Équipe 102

Fais-moi un dessin
Document d'architecture logicielle

Version 1.2

Historique des révisions

Date	Version	Description	Auteur
2020-09-10	1.0	Version initiale	Équipe 102
2020-09-30	1.1	Révision du document	Vincent L'Ecuyer-Simard, Simon Berhanu, Abderrahim Ammour
2020-10-01	1.2	Mise en forme et correction du français	Vincent L'Ecuyer-Simard

Table des matières

Document d'architecture logicielle

1 Introduction

Le présent document détaille le flot des fonctionnalités majeures du logiciel *Fais-moi un dessin* en plus de donner certains détails sur son implémentation prévue. Les objectifs et les contraintes architecturales seront d'abord exposés. Par la suite, plusieurs cas d'utilisation seront présentés afin de pouvoir visualiser les actions que pourra accomplir un utilisateur au sein de plusieurs fonctionnalités majeures, en faisant abstraction de l'implémentation de celles-ci.

La section suivante, soit celle de la vue logique, permettra d'entrer dans les détails de la structure statique prévue du logiciel. Ensuite, la vue des processus permettra de conjuguer les 2 vues précédentes afin de visualiser comment interagissent les composants logiciels statiques pour permettre la complétion de cas d'utilisation importants. La vue de déploiement permettra par la suite de visualiser l'architecture physique du logiciel, soit le support physique des différentes entités logicielles faisant partie de *Fais-moi un dessin* ainsi que les moyens de communication entre elles. Finalement, différentes contraintes en lien avec la taille et les performances du logiciel seront abordées.

2 Objectifs et contraintes architecturaux

L'objectif de l'architecture actuelle est de permettre la complétion des différentes exigences essentielles du projet, tout en demeurant assez modulable pour permettre l'ajout de fonctionnalités souhaitables par la suite. Elle devra donc être à la fois simple et ingénieuse et les responsabilités de chaque composante devront être bien définies. L'architecture devra également être utilisable autant pour la version bureau que la version mobile. Finalement, elle devra être assez précise pour diviser clairement les composants et donner une vision claire du projet à compléter, mais tout en demeurant assez flexible pour permettre la créativité en cours de projet et pour éviter de gaspiller des ressources en surconcevant étant donné les variations inhérentes aux projets logiciels.

Certaines contraintes sont spécifiées pour ce projet. Tout d'abord, la réalisation de deux clients est nécessaire, soit un client lourd pour ordinateur Windows 10 et un client léger conçu pour tablette Android 9.0 Pie. Le client lourd doit être conçu à l'aide de l'échafaudage Angular et du langage de programmation TypeScript. L'échafaudage Electron doit ensuite être utilisé pour permettre l'utilisation de ce projet Angular sur bureau. Le client léger doit être développé en Java ou en Kotlin. Des parties entre des utilisateurs utilisant différents types de clients doivent être possibles.

Du côté des coûts, bien qu'il n'y ait pas formellement de maximum, ceux-ci doivent être estimés à 100$/h pour le développement et 125$/h pour la gestion du projet. Au niveau de l'échéancier, le projet doit être complété à l'intérieur d'une période de 2 mois. Ainsi, l'architecture devra être la moins risquée possible (en utilisant des patrons connus par exemple) et recourir à des librairies libres de droits au besoin afin de minimiser les coûts et maximiser l'efficacité.

Finalement, par souci de confidentialité et de sécurité, les clés d'accès aux données des utilisateurs ne devront pas être accessibles directement par les clients, pour éviter toute rétro-ingénierie pour les décoder de la part d'individus malveillants.

3 Vue des cas d'utilisation

La section suivante présente les diagrammes des cas d'utilisation les plus importants de *Fais-moi un dessin*.

3.1 Diagramme de cas d'utilisation du clavardage

La Figure 3.1 présente le diagramme de cas d'utilisation du clavardage, soit les différentes actions possibles au niveau du système de clavardage des clients lourds et légers.

Figure 3.1. Diagramme de cas d'utilisation du clavardage

3.2 Diagramme de cas d'utilisation de la gestion de profil d'utilisateur

La Figure 3.2 présente le diagramme de cas d'utilisation de la gestion de profil utilisateur, soit les différentes actions possibles au niveau du système de gestion et de visualisation de profil des clients lourds et légers.

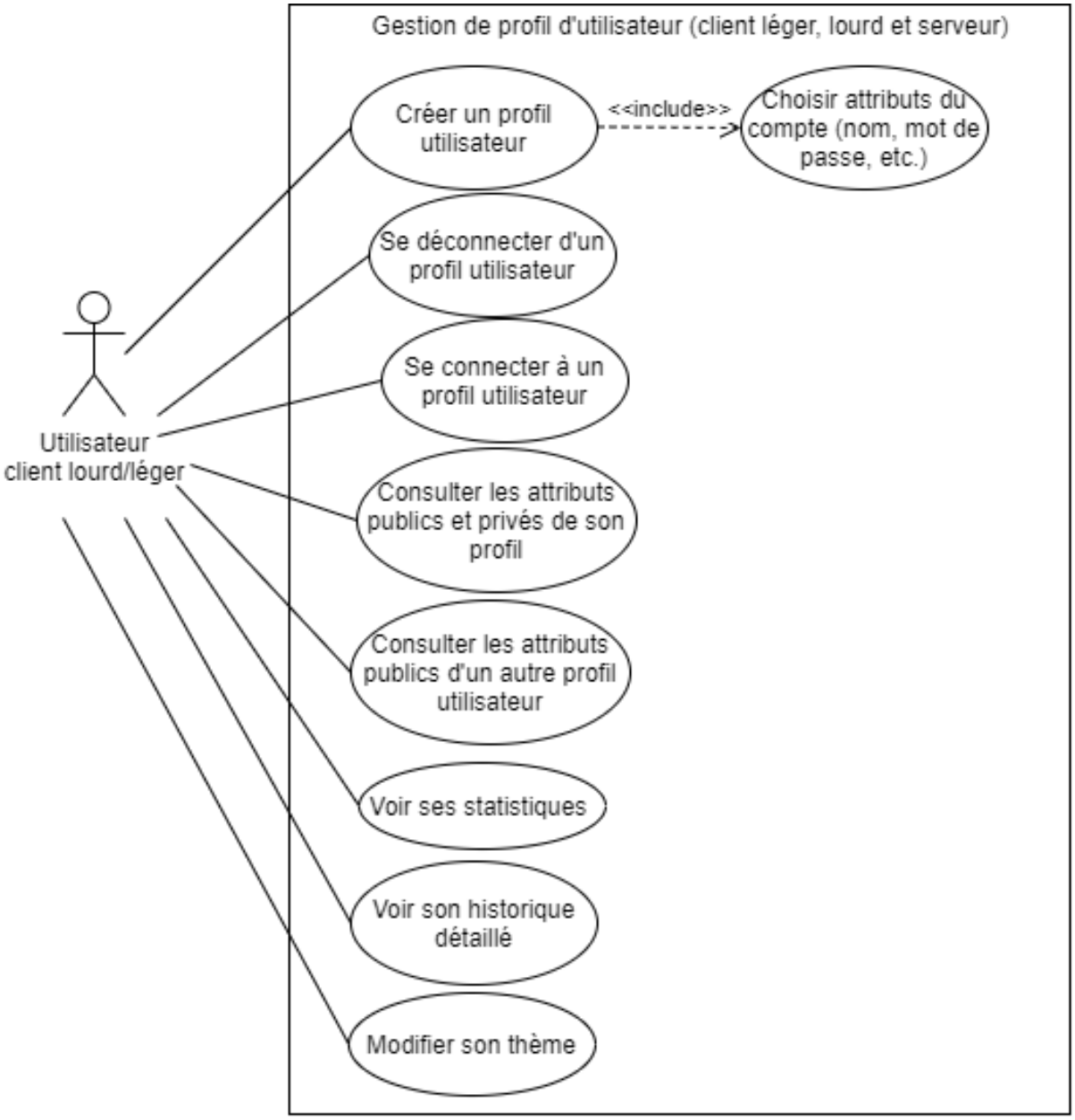

Figure 3.2. Diagramme de cas d'utilisation de la gestion de profil utilisateur

3.3 Diagramme de cas d'utilisation de la navigation dans le menu principal

La Figure 3.3 présente le diagramme de cas d'utilisation de la navigation dans le menu principal, soit les différentes actions possibles à partir du menu principal, avec une emphase sur les actions menant au début d'une partie.

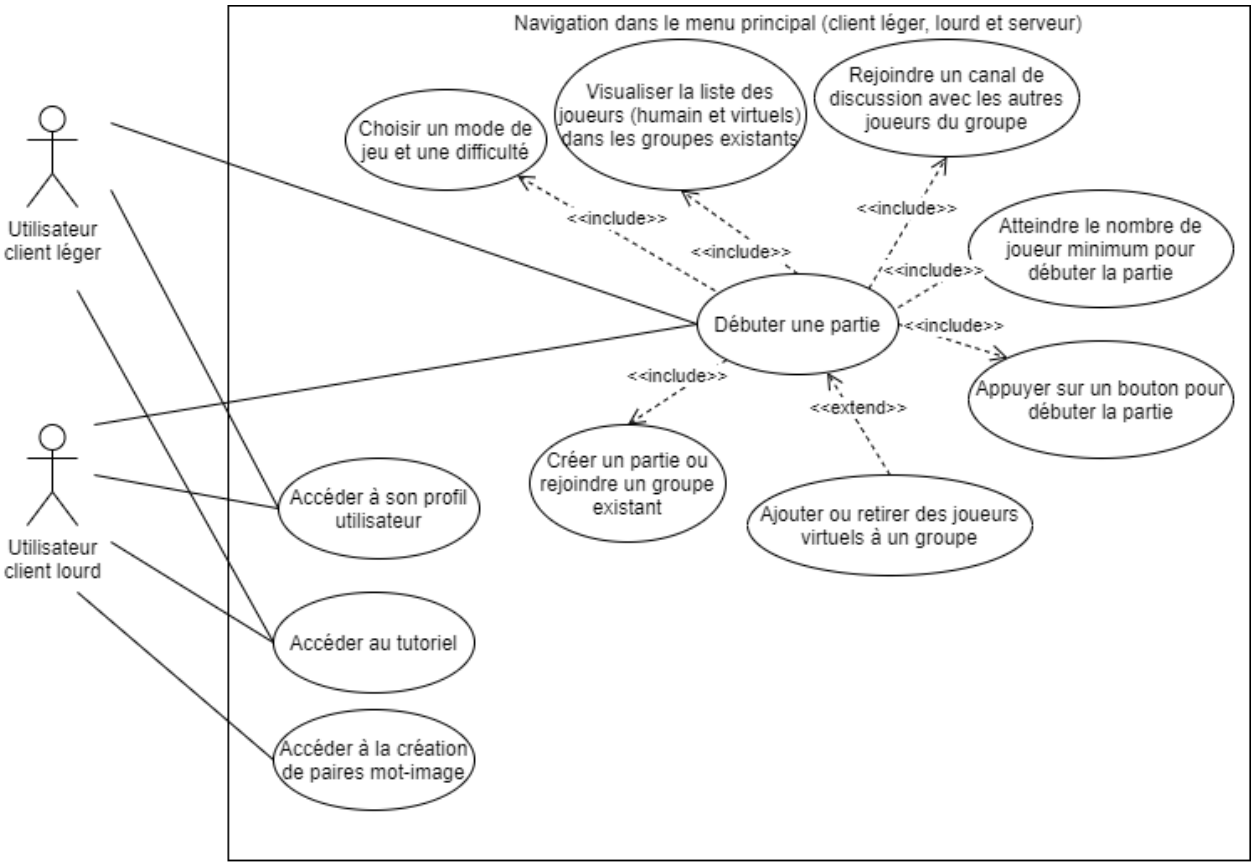

Figure 3.3. Diagramme de cas d'utilisation de la navigation dans le menu principal

3.4 Diagramme de cas d'utilisation de la réalisation d'un dessin

La Figure 3.4 présente le diagramme de cas d'utilisation de la réalisation d'un dessin, soit les différentes actions possibles lorsqu'un joueur assume le rôle de dessinateur durant une partie.

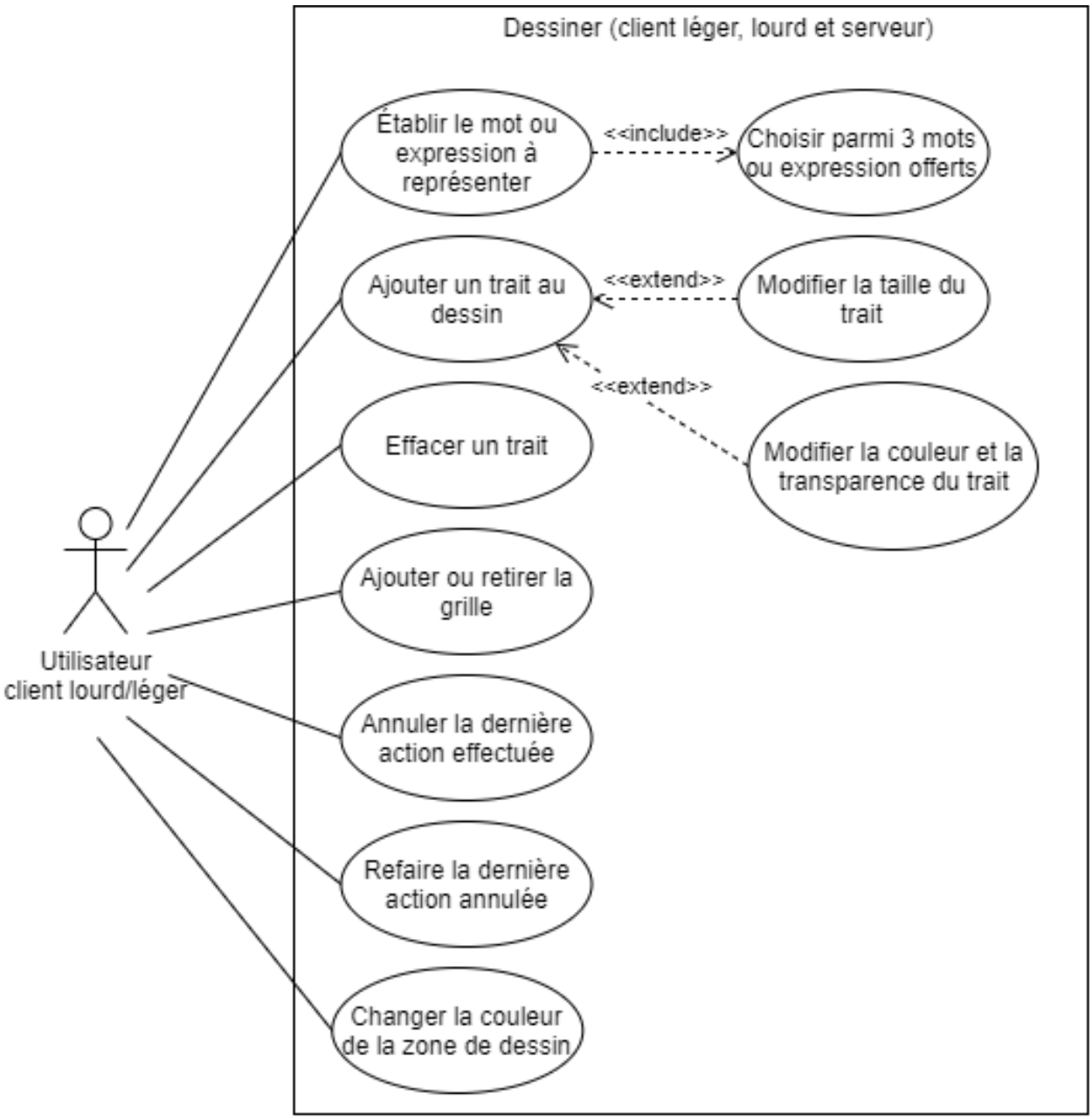

Figure 3.4. Diagramme de cas d'utilisation de la réalisation d'un dessin

3.5 Diagramme de cas d'utilisation de la création d'une paire mot-image

La Figure 3.5 présente le diagramme de cas d'utilisation de la création d'une paire mot-image, soit les différentes actions possibles lorsqu'un utilisateur souhaite créer une nouvelle paire mot-image à partir du client lourd.

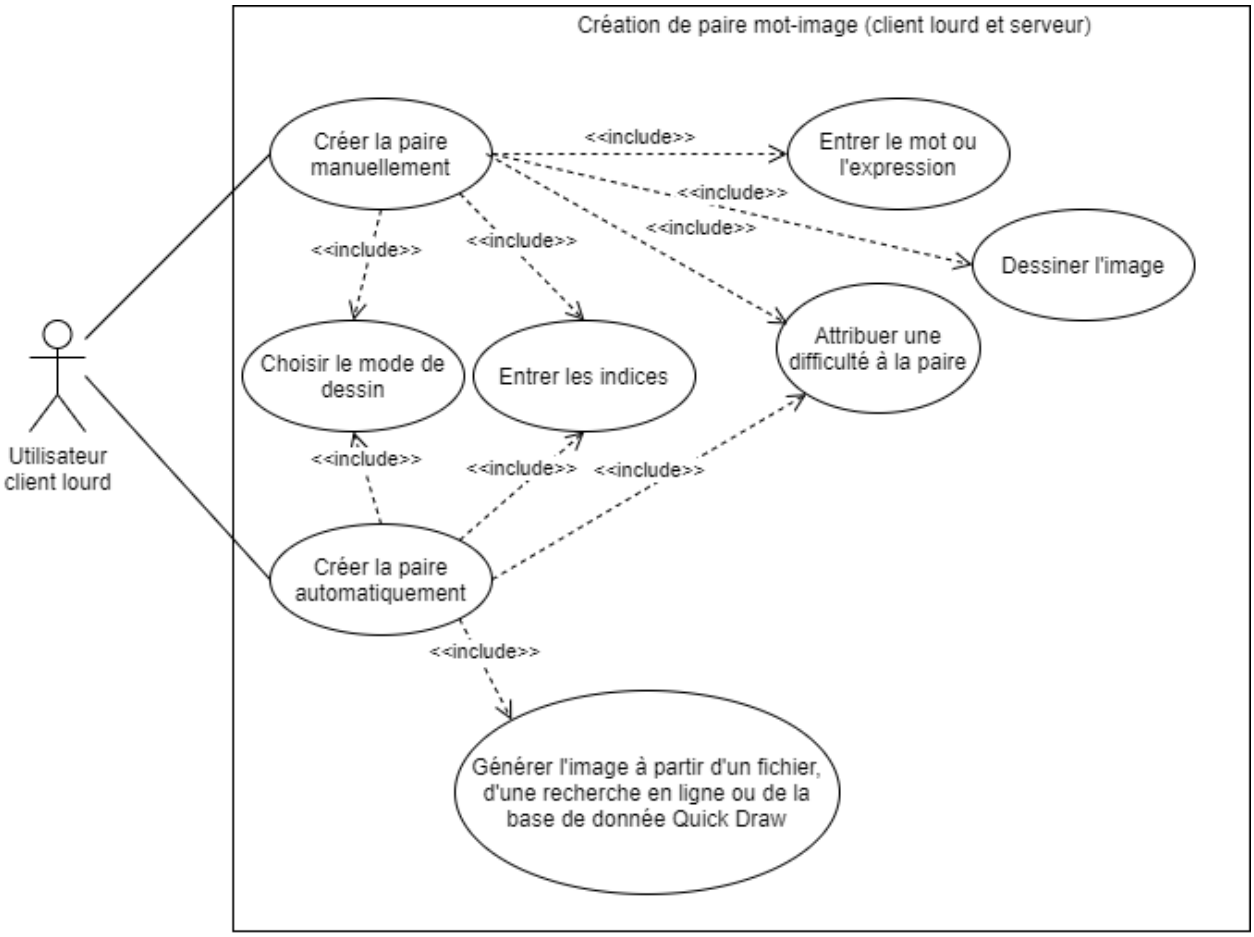

Figure 3.5. Diagramme de cas d'utilisation de la création d'une paire mot-image

3.6 Diagramme de cas d'utilisation pour un devineur

La Figure 3.6 présente le diagramme de cas d'utilisation pour un devineur, soit les différentes actions possibles pour l'utilisateur lorsqu'il assume le rôle de devineur pendant une partie.

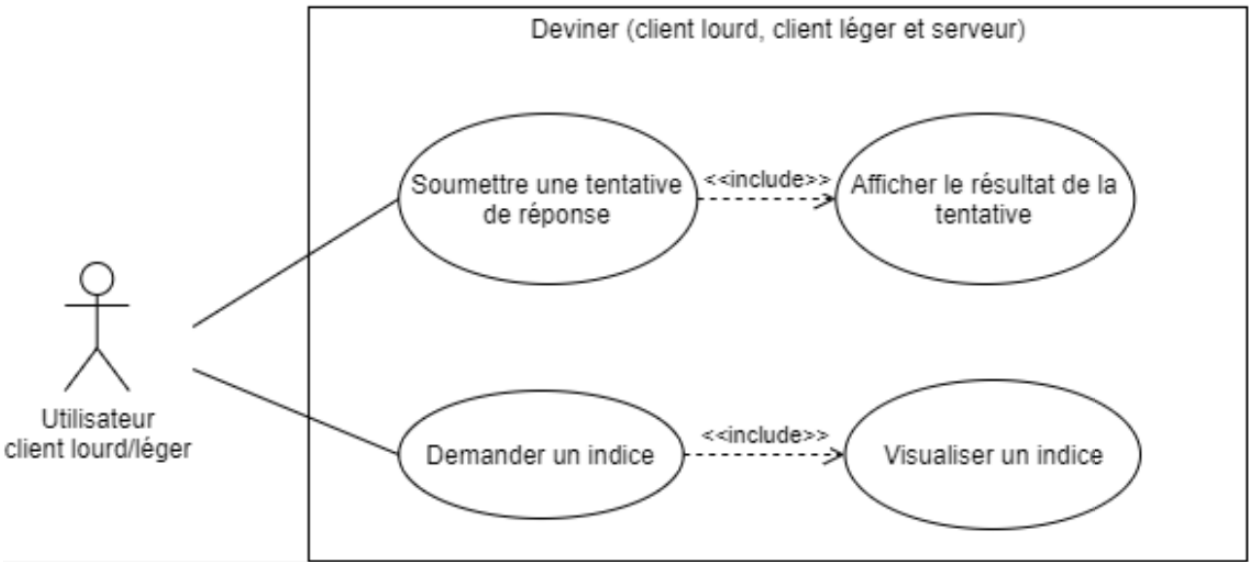

Figure 3.6. Diagramme de cas d'utilisation pour un devineur

4 Vue logique

Dans la présente section, un diagramme de paquetages sera présenté. Par la suite, des diagrammes de classes de différentes sections architecturalement significatives seront présentés, autant pour les clients que pour le serveur. Tous les diagrammes seront accompagnés d'un tableau permettant de définir davantage le rôle de chaque paquetage ou classe présenté.

4.1 Paquetages de haut niveau

4.1.1 Diagramme de paquetages de haut niveau

La Figure 4.1 présente le diagramme de paquetages de haut niveau du logiciel *Fais-moi un dessin*. On peut y voir le logiciel en entier et ses 3 principales entités, soit les clients léger et lourd ainsi que le serveur.

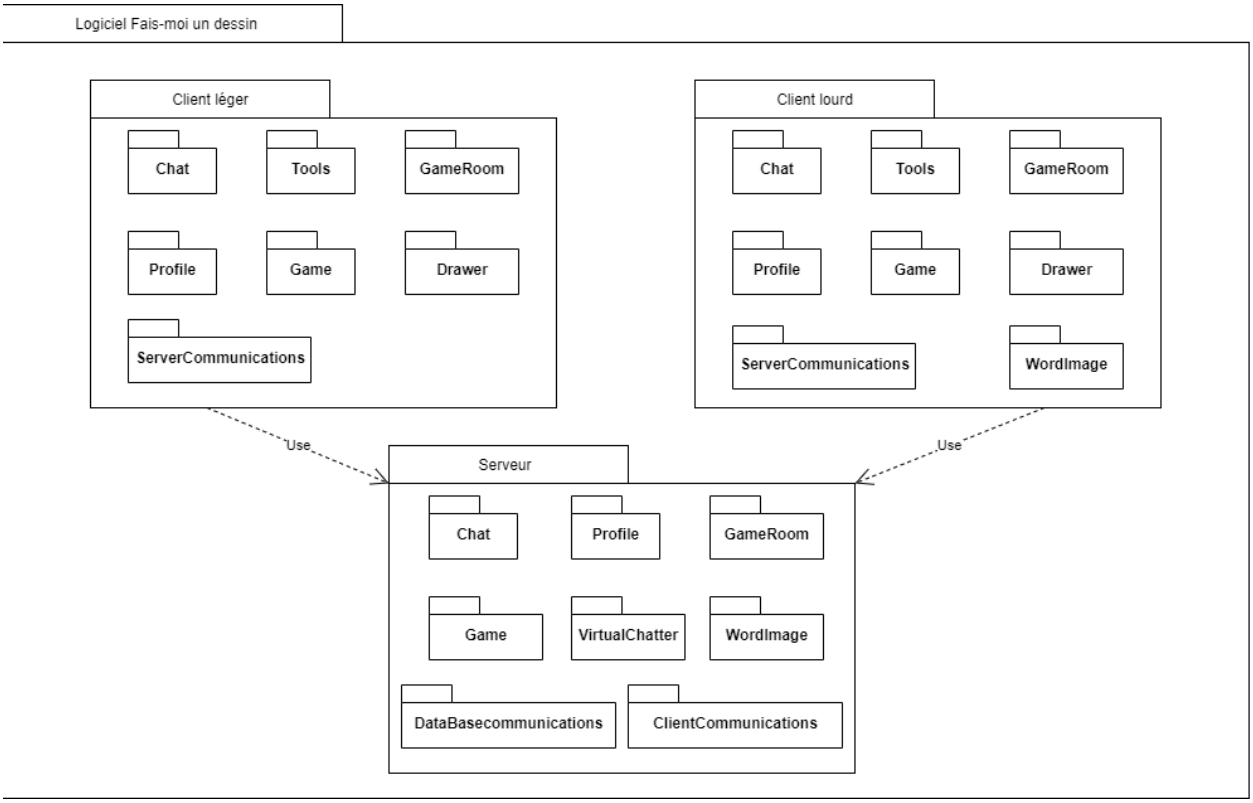

Figure 4.1. Diagramme de paquetages de haut niveau

4.1.2 Responsabilités des paquetages de haut niveau du client léger et du client lourd

Le Tableau 4.1 présente la responsabilité de chacun des paquetages appartenant aux clients, soit les paquetages englobants nommés "Client lourd" et "Client léger" sur la Figure 4.1. Puisque les paquetages partageant le même nom ont le même rôle pour chacun des clients, chacun ne sera présenté qu'une seule fois.

Chat
Ce paquetage regroupe l'ensemble des classes utiles au clavardage, donc à la gestion des canaux de discussion et des messages entrants et sortants.
Tools
Ce paquetage regroupe l'ensemble des classes permettant au joueur de modifier le contenu de la surface de dessin lorsqu'il est dessinateur.

GameRoom
Ce paquetage regroupe l'ensemble des classes utiles à la gestion des groupes de joueurs.
Profile
Ce paquetage regroupe les classes utiles à la création, la connexion et l'affichage du profil utilisateur du joueur ou encore des autres joueurs rencontrés en ligne.
Game
Ce paquetage regroupe les classes utiles au bon déroulement d'une partie.
Drawer
Ce paquetage regroupe les classes utiles pour représenter les différents types de dessinateurs, soit local, distant ou virtuel.
ServerCommunications
Ce paquetage regroupe les classes utiles à la communication avec le serveur, donc celles impliquées dans la transmission ou la réception d'information avec celui-ci.
WordImage (client lourd seulement)
Ce paquetage regroupe les classes utiles à la création d'une paire mot-image.

Tableau 4.1. Responsabilités des paquetages de haut niveau du client léger et du client lourd

4.1.3 Responsabilités des paquetages de haut niveau du serveur

Le Tableau 4.2 présente la responsabilité de chacun des paquetages appartenant au serveur, soit le paquetage englobant nommé "Serveur" sur la Figure 4.1.

Chat
Ce paquetage regroupe l'ensemble des classes utiles au clavardage, donc à la gestion des canaux de discussion, des messages entrants et sortants et à la distribution des changements auprès des clients.
GameRoom
Ce paquetage regroupe l'ensemble des classes utiles à la gestion des groupes de joueurs et à la mise à jour de ceux-ci auprès des clients lors de changements.
Profile
Ce paquetage regroupe les classes utiles à la création, la connexion et la récupération de profils utilisateurs.
Game
Ce paquetage regroupe les classes utiles au bon déroulement des parties en cours, de la synchronisation avec les clients lors de celles-ci et à la sauvegarde des résultats lorsqu'elles sont terminées.
VirtualChatter
Ce paquetage regroupe les classes utiles pour permettre les interactions personnalisées des joueurs virtuels sur les différents canaux de clavardage dédiés aux parties.
WordImage
Ce paquetage regroupe les classes utiles à l'ajout d'une paire mot-image à la base de données et à sa récupération lorsque nécessaire pendant une partie.
DatabaseCommunications
Ce paquetage regroupe les classes utiles à la communication avec la base de données, donc celles impliquées dans l'ajout ou la récupération d'informations à partir de celle-ci.

ClientCommunications
Ce paquetage regroupe les classes utiles à la communication avec les clients, donc celles impliquées dans la transmission ou la réception d'information avec ceux-ci pour assurer la synchronisation.

Tableau 4.2. Responsabilités des paquetages de haut niveau du serveur

4.2 Classes des clients

Étant donné qu'il y a peu de différences architecturalement significatives entre le client léger et le client lourd, les classes abordées dans cette section seront utilisées de la même façon pour les deux clients, sauf indication contraire. Étant donné la séparation logique possible entre le clavardage et les autres fonctionnalités au sein du logiciel, les classes liées au clavardage seront d'abord présentées, suivies des classes liées aux autres fonctionnalités.

4.2.1 Diagramme des classes liées au clavardage

La Figure 4.2 présente le diagramme de classes des sections architecturalement significatives des clients en lien avec le clavardage.

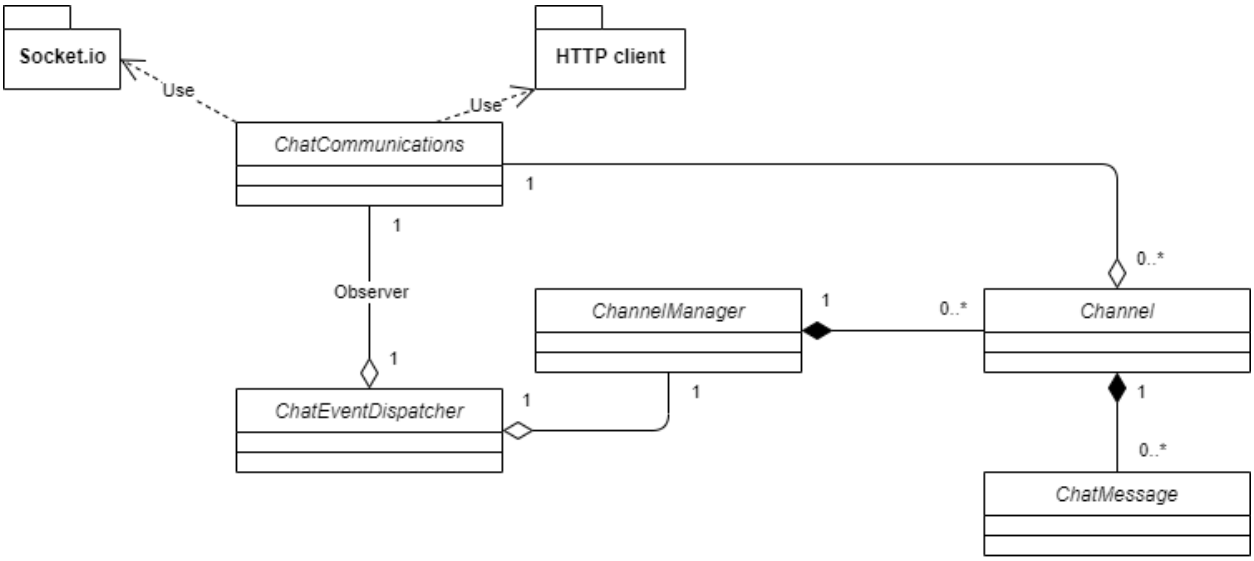

Figure 4.2. Diagramme des classes des clients liées au clavardage

4.2.2 Responsabilités des classes liées au clavardage

Le Tableau 4.3 présente la responsabilité de chacune des classes présentées sur la Figure 4.2.

ChatCommunications
Cette classe est le point d'entrée et de sortie de toutes les communications avec le serveur en lien avec le clavardage. Elle est responsable de préparer les données pour leur sortie et de les acheminer au serveur. Afin d'y arriver, elle utilisera la librairie Socket.io pour les communications bidirectionnelles et des appels HTTP REST pour formuler des requêtes unidirectionnelles au besoin.
ChatEventDispatcher
Cette classe sert à récupérer et à trier les communications entrantes du ChatCommunications. Elle observe donc l'arrivée de messages, prépare les données qu'ils contiennent et communique ensuite avec le ChannelManager pour accomplir les actions nécessaires. Cette structure permet d'éviter les dépendances circulaires dans un contexte de communication bidirectionnelle.

ChannelManager
Cette classe sert à répertorier les canaux de discussion disponibles ainsi que le statut de l'utilisateur local au sein de ces canaux. En cas de communication entrante concernant des changements au niveau d'un canal, celui-ci achemine les changements au canal en question.
Channel
Cette classe représente localement le contenu d'un canal de discussion. Celui-ci est en mesure de communiquer avec le serveur grâce au ChatCommunication en cas d'évènements tels que la composition d'un nouveau message par l'utilisateur local ou encore sa déconnexion d'un canal.
ChatMessage
Cette classe représente un message ayant été envoyé par un utilisateur. Elle contient les informations pertinentes à propos de celui-ci et permet ainsi son affichage.

Tableau 4.3. Responsabilités des classes des clients liées au clavardage

4.2.3 Diagramme des classes indépendantes du clavardage

La Figure 4.3 présente le diagramme de classes des sections architecturalement significatives sans lien avec le clavardage pour le client lourd. Le diagramme de classes pour le client léger est identique, à l'exception de quelques classes à retirer (voir le Tableau 4.4 pour les classes faisant partie uniquement du client lourd).

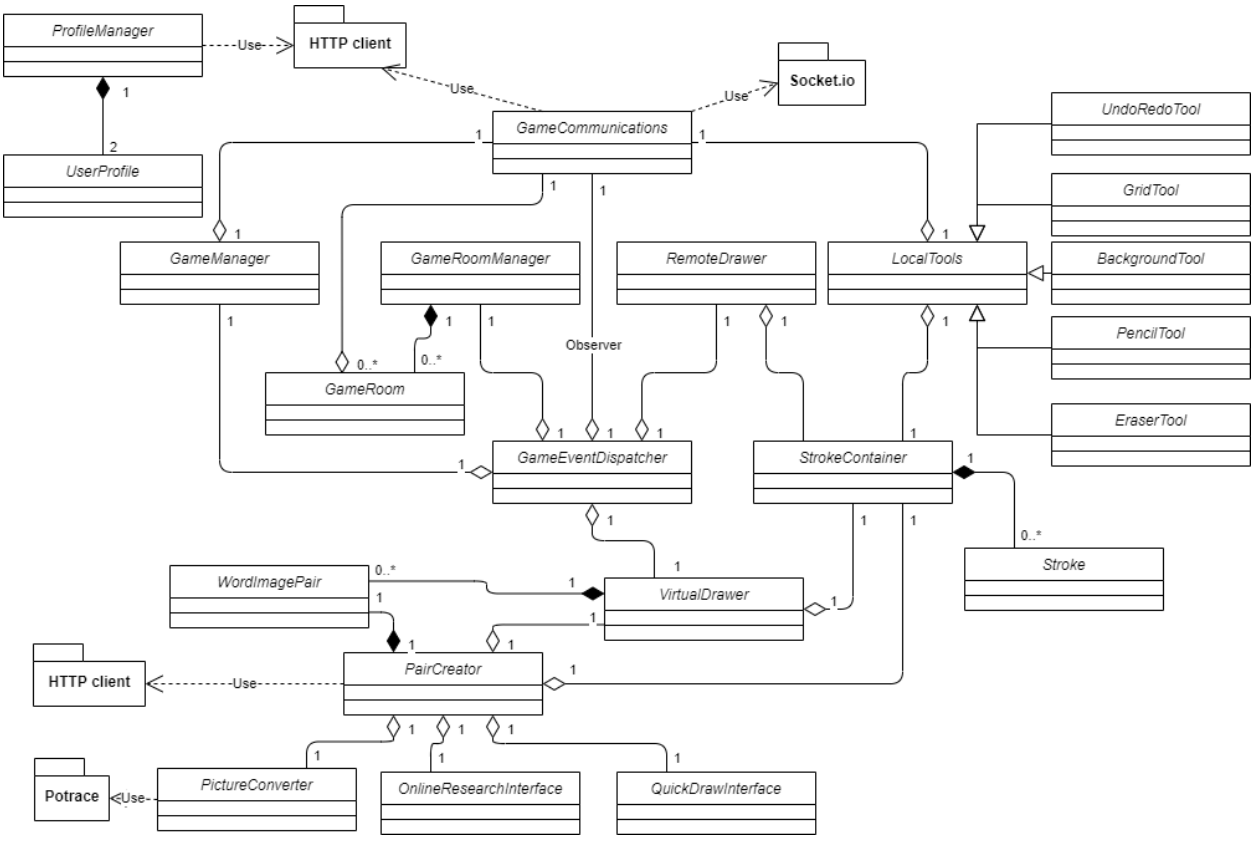

Figure 4.3. Diagramme des classes des clients indépendantes du clavardage

4.2.4 Responsabilités des classes indépendantes du clavardage

Le Tableau 4.4 présente la responsabilité de chacune des classes présentées sur la Figure 4.3.

GameCommunications
Cette classe est le point d'entrée et de sortie de toutes les communications avec le serveur en lien avec une partie. Elle est responsable de préparer les données pour leur sortie et de les acheminer au serveur. Afin d'y arriver, elle utilisera la librairie Socket.io pour les communications bidirectionnelles et des appels HTTP REST pour formuler des requêtes unidirectionnelles au besoin.
GameEventDispatcher
Cette classe sert à récupérer et à trier les communications entrantes du GameCommunications. Elle observe donc l'arrivée de messages, prépare les données qu'ils contiennent et communique ensuite avec le GameManager, le GameRoomManager ou encore le RemoteDrawer pour accomplir les actions nécessaires. Cette structure permet d'éviter les dépendances circulaires dans un contexte de communication bidirectionnelle.
GameRoomManager
Cette classe sert à répertorier localement les GameRoom existantes afin de permettre leur affichage et leur navigation.
GameRoom
Cette classe représente un groupe de joueurs et peut communiquer avec le serveur à l'aide de GameCommunications si un évènement local a lieu, par exemple si le joueur local rejoint un groupe de joueur ou le quitte.
GameManager
Cette classe sert à gérer une partie en cours et à assurer la synchronisation entre ce qui se passe localement avec le serveur et, inversement, a reflété localement les changements distants.
RemoteDrawer
Cette classe représente un dessinateur distant, c'est-à-dire un autre joueur humain qui dessine. Elle est responsable de mettre à jour le StrokeContainer avec les Stroke qui lui parviennent depuis le serveur.
VirtualDrawer
Cette classe représente un dessinateur virtuel. Elle est responsable de stocker la paire mot-image que le dessinateur virtuel doit dessiner et de mettre à jour le StrokeContainer de façon à reproduire l'image désirée.
StrokeContainer
Cette classe est responsable de stocker les différents Stroke ayant été ajoutés sur la surface de dessin. Elle permet également de modifier des Stroke existant, par exemple en les effaçant ou en leur ajoutant des coordonnées.
Stroke
Cette classe représente un trait de crayon.
LocalTools
Cette classe représente un outil local, c'est-à-dire un outil pouvant être utilisé par l'utilisateur local pour modifier le StrokeContainer.
UndoRedoTool
Cette classe représente l'outil annuler-refaire et permet de modifier le StrokeContainer selon les dernières actions ayant été réalisés ou annulées.

GridTool
Cette classe représente l'outil de grille qui permet d'afficher ou non la grille sur la surface de dessin.
BackgroundTool
Cette classe représente l'outil permettant de modifier la couleur de l'arrière-plan de la surface de dessin.
PencilTool
Cette classe représente l'outil crayon ainsi que ses paramètres. Il permet d'ajouter des Stroke au StrokeContainer ou encore de modifier des Stroke existant en leur ajoutant des coordonnées.
EraserTool
Cette classe représente l'outil efface et permet de cacher des Stroke contenus dans le StrokeContainer.
PairCreator (client lourd seulement)
Cette classe a la responsabilité de gérer le processus de création d'une WordImagePair. Dans le cas d'un dessin manuel, il peut recueillir ce que l'utilisateur ajoute au StrokeContainer, sinon il a accès aux différents modules de génération automatiques. Grâce au VirtualDrawer, il peut également reproduire l'aspect qu'aura l'image en cours de partie. Une fois la paire créée, il peut communiquer directement avec le serveur pour l'ajouter à la banque de paires mot-image.
WordImagePair
Cette classe représente une paire mot-image et contient toutes les informations nécessaires à sa reproduction par un VirtualDrawer.
OnlineResearchInterface (client lourd seulement)
Cette classe permet d'utiliser un outil de recherche d'image en ligne afin de générer une paire mot-image.
QuickDrawInterface (client lourd seulement)
Cette classe permet d'utiliser la banque de paires mot-image QuickDraw de Google pour générer la paire mot-image.
PictureConverter (client lourd seulement)
Cette classe permet d'utiliser la librairie Potrace pour produire des images vectorielles à partir d'images matricielles.
ProfileManager
Cette classe a comme responsabilité d'interagir avec le serveur pendant le processus de connexion/déconnexion, en plus de récupérer le profil du joueur ou encore celui d'un autre joueur si l'utilisateur désire le consulter.
UserProfile
Cette classe représente un profil utilisateur et contient les différentes informations en lien avec un profil en particulier.

Tableau 4.4. Responsabilités des classes des clients indépendantes du clavardage

4.3 Classes du serveur

Contrairement au client léger, on ne peut séparer logiquement les classes liées au clavardage puisqu'il y a certaines interactions entre celles-ci et les groupes de joueurs et les joueurs virtuels. Ainsi, la présente section présentera l'entièreté des classes architecturalement significatives du serveur.

4.3.1 Diagramme de classes du serveur

La Figure 4.4 présente le diagramme de classes des sections architecturalement significatives du serveur.

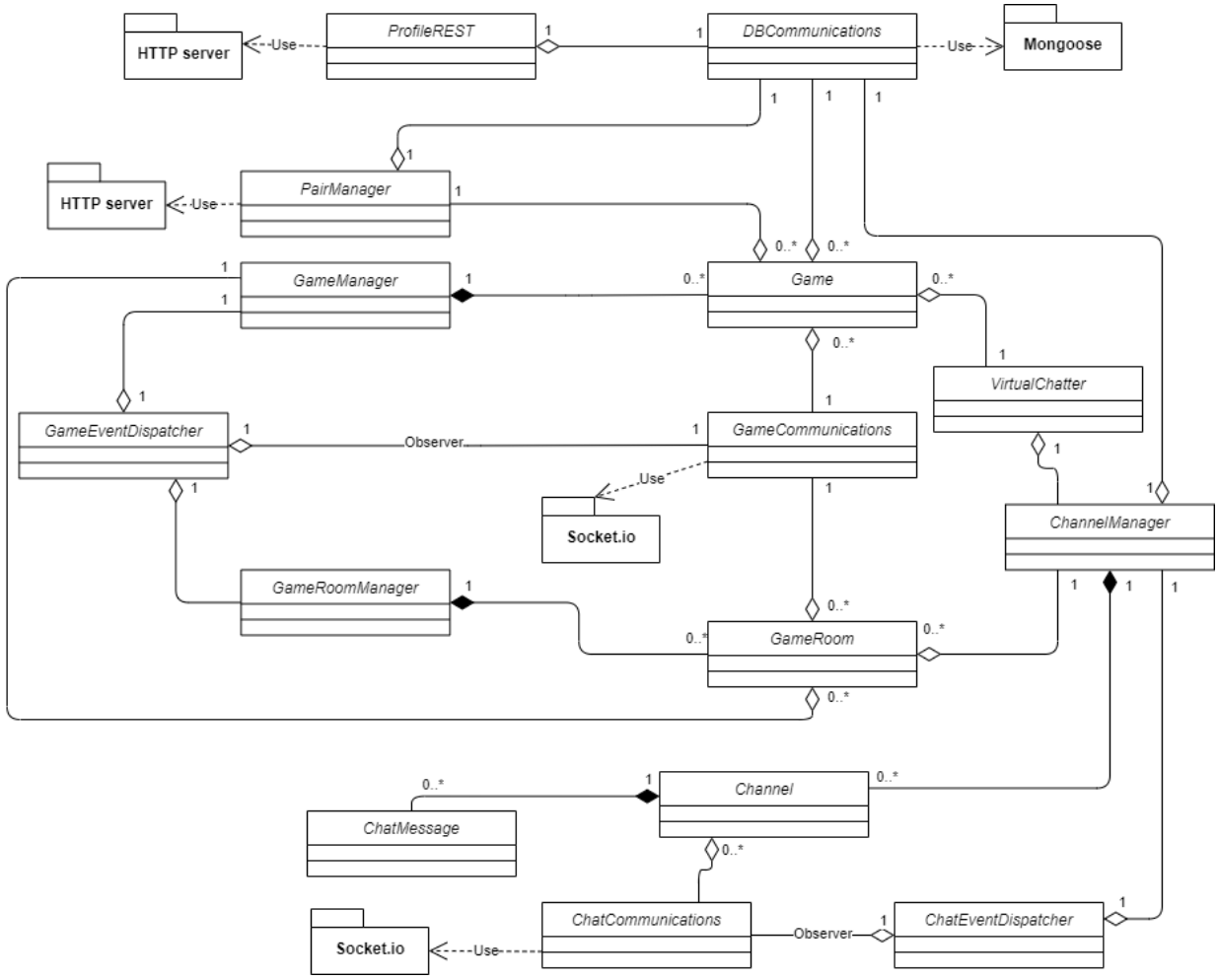

Figure 4.4. Diagramme des classes du serveur

4.3.2 Responsabilités des classes du serveur

Le Tableau 4.5présente la responsabilité de chacune des classes du serveur présenté dans la Figure 4.4.

GameCommunications
Cette classe est le point d'entrée et de sortie de toutes les communications avec les clients en lien avec une partie. Elle est responsable de préparer les données pour leur sortie et de les acheminer aux clients concernés. Afin d'y arriver, elle utilisera la librairie Socket.io pour les communications bidirectionnelles.
GameEventDispatcher
Cette classe sert à récupérer et à trier les communications entrantes du GameCommunications. Elle observe donc l'arrivée de messages, prépare les données qu'ils contiennent et communique ensuite avec le GameManager ou le GameRoomManager pour accomplir les actions nécessaires. Cette structure permet d'éviter les dépendances circulaires dans un contexte de communication bidirectionnelle.

GameRoomManager
Cette classe sert à répertorier toutes les GameRoom existantes afin de permettre leur manipulation (ajout, retrait, etc.).
GameRoom
Cette classe représente un groupe de joueurs et peut communiquer avec les clients à l'aide de GameCommunications pour leur communiquer des changements au niveau de celui-ci, comme, par exemple, si un joueur le rejoint ou le quitte. Cette classe communique également avec le GameManager pour déclencher une nouvelle partie lorsque désiré et le ChannelManager pour créer un nouveau Channel lors de la création du groupe de joueur.
GameManager
Cette classe sert à répertorier toutes les Game actives, à en créer de nouvelles au besoin et à acheminer les évènements entrant à la Game concernée.
Game
Cette classe représente une partie en cours et est responsable de gérer le flot de celle-ci. Grâce à sa connexion avec le GameCommunications, elle est en mesure de synchroniser les clients. Grâce à ses autres connexions, elle est également en mesure de sauvegarder une partie terminée dans la base de données, de récupérer une paire mot-image lorsqu'un joueur virtuel est dessinateur ou encore de signaler au VirtualChatter qu'il doit interagir dans le clavardage.
ChatCommunications
Cette classe est le point d'entrée et de sortie de toutes les communications avec les clients en lien avec le clavardage. Elle est responsable de préparer les données pour leur sortie et de les acheminer à ceux-ci. Afin d'y arriver, elle utilisera la librairie Socket.io pour les communications bidirectionnelles.
ChatEventDispatcher
Cette classe sert à récupérer et à trier les communications entrantes du ChatCommunications. Elle observe donc l'arrivée de messages, prépare les données qu'ils contiennent et communique ensuite avec le ChannelManager pour accomplir les actions nécessaires. Cette structure permet d'éviter les dépendances circulaires dans un contexte de communication bidirectionnelle.
ChannelManager
Cette classe sert à répertorier les Channel. En cas de communication entrante concernant des changements au niveau d'un Channel, celle-ci achemine les changements au Channel en question. Sa communication avec la base de données lui permet de mettre à jour la liste et le statut des Channel lors d'un démarrage.
Channel
Cette classe représente un canal de discussion. Celui-ci est en mesure de communiquer avec les clients grâce au ChatCommunication en cas d'évènements tels que la composition d'un nouveau message pour avertir les clients ayant rejoint ce groupe.
ChatMessage
Cette classe représente un message ayant été envoyé par un utilisateur. Elle contient les informations pertinentes à propos de celui-ci.
PairManager
Cette classe a la responsabilité de permettre la récupération de paires mot-images dans la base de données ou encore d'y en ajouter.

ProfileREST
Cette classe a comme responsabilité d'interagir avec la base de données pour récupérer ou ajouter des profils utilisateurs et gérer le processus de connexion ou de déconnexion lorsqu'il reçoit des requêtes HTTP REST à cet effet de la part des clients.
VirtualChatter
Cette classe est responsable de simuler des interactions entre les joueurs virtuels et les joueurs en cours de partie. Ses connexions avec le GameManager et le ChannelManager lui permettent d'atteindre cet objectif.
DBCommunications
Cette classe est responsable d'accomplir les requêtes auprès de la base de données grâce à la librairie Mongoose.

Tableau 4.5. Responsabilités des classes du serveur

5 Vue des processus

La section suivante présente les diagrammes de séquence à propos des fonctionnalités les plus importantes de *Fais-moi un dessin*.

5.1 Diagramme de séquence de la création et de la connexion d'un profil utilisateur

La Figure 5.1 présente le diagramme de séquence de la création et de la connexion d'un profil utilisateur, soit la séquence d'évènements se produisant pour accomplir ces tâches.

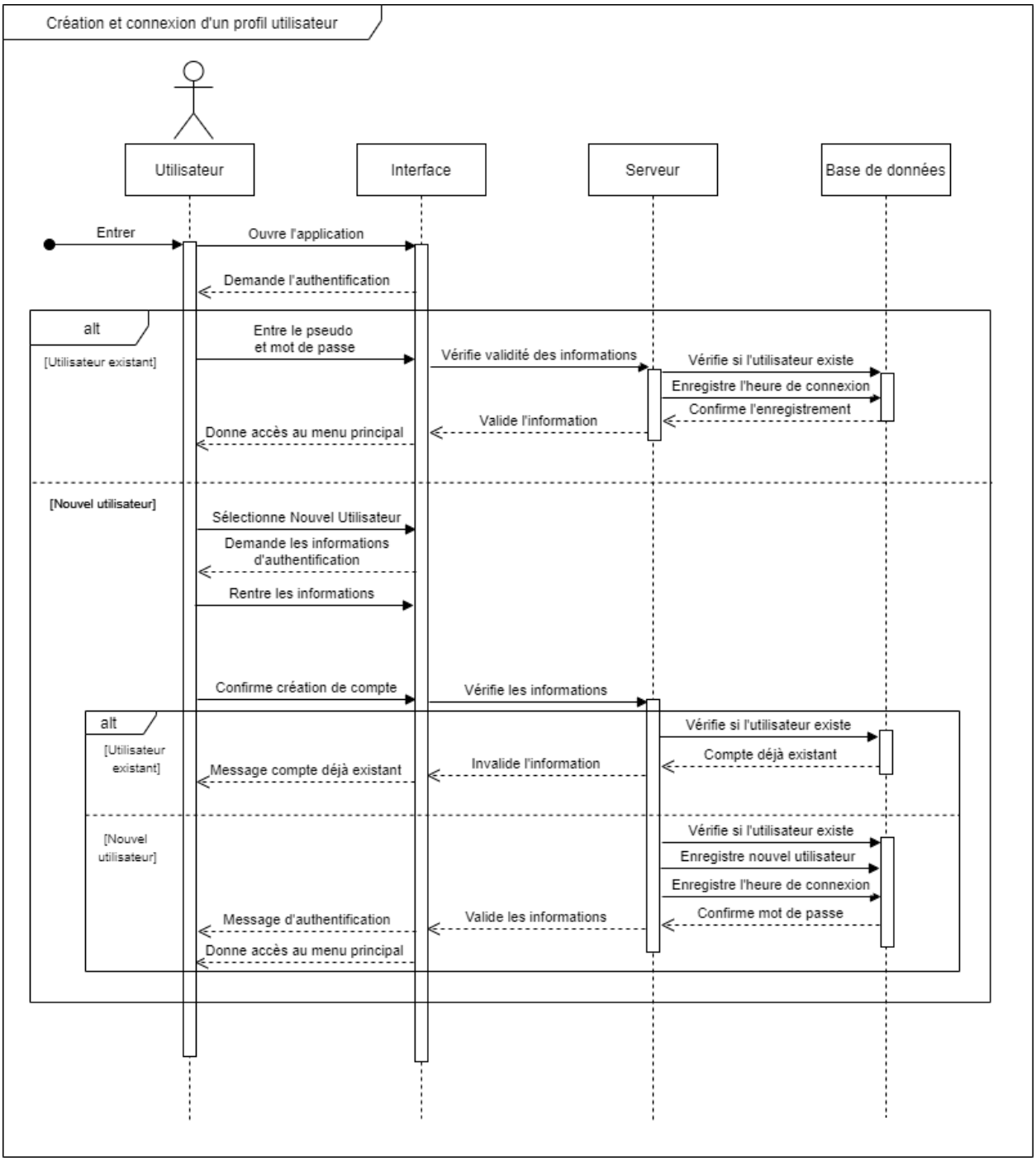

Figure 5.1. Diagramme de séquence de la création et de la connexion d'un profil utilisateur

5.2 Diagramme de séquence de la création, jonction et retrait d'un canal de discussion

La Figure 5.2 présente le diagramme de séquence de la création, jonction et retrait d'un canal de discussion, soit la séquence d'évènements se produisant pour accomplir ces tâches.

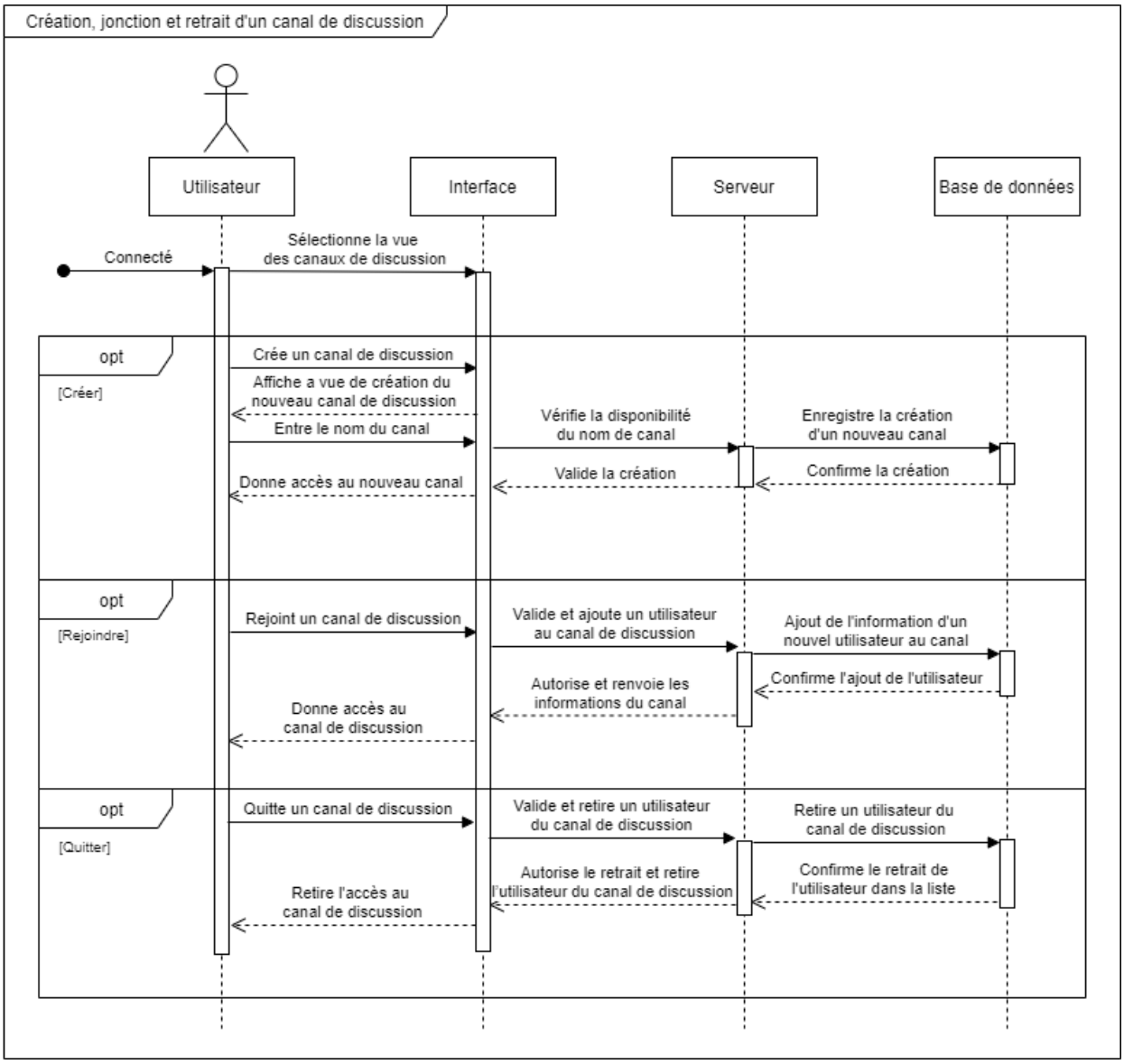

Figure 5.2. Diagramme de séquence de la création, jonction et retrait d'un canal de discussion

5.3 Diagramme de séquence de la création d'une nouvelle partie

La Figure 5.3 présente le diagramme de séquence de la création d'une nouvelle partie, soit la séquence d'évènements se produisant pour accomplir cette tâche.

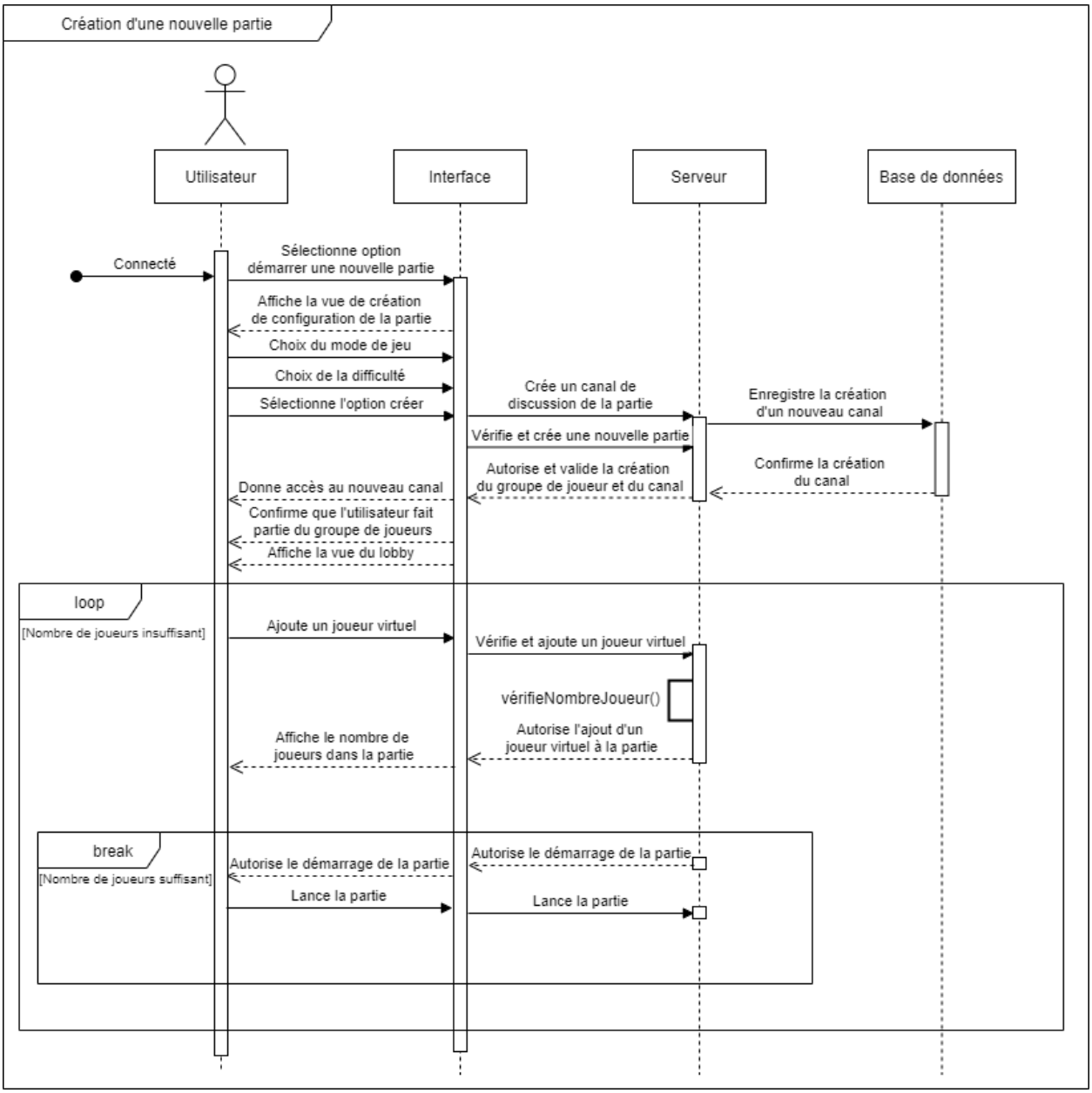

Figure 5.3. Diagramme de séquence de la création d'une nouvelle partie

5.4 Diagramme de séquence du rôle de devineur

La Figure 5.4 présente le diagramme de séquence du rôle de devineur, soit la séquence d'évènements se produisant pour accomplir cette tâche et les différentes possibilités offertes.

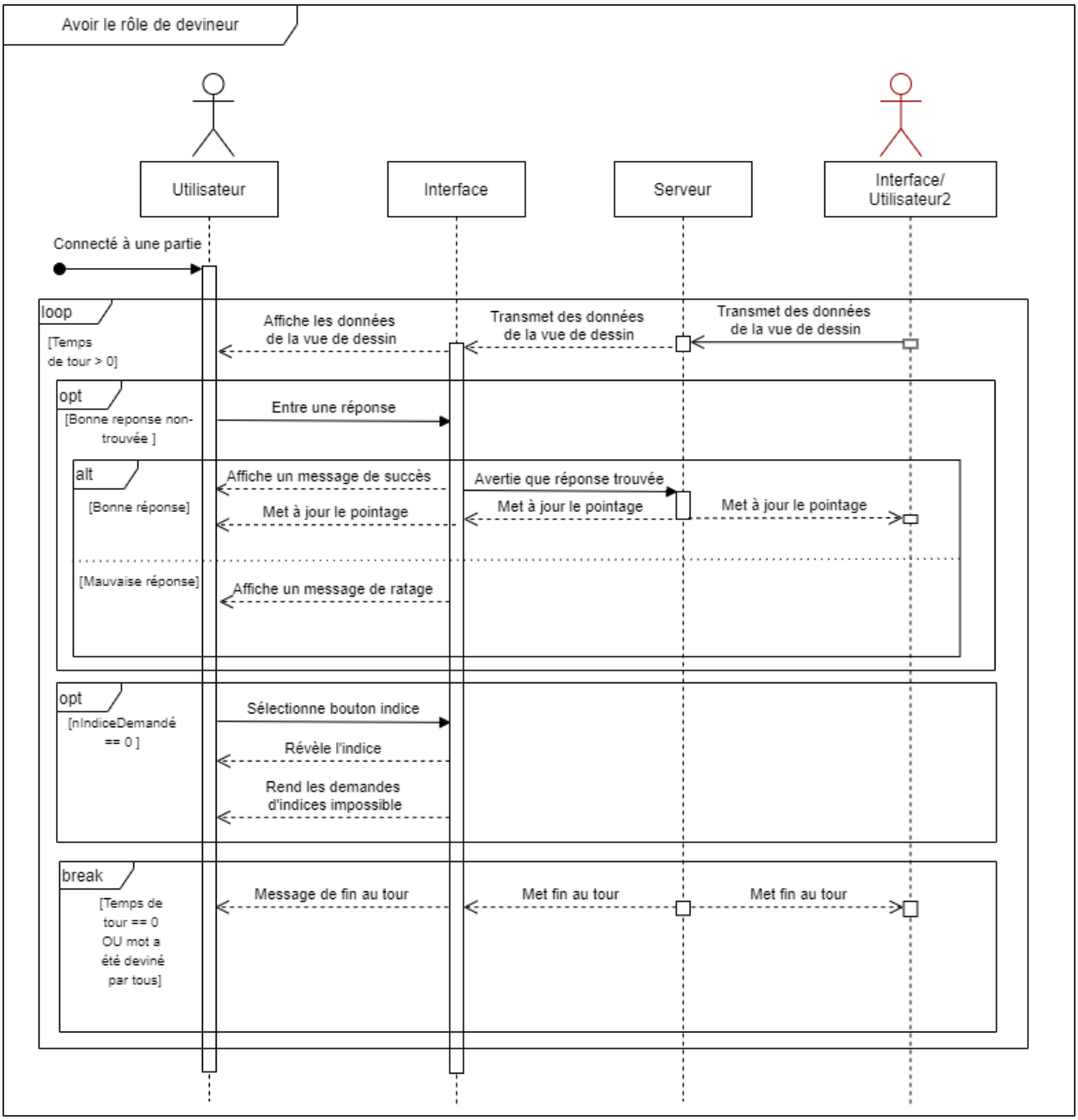

Figure 5.4. Diagramme de séquence du rôle de devineur

5.5 Diagramme de séquence du rôle de dessinateur

La Figure 5.5 présente le diagramme séquence du rôle de dessinateur, soit la séquence d'évènements se produisant pour accomplir cette tâche et les différentes possibilités offertes.

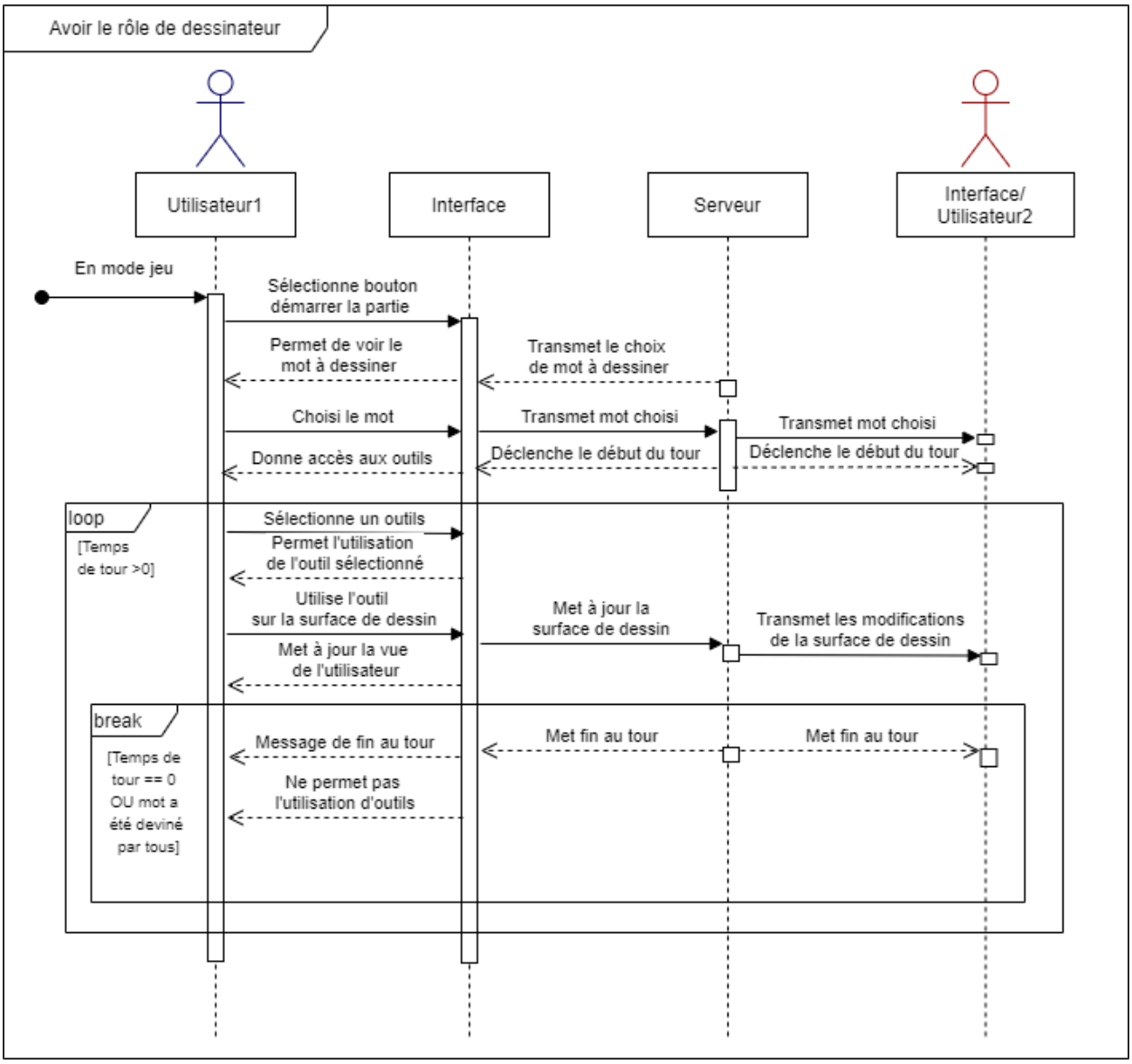

Figure 5.5. Diagramme de séquence du rôle de dessinateur

5.6 Diagramme de séquence de la création d'une paire mot-image

La Figure 5.6 présente le diagramme séquence de la création d'une paire mot-image, soit la séquence d'évènements se produisant pour accomplir cette tâche et les différentes possibilités offertes.

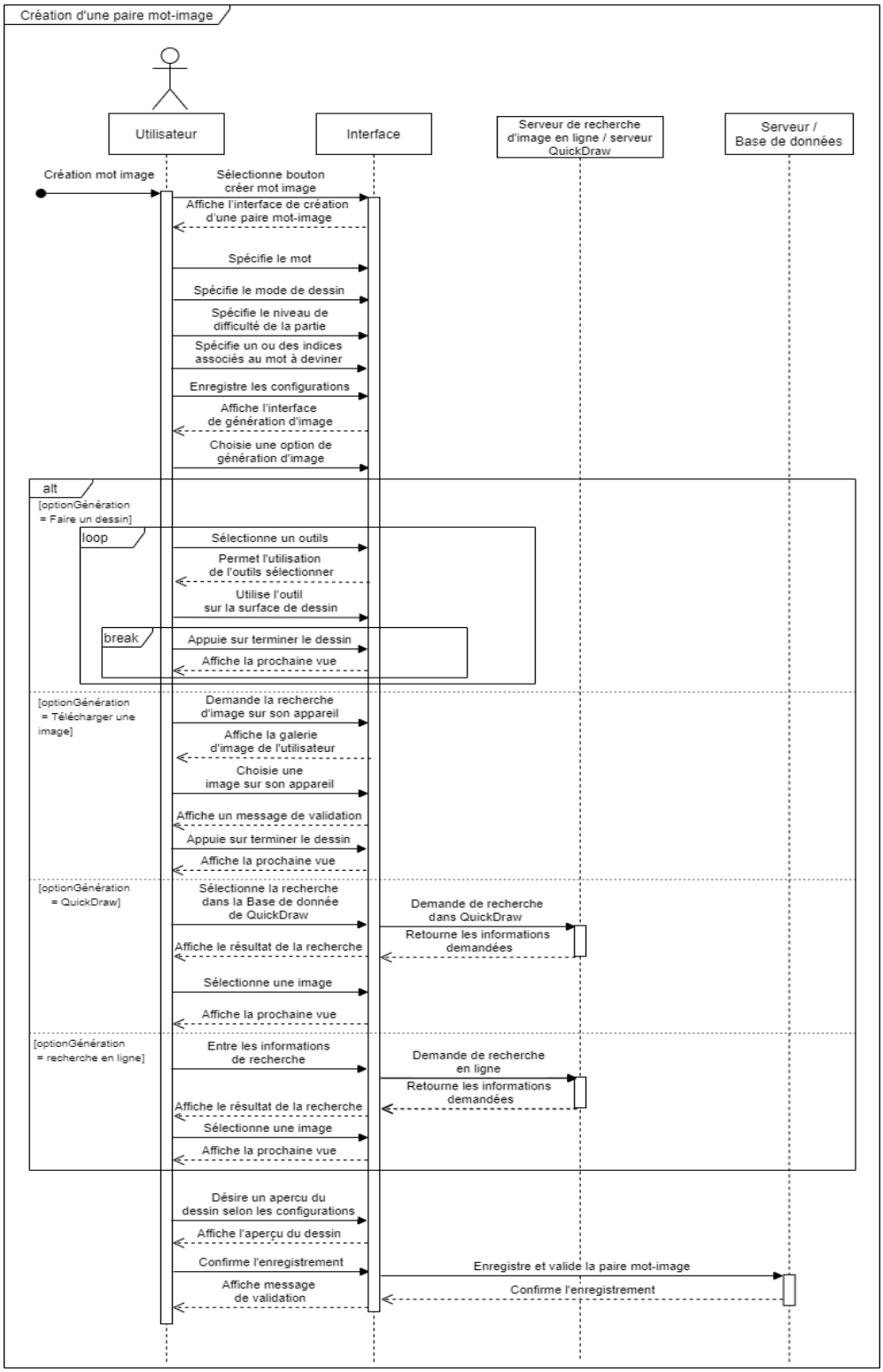

Figure 5.6. Diagramme de séquence de la création d'une paire mot-image

6 Vue de déploiement

La Figure 6.1 présente le diagramme de déploiement pour *Fais-moi un dessin*. Les nœuds physiques y sont montrés ainsi que leur moyen de communication.

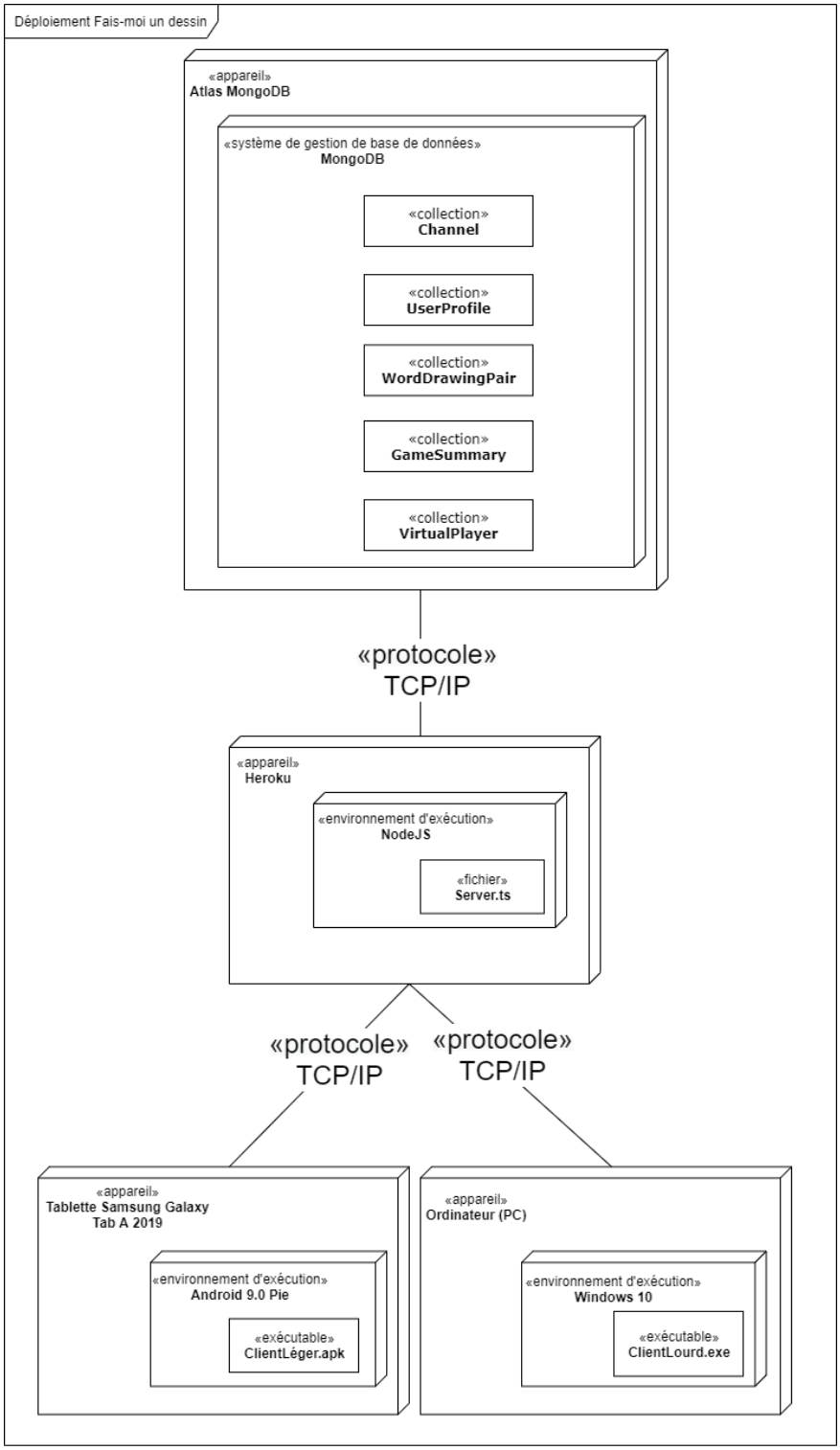

Figure 6.1. Diagramme de déploiement

7 Taille et performance

Comme mentionné dans l'appel d'offres, l'architecture devra permettre de supporter plusieurs parties de 4 joueurs à la fois sans ralentissement notable. La latence lors de la transmission des dessins aux autres utilisateurs devra être minimale afin de rendre l'expérience la plus fluide possible. Elle devra aussi permettre le support du clavardage pour un nombre équivalent d'utilisateurs sans retard notable dans l'envoi des messages textes. Ainsi, le serveur devra être conçu de façon à pouvoir supporter cette charge de travail, soit en minimisant le nombre d'appels vers celui-ci et la complexité des manipulations qu'il doit effectuer pour chacun de ces appels. Les détails quantitatifs en lien avec ces contraintes se trouvent dans le document de spécification des requis (SRS). Afin de vérifier que ces résultats de performance sont atteints, des tests dans les circonstances décrites seront effectués et les délais et temps de réponses seront mesurés. Outre l'aspect réseau, l'architecture devra permettre des interactions simples entre les différents composants de celui-ci et être organisée de façon à minimiser les opérations coûteuses.

Étant donné que le logiciel devra s'exécuter sur mobile, la consommation d'espace disque et de mémoire vive devra également être minimisée pour obtenir des performances optimales sur ce support. Les détails quantitatifs en lien avec ces contraintes se trouvent également dans le SRS. Afin de vérifier que ces résultats de taille sont atteints, la taille finale de l'application installée sera vérifiée et sa consommation de mémoire vive sera également évaluée.

Du côté du client lourd, n'importe quel ordinateur muni d'un processeur moderne (Intel i3 ou mieux) et possédant plus de 2 Go de mémoire vive devrait être en mesure d'exécuter le client lourd en atteignant les objectifs de performances décrites dans le SRS.

Du côté du réseau, bien que les paquets qui doivent transiger sur celui-ci sont plutôt petits, le débit de ceux-ci pourrait parfois être élevé, par exemple durant la transmission d'un dessin en création par un autre utilisateur. Ainsi, afin de pouvoir satisfaire les critères de cohérence et de latence explicités dans le SRS, des vitesses de téléchargement et de téléversement d'environ 1 Mb/s seraient minimales. Ces vitesses étant plutôt basses par rapport aux vitesses médianes disponibles au Canada, cela ne devrait pas entraîner de problèmes pour la très grande majorité des utilisateurs.