Guide de corrections des erreurs de console pour MetaSign

Ce document liste les erreurs détectées dans la console du navigateur et propose des solutions pour les résoudre conformément aux bonnes pratiques de développement.

1. Erreur 404 sur l'API de notifications

Erreur détectée :

```
GET http://localhost:3000/api/notifications 404 (Not Found)
NotificationProvider.useEffect.fetchNotifications @ NotificationContext.tsx:25
```

Problème :
Le composant `NotificationProvider` tente d'accéder à une API qui n'existe pas, entraînant une erreur 404.

Solution :

1. Créer le point d'API manquant :

```typescript
// src/app/api/notifications/route.ts
import { NextResponse } from 'next/server';
import type { NextRequest } from 'next/server';

export async function GET(request: NextRequest) {
  // Implémenter la logique pour récupérer les notifications
  // Pour un développement initial, retourner un tableau vide est une solution correcte
  return NextResponse.json({
    notifications: [],
    timestamp: new Date().toISOString()
  });
}
```

2. Modifier le `NotificationContext.tsx` pour gérer les cas d'erreur :

typescript

```typescript
// Dans src/context/NotificationContext.tsx
const fetchNotifications = async () => {
  try {
    const response = await fetch('/api/notifications');

    // Vérifier si la réponse est OK
    if (!response.ok) {
      // Gérer l'erreur silencieusement en développement
      console.warn('Erreur lors de la récupération des notifications:', response.status);
      return [];
    }

    const data = await response.json();
    return data.notifications || [];
  } catch (error) {
    // Gérer les erreurs réseau silencieusement en développement
    console.warn('Erreur réseau lors de la récupération des notifications:', error);
    return [];
  }
};
```

2. Erreur JSON invalide

Erreur détectée :

```
Uncaught (in promise) SyntaxError: Unexpected token '<', "<!DOCTYPE "... is not valid JSON
```

Problème :
Cette erreur indique qu'une réponse HTML est reçue lorsqu'une réponse JSON est attendue. Cela se produit généralement dans l'un des cas suivants :

- Une requête à l'API redirige vers une page HTML (par exemple, une page d'erreur)
- Un point d'API renvoie un format incorrect
- Une erreur d'authentification redirige vers une page de connexion

Solutions :

1. Vérifier et corriger les appels API :

typescript

```typescript
// Modèle pour toutes les fonctions d'appel API
async function apiCall(url: string, options?: RequestInit) {
  try {
    const response = await fetch(url, {
      ...options,
      headers: {
        'Content-Type': 'application/json',
        ...(options?.headers || {})
      }
    });

    // Vérifier si la réponse est au format JSON
    const contentType = response.headers.get('content-type');
    if (!contentType || !contentType.includes('application/json')) {
      console.error('Réponse non-JSON reçue:', contentType);
      throw new Error(`Réponse non-JSON reçue: ${contentType}`);
    }

    // Traiter la réponse JSON
    return await response.json();
  } catch (error) {
    console.error('Erreur API:', error);
    // Gérer l'erreur de manière appropriée
    throw error;
  }
}
```

2. Gérer les états d'authentification :

typescript

```typescript
// Dans les composants qui font des appels API
const [isAuthenticated, setIsAuthenticated] = useState<boolean>(false);

useEffect(() => {
  // Vérifier l'état d'authentification au chargement
  const checkAuth = async () => {
    try {
      const response = await fetch('/api/auth/session');

      if (!response.ok) {
        // Rediriger vers la page de connexion si nécessaire
        setIsAuthenticated(false);
        return;
      }

      const session = await response.json();
      setIsAuthenticated(!!session.user);
    } catch (error) {
      console.error('Erreur lors de la vérification de l\'authentification:', error);
      setIsAuthenticated(false);
    }
  };

  checkAuth();
}, []);

// Conditionnez les appels API à l'état d'authentification
useEffect(() => {
  if (isAuthenticated) {
    // Faire les appels API sécurisés ici
  }
}, [isAuthenticated]);
```

3. Bonnes pratiques pour les appels API

Pour éviter ces erreurs à l'avenir, voici quelques recommandations :

1. **Centraliser la logique d'API :**

typescript

```typescript
// src/services/api.ts
interface ApiResponse<T> {
  data: T | null;
  error: string | null;
  status: number;
}

export async function fetchData<T>(url: string, options?: RequestInit): Promise<ApiResponse<T>>
  try {
    const response = await fetch(url, {
      ...options,
      headers: {
        'Content-Type': 'application/json',
        ...(options?.headers || {})
      }
    });

    // Gestion des réponses non-JSON
    const contentType = response.headers.get('content-type');
    if (contentType && contentType.includes('application/json')) {
      const data = await response.json();
      return {
        data: response.ok ? data : null,
        error: response.ok ? null : data.message || 'Une erreur est survenue',
        status: response.status
      };
    }

    // Réponse non-JSON
    return {
      data: null,
      error: `Réponse non-JSON reçue (${response.status})`,
      status: response.status
    };
  } catch (error) {
    console.error('Erreur réseau:', error);
    return {
      data: null,
      error: error instanceof Error ? error.message : 'Erreur réseau inconnue',
      status: 0 // Code d'erreur réseau
    };
  }
}
```

2. **Créer des hooks personnalisés pour les fonctionnalités communes :**

typescript

```typescript
// src/hooks/useApi.ts
import { useState, useEffect } from 'react';
import { fetchData } from '@/services/api';

export function useApi<T>(url: string, options?: RequestInit) {
  const [data, setData] = useState<T | null>(null);
  const [error, setError] = useState<string | null>(null);
  const [loading, setLoading] = useState<boolean>(true);

  useEffect(() => {
    let isMounted = true;

    const fetchApi = async () => {
      setLoading(true);
      const response = await fetchData<T>(url, options);

      if (isMounted) {
        setData(response.data);
        setError(response.error);
        setLoading(false);
      }
    };

    fetchApi();

    return () => {
      isMounted = false;
    };
  }, [url, JSON.stringify(options)]);

  return { data, error, loading };
}
```

3. **Implémenter la gestion des erreurs globale :**

typescript

```typescript
// src/context/ErrorContext.tsx
import React, { createContext, useContext, useState, ReactNode } from 'react';

interface ErrorContextType {
  error: string | null;
  setError: (error: string | null) => void;
  clearError: () => void;
}

const ErrorContext = createContext<ErrorContextType | undefined>(undefined);

export function ErrorProvider({ children }: { children: ReactNode }) {
  const [error, setError] = useState<string | null>(null);

  const clearError = () => setError(null);

  return (
    <ErrorContext.Provider value={{ error, setError, clearError }}>
      {error && (
        <div className="error-toast">
          <p>{error}</p>
          <button onClick={clearError}>Fermer</button>
        </div>
      )}
      {children}
    </ErrorContext.Provider>
  );
}

export function useError() {
  const context = useContext(ErrorContext);
  if (context === undefined) {
    throw new Error('useError doit être utilisé dans un ErrorProvider');
  }
  return context;
}
```

4. **Utiliser un intercepteur pour les appels fetch :**

Pour une solution plus complète, envisagez d'implémenter un intercepteur qui gère automatiquement les erreurs et l'authentification :

typescript

```typescript
// src/utils/fetchInterceptor.ts
const originalFetch = window.fetch;

window.fetch = async function(input: RequestInfo, init?: RequestInit) {
  try {
    // Ajouter des en-têtes par défaut
    const modifiedInit = {
      ...init,
      headers: {
        'Content-Type': 'application/json',
        ...(init?.headers || {})
      }
    };

    // Effectuer l'appel fetch d'origine
    const response = await originalFetch(input, modifiedInit);

    // Gérer les erreurs d'authentification
    if (response.status === 401) {
      // Rediriger vers la page de connexion ou rafraîchir le token
      console.warn('Session expirée, redirection vers la page de connexion');
      window.location.href = '/login';
      return response;
    }

    // Gérer les autres erreurs HTTP
    if (!response.ok) {
      console.warn(`Erreur HTTP ${response.status} lors de l'appel à ${input}`);
    }

    return response;
  } catch (error) {
    console.error('Erreur réseau interceptée:', error);
    throw error;
  }
};
```

Implémentation pratique pour NotificationContext.tsx

Voici une implémentation améliorée du `NotificationContext.tsx` qui résout les problèmes identifiés :

typescript

```tsx
// src/context/NotificationContext.tsx
import { createContext, useContext, useState, useEffect, ReactNode } from 'react';
import { fetchData } from '@/services/api';

// Types
interface Notification {
  id: string;
  message: string;
  type: 'info' | 'success' | 'warning' | 'error';
  timestamp: string;
  read: boolean;
}

interface NotificationContextType {
  notifications: Notification[];
  unreadCount: number;
  markAsRead: (id: string) => void;
  markAllAsRead: () => void;
  addNotification: (notification: Omit<Notification, 'id' | 'timestamp' | 'read'>) => void;
}

// Création du contexte
const NotificationContext = createContext<NotificationContextType | undefined>(undefined);

// Provider
export function NotificationProvider({ children }: { children: ReactNode }) {
  const [notifications, setNotifications] = useState<Notification[]>([]);
  const [loading, setLoading] = useState<boolean>(true);
  const [error, setError] = useState<string | null>(null);

  // Calculer le nombre de notifications non lues
  const unreadCount = notifications.filter(notification => !notification.read).length;

  // Fonction pour récupérer les notifications
  const fetchNotifications = async () => {
    setLoading(true);
    try {
      const response = await fetchData<{ notifications: Notification[] }>('/api/notifications')

      if (response.data) {
        setNotifications(response.data.notifications || []);
      } else if (response.error) {
        setError(response.error);
        // En développement, utiliser des données fictives pour éviter les erreurs dans l'UI
        if (process.env.NODE_ENV === 'development') {
          setNotifications([]);
```

```
      }
    }
  } catch (error) {
    setError(error instanceof Error ? error.message : 'Erreur inconnue');
    if (process.env.NODE_ENV === 'development') {
      setNotifications([]);
    }
  } finally {
    setLoading(false);
  }
};

// Charger les notifications au montage du composant
useEffect(() => {
  fetchNotifications();

  // Configurer un intervalle pour rafraîchir les notifications
  const intervalId = setInterval(fetchNotifications, 60000); // 1 minute

  // Nettoyer l'intervalle au démontage
  return () => clearInterval(intervalId);
}, []);

// Marquer une notification comme lue
const markAsRead = async (id: string) => {
  try {
    // Optimistic update
    setNotifications(prev =>
      prev.map(notification =>
        notification.id === id
          ? { ...notification, read: true }
          : notification
      )
    );

    // Appel API pour mettre à jour côté serveur
    await fetchData('/api/notifications/mark-read', {
      method: 'POST',
      body: JSON.stringify({ id })
    });
  } catch (error) {
    console.error('Erreur lors du marquage de la notification:', error);
    // Réversion en cas d'échec
    await fetchNotifications();
  }
};
```

```
// Marquer toutes les notifications comme lues
const markAllAsRead = async () => {
  try {
    // Optimistic update
    setNotifications(prev =>
      prev.map(notification => ({ ...notification, read: true }))
    );

    // Appel API pour mettre à jour côté serveur
    await fetchData('/api/notifications/mark-all-read', {
      method: 'POST'
    });
  } catch (error) {
    console.error('Erreur lors du marquage de toutes les notifications:', error);
    // Réversion en cas d'échec
    await fetchNotifications();
  }
};

// Ajouter une nouvelle notification (locale uniquement)
const addNotification = (notification: Omit<Notification, 'id' | 'timestamp' | 'read'>) => {
  const newNotification: Notification = {
    ...notification,
    id: `local-${Date.now()}-${Math.random().toString(36).substring(2, 9)}`,
    timestamp: new Date().toISOString(),
    read: false
  };

  setNotifications(prev => [newNotification, ...prev]);
};

// Valeur du contexte
const contextValue: NotificationContextType = {
  notifications,
  unreadCount,
  markAsRead,
  markAllAsRead,
  addNotification
};

return (
  <NotificationContext.Provider value={contextValue}>
    {loading && process.env.NODE_ENV === 'development' && <div hidden>Chargement des notifica
    {error && process.env.NODE_ENV === 'development' && <div hidden>Erreur: {error}</div>}
    {children}
  </NotificationContext.Provider>
);
```

```
}

// Hook personnalisé
export function useNotifications() {
  const context = useContext(NotificationContext);
  if (context === undefined) {
    throw new Error('useNotifications doit être utilisé dans un NotificationProvider');
  }
  return context;
}
```

Ces solutions devraient résoudre les erreurs de console tout en améliorant la robustesse de l'application.