Les Décalés

LNAH 2K17
Document d'architecture logicielle

Version 3.1

Historique des révisions

Date	Version	Description	Auteur
2017-09-18	1.0	Rédaction de l'introduction	Mikaël Ferland
2017-09-21	2.0	Dessiner les diagrammes de cas d'utilisation et les diagrammes de séquence	Mikaël Ferland et Jean-Marc Al-Romhein
2017-09-26	2.1	Réviser les diagrammes de cas d'utilisation	Mikaël Ferland
2017-09-26	2.2	Rédaction du texte pour toutes les sections au brouillon	Jean-Marc Al-Romhein
2017-09-27	2.3	Relecture et correction de l'introduction et des contraintes architecturales, des diagrammes de paquetage	Pierre To
2017-09-27	2.4	Ajout des diagrammes de paquetages	Ariane Tourangeau
2017-09-27	2.5	Réviser les diagrammes de séquence et intégrer les diagrammes de cas d'utilisation dans le rapport	Mikaël Ferland
2017-09-28	2.6	Insertion des diagrammes de séquences et leurs descriptions. Insertion du diagramme de déploiement. Rédaction de la section taille et performance	Jean-Marc Al-Romhein
2017-09-28	2.7	Modification des diagrammes de paquetages et ajout des descriptions de paquetage	Ariane Tourangeau
2017-09-28	2.8	Ajout du diagramme de paquetage du site web	Jean-Marc Al-Romhein
2017-09-28	2.9	Correction du document en entier	Ariane Tourangeau Philippe Lelièvre Jean-Marc Al-Romhein
2017-09-29	3.0	Ajout des nouveaux diagrammes de paquetage	Mikaël Ferland
2017-09-29	3.1	Relecture et mise en page	Pierre To

Table des matières

Document d'architecture logicielle

1. Introduction

Le présent document offre un aperçu de l'architecture logicielle de haut niveau du client lourd, du client léger, du client web et du serveur pour l'application d'*air hockey*. En premier lieu, il s'agit d'énoncer les objectifs que doit atteindre l'architecture logicielle ainsi que les critères à prendre en compte au moment de la conception et de l'implémentation. En second lieu, les cas d'utilisation les plus importants sont illustrés. La vue logique de l'application, la vue des processus et celle du déploiement sont aussi abordées. Finalement, il s'agit d'expliquer en quoi les caractéristiques de taille et de performance de l'application ont influencé la structure du projet.

2. Objectifs et contraintes architecturaux

L'objectif du projet est de réaliser une application d'*air hockey* sur un client lourd et sur un client léger, le tout accompagné d'un site web. De prime abord, une architecture client-serveur est choisie. Étant donné la nécessité d'échanger de l'information en temps réel pour le mode de jeu et l'édition, l'utilisation de socket a été choisi. Enfin, pour les autres types de requêtes ne nécessitant pas de notion de temps réel, les requêtes REST seront utilisées pour leur comportement prévisible.

Le code source produit doit posséder une bonne architecture pour faciliter le développement ainsi que le maintien du code. Les composantes du projet doivent donc être modulaires pour permettre aux futurs développeurs de facilement ajouter des fonctionnalités. Ainsi, l'utilisation des patrons de conception, dont le patron visiteur, le patron état et le patron usine, est favorisée.

Une des principales contraintes architecturales provient du projet préexistant. Il est important de conserver l'architecture déjà en place pour garder une uniformité dans le code et pour préserver les fonctionnalités de base. Se conformer à cette architecture peut imposer des limitations au niveau de l'implémentation des nouvelles fonctionnalités.

Pour le client léger, le choix de développer en Swift réduit la portabilité du produit. La plateforme sera limitée à des produits iOS. Au niveau de l'implémentation du client léger, il sera avantageux de respecter l'architecture du client lourd, étant donné qu'il s'agit des mêmes fonctionnalités dans la plupart des cas.

L'utilisation d'une base de données ajoute une couche supplémentaire à l'architecture pour pouvoir faire le lien entre les objets utilisés dans le code et les éléments dans la base de données. Les valeurs des colonnes dans les tables de la base de données sont liées aux attributs des entités.

3. Vue des cas d'utilisation

La section qui suit présente douze principaux cas d'utilisation (figures 1 à 12) pour le système d'*air hockey*. Notons que les cas d'utilisation illustrent une vision globale du comportement fonctionnel du logiciel ainsi que l'interaction entre l'utilisateur et le système.

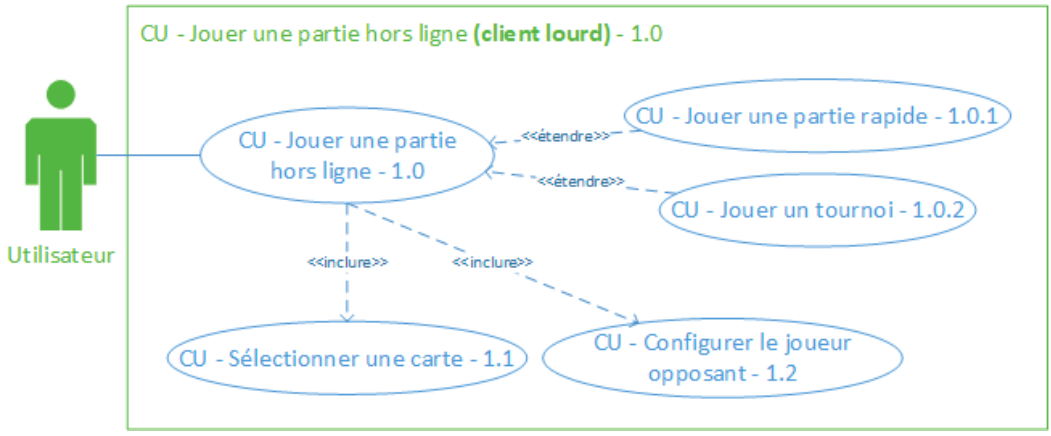

Figure 1 : Cas d'utilisation 1.0 - Jouer une partie hors ligne (client lourd)

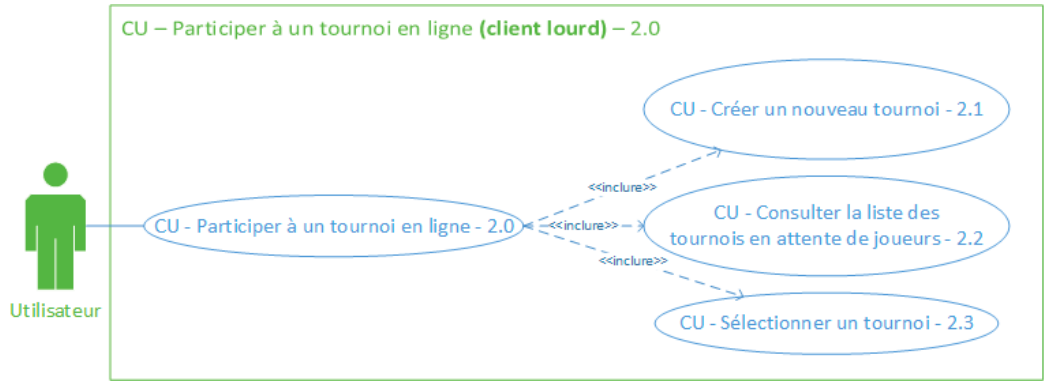

Figure 2 : Cas d'utilisation 2.0 - Participer à un tournoi en ligne (client lourd)

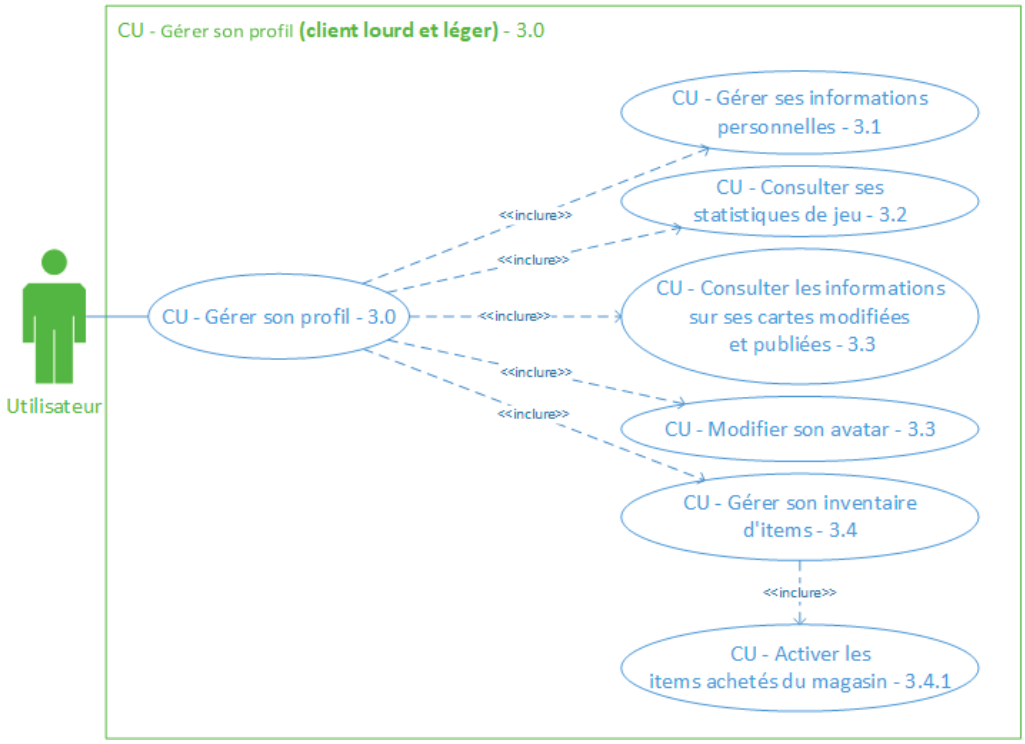

Figure 3 : Cas d'utilisation 3.0 - Gérer son profil (client lourd et léger)

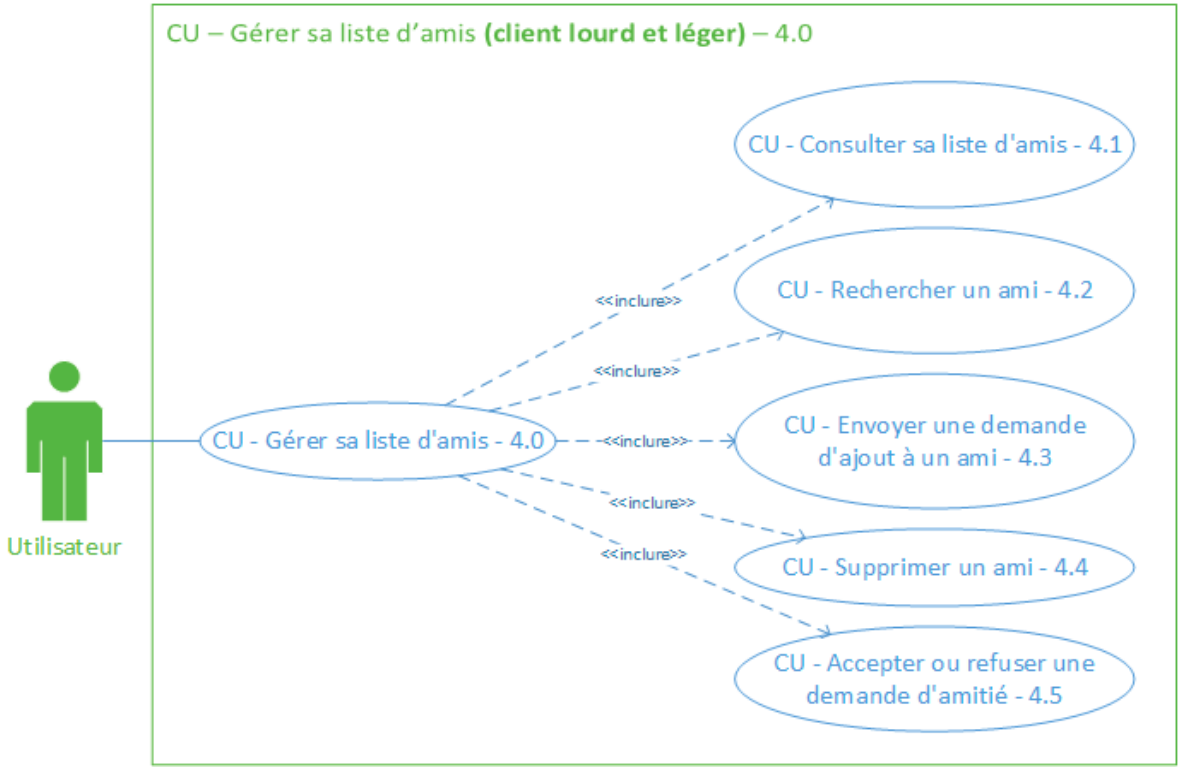

Figure 4 : Cas d'utilisation 4.0 - Gérer sa liste d'amis (client lourd et léger)

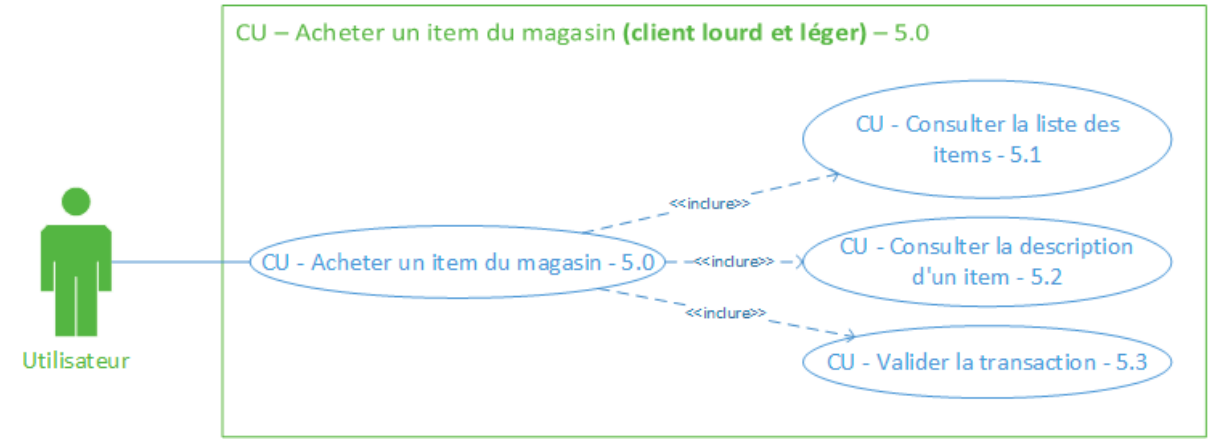

Figure 5 : Cas d'utilisation 5.0 - Acheter un item du magasin (client lourd et léger)

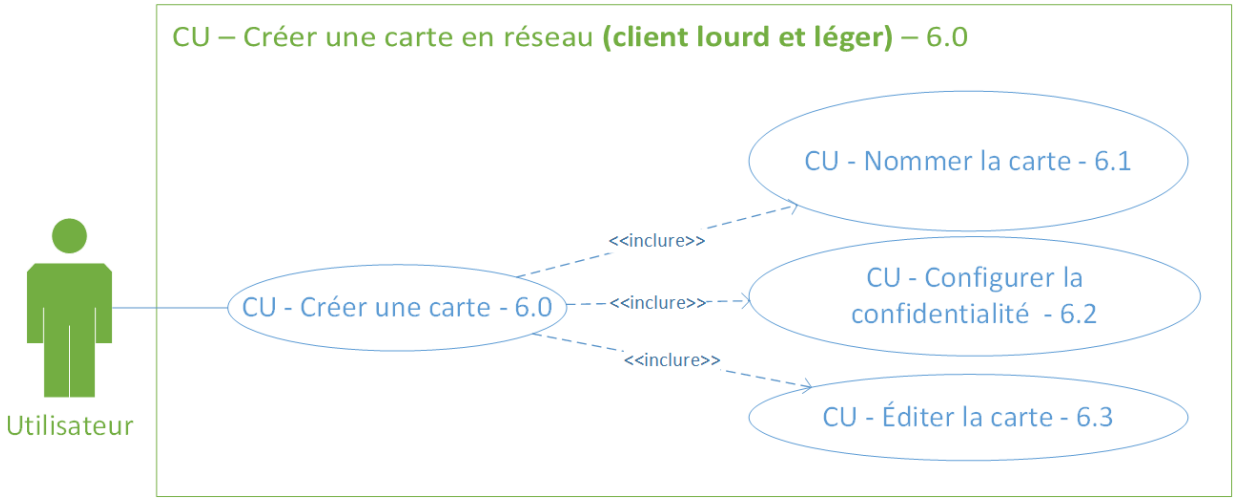

Figure 6 : Cas d'utilisation 6.0 - Créer une carte en réseau (client lourd et léger)

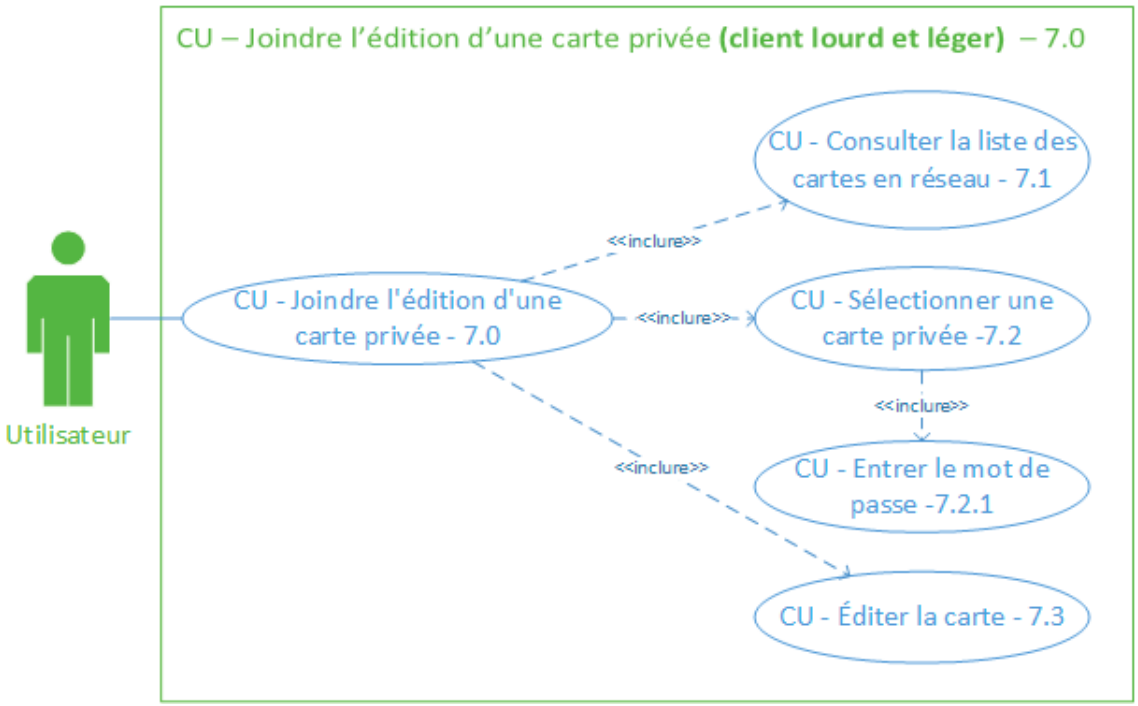

Figure 7 : Cas d'utilisation 7.0 - Joindre l'édition d'une carte privée (client lourd et léger)

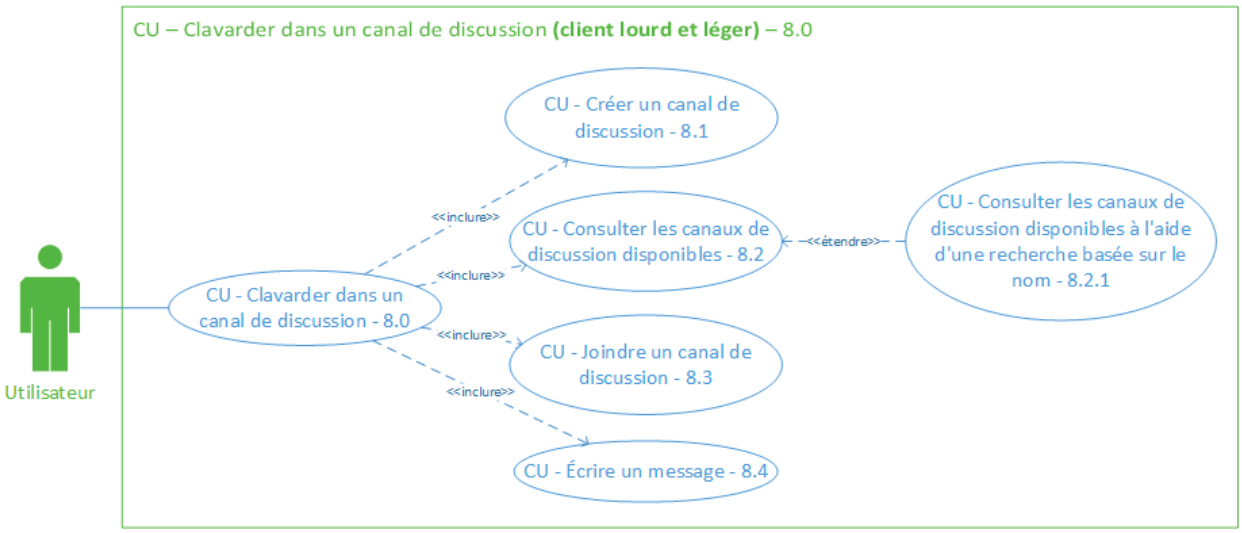

Figure 8 : Cas d'utilisation 8.0 - Clavarder dans un canal de discussion (client lourd et léger)

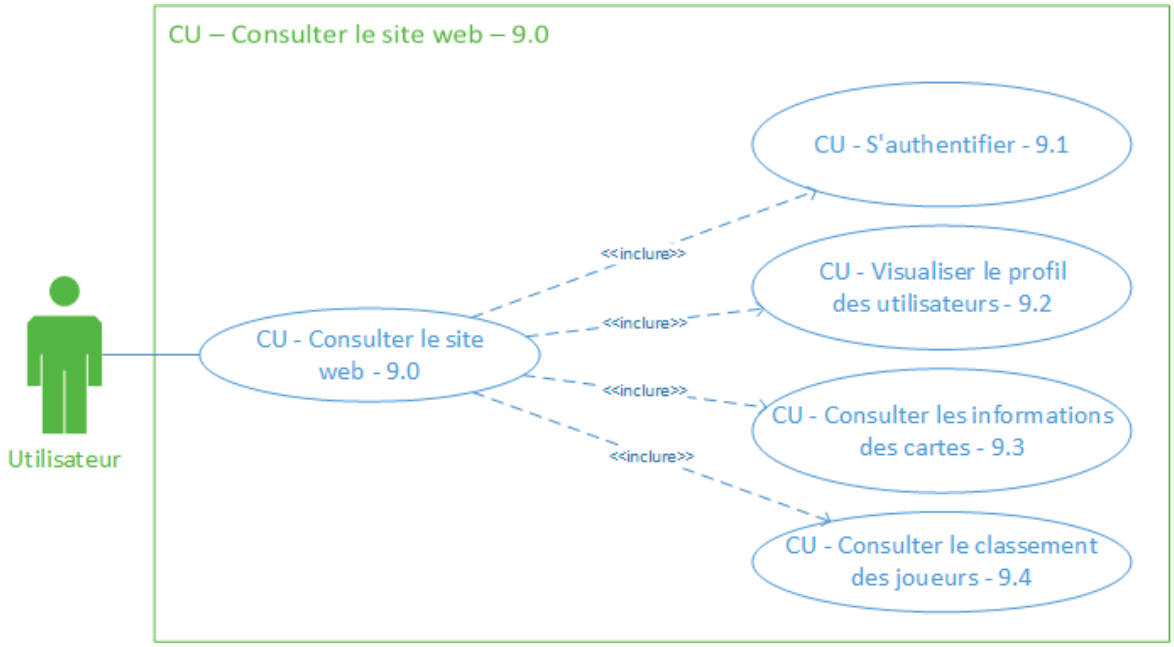

Figure 9 : Cas d'utilisation 9.0 - Consulter le site web

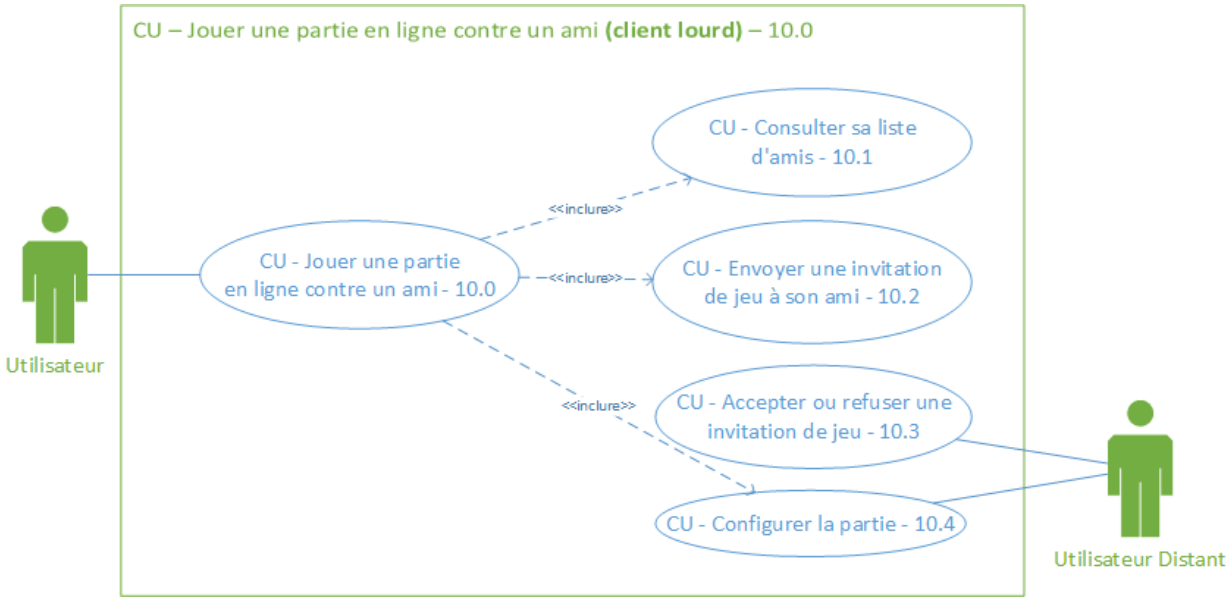

Figure 10 : Cas d'utilisation 10.0 - Jouer une partie en ligne contre un ami (client lourd)

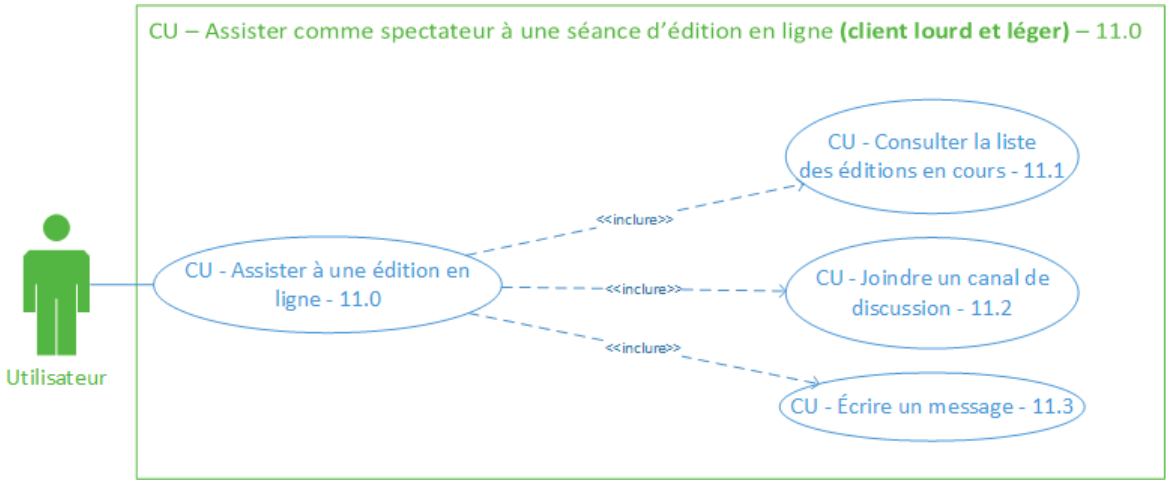

Figure 11 : Cas d'utilisation 11.0 - Assister comme spectateur à une séance d'édition en ligne (client lourd)

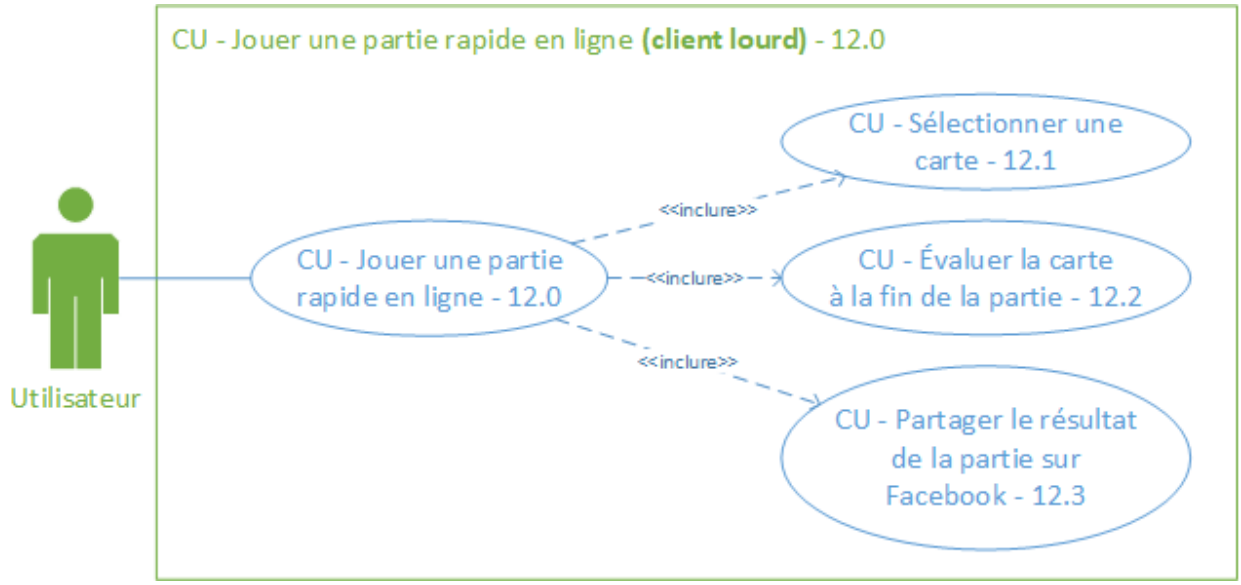

Figure 12 : Cas d'utilisation 12.0 - Jouer une partie rapide en ligne

4. Vue logique

La section qui suit présente les quatre diagrammes de paquetage pour le système d'*air hockey*, c'est-à-dire le client lourd, le client léger, le serveur et le client web. Afin d'améliorer la visibilité des diagrammes, un document en pièce jointe nommé ***4.1-Diagrammes_paquetages-LNAH_2K17-Les_Decales.pdf*** est fourni.

La figure suivante montre le diagramme de paquetage du client lourd. Des tableaux indiquent la responsabilité de tous les paquetages présentés dans chaque composant du produit.

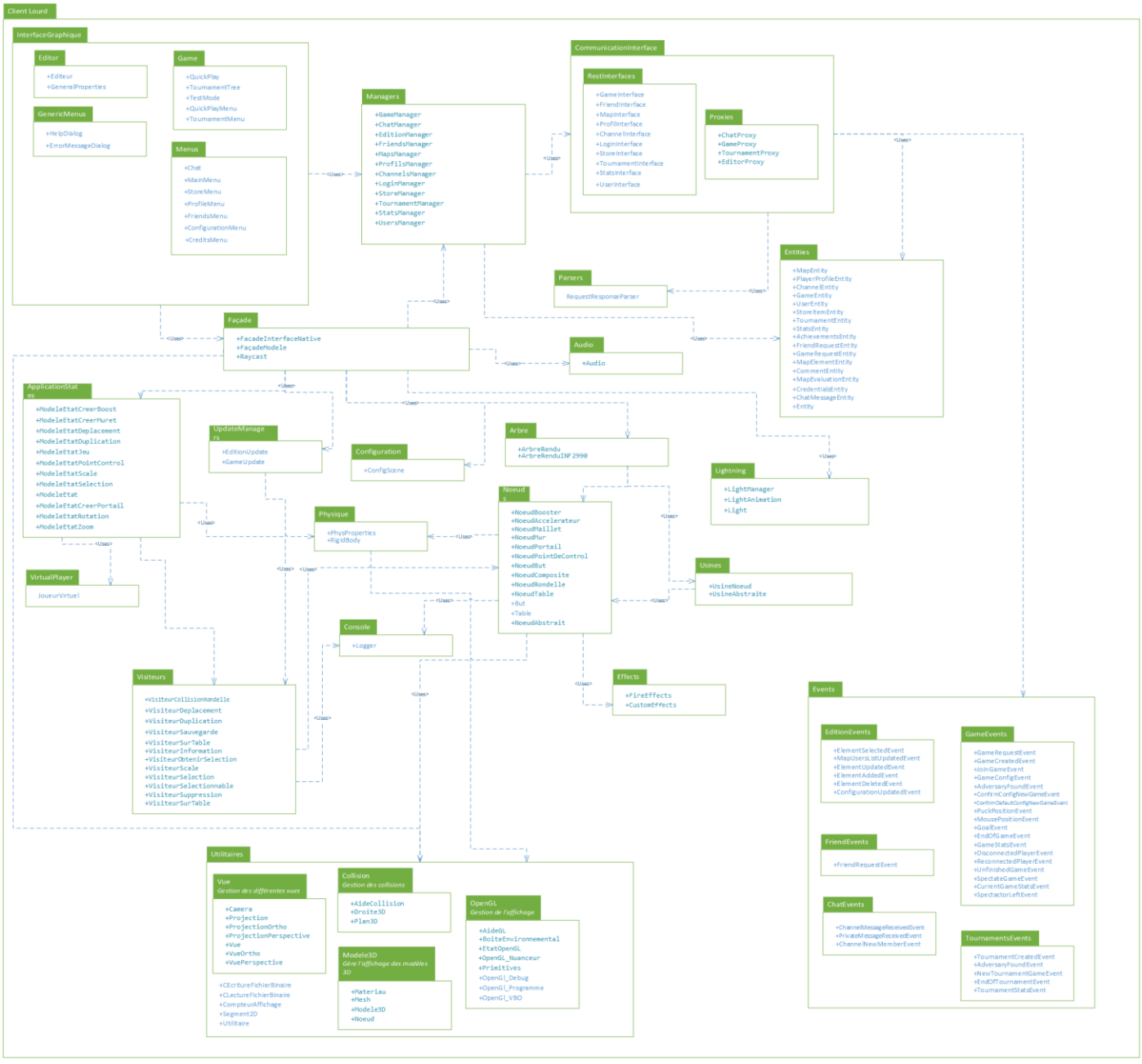

Figure 13 : Diagramme de paquetage du client lourd

InterfaceGraphique
Ce paquetage contient les classes (et paquetages) permettant à l'utilisateur d'interagir avec l'application. Il capture les différents types d'entrée afin de transmettre l'entrée au *manager*.

Editor
Ce paquetage contient les classes permettant à l'utilisateur d'interagir avec le mode édition de zones de jeu en ligne ou localement.

Game
Ce paquetage contient les classes permettant à l'utilisateur d'interagir avec le mode de jeu "partie rapide" ou "tournoi".

GenericMenus
Ce paquetage contient les classes permettant à l'utilisateur d'interagir avec les menus génériques tels que les messages d'erreurs et le tutoriel.

Menu
Ce paquetage contient les classes permettant à l'utilisateur d'interagir avec les différents modes "simples", c'est-à-dire qu'ils ne devraient pas nécessiter plus d'une ou deux vues. On retrouve par exemple le clavardage, le magasin, le profil du joueur ou encore la liste d'amis.

Managers
Ce paquetage contient les différents *managers* utilisés pour réagir adéquatement suite à une entrée de l'utilisateur. Certaines actions nécessitent une synchronisation avec le serveur, l'exécution d'une action dans le noyau ou les deux. Les *managers* permettent ainsi de coordonner les différentes actions. Aussi, les *managers* réagissent aux évènements lancés par le serveur afin de mettre la jour la vue et l'arbre de rendu.

CommunicationInterface
Ce paquetage contient les différents paquetages permettant la communication entre le client et le serveur. Les paramètres envoyés au serveur sont toujours une chaîne de caractère où la valeur est la sérialisation en *Json* des différentes classes du paquetage *Entities*.

Proxies
Chaque classe contenue dans ce paquetage gère une connexion par socket avec le serveur. Le client lourd entretiendra en tout temps, à l'exception d'être en mode hors ligne, une connexion pour le chat, l'édition en ligne, les parties rapides en ligne et les tournois en ligne. Chaque classe s'abonne à certains évènements lancés par le serveur pour ensuit les transmettre au *manager* adéquat.

RestInterface
Chaque classe contenue dans ce paquetage gère l'accès au REST API sur le serveur. Elles permettent de modifier, créer, mettre à jour ou récupérer des entités. Elles utilisent le paquetage *Parsers* pour désérialiser les réponses HTTP du serveur.

Entities
Chaque classe contenue dans ce paquetage représente un objet du domaine de l'application. Ces objets permettent de partager l'information uniformément à travers les différentes couches de l'application.

Parsers
Ce paquetage a pour responsabilité de recevoir en paramètre les réponses HTTP du serveur et de désérialiser le contenu *Json* en *entities*.

Façade
Les classes de ce paquetage permettent de gérer l'appel de fonctions C++ à partir de l'interface codée en C#. Étant le point d'entrée du noyau, il agit à titre de *dispatcher*.

Audio
Ce paquetage gère l'audio de l'application. Il permet de contrôler les trames sonores ainsi que les différents effets sonores.

ApplicationStates
Ce paquetage permet de gérer les différents états de l'application. Selon l'état dans lequel l'application se trouve, les entrées de l'utilisateur impacte l'application d'une manière différente. Ainsi, chaque classe implémente sa propre réaction suite à un type d'entrée précis.

UpdateManager
Chaque classe contenue dans ce paquetage gère la mise à jour d'éléments par d'autres utilisateurs sur le réseau. Il appliquera les changements reçus au bon nœud dans l'arbre de rendu à l'aide du paquetage *visiteurs*.

Configuration
Ce paquetage gère les changements apportés par l'utilisateur au sein de la configuration. Il conserve la valeur de chacun des facteurs configurables et permet l'accès.

Visiteurs
Chaque classe de ce paquetage a pour but de visiter les différents nœuds de l'arbre de rendu et d'exécuter une action spécifique au type de nœud.

Nœuds
Chaque classe contenue dans ce paquetage représente un élément présent au dans le jeu. Ces éléments contiennent les informations nécessaires pour exécuter des actions telles que la duplication ou la translation et l'affichage au bon emplacement des éléments dans la vue.

Arbre
Ce paquetage a pour responsabilité de hiérarchiser les différents nœuds (élément du jeu). Il permet l'exécution de fonctions simples telles que l'affichage, l'ajout, la suppression etc. Il est parcouru par un visiteur afin d'appliquer les changements sur chacun des nœuds.

VirtualPlayer
Ce paquetage a pour but de calculer les informations relatives au maillet d'un joueur tels que la vitesse ou encore la position.

Lighting
Ce paquetage a pour but de gérer les différents types de lumière disponibles dans le jeu ainsi que de modifier les différents angles.

Usines
Les classes contenues dans ce paquetage permettent de créer les différents nœuds et de les ajouter dans l'arbre de rendu. Cela permet d'abstraire la manière dont on crée les objets.

Physique
Ce paquetage a pour de but de calculer la nouvelle position des éléments suite à une action d'un utilisateur. Le calcul se fait à l'aide du paquetage *utilitaire*.

Console
Ce paquetage a pour but d'afficher de manière uniforme de l'information dans la console.

Events
Les différentes classes représentent les évènements qui peuvent être lancés par le serveur et capté par le client. Le client peut donc réagir selon le type d'événement capté.

EditionEvents
Ce paquetage représente les différents événements reliés à l'édition en ligne d'une zone de jeu tels que la translation, la rotation ou encore la suppression d'un objet par un utilisateur distant.

GameEvents
Ce paquetage représente les différents événements reliés à une partie jouée en ligne tels que la position de la rondelle, la position de l'adversaire ou encore la fin d'une partie. C'est par le biais de ces évènements que la gestion d'une partie entre le serveur et les différents joueurs s'effectue.

ChatEvents
Ce paquetage représente les différents événements reliés au chat. Un évènement sera envoyé lors de la réception d'un message dans un canal ou de la part d'un joueur spécifique et lorsqu'un joueur est ajouté à un canal.

FriendEvents
Ce paquetage représente les événements reliés aux amis de l'utilisateur. Pour l'instant le seul événement auquel le client peut s'abonner est la réception d'une demande d'amis.

TournamentEvent
Ce paquetage représente les différents événements reliés aux tournois en ligne tels que le démarrage d'une nouvelle ronde du tournoi ou encore la fin de ce dernier. C'est par le biais de ces évènements que la gestion d'un tournoi entre le serveur et les différents joueurs s'effectue.

Effects
Ce paquetage représente les différents effets possibles durant une partie. Ce dernier a pour responsabilité d'appliquer l'impact d'un effet sur une partie.

Utilitaires
Ce paquetage permet de faciliter certains calculs et l'utilisation des librairies *openGL*. Les tâche indépendantes du contexte sont exécutées au sein de ce paquetage.

Vue
Ce paquetage permet le changement entre les différentes vues. Il effectue les calculs nécessaires à chaque vue.

Collision
Ce paquetage permet de calculer si une collision entre deux corps survient à un moment précis selon la position de deux éléments.

Modele3D
Ce paquetage permet de créer et d'afficher les différents modèles reliés aux nœuds dans l'arbre de rendu.

OpenGL
Ce paquetage permet l'affichage graphique via les différentes librairies.

La figure suivante montre le diagramme de paquetage du client léger.

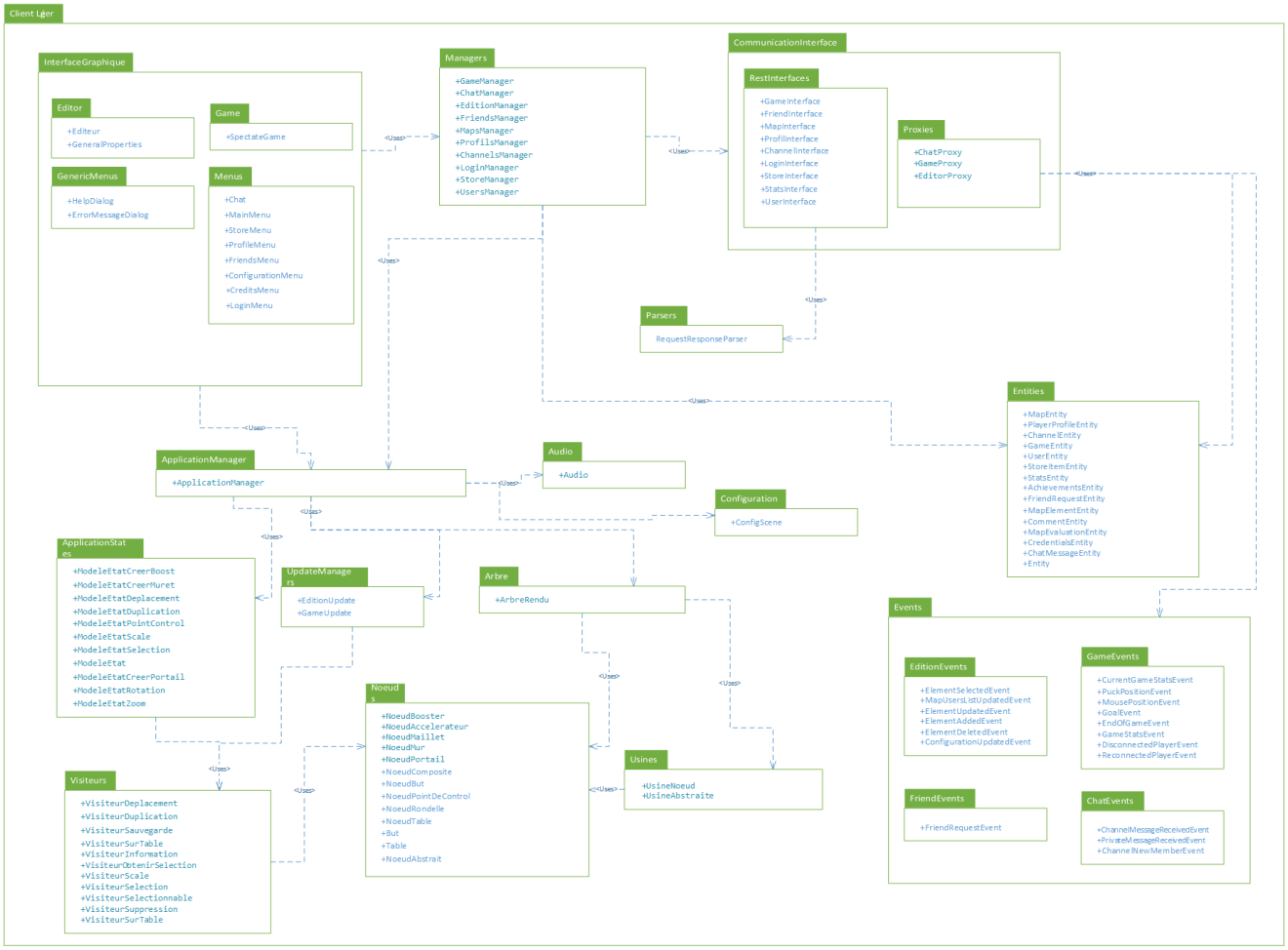

Figure 14 : Diagramme de paquetage du client léger

InterfaceGraphique
Ce paquetage contient les classes (et paquetages) permettant à l'utilisateur d'interagir avec l'application. Il capture les différents types d'entrée tactile et transmet cette dernière au *manager*.

Editor
Ce paquetage contient les classes permettant à l'utilisateur d'interagir avec le mode édition de zones de jeu en ligne ou localement.

Game
Ce paquetage contient les classes gérant le mode spectateur.

Menu
Ce paquetage contient les classes permettant à l'utilisateur d'interagir avec les différents modes "simples", c'est-à-dire qu'ils ne devraient pas nécessiter plus d'une ou deux vues. On retrouve par exemple le clavardage, le magasin, le profil du joueur ou encore la liste d'amis.

GenericMenus
Ce paquetage contient les classes permettant à l'utilisateur d'interagir avec les menus génériques tels que les messages d'erreurs et le tutoriel.

Managers
Ce paquetage contient les différents *managers* utilisés pour réagir adéquatement suite à une entrée de l'utilisateur. Certaines actions nécessitent une synchronisation avec le serveur, l'exécution d'une action dans le noyau ou les deux. Les *managers* permettent ainsi de coordonner les différentes actions. Aussi, les *managers* réagissent aux évènements lancés par le serveur afin de mettre la jour la vue et l'arbre de rendu.

CommunicationInterface
Ce paquetage contient les différents paquetages permettant la communication entre le client et le serveur. Les paramètres envoyés au serveur sont toujours une chaîne de caractère où la valeur est la sérialisation en *Json* des différentes classes du paquetage *Entities*.

Proxies
Chaque classe contenue dans ce paquetage gère une connexion par socket avec le serveur. Le client léger entretiendra en tout temps, à l'exception d'être en mode hors ligne, une connexion pour le chat, l'édition en ligne, les parties en lignes en mode spectateur. Chaque classe s'abonne à certains évènements lancés par le serveur pour ensuit les transmettre au *manager* adéquat.

RestInterface
Chaque classe contenue dans ce paquetage gère l'accès au REST API sur le serveur. Elles permettent de modifier, créer, mettre à jour ou récupérer des entités. Elles utilisent le paquetage *Parsers* pour désérialiser les réponses HTTP du serveur.

Entities
Chaque classe contenue dans ce paquetage représente un objet du domaine de l'application. Ces objets permettent de partager l'information uniformément à travers les différentes couches de l'application.

Parsers
Ce paquetage a pour responsabilité de recevoir en paramètre les réponses HTTP du serveur et de désérialiser le contenu *Json* en *entities*.

Audio
Ce paquetage gère l'audio de l'application. Il permet de faire jouer une chanson et de l'arrêter ainsi que les différents effets sonores.

ApplicationManager
Ce paquetage détient l'état l'application et agit à titre de *dispatcher*. Il propose les actions possibles relatives à l'arbre de rendu aux *managers.*

ApplicationStates
Ce paquetage permet de gérer les différents états de l'application. Selon l'état dans lequel l'application se trouve, les entrées de l'utilisateur impacte l'application d'une manière différente. Ainsi, chaque classe implémente sa propre réaction suite à un type d'entrée précis.

UpdateManager
Chaque classe contenue dans ce paquetage gère la mise à jour d'éléments par d'autres utilisateurs sur le réseau. Il appliquera les changements reçus au bon nœud dans l'arbre de rendu à l'aide du paquetage *visiteurs*.

Visiteurs
Chaque classe de ce paquetage a pour but de visiter les différents nœuds de l'arbre de rendu et d'exécuter une action spécifique au type de nœud.

Nœuds
Chaque classe contenue dans ce paquetage représente un élément présent au dans le jeu. Ces éléments contiennent les informations nécessaires pour exécuter des actions telles que la duplication ou la translation et l'affichage au bon emplacement des éléments dans la vue.

Arbre
Ce paquetage a pour responsabilité de hiérarchiser les différents nœuds (élément du jeu). Il permet l'exécution de fonctions simples telles que l'affichage, l'ajout, la suppression etc. Il est parcouru par un visiteur afin d'appliquer les changements sur chacun des nœuds.

Usines
Les classes contenues dans ce paquetage permettent de créer les différents nœuds et de les ajouter dans l'arbre de rendu. Cela permet d'abstraire la manière dont on crée les objets.

Events
Les différentes classes représentent les évènements qui peuvent être lancés par le serveur et capté par le client. Le client peut donc réagir selon le type d'événement capté.

EditionEvents
Ce paquetage représente les différents événements reliés à l'édition en ligne d'une zone de jeu tels que la translation, la rotation ou encore la suppression d'un objet par un utilisateur distant.

GameEvents
Ce paquetage représente les différents événements reliés à la participation à une partie en ligne en mode spectateur.

ChatEvents
Ce paquetage représente les différents événements reliés au chat. Un évènement sera envoyé lors de la réception d'un message dans un canal ou de la part d'un joueur spécifique et lorsqu'un joueur est ajouté à un canal.

FriendEvents
Ce paquetage représente les événements reliés aux amis de l'utilisateur. Pour l'instant le seul événement auquel le client peut s'abonner est la réception d'une demande d'amis.

Modele2D
Ce paquetage permet l'affichage des différents modèles 2D dans l'éditeur et le mode spectateur.

La figure suivante montre le diagramme de paquetage du site web. Étant donné que le site web sera implémenté à l'aide d'*Angular*, le diagramme de paquetage du site web respecte l'architecture proposée par *Angular*.

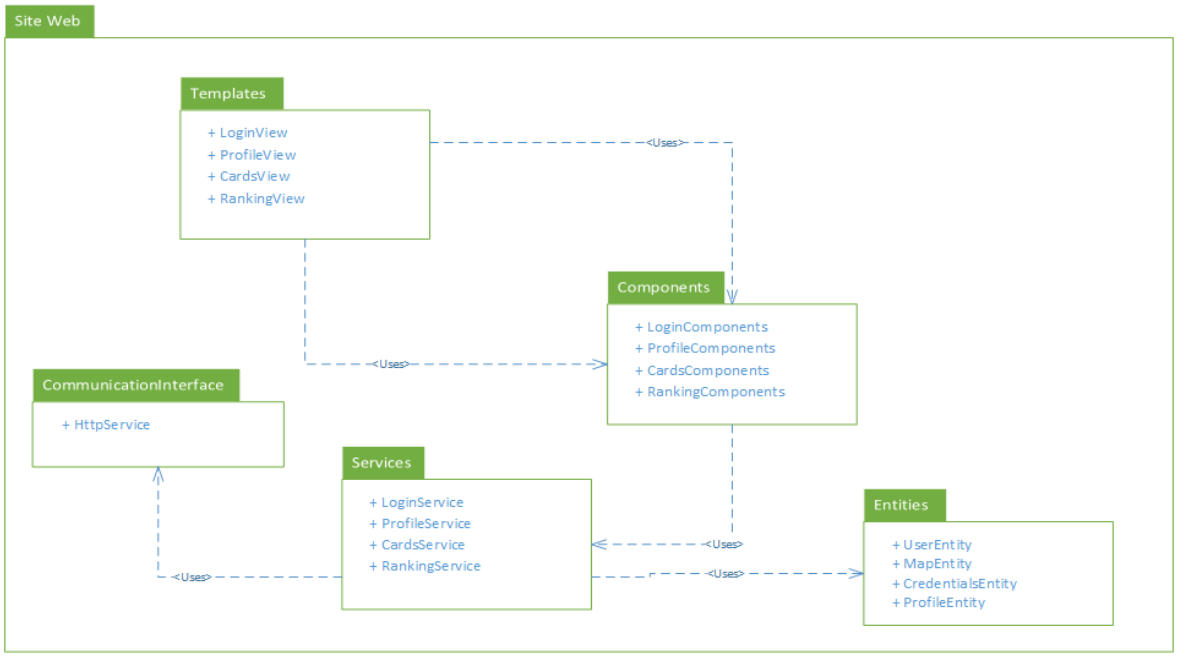

Figure 15 : Diagramme de paquetage du site web

Templates
Ce paquetage permet à l'utilisateur d'interagir avec le site web. Il capture les différents types d'entrée tactile et les transmet aux *components*.

Components
Ce paquetage est utilisé pour réagir adéquatement à une entrée de l'utilisateur. Certaines actions nécessitent une synchronisation avec le serveur, le changement de vue ou les deux. Les *components* permettent ainsi de coordonner les différentes actions. La plupart du temps, il utilisera déléguera les actions au paquetage *Services.*

CommunicationInterface
Ce paquetage permet la communication entre le client et le REST API du serveur à l'aide de requête HTTP. Les paramètres envoyés au serveur sont toujours une chaîne de caractère où la valeur est la sérialisation en *Json* des différentes classes du paquetage *Entities*.

Services
Ce paquetage représente les services qui ont pour responsabilité gérer les entités disponibles aux *components*. Ces services permettent de modifier, créer, supprimer, mettre à jour ou récupérer des entités. Ils utilisent principalement le paquetage *CommunicationInterface* afin d'accéder au Rest.

Entities
Chaque classe contenue dans ce paquetage représente un objet du domaine de l'application. Ces objets permettent de partager l'information uniformément à travers les différentes couches de l'application.

La figure suivante montre le diagramme de paquetage du client serveur. Un tableau indique la responsabilité de tous les paquets présentés dans la figure.

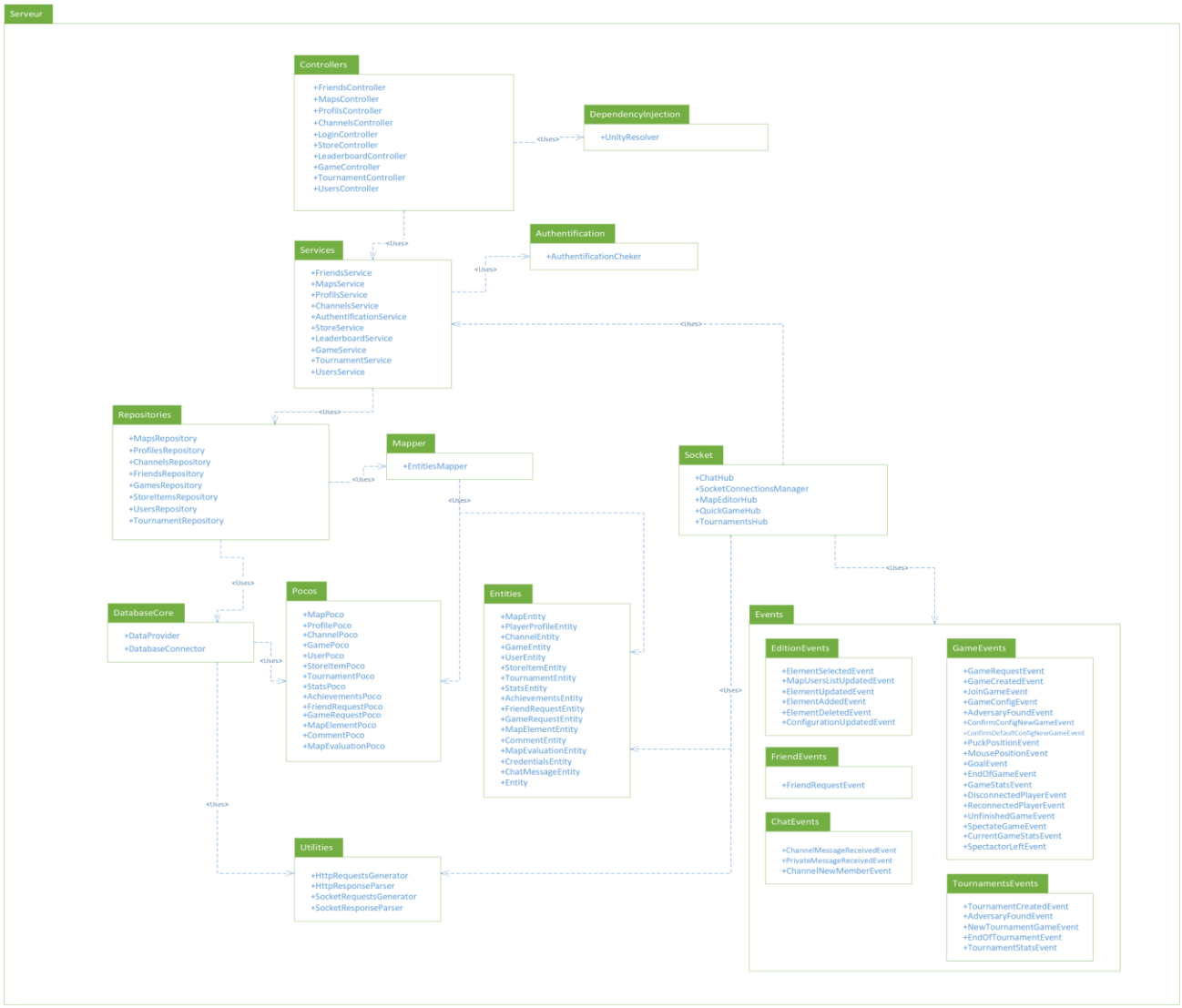

Figure 16 : Diagramme de paquetage du serveur

Contrôleurs
Ce paquetage est le point d'entrée du Rest Api. Il contient tous les contrôleurs de toutes les requêtes HTTP. Selon la requête captée par le contrôleur, la méthode du service du contrôleur en question sera exécutée.

Services
Ce paquetage contient tous les services qui implémentent la logique du côté serveur. Certaines requêtes des clients nécessitent une synchronisation avec la base de données, des évènements à lancer ou encore de mettre à jour les connections *Socket*. Les services permettent ainsi de coordonner les différentes actions.

Authentification
Ce paquetage contient les classes qui implémentent la vérification de l'authentification d'un utilisateur lorsque celui-ci essaie d'exécuter des actions restreintes aux utilisateurs connectés.

Repositories
Ce paquetage introduit une couche supplémentaire au-dessus de la base de données afin d'abstraire son utilisation. Ainsi, chaque classe de ce paquetage définit comment structurer de façon persistante chaque type d'entité utilisée dans l'application (en particulier, chaque classe définit les opérations de création, modification, obtention et suppression pour chaque type d'entité) et utilise le paquetage *Mapper* pour passer des *entities* aux *pocos* (et vice-versa).

Socket
Le paquetage *Socket* contient tous les *Hubs*. Chaque classe implémente un différent point d'entrée au serveur auquel les clients peuvent se connecter en utilisant la communication via sockets.

DatabaseCore
Ce paquetage contient toutes les classes utilitaires nécessaire pour accéder à la base de données et gérer adéquatement la connexion.

Pocos
La base de données gère un ensemble d'objets. Ainsi, chaque objet stocké dans la base de données peut être représenté par une classe. Le paquetage *Pocos* contient toutes les classes représentant les objets tels qu'ils sont présentés dans la base de données.

Entities
Les entités constituent les objets manipulés dans le domaine de l'application. Ce sont ainsi objets utilisés dans la logique de l'application (que ce soit côté serveur ou client) et qui sont échangés entre le serveur et les clients.

Mapper
Sachant que les objets présents dans la base de données ne sont pas nécessairement utilisés tel quel dans l'application et que réciproquement, les entités présentes dans l'application peuvent être construites à partir d'objets présents dans la base de données, il est nécessaire qu'un paquetage fasse le lien entre les deux. C'est le but du paquetage *Mapper* qui fait la transformation d'*entities* vers *pocos* et vice-versa.

Utilities
Ce paquetage contient des classes utilitaires relatives à la communication par HTTP ou par sockets. Ces classes permettent en particulier de faciliter la sérialisation et désérialisation des requêtes et réponses.

Events
Les différentes classes représentent les évènements qui peuvent être lancés par le serveur et capté par le client. Le client peut donc réagir selon le type d'événement capté.

EditionEvents
Ce paquetage représente les différents événements reliés à l'édition en ligne d'une zone de jeu tels que la translation, la rotation ou encore la suppression d'un objet par un utilisateur distant.

GameEvents
Ce paquetage représente les différents événements reliés à une partie jouée en ligne tels que la position de la rondelle, la position de l'adversaire ou encore la fin d'une partie. C'est par le biais de ces évènements que la gestion d'une partie entre le serveur et les différents joueurs s'effectue.

ChatEvents
Ce paquetage représente les différents événements reliés au chat. Un évènement sera envoyé lors de la réception d'un message dans un canal ou de la part d'un joueur spécifique et lorsqu'un joueur est ajouté à un canal.

FriendEvents
Ce paquetage représente les événements reliés aux amis de l'utilisateur. Pour l'instant le seul événement auquel le client peut s'abonner est la réception d'une demande d'amis.

TournamentEvent
Ce paquetage représente les différents événements reliés aux tournois en ligne tels que le démarrage d'une nouvelle ronde du tournoi ou encore la fin de ce dernier. C'est par le biais de ces évènements que la gestion d'un tournoi entre le serveur et les différents joueurs s'effectue.

5. Vue des processus

La vue des processus est représentée sous forme de diagrammes de séquence. Les diagrammes de séquence permettent de voir les interactions entre l'utilisateur et les objets du système en considérant les cas d'utilisation. Ils ajoutent une dimension temporelle par rapport aux diagrammes de cas d'utilisation. La présente section décrit trois diagrammes de séquence principaux. Afin d'améliorer la visibilité des diagrammes, un document en pièce jointe nommé ***4.2-Diagrammes_sequences-LNAH_2K17-Les_Decales.pdf*** est fourni.

La figure suivante montre la séquence où un utilisateur démarre une partie rapide en ligne et configure la partie. Lorsque l'utilisateur désire lancer une partie rapide, s'il n'est pas jumelé directement à un autre joueur, il se retrouve dans une salle d'attente. Lorsque le joueur qui lance une partie rapide est jumelé à un autre joueur, ceux-ci sont redirigés vers un menu de sélection de la carte. Les deux joueurs ont alors 30 secondes pour choisir la même carte, faute de quoi une carte par défaut sera sélectionnée.

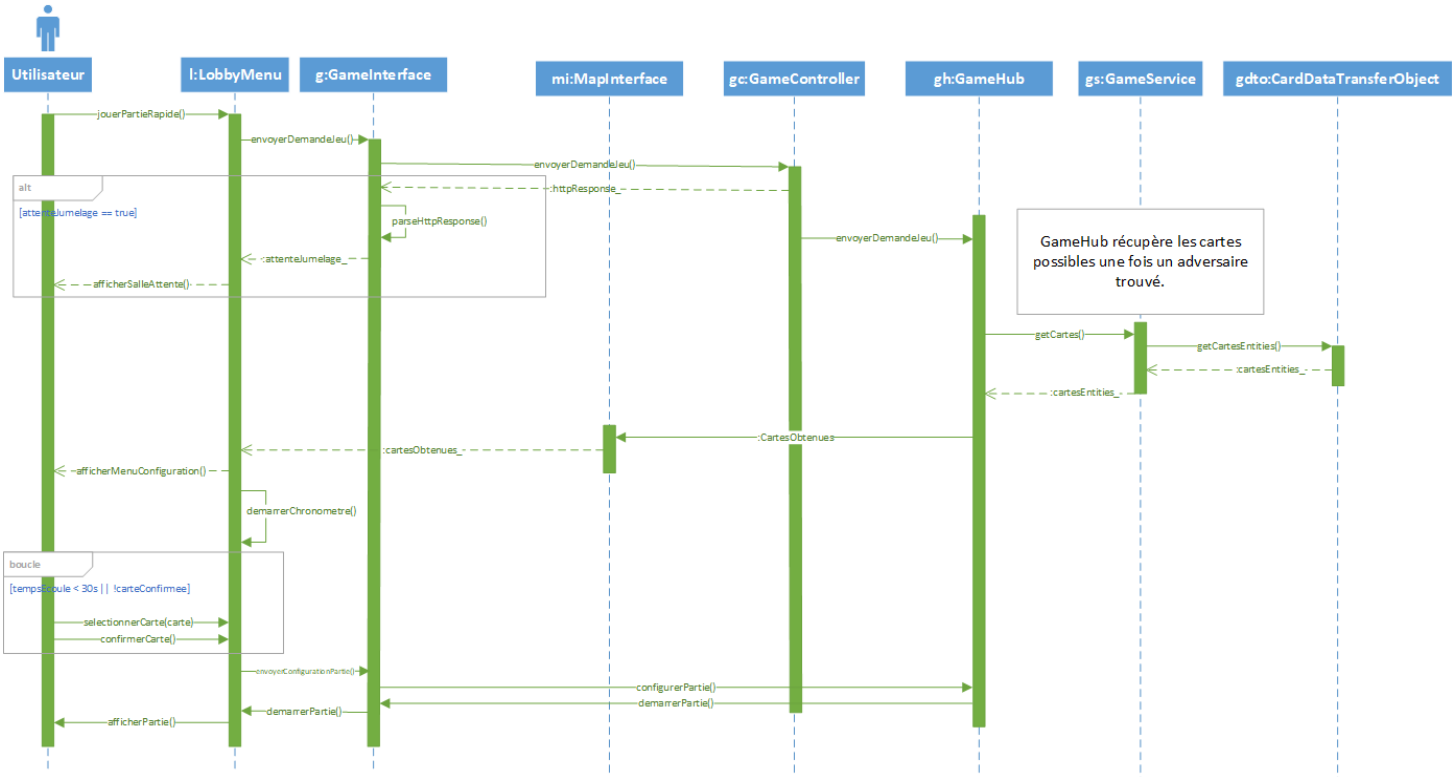

Figure 17: Diagramme de séquence du démarrage d'une partie rapide en ligne et de la configuration de la partie

La figure suivante montre la séquence où un utilisateur crée une carte. Si le nom de la carte entrée par l'utilisateur existe déjà, une erreur est lancée, sinon le menu de création de la carte est fermé et le mode édition lancé.

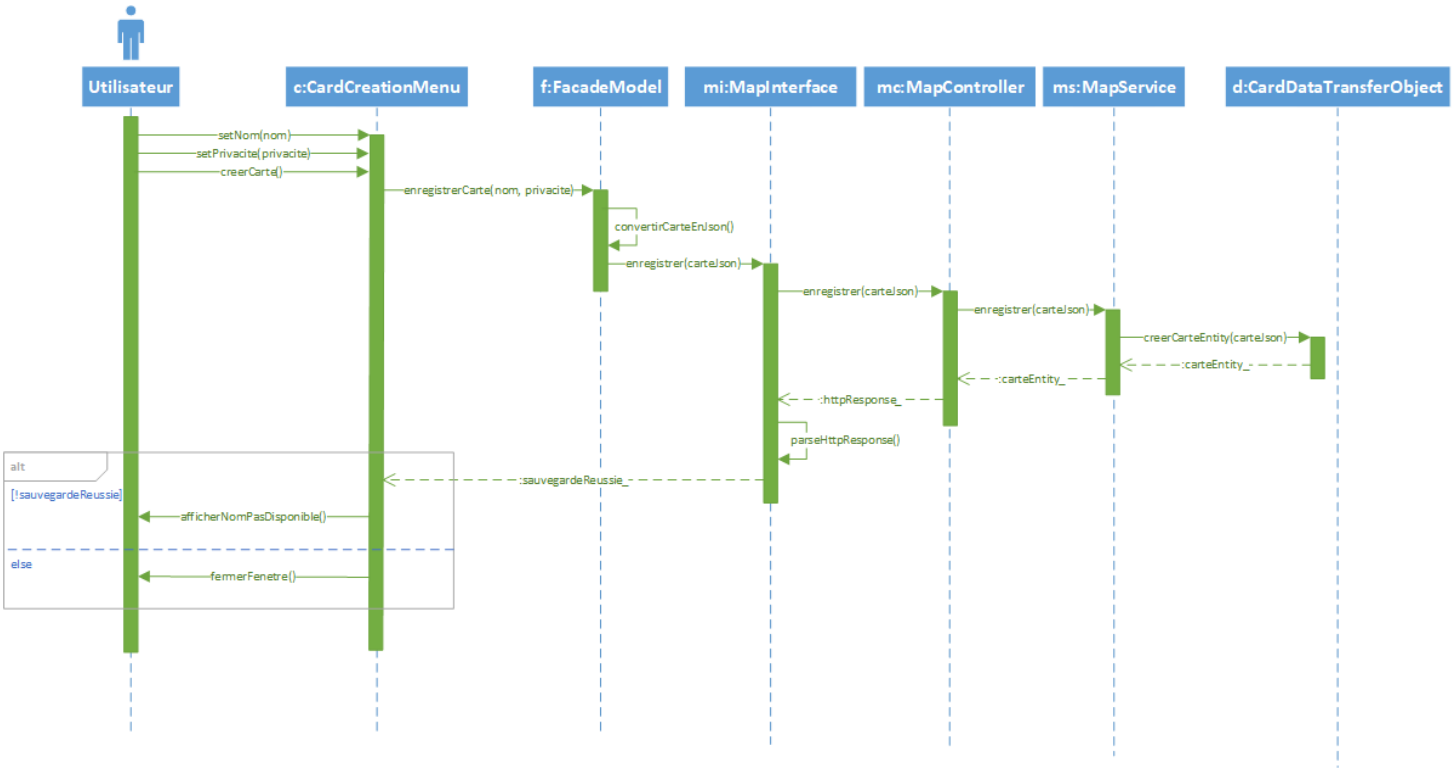

Figure 18 : Diagramme de séquence de la création d'une carte

La figure suivante montre la séquence où un utilisateur crée un canal de discussion et y envoie un message. L'utilisateur crée un nouveau canal en entrant le nom. Si le nom du canal existe déjà, une erreur est lancée, sinon le canal est créé. L'utilisateur peut ensuite envoyer des messages à tous les membres du canal.

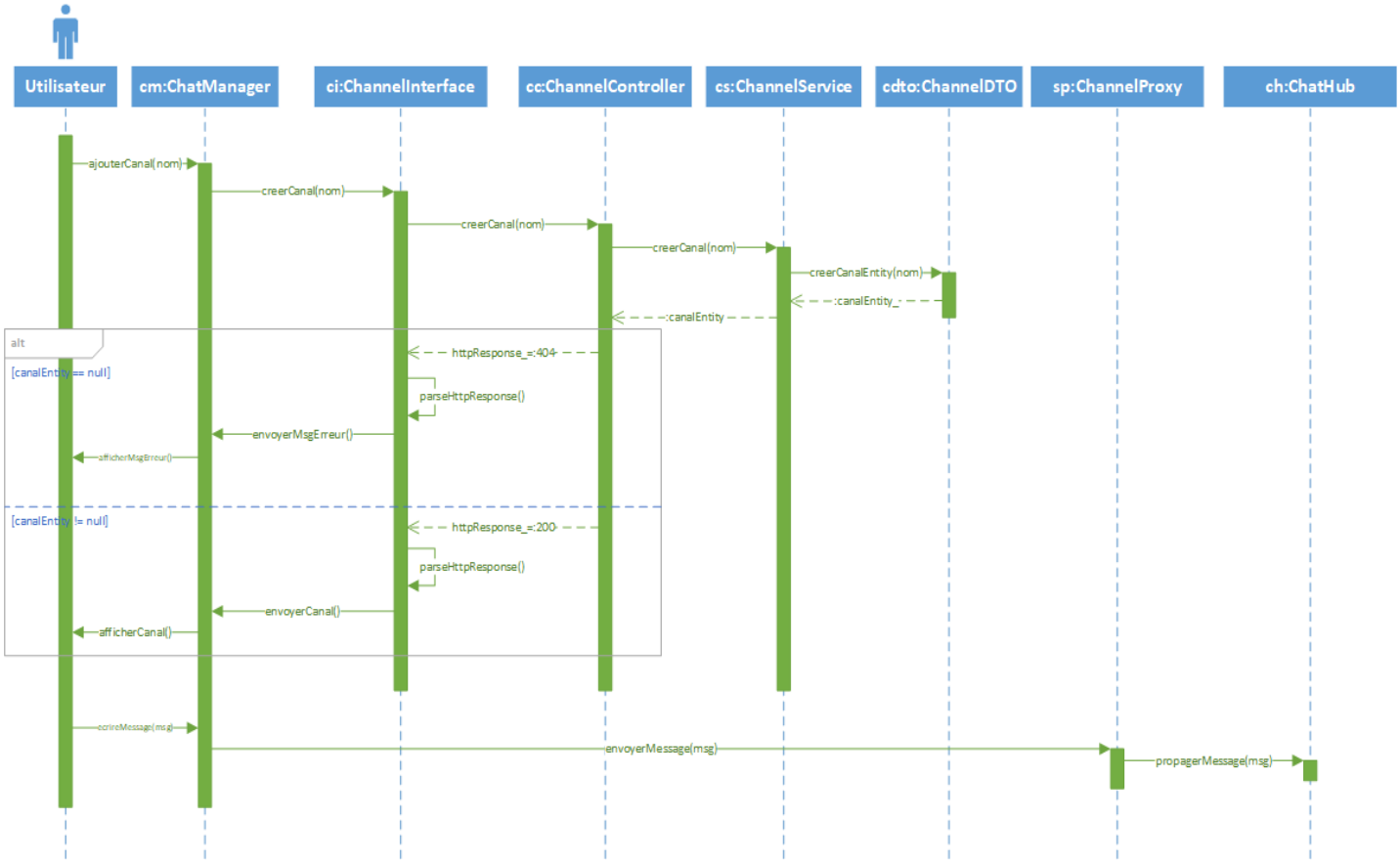

Figure 19 : Diagramme de séquence de la création d'un canal de chat et de l'envoi d'un message

6. Vue de déploiement

Le système contient quatre composantes principales, soit le client lourd, le client léger, le client web et le serveur. Le client lourd et le serveur sont déployés sur le système d'exploitation Windows 10, tandis que le client léger est déployé sur iOS 10.3. Le serveur contient l'application serveur pour le jeu ainsi qu'une application web.

La communication entre les clients lourds et le serveur est fait par l'entremise de *sockets* (TCP/IP) ainsi que par des requêtes HTTP(TCP/IP) au travers du réseau filaire de l'École Polytechnique de Montréal. Le client léger communique avec le serveur de la même façon que le client lourd, mais au travers du réseau sans fil sécurisé d'*eduroam*. Le client web communique avec le serveur uniquement à l'aide de requête HTTP. Il en est de même pour la communication entre le serveur et la base de données.

La figure suivante montre le déploiement du système.

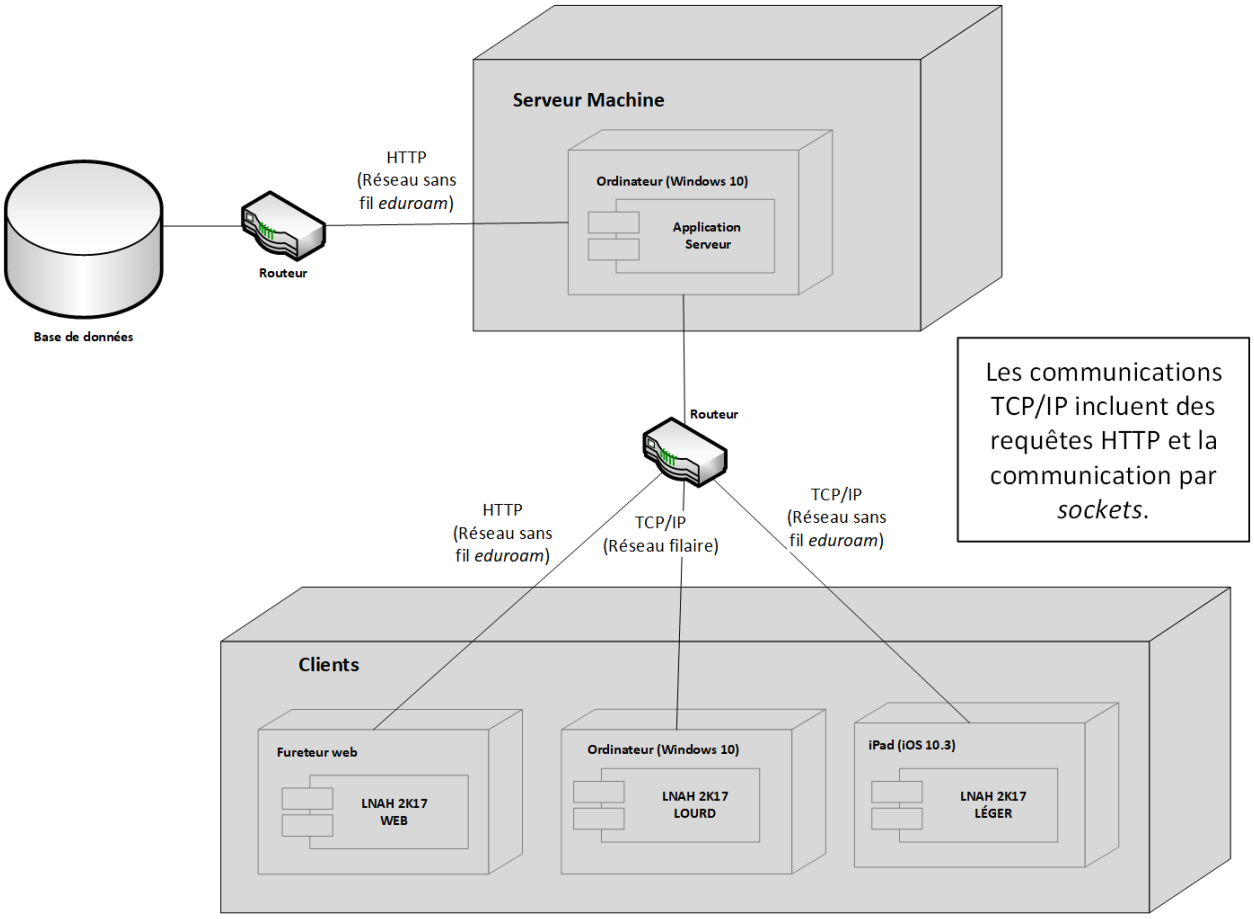

Figure 20 : Diagramme de déploiement du système

7. Taille et performance

La taille maximale théorique d'un paquet pouvant être envoyé par TCP est de 1.5 ko. Dans le cas où un message de plus 1.5 ko doit être envoyé, il sera décomposé en plusieurs paquets.

Afin de réduire les délais maximaux, des sockets sont utilisés. Ils permettent de réduire significativement les en-têtes HTTP requises. Les en-têtes ne devront donc pas être retransmises pour l'envoi de chaque message. Cette technologie permettra donc de respecter le délai maximal de 200 millisecondes que nous nous sommes imposés.

La taille du projet tournera autour de 100 Mo pour le client lourd, 60 Mo pour le client léger et 100 à 200 Mo pour le serveur. Au vu de cette taille très raisonnable, aucune architecture de déploiement plus complexe que celle précédemment montrée n'est requise.