Chapitre 1

Configuration des Interfaces WebRTC

1.1 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.[**?**]

1.2 RTCPeerConnection

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

1.2.1 Configuration et initialisation

1.2.1.1 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].[**?**]

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.[**?**]

1.2.1.2 Initialisation

Le bout de code suivant tiré de [**?**] 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);
```

1.2.2 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.[**?**]

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(**??**), 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.[**?**]

1.2.2.1 Signalisation

La spécification WebRTC inclut des API pour communiquer avec un serveur ICE , mais le composant de signalisation n'en fait pas partie(**??**). 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.[**?**]

L'extrait de code provenant de [**?**] 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!');
```

1.2.2.2 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.[**?**]

```
// 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.[**?**]

```
// 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});
  }
});
```