

# Correction des Devises et Affichage des Montants

---

## Résumé des Problèmes et Solutions

---

Ce document détaille les 6 problèmes identifiés concernant l’affichage des devises et des montants, ainsi que les solutions mises en œuvre pour les corriger.

---

## Problèmes Identifiés

---

### Problème 1 : Notifications de paiement trop verbeuses

**Description :** Les notifications affichaient trop d’informations (montant brut, marge, commission, calculs internes).

**Comportement souhaité :** “Paiement reçu - Vous avez reçu un paiement de 500.00 GBP.”

### Problème 2 : Devise EUR affichée partout au lieu de la devise Stripe

**Description :** EUR s’affichait partout (earnings, création d’offres, listings) alors que le compte Stripe du créateur était en GBP.

**Comportement souhaité :** Afficher la devise du compte Stripe Connect du créateur partout.

### Problème 3 : Historique des paiements avec montants faux

**Description :** L’historique affichait le montant après marge (85) au lieu du montant réel payé (100).

**Comportement souhaité :** Afficher le montant payé par l’utilisateur, pas après marge.

### Problème 4 : Explications des marges dans les notifications

**Description :** Les explications de marge (% plateforme + frais Stripe) étaient affichées dans les notifications et l’historique simple.

**Comportement souhaité :** Déplacer les explications dans une page dédiée/settings/tooltip.

### Problème 5 : Création de demandes/offres avec mauvaise devise

**Description :** Lors de la création de demandes, le prix s’affichait en EUR au lieu de la devise réelle du créateur.

**Comportement souhaité :** Afficher le prix dans la devise du créateur.

### Problème 6 : Symbole de devise incorrect

**Description :** Certains paiements affichaient \$ avec GBP ou CHF (ex: \$500 GBP).

**Comportement souhaité :** Ne jamais afficher \$ par défaut, toujours afficher le code ISO (500.00 GBP, 500.00 CHF).

---

## ✓ Solutions Appliquées

### Solution 1 : Simplification des Notifications de Paiement

Fichier modifié : `/app/api/payments/webhook/route.ts`

Changements :

```
// AVANT (ligne 577-598)
await createNotification({
  userId: booking.callOffer.creator.userId,
  type: 'PAYMENT_RECEIVED',
  title: '💰 Nouveau paiement reçu',
  message: `Vous avez reçu un paiement de ${totalAmount.toFixed(2)} ${currency} pour "${booking.callOffer.title}". Votre part : ~${creatorAmount.toFixed(2)} ${currency} (après commission de ${platformFee.toFixed(2)} ${currency}). Les fonds sont disponibles sur votre compte Stripe.`,
  link: '/dashboard/creator/payouts',
  metadata: { ... },
});

// APRÈS (ligne 577-593)
await createNotification({
  userId: booking.callOffer.creator.userId,
  type: 'PAYMENT_RECEIVED',
  title: 'Paiement reçu',
  message: `Vous avez reçu un paiement de ${totalAmount.toFixed(2)} ${currency}.`,
  link: '/dashboard/creator/payments',
  metadata: { ... },
});
```

**Impact :** Les notifications sont maintenant simples et claires, sans détails de commission.

### Solution 2 : Récupération Automatique de la Devise Stripe

Fichiers modifiés :

1. `/lib/stripe.ts` - Ajout de fonctions `getCreatorCurrency()` et `getCreatorCurrencyByStripeAccount()`
2. `/app/api/payments/webhook/route.ts` - Mise à jour automatique de la devise lors du webhook `account.updated`

Changements :

## 1. Ajout de fonctions utilitaires dans `/lib/stripe.ts`

```
/**
 * Get creator's currency from DB or Stripe account
 *
 * ✅ Priority:
 * 1. Check creator.currency in DB (cached value)
 * 2. Fetch from Stripe account if not in DB
 * 3. Update DB with fetched currency
 * 4. Default to 'EUR' if all fails
 */
export async function getCreatorCurrency(creatorId: string): Promise<string> {
  try {
    const creator = await prisma.creator.findUnique({
      where: { id: creatorId },
      select: { currency: true, stripeAccountId: true },
    });

    if (!creator) return 'EUR';
    if (creator.currency) return creator.currency.toUpperCase();

    if (!creator.stripeAccountId) return 'EUR';

    const stripeAccount = await stripe.accounts.retrieve(creator.stripeAccountId);
    const currency = (stripeAccount.default_currency || 'eur').toUpperCase();

    await prisma.creator.update({
      where: { id: creatorId },
      data: { currency },
    });

    return currency;
  } catch (error) {
    console.error(`[getCreatorCurrency] Error:`, error);
    return 'EUR';
  }
}
```

## 2. Mise à jour automatique dans le webhook `account.updated`

```
// Dans handleAccountUpdated() - ligne 1713-1724
const currency = (account.default_currency || 'eur').toUpperCase();

await prisma.creator.update({
  where: { id: creator.id },
  data: {
    isStripeOnboarded: accountStatus.isFullyOnboarded,
    payoutBlocked: shouldBlockPayout,
    payoutBlockedReason: blockReason,
    currency: currency, // ✅ Mise à jour de la devise
  },
});
```

**Impact :** La devise du créateur est automatiquement récupérée depuis Stripe et mise en cache dans la DB. Elle est mise à jour lors de chaque changement du compte Stripe.

## Solution 3 : Affichage des Montants Bruts

### Fichiers modifiés :

1. `/app/dashboard/creator/payments/page.tsx`
2. `/app/dashboard/creator/earnings/page.tsx`
3. `/app/api/creator/earnings/route.ts`

### Changements :

#### 1. Dans `/app/dashboard/creator/payments/page.tsx`

```
// AVANT (ligne 73)
const totalAmount = payments.reduce((sum, p) => sum + Number(p.creatorAmount ?? 0), 0)
;

// APRÈS (ligne 73-74)
// ✅ FIX: Display gross amounts (what users paid), not net amounts after commission
const totalAmount = payments.reduce((sum, p) => sum + Number(p.amount ?? 0), 0);

// AVANT (ligne 178)
amount={Number(payment.creatorAmount)}

// APRÈS (ligne 179)
amount={Number(payment.amount)}
```

#### 2. Dans `/app/api/creator/earnings/route.ts`

```
// AVANT (ligne 96-97)
const paidPayments = payments.filter(p => p.payoutStatus === PayoutStatus.PAID);
totalEarnings = paidPayments.reduce((sum, p) => sum + Number(p.creatorAmount), 0);

// APRÈS (ligne 95-97)
// ✅ FIX: Use gross amount (what users paid), not net after commission
const paidPayments = payments.filter(p => p.payoutStatus === PayoutStatus.PAID);
totalEarnings = paidPayments.reduce((sum, p) => sum + Number(p.amount), 0);
```

**Impact :** Les montants affichés correspondent maintenant au montant total payé par l'utilisateur (100 EUR) au lieu du montant après commission (85 EUR).

## Solution 4 : Création d'une Page Dédiée aux Frais

**Fichier créé :** `/app/dashboard/creator/fees/page.tsx`

### Contenu :

- Explication de la commission de la plateforme (15%)
- Explication des frais Stripe (2.9% + 0.30 EUR)
- Calculateur d'exemple montrant la répartition
- Notes importantes sur les transferts et les montants affichés

**Fichier modifié :** `/app/dashboard/creator/payments/page.tsx`

### Changements :

```
// Ajout d'un lien vers la page des frais dans l'Info Card (ligne 208)
<p><strong>Montants
affichés :</strong> Les montants affichés correspondent au montant total payé par
l'utilisateur. Pour en savoir plus sur les frais et commissions, consultez la page
<Link href="/dashboard/creator/fees" className="underline font-semibold">Frais et
commissions</Link>.</p>
```

**Impact :** Les explications détaillées des frais sont maintenant dans une page dédiée, accessibles via un lien. Les notifications et l'historique restent simples.

## Solution 5 : Correction du Formatage des Devises

### Fichiers modifiés :

1. `/lib/currency-utils.ts`
2. `/components/ui/currency-display.tsx`

### Changements :

#### 1. Dans `/lib/currency-utils.ts`

```
// AVANT (ligne 55-73)
export function formatCurrency(
  amount: number | string,
  currency: string = 'EUR',
  locale: string = 'fr-FR'
): string {
  const numAmount = typeof amount === 'string' ? parseFloat(amount) : amount;

  return new Intl.NumberFormat(locale, {
    style: 'currency',
    currency: currency.toUpperCase(),
    minimumFractionDigits: 2,
    maximumFractionDigits: 2,
  }).format(numAmount);
}

// APRÈS (ligne 55-73)
/**
 * Format amount with currency CODE (not symbol)
 *
 * ☒ ALWAYS displays "amount CODE" format (e.g., "500.00 GBP")
 * ☒ Never displays incorrect symbols (e.g., no more "$500 GBP")
 */
export function formatCurrency(
  amount: number | string,
  currency: string = 'EUR'
): string {
  const numAmount = typeof amount === 'string' ? parseFloat(amount) : amount;
  const currencyCode = currency.toUpperCase();

  return `${numAmount.toFixed(2)} ${currencyCode}`;
}

// Ajout d'une fonction legacy pour les cas spéciaux
export function formatCurrencyWithSymbol(...) { ... }
```

## 2. Dans `/components/ui/currency-display.tsx`

```
// AVANT
import { getCurrencySymbol } from '@lib/currency-converter';

export function CurrencyDisplay({ amount, currency, className = '', showSymbol = false }) {
  const displayCurrency = showSymbol ? getCurrencySymbol(currency) : currency.toUpperCase();
  return <span className={className}>{amount.toFixed(2)} {displayCurrency}</span>;
}

// APRÈS
import { formatCurrency } from '@lib/currency-utils';

export function CurrencyDisplay({ amount, currency, className = '' }) {
  const formattedAmount = formatCurrency(amount, currency);
  return <span className={className}>{formattedAmount}</span>;
}
```

### Impact :

- Les montants s'affichent toujours au format "500.00 GBP" (jamais "\$500 GBP")
- Le code ISO est toujours utilisé (GBP, EUR, CHF) au lieu des symboles
- Cohérence visuelle dans toute l'application

## Solution 6 : Utilisation Systématique de la Devise du Créateur

### Fichiers concernés :

- Tous les fichiers utilisent maintenant `creator.currency || 'EUR'` comme fallback
- La devise est automatiquement mise à jour via le webhook
- Les pages frontend récupèrent la devise depuis `userData.user.creator.currency`

### Exemples :

```
// Dans les pages frontend
const [creatorCurrency, setCreatorCurrency] = useState<string>('EUR');

useEffect(() => {
  if (userData?.user?.creator?.currency) {
    setCreatorCurrency(userData.user.creator.currency);
  }
}, [userData]);

// Dans les APIs backend
const currency = creator.currency || 'EUR';
```

**Impact :** La devise du créateur est utilisée de manière cohérente partout dans l'application.

## Fichiers Modifiés

### Fichiers Backend (API)

1. `/app/api/payments/webhook/route.ts` - Simplification notifications, mise à jour devise

2. `/app/api/creator/earnings/route.ts` - Affichage montants bruts

## Fichiers Frontend (Pages)

1. `/app/dashboard/creator/payments/page.tsx` - Affichage montants bruts, lien vers page fees
2. `/app/dashboard/creator/earnings/page.tsx` - Affichage montants bruts
3. `/app/dashboard/creator/fees/page.tsx` - **NOUVEAU** - Page explicative des frais

## Fichiers Librairie

1. `/lib/stripe.ts` - Ajout `getCreatorCurrency()` et `getCreatorCurrencyByStripeAccount()`
2. `/lib/currency-utils.ts` - Modification `formatCurrency()` pour toujours utiliser code ISO
3. `/components/ui/currency-display.tsx` - Simplification composant



## Tests et Vérification

### Tests Manuels à Effectuer

#### 1. Test de la Devise

- [ ] Vérifier que la devise du créateur est correctement affichée dans le dashboard
- [ ] Créer une offre et vérifier que la devise affichée est correcte
- [ ] Vérifier que les montants dans l'historique utilisent la bonne devise

#### 2. Test des Notifications

- [ ] Simuler un paiement et vérifier que la notification est simple
- [ ] Vérifier que le format est : "Vous avez reçu un paiement de X.XX YY."
  - [ ] Vérifier que la devise affichée est correcte (GBP, EUR, CHF, etc.)

#### 3. Test de l’Affichage des Montants

- [ ] Vérifier que l'historique des paiements affiche les montants bruts
- [ ] Vérifier que les totaux dans la page earnings sont corrects
- [ ] Vérifier qu’aucun symbole \$ n’apparaît avec GBP/CHF/EUR

#### 4. Test de la Page des Frais

- [ ] Accéder à `/dashboard/creator/fees`
- [ ] Vérifier que la commission affichée est correcte (15%)
- [ ] Vérifier que le calculateur fonctionne correctement
- [ ] Vérifier que les liens vers les autres pages fonctionnent

#### 5. Test du Webhook Stripe

- [ ] Créer un nouveau compte Stripe Connect (ou mettre à jour)
- [ ] Vérifier que la devise est automatiquement mise à jour dans la DB
- [ ] Vérifier que la devise est correctement affichée dans le dashboard

### Tests Automatisés

Pour tester les fonctions utilitaires :

```
// Test formatCurrency
import { formatCurrency } from '@lib/currency-utils';

console.log(formatCurrency(100, 'GBP')); // "100.00 GBP" ✓
console.log(formatCurrency(500.50, 'EUR')); // "500.50 EUR" ✓
console.log(formatCurrency(1250, 'CHF')); // "1250.00 CHF" ✓

// Test getCreatorCurrency
import { getCreatorCurrency } from '@lib/stripe';

const currency = await getCreatorCurrency('creator_id');
console.log(currency); // "GBP" ou "EUR" ou "CHF" ✓
```

## Flux de Données

### Comment la Devise est Récupérée et Utilisée

```
graph TD
    A[Créateur crée compte Stripe] --> B[Webhook account.updated]
    B --> C[Récupération account.default_currency]
    C --> D[Mise à jour creator.currency dans DB]
    D --> E[Cache en DB]

    F[Page Dashboard] --> G[GET /api/auth/me]
    G --> H[Retourne user avec creator.currency]
    H --> I[Affichage avec devise correcte]

    E --> H
```

### Comment les Montants sont Affichés

```
graph TD
    A[Payment créé] --> B[payment.amount = montant brut]
    B --> C[payment.creatorAmount = montant après commission]

    D[Page Payments] --> E[Affiche payment.amount]
    E --> F[Utilisateur voit montant brut 100.00 GBP]

    G[Page Fees] --> H[Explique calculs détaillés]
    H --> I[Commission + Frais Stripe]
```

## Notes Importantes

### Comportement des Devises

1. **Fallback EUR** : Si la devise du créateur n'est pas disponible, on utilise EUR par défaut
2. **Mise à jour automatique** : La devise est mise à jour automatiquement via le webhook Stripe `account.updated`
3. **Cache DB** : La devise est stockée dans `creator.currency` pour éviter des appels API répétés



## Comportement des Montants

1. **Montants bruts affichés** : Les utilisateurs voient toujours le montant payé par le client (100 EUR)
2. **Calculs en interne** : La plateforme calcule en interne la commission et les frais Stripe
3. **Transparence** : La page `/dashboard/creator/fees` explique en détail les calculs

## Format d’Affichage

1. **Code ISO toujours** : On affiche toujours “500.00 GBP” au lieu de “£500.00” ou “\$500 GBP”
2. **Cohérence visuelle** : Tous les montants utilisent le même format partout
3. **Pas de symboles ambigus** : Plus de “\$” avec GBP/CHF/EUR



## Déploiement

### Étapes de Déploiement

1. **Vérifier les migrations DB** : Le champ `creator.currency` existe déjà dans le schéma Prisma
2. **Déployer le code** : Push des changements vers production
3. **Tester les webhooks** : S’assurer que les webhooks Stripe fonctionnent correctement
4. **Vérifier les pages** : Tester toutes les pages concernées

### Rollback si Nécessaire

En cas de problème :

1. Les anciens montants utilisent `creatorAmount` - possibilité de revenir en arrière
2. Les devises ont un fallback à ‘EUR’ - pas de crash
3. La page fees peut être désactivée sans impact sur le reste



## Maintenance Future

### Ajout d’une Nouvelle Devise

Pour supporter une nouvelle devise (ex: USD, CAD) :

1. Aucune modification de code nécessaire
2. La devise sera automatiquement récupérée depuis Stripe
3. Le formatage utilisera le code ISO (ex: “500.00 USD”)

### Modification de la Commission

Pour changer la commission de la plateforme :

1. Modifier via `/dashboard/admin/settings`
2. La page fees récupère automatiquement le nouveau taux
3. Aucune modification de code nécessaire

### Ajout de Frais Supplémentaires

Si besoin d’ajouter d’autres frais :

1. Modifier la page `/app/dashboard/creator/fees/page.tsx`
2. Ajouter les nouveaux frais dans le calculateur
3. Mettre à jour la documentation

---

## Checklist de Validation

---

- [x] Notifications simplifiées (uniquement montant brut)
  - [x] Devise récupérée depuis Stripe Connect
  - [x] Montants bruts affichés dans l'historique
  - [x] Page dédiée aux frais créée
  - [x] Formatage des devises corrigé (code ISO uniquement)
  - [x] Devise du créateur utilisée partout
  - [x] Webhook mise à jour pour synchroniser la devise
  - [x] Documentation créée et complète
- 

## Ressources Supplémentaires

---

- [Documentation Stripe Connect](https://stripe.com/docs/connect) (<https://stripe.com/docs/connect>)
  - [Documentation Prisma](https://www.prisma.io/docs/) (<https://www.prisma.io/docs/>)
  - [API Stripe Accounts](https://stripe.com/docs/api/accounts) (<https://stripe.com/docs/api/accounts>)
  - [Webhooks Stripe](https://stripe.com/docs/webhooks) (<https://stripe.com/docs/webhooks>)
- 

**Date de création :** 27 décembre 2025

**Version :** 1.0

**Auteur :** DeepAgent - Abacus.AI