Équipe 107

DrawHub

Document d'architecture logicielle

Version 2.0

Historique des révisions

Date	Version	Description	Auteur
2020-10-02	1.0	Première proposition d'architecture	Bruno Curzi-Laliberté William Glazer-Cavanagh Thomas Houtart Antoine Martin Simon Robatto-Simard Vincent Tessier
2020-11-30	2.0	Architecture finale	Bruno Curzi-Laliberté William Glazer-Cavanagh Thomas Houtart Antoine Martin Simon Robatto-Simard Vincent Tessier

Table des matières

Document d'architecture logicielle

1. Introduction

Le document d’architecture logicielle est divisé en 6 parties distinctes. La première partie consiste à détailler les objectifs et les contraintes qui auront un impact sur l’architecture de notre logiciel. La section qui suit est la vue des cas d’utilisations. Par la suite il y a, la vue logique contenant les diagrammes de classes et paquetages, la vue des processus puis la vue de déploiement. Finalement la section taille et performance qui décrit les caractéristiques qui peuvent affecter l’architecture et le design de notre logiciel au niveau de ses performances.

2. Objectifs et contraintes architecturaux

2.1. Objectifs

L’architecture du logiciel DrawHub a pour objectif d’être en mesure d’assurer aux utilisateurs une expérience de jeu fluide et agréable. De plus, l’architecture de notre logiciel doit assurer la sécurité et la confidentialité des données des utilisateurs.

2.2 Contraintes

L’architecture de notre logiciel doit permettre à celui-ci d’être fonctionnel sur le client lourd avec Windows 10 ainsi que sur le client léger avec tablette Galaxy Tab A 2019 sous Android 9. Aucun coût monétaire ne doit être engendré par l’utilisation de code, de librairies, de serveur ou de logiciels développés par une source externe. Pour le client lourd, il sera développé en HTML/CSS avec Angular 10. Le client léger sera quant à lui développé en XML et Kotlin. Le client léger aura une architecture MVVM pour entre autres permettre une meilleure maintenabilité et réutilisabilité du code.

3. Vue des cas d’utilisation

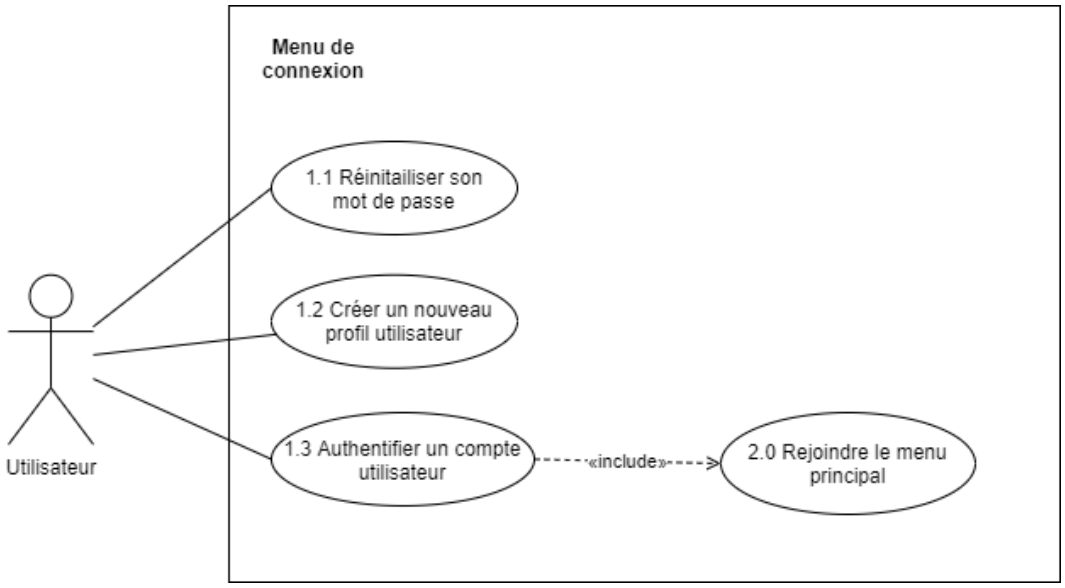

Figure 3.1. Diagramme des cas d’utilisation d’une connexion à DrawHub.

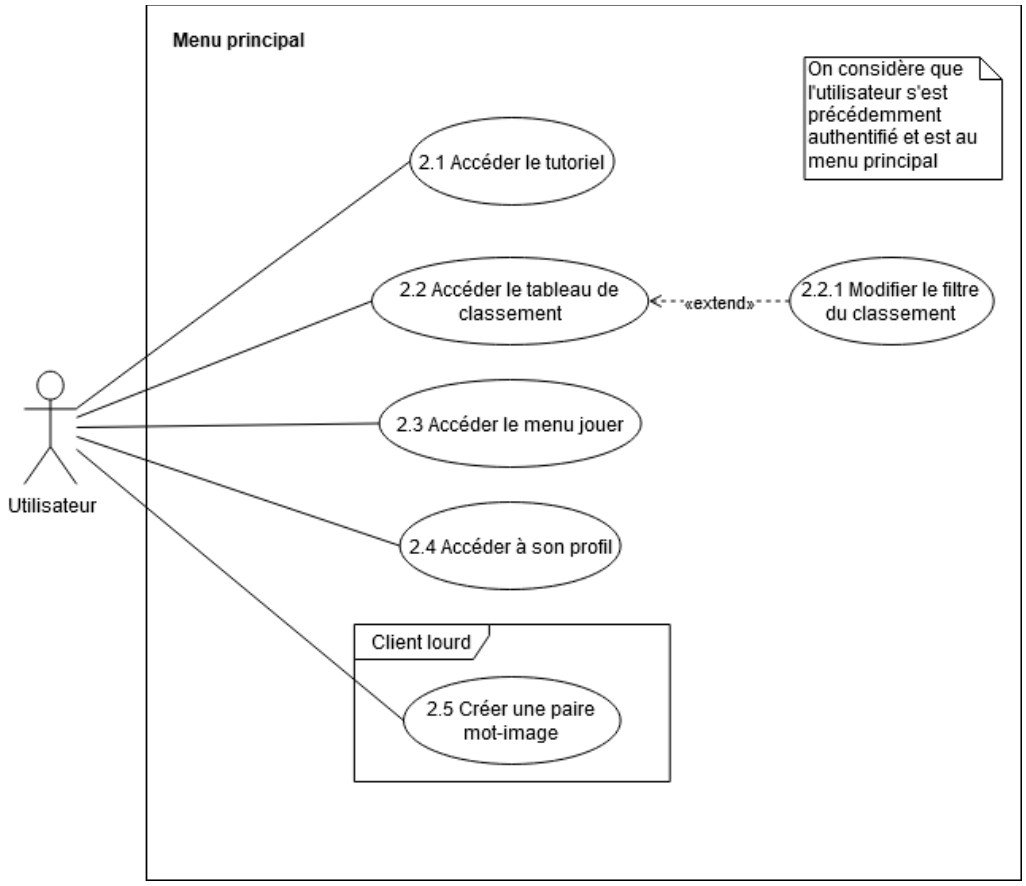

Figure 3.2. Diagramme des cas d’utilisation de la navigation à partir du menu principal.

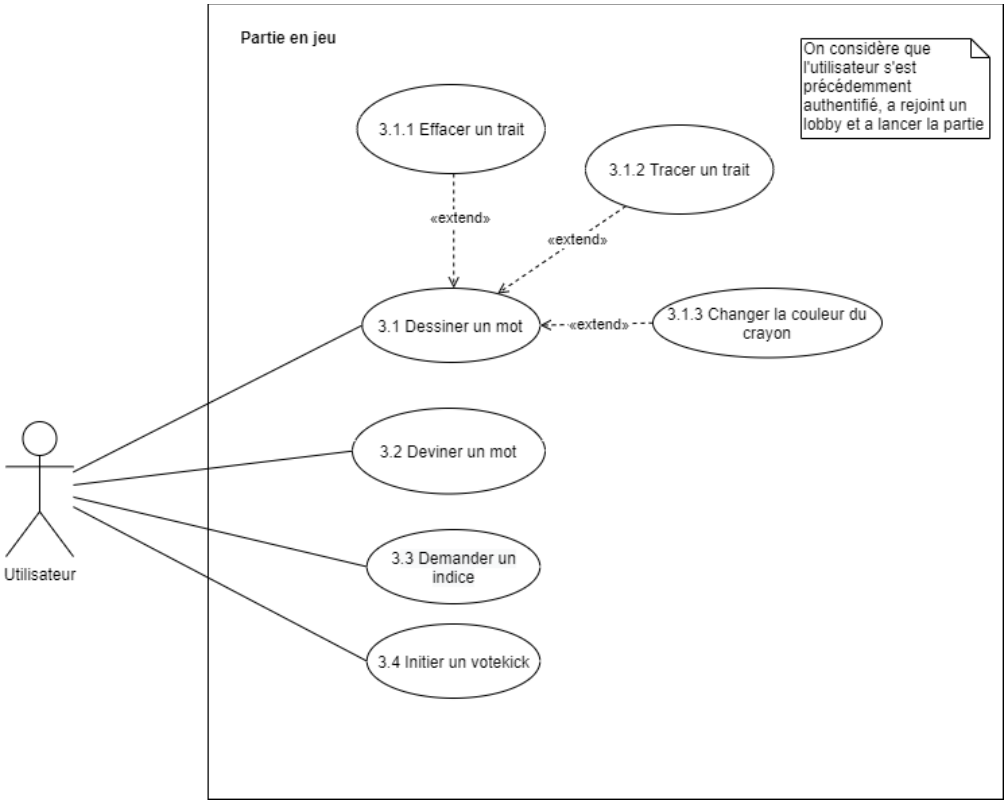

Figure 3.3. Diagramme des cas d'utilisation d'un utilisateur dans une partie.

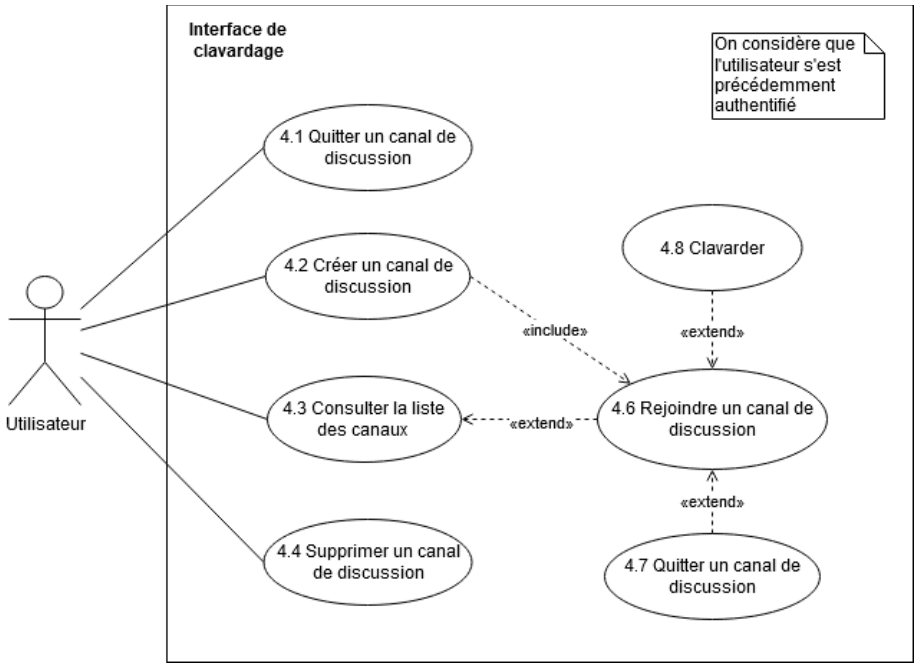

Figure 3.4. Diagramme des cas d'utilisation d'un utilisateur dans l'interface de clavardage.

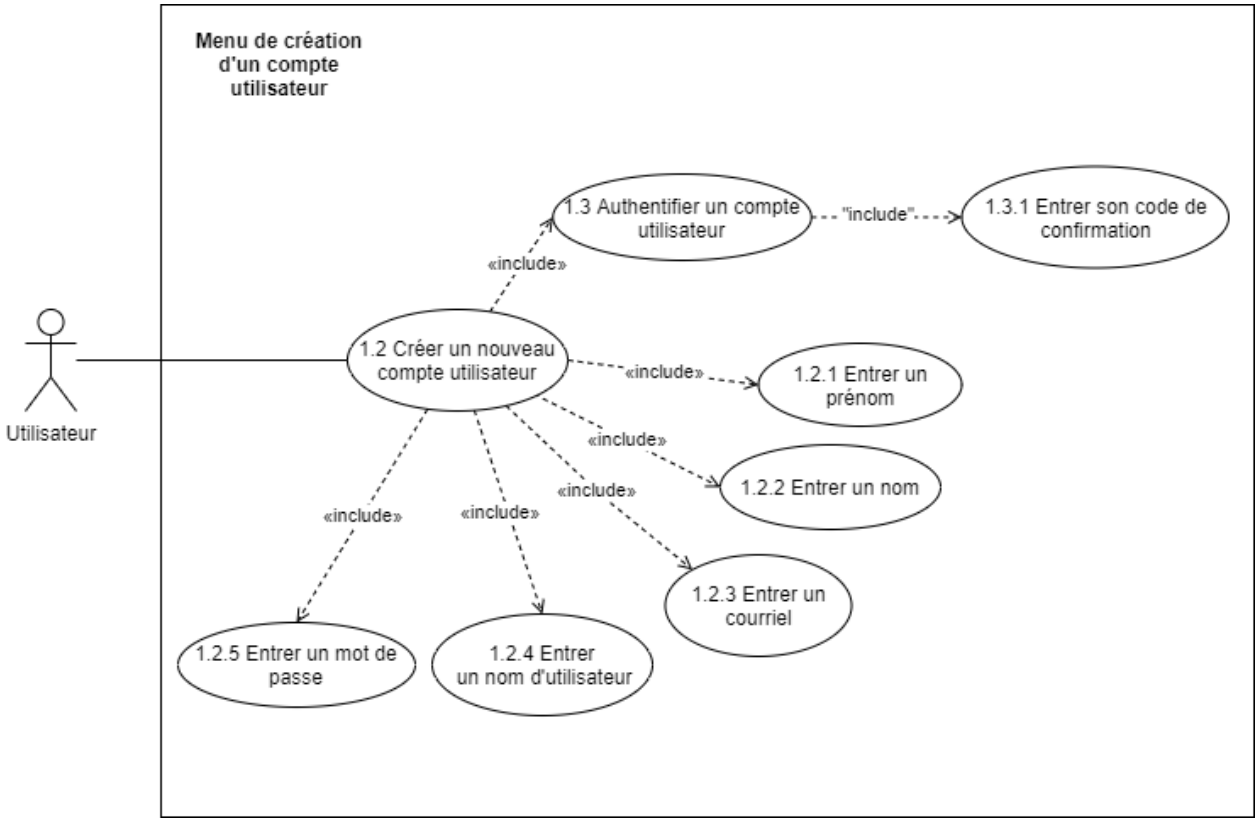

Figure 3.5. Diagramme des cas d'utilisation de la création d'un nouveau compte utilisateur.

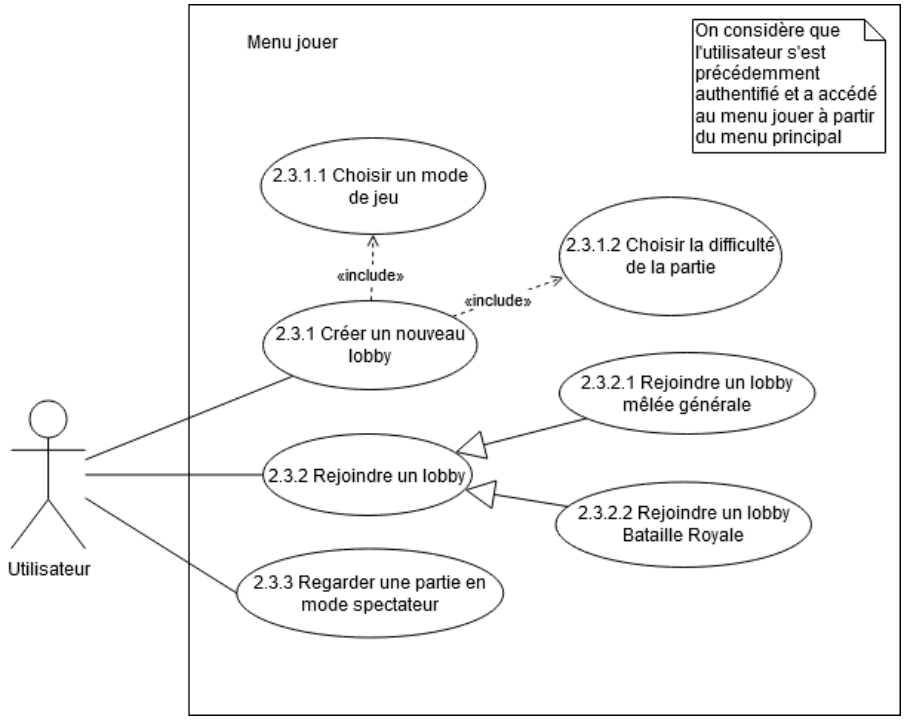

Figure 3.6. Diagramme des cas d'utilisation d'un utilisateur dans le menu jouer.

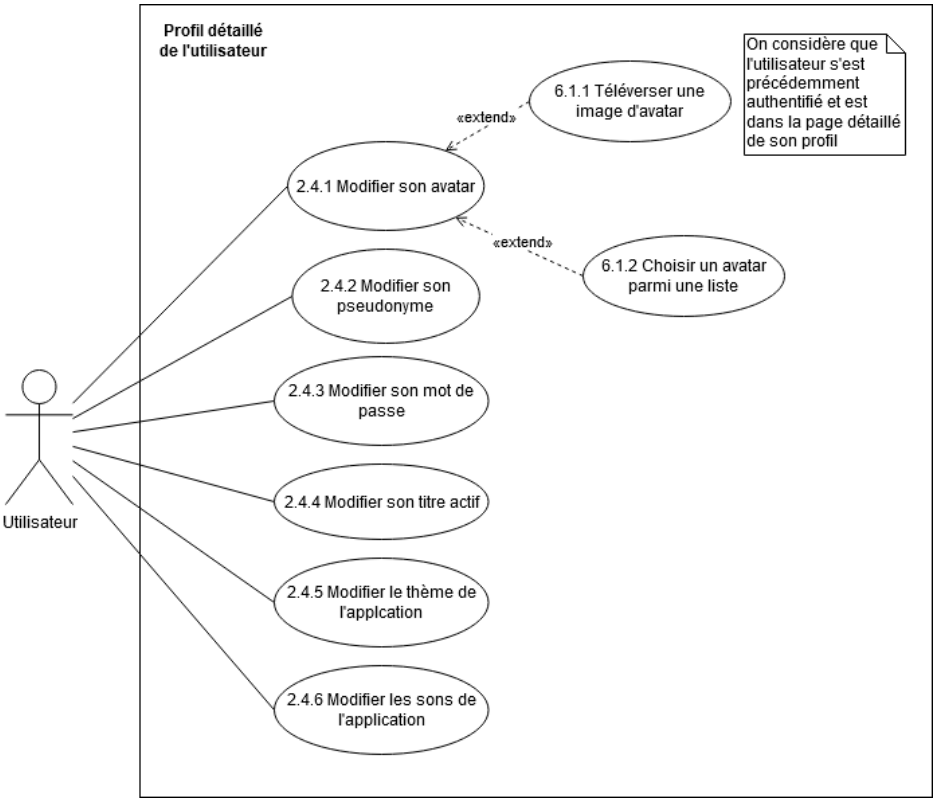

Figure 3.7. Diagramme des cas d'utilisation d'un profil utilisateur détaillé.

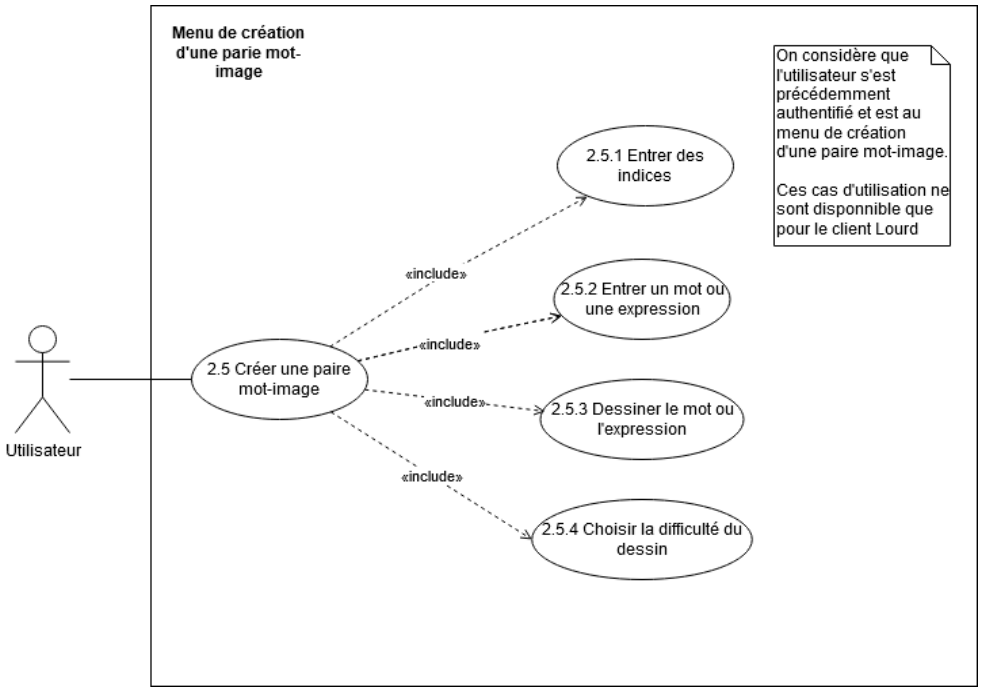

Figure 3.8. Diagramme des cas d'utilisation de la création d'une paire mot-image.

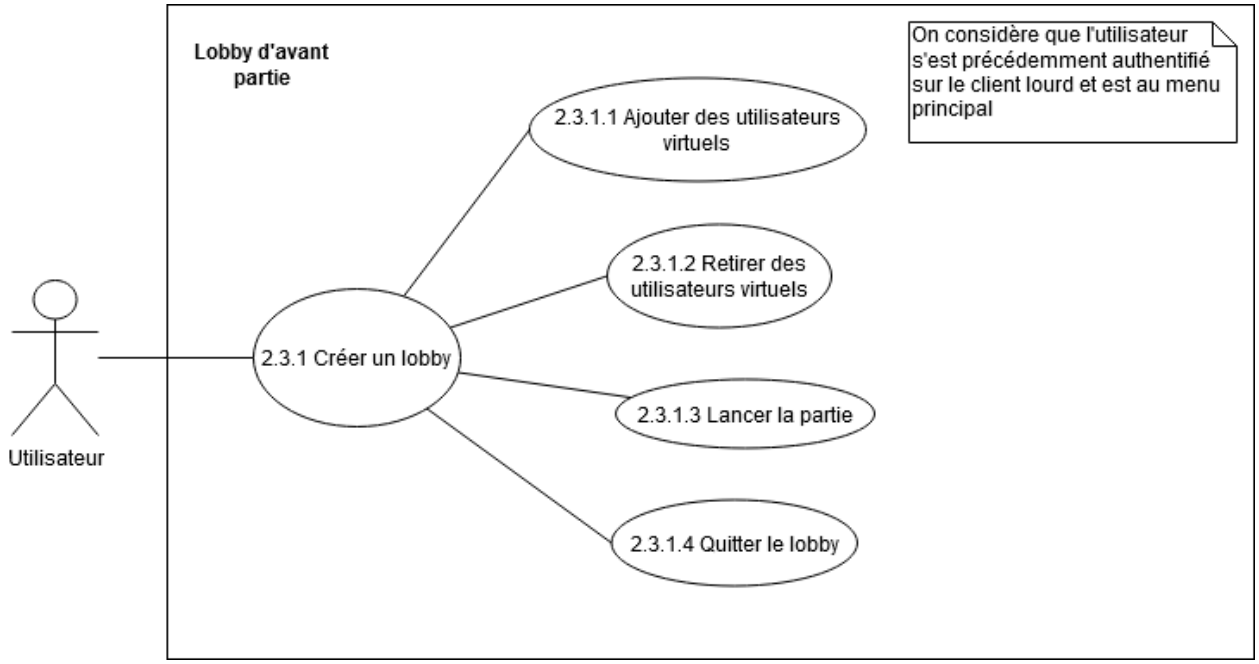

Figure 3.9. Diagramme des cas d'utilisation d'un utilisateur dans un lobby.

4. Vue logique

Cette section présente des diagrammes de paquetages et de classes des différents paquetages envisagés pour bâtir l'architecture multi niveau du logiciel *DrawHub.*

4.1. Client léger

Afin de mieux comprendre la suite de l'architecture du client léger, la prochaine figure montre une vue haut niveau des modules composant le client léger sur Android Studio. Chaque figure est suivie d'un tableau explicatif décrivant les paquetages en détail.

4.1.1. Haut niveau

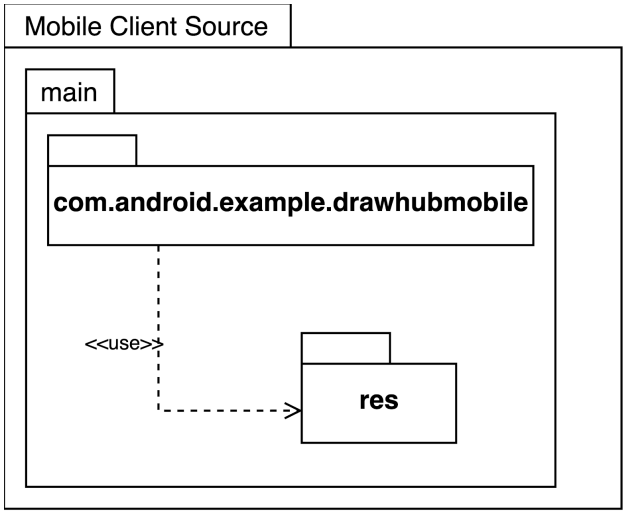

Figure 4.1. Diagramme de paquetage haut niveau pour le client léger.

com.android.example.DrawHubmobile
Ce paquetage contient tout le code écrit en Kotlin et utilise res, qui est spécifié à la figure 4.3.

res
Ce paquetage contient toutes les ressources qui ne sont pas du code écrit en Kotlin et est spécifié à la figure 4.2.

4.1.2. Ressources

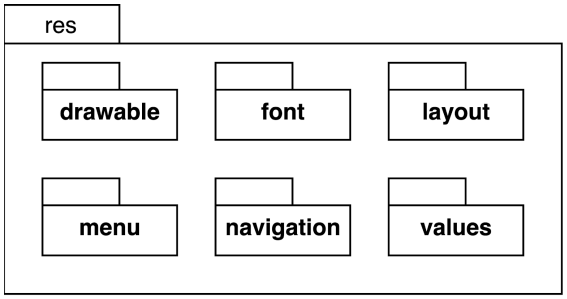

Figure 4.2. Diagramme de paquetage pour le paquetage "*res*".

drawable
Ce paquetage contient toutes les ressources qui peuvent être dessinées. Notamment il contient les icônes, les images et les Logos de l'application.

font
Ce paquetage contient les fichiers ".ttf" représentant les diverses polices utilisées par l'application.

layout
Ce paquetage contient tous les fichiers XML définissant la structure des vues de l'application. Il englobe les "Fragments" et les "Activities" de l'application Android.

menu
Ce paquetage contient les fichiers XML permettant de définir les divers menus accessibles par les barres de l'application.

navigation
Ce paquetage contient le fichier XML qui définit comment les diverses vues interagissent ensemble.

values
Ce paquetage contient les fichiers XML des diverses valeurs constantes partagées par les composantes de l'application. Plus spécifiquement, il contient les définitions pour les couleurs, les chaînes de caractères, les dimensions et les divers styles graphiques.

4.1.3. Application

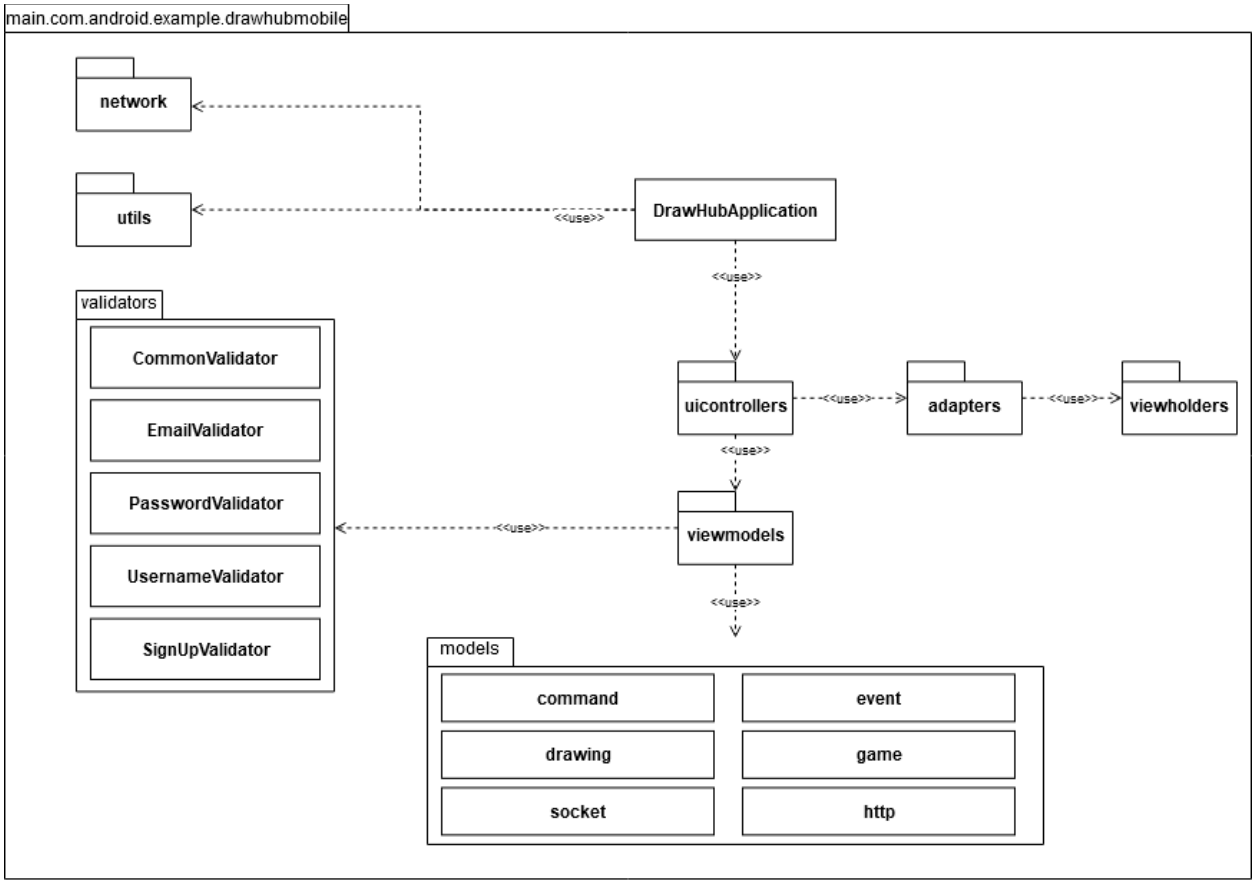

Figure 4.3. Diagramme de paquetage pour le paquetage "*com.android.example.DrawHubmobile*".

network
Ce paquetage contient toutes les classes dont la responsabilité est de communiquer avec le serveur. Son contenu est explicité dans le diagramme de classes à la figure 12.

utils
Ce paquetage contient des classes d'outils générales. Plus spécifiquement on y trouve des classes faisant la conversion de la date reçue par le serveur à un objet de type "LocaleDateTime". Puis, une classe qui convertit les traits de dessin du client en format compréhensible par le serveur. Il est à noter que ce paquetage peut grandir pour accueillir davantage d'outils.

validators
Ce paquetage contient toutes les classes responsables de valider le contenu entré dans des boîtes de texte par l'utilisateur. Chaque classe est responsable d'un seul type de validation.

models
Ce paquetage contient les définitions des modèles utilisés dans l'application. Plus spécifiquement, il contient des classes majoritairement de type "data class" qui servent à localiser les données partagées dans l'application. Aussi, on y trouve des énumérations et des constantes communes au contexte global.

viewholders
Ce paquetage contient les contrôleurs pour les vues utilisées par le RecyclerViewAdapter.

adapters
Ce paquetage contient les sous-classes de RecyclerViewAdapter qui seront utilisées par RecyclerView afin d'organiser son contenu. Il utilise les classes dans le paquetage "*viewholders*".

uicontrollers
Ce paquetage contient les contrôleurs des diverses vues de l'application. Les classes dans ce paquetage sont responsables de capter les événements utilisateurs et les envoyer à leur ViewModel respectif. Le paquetage est explicité à la figure 13. Ce paquetage est hautement couplé avec le paquetage "*viewmodels*" car chaque contrôleur possède une instance de son ViewModel Les classes représentant des vues qui contiennent des RecyclerView

utilisent les classes dans le paquetage “*adapters*”.

viewmodels
Ce paquetage contient les sous-classes de ViewModel dont la responsabilité est de traiter les données obtenues par les contrôleurs. Plus spécifiquement, les classes dans ce paquetage s’occupent de la communication avec le serveur, de la traduction des données externes ou internes et de la validation des évènements. Ce paquetage est exploré en détail à la figure 14.

DrawHubApplication
Cette classe représente l’application et contient le contexte de celle-ci. Plus spécifiquement, cette classe est un Singleton qui est sous-classe de “Application”. Elle contient les informations sur l’utilisateur courant, le “Socket” de Socket.IO, l’instanciation de ServerApi et la liste des émetteurs de messages du serveur.

4.1.4. Émetteurs d'évènements

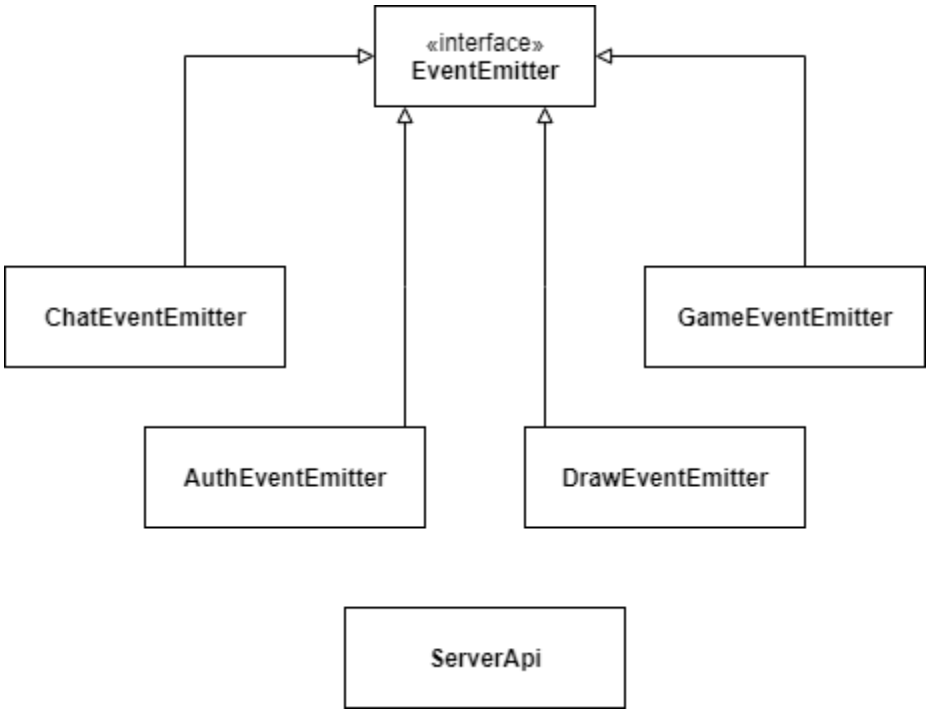

Figure 4.4. Diagramme de classes pour les classes pour les *EventEmitters.*

EventEmitter
Cette classe est responsable d'émettre les évènements vers le serveur. Elle est utilisée par les 4 emitter spécifiques (ChatEventEmitter, GameEventEmitter, AuthEventEmitter, DrawEventEmitter)

ChatEventEmitter
Cette classe est responsable de communiquer avec EventEmitter pour les évènements liés au chat.

AuthEventEmitter
Cette classe est responsable de communiquer avec EventEmitter pour les évènements liés à l'authentification.

DrawEventEmitter
Cette classe est responsable de communiquer avec EventEmitter pour les évènements liés à la logique du dessin.

GameEventEmitter
Cette classe est responsable de communiquer avec EventEmitter pour les évènements liés à une partie.

ServerApi
Cette classe est responsable de la communication https avec le serveur.

4.1.5. Contrôleurs d'interfaces utilisateur

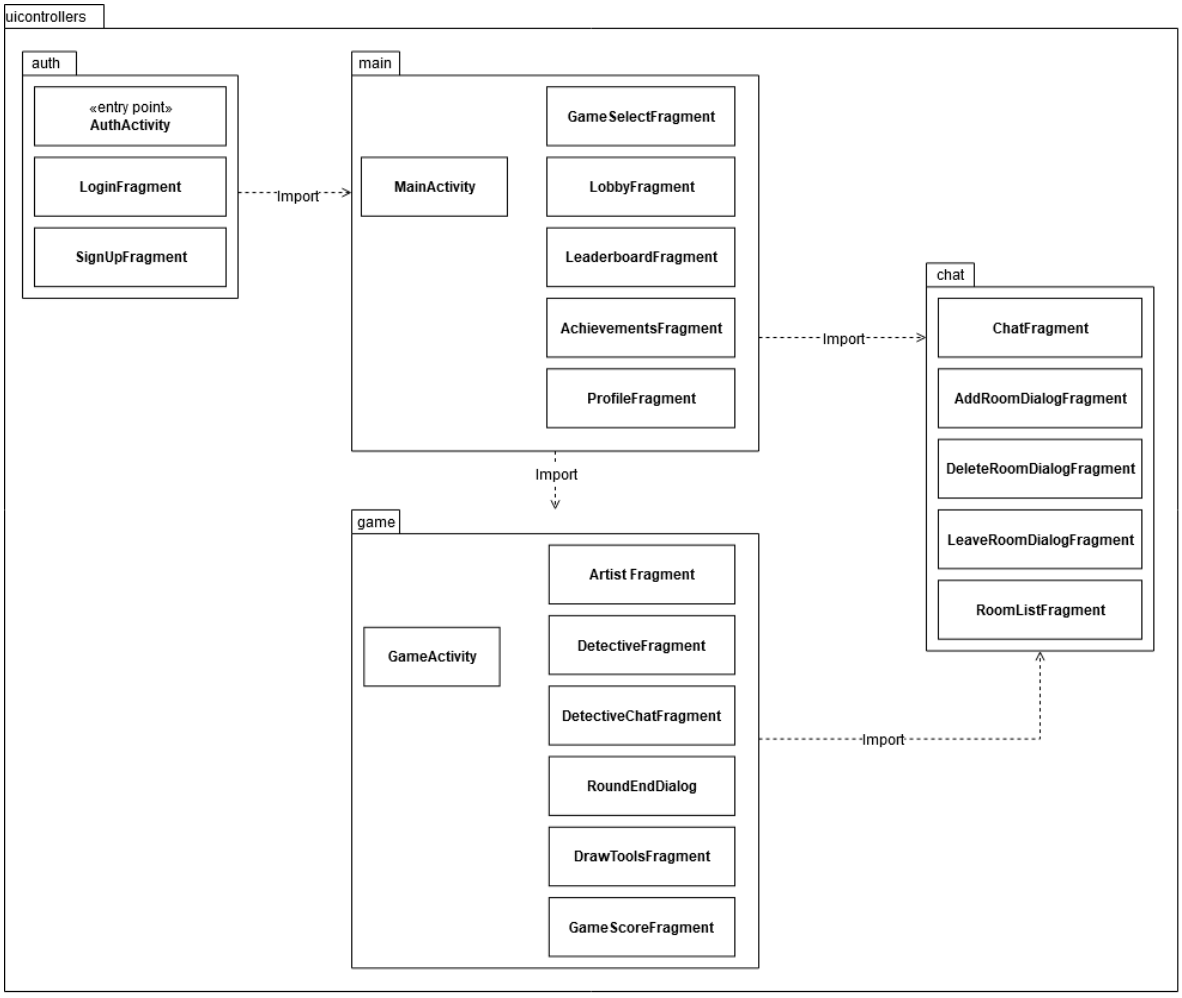

Figure 4.5. Diagramme de paquetage pour le paquetage "*uicontrollers*".

auth
Ce paquetage contient les contrôleurs des vues du menu d'authentification. Les classes dans ce paquetage sont responsables de capter les événements utilisateurs et les envoyer à leur ViewModel respectif.

main
Ce paquetage contient les contrôleurs des vues du menu principal. Les classes dans ce paquetage sont responsables de capter les événements utilisateurs et les envoyer à leur ViewModel respectif.

game
Ce paquetage contient les contrôleurs des vues nécessaires durant une partie. Les classes dans ce paquetage sont responsables de capter les événements utilisateurs et les envoyer à leur ViewModel respectif.

chat
Ce paquetage contient les contrôleurs des vues de l'interface de chat. Les classes dans ce paquetage sont responsables de capter les événements utilisateurs et les envoyer à leur ViewModel respectif.

4.1.6. Modèle de vue

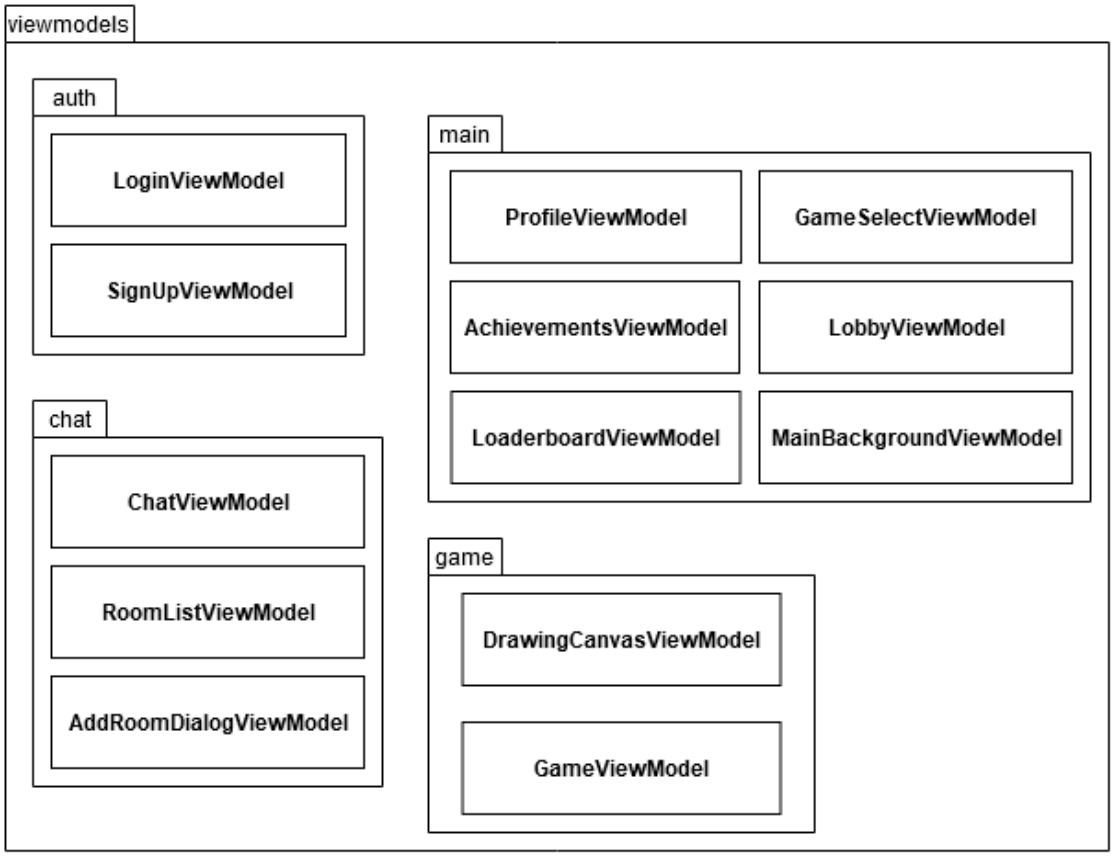

Figure 4.6. Diagramme de paquetage pour le paquetage “viewmodels”.

auth
Ce paquetage a la responsabilité de traiter les données obtenues par les contrôleurs de l’interface d’authentification. Les classes dans ce paquetage s’occupent de la communication avec le serveur, de la traduction des données externes ou internes et de la validation des évènements.

main
Ce paquetage a la responsabilité de traiter les données obtenues par les contrôleurs de l’interface du menu principal. Les classes dans ce paquetage s’occupent de la communication avec le serveur, de la traduction des données externes ou internes et de la validation des

évènements.

game
Ce paquetage a la responsabilité de traiter les données obtenues par les contrôleurs d'une partie en cours. Les classes dans ce paquetage s'occupent de la communication avec le serveur, de la traduction des données externes ou internes et de la validation des évènements.

chat
Ce paquetage a la responsabilité de traiter les données obtenues par les contrôleurs de l'interface de clavardage. Les classes dans ce paquetage s'occupent de la communication avec le serveur, de la traduction des données externes ou internes et de la validation des évènements.

4.2. Client lourd

Afin de mieux comprendre la suite de l'architecture du client lourd, la prochaine figure montre une vue haut niveau des modules composant le client lourd sur Angular. Chaque figure est suivie d'un tableau explicatif décrivant les paquetages en détail.

4.2.1. Application

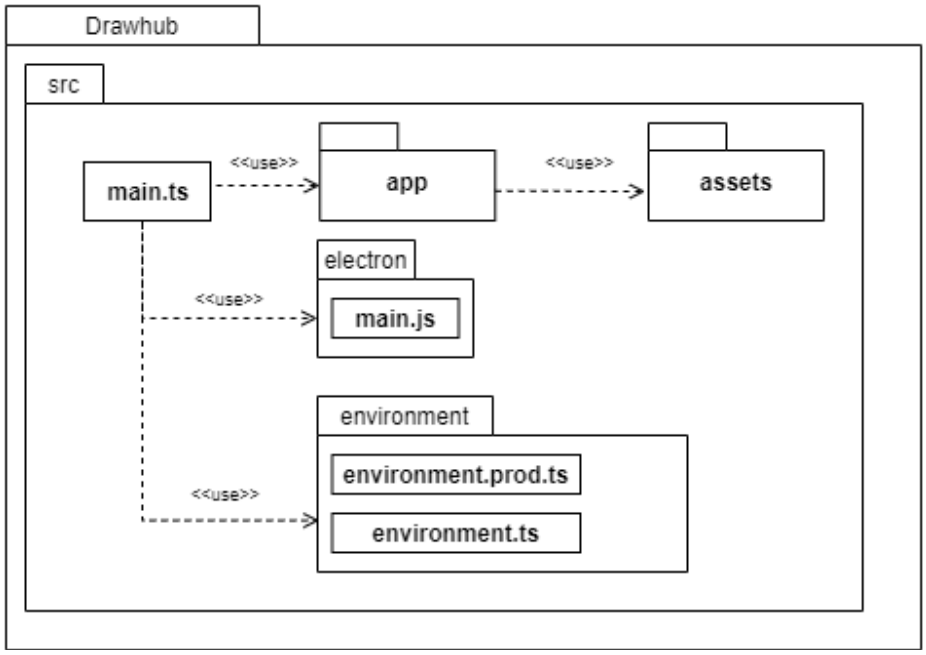

Figure 4.7. Diagramme de paquetage pour l'application.

Figure 15. Diagramme de paquetage du dossier de l'application **Electron**
Ce paquetage contient la logique d'exécution pour Electron.

environment
Ce paquetage contient l'environnement de l'application Angular.

app
Ce paquetage contient la logique de l'application Angular (Figure 16).

assets

Ce paquetage contient des ressources (Ex: images, données, etc.) utilisées par le paquetage app.

4.2.2. Architecture Angular

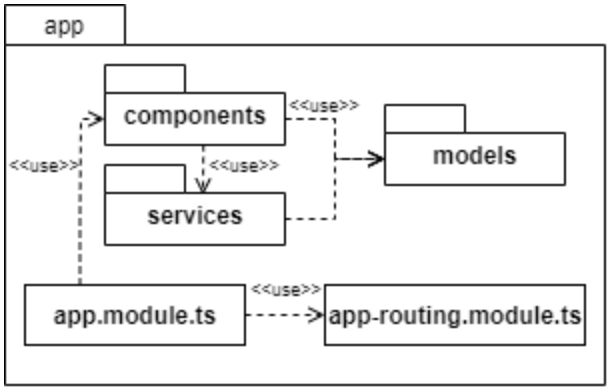

Figure 4.8. Diagramme de paquetage d'architecture Angular.

components
Ce paquetage contient l'ensemble des composants Angular de l'application (Figure 17).

models
Ce paquetage contient les interfaces, classes, constantes, enum et fonctions utilisés par le paquetage component et le paquetage service.

services
Ce paquetage contient l'ensemble des services utilisé par le paquetage component et le paquetage service (Figure 18).

4.2.3. Présentation des paquetages de composantes

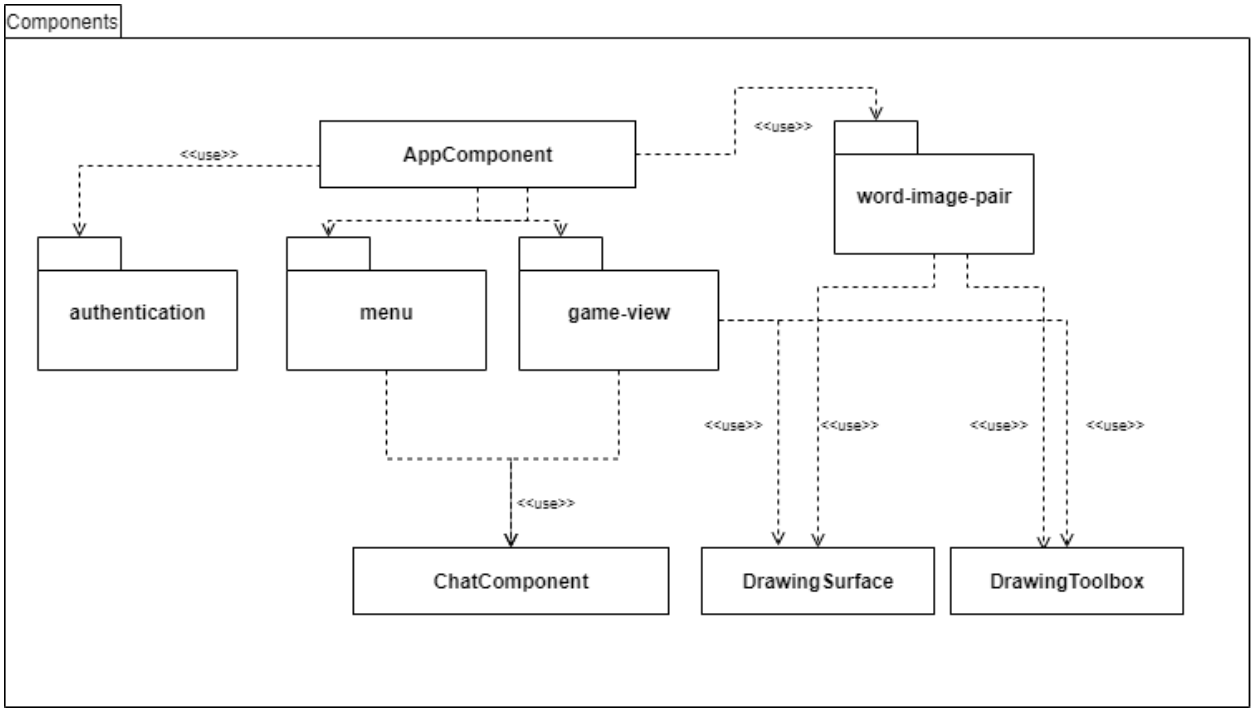

Figure 4.9. Diagramme de paquetage du paquetage components.

authentication
Ce paquetage contient les composantes Angular liés à l'authentification (Figure 18).

DrawingSurface
Cette classe est responsable de la surface de dessin.

DrawingToolbox
Cette classe est responsable des outils qui permettent de dessiner sur la surface de dessin.

word-image-pair
Ce paquetage contient les composantes Angular lié au word-image-pair (Figure 18)..

menu
Ce paquetage comprend les composantes lié au menu principal (Figure 18).

game-view
Ce paquetage comprenant les composantes lié à l'espace de jeu (Figure 18).

4.2.4. Diagramme de paquetage des composantes

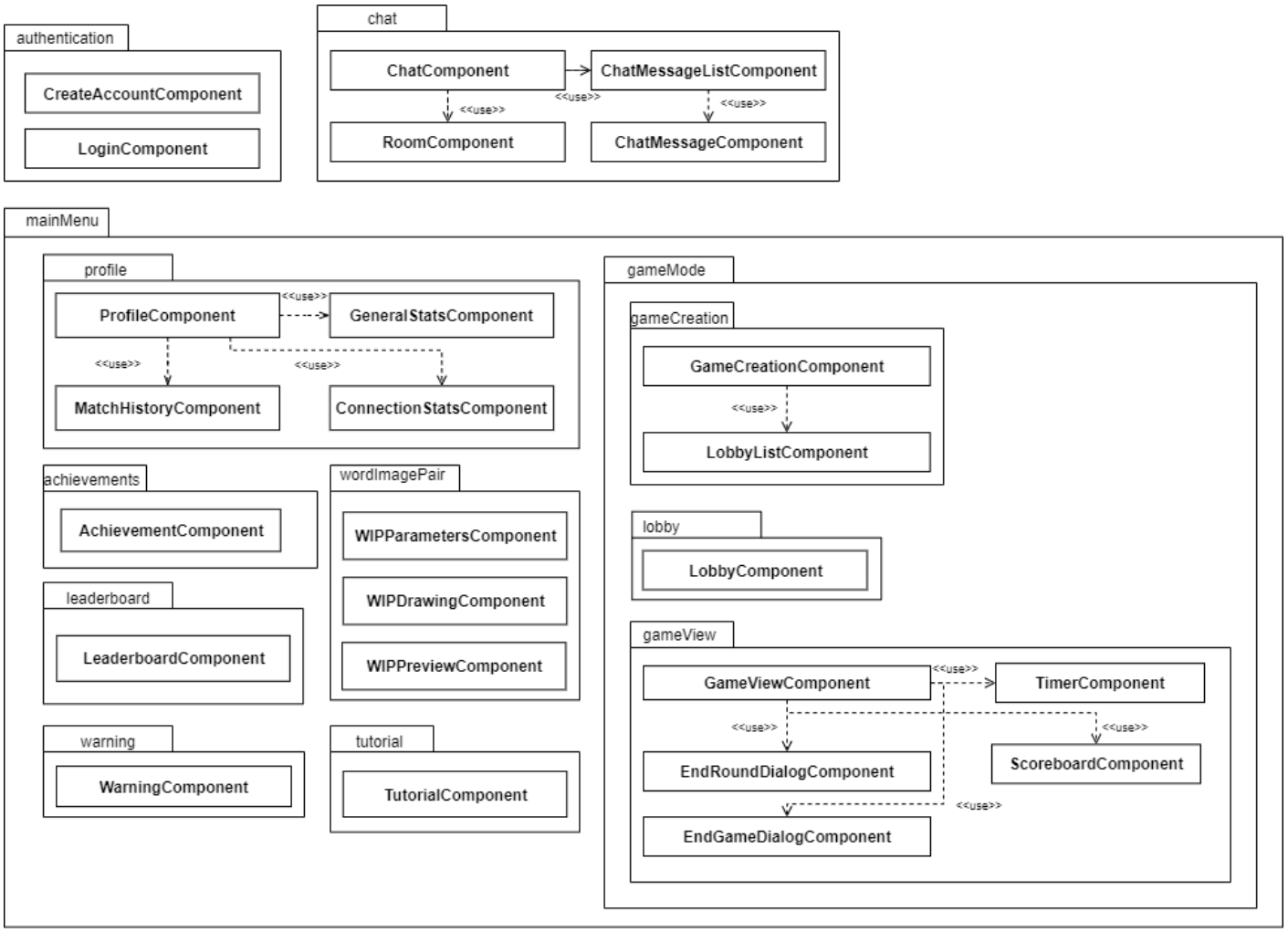

Figure 4.10. Diagramme de paquetage de l'application complète.

tutorial
Ce paquetage contient le composant Angular s'occupant du tutoriel de notre jeu.

gameMode
Ce paquetage contient l'ensemble des composants Angular s'occupant de la vue de création, d'attente et de jeu d'une partie .

lobby
Ce paquetage contient les composantes Angular s'occupant de la vue dans d'un lobby

gameView
Ce paquetage contient l'ensemble des composantes Angular s'occupant de la vue d'une partie.

gameCreation
Ce paquetage contient les composantes Angular s'occupant de la création d'un lobby

profile
Ce paquetage contient l'ensemble des composantes Angular faisant partie de la vue de profil.

warning
Ce paquetage contient les composantes Angular s'occupant dess dialogs ("pop-up") qui demandent une vérification avant de faire une action (Ex: Quitter une page).

achievements
Ce paquetage contient la composante Angular s'occupant de la vue des trophées.

mainMenu
Ce paquetage contient l'ensemble des composants Angular faisant partie du menu principal

wordImagePair
Ce paquetage contient l'ensemble des composantes Angular s'occupant de la créationdes paires mot-images.

leaderboard
Ce paquetage contient l'ensemble des composantes Angular s'occupant du tableau de classement.

authentication
Ce paquetage contient l'ensemble des composantes Angular s'occupant de l'identification de l'utilisateur et de la création de comptes.

chat
Ce paquetage contient l'ensemble des composants Angular s'occupant de la vue du clavardage.

4.2.5 Services

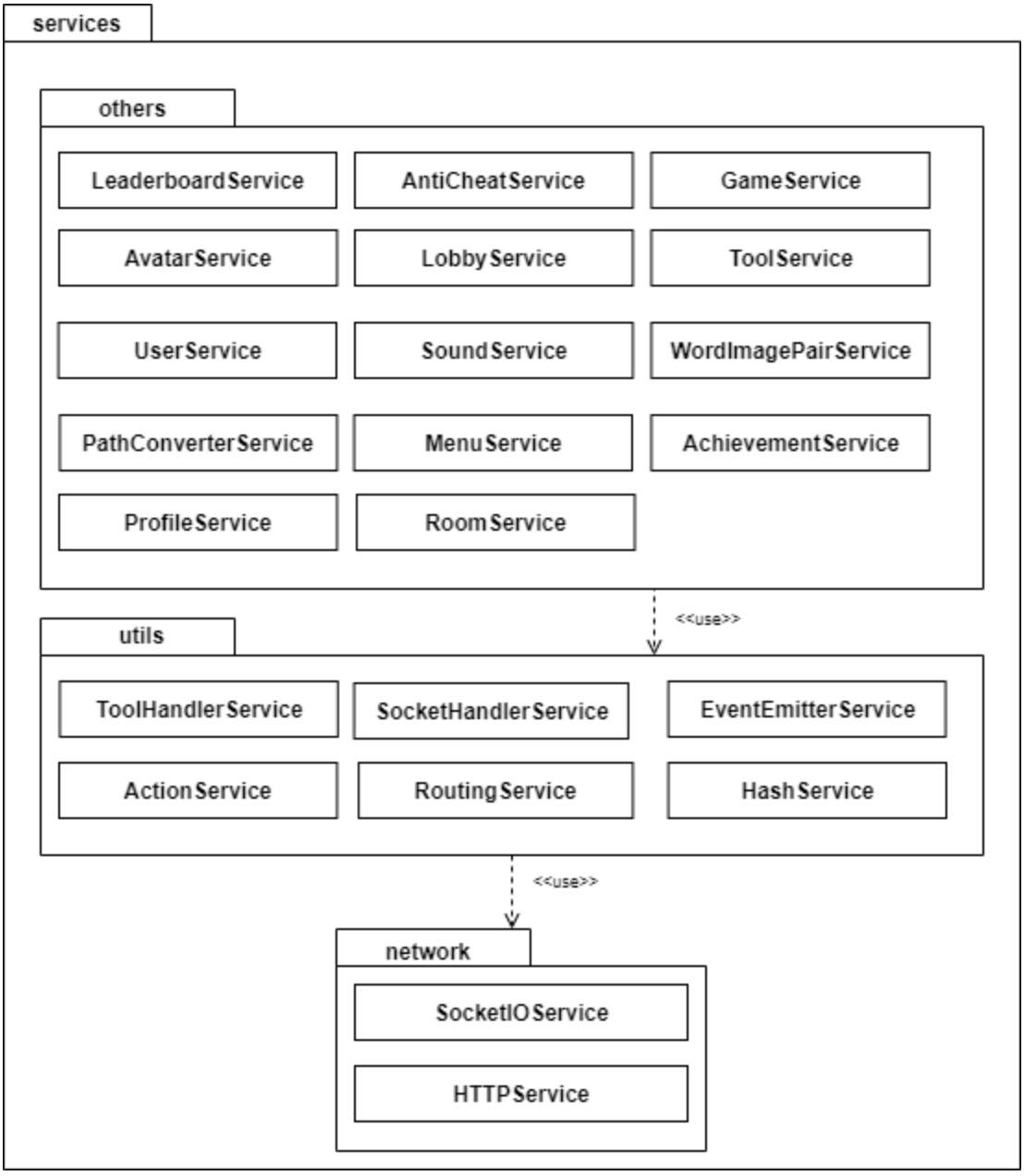

Figure 4.11. Diagramme de paquetage du client lourd.

network
Le paquetage *network* contient les services nécessaires à la communication avec le serveur. SocketIOService établie et se charge de la communication socket et HTTPService envoie et reçoit les échanges https.

utils
Ce paquetage est l'intermédiaire entre la logique client et celle serveur.

others
Ce paquetage contient l'ensemble de services généraux qui seront utilisés par les diverses composantes de l'application. Plus spécifiquement, les services dans ce paquetage servir.

4.3. Serveur

4.3.1. Architecture du serveur

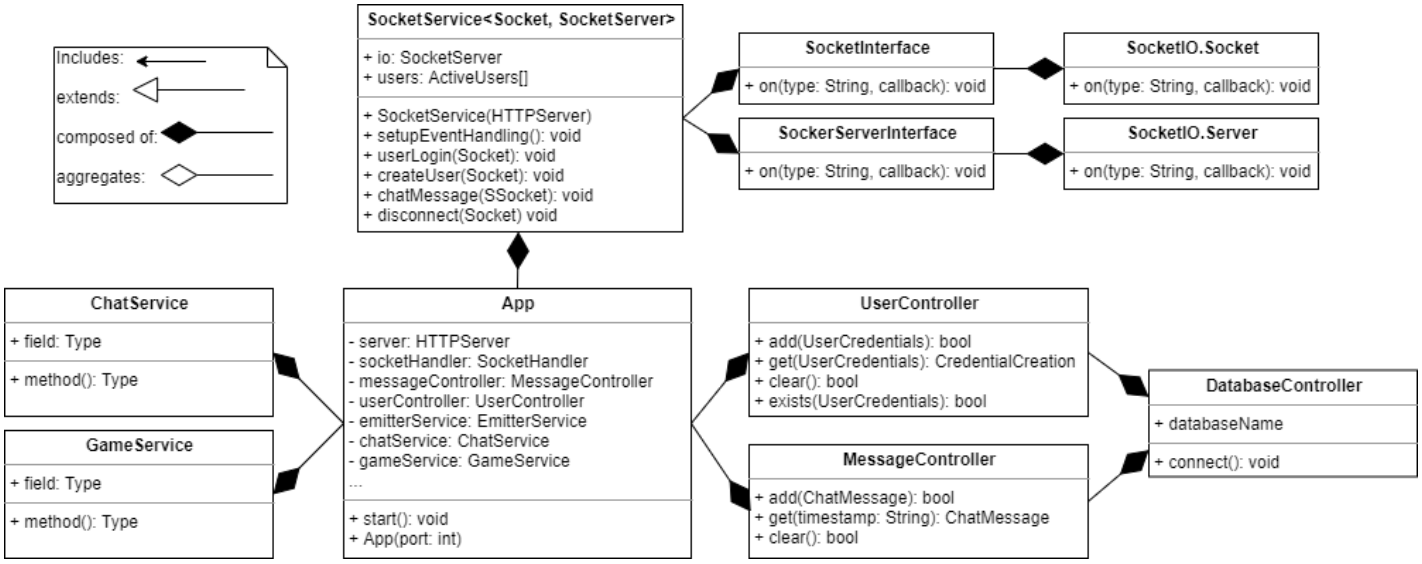

Figure 4.12. Diagramme de classes simplifié du serveur

App
Cette classe contient et instancie toutes les classes. Elle sert aussi à instancier le serveur HTTP ainsi que la classe express.Express servant à démarrer le serveur.

GameService
Gère les événements émis par les utilisateurs lors des parties. Contient la liste de *Game* en cours ainsi que de *Lobby* et appelle les méthodes concernées de ces classes lors d'événements. La logique des parties est gérée par la classe *Game.*

ChatService
Gère les événements émis par les utilisateurs ainsi que la logique de messagerie et de *rooms*.

DataBaseController
Gère la connexion à la base de données MongoDB distante.

UserController
Gère les méthodes *Mongoose* et procure une interface haut niveau de gestion dans la base de données. Gère les informations des utilisateurs.

MessageController
Gère les méthodes *Mongoose* et procure une interface haut niveau de gestion dans la base de données. Gère les informations des salles de messageries.

SocketService
Gère les événements de Sockets utilisateurs et appelle émets les événements serveurs appropriés.

SocketInterface && SocketServerInterface
Wrapper permettant l'utilisation de la librairies Socket.IO pour les événements.

4.3.2 Services

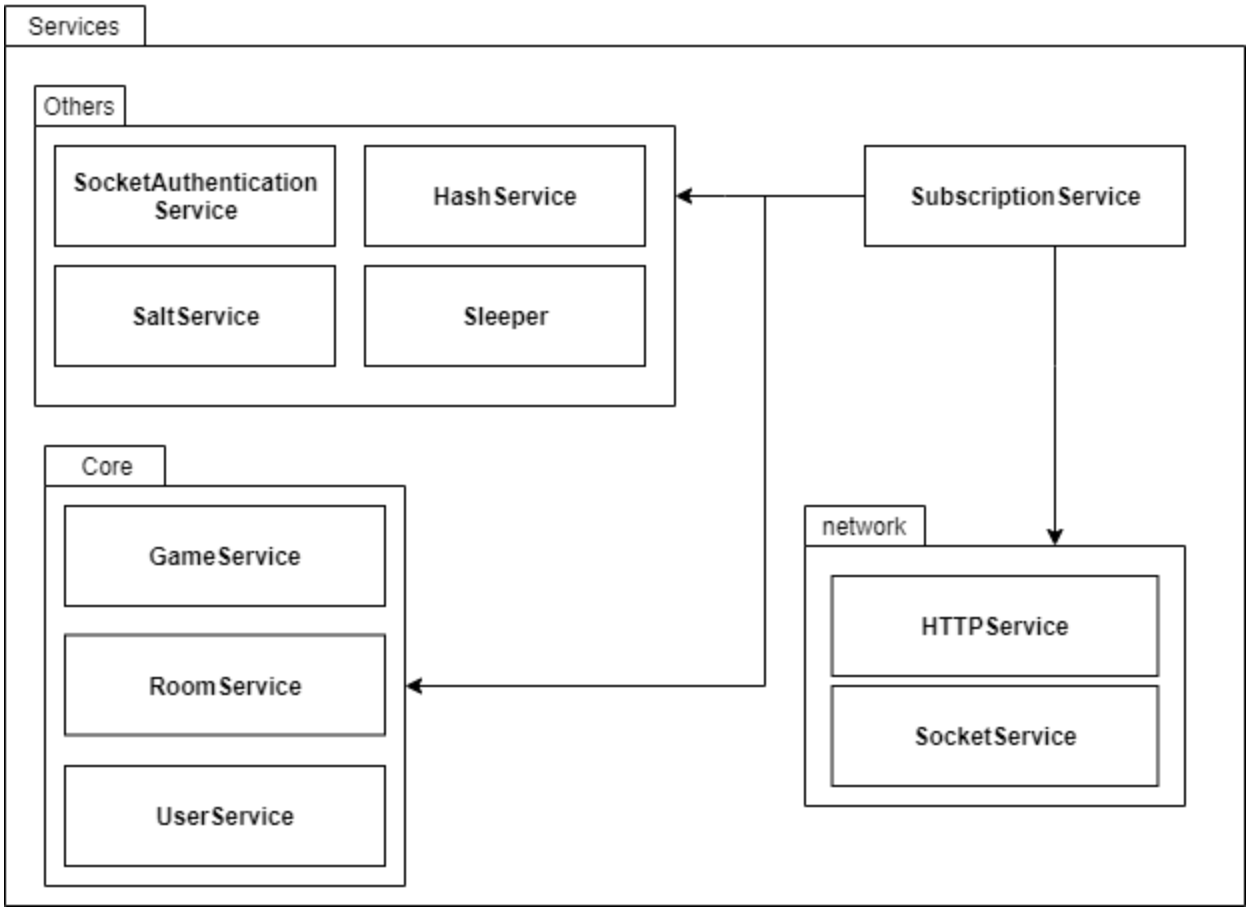

Figure 4.12. Diagramme de paquetage du serveur.

SubscriptionService
Gère tous les appels de méthodes lors des événements serveurs.

Network
Gère la logique de communication client et contient HTTP service qui gère le REST API permettant d'obtenir les informations clients permanentes (*trophés, statistiques, etc.*). Il contient aussi le SocketService qui gère les événements par sockets des utilisateurs.

Core
Paquetage gérant la logique du serveur et contient toutes les classes actives sur le serveur (*toutes les parties, utilisateurs et rooms*) qui sont dans leurs classes respectives.

Others
Paquetage gérant la logique d'authentification de l'utilisateur lors de la connexion et création de compte.

4.3.3 Gestion des évènements sur le serveur

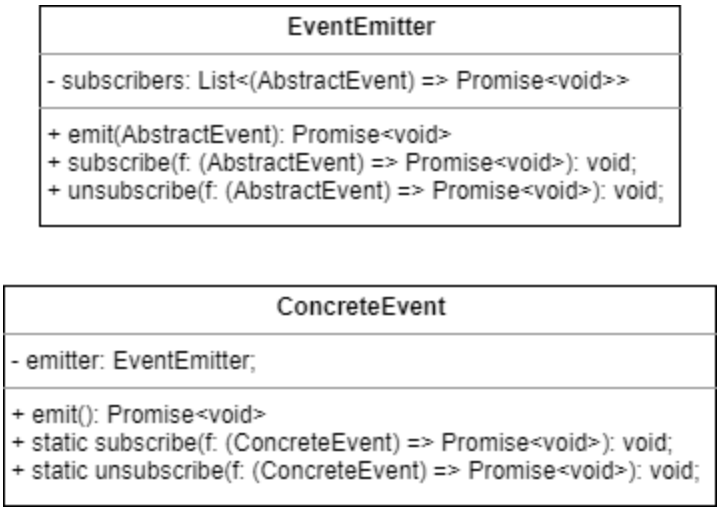

Figure 4.13. Exemple de classe concrète Event

Cette architecture nous permet de faire par exemple: "ChatEvent.subscribe((ev) => userService.message(ev))" et "await new ChatEvent(...).emit()", ce qui est très pratique et élégant.

4.3.4 Base de Données

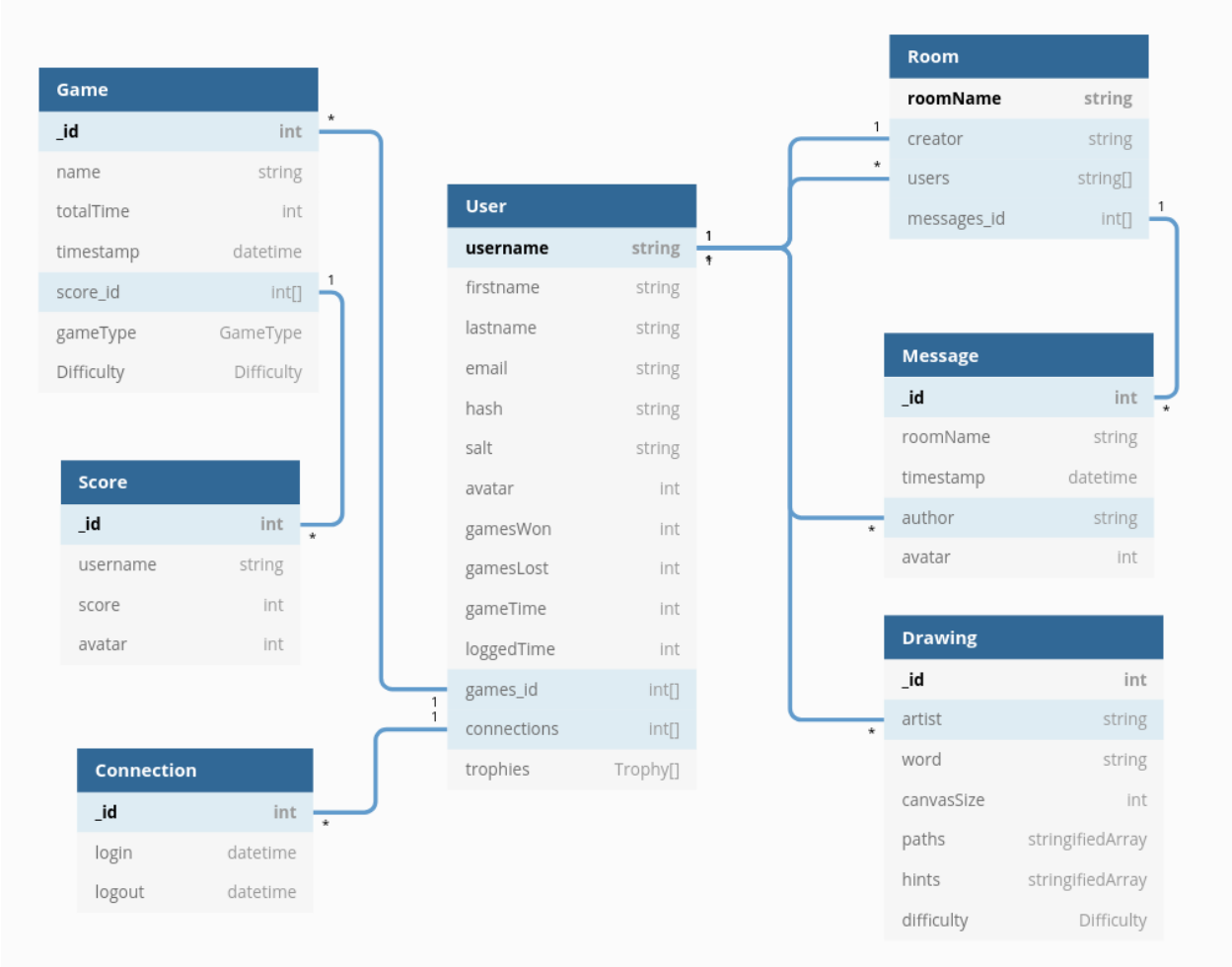

Figure 4.14 Schéma de notre base de données

5. Vue des processus

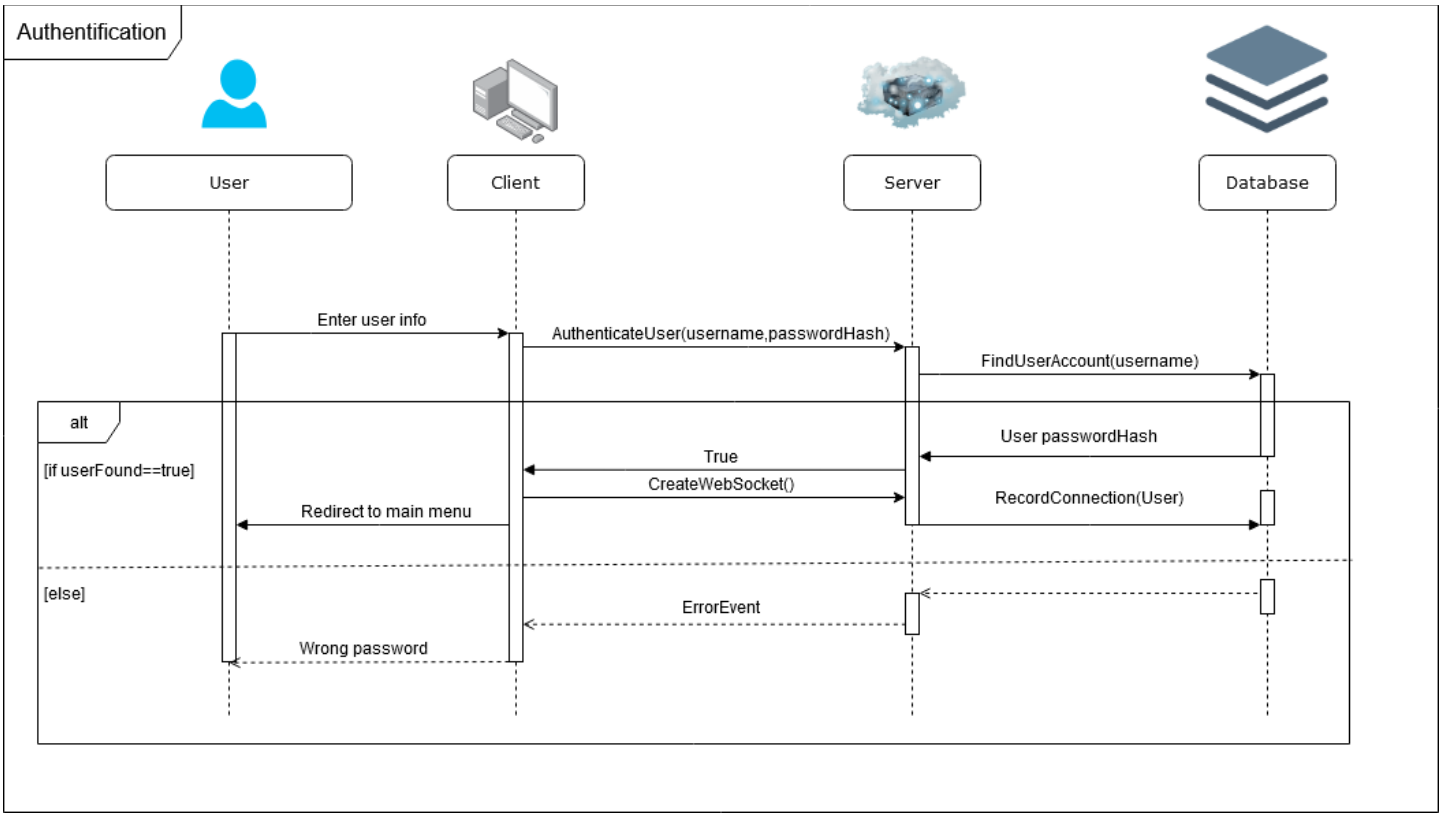

Figure 5.1. Diagramme de séquence 1: Authentification.

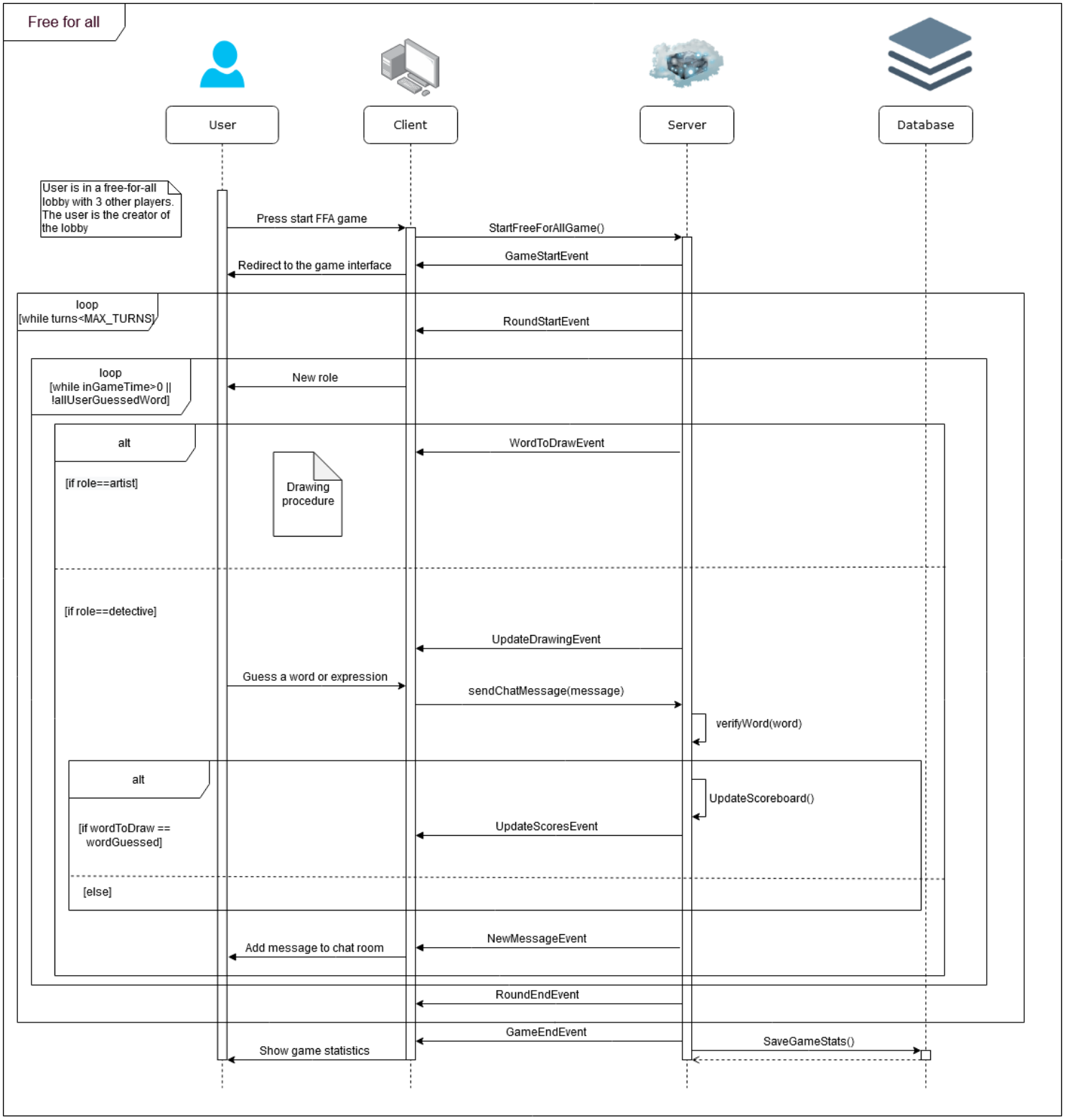

Figure 5.2. Diagramme de séquence 2: Partie mêlée générale.

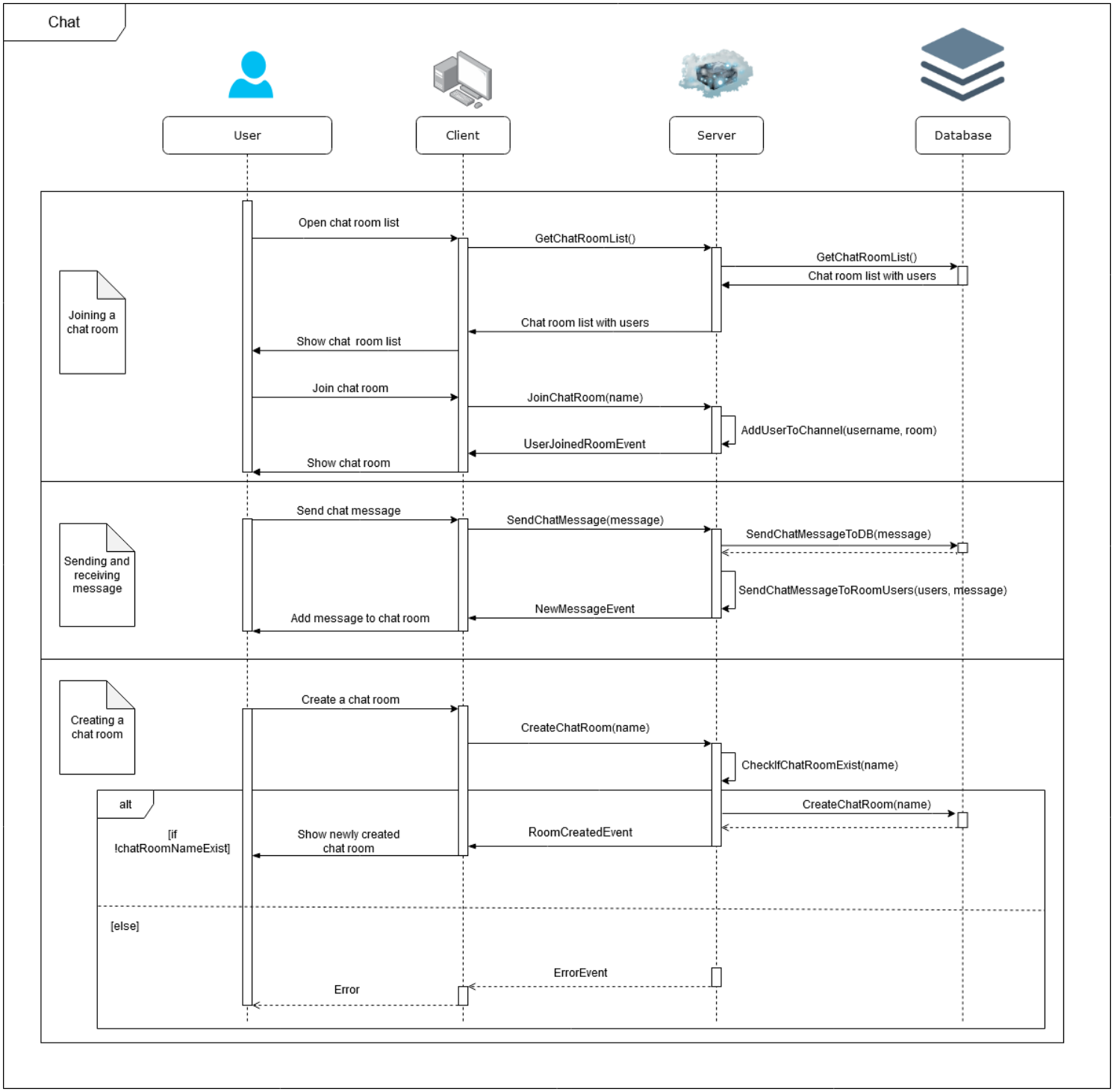

Figure 5.3. Diagramme de séquence 3: Clavardage.

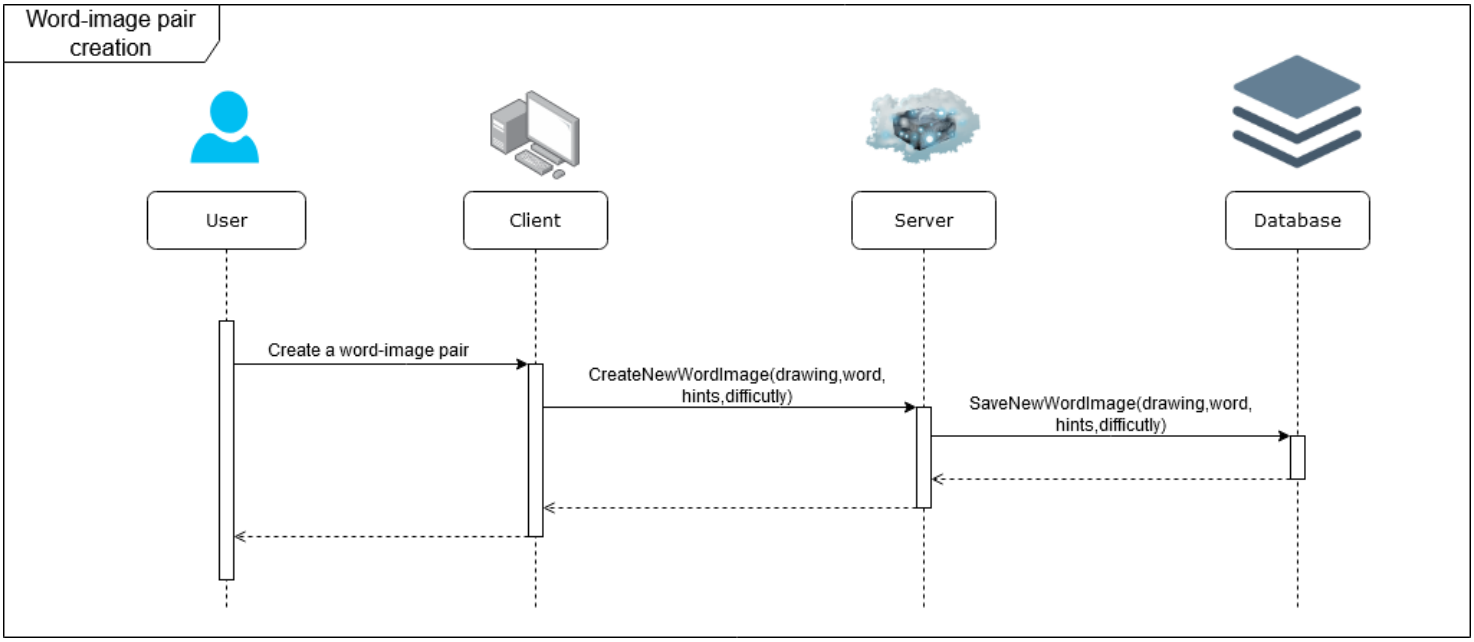

Figure 5.4. Diagramme de séquence 4: Création d'une paire mot-image.

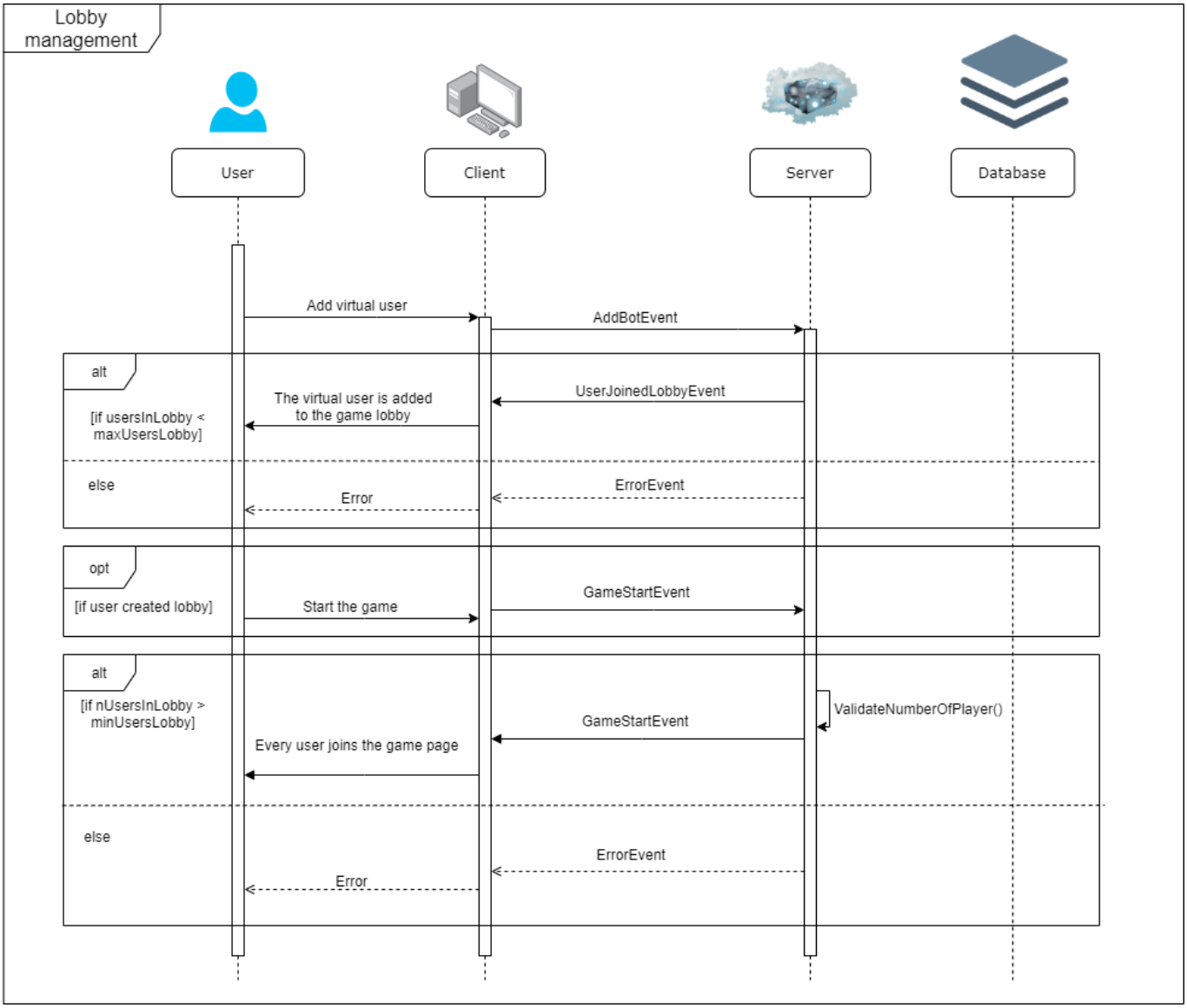

Figure 5.5. Diagramme de séquence 5: Gestion d'un Lobby.

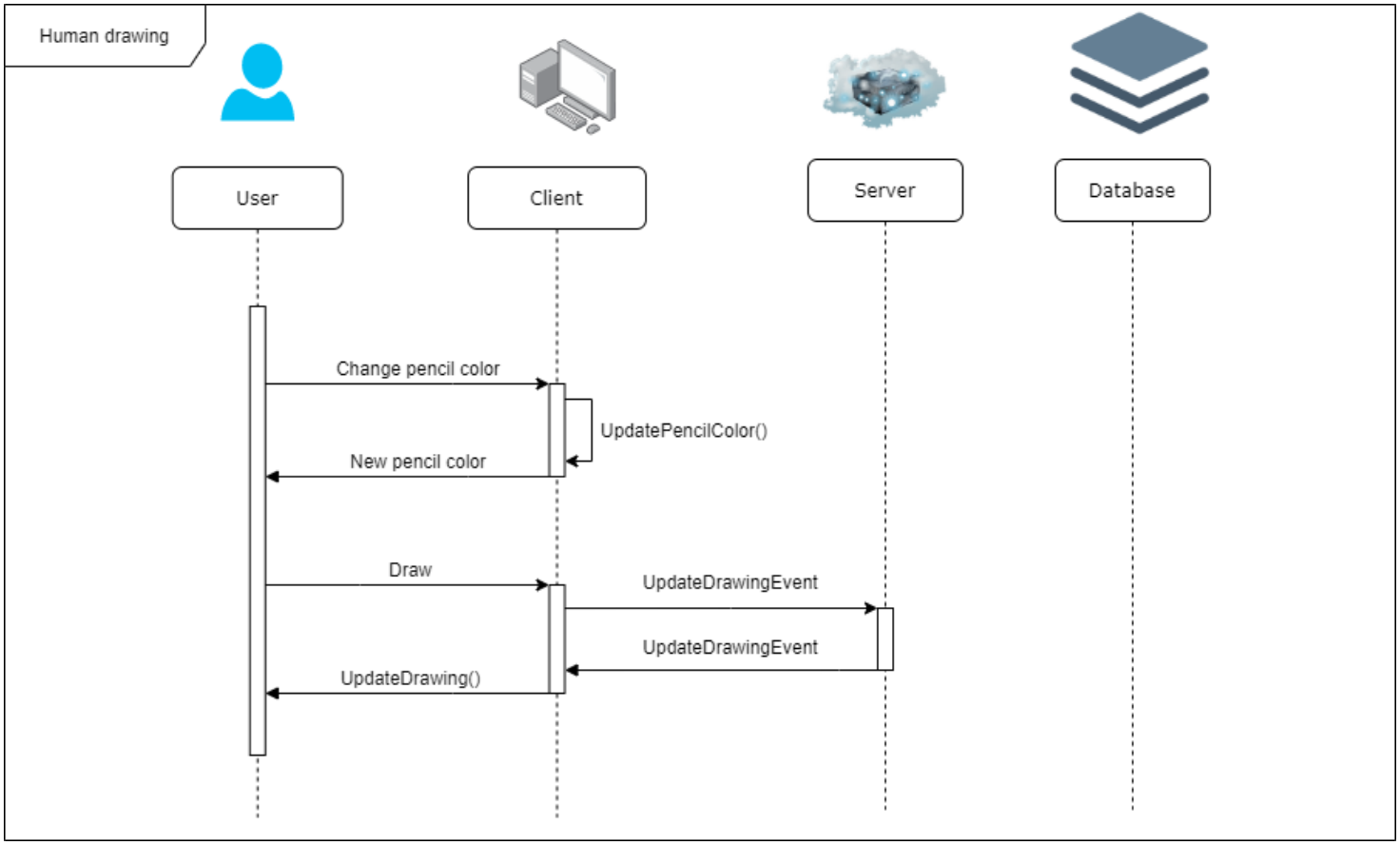

Figure 5.6. Diagramme de séquence 6: Dessin d'un utilisateur.

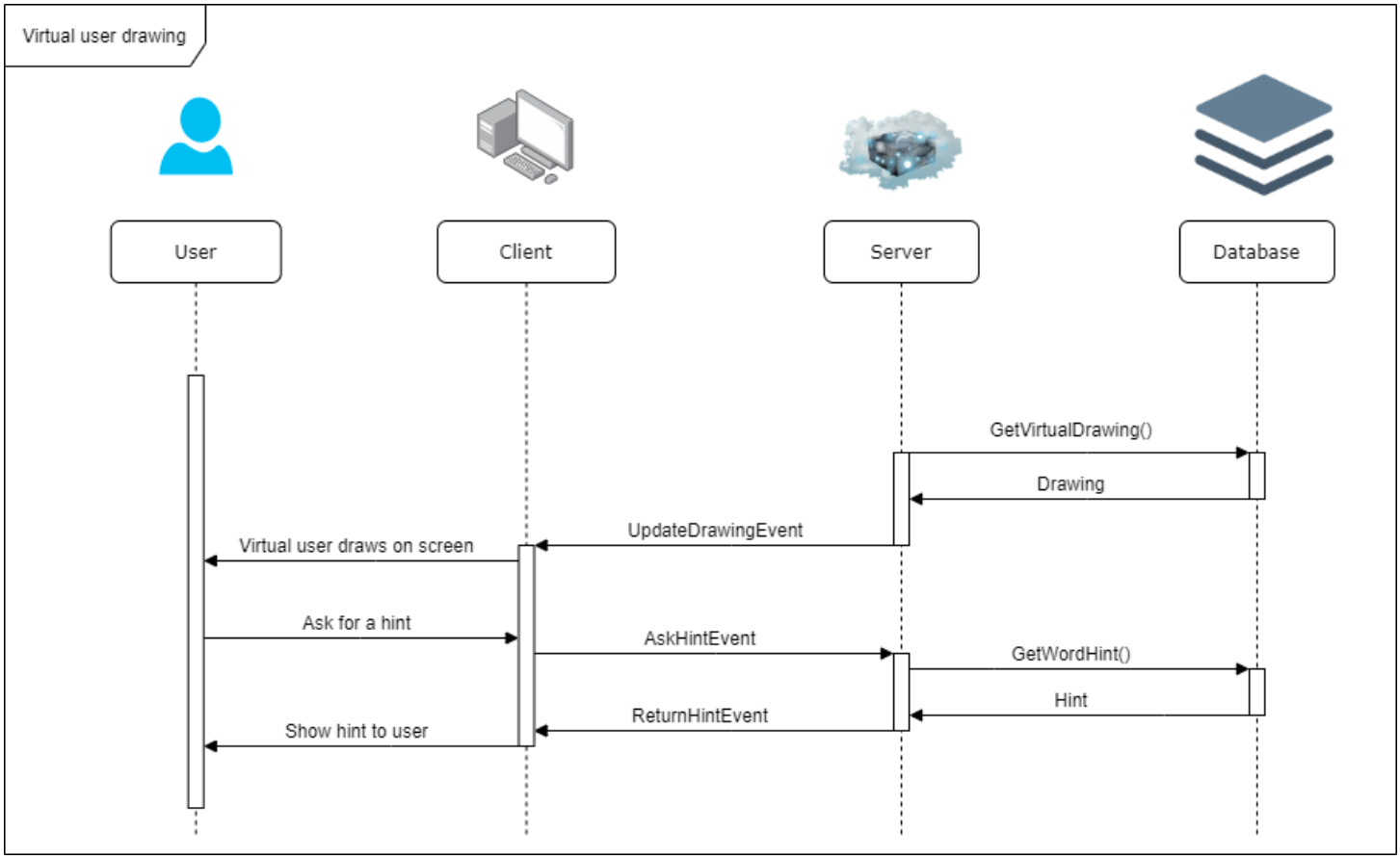

Figure 5.7. Diagramme de séquence 7: Dessin d'un utilisateur virtuel.

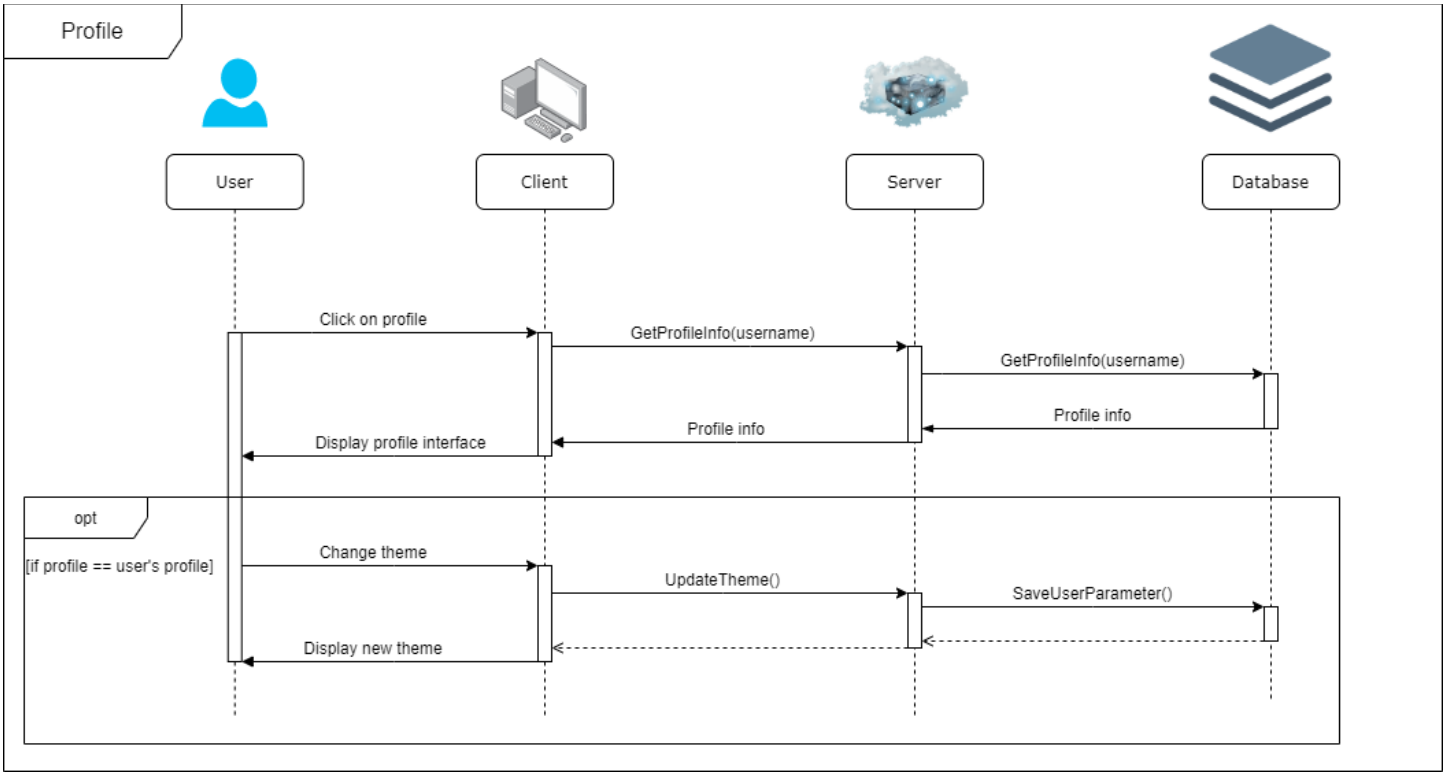

Figure 5.8. Diagramme de séquence 8: Accès et personnalisation du profil.

6. Vue de déploiement

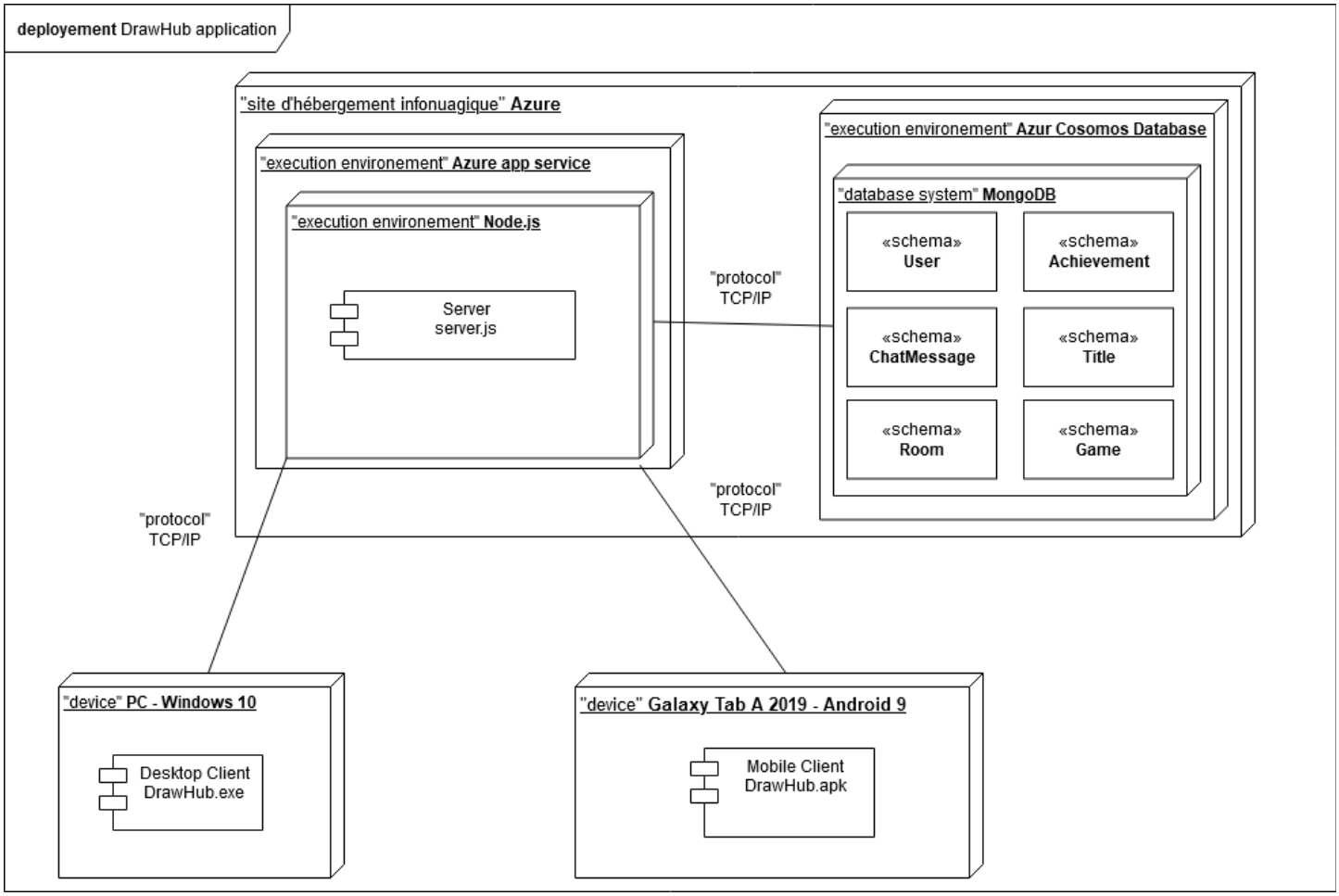

Figure 6.1. Diagramme de déploiement de DrawHub.

7. Taille et performance

L'architecture de notre logiciel doit être en mesure de respecter les caractéristiques de taille et de performance impliquées dans les spécifications des requis du système (SRS). Le système doit donc être capable de supporter 4 parties simultanées d'au minimum 2 utilisateurs sans avoir plus de 500 ms de latence.

Le serveur doit traiter une requête envoyée par le client lourd et léger et acheminer la réponse aux clients en moins de 250 ms.

Pour le client lourd, il est nécessaire d'avoir au moins 1 GB de mémoire vive disponible sur son appareil pour s'assurer qu'il n'y ait en aucun temps des ralentissements notables qui pourraient affecter l'expérience utilisateur. Pour les mêmes raisons, le client léger doit au minimum avoir 500 Mb de mémoire vive disponible.

L'espace mémoire minimum requis sur la tablette Android est de 25 Mb pour pouvoir télécharger et installer correctement DrawHub. Pour le client lourd, un espace de stockage minimal de 200 Mb est requis. Pour optimiser l'espace utilisé par l'application, les informations stockées sur la base de données comme les détails du profil utilisateur seront téléchargées uniquement lorsque l'utilisateur en fera la demande, soit lorsqu'il accédera à la page détaillée de son profil.

La taille maximale de l'image de 32x32 pixels représentant l'avatar de l'utilisateur pouvant être téléversé par celui-ci est de 20 Kb.

Pour le client lourd, une utilisation maximale de 50% d'un processeur de bas de gamme de dernière génération (2020) pourrait être utilisé lors d'une partie de DrawHub. Pour le client léger, un maximum de 30% du processeur de la tablette peut être consommé lors de l'utilisation du logiciel DrawHub.