Équipe 207

Protocole de communication

Version 1.0

Historique des révisions

Date	Version	Description	Auteur
2023-10-23	1.0	Ajout d'informations dans le tableau 3.1	Youssef Ouarad
2023-11-04	2.0	Ajout d'informations dans les tableaux 3.2 et 3.3	Youssef Ouarad
2023-11-06	3.0	Ajout d'informations dans les tableaux 3.1, 3.2 et 3.3	Rima Al-Zawahra
2023-11-07	4.0	Complétion du tableau 3.2	Youssef Ouarad

Table des matières

Protocole de communication

1. Introduction

Dans le cadre de ce projet, deux principaux protocoles de communication sont utilisés pour faciliter les interactions entre le client et le serveur : HTTP et WebSocket. Chacun de ces protocoles a des utilisations spécifiques en fonction des fonctionnalités requises par l'application.

Le protocole HTTP est principalement utilisé pour les demandes et réponses unidirectionnelles. Il est idéal pour les actions qui nécessitent une simple requête au serveur et une réponse correspondante. Par exemple, la validation des entrées des utilisateurs, le chargement des données d'un jeu questionnaire, la gestion des jeux questionnaires et l'historique des parties, et le chargement et l'envoi de la liste des joueurs et des messages de la zone de clavardage.

D'autre part, le protocole WebSocket est utilisé pour les communications bidirectionnelles en temps réel. Il est essentiel pour les fonctionnalités qui nécessitent une interaction instantanée entre les clients et le serveur, comme le clavardage entre les utilisateurs, la gestion du compte à rebours, la soumission des réponses des joueurs, l'envoi des points, la gestion de l'accès des joueurs à une salle, et la gestion générale d'une partie.

Les sections suivantes décrivent les choix du moyen de communication entre les clients et le serveur pour chaque fonctionnalité majeure du de l'application en plus de deux tableaux détaillant le contenu des différents types de paquets HTTP et Websocket utilisés dans l'application nécessaire.

2. Communication client-serveur

Fonctionnalité utilisant HTTP

- Demande et validation du mot de passe pour accéder à la page admin
- Demande et validation du code pour rejoindre une partie
- Demande et validation du nom du joueur
    - Raison: Permet de valider une entrée d'un utilisateur avec un simple envoie au serveur qui lui retourne une réponse (Ok ou Erreur) dépendamment de la validité du code.

- Chargement d'un jeu questionnaire
    - Raison: Permet de récupérer les données associé à un jeu questionnaire dans le serveur et de les afficher au client avec une simple requête au serveur.

- Gestion d'un jeu questionnaire (Créer/Modifier/Supprimer/Visibilité/Importation)
    - Raison: Permet de faire plusieurs action avec un jeu de questionnaire avec une simple requête au serveur

- Gestion de l'historique des partie (Affichage/filtrage/supprimer)
    - Raison:Permet de stocker les parties dans le serveur, ainsi que de les afficher et de filtrer avec une simple requête au serveur.

- Chargement et envoie de la liste des joueurs dans une partie
    - Raison: Permet de recevoir la liste des joueurs dans une partie une fois que celle-ci est commencée. Permet aussi d'envoyer la liste de la partie au serveur afin que celle ci soit chargé dans la vue des résultats
- Chargement et envoie de tous les message de la zone de clavardage

    - Raison: Permet d'envoyer la liste des messages qui ont été envoyés durant une partie dans le serveur, afin de la récupérer dans la vue des résultats.

Fonctionnalité utilisant WebSocket

- Clavardage entre les utilisateurs (Envoie message, droit d'écriture)
    - Raison: Permet une communication bidirectionnelle entre tous les utilisateurs d'une même salle. Permet aux utilisateurs d'envoyer et recevoir leurs message de manière instantanés.

- Gestion du compte à rebours (Débuter, Stopper, Réinitialiser)
    - Raison: Permet d'envoyer des évènement à tous les utilisateur connecté dans une salle afin qu'ils soient tous synchronisé au même compte à rebours.
- Gestion des soumissions de réponses des joueurs:
    - Raison: Permet au joueur d'envoyer leur réponse et à l'organisateur de réagir aux réponses envoyées par les joueurs par l'entremise de l'histogramme.

- Gestion de l'envoie des points (Bonus inclus):
    - Raison: Permet à un joueur d'envoyer ces points gagnés à tous les joueurs à l'organisateur afin que celui-ci puisse avoir accès à tous. Cela permet de mettre à jour les scores de manière instantanée et de tenir les joueurs informés de leur progression.

- Gestion de l'accès d'un joueur dans une salle (Nom Banni, Verrouillage/Déverrouillage):
    - Raison: Permet au joueur à l'organisateur d'émettre des évènement qui permet à tous les joueurs d'une salle d'être notifié de l'état de la salle (verrouillé ou déverrouillé). Permet aussi à un joueur d'être notifié instantanément lorsque l'organisateur.

3. Description des paquets

Format

Quiz:

{ "$schema": "string",
"id": "string", "title": "string",
"duration": 0,
"lastModification": "string",
"description": "string",
"visibility": true,
"questions": ["string"] }

3.1 Paquets HTTP

Méthode	URI	Paramètres URI	Description	Corps de la Requête	Corps de la Réponse	Code(s) de retour
GET	quizzes	—	Récupérer tous les quiz	—	*Liste de tous les Quiz*	Succès: 200 Échec: 404
GET	quizzes/:id	id: string	Récupérer le quiz avec le id envoyé	—	Quiz	Succès: 200 Échec: 404
GET	quizzes/available/:id	id: string	Vérifier si un quiz est disponible	—	Succès: Booléen	Succès: 200 Échec: 404
GET	quizzes/visible/:id	id: string	Vérifier si un quiz est visible	—	Succès: Booléen	Succès: 200 Échec: 404

GET	rooms/:roomId/players	roomId: string	Récupérer les joueurs présent dans la salle	—	Liste de Joueurs dans une salle	Succès: 200 Échec: 404
GET	rooms/:roomId/results	roomId: string	Récupérer les résultats des joueurs	—	Liste des joueurs avec leurs points	Succès: 200 Échec: 404
GET	rooms/:roomId/chat	roomId: string	Récupérer les messages du chat	—	Liste des messages des utilisateurs	Succès: 200 Échec: 404
GET	rooms/:roomId/quiz	roomId string	Récupérer le questionnaire	—	Quiz	Succès: 200 Échec: 404
POST	quizzes/	—	Ajouter un quiz une la liste de quiz	Quiz	Quiz	Succès: 201 Échec: 404
POST	quizzes/import	—	Importer un quiz à la liste de quiz	Quiz	Quiz	Succès: 201 Échec: 404
POST	admin/login	—	Valider le mot de passe saisi par le client	{ password: string }	Succès: Booléen Échec: —	Succès: 200 Échec: 404

POST	rooms/:roomId/name	—	Valide le nom de l'utilisateur	{ name: string, socketId: string}	Booléen	Succès: 200 Échec: 500
POST	rooms/:roomId/join	roomId: string	Vérifie l'état de la salle (si elle existe et si elle est verrouillée ou non)	{ socketId: string }	{ roomState: 'IS_LOCKED' | 'INVALID' | 'OK', quiz: null | Quiz }	Succès: 200 Échec: 500
POST	rooms/new	—	Renvoie le roomId de la nouvelle salle créée	{ quiz: Quiz, socketId: string }	roomId	Succès: 200 Échec: 500
PUT	quizzes/:id	id: string	Mettre à jour un quiz dans la liste de quiz	Quiz	Quiz	Succès: 200 Échec: 404
DELETE	quizzes/:id	id: string	Supprimer un quiz de la liste de quiz	—	—	Succès: 200 Échec: 404

3.2 Paquets WebSockets

Événement	Source	Description	Données	Événements Potentiellement déclenchés
roomMessage	Client	Envoyer au serveur un message d'un utilisateur	{ roomId: string, message: string }	newRoomMessage
newRoomMessage	Serveur	Envoyer à tous les utilisateurs d'une salle un message reçu au serveur	{ authorName: string, timeString: string, message: string, sentByUser: boolean }	—
playerLeaveGame	Client	emander l'abandon d'un joueur dans une rtie (déconnecter le client et l'enlever de la salle)	roomId: string, isInGame: boolean }	playerAbandonedGame
startGame	Client et serveur	otifier les clients du début d'une partie	roomId: string	—
endGame	Client	emander l'arrêt de la partie (déconnecter les joueurs et suppression de la salle)	{ roomId: string, gameAborted: boolean }	gameAborted
goodAnswerOnClick	Client et serveur	Notifie l'organisateur d'une bonne réponse après la soumission d'un utilisateur	roomId: string	goodAnswerOnClick
goodAnswerOn FinishedTimer	Client et serveur	Notifie l'organisateur d'une bonne réponse après l'expiration du timer	*roomId: string*	goodAnswerOn FinishedTimer
badAnswerOnClick	Client et serveur	Notifie l'organisateur d'une mauvaise réponse après la soumission d'un utilisateur	roomId: string	BadAnswerOnClick
badAnswerOn FinishedTimer	Client et serveur	Notifie l'organisateur d'une mauvaise réponse après l'expiration du timer	roomId: string	BadAnswerOn FinishedTimer
questionChoiceSelect	Client et serveur	Notifie l'organisateur d'une sélection d'un joueur	{ roomId: string; estionChoiceIndex: number }	QuestionChoiceSelect
questionChoiceUnselect	Client et serveur	Notifie l'organisateur d'une sélection d'un joueur	*{ roomId: string; estionChoiceIndex: number }*	QuestionChoiceUnselect
giveBonus	Client et serveur	Notifie l'organisateur et le joueur ayant reçu le bonus	*roomId: string*	GiveBonus BonusUpdate
addPointsToPlayer	Client et serveur	Notifie l'organisateur et d'ajouter les points du joueurs dans sa liste des joueurs	*{ roomId: string; points: number }*	AddPointsToPlayer
nextQuestion	Client et serveur	Notifie l'organisateur et les joueurs qu'on se dirige vers la prochaine question	*roomId: string*	NextQuestion
playerAbandonedGame	Serveur	Notifier de l'abandon d'un joueur	–	–

gameAborted	Serveur	Notifier de l'arrêt d'une partie aux clients connectés	–	–
showResults	Client et serveur	Notifie	*roomId: string*	ShowResults
joinRoom	Serveur	Demande l'accès à une salle	*roomId: string*	-
successfulJoin	Client	Notifie qu'un joueur a rejoint la salle	*{ roomId: string, name: string}*	PlayerHasJoined
playerHasJoined	Serveur	Notifie à tous les gens d'une salle qu'un nouveau joueur à rejoins	*name: string*	-
startTimer	Client	Commence le compte à rebours	{ initialTime: number; ickRate: number; roomId: string }	CurrentTimer TimerFinished
stopTimer	Serveur	Arrête le compte à rebours	*roomId: string*	__
currentTimer	Serveur	Notifie tous les gens de la salle du temps présent du timer	*counter: number*	__
timerFinished	Serveur	Notifie tous les gens dans une salle que le compte à rebours a expiré	__	__
lockRoom	Client	Demander de verrouiller la salle	__	__
unlockRoom	Client	Demander de déverrouiller la salle	__	__
banName	Client	Demander de bannir le nom d'un joueur	*{ roomId: string; name: string }*	banNotification
banNotification	Serveur	Notifier un joueur qu'il a été bani	__	__

3.3 Interfaces

Nom	Description	Structure

ChatMessage	Informations sur un message dans le clavardage	{
		authorName: string,
		time: string,
		message: string,
		sentByUser: boolean
		}

AddPointsResponse	Informations sur l'ajoute
	de points à un joueur
		{
		pointsToAdd: number,
		name: string
		}

ErrorDialogData	Message d'erreur dans un popup	{
		errorMessage: string
		}

JoinRoomResponse	Information sur l'état d'une salle	{
		roomState: string,
		quizId: string | null
		}

PlayerInfo	Information sur le joueur	{
		name: string,
		score: number,
		hasAbandoned: boolean,
		bonusCount: number
		}

Quiz	Informations sur un quiz	{
		id: string,
		title: string,
		duration: number,
		lastModification: string,
		description: string,
		visibility?: boolean,
		question: Question[]
		}

Question	Informations sur une
	question à choix multiples d'un quiz
		{
		type: string,
		text: string,
		points: number,
		choice: choices[]
		}

Choice	Informations sur les choix de réponses d'une question	{
		text: string,
		isCorrect: boolean
		}

QuestionQRL	Information sur une question à réponse longue d'un quiz	{
		type: string,
		text: string,
		points: number
		}

Message	Structure d'un message	{
		title: string,
		body: string
		}

Room	Informations sur la salle de jeu	{
		id: string,
		quizId: string,
		organizer: Organizer,
		players: Player[],
		isLocked: boolean,
		bannedNames: string[],
		answersTime: AnswerTime[]
		}

User	Informations sur l'utilisateur	{
		socketId: string,
		name: string
		}

Player extends User	Information sur le joueur (qui est un utilisateur)	{
		points: number,
		bonusCount: number
		}

Organizer extends User	Information sur l'organisateur (qui est un utilisateur)	{
		name: string,
		socketId: number
		}

AnswerTime	Information sur le temps de soumission de la réponse d'un joueur	{
		userId: string
		timeStamp: number
		}

PopupMessageConfig	Informations sur la configuration d'un Popup	{
		message: string,
		hasCancelButton: boolean,
		okButtonText?: string
		cancelButtonText?: string
		okButtonFunction?: () => void
		cancelButtonFunction?: () => void
		}