Plateforme sécurisée de collaboration multimédia pour les équipes de recherche scientifique

Présenté par :

HOTANY OUMNIA
LAAMIR DOUNIA
OUBDA DIMITRI-GAETAN
SANOGO CHEICKNA
YAHYAOUI ABDERRAHMANE

Encadré par:

Mme WISSAM ABBASS

Introduction Générale

Les plateformes de collaboration multimédia (Microsoft Teams, Zoom, Slack, Webex, etc.) sont largement utilisées pour faciliter la communication et le partage d'informations dans les entreprises et les organisations. Cependant, elles présentent des vulnérabilités exploitables par des attaquants. Il est essentiel de connaître les menaces et de mettre en place des solutions de sécurité adaptées.

Menaces de Sécurité

Interception et espionnage

- Attaques de type "Man-in-the-Middle" (MITM) interceptant les communications.
- Ecoute clandestine des réunions non sécurisées.

Usurpation d'identité et hameçonnage

- Attaques par phishing visant à voler des identifiants d'accès.
- Usurpation d'identité pour accéder à des discussions sensibles.

Vulnérabilités logicielles

- Exploitation de failles dans les applications de visioconférence.
- Injection de code malveillant via des liens ou fichiers partagés.

Fuites de données

- Partage involontaire ou malveillant d'informations sensibles.
- Sauvegarde non sécurisée des vidéos et fichiers dans le cloud.

Solutions de Sécurité

Authentification et gestion des accès

Chiffrement des communications

Sensibilisation et formation des utilisateurs

Surveillance et réponse aux incidents

Contrôle des accès et gestion des permissions

LE RASPBERRY PI

Le Raspberry Pi 4 est un nano-ordinateur monocarte conçu pour offrir des performances améliorées par rapport aux versions précédentes. Il est idéal pour l'informatique embarquée, l'apprentissage du codage, les serveurs domestiques et les projets IoT.
C'est le système embarqué utilisé pour hébérger notre plateforme.

Présentation de notre plateforme : Collab

Authentification des utilisateurs

Dépendances utilisées

1. **DJANGO**
   Un framework Python qui facilite le développement d'applications web sécurisées et maintenables. Il inclut des fonctionnalités intégrées pour la gestion de l'authentification, des sessions utilisateurs, et d'autres aspects essentiels du développement web.

2. **PYOTP**
   Une bibliothèque Python qui permet de générer des codes d'authentification à usage unique basés sur le temps (TOTP). Cela est couramment utilisé pour mettre en œuvre l'authentification à deux facteurs (2FA) dans les applications.

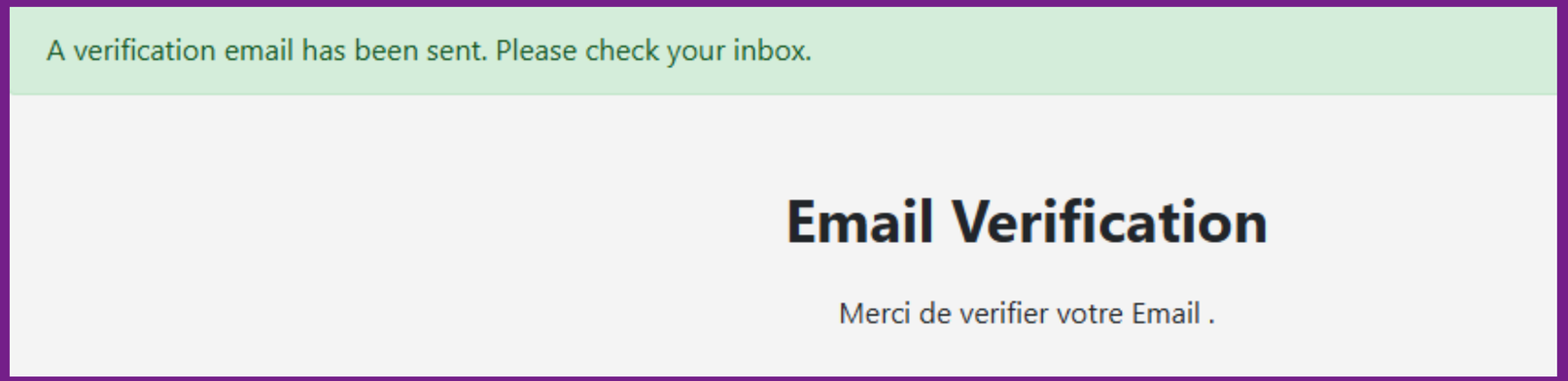
3. **QRCODE**
   Une bibliothèque Python pour générer des codes QR.

4. **PILLOW**
   Une bibliothèque de traitement d'images en Python qui permet de créer et de manipuler des images. Souvent utilisée pour générer des images de codes QR

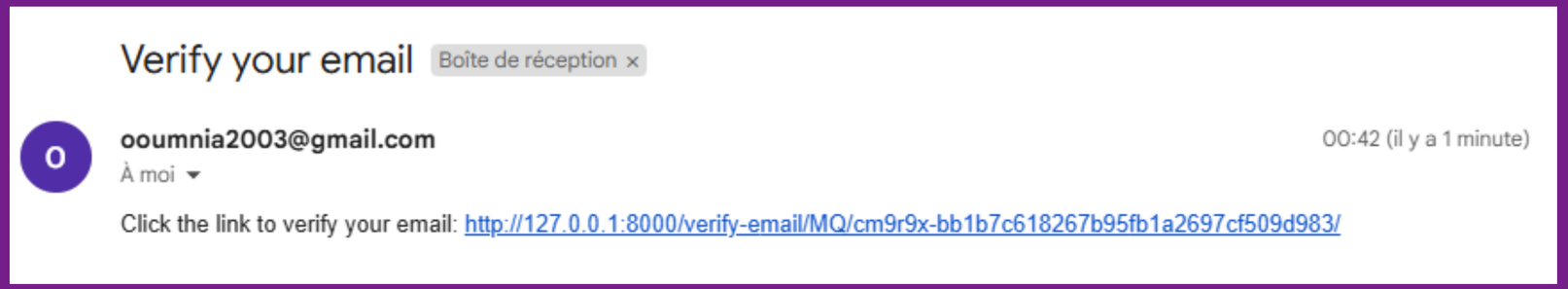
5. **SMTPLIB**
   Une bibliothèque Python permettant d'envoyer des e-mails via le protocole SMTP. Elle est souvent utilisée pour l'envoi de messages, comme les e-mails de vérification d'adresse, en utilisant des services tels que Gmail.

Processus d'authentification

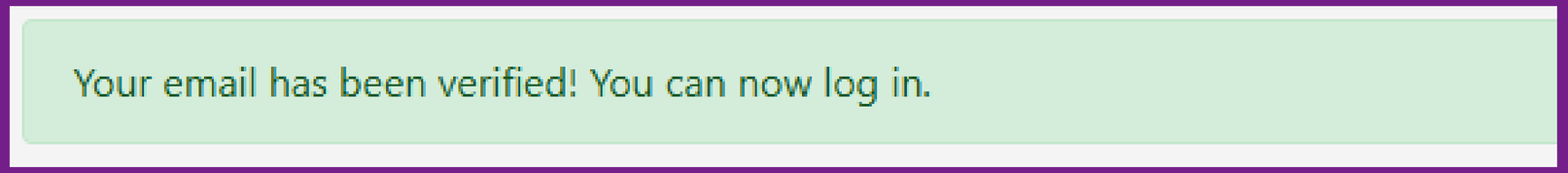
01 VÉRIFICATION DE L'E-MAIL



Après l'inscription, un e-mail de confirmation est envoyé à l'adresse fournie par l'utilisateur.

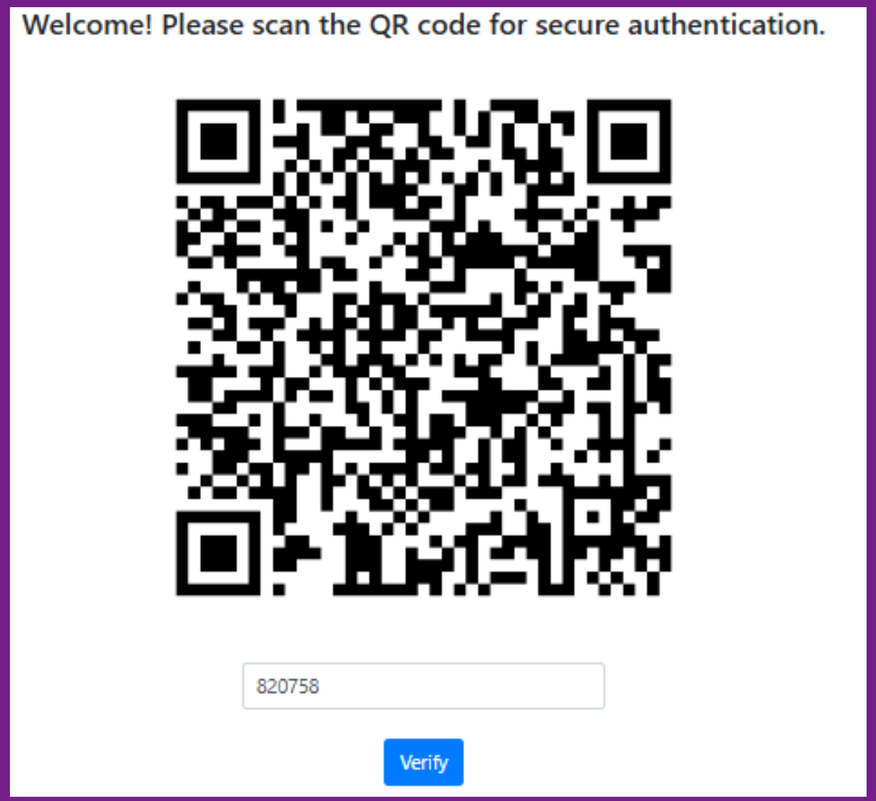


Le envoyé valide l'adresse e-mail de l'utilisateur et l'autorise à poursuivre le processus d'activation de son compte.



Processus d'authentification

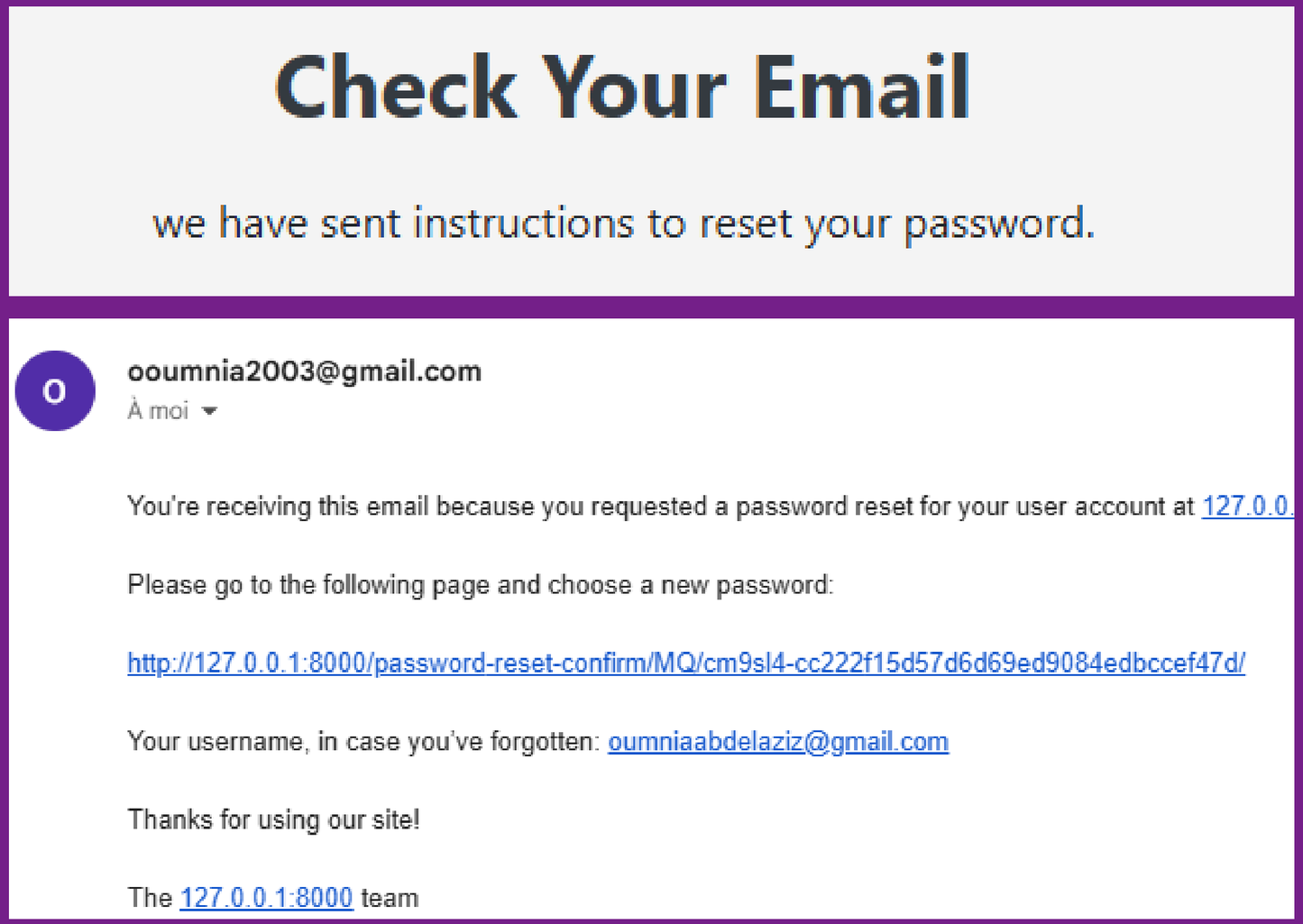
02 ACTIVATION DU MFA VIA QR CODE



Lors de sa première connexion, l'utilisateur est dirigé vers une page sur laquelle un QR Code est généré.
L'utilisateur doit scanner ce QR Code avec une application d'authentification comme Google Authenticator.
Enfin, il est obligé de saisir le code OTP génèré .

Processus d'authentification

03 MOT DE PASSE OUBLIÉ

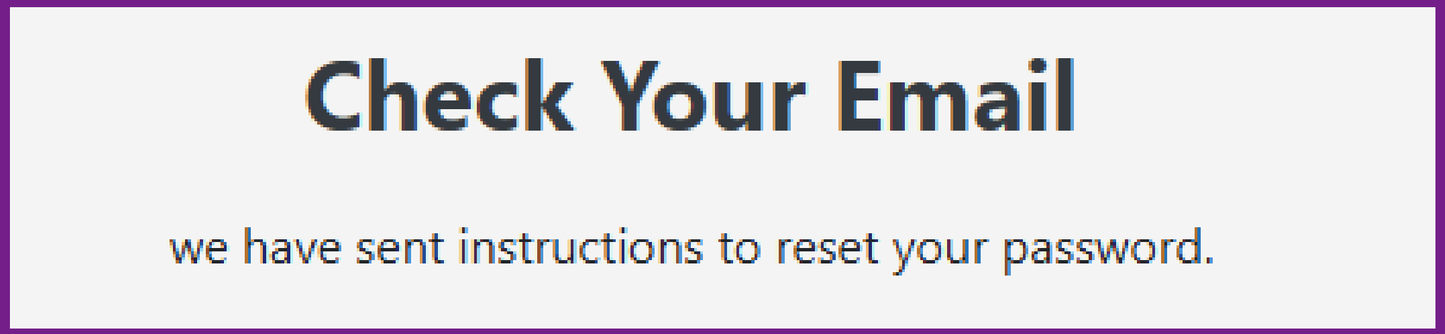


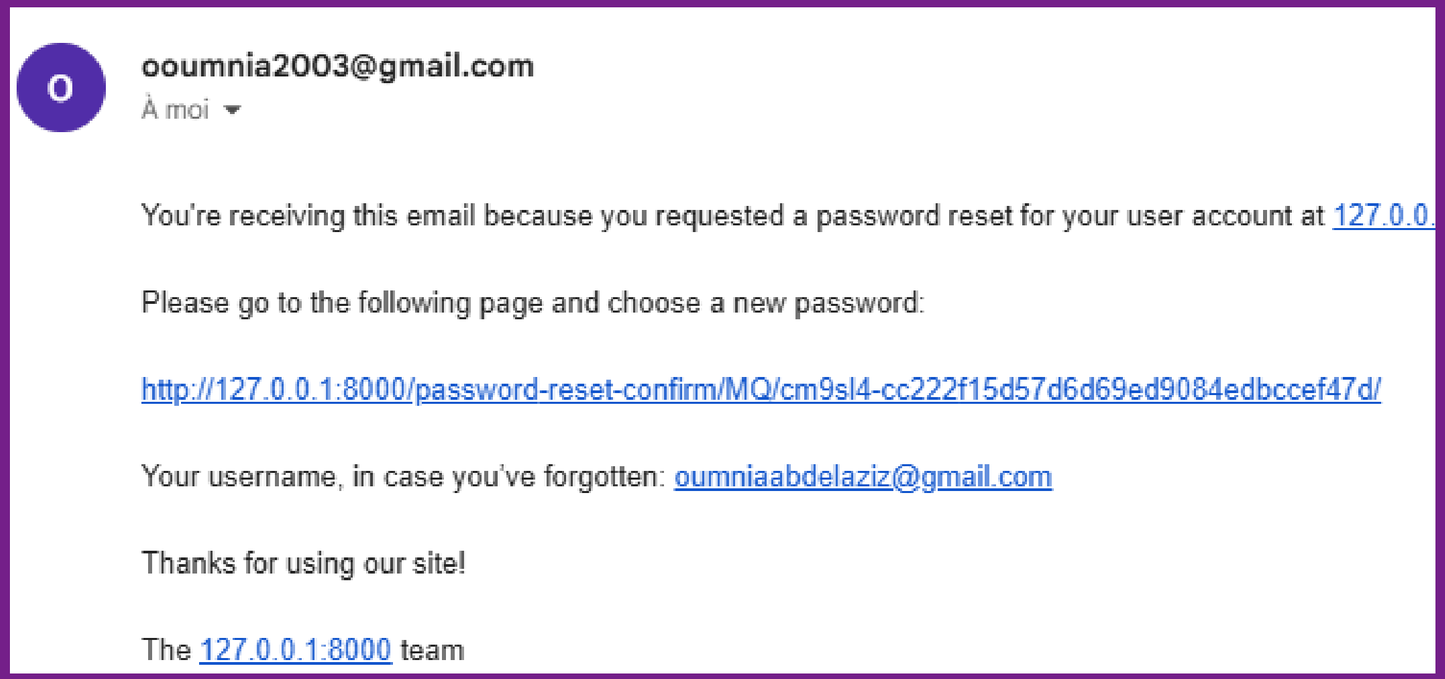
Si l'utilisateur a oublié son mot de passe, un lien de réinitialisation lui est envoyé. En cliquant sur ce lien, il sera redirigé vers une page où il pourra changer son mot de passe.

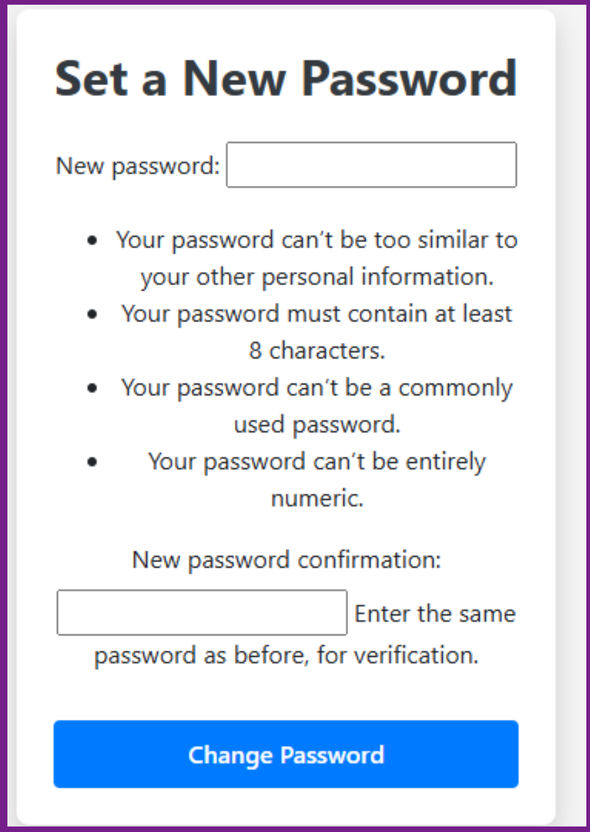
Processus d'authentification

03 MOT DE PASSE OUBLIÉ

Si l'utilisateur a oublié son mot de passe, un lien de réinitialisation lui est envoyé. En cliquant sur ce lien, il sera redirigé vers une page où il pourra changer son mot de passe.







Mesures de Sécurité Additionnelles

1. **EXPIRATION DES LIENS DE VALIDATION D’E-MAIL**
2. **PROTECTION CSRF ET XSS**
3. **BLOCAGE TEMPORAIRE EN CAS D’ÉCHECS RÉPÉTÉS**

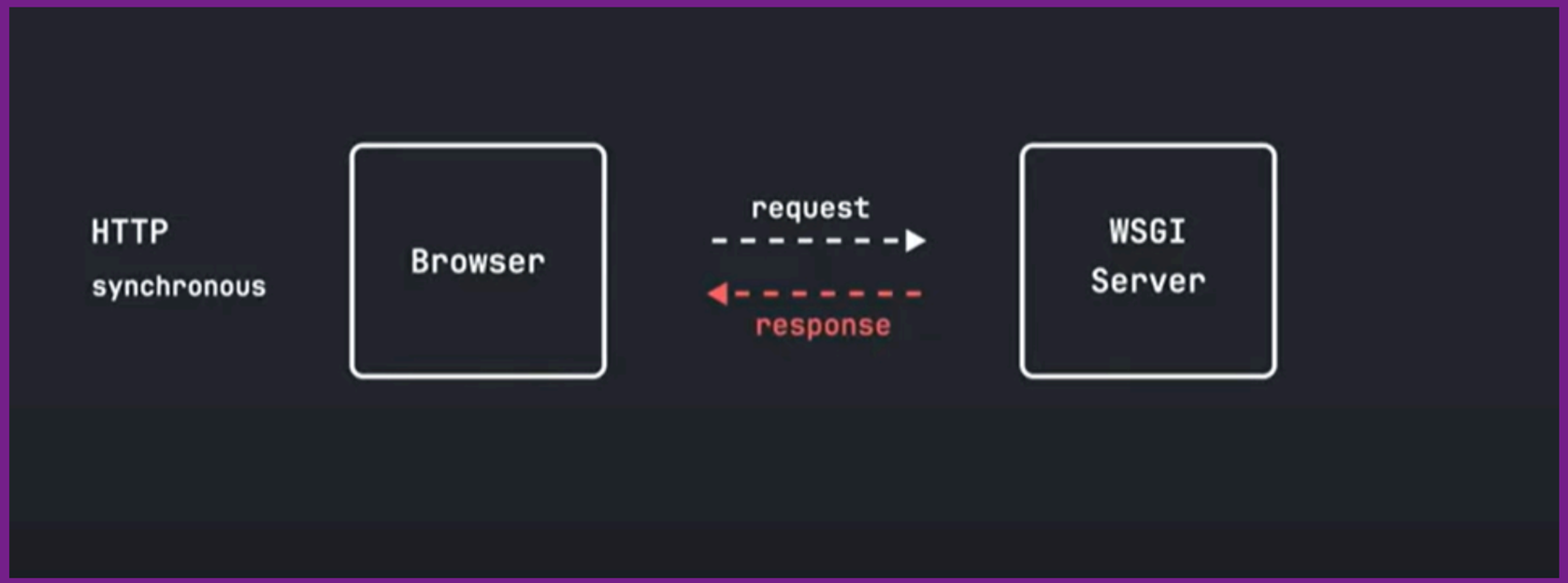
Salons d’échanges

Technologies utilisées

1. **DJANGO CHANNEL** — Extension de Django pour gérer les WebSockets.
2. **REDIS** — Base de données en mémoire utilisée comme broker pour Django Channels.
3. **WEBSOCKETS** — Protocole permettant la communication en temps réel entre le serveur et les clients.
4. **DAPHNE** — Serveur ASGI permettant de gérer les connexions WebSocket.
5. **HTMX/REACT** — Technologies utilisées pour mise à niveau et l'interface utilisateur du chat.

Fonctionnement

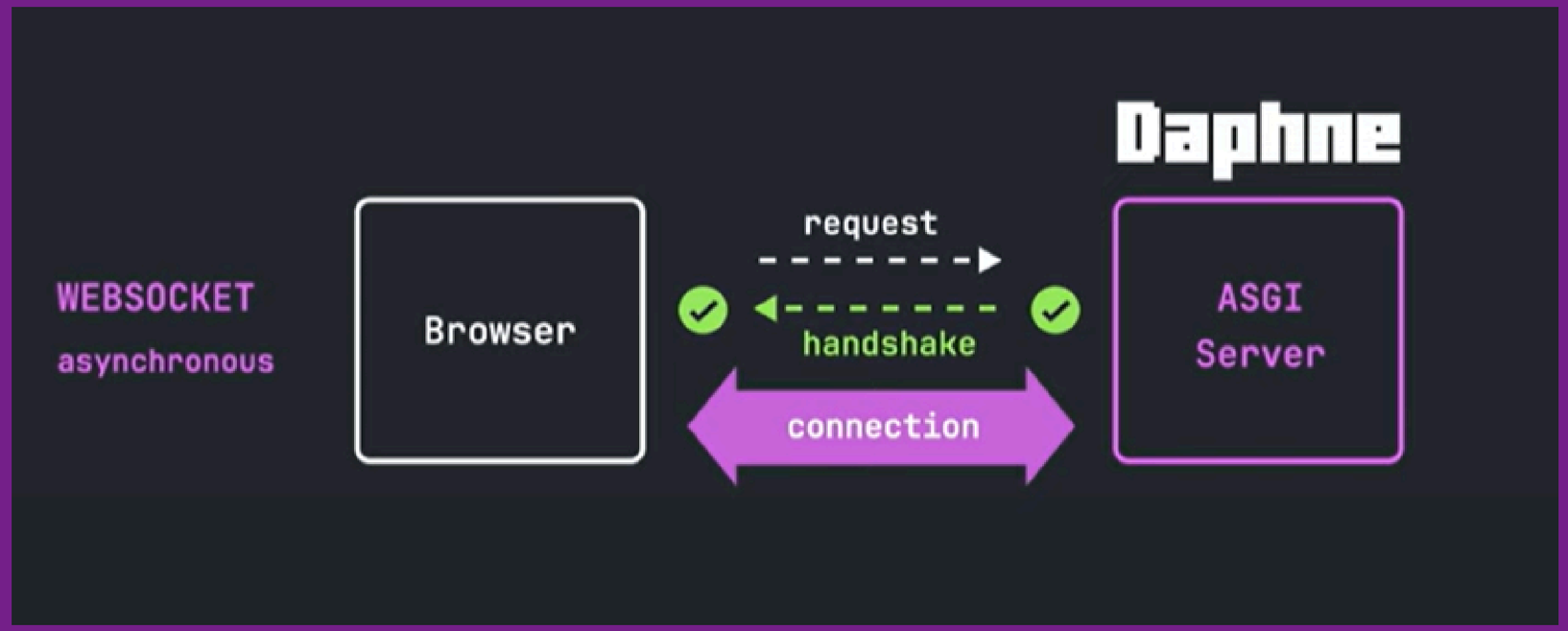
01 WEBSOCKETS VS HTTP (CLASSIQUE)



- Serveur WSGI
- Communication synchrone
- Connexion fermée après la réponse

Fonctionnement

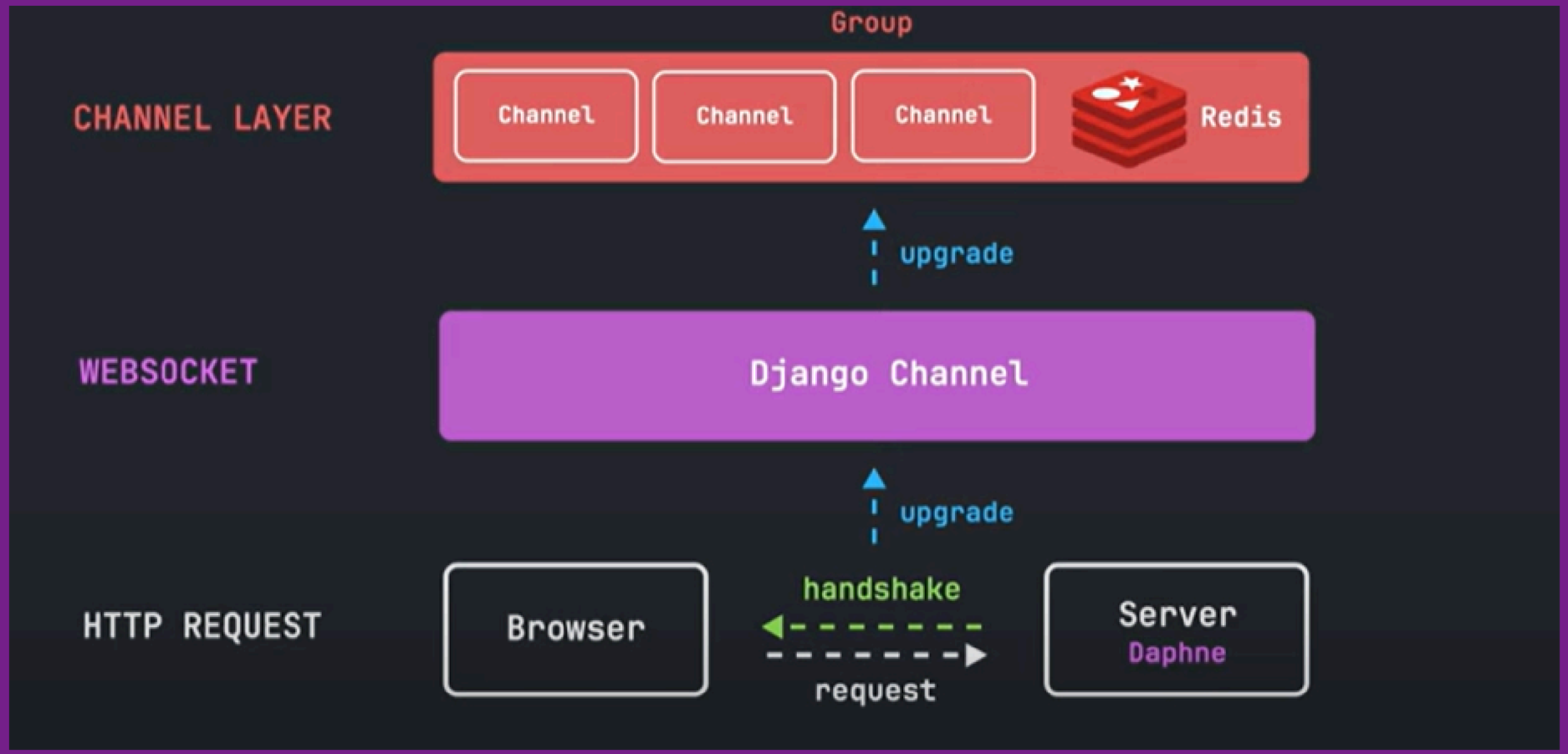
01 WEBSOCKETS VS HTTP (CLASSIQUE)



- Serveur ASGI (Daphne)
- Communication asynchrone
- Connexion reste ouverte

Fonctionnement

02 FONCTIONNEMENT GLOBAL



Mesures de sécurité

01 CHIFFREMENT DE BOUT EN BOUT (E2EE) AVEC AES

L'AES (Advanced Encryption Standard) est l'un des algorithmes les plus sécurisés et efficaces. Il est utilisé par WhatsApp et Signal.

AES-256 (mode GCM)

- Sécurité élevée
- Chiffrement rapide avec support matériel sur de nombreux appareils
- Utilisation possible même dans des environnements embarqués

Mesures de sécurité

02 ECHANGE DE CLES AVEC ECDH

L'ELLIPTIC CURVE DIFFIE-HELLMAN (ECDH) EST UNE VARIANTE PLUS RAPIDE ET SÉCURISÉE DU PROTOCOLE CLASSIQUE DIFFIE-HELLMAN (DH).

Avantages:

1. Sécurisé
2. Rapide et efficace
3. Parfait pour les systemes embarques.
4. Résistant aux attaques quantiques (pas totalement)

Mesures de sécurité

02 ECHANGE DE CLES AVEC ECDH

Principe :

1. Chaque utilisateur génère une clé publique et une clé privée.
2. Ils échangent leurs clés publiques.
3. Ils génèrent une clé de session commune en combinant leur clé privée et la clé publique de l'autre utilisateur.
4. Cette clé est ensuite utilisée pour chiffrer les messages avec AES.

Mesures de sécurité

03 SIGNATURE DES MESSAGES POUR ÉVITER LES ATTAQUES MITM (ED25512)

ED25519 EST UNE MÉTHODE DE SIGNATURE NUMÉRIQUE BASÉE SUR LA CRYPTOGRAPHIE À COURBES ELLIPTIQUES, PLUS PRÉCISÉMENT SUR CURVE25519

Avantages:

1. Sécurisé
2. Rapide et efficace
3. Petites clés, grande sécurité
4. Parfait pour les systemes embarques.
5. Facilité d’utilisation

Mesures de Sécurité et fonctionnalités Additionnelles

1. **UN SYSTÈME DE DÉTECTION D’ACTIVITÉS SUSPECTES ET DE PRÉVENTION DES ATTAQUES.**
2. **UN MÉCANISME DE SUPPRESSION AUTOMATIQUE DES MESSAGES APRÈS UN CERTAIN TEMPS.**
3. **UNE MEILLEURE GESTION DES PERMISSIONS POUR LES ADMINISTRATEURS DE ROOMS.**

Appel vidéo

Technologies utilisées

1. **WEB-SOCKETS**

Fournit un canal de communication bidirectionnel persistant entre le client et le serveur. Dans ce script, les WebSockets sont utilisés pour la signalisation (échange de messages de configuration d'appel, de candidats ICE et de notifications de fin d'appel).

2. **WEBRTC**

Permet la communication peer-to-peer en temps réel pour l'audio et la vidéo. Il est chargé d'établir la connexion directe entre les utilisateurs à l'aide de composants tels que RTCPeerConnection.

3. **GETUSER MEDIA API**

Une API de navigateur qui demande l'accès à la caméra et au microphone de l'utilisateur. Cette API est utilisée pour capturer le flux multimédia local qui est ensuite transmis via la connexion WebRTC.

4. **DAPHNE**

Daphne est un serveur ASGI (Asynchronous Server Gateway Interface) utilisé pour servir les applications Django Channels. Il permet de gérer les WebSockets, les interrogations longues et d'autres protocoles asynchrones, ce qui le rend essentiel pour les applications en temps réel comme les appels vidéo.

Processus d’appel

01 SÉCURITÉ AVANT TOUT

Avant que index.html puisse fonctionner correctement, Daphne doit servir l'application via HTTPS/WSS pour autoriser les connexions getUserMedia() et WebRTC.

```
>daphne -e ssl:8443:privateKey=key.pem:certKey=cert.pem videocall.asgi:application -p 8000 -b 192.168.0.188
```

Pourquoi HTTPS/WSS ?

- WebRTC (getUserMedia()) nécessite HTTPS pour la sécurité.
- Les WebSockets (ws:// ou wss://) nécessitent HTTPS pour une signalisation sécurisée.
- Permet la communication entre homologues sur différents réseaux (avec STUN/TURN).

Processus d’appel

02 OUVERTURE DE CONNEXION WEBSOCKET

```
function connectSocket() {
  let ws_scheme = window.location.protocol == "https:" ? "wss://" : "ws://";
  console.log(ws_scheme);

  callSocket = new WebSocket(ws_scheme + window.location.host + '/ws/call/');

  callSocket.onopen = event => {
    callSocket.send(JSON.stringify({
      type: 'login',
      data: { name: myName }
    }));
  }
```

- Détermine le schéma WebSocket : Utilise wss:// si HTTPS est activé, ws:// sinon.
- Ouvre une connexion WebSocket à /ws/call/.
- Envoie un message de « connexion » pour enregistrer l'utilisateur

Processus d’appel

03 INITIALISATION DE L’APPEL

```
function call() {
  let userToCall = document.getElementById("callName").value;
  otherUser = userToCall;
  beReady().then(() => {
    processCall(userToCall)
  });
}
```

- Obtient le nom d'utilisateur à appeler (userToCall).
- Obtient l'accès à la caméra/au microphone à l'aide de beReady().
- Crée une offre WebRTC et l'envoie via WebSocket (processCall(userToCall)) → signale le destinataire.

Processus d’appel

04 ETABLISSEMENT DE LA CONNEXION WEBRTC

```
function beReady() {
  return navigator.mediaDevices.getUserMedia({
      audio: true,
      video: true
    })
    .then(stream => {
      localStream = stream;
      localVideo.srcObject = stream;
      return createConnectionAndAddStream()
    })
    .catch(function (e) {
      alert('getUserMedia() error: ' + e.name);
    });
}
```

- Appelle navigator.mediaDevices.getUserMedia() pour demander la caméra et le microphone.
- Appelle createConnectionAndAddStream() pour configurer la connexion WebRTC.

Processus d’appel

04 ETABLISSEMENT DE LA CONNEXION WEBRTC

```
function createConnectionAndAddStream() {
  createPeerConnection();
  peerConnection.addStream(localStream);
  return true;
}
```

```
function createPeerConnection() {
  try {
    peerConnection = new RTCPeerConnection(pcConfig);
    peerConnection.onicecandidate = handleIceCandidate;
    peerConnection.onaddstream = handleRemoteStreamAdded;
    peerConnection.onremovestream = handleRemoteStreamRemoved;
    console.log('Created RTCPeerConnection');
  } catch (e) {
    console.log('Failed to create PeerConnection, exception: ' + e.message);
    alert('Cannot create RTCPeerConnection object.');
  }
}
```

Processus d’appel

05 SIGNALISATION WEBRTC

```
function processCall(userName) {
  peerConnection.createOffer((sessionDescription) => {
    peerConnection.setLocalDescription(sessionDescription);
    sendCall({
      name: userName,
      rtcMessage: sessionDescription
    })
  }, (error) => {
    console.log("Error creating offer");
  });
}
```

- Crée une offre WebRTC (peerConnection.createOffer()).
- La définit comme description locale.
- Envoie l'offre via WebSocket au destinataire.

Processus d’appel

06 RÉPONDRE À L’APPEL

```
function answer() {
  beReady().then(() => {
    processAccept();
  });
  document.getElementById("answer").style.display = "none";
}
```

Processus d’appel

07 ÉCHANGE D’ICE (INTERACTIVE CONNECTIVITY ESTABLISHMENT)

```
function handleIceCandidate(event) {
  if (event.candidate) {
    console.log("Local ICE candidate");
    sendICEcandidate({
      user: otherUser,
      rtcMessage: {
        label: event.candidate.sdpMLineIndex,
        id: event.candidate.sdpMid,
        candidate: event.candidate.candidate
      }
    })
```

- Un candidat ICE contient des détails sur le réseau (IP, port, protocole) pour aider les pairs à trouver le meilleur chemin de connexion.

Processus d’appel

07 ÉCHANGE D’ICE (INTERACTIVE CONNECTIVITY ESTABLISHMENT)

```
const onICECandidate = (data) => {
  console.log("GOT ICE candidate");
  let message = data.rtcMessage;
  let candidate = new RTCIceCandidate({
    sdpMLineIndex: message.label,
    candidate: message.candidate
  });
  if (peerConnection) {
    console.log("ICE candidate Added");
    peerConnection.addIceCandidate(candidate);
```

- Fait partie de la fonction ConnectSocket.
- Reçoit les candidats ICE via WebSockets et Les ajoute à RTCPeerConnection.

Processus d’appel

08 APPEL EN COURS

```
function answerCall(data) {
  callSocket.send(JSON.stringify({
    type: 'answer_call',
    data
  }));
  callProgress();
}
```

```
function callProgress() {
  document.getElementById("videos").style.display = "block";
  document.getElementById("otherUserNameC").innerHTML = otherUser;
  document.getElementById("inCall").style.display = "block";
  document.getElementById("endVideoButton").style.display = "block";
  callInProgress = true;
}
```

- Affiche les flux vidéo et le bouton “End Call”

Processus d’appel

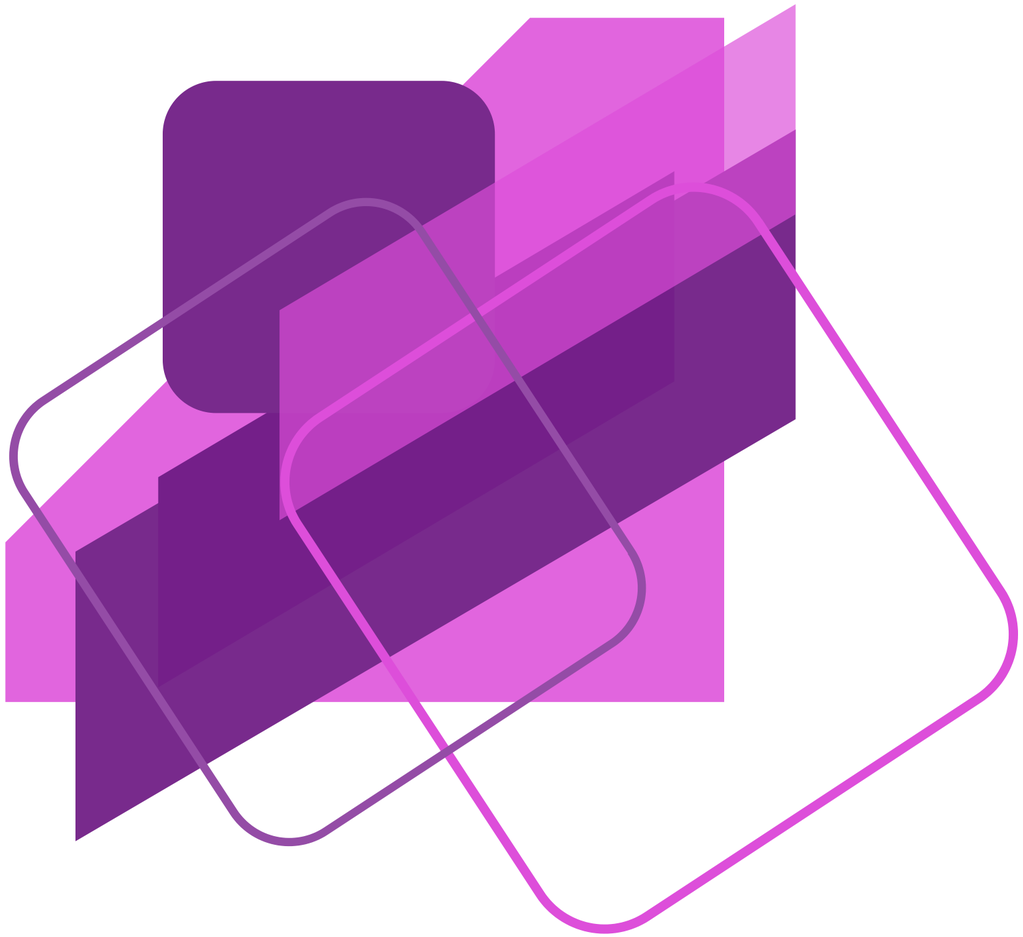
09 HANG UP

```
function endCall() {
  if (callSocket && callSocket.readyState === WebSocket.OPEN) {
    callSocket.send(JSON.stringify({
      type: 'end_call',
      data: {}
    }));
  }
  stop();
}
```

- Envoie un message “end_call” via WebSocket.
- Appelle stop() pour nettoyer les flux et l'interface utilisateur.

Démonstration

Conclusion



Merci pour votre écoute !

Team Collab