209

Protocole de communication

Version 1.0

Historique des révisions

Date	Version	Description	Auteur
2023-03-18	1.0	L’introduction	Radwan Rahman
2023-03-19	1.0	communication client & desc. paquets	Radwan Rahman
2023-03-20	1.2	paquets HTTP	Radwan Radwan
2023-03-20	1.2	paquets HTTP	Skander Hannachi
2023-03-20	1.2	paquets WebSocket	Mohamad Awad
2023-04-16	2.1	Mise a jour Intoduction	Radwan Rahman
2023-03-20	2.2	Mise a Jour Communication Client-Serveur	Zied Kaabi
2023-03-20	2.3	Mise a jour paquets WebSocket	Mohamad Awad

Table des matières

Protocole de communication

1. Introduction

Notre site web de jeu propose une expérience interactive et amusante. Vous êtes invité à observer attentivement deux images similaires et à identifier toutes les différences entre ces dernières. Ce jeu classique est présenté sous une forme moderne et en ligne, avec des graphismes attrayants et des fonctionnalités intuitives pour vous aider à jouer seul ou avec un adversaire, et à améliorer votre score.

Pour que vous puissiez profiter pleinement de notre jeu de trouver les différences, nous avons développé une architecture de communication efficace entre le client et le serveur. Cette communication se fait grâce à une technologie avancée qui assure une réponse rapide et une expérience de jeu fluide. On utilise une combinaison de technologies front-end et back-end pour créer cette architecture de communication, qui permet une transmission de données rapide, efficace et sécurisée entre le client et le serveur.

Notre application est conçue pour être facile à utiliser, avec une interface utilisateur intuitive qui permet aux joueurs de commencer à jouer rapidement et facilement. Nous avons également inclus des fonctionnalités supplémentaires, tels q'un chronomètre, un compteur de score et des indices pour aider les joueurs à suivre leur progression et à améliorer leurs compétences. Notre jeu de trouver les différences est divertissant tout en stimulant votre concentration et acuité visuelle. En plus des fonctionnalités de base, nous avons inclus des options de configuration pour personnaliser votre jeu selon vos préférences. Nous avons intégré la fonctionnalité de compte à rebours, qui vous permet de régler le temps fixé pour trouver toutes les différences entre les images. Cette fonctionnalité ajoute un niveau supplémentaire de défi et d'excitation à vos parties, car vous devrez trouver toutes les différences avant que le temps ne s'écoule. Vous pouvez également régler les pénalités de temps et le temps boni accordé pour avoir trouvé une différence pour rendre le jeu encore plus intéressant et stimulant.

En bref, notre site web de jeu de trouver les différences offre une expérience de jeu amusante, interactive et moderne, avec une architecture de communication efficace entre le client et le serveur. Réussirez-vous à être en haut des classements? Testez vous dès maintenant!

2. Communication client-serveur

Pour notre projet d'application web de jeu, la communication entre le client et le serveur est un élément fondamental pour offrir une expérience utilisateur interactive et sans faille. Afin d'atteindre cet objectif, le choix de la technologie de communication appropriée est primordial. Initialement, nous avons choisi les requêtes HTTP comme technologie de communication, mais avec la fin des 2 sprints, nous avons eu une rétrospection collective, et avons donc opté pour les WebSockets pour la majorité des fonctionnalités de notre application, car elles proposent plusieurs avantages.

Les WebSockets sont une technologie de communication en temps réel qui facilite l'échange bidirectionnel de données entre les clients et les serveurs, sans attendre une réponse du serveur avant de continuer. Les jeux en ligne bénéficient particulièrement de cette fonctionnalité, car les délais de latence peuvent considérablement altérer l'expérience utilisateur. Les WebSockets sont également adaptés aux applications web modernes, car ils offrent une connexion persistante qui permet une utilisation optimale des ressources et une réduction de la charge du serveur . Cette technologie nous a permis d'établir une communication de bas niveau tout en permettant de personnaliser davantage la façon dont les données sont envoyées et reçues entre le client et le serveur. Enfin, les WebSockets permettent une mise à jour continue de tous les clients de l'application, ce qui est un avantage crucial pour les jeux en ligne, tel qu'il est le cas de notre projet.

Au premier sprint, nous n'avons fait que des paquets de requêtes HTTP. Les paquets avaient le type de méthode, la route, le corps et/ou le(s) paramètre(s) et/ou une réponse associée. Nous avons plusieurs paquets qui ont été renvoyés du client, et chacuns de ses paquets sont expliqués et schématisés ci-dessous.

Avec la fin des deux premiers remises, nous avons eu un changement d'idées, et avons décidé de changer pratiquement tous les types de paquets pour la communication client-serveur vu qu'on avait pas le choix pour certaines fonctionnalités sauf pour la comparaison d'images à la création d'une nouvelle fiche vu que ça serait plus fluide avec les requêtes traditionnelles.

Par exemple dans le cadre des parties multijoueur on était obligé de faire communiquer deux joueurs ensemble, chose qu'on pouvait pas faire avec les requêtes HTTP, nous avons donc choisi de procéder avec le même protocole pour tous les modes de Jeu par souci d'homogénéité, et donc changer la logique des sprint 1 et 2.

En fonction des besoins, les WebSocket offrent plusieurs operations pour une communication en temps reel, on cite notamment :

fonction	définition
.emit()	fonction d'émission qui permet de transférer des données
.broadcast()	fonction de diffusion qui permet d'envoyer des données à tous les clients connectés à la même socket , excepté à l'émetteur
.on()	fonction de réception qui permet d'écouter les données envoyées

Les attributs principaux du socket (socket.id, socket.rooms et socket.data) sont importants pour toutes ces opérations.

+ Pour les événements qui sont générés par l'utilisateur dans le contexte d'une partie comme la création d'une parti Multijoueur ou les clic sur les différences, le serveur écoute constamment ces événement avec **.on()** qui sont envoyés par le client à l'aide de **.emit()**

+ Pour les événement demandes par l'utilisateur tel que l'historique des parties , les constantes de jeu ou l'utilisation des indices , le client envoie une demande avec **.emit()** le serveur a son tour écoute la demande a l'aide de **.on()** et y répond immédiatement en envoyant sa réponse avec **.emit()** qui sera écouté par le client avec la fonction **.on()**

+ Pour les événements reçus au deuxième joueur dans le cadre d'une partie classique 1v1 ou TL COOP comme les cliques de l'adversaire/allie ainsi que les messages texte, le serveur reçoit l'information et la diffuse avec .broadcast() le client a son tour écoute ces événements à l'aide de .on()

Protocoles de Communications entre client et serveur Sprint 2:

HTTP	WebSocket
Chercher toutes les fiches des Jeux	détecter les cliques et récupérer les coordonnées
Chercher une fiche de Jeu par son ID	Message de parties (messages locaux + evénements)
Supprimer une fiche de Jeu avec son ID	Changer l'attribut "isJoinable" d'une fiche de Jeu pour mode Classique multijoueur
Mettre à jour d'une fiche de Jeu	Gérer les images des différences pour le mode triche
Comparer les images (originale et modifiée) et vérifier les coordonnées de différence pour la creation	
Afficher les meilleurs temps des Jeux	

Protocoles de Communications entre client et serveur Sprint 3:

HTTP	WebSocket
Comparer les images (originale et modifiée) et vérifier les coordonnées de différence pour la creation	Chercher une fiche de Jeu par son ID
	Chercher toutes les fiches des Jeux
	Supprimer une fiche de Jeu avec son ID
	Supprimer toutes les fiches des Jeux
	Afficher et comparer les meilleurs temps des Jeux
	Manipuler et chercher les constantes de Jeu
	Message de parties (messages locaux + evénements)

	Indices de Jeux
	Gérer les images des différences pour le mode triche
	Changer l'attribut "isJoinable" d'une fiche de Jeu pour mode Classique multijoueur
	Mettre à jour d'une fiche de Jeu
	Remettre les données des Jeux à leur état initial
	Changer l'attribut "isJoinable" d'une fiche de Jeu pour mode TL COOP
	Gérer les fiches de Jeux pour le mode TL (solo & coop)
	Messages Globaux

3. Description des paquets

Pour permettre une visualisation des paquets, nous avons des schémas des paquets de protocoles HTTP, ainsi que les protocoles Websockets.

Voici les paquets de communication entre le client et le serveur qui ont été créé:

3.1 Protocole HTTP

Cas d'utilisation : *Envoie et réception des fichiers dans la vue de création*

Dans la vue de création, un utilisateur a pour option de téléverser des images afin de créer une fiche de jeu qui inclut les URL des images téléversées, soit l'image originale et l'image altérée. Le serveur doit également envoyer un feedback afin de valider le nombre de différences générées par la sélection/création des images utilisées pour la fiche. Si le nombre de différences respecte la contrainte imposée par le projet, soit entre 3 et 9 différences, la génération de la fiche de jeu aura lieu. Dans le cas contraire, l'utilisateur sera prévenu que ses images entrées ne respectent pas la norme et devra téléverser de nouvelles images jusqu'à ce que le nombre de différences soit valide.

Contrôleurs utilisés ainsi que ses routes utiles au cas d'utilisation

ImageController /image			
TYPE	ENDPOINT	PARAMS / BODY	UTILITÉ
POST	/compare	Body	Cette route a pour but d'intercepter les deux fichiers d'image téléversés (contenu dans le corps de la requête), de comparer les deux images en faisant appel au service de détection des différences et de renvoyer au client le nombre de différences trouvées entre les deux images.

SheetController /sheet		
TYPE	ENDPOINT	UTILITÉ
POST	/	Cette route a pour but d'intercepter les deux fichiers d'image téléversés, elle se fera appelée par le client associé lors de la confirmation que les deux images contiennent bel et bien le nombre de différences requis. De plus, le middleware se chargera d'effectuer une création d'un objet *Sheet.* Cet objet est une interface respectant le schéma des objets contenus dans la base de données, il représentera une partie créée par l'utilisateur et sera stocké.

Cas d'utilisation : *Affichage des parties créées dans la vue de sélection*

Lorsque l'utilisateur se dirige vers la vue de sélection, il devrait pouvoir obtenir un affichage complet des parties présentes dans la base de données afin qu'il puisse sélectionner la fiche de jeu sur laquelle il voudra jouer.

Contrôleurs utilisés ainsi que ses routes utiles au cas d'utilisation

SheetController /sheet			
TYPE	ENDPOINT	PARAMS/ BODY	UTILITÉ
GET	/		Cette route a pour but de renvoyer au client l'ensemble des fiches de jeu présentes dans la base de données afin de les présenter à l'utilisateur, le middleware fera un appel find générique pour retourner un tableau contenant l'ensemble des fiches de jeu. La réponse contiendra ce tableau.

Cas d'utilisation : *Suppression d'une fiche de jeu dans la vue de création*

Sur la vue de création, l'utilisateur a pour option de cliquer sur un bouton de suppression associé à une fiche de jeu en particulier. Il enverra ainsi une requête au serveur pour le prévenir que cette fiche est à retirer de la base de données.

Contrôleurs utilisés ainsi que ses routes utiles au cas d'utilisation

SheetController **/sheet**			
TYPE	ENDPOINT	PARAMS/ BODY	UTILITÉ
GET	/:id	PARAMS	Cette route a pour but d'abord de récupérer le paramètre *id* envoyé lors de la requête afin de supprimer la fiche de jeu associée à l'identifiant envoyé. Le middleware va effectuer une requête de suppression vers la base de données sur la fiche de jeu voulue. Sur une suppression réussie le serveur renvoie OK et sinon NOTFOUND.

Cas d'utilisation : *Récupération de la fiche de jeu ainsi que des images utilisées au lancement d'une partie*

Lorsque l'utilisateur aura sélectionné la partie sur laquelle il désire jouer, il sera redirigée vers la vue du jeu. À ce moment, le client demandera au serveur de lui servir les images de jeu, originales et altérées ainsi que la fiche complète de jeu.

Contrôleurs utilisés ainsi que ses routes utiles au cas d'utilisation

SheetController **/sheet**			
TYPE	ENDPOINT	PARAMS/ BODY	UTILITÉ
GET	/:_id	PARAMS	Cette route a pour but d'abord de récupérer le paramètre *_id* envoyé lors de la requête. Le middleware récupérera alors depuis la base de donnée la fiche de jeu associée à l'identifiant voulu et se chargera de la transmettre au client avec le message OK, dans le cas où la fiche n'existe pas une réponse NOTFOUND sera renvoyée au client.

ImageController **/image**			
TYPE	ENDPOINT	PARAMS / BODY	UTILITÉ
GET	/:filename	PARAMS	Cette route a pour but de récupérer le nom du fichier associé à l'image désirée et de renvoyer au client le fichier dans son intégralité pour pouvoir jouer dessus avec la réponse OK. Dans le cas contraire, la réponse NOTFOUND sera renvoyée au client.

Cas d'utilisation : *Enregistrement des scores sur une fiche de jeu*

Contrôleurs utilisés ainsi que ses routes utiles au cas d'utilisation

SheetController **/sheet**			
TYPE	ENDPOINT	PARAMS/ BODY	UTILITÉ
Patch	/	BODY	Cette route a pour but d'abord de récupérer dans le corps de la requête un objet partiel (patron proxy) de la fiche de jeu sur laquelle la partie s'est jouée, cet objet servira à apporter des modification sur la fiche de jeu notamment en sauvegardant un meilleur joueur avec son score s'il y a lieu. Le middleware fera ainsi une mise à jour de la fiche associée sur la base de données et retourne une réponse OK en cas de succès et NOTFOUND en cas d'échec.

cas d'utilisation: *Envoie et réception des fichiers dans la vue de création*

Services client utilisés ainsi que ses routes utiles au cas d'utilisation :

ImageService **/image**			
TYPE	ENDPOINT	PARAMS / BODY	UTILITÉ
POST	/compare	BODY	Cette route accepte un objet sheet avec des images à comparer. Sa retourne une requête POST au serveur, qui permet d'envoyer l'objet FormData en tant que corps de la requête. En cas de succès, on a OK, sinon

SheetService **/sheet**			
TYPE	ENDPOINT	PARAMS / BODY	UTILITÉ
POST	/create	PARAMS	Cette route a pour but d'accepter un objet FormData comme parametre. Il envoie une requete HTTP POST au serveur, pour pouvoir

Cas d'utilisation : *Affichage des parties créées dans la vue de sélection*

Services client utilisés ainsi que ses routes utiles au cas d'utilisation :

SheetService **/sheet**			
TYPE	ENDPOINT	PARAMS / BODY	UTILITÉ
GET	/get	PARAMS	Cette route a pour but de retourner un observable d'un array d'objets Sheet. Il envoie une requete HTTP GET au serveur, et s'attend à recevoir un array d'objets sheet en reponse. Il inclus aussi un handler d'erreur qui retourne un array vide en cas d'erreur.

Cas d'utilisation : *Suppression d'une fiche de jeu dans la vue de création*

Service client utilisés ainsi que ses routes utiles au cas d'utilisation :

SheetService **/image**			
TYPE	ENDPOINT	PARAMS / BODY	UTILITÉ
DELETE	/id	PARAMS	Cette route a pour but d'accepter un paramètre id. Il supprime la fiche de jeu que l'on ne desire plus avoir. Il inclut aussi handler d'erreur qui utilise un opérateur catchError pour retourner une reponse vide en cas d'erreur.

Cas d'utilisation : *Récupération de la fiche de jeu ainsi que des images utilisées au lancement d'une partie*

Service client utilisés ainsi que ses routes utiles au cas d'utilisation :

SheetService **/sheet**			
TYPE	ENDPOINT	PARAMS / BODY	UTILITÉ
GET	/id	PARAMS	Cette route a pour but d'accepter un parametre id. Il envoie un GET au serveur. Ceci est pour avoir Il inclus aussi un handler d'erreur utilisant un opérateur catchError pour retounre un objet Sheet vide en cas d'erreur.

ImageService **/image**			
TYPE	ENDPOINT	PARAMS / BODY	UTILITÉ
GET	/	BODY	Cette route a pour but de faire un get au serveur avec le type de reponse 'blob' pour indiquer que le type de reponse devrait etre un objet binaire representant l'image. En d'autre terme, il prend du serveur le nom du fichier avec l'image du jeux qui est désiré.

Cas d'utilisation : *Enregistrement des scores sur une fiche de jeu*

Service client utilisés ainsi que ses routes utiles au cas d'utilisation :

SheetService **/sheet**			
TYPE	ENDPOINT	PARAMS / BODY	UTILITÉ
PATCH	/id	BODY	Cette route a pour but d'avoir le paramètre id du sheet on veut update, et un FormData avec le nom de parametre etant sheetForm, qui contient la donnee mise a jour. Il retounre un observable de type objet any. Elle enovie une requete HTTP PATCH au serveur avec le corps ayant le FormaData.

Avec la fin de la première remise, nous avons eu un changement d'idées, et avons décidé de changer complètement la manière de procéder avec la communication client-serveur. En effet, nous avons, pour les deux derniers sprints, décidé d'utiliser les web-sockets. Cette décision a été prise après avoir lu les requis du deuxième et du troisième sprint. Pour les paquets websockets, la structure est ainsi: le nom d'événement, sa source et son contenu (si lieu). En outre, il est aisé de constater que nous faisons usage de salles pour notre site web. Ces dernières nous permettent de regrouper des sockets du côté du serveur. Par la suite, le serveur communique avec un sous-ensemble de ses clients, comme par exemple notre s Un socket peut communiquer avec les autres de sa salle ○ Par défaut chaque socket a sa propre salle qui correspond à son id

3.2 Protocole WebSocket:

Pour assurer une émission/réception robuste d'événements du côté client , nous utilisons un service Socket Client Service

Ce service est injectable et permet la communication en temps réel avec un serveur en utilisant la librairie Socket.IO. Ce service peut être injecté dans n'importe quelle page ou composant dans le client qui nécessite une connexion au serveur.

Le service fournit une interface générique pour communiquer avec le serveur, ce qui nous permet de se concentrer sur la mise en œuvre des fonctionnalités dont ils ont besoin sans se soucier des détails de l'établissement d'une connexion avec le serveur. Le service dispose de plusieurs méthodes, notamment isSocketAlive(), qui vérifie si la connexion est active, connect(), qui établit une connexion avec le serveur, disconnect(), qui ferme la connexion avec le serveur, on<T>(event: string, action: (data: T) => void), qui enregistre un écouteur d'événement, et send<T>(event: string, data?: T), qui envoie un événement au serveur avec une charge utile de données facultative.

Étant donné l'injection de dépendance , toutes les composantes injectant ce service partageront le même socket et donc chacune peut recevoir tous les événements destinés aux salles de ce dernier ,chaque composante choisit de gérer les événements qui les intéressent.

Côté serveur, nous utilisons une implémentation de classe TypeScript d'un gateway WebSocket. Elle utilise la librairie Socket.IO pour gérer les messages WebSocket entrants et sortants.

La classe ChatGateway et GameGateway implémente les interfaces OnGatewayConnection, OnGatewayDisconnect et OnGatewayInit, ce qui lui permet de gérer les événements de connexion et de déconnexion WebSocket et d'initialiser le serveur WebSocket. Elle définit également plusieurs gestionnaires d'événements de messages à l'aide du décorateur @SubscribeMessage.

La classe ChatGateway est responsable de la création et de la gestion des salles de jeux, de la gestion de la logique de jeu, et de l'envoi et de la réception de messages entre les clients.

La classe ChatGateway agit en tant que médiateur entre les clients et la logique de jeu, et gère l'état du jeu et la communication entre les clients.

La classe GameGateway gère tout ce qui est en rapport avec le mode temps limité

Voici les paquets WebSocket de communication entre le client et le serveur.

Cas d'utilisation : Créer un jeu en Solo

NOM DU EVENT	BODY	UTILITÉ	Source/Destination
createSoloGame	RoomInfo {name,sheetId, roomName}	créer un jeu classique en Solo , envoyer le nom de la room qu'on veut avec l'id de la fiche et le nom du joueur	SocketClientService.socket ->WebSocketGateWay (chat.gateway.ts)
roomInfo	RoomInfo {... startTime, numberOfDifferences, gameMode,isGameDone }	Envoyer les informations de la partie à la page du jeu pour les utiliser dans l'affichage et les mettre à jour	WebSocketGateWay (chat.gateway.ts)-> SocketClientService.socket

Cas d'utilisation : Créer un jeu en Mode 1 contre 1.

NOM DU EVENT	BODY	UTILITÉ	Source/Destination
joinGridRoom	void	À l'initialisation du carrousel , celui-ci envoie un message au serveur pour le faire joindre la salle des carrousels.	SocketClientService.socket ->WebSocketGateWay (chat.gateway.ts)

gameJoinable	sheetId : string	Dès que le bouton “créer” est cliqué ,le carrousel envoie au serveur la fiche du jeu, le serveur crée une salle portant le nom `GameRoom${sheetId}` et ajoute le socket du client créateur dans cette salle où il peut recevoir , accepter ou refuser un client qui veut joindre la partie	SocketClientService.socket ->WebSocketGateWay (chat.gateway.ts)
Joinable	sheetId: string	le serveur renvoie à tous les clients, l’information qui leur permettra de changer les boutons “créer” en “joindre”	WebSocketGateWay (chat.gateway.ts)-> SocketClientService.socket room:GridRoom (BroadCast)
joinGame	JoinGame {name, sheetId}	rentrer dans la liste d’attente des joueurs qui veulent joindre la partie	SocketClientService.socket ->WebSocketGateWay (chat.gateway.ts)
UserJoined	playerName : string	le carrousel utilise son dialog service pour afficher les noms des joueurs qui demandent de joindre la partie dans une file d’attente.	WebSocketGateWay (chat.gateway.ts)-> SocketClientService.socket room: (BroadCast)`GameRoom${sheetId}`
playerRejected	Reject {playerName, sheetId}	Un client créateur rejette une demande de joindre, le serveur reçoit le nom du joueur à faire sortir de la salle correspondante à la fiche de jeu.	SocketClientService.socket ->WebSocketGateWay (chat.gateway.ts)
playerConfirmed	Confirm {player1:string, player2:string,sheetId}	Un client créateur accepte une demande de joindre, envoie au serveur les noms des deux joueurs et la fiche du jeu , le serveur crée une instance de l’interface PlayRoom avec les noms des joueurs	

MultiRoom Created	OpponentInfo {player2:string, roomName}	Le client reçoit cet event et vérifie si il est correspondant au client accepté et renvoie un signal avec le même body au serveur , dans le cas contraire le client envoie un évènement "***quit***" pour quitter la salle	WebSocketGateWay (chat.gateway.ts)-> SocketClientService.socket room: (BroadCast)`GameRoom${sheetId}`
Player2Joined	OpponentInfo	Le serveur reçoit les information du deuxième joueur et ajoute son socket à la salle créée pour le jeu et son nom	SocketClientService.socket (player2) ->WebSocketGateWay (chat.gateway.ts)
quit	sheetId: string	Le serveur reçoit les informations de tous les autres clients qui n'ont pas été acceptés et les fait sortir de la salle de liste d'attente.	
JoinedRoom	PlayRoom (interface)	les deux joueurs reçoivent les informations finales de la room pour pouvoir les utiliser en affichage et mises à jour	WebSocketGateWay (chat.gateway.ts)-> SocketClientService.socket room: (BroadCast)`GameRoom${sheetId}`

Scénario Alternatifs dans le même cas

- À tout moment , le client créateur peut annuler le jeu, le serveur reçoit son signal et le renvoie à la salle des carrousels qui vont remettre le bouton ***joindre*** à ***créer***
- À tout moment , un client dans une liste d'attente d'un jeu peut annuler sa demande , il envoie son signal au serveur qui le transmet à son tour au client créateur et il sera retiré de la liste d'attente.

Cas d'utilisation : Jouer une partie.

Comme mentionné dans les deux cas précédents , à l'initialisation de la Game-Page Component , celle-ci reçoit du serveur les noms des joueurs et le nombre des différences totales pour les utiliser dans l'affichage.
La Play-Area Component qui fait partie de la game-page, injecte un service Game-Logic Service qui est responsable de la communication avec le serveur pour tout ce qui est validation de clique et

NOM DU EVENT	BODY	UTILITÉ	Source/Destination

click	clickData{x:,y: ,roomName: playerName, };	envoyer les informations du click au serveur	SocketClientService.socket ->WebSocketGateWay (chat.gateway.ts)
clickFeedBack	clickFB { coord, isError }	traiter les informations du click et envoyer un feedback	WebSocketGateWay (chat.gateway.ts) -> SocketClientService.socket
gameDone	message : string	signaler une fin de partie avec un message dépendant de la façon de fin.	WebSocketGateWay (chat.gateway.ts) -> SocketClientService.socket
clock	Date()	cet event est émis chaque seconde, la game-page component le reçoit et sourtrait le temps du début de la partie , pour avancer son ***timer***	WebSocketGateWay (chat.gateway.ts) -> SocketClientService.socket

Scénario Alternatif : en Mode 1v1 , à tout moment un joueur peut quitter la partie , la méthode handleDisconnect : cherche la salle de du socket de ce joueur dans son tableau des PlayRooms et fait sortir le joueur de la salle , si le jeu est terminé , le serveur supprime la salle de jeu, sinon , le serveur émet un message au joueur restant que son adversaire a quitté et qu'il a gagné.

Cas d'utilisation : Clavardage et messages de parties.

NOM DU EVENT	BODY	UTILITÉ	Source/Destination

roomMessage	ChatMessage : { content: string, type: string name? time?}	envoyer un message avec ses informations	SocketClientService.socket ->WebSocketGateWay (chat.gateway.ts)
		recevoir des messages de partie locaux ou globaux ou clavardage	WebSocketGateWay (chat.gateway.ts) -> SocketClientService.socket

Cas d'utilisation : Jeu en Mode Temps Limité

NOM DU EVENT	BODY	UTILITÉ	Source/Destination

createLimitedTimeSolo	RoomData:
	
	{
	
	player: Player (nom et socketID),
	
	timeBonus,
	
	timeLimit,
	
	hintsLeft?,
	
	};
		Créer une partie solo en temps limité.
		
		(conserver toutes les informations de la partie dans le tableau des rooms de game.gateway)
			SocketClientService.socket ->WebSocketGateWay (game.gateway.ts)

limitedTimeRoomCreated	RoomAssets { room (interface de jeu temps limité), left : buffer , right : buffer ,}	le serveur renvoie les informations du jeu créé au client avec les buffers à dessiner sur les canvas dès que la partie est déclenchée	WebSocketGateWay (game.gateway.ts)-> SocketClientService.socket
createLimitedTimeCoop	RoomData	Créer une partie coopérative en temps limité. (conserver toutes les informations de la partie dans le tableau des rooms de game.gateway)	SocketClientService.socket ->WebSocketGateWay (game.gateway.ts)
seconPlayerJoined	RoomAssets	le serveur renvoie les informations du jeu créé au client avec les buffers à dessiner sur les canvas dès que la partie est déclenchée	WebSocketGateWay (game.gateway.ts)-> SocketClientService.socket
sheetDeleted	id : string	mettre à jour les fiches de jeu pour une partie en cours	SocketClientService.socket ->WebSocketGateWay (game.gateway.ts)
sheetCreated	id : string	mettre à jour les fiches de jeu pour une partie en cours	SocketClientService.socket ->WebSocketGateWay (game.gateway.ts)
cancelGame	Vide	Annuler la tentative de joindre une partie temps limité en mode coopérative	SocketClientService.socket ->WebSocketGateWay (game.gateway.ts)
timeout	{roomName:string, player:Player, allyGaveUp?: boolean}	Informer le serveur que le temps est écoulé avant que toutes les fiches de jeux aient été défilées, le boolean sert à déterminer comment on va conserver l'historique de la partie.	SocketClientService.socket ->WebSocketGateWay (game.gateway.ts)

GameOver	message: string	Le serveur envoi à la room concernée un message de fin de partie selon la façon dans laquelle la partie est finie	WebSocketGateWay (game.gateway.ts)-> SocketClientService.socket
clickTl	{ target , Coord (x,y)}	Informer le serveur d'un click sur le canvas	SocketClientService.socket ->WebSocketGateWay (game.gateway.ts)
clickValidated	RoomAssets { … , click : coord }	changer les buffers pour afficher la nouvelle fiche de jeu en cas ou une différence est trouvée dans la présente fiche	WebSocketGateWay (game.gateway.ts)-> SocketClientService.socket
hint	Vide	Informer le serveur que le client veut un indice, on a besoin juste du id du socket pour renvoyer un message de partie (roomMessage)	SocketClientService.socket ->WebSocketGateWay (game.gateway.ts)
playerLeft	Vide	Informer le joueur que sont allié a abandonné et basculer le jeu en mode solo	WebSocketGateWay (game.gateway.ts)-> SocketClientService.socket