📋 Flux de réservation et paiement - Call a Star

🔄 Vue d'ensemble

Le système de réservation et paiement a été **complètement refactorisé** pour résoudre un problème critique où les créneaux étaient bloqués sans paiement effectué.

❌ Ancien flux (PROBLÉMATIQUE)

1. L'utilisateur clique sur "Réserver"
2. **Le booking est créé immédiatement** (status: PENDING)
3. Le créneau devient **indisponible** pour tous les autres utilisateurs
4. L'utilisateur arrive sur la page de paiement
5. **Si l'utilisateur abandonne/quitte** → le créneau reste bloqué ❌

Problème : Les créneaux étaient bloqués même si aucun paiement n'était effectué.

✅ Nouveau flux (CORRECT)

1. L'utilisateur clique sur "Réserver"
2. **Création du Payment Intent Stripe uniquement** (pas de booking)
3. Vérification que le créneau est disponible (status = AVAILABLE)
4. L'utilisateur arrive sur la page de paiement
5. **Si l'utilisateur abandonne** → le créneau reste disponible ✅
6. **Si le paiement réussit** → webhook `payment_intent.succeeded` reçu
7. **Le booking est créé dans le webhook** (transaction atomique)
8. Création de la room Daily.io
9. Envoi des emails de confirmation

Avantages :
- ✅ Les créneaux ne sont plus bloqués sans paiement
- ✅ Protection anti multi-booking avec transaction atomique
- ✅ Le booking est créé uniquement après paiement confirmé

📁 Fichiers modifiés

1. Backend

`app/api/payments/create-intent/route.ts`

Changements :
- ✅ Accepte `callOfferId` au lieu de `bookingId`
- ✅ Vérifie la disponibilité du CallOffer avant de créer le Payment Intent
- ✅ Ajoute les métadonnées nécessaires : `callOfferId`, `userId`, `totalPrice`, `platformFee`, `creat-

`orAmount`, `bookingFlow: 'payment_first'`
- ✅ **NE crée PAS de booking ni de payment record**

Exemple de requête :

```
POST /api/payments/create-intent
{
  "callOfferId": "clx123abc" // Au lieu de bookingId
}
```

Réponse :

```
{
  "clientSecret": "pi_xxx_secret_xxx",
  "paymentIntentId": "pi_xxx",
  "currency": "EUR",
  "amount": 100
}
```

`app/api/payments/webhook/route.ts`

Changements :
- ✅ Détecte le nouveau flux via `metadata.bookingFlow === 'payment_first'`
- ✅ Crée le booking lors du `payment_intent.succeeded` (avec transaction atomique)
- ✅ Vérifie que le CallOffer est toujours disponible
- ✅ Crée la room Daily.io
- ✅ Crée le payment record
- ✅ Envoie les emails de confirmation
- ✅ Gère les cas d'erreur (créneau déjà pris → refund requis)
- ✅ Maintient la rétrocompatibilité avec l'ancien flux (legacy)

Flux de traitement :

```
payment_intent.succeeded →
  Détection du flux (bookingFlow: 'payment_first') →
  Transaction atomique :
    - Vérification CallOffer disponible
    - Création du booking
    - Mise à jour du CallOffer (status: BOOKED)
  → Création room Daily.io
  → Création payment record
  → Envoi emails
```

`app/api/bookings/route.ts`

Changements :
- ✅ Route POST **désactivée** (retourne 410 Gone)
- ✅ Message d'erreur clair expliquant le nouveau flux
- ✅ Code legacy conservé en commentaire pour référence
- ✅ Route GET maintenue (récupération des bookings existants)

Réponse si la route est appelée :

```
{
  "error": "Cette route est obsolète. Utilisez /api/payments/create-intent avec
callOfferId pour initier un paiement.",
  "details": "Le booking sera créé automatiquement après confirmation du paiement."
}
```

---

2. Frontend

`app/[locale]/book/[offerId]/page.tsx`

Changements :
- ✅ Appelle directement `/api/payments/create-intent` avec `callOfferId`
- ✅ **N'appelle PLUS** `/api/bookings` pour créer un booking
- ✅ Utilise un booking temporaire pour l'affichage UI (status: PENDING_PAYMENT)
- ✅ Le booking réel sera créé après paiement confirmé

Ancien code :

```
// 1. Créer le booking
const bookingResponse = await fetch('/api/bookings', {
  method: 'POST',
  body: JSON.stringify({ callOfferId }),
});

// 2. Créer le Payment Intent
const intentResponse = await fetch('/api/payments/create-intent', {
  method: 'POST',
  body: JSON.stringify({ bookingId }),
});
```

Nouveau code :

```
// ✅ Créer le Payment Intent directement (pas de booking)
const intentResponse = await fetch('/api/payments/create-intent', {
  method: 'POST',
  body: JSON.stringify({ callOfferId }), // callOfferId au lieu de bookingId
});

// Le booking sera créé après paiement (webhook)
```

🔐 Sécurité et protection anti multi-booking

Transaction atomique dans le webhook

Le booking est créé avec une **transaction atomique** Prisma pour éviter les race conditions :

```
const result = await prisma.$transaction(async (tx) => {
  // 1. Vérifier que le CallOffer existe et est disponible
  const callOffer = await tx.callOffer.findUnique({
    where: { id: callOfferId },
    include: { booking: true },
  });

  if (!callOffer) throw new Error('OFFER_NOT_FOUND');
  if (callOffer.status !== 'AVAILABLE') throw new Error('OFFER_NOT_AVAILABLE');
  if (callOffer.booking) throw new Error('OFFER_ALREADY_BOOKED');

  // 2. Créer le booking atomiquement
  const booking = await tx.booking.create({
    data: {
      userId,
      callOfferId,
      totalPrice,
      status: 'PENDING',
      stripePaymentIntentId: paymentIntent.id,
    },
  });

  // 3. Mettre à jour le CallOffer
  await tx.callOffer.update({
    where: { id: callOfferId },
    data: { status: 'BOOKED' },
  });

  return booking;
});
```

Protections :
- ✅ La transaction garantit l'atomicité (tout ou rien)
- ✅ La contrainte unique sur `callOfferId` dans le modèle Booking empêche les doublons
- ✅ Si deux utilisateurs paient simultanément, seul le premier réussira

---

🔄 Gestion des cas d'erreur

Cas 1 : Créneau déjà pris pendant le paiement

Si le créneau est pris par un autre utilisateur PENDANT que le premier utilisateur paie :

```
if (error.message === 'OFFER_ALREADY_BOOKED') {
  console.error('[Webhook] CRITICAL: Slot already booked by another user during pay-
ment');
  // ✅ TODO: Implement automatic refund here
  throw new Error('SLOT_TAKEN_REFUND_REQUIRED');
}
```

Action requise : Implémenter le remboursement automatique (TODO)

Cas 2 : Paiement abandonné/expiré

Ancien flux : Le créneau restait bloqué ❌
Nouveau flux : Le créneau reste disponible automatiquement ✅

Aucune action requise car aucun booking n'a été créé.

Cas 3 : Webhook Stripe non reçu

Risque : Le paiement est effectué mais le booking n'est pas créé.

Solutions :
1. Stripe réessaie automatiquement les webhooks pendant 3 jours
2. Vérifier les logs Stripe Dashboard pour les webhooks échoués
3. Rejouer manuellement les webhooks si nécessaire

📊 Métadonnées du Payment Intent

Le Payment Intent contient toutes les données nécessaires pour créer le booking dans le webhook :

```
{
  "callOfferId": "clx123abc",
  "userId": "user123",
  "creatorId": "creator456",
  "totalPrice": "100.00",
  "platformFee": "15.00",
  "creatorAmount": "85.00",
  "currency": "EUR",
  "useStripeConnect": "true",
  "bookingFlow": "payment_first" // ✅ Indicateur du nouveau flux
}
```

🧪 Tests

Test 1 : Vérifier qu'un créneau n'est pas bloqué sans paiement

1. Ouvrir la page de réservation d'un créneau
2. Arriver sur la page de paiement
3. **Fermer l'onglet sans payer**
4. Vérifier que le créneau est toujours disponible ✅

Test 2 : Vérifier que le booking est créé après paiement

1. Réserver un créneau et payer
2. Vérifier dans la base de données :
   - Le booking est créé avec le bon `stripePaymentIntentId`

- Le CallOffer a `status: 'BOOKED'`
- La room Daily.io est créée

3. Vérifier que l'email de confirmation est envoyé ✅

Test 3 : Protection anti multi-booking

1. Deux utilisateurs ouvrent la même page de réservation
2. Les deux paient simultanément
3. **Seul le premier paiement réussit** (le deuxième reçoit une erreur) ✅

🔧 Migration et rétrocompatibilité

Gestion des Payment Intents existants

Le webhook détecte automatiquement l'ancien flux :

```
const isNewFlow = metadata?.bookingFlow === 'payment_first';

if (isNewFlow) {
  // ✅ Nouveau flux : créer le booking
  // ...
} else {
  // ✅ Legacy : le booking existe déjà
  // ...
}
```

Rétrocompatibilité :
- ✅ Les Payment Intents créés avec l'ancien flux (avant cette refactorisation) continuent de fonctionner
- ✅ Les bookings existants ne sont pas affectés

📝 Checklist de déploiement

Avant de déployer en production :

- [x] Vérifier que les webhooks Stripe sont configurés (`payment_intent.succeeded`)
- [x] Tester le nouveau flux en mode test Stripe
- [x] Vérifier que les emails de confirmation sont envoyés
- [x] Tester la protection anti multi-booking
- [x] Vérifier les logs système (SystemLog) pour les erreurs
- [ ] Implémenter le remboursement automatique en cas de créneau déjà pris (TODO)
- [ ] Monitorer les webhooks Stripe pendant les premières 24h après déploiement

---

🚨 Points d'attention

1. Webhooks Stripe

CRITIQUE : Sans webhooks configurés, les bookings ne seront jamais créés !

Vérifier :
- L'URL du webhook est correcte : `https://your-domain.com/api/payments/webhook`
- Le secret webhook est configuré : `STRIPE_WEBHOOK_SECRET`
- L'événement `payment_intent.succeeded` est activé

---

2. Remboursement automatique (TODO)

Si le créneau est pris pendant le paiement, le remboursement doit être automatique :

```
if (error.message === 'OFFER_ALREADY_BOOKED') {
  // TODO: Implement automatic refund
  const refund = await stripe.refunds.create({
    payment_intent: paymentIntent.id,
    reason: 'requested_by_customer',
  });

  // Send email notification to user
}
```

---

3. Monitoring

Surveiller :
- Les webhooks échoués dans Stripe Dashboard
- Les erreurs `OFFER_ALREADY_BOOKED` dans les logs
- Les Payment Intents sans booking associé (après 24h)

📚 Ressources

- Stripe Payment Intents (https://stripe.com/docs/payments/payment-intents)
- Stripe Webhooks (https://stripe.com/docs/webhooks)
- Prisma Transactions (https://www.prisma.io/docs/concepts/components/prisma-client/transactions)

Auteur : DeepAgent
Date : 2026-01-01
Version : 1.0