

**Projet d'évolution d'un logiciel
Document d'architecture logicielle**

Version 1.3

Historique des révisions

Date	Version	Description	Auteur
2021-02-10	1.0	Début du document	Antoine Morcel Étienne Plante Simon Malouin Samuel Poulin Maxime Fecteau Guillaume Beausoleil
2021-04-5	1.1	Révision du diagramme de déploiement	Antoine Morcel Simon Malouin
2021-04-18	1.2	Révision du diagramme de paquetage serveur	Antoine Morcel Maxime Fecteau
2021-04-19	1.3	Révisions des diagrammes de séquence	Etienne Plante
2021-04-19	1.3	Révision des diagrammes de cas d'utilisation Révision du diagramme de paquetage du client léger	Guillaume Beausoleil

Table des matières

1. Introduction	4
2. Objectifs et contraintes architecturaux	4
Objectifs :	4
Sécurité :	4
Maintenabilité :	4
Contraintes :	4
Coûts :	4
Échéancier :	4
Outils de développement :	4
Langages de développement :	4
3. Vue des cas d'utilisation	5
4. Vue logique	Error! Bookmark not defined.
5. Vue des processus	13
6. Vue de déploiement	24
7. Taille et performance	26

Document d'architecture logicielle

1. Introduction

Ce document introduira premièrement nos objectifs et nos contraintes architecturales. Puis, diverses vues UML de notre architecture de projet. Le diagramme de cas d'utilisation, les vues logiques, les diagrammes de séquence, puis le diagramme de déploiement.

2. Objectifs et contraintes architecturaux

Objectifs :

Sécurité :

Tout ce qui est de la gestion de compte doit pouvoir être fait de manière sécuritaire, ainsi que l'envoi de messages privés qui doit être encrypté.

Maintenabilité :

Implique l'utilisation de linting pour avoir un code qui suit des standards déterminés. Ceci assure une certaine qualité de code de base. Implique aussi une structure de classe et d'interface qui sont facilement maintenables et réutilisables.

Contraintes :

Coûts :

Nous voulons minimiser les coûts, donc pour le serveur et la base de données nous les hébergeons sur un serveur personnel d'un membre de l'équipe, ainsi nous sommes indépendants de services cloud comme AWS ou Azure.

Échéancier :

Il faut remettre un prototype pour le 19 février qui supporte une plateforme de messagerie entre le client lourd et léger ainsi que le serveur. Ensuite, nous avons jusqu'au 19 avril pour remettre le projet fini et fonctionnel.

Outils de développement :

Pour le serveur nous utilisons NodeJs et Express. Le serveur utilise un API Http pour la gestion de compte sécurisé et Socket.io pour tout ce qui est logique de jeu et messages en partie. Pour notre base de données, nous utilisons MongoDB. Pour le client lourd nous utilisons Angular ainsi que Electron pour produire une application native. Pour le client léger nous utilisons Android Studio.

Langages de développement :

Le serveur et le client lourd sont développés en Typescript. Le client léger est développé en Kotlin

3. Vue des cas d'utilisation

Diagramme 1 : Cas d'utilisation général
Precondition: L'utilisateur est déjà connecté.

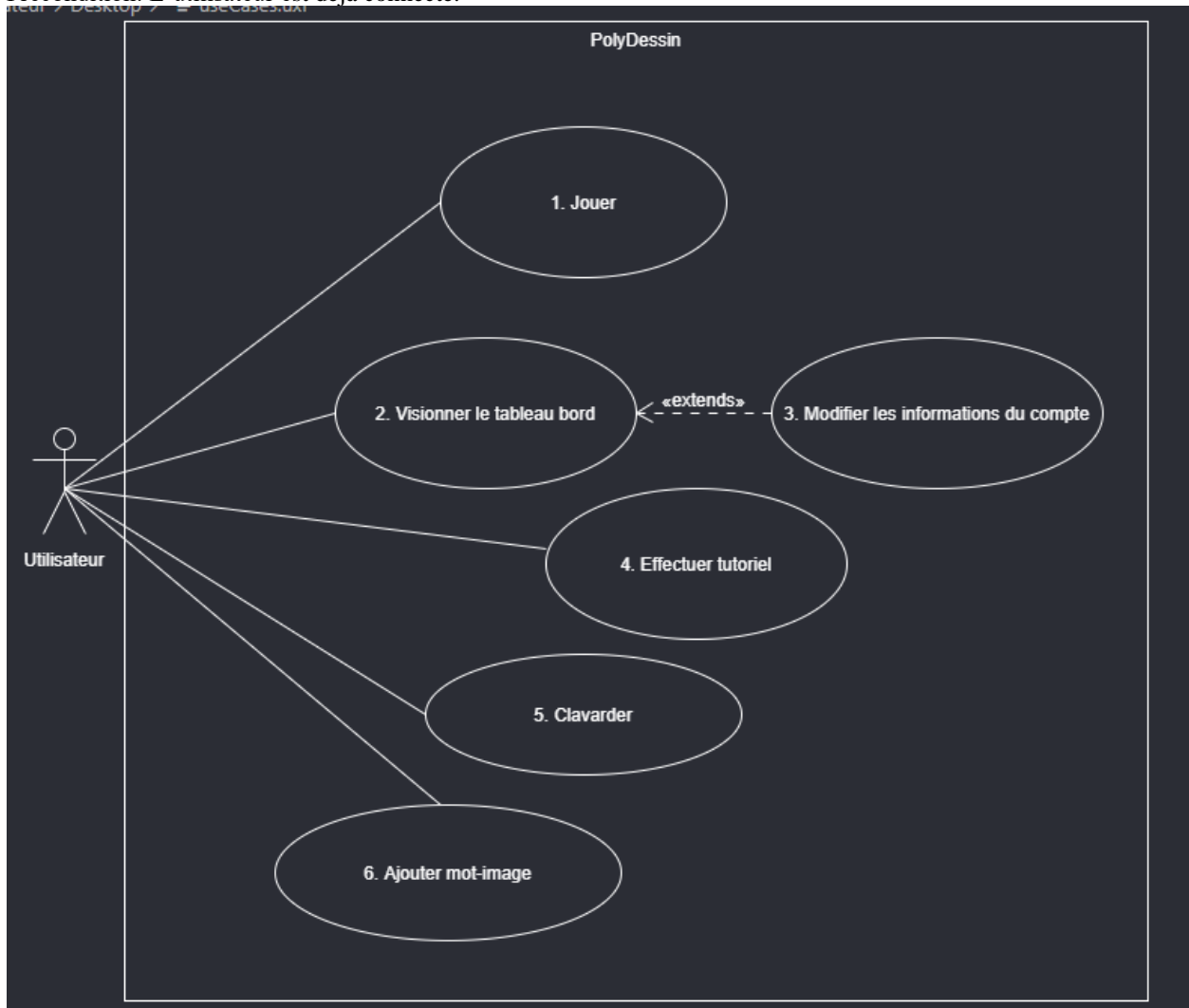


Diagramme 2: Cas d'utilisation de connection

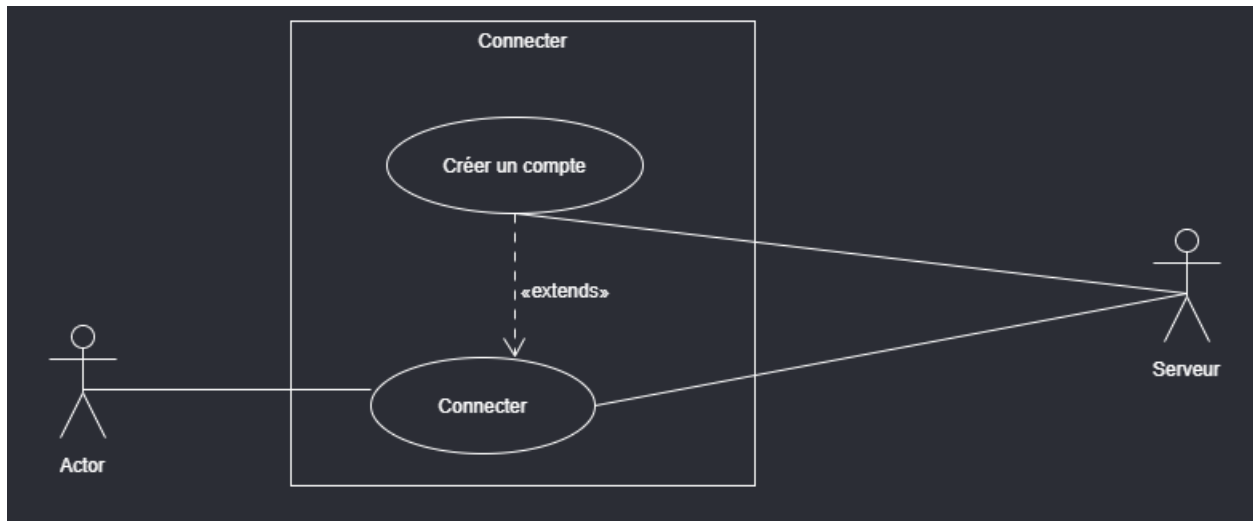


Diagramme 3: Cas d'utilisation jouer

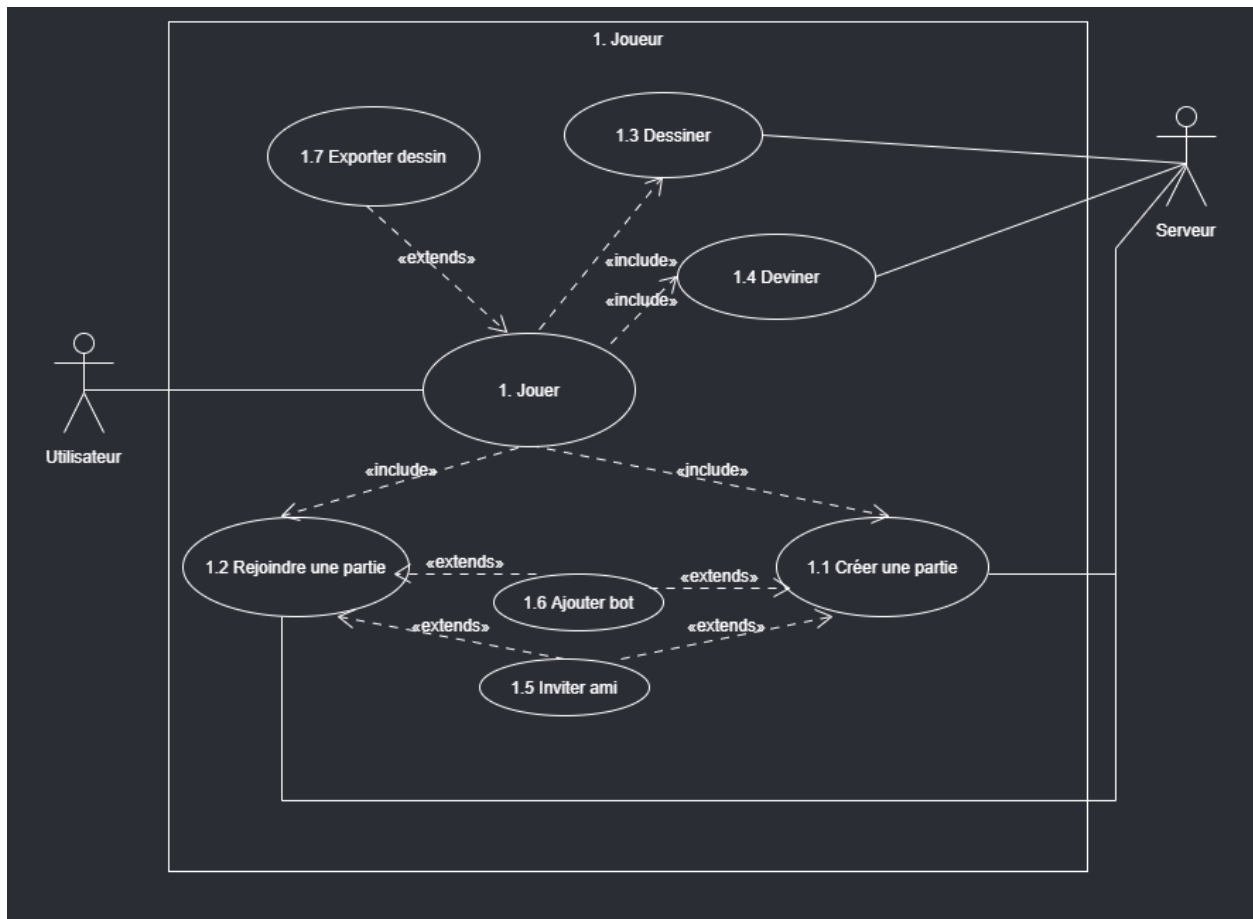
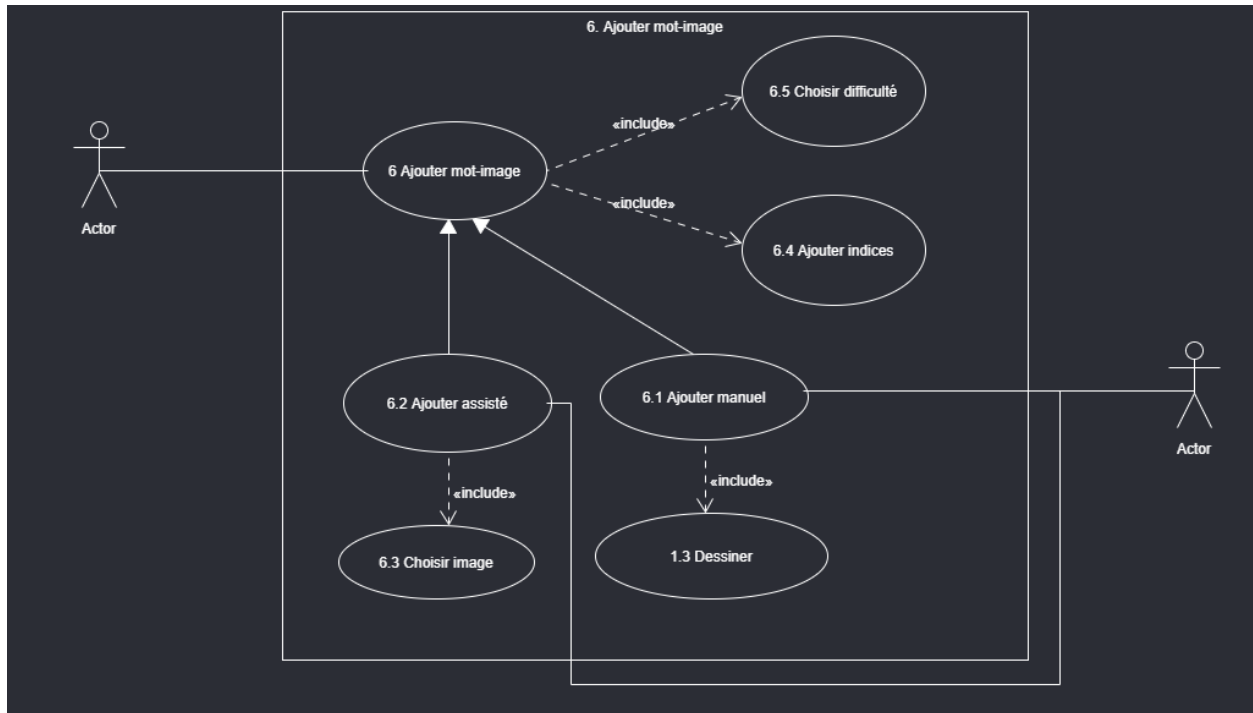
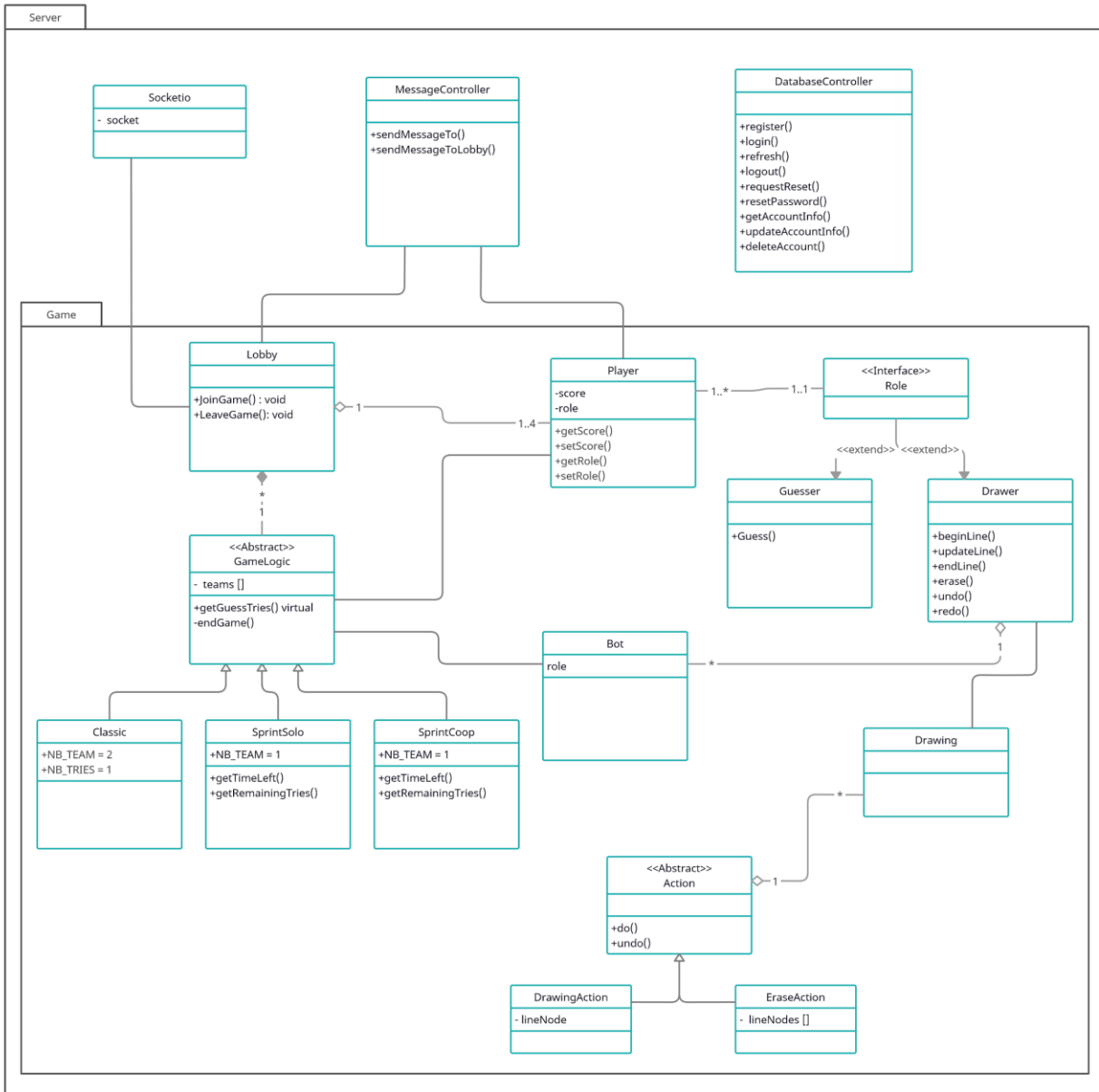


Diagramme 4: Cas d'utilisation d'ajout mot-image



4. Vue logique

Server
Paquetage contenant le serveur. S'occupe de la logique de jeu et de recevoir et traiter les actions des clients lourds et légers. Le serveur contient deux autres paquetages importants :
<div>API HTTP</div>
Paquetage contenant les points d'accès sécurisé pour la gestion de compte.
<div>Game</div>
Paquetage contenant la logique du jeu



ClientLourd

Paquetage contenant l'application lourde pour ordinateur. Regroupe l'ensemble des fonctionnalités disponibles à l'utilisateur sur cette plateforme.

Login

Contient l'interface de connexion de l'utilisateur ainsi que les méthodes associées pour la communication avec le serveur.

Chat

Contient toutes les fonctionnalités d'échange textuel entre les utilisateurs ainsi qu'une partie des

fonctionnalités servant à deviner le mot dessiné.

Home

Contient l'interface du menu principal ainsi que la logique permettant au joueur d'accéder au reste de l'application.

Game

Contient la surface de dessin, la toolbar et la logique qui permet au joueur de jouer en ligne.

Surface de dessin

Contient la surface permettant au joueur de voir les traits tracés à l'aide des outils de la toolbar ainsi que la logique d'export et d'import de dessins.

Toolbar

Contient la toolbar qui permet au joueur de sélectionner un outil à utiliser sur la surface de dessin.

APIService

Contient la logique qui permet au client lourd de faire des requêtes http lorsqu'il requiert des données stockées sur la base de données.

SocketService

Contient la logique qui permet au client lourd de se synchroniser en temps réel avec la partie dans laquelle il participe ainsi qu'avec le chat et ses messages.

GameService

Contient la logique qui permet à l'application de rester synchronisée avec l'état de la partie donné par le serveur.

ChatService

Contient la logique qui permet au chat de s'adapter selon l'état de la partie et le mode d'intégration du

chat (fenêtré ou intégré)

UserService

Contient la logique d'authentification.

LocalSaveService

Contient la logique de sauvegarde des informations de compte ainsi que des "tokens" d'authentification afin d'éviter d'avoir besoin de recharger la page de login chaque fois que l'application est initialisée.

Tutoriel

Contient l'overlay et la logique qui permet aux pages ayant un tutoriel de guider le joueur de façon interactive à travers les fonctionnalités du client lourd.

Fin de partie

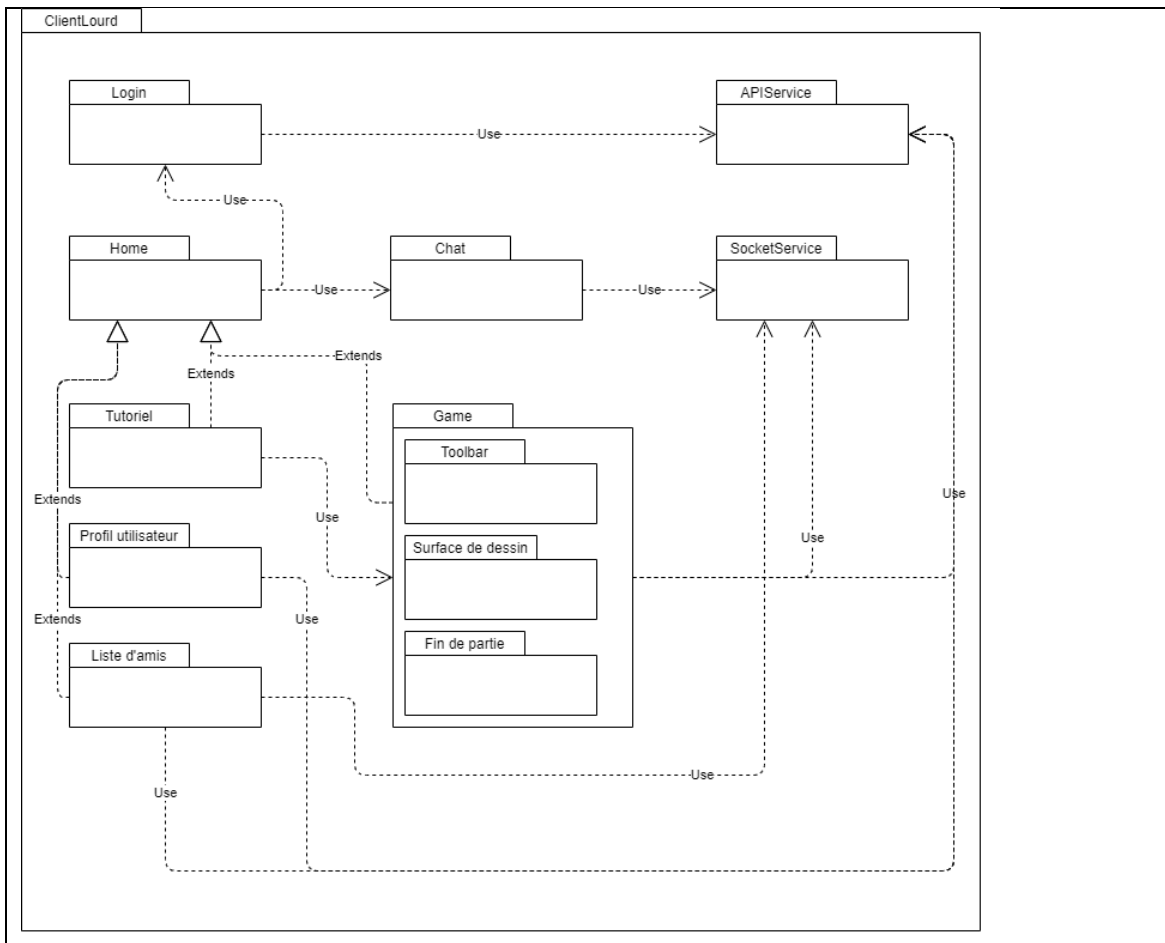
Contient l'écran de fin de partie ainsi que la logique de génération du gif récapitulatif du dessin.

Liste d'amis

Permet de gérer l'ajout d'amis à la liste d'amis et de consulter celle-ci.

Profil utilisateur

Permet à l'utilisateur de créer son profil, de le consulter et de le modifier.



ClientLeger

Paquetage contenant l'application légère pour tablette Android.

Connexion

Responsable de la connexion de l'utilisateur.

chat

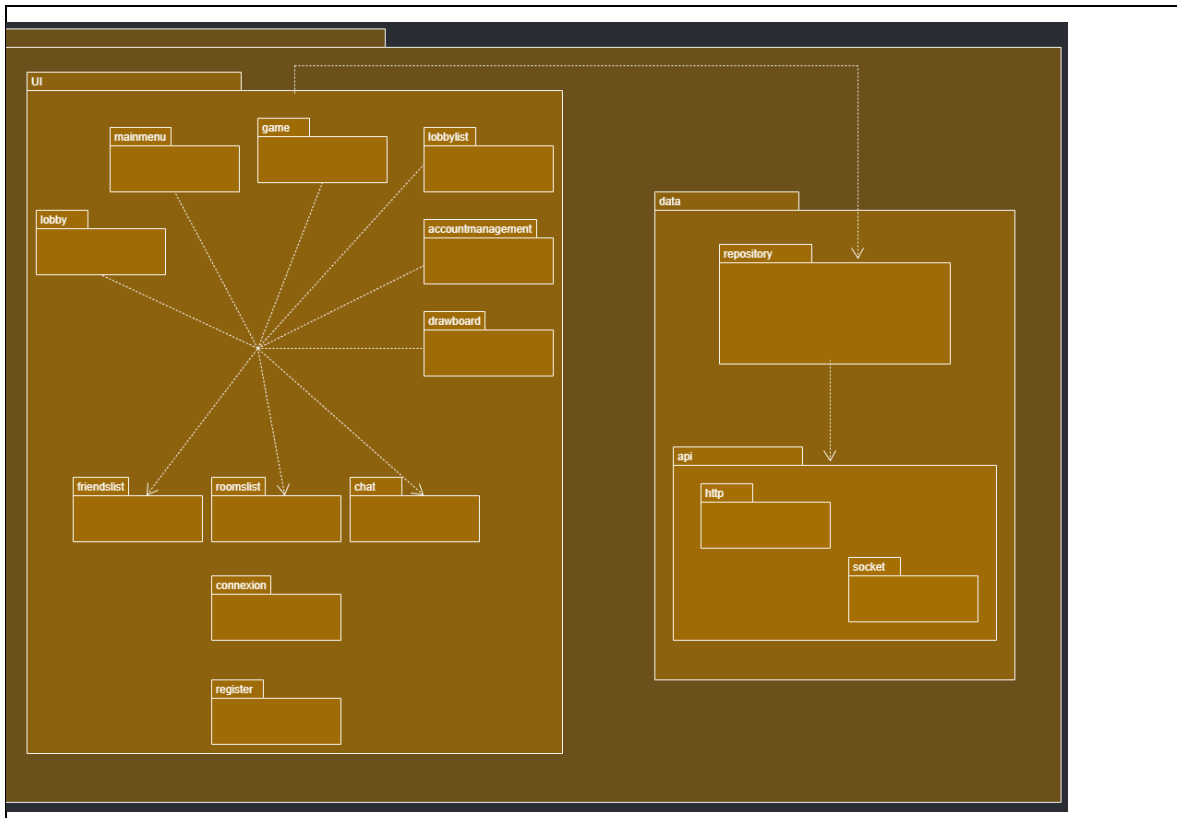
Responsable du clavardage avec canal de discussion.

mainMenu

Responsable du menu principal affichant les actions possibles pour l'utilisateur.

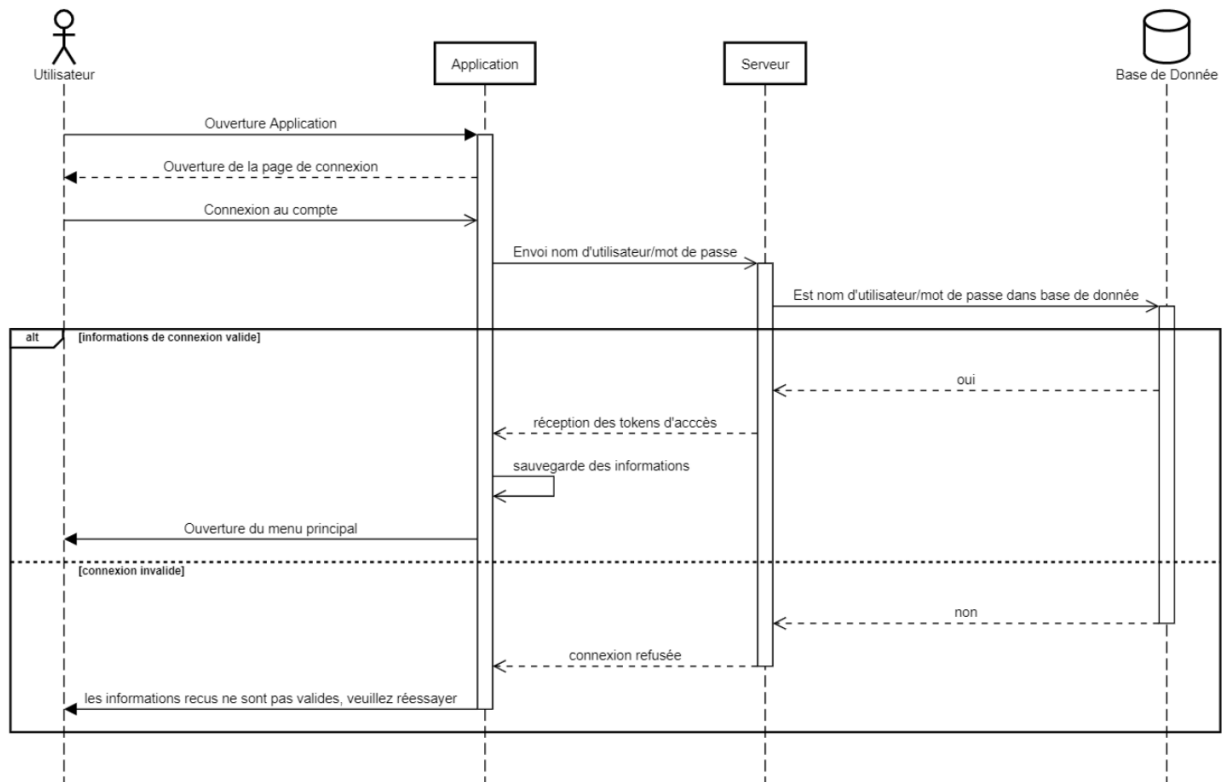
lobby

Responsable de la salle de parties disponible à l'utilisateur.
game
Responsable des paramètres, de la logique et de l'exécution des différents types de parties.
drawingboard
Responsable de l'espace de dessin avec les outils.
lobbylist
Responsable du menu affichant les lobbies disponibles.
friendlist
Responsable de l'affichage de liste d'ami et des interactions avec ceux-ci.
data
Responsable de la gestion des données et des interactions avec le serveur.
repository
Responsable des interactions entre la couche UI et les données.
api
Responsable des interactions avec le serveur

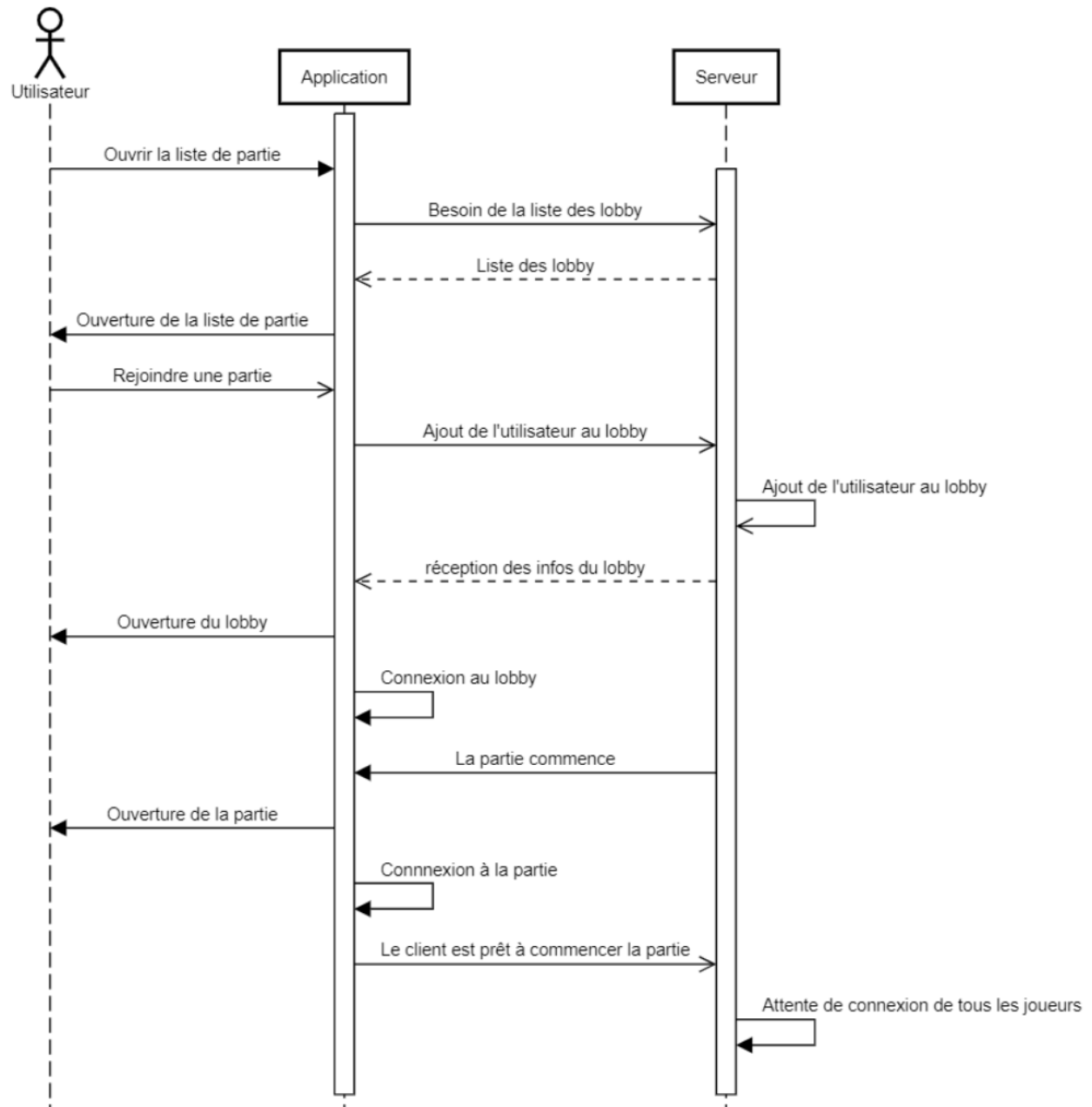


5. Vue des processus

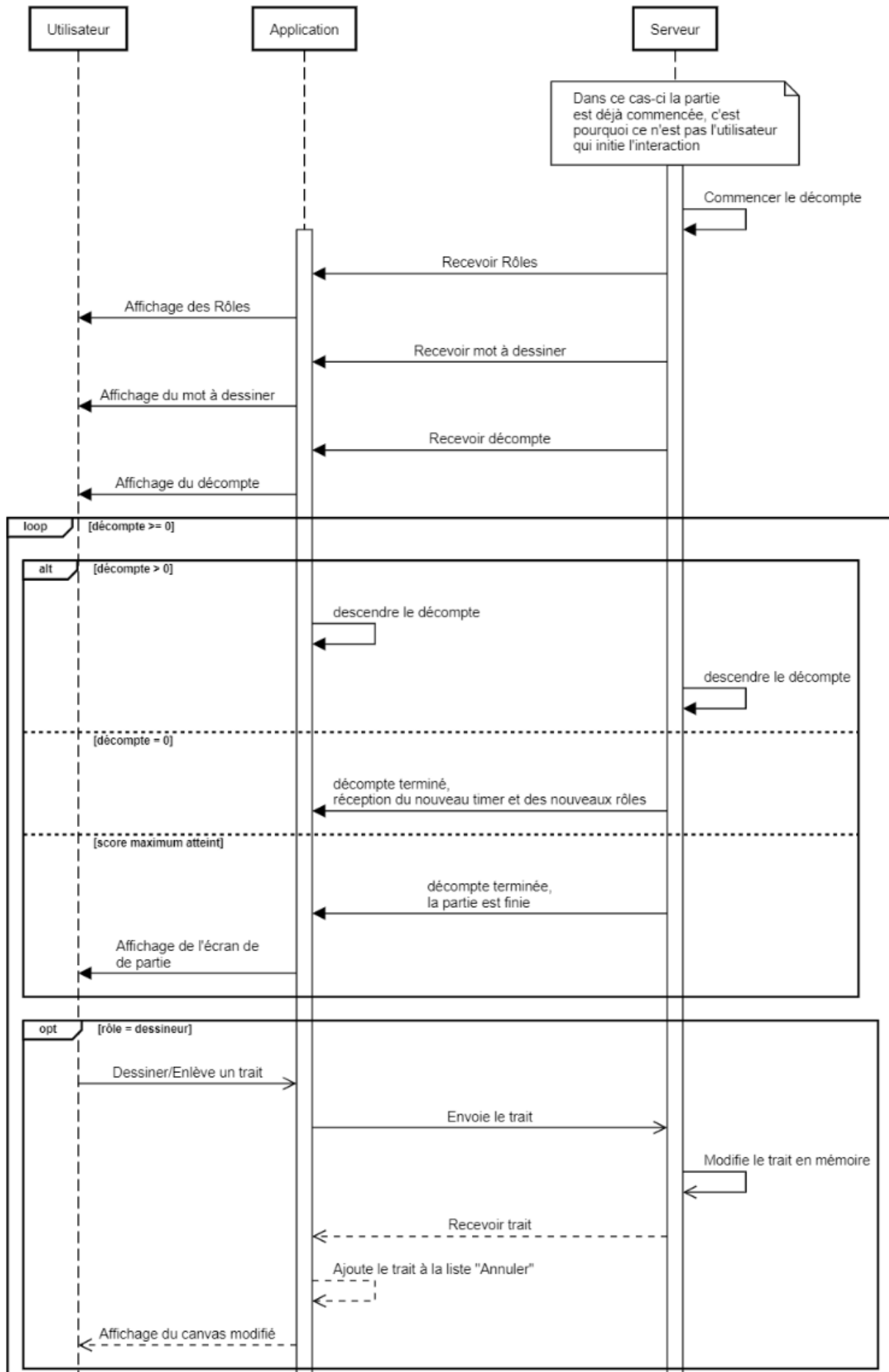
Connexion à un compte



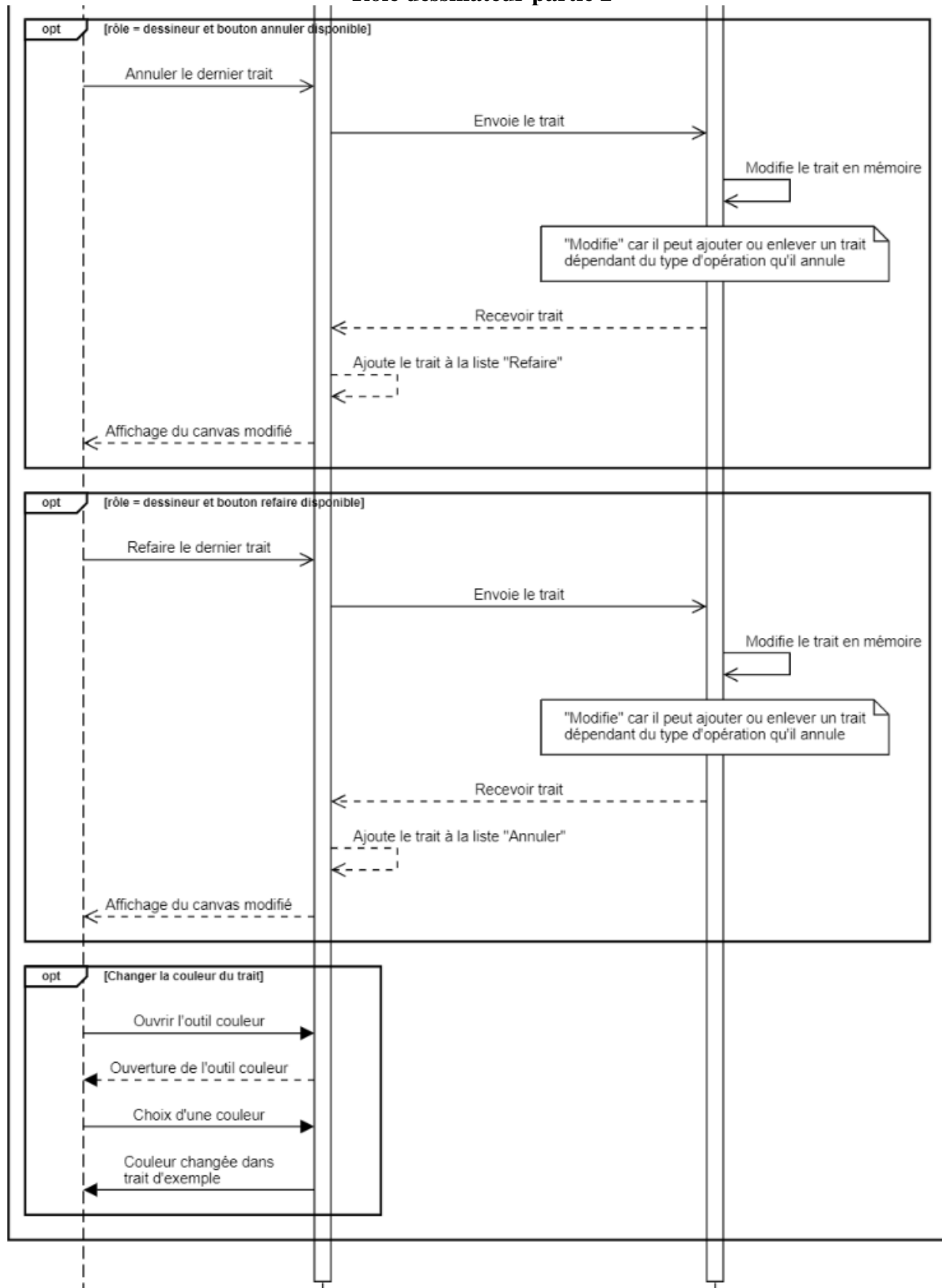
Connexion à une partie



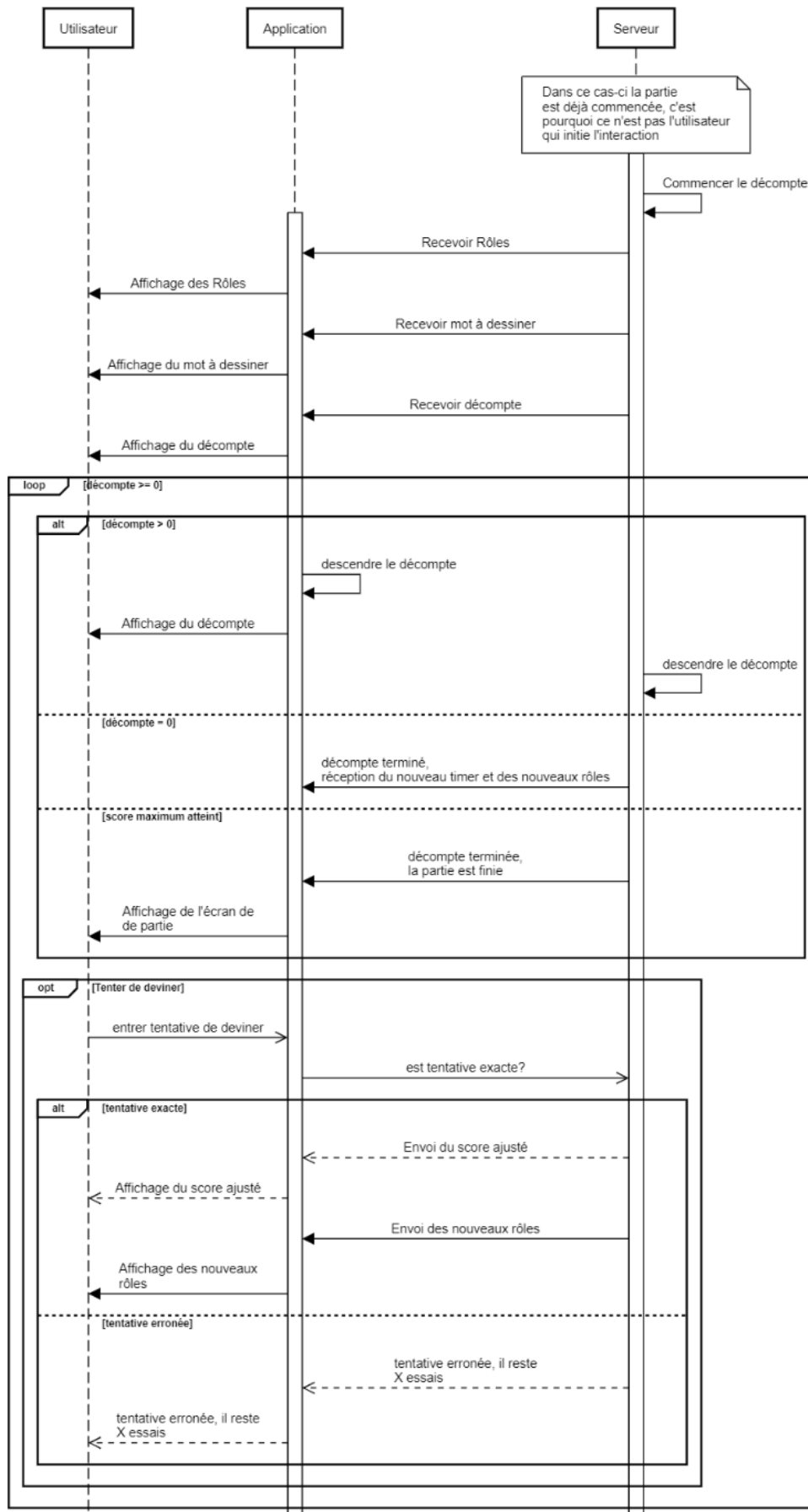
Rôle dessinateur



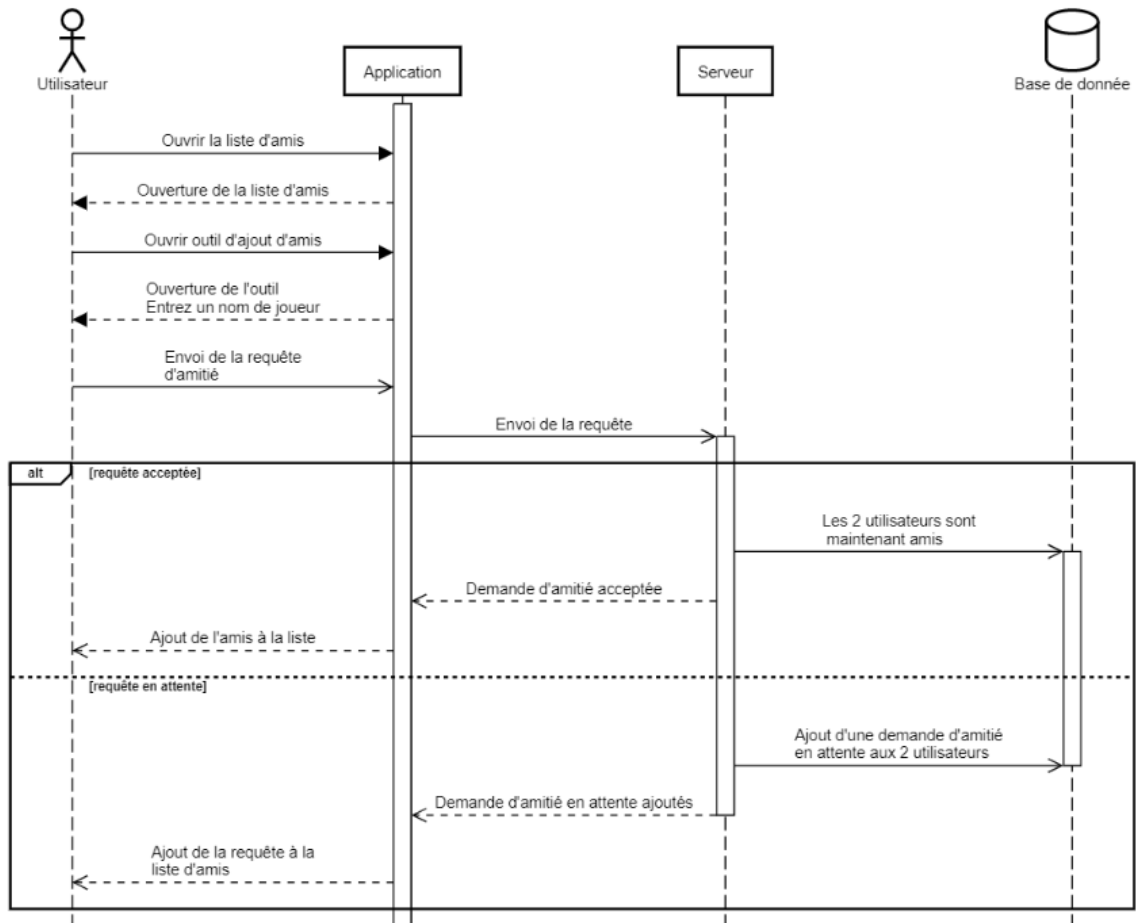
Rôle dessinateur partie 2



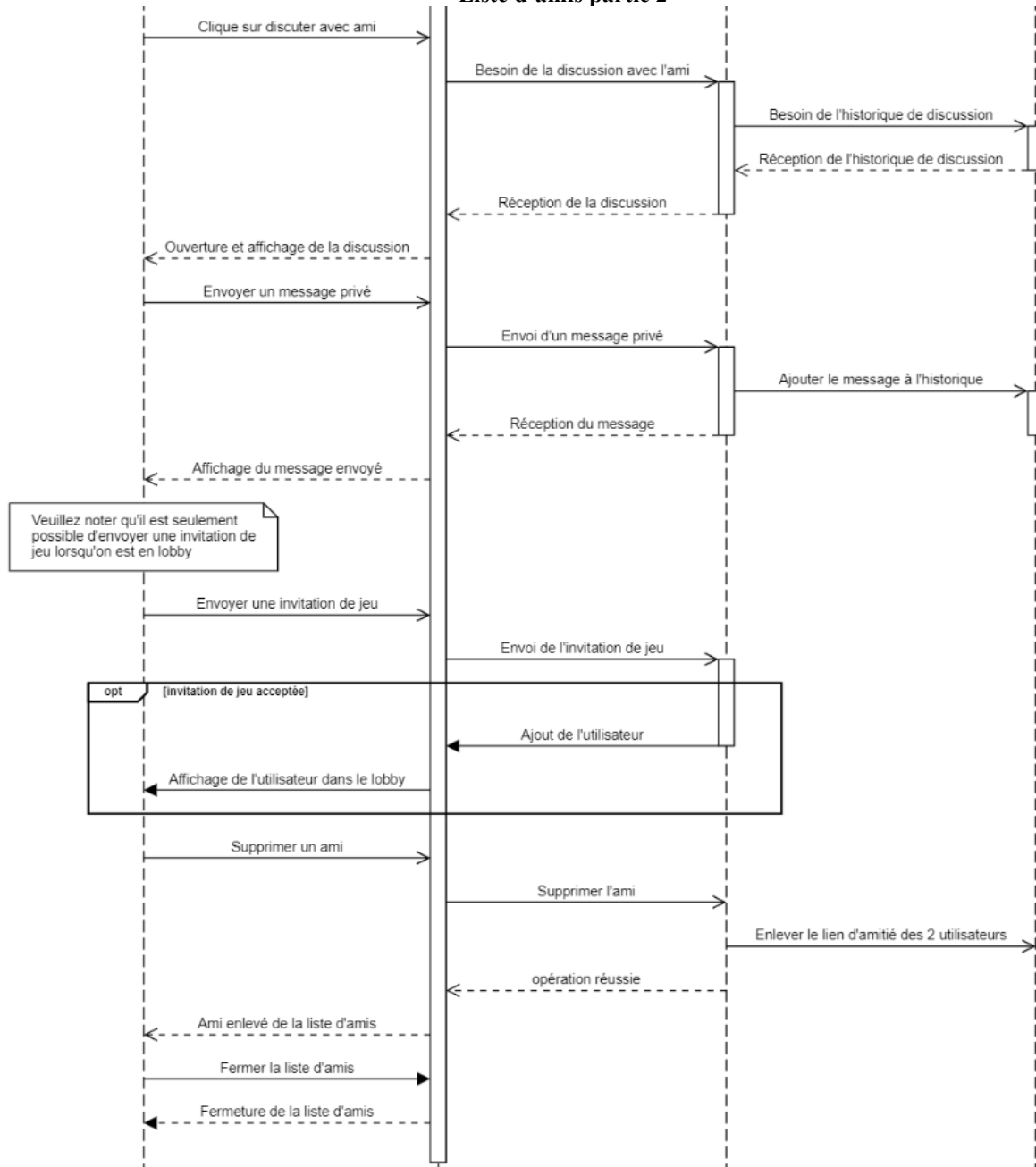
Rôle devineur



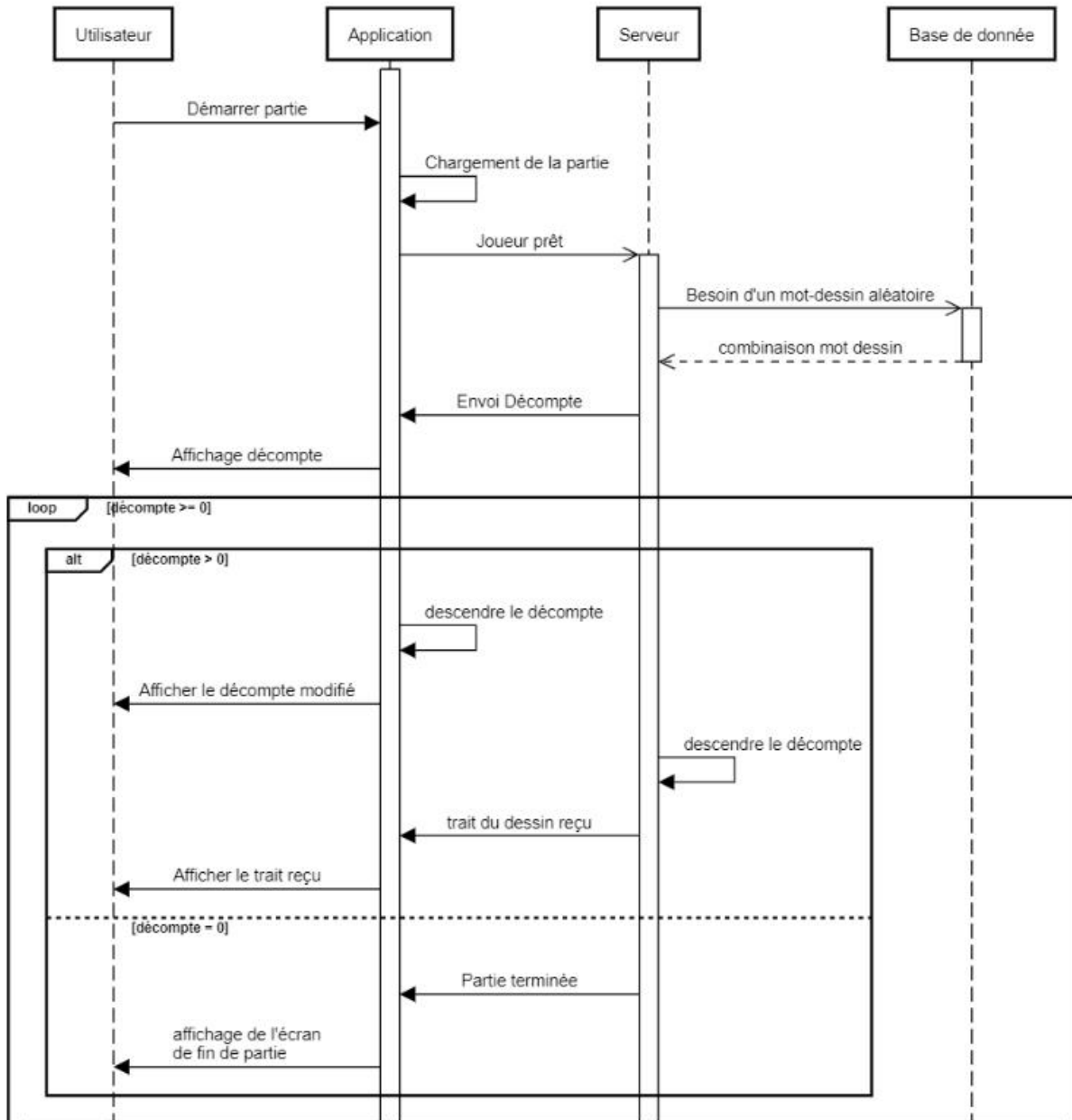
Liste d'amis



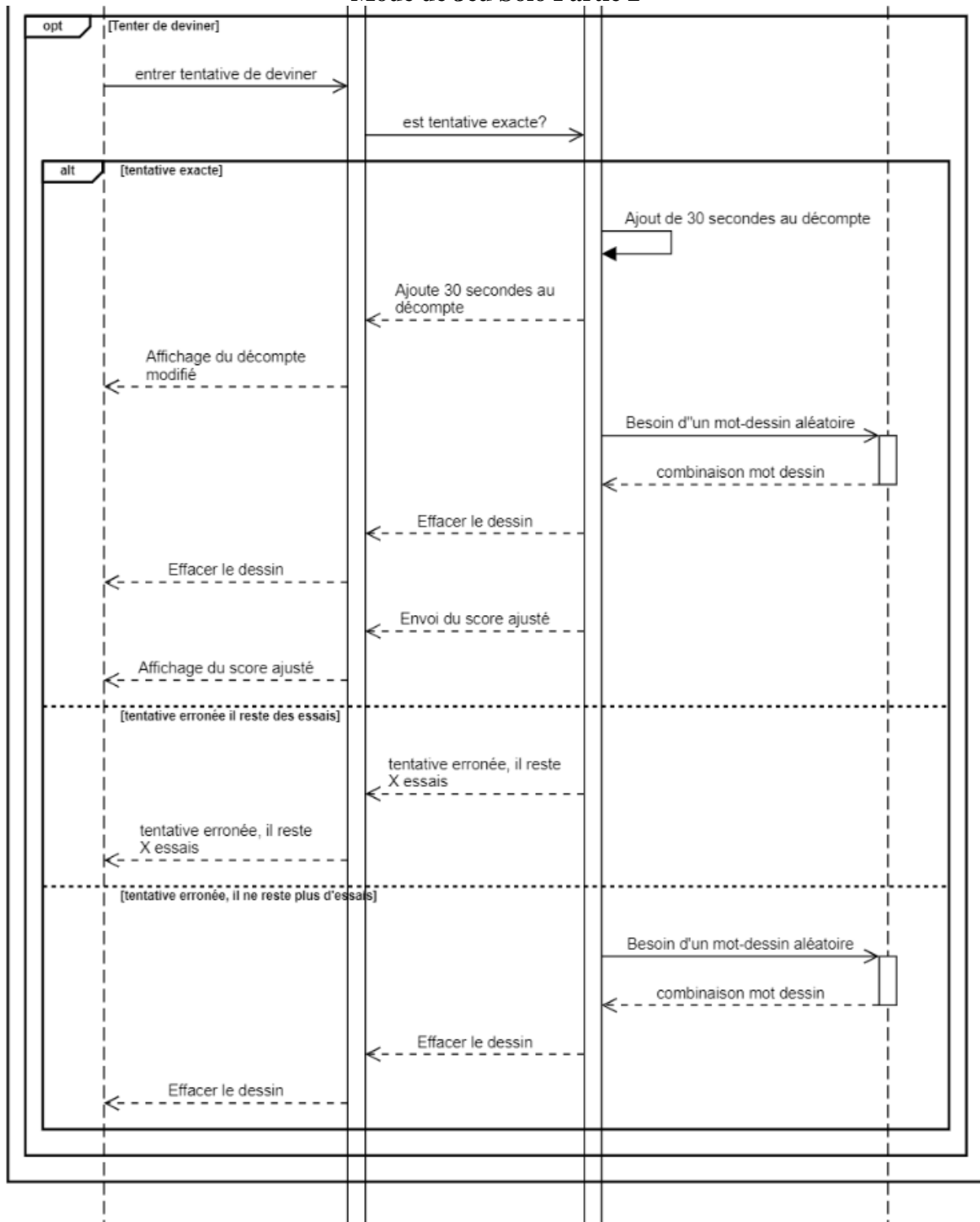
Liste d'amis partie 2



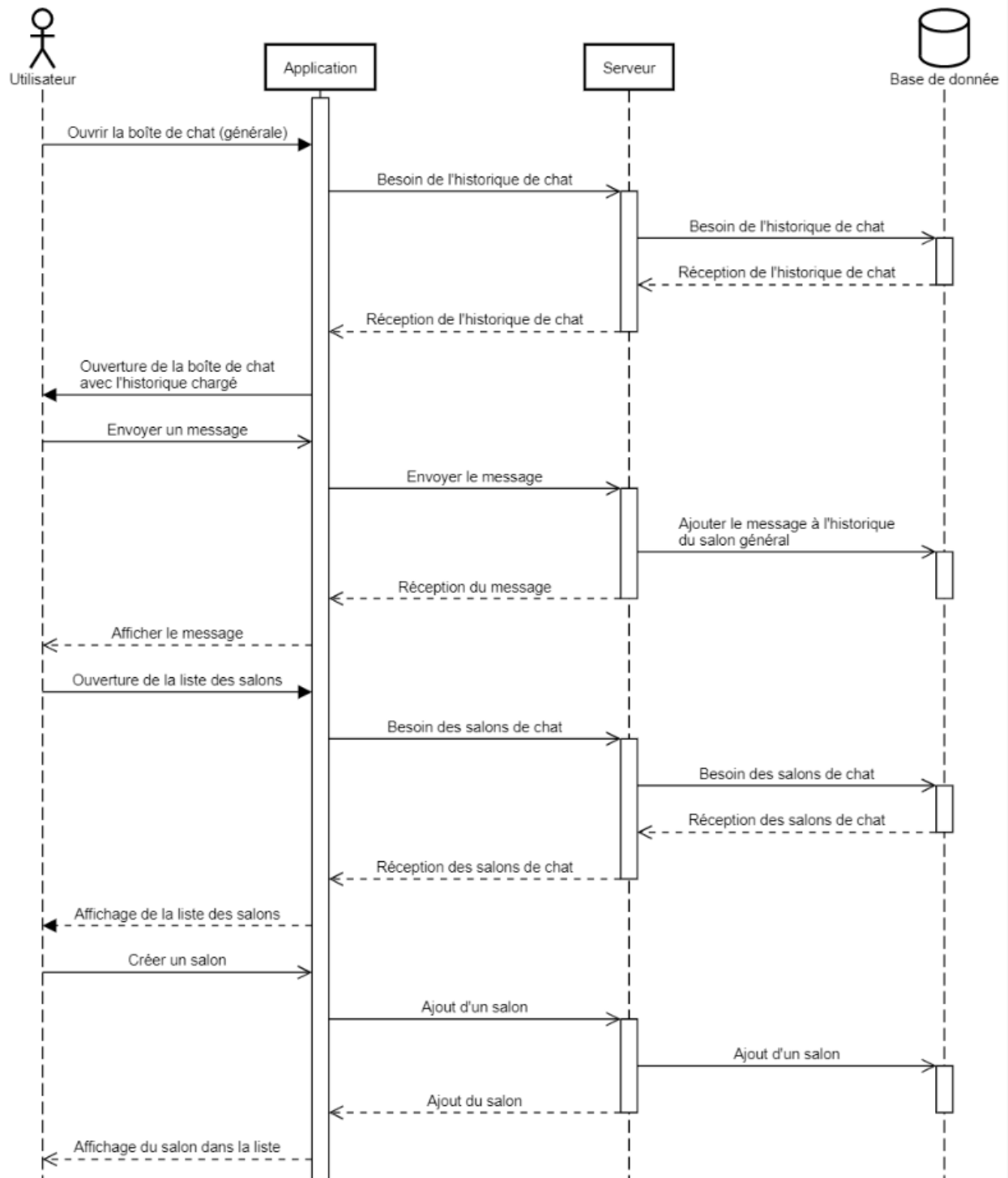
Mode de jeu solo



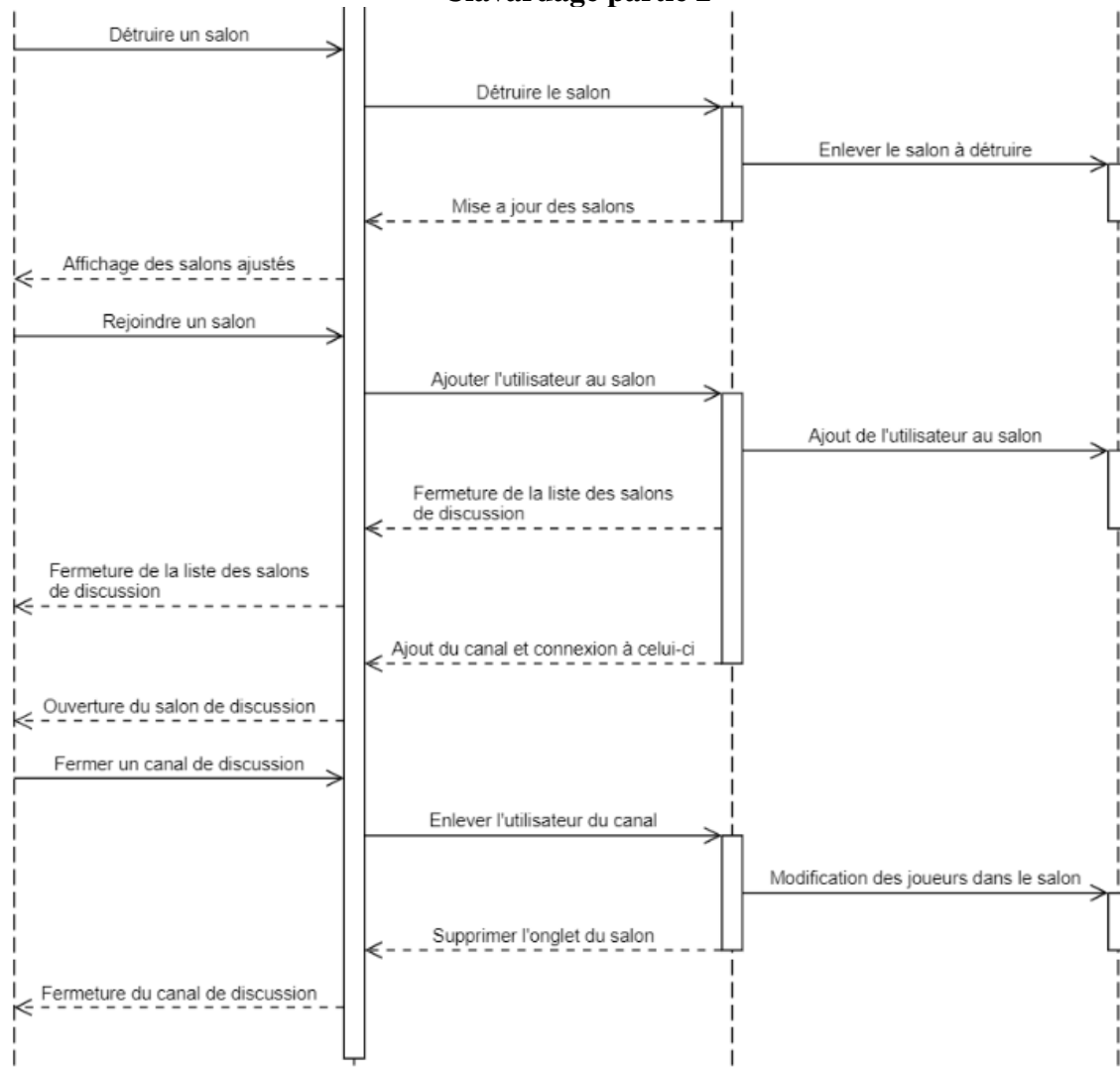
Mode de Jeu Solo Partie 2



Clavardage

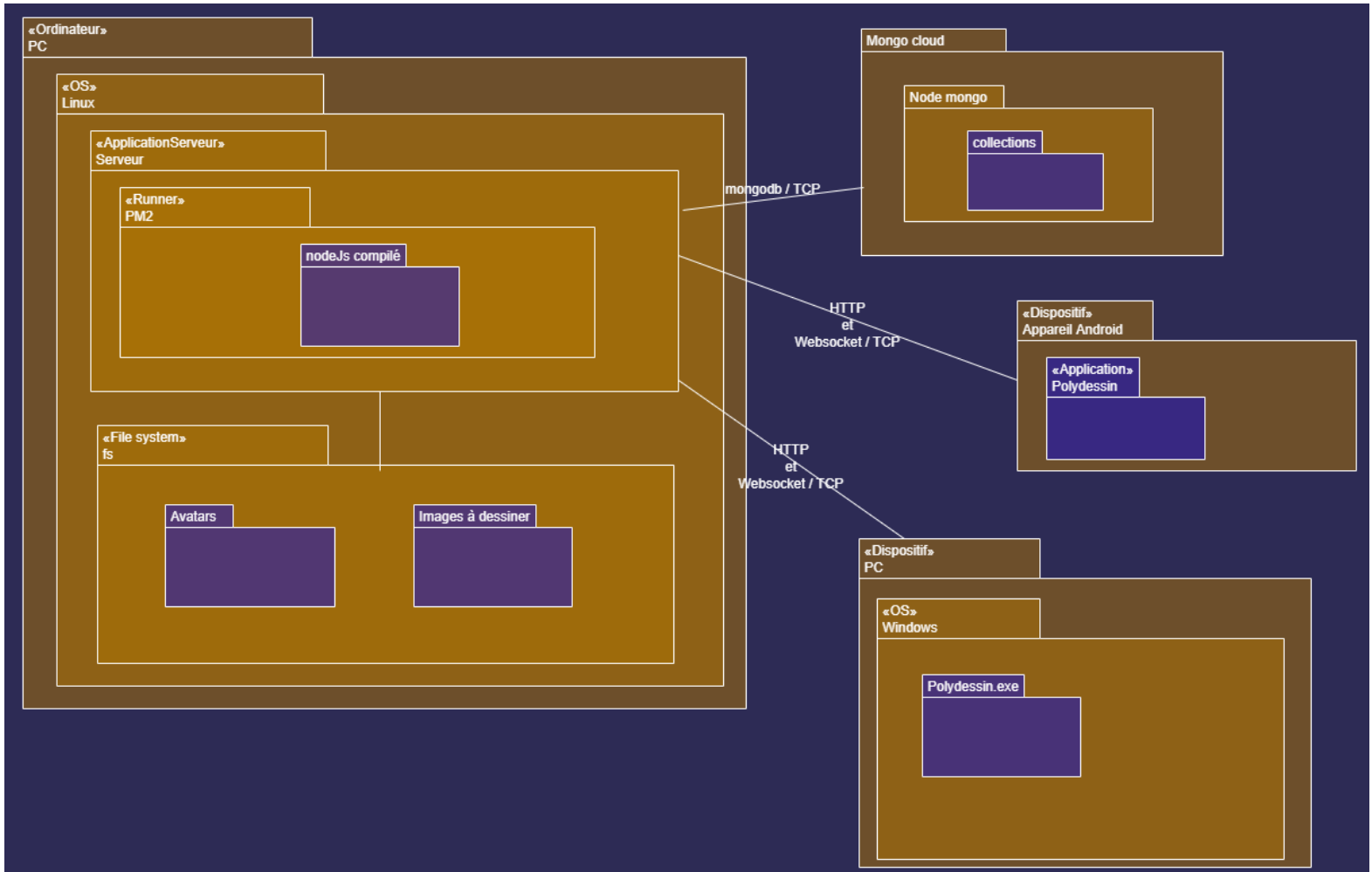


Clavardage partie 2



6. Vue de déploiement

Diagramme x : Vue de déploiement de Polydessin



7. Taille et performance

Avec l'utilisation de la librairie Socket.io et le nombre d'utilisateurs attendus assez bas, nous doutons fortement que nous allons rencontrer des problèmes de communications. Dans le pire cas, nous pouvons mettre à niveau la librairie Socket.io pour pouvoir supporter plus de joueurs. Par contre, on s'attend à avoir certaines capacités minimales pour le serveur et les ordinateurs des clients.

On s'attend au minimum d'avoir les métriques suivantes de disponible pour le serveur:

Une utilisation "idle" de ~2% pour le processeur du serveur avec une vitesse d'au moins 1 GHz single-core.
Au moins 100 MB de mémoire vive disponible.
Un temps moyen de 30 ms pour les requêtes HTTP avec 95% des requêtes prenant moins de 71 ms.

Pour l'application électron, on s'attend d'avoir les métriques suivantes de disponible :

Une utilisation d'environ ~5% pour le processeur du client avec une vitesse d'au moins 2 GHz single-core.
Au moins 500 MB de mémoire vive disponible.
Au moins 320 MB de disponible sur un disque dur pour l'application électron.