Protocole de communication

Version 1.3

Historique des révisions

Date	Version	Description	Auteur
2023-10-23	1.0	<Début du travail : introduction , communication client-serveur , paquet http , début paquets Websockets , interfaces fini dans leur première itération>	<Gabriel Gascon-Parent>
2023-11-05	1.1	<Fini l'ajout des requête http des deux controllers , puis fini L'ajout des dernières interfaces actuellement utilisé au serveur >	<Gabriel Gascon-Parent>
2023-11-05	1.2	<Complétion des paquets websocket de tous les events utilisés et corrections mineures dans la partie 2>	<Julie Malosse>
2023-11-07	1.3	<Addition des paquets websockets de touts les events prévues pour le sprint 3>	<Allan-André Tchokogué, Gabriel Gascon-Parent >

Table des matières

Table des matières

Protocole de communication

1. Introduction

La communication entre le client et le serveur joue un rôle essentiel dans le fonctionnement de n'importe quelle application web moderne. De ce fait, nous allons, dans ce document, présenter l'architecture de notre protocole de communication conçue pour notre application web. Nous allons tout d'abord aborder la communication client-serveur, puis décrire le contenu des différents paquets utilisés pour offrir la meilleure expérience à nos utilisateurs.

2. Communication client-serveur

Dans le cadre du développement de notre application web, nous avons utilisé deux protocoles de communication différents : HTTP et WebSocket. Les deux jouent des rôles différents dans la conception pour répondre aux besoins spécifiques des requis des fonctionnalités.

Dans le cas du protocole WebSocket, il est utilisé pour le chat entre les utilisateurs, tout ce qui concerne le déroulement d'une partie ainsi que la vue d'attente. En effet, ces fonctionnalités requièrent une expérience en temps réelle et synchronisée pour les utilisateurs. Par exemple, dans la vue d'attente, il est essentiel que chaque joueur puisse voir la liste de joueurs changer au fur et à mesure que d'autres joueurs rejoignent la partie. Le chat bien évidemment doit assurer l'interaction en temps réelle de plusieurs clients dans une même partie. C'est pourquoi les WebSockets sont idéales et plus particulièrement les rooms de socket io. De la même manière, le déroulement d'une partie est affecté par les actions l'organisateur. Ainsi, le serveur doit réagir et envoyer de l'information aux joueurs de façon instantanée. Enfin, l'organisateur doit aussi voir son interface changer en temps réel en fonction des choix des joueurs. Ces exemples de notre projet sont tous des utilisations propices, efficace et nécessaire des WebSockets pour offrir ses fonctionnalités aux utilisateurs ainsi que la meilleure expérience utilisateur.

Au niveau du protocole http, il est utilisé pour des fonctionnalités qui ne nécessitent pas d'expérience en temps réelle ou de l'interaction avec les autres clients. Il est utile pour charger de l'information de la base de données comme afficher les listes de jeux ou pour la connexion de l'utilisateur à la vue d'administration.

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	-----	Obtenir la liste de tous les jeux dans la base de données	--------	Succès :
					{
					CreateGameDTO[]
					}
					Échec :---
						Succès :200
						Échec :404

GET	/games/exist:id	Id :string	Savoir si un jeu existe déjà dans la base de données	----------	Succès :
					{
					isPresent:boolean
					}
					Échec :-----
						Succès :200
						Échec :404

GET	/games /visibility	---------	Avoir la liste de tous les jeux visibles dans la base de données	----------	Succès :
					{
					CreateGameDTO[]
					}
					Échec :-----
						Succès :200
						Échec :404

GET	/games /title:name	Name :string	Savoir si le titre d'un jeu existe dans la base de données	----------	Succès :
					{
					isUnique:boolean
					}
					Échec:-----
						Succès :200
						Échec :404

DELETE	/games /:id	Id :string	Delete un jeu dans la base de données	----------	Succès:------- Échec :---------	Succès :204 Échec :404
PATCH	/games /visibility/:id	Id :string	Changer la visibilité d'un jeu dans la base de données	----------	Succès :------- Échec :--------	Succès :204 Échec :404

PUT	/games	----------	Ajouter un jeu à la base de données	{ CreateGameDTO }	Succès :---------- Échec :----------	Succès :201 Échec :400
PUT	/games /modify/:id	Id :string	Modifier un jeu dans la base de données	{ CreateGameDTO }	Succès :---------- Échec :----------	Succès :204 Échec :404
POST	/games /file	----------	Ajouter un jeu à la base de données à partir d'un fichier JSON importé	{ CreateGameDTO }	Succès : -------- Échec :----------	Succès :201 Échec :404
POST	/login	----------	Vérifier le mot de passe entré par l'utilisateur pour accéder à la vue d'admin	{ LoginDTO }	Succès : True Échec :False	Succès : 200 Échec : 200
GET	/record	----------	Récupérer la liste d'historique des jeux Joués en entier.	----------	Succès :{ RecordDto } Échec :------	Succès :200 Échec : 404
DELETE	/record/reset	----------	Remettre à zéro l'historique	----------	Succès :---------- Écher : ---------	Succès : 204 Échec : 500

3.2 Paquets WebSockets

Événement	Source	Description	Données	Événements potentiellement déclenchés
StartGame	Client	Commencer une partie avec d'autres joueurs	*roomId:string*	GiveScore GiveQuestion GiveTotalQuestion TimerTick WaitVerify
StartTestGame	Client	Permet de commencer le test d'un jeu	*currentQuiz : Quiz*	GiveScore GiveQuestion GiveTotalQuestion TimerTick TimerDone
OrganisatorLeft	Client	L'organisateur a quitté la partie	----------	OrgLeft
NextQuestion	Client	L'organisateur change la question du jeu	----------	OrgChangeQuestion ChangeTopBar
ShowResult	Client	L'organisateur veut passer à la vue des résultats	----------	GameResult RoutePlayerResult
PlayerAnswerTest	Client	Reçoit la réponse du client pour la changer côté serveur	*answer :boolean[]*	EveryPlayerAnswered
PlayerAnswerNoSubmit	Client	Reçoit la réponse du client qui n'a pas validé sa réponse pour la changer côté serveur	*answer :boolean[]*	EveryPlayerAnswered
JoinTestRoom	Client	Mettre le client dans une salle de test pour tester le jeu	----------	----------
LeaveTestRoom	Client	Enlève le client de la salle test et la détruit	----------	----------
SubmitTestRoom	Client	En mode test écoute pour la soumission de la réponse du client pour passer à la question suivante	----------	----------
SelectChoice	Client	Reçoit la sélection de réponses du client pour mettre à jour les statistiques côté serveur	*isSelectedArr :boolean[]*	GiveStatAnswer
PlayerAbandon	Client	Enlève le client de la salle et met à jour la liste des joueurs	----------	GiveScore
PlayerLeft	Client	Un joueur a quitté la partie	----------	----------

ChangeTopBar	Serveur	Bascule la visibilité de la top bar	----------	ChangeTopBar
GiveScore	Serveur	Récupère le scoreboard	*scoreboard : Map<string, PlayerScore>*	GiveScores
GiveQuestion	Serveur	Récupère la question et ses réponses	*question : CreateQuestionDto* *answers : boolean[]*	GiveCurrentQuestion GiveCurrentQuestion Answers
GiveStatAnswer	Serveur	Récupère les statistiques de réponses	*statAnswer : number[]*	GiveCurrentStatAnswer
TimerDoneClient	Serveur	Avertit le client que le timer a atteint 0 pour fermer la question	----------	TimerDoneClient PlayerAnswerNoSubmit
EndGame	Serveur	Émet les évènements pour finir la partie	*finalResult : PlayerResult[]*	GameResult GameDone
CanChangeQuestion	Serveur	Émet l'évènement pour changer de question	----------	CanChangeQuestion
EveryPlayerAnswered	Serveur	Affiche les statistiques pour l'organisateur	----------	EveryPlayerAnswered
GiveTotalQuestion	Serveur	Récupère le nombre de questions	*totalQuestion : number*	GiveTotalQuestion
GameResult	Serveur	Envoie les résultats de tous les joueurs à la fin de la partie pour la vue des résultats	*gameResult : PlayerResult[]*	----------
GameDone	Serveur	Avertit le client que la partie est terminée pour envoyer les joueurs dans la vue des résultats ou la sélection des jeux dépendamment du mode test ou game	----------	LeaveTestRoom
GiveCurrentStatAnswer	Serveur	Initialise le diagramme de statistique de réponses	*statAnswer : number[]*	----------
GiveCurrentQuestion Answer	Serveur	Donne les réponses de la question actuelle	*answer :boolean[]*	----------
GiveCurrentQuestion	Serveur	Donne les détails sur la question actuelle du jeu	*question : CreateQuestionDto*	----------
GiveScores	Serveur	Donne l'état du scoreboard après chaque question	*scoreBoard :Array <[string,number]>*	----------
RoutePlayerResult	Serveur	Redirige les membres d'une room à la vue des résultats à la fin d'une partie	----------	----------
OrgLeft	Serveur	Redirige les joueurs vers la page d'accueil	----------	----------
CreateRoom	Client	Crée une nouvelle room	----------	RoomCreated GetRoomID

DeleteRoom	Client	Supprime une room	*roomID : string*	Kicked
JoinGame	Client	Vérifie si le joueur est autorisé à entrer dans le jeu	*data : DataPlayer*	ErrorMessage JoinRoom UpdateRoomPlayer
LeaveRoom	Client	Retire le joueur de la room	----------	RoomLeaved UpdateRoomPlayer
RemovePlayer	Client	Banni le joueur de la room	*data : DataPlayer*	UpdateRoomPlayer Kicked SocketList
LockRoom	Client	Verrouille la room	*data : string*	RoomStateChanged SocketList
Start	Serveur	Initialise le jeu et commence la transition vers la vue de jeu	*data = {roomID : string, quiz :Quiz, players:string[][]}*	SocketList TimerTick DisplayTimer
RemovePlayerRoom	Serveur	Réinitialise la room en déconnectant tous les joueurs	----------	----------
GetRoomID	Serveur	Donne l'identifiant de la room	*roomID : string*	----------
Kicked	Serveur	Redirige le client vers la page d'accueil	----------	----------
UpdateRoomPlayer	Serveur	Met à jour la liste de joueur	*data :string[]*	----------
ErrorMessage	Serveur	Met à jour les messages d'erreur	*data : string*	----------
JoinRoom	Serveur	Met à jour le booléen d'acceptation d'un joueur dans une room	*data : boolean*	----------
RoomStateChanged	Serveur	Met à jour l'état de la room	*data : boolean*	----------
SocketList	Serveur	Met à jour la liste des joueurs	*data = {socketId : Socket, playername : string}*	----------
DisplayTimer	Serveur	Commence la partie dans la vue d'attente	----------	----------
RoutePlayer	Serveur	Envoie les joueurs dans la vue du joueur et l'organisateur dans la vue de l'organisateur	*roomId:string*	StartGame
SendMessage	Client	Initialise le nouveau message	*data : Message*	NewMessage
NewMessage	Serveur	Envoie le message	*data : string*	----------

TimerTick	Serveur	Met à jour l'affichage du timer	*time : number*	----------
TimerHit0	Serveur	Émet l'évènement pour rediriger les joueurs	----------	RoutePlayer
TimerDone	Serveur	Gère la fin de la partie ou le passage à la question suivante	----------	ChangeTopBar TimerDoneClient GiveScore GiveStatAnswer GiveQuestion TimerTick TimerDone WaitVerify EndGame RemovePlayerRoom
OrgChangeQuestion	Serveur	Gère la fin de la partie ou le passage à la question suivante	----------	ChangeTopBar TimerDoneClient GiveScore GiveStatAnswer GiveQuestion TimerTick TimerDone WaitVerify EndGame RemovePlayerRoom
WaitVerify	Serveur	Gère l'attente du client	----------	TimerDoneClient GiveScore CanChangeQuestion
SendQRLAnswer	Client	Envoi la réponse de la QRL au serveur	*data : string*	SendQRLAnswersToOrg
SendQRLAnswersToOrg	Serveur	Envoi les réponses à l'organisateur pour la correction	*data : Map<string, string>*	
CorrectionDone	Client	Envoi la correction de tous les joueurs au serveur	*data : Map <string, number>*	SendGrades
SendGrades	Serveur	Envoi au client la correction de leur réponse	*data : number*	
PauseTimer	Client	Met le timer en pause	-------	
ResumeTimer	Client	Continue le timer s'il est en pause	-------	
StartPanicMode	Client	Premet de commencer le mode panique	-------	PanicModeActivated
EnablePanicMode	Serveur	Rends possible l'activation du mode panique	------	
DisablePanicMode	Serveur	Bloque l'activation du mode panique	------	

PanicModeActivated	Serveur	Informe le joueur de l’activation du mode panique	--------	--------
PlayerChatChange	Client	Active/Désactive le droit d’écrire dans le chat	-------	----------
SubmitForColor	Serveur	Donne le statut d‘envoi d’un joueur	*name : string*	---------
UpdateColor	Serveur	Donne le statut d’interaction d’un joueur	*name : string*	-------
ResetStates	Serveur	Réinitialise les status lors d’une nouvelle question	-------	----------
SendGraphs	Serveur	Envoi les graphiques pour chaque question	*Number[][]*	---------

3.3 Interfaces

Nom	Description	Structure

CreateGameDTO	Information sur la création d'un jeu quiz	{
		id?:number,
		title:string,
		duration:number,
		lastModification:string,
		description:string,
		visibility:Boolean,
		question:CreateQuestionDTO[],
		}

CreateQuestionDTO	nformation sur la création d'une question dans un jeu quiz	{
		type :string,
		text :string,
		points :number,
		choices?:CreateChoiceDTO[],
		
		}

CreateChoiceDTO	formation sur la création d'un choix dans un jeu quiz	{
		Text :string,
		isCorrect?:boolean | null | undefined,
		}

PlayerScore	formations du joueur sur sa partie courante. Contient son score et le nombre de bonus obtenu	{
		Score :number,
		Bonus :number,
		}

PlayerResult	Informations finale de la partie du joueur rmaté exactement comme il le faut pour le leau de material angular	{
		Name:string,
		Score:number,
		Bonus:number,
		Rank:number,
		}

DataPlayerJoin	formation pour mettre l'utilisateur dans une room.	{
		roomCode:string,
		username:string
		}

RecordDto	Information d'un jeu dans l'historique	{
		Name:string
		Date:Date
		Totalplayer:number
		bestScore:number
		}

LoginDto	L'information envoyé par l'utilisateur Lors d'une connection à la vue admin	{
		Password:string
		}

Message	la structure d’un message envoyé et reçu Pour le chat	{
		Id:string
		Sender:string
		Content:string
		Time:Date
		}

GameState	Les informations nécessaires pour Décrire l’état d’un jeu en cours	{
		Questions:CreateQuestionDto[]
		currentQuestion:CreateQuestionDto
		currentIndex:number
		totalQuestion:number
		timeQuestion:number
		isTest:Boolean
		gameName:string
		StartDate:Date
		TotalPlayers:number
		}