Les Décalés

LNAH 2K17
Protocole de communication

Version 4.1

Historique des révisions

Date	Version	Description	Auteur
2017-09-19	1.0	Description de la communication client-serveur (websockets et REST). Description des paquets utilisés par les web sockets.	Ariane Tourangeau
2017-09-20	2.0	Écriture de l'introduction et des explications sur la connexion par sockets et l'architecture REST.	Michaël Sghaïer
2017-09-21	3.0	Ajout des descriptions du protocole de communication.	Ariane Tourangeau Michaël Sghaïer
2017-09-22	3.1	Ajout des descriptions du protocole de communication (suite et fin).	Ariane Tourangeau Michaël Sghaïer
2017-09-28	4.0	Relecture finale du document.	Mikaël Ferland
2017-09-29	4.1	Formatage	Mikaël Ferland Jean-Marc Al-Romhein

Table des matières

Protocole de communication

1. Introduction

LNAH 2K17 étant un jeu en réseau, il est nécessaire d'établir en détail un protocole de communication qui pourra supporter les échanges entre les différentes composantes de l'architecture du système. Le présent document a ainsi pour but de présenter dans un premier temps l'architecture client-serveur qui sera utilisée puis de décrire les paquets qui seront échangés au sein de cette architecture.

2. Communication client-serveur

2.1 Socket TCP

La technologie de communication pour l'actualisation en temps réel des clients sera les sockets TCP. Cette méthode permet d'éviter l'exécution de requête de la part des clients à une intervalle de temps fixe pour obtenir les données à jour. En effet, c'est plutôt la responsabilité du serveur d'avertir le(s) client(s) lorsque du nouveau contenu est disponible en leur envoyant directement les données.

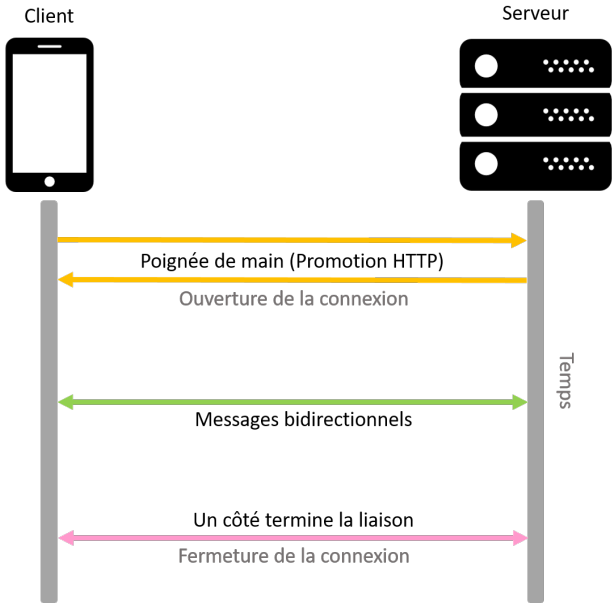

Figure 1 : Illustration de la communication entre un client et un serveur au moyen de TCP

Tel qu'on peut voir en vert dans la figure 1, les sockets se basent sur la création d'une connexion permanente entre le client et le serveur. Notons toutefois que la connexion est initiée par un TCP « *3-way handshake* » où serveur et client s'assurent de pouvoir communiquer entre eux et non via une promotion HTTP. À l'instant où cette poignée de main est négociée, la connexion est active entre les deux parties qui peuvent dorénavant communiquer tant et aussi longtemps qu'aucune ne ferme la connexion.

La communication étant bidirectionnelle et active en tout temps, cela permet de mettre à jour les clients en temps réel. Dans notre cas, il est primordial de mettre à jour les données le plus rapidement possible pour assurer une belle expérience de jeu ou d'édition de cartes.

2.2 Architecture REST

Les autres requêtes du client envers le serveur utiliseront l'architecture REST. Afin d'éviter de maintenir une connexion, REST favorise les requêtes singulières des clients envers le serveur pour obtenir entre autres des données. L'authentification de l'utilisateur ainsi que l'obtention des cartes (pour l'affichage) sont des exemples de requêtes qui se prêtent bien à REST. Cette technologie est pertinente si des données statiques sont en jeu ou lorsque les mises à jour sont rares.

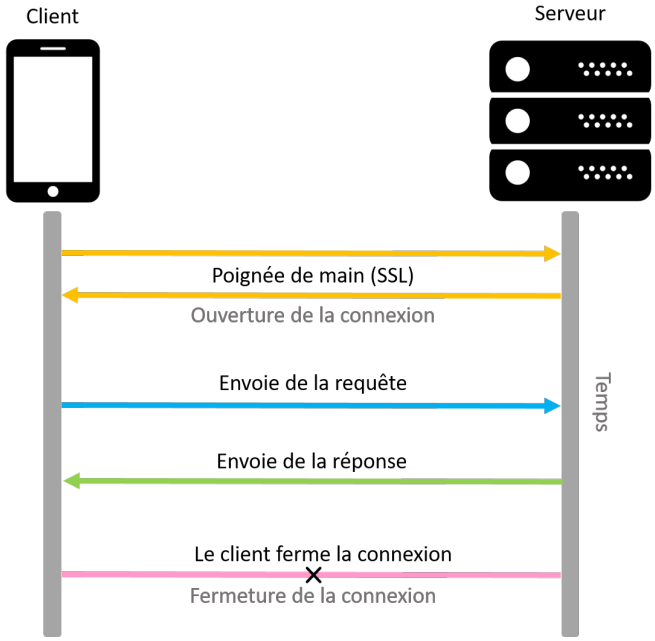

Figure 2 : Illustration de la communication entre un client et un serveur au moyen de l'API REST

L'architecture REST se base sur les requêtes HTTP. Tel que présenté dans la figure 2, une poignée de main est initialement négociée entre le client et le serveur à l'aide d'un TCP « *3-way handshake* » (HTTP utilisant la couche transport TCP). Notons que l'utilisation de SSL au moment de la poignée de main est uniquement nécessaire si HTTPS est utilisé. Une fois celle-ci achevée, la connexion entre le client et le serveur est établie. Le client peut alors envoyer sa requête HTTP (GET, POST, etc.) au serveur qui lui renvoie la réponse. Une fois que celle-ci reçue, la connexion entre le client et le serveur prend fin.

REST étant une architecture de haut niveau, l'implémentation nécessite moins de code de la part des développeurs. Le fait qu'elle soit orientée « ressources » (c.-à-d. ayant le but de desservir le contenu présent dans le serveur aux différents clients) permet également de gérer proprement toutes les requêtes de contenu de la part des clients telles que l'obtention de la liste des parties en cours, des tournois, des profils d'utilisateurs, etc. Et il va sans dire que cette architecture convient tout naturellement au client web qui se contentera d'afficher du contenu statique.

3. Description des messages

Cette section présente de manière exhaustive et détaillée les différents messages envoyés du client (lourd et léger) au serveur et vice-versa. Pour chaque action possible, issue du document SRS, nous détaillons la séquence de requêtes client-serveur qui sera utilisée pour implémenter ladite action.

Le format de description est le suivant :

X.Y : Une certaine catégorie d'actions (ex: jeu en ligne, édition en ligne) ou une action sans catégorie (ex: authentification, création de compte)

x.y.z : une action spécifique (ex. : récupérer les profils des joueurs)

a. Requête [Type de requête : REST (si via HTTP) ou Socket] (dans le cas du client vers le serveur):
Verbe HTTP (ex: POST, GET, etc.) [ip]:[port]/URI/ (dans le cas d'une requête REST)
{Message JSON envoyé au serveur} (dans le cas d'une requête REST POST ou Socket)

b. Réponse:

   i. Succès: {Message JSON renvoyé par le serveur} ou code HTTP (200)

   ii. Erreur: {"error": "some error"}

c. Broadcast [Socket] (dans le cas du serveur vers le client):
{Description JSON du message envoyé}

3.1 Authentification

a. Requête [REST] :
   POST [ip]:[port]/api/login
   {"username": "…", "password": "…"}
b. Réponse :
   i. Succès: {"token": "…"}
   ii. Erreur: {"error": "some error"}

3.2 Création de compte

a. Requête [REST] :
   POST [ip]:[port]/api/account
   {"username": "…", "…": "…"}
b. Réponse :
   i. Succès: HTTP 200 code
   ii. Erreur: {"error": "some error"}

3.3 Profils

3.3.1 Récupérer le profil d'un joueur spécifique

a. Requête [REST] : GET [ip]:[port]/api/profile/{id}
b. Réponse

i. Succès:

```
{
 "Avatar": "…",
 "Username": "…",
 "Stats":
        {
        "GameVictories": "…",
        "GameDefeats": "…",
        "TournamentVictories": "…",
        "TournamentDefeats": "…",
        "GamesPlayed": "…",
        "TournamentsPlayed": "…",
        "MapCollaborations": "…",
        "TotalPoints": "…",
        "Achievements":
                {
                        {
                        "Id": "…",
                        "Name": "…",
                        "Type": "…",
                        "Description": "…"
                        },
                        { … }
                }

        }
}
```

ii. Erreur: {"error": "some error"}

3.3.2 Récupérer les profils des joueurs

a. Requête [REST] : GET [ip]:[port]/api/profiles
b. Réponse
   i. Succès :

```
{
        {
                "Avatar": "…",
                "Username": "…",
                "Stats":
                {
                        "GameVictories": "...",
                        "GameDefeats": "...",
                        "TournamentVictories": "...",
                        "TournamentDefeats": "...",
                        "GamesPlayed": "...",
                        "TournamentsPlayed": "...",
                        "MapCollaborations": "...",
                        "TotalPoints": "...",
                        "Achievements":
                        {
                                {
                                        "Id": "...",
                                        "Name": "...",
                                        "Type": "...",
                                        "Description": "..."
                                },
                                { … }
                        }

                }
        },
        { ... }
}
```

   ii. Erreur: {"error": "some error"}

3.4 Gestion des amis

3.4.1 Visualiser sa liste d'amis

a. Requête [REST] : GET [ip]:[port]/api/profile/{user_id}/friends/
b. Réponse :
   i. Succès :

```
{
        {
                "UserId": "...",
                "ProfileUrl": "...",
                "Online": "true/false",
                "InGame" : "true/false",
                "Spectating" : "true/false"
                "SessionId": "..."
        },
        { … }
}
```

   ii. Erreur : {"error": "some error"}

3.4.2 Ajouter un ami

a. Requête [REST] :
   POST [ip]:[port]/api/profile/{user_id}/friends/
   {“UserId”: “...”}
b. Réponse :
   i. Succès : HTTP 200 code
   ii. Erreur : {“error”: “some error”}

3.4.3 Supprimer un ami

a. Requête [REST] :
   DELETE [ip]:[port]/api/profile/{user_id}/friends/{friend_id}
b. Réponse :
   i. Succès : HTTP 200 code
   ii. Erreur : {“error”: “some error”}

3.4.4 Accepter/refuser une demande d’ami

a. Requête [REST] :
   POST [ip]:[port]/api/profile/{user_id}/friend_request
   {“FriendRequestId”: “...”, “RequestAccepted”: “true/false”}
b. Réponse :
   i. Succès : HTTP 200 code
   ii. Erreur : {“error”: “some error”}

3.4.5 Notifier un client d’une demande d’amis

a. Broadcast [Socket] :

```
{
    “Event”: “friendRequestEvent”,
    “FriendRequestId”: “...”,
    “FriendId”: “...”,
    “FriendName”: “...”
}
```

3.4.6 Proposer une partie rapide à un ami en ligne

a. Requête [Socket] :

```
{
    “Service”: “game”,
    “Action”: “challengeFriend”,
    “FriendId”: “...”,
    “Timestamp”: “...”
}
```

b. Réponse :

i. Succès :

```
{
    “Service”: “game”,
    “Event”: “friendChallengedEvent”,
    “Acknowledge”: “...”
}
```

ii. Erreur : {“error”: “some error”}

c. Broadcast [Socket] :

```
{
    “Service”: “game”,
    “Event”: “friendChallengedEvent”,
    “FriendName”: “...”,
    “Timestamp”: “...”
}
```

3.5 Cartes de jeu

3.5.1 Récupérer les cartes

a. Requête [REST]:

Récupérer toutes les cartes → GET [ip]:[port]/api/maps

Récupérer les cartes d'un utilisateur spécifique → GET [ip]:[port]/api/maps/user/{id}

b. Réponse

i. Succès

```
{
    {
        "Id": "...",
        "Name": "...",
        "Picture": "...",
        "Author": "...",
        "CreationDate": "...",
        "Privacy": "public/private",
        "Password": "...",
        "Rating": "...",
        "MapElements":
        {
            {
                "Id": "...",
                "ElementType": "...",
                "Position": "...",
                "Angle": "...",
                "Size": "..."
            },
            { … }
        },
        "Comments":
        {
            {
                "Username": "...",
                "Comment": "...",
                "Timestamp": "..."
            },
            { … }
        }
    },
    {...}
}
```

ii. Erreur : {"error": "some error"}

3.5.2 Créer ou modifier une carte

a. Requête [REST]:

Création d'une carte → POST [ip]:[port]/api/maps/
Modification d'une carte → PUT [ip]:[port]/api/maps/{id}

```
{
    "Id": null,
    "MapName": "...",
    "Picture": "...",
    "Creator": "...",
    "CreationDate": "...",
    "Privacy": "public/private",
    "Password": "...",
    "Rating": "...",
    "MapElements":
    {
        {
            "Id": "...",
            "ElementType": "...",
            "Position": "...",
            "Angle": "...",
            "Size": "..."
        },
        { … }
    },
    "Comments":
    {
        {
            "Username": "...",
            "Comment": "...",
            "Timestamp": "..."
        },
        { … }
    }
}
```

b. Réponse

i. Succès:

```
{
    "Id": "...",
    "MapName": "...",
    "Picture": "...",
    "Creator": "...",
    "CreationDate": "...",
    "Privacy": "public/private",
    "Password": "...",
    "Rating": "...",
    "MapElements":
    {
        {
            "Id": "...",
            "ElementType": "...",
            "Position": "...",
            "Angle": "...",
```

```
                                        “Size”: “...”
                                },
                                { … }
                        },
                        “Comments”:
                        {
                                {
                                        “Username”: “...”,
                                        “Comment”: “...”,
                                        “Timestamp”: “...”
                                },
                                { … }
                        }
                }
```

ii. Erreur : {“error”: “some error”}

3.5.3 Ajouter une évaluation à une carte

a. Requête [REST]:

POST [ip]:[port]/api/maps/{map_id}/rating

{“UserId”: “...”, “Rating”: “...”}

b. Réponse
   i. Succès : HTTP 200 code
   ii. Erreur : {“error”: “some error”}

3.5.4 Ajouter un commentaire à une carte

a. Requête [REST]:

```
POST [ip]:[port]/api/maps/[map_id]/comment
{
        “UserId”: “...”,
        “CommentValue”: “...”,
        “Timestamp”: “...”
}
```

b. Réponse
   i. Succès : HTTP 200 code
   ii. Erreur : {“error”: “some error”}

3.6 Déconnexion

a. Requête [REST] :

GET [ip]:[port]/api/logout?id=””

b. Réponse :
   i. Succès : HTTP 200 code
   ii. Erreur : {“error”: “some error”}

3.7 Jeu en ligne

3.7.1 Jouer une partie rapide

a. Requête [Socket] :

```
{
        “Service”: “game”,
        “Action”: “joinGame”,
        “UserId”: “...”,
        “Timestamp”: “...”
}
```

b. Réponse :

i. Succès :

```
{
        “Service”: “game”,
        “Event”: “joinGameEvent”,
        “GameId”: “...”,
        “Acknowledge”: “ok”
}
```

ii. Erreur : {“error”: “some error”}

3.7.2 Notifier le serveur d’un changement de configuration avant le lancement d’une partie

a. Requête [Socket] :

```
{
        “Service”: “game”,
        “Action”: “updateConfigNewGame”,
        “Configuration”:
        {
                “PuckSpeed”: “...”,
                “Friction”: “...”,
                “Card”: “...”,
                { … }
                }
        }
```

b. Réponse :

i. Succès :

```
{
        “Service”: “game”,
        “Event”: “updateConfigNewGameEvent”,
        “Acknowledge”: “ok”
}
```

ii. Erreur : {“error” : “some error”}

3.7.3 Notifier le créateur de la partie de l'arrivée de son adversaire (jumelage)

a. Broadcast [Socket] :

```
{
        "Service": "game",
        "Event": "adversaryFoundEvent",
        "GameId": "id",
        "Opponent": "...",
        "Timestamp": "..."
}
```

3.7.4 Confirmer au serveur la configuration de la nouvelle partie

a. Requête [Socket] :

```
{
        "Service": "game",
        "Action": "confirmConfigNewGame",
        "Timestamp": "..."
}
```

b. Réponse :

i. Succès :

```
{
        "Service": "game",
        "Event": "confirmConfigNewGameEvent",
        "Acknowledge": "ok"
}
```

ii. Erreur : {"error" : "some error"}

3.7.5 Notifier les joueurs de l'utilisation des configurations et d'une carte par défaut (au cas où les utilisateurs n'ont rien confirmé)

a. Broadcast [Socket] :

```
{
        "Service": "game",
        "Event": "confirmDefaultConfigNewGameEvent",
        "Timestamp": "..."
}
```

3.7.6 Envoyer la position de la rondelle à l'adversaire

a. Requête [Socket] :

```
{
        "Service": "game",
        "Action": "updatePuckPosition",
        "Timestamp": "..."
        "Coordinates":
        {
                "xPos": "...",
                "yPos": "...",
                "zPos": "..."
        }
}
```

b. Broadcast [Socket] :

```
{
        "Service": "game",
        "Event": "updatePuckPositionEvent",
        "Timestamp": "..."
        "Coordinates":
        {
                "xPos": "...",
                "yPos": "...",
                "zPos": "..."
        }
}
```

3.7.7 Envoyer la position du maillet à l'adversaire

a. Requête [Socket] :

```
{
        "Service": "game",
        "Event": "updateStrikerPosition",
        "Timestamp": "..."
        "Coordinates":
        {
                "xPos": "...",
                "yPos": "...",
                "zPos": "..."
        }
}
```

b. Broadcast [Socket] :

```
{
        "Service": "game",
        "Event": "updateStrikerPositionEvent",
        "Timestamp": "..."
        "Coordinates":
        {
                "xPos": "...",
                "yPos": "...",
                "zPos": "..."
        }
}
```

3.7.8 Notifier les joueurs d'un but

a. Requête [Socket] :

```
{
        "Service": "game",
        "Action": "notifyGoal",
        "PlayerId": "...",
        "Timestamp": "..."
}
```

b. Réponse :

   i. Succès :

```
{
        "Service": "game",
        "Event": "notifyGoalEvent",
        "Acknowledge": "ok"
}
```

   ii. Erreur : {"error": "some error"}

c. Broadcast [Socket] :

```
{
        "Service": "game",
        "Event": "notifyGoalEvent",
        "PlayerId": "...",
        "Timestamp": "..."
}
```

3.7.9 Signaler à l'adversaire la fin d'une partie

a. Requête [Socket] :

```
{
        "Service": "game",
        "Action": "endOfGame",
        "WinnerId": "...",
        "Timestamp": "..."
}
```

b. Réponse :

   i. Succès :

```
{
        "Service": "game",
        "Event": "endOfGameEvent",
        "Acknowledge": "ok"
}
```

   ii. Erreur : {"error": "some error"}

c. Broadcast [Socket] :

```
{
        "Service": "game",
        "Event": "endOfGameEvent",
        "WinnerId": "...",
        "Timestamp": "..."
}
```

3.7.10 Résumé de partie

a. Broadcast [Socket] :

```
{
        “Service”: “game”,
        “Event”: “gameStatsEvent”,
        “Player1”: “...”,
        “Player2”: “...”,
        ‘Player1Score”: “...”,
        “Player2Score”: “...”,
        “Winner”: “...”,
        “PointsWon”: “...”,
        “AchievementsUnlocked”:
        {
                {
                        “Id”: “...”,
                        “Name”: “...”,
                        “Type”: “...”,
                        “Description”: “...”
                },
                { … }
        },
        “Timestamp”: “...”
}
```

3.7.11 Déconnexion de la part d’un joueur

a. Broadcast [Socket] :

```
{
        “Service”: “game”,
        “Event”: “disconnectedPlayerEvent”,
        “PlayerId”: “...”,
        “Timestamp”: “...”
}
```

3.7.12 Reconnexion et retour dans la partie d’un joueur déconnecté

a. Broadcast [Socket] :

```
{
        “Service”: “game”,
        “Event”: “reconnectedPlayerEvent”,
        “PlayerId”: “...”,
        “Timestamp”: “...”
}
```

3.7.13 Notification au joueur reconnecté que sa partie est toujours en cours

a. Broadcast [Socket] :

```
{
        “Service”: “game”,
        “Event”: “unfinishedBusinessEvent”,
        “GameId”: “...”,
        “Timestamp”: “...”
}
```

3.7.14 Reprise de la partie par un joueur reconnecté

a. Requête [Socket] :

```
{
        “Service”: “game”,
        “Action”: “returnToGame”,
        “GameId”: “...”,
        “Timestamp”: “...”
}
```

b. Réponse :

i. Succès :

```
{
        “Service”: “game”,
        “Event”: “returnToGameEvent”,
        “Acknowledge”: “ok”
}
```

ii. Erreur : {“error”: “some error”}

3.7.15 Récupérer la liste des parties en cours

a. Requête [REST] :

GET [ip]:[port]/api/games/

c. Réponse :

i. Succès :

```
{
        {
                “GameId”: “...”,
                “Player1”: “...”,
                “Player2”: “...”,
                “Player1Score”: “1”,
                “Player2Score”: “2”
        },
        { … }
}
```

ii. Erreur : {“error”: “some error.”}

3.7.16 Demander à joindre une partie en tant que spectateur

a. Requête [Socket] :

```
{
        “Service”: “game”,
        “Action”: “spectateGame”,
        “GameId”: “...”,
        “Timestamp”: “...”
}
```

b. Réponse :

   i. Succès :

```
{
        “Service”: “game”,
        “Event”: “spectateGameEvent”,
        “Acknowledge”: “ok”
}
```

   ii. Erreur : {“error”: “some error.”}

c. Broadcast [Socket] :

```
{
        “Service”: “game”,
        “Event”: “spectatorJoinedEvent”,
        “Name”: “...”,
        “Timestamp”: “...”
}
```

3.7.17 Envoyer les statistiques d’une partie en cours au nouveau spectateur

a. Broadcast [Socket]:

```
{
        “Service”: “game”,
        “Event”: “currentGameStatsEvent”,
        “Player1”: “...”,
        “Player2”: “...”,
        “Player1Score”: “...”,
        “Player2Score”: “...”,
        “timestamp”: “...”
}
```

3.7.18 Gérer la déconnexion d'un spectateur

a. Requête [Socket] :

```
{
        "Service": "game",
        "Action": "leaveCurrentGame",
        "SpectatorId": "",
        "Timestamp": "..."
}
```

b. Réponse :

i. Succès :

```
{
        "Service": "game",
        "Event": "leaveCurrentGameEvent",
        "Acknowledge": "ok"
}
```

ii. Erreur : {"error": "some error."}

c. Broadcast [Socket] :

```
{
        "Service": "game",
        "Event": "spectatorLeftEvent",
        "SpectatorId": "...",
        "Timestamp": "..."
}
```

3.8 Magasin

3.8.1 Récupérer les items à vendre

a. Requête [REST]:

GET [ip]:[port]/api/store/items

b. Réponse

i. Succès :

```
{
        {
                "Id": "...",
                "Type": "...",
                "Description": "...",
                "Price": "...",
                "Picture": "..."
        },
        {...}
}
```

ii. Erreur : {"error": "some error"}

3.8.2 Acheter un item

a. Requête [REST]:

POST [ip]:[port]/api/shop/items/{item_id}

{“UserId”: “...”}

b. Réponse
   i. Succès : HTTP 200 code
   ii. Erreur : {“error”: “some error ”}

3.8.3 Récupérer l’inventaire d’un utilisateur spécifique

a. Requête [REST]:

GET [ip]:[port]/api/profile/{id}/inventory

b. Réponse
   i. Succès :

```
{
        {
                “Id”: “...”,
                “Type”: “...”,
                “Description”: “...”,
                “Picture”: “...”,
                “Activated”: “true/false”
        },
        {...}
}
```

   ii. Erreur : {“error”: “some error ”}

3.8.4 Activer/désactiver un élément de l’inventaire

a. Requête [REST]:

PUT [ip]:[port]/api/profile/{user_id}/inventory/{item_id}
{“Activate”: “true/false”}

b. Réponse
   i. Succès : HTTP 200 code
   ii. Erreur : {“error”: “some error”}

3.9 Édition en ligne

3.9.1 Mise à jour d'un objet de la carte

a. Requête [Socket]

```
{
        "Service": "maps",
        "Action": "updateElementProperties"
        "ElementId": "...",
        "MapId": "...",
        "Position": "...",
        "Angle": "...",
        "Size": "...",
        "Timestamp": "..."
}
```

b. Réponse

   i. Succès :

```
{
        "Service": "maps",
        "Event": "updateElementPropertiesEvent",
        "Acknowledge": "ok"
}
```

   ii. Erreur : {"error": "some error"}

c. Broadcast [Socket] :

```
{
        "Service": "maps",
        "Event": "updateElementPropertiesEvent",
        "ElementId": "...",
        "MapId": "...",
        "Position": "...",
        "Angle": "...",
        "Size": "...",
        "Timestamp": "..."
}
```

3.9.2 Sélection d'un objet par un utilisateur

a. Requête [Socket]

```
{
        "Service": "maps",
        "Action": "elementSelected",
        "UserId": "...",
        "ElementId": "...",
        "MapId": "..."
}
```

b. Réponse

   i. Succès :

```
{
        "Service": "edition",
        "Event": "elementSelectedEvent",
        "Acknowledge": "ok"
}
```

   ii. Erreur : {"error": "some error"}

c. Broadcast [Socket] :

```
{
        "Service": "maps",
        "Event": "elementSelectedEvent",
        "UserId": "...",
        "ElementId": "...",
        "MapId": "...",
        "OutlineColor": "...",
        "Timestamp": "..."
}
```

3.9.3 Récupérer les utilisateurs éditant une carte spécifique en ligne

a. Requête [REST]

GET [ip]:[port]/api/maps/[map_id]/users

b. Réponse

   i. Succès :

```
{
        {
         "UserId": "...",
         "Username": "..."
        },
        { … }
}
```

   ii. Erreur: {"error": "some error"}

3.9.4 Notifier les clients d'une nouvelle liste d'utilisateurs disponible

a. Requête [Socket]

```
{
        "Service": "edition",
        "Event" : "MapUsersListUpdatedEvent",
        "MapId": "...",
        "Users" :
        {
                {
                        "UserId": "...",
                        "Username": "..."
                },
                { … }
        }
}
```

b. Réponse

```
{
        "Service": "edition",
        "Event": "mapUsersListUpdatedEvent",
        "Acknowledge": "ok"
}
```

3.9.5 Ajouter un objet

a. Requête [Socket]

```
{
        "Service" : "maps",
        "Action": "addObject"
        "MapId": "...",
        "ElementId": "...",
        "Type": "...",
        "Position": "...",
        "Timestamp": "..."
}
```

b. Réponse

i. Succès

```
{
        "Service": "maps",
        "Event": "addObjectEvent"
        "Acknowledge": "ok"
}
```

ii. Erreur : {"error": "some error"}

c. Broadcast [Socket] :

```
{
        "Service": "maps",
        "Event": "addObjectEvent",
        "MapId": "...",
        "ElementId": "...",
        "Type": "...",
        "Position": "...",
        "Timestamp": "..."
}
```

3.9.6 Supprimer un élément

a. Requête [Socket]:

```
{
        “Service” : “maps”,
        “Action”: “deleteMapElement”
        “MapId”: “...”,
        “ElementId”: “...”,
        “Timestamp”: “...”
}
```

b. Réponse

   i. Succès

```
{
        “Service”: “maps”,
        “Event”: “deleteMapEvent”,
        “Acknowledge”: “ok”
}
```

   ii. Erreur : {“error”: “some error”}

c. Broadcast [Socket] :

```
{
        “Service”: “maps”,
        “Event”: “deleteMapEvent”,
        “MapId”: “...”,
        “ElementId”: “...”,
        “Timestamp”: “...”
}
```

3.9.7 Modification du coefficient de friction ou du rebond ou de l'accélération

a. Requête [Socket]:

```
{
        "Service" : "maps",
        "Action": "configurationUpdated"
        "MapId": "...",
        "ItemId": "...",
        "ConfigurationType": "...",
        "ConfigurationValue": "...",
}
```

b. Réponse
   i. Succès

```
{
        "Service": "maps",
        "Event": "configurationUpdatedEvent",
        "Acknowledge": "ok"
}
```

   ii. Erreur : {"error": "some error"}

c. Broadcast [Socket]

```
{
        "Service" : "maps",
        "Action": "configurationUpdatedEvent"
        "MapId": "...",
        "ConfigurationType": "...",
        "ConfigurationValue": "...",
}
```

3.10 Clavardage

3.10.1 Le client envoie un message

a. Requête [Socket]:

```
{
        "Service": "chat",
        "Action": "sendMessage",
        "Sender": "...",
        "Recipient": "...",
        "MessageValue": "...",
        "Timestamp": "..."
}
```

b. Réponse
   i. Succès

```
{
        "Service": "chat",
        "Event": "sendMessageEvent",
        "Acknowledge": "ok"
}
```

   ii. Error: {"error": "some error"}

3.10.2 Client reçoit un message du serveur

a. Requête [Socket]

```
{
        “Service”: “chat”,
        “Event”: “messageReceivedEvent”,
        “Recipient”: “...”,
        “Sender”: “...”,
        “MessageValue”: “...”,
        “Timestamp”: “...”
}
```

b. Réponse

```
{
        “Service”: “chat”,
        “Event”: “messageReceivedEvent”,
        “Acknowledge”: “ok”
}
```

3.10.3 Récupérer les canaux de discussion

a. Requête [Rest]:

GET [ip]:[port]/api/channels

b. Réponse

i. Succès:

```
{
        “ChannelId”: “...”,
        “Name”: “...”,
        “Members”:
        {
                {
                        “UserId”: “...”,
                        “UserName”: “...”
                },
                {...}
        }

}
```

ii. Erreur : {“error” : “some error”}

3.10.4 Joindre un canal de discussion

a. Requête [REST]:

POST [ip]:[port]/api/profile/{id}/channel

Body : {“ChannelId”: “...”}

b. Réponse

i. Succès: HTTP 200 code

ii. Erreur : {“error” : “some error”}

3.10.5 Créer un canal de discussion

a. Requête [REST]:

Post [ip]:[port]/api/channel/

Body : {“ChannelId”: “...”, “Name”: “...”}

b. Réponse
   i. Succès: {“ChannelId”: “...”, “Name”: “...”}
   ii. Erreur : {“error” : “some error”}

3.10.6 Quitter un canal de discussion

a. Requête [REST]:

DELETE [ip]:[port]/api/profile/{id}/channel/{id}

b. Réponse
   i. Succès: HTTP 200 code
   ii. Erreur : {“error” : “some error”}

3.11 Classement

3.11.1 Récupérer le classement

a. Requête [REST]:

GET [ip]:[port]/api/leaderboard?order_by=tournaments_won
GET [ip]:[port]/api/leaderboard?order_by=games_won

b. Réponse
   i. Succès:

```
{
        {
                “UserId”: “...”,
                “Username”: “...”,
                “Points”: “...”,
                “ProfilePicture”: “...”,
                “Rank”: “...”
        },
        { … }
}
```

   ii. Erreur : {“error” : “some error”}

3.12 Tournoi en ligne

3.12.1 Créer un tournoi

a. Requête [Socket] :

```
{
        “Service”: “tournament”,
        “Action”: “newTournament”,
        “Timestamp”: “...”
}
```

b. Réponse

i. Succès:

```
{
        {
                “Service”: “tournament”,
                “Event”: “newTournamentEvent”,
                “Acknowledge”: “ok”
        },
        { … }
}
```

ii. Erreur : {“error” : “some error”}

3.12.2 Joindre un tournoi

a. Requête [Socket] :

```
{
        “Service”: “tournament”,
        “Action”: “joinTournament”,
        “TournamentId”: “...”,
        “Timestamp”: “...”
}
```

b. Réponse

i. Succès:

```
{
        {
                “Service”: “tournament”,
                “Event”: “joinTournamentEvent”,
                “TournamentId”: “...”,
                “Acknowledge”: “ok”
        },
        { … }
}
```

ii. Erreur : {“error” : “some error”}

3.12.3 Notifier un client de l'arrivée d'un nouvel adversaire

a. Broadcast [Socket] :

```
{
        "Service": "tournament",
        "Event": "adversaryFoundEvent",
        "TournamentId": "...",
        "Adversary": {
                "UserId": "...",
                "Username": "..."
        }
        "Timestamp": "..."
}
```

3.12.4 Notifier les participants du début d'une nouvelle partie du tournoi

a. Requête [Socket] :

```
{
        "Service": "tournament",
        "Action": "newTournamentGame",
        "GamesLeft": "...",
        "Winner": "...",
        "Timestamp": "..."
}
```

b. Réponse :

```
{
        "Service": "tournament",
        "Event": "newTournamentGameEvent",
        "Acknowledge": "ok"
}
```

c. Broadcast [Socket] :

```
{
        "Service": "tournament",
        "Event": "newTournamentGameEvent",
        "Winner": "...",
        "Timestamp": "...",
        "GamesLeft": "..."
}
```

3.12.5 Notifier les participants de la fin d'un tournoi

a. Broadcast [Socket] :

```
{
        "Service": "tournament",
        "Action": "endOfTournamentEvent",
        "Winner": "...",
        "Timestamp": "..."
}
```

3.12.6 Résumé de tournoi

a. Broadcast [Socket] :

```
{
        “Service”: “tournament”,
        “Event”: “tournamentStatsEvent”,
        “Player1”: “...”,
        “Player2”: “...”,
        “Player3”: “...”,
        “Player4”: “...”,
        “GamesWonPlayer1”: “...”,
        “GamesWonPlayer2”: “...”,
        “GamesWonPlayer3”: “...”,
        “GamesWonPlayer4”: “...”,
        “Winner”: “...”,
        “PointsYouWon”: “...”,
        “AchievementsUnlocked”:
                {
                        {
                                “Id”: “achievementID”,
                                “Name”: “youWontBelieveThisAmazingAchievement”,
                                “Description”: “...”,
                                “NumberOfPointsRequired”: “...”,
                                “Type”: “...”
                        },
                        {…}
                },
        “Timestamp”: “...”
}
```

3.12.7 Récupérer les tournois en cours

a. Requête [REST]:
   GET [ip]:[port]/api/tournaments/
b. Réponse :
   i. Succès :

```
{
        “TournamentId”: “...”,
        “CreationDate”: “...”,
        “CreatorId”: “...”,
        “CreatorName”: “...”
}
```

   ii. Erreur : {“error” : “some error”}

3.12.8 Demander à joindre un tournoi à titre de spectateur

a. Requête [Socket] :

```
{
        “Service”: “tournament”,
        “Action”: “joinAsSpectactor”,
        “TournamentId”: “...”
}
```

b. Réponse :

i. Succès :

```
{
        “Service”: “tournament”,
        “Event”: “joinAsSpectactorEvent”,
        “Acknowledge”: “ok”
}
```

ii. Erreur : {“error” : “some error”}

3.12.9 Envoyer les statistiques d’un tournoi en cours au spectateur qui vient d’arriver

a. Broadcast [Socket]:

```
{
        “Service”: “tournament”,
        “Event”: “currentTournamentStatsEvent”,
        “Player1”: “...”,
        “Player2”: “...”,
        “Player3”: “...”,
        “Player4”: “...”,
        “GamesWonByPlayer1”: “...”,
        “GamesWonByPlayer2”: “...”,
        “GamesWonByPlayer3”: “...”,
        “GamesWonByPlayer4”: “...”,
        “Timestamp”: “...”
}
```

4. Références

Gupta, A. (24 Février 2014). REST vs WebSocket comparison and benchmarks. Tiré de http://blog.arungupta.me/rest-vs-websocket-comparison-benchmarks/

Jaitla, J. (5 Janvier 2015). Websockets vs REST: understanding the difference. Tiré de https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/

West, M. (18 Octobre 2013). An introduction to WebSockets. Tiré de http://blog.teamtreehouse.com/an-introduction-to-websockets