REPOBLIKAN'I MADAGASIKARA
FITIAVANA - TANINDRAZANA - FANDROSOANA

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE SUPÉRIEURE POLYTECHNIQUE D'ANTSIRANANA

Mention STIC

PROJET DE FIN DE SEMESTRE

Parcours Télécommunication et Réseau

Réalisation d'une application de communication en temps réel (WebRTC)

Par :

ANDRIANARISOA Daniel

Encadreurs :

Mme. RAOELIVOLOLONA Tefy

Mr. RAKOTOARIJAONA Raonirivo

⋆ Année Universitaire : 2019 - 2020 ⋆

REPOBLIKAN'I MADAGASIKARA
FITIAVANA - TANINDRAZANA - FANDROSOANA

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE SUPÉRIEURE POLYTECHNIQUE D'ANTSIRANANA

Mention STIC

PROJET DE FIN DE SEMESTRE

Parcours Télécommunication et Réseau

Réalisation d'une application de communication en temps réel (WebRTC)

Par :

ANDRIANARISOA Daniel

Encadreurs :

Mme. RAOELIVOLOLONA Tefy
Mr. RAKOTOARIJAONA Raonirivo

Membre de jury :

Pr./Dr./Mme/Mr. NOM Prénoms	Grade	Titre
Pr./Dr./Mme/Mr. NOM Prénoms,	Grade	Titre
Pr./Dr./Mme/Mr. NOM Prénoms,	Grade	Titre
Pr./Dr./Mme/Mr. NOM Prénoms,	Grade	Titre

⋆ Année Universitaire : 2019 - 2020 ⋆

Remerciements

Je tiens à remercier spécialement Madame RAOELIVOLOLONA Tefy et Monsieur RAKOTOARIJAONA Rivo pour leur encadrement, orientation, conseils et disponibilité qu'ils nous ont témoignés pour me permettre de mener à bien ce projet.

Je souhaite aussi à exprimer mes remerciements aux membres du jury de soutenance, pour leur précieux jugement de ce projet. Qu'ils trouvent ici l'expression d'une reconnaissance profonde et sincère.

Sans oublier de remercier mes collègues étudiants de L'ESP Antsiranana et les gens de mon entourage, pour l'environnement serrein qui'ls ont apportés, propice à la completion de ce projet.

Un grand merci à toutes les personnes qui ont contribué au succès de ce projet et qui m'ont aidée lors de la rédaction de ce rapport.

Table des matières

Liste des figures

Liste des acronymes et symboles

Acronymes

API Application Programming Interface

CSS Cascading Style Sheet

HTML HyperText Markup Language

HTTP HyperText Transfert Protocol

ICE Interactive Connectivity Establishment

IETF Internet Engineering Task Force

IP Internet Protocol

RTC Real Time Communication

SDP Session Description Protocol

SIP Session Initiation Protocol

UDP User Datagram Protocol

W3C World Wide Web Consortium

WebRTC Web Real-Time Communications

Chapitre 1

Introduction au WebRTC

Sommaire

1.1 Introduction

La conception structurelle web classique est basée sur un modèle client-serveur, dans lequel les navigateurs envoient une demande de contenu HTTP au serveur web, qui répond en leur envoyant une réponse contenant les informations demandées. Le décodage de la réponse par les navigateurs est possible car elles contiennent les composants pour se conformer aux normes du W3C, ici en l'occurrence, de balises HTML, styles CSS et code Javascript[2, 8].

De même, en ce qui concerne le modèle de navigateurs, afin de supporter WebRTC, chaque navigateur doit ajouter un autre composant pour pouvoir la prendre en charge. La plupart des principaux navigateurs ont ajouté le composant WebRTC depuis 2014[2].

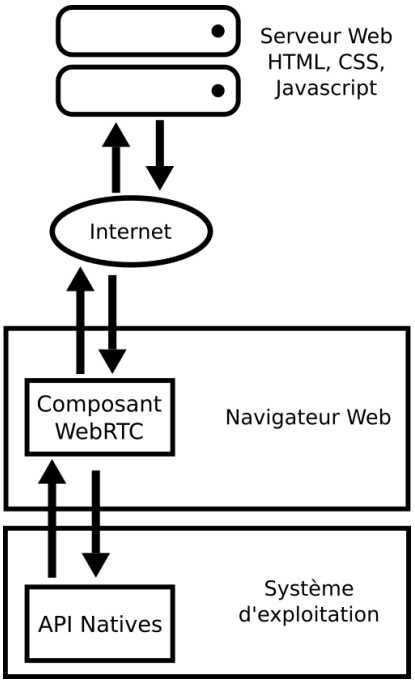

FIGURE 1.1 – composant WebRTC

À partir du diagramme 1.1, nous pouvons clairement voir le composant WebRTC intégré dans le navigateur. Il se charge d'appeler les API audio et vidéo natives du système d'exploitation.

Javascript joue un rôle important dans WebRTC, car la technologie s'exécute sur le navigateur. Javascript est le choix idéal en tant que langage de programmation pour assurer la médiation entre le WebRTC et l'application via un ensemble d'API exposées[2].

1.2 Normes WebRTC

Les normes WebRTC sont actuellement développées conjointement par le W3C et l'IETF. Le W3C travaille sur la définition des API nécessaires aux applications web Javascript pour interagir avec la fonction RTC du navigateur. L'IETF développe les protocoles utilisés par la fonction RTC du

navigateur pour communiquer avec un autre navigateur ou point de terminaison de communications Internet[3, p.11]. Le travail du W3C est centré sur le groupe de travail WebRTC et le travail de l'IETF est centré sur le groupe de travail RTCWeb (Real-Time Communications Web). Les deux groupes sont indépendants, mais se coordonnent étroitement[3, p.11].

1.3 Architecture WebRTC

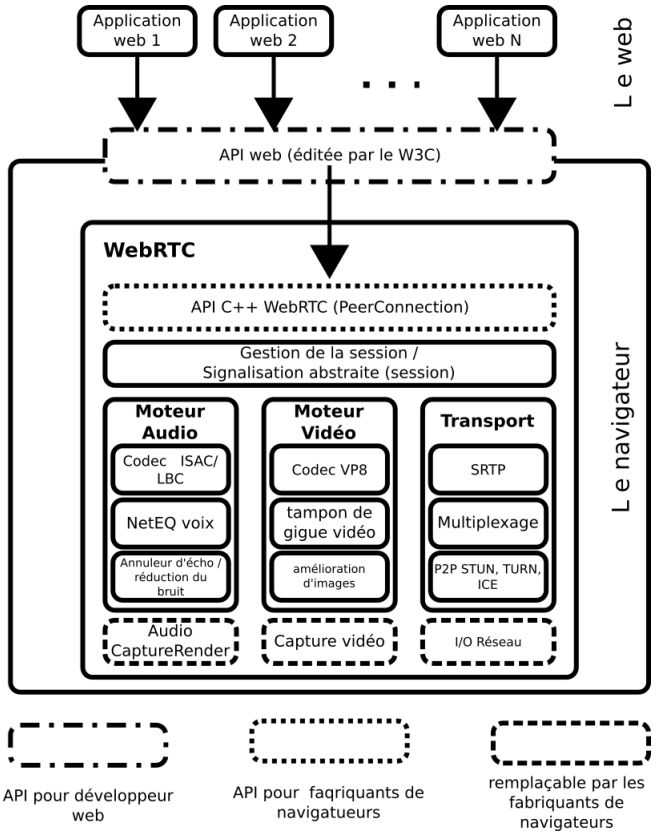

FIGURE 1.2 – architecture WebRTC

Le navigateur cache la grande partie de l'implémentation WebRTC, tout le travail de capture des médias, de la description de session et des transmissions réseau, tout est abstrait de la surface et n'expose que de simples API. L'application ne peut appeler que l'API web qui est normalisée par le W3C, à consommer par les développeurs web[2].

1.3.1 Le triangle WebRTC

Le scénario le plus répandu est probablement celui où les deux navigateurs exécutent la même application web WebRTC, téléchargée à partir de la même page web. Cela produit le «triangle» WebRTC illustré à la figure 1.3. Cet arrangement est appelé triangle en raison de la forme de la signalisation (côtés du triangle) et des flux médiatiques ou de données (base du triangle) entre les

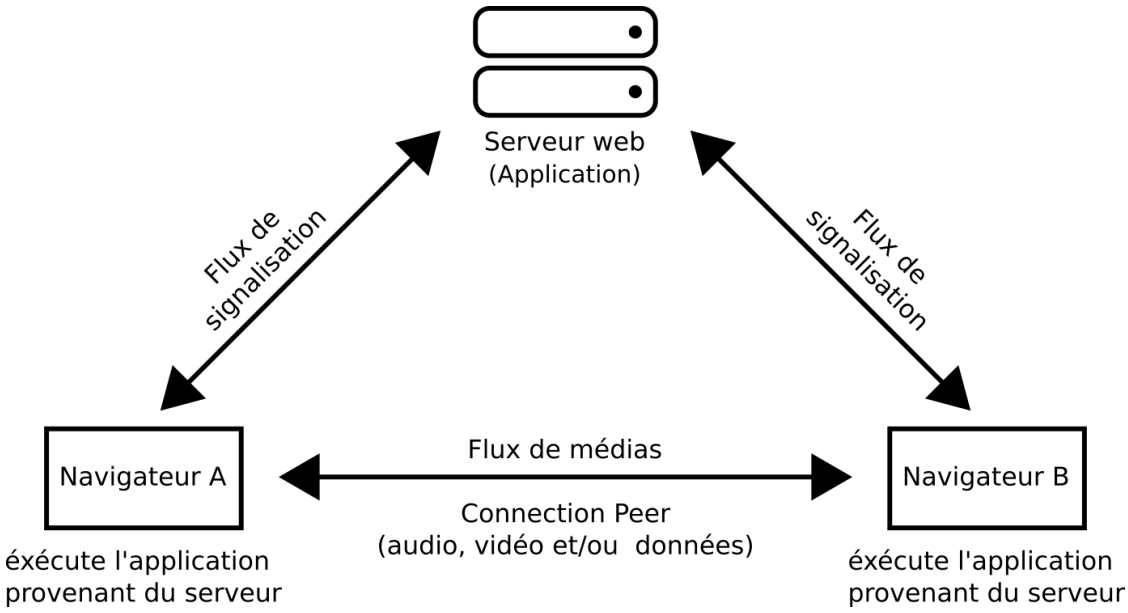

FIGURE 1.3 – triangle WebRTC

trois éléments. Une connexion peer établit le transport pour les médias vocaux et vidéo et les flux de canaux de données directement entre les navigateurs[3, p.4-5].

1.4 Composants Importants du WebRTC

1.4.1 Flux de médias

Les appareils d'aujourd'hui sont dotés de plusieurs capacités matérielles multimédias. Dans les appareils mobiles, vous obtenez une caméra avant, une caméra arrière et un microphone audio. Sur les ordinateurs de bureau et les tablettes, vous obtenez le même ensemble de sources multimédias. Chacune de ces sources multimédia peut générer un flux multimédia[2].

Ce flux multimédia peut être capturé par l'API du navigateur WebRTC et fournira ce flux en tant que variable au langage Javascript pourque puissent les développeurs d'applications l'afficher sur une page comme n'importe quel flux vidéo normal[2].

1.4.2 Peer to peer

L'un des avantages distinctifs du WebRTC est le fait qu'il s'agit d'une communication peer to peer. Cela signifie évidemment que la vidéo passera directement de l'ordinateur d'un utilisateur à un autre[2].

1.4.3 Description de la session

En général, pour qu'une communication ait lieu entre deux périphériques quelconques, il existe un mécanisme de prise de contact qui doit être effectué en premier, pour établir la session entre les deux périphériques, puis le flux des messages ou des paquets peut démarrer[2].

C'est la même chose avec WebRTC. Il existe un protocole de déscription de session : SDP. Il s'agit d'un ensemble de messages qui doivent être échangés entre les deux navigateurs avant qu'ils ne commencent à envoyer des flux multimédias[2].

Pour échanger les données de description de session, nous avons besoin d'une machine intermédiaire. C'est là que le concept de signalisation entre en jeu[2].

1.4.4 Signalisation

Supposons que le navigateur A essaie de communiquer avec le navigateur B. Le navigateur A enverra ses données de description de session au serveur de signalisation S, et le même processus pour le navigateur B également[2].

Une fois ce processus terminé, chaque navigateur a maintenant à sa disposition tout le nécessaire pour établir une connexion et envoyer le flux multimédia à l'autre navigateur[2]. La figure 1.3 aidera à expliquer ce concept.

1.4.4.1 Rôle de la signalisation

La signalisation a un rôle important dans les communications en temps réel :

— Négociation des capacités et des paramètres des médias ntre les appareils dans le même appel

— Identification et authentification des participants à une session

— Contrôler la session multimédia, indiquer la progression, modifier et terminer la session

— Résolution de l'éblouissement, lorsque les deux côtés d'une session essaient d'établir ou de modifier une session en même temps

Pourquoi la signalisation n'a pas été standardisé ?

La signalisation n'est pas normalisée dans WebRTC pour permettre l'interopérabilité entre différents navigateurs. Son flux se situe entre les navigateurs web et le serveur web, et non entre deux navigateurs dans le même appel ou session. Dans ce cas, le serveur ou plutôt le developpeur sélectionne le protocole de signalisation et s'assure que les utilisateurs des applications web ou du support du site utilisent le même protocole[2].

1.4.4.2 Signalisation et négotiation des médias

La fonction la plus importante de la signalisation est l'échange d'informations contenues dans l'objet de protocole de description de session (SDP) entre les navigateurs. SDP contient toutes les informations nécessaires au fonctionnement de la communication, y compris les types de codecs multimédias (audio, vidéo et données) utilisés et des informations sur la bande passante de la connexion[2].

Un autre rôle de la signalisation est l'établissement de connectivité interactive (ICE). Il s'agit de l'adresse candidate représentant l'adresse IP et les ports UDP où des paquets multimédias potentiels pourraient être reçus par le navigateur. Mais il ne peut pas être démarré tant que l'adresse candidate n'a pas été échangée sur le canal de signalisation[2].

Identification et authentification

Le canal de signalisation peut également fournir l'identité des participants à un appel. Par exemple, dans une application web, l'application se chargera d'abord d'authentifier les utilisateurs, et lorsqu'un utilisateur doit appeler un autre utilisateur, l'application web présentera le nom d'utilisateur pour le canal de signalisation comme un identifiant authentifié[2].

WebRTC a d'autres moyens d'authentifier les utilisateurs sans s'appuyer sur des applications externes. En utilisant le canal multimédia, qui ne repose pas sur des applications externes de confiance, une identité et une authentification de l'appelant peuvent être fournies dans le chemin multimédia sans dépendre du tout du canal de signalisation[2].

Contrôle de la session de médias

La signalisation est nécessaire pour lancer l'appel ; cependant, il n'est pas nécessaire d'indiquer l'état ou de mettre fin à une session. Il y a une machine d'état ICE dans le navigateur qui peut fournir ces informations[2].

1.4.4.3 Résolution de l'éclat

La résolution de l'éblouissement intervient lorsque les deux côtés de la communication essaient de changer de session en même temps. C'est une condition de concurrence qui pourrait entraîner un état non déterminé pour la session. Le protocole de signalisation SIP intègre une résolution d'éblouissement[2].

1.5 Les API WebRTC

WebRTC a trois fonctionnalités principales :

1. Acquérir l'audio et la vidéo

2. Communication audio et vidéo
3. Communication de données arbitraires

Et il existe trois API Javascript majeures exposées pour appeler ces foncionnalités :

1. Flux multimédia (getUserMedia)
2. Connexion Peer RTC (RTCPeerConnection)
3. Canal de données RTC (RTCDataChannel)

1.5.1 Flux multimédia

Nous désignons les caméras et les microphones des ordinateurs des utilisateurs sous le nom de média local. Par conséquent, la première étape consiste à obtenir un flux multimédia local. Cela peut être fait de plusieurs manières. Voici un moyen simple de le faire, juste une méthode Javascript que nous pouvons appeler `getUserMedia()`[2]. Cette méthode récupère les médias des périphériques de la machine.

Pour les problèmes de confidentialité, le navigateur doit obtenir l'autorisation de l'utilisateur pour accéder à la caméra et au microphone de l'utilisateur. Le navigateur demandera à l'utilisateur d'autoriser l'accès. Une fois l'accès accordé, l'application recevra un flux de la caméra et du microphone[2].

1.5.2 Connexion peer RTC

Il est important de noter que cette connexion est une connexion peer. Cela signifie que cela peut être fait avec n'importe quel ordinateur ou appareil sur Internet. Et la connexion est directement établie entre les deux machines sans serveur impliqué[2] : les paquets vidéo et audio circuleront d'un navigateur à un autre.

Un fait très important qui mérite d'être mentionné est le nombre de connexions qui seront créées pour chaque utilisateur ajouté à une conférence téléphonique[2].

Disons, par exemple, un appel démarré avec deux utilisateurs, tels que l'utilisateur A et l'utilisateur B. Cela signifie qu'une connexion peer sera créée entre les navigateurs des deux utilisateurs, une connexion de A à B.

Si un autre utilisateur rejoint l'appel, disons l'utilisateur C, cela modifiera les connexions pour chaque utilisateur de l'appel. Comme nous l'avons expliqué précédemment, il s'agira d'une connexion peer, ce qui signifie que chaque utilisateur doit être connecté à la machine de chaque utilisateur lors de l'appel.

Pour comprendre combien de connexions peer seront créées par machine :

$$Connexions\ par\ machine = nombre\ d'utilisateurs\ en\ appel - 1$$
$$Connexions\ par\ machine = 3 - 1$$

Chaque machine aura 2 connexions peer établies si l'appel a 3 utilisateurs.

1.5.3 Canal de données RTC

Les canaux de données prennent en charge les connexions à haut volume et à faible latence ; un canal de données est un canal non multimédia qui prend uniquement en charge le transfert de données[2].

Il contourne les serveurs et fournit aux développeurs web des canaux configurables pour transférer des données. Les canaux de données dans WebRTC sont construits sur des WebSockets. De cette façon, il obtient une fonctionnalité en temps réel. Des méthodes simples telles que l'envoi sur le gestionnaire de messages peuvent être définies pour obtenir les données. Le canal de données utilise également la même connexion peer que le média, ce qui est une bonne chose, car cela signifie qu'un seul processus de négociation offre/réponse est nécessaire[2].

Les canaux de données prennent également en charge les deux modes de livraison : livraison garantie (fiable) et rapide (non fiable). Le premier peut être utilisé pour des événements critiques et le second peut être utilisé pour les mises à jour de la position d'un jeu en ligne[2].

Chapitre 2

Configuration des Interfaces WebRTC

Sommaire

2.1 Introduction

WebRTC offre des opportunités sans précédent aux développeurs qui souhaitent intégrer des communications en temps réel dans leurs applications.

Les API WebRTC getUserMedia, RTCPeerConnection et RTCDataChannel jouent chacune leur propre rôle dans la capture, la transmission et la diffusion de données en temps réel (depuis la webcam et le microphone d'un ordinateur) vers un autre navigateur, sans qu'un utilisateur ait à télécharger des plug-ins ou des modules complémentaires[7].

2.2 RTCPeerConnection

L'API RTCPeerConnection est chargé de connecter deux (02) navigateurs ensemble afin qu'ils puissent partager des médias en temps réel[7]. Elle est au cœur de la connexion peer-to-peer entre chacun des navigateurs.

2.2.1 Configuration et initialisation

2.2.1.1 RTCConfiguration

Chaque connexion peer est gérée par un objet RTCPeerConnection. Le constructeur de cette classe prend un seul objet RTCConfiguration comme paramètre. Cet objet définit la configuration de la connexion peer et doit contenir des informations sur les serveurs ICE à utiliser [1][11].

L'établissement de connectivité interactive (ICE) étant une technique utilisée dans les réseaux informatiques pour trouver des moyens pour que deux ordinateurs communiquent aussi directement que possible dans les réseaux peer-to-peer. Il est le plus souvent utilisé pour les médias interactifs tels que la voix sur IP (VoIP), les communications peer-to-peer, la vidéo et la messagerie instantanée. Dans de telles applications, mieux vaut éviter de communiquer via un serveur central (ce qui ralentirait la communication et serait coûteux), mais la communication directe entre les applications clientes sur Internet est très délicate en raison des traducteurs d'adresses réseau (NAT), des pare-feu et d'autres barrières du réseau[12].

2.2.1.2 Initialisation

Le bout de code suivant tiré de [11] montre comment initialiser l'API RTCPeerConnection, avec un objet RTCConfiguration.

```
const configuration: RTCConfiguration = {
  iceServers: [{'urls': 'stun:stun.l.google.com:19302'}],
```

1. Le dictionnaire complet des membres de RTCConfiguration peut être accédée à l'addresse https ://developer.mozilla.org/en-US/docs/Web/API/RTCConfiguration

```
};
const peerConnection = new RTCPeerConnection(configuration);
```

2.2.2 RTCSessionDescription

RTCPeerConnection coordonne l'échange de métadonnées cruciales entre deux navigateurs. Ces données définissent le numéro IP et l'adresse de port publiquement identifiables d'un navigateur afin que les médias en temps réel puissent être échangés.

Pour que deux points de terminaison WebRTC commencent à se parler, trois types d'informations doivent être relayées :

1. Les informations de contrôle de session déterminent quand initialiser, fermer et modifier les sessions de communication.
2. Les données réseau relayent l'adresse IP et le numéro de port de chaque point de terminaison afin que les appelants puissent trouver les personnes appelées.
3. Les données média concernent les codecs et les types de média que les appelants ont en commun.

Ces informations seront encapsulées dans un objet RTCSessionDescription.

L'interface RTCSessionDescription décrit une extrémité d'une connexion (ou connexion potentielle) et comment elle est configurée. Chaque objet RTCSessionDescription consiste en un type de description indiquant quelle partie du processus de négociation il décrit (offre ou réponse) et du descripteur SDP de la session[6].

Le processus de négociation d'une connexion entre deux peers implique l'échange d'objets RTCSessionDescription dans les deux sens, par le biais du canal de signalisation(1.4.4.2), chaque description suggérant une combinaison d'options de configuration de connexion que l'expéditeur de la description prend en charge. Une fois que les deux pairs se sont mis d'accord sur une configuration pour la connexion, la négociation est terminée[6].

2.2.2.1 Signalisation

La spécification WebRTC inclut des API pour communiquer avec un serveur ICE , mais le composant de signalisation n'en fait pas partie(1.4.4.1). La signalisation est nécessaire pour que deux pairs partagent la façon dont ils doivent se connecter. Habituellement, cela est résolu via une API Web standard basée sur HTTP (c'est-à-dire un service REST ou un autre mécanisme RPC) où les applications Web peuvent relayer les informations nécessaires avant que la connexion entre homologues ne soit lancée[11].

L'extrait de code provenant de [11] suivant, montre comment un service de signalisation fictif peut être utilisé pour envoyer et recevoir des messages de manière asynchrone.

```
// Configurer un canal de communication asynchrone qui
// sera utilisé lors de la configuration de la connexion peer
const signalingChannel = new SignalingChannel(remoteClientId);
signalingChannel.addEventListener('message', message => {
  // Nouveau message du client distant reçu
});

// Envoyer un message asynchrone au client distant
signalingChannel.send('Hello!');
```

2.2.2.2 Initiation de la description de session

Voici comment une desscription de session sera initiée en tant qu'une offre puis envoyée de la part du client local vers un client distant, sans oublier comment ce même client négociera une potentielle réponse de son correspondant[11].

```
// Écouter la réception d'un message sur le canal de signalisation
signalingChannel.addEventListener('message', async message => {
  if (message.answer) { // Si le message comporte une réponse
    // assigner la réponse en tant que description de session distante
    const remoteDesc = new RTCSessionDescription(message.answer);
    await peerConnection.setRemoteDescription(remoteDesc);
  }
});

// Créer une description de session associée à l'offre
const offer = await peerConnection.createOffer();
await peerConnection.setLocalDescription(offer);
// Puis envoyer cette offre au client distant
signalingChannel.send({'offer': offer});
```

Et ci-après, comment du côté d'un client distant auquel sera demandé une communication, la négiociation d'une offre qui lui aboutit sera faite[11].

```
// Écouter la réception d'un message sur le canal de signalisation
signalingChannel.addEventListener('message', async message => {
  if (message.offer) { // Si le message contient une offre
  // assigner l'offre en tant que description de session distante
```

```
    const remoteDesc = new RTCSessionDescription(message.offer);
    peerConnection.setRemoteDescription(remoteDesc);

    // Puis créer une description de session associée à la réponse
    const answer = await peerConnection.createAnswer();
    await peerConnection.setLocalDescription(answer);

  // Efin renvoyer cette réponse au client ayant envoyé l'offre
    signalingChannel.send({'answer': answer});
  }
});
```

2.3 Capture des périphériques médias

L'API getUserMedia permet d'accéder aux flux multimédias (vidéo, audio ou les deux) à partir d'appareils locaux. À elle seule, cette API est uniquement capable d'acquérir de l'audio et de la vidéo, sans envoyer les données ni les stocker dans un fichier. Pour avoir un chat fonctionnel, nous devons envoyer ces données en utilisant l'API RTCPeerConnection.

À elle seule, cette API est uniquement capable d'acquérir de l'audio et de la vidéo, sans envoyer les données ni les stocker dans un fichier. Pour avoir un chat fonctionnel, nous devons envoyer ces données en utilisant l'API RTCPeerConnection[10].

2.3.1 configuration de getUserMedia

Le code permettant d'obtenir un flux à partir des périphéques médias dans le navigateur s'écrit comme dans l'éxemple suivant :

```
const constraints: MediaStreamConstraint = { audio: true, video: true };
navigator.mediaDevices.getUserMedia(constraints);
```

L'interface Navigator représente l'état et l'identité de l'agent utilisateur. Il permet aux scripts de l'interroger et de s'enregistrer pour effectuer certaines activités. Un objet Navigator peut être récupéré à l'aide de la propriété readonly `window.navigator`[5].

L'interface Navigator dans le navigateur fournit des fonctions et des propriétés pour accéder à l'état et aux fonctionnalités du navigateur grâce auxquelles nous pouvons obtenir l'état du navigateur ou accéder à un large éventail de fonctionnalités. Par exemple, en utilisant `navigator.online`, nous pouvons obtenir l'état de connexion du navigateur avec Internet[9].

L'interface MediaDevices permet d'accéder aux périphériques d'entrée multimédia connectés tels que les caméras et les microphones, ainsi que le partage d'écran. En substance, il vous permet d'accéder à n'importe quelle source matérielle de données multimédias[5].

Les fonctions de l'objet mediaDevices fournissent des fonctionnalités telles que le partage d'écran, l'obtention de flux depuis la caméra et les microphones. La fonction getUserMedia est celle dont nous avons besoin pour récupérer les flux de la caméra et des microphones[9].

La méthode `getUsermedia()` à besoin qu'on lui passe un objet MediaStreamConstraints pour qu'uelle fonctionne. Cet argument, active le type de communication voulu (vidéo, audio ou les deux) et contient les contraintes ainsi que les paramètres du flux de média qui sera généré par la suite. La suppression de bruit, l' annulation de l'écho, le ratio de la video, le volume du son, la luminosité de l'image, etc sont quelques éxemples de ces paramètres en question [2].

Le dictionnaire MediaStreamConstraints est utilisé lors de l'appel à getUsermedia pour spécifier les types de pistes à inclure dans le MediaStream renvoyé et, éventuellement, pour établir des contraintes pour les paramètres de ces pistes[4].

Tout comme dans l'exemple précédent, l'implémentation minimale requise par l'interface MediaStreamConstraints est lasuivante : `{ video: true, audio: true }`.

Ce dernier, indique que nous voulons que le flux généré comporte une voix et une vidéo en utilisant les configurations de contraintes et paramètres par défaut.

2. Une liste complète des contraintes et paramètres de flux de média peut être consultée à l'addresse url : https ://developer.mozilla.org/en-US/docs/Web/API/MediaTrackConstraints

Chapitre 3

Relevé des protocoles mis en jeux dans WebRTC

Sommaire

3.1 Introduction

Références

[1] H.W. BARZ et G.A. BASSETT. *Multimedia Networks : Protocols, Design and Applications*. Anglais. 1re éd. Wiley, 2016.

[2] A. HAYTHAM. "Real-time Communication Using WebRTC". Anglais. In : *Special Interest Group on Computer-Human Interaction* (2018).

[3] A.B. JOHNSTON et D.C. BURNETT. *WebRTC : APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. Anglais. 2e éd. Smashwords Edition, 2013. ISBN : 978-0-9859788-5-3.

[4] MOZILLA ORG. *MediaTrackConstraints - Web APIs*. Américain. 14 juin 2021. URL : https://developer.mozilla.org/en-US/docs/Web/API/MediaTrackConstraints.

[5] MOZILLA ORG. *Navigator - Web APIs*. Anglais. 15 juin 2021. URL : https://developer.mozilla.org/en-US/docs/Web/API/Navigator (visité le 16/06/2021).

[6] MOZILLA ORG. *RTCSessionDescription - Web APIs*. Anglais. 31 mai 2021. URL : https://developer.mozilla.org/en-US/docs/Web/API/RTCSessionDescription (visité le 16/06/2021).

[7] ONSIP. *RTCPeerConnection - WebRTC Explained*. Anglais. 2021. URL : https://www.onsip.com/voip-resources/voip-fundamentals/rtcpeerconnection (visité le 16/06/2021).

[8] K.G. PATHARE et P.M. CHOURAGADE. "WebRTC Implementation and Architecture Analysis". Anglais. In : *International Journal of Scientific & Engineering Research* 7 (2016).

[9] S. RAJAN. *getUserMedia API - An introduction*. Anglais. 16 jan. 2021. URL : https://www.thegeeksclan.com/getusermedia-api-an-introduction/ (visité le 18/06/2021).

[10] A. ROSA. *An Introduction to the getUserMedia API*. Anglais. 1er jan. 2014. URL : https://www.sitepoint.com/introduction-getusermedia-api/ (visité le 18/06/2021).

[11] WEBRTC ORG. *Getting started with peer connections*. Anglais. 2014. URL : https://webrtc.org/getting-started/peer-connections (visité le 16/06/2021).

[12] WIKIPEDIA CONTRIBUTORS. *Interactive Connectivity Establishment*. Anglais. 23 août 2020. URL : https://en.wikipedia.org/wiki/Interactive_Connectivity_Establishment (visité le 16/06/2021).

Annexes

A Exemple test de code Javascript dans LaTeX

A.1 Premier snippet

```
(function(){
  var user = getCookie("username");
  document.getElementById("#date-field").innerHTML = new Date();
  document.getElementById("#greetings").innerHTML = "<p>Hello, " + user.name +
    ".</p>";
})()

function getCookie(cname) {
  var name = cname + "=";
  var decodedCookie = decodeURIComponent(document.cookie);
  var ca = decodedCookie.split(';');
  for(var i = 0; i < ca.length; ++i) {
    var c = ca[i];
    while (c.charAt(0) == ' ') {
      c = c.substring(1);
    }
    if (c.indexOf(name) == 0) {
      return c.substring(name.length, c.length);
    }
  }
  return "";
}
```

A.2 Second snippet

```
/* eslint-env es6 */
/* eslint-disable no-unused-vars */
import Axios from 'axios'
import { BASE_URL } from './utils/api'
import { getAPIToken } from './utils/helpers'

export default class User {
  constructor () {
    this.id = null
    this.username = null
    this.email = ''
    this.isActive = false
    this.lastLogin = ''  // ISO 8601 formatted timestamp.
    this.lastPWChange = ''  // ISO 8601 formatted timestamp.
  }
}

const getUserProfile = async (id) => {
  let user = new User()
  await Axios.get(
    `${BASE_URL}/users/${id}`,
    {
      headers: {
        'Authorization': `Token ${getAPIToken()}`,
      }
    }
  ).then{response => {
    // ...
  }).catch(error => {
    // ...
  }) }
```