108

Protocole de communication

Version 2.0

Historique des révisions

Date	Version	Description	Auteur
2023-03-21	1.0	Fait le jour de la remise du sprint2	Toute l'équipe
2023-04-20	2.0	Fait le jour de la remise du sprint3	Toute l'équipe

Table des matières

Protocole de communication

1. Introduction

Ce document fera office de présentation quant à la communication client-serveur et décrira les différents paquets présents dans l'ensemble de notre projet. Il sera présenté en deux parties distinctes. De manière générale, la première partie portant sur la communication client-serveur se chargera d'expliquer notre moyen de communication: les sockets. De plus, les utilités de tous nos protocoles de communication seront abordées. En ce qui concerne la seconde partie de ce document, celle-ci porte sur la description des paquets, qui sont en fait des morceaux d'informations communiqués au sein d'un réseau. Cette partie servira principalement à décrire le contenu des différents types de paquets utilisés tout au cours de nos protocoles, ainsi que de l'information utile supplémentaire par rapport aux sprints 2 et 3.

2. Communication client-serveur

Pour établir la communication client-serveur nous avons utilisé la librairie Socket.io et des requêtes HTTP. D'un côté, Socket.io, basée sur le protocole WebSocket, nous a permis d'établir une connexion bidirectionnelle entre le client et le serveur, qui à son tour s'occupe de mettre à jour le client dynamiquement, ainsi que d'établir la connexion entre les joueurs en mode multijoueur. En effet, un exemple qui illustre l'application pratique de ce type de communication est la mise à jour de fiches de jeu dynamiquement dans la vue de sélection de jeu lorsqu'une fiche est créée ou supprimée. De plus, nous utilisons les sockets pour créer des salles de jeu pour deux joueurs et pour permettre l'envoi de messages aux joueurs.

De manière générale, les clients se connectent au serveur à l'aide de la librairie Socket.io, puis lorsque les données sont mis à jour du côté du serveur, ce dernier notifie tous les clients qui ont préalablement établi une connexion. Comme mentionné ci-haut, grâce à cette connexion bidirectionnelle, le client est mis à jour dynamiquement. En fait, lorsqu'un changement se produit sur le serveur, le client est aussitôt avisé qu'il doit effectuer un nouveau téléchargement. Par exemple, lorsqu'une modification se produit sur l'ensemble des fiches de jeu (fiche supprimée, meilleur temps modifié, etc.), le serveur notifie tous les clients de ce changement. De cette manière, le client est mis au courant et il peut procéder à télécharger les fiches de jeu à nouveau. Une telle approche ne serait pas possible avec des requêtes HTTP, car ce protocole ne permet pas au serveur d'initier une requête au client.

De plus, les sockets permettent de gérer très efficacement la connexion et la communication des joueurs en mode multijoueur (un contre un, ou temps limité) en utilisant les salles de Socket.io. Lorsqu'un joueur se connecte en mode multijoueur, le serveur vérifie si le jeu de certain type existe tel qu'il y a un seul joueur dans la salle, le cas échéant le requérant est connecté dans la salle, sinon il crée une nouvelle salle. Suite à ce traitement du côté serveur, le client est notifié, puis il peut faire d'autres requêtes qui lui permettront d'entamer la partie. Durant la partie, les joueurs reçoivent des messages, que ce soit des messages globaux ou des messages envoyés par les joueurs de la partie. Dans tous les cas, les joueurs reçoivent les messages au moyen de sockets, car ils leurs sont envoyés par le serveur. Sans les sockets et leur connexion bidirectionnelle, il ne serait pas possible aux clients de recevoir des messages à travers des requêtes HTTP (du moins ce serait très peu efficace).

D'un autre côté, la communication HTTP est utilisée pour charger les fiches initialement dans la vue de sélection de jeu, elle permet aussi de charger les nouveaux jeux créés dans le système. La suppression de jeux du système se fait aussi par HTTP. Nous avons utilisé ce protocole surtout lorsque le client avait besoin de télécharger ou de modifier des données du serveur, puis agir en conséquence de la réponse HTTP. Dans ce contexte, le client reçoit une réponse suite à l'envoi d'une requête, il n'y a pas de communication bidirectionnelle. Par exemple, lorsqu'une fiche est créée, le client envoie une requête HTTP avec la fiche nouvellement créée. Cette requête contient la fiche de jeu dans le corps de la requête et elle est envoyée avec la méthode POST, car il s'agit de création d'une nouvelle ressource. On en déduit que l'utilisation du protocole HTTP dans un tel cas est favorable aux sockets, car elle requiert un simple envoie et une réponse du serveur pour valider l'opération. De plus, la requête HTTP permet de rendre le code plus compréhensible et plus sécuritaire en appliquant les principes REST.

En somme, dans la majorité des cas où le serveur avait besoin d'envoyer des données aux clients sans attendre une requête de leur part, nous avons favorisé l'utilisation des sockets, car cette technique n'est pas possible avec HTTP.

3. Description des paquets

HTTP :

	Requête				Réponse	
Nom de la méthode dans le code	Méthode HTTP	Chemin	Corps	Query	Codes de retour	Contenu
uploadImage	POST	api/fs/image	image sous forme de chaîne en base64	-	201, 400	id de la nouvelle image créée sur le serveur
downloadImage	GET	api/fs/image	-	L'id de l'image désirée	200, 204	id de l'image téléchargée
uploadGameCard	POST	api/fs/gameCard	fiche de jeu désirée	-	201, 400, 500	id de la nouvelle fiche de jeu créée sur le serveur
downloadGameCards	GET	api/fs/gameCards	-	-	200, 204	toutes les fiches de jeu téléchargées sous forme de json
deleteGameCard	DELETE	api/fs/gameCard	-	L'id de la fiche de jeu désirée	200, 204	-
deleteAllCards	DELETE	api/fs/gameCards	-	-	200, 500	-
addGameToHistory	POST	api/fs/gameEnded	l'objet de la partie terminée	-	201, 400, 500	-
getHistory	GET	api/fs/history	-	-	200, 500	historique sous forme de json
deleteHistory	DELETE	api/fs/history	-	-	200, 500	-
setNewTime	POST	api/fs/newTime	nouveau meilleur temps	-	200, 201, 400, 500	-
getBestTimes	GET	api/fs/bestTimes	-	L'id de la fiche de jeu	200, 400, 500	meilleurs temps sous forme de json
resetBestTimes	DELETE	api/fs/bestTimes	-	L'id de la fiche de jeu	200, 400, 500	-

resetAllBestTimes	DELETE	api/fs/bestTimes/all	-	-	200, 500	-
getConstants	GET	api/fs/constants	-	-	200, 500	constantes sous forme de json
setConstants	POST	api/fs/constants	nouvelles constantes	-	201, 400, 500	-

Sockets:

Nom d'événement	Source	Contenu	Tâche
gameCardStatus	Serveur	L'id de la fiche et deux boolean qui définissent si le jeu est en attente	Met à jour le statut du bouton joindre/créer pour une fiche donnée
gameFull	Serveur	L'id de la fiche de jeu	Notifie le client que la partie est pleine
createdNewRoom	Serveur	L'id de la fiche de jeu	Notifie le client qu'une nouvelle salle a été créée
startGame	Serveur	L'id de la fiche de jeu	Avise le client que la partie est commencée
abortGame	Serveur	L'id de la fiche de jeu et le message justificatif	Avise le client que la partie est interrompue
orderSent	Serveur	L'ordre des fiches pour le jeu	Transmet l'ordre des fiches au client
startGameCoop	Serveur	L'id de la fiche de jeu, le nom du joueur 1, le nom du joueur 2 et l'ordre des fiches	Invite le client à commencer la partie en temps limité en mode coop
playerLeft	Serveur	-	Aviser le joueur que l'adversaire a quitté la partie
gameCardsModified	Serveur	-	Avise le joueur que les fiches de jeu ont été mis à jour
newPlayer	Serveur	Le nom du deuxième joueur	Aviser le client en attente qu'un joueur a rejoint sa partie
validation	Serveur	Résultat de la validation du click	Arrêter le troisième indice
leader	Serveur	-	Avise le client qu'il est le créateur de la partie

diffFound	Serveur	Coordonnée du click et un booléen qui indique si la différence est trouvée par l'adversaire	Met à jour les compteurs de différences
End	Serveur	Le nom du gagnant	Avise le client que son jeu est terminé
otherPlayerQuit	Serveur	-	Avise le client que son adversaire a quitté la partie
newMessage	Serveur	Le message reçu	Reçoit le message
askGameCardStatus	Client	L'id de la fiche de jeu	Sert d'une requête pour demander au serveur le statut des fiches de jeu
joinGameClassic1v1	Client	L'id de la fiche de jeu et le nom de joueur du client	Demande de rejoindre une partie du jeu spécifié en mode classique en un contre un
joinGameSolo	Client	L'id de la fiche de jeu	Demande de rejoindre une partie du jeu spécifié en mode classique en solo
leaveGame	Client	-	Quitter la partie en cours
bestTimesUpdate	Client	-	Aviser les autres que les meilleurs ont été mis à jour
gameCardDeleted	Client	L'id de la fiche de jeu	Aviser les autres que la fiche de jeu spécifiée a été supprimée
gameCardsDeleted	Client	-	Aviser les autres que les fiches de jeu ont été supprimée
startGameSolo	Client	Le nom de joueur du client	Demande de commencer une partie du jeu spécifié en temps limité en solo
startGame	Client	Le nom du jeu	Commencer la partie en attente
joinGameCoop	Client	Le nom de joueur du client	Demande de commencer une partie du jeu spécifié en temps limité en coop
rejectPlayer	Client	-	Rejeter le joueur nouvellement connecté
localMessage	Client	Le message envoyé par le client	Envoie le message local du client
gameEnded	Client	Un booléen qui indique si le jeu a été abandonné	Signaler la fin du jeu
click	Client	La position du click et le tableau des différences	Envoie le click pour validation