303

Protocole de communication

Version 2.3

Historique des révisions

Date	Version	Description	Auteur
2023-03-07	1.1	Section Introduction	Anne Raymond
2023-03-19	1.2	Description des paquets de Socket.io	Anne Raymond et Enrique Arsenio Rodriguez Rodrigue
2023-03-20	1.3	Description de la communication client-serveur	Aurélie Nichols et Enrique Arsenio Rodriguez Rodrigue
2023-03-21	1.4	Description des paquets du protocole HTTP et ajout des définitions communes	Anne Raymond
2023-03-21	1.4.1	Mise à jour du tableau du protocole HTTP	Farid Bakir
2023-03-21	1.5	Correction des paquets du protocole http après des modifications du serveur	Louis-Philippe Daigle
2023-03-21	1.6	Révision finale de la totalité du document (sprint 2)	Enrique Arsenio Rodriguez Rodriguez et Sucy Han
2023-04-18	2.0	Mise à jour de la section communication client-serveur	Sucy Han
2023-04-20	2.1	Mise à jour de la description des paquets	Anne Raymond
2023-04-20	2.2	Mise à jour de la description des paquets	Louis-Philippe Daigle
2023-04-20	2.3	Mise à jour de la description de la communication par WS	Louis-Philippe Daigle

Table des matières

Protocole de communication

1. Introduction

Ce document contient toutes les informations sur les protocoles de communication utilisés dans notre projet, soit le protocole HTTP et le protocole WebSocket avec Socket.io. D'abord, ce document présente la communication client-serveur et explique les choix de protocole selon les fonctionnalités programmées. Ensuite, ce document présente la description des paquets utilisés par les protocoles de communication employés. Cette section contient aussi la description des interfaces et messages utilisés par les protocoles.

2. Communication client-serveur

Tout d'abord, nous avons utilisé le protocole HTTP pour traiter des données statiques qui ne nécessitent pas d'être mise à jour en temps réel. En effet, le serveur envoie une réponse seulement lorsque le client fait une demande. Par exemple, pour ce qui est de l'enregistrement et l'affichage des jeux, le protocole HTTP a été choisi car il s'agit de transferts de données ponctuels. En ce qui concerne la suppression de jeux, nous avons aussi choisi le protocole HTTP pour supprimer les données du serveur. La gestion des affichages des jeux supprimés se fait toutefois avec les Web Sockets puisqu'on veut traiter des données en temps réel et avoir l'affichage correspondant.

Ensuite, le protocole Web Socket a été utilisé lorsqu'un traitement de données en temps réel était préférable. Comme Socket.io permet une communication continue et bidirectionnelle, le client et le server peuvent maintenir une connexion persistante et s'envoyer des messages simultanément. Cela est donc extrêmement utile pour l'envoi des messages local et global, car nous voulons que les joueurs puissent voir des messages des autres joueurs ou du système tout en pouvant envoyer des messages simultanément. Pour ce qui est de l'historique des parties jouées, le protocole Web Socket a été utilisé puisque nous souhaitons mettre à jour l'historique en temps réel. Par exemple, si un joueur est en train de voir l'historique et un autre joueur termine une partie, la partie terminée doit immédiatement apparaître sur l'historique de l'autre joueur. Par la suite, nous avons aussi utilisé ce type de protocole pour les meilleurs temps car nous voulons mettre à jour fréquemment le classement. De plus, en ce qui concerne les salles d'attente, les Web Sockets sont nécessaires afin d'établir une connexion entre plusieurs joueurs. Finalement, lors d'un jeu en modes classique et temps limité, la gestion de la minuterie et des valeurs des constantes est faite grâce au protocole HTTP puisque ce sont des données dynamiques.

3. Description des paquets

Protocole HTTP

Requête					Réponse	
Méthode	Route	Description	Corps	Paramètres	Code de retour	Contenu du corps
Storage Controller						
POST	/api/storage/read	Lire les fichiers de base pour un jeu	StorageMessageWithIdAndType : L'id du jeu et le type de fichier voulu	-	200 : OK	Le fichier contenant soit l'information du jeu ou une image (string)
POST	/api/storage/modify	Modifier les fichiers de base pour un jeu	StorageMessageWithIdTypeAndData : L'id du jeu, le type de fichier voulu et les nouvelles données pour modifier	-	201 : CREATED	Le fichier contenant soit l'information du jeu ou une image (string)
POST	/api/storage/create	Pour ajouter un jeu au serveur lorsqu'on le créer à partir de la page de création	StorageMessageNewGame: L'image originale, l'image modifiée et le fichier json d'informations sur le jeu	-	201 : CREATED	Le fichier contenant soit l'information du jeu ou une image (string)
DELETE	/api/storage/game	Pour supprimer un jeu selon son ID sur la page de configuration	StorageMessageWithId : L'id du jeu qu'on veut supprimer	-	204: NO CONTENT	-
DELETE	/api/storage/games	Pour supprimer tous les jeux en même temps à partir de la page de configuration	-	-	204 : NO CONTENT	-
GET	/api/storage/getValidIds	Pour aller chercher les ID de tous les jeux	-	-	200 : OK	Un tableau de tous les id des jeux sur le serveur (ValidIdsMessage : {validIds : [] })
GET	/api/storage/originalImage/:X.bmp	Pour accéder à l'image originale d'un jeu selon son ID	-	L'id (X) du jeu voulu Où X ∈ N*	200: OK / 304 : NOT MODIFIED	L'image originale (bytes)

GET	/api/storage/modifiedImage/ :X.bmp	Pour accéder à l'image modifiée d'un jeu selon son ID	-	L'id (X) du jeu voulu Où X ∈ N*	200: OK / 304 : NOT MODIFIED	L'image modifié (bytes)
POST	/api/storage/score	Pour aller chercher le score d'un jeu selon son ID et son type de partie classique (solo ou 1v1	L'id du jeu et le type de jeu classique (solo ou 1v1).	-	200 : OK	Un tableau contenant les 3 meilleurs scores (Userscore[])

Protocole WS avec Socket.io

Évènements reçus par le serveur

Évènements gérés par le Socket Manager

Évènement	Contenu	Utilité
joinRoom	L'identifiant de la salle (string).	Placer un client dans une salle. Les salles sont utilisées pour les jeux multijoueur ainsi que pour les salles d'attentes avant d'entrer dans une partie multijoueur
leaveRoom	L'identifiant de la salle (string).	Enlever un client d'une salle. Les salles sont utilisées pour les jeux multijoueur ainsi que pour les salles d'attentes avant d'entrer dans une partie multijoueur
disconnect	-	Déconnecter les clients des sockets.

Évènements gérés par le Game Room Handler

Évènement	Contenu	Utilité
joinGame	Le nom de l'utilisateur, l'id de la salle (string) et l'id du jeu (number)	Placer les deux joueurs d'une partie multijoueur dans la même salle pour commencer la partie en mode classique ou temps limité.
updateGamePlayers	L'information sur les joueurs (PlayerData[]), l'id de la salle (string) et l'id du jeu (number)	Mettre à jour les informations sur les joueurs dans une même salle.
scoreEmit	L'id du jeu (number), le mode de jeu (string) et le score à ajouter (UsersScore)	Mettre à jour le classement d'un jeu sur la base de données lorsqu'une partie a été jouée.
resetGameScores	L'id du jeu (number)	Remettre le classement d'un jeu aux valeurs par défault.
resetAllGames	-	Remettre le classement de tous les jeux aux valeurs par défault.
globalMessage	Le message envoyé (ChatMessage).	Envoyer des messages globaux de parties à tous les joueurs.
leaveGameRoom	L'id de la salle (string) et l'id du jeu (number)	Enlever un joueur d'une partie multijoueur.
getScores	L'id du jeu (number) et le mode de jeu (string)	Accéder au classement d'un jeu pour un mode de jeu spécifique.
gatAllScores	L'id du jeu (number)	Accéder au classement d'un jeu pour le mode classique solo et 1v1.
startTimer	L'id de la salle (string) et l'id du jeu (number)	Démarrer la minuterie.

stopTimer	L'id de la salle (string) et l'id du jeu (number)	Arrêter la minuterie
chatMessage	Le message envoyé (ChatMessage), l'id de la salle (string) et l'id du jeu (number)	Envoyer un message à son adversaire dans une partie multijoueur classique.
timedchatMessage	Le message envoyé (ChatMessage) et l'id de la salle (string)	Envoyer un message à son coéquipier dans une partie multijoueur temps limité.
loadNextTimedGame	L'id de la salle (string) et l'id du jeu (number)	Charger le prochain jeu pour une partie avec le mode temps limité.
leaveGame	L'id de la salle (string) et l'id du jeu (number)	Quitter une partie classique en cours.
leaveTimedGame	L'id de la salle (string) et l'id du jeu (number)	Quitter une partie temps limité en cours.
validateMove	Un tableau contenant le point cliqué (Point), les informations des joueurs (PlayerData[]) et le type jeu (GameType).	Valider l'action d'un joueur pour savoir s'il a trouvé une différence ou non.
timedGameEnded	L'id de la salle (string).	Gérer la fin de partie pour le mode temps limité.
addGameHistory	L'information nécessaire pour l'historique d'un jeu (HistoryData).	Ajouter une partie jouée à l'historique des parties jouées sur la base de données.
getHistory	-	Accéder à l'historique des parties jouées.
deleteHistory	-	Supprimer l'historique des parties jouées sur la base de données.

Évènements gérés par le Waiting Room Handler

Évènement	Contenu	Utilité
createWaitingRoom	Un tableau contenant l'id du jeu (number) et le nom du jeu (string)	Créer la salle d'attente pour une partie multijoueur.
deletedFromServer	L'id du jeu (number).	Communiquer avec les clients que le jeu a été supprimé du serveur. La suppression des données est faite par http et cet évènement sert de confirmation après la suppression.
deleteWaitingRoom	L'id du jeu (number).	Fermer la salle d'attente pour un jeu lorsque le créateur d'une partie quitte.

deletedEverythingFromServer	-	Communiquer avec les clients que tous les jeux ont été supprimés du serveur. La suppression des données est faite par http et cet évènement sert de confirmation après la suppression.
verifyUsername	usernameMessage : UsernameMessage	Valider le nom d'utilisateur entré pour un jeu en mode classique.
isCardCreating	L'id du jeu (number).	Vérifier si une partie a été créé pour un jeu et qu'un joueur est en attente.
rejected	L'id du jeu (number).	Rejeter un adversaire qui demande à se joindre à une partie multijoueur classique.
accepted	Un tableau contenant le mode de jeu (string), l'id du jeu (number) et le nom du joueur (string)	Accepter un adversaire qui demande à se joindre à une partie multijoueur classique ou accepter un joueur qui commence une partie en solo.
quitLine	L'id du jeu (number).	Modifier la file d'attente pour une partie lorsqu'un joueur quitte cette file.
joinWaitingRoom	Un tableau contenant l'id du jeu (string) et les informations du nouveau joueur (UserWaiting).	Ajouter un joueur à la file d'attente pour un jeu
timedCheckUserName	Le nom d'utilisateur (string)	Valider le nom d'utilisateur entré pour un jeu en mode temps limité.
timedFindGame	Le nom d'utilisateur (string)	Effectue une vérification pour savoir si un jouer est déjà en attente. Si oui, les deux joueurs reçoivent le message pour commencer une partie avec des images aléatoire. Si non, le joueur est mis dans la file d'attente.
timedAbandon	-	Abandonner une partie en mode temps limité.
createdGameCard	-	Indiquer aux parties mode temps limité en cours qu'il y a un nouveau jeu qui a été créé pour l'ajouter à la suite de jeux aléatoire.

Évènements gérés par le Constants Service

Évènement	Contenu	Utilité
constant	Les nouvelles valeurs des constantes qui sont modifier.	Modifier les constantes pour tous les joueurs sur le site.
getConstant	-	Lire et obtenir les constantes du site.

Évènements envoyés par le serveur

Évènements gérés par le Client Socket Handler

Nom de l'évènement	Contenu	Utilité
joined	-	Indiquer au client qu'il a rejoint la file d'attente pour un jeu.
deletedRoom	-	Indiquer au client que la partie pour laquelle il était en attente a été annulé.
roomClosed	L'id du jeu (number).	Indiquer au client que la partie pour laquelle il était en attente a été annulé.
rejected	L'id du jeu (number).	Indiquer au client que le créateur de la partie l'a rejeté.
accepted	L'URL de la partie (string).	Indiquer au client que le créateur de la partie l'a accepté et commencer la partie.
emptyLine	-	Vider la file d'attente pour un jeu.
nextOpponent	L'information sur le joueur en attente (Userwaiting)	Mettre à jour l'adversaire affiché au créateur de la partie.
deletedFromServer	-	Avertir les joueurs en attente que le jeu a été supprimer du serveur.

Évènements gérés par le Controller Game Service

Nom de l'évènement	Contenu	Utilité
updateGamePlayers	L'information des joueurs (PlayerData[])	Mettre à jour les informations des joueurs d'une partie.
clock	Le temps de minuterie d'un jeu (number)	Mettre à jour la minuterie des parties.
chatMessage	Le message envoyé (ChatMessage)	Envoyer un message à son adversaire.
playerLeft	Le nom d'utilisateur du joeur qui a quitté (string)	Avertir qu'un joueur a quitté la partie.
opponentDisconnected	-	Avertir qu'un joueur s'est déconnecté.
validMoveMade	La différence trouvée (Difference)	Indiquer que l'action prise par un joueur est valide.
invalidMoveMade	Le point qui a été cliqué (Point)	Indiquer que l'action prise par un joueur est invalide.
Constant	Les constantes de jeu (GameConstants[]).	Mettre à jour les constantes de jeu sur le client.
loadConstant	Les constantes de jeu (GameConstants[]).	Charger les constantes de jeu sur le client.
loadNextTimedGame	L'id du jeu (number).	Charger le prochain jeu dans une partie en mode temps limité.
timedGameEnded	-	Indiquer qu'une partie en temps limité est terminée.

Évènements gérés par le Timed Waiting Page Component

Nom de l'évènement	Contenu	Utilité
timedUsernameTaken	-	Avertir le client que le nom d'utilisateur voulu est indisponible.
timedUsernameAvailable	-	Indiquer au client que son nom d'utilisateur est valide.
timedGameFound	Un tableau qui contient l'id du premier jeu à charger, le nom de la room et le nom d'utilisateur.	Effectuer le chargement initial d'une partie sur les clients.
numberGamesChanged	-	Avertir le client que le nombre de jeux disponibles pour une partie en temps limité a changé.

Évènements gérés par le Game Selection Page Component

Nom de l'évènement	Contenu	Utilité
cardDeleted	-	Indiquer qu'un jeu a été supprimé.
resetAllGames	-	Mettre à jour les classements de tous les jeux.
numberGamesChanged	-	Indiquer que le nombre de jeu a changé.

Évènements gérés par le Configuration Page Component

Nom de l'évènement	Contenu	Utilité
cardDeleted	-	Indiquer qu'un jeu a été supprimé.
resetAllGames	-	Mettre à jour les classements de tous les jeux.
updateHistory	Un tableau contenant tous l'historique des parties jouées (HistoryData[]).	Mettre à jour l'historique des parties jouées sur le client.

Évènements gérés par le User Name Input Component

Nom de l'évènement	Contenu	Utilité
verifyUsername	La réponse de la validation (boolean).	Valider le nom de l'utilisateur.

Évènements gérés par le Game Card Component

Nom de l'évènement	Contenu	Utilité
createdGame	L'id du jeu (number).	Indiquer qu'une partie multijoueur a été créée et est en attente.
deletedWaitingRoom	L'id du jeu (number).	Indiquer qu'une partie multijoueur en attente a été annulé.
CreatorLeft	L'id du jeu (number).	Indiquer que le créateur d'une partie multijoueur en attente a quitté.

IsCardCreating	Un tableau contenant une valeur boolean (est disponible pour créer) et l'id du jeu.	Effectue la gestion du chargement des cartes de jeu.
getAllScores	Un tableau contenant l'id du jeu (number) et les scores (GameRankings).	Obtiens tous les scores pour le jeu.
newRecord	L'id du jeu, le mode de jeu et le score pour ce jeu.	Assure que les score afficher à l'utilisateur sont à jour.

Évènements gérés par le End Game Popup Component

Nom de l'évènement	Contenu	Utilité
newRecord	L'id du jeu, le mode de jeu et le score pour ce jeu.	Indique qu'il y a eu un nouveau record pour ce jeu
getScores	La liste des records pour le jeu courant.	Assure que les score afficher à l'utilisateur sont à jour.

Évènements gérés par le Constants Component

Nom de l'évènement	Contenu	Utilité
constant	Un tableau des constantes.	Met à jour les constantes sur la page de l'utilisateur.
loadConstant	Un tableau des constantes.	Met à jour les constantes sur la page de l'utilisateur.

Définitions communes

Interfaces

```
PlayerData :
    username: string;
    differencesFound: Difference[];
    invalidMoves: Point[];
    socketId : string;
    userName : string;
UserWaiting :
    socketId : string;
    userName : string;
Point:
    x: number;
    y: number;
ImageDiffs:
    differences: Point [][];
    difficulty: string;
TimerObject:
    time: number;
    totalTime: number;
    gameMode: string;
    isActive: boolean;
UsernamesObject:
    usernames: string[];
    id: number;
UserWaiting:
    socket: string;
    userName: string;
GameConstants :
    name: string;
    time: number;
DisconnectMessage:
    gameId: number;
    totalTime: number;
    waitingLineIndex: number;
IsUserPosition:
    isInLine: boolean;
    position: number;
```

```
GameRankings :
    gameId: number;
    singlePlayer: UsersScore[];
    multiplayer: UsersScore[];
CanvasState:
    context1: ImageData;
    context2: ImageData ;
DifferenceEvent:
    difference: Difference;
    username: string;
UserGame:
    username: string;
    socketId: string;
GameRoom:
    room: string;
    id: number;
    users: UserGame[];
    players: playerData[];
    timer: TimerObject
WaitingRoom:
    creatorId: string;
    room: string;
    gameId: number;
    waitingLine: UserWaiting[];
messageScoreInfo:
    position: number;
    gameName: string;
    mode: string;
GameCardType:
    game: GameData;
    isAvailable: boolean;
ClientWaitingObject:
    gameId: number;
    playerName: string;
    opponentName: string;
    creatorName: string;
    IsCreator: string
```

```
Rectangle:
    point1: Point;
    point2: Point;
Difference :
    rectangles: Rectangle[];
UsersScore:
    name: string;
    time : string;
gameHistory :
    time : number;
    duration : number;
    gameMode : string;
    player1 : string;
    player2 : string;
    winner : string;

UsersScore:
    name: string;
    time: number;
GameEvent:
    type: GAME_EVENTS;
    time: number;
    eventData: any;
PlayerData:
    username: string;
    differencesFound:
    Difference[];
    invalidMoves: Point[];
    isActive: boolean;
ChatMessage :
    username: string;
    message: string;
    time: number;
    textColor: ChatColor;
    backgroundColor: ChatColor;
ChatColor :
    r: number;
    g: number;
    b: number;
GameData:
    id: number;
```

CanvasChanges:
    past: CanvasState[];
    next: CanvasState[];

Dimension:
    width: number;
    height: number;

DatabaseGame:
    gameId: number;
    singlePlayer: UsersScore[];
    multiplayer: UsersScore[];

    NewOpponent: boolean
    IsWaiting: boolean
    WaitingMessage: string

HistoryData:
    date: string;
    duration: number;
    mode: string
    player1: PlayerInfo
    player2: PlayerInfo

    title: string;
    difficulty: string;
    numberOfDifferences: number;

PlayerInfo:
    name: string;
    isWinner: boolean;
    isQuitter: boolean

CallbackSignature:
    (params: any) => void
SocketParams: any
PrivateFunction:any

Messages

StorageMessageWithId:
    Id: number;

StorageMessageWithIdAndType :
    Id: number;
    Type: FILE_TYPE;

StorageMessageWithIdTypeAndData:
    Id: number;
    Type: FILE_TYPE;
    Data: string;

IdMessage:
    Id: number;

TimeMessage:
    time: number;

StorageMessageWithType:
    Type: FILE_TYPE;

ValidIdsMessage:
    vallidIds: number[];

AllUsernamesMessage:
    ValidIds:number[];
UsernameMessageWithTime:
    time: number;
    username: string;

StorageMessageNewGame:
    originalImageData: string;
    modifiedImageData: string;
    imageJSON: string;

StorageMessageGameConst :
    Init : number;
    Penalty : number;
    Bonus : number;

UsernameMessage:
    username: string;
    id: number;

FileTypeMessage:
    Type: FILE_TYPE;

ValidityMessage:
    valid: boolean;

validMoveResponseMessage:
    Difference?: Difference;

IsUsernameAvailableMesage:
    vallidIds: number[];

ValidMoveResponseMessage:
    valid: boolean;
    difference?: Difference;