Équipe 205

Protocole de communication

Version 4.0

Historique des révisions

Date	Version	Description	Auteur
2023-11-01	1.0	<Introduction>	<Damaris Calestrov>
2023-11-01	2.0	<2 - Communication Client-Serveur 3.1 - Paquets HTTP 3.3 - Interfaces>	<Damaris Calestrov et Loïc Nguemegne Temena >
2023-11-01	3.0	<3.2 - Paquets WebSockets>	<Salma Sahnoune>
2023-11-02	4.0	<Révision du document>	<Aymen Ghodbane, Nour-El Houda El Hasni et Roman Alejandro Roman Canizalez>

Table des matières

Protocole de communication

1. Introduction

Ce rapport a pour but de présenter la communication client-seveur et la description des différents paquets présents dans le projet effectué au cours de cette session portant sur une plateforme de jeux-questionnaire. Tout d'abord nous allons décrire la communication entre les clients et le serveur en parlant des moyens de communication, ainsi que des différentes raisons pour lesquelles on a choisi les protocoles de communications présents dans notre projet. Par la suite, nous allons présenter la description des différents paquets dans 3 tableaux différents. En premier, nous présenterons les paquets HTTP, ensuite nous parleront des paquets WebSockets et nous terminerons par les interfaces utilisées dans notre projet.

2. Communication client-serveur

Tout au long de notre projet, nous avons utilisé les requêtes HTTP ou le protocole WebSocket pour différentes communications entre le client et le serveur.

Dans le cas des requêtes HTTP, nous les avons utilisés pour effectuer des demandes ou requêtes ponctuelles au serveur et attendre des réponses. Ce type de communication est plus souvent utilisé lorsqu'on n'a pas besoin de garder une communication active avec le serveur et dans le cadre de notre application, nous avons utilisé les requêtes http pour des action comme obtenir, créer supprimer ou modifier des matchs ou des jeux sur le serveur.

Pour la communication WebSocket, ce type de communication est idéale lorsqu'on a besoin de maintenir une connexion active entre le serveur et les clients. Ce protocole était donc idéal pour le déroulement de la partie, car pendant une partie on envoie continuellement de l'information au serveur et on doit également réagir aux évènements des autres utilisateurs. Ainsi, lorsqu'on débute une partie chaque participant ainsi que l'organisateur se trouvent dans la même salle, la communication reste active pour tous les clients se trouvant dans cette salle et de ce fait, ils peuvent communiquer entre eux par des évènements. Nous avons utilisé ce protocole pour les fonctionnalités comme le clavardage, la navigation à la prochaine question, la présentation des résultats et autres.

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	/api/games	---	Récupérer la liste de tous les jeux	---	Game[]	Succès : 200 Échec: 404 409
POST	/api/games	---	Ajouter un nouveau jeu dans la liste de tous les jeux	Game	Game	Succès : 201 Échec : 404
PUT	/api/games	---	Modifier un jeu	Game	Game	Succès : 200 Échec : 404

POST	/api/games/title	---	Vérifier qu'un jeu existe avec son titre	{title : string}	{ titleExists : boolean }	Succès : 200 Échec : 404
GET	/api/games/:id	id: string	Récupérer un jeu selon son id	---	Game	Succès : 200 Échec : 404
DELETE	/api/games/:id	id: string	Supprimer un jeu selon son id	---	---	Succès : 200 Échec : 404
PATCH	/api/games/:id/ update-visibility	id: string	Mettre à jour la visibilité du jeu selon son id	---	---	Succès : 200 Échec : 404
POST	/api/auth	---	Authentifier un utilisateur	{pwd : string}	{token : string}	Succès : 200 Échec : 403 404
GET	/api/match	---	Récupérer la liste de tous les matchs	---	Match[]	Succès : 200 Échec : 404
DELETE	/api/match	---	Effacer tous les matchs de la base de données	{pwd : string}	---	Succès : 200 Échec : 401 500
POST	/api/match	---	Créer un nouveau match	Match	---	Succès : 201 Échec : 403 500
GET	/api/match/validity/: accessCode	accessCode: string	Récupérer la validité d'un code d'accès	---	boolean	Succès : 200 Échec : 404
GET	/api/match/accessibi lity/:accessCode	accessCode: string	Récupérer l'accessibilité à un match	---	boolean	Succès : 200 Échec : 404
PATCH	/api/match/accessibi lity/:accessCode	accessCode: string	Modifier l'accessibilité	---	---	Succès : 200

			d'un match			Échec : 404
GET	/api/match/:accessCode	accessCode: string	Récupérer un match à partir de son code d'accès	---	Match	Succès : 200 Échec : 404
DELETE	/api/match/:accessCode	accessCode: string	Effacer un match à partir de son code d'accès	---	---	Succès : 200 Échec : 404
POST	/api/match/playerNameValidity	--	Vérifier l'existence d'un nom de joueur	Validation	boolean	Succès : 200 Échec : 404
POST	/api/match/player	---	Ajouter un joueur dans la liste des joueurs du match	UpdateMatch	---	Succès : 200 Échec : 404

3.2 Paquets WebSockets

Événement	Source	Description	Données	Événements potentiellement déclenchés
joinMatchRoom	Client	Demander à rejoindre la salle d'un match	PlayerRequest	newPlayer
newPlayer	Serveur	Notifier de l'ajout d'un joueur dans la salle d'attente d'un match	Player[]	-
sendMessage	Client	Envoyer un message dans le clavardage d'un match	Message	chatMessage
chatMessage	Serveur	Notifier de la réception d'un nouveau message dans le clavardage d'un match	Message	-
switchQuestion	Client	Envoyer aux joueurs une demande pour passer à la prochaine	Room	nextQuestion
nextQuestion	Serveur	Notifier les joueurs du passage à la prochaine question	-	-
updateAnswer	Client	Envoyer une demande de changement des réponses d'un joueur après le changement de sélection des choix de réponses	UpdateAnswerRequest	answerUpdated
answerUpdated	Serveur	Notifier l'organisateur d'un changement dans la	PlayerAnswers[]	-

		sélection de choix de réponse d'un joueur		
startTimer	Client	Envoyer une demande au serveur pour démarrer le timer	TimerRequest	newTime
newTime	Serveur	Notifier les joueurs et organisateur du changement de la valeur du timer à chaque seconde	TimerRequest	-
stopTimer	Client	Envoyer une demande su serveur pour arrêter le timer	Room	-
cancelGame	Client	Envoyer une demande d'annulation du match par l'organisateur à partir de la salle d'attente	Room	-
gameCanceled	Serveur	Notifier les joueurs de l'annulation du match par l'organisateur dans la salle d'attente	-	-
finishMatch	Client	Envoyer une demande de la fin fu match par l'organisateur durant la partie	Room	matchFinished
matchFinished	Serveur	Notifier les joueurs de la fin du match par l'organisateur durant la partie	-	-
beginMatch	Client	Envoyer par l'organisateur une demande aux joueurs pour joindre le match commencé	Room	joinMatch
joinMatch	Serveur	Notifier les joueurs du début du match	Match	-
removePlayer	Client	Envoyer une demande pour retirer un joueur du match	PlayerRequest	playerRemoved
playerRemoved	Serveur	Notifier les joueurs et l'organisateur du départ d'un joueur	Player[]	-
updatePlayerScore	Client	Envoyer une demande pour mettre à jour le score d'un joueur dans le match stocké dans le serveur	QuestionRequest	updatedPlayerScore
updatedPlayerScore	Serveur	Notifier du changement	Player	-

		de score pour un joueur afin de synchroniser les scores entre les matchs stockés côté serveur et côté client		
setFinalAnswer	Client	Envoyer qu'une réponse est finale	UpdateAnswerRequest	finalAnswerSet
finalAnswerSet	Serveur	Notifier l'organisateur que la réponse du joueur est finale	PlayerAnswers	-
playerLeftAfterMatchBegun	Client	Envoyer au serveur qu'un joueur a quitté le match après que ce dernier ait commencé	QuestionRequest	playerDisabled
playerDisabled	Serveur	Notifier qu'un joueur n'est plus actif (a quitté le match pendant le déroulement de ce dernier)	Player	-
allPlayersResponded	Serveur	Notifier l'organisateur que tous les joueurs ont déclaré leurs réponses finales pour arrêter le timer et passer aux résultats de la question	-	-

3.3 Interfaces

Nom	Description	Structure

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

CreateGameDTO	Informations utilisées dans la création d'un jeu	{
		id: string;
		title: string;
		description: string;
		duration: number;
		lastModification: string;
		questions: Question[];
		isVisible: boolean;
		}

Question	Interface des questions dans un jeu	{
		id: string;
		type: string;
		text: string;
		points: number;
		choices: Answer[];
		timeAllowed: number;
		}

Answer	Interface pour les réponses	{
		text: string;
		isCorrect: boolean;
		playerId: string;
		didPlayerAnswer: boolean;
		questionId: string;
		point: number;
		lastAnswerTime: Date;
		final: boolean;
		points: number;
		}

Player	Interface des joueurs dans une partie	{
		name: string;
		isActive: boolean;
		score: number;
		nBonusObtained: number;
		}

PlayerAnswers	Interface pour communiquer les réponses d'un joueur	{
		name: string;
		answers: Choice[];
		lastAnswerTime: string;
		final: boolean;
		questionId: string;
		obtainedPoints: number;
		}

Match	Interface du match utilisé lors du déroulement des parties	{
		game: Game;
		begin: Date;
		end: Date;
		bestScore: number;
		accessCode: string;
		testing: boolean;
		players: Player[];
		managerName: string;
		isAccessible: boolean;
		bannedNames: string[];
		playerAnswers: PlayerAnswers[];
		panicMode: boolean = false;

		timer: number;
		timing: boolean = true;
		}

UpdateMatch	Interface pour mettre à jour une valeur dans un match	{
		accessCode: string;
		player: Player;
		}

Validation	Interface pour valider si le nom d'un joueur existe	{
		accessCode: string;
		name: string;
		}

Message	Interface pour envoyer et recevoir des messages	{
		playerName: string;
		matchAccessCode: string;
		time: string;
		data: string;
		}

PlayerRequest	Interface pour transférer les données en lien avec le joueur pour les événements durant le déroulement de la partie	{
		roomId: string;
		name: string;
		hasPlayerLeft?: boolean;
		}

QuestionRequest	Interface pour transférer les données nécessaires pour mettre à jour les informations pour une question durant une partie	{
		matchAccessCode: string;
		player: Player;
		questionId: string;
		}

Room	Interface pour finir	{
		id: string;
		}

TimerRequest	Interface pour transférer les données relatives au temps	{
		roomId: string;
		timer: number;
		timeInterval?: number;
		}

UpdateAnswerRequest	Interface pour transférer les données relatives aux réponses des joueurs	{
		matchAccessCode: string;
		playerAnswers: PlayersAnswers
		}