203

Protocole de communication

Version 1.0

Historique des révisions

Date	Version	Description	Auteur
aaaa-mm-jj	x.x	<Détails précis du travail effectué>	<Nom>
2023-03-17	1.0	Communication Client/Serveur et Description des paquets complétés	Maxime Aspros

Table des matières

Protocole de communication

1. Introduction

L'intention de ce document est de présenter les solutions apportées par l'équipe 203 pour la communication de la version finale du projet. L'implémentation et les raisons d'utilisation de chacun des moyens de communication sont détaillées dans la première partie et la seconde partie traite la description des paquets utilisés au sein des protocoles de communication.

2. Communication client-serveur

Fonctionnalité	Moyen de communication
Clavardage et messages de partie (global)	Socket.io, notamment l'événement 'messsageBetweenPlayers' pour les messages entre les joueurs Les messages globaux sont initiés par une partie en cours qui envoie un événement au SocketManagerService. La classe MatchManagerServer, contenant la liste des parties en cours, s'occupe de transmettre le message global à toutes les parties concernées.
Chargement de l'historique	Le client envoie une requête HTTP au serveur dynamique. Celui-ci accède à MongoDB et renvoie l'historique au client.
Création d'un jeu	Génération de l'image des différences: le client envoie les deux images au serveur dynamique avec une requête HTTP. C'est ImageProcessingController qui s'occupe de traiter les images et retourner au client l'image en noir et blanc. La fiche de jeu n'est pas encore générée, puisqu'il faut d'abord valider le nombre de différences. Envoi du titre du jeu: On envoie une requête HTTP réceptionnée par GamesController. Ensuite, GameStorageService sauvegarde définitivement les images localement et on stocke la fiche de jeu sur MongoDB.
Création, fonctionnement, et enregistrement d'une partie (Mode Classique)	Création et enregistrement d'une partie: Une salle Socket.io est créée à partir du moment où un joueur clique sur "Jouer" ou "Créer" / "Rejoindre" depuis la page de sélection de jeu. Une fois dans la Registration Page, le premier joueur peut entrer son nom. Ensuite, il attend à ce qu'un autre joueur demande de jouer avec lui. Une fois que le *host* (l'hôte est toujours le joueur 1) accepte cette demande, le joueur 2 est définitivement enregistré dans la partie. Plus spécifiquement, c'est la méthode "setMatchPlayer" de la classe MatchManagerService qui s'occupe de cette fonctionnalité. Données du jeu: Chaque fiche de jeu contient un

	identifiant unique, qui est accessible depuis l'URL de la page Classique. Le client envoie une requête HTTP (contenant l'identifiant unique) afin de récupérer du serveur les données du jeu ('GameData'). Détection d'une différence: L'événement 'validateDifference' de SocketManagerService prend en paramètre un booléen indiquant le joueur responsable du clic. Cela permet d'isoler les interactions des deux joueurs tout en synchronisant les images.
Historique des parties jouées	Le client envoie une requête HTTP au serveur, qui ensuite demande à la classe GameStorageService de mettre à jour ou de récupérer l'historique des parties jouées depuis MongoDB. Les meilleurs temps sont aussi consultés en même temps (voir ci-dessous).
Meilleurs temps	Les meilleurs temps sont aussi conservés sur MongoDB, et accédés de la même façon que l'historique.
Constantes de jeu	Les constantes de jeux sont aussi conservées sur MongoDB, et accédées de la même façon que l'historique. On note qu'il faut fournir les constantes de jeu au client de la même façon que la fiche de jeu (GameData) afin de ne pas modifier les constantes d'une partie en cours.
Reprise Vidéo	Au cours de la partie, le client doit continuellement envoyer des événements Socket.io à chaque fois qu'un joueur fait une action pour l'enregistrer. Une fois qu'un joueur active le mode Reprise, il faut donc renvoyer au client la liste des actions effectuées pour les diffuser en ordre chronologique.

3. Description des paquets

Requêtes HTTP:

Méthode: GET

Route: ‘/api/games/fetchGame/:id’

Corps: Rien

Paramètres: Aucun

Contenu dans le corps de la réponse: Objet GameData ainsi que deux images

Code de retour: 200/404

Méthode: GET

Route: ‘/api/games/:id’

Corps: Rien

Paramètres: Aucun

Contenu dans le corps de la réponse: Jusqu'à 4 fiches de jeu, ainsi qu’une variable indiquant le nombre de jeux envoyés.

Code de retour: 200/404

Méthode: POST

Route: ‘/api/games/saveGame’

Corps: Objet GameData ainsi que 2 images

Paramètres:

Contenu dans le corps de la réponse: Le titre du jeu.

Code de retour: 201/404

Méthode: DELETE

Route: ‘/api/games/deleteAllGames’

Corps:

Paramètres:

Contenu dans le corps de la réponse: La taille de la liste des jeux.

Code de retour: 200/404

Méthode: DELETE

Route: ‘/api/games/:id’

Corps:

Paramètres:

Contenu dans le corps de la réponse: La taille de la liste des jeux.

Code de retour: 200/404

Méthode: POST

Route: ‘/api/image_processing/send-image’

Corps: 2 images .bmp

Paramètres:

Contenu dans le corps de la réponse: Une image .bmp en noir et blanc avec les differences, ainsi que le nombre de différences.

Code de retour: 200/404

Evénements Socket:

Nom d'événement: ‘registerGameData’

Source: ClassicPageComponent

Contenu: Un objet de type GameData est envoyé par le client, et sauvegardé dans socket.data.

Nom d'événement: ‘validateDifference’

Source: ClassicPageComponent

Contenu: Le client envoie la position du clic, ainsi que la liste des différences trouvées et un booléen indiquant le joueur concerné. Le serveur utilise la méthode getDifferenceIndex de MatchingDifferencesService afin de valider la différence ou non, et met à jour la liste des différences si besoin. La validation est ensuite retournée au client avec l'événement ‘validationReturned’.

Nom d'événement: ‘validationReturned’

Source: SocketManagerService

Contenu: Le serveur renvoie la liste des différences, l’indice de la différence trouvée, et un booléen indiquant le joueur qui a cliqué sur une différence.

Nom d'événement: ‘disconnect’

Source: Client (toutes les pages qui utilisent un Socket)

Contenu: Aucun

Utilité: Permet de prévenir le MatchManagerService d’une déconnexion afin d'attribuer la victoire à un joueur.

Nom d'événement: ‘createMatch’

Source: La fonction ‘createGame’ de MatchmakingService. Celle-ci est appelée quand le client appuie sur “Jouer” ou “Créer” depuis la page de sélection de jeu:

Contenu: L’identifiant unique de la fiche de jeu, ainsi que le SocketId de la salle, c’est-a-dire le SocketId du joueur 1 qui lance la partie.

Nom d'événement: 'setMatchType'

Source: Page de sélection de jeu, juste après l’appel à ‘createMatch’.

Contenu: L’identifiant de la partie, ainsi que le type de la partie (Solo, 1v1, etc…)

Nom d'événement: ‘setMatchPlayer’

Source: La fonction ‘setCurrentMatchPlayer’ de MatchMakingService. Celle-ci est appelée quand un joueur entre son nom dans la RegistrationPage.

Contenu: L’identifiant de la partie, ainsi qu’un objet Player, qui contient le playerId et le nom du joueur.

Nom d'événement: ‘requestToJoinMatch’

Source: RegistrationPage

Contenu: L’identifiant de la partie, ainsi qu’un objet Player, qui contient le playerId et le nom du joueur. Envoie l'événement ‘incomingPlayerRequest’ à la salle du match concerné.

Nom d'événement: ‘incomingPlayerRequest’

Source: SocketManagerService

Contenu: Un objet Player, qui contient le playerId et le nom du joueur. Le client ajoute le joueur à la liste des joueurs en attente.

Nom d'événement: ‘cancelJoinMatch’

Source: RegistrationPage

Contenu: L’identifiant de la partie, ainsi qu’un objet Player, qui contient le playerId et le nom du joueur. Appelle la fonction ‘sendJoinMatchCancel’.

Nom d'événement: ‘sendIncomingPlayerRequestAnswer’

Source: RegistrationPage

Contenu: L’identifiant de la partie, un objet Player, et un booléen indiquant si le joueur a été accepté ou non. Envoie l'événement ‘incomingPlayerRequestAnswer’.

Nom d'événement: ‘incomingPlayerCancel’

Source: SocketManagerService

Contenu: L’identifiant du joueur. Sert à retirer un joueur de la liste des joueurs en attente.

Nom d'événement: ‘incomingPlayerRequestAnswer’

Source: SocketManagerService

Contenu: L’identifiant de la partie, un objet Player, et un booléen indiquant si le joueur a été accepté ou non.

Nom d'événement: ‘deleteAllGames’

Source: Page de configuration

Contenu: Rien, envoie un événement pour relancer la page avec les jeux.

Nom d'événement: ‘deletedGame’

Source: Page de configuration

Contenu: Rien, redirige les utilisateurs qui étaient dans la RegistrationPage du jeu supprimé.

Nom d'événement: ‘sendingMessage’

Source: ClassicPage

Contenu: Le nom de l’utilisateur ayant envoyé le message, le contenu du message, ainsi qu’un booléen indiquant le joueur ayant envoyé le message.