Équipe 106

Projet 3 : Fais-moi un dessin
Document d'architecture logicielle

Version 1.4

Historique des révisions

Date	Version	Description	Auteur
2020-02-02	1.0	Complété sections 1, 2, 3 et 7	Karima Salem
2020-02-06	1.1	Ajouter diagramme de paquetages client léger	Nicole Joyal
2020-02-06	1.2	Ajouter diagramme de paquetage client lourd	Émilio
2020-02-07	1.3	Ajouter diagramme de paquetage serveur	Hubert et Zakari
2020-02-07	1.4	Mise en page et révision finale	Équipe 5

Table des matières

1. Introduction 4

2. Objectifs et contraintes architecturaux 4

2.1. Objectifs 4

2.1.1. Sécurité et confidentialité 4

2.1.2. Réutilisation 5

2.1.3. Portabilité 5

2.1. Contraintes 4

2.2.1. Échéancier 4

2.2.2. Langage de développement 5

2.2.3. Outil de développement 5

3. Vue des cas d'utilisation 5

4. Vue logique 8

4.1. Client léger 8

4.2. Client lourd 12

4.3. Serveur 18

5. Vue des processus 21

6. Vue de déploiement 24

7. Taille et performance 24

Document d'architecture logicielle

1. Introduction

Ce document a pour but de détailler l'architecture du projet à réaliser. Pour ce faire, il y est énoncé les divers éléments ayant un impact concret sur la conception, la réalisation et l'utilisation du projet. Tout d'abord, il est question des objectifs et des contraintes quant à l'architecture. Cette section porte sur les limitations en ce qui concerne l'architecture du logiciel. La vue des cas d'utilisation présente les multiples scénarios dans lesquels les acteurs emploient directement le logiciel implémenté. Ainsi, il est possible de réaliser le projet en tenant compte des différentes façons dont il sera utilisé. Ensuite, la vue logique propose les différents paquetages utilisés pour le client lourd, le client léger ainsi que le serveur. La vue des processus permet de comprendre les différentes interactions entre les divers processus. Puis, la vue de déploiement donne un aperçu de la liaison entre le matériel logiciel et le matériel physique. Finalement, la section portant sur la taille et la performance présente les contraintes en ce qui a trait à la qualité du logiciel.

2. Objectifs et contraintes architecturaux

2.1. Objectifs

2.1.1. Sécurité et confidentialité

Tout au long du développement de l'application, les développeurs doivent garder en tête la confidentialité des utilisateurs. Il est primordial que les informations personnelles collectées lors de l'inscription et tout au long de l'utilisation de l'application demeurent privées. Cela ne tient pas compte des statistiques liées aux parties, au classement des usagers ainsi qu'aux informations que les utilisateurs partagent publiquement sur leur profil. Toutefois, une fois la suppression du compte confirmée, toutes les informations propres à l'utilisateur seront détruites.

2.1.2. Réutilisation

Il est important d'implémenter des fonctionnalités qui seront faciles à réutiliser dans d'autres parties du code de l'application. Cela est non seulement une bonne pratique de codage, mais simplifiera la tâche des développeurs quand l'application s'alourdit.

2.1.3. Portabilité

Selon les exigences du client, il est important de considérer la portabilité de l'application. Puisque l'appel d'offres spécifie qu'il soit au moins possible d'utiliser l'application sur un ordinateur Windows ainsi qu'une tablette Android.

2.1. Contraintes

2.2.1. Échéancier

Un prototype du projet contenant une page de connexion ainsi qu'une plateforme de

clavardage doit être livré d'ici le 7 février 2020. Le projet complet doit être rendu avant le 13 avril 2020. Entre temps, l'équipe de développement réalisera des fonctionnalités supplémentaires à chaque semaine, tel que planifié dans l'échéancier du plan de projet.

2.2.2. Langage de développement

Les langages de développement sont imposés de sorte que le client lourd est implémenté en C# avec une interface utilisateur en WPF. Pour le client léger, les fonctionnalités sont faites en Kotlin. Pour le serveur, le langage utilisé est TypeScript.

2.2.3. Outil de développement

Parmi les outils de développement, on retrouve le cadriciel .NET qui nous a été imposé pour le client lourd. Ensuite, en ce qui concerne la communication du serveur avec le client léger et le client lourd nous avons priorisé l'usage de la librairie SocketIO. Enfin, NodeJS et Express seront utilisés pour le serveur.

3. Vue des cas d'utilisation

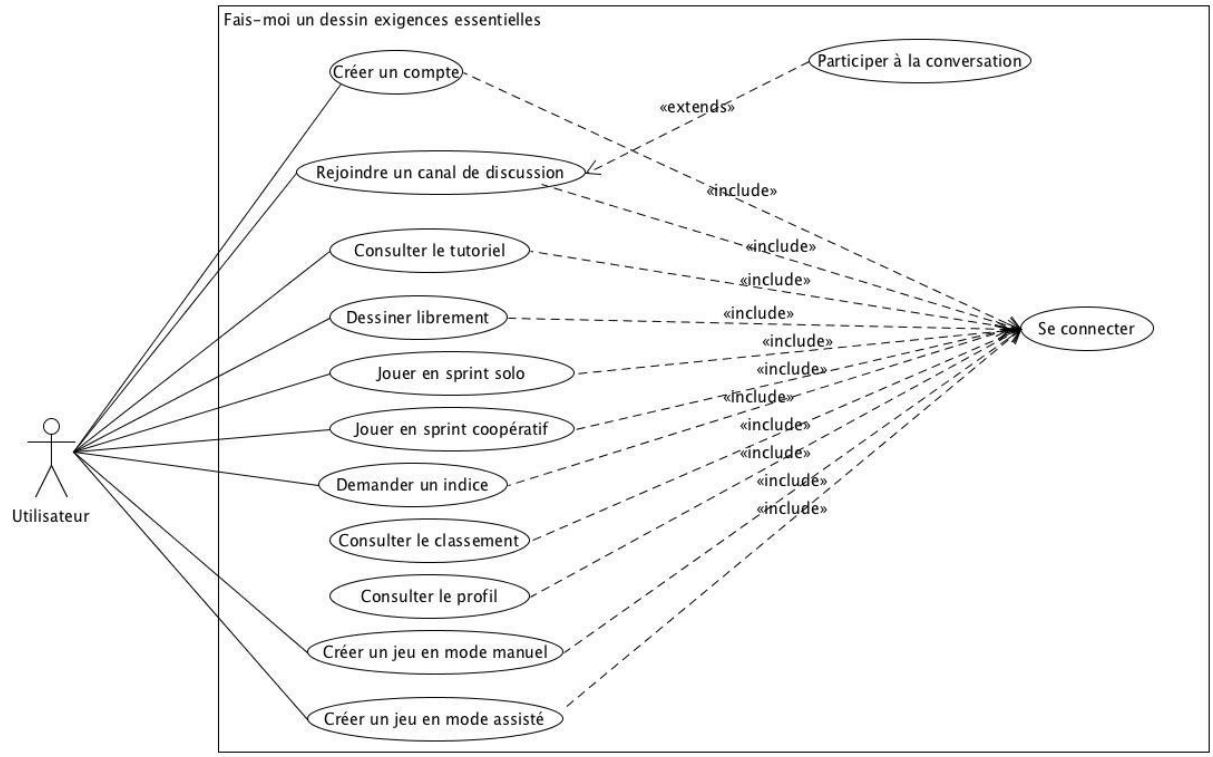

Figure 1: Diagramme de vue des cas d'utilisation des exigences essentielles du côté client lourd et client léger

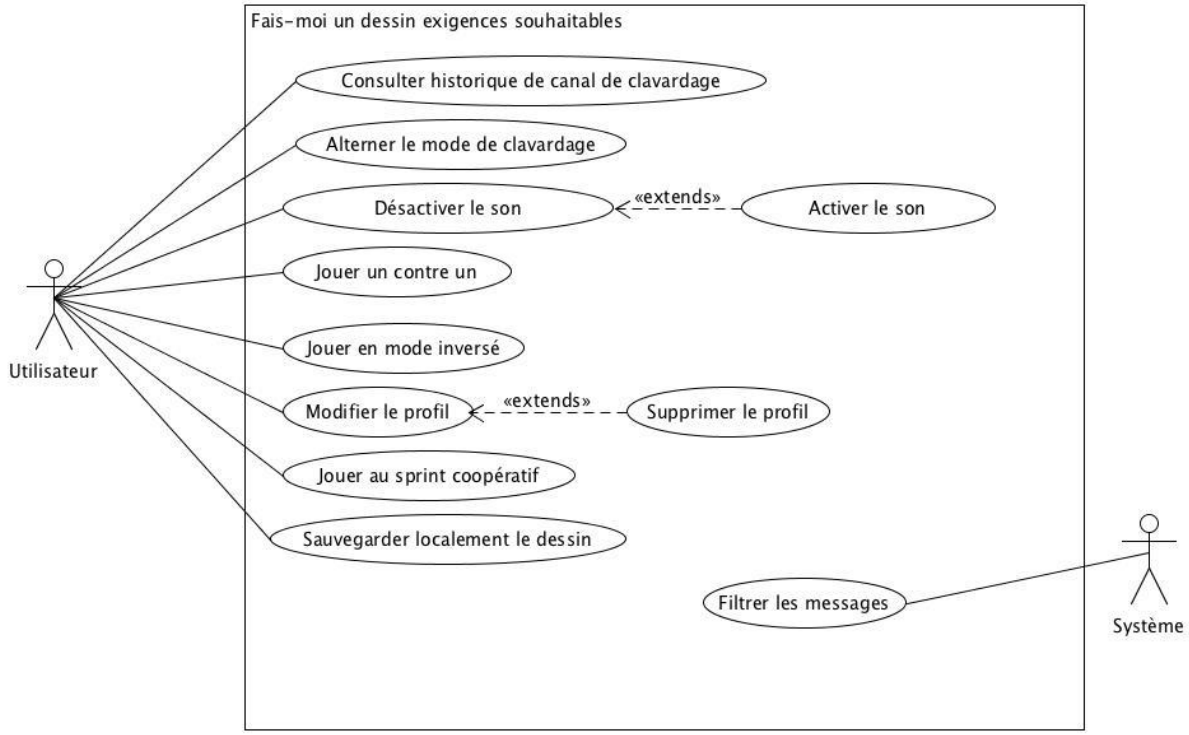

Figure 2: Diagramme de vue des cas d'utilisation des exigences souhaitables du côté client lourd et client léger

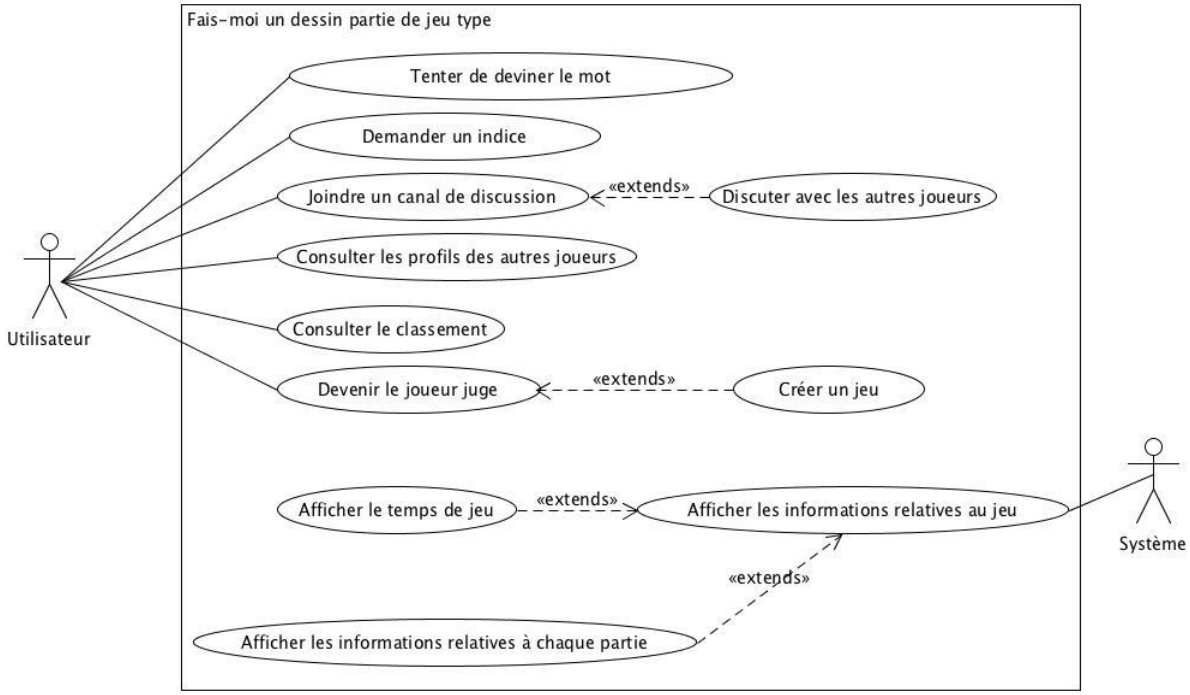

Figure 3: Diagramme de vue des cas d'utilisation d'une partie de jeu type du côté client lourd et client léger

4. Vue logique

4.1. Client léger

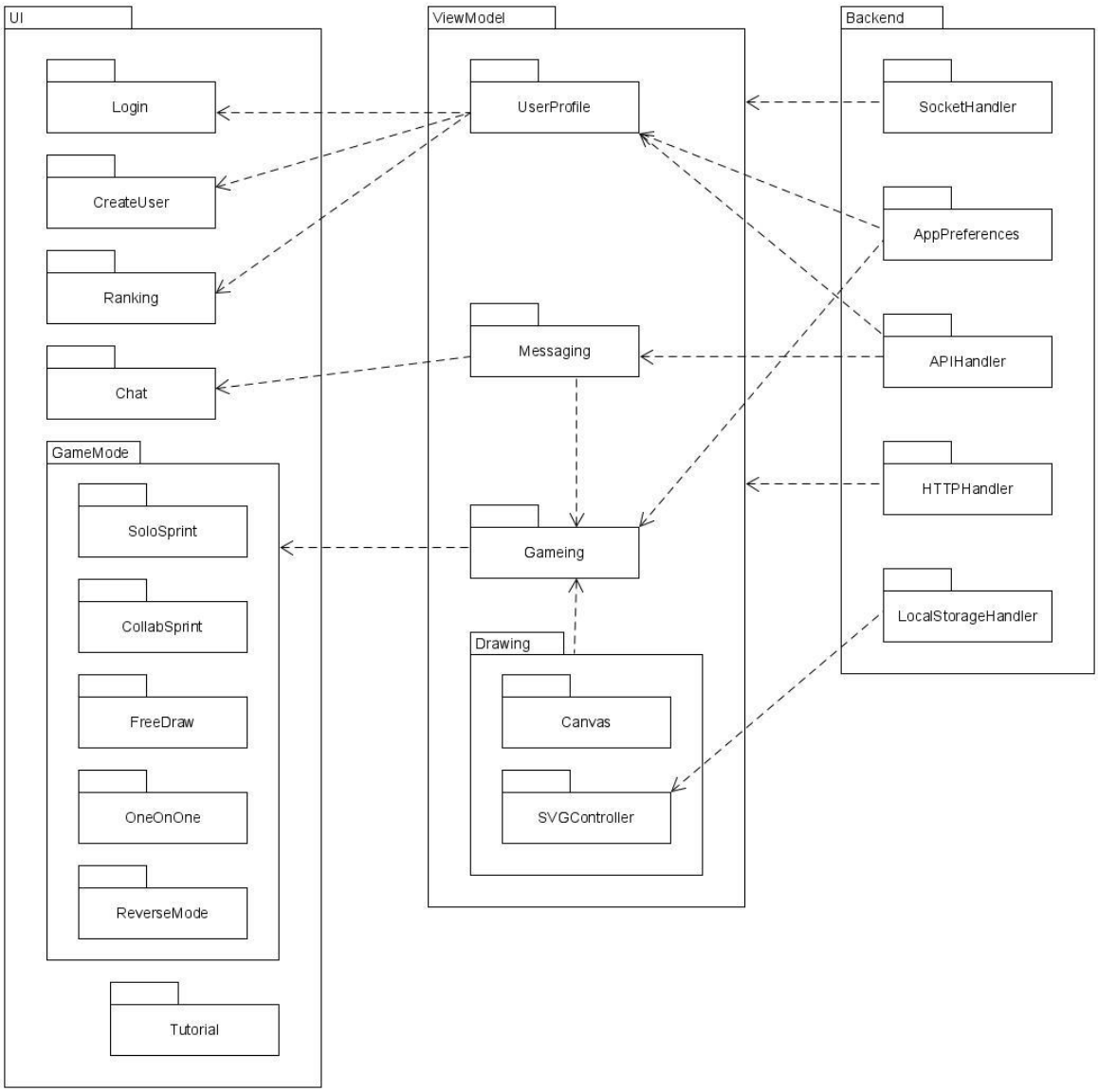

Figure 4 : Diagramme de paquetage du client léger

UI
Le paquet UI contient toutes les activités (ou fragments) de l'application. Les activités et les fragments Android s'occupent de la gestion d'évènements de l'utilisateur. Dans ce paquet, nous retrouvons les différentes parties de l'interface essentielles à l'application.

Login
Le paquet login comporte les composantes reliées à l'interface de connexion de l'application.

CreateUser

Le paquet CreateUser comporte les composantes reliées à la création d'un profil utilisateur de l'application. Il est utilisé notamment lorsque l'utilisateur choisit l'option de se créer un compte à la page de connexion.

Ranking

Le paquet Ranking présente les informations relatives au classement des utilisateurs.

Chat

Le paquet Chat gère les évènements reliés au messagerie de l'application. Il gère l'affichage des messages et l'affichage de l'historique des messages (significatif).

GameMode

Le paquet GameMode englobe les différentes affichages de chaque mode de jeu. Plus particulièrement, il implémente le lobby qui, lui, affiche le mode de jeu selon les entrées de l'utilisateur. Lui et ses paquets fils font le lien avec le le serveur via le ViewModel Gaming.

SoloSprint

Ce paquet contient les affichages relatives au mode de jeu *sprint solo*.

CollabSprint

Le paquet CollabSprint se servira en grande partie des interfaces du *sprint solo*, avec quelques ajouts afin de gérer les fonctionnalités ajoutées.

FreeDraw

Le paquet FreeDraw contient une affichage simple avec simplement le canvas de dessin et la fonction de sauvegarde (qui est unique à cette interface).

OneOnOne

Ce paquet gère l'affichage relative au mode de jeu un contre un.

ReverseMode

Ce paquet gère l'affichage pour le mode de jeu inversé. Il doit donc inverser les rôles typiques des joueurs via le paquet Gaming du ViewModel.

Tutorial

Le paquet Tutorial génère les pages du tutoriel et s'assure de la bonne séquence des informations. La particularité de ce paquet est qu'il ne doit pas faire appel au ViewModel, car il génère tout simplement des *GIFs* à l'utilisateur.

ViewModel

Le paquet ViewModel fait le lien entre l'affichage à l'utilisateur et la récupération des informations du serveur ou des composantes invisibles à l'utilisateur. Plus particulièrement, il fait la récupération ou l'envoi des informations au serveur. Il utilise les informations récupérées du Backend afin de permettre aux paquets UI d'afficher à l'utilisateur en fonction.

UserProfile

Ce ViewModel gère les appels de création d'utilisateur et les connexions. Il s'occupe aussi du sauvegarde des paramètres d'utilisation.

Messaging

Le paquet Messaging gère les appels pour envoyer et recevoir un message d'un canaux particulier, avec les informations utilisateur correspondantes. Il gère aussi les appels à la base de données afin de récupérer l'historique des messages. Il fait aussi appel aux API nécessaires pour le filtrage des mots.

Gaming

Ce paquet gère plusieurs aspects d'un jeu qui sont communs à tous les modes. Par exemple, il gère l'assignation de rôles des joueurs, ainsi que l'alternance des rôles. Il associe les points aux joueurs à la fin de chaque manche et gère la progression d'une partie.

Drawing

Le paquet Drawing gère notamment les différentes opérations de dessin. Il gère aussi le sauvegarde des images et, par conséquent, le lien avec le storage des images localement.

Canvas

Ce paquet se charge de la gestion des options de dessin.

SVGController

SVGController gère le lien avec le storage local de l'appareil. Il se sert du paquet LocalStorageHandler afin de permettre la sauvegarde de l'image.

Backend

Le paquet Backend contient les classes relatives aux modifications et à la récupération des données externes, serveur, ainsi que l'appareil propre.

SocketHandler

SocketHandler est un paquet qui se charge de la gestion de la connexion socket avec le serveur. Il gère les appels au serveur lorsqu'une réponse pertinente est requise du côté client (ex. une connexion réussi ou non).

AppPreferences

Le paquet AppPreferences gère les préférences de l'utilisateur dans l'application (ex. le volume du son, l'état de connexion etc).

APIHandler

Ce paquet effectue les appels aux API externes utilisés. Par exemple, pour filtrer les messages nous faisons appel à un service externe.

HTTPHandler

HTTPHandler contrôle les requêtes HTTP nécessaires à l'application. Souvent, il sera utile pour la récupération et la modification de données de la base de données (ex. historique des messages)

LocalStorageHandler

Ce paquet contrôle l'accès au storage local de l'appareil. Il est utilisé notamment pour le sauvegarde d'images du mode de jeu dessin libre.

4.2. Client lourd

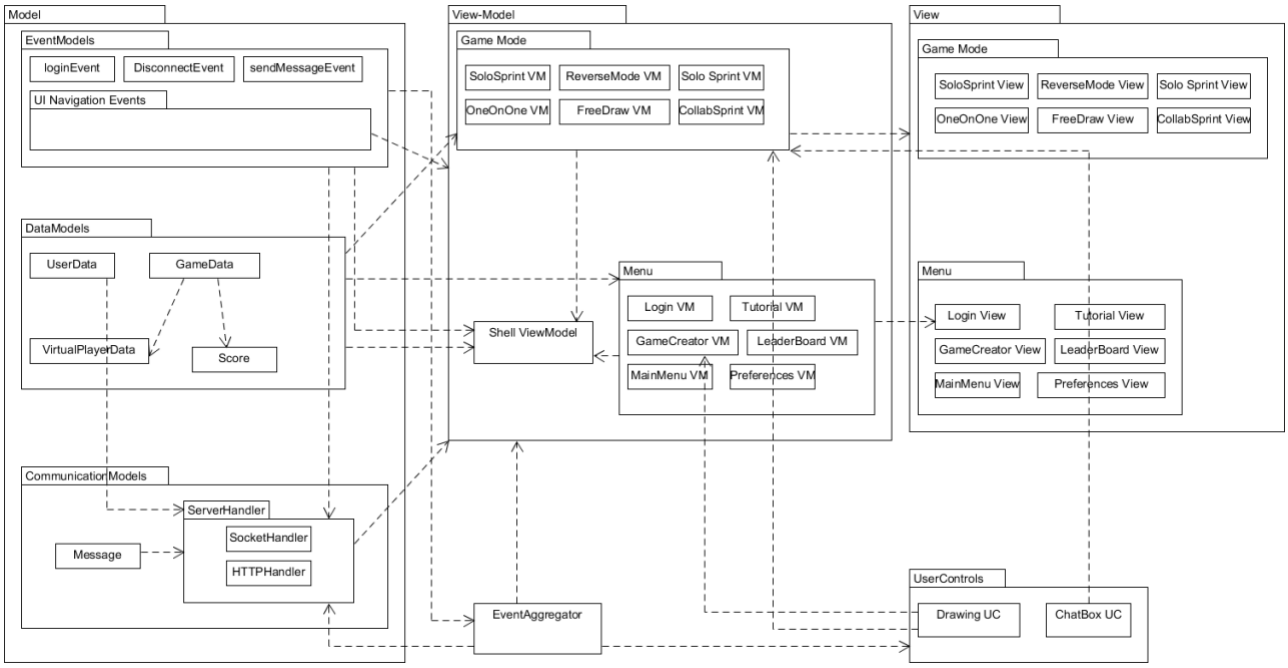

Figure 5 : Diagramme de paquetage du client lourd

Model

Le paquet Model regroupe les paquets relatifs aux modèles d'évènements, de communication et de données.

EventModels

EventModels est un paquet qui contient les modèles des différents événements qui seront transmis entre les vues et les classes de l'application via le singleton EventAgreggator.

LoginEvent

Événement associé à l'appui du bouton login dans la vue Login. Permet d'engendrer un changement de vue via le Shell ViewModel.

DisconnectEvent

Événement associé à l'appui du bouton disconnect dans la vue MainMenu. Permet d'engendrer un changement de vue via le Shell ViewModel.

ConnectEvent

Événement utilisé par le SocketHandler pour vérifier la disponibilité d'un compte dans le serveur.

sendMessageEvent

Événement associé à l'envoi d'un message, permet d'avertir le SocketHandler de faire l'émission du message écrit dans le ChatBox UC.

UI Navigation Events

Paquet regroupant le reste des événements permettant de changer de vue via la vue modèle squelette (Shell ViewModel).

DataModel

Le paquet DataModel regroupe les paquets relatifs aux modèles de gestion des données dans l'application.

UserData

Classe regroupant les données de l'utilisateur actif. Par exemple son nom d'utilisateur, son avatar, ses préférences utilisateur, ses messages etc.

GameData

Classe regroupant les données d'un jeu. Par exemple les dessins, les indices, le nombre de tours, le score final pour chaque joueur etc.

VirtualPlayerData

Classe regroupant les données d'un joueur virtuel. Par exemple, ses traits de personnalité, la liste de son pointage des parties ou il a participé, etc.

Score

Classe regroupant les données d'un pointage. Par exemple: le type de partie joué, le nombre de points, l'identifiant unique du joueur qui a réalisé le pointage, etc.

CommunicationModels

Paquet regroupant tous les paquets essentiels à la communication avec le serveur, notamment la classe Message, la classe APIHelper et le paquet ServerHandler.

Message

Classe décrivant un message, c'est à dire un contenu, une estampille de temps et un nom d'utilisateur.

ServerHandler

Paquet regroupant les classes permettant de gérer les interactions avec le serveur.

SocketHandler

Classe permettant de gérer la connection socket entre le client et le serveur ainsi que la réception de messages et les événements des vues de jeu. Permet notamment de gérer les événements sur le socket.

HHTPHandler

Classe permettant de gérer les appels HTTP entre le client et le serveur. Utile pour les requêtes de données.

ViewModel

Le paquet regroupe les vues modèles faisant le lien entre l'affichage (Views) et les différents modèles logiques de l'application (Models).

GameMode

Paquet regroupant les vues modèles des modes de jeu de l'application.

SprintSoloVM

Vue modèle du mode de jeu sprint solo. Contient une variable pointage en cours.

ReverseModeVM

Vue modèle du mode de jeu inversé. Contient une variable pointage en cours.

OneOnOneVM

Vue modèle du mode de jeu 1v1. Contient une variable pointage en cours.

FreeDrawVM

Vue modèle du mode de jeu dessin libre. Contient une variable booléenne permettant de cacher ou non les boutons de sauvegarde du dessin.

Shell ViewModel

Classe de la vue-modèle squelette de l'application. C'est-à-dire que toutes les vues de l'application seront contenues dans cette vue. Cette classe permet aussi de gérer les événements de changement de vues (UI Navigation Events) pouvant être émis par les vues.

Menu

Paquet regroupant les vues-modèles des vues ayant un lien avec le menu principal de l'application.

MainMenu VM

Vue modèle de la vue du menu principal.

Login VM

Vue modèle de la vue de connexion.

LeaderBoard VM

Vue modèle de la vue de classement. Contient une liste de pointage.

Tutorial VM

Vue modèle de la vue d'un tutoriel. Contient une liste d'images à afficher.

Preferences VM

Vue modèle de la vue des préférences de l'application.

EventAggregator

Classe permettant de centraliser tous les événements de l'application, par exemple un changement de vue, la réception d'un message, la réception de données, etc.

UserControls

Paquet regroupant les classes de contrôle utilisateur, celles-ci pouvant être intégrées dans les vues ou affichées dans une vue séparée (utile par exemple pour la boîte de chat ou les modes de jeu).

ChatBox UC

Classe de contrôle utilisateur d'une boîte de chat.

Drawing UC

Classe de contrôle utilisateur d'une boîte de dessin.

View

Paquet regroupant les paquets de vue de l'application.

GameMode

Paquet regroupant les vues modèles des modes de jeu de l'application.

SprintSolo View

Vue du mode de jeu sprint solo. Contient l'UC Drawing, l'UC Chatbox, l'affichage des points, le chronomètre, etc

ReverseMode View

Vue du mode de jeu inversé. Contient l'UC Drawing, l'UC Chatbox, l'affichage des points, le chronomètre, le mot décrivant le dessin demandé, etc

OneOnOne View

Vue du mode de jeu 1v1. Contient l'UC Drawing, l'UC Chatbox, l'affichage des points, le chronomètre, les indices, etc

FreeDraw View

Vue du mode de jeu dessin libre. Contient l'UC Drawing.

Menu

Paquet regroupant les vues-modèles des vues ayant un lien avec le menu principal de l'application.

MainMenu View

Vue modèle de la vue du menu principal. Contient des boutons dirigeant aux autres vues menu.

Login View

Vue modèle de la vue de connexion.

LeaderBoard View

Vue modèle de la vue de classement. Affiche une liste des 10 premiers meilleurs pointages d'un mode de jeu, ainsi que la position du joueur actif.

Tutorial View

Vue modèle de la vue d'un tutoriel. Permet de passer des images une par unes avec un bouton suivant et précédent.

Preferences View

Vue modèle de la vue des préférences de l'application. Permet d'activer/désactiver le son, changer les paramètres de son compte utilisateur et un bouton pour sauvegarder les changements.

4.3. Serveur

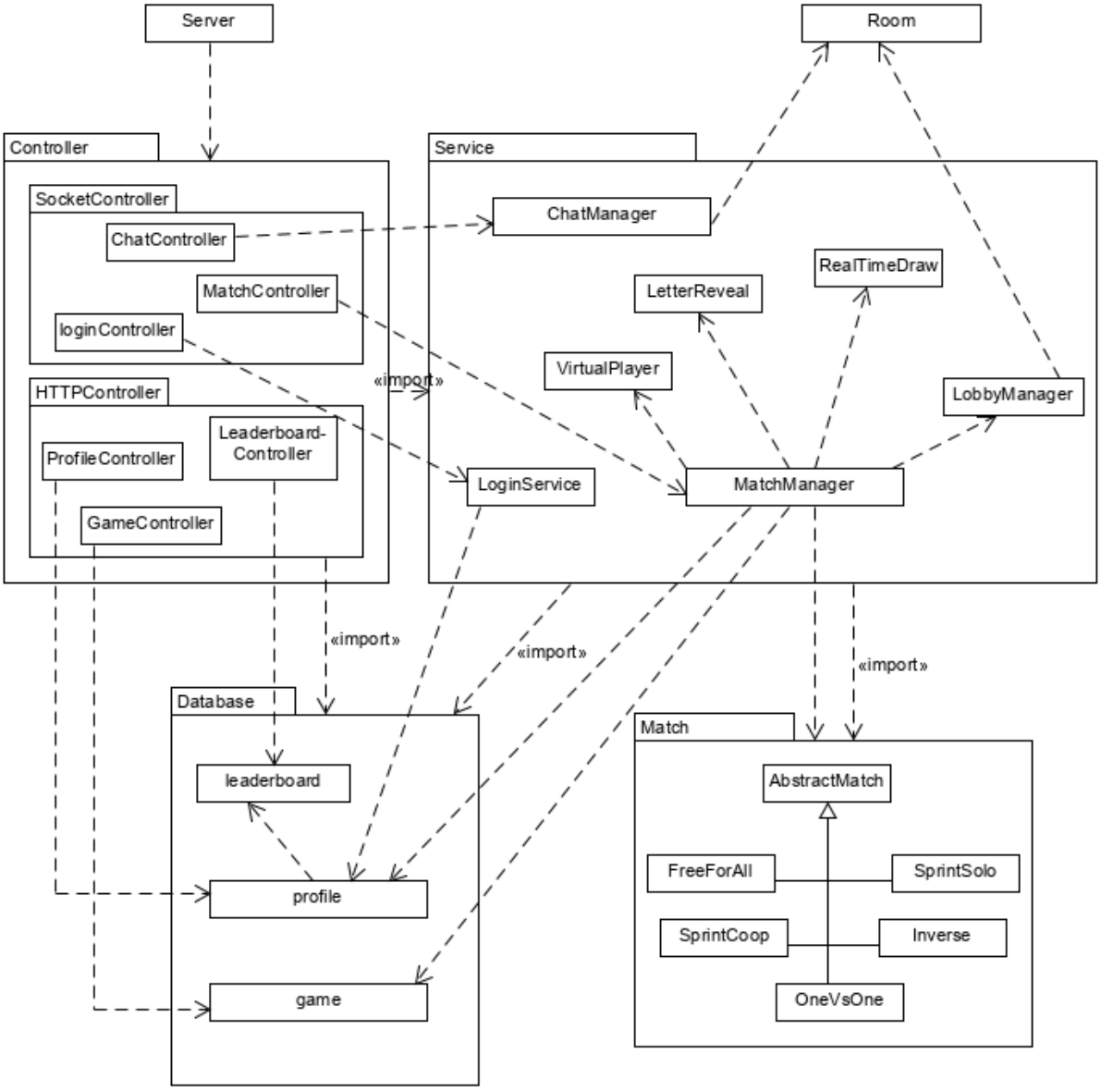

Figure 6 : Diagramme de paquetage du serveur

Server
Point de départ de l'application. C'est de cette classe que les controllers sont démarrés et que le serveur se met en écoute des requêtes réseaux imminentes.

Controller
Englobe les paquets SocketController et HTTPController

SocketController

Englobe les classes qui s'occupent de la connexion Socket.io

ChatController

Classe qui s'occupe de recevoir les messages concernant le clavardage.

LoginController

Classe qui s'occupe de la connexion de l'utilisateur au serveur.

MatchController

Classe qui s'occupe des messages envoyés durant une partie.

HTTPController

Englobe les classes qui s'occupent de la connexion avec requêtes HTTP.

Profile controller

Classe qui gère les changements liés aux profils à la demande de l'utilisateur.

GameController

Classe qui s'occupe d'insérer les jeux créés dans la base de données.

LeaderBoardController

Classe qui s'occupe d'envoyer le classement au client lorsque celui-ci le demande.

Database

Englobe les classes permettant d'ajouter, modifier et supprimer les éléments des différentes tables de la base de données MongoDB. Chaque classe ci-dessous correspond à une table dans la base de données.

leaderboard

Classe permettant l'accès à la table regroupant les classements des joueurs pour chaque mode de

jeu de la base de données.

profile

Classe permettant l'accès à la table regroupant les informations sur les profils des utilisateurs et les statistiques de partie de ceux-ci.

game

Classe permettant l'accès à la table regroupant les jeux créés par les usagers sur le client lourd.

Service

ChatManager

Service qui s'occupe de faire le traitement sur le clavardage (messages insultants, mots devinés, demande d'indices)

MatchManager

Service qui s'occupe d'effectuer toutes les actions déclenchées par les évènements dans le controller lié à une partie, tels que début d'une partie, début d'une manche, fin d'une manche, fin d'une partie.

LoginService

Service qui s'occupe de gérer les connexions et les déconnexions des utilisateurs.

VirtualPlayer

Service qui s'occupe de répertorier tous les comportements des joueurs virtuels selon leur personnalité.

LetterReveal

Service qui s'occupe de dévoiler une lettre selon un certain intervalle de temps au cours d'une partie.

RealTimeDraw

Service qui s'occupe de gérer le dessin en temps réel vu par tous les joueurs d'une partie.

LobbyManager

Service qui s'occupe de répertorier les joueurs d'un même lobby.

Match

Paquet regroupant les classes qui s'occupent des différents fonctionnements des parties.

AbstractMatch

Classe abstraite avec les méthodes de base à implémenter pour la fonctionnement d'une partie.

FreeForAll

Classe dérivant de AbstractMatch s'occupant du fonctionnement d'une partie du mode de jeu FreeForAll.

SprintSolo

Classe dérivant de AbstractMatch s'occupant du fonctionnement d'une partie du mode de jeu SprintSolo.

SprintCoop

Classe dérivant de AbstractMatch s'occupant du fonctionnement d'une partie du mode de jeu SprintCoop.

OneVsOne

Classe dérivant de AbstractMatch s'occupant du fonctionnement d'une partie du mode de jeu OneVsOne.

Inverse

Classe dérivant de AbstractMatch s'occupant du fonctionnement d'une partie du mode de jeu Inverse.

Room

Classe objet implémentant le fonctionnement de nos canaux dans socket.io.

5. Vue des processus

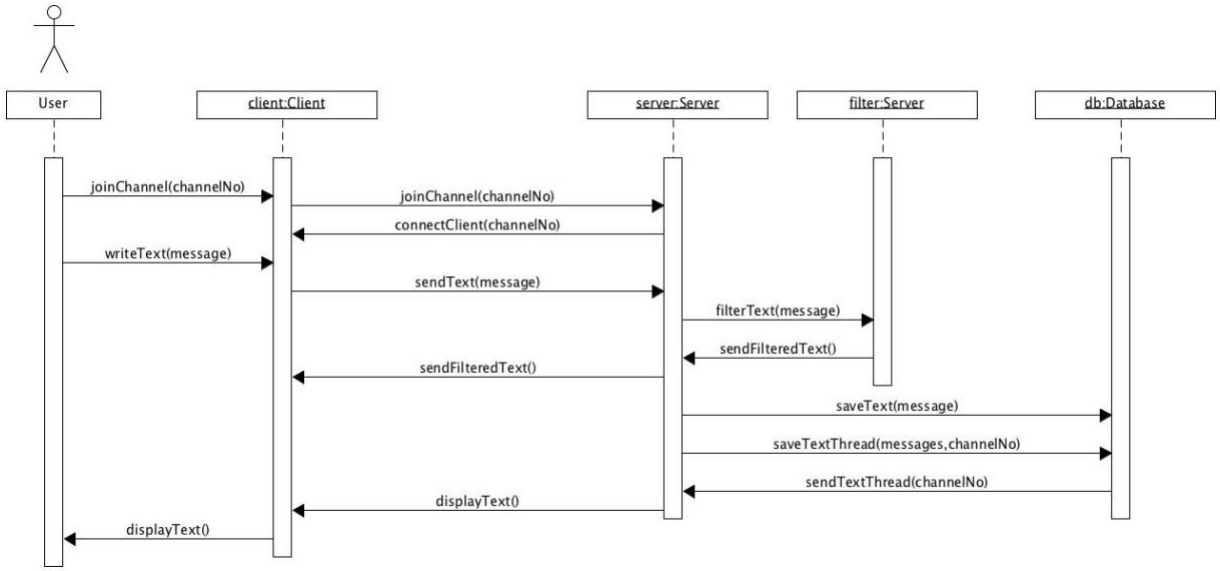

Figure 7: Diagramme de séquence du clavardage

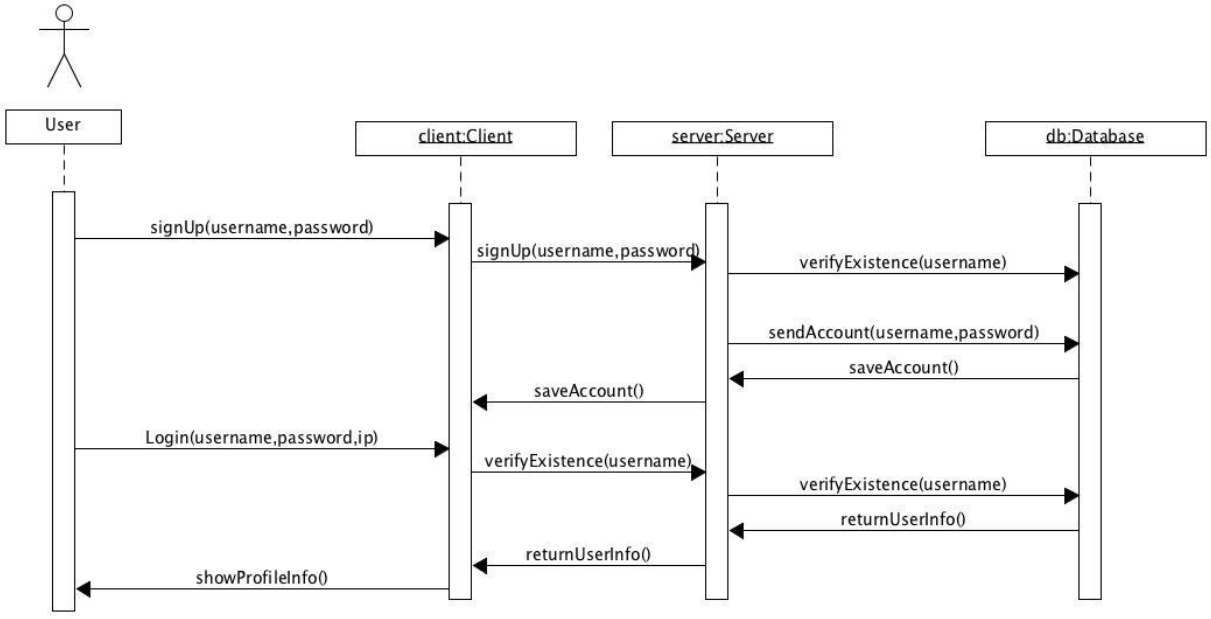

Figure 8: Diagramme de séquence de l'inscription et de la connexion

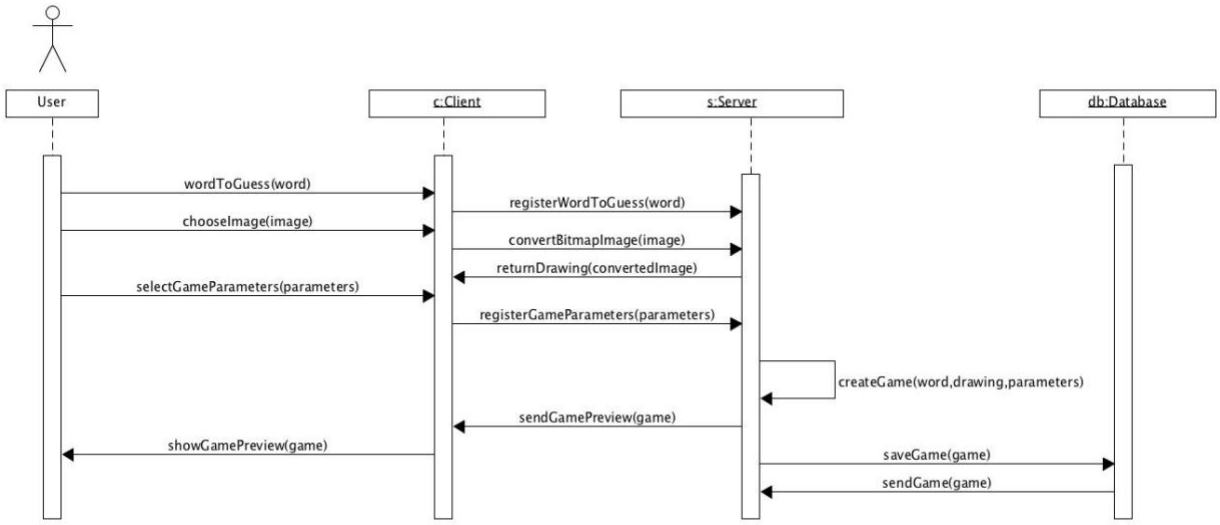

Figure 9: Diagramme de séquence de la création de jeu en mode Assisté I

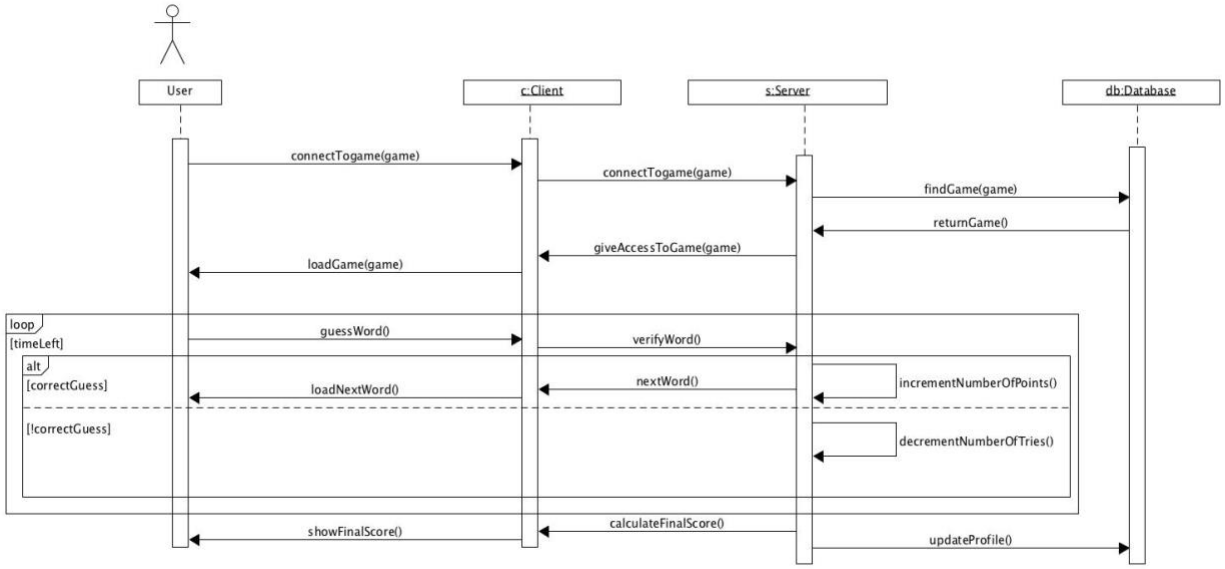

Figure 10: Diagramme de partie de jeu en sprint solo

6. Vue de déploiement

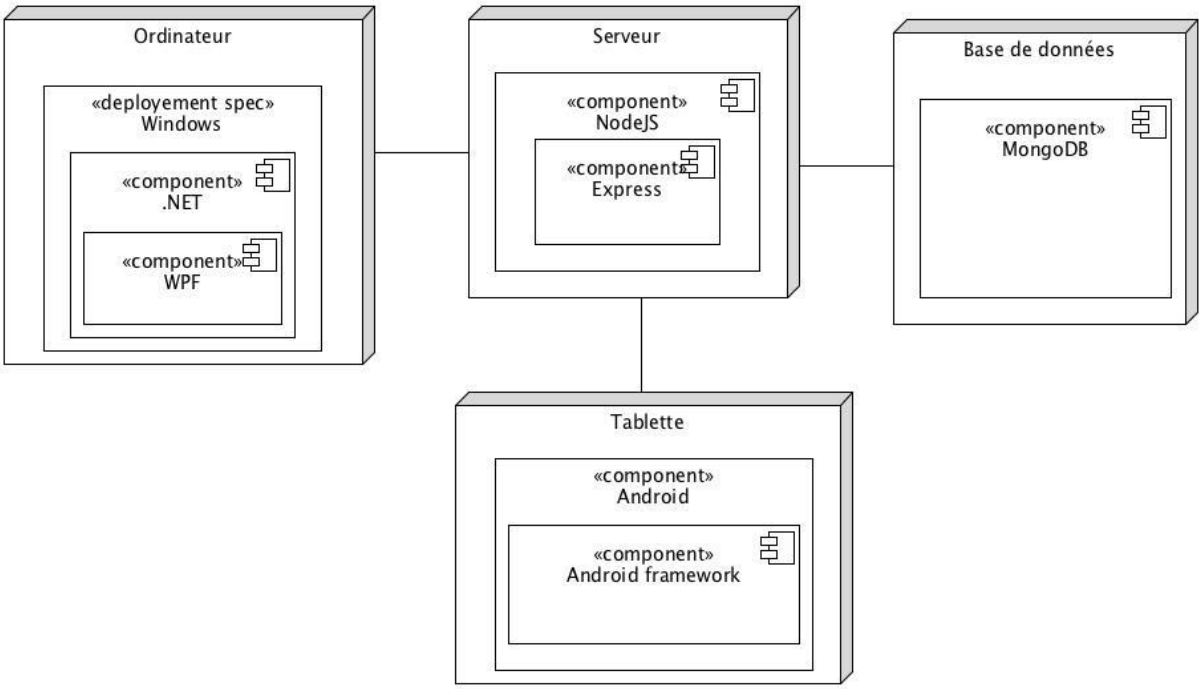

Figure 11: Diagramme de vue de déploiement du système

7. Taille et performance

Il est impératif de considérer la taille et la performance de l'application lors de sa planification. Donc, pour le client lourd il serait important d'imposer à l'utilisateur de libérer au moins 1 Go d'espace sur sa machine. Du côté performance, l'application ne devrait nécessiter que 1 Go de RAM pour bien fonctionner. L'exploitation du processeur ne devrait jamais surpasser 25 %. Pour assurer le moins de frustration possible de la part de l'utilisateur, le lancement de l'application devrait prendre au maximum 15 secondes, mais il serait idéal que cela prenne moins de 5 secondes. Ensuite, pour le client léger, la tablette pour laquelle les dimensions de l'application ont été optimisées est la Galaxy Tab E. Afin d'assurer un bon fonctionnement adéquat de l'application sur la tablette, cette dernière nécessite au moins 1 Go d'espace mémoire. Pour la performance, tout comme pour le client léger, 1 seul Go de RAM suffit pour que l'application roule bien sur la machine. Par ailleurs, l'exploitation du processeur ne doit pas surpasser 25 %. Aussi, l'application devrait pouvoir s'ouvrir et se lancer en moins de 10 secondes. Finalement, le serveur doit être capable de supporter au moins 100 canaux de discussions simultanés et 25 parties jouées en même temps en n'importe quel mode de jeu.