Document d'architecture logicielle

Version 1.0

Historique des révisions

Date	Version	Description	Auteur
2023-03-19	1.0	Structure cas d'utilisation	Julien
2023-03-21	1.0	Diagramme de paquetage et classes	Majeed
2023-03-21	1.0	Diagrammes de séquence	Maxime

Table des matières

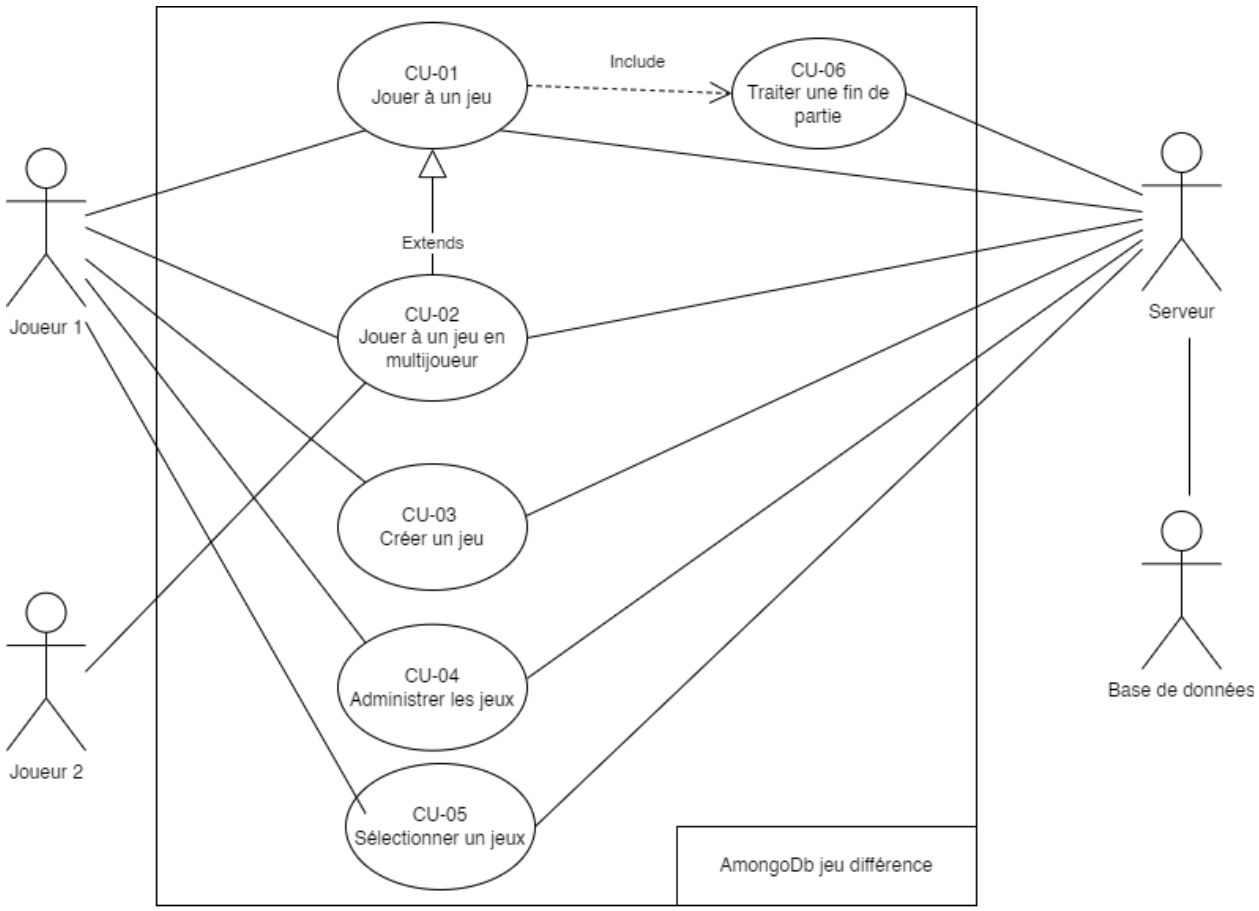
Document d'architecture logicielle

1. Introduction

Cette documentation a pour objectif de fournir une vue d'ensemble du système avec les fonctionnalités du sprint 3 à l'aide de différents diagrammes UML. Cette documentation se compose de cinq parties principales: la vue des cas d'utilisation, la vue des processus, la vue logique et vue de déploiement.
La vue des cas d'utilisation permet de comprendre les différentes interactions possibles entre les utilisateurs et le système. Elle décrit les fonctionnalités du système à partir du point de vue des acteurs impliqués.
La vue des processus décrit comment le système fonctionne en interne. Elle montre comment les différentes fonctions du système interagissent entre elles pour réaliser les fonctionnalités du système.
La vue logique présente la structure interne du système. Elle montre comment les différentes classes et objets du système sont organisés et interagissent entre eux.
La vue de déploiement décrit comment le système sera déployé sur les différentes plates-formes informatiques. Elle montre comment les différents composants du système seront distribués sur les différents ordinateurs et serveurs.

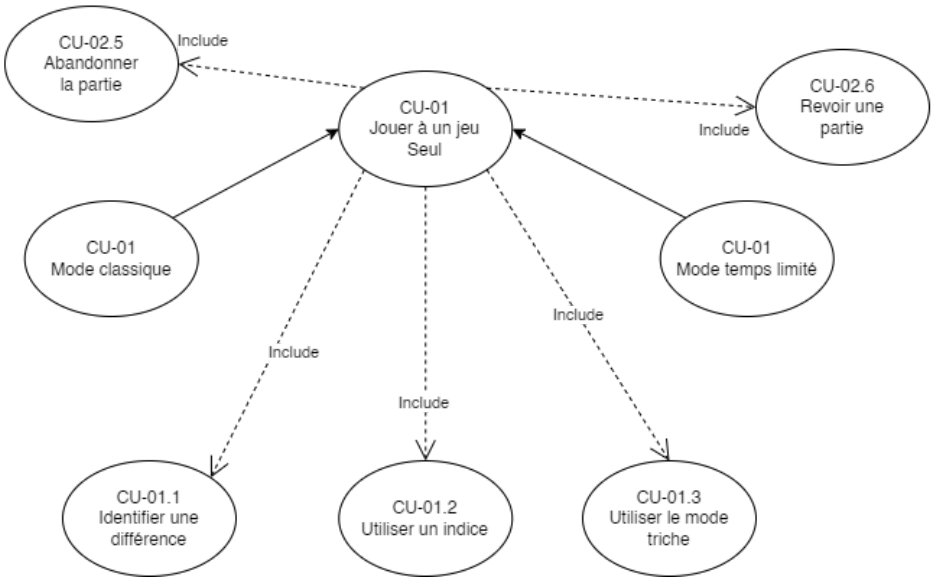
2. Vue des cas d'utilisation

a. Diagramme de contexte générale



* Le joueur 2 n'a pas été connecté à tous les CU pour clarté.

b. Diagramme de contexte : CU-01 Jouer à un jeu seul

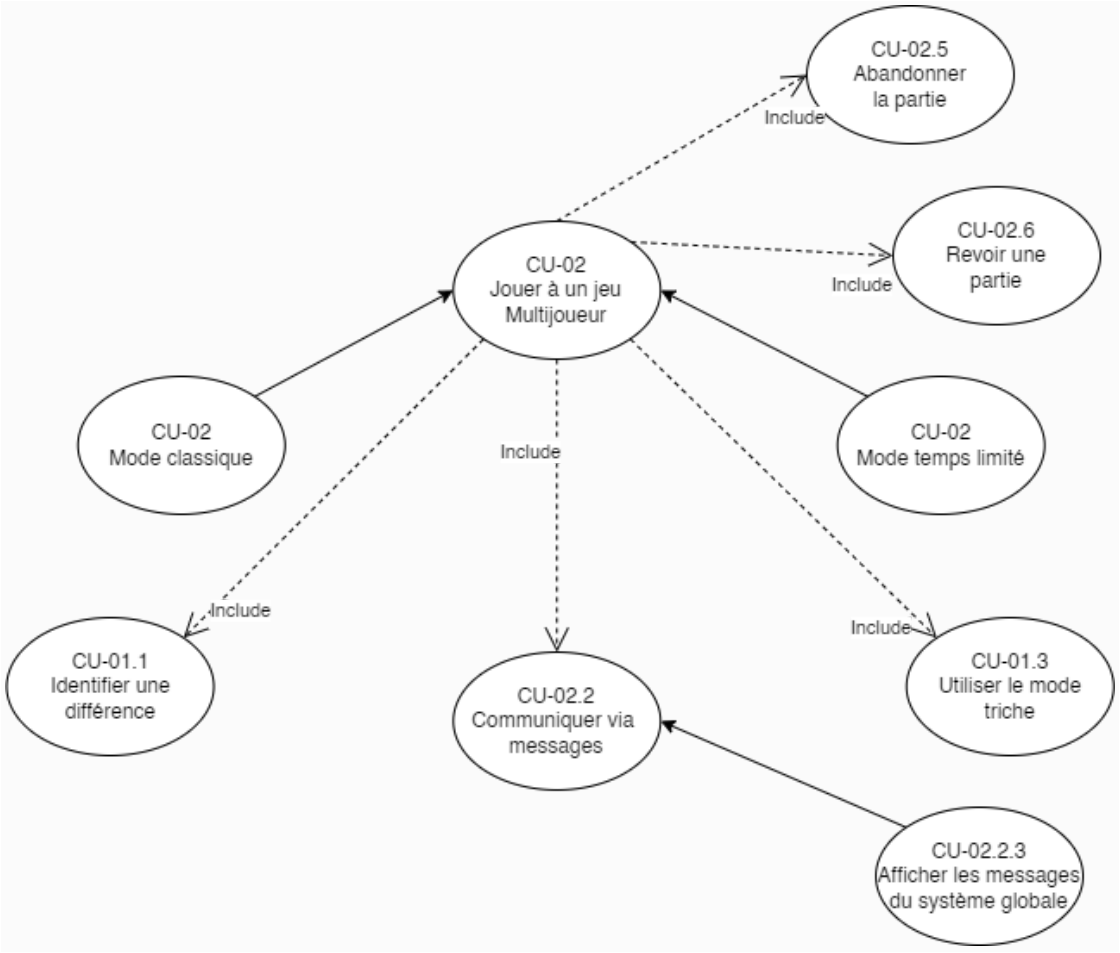


Cas d'utilisation:	**CU-01 Mode temps limité**
Acteur(s):	**Joueur**
Acteur(s) secondaire:	**Serveur**
Type:	**Primaire**
Description:	**Le joueur cherche une différence entre deux images, dès qu'une différence est trouvée la pair d'image est changé. Le jeu continu tant que le minuteur n'est pas arrivé à 0 ou qu'il reste des images qui n'ont pas été affiché au joueur**

Cas d’utilisation:	CU-01.2 Utiliser un indice
Acteur(s):	Joueur
Type:	Primaire
Description:	Le joueur dispose de 3 indices, à chaque utilisation une pénalité est appliquée sur le temps restant ou écoulé en fonction du mode de jeu. Chaque indice donne un cadre dans lequel se trouve une différence. La zone indiquée par l’indice se réduit à chaque utilisation.

Cas d’utilisation:	CU-01.6 Revoir une partie
Acteur(s):	Joueur
Type:	Primaire
Description:	À la fin d’un partie on peut activer le mode reprise vidéo. Ce mode permet au joueur de revoir le déroulé d’une partie en x1, x2, x4. Il peut mettre en pause ou recommencé le visionnage.

c. Diagramme de contexte : CU-02 Jouer à un jeu Multijoueur

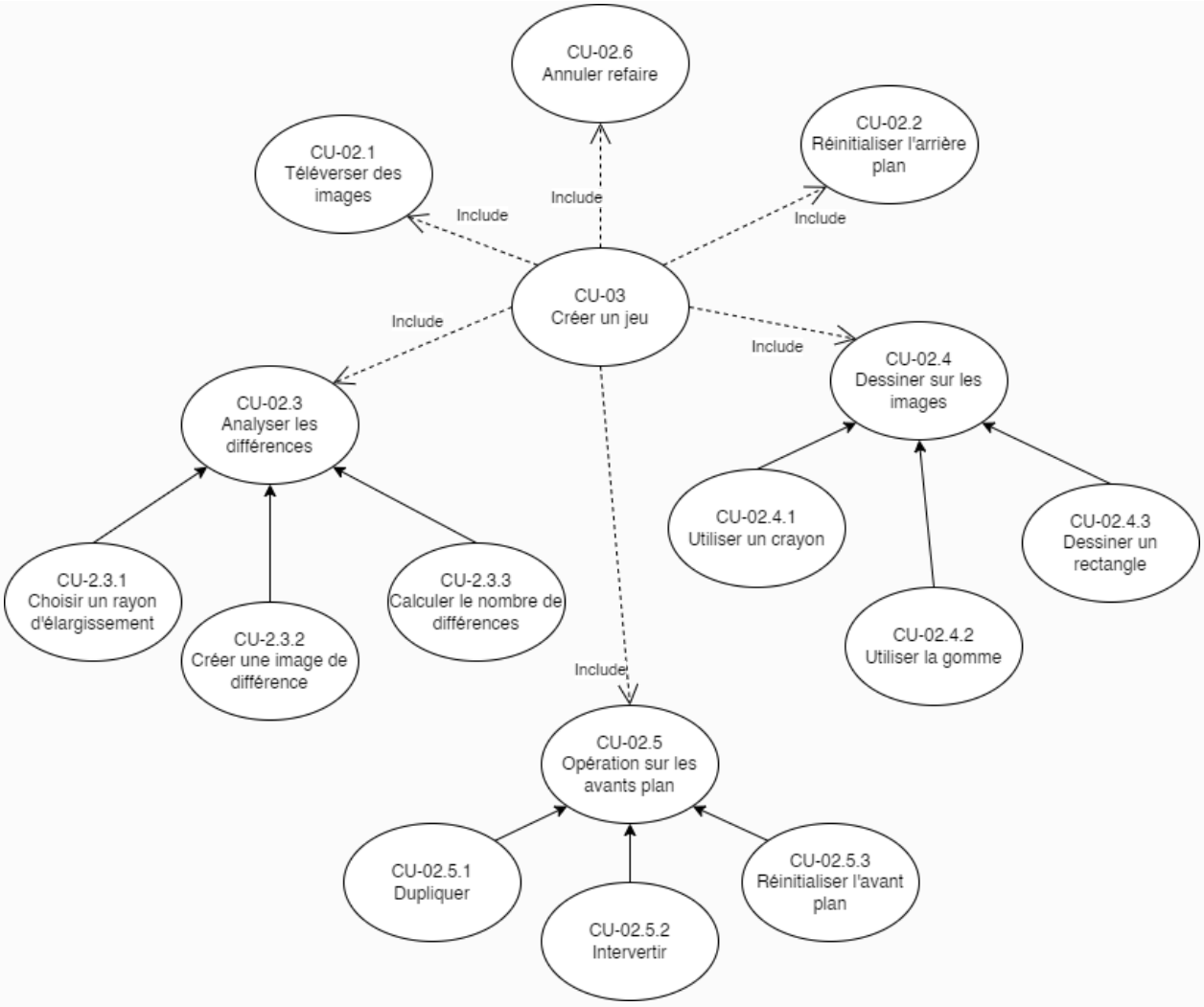


Cas d'utilisation:	**CU-02.2.3 Afficher les messages du système**
Acteur(s):	**Joueur, Serveur**
Type:	**Secondaire**
Description:	**Le serveur envoie aux différents joueurs des messages d'événement globale comme un record. Le message sera affiché dans le chat de la partie.**

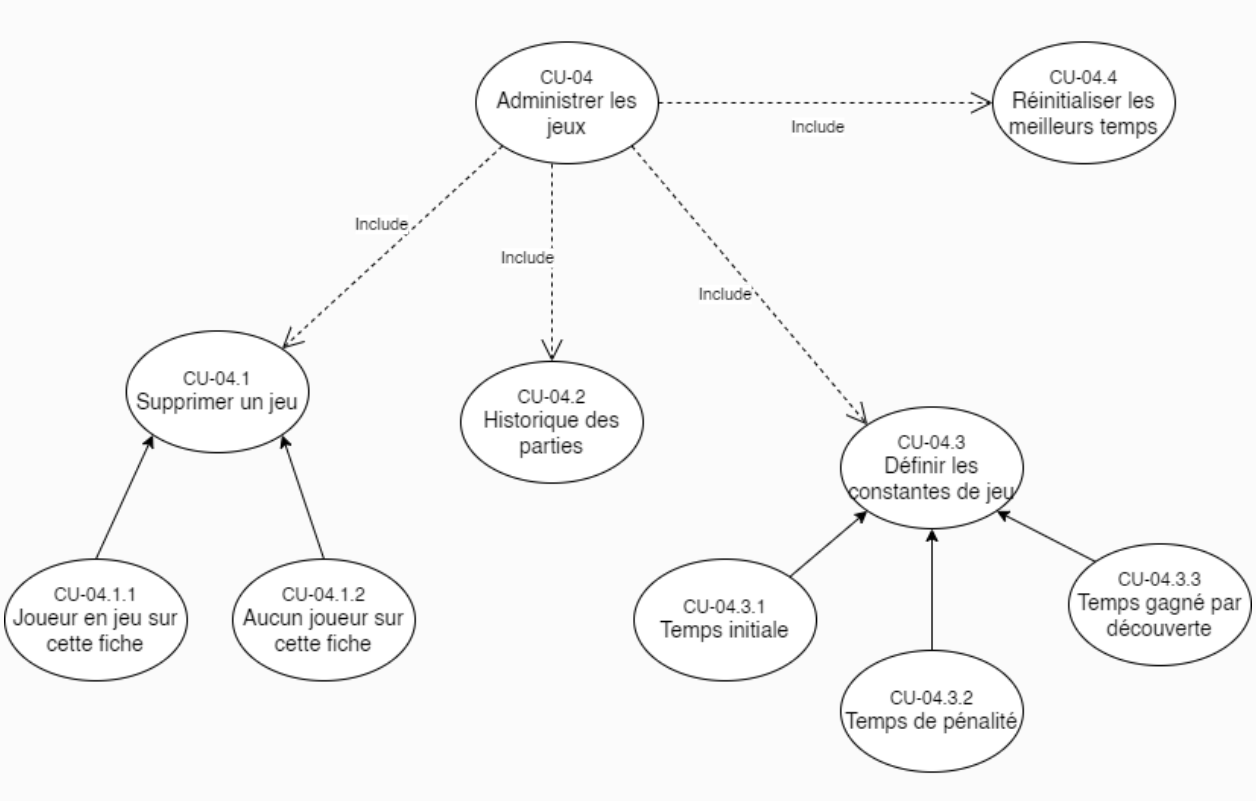
Cas d'utilisation:	**CU-02 Mode temps limité**
Acteur(s):	**Joueur 1 et joueur 2**
Voir CU-01	

Cas d'utilisation:	CU-02.6 Revoir une partie
Voir CU-01.6	

d. Diagramme de contexte : CU-03 Créer un jeu



e. Diagramme de contexte : CU-04 Administrer les jeux

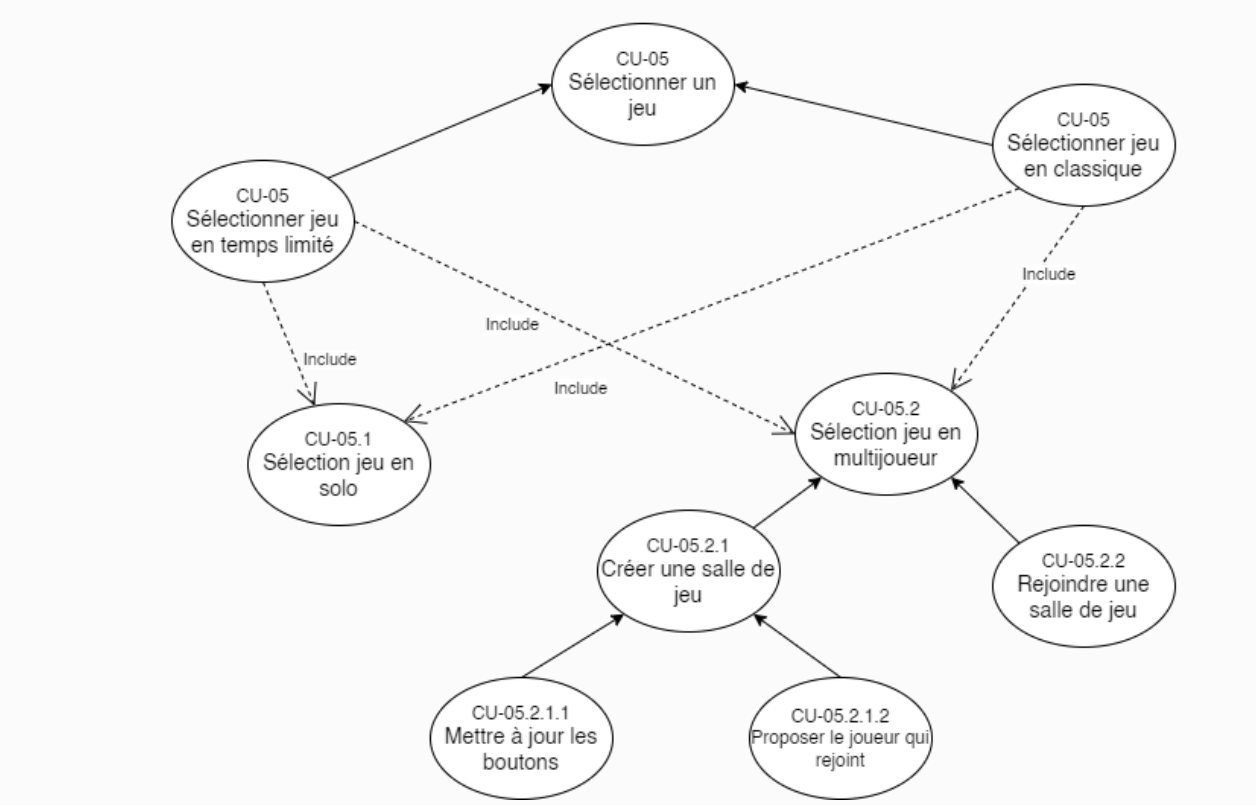


Cas d'utilisation:	**CU-04.2 Historique des parties**
Acteur(s):	**Joueur, serveur, base de données**
Type:	**Primaire**
Description:	**Les parties jouées sont enregistrées dans un historique dans la base de données par le serveur afin que le joueur puisse le consulter depuis la page de configuration.**

Cas d’utilisation:	CU-04.3 Définir les constantes de jeu
Acteur(s):	Joueur
Type:	secondaire
Description:	Le joueur peut définir les constantes de jeu comme la pénalité par indice utilisé ou le temps de la partie pour le mode temps limité depuis la page de configuration.

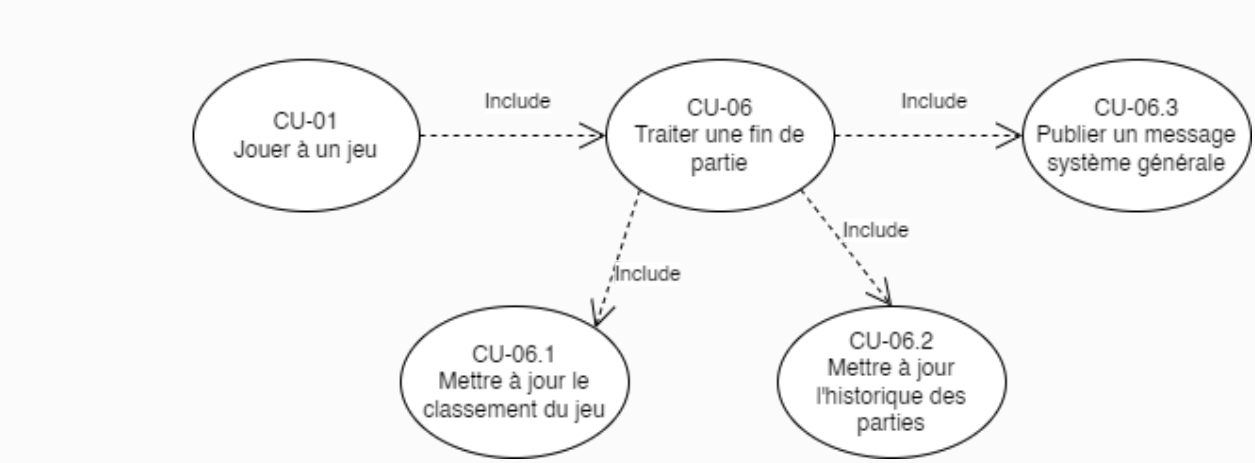
Cas d’utilisation:	CU-04.4 Réinitialiser les meilleurs temps
Acteur(s):	joueur, serveur, base de données
Type:	Primaire
Description:	Depuis la vue de configuration le joueur peut réinitialiser les meilleurs temps de chaque jeu indépendamment. Le serveur va ainsi supprimer les données correspondantes dans la base de données.

f. Diagramme de contexte : CU-05 Sélectionner un jeu



Cas d'utilisation:	CU-05 Sélectionné jeu en temps limité
Acteur(s):	Joueur
Type:	Primaire
Description:	Pouvoir sélectionner des jeux via la page de sélection pour jouer en mode temps limité.

g. Diagramme de contexte : CU-06 Traiter une fin de partie



Cas d'utilisation:	**CU-06 Traiter une fin de partie**
Acteur(s):	**Serveur**
Type:	**Primaire**
Description:	**Le serveur fait certains traitements en fin de partie pour enregistrer certaines données.**

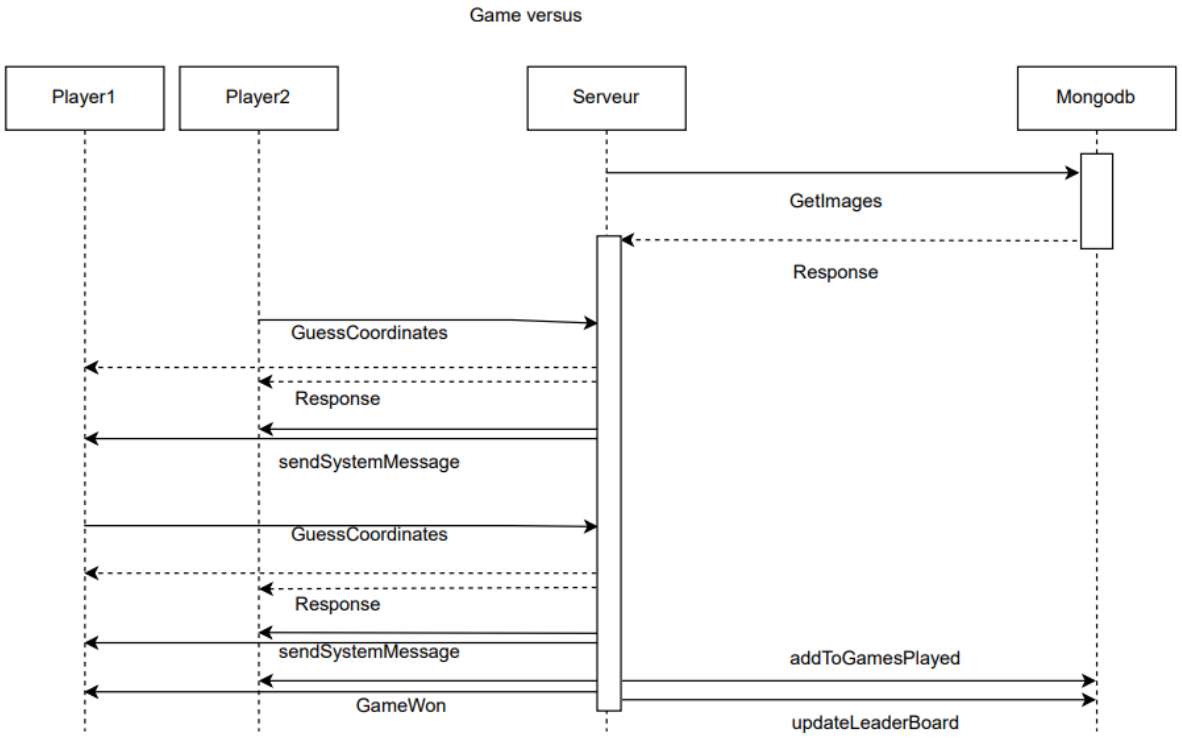
Cas d'utilisation:	**CU-06.1 Mettre à jour le classement du jeu**
Acteur(s):	**Serveur**
Acteur(s) secondaire:	**Joueur**
Type:	**secondaire**
Description:	**Le serveur met à jour le classement du jeu en fonction du temps que le joueur a mis pour compléter le jeu.**

Cas d'utilisation:	**CU-06.2 Mettre à jour l'historique des parties**
Acteur(s):	**Serveur**
Type:	**secondaire**
Description:	**Le serveur insère les informations sur la partie qui vient de se terminer dans l'historique.**

Cas d’utilisation:	**CU-06.3 Publier un message système générale**
Acteur(s):	**Serveur**
Type:	**secondaire**
Description:	**Dans le cas où le meilleur temps d’un jeu à été battu on notifie tous les clients qui sont en jeu qu’un record à été battu.**

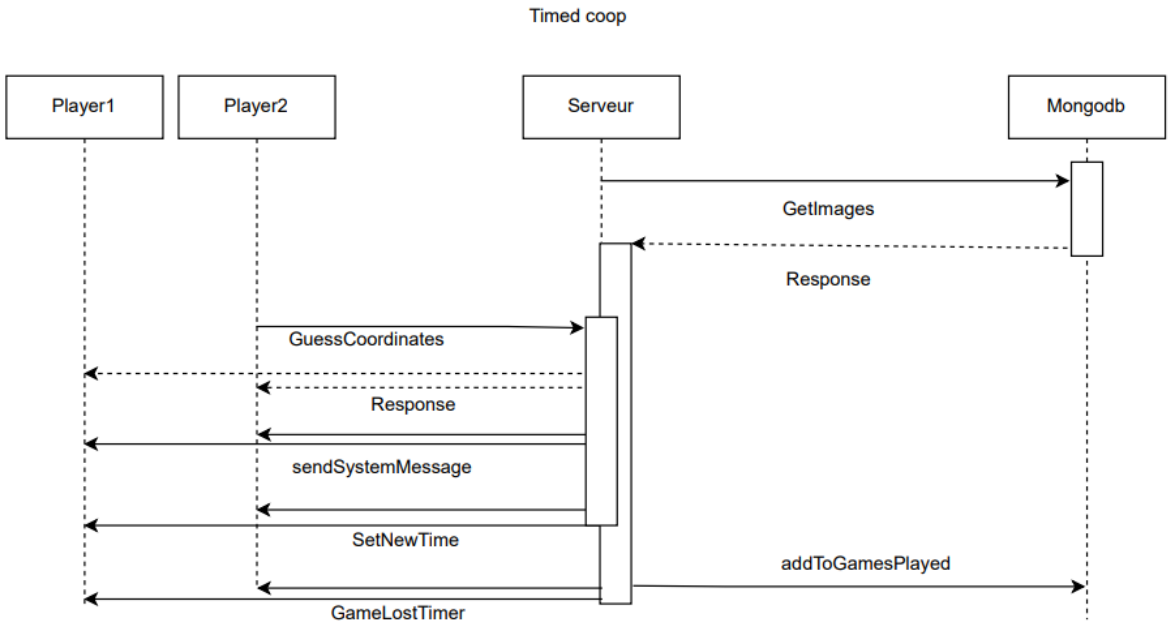
3. Vue des processus - MAXIME

a. Diagramme de séquence: partie 1v1



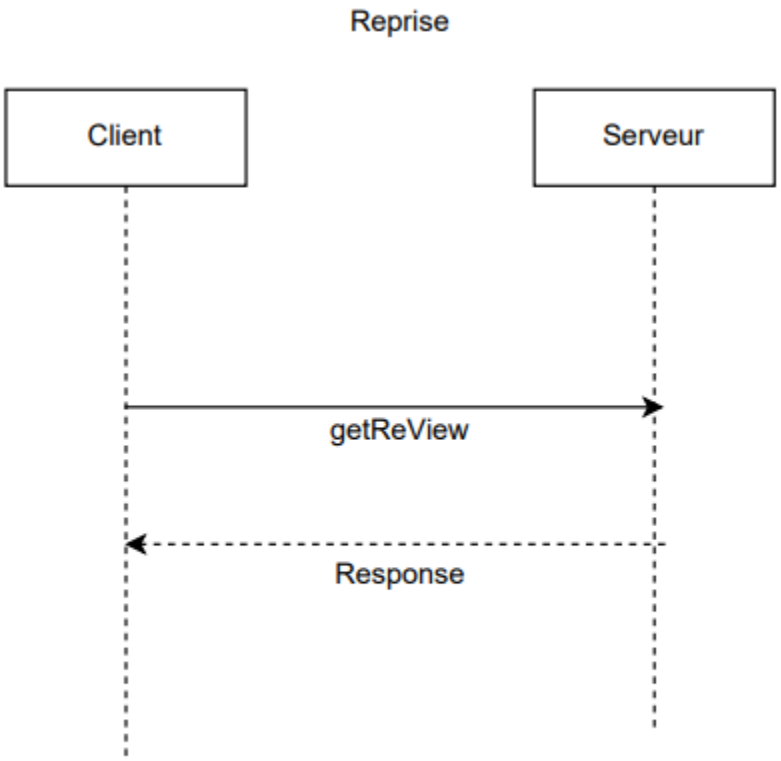
Note: Fonctionnalité du Sprint 1 en majorité, nous incluons pour montrer à quel moment d'une partie (la fin) la partie est enregistrée dans la liste de parties jouées. Évidemment, addToGamesPlayed est appelée à toute fin de jeu, mais updateLeaderBoard seulement quand une partie se finit par une victoire (pas par un abandon par un des joueurs)

b. Diagramme de séquence: partie coopérative en contre la montre



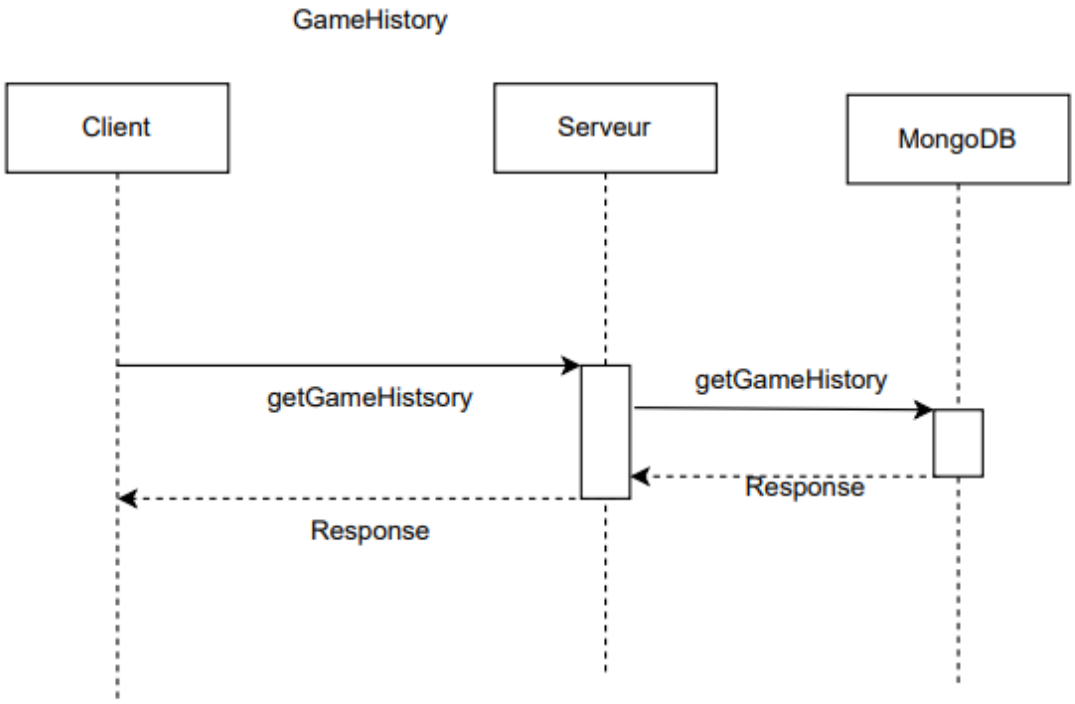
Note: timed solo utilise les mêmes séquences, sauf que les messages du serveur arrivent à un seul client. Toutes les différences se font au niveau du matchmaking

c. Diagramme de séquence: visualisation de reprise de partie

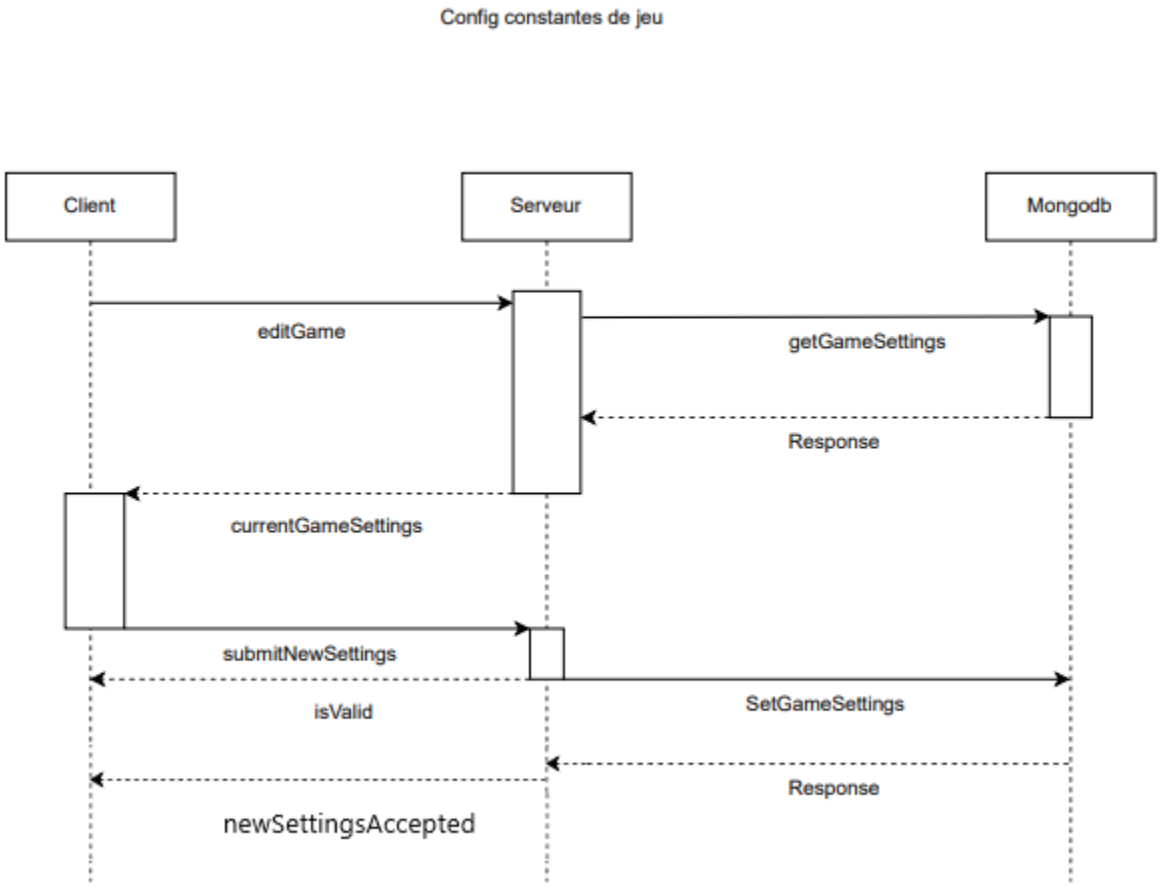


Note: Au cours de la partie, le serveur enregistre toutes les différences et les états de jeu. A la fin de la partie, il génère un fichier vidéo et l'envoie au client sous forme de BLOB. Toute la logique de vitesse est faite au niveau du client.

d. Diagramme de séquence: visualisation de l'historique des parties

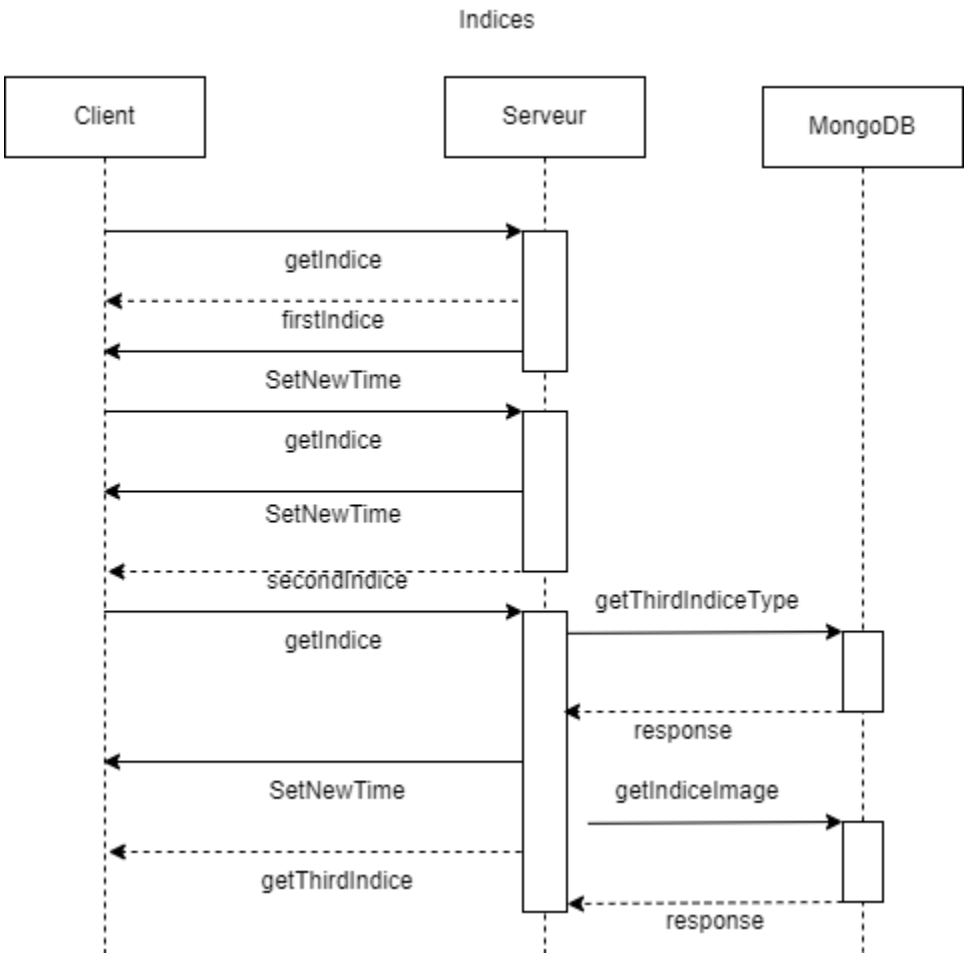


e. Diagramme de séquence: modification des paramètres de jeu contre la montre par le panneau de configuration



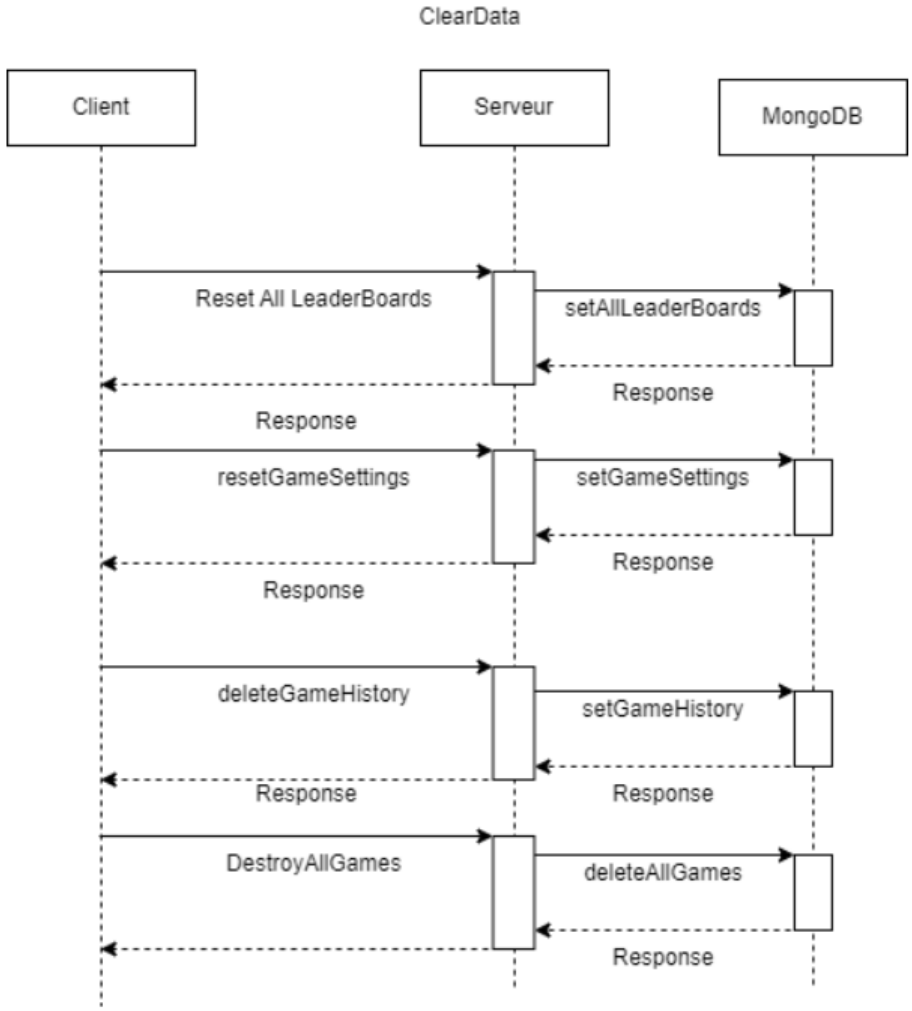
Note: Les GameSettings sont stockés dans les fichiers de jeu, ils sont donc get en même temps que les images par le serveur, c'est-à- dire au tout début d'une partie.

f. Diagramme de séquence: obtention d'indices



Note: Les deux premiers indices n'ont pas besoin d'accéder à la db car les images sont déjà sur le serveur. Pour le troisième indice, le serveur fetch quel type d'indice est nécessaire (flèche ou cercle). Ensuite, le serveur fetch de mongoDB l'image appropriée pour mettre en évidence la différence.

g. Diagramme de séquence: modification des paramètres de jeu contre la montre par le panneau de configuration

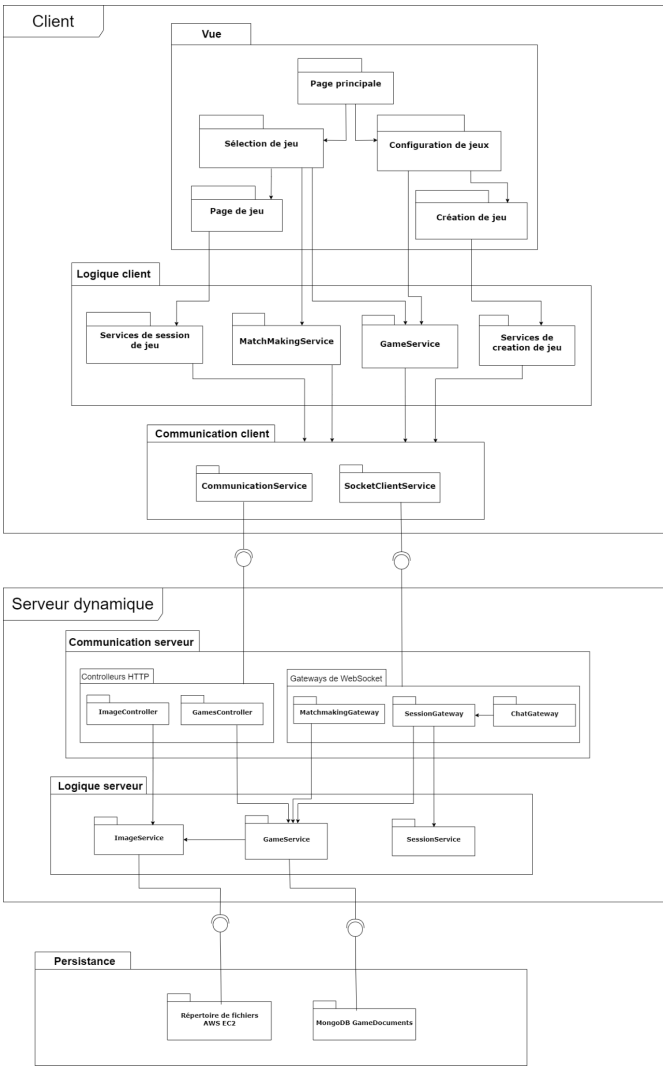


Note: Ces 4 requêtes ne sont pas forcément séquentielles, elles sont arrangées ici dans le même graphique par économie d'espace.

4. Vue logique

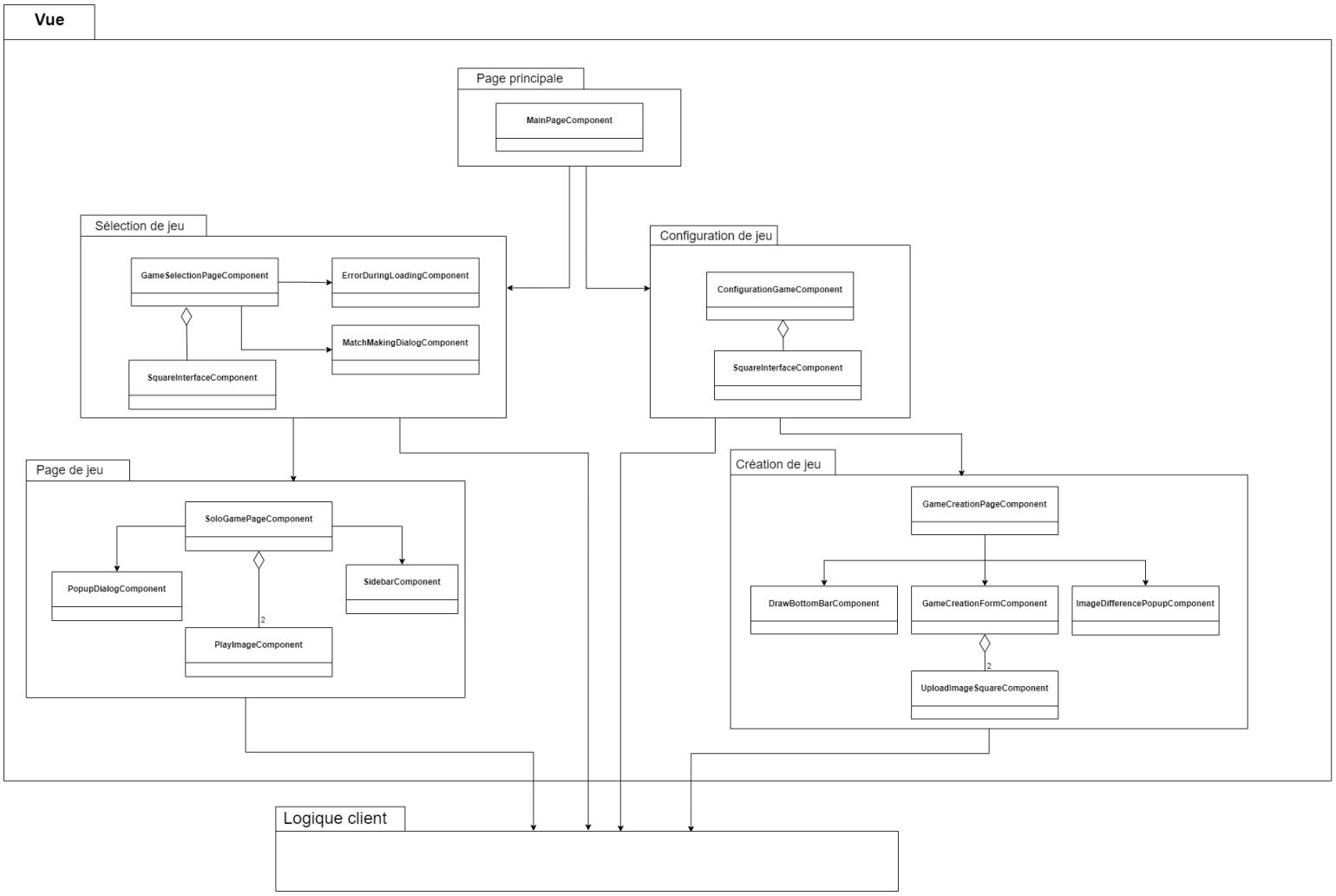
*Note: Nous utilisons le français pour les noms de paquetages, mais l'anglais quand il s'agit d'une classe spécifique de notre code puisqu'on programme en anglais (comme GameService, SessionGateway…).

a. Diagramme de paquetage général du projet:



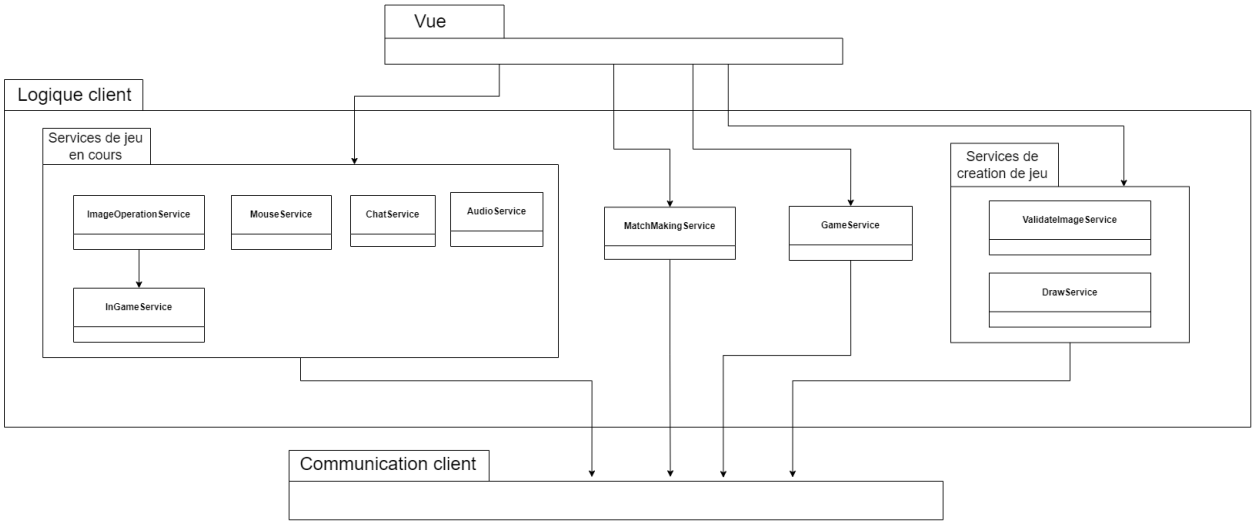
b. Diagramme de paquetage: Vue

Vue
Ce paquetage comprend l'ensemble du visuel de l'application web client. Il est composé de 5 types de pages, qui ont chacune plusieurs composantes Angular. Les pages de sélection de jeu et de la page de jeu offrent (ou offriront au sprint 3) des variations visuelles en fonction du mode de jeu, c'est-à-dire solo ou multijoueur, et mode classique ou temps limité.



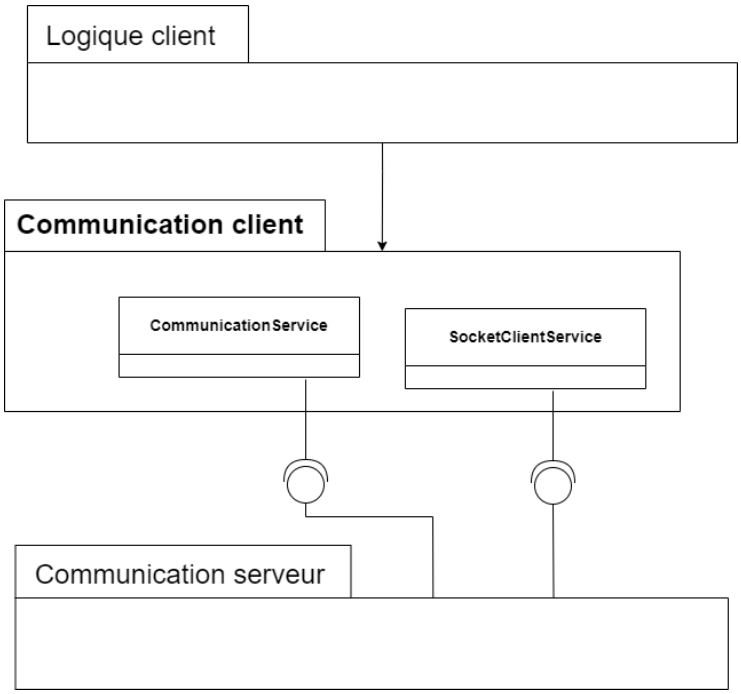
c. Diagramme de paquetage: Logique Client

Logique Client
Ce paquetage contient les services de logique du côté client, pour la gestion de jeux, le matchmaking. Cela inclut également la gestion d'un jeu en cours (donc une session), et la logique de création de jeu et de modification d'avant-plan d'images.



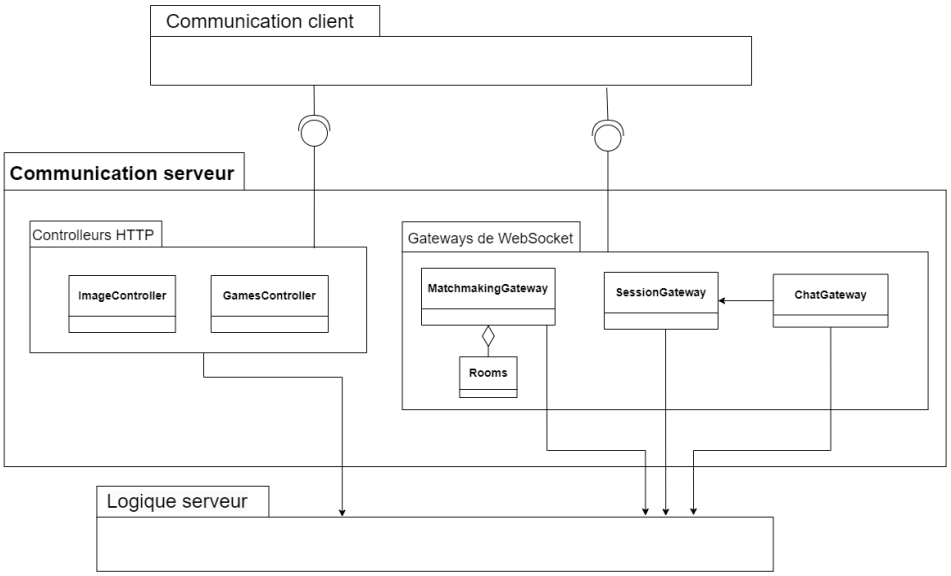
d. Diagramme de paquetage: Communication client

Communication client
Le paquetage de communication client se charge de communiquer avec le serveur dynamique en utilisant 2 protocoles différents. Le premier est le protocole HTTP, à travers CommunicationService, et le deuxième et les WebScocket à travers SocketClientService. Ces 2 méthodes opèrent indépendamment.



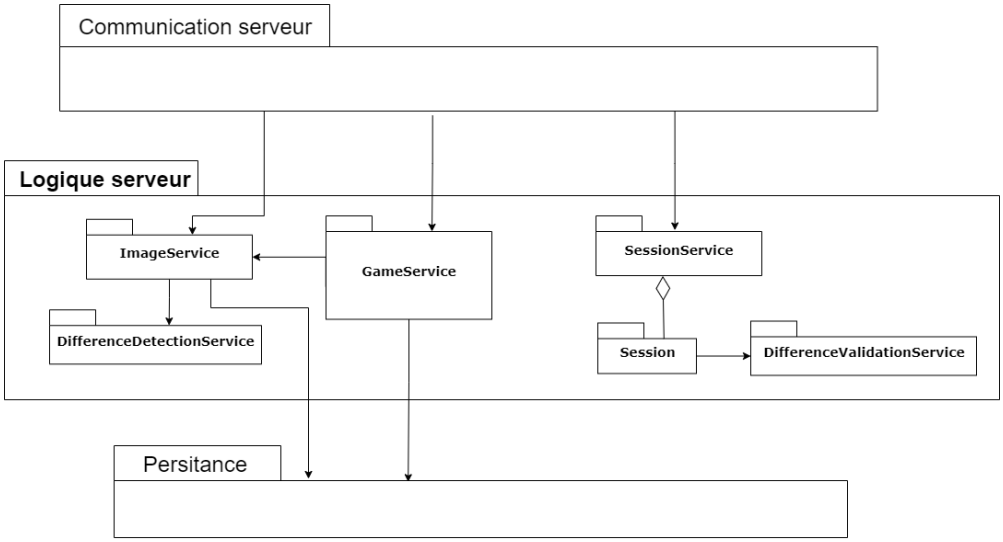
e. Diagramme de paquetage: Communication serveur

Communication serveur
Du côté du serveur, le paquetage de communication permet la réception de messages HTTP des clients à travers les controlleurs HTTP et les gateway WebSockets. Le sous-paquetage controlleurs HTTP permet la création et la recherche de jeux et d'images, donc les ressources du serveur dynamique. Le sous-paquetage de gateway WebSockets regroupe quant à lui la gestion d'énènements WebSocket, qui permettent le matchmaking des jeux multijoueurs, la gestion des sessions de jeu et les fonctionnalités de clavardage.



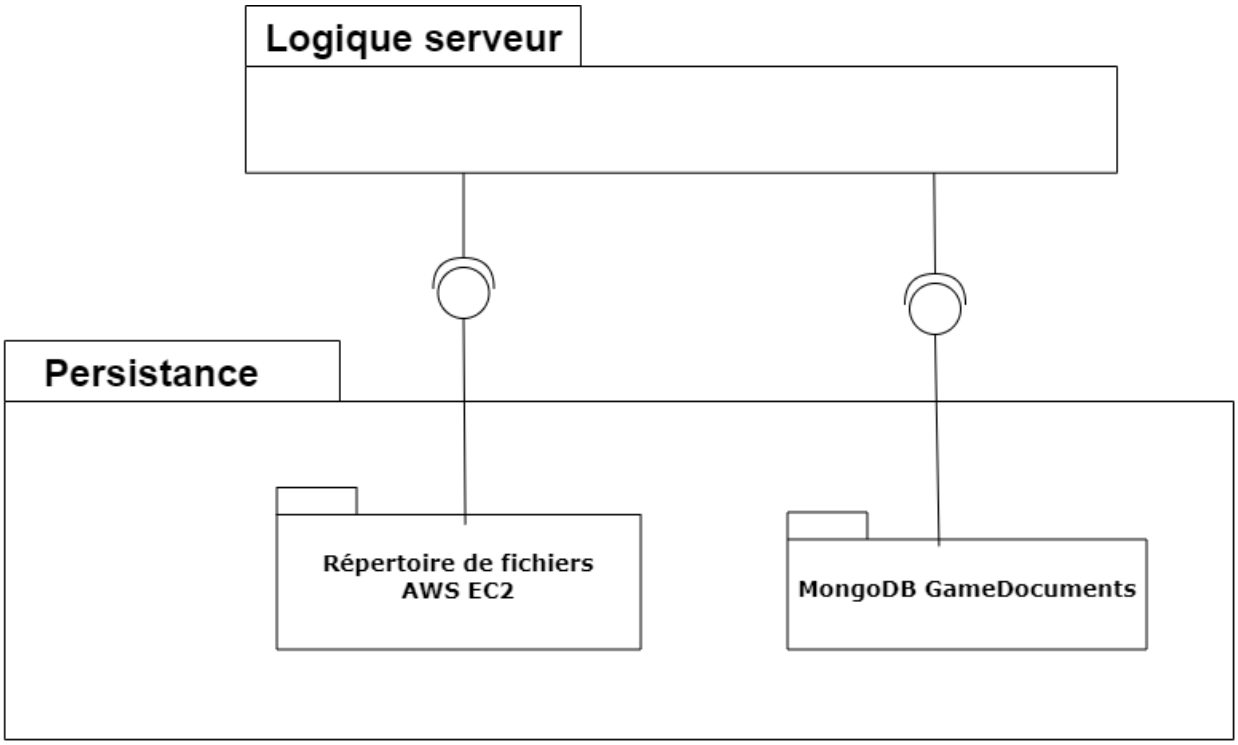
f. Diagramme de paquetage: Logique serveur

Logique serveur
Le paquetage de logique serveur se charge de la gestion de toute la "business logic" du serveur dynamique. Ce paquetage reçoit et fait la validation de tout nouveau jeu, et analyse les différences des images envoyées avec le DifferenceDetectionService. Il gère la création et la suppression de jeux, et gèrera aussi la modification des paramètres et la réinitialisation des scores d'un jeu. Ces fonctionnalités sont gérées à partir du GameService en communiquant avec la persistance. Il s'occupe ensuite de traiter le progrès de toute session de jeu en cours dans SessionService, en validant les différences trouvées et en notifiant les joueurs des évènements comme la fin du jeu lorsque les différences sont trouvées ou le temps est écoulé dans le mode temps limité. Chaque session est traitée en parallèle dans un objet Session indépendant.



g. Diagramme de paquetage: Persistance

Persistance
Enfin, la persistance correspond à la partie de l'architecture qui s'occupe de sauvegarder toutes les informations nécessaires pour jouer un jeu. Les informations des jeux tels leurs nom, les meilleurs scores, et les temps de pénalités par indice sont stockés dans une base de donnée MongoDB, gérée à travers un objet GameDocuments. Les images et les listes de diffjrences par jeu sont sauvegardées en tant qu'images BMP et fichiers JSON respectivement, dans le répertoire de dossier de notre machine EC2 de AWS.



5. Vue de déploiement

a. Diagramme de déploiement