Équipe 6

Fais-moi un dessin
Protocole de communication

Version 2.0

Historique des révisions

Date	Version	Description	Auteur
2019-09-27	1.0	Rédaction d'une première version	Georges, Sami, Bassam, Syphax, Amine
2019-11-23	2.0	Mise-à-jour du protocole de communication	Équipe 6

Table des matières

Protocole de communication

1. Introduction

Le jeu à développer, « fais-moi un dessin », aura 2 versions. La première version va être mise en marche sur un ordinateur sous Windows 10. La deuxième version va être mis en marche sur une tablette Android. Ainsi, il est alors nécessaire d'avoir un serveur qui réunit ces 2 versions. En effet, un joueur jouant sur la version Android devrait pouvoir jouer avec un joueur jouant sur la version Windows. Dans ce document, la communication entre les différentes entités, c'est-à-dire entre les 2 clients (Android et Windows) et le serveur sera expliqué. Ensuite, on détaillera les paquets envoyés et reçus.

2. Communication client-serveur

Nous avons choisi d'utiliser les requêtes HTTP et les sockets pour effectuer la communication entre les deux clients et le serveur.

Requêtes HTTP

Les requêtes vont être utilisés à travers un *RESTful API*. Ainsi, cela nous donnera la possibilité d'effectuer des requêtes au serveur tel que *GET*, *PUT*, *PATCH* et *DELETE*. Cette utilisation des requêtes HTTP sera utilisé lorsque des écritures dans la base de données du serveur devront être faite pour une utilisation à long terme. Cette manière de communiquer avec le serveur garantit que les opérations seront faites.

Ainsi, le processus d'enregistrement d'un nouvel utilisateur et le processus de connexion d'un utilisateur utilisent des requêtes HTTP pour communiquer au serveur. De même, lorsqu'on veut stocker des données, tel que les résultats d'une partie, le profil des utilisateurs et, enfin, les jeux construis.

Sockets

Les Sockets vont être utilisés lorsqu'on voudrait communiquer avec le serveur de manière spontanée et en temps réel. Ainsi, la communication au serveur est alors facilitée dans les deux sens (serveur au client, client au serveur). Aussi, les données récoltées durant la session de communication faites grâce aux Sockets, seront perdus, sauf si on met en place un processus de sauvegarde, tel que pour les messages du clavardage. En général, dans notre application, toutes les informations des Sockets reliés à l'utilisateur, sont jetés lors de la déconnexion de celui-ci avec le serveur. Cette manière de communiquer avec le serveur garantit que les opérations seront faites plus rapidement pour que les entrées de l'utilisateur soient directement reflétées sur l'application.

Ainsi, ce processus de communication sera utilisé pendant le déroulement d'une partie. En effet, tous les utilisateurs, jouant à cette même partie, pourront voir, en temps réel, le déroulement de celle-ci. Les Sockets seront aussi utilisés dans les canaux de discussion pour envoyer des messages aux autres utilisateurs.

3. Description des paquets

On retrouve les activités suivantes; l'authentification, le clavardage, joindre ou créer une partie, le déroulement d'une partie et, finalement, le tutorial.

Authentification

Http : POST /auth :	Input:	Output:
	username: string;	msg: string;
	password: string;	user: IUser | null;
		tutorialsCompleted: string[];

Clavardage

Http : POST /channel	Input:	Output:
	name: string;	msg: string;

Clavardage rejoindre canal

Http : POST /channel/join	Input:	Output:
	username: string;	msg: string;
	channelName: string;	

Socket - Server:

	Input:
EVENT init chat	username: string;
EVENT enter channel	channelName: string; username: string;
EVENT leave channel	channelName: string; username: string;

EVENT new message	message:
	{
	author: string;
	content: string;
	channel: string;
	created: string;
	};

Socket – Client :

	Input :

EVENT new message	message :
	{
	author : string;
	content : string;
	channel : string;
	created : string;
	};

Joindre / Créer une partie

http :	Input :	Output :
POST /game/list	settings : IgameSettings;	games : Igame[];
POST /game	username : string; settings : IgameSettings;	msg : string;

Socket - Server:

	Input :
EVENT join game	username: string; gameId: string;
EVENT add virtual user	gameId: string; position: int;
EVENT vote start game	gameId: string; username: string;

Socket – Client :

	Input :
EVENT update games	games: IGame[];
EVENT join game	username: string;
EVENT start game	gameId: string; status: GameStatus;

Déroulement d'une partie

http :	Input :	Output :
POST /game/list	settings : IgameSettings;	games : Igame[];
POST /game	username : string; settings : IgameSettings;	msg : string;

Socket - Server

	Input :
EVENT draw image	username: string; gameId: string; image: ImagePart[];
EVENT guess image	username: string; gameId: string; guess: string;

Socket - Client

	Input :
EVENT draw image	gameId: string; image: ImagePart[];
EVENT update game status	gameId: string; status: IGameStatus;

Tutoriel

http :	Input :	Output :
POST /tutorial/complete	username: string; tutorial: string;	msg: string;

Construction d'un jeu

http :	Input :	Output :
GET /image/search/:keyword	keyword: string	images: Image[]
GET /image/:id	id: string	image: Image
POST /image/choose	url: string id: string	image: Image
POST /image/upload	id: string	image: Image
POST /image/convert	id: string	image: Image
POST /image/data	data: ImageStructure id: ImageService	image: Image