Equipe 102

PolyScrabble
Protocole de communication

Version 1.1

Historique des révisions

Date	Version	Description	Auteur
2022-09-16	1.0	Rédaction initiale	Equipe 102
2022-09-28	1.1	Révision du document	Équipe 102

Table des matières

Protocole de communication

1. Introduction

Dans ce document se trouve une première partie s'occupant d'expliquer notre communication client-serveur expliquant les technologies utilisées pour les communications en détail et expliquant les raisons des choix de ces technologies. Ces explications passeront par la présentation des types de données envoyées et reçues et la cohérence du type de technologie de communication en fonction du paquet.
La deuxième partie de ce rapport servira ici à aller plus en profondeur dans le protocole qui sera utilisé et parlera des différents paquets envoyés du client au serveur et du serveur au client. Nous parlerons plus précisément de la structure, le contenu, et le type de chaque paquet. Pour chacun de ces paquets nous préciserons aussi les routes utilisées pour envoyer et recevoir ces paquets.

2. Communication client-serveur

Pour notre application, nous utiliserons deux protocoles différents; WebSocket et REST API. Pour la partie socket, les librairies utilisées seront la librairie "socket.io" pour le serveur et la librairie socket-client pour notre client. Ces librairies nous permettent de coder en TypeScript et en Dart ce qui est parfait pour notre client lourd et notre client léger.

Dans notre cas, la librairie SocketIO sera utilisée pour toutes nos opérations demandant une connexion bidirectionnelle entre notre client et le serveur. La communication entre socket client et socket serveur se fait à partir du protocole WebSocket.

Nous utiliserons cette technologie pour notre système de jeu, de tour de jeu et d'interaction entre les joueurs notamment au niveau du clavardage entre eux. Toutes les informations sur le statut de la partie et des joueurs eux-mêmes.

Ce choix est dû au fait que notre jeu doit se dérouler principalement en temps réel mais sans requête très fréquente. Effectivement des requêtes sont envoyées seulement en fin de tour de jeu et pour la partie clavardage dans le jeu. Nous avons ici aussi besoin que le serveur parle à notre client pour lui donner l'état de la partie, le score de chacun des joueurs, leur statut (joueur réel/joueur virtuel) ainsi que des alertes pour prévenir le client qu'un joueur a quitté, qu'il a placé un mauvais mot ou qu'il a envoyé un message.

Nous utilisons aussi des requêtes HTTP pour des connexion unidirectionnel entre le client et le serveur. Dans notre cas, ces connexions sont utiles pour la partie connexion des usagers à leur compte personnel. Cela inclut la connexion en elle-même et la déconnection des usagers. Nous utiliserons aussi les requêtes pour rechercher des utilisateurs sur la base de données.

3. Description des paquets

3.1 Communications socketIO

3.1.1 Communication réception statut de partie (vers client)

3.1.1.1 Évènement pour rafraichir le statut du joueur et son stand

Nom de l'évènement	**Données envoyées**
player-update	{player: PlayerObject}

3.1.1.2 Évenèment pour rafraichir le statut du plateau de jeu

Nom de l'évènement	**Données envoyées**
game-update	{game: GameObject}

3.1.1.3 Évènement pour rafraichir la liste des personnes dans chaque partie disponible

Nom de l'évènement	**Données envoyées**
room-players-specs-update	{roomName: string, players: Player[], spectators: Spectators[]}

3.1.1.4 Évènement pour faire connaitre si le createur de la partie devrait pouvait lancer la partie

Nom de l'évènement	**Données envoyées**
start-game-button	{}

3.1.1.5 Évènement pour connaitre le tour de chaque joueur

Nom de l'évènement	**Données envoyées**
turn-status	{isTurnOurs: boolean}

3.1.1.5 Évènement pour faire connaitre si le createur de la partie devrait pouvait lancer la partie

Nom de l'évènement	**Données envoyées**
turn-status	{}

3.1.1.6 Évènement pour faire connaitre au client son statut (spectateur ou joueur)

Nom de l'évènement	Données envoyées
client-status	{isSpectator: boolean}

3.1.2 Communication réception statut du timer local (vers client*)*

3.1.2.1 Évènement pour changer le nom de la personne qui doit jouer

Nom de l'évènement	Données envoyées
display-player-change	{displayChange: string}

3.1.2.2 Évènement pour lancer le compteur local du temps pour chaque tour

Nom de l'évènement	Données envoyées
timer-start	{minutesByTurn: int, currentPlayerPaying: string}

3.1.2.3 Évènement pour arreter le compteur local du temps pour chaque tour

Nom de l'évènement	Données envoyées
timer-stop	{}

3.1.3 Communication réception données sur les salles de jeux disponibles (vers client*)*

3.1.3.1 Évènement pour met à jour une partie de la listes de parties disponibles

Nom de l'évènement	Données envoyées
room-update	{roomName: string, timeTurn: int, isBonusRandom: boolean, players: Player[], spectators: Spectators[]}

3.1.3.2 Évènement pour supprimer une partie de la listes de parties disponibles

Nom de l'évènement	Données envoyées
room-delete	{roomName: string}

3.1.3.3 Évènement pour indiquer une réussite de changement de salle

Nom de l'évènement	Données envoyées
room-change	{page: string}

3.1.4 Communication réception dictionnaires (vers client*)*

3.1.4.1 Évènement pour supprimer une partie de la listes de parties disponibles

Nom de l'évènement	**Données envoyées**
dictionnaries	{dictionnaries: Dict[]}

3.1.4.2 Évènement pour supprimer un dictionnaire

Nom de l'évènement	**Données envoyées**
dictionaries-delete	{dictName: string}

3.1.5 Communication réception des actions du client (vers serveur*)*

3.1.5.1 Évènement nement pour gérer la fin du tour

Nom de l'évènement	**Données envoyées**
turn-finished	{}

3.1.5.2 Évènement pour gérer le click sur le board

Nom de l'évènement	**Données envoyées**
board-click	{coordinateClick: Vec2}

3.1.5.3 Évènement pour gérer le click du bouton exchange

Nom de l'évènement	**Données envoyées**
exchange-click	{}

3.1.5.4 Évènement pour gérer le click du bouton exchange

Nom de l'évènement	**Données envoyées**
annuler-click	{}

3.1.5.5 Évènement pour gérer la sélection des tuiles du stand

Nom de l'évènement	**Données envoyées**
keyboard-selection	{eventString: string}

3.1.5.6 Évènement pour gérer la sélection des tuiles du stand

Nom de l'évènement	**Données envoyées**
keyboard-mouse-manipulation	{eventString: string}

3.1.5.7 Évènement pour gérer la sélection des tuiles du stand avec le clic gauche de la

souris

Nom de l'évènement	**Données envoyées**
left-click-selection	{coordinateXClick: number}

3.1.5.8 Évènement pour gérer la sélection des tuiles du stand avec le clic droit de la souris

Nom de l'évènement	**Données envoyées**
right-click-exchange	{coordinateXClick: number}

3.1.5.9 Évènement pour réinitialiser les tuiles du stand

Nom de l'évènement	**Données envoyées**
reset-tiles-stand	{}

3.1.5.1 Évènement ment pour envoyer les scores dans la base de données

Nom de l'évènement	**Données envoyées**
db-reception	{eventString: string}

3.1.5.1 Évènement pour gérer la sélection du dictionnaire

Nom de l'évènement	**Données envoyées**
dictionnary-selected	{dictionary: mockDict}

3.1.6 Communication réception statut de partie (vers serveur*)*

3.1.6.1 Évènement pour ajouter un autre utilisateur

Nom de l'évènement	**Données envoyées**
new-user	{name: string}

3.1.6.2 Évènement pour ajouter une autre room

Nom de l'évènement	**Données envoyées**
create-room-game	{roomName: string, playerName: string, timeTurn: number, isBonusRandom: boolean, gameMode: string, isLog2990Enabled: boolean, nameOpponent: string, vpLevel: string}

3.1.6.3 Évènement pour joindre une autre room

Nom de l'évènement	Données envoyées
join-room	{roomName: string, playerId: string}

3.1.6.4 Évènement pour ajouter une room

Nom de l'évènement	Données envoyées
list-room	{}

3.1.6.5 Évènement pour transformer la partie en partie solo

Nom de l'évènement	Données envoyées
convert-game–in-solo	{nameOpponent: string, vpLevel: string}

3.1.6.6 Évènement pour quitter la partie

Nom de l'évènement	Données envoyées
leave-game	{}

3.1.5 Communication réception de la déconnection (vers client*)*

3.1.7.1 Évènement pour se déconnecter de la partie

Nom de l'évènement	Données envoyées
disconnect	{}

3.1.7.2 Évènement pour abandonner la partie

Nom de l'évènement	Données envoyées
give-up-game	{}

3.1.8 Communication réception de message de la boite de communication (vers serveur*)*

3.1.8.1 Évènement pour ajouter au chat le nouveau message du client

Nom de l'évènement	Données envoyées
new-message-client	{inputclient: string}

3.2 Communications REST API

3.2.1 Communication comptes utilisateurs

3.2.1.1 Route pour créer son compte utilisateur

Méthode	**Requête**	**Données envoyées**	**Données retour**	**Message de statut retour**

POST	/signup	{
		"email": string,
		"name": string,
		"password": string,
		}
			{
			“data”: User,
			“message”: “signup”,
			}
				Code 201

3.2.1.2 Route pour se connecter à son compte utilisateur

Méthode	**Requête**	**Données envoyées**	**Données retour**	**Message de statut retour**

POST	/login	{
		"email": string,
		"password": string,
		}
			{
			“data”: User,
			“message”: “logged in”,
			}
				Code 200

3.2.1.3 Route pour se déconnecter de son compte utilisateur

Méthode	**Requête**	**Données envoyées**	**Données retour**	**Message de statut retour**

POST	/logout	{
		"email": string,
		"password": string,
		}
			{
			“data”: User,
			“message”: “logged in”,
			}
				Code 200

3.2.1.4 Route pour obtenir la liste de tout les utilisateurs

Méthode	**Requête**	**Données envoyées**	**Données retour**	**Message de statut retour**

GET	/users	N/A	{
			“data”: User[],
			“message”: “findAll”,
			}
				Code 200

3.2.1.4 Route pour obtenir un utilisateur précis

Méthode	**Requête**	**Données envoyées**	**Données retour**	**Message de statut retour**

GET	/users/:id	N/A	{
			“data”: User,
			“message”: “findOne”,
			}
				Code 200

3.2.1.5 Route pour mettre à jour le profil d’un utilisateur

Méthode	**Requête**	**Données envoyées**	**Données retour**	**Message de statut retour**

PUT	/users/:id	{
		"email": string,
		"password": string,
		}
			{
			“data”: User,
			“message”: “updated”,
			}
				Code 200

3.2.1.5 Route pour supprimer le profil d’un utilisateur

Méthode	**Requête**	**Données envoyées**	**Données retour**	**Message de statut retour**

DELETE	/users/:id	{
		"email": string,
		"password": string,
		}
			{
			“data”: User,
			“message”: “deleted”,
			}
				Code 200