Équipe 102

Fais-moi un dessin
Protocole de communication

Version 1.4

Historique des révisions

Date	Version	Description	Auteur
2021-02-03	1.0	Première version de la section 1	Ming Xiao Yuan
2021-02-15	1.1	La communication client vers le serveur est complète	Benjamin Boucher-Charest
2021-02-16	1.2	La communication serveur vers le client est complète	Justin Caisse
2021-02-17	1.3	Révision de la communication client vers le serveur	Laura Beaudoin et Vlad Drelciuc
2021-02-18	1.4	Révision de la communication serveur vers le client	Benjamin Boucher-Charest et Vlad Drelciuc

Table des matières

Protocole de communication

1. Introduction

Le présent document, intitulé *Protocole de communication*, a pour but de représenter schématiquement les communications effectuées entre les composantes de l'application *Fais-moi un dessin*. La section 2 (Communication client-serveur) vise à schématiser notre choix de moyen de communication entre les utilisateurs de cette application et le serveur *Firebase,* tandis que la section 3 (Description des paquets) vise à expliciter les différentes types de paquets utilisées au sein de notre protocole de communication.

2. Communication client-serveur

Les clients communiqueront avec le serveur à l'aide de l'API de Firebase. Il n'y aucune restriction (port ou adresse IP) sur les connexions utilisateurs, puisque le serveur doit être accessible pour n'importe quel utilisateur qui désire jouer. Firebase donne l'option d'utiliser des requêtes HTTP ou WebSocket. Pour ce projet, le protocole WebSocket a été favorisé, car celui-ci est le standard dans les appels de l'API Firebase.

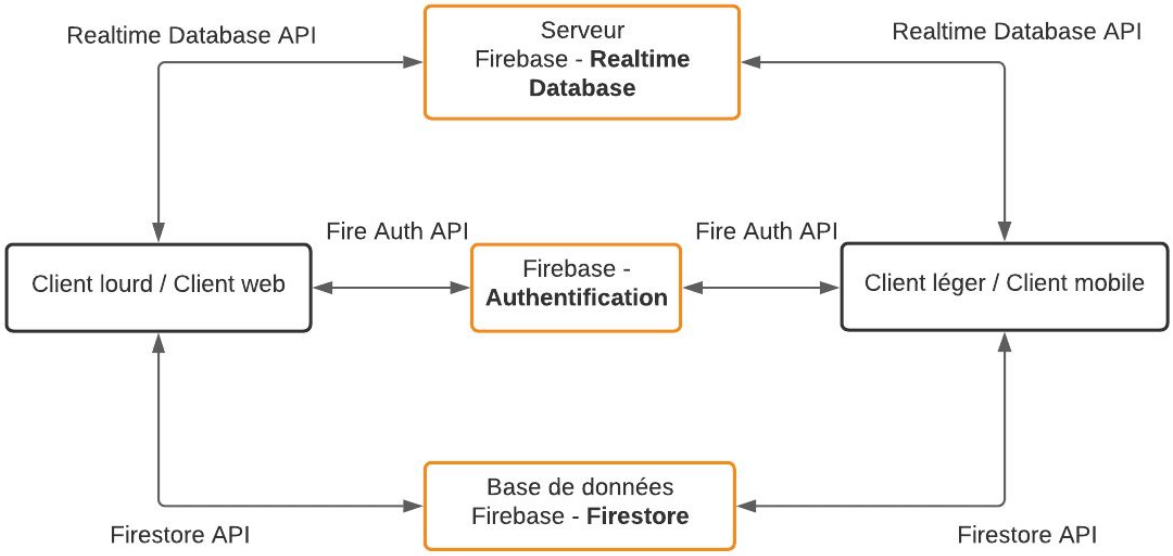

Fig. 1 - Image de la communication entre les clients et Firebase.

Firebase sera utilisé pour le serveur et pour la base de données afin de minimiser le transfert de données entre plusieurs plateformes. Les données des utilisateurs seront authentifiées avec Fire Authentification, le clavardage se fera dans le Realtime Database de Firebase et finalement, l'historique des messages ainsi que les paires mot-image seront dans le Cloud Firestore. Ces options sont toutes disponibles avec l'API de Firebase et ces services sont tous disponibles sur la plateforme de Firebase.

3. Description des paquets

3.1 Définitions

3.1.1 Protocole TCP/IP

Le protocole de communication utilisé pour la transmission d'information est le TCP/IP. La quantité de données dans un paquet sera limitée par le protocole (environ 1500 bytes de données par paquet), de même que la détection d'erreurs de transmission. Un paquet TCP/IP devrait respecter la forme suivante :

Description	Type de Message	Données

Fig. 2 - Image d'un paquet TCP/IP

3.1.2 Payload

Il s'agit des données du paquet. Dans la communication client vers le serveur, le payload va contenir l'information nécessaire pour soit remplacer un champ, ajouter un nouvel élément, ou supprimer un élément.

Voici les données possible que l'on peut sauvegarder sur Firestore :

Array	Boolean	Geopoint
Map	Null	Number
Reference	String	Timestamp

Fig. 3 - Tableau des données possibles dans une base de données Firestore.

3.1.3 Type

Le type va diriger le fonctionnement du payload lorsqu'il sera reçu dans Firestore. Trois types de paquets Firestore sont offerts pour un paquet :

- *ADDED*

  Ce type est assigné lorsqu'un utilisateur ajoute un élément à un document dans une collection Firestore, sinon il est aussi possible de créer un document. La nomenclature de nommage pour les requêtes du type *ADDED* est la suivante : *add*

- *MODIFIED*

  Ce type est assigné lorsqu'un élément existant dans un document est modifié dans le Firestore. Une modification peut être limitée à un champ ou plusieurs champs à la fois. La nomenclature de nommage pour les requêtes du type *MODIFIED* est la suivante : *set*

- *REMOVED*

  Ce type est assigné lorsqu'un élément doit être supprimé d'un document dans Firestore. La nomenclature de nommage pour le type *REMOVED* est la suivante: *delete*

3.1.4 Éléments du Firestore

- *Collection*
  Une collection contient des documents dans Firestore. Afin d'avoir des documents, il est nécessaire d'avoir une collection existante.
- *Document*
  Un document contient des champs dans Firestore. Il a une limite de 1 Mb pour des données. Afin d'avoir des champs, il est nécessaire d'avoir un document existant. Il regroupe les champs d'un élément spécifique ensemble.
- *Champ*
  Un champ est contenu dans un document Firestore et il peut contenir une valeur, selon les types disponibles - **section 3.1.2**.

3.2 Communication Firestore

3.2.1 Profil Utilisateur

Lire le profil utilisateur de joueur

Nom de la requête	**Description**	**Paramètres**	**Valeur de retour**
getProfile	Le client envoie à Firestore une requête pour le profil d'utilisateur. Celui-ci est retrouvé avec le nom d'utilisateur unique passé en paramètre.	username: String	firstname: String lastname: String avatar: String username: String experience: Number level: Number money: Number badges: Array<string>

Fig. 4 - Description de la requête pour recevoir l'information d'un profil utilisateur dans le Firestore.

3.2.2 Classement de joueur

Lire le classement de joueur

Nom de la requête	**Description**	**Paramètres**	**Valeur de retour**
getLeaderboard	Le client envoie à Firestore une requête pour recevoir le classement des dix meilleurs joueurs de la journée.	gameMode: String currentDate: Date endDate: Date	Array *contenant des objets avec les attributs suivants:* username: String score: Number

Fig. 5 - Description de la requête pour recevoir le classement des meilleurs joueurs dans le Firestore.

Les valeurs possibles de **endDate** sont limités aux valeurs suivantes :

- Une journée avant la date présente
- Une semaine avant la date présente
- Un mois avant la date présente
- Depuis la conception de l'application

Modifier le classement de joueur - *MODIFIED*

Nom de la requête	Description	Paramètres	Valeur de retour
setLeaderboard	Le client envoie à Firestore une requête pour mettre à jour le classement de joueur.	gameMode: String username: String currentDate: Date score: Number	void

Fig. 6 - Description de la requête pour modifier le classement des joueurs dans le Firestore.

3.2.3 Paire mot-image

Ajouter une paire mot-image dans la base de données **-** *ADDED*

Nom de la requête	Description	Paramètres	Valeur de retour
addImagePair	Le client envoie à Firestore une requête pour ajouter une paire de mot-images.	image:Array<SVGElement> hint: Array<String> word: String difficulty: String	void

Fig. 7 - Description de la requête pour modifier le classement des joueurs dans le Firestore.

Obtenir l'image associé avec une paire mot-image

Nom de la requête	Description	Paramètres	Valeur de retour
getImage	Le client envoie à Firestore une requête pour recevoir une image associée à un mot.	imageWord: string	image:Array<SVGElement> hint: Array<String> difficulty: String

Fig. 8 - Description de la requête pour recevoir une paire mot-image dans le Firestore.

Les valeurs possibles de **imageWord** sont limitées à des mots existant dans la base de données. Cette liste est téléchargée lors du début de la partie dans une communication serveur vers le client.

Obtenir la liste de mot pour les paires mot-image disponibles

Nom de la requête	Description	Paramètres	Valeur de retour
getWordList	Le client envoie à Firestore une requête pour recevoir la liste des mots de tous les paires mot-images.	-	words: Array<String>

Fig. 9 - Description de la requête pour recevoir la liste de mots disponible pour une paire mot-image.

3.2.4 Clavardage - Historique

Charger l'historique d'un canal de clavardage

Nom de la requête	Description	Paramètres	Valeur de retour
getChatHistory	Le client envoie à Firestore une requête pour recevoir plus de messages dans l'historique de clavardage entre utilisateurs.	timeStamp: Date channelID: String	Array *contenant des objets avec les attributs suivants :* timeStamp: Date username: String content: String channelID: String

Fig. 10 - Description de la requête pour recevoir plus de messages dans l'historique de clavardage dans le Firestore.

L'attribut **timeStamp** fait référence au timestamp lors de la connexion au canal par l'utilisateur. Par exemple, si l'utilisateur s'est connecté au canal à 10h et qu'il est maintenant 10h15, lorsqu'il remonte tout en haut du chat, par défaut, il voit les messages envoyés depuis 10h. Lorsqu'il clique sur "View history", l'historique complet des messages envoyés avant 10h sera chargé (10h étant le timestamp qu'on envoie au serveur).

Ajout d'un message dans l'historique d'un canal de clavardage **-** *ADDED*

Nom de la requête	Description	Paramètres	Valeur de retour
addChatHistory	Le client envoie à Firestore une requête pour ajouter des éléments dans l'historique de clavardage entre utilisateurs.	timeStamp: Date username: String content: String channelID: String	void

Fig. 11 - Description de la requête pour ajouter des messages dans l'historique d'un canal de clavardage dans le Firestore.

Une liste de **channelID** est contenu dans le profil d'un utilisateur du côté backend de l'application. Lorsqu'un message est envoyé, il est nécessaire de spécifier l'identifiant du canal (channelID) de clavardage afin d'apporter les bonnes modifications à son historique.

3.2.5 Clavardage - Historique

Création d'un canal de clavardage - ADDED

Nom de la requête	Description	Paramètres	Valeur de retour
addChannel	Le client envoie à Firestore une requête pour créer un nouveau canal de clavardage et ajouter un nouveau document dans le Firestore.	channelID: String channelName: String channelType: String	void

Fig. 12 - Description de la requête pour ajouter un nouveau canal dans le Firestore en forme de document.

Les valeurs possible du paramètre **channelType** sont:

- *private*

  Le canal n'affiche pas sur la liste publique de canal - seulement ceux qui connaissent le nom du canal peuvent le rejoindre.

- *public*

  Le canal est visible pour n'importe quel utilisateur qui ouvre la liste de canal.

Rejoindre un canal de clavardage - *ADDED*

Nom de la requête	**Description**	**Paramètres**	**Valeur de retour**
addUserToChannel	Le client envoie à Firestore une requête pour rejoindre un nouveau canal de clavardage et ajouter l'utilisateur à la liste de d'utilisateurs dans le canal.	channelID: String username: String	void

Fig. 13 - Description de la requête pour rejoindre un canal existant dans le Firestore.

Quitter un canal de clavardage - *REMOVED*

Nom de la requête	**Description**	**Paramètres**	**Valeur de retour**
deleteUserChannel	Le client envoie à Firestore une requête pour quitter le canal de clavardage et supprimer l'utilisateur de la liste de membres.	channelID: String username: String	void

Fig. 14 - Description de la requête pour quitter un canal existant dans le Firestore.

Supprimer un canal de clavardage - REMOVED

Nom de la requête	**Description**	**Paramètres**	**Valeur de retour**
deleteChannel	Le client envoie à Firestore une requête pour supprimer le canal de clavardage.	channelID: String	void

Fig. 15 - Description de la requête pour supprimer un canal existant dans le Firestore.

3.2.6 Dessiner un dessin

Ajouter un trait - *ADDED*

Nom de la requête	Description	Paramètres	Valeur de retour
addStroke	Le client envoie à Firestore une requête pour ajouter un nouveau trait dans la base de données durant la partie.	path: SVGElement lobbyID: string	void

Fig. 16 - Description de la requête pour ajouter un trait dans le dessin durant une partie dans le Firestore.

Effacer un trait - *REMOVED*

Nom de la requête	Description	Paramètres	Valeur de retour
deleteStroke	Le client envoie à Firestore une requête pour effacer un trait de dessin.	path: SVGElement lobbyID: string	void

Fig. 17 - Description de la requête pour supprimer un trait d'un dessin durant une partie dans le Firestore.

3.2.7 Lobby de jeu

Rejoindre un lobby - *MODIFIED*

Nom de la requête	Description	Paramètres	Valeur de retour
addUserLobby	Le client envoie à Firestore une requête pour rejoindre le lobby de clavardage et modifier la liste de membres.	lobbyID: String username: String	void

Fig. 18 - Description de la requête pour ajouter un utilisateur dans un lobby avant une partie dans le Firestore.

Quitter un lobby - *REMOVED*

Nom de la requête	Description	Paramètres	Valeur de retour
deleteUserLobby	Le client envoie à Firestore une requête pour quitter un lobby d'une partie et supprimer l'utilisateur du lobby.	lobbyID: String username String	void

Fig. 19 - Description de la requête pour quitter un lobby durant/avant une partie dans le Firestore.

Créer un lobby - *ADDED*

Nom de la requête	Description	Paramètres	Valeur de retour
addLobby	Le client envoie à Firestore une requête pour créer un nouveau lobby et ajouter un nouveau document dans le Firestore.	lobbyID: String lobbyType: String ownerUsername: String	void

Fig. 20 - Description de la requête pour créer un lobby dans le Firestore en forme de document

Les valeurs possible du paramètre **lobbyType** sont:

- private

  Le lobby n'affiche pas sur la liste publique de lobby - seulement ceux qui connaissent le nom du lobby peuvent le rejoindre.

- public

  Le lobby est visible pour n'importe quel utilisateur qui ouvre la liste de lobby.

Supprimer un lobby - *REMOVED*

Nom de la requête	Description	Paramètres	Valeur de retour
deleteLobby	Le client envoie à Firestore une requête pour supprimer le lobby dans le Firestore.	lobbyID: String	void

Fig. 21 - Description de la requête pour supprimer un lobby dans le Firestore.

3.2.8 Réaction graphique

Modifier une réaction - *MODIFIED*

Nom de la requête	Description	Paramètres	Valeur de retour
setReaction	Le client envoie à Firestore une requête pour modifier le champ de réaction de l'utilisateur	reaction: String username: String lobbyID: String	void

Fig. 22 - Description de la requête pour modifier une réaction dans le Firestore

3.2.9 Expérience de l'utilisateur

Modifier l'expérience de l'utilisateur - *MODIFIED*

Nom de la requête	Description	Paramètres	Valeur de retour
setExperience	Le client envoie à Firestore une requête pour modifier le contenu du champ d'expérience sur le profil d'utilisateur.	username: String newExperienceValue:Number	void

Fig. 23 - Description de la requête pour modifier le contenu du champ d'expérience sur le profil utilisateur dans Firestore.

Modifier le niveau de l'utilisateur - *MODIFIED*

Nom de la requête	Description	Paramètres	Valeur de retour
setLevel	Le client envoie à Firestore une requête pour modifier le contenu du champ de niveau sur le profil d'utilisateur.	username: String newLevel: Number	void

Fig. 24 - Description de la requête pour modifier le contenu du champ de niveau sur le profil d'utilisateur dans Firestore.

3.2.10 Magasin

Acheter un item / Ajouter de la monnaie virtuel- MODIFIED

Nom de la requête	Description	Paramètres	Valeur de retour
setMoney	Le client envoie à Firestore une requête pour modifier le contenu du champ d'argent	newAmount: Number username: string	void

Fig. 25 - Description de la requête pour modifier le montant de jetons dans le profil d'utilisateur dans Firestore.

Cette requête peut être utilisé pour deux raisons :

- Lorsqu'un utilisateur achète un item.
- Lorsqu'un utilisateur gagne de la monnaie après la partie.

Ajouter un item au profil utilisateur - ADDED

Nom de la requête	Description	Paramètres	Valeur de retour
addItem	Le client envoie à Firestore une requête pour ajouter un item dans le profil d'utilisateur	itemName: String username: String	void

Fig. 26 - Description de la requête pour ajouter un nouvel item dans le profil d'utilisateur dans Firestore.

Charger les items attaché au profil utilisateur

Nom de la requête	Description	Paramètres	Valeur de retour
getItems	Le client envoie à Firestore une requête pour recevoir les items attaché au profil de l'utilisateur	username: String	items: Array<String>

Fig. 27 - Description de la requête pour recevoir la liste d'items que possède le joueur dans Firestore.

3.2.11 Historique des parties de l'utilisateur

Charger l'historique de partie d'un utilisateur

Nom de la requête	Description	Paramètres	Valeur de retour
getUserHistory	Le client envoie à Firestore une requête pour recevoir l'historique des parties de l'utilisateur	username: String	Array *contenants des objets qui contients:* score: Number gameMode: String gameTime: Number

Fig. 28 - Description de la requête pour ajouter un nouvel item dans le profil d'utilisateur dans Firestore.

Ajouter une partie dans l'historique d'un utilisateur

Nom de la requête	Description	Paramètres	Valeur de retour
addGameToHistory	Le client envoie à Firestore une requête pour ajouter un item dans le profil d'utilisateur	username: String score: Number gameTime: Number	void

Fig. 29 - Description de la requête pour ajouter un nouvel item dans le profil d'utilisateur dans Firestore.

3.2.12 Badges

Modifier les badges - *MODIFIED*

Nom de la requête	Description	Paramètres	Valeur de retour
setBadge	Le client envoie à Firestore une requête pour modifier les badges attachés au profil utilisateur.	username: String badgeName: string	void

Fig. 30 - Description de la requête pour modifier les badges dans le profil utilisateur dans le Firestore.

3.3 Realtime Database

3.3.1 Clavardage - Messages

Envoyer un message

Nom de la requête	Description	Paramètres	Valeur de retour
sendMessage	Le client envoie à Realtime Database une requête pour envoyer le message aux autres utilisateurs connectés.	username: String channelID: String timeStamp: Date content: String	void

Fig. 31 - Description de la requête pour envoyer un message dans le Realtime Database.

3.4 Fire Authentification

Authentifier le compte lors d'une connexion

Nom de la requête	Description	Paramètres	Valeur de retour
authenticateAccount	Le client envoie à Fire Authentification une requête pour authentifier la connexion de l'utilisateur	username: String password: String	void

Fig. 32 - Description de la requête pour authentifier un compte dans le Fire Authentification

Ajouter un compte dans le Fire Authentification

Nom de la requête	Description	Paramètres	Valeur de retour
addProfile	Le client envoie à Fire Authentification une requête pour ajouter un profil d'utilisateur.	username: String password: String firstName: String lastName: String avatar: Reference	void

Fig. 33 - Description de la requête pour ajouter un nouveau compte dans le Fire Authentification

3.5 Traitement des erreurs

Lorsqu'une valeur de retour est *void* pour les appels au serveur, les requêtes écoutent toujours pour des erreurs. Les appels sont envoyés dans une *Promise,* alors la valeur de retour est *Promise<void>* pour les requêtes et cela garantit un comportement contrôlé lors d'une erreur. La même chose est ajoutée aux requêtes qui ont une valeur de retour autre que *void*. Ainsi, il est impossible pour l'application d'entrée dans un état indéfinie à la suite d'une valeur de retour erronée.