106

Protocole de communication

Version 1.3

Historique des révisions

Date	Version	Description	Auteur
2023-11-01	1.0	Première version de la communication client-serveur	Eugène Vachon
2023-11-01	1.1	Énumération des interfaces utilisés dans l'application (3.3)	Noah Namoko
2023-11-02	1.2	Paquets HTTP et WebSockets ainsi qu'une brève introduction	Eugène Vachon
2023-11-07	1.3	Ajustement des paquets WebSockets et des interfaces	Eugène Vachon

Table des matières

Protocole de communication

1. Introduction

Ce document présente les différentes fonctionnalités concernant la communication entre le client et le serveur. En premier lieu, la communication sera expliquée. De plus, le protocole utilisé pour chaque fonctionnalité sera justifié. En second lieu, l'architecture des paquets HTTP et WebSockets pour chaque route sera illustrée dans des tableaux. En troisième lieu, les interfaces utilisées dans le code sont documentées dans un tableau à la fin.

2. Communication client-serveur

Le protocole HTTP est utilisé pour la vérification du mot de passe pour la page d'administration des jeux. La raison derrière ce choix est que le client doit obtenir une réponse suite à la requête. Le mot de passe est soit valide ou invalide, le serveur envoie donc une réponse directement. Le protocole HTTP est aussi utilisé pour l'obtention de la liste des jeux questionnaires car le serveur répond directement avec la liste ou un code d'erreur. Pour l'exportation de jeux questionnaires, nous avons aussi choisi le protocole HTTP car il permet de retourner le jeux exporter ou un code d'erreur en cas de problème. Nous avons utilisé une requête HTTP "patch" pour la modification et le changement de la visibilité des jeux car il est simple de savoir si l'opération à fonctionné avec les code de retour. Pour l'ajout et la suppression des jeux dans la liste, nous avons opté pour HTTP pour les mêmes raisons. L'obtention et la réinitialisation de l'historique des parties se fait par HTTP pour l'utilisation des codes de retour. De plus, pour toutes ces actions, la communication entre clients n'est pas nécessaire, il est donc préférable d'utiliser HTTP au lieu de WebSocket.

Le protocole WebSocket, quant à lui, est utilisé pour créer ou rejoindre une salle. La raison est que ce protocole permet la communication avec plusieurs clients bien plus simple grâce aux "room". De plus, ce protocole permet au serveur d'envoyer des signaux au client de lui-même. La validation du nom d'un joueur est elle aussi faite par WebSocket car il faut vérifier si le nom est valide dans une salle en particulier. Nous utilisons aussi le protocole WebSocket pour obtenir la liste des joueurs d'une salle car encore une fois, l'identifiant de la salle est requis. L'opération d'exclusion d'un joueur est effectuée par ce même protocole car cette action se déroule elle aussi à l'intérieur d'une salle. Le verrouillage de la salle suit la même logique. Lorsqu'un joueur quitte ou est déconnecté, le protocole WebSocket est encore une fois utilisé car l'action est reliée à une salle. Pour le commencement de la partie et l'envoi d'un message dans le chat, la raison est qu'il faut que le serveur puisse envoyer de lui-même le message aux clients de la salle. Ensuite, le client envoie les réponses entrées par le joueur au serveur, qui l'envoie à l'organisateur. De la même manière, l'organisateur note les réponses des QRL et les transmet au serveur qui les envoie au joueurs. Ces actions sont gérées par WebSocket car le serveur n'a pas besoin de répondre aux requêtes. De plus, il envoie directement les résultats aux destinataires. Il serait donc inapproprié d'utiliser HTTP. Pour la minuterie lors d'une partie, le protocole WebSocket est plus approprié car il permet d'envoyer de lui-même le temps aux membres de la salle. De plus, lorsque l'organisateur met en pause la minuterie ou active le mode panique, il faut que le serveur passe le message aux joueurs de la salle. La désactivation du clavardage d'un joueur par l'organisateur se fait par WebSocket car c'est une action qui se passe entre deux clients distincts. L'envoi des résultats aux joueurs se fait par ce même protocole car il permet au serveur de les envoyer de lui-même.

3. Description des paquets

3.1 Paquets HTTP

Méthode	URI	ramètres URI	Description	rps de la Requête	Corps de la Réponse	Code(s) de retour
POST	/api/admin/verify-admin-passw	—	Vérifier le mot de passe pour la page	{ password: string	{ valid: boolean }	Succès: 200 Échec: 401

	ord		d'administration	}		
GET	/api/games/	—	Obtenir la liste des jeux questionnaires	—	Succès: [game1: Game, game2: Game, …] Échec: —	Succès: 200 Échec: 400
GET	/api/games/	id: string	Exporter le jeu avec l'id spécifié	—	—	Succès: 204 Échec: 404

PATCH	/api/games/	id: string	Modifier le jeu avec l'id spécifié	{
				modifiedGame: Game
				}
					Succès:
					[
					game1: Game,
					game2: Game,
					…
					]
					
					Échec: —
						Succès: 200
						Échec: 404

PATCH	/api/games /visibility/	id: string	Modifier la visibilité du jeu avec l'id spécifié	{
				isVisible: boolean
				}
					Succès:
					[
					game1: Game,
					game2: Game,
					…
					]
					
					Échec: —
						Succès: 200
						Échec: 404

POST	/api/games/	—	Ajouter un jeu à la liste	{
				game: Game
				}
					Succès:
					[
					game1: Game,
					game2: Game,
					…
					]
					
					Échec:
					{
					message: string
					}
						Succès: 201
						Échec: 400

DELETE	/api/games/	id: string	Supprimer le jeu avec l'id spécifié de la liste	—	—	Succès: 204 Échec: 404
GET	/api/history/	—	Obtenir l'historique des parties jouées	—	Succès: [{ gameName: string, date: Date, playerCount: number, bestScore: number }, …] Échec: —	Succès: 200 Échec: 400
PATCH	/api/history/	—	Réinitialiser l'historique des parties	—	—	Succès: 204 Échec: 400

3.2 Paquets WebSockets

Événement	Source	Description	Données	Événements Potentiellement déclenchés
getPlayers	Client	Obtenir la liste des joueurs de la salle	—	latestPlayerList
leaveLobby	Client	Quitter la salle	—	lobbyClosed noPlayers latestPlayerList
banPlayer	Client	Exclure le joueur de la salle	socketId name	lobbyClosed
toggleLock	Client	Verrouiller la salle	—	lockToggled
joinLobby	Client	Rejoindre la salle	pin	failedLobbyConnection successfulLobbyConnection
createLobby	Client	Créer une salle	game	successfulLobbyCreation failedLobbyCreation

validateName	Client	Vérifier le nom du joueur	nameToValidate	invalidName validName latestPlayerList
startCountDown	Client	Commencer le compte à rebours	initialCount isQuestionTransition gameMode	isQuestionTransition countDown countDownEnd
stopCountDown	Client	Arrêter le compte à rebous	—	—
chatMessage	Client	Envoyer un message dans le chat de la salle	messageData	messageReceived
latestPlayerList	Serveur	Envoyer la liste des joueurs	lobbyDetails	—
lobbyClosed	Serveur	Retirer le joueur de la salle	reason message	joinLobby
failedLobbyConnection	Serveur	Signaler le problème lors de la connection à la salle	message	—
successfulLobbyConnection	Serveur	Signaler la réussite lors de la connection à la salle	game pin	—
invalidName	Serveur	Signaler un nom invalide	message	—
validName	Serveur	Signaler que le nom choisi est valide	name	—
successfulLobbyCreation	Serveur	Signaler la réussite lors de la création d'une salle	pin	—
failedLobbyCreation	Serveur	Signaler l'échec lors de la création d'une salle	reason	—
lockToggled	Serveur	Signaler que la salle a bien été verrouillée	isLocked	—
messageReceived	Serveur	Envoyer le message au joueurs	messageData	—
countDown	Serveur	Envoyer la minuterie au client	countDown	—
submitScore	Client	Envoyer au serveur son score	score	scoreUpdated
showResults	Serveur	Signaler au client de montrer les résultats	—	—
allSubmitted	Serveur	Signaler au client que toutes les réponses ont été soumises	playerWithBonus	stopCountDown updateBonusTimes
canLoadNextQuestion	Serveur	Signaler au client de charger la prochaine question	—	stopCountDown
answerSubmitted	Client	Envoyer au serveur la réponse du client	isCorrect submittedFromTimer	allSubmitted canLoadNextQuestion

gameEnded	Client	Signaler au serveur que la partie est terminée	—	showResults
histogramUpdate	Client	Mettre à jour l'histogramme	updateData	updateHistogram
resetHistogram	Client	Réinitialiser l'histogramme	—	updateHistogram
updateBonusTimes	Client	Mettre à jour le nombre de bonus reçus	bonusTimes	latestPlayerList
noPlayers	Serveur	Signaler qu'il n'y a plus de joueurs dans la salle	—	toggleLock stopCountDown
isQuestionTransition	Serveur	Signaler si le compte à rebours est pour une transition de question	isQuestionTransition	—
countDownEnd	Serveur	Signaler que le compte à rebours est terminé	lastCount	stopCountDown gameEnded histogramUpdate startCountDown answerSubmitted updateBonusTimes
scoreUpdated	Serveur	Signaler que le score à bien été mis à jour	updatedPlayer	—
updateHistogram	Serveur	Confirmation de la mise à jour de l'histogramme	histogram	—
pauseCountDown	Client	Mettre en pause la minuterie	—	—
panicMode	Client	Activer le mode panique	—	alertPanicMode
alertPanicMode	Serveur	Alerter les joueurs que le mode panique est activé	—	—
mutePlayer	Client	Désactiver le clavardage d'un joueur	player	alertMuted
alertMuted	Serveur	Avertir le joueur qu'il ne peut plus clavarder avec les autres joueurs	—	—

3.3 Interfaces

Nom	Description	Structure

Choice	Information sur un choix de réponse	{
		text: string;
		isCorrect: boolean;
		}

Question	Informations sur une question d'un jeu	{
		text: string;
		points: number;
		type: QuestionType;
		choices: Choice[];
		}

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

LobbyDetails	Informations sur un lobby de jeu	{
		isLocked: boolean;
		players: Player[];
		bannedNames: string [];
		game: Game;
		bonusRecipient?: string;
		histogram?: { [key: string]: number };
		}

Player	Informations sur un joueur	{
		socketId: SocketId;
		name: string;
		answerSubmitted: boolean;
		score: number;
		bonusTimes: number;
		isStillInGame: boolean;
		isAbleToChat: boolean;
		}

MessageData	Informations sur un message de clavardage	{
		sender: string;
		content: string;
		time: string;
		}

Button	Informations sur un bouton	{
		color: string;
		selected: boolean;
		text: string;
		isCorrect: boolean;
		id: number;
		showCorrectButtons: boolean;
		showWrongButtons: boolean;
		}