

Rapport SAE5.ROM.03

PLUVIOSE Louis - KARAPETYAN Mikhail

Application de communication WebRTC



Sujet proposé par : Monsieur Philippe Hensel

Université de Haute-Alsace
Institut Universitaire de Technologie de Colmar
Département Réseaux et Télécommunications

10 décembre 2023

Table des matières

1	Introduction	5
2	Diagrammes de séquence	6
2.1	Connexion à la salle de réunion	6
2.2	Connection WebRTC	7
2.3	Échanges de candidats ICE	8
2.4	fonctionnalités supplémentaires	9
3	Utilisation de l'application	10
3.1	Les Réunions	10
3.2	La Messagerie en Temps Réel	11
4	Fonctionnement de l'application de réunion	12
4.1	DOM Elements	12
4.2	Variables	13
4.3	Button Listeners	14
4.4	Socket Event Callbacks	15
4.5	Fonctions Principales	16
5	Fonctionnement de l'application de messagerie	17
5.1	Initialisation et Connexion Socket.IO	17
5.2	Éléments de l'Interface Utilisateur	17
5.3	Envoi de Messages	17
5.4	Réception et Affichage de Messages	17
5.5	Résumé	18
6	Fonctionnement du serveur de l'application	19
6.1	Initialisation d'Express et Configuration SSR	19
6.2	Configuration du Serveur HTTP et Socket.IO	19
6.3	Gestion des Messages et Événements Socket.IO	19
6.4	Gestion de la Déconnexion	19
6.5	Démarrage du Serveur	19
6.6	Résumé	20
7	UI/UX de l'application	21
7.1	Technologies utilisées	21
7.2	Contexte de l'Application	22
8	Analyse UI/UX	24
8.1	Disposition principal de l'application	24
8.2	Navigation	25
8.3	Page - Réunion	27
8.4	Page - Messagerie	28
8.5	Pied de page - Footer	29
9	Performances de l'application	30
9.1	Page Load Time	30
9.2	Core Web Vitals	33
10	Installation de l'application	36
10.1	Installation en dur	36
11	Bugs	37
11.1	Bugs connus	37

	11.2	Bugs inconnus	37
12		Annexes	38
	12.1	Code du frontend	38
	12.2	Code du backend	46
13		Conclusion	58

LISTINGS

1	Page - Réunions	38
2	Page - Messagerie	41
3	Component - NavBar	42
4	Component - MainHead	43
5	Component - Layout	44
6	Component - Footer	45
7	DOM Elements	46
8	Variables	46
9	Button Listeners	46
10	Socket Event Callbacks	47
11	Fonctions Principales	47
12	Code de la messagerie	55
13	Code du serveur	56

Table des figures

0.1	Page Réunion de l'application	21
0.2	Page Messagerie de l'application	21
0.3	Barre de navigation sur un écran large	25
0.4	Bouton de menu pour les écrans de taille réduite	25
0.5	Menu ouvert sur les écrans de taille réduite	26
0.6	La page de réunion	27
0.7	La page de Messagerie	28
0.8	Le pied de page	29

1 Introduction

Dans un monde de plus en plus connecté, la communication en temps réel via Internet est devenue une nécessité cruciale, tant pour les interactions personnelles que professionnelles. Notre projet pour la SAE5.ROM.03, s'inscrit dans cette dynamique en offrant une solution de communication basée sur la technologie WebRTC (Web Real-Time Communication). Nous avons voulu via ce projet nous lancer un défi : celui de permettre des échanges audio et vidéo en temps réel directement depuis le navigateur web, sans nécessité de télécharger des logiciels tiers ou de créer des comptes d'utilisateur.

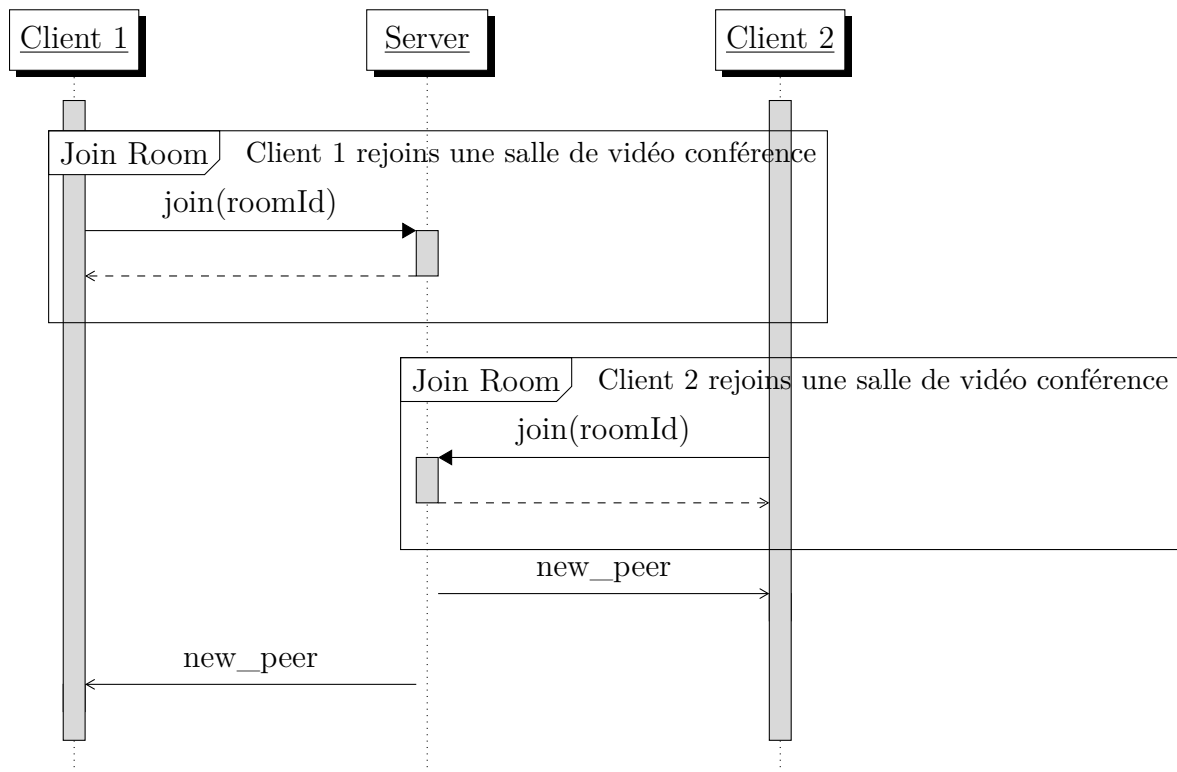
La technologie WebRTC, un standard ouvert et gratuit, permet de réaliser des appels vidéo et audio de haute qualité avec une faible latence, garantissant ainsi une communication fluide et efficace. Notre application SAE5.ROM.03 est conçue pour être intuitive et facilement accessible, offrant une interface utilisateur élégante et des fonctionnalités adaptées à divers contextes, que ce soit pour des réunions, des sessions de travail collaboratif, ou des conversations personnelles.

L'accent est mis sur la facilité d'utilisation. Les utilisateurs peuvent créer et rejoindre des salles de conférence virtuelles en quelques clics, tout en bénéficiant d'une connexion fiable. De plus, l'application prend en charge plusieurs participants.

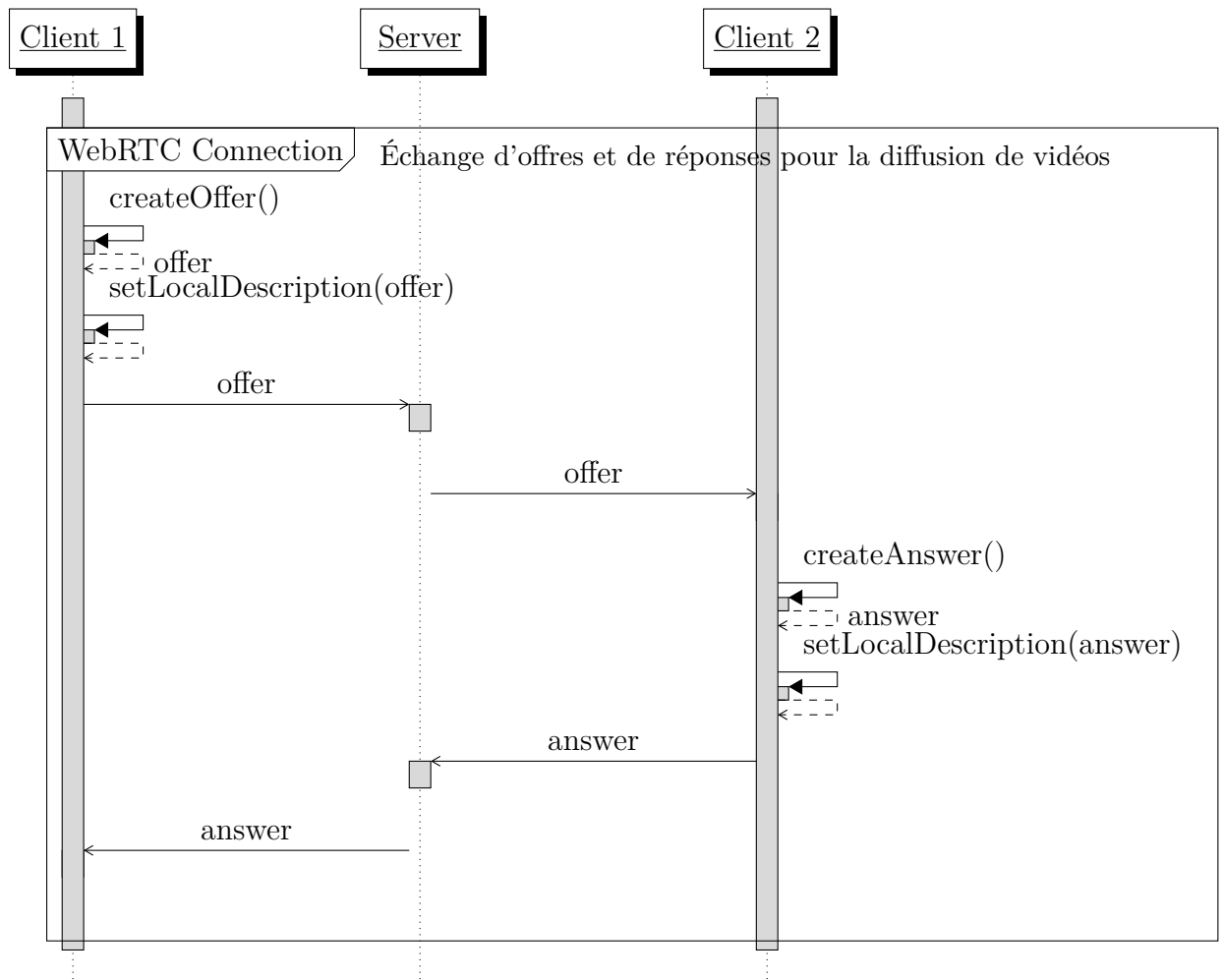
Une démonstration de l'application est disponible à l'adresse suivante : [`https://webrtc.louispluviose.fr/`](https://webrtc.louispluviose.fr/)

2 Diagrammes de séquence

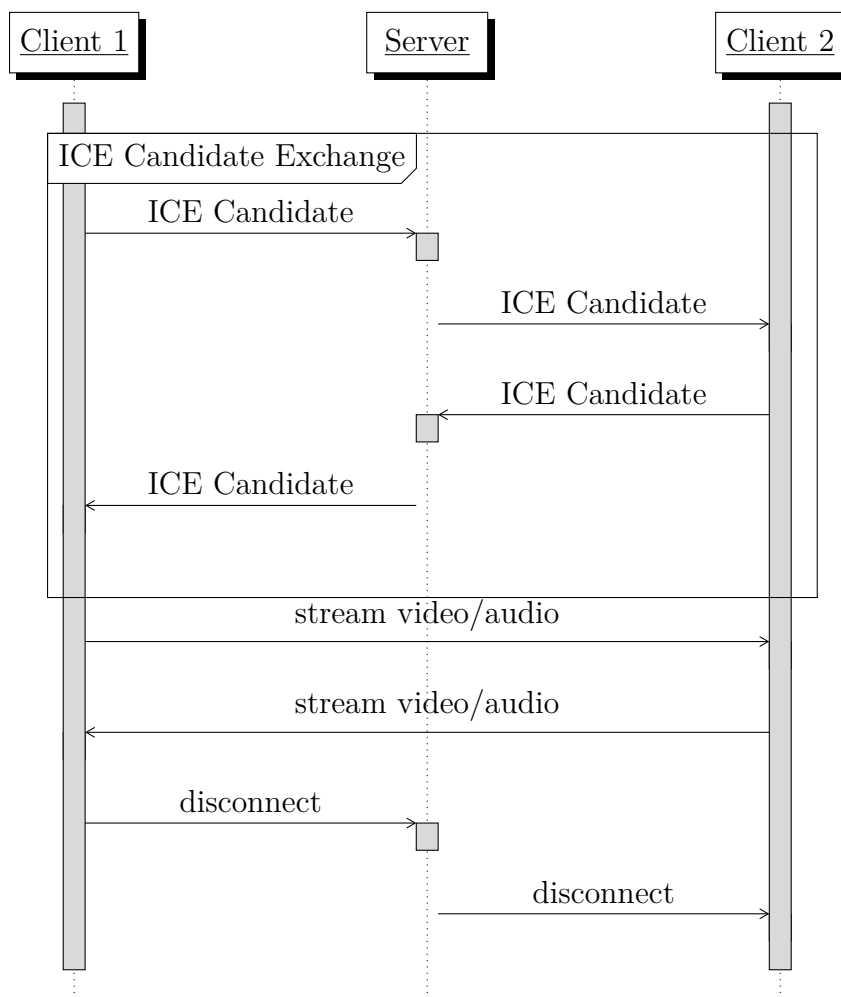
2.1 Connexion à la salle de réunion



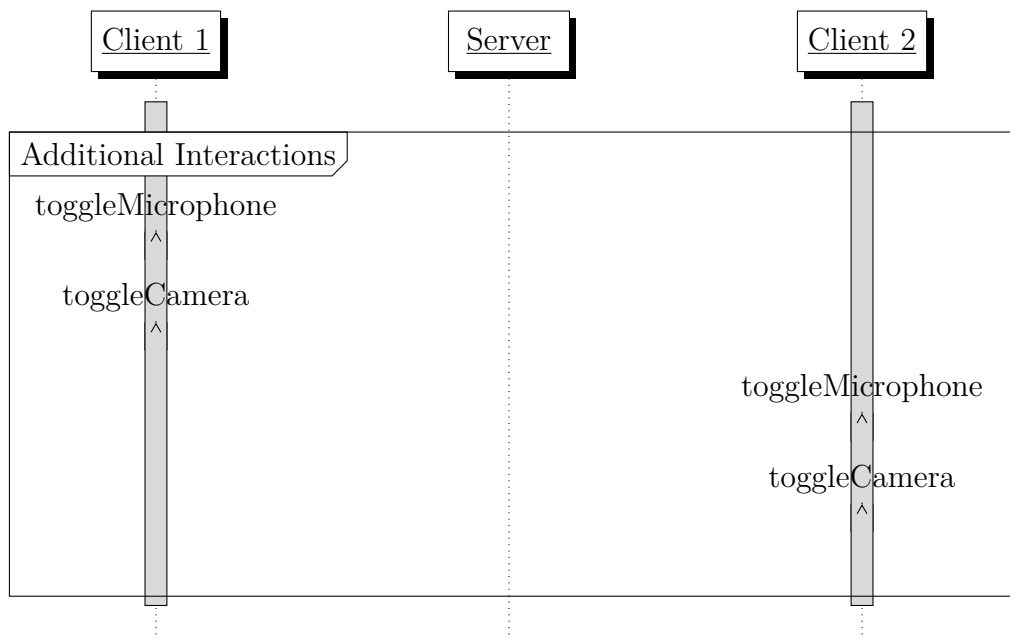
2.2 Connection WebRTC



2.3 Échanges de candidats ICE



2.4 fonctionnalités supplémentaires



3 Utilisation de l'application

3.1 Les Réunions

La page Réunion est spécialement conçue pour orchestrer des réunions virtuelles en temps réel en utilisant une application WebRTC. Découvrez comment les utilisateurs peuvent tirer parti de cette fonctionnalité immersive :

Entrer dans une salle de réunion :

- Saisir le numéro de la salle dans le champ prévu. (Le numéro de la salle peut être choisi librement sans aucune restriction.)
- Cliquer sur "Rejoindre la salle de réunion" pour établir la connexion.
- Lors de la première connexion, l'utilisateur est invité à autoriser l'accès à la caméra et au microphone en fonction du navigateur utilisé.
- Cliquer sur "Autoriser" pour permettre l'accès à la caméra et au microphone.
- L'utilisateur est désormais connecté à la salle de réunion, prêt à participer de manière interactive.

Vidéoconférence :

- L'interface vidéo offre une vue complète avec la vidéo locale de l'utilisateur et celles des autres participants.
- Après s'être connecté à la salle de réunion, l'utilisateur peut visualiser sa propre vidéo ainsi que celles des autres participants, créant une expérience immersive.
- La plateforme prend en charge jusqu'à trois vidéos distantes simultanées, favorisant une collaboration fluide.

Gestion des actions :

- Bouton "Abandonner l'appel" pour quitter la réunion avec une sortie aisée.
- Bouton "Activer/Désactiver le microphone" pour contrôler la diffusion audio.
- Bouton "Activer/Désactiver la caméra" pour gérer la visibilité vidéo selon les besoins de l'utilisateur.

3.2 La Messagerie en Temps Réel

La page Messagerie est dédiée à la communication textuelle en temps réel, offrant une plateforme de discussion instantanée. Découvrez comment les utilisateurs peuvent exploiter cette fonctionnalité :

Saisie d'informations | Envoi de messages :

- Entrer le nom d'utilisateur dans le champ prévu. (Ce nom sera affiché dans les messages, permettant une identification claire pour les autres participants.)
- Saisir le message dans le champ dédié, permettant une expression libre et instantanée.
- Cliquer sur le bouton "Envoyer" pour partager le message avec l'ensemble des participants, favorisant l'interaction dynamique.

Affichage des messages :

- Les messages s'affichent clairement avec le nom de l'expéditeur et le contenu du message.
- Un style distinct est appliqué aux messages de l'utilisateur actuel, facilitant une identification rapide.
- Les messages sont présentés dans l'ordre chronologique, offrant une vue contextuelle de la conversation.

4 Fonctionnement de l'application de réunion

Le code relatif à l'application est disponible sur GitHub à l'adresse suivante : et en annexe de ce document.

4.1 DOM Elements

- `const roomSelectionContainer = document.getElementById('room-selection-container')`
Cette ligne crée une constante `roomSelectionContainer` et lui attribue l'élément DOM (Document Object Model) dont l'identifiant (ID) est `'room-selection-container'`. L'élément récupéré est un conteneur dans le code HTML où les utilisateurs peuvent sélectionner ou entrer des informations pour rejoindre une salle de chat vidéo.
- `const roomInput = document.getElementById('room-input');` : cette ligne crée une constante `roomInput` qui référence l'élément DOM avec l'ID `'room-input'`. Cet élément est un champ de saisie où les utilisateurs peuvent entrer le nom ou le numéro de la salle qu'ils souhaitent rejoindre.
- `const connectButton = document.getElementById('connect-button');` : Cette ligne crée une constante `connectButton` qui référence un élément DOM, probablement un bouton, avec l'ID `'connect-button'`. Ce bouton est utilisé pour initier la connexion à la salle de chat vidéo après que l'utilisateur a saisi les informations nécessaires.
- `const videoChatContainer = document.getElementById('video-chat-container');` : `videoChatContainer` est défini comme référençant l'élément DOM avec l'ID `'video-chat-container'`. Cet élément est le conteneur principal où la vidéo du chat et d'autres éléments d'interface liés au chat vidéo seront affichés une fois la connexion établie.
- `const localVideoComponent = document.getElementById('local-video');` : cette ligne définit `localVideoComponent` comme une référence à l'élément DOM avec l'ID `'local-video'`. Cet élément est utilisé pour afficher la vidéo locale de l'utilisateur, c'est-à-dire, la vidéo capturée par la caméra de l'appareil de l'utilisateur.

En résumé, ce code initialise des constantes pour référencer divers éléments d'une interface de chat vidéo, probablement définis dans le HTML. Ces références sont ensuite utilisées dans d'autres parties du code JavaScript pour gérer la logique de connexion à une salle de chat vidéo et pour contrôler l'affichage et la mise à jour de l'interface utilisateur en fonction des actions de l'utilisateur.

4.2 Variables

- `const socket = io();` : Cette ligne crée une variable `socket` en utilisant `io()`, qui est une fonction de la bibliothèque `Socket.IO`. `Socket.IO` est une bibliothèque JavaScript pour les applications web en temps réel qui permet une communication bidirectionnelle en temps réel entre les navigateurs web et les serveurs. `socket` serait utilisé pour gérer les communications en temps réel, comme envoyer et recevoir des signaux pour établir des connexions vidéo.
- `const mediaConstraints = { audio: true, video: { width: 1280, height: 720 } };` : Cette ligne définit les contraintes pour les médias qui seront utilisés dans la communication. Ici, `audio : true` signifie que l'audio sera activé, et `video : width : 1280, height : 720` définit la résolution de la vidéo à 1280x720 pixels.
- `let localStream;` : Déclaration d'une variable `localStream`. Cette variable sera utilisée plus tard pour stocker le flux de médias local (audio et vidéo) de l'utilisateur.
- `let roomId;` : Déclaration d'une variable `roomId` qui sera utilisée pour stocker l'identifiant de la salle de chat vidéo à laquelle l'utilisateur se connecte.
- `let peerConnections = {};` : Cette ligne initialise un objet JavaScript `peerConnections` comme un dictionnaire vide. Ce dictionnaire sera utilisé pour stocker toutes les connexions peer-to-peer (P2P) établies durant la session de chat vidéo. Chaque connexion P2P représente une connexion directe à un autre utilisateur dans le chat vidéo.
- `const iceServers = { ... };` : Cette partie définit un objet `iceServers` qui contient la configuration des serveurs ICE (Interactive Connectivity Establishment). ICE est utilisé dans les applications WebRTC pour faciliter la connexion entre les pairs à travers différents types de réseaux et de pare-feu/NAT.

À l'intérieur de cet objet, deux types de serveurs sont configurés :

Un serveur STUN (`stun :stun.l.google.com :19302`), qui aide à découvrir l'adresse IP publique de l'utilisateur.

Un serveur TURN (`turn :relay1.expressturn.com :3478` avec un nom d'utilisateur et un mot de passe), qui est utilisé pour relayer le trafic si une connexion P2P directe n'est pas possible.

En résumé, Ce code initialise des variables clés pour la gestion de la communication en temps réel et du streaming vidéo. Il crée une connexion `Socket.IO`, définit les contraintes des médias (audio et vidéo), prépare des variables pour stocker le flux local et l'ID de la salle, et configure un dictionnaire pour gérer les connexions peer-to-peer. Il définit également des serveurs ICE pour aider à la connectivité réseau dans WebRTC.

4.3 Button Listeners

- `connectButton.addEventListener('click', () => { joinRoom(roomInput.value); });` : Cette ligne attache un écouteur d'événement au bouton `connectButton`. Lorsque ce bouton est cliqué, la fonction anonyme déclenche l'appel de la fonction `joinRoom` avec la valeur actuelle de l'élément `roomInput` comme argument. `joinRoom` est une fonction qui gère la logique pour rejoindre une salle de chat vidéo spécifique basée sur l'identifiant de la salle fourni.
- Déclarations des boutons `hangUpButton`, `toggleMicButton` et `toggleCameraButton` : Ces lignes récupèrent des références à trois boutons de l'interface utilisateur : un pour raccrocher l'appel, un pour activer/désactiver le microphone, et un pour activer/désactiver la caméra.
- Ajout de gestionnaires d'événements pour les boutons :
 - `hangUpButton.addEventListener('click', hangUpCall);` : ajoute un gestionnaire d'événement pour gérer l'action de raccrocher l'appel.
 - `toggleMicButton.addEventListener('click', toggleMicrophone);` : ajoute un gestionnaire pour activer ou désactiver le microphone.
 - `toggleCameraButton.addEventListener('click', toggleCamera);` : ajoute un gestionnaire pour activer ou désactiver la caméra.
- `let isRoomCreator = false;` : Cette ligne déclare une variable `isRoomCreator` et l'initialise à `false`. Cette variable est utilisée pour suivre si l'utilisateur actuel est celui qui a créé la salle de chat vidéo. Cela peut affecter la logique de l'application, comme qui a le droit de contrôler certaines fonctionnalités de la salle ou d'inviter d'autres utilisateurs.

En résumé, Dans cet extrait, des écouteurs d'événements sont ajoutés à divers boutons de l'interface utilisateur. Ces boutons permettent de rejoindre une salle de chat vidéo, de raccrocher l'appel, d'activer/désactiver le microphone, et d'activer/désactiver la caméra. Une variable `isRoomCreator` est également définie pour suivre si l'utilisateur est le créateur de la salle.

4.4 Socket Event Callbacks

- `socket.on('room_created', async () => { ... });` : Quand le serveur envoie un message `room_created`, cela signifie que l'utilisateur a créé une nouvelle salle. Le flux vidéo local est configuré et `isRoomCreator` est défini sur `true`.
- `socket.on('room_joined', async () => { ... });` : Ce gestionnaire réagit au message `room_joined`, indiquant que l'utilisateur a rejoint une salle existante. Le flux local est configuré, `isRoomCreator` est défini sur `false`, et un message `start_call` est émis pour commencer l'appel.
- `socket.on('full_room', () => { ... });` : Réagit au message `full_room`, indiquant que la salle que l'utilisateur tente de rejoindre est pleine. Affiche une alerte pour informer l'utilisateur.
- `socket.on('start_call', async () => { ... });` : Lors de la réception d'un message `start_call`, si l'utilisateur est le créateur de la salle, il initie les connexions Peer-to-Peer (P2P) avec les autres participants.
- `socket.on('webrtc_offer', async (data) => { ... });` : Gère les offres WebRTC reçues d'autres pairs. Si le flux local n'est pas encore configuré, il le fait, puis traite l'offre WebRTC.
- `socket.on('webrtc_answer', (data) => { ... });` : Gère les réponses WebRTC reçues en réponse à une offre envoyée précédemment.
- `socket.on('webrtc_ice_candidate', (data) => { ... });` : Gère les messages de candidats ICE, qui sont des informations nécessaires pour établir la connectivité réseau dans WebRTC.
- `socket.on('new_peer', async (peerId) => { ... });` : Réagit à un message indiquant qu'un nouveau pair a rejoint la salle, et initie la configuration d'une connexion P2P avec ce nouveau pair.

En résumé, cet extrait définit plusieurs gestionnaires d'événements pour gérer les communications via Socket.IO dans notre application de chat vidéo WebRTC. Ces événements incluent la création et la jonction à des salles, la gestion de salles pleines, le démarrage d'appels, et la gestion des offres, réponses, et candidats ICE de WebRTC, ainsi que la gestion de l'arrivée de nouveaux pairs.

4.5 Fonctions Principales

1. `joinRoom(room)`
 - But : Afficher l'interface de la conférence vidéo.
 - Fonctionnement : Masque le conteneur de sélection de salle et affiche le conteneur de chat vidéo. Masque également le texte d'attente.
2. `showVideoConference()`
 - But : Afficher l'interface de la conférence vidéo.
 - Fonctionnement : Masque le conteneur de sélection de salle et affiche le conteneur de chat vidéo. Masque également le texte d'attente.
3. `setLocalStream(mediaConstraints)`
 - But : Configurer le flux de médias local.
 - Fonctionnement : Obtient le flux multimédia local selon les contraintes fournies et l'affecte à l'élément vidéo local.
4. `handleNewPeer(peerId)`
 - But : Gérer l'arrivée d'un nouveau pair.
 - Fonctionnement : Crée une connexion peer-to-peer avec le nouveau pair, ajoute le flux local à cette connexion, crée une offre WebRTC et l'envoie au pair via le socket.
5. `setupPeerConnection(peerId, isInitiator)`
 - But : Configurer une connexion peer-to-peer.
 - Fonctionnement : Crée une nouvelle connexion WebRTC avec les serveurs ICE configurés, ajoute des gestionnaires pour les événements track et icecandidate, et si l'utilisateur est l'initiateur, crée et envoie une offre WebRTC.
6. `addRemoteStream(event, peerId)`
 - But : Ajouter un flux distant à l'interface utilisateur.
 - Fonctionnement : Crée un nouvel élément vidéo pour le flux distant ou utilise un élément existant, puis associe le flux distant à cet élément vidéo.
7. `handleIceEvent(event, peerId)`
 - But : Traiter une offre WebRTC reçue.
 - Fonctionnement : Configure la description distante, gère les candidats ICE mis en cache, crée une réponse et l'envoie au pair.
8. `handleAnswer(data)`
 - But : Traiter une réponse WebRTC reçue.
 - Fonctionnement : Met à jour la description distante de la connexion peer-to-peer.
9. `handleIceCandidate(data)`
 - But : Gérer les candidats ICE reçus.
 - Fonctionnement : Ajoute les candidats ICE à la connexion peer-to-peer correspondante.
10. `hangUpCall()`
 - But : Terminer l'appel en cours.
 - Fonctionnement : Ferme toutes les connexions peer-to-peer, arrête le flux local et nettoie l'interface utilisateur.
11. `toggleMicrophone()`
 - But : Activer ou désactiver le microphone.
 - Fonctionnement : Bascule l'état d'activation de la piste audio du flux local.
12. `toggleCamera()`
 - But : Activer ou désactiver la caméra.
 - Fonctionnement : Bascule l'état d'activation de la piste vidéo du flux local.

5 Fonctionnement de l'application de messagerie

5.1 Initialisation et Connexion Socket.IO

- Écouteur d'événement `'DOMContentLoaded'` : Le code s'exécute une fois que le contenu de la page est complètement chargé.
- Création d'une socket : `const socket = io();` : initialise une connexion Socket.IO avec le serveur.
- Génération d'un identifiant utilisateur : `const userId = Date.now().toString();` crée un identifiant unique pour chaque utilisateur en utilisant le timestamp actuel.
- Connexion à Socket.IO : L'écouteur `socket.on('connect', ...)` est déclenché lorsque le client est connecté au serveur Socket.IO.

5.2 Éléments de l'Interface Utilisateur

- Sélection des éléments DOM : Les éléments pour le bouton d'envoi, la zone d'affichage des messages, le champ de saisie du nom d'utilisateur et le champ de saisie du message sont récupérés.

5.3 Envoi de Messages

- Écouteur d'événement pour le bouton d'envoi : Lorsque le bouton d'envoi est cliqué, le nom d'utilisateur et le message sont récupérés.
- Validation et envoi du message : Si le message n'est pas vide, il est envoyé au serveur via `socket.emit('chat_message', userId, username, message)`.
- Réinitialisation du champ de saisie du message : Le champ de saisie du message est vidé après l'envoi du message.

5.4 Réception et Affichage de Messages

- Écouteur d'événement pour `'chat_message'` : Lorsqu'un message est reçu du serveur, il déclenche l'exécution du code suivant :
- Création de conteneurs pour le message : Un div pour contenir le message et un autre div pour le texte du message sont créés et stylisés.
- Identification de l'expéditeur : Un élément strong est utilisé pour afficher le nom de l'expéditeur en gras. Il affiche "Vous" si l'identifiant de l'expéditeur correspond à celui de l'utilisateur actuel, sinon le nom d'utilisateur de l'expéditeur.
- Ajout du message : Le texte du message est ajouté après le nom de l'expéditeur.
- Stylisation basée sur l'expéditeur : Les styles sont appliqués différemment pour distinguer visuellement les messages envoyés par l'utilisateur actuel de ceux des autres utilisateurs.
- Ajout du message à l'interface utilisateur : Le conteneur de message est ajouté à `messagesDiv`.
- Défilement automatique : L'affichage des messages défile automatiquement vers le bas pour montrer les nouveaux messages.

5.5 Résumé

Ce script crée un client de chat en temps réel. Lorsqu'un utilisateur envoie un message, il est transmis au serveur via Socket.IO, puis distribué à tous les clients connectés. Les messages entrants sont affichés dans l'interface utilisateur avec une distinction entre les messages de l'utilisateur actuel et ceux des autres utilisateurs. Ce script démontre une utilisation typique de Socket.IO pour une application de chat interactive.

6 Fonctionnement du serveur de l'application

6.1 Initialisation d'Express et Configuration SSR

- Importation des modules nécessaires : Express, le module HTTP, Socket.IO, et le gestionnaire SSR d'Astro.
- Création d'une application Express : `const app = express();`
- Configuration du chemin de base : `const base = '/'`; définit le chemin de base pour l'application.
- Servir les fichiers statiques : Les fichiers générés par Astro sont servis en tant que fichiers statiques.
- Gestion SSR pour toutes les requêtes GET : `app.get('*', ssrHandler)`; utilise le gestionnaire SSR d'Astro pour toutes les requêtes GET, permettant le rendu des pages côté serveur.

6.2 Configuration du Serveur HTTP et Socket.IO

- Création d'un serveur HTTP : Enveloppe l'application Express avec un serveur HTTP.
- Initialisation de Socket.IO : Crée une nouvelle instance de Socket.IO liée au serveur HTTP.

6.3 Gestion des Messages et Événements Socket.IO

- Stockage des messages : Un tableau `messages` est utilisé pour stocker les messages du chat.
- Gestion des connexions Socket.IO :
 - Lorsqu'un utilisateur se connecte, son ID de socket est enregistré, et les messages précédents sont envoyés à ce client.
 - Écoute les messages de chat entrants, les stocke, et les diffuse à tous les clients connectés.
 - Gère l'adhésion des utilisateurs aux salles de chat. Si une salle a moins de 4 utilisateurs, un nouvel utilisateur peut rejoindre. Sinon, un événement `full_room` est émis.
 - Gère le démarrage d'un appel vidéo en diffusant un événement `start_call` aux pairs dans la même salle.
 - Traite les offres WebRTC, les réponses et les candidats ICE en les transmettant aux pairs concernés.

6.4 Gestion de la Déconnexion

- Déconnexion d'un utilisateur : Lorsqu'un utilisateur se déconnecte, un message est enregistré, et d'autres actions peuvent être effectuées comme la gestion de l'état de la salle.

6.5 Démarrage du Serveur

- Port du serveur : Le serveur écoute sur un port spécifié ou sur le port 8080 par défaut.

— Démarrage du serveur : Le serveur commence à écouter les connexions entrantes.

6.6 Résumé

Ce script serveur gère une application de chat en temps réel avec des fonctionnalités de chat vidéo en utilisant WebRTC et Socket.IO. Il gère également le rendu côté serveur pour les pages front-end construites avec Astro, offrant une expérience utilisateur fluide et intégrée.

7 UI/UX de l'application

L'expérience utilisateur (UX) et la conception de l'interface utilisateur (UI) jouent un rôle essentiel dans le succès d'une application web. Cette partie du rapport se penche attentivement sur ces deux aspects cruciaux, examinant de près l'UI/UX de l'application en question. Notre objectif est de fournir une évaluation approfondie de la convivialité, de l'esthétique et de la fonctionnalité de l'interface, ainsi que de l'expérience globale qu'elle offre à ses utilisateurs.

Au cours de cette analyse, nous examinerons la conception visuelle, la facilité de navigation et la réactivité sur différentes plateformes. À travers cette démarche, nous chercherons à identifier les points forts de l'application ainsi que les domaines qui pourraient bénéficier d'améliorations. Les recommandations formulées dans ce rapport visent à optimiser l'interaction des utilisateurs avec l'application, favorisant ainsi une expérience utilisateur exceptionnelle.

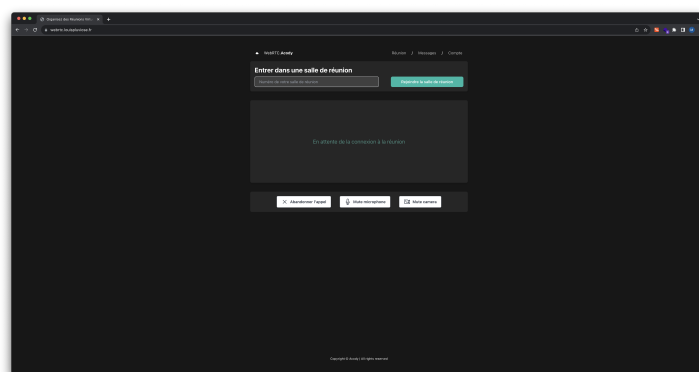


FIG. 0.1: Page Réunion de l'application

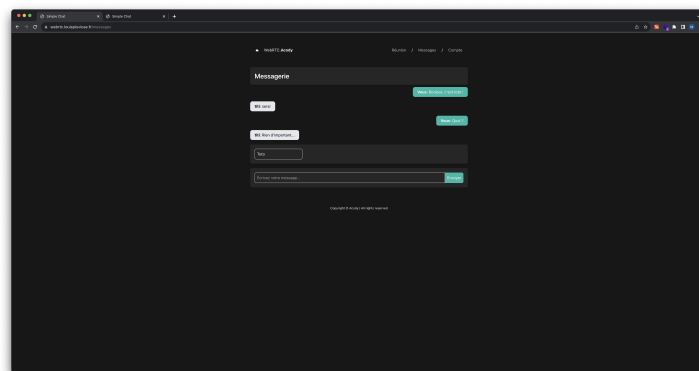


FIG. 0.2: Page Messagerie de l'application

7.1 Technologies utilisées

7.1.1 Astro.js - Framework frontend utilisé

Nous avons utilisé le framework Astro.js pour la réalisation de la partie frontend de notre application web. C'est un framework récent sur le marché qui adopte une approche "server-first", privilégiant le rendu côté serveur par rapport au rendu côté client dans le navigateur. Cela offre des performances très élevées, ce qui nous intéresse dans le cas de notre projet. Également dans ce framework, tout est réuni dans un seul fichier `texttt.astro` :

HTML, CSS et JavaScript. De plus, sa compatibilité avec des frameworks tels que "TailwindCSS" et "React.js" nous a permis d'appliquer un design moderne et convivial pour l'utilisateur.

7.1.2 TailwindCSS et React.js

Tailwind CSS est un framework CSS utilitaire qui simplifie le développement et la stylisation des interfaces utilisateur. Contrairement aux frameworks CSS traditionnels basés sur des composants prédéfinis, Tailwind CSS fournit des classes utilitaires directement applicables dans le code HTML. Ces classes permettent de définir rapidement et de manière cohérente des styles tels que la couleur, la taille, la marge, le rembourrage, etc. L'approche de Tailwind CSS encourage une flexibilité accrue tout en offrant une base solide pour la conception.

React.js est une bibliothèque JavaScript développée par Facebook pour la construction d'interfaces utilisateur interactives. React utilise une approche basée sur les composants, permettant de créer des morceaux d'interface réutilisables et modulaires. Cette approche facilite la gestion de l'état de l'application et la mise à jour dynamique de l'interface en réponse aux changements.

Dans le cadre de notre projet, nous avons utilisé Tailwind CSS pour simplifier la stylisation en exploitant ses classes utilitaires directement dans le code HTML. Cela a accéléré le processus de conception en nous permettant de définir rapidement et de manière cohérente les styles des éléments.

Parallèlement, nous avons intégré React.js pour structurer l'interface utilisateur de manière modulaire. Les composants React ont été utilisés pour diviser l'application en parties réutilisables, facilitant ainsi la maintenance et la gestion de l'état global de l'application.

7.2 Contexte de l'Application

L'application vise à faciliter les réunions virtuelles en temps réel. Elle offre une plateforme pour la communication vidéo instantanée, améliorant ainsi la collaboration à distance. Les fonctionnalités clés comprennent la possibilité d'entrer dans une salle de réunion en utilisant un numéro spécifique, la gestion du son et de la caméra, ainsi que la capacité à abandonner les appels en un clic. En plus des fonctionnalités de réunion vidéo en temps réel, l'application intègre également une fonctionnalité de messagerie pour une communication asynchrone entre les utilisateurs. Cette fonctionnalité permet aux utilisateurs d'échanger des messages textuels, complétant ainsi l'expérience de communication collaborative.

7.2.1 Public Cible

L'application cible un large éventail d'utilisateurs professionnels cherchant à optimiser leurs réunions virtuelles. Elle s'adresse particulièrement aux équipes travaillant à distance, aux entreprises cherchant à améliorer la communication interne et externe, ainsi qu'aux individus ayant besoin d'une solution efficace pour les réunions en ligne.

7.2.2 Objectifs de l'Application (Réunions)

1. **Faciliter les Réunions Virtuelles :** Offrir une plateforme conviviale pour l'organisation de réunions virtuelles.
2. **Communication Vidéo en Temps Réel :** Permettre des appels vidéo en temps réel pour une interaction plus dynamique.

3. **Simplicité d'Utilisation** : Assurer une expérience utilisateur intuitive pour maximiser l'adoption de l'application.
4. **Optimiser la Collaboration à Distance** : Fournir des fonctionnalités simples et efficaces pour améliorer la collaboration à distance.

7.2.3 Objectifs de l'Application (Messagerie)

1. **Communication Asynchrone** : Fournir une plateforme de messagerie pour permettre des échanges asynchrones entre les utilisateurs.
2. **Facilité d'Utilisation** : Assurer une interface conviviale pour la saisie et l'envoi de messages.
3. **Identité de l'Utilisateur** : Permettre aux utilisateurs de spécifier leur nom d'utilisateur pour une identification personnalisée.

7.2.4 Principales Fonctionnalités (Réunions)

1. **Entrée dans une Salle de Réunion** : Les utilisateurs peuvent rejoindre une salle de réunion en saisissant un numéro spécifique.
2. **Gestion du Son et de la Caméra** : Contrôle de la fonction audio et vidéo pour une expérience personnalisée.
3. **Abandonner les Appels en un Clic** : Facilité pour quitter rapidement une réunion.
4. **Affichage Vidéo en Temps Réel** : Possibilité de visualiser les flux vidéo en temps réel des participants.

7.2.5 Principales Fonctionnalités (Messagerie)

1. **Saisie de Nom d'Utilisateur** : Les utilisateurs peuvent entrer leur nom d'utilisateur pour une identification personnalisée.
2. **Saisie de Message** : Interface pour la saisie et l'envoi de messages texte.
3. **Affichage des Messages** : Les messages échangés sont affichés dans une interface dédiée.
4. **Styles Différenciés** : Les messages de l'utilisateur actuel sont stylisés différemment pour une distinction visuelle.

7.2.6 Implémentation Technique (Réunions)

1. **Utilisation de Socket.io** : Les connexions peer-to-peer WebRTC sont établies via les événements de socket, notamment les événements `room_created`, `room_joined`, et `start_call`.
2. **RTCPeerConnection** : Pour créer et gérer les connexions peer-to-peer.
3. **Interface Utilisateur** : La partie vidéo est divisée en un conteneur local et plusieurs conteneurs distants.

7.2.7 Implémentation Technique (Messagerie)

1. **Utilisation de Socket.io** : La messagerie utilise Socket.io pour la gestion des communications en temps réel.
2. **Identifiant Unique de l'Utilisateur** : Chaque utilisateur est associé à un identifiant unique généré au moment de la connexion.
3. **Événements Socket.io** : L'application utilise des événements Socket.io tels que `'connect'` et `'chat_message'` pour gérer la communication.

8 Analyse UI/UX

8.1 Disposition principal de l'application

Un composant principal de l'application est mise en place pour gérer la disposition globale de l'application. Ce composant est utilisé pour encapsuler les composants de navigation et de pied de page, ainsi que le contenu de la page. Il est également utilisé pour définir les paramètres globaux de l'application, tels que la couleur de fond, la taille maximale de la page, la police et etc...

Le code source est présent en annexe (voir Annexe 4). Voici une explication technique à travers différents points sur la structure du composant principal de l'application.

8.1.1 Structure Globale

Le code présente une structure HTML en Astro, avec l'inclusion de composants tels que `MainHead`, `Nav`, et `Footer`. Le corps de la page est défini avec des classes CSS pour la mise en page, la couleur de fond et la taille maximale.

8.1.2 Barre de Navigation (<Nav />)

La barre de navigation est intégrée via le composant `Nav`. Cependant, le code spécifique à la barre de navigation elle-même n'est pas inclus ici. Nous importons le composant `Nav` depuis le fichier `Nav.astro` (voir Annexe 3).

8.1.3 Styles Globaux

Des styles globaux sont définis, notamment pour le corps de la page, la police (San Francisco) et la couleur de fond.

8.1.4 Police San Francisco

Une police personnalisée, San Francisco, est intégrée via `@font-face`. Cela peut contribuer à l'esthétique globale de la page, mais la police étant chargée depuis un lien externe, la vitesse de chargement peut être affectée.

8.1.5 Utilisation de Variables Dynamiques

Des variables telles que `title`, `seoTitle`, et `seoDesc` provenant du composant `MainHead` sont présentes. Cela permet une personnalisation dynamique du titre et de la description SEO.

8.2 Navigation

La disposition et la navigation au sein de l'application s'effectue via une barre de navigation positionnée en haut de la page. Cette barre comprend deux boutons : "Réunions" et "Messages". L'utilisateur final est ainsi invité à faire son choix à travers cette barre de navigation, lui offrant une manière claire et accessible d'accéder aux fonctionnalités désirées.



FIG. 0.3: Barre de navigation sur un écran large

8.2.1 Réalisation Technique

Le code source est présent en annexe (voir Annexe 3 - 15.1.3). Voici une explication technique à travers différents points sur la structure de la "NavBar" (barre de navigation) de l'application.

1. **Structure HTML et classes CSS :** - La structure de la barre de navigation est définie dans une balise `header`. - La classe `lg:flex` rend la barre de navigation flexible sur les écrans de taille large. - La balise `Astronav` encapsule l'ensemble de la barre de navigation.
2. **Logo et titre :** - Un logo représenté par un fichier SVG est inclus, utilisant la classe `icon`. - Le titre de la page, "WebRTC.Acody," est placé à côté du logo et est stylisé avec des classes CSS.
3. **Bouton de menu pour les écrans de taille réduite :** - Pour les écrans de taille réduite (inférieure à lg), un bouton de menu est affiché grâce à la classe `icon-menus`.

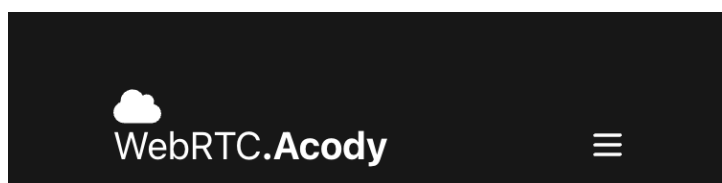


Fig. 0.4: Bouton de menu pour les écrans de taille réduite

4. **MenuItems et liens de navigation :** - La balise `MenuItems` enveloppe la liste de navigation et la classe `hidden` la rend initialement invisible sur les écrans larges. - La liste de navigation est structurée avec des liens vers les sections du site telles que "Réunion," "Messages".
5. **Styles CSS :** - Des styles CSS personnalisés sont définis pour les séparateurs, les liens de la liste, le survol des liens, l'opacité, et les filtres d'image. - Des classes telles que `separator-mob` sont utilisées pour ajuster le style en fonction de la taille de l'écran. - Les filtres d'image, tels que `hue-rotate`, sont appliqués pour des effets visuels lors du survol.
6. **Media Queries :** - Des règles de media queries sont utilisées pour adapter le style en fonction de la largeur de l'écran. - Les séparateurs sont masqués sur les écrans de taille réduite, et des styles spécifiques sont appliqués aux éléments pour une meilleure lisibilité.

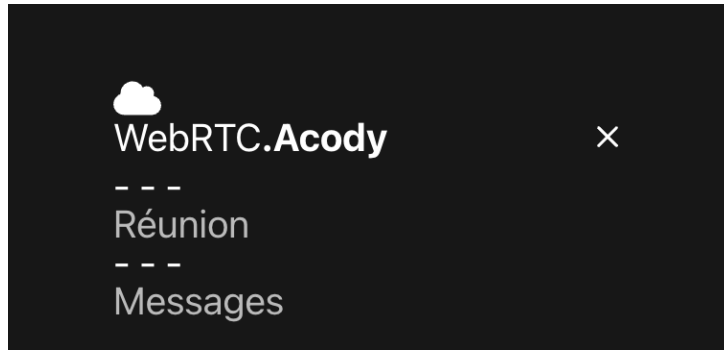


FIG. 0.5: Menu ouvert sur les écrans de taille réduite

8.2.2 Expérience utilisateur

1. **Clarté et Simplicité :** - La barre de navigation est simple et claire, avec seulement deux boutons principaux : "Réunions" et "Messages". Cela évite toute confusion et facilite la navigation.
2. **Logo et Titre :** - L'inclusion d'un logo et du titre "WebRTC.Acody" renforce l'identité visuelle de la page. Cela améliore la reconnaissance du projet et aide les utilisateurs à comprendre le contexte de la page.
3. **Adaptabilité pour les Petits Écrans :** - La barre de navigation est conçue pour s'adapter aux petits écrans. L'utilisation d'un bouton de menu (MenuIcon) pour les écrans de taille réduite montre une considération pour les utilisateurs sur des appareils mobiles.
4. **Effets Visuels au Survole :** - Les effets visuels au survol, tels que le changement d'opacité et la rotation de teinte, ajoutent une dimension interactive à la barre de navigation, améliorant l'expérience visuelle.

8.3 Page - Réunion

La page de réunion Webrtc représente le cœur de l'application WebRTC. Conçue pour faciliter les réunions virtuelles en temps réel, cette page offre une interface conviviale et des fonctionnalités robustes de communication vidéo. L'objectif est de permettre aux utilisateurs de se connecter facilement, de participer à des réunions de manière efficace, tout en offrant une expérience visuelle optimale.

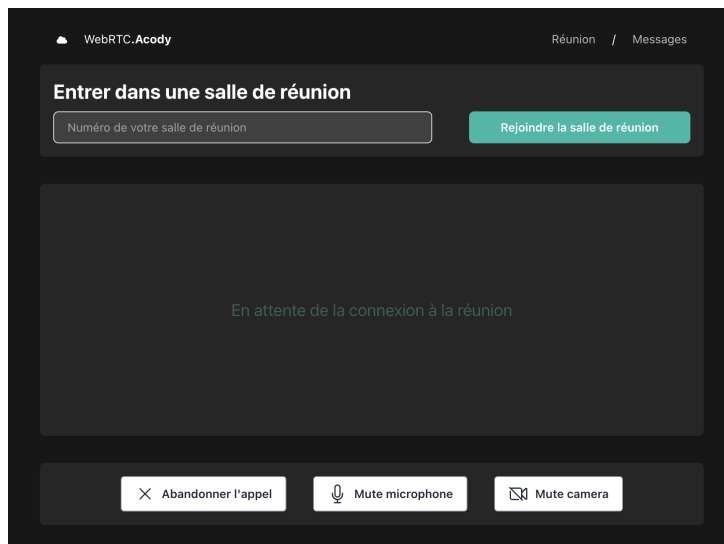


FIG. 0.6: La page de réunion

8.3.1 Réalisation Technique

La page de réunion Webrtc est élaborée avec une approche technique solide, combinant l'utilisation d'un composant de mise en page et des styles CSS soigneusement définis

1. **Composant de Mise en Page :** L'utilisation du composant `Layout` assure une gestion centralisée du contenu, améliorant ainsi la maintenance.
2. **Styles CSS pour Vidéos :** Les règles CSS détaillées assurent une disposition précise des vidéos, garantissant une expérience visuelle optimale.
3. **Affichage Conditionnel Dynamique :** Des règles conditionnelles sont utilisées pour ajuster dynamiquement l'affichage en fonction de l'état de la page.

8.3.2 Expérience Utilisateur

La page de réunion Webrtc offre une expérience utilisateur immersive et interactive, grâce à des fonctionnalités pensées pour faciliter les interactions en temps réel.

1. **Interface Conviviale :** La mise en page soigneusement conçue assure une navigation intuitive et une utilisation facile.
2. **Visualisation Vidéo Optimal :** Les dimensions et les positions des vidéos sont gérées de manière à offrir une présentation visuelle optimale.
3. **Interaction Intuitive :** Des boutons d'action avec des icônes permettent des actions rapides, améliorant ainsi l'expérience utilisateur.
4. **Affichage Conditionnel :** Certains éléments sont dynamiquement affichés ou masqués en fonction de l'état de la page, optimisant ainsi l'interface utilisateur.

8.4 Page - Messagerie

Conçue pour simplifier les échanges professionnels, cette page présente une interface utilisateur conviviale et des fonctionnalités de messagerie robustes.

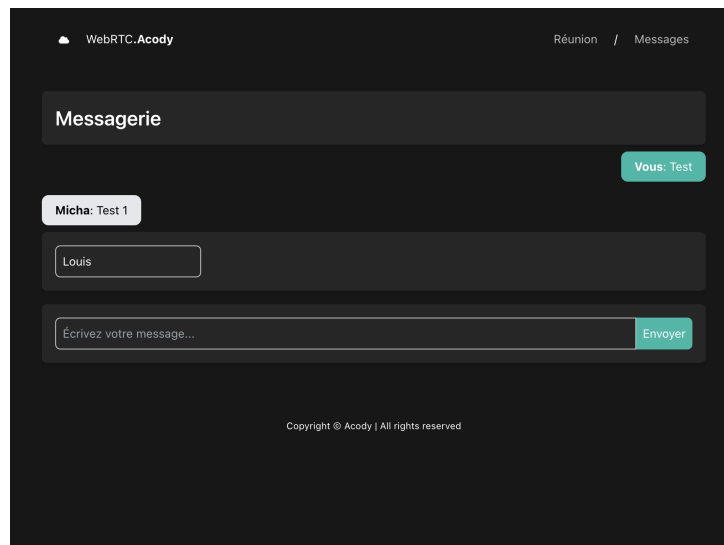


FIG. 0.7: La page de Messagerie

8.4.1 Réalisation Technique

La page de messagerie Webrtc est construite avec une approche technique solide, tirant parti du composant de mise en page et de styles CSS détaillés.

1. **Composant de Mise en Page :** L'utilisation du composant `Layout` permet une gestion centralisée du contenu, simplifiant ainsi la maintenance.
2. **Styles CSS pour la Messagerie :** Des règles CSS détaillées sont employées pour styliser la zone de chat, assurant une expérience visuelle cohérente.

8.4.2 Expérience Utilisateur

La page de messagerie Webrtc offre une expérience utilisateur immersive grâce à des fonctionnalités dédiées à la communication en temps réel.

1. **Interface Intuitive :** La section de messagerie est conçue avec une mise en page claire, facilitant la lecture et l'envoi de messages.
2. **Chat en Temps Réel :** Les messages s'affichent dynamiquement, offrant une communication instantanée et fluide.
3. **Gestion Simple des Utilisateurs :** L'ajout d'un nom d'utilisateur est facilité par une zone de saisie dédiée, permettant une personnalisation rapide.
4. **Envoi Facile de Messages :** La zone de saisie de message et le bouton "Envoyer" simplifient le processus d'envoi de messages.

8.5 Pied de page - Footer

Le pied de page (footer) de l'application est défini de manière concise et centrée, avec une structure HTML minimale. Voici une explication rapide du code du pied de page.

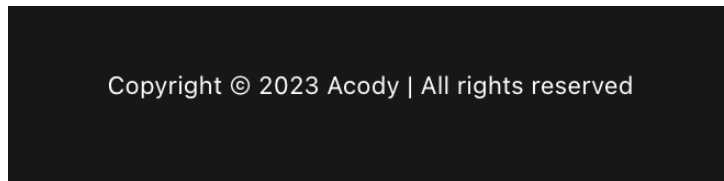


FIG. 0.8: Le pied de page

8.5.1 Réalisation Technique

1. Structure HTML :

- La balise `<footer>` encadre le contenu du pied de page.
- Les classes CSS comme `'mt-auto'`, `'py-4'`, et `'text-center'` sont utilisées pour ajuster la présentation, offrant une marge supérieure automatique, un espacement vertical optimal, et un centrage du texte sur tous les écrans.

2. Dynamisme de l'Année :

- La balise `` permet la mise à jour dynamique de l'année actuelle. c'est effectué grace à un simple JavaScript.

8.5.2 Expérience Utilisateur

Le pied de page assure une expérience utilisateur enrichissante en combinant une présentation soignée avec des effets visuels interactifs.

1. Styles Visuels des Liens :

- Les liens dans le pied de page bénéficient d'une amélioration visuelle lors du survol.
- Une transition fluide d'opacité et une rotation de teinte apportent un dynamisme visuel agréable.

9 Performances de l'application

9.1 Page Load Time

Le rapport ci dessous, fourni par CloudFlare nous permet de voir les performances de notre application.

Temps de chargement global : le temps de chargement de la page de 150 ms est exceptionnel et bien inférieur au temps de chargement moyen sur l'internet, qui dépasse souvent 1 000 ms. Cela indique que le backend et le réseau sont hautement optimisés.

Implications pour l'expérience utilisateur : Un temps de chargement rapide, tel que celui obtenu, est susceptible de contribuer à une expérience utilisateur positive, en réduisant les taux de rebond et en améliorant potentiellement les taux de conversion.
Avantages du rendu côté serveur (SSR) : Le SSR fourni par Astro.js permet au contenu d'être rendu sur le serveur et envoyé au client sous la forme d'une page statique.

Web Analytics for louispluviose.fr



Dec 3rd 2023 12:40 (UTC +01:00) → Dec 10th 2023 12:40 (UTC +01:00)

Filters:

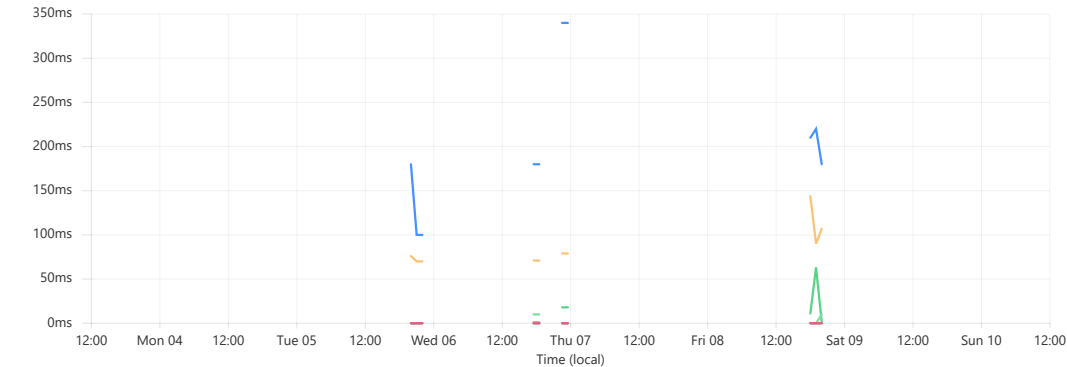
- Site is in louispluviose.fr
- Exclude bots equals Yes
- Referer equals webtrc.louispluviose.fr

P50 (Median)

Page load time summary

Total amount of time it took to load the page.

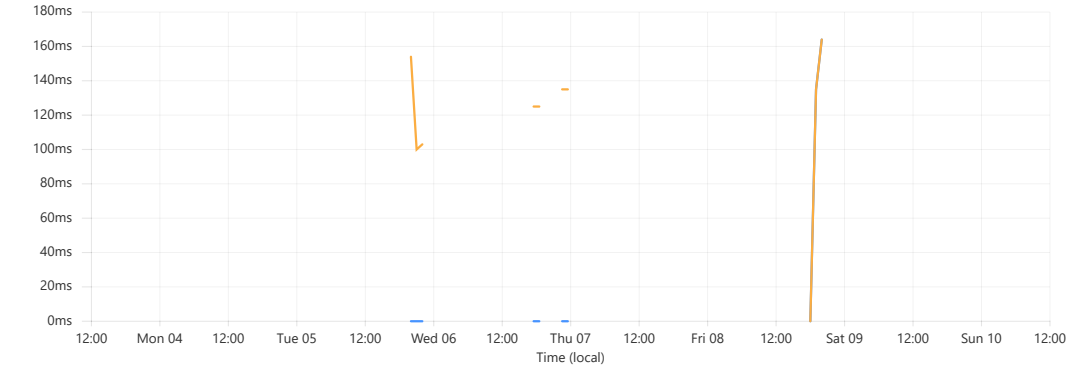
Page Load	DNS	TCP	Request	Response	Processing
150ms	0ms	0ms	72ms	0ms	0ms
Load Event					
0ms					



Paint timing summary

Provides timing information about paint operations during web page construction.

First Paint	First Contentful Paint
0ms	125ms



Page load time by country

France	150 ms	50
--------	--------	----

1 to 1 of 1 items

Page load time by source

Referers			Paths		
webrtc.louispluviose.fr	150 ms	50	/messages	170 ms	28
			/	130 ms	18
			/account	230 ms	4
Hosts			Browsers		
webrtc.louispluviose.fr	150 ms	50	Firefox	130 ms	34
			MobileSafari	210 ms	8
			Opera	230 ms	4
			Chrome	220 ms	4
Operating systems			Device types		
Windows	140 ms	38	Desktop	150 ms	42
iOS	210 ms	8	Mobile	210 ms	8
MacOSX	220 ms	4			

9.2 Core Web Vitals

Le rapport ci dessous, fourni par CloudFlare nous permet de voir une analyse de l'expérience utilisateur de notre application.

Le rapport montre d'excellentes mesures de performance pour votre application. Le Largest Contentful Paint (LCP) pour différents percentiles (P50, P75, P90, P99) varie de 197 ms à 258 ms, ce qui est bien en deçà du seuil requis pour une bonne expérience utilisateur. Ces mesures indiquent que votre application charge toujours rapidement l'élément de contenu le plus volumineux pour la grande majorité des utilisateurs.

En outre, le délai de première entrée (FID) est bon à 100 %, ce qui suggère que les utilisateurs peuvent interagir avec votre application presque immédiatement après le chargement du contenu. Cet aspect est crucial pour l'engagement et la fidélisation des utilisateurs.

Le décalage cumulatif de la mise en page (CLS) est bon à 92 %, avec seulement 8 % d'améliorations à apporter. Cela montre que la plupart des utilisateurs bénéficient d'une mise en page visuelle stable avec un minimum de décalages inattendus, ce qui est essentiel pour une bonne expérience utilisateur.

Dans l'ensemble, l'application de SSR avec Astro.js contribue clairement à une expérience utilisateur rapide et fiable à travers diverses mesures de performance.

Web Analytics for louispluviose.fr



Dec 3rd 2023 12:40 (UTC +01:00) → Dec 10th 2023 12:40 (UTC +01:00)

Filters:

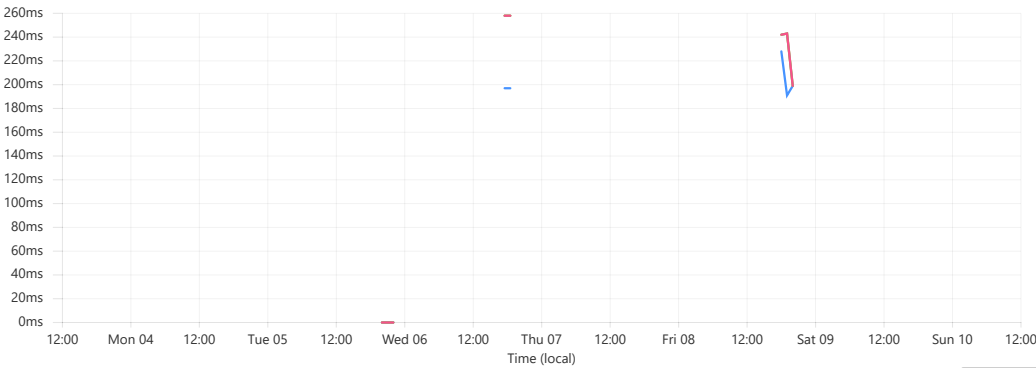
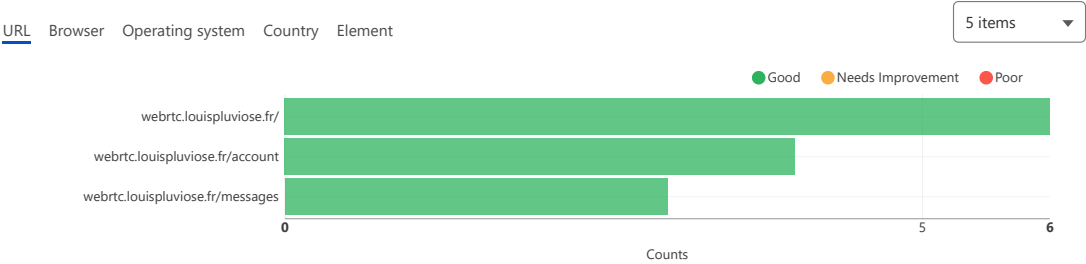
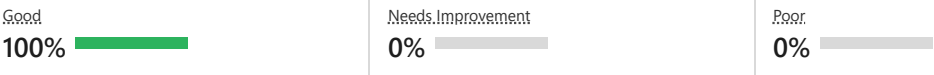
Site is in louispluviose.fr

Exclude bots equals Yes

Referer equals webtrc.louispluviose.fr

Core Web Vitals

Largest Contentful Paint (LCP)



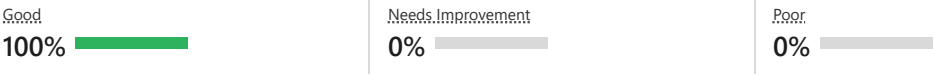
Debug View

5 items

Click each row to expand

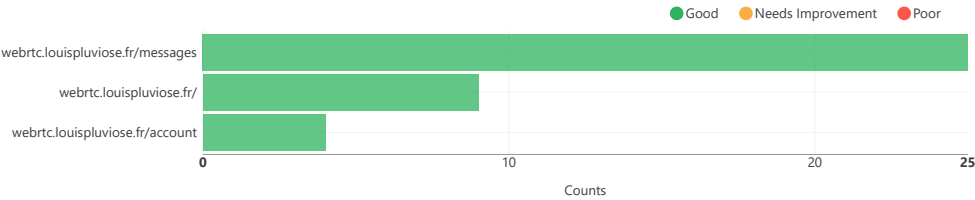
Element	LCP	Counts
No data		

First Input Delay (FID)



URL Browser Operating system Country Element

5 items



Debug View ⓘ

5 items

Click each row to expand

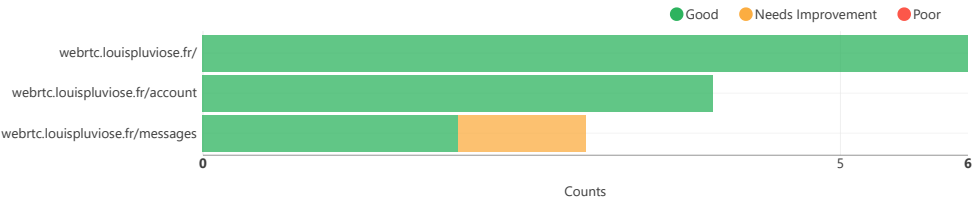
Element	FID ⓘ	Counts
No data		

Cumulative Layout Shift (CLS)



URL Browser Operating system Country Element

5 items



Debug View ⓘ

5 items

Click each row to expand

Element	CLS ⓘ	Counts
body.blob-container.mx-auto.flex.max-w-[872px].flex-col.gap-0.bg-neutral-900.px-10.py-8.text-sm.text-w...	0.13	1

10 Installation de l'application

10.1 Installation en dur

ATTENTION : au préalable, il faut installer NodeJS et NPM sur la machine !

Pour installer l'application en dur, il suffit de cloner le dépôt git suivant : `https://github.com/mIKIII77/SAE5.ROM.03` avec la commande
`git clone https://github.com/mIKIII77/SAE5.ROM.03.git`

Ensuite, il faut naviger dans le dossier `SAE5.ROM.03/webssr` et lancer la commande `npm install` pour installer les dépendances.
Enfin, il faut lancer la commande `npm run build` puis `node ./run-server.mjs` pour lancer l'application.

11 Bugs

11.1 Bugs connus

Nous avons pu détecter 3 bugs dans notre application.

11.1.1 Premier bug

Lors de certains appels, il arrive que la connexion se fasse mal et que un des interlocuteurs ne voit pas l'autre. Voici plusieurs façon de résoudre ce problème :

- 1 : Il faut cliquer sur le bouton "Rejoindre la réunion"
- 2 : Si le problème persiste, Recharger la page
- 3 : Si le problème persiste toujours, il faut quitter la salle et en rejoindre une nouvelle

11.1.2 Deuxième bug

Au moment de raccrocher l'appel, il se peut la vidéo de l'autre interlocuteur reste affichée. Pour résoudre ce problème, il faut rafraichir la page.

11.1.3 Troisième bug

Lors de certains appels, il se peut que l'affichage se fasse mal et que les vidéos des interlocuteurs se chevauchent. Pour résoudre ce problème, il faut rafraichir la page.

11.2 Bugs inconnus

Si vous rencontrez d'autres bugs, merci de nous les signaler à l'adresse suivante : contact@louispluviose.fr.

12 Annexes

12.1 Code du frontend

12.1.1 Page - Réunions

```
1
2 ---
3 import Layout from '../layouts/Layout.astro';
4 ---
5 <Layout
6   seoTitle="Reunion Webrtc - Application de Communication Video en
7     Temps Reel | Acody"
8   seoDesc="Facilitez vos reunions professionnelles avec notre
9     application de projet Webrtc. Effectuez des appels video en
10     temps reel, abandonnez les appels en un clic, et gerez
11     facilement le son et la camera. Optimisez votre collaboration a
12     distance avec des fonctionnalites simples et efficaces."
13 >
14 <div class="flex flex-col bg-neutral-900 rounded-md">
15   <div class="flex-none bg-neutral-800 p-4 rounded-md shadow-md flex
16     flex-col">
17     <h1 class="text-2xl font-bold"> Entrer dans une salle de reunion
18     </h1>
19     <div class="grid grid-rows-1 mt-2">
20       <input id="room-input" name="room" class="w-full px-4 py-2
21         border rounded-md focus:outline-none focus:border-teal-500
22         bg-neutral-700 col-span-6" placeholder="Numero de votre
23         salle de reunion" />
24       <button id="connect-button" type="submit" class="font-semibold
25         w-full bg-teal-500 text-white px-4 py-2 rounded-md
26         hover:bg-teal-700 focus:outline-none col-start-8">
27         Rejoindre la salle de reunion
28       </button>
29     </div>
30   </div>
31 </div>
32
33 <style type="text/css">
34   body, html {
35     margin: 0;
36     padding: 0;
37     height: 100%;
38     overflow: hidden;
39   }
40
41   #video-chat-container {
42     display: grid;
43     grid-template-columns: 1fr 1fr; /* Deux colonnes /
44     grid-template-rows: 1fr 1fr; / Deux rangees */
45     gap: 10px;
46     width: 100vw;
47     height: 30vh;
48     padding: 10px;
49   }
50
51   video {
52     width: 50%;
53     height: 40%;
54     object-fit: cover;
55   }
```

```

45
46     #local-video {
47         grid-row: 2;
48         grid-column: 1;
49     }
50
51     .remote-video-1 {
52         grid-row: 1;
53         grid-column: 1;
54     }
55
56     .remote-video-2 {
57         grid-row: 1;
58         grid-column: 2;
59     }
60
61     .remote-video-3 {
62         grid-row: 2;
63         grid-column: 2;
64     }
65
66     .room-selection-container {
67         display: none;
68     }
69 </style>
70
71
72 <!-- Partie fixe pour la video avec une hauteur personnalisée -->
73 <div class="relative flex-grow">
74     <div class="sticky top-0 bg-neutral-800 p-4 rounded-md shadow-md
75         h-16 my-8 min-h-[300px] flex items-center justify-center">
76         <div class="rounded-md p-4 max-w-sm w-full mx-auto">
77             <div class="animate-pulse">
78                 <h1 id="waitingText" class="text-teal-300 opacity-50
79                     text-xl text-center">En attente de la connexion à la
80                     reunion</h1>
81             </div>
82         </div>
83         <div id="room-selection-container" class="display: hidden">
84             <h1>WebRTC video conference</h1>
85             <label>Enter the number of the room you want to
86                 connect</label>
87             <input id="room-input" type="text" />
88             <button id="connect-button">CONNECT</button>
89         </div>
90
91         <div id="video-chat-container" class="video-position"
92             style="display: none">
93             <div id="local-video-container">
94                 <video id="local-video" autoplay muted></video>
95             </div>
96             <div id="remote-videos-container">
97                 <video id="remote-video-1"></video>
98                 <video id="remote-video-2"></video>
99                 <video id="remote-video-3"></video>
100             </div>
101         </div>
102     </div>
103
104 <!-- Contenu principal -->
105 <div class="bg-neutral-800 p-4 rounded-md flex flex-col md:flex-row
106     justify-center items-center space-y-4 md:space-y-0 md:space-x-8"

```



```

102     data-astro-cid-j7pv25f6="">
103     <button class="bg-white hover:bg-gray-100 text-gray-800
        font-semibold py-2 px-4 border border-gray-400 rounded shadow
        flex items-center" id="hangup-button"
        data-astro-cid-j7pv25f6="">
104         <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0
            0 24 24" stroke-width="1.5" stroke="currentColor" class="w-6
            h-6 mr-2" data-astro-cid-j7pv25f6="">
105             <path stroke-linecap="round" stroke-linejoin="round" d="M6
                18L18 6M6 6L12 12" data-astro-cid-j7pv25f6=""></path>
106         </svg>
107         Abandonner l'appel
108     </button>
109
110     <button class="bg-white hover:bg-gray-100 text-gray-800
        font-semibold py-2 px-4 border border-gray-400 rounded shadow
        flex items-center" id="toggle-mic-button"
        data-astro-cid-j7pv25f6="">
111         <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0
            0 24 24" stroke-width="1.5" stroke="currentColor" class="w-6
            h-6 mr-2" data-astro-cid-j7pv25f6="">
112             <path stroke-linecap="round" stroke-linejoin="round" d="M12
                18.75a6 6 0 06-6v-1.5m-6 7.5a6 6 0 01-6-6v-1.5m6
                7.5v3.75m-3.75 0h7.5M12 15.75a3 3 0 01-3-3V4.5a3 3 0 16
                0v8.25a3 3 0 01-3 3z" data-astro-cid-j7pv25f6=""></path>
113             </svg>
114             Mute microphone
115         </button>
116
117     <button class="bg-white hover:bg-gray-100 text-gray-800
        font-semibold py-2 px-4 border border-gray-400 rounded shadow
        flex items-center" id="toggle-camera-button"
        data-astro-cid-j7pv25f6="">
118         <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0
            0 24 24" stroke-width="1.5" stroke="currentColor" class="w-6
            h-6 mr-2" data-astro-cid-j7pv25f6="">
119             <path stroke-linecap="round" stroke-linejoin="round"
                d="M15.75 10.5l4.72-4.72a.75.75 0 011.28.53v11.38a.75.75 0
                01-1.28.53l-4.72-4.72M12 18.75H4.5a2.25 2.25 0
                01-2.25-2.25V9m12.841 9.091L16.5
                19.5m-1.409-1.409c.407-.407.659-.97.659-1.591v-9a2.25 2.25
                0 00-2.25-2.25h-9c-.621 0-1.184.252-1.591.659m12.182
                12.182L2.909 5.909M1.5 4.5l1.409 1.409"
                data-astro-cid-j7pv25f6=""></path>
120             </svg>
121             Mute camera
122         </button>
123     </div>
124 </div>
125 </div>

```

12.1.2 Page - Messagerie

```
1  ---
2  import Layout from '../layouts/Layout.astro';
3  ---
4  <Layout seoTitle="Messagerie Webrtc - Application de Communication
   Video en Temps Reel" seoDesc="Facilitez vos reunions
   professionnelles avec notre application de projet Webrtc.
5  Effectuez des appels video en temps reel, abandonnez les appels en un
   clic, et gerez facilement le son et la camera.
6  Optimisez votre collaboration a distance avec des fonctionnalites
   simples et efficaces.">
7    <div class="bg-neutral-800 text-white p-4 rounded-md shadow-md
      mt-8">
8      <h1 class="text-2xl font-semibold">Messagerie</h1>
9    </div>
10
11    <!-- Zone de chat (ajoutée) -->
12    <div id="messages" class="messages-container">
13      <!-- Les messages s'afficheront ici -->
14    </div>
15
16
17    <!-- Zone de saisie de nom d'utilisateur -->
18    <div class="bg-neutral-800 p-4 rounded-md shadow-md mb-4 mt-4">
19      <input
20        id="username"
21        type="text"
22        placeholder="Entrez votre nom d'utilisateur..."
23        class="flex-1 p-2 border border-neutral-300 rounded-md
          bg-neutral-800"
24      />
25    </div>
26
27    <!-- Zone de saisie de message -->
28    <div class="bg-neutral-800 p-4 rounded-md shadow-md mb-12">
29      <div class="flex">
30        <input
31          id="message"
32          type="text"
33          placeholder="Ecrivez votre message..."
34          class="flex-1 p-2 border border-neutral-300 rounded-l-md
            bg-neutral-800"
35        />
36        <button id="sendButton" class="bg-teal-500 text-white p-2
          rounded-r-md">Envoyer</button>
37      </div>
38    </div>
39
40    <style>
41      #messages {
42        display: flex;
43        flex-direction: column;
44        overflow-y: auto;
45      }
46    </style>
47
48  </Layout>
```

2: Page - Messagerie

12.1.3 Component - NavBar

```
1  ---
2  import { Astronav, MenuItems, MenuIcon } from 'astro-navbar'
3  ---
4
5  <header class="gap-5 p-5 lg:flex">
6    <Astronav>
7      <svg class="icon mt-1" xmlns="http://www.w3.org/2000/svg"
8        height="1em" viewBox="0 0 640 512"
9        ><!--! Font Awesome Free 6.4.2 by @fontawesome -
10         https://fontawesome.com License -
11         https://fontawesome.com/license (Commercial License)
12         Copyright 2023 Fonticons, Inc. -->
13      <style>
14        svg {
15          fill: #ffffff;
16        }
17      </style><path
18        d="M0 336c0 79.5 64.5 144 144 144H512c70.7 0 128-57.3
19          128-128c0-61.9-44-113.6-102.4-125.4c4.1-10.7 6.4-22.4
20          6.4-34.6c0-53-43-96-96-96c-19.7 0-38.1 6-53.3 16.2C367
21          64.2 315.3 32 256 32C167.6 32 96 103.6 96 192c0 2.7 .1 5.4
22          .2 8.1C40.2 219.8 0 273.2 0 336z"
23      ></path></svg>
24    >
25    <div class="flex w-full justify-between">
26      <a href="/" class="icon">WebRTC<strong>.Acody</strong></a>
27      <div class="block lg:hidden">
28        <MenuIcon class="icon-menus h-5 w-6 text-white" />
29      </div>
30    </div>
31    <MenuItems class="test hidden lg:flex">
32      <ul class="flex flex-col lg:flex-row lg:gap-5">
33        <li class="separator-mob">- - </li>
34        <li><a href="/">Reunion</a></li>
35        <li class="separator"></li>
36        <li class="separator-mob">- - </li>
37        <li><a href="/messages">Messages</a></li>
38      </ul>
39    </MenuItems>
40  </Astronav>
41 </header>
42
43 <style>
44   .separator-mob {
45     display: none;
46   }
47
48   ul.flex li a {
49     opacity: 0.7;
50     -webkit-transition:
51       opacity 0.5s ease-in-out,
52       filter 0.5s ease-in-out;
53     transition:
54       opacity 0.5s ease-in-out,
55       filter 0.5s ease-in-out;
56   }
57
58   ul.flex li a:hover {
59     opacity: 1;
60     -webkit-filter: hue-rotate(90deg);
61   }
```

```

54     filter: hue-rotate(90deg);
55 }
56
57 .icon {
58     -webkit-filter: hue-rotate(90deg);
59     filter: hue-rotate(90deg);
60 }
61
62 @media (max-width: 768px) {
63     .separator {
64         display: none;
65     }
66     .separator-mob {
67         display: block;
68         font-size: 20px;
69     }
70     li {
71         font-size: 20px;
72     }
73     .icon {
74         font-size: 20px;
75     }
76 }
77 </style>

```

3: Component - NavBar

12.1.4 Component - MainHead

```

1  ---
2  import { SEO } from 'astro-seo'
3  const { title, seoTitle, seoDesc } = Astro.props
4  ---
5
6  <head>
7      <SEO title={seoTitle} description={seoDesc} />
8      <meta charset="utf-8" />
9      <meta name="viewport" content="width=device-width" />
10     <meta name="generator" content={Astro.generator} />
11     <title>{title}</title>
12     <link rel="icon" type="image/x-icon"
13         href="../../../public/favicon.svg" sizes="any" />
14     <link rel="preconnect" href="https://fonts.googleapis.com" />
15     <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin
16         />
17     <link href="https://fonts.googleapis.com/css2?family=Roboto"
18         rel="stylesheet" />
19 </head>

```

4: Component - MainHead

12.1.5 Component - Layout

```
1  ---
2  import MainHead from './MainHead.astro'
3  const { title, seoDesc, seoTitle } = Astro.props
4  import Nav from '../components/Nav.astro'
5  import Footer from '../components/Footer.astro'
6  ---
7
8  <html lang="en">
9    <MainHead title={title} seoTitle={seoTitle} seoDesc={seoDesc} />
10   <body
11     class="blob-container mx-auto flex max-w-[872px] flex-col gap-0
12       bg-neutral-900 px-10 py-8 text-sm text-white"
13   >
14     <Nav />
15     <slot />
16     <Footer />
17     <style>
18       @font-face {
19         font-family: 'San Francisco';
20         font-weight: 400;
21         src: url('https://applesocial.s3.amazonaws.com/assets/
22           styles/fonts/sanfrancisco/sanfranciscodisplay.woff');
23       }
24       /* {
25         font-family: 'San Francisco', 'Helvetica Neue', 'Lucida
26           Grande', sans-serif;
27       }
28       body {
29         background-color: #0d0f14;
30       }
31     </style>
32   </body>
33 </html>
```

5: Component - Layout

12.1.6 Component - Footer

```
1  ---
2
3  ---
4
5  <footer class="mt-auto py-4 text-center">
6    <small
7      >Copyright &copy; <span id="copyrighspan">Year</span> Acody | All
        rights reserved</small
8    >
9  </footer>
10
11 <script>
12   document.querySelector('#copyrighspan').textContent = new
        Date().getFullYear().toString()
13 </script>
14
15 <style>
16   a {
17     opacity: 0.7;
18     transition:
19       opacity 0.5s ease-in-out,
20       filter 0.5s ease-in-out;
21   }
22
23   a:hover {
24     opacity: 1;
25     filter: hue-rotate(90deg);
26   }
27 </style>
```

6: Component - Footer

12.2 Code du backend

12.2.1 Client

12.2.1.1 DOM elements

```
1  const roomSelectionContainer =  
    document.getElementById('room-selection-container');  
2  const roomInput = document.getElementById('room-input');  
3  const connectButton = document.getElementById('connect-button');  
4  const videoChatContainer =  
    document.getElementById('video-chat-container');  
5  const localVideoComponent = document.getElementById('local-video');
```

7: DOM Elements

12.2.1.2 Variables globales

```
1  const socket = io();  
2  const mediaConstraints = { audio: true, video: { width: 1280, height:  
    720 } };  
3  let localStream;  
4  let roomId;  
5  let peerConnections = {}; // Dictionary to hold all peer connections  
6  
7  const iceServers = {  
8      iceServers: [  
9          { urls: 'stun:stun.l.google.com:19302' }, // Serveur STUN  
            existant  
10         // Ajout de la configuration TURN  
11         {  
12             urls: 'turn:relay1.expressturn.com:3478', // URL du  
                serveur TURN  
13             username: 'efJJOL8OUOGANH5VOA', // Nom d'utilisateur  
14             credential: 'L05Tdr8aoKohTHDL' // Mot de passe  
15         }  
16     ]  
17 };
```

8: Variables

12.2.1.3 Button listeners

```
1  connectButton.addEventListener('click', () => {  
2      joinRoom(roomInput.value);  
3  });  
4  
5  const hangUpButton = document.getElementById('hangup-button');  
6  const toggleMicButton = document.getElementById('toggle-mic-button');  
7  const toggleCameraButton =  
    document.getElementById('toggle-camera-button');  
8  
9  hangUpButton.addEventListener('click', hangUpCall);  
10 toggleMicButton.addEventListener('click', toggleMicrophone);  
11 toggleCameraButton.addEventListener('click', toggleCamera);  
12  
13 let isRoomCreator = false;
```

9: Button Listeners

12.2.1.4 Socket event callbacks

```
1  socket.on('room_created', async () => {
2    console.log('Socket event callback: room_created');
3    await setLocalStream(mediaConstraints);
4    isRoomCreator = true;
5  });
6
7  socket.on('room_joined', async () => {
8    console.log('Socket event callback: room_joined');
9    await setLocalStream(mediaConstraints);
10   isRoomCreator = false;
11   socket.emit('start_call', roomId);
12 });
13
14 socket.on('full_room', () => {
15   console.log('Socket event callback: full_room');
16   alert('The room is full, please try another one');
17 });
18
19 socket.on('start_call', async () => {
20   console.log('Socket event callback: start_call');
21   if (isRoomCreator) {
22     createPeerConnections();
23   }
24 });
25
26 socket.on('webrtc_offer', async (data) => {
27   console.log('Socket event callback: webrtc_offer');
28
29   if (!localStream) {
30     console.log("Waiting to set local stream before handling offer");
31     await setLocalStream(mediaConstraints);
32   }
33
34   if (!peerConnections[data.peerId]) {
35     await setupPeerConnection(data.peerId, false); // false = not an
36     initiator
37   }
38   await handleOffer(data);
39 });
40
41 socket.on('webrtc_answer', (data) => {
42   console.log('Socket event callback: webrtc_answer');
43   handleAnswer(data);
44 });
45
46 socket.on('webrtc_ice_candidate', (data) => {
47   console.log('Socket event callback: webrtc_ice_candidate');
48   handleIceCandidate(data);
49 });
50
51 socket.on('new_peer', async (peerId) => {
52   console.log('Socket event callback: new_peer');
53   await createPeerConnection(peerId, false);
54 });
```

10: Socket Event Callbacks

12.2.1.5 Fonctions

```
1  async function joinRoom(room) {
2    if (room === '') {
```



```

3     alert('Please type a room ID');
4 } else {
5     roomId = room;
6     socket.emit('join', room);
7     showVideoConference();
8
9     try {
10         localStream = await
            navigator.mediaDevices.getUserMedia(mediaConstraints);
11         document.getElementById('local-video').srcObject = localStream;
12     } catch (error) {
13         console.error('Could not get user media', error);
14     }
15 }
16 }
17
18 function showVideoConference() {
19     roomSelectionContainer.style = 'display: none';
20     videoChatContainer.style = 'display: block';
21
22     // Masquer le texte d'attente
23     const waitingText = document.getElementById('waitingText');
24     if (waitingText) {
25         waitingText.style.display = 'none';
26     }
27 }
28
29
30 async function setLocalStream(mediaConstraints) {
31     if (!localStream) {
32         try {
33             const stream = await
                navigator.mediaDevices.getUserMedia(mediaConstraints);
34             console.log('Local stream obtained', stream);
35             localStream = stream;
36             const localVideoComponent =
                document.getElementById('local-video');
37             if (localVideoComponent) {
38                 localVideoComponent.srcObject = stream;
39             } else {
40                 console.error('The local video element was not found in the
                    DOM. ');
41             }
42         } catch (error) {
43             console.error('Could not get user media', error);
44         }
45     }
46 }
47
48 async function handleNewPeer(peerId) {
49     const peerConnection = await createPeerConnection(peerId);
50     localStream.getTracks().forEach(track => {
51         peerConnection.addTrack(track, localStream);
52     });
53
54     const offer = await peerConnection.createOffer();
55     await peerConnection.setLocalDescription(offer);
56     socket.emit('webrtc_offer', {
57         type: 'webrtc_offer',
58         sdp: offer,
59         roomId,
60         peerId,
61     });

```

```

62 }
63
64 async function setupPeerConnection(peerId, isInitiator) {
65     const peerConnection = new RTCPeerConnection(iceServers);
66
67     if (localStream) {
68         localStream.getTracks().forEach(track =>
69             peerConnection.addTrack(track, localStream));
70     } else {
71         console.error("Local stream is not defined in setupPeerConnection");
72         return;
73     }
74
75     peerConnection.ontrack = (event) => addRemoteStream(event.streams[0],
76         peerId);
77     peerConnection.onicecandidate = (event) => handleIceEvent(event,
78         peerId);
79
80     peerConnections[peerId] = peerConnection;
81
82     if (isInitiator) {
83         const offer = await peerConnection.createOffer();
84         await peerConnection.setLocalDescription(offer);
85         socket.emit('webrtc_offer', { roomId, sdp: offer, peerId });
86     }
87 }
88
89 function addRemoteStream(event, peerId) {
90     let remoteVideoElement =
91         document.getElementById('remote-video-${peerId}');
92     if (!remoteVideoElement) {
93         remoteVideoElement = createRemoteVideoElement(peerId);
94     }
95
96     let stream;
97     if (event.streams && event.streams.length > 0) {
98         // Use the stream from the event if it exists
99         stream = event.streams[0];
100     } else {
101         // Create a new stream and add the track to it
102         stream = new MediaStream();
103         if (event.track) {
104             stream.addTrack(event.track);
105         }
106     }
107
108     // Additional logging to debug
109     console.log('Adding remote stream for peer ${peerId}', stream);
110     if (stream.getTracks().length === 0) {
111         console.error('No tracks in the remote stream');
112     }
113
114     remoteVideoElement.srcObject = stream;
115     remoteVideoElement.muted = false; // Ensure remote video is not muted
116     remoteVideoElement.volume = 1; // Ensure volume is set to maximum
117 }
118
119 function createRemoteVideoElement(peerId) {
120     const remoteVideo = document.createElement('video');
121     remoteVideo.id = 'remote-video-${peerId}';
122     remoteVideo.autoplay = true;
123     remoteVideo.playsInline = true;
124     remoteVideo.classList.add('remote-video', ...tailwindClasses);

```

```

121 |
122 | document.getElementById('remote-videos-container')
123 |     .appendChild(remoteVideo);
124 | return remoteVideo;
125 | }
126 |
127 |
128 | async function handleIceEvent(event, peerId) {
129 |     if (event.candidate) {
130 |         socket.emit('webrtc_ice_candidate', {
131 |             roomId,
132 |             candidate: event.candidate,
133 |             peerId
134 |         });
135 |     }
136 | }
137 |
138 | async function createPeerConnection(peerId) {
139 |     const peerConnection = new RTCPeerConnection(iceServers);
140 |
141 |     \begin{verbatim}
142 |     localStream.getTracks().forEach(track =>
143 |         peerConnection.addTrack(track, localStream));
144 |     \end{verbatim}
145 |     // localStream.getTracks().forEach(track =>
146 |         peerConnection.addTrack(track, localStream));
147 |     \end{verbatim}
148 |     localStream.getTracks().forEach(track =>
149 |         peerConnection.addTrack(track, localStream));
150 |
151 |     // Gestion de l'ajout des streams distants
152 |     peerConnection.ontrack = (event) => addRemoteStream(event, peerId);
153 |
154 |     // Gestion de l'ajout des streams distants
155 |     peerConnection.ontrack = (event) => {
156 |         let remoteVideoElement =
157 |             document.getElementById('remote-video-${peerId}');
158 |         if (!remoteVideoElement) {
159 |             remoteVideoElement = document.createElement('video');
160 |             remoteVideoElement.id = 'remote-video-${peerId}';
161 |             remoteVideoElement.autoplay = true;
162 |             remoteVideoElement.playsInline = true;
163 |             remoteVideoElement.classList.add('remote-video');
164 |             document.getElementById('remote-videos-container')
165 |                 .appendChild(remoteVideoElement);
166 |         }
167 |         remoteVideoElement.srcObject = event.streams[0];
168 |     };
169 |
170 |     // Gestion des candidats ICE
171 |     peerConnection.onicecandidate = (event) => {
172 |         if (event.candidate) {
173 |             console.log('Sending ICE candidate to peer ${peerId}',
174 |                 event.candidate);
175 |             socket.emit('webrtc_ice_candidate', {
176 |                 type: 'webrtc_ice_candidate',
177 |                 candidate: event.candidate,
178 |                 roomId,
179 |                 peerId,
180 |             });
181 |         }
182 |     };

```

```

179
180 peerConnections[peerId] = peerConnection;
181
182 if (!isRoomCreator) {
183     const offer = await peerConnection.createOffer();
184     await peerConnection.setLocalDescription(offer);
185     socket.emit('webrtc_offer', {
186         type: 'webrtc_offer',
187         sdp: offer,
188         roomId,
189         peerId,
190     });
191 }
192 }
193
194
195 async function handleOffer(data) {
196     try {
197         if (!peerConnections[data.peerId]) {
198             await createPeerConnection(data.peerId);
199         }
200
201         const peerConnection = peerConnections[data.peerId];
202         console.log(L'Etat de la connexion Peer avant setRemoteDescription:
203             peerConnection.signalingState');
204
205         await peerConnection.setRemoteDescription(new
206             RTCSessionDescription(data.sdp));
207
208         if (peerConnection.cachedIceCandidates) {
209             peerConnection.cachedIceCandidates.forEach(cachedCandidate => {
210                 peerConnection.addIceCandidate(new
211                     RTCIceCandidate(cachedCandidate));
212             });
213             peerConnection.cachedIceCandidates = [];
214         }
215
216         const answer = await peerConnection.createAnswer();
217         await peerConnection.setLocalDescription(answer);
218
219         socket.emit('webrtc_answer', {
220             type: 'webrtc_answer',
221             sdp: answer,
222             roomId,
223             peerId: data.peerId,
224         });
225     } catch (error) {
226         console.error('Erreur dans handleOffer pour le pair
227             ${data.peerId}:', error);
228     }
229 }
230
231
232 async function handleAnswer(data) {
233     const peerConnection = peerConnections[data.peerId];
234     console.log('Peer connection state before setting remote description:
235         ${peerConnection.signalingState}');
236
237     if (peerConnection.signalingState === 'have-local-offer') {
238         try {
239             await peerConnection.setRemoteDescription(new
240                 RTCSessionDescription(data.sdp));
241             console.log('Remote description set for peer ${data.peerId}');
242         } catch (error) {
243

```

```

236         console.error('Error in handleAnswer for peer ${data.peerId}:',
237             error);
238     } else {
239         console.log('Peer connection not in the correct state to set remote
240             description, current state: ${peerConnection.signalingState}');
241     }
242 }
243
244 async function handleIceCandidate(data) {
245     const peerConnection = peerConnections[data.peerId];
246     if (peerConnection) {
247         if (!peerConnection.remoteDescription) {
248             console.log("Queueing ICE candidate as remote description is not
249                 yet set");
250             if (!peerConnection.cachedIceCandidates) {
251                 peerConnection.cachedIceCandidates = [];
252             }
253             peerConnection.cachedIceCandidates.push(data.candidate);
254         } else {
255             console.log("Adding ICE candidate");
256             await peerConnection.addIceCandidate(new
257                 RTCIceCandidate(data.candidate));
258         }
259     }
260 }
261
262 function handleNewICECandidateMsg(data) {
263     const peerConnection = peerConnections[data.peerId];
264     peerConnection.addIceCandidate(new RTCIceCandidate(data.candidate));
265 }
266
267 function sendIceCandidate(candidate, peerId) {
268     socket.emit('webrtc_ice_candidate', {
269         roomId,
270         candidate,
271         peerId,
272     });
273 }
274
275 function addVideoStream(videoElement, stream, isLocal = false) {
276     \texttt{console.log('Adding videoElement.srcObject = stream;')}
277     videoElement.srcObject = stream;
278     videoElement.autoplay = true;
279     videoElement.playsInline = true;
280     videoElement.muted = isLocal;
281     if (isLocal) {
282         videoElement.id = 'local-video';
283         videoElement.style.backgroundColor = 'red';
284     } else {
285         videoElement.classList.add('remote-video');
286         videoElement.style.backgroundColor = 'green';
287     }
288     videoChatContainer.appendChild(videoElement);
289 }
290
291 function handleRemoteStreamAdded(stream, peerId) {
292     console.log('handleRemoteStreamAdded called with peerId: ${peerId}');
293     let videoElementId = 'remote-video-${peerId}';
294     let remoteVideoElement = document.getElementById(videoElementId);

```

```

295
296 if (!remoteVideoElement) {
297     console.log('Creating new video element for peer ${peerId}');
298     remoteVideoElement = document.createElement('video');
299     remoteVideoElement.id = videoElementId;
300     remoteVideoElement.autoplay = true;
301     remoteVideoElement.playsInline = true;
302     remoteVideoElement.classList.add('remote-video');
303     document.getElementById('remote-videos-container')
304         .appendChild(remoteVideoElement);
305 }
306 else {
307     console.log('Replacing video element for peer ${peerId}');
308 }
309
310 remoteVideoElement.srcObject = stream;
311 }
312
313
314
315 function hangUpCall() {
316     console.log("Hang Up Call");
317     for (let peerId in peerConnections) {
318         peerConnections[peerId].close();
319         delete peerConnections[peerId];
320     }
321
322     if (localStream) {
323         localStream.getTracks().forEach(track => track.stop());
324         localStream = null;
325     }
326
327     let remoteVideosContainer =
328         document.getElementById('remote-videos-container');
329     while (remoteVideosContainer.firstChild) {
330         remoteVideosContainer.removeChild(remoteVideosContainer.firstChild);
331     }
332
333     const waitingText = document.getElementById('waitingText');
334     if (waitingText) {
335         waitingText.style.display = 'block';
336     }
337
338     videoChatContainer.style.display = 'none';
339
340     roomSelectionContainer.style.display = 'none';
341 }
342
343 function toggleMicrophone() {
344     const audioTrack = localStream.getAudioTracks()[0];
345     if (audioTrack) {
346         audioTrack.enabled = !audioTrack.enabled;
347         console.log("Microphone toggled. Now enabled:", audioTrack.enabled);
348
349         // Update the track on all peer connections
350         for (let peerId in peerConnections) {
351             const sender = peerConnections[peerId].getSenders().find(s =>
352                 s.track.kind === audioTrack.kind);
353             if (sender) {
354                 sender.replaceTrack(audioTrack);
355             }
356         }
357     }
358 }

```

```

356     }
357 }
358
359
360 function toggleCamera() {
361     const videoTrack = localStream.getVideoTracks()[0];
362     if (videoTrack) {
363         videoTrack.enabled = !videoTrack.enabled;
364         console.log("Camera toggled. Now enabled:", videoTrack.enabled);
365
366         // Update the track on all peer connections
367         for (let peerId in peerConnections) {
368             const sender = peerConnections[peerId].getSenders().find(s =>
369                 s.track.kind === videoTrack.kind);
370             if (sender) {
371                 sender.replaceTrack(videoTrack);
372             }
373         }
374     }

```

11: Fonctions Principales

12.2.2 Messagerie

```
1  document.addEventListener('DOMContentLoaded', () => {
2      const socket = io();
3      const userId = Date.now().toString();
4
5      socket.on('connect', () => {
6          console.log('Connecte au serveur Socket.io');
7      });
8
9      const sendButton = document.getElementById('sendButton');
10     const messagesDiv = document.getElementById('messages');
11     const usernameInput = document.getElementById('username');
12     const messageInput = document.getElementById('message');
13
14     if (sendButton && messagesDiv && usernameInput && messageInput) {
15         sendButton.addEventListener('click', () => {
16             const username = usernameInput.value || 'Anonyme';
17             const message = messageInput.value;
18
19             if (message.trim() !== '') {
20                 socket.emit('chat_message', { userId, username, message });
21                 messageInput.value = '';
22             }
23         });
24
25         socket.on('chat_message', (data) => {
26             const messageContainer = document.createElement('div');
27             messageContainer.className = "flex my-2";
28
29             const messageElement = document.createElement('div');
30             messageElement.className = "px-4 py-2 rounded-lg shadow";
31
32             const senderName = document.createElement('strong');
33             senderName.textContent = data.userId === userId ? 'Vous' :
34                 data.username;
35
36             messageElement.appendChild(senderName);
37             messageElement.appendChild(document.createTextNode(`:
38                 ${data.message}`));
39
40             if (data.userId === userId) {
41                 messageContainer.classList.add('justify-end');
42                 messageElement.classList.add('bg-teal-500', 'text-white');
43             } else {
44                 messageContainer.classList.add('justify-start');
45                 messageElement.classList.add('bg-gray-200', 'text-black');
46             }
47
48             messageContainer.appendChild(messageElement);
49             messagesDiv.appendChild(messageContainer);
50             messagesDiv.scrollTop = messagesDiv.scrollHeight;
51         });
52     }
53 }
```

12: Code de la messagerie

12.2.3 Serveur

```
1  import express from 'express';
2  import { createServer } from 'http';
3  import { Server as SocketIOServer } from 'socket.io';
4  import { handler as ssrHandler } from '../dist/server/entry.mjs';
5
6  const app = express();
7  const base = '/';
8
9  // Serve the static files from the Astro build
10 app.use(base, express.static('dist/client/'));
11 app.use('/scripts', express.static('dist/scripts'));
12
13 // Handle SSR for all get requests
14 app.get('*', ssrHandler);
15
16 // Create an HTTP server and configure Socket.io
17 const server = createServer(app);
18 const io = new SocketIOServer(server);
19
20 let messages = [];
21
22 // Socket.io event handling
23 io.on('connection', (socket) => {
24   console.log('A user connected: ' + socket.id);
25
26   messages.forEach((message) => {
27     socket.emit('chat_message', message);
28   });
29
30   socket.on('chat_message', (data) => {
31     messages.push(data);
32     console.log('message: ' + data);
33     io.emit('chat_message', data);
34   });
35
36   socket.on('disconnect', () => {
37     console.log('User disconnected: ' + socket.id);
38   });
39
40   socket.on('join', (roomId) => {
41     const selectedRoom = io.sockets.adapter.rooms.get(roomId);
42     const numberOfClients = selectedRoom ? selectedRoom.size : 0;
43
44     console.log(`Client ${socket.id} requesting to join room
45       ${roomId}`);
46     if (numberOfClients < 4) {
47       console.log(`Joining room ${roomId} and emitting room_joined
48         socket event`);
49       socket.join(roomId);
50       socket.to(roomId).emit('new_peer', socket.id); // Notify others
51       in the room
52     } else {
53       console.log(`Can't join room ${roomId}, emitting full_room
54         socket event`);
55       socket.emit('full_room', roomId);
56     }
57   });
58
59   socket.on('start_call', (roomId) => {
60     console.log(`Broadcasting start_call event to peers in room
61       ${roomId}`);
62   });
63 });
```

```

57     socket.broadcast.to(roomId).emit('start_call');
58 });
59
60 socket.on('webrtc_offer', (data) => {
61     console.log('Received offer from ${socket.id} in room
62         ${data.roomId}');
63     socket.to(data.peerId).emit('webrtc_offer', {
64         peerId: socket.id,
65         sdp: data.sdp
66     });
67 });
68
69 socket.on('webrtc_answer', (data) => {
70     console.log('Received answer from ${socket.id} in room
71         ${data.roomId}');
72     socket.to(data.peerId).emit('webrtc_answer', {
73         peerId: socket.id,
74         sdp: data.sdp
75     });
76 });
77
78 socket.on('webrtc_ice_candidate', (data) => {
79     console.log('Received ICE candidate from ${socket.id} for room
80         ${data.roomId}');
81     socket.to(data.peerId).emit('webrtc_ice_candidate', {
82         peerId: socket.id,
83         label: data.label,
84         candidate: data.candidate
85     });
86 });
87
88 // Handle user disconnect
89 socket.on('disconnect', () => {
90     console.log('User disconnected: ' + socket.id);
91 });
92
93 // Start the server on the specified port or default to port 8080
94 const PORT = process.env.PORT || 8080;
95 server.listen(PORT, () => {
96     console.log('Server is running on port ${PORT}');
97 });

```

13: Code du serveur

13 Conclusion

Ce projet a représenté un défi stimulant et enrichissant pour notre groupe. L'opportunité d'explorer et de maîtriser la technologie WebRTC a marqué une étape importante dans notre parcours de développement. La création d'une application de communication en temps réel nous a permis non seulement d'approfondir nos compétences en Docker et NodeJS, mais aussi de comprendre les subtilités de la communication en ligne moderne.

Bien que nous ayons nourri l'ambition d'intégrer des fonctionnalités supplémentaires telles que le partage d'écran, un tableau blanc interactif, la gestion de contacts, la création de comptes utilisateurs, et même la liaison avec un serveur Asterisk, le temps imparti n'a pas suffi pour réaliser toutes ces idées. Cependant, cette limitation n'a en rien diminué notre fierté face aux accomplissements réalisés. Nous avons réussi à créer une application fonctionnelle et intuitive, dotée de fonctionnalités de base efficaces et faciles à utiliser.

Ce sentiment d'accomplissement est renforcé par notre engagement à poursuivre le développement de ce projet. Notre vision à long terme est de le rendre plus complet, plus performant et de le transformer en un produit open source. En continuant à travailler sur ce projet, nous espérons non seulement enrichir notre propre expérience, mais également contribuer à la communauté en offrant une solution de communication avancée et accessible à tous.

En somme, ce projet est plus qu'une simple réalisation technique ; il représente notre passion pour l'innovation et notre désir de repousser les limites de ce qui est possible dans le domaine de la communication numérique. Nous sommes impatients de voir où ce chemin nous mènera et sommes déterminés à faire de ce projet une référence dans le monde de la communication WebRTC.