🔧 Correction des Routes de Cron Next.js

📋 Résumé

Ce document explique les corrections apportées aux routes de cron de Call a Star pour résoudre l'erreur **"Dynamic server usage"** qui empêchait leur fonctionnement sur Vercel.

❌ Le Problème

Erreur Originale

```
Dynamic server usage: Route /api/cron/process-payouts couldn't be rendered statically
because it used request.headers
```

Explication Technique

Les routes de cron Next.js utilisaient `request.headers` pour vérifier le token de sécurité `CRON_SECRET`. Cependant, Next.js 13+ essaie par défaut de rendre les routes **statiquement** lors du build, ce qui est incompatible avec l'utilisation de `request.headers` (qui nécessite un rendu dynamique).

Le problème : Les routes de cron **ne doivent JAMAIS être rendues statiquement** car elles doivent s'exécuter dynamiquement à chaque appel.

Routes Affectées

1. `/api/cron/cleanup-logs` - Nettoyage automatique des logs
2. `/api/cron/process-payouts` - Traitement des payouts automatiques (mode horaire/CRON)
3. `/api/cron/process-automatic-payouts` - Traitement des payouts automatiques (mode DAILY/WEEKLY)

✅ La Solution

1. Force Dynamic Rendering

Ajout de directives d'export en haut de chaque fichier de route :

```
// Force dynamic rendering for cron routes (prevents static rendering errors)
export const dynamic = 'force-dynamic';
export const runtime = 'nodejs';
```

Explication :
- `export const dynamic = 'force-dynamic'` : Force Next.js à rendre la route **dynamiquement** à chaque requête (jamais en cache)
- `export const runtime = 'nodejs'` : Spécifie que la route doit s'exécuter dans l'environnement Node.js (pas Edge Runtime)

2. Amélioration de la Gestion d'Erreur

Chaque route de cron a été améliorée avec :

a) Vérification Robuste du CRON_SECRET

```
const cronSecret = process.env.CRON_SECRET;

if (!cronSecret) {
  await logSystemError(
    'CRON_CONFIG_ERROR',
    LogActor.SYSTEM,
    'CRON_SECRET non configuré',
    undefined,
    { endpoint: '/api/cron/...' }
  );
  return NextResponse.json({ error: 'Cron secret not configured' }, { status: 500 });
}

if (authHeader !== `Bearer ${cronSecret}`) {
  await logSystemError(
    'CRON_UNAUTHORIZED',
    LogActor.SYSTEM,
    'Tentative d\'accès avec un secret invalide',
    undefined,
    { endpoint: '/api/cron/...' }
  );
  return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
}
```

b) Logging Complet des Opérations

```
// Log au démarrage
await logSystem(
  'CRON_STARTED',
  'Cron démarré',
  LogLevel.INFO,
  { startTime: new Date().toISOString(), endpoint: '/api/cron/...' }
);

// Log à la fin
await logSystem(
  'CRON_COMPLETED',
  'Cron terminé avec succès',
  LogLevel.INFO,
  {
    endTime: new Date().toISOString(),
    durationMs: duration,
    summary: { ... }
  }
);
```

c) Try-Catch Global

Toutes les routes ont un try-catch global qui :
- Capture toutes les erreurs non gérées
- Log les erreurs fatales avec `logSystemError()`
- Retourne toujours une réponse HTTP appropriée (500)
- **Ne laisse jamais une erreur bloquer le système**

```
} catch (error: any) {
  await logSystemError(
    'CRON_FATAL_ERROR',
    LogActor.SYSTEM,
    `Erreur fatale: ${error.message}`,
    undefined,
    {
      errorMessage: error.message,
      errorStack: error.stack,
      endpoint: '/api/cron/...'
    }
  );

  return NextResponse.json(
    { error: 'Internal server error', message: error.message },
    { status: 500 }
  );
}
```

🧪 Comment Tester les Crons

Option 1 : Script de Test Automatisé (Recommandé)

Un script de test a été créé pour faciliter les tests : `scripts/test-cron-routes.ts`

```
# S'assurer que CRON_SECRET est configuré dans .env.local
echo "CRON_SECRET=your-secret-token-here" >> .env.local

# Installer les dépendances si nécessaire
npm install

# Exécuter le script de test
npm run dev  # Démarrer le serveur de développement dans un terminal
npx ts-node scripts/test-cron-routes.ts  # Dans un autre terminal
```

Le script teste :
- ✅ Accès autorisé avec le bon token
- ❌ Accès refusé avec un mauvais token
- 📊 Réponses de chaque endpoint

Option 2 : Test Manuel avec cURL

Test de cleanup-logs

```
# Avec authentification (devrait réussir)
curl -X POST http://localhost:3000/api/cron/cleanup-logs \
  -H "Authorization: Bearer YOUR_CRON_SECRET_HERE" \
  -H "Content-Type: application/json"

# Sans authentification (devrait échouer avec 401)
curl -X POST http://localhost:3000/api/cron/cleanup-logs \
  -H "Content-Type: application/json"
```

Test de process-payouts

```
# Avec authentification (devrait réussir)
curl -X GET "http://localhost:3000/api/cron/process-payouts" \
  -H "x-cron-secret: YOUR_CRON_SECRET_HERE"

# Sans authentification (devrait échouer avec 401)
curl -X GET "http://localhost:3000/api/cron/process-payouts"
```

Test de process-automatic-payouts

```
# Avec authentification (devrait réussir)
curl -X GET http://localhost:3000/api/cron/process-automatic-payouts \
  -H "Authorization: Bearer YOUR_CRON_SECRET_HERE"

# Sans authentification (devrait échouer avec 401)
curl -X GET http://localhost:3000/api/cron/process-automatic-payouts
```

Option 3 : Test avec Postman/Insomnia

1. Créer une nouvelle requête
2. Configurer l'URL : `http://localhost:3000/api/cron/[route-name]`
3. Ajouter le header d'authentification :
   - Pour cleanup-logs et process-automatic-payouts : `Authorization: Bearer YOUR_CRON_SECRET`
   - Pour process-payouts : `x-cron-secret: YOUR_CRON_SECRET`
4. Envoyer la requête

⚙️ Configuration pour Vercel

1. Variables d'Environnement

Dans le dashboard Vercel, ajouter la variable d'environnement :

```
CRON_SECRET=your-super-secret-token-here
```

Important : Utiliser un token sécurisé et aléatoire (ex: généré avec `openssl rand -hex 32`)

2. Configuration des Cron Jobs (vercel.json)

Créer ou mettre à jour le fichier `vercel.json` à la racine du projet :

```
{
  "crons": [
    {
      "path": "/api/cron/cleanup-logs",
      "schedule": "0 2 * * *"
    },
    {
      "path": "/api/cron/process-payouts",
      "schedule": "0 3 * * *"
    },
    {
      "path": "/api/cron/process-automatic-payouts",
      "schedule": "0 4 * * *"
    }
  ]
}
```

Schedules (format cron) :
- `0 2 * * *` = Tous les jours à 2h00 UTC
- `0 3 * * *` = Tous les jours à 3h00 UTC
- `0 4 * * *` = Tous les jours à 4h00 UTC

Note : Les crons Vercel envoient automatiquement le header d'authentification correct.

3. Vérification du Déploiement

Après le déploiement :

1. Vérifier que les variables d'environnement sont configurées
2. Consulter les logs dans Vercel Dashboard > Logs
3. Vérifier les logs système dans le dashboard admin (section Logs)

📊 Surveillance et Logs

Logs Système

Toutes les opérations de cron sont loggées dans le système de logging centralisé :

- **Événements SUCCESS** : `CRON_*_STARTED`, `CRON_*_COMPLETED`
- **Événements ERROR** : `CRON_*_CONFIG_ERROR`, `CRON_*_UNAUTHORIZED`, `CRON_*_FATAL_ERROR`

Consulter les Logs

1. Accéder au dashboard admin : `/dashboard/admin`
2. Aller dans la section "System Logs"
3. Filtrer par :
   - Type : `SYSTEM`
   - Rechercher : `CRON`

Logs à Surveiller

Event Type	Niveau	Description	Action Requise
`CRON_*_STARTED`	INFO	Cron démarré	✅ Normal
`CRON_*_COMPLETED`	INFO	Cron terminé avec succès	✅ Normal
`CRON_*_CONFIG_ERROR`	ERROR	CRON_SECRET non configuré	⚠️ Configurer CRON_SECRET
`CRON_*_UNAUTHORIZED`	ERROR	Tentative d'accès non autorisé	⚠️ Vérifier la sécurité
`CRON_*_FATAL_ERROR`	ERROR	Erreur fatale dans le cron	🚨 Investiguer immédiatement

---

🔒 Sécurité

Recommandations

1. **CRON_SECRET fort** : Utiliser un token long et aléatoire
   ```
   bash
   # Générer un token sécurisé
   openssl rand -hex 32
   ```
2. **Ne jamais exposer le secret** : Le secret ne doit jamais être commit dans Git ou exposé publiquement
3. **Rotation régulière** : Changer le secret périodiquement (ex: tous les 3-6 mois)
4. **Monitoring** : Surveiller les tentatives d'accès non autorisées dans les logs

En cas de Compromission

Si le `CRON_SECRET` est compromis :

1. Générer un nouveau secret immédiatement
2. Mettre à jour la variable d'environnement sur Vercel
3. Redéployer l'application
4. Vérifier les logs pour détecter des accès non autorisés

📝 Changements Apportés

Fichiers Modifiés

1. ✅ `/app/api/cron/cleanup-logs/route.ts`
   - Ajout de `export const dynamic = 'force-dynamic'`
   - Ajout de `export const runtime = 'nodejs'`

2. ✅ `/app/api/cron/process-payouts/route.ts`
   - Ajout de `export const dynamic = 'force-dynamic'`
   - Ajout de `export const runtime = 'nodejs'`

3. ✅ `/app/api/cron/process-automatic-payouts/route.ts`
   - Ajout de `export const dynamic = 'force-dynamic'`
   - Ajout de `export const runtime = 'nodejs'`
   - Amélioration de la vérification du CRON_SECRET
   - Ajout de logs au démarrage et à la fin
   - Amélioration du bloc try-catch global

Fichiers Créés

1. ✅ `/scripts/test-cron-routes.ts` - Script de test automatisé
2. ✅ `/CRON_FIXES.md` - Ce fichier de documentation

🚀 Prochaines Étapes

1. ✅ Déployer les changements sur Vercel
2. ✅ Configurer `CRON_SECRET` dans les variables d'environnement Vercel
3. ✅ Créer/mettre à jour `vercel.json` avec les schedules de cron
4. ✅ Vérifier que les crons s'exécutent correctement (consulter les logs)
5. ✅ Monitorer les logs système pour détecter les erreurs

📚 Ressources

- Next.js Dynamic Rendering (https://nextjs.org/docs/app/building-your-application/rendering/static-and-dynamic-rendering)
- Vercel Cron Jobs (https://vercel.com/docs/cron-jobs)
- Next.js Route Segment Config (https://nextjs.org/docs/app/api-reference/file-conventions/route-segment-config)

💡 Notes Importantes

- **Les routes de cron ne doivent JAMAIS être statiques** : Elles doivent être rendues dynamiquement à chaque appel
- **Toujours utiliser `force-dynamic`** : C'est la solution recommandée pour les routes API qui utilisent des headers/cookies

- **Logging complet** : Facilite le debugging et la surveillance en production
- **Gestion d’erreur robuste** : Empêche les erreurs fatales de bloquer le système

---

Date de création : 31 Décembre 2025
Dernière mise à jour : 31 Décembre 2025
Auteur : DeepAgent - Abacus.AI