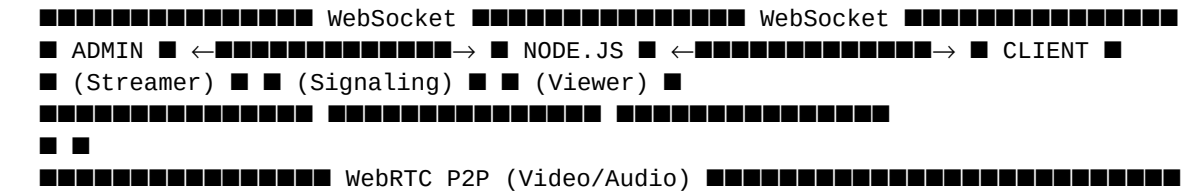
■ Résumé du projet Live Streaming WebRTC

■■ Architecture globale

Le projet utilise une architecture peer-to-peer avec un serveur de signalisation pour établir les connexions directes entre l'admin (streamer) et les clients (viewers).



■ Composants du système

1. Serveur Node.js (Signaling Server)

Port : 9090
Rôle : Faciliter l'échange de messages entre streamers et viewers pour établir les connexions WebRTC.
Fonctionnalités :
- Gère les connexions WebSocket des clients
- Maintient une liste des streamers et viewers connectés
- Route les messages de signalisation (offers, answers, ICE candidates)
- Nettoie automatiquement les connexions fermées

```
Messages gérés :

// Connexion d'un streamer { type: 'streamer' } // Connexion d'un viewer { type:
'viewer', viewerId: 'user_id' } // Notification nouveau viewer { type: 'newViewer',
viewerId: 'user_id' } // Négociation WebRTC { type: 'offer', offer: {...}, viewerId:
'user_id' } { type: 'answer', answer: {...}, viewerId: 'user_id' } { type:
'candidate', candidate: {...}, target: 'viewer/streamer', viewerId: 'user_id' }
```

2. Interface Admin (Streamer)

Framework : Symfony/Twig
Rôle : Diffuser le flux vidéo/audio vers les clients connectés.
Processus de diffusion :

```
1. Initialisation WebSocket

const signalingServer = new WebSocket('ws://localhost:9090');
signalingServer.send(JSON.stringify({ type: 'streamer' }));

2. Capture média

navigator.mediaDevices.getUserMedia({ video: true, audio: true })
```

3. Gestion des nouveaux viewers :
- Reçoit { type: 'newViewer', viewerId: 'X' }
- Crée un RTCPeerConnection pour ce viewer
- Ajoute les tracks audio/vidéo à la connexion
- Génère et envoie une offer WebRTC

4. Négociation WebRTC
Échange des ICE candidates pour établir la connexion P2P, traite les answers des viewers, maintient une connexion directe avec chaque viewer.

```
État des connexions :

const peerConnections = new Map(); // viewerId => RTCPeerConnection
```

3. Interface Client (Viewer)

Framework : Symfony/Twig
Rôle : Recevoir et afficher le flux vidéo/audio du streamer.
Processus de réception :

```
1. Connexion au serveur
```

```
signalingServer.send(JSON.stringify({ type: 'viewer', viewerId: userId }));
```

2. Réception de l'offre et création de la PeerConnection

```
3. Gestion des tracks
```

```
peerConnection.ontrack = event => { video.srcObject = event.streams[0]; };
```

4. Envoi de la réponse
Génère un answer, échange les ICE candidates, établit la connexion directe avec le streamer.
Gestion de l'autoplay :
- Tentative d'autoplay automatique (avec muted)
- Bouton de secours si l'autoplay est bloqué par le navigateur

■ Flux de données complet

Étape 1 : Démarrage du live
Admin démarre le live → Obtient accès caméra → Se connecte au serveur WebSocket
Étape 2 : Connexion d'un viewer
Client se connecte → Envoie son ID au serveur → Serveur notifie l'admin
Étape 3 : Négociation WebRTC
1. Admin crée une PeerConnection pour ce viewer 2. Admin ajoute ses tracks audio/vidéo 3. Admin génère une "offer" SDP 4. Serveur route l'offer vers le viewer 5. Viewer crée sa PeerConnection 6. Viewer génère une "answer" SDP 7. Serveur route l'answer vers l'admin 8. Échange des ICE candidates via le serveur 9. Connexion P2P établie directement
Étape 4 : Streaming
Admin → (flux direct WebRTC) → Client

■ Technologies utilisées

| Composant | Technologies | |---------------|---------------------------| | Backend | PHP/Symfony, Twig | | Signaling | Node.js, WebSocket | | Streaming | WebRTC (RTCPeerConnection)| | Media | getUserMedia API | | Transport | UDP/DTLS (WebRTC) |

■ Points clés de sécurité

- STUN servers : Utilisés pour traverser les NAT/firewalls
- ICE candidates : Négocient la meilleure route de connexion
- DTLS : Chiffrement automatique des flux WebRTC
- Origin policies : Contrôle d'accès aux médias

■ Avantages de cette architecture

■ Scalabilité : Connexions P2P directes (pas de relais serveur)
■ Faible latence : Pas d'intermédiaire pour le streaming
■ Économique : Le serveur ne traite que la signalisation
■ Qualité : Pas de re-encodage côté serveur
■ Robustesse : Reconnexion automatique en cas de problème

■ Flux d'utilisation

Admin :
1. Se connecte à l'interface admin
2. Clique sur "Démarrer un Live"
3. Autorise l'accès à la caméra/micro
4. Le stream est diffusé automatiquement aux viewers
Client :
1. Se connecte à l'interface de visionnage
2. La connexion s'établit automatiquement
3. Clique sur le bouton play si nécessaire
4. Regarde le live en temps réel

■ Monitoring possible

- État des connexions WebRTC (connectionState)
- Qualité des flux (getStats API)
- Nombre de viewers connectés
- Bande passante utilisée
- Latence des connexions