Équipe 203

PolyDraw
Document d'architecture logicielle

Version 1.0

Historique des révisions

Date	Version	Description	Auteur
2020-02-07	1.0	Document d'architecture pour le prototype	Allan Beddouk Philippe Côté Pascal Alexandre-Morel Martin Pouliot Samuel Saito-Gagné Cédric Tessier

Table des matières

Document d'architecture logicielle

1. Introduction

Le but de ce document est de présenter l'architecture du logiciel et de ses différentes composantes. Ce document est divisé en plusieurs sections. D'abord, la section « Objectifs et contraintes architecturaux » décrit toutes les contraintes qui affectent l'architecture du logiciel. Ensuite, la section « Vue des cas d'utilisation » décrit tous les cas d'utilisations. Ces cas sont présentés sous la forme de diagrammes. La section « Vue logique » montre l'interaction entre les différentes composantes du logiciel sur les diverses plateformes. Des sections précédentes s'ajoutent la « Vue des processus » et la « Vue de déploiement ». La « Vue des processus » a pour but mettre en évidence les différents processus et leurs interactions: il s'agit principalement des processus de communication avec le serveur. En effet, certaines étapes doivent se faire en plusieurs messages. La dernière section concerne la « Vue de déploiement ». Elle démontre comment le logiciel s'exécute sur les différents appareils. Finalement, la section « Taille et performance » explique les décisions prises et leur impact sur l'utilisation du logiciel.

Ce document sera tenu à jour dans le cas où l'architecture devrait évoluer durant la conception.

2. Objectifs et contraintes architecturaux

La portabilité du serveur est l'un des critères. Celle-ci affecte donc l'architecture de déploiement puisque le serveur est déployé dans Docker. Docker permet au serveur d'être portable, et ce, peu importe le système d'exploitation. Le langage Go du serveur affecte également l'architecture du serveur. Go ne supporte pas l'héritage. Ceci a donc pour effet que les diagrammes ne peuvent pas avoir de relations d'héritage. Go supporte plusieurs structures pour synchroniser les différents fils d'exécution. Ces structures comme les *channels* affectent l'architecture. L'utilisation de la librairie *Gorm* permet d'éviter d'avoir une couche qui s'occupe de faire des requêtes SQL à la base de données. La sécurité affecte également l'architecture du serveur, mais également au niveau des autres plateformes. En effet, le passage du jeton de REST au socket a un impact sur l'architecture.

L'utilisation de WPF implique que le client lourd doit utiliser le patron MVVM. Ce patron va se refléter dans l'architecture. La sécurité avec la gestion des jetons va avoir un impact sur l'architecture. Le langage C# possède des mots clés pour exécuter des tâches de façon asynchrone. Ces fonctionnalités ont un impact sur l'architecture.

L'utilisation d'Android implique d'utiliser le cycle de vie de l'application. Cette façon de développer aura un impact sur l'architecture. Certaines tâches devront être dans des services découplés de l'interface. Il faut donc établir dans l'architecture une communication entre ces deux services. De plus, Android priorise les tâches asynchrones afin de faire une bonne gestion du temps d'attente dans le cas de requêtes web. Les tâches asynchrones auront donc un impact sur l'architecture du client léger.

L'échéancier est aussi un facteur à considérer. Puisque l'échéancier est relativement court, il est impossible de faire des recherches et explorer la littérature pour trouver la meilleure architecture. Ces compromis affectent l'architecture. Celle-ci sera fonctionnelle, mais n'est probablement pas idéale. Un échéancier qui serait plus long permettrait d'allouer davantage de temps à la conception de l'architecture et celle-ci serait probablement plus efficace.

3. Vue des cas d'utilisation

La présente section vise à illustrer les cas d'utilisations les plus pertinents de l'application. Le premier cas d'utilisation présenté est celui de la connexion de l'utilisateur.

3.1 Se connecter

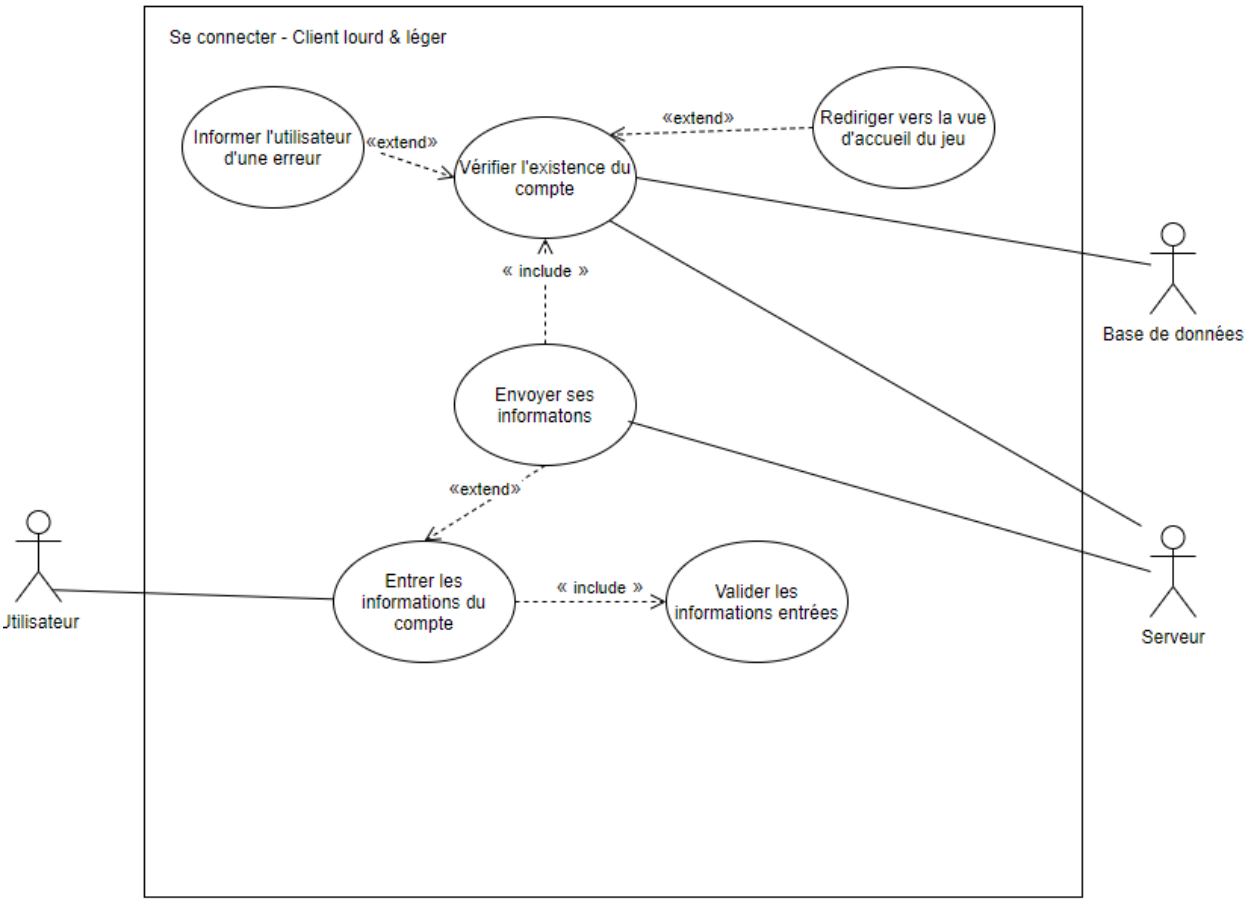

Figure 3.1: Présentation du cas d'utilisation «se connecter».

Pour ce cas d'utilisation, l'utilisateur doit d'abord entrer les informations de son compte (nom d'utilisateur et mot de passe). Ensuite, le client se chargera de valider ces informations. Si les champs à entrer sont invalides, l'utilisateur en sera informé. Une fois les champs remplis, l'utilisateur peut envoyer ses informations au serveur. Un lien «extend» est utilisé entre la bulle «Entrer les informations du compte» et «Envoyer ses informations» pour souligner le fait que l'utilisateur ne peut pas envoyer son nom d'utilisateur et son mot de passe s'ils ne sont pas valides. Après avoir envoyé ses informations, une vérification est faite par le serveur et la base de données. Cette vérification cherche à s'assurer que le compte utilisateur existe et que les informations entrées sont les bonnes. Dans le cas où les informations entrées sont mauvaises, le client se chargera d'afficher un message d'erreur. Si elles sont bonnes, on redirige l'utilisateur à la vue d'accueil du jeu (voir la section 3.9).

Le prochain cas d'utilisation présente la création d'un profil utilisateur (*register*).

3.2 Créer un profil d'utilisateur

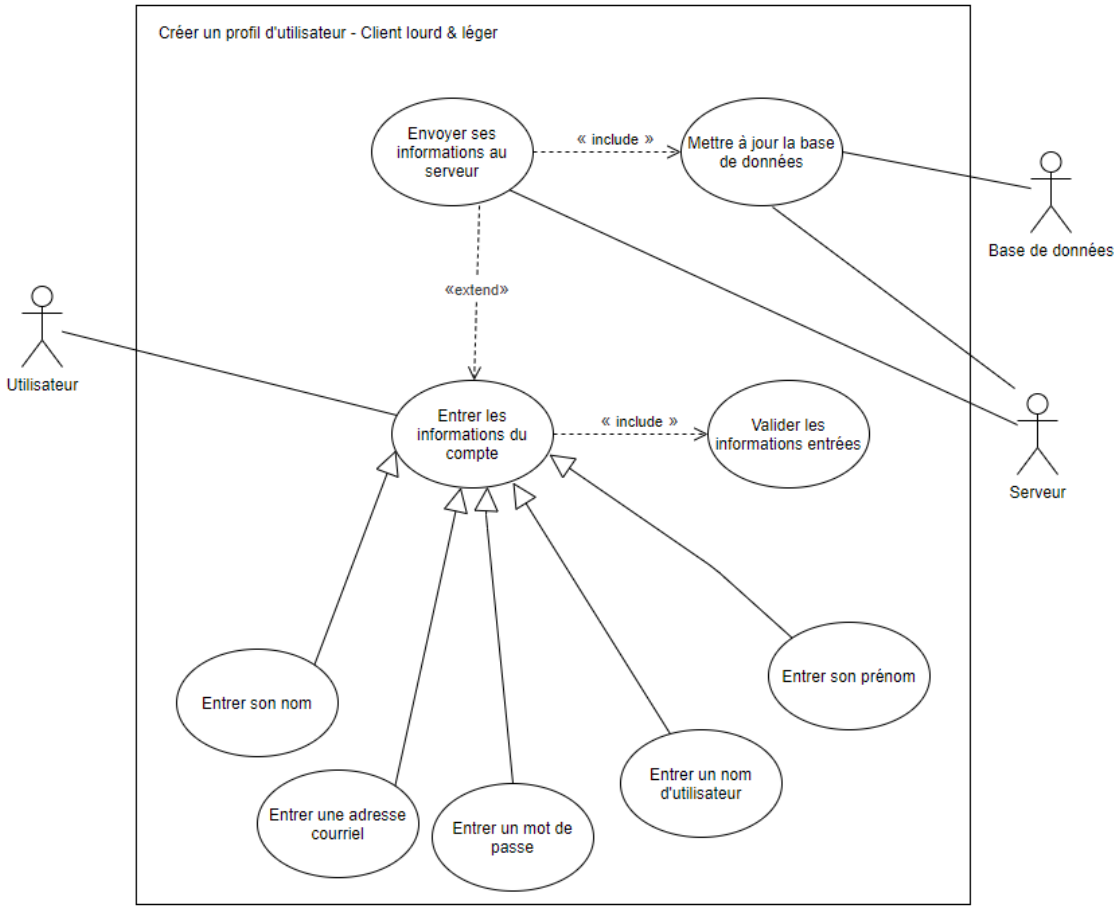

Figure 3.2: Présentation du cas d'utilisation «créer un profil d'utilisateur».

Ici, un utilisateur entre ses informations de compte. Ces informations sont validées par le client. Elles peuvent être envoyées au serveur dans le cas où les informations sont valides. Lorsque les informations sont reçues, le serveur demande à la base de données de mettre à jour sa table d'utilisateurs en y ajoutant le nouveau compte.

3.3 Clavarder

Le cas d'utilisation « clavarder » varie en fonction de l'application bureau (client lourd) et de l'application mobile (client léger).

3.3.1 Client lourd

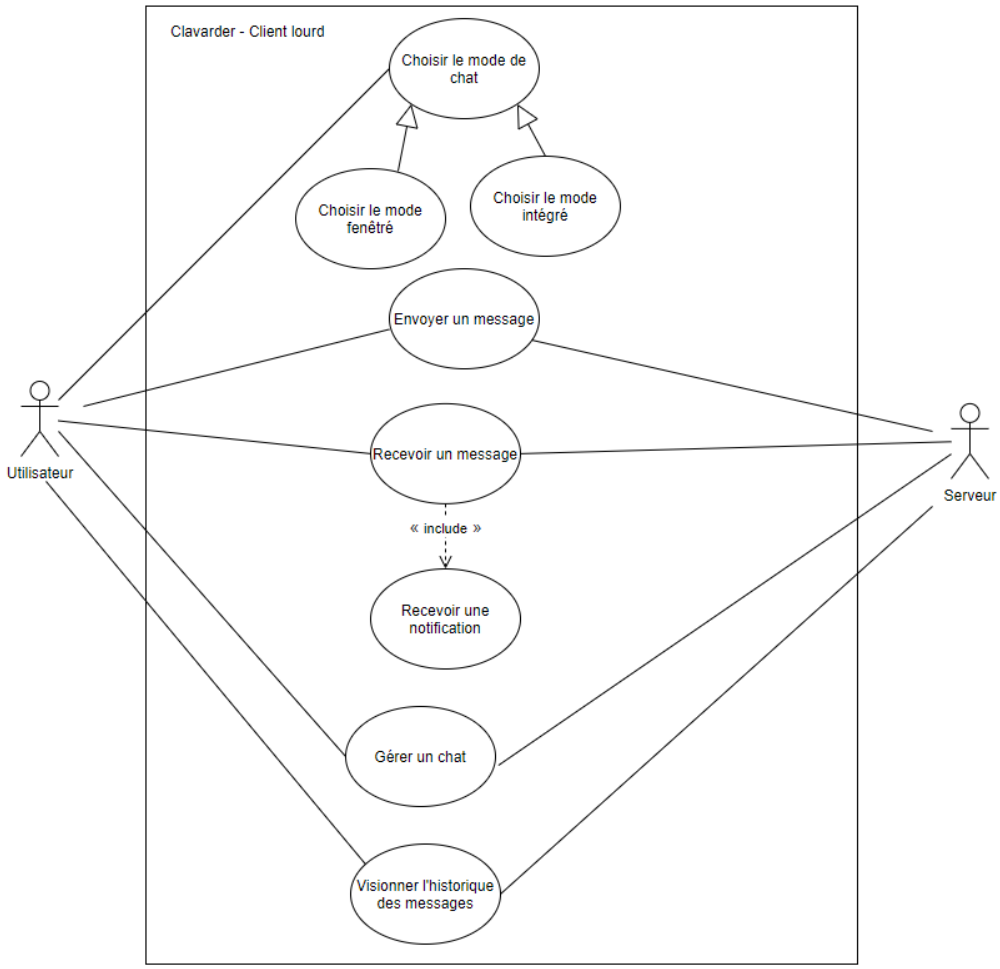

Figure 3.3.1: Présentation du cas d'utilisation «clavarder» pour le client lourd.

Pour clavarder, l'utilisateur a à sa disposition plusieurs fonctionnalités. Entre autres, il peut envoyer et recevoir un message. Lors de la réception d'un message, un effet visuel doit être présent pour notifier l'utilisateur. L'utilisateur de l'application du client lourd peut aussi choisir le mode de fenêtre de son *chat*. Il a le choix entre le mode fenêtré et le mode intégré. De plus, il a la possibilité de visionner l'historique des messages des canaux de discussion. Pour ce faire, le serveur lui envoie l'historique par protocole REST. Finalement, l'utilisateur peut gérer un *chat*. Pour rendre le diagramme de cas d'utilisation plus lisible, un diagramme plus détaillé du cas d'utilisation «Gérer un chat» a été conçu plus bas (voir la section 3.4).

3.3.2 Client léger

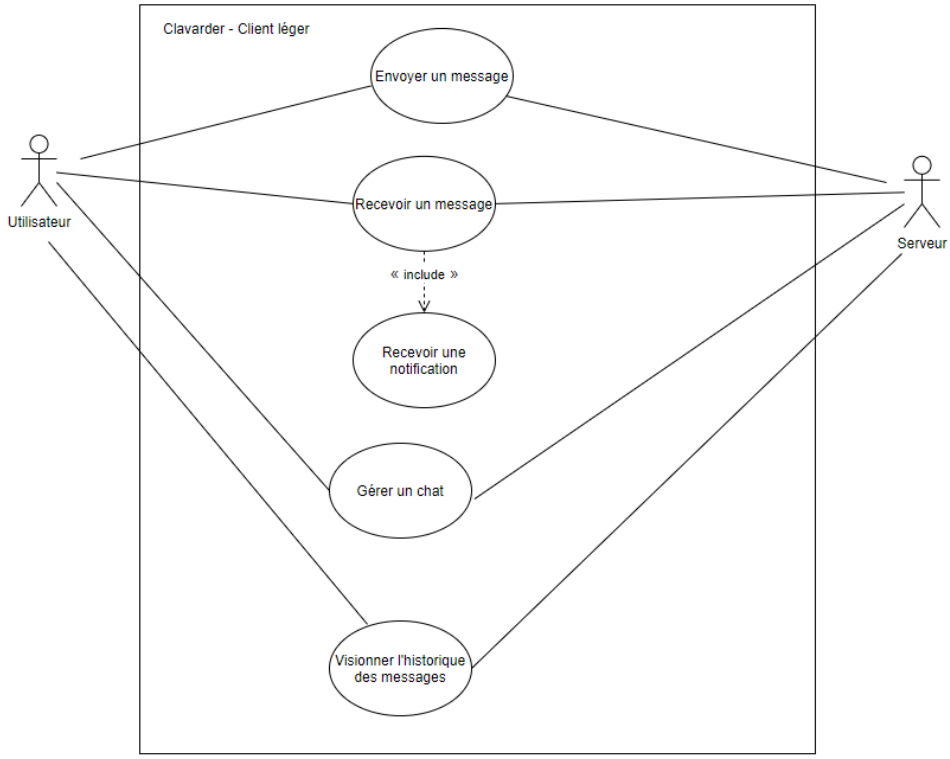

Figure 3.3.2: Présentation du cas d'utilisation «clavarder» pour le client léger.

Le cas d'utilisation du clavardage pour le client léger est identique à celui du client lourd. Par contre, l'utilisateur du client léger n'a pas la possibilité de changer le mode de visionnement du *chat* (mode fenêtré et intégré). Un diagramme plus détaillé du cas d'utilisation «Gérer un chat» a été conçu plus bas (voir la section 3.4).

3.4 Gérer un chat

Il est à noter que ce cas d'utilisation est un sous-cas d'utilisation du diagramme «clavarder» (voir section 3.3). Pour mieux illustrer comment un utilisateur peut gérer un *chat*, le diagramme de cas d'utilisation suivant est présenté.

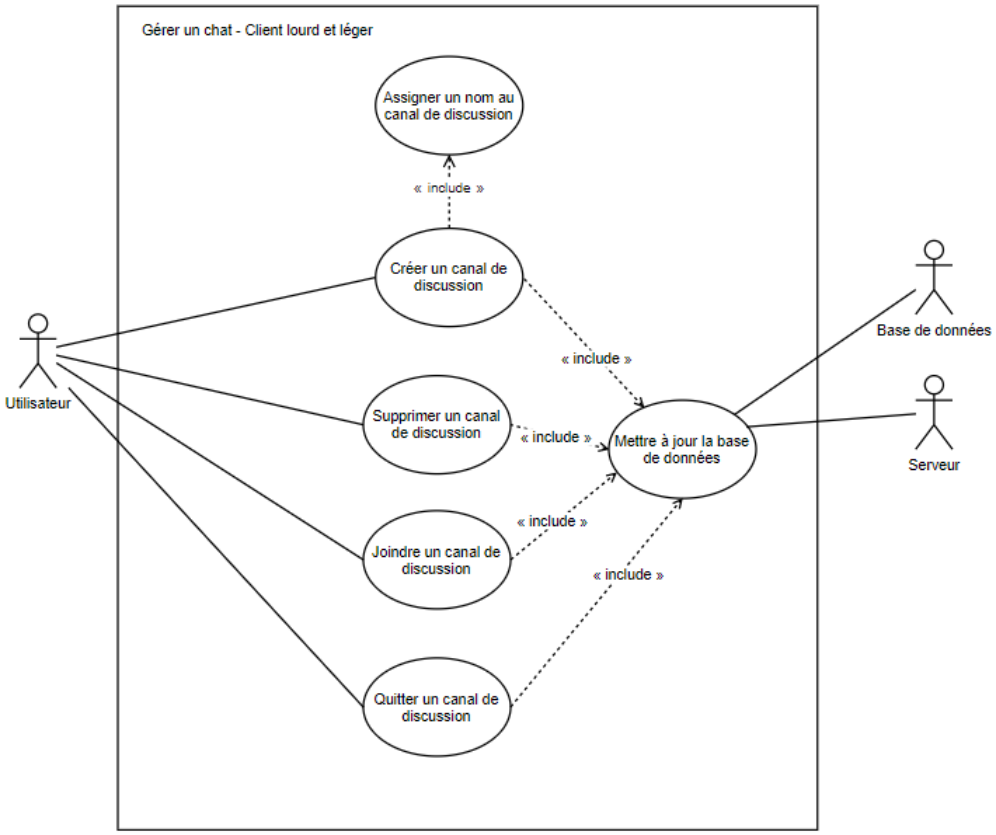

Figure 3.4: Présentation du cas d'utilisation «gérer un *chat*».

Un utilisateur peut créer un canal de discussion, en supprimer un, en joindre un et il peut quitter un canal de discussion. S'il crée un canal de discussion, il devra préciser un nom qui lui sera assigné. Pour effectuer toutes ces actions, le serveur doit demander à la base de données de mettre à jour la table du canal de discussion modifié.

3.5 Créer un jeu

Créer un jeu est un cas d'utilisation qui varie en fonction de la plateforme de l'application.

3.5.1 Client lourd

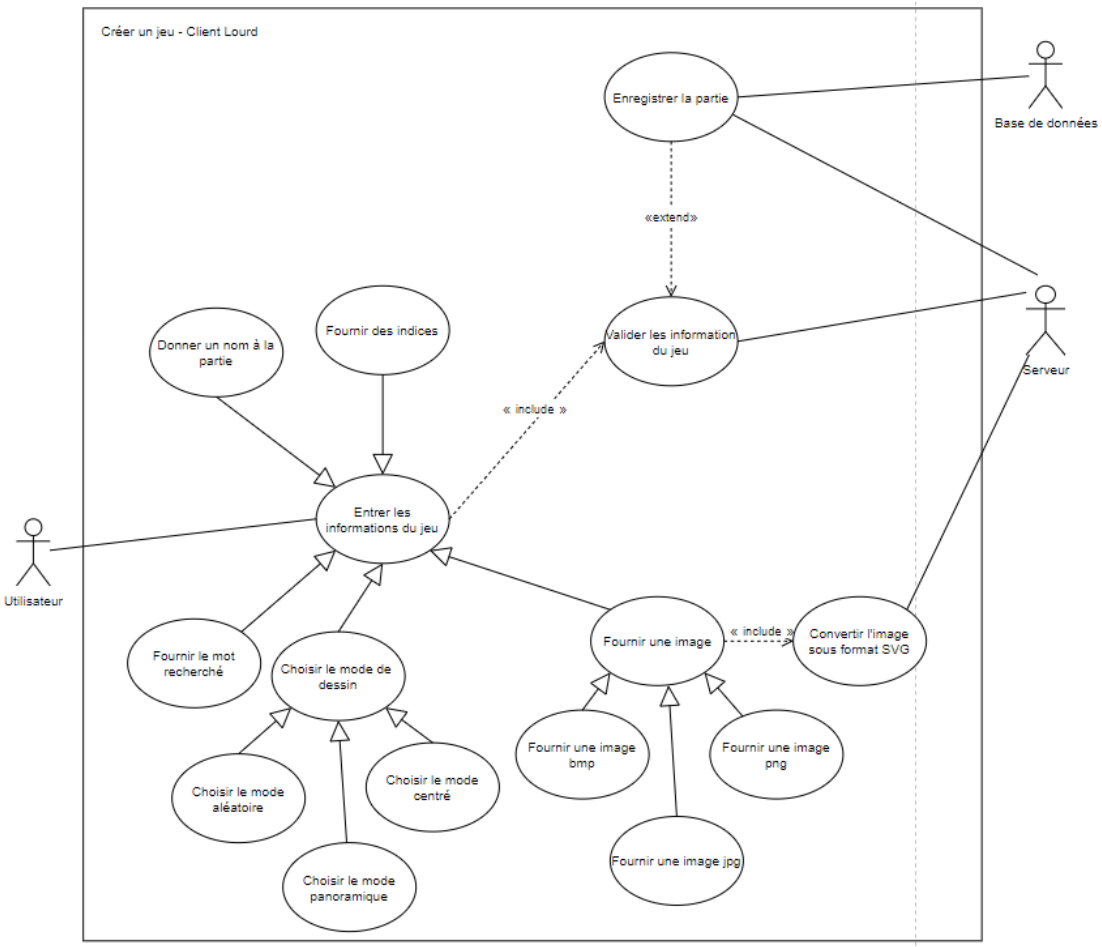

Figure 3.5.1: Présentation du cas d'utilisation «créer un jeu» pour le client lourd.

La création d'un jeu est possible en entrant d'abord toutes les informations nécessaires. Il faut alors entrer le mot à deviner du jeu, les indices, le nom du jeu, le mode de dessin du jeu et l'image. Ayant choisi la création assistée I d'un jeu, l'utilisateur ne peut pas dessiner lui-même l'image du jeu. Il doit plutôt téléverser une image sous format BMP, JPG ou PNG. Cette image doit ensuite être convertie sous format SVG par le serveur (à l'aide de Potrace). Le mode de dessin du jeu peut être le mode aléatoire, panoramique ou le mode centré. Après avoir entré toutes les informations, le client ainsi que le serveur se chargent de valider les informations. Si les informations sont valides, le serveur enregistrera le jeu dans la base de données.

3.5.2 Client léger

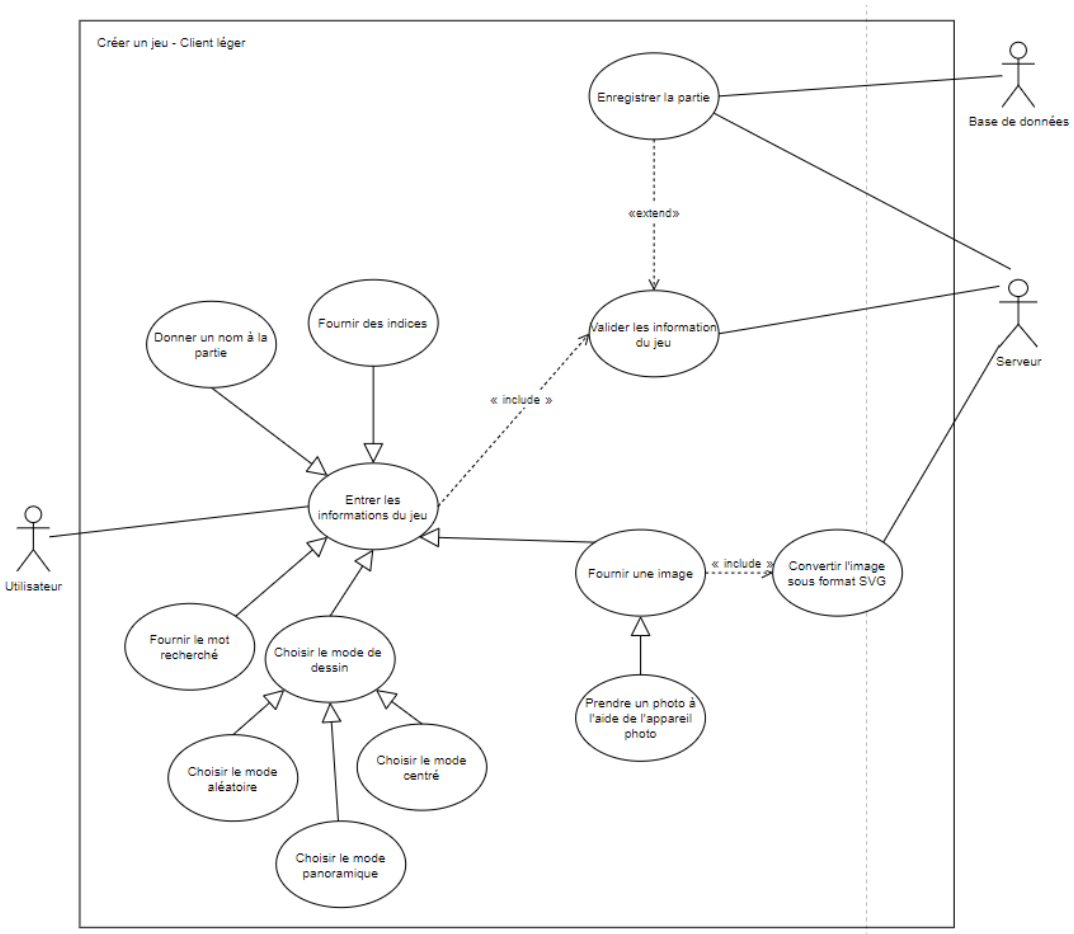

Figure 3.5.2: Présentation du cas d'utilisation «créer un jeu» pour le client léger.

Ce cas d'utilisation est similaire à celui du client lourd. Par contre, l'image de jeu fournie doit être prise par l'appareil photo de la tablette.

3.6 Administrer l'application

Ce cas d'utilisation est uniquement disponible pour le client lourd. Il permet à un administrateur d'effectuer plusieurs tâches.

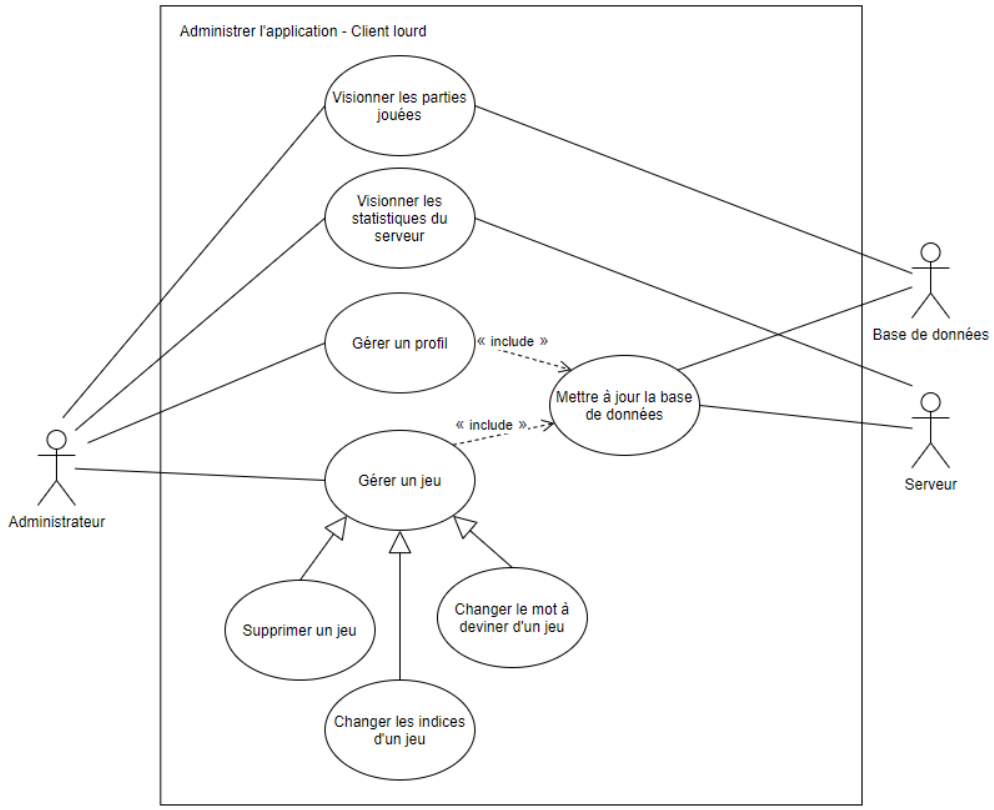

Figure 3.6: Présentation du cas d'utilisation «administrer l'application» pour le client lourd.

Ici, l'administrateur peut visionner les parties enregistrées dans la base de données. Il peut aussi visionner les statistiques du serveur, comme le nombre d'utilisateurs connectés, le temps de disponibilité du serveur, le taux d'utilisation de la mémoire, etc. Ensuite, il peut gérer son profil ou les profils d'autres utilisateurs. Pour mieux comprendre comment un administrateur peut gérer un profil, un diagramme détaillé du cas d'utilisation «Gérer un profil» est illustré à la section 3.7. Finalement, un administrateur peut gérer un jeu. Pour gérer un jeu, l'administrateur peut le supprimer, changer ses indices et changer son mot à deviner. Pour gérer un profil ou un jeu, le serveur et la base de données interagissent pour sauvegarder les modifications.

3.7 Gérer un profil

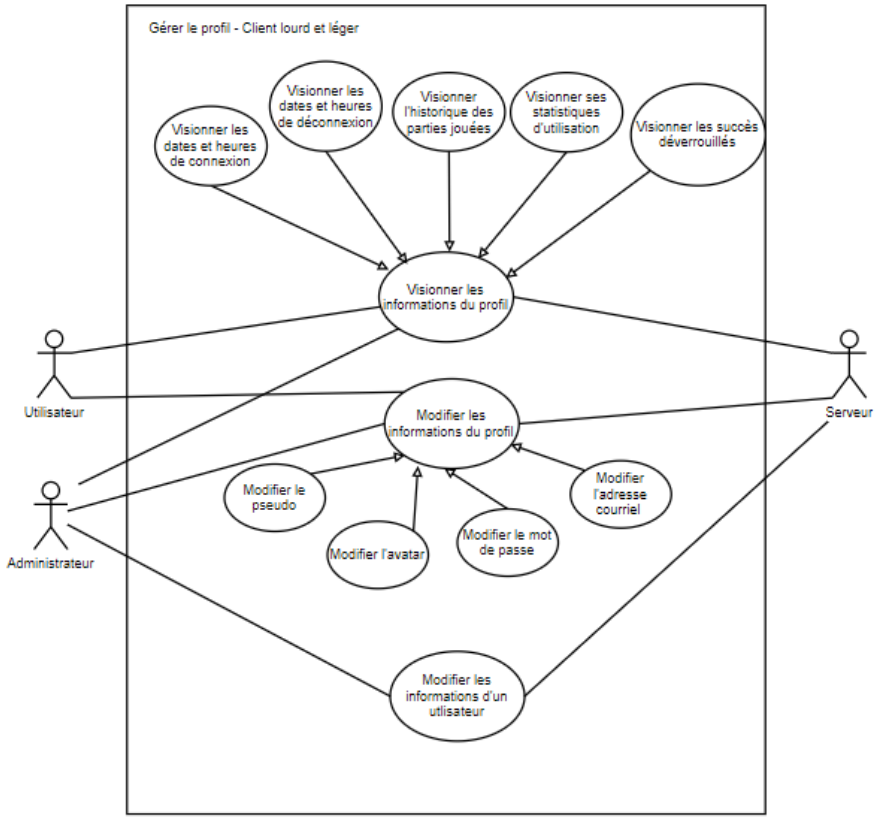

Figure 3.7: Présentation du cas d'utilisation «gérer un profil».

Ce cas d'utilisation est utilisé dans le diagramme du cas d'utilisation «administrer l'application». Il permet à un utilisateur et à un administrateur de modifier ses propres informations (nom d'utilisateur, image d'avatar, mot de passe et adresse courriel). De plus, il permet à un administrateur de modifier les informations d'un autre utilisateur. Ensuite, il permet à un utilisateur et à un administrateur de visionner les informations de leur profil. Ces informations sont les suivantes: les dates et les heures de connexion, les dates et les heures de déconnexion, l'historique des parties jouées, les statistiques d'utilisation et les succès déverrouillés.

3.8 Créer une partie (lobby)

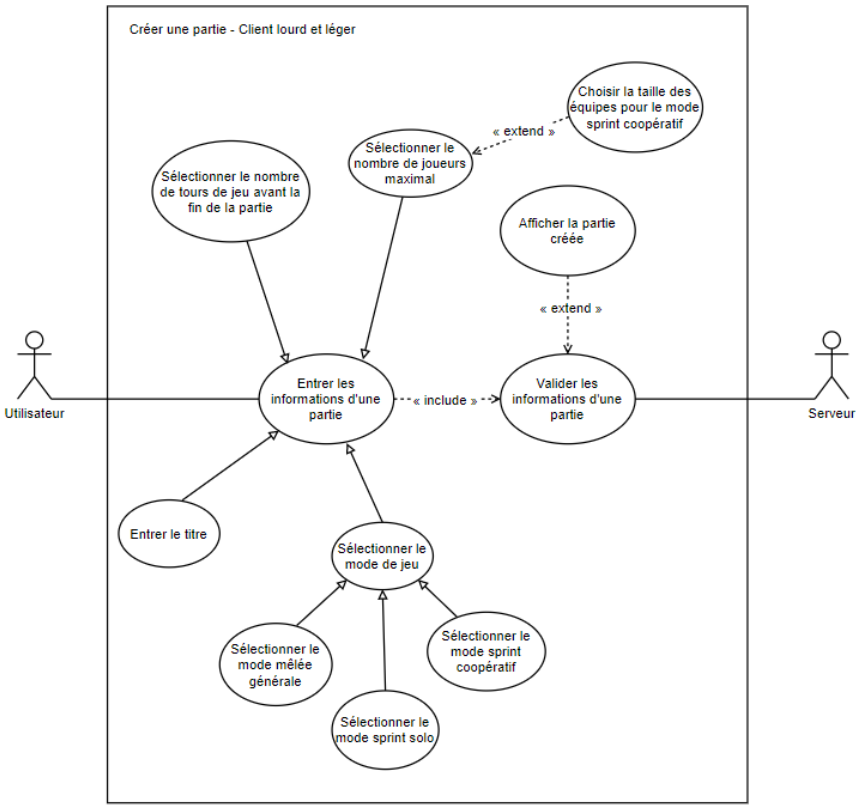

Figure 3.8: Présentation du cas d'utilisation «créer une partie».

Pour créer une partie, un utilisateur devra entrer les informations importantes de la partie. Il entrera alors le titre, le mode de jeu, le nombre de tours et le nombre de joueurs maximum. Si le mode de jeu est celui du sprint coopératif, il faudra également spécifier la taille des équipes. Les informations sont finalement validées par le serveur. Si les informations sont valides, la partie sera affichée par le client lourd et par le client léger.

3.9 Interagir avec la vue d'accueil

La vue d'accueil est la vue affichée après s'être connectée à l'application. Dépendamment de la version bureau ou de la version mobile de l'application, les fonctionnalités disponibles pour le cas d'utilisation «interagir avec la vue d'accueil» seront légèrement différentes.

3.9.1 Client lourd

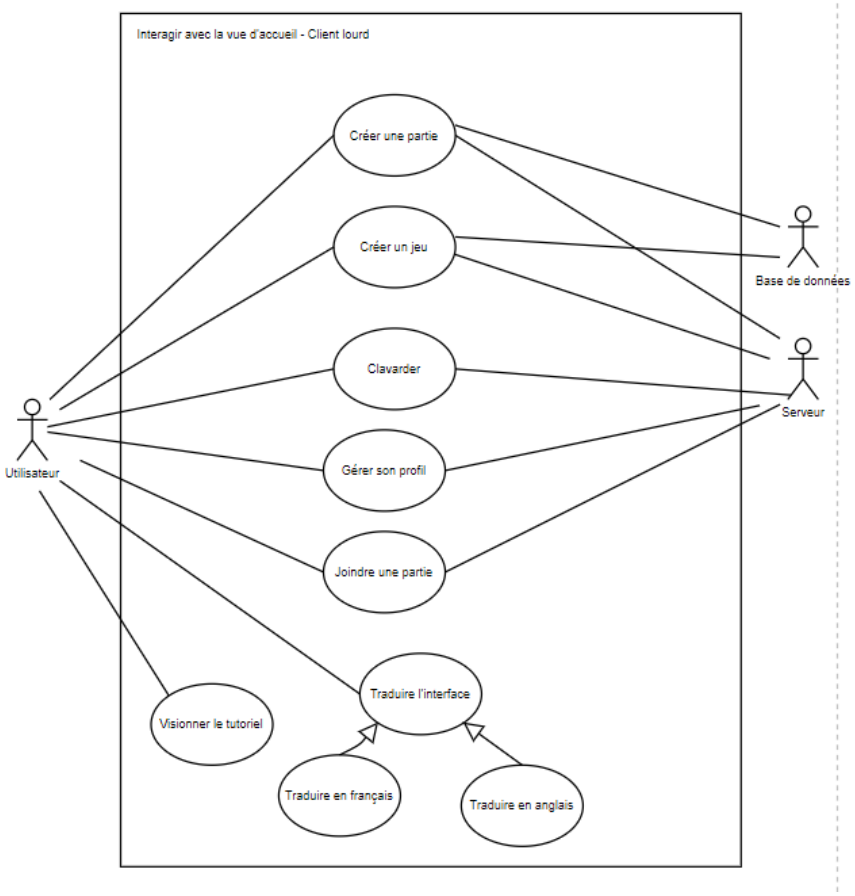

Figure 3.9.1: Présentation du cas d'utilisation «interagir avec la vue d'accueil» pour le client lourd.

Ce cas d'utilisation permet à un utilisateur d'effectuer plusieurs cas d'utilisation déjà définis plus haut. Par exemple, les bulles «créer une partie», «créer un jeu», «clavarder» et «gérer son profil» sont tous des cas d'utilisation déjà expliqués. À ces cas d'utilisation viennent s'ajouter les fonctionnalités de joindre une partie, de visionner le tutoriel et de traduire l'interface. L'interface peut être traduite en anglais ou en français.

3.9.2 Client léger

Figure 3.9.2: Présentation du cas d'utilisation «interagir avec la vue d'accueil» pour le client léger.

Ce diagramme est identique à celui plus haut. Par contre, un utilisateur du client léger peut également choisir le thème de couleur de l'application.

3.10 Jouer à une partie FFA

Figure 3.10: Présentation du cas d'utilisation «jouer à une partie FFA».

Pour le mode de partie mêlée générale, un utilisateur peut demander un indice à un joueur virtuel. Ensuite, il peut tenter de deviner un mot, qui sera par la suite validé par le serveur. Si le mot est valide, l'utilisateur gagnera des points. En tout temps, l'utilisateur peut clavarder, tel que défini à la section 3.3. Si c'est le tour de l'utilisateur à dessiner, il peut utiliser les outils de dessin disponibles pour tracer son image. Si un autre utilisateur devine le mot dessiné, le dessinateur gagnera des points.

3.11 Jouer à une partie en mode solo

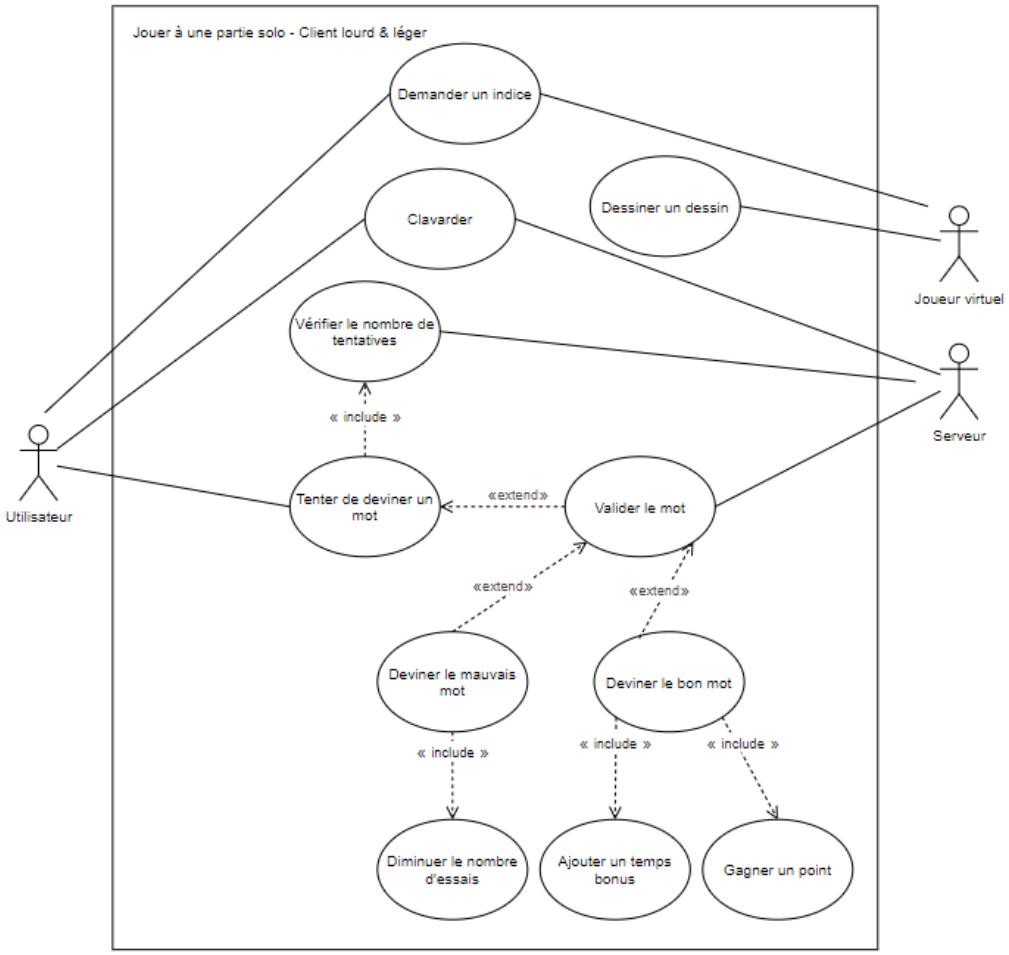

Figure 3.11: Présentation du cas d'utilisation «jouer à une partie en mode solo».

Comme pour une partie en mode FFA, un utilisateur peut demander un indice et clavarder. Par contre, contrairement au mode de mêlée générale, le dessinateur est un joueur virtuel. De plus, un utilisateur se voit attribuer un nombre maximal de tentatives pour deviner un mot. Lors d'une tentative, le serveur doit alors vérifier que l'utilisateur a toujours le droit de deviner un mot. S'il a le droit, le serveur s'assurera de valider le mot. Le mot peut soit être valide ou soit être invalide. Dans le cas où le mot envoyé est valide, un temps bonus et un point sont alloués à l'utilisateur. Si le mot est invalide, le nombre d'essais de l'utilisateur sera incrémenté.

3.12 Jouer à une partie en mode coopératif

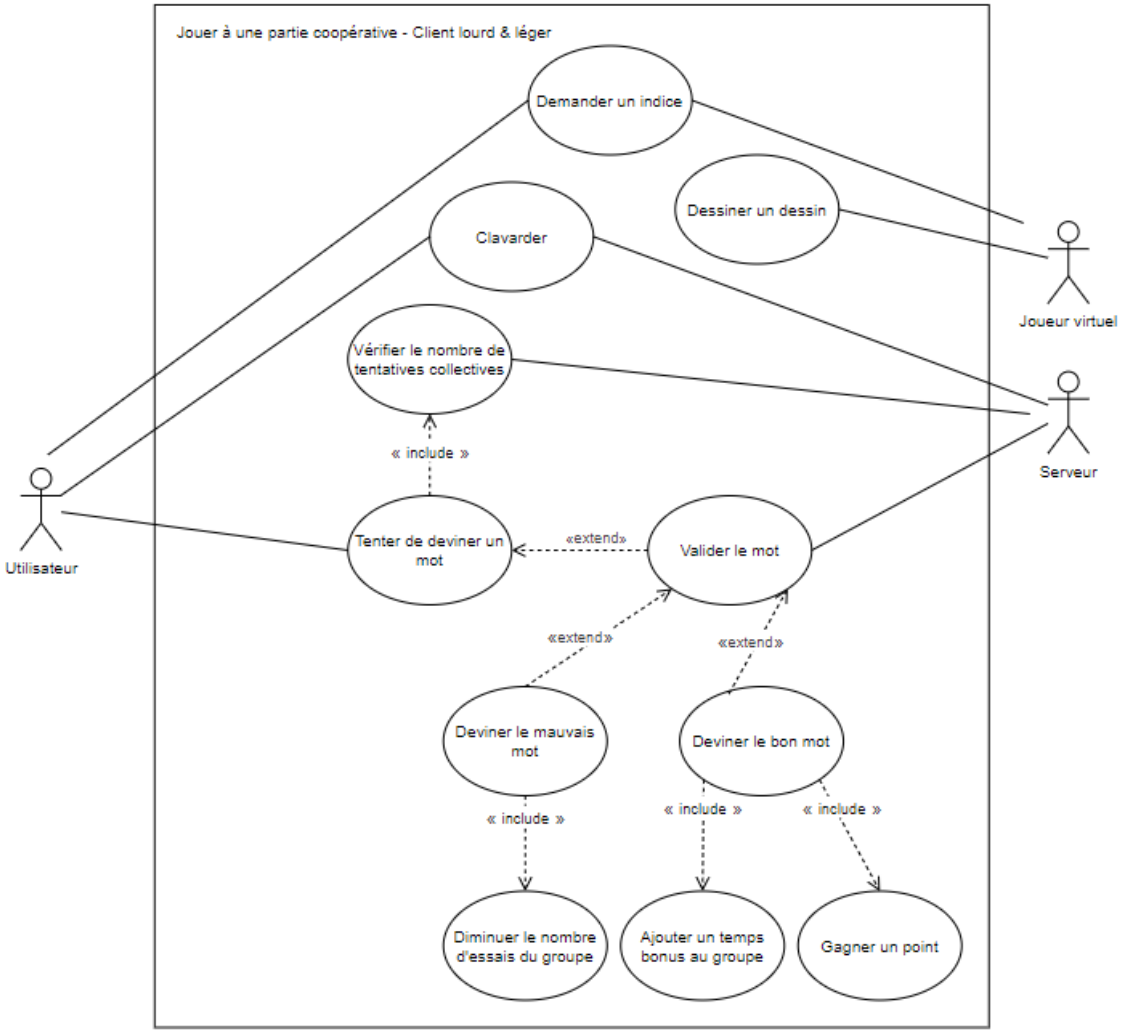

Figure 3.12: Présentation du cas d'utilisation «jouer à une partie en mode coopératif».

Ce cas d'utilisation est identique à celui d'une partie en mode solo. Par contre, il intègre le concept d'équipes. Le nombre de tentatives maximum est maintenant partagé entre tous les membres d'une équipe. Cela veut ainsi dire que si un utilisateur devine le mauvais mot, toute l'équipe sera pénalisée et perdra une tentative de deviner le mot. De la même manière, si un utilisateur devine le bon mot, toute l'équipe gagnera et point et se verra allouer un temps supplémentaire.

4. Vue logique

4.1 Client lourd

ClientLourd
Ce paquetage a pour responsabilité d'assurer un couplage faible entre les différents sous-paquetages en plus d'assurer leur bonne intégration

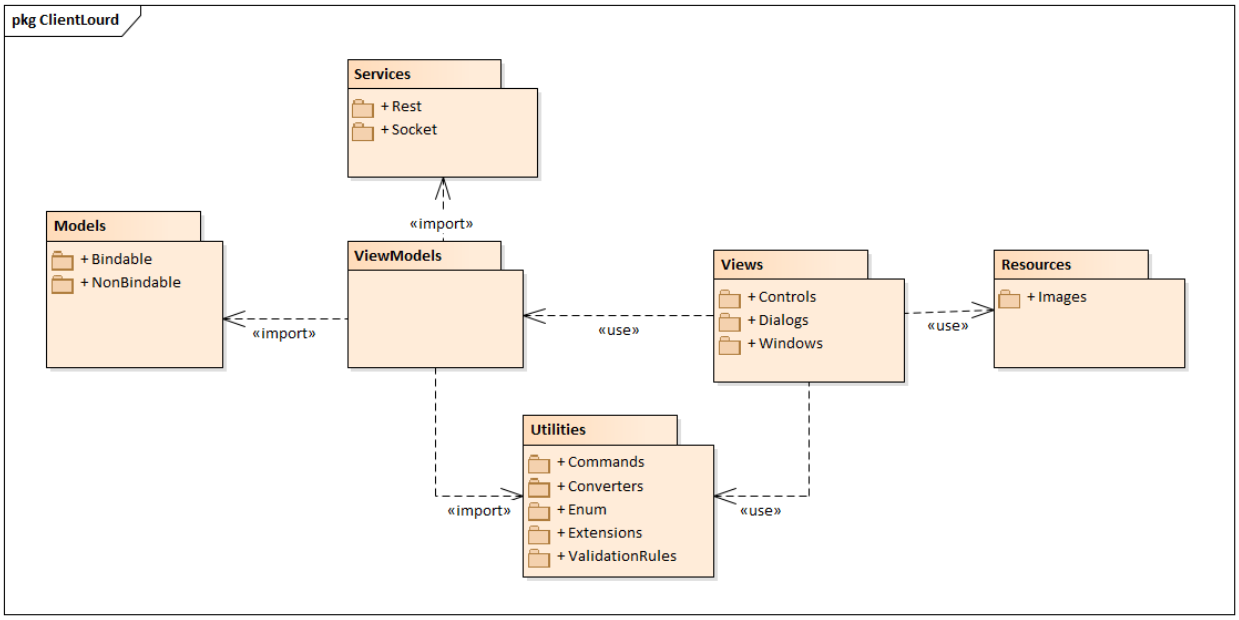

Figure 4.1.1: Diagramme de paquetage ClientLourd

4.1.1 Models

ClientLourd::Models
Ce paquetage a pour responsabilité d'offrir au paquetage ViewModels des classes pouvant être instanciée. Ce paquetage ne devrait pas contenir de logique.

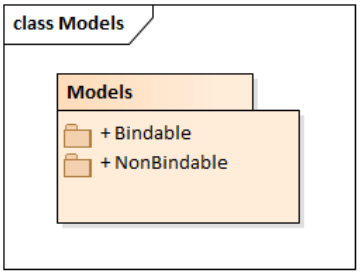

Figure 4.1.1.1: Diagramme de paquetage ClientLourd::Models

ClientLourd::Models::Bindable
Ce paquetage a pour responsabilité d'offrir l'accès aux Classes implémentant l'interface INotifyPropertyChanged. Ces classes peuvent être utilisées en binding.

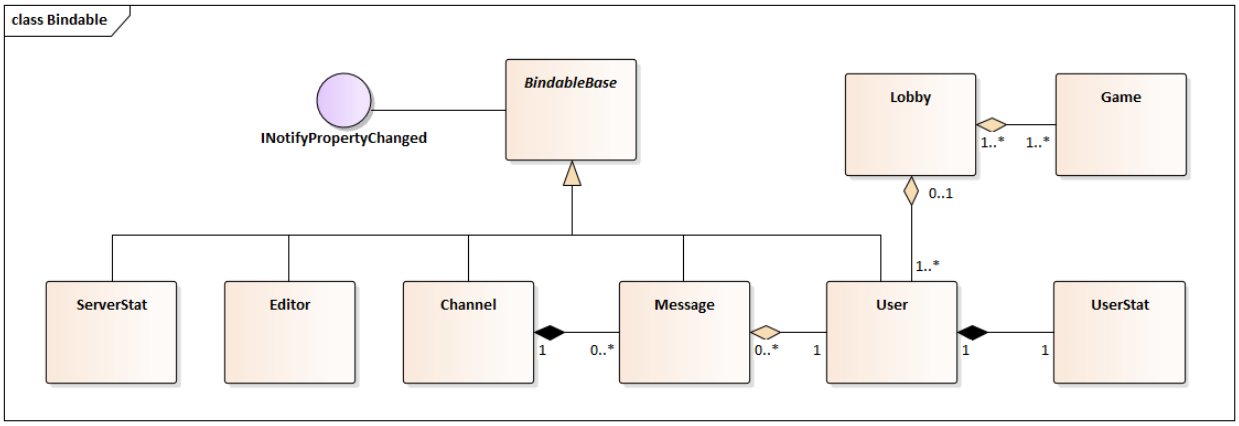

Figure 4.1.1.2: Diagramme de classe ClientLourd::Models::Bindable

ClientLourd::Models::NonBindable
Ce paquetage a pour responsabilité d'offrir l'accès aux Classes générales de l'application qui ne seront pas utilisées dans du binding

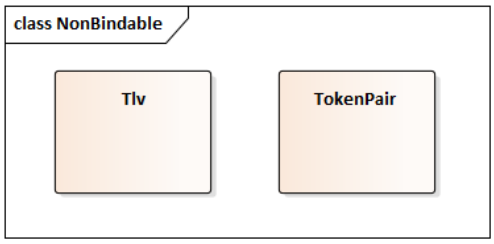

Figure 4.1.1.3: Diagramme de classe ClientLourd::Models::NonBindable

4.1.2 Views

ClientLourd::Views
Ce paquetage a pour responsabilité d'offrir une représentation du paquetage ViewModels aux utilisateurs en plus d'offrir une interaction avec les informations dans ce paquetage via le binding

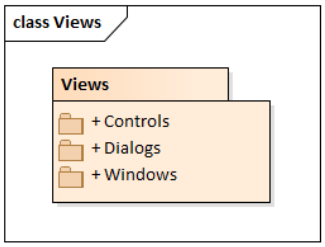

Figure 4.1.2.1: Diagramme de paquetage ClientLourd::Views

ClientLourd::Views::Controls
Ce paquetage a pour responsabilité d'offrir l'accès aux différents contrôles

utilisateurs (UserControl) personnalisés

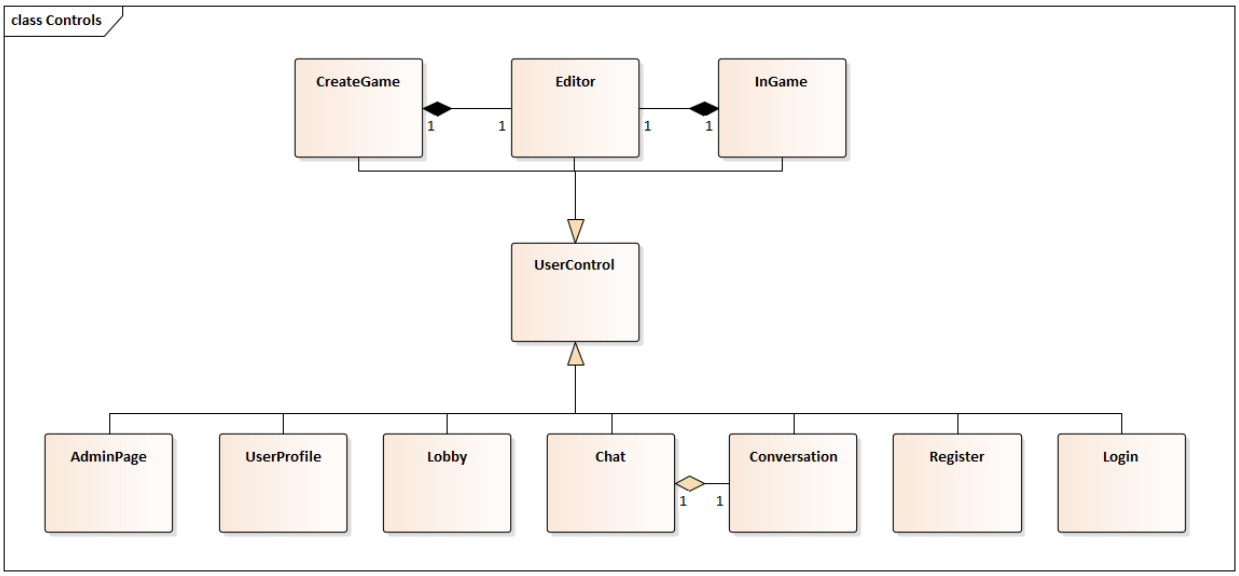

Figure 4.1.2.2: Diagramme de classe ClientLourd::Views::Controls

ClientLourd::Views::Dialogs
Ce paquetage a pour responsabilité d'offrir l'accès aux différents contrôles utilisateur (UserControl) à utiliser en tant que dialogue

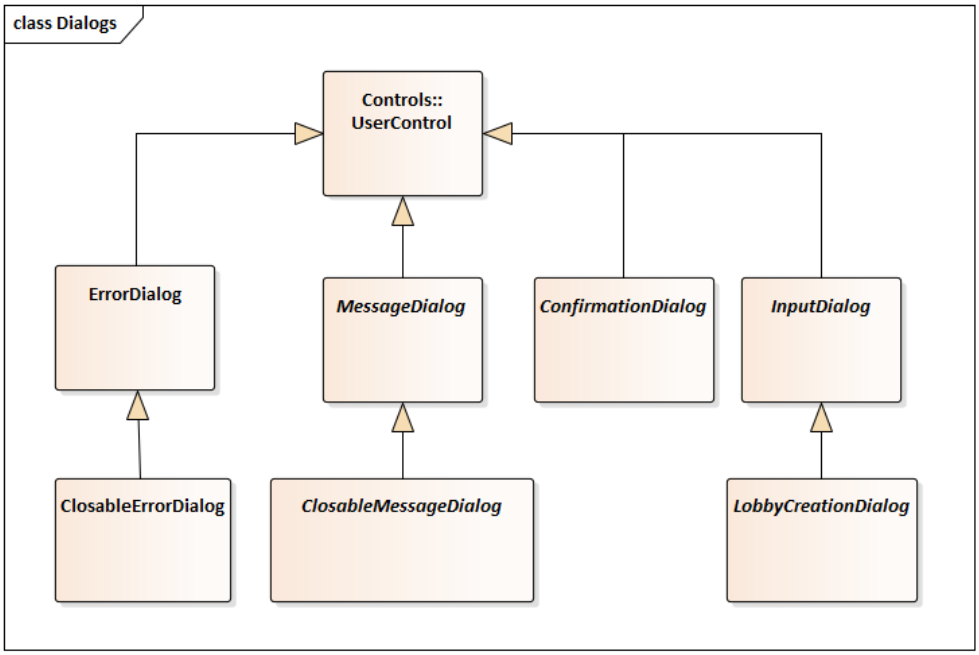

Figure 4.1.2.3: Diagramme de classe ClientLourd::Views::Dialogs

ClientLourd::Views::Windows
Ce paquetage a pour responsabilité d'offrir l'accès aux différentes fenêtres (Window) personnalisées

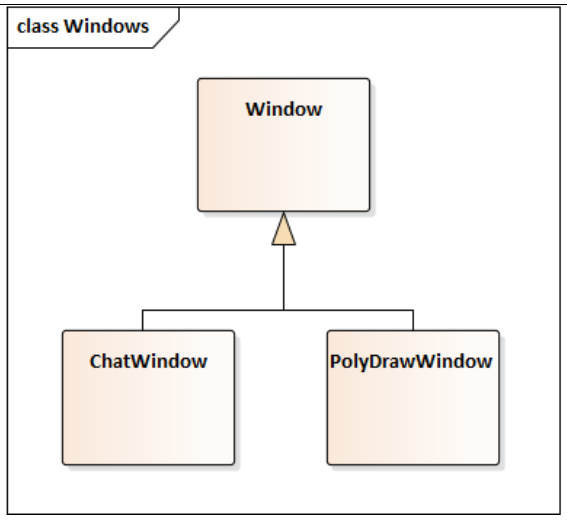

Figure 4.1.2.4: Diagramme de classe ClientLourd::Views::Windows

4.1.3 ViewModels

ClientLourd::ViewModels
Ce paquetage a pour responsabilité d'offrir au paquetage Views une abstraction du paquetage Models afin d'utiliser le binding. La majorité des ClientLourd::Views::Controls auront leur propre view models

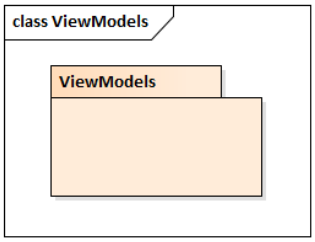

Figure 4.1.3: Diagramme de paquetage ClientLourd::ViewsModels

4.1.4 Services

ClientLourd::Services
Ce paquetage a pour responsabilité de contenir tous les services logiques indépendants du paquetage Views et ViewModels

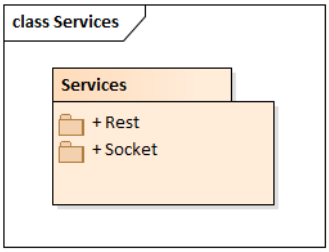

Figure 4.1.4.1: Diagramme de paquetage ClientLourd::Services

ClientLourd::Services:Socket
Ce paquetage a pour responsabilité d'implémenter les classes et les événements pour la communication en temps réel

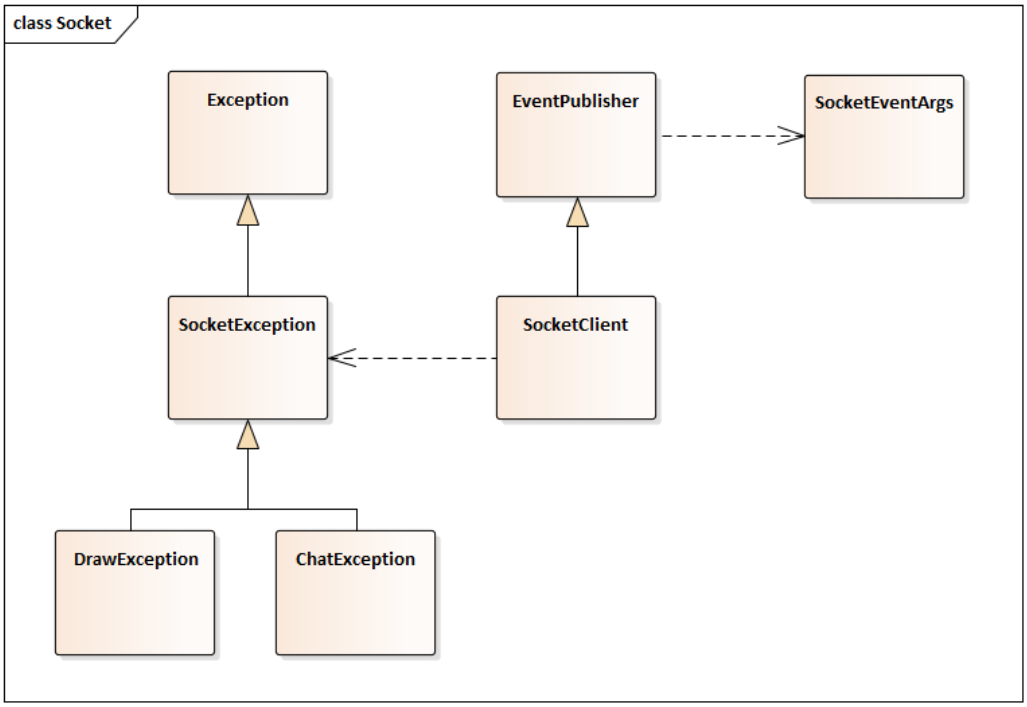

Figure 4.1.4.2: Diagramme de classe ClientLourd::Services::Socket

ClientLourd:Services:Rest
Ce paquetage a pour responsabilité d'implémenter les classes et les événements pour la communication REST.

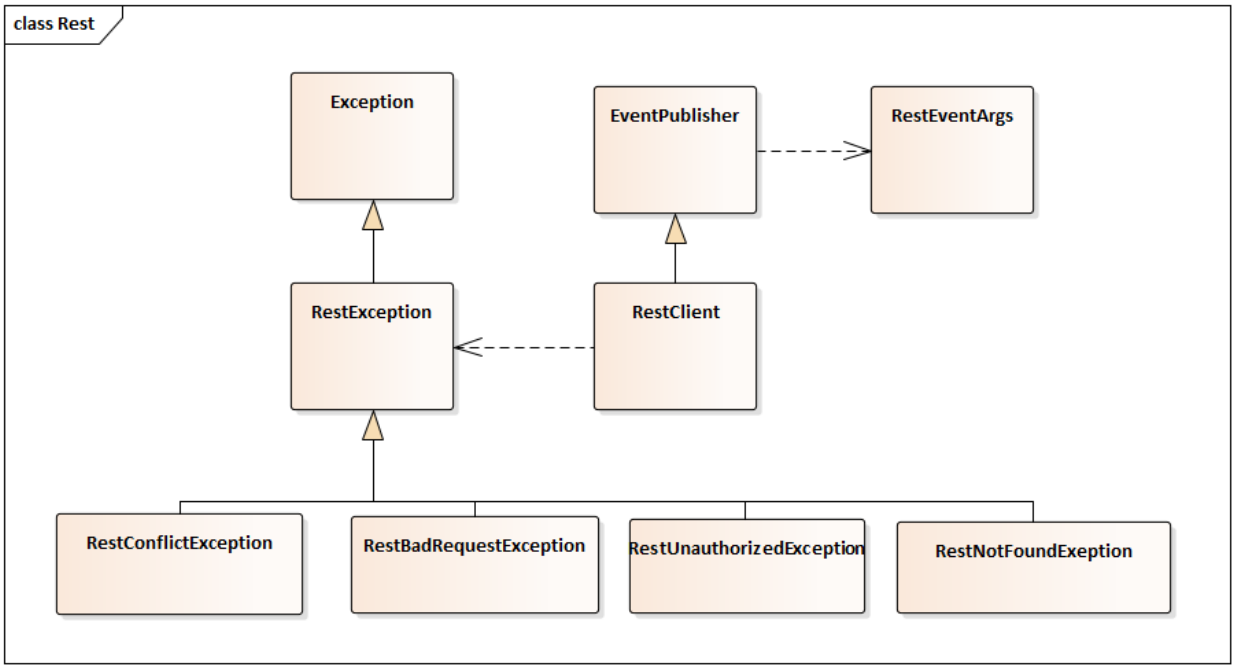

Figure 4.1.4.3: Diagramme de classe ClientLourd::Services::Rest

4.1.5 Utilities

ClientLourd:Utilities
Ce paquetage a pour responsabilité d'offrir et de regrouper différents outils aux paquetages Views et ViewModels pour simplifier le binding

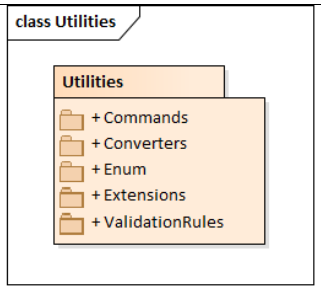

Figure 4.1.5.1: Diagramme de paquetage ClientLourd::Utilities

ClientLourd::Utilities:Enums
Ce paquetage a pour responsabilité d'offrir l'accès aux Enums de l'application

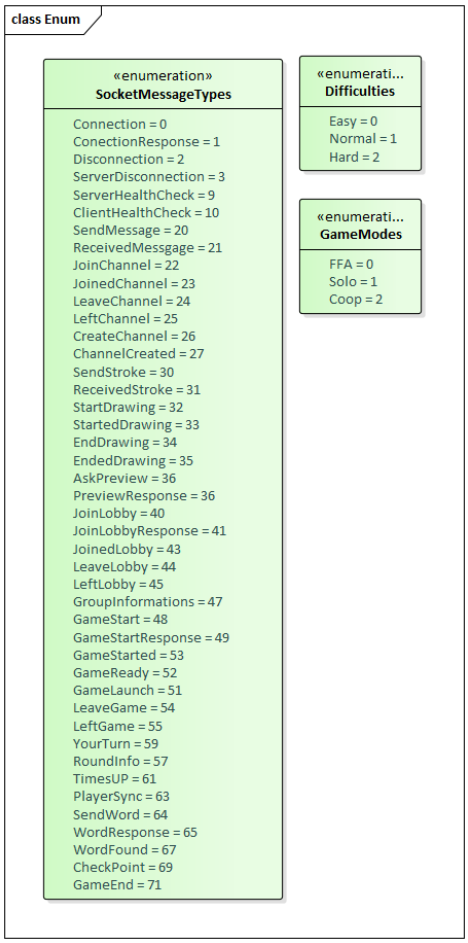

Figure 4.1.5.2: Diagramme de classe ClientLourd::Utilities::Enums

ClientLourd::Utilities::Converters
Ce paquetage a pour responsabilité de contenir l'implémentation des interfaces IValueConverter ainsi que IMultiValueConverter

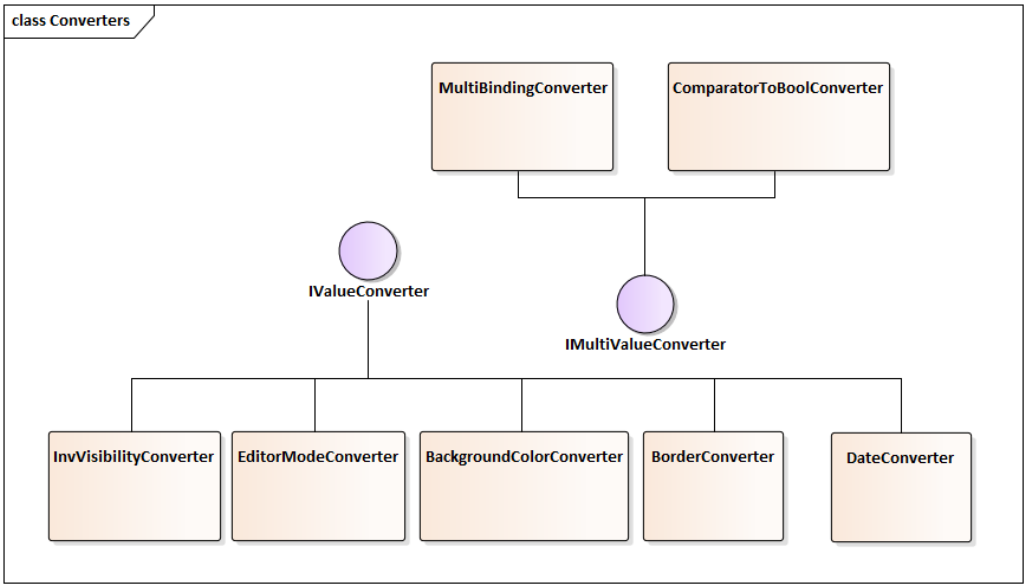

Figure 4.1.5.3: Diagramme de classe ClientLourd::Utilities::Converters

ClientLourd::Utilities::ValidationRules
Ce paquetage a pour responsabilité de contenir l'implémentation des classes ValidationRule

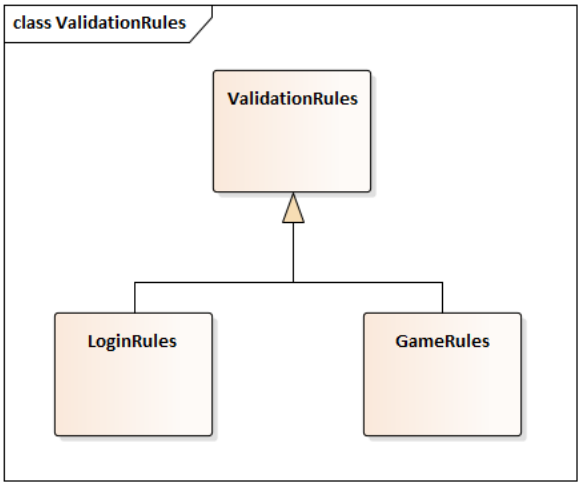

Figure 4.1.5.4: Diagramme de classe ClientLourd::Utilities::ValidationRules

ClientLourd::Utilities::Commands
Ce paquetage a pour responsabilité d'offrir des implémentations personnalisées de l'interface ICommand

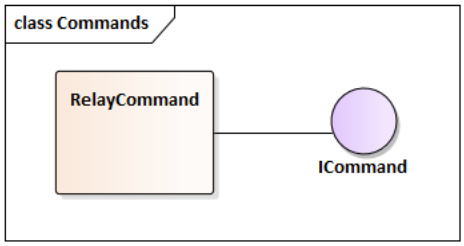

Figure 4.1.5.5: Diagramme de classe ClientLourd::Utilities::Commands

ClientLourd::Utilities::Extensions
Ce paquetage a pour responsabilité de contenir les différentes extensions développées pour les Control XAML existants

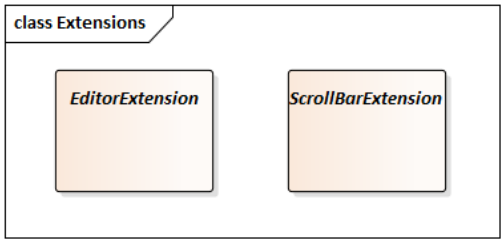

Figure 4.1.5.6: Diagramme de classe ClientLourd::Utilities::Extensions

4.1.6 Resources

ClientLourd::Resources
Ce paquetage a pour responsabilité de contenir les différentes ressources utilisées par les autres paquetages (images, icônes, etc.)

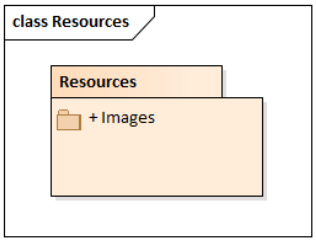

Figure 4.1.6: Diagramme de paquetage ClientLourd::Resources

4.2 Client léger

Une autre méthode de paquetage a été utilisée pour le client léger. Plutôt que de séparer les paquetages selon leur type de composantes logiciel, ils ont été séparés selon les fonctionnalités qu'ils affectent. Cette méthode de paquetage a l'avantage de garder une haute cohésion à l'intérieur des paquetages, mais le désavantage de créer beaucoup plus de paquetage. Ainsi, il y a beaucoup d'éléments dans le diagramme de paquetage de haut niveau. Pour cette raison, certains liens sont omis du diagramme, mais sont mentionnés dans le texte.

ClientLéger
Ce paquetage est le paquetage de haut niveau pour le client léger. Il s'occupe simplement de regrouper tous les autres paquetages utilisés pour le client léger ainsi que certains fichiers de configuration. Tous les paquetages ont une relation "use" avec le paquetage *shared*, mais les traits sur le diagramme sont omis pour fin de clarté.

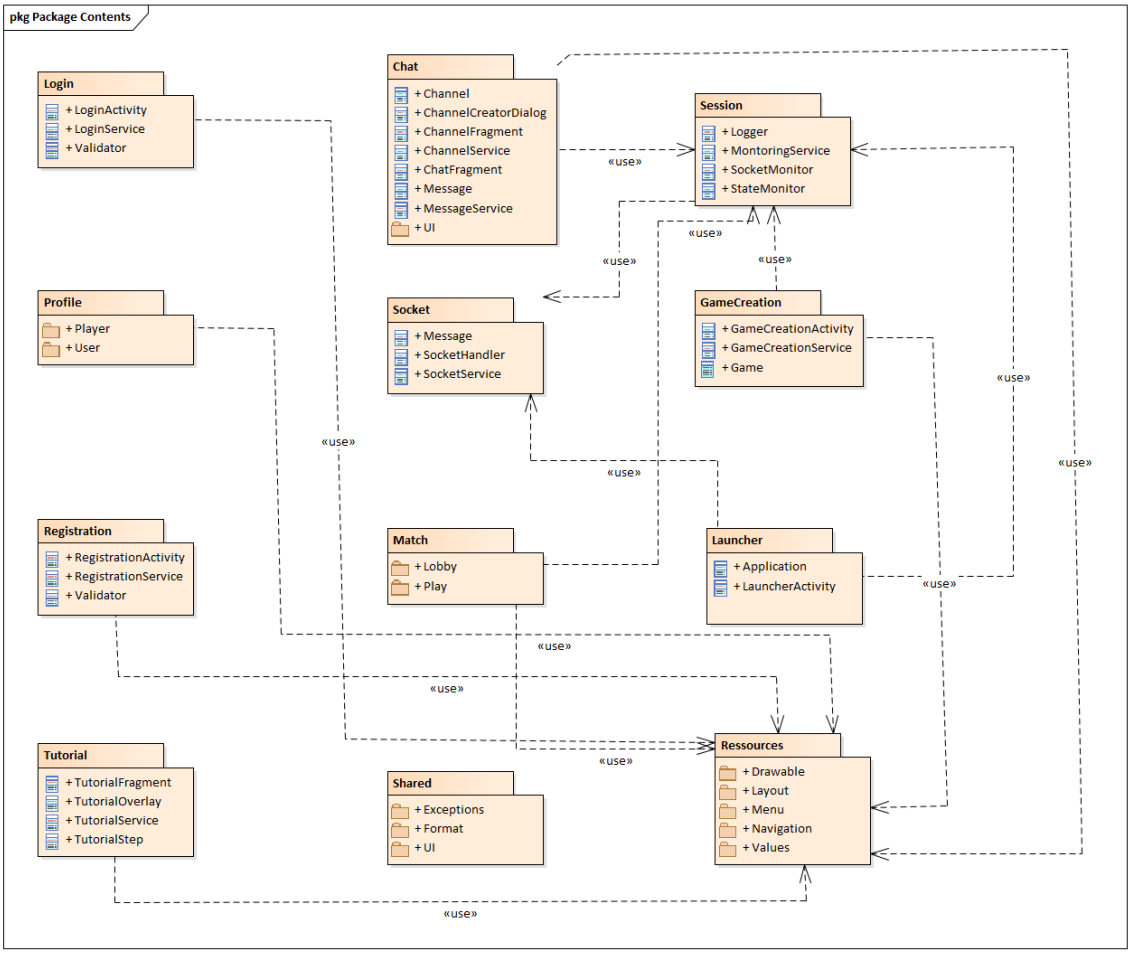

Figure 4.2: Diagramme de paquetage ClientLéger

4.2.1 Chat

ClientLéger::Chat
Ce paquetage regroupe toutes les fonctionnalités pour le clavardage. On y retrouve les fonctionnalités permettant d'écrire et afficher des messages ainsi que joindre, quitter, créer et retirer des canaux de communication.

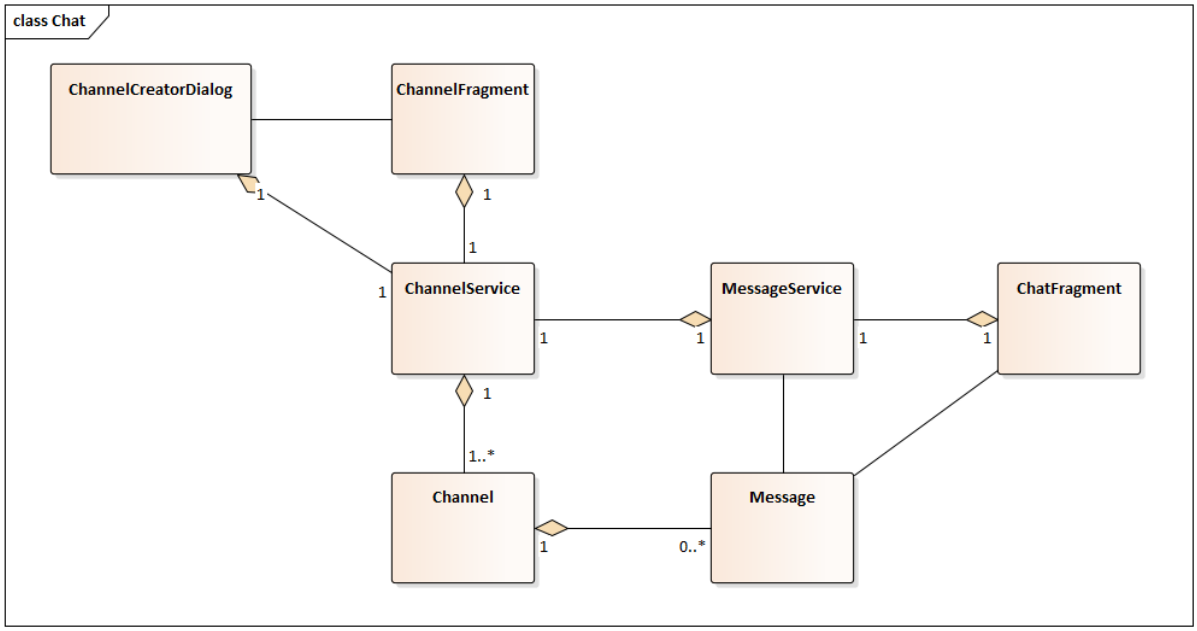

Figure 4.2.1: Diagramme de classe ClientLéger::Chat

4.2.2 UI

ClientLéger::Chat::UI
Ce paquetage regroupe des classes qui s'occupent uniquement de l'interface utilisateur du Chat. On y retrouve les adaptateurs de messages et de canaux.

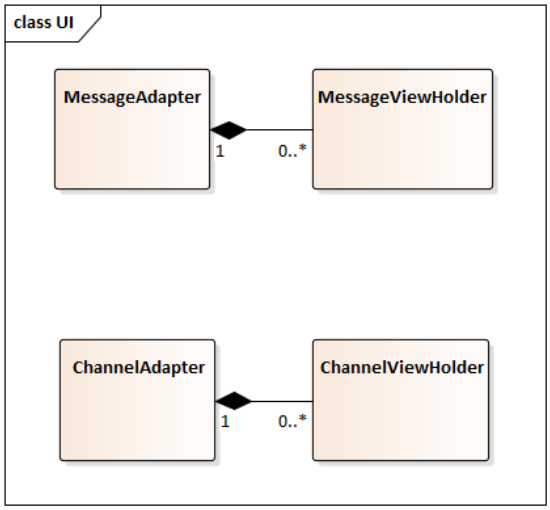

Figure 4.2.2: Diagramme de classe ClientLéger::Chat::UI

4.2.3 Match

ClientLéger::Match
Ce paquetage regroupe les fonctionnalités utilisées pour les parties de jeu. On y retrouve les paquetages du lobby de jeu et pour jouer une partie.

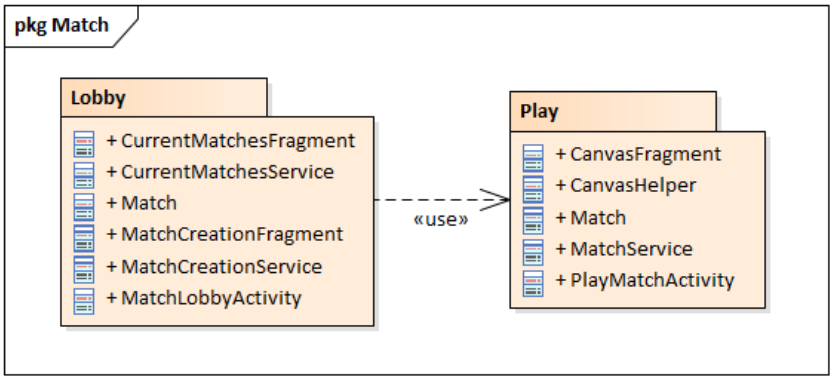

Figure 4.2.3: Diagramme de paquetage ClientLéger::Match

4.2.4 Lobby

ClientLéger::Match::Lobby
Ce paquetage regroupe les fonctionnalités utilisées pour le lobby de jeu. On y retrouve les fonctionnalités pour créer une nouvelle partie, rejoindre une partie en cours, voir les informations sur les parties courantes, etc.

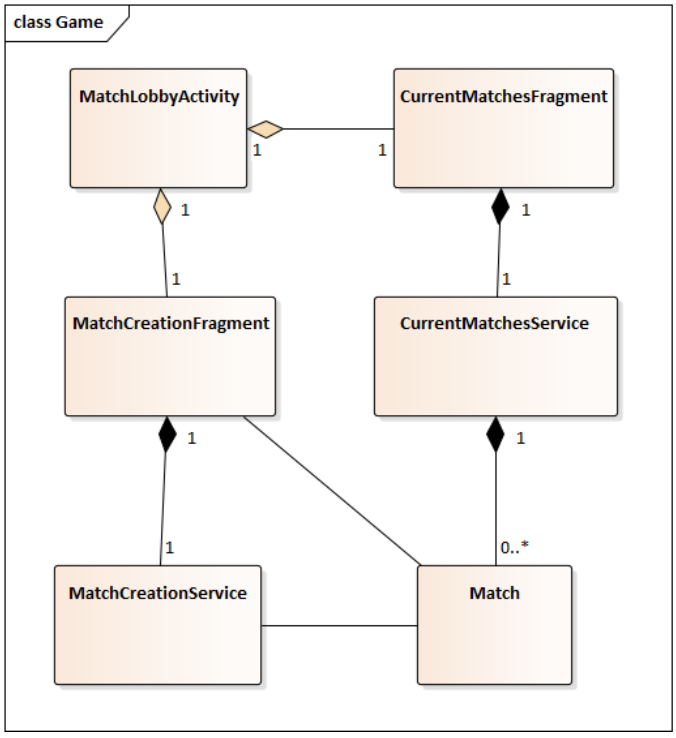

Figure 4.2.4: Diagramme de classe ClientLéger::Match::Lobby

4.2.5 Play

ClientLéger::Match::Play
Ce paquetage regroupe les fonctionnalités utilisées lorsque l'utilisateur joue à une partie. On y retrouve tout ce qui touche l'interface pour dessiner, l'interface pour deviner un mot, le gestionnaire de partie, etc.

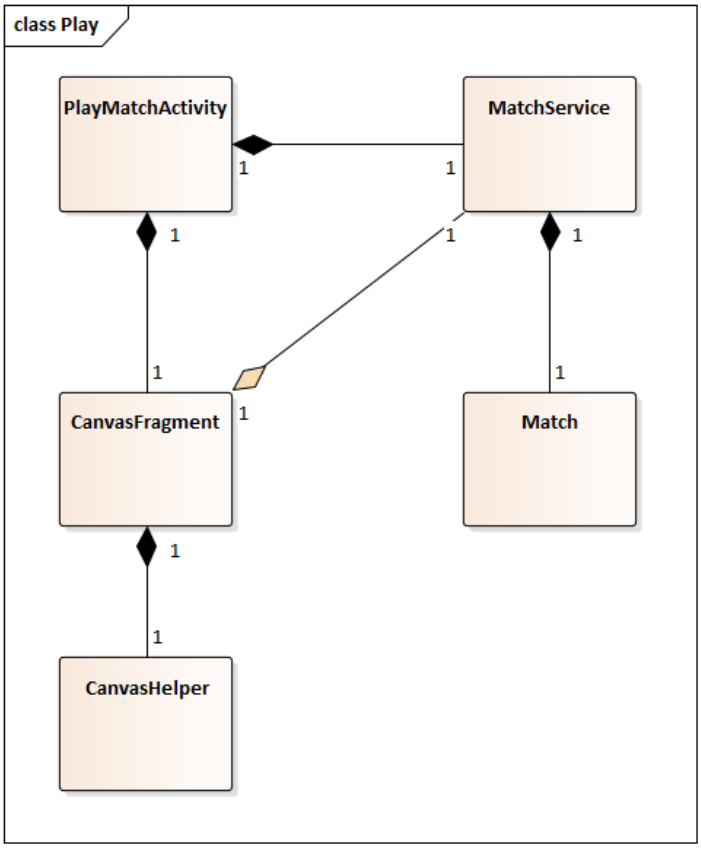

Figure 4.2.5: Diagramme de classe ClientLéger::Match::Play

4.2.6 GameCreation

ClientLéger::GameCreation
Ce paquetage regroupe les fonctionnalités utilisées pour créer un nouveau jeu. On y retrouve les fonctionnalités pour prendre une photo, entrer l'information pour la création d'une partie, etc.

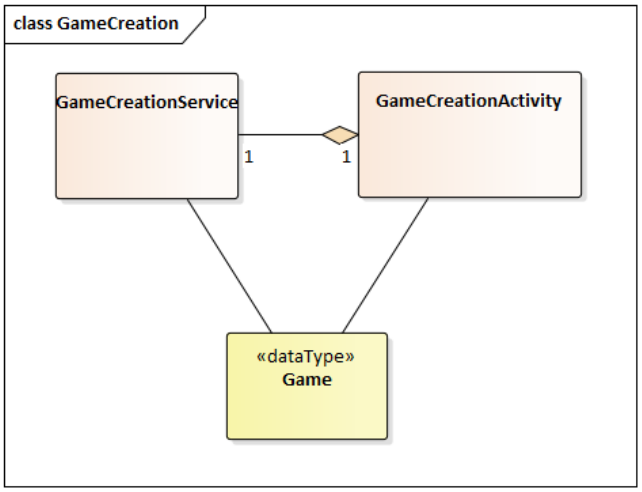

Figure 4.2.6: Diagramme de classe ClientLéger::GameCreation

4.2.7 Launcher

ClientLéger::Launcher
Ce paquetage regroupe les fonctionnalités utilisées pour démarrer l'application. Toute l'initialisation se fait dans ce paquetage et il s'assure de pouvoir retrouver les informations de l'utilisateur.

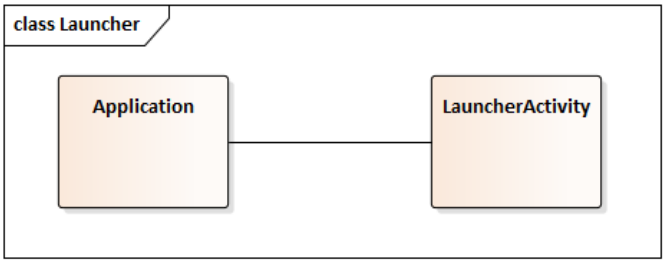

Figure 4.2.7: Diagramme de classe ClientLéger::Launcher

4.2.8 Login

ClientLéger::Login
Ce paquetage regroupe les fonctionnalités utilisées pour se connecter à l'application. Elle s'occupe entre autres de valider les informations de l'utilisateur et de faire les requêtes au serveur pour valider l'utilisateur.

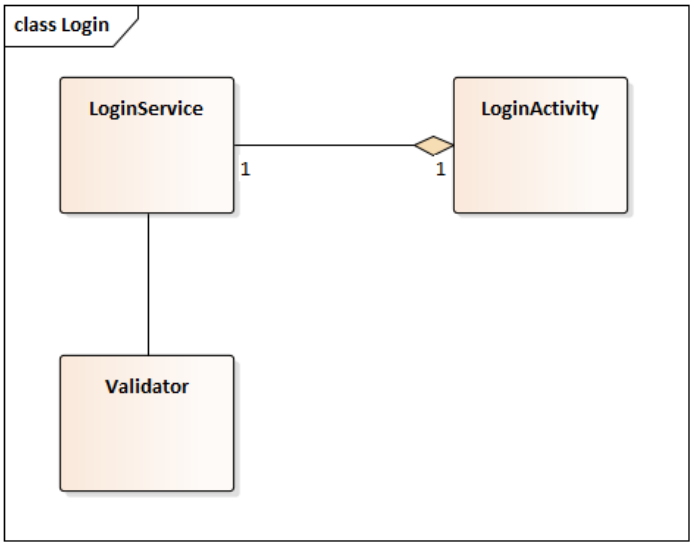

Figure 4.2.8: Diagramme de paquetage ClientLéger::Login

4.2.9 Profile

ClientLéger::Profile
Ce paquetage regroupe les fonctionnalités utilisées pour voir son propre profil utilisateur ou celui d'autres joueurs. On y retrouve aussi le menu des options de l'utilisateur pour modifier son thème ou sa langue.

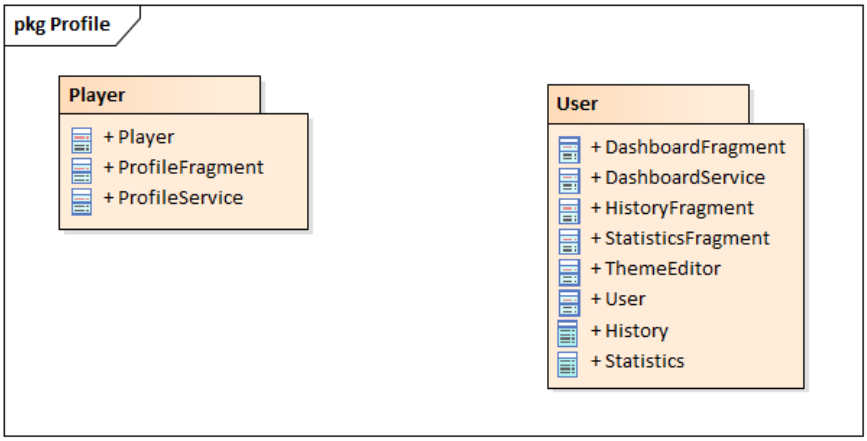

Figure 4.2.9: Diagramme de paquetage ClientLéger::Profile

4.2.10 Player

ClientLéger::Profile::Player
Ce paquetage regroupe les fonctionnalités utilisées pour visionner le profil d'autres joueurs et interagir avec eux.

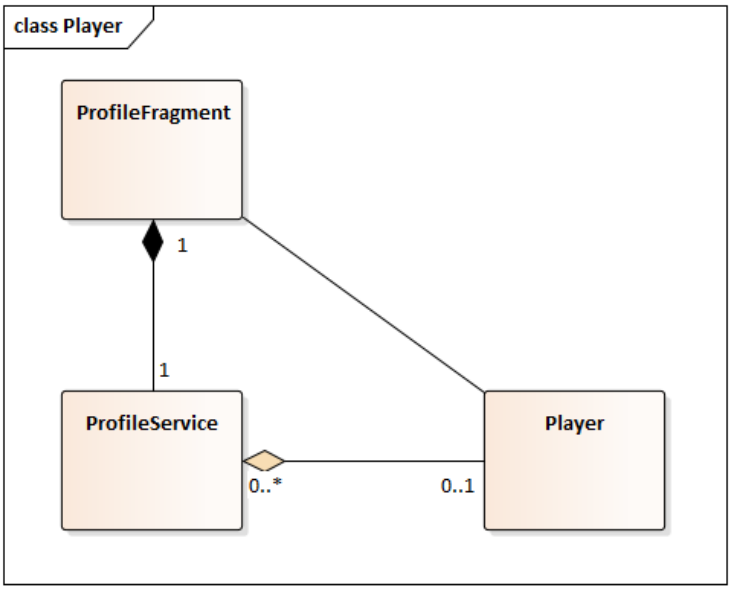

Figure 4.2.10: Diagramme de classe ClientLéger::Profile::Player

4.2.11 User

ClientLéger::Profile::User
Ce paquetage regroupe les fonctionnalités utilisées pour visionner l'historique de l'utilisateur de l'application, ses statistiques, ainsi que modifier ses options telles que la langue et le thème.

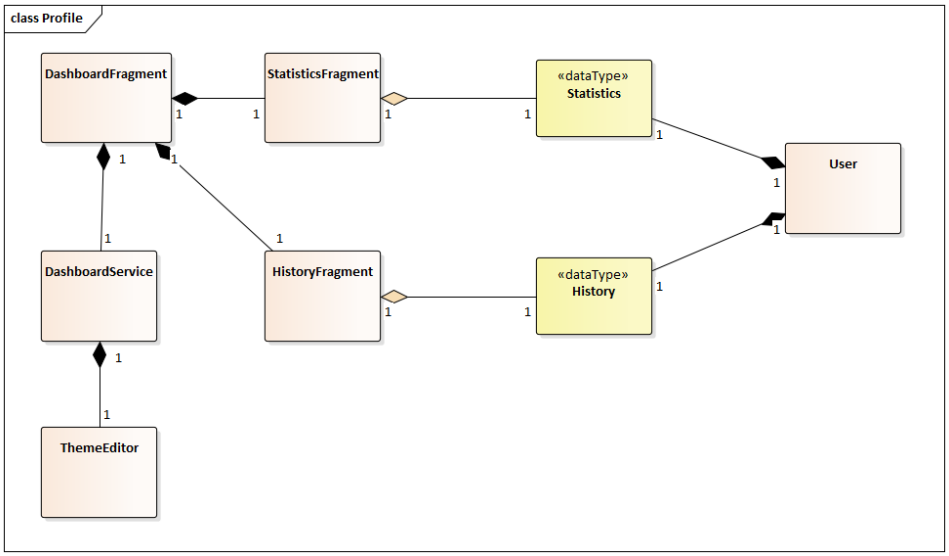

Figure 4.2.11: Diagramme de classe ClientLéger::Profile::User

4.2.12 Registration

ClientLéger::Registration
Ce paquetage regroupe les fonctionnalités utilisées pour se créer un compte lors de la première visite de l'application.

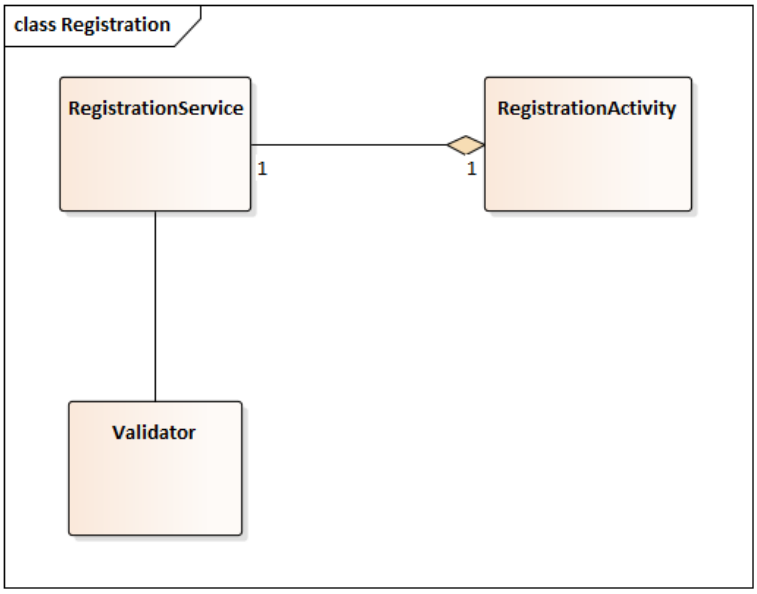

Figure 4.2.12: Diagramme de classe ClientLéger::Registration

4.2.13 Socket

ClientLéger::Socket
Ce paquetage regroupe les fonctionnalités utilisées pour assurer une connexion *Socket* au serveur.

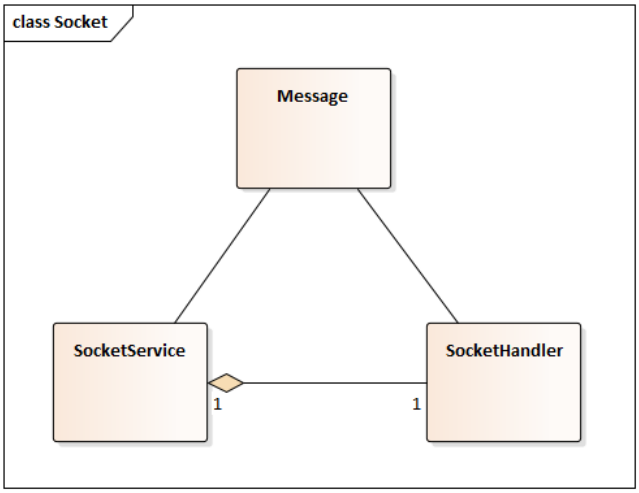

Figure 4.2.13: Diagramme de paquetage ClientLéger::Chat

4.2.14 Session

ClientLéger::Session
Ce paquetage regroupe les fonctionnalités utilisées pour gérer une session. Une session est tout ce qui touche au cycle de vie de l'application et la connexion au serveur. *Session* gère les événements et assure une cohérence d'état de l'application.

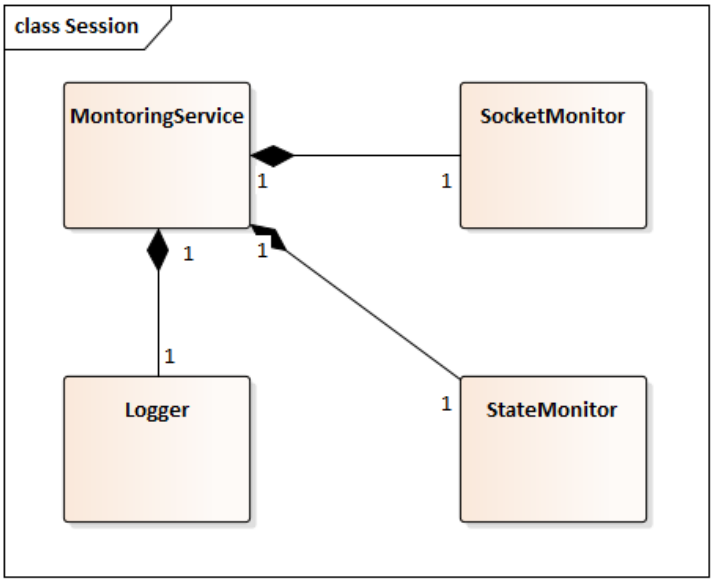

Figure 4.2.14: Diagramme de classe ClientLéger::Session

4.2.15 Shared

ClientLéger::Shared
Ce paquetage regroupe plusieurs classes et paquetages qui ne correspondent pas à une fonctionnalité en particulier et qui sont partagés par plusieurs paquetages. L'utilisation de ce paquetage est à la discrétion des développeurs afin de réduire le code dupliqué.

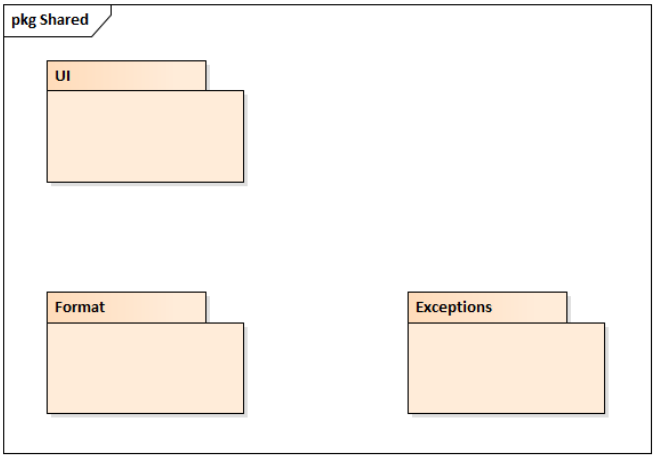

Figure 4.2.15: Diagramme de paquetage ClientLéger::Shared

4.2.16 Tutorial

ClientLéger::Tutorial
Ce paquetage regroupe toutes les fonctionnalités permettant d'offrir un tutoriel à l'utilisateur. On y retrouve les étapes du tutoriel, un service de tutoriel et des éléments d'interface utilisateur pour le tutoriel.

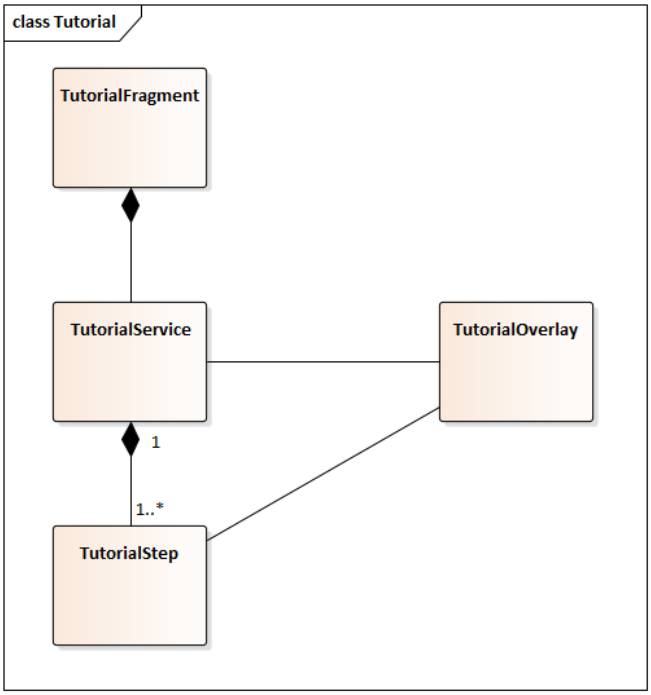

Figure 4.2.16: Diagramme de classe ClientLéger::Tutorial

4.2.17 Resources

ClientLéger::Resources
Ce paquetage regroupe toutes les ressources utilisées par le client léger. On y retrouve des gabarits *XML*, de chaînes de caractères, des menus, etc.

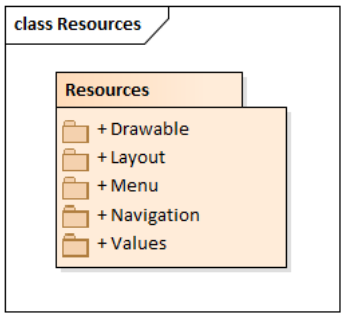

Figure 4.2.17: Diagramme de paquetage ClientLéger::Resources

4.2.18 Drawable

ClientLéger::Ressources::Drawable
Ce paquetage regroupe tous les *drawables*, des images pouvant être tracées et générées sur l'écran. On y retrouve des assistants pour tracer des coins arrondis, des effets visuels et plus.

4.2.19 Layout

ClientLéger::Ressources::Layout
Ce paquetage regroupe tous les gabarits *XML* afin de définir les interfaces utilisateur.

4.2.20 Menu

ClientLéger::Ressources::Menu
Ce paquetage regroupe tous les menus utilisés dans l'application. Par exemple, on y retrouve des menus pour les options, pour dessiner, etc.

4.2.21 Navigation

ClientLéger::Ressources::Navigation
Ce paquetage permet de centraliser la navigation dans l'application et de gérer les transitions entre les différentes activités et fragments.

4.2.22 Values

ClientLéger::Ressources::Values
Ce paquetage contient plusieurs fichiers regroupant certains types de valeurs qui sont partagées dans l'application. Ce paquetage permet d'éviter de la répétition et de l'incohérence. Par exemple, on retrouve un fichier *strings.xml* qui contient toutes les chaînes de caractères de l'interface utilisateur dans toutes les langues supportées, un fichier *colors.xml* qui regroupe des couleurs de l'interface utilisateur, un fichier *dimens.xml* qui contient des dimensions de l'interface utilisateur et bien plus.

4.3 Serveur

Le serveur possède une architecture qui est orientée en services. Les messages sont envoyés entre les différents services selon le patron observateur. Ceci permet d'avoir une architecture qui est découplée entre les services. Les services peuvent donc être développés en parallèle et accélérer le développement. De plus, il est facile d'ajouter de nouvelles fonctionnalités. Une des fonctionnalités de Go Lang a été utilisée pour l'architecture. Il s'agit des canaux. Ces canaux permettent d'envoyer les messages aux observateurs et dans ce cas-ci les services. Ces canaux s'occupent d'avoir un tampon de messages pouvant ainsi s'assurer que les services lisent les messages lorsqu'ils sont prêts à les lire. Ces canaux sont utilisés pour communiquer entre les différents fils d'exécutions. Ceci permet donc de tirer profit des ressources du serveur et donc supporter plusieurs connexions.

Serveur
Le serveur possède deux parties différentes: une API REST et une avec des Sockets. Ces deux paquetages partagent plusieurs composantes communes. Ils communiquent en grande partie avec le patron observateur global dans le paquetage cbroadcast.

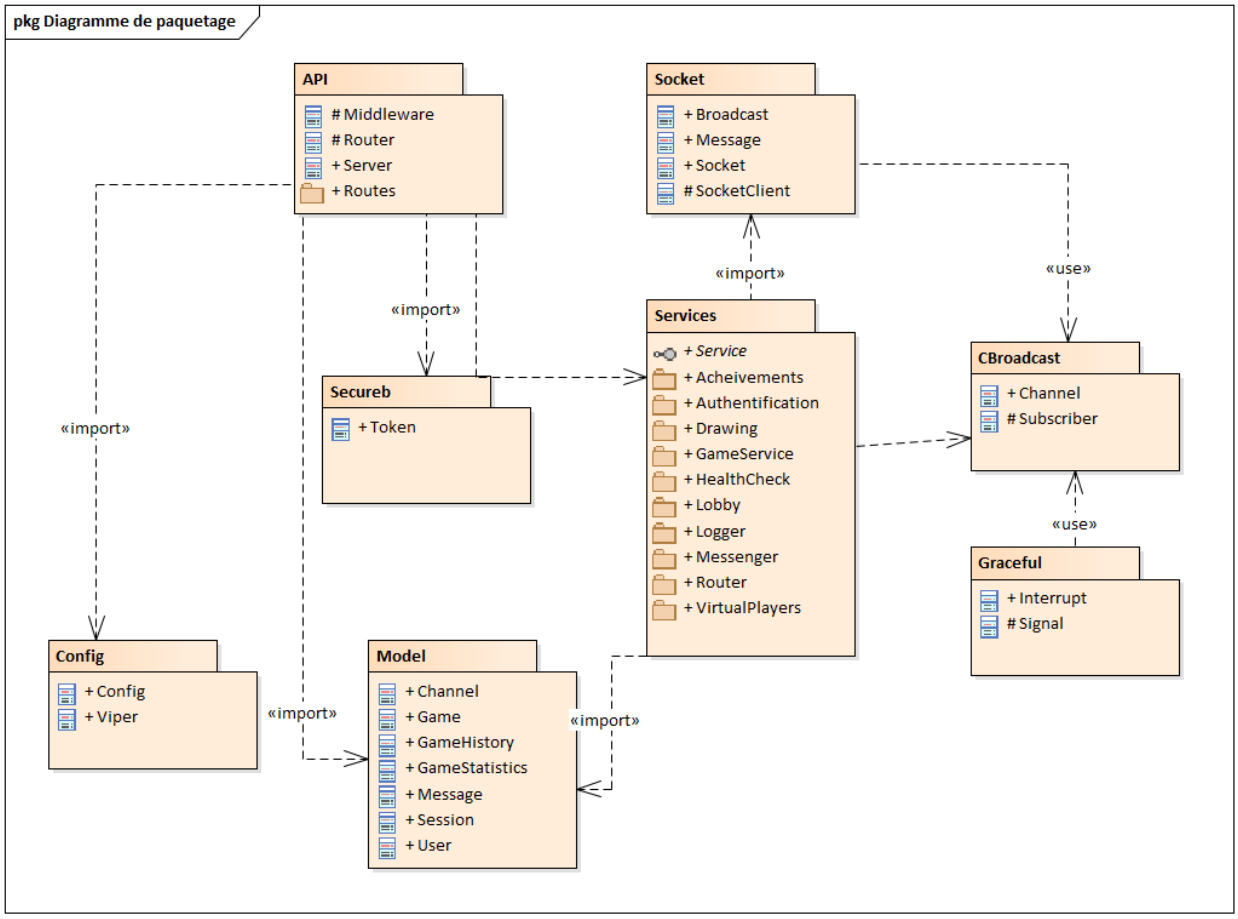

Figure 4.3: Diagramme de paquetage serveur

4.3.1 CBroadcast (Patron observateur)

Serveur::CBroadcast
Ce paquetage est utilisé pour agir comme patron observateur. Il permet à plusieurs composantes découplées de s'abonner à différents canaux de messages. Lorsqu'un émetteur veut émettre un message, il peut le faire à plusieurs observateurs dans toute l'application. Ceci permet d'avoir des canaux de Go d'une façon one to many.

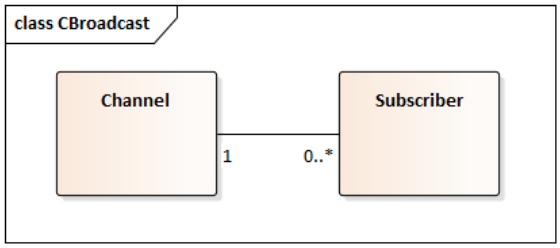

Figure 4.3.1: Diagramme de classe Serveur::CBroadcast

4.3.2 Graceful

Serveur::Graceful
Ce paquetage est utilisé pour répondre au signal SIGTERM. Il s'occupe de bien fermer le serveur et les connexions en cours et d'enregistrer toutes les transactions dans la base de données.

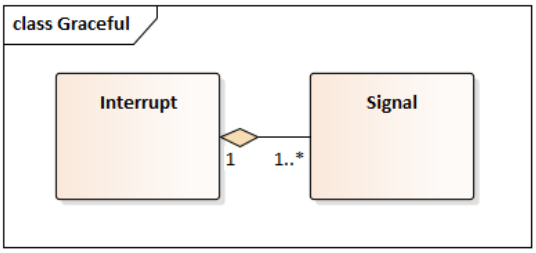

Figure 4.3.1: Diagramme de classe Serveur::CBroadcast

4.3.3 Secureb

Serveur::Secureb
Ce paquetage est utilisé pour générer des octets et des jetons sécuritaires. Il est utilisé dans les API de l'authentification. Le paquetage contient une seule classe.

4.3.4 Config

Serveur::Config
Ce paquetage est utilisé pour gérer les configurations du serveur. Il est possible de mettre des valeurs par défaut dans le cas où aucun fichier de configuration n'est présent. Il y a une dépendance sur la bibliothèque viper pour lire les fichiers de configurations.

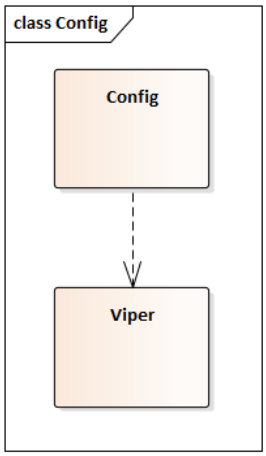

Figure 4.3.4: Diagramme de classe Serveur::Config

4.3.5 Model

Serveur::Model
Ce paquetage est utilisé pour gérer les modèles du SGBD. Ces modèles sont utilisés dans un ORM afin d'éviter d'avoir à rédiger du SQL. Les modèles sont des dépendances de plusieurs autres paquetages.

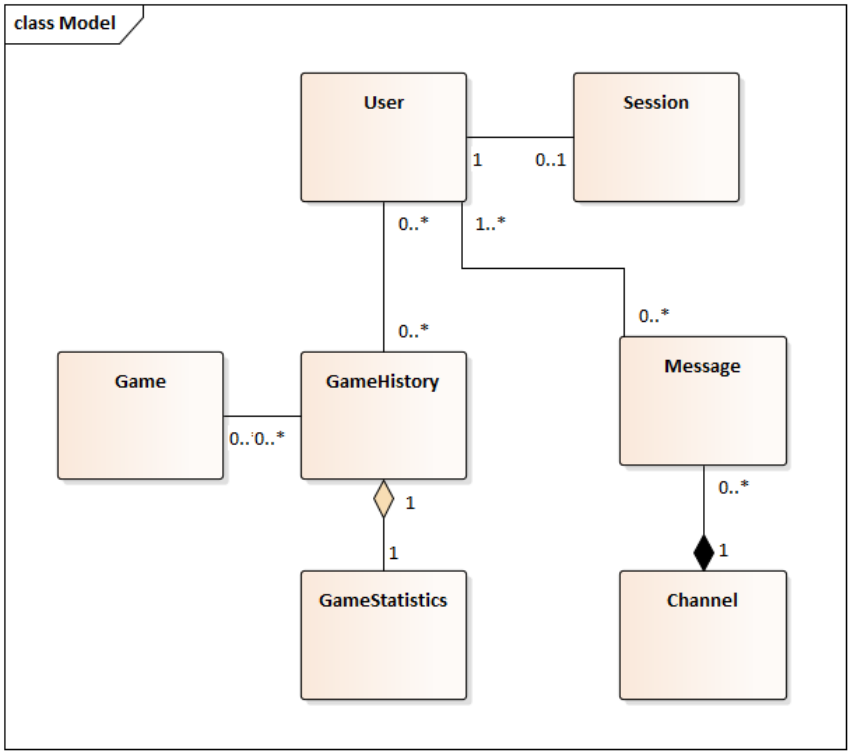

Figure 4.3.5: Diagramme de classe Serveur::Model

4.3.6 API et Socket

Serveur::API
Ce paquetage héberge toutes les routes de l'API. Il contient également le système de routage ainsi que les différents intergiciels tels que l'authentification et la journalisation.

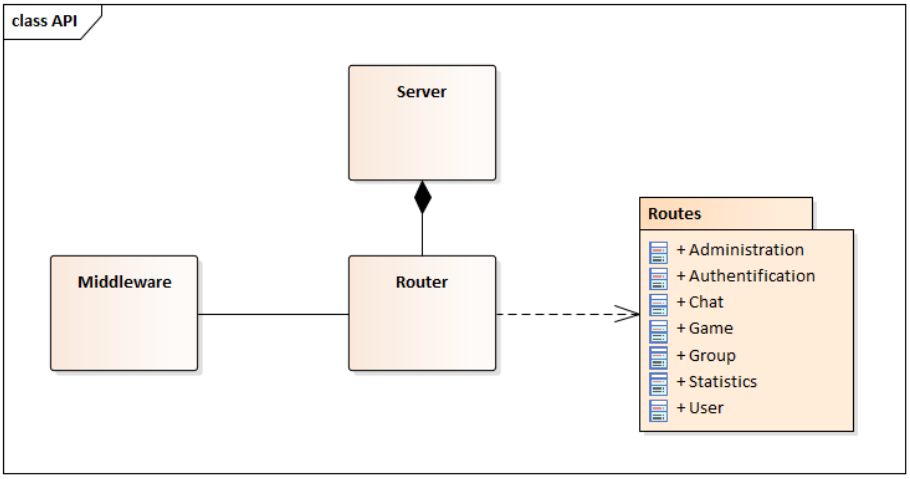

Figure 4.3.6.1: Diagramme de classe Serveur::API

Serveur::Socket
Ce paquetage inclut la logique de connexion et de déconnexion des clients. Il s'occupe d'envoyer des messages aux clients et de les recevoir. Ces messages sont ensuite envoyés aux observateurs de ce type de message.

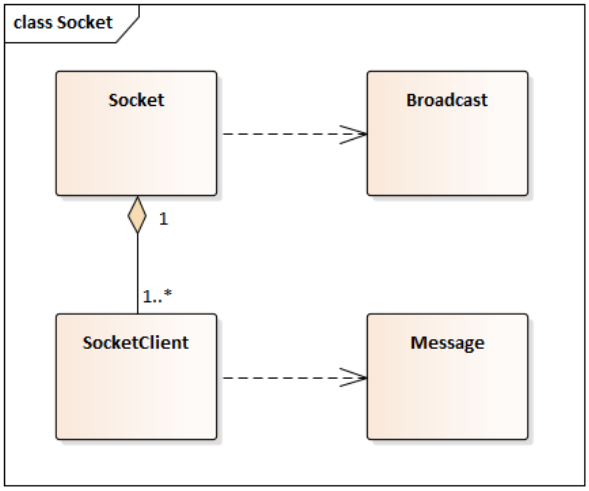

Figure 4.3.6.2: Diagramme de classe Serveur::Socket

4.3.7 Services

Serveur::Services
Ce paquetage représente tous les services utilisés par le serveur. Ils sont tous découplés et communiquent ensemble via le patron observateur.

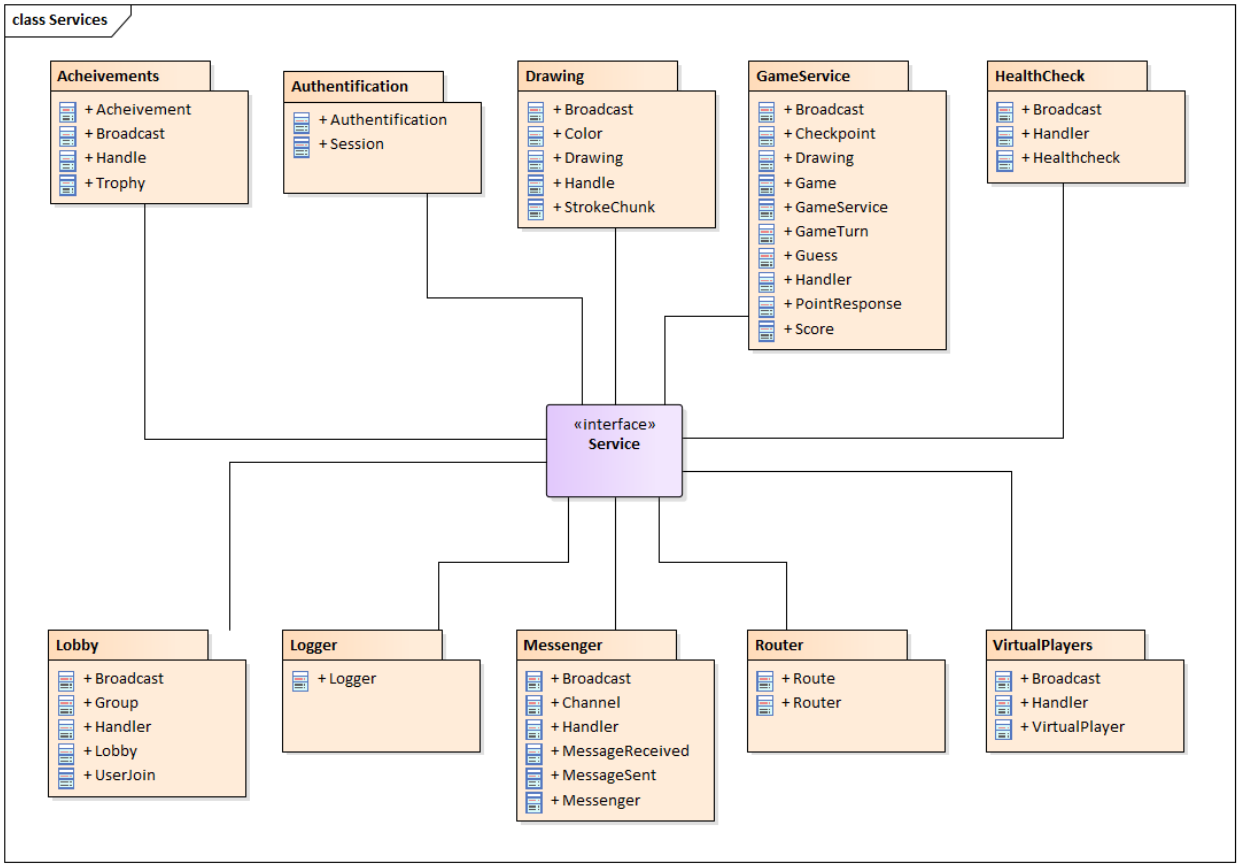

Figure 4.3.7.1: Diagramme de classe Serveur::Services

Serveur::Services::Achievements
Ce paquetage est utilisé pour faire la gestion des lancements des trophées au courant des parties. Il écoute sur des événements et lorsque des conditions sont remplies, il donne des trophées aux joueurs.

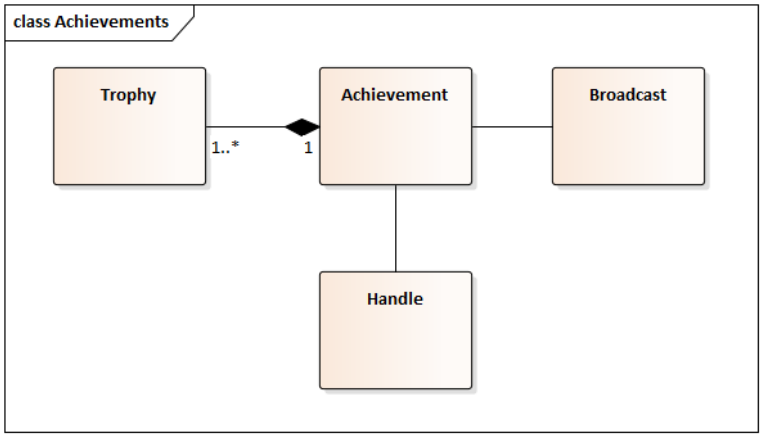

Figure 4.3.7.2: Diagramme de classe Serveur::Services::Achievements

Serveur::Services::Authentification
Ce paquetage est utilisé pour gérer la connexion et l'autorisation d'accéder aux diverses sessions. Il s'occupe de faire le pont entre les deux interfaces soit celle d'API et le Socket.

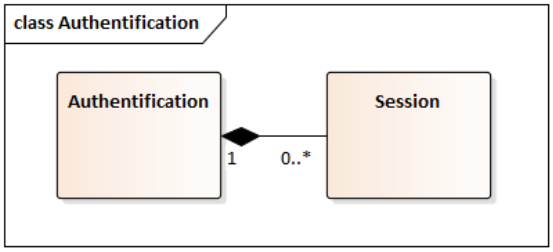

Figure 4.3.7.3: Diagramme de classe Serveur::Services::Authentification

Serveur::Services::Drawing
Ce paquetage est utilisé pour gérer l'envoi des traits de dessins sur le socket. Il s'occupe également de la réception des traits dessinés par un client et de les envoyer aux bons clients.

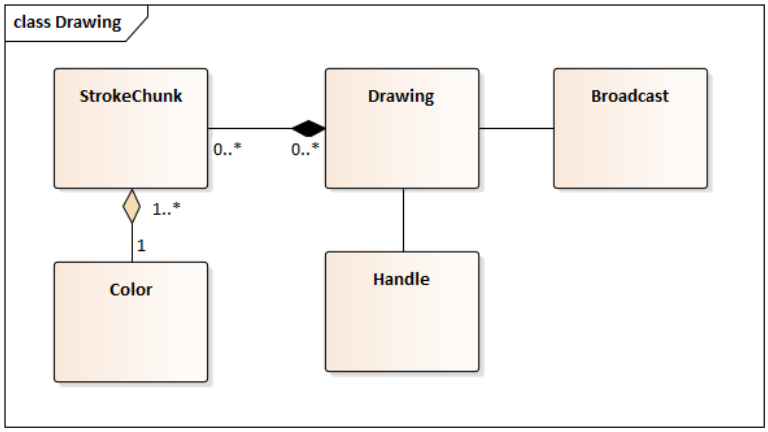

Figure 4.3.7.4: Diagramme de classe Serveur::Services::Drawing

Serveur::Services::GameService
Ce paquetage est utilisé pour gérer les parties en cours. Il s'occupe de gérer le temps de chaque partie et gérer les événements. S'il y a des joueurs virtuels, ils passent la tâche de dessiner au service de dessin.

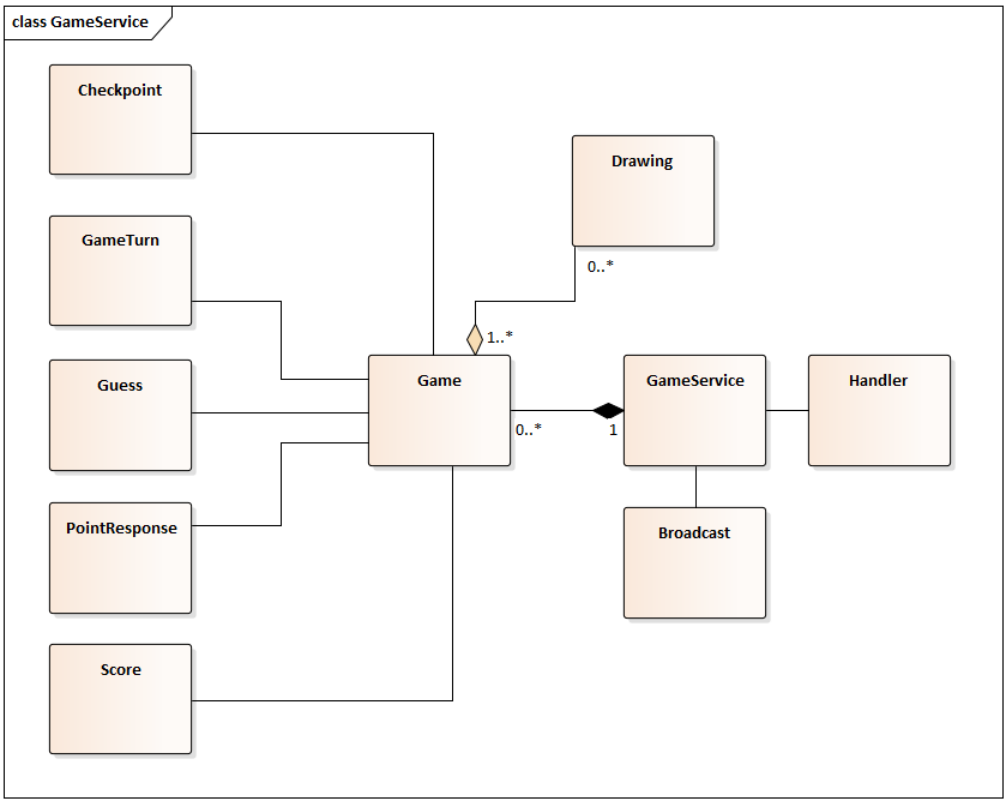

Figure 4.3.7.5: Diagramme de classe Serveur::Services::GameService

Serveur::Services::Healthcheck
Ce paquetage est utilisé pour s'assurer que les connexions des clients existent toujours. Dans le cas où le client ne répond pas dans les délais prescrits, la connexion est fermée. Le GameService en est aussi informé afin de prendre une décision si la partie doit continuer ou non.

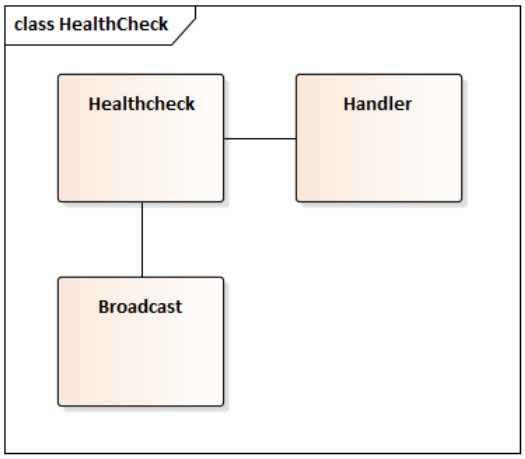

Figure 4.3.7.6: Diagramme de classe Serveur::Services::Healthcheck

Serveur::Services::Logger
Ce paquetage est utilisé pour écrire dans les journaux du serveur. Il écoute pour divers événements et il écrit dans les journaux lorsqu'un événement digne d'intérêt se produit.

Serveur::Services::Router
Ce paquetage est utilisé pour envoyer les messages reçus au socket aux bons services. Il peut router plusieurs messages à différents services.

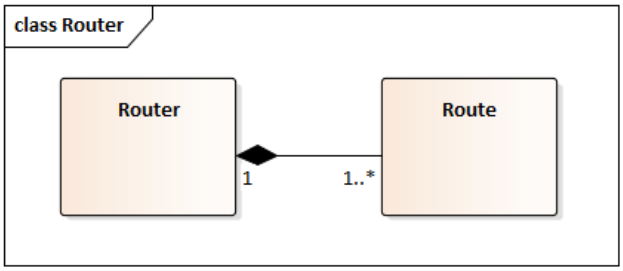

Figure 4.3.7.7: Diagramme de classe Serveur::Services::Router

Serveur::Services::VirtualPlayers
Ce paquetage est utilisé pour gérer le comportement des joueurs virtuels. Ils répondent aux différents événements de la partie.

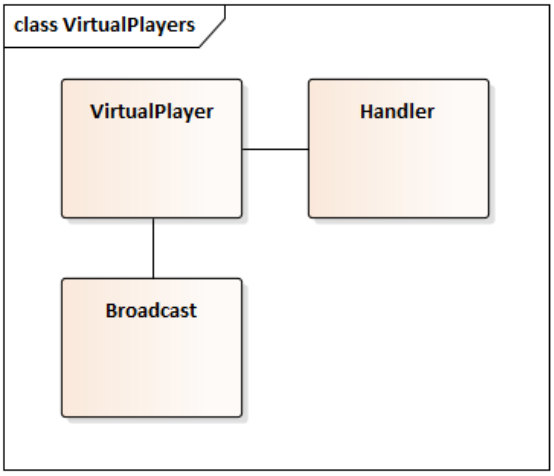

Figure 4.3.7.8: Diagramme de classe Serveur::Services::VirtualPlayers

Serveur::Services::Lobby
Ce paquetage est utilisé pour gérer les salles d'attente et les groupes avant de créer une partie.

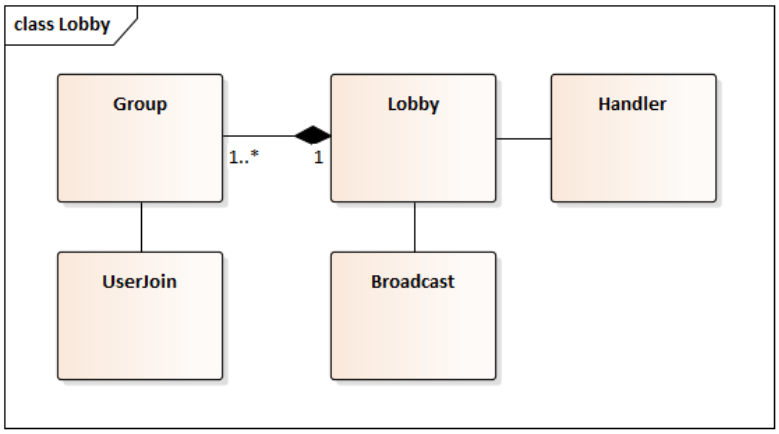

Figure 4.3.7.9: Diagramme de classe Serveur::Services::Lobby

Serveur::Services::Messenger
Ce paquetage est utilisé pour gérer le clavardage ainsi que toutes les fonctionnalités comme les salles de discussions. Il s'occupe d'envoyer les messages reçus aux bons clients.

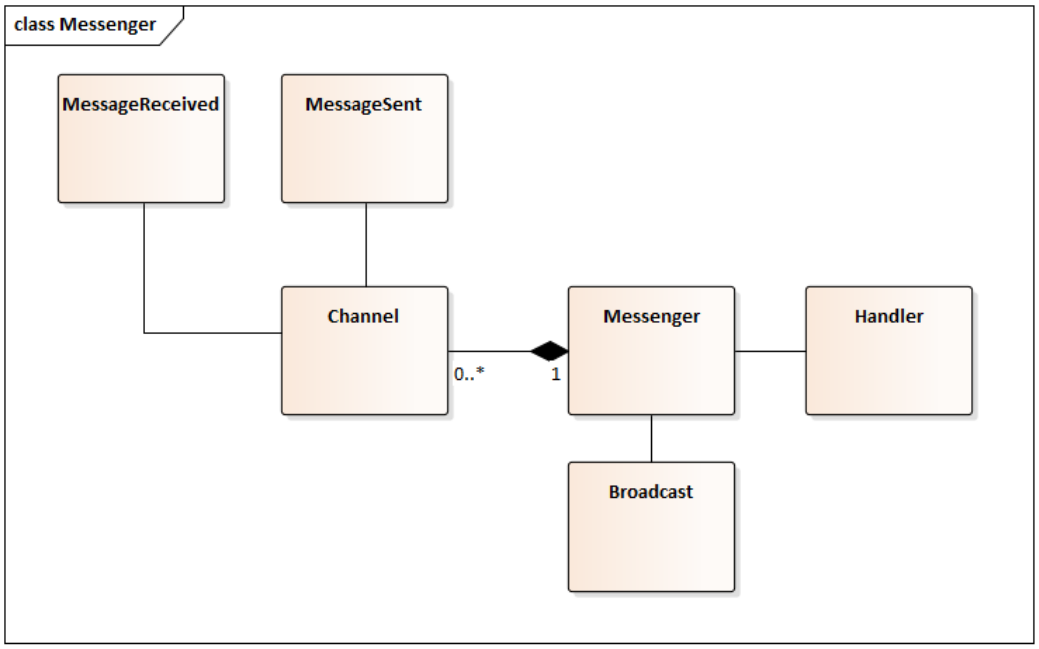

Figure 4.3.7.10: Diagramme de classe Serveur::Services::Messenger

5. Vue des processus

La présente section vise à illustrer les séquences les plus pertinentes de l'application. La première séquence présentée est celle de la connexion de l'utilisateur.

5.1 Se connecter

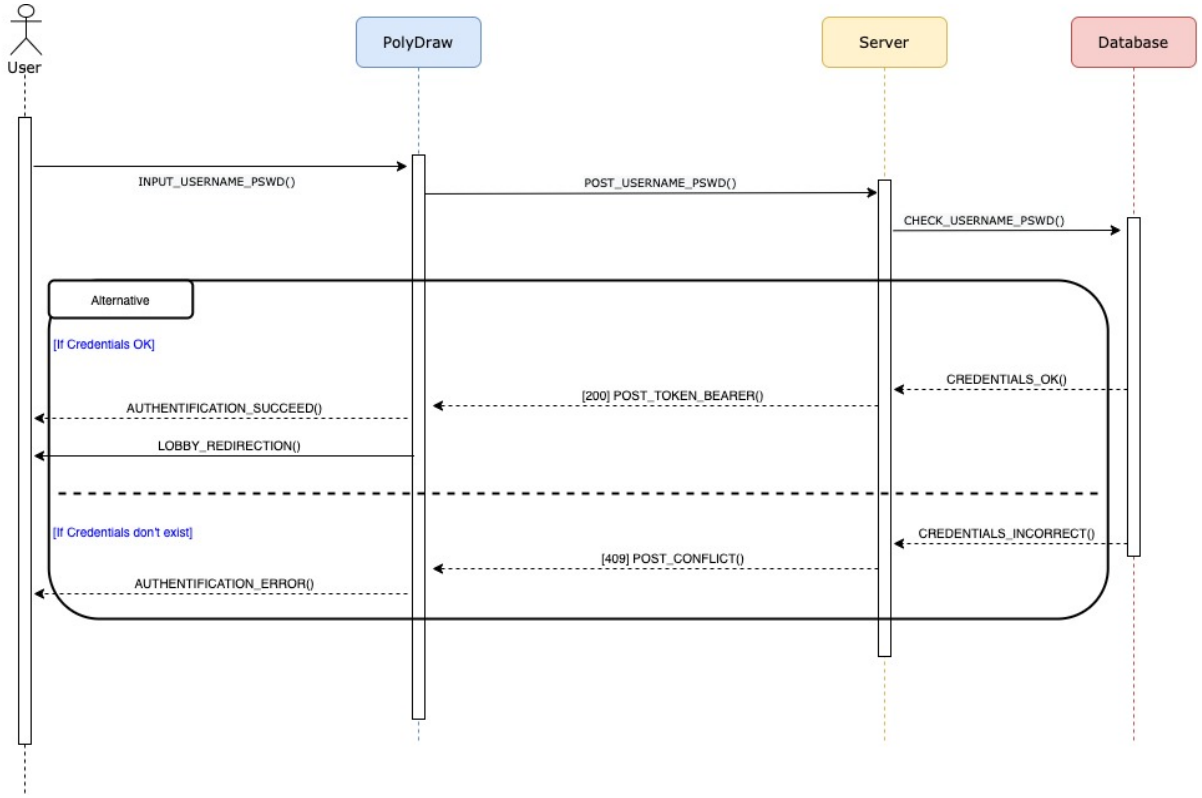

Figure 5.1.1: Présentation de la séquence «se connecter».

Cette séquence s'exécute lors de la connexion (vu en section 3.1). Après avoir saisi le nom d'utilisateur et le mot de passe, ces derniers sont envoyés au serveur qui va vérifier s'ils sont valides. S'ils le sont, l'utilisateur sera redirigé vers la vue d'accueil du jeu. En revanche, si les informations saisies sont incorrectes, un message d'erreur va s'afficher.

5.2 Créer un profil d'utilisateur

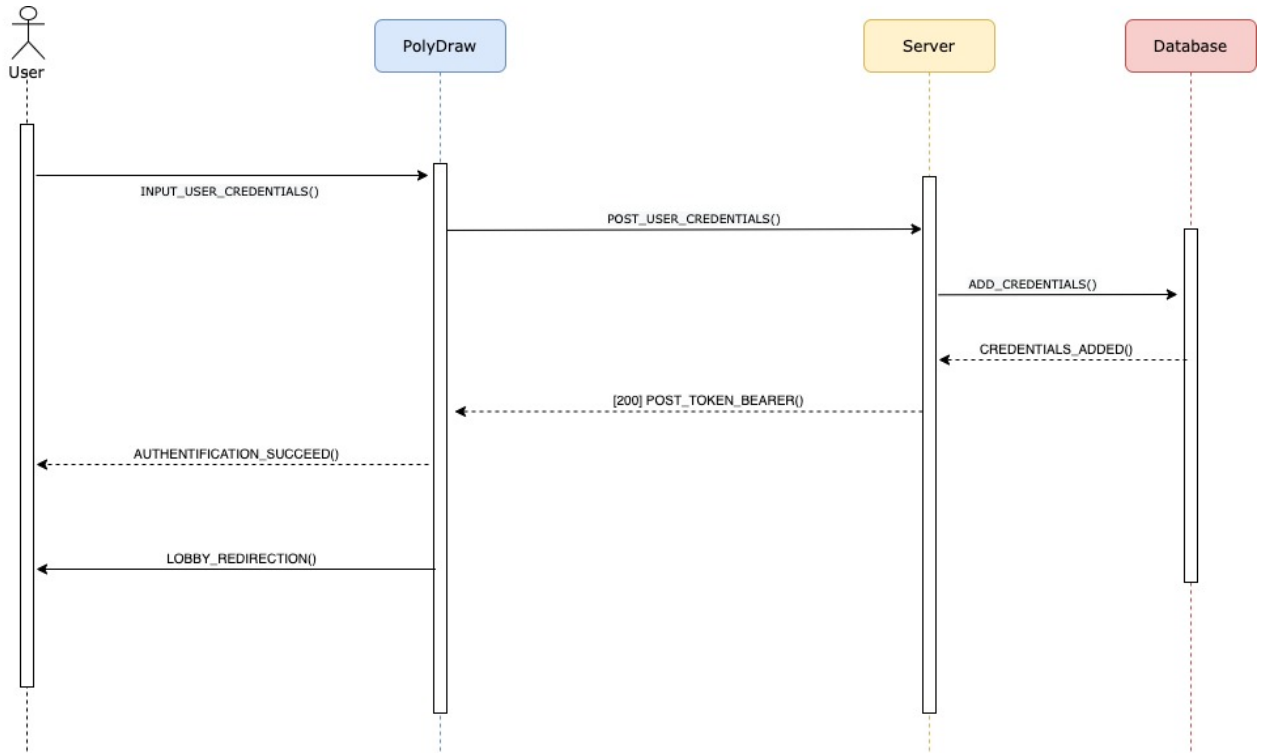

Figure 5.2.1: Présentation de la séquence «créer un profil d'utilisateur».

Cette séquence s'exécute lors de la création d'un profil (vu en section 3.2). Après avoir saisi le nom d'utilisateur et le mot de passe, ces derniers sont envoyés au serveur qui va procéder à la création du profil sur la base de données.

5.3 Clavarder

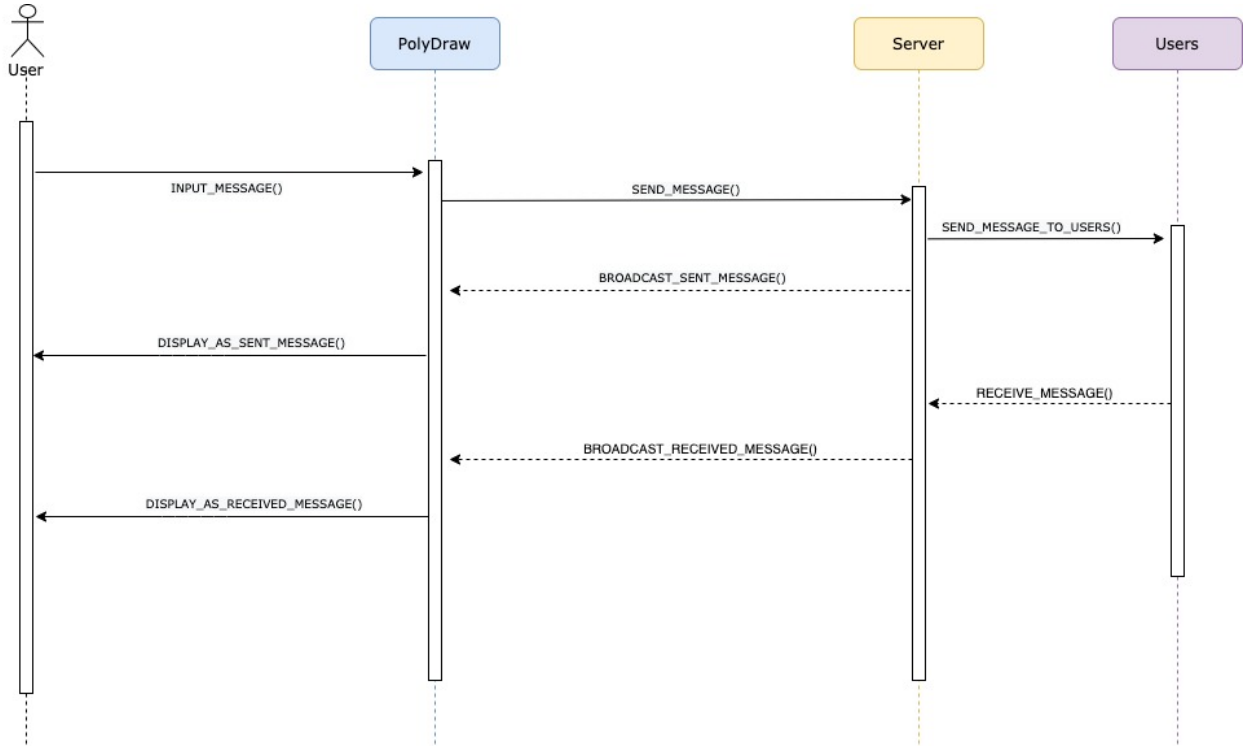

Figure 5.3.1: Présentation de la séquence «clavarder».

Cette séquence s'exécute lorsque l'utilisateur clavarde sur le chat (vu en section 3.3).

Lors de l'envoi de message, le contenu de ce dernier sera envoyé sur le serveur puis transmis à tous les utilisateurs présents dans ce canal.

Lors de la réception de message, d'autres usagers issus du même canal envoient des messages au serveur, qui pourra ensuite les transmettre à l'utilisateur.

5.4 Gérer un chat

Les séquences de cette partie s'exécutent lorsque l'utilisateur gère un chat (vu en section 3.4).

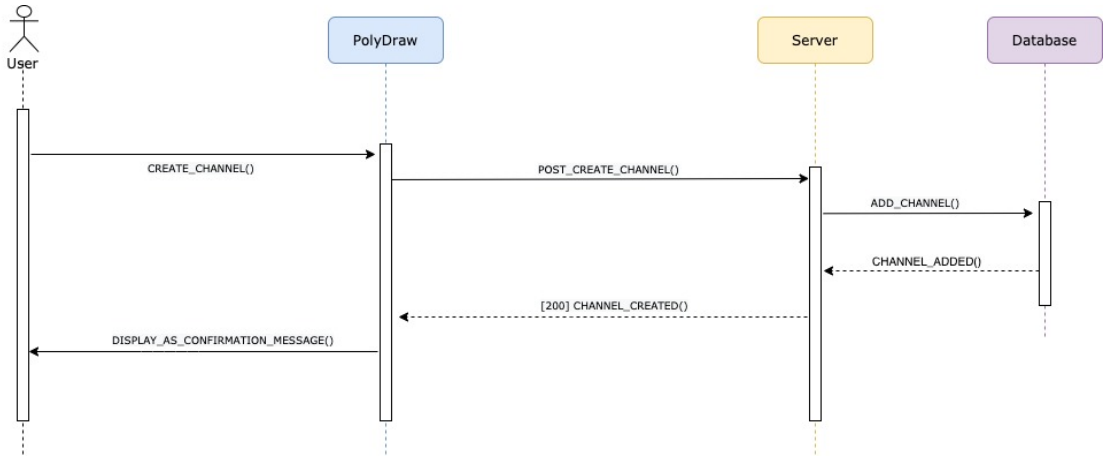

Figure 5.4.1: Présentation de la séquence *créer un canal* issue de «gérer un chat».

Après avoir saisi le nom du canal, les informations du canal seront envoyées au serveur qui pourra procéder à la création de ce dernier.

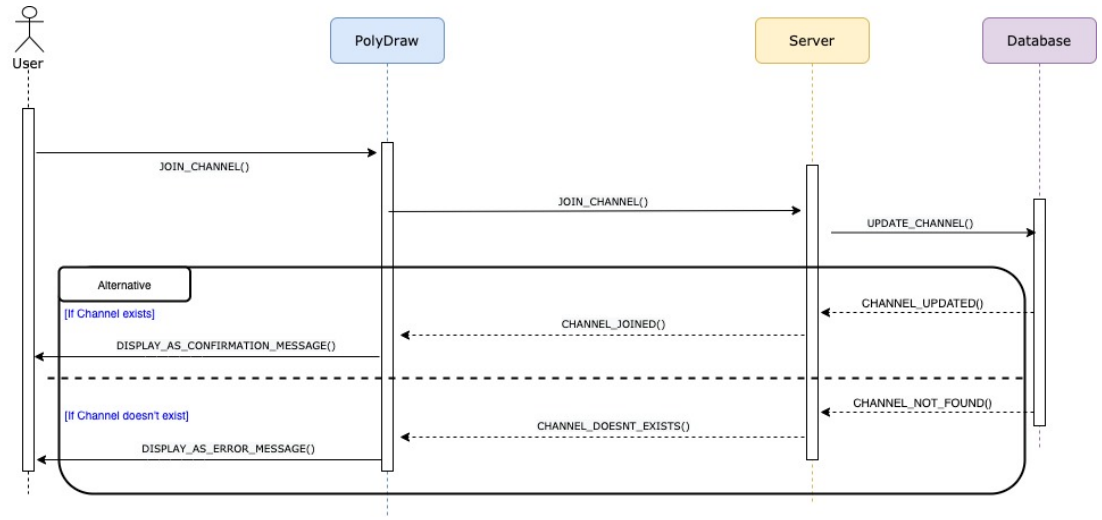

Figure 5.4.2: Présentation de la séquence *rejoindre un canal* issue de «gérer un *chat*».

Après avoir saisi le canal qu'on souhaite rejoindre, le serveur vérifiera son existence. S'il existe, le serveur permettra à l'utilisateur d'écouter les messages de ce canal. Si le canal n'existe pas, le serveur l'indiquera au client qui l'affichera à l'utilisateur sous forme d'un message d'erreur.

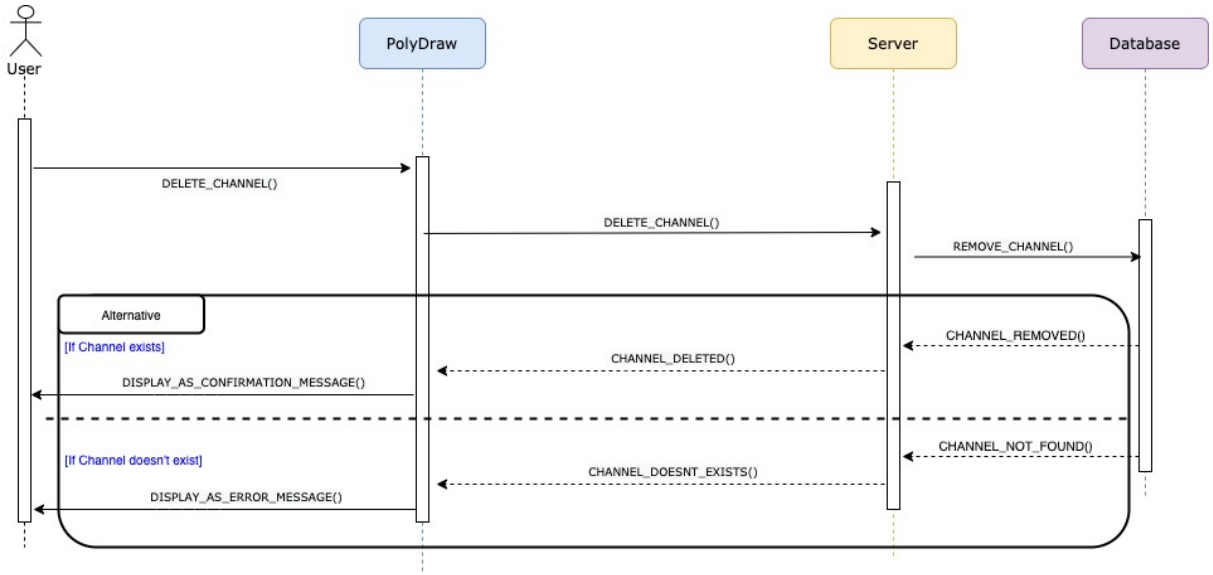

Figure 5.4.3: Présentation de la séquence ***supprimer un canal*** issue de «gérer un *chat*».

Après avoir saisi le canal nom du canal qu'on souhaite supprimer, le serveur essaiera de le supprimer sur la base de données. S'il existe, le serveur procèdera à la suppression de ce canal. En revanche, si le canal n'existe pas, le serveur l'indiquera au client qui l'affichera à l'utilisateur sous forme d'un message d'erreur.

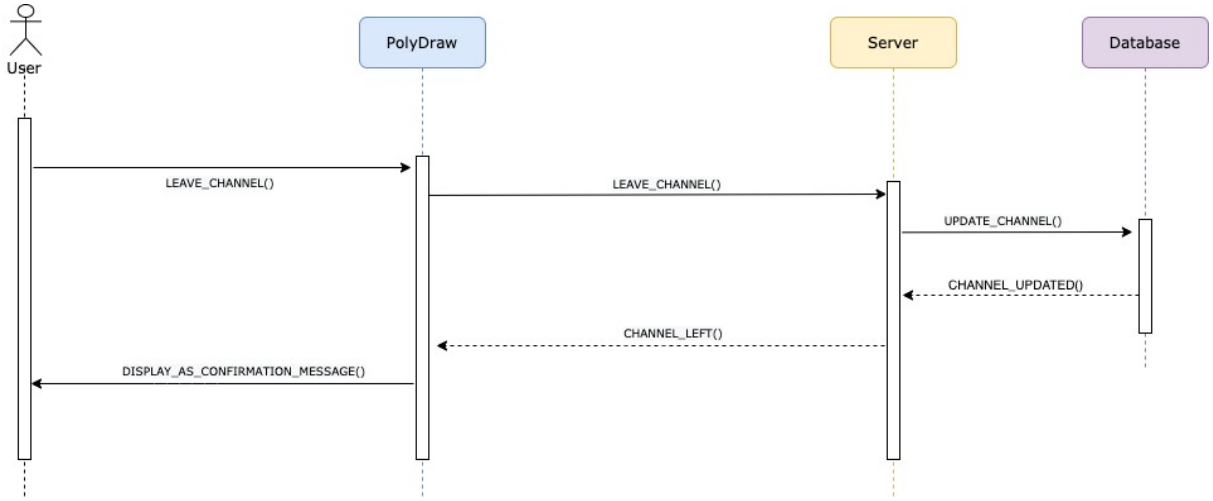

Figure 5.4.4: Présentation de la séquence ***quitter un canal*** issue de «gérer un *chat*».

Après avoir saisi le nom du canal qu'on souhaite quitter, le client demandera au serveur de couper la connexion avec ce dernier et ne permettra plus à l'utilisateur d'écouter les messages de ce canal.

5.5 Créer un jeu

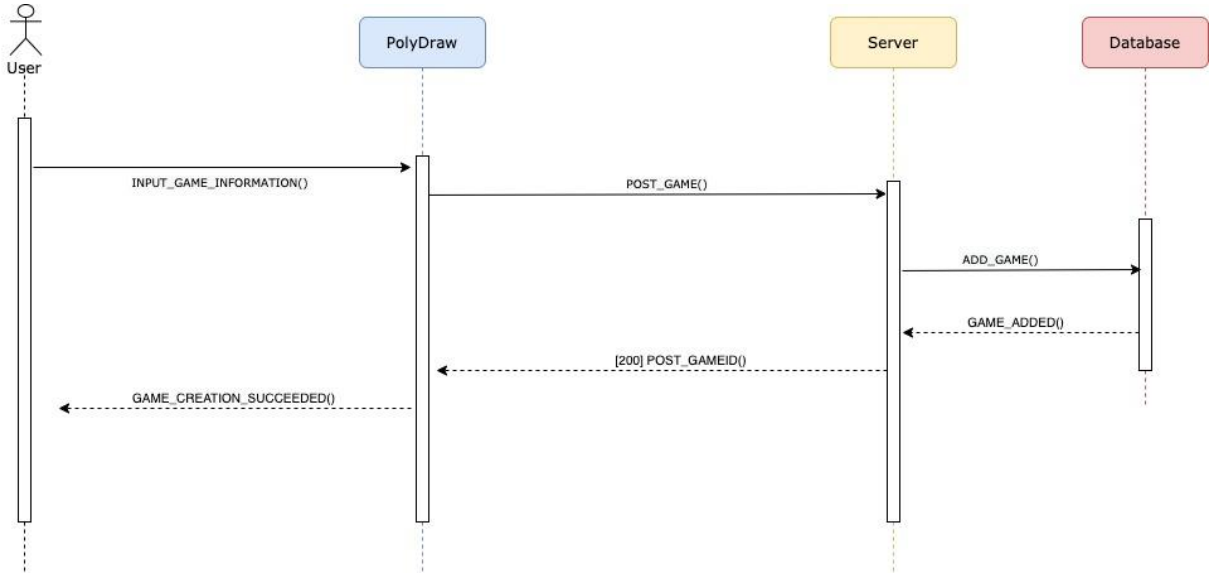

Figure 5.5.1: Présentation de la séquence «créer un jeu».

Cette séquence s'exécute lorsque l'utilisateur souhaite créer un nouveau jeu (vu en section 3.5). Une fois que l'utilisateur aura saisi toutes les informations du jeu (nom, indices, etc.), le client les enverra au serveur. Ensuite, le serveur procèdera à la création du jeu sur la base de données.

5.6 Gérer un profil

Les séquences de cette partie s'exécutent lorsque l'utilisateur gère son profil (vu en section 3.7).

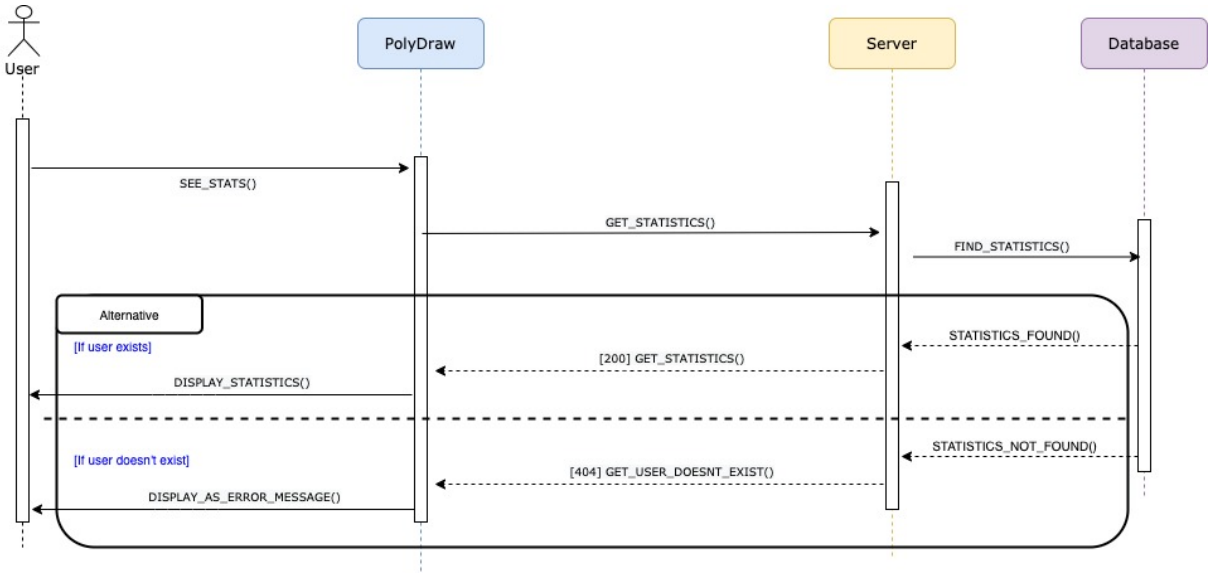

Figure 5.6.1: Présentation de la séquence *<u>récupérer les statistiques</u>* issue de «gérer un profil».

Lorsque l'utilisateur souhaite accéder à ses statistiques, il fera une requête sur le serveur. S'il y a des statistiques en base de données, le serveur les enverra au client et l'utilisateur pourra donc les voir. En revanche, s'il n'y a pas encore de statistiques, un message d'erreur s'affichera.

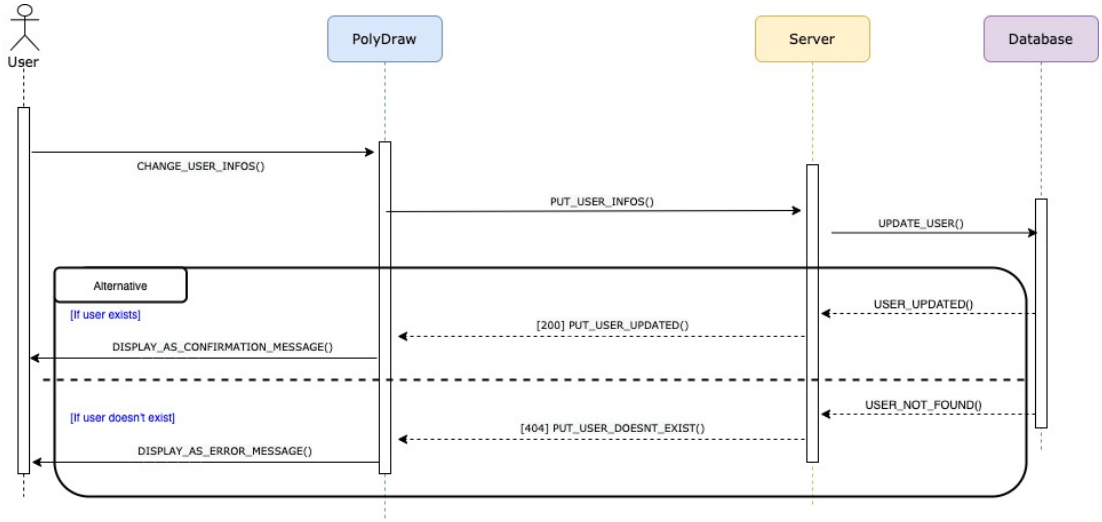

Figure 5.6.2: Présentation de la séquence *<u>modifier les informations utilisateur</u>* issue de «gérer un profil».

Lorsque l'utilisateur souhaite modifier ses informations, il fera une requête sur le serveur. Si son profil est trouvé en base de données, les changements seront effectués. Si le profil n'est pas trouvé, un message d'erreur s'affichera.

5.7 Créer une partie (lobby)

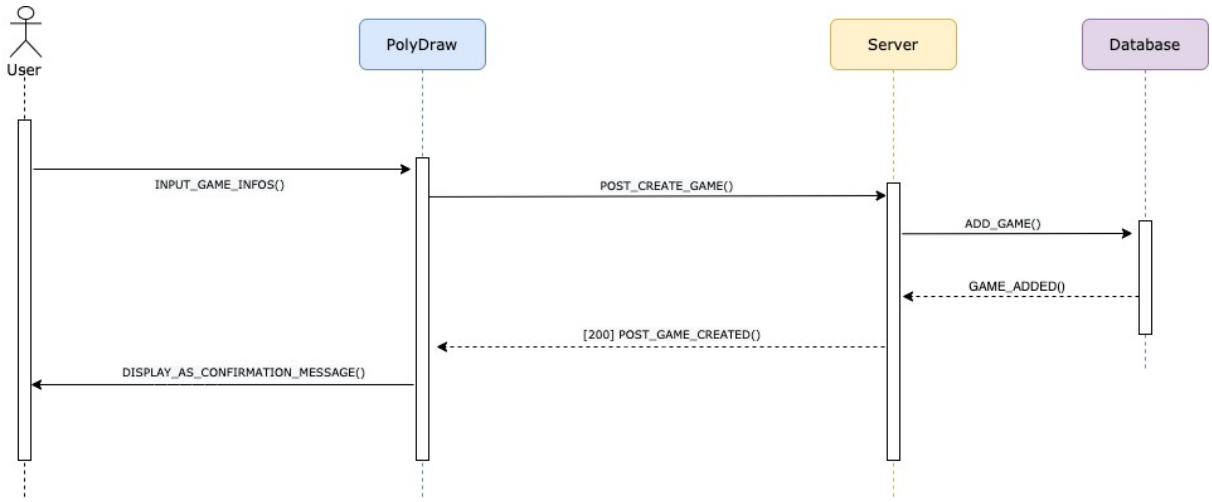

Figure 5.7.1: Présentation de la séquence «créer une partie».

Cette séquence s'exécute lorsque l'utilisateur souhaite créer un nouveau jeu (vu en section 3.8). Une fois que l'utilisateur aura saisi toutes les informations de la partie, le client les enverra au serveur. Ensuite, le serveur procèdera à la création de la partie sur la base de données.

5.8 Jouer à une partie

Lorsqu'on joue à une partie, qu'il s'agisse d'une partie en mode FFA, solo ou coopératif, on peut distinguer des séquences similaires. En effet, on peut le voir sur les diagrammes d'utilisations (figures 3.10, 3.11, 3.12 respectivement pour FFA, solo et coopératif), l'utilisateur peut demander des indices tant qu'il y en a des disponibles pour l'aider à soumettre un mot.
Ici, nous avons décidé de représenter les séquences "demander un indice" et "soumettre un mot".
Les séquences de cette partie s'exécutent lorsque l'utilisateur crée une partie (vus en section 3.10, 3.11 et 3.12).

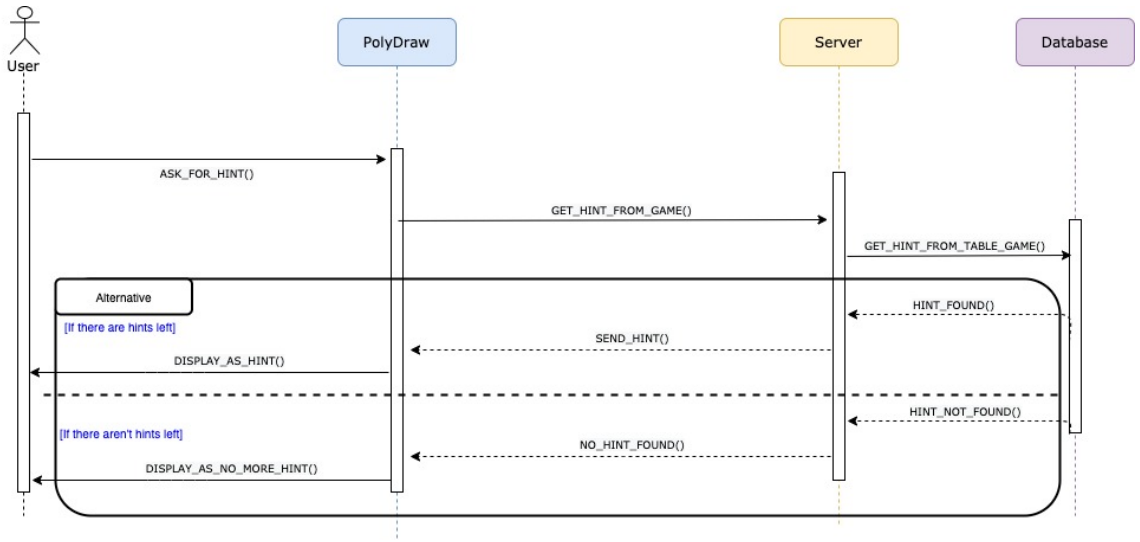

Figure 5.8.1: Présentation de la séquence *demander un indice* issue de «jouer à une partie []».

L'utilisateur peut demander des indices. Il va faire une demande au serveur, qui fera une recherche sur la base de données. Si un indice est trouvé, il sera envoyé au client qui l'affichera pour l'utilisateur. En revanche, si aucun indice n'est trouvé, le client affichera un message indiquant qu'il n'y a plus d'indices disponibles.

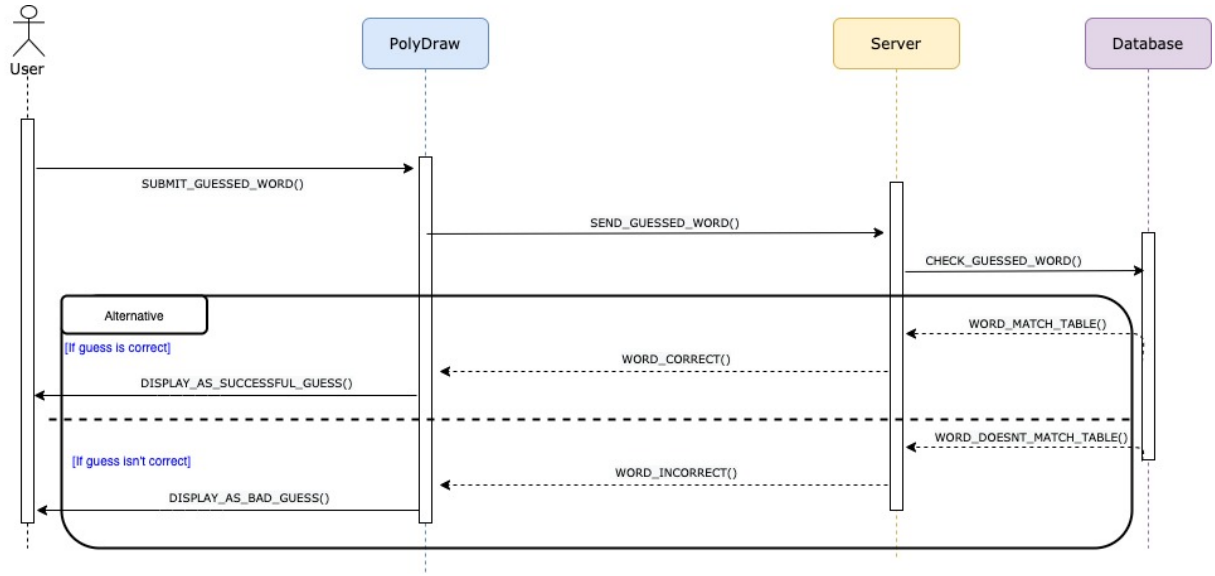

Figure 5.8.2: Présentation de la séquence *soumettre un mot* issue de «jouer à une partie []».

L'utilisateur peut essayer de deviner un mot. Le client va alors envoyer au serveur ce dernier, et fera une recherche dans la base de données. Si le mot est le bon, le client félicitera l'utilisateur qui pourra passer à la prochaine. En revanche, si le mot est incorrect, le client affichera un message pour l'indiquer.

6. Vue de déploiement

Le logiciel PolyDraw s'exécute sur trois plateformes différentes. Il s'exécute sur Android et sur un ordinateur. Ces deux clients communiquent au serveur via une connexion Ethernet. Cette connexion peut être filaire ou peut être sur Wifi. Le serveur est virtualisé et possède 4 coeurs virtuels permettant de faire du travail en parallèle. Il est donc possible d'avoir plusieurs connexions d'une façon simultanée.

Le serveur emploie différents mécanismes pour s'assurer d'une haute fiabilité. Le serveur est déployé dans Docker pour s'assurer que celui-ci s'exécute toujours dans le même environnement. En plus de s'exécuter dans Docker les journaux et l'état du conteneur sont surveillés par Datadog. Ce service permet d'envoyer des alertes dans le cas où le serveur planterait afin de détecter les bogues potentiels.

Les données sont sauvegardées dans une base de données PostgreSQL. Cette base de données est présente également sous la forme d'un conteneur. En plus de la base de données, un autre conteneur est présent pour envoyer des courriels. Il s'agit du logiciel Postfix. Finalement afin de gérer ces conteneurs, le logiciel docker-compose est utilisé. Le tout est intégré à de l'intégration continue afin de s'assurer que la version du serveur est toujours à jour et que le logiciel est toujours testé.

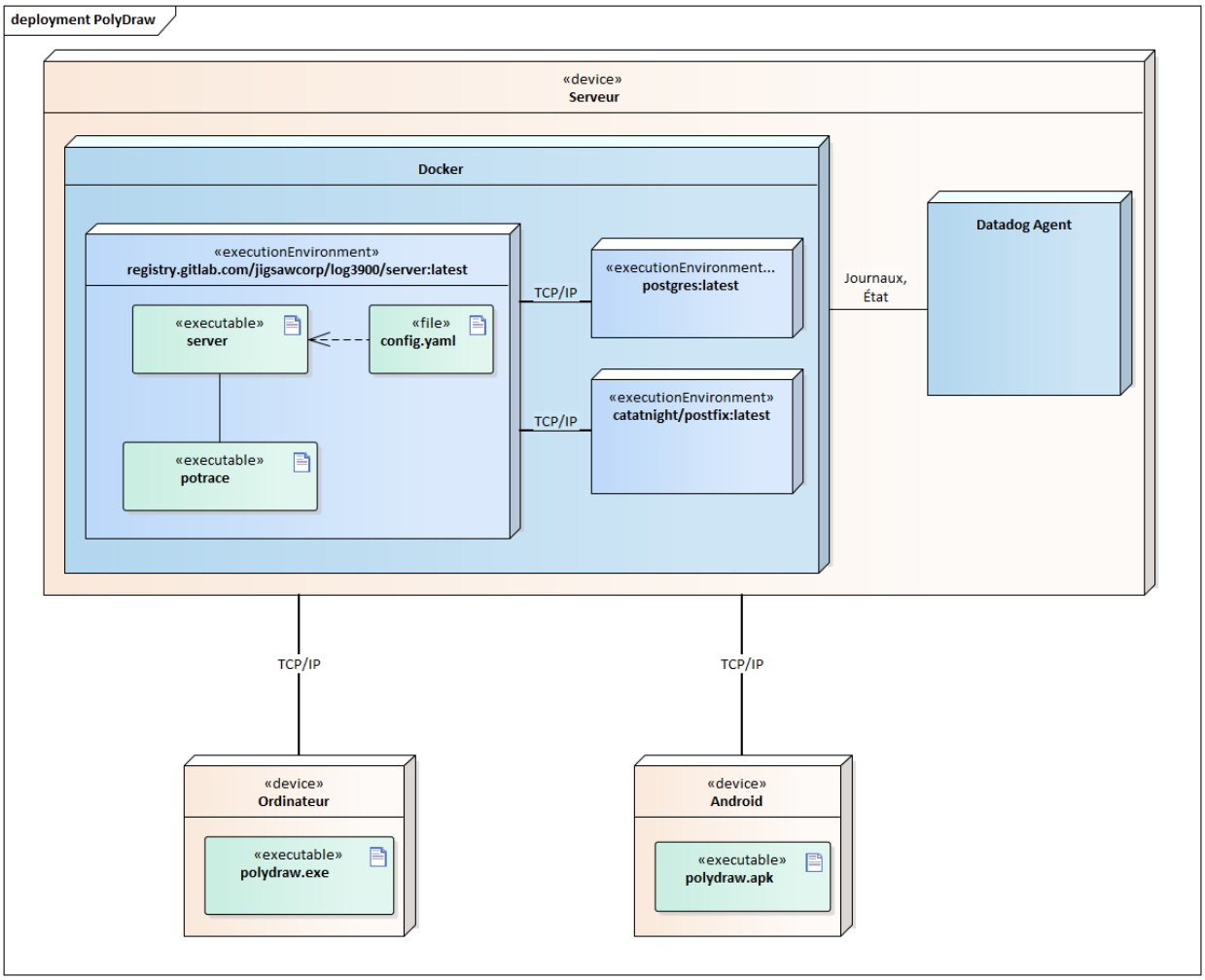

Figure 6.1: Diagramme de déploiement de PolyDraw

7. Taille et performance

La taille et performance doit être évaluée sur les trois plateformes, client léger, client lourd et serveur.

Dans le cas du client léger, il faut que l'architecture utilise moins de 2 Go de ram de la tablette Android. Il s'agit de la limite d'espace de mémoire imposée par la tablette. Idéalement l'application doit utiliser moins de 1 Go afin de s'assurer d'avoir une marge de manoeuvre. De plus au niveau du stockage, la tablette possède 32 Go. Il faut donc que la taille de l'exécutable soit inférieure à celle-ci. Considérant que l'application ne possède pas beaucoup de ressources, la taille de l'application devrait donc être inférieure à 200 Mo. Ceci permettra d'avoir une application qui est légère et facile à distribuer. De plus, la nature mobile de l'application fait qu'il faut que l'architecture soit robuste aux erreurs associées au réseau, par exemple déconnexion wifi, changement wifi au LTE etc. De plus, l'application doit gérer les cas où l'application est mise en pause ou autre.

Au niveau du client lourd, il n'a pas vraiment de limitation. Cependant, on peut affirmer que l'application doit rouler sur un ordinateur qui a au moins 4 Go de ram. De plus, l'utilisation des ressources doit être raisonnable et utiliser au maximum 25% des ressources du système. Il faut donc que l'application implémente des mécanismes pour ne pas utiliser les ressources au maximum. L'ordinateur ne doit pas nécessairement avoir une accélération graphique puisque, le canevas du .NET Framework est utilisé. La taille de l'exécutable ne devrait pas dépasser 200 Mo. Ceci a pour but de réduire la taille des artefacts et faciliter la distribution de l'exécutable.

Finalement, le serveur est une machine virtuelle. Cette machine virtuelle possède 4 Go de ram et 4 coeurs virtuels. Il y a également d'autres services qui roulent sur le serveur comme un agent de surveillance, un SGBD, et un MTA. Il est donc réaliste de dire que le serveur devrait utiliser moins de 512 Mo ram avec moins de 8 connexions. De plus, l'utilisation du processeur devrait être réduite le plus possible dans le but de ne pas monopoliser les ressources des autres processus. Dans le cas, où l'architecture ne respecterait pas ces critères, l'agent de surveillance va avertir l'équipe d'un problème avec le serveur.

Cette architecture permet donc de satisfaire le besoin d'avoir un logiciel stable et polyvalent. Les façons d'évaluer la fiabilité du logiciel ont été établies dans le SRS.