# 204

# Protocole de communication

**Version 1.3**

# 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 |

# 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 à venir du Sprint 3. Dernièrement, 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 (get, post, patch, delete, etc.). On voit par exemple que la fonctionnalité de création de jeu utilise la méthode post. 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 qui utilise la méthode delete. 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. On gère le mode classique un contre un de la même façon que pour le mode classique en solo. Cependant, nous ajoutons la synchronisation des données non seulement entre le joueur et le serveur, mais aussi entre le joueur et l'opposant. De plus, grâce au protocole, on implémente une logique pour créer et joindre une partie multijoueur. Du côté de la section message, nous utilisons WebSocket pour gérer le clavardage entre les joueurs, mais aussi les messages provenant du système. Le fait que le protocole permette au serveur d'envoyer des informations au(x) client(s) sans nécessiter une requête de ce(s) dernier(s) est un gros avantage au niveau de l'efficacité. WebSocket est donc 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, par exemple avec les méthodes patch et delete. 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 plus loin dans le document.

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

*Tableau 1: Tableau des méthodes de Game Contrôleur*

| Méthode | Route | Corps de requête | Description | Réponse | HTTP code |
|---|---|---|---|---|---|
| GET | /config/history | Aucun | Renvoie l’historique des parties | GameHistory[] | 200 |
| GET | /config/constants | Aucun | Renvoie les constantes de jeu | Constants | 200 |
| PUT | /config/times/:name | BestTimes | Change les meilleurs temps du jeu | BestTimes | 200 |
| PUT | /config/constants | Constants | Modifie les constantes des jeux | Constants | 200 |
| DELETE | /config/history | Aucun | Supprime l’historique des parties | Aucun | 200 |
| DELETE | /config/times | Aucun | Supprime tous les meilleurs temps | Aucun | 200 |
| DELETE | /config/times/:name | Aucun | Supprime les meilleurs temps du jeu | Aucun | 200 |

*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. ClassicMode 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 | Destination | Contenu | Description |
|---|---|---|---|---|
| start | Client | Serveur | roomId | Demande au serveur de lancer la partie |
| started | Serveur | Room | Aucun | Informe les joueurs que la partie démarre |
| validate | Client | Serveur | DifferencePos, roomId, username | Demande confirmation du serveur pour une tentative de clic |
| validated | Serveur | Room | Validated *(bool)*, differencePos, username | Informe les joueurs si *username* a trouver une différence (*validated* à true) |
| endGame | Client | Serveur | RoomId, username | Demande la terminaison de la partie |
| gameFinished | Serveur | Room | Aucun | Informe les joueurs que la partie est terminée |
| abandoned | Client | Serveur | RoomId, username | Demande au serveur d'informer les joueurs que *username* abandonne le jeu |
| abandoned | Serveur | Room | username | Informe les joueurs que *username* abandonne la partie |
| checkGame | Client | Serveur | gameName | Demande au serveur confirmation que *gameName* existe |
| gameFound | Serveur | Client | gameName | Informe le client que *gameName* existe bien |
| createGame | Client | Serveur | gameRoom | Créer une nouvelle partie avec les informations de *GameRoom* |
| gameCreated | Serveur | Room | gameRoom | Indique aux joueurs que la partie est initialisée |
| GameDeleted | Serveur | Broadcast | gameName | Indique à tous les joueurs que la partie *gameName* est supprimée |
| canJoinGame | Client | Serveur | GameName, username | Demande si *username* peut rejoindre la partie *gameName* |
| canJoinGame | Serveur | Client | Aucun | Indique au client qu'il peut rejoindre la partie |
| cannotJoinGame | Serveur | Client | Aucun | Indique au client qu'il ne peut pas rejoindre la partie |

| | | | | |
|---|---|---|---|---|
| askingToJoinGame | Client | Serveur | GameName, username | Demande les informations de la partie pour la rejoindre |
| gameInfo | Serveur | Broadcast | GameRoom | Envoie les informations de la partie (et surtout les joueurs présents ou en attente) |
| abortGameCreation | Client | Serveur | Aucun | Met fin à la partie en attente dont le client attendait le début |
| gameCanceled | Serveur | Broadcast | GameName | Indique aux clients que la partie est annulée |
| leaveGame | Client | Serveur | RoomId, username | Prévient le serveur que le joueur *username* a quitté la partie |
| playerRejected | Serveur | Room | GameRoom | Prévient les utilisateurs de la partie *GameRoom* que la liste des joueurs a changé (un joueur n'est plus dans PotentialPlayers) |
| playerAccepted | Serveur | Room | GameRoom | Prévient les utilisateurs de la partie *GameRoom* que la liste des joueurs a changé (un joueur est en plus dans PotentialPlayers) |
| acceptPlayer | Client | Serveur | RoomId, username | Prévient le serveur que *username* a été accepté à jouer dans la partie *RoomId* |
| rejectPlayer | Client | Serveur | RoomId. username | Prévient le serveur que *username* a été rejeté de la partie en attente *roomId* |
| timer | Serveur | Room | timer | Envoie à chaque partie, son timer incrémenté si la partie est en cours |
| nextGame | Serveur | Room | GameRoom | Envoie le prochain jeu dans le cas du mode temps limité |

*Tableau 3: Tableau des événements de ClassicMode Gateway*

| Nom d'évènement | Source | Destination | Contenu | Description |
|---|---|---|---|---|
| sendMessage | Client | Serveur | Message, Username, roomId | Demande au serveur d'envoyer *Message* à la room ayant l'Id *roomId* |
| message | Serveur | Room | Message, Username | Envoie à la room le message et le username du correspondant |

*Tableau 4: Tableau des méthodes de Chat Gateway*

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

**GameForm:**

GameForm est un objet qui contient le moins d'informations possible puisqu'il est envoyé en tableau lorsque tous les jeux sont envoyés au client, donc on y retrouve seulement le nécessaire à l'affichage de sélection/configuration.

Contient:

- **Name**: Est une chaine de charactères pour indiquer le nom du jeu
- **NbDifference**: Est le nombre de différence à trouver pour ce jeu
- **Image1url**: L'URL de l'image originelle, pour que le client l'affiche directement dans le component
- **Image2url**: L'URL de l'image modifiée, pour que le client l'affiche directement dans le component
- **Difficulty**: Est une chaine donnant le niveau de difficulté pour ce jeu
- **SoloBestTimes**: Donne un tableau de BestTimes pour informer des meilleurs temps en solo
- **vsBestTimes**: Donne un tableau de BestTimes pour informer des meilleurs temps en multijoueur

**GameData:**

GameData est l'objet envoyé aux joueurs lorsqu'ils entrent dans une partie, il possède de ce fait des informations supplémentaires à GameForm.

Contient:

- **GameForm:** Est l'objet décrit plus tôt, pour faciliter l'accès aux données
- **DifferenceMatrix:** Est un tableau de nombres donnant la position des différences sur le jeu

**NewGame:**

NewGame est présent seulement lors de la création d'un nouveau jeu, pour envoyer les images en plus des informations classiques.

Contient:

- **Name:** Le nom du nouveau jeu à créer
- **Image1:** Une chaine de charactère contenant l'image en format base64
- **Image2:** Une chaine de charactère contenant l'image en format base64
- **NbDifference:** Le nombre de différences du nouveau jeu
- **DifferenceMatrix:** Est un tableau de nombre donnant la position des différences sur le jeu

**GameHistory:**

Contient l'historique d'une partie jouée et les informations concernant le vainqueur et le perdant.

Contient:

- **GameName**: Le nom du jeu
- **Date**: La date et l'heure du début de la partie
- **Duration**: Le temps que la partie a duré
- **GameMode**: Le mode de jeu de la partie
- **Username**: Le nom du joueur principal de la partie effectuée
- **Username2**: Le nom du deuxième joueur dans le cas d'une partie multijoueur

**BestTimes:**

**Constants:**

Contient les trois constantes définissant certains aspects de jeu.

Contient:

- **Countdown:** Un nombre décrivant le compte à rebours initial du mode temps limité
- **Penalty:** Un nombre pour décrire le temps de pénalité appliqué en cas d'indice utilisé
- **TimeBonus:** Le temps supplémentaire donné au joueur si une différence est trouvée

**DifferencePos:**

Est en réalité l'objet Vector2D, qui est simplement un objet contenant deux nombre **x** et **y**. Utile pour les positions du curseur pour de potentielles tentatives de clic durant la partie.

**GameRoom:**

GameRoom est un objet important puisqu'il décrit des attributs permettant le bon fonctionnement du serveur et des communications concernant une partie.

Contient:

- **UserGame:** Contient l'objet UserGame
- **RoomId:** Un nombre indiquant l'id de la room pour que le serveur communique avec tous les joueurs
- **Started:** Un Boolean donnant juste l'information si la partie a commencé ou non

**UserGame:**

UserGame est l'objet contenant les informations principales sur une partie en cours (sans les attributs du serveur pour la bonne communication).

Contient:

- **Username:** Est une chaine de charactère du nom du créateur de la partie
- **Username2:** Est le nom du deuxième joueur s'il y a lieu
- **PotentielPlayers**: Est la liste des joueurs potentiels en attente d'être accepté/rejeté par le créateur du jeu
- **GameData**: Est l'objet GameData décrit plus haut, contenant les informations statiques sur le jeu joué
- **NbDifferenceFound**: Donne le nombre de différences trouvées au total, change durant la partie
- **Timer**: Nombre très important mis à jour chaque seconde pour informer du temps actuel sur le jeu