

**TD5 - WOB**

**Compte-rendu**

**Avant-propos :**

Le but de ce TD est d’implémenter les fonctionnalités qui ont été faites en TD3 et en TD4 mais cette fois-ci avec socket.io.

Nous allons donc voir les détails de ce qui a été implémenté, ainsi que les difficultés rencontrées.

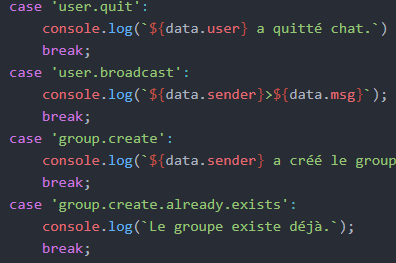
De plus, au TD précédent, il m’a manqué du temps pour implémenter la base de données. J’explique après la partie sur le code des fonctionnalités du chat comment j’ai procéder pour élaborer cette base de données.

**Clarté du code :**

En vue d’améliorer la lisibilité du code par rapport à ce qui a été fait en TD3 et TD4, j’ai essayé de l’aérer un maximum et de commenter en détails les concepts pour les expliquer. Dans le switch du serveur, voilà le procédé qui a été adopté :



Chaque « case » du switch ne comporte que deux lignes pour aérer au maximum. A chaque « case », on fait appel à la fonction appropriée. Les fonctions sont toutes définies en dessous du code. Ainsi, la personne qui veut y accéder pourra distinguer les différentes parties et observer seulement ce qui l’intéresse.



Côté client, il a été décidé de ne pas utiliser de fonction cette fois-ci, puisque le contenu des « case » est léger. Cela évite de limiter un peu plus la longueur du code et par conséquent de gagner en lisibilité.

La principale différence que l’on va observer ici sera l’utilisation du « socket.emit » plutôt que « socket.write » que l’on utilisait dans les TDs précédents.

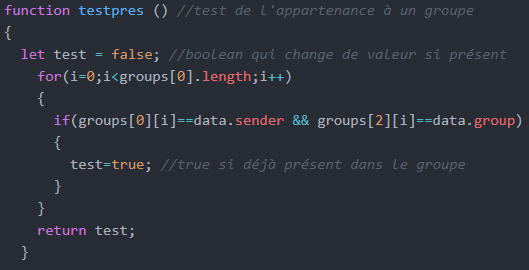
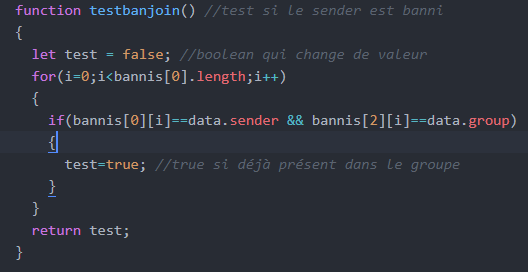
**Guide de l’utilisateur :**

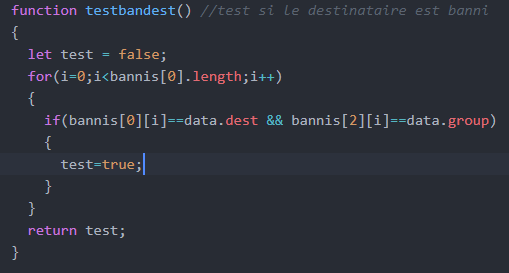
Voici les commandes qu’un utilisateur peut utiliser lorsqu’il est connecté :

|  |  |  |
| --- | --- | --- |
| **Intitulé** | **Effet de la commande** | **Commande** |
| Broadcast | Envoyer un message à tous les utilisateurs | b; |
| List | Afficher la liste de tous les utilisateurs | ls; |
| Quit | Se déconnecter | q; |
| Create Group | Créer un groupe | cg; |
| Join | Joindre un groupe | j; |
| Broadcast Group | Diffuser un message à tous les utilisateurs d’un groupe | bg; |
| List members | Afficher les membres d’un groupe | members; |
| List messages | Afficher l’historique des messages d’un groupe | messages; |
| List groups | Afficher la liste des groupes | groups; |
| Leave | Quitter un groupe | leave; |
| Invite | Inviter un utilisateur dans un groupe | invite; |
| Kick | Exclure un utilisateur d’un groupe | kick; |
| Ban | Bannir un utilisateur d’un groupe | ban; |
| Unban | Débannir un utilisateur d’un groupe | unban; |
| States | Afficher les opérations effectuées dans un groupe | states; |

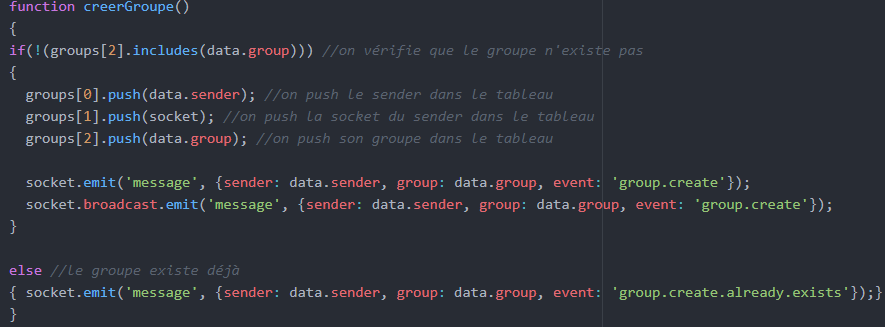
Ci-dessous, j’expliquerai en détails comment les fonctionnalités voulues ont été codées.

Voici trois fonctions primordiales qui ont été créées. Il s’agit de fonction qui vont vérifier si un utilisateur est présent dans un groupe, si l’utilisateur expéditeur est banni d’un groupe et si l’utilisateur destinataire est banni d’un groupe. Elles parcourent les tableaux des utilisateurs et des utilisateurs bannis et renvoient un Boolean selon si l’utilisateur est présent ou pas. Elles seront très utiles pour coder les fonctionnalités du chat.





**Create Group**



On distingue ici deux cas : soit le groupe existe déjà et on renvoie à l’utilisateur qu’il ne peut pas créer un groupe qui existe déjà, soit le groupe n’existe pas et il peut le créer. Pour cela, on parcourt le tableau qui contient les groupes, et on ajoute le nom du nouveau groupe s’il n’existe pas déjà.

**Join**



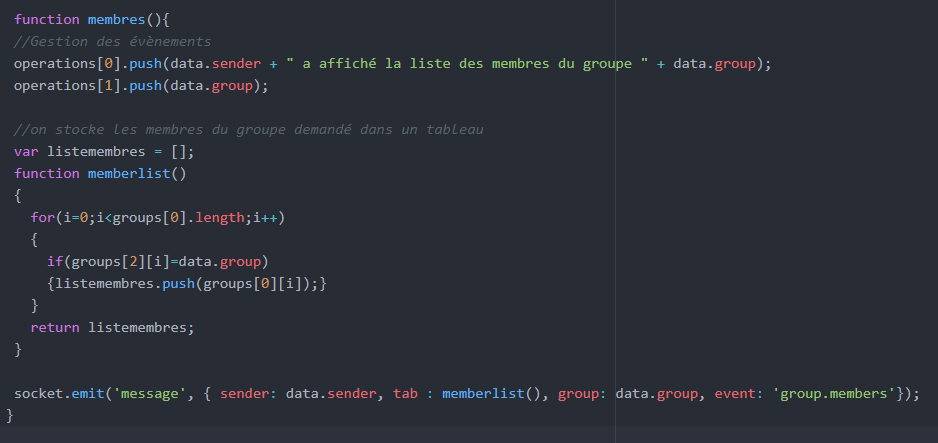
La fonctionnalité join doit gérer plusieurs cas de figure. Avec la fonction testpres() décrite précédemment, on teste si l’utilisateur appartient déjà au groupe qu’il souhaite joindre, et on lui renvoie le message adéquat. Avec cette même fonction couplée à testbanjoin() pour voir si l’utilisateur est banni du groupe qu’il souhaite joindre ainsi qu’avec un test pour savoir si le groupe existe on distingue un autre cas. On ajoute alors l’utilisateur dans le tableau associé à son nom de groupe. Le dernier cas que l’on doit gérer est de voir si le groupe n’existe pas. On renvoie là aussi le message approprié.

**Broadcast Group**



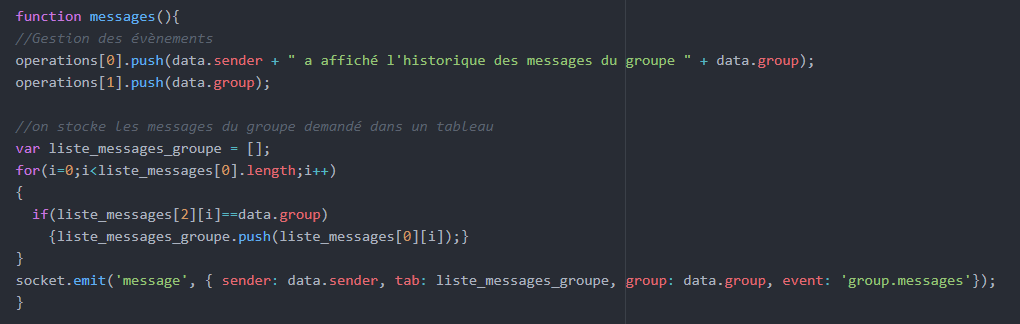
Pour cette fonctionnalité, on test si le groupe dans lequel le message doit être envoyé existe et si l’utilisateur fait partie du groupe. Si c’est le cas, on envoie à tous les utilisateurs du tableau qui ont pour nom de groupe celui qui a été mentionné dans la commande. Si on ne se trouve pas dans ce cas, on refuse l’envoie d’un message.

**List members**



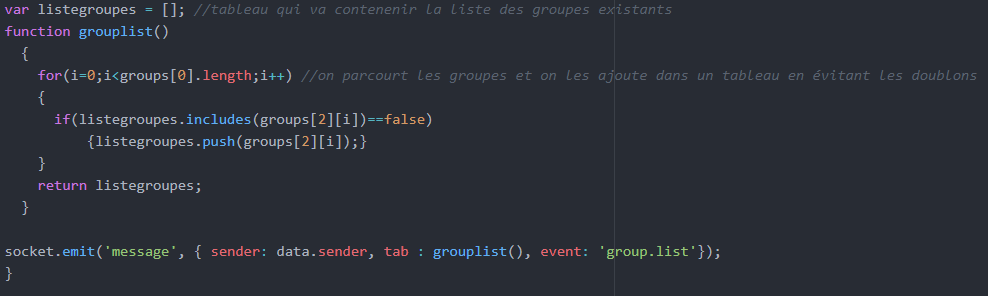
Pour affiché la liste des membres appartenant à un groupe, j’ai décidé d’adopter la stratégie suivante : créer un nouveau tableau listemembres et stocker dans ce tableau tous les utilisateurs associés à endroit dans le tableau groups. Après le parcours de ce tableau terminé, on peut afficher listemembres.

**List messages**



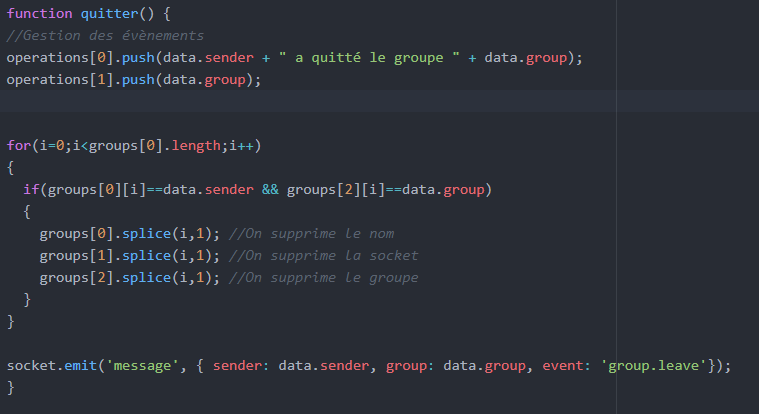
Pour cette fonctionnalité, on crée un tableau liste\_messages\_groupe. Dans la fonctionnalité broadcast, on push dans un tableau liste\_messages chaque message qui a été envoyé, quel que soit l’utilisateur, quel que soit son groupe. Ainsi, on peut parcourir ce tableau et on push dans list\_messages\_groupe uniquement les messages correspond au nom de groupe mentionné dans la commande. Enfin, on peut le renvoyer pour pouvoir l’afficher.

**List groups**



On crée un tableau listegroupes qui contiendra la liste des groupes. On parcourt le tableau groups et on ajoute le nom du groupe lu dans listegroupes. On vérifie à chaque itération que le tableau listegroupes ne contient pas déjà le groupe pour ne pas avoir de doublons et on affiche la liste.

**Leave**



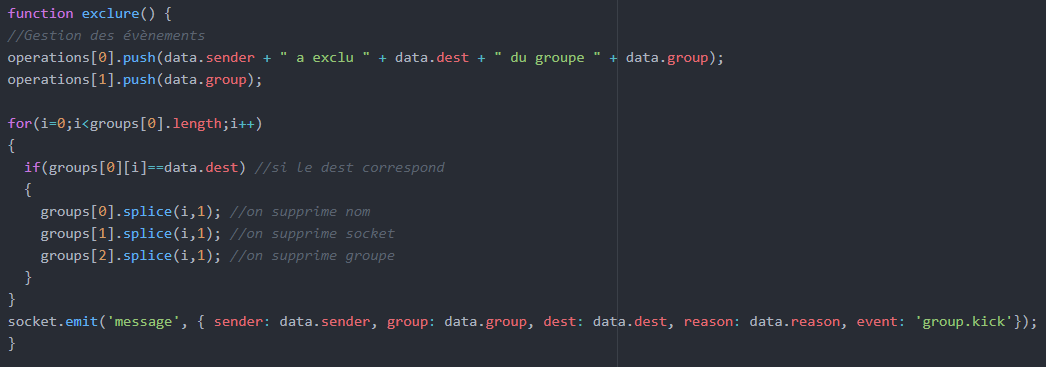
Pour la fonctionnalité « leave », il faut ici parcourir le tableau groups. On regarde la correspondance avec l’utilisateur qui demande à partir et son groupe, et on le supprime du tableau groups avec la commande splice().

**Invite**



Pour inviter un utilisateur, on vériie que l’utilisateur destinataire n’est pas banni du groupe demandé. Si tel est le cas, alors on peut l’ajouter dans le tableau groups avec le nom du groupe associé.

**Kick**



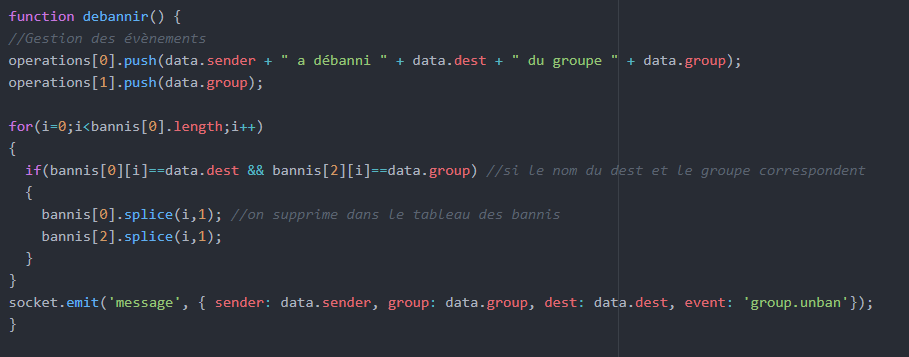
Il y a une grande ressemblance ici avec la fonctionnalité « leave ». La seule différence et qu’au lieu de regarder la correspondance avec l’utilisateur qui envoie la commande, on regarde la correspondance avec l’utilisateur destinataire. On le supprime du tableau groups avec splice().

**Ban**



Ici, en plus d’exclure l’utilisateur destinataire en le supprimant du tableau groups, on l’ajoute dans un nouveau tableau nommé bannis. Quand l’utilisateur voudra joindre un groupe ou quand il sera invité dans un groupe, on vérifiera d’abord qu’il n’est pas présent dans le tableau des bannis.

**Unban**



Pour débannir un utilisateur, il suffit simplement de le supprimer du tableau bannis avec la commande splice().

**States**

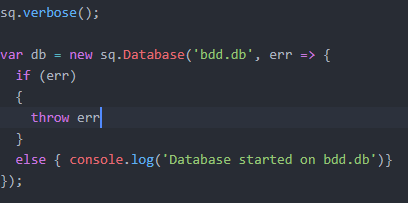


Pour la fonctionnalité « states », on crée un tableau nommé operations. Dans chaque fonctionnalité codée jusqu’ici, on remarquera à chaque fois, on push la description du résultat de la commande ainsi que le groupe dans lequel elle a été faite. On pourra alors parcourir operations en regardant le groupe qui nous intéresse et afficher toutes les événement qui ont eu lieu dans ce groupe.

**Base de données :**

J’ai pu implémenter la fonctionnalité de base de données mentionnée dans le TD4.

Je commence par utiliser require pour pouvoir utiliser les base de données.



Le code ci-contre me permet de créer une base de données et de vérifier qu’elle a bien été crée. J’affiche que la base de données a bien été créée si c’est le cas et sinon j’affiche une erreur.

En exécutant le programme, on remarque que la base de données a bien été créée.



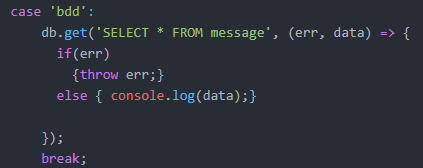
On peut facilement le vérifier en vérifiant dans le dossier de création. Ici, le fichier bdd.db a bien été créé après l’exécution de la commande.



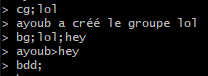
En utilisant CREATE TABLE, il est possible de créer une table nommée message qui va contenir les messages envoyés, les noms des utilisateurs qui ont envoyé le message ainsi que le groupe dans lequel il a été envoyé.

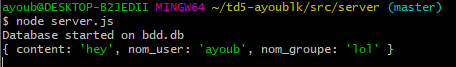


Dans la fonctionnalité de broadcast, on peut alors mettre la ligne de code ci-dessous avec un INSERT INTO. Dans la table, on insère le message, l’expéditeur du message et le groupe dans lequel le message a été envoyé. A chaque fois qu’un utilisateur enverra un message, la base de données sera alors actualisée.



On crée un nouveau case dans le switch qui va permettre à l’utilisateur de demander l’affichage de la base de données. On fait ici un SELECT pour récupérer les informations que l’on souhaite.

Ici, on crée un groupe dans lequel l’utilisateur envoie le message « hey ». Ensuite, il utilise la commande bdd;.

La base de données avec les informations demandées est alors affichée.