Équipe 204

Projet d’évolution d’un logiciel
Protocole de communication

Version 1.0

Historique des révisions

Date	Version	Description	Auteur
aaaa-mm-jj	x.x	<Détails précis du travail effectué>	<Nom>
2021-02-03	1.0	Premier jet du document de protocole de communication	Antoine Morcel Maxime Fecteau
2021-02-09	1.1	Continuation de la première itération du protocole de communication	Antoine Morcel Maxime Fecteau

Table des matières

Protocole de communication

1. Introduction

Ce document contient l'information relative à la librairie de communication client-serveur utilisée ainsi que la description de l'interface du serveur. De plus, il contient la description des paquets utiles à chaque fonctionnalité. Nous décrivons premièrement ce que les clients peuvent demander au serveur, ensuite nous décrivons la structure des paquets JSON que les clients doivent envoyer au serveur pour pouvoir communiquer.

2. Communication client-serveur

La communication client-serveur est faite avec l'aide de la librairie Socket.io. Nous utiliserons des structures JSON la majorité du temps pour pouvoir envoyer les informations nécessaires. Pour tout ce qui est gestion de compte nous utilisons un api http classique.

2.0 Gestion des sockets

- Ouverture du socket côté client et connexion au serveur lorsque l'utilisateur se connecte à l'application avec son compte utilisateur.
- Fermeture du socket côté client et déconnexion au serveur lorsque l'utilisateur se déconnecte ou ferme l'application.

2.1 Gestion des salons

- Rejoindre un salon de jeu. Le client avise le serveur que le joueur rejoint un salon.
- Quitter un salon de jeu. Le client avise le serveur que le joueur quitte un salon.
- Statut des salons de jeu publics. Le client demande au serveur une liste des parties publiques qui sont disponibles.
- Inviter un utilisateur à un salon de jeu. Le joueur envoie une invitation à un autre joueur qui est dans sa liste d'amis et qui est en ligne.

2.2 Gestion de messagerie

- Envoyer un message au salon de jeu courant.
- Envoyer un message à un utilisateur
- Obtenir l'historique de message à un utilisateur. Le client demande les derniers messages transmis entre l'utilisateur qui demande et l'utilisateur spécifié. Entre autres, ceci est pour voir l'historique des messages envoyés à ces amis.

2.3 Gestion de compte (api Http, contrairement au reste qui utilise socket.io)

- Créer un nouveau compte
- Connexion à un compte
- Déconnexion de l'utilisateur
- Changer le mot de passe d'un compte à travers un système de courriel électronique
- Obtenir les informations du compte courant (Nom d'usager, nombre de parties jouées, etc)
- Modifier l'information de compte (nom d'usager, courriel, nom)

2.4 Gestion des dessins

- Créer un trait de dessin. La création d'un trait de dessin se fait en trois étapes distinctes qui ont chacune leurs propre appel de socket (socket.on) pour permettre la transmission fluide et en directe du dessin aux autres clients.
    - Premièrement, lorsque le client commence à faire un trait (onMouseDown), celui relaie l'information de la balise svg créer au serveur, qui lui, la renvoie aux autres clients.
    - Deuxièmement, lorsque le client exécute des mouvements avec son trait, il relaie les modifications

de cette balise svg au serveur, qui lui, la renvoie aux autres clients.
    - Troisièmement, lorsque le client finit de dessiner son trait, il le relaie au serveur, qui lui, relaie aux autres clients que ce trait est terminé et que la balise svg ne subira plus de modifications.
- Effacer un trait de dessin. Le client envoie les informations relatives aux traits qui sont effacés par le dessinateur au serveur, qui lui renvoie ces informations aux autres clients.
- Défaire. Le client qui est en train de dessiner indique au serveur qu'il a défait une action.
- Refaire. Le client qui est en train de dessiner indique au serveur qu'il a refait une action.

2.5 Gestion modes jeux

- Début de partie
- Fin de partie
- Sélection de mot pour le dessinateur (s'il n'y a pas de joueur virtuel dans l'équipe active)
- Deviner (la logique côté serveur va changer selon le mode de jeu)

2.6 Joueur Virtuel

Les joueurs virtuels communiquent avec les clients de manière unidirectionnelle. Ils doivent pouvoir:

- Dessiner des traits (voir la section 2.4)
- Envoyer des messages dans les salons de jeu

3. Description des paquets

Ci-dessous sont la structure des paquets (qui sont sujet à changement) pour les différentes fonctionnalités du serveur que les clients peuvent utiliser.

3.1 Gestion des salons

- Pour rejoindre un salon de jeu on a :

```
{
    "roomToJoin": "roomId"
}
```

- Pour quitter un salon de jeu on a :

```
{

    "roomToQuit": "roomId"

}
```

- Pour le statut des salons publics on a :

```
{
    "rooms" : [ {
        "room" :"id",
        "roomName" : "name1",
        "players" :
        [
            {
                "playerName" : "name1"
            },
```

```
            {
                "playerName" : "name2"
            }
        ]
    }]
}
```

- Pour inviter un utilisateur à un salon de jeu on a :

```
{
    "invitedPlayer" : "name",
    "roomInvited" : "roomId"
}
```

3.2 Gestion de messagerie

- Pour envoyer un message au salon courant on a :

```
{
      "user" : "user",
      "message" : "message",
      "timestamp" : "timestamp"
}
```

- Pour envoyer un message à un utilisateur on a :

```
{
    "sendTo" : "recipientUsername",
    "message" : "message"
}
```

- Pour obtenir l'historique de message à un utilisateur on a :

```
{
    "userToGetHistory" : "username"
}
```

le client reçoit

```
{
    "messages" : [
        {
            "user" : "user",
            "timestamp": "timestamp",
            "content": "content"
        },
        {
            "user" : "user",
```

```
            "timestamp" :"timestamp",
            "content" :"content"
        }
    ]
}
```

3.3 Gestion de compte

- Pour créer un nouveau compte : POST /api/database/auth/register avec comme corps

```
export interface Register {
    name: string;
    username: string;
    email: string;
    password: string;
    passwordConfirm: string;
}
```

- Pour se connecter à un compte : POST /api/database/auth/login avec comme corps

```
export interface login {
    username: string;
    password: string;
}
```

- Pour se déconnecter : DELETE /api/databse/auth/logout avec comme corps

```
export interface logout{
    refreshToken: string;
}
```

- Pour obtenir les informations du compte : GET /api/database/account avec dans les headers :

```
authorization : jwt_token
```

- Pour supprimer le compte : DELETE /api/database/account avec dans les headers :

```
authorization : jwt_token
```

- Pour modifier le compte POST /api/database/account avec dans les headers :

```
authorization : jwt_token
```

et comme corps

```
export interface Account {
  name: string;
  username: string;
  email: string;
}
```

- Pour réinitialiser le mot de passe GET /api/database/account/reset avec dans les headers :

```
authorization : jwt_token
```

- Ce GET va envoyer un courriel contenant un lien vers l'application avec un nouveau jwt_token généré à partir d'une clé privée de réinitialisation de mot de passe.
- Pour finaliser la réinitialisation on passe par POST /api/database/account/reset avec dans les headers:

```
authorization : jwt_reset_pwd_token
```

et comme corps

```
export interface ResetPwd{
  newPassword: string;
  newPasswordConfirm: string;
}
```

3.4 Gestion des dessins

- Pour que le client du dessinateur communique au serveur le trait dessin:

```
{
    "lineToDraw": "svgOfDrawedLine"
}
```

- Pour que le client du dessinateur communique au serveur le trait effacer:

```
{
    "lineToErase": "svgOfErasedLine"
}
```

- Pour gérer les undo/redo du client, celui-ci envoie la commande au serveur:

```
{
    "commandToPass": "command"
}
```

3.5 Gestion modes jeux

- Pour que le serveur envoie le choix de mots au client (dessinateur)

```
{
    "wordChoices": [
        {
            "firstWord": "word"
        },
        {
            "secondWord": "word"
        },
        {
            "thirdWord": "word"
        }
    ]
}
```

- Pour que le dessinateur choisisse son mot et informe le serveur:

```
{
    "wordChosen": "word"
}
```

3.6 Joueur Virtuel

- Pour envoyer un message au client à partir du serveur, le joueur virtuel utilise:

```
{
    "message" : "message"
```

```
}
```

- Pour envoyer les trait à dessiner du serveur, le joueur virtuel utilise:

```
{
    "lineToDraw": "svgOfDrawedLine"
}
```