Équipe 210

Draw It Up!
Document d'architecture logicielle

Version 1.3

Historique des révisions

Date	Version	Description	Auteur
2021-02-15	1.0	Remplissage du tableau pour la vue logique du serveur.	Benjamin Saint-Cyr, Marc-André Filion
2021-02-18	1.1	Ajout des diagrammes de paquetage. Ajout des diagrammes de séquence pour la vue des processus. Ajout des diagrammes de vue logique des clients.	Marc-André Filion, Charles-Édouard Turcot, Bouchra-nour-el-yakin Dahamni, Vincenzo Pucci-Barbeau
2021-02-19	1.2	Dernières modifications et révision finale en équipe avant la remise.	Équipe 210
2021-04-19	1.3	Révision finale de l'équipe avant la remise	Équipe 210

Table des matières

Document d'architecture logicielle

1. Introduction

Dans ce document, nous allons parler de l'architecture logicielle de notre projet. Tout d'abord, nous allons parler de nos objectifs et contraintes architecturaux que nous avons. Deuxièmement, nous allons présenter les cas d'utilisation de nos produits. Troisièmement, nous allons présenter les vues logiques de nos produits. Ensuite, nous allons présenter les vues de processus et de déploiement de nos produits. Finalement, nous allons parler des contraintes de taille et de performance de nos applications.

2. Objectifs et contraintes architecturaux

L'architecture doit permettre aux différents clients de fonctionner sur leurs plateformes respectives.

2.1 Client léger

2.2.1 Performance

L'architecture doit permettre la bonne performance en utilisant adéquatement les concepts de fils d'exécution afin de ne pas surcharger le fil d'exécution responsable du rendu visuel de l'application et d'exécuter les calculs lourds sur des fils en arrière-plan. Le client pourra donc s'exécuter de façon fluide et performante.

2.2.2 Modularité du code

L'utilisation du concept de fragments en Kotlin nous permet d'avoir des modules de code qui sont présents à plusieurs endroits dans l'application sans avoir à répliquer le code à divers endroits. Cela nous permet aussi de facilement intégrer des éléments de l'interface utilisateur à plusieurs endroits et d'avoir un contrôle plus compartimenté sur la vue de l'application.

2.2 Client lourd

2.2.1 Performance

L'architecture doit permettre la bonne performance de l'application sur différents systèmes. Notre architecture doit être légère et facile à exécuter pour permettre au plus grand nombre d'utilisateurs de jouer.

2.2.2 Réutilisation du code

L'emploi d'Angular pour réaliser l'application permet de créer ce qui est, en quelque sorte, des balises HTML personnalisées pour la vue d'application, que l'on appelle des *components.* Ainsi, ces *components* peuvent être appelés lorsque nécessaire pour répliquer un élément déjà construit sur différentes sections de l'application, ce qui favorise la réutilisation du code.

2.3 Serveur

2.3.1 Confidentialité des données

L'architecture doit protéger les mots de passe des utilisateurs et leur information privée. La base de données est protégée sur une instance S3 de AWS.

2.3.2 Fiabilité

Tous les clients se basent sur le serveur pour accéder à l'information des comptes et les différents services de jeu. Il est donc crucial que le serveur soit disponible 99.99% du temps. C'est pourquoi nous utilisons une instance t2.micro du service Amazon EC2. De plus, afin de gérer les demandes de plusieurs clients en même temps nous utilisons l'implémentation de la communication de processus séquentiel (CSP) offerte par Go ce qui nous permet une façon facile et sans deadlock de gérer la concurrence.

2.3.3 Performances

Le serveur doit pouvoir gérer plusieurs clients avec un faible temps de réponse afin de préserver la qualité de l'expérience utilisateur. Alors, nous déployons notre application sur un service infonuagique avec peu de latence. De plus, nous utilisons un langage compilé afin de réduire le temps nécessaire pour gérer les demandes au serveur.

3. Vue des cas d’utilisation

3.1 Diagramme de cas d’utilisation pour identification (client léger et client lourd)

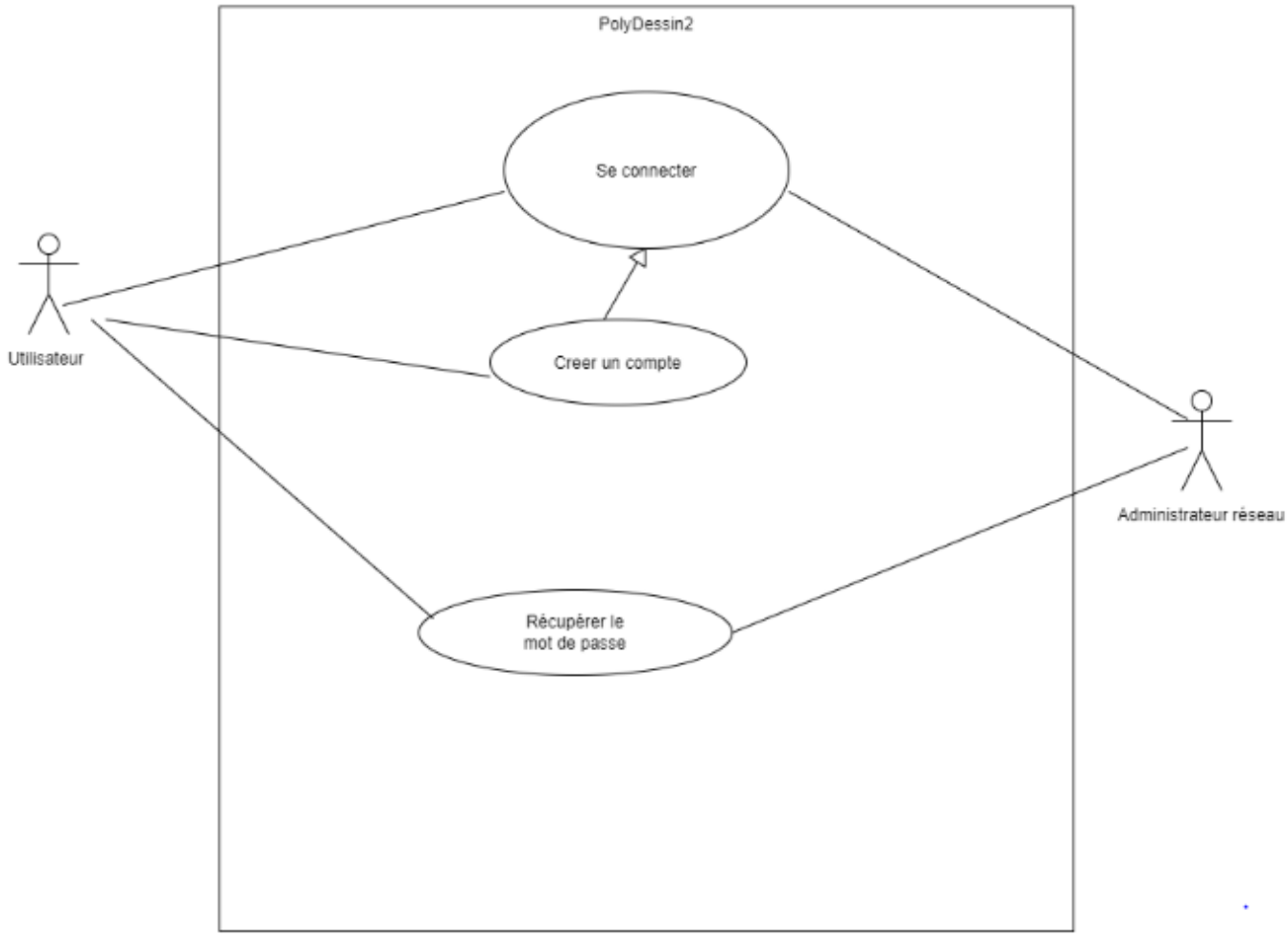

Figure 1 - Diagramme de cas d’utilisation pour l’identification d’un utilisateur

3.2 Diagramme de cas d'utilisation pour le clavardage (client lourd)

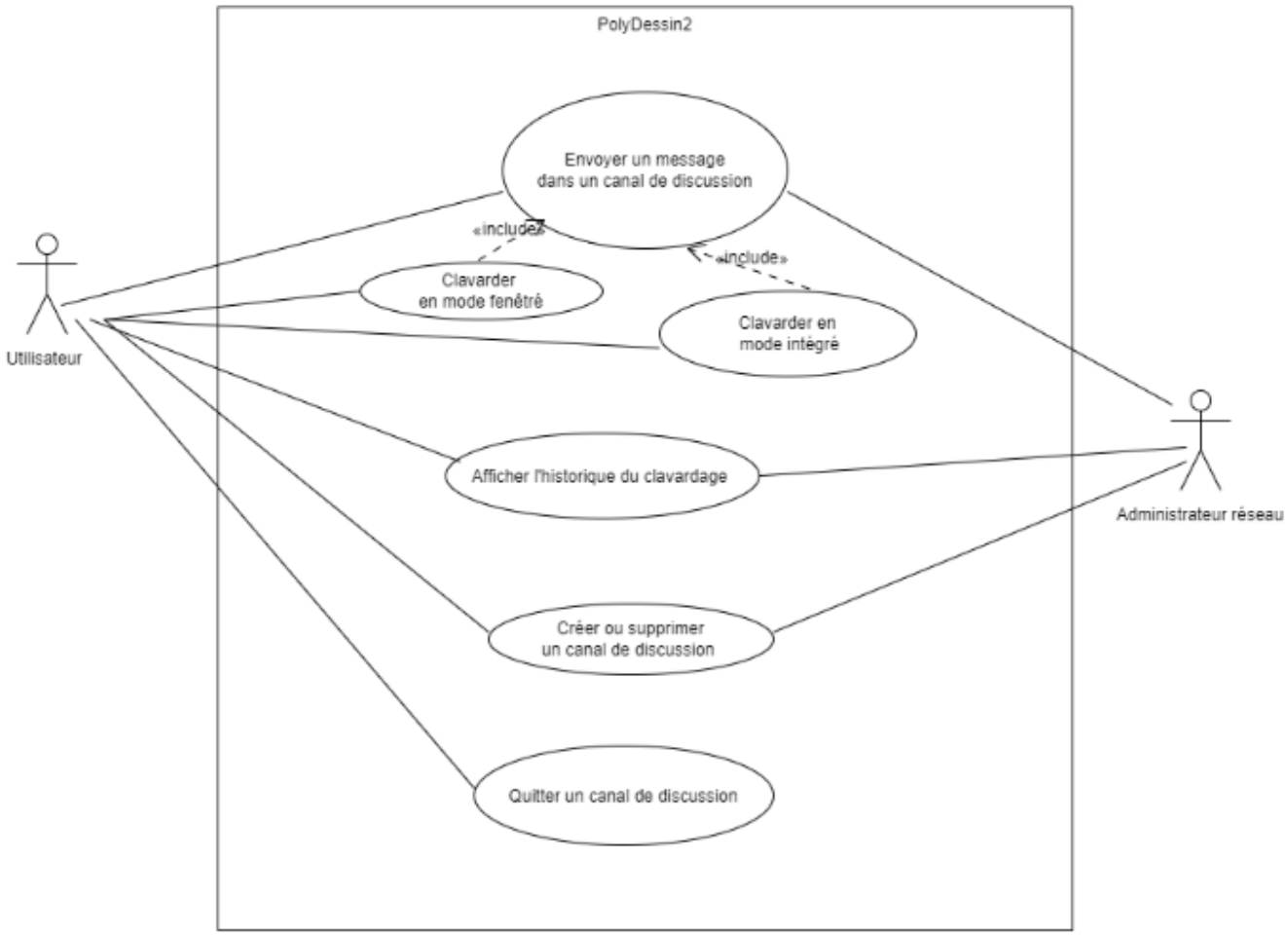

Figure 2 : Diagramme de cas d'utilisation pour le clavardage

3.3 Diagramme de cas d'utilisation pour clavardage (client léger)

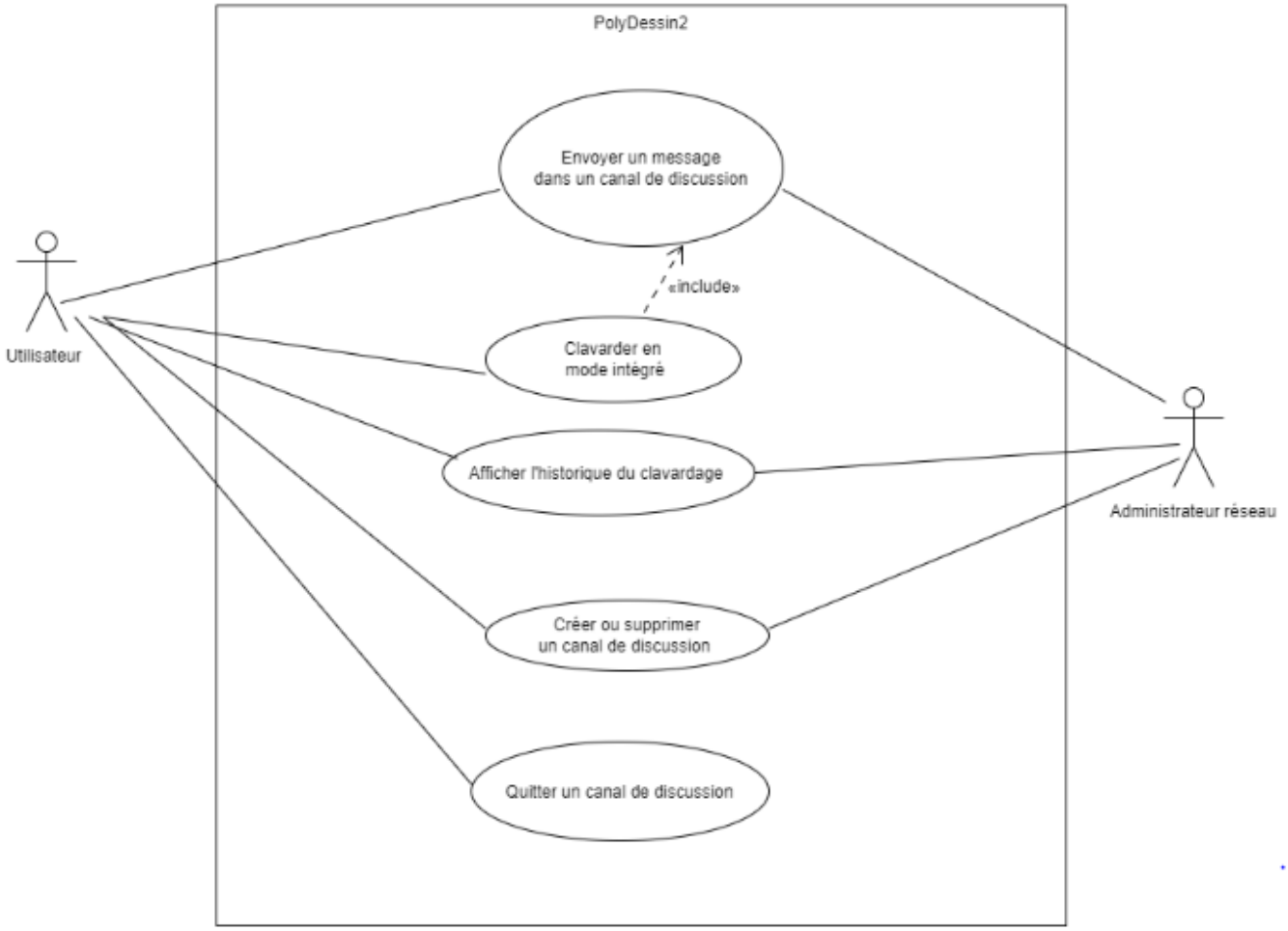

Figure 3 : Diagramme de cas d'utilisation pour le clavardage

3.4 Diagramme de cas d’utilisation pour tutoriel (client léger et client lourd)

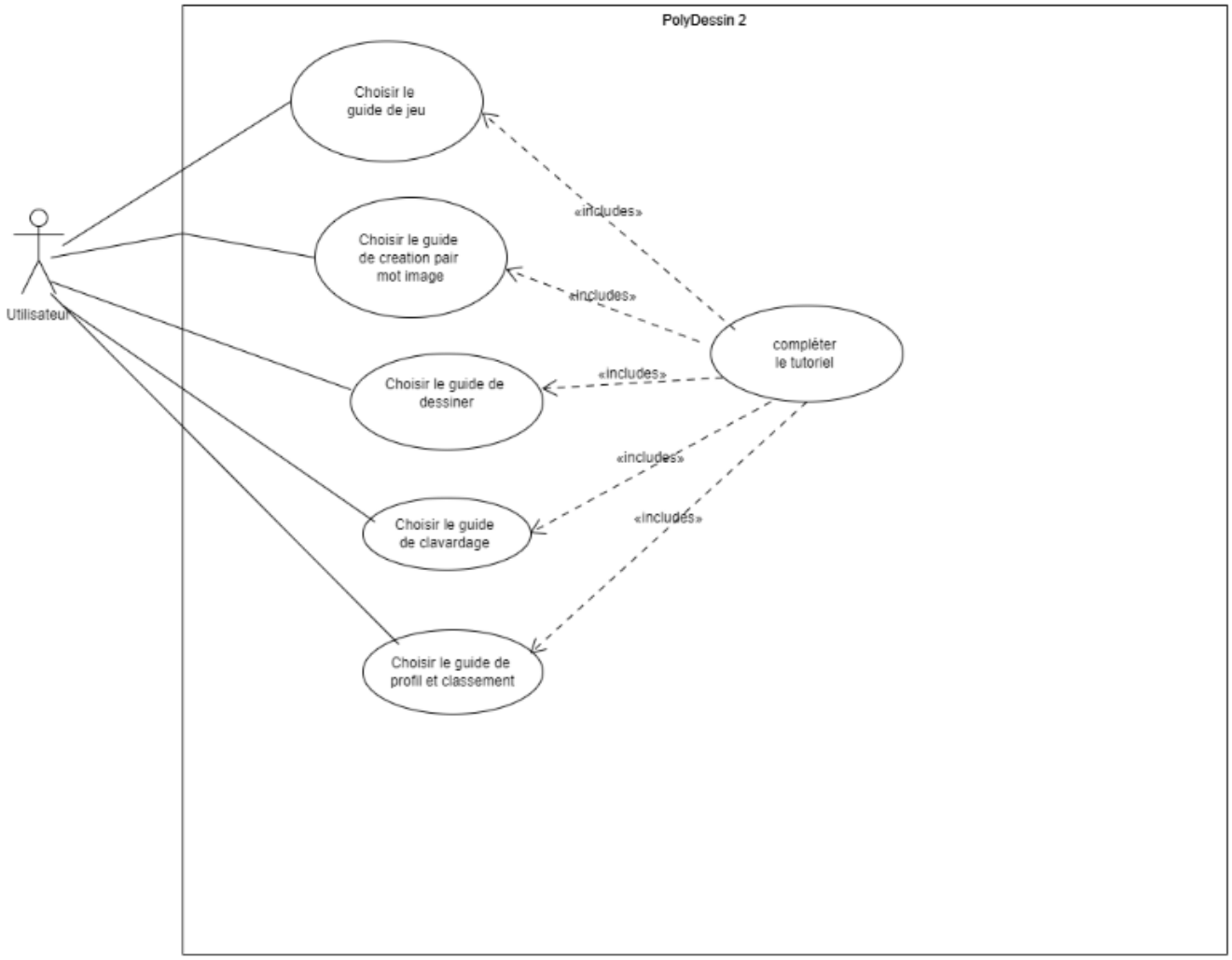

Figure 4 : Diagramme de cas d’utilisation pour le tutoriel

3.5 Diagramme de cas d'utilisation pour profil utilisateur, historique, classement et trophées (client léger et client lourd)

Figure 5 : Diagramme de cas d'utilisation pour l'accès au profil utilisateur, à l'historique et au classement et trophées.

3.6 Diagramme de cas d’utilisation pour jeu et équipe (client léger et client lourd) :

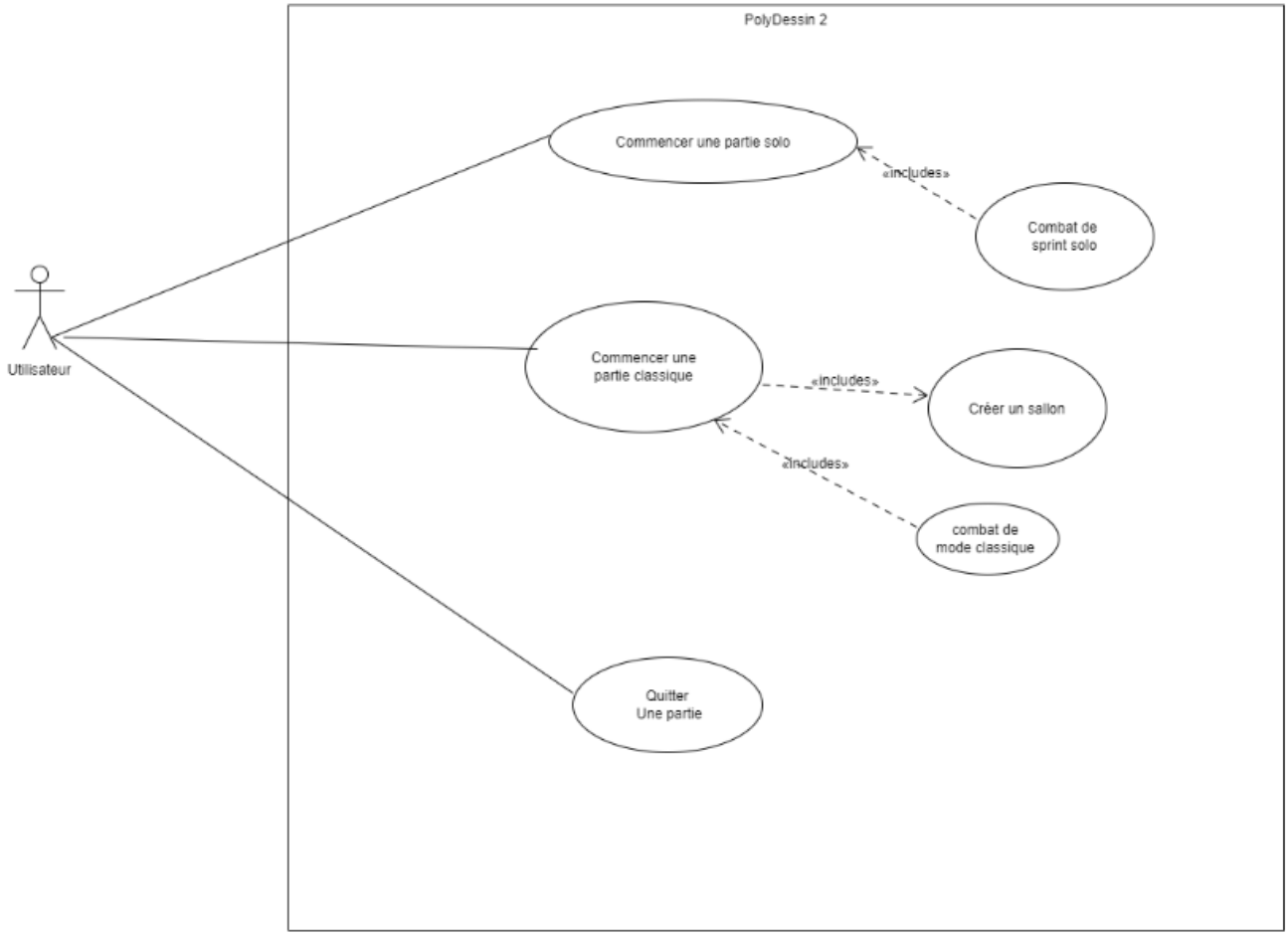

Figure 6 : Diagramme de cas d’utilisation pour les modes de jeu

3.7 Diagramme de cas d'utilisation pour création d'un jeu (client lourd)

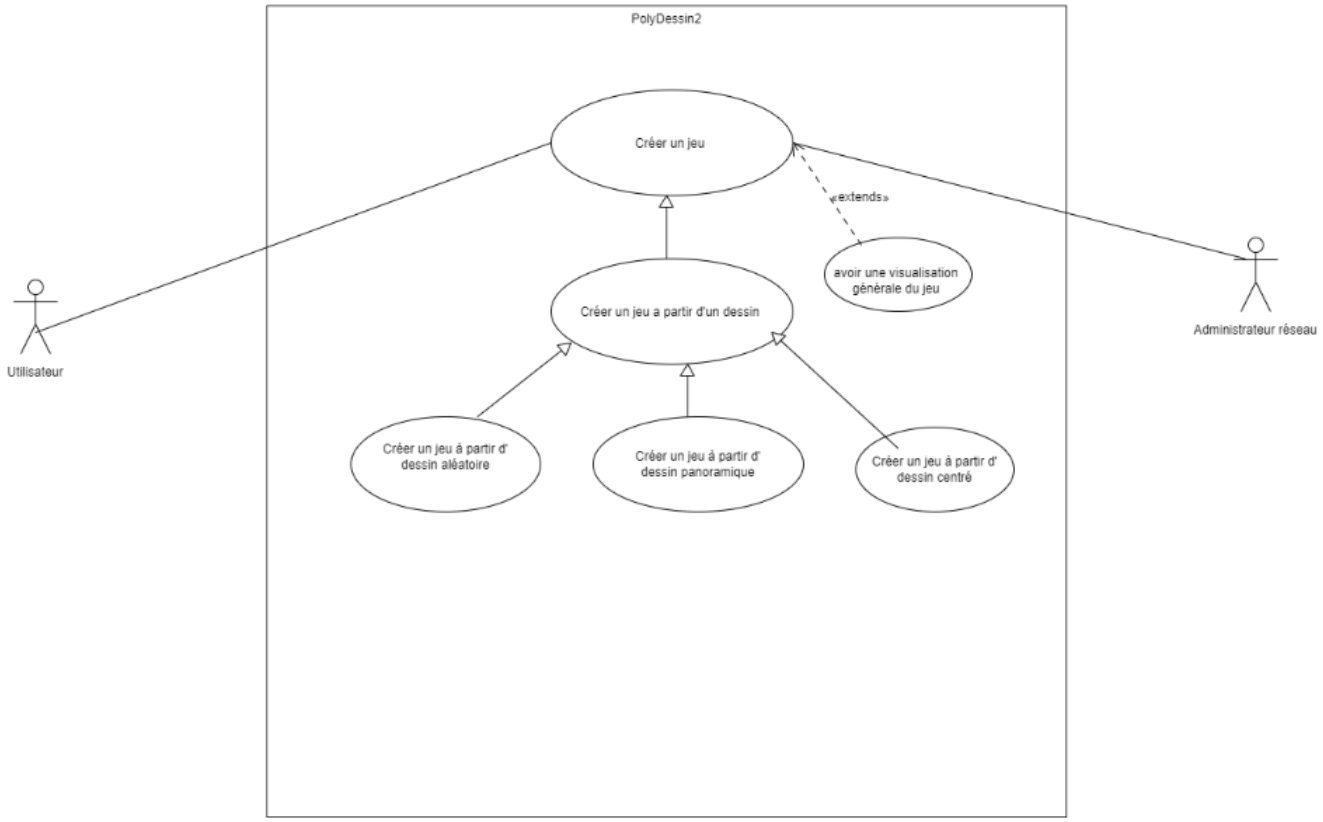

Figure 7 : Diagramme de cas d'utilisation pour la création d'un jeu

4. Vue logique

4.1. Fais-moi un dessin (vue globale)

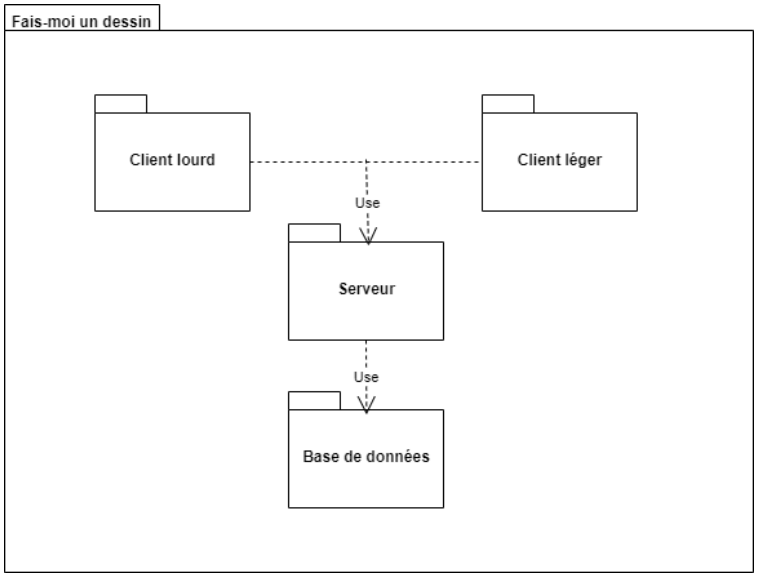

Figure 8 : Diagramme de paquetage d'application

4.2. Serveur

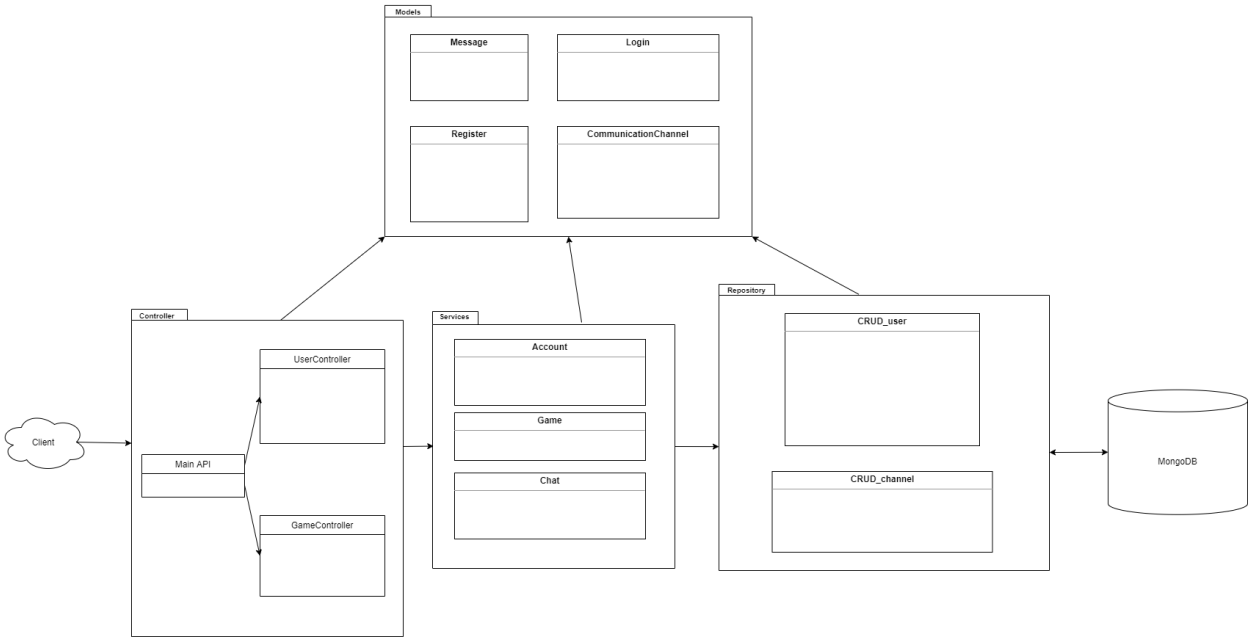

Figure 9 : Diagramme de paquetage du serveur de moyen niveau

Repositery
Ce paquet englobe les "classes" qui permettent d'effectuer les tâches CRUD (create, read, update, delete) en communiquant avec la base de données. Le CRUD_user s'occupera des opérations CRUD en lien avec les utilisateurs, puis le CRUD_channel pour les canaux de discussion.

Models
Ce paquet contient les structures (struct) des différents types d'objets créés.

Controller
Ce paquet englobe les contrôleurs qui serviront au routage vers les différents chemins selon le type de requête effectuée.

Services
Ce paquet contient la logique permettant la transition d'une requête via le controller vers le repository.

4.3. Client lourd

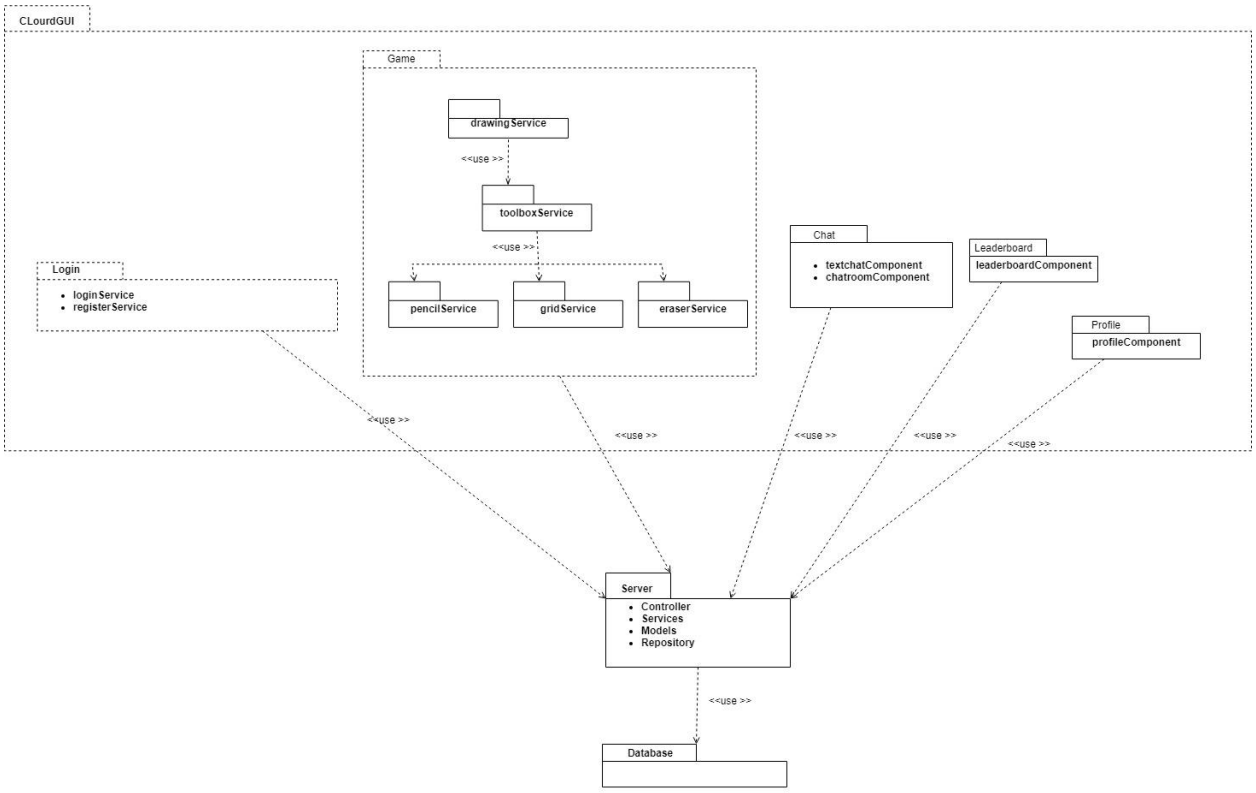

Figure 10 : Diagramme de paquetage du client lourd de moyen niveau

CLourdGUI
Ce paquet contient les différents paquets du client lourd.

Login
Ce paquet contient les différents services pour la connexion et la création de comptes pour l'application.

Game
Ce paquet contient les différentes composantes nécessaires pour dessiner, soit le canevas et les différents outils.

Chat
Ce paquet contient les différentes composantes nécessaires pour les canaux de discussions.

Leaderboard
Ce paquet contient les différentes composantes pour le classement dans l'application.

Server
Ce paquet contient les différentes composantes du serveur.

4.4. Client léger

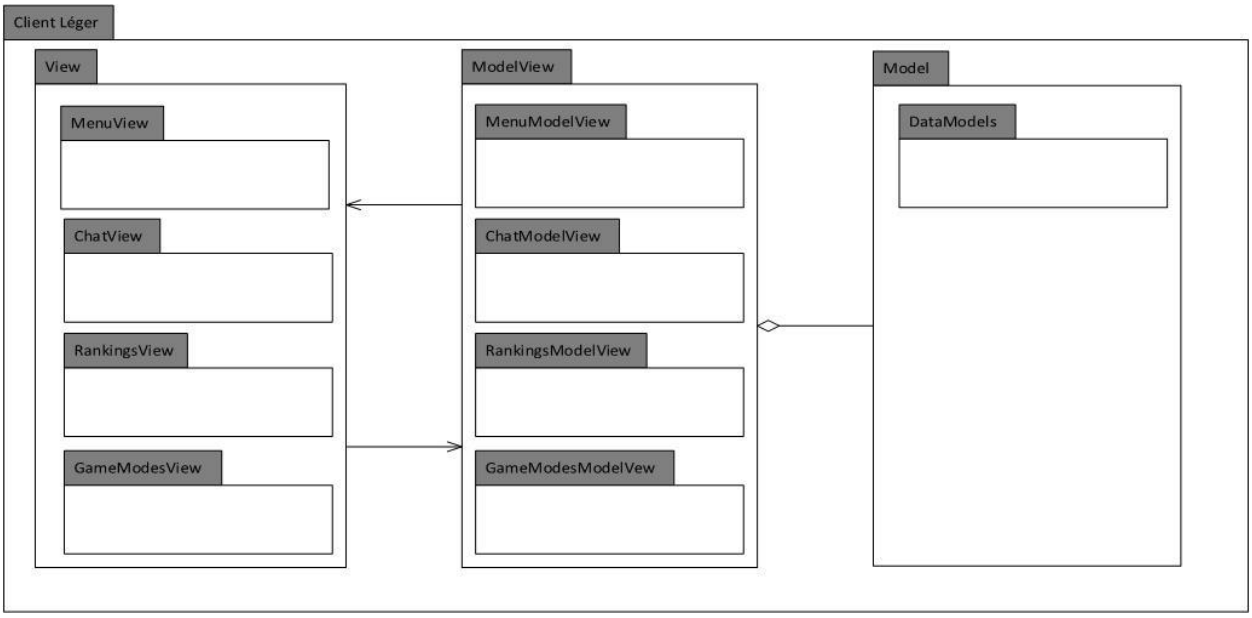

Figure 11 : Diagramme de paquetage du client léger de moyen niveau

MenuView
Ce paquet contient les différentes vues reliées au menu comme la connexion, la création de compte, mot de passe oublié ou menu principal.

ChatView
Ce paquet contient les vues de clavardage..

RankingsView
Ce paquet contient les vues liées au classement d'usagers et d'équipes ainsi que la vue des trophées.

GameModesView
Ce paquet contient les vues associées aux modes de jeu.

MenuModelView
Ce paquet contient les différents modèles de vues reliées au menu comme la connexion, la création de compte, mot de passe oublié ou menu principal.

ChatModelView
Ce paquet contient les modèles de vues de clavardage..

RankingsModelView
Ce paquet contient les modèles de vues liées au classement d'usagers et d'équipes ainsi que la vue des trophées.

GameModesModelView
Ce paquet contient les modèles de vues associées aux modes de jeu.

DataModels
Ce paquet contient les modèles de toutes les structures de données utilisées.

5. Vue des processus

5.1. Inscription

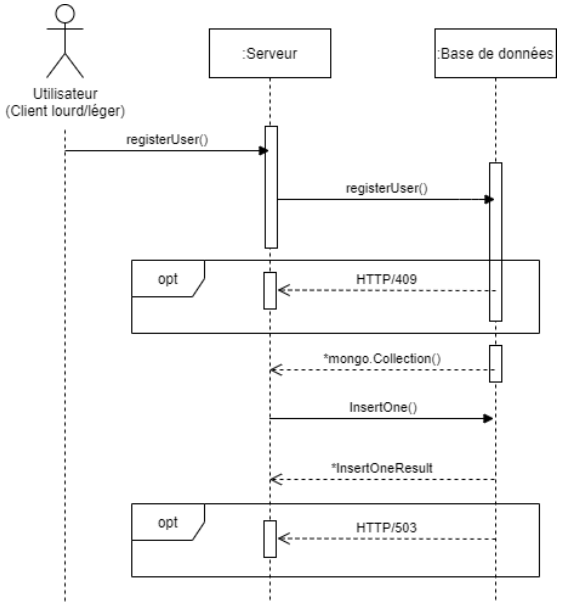

Figure 12 : Diagramme de séquence pour la création d'un compte

5.2. Connexion

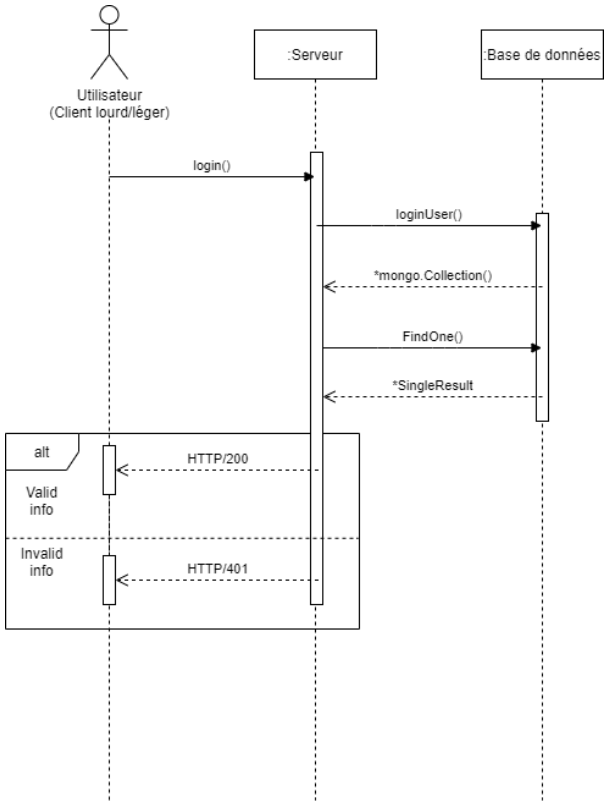

Figure 13 : Diagramme de séquence pour la connexion d'un utilisateur

5.3. Clavardage

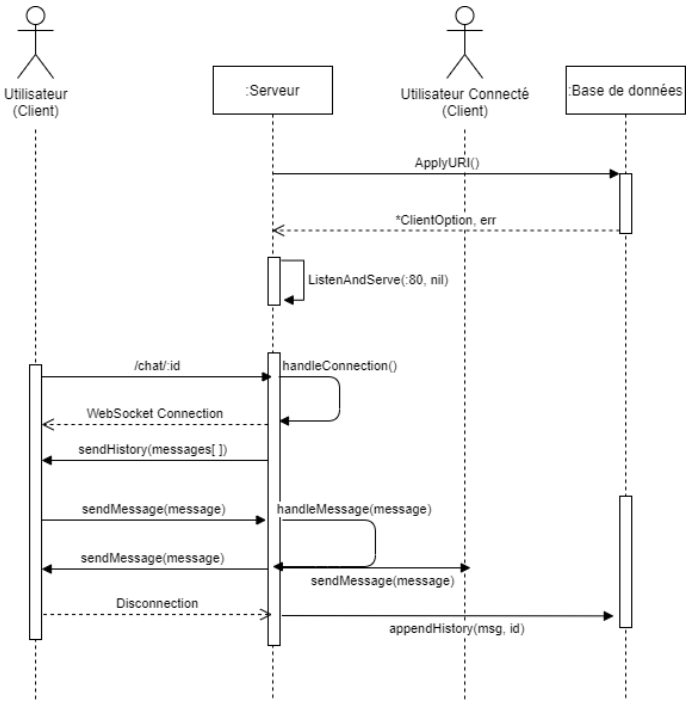

Figure 14 : Diagramme de séquence pour le clavardage instantané

5.4. Traçage de dessin

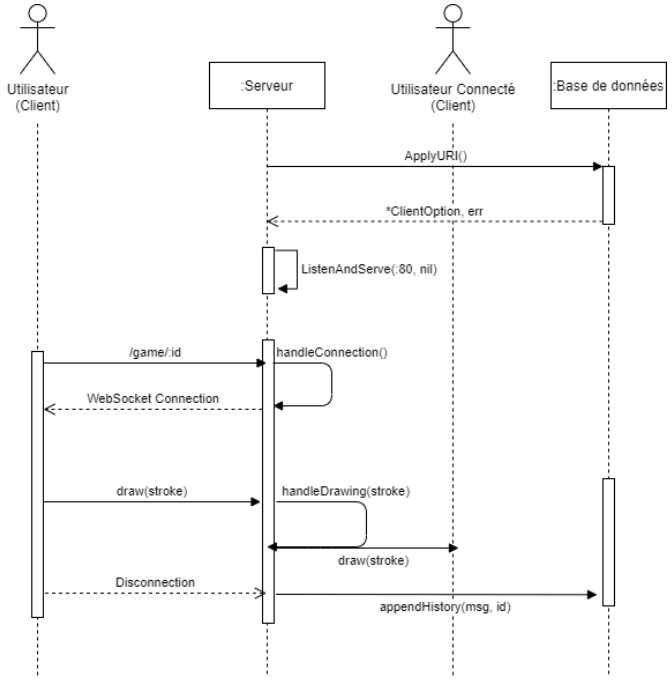

Figure 15 : Diagramme de séquence pour le traçage de dessin en temps réel

6. Vue de déploiement

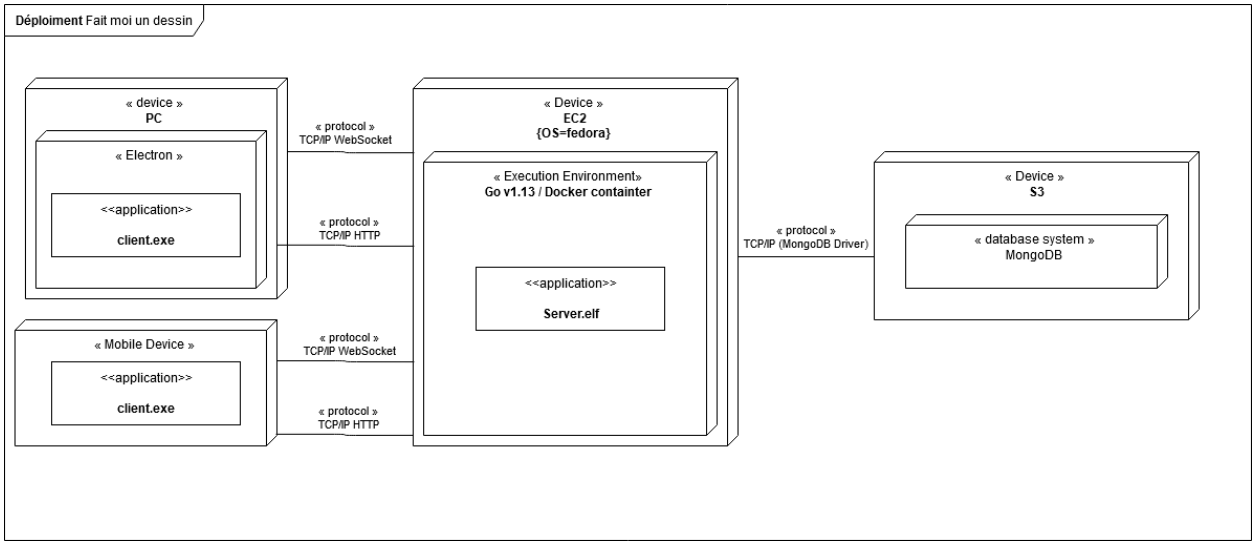

Figure 16 : Diagramme de déploiement pour l'application

7. Taille et performance

L'architecture logicielle de l'application a été conçue de façon à prendre en compte les limitations telles que décrites dans le SRS. Le serveur est limité par l'instance gratuite où il est déployé sur le nuage AWS, avec un maximum de 5 GB de mémoire et une base de données de 512 MB. Nous sommes aussi limités au niveau des machines qui exécutent les clients, principalement du côté du client léger puisque l'on a accès à seulement 2 GB de mémoire vive. Ce sont ces limitations qui ont poussé à la conception d'un serveur en Go puisque ce langage et l'architecture choisie permettent une performance rapide et efficace. Ce choix permettrait donc de limiter la latence lors des communications avec le serveur et aussi d'avoir une grande proportion des opérations de jeu effectuées côté serveur afin de limiter la mémoire utilisée par le client. Cela permettrait donc d'assurer une latence inférieure à 200 ms. Cette limitation a donc aussi eu un impact au niveau de la conception de l'architecture des clients, puisqu'il était essentiel de ne pas conserver plus que le strict nécessaire localement lors de l'utilisation des différentes fonctionnalités sur les clients.