🔧 MediScribe - Résolution du Bug "Utilisateur non connecté"

🐛 Le Problème

Lors de la tentative de transcription audio, l'application affiche :

```
Erreur: Utilisateur non connecté
```

Symptômes :

- ✅ L'authentification fonctionne (connexion OK)
- ✅ La clé API Mistral est sauvegardée
- ✅ Le backend répond (testé avec N8N)
- ❌ La transcription échoue avec "Utilisateur non connecté"
- ❌ Aucune erreur dans la console navigateur

Environnements affectés :

- Local (localhost:8080)
- Production (Render.com)

🔍 Diagnostic de l'IA

Étape 1 : Localisation de l'erreur

L'IA a d'abord examiné le backend (`server.mjs`) et n'a trouvé **aucun message** "Utilisateur non connecté".

Conclusion : L'erreur vient du **frontend**, pas du backend.

Étape 2 : Analyse du code frontend

Dans `/src/pages/RecordPage.tsx` (ligne 53-59) :

```typescript
```

```
const handleTranscribe = async (blob: Blob) => {
  if (!user || !profile) {  // ⚠️ PROBLÈME ICI
    toast({
      title: 'Erreur',
      description: 'Utilisateur non connecté.',
      variant: 'destructive',
    });
    return;  // ❌ Arrêt immédiat
  }

  // Le reste du code n'est jamais exécuté...
}
```

Étape 3 : Identification de la cause racine

Le problème :

- `user` existe ✅ (utilisateur authentifié)
- `profile` est `null` ❌ (profil non chargé)

Pourquoi le profil est null ?

Dans `/src/contexts/AuthContext.tsx`, le chargement du profil est **asynchrone** :

```typescript
const fetchProfile = async () => {
  // Récupération du profil depuis Supabase
  // Peut échouer silencieusement si :
  // - Timeout réseau
  // - Profil inexistant en DB
  // - Erreur Supabase
}
```

Scénario du bug :

1. Utilisateur se connecte → `user` est défini
2. Profil commence à se charger (asynchrone)
3. Utilisateur clique sur "Transcrire" rapidement
4. `profile` n'est pas encore chargé → **null**
5. Vérification `!profile` → erreur affichée

✅ Solutions

Solutions

Solution 1 : Attendre le chargement du profil (Rapide)

Modifier `RecordPage.tsx` ligne 53 :

typescript

```typescript
const handleTranscribe = async (blob: Blob) => {
  // Vérifier uniquement l'utilisateur
  if (!user) {
    toast({
      title: 'Erreur',
      description: 'Utilisateur non connecté.',
      variant: 'destructive',
    });
    return;
  }

  // Si le profil n'est pas chargé, attendre ou utiliser user.id directement
  const userId = profile?.id || user.id;
  const apiKey = await getDecryptedApiKey(userId);

  // ... reste du code
}
```

Solution 2 : Afficher un état de chargement (UX améliorée)

typescript

```typescript
const handleTranscribe = async (blob: Blob) => {
  if (!user) {
```

```
    toast({
      title: 'Erreur',
      description: 'Utilisateur non connecté.',
      variant: 'destructive',
    });
    return;
  }

  // Afficher un loader si le profil charge
  if (!profile) {
    setProcessing(true);
    toast({
      title: 'Chargement',
      description: 'Préparation en cours...',
    });

    // Attendre max 5 secondes
    await new Promise(resolve => setTimeout(resolve, 5000));

    if (!profile) {
      toast({
        title: 'Erreur',
        description: 'Impossible de charger le profil.',
        variant: 'destructive',
      });
      setProcessing(false);
      return;
    }
  }

  // Continuer la transcription...
}
```

Solution 3 : Créer le profil automatiquement (Robuste)

Modifier `AuthContext.tsx` pour créer le profil si inexistant :

typescript

```
const fetchProfile = async () => {
  try {
    const { data, error } = await supabase
      .from('profiles')
      .select('*')
      .eq('id', user.id)
      .single();

    if (error) {
```

```
    // Si le profil n'existe pas, le créer
    if (error.code === 'PGRST116') {
      const { data: newProfile, error: createError } = await supabase
        .from('profiles')
        .insert([
          {
            id: user.id,
            email: user.email,
            specialty: 'Médecine générale', // Par défaut
            created_at: new Date().toISOString(),
          }
        ])
        .select()
        .single();

      if (!createError && newProfile) {
        setProfile(newProfile);
        return;
      }
    }
    throw error;
  }

  setProfile(data);
 } catch (err) {
  console.error('❌ Erreur profil:', err);
  setProfile(null);
 }
};
```

🚀 Pour déploiement sur Render

Checklist de vérification :

1. **Variables d'environnement** ✅

```
env

VITE_SUPABASE_URL=https://xxx.supabase.co
VITE_SUPABASE_ANON_KEY=xxx
VITE_MISTRAL_API_KEY=xxx
VITE_ENCRYPTION_KEY=xxx (32+ caractères)
```

2. **Base de données Supabase** ✅

Table `profiles` existe

- Table `profiles` existe
- Trigger `create_profile_on_signup` fonctionne
- Row Level Security configurée

3. **Tester la création de profil manuellement :**

```sql
-- Dans l'éditeur SQL Supabase
SELECT * FROM profiles WHERE id = 'USER_ID_ICI';

-- Si vide, créer manuellement :
INSERT INTO profiles (id, email, specialty)
VALUES ('USER_ID_ICI', 'email@example.com', 'Médecine générale');
```

4. **Vérifier les logs Render :**

```bash
# Dans le dashboard Render, onglet "Logs"
# Chercher des erreurs Supabase ou timeout
```

5. **Test en production :**
   - Ouvrir la console navigateur (F12)
   - Aller sur RecordPage
   - Chercher les logs : `🔄 Chargement profil`, `✅ Profil chargé`, `❌ Erreur`

🎯 Solution Recommandée (Combinée)

Fichier : `/src/pages/RecordPage.tsx`

```typescript
const handleTranscribe = async (blob: Blob) => {
  // Vérification utilisateur uniquement
  if (!user) {
    toast({
      title: 'Erreur',
      description: 'Veuillez vous reconnecter.',
      variant: 'destructive',
    });
    return;
  }
```

```
  try {
    setProcessing(true);

    // Utiliser user.id directement (plus fiable que profile?.id)
    const userId = user.id;

    // Récupérer la clé API
    const apiKey = await getDecryptedApiKey(userId);

    if (!apiKey) {
      toast({
        title: 'Configuration requise',
        description: 'Veuillez configurer votre clé API Mistral dans les paramètres.',
        variant: 'destructive',
      });
      navigate('/settings');
      return;
    }

    // Transcription
    const transcription = await BackendMistralService.transcribeAudio(
      blob,
      userId, // ✅ Utilise user.id directement
      apiKey
    );

    // ... reste du code
  } catch (error) {
    console.error('❌ Erreur transcription:', error);
    toast({
      title: 'Erreur',
      description: error instanceof Error ? error.message : 'Erreur transcription',
      variant: 'destructive',
    });
  } finally {
    setProcessing(false);
  }
};
```

Avantages :

- ✅ Utilise `user.id` directement (toujours disponible)
- ✅ Gère les erreurs proprement
- ✅ Ne dépend plus de `profile`

- ✅ Fonctionne en local ET sur Render

📝 Instructions de déploiement

1. Appliquer le fix

```bash
# 1. Modifier RecordPage.tsx avec le code ci-dessus
# 2. Tester en local
npm run dev

# 3. Commiter et pusher
git add src/pages/RecordPage.tsx
git commit -m "fix: Résolution bug 'Utilisateur non connecté'"
git push origin main
```

2. Sur Render

Le déploiement se fera automatiquement si lié à GitHub.

Ou manuellement :

```bash
# Dans le dashboard Render
1. Aller sur votre service
2. Cliquer "Manual Deploy" → "Deploy latest commit"
3. Attendre le build (~2-3 min)
4. Tester l'application
```

3. Vérification post-déploiement

```bash
# Test API backend
curl https://votre-app.onrender.com/api/health
```

```
# Devrait retourner :
{
  "status": "OK",
  "timestamp": "...",
  "ai_provider": "Mistral AI"
}
```

🆘 Si le problème persiste

Debug checklist :

1. **Console navigateur (F12) :**
   - Y a-t-il des erreurs CORS ?
   - Les requêtes API passent-elles ?
   - Le token Supabase est-il valide ?
2. **Logs Render :**
   - Erreurs 500 backend ?
   - Timeout Supabase ?
   - Clés API manquantes ?
3. **Supabase Dashboard :**
   - Table `profiles` vide ?
   - Trigger de création de profil désactivé ?
   - RLS trop restrictive ?
4. **Test direct de l'API :**

```bash
# Tester transcription directement
curl -X POST https://votre-app.onrender.com/api/transcribe \
  -H "x-user-id: USER_ID_ICI" \
  -H "x-api-key: MISTRAL_KEY_ICI" \
  -F "file=@test.mp3"
```

📚 Ressources

- **Documentation Supabase Auth :** https://supabase.com/docs/guides/auth

- **Mistral API Docs :** https://docs.mistral.ai/
- **Render Deploy Guide :** https://render.com/docs/deploy-vite

---

✅ **Résumé :** Le bug venait d'une vérification trop stricte dans le frontend qui bloquait si `profile` n'était pas chargé. La solution consiste à utiliser `user.id` directement au lieu de `profile?.id`, ce qui est plus fiable et fonctionne immédiatement après la connexion.