Équipe 204

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

Version 1.0

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

Table des matières

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.

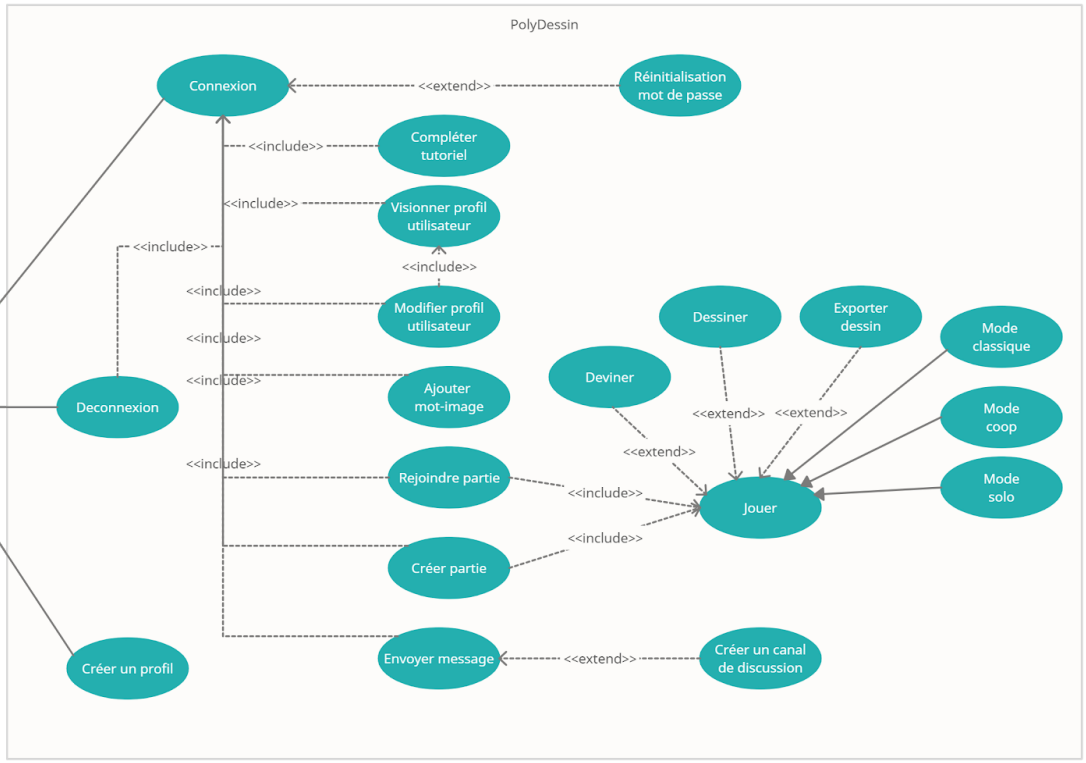
Outils de développement :

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

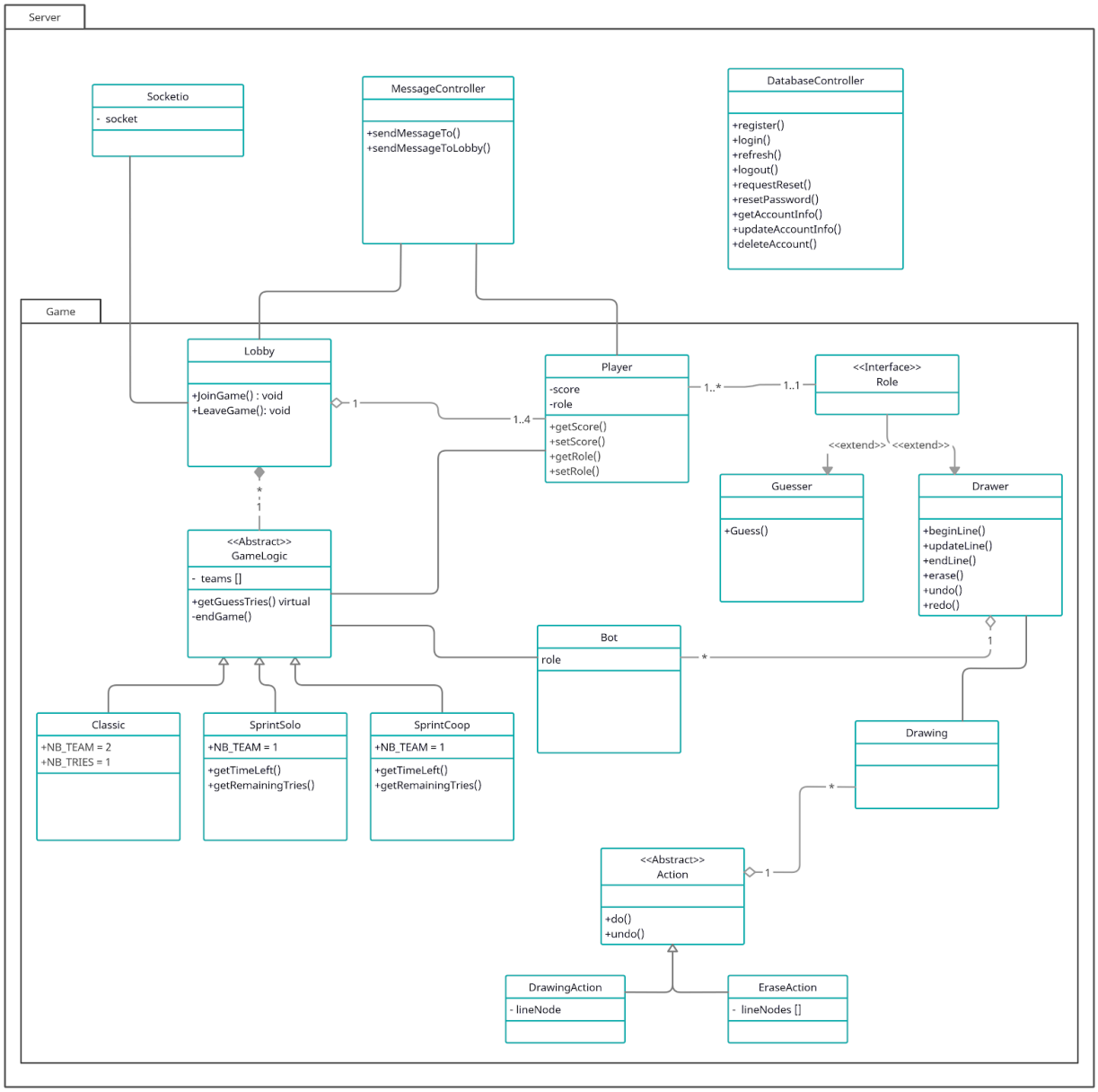


4. Vue logique

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 trois autres paquetages importants :

API HTTP
Paquetage contenant les points d'accès sécurisé pour la gestion de compte.

Game
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.

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

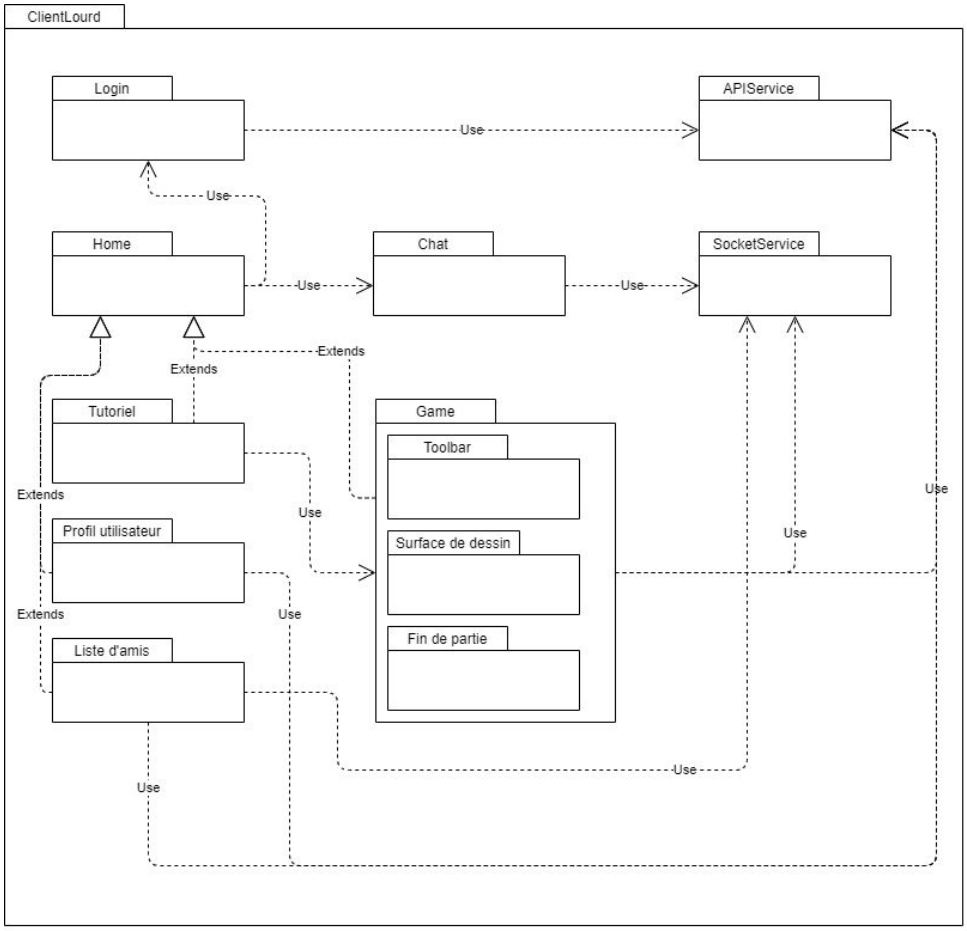
Contient l'écran de fin de partie ainsi que la logique de génération du gif récapitulatif du dessin.

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.à

login

Responsable pour la connexion de l'utilisateur, réinitialisation de mot de passe et la création de compte.

chat
Responsable du clavardage avec canal de discussion.

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

tutoriel
Responsable du tutoriel 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.

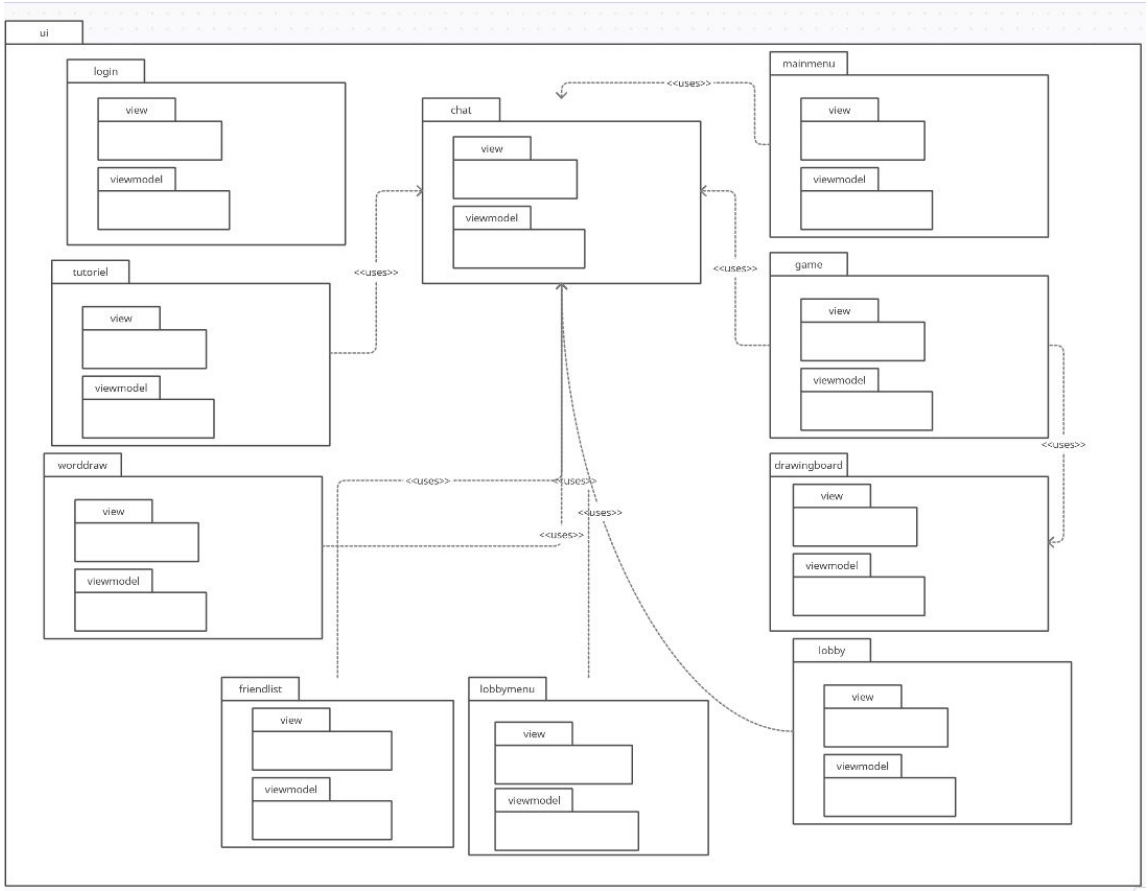
drawingboard
Responsable de l'espace de dessin

lobbymenu
Responsable du menu affichant les lobbies disponibles

friendlist
Responsable de l'affichage de liste d'ami et des interactions avec ceux-ci

worddraw

Responsable de l'ajout des mots-images pour les joueurs virtuelles

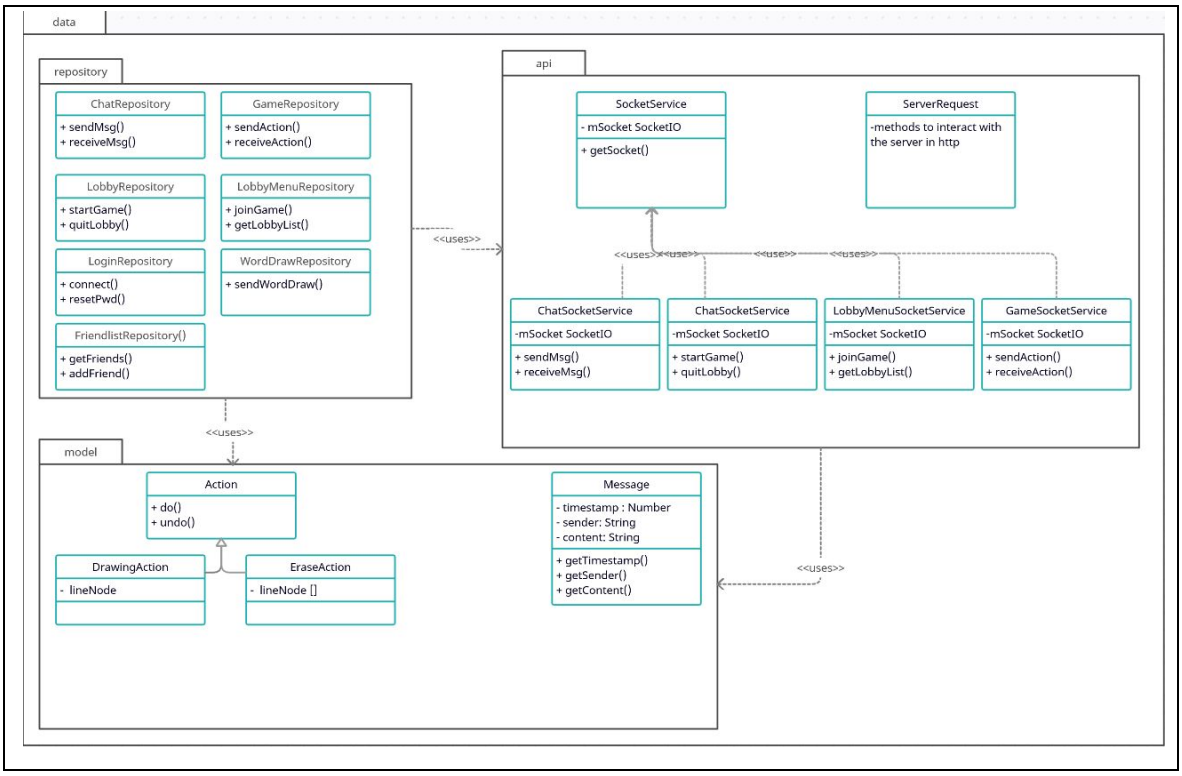


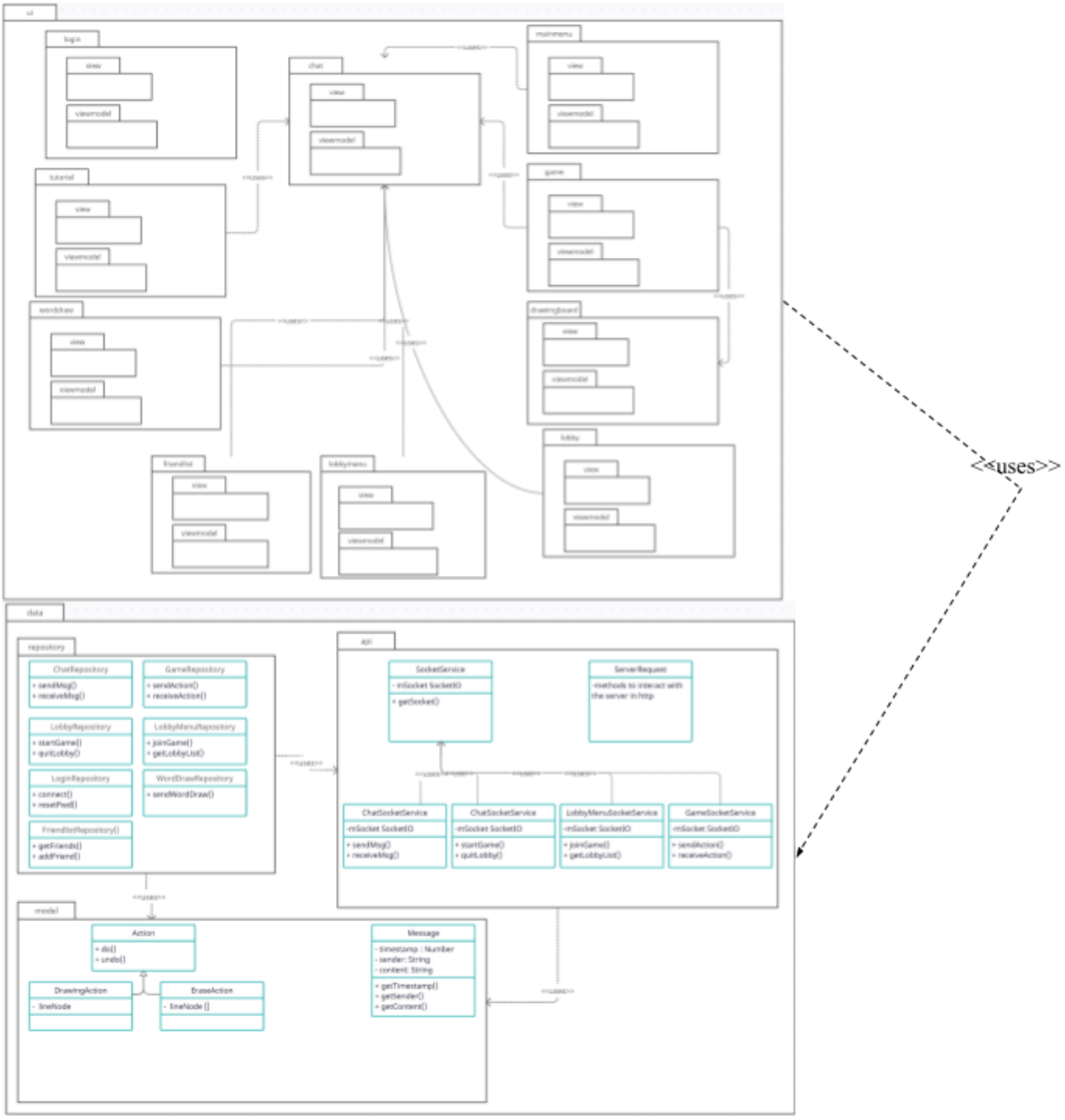
data
Responsable de la gestion des données et les interactions avec le serveur

repository
Responsable des interactions entre la couche UI et les données.

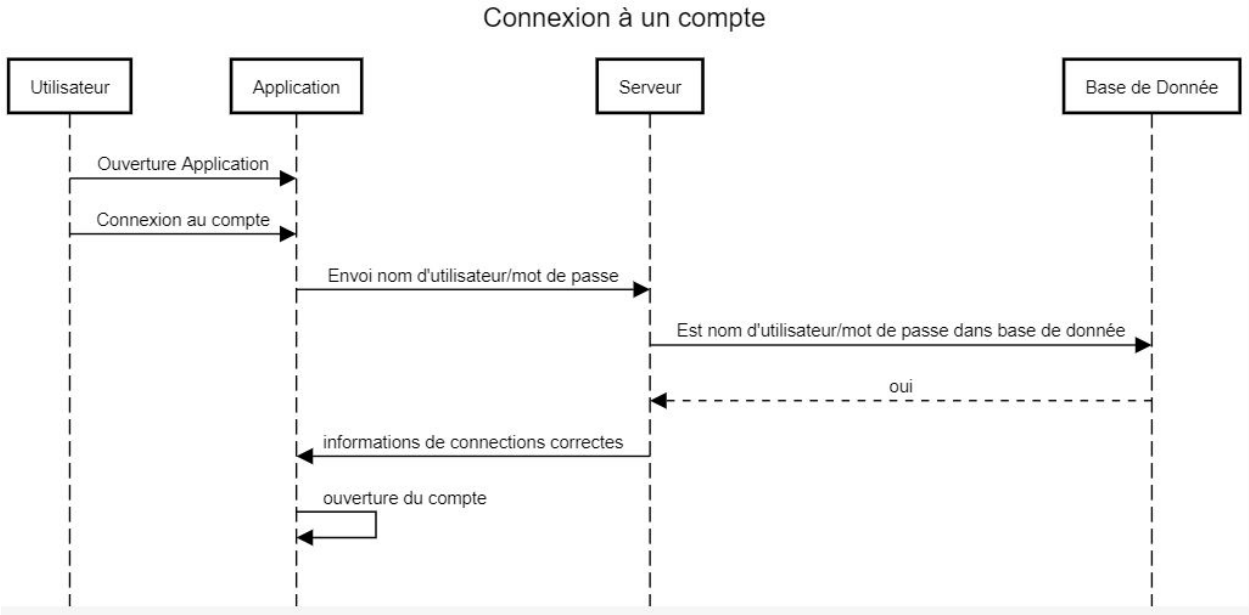
api
Responsable des interactions avec le serveur

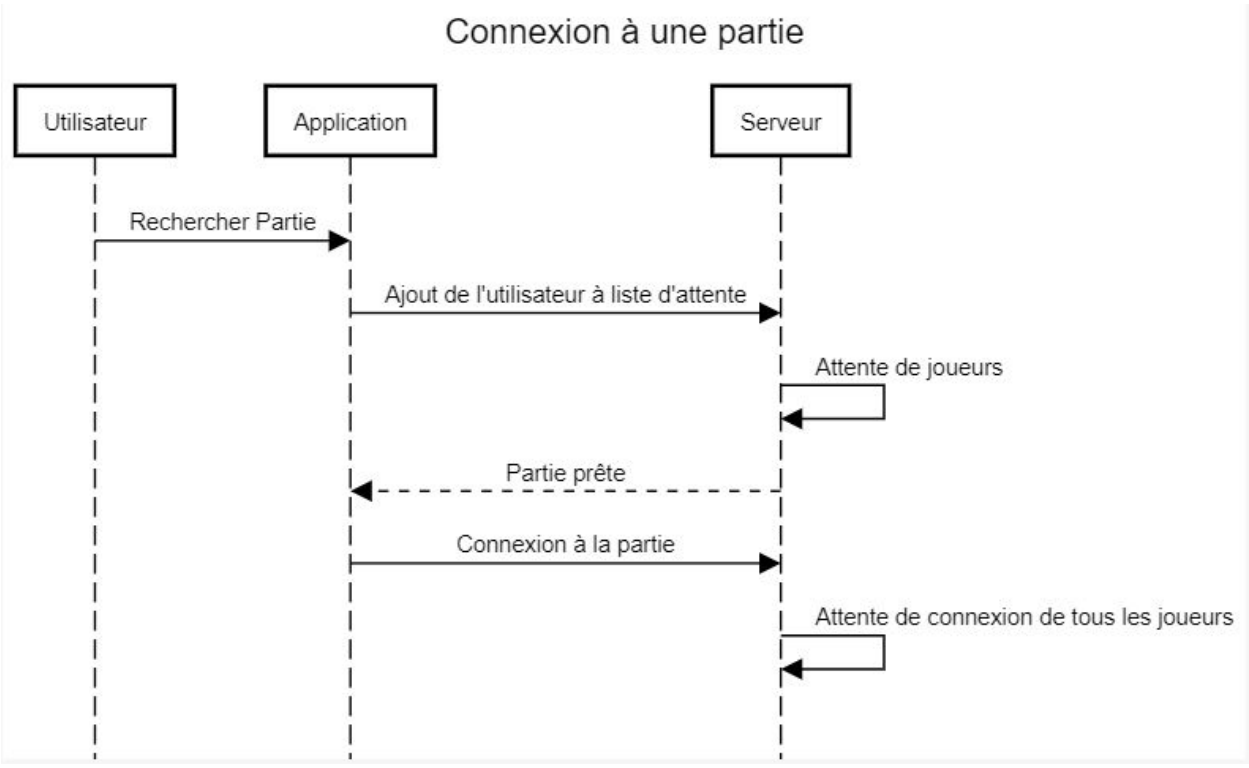
model
Responsable de la représentation des données.

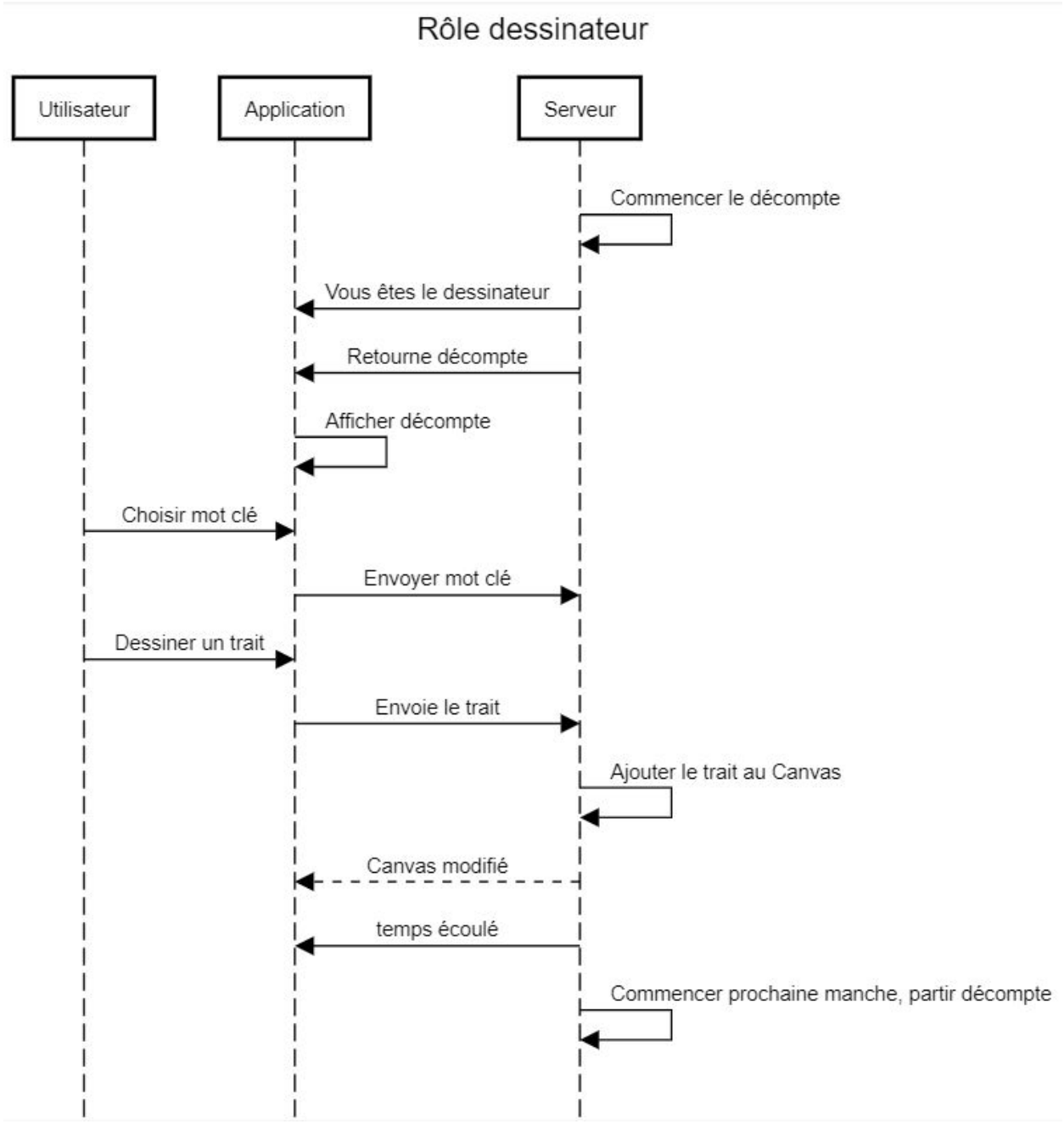

data
repository
ChatRepository
+ sendMsg()
+ receiveMsg()
GameRepository
+ sendAction()
+ receiveAction()
LobbyRepository
+ startGame()
+ quitLobby()
LobbyMenuRepository
+ joinGame()
+ getLobbyList()
LoginRepository
+ connect()
+ resetPwd()
WordDrawRepository
+ sendWordDraw()
FriendlistRepository()
+ getFriends()
+ addFriend()
<<uses>>
api
SocketService
- mSocket SocketIO
+ getSocket()
ServerRequest
-methods to interact with the server in http
<<uses>>
<<uses>>
<<uses>>
<<uses>>
ChatSocketService
-mSocket SocketIO
+ sendMsg()
+ receiveMsg()
ChatSocketService
-mSocket SocketIO
+ startGame()
+ quitLobby()
LobbyMenuSocketService
-mSocket SocketIO
+ joinGame()
+ getLobbyList()
GameSocketService
-mSocket SocketIO
+ sendAction()
+ receiveAction()
<<uses>>
model
Action
+ do()
+ undo()
DrawingAction
- lineNode
EraseAction
- lineNode []
Message
- timestamp : Number
- sender: String
- content: String
+ getTimestamp()
+ getSender()
+ getContent()
<<uses>>

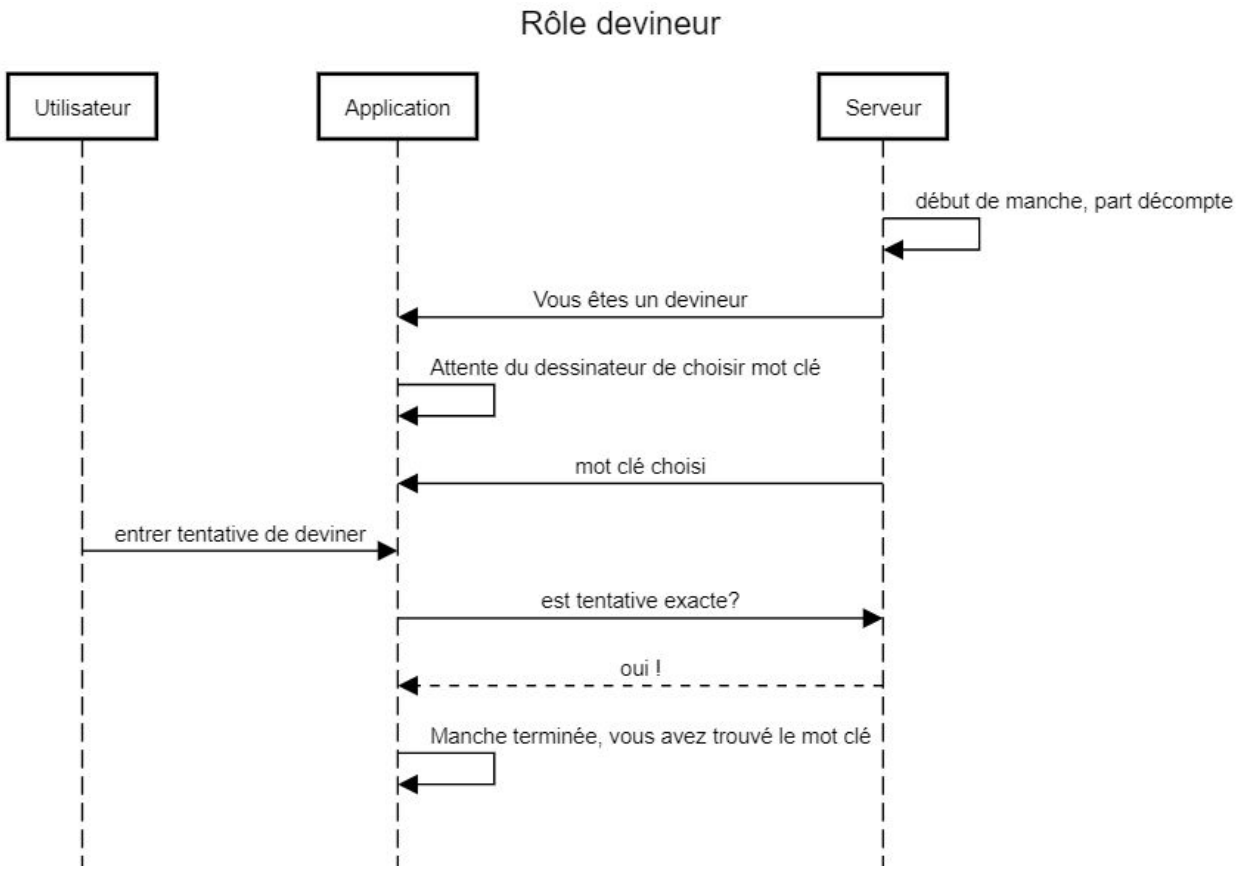

ui
login
view
viewmodel
chat
view
viewmodel
mainmenu
view
viewmodel
tutorial
view
viewmodel
game
view
viewmodel
worddraw
view
viewmodel
drawingboard
view
viewmodel
lobby
view
viewmodel
friendlist
view
viewmodel
lobbymenu
view
viewmodel
<<uses>>
data
repository
ChatRepository
+ sendMsg()
+ receiveMsg()
GameRepository
+ sendAction()
+ receiveAction()
LobbyRepository
+ startGame()
+ quitLobby()
LobbyMenuRepository
+ joinGame()
+ getLobbyList()
LoginRepository
+ connect()
+ resetPwd()
WordDrawRepository
+ sendWordDraw()
FriendListRepository()
+ getFriends()
+ addFriend()
api
SocketService
- mSocket SocketIO
+ getSocket()
ServerRequest
- methods to interact with the server in http
ChatSocketService
-mSocket SocketIO
+ sendMsg()
+ receiveMsg()
ChatSocketService
-mSocket SocketIO
+ startGame()
+ quitLobby()
LobbyMenuSocketService
-mSocket SocketIO
+ joinGame()
+ getLobbyList()
GameSocketService
-mSocket SocketIO
+ sendAction()
+ receiveAction()
model
Action
+ do()
+ undo()
DrawingAction
- lineNode
EraseAction
- lineNode []
Message
- timestamp : Number
- sender: String
- content: String
+ getTimestamp()
+ getSender()
+ getContent()

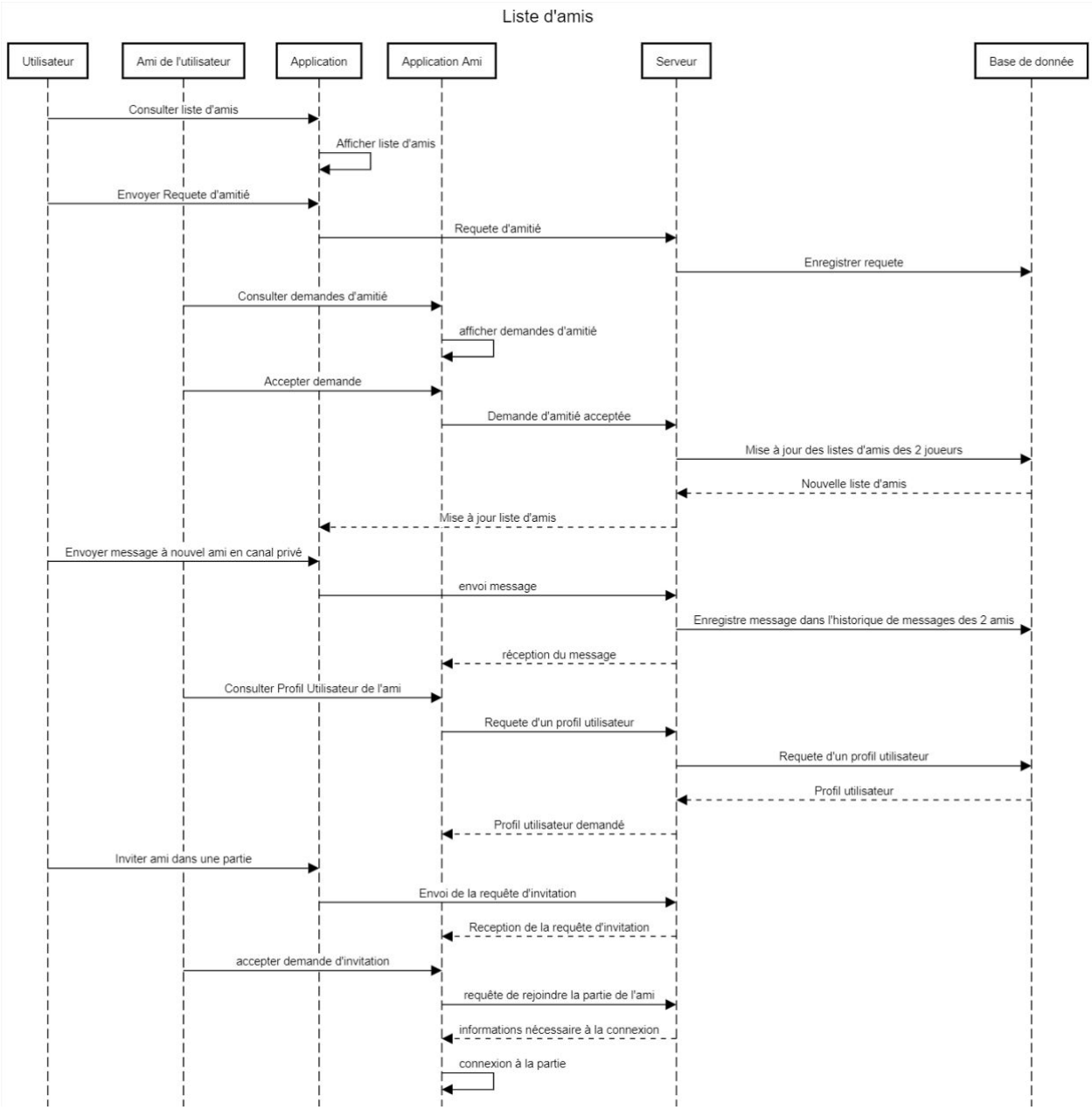
5. Vue des processus



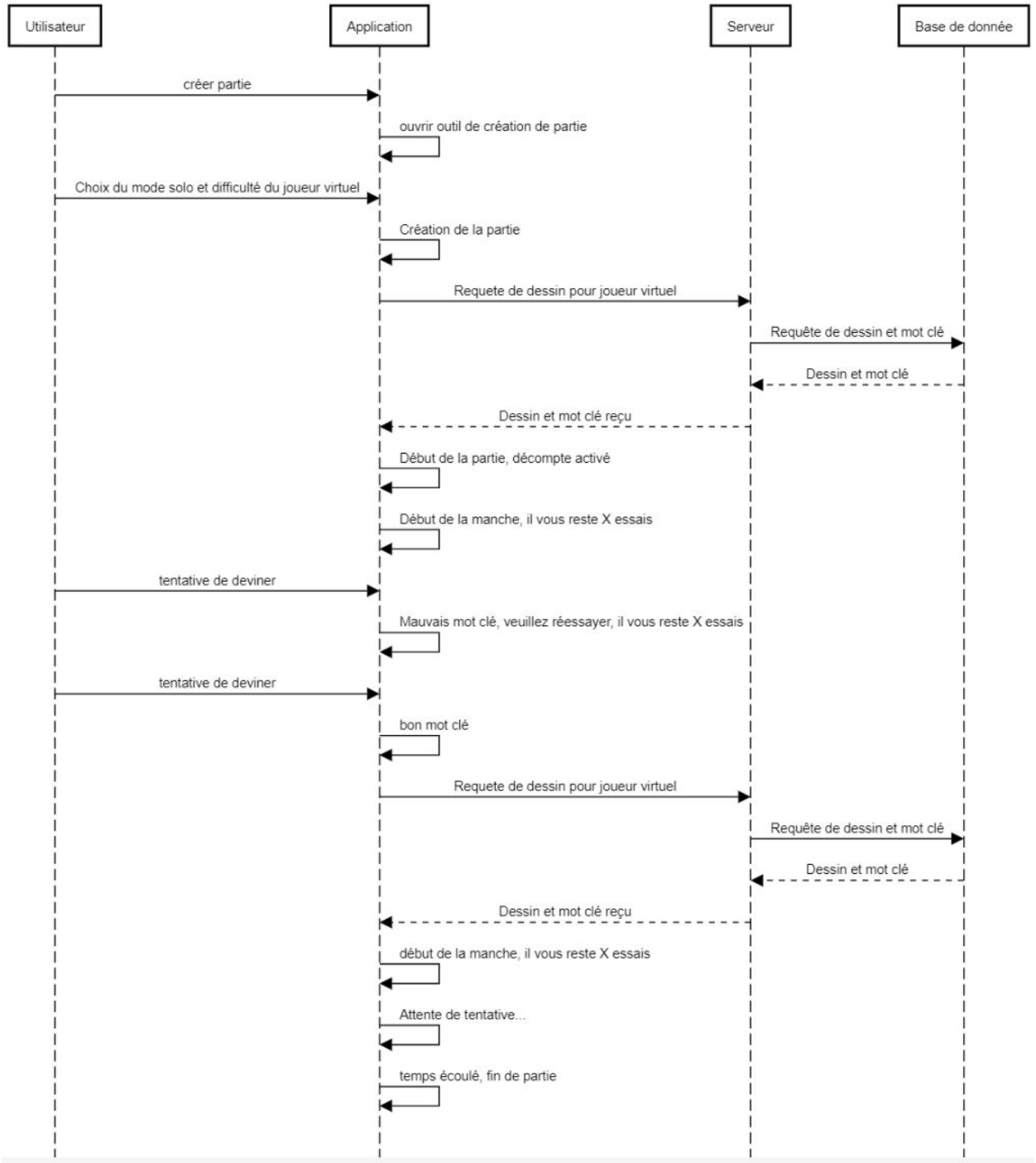


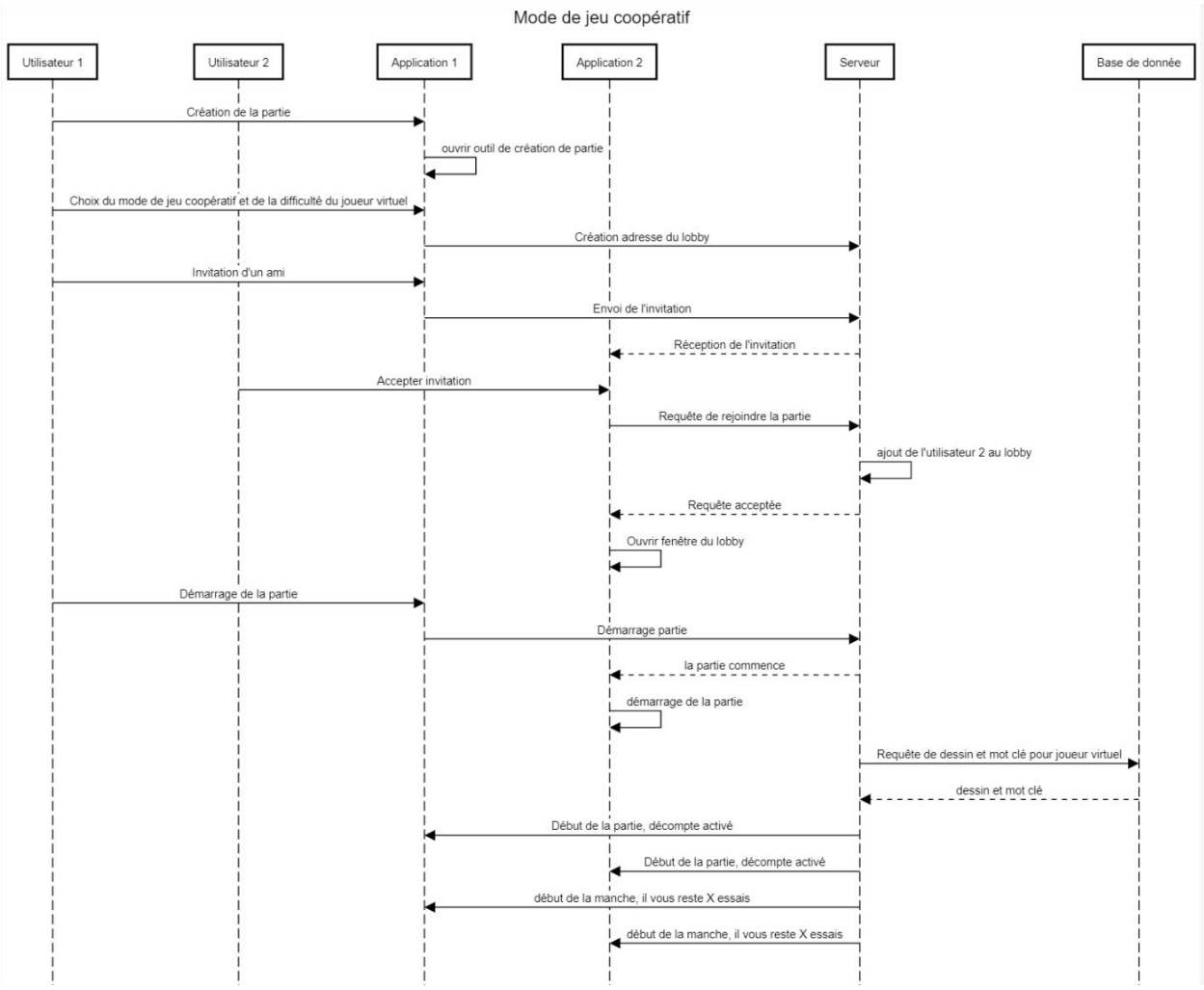

Rôle dessinateur
Utilisateur
Application
Serveur
Commencer le décompte
Vous êtes le dessinateur
Retourne décompte
Afficher décompte
Choisir mot clé
Envoyer mot clé
Dessiner un trait
Envoie le trait
Ajouter le trait au Canvas
Canvas modifié
temps écoulé
Commencer prochaine manche, partir décompte


Rôle devineur
Utilisateur
Application
Serveur
début de manche, part décompte
Vous êtes un devineur
Attente du dessinateur de choisir mot clé
mot clé choisi
entrer tentative de deviner
est tentative exacte?
oui !
Manche terminée, vous avez trouvé le mot clé

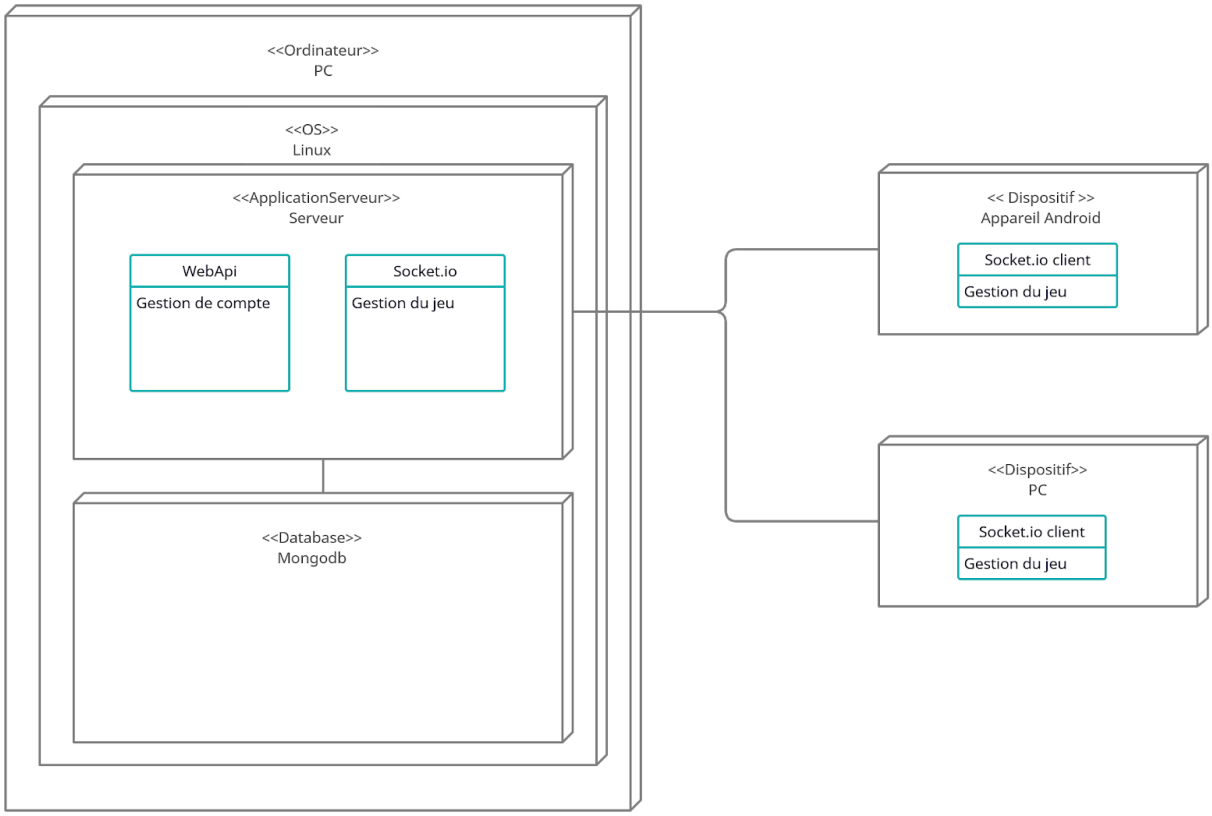

Liste d'amis
Utilisateur
Ami de l'utilisateur
Application
Application Ami
Serveur
Base de donnée
Consulter liste d'amis
Afficher liste d'amis
Envoyer Requete d'amitié
Requete d'amitié
Enregistrer requete
Consulter demandes d'amitié
afficher demandes d'amitié
Accepter demande
Demande d'amitié acceptée
Mise à jour des listes d'amis des 2 joueurs
Nouvelle liste d'amis
Mise à jour liste d'amis
Envoyer message à nouvel ami en canal privé
envoi message
Enregistre message dans l'historique de messages des 2 amis
réception du message
Consulter Profil Utilisateur de l'ami
Requete d'un profil utilisateur
Requete d'un profil utilisateur
Profil utilisateur
Profil utilisateur demandé
Inviter ami dans une partie
Envoi de la requête d'invitation
Reception de la requête d'invitation
accepter demande d'invitation
requête de rejoindre la partie de l'ami
informations nécessaire à la connexion
connexion à la partie

Mode de jeu solo




Mode de jeu coopératif
Utilisateur 1
Utilisateur 2
Application 1
Application 2
Serveur
Base de donnée
Création de la partie
ouvrir outil de création de partie
Choix du mode de jeu coopératif et de la difficulté du joueur virtuel
Création adresse du lobby
Invitation d'un ami
Envoi de l'invitation
Réception de l'invitation
Accepter invitation
Requête de rejoindre la partie
ajout de l'utilisateur 2 au lobby
Requête acceptée
Ouvrir fenêtre du lobby
Démarrage de la partie
Démarrage partie
la partie commence
démarrage de la partie
Requête de dessin et mot clé pour joueur virtuel
dessin et mot clé
Début de la partie, décompte activé
Début de la partie, décompte activé
début de la manche, il vous reste X essais
début de la manche, il vous reste X essais

6. Vue de déploiement



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. La communication par websocket s'en tient à des objets simples qui ne devraient pas trop surcharger le réseau.