Équipe 106

Projet 3 : Fais-moi un dessin
Protocole de communication

Version 1.1

Historique des révisions

Date	Version	Description	Auteur
2020-02-03	1.0	Introduction et communication client-serveur	Hubert et Zakari
2020-02-06	1.1	Description des paquets	Hubert et Zakari

Table des matières

Protocole de communication

1. Introduction

Ce document présente les différents protocoles de communication que notre jeu multiplateforme utilisera pour fonctionner adéquatement. Après quelques rencontres en équipe, nous avons choisi d'utiliser l'architecture REST et les sockets pour la communication entre nos clients (lourd et léger) et le serveur. D'abord, les choix que nous avons effectués seront présentés et justifiés dans la section communication client-serveur, puis la description détaillée et le contenu des paquets seront éclaircis dans la section description des paquets.

2. Communication client-serveur

La communication entre le client et le serveur sera faite par deux principales technologies : REST API et les sockets. Dans la majorité des cas, la connexion sera assurée par les sockets, car ils permettent d'établir une discussion stable entre le client et le serveur. REST API sera réservée pour les fonctionnalités de création d'un jeu, de gestion du profil utilisateur et d'accès au classement des modes de jeu. Les sockets seront implémentés à l'aide de la librairie Socket.IO parce qu'elle simplifie la communication entre les clients grâce à la création de canaux personnalisés (rooms). Les sockets seront utilisés pour les fonctionnalités de clavardage et lors du déroulement du jeu.

Le serveur sera conçu sous l'environnement de développement Node.js et les informations importantes qui doivent être gardées en mémoire telles que les profils et les jeux seront sauvegardés dans une base données. MongoDB sera utilisé comme système de gestion de base de données, car il est orienté documents et les requêtes à utiliser pour extraire l'information nécessaire seront relativement simples.

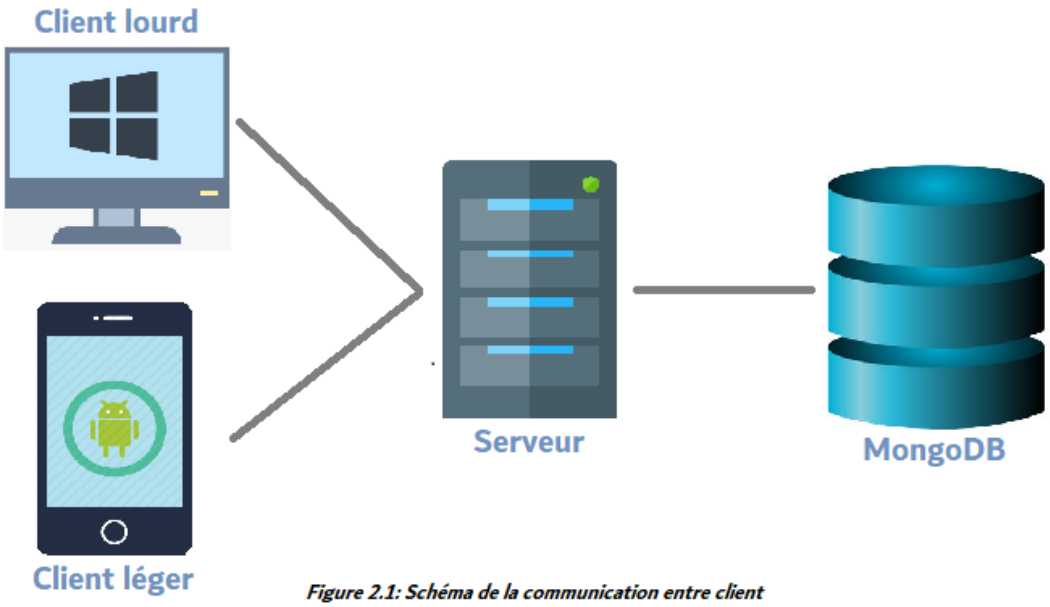

Figure 2.1: Schéma de la communication entre client lourd, client léger, serveur et base de données

Le type de format utilisé pour l'envoi et la réception de données est le format JSON, tant pour les requêtes HTTP que pour les évènements des sockets. Le client et le serveur s'assurent de sérialiser l'objet avant l'envoi et de le désérialiser après réception. Cela permet d'être cohérent au niveau du type de données utilisées et de répliquer des classes dans un langage différent. Plus précisément, une classe avec le même nom avec les mêmes noms d'attributs et les mêmes types d'attributs aura le même format JSON que ce soit en C#, Kotlin ou TypeScript. La traduction du JSON en objet se fera grâce à la librairie Gson en Kotlin et au cadriciel Json.NET en C#. De plus, les JSON se traduisent automatiquement en objet TypeScript.

3. Description des paquets

Dans cette section, les paquets utilisés pour la communication entre les clients et le serveur seront décrits en détail. Tout d'abord, les paquets des requêtes HTTP seront présentés, ensuite les paquets de la communication par sockets avec Socket.IO seront présentés.

3.1 Paquets utilisés pour gérer le profil de l'utilisateur (HTTP)

Pour gérer le profil utilisateur, nous avons décidé de choisir le pseudonyme comme identifiant unique d'un utilisateur. De cette façon, nous pouvons chercher les utilisateurs dans la base de données en ayant uniquement leur pseudonyme. Les paquets qui permettent de gérer le profil utilisateur sont détaillés dans le tableau I.

Tableau I - Paquets concernant la gestion du profil de l'utilisateur

Requête	Chemin (/profile...)	Description	Contenu paquet
POST	/create	Création d'un nouvel utilisateur.	- user / User*
GET	/public	Récupération des informations publiques d'un utilisateur	- username / string - avatar / .svg
GET	/private	Récupération des informations privées d'un utilisateur	- user / User*
GET	/stats	Récupération des statistiques	- stats / Stats*
DELETE	/	Suppression du profil utilisateur associé au username fourni	- username / string
PUT	/update/username	Mise à jour du pseudo	- username / string - newUsername / string
PUT	/update/firstname	Mise à jour du prénom	- username / string - newFirstname / string
PUT	/update/lastname	Mise à jour du nom	- username / string - newLastname / string
PUT	/update/password	Mise à jour du mot de passe	- username / string - newPassword / string
PUT	/update/avatar	Mise à jour de l'avatar	- username / string - avatar / .svg

* Voir annexe pour la description des objets commun au serveur et au client

3.2 Paquets utilisés pour gérer les jeux créés (HTTP)

Pour la création des jeux, le mot à deviner sera l'identifiant unique des jeux créés. De ce fait, on peut récupérer un jeu avec le mot à deviner. De cette façon, il n'y aura pas de redondance dans les dessins des joueurs virtuels. Par exemple, il ne pourra pas y avoir deux fois le dessin d'une orange, car l'utilisateur sera averti qu'il y a déjà un dessin pour ce mot lors de la création. De plus, le temps de la manche est défini par sa difficulté, c'est pour cette raison qu'il n'y a pas ce champ dans le POST au chemin /create. Les paquets permettant de gérer les jeux créés sont détaillés dans le tableau II.

Tableau II - Paquets concernant la gestion des jeux créés

Requête	Chemin (/game...)	Description	Contenu paquet
POST	/create	Création d'un jeu	- game / Game*
GET	/	Récupération de tous les jeux créés	- gamesCreated / Array<Game*>
DELETE	/	Suppression d'un jeu	- word / string

* Voir annexe pour la description des objets commun au serveur et au client

3.3 Paquet utilisé pour consulter le classement (HTTP)

Pour les classements, un joueur pourra consulter le classement top 10 par mode de jeu. Les différents classements sont mis à jour à la fin de chaque partie, donc on a seulement besoin d'une requête qui permet de récupérer le top 10 pour un mode de jeu en question (le mode de jeu est identifié par l'énumération EGameMode, voir annexe). Par exemple, on peut récupérer le top 10 du mode 1 contre 1 en faisant la requête GET au chemin /top/3. Le top 10 récupéré contient le nombre de parties gagnées des 10 meilleurs joueurs. Il contient également le nombre de parties gagnées du demandeur dans le dernier élément de la Map. Les paquets permettant de consulter le classement sont détaillés dans le tableau III.

Tableau III - Paquet concernant le classement

Requête	Chemin (/top...)	Description	Contenu paquet
GET	/	Récupération du top 10 d'un mode de jeu, ainsi que notre propre classement dans ce mode.	- top10 / Map<string (username), number (gamesWon)>

3.4 Paquets utilisés pour la connexion et la déconnexion d'un utilisateur (Socket.IO)

Pour la connexion d'un utilisateur, ce dernier envoie un paquet contenant le nom d'utilisateur et le mot de passe. Si ces identifiants correspondent à un profil dans la base de données, l'identifiant du socket envoyé est synchronisé avec l'utilisateur dans un objet Map qui contient les utilisateurs connectés. Lors de la déconnexion, l'utilisateur et l'identifiant du socket sont supprimés de l'objet Map.

Tableau IV - Paquets pour la connexion/déconnexion d'un utilisateur

ID de l'évènement	Description	Contenu paquet
sign_in	Connexion de l'utilisateur à partir de pseudo et de son mot de passe.	- socketId / string - username / string - password / string
sign_out	Déconnexion de l'utilisateur	- socketId / string

3.5 Paquets utilisés lors de la création d'une partie (Socket.IO)

Pour créer une partie, le joueur hôte décide de certains paramètres tels que le nombre de manches, le mode de jeu et le dévoilement de lettres. Les joueurs sont mémorisés dans une liste associée à la partie du côté serveur et sont ajoutés à celle-ci lorsqu'ils rejoignent la partie. Le joueur hôte est défini par l'indice 0 de cette liste. Une fois la partie créée et un nombre minimum de joueurs (selon le mode), l'hôte de la partie peut démarrer une partie. Les autres joueurs peuvent joindre la partie à l'aide d'un identifiant connu par l'hôte. Les utilisateurs peuvent également quitter la partie. Si l'hôte quitte la partie, le premier joueur l'ayant rejoint devient l'hôte et ainsi de suite si ce dernier quitte la partie. Si l'hôte est seul dans la partie et qu'il la quitte, la partie est supprimée.

Tableau V - Paquets pour la création d'un partie

ID de l'évènement	Description	Contenu paquet
create_match	Création d'un match (par l'hôte uniquement) avec nombre de manches, mode de jeu, dévoilement d'indices et le dévoilement de lettre.	- rounds / number - gameMode / EGameMode* - letterReveal / boolean
join_match	Joindre un match préalablement créé par l'hôte à l'aide d'un identifiant de la partie connu par l'hôte	- gameId: string
leave_match	Quitter un match dans lequel l'utilisateur est déjà présent.	
delete_match	Suppression d'un match lorsque l'hôte est seul dans la partie et qu'il la quitte	

* Voir annexe pour la description des objets commun au serveur et au client

3.6 Paquets utilisés lors du déroulement d'une partie (Socket.IO)

Comme nous avons ajouté les modes de jeu 1 contre 1 et inversé, certaines précisions

sont à faire. Pour les modes réguliers (mêlée générale, sprint solo, sprint coopératif et 1 contre 1), les paquets utilisés seront uniquement ceux du tableau VI. Dans le tableau VII, certains paquets seront modifiés pour le mode inversé.

Note importante : Le joueur courant (isCurrentPlayer) est celui qui dessine dans les modes mêlée générale, sprint solo, sprint coopératif et 1 contre 1. Dans le mode inversé, le joueur courant est le joueur juge. De plus, le mot qui sera envoyé au client sera complet pour le joueur courant (ex: banane) et pour les autres joueurs il sera composé de traits d'union (ex: ------). Lorsqu'une lettre est dévoilée, le mot sera mis à jour pour les autres joueurs (ex: -an--e).

Tableau VI - Paquets concernant le déroulement d'une partie d'un mode régulier

ID de l'évènement	Description	Contenu paquet
start_match	Envoyé par le serveur aux joueurs du salon lorsque l'hôte de la partie décide de débuter la partie.	- isCurrentPlayer / boolean - nbrOfRounds / number (pour l'affichage dans l'interface round 5 of **10**)
end_match	Envoyé par le serveur lorsque la partie est terminée.	- winner / string - finalScores / Map<string (username), number (score)>
start_round	Envoyé par le serveur lorsque le joueur courant a choisi son mot et que la manche peut débuter. Indique au client de lancer son chrono.	- word / string - time / number (in seconds)
end_round	Envoyé par le serveur lorsque la manche est terminé. Indique au client d'arrêter son chrono.	- isCurrentPlayer / boolean - scores / Map<string (username), number (score)>
draw	Envoyé par le joueur dessinateur en temps réel lorsqu'il dessine et transmis à tous les joueurs du salon qui devine.	- stroke / Line*
letter_reveal	Envoyé par le serveur lorsque l'option de dévoilement des lettres est activé.	- wordUpdated / string
hint	Envoyé par le serveur si un des joueurs demande un indice dans le chat.	- hint / string

* Voir annexe pour la description des objets commun au serveur et au client

Tableau VII - Paquets concernant le déroulement d'une partie du mode inversé

ID de l'évènement	Description	Contenu paquet
start_match	Envoyé par le serveur au joueur du	- isCurrentPlayer / boolean

	lobby lorsque l'hôte de la partie décide de débuter la partie. Le joueur juge reçoit une banque de mots et en choisit un.	- nbrOfRounds / number (pour l'affichage dans l'interface round 5 of **10**) - **words: Array <string>****
end_round	Envoyé par le serveur lorsque la manche est terminé. Indique au client d'arrêter son chrono et envoie au joueur juge les dessins de tous les autres joueurs.	- isCurrentPlayer / boolean - scores / Map<string (username), number (score)> - **drawings / Map<string (username), Drawing*>****
top_3	Envoyé par le joueur juge afin de classer les 3 meilleurs dessins.	- top_3 / Array<string (username)>

* Voir annexe pour la description des objets commun au serveur et au client
** Contenu ajouté au paquet original d'une partie dans un mode régulier

3.7 Paquets utilisés pour le clavardage (Socket.IO)

Pour le clavardage, l'utilisateur peut créer un canal de discussion ou en joindre un existant. Une fois dans ce canal, il va charger l'historique de ces messages précédant son arrivée. Puis, il pourra à son tour envoyer un message. Si ce canal de discussion est associé à un jeu, une vérification doit être faite du côté serveur afin de voir si le message envoyé par l'utilisateur correspond au mot à deviner. Le cas échéant, le message n'est pas envoyé aux autres utilisateurs et les points sont mis à jour. À tout moment, un utilisateur peut quitter un canal de discussion (sauf s'il est lié à une partie). Pour la suppression d'un canal de discussion (qui n'est pas lié à une partie), seul le créateur du canal peut le supprimer. À ce moment-là, les autres utilisateurs vont quitter automatiquement le canal de discussion.

Tableau VIII - Paquets utilisés pour le clavardage

ID de l'évènement	Description	Contenu paquet
create_chat_room	Création d'un canal de discussion par un utilisateur. Un nom doit être attribué au canal et il ne doit pas déjà exister dans la liste des canaux déjà présents (chatRoom sert d'identifiant unique au canal de discussion).	- socketId / string - chatRoom / string
join_chat_room	L'utilisateur rejoint un canal de discussion existant.	- socketId / string - chatRoom / string
leave_chat_room	L'utilisateur quitte un canal de discussion duquel il est déjà présent.	- socketId / string - chatRoom / string
delete_chat_room	Suppression d'un canal de discussion. À ce moment-là, l'événement leave_chat_room est appelé pour les autres utilisateurs du canal. Seul le	- socketId / string - chatRoom / string

	créateur du cana de discussion peut appeler cet évèment.	
send_message	Envoi d'un message dans le canal de discussion.	- socketId / string - message / Message*
room_history	Envoi l'historique des messages d'une chaîne.	- chatRoom / string - messages / Array<Message*>

* Voir annexe pour la description des objets commun au serveur et au client

4. Annexe

User : Objet
{
    firstname : string
    lastname : string
    username : string
    password : string
    avatar : .svg
}

Stats : Objet
{
	gamesPlayed: number
	victoryPercentage: number
	averageMatchTime: number (in seconds)
	totalTimePlayed: number (in seconds)
	timeInfos: Array<ConnectTime>
	gameHistory: Array<GameInfos>
}

GameInfos: Objet
{
	timestamp: number
	players: Array<string>
	gameMode: EGameMode
}

ConnectTime : Objet
{
	connection: number (probably a timestamp)
	deconnection: number (probably a timestamp)
}

EDifficulty : énumération
{
	Easy = 0
	Medium = 1
	Hard = 2
}

DrawMode: énumération
{
	Classic = 0
	Random = 1
	Panoramic = 2
	Centered = 3
}

Game : Objet
{
	word : string
	clues : Array<string>
	difficulty : EDifficulty*
	image : .svg
	drawMode : DrawMode
}

```
EGameMode : énumération
{
	freeForAll = 0
	sprintSolo = 1
	sprintCoop = 2
	1vs1 = 3
	inverted = 4
}

Message : Object
{
	author: User
	content: string
}

Line : Object
{
	rgb : Color
	startPos : Position
	endPos: Position
	width: number
	...
}

Color: Object
{
	r: number
	g: number
	b: number
}

Position: Object
{
	x: number
	y: number
	z: number
}

Drawing: Object
{
	lines: Array<Line>
}
```