Équipe 6

Fais-moi un dessin
Document d'architecture logicielle

Version 3.0

Historique des révisions

Date	Version	Description	Auteur
2019-09-20	1.0	Rédaction de la partie 3,4,5,6.	Georges, Sami, Bassam, Syphax, Amine
2019-09-26	2.0	Rédaction de la partie 1,2,7	Georges, Sami, Bassam, Syphax, Amine
2019-11-27	3.0	Mise-à-jour de l'architecture logicielle	Équipe 6

Table des matières

Document d'architecture logicielle

1. Introduction

L’architecture logicielle représente les différentes composantes de notre logicielle et les types de relations qui les relient. On verra dans un premier lieu les objectifs et les contraintes architecturales, ensuite la vue des cas d’utilisation qui représente les aspects pertinents de notre logicielle, la vue logique qui représente les parties architecturalement significatives de notre modèle de design, la vue des processus qui décrit le système en termes d’interactions entre les différents processus significatifs, la vue de déploiement qui décrit les configurations de matériel physique et enfin la taille et performance qui présente une description des caractéristiques de taille et de performance pouvant avoir un impact sur l’architecture et le design de notre logiciel.

2. Objectifs et contraintes architecturaux

Les objectifs et les contraintes qui possèdent un impact architectural seront les suivants:

1. Architecture logicielle:

L’architecture logicielle de notre application se compose en plusieurs sous-systèmes:

1.1 Client lourd:

Notre client lourd est l’application qui sera utilisée sur un ordinateur sous l’environnement Windows 10. Cette application sera implémentée avec le langage de programmation C# et WPF. Cette application aura la possibilité de joindre des canaux pour clavarder, dessiner des dessins pour les faire deviner par un autre client ou bien deviner des dessins qui seront dessinés par un joueur virtuel.

1.2 Client léger:

Notre client léger est l’application qui sera utilisée sur une tablette. Cette application aura une interface tactile à la place d’une souris. Cette dernière sera implémentée en langage Java et aura presque toutes les fonctionnalités qui existent dans le client léger.

1.3 Serveur:

Notre serveur est implémenté en JavaScript sur NodeJS. Celui-ci gère toutes les communications entre notre client lourd et notre client léger. De plus, il sera responsable de gérer notre base de données, ainsi qu'à vérifier les authentifications des utilisateurs.

2. Sécurité :

Notre application offre aux utilisateurs la sécurité, puisqu’on ne sauvegarde pas le mot de passe de l’utilisateur dans une variable local. De plus, le mot de passe est encrypté par le serveur.

3. Performance:

Le temps de réponse entre le temps qu’un message soit transmis et sa réception est petite, ainsi pour le temps de réponse entre le temps qu’une image est dessinée et affichée.

3. Vue des cas d'utilisation

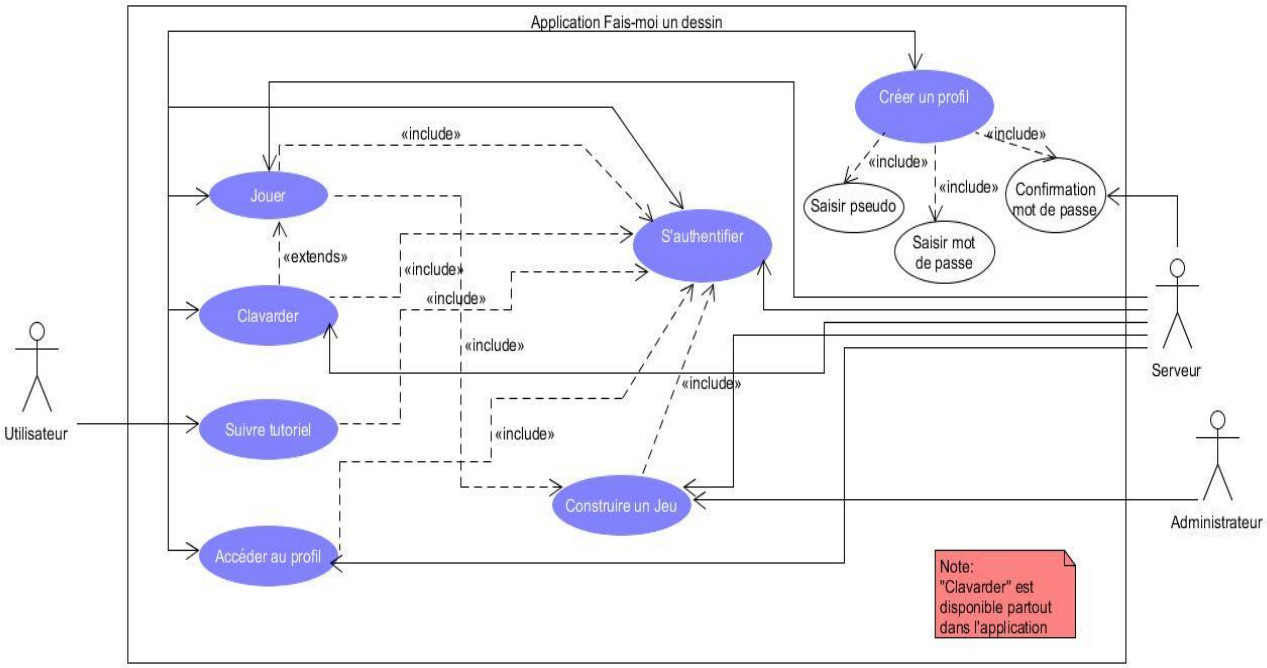

Fig. 1: Diagramme de cas d'utilisation de l'application «Fais-moi un dessin »

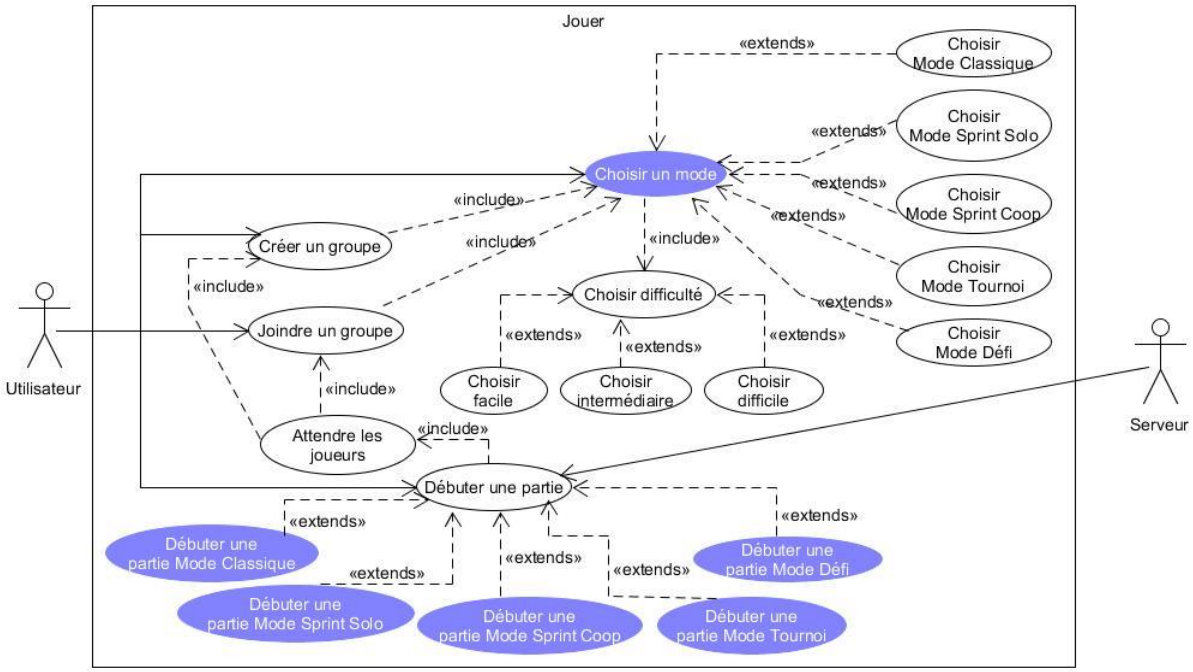

Fig. 2: Diagramme de cas d'utilisation « Jouer »

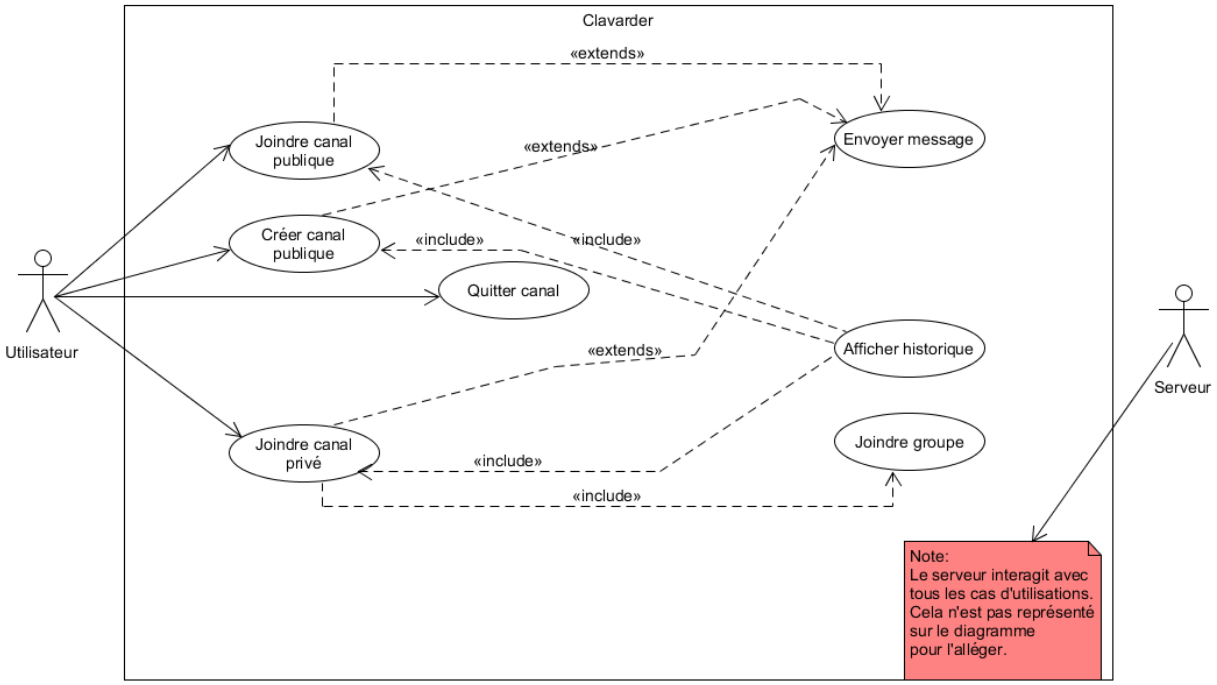

Fig. 3: Diagramme de cas d'utilisation « Clavarder »

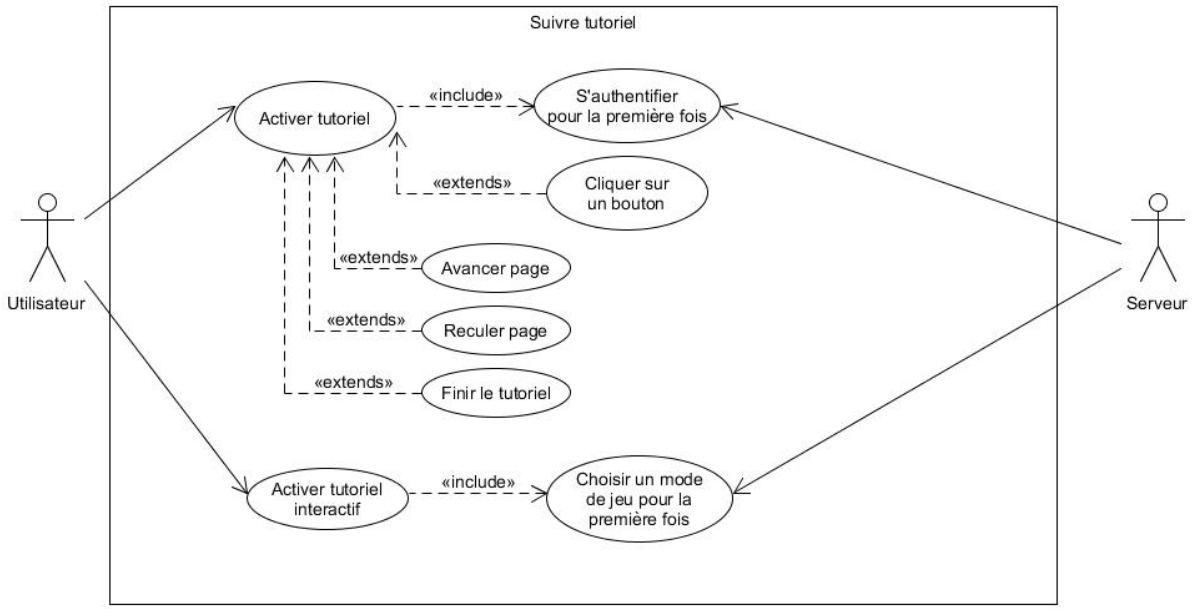

Fig. 4: Diagramme de cas d'utilisation « Suivre tutoriel »

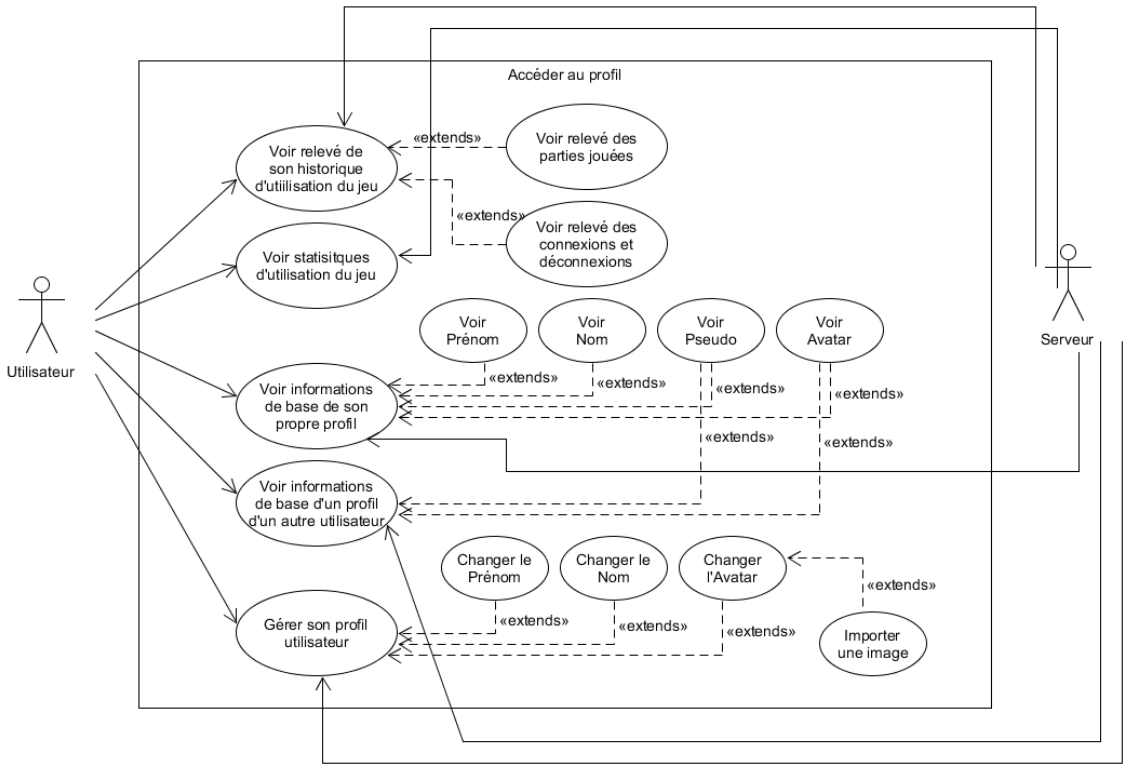

Fig. 5: Diagramme de cas d'utilisation « Accéder au profil »

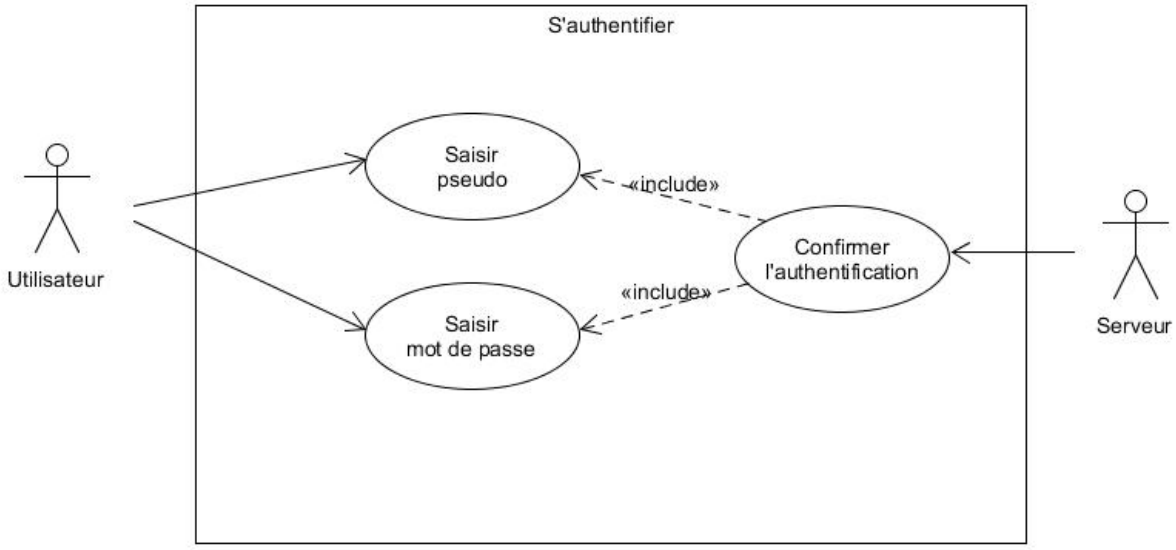

Fig. 6: Diagramme de cas d'utilisation « S'authentifier »

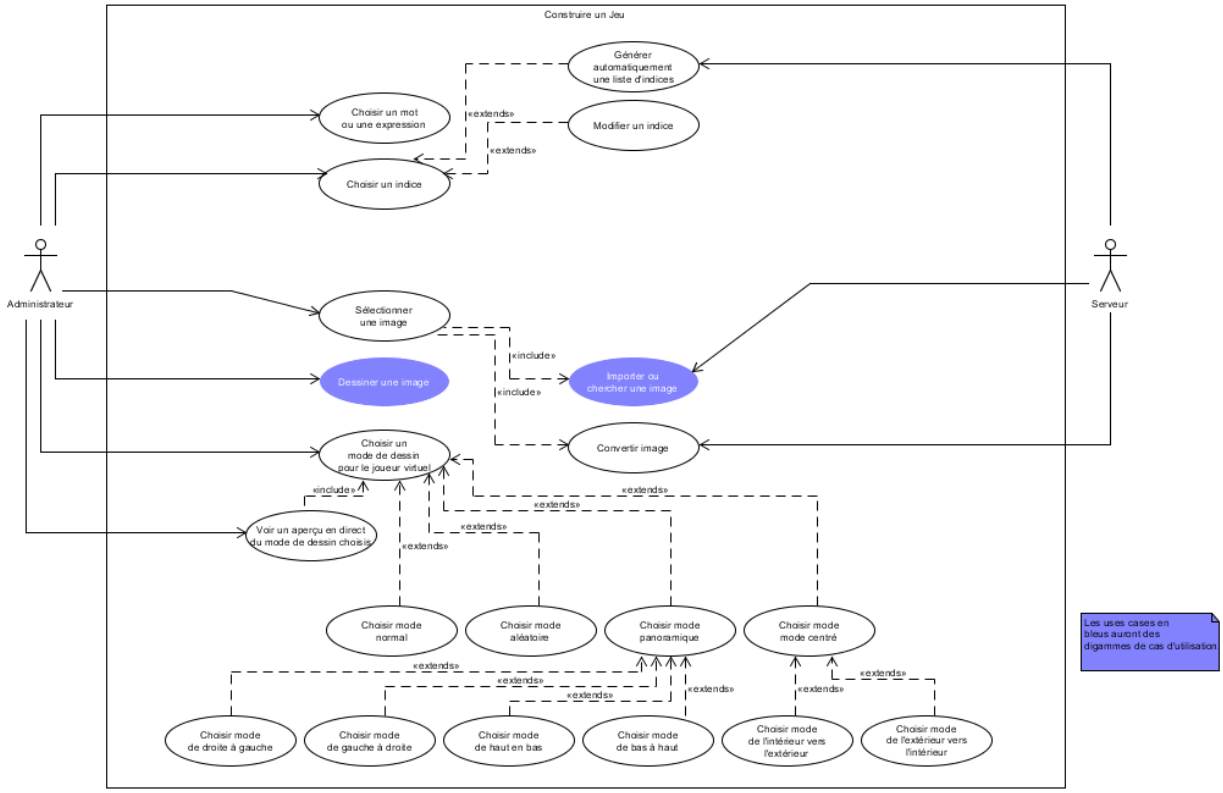

Fig. 7: Diagramme de cas d'utilisation « Construire un jeu »

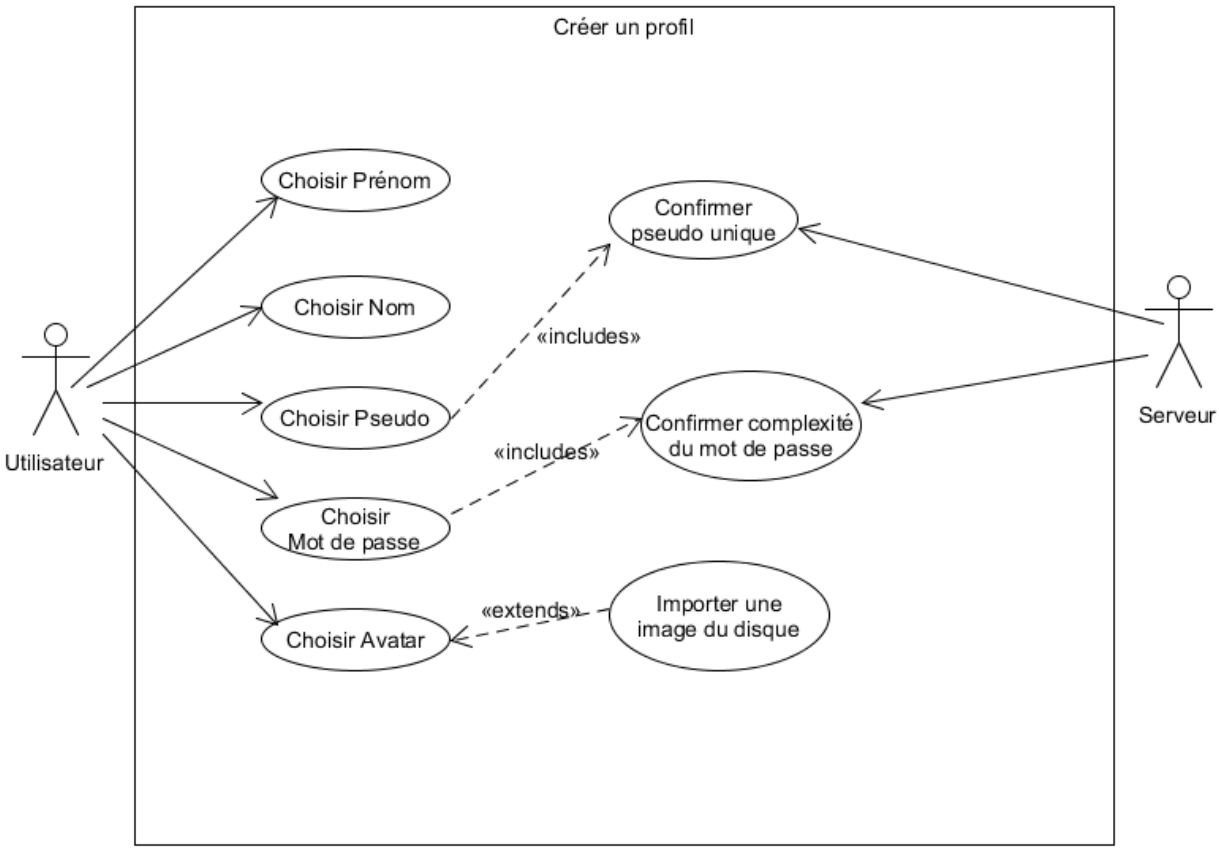

Fig. 8: Diagramme de cas d'utilisation « Créer un profil »

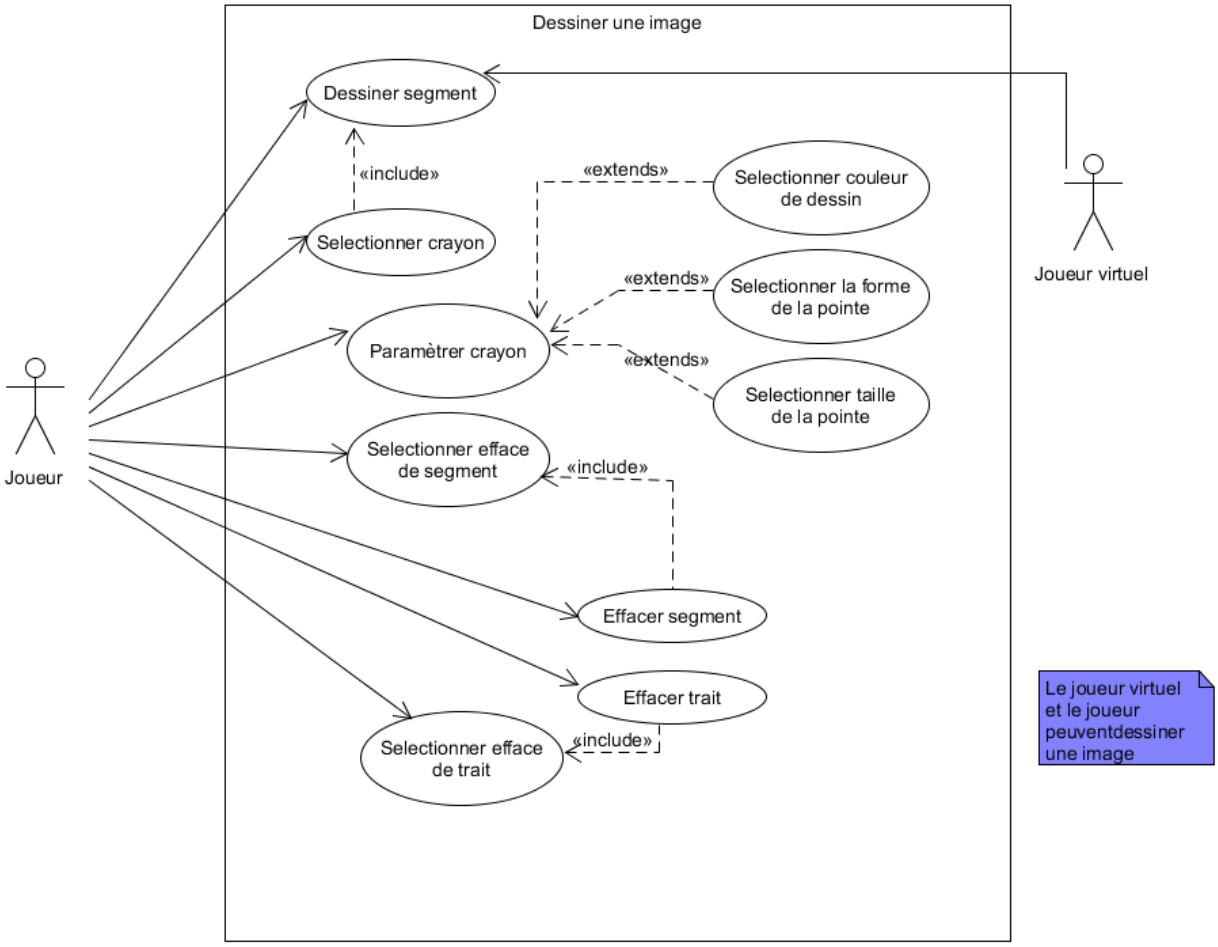

Fig. 9: Diagramme de cas d'utilisation « Dessiner une image »

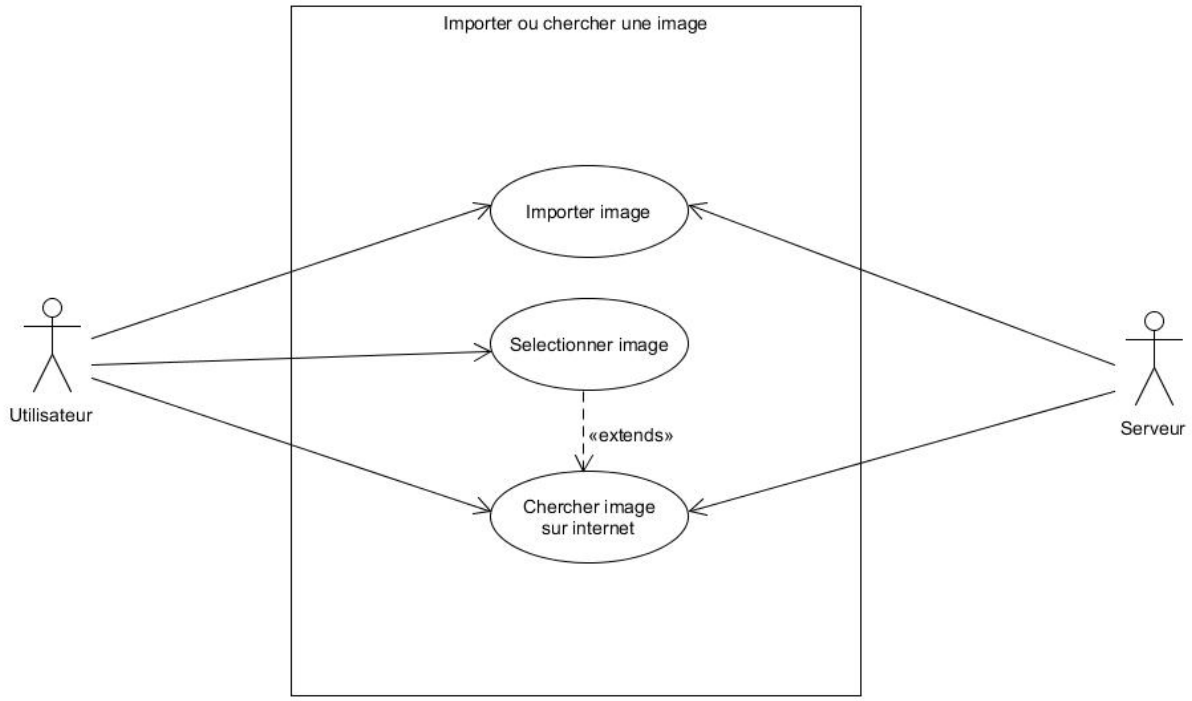

Fig. 10: Diagramme de cas d'utilisation « Importer ou chercher une image »

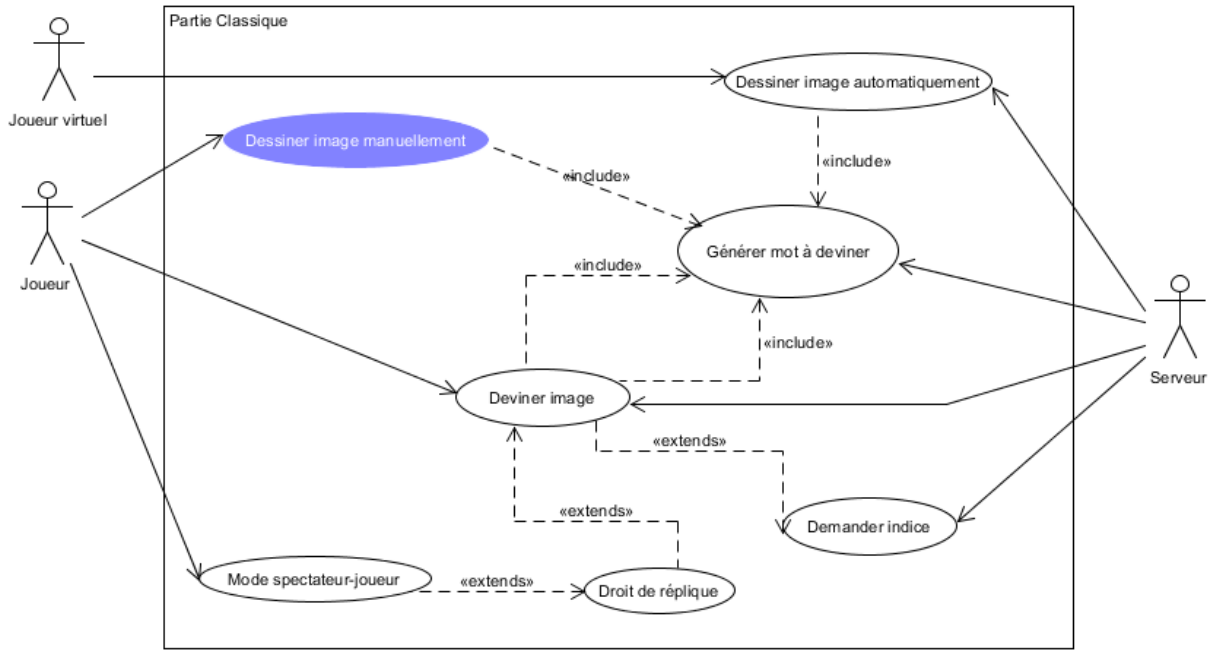

Fig. 11: Diagramme de cas d'utilisation « Jouer à une partie Classique »

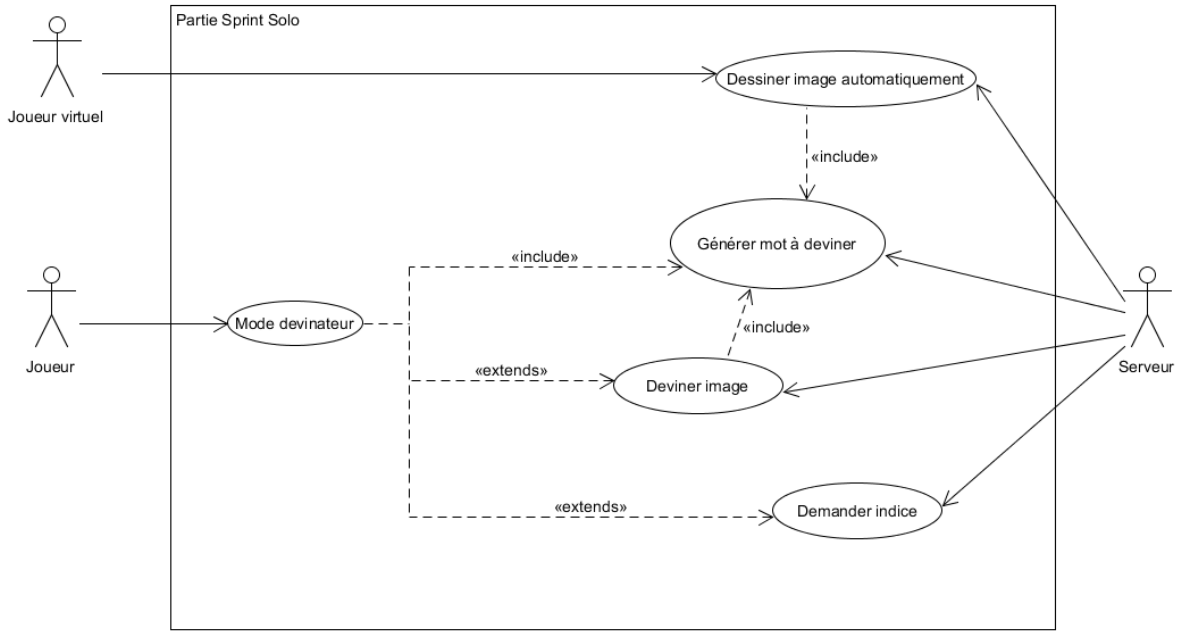

Fig. 12: Diagramme de cas d'utilisation « Jouer à une partie Sprint Solo »

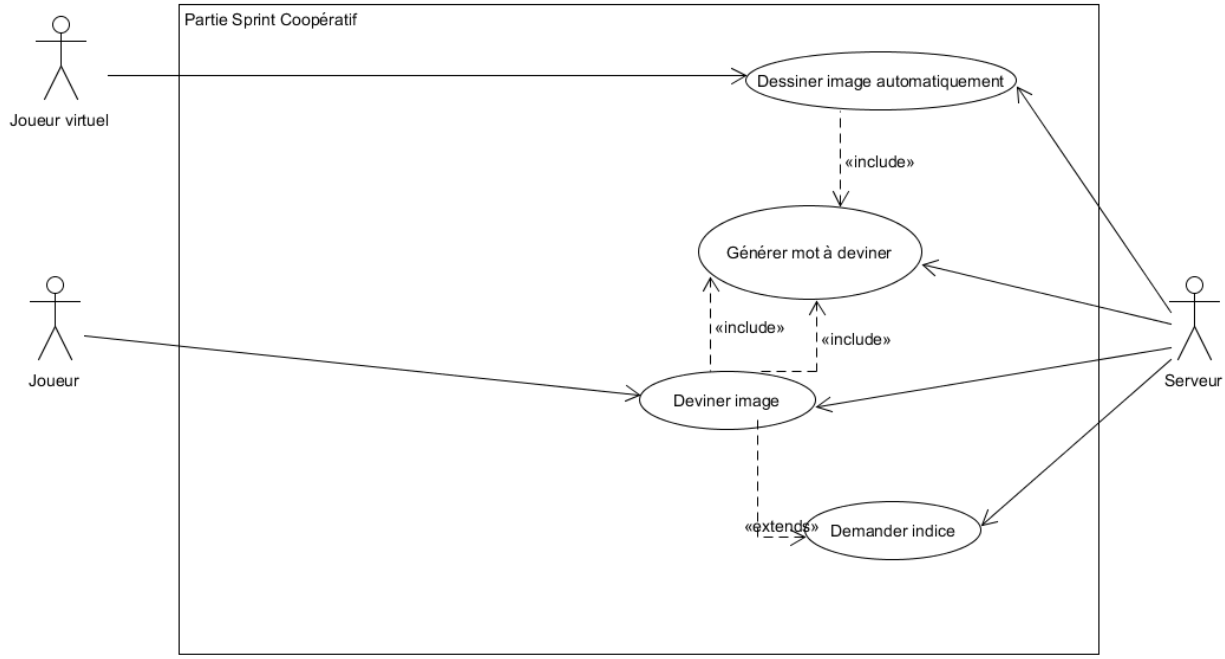

Fig. 13: Diagramme de cas d'utilisation « Jouer à une partie Sprint Coopératif »

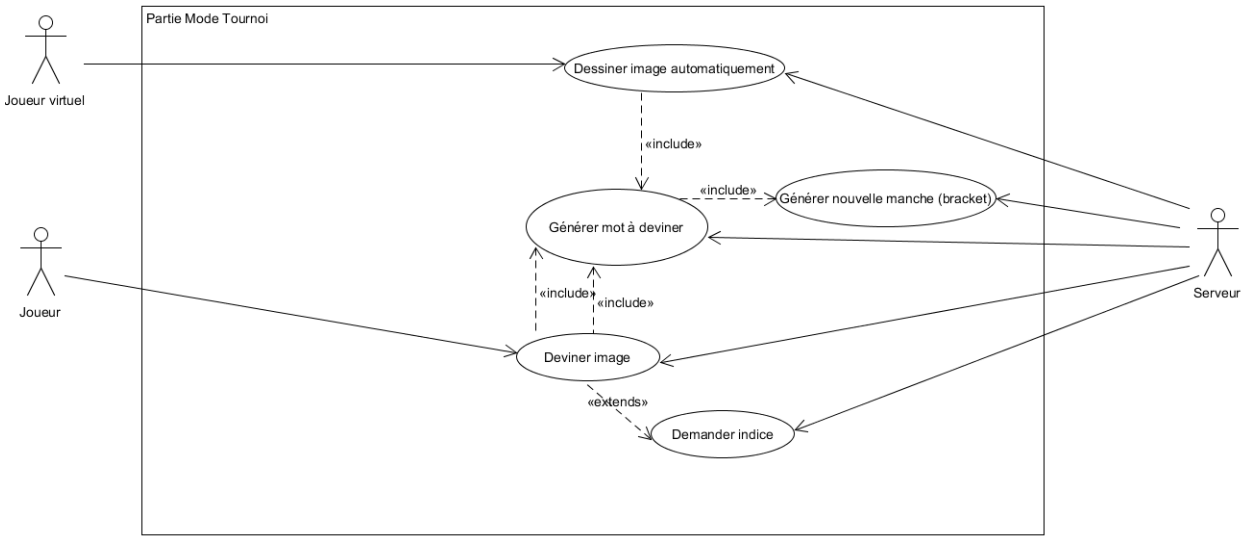

Fig. 14: Diagramme de cas d'utilisation « Jouer à une partie Tournoi »

4. Vue logique

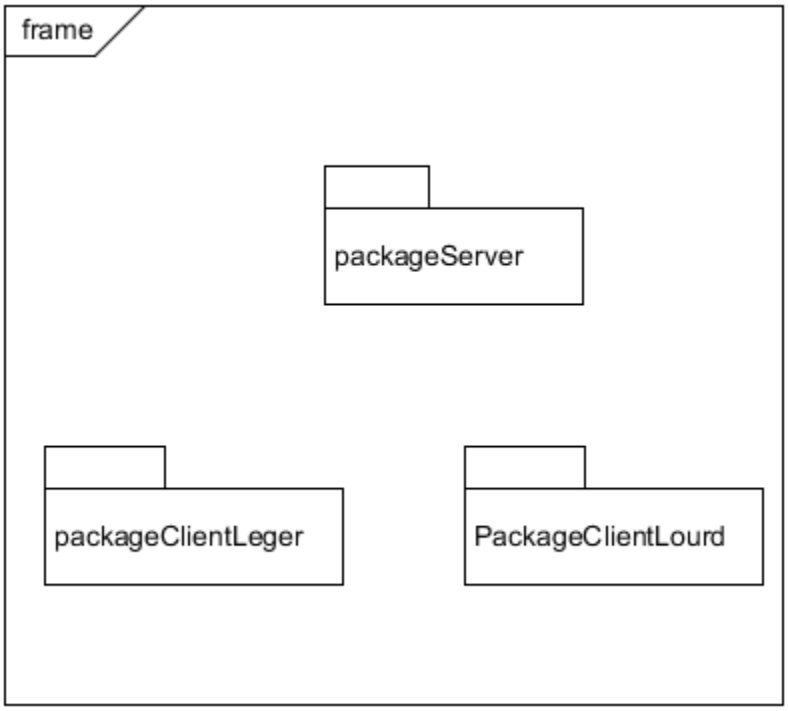

Fig. 15: Diagramme de paquetage global

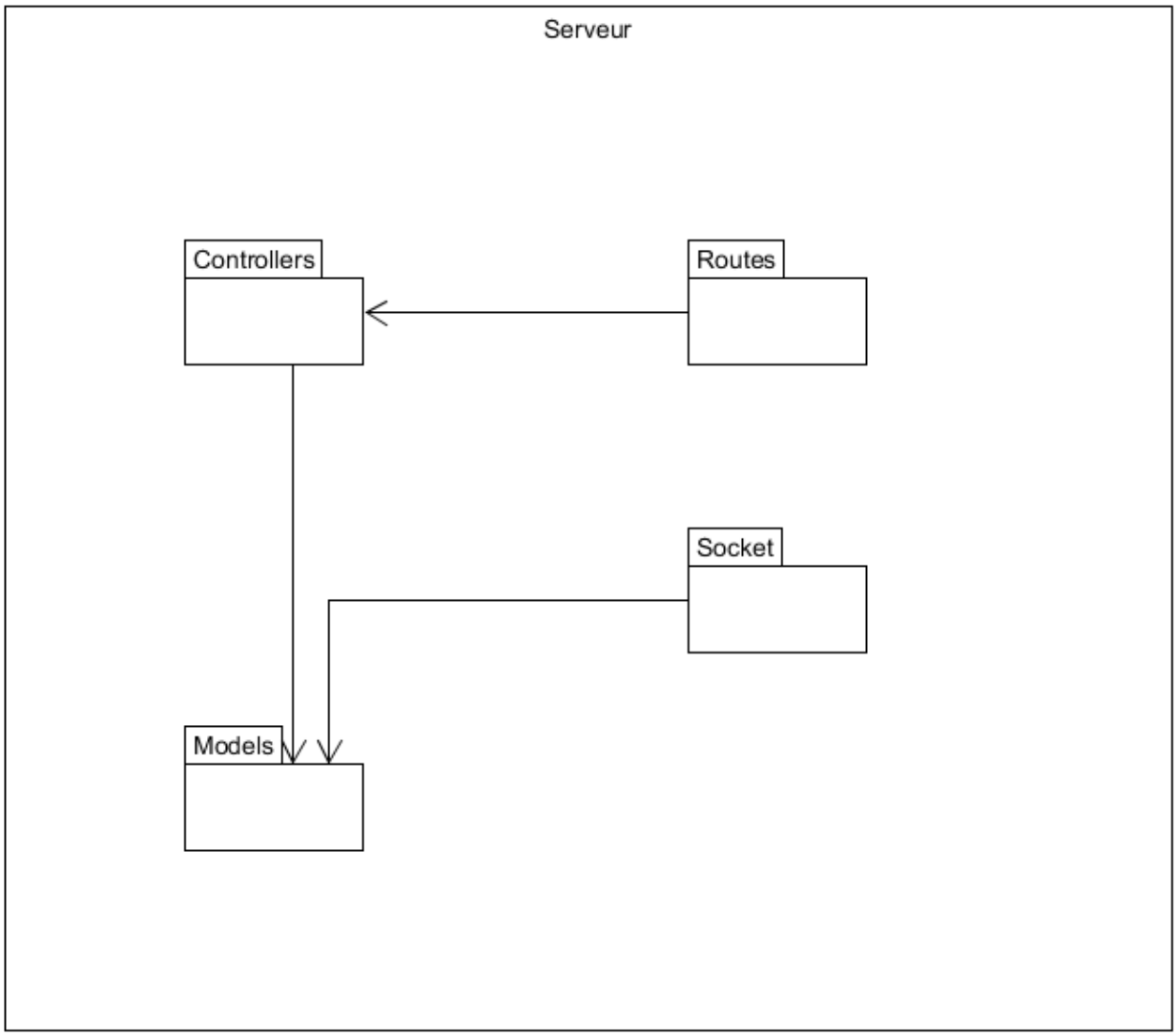

Fig. 16: Diagramme de paquetage « Serveur »

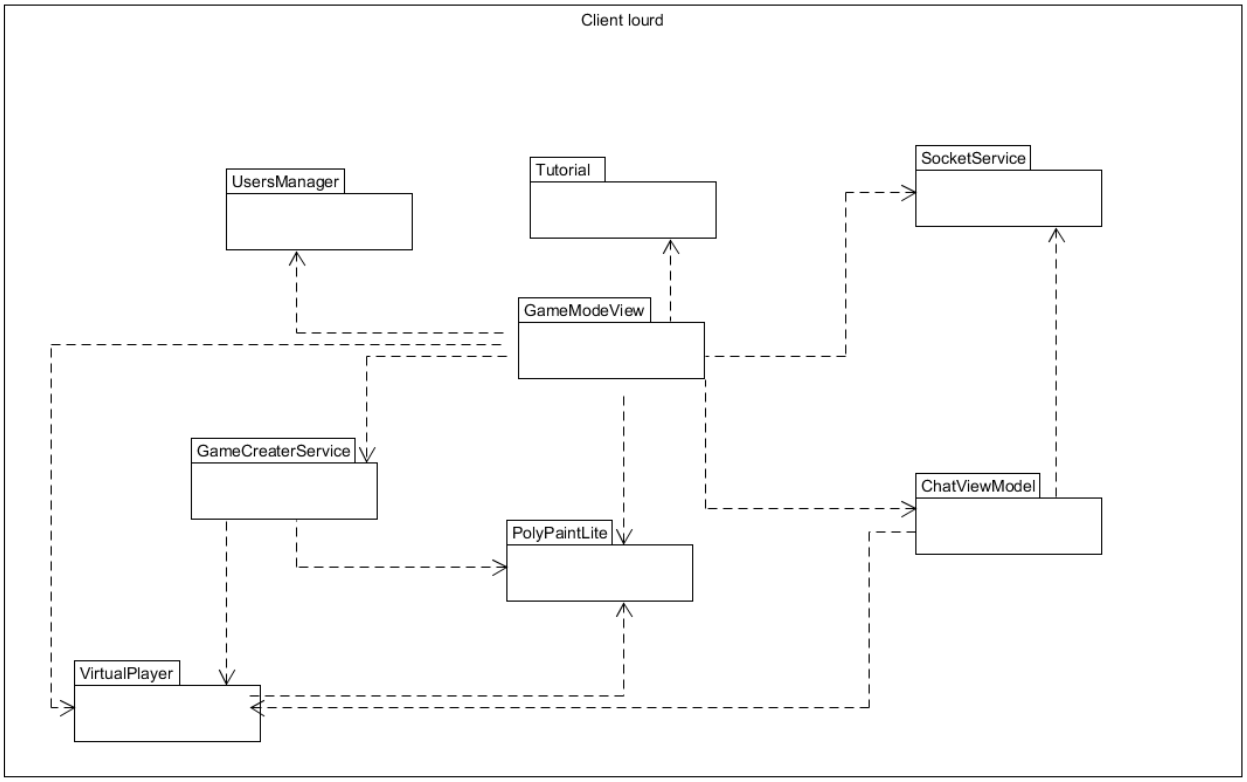

Fig. 17: Diagramme de paquetage « Client lourd »

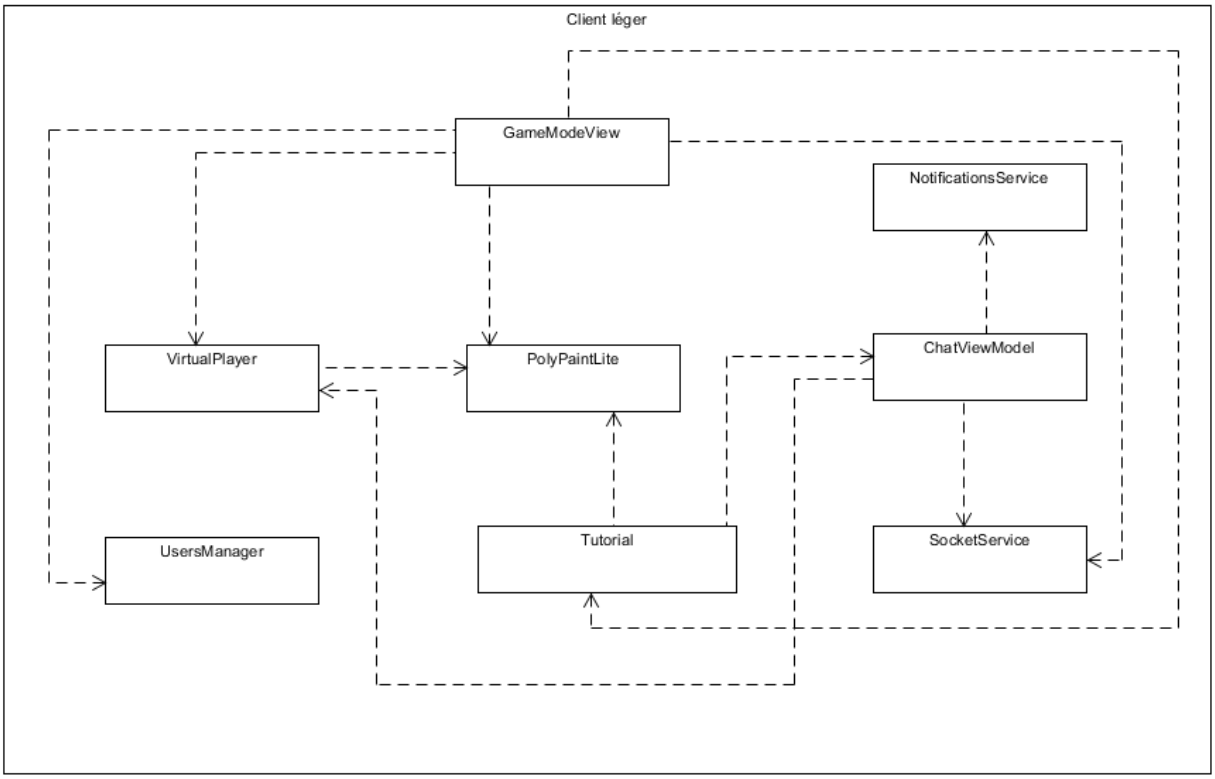

Fig. 18: Diagramme de paquetage « Client léger »

Routes
Dans ce paquetage, on définit les modèles URL pour que le moteur de routage puisse déterminer la classe ou la fonction appropriée pour une demande entrante.

Controllers
Dans ce paquetage, on trouve nos contrôleurs. Ces derniers sont chargés de contrôler la manière dont l'utilisateur interagit avec notre application. Les contrôleurs déterminent la réponse à renvoyer à un utilisateur lorsqu'il fait une demande.

Socket
Le socket nous permet de faire des communications avec notre serveur. Fondamentalement, il s'agit d'une configuration unidirectionnelle client et serveur dans laquelle un client se connecte, envoie des messages au serveur et le serveur les affiche via une connexion socket.

Models
Dans ce paquetage, on trouvera les modèles qui représentent des données spécifiques à un domaine. Ces modèles maintiennent les données de l'application.

UsersManager
Dans ce paquetage, on définit les utilisateurs de l'application. Leur historique de jeu, leurs statistiques, ainsi que toutes les informations reliées à leur compte vont être gérés par ce paquet.

Tutorial
Dans ce paquetage, on y retrouve le tutorial interactif auquel l'utilisateur peut participer afin de comprendre les différents modes de jeu que l'on retrouve dans le paquetage GameModeView.

GameModeView
Dans ce paquetage, on y retrouve les différents modes de jeu auquel l'utilisateur peut jouer. Ce paquetage contient le déroulement d'une partie. Il va utiliser PolyPaintLite, ChatViewModel, SocketService, Tutorial, UserManager, VirtualPlayer, ainsi que GameCreaterService pour engendrer une partie multijoueur.

SocketService
Ce paquetage, comme le paquetage Socket, permet de faire la communication entre le serveur et le client en temps réel. Permet de progresser la partie multijoueur et de clavarder.

GameCreaterService
Dans ce paquetage, on y retrouve le nécessaire pour créer un jeu qui contiendra un mot (ou expression), ainsi que les informations associées à ce mot (ou expression) tel que le dessin et les indices. Il utilisera PolyPaintLite et VirtualPlayer pour gérer le mode de création du jeu.

VirtualPlayer
Dans ce paquetage, on y retrouve la logique nécessaire pour dessiner automatiquement un dessin sauvegardé, trait par trait. VirtualPlayer sera utilisé par GameViewMode et GameCreaterService.

PolyPaintLite
Dans ce paquetage, il contient la logique nécessaire pour permettre à l'utilisateur d'avoir les outils nécessaires afin de dessiner manuellement un dessin. PolyPaintLite sera utilisé dans GameCreaterService, ainsi que le GameViewMode.

ChatViewModel
Dans ce paquetage, il contient la logique nécessaire afin de pouvoir clavarder avec d'autres utilisateurs. Il est utilisé par GameViewMode, car lors d'une partie on doit aussi pouvoir clavarder entre les joueurs de cette partie. De plus, il gère plusieurs canaux de discussions.

NotificationsService
Dans ce paquetage, spécifiquement à Android, il gère les notifications envoyées à l'utilisateur lorsque celui-ci reçoit un message dans un canal de discussion auquel il est abonné.

5. Vue des processus

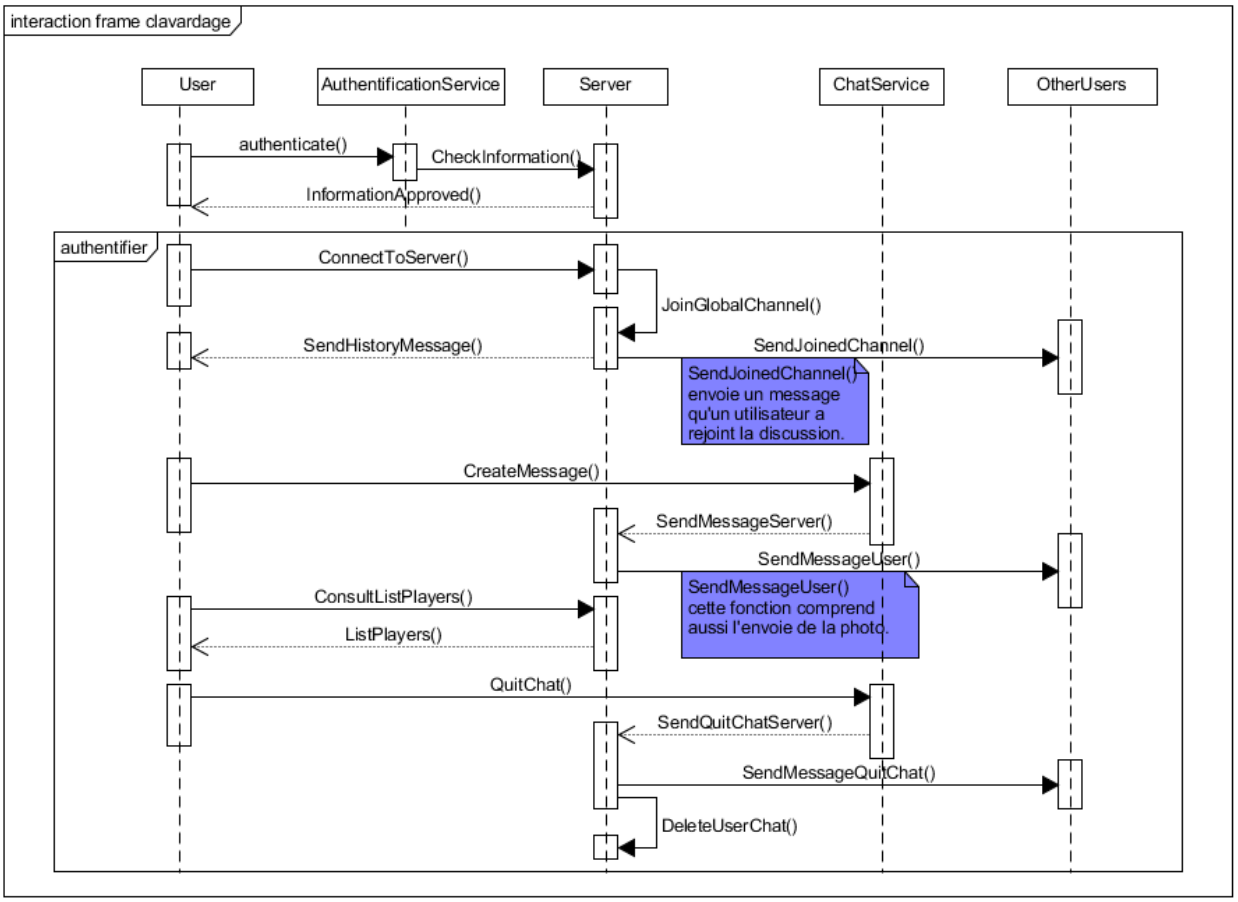

Fig. 19: Diagramme de séquence du « Clavardage »

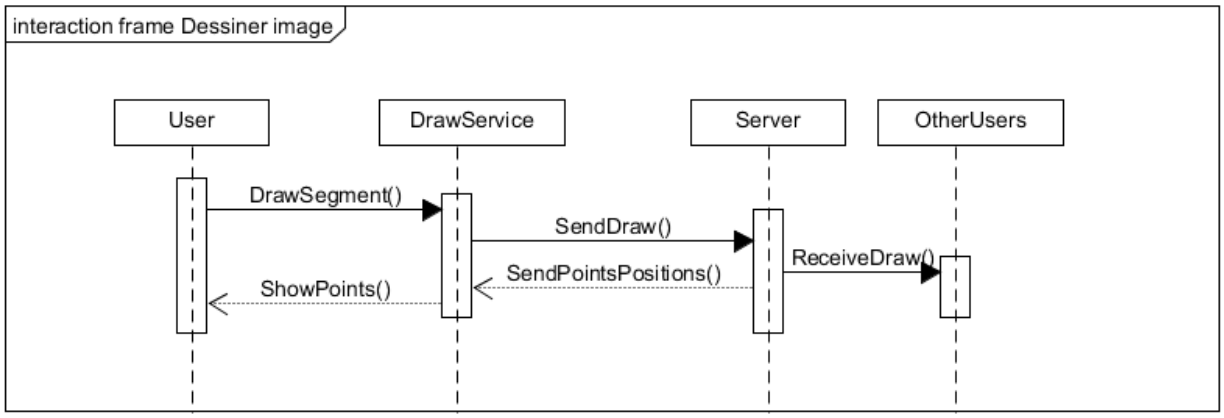

Fig. 20: Diagramme de séquence de « Dessiner Image »

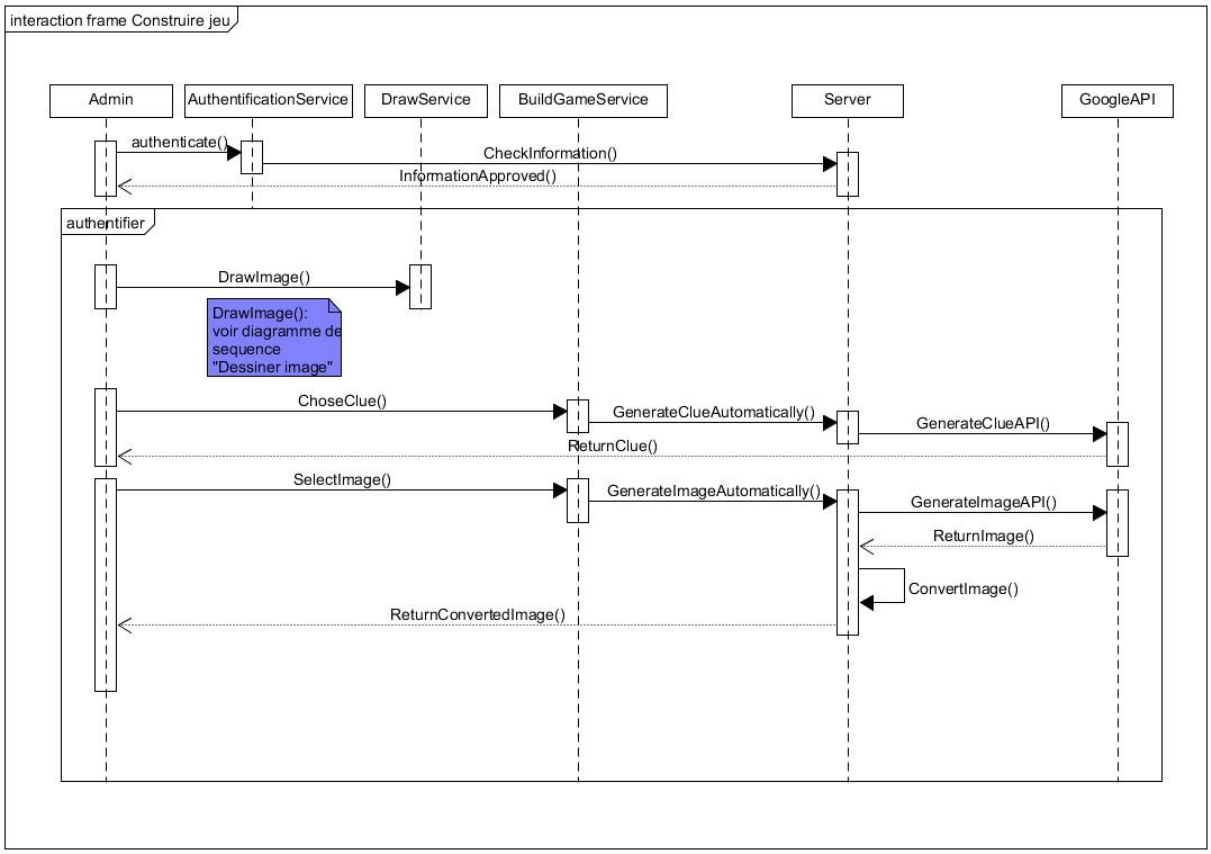

Fig. 21: Diagramme de séquence de « Construire jeu »

6. Vue de déploiement

Comme la figure ci-dessous nous montre (Fig. 23), la communication entre les deux clients: le client lourd qui sera sur un ordinateur, et le client léger qui sera sur une tablette, sera fait en communiquant premièrement avec le serveur. Cette communication sera possible grâce aux SocketIO. De plus, le serveur est le seul qui communiquera avec notre base de données.

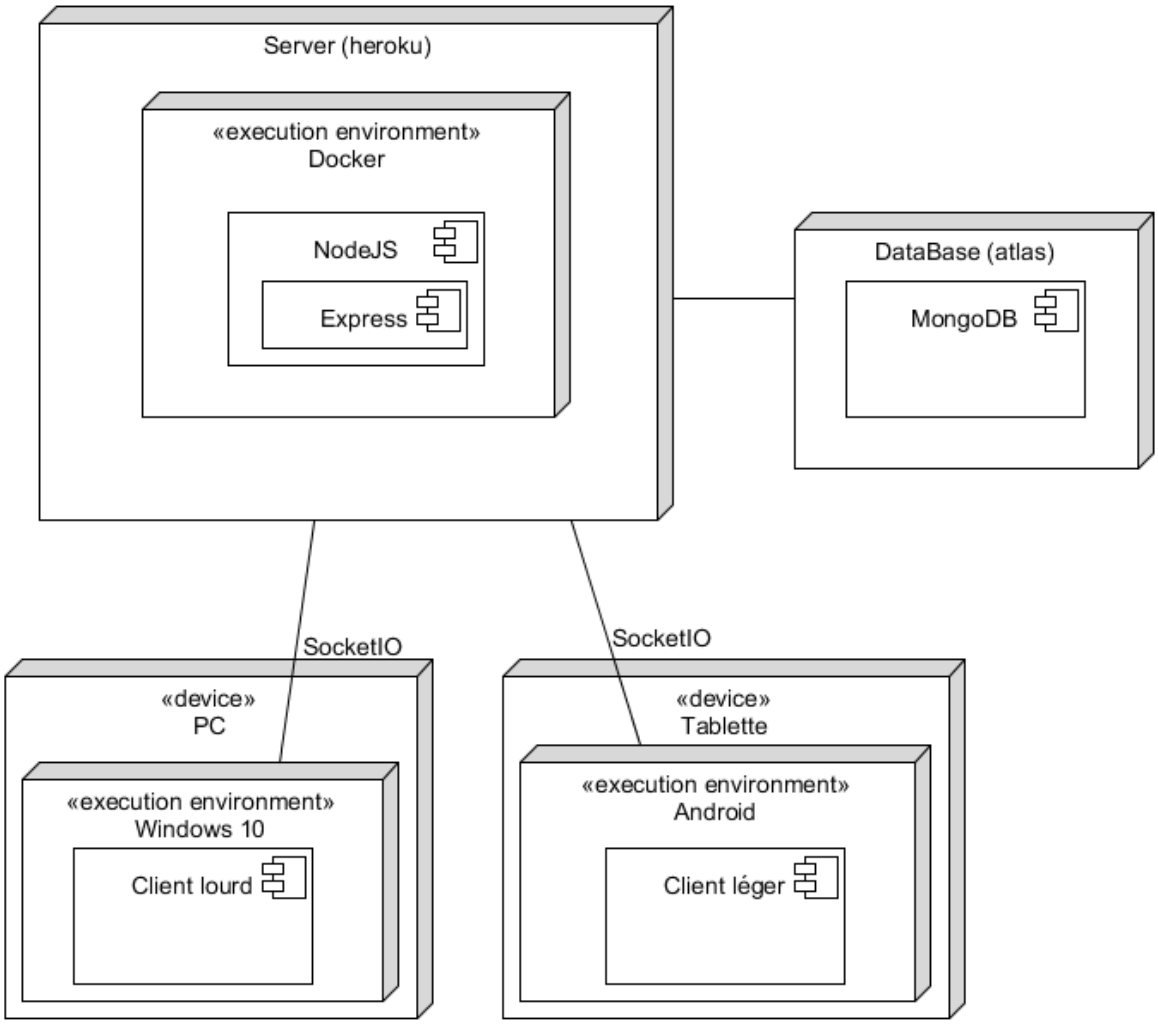

Fig. 22: Diagramme de déploiement

7. Taille et performance

Les caractéristiques de taille et de performance qui peuvent avoir un impact sur notre architecture et sur notre design du logiciel seraient du côté du serveur. Notre serveur, qui est implémenté en JavaScript sur NodeJS, gère toutes les interactions entre les deux clients, client lourd et client faible. Notre serveur doit supporter jusqu'à 16 utilisateurs simultanément, vu qu'on a ajouté un mode tournoi aux modes de jeu. En ce qui concerne le clavardage, notre serveur doit être capable d'avoir au moins 16 canaux de clavardage, tous les messages qui seront transmis entre les clients ne doivent pas avoir un grand délai (plus que 2 secondes) entre le temps que le message est envoyé et reçu. L'application ne doit pas avoir des gels soudain. La mémoire vive et le CPU ne sont pas trop chargés, vu qu'on n'utilise pas des algorithmes récursifs. De plus, le temps de calcul pour le client lourd est assez rapide pour qu'on ne le détecter pas à l'œil nu. Enfin, on s'attend à ce qu'on ne reçoive pas un grand délai (plus que 2 secondes) entre le temps qu'une image est dessinée et affichée pour le client léger.