LOG2990 – Projet intégrateur développement application web

Automne 2023

Protocol de communication

Groupe vendredi 01
Équipe 103

7 novembre 2023

Historique de version

Date	Version	Description	Auteur
2023-10-23	1.0	Établissement des paquets pour les Websockets de joinRoom et de waitingRoom	Hamza et Arman
2023-11-01	2.0	Établissement des paquets pour les Websockets de j pour le jeu QCM	Hamza, Rayan, Arman et Mehdi
2023-11-03	2.1	Ajout des paquets HTTP pour l'ensemble du site	Rayan et Ely
2023-11-05	2.2	Mise à jour des interfaces pour toutes les communications	Adlane
2023-11-06	3.0	Ajout des des paquets pour les Websockets pour le sprint 3	Rayan, Arman, Hamza' Ely et Adlane

Table des matières

Protocole de communication

1. Introduction

Dans le contexte du cours LOG2990, nous avons été chargés de créer un jeu de questionnaires multijoueurs. L'objectif de ce document est de fournir une description détaillée de toutes les décisions prises concernant le protocole de communication entre le serveur et les clients. Le document est divisé en deux parties : la première traite de la communication entre le client et le serveur, en expliquant les protocoles de communication utilisés et les raisons qui ont motivé ces choix. Enfin, la seconde partie se concentre sur la description des paquets, en détaillant la structure concrète de notre communication, en précisant quelles informations sont échangées dans les paquets lors des communications réseau.

2. Communication Client-Serveur

En ce qui concerne la communication entre le client et le serveur, nous utilisons deux méthodes distinctes de communication. Nous utilisons le protocole WebSocket avec la bibliothèque Socket.io, ainsi que le protocole HTTP avec la bibliothèque ExpressJS.

Pour la page de jeu, seule la communication via le protocole WebSocket est actuellement utilisée, bien que le protocole HTTP puisse être envisagé à l'avenir. Le protocole WebSocket est utilisé pour gérer les choix de réponses, la mise à jour des scores, les abandons de partie, ainsi que le chat entre les joueurs. Cette décision a été prise en raison de la nature des messages, pouvant être transmis à tout moment, ce qui rend une connexion continue plus appropriée. Le même choix a été fait pour la validation des réponses des joueurs. Bien qu'il soit techniquement possible de l'implémenter avec le protocole HTTP, nous avons opté pour WebSocket afin d'assurer une compatibilité totale aussi bien dans les parties multijoueurs que dans les parties en mode test. Dans le cas des parties multijoueurs, il est essentiel que les joueurs et l'hôte restent en communication continue avec le serveur, de même lorsque l'un des joueurs abandonne la partie, l'hôte doit être informé instantanément.

Lors de la création et de la modification d'un quiz, nous utilisons exclusivement le protocole HTTP lors de la validation des questions/réponses et lors de la confirmation de la sauvegarde du quiz. Cette décision a été prise en raison de l'absence d'événements imprévus à gérer, et de la nature séquentielle des requêtes, où une demande est effectuée sans nécessité d'attendre une réponse à un moment indéterminé.

3. Description des paquets

Protocole HTTP:

Chemin (URL)	Méthode	Requête (corps)	Code de retour		Réponse	
			Succès	Erreur	Succès	Erreur
/	**-**	-	200	500	Swagger	{ erreur }
/api/docs	**-**	-	200	500	Swagger	{ erreur }
/api/quiz/	**GET**	-	200	500	Quiz[]	{ erreur }
/api/quiz/visible	**GET**	-	200	500	Quiz[]	{ erreur }
/api/quiz/:id	**GET**	-	200	500	Quiz	{ erreur }
/api/quiz/	**POST**	{ Quiz }	201	500	Quiz	{ erreur }
/api/quiz/	**PUT**	{ Quiz }	200	500	Quiz	{ erreur }
/api/quiz/:id	**PATCH**	{ visible : boolean }	200	500	{ visible : boolean }	{ error: string }
/api/quiz/checkTitleUniqueness	**POST**	{ title : string }	200	500 400	{ isUnique : boolean }	{ erreur }
/api/quiz/:id	**DELETE**	-	200	500	-	{ erreur }
/api/auth/admin-password	**POST**	{ password : string }	200 ou 401	500	{ message : string }	{ erreur }

Protocole WebSocket:

Nom de l'évenement	Source	Description	Destination	Données
'endQuestion'	Serveur	Avertit la fin d'une question	Client	
'getScore'	Client	Le client obtient son score	Serveur	roomId: number, username: string
'timeTransition'	Serveur	Envoie événement de la transition entre deux questions chaque seconde	Client	timeValue: number
'finalTimeTransition'	Serveur	Dernière transition (entre dernière question et vue de résultats) envoyée à chaque seconde	Client	timeValue: number
'removedFromGame'	Serveur	Fait quitter le joueur de la vue d'attente ou de la vue du joueur	Client	
'startTransition'	Client	L'organisateur demande au serveur de commencer la transition entre deux questions	Serveur	roomId: number
'showResult'	Client	L'organisateur demande au serveur d'envoyer les résultats finaux à chaque joueur	Serveur	roomId: number
'nextQuestion'	Client	L'organisateur demande au serveur de passer à prochaine question	Serveur	roomId: number

‘refreshChoicesStats’	Serveur	Le serveur envoie les statistiques utilisées pour l’histogramme à temps réel	Client	choicesStatsValue: number[]
‘getInitialQuestion’	Serveur	Le serveur envoie les données nécessaires pour la première question	Client	data: InitialQuestionData
‘getNextQuestion’	Serveur	Le serveur envoie les données de la question demandée par l’organisateur	Client	data: NextQuestionData
‘removedPlayer’	Serveur	Le serveur informe toute la salle qu’un joueur à quitté	Client	username: string
‘gatherPlayersUsername’	Client	Un client demande au serveur la liste des joueurs afin d’avoir le classement des joueurs	Serveur	roomId: number
‘playerJoin’	Client	Lorsqu’un joueur join la partie, avec son nom d’utilisateur	Serveur	roomId: number, username: string
‘validateUsername’	Client	Un client demande au serveur de valider sa réponse, la réponse est retournée en callback	Serveur	isValid: boolean, error: string
‘validateRoomID’	Client	Permet de valider si on est dans le bon salon avec le	Serveur	roomId: number

		roomID		
‘newMessage’	Client	Permet au client d’envoyer un nouveau message dans le chat	Serveur	roomId: number, message: {sender : string, content: string, time: string}
‘getMessages’	Client	Permet au client de recevoir un message emit par un autre joueur afin de l’afficher dans le chat	Serveur	roomId: number
‘getUsername’	Client	Demande l’accès à son propre nom d’utilisateur pour utiliser dans le clavardage	Serveur	roomId: number
‘messageReceived’	Serveur	Lorsqu’un joueur envoie un message, le serveur envoie cet événements avec le message à tout les utilisateurs	Client	message: Message
‘stopTimer’	Client	Permet d'arrêter la minuterie	Serveur	roomId: number
‘createRoom’	Client	Permet à l’organisateur de créer un room unique	Serveur	roomId: number
‘banPlayer’	Client	Permet à l’organisateur de bannir un joueur de la partie indéfinemement	Serveur	roomId: number, username: string
‘toggleRoomLock’	Client	Permet à l’organisateur de verrouiller le room avant de lancer la partie	Serveur	roomId: number

‘start’	Client	Permet à l’organisateur de lancer la partie et de notifier tous les joueurs du début de la partie	Serveur	roomId: number, time: number
‘gatherPlayersUsername’	Client	Permet au client de récupérer le nom des joueurs d’un game	Serveur	roomId: number
‘newPlayer’	Serveur	Notifie les client de l’apparition d’un nouveau joueur	Client	players: string[]
‘removedFromGame’	Serveur	Notifie le client qui a été déconnecté de la partie afin de fermer son socket et le rediriger	Client	
‘removedPlayer’	Serveur	Notifie tout les clients du room du retrait d’un joueur	Client	username: string
‘time’	Serveur	Permet l’envoie du temps pendant le jeux sans mode panique	Client	timeValue: number
‘submitAnswer’	Client	Soumission des réponses finales du client	Server	roomId: number, answers : string | null, timer: number, username: string
‘updateSelection’	Client	Enregistrer les sélections de réponse du client	Serveur	roomId: number, isSelected: boolean, index: number
‘pause timer’	Client	permet de mettre la	Serveur	roomId: number

		minuterie en pause niveau client		
‘pause timer’	Server	permet de mettre la minuterie en pause niveau server	Client	
‘panic mode’	Client	Quand l’organisateur active le mode panique	Serveur	roomId: number
‘panic mode’	Serveur	Le serveur broadcast à tous les clients du room le nouveau temps	Client	
‘modified QRL’	Client	Permet au server de tiendre en compte les modifications	Serveur	roomId: number username: string
‘updateQRLData’	Serveur	Permet de mettre à jour l’histogramme (décrémente si vrai, incrémente si faux)	Client	fiveSecExpired : boolean
‘QRLending’	Serveur	Indique au joueur que l’evaluation est en cours	Client	roomId: number
‘QRLEvaluation’	Client	Quand l’organisateur note la réponse d' un joueur	Serveur	roomId: number username: string multiplier : number
‘AllQRLEvaluated’	Serveur	Quand tous les reponses QRL sont evalues	Client	
‘banUserChatting’	Client	Quand organisateur bannie les chat d’un joueur	Serveur	roomId: numbe username: string
‘chatBanned’	Serveur	Emis a un client	Client	isBanned:

		pour lui dire qu'il peut chat ou non		boolean

Interfaces et Enums utilisés

Common Interfaces:

interface NextQuestionData :
    question: QuizQuestion;
    index: number;
    isLast: boolean;
Description : Contient l'information pour passer à la prochaine question

interface InitialQuestionData :
    question: QuizQuestion;
    username: string;
    index: number;
    numberOfQuestions: number;
Description : Contient les informations nécessaires pour avoir la première question

interface Message :
    sender: string;
    content: string;
    time: string;
Description : Contient l'information nécessaire pour envoyer un message

enum QuestionType :
    QCM = 0;
    QLR = 1;
Description: Permet de différencier le type de question utilisé

interface Quiz :
    id: string;
    title: string;
    description: string;
    duration: number;
    lastModification: string | null;
    questions: QuizQuestion[];
    visible?: boolean;
Description: Contient l'information nécessaire pour la création d'un quiz

interface QuizQuestion :
    type: QuestionType;
    text: string;
    points: number;
    choices?: QuizChoice[];
Description: Contient les informations nécessaires à une question d'un quiz

interface QuizChoice :
    text: string;
    isCorrect?: boolean | null;
Description: Contient les informations nécessaire à un choix de réponse dans une question

interface Score :
    points: number;
    bonusCount: number;
    isBonus: boolean;
Description: Contient les informations nécessaires pour pouvoir utiliser un score

interface PlayerUsername :
    roomId: number;
    username: string;
Description: Contient les informations nécessaires d'un joueur pour pouvoir le connecter correctement

interface PlayerMessage :
    roomId: number;
    message: Message;
Description: Contient les informations nécessaires pour affilié un message envoyé à un joueur

interface PlayerTime :
    roomId: number;
    time: number;
Description: Contient les informations nécessaires pour pouvoir savoir le temps de réponses d'un joueur

interface PlayerAnswerData :
    roomId: number;
    answers: string[];
    timer: number;
    username: string;
Description: Contient les informations nécessaires pour pouvoir mettre à jour les réponses validés d'un joueur

interface PlayerSelection :
    roomId: number;
    isSelected: boolean;
    index: number;

Description: Contient les informations nécessaires pour pouvoir mettre à jour les sélections de joueurs sur une question

Client Interfaces:

interface GameServiceInterface :
    question: QuizQuestion | null;
    locked: boolean;
    validated: boolean;
Description: Contient les informations utile afin d'initier l'interface de jeu

Server Interfaces:

interface Answers :
    answers: string[];
    time: number;
Description: Contient les informations nécessaires pour pour pouvoir enregistrer les informations sur les réponses choisis d'un joueur en plus de son temps