Troubleshooting - DMCA Guard Platform

Guia completo para resolução de problemas comuns na plataforma DMCA Guard, incluindo diagnósticos, soluções e prevenção.

Problemas Críticos

1. Aplicação Não Inicia

Sintomas

```
❌ Erro ao iniciar aplicação:
• "Cannot connect to database"
• "Port 3000 already in use"
• "Module not found"
• "Environment variables missing"
```

Diagnóstico

```bash
# Verificar status dos serviços
systemctl status postgresql
systemctl status redis
systemctl status nginx

# Verificar portas em uso
netstat -tulpn | grep :3000
lsof -i :3000

# Verificar logs
tail -f logs/application.log
journalctl -u dmca-guard -f

# Verificar variáveis de ambiente
env | grep -E "(DATABASE_URL|NEXTAUTH_SECRET|OPENAI_API_KEY)"
```

Soluções

Problema de Banco de Dados:

```bash
# Verificar conexão
psql $DATABASE_URL -c "SELECT version();"

# Reiniciar PostgreSQL
sudo systemctl restart postgresql

# Verificar configuração
cat /etc/postgresql/*/main/postgresql.conf | grep listen_addresses
```

Porta em Uso:

```
# Encontrar processo usando a porta
sudo lsof -i :3000

# Matar processo
sudo kill -9 $(lsof -t -i:3000)

# Usar porta alternativa
PORT=3001 yarn dev
```

Módulos Ausentes:

```
# Limpar cache e reinstalar
rm -rf node_modules package-lock.json yarn.lock
yarn install

# Verificar versão do Node.js
node --version  # Deve ser 18+
```

Variáveis de Ambiente:

```
# Verificar arquivo .env
ls -la .env*
cat .env.local

# Copiar exemplo
cp .env.example .env.local
nano .env.local
```

2. Erro 500 - Internal Server Error

Sintomas

```
  Erro 500 em produção:
• Páginas retornam erro interno
• APIs não respondem
• Logs mostram stack traces
```

Diagnóstico

```
# Verificar logs detalhados
tail -f logs/error.log
grep "ERROR" logs/application.log | tail -20

# Verificar uso de recursos
htop
df -h
free -m

# Verificar conexões de banco
psql $DATABASE_URL -c "SELECT count(*) FROM pg_stat_activity;"
```

Soluções

Erro de Memória:

```
# Aumentar limite de memória Node.js
export NODE_OPTIONS="--max-old-space-size=4096"

# Verificar memory leaks
node --inspect app.js
# Abrir chrome://inspect
```

Erro de Banco:

```
# Verificar conexões ativas
psql $DATABASE_URL -c "
SELECT pid, usename, application_name, state
FROM pg_stat_activity
WHERE state = 'active';
"

# Matar conexões órfãs
psql $DATABASE_URL -c "
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
WHERE state = 'idle' AND state_change < now() - interval '1 hour';
"
```

Erro de Prisma:

```
# Regenerar cliente Prisma
npx prisma generate

# Verificar schema
npx prisma db pull
npx prisma migrate status

# Reset do banco (CUIDADO!)
npx prisma migrate reset
```

3. APIs Externas Falhando

Sintomas

```
  Falhas de API:
• OpenAI retorna 429 (Rate Limit)
• SendGrid retorna 401 (Unauthorized)
• Timeouts frequentes
```

Diagnóstico

```
# Testar APIs manualmente
curl -H "Authorization: Bearer $OPENAI_API_KEY" \
     https://api.openai.com/v1/models

curl -H "Authorization: Bearer $SENDGRID_API_KEY" \
     https://api.sendgrid.com/v3/user/profile

# Verificar logs de API
grep "API_ERROR" logs/application.log
```

Soluções

Rate Limiting OpenAI:

```
// lib/openai-retry.js
import { OpenAI } from 'openai';

const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY,
  maxRetries: 3,
  timeout: 30000,
});

export async function callOpenAIWithRetry(prompt, retries = 3) {
  for (let i = 0; i < retries; i++) {
    try {
      const response = await openai.chat.completions.create({
        model: 'gpt-3.5-turbo',
        messages: [{ role: 'user', content: prompt }],
      });
      return response;
    } catch (error) {
      if (error.status === 429 && i < retries - 1) {
        const delay = Math.pow(2, i) * 1000; // Exponential backoff
        await new Promise(resolve => setTimeout(resolve, delay));
        continue;
      }
      throw error;
    }
  }
}
```

SendGrid Authentication:

```
# Verificar chave API
echo $SENDGRID_API_KEY | cut -c1-10

# Testar com curl
curl -X POST https://api.sendgrid.com/v3/mail/send \
  -H "Authorization: Bearer $SENDGRID_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "personalizations": [{"to": [{"email": "test@example.com"}]}],
    "from": {"email": "noreply@yourdomain.com"},
    "subject": "Test",
    "content": [{"type": "text/plain", "value": "Test"}]
  }'
```

Problemas de Monitoramento

1. Detecção Não Funciona

Sintomas

```
  Problemas de detecção:
• Sessões ficam "em progresso" indefinidamente
• Nenhum conteúdo é detectado
• Muitos falsos positivos
• Erro "Scraping failed"
```

Diagnóstico

```
# Verificar logs de scraping
grep "SCRAPING" logs/application.log | tail -20

# Testar conectividade
curl -I https://pornhub.com
curl -I https://xvideos.com

# Verificar proxies
curl --proxy $PROXY_URL -I https://pornhub.com
```

Soluções

Bloqueio por IP:

```
// lib/scraping/proxy-rotation.js
const proxies = [
  'http://proxy1:8080',
  'http://proxy2:8080',
  'http://proxy3:8080'
];

let currentProxyIndex = 0;

export function getNextProxy() {
  const proxy = proxies[currentProxyIndex];
  currentProxyIndex = (currentProxyIndex + 1) % proxies.length;
  return proxy;
}

export async function scrapeWithProxy(url) {
  const proxy = getNextProxy();

  try {
    const response = await fetch(url, {
      agent: new HttpsProxyAgent(proxy),
      headers: {
        'User-Agent': getRandomUserAgent(),
      }
    });
    return response;
  } catch (error) {
    console.error(`Proxy ${proxy} failed:`, error);
    throw error;
  }
}
```

Rate Limiting:

```
// lib/scraping/rate-limiter.js
export class ScrapingRateLimiter {
  constructor(requestsPerMinute = 10) {
    this.requests = [];
    this.limit = requestsPerMinute;
  }

  async waitIfNeeded() {
    const now = Date.now();
    const oneMinuteAgo = now - 60000;

    // Remove requests older than 1 minute
    this.requests = this.requests.filter(time => time > oneMinuteAgo);

    if (this.requests.length >= this.limit) {
      const oldestRequest = Math.min(...this.requests);
      const waitTime = 60000 - (now - oldestRequest);

      if (waitTime > 0) {
        await new Promise(resolve => setTimeout(resolve, waitTime));
      }
    }

    this.requests.push(now);
  }
}
```

Falsos Positivos:

```
// lib/detection/similarity-check.js
export function improvedSimilarityCheck(originalImage, detectedImage) {
  // Usar múltiplos algoritmos
  const hashSimilarity = compareHashes(originalImage, detectedImage);
  const featureSimilarity = compareFeatures(originalImage, detectedImage);
  const facialSimilarity = compareFaces(originalImage, detectedImage);

  // Peso combinado
  const combinedScore = (
    hashSimilarity * 0.3 +
    featureSimilarity * 0.4 +
    facialSimilarity * 0.3
  );

  return {
    score: combinedScore,
    confidence: calculateConfidence([hashSimilarity, featureSimilarity, facialSimilar-
ity]),
    breakdown: {
      hash: hashSimilarity,
      features: featureSimilarity,
      facial: facialSimilarity
    }
  };
}
```

2. Sessões Travadas

Sintomas

```
  Sessões não finalizam:
• Status "running" por horas
• Progresso parado em X%
• Timeout errors nos logs
```

Diagnóstico

```sql
-- Verificar sessões ativas
SELECT
  id,
  status,
  created_at,
  updated_at,
  EXTRACT(EPOCH FROM (NOW() - updated_at))/60 as minutes_stuck
FROM monitoring_sessions
WHERE status = 'running'
ORDER BY created_at DESC;
```

Soluções

Timeout de Sessões:

```
// lib/monitoring/session-manager.js
export class SessionManager {
  static async cleanupStuckSessions() {
    const stuckSessions = await prisma.monitoringSession.findMany({
      where: {
        status: 'running',
        updatedAt: {
          lt: new Date(Date.now() - 30 * 60 * 1000) // 30 minutos
        }
      }
    });

    for (const session of stuckSessions) {
      await this.forceCompleteSession(session.id);
    }
  }

  static async forceCompleteSession(sessionId) {
    await prisma.monitoringSession.update({
      where: { id: sessionId },
      data: {
        status: 'completed',
        error: 'Session timed out and was force completed',
        completedAt: new Date()
      }
    });
  }
}

// Executar limpeza a cada 15 minutos
setInterval(SessionManager.cleanupStuckSessions, 15 * 60 * 1000);
```

Retry Mechanism:

```
// lib/monitoring/retry-logic.js
export async function executeWithRetry(operation, maxRetries = 3) {
  for (let attempt = 1; attempt <= maxRetries; attempt++) {
    try {
      return await operation();
    } catch (error) {
      console.error(`Attempt ${attempt} failed:`, error);

      if (attempt === maxRetries) {
        throw new Error(`Operation failed after ${maxRetries} attempts: $
{error.message}`);
      }

      // Exponential backoff
      const delay = Math.pow(2, attempt) * 1000;
      await new Promise(resolve => setTimeout(resolve, delay));
    }
  }
}
```

Problemas de DMCA

1. Emails Não Enviados

Sintomas

```
  Falhas no envio:
• Status "failed" nas notificações
• Erro "Authentication failed"
• Emails na fila por muito tempo
```

Diagnóstico

```
# Verificar fila de emails
redis-cli LLEN email_queue

# Verificar logs do SendGrid
grep "SENDGRID" logs/application.log | tail -20

# Testar configuração
node -e "
const sgMail = require('@sendgrid/mail');
sgMail.setApiKey(process.env.SENDGRID_API_KEY);
console.log('SendGrid configured');
"
```

Soluções

Configuração SendGrid:

```
// lib/email/sendgrid-config.js
import sgMail from '@sendgrid/mail';

// Configuração robusta
sgMail.setApiKey(process.env.SENDGRID_API_KEY);

export async function sendEmailWithRetry(emailData, maxRetries = 3) {
  for (let attempt = 1; attempt <= maxRetries; attempt++) {
    try {
      const response = await sgMail.send(emailData);

      // Log sucesso
      console.log(`Email sent successfully on attempt ${attempt}`);
      return response;

    } catch (error) {
      console.error(`Email attempt ${attempt} failed:`, error);

      if (attempt === maxRetries) {
        // Salvar na fila para retry posterior
        await saveToRetryQueue(emailData, error);
        throw error;
      }

      // Aguardar antes do próximo attempt
      await new Promise(resolve => setTimeout(resolve, attempt * 2000));
    }
  }
}

async function saveToRetryQueue(emailData, error) {
  await prisma.emailRetryQueue.create({
    data: {
      emailData: JSON.stringify(emailData),
      error: error.message,
      attempts: 0,
      nextRetry: new Date(Date.now() + 60000) // 1 minuto
    }
  });
}
```

Processamento de Fila:

```
// lib/email/queue-processor.js
export class EmailQueueProcessor {
  static async processRetryQueue() {
    const retryEmails = await prisma.emailRetryQueue.findMany({
      where: {
        nextRetry: { lte: new Date() },
        attempts: { lt: 5 }
      }
    });

    for (const retryEmail of retryEmails) {
      try {
        const emailData = JSON.parse(retryEmail.emailData);
        await sendEmailWithRetry(emailData, 1);

        // Remover da fila após sucesso
        await prisma.emailRetryQueue.delete({
          where: { id: retryEmail.id }
        });

      } catch (error) {
        // Incrementar tentativas
        await prisma.emailRetryQueue.update({
          where: { id: retryEmail.id },
          data: {
            attempts: retryEmail.attempts + 1,
            nextRetry: new Date(Date.now() + Math.pow(2, retryEmail.attempts) * 60000),
            lastError: error.message
          }
        });
      }
    }
  }
}

// Processar fila a cada 5 minutos
setInterval(EmailQueueProcessor.processRetryQueue, 5 * 60 * 1000);
```

2. Templates Malformados

Sintomas

```
  Problemas de template:
• Variáveis não substituídas: {{nome}}
• HTML quebrado
• Caracteres especiais corrompidos
```

Soluções

Validação de Template:

```
// lib/email/template-validator.js
export function validateTemplate(template, variables) {
  const errors = [];

  // Verificar variáveis obrigatórias
  const requiredVars = ['creatorName', 'platformName', 'infringingUrl'];
  for (const varName of requiredVars) {
    if (!variables[varName]) {
      errors.push(`Missing required variable: ${varName}`);
    }
  }

  // Verificar sintaxe do template
  const variablePattern = /\{\{(\w+)\}\}/g;
  let match;
  while ((match = variablePattern.exec(template)) !== null) {
    const varName = match[1];
    if (!variables[varName]) {
      errors.push(`Template uses undefined variable: ${varName}`);
    }
  }

  // Verificar HTML válido
  if (template.includes('<') && !isValidHTML(template)) {
    errors.push('Invalid HTML in template');
  }

  return {
    isValid: errors.length === 0,
    errors
  };
}

function isValidHTML(html) {
  try {
    const parser = new DOMParser();
    const doc = parser.parseFromString(html, 'text/html');
    return !doc.querySelector('parsererror');
  } catch {
    return false;
  }
}
```

Renderização Segura:

```
// lib/email/template-renderer.js
export function renderTemplate(template, variables) {
  // Escapar variáveis para prevenir XSS
  const escapedVars = {};
  for (const [key, value] of Object.entries(variables)) {
    escapedVars[key] = escapeHtml(String(value));
  }

  // Substituir variáveis
  let rendered = template;
  for (const [key, value] of Object.entries(escapedVars)) {
    const regex = new RegExp(`\\{\\{${key}\\}\\}`, 'g');
    rendered = rendered.replace(regex, value);
  }

  // Verificar se restaram variáveis não substituídas
  const unresolved = rendered.match(/\{\{\w+\}\}/g);
  if (unresolved) {
    console.warn('Unresolved template variables:', unresolved);
  }

  return rendered;
}

function escapeHtml(text) {
  const map = {
    '&': '&amp;',
    '<': '&lt;',
    '>': '&gt;',
    '"': '&quot;',
    "'": '&#039;'
  };
  return text.replace(/[&<>"']/g, m => map[m]);
}
```

Problemas de Autenticação

1. Login Não Funciona

Sintomas

```
  Falhas de login:
• "Invalid credentials" para senhas corretas
• Redirecionamento infinito
• Sessão expira imediatamente
```

Diagnóstico

```
# Verificar configuração NextAuth
grep "NEXTAUTH" .env.local

# Verificar logs de autenticação
grep "AUTH" logs/application.log | tail -20

# Testar hash de senha
node -e "
const bcrypt = require('bcrypt');
console.log(bcrypt.compareSync('password123', 'hash_from_db'));
"
```

Soluções

Configuração NextAuth:

```
// app/api/auth/[...nextauth]/route.js
import NextAuth from 'next-auth';
import CredentialsProvider from 'next-auth/providers/credentials';
import bcrypt from 'bcrypt';

const handler = NextAuth({
  providers: [
    CredentialsProvider({
      name: 'credentials',
      credentials: {
        email: { label: 'Email', type: 'email' },
        password: { label: 'Password', type: 'password' }
      },
      async authorize(credentials) {
        try {
          const user = await prisma.user.findUnique({
            where: { email: credentials.email }
          });

          if (!user) {
            console.log('User not found:', credentials.email);
            return null;
          }

          const isValid = await bcrypt.compare(credentials.password, user.password);

          if (!isValid) {
            console.log('Invalid password for:', credentials.email);
            return null;
          }

          return {
            id: user.id,
            email: user.email,
            name: user.name
          };
        } catch (error) {
          console.error('Auth error:', error);
          return null;
        }
      }
    })
  ],
  session: {
    strategy: 'jwt',
    maxAge: 30 * 24 * 60 * 60, // 30 dias
  },
  callbacks: {
    async jwt({ token, user }) {
      if (user) {
        token.id = user.id;
      }
      return token;
    },
    async session({ session, token }) {
      session.user.id = token.id;
      return session;
    }
```

```
  },
  pages: {
    signIn: '/auth/signin',
    error: '/auth/error'
  },
  debug: process.env.NODE_ENV === 'development'
});

export { handler as GET, handler as POST };
```

Reset de Senha:

```
// lib/auth/password-reset.js
export async function resetPassword(email) {
  const user = await prisma.user.findUnique({
    where: { email }
  });

  if (!user) {
    // Não revelar se o email existe
    return { success: true };
  }

  const resetToken = crypto.randomBytes(32).toString('hex');
  const resetExpires = new Date(Date.now() + 60 * 60 * 1000); // 1 hora

  await prisma.user.update({
    where: { id: user.id },
    data: {
      resetToken,
      resetExpires
    }
  });

  await sendPasswordResetEmail(email, resetToken);

  return { success: true };
}

export async function confirmPasswordReset(token, newPassword) {
  const user = await prisma.user.findFirst({
    where: {
      resetToken: token,
      resetExpires: { gt: new Date() }
    }
  });

  if (!user) {
    throw new Error('Token inválido ou expirado');
  }

  const hashedPassword = await bcrypt.hash(newPassword, 12);

  await prisma.user.update({
    where: { id: user.id },
    data: {
      password: hashedPassword,
      resetToken: null,
      resetExpires: null
    }
  });

  return { success: true };
}
```

2. Sessões Expiram Rapidamente

Sintomas

```
  Problemas de sessão:
• Logout automático frequente
• "Session expired" em poucos minutos
• Perda de estado da aplicação
```

Soluções

Configuração de Sessão:

```
// lib/auth/session-config.js
export const sessionConfig = {
  strategy: 'jwt',
  maxAge: 30 * 24 * 60 * 60, // 30 dias
  updateAge: 24 * 60 * 60, // Atualizar a cada 24h

  // Configurações de cookie
  cookies: {
    sessionToken: {
      name: 'next-auth.session-token',
      options: {
        httpOnly: true,
        sameSite: 'lax',
        path: '/',
        secure: process.env.NODE_ENV === 'production',
        maxAge: 30 * 24 * 60 * 60 // 30 dias
      }
    }
  }
};
```

Auto-refresh de Token:

```
// hooks/useAuthRefresh.js
import { useSession } from 'next-auth/react';
import { useEffect } from 'react';

export function useAuthRefresh() {
  const { data: session, update } = useSession();

  useEffect(() => {
    if (!session) return;

    const refreshInterval = setInterval(async () => {
      try {
        await update(); // Refresh da sessão
      } catch (error) {
        console.error('Failed to refresh session:', error);
      }
    }, 15 * 60 * 1000); // A cada 15 minutos

    return () => clearInterval(refreshInterval);
  }, [session, update]);
}
```

Problemas de Banco de Dados

1. Queries Lentas

Sintomas

```
Performance ruim:
• Páginas carregam lentamente
• Timeouts em operações
• Alto uso de CPU no banco
```

Diagnóstico

```
-- Verificar queries lentas
SELECT
  query,
  calls,
  total_time,
  mean_time,
  rows
FROM pg_stat_statements
ORDER BY total_time DESC
LIMIT 10;

-- Verificar locks
SELECT
  blocked_locks.pid AS blocked_pid,
  blocked_activity.usename AS blocked_user,
  blocking_locks.pid AS blocking_pid,
  blocking_activity.usename AS blocking_user,
  blocked_activity.query AS blocked_statement,
  blocking_activity.query AS current_statement_in_blocking_process
FROM pg_catalog.pg_locks blocked_locks
JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid = blocked_loc
ks.pid
JOIN pg_catalog.pg_locks blocking_locks ON blocking_locks.locktype = blocked_locks.lock
type
JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid = block-
ing_locks.pid
WHERE NOT blocked_locks.granted;
```

Soluções

Otimização de Queries:

```
-- Adicionar índices necessários
CREATE INDEX CONCURRENTLY idx_monitoring_sessions_user_status
ON monitoring_sessions(user_id, status);

CREATE INDEX CONCURRENTLY idx_detected_content_created_at
ON detected_content(created_at DESC);

CREATE INDEX CONCURRENTLY idx_dmca_requests_status_created
ON dmca_requests(status, created_at);

-- Analisar planos de execução
EXPLAIN ANALYZE SELECT * FROM monitoring_sessions
WHERE user_id = $1 AND status = 'active';
```

Connection Pooling:

```
// lib/database/connection-pool.js
import { PrismaClient } from '@prisma/client';

const globalForPrisma = globalThis;

export const prisma = globalForPrisma.prisma || new PrismaClient({
  datasources: {
    db: {
      url: process.env.DATABASE_URL
    }
  },
  log: process.env.NODE_ENV === 'development' ? ['query', 'error', 'warn'] : ['error'],
});

if (process.env.NODE_ENV !== 'production') {
  globalForPrisma.prisma = prisma;
}

// Configurar connection pool
export const poolConfig = {
  connectionLimit: 20,
  acquireTimeoutMillis: 60000,
  createTimeoutMillis: 30000,
  destroyTimeoutMillis: 5000,
  idleTimeoutMillis: 30000,
  reapIntervalMillis: 1000,
  createRetryIntervalMillis: 200,
};
```

2. Deadlocks

Sintomas

```
  Deadlocks frequentes:
• "deadlock detected" nos logs
• Transações que falham aleatoriamente
• Timeouts em operações simultâneas
```

Soluções

Retry com Backoff:

```
// lib/database/deadlock-retry.js
export async function executeWithDeadlockRetry(operation, maxRetries = 3) {
  for (let attempt = 1; attempt <= maxRetries; attempt++) {
    try {
      return await operation();
    } catch (error) {
      if (error.code === 'P2034' && attempt < maxRetries) { // Prisma deadlock code
        const delay = Math.random() * Math.pow(2, attempt) * 100;
        await new Promise(resolve => setTimeout(resolve, delay));
        continue;
      }
      throw error;
    }
  }
}

// Uso
await executeWithDeadlockRetry(async () => {
  return await prisma.$transaction([
    prisma.user.update({ where: { id: userId }, data: { ... } }),
    prisma.subscription.create({ data: { ... } })
  ]);
});
```

Ordem Consistente de Locks:

```
// lib/database/lock-ordering.js
export async function updateUserAndSubscription(userId, subscriptionData) {
  // Sempre fazer operações na mesma ordem para evitar deadlocks
  return await prisma.$transaction(async (tx) => {
    // 1. Primeiro, operações em user (menor ID de tabela)
    const user = await tx.user.findUnique({
      where: { id: userId }
    });

    // 2. Depois, operações em subscription
    const subscription = await tx.subscription.upsert({
      where: { userId },
      create: { userId, ...subscriptionData },
      update: subscriptionData
    });

    return { user, subscription };
  });
}
```

Ferramentas de Diagnóstico

1. Health Check Completo

```
#!/bin/bash
# scripts/health-check.sh

echo "  DMCA Guard - Health Check"
echo "================================="

# Verificar aplicação
echo "  Aplicação:"
if curl -f http://localhost:3000/api/health > /dev/null 2>&1; then
    echo "    Aplicação respondendo"
else
    echo "    Aplicação não responde"
fi

# Verificar banco de dados
echo "  Banco de Dados:"
if pg_isready -h localhost -p 5432 > /dev/null 2>&1; then
    echo "    PostgreSQL ativo"

    # Verificar conexões
    CONNECTIONS=$(psql $DATABASE_URL -t -c "SELECT count(*) FROM pg_stat_activity;")
    echo "    Conexões ativas: $CONNECTIONS"

    if [ "$CONNECTIONS" -gt 50 ]; then
        echo "  ⚠ Muitas conexões ativas"
    fi
else
    echo "    PostgreSQL não responde"
fi

# Verificar Redis
echo "  Redis:"
if redis-cli ping > /dev/null 2>&1; then
    echo "    Redis ativo"

    MEMORY=$(redis-cli info memory | grep used_memory_human | cut -d: -f2)
    echo "    Memória usada: $MEMORY"
else
    echo "    Redis não responde"
fi

# Verificar APIs externas
echo "  APIs Externas:"

# OpenAI
if curl -f -H "Authorization: Bearer $OPENAI_API_KEY" https://api.openai.com/v1/models
> /dev/null 2>&1; then
    echo "    OpenAI API"
else
    echo "    OpenAI API"
fi

# SendGrid
if curl -f -H "Authorization: Bearer $SENDGRID_API_KEY" https://api.sendgrid.com/v3/
user/profile > /dev/null 2>&1; then
    echo "    SendGrid API"
else
    echo "    SendGrid API"
```

```
fi

# Verificar espaço em disco
echo "  Armazenamento:"
DISK_USAGE=$(df -h / | awk 'NR==2 {print $5}' | sed 's/%//')
echo "    Uso do disco: ${DISK_USAGE}%"

if [ "$DISK_USAGE" -gt 90 ]; then
    echo "  ⚠ Pouco espaço em disco"
fi

# Verificar memória
echo "  Memória:"
MEMORY_USAGE=$(free | grep Mem | awk '{printf "%.0f", $3/$2 * 100.0}')
echo "    Uso de memória: ${MEMORY_USAGE}%"

if [ "$MEMORY_USAGE" -gt 90 ]; then
    echo "  ⚠ Pouca memória disponível"
fi

echo "================================="
echo "  Health check concluído"
```

2. Script de Diagnóstico Avançado

```
// scripts/advanced-diagnostics.js
import { PrismaClient } from '@prisma/client';
import { performance } from 'perf_hooks';

const prisma = new PrismaClient();

async function runDiagnostics() {
  console.log('  Executando diagnósticos avançados...\n');

  // 1. Performance do banco
  await testDatabasePerformance();

  // 2. Verificar integridade dos dados
  await checkDataIntegrity();

  // 3. Analisar uso de recursos
  await analyzeResourceUsage();

  // 4. Verificar configurações
  await checkConfiguration();

  console.log('\n  Diagnósticos concluídos');
}

async function testDatabasePerformance() {
  console.log('  Testando performance do banco...');

  const tests = [
    {
      name: 'Query simples de usuários',
      query: () => prisma.user.findMany({ take: 10 })
    },
    {
      name: 'Query complexa com joins',
      query: () => prisma.user.findMany({
        include: {
          brandProfiles: true,
          monitoringSessions: true
        },
        take: 5
      })
    },
    {
      name: 'Agregação de dados',
      query: () => prisma.detectedContent.aggregate({
        _count: true,
        _avg: { similarity: true }
      })
    }
  ];

  for (const test of tests) {
    const start = performance.now();
    try {
      await test.query();
      const duration = performance.now() - start;
      console.log(`    ${test.name}: ${duration.toFixed(2)}ms`);
```

```
      if (duration > 1000) {
        console.log(`    ⚠ Query lenta detectada`);
      }
    } catch (error) {
      console.log(`    ${test.name}: ${error.message}`);
    }
  }
}

async function checkDataIntegrity() {
  console.log('\n  Verificando integridade dos dados...');

  // Verificar usuários órfãos
  const orphanedSessions = await prisma.monitoringSession.count({
    where: {
      user: null
    }
  });

  if (orphanedSessions > 0) {
    console.log(`  ⚠ ${orphanedSessions} sessões órfãs encontradas`);
  } else {
    console.log('    Nenhuma sessão órfã');
  }

  // Verificar sessões travadas
  const stuckSessions = await prisma.monitoringSession.count({
    where: {
      status: 'running',
      updatedAt: {
        lt: new Date(Date.now() - 30 * 60 * 1000) // 30 minutos
      }
    }
  });

  if (stuckSessions > 0) {
    console.log(`  ⚠ ${stuckSessions} sessões travadas encontradas`);
  } else {
    console.log('    Nenhuma sessão travada');
  }
}

async function analyzeResourceUsage() {
  console.log('\n  Analisando uso de recursos...');

  const memoryUsage = process.memoryUsage();
  console.log(`    Memória RSS: ${(memoryUsage.rss / 1024 / 1024).toFixed(2)} MB`);
  console.log(`    Heap usado: ${(memoryUsage.heapUsed / 1024 / 1024).toFixed(2)} MB`);
  console.log(`    Heap total: ${(memoryUsage.heapTotal / 1024 / 1024).toFixed(2)} MB`)
;

  if (memoryUsage.heapUsed / memoryUsage.heapTotal > 0.9) {
    console.log('  ⚠ Alto uso de memória heap');
  }
}

async function checkConfiguration() {
  console.log('\n⚙ Verificando configurações...');
```

```
  const requiredEnvVars = [
    'DATABASE_URL',
    'NEXTAUTH_SECRET',
    'OPENAI_API_KEY',
    'SENDGRID_API_KEY'
  ];

  for (const envVar of requiredEnvVars) {
    if (process.env[envVar]) {
      console.log(`    ${envVar} configurado`);
    } else {
      console.log(`    ${envVar} não configurado`);
    }
  }
}

runDiagnostics().catch(console.error);
```

Quando Buscar Ajuda

1. Problemas que Requerem Suporte

```
  Entre em contato quando:
• Perda de dados
• Falhas de segurança
• Performance extremamente degradada
• Problemas de compliance/legal
• Falhas de infraestrutura
```

2. Informações para Incluir no Suporte

```
📋 Sempre inclua:
• Versão da aplicação
• Logs relevantes (últimas 24h)
• Passos para reproduzir o problema
• Impacto nos usuários
• Tentativas de solução já realizadas

📞 Contatos:
• Suporte Técnico: suporte@dmcaguard.com
• Emergências: +55 11 99999-9999
• Discord: discord.gg/dmcaguard
```

3. Escalação de Problemas

```
  Níveis de Escalação:

Nível 1 - Suporte Básico:
• Problemas de usuário
• Dúvidas de configuração
• Bugs menores

Nível 2 - Suporte Técnico:
• Problemas de performance
• Falhas de integração
• Bugs críticos

Nível 3 - Engenharia:
• Falhas de infraestrutura
• Problemas de arquitetura
• Emergências de segurança
```

Lembre-se: A maioria dos problemas pode ser resolvida seguindo este guia. Quando em dúvida, sempre verifique os logs primeiro e teste em ambiente de desenvolvimento antes de aplicar correções em produção.