Équipe 107

DrawHub
Protocole de communication

Version 2.0

Historique des révisions

Date	Version	Description	Auteur
2020-10-02	1.0	Première version remise pour évaluation	Bruno Curzi-Laliberté William Glazer-Cavanagh Thomas Houtart Antoine Martin Simon Robatto-Simard Vincent Tessier
2020-12-02	2.0	Révision suite aux commentaires et aux changements de l'architecture.	Bruno Curzi-Laliberté William Glazer-Cavanagh Thomas Houtart Antoine Martin Simon Robatto-Simard Vincent Tessier

Table des matières

Protocole de communication

1. Introduction

Ce document présente les moyens de communication utilisés par DrawHub. L'objectif est de répertorier l'ensemble des types de messages pouvant être envoyés par un client ou le serveur. Le document sera tenu à jour en fonction des changements en cours de développement.

Le contexte des messages ne sera pas explicité dans le présent document laissant ce rôle aux diagrammes de séquence dans le document d'architecture logicielle, un minimum a tout de même été fait pour présenter l'intérêt de chaque type de message.

Le présent document contient trois sections:

- Le modèle de communication qui est la vision haut niveau du protocole;
- Le processus d'authentification des utilisateurs;
- L'ensemble des paquets supportés par l'application, organisés en catégorie et séparés par mode de communication.

2. Communication client-serveur-base de données

2.1. Client-Serveur

La solution choisie pour le mode de communication est une combinaison de webSockets avec la librairie Socket.IO et d'API REST par protocole HTTPS. Le serveur étant hébergé sur le service Heroku, les communications auront toutes lieu à l'adresse https://draw-hub.herokuapp.com/.

Socket.IO

Plusieurs communications nécessitent une communication à intervalle non régulier et à fréquence élevée entre le serveur et le client (par exemple les parties en cours). Ce type de communication événementielle et bidirectionnel serveur-client se prête mieux aux webSockets.

Socket.IO permet d'utiliser un protocole de communication à base d'événements. L'émetteur peut ainsi envoyer un type d'événement au receveur ainsi que le paquet JSON associé. Le receveur ayant défini la procédure à suivre selon le type d'événement traite le paquet reçu. La librairie abstrait la logique des websockets derrière un API simple et robuste.

Ainsi les sockets de Socket.IO seront utilisés pour la gestion des tous les événements. Donc lors des parties en cours, lors des nouveaux messages et lors des changements dans les Lobbies.

HTTPS

Des requêtes simples allant chercher de l'information vers la base de données (par exemple les préférences profil des utilisateurs) se prêtent mieux à l'architecture REST.

Les communications HTTPS seront uniquement des POST ou GET, le format du POST body est plutôt difficile à généraliser donc chaque cas aura son format. Le "type" des communications Socket.IO est ici remplacé par une route et un type de requête, ex. : un GET à /profile/?user=_.

Ainsi les requêtes HTTPS seront utilisées pour toute l'information à portée permanente sur le serveur requérant une requête à la base de données ou de l'état du serveur. Donc les préférences utilisateurs, les listes de canaux de communications, les listes de trophées, les informations sur une partie en cours se feront par requête HTTPS.

2.2. Serveur-Base de données

La base de données de type MongoDB est aussi hébergée sur le service Azure avec CosmosDB. La communication est triviale et utilise la librairie Mongoose afin de téléverser des documents et de faire des requêtes. Cette solution garantit une communication rapide et sans latence ainsi qu'un MTTF (*mean time to failure*) élevé.

3. Authentification

3.1. L’authentification du client auprès du serveur

1. Le client fait un http GET à /auth/salt/?user=_
2. Le serveur génère un nouveau *salt* temporaire ainsi qu’un permanent et envoie ceux-ci au client
3. Le client ouvre un Socket.IO et envoie son username et *hash(salt_temp* + *hash(salt_perm* + *hash(mot de passe)))*
4. Le serveur accède au mot de passe encrypté sur la base de données *hash(mot de passe)* et calcule le *hash* avec le *salt* temporaire et vérifie si le tout est identique au code envoyé par le client.

3.2. La création d'un nouveau compte

1. Le client envoie une requête POST HTTPS à /createaccount/, cette requête contient le prénom, nom de famille, pseudonyme voulu par le joueur ainsi que son courriel associé et le hash du mot de passe.
2. Le serveur crée une entrée avec ces informations mais hash en double le hash du password avec un salt aléatoire maintenant lié au compte dans une table temporaire et note un code unique associé au compte, valide pour 15 minutes (voir le SRS pour plus de détails sur le fonctionnement point de vue utilisateur).
3. Le serveur envoie un courriel à l’utilisateur contenant le code unique.
4. L’utilisateur demande à confirmer son compte, cette fois-ci par Socket.IO, il envoie une requête de login comme vue précédemment, les étapes 3.1.1 à 3.1.4 sont répétées.
5. Le serveur cette fois-ci ne cherche pas dans la table primaire des hash des mots de passe, mais plutôt dans la table temporaire des entrées valides pour 15 minutes et peut alors valider le compte si les informations sont valides. Si elles le sont, le compte devient permanent, il est retiré de la table temporaire de 15 minutes et l’utilisateur est authentifié.

4. Lobbies et parties

4.1. Création d’un lobby

Voici le flot de messages lorsqu’un client décide de créer un lobby :

1. Le client envoie un message de type CreateLobby au serveur.
2. Le serveur envoie un CreateLobby à tous les utilisateurs.
3. Le serveur envoie un JoinLobbyPlayer à l’utilisateur ayant envoyé la demande de création.
4. Le serveur envoie un UpdateLobby à tous les utilisateurs.

5. Description des paquets Socket.IO

L'objet JSON est le contenu, le type sera utilisé par Socket.IO avec io.emit('type', …) et io.on('type', …);

5.1. Erreurs générales

Tableau x : Contenu d'un message par Socket.IO d'erreurs générales

Message Socket.IO	Description
InternalServerError	Envoyé si le serveur n'est pas en mesure de traiter le message.
NotLoggedInError	Envoyé si l'utilisateur n'est pas authentifié dans le serveur.
PermissionError	Envoyé si le client manque de permissions pour exécuter une requête.

5.2. Communications du clavardage

5.2.1. Serveur → Client

Envoi d'un message

```
ChatMessage
```

Tableau x : Contenu d'un message par Socket.IO de l'envoi d'un message dans un canal de discussion

Nom	Type	Description
_id	String	Le id du message tel que dans la base de données.
author	String	Le pseudonyme du créateur du message.
content	String	Le contenu du message.
timestamp	String	La date du serveur lors de l'envoi du message.
roomName	String	Le nom du canal de discussion dans lequel a été envoyé le message.

Exemple

```
{
  _id: "5v9c79f652w2ecc835638448",
  author: "Bruno",
  content: "Hi Simon!",
  timestamp: "Fri Oct 30 2020 16:37:22 GMT-0400 (Eastern Daylight Time)",
  roomName: "Antoine's Room",
}
```

Canal ajouté

```
CreateRoom
```

Tableau x : Contenu d'un message par Socket.IO de l'ajout du canal de discussion

Nom	Type	Description
roomName	String	Le nom de la salle de clavardage.
username	String	Le pseudonyme de l'utilisateur qui a créé le canal de discussion.

Exemple

```
{
  roomName: "Test room",
  username: "Tony",
}
```

Canal supprimé

```
DeleteRoom
```

Tableau x : Contenu d'un message par Socket.IO de la suppression d'un canal de discussion

Nom	Type	Description
roomName	String	Le nom du canal de discussion.

Exemple

```
{
  roomName: "Antoine's Room",
}
```

Un joueur rejoint un canal

```
JoinRoom
```

Tableau x : Contenu d'un message par Socket.IO lors d'une addition à un canal de discussion

Nom	Type	Description
roomName	String	Le nom du canal de discussion.
username	String	Le pseudonyme de l'utilisateur qui rejoint le canal de discussion.
creator	String	Le pseudonyme de l'utilisateur qui a créé le canal de discussion.

Exemple

```
{
  roomName: "Test room",
  username: "Tony",
  creator: "Simon",
}
```

Un joueur quitte un canal

```
LeaveRoom
```

Tableau x : Contenu d'un message par Socket.IO de la sortie d'un canal de discussion

Nom	Type	Description
roomName	String	Le nom du canal de discussion.
username	String	Le pseudonyme de l'utilisateur qui quitte le canal de discussion.

Exemple

```
{
  roomName: "Test room",
  username: "Tony",
}
```

5.2.2. Client → Serveur

Envoi d'un message

```
ChatMessage
```

Tableau x : Contenu d'un message par Socket.IO de l'envoi d'un message dans un canal de discussion

Nom	Type	Description
content	String	Le contenu du message.
roomName	String	Le nom du canal de discussion dans lequel a été envoyé le message.

Exemple

```
{
  content: "Hi everyone!",
  roomName: "Test Room",
}
```

Canal ajouté

```
CreateRoom
```

Tableau x : Contenu d'un message par Socket.IO de l'ajout d'un canal de discussion

Nom	Type	Description
roomName	String	Le nom du nouveau canal de discussion.

Exemple

```
{
  roomName: "Test Room",
}
```

Canal supprimé

```
DeleteRoom
```

Tableau x : Contenu d'un message par Socket.IO de la suppression d'un canal de discussion

Nom	Type	Description
roomName	String	Le nom du canal de discussion à supprimer.

Exemple

```
{
  roomName: "Test Room",
}
```

Canal rejoint

```
JoinRoom
```

Tableau x : Contenu d'un message par Socket.IO lors d'une addition à un canal de discussion

Nom	Type	Description
roomName	String	Le nom du canal de discussion à rejoindre.

Exemple

```
{
  roomName: "Test Room",
}
```

Canal quitté

```
LeaveRoom
```

Tableau x : Contenu d'un message par Socket.IO lors de la sortie d'un canal de discussion

Nom	Type	Description
roomName	String	Le nom du canal de discussion à quitter.

Exemple

```
{
  roomName: "Test Room",
}
```

5.2.3. HTTP

Historique des messages

```
@GET /room/messagehistory/
```

Tableau x : Paramètres d'une requête http de l'historique des messages d'un canal de discussion

Nom	Type	Description
roomName	String	Le nom du canal dont on veut avoir l'historique des messages.
firstKnownId	String	L'id du premier message connu par le client.

Exemple de réponse

```
{
  messageHistory: [
    {
      _id: "5v9c79f652w2ecc835638448",
      author: "test",
      content: "Hi everyone!",
      timestamp: "Fri Oct 30 2020 16:37:22 GMT-0400 (Eastern Daylight Time)",
      roomName: "Test Room",
    },
    {
      _id: "5f9c9dfc52c2egc381202449",
      author: "simon",
      content: "Comment ça va?",
      timestamp: "Fri Oct 30 2020 16:38:42 GMT-0400 (Eastern Daylight Time)",
      roomName: "Test Room",
    },
  ]
}
```

Liste des canaux de discussion

```
@GET /room/list/
```

Tableau x : Paramètres d'une requête http de la liste des canaux de discussion

Nom	Type	Description
user	String	Le pseudonyme de l'utilisateur dont on veut connaître les canaux de discussion rejoints. Si ce champ n'est pas spécifié, la requête retourne tous les canaux de discussion existants.

Exemple de réponse

```
{
  rooms: [
    { roomName: "Team 7 room", creator: “Tony” },
    { roomName: "CSGO Winners", creator: “Asdf” },
    { roomName: "Project 3 winners", creator: “Poulet” }
  ]
}
```

5.2.4. Erreurs

Tableau x : Contenu d'un message par Socket.IO des erreurs de clavardage

Message Socket.IO	Description
NotInRoomError	Envoyé lors d'une requête concernant un utilisateur qui ne fait pas partie du canal spécifié.
RoomDoesNotExistError	Envoyé lorsqu'une requête spécifie le nom d'un canal qui n'existe pas.
RoomAlreadyExistsError	Envoyé lorsqu'un client essaie de créer un canal avec un nom déjà existant.
MaxRoomsJoinedError	Envoyé lorsqu'un utilisateur veut joindre un canal de discussion alors qu'il a atteint le nombre maximal de canaux auxquels il peut appartenir.
AlreadyInRoomError	Envoyé lorsqu'un utilisateur essaie de rejoindre un canal auquel il fait déjà partie.

5.3. Communication d’authentification

5.3.1. Serveur → Client

Authentification d’un utilisateur

```
UserAuthenticated
```

Tableau x : Contenu d’un message par Socket.IO de l’authentification d’un utilisateur

Nom	Type	Description
hashSocketId	String	Hash de l’id du socket associé au client authentifié.
avatar	Number	Avatar de l’utilisateur
firstName	String	Prénom de l’utilisateur qui a été authentifié.
lastName	String	Nom de famille de l’utilisateur qui a été authentifié.
firstTime	Bool	Un booléen indiquant si c’est la première fois que l’utilisateur se connecte

Exemple

```
{
  hashSocketId: “a5v9c79f652w2ecc835638448”,
  firstName: "Antoine",
  lastName: "Martin",
  firstTime: true,
}
```

L’utilisateur ne peut se connecter car il est puni

```
UserCheatedError
```

Cette requête n’a aucun contenu.

5.3.2. Client → Serveur

Authentification d'un utilisateur

```
UserLogin
```

Tableau x : Contenu d'un message par Socket.IO de l'authentification d'un utilisateur

Nom	Type	Description
username	String	Le pseudonyme de l'utilisateur qui tente de s'authentifier.
hash	String	Le hash du mot de passe de l'utilisateur qui tente de s'authentifier.

Exemple

```
{
  username: “Tony”,
  hash: "a7sdjs9a8j798hdsah908has98dj",
}
```

Déconnexion d’un utilisateur

```
UserLogout
```

Cette requête n’a aucun contenu.

5.3.3. HTTP

Création d'un compte

```
@POST /auth/register/
```

Tableau x : Contenu d'une requête http pour la création d'un compte

Nom	Type	Description
firstName	String	Le prénom de l'utilisateur.
lastName	String	Le nom de famille de l'utilisateur.
username	String	Le pseudonyme de l'utilisateur.
email	String	L'adresse courriel de l'utilisateur.
hash	String	Le hash du mot de passe de l'utilisateur.
avatar	Int	Le id de l'avatar selon l'annexe D.

Exemple

```
{
    firstName: "Tony",
    lastName: "Clark",
    username: "TonyLeJolie",
    email: "TonyClark@hotmail.com",
    hash: "asd987987ashy987dyuas9duh9a8sud",
    avatar: 2
}
```

Tableau x : Codes d'erreur pour la requête de création d'un compte

Code	Signification	Description
409	UsernameExistsError	Envoyé si le pseudonyme spécifié lors de la création d'un compte est déjà utilisé.
410	EmailExistsError	Envoyé si l'adresse courriel spécifiée lors de la création d'un compte est déjà utilisée.

Salage du mot de passe

```
@GET /auth/salt/
```

Tableau x : Paramètres d'une requête http pour l'obtention du Salt

Nom	Type	Description
username	String	Le pseudonyme de l'utilisateur.

Exemple de réponse

```
{
    tempSalt: "2vKUUIrVAl6Ir6w0FfGT",
    permSalt: "7Q0FCqNAE9yeDPbsBe6h"
}
```

5.3.4. Erreurs

Tableau x : Contenu d'un message par Socket.IO des erreurs d'utilisateur

Message Socket.IO	Description
UserDoesNotExistError	Envoyé lors d'une requête spécifiant un pseudonyme inexistant.
BadPasswordError	Envoyé lors d'une requête contenant un mot de passe erroné.
UserAlreadyLoggedInError	Envoyé lors d'une requête d'authentification avec le pseudonyme d'un utilisateur déjà connecté.

5.4. Communication du menu principal

5.4.1. HTTP

Obtenir la liste des lobbys et parties en cours

```
@GET /game/list/
```

Réponse

```
{
   games: [{
      gameName: “Tony’s game”,
      playerCount: 3,
      gameMode: “BR”,
      difficulty: “Easy”
   }],
   lobbies: [{
      gameName: “Simon’s game”,
      playerCount: 1,
      gameMode: “FFA”,
      difficulty: “Hard”
   },{
      gameName: “Bruno’s game”,
      playerCount: 4,
      gameMode: “BR”,
      difficulty: “Easy”
   }]
}
```

5.4.2. Erreurs

Tableau x : Contenu d'un message par Socket.IO des erreurs de lobbies et de parties

Message Socket.IO	Description
LobbyDoesNotExistError	Envoyé lorsqu'une requête spécifie un lobby qui n'existe pas.
LobbyAlreadyExistsError	Envoyé lorsqu'un client essaie de créer un lobby avec un nom qui existe déjà.
LobbyFullError	Envoyé si un client veut rejoindre un lobby en tant que joueur alors que le nombre maximal de joueurs a été atteint.
AlreadyInLobbyError	Envoyé si un utilisateur veut rejoindre un lobby alors qu'il en fait déjà partie.

5.5. Communication du profil

5.5.1. HTTP

Obtenir les informations du profil

```
@GET /profile/?username=_
```

Tableau x : Paramètres d'une requête http des informations de profil d'un utilisateur

Nom	Type	Description
username	String	Le nom de l'utilisateur

Réponse

```
{
   nWins: 644,
   nLosses: 321,
   totalGameTimeMinutes: 2554,
   totalTimeMinutes: 3224,
   connections: [
      { login: "10-24-2020 14:55:26", logout: "10-24-2020 15:22:13" }
   ],
   FFA: {
      Easy: [
         {
            name: "myGame",
            timestamp: "10-24-2020 14:55:26",
            totalTime: 231,
            scores: [
               { username: "Bruno", score: 10000, avatar: 0 },
               { username: "Antoine", score: 0, avatar: 1 },
               { username: "Simon", score: 0, avatar: 2 },
            ]
         },
      ],
      Normal: [],
      Hard: []
   }
   BR: {
      Easy: [
         {
            name: "myGame",
            timestamp: "10-24-2020 14:55:26",
            totalTime: 231,
            scores: [
               { username: "Bruno", score: 10000, avatar: 0 },
               { username: "Antoine", score: 0, avatar: 1 },
               { username: "Simon", score: 0, avatar: 2 },
            ]
         },
      ],
      Normal: [],
      Hard: []
   }
}
```

5.6. Communication du lobby

5.6.1. Serveur → Client

Création d'un lobby

```
CreateLobby
```

Tableau x : Contenu d'un message par Socket.IO lorsqu'un lobby est créé

Nom	Type	Description
gameName	String	Le nom du lobby et de la partie associée à ce lobby.
username	String	Le pseudonyme de l'utilisateur ayant créé le lobby.
gameMode	String	Le type de partie créée.
difficulty	String	La difficulté de la partie créée.

Exemple

```
{
  gameName: "Jamie's game",
  username: "Jamie",
  gameMode: "Battle Royale",
  difficulty: "Easy"
}
```

Rejoindre un lobby en tant que joueur

```
JoinLobbyPlayer
```

Tableau x : Contenu d'un message par Socket.IO lorsqu'un joueur rejoint un lobby

Nom	Type	Description
username	String	Le pseudonyme de l'utilisateur qui a rejoint le lobby.

Exemple

```
{
  username: "Tony"
}
```

Rejoindre un lobby en tant que spectateur

```
JoinLobbySpectator
```

Tableau x : Contenu d'un message par Socket.IO lorsqu'un spectateur rejoint un lobby

Nom	Type	Description
username	String	Le pseudonyme de l'utilisateur qui a rejoint le lobby.

Exemple

```
{
  username: "Tony"
}
```

Mise à jour des lobbies

```
UpdateLobby
```

Tableau x : Contenu d'un message par Socket.IO lorsque le nombre d'utilisateurs d'un lobby change

Nom	Type	Description
gameName	String	Le nom du lobby et de la partie associée à ce lobby.
playerCount	Int	Le nombre de joueurs dans le lobby.

Exemple

```
{
  gameName: "Tony's game",
  playerCount: 3
}
```

Début d’une partie

```
StartGame
```

Tableau x : Contenu d’un message par Socket.IO lorsque le nombre d’utilisateurs d’un lobby change

Nom	Type	Description
gameName	String	Le nom du lobby et de la partie associée à ce lobby.

Exemple

```
{
  gameName: “Tony’s game”
}
```

Information sur un lobby

```
LobbyInfo
```

Tableau x : Contenu d'un message par Socket.IO contenant la description d'un lobby

Nom	Type	Description
players	String[ ]	Les pseudonymes des joueurs dans le lobby.
spectators	String[ ]	Les pseudonymes des spectateurs dans le lobby.

Exemple

```
{
  players: [“Tony”, “Bruno29”, “Testes99”],
  spectators: [“BigBoy”, “Marianne”]
}
```

Quitter un lobby en tant que joueur

```
LeaveLobbyPlayer
```

Tableau x : Contenu d'un message par Socket.IO lorsqu'un joueur quitte un lobby

Nom	Type	Description
username	String	Le pseudonyme de l'utilisateur qui a quitté le lobby.

Exemple

```
{
  username: "Simon"
}
```

Quitter un lobby en tant que spectateur

```
LeaveLobbySpectator
```

Tableau x : Contenu d'un message par Socket.IO lorsqu'un spectateur quitte un lobby

Nom	Type	Description
username	String	Le pseudonyme de l'utilisateur qui a quitté le lobby.

Exemple

```
{
  username: "Simon"
}
```

Supprimer un lobby

```
DeleteLobby
```

Tableau x : Contenu d'un message par Socket.IO lorsqu'un lobby est supprimé

Nom	Type	Description
gameName	String	Le nom du lobby et de la partie associée à ce lobby.

Exemple

```
{
  gameName: "Tony's game"
}
```

5.6.2. Client → Serveur

Création d'un lobby

```
CreateLobby
```

Tableau x : Contenu d'un message par Socket.IO de la création d'un lobby

Nom	Type	Description
gameName	String	Le nom du lobby et de la partie associée à ce lobby.
gameMode	String	Le type de partie créée.
difficulty	String	La difficulté de la partie créée.

Exemple

```
{
  gameName: "Tony's game",
  gameMode:  "FFA",
  difficulty:  "Hard"
}
```

Rejoindre un lobby en tant que joueur

```
JoinLobbyPlayer
```

Tableau x : Contenu d'un message par Socket.IO pour rejoindre un lobby en tant que joueur

Nom	Type	Description
gameName	String	Le nom du lobby et de la partie associée à ce lobby.

Exemple

```
{
  gameName: "Tony's game"
}
```

Rejoindre un lobby en tant que spectateur

```
JoinLobbySpectator
```

Tableau x : Contenu d'un message par Socket.IO pour rejoindre un lobby en tant que spectateur

Nom	Type	Description
gameName	String	Le nom du lobby et de la partie associée à ce lobby.

Exemple

```
{
  gameName: "Tony's game"
}
```

Début d’une partie

```
StartGame
```

Ce message n’a pas de contenu.

Quitter un lobby en tant que joueur

```
LeaveLobbyPlayer
```

Ce message n’a pas de contenu.

Quitter un lobby en tant que spectateur

```
LeaveLobbySpectator
```

Ce message n’a pas de contenu.

Supprimer un lobby

```
DeleteLobby
```

Ce message n’a pas de contenu.

Ajouter un utilisateur virtuel

```
AddBot
```

Ce message n’a pas de contenu.

Expulser un joueur de la partie

```
KickPlayer
```

Tableau x : Contenu d'un message par Socket.IO pour expulser un utilisateur d'un lobby

Nom	Type	Description
username	String	Le pseudonyme de l'utilisateur expulsé.

Exemple

```
{
  username: "Bobby"
}
```

5.6.3. Erreurs

Tableau x : Contenu d'un message par Socket.IO des erreurs de lobbies et de parties

Message Socket.IO	Description
UserNotInLobbyError	Envoyé lorsqu'un client essaie de quitter un lobby dont il ne fait pas partie.
LobbyDoesNotExistError	Envoyé lorsqu'une requête spécifie un lobby qui n'existe pas.
NotEnoughPlayersError	Envoyé si un client essaie de démarrer une partie sans avoir atteint le nombre minimal d'utilisateurs.

5.7. Communication lors d’une partie

5.7.1. Serveur → Client

Début d’une manche

```
StartRound
```

Tableau x : Contenu d’un message par Socket.IO du début d’une manche

Nom	Type	Description
artist	String	Le pseudonyme de l’utilisateur qui adopte le rôle d’artiste pour cette manche.

Exemple

```
{
  artist: "Tony"
}
```

Rejoindre une partie en tant que spectateur

```
JoinGameSpectator
```

Tableau x : Contenu d'un message par Socket.IO lorsqu'un spectateur rejoint une partie

Nom	Type	Description
username	String	Le pseudonyme de l'utilisateur qui a rejoint le lobby.

Exemple

```
{
  username: "Tony"
}
```

Quitter une partie en tant que joueur

```
LeaveGamePlayer
```

Tableau x : Contenu d'un message par Socket.IO lorsqu'un joueur quitte une partie

Nom	Type	Description
username	String	Le pseudonyme de l'utilisateur qui a quitté le lobby.

Exemple

```
{
  username: "Simon"
}
```

Quitter une partie en tant que spectateur

```
LeaveGameSpectator
```

Tableau x : Contenu d'un message par Socket.IO lorsqu'un joueur quitte une partie

Nom	Type	Description
username	String	Le pseudonyme de l'utilisateur qui a quitté le lobby.

Exemple

```
{
  username: "Simon"
}
```

Mot à dessiner

```
WordToDraw
```

Tableau x : Contenu d'un message par Socket.IO du début d'une manche

Nom	Type	Description
word	String	Le mot ou l'expression à dessiner.

Exemple

```
{
  word: "Dog"
}
```

Fin d’une manche

```
EndRound
```

Tableau x : Contenu d’un message par Socket.IO de la fin d’une manche

Nom	Type	Description
scores	Player[ ]	La liste du pointage de chaque utilisateur dans la partie.
word	String	le mot qui était à trouver

Exemple

```
{
  scores: [
      { username: "Tony", score: 123 },
      { username: "Charly", score: 456 },
      { username: "Roxanne", score: 789 }
   ],
   word: "Banana"
}
```

Fin d’une partie

```
EndGame
```

Tableau x : Contenu d’un message par Socket.IO de la fin d’une partie

Nom	Type	Description
gameName	String	Le nom de la partie terminée.

Exemple

```
{
  gameName: "my amazing game"
}
```

Demande d’un indice

```
Hint
```

Tableau x : Contenu d’un message par Socket.IO de la réponse à une demande d’indice

Nom	Type	Description
hint	String	Indice sur le mot à trouver.

Exemple

```
{
  hint: "It is an animal with four legs and fur"
}
```

Mot trouvé

```
WordFound
```

Tableau x : Contenu d'un message par Socket.IO pour indiquer qu'un utilisateur a trouvé le mot

Nom	Type	Description
username	String	Le pseudonyme de l'utilisateur ayant trouvé le mot.

Exemple

```
{
  username: "Tony"
}
```

Mise à jour du chronomètre d’une partie

```
GameTick
```

Tableau x : Contenu d’un message par Socket.IO du temps du jeu

Nom	Type	Description
timeLeft	Int	Le nombre de secondes restantes à la manche en cours.

Exemple

```
{
  timeLeft: 41
}
```

Élimination lors d’une bataille royale

```
Eliminate
```

Tableau x : Contenu d’un message par Socket.IO de notification d’élimination d’un joueur

Nom	Type	Description
username	String	Le pseudonyme de l’utilisateur qui a été éliminé lors de la manche.

Exemple

```
{
  username: "Tony"
}
```

Mot à dessiner

```
WordToDraw
```

Tableau x : Contenu d'un message par Socket.IO de mot à dessiner pour l'artiste

Nom	Type	Description
word	String	Le mot à dessiner par l'artiste.

Exemple

```
{
  word: "Dog"
}
```

Un utilisateur a été expulsé par les autres joueurs

```
UserKicked
```

Tableau x : Contenu d'un message par Socket.IO de notification d'élimination d'un joueur

Nom	Type	Description
username	String	Le pseudonyme de l'utilisateur qui a été expulsé.

Exemple

```
{
  username: "Tony"
}
```

5.7.2. Client → Serveur

Rejoindre une partie en tant que spectateur

```
JoinGameSpectator
```

Tableau x : Contenu d'un message par Socket.IO lorsqu'un spectateur rejoint une partie

Nom	Type	Description
gameName	String	Le pseudonyme de l'utilisateur qui a rejoint le lobby.

Exemple

```
{
  gameName: "Tony's game"
}
```

Quitter une partie en tant que spectateur

```
LeaveGameSpectator
```

Ce message n’a pas de contenu.

Quitter une partie en tant que joueur

```
LeaveGamePlayer
```

Ce message n’a pas de contenu.

Information sur une partie

```
GameInfo
```

Tableau x : Contenu d'un message par Socket.IO contenant la description d'une partie

Nom	Type	Description
scores	UsernameScoreAvatar[ ]	Les scores des joueurs et leurs avatars.

Exemple

```
{
  scores: [
    {username: "Anmaru", score: 121, avatar: "1"},
    {username: "BigBoy", score: 1331, avatar: "8"},
    {username: "Bruno29", score: 14641, avatar: "0"}
  ]
}
```

Mot à dessiner

```
WordToDraw
```

Tableau x : Contenu d'un message par Socket.IO du début d'une manche

Nom	Type	Description
word	String	Le mot ou l'expression à dessiner.

Exemple

```
{
  word: "Dog"
}
```

L’utilisateur a triché

```
UserCheated
```

Ce message n’a pas de contenu.

5.8. Communication de la création d'une paire mot-image

5.8.1. HTTP

```
@POST /wordImagePair/
```

Tableau x : Paramètres d'une requête POST http pour la création d'une pair mot-image

Nom	Type	Description
hashSocketId	String	Hash de l'id du socket associé au client authentifié.
word	String	Mot qui sera à deviner avec cette pair mot-image
canvasSize	Int	Taille du carré original en pixels
paths	path[ ]	Taille du canevas, couleur du trait, taille du trait et liste de coordonnées
hints	String[ ]	Les indices associés à la pair mot-image
difficulty	String	La difficulté de la partie créée → liste dans l'annexe B

Exemple

```
{
   hashSocketId: "jasd980u798jasd98jas",
   word: "Cat",
   paths: [
      {
         canvasSize: 800,
         color: "#FF381D9A",
         strokeWidth: 12.3,
         path: "100 100 200 150 150 200 0 0"
      }
   ],
   hints: [
      "It lives in your house",
      "Vvvvvrrrrr it's all furry!",
      "It has four legs"
   ],
   difficulty: "Easy"
}
```

5.9. Communication de l’interface de dessin

5.9.1. Serveur → Client

Assigner un trait

```
SetPath
```

Tableau x : Contenu d’un message par Socket.IO d’un nouveau trait de dessin

Nom	Type	Description
pathId	int	L’identifiant du trait de dessin à modifier. Il s’agit d’un nombre partant à 0 et incrémentant selon le nombre de traits.
color	String	La couleur du trait en format hexadécimal : #AARRGGBB.
strokeWidth	Float	La largeur du trait.
path	String	Le trait représenté par tous les points en x et y (x1 y1 x2 y2 x3 y3 …)
canvasSize	Int	La taille du canevas en pixels. Le canevas doit être carré.

Exemple

```
{
  pathId: 4,
  color: “#FF381D9A”,
  strokeWidth: 42.0,
  path: “10.12 20.06 131.16 23.185 12.54 13.13 29.21 34.111”,
  canvasSize: 800
}
```

Ajouter à un trait

```
AppendToPath
```

Tableau x : Contenu d'un message par Socket.IO de la modification d'un trait de dessin

Nom	Type	Description
x	Float	Coordonnée en x du nouveau point.
y	Float	Coordonnée en y du nouveau point.

Exemple

```
{
  x: 10.21,
  y: 20.31
}
```

5.9.2. Client → Serveur

Assigner un trait

```
SetPath
```

Tableau x : Contenu d'un message par Socket.IO d'un nouveau trait de dessin

Nom	Ty7pe	Description
pathId	Int	L'identifiant du trait de dessin à modifier. Il s'agit d'un nombre partant à 0 et incrémentant selon le nombre de traits.
color	String	La couleur du trait en format hexadécimal : #AARRGGBB.
strokeWidth	Float	La largeur du trait.
path	String	Le trait représenté par tous les points en x et y (x1 y1 x2 y2 x3 y3 …)
canvasSize	Int	La taille du canevas en pixels. Le canevas doit être carré.

Exemple

```
{
  pathId: 12,
  color: "#FF381D9A",
  strokeWidth: 56.7,
  path: "10.12 20.06 131.16 23.185 12.54 13.13 29.21 34.111",
  canvasSize: 425
}
```

Ajouter à un trait

```
AppendToPath
```

Tableau x : Contenu d'un message par Socket.IO de la modification d'un trait de dessin

Nom	Type	Description
x	Float	Coordonnée en x du nouveau point.
y	Float	Coordonnée en y du nouveau point.

Exemple

```
{
  x: 10.21,
  y: 20.31
}
```

5.9.3. Erreurs

Tableau x : Contenu d'un message par Socket.IO des erreurs du dessin

Message Socket.IO	Description
NotArtistError	Envoyé si un client essaie de dessiner alors qu'il n'est pas l'artiste désigné pour la manche.

5.10. Communication des trophées

5.10.1. HTTP

Obtenir les trophées

```
@GET /achievement/?username=_
```

Tableau x : Paramètres d'une requête GET http pour obtenir les trophées

Nom	Type	Description
username	String	Le nom de l'utilisateur.

Réponse

```
{
   trophies: [
      {
         trophy: "win1Game",
         hint: "try to win a game",
         isUnlocked: true,
         rank: "silver",
      },

   ]
}
```

**La liste des trophées présentée à l'annexe A*

5.11. Communication du tableau de classement

5.11.1. HTTP

Obtenir le tableau de classement

```
@GET /leaderboard/
```

Tableau x : Paramètres d'une requête GET http pour obtenir le classement des joueurs.

Réponse

```
{
   FFA: {
       Easy: {
           Day: [
                  {
                    username: "Vincent",
                    nbOfGameWon: 51
                  },
                  {
                    username: "Sophie",
                    nbOfGameWon: 42
                  },
                ],
           Week: []
           Total: []
       }
       Normal: ... ,
       Hard: …,
   }
   BR: ...
}
```

**La liste des trophées est présentée à l'annexe A*

Annexe A

Rang	Identifiant	Trophée
Bronze	New Beginnings	Jouer 1 partie
Silver	Participation Award	Jouer 10 parties
Gold	Bigger Participation Award	Jouer 100 parties
Bronze	PogChamp	Gagner 1 partie
Silver	Try Hard	Gagner 10 parties
Gold	Get a Life	Gagner 100 parties
Bronze	A New Addiction	Gagner 1 partie
Silver	The Downfall	Gagner 10 parties
Gold	The bottom of the Pit	Gagner 100 parties

Annexe B

Identifiant	Difficulté
Easy	Facile
Normal	Normal
Hard	Difficile

Annexe C

Identifiant	Difficulté
FFA	Mêlée générale
BR	Bataille royale

Annexe D

Identifiant	Avatar
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11