Équipe 111 (Stack Overwork)

Projet d'évolution logiciel
Protocole de communication

Version 2.3

Historique des révisions

Date	Version	Description	Auteur
2021-02-08	1.0	Ajouts à la section 2	Jaafar Kaoussarani
2021-02-09	1.1	Ajouts aux section 2.1, 2.2 et 2.3	Jaafar Kaoussarani
2021-02-11	1.2	Ajouts aux sections 2.2 et 2.3	Jaafar Kaoussarani
2021-02-12	1.3	Ajouts à la section 3.1	Jaafar Kaoussarani
2021-02-12	1.4	Ajout de la section 3.2	Théo Turell
2021-02-13	1.5	Description des protocoles de la sections 3.2	Théo Turell
2021-02-14	1.6	Révision de la section 3.2	Jaafar Kaoussarani
2021-02-15	1.7	Révision pour l'orthographe Ajout de la section 3.3	Théo Turell
2021-02-19	1.8	Ajouts et corrections à la section 3.3 Révision de la section 3.2	Théo Turell Jaafar Kaoussarani
2021-04-17	2.0	Mise à jour de la section 2.2	Jaafar Kaoussarani
2021-04-17	2.1	Mise à jour de la section 3.2	Théo Turell
2021-04-19	2.2	Fini la section 3.2	Théo Turell
2021-04-20	2.3	Refaire la section 3.3	Théo Turell

Table des matières

Protocole de communication

1. Introduction

Le but de ce document est de présenter sommairement le fonctionnement de la communication entre les clients et le serveur. Pour ce faire, nous présenterons d'abord les détails de cette communication, soit l'API utilisée, la méthode d'hébergement du serveur et le format des données, avant de passer à la description des paquets de données et de leurs contenus.

2. Communication client-serveur

2.1. API de communication

La communication client-serveur se fait à partir de l'API *socket.io*. Cette dernière permet d'établir une connexion client-serveur et d'envoyer et recevoir des données extrêmement facilement. Nous n'utiliserons pas d'API REST, puisque cela n'est pas obligatoire avec *socket.io*. Si l'application avait été d'une portée plus large, nous l'aurons considéré afin de favoriser la séparation de la logique dans nos fichiers, mais puisque la grande majorité de nos fonctionnalités sont interconnectées, il nous semble raisonnable d'inclure toutes nos fonctions côté serveur dans le même point d'accès (*endpoint)*.

Pour se faire, la composante serveur est écrite en TypeScript et établit un point de connexion pour les sockets grâce à Express et le protocole HTTP, qui sont nécessaires à l'initialisation de *socket.io*. Lors du déploiement, le code est compilé en JavaScript afin de pouvoir rouler avec Node.js. Au niveau du client, on établit une connexion avec le serveur en générant une socket avec l'url du *endpoint* du serveur. Quand le serveur reçoit confirmation de l'établissement de la communication, il génère un objet qui contient plusieurs informations relatives au client qui s'est connecté, ainsi que diverses méthodes permettant d'émettre des données et d'en recevoir sur différents canaux d'événement identifiables par leur nom en *string*. Le principe est similaire du côté du client, qui peut aussi émettre et recevoir des données sur ces mêmes canaux d'événement.

Cet API sera utilisé pour toutes les fonctionnalités demandant d'établir une communication entre des clients ou de transférer des données du serveur qui doivent être partagées entre ces clients. On parle ici du système d'authentification, de la messagerie, des fonctionnalités de partie, du dessinage et des joueurs virtuels.

2.2. Hébergement du serveur

Plutôt que d'héberger notre serveur sur l'ordinateur d'un des développeurs, ce qui n'est pas très fiable et pas très pratique pour le déploiement continu de notre serveur, nous utiliserons un service d'hébergement. Nous avons choisi Heroku, puisqu'il permet de déployer notre application extrêmement facilement et rapidement et parce qu'il permet de rouler Node.js directement dans son environnement d'exécution. Le service est initialement gratuit et offre un *dyno*, soit un conteneur Linux isolé qui est exécuté du côté des serveurs de Heroku, du tier gratuit. Cependant, ce tier n'est pas fiable puisqu'il se met en mode *sleep* après trente minutes d'inactivité. Ainsi, nous avons décidé d'utiliser un *dyno* de type *hobby* pendant le développement, qui offre la même performance qu'un *dyno* gratuit, mais sans se mettre en mode *sleep*. De plus, après la livraison du produit final, nous utiliserons temporairement un *dyno* de type *Standard 2x*, ce qui offre deux fois plus de ressources que le type *hobby*. Nous nous servirons des crédits obtenus par le programme *Heroku for GitHub Students* afin de pouvoir utiliser nos *dyno* gratuitement, mais en faisant attention de ne pas dépasser la limite d'heures. Pour ce faire, nous mettrons notre *dyno* au tier gratuit lorsque nous ne faisons pas du développement et nous utiliserons des instances de serveur locales lorsque nous testons l'implémentation d'une nouvelle fonctionnalité. Cela devrait faire en sorte que notre nombre d'heures gratuites reste en dessous du seuil maximal.

2.3 Format des données

Afin d'assurer la compatibilité entre les données des deux clients, ces derniers envoient des données vers le serveur en utilisant le format JSON (JavaScript Object Notation), qui est un moyen de représenter des objets en format *string*. Cette structure est supportée par défaut dans le langage du client lourd, TypeScript, qui est un surensemble syntaxique de JavaScript. Au niveau du client léger, nous utiliserons la librairie Gson de Google, qui permet de convertir des objets Java en notation JSON et vice-versa. Ainsi, en travaillant avec JSON au niveau du serveur, on peut garantir un certain niveau d'uniformité et donc réduire les diverses sources d'erreurs qui viennent avec le développement multi-plateforme.

3. Description des paquets de données

3.1 Format des paquets

De manière générale, l'information contenue dans les paquets de données peut être divisée en deux sections, soit les informations relatives à la communication entre les clients et les données à transférer elles-mêmes. Cette première section contient suffisamment d'informations afin d'assurer que les données se rendront toujours aux bons clients. Par exemple, au niveau d'un message chat envoyé lors d'une partie, cette section contiendra l'identification de la partie, du canal de messagerie et le l'identifiant de l'usager ayant envoyé le message. La deuxième section, quant à elle, contiendra toutes informations supplémentaires, incluant les données à transférer. En reprenant notre exemple, cela inclurait le message lui-même et la date.

Cette structure s'applique à presque tous les types de données que l'on souhaite transférer, qu'il s'agisse de messages, de figures SVG ou d'informations de partie, notamment grâce au format JSON comme indiqué à la section 2.3. Notons cependant que les informations envoyées lors de l'authentification et de l'inscription, bien que représentées par le format JSON, n'auront pas une section détaillant l'information relative à la communication inter-clients, puisque ces informations ne seront jamais transférées à un utilisateur autre que celui qui a entré ces données.

3.2 Paquets au format JSON

La section suivante a été mise au format paysage pour que le tableau conserve un format rendant la lecture plus facile.
Rappelons aussi que toutes les communications sont faites en utilisant la bibliothèque Socket.Io.

Auteur	Nom du socket	Format du JSON	Description
Gestion de la connexion			
Client	connection	socket: Socket	Ce paquet est utilisé pour initialiser une connexion avec un socket et le serveur
Client	manual-disconnect	null	Ce parquet est envoyé lors de la déconnexion manuelle
Client	disconnect	null	Ce paquet est envoyé lorsque le client est fermé
Gestion des comptes			
Client	request-login	username: string	Ce paquet permet de demander si un joueur est déjà connecté avec un profil portant le même nom
Serveur	login-pass	username: string	Ce paquet permet d'autoriser la connexion au profil
Serveur	login-fail	username: string	Ce paquet permet de refuser la connexion au profil
Gestion des dessins			
Client	start-draw	path: string	Ce paquet permet d'envoyer les informations d'un trait qui commence à être tracé. Le paquet reçoit juste le string pour le renvoyer. Il ne va pas le traiter.
Serveur	receive-start-draw	strokeColor: Color strokeWidth: number PrimitiveType: PrimitiveType commandSvg: string points: Point[]	Ce paquet permet d'envoyer à tous les clients un trait qui commence à être tracé.
Client	update-draw	path: string	Ce paquet permet d'envoyer les informations d'un trait en train d'être dessiné. Le paquet reçoit juste le string pour le renvoyer. Il ne va pas le traiter.
Serveur	receive-update-draw	strokeColor: Color	Ce paquet permet d'envoyer les informations d'un trait en train d'être dessiné pour

		strokeWidth: number PrimitiveType: PrimitiveType commandSvg: string points: Point[]	le mettre à jour en direct sur tous les clients.
Client	finish-draw	path: string	Ce paquet permet d'envoyer les informations d'un trait qui vient d'être complété. Le paquet reçoit juste le string pour le renvoyer. Il ne va pas le traiter.
Serveur	receive-finish-draw	strokeColor: Color strokeWidth: number PrimitiveType: PrimitiveType commandSvg: string points: Point[]	Ce paquet permet d'envoyer les informations d'un trait qui vient d'être complété pour les transmettre en direct sur tous les clients.
Client	redraw-canvas	paths: string	Ce paquet permet d'annoncer qu'il y'a des traits qui ont été supprimés
Serveur	receive-redraw-canvas	paths: Path[]	Ce paquet permet d'annoncer qu'il y'a des traits qui ont été supprimés à tous les clients
Client	update-background	color: string	Ce paquet permet d'annoncer que l'utilisateur veut changer la couleur du fond du dessin
Serveur	receive-update-background	color: Color	Ce paquet permet de dire que le dessinateur à changer la couleur de fond du dessin
Gestion des messages			
Client	message	message: ChatMessage	Ce paquet permet de connaître le texte d'un message envoyé, le canal de discussion et l'auteur
Serveur	message-broadcast	message: ChatMessage	Ce paquet permet diffuser le texte d'un message envoyé, le canal de discussion et l'auteur
Client	send-new-channel	channel: ChatChannel	Ce paquet permet de dire à tous qu'un nouveau canal de discussion a été créé
Serveur	receiver-new-channel	newChannel: ChatChannel	Ce paquet permet de recevoir les nouveaux canaux générés
Client	request-init-channels	gameId: string	Ce paquet permet de demander les canaux dans lesquels l'utilisateur est actif

		sender: string	
Serveur	receive-channels-list	channelsList: ChatChannel[] joinedChannelsList: string[]	Ce paquet permet d'envoyer tous les canaux dans lesquels l'utilisateur est présent
Client	user-joined-channel	gameId: string channel_name: string	Ce paquet permet d'indiquer qu'un utilisateur vient de rejoindre un canal.
Client	user-left-channel	gameId: string channel_name: string	Ce paquet permet d'indiquer au canal concerné qu'un utilisateur vient de le quitter
Serveur	receive-delete-channel	gameId: string channel_name: string	Ce paquet permet d'indiquer quels sont les canaux qui viennent d'être supprimés
Gestion de la salle d'attente			
Client	createGame	gameMode,: number difficulty: number host: string password: string	Ce paquet permet d'indiquer qu'un utilisateur créer une partie avec un mode de jeux, une difficulté et, s'il y a lieu, un mot de passe
Serveur	hostJoinLobby	gameId: string	Ce paquet confirme qu'une salle d'attente a été créée et que créateur la rejoint en tant qu'hôte
Client	requestGameWaiting	null	Ce paquet demande toutes les salles d'attente
Serveur	lobbyList	Game[]	Ce paquet permet d'obtenir toutes les salles d'attente
Client	joiningLobby	gameId: string	Ce paquet indique qu'un utilisateur veut joindre une salle d'attente
Serveur	lobbyAcces	isAccepted: boolean	Ce paquet indique que l'utilisateur a réussi et rejoint le lobby
Client	askTheGameData	gameId: string	Ce paquet permet de demander les données de la partie choisie
Serveur	getTheGameData	activeGame: Game teams: Team[]	Ce paquet permet de recevoir les données de la partie choisie
Client	leaveTheLobby	gameId: string	Ce paquet indique qu'un utilisateur a quittés la salle d'attente

		user: string	
Serveur	removeAllPlayers	game: Game message: string	Ce paquet indique que tous les doivent joueurs quitté la salle d'attente
Client	add-bot	gameId: number team: number botName: string	Ce paquet indique que l'hôte d'une partie classique veut ajouter un joueur virtuel dans une équipe
Vote d'expulsion			
Client	vote-kick	gameId: strgin user: string action: string	Ce paquet permet d'initier l'expulsion d'un joueur dans une salle d'attente
Serveur	get-vote-kick	voteKick: VoteKick	Ce paquet permet de mettre à jour l'avancement du vote d'expulsion
Serveur	end-vote-kick	null	Ce paquet termine un vote d'expulsion
Client	vote-kick-in-game	gameId: string user: string action: string	Ce paquet permet d'initier l'expulsion d'un joueur dans une partie en déroulement
Client	kick-player-in-game	gameId: string user: string	Ce paquet permet d'expulser un joueur dans une partie en déroulement
Client	kick-player	gameId: string user: string	Ce paquet permet d'expulser un joueur dans une salle d'attente
Gestion du déroulement de la partie			
Client	loadGame	gameId: string teams: string[][]	Ce paquet permet confirmées les différentes équipes d'une partie avant que celle ci ne commence
Client	ask-active-game-data	gameId: string	Ce paquet permet de demander les données de la partie en cours avec son idée
Serveur	gameStart	null	Ce paquet permet de dire aux clients dans la partie que la partie à démarrer
Serveur	receiveClockTime	secondsRemaining: number	Ce paquet permet de dire aux joueurs humains dans la partie combien de secondes

			il reste dans le tour
Client	ready-to-start	gameId: string	Ce paquet signal permet de signaler que ce client a reçu les données de la partie et est prêt de lancer une partie
Serveur	nextDrawing	userPlaying: string teamPlaying: Team	Ce paquet permet d'indiquer que l'ancien tour est fini et que le nouveau commence
Serveur	wordSuggestions	pairs: Pair[] user: string	Ce paquet permet de proposer trois mots au joueur humain qui s'apprête à dessiner
Client	chooseWord	gameId: string word: string hints: string[]	Ce paquet permet de déterminer quel mot a choisi l'utilisateur pour son tour
Client	attemptWord	gameId: string word: string	Ce paquet permet au joueur d'essayer de deviner le mot
Serveur	correctGuess	null	Ce paquet permet de confirmer au joueur qu'il a réussi à trouver le mot correspondant au dessin
Serveur	incorrectguess	null	Ce paquet permet de dire au joueur que sa tentative est infructueuse
Serveur	drawingTurn	word: string hints: string[] userPlaying: string	Ce paquet permet de dire aux joueurs qui deviennent le mot qu'ils devront dessiner ainsi que les indices associés au mot
Serveur	roundEnd	currentRound: string	Ce paquet permet de dire à tous les joueurs de la partie qu'une manche est terminée
Serveur	receivePoints	users: string[] points: number	Ce paquet permet de mettre à jour les points gagnés par les joueurs
Serveur	endGame	winnerUsers: string[]	Ce paquet permet de signaler la fin de la partie
Serveur	recapGame	turnsRecap: TurnRecap[]	Ce paquet contient tous les dessins faits pendant la partie
Serveur	relaunch	teamRelauching: Team	Ce paquet permet à la seconde équipe de pouvoir deviner pendant une relance (uniquement en mode classique)

Status			
Client	request-status	userList: string	Ce paquet permet de demander les statuts des amis de la liste d'amis d'un joueur
Serveur	receive-status	statuses: Status[]	Ce paquet permet d'obtenir les statuts des joueurs dans la liste d'amis d'un joueur
Client	is-inactive	user: string	Ce paquet permet de dire qu'un utilisateur a mis son statut en tant que: actif
Client	not-inactive	user: string	Ce paquet permet de dire qu'un utilisateur a mis son statut en tant que: inactif
Client	set-do-not-distrub	user: string	Ce paquet permet de dire qu'un utilisateur a mis son statut en tant que: ne pas déranger
Client	unset-do-not-distrub	user: string	Ce paquet permet de dire qu'un utilisateur a retiré son statut: ne pas déranger
Client	is-do-not-distrub	user: string	Ce paquet permet de mettre cet utilisateur en tant que: ne pas déranger
Serveur	receive-is-do-not-disturb	isDoNotDisturb: boolean	Ce paquet permet de retirer cet utilisateur du statut: ne pas déranger
Gestion sociale			
Client	update-friends-list	userToUpdate: string	Ce parquet permet de demander les amis dans la liste d'amis de l'utilisateur
Serveur	receive-update-friends-list	users: string[]	Ce parquet permet de recevoir les amis dans la liste d'amis de l'utilisateur
Client	update-friend-requests	userToUpdate: string	Ce paquet permet de demander les demandes d'amis à l'attention de l'utilisateur
Serveur	receive-update-friend-requests	null	Ce parquet permet de recevoir les demandes d'amis à l'attention de l'utilisateur
Gestion des réactions			
Client	send-reaction	gameId: string username: string reaction: string	Ce paquet permet d'envoyer des réactions à tous les joueurs de la partie
Serveur	receive-reaction	gameId: string username: string reaction: string	Ce paquet permet de recevoir les réactions envoyées par les joueurs de la partie
Autres			

Serveur	receivePopup	message: string	Ce paquet permet de faire apparaître une fenêtre avec un message à l'intérieur

3.3 Objets JSON intervenants dans les protocoles

Cette partie est aussi au format paysage pour que la mise ne place du tableaux soit plus aérée et donc plus simple à lire.
Pour le tableau, la variable gameId se référera à l'identifiant de la partie qui est un nombre. Il peut être stocké sous forme d'un "string", mais aussi d'un "number".

ChatMessage	username: string date: TimeStamp formatedTime: string text: string channel_name: string gameId: string	Ce JSON permet de transmettre toutes les informations pour que les clients puissent afficher un message correctement. "username" correspond à l'utilisateur qui a écrit le message. "text" se réfère au texte envoyé par le message. "channel_name" se réfère au nom du canal donné par son créateur lors de sa création
ChatChannel	gameId: string name: string messages: ChatMessage[] creator: string	Ce JSON permet de transmettre toutes les informations pour que les clients puissent afficher les messages contenus dans un canal. "name" se réfère au nom du canal donné par son créateur lors de sa création
Game	gameId: number isClassic: boolean difficulty: number host: string users: string[] password: string	Ce JSON permet de transmettre toutes les informations en rapport avec une salle d'attente ou une partie.
Player	name: string avatar: string points: number	Ce JSON permet de transmettre toutes les informations en rapport avec un joueur dans une salle d'attente ou une partie. "avatar" est un lien qui renvoie à l'image stockée dans la base de données FireBase.
Team	users: string[] points: number hasBot: boolean	Ce JSON permet de transmettre toutes les informations d'une équipe. (ce format et surtout utilisé lors des parties en mode "Classique"
Bot	name: string personality: number	Ce JSON permet de transmettre les informations se référant à un joueur virtuel

VoteKick	gameId: number user: string votes: number rejections: number	Ce JSON permet de transmettre l'état d'un vote d'expulsion pour que les clients légers puissent l'afficher. "votes" est le nombre de votes pour l'expulsion du joueur mentionner dans la variable "user" "rejections" est le nombre de votes contre l'expulsion du joueur mentionner dans la variable "user"
Path	strokeColor: Color strokeWidth: number type: string commandSvg: string points: Point[] texture: string	Ce JSON permet de transmettre les informations pour tracer un trait. "type" et "texture" sont des variables de l'ancien projet que nous avons choisi comme base. Leurs valeurs ne changent pas et seront identiques qu'importe le trait que l'on dessine. "commandSvg" sont les données devant être données pour un "Path" vectoriel. "points" est un tableau présent pour stocker les premières coordonnées du path. Cette variable est aussi un vestige du projet nous servant de base.
Color	r: number g: number b: number a: number	Ce JSON permet d'encapsuler une couleur au format rgb.
Point	x: number y: number	Ce JSON permet d'encapsuler un point dans un espace en deux dimensions.
Pair	word:string hints:string[]	Ce JSON permet de transmettre le mot que les joueurs doivent deviner. "hints" est un tableau d'indices. Sa taille dépend du nombre d'indices mis par le créateur lors de la conception du mot.
Status	username: string status: number	Ce JSON permet de transmettre le statut d'un joueur. "status" est un chiffre qui change en fonction du statut souhaité par le joueur. 0:non-connecté, 1:actif, 2:inactif, 3:ne pas déranger
TurnRecap	playerWhoDraw: string word: string round: number playersWhoGuess: string[] svg: any[] backgroundColor: Color	Ce JSON permet de transmettre les informations pour afficher l'écran récapitulatif. "word" est le mot censé décrire le dessin "round" est le nombre de la manche à laquelle il a été dessiné "svg" est un tableau de primitive permettant d'afficher le dessin d'un utilisateur