Équipe 110

Pin-ture
Protocole de communication

Version 1.1

Historique des révisions

Date	Version	Description	Auteur
2020-02-04	0.1	première ébauche du document	Jacob Dorais et Duc-Thien Nguyen
2020-02-05	1.0	finalisation et vérification finale	Jacob Dorais
2020-02-07	1.1	Ajout de certains requête et événement manquante	Duc-Thien Nguyen

Table des matières

Protocole de communication

1. Introduction

Ce document a pour but de présenter les interactions entre les clients (léger et lourd) avec le serveur. Son objectif est de décrire les flux de communications pour présenter une idée plus générale de ces interactions.

Le texte commence initialement avec la description des choix des moyens de communication entre les clients(léger et lourd) avec le serveur ainsi qu'une courte description de leurs implantations. Suivra ensuite la description détaillée des différents types de paquets utilisés au sein des protocoles de communication. Finalement, un tableau fera la description des objets complexes utilisés dans les paquets.

2. Communication client-serveur

Pour la communication entre le client et le serveur, il y aura deux types de communications. Nous allons utiliser des sockets et REST API pour les requêtes HTTP. Pour les sockets, nous utiliserons la librairie de socket.io pour le serveur et le socket-client pour le côté client. Cette librairie peut être utilisée avec Typescript, C# ainsi qu'avec Kotlin. Ces sockets seront utilisés pour les communications instantanées, comme la réception et l'envoi de messages pour les chaînes de communications présentes sur l'interface des clients. Par ailleurs, les requêtes seront utilisées pour les cas qui ne sont pas instanciés, par exemple la connexion et la déconnexion d'un usager. Ce sont des requêtes qui ne se feront pas beaucoup donc, ce sont des HTTP.

Pour les connexions, il faut avoir une adresse IP ainsi que des ports disponibles pour le serveur et les clients. Les communications seront faites en TCP/IP. Le type de données sera transféré dans des objets JSON.

3. Description des paquets

Dans cette section, il y aura les descriptions pour les requêtes API REST ainsi que Socket. Chaque section contient la route ou un évènement, avec le type de données envoyées et reçues. Pour les objets plus complexes, la section 4 en décrit le contenu.

3.1 Communication REST API

3.1.1 Communication des comptes

requête	**données envoyé**	**données retourné**	**description**

POST
/account/register
	{
	username: string,
	password: string,
	firstname: string,
	lastname: string,
	avatar: BLOB
	}
		{
		status: int,
		message: string
		}
			Création d'un nouvel utilisateur.

POST
/account/login
	{
	username: string,
	password: string
	}
		{
		status: int,
		message: string
		}
			Connexion d'un utilisateur.

GET
/account/profile/#username
	null	{
		username: string,
		firstname: string,
		lastname: string,
		histories: game[],
		connexions: connection[],
		avatar: BLOB
		}
			Retrouver toutes les informations de profile de l'utilisateur.

GET
/account/profile/other
/#username
	null	{
		username: string,
		avatar: BLOB
		}
			Demander les informations publiques d'un utilisateur.

PUT
/account/profile/picture
	{
	username: string,
	image: BLOB
	}
		{
		status: int,
		message: string
		}
			Requête pour mettre à jour un nouveau photo de profil d'un utilisateur.

3.1.2 Communication des messageries

requête	**données envoyé**	**données retourné**	**description**

POST
/chat/create/channel
	{
	channel_id: int,
	username: string
	}
		{
		status: int,
		message: string
		}
			Requête pour créer un nouveau chaîne de discussion pour le compte d'un utilisateur.

DELETE
/chat/remove/channel
/#id
	null	{
		status: int,
		message: string
		}
			Requête pour enlever une chaîne spécifique pour un utilisateur.

GET
/chat/channelUsers
/#channel_id
	null	{
		users: users[]
		}
			Requête pour avoir la liste des utilisateurs dans une chaîne spécifique.

GET
/chat/message
/#channel_id
	null	{
		messages: message[]
		}
			Requête pour retrouver les anciens messages pour un canal spécifique.

3.1.3 Communication des créations de jeu

requête	données envoyé	données retourné	description

GET
/game/suggestions
	null	{
		drawName: string[]
		}
			Requête pour recueillir des suggestions de mots pour le dessin.

POST
/game/create
	{
	username: string
	password: string,
	game_name: string,
	game_mode: enum,
	image: BLOB,
	privacy: enum
	password:
	string(optional)
	}
		null	Requête pour la création d'une partie par un utilisateur.

GET
/game/active
	null	{
		gameList: game[]
		}
			Requête pour demander des parties en cours.

3.1.3 Communication pour un jeu

requête	données envoyé	données retourné	description

GET
/game/join/#id
	null	{
		game_id: int,
		image: blob,
		users: user[],
		image: blob,
		}
			Requête pour joindre une partie.

POST
/game/histories
	{
	username: string,
	users: users[]
	}
		null	Requête pour mettre à jour les historiques de jeu pour un utilisateur.

POST
/game/end
	{
	username: string,
	winner: string,
	users: users[]
	}
		{
		status: int,
		message: string
		}
			Requête pour signaler la fin de une partie.

3.1.4 Communication autre

requête	données envoyé	données retourné	description

POST
/share/game
	{
	game: game
	}
		{
		status: int,
		message: string
		}
			Requête pour faire le partage de la partie sur des réseaux sociaux.

3.2 communication via Socket

3.2.1 Communication client vers serveur

évènement	données	description

login	{
	username: string
	}
		Événement pour connecter un utilisateur sur les sockets du serveurs.

logout	null	Événement pour déconnecter un utilisateur des sockets du serveur.

chat	{
	username: string,
	channel_id: int,
	content: string
	}
		Événement pour envoyer un message vers le serveur.

draw	{
	game_id: int,
	drawing: Drawing
	}
		Événement pour envoyer les dessins d'un usager vers les autres se trouvant dans la même partie.

game-invite	{
	username: string,
	invited: string
	}
		Événement pour inviter d'autres utilisateurs dans une partie privée.

endgame	{
	status: string
	}
		Événement pour la fin d'une partie.

leavegame	{
	status: string
	}
		Événement envoyer lorsqu'un utilisateur quitte avant la fin d'une partie.

3.2.2 Communication serveur vers client

évènement	**données**	**description**

chat	{
	username: string,
	channel_id: int,
	content: string,
	timestamp: string
	}
		Événement de réception d'un message du serveur.

draw	{
	drawing: Drawing
	}
		Événement de réception des dessins faits par les utilisateurs d'un même partie.

notification	{
	message: string
	}
		Événement de réception de notification du serveur.

game-invite	{
	host: string,
	game_id: int,
	message: string
	}
		Événement pour la réception d'une invitation pour rejoindre une partie.

game-effects	{
	type: string
	}
		Événement pour activer un effet sonore ou visuel.

4. Détail des objets dans les paquets

Game	{
	leaderboard: users[],
	timestamp_start: string,
	timestamp_end: string,
	gameid: int
	}

Connexion	{
	timestamp: string,
	message: string
	}

Message	{
	channel_id: int
	username: string,
	content: string,
	timestamp: string
	}

Drawing	{
	id: int,
	mark: Line[]
	}

Line	{
	thickness: double,
	length: double,
	direction: double,
	color: {int, int, int}
	}

User	{
	username: string,
	profile: BLOB
	}