Équipe 110

Pïntùré

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 communication 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.

Tout d'abord, pour les communications de protocole HTTP, celui-ci fait des requêtes sans états. Le client peut utiliser ce protocole pour recherches des informations ou pour des modifications. Il peut par exemple utiliser celui-ci pour faire le faire la connexion, la déconnexion, rechercher les informations de profil, etc. Celui-ci est très puissant, par contre, il pose un désavantage. Cette technologie ne permet pas d'envoyer des informations à partir d'un serveur vers un client. Elle est "one way", et c'est avantageux au moment que le client n'a pas toujours besoin d'une connexion avec le serveur puisque lors de la requête, il y a une connexion qui doit s'établir. Pour ces requêtes, elles seront acceptées au port 3000.

Par ailleurs, pour contrer le désavantages du protocole HTTP, il y aura l'utilisation des sockets TCP sur le serveur. Celle-ci permet d'envoyer des informations à partir du serveur. Un très bon avantage de celle-ci est qu'il garde la connexion entre client-serveur, alors le temps entre l'envoie et la réception est beaucoup plus rapide que la communication HTTP. Aussi, les sockets aideront grandement pour garder les sessions des clients ouvertes pendant leur connexion. Des exemples d'utilisation sont: les envois de messages instantanées dans les canaux de discussions ainsi que les canaux de parties, les envois et réceptions des dessins en multijoueur. Le port utilisé pour les sockets est 3001.

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. Pour ce projet, les données ne seront pas cryptés lors des transferts.

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 envoyé et reçu. Pour les objets plus complexes, la section 4 en décrit le contenu.

3.1 Communication REST API

3.1.1 Communication des comptes

3.1.1.1 Route pour enregistrer un nouveau compte

Méthode	**requête**	**données envoyées**	**données retournées**
POST	/account/register	IRegistration	message: string

L'avatar n'est pas nécessaire au moment de la création.
Lors d'une réussite, le statut a le code 201.

3.1.1.2 Route pour la connexion d'un compte

Méthode	**requête**	**données envoyées**	**données retournées**
POST	/account/login	ILogin	{message: string}

Lors d'une réussite, le statut a le code 200.

3.1.1.3 Route pour la modification d'un avatar

Méthode	**requête**	**données envoyées**	**données retournées**
POST	/account/avatar	{username: string, avatar: string}	{message: string}

Lors d'une réussite, le statut a le code 201.

3.1.1.4 Route pour la recherche de l'image de profil

Méthode	**requête**	**données envoyées**	**données retournées**
GET	/account/avatar/#username	N/A	{avatar: string}

Lors d'une réussite, le statut a le code 200.

3.1.1.5 Route pour chercher un utilisateur en ligne avec le nom

Méthode	**requête**	**données envoyées**	**données retournées**	**description**
GET	/account/users/online/#word	N/A	{usernames: string[]}	Demander tout les nom utilisateurs commencent par "word" (non nécessaire)

Lors d'une réussite, le statut a le code 200.

3.1.1.6 Route pour la recherches des informations privés

Méthode	**requête**	**données envoyées**	**données retournées**
GET	/account/user/info/#username	N/A	IInfoUser

Lors d'une réussite, le statut a le code 200.

3.1.2 Communication des canaux de discussions

3.1.2.1 Route pour chercher les messages d'un canal de discussion

Méthode	**requête**	**données envoyées**	**données retournées**
GET	/chat/messages/ #channel_id	N/A	{messages: IMessage[]}

Lors d'une réussite, le statut a le code 200.

3.1.2.2 Route pour chercher un canal de discussion par le nom

Méthode	**requête**	**données envoyées**	**données retournées**	**description**
GET	/chat/channels/search/ #username/#word	N/A	{channel_ids: string[]}	Requête pour avoir la liste des chaînes de messagerie dont le nom commence par "#word".

Lors d'une réussite, le statut a le code 200.

3.1.2.3 Route pour chercher un canal non souscrit par un utilisateur

Méthode	**requête**	**données envoyées**	**données retournées**
GET	/chat/channels/notsub/ #username	N/A	{channel_ids: string[]}

Lors d'une réussite, le statut a le code 200.

3.1.2.4 Route pour chercher un canal souscrit par un utilisateur

Méthode	**requête**	**données envoyées**	**données retournées**	**description**
GET	/chat/channels/sub/ #username	null	{channel_ids: string[]}	Requête pour avoir la liste des chaînes de messagerie dont

				un utilisateur fait parti.

Lors d'une réussite, le statut a le code 200.

3.1.2.5 Route pour joindre un canal de discussion

Méthode	**requête**	**données envoyées**	**données retournées**

PUT	/chat/channels/join/#username/#channel	N/A	{
			message: string
			}

Lors d'une réussite, le statut a le code 200.

3.1.2.6 Route pour quitter un canal de discussion

Méthode	**requête**	**données envoyées**	**données retournées**

DELETE	/chat/channels/leave/#username/#channel	N/A	{
			message: string
			}

Lors d'une réussite, le statut a le code 200.

3.1.3 Communication des parties

3.1.3.1 Route pour joindre/créer un lobby

Méthode	**requête**	**données envoyées**	**données retournées**

POST	/game/lobby/join	IJoinLobby	{
			message: string
			}

Lors d'une réussite, le statut a le code 204.

3.1.3.2 Route pour quitter un lobby / expulser un utilisateur du lobby

Méthode	**requête**	**données envoyées**	**données retournées**

POST	/game/lobby/leave	{
		username: string,
		lobbyname: string,
		isKicked: boolean
		}
			message: string

Lors d'une réussite, le statut a le code 200.

3.1.3.3 Route pour l'invitation d'un utilisateur dans un lobby

Méthode	**requête**	**données envoyées**	**données retournées**
POST	/game/lobby/invite	{username: string, lobyname: string}	message: string

Lors d'une réussite, le status a le code 200.

3.1.3.4 Route pour refuser une invitation

Méthode	**requête**	**données envoyées**	**données retournées**
POST	/game/lobby/invite/refuse	{ username: string, lobyname: string}	message: string

Lors d'une réussite, le status a le code 200.

3.1.3.5 Route pour chercher les lobby actifs selon le mode

Méthode	**requête**	**données envoyées**	**données retournées**
GET	/game/lobby/active/#mode	N/A	IGetLobby[]

Lors d'une réussite, le status a le code 200.

3.1.3.6 Route pour chercher les utilisateurs dans un lobby

Méthode	**requête**	**données envoyées**	**données retournées**

GET	/game/lobby/users/#lobbyname	N/A	{
			usernames: string[]
			}

Lors d'une réussite, le status a le code 200.

3.1.3.7 Route pour chercher les messages d'une lobby

Méthode	**requête**	**données envoyées**	**données retournées**

GET	/game/lobby/messages/#lobbyname	N/A	{
			lobbyName: string,
			username: string,
			content: string,
			}

Lors d'une réussite, le status a le code 200.

3.1.3.8 Route pour chercher les messages d'un jeu

Méthode	**requête**	**données envoyées**	**données retournées**	**description**

GET	/game/arena/messages/#username	N/A	{
			username: string,
			content: string,
			isServer: boolean,
			}
				son propre username doit être envoyé dans la barre d'adresse.

Lors d'une réussite, le status a le code 200.

3.1.3.9 Route pour commencer une partie

Méthode	**requête**	**données envoyées**	**données retournées**

GET	/game/start/#lobbyName	N/A	{
			message: string
			}

Lors d'une réussite, le status a le code 200.

3.1.4 Communication des créations de jeux

3.1.4.1 Route pour chercher une suggestion de dessin

Méthode	**requête**	**données envoyées**	**données retournées**
GET	/game/suggestion	N/A	{suggestion: ISuggestion}

Lors d'une réussite, le status a le code 200.

3.1.4.2 Route pour créer un nouveau jeu

Méthode	**requête**	**données envoyées**	**données retournées**	**description**

POST	/game/new	{
		game: IManuel1
		}
			{
			status: int,
			message: string
			}
				Requête pour la création d'une partie par un utilisateur.

Lors d'une réussite, le 'status" a le code 201.

3.2 Communication via socket

3.2.1 Communication de connexion

3.2.1.1 Évènement de connexion

Évènement: login	Données: N/A

Cet évènement est utilisé pour connecter un utilisateur sur les sockets du serveur.

3.2.1.2 Évènement de déconnexion

Évènement: logout	Données: N/A

Cet évènement est utilisé pour déconnecter un utilisateur des sockets du serveur.

3.2.2 Communication canal de discussion

3.2.2.1 Évènement de transmission de message dans les canaux de discussions

Évènement: chat	Données: {username: str, channelID: str, message: str}

Cet évènement est utilisé pour l'envoi et la réception des messages dans un canal de discussion spécifique.

3.2.2.2 Évènement de création d'un nouveau canal de discussion

Évènement: channel-new	Données: {channelID: str}

Cet évènement est utilisé pour envoyer à tous les clients, sauf le créateur, lors de la création d'un nouveau canal de discussion.

3.2.2.1 Évènement de supprimation de canal de discussion

Évènement: channel-delete	Données: {channelID: str}

Cet évènement est utilisé pour envoyer à tous les clients, sauf un client, lors de la supprimation d'un canal de discussion.

3.2.3 Communication de lobby

3.2.3.1 Évènement transmission de message pour un lobby

Évènement: lobby-chat	Données: {username: str, lobbyName: str, message: str}

Cet évènement est utilisé pour envoyer et recevoir les messages envoyés dans un canal de lobby.

3.2.3.2 Évènement de transmission pour les notifications dans un lobby

Évènement: lobby-notif	Données: { type: str, lobbyName: str, users: str[], private: bool, size: int, mode: str}

Cet évènement est utilisé pour envoyer des notifications reliés au lobby. Dans les données envoyés, il y a un attribut "Type" qui est utilisé pour distinguer entre: création, supprimer, joindre et quitter.

3.2.3.3 Évènement d'invitation pour joindre un lobby

Évènement: lobby-invitation	Données: {type: str, lobbyName: str}

Cet évènement est utilisé pour envoyer au client une invitation pour rejoindre une partie.

3.2.4 Communication des parties

3.2.4.1 Évènement de début d'une partie

Évènement: game-start	Données: N/A

Cet évènement est utilisé pour notifier les utilisateurs visés que la partie commence.

3.2.4.2 Évènement de transmission de messages d'une partie

Évènement: game-chat	Données: {username: str, content: str, isServer: str}

Cet évènement est utilisé pour transférer les messages du canal d'une partie.

3.2.4.3 Évènement de fin de partie

Évènement: game-over	Données: {pts: IPoint[]}

Cet évènement est utilisé pour notifier les utilisateurs en jeu de la fin de la partie.

3.2.4.4 Évènement du nombre d'essais

Évènement: game-guessLeft	Données: {guess: int}

Cet évènement est utilisé pour envoyer aux utilisateurs visés, le nombre d'essai qu'il reste en temps réel.

3.2.4.5 Évènement du nombre de points en temps réel

Évènement: game-points	Données: {point: int}

Cet évènement est utilisé pour envoyer aux utilisateurs visés, le nombre de point cumulé.

3.2.4.6 Évènement pour effacer les dessins sur le canvas

Évènement: game-clear	Données: N/A

Cet évènement est utilisé pour effacer les traits se trouvant sur l'interface de dessin.

3.2.4.7 Évènement pour l'envoie d'informations pour la partie

Évènement: gameplay	Données: IDrawing

Cet évènement est utilisé pour envoyer et recevoir des traits.

3.2.4.8 Évènement pour l'envoie de l'état "prêt"

Évènement: ready	Données: IGameplayReady

Cet évènement est utilisé pour notifier le serveur que la vue d'un utilisateur est prêt pour recevoir des traits.

4. Détail des objets dans les paquets

IMessage	{
	content: string,
	time: string,
	}

IGetLobby	{
	usernames: string[]
	isPrivate: boolean,
	lobbyName: string,
	size: number,
	mode: GameMode,
	}

IGameplayDraw	{
	event: EventType,
	username: string,
	startPosX: number,
	startPosY: number,
	endPosX: number,
	endPosY: number,
	color: string,
	width: number,
	isEnd: boolean,
	format: Format,
	type: Type,
	}

IGameplayChat	{
	event: EventType,
	username: string,
	content: string,
	}

IGameplayReady	{
	event: EventType,
	username: string,
	}

IRegistration	{
	username: string,
	password: string,
	firstname: string,
	lastname: string,
	avatar?: string
	}

ILogin	{
	username: string,
	password: string
	}

IInfoUser	{
	username: string,
	firstname: string,
	lastname: string,
	games: IGame[],
	connections:
	Iconnection[],
	stats: IStats
	}

ICreation	{
	drawing: IDrawingCreator[],
	solution: string,
	clues: string[],
	difficulty: Difficulty,
	displayMode: DisplayMode,
	side: Side,
	}

IDrawingCreator	{
	color: string,
	width: number,
	points: IPoint[],
	}

IGame	{

	mode: string, date: string, players: IPlayer[], }

ISuggestion	{
	drawPng: string,
	drawPxl: IDrawingCreator[];
	object: string,
	}

IJoinLobby	{
	username: string,
	isPrivate: boolean,
	lobbyName: string,
	size?: number,
	mode?: GameMode,
	password?: string,
	}

IDrawing	{
	startPosX: number,
	startPosY: number,
	endPosX: number,
	endPosY: number,
	color: string,
	width: number,
	isEnd: boolean,
	format: Format,
	type: Type,
	}

IPoint	{
	username: string,
	point: number
	}