202

Protocole de communication

Version 1.0

Historique des révisions

Date	Version	Description	Auteur
2023-11-05	1.0	Création du document	Mohamed Laraki, Mohamed Lamine Gning, Zakarya Khnissi, Halima Sadia Diarra, Bailly Jonathan Clovis Yaro, Youness Lazzali
2023-11-07	1.1	Modifications des codes de retour 500 en codes de retour 404	Mohamed Laraki, Halima Sadia Diarra

Table des matières

Table des matières

Protocole de communication

1. Introduction

Tout au long de la session, notre objectif aura été de développer une application web de type « jeu-questionnaire ». Une certaine interactivité avec l'utilisateur est de mise pour ce type d'application. Elle se base sur une multitude de communications entre les clients et le serveur pour les différentes fonctionnalités du projet comme le clavardage, la gestion des réponses, etc.
Ce document décrit ainsi les différents protocoles de communication utilisés dans ce projet. Nous allons, tout d'abord, décrire nos choix de moyen de communication entre les différents clients et le serveur. Ensuite, nous décrirons les différents types de paquets utilisés au sein de nos protocoles de communication.

2. Communication client-serveur

La communication bidirectionnelle entre les clients et le serveur dans une application web est un aspect fondamental afin d'offrir à l'utilisateur une expérience fluide et optimale. Le choix des protocoles de communication est donc essentiel afin d'envoyer certains types de requêtes/messages de manière idéale. Ainsi, nous avons choisis d'utiliser deux protocoles de communication : HTTP et WebSocket.

Les requêtes HTTP permettent un envoi de données massives. C'est pour cela que nous l'utilisons notamment pour la vue administrateur ou la vue de test du jeu-questionnaire sélectionné. Les paquets possédaient le type de méthode (GET, POST), la route, le corps et le(s) paramètre(s) et une réponse associée (200 : OK, 404 : Not Found, 501 : Not Implemented, …). Tout ceci sera expliqué lors de la description des différents paquets.

WebSocket est un protocole de communication permettant un envoi plus rapide de données que les requêtes HTTP, mais la quantité de données envoyée autorisée est moindre. Ceci permet un échange de données quasi-instantané qui sera extrêmement utile pour le clavardage ou la soumission des réponses au moment de jouer, par exemple. Plusieurs opérations sont mises à disposition pour offrir une communication quasi-instantanée. On peut notamment citer :

- .emit() : fonction permettant le transfert de données à tous les utilisateurs de la room (messages ou autres),
- .on() : fonction permettant la réception des données émises à tous les utilisateurs de la room.

Les attributs principaux (client.id et client.rooms) sont essentiels pour toutes ces opérations. Le premier permet de différencier les utilisateurs présents dans la même room en fonction de son rôle (Organisateur ou Joueur) et entre eux tandis que le second permet de lister les différentes parties créées pour les jeux rendus visibles.

HTTP	WebSocket
Chercher la liste de tous les jeux	Créer la room
Chercher un jeu par son ID	Joindre la room
Ajouter un jeu dans la liste des jeux	Vérifier l'authentification du code d'accès à la partie
Mettre à jour la liste des jeux	Verrouiller/Déverrouiller la partie
Supprime un jeu de la liste des jeux	Supprimer un joueur de la room (banni ou abandon)
Créer une partie avec le jeu sélectionné	Commencer la partie
Vérifier l'authentification du code d'accès à la partie	Sélectionner/déselectionner une/des réponse(s)
Vérifier l'authentification du nom du joueur	Passer à la prochaine question
Historique des parties	Soumettre les réponses aux questions
	Montrer les résultats à la fin de la partie
	Clavarder
	Soumettre les réponses notées des QRL
	Activer/Désactiver le chat pour un joueur
	Changer l'état du compte à rebours (Mode panique, Pause)
	Détecter lorsqu'un joueur écrit dans le champ d'une QRL

3. Description des paquets

3.1 Paquets HTTP

Méthode	URI	Paramètres URI	Description	Corps de la Requête	Corps de la Réponse	Code(s) de retour
GET	/games	—	Récupérer tous les jeux de la liste des jeux	—	Game[]	Succès: 200 Échec: 404
GET	/games/${id}	id: string	Récupérer les informations d'un jeu	—	Game	Succès: 200 Échec: 404

POST	/games/send	—	Ajoute un nouveau jeu à la liste	{
				message : Game
				}
					Succès:
					{
					game: Game
					}
					
					Échec: —
						Succès: 201
						Échec: 404

POST	/match/create	—	Crée une partie	{
				gameId : string
				}
					Succès :
					{
					matchId: string
					}
					Échec :
					{
					e.message: string
						Succès : 200
						Échec : 500

					}	
POST	/match/check/code	—	Vérifie le code d'accès	{ code: string }	Succès : { string } Échec : { e.message: string }	Succès : 200 Échec : 403, 404
POST	/match/check/username	—	Vérifie le nom d'utilisateur	{ code: string, username: string }	Succès : { message: MatchInfo } Échec { e.message: string }	Succès : 200 Échec : 403, 409
PUT	/games/${message.id}	message.id : string	Modifie un jeu	{ message : Game }	Succès: { game: Game } Échec: —	Succès : 200 Échec : 404
DELETE	/games/${id}	id: string	Supprime un jeu	{ message : Game }	Succès: { boolean } Échec: { boolean }	Succès : 204 Échec : 404
GET	/games/history	—	Récupérer l'historique des parties terminées	—	FinishedMatch[]	Succès: 200 Échec: 500

3.2 Paquets WebSockets

Événement	Source	Description	Données	Événements Potentiellement déclenchés

createWaitingRoom	Client	Demander la création de la vue d’attente	String	waitingRoomCreated
waitingRoomCreated	Serveur	Notifier de la création de la vue d’attente.	String	—
gameJoined	Client	Demander qu’un utilisateur joigne une partie spécifique	String	updatePlayers
lockRoom	Client	Demander le verrouillage de la partie	String	RoomLocked
unlockRoom	Client	Demander le déverrouillage de la partie	String	RoomUnlocked
RoomLocked, RoomUnlocked	Serveur	Notifier de l’état de verrouillage	-	—
removePlayer	Client	Demander le retrait d’un joueur	Object	playerRemoved, updateBlacklist, kickedOut
playerRemoved	Serveur	Notifier du retrait d’un joueur	String	—

updateBlacklist	Serveur	Notifier de la mise à jour de la liste noire	String[]	—
kickedOut	Serveur	Notifier du bannissement du joueur	-	—
StartGame	Client	Demander le début de la partie	String	GameStarted, QuestionCountdown
GameStarted	Serveur	Notifier du début du jeu	-	—
QuestionCountdown	Serveur	Démarrer l'activation du compte à rebours avant le début de la partie	TimeInfo	—
LeaveGame	Client	Demander de quitter la partie en cours	String	OrganizerLeft (si c'est l'organisateur), playerRemoved, kickedOut, updateBlacklist, PlayerResult, allPlayerResults
OrganizerLeft	Serveur	Notifier le retrait de l'organisateur de la partie	-	—
SubmitAnswers	Client	Demander la soumission des réponses	Object	allPlayersResults, playerResult

PlayerResult	Serveur	Notifier la réception des résultats de chaque joueur	PlayerResult	—
allPlayersResults	Serveur	Notifier la réception de tous les résultats des joueurs	Player[]	—
ShowResults	Client	Demande l'affichage des résultats de tous les joueurs	String	FinalResults
FinalResults	Serveur	Notifier l'affichage des résultats de tous les joueurs	Player[]	—
NextQuestion	Client	Demander de passer à la question suivante	String	NextQuestion(côté serveur), QuestionCountdown, PlayerResult, allPlayerResults
NextQuestion	Serveur	Notifier le passage à la question suivante	Number	—
RoomMessage	Client	Demander d'envoyer un message	Object	RoomMessage
RoomMessage	Serveur	Notifier la réception du message	String	—

enableChat	Client	Demander d’activer le chat pour un joueur	String	chatEnabled
chatEnabled	Serveur	Notifier l’activation du chat	-	—
disableChat	Client	Demander de désactiver le chat pour un joueur	String	chatDisabled
chatDisabled	Serveur	Notifier la désactivation du chat	-	—
changeTimerState	Client	Demander de changer l’état du minuteur	TimerState	timerStateChanged
timerStateChanged	Serveur	Notifier du changement de l’état du minuteur	TimerState	—
gradedAnswers	Client	Envoyer les réponses des joueurs aux QRL notées par l'organisateur		playerResult
writeInField	Client	Informer le serveur que l'utilisateur a écrit dans le champ de réponse pour une QRL	-	userWroteInField

userWroteInField	Serveur	Informer l'organisateur que l'utilisateur a écrit dans le champ de réponse pour une QRL	String	—

3.3 Interfaces

Nom	Description	Structure

Game	Informations sur un jeu	{
		id: string;
		title: string;
		description?: string;
		duration: number;
		questions: Question[];
		lastModification: string;
		visible: boolean;
		}

MatchInfo	Informations sur une partie	{
		id: string;
		gameId: string;
		players: Player[];
		blackList: string[];
		currentQuestionIndex: number;
		beginDate: Date;
		status: MatchStatus;
		}

MatchStatus	Statut de la partie	{
		WAITING: number,
		FINISHED: number,
		LOCKED: number,
		}

Player	Informations sur le joueur	{
		clientId: string;
		username: string;
		score: number;
		bonusCount: number;
		}

PlayerResult	Informations sur les résultats du joueur	{
		score: number;
		isCorrect: boolean;
		hasBonus: boolean;
		correctChoices: number[];
		}

TimeInfo	Informations sur le temps	{
		state: boolean;
		msg: string;
		}

FinishedMatch	Informations sur les parties finies	{
		gameName : string
		beginDate : Date
		playersAmount : number
		bestScore : number
		}

TimerState (enum)	Statut du minuteur	{
		PAUSED,
		RESUMED,
		PANIC
		}