Équipe 208

Protocole de communication

Version 1.0

Historique des révisions

Date	Version	Description	Auteur
2023-10-31	1.0	Ajout des interfaces côté Client	Ahmed Zghal
2023-11-02	1.1	Ajout de l'introduction et l'explication du choix de communication des websockets pour le clavardage	Jasmine Romain
2023-11-06	1.2	Ajout des paquets webSockets	Jasmine Romain et Ahmed Zghal
2023-11-06	1.3	Ajout des paquets HTTP et paragraphes de descriptions des différents protocoles	Evan BLANC
2023-11-07	1.4	Révision finale pour s'assurer de la concordance du document avec l'architecture du déploiement de la remise sprint 2.	Nicolas Poirier, Miguel Gascon, Ahmed Zghal, Xavier Schlegel, Jasmine Romain, Evan BLANC

Table des matières

1. Introduction 3
2. Communication client-serveur 3
3. Description des paquets 4
   3.1 Paquets HTTP 4
   3.2 Paquets WebSockets 6
   3.3 Interfaces 9

Protocole de communication

1. Introduction

Dans ce présent travail, nous présenterons les protocoles de communications nécessaires pour le bon fonctionnement de notre site web. Nous commencerons par nommer les protocoles utilisés pour la communication du client-serveur de notre site web et la raison de cette décision. Pour terminer, chacun des paquets, qu'ils soient HTTP, sockets ou interface, sera illustré à l'aide de trois tableaux distincts à la section 3.

2. Communication client-serveur

Nous avons opté pour l'utilisation des WebSocket dans la gestion du déroulement des parties étant donné que celle-ci requiert l'accès à toute sorte d'informations et de logiques en temps réel. Les "Lobbies" utilisent les "rooms" de ce protocole afin de regrouper toutes les communications concernant les utilisateurs d'un même "Lobby". Cette approche permet une communication hermétique entre les utilisateurs d'une même "room" et donc le déroulement de plusieurs parties en parallèle, puisque les joueurs externes n'ont pas accès à ces envois de données. Cette approche permet également l'envoi synchronisé de l'information à tous les joueurs participants. Les WebSockets permettent également la gestion d'événements et d'interruptions sans se résoudre à une écoute active. Ces évènements peuvent être déclenchés du côté serveur, ce qui ne serait pas possible avec l'implémentation d'un protocole HTTP.

Pour le clavardage, nous utilisons également un protocole WebSocket pour permettre la communication entre tous les membres d'une partie. Ce protocole permet non seulement d'envoyer un message de façon instantanée et de synchroniser à tous les utilisateurs présents dans la salle, mais également que cet envoi soit unique aux utilisateurs d'une room donnée. Les WebSocket permettent en plus au serveur d'informer un client de la réception d'un nouveau message, ce qui ne serait pas possible dans un protocole HTTP. Dans ce scénario alternatif, le client devrait lui-même faire une requête HTTP pour pouvoir être informé d'un changement sur ses messages.

L'utilisation du protocole HTTP est plus concentrée au sein des fonctionnalités propres au sprint 1, puisque celle-ci ne nécessitait pas la mise à jour de l'information du client par les serveurs. Ce protocole est toujours en place pour ce qui a trait aux accès à la base de données, par exemple dans la vue de création de parties, et également dans les vues propres à la création, la mise à jour, l'import et l'export des quiz du côté de l'administrateur. Dans ce contexte, le protocole est nettement suffisant pour accomplir les tâches demandées et il restera en place pour la continuité du projet. Les fonctionnalités de confirmation de mots de passe dans la vue d'authentification utilisent également ce protocole pour assurer une vérification du côté serveur et non du côté client.

Nous avons décidé d'utiliser le protocole HTTP pour la gestion de l'historique, car il s'agit de communication avec la base de données. Les fonctionnalités que nous apporteraient ici les WebSocket seraient peu pertinentes et nous préférons garder l 'homogénéité des appels à la BDD via HTTP comme depuis le début du projet.

3. Description des paquets

rouge: requêtes pas encore implémentées

3.1 Paquets HTTP

Méthode	URI	Paramètres URI	Description	Corps de la Requête	Corps de la Réponse	Code(s) de retour
GET	/game/	—	Récupérer tous les jeux visibles	—	*Game[]*	Succès: 200 Échec: 500 ou 404
GET	/game/admin	—	Récupérer tous les jeux	—	*Game[]*	Succès: 200 Échec: 500 ou 404
GET	/game/:id	id: string	Récupérer un jeu grâce à son id	—	*Game*	Succès: 200 Échec: 500 ou 404
GET	/game/admin/:id	id: string	Récupérer un jeu avec l'attribut isCorrect	—	*Game*	Succès: 200 Échec: 500 ou 404
POST	/game/	—	Créer un nouveau jeu	CreateGameDto	—	Succès: 201 Échec: 400 ou 403 ou 500
POST	/game/export/:id	id: string	Exporter un jeu au format JSON		JSON data	Succès: 200 Échec: 404 ou 500
PATCH	/game/:id	id: string	Modifier un jeu	UpdateGameDto	—	Succès: 200 Échec: 400 ou 403 ou 404 ou 500
DELETE	/game/:id	id: string	Supprimer un jeu	—	—	Succès: 200 Échec: 404 ou 500
GET	/game/:gameId/question/	gameId: string	Récupérer les questions d'un jeu grâce à son id	—	QCM[] ou QRL[]	Succès: 200 Échec: 404 ou 500

GET	/game/:gameId/ question/admin	gameId: string	Récupérer les questions et les réponses d'un jeu grâce à son id	—	QCM[] ou QRL[]	Succès: 200 Échec: 404 ou 500
GET	/game/:gameId/ question/ :questionNumber	gameId: string questionNumber: number	Récupérer une question d'un jeu	—	QCM ou QRL	Succès: 200 Échec: 404 ou 500
GET	/game/:gameId/ question/ :questionNumber/ admin	gameId: string questionNumber: number	Récupérer une question et les réponses d'un jeu	—	QCM ou QRL	Succès: 200 Échec: 404 ou 500
POST	/game/:gameId/ question/	gameId: string	Ajouter une question a un jeu	CreateQCMDto ou CreateQRLDto	—	Succès: 201 Échec: 400 ou 404 ou 500
PATCH	/game/:gameId/ question/ :questionNumber/	gameId: string questionNumber: number	Modifier une question	UpdateQCMDto ou UpdateQRLDto	—	Succès: 200 Échec: 400 ou 404 ou 500
DELETE	/game/:gameId/ question/ :questionNumber/	gameId: string questionNumber: number	Supprimer une question	—	—	Succès: 200 Échec: 404 ou 500
POST	/auth/	—	Authentifier l'administrateur	password: string	JSON { isValid}	Succès: 200 Échec: 403 ou 500
GET	/history/	—	Récupérer l'historique des parties	—	—	Succès: 200 Échec: 404 ou 500
POST	/history	—	Ajouter une partie à l'historique	{nom: string Questionnaire:string, Date: string, nbJoueurs: number, meilleurScore:number }	—	Succès: 201 Échec: 500

3.2 Paquets WebSockets

Événement	Source	Description	Données	Événements Potentiellement déclenchés
countdown	Client	Demande de lancer un countdown d'une durée déterminée à un lobby déterminé	*{ lobbyId: string, countdownDuration: number }*	countdown
countdown	Serveur	Renvoie un countdown avec la durée déterminée	*countdownDuration*: *number*	—
ccreateRoom	Client	Demande la création d'une nouvelle room	*gameId: string*	roomCreation
roomCreation	Serveur	Notifier le succès ou l'échec de la création de la room	*{ success: booléen, message: string }*	—
checkRoom	Client	Vérifié si la room qu'on veut rejoindre existe ou pas	*lobbyId: string*	roomJoining
roomJoining	Serveur	Notifier le succès ou l'échec de join room	*{ success: boolean, message: string }*	—
joinRoom	Client	Ajouter le nouveau joueur a la room	*playerName*	newPlayer, chooseNameError, validName
newPlayer	Serveur	envoie la liste de tous les joueurs mis à jour	*playerNames: string[]*	—
chooseNameError	Serveur	Envoie une erreur de nom au client	*aswer: string*	—
validName	Serveur	Confirm au client un nom valide	—	—
startGame	Client	Lancer le jeu-questionnaire	—	—
requestRoomId	Client	Demander le roomId	—	roomId
roomId	Serveur	Renvoie le roomId	*roomId: string*	—
requestCurrent Players	Client	Demande la liste des joueurs d'une room	—	newPlayer
requestName	Client	Demande le nom du client	—	playerName
playerName	Serveur	renvoie le nom du client	*playerName: string*	—
lockLobby	Client	Demande de fermer l'accés à la room	*lobbyLockState: boolean*	—
leaveLobby	Client	Demande de quitter la room	—	playerDisconnected, disconnected

player Disconnected	Serveur	Renvoie le nom du joueur déconnecté	*playerName: string*	—
kickPlayer	Client	Demande d'enlever un joueur d'une room	*playerName: string*	playerKicked
playerKicked	Serveur	Confirme que le joueur est effacé de la room	—	—
retrieveGameId	Client	Demande le gameId de la room	—	retrieveGameId
retrieveGameId	Serveur	Envoie le gameId au demandeur	*gameId: string*	—
disconnected	Serveur	confirme la déconnexion du client	—	—
firstQuestion	Client	Demande l'envoi de la première question du quiz	—	newQuestion, countdown
newQuestion	Serveur	Renvoie une nouvelle question	*question: Question*	—
nextQuestion	Client	Demande l'envoi de la prochaine question	*newQuestion: Question*	newQuestion, countDown, endGame
endGame	Serveur	Informer l'organisateur qu'il reçoit la dernière question	—	—
nextQuestion Countdown	Client	Demande de lancer un countdown pour passer a la prochaine question	—	coutndown
submitAnswer	Client	Envoie la réponse de l'utilisateur au serveur	answerSubmit: AnswerSubmit	showAnswer ,wait
showAnswer	Serveur	Envoie les bonnes réponses relatives a la question	correctchoices: number[]	—
wait	Serveur	Envoie un état d'attente jusqu'a ce que tous les joueurs aient répondus	—	—
correctChoices	Client	L'organisateur demande les bonnes réponses	—	correctChoices
correctChoices	Serveur	Le serveur renvoie les bonnes réponses a l'organisateur	correctchoices: number[]	—
points	Client	Demande les points du joueur	—	points
points	Serveur	Renvoie les points du joueur	points: Points	—
playersPoints	Client	Demande la liste des joueurs avec leurs points	—	playersPoints
playersPoints	Serveur	Renvoie la liste des joueurs avec leurs points	points: Points[]	—

newSelection	Client	Demander une modification du nombre de sélection d'une réponse	selection: number	newSelection
newSelection	Serveur	Notifier une modification du nombre de sélection d'une réponse	selection: number	—
newDeselection	Client	Demander une modification du nombre de sélection d'une réponse	selection: number	newDeselection
newDeselection	Serveur	Notifier une modification du nombre de sélection d'une réponse	selection: number	—
startGame Countdown	Client	Demande de lancer un countdown pour commencer le quiz	—	startGame, countdown
retrieveLobby Scores	Client	Demander le score de tous les joueurs d'une room	—	lobbyScores
lobbyScores	Serveur	Renvoyer le score de tous les joueurs d'une room	scores: Scores[]	—
navigateTo Results	Client	l'organisateur demande de renvoyer tous les joueurs à la vue des résultats	—	navigateToResults
navigateTo Results	Serveur	Notifie tous les joueurs pour aller à la vue des résultats	—	—
requestMessage History	Client	Envoie au serveur le nouveau message écrit dans le clavardage	—	roomMessages
newMessage	Client	Ajouter le nouveau message dans la liste de roomMessages	message: Message	roomMessages
roomMessages	Serveur	Envoie les messages au client	messages: Message[]	—

3.3 Interfaces

Nom	Description	Structure

Choice	Informations sur un choix de réponse	{
		text: string,
		isCorrect: boolean,
		selected?: boolean,
		showCorrect?: boolean,
		}

HistogramChoice	Informations sur un choix de réponse
	+le nombre de sélection de la réponse
	par les utilisateurs
		{
		text: string,
		isCorrect: boolean,
		selected?: boolean,
		showCorrect?: boolean,
		selectedCount?:number
		}

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

Player	Informations sur le joueur	{
		id: string,
		name: string,
		points: number,
		bonusCount: number,
		answerResponse: AnswerSubmit,
		}

Question	Informations sur une question	{
		text: string,
		choices: Choice[],
		points: number,
		type: string
		}

QuestionForm	Informations sur une question à créer
	dans le contexte du FormGroup
		{
		text: string;
		choices: Choice[];
		points: number;
		index?: number;
		showDetails?: boolean;
		}

Message	Informations sur un message	{
		playerName: string;
		inputMessage: string;
		time: string;
		}

Score	Score d'un joueur	{
		playerName: string,
		points: number,
		bonusCount: number
		}

QCM	Question à choix multiple	{
		choices: Choice[],
		points: number,
		text: string,
		type: string
		}

QRL	Question à réponse longue	{
		answer: string,
		points: number,
		text: string,
		type: string
		}

Points	Les points d'un joueur pour une réponse donnée	{
		name: string,
		points: number,
		hasBonus?: boolean
		}

Lobby	Contiens les informations propres à une partie en cours	{
		lobbyId: string;
		players: Map<string, Player>;
		sockets: string[];
		nameBan: string[];
		timer: NodeJS.Timeout;
		timeLeft: number;
		isLocked: boolean;
		game: Game;
		dateStart: string;
		submitAnswerCount: number;
		questionIndex: number;
		currentMessages: Message[];
		lobbyScores: Score[];
		playerBonuses: string;
		inGame: boolean;
		}

CustomHTTPError	Type d'erreur HTTP envoyé au client	{
		statusCode: number
		message: string[],
		error: string
		}

AnswerSubmit	Soumission des réponses du joueur	{
		asnwerTime: number,
		answer: number[]
		}