EURÊKA solutions

CHAT EN NODEJS

Emilien GERMEZ

EURÊKA SOLUTIONS - Siège Social : 52 rue de Dornach - 68120 PFASTATT
Tél. 03 89 51 20 20 - Agences : Mulhouse - Nantes www.eureka-solutions.fr
S.A.S. au capital de 330.000 € - RCS Mulhouse B 439 442 583 - N° Identification TVA FR 82 439 442 583 - APE 5829 C

EURÊKA solutions

Sommaire

1. Diagramme de flux 3

2. Installation de node et de NPM 3

3. Création du projet 4

4. Création du server 6

5. Création du client (Ligne de commande) 13

5.1 Architecture de notre chat 14

5.2 Installer la bibliothèque cliente 14

5.3 Lancer les programmes 16

5.4 Explication des principales instructions 17

5.5 Résultat 18

6. Création du client Web 19

6.1 Résultat 29

7. Pouvoir envoyer des messages PRIVES 30

7.1 Architecture 30

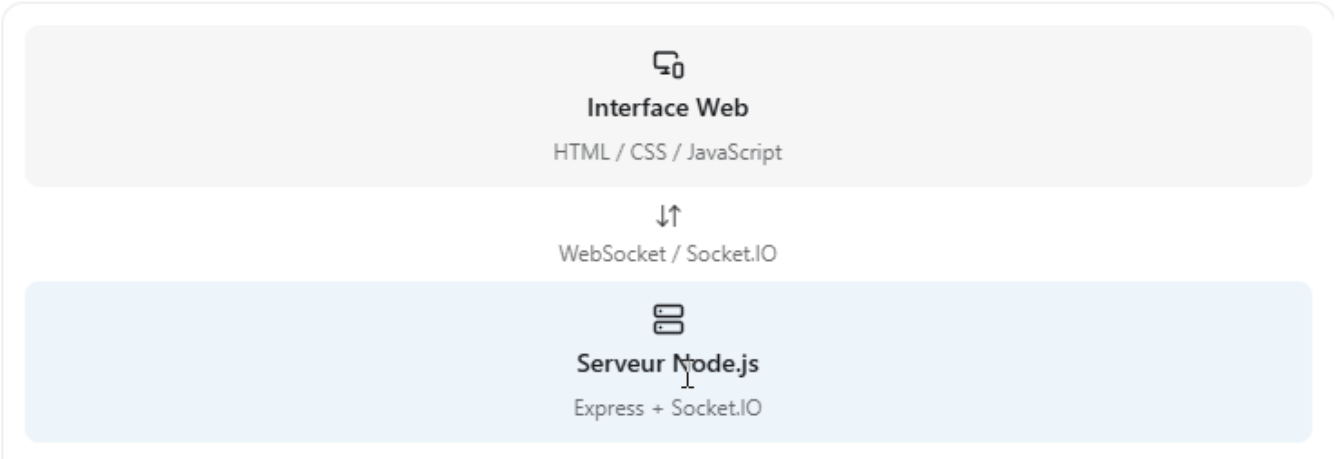
7.2 Création server2.js 30

7.3 Création du client console 34

7.4 Modifier le client web 37

7.5 Pour tester 44

1. DIAGRAMME DE FLUX



2. INSTALLATION DE NODE ET DE NPM

Ouvrir une session powershell

Cmd d'installation de nodejs

- **winget install OpenJS.NodeJS.LTS**

Cette commande installe la version **LTS (Long Term Support)** de Node.js, recommandée pour ton projet de chat.

L'installation comprend :

- **Node.js** : moteur d'exécution JavaScript.
- **npm** : gestionnaire de bibliothèques Node.js.

Vérifie la version installée

- **Node** -v4

Si tu as une erreur c'est que le chemin d'acc-s ou PATH n'est pas bien défini

Vérifie le chemin avec powershell

- **Test-Path "C:\Program Files\nodejs\node.exe"**

Ajoute ce chemin au PATH

- `$env:Path += ";C:\Program Files\nodejs"`

Retest node -v et npm -v

Par contre cet ajout n'est que temporaire pour le rajouter définitivement à ton PATH voici les commande à lancer dans powershell

```
$nodePath = "C:\Program Files\nodejs"

$userPath = [Environment]::GetEnvironmentVariable(
  "Path", "User"
)

$paths = $userPath -split ";"

if ($nodePath -notin $paths) {
  $newPath = ($paths + $nodePath) -join ";"

  [Environment]::SetEnvironmentVariable(
    "Path",
    $newPath,
    "User"
  )
}
```

3. CREATION DU PROJET

Tu vas maintenant créé ton projet

Toujours dans powershell

Cd c:\

mkdir eureka-chat

cd eureka-chat

npm init -y

npm install express socket.io

Explications

- npm : gestionnaire de paquets de Node.js.
- init : initialise un nouveau projet.
- -y : accepte automatiquement les valeurs par défaut, sans poser de questions.

À quoi sert package.json ?

Il permet notamment de :

- Définir le nom et la version de ton application.
- Déclarer les bibliothèques utilisées (express, socket.io, etc.).
- Définir les commandes de démarrage.
- Gérer les dépendances de ton application.

Ensuite, tu peux ajouter dans package.json :

```
"scripts": {
  "start": "node server.js"
}
```

Tu pourras alors démarrer ton serveur simplement avec :

npm start

4. CREATION DU SERVER

Le fichier server.js complet

Voici notre serveur avec des commentaires directement dans le code.

```
// 1. Importer la bibliothèque Express
const express = require("express");

// 2. Importer le module HTTP de Node.js
const http = require("http");

// 3. Importer Socket.IO pour la communication en temps réel
const { Server } = require("socket.io");

// 4. Créer notre application Express
const app = express();

// 5. Créer un serveur HTTP utilisant Express
const server = http.createServer(app);

// 6. Associer Socket.IO à notre serveur HTTP
const io = new Server(server);

// 7. Rendre accessibles les fichiers HTML, CSS et JavaScript
app.use(express.static("public"));

// 8. Détecter la connexion d'un utilisateur
io.on("connection", (socket) => {

    // 9. Afficher un message dans la console
    console.log("Utilisateur connecté :", socket.id);

    // 10. Écouter les messages envoyés par cet utilisateur
    socket.on("chat:message", (message) => {

        // 11. Vérifier que le message est une chaîne
        if (typeof message !== "string") return;

        // 12. Nettoyer et limiter le texte
        const texte = message.trim().slice(0, 2000);

        // 13. Ignorer les messages vides
```

```
        if (!texte) return;

        // 14. Envoyer le message à tous les utilisateurs
        io.emit("chat:message", {
            utilisateur: socket.id,
            texte: texte,
            date: new Date().toISOString()
        });
    });

    // 15. Détecter la déconnexion d'un utilisateur
    socket.on("disconnect", () => {
        console.log("Utilisateur déconnecté");
    });
});

// 16. Démarrer le serveur sur le port 3000
server.listen(3000, () => {
    console.log("Serveur démarré sur http://localhost:3000");
});
```

Explication détaillée, ligne par ligne

Ligne 1 — Importer Express

const express = require("express");

Décomposons cette instruction :

- const : déclare une variable dont la valeur ne pourra pas être réaffectée.
- express : nom de la variable.
- require() : charge un module Node.js.
- "express" : nom de la bibliothèque à charger.

Express est une bibliothèque qui simplifie la création d'un serveur Web.

Par exemple, elle permet de créer des URL accessibles depuis un navigateur, comme /clients, /articles ou /messages.

Ligne 2 — Importer HTTP

const http = require("http");

Contrairement à Express, http est un module déjà intégré à Node.js.

Il permet de créer un serveur HTTP capable de recevoir des requêtes provenant d'un navigateur.

Ligne 3 — Importer Socket.IO

const { Server } = require("socket.io");

Socket.IO permet de communiquer en temps réel entre le serveur et les navigateurs.

La syntaxe { Server } récupère uniquement la propriété Server exportée par la bibliothèque. C'est ce qu'on appelle la *déstructuration* en JavaScript.

Dans notre chat, Socket.IO permet notamment de recevoir un message et de le diffuser immédiatement aux autres utilisateurs, sans actualiser la page.

Ligne 4 — Créer l'application Express

const app = express();

Cette instruction appelle la fonction express().

Elle crée un objet app qui représente notre application Web.

Nous utiliserons cet objet pour définir des routes HTTP et configurer le serveur.

Ligne 5 — Créer le serveur HTTP

const server = http.createServer(app);

Cette instruction crée le véritable serveur HTTP.

L'objet app est transmis au serveur afin qu'Express puisse traiter les requêtes reçues.

Ligne 6 — Initialiser Socket.IO

const io = new Server(server);

Nous créons un serveur Socket.IO et nous l'associons au serveur HTTP existant.

Le mot-clé new sert à créer une nouvelle instance d'une classe JavaScript.

Ainsi, Express et Socket.IO partagent le même serveur et le même port.

Ligne 7 — Publier les fichiers Web

app.use(express.static("public"));

Cette ligne indique à Express de rendre accessibles les fichiers contenus dans le répertoire public.

Par exemple, avec cette structure :

eureka-chat/

server.js

package.json

public/

index.html

style.css

client.js

Lorsqu'un navigateur demande http://localhost:3000/, Express peut automatiquement lui renvoyer le fichier public/index.html.

Ligne 8 — Détecter les connexions

io.on("connection", (socket) => {

C'est une des instructions les plus importantes.

Elle signifie : lorsqu'un utilisateur se connecte à Socket.IO, exécute le code situé entre les accolades.

Décomposition :

- io : notre serveur Socket.IO.
- .on() : permet d'écouter un événement.
- "connection" : événement déclenché lors d'une nouvelle connexion.
- socket : objet représentant la connexion de cet utilisateur.
- => : syntaxe d'une fonction fléchée JavaScript.

Le code situé à l'intérieur sera exécuté pour chaque nouvelle connexion.

Ligne 9 — Afficher la connexion

console.log("Utilisateur connecté :", socket.id);

console.log() affiche un message dans le terminal PowerShell où tourne Node.js.

socket.id est l'identifiant attribué à cette connexion par Socket.IO.

Exemple :

Utilisateur connecté : AbCd123xyz

Attention : cet identifiant représente une connexion Socket.IO, pas une identité utilisateur permanente.

Ligne 10 — Recevoir les messages

socket.on("chat:message", (message) => {

Cette instruction demande au serveur d'écouter un événement nommé chat:message envoyé par le navigateur.

Le nom chat:message est choisi par nous. Ce n'est pas un mot réservé de JavaScript.

Le paramètre message contient les données envoyées par le client.

Par exemple, si le navigateur exécute :

socket.emit("chat:message", "Bonjour !");

Le serveur recevra :

message = "Bonjour !";

Lignes 11 à 13 — Vérifier le message

if (typeof message !== "string") return;

const texte = message.trim().slice(0, 2000);

if (!texte) return;

Ces lignes effectuent trois vérifications :

1. typeof vérifie que le message est une chaîne de caractères.
2. trim() supprime les espaces au début et à la fin.
3. slice(0, 2000) conserve au maximum 2 000 caractères.

Enfin, if (!texte) return; arrête le traitement si le texte est vide.

Le mot-clé return permet ici de quitter immédiatement la fonction.

Ligne 14 — Diffuser le message

io.emit("chat:message", {

```
    utilisateur: socket.id,
    texte: texte,
    date: new Date().toISOString()
});
```

C'est le cœur de notre application.

io.emit() envoie un événement à **tous les utilisateurs connectés**, y compris celui qui a envoyé le message.

Nous transmettons un objet JavaScript contenant trois informations :

Propriété	Contenu
utilisateur	Identifiant de connexion Socket.IO
texte	Message envoyé
date	Date et heure au format ISO 8601

Par exemple :

```
{
  "utilisateur": "AbCd123xyz",
  "texte": "Bonjour à tous !",
  "date": "2026-11-22T09:15:00.000Z"
}
```

La lettre Z indique que la date est exprimée en UTC.

Ligne 15 — Détecter la déconnexion

```
socket.on("disconnect", () => {
    console.log("Utilisateur déconnecté");
});
```

Cet événement est déclenché lorsque la connexion Socket.IO est interrompue.

Cela peut arriver si l'utilisateur ferme son navigateur, perd sa connexion réseau ou si le serveur s'arrête.

Ligne 16 — Démarrer le serveur

```
server.listen(3000, () => {
    console.log("Serveur démarré sur http://localhost:3000");
});
```

server.listen(3000) demande au serveur d'écouter les connexions sur le port TCP 3000.

La fonction située après => est exécutée lorsque le serveur a commencé à écouter.

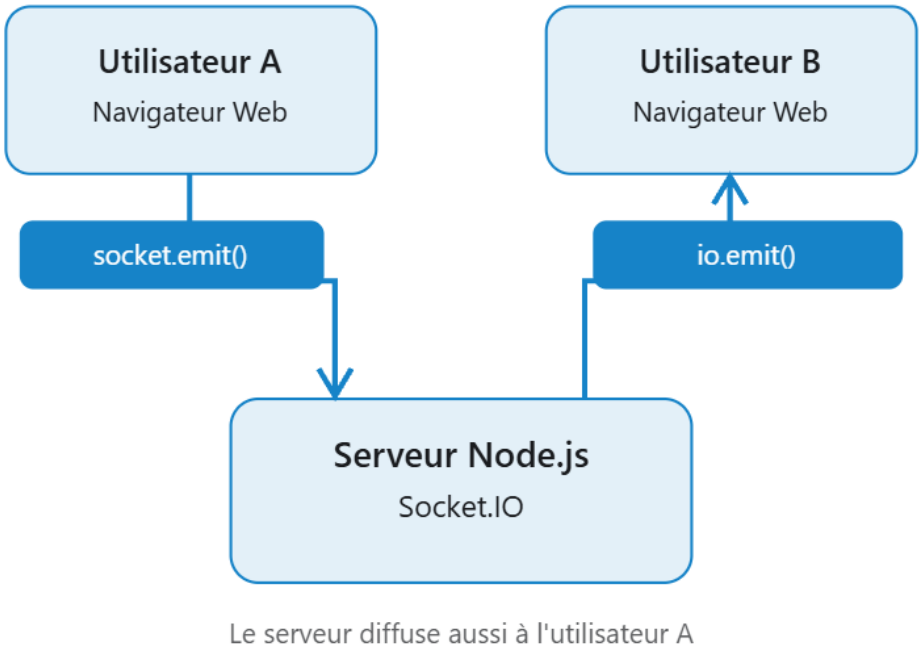
Pour démarrer l'application depuis PowerShell :

node server.js

Tu dois obtenir :

Serveur démarré sur http://localhost:3000

Comment circule un message ?



Le fonctionnement est le suivant :

1. L'utilisateur A écrit « Bonjour » dans son navigateur.
2. Son navigateur envoie le message au serveur avec socket.emit().
3. Le serveur Node.js reçoit l'événement grâce à socket.on().

4. Le serveur diffuse le message avec io.emit().
5. Tous les navigateurs connectés reçoivent le message et peuvent l'afficher.

La différence entre on() et emit()

C'est probablement le concept le plus important à retenir pour utiliser Socket.IO.

Instruction	**Rôle**
socket.on()	Écouter et recevoir un événement
socket.emit()	Envoyer un événement à une connexion
io.emit()	Envoyer un événement à toutes les connexions
socket.broadcast.emit()	Envoyer à toutes les autres connexions, sauf l'émetteur

Par exemple :

```
socket.on("bonjour", (message) => {
    console.log(message);
});
```

Ce code attend de recevoir l'événement bonjour.

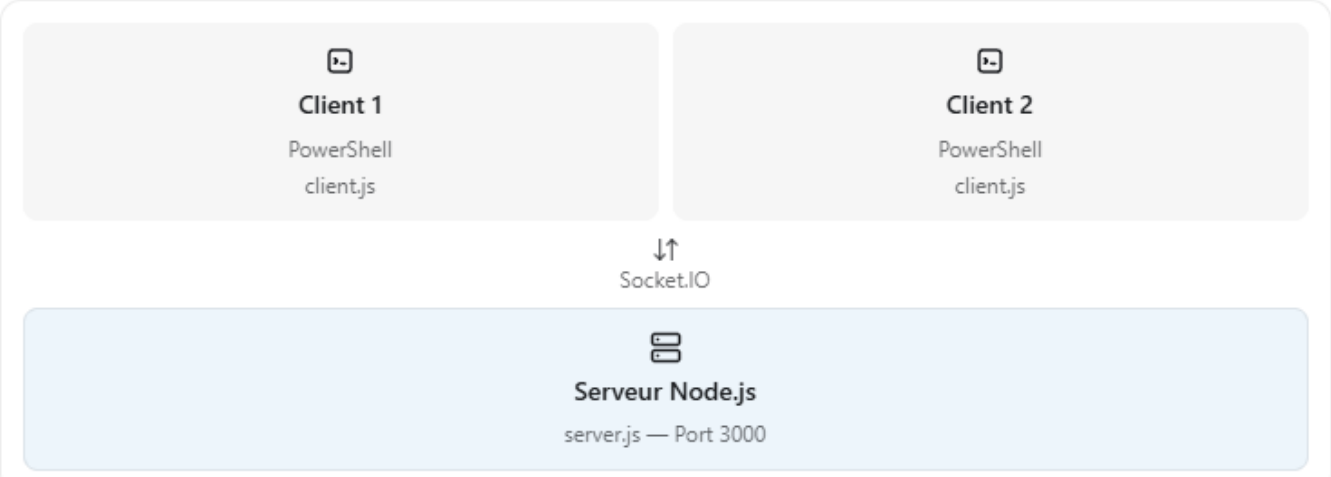
Alors que :

```
socket.emit("bonjour", "Salut !");
```

Ce code envoie l'événement bonjour avec le texte « Salut ! ».

5. CREATION DU CLIENT (LIGNE DE COMMANDE)

5.1 Architecture de notre chat



Client 1

PowerShell

client.js

Client 2

PowerShell

client.js

Socket.IO

Serveur Node.js

server.js — Port 3000

5.2 Installer la bibliothèque cliente

Dans ton répertoire eureka-chat, exécute :

npm install socket.io-client

Cette bibliothèque permet à une application Node.js de se connecter à un serveur Socket.IO.

Nous allons également utiliser readline, une bibliothèque déjà intégrée à Node.js pour lire ce que tu tapes au clavier.

Créer le fichier client.js

Dans le même répertoire que server.js, crée un fichier client.js contenant ce code :

```
// 1. Importer le client Socket.IO
const { io } = require("socket.io-client");

// 2. Importer readline pour lire le clavier
const readline = require("readline");

// 3. Demander le nom de l'utilisateur
const nom = process.argv[2] || "Anonyme";

// 4. Se connecter au serveur
const socket = io("http://localhost:3000");

// 5. Créer une interface de saisie console
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout,
    prompt: "> "
});

// 6. Connexion au serveur réussie
socket.on("connect", () => {
    console.log("Connecté au serveur !");
    console.log("Identifiant :", socket.id);
    console.log("Tape ton message ou /quit pour quitter.");
    rl.prompt();
});

// 7. Réception d'un message
socket.on("chat:message", (message) => {
    console.log(
        "\n[" + message.utilisateur + "] " +
        message.texte
    );

    rl.prompt();
});

// 8. Lecture des messages saisis au clavier
rl.on("line", (ligne) => {

    const texte = ligne.trim();

    // Quitter le programme
```

```
    if (texte === "/quit") {
        socket.disconnect();
        rl.close();
        return;
    }

    // Envoyer un message au serveur
    if (texte && socket.connected) {
        socket.emit("chat:message", nom + " : " + texte);
    } else if (texte) {
        console.log("Serveur non connecté.");
    }

    rl.prompt();
});

// 9. Déconnexion du serveur
socket.on("disconnect", () => {
    console.log("\nDéconnecté du serveur.");
});

// 10. Gestion des erreurs de connexion
socket.on("connect_error", (erreur) => {
    console.log("Erreur de connexion :", erreur.message);
});

// 11. Fermeture du client
rl.on("close", () => {
    socket.disconnect();
    process.exit(0);
});
```

Ce client est compatible avec le server.js que nous avons créé précédemment.

5.3 Lancer les programmes

Fenêtre PowerShell n°1 — Serveur

node server.js

Résultat :

Serveur démarré sur http://localhost:3000

Fenêtre PowerShell n°2 — Client Emmanuel

node client.js Emmanuel

Fenêtre PowerShell n°3 — Client Sophie

node client.js Sophie

Exemple de conversation

Emmanuel

Connecté au serveur !

> Bonjour Sophie

[ID] Emmanuel : Bonjour Sophie

[ID] Sophie : Salut Emmanuel !

> _

Sophie

Connecté au serveur !

[ID] Emmanuel : Bonjour Sophie

> Salut Emmanuel !

[ID] Sophie : Salut Emmanuel !

> _

5.4 Explication des principales instructions

Instruction	Explication
require("socket.io-client")	Charge la bibliothèque cliente Socket.IO
require("readline")	Permet de lire les saisies clavier
process.argv[2]	Récupère le nom passé en paramètre
io("http://localhost:3000")	Se connecte au serveur
socket.on("connect")	Détecte la connexion
socket.on("chat:message")	Reçoit un message du serveur

rl.on("line")	Détecte une ligne saisie puis validée par Entrée
socket.emit("chat:message", texte)	Envoie un message au serveur
socket.disconnect()	Ferme la connexion Socket.IO

Zoom sur process.argv

Lorsque tu exécutes :

node client.js Emmanuel

Node.js reçoit plusieurs arguments :

process.argv[0] // Chemin de node.exe

process.argv[1] // Chemin de client.js

process.argv[2] // "Emmanuel"

C'est pourquoi nous utilisons :

const nom = process.argv[2] || "Anonyme";

L'opérateur || signifie ici : utiliser le nom fourni, ou "Anonyme" si aucun nom n'est renseigné.

Zoom sur readline

```
rl.on("line", (ligne) => {
    console.log("Tu as saisi :", ligne);
});
```

Cette fonction est appelée chaque fois que l'utilisateur appuie sur **Entrée** après avoir saisi du texte.

C'est ce mécanisme qui permet de transformer un simple programme Node.js en application interactive en console.

5.5 Résultat

Coté Emilien

```
PS C:\Eureka\stage 3ème\Emilien> node client.js Emilien
Connecté au serveur !
Identifiant : 9hWQYq9qgCXWhZI1AAAB
Tape ton message ou /quit pour quitter.
> salut
>
[9hWQYq9qgCXWhZI1AAAB] Emilien : salut
>
[bghmGLYb7ijIkJK0AAAA] Manu : comment vas tu ?
>
```

Côté manu

```
PS C:\Eureka\stage 3ème\Emilien> node client.js Manu
Connecté au serveur !
Identifiant : bghmGLYb7ijIkJK0AAAA
Tape ton message ou /quit pour quitter.
>
[9hWQYq9qgCXWhZI1AAAB] Emilien : salut
> comment vas tu ?
>
[bghmGLYb7ijIkJK0AAAA] Manu : comment vas tu ?
>
```

6. CREATION DU CLIENT WEB

Organisation des fichiers



Dans ton projet existant eureka-chat, crée un dossier public avec trois fichiers :

eureka-chat/

server.js

Existant

client.js

Client console

package.json

public/

index.html

style.css

client.js

Dans PowerShell :

mkdir public

New-Item public\index.html -ItemType File

New-Item public\style.css -ItemType File

New-Item public\client.js -ItemType File

Fichier public/index.html

Ce fichier représente la structure de notre interface Web.

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>EUREKA Chat</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

    <div class="chat-container">

        <header>
            <h2>EUREKA Chat</h2>
            <span id="status">Déconnecté</span>
        </header>

        <div class="user-bar">
            <label for="username">Utilisateur :</label>
            <input type="text" id="username"
                   placeholder="Votre nom">
        </div>

        <div id="messages"></div>

        <form id="chat-form">
            <input type="text" id="message"
                   placeholder="Écrire un message..."
                   autocomplete="off" required>

            <button type="submit">Envoyer</button>
        </form>

    </div>
```

```
    <!-- Bibliothèque cliente Socket.IO -->
    <script src="/socket.io/socket.io.js"></script>

    <!-- Notre programme JavaScript -->
    <script src="client.js"></script>

</body>
</html>
```

Quelques éléments importants :

- : zone dans laquelle nous afficherons les messages reçus.
- <form id="chat-form"> : formulaire permettant d'envoyer un message.
- <input id="username"> : nom de l'utilisateur.
- <script src="/socket.io/socket.io.js"> : bibliothèque Socket.IO fournie automatiquement par le serveur.
- <script src="client.js"> : notre programme JavaScript côté navigateur.

Fichier public/style.css

Ce fichier gère uniquement la présentation visuelle.

```
* {
    box-sizing: border-box;
}

body {
    margin: 0;
    font-family: Arial, sans-serif;
    background: #f1f5f9;
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
}

.chat-container {
    width: 600px;
    max-width: 95%;
```

```
    height: 650px;
    background: white;
    border-radius: 14px;
    box-shadow: 0 8px 30px rgba(0,0,0,0.1);
    display: flex;
    flex-direction: column;
    overflow: hidden;
}

header {
    background: #1769aa;
    color: white;
    padding: 18px 22px;
    display: flex;
    justify-content: space-between;
    align-items: center;
}

header h2 {
    margin: 0;
}

#status {
    font-size: 13px;
}

.user-bar {
    padding: 12px 18px;
    background: #f8fafc;
    display: flex;
    align-items: center;
    gap: 12px;
}

.user-bar input {
    padding: 9px;
    border: 1px solid #cbd5e1;
    border-radius: 6px;
    flex: 1;
    min-width: 0;
}

#messages {
    flex: 1;
```

```
    overflow-y: auto;
    padding: 20px;
    display: flex;
    flex-direction: column;
    gap: 12px;
}

.message {
    background: #e2e8f0;
    padding: 10px 14px;
    border-radius: 10px;
    max-width: 80%;
    align-self: flex-start;
    overflow-wrap: anywhere;
}

.message.mine {
    background: #dbeafe;
    align-self: flex-end;
}

.message .author {
    font-size: 12px;
    font-weight: bold;
    margin-bottom: 5px;
    color: #334155;
}

#chat-form {
    display: flex;
    gap: 10px;
    padding: 15px;
    border-top: 1px solid #e2e8f0;
}

#message {
    flex: 1;
    min-width: 0;
    padding: 12px;
    border: 1px solid #cbd5e1;
    border-radius: 8px;
}

button {
```

```
    background: #1769aa;
    color: white;
    border: none;
    border-radius: 8px;
    padding: 0 20px;
    cursor: pointer;
}

button:hover {
    background: #10538a;
}
```

Fichier public/client.js

C'est le fichier le plus intéressant : il va gérer les communications entre le navigateur et ton serveur Node.js.

```
// 1. Connexion au serveur Socket.IO
const socket = io();

// 2. Récupération des éléments HTML
const messages = document.getElementById("messages");
const form = document.getElementById("chat-form");
const input = document.getElementById("message");
const username = document.getElementById("username");
const status = document.getElementById("status");

// 3. Connexion au serveur
socket.on("connect", () => {
    status.textContent = "Connecté";
    status.style.color = "#b9fbc0";
});

// 4. Déconnexion du serveur
socket.on("disconnect", () => {
    status.textContent = "Déconnecté";
    status.style.color = "#ffb3b3";
});

// 5. Réception d'un message
```

```
socket.on("chat:message", (message) => {

    // Créer un bloc HTML pour le message
    const div = document.createElement("div");

    div.classList.add("message");

    // Identifier nos propres messages
    if (message.utilisateur === socket.id) {
        div.classList.add("mine");
    }

    // Créer le nom de l'expéditeur
    const author = document.createElement("div");
    author.className = "author";
    author.textContent = message.utilisateur;

    // Créer le texte du message
    const text = document.createElement("div");
    text.textContent = message.texte;

    // Ajouter les éléments dans le message
    div.appendChild(author);
    div.appendChild(text);

    // Ajouter le message à la conversation
    messages.appendChild(div);

    // Faire défiler vers le dernier message
    messages.scrollTop = messages.scrollHeight;
});

// 6. Envoi d'un message
form.addEventListener("submit", (event) => {

    // Empêcher le rechargement de la page
    event.preventDefault();

    const texte = input.value.trim();

    if (!texte || !socket.connected) return;

    // Récupérer le nom de l'utilisateur
    const nom = username.value.trim() || "Anonyme";
```

```
    // Envoyer au serveur
    socket.emit("chat:message", nom + " : " + texte);

    // Vider la zone de saisie
    input.value = "";
    input.focus();
});
```

Important : nous conservons volontairement le même format que ton client console. Le serveur reçoit une simple chaîne de caractères, et non un objet contenant séparément le nom et le texte.

Le serveur transmet ensuite le message à tous les clients connectés.

Démarrer et tester le chat

Démarre ton serveur depuis PowerShell :

node server.js ou npm start

Ouvre ensuite ton navigateur à cette adresse :

http://localhost:3000

Pour tester la communication entre plusieurs clients, tu peux ouvrir deux onglets du navigateur, puis lancer également ton client console :

node client.js Emmanuel

Tu pourras ainsi échanger des messages entre les trois clients.

Les points importants à comprendre

Élément	Côté serveur	Côté navigateur
Programme	server.js	public/client.js
Environnement	Node.js	Navigateur Web
Réception	socket.on()	socket.on()
Envoi	io.emit()	socket.emit()
Affichage	console.log()	document.createElement()

Lecture utilisateur Événements réseau Formulaire HTML

La principale différence est que **le serveur Node.js utilise** require() **pour charger ses modules**, tandis que notre navigateur utilise les bibliothèques chargées par les balises <script>.

Autre différence : le navigateur dispose de l'objet document, qui permet de manipuler les éléments HTML. Cet objet n'existe pas dans un programme Node.js classique.

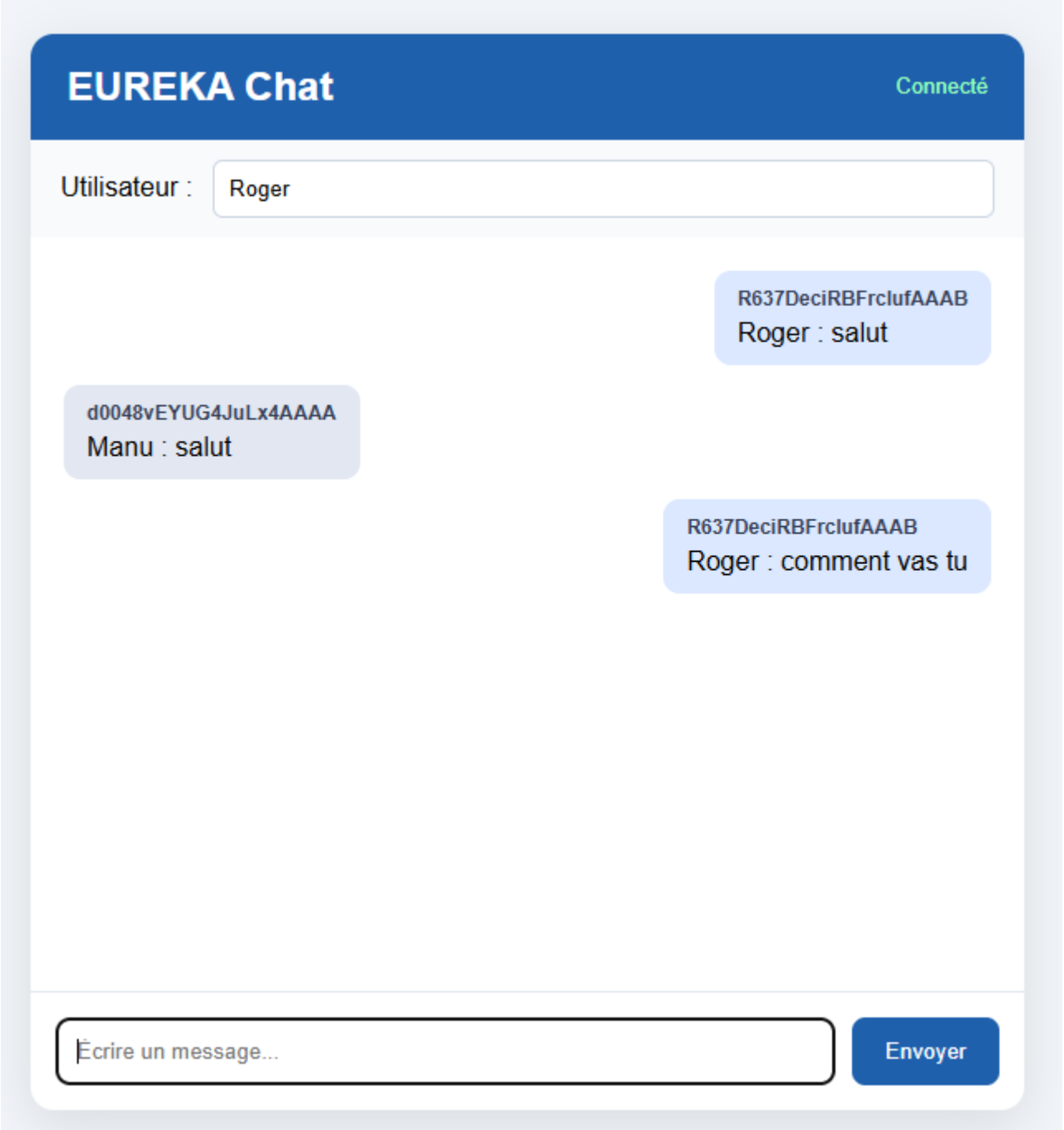
Ce que nous avons maintenant

Nous disposons d'une première application de chat complète :

- Un serveur Node.js.
- Un client console Node.js.
- Un client Web HTML/CSS/JavaScript.
- Une communication bidirectionnelle en temps réel.

Pour l'instant, il s'agit d'un chat public sans authentification ni conservation des messages.

6.1 Résultat


EUREKA Chat
Connecté
Utilisateur :
Roger
R637DeciRBFrclufAAAB
Roger : salut
d0048vEYUG4JuLx4AAAA
Manu : salut
R637DeciRBFrclufAAAB
Roger : comment vas tu
Écrire un message...
Envoyer

7. POUVOIR ENVOYER DES MESSAGES PRIVES

Nous allons faire évoluer **EUREKA Chat** pour que le serveur Node.js gère les utilisateurs connectés et permette d'envoyer des **messages privés à un utilisateur précis**.

Nous allons conserver Socket.IO et modifier trois fichiers : `server.js`, `client.js` (console) et `public/client.js` (Web).

7.1 Architecture



Le serveur conservera en mémoire un registre des utilisateurs connectés, avec leur nom et leur identifiant Socket.IO.

Nous allons utiliser une structure JavaScript appelée Map.

7.2 Création server2.js

Créer le fichier server2.js :

```
const express = require("express");
const http = require("http");
const { Server } = require("socket.io");

const app = express();
const server = http.createServer(app);
const io = new Server(server);
```

```
app.use(express.static("public"));

// Liste des utilisateurs connectés
// Clé : socket.id
// Valeur : nom de l'utilisateur
const utilisateurs = new Map();

// Envoyer la liste à tous les clients
function diffuserUtilisateurs() {
    const liste = Array.from(utilisateurs, ([id, nom]) => ({
        id,
        nom
    }));

    io.emit("users:list", liste);
}

// Connexion d'un client
io.on("connection", (socket) => {

    console.log("Connexion :", socket.id);

    // Identification de l'utilisateur
    socket.on("user:login", (nom) => {

        if (typeof nom !== "string") return;

        nom = nom.trim().slice(0, 50);
        if (!nom) return;

        utilisateurs.set(socket.id, nom);

        console.log(nom + " est connecté");

        diffuserUtilisateurs();
    });

    // Message public
    socket.on("chat:message", (texte) => {

        const nom = utilisateurs.get(socket.id);
        if (!nom || typeof texte !== "string") return;
```

```
        texte = texte.trim().slice(0, 2000);
        if (!texte) return;

        io.emit("chat:message", {
            utilisateur: socket.id,
            nom,
            texte,
            date: new Date().toISOString()
        });
    });

    // Message privé
    socket.on("chat:private", (data) => {

        const nom = utilisateurs.get(socket.id);
        if (!nom || !data || typeof data !== "object") return;

        const { destinataire, texte } = data;

        if (typeof destinataire !== "string" ||
            typeof texte !== "string") return;

        const message = texte.trim().slice(0, 2000);

        if (!message || !utilisateurs.has(destinataire)) {
            socket.emit("chat:error", "Destinataire indisponible");
            return;
        }

        const payload = {
            utilisateur: socket.id,
            nom,
            destinataire,
            texte: message,
            date: new Date().toISOString()
        };

        // Envoyer uniquement au destinataire
        io.to(destinataire).emit("chat:private", payload);

        // Envoyer une copie à l'expéditeur
        if (destinataire !== socket.id) {
            socket.emit("chat:private", payload);
        }
```

```
    });

    // Déconnexion
    socket.on("disconnect", () => {

        const nom = utilisateurs.get(socket.id);

        utilisateurs.delete(socket.id);

        console.log("Déconnexion :", nom || socket.id);

        diffuserUtilisateurs();
    });
});

server.listen(3000, () => {
    console.log("EUREKA Chat démarré sur le port 3000");
});
```

Les nouvelles instructions importantes

La Map des utilisateurs :

const utilisateurs = new Map();

Une Map permet d'associer une clé à une valeur.

Par exemple :

utilisateurs.set("socket123", "Emmanuel");

utilisateurs.set("socket456", "Sophie");

Pour récupérer un utilisateur :

const nom = utilisateurs.get("socket123");

Pour vérifier qu'il existe :

utilisateurs.has("socket123");

Pour le supprimer :

utilisateurs.delete("socket123");

L'envoi privé :

io.to(destinataire).emit("chat:private", payload);

Contrairement à io.emit(), qui diffuse à tous les clients, io.to() cible ici une connexion Socket.IO particulière.

7.3 Création du client console

Création de client2.js

```
const { io } = require("socket.io-client");
const readline = require("readline");

const nom = process.argv[2] || "Anonyme";
const socket = io("http://localhost:3000");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout,
    prompt: "> "
});

let utilisateurs = [];

// Connexion
socket.on("connect", () => {
    socket.emit("user:login", nom);

    console.log("Connecté en tant que " + nom);
    console.log("Commandes :");
    console.log("  /users             Liste des utilisateurs");
    console.log("  /msg NOM MESSAGE   Message privé");
    console.log("  /quit              Quitter");
    rl.prompt();
});

// Liste des utilisateurs
socket.on("users:list", (liste) => {
    utilisateurs = liste;

    console.log("\nUtilisateurs connectés :");
    liste.forEach(u => {
```

```
        console.log(" - " + u.nom + " (" + u.id + ")");
    });

    rl.prompt();
});

// Message public
socket.on("chat:message", (message) => {
    console.log("\n" + message.nom + " : " + message.texte);
    rl.prompt();
});

// Message privé
socket.on("chat:private", (message) => {
    const envoye = message.utilisateur === socket.id;

    console.log(
        "\n[PRIVÉ] " +
        (envoye ? "Vous" : message.nom) +
        " : " + message.texte
    );

    rl.prompt();
});

socket.on("chat:error", (erreur) => {
    console.log("\nErreur : " + erreur);
    rl.prompt();
});

// Saisie clavier
rl.on("line", (ligne) => {

    const texte = ligne.trim();

    if (texte === "/quit") {
        rl.close();
        return;
    }

    if (texte === "/users") {
        console.log(utilisateurs);
        rl.prompt();
        return;
```

```
    }

    if (texte.startsWith("/msg ")) {

        const morceaux = texte.split(" ");
        const destinataireNom = morceaux[1];
        const message = morceaux.slice(2).join(" ");

        const destinataire = utilisateurs.find(
            u => u.nom === destinataireNom
        );

        if (!destinataire) {
            console.log("Utilisateur introuvable");
        } else if (!message.trim()) {
            console.log("Message vide");
        } else {
            socket.emit("chat:private", {
                destinataire: destinataire.id,
                texte: message
            });
        }

    } else if (texte && socket.connected) {
        socket.emit("chat:message", texte);
    }

    rl.prompt();
});

socket.on("disconnect", () => {
    console.log("\nDéconnecté du serveur");
});

rl.on("close", () => {
    socket.disconnect();
    process.exit(0);
});
```

Tu peux maintenant lancer :

Node server2.s

Puis dans une première fenêtre

node client2.js Manu

Dans une deuxième fenêtre :

node client2.js Robert

Dans une troisième fenêtre :

node client2.js Damien

Puis, depuis manu :

/msg Robert Bonjour robert, Damine il est lourd !

Seuls Manu et Robert recevront ce message.

7.4 Modifier le client web

Création de public/index2.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>EUREKA Chat</title>
    <link rel="stylesheet" href="style2.css">
</head>
<body>
<div class="chat-container">

    <header>
        <h2>EUREKA Chat</h2>
        <span id="status">Déconnecté</span>
    </header>

    <div class="user-bar">
        <input id="username" placeholder="Votre nom">
        <button id="login" type="button">Connexion</button>
    </div>

    <div class="user-bar">
        <label for="recipient">Destinataire :</label>
        <select id="recipient">
            <option value="">Tous les utilisateurs</option>
```

```
        </select>
    </div>

    <div id="messages"></div>

    <form id="chat-form">
        <input id="message"
               placeholder="Votre message..."
               autocomplete="off"
               required>
        <button type="submit">Envoyer</button>
    </form>

</div>

<script src="/socket.io/socket.io.js"></script>
<script src="client2.js"></script>
</body>
</html>
```

Création de public/client2.js

```
const socket = io();

const messages = document.getElementById("messages");
const form = document.getElementById("chat-form");
const input = document.getElementById("message");
const username = document.getElementById("username");
const login = document.getElementById("login");
const recipient = document.getElementById("recipient");
const status = document.getElementById("status");

let nomConnecte = "";

// Connexion au serveur
socket.on("connect", () => {
    status.textContent = "Connecté";
    status.style.color = "#b9fbc0";

    // Réidentifier l'utilisateur après reconnexion
    if (nomConnecte) {
        socket.emit("user:login", nomConnecte);
```

```
    }
});

socket.on("disconnect", () => {
    status.textContent = "Déconnecté";
    status.style.color = "#ffb3b3";
});

// Identification
login.addEventListener("click", () => {
    const nom = username.value.trim();

    if (!nom) {
        alert("Saisissez votre nom");
        return;
    }

    nomConnecte = nom;
    username.value = nom;
    socket.emit("user:login", nom);
});

// Liste des utilisateurs
socket.on("users:list", (liste) => {

    const selection = recipient.value;

    recipient.replaceChildren();

    const tous = new Option("Tous les utilisateurs", "");
    recipient.add(tous);

    liste.forEach(u => {
        if (u.id !== socket.id) {
            recipient.add(new Option(u.nom, u.id));
        }
    });

    if ([...recipient.options].some(o => o.value === selection)) {
        recipient.value = selection;
    }
});

// Affichage des messages
```

```
function afficherMessage(message, prive) {

    const div = document.createElement("div");
    div.className = "message";

    if (message.utilisateur === socket.id) {
        div.classList.add("mine");
    }

    const author = document.createElement("div");
    author.className = "author";
    author.textContent = (prive ? "[PRIVÉ] " : "") + message.nom;

    const text = document.createElement("div");
    text.textContent = message.texte;

    div.appendChild(author);
    div.appendChild(text);
    messages.appendChild(div);

    messages.scrollTop = messages.scrollHeight;
}

// Réception publique
socket.on("chat:message", message => {
    afficherMessage(message, false);
});

// Réception privée
socket.on("chat:private", message => {
    afficherMessage(message, true);
});

socket.on("chat:error", erreur => {
    alert(erreur);
});

// Envoi
form.addEventListener("submit", event => {

    event.preventDefault();

    if (!nomConnecte) {
        alert("Identifiez-vous d'abord");
```

```
        return;
    }

    if (!socket.connected) {
        alert("Serveur déconnecté");
        return;
    }

    const texte = input.value.trim();
    if (!texte) return;

    if (recipient.value) {
        socket.emit("chat:private", {
            destinataire: recipient.value,
            texte
        });
    } else {
        socket.emit("chat:message", texte);
    }

    input.value = "";
    input.focus();
});
```

Creation de public/style2.css

```
* {
    box-sizing: border-box;
}

body {
    margin: 0;
    font-family: Arial, sans-serif;
    background: #f1f5f9;
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
}

.chat-container {
    width: 600px;
```

```
    max-width: 95%;
    height: 650px;
    background: white;
    border-radius: 14px;
    box-shadow: 0 8px 30px rgba(0,0,0,0.1);
    display: flex;
    flex-direction: column;
    overflow: hidden;
}

header {
    background: #1769aa;
    color: white;
    padding: 18px 22px;
    display: flex;
    justify-content: space-between;
    align-items: center;
}

header h2 {
    margin: 0;
}

#status {
    font-size: 13px;
}

.user-bar {
    padding: 12px 18px;
    background: #f8fafc;
    display: flex;
    align-items: center;
    gap: 12px;
}

.user-bar input {
    padding: 9px;
    border: 1px solid #cbd5e1;
    border-radius: 6px;
    flex: 1;
    min-width: 0;
}

#messages {
```

```
    flex: 1;
    overflow-y: auto;
    padding: 20px;
    display: flex;
    flex-direction: column;
    gap: 12px;
}

.message {
    background: #e2e8f0;
    padding: 10px 14px;
    border-radius: 10px;
    max-width: 80%;
    align-self: flex-start;
    overflow-wrap: anywhere;
}

.message.mine {
    background: #dbeafe;
    align-self: flex-end;
}

.message .author {
    font-size: 12px;
    font-weight: bold;
    margin-bottom: 5px;
    color: #334155;
}

#chat-form {
    display: flex;
    gap: 10px;
    padding: 15px;
    border-top: 1px solid #e2e8f0;
}

#message {
    flex: 1;
    min-width: 0;
    padding: 12px;
    border: 1px solid #cbd5e1;
    border-radius: 8px;
}
```

```css
button {
    background: #1769aa;
    color: white;
    border: none;
    border-radius: 8px;
    padding: 0 20px;
    cursor: pointer;
}

button:hover {
    background: #10538a;
}
.user-bar select {
    flex: 1;
    min-width: 0;
    padding: 9px;
    border: 1px solid #cbd5e1;
    border-radius: 6px;
    background: white;
}

.user-bar button {
    min-height: 36px;
}
```

7.5 Pour tester

node server2.js

navigateur 1

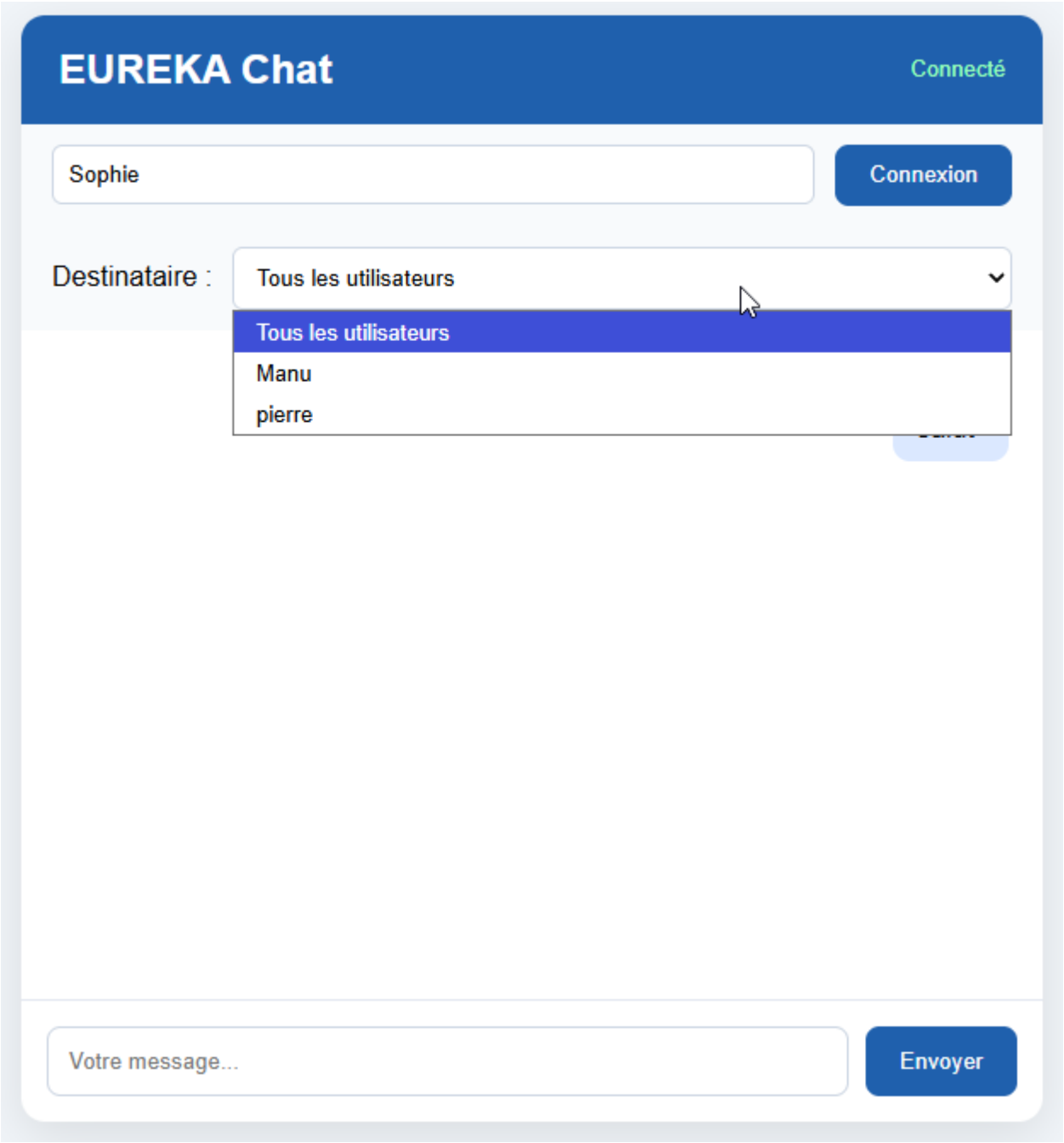
http://localhost:3000/index2.html

navigateur 2

http://localhost:3000/index2.html

navigateur 3

http://localhost:3000/index2.html


EUREKA Chat
Connecté
Sophie
Connexion
Destinataire :
Tous les utilisateurs
Tous les utilisateurs
Manu
pierre
Votre message...
Envoyer