Équipe 203

PolyDraw
Document d'architecture logicielle

Version 1.3

Historique des révisions

Date	Version	Description	Auteur
2020-02-07	1.0	Document d'architecture pour le prototype	Allan Beddouk Philippe Côté-Morneault Pascal Alexandre-Morel Martin Pouliot Samuel Saito-Gagné Cédric Tessier
2020-04-09	1.1	Changements de quelques exigences suite à la complétion du projet.	Philippe Côté-Morneault
2020-04-11	1.2	Mise à jour de l'architecture du client léger, client lourd et serveur	Philippe Côté-Morneault Cédric Tessier Martin Pouliot
2020-04-11	1.3	Correction des fautes de français	Philippe Côté-Morneault

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 objectifs. 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 c'est donc une contrainte. 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.

L'utilisation de WPF implique que le client lourd va utiliser le patron MVVM. Celle-ci n'est pas nécessaire, mais très encouragé avec WPF. 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.

L'utilisation d'Android implique d'utiliser le cycle de vie de l'application. Cette façon de développer aura un effet 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 vont donc affecter 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

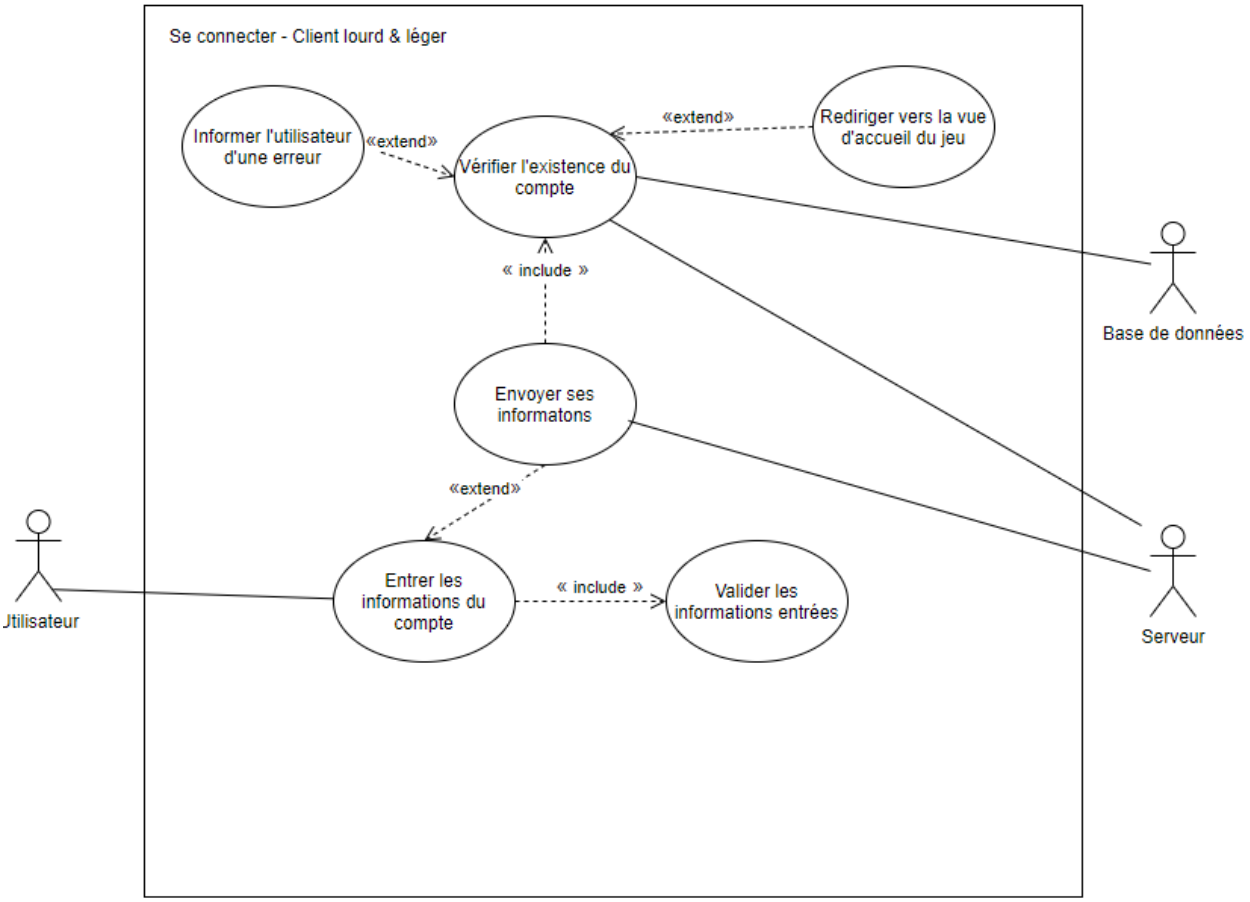

Figure 3.1: Présentation du cas d'utilisation « se connecter »

3.2 Créer un profil d'utilisateur

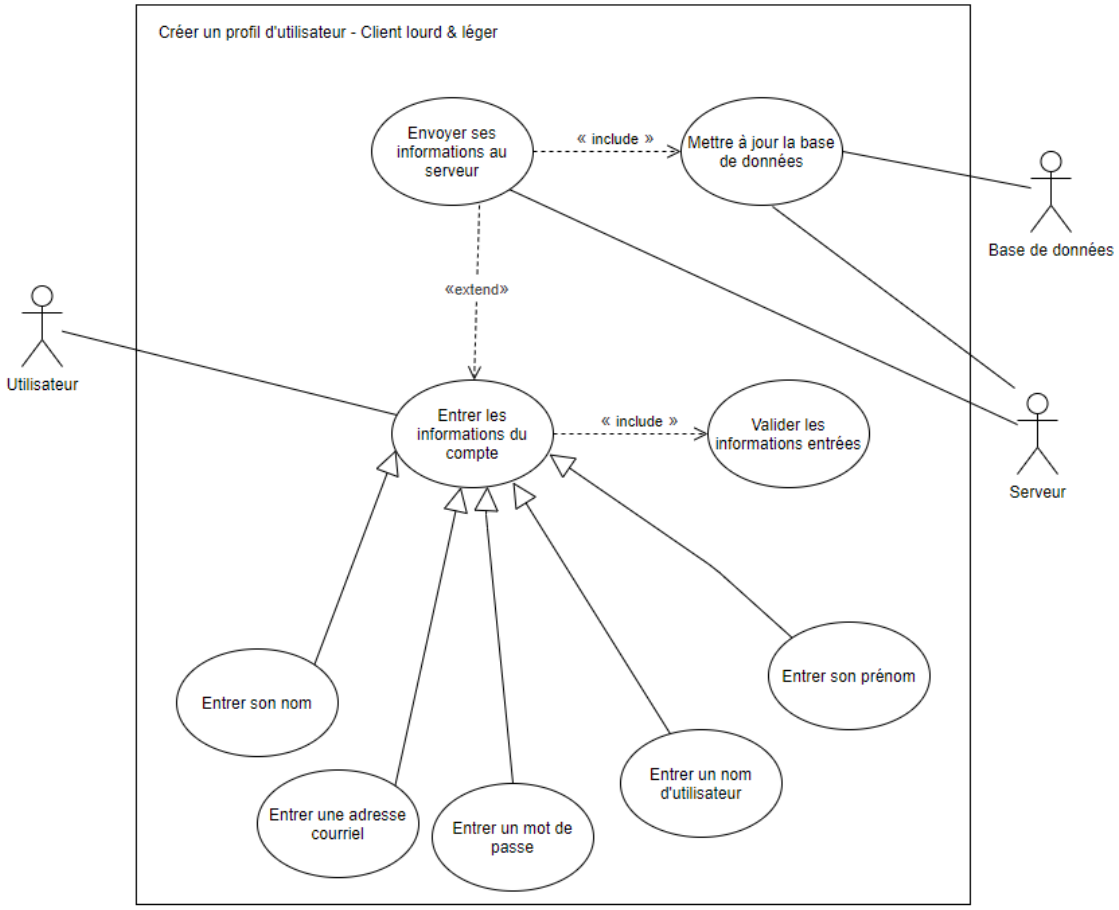

Figure 3.2: Présentation du cas d'utilisation « créer un profil d'utilisateur »

3.3 Clavarder

Pour le client lourd, l'application permet aussi de choisir le mode de la fenêtre de clavardage (mode intégré ou fenêtré).

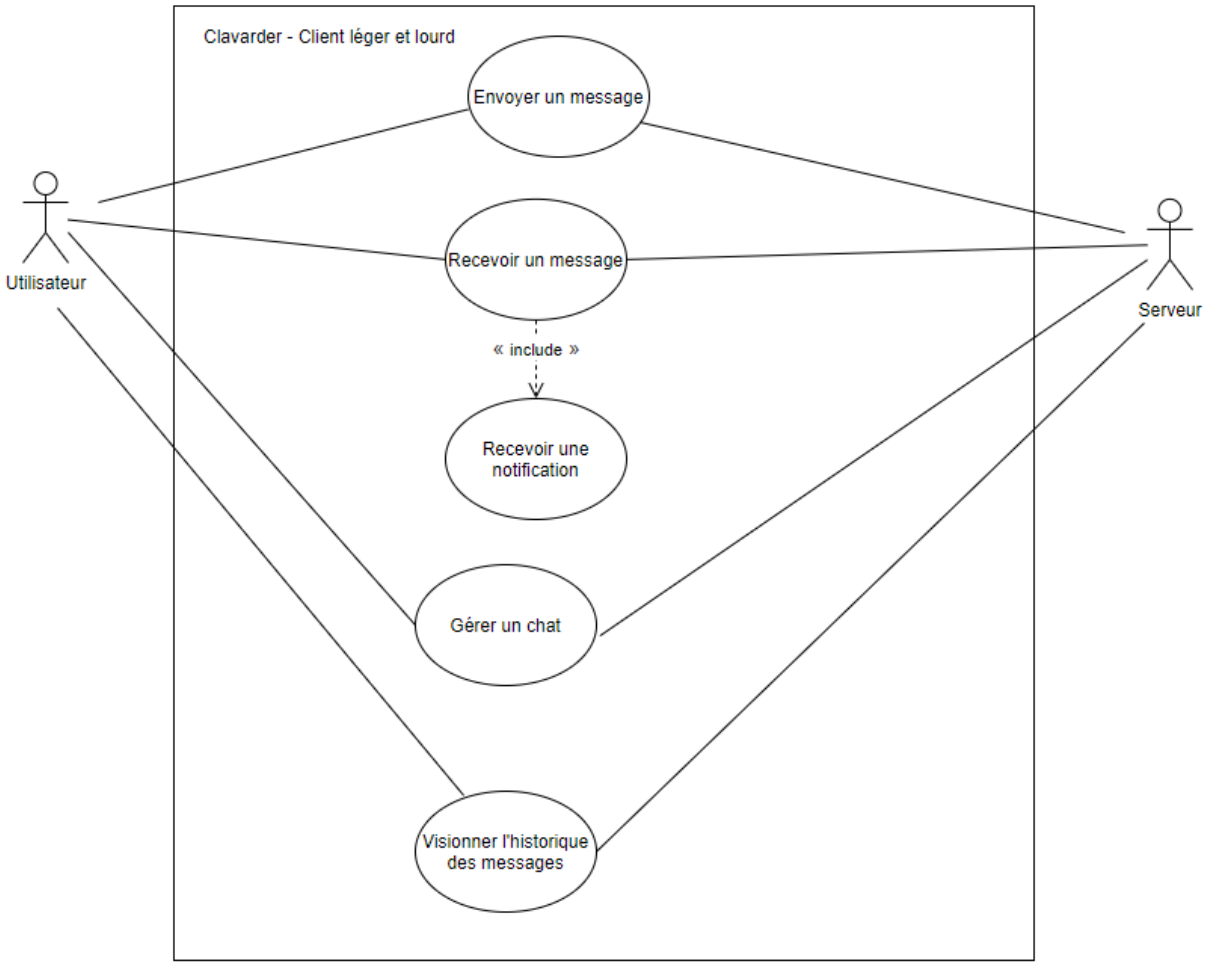

Figure 3.3: Présentation du cas d'utilisation « clavarder » pour le client léger et lourd.

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.

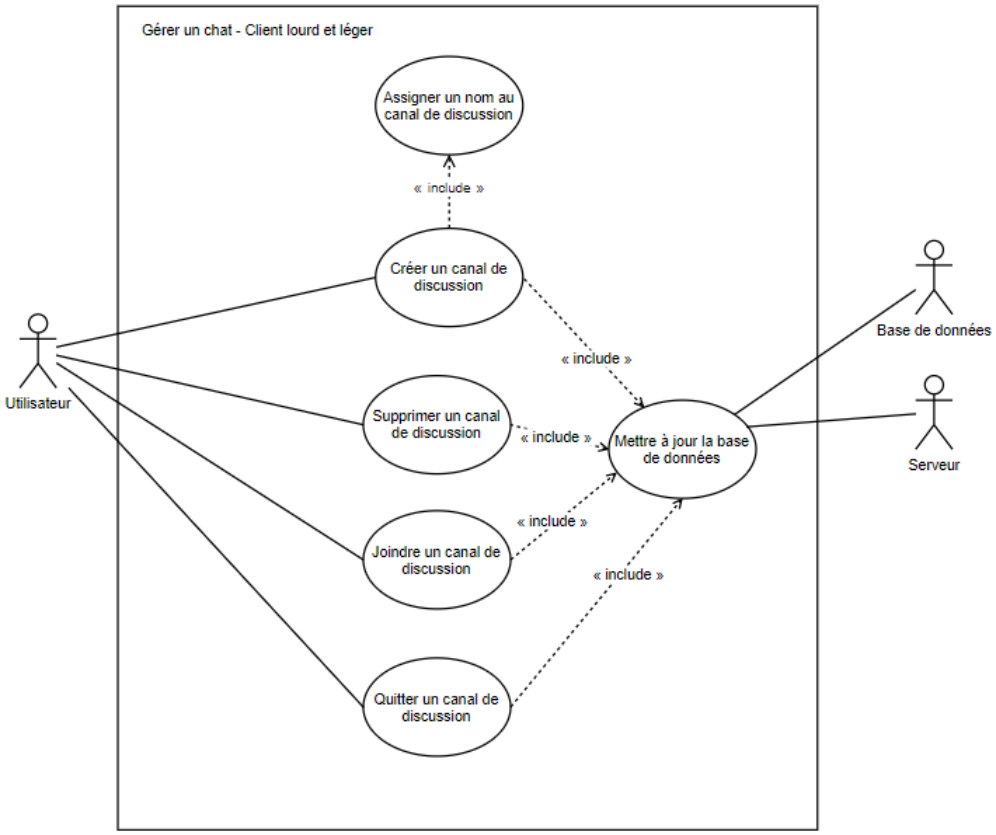

Figure 3.4: Présentation du cas d'utilisation « gérer un *chat* »

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

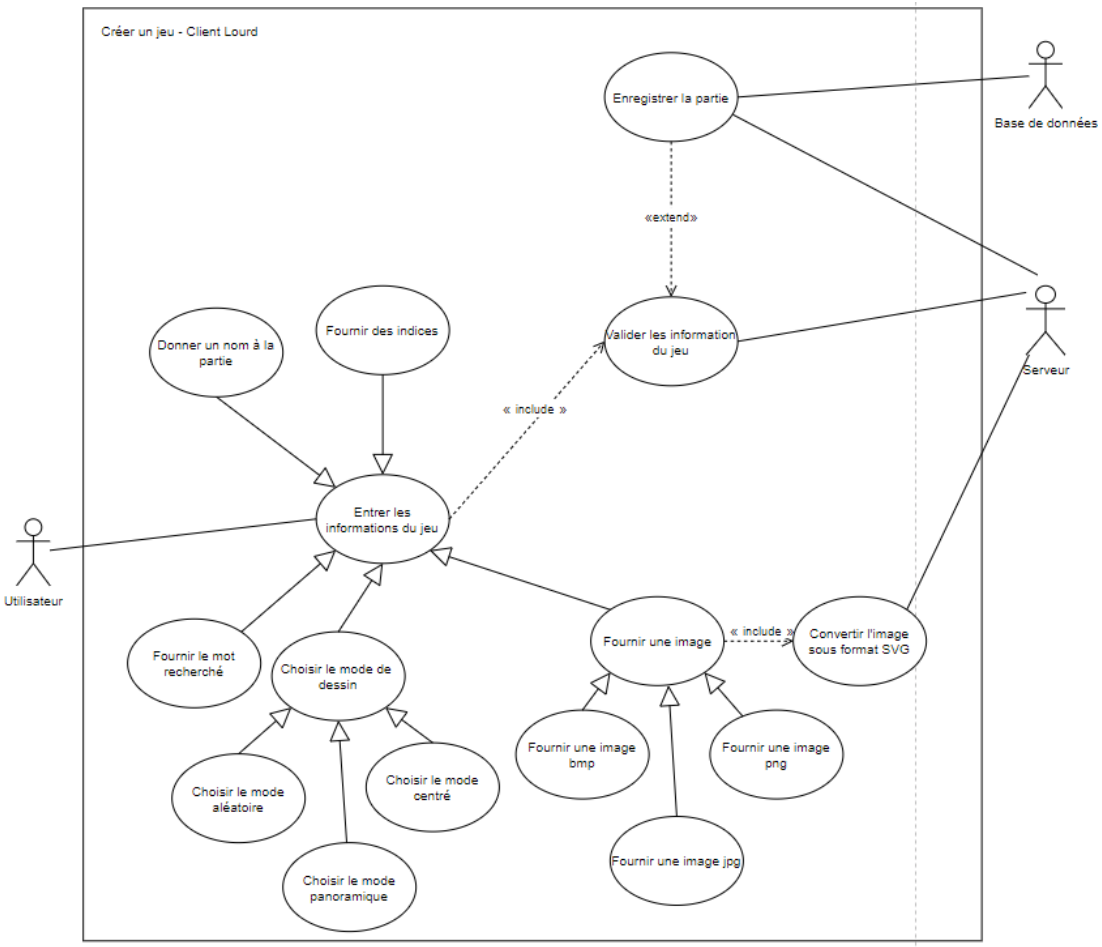

Figure 3.5.1: Présentation du cas d'utilisation « créer un jeu » pour le client lourd

3.5.2 Client léger

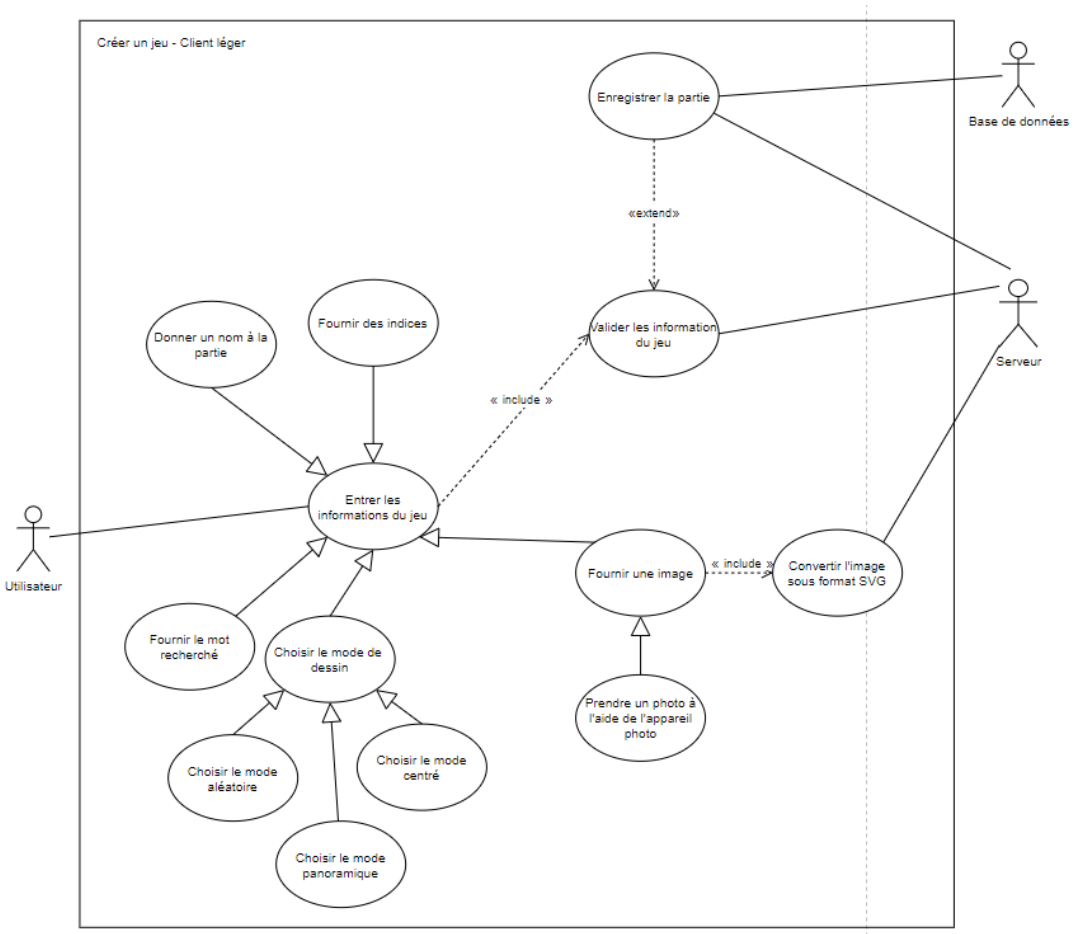

Figure 3.5.2: Présentation du cas d'utilisation « créer un jeu » pour le client léger

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. Cette fonctionnalité a été retirée en raison des deux semaines de retrait.

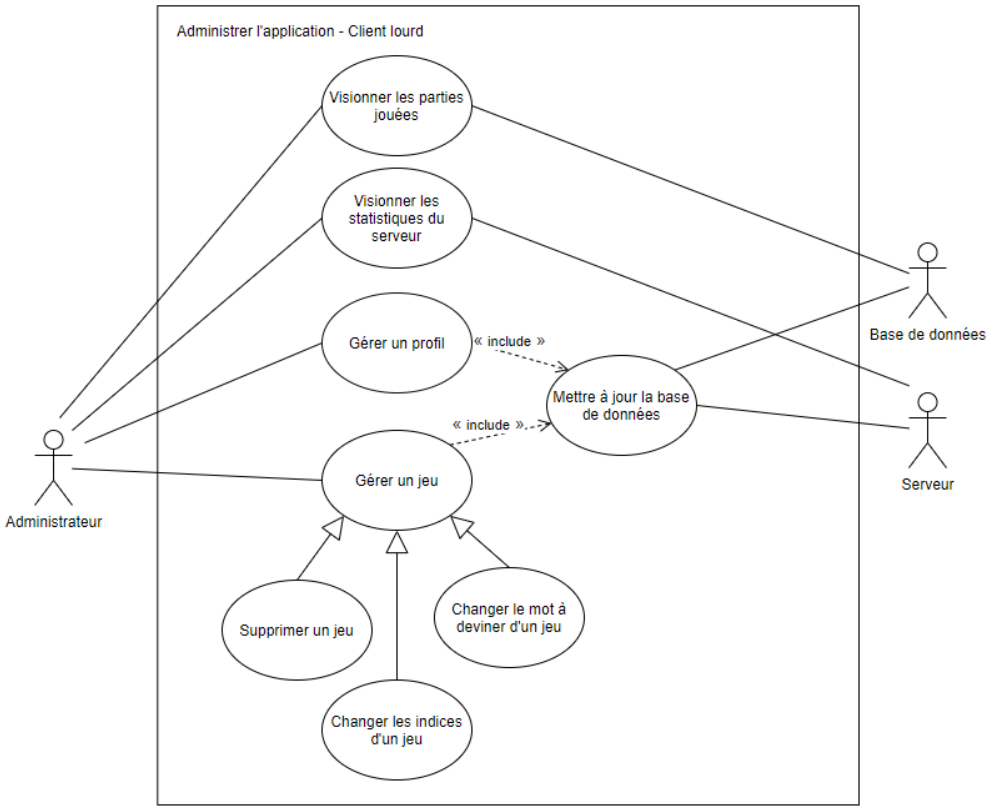

Figure 3.6: Présentation du cas d'utilisation « administrer l'application » pour le client lourd

3.7 Gérer un profil

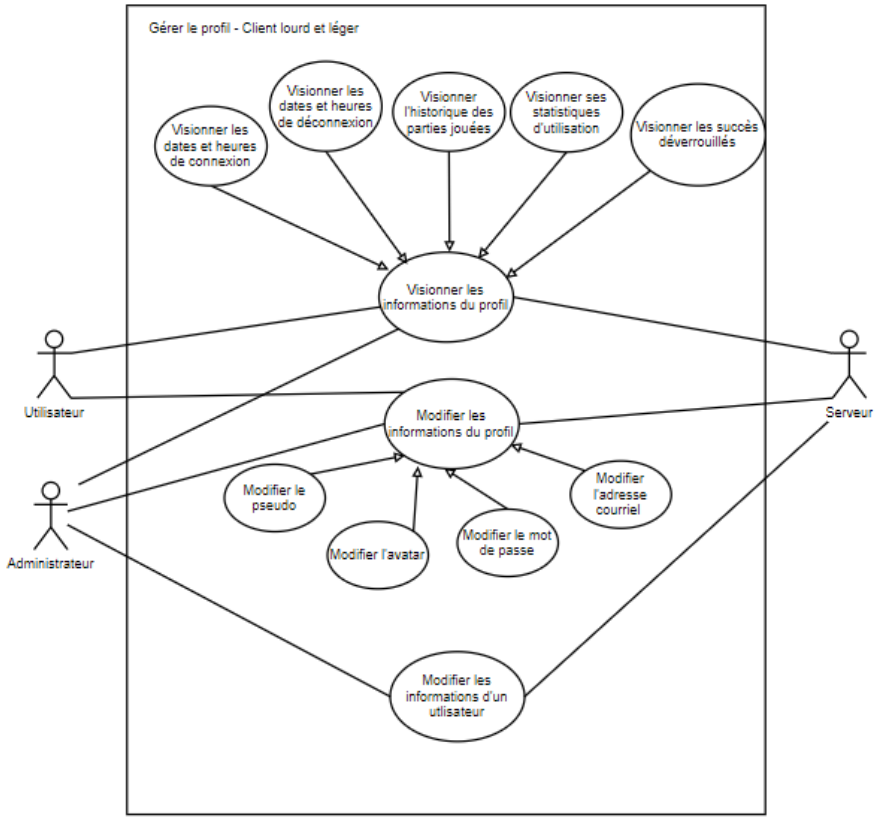

Figure 3.7: Présentation du cas d'utilisation « gérer un profil »

3.8 Créer une partie (lobby)

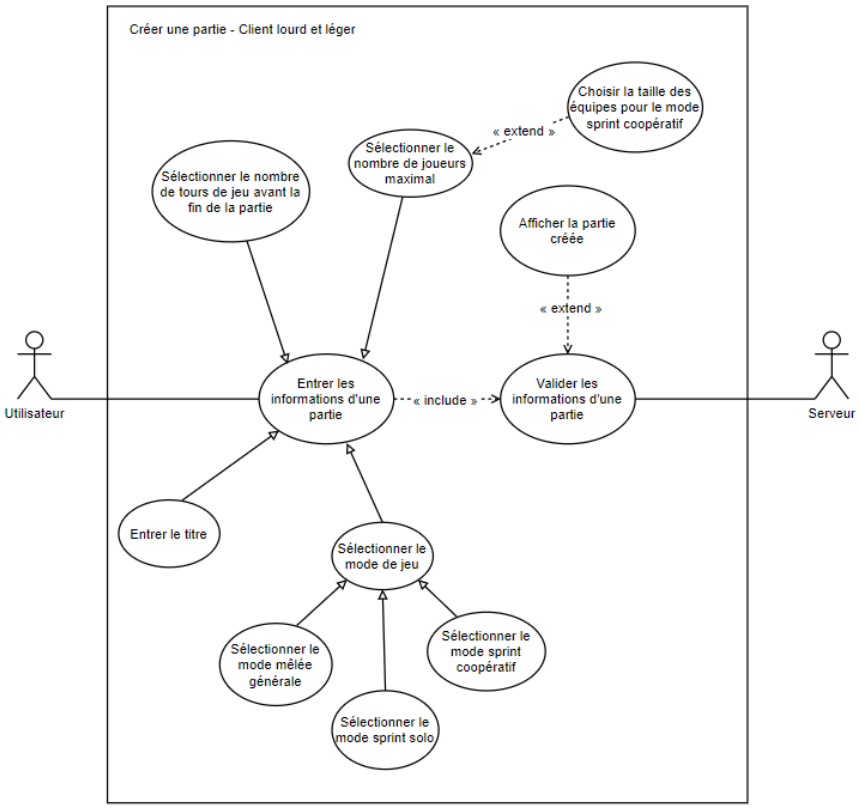

Figure 3.8: Présentation du cas d'utilisation « créer une partie »

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. En effet, pour le client léger, il est possible de choisir le thème de l'application.

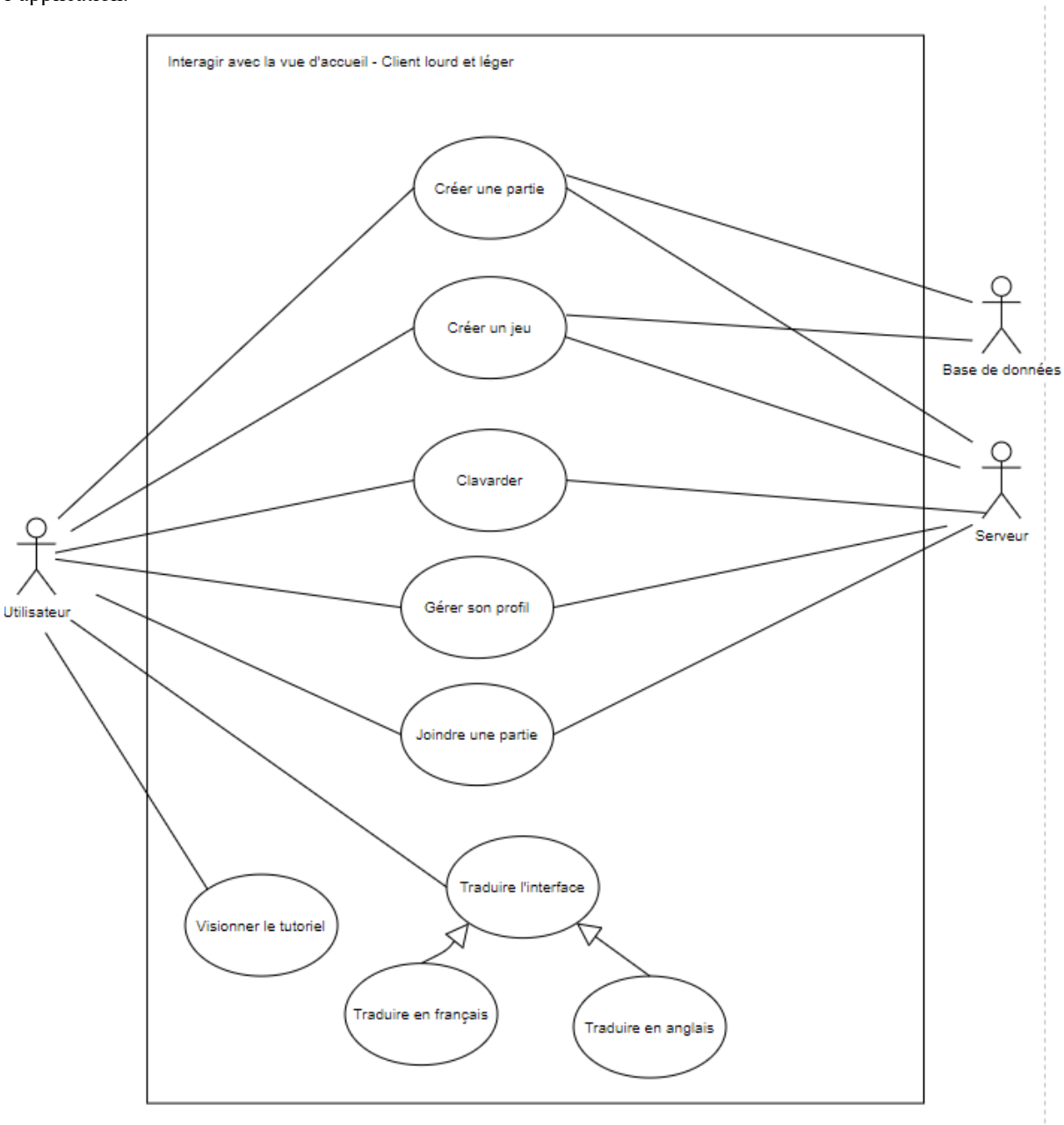

Figure 3.9: Présentation du cas d'utilisation « interagir avec la vue d'accueil » pour le client lourd et léger

3.10 Jouer à une partie en mode mêlée générale

Figure 3.10: Présentation du cas d'utilisation « jouer à une partie en mode mêlée générale »

3.11 Jouer à une partie en mode solo

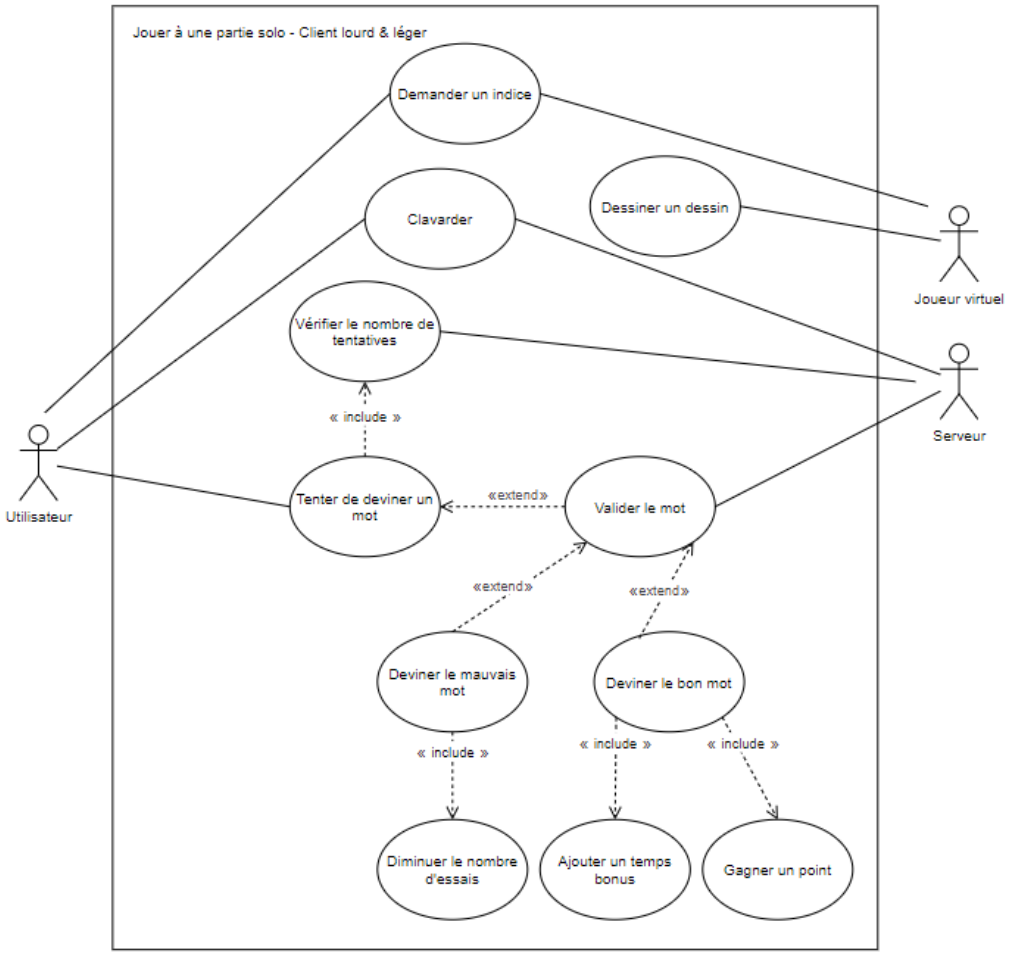

Figure 3.11: Présentation du cas d'utilisation « jouer à une partie en mode solo »

3.12 Jouer à une partie en mode coopératif

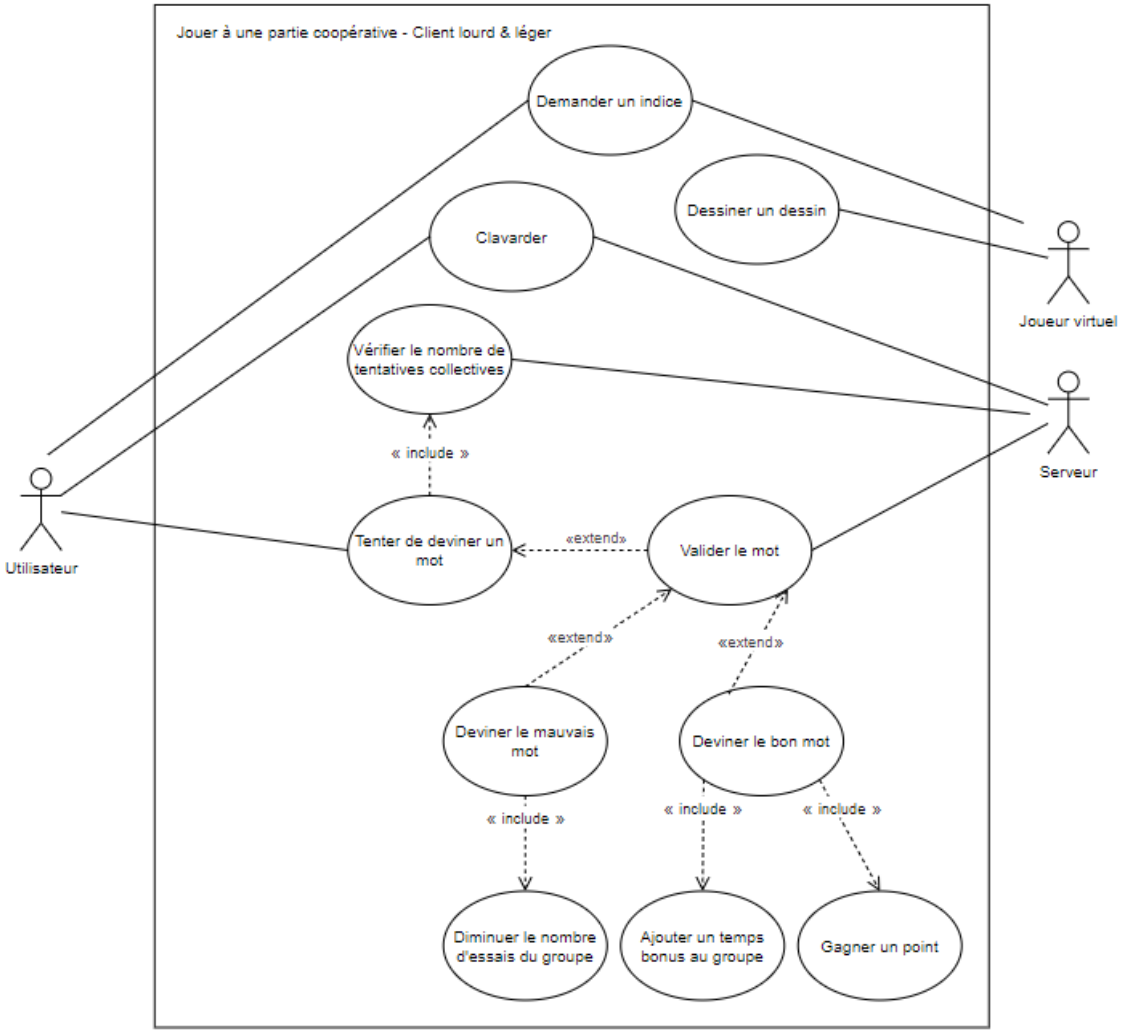

Figure 3.12: Présentation du cas d'utilisation « jouer à une partie en mode coopératif »

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

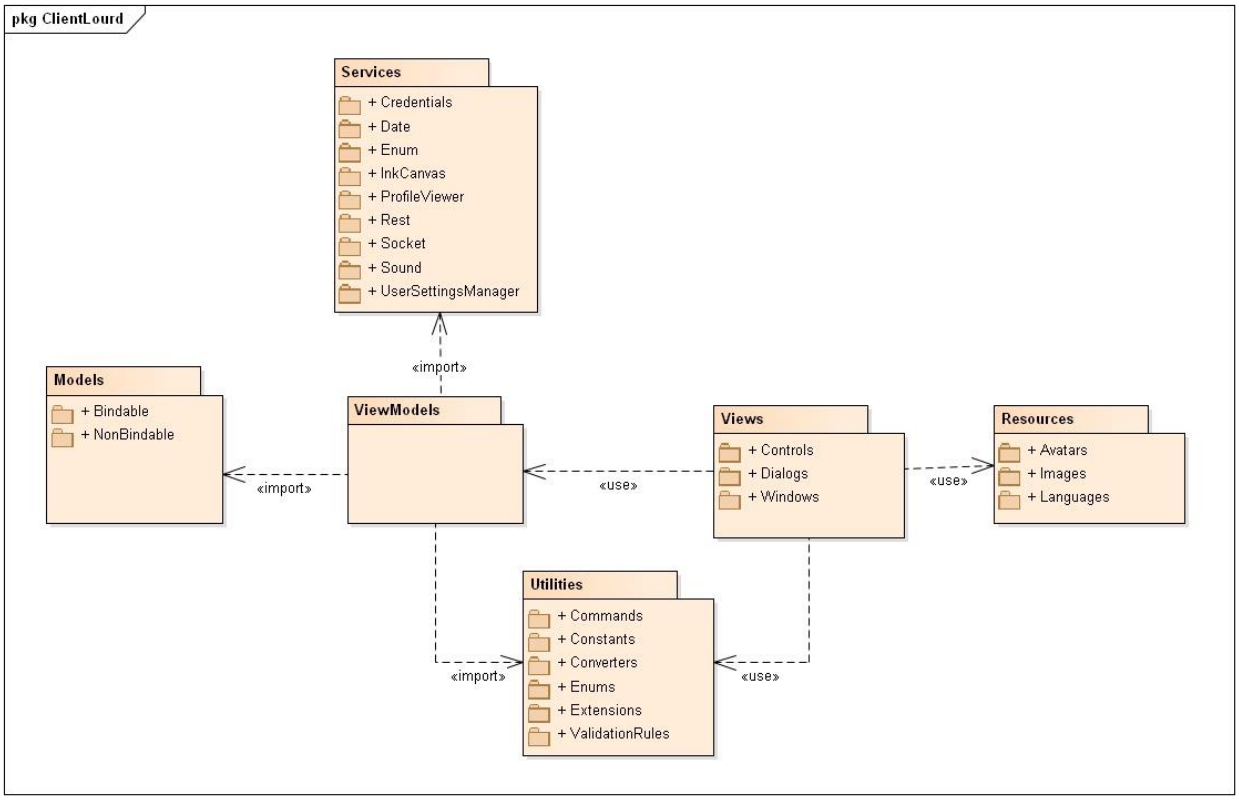

Figure 4.1.1: Diagramme de paquetage ClientLourd

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.

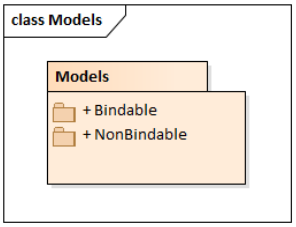

Figure 4.1.1.1: Diagramme de paquetage ClientLourd::Models

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.

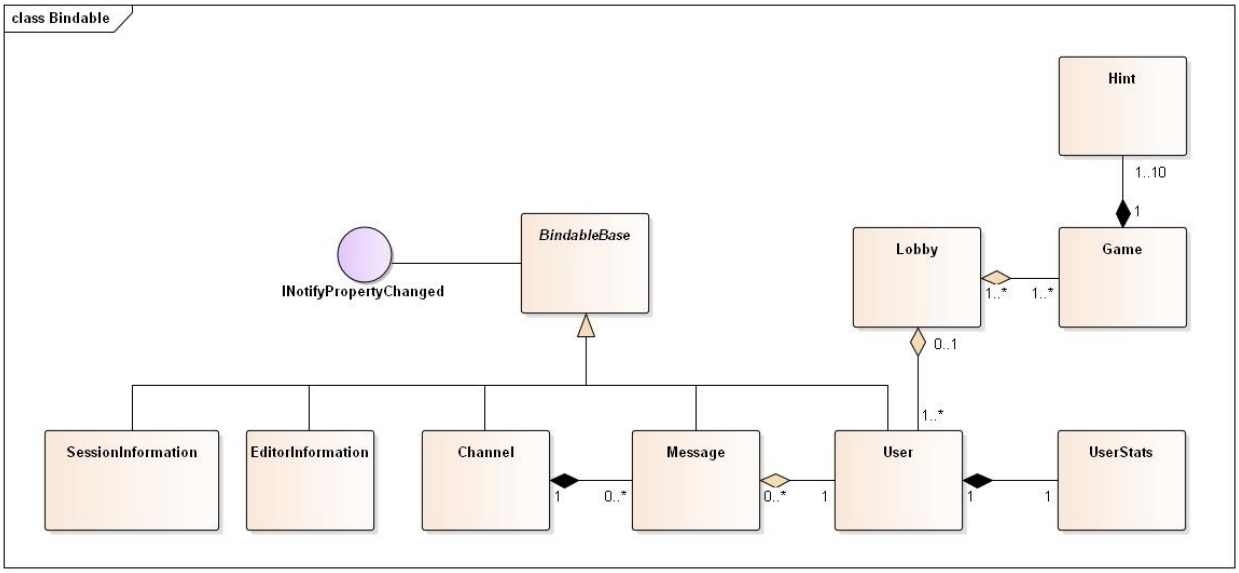

Figure 4.1.1.2: Diagramme de classe ClientLourd::Models::Bindable

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. TLV (type length value) est le modèle utilisé pour les messages socket.

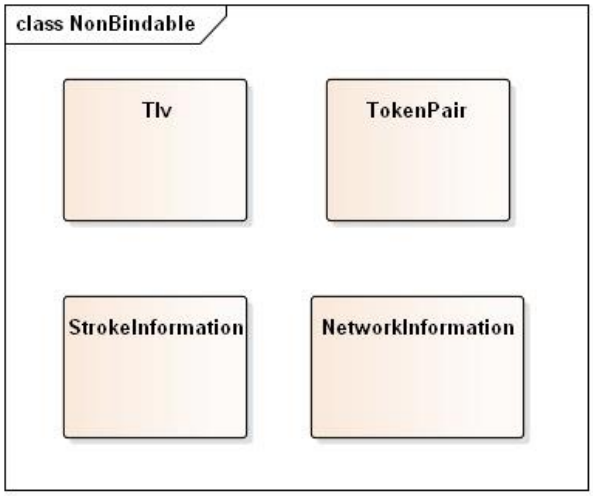

Figure 4.1.1.3: Diagramme de classe ClientLourd::Models::NonBindable

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

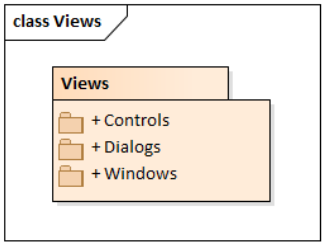

Figure 4.1.2.1: Diagramme de paquetage ClientLourd::Views

ClientLourd::Views::Controls
Ce paquetage a pour responsabilité d'offrir l'accès aux différents contrôles utilisateurs (UserControl) personnalisés

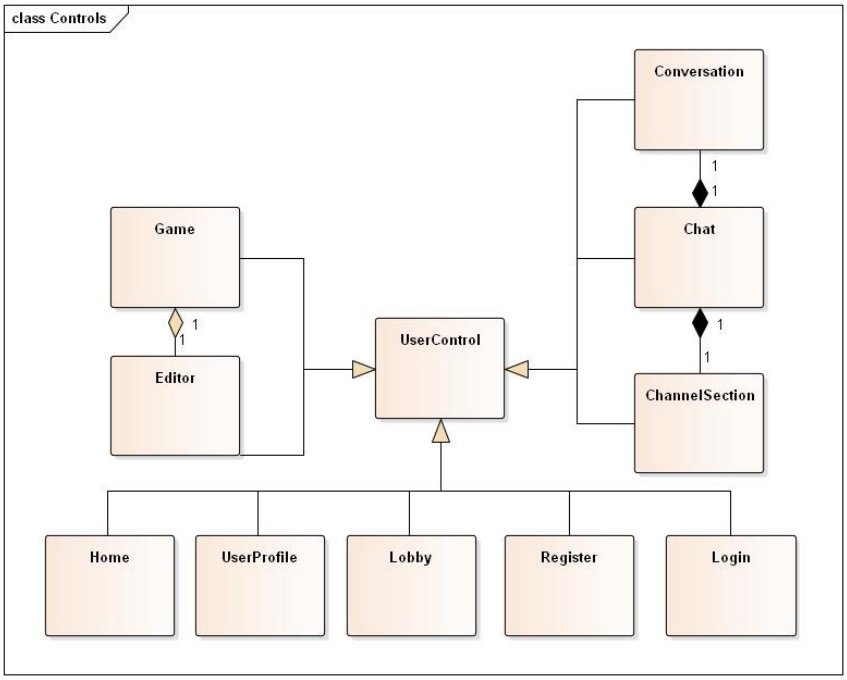

Figure 4.1.2.2: Diagramme de classe ClientLourd::Views::Controls

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

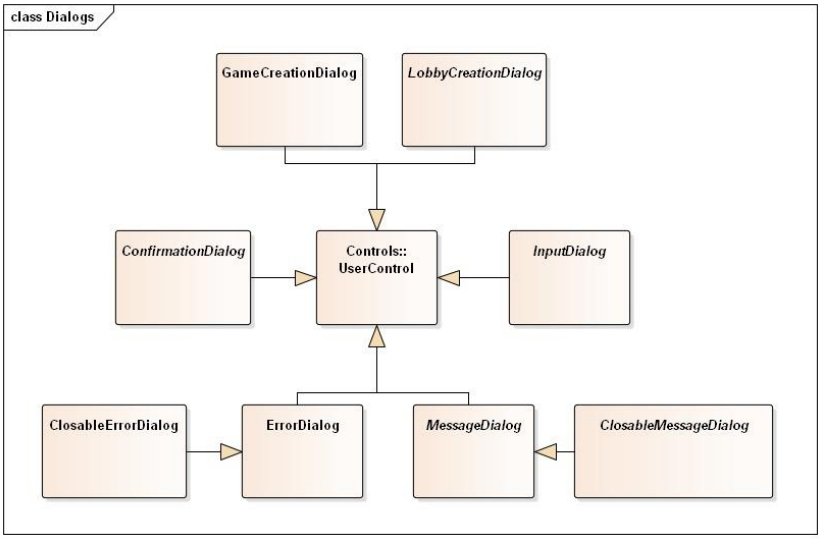

Figure 4.1.2.3: Diagramme de classe ClientLourd::Views::Dialogs

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

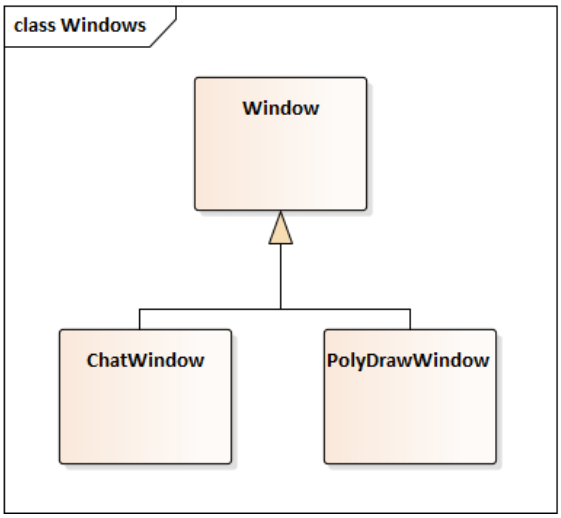

Figure 4.1.2.4: Diagramme de classe ClientLourd::Views::Windows

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

4.1.4 Services

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

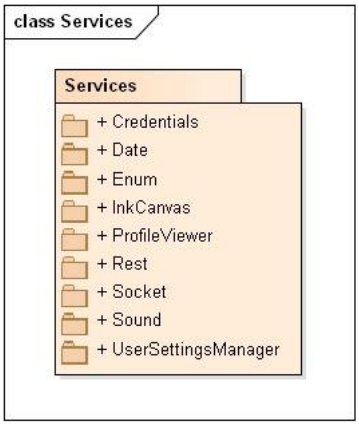

Figure 4.1.4.1: Diagramme de paquetage ClientLourd::Services

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

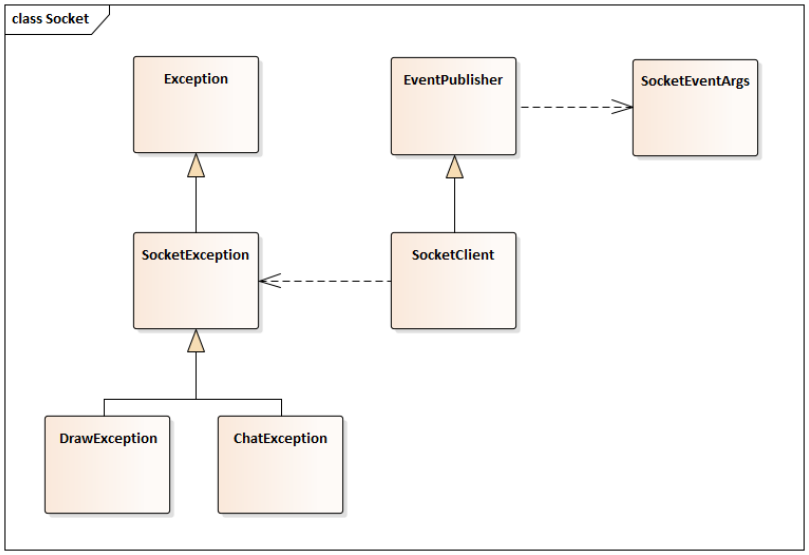

Figure 4.1.4.2: Diagramme de classe ClientLourd::Services::Socket

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

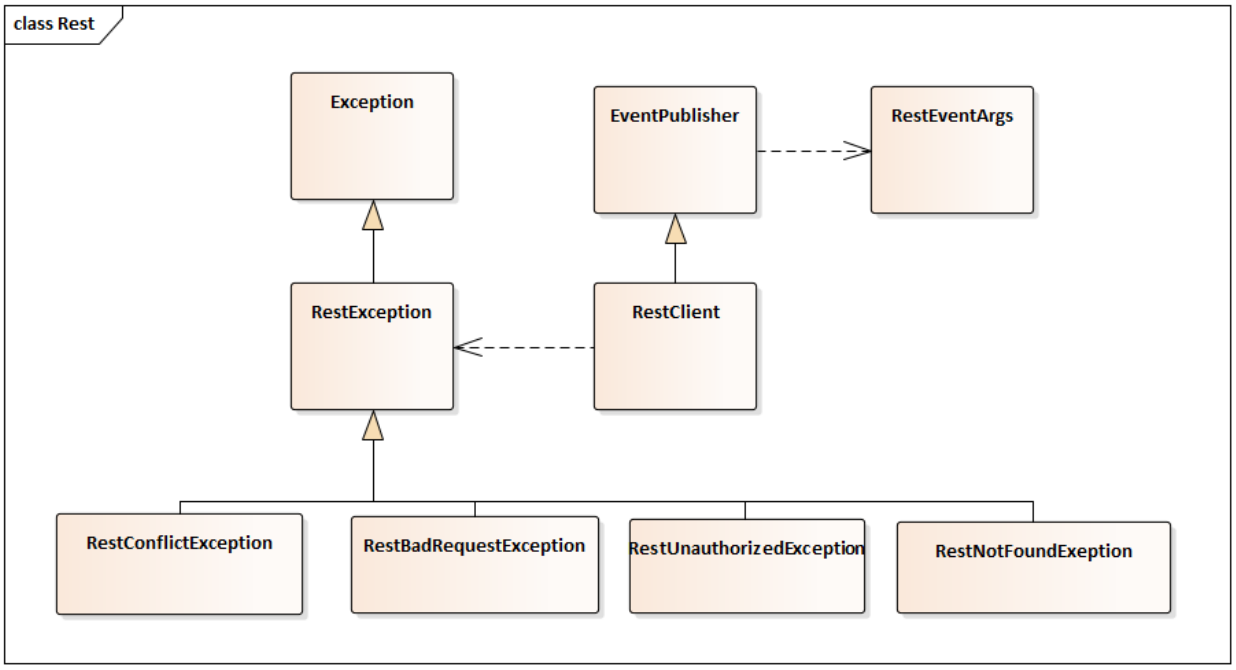

Figure 4.1.4.3: Diagramme de classe ClientLourd::Services::Rest

ClientLourd:Services:InkCanvas
Ce paquetage a pour responsabilité d'implémenter les classes et la logique pour gérer l'envoie et la réception de trait sur le réseau.

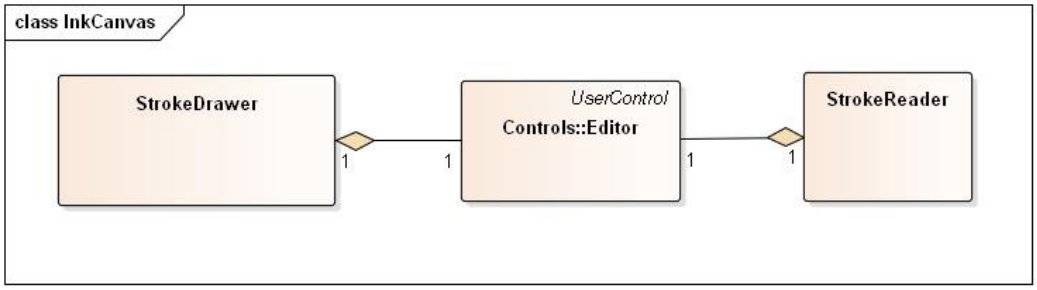

Figure 4.1.4.4: Diagramme de classe ClientLourd::Services::InkCanvas

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.

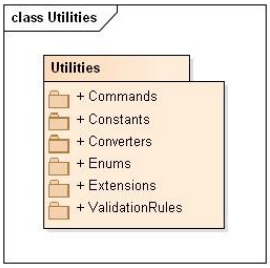

Figure 4.1.5.1: Diagramme de paquetage ClientLourd::Utilities

ClientLourd::Utilities:Enums
Ce paquetage a pour responsabilité d'offrir l'accès aux Enums de l'application. Ces valeurs ainsi que leur signification est définie dans le protocole de communication. Pour l'enum SocketMessageTypes, Les valeurs paires sont associées à un message d'envoi tandis que les valeurs impaires sont associées à un message de réception. Certaines valeurs sont manquantes pour simplifier l'ajout de message au besoin.

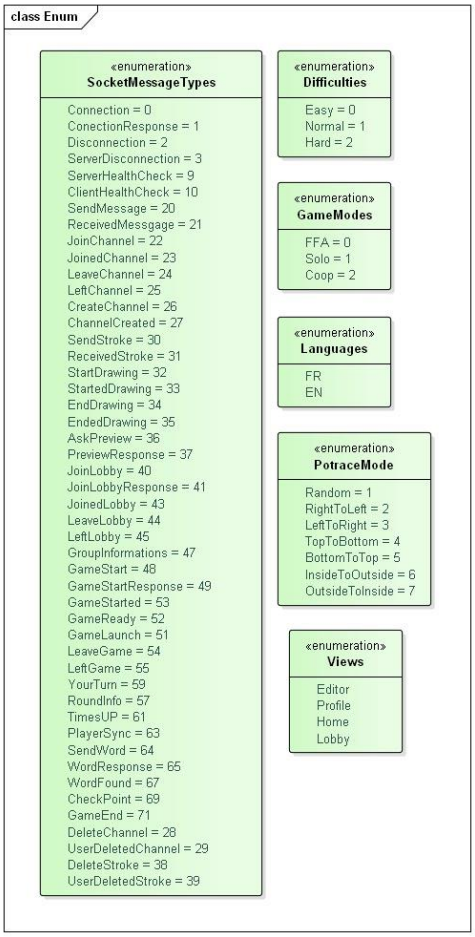

Figure 4.1.5.2: Diagramme de classe ClientLourd::Utilities::Enums

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

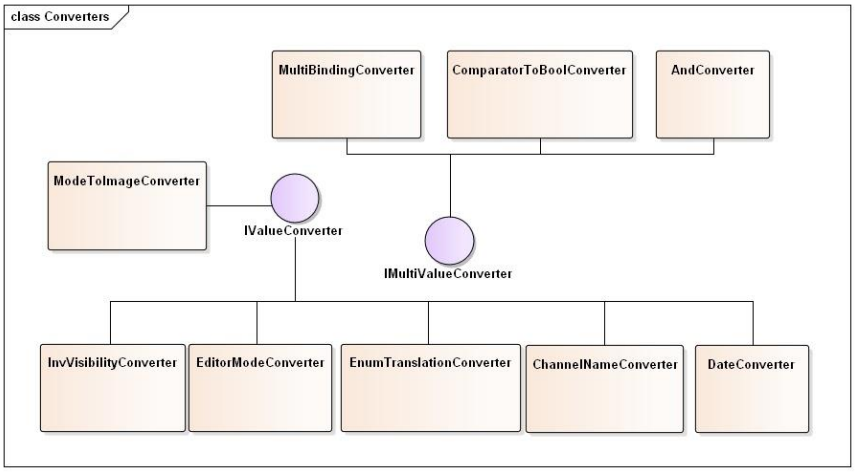

Figure 4.1.5.3: Diagramme de classe ClientLourd::Utilities::Converters

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

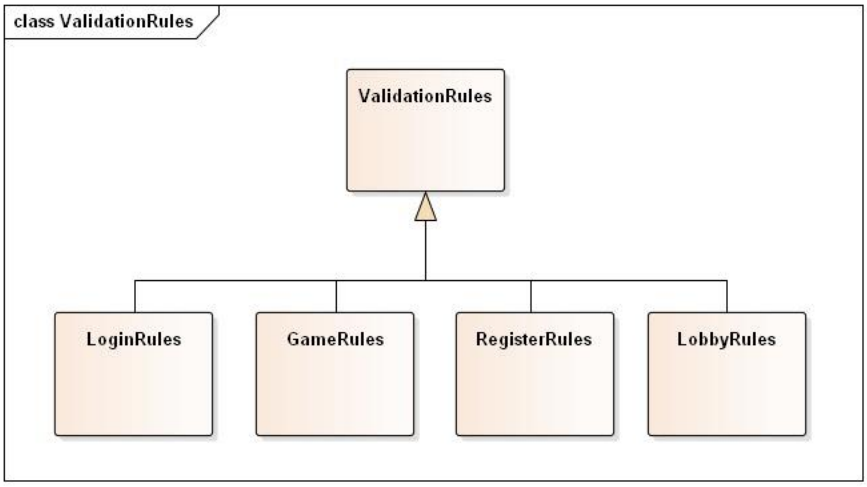

Figure 4.1.5.4: Diagramme de classe ClientLourd::Utilities::ValidationRules

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

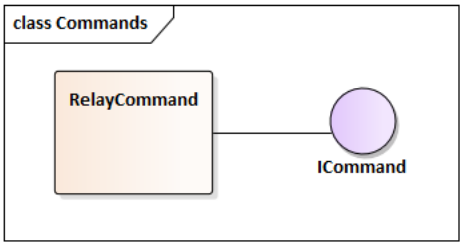

Figure 4.1.5.5: Diagramme de classe ClientLourd::Utilities::Commands

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

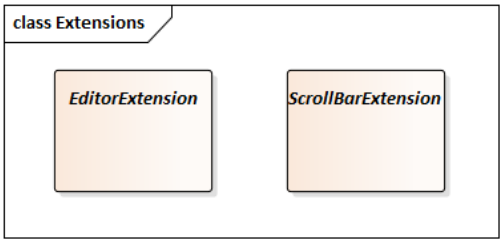

Figure 4.1.5.6: Diagramme de classe ClientLourd::Utilities::Extensions

ClientLourd::Utilities::Constants
Ce paquetage a pour responsabilité de contenir les différentes constantes de l'application

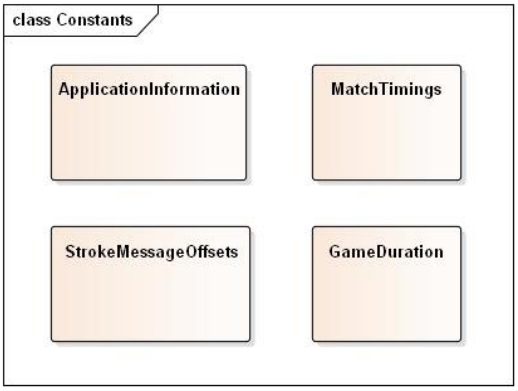

Figure 4.1.5.7: Diagramme de classe ClientLourd::Utilities::Constants

4.1.6 Resources

ClientLourd::Resources
Ce paquetage a pour responsabilité de contenir les différentes ressources utilisées par les autres paquetages (images, icônes, etc.). Ce package contient également les deux dictionnaires utilisés pour la traduction de l'application.

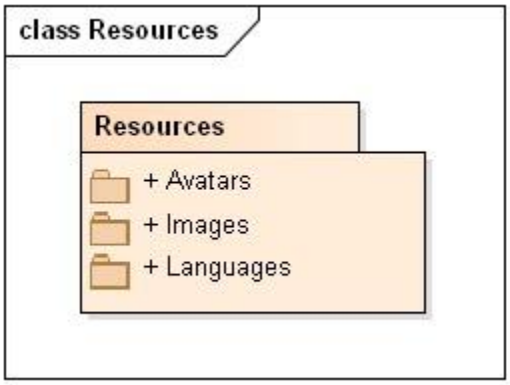

Figure 4.1.6: Diagramme de paquetage ClientLourd::Resources

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 les paquetages *shared*, *Resources*, *Utils* et *Socket*, mais les traits sur le diagramme sont omis pour fin de clarté.

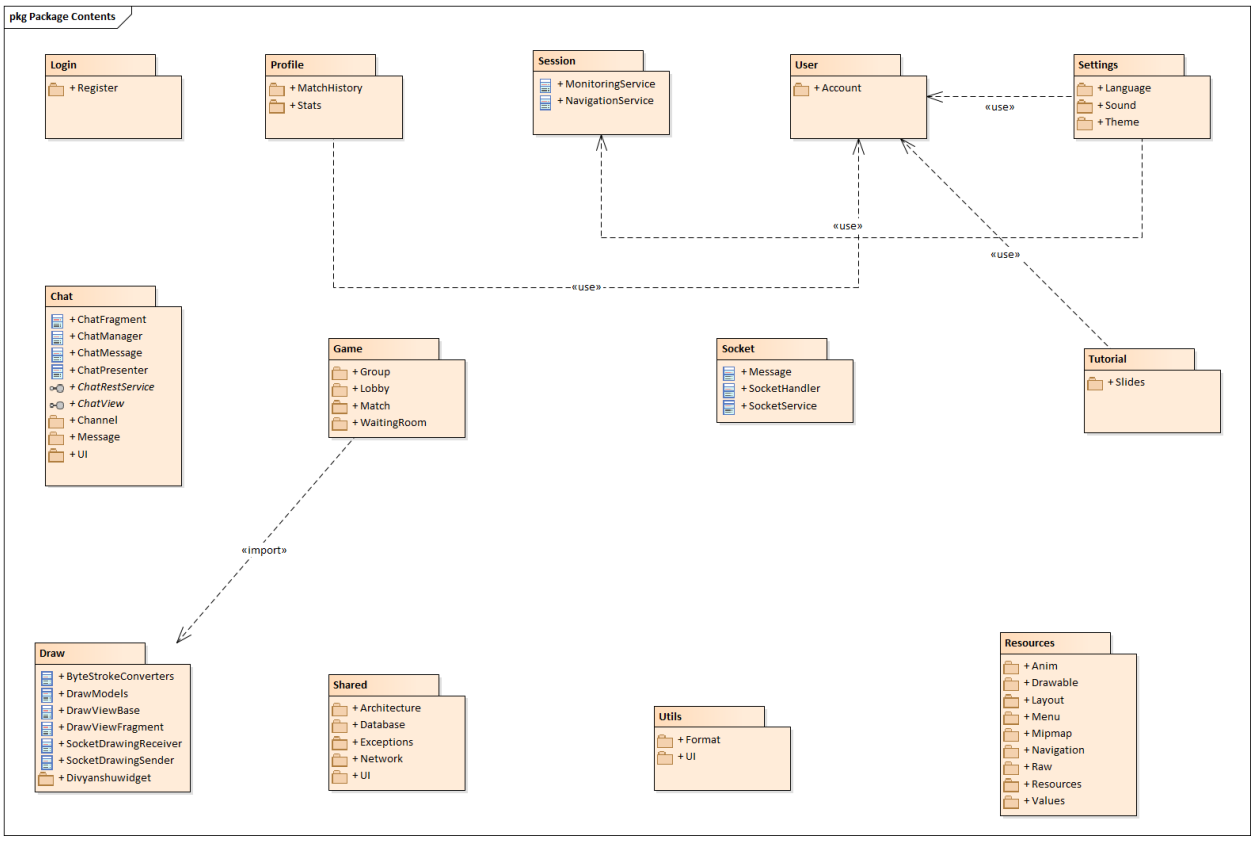

Figure 4.2: Diagramme de paquetage ClientLéger

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.

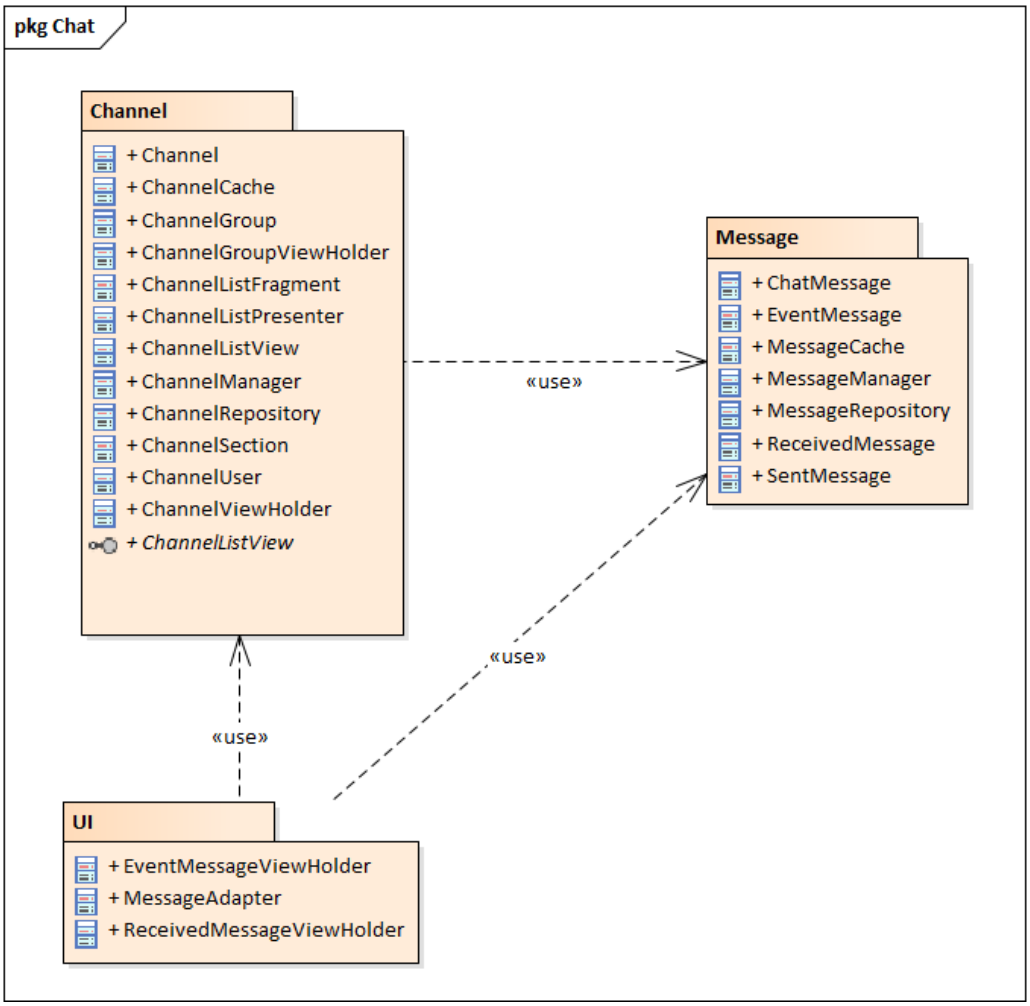

Figure 4.2.1: Diagramme de paquetage ClientLéger::Chat

4.2.1.1 Channel

ClientLéger::Chat::Channel
Ce paquetage regroupe toutes les fonctionnalités pour les canaux de communication. On y retrouve les fonctionnalités pour faire la communication réseau des canaux, pour afficher les canaux, pour créer des canaux, etc.

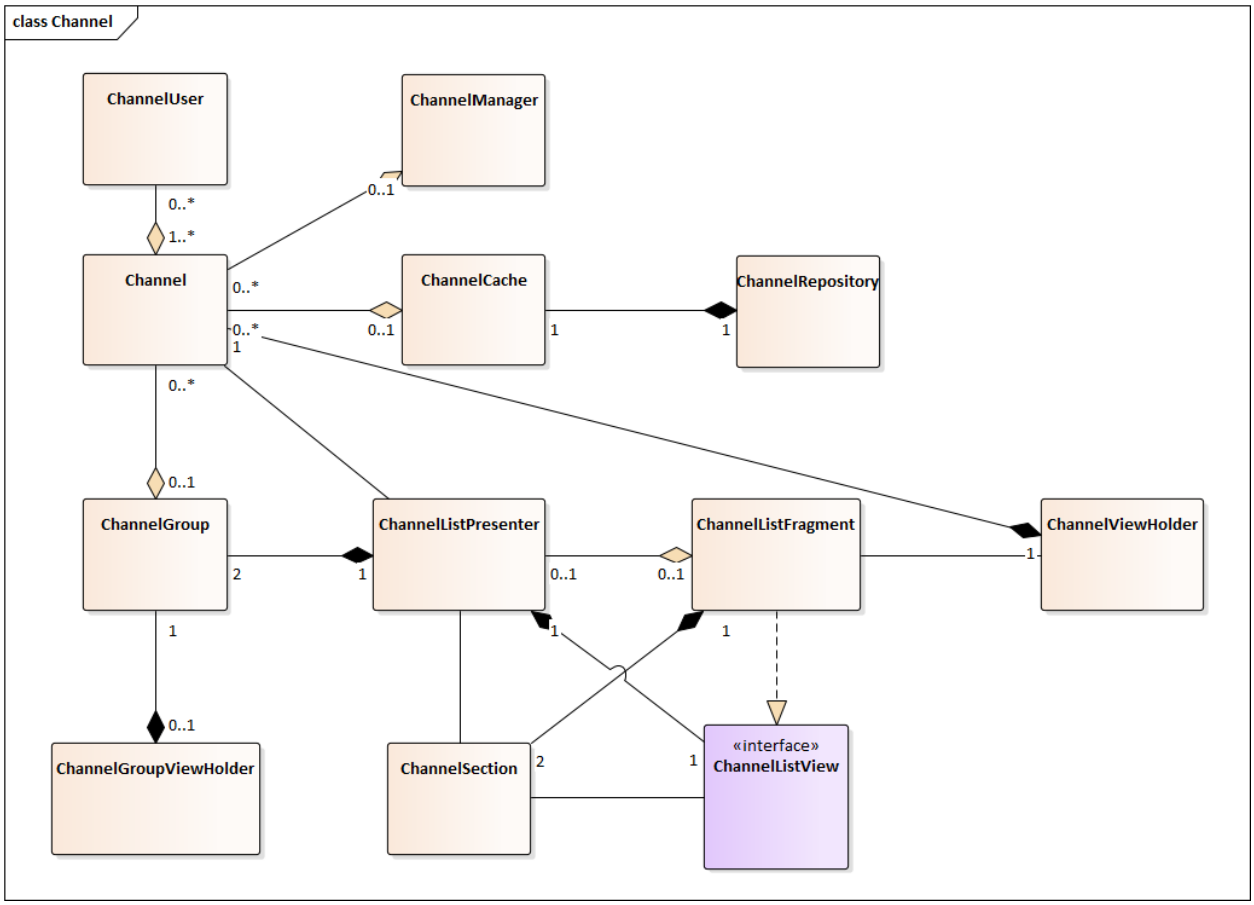

Figure 4.2.1.1: Diagramme de paquetage ClientLéger::Chat::Channel

4.2.1.2 Message

ClientLéger::Chat::Message
Ce paquetage regroupe toutes les fonctionnalités pour les messages chat. On y retrouve les fonctionnalités pour faire la communication réseau des messages, pour formater les messages et pour gérer les multiples messages des différents canaux.

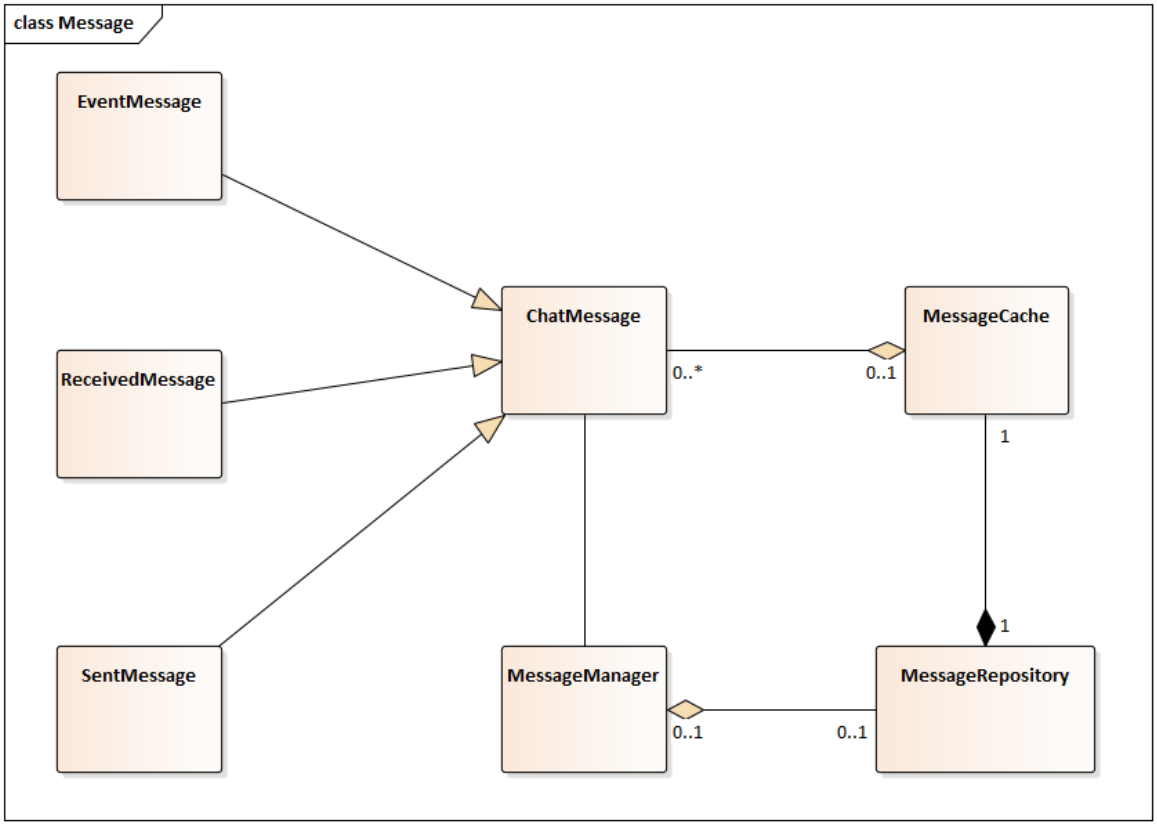

Figure 4.2.1.2: Diagramme de paquetage ClientLéger::Chat::Message

4.2.1.3 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, ainsi que les conteneurs de messages et canaux.

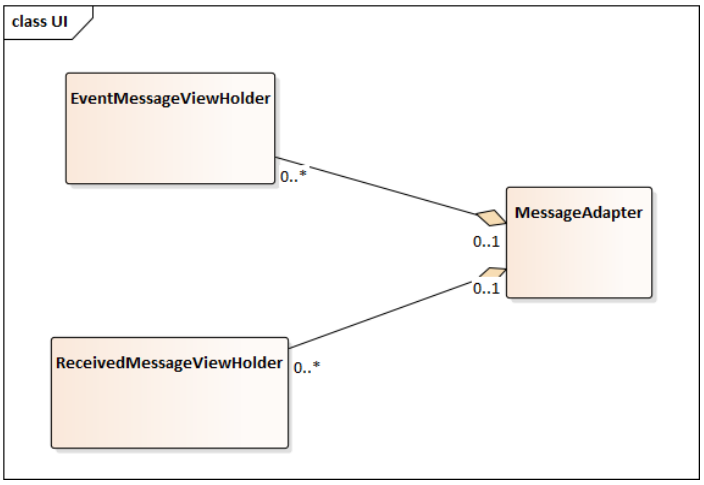

Figure 4.2.1.3: Diagramme de classe ClientLéger::Chat::UI

4.2.2 Draw

ClientLéger::Draw
Ce paquetage regroupe les fonctionnalités utilisées pour dessiner sur un canevas ainsi qu'afficher des traits provenant du serveur. Ce paquetage contient un paquetage, *Divyanshuwidget*, qui est le code d'une librairie externe qui a été légèrement adapté pour notre utilisation.

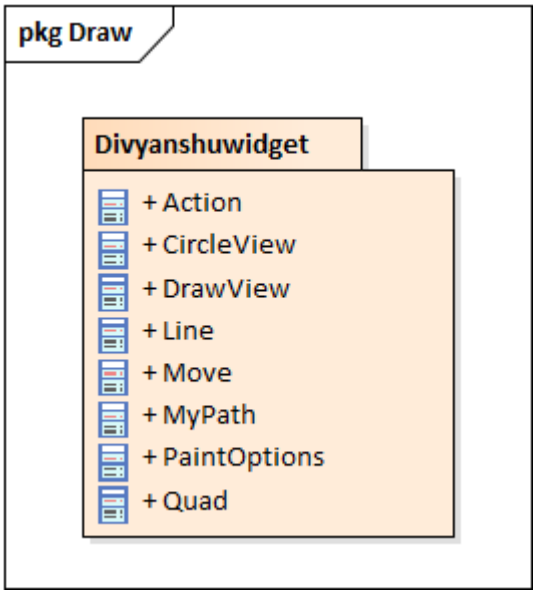

Figure 4.2.2: Diagramme de classe ClientLéger::Draw

4.2.3 Game

ClientLéger::Game
Ce paquetage regroupe les fonctionnalités utilisées pour les parties. On y retrouve quatre paquetages. *Group* est un paquetage qui contient les fonctionnalités liées à des groupes d'utilisateurs en attente de jouer une partie. *Lobby* est un paquetage qui contient les fonctionnalités pour afficher la liste des parties en attente de joueurs et pour créer une nouvelle partie. *Match* contient toutes les fonctionnalités pour jouer une partie. *WaitingRoom* contient

les fonctionnalités dans les salles d'attente pour des parties.

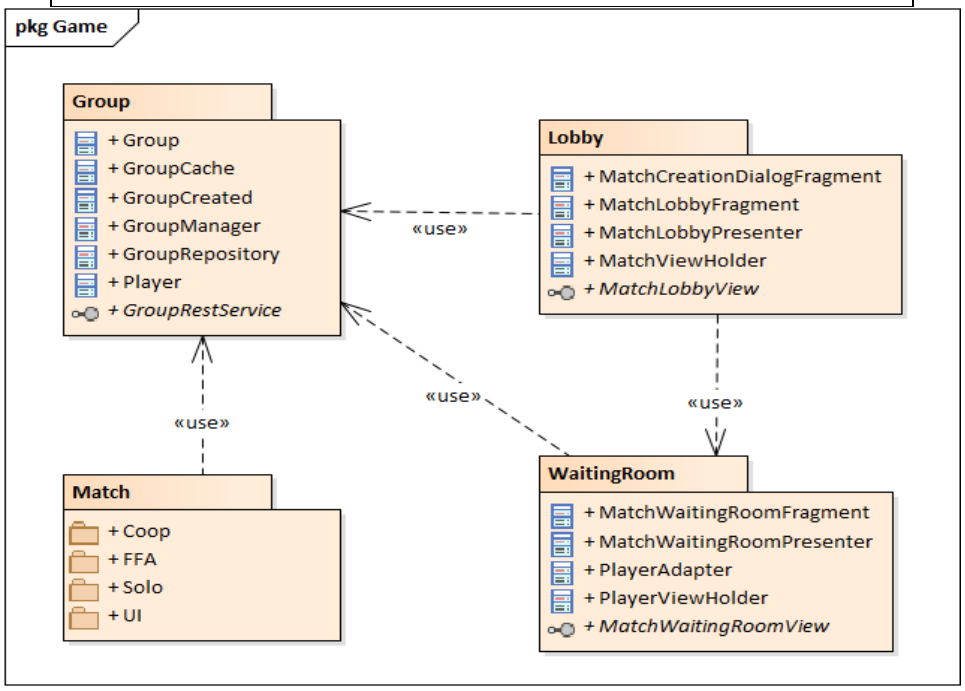

Figure 4.2.3: Diagramme de paquetage ClientLéger::Game

4.2.3.1 Group

ClientLéger::Game::Group
Ce paquetage regroupe les fonctionnalités utilisées pour les groupes de personnes en attente de jouer une partie. On y retrouve les fonctionnalités pour communiquer au serveur, pour gérer les nombreux groupes et pour intercepter les événements liés aux groupes.

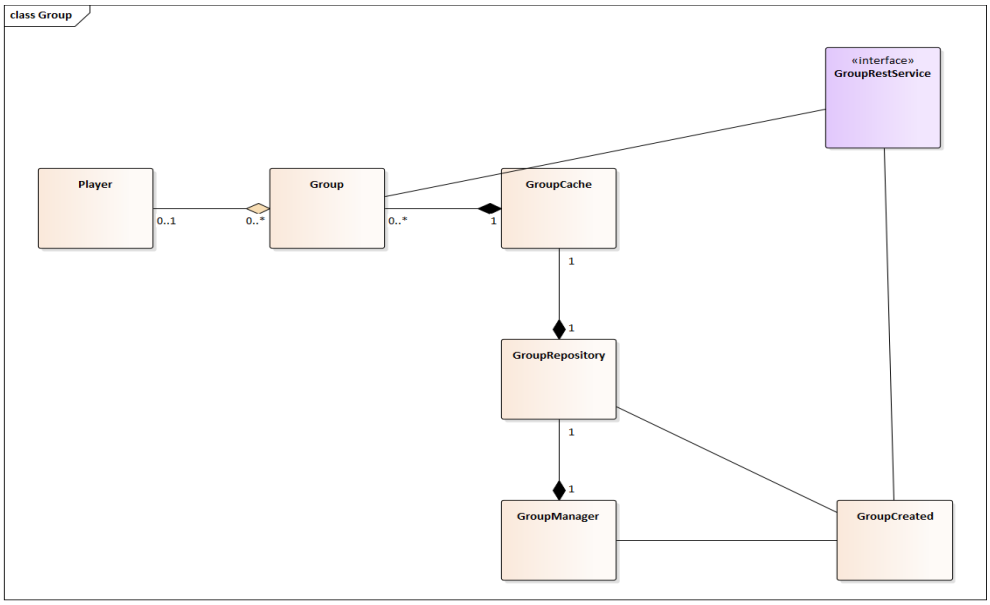

Figure 4.2.3.1: Diagramme de paquetage ClientLéger::Game::Group

4.2.3.2 Lobby

ClientLéger::Game::Lobby
Ce paquetage regroupe toutes les fonctionnalités pour le lobby principal. On peut y retrouver ce qui concerne l'affichage d'une partie en attente, la création de parties et ce qui touche à rejoindre une partie.

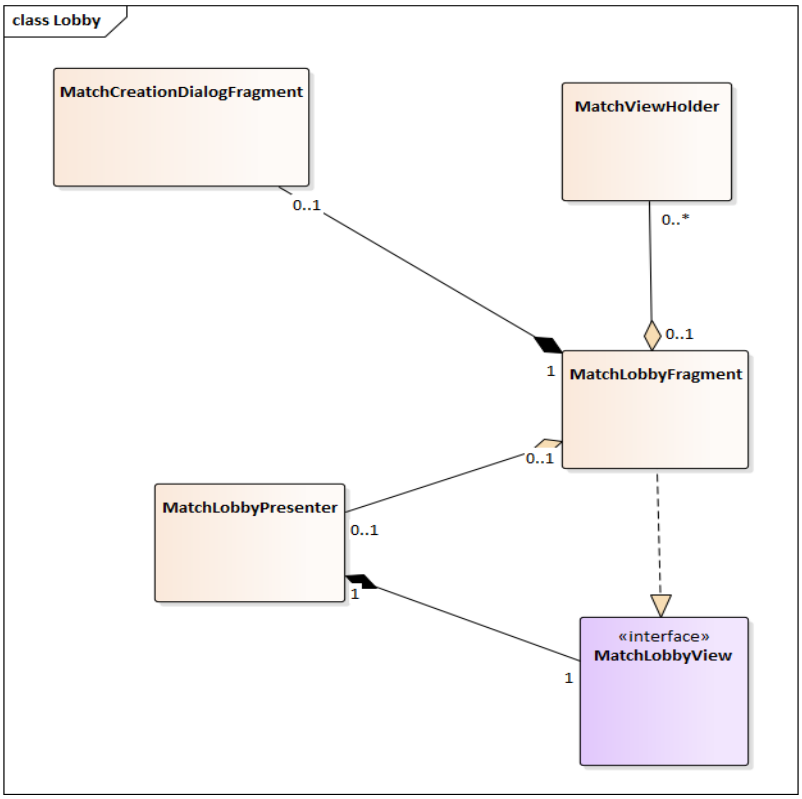

Figure 4.2.3.2: Diagramme de paquetage ClientLéger::Game::Lobby

4.2.3.3 Match

ClientLéger::Game::Match
Ce paquetage regroupe toutes les fonctionnalités pour les parties de jeu. On retrouve les gestionnaires de parties ainsi que tout ce qui touche l'affichage des parties.

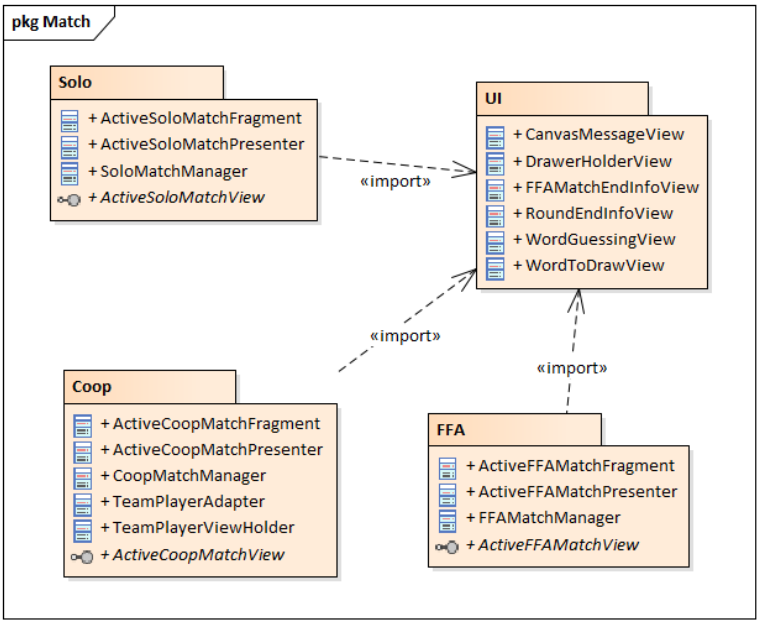

Figure 4.2.3.3: Diagramme de paquetage ClientLéger::Game::Match

4.2.3.3.1 Coop

ClientLéger::Game::Match::Coop
Ce paquetage regroupe les fonctionnalités pour jouer une partie Coop. On retrouve la logique d'affichage de parties Coop ainsi que le gestionnaire de parties Coop.

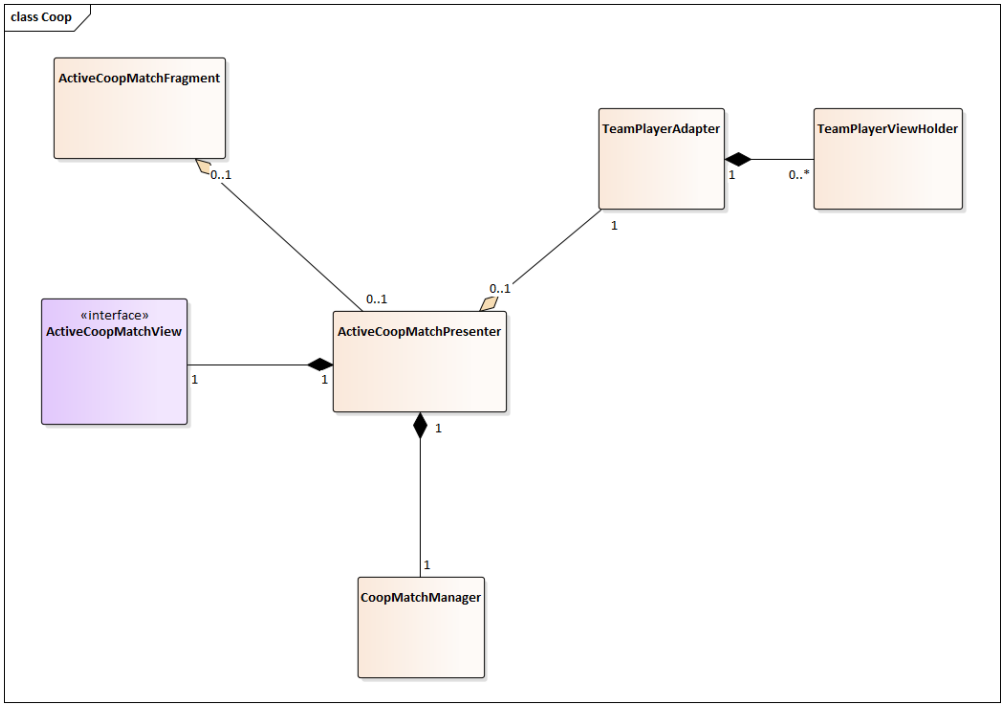

Figure 4.2.3.3.1: Diagramme de paquetage ClientLéger::Game::Match::Coop

4.2.3.3.2 FFA

ClientLéger::Game::Match::FFA
Ce paquetage regroupe les fonctionnalités pour jouer une partie mêlée générale. On retrouve la logique d'affichage de parties mêlée générale ainsi que le gestionnaire de parties mêlée générale.

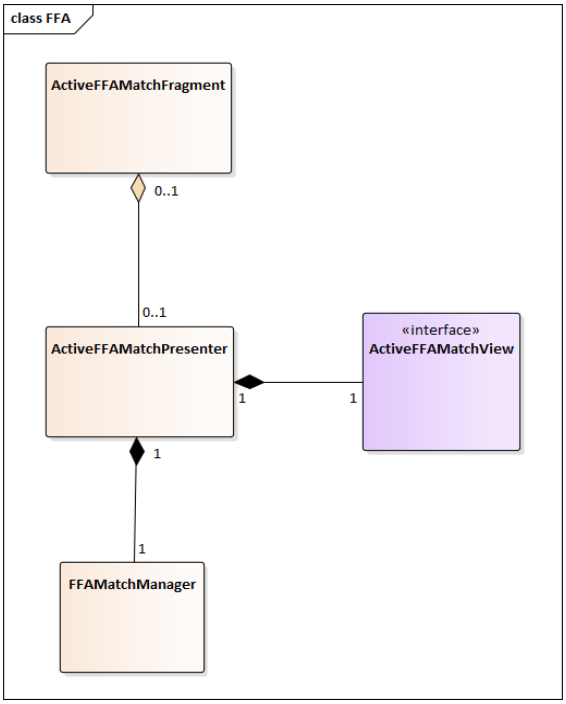

Figure 4.2.3.3.2: Diagramme de paquetage ClientLéger::Game::Match::FFA

4.2.3.3.3 Solo

ClientLéger::Game::Match::Solo
Ce paquetage regroupe les fonctionnalités pour jouer une partie solo. On retrouve la logique d'affichage de parties solo ainsi que le gestionnaire de parties solo.

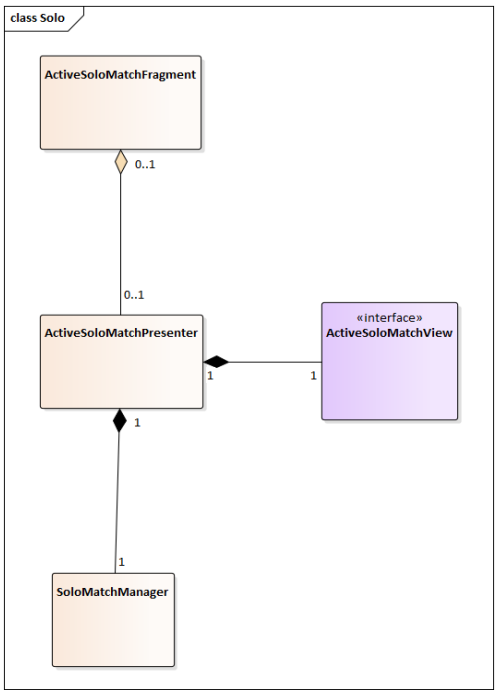

Figure 4.2.3.3.3: Diagramme de paquetage ClientLéger::Game::Match::Solo

4.2.3.3.4 UI

ClientLéger::Game::Match::UI
Ce paquetage regroupe les fonctionnalités pour l'affichage de divers éléments d'interface utilisateur pour les différentes parties. On peut y retrouver les messages sur le canevas, les messages de fin de ronde, etc.

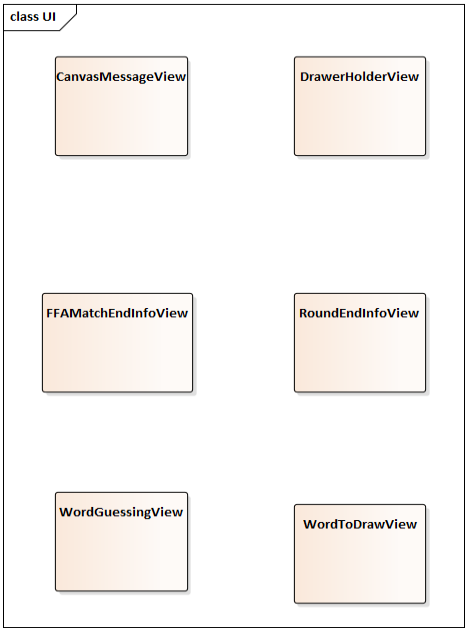

Figure 4.2.3.3.4: Diagramme de paquetage ClientLéger::Game::Match::UI

4.2.3.4 WaitingRoom

ClientLéger::Game::WaitingRoom
Ce paquetage regroupe les fonctionnalités pour les salles d'attente de parties. On y retrouve la logique pour renvoyer des joueurs, ajouter des joueurs virtuels, commencer la partie, etc.

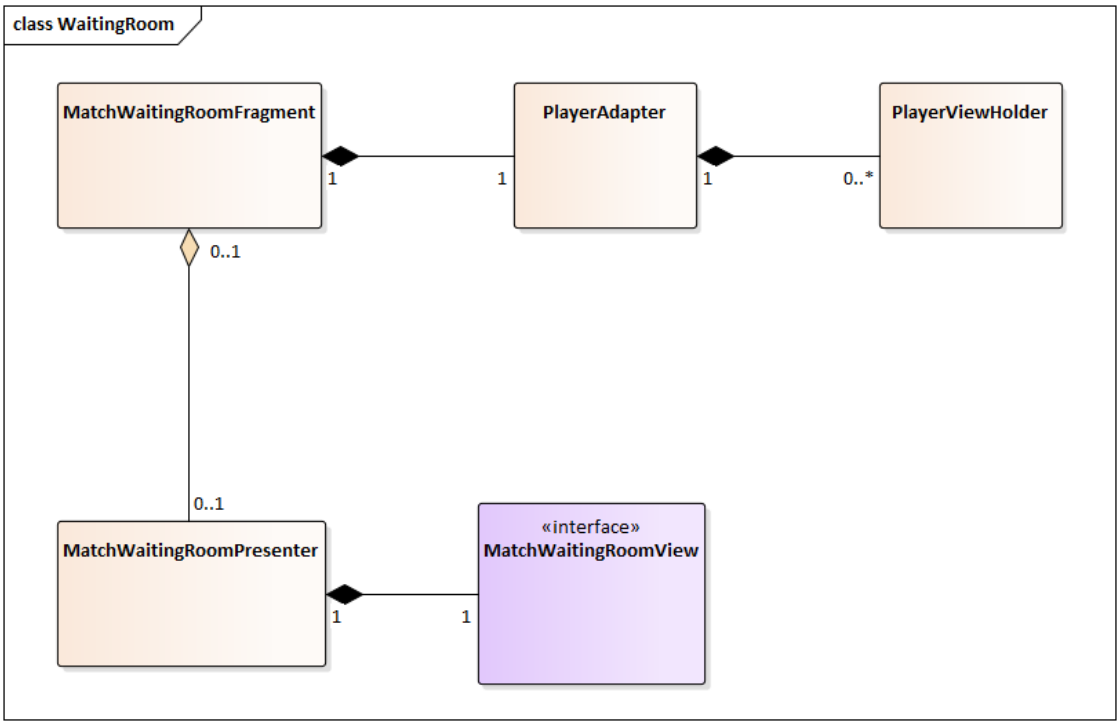

Figure 4.2.3.4: Diagramme de paquetage ClientLéger::Game::WaitingRoom

4.2.4 Login

ClientLéger::Login
Ce paquetage regroupe les fonctionnalités pour se connecter à l'application ou créer un nouveau compte.

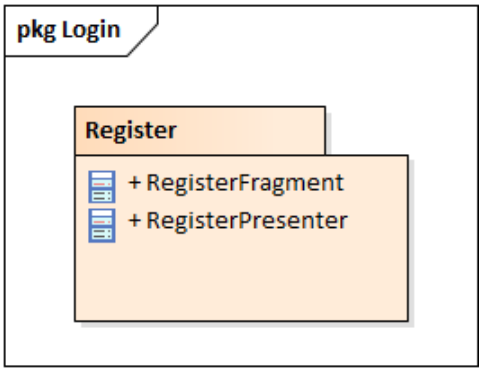

Figure 4.2.4: Diagramme de paquetage ClientLéger::Login

4.2.4.1 Register

ClientLéger::Login::Register
Ce paquetage regroupe les fonctionnalités pour se créer un nouveau compte. On y retrouve la logique pour la communication réseau, pour entrer ses informations pour un nouveau compte, etc.

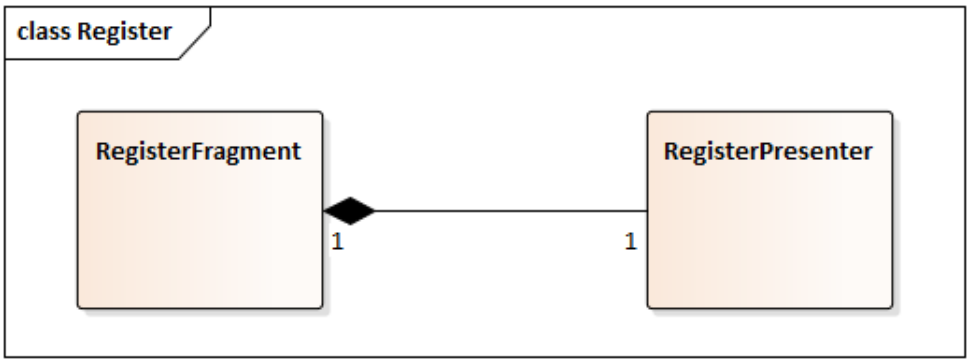

Figure 4.2.4.1: Diagramme de paquetage ClientLéger::Login::Register

4.2.5 Profile

ClientLéger::Profile
Ce paquetage regroupe les fonctionnalités pour l'affichage des comptes. On y retrouve la logique pour modifier son compte, visionner son historique de connexion, visionner ses statistiques, etc.

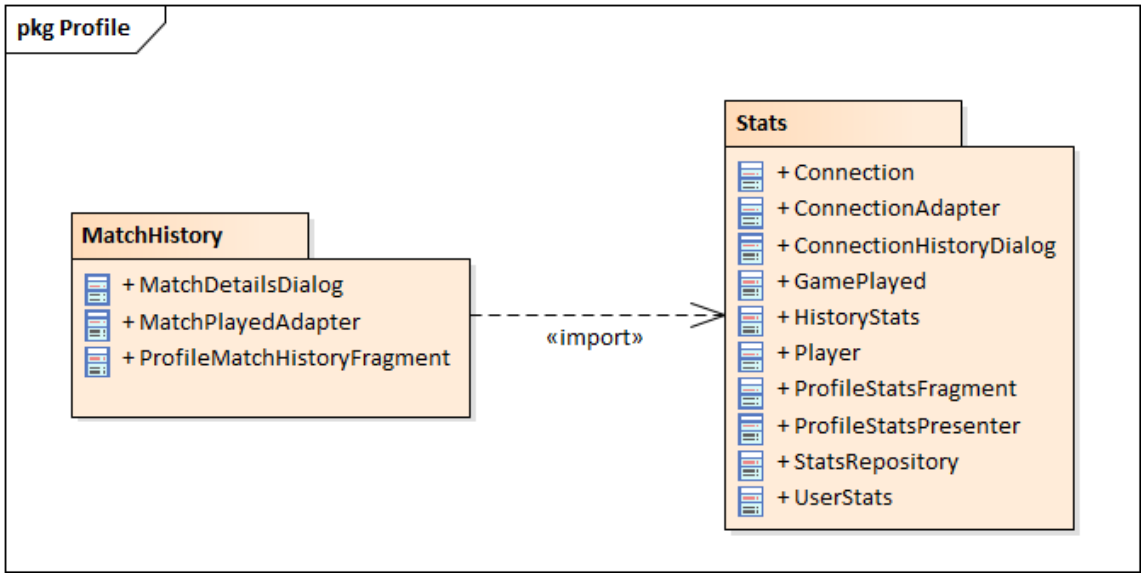

Figure 4.2.5: Diagramme de paquetage ClientLéger::Profile

4.2.5.1 MatchHistory

ClientLéger::Profile::MatchHistory
Ce paquetage regroupe les fonctionnalités pour visionner son historique de parties.

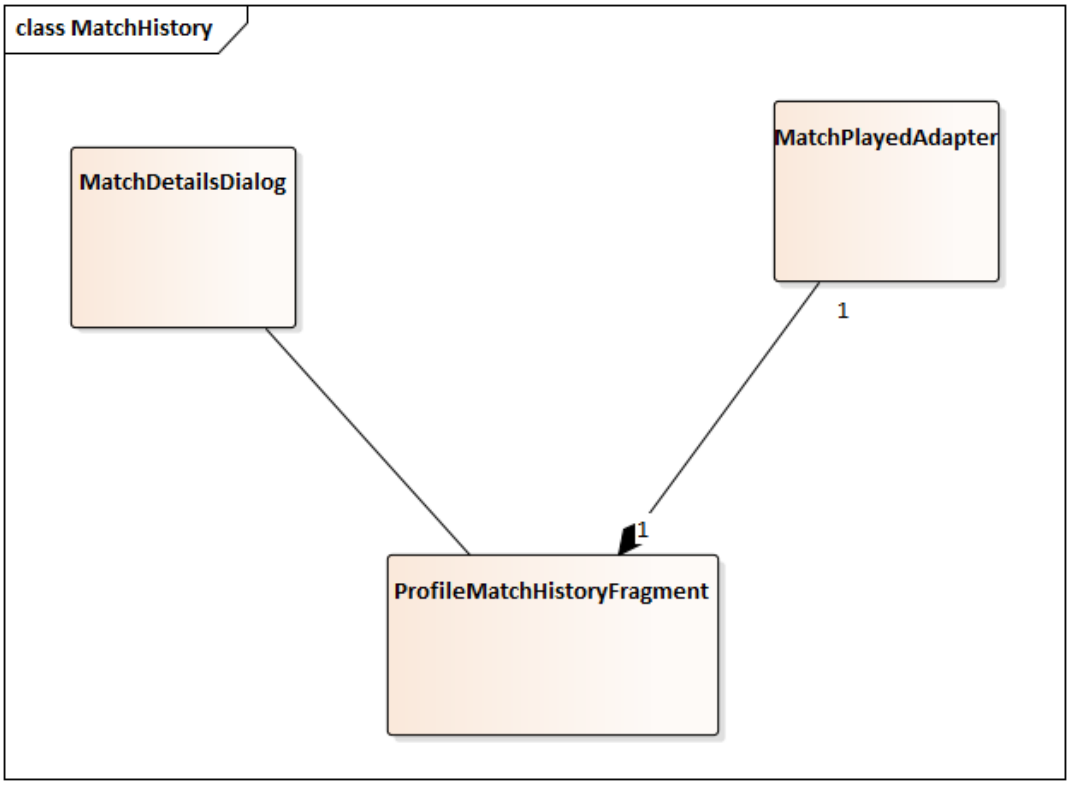

Figure 4.2.5.1: Diagramme de paquetage ClientLéger::Profile::MatchHistory

4.2.5.2 Stats

ClientLéger::Profile::Stats
Ce paquetage regroupe les fonctionnalités pour visionner les statistiques de connexion et de parties d'un utilisateur.

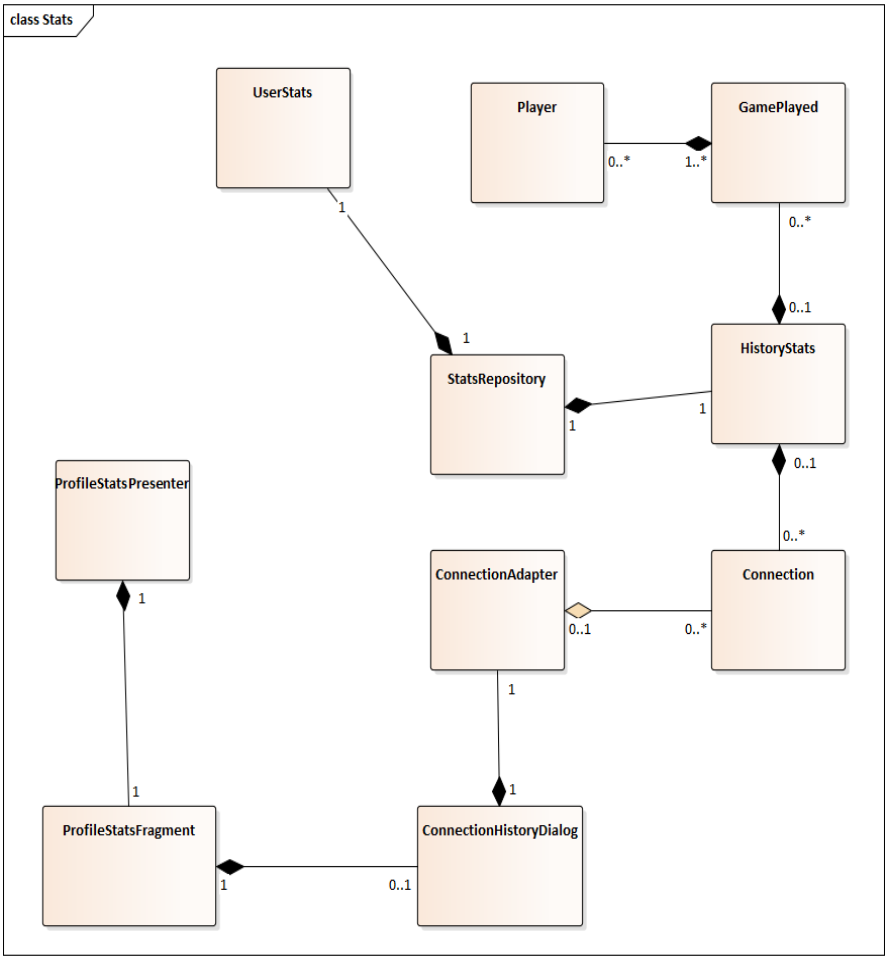

Figure 4.2.5.2: Diagramme de paquetage ClientLéger::Profile::Stats

4.2.6 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.

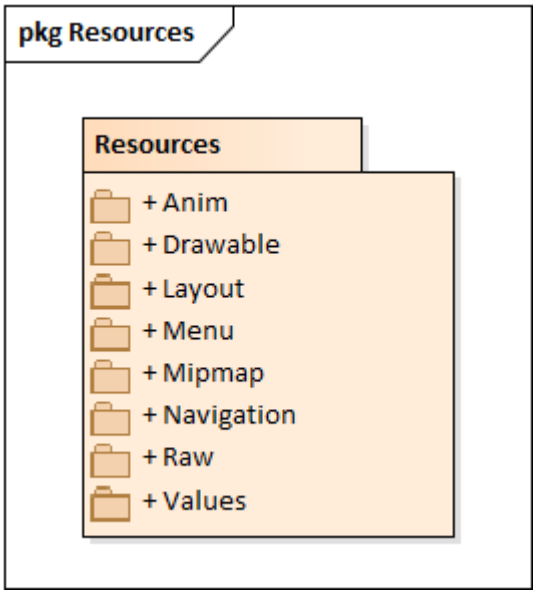

Figure 4.2.6: Diagramme de paquetage ClientLéger::Resources

Chaque paquetage du paquetage *Resources e*st décrit ci-dessous. Les diagrammes ont été omis puisqu'une ressource n'est pas une classe et qu'il n'y a aucun lien logique entre les différentes ressources.

4.2.6.1 Anim

ClientLéger::Ressources::Anim
Ce paquetage regroupe plusieurs animations utilisées dans l'application.

4.2.6.2 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.

4.2.6.3 Layout

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.

4.2.6.4 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.

4.2.6.5 Mipmap

ClientLéger::Ressources::Mipmap
Ce paquetage regroupe plusieurs images de l'application telles que le logo, des avatars, des icônes, etc.

4.2.6.6 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.

4.2.6.7 Raw

ClientLéger::Ressources::Raw
Ce paquetage regroupe tous les effets sonores utilisés par l'application.

4.2.6.8 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.

4.2.7 Socket

ClientLéger::Socket
Ce paquetage regroupe les fonctionnalités pour assurer la communication synchrone par *Socket*.

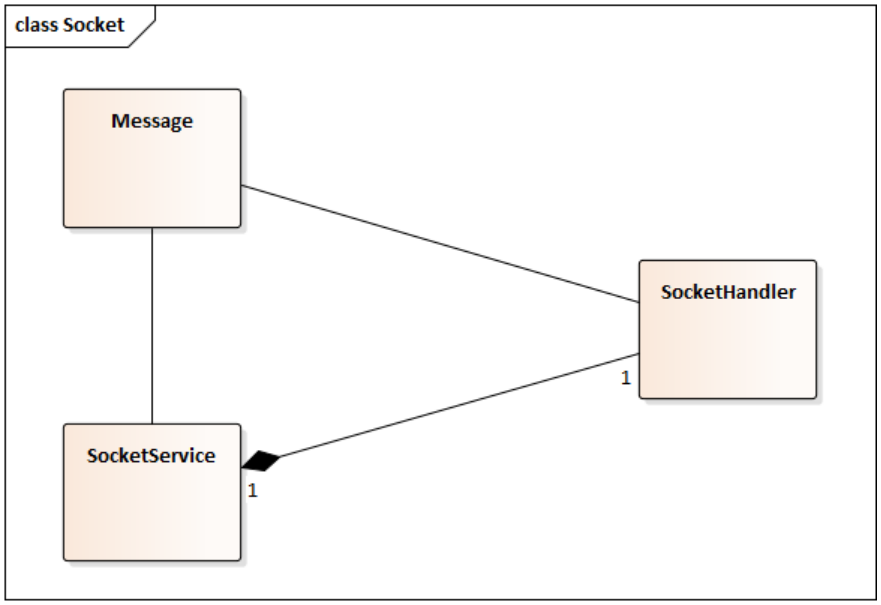

Figure 4.2.7: Diagramme de paquetage ClientLéger::Socket

4.2.8 Tutorial

ClientLéger::Tutorial
Ce paquetage regroupe les fonctionnalités pour gérer le tutoriel. On y retrouve un gestionnaire de tutoriel ainsi que chaque diapositive du tutoriel.

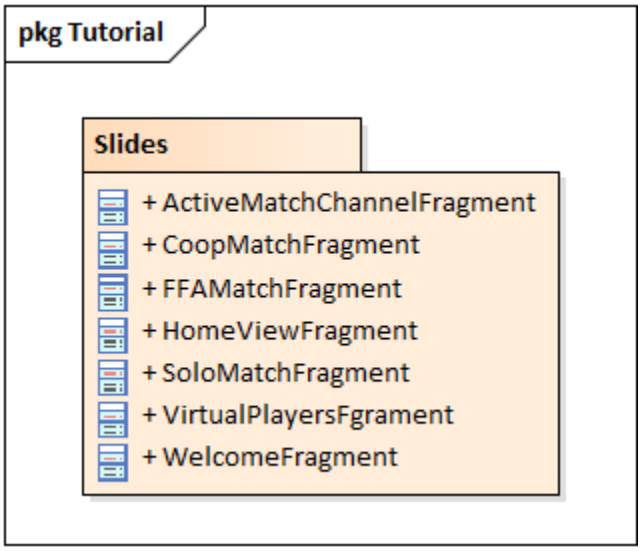

Figure 4.2.8: Diagramme de paquetage ClientLéger:Tutorial

4.2.8.1 Slides

ClientLéger::Tutorial::Slides
Ce paquetage regroupe toutes les diapositives utilisées par le tutoriel.

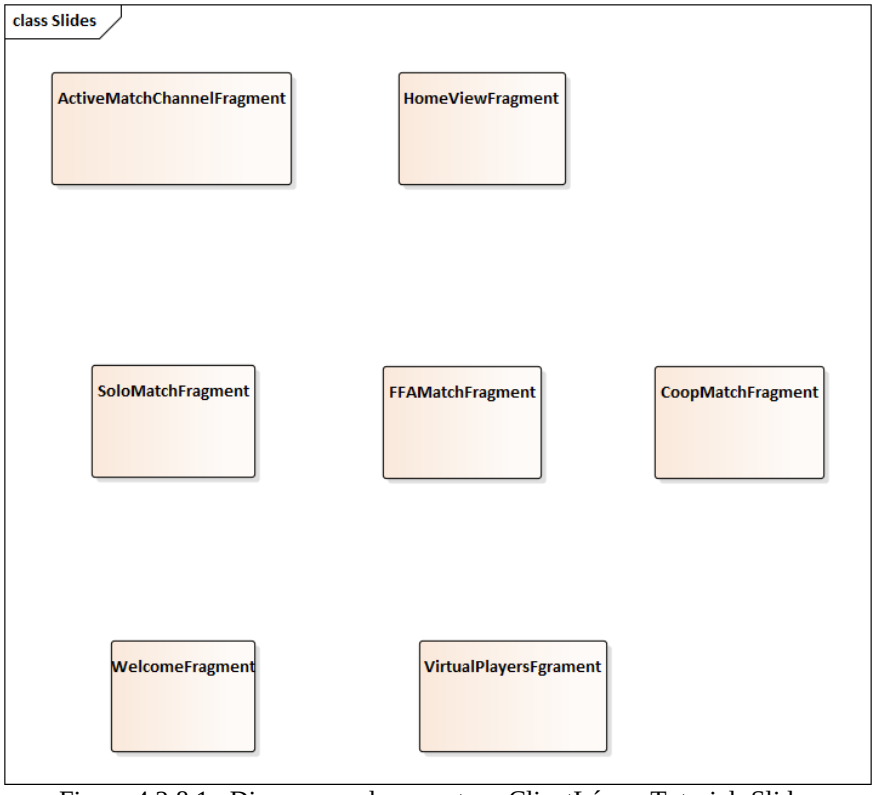

Figure 4.2.8.1: Diagramme de paquetage ClientLéger::Tutorial::Slides

4.2.9 User

ClientLéger::User
Ce paquetage regroupe les différents types d'utilisateurs ainsi que certaines fonctionnalités liées à la gestion d'utilisateurs.

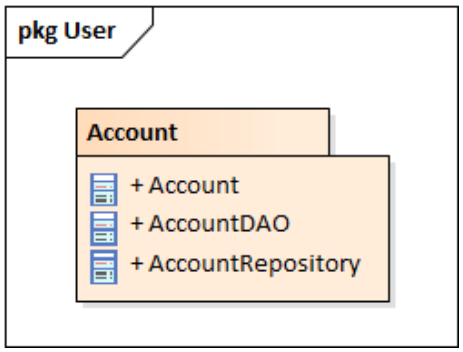

Figure 4.2.9: Diagramme de paquetage ClientLéger::User

4.2.9.1 Account

ClientLéger::User::Account
Ce paquetage regroupe les fonctionnalités pour gérer l'utilisateur courant, tel que la base de données de préférences ainsi que le gestionnaire de comptes.

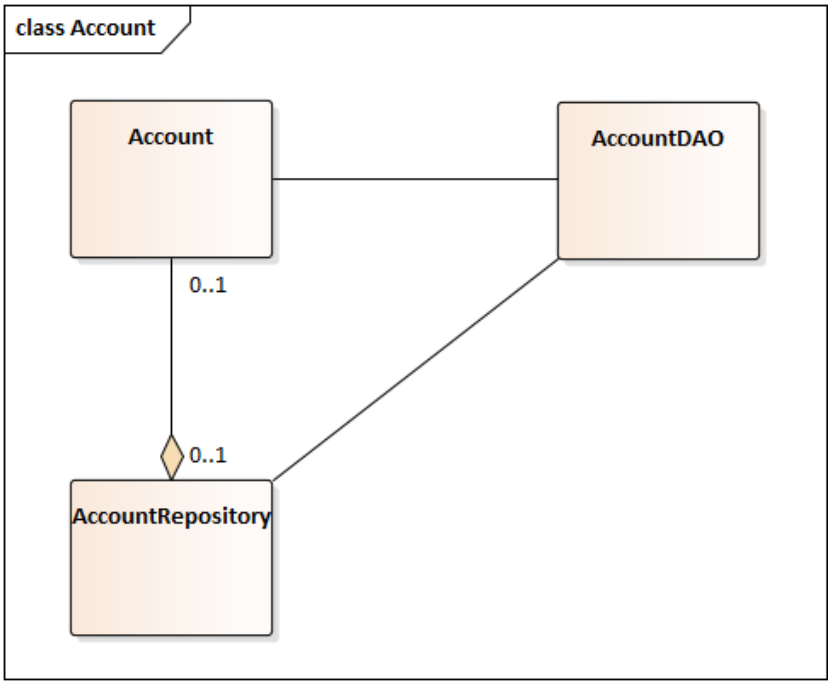

Figure 4.2.9.1: Diagramme de paquetage ClientLéger::User::Account

4.2.10 Utils

ClientLéger::Utils
Ce paquetage regroupe divers outils utilisés dans plusieurs paquetages. On y retrouve des fonctionnalités pour convertir des données, formater des données et des outils pour traiter des événements de l'interface utilisateur.

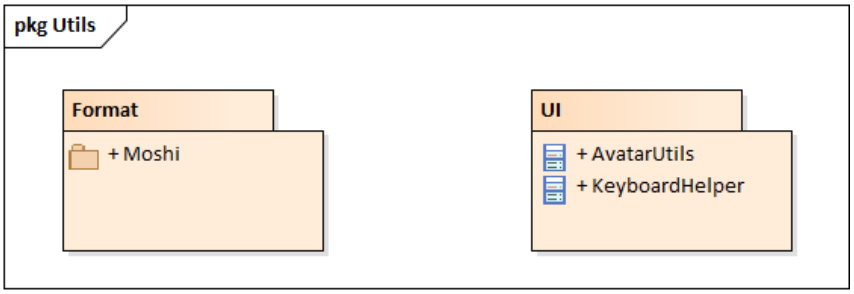

Figure 4.2.10: Diagramme de paquetage ClientLéger::Utils

4.2.10.1 Format

ClientLéger::Utils::Format
Ce paquetage regroupe divers outils pour formater des données provenant du serveur tel que des dates ou des objets à convertir.

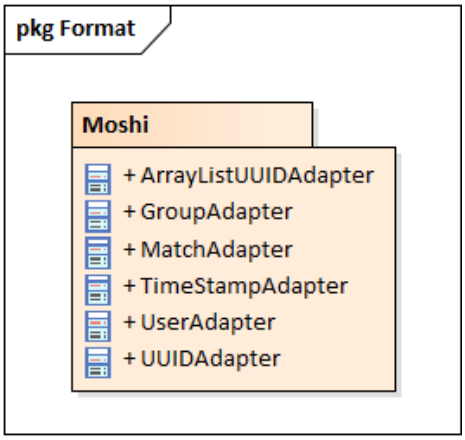

Figure 4.2.10.1: Diagramme de paquetage ClientLéger::Utils::Format

4.2.10.1.1 Moshi

ClientLéger::Utils::Format::Moshi
Ce paquetage regroupe des outils pour convertir des objets envoyés par le serveur.

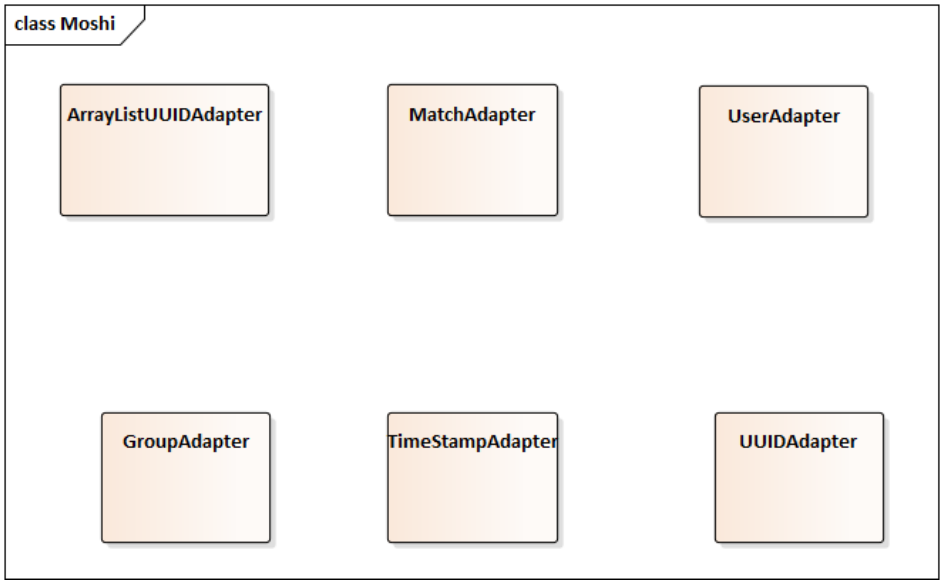

Figure 4.2.10.1.1: Diagramme de paquetage ClientLéger::Utils::Format::Moshi

4.2.10.2 UI

ClientLéger::Utils::UI
Ce paquetage regroupe divers outils pour gérer des événements d'interface utilisateur tels que des événements du clavier ou sur des images.

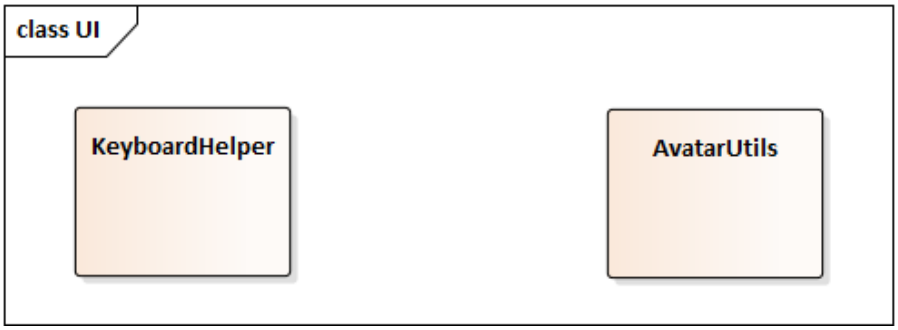

Figure 4.2.10.2: Diagramme de paquetage ClientLéger::Utils::UI

4.3 Serveur

Le serveur possède une architecture qui est orientée 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.

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.

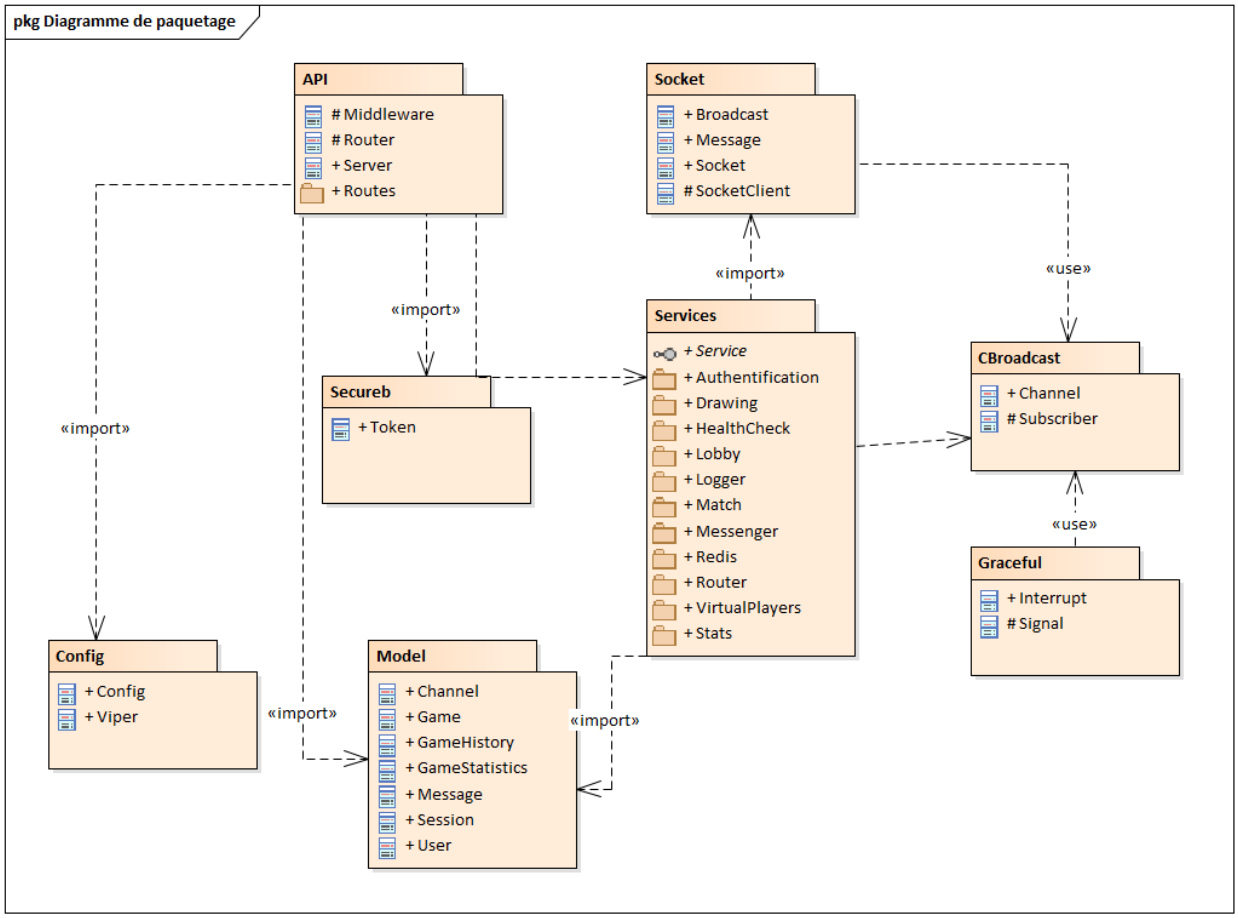

Figure 4.3: Diagramme de paquetage serveur

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.

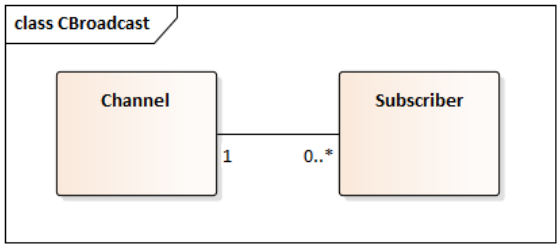

Figure 4.3.1: Diagramme de classe Serveur::CBroadcast

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.

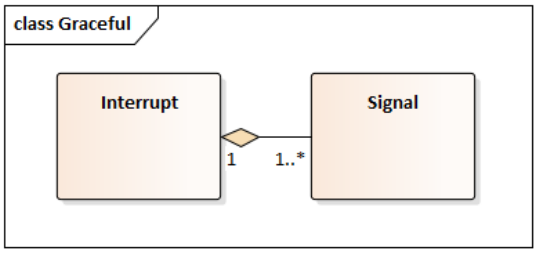

Figure 4.3.2: Diagramme de classe Serveur::Graceful

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.

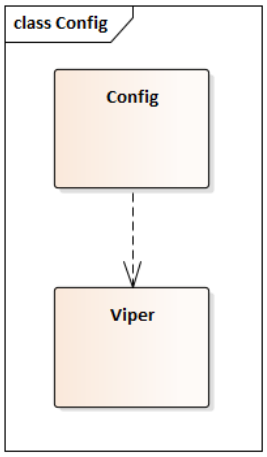

Figure 4.3.4: Diagramme de classe Serveur::Config

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.

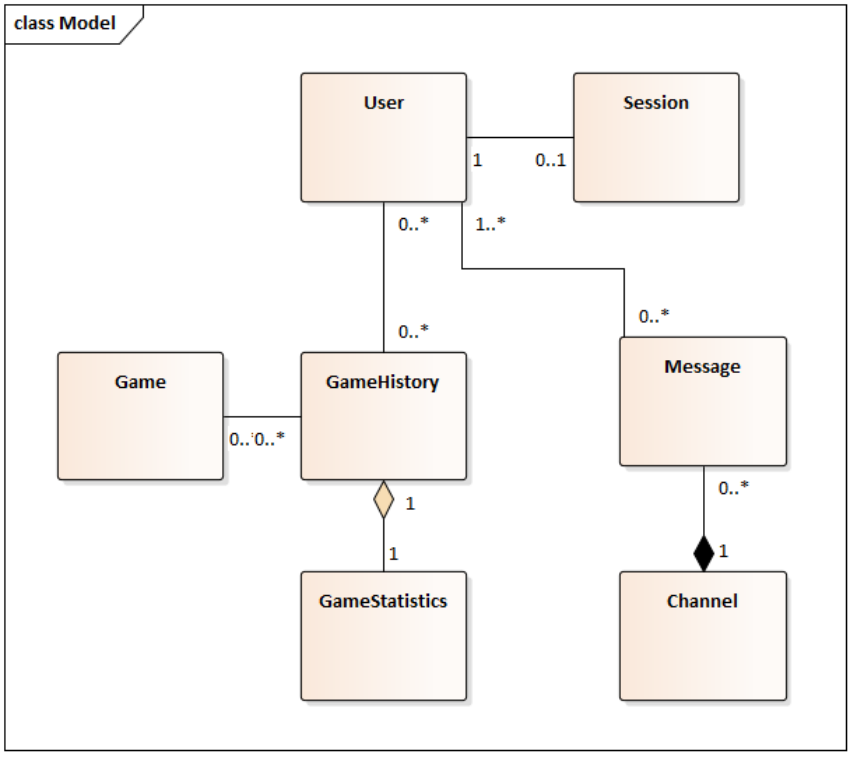

Figure 4.3.5: Diagramme de classe Serveur::Model

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.

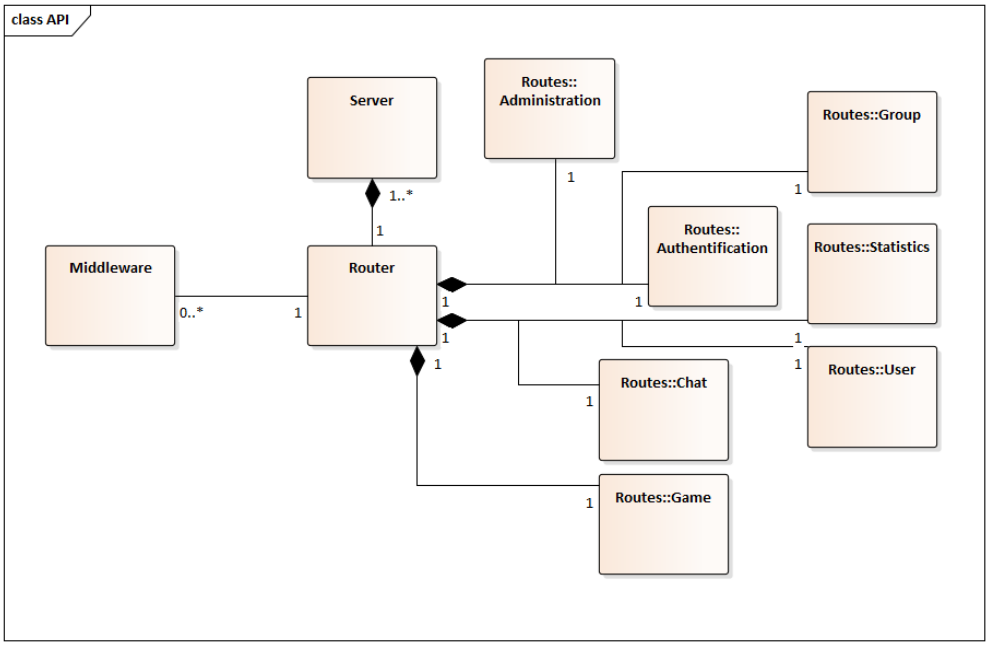

Figure 4.3.6.1: Diagramme de classe Serveur::API

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.

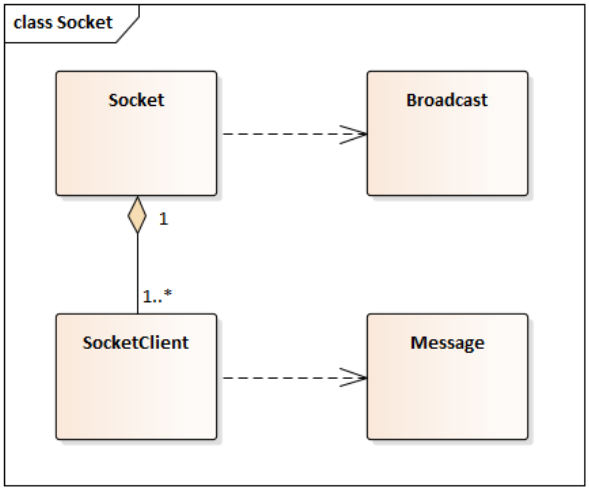

Figure 4.3.6.2: Diagramme de classe Serveur::Socket

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.

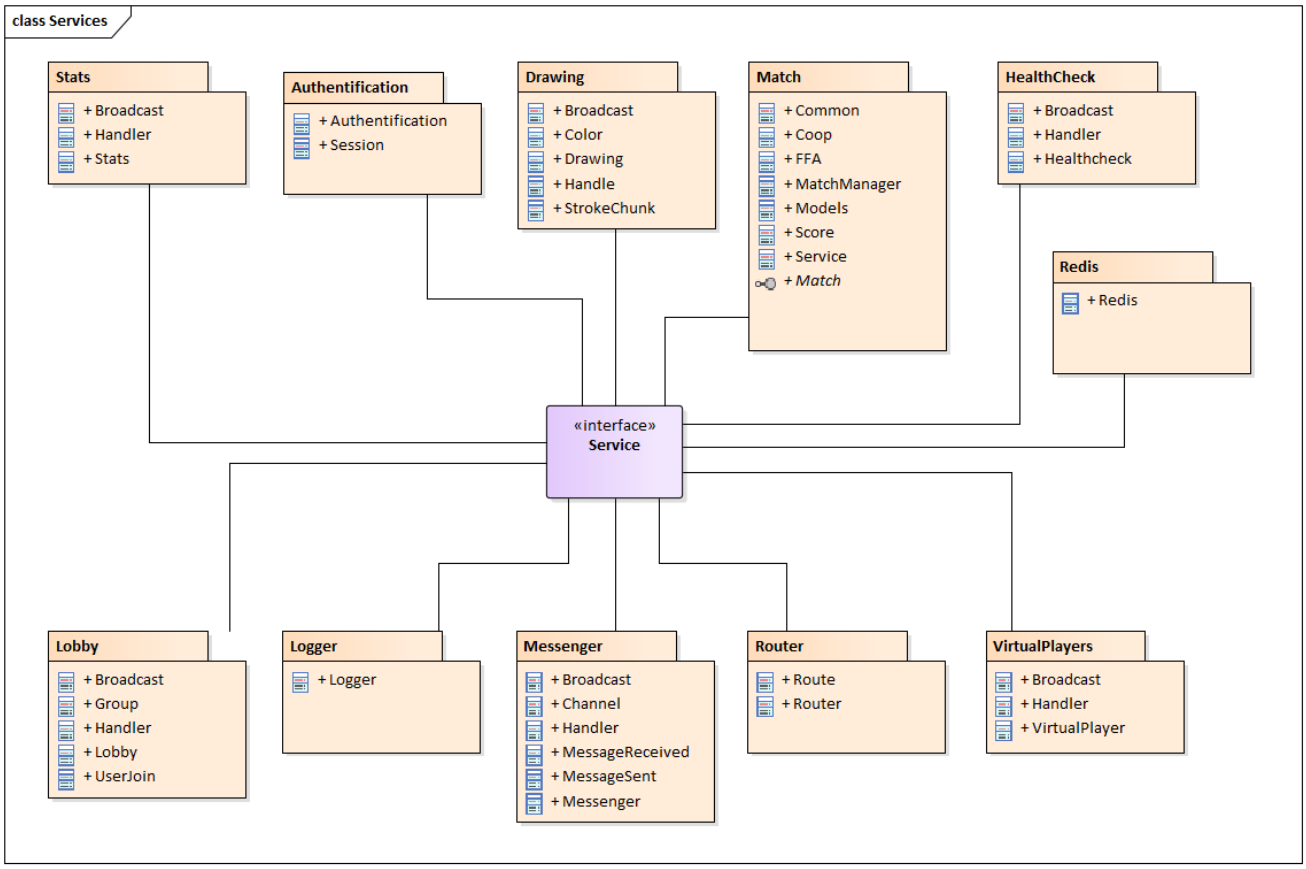

Figure 4.3.7.1: Diagramme de classe Serveur::Services

Serveur::Services::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.

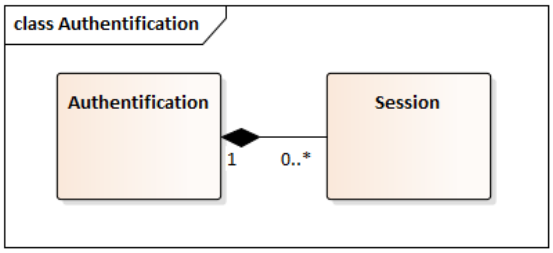

Figure 4.3.7.2: Diagramme de classe Serveur::Services::Authentification

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.

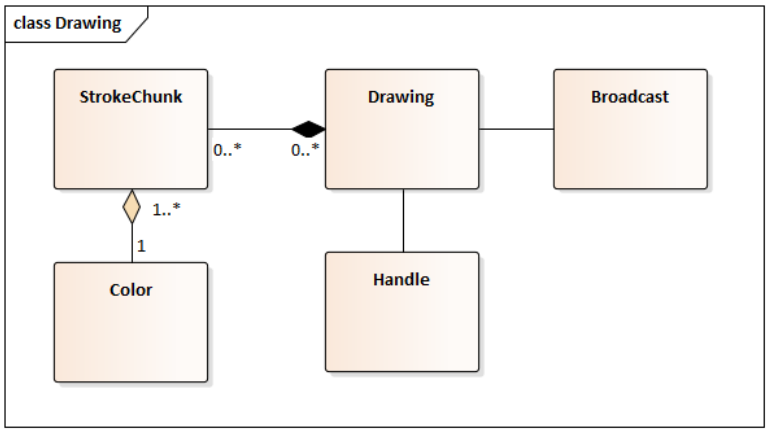

Figure 4.3.7.3: Diagramme de classe Serveur::Services::Drawing

Serveur::Services::Match
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.

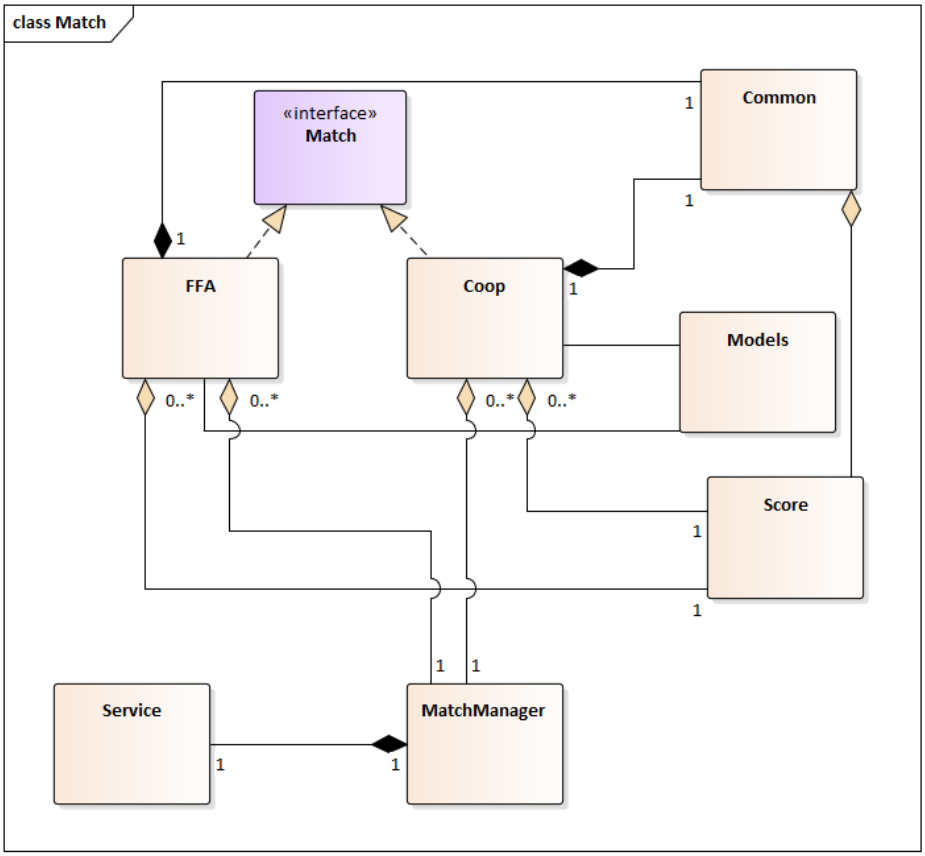

Figure 4.3.7.4: Diagramme de classe Serveur::Services::Match

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.

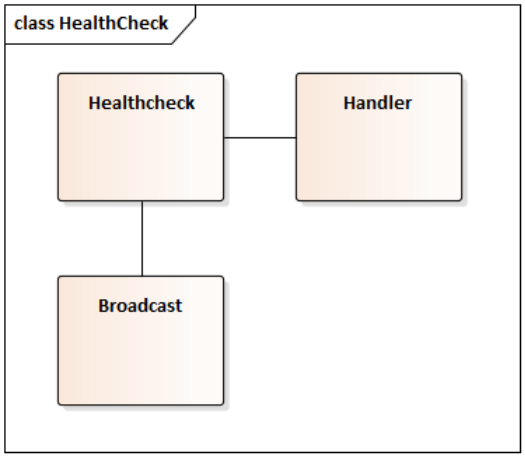

Figure 4.3.7.5: Diagramme de classe Serveur::Services::Healthcheck

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::Redis
Ce paquetage est utilisé pour maintenir la connexion avec Redis. Dans le cas où celui-ci serait déconnecté, l'application sera automatiquement fermée. Ceci à pour but d'éviter des états non déterminés.

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.

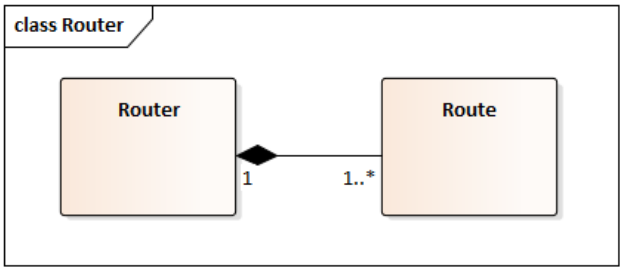

Figure 4.3.7.6: Diagramme de classe Serveur::Services::Router

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

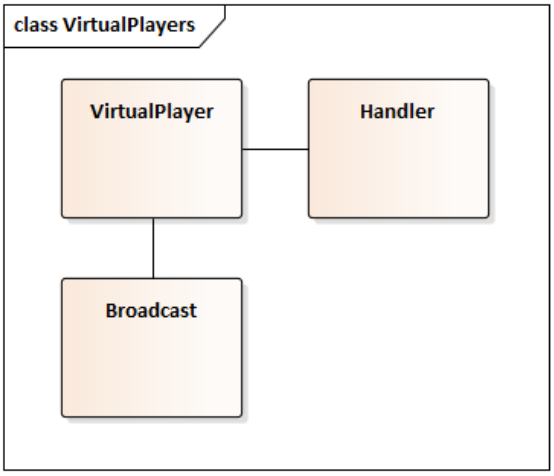

Figure 4.3.7.7: Diagramme de classe Serveur::Services::VirtualPlayers

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

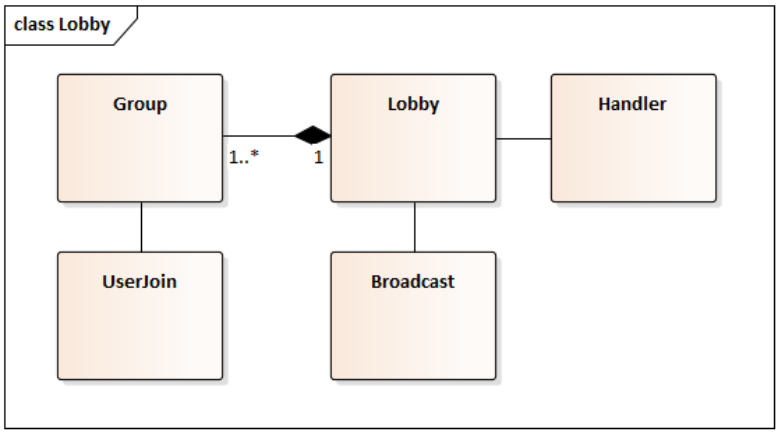

Figure 4.3.7.8: Diagramme de classe Serveur::Services::Lobby

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.

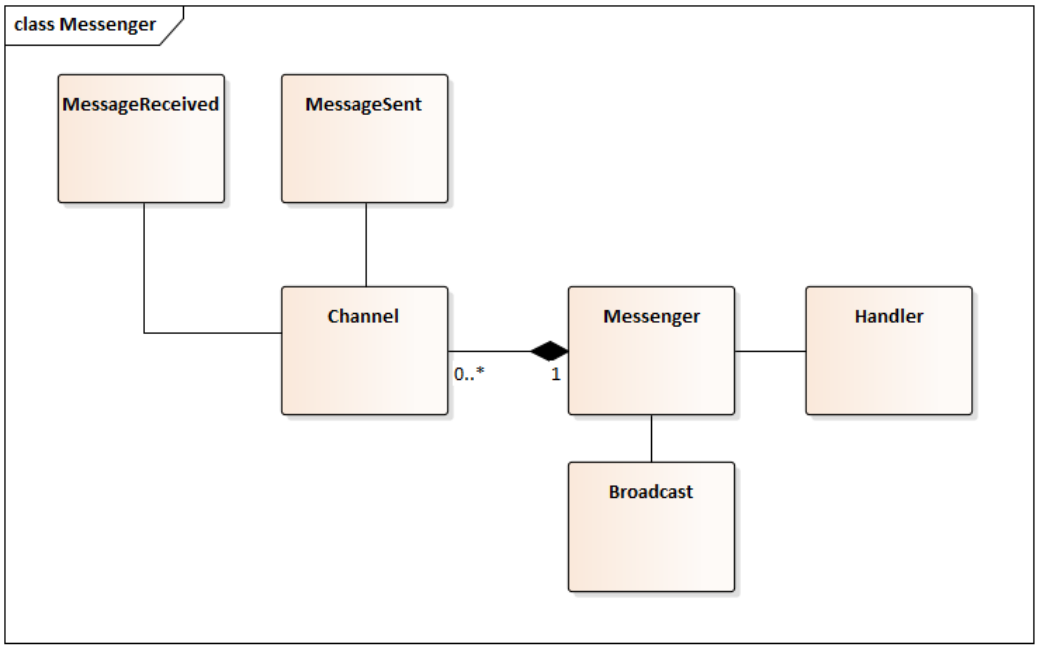

Figure 4.3.7.9: Diagramme de classe Serveur::Services::Messenger

Serveur::Services::Stats
Ce paquetage est utilisé pour gérer les statistiques. Il reçoit tous les événements de parties, et de connexions, déconnexions et les enregistrer dans la base de données.

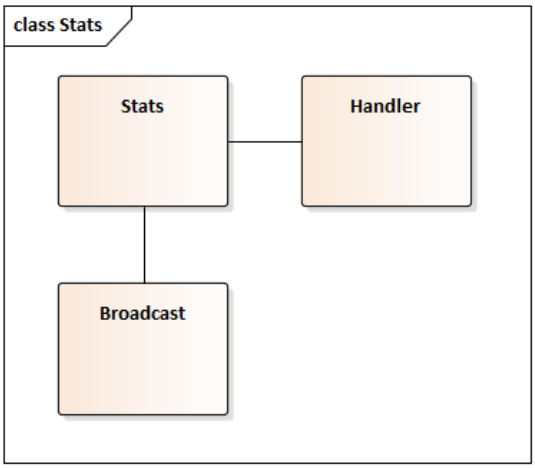

Figure 4.3.7.10: Diagramme de classe Serveur::Services::Stats

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

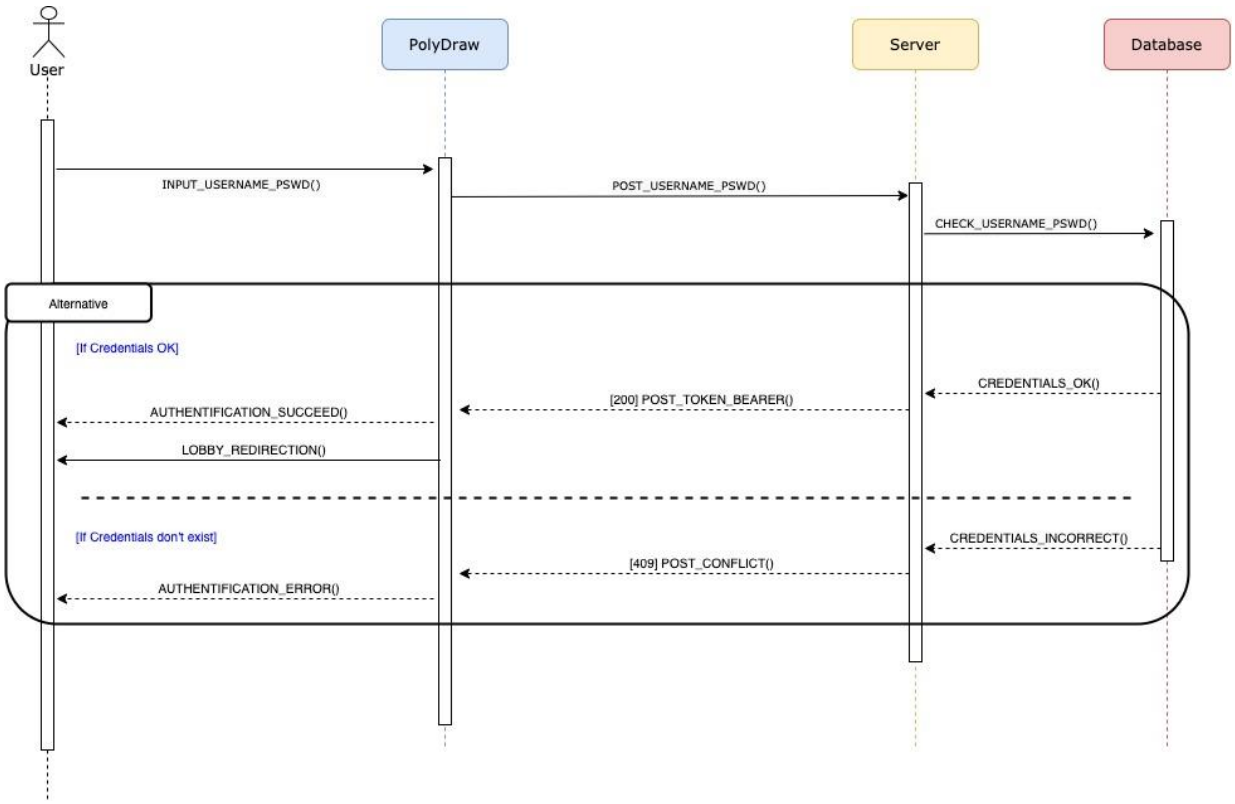

Figure 5.1: Présentation de la séquence «se connecter »

5.2 Créer un profil d'utilisateur

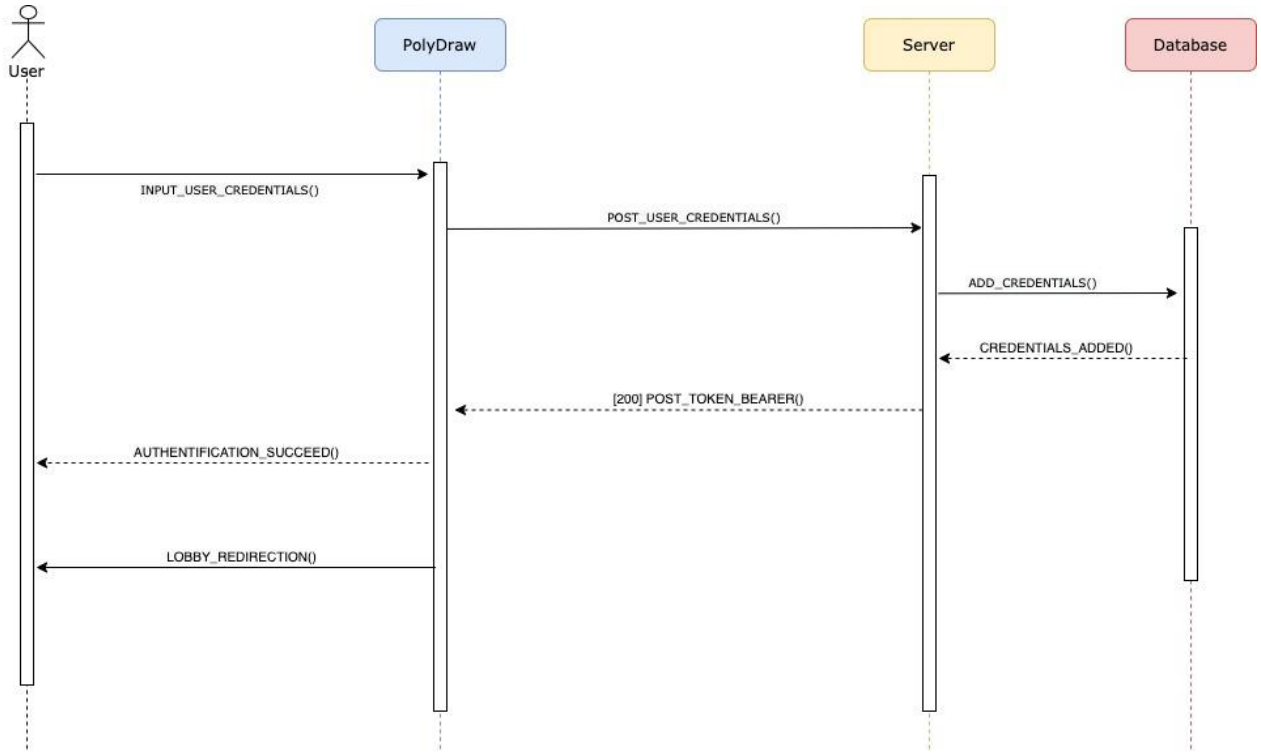

Figure 5.2: Présentation de la séquence « créer un profil d'utilisateur »

5.3 Clavarder

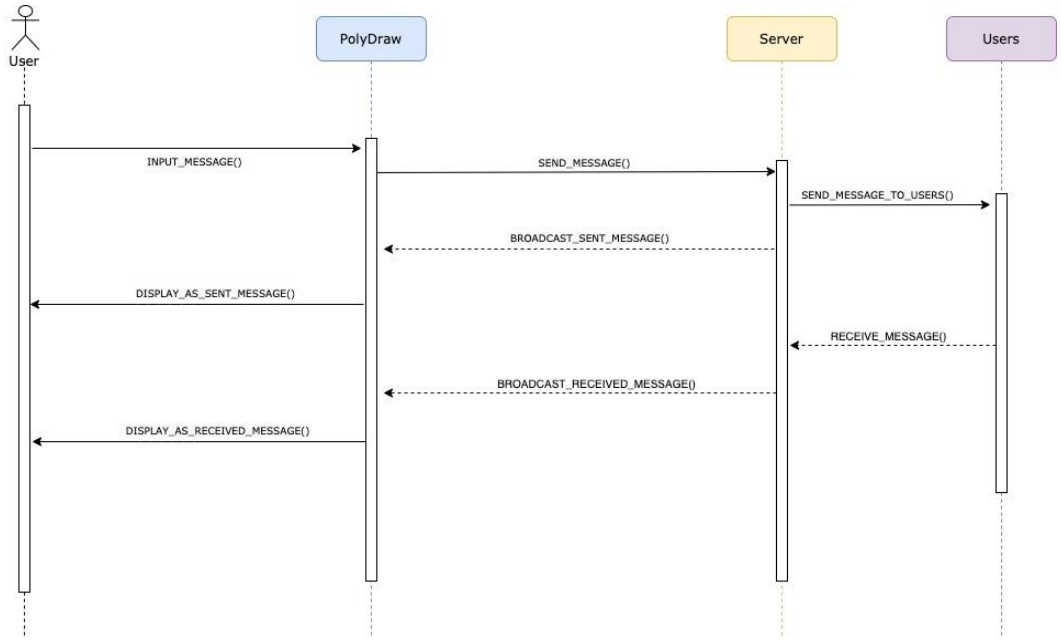

Figure 5.3: Présentation de la séquence « clavarder »

5.4 Gérer un chat

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

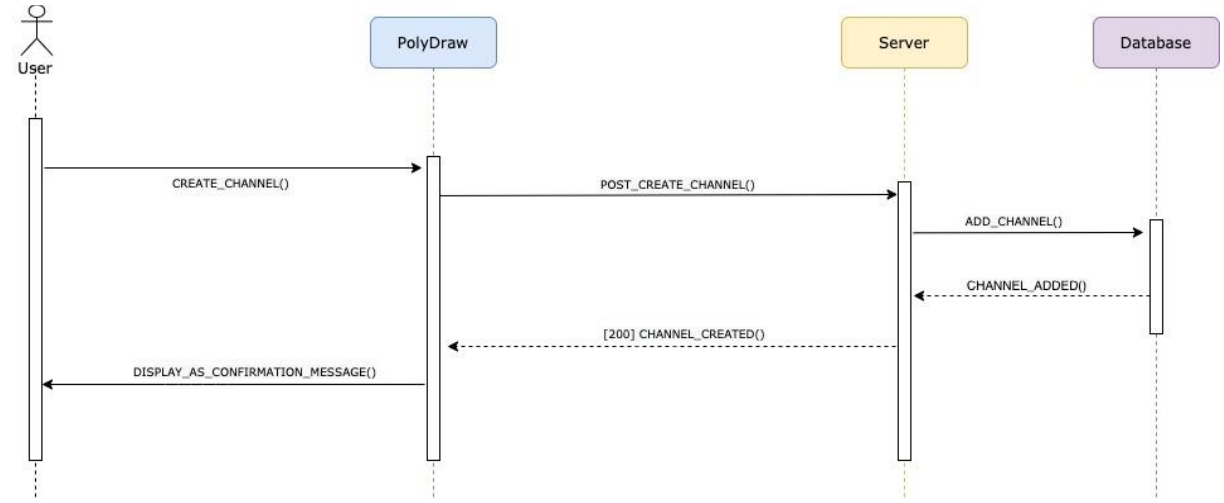

Figure 5.4.1: Présentation de la séquence *créer un canal* issue de « gérer un chat »

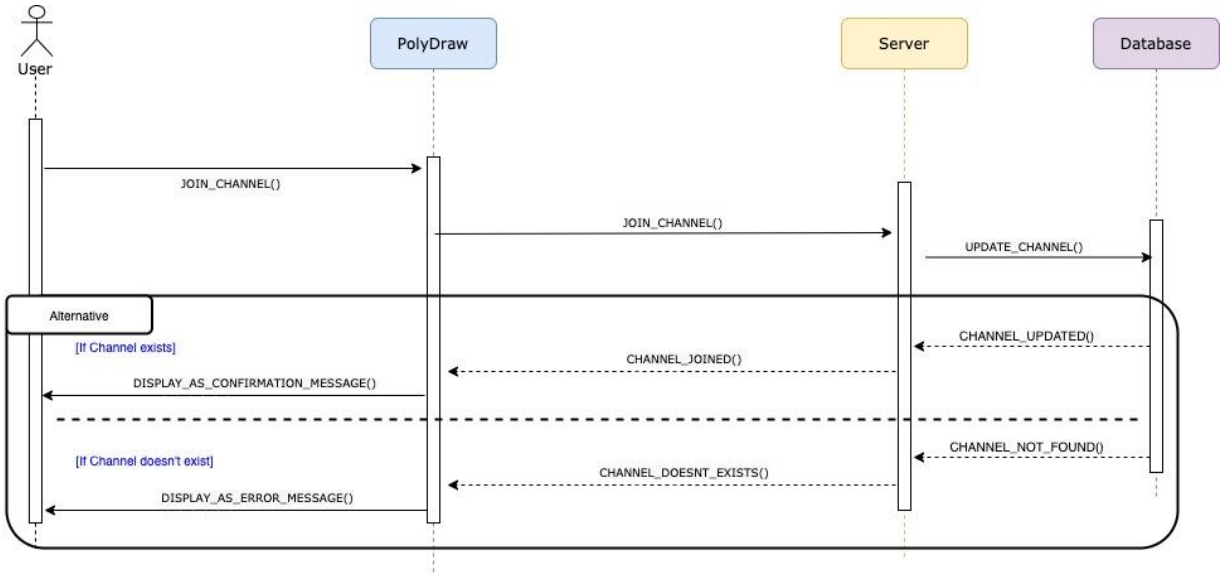

Figure 5.4.2: Présentation de la séquence *rejoindre un canal* issue de « gérer un *chat* »

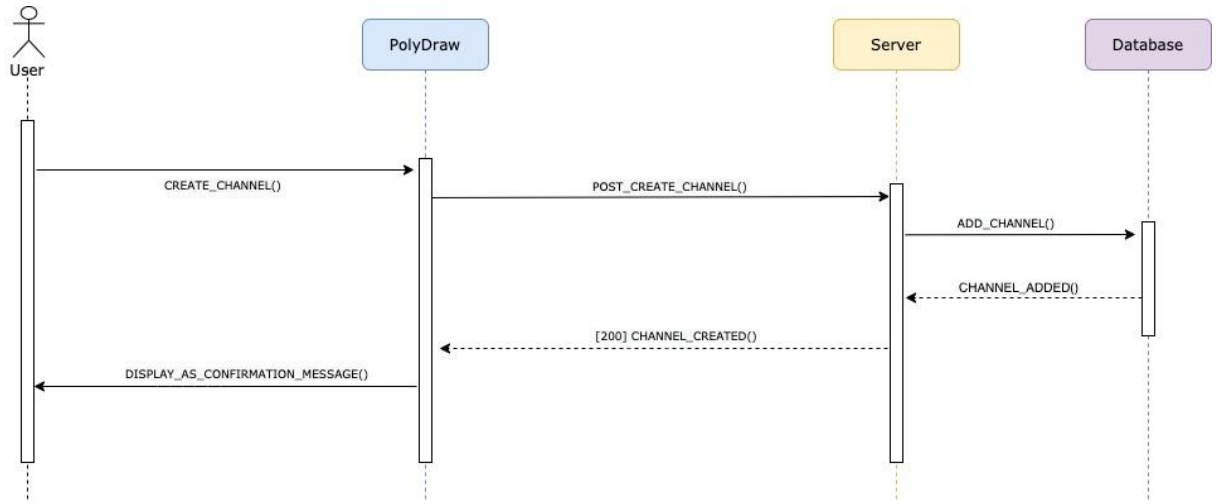

Figure 5.4.3: Présentation de la séquence ***supprimer un canal*** issue de « gérer un *chat* »

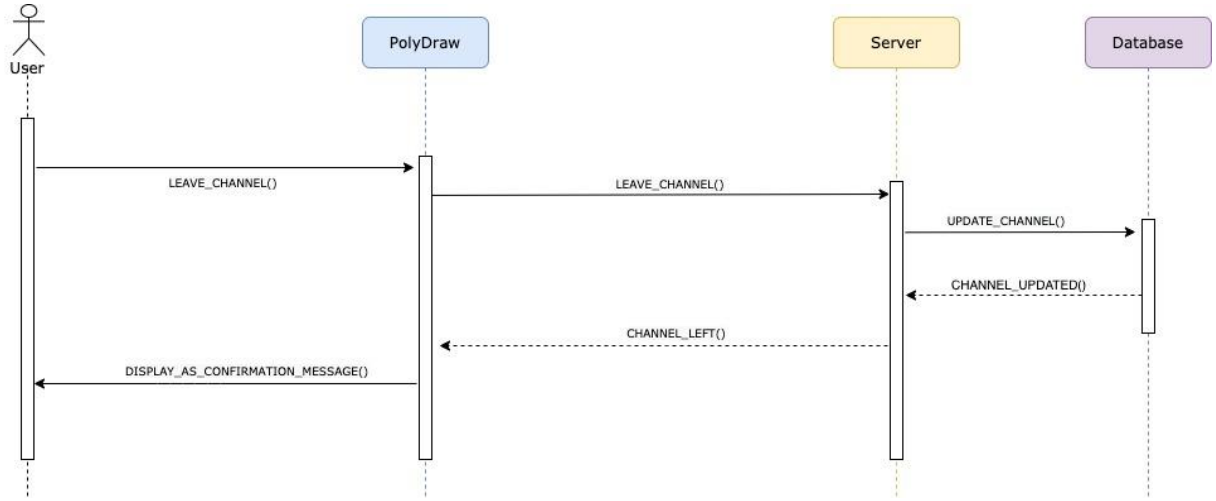

Figure 5.4.4: Présentation de la séquence ***quitter un canal*** issue de « gérer un *chat* »

5.5 Créer un jeu

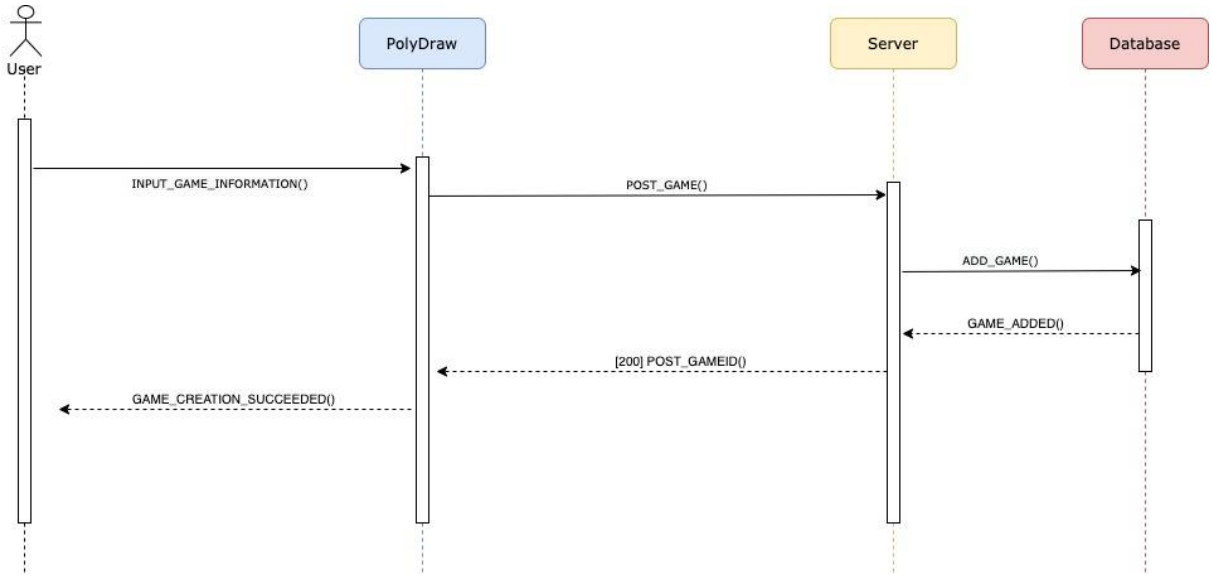

Figure 5.5: Présentation de la séquence « créer un jeu »

5.6 Gérer un profil

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

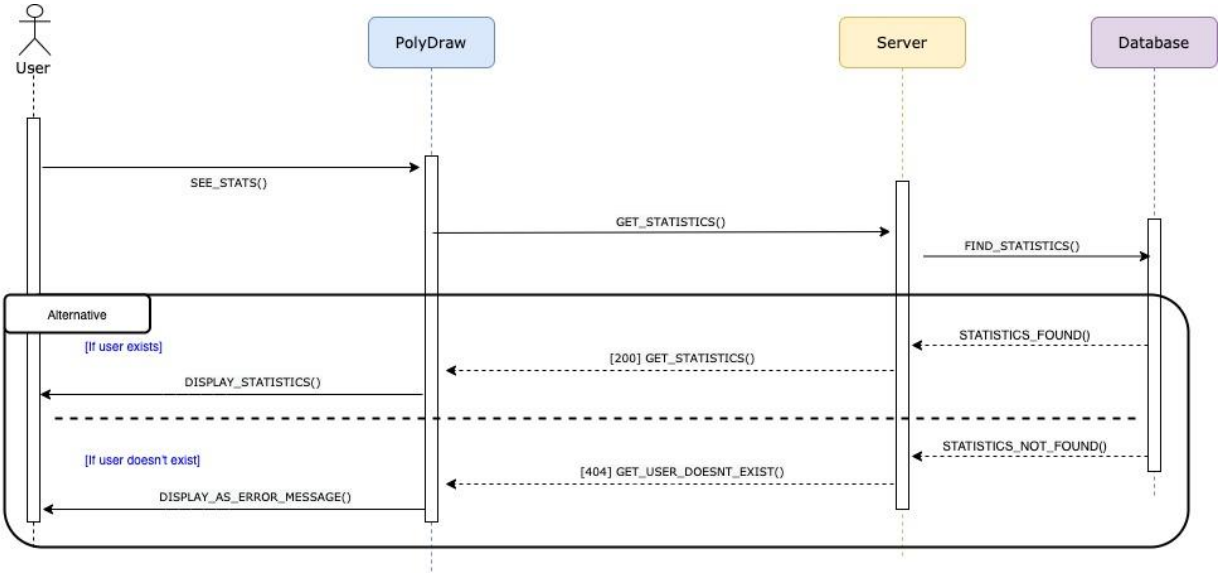

Figure 5.6.1: Présentation de la séquence *récupérer les statistiques* issue de « gérer un profil »

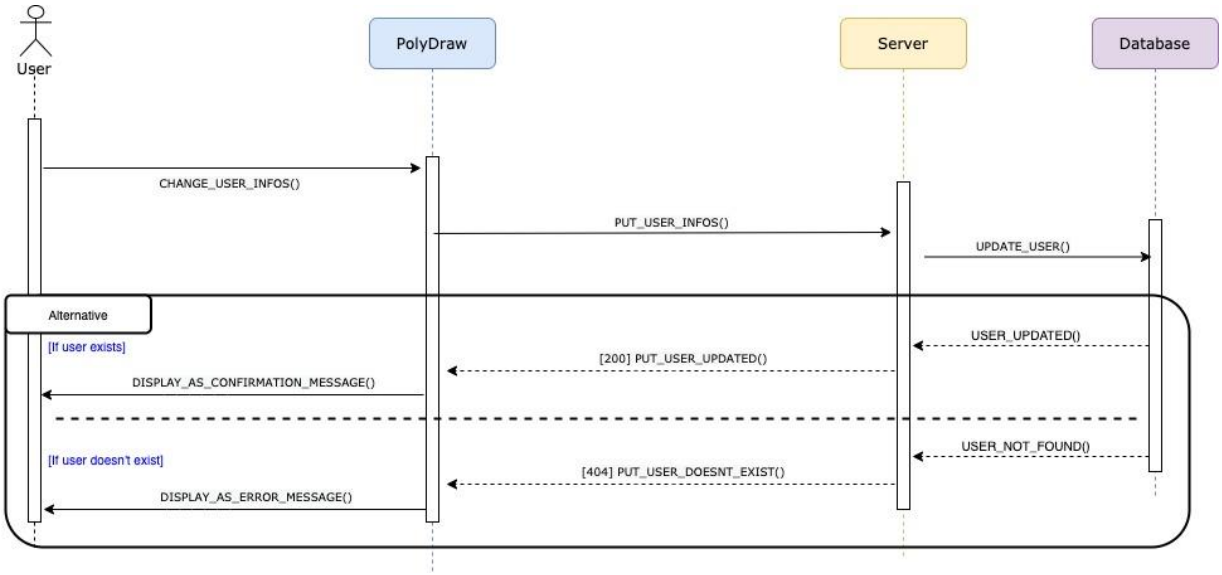

Figure 5.6.2: Présentation de la séquence *modifier les informations utilisateur* issue de « gérer un profil »

5.7 Créer une partie (lobby)

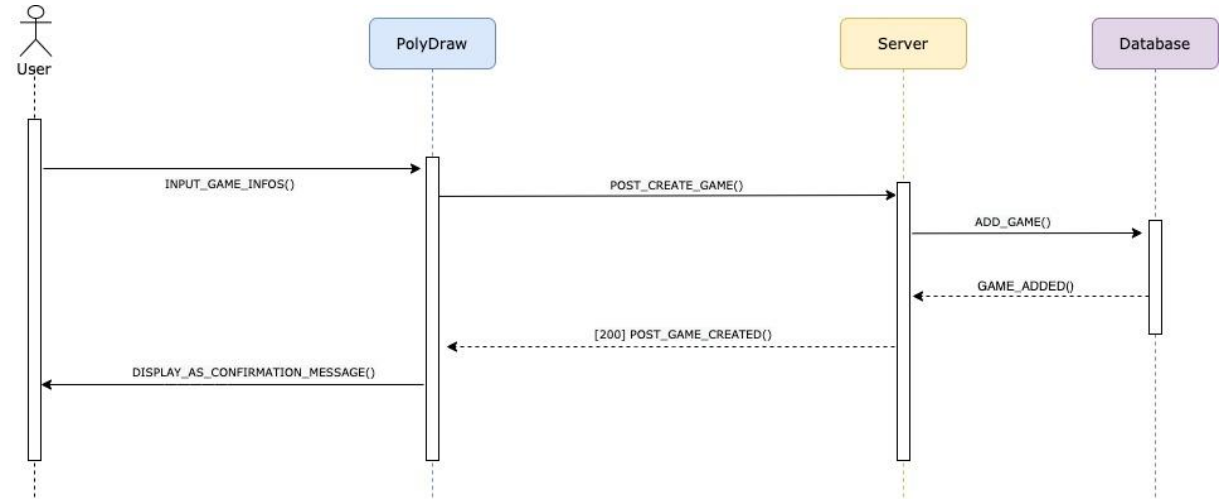

Figure 5.7: Présentation de la séquence « créer une partie »

5.8 Jouer à une partie

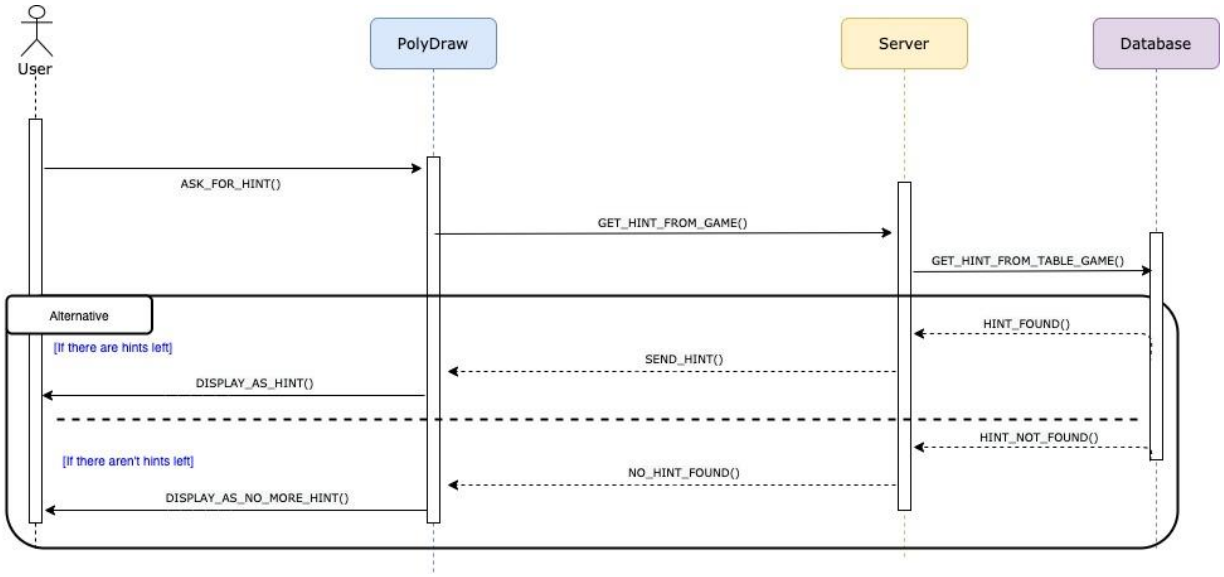

Figure 5.8.1: Présentation de la séquence *demander un indice* issue de « jouer à une partie »

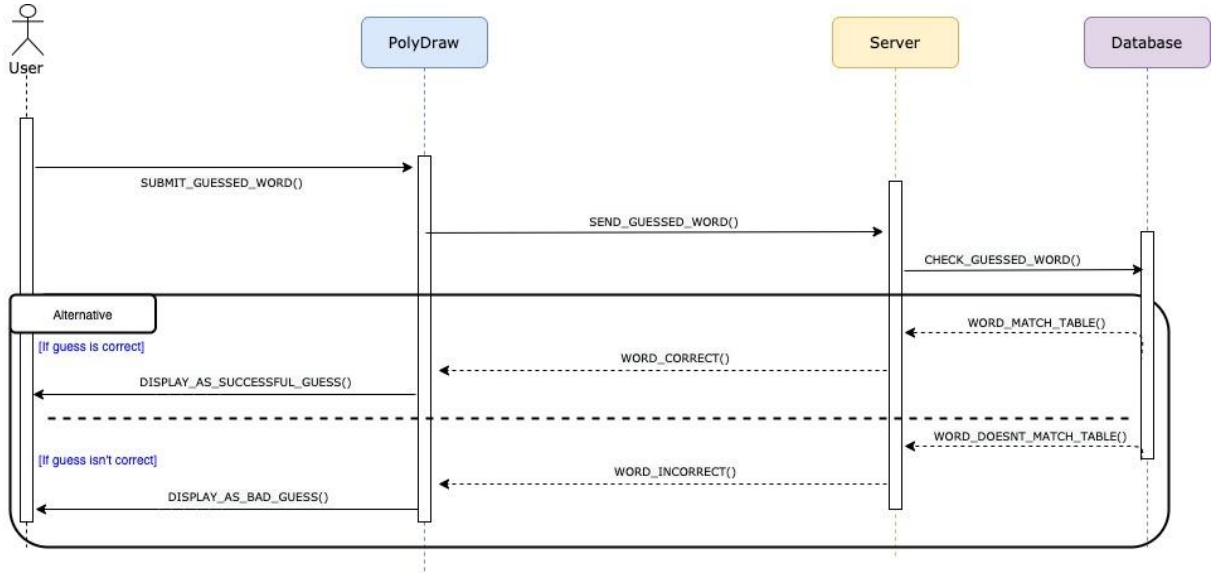

Figure 5.8.2: Présentation de la séquence *soumettre un mot* issue de « jouer à une partie »

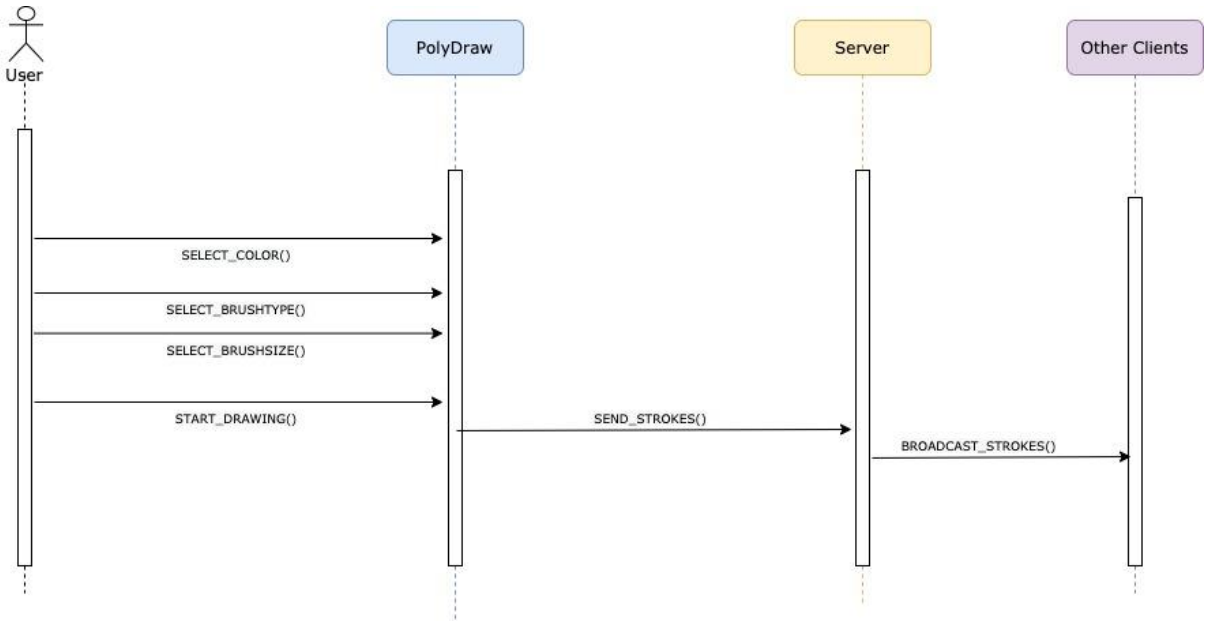

Figure 5.8.3: Présentation de la séquence ***dessiner*** issue de « jouer à une partie »

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 cœurs 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 s'occuper des listes. 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.

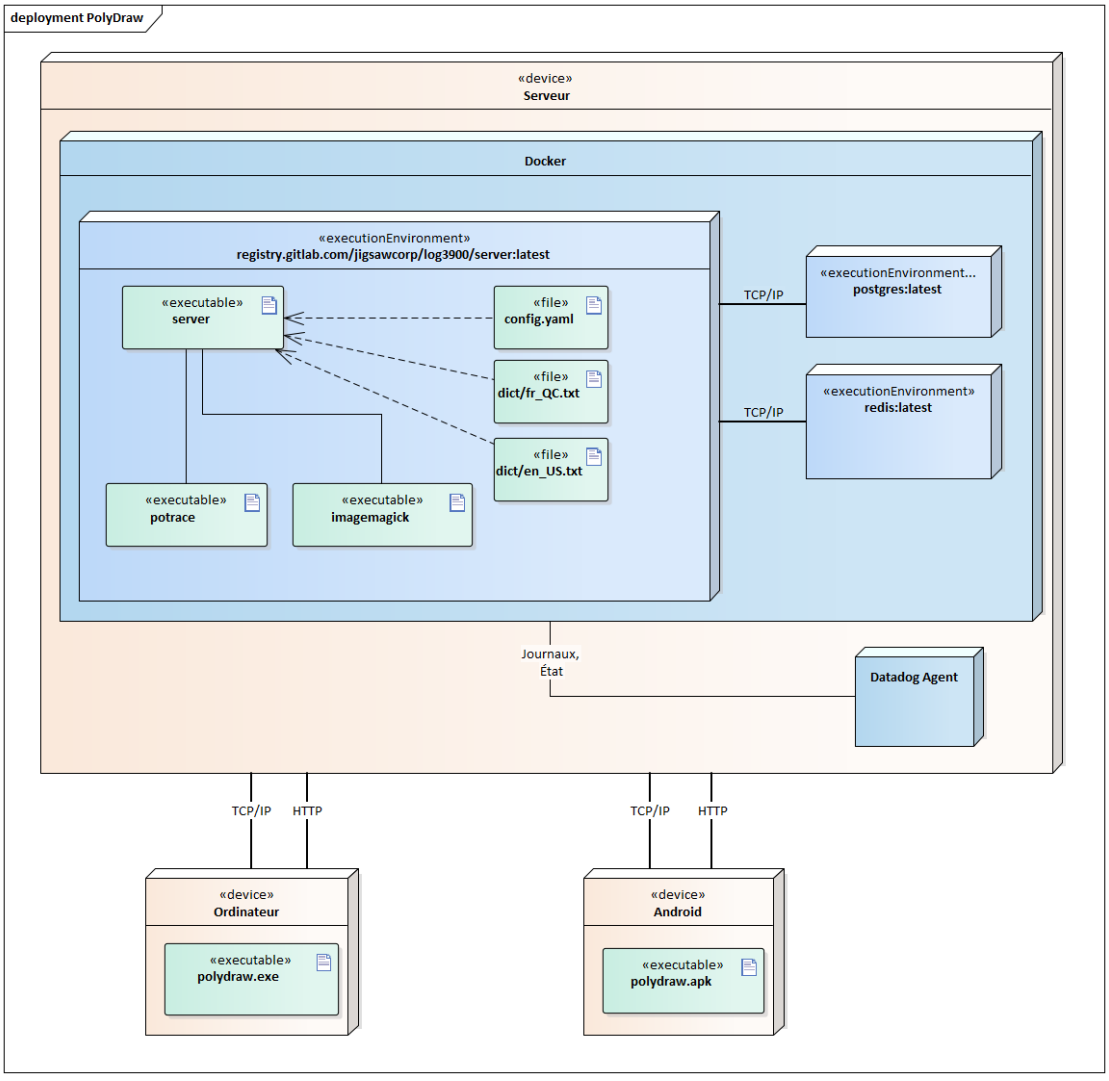

Figure 6: Diagramme de déploiement de PolyDraw

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 manœuvre. 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 2 cœurs virtuels. Il y a également d'autres services qui roulent sur le serveur comme un agent de surveillance, un SGBD, et Redis. 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. La base de données PostgreSQL doit pouvoir répondre en moyenne en moins de 5ms. Ce taux de réponse à pour but de s'assurer que l'application n'est pas trop ralentie par la base de données. L'agent Datadog permet d'avertir dans le cas où les requêtes seraient trop lentes. Le système de cache de Redis doit répondre à toute les requêtes O(1) en moins de 1ms. Ce cache est utilisé principalement dans le cas L'agent de surveillance surveille ces performances. 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.

Afin d'offrir des performances une latence maximale à 100 ms est recommandée. Le logiciel possède une fonctionnalité de vérification de l'état de la connexion. Le serveur envoie un message aux 5 secondes afin de s'assurer que le client est toujours connecté. Il autorise les clients cependant à prendre jusqu'à 5s pour répondre à ces messages. Cependant l'expérience utilisateur risque d'être dégradée si le client à une latence de plus de 300 ms avec le serveur. Dans le cas où la connexion serait intermittente l'utilisateur serait déconnecté de l'application avec un message d'erreur lui expliquant que la connexion a coupée. Si l'utilisateur est dans une partie, celle-ci va réagir en conséquence et s'occuper de fermer celle-ci s'il n'y a pas assez de joueurs.

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.