204

Protocole de communication

Version 1.6

Historique des révisions

Date	Version	Description	Auteur
2023-03-19	1.1	Complétion de la partie communication client-serveur	Équipe 204
2023-03-20	1.2	Complétion de la partie description des paquets	Équipe 204
2023-03-21	1.3	Complétion de l'introduction	Équipe 204
2023-04-16	1.4	Modification de la partie communication client-serveur	Équipe 204
2023-04-17	1.5	Modification de la partie description des paquets et ajout de l'annexe	Équipe 204
2023-04-18	1.6	Modification de l'introduction	Équipe 204

Table des matières

Protocole de communication

1. Introduction

Ce document décrit nos décisions de protocole de communication du projet en fournissant différentes explications sur la communication client-serveur et les paquets utilisés pour faciliter la compréhension et la gestion du système de communication. On commence par la communication client-serveur des fonctionnalités déjà complétées des Sprints 1 et 2 avec une justification de l'utilisation des protocoles HTTP et WebSocket. Ensuite, nous détaillons les rôles futurs de ces protocoles pour l'implémentation des fonctionnalités du Sprint 3. Dernièrement, on présente la description des paquets liste toutes les méthodes HTTP implémentées avec une introduction pour chacune ainsi que tous les évènements WebSocket et leur contenu. Cela permet de justifier leur utilisation et de présenter notre logique de réalisation de la communication client-serveur.

2. Communication client-serveur

Le protocole HTTP a été introduit directement dans le sprint 1 avec l'implémentation des vues de sélection de partie et de configuration. En effet, ces pages nécessitaient la visualisation de fiches de jeu que nous avons rendu accessibles à partir du serveur. Le client envoie donc une requête au serveur pour accéder aux données de chaque fiche et renvoyer à l'affichage le nom du jeu, son image originale ainsi que le niveau de difficulté et les meilleurs scores atteints (statiques et non implémentés pour ce sprint). Pour implémenter la vue de jeu en solo, le même protocole est utilisé pour accéder en plus à la matrice des différences. On gère d'ailleurs le mode classique en solo avec le protocole WebSocket qui permet la synchronisation de la minuterie entre le client et le serveur ainsi que la revalidation des différences de la part du serveur. HTTP est donc assez adapté pour nous car il possède une certaine implémentation native des différentes actions que l'on souhaite effectuer avec les fiches de jeu. Le protocole permet de fournir un objet intégrant les données nécessaires de chaque jeu, grâce à une requête du client, il n'est pas nécessaire d'attendre une initiative du serveur.

Pour le sprint 2, nous avons continué d'utiliser le protocole HTTP pour les mêmes raisons qu'au sprint 1. D'ailleurs, une nouvelle fonctionnalité HTTP est la suppression de jeu. Le protocole WebSocket a été poussé avec l'implémentation du jeu en mode un contre un et la section des messages. En effet, WebSocket est utile pour tout ce qui nécessite la liaison de deux clients entre eux, utilisant le serveur comme intermédiaire. Cette situation se retrouve dans le mode classique un contre un puisqu'il est possible d'effectuer une partie en multijoueur permettant à deux joueurs de compétitionner. Concrètement, la logique du mode un contre un est similaire à celle du mode solo, mais avec l'ajout de la synchronisation des données entre les deux joueurs. Le protocole WebSocket permet de créer et de rejoindre une partie multijoueur, de synchroniser la minuterie entre les joueurs et le serveur, ainsi que de revalider les différences trouvées par les joueurs en temps réel. De plus, grâce à WebSocket, le serveur peut envoyer des informations aux clients sans qu'ils aient besoin de lui envoyer une requête, ce qui est un avantage considérable en termes de performances. En utilisant WebSocket pour le jeu en mode un contre un, nous assurons une expérience fluide et réactive aux joueurs, même si leur connexion Internet est moins rapide. En effet, le protocole WebSocket permet de minimiser la latence et le nombre de requêtes entre le client et le serveur, ce qui garantit une expérience de jeu en temps réel sans ralentissements ni décalages. Du côté de la section message, nous utilisons WebSocket pour gérer le clavardage entre les joueurs, mais aussi les messages provenant du système. Au final, WebSocket est donc bien adapté pour nous car les sockets sont nécessaires pour permettre au client d'effectuer des requêtes au serveur et vice-versa. Le serveur est donc réactif aux requêtes du client et celui-ci est aussi réactif aux données envoyées par le serveur.

Pour le sprint 3, nous utiliserons toujours HTTP et WebSocket. Les fonctionnalités de la page de configuration concernant les données des constantes de jeu, de l'historique de parties et des meilleurs scores utiliseront HTTP qui sera très utile pour leur réinitialisation. Ce protocole sera encore utilisé afin d'accéder aux données de la page de jeu, il sera d'ailleurs beaucoup plus sollicité car nous devons accéder à une nouvelle fiche de jeu à chaque différence trouvée. WebSocket sera quant à lui utilisé pour les nouvelles fonctionnalités de section de message. En effet, nous devrons notifier les joueurs en cours de partie des nouveaux meilleurs scores obtenus. De plus, la fonctionnalité d'indice envoie une requête au serveur afin que celui-ci envoie en réponse une pénalité de temps et un message de mise à jour d'indice dans le clavardage.

3. Description des paquets

La communication client-serveur avec le protocole HTTP se fait via diverses routes, chaque route possède cependant le préfixe /api. Les routes contenant un nom précédé par un ‘:’ (*ex: :name*), indique un paramètre à ajouter au chemin pour changer le comportement de la route. “Corps de requête” désigne les données envoyées avec la requête, “Aucun” n'indique aucune donnée nécessaire pour cette route. La réponse est l’objet de réponse s'il y a lieu.

Certaines données de requête ou de réponse sont soulignées, cela indique que ce sont des objets décrits en annexe.

Méthode	**Route**	**Corps de requête**	**Description**	**Réponse**	**HTTP code**	**HTTP erreur code**
GET	/game/	Aucun	Renvoie tous les jeux	GameData[]	200	500
GET	/game /:name	Aucun	Renvoie le jeu et ses données	GameData	200	404
POST	/game/	NewGame	Créée un nouveau jeu	Aucun	201	500
DELETE	/game /:name	Aucun	Supprime le jeu	Aucun	200	404
DELETE	/game/	Aucun	Supprime tous les jeux	Aucun	200	500

Tableau 1: Tableau des méthodes de Game Contrôleur

Méthode	**Route**	**Corps de requête**	**Description**	**Réponse**	**HTTP code**	**HTTP erreur code**
GET	/config/history	Aucun	Renvoie l’historique des parties	GameHistory[]	200	500
GET	/config/constants	Aucun	Renvoie les constantes de jeu	Constants	200	500
GET	/config/times/:name	Aucun	Renvoie les meilleurs temps du jeu	BestTime[], BestTime[]	200	404
PUT	/config/times/:name	NewBestTime	Change les meilleurs temps du jeu et renvoie la position dans le classement	Position	201	404
PUT	/config/constants	GameConstants	Modifie les constantes des jeux	Aucun	204	500
DELETE	/config/history	Aucun	Supprime l’historique des parties	Aucun	200	500
DELETE	/config/times	Aucun	Supprime tous les meilleurs temps	Aucun	200	500
DELETE	/config/times/:name	Aucun	Supprime les meilleurs temps du jeu	Aucun	200	404

Tableau 2: Tableau des méthodes de Config Contrôleur

Le deuxième protocole de communication étant Web Socket (Socket IO), nous avons créé des Gateway comme moyen de communiquer avec le client. GameMode, GameFinder, WaitingRoom et Chat sont des Gateway qui servent le client de différentes routes WS décrites ci-dessous. Lorsque la route indique une destination tel que "Room", c'est dans le cas d'une partie en cours ou en attente, chaque partie étant dans une room spécifique que ce soit pour le mode multijoueur ou non pour que le serveur puisse communiquer directement avec tous les joueurs de la partie en même temps. Certains évènements sont bidirectionnels, un client peut l'envoyer au serveur, et le serveur peut renvoyer le même évènement, c'est le cas pour "abandoned" et "canJoinGame".

Nom d'évènement	Source	Contenu	Description
validate	Client	DifferencePos, roomId, username	Demande confirmation du serveur pour une tentative de clic
validated	Serveur	Validated *(bool)*, differencePos, username	Informe les joueurs si *username* à trouver une différence (*validated* à true)
endGame	Client	EndGame	Demande la terminaison de la partie
gameFinished	Serveur	Aucun	Informe les joueurs que la partie est terminée
abandoned	Client	Username	Demande au serveur d'informer les joueurs que *username* abandonne le jeu
abandoned	Serveur	GameRoom, username	Informe les joueurs que *username* abandonne la partie
GameDeleted	Serveur	GameName, gameMode	Indique à **tous** les joueurs que la partie *gameName* est supprimée
gameCanceled	Serveur	GameName	Indique à tous les joueurs que la partie *GameName* est annulée
timer	Serveur	Timer	Envoie à chaque partie, son timer incrémenté si la partie est en cours
changeTime	Client	RoomId, time	Incrémente ou décremente le timer de la room spécifié
nextGame	Serveur	GameRoom	Envoie le prochain jeu dans le cas du mode temps limité

Tableau 3: Tableau des événements de GameMode Gateway

Nom d'évènement	Source	Contenu	Description
checkGame	Client	GameName, gameMode	Demande au serveur si une partie est en attente pour le jeu *gameName* en *gameMode*
gameFound	Serveur	GameName, gameMode	Informe le client qu'une partie en attente a été trouvé pour ce jeu et mode de jeu
canJoinGame	Client	GameName, username	Demande si *username* peut rejoindre la partie *gameName*
canJoinGame	Serveur	Aucun	Indique au client qu'il peut rejoindre la partie
cannotJoinGame	Serveur	Aucun	Indique au client qu'il ne peut pas rejoindre la partie

Tableau 4: Tableau des méthodes de GameFinder Gateway

Nom d'évènement	Source	Contenu	Description
acceptPlayer	Client	RoomId, username	Prévient le serveur que *username* a été accepté à jouer dans la partie *RoomId*
rejectPlayer	Client	RoomId. username	Prévient le serveur que *username* a été rejeté de la partie en attente *roomId*
abortGameCreation	Client	RoomId	Met fin à la partie en attente ayant l'identifiant *roomId*
leaveGame	Client	RoomId, username	Prévient le serveur que le joueur *username* a quitté la partie en attente
start	Client	RoomId	Demande au serveur de lancer la partie
started	Serveur	Aucun	Informe les joueurs que la partie démarre
createGame	Client	GameRoom	Créer une nouvelle partie avec les informations de *GameRoom*
askingToJoinGame	Client	GameName, username, gameMode	Demande les informations de la partie pour la rejoindre
gameInfo	Serveur	GameRoom	Envoie les informations de la partie (et surtout les joueurs présents ou en attente) a **tous** les joueurs
gameCreated	Serveur	RoomId	Indique aux joueurs que la partie est initialisée
playerRejected	Serveur	GameRoom	Prévient les utilisateurs de la partie *GameRoom* que la liste des joueurs a changé (un joueur n'est plus dans PotentialPlayers)
playerAccepted	Serveur	GameRoom	Prévient les utilisateurs de la partie *GameRoom* que la liste des joueurs a changé (un joueur est en plus dans PotentialPlayers)
gameCanceled	Serveur	GameName	Indique à tous les joueurs que la partie *GameName* est annulée
gameFound	Serveur	GameName, gameMode	Informe le client que *gameName* existe bien
GameDeleted	Serveur	GameName, gameMode	Indique à **tous** les joueurs que la partie *gameName* est supprimée

Tableau 5: Tableau des méthodes de WaitingRoom Gateway

Nom d'évènement	Source	Contenu	Description
sendMessage	Client	Message, Username, roomId	Demande au serveur d'envoyer *Message* à la room ayant l'Id *roomId*
message	Serveur	Message, Username	Envoie à la room ou à **tous** les joueurs le message et le username du correspondant s'il y a lieu

Tableau 6: Tableau des méthodes de Chat Gateway

4. Annexe

Voici les différents types d'objet transités dans les communications HTTP/Web Socket.

GameData:

- **Name**: nom du jeu
- **NbDifference**: nombre de différence à trouver pour ce jeu
- **Image1url**: L'URL de l'image originelle
- **Image2url**: L'URL de l'image modifiée
- **Difficulty**: niveau de difficulté
- **SoloBestTimes**: tableau de BestTimes contenant les meilleurs temps solos
- **vsBestTimes**: tableau de BestTimes contenant les meilleurs temps multijoueur
- **DifferenceMatrix**: matrice donnant la position des différences

NewGame:

- **Name:** nom du jeu à créer
- **Image1:** contient l'image originelle en base64
- **Image2:** contient l'image modifiée en base64
- **NbDifference:** nombre de différences
- **DifferenceMatrix:** matrice donnant la position des différences
- **Difficulty**: niveau de difficulté

GameHistory:

- **Name**: nom du jeu
- **startTime**: date et heure du début de la partie
- **timer**: durée du jeu
- **GameMode**: mode de jeu de la partie
- **Username**: nom du joueur 1
- **Username2**: nom du joueur 2 (si multijoueur)
- **Abandonned**: tableau des joueurs ayant abandonné
- **Winner**: gagnant de la partie

NewBestTimes:

- **GameName:** nom du jeu
- **IsSolo**: booléen, partie solo ou multijoueur
- **Name**: nom du joueur
- **Time**: meilleur temps effectué

BestTime:

- **Name:** nom du jeu
- **Time:** meilleur temps

GameConstants:

- **initialTime:** compte à rebours initial du mode temps limité
- **PenaltyTime:** temps de pénalité appliqué en cas d'indice utilisé
- **bonusTime:** temps supplémentaire donné en mode temps limité

EndGame:

- **Winner:** booléen, si gagnant ou non
- **RoomId:** ID de la partie
- **Username**: nom du joueur
- **GameFinished**: booléen, si partie finie ou non

GameRoom:

- **UserGame:** contient l'objet UserGame
- **RoomId:** ID de la partie
- **Started:** booléen, si partie commencée ou non
- **GameMode:** mode de jeu (classique ou temps-limité)

UserGame:

- **Username**: nom du joueur 1
- **Username2**: nom du joueur 2 (si multijoueur)
- **PotentielPlayers**: liste des joueurs essayant de rejoindre la partie
- **GameData**: Contient l'objet GameData
- **NbDifferenceFound**: nombre de différences trouvées
- **Timer**: temps de la partie