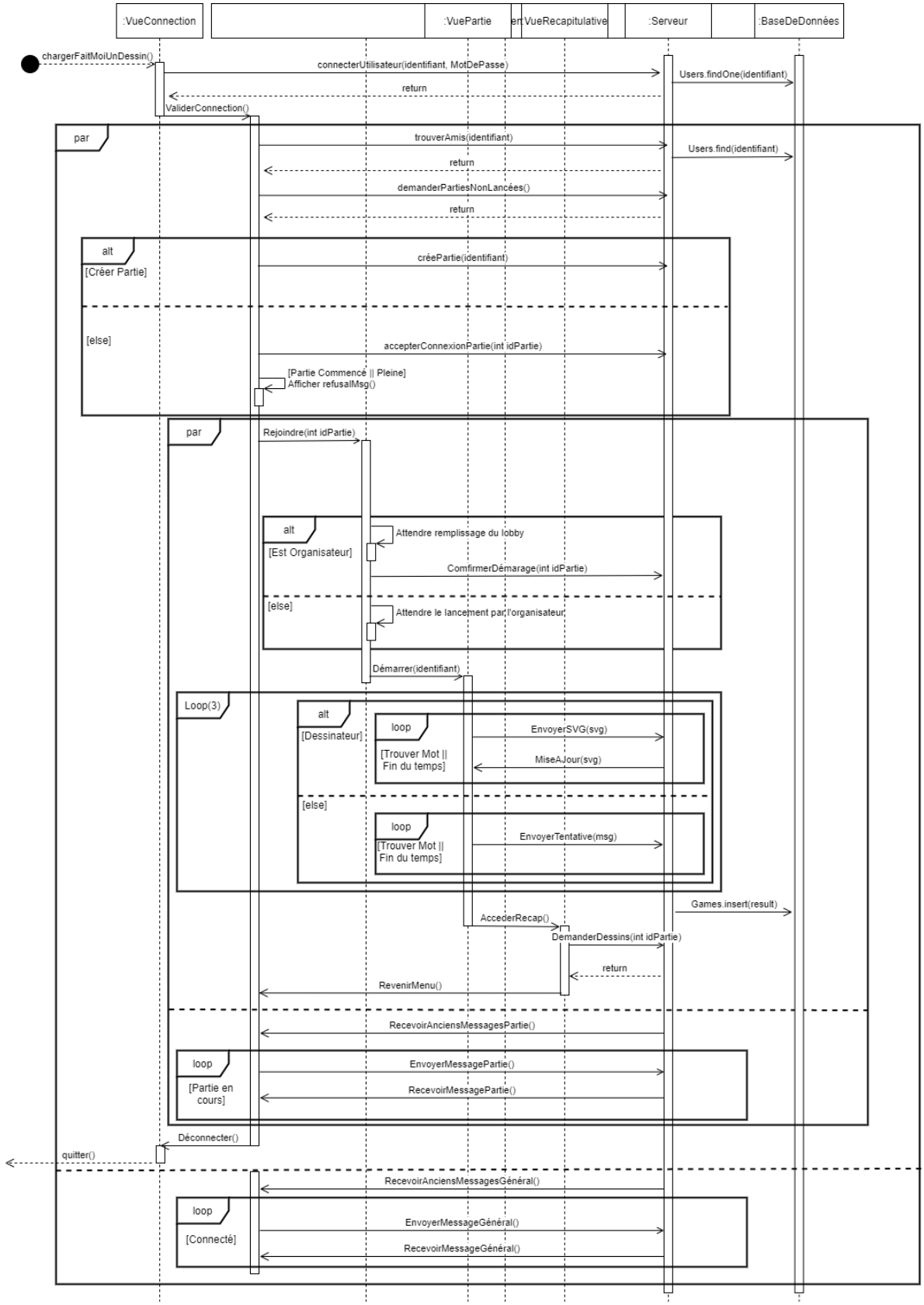

Figure 2 : Diagramme de séquence de l'application *Fais-moi un dessin*.

6. Vue de déploiement

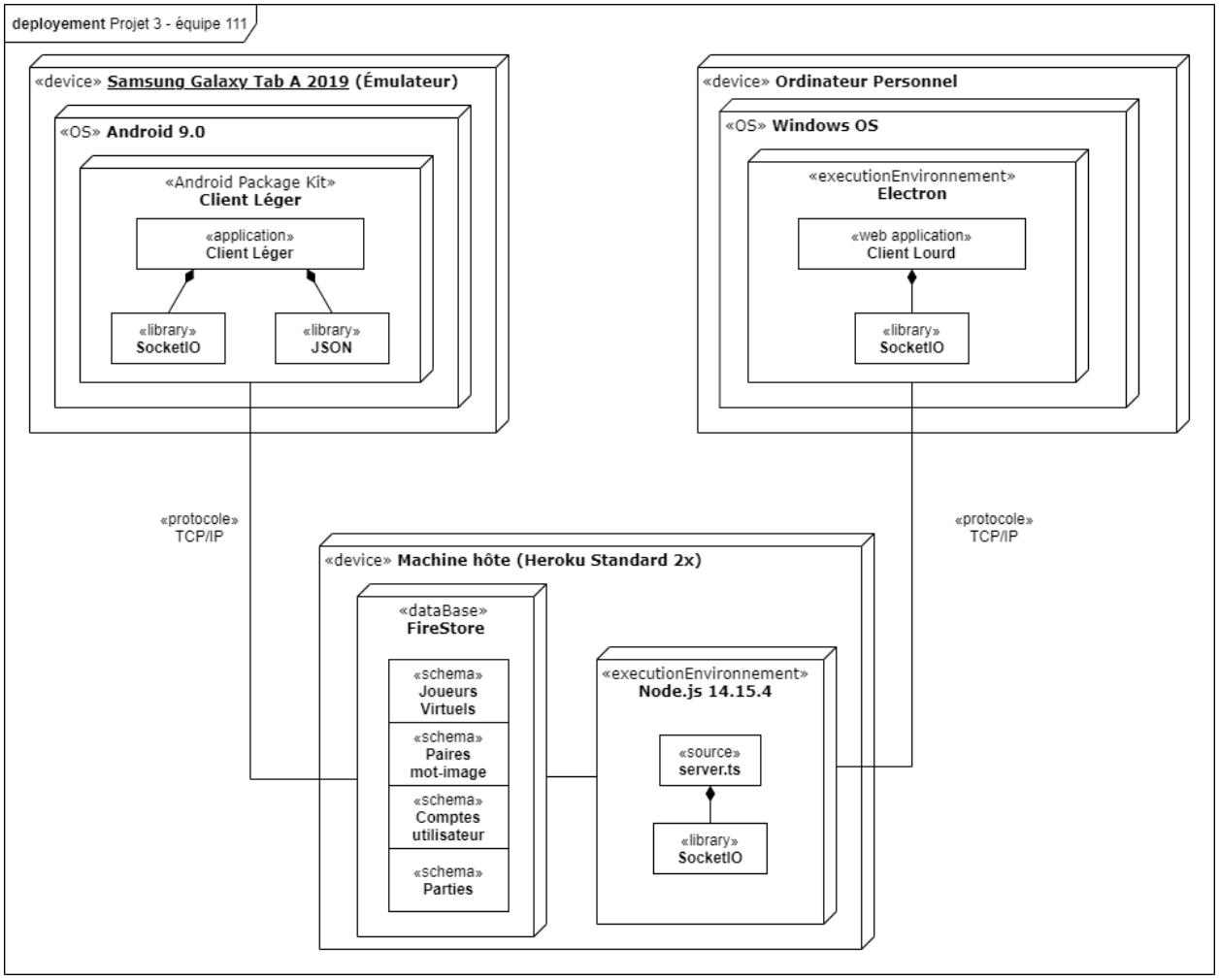

Figure 3: Diagramme de déploiement de l'application *Fais-moi un dessin.*

7. Taille et performance

Un aspect de performance pouvant impacté notre architecture serait la latence qui pourrait potentiellement être créée lors de la création de dessins dans des parties se déroulant simultanément. En effet, les attributs d'un objet SVG peuvent rapidement devenir extrêmement grands. L'impact sur la performance du serveur deviendrait exponentiellement plus grand au fur et à mesure que ces objets se multiplient et deviennent plus complexe. Il advient donc d'essayer de transmettre des petits bouts d'information SVG et de les reconstruire au fur et à mesure du côté des clients, plutôt que de tout simplement transmettre les objets directement. Des tests préliminaires sur la machine hôte du serveur ont démontré qu'on peut s'attendre à un temps de réponse maximal de 1.5 à 2 secondes dans les pires cas, ce qui veut dire que l'on doit concevoir une architecture de serveur qui garantit que les utilisateurs ne puissent pas faire des actions s'il y a un grand temps de latence (Par exemple, essayer de deviner plusieurs fois pendant que le serveur est « bloqué » pour recevoir plus de points).

Parallèlement, il advient de s'assurer que cette reconstruction et l'affichage des primitives sur le canvas se font d'une façon optimale. En effet, en considérant la mémoire vive relativement limitée du client léger (2GB), il serait fort intéressant de considérer un design introduisant des techniques d'optimisation au niveau de l'affichage des

primitives SVG et toute autre collection de structures complexes. Cela s'applique aussi au niveau du client lourd, puisqu'il n'y a aucune garantie que l'utilisateur aura un ordinateur puissant. En effet, Windows 10, qui est le système d'exploitation principale du client lourd, peut rouler sur un ordinateur ayant 1GB de mémoire vive au minimum [1], ce qui est considéré comme étant très faible selon les standards modernes.

Une des limitation ayant un impact sur le design du système d'accès à la base de données est la taille limitée des documents dans Firestore, qui est de 1MB [2]. Il nous conviendrait donc de mettre toute structure complexe dans son propre document et de la référencer en utilisant son identifiant de document unique. Cela nous permettrait non seulement d'éviter de stocker des documents trop larges, mais aussi d'assurer une cohérence entre les données dans le cas où un objet complexe serait utilisé dans plusieurs endroits de la base de données. De plus, en testant notre base de données Firebase, nous voyons que les *reads* sont relativement rapides dans la plupart des cas (400-500ms en moyenne), mais qu'il est possible d'avoir des cas où les requêtes prennent 1 ou 2 secondes à répondre. Pour les *writes*, la latence est généralement beaucoup plus volatile, puisque cela dépend aussi de la complexité des données à insérer. Sans mentionner bien sur que cela dépend aussi de la qualité de la connexion à internet de l'utilisateur. Il faut donc s'assurer que l'application peut quand même bien fonctionner indépendamment du temps de réponse de la base de données.

Annexe

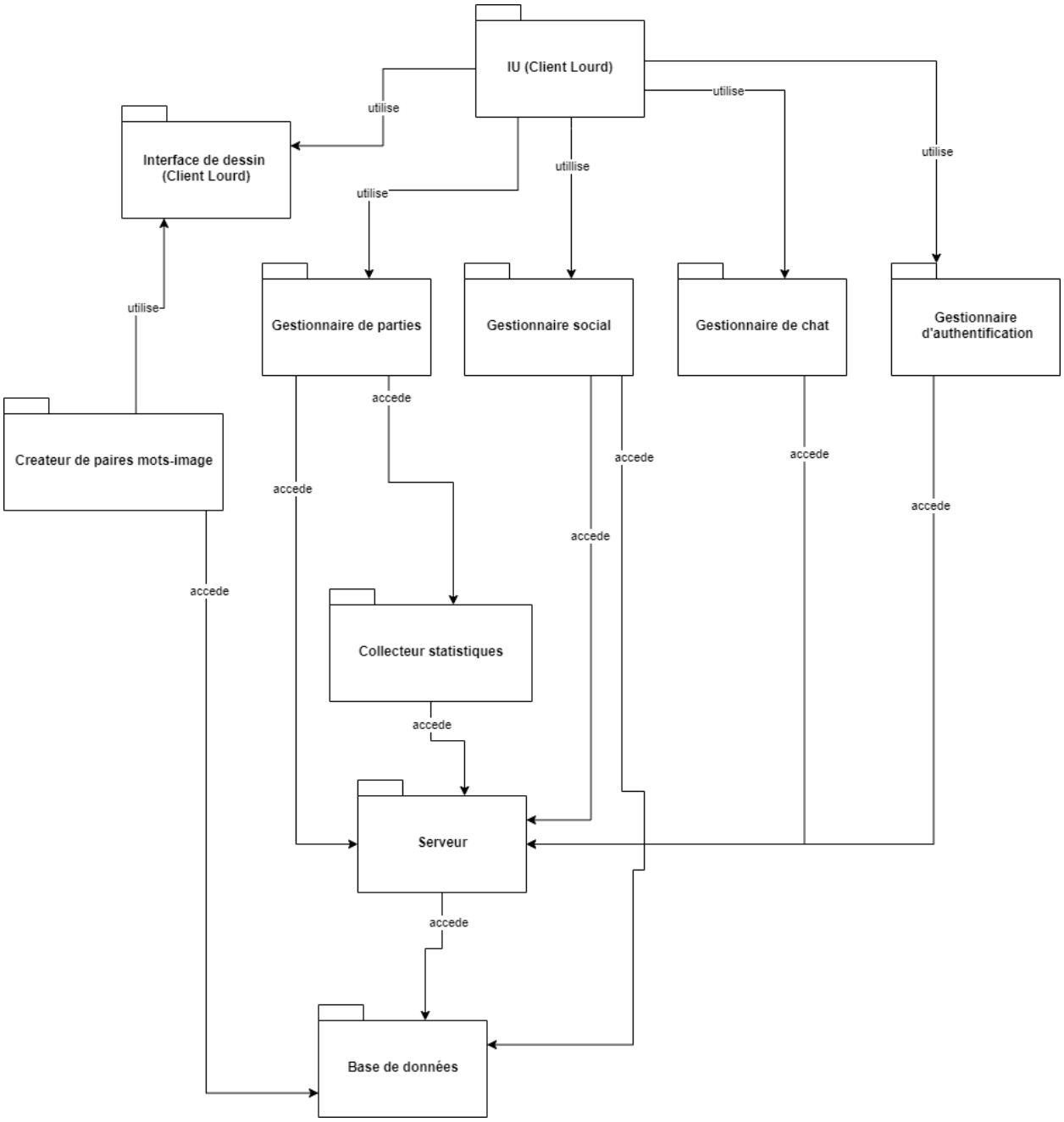

Figure A.1: Vue d'ensemble des paquetages de du client lourd de *Fais-moi un dessin* (Diagramme de paquetage)

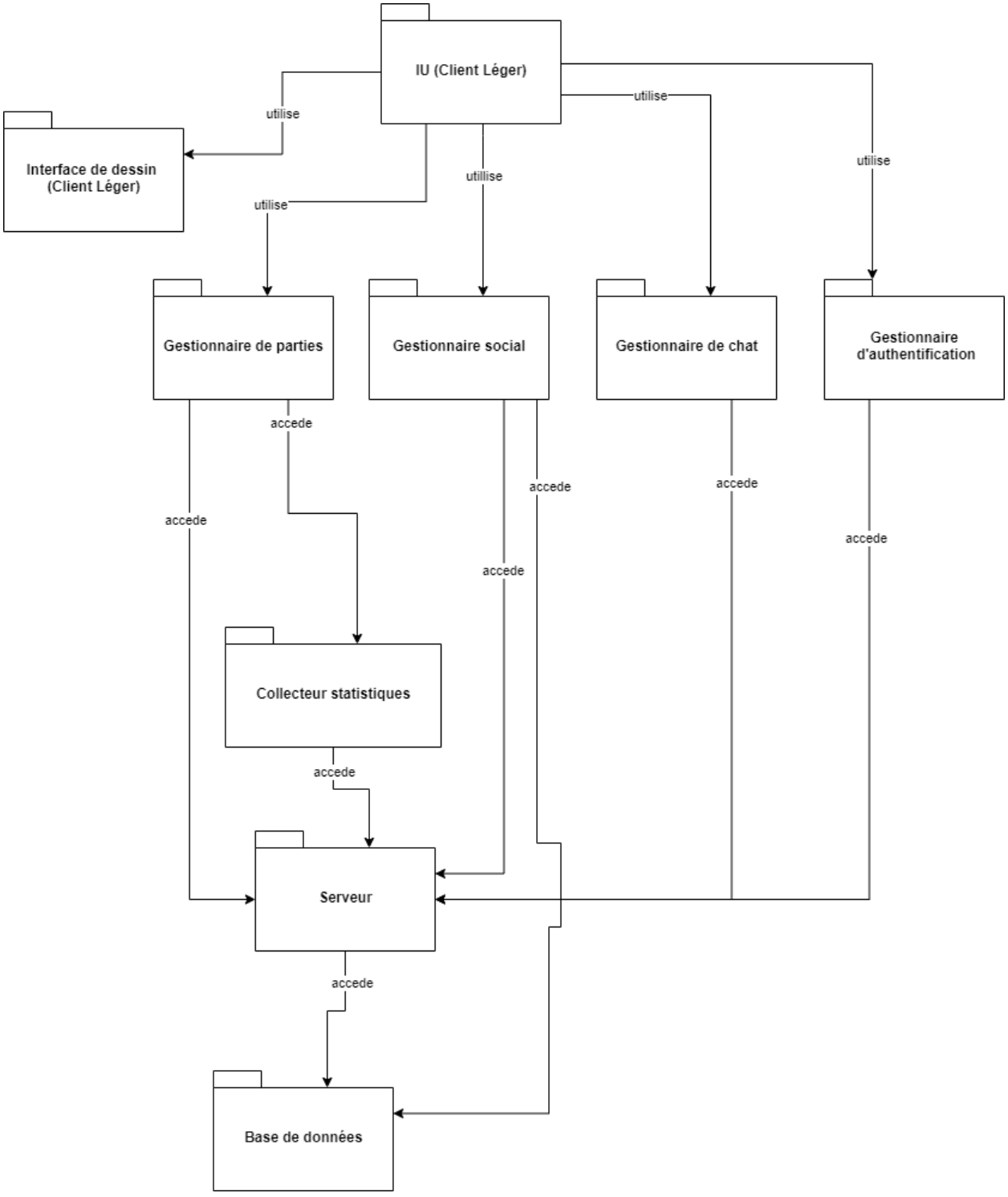

Figure A.2: Vue d'ensemble des paquetages de du client léger de *Fais-moi un dessin* (Diagramme de paquetage)

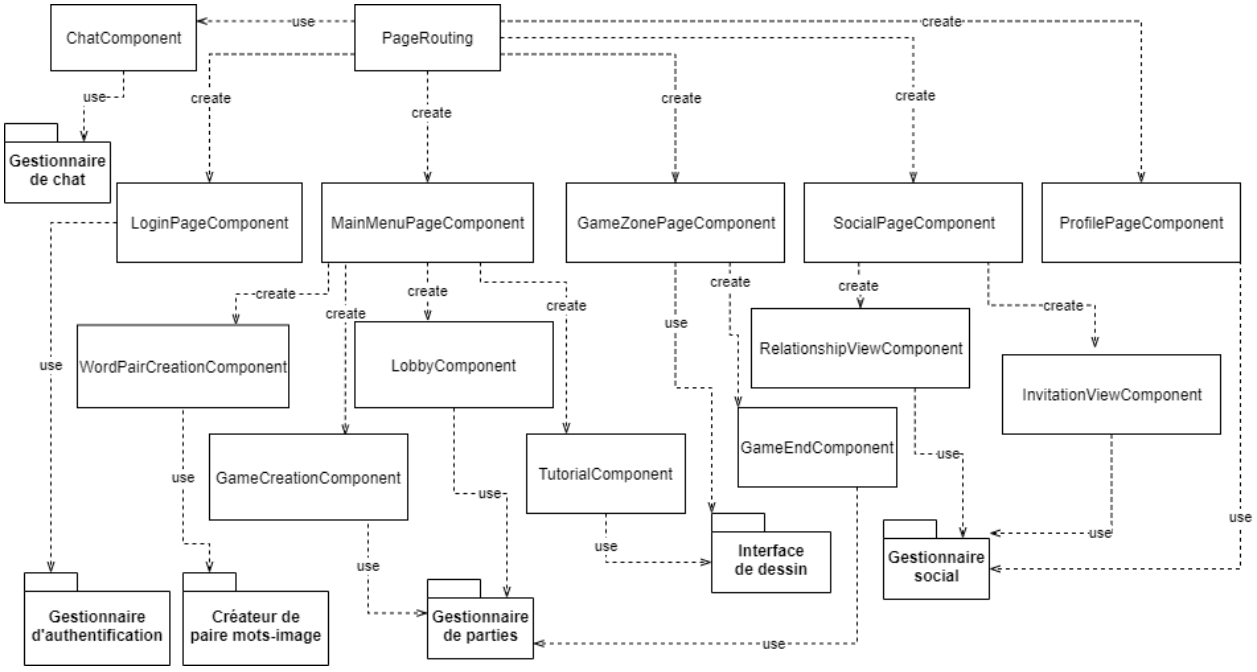

Figure A.3: Diagramme de classe de l'IU du client lourd

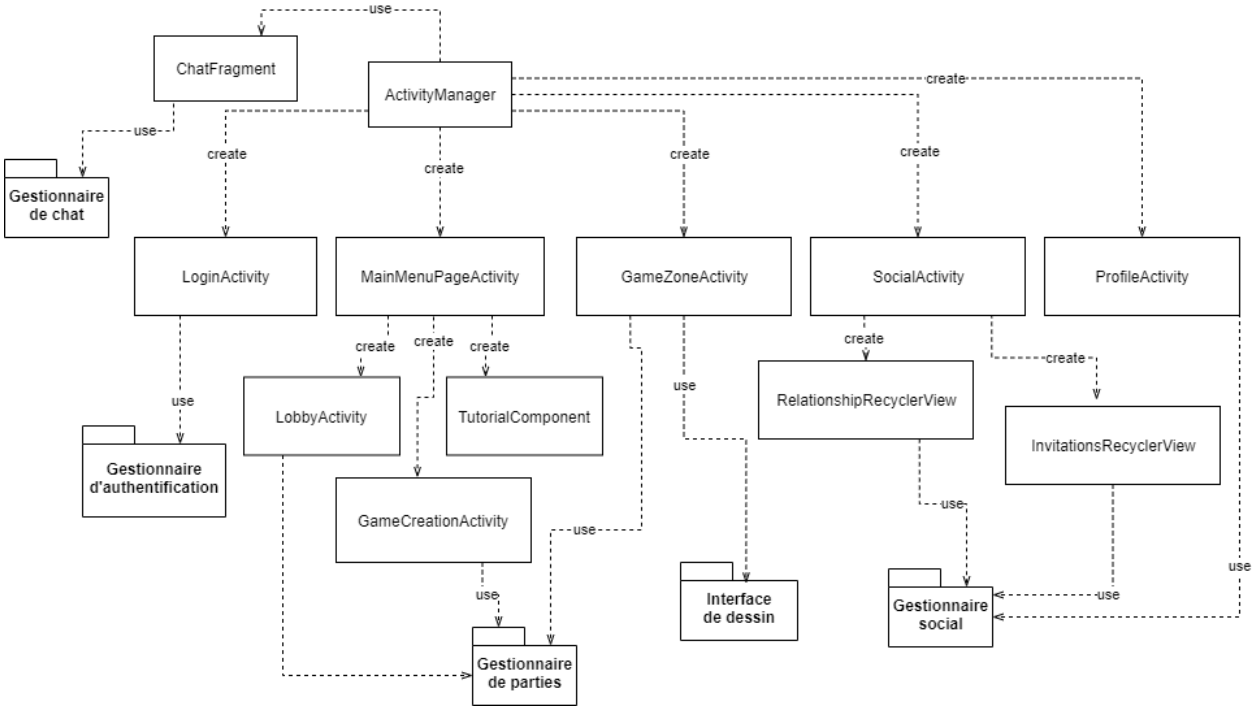

Figure A.4: Diagramme de classe de l'IU du client léger

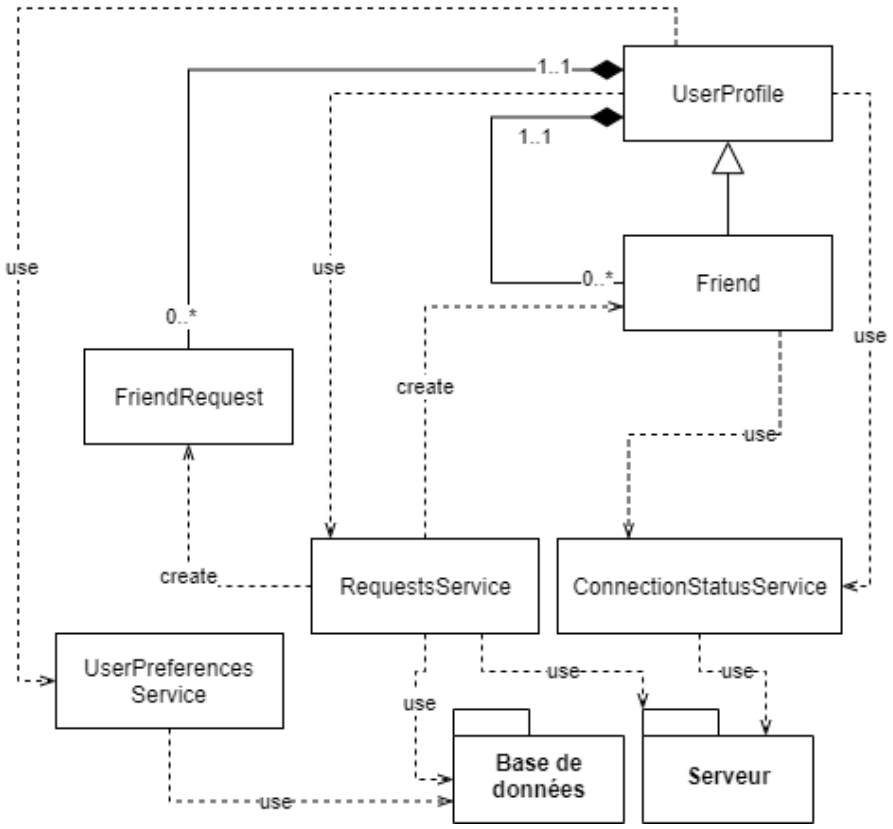

Figure A.5: Diagramme de classe du gestionnaire sociale

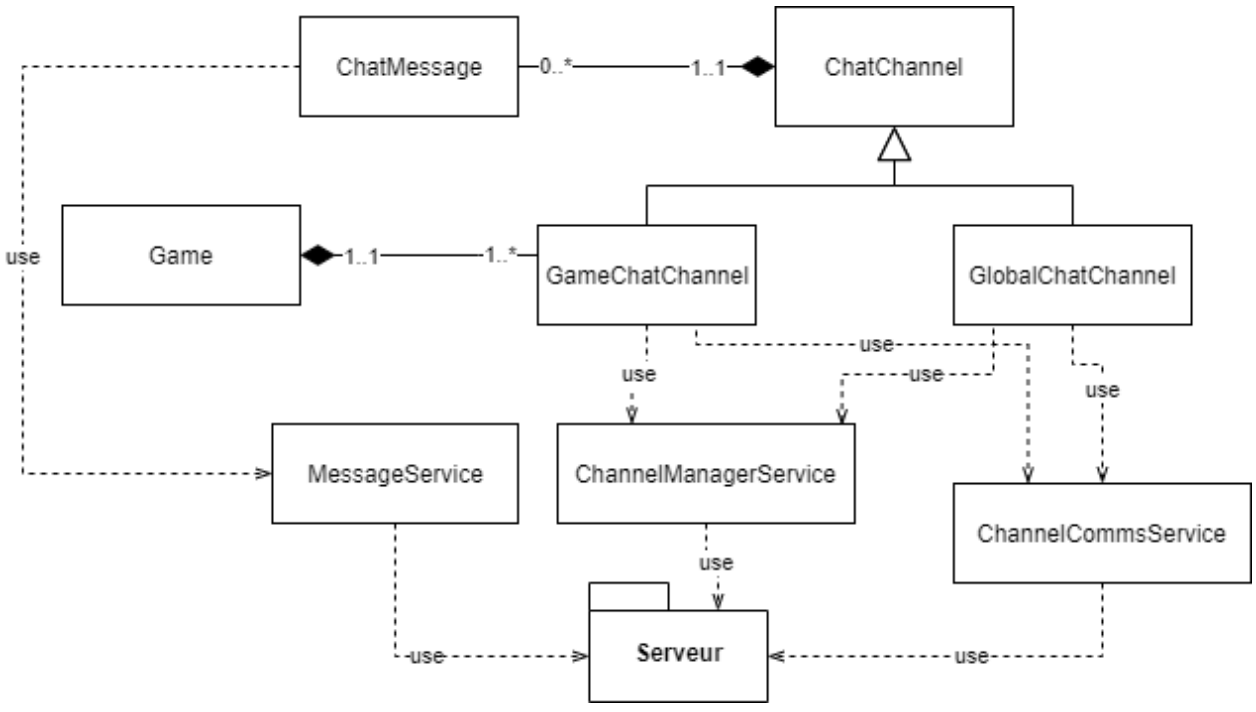

Figure A.6: Diagramme de classe du gestionnaire de chat

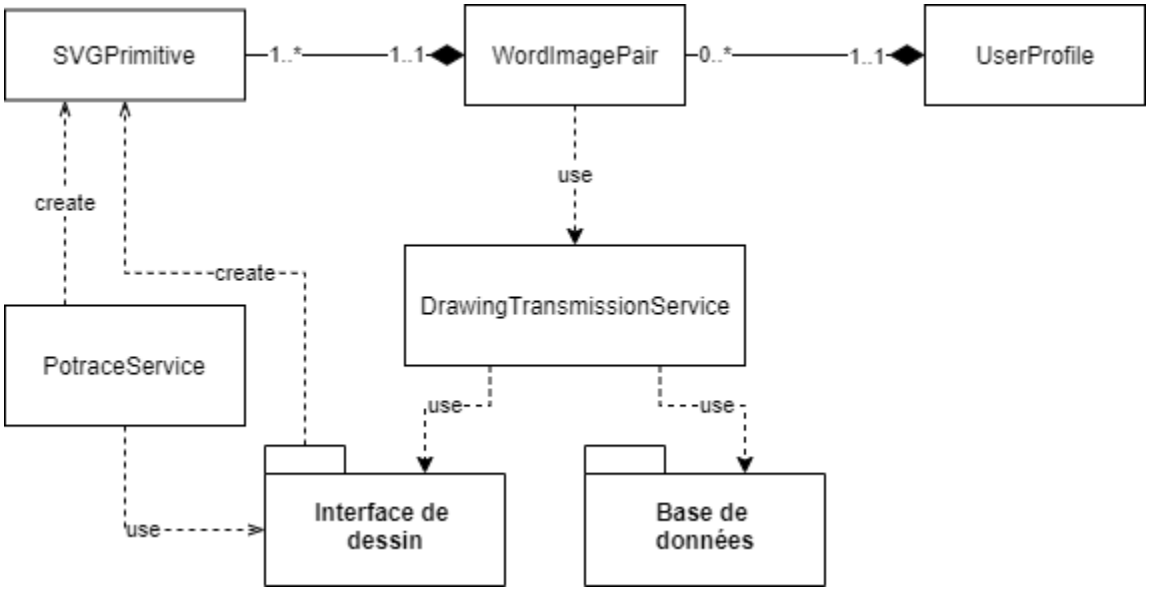

Figure A.7: Diagramme de classe du créateur de paires mot-image

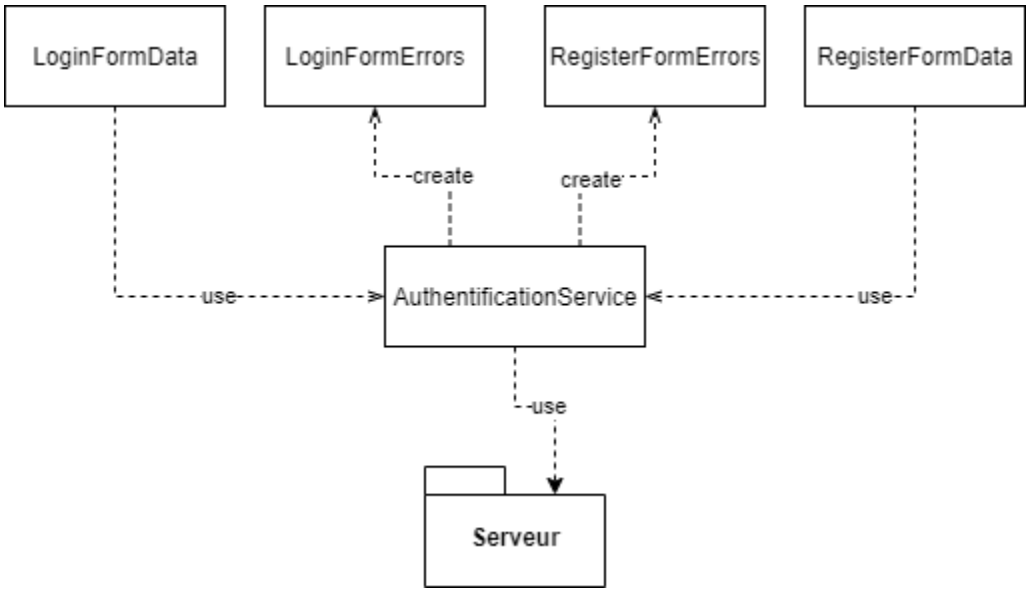

Figure A.8: Diagramme de classe du gestionnaire d'authentification

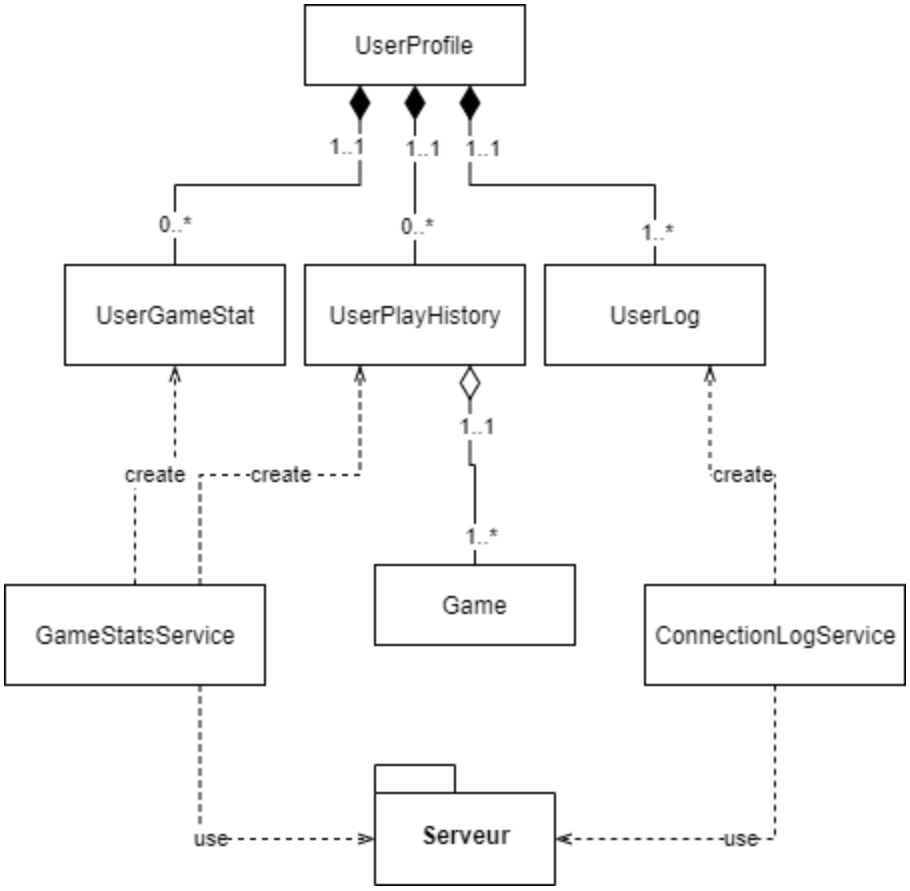

Figure A.9: Diagramme de classe du collecteur de statistiques

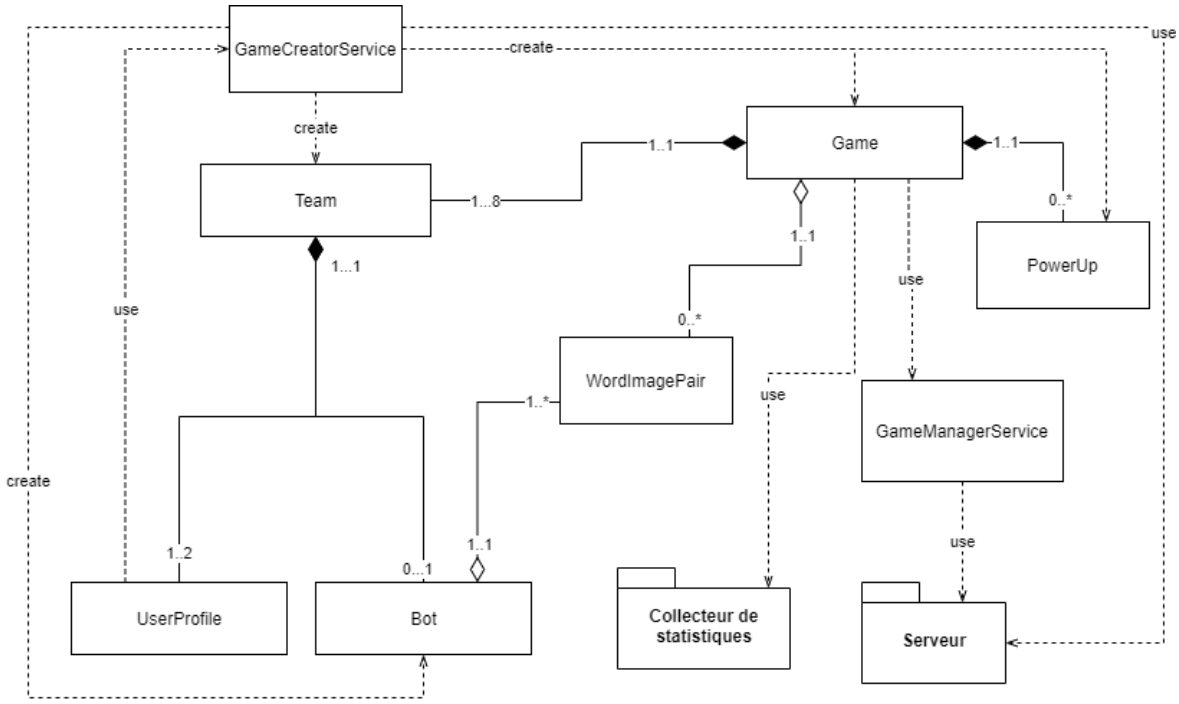

Figure A.10: Diagramme de classe du gestionnaire de partie

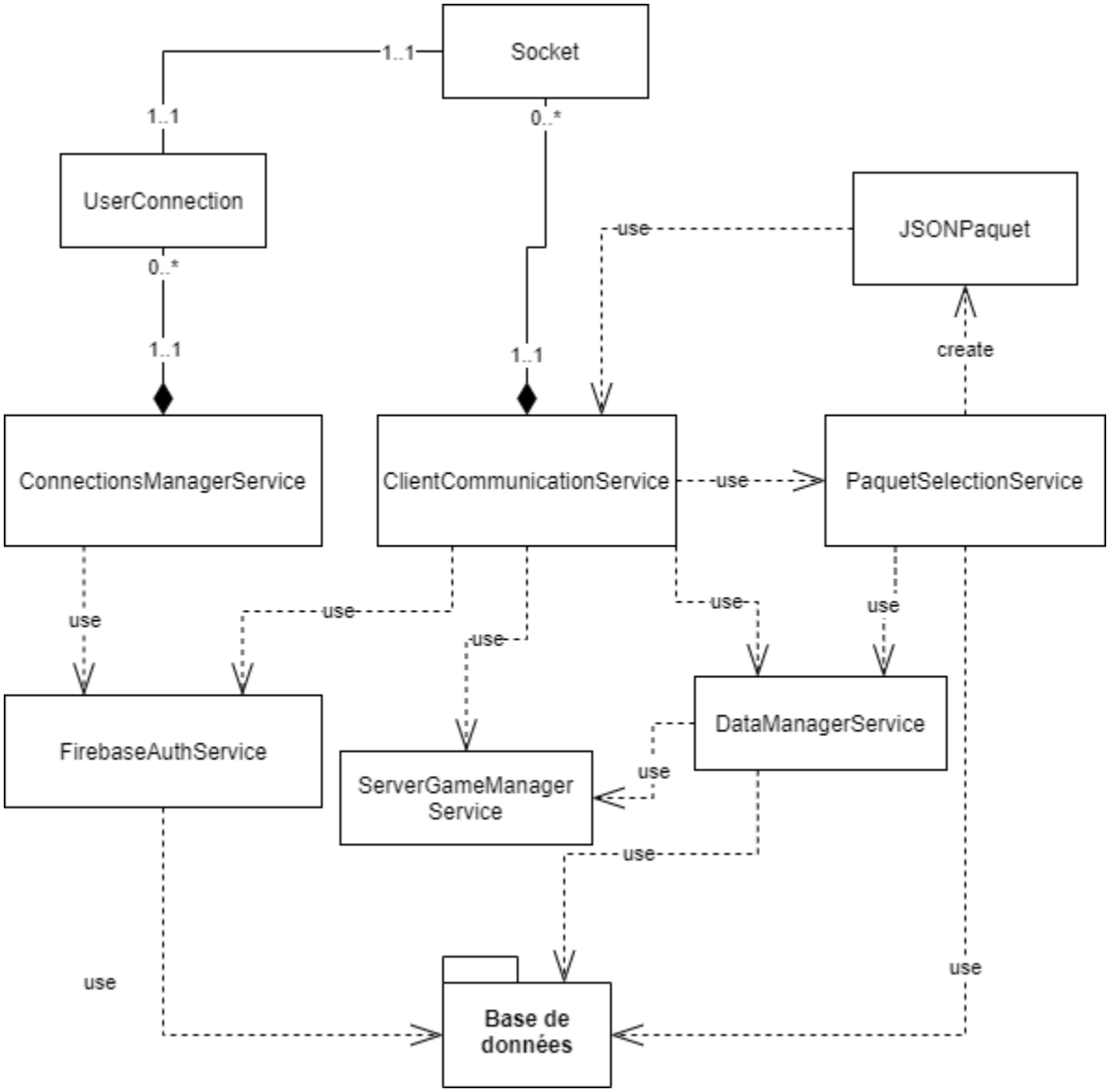

Figure A.11: Diagramme de classe du serveur

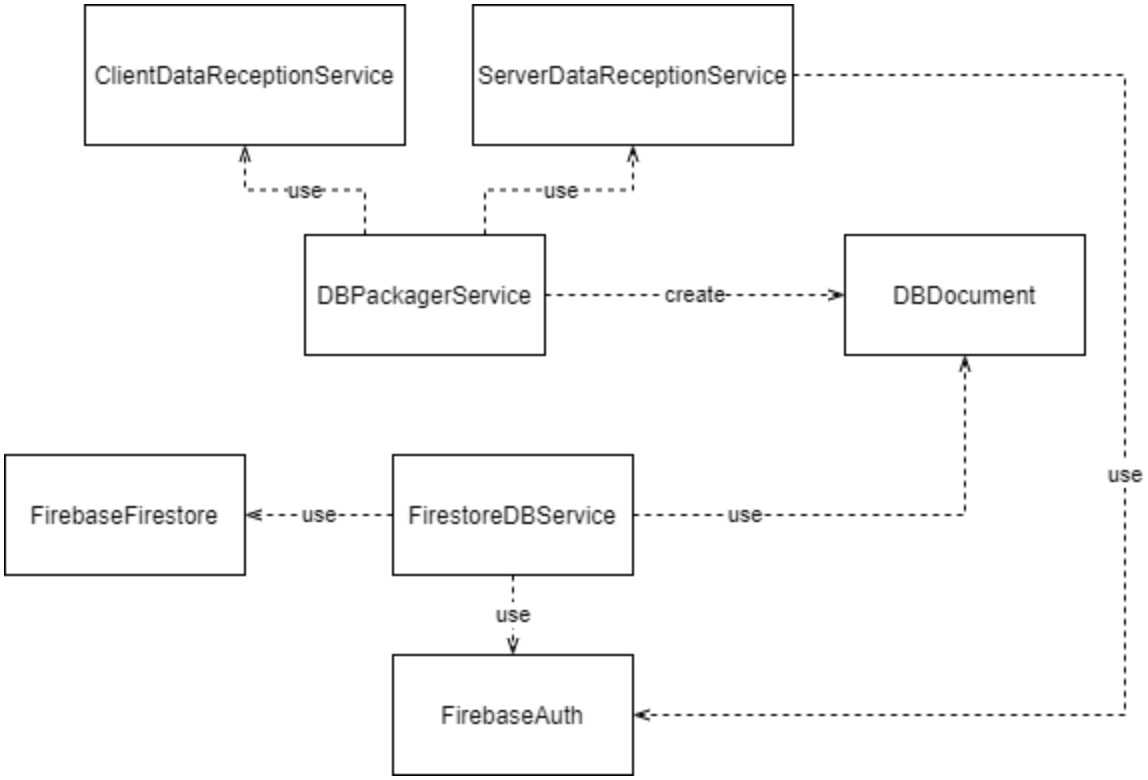

Figure A.12: Diagramme de classe pour la base de données

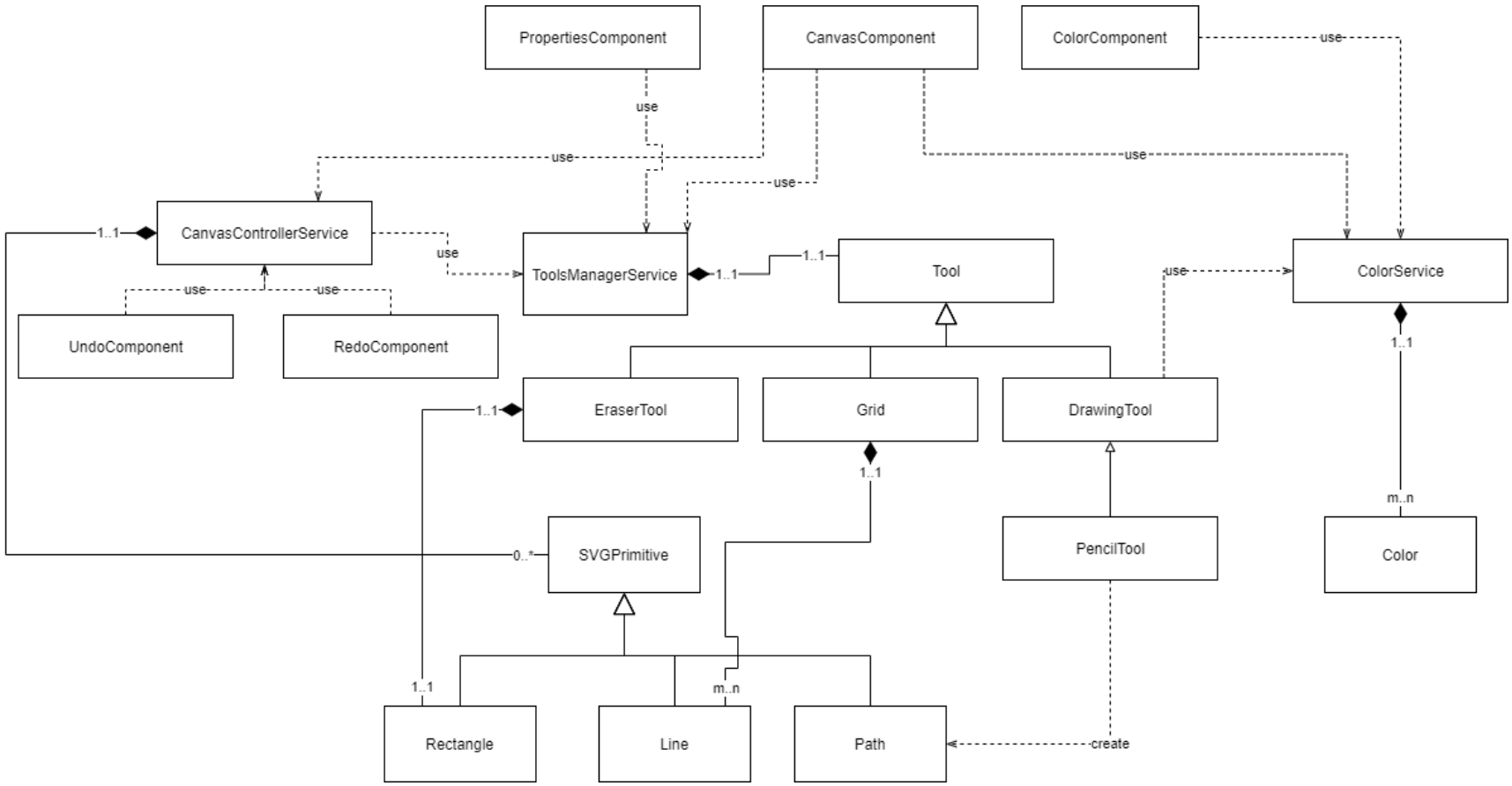

Figure A.13: Diagramme de classe de l'interface de dessin du client lourd

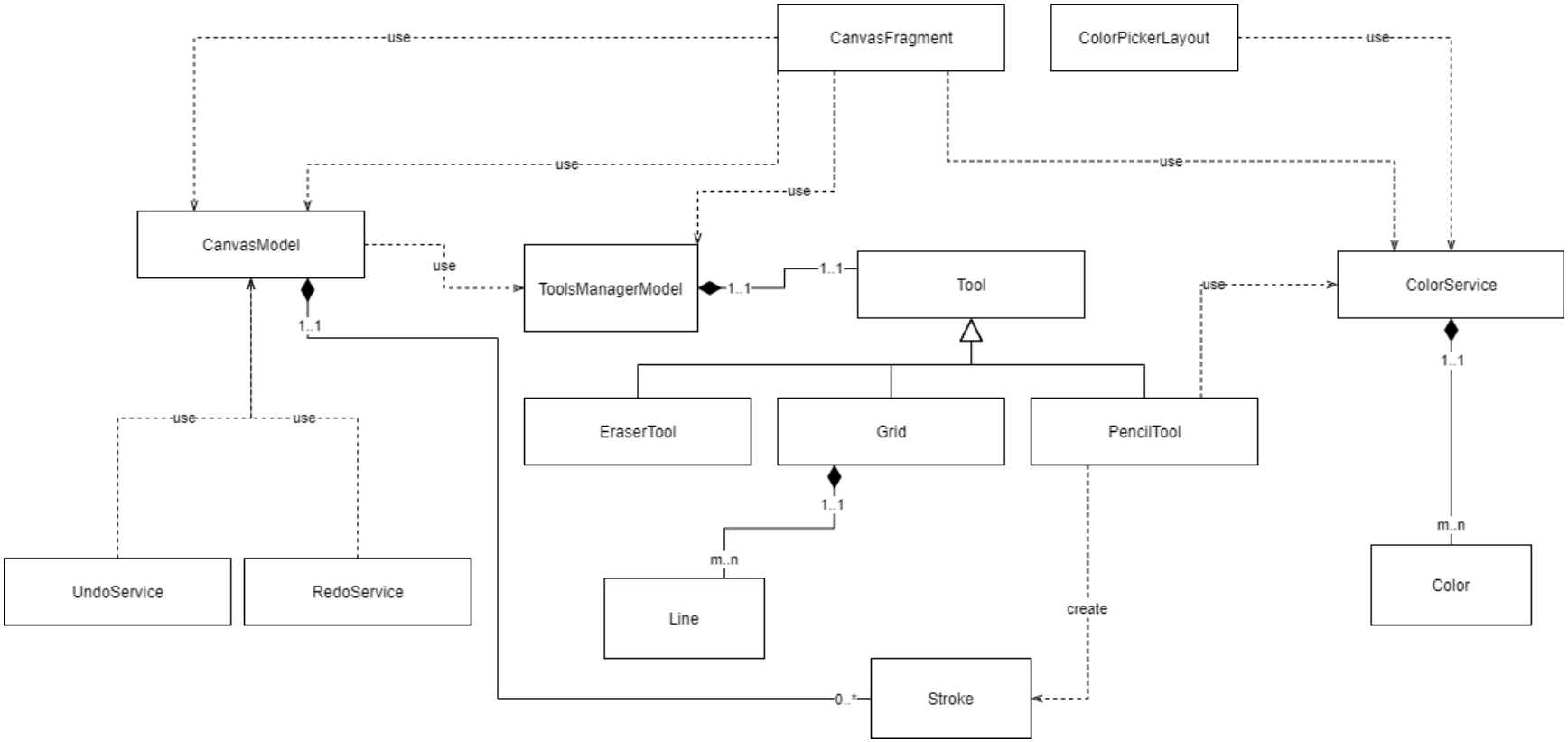

Figure A.14: Diagramme de classe de l'interface de dessin du client léger

Références

[1] Microsoft. (2021) Windows 10 system requirements. [En ligne]. Disponible: https://support.microsoft.com/en-us/windows/windows-10-system-requirements-6d4e9a79-66bf-7950-467c-795cf0386715

[2] Firebase. (2021) Usage and limits. [En ligne]. Disponible: https://firebase.google.com/docs/firestore/quotas