Introduction

Le projet consiste à créer un serveur de messagerie instantanée en utilisant Node.js avec Express.js pour le backend et React.js pour le frontend. Le serveur accepte des connexions multiples, permet de créer, gérer et supprimer des canaux de discussion, et persiste les messages et les canaux. La communication entre le client et le serveur se fait via Socket.IO.

Architecture du Système

1. Frontend (React.js)

L'interface utilisateur est construite avec React.js. Elle est composée des éléments suivants :

- **Page de connexion :**
  - Permet à l'utilisateur de se connecter ou de créer un compte.
  - Propose un champ pour entrer un pseudonyme.
- **Page principale :**
  - **Liste des conversations :** Une barre latérale affichant les canaux disponibles avec une barre de recherche pour filtrer les résultats.
  - **Chat :** À droite, l'utilisateur peut voir le contenu du canal sélectionné et envoyer des messages.
  - **Console de commandes :** Un champ au-dessus de la partie chat où l'utilisateur peut entrer des commandes IRC.

2. Backend (Node.js + Express.js)

Le backend est responsable de la gestion des connexions des utilisateurs, des canaux, des messages et de la persistance des données.

- **Serveur Express.js :**
  - Gère les routes HTTP pour la connexion et la gestion des canaux.
  - Gère la communication en temps réel avec Socket.IO.
- **Socket.IO :**
  - Permet la communication en temps réel entre le serveur et le client.
  - Gère les événements tels que la connexion d'un utilisateur, l'entrée/sortie d'un canal, et l'envoi de messages dans les canaux.
- **Persistance des données :**
  - Les données seront stockées dans une base de données (ex. SQLite ou MongoDB), y compris les informations sur les canaux, les messages et les utilisateurs.
  - Les messages et les canaux sont conservés de manière persistante.

3. Base de données

La persistance des données est gérée à travers une base de données. Les éléments suivants sont stockés :

- **Utilisateurs :**
  - Pseudonymes des utilisateurs.

    - Historique des messages privés.
- **Canaux :**
    - Liste des canaux existants.
    - Liste des utilisateurs dans chaque canal.
    - Messages dans chaque canal.
- **Messages :**
    - Historique des messages envoyés dans chaque canal.
    - Messages privés entre utilisateurs.

Fonctionnalités

1. Commandes client

L'utilisateur peut interagir avec le serveur via les commandes IRC suivantes :

- **/nick [nickname]** : Définit le pseudonyme de l'utilisateur.
- **/list** : Affiche la liste des utilisateurs.
- **/create [channel]** : Crée un canal avec le nom spécifié.
- **/delete [channel]** : Supprime le canal spécifié.
- **/join [channel]** : Permet à l'utilisateur de rejoindre un canal.
- **/quit [channel]** : Permet à l'utilisateur de quitter un canal.
- **/users** : Liste les utilisateurs présents dans le canal.
- **/msg [nickname]** : Crée un message privé avec l'utilisateur spécifié.

2. Gestion des canaux

- Création, suppression, et gestion des canaux via des commandes.
- Chaque canal peut avoir une liste d'utilisateurs et un historique de messages.

3. Messages

- Les messages envoyés dans un canal sont visibles par tous les utilisateurs du canal.
- Les messages privés sont envoyés directement d'un utilisateur à un autre.

4. Comportement en temps réel

- Utilisation de Socket.IO pour une communication en temps réel entre le client et le serveur.
- Affichage en direct des messages entrants, ainsi que des notifications de l'entrée et de la sortie des utilisateurs dans les canaux.

5. Persistance des données

- Les canaux et messages sont stockés dans une base de données pour garantir la persistance.
- Les utilisateurs peuvent rejoindre et quitter les canaux à tout moment, et les messages sont enregistrés de façon permanente.

Technologies utilisées

- **Frontend** : React.js, Socket.IO-client, TailwindCss pour le style.
- **Backend** : Node.js, Express.js, Socket.IO.
- **Base de données** : MongoDB pour stocker les données de canaux, messages, et utilisateurs.

Sécurité et Améliorations futures

- **Notifications** : Système de notifications quand on reçois un message.
- **Améliorations UI/UX** : Ajouter des fonctionnalités comme la personnalisation du profil.
- **Tests** : Ajouter des tests unitaires et d'intégration pour les fonctionnalités critiques du serveur et du client.
- **Channels** : Mise en place de la partie Front des channels pour qu’ils soient utilisables.