213

Protocole de communication

Version 2.0

Historique des révisions

Date	Version	Description	Auteur
2023-03-16	1.0	Version initiale du document	Léo Banno-Cloutier
2023-03-20	1.1	Renommer Game Data à Game Template	Léo Banno-Cloutier
2023-04-18	2.0	Mise à jour du protocole basé sur le code du sprint 3	Léo Banno-Cloutier

Table des matières

Protocole de communication

1. Introduction

Le présent document décrit le protocole de communication entre les clients et le serveur de notre site web du jeu des différences. La section 2 donne un aperçu haut niveau des différents moyens de communication utilisés par notre application, tandis que la section 3 décrit chaque route HTTP et chaque événement Websocket utilisé dans le projet afin de faire fonctionner les différentes fonctionnalités en détail.

2. Communication client-serveur

Notre projet utilise divers moyens de communications afin d'échanger l'information entre les clients et le serveur. Dû à la nature de jeu de notre site web, la majorité de la communication lors d'une partie doit être fait en temps réel, et nous avons donc choisi d'utiliser les Websockets pour toute la logique pour créer et joindre une partie, le clavardage, pour diffuser le début et la fin d'une partie, les événements lorsqu'un joueur trouve ou ne trouve pas une différence, les indices, le mode triche, l'afficheur de temps et le compteur de différences.

Pour le reste des fonctionnalités, nous avons utilisé les requêtes HTTP afin d'avoir une API simple et facile à comprendre, car l'information n'a pas besoin d'être diffusée en temps réel. HTTP est utilisé pour les informations de la partie, la création et la suppression de jeu, les meilleurs temps des jeux, les constantes de jeu, l'historique des parties jouées et le calcul des différences de jeu.

3. Description des paquets

HTTP

Toutes les routes HTTP ont le préfixe "`/api`", qui va être omis ci-dessous.

Plusieurs types sont réutilisés à de multiples reprises, donc voici la définition de ces types, prises directement du dossier `common` dans la base de code du projet:

```
type GameTemplateId = string;
type InstanceId = string;
type PlayerId = string;
type Username = string;
export enum Difficulty {
    Easy = 'easy',
    Hard = 'hard',
}
interface LeaderboardData {
    player: string;
    time: number;
}
interface Vec2 {
    x: number;
    y: number;
}
enum GameMode {
    Solo = 'solo',
    OneVOne = '1v1',
    TimeLimitSolo = 'timeLimitSolo',
    TimeLimitCoop = 'timeLimitCoop',
}
interface TimeConfigDto {
    totalTime: number;
    hintPenaltyTime: number;
    rewardTime: number;
}
```

```
enum ImageClicked {
    Left = 'left',
    Right = 'right',
}
interface GameTemplate {
    name: string;
    difficulty: Difficulty;
    leaderboardSolo: LeaderboardData[];
    leaderboard1v1: LeaderboardData[];
    firstImage: string;
    secondImage: string;
    nGroups: number;
    _id: GameTemplateId;
}
interface HistoryDto {
    startTime: number;
    duration: number;
    gameMode: GameMode;
    winners: Username[];
    losers: Username[];
    quitters: Username[];
}
class ErrorResponseDto {
    statusCode: number;
    message: string;
    error: string;
}
```

Game Template

Méthode: POST

Route: /game-template

Corps: ServerCreateGameDto

Paramètres: Pas de paramètre

Code(s) de retour: 201, 400

Contenu dans le corps de la réponse: Réponse vide, ErrorResponseDto

Méthode: DELETE

Route: /game-template

Corps: Corps vide

Paramètres: Pas de paramètre

Code(s) de retour: 200

Contenu dans le corps de la réponse: Réponse vide

Méthode: GET

Route: /game-template/length

Description: Retourne le nombre de GameTemplate qui existe dans la base de données

Corps: Corps vide

Paramètres: Pas de paramètre

Code(s) de retour: 200

Contenu dans le corps de la réponse: number

Méthode: GET

Route: /game-template/{id}

Corps: Corps vide

Paramètres: id: string

Code(s) de retour: 200, 400, 404

Contenu dans le corps de la réponse: GameTemplate, ErrorResponseDto, ErrorResponseDto

Méthode: DELETE

Route: /game-template/{id}

Corps: Corps vide

Paramètres: id: string

Code(s) de retour: 200, 400, 404

Contenu dans le corps de la réponse: Réponse vide, ErrorResponseDto, ErrorResponseDto

Méthode: GET

Route: /game-template/page/{page}

Corps: Corps vide

Paramètres: page: string

Code(s) de retour: 200, 400

Contenu dans le corps de la réponse: GameTemplate[], ErrorResponseDto

Méthode: DELETE
Route: /game-template/leaderboard
Corps: Corps vide
Paramètres: Pas de paramètre
Code(s) de retour: 200
Contenu dans le corps de la réponse: Réponse vide

Méthode: DELETE
Route: /game-template/leaderboard/{id}
Corps: Corps vide
Paramètres: id: string
Code(s) de retour: 200, 400
Contenu dans le corps de la réponse: Réponse vide, ErrorResponseDto

Images Differences

Méthode: POST
Route: /images-differences
Corps: CreateImagesDifferencesDto
Paramètres: Pas de paramètre
Code(s) de retour: 201, 400
Contenu dans le corps de la réponse: ServerCreateDiffResult, ErrorResponseDto

History

Méthode: GET
Route: /history
Corps: Corps vide
Paramètres: Pas de paramètre
Code(s) de retour: 200
Contenu dans le corps de la réponse: HistoryDto[]

Méthode: DELETE
Route: /history
Corps: Corps vide
Paramètres: Pas de paramètre
Code(s) de retour: 200
Contenu dans le corps de la réponse: Réponse vide

Time Config

Méthode: GET

Route: /time-config

Corps: Corps vide

Paramètres: Pas de paramètre

Code(s) de retour: 200

Contenu dans le corps de la réponse: TimeConfig

Méthode: PUT

Route: /time-config

Corps: TimeConfig

Paramètres: Pas de paramètre

Code(s) de retour: 200, 400

Contenu dans le corps de la réponse: Réponse vide, ErrorResponseDto

Static Images

Méthode: GET

Route: /images/{filename}

Corps: Corps vide

Paramètres: filename: string

Code(s) de retour: 200, 404

Contenu dans le corps de la réponse: Réponse vide, ErrorResponseDto

Websocket

Il est aussi possible d'avoir la même vue d'ensemble de tous les événements Websocket en lisant le fichier `common/game-manager.ts`, où les interfaces `ClientToServerEventsMap` et `ServerToClientEventsMap` montrent toutes les requêtes possibles du client vers le serveur et vice-versa. Tous les noms de méthodes publics commençant par `on...` signifie un événement envoyé par le serveur au client, tandis que le reste des méthodes décrit les événements créés par les clients.

Nom	Source	Contenu
joinGame	client	`{ gameTemplateId: GameTemplateId; gameMode: GameMode; username: Username; }`
identifyDifference	client	`{ position: Vec2; imageClicked: ImageClicked; }`
messageEvent	client	`{ message: string; sender: string; time: string; }`
approvePlayer	client	`PlayerId`
rejectPlayer	client	`PlayerId`
getWaitingGames	client	`void`
cheatModeEvent	client	`void`
sendHint	client	`void`
startGame	server	`{ nGroups: number; idToUsername: Record<PlayerId, string>; }`
endGame	server	`{ winners: PlayerId[]; losers: PlayerId[]; totalTimeMs: number; }`
differenceFound	server	`{ playerId: PlayerId; }`
differenceNotFound	server	`{ playerId: PlayerId; }`

showError	server	{
		  position: Vec2;
		  imageClicked: ImageClicked;
		}

removePixels	server	{
		  pixels: Vec2[];
		}

timeEvent	server	{
		  timeMs: number;
		}

messageEvent	server	{
		  message: string;
		  sender: string;
		  time: string;
		}

joinRequest	server	{
		  username: Username;
		  playerId: PlayerId;
		}

assignGameMaster	server	void
waitingRefusal	server	void
abortDiscrimination	server	PlayerId
waitingGames	server	GameTemplateId[]
gameTemplateDeletion	server	void

cheatModeEvent	server	{
		  groupToPixels: Vec2[][];
		}

changeTemplate	server	{
		  nextGameTemplateId?: GameTemplateId;
		}

playerLeave	server	{
		  playerId: PlayerId;
		}

receiveHint	server	{
		  rect?: [Vec2, Vec2];
		  angle?: number;
		  givenHints: number;
		  time: UnixTimeMs;
		}