101

Protocole de communication

Version 2.5

Historique des révisions

Date	Version	Description	Auteur
2023-03-18	1.0	Ajout des features utilisés avec leur protocoles associés (sprint 1 et 2)	Zakaria Zair
2023-03-19	1.1	Modification des features avec leur protocoles associées et ajout de ceux du (sprint 3)	Zakaria Zair
2023-03-20	1.2	Ajout des descriptions des paquets (sprint 1 et 2)	Zakaria Zair
2023-03-20	1.2	Modification des descriptions des paquets (sprint 1 et 2)	Zakaria Zair
2023-03-20	1.3	Ajout des descriptions des paquets (sprint 3)	Zakaria Zair
2023-03-21	1.3	Révision du document	Edgar Kappauf
2023-04-15	2.0	Ajout d'une introduction pour la fonctionnalité principale	Zakaria Zair
2023-04-15	2.1	Modification de l'introduction et protocole	Zakaria Zair
2023-04-16	2.2	Modification des raisons d'utilisations	Zakaria Zair
2023-04-17	2.2	Modification des raisons d'utilisations	Zakaria Zair
2023-04-18	2.3	Ajustement des événement du protocole WebSocket et regroupement par fonctionnalité	Sulayman Hosna
2023-04-18	2.4	Ajustement du protocole HTTP et réorganisation des informations de requête et de réponse	Zakaria Zair
2023-04-19	2.4	Description des interfaces	Edgar Kappauf, Jeremy Ear, Abderrahim Zebiri et Sulayman Hosna
2023-04-19	2.5	Révision du document	Abderrahim Zebiri et Zakaria Zair

Table des matières

Protocole de communication

1. Introduction

Ce document décrit l'implémentation des fonctionnalités d'un projet de communication client-serveur. Il présente les protocoles de communication utilisés, ainsi que la description des paquets échangés entre les clients et le serveur. Les protocoles de communication utilisés dans ce projet sont HTTP et WebSocket, chacun ayant des fonctions spécifiques pour permettre une communication efficace et rapide. Les paquets échangés sont décrits en détail, y compris les informations transmises et leurs formats pour chaque protocole.

2. Communication client-serveur

Les protocoles utilisés pour la communication client-serveur du projet sont WebSocket et HTTP. Le protocole HTTP est utilisé là où une communication bi-directionnelle en temps réel n'est pas nécessaire tandis que le protocole WebSocket est utilisé là où une communication bi-directionnelle en temps réel est nécessaire et pour que le serveur force les clients à demander les données à jour.

La fonctionnalité principale du projet est le déroulement d'une partie. Le concept du jeu s'inspire du jeu des sept différences. Il est possible de jouer en mode classique, tout seul ou en un contre un, et également en mode temps limité, tout seul ou en coopération. Durant une partie tout seul, en mode classique, le joueur doit trouver les différences entre les deux images qui lui sont présentées. En mode temps limité, le joueur doit passer à travers tous les jeux dans le système. Pour faire cela, il doit trouver une seule différence dans un jeu pour passer au prochain jusqu'à qu'il y en ait plus. Pour le multijoueur, les règles en mode solo sont reprises, avec certaines différences : en mode classique, le joueur qui a trouvé la majorité des différences en premier remporte la partie. En mode temps limité, les deux joueurs s'entraident à trouver les différences le plus rapidement possible.

Fonctionnalités	**Protocole utilisé**	**Raison d'utilisation**
Créer un jeu	HTTP et WS	HTTP sert à ajouter ou supprimer un jeu ou tous les jeux dans la base de données et WS permet de notifier à tous les clients "ciblés" en temps réel la mise à jour du carrousel.
Supprimer un jeu	HTTP et WS	
Supprimer tous les jeux	HTTP et WS	
Réinitialiser à défaut les meilleurs temps et voir ces derniers	HTTP et WS	
Voir les constantes de jeu	HTTP et WS	HTTP sert à récupérer les données et les charger dans la vue de configuration et WS sert à envoyer au serveur les données afin de les mettre à jour dans la base de données et ainsi de notifier à tous les clients "ciblés" en temps réel la mise à jour de la vue de configuration.
Voir l'historique des parties	HTTP et WS	
Supprimer l'historique des parties	HTTP et WS	
Réinitialiser à défaut les constantes de jeu	HTTP et WS	
Entrer un nom de jeu pour créer un jeu	HTTP	HTTP sert à vérifier dans la base de données si le nom du jeu qui est entré existe ou non parmi ceux dans la base de données
Voir les fiches de jeu	HTTP	HTTP sert à récupérer les fiches de jeux en groupe de carrousel lorsqu'on charge le component et permet de naviguer à travers un carrousel différent si le bouton

		suivant/précédent est appuyé
Créer une partie en multijoueur (mode classique)	WS	WS est utilisé pour notifier le serveur en temps réel si une room est en train de se faire créer. Le créateur communique au serveur la personne acceptée.
Créer une partie en multijoueur (mode temps limité)	WS	WS est utilisé pour notifier le serveur en temps réel si une room est créée dans le mode coopératif.
Rejoindre en partie en multijoueur (mode classique)	WS	WS est utilisé pour notifier le serveur en temps réel si un joueur rejoint une certaine room. Le joueur qui a rejoint est en état d'attente, jusqu'à ce que le serveur notifie les joueurs en état d'attente. Le serveur communique aux joueurs qui ont été refusés et celui accepté pour qu'ils soient redirigés aux bons endroits.
Rejoindre en partie en multijoueur (mode temps limité)	WS	WS est utilisé pour notifier le serveur en temps réel si un joueur rejoint le mode. Le serveur notifie le créateur lorsqu'une personne rejoint le mode coopératif et démarre la partie pour les deux joueurs.
Démarrer une partie de jeu (en solo ou multijoueur)	WS	WS est utilisé pour transmettre les informations nécessaires pour démarrer une partie et, en multijoueur, s'assure que les deux joueurs commencent la partie en même temps. Le serveur communique également toutes les informations nécessaires pour le bon déroulement de la partie.
Abandonner une partie	WS	WS est utilisé pour communiquer en temps réel le fait qu'un joueur a abandonné la partie et transmettre, si c'est le cas, à l'autre joueur la réponse adaptée.
Mise à jour du compteur	WS	WS est utilisé pour que le serveur notifie la valeur du compteur aux clients.
Cliquer sur un endroit dans un canva	WS	WS est utilisé pour communiquer aux joueurs si la différence à été trouvé ou non, et par qui, ainsi que pour vérifier si une partie est terminée ou non. Le serveur notifie ensuite aux joueurs de la room ciblé la réponse appropriée.
Activer un indice	WS	WS est utilisé pour notifier un indice a été activé. Le compteur est donc mis à jour selon le mode jeu et envoyé aux clients.
Clavarder un mode multijoueur	WS	WS est utilisé pour le clavardage entre joueurs (envoyer et recevoir)
Recevoir les messages de la console (messages locaux et globaux)	WS	WS est utilisé pour recevoir les messages de la console en temps réel (différences trouvés, erreur, abandon du jeu, meilleur temps, etc..)

3. Description des paquets

Protocole HTTP :

Méthode	Route *:3000/api	Paramètre(s) (requête)	Type du corps (requête)	Informations du paquet (réponse)	Type du corps (réponse)	Code(s) de retour (réponse)
GET	/api/games/constants	-	-	Les constantes de jeu	GameConfigConst	Réussite:200 Echec:404
PUT	/api/games/constants	Les constantes de jeu mis à jour	GameConfigConst	Les constantes de jeu	GameConfigConst	Réussite:200 Echec:204
GET	/api/games/history	-	-	L'historique des parties	GameHistory	Réussite:200 Echec:404
DELETE	/api/games/history	-	-	-	-	Réussite:200 Echec:204
GET	/api/games/carousel/{index}	Index du carrousel à afficher	string	L'ensemble de quatre fiche de jeu à présenter au client	CarouselPaginator	Réussite:200 Echec:404
DELETE	/api/games/{id}	Index du jeu à supprimer	string	-	-	Réussite:200 Echec:204
GET	/api/games	Nom du jeu	string	Si le nom du jeu existe déjà ou non	boolean	Réussite:200 Echec:404
POST	/api/games	Informations du jeu créé	GameDetails	-	-	Réussite:201 Echec:400
DELETE	/api/games	-	-	-	-	Réussite:200 Echec:204

* La route commence par l'adresse du serveur suivi par le port et /api

Interface	Description	Contenu
GameDetails	Elle contient les informations nécessaire pour construire un jeu et assigner les bonnes informations de jeu	name: string; originalImage: string; modifiedImage: string; nDifference: number; differences: Coordinate[][]; isHard: boolean;
Players	Elle permet de réunir les 2 joueurs qui vont jouer ensemble dans une même room.	player1: Player; player2?: Player;
Player	Elle contient les informations sur un joueur	playerId?: string; name: string; differenceData: Differences;
PlayerInfo	Elle permet de contenir les informations utiles d'un joueur qui vont être inscrit dans l'historique	name: string; isWinner: boolean; isQuitter: boolean;
ClientSideGame	Elle contient les données utilisés lors d'une partie (peu importe le mode), c'est-à-dire l'affichage des deux images, du nom du jeu, du mode de jeu, du niveau de difficulté et du compteur de différences	id: string; name: string; mode: string; original: string; modified: string; isHard: boolean; differencesCount: number;
GameCard	Elle contient les informations des cartes de jeu. Ce sont ces attributs qui vont être affichés pour chaque jeu dans la vue de sélection et de configuration.	_id: string; name: string; difficultyLevel: boolean; soloTopTime: PlayerTime[]; oneVsOneTopTime: PlayerTime[]; thumbnail: string;
CarouselPaginator	Elle contient les cartes des jeux avec deux propriétés booléens pour informer le client si il y a encore des cartes à afficher ou non (suivant et précedent)	hasNext: boolean; hasPrevious: boolean; gameCards: GameCard[];
GameConfigConst	Elle contient les constantes de jeu qui sont configurables : temps du compte à rebours, temps de pénalité et temps du bonus	countdownTime: number; penaltyTime: number; bonusTime: number;
GameRoom	Collection de données qui représente une salle qui contient toutes les infos nécessaires dont on a besoin pour le bon déroulement d'une partie	roomId: string; clientGame: ClientSideGame; endMessage: string; timer: number; originalDifferences: Coordinate[][]; gameConstants: GameConfigConst; player2?: Player; player1: Player;

ChatMessage	Message du chat	tag: MessageTag; message: string;
Coordinate	Coordonnées x et y	x: number; y: number;
NewRecord	Collection des données nécessaires à la construction des messages de nouveaux records dans la boîte de clavardage	gameName: string; playerName: string; rank: number; gameMode: string;
WaitingPlayerNameList	Elle contient le nom des joueurs qui sont en attente d'être acceptés pour une certaine partie d'un jeu.	gameId: string; playerNamesList: string[];
AcceptedPlayer	Elle contient le nom du joueur qui a été accepté pour rejoindre une partie et la salle que celui-ci et son partenaire vont rejoindre.	gameId: string; roomId: string; playerName: string;
PlayerNameAvailability	Permet de vérifier si le nom d'un joueur qui veut rejoindre une certaine partie est disponible (éviter d'avoir deux joueurs avec le même nom)	gameId: string; isNameAvailable: boolean;
RoomAvailability	Elle permet de vérifier si un joueur peut rejoindre une partie d'un certain jeu.	gameId: string; isAvailableToJoin: boolean; hostId: string;
Differences	Elle contient les coordonnées d'une différence et le nombre de différences trouvés	currentDifference: Coordinate[]; differencesFound: number;
TimerMode	Elle contient les informations sur le compteur : si il est en train de compter et si il faut un deuxième joueur	isCountdown: boolean; requiresPlayer2?: boolean;

Protocole WebSocket :

Tableau des événements liés à une partie en cours : **GameEvents**

Nom de l'événement	**Source**	**Contenu**
CheckStatus (client demande au serveur si la partie est fini)	Client	*Vide*
EndGame	Serveur	endingMessage: string
TimerUpdate	Serveur	timer: number
RemoveDifference	Serveur	coords: Coordinate[]
GameStarted	Serveur	room: GameRoom
AbandonGame	Client	*Vide*
StartGameByRoomId	Client	roomId: string et playerName: string
StartNextGame	Client	*Vide*
RequestHint	Client	*Vide*
UpdateDifferencesFound	Client	differencesFound: number
GameModeChanged	Server	*Vide*
GamePageRefreshed	Server	*Vide*

Tableau des événements de la fonctionnalité : Recevoir les messages de la console: **MessageEvents**

Nom de l'événement	**Source**	**Contenu**
GlobalMessage	Client	tag: MessageTag.received et message: textMessage ou ChatMessage
LocalMessage	Client	tag: MessageTag.received et message: string

Tableau des événements liés à la gestion des joueurs : **PlayerEvents**

Nom de l'événement	Source	Contenu (variable: type)
PlayerRefused	Server	playerId: string
GetJoinedPlayerNames (Le client créateur envoie une socket au serveur pour obtenir une liste à jour du nom des joueurs en liste d'attente)	Client	gameId: string
PlayerAccepted	Serveur	isAccepted: boolean
AcceptPlayer	Client	gameId: string, roomId: string, playerName: string
CancelJoining (sur cet événement Le client créateur envoie une socket pour notifier le serveur sur quel joueur est refusé)	Client	roomId: string et playerName: string
RefusePlayer	Client	playerPayLoad: PlayerData
PlayerNameTaken	Client	playerNameAvailability: PlayerNameAvailability
UpdateWaitingPlayerNameList (si un joueur rejoint une partie, on update la liste des joueurs en attente)	Client	playerPayLoad: PlayerData
CheckIfPlayerNameIsAvailable	Client	*Vide*
WaitingPlayerNameListUpdated (réponse du serveur à l'événement UpdateWaitingPlayerNameList)	Serveur	waitingPlayerNameList: string[]

Tableau des événements liés à la gestion du carrousel : **GameCardEvents**

(sélectionPage et configurationPage)

Nom de l'événement	**Source**	**Contenu**
ResetTopTime	Client	gameId: string
ResetAllTopTimes	Client	*Vide*
GameCardDeleted	Client	gameId: string
RequestReload	Serveur	*Vide*
AllGamesDeleted	Client	*Vide*
GameDeleted	Serveur	gameId: string
GameConstantsUpdated	Client	*Vide*
GamesHistoryDeleted	Client	*Vide*

Tableau des événements liés à la gestion de l'historique : **HistoryEvents**

Nom de l'événement	**Source**	**Contenu**
RequestReload	Serveur	*Vide*

Tableau des événements liés à la gestion des rooms : **RoomEvents**

Nom de l'événement	**Source**	**Contenu**
CreateClassicSoloRoom	Client	*Vide*
RoomSoloCreated	Serveur	roomId: string
RoomLimitedCreated (informe le client que la room est créé)	Serveur	roomId: string
CreateLimitedRoom	Serveur	playerPayLoad: PlayerData
RoomOneVsOneCreated	Serveur	roomId: string
RoomOneVsOneAvailable	Serveur	availabilityData: RoomAvailability

CreateOneVsOneRoom	Client	playerPayLoad: PlayerData
CheckRoomOneVsOneAvailability	Client	gameId: string
UpdateRoomOneVsOneAvailability (Client informe le serveur que une salle est joignable pour un jeux en particulier)	Client	gameId: string
OneVsOneRoomDeleted	Serveur	availabilityData: RoomAvailability
UndoRoomCreation	Serveur	gameId: string
DeleteCreatedOneVsOneRoom	Client	roomId: string
JoinOneVsOneRoom	Serveur	*Vide*
CheckIfAnyCoopRoomExists	Client	playerPayLoad: PlayerData
LimitedCoopRoomJoined (informe le client qu'il peut commencer le jeu en temps limité en mode coopératif)	Serveur	*Vide*
DeleteCreatedCoopRoom	Serveur	roomId: string
NoGameAvailable	Serveur	*Vide*