<107>

Protocole de communication

Version 3.0

Historique des révisions

Date	Version	Description	Auteur
2023-03-20	1.0	Rédaction de la documentation (fin du sprint 2)	Sébastien Roy
2023-04-18	2.0	Ajustement des changements du sprint 3	Majeed Abdul Baki
2023-04-20	3.0	Ajustement des changements sprint 3 de websocket et interfaces	Victor Gilbert

Table des matières

Protocole de communication

1. Introduction

Le projet AmongoDB, un jeu de recherche de différences entre 2 images, est un jeu web qui offre de nombreuses fonctionnalités et modes de jeu, dont des parties multijoueur. Cela nécessite donc un système de communication entre différnts logiciels, suivant des protocoles comme WebSocket et HTTP notamment.

Ce document présente donc le protocole de communication du projet, c'est-à-dire une synthèse des différents protocoles utilisés pour la communication, ainsi que le format et les conventions utilisés pour l'envoi des différents paquets à travers le réseau.

Ce document est divisé en 2 parties contenant 1 tableau et 2 tableaux respectivement.

La première partie contient le tableau 2.1 qui indique les décisions de protocole de transmission d'information entre l'application client et le serveur. Ce tableau présente chaque fonctionnalité de notre système, leur protocole de communication et le raisonnement derrière nos décisions de protocole.

La deuxième partie de ce document présente le format des données qui sont échangées entre notre application client et le serveur. Le tableau 3.1 est dédié aux interfaces qui sont échangées par le protocole HTTP, alors que le tableau 3.2 présente les interfaces qui sont échangées par le protocole WebSocket.

2. Communication client-serveur

Tableau 2.1 - Moyens de communication

Sujet	**Protocole**	**Raisonnement**
Création de jeux	HTTP	- C'est toujours le client qui initialise les demandes de création de jeu. Le serveur n'aura qu'à répondre aux requêtes du client.
Indices de jeu	WebSocket	- Permet de combiner au Sessions qui utilisent déjà une connexion WebSocket, en n'ajoutant qu'un message. - Permet d'envoyer des indices lorsque demandé en partie solo.
Suppression de jeux	WebSocket/ HTTP	- HTTP permet de combiner la création et la suppression de jeux en un seul contrôleur, puisque ce sont des opérations *stateless* (sans état persistant du client) et donc adaptées. - Websocket permet de notifier les joueurs en attente que le jeu n'existe plus et ne peut donc plus être joué, action initiée par le serveur donc mieux adaptée à du WebSocket.
Historique des parties	WebSocket/ HTTP	- Les WebSockets du système de détection de différences permettent d'enregistrer les coordonnés des essais de l'utilisateur et à quels moments elles ont été faites, ainsi que les résultatas des parties. Ceci est un aller-retour entre client et serveur donc les WebSockets permettent de garder le même état. - HTTP permet de récupérer l'historique présent, ainsi que d'envoyer une demande pour le supprimer. - Est combinée à la logique de détection des différences et écoute pour une demande de la reprise vidéo après que le message de victoire ai été envoyé au joueur par le serveur..
Détection de différences	WebSocket/ HTTP	- Les Websockets permettent au serveur de notifier tous les joueurs qu'une différence a été trouvée grâce au concept de salle. Ceci est aussi un aller-retour entre client et serveur donc les WebSockets permettent de faire persister l'état d'une session de jeu et garder en mémoire les différences déjà trouvées, - Permet au serveur d'envoyer différents messages aux joueurs d'une même session, car ils sont dans la même salle. - HTTP permet de comparer 2 images lors de la création de nouveaux jeux, puisque c'est une opération *stateless*.
Clavardage entre joueurs	WebSocket	- Permet la communication entre joueurs (en passant par le serveur) et se charge également d'envoyer des messages globaux à toutes les parties en cours. - Nous avons choisi les WebSockets car cette communication est bidirectionnelle, et les WebSockets permettent aussi de grouper les utilisateurs pouvant communiquer entre eux dans des salles.
Minuterie de jeux	WebSocket	- Permet d'assurer la validité des temps enregistrés dans les tableaux des meilleurs temps (moins de chance de triche) en gardant la minuterie sur le serveur.

		- Permet de contrôler la mise à jour du temps sur toutes les interfaces utilisateurs à partir du serveur. (le serveur envoie un message à chaque seconde), les WebSockets permettent une synchronisation initiée par le serveur.
Tableaux des meilleurs temps	WebSocket/ HTTP	- Intégré dans notre logique de WebSockets pour la détection de différences. (Lorsqu'on obtient un vainqueur, on enregistre son temps et on le compare aux meilleurs temps) - Mise à jour en temps réel des tableaux des meilleurs résultats pour les clients sur la page de sélection de partie. - Est combiné avec les messages de partie qui sont en WebSocket. - La réinitialisation des meilleurs temps passe par des requêtes HTTP, puisque c'est une demande du client qui n'attend pas de dialogue.
Recherche de parties	WebSocket	- Mise à jour en temps réel de l'interfaces de tous les joueurs cherchant des parties en multijoueur, les WebSockets permettent au client d'attendre une mise-à-jour du serveur sans avoir besoin de faire des demandes en boucle pour avoir les mise-à-jours. - Facilite la communication du serveur aux joueurs dans une session grâce au concept de salle.
Messages de partie	WebSocket	- Permet d'initier l'envoie de messages à tous les joueurs actifs à partir du serveur. Encore une fois, les WebSockets permettent une synchronisation des messages entre tous les clients, puisque le serveur peut envoyer à tous les clients connectés le même message.

3. Description des paquets

Tableau 3.1 - Requêtes HTTP

Route		Méthode	Paramètres	Corps	Code(s) de la réponse	Corps de la réponse
api	/games	POST newGame		name: string, radius: string, files: GameImageInput	200, 400	Game
		GET getAllGames			200, 500	Game[]
		GET getGame	id: string		200, 404, 500	Game
		DELETE deleteById	id: string		200	
		DELETE* deleteAllGames			204	
	/games/constants	GET* getGameConstants			200	GameConstants
		PATCH* configureConstants		gameConstsInput: GameConstantsInput	204, 400	
	/games /leaderboards	DELETE resetAllLeaderboards			204, 500	
		DELETE resetLeaderboard	id: string		204	
	/history	GET* getHistory			200, 500	GameHistory[]
		POST* addToHistory	newHistoryEntry: GameHistory		200, 400, 500	GameHistory
		DELETE* deleteHistory			200, 500	
	/images	GET getAll			200	number[]
		GET serveImage	id: string		200, 404	StreamableFile
	/images/compare	POST		radius: string,	200,	Promise<

		compareImages		files: GameImageInput	400	ImageComparisonResult>

* fonctionnalités implémentées au sprint 3

Tableau 3.2.1 - Événements WebSocket

Dossier	Source	Nom d'évènement	Contenu	Contenu du retour
Matchmaking	Client	startMatchMaking	gameId: string	
		someOneWaiting	gameId: string	response: boolean
		roomCreatedForThisGame	gameId: string	response: boolean
		leaveWaitingRoom	gameId: string	
		joinRoom	{ gameId: string, playerName: string }	
		acceptOpponent	opponentName: string	response: boolean
		anyGamePlayable		
		rejectOpponent	{ gameId: string, playerName: string }	
	Serveur	opponentJoined	opponentName: string	
		opponentLeft		
		acceptOtherPlayer	opponentName: string	
		rejectOtherPlayer	playerName: string	
		roomReachable		
		sessionId	sessionId: number	
		updateRoomView		
Session	Client	getClientId		id: string
		closeSession	sessionId: number	sessionId: number
		startClassicSession	data: StartSessionData	(sessionId: number) ou undefined
		startLimitedTimeSession	isSolo: boolean	(sessionId: number) ou undefined
		leaveRoom		
		submitCoordinateSoloGame	data: [sessionId: number, coordinates: Coordinates]	result: GuessResult

		submitCoordinateMultiGame	data: [sessionId: number, coordinates: Coordinates]	
		submitCoordinateLimitedTimeGame	data: [sessionId: number, coordinates: Coordinates]	
		cheatGetAllDifferences	sessionId: number	(response: Coordinates[][]) ou undefined
		playerLeft	sessionId: number	
		playerName	clientName: string	
		askForClue	sessionId: number	response: Clue

* fonctionnalités implémentées au sprint 3

Tableau 3.2.2 - Événements WebSocket (suite du tableau 3.2.1)

Sujet	Source	Nom d'évènement	Contenu	Contenu du retour
Session	Serveur	playerWon	winnerInfo: WinnerInfo	
		sessionId	sessionId: number	
		differenceFound	differenceFound: GuessResult	
		opponentLeftGame		
		provideName		
		timerUpdate	time: string	
		newGame	data[newGame: Game, nDifferencesFound: number]	
		endedGame	timer: string	
Chat	Client	giveName	playerName: string	
		messageFromClient	message: Message	
	Serveur	systemMessageFromServer	systemMessage: SystemMessage	
		messageFromServer	newMessage: Message	
		broadCastHighScore	highScoreMessage: string	
		giveClientID	receivedId: string	

* fonctionnalités implémentées au sprint 3

Tableau 3.3 - Interfaces principales utilisées

Nom de l'interface	Attributs	Description
Game	name: string imageMain: number imageAlt: number scoreBoardSolo: [string, number][] scoreBoardMulti: [string, number][] isValid: boolean isHard: boolean differenceCount: number radius: number id: string	Sauvegarde et récupére les informations persistantes d'un jeu, comme son nom, les images qui y sont associées, la difficulté et les meilleurs scores.
GameConstants	time: number penalty: number reward: number	Les 3 constantes de jeu globales au projet: le temps de départ en temps limité, le temps de pénalité par indice et le temps offert lorsqu'on trouve une différence en temps limité.
GameConstantsInput	time: string penalty: string reward: string	Les constantes venant du client, qui sont transmises en tant que chaîne de caractères.
GameImageInput	mainFile: Express.Multer.File[] altFile: Express.Multer.File[]	Paire d'images d'un jeu, avec les types de fichiers utilisés par NestJS.
GameHistory	gameId: string startDateTime: string duration: string gameMode: string playerOne: string playerTwo: string	Une entrée de l'historique avec les informations d'une partie.
ImageComparisonResult	isValid: boolean isHard: boolean differenceCount: number differenceImageBase64: string	Le résultat d'une comparaison entre 2 images, utilisé lors de la création d'un jeu.
Coordinates	x: number y: number	Un point sur l'image en 2 dimensions.
GuessResult	isCorrect: boolean differencesByPlayer: [userSocketId: string, nDifferences: number][] differencePixelList: Coordinate[] winnerName: string | undefined	Le résultat d'un essai de recherche de différence lors d'une partie, qui permet au client de savoir si c'est un coup réussi et de clignoter les pixels de la différence.
StartSessionData	gameId: string isSolo: boolean	Les informations nécessaires pour créer une session classique
Clue	coordinates: Coordinate[] nbCluesLeft	Les informations nécessaire pour montrer un indice

WinnerInfo	name: string socketId: string	Les informations sont données au client lorsqu'un joueur gagne.
SystemMessage	playerName: string systemCode: string	Un message système à afficher dans le clavardage d'une partie
Message	socketId: string isFromSystem: boolean sessionID: number author: string time: number message: string	Un message à afficher dans le clavardage d'une partie.