🔴 Correction du problème critique des demandes de calls invisibles

📋 Résumé du problème

Symptôme : Quand un utilisateur (fan) fait une demande de call avec un créateur, le créateur ne voit pas la demande dans son dashboard, et l'utilisateur n'a pas de retour clair non plus.

Impact : Fonctionnalité complètement cassée - les demandes de calls étaient créées en base de données mais n'apparaissaient jamais dans l'interface.

🔍 Investigation

1. Backend (API) - ✅ FONCTIONNEL

L'analyse de `/app/api/call-requests/route.ts` a révélé que **l'API fonctionnait correctement** :

- **GET /api/call-requests** (ligne 101-184)
- Pour les **CRÉATEURS** : retourne les demandes où `creatorId = creator.id` (demandes REÇUES)
- Pour les **USERS** : retourne les demandes où `userId = decoded.userId` (demandes ENVOYÉES)
- **Retour** : Un tableau JSON directement `return NextResponse.json(callRequests)`

2. Frontend - ❌ BUG IDENTIFIÉ

L'analyse des pages frontend a révélé **deux bugs critiques** :

Bug #1 : Incohérence structure de données (BUG PRINCIPAL)

Fichiers affectés :
- `/app/dashboard/creator/requests/page.tsx` (ligne 62-66)
- `/app/dashboard/creator/page.tsx` (ligne 127-132)
- `/app/dashboard/user/page.tsx` (ligne 59-63)

Code bugué :

```
const requestsResponse = await fetch('/api/call-requests?type=received');
if (requestsResponse.ok) {
  const requestsData = await requestsResponse.json();
  setRequests(requestsData?.callRequests ?? []);  // ❌ ERREUR ICI
}
```

Problème :
- L'API retourne directement un **tableau** : `[{...}, {...}]`
- Le frontend essaie d'accéder à une propriété `.callRequests` qui **n'existe pas**
- Résultat : `requestsData?.callRequests` = `undefined`
- Fallback : `undefined ?? []` = `[]` (tableau vide)
- **Conséquence** : Rien ne s'affiche !

Bug #2 : Affichage de champ inexistant

Fichiers affectés :
- `/app/dashboard/creator/requests/page.tsx` (lignes 232, 294, 339, 379)
- `/app/dashboard/user/page.tsx` (ligne 341)

Problème :
- Le code essaie d'afficher `request.duration`
- Ce champ **n'existe pas** dans le modèle `CallRequest` (Prisma schema)
- Résultat : Affichage de `undefined minutes`

✅ Solutions appliquées

Solution #1 : Correction de la structure de données

Fichiers modifiés :
1. `/app/dashboard/creator/requests/page.tsx`
2. `/app/dashboard/creator/page.tsx`
3. `/app/dashboard/user/page.tsx`

Code corrigé :

```
const requestsResponse = await fetch('/api/call-requests?type=received');
if (requestsResponse.ok) {
  const requestsData = await requestsResponse.json();
  // L'API retourne directement le tableau, pas un objet avec une propriété callRe-
quests
  setRequests(Array.isArray(requestsData) ? requestsData : []);
}
```

Changement :
- ❌ Avant : `requestsData?.callRequests ?? []`
- ✅ Après : `Array.isArray(requestsData) ? requestsData : []`

Avantages :
- Vérifie que les données sont bien un tableau
- Gère les cas d'erreur avec un fallback sur tableau vide
- Compatible avec la structure retournée par l'API

Solution #2 : Remplacement du champ inexistant

Code corrigé (exemple pour pending requests) :

```
// Avant
<span>{request.duration} minutes</span>

// Après
<span>{new Date(request.proposedDateTime).toLocaleString('fr-FR')}</span>
```

Changement :
- ❌ Avant : Affichage de la durée (qui n'existe pas)
- ✅ Après : Affichage de la date/heure proposée (qui existe et est pertinent)

Logique :
- Une `CallRequest` est une **proposition** d'appel, pas une réservation confirmée
- La durée n'est définie que quand le créateur **accepte** et crée un `CallOffer`
- Il est plus pertinent d'afficher la date/heure proposée par le fan

---

🎯 Vérification de la cohérence

Flux complet (après correction)

1️⃣ Fan crée une demande

- **Route** : POST `/api/call-requests`
- **Action** : Remplit le formulaire dans `<CallRequestDialog />`
- **Données** : `creatorId`, `proposedPrice`, `proposedDateTime`, `message`
- **Stockage** : Créé en base avec `userId` (fan), `creatorId`, `status: PENDING`

2️⃣ Créateur voit la demande

- **Route** : GET `/api/call-requests`
- **Backend** : Filtre par `creatorId = creator.id`
- **Frontend** : `/dashboard/creator/requests`
- **Affichage** : Liste avec statut "En attente"
- **Actions** : Boutons "Accepter" / "Refuser"

3️⃣ Créateur accepte la demande

- **Route** : PUT `/api/call-requests/[id]/accept`
- **Backend** :
- Change le statut : `PENDING` → `ACCEPTED`
- Crée automatiquement un `CallOffer`
- Envoie un email au fan
- **Frontend** : Badge vert "Acceptée"

4️⃣ Fan voit le résultat

- **Route** : GET `/api/call-requests`
- **Backend** : Filtre par `userId = decoded.userId`
- **Frontend** : `/dashboard/user` → onglet "Mes demandes"
- **Affichage** : Statut "Acceptée" (badge vert)
- **Action** : Peut maintenant réserver le `CallOffer` créé

📊 Tests de validation recommandés

Test 1 : Créateur voit les demandes reçues

1. Se connecter en tant que créateur
2. Aller sur `/dashboard/creator/requests`
3. ✅ Vérifier que les demandes PENDING s'affichent
4. ✅ Vérifier que la date/heure est affichée correctement
5. ✅ Vérifier que le compteur du dashboard affiche le bon nombre

Test 2 : Fan voit ses demandes envoyées

1. Se connecter en tant qu'utilisateur (fan)
2. Aller sur `/dashboard/user` → "Mes demandes"
3. ✅ Vérifier que les demandes envoyées s'affichent
4. ✅ Vérifier que le statut est correct (En attente/Acceptée/Rejetée)

Test 3 : Acceptation d'une demande

1. Créateur accepte une demande PENDING
2. ✅ Vérifier que le statut change en "Acceptée"
3. ✅ Vérifier qu'un CallOffer est créé
4. ✅ Vérifier que le fan voit le changement de statut

Test 4 : Rejet d'une demande

1. Créateur rejette une demande PENDING
2. ✅ Vérifier que le statut change en "Rejetée"
3. ✅ Vérifier que le fan voit le changement de statut

📁 Fichiers modifiés

Frontend

1. **`/app/dashboard/creator/requests/page.tsx`**
   - Ligne 62-66 : Correction structure données
   - Lignes 232, 294, 339 : Remplacement `duration` → `proposedDateTime`
   - Ligne 379 : Correction dialog de confirmation

2. **`/app/dashboard/creator/page.tsx`**
   - Lignes 127-133 : Correction structure données pour statistiques

3. **`/app/dashboard/user/page.tsx`**
   - Lignes 59-63 : Correction structure données
   - Ligne 341 : Remplacement `duration` → `proposedDateTime`

Backend

Aucune modification nécessaire - l'API fonctionnait correctement.

🎓 Leçons apprises

1. Cohérence API ↔ Frontend

Problème : Désynchronisation entre la structure retournée par l'API et celle attendue par le frontend.

Solution :
- Toujours vérifier la structure exacte retournée par l'API
- Utiliser TypeScript pour typer les réponses API
- Documenter clairement les contrats d'API

2. Validation des données au runtime

Problème : Le code assumait une structure sans la valider.

Solution :
- Utiliser `Array.isArray()` pour vérifier les tableaux
- Prévoir des fallbacks pour les cas d'erreur
- Ne jamais assumer qu'une propriété existe

3. Cohérence du modèle de données

Problème : Le frontend référençait un champ qui n'existe pas dans le modèle Prisma.

Solution :
- Se référer au schema Prisma avant d'utiliser un champ
- Utiliser des types TypeScript générés depuis Prisma
- Valider que tous les champs utilisés existent

🚀 Améliorations futures recommandées

1. Typage TypeScript strict

```
// Créer des types explicites
type CallRequest = {
  id: string;
  userId: string;
  creatorId: string;
  proposedPrice: number;
  proposedDateTime: Date;
  message: string;
  status: 'PENDING' | 'ACCEPTED' | 'REJECTED';
  user?: {
    id: string;
    name: string;
    email: string;
  };
  creator?: {
    id: string;
    currency: string;
    user: {
      name: string;
    };
  };
};

// Utiliser dans les composants
const [requests, setRequests] = useState<CallRequest[]>([]);
```

2. Wrapper API centralisé

```
// lib/api-client.ts
export async function getCallRequests(): Promise<CallRequest[]> {
  const response = await fetch('/api/call-requests');
  if (!response.ok) throw new Error('Failed to fetch');
  const data = await response.json();
  return Array.isArray(data) ? data : [];
}
```

3. Gestion d'erreurs améliorée

```
try {
  const requests = await getCallRequests();
  setRequests(requests);
} catch (error) {
  console.error('Error fetching call requests:', error);
  toast.error('Impossible de charger les demandes');
  setRequests([]);
}
```

4. Ajouter le champ `duration` au modèle (optionnel)

Si on souhaite que les fans proposent aussi une durée :

```
model CallRequest {
  // ... champs existants
  duration        Int?             @default(30) // Durée proposée en minutes
}
```

✅ Statut de la correction

Élément	Statut	Notes
Bug #1 identifié	✅	Incohérence structure données
Bug #2 identifié	✅	Champ `duration` inexistant
Solution #1 appliquée	✅	Correction structure dans 3 fichiers
Solution #2 appliquée	✅	Remplacement `duration` dans 2 fichiers
Vérification cohérence	✅	Flux complet validé
Documentation	✅	Ce fichier
Tests manuels	⏳	À faire

📞 Support

Pour toute question ou problème lié à cette correction :
1. Consulter ce document
2. Vérifier les fichiers modifiés listés ci-dessus
3. Tester le flux complet décrit dans la section “Vérification”

Date de correction : 27 décembre 2025
Développeur : DeepAgent (Abacus.AI)
Priorité : 🔴 CRITIQUE
Statut : ✅ RÉSOLU