ChatApp

Nous allons créer une application de chat type groupe whats app / discord.

Fonctionnalités.

- Se connecter à un serveur de discussion
- Envoyer un message à tout les participants de la discussion.
- Afficher les messages entrants.

Fonctionnement

Une application de chat néccessite l'utilisation du protocole TCP pour la réception des messages en temps réel.

Habituellement en developpement web vous utilisez uniquement le HTTTP pour récupérer des données quand **VOUS LE SOUHAITEZ**. La réponse du serveur arrive au client uniquement si une requete du client est faite. Dans une application de chat les messages peuvent arriver à n'importe quel moments, le client ne sait donc pas quand envoyer une requete pour reçevoir une réponse contenant les messages.

Une solution serait d'envoyer à intervale régulié une requête au serveur pour se tenir informé des messages entrant, c'est une très mauvaise solution car le client spam son réseau internet sans être assuré de reçevoir un message à chaque requête.

Non, ce qu'il nous faudrait c'est un moyen de se connecter **UNE** fois au serveur et ensuite attendre que le serveur envoie un message au client. Cette solution c'est le protocole TCP.

Les socket TCP

Un socket est un point de connexion entre un client et un serveur.

Connexion

Le socket client envoie une requete au socket serveur qui envoie une réponse au client, on appel cette étape le *handshake* ou la connexion.

Maintient de la connexion

Une fois que le client est connecté au serveur, le serveur va ecouter les messages que le client lui envoi et les renvoyer à TOUT LES CLIENTS. On appel ça le broadcasting.

Cahier des charges

- Une serveur nodejs qui retourne tout les messages qu'il reçoit aux clients.
- Un client react qui envoie un message au serveur quand l'utilisateur rempli un formulaire d'envoi.
- Le client React écoute l'envoi de message du serveur pour ensuite afficher tout les messages.

Mise en place des projets client et serveur

Notre application est en deux parties : un client React et un serveur NodeJS.

0. Créez un dossier vide appelé ChatApp
1. Créez un projet react vite vierge nommée client
2. Créez un dossier vide dans ChatApp appelé serveur

Dans le projet client, les fichiers main.jsx et App.jsx sont comme ceci.

main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import {App} from './App.jsx'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

App.jsx

```
export function App(){

    return (
    <div>
        <h1>ChatApp</h1>
    </div>
    )
}
```

Le serveur NodeJS

1. Installez le module socket.io dans le dossier serveur

```
cd serveur
npm init
npm install socket.io
```

Dans le dossier serveur ajoutez un script *main.js*.

serveur/main.js

```
const { createServer } = require('node:http');
const io = require('socket.io');

// Création du serveur HTTP
const httpServer = createServer();

// Liaison du socket TCP avec le serveur HTTP pour le handshake
const server = io(httpServer,{ cors : { origin : "*"} });
// CORS Autorise le serveur et le client à avoir le même nom de domaine à savoir locahos

server.on('connection', (client) => {
    // Quand un client se connecte
    console.log('a user connected : '+ client.id);

    // J'ecoute l'evenement send_msg
    client.on("send_msg",(data)=>{
        // Quand je reçois un message d'un client
        console.log(data);

        // J'envoie le message à tout les clients
        // server.emit("new_msg",{
        //     content : message.content,
        //     author : message.author,
        //     date : message.date
        // });
        server.emit("new_msg",data);
    });
});

httpServer.listen(3000, () => {
  console.log('server running at http://localhost:3000');
});

function createMsg(author,content,date = Date.now()){
    return {
        author,
        content,
        date
    };
}
```

Pour lancer le serveur tapez :

```
node main.js
```

Connexion au serveur

La connexion au serveur par le client se fait via la fonction `io()` du module `socket.io-client` .

```
import {io} from "socket.io-client";

const socket = io("localhost:3000");
```

Il est néccessaire d'installer le module `socket.io-client` dans le projet React.
Dans le dossier client généré par vite tapez :

```
npm install socket.io-client
```

Envoi de message

Le serveur reçoit les messages via l'évenement `new_msg` .

Le client React devra émettre cette évenement pour envoyer un message, de cette manière :

```
// Dans React on ferra
const msg = {
      content : "Salut je suis un message",
      date : Date.now(),
      author : "Martin"
}
socket.emit("send_msg",msg);  // Envoi du message au serveur
```

Réception de message

La réception de message du serveur par le client se fait via l'évenement `new_msg` .
Le client React écoutera les nouveaux messages de cette manière.

```
socket.on("new_msg",(newMessage)=>{
      console.log(newMessage)
});
```

Le client React

Plan d'action

- Mettre en place le HTML
- Fournit le socket à au module `App` .
- Brancher le formulaire de connexion qui affiche le formulaire d'envoi de message lorsque le client est connecté.
- Brancher le forumlaire d'envoi de message qui envoi un message au serveur lorsque le client submit. Pour ceci il utilise l'évenement `send_msg` .
- Ecouter l'évenement `new_msg` qui met a jour un state messages. Le state messages contient tout les messages reçu du serveur
- Afficher le state messages via un map

Le HTML

Le HTML du client est composée de deux formulaires et une liste de messages.

```
export function App(){

return (
    <div>
      <h1>Client Chat</h1>
      <form >
        <input type="text" id="author" />
        <button >Se connecter</button>
      </form>

      <form >
        <input type="text" id="message" />
        <button>Envoyer</button>
      </form>

      <div>
      </div>
    </div>
  )
}
```

Fournir le socket à App

Connectez vous au serveur dans `main.jsx` et passez le socket en prop à App.

main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client.js'
import {App} from './App.jsx'

import {io} from "socket.io-client";

const socket = io("localhost:3000");

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App socket={socket}/>
  </React.StrictMode>,
)
```

App possède maintenant le socket de connexion vers le serveur.

Ce socket permet d'écouter des évenements provenent du serveur et d'émettre des évenements vers le serveur.

Pour rappel :

- Pour envoyer un message il faut emmettre l'évenement `send_msg` avec la fonction `socket.emit()`
- Pour recevoir les messages il faut écouter l'évenement `new_msg` avec la fonction `socket.on()`

Fomulaire de connexion

Lorsque le formulaire de connexion est submit je défini le nom de l'utilisateur dans un state et j'affiche le forumlaire d'envoie de message

```
import { useState } from "react";

export function App({socket}){
  const [author,setAuthor] = useState(null);

  function handleConnexion(event){
    event.preventDefault();

    const author = event.target.querySelector("#author").value;
    setAuthor(author);
  }



  return (
    <div>
      <h1>Client Chat</h1>
      <form onSubmit={handleConnexion} hidden={author != null}>
        <input type="text" id="author" />
        <button >Se connecter</button>
      </form>

      <form hidden={author == null}>
        <input type="text" id="message" />
        <button>Envoyer</button>
      </form>

      <div>
      </div>
    </div>
  )
}
```

Grâce à l'attribut html hidden j'affiche les forumlaires en fonction de si l'utilisateur est connecté ou non.

Envoi de message au serveur

Pour envoyer un message il faut émettre l'évenement "send_msg" accompagné d'un objet message contenant : contenu, auteur et date.

L'émission de cet evenement se fait via la fonction `socket.emit()` .

```
const msg = {
    content : "Salut ça va ?",
    date : Date.now(),
    author : "Massinissa"
}

socket.emit("send_msg",msg);  // Envoi du message au serveur
```

Ajoutez une fonction `sendMessage` qui est appelée quand le formulaire d'envoi de message est `submit` . C'est dans cette fonction que le message sera envoyé.

```
import { useState } from "react";

export function App({socket}){
    const [author,setAuthor] = useState(null);

    function sendMessage(event){
        event.preventDefault();

        const content = event.target.querySelector("#message").value;
        const msg = {
            content : content,
            date : Date.now(),
            author : author
        }

        socket.emit("send_msg",msg);  // Envoi du message au serveur
    }

    function handleConnexion(event){
        event.preventDefault();

        const author = event.target.querySelector("#author").value;
        setAuthor(author);
    }

    return (
    <div>
        <h1>Client Chat</h1>
        <form onSubmit={handleConnexion} hidden={author != null}>
        <input type="text" id="author" />
        <button >Se connecter</button>
        </form>

        <form onSubmit={sendMessage} hidden={author == null}>
        <input type="text" id="message" />
        <button>Envoyer</button>
        </form>

    </div>
    )
}
```

Si tout fonctionne, lorsque l'utilisateur tape un message dans le formulaire et l'envoi, vous devriez voir un message dans la console du serveur node.

```
{ content: 'Salut', date: 1708507549831, author: 'Massinissa' }
```

Réception des messages

Lorsque un utilisateur envoi un message le serveur le retourne à tout les utilisateurs connectée.

La réception des messages retournée se fait via l'écoute de l'évenement `new_msg` .

Pour écouter un evenement du socket j'utilise la fonction `socket.on()` .

```
socket.on("new_msg",(newMessage)=>{
    // C'est ici que je récupère le nouveau message.
    console.log(newMessage);
});
```

Pour affichez les messages au fur et à mesure qu'il arrives j'ai besoin d'un state nommé `messages` qui est un tableau d'objet message.

Je crée ce state dans App.

```
const [messages,setMessages] = useState([]);
```

Puis j'ajoute l'écoute de l'évenement `new_msg` dans la fonction de connexion `handleConnexion` .

Cette évenement fournit un nouveau message que je rajoute dans l'etat `messages` avec setMessage.

```
function handleConnexion(event){
    event.preventDefault();

    const author = event.target.querySelector("#author").value;
    setAuthor(author);

    socket.on("new_msg",(newMessage)=>{

        setMessages((prevMessages)=>[
            ...prevMessages,
            newMessage
        ]);

    });
  }
```

Et j'affiche tout les messages dans le html grâce à un map.

```
import { useState } from "react";

export function App({socket}){
  const [author,setAuthor] = useState(null);
  const [messages,setMessages] = useState([]);

  function sendMessage(event){
    event.preventDefault();

    const content = event.target.querySelector("#message").value;
    const msg = {
      content : content,
      date : Date.now(),
      author : "Massinissa"
    }

    socket.emit("send_msg",msg);
  }

  function handleConnexion(event){
    event.preventDefault();

    const author = event.target.querySelector("#author").value;
    setAuthor(author);

    /**
     * Quand un message arrive (event new_msg)
     * Je met à jour l'etat messages
     */
    socket.on("new_msg",(newMessage)=>{
      setMessages((prevMessages)=>[
        ...prevMessages,
        newMessage
        ]);
    });
  }

  /**
   * Je génère mes balises messages
   */
  const messageElements = messages.map((message,i)=>{
    const date = (new Date(message.date)).toUTCString();
    return (
```

```
      <div key={i} className="message">
        <p>{message.content}</p>
        <p>{message.author}</p>
        <p>{date}</p>
      </div>

    );
  });

  return (
    <div>
      <h1>Client Chat</h1>
      <form onSubmit={handleConnexion} hidden={author != null}>
        <input type="text" id="author" />
        <button >Se connecter</button>
      </form>

      <form onSubmit={sendMessage} hidden={author == null}>
        <input type="text" id="message" />
        <button>Envoyer</button>
      </form>
      {/* J'affiche tout les messages */}
      <div>
        {messageElements}
      </div>
    </div>
  )
}
```

> Pourquoi utilisez cette syntaxe alternative.
>
> ```
> setMessages((prevMessages)=>[...prevMessages, newMessage]);
> ```
>
> La syntaxe alternative des fonctions setState permet d'utiliser la valeur précedente d'un etat pour le mettre à jour.
>
> Je suis dans une fonction callback je n'ai donc pas accès à la dernière version de messages. React enregistre les fonctions callback uniquement lors du premier affichage du composant. Pour React, messages sera toujours égal à une tableau vide. J'utilise donc une fonction en paramètre de setMessages pour. Cette syntaxe alternative permet de récuperer à coup sur l'etat messages le plus récent.