207

Protocole de communication

Version 2.0

Historique des révisions

Date	Version	Description	Auteur
2023-03-21	1.1	Rédaction de l'architecture de communication	Amira Tamakloe
2023-04-19	2.0	Rédaction de l'architecture de communication	Amira Tamakloe

Table des matières

Protocole de communication

1. Introduction

Le document relate des différents aspects interactions entre un client et le serveur dans le contexte de notre application web. Son but premier est de présenter les protocoles utilisés dans notre application ainsi que leur La deuxième section aborde l'usage de deux protocoles, soit WebSocket et HTTP pour la communication entre le client et le serveur de notre application et les raisons pour lesquelles un protocole a été privilégié aux dépens de l'autre pour l'implémentation de fonctionnalité de notre application. La dernière section quant à elle détaille le contenu des différents types de paquets utilisés pour interagir dans notre application pour les deux protocoles de communication. Pour le protocole HTTP, nous préciserons la méthode, la route, le corps et les paramètres si présents ainsi que les informations de la réponse. Pour le protocole WebSocket, nous indiquerons le nom de l'événement, sa source et son contenu. Nous inclurons également toute autre information pertinente pour la compréhension des paquets échangés dans le cadre de notre application.

2. Communication client-serveur

La communication client-serveur du projet se fait à travers des requêtes HTTP et par le protocole WebSocket. Les requêtes brèves ne nécessitant pas une communication persistante entre le serveur et le client ont été implémentés grâce au protocole HTTP. En revanche les requêtes ayant besoin de persistance, c'est-à-dire d'une communication constante entre le client et le serveur, et qui nécessitent que le serveur puisse envoyer des informations au client sans que celui-ci le requestionne ont été implémentés à l'aide de WebSocket.

Le clavardage, notamment, utilise le protocole WebSocket puisqu'une communication en temps réelle entre le serveur et le client est nécessaire. En utilisant WebSocket, nous n'avons pas besoin d'envoyer une requête à chaque fois que le client souhaite communiquer avec le serveur, la liaison étant déjà mise en place. De plus, le fait que la liaison soit persistante fait que nous réduisons considérablement le délai d'attente pour la réception de l'information. Le serveur aussi envoie certains évènements directement au client sans que celui-ci le demande ce qui est impossible avec l'utilisation de HTTP. L'envoi de l'heure auquel un évènement s'est produit dans le clavardage utilise aussi le protocole WebSocket puisqu'une liaison constante doit être établie afin que le serveur puisse déterminer quand est-ce que l'évènement s'est produit depuis l'initialisation de la liaison.

La gestion de toutes les salles d'attente que ce soit celle du temps limité ou du mode classique multijoueur ainsi que toutes les salles de jeu utilisent le protocole WebSocket. L'utilisation de WebSocket nous permet de pouvoir identifier plusieurs clients, les relier ensemble en temps réel ainsi que de les placer dans une salle. Un utilisateur en mode solo classique ou temps limité utilise le protocole WebSocket lors d'une partie pour être en mesure de voir les différents évènements dans le chat. Pour être en mesure de voir ses évènements, il doit se trouver dans une salle individuelle d'où l'usage des sockets et du principe de salle. En multijoueur, les sockets nous permettent de relier deux joueurs ensemble et de leur permettre de communiquer ensemble et de ce fait même de voir les évènements relatifs au deux lors de la partie à laquelle ils prennent part en temps réels. Le protocole WebSocket est indispensable puisque les fonctionnalités de salle sont primordiales. De plus, puisque le serveur doit renvoyer à un utilisateur tous les autres usagers qui sont dans la même salle que lui sans nécessairement que l'usager en plus de permettre la collaboration ou un duel dans un même jeu, l'implémentation du protocole était logique.

Les différents évènements lors d'une partie sont émis par les sockets comme l'usage de constante, le clignotement de pixel, la découverte de différence ou une erreur commise pour être en mesure d'envoyer ces évènements là à des utilisateurs se situant dans une même salle sans que celui-ci ne le réquisitionne. Le choix du protocole WebSocket est du à cette écoute constante des évènements dans une partie et par le fait que le serveur doit être en mesure d'envoyer des évènements sans que ce soit demandés.

La gestion de la reprise vidéo est effectué en partie par les sockets pour être en mesure d'envoyer à plusieurs components un évènement relatif à un socket particulier. L'usage du socket permet à l'utilisateur de visionner le jeu

qu'il vient de jouer et non celui d'une autre personne ainsi il doit se retrouver dans une salle. La reprise vidéo fait aussi appelle au chat donc on utilise les sockets pour envoyer des messages du serveur au chat. En somme les sockets en reprise vidéo nous permettent d'associer des évènements au bon utilisateur.

En somme, les différentes fonctionnalités qui nécessitent la communication constante entre les deux usagers qui jouent ensemble, implémentent le protocole WebSocket puisqu'il faut constamment mettre à jour des informations sur leurs deux vues respectives, en plus de leur permettre de communiquer.

Le protocole HTTP a été le choix privilégié pour toutes requêtes qui nécessite un accès à de l'information sur la base de données, MongoDB. Le client envoie une requête HTTP qui sera géré par Node.js et Express qui récupéreront l'information souhaitée depuis la base de données et qui la renverront au client en réponse à la requête http. Le tout en s'assurant de mettre à jour la vue avec l'information demandée. Le protocole HTTP est utilisé pour toutes les requêtes qui demandent les ressources de MongoDB, car la connexion n'a pas besoin d'être maintenue. Une fois que la ressource est envoyée, il n'y a aucune utilité à ce que la connexion soit gardée. Les différents types de requête disponibles avec ce protocole permettent de rendre le code plus lisible et claire. La méthode « get » a été utilisée pour récupérer un jeu, l'ensemble des jeux stockés sur MongoDB, une fiche d'historique de partie ou l'ensemble de l'historique des parties ayant été joués. La méthode « post » permet de sauvegarder un jeu ou un historique crée par l'utilisateur, sur la base de données. La méthode « delete » permet de supprimer la ressource c'est-à-dire soit la fiche d'historique ou le jeu ou même l'entièreté des historique ou des jeux de la base de données. La méthode « patch » permet de modifier une information de la base de données. La méthode patch n'a pas été implémenté pour l'historique car elle n'est pas requise. Cependant elle l'est pour changer des champs dans les jeux. La méthode patch est utile pour changer les meilleurs temps dans le mode solo et multijoueur, réinitialiser tous les temps d'une fiche ou même de toutes les fiches. Les méthodes énumérées ci-dessus retournent ou envoient des données. Une fois que l'information est obtenue, il est inutile de conserver la liaison avec le serveur.

Le protocole HTTP est utilisé pour enregistrer les images téléchargées à la suite de la création d'un nouveau jeu dans le serveur. La méthode « post » permet la sauvegarde de l'image venant d'être téléchargée dans le serveur. Cette sauvegarde ne se fait qu'une seule fois et uniquement lorsqu'on décide de créer un nouveau jeu. La méthode « get » quant à elle permet au client de récupérer ces images là pour les afficher. La méthode « delete » permet la suppression de l'image sélectionner. Ainsi, il est inutile de garder une connexion entre le client-serveur puisqu'une fois l'obtention, la délétion ou le téléchargement d'une image est complété, la connexion n'est plus utile. D'où l'usage du protocole HTTP. De plus HTTP permet la gestion de plusieurs formats dont BMP, et ce, même si la taille des fichiers est grande.

Le protocole HTTP est aussi utilisé pour vérifier qu'une différence a bel et bien été trouvée. La position du clic de la souris ainsi que le tableau de différence sont envoyées au serveur à travers une méthode « post ». Cela nous permet de vérifier que la position du clic correspond à une différence entre l'image originale et celle qui a été modifiée. Le client a uniquement besoin de la confirmation ou l'infirmation qu'il a trouvé une différence. Ainsi, une connexion brève

peut être d’usage d’où l’utilisation de ce protocole.

Le protocole HTTP est d’usage pour modifier les constantes de jeux. La méthode « patch » nous permet d’accéder au fichier constant.json et de modifier les valeurs inscrites. La méthode « get » permet de récupérer les valeurs inscrites dans le fichier pour qu’elles puissent être utilisé par le client. Le choix de ce protocole est dû au fait qu’une fois les constantes sont fixés par l’utilisateur, et les valeurs des constantes ont été initialisés par le jeu on n’a plus besoin de les avoir ainsi une connexion persistante n’est pas requise.

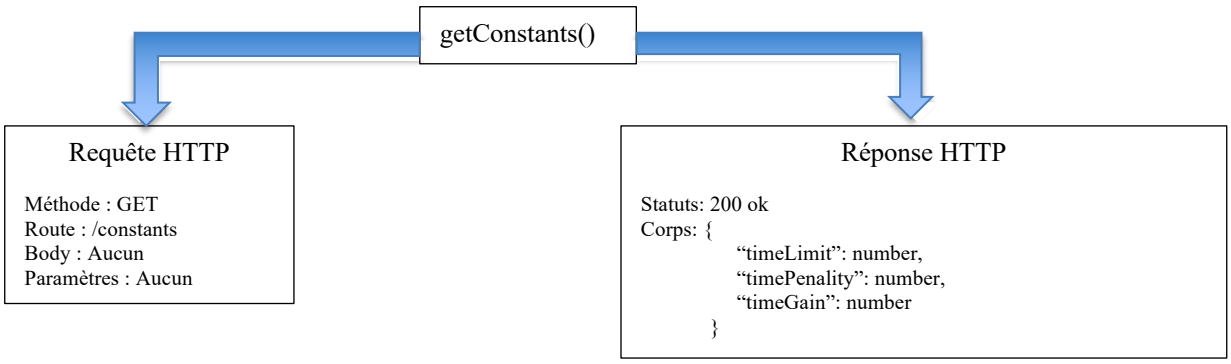
En somme, http est le protocole d’usage pour communiquer avec MongoDB ainsi que de toutes opérations qui ne requiert plus de communication une fois que la requête a été complété.

3. Description des paquets

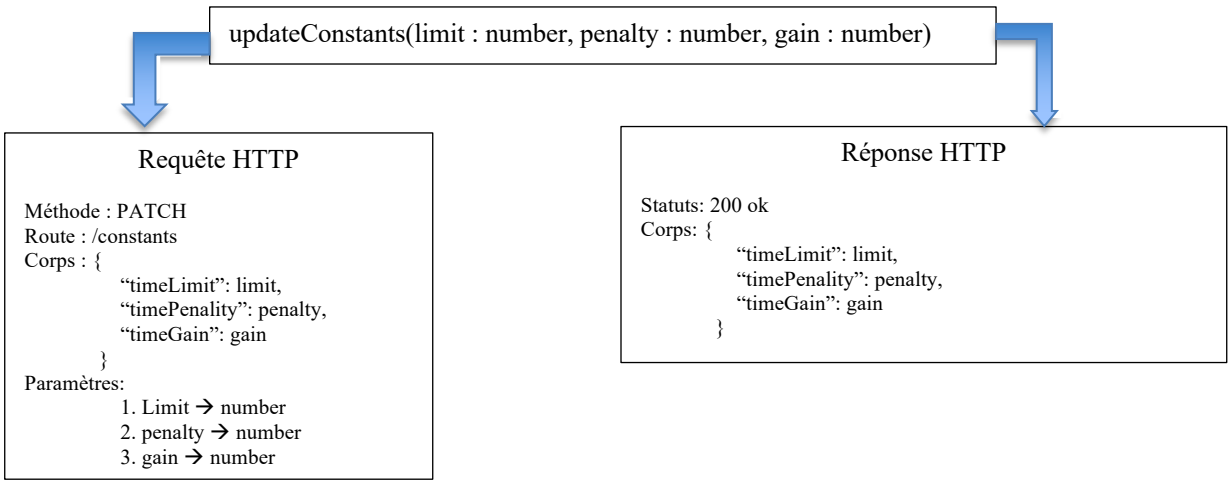
Paquets HTTP

1. Gestion des constantes

1.1 Schéma de communication HTTP de la méthode getConstants

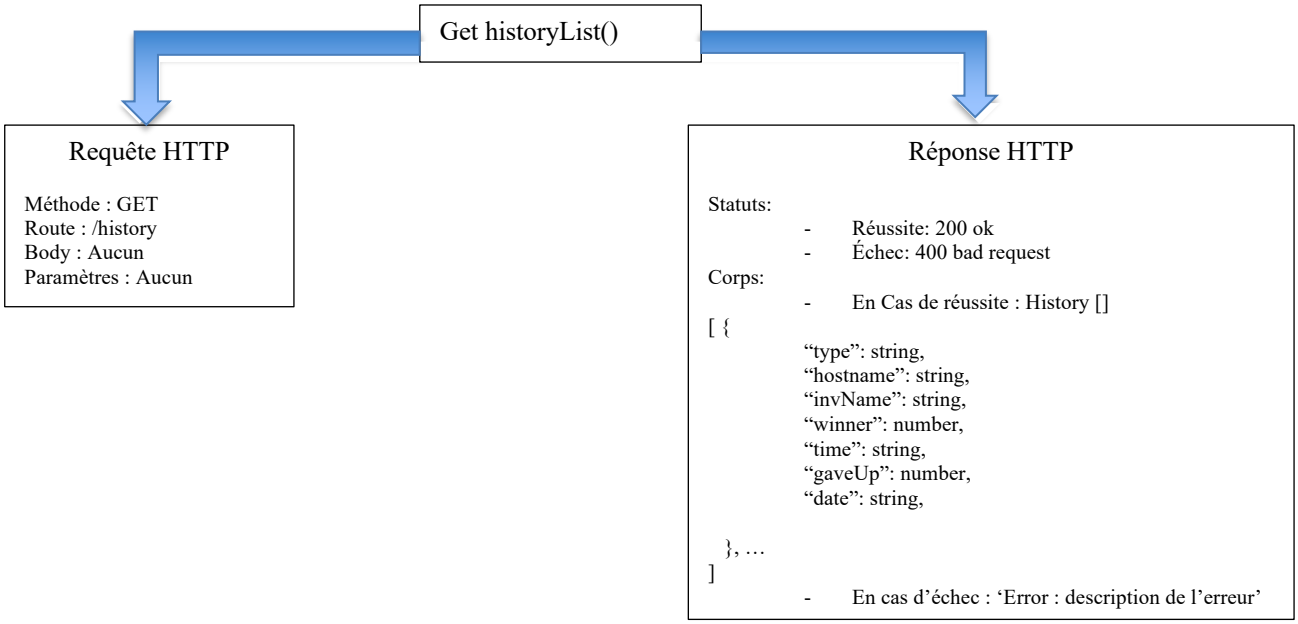


1.2 Schéma de communication HTTP de la méthode updateConstants

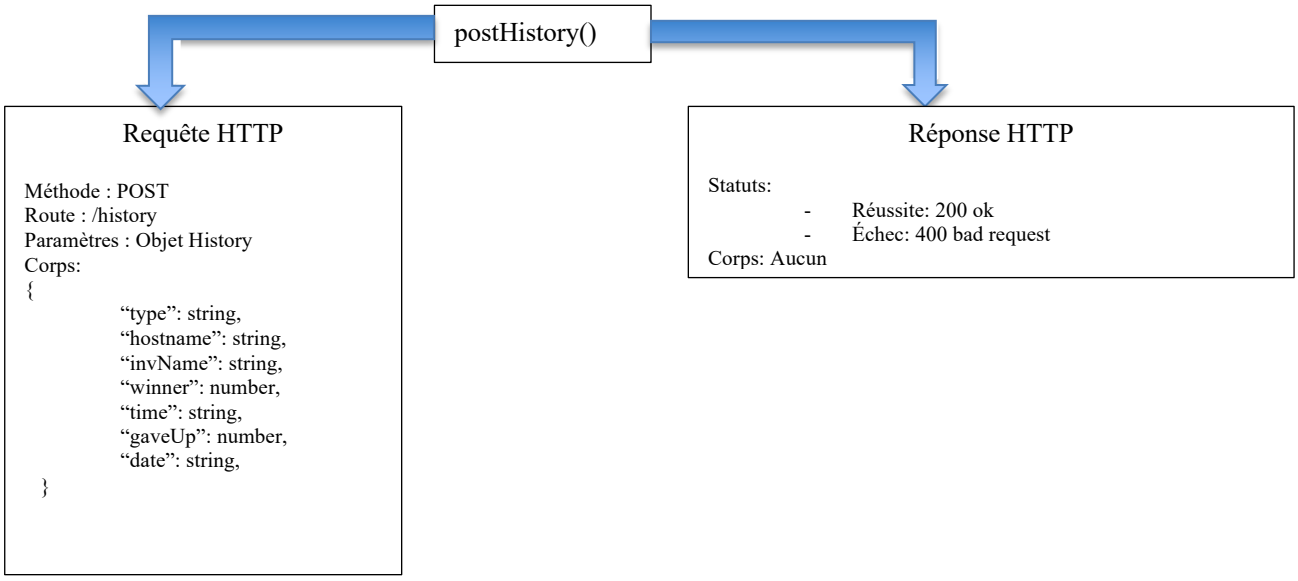


2. Historique des parties

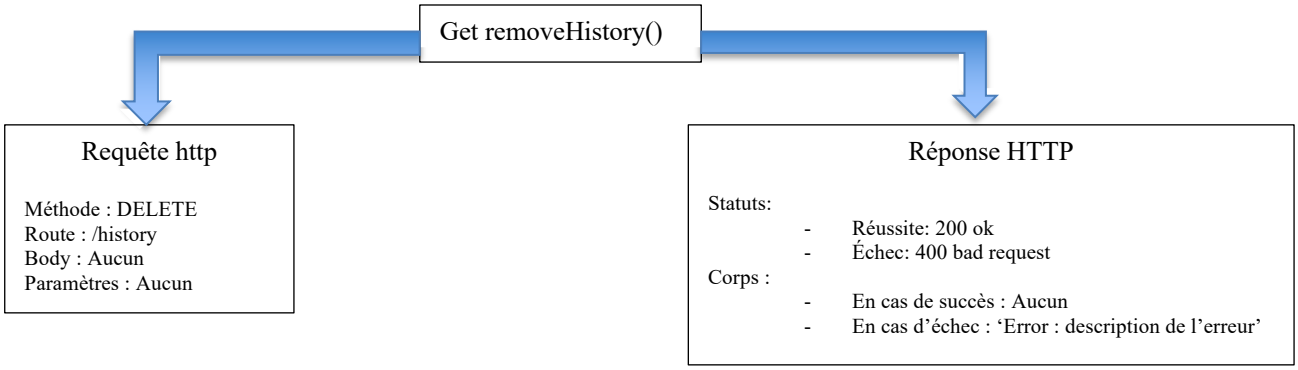
2.1 Schéma de communication HTTP de la méthode get historyList



2.2 Schéma de communication HTTP de la méthode postHistory

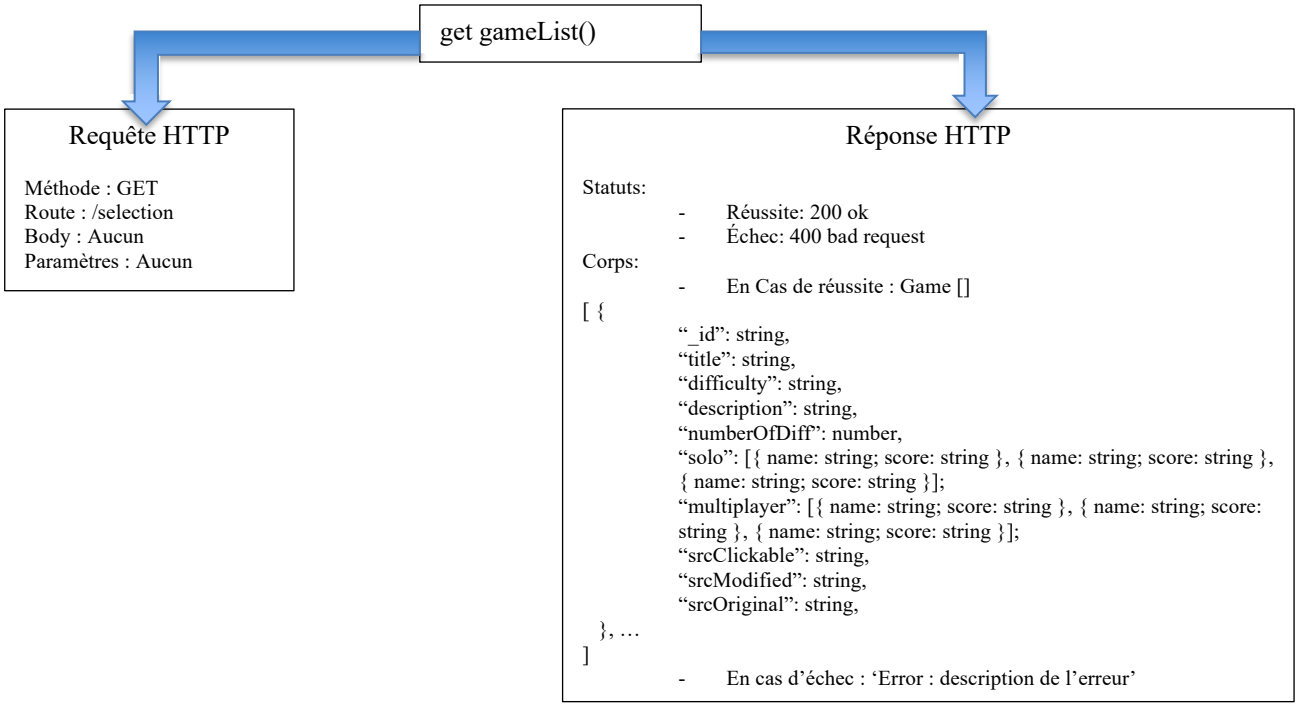


2.3 Schéma de communication HTTP de la méthode get removeHistory

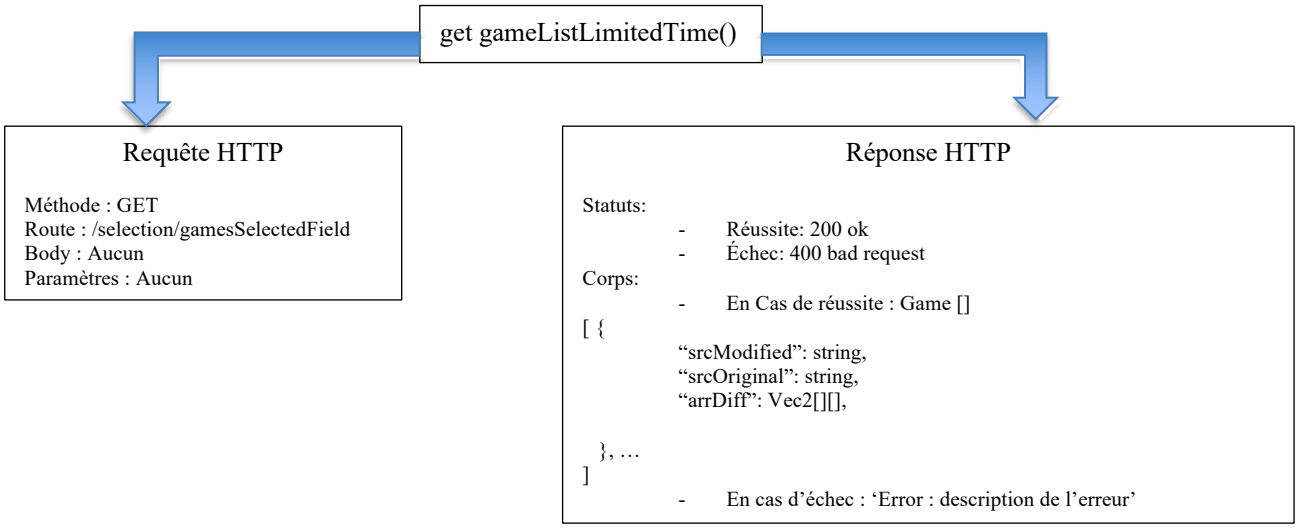


3. Gestion des Jeux

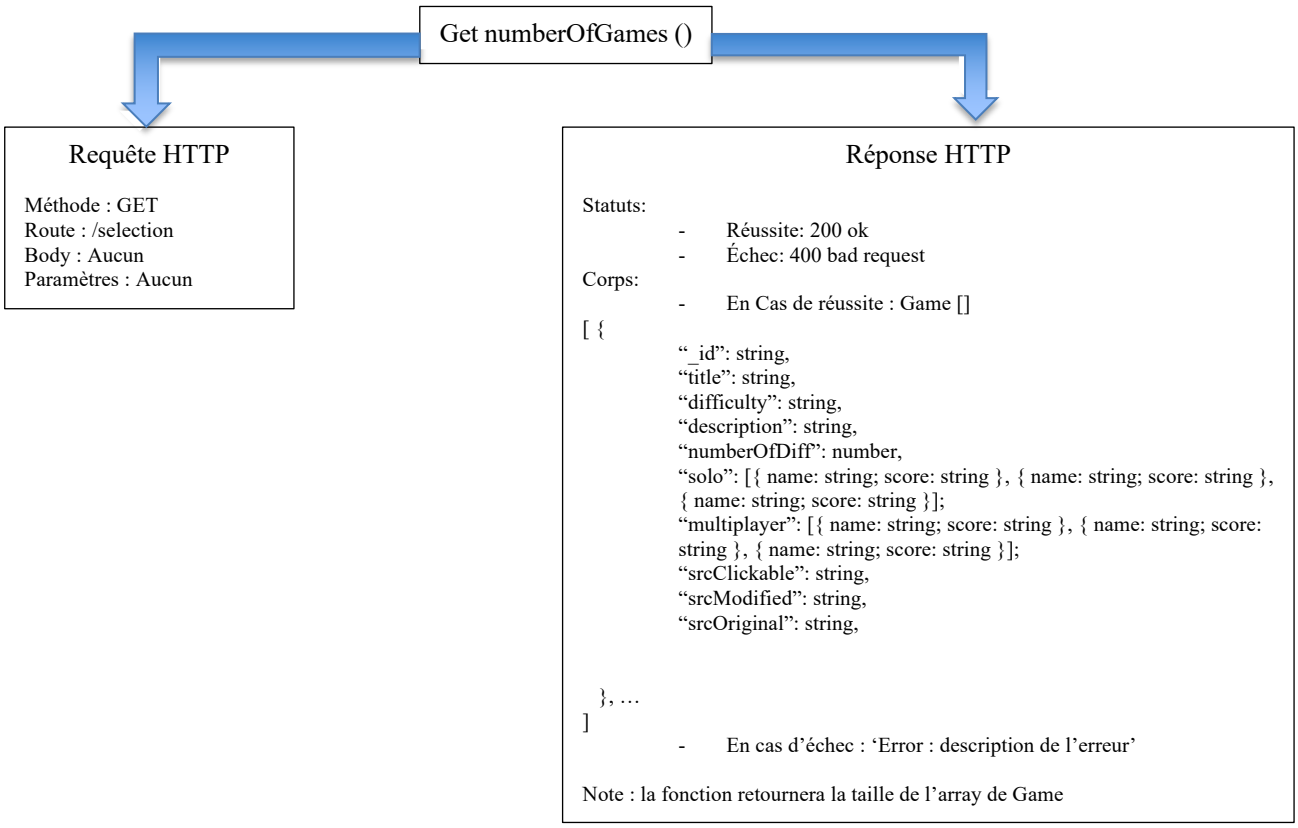
3.1 Schéma de communication HTTP de la méthode get gameList



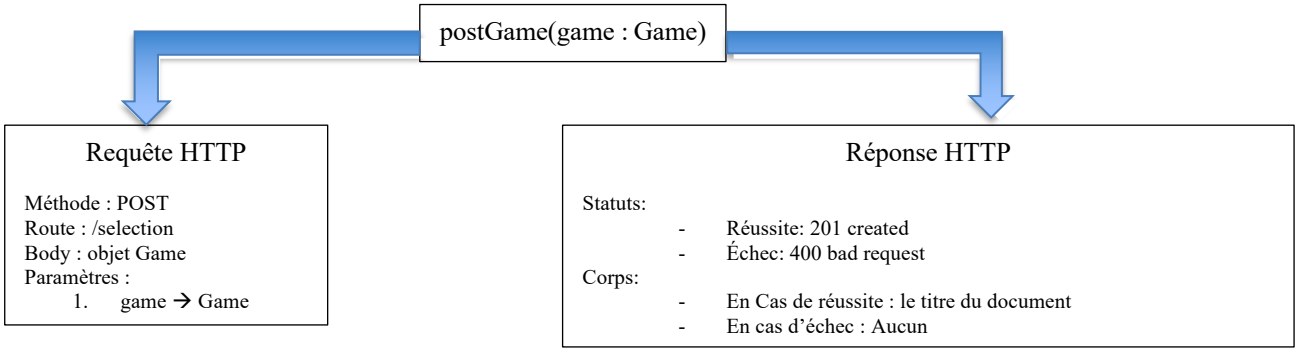
3.2 Schéma de communication HTTP de la méthode get gameListLimitedTime



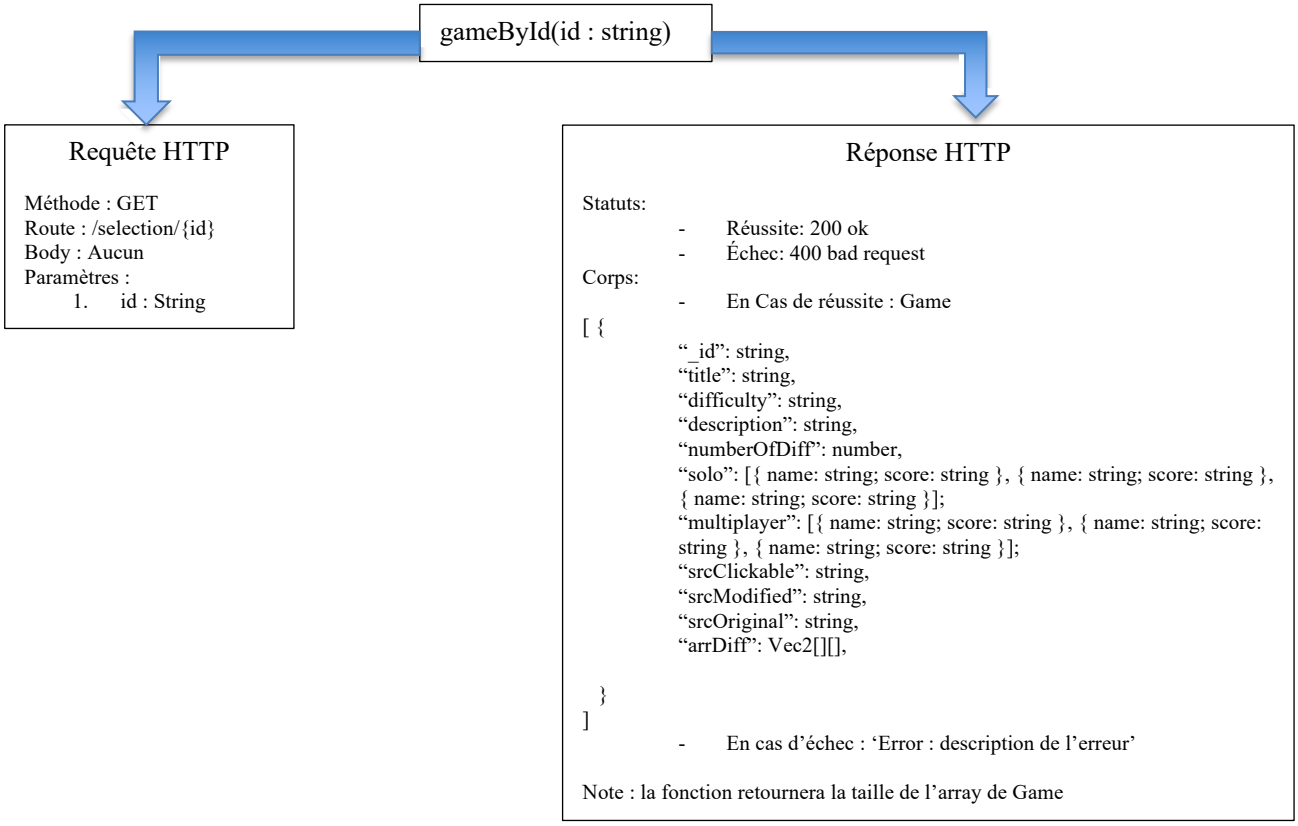
3.3 Schéma de communication HTTP de la méthode get numberOfGames



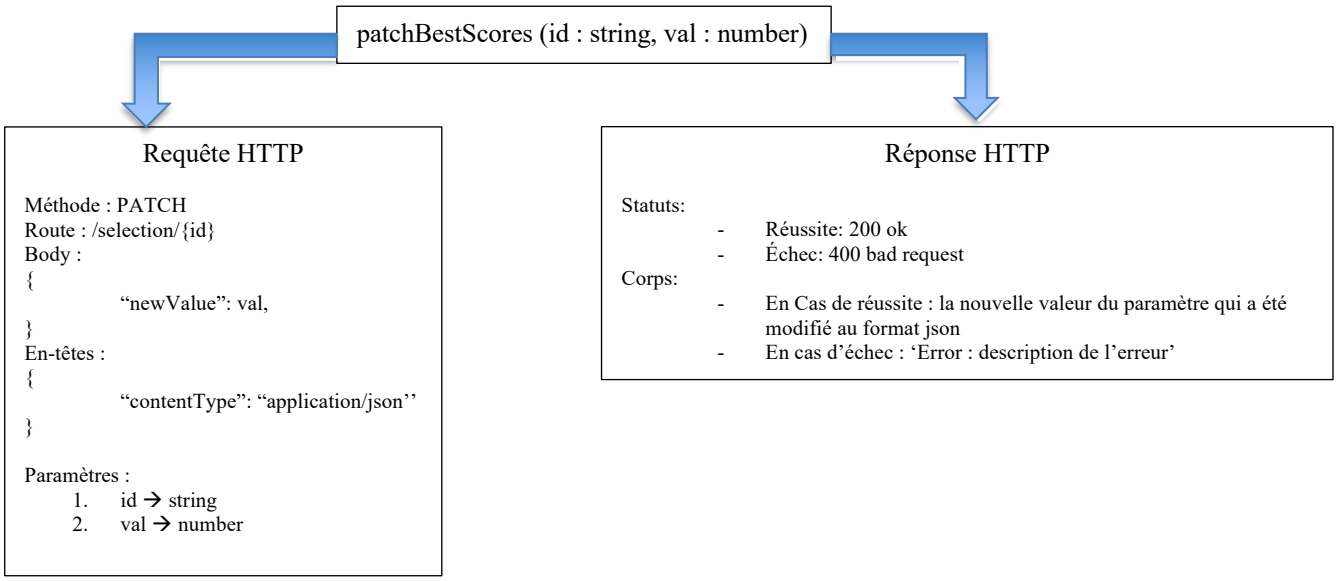
3.4 Schéma de communication HTTP de la méthode postGame



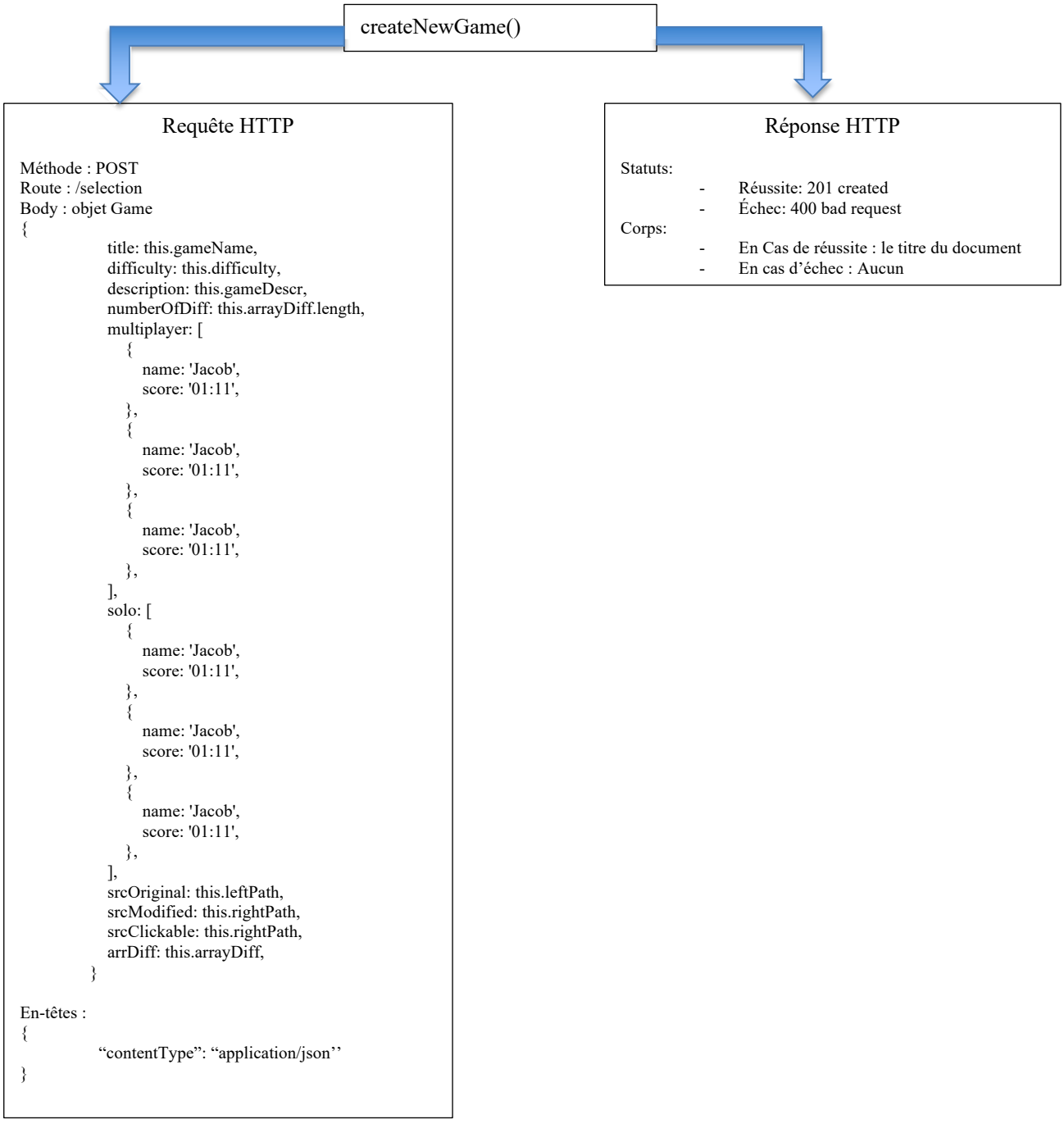
3.5 Schéma de communication HTTP de la méthode gameById



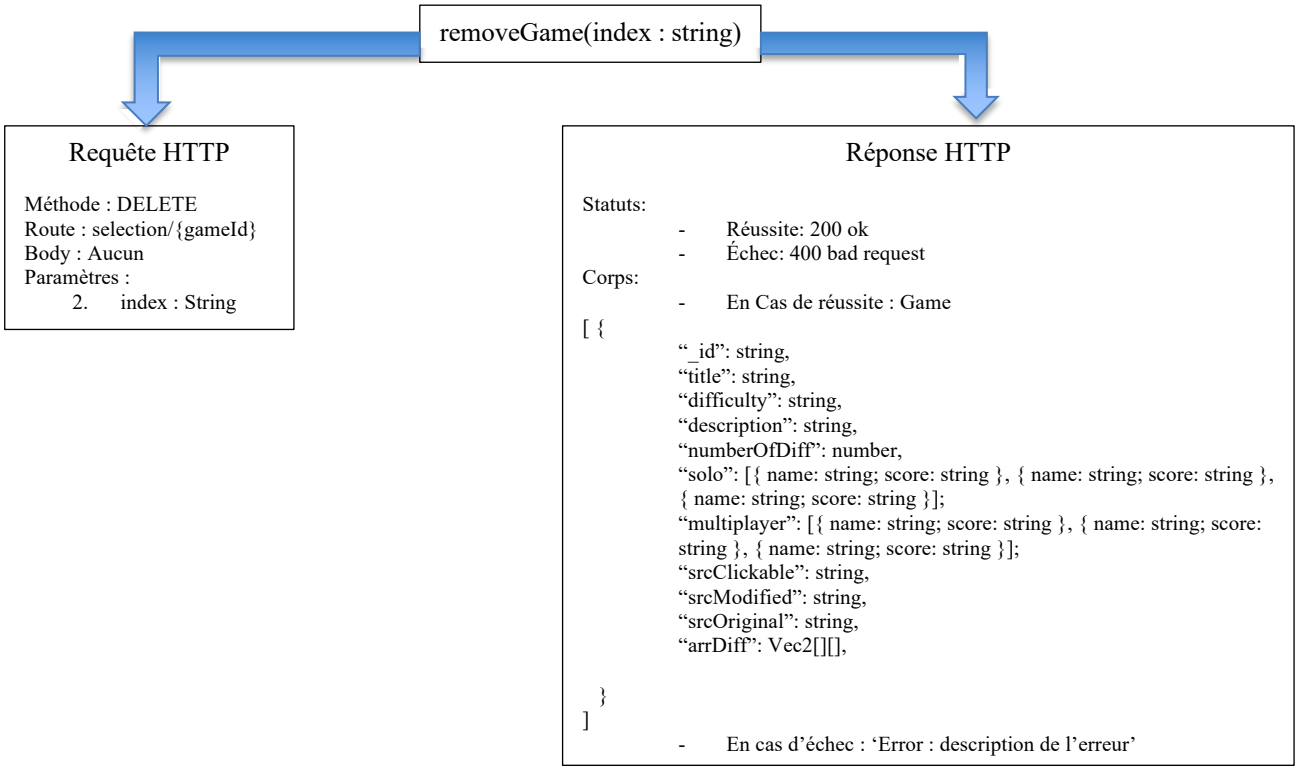
3.6 Schéma de communication HTTP de la méthode patchBestScores



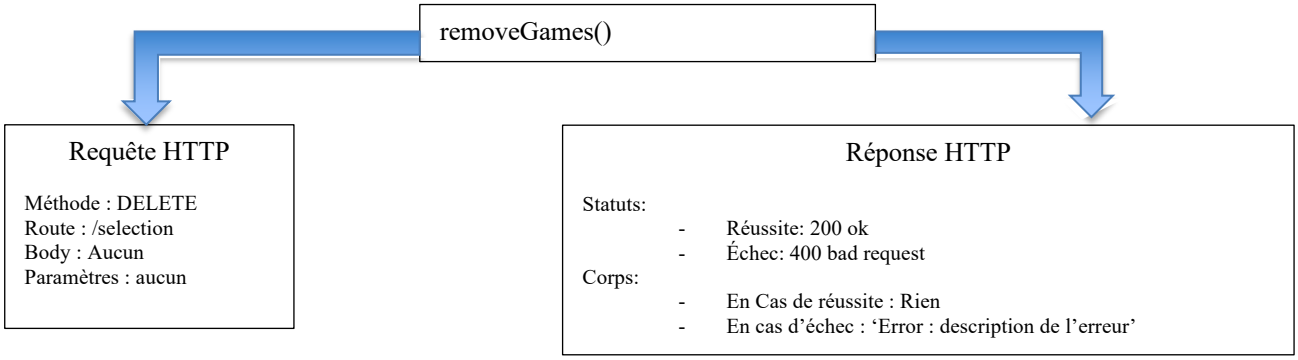
3.7 Schéma de communication HTTP de la méthode createNewGame



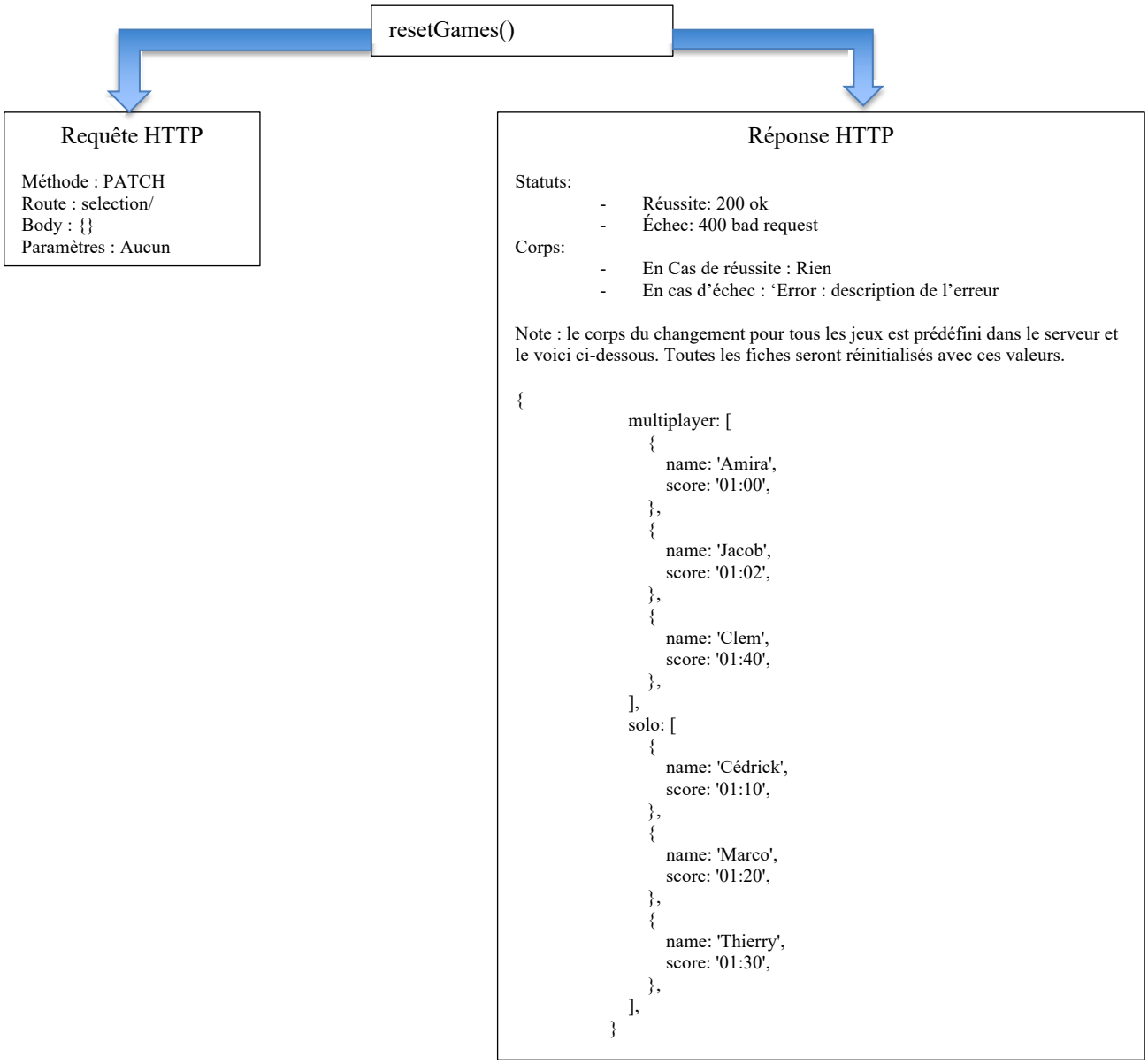
3.8 Schéma de communication HTTP de la méthode removeGame



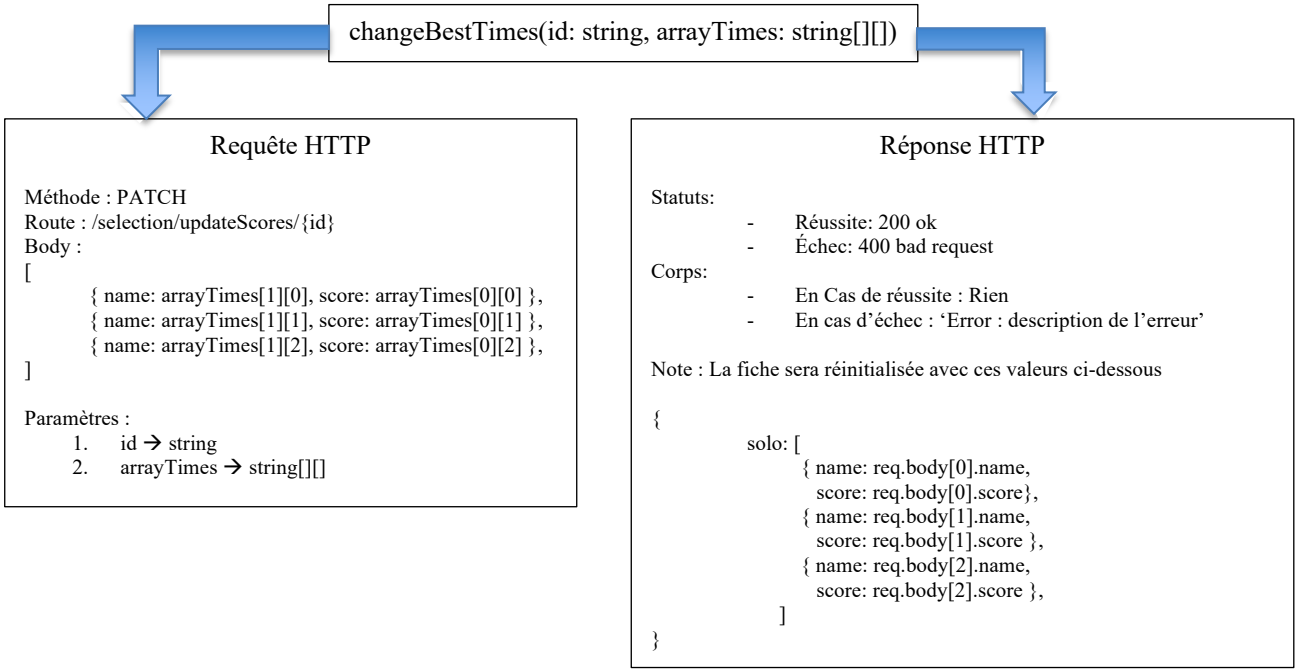
3.9 Schéma de communication HTTP de la méthode removeGames



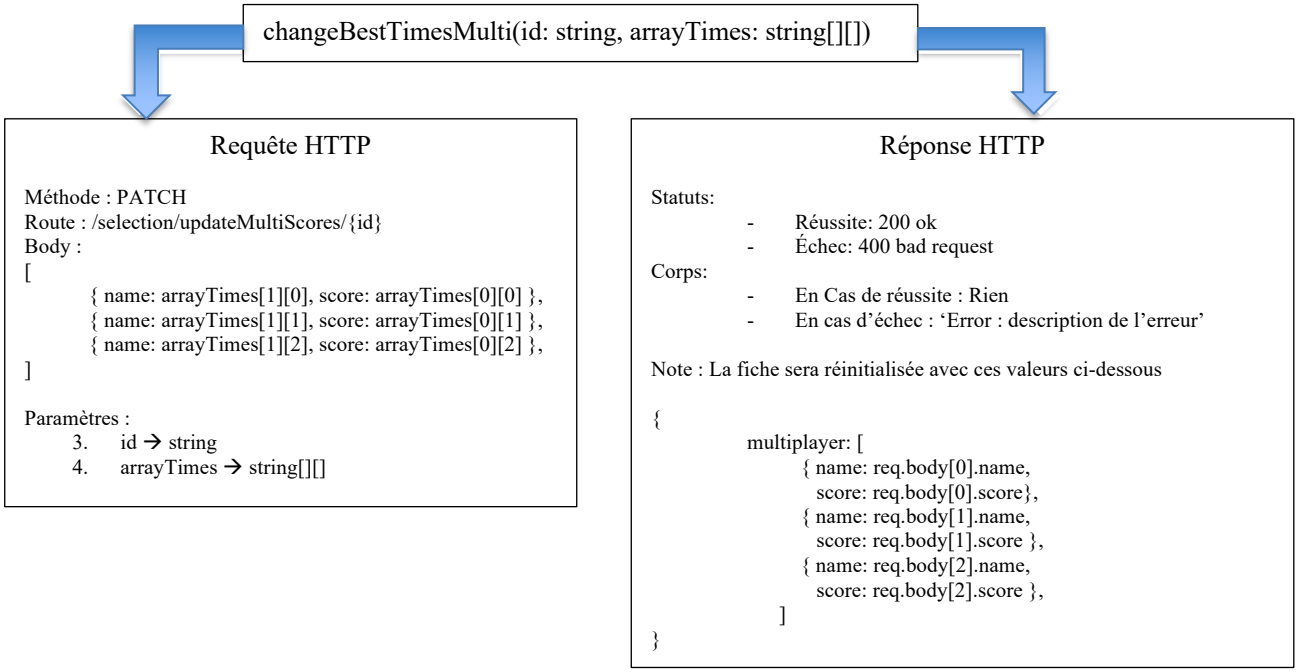
3.10 Schéma de communication HTTP de la méthode resetGames



3.11 Schéma de communication HTTP de la méthode changeBestTimes

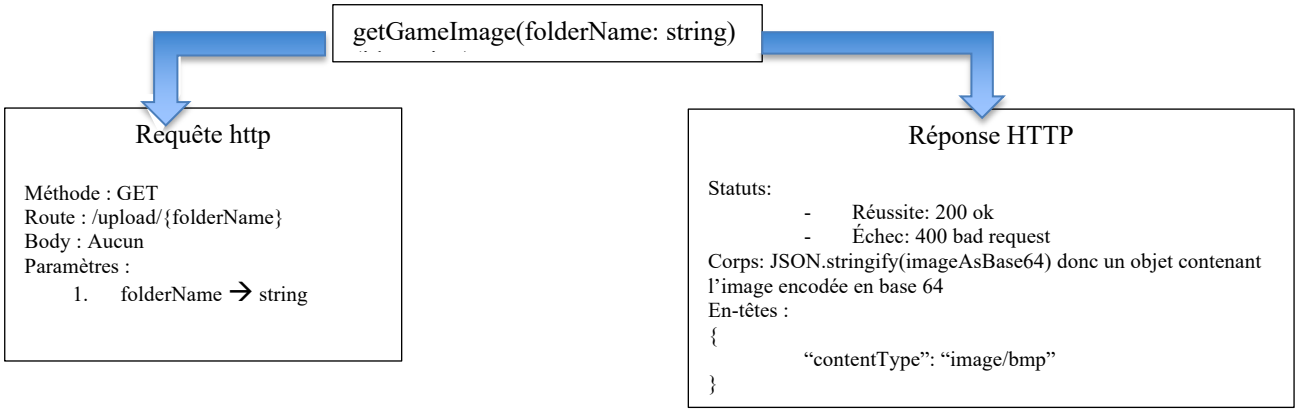


3.12 Schéma de communication HTTP de la méthode changeBestTimesMulti

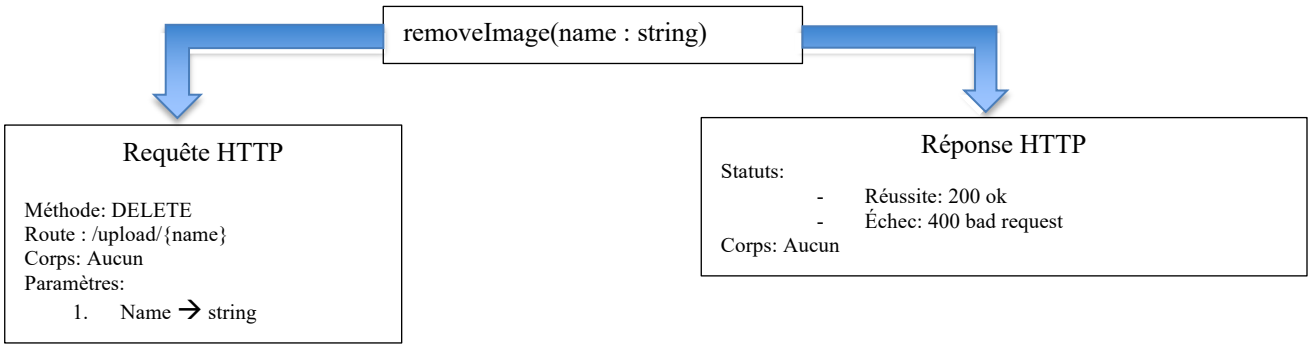


4. Gestion des Images

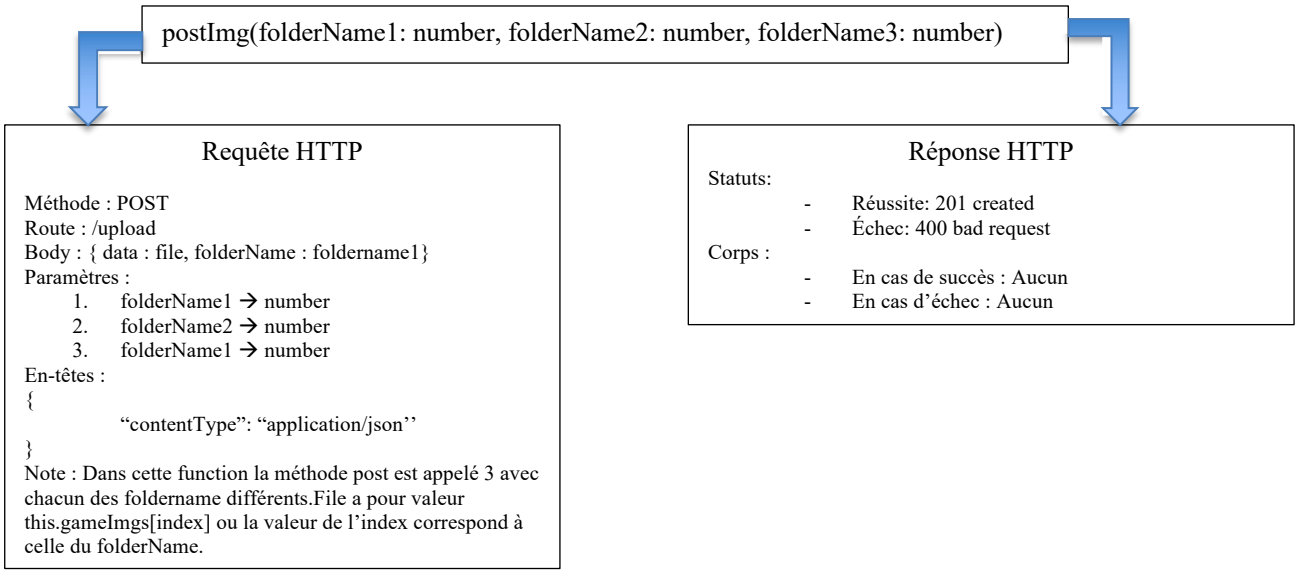
4.1 Schéma de communication HTTP de la méthode getGameImage



4.2 Schéma de communication HTTP de la méthode removeImage

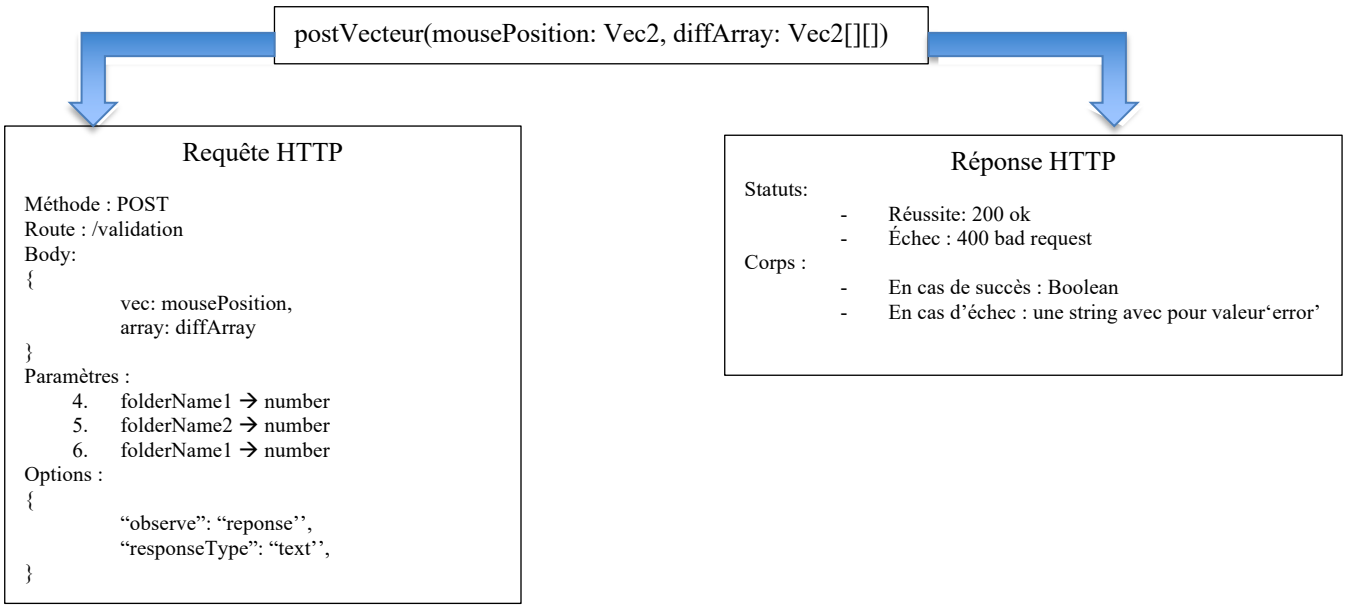


4.3 Schéma de communication HTTP de la méthode postImage



5. Gestion des différences

5.1 Schéma de communication HTTP de la méthode postVecteur



Paquets WebSocket

1. Fonctions

1.1 Tableau de description de l'évènement clock dans la fonction emitTime() du serveur

Nom de l'évènement	'clock'
Source de l'évènement	Serveur
Contenu de l'évènement	New Date().toLocaleTimeString().slice(0, NEG_THREE)
Note	Cette fonction envoie l'heure local en format heure : minute : seconde et permet à la fonction handleSockets d'être rappelé à chaque seconde et ainsi de pouvoir écouter tous les évènements qui se produise chaque seconde

1.2 Tableau de description de la fonctin handleSockets()

Nom de la fonction	handleSockets()
Emplacement	Serveur
Contenu de l'évènement	Tous les évènements que le client envoie au seveur
Note	Cette fonction permet la gestion et l'écoute des évènements envoyés par le client ainsi que l'envoie d'évènement au client

2. Les évènements dans le gestionnaire de socket côté serveur

2.1 Tableau de description de l'évènement connection

Nom de l'évènement	'connection'
Source de l'évènement	client
Contenu de l'évènement	socket: Socket
Note	Cet évènement permet de connecter le socket au serveur et de pouvoir écouter les évènements qu'il émet.

2.2 Tableau de description de l'évènement hintUse

Nom de l'évènement	'hintUse'
Source de l'évènement	Client
Contenu de l'évènement	nHints: number
Note	Cet évènement envoie le numéro d'indice utilisé pour pouvoir mettre à jour le compteur et alerté le chat. Elle émet un évènement au client nommé updateHintCounter et un évènement roomBroadcastMessage pour le compteur.

2.3 Tableau de description de l'évènement noMoreHints

Nom de l'évènement	'noMoreHints
Source de l'évènement	Client
Contenu de l'évènement	Aucun
Note	Cet évènement émet un évènement nommé roomBroadcastMessage pour afficher un message dans le chat.

2.4 Tableau de description de l'évènement joinRoom

Nom de l'évènement	'joinRoom'
Source de l'évènement	Client
Contenu de l'évènement	Aucun
Note	Cet évènement indique que l'utilisateur à joint une salle dans laquelle il sera le seule. Un évènement nommé userNameSolo sera par la suite émit au client.

2.5 Tableau de description de l'évènement joinGameRoom

Nom de l'évènement	'joinGameRoom'
Source de l'évènement	Client

Contenu de l'évènement	Un objet qui a pour valeur :
	{
	room: string,
	invId: string,
	gameId: string,
	hostId: string,
	}

Note	Cet évènement place un utilisateur invité dans une salle de jeu et attribue une couleur à l'hôte ainsi que l'invité . Un évènement nommé joinGame est par la suite envoyé à l'invité.

2.6 Tableau de description de l'évènement hostJoinGameRoom

Nom de l'évènement	'hostJoinGameRoom'
Source de l'évènement	Client

Contenu de l'évènement	Un objet qui a pour valeur :
	{
	room: string,
	hostId: string,
	invId: string,
	hostName: string,
	inName: string,
	gameId: string,
	}

Note	Cet évènement fait rejoindre le socket à une salle et retire l'invité de toute les salles dans lequel il est. Il se retire aussi de ces propres salles et l'hôte émet par la suite deux évènements au client un qui est notPicked à tous les sockets présent dans la salle d'attente et un second qui est gameStarter au socket qui se retrouve dans la salle de jeu qu'il est prêt à entamer.

2.7 Tableau de description de l'évènement joinWaitRoom

Nom de l'évènement	'joinWaitRoom'
Source de l'évènement	Client

Contenu de l'évènement	Un string qui a pour valeur : `wait${id}`
Note	Cet évènement fait rejoindre le socket à la salle d'attente du jeu qu'il a sélectionné. Le serveur émettra par la suite deux évènements, le premier qui est listUsers à la salle d'attente qu'il a rejoint ainsi qu'un deuxième qui se nomme youreHost.

2.8 Tableau de description de l'évènement gameDeleted

Nom de l'évènement	'gameDeleted'
Source de l'évènement	Client
Contenu de l'évènement	Id : string
Note	Cet évènement émet un évènement notPicket à tous les sockets se situant dans la salle d'attente passé en paramètre puisqu'elle vient d'être supprimé.

2.9 Tableau de description de l'évènement joinLTWaitlist

Nom de l'évènement	'joinLTWaitlist'
Source de l'évènement	Client
Contenu de l'évènement	
Note	

2.10 Tableau de description de l'évènement joinLTGameRoom

Nom de l'évènement	'joinLTGameRoom'
Source de l'évènement	Client
Contenu de l'évènement	
Note	

2.11 Tableau de description de l'évènement hostJoinLTGameRoom

Nom de l'évènement	'hostJoinLTGameRoom'
Source de l'évènement	Client
Contenu de l'évènement	
Note	

2.12 Tableau de description de l'évènement playPause

Nom de l'évènement	'playPause'
Source de l'évènement	Client
Contenu de l'évènement	Aucun
Note	Cet évènement renvoie l'évènement playPause au client

2.13 Tableau de description de l'évènement restartVideo

Nom de l'évènement	'restartVideo'
Source de l'évènement	Client
Contenu de l'évènement	Aucun
Note	Cet évènement renvoie l'évènement restartVideo au client

2.14 Tableau de description de l'évènement diffFound

Nom de l'évènement	'diffFound'
Source de l'évènement	Client

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	diffHost: number,
	diffInv: number,
	idHost: string,
	idGame: string,
	}

Note	Cet évènement renvoie deux évènements au client. Un premier nommé updateDiffs au sockets se situant dans la salle avec les scores mis à jour. Le second évènement est roomBroadcastMessage pour affiché l'évènement dans le chat.

2.15 Tableau de description de l'évènement blinkDiff

Nom de l'évènement	'blinkDiff'
Source de l'évènement	Client

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	position: number,
	idHost: string,
	idGame: string,
	}

Note	Cet évènement renvoie au client un nommé blinkDiff avec la position de l'erreur

2.16 Tableau de description de l'évènement userName

Nom de l'évènement	'userName'
Source de l'évènement	Client
Contenu de l'évènement	username: string
Note	Cet évènement associe le nom d'un client à son socket

2.17 Tableau de description de l'évènement roomMessage

Nom de l'évènement	'roomMessage'
Source de l'évènement	Client

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	text: string,
	gameId: string,
	hostId: string,
	}

Note	Cet évènement envoie un évènement nommé roomMessage à la salle de jeu dans lequel le socket se situe. Le message envoyé dépend de la taille de celui-ci.

2.18 Tableau de description de l'évènement leaveWaitRoom

Nom de l'évènement	'leaveWaitRoom'
Source de l'évènement	Client
Contenu de l'évènement	Waitroom : string
Note	Cet évènement enlève le socket du client d'une salle particulière

2.19 Tableau de description de l'évènement roomBroadcastMessage

Nom de l'évènement	'roomBroadcastMessage'
Source de l'évènement	Client

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	text: string,
	gameId: string,
	hostId: string,
	}

Note	Cet évènement envoie un évènement roomBroadcastMessage au client

2.20 Tableau de description de l'évènement refused

Nom de l'évènement	'refused'
Source de l'évènement	Client

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	invId: string,
	gameId: string,
	}

Note	Cet évènement envoie 3 évènements un nommé notPicked, un autre nommé listUsers et un dernier nommé youreHost.

2.21 Tableau de description de l'évènement differenceFound

Nom de l'évènement	'differenceFound'
Source de l'évènement	Client
Contenu de l'évènement	Rien
Note	Envoie deux évènements un nommée roomBroadcastMessage et un second nommé bonusTime à la seule de jeu individuelle dans laquelle le socket se situe.

2.22 Tableau de description de l'évènement errorFound

Nom de l'évènement	'errorFound'
Source de l'évènement	Client

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	idHost: string,
	idGame: string,
	}

Note	Cet évènement renvoie au client un évènement nommé roomBroadcastMessage

2.23 Tableau de description de l'évènement soloErrorFOund

Nom de l'évènement	'soloErrorFound'
Source de l'évènement	Client
Contenu de l'évènement	Rien
Note	Cet évènement émet deux évènements un nommé roomBroadcastMessage ainsi qu'un second nommé penltyTime

2.24 Tableau de description de l'évènement newTime

Nom de l'évènement	'newTime'
Source de l'évènement	Client
Contenu de l'évènement	newTime : string
Note	Cet évènement renvoie au client un évènement nommé roomBroadcastMessage

2.25 Tableau de description de l'évènement giveUp

Nom de l'évènement	'giveUp'
Source de l'évènement	Client

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	idHost: string,
	idGame: string,
	}

Note	Cet évènement supprime le socket de toutes les salles dans lequel il se situe et émet deux évènements. Le premier est playerLeft dans la salle dans laquelle il se situe. Par la suite un roomBroadcastMessage est émis.

2.26 Tableau de description de l'évènement replayTime

Nom de l'évènement	'replayTime'
Source de l'évènement	Client

Contenu de l'évènement	Un objet ayant poour valeur :
	{
	text: string,
	color: string,
	}

Note	Un évènement nommé replayTimeMessage est renvoyé au serveur du socket.

2.27 Tableau de description de l'évènement disconnect

Nom de l'évènement	'disconnect'
Source de l'évènement	Client
Contenu de l'évènement	Reason : io.DisconnectReason

Note	Cet évènement déconnecte le socket de toutes les salles restantes et émet des évènements playerLeft dans toutes les salles dans lesquels il se situe. Par la suite, si le client qui s'est déconnecté était l'hôte il envoie un évènement notPicked a tous les sockets qui étaient dans cet salle d'Attente. S'il n'était pas hôte d'une salle d'attente deux évènements listUsers et youreHost seront envoyés pour raffraîchir les informations de la salle d'attente

2.28 Tableau de description de l'évènement leave

Nom de l'évènement	'roomBroadcastMessage'
Source de l'évènement	Serveur
Contenu de l'évènement	Message : string
Note	Envoie un message au client pour qu'il puisse l'ajouter à l'array de message envoyé

3. Les évènements de socket du côté serveur

3.1 Tableau de description de l'évènement updateHintCounter

Nom de l'évènement	'updateHintCounter'
Source de l'évènement	Serveur
Contenu de l'évènement	nHints: number
Note	Cet évènement envoyé par le serveur permet d'accorder uniquement 3 indices sans plus

3.2 Tableau de description de l'évènement userName

Nom de l'évènement	'userNameSolo'
Source de l'évènement	Serveur
Contenu de l'évènement	userName: string
Note	Cet évènement envoie le nom associé au socket du client

3.3 Tableau de description de l'évènement joinGame

Nom de l'évènement	'joinGame'
Source de l'évènement	Serveur

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	gameId: string,
	hostID: string,
	}

Note	Cet évènement envoie les informations nécessaires pour que le client puisse avoir à la vue de jeu spécifique à son jeu

3.4 Tableau de description de l'évènement notPicked

Nom de l'évènement	'notPicked'
Source de l'évènement	Serveur

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	gameId: string,
	msg: string,
	}

Note	Cet évènement envoie un message au client pour lui indiquer qu'il n'a pas été choisi pour le jeu et de ce fait même le rediriger vers la vue de sélection

3.5 Tableau de description de l'évènement gameStarted

Nom de l'évènement	'gameStarted'
Source de l'évènement	Serveur
Contenu de l'évènement	Un array qui a pour valeur : [hostName, invName, hostId, gameId]
Note	Cet évènement envoie au client les informations nécessaires à la page de jeu de deux individus pour la vue.

3.6 Tableau de description de l'évènement listUsers

Nom de l'évènement	'listUsers'
Source de l'évènement	server
Contenu de l'évènement	Un array d'array de string a pour valeur : [names, idPlayers] → string[][]
Note	Cet évènement envoie dans le premier index un array de tous les noms associé au socket présent dans une salle spécifique et dans le deuxième index l'idnetifiant du socket associé à chaque nom pour que la salle d'Attente puisse afficher les noms.

3.7 Tableau de description de l'évènement youreHost

Nom de l'évènement	'youreHost'
Source de l'évènement	Serveur
Contenu de l'évènement	Rien
Note	Cet évènement indique à un socket s'il a le titre d'hôte.

3.8 Tableau de description de l'évènement gameStartedLT

Nom de l'évènement	'gameStartedLT'
Source de l'évènement	Serveur
Contenu de l'évènement	Un objet ayant pour valeur :
Note	Cet évènement envoie le nom associé au socket du client

3.9 Tableau de description de l'évènement playPause

Nom de l'évènement	'playPause'
Source de l'évènement	Serveur
Contenu de l'évènement	Rien
Note	Cet évènement permet de gérer l'affichage de la reprise vidéo

3.10 Tableau de description de l'évènement restartVideo

Nom de l'évènement	'restartVideo'
Source de l'évènement	Serveur
Contenu de l'évènement	Rien
Note	Cet évènement indique au client de rejouer la reprise vidéo

3.11 Tableau de description de l'évènement updateDiffs

Nom de l'évènement	'updateDiffs'
Source de l'évènement	Serveur

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	diffH: number,
	diffI: number,
	}

Note	Cet évènement envoie au client les scores des deux clients dans la salle mis à jour à la suite d'une différence trouvé

3.12 Tableau de description de l'évènement blinkDiff

Nom de l'évènement	'blinkDiff'
Source de l'évènement	Serveur

Contenu de l'évènement	Un array ayant pour valeur :
	{
	position: number,
	}

Note	Cet évènement envoie la position de la différence trouvé au client

3.13 Tableau de description de l'évènement roomMessage

Nom de l'évènement	'roomMessage
Source de l'évènement	Serveur

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	text: string,
	color: string,
	}

Note	Cet évènement permet d'envoyer au chat un message envoyer par un des clients accompagné de la couleur associé à celui-ci pour que le chat puisse faire la distinction entre les clients.

3.14 Tableau de description de l'évènement bonusTime

Nom de l'évènement	'bonusTime'
Source de l'évènement	Serveur
Contenu de l'évènement	Rien
Note	Cet évènement indique au timer du client d'ajouter du temps au compteur

3.15 Tableau de description de l'évènement penaltyTime

Nom de l'évènement	'penaltyTime'
Source de l'évènement	Serveur
Contenu de l'évènement	Rien
Note	Cet évènement indique au timer du client de retirer du temps au compteur

3.16 Tableau de description de l'évènement relayTimeMessage

Nom de l'évènement	'replayTimeMessage'
Source de l'évènement	Serveur

Contenu de l'évènement	Un array ayant pour valeur :
	{
	text: string,
	color: string,
	}

Note	Cet évènement envoie un mesage au chat pour qu'il ajoute le message à l'array de message et l'Affiche avec la couleur associé

3.17 Tableau de description de l'évènement roomMessage

Nom de l'évènement	'roomMessage
Source de l'évènement	Serveur

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	text: string,
	color: string,
	}

Note	Cet évènement permet d'envoyer au chat un message envoyer par un des clients accompagné de la couleur associé à celui-ci pour que le chat puisse faire la distinction entre les clients.

3.18 Tableau de description de l'évènement playerLeft

Nom de l'évènement	'playerLeft'
Source de l'évènement	Serveur

Contenu de l'évènement	Un objet ayant pour valeur :
	{
	quitterName: string,
	quitterId: string,
	}

Note	Cet évènement indique au client les informations du client qui a quitté