Guia de Administração - DMCA Guard Platform

Guia completo para administradores da plataforma DMCA Guard, cobrindo gestão de usuários, monitoramento do sistema, configurações avançadas e manutenção.

Acesso Administrativo

1. Níveis de Acesso

Hierarquia de Permissões

```
  Super Admin (Nível 5):
• Acesso total ao sistema
• Gerenciamento de outros admins
• Configurações de infraestrutura
• Acesso a dados sensíveis

  Admin (Nível 4):
• Gerenciamento de usuários
• Configurações da plataforma
• Relatórios e analytics
• Suporte avançado

   Moderador (Nível 3):
• Revisão de conteúdo
• Suporte a usuários
• Moderação de falsos positivos
• Relatórios básicos

  Suporte (Nível 2):
• Atendimento a usuários
• Tickets de suporte
• FAQ e documentação
• Escalação para níveis superiores
```

2. Painel Administrativo

Acesso ao Admin Panel

```
URL: https://dmcaguard.com/admin
Login: admin@dmcaguard.com
2FA: Obrigatório para todos os admins

Dashboard Principal:
┌─────────────────────────────────────────────────────┐
│  DMCA Guard - Painel Administrativo                 │
├─────────────────────────────────────────────────────┤
│  Métricas em Tempo Real:                            │
│  • Usuários Online: 247                             │
│  • Sessões Ativas: 89                               │
│  • DMCAs Enviadas (24h): 156                        │
│  • Taxa de Sucesso: 78%                             │
│  • Uptime: 99.97%                                   │
├─────────────────────────────────────────────────────┤
│  Alertas Críticos:                                  │
│  • API OpenAI: Rate limit próximo (85%)             │
│  • Banco de dados: Uso de disco alto (92%)          │
│  • SendGrid: Quota mensal em 70%                    │
└─────────────────────────────────────────────────────┘
```

Gerenciamento de Usuários

1. Dashboard de Usuários

Visão Geral

```sql
-- Query para métricas de usuários
SELECT
  COUNT(*) as total_users,
  COUNT(CASE WHEN created_at >= NOW() - INTERVAL '30 days' THEN 1 END) as new_users_30d
,
  COUNT(CASE WHEN last_login >= NOW() - INTERVAL '7 days' THEN 1 END) as act-
ive_users_7d,
  COUNT(CASE WHEN subscription_status = 'active' THEN 1 END) as paid_users
FROM users;
```

Interface de Gerenciamento

```
□ Gerenciamento de Usuários:

□ Filtros:
• Status: [Todos] [Ativos] [Suspensos] [Cancelados]
• Plano: [Gratuito] [Pro] [Enterprise]
• Registro: [Últimos 7 dias] [Último mês] [Personalizado]
• Atividade: [Ativos] [Inativos] [Nunca logaram]

□ Lista de Usuários:
┌──────────────────────────────────────────────────────────┐
│ ID  │ Nome          │ Email              │ Plano │ Status│
├──────────────────────────────────────────────────────────┤
│ 001 │ Maria Silva   │ maria@email.com    │ Pro   │ ●     │
│ 002 │ Ana Costa     │ ana@email.com      │ Free  │ ●     │
│ 003 │ Julia Santos  │ julia@email.com    │ Ent   │ ●     │
└──────────────────────────────────────────────────────────┘

□ Ações em Massa:
[□ Enviar Email] [□ Suspender] [□ Alterar Plano] [□ Exportar]
```

2. Perfil Detalhado do Usuário

Informações Completas

```jsx
// Componente UserProfile.jsx
export function UserProfile({ userId }) {
  return (
    <div className="user-profile">
      <UserHeader user={user} />

      <Tabs>
        <Tab label="Informações Gerais">
          <UserInfo user={user} />
          <SubscriptionInfo subscription={user.subscription} />
          <ActivityLog activities={user.activities} />
        </Tab>

        <Tab label="Perfis de Marca">
          <BrandProfiles profiles={user.brandProfiles} />
        </Tab>

        <Tab label="Monitoramento">
          <MonitoringSessions sessions={user.sessions} />
          <DetectedContent content={user.detectedContent} />
        </Tab>

        <Tab label="DMCA">
          <DMCAHistory dmcas={user.dmcaRequests} />
          <SuccessRate rate={user.dmcaSuccessRate} />
        </Tab>

        <Tab label="Suporte">
          <SupportTickets tickets={user.supportTickets} />
          <Notes notes={user.adminNotes} />
        </Tab>
      </Tabs>
    </div>
  );
}
```

Ações Administrativas

```
 Ações Disponíveis:

 Conta:
• Editar informações
• Resetar senha
• Ativar/Suspender conta
• Verificar email/telefone
• Adicionar notas administrativas

 Assinatura:
• Alterar plano
• Aplicar desconto
• Estender período gratuito
• Cancelar assinatura
• Reembolsar pagamento

 Técnico:
• Limpar cache do usuário
• Reprocessar sessões
• Corrigir dados inconsistentes
• Exportar dados (LGPD)
• Excluir conta permanentemente
```

3. Moderação de Conteúdo

Sistema de Revisão

```
 Fila de Moderação:

 Itens Pendentes: 23
┌──────────────────────────────────────────────────────────────┐
  Tipo           Usuário         Status     Prioridade
├──────────────────────────────────────────────────────────────┤
  Falso Positivo  Maria Silva     Pendente    Alta
  Conteúdo Novo  Ana Costa       Revisão     Média
  Disputa DMCA   Julia Santos    Pendente    Alta
└──────────────────────────────────────────────────────────────┘

 Ações de Moderação:
• [ ] Aprovar
• [ ] Rejeitar
• [ ] Solicitar mais informações
• [ ] Marcar como spam
• [ ] Adicionar nota
```

Ferramentas de Moderação

```
// lib/admin/moderation.js
export class ModerationTools {
  static async reviewContent(contentId, decision, reason) {
    const content = await prisma.detectedContent.findUnique({
      where: { id: contentId },
      include: { user: true, brandProfile: true }
    });

    await prisma.moderationAction.create({
      data: {
        contentId,
        adminId: getCurrentAdmin().id,
        action: decision,
        reason,
        timestamp: new Date()
      }
    });

    // Notificar usuário
    await NotificationService.sendModerationResult(content.user, decision);
  }

  static async bulkModeration(contentIds, action) {
    return await prisma.$transaction(
      contentIds.map(id =>
        prisma.detectedContent.update({
          where: { id },
          data: { status: action }
        })
      )
    );
  }
}
```

Analytics e Relatórios

1. Dashboard Executivo

KPIs Principais

```jsx
// components/admin/ExecutiveDashboard.jsx
export function ExecutiveDashboard() {
  const [metrics, setMetrics] = useState({
    revenue: { current: 0, growth: 0 },
    users: { total: 0, active: 0, churn: 0 },
    platform: { uptime: 0, performance: 0 },
    dmca: { sent: 0, success_rate: 0 }
  });

  return (
    <div className="executive-dashboard">
      <MetricCard
        title="Receita Mensal"
        value={formatCurrency(metrics.revenue.current)}
        growth={metrics.revenue.growth}
        icon=" "
      />

      <MetricCard
        title="Usuários Ativos"
        value={metrics.users.active}
        subtitle={`${metrics.users.total} total`}
        icon=" "
      />

      <MetricCard
        title="Uptime"
        value={`${metrics.platform.uptime}%`}
        status={metrics.platform.uptime > 99.5 ? 'good' : 'warning'}
        icon=" "
      />

      <MetricCard
        title="Taxa DMCA"
        value={`${metrics.dmca.success_rate}%`}
        subtitle={`${metrics.dmca.sent} enviadas`}
        icon=" "
      />
    </div>
  );
}
```

Gráficos Avançados

```
// components/admin/AdvancedCharts.jsx
export function AdvancedCharts() {
  return (
    <div className="charts-grid">
      <Chart
        type="line"
        title="Crescimento de Usuários"
        data={userGrowthData}
        timeRange="6months"
      />

      <Chart
        type="bar"
        title="Receita por Plano"
        data={revenueByPlanData}
        breakdown="monthly"
      />

      <Chart
        type="heatmap"
        title="Atividade por Hora"
        data={activityHeatmapData}
        timezone="America/Sao_Paulo"
      />

      <Chart
        type="funnel"
        title="Funil de Conversão"
        data={conversionFunnelData}
        stages={['Registro', 'Primeiro Login', 'Perfil Criado', 'Upgrade']}
      />
    </div>
  );
}
```

2. Relatórios Automatizados

Relatório Diário

```
// scripts/daily-report.js
export async function generateDailyReport() {
  const yesterday = new Date();
  yesterday.setDate(yesterday.getDate() - 1);

  const report = {
    date: yesterday.toISOString().split('T')[0],
    users: await getUserMetrics(yesterday),
    revenue: await getRevenueMetrics(yesterday),
    platform: await getPlatformMetrics(yesterday),
    dmca: await getDMCAMetrics(yesterday),
    support: await getSupportMetrics(yesterday)
  };

  // Enviar para equipe
  await EmailService.sendDailyReport(report);

  // Salvar no banco
  await prisma.dailyReport.create({ data: report });

  return report;
}

async function getUserMetrics(date) {
  return {
    newRegistrations: await prisma.user.count({
      where: { createdAt: { gte: date } }
    }),
    activeUsers: await prisma.user.count({
      where: { lastLogin: { gte: date } }
    }),
    upgrades: await prisma.subscription.count({
      where: {
        createdAt: { gte: date },
        status: 'active'
      }
    }),
    cancellations: await prisma.subscription.count({
      where: {
        canceledAt: { gte: date }
      }
    })
  };
}
```

Relatório Semanal

```bash
#!/bin/bash
# scripts/weekly-report.sh

echo "  Gerando relatório semanal..."

# Executar análises
node scripts/weekly-analytics.js

# Gerar gráficos
python scripts/generate-charts.py --period=week

# Compilar relatório
pandoc reports/weekly-template.md \
  --data reports/weekly-data.json \
  --output reports/weekly-report-$(date +%Y%m%d).pdf

# Enviar para stakeholders
node scripts/send-weekly-report.js

echo "  Relatório semanal enviado!"
```

Configurações do Sistema

1. Configurações Gerais

Painel de Configurações

```jsx
// components/admin/SystemSettings.jsx
export function SystemSettings() {
  return (
    <SettingsPanel>
      <Section title="Plataforma">
        <Setting
          name="maintenance_mode"
          label="Modo Manutenção"
          type="boolean"
          description="Ativar para manutenção programada"
        />

        <Setting
          name="new_registrations"
          label="Novos Registros"
          type="boolean"
          description="Permitir novos usuários"
        />

        <Setting
          name="max_users_per_plan"
          label="Limite de Usuários"
          type="object"
          schema={{
            free: { type: 'number', default: 1000 },
            pro: { type: 'number', default: 10000 },
            enterprise: { type: 'number', default: -1 }
          }}
        />
      </Section>

      <Section title="Monitoramento">
        <Setting
          name="monitoring_interval"
          label="Intervalo de Monitoramento"
          type="select"
          options={[
            { value: 3600000, label: '1 hora' },
            { value: 21600000, label: '6 horas' },
            { value: 86400000, label: '24 horas' }
          ]}
        />

        <Setting
          name="max_concurrent_sessions"
          label="Sessões Simultâneas"
          type="number"
          min={1}
          max={100}
        />
      </Section>

      <Section title="DMCA">
        <Setting
          name="auto_dmca_threshold"
          label="Threshold Automático"
          type="range"
          min={0.7}
```

```
          max={1.0}
          step={0.05}
          description="Similaridade mínima para DMCA automática"
        />

        <Setting
          name="dmca_templates"
          label="Templates DMCA"
          type="textarea"
          description="Templates personalizados por idioma"
        />
      </Section>
    </SettingsPanel>
  );
}
```

2. Configurações de APIs

Gerenciamento de Chaves

```
// lib/admin/api-management.js
export class APIManagement {
  static async rotateAPIKeys() {
    const services = ['openai', 'sendgrid', 'stripe'];

    for (const service of services) {
      try {
        const newKey = await this.generateNewKey(service);
        await this.updateServiceKey(service, newKey);
        await this.notifyKeyRotation(service);

        console.log(`  ${service} key rotated successfully`);
      } catch (error) {
        console.error(`  Failed to rotate ${service} key:`, error);
        await this.alertAdmins(`Key rotation failed for ${service}`);
      }
    }
  }

  static async monitorAPIUsage() {
    const usage = await Promise.all([
      this.getOpenAIUsage(),
      this.getSendGridUsage(),
      this.getStripeUsage()
    ]);

    const alerts = usage.filter(service =>
      service.usage > service.limit * 0.8
    );

    if (alerts.length > 0) {
      await this.sendUsageAlerts(alerts);
    }

    return usage;
  }
}
```

Rate Limiting

```
// lib/admin/rate-limiting.js
export const rateLimitConfig = {
  global: {
    windowMs: 15 * 60 * 1000, // 15 minutos
    max: 1000, // requests por IP
  },

  api: {
    windowMs: 60 * 1000, // 1 minuto
    max: 60, // requests por usuário
  },

  dmca: {
    windowMs: 60 * 60 * 1000, // 1 hora
    max: 10, // DMCAs por usuário
  },

  upload: {
    windowMs: 60 * 1000, // 1 minuto
    max: 5, // uploads por usuário
  }
};

export function createRateLimiter(config) {
  return rateLimit({
    ...config,
    message: {
      error: 'Rate limit exceeded',
      retryAfter: Math.ceil(config.windowMs / 1000)
    },
    standardHeaders: true,
    legacyHeaders: false,
  });
}
```

Monitoramento e Alertas

1. Sistema de Alertas

Configuração de Alertas

```
// lib/admin/alerting.js
export class AlertingSystem {
  static alerts = [
    {
      name: 'high_error_rate',
      condition: 'error_rate > 5%',
      severity: 'critical',
      channels: ['email', 'slack', 'sms']
    },
    {
      name: 'low_disk_space',
      condition: 'disk_usage > 90%',
      severity: 'warning',
      channels: ['email', 'slack']
    },
    {
      name: 'api_quota_exceeded',
      condition: 'api_usage > 80%',
      severity: 'warning',
      channels: ['email']
    },
    {
      name: 'dmca_success_rate_low',
      condition: 'dmca_success_rate < 70%',
      severity: 'warning',
      channels: ['email']
    }
  ];

  static async checkAlerts() {
    for (const alert of this.alerts) {
      const triggered = await this.evaluateCondition(alert.condition);

      if (triggered) {
        await this.sendAlert(alert);
      }
    }
  }

  static async sendAlert(alert) {
    const message = this.formatAlertMessage(alert);

    for (const channel of alert.channels) {
      switch (channel) {
        case 'email':
          await EmailService.sendAlert(message);
          break;
        case 'slack':
          await SlackService.sendAlert(message);
          break;
        case 'sms':
          await SMSService.sendAlert(message);
          break;
      }
    }
  }
}
```

2. Logs e Auditoria

Sistema de Logs

```
// lib/admin/logging.js
export class LoggingSystem {
  static async logAdminAction(adminId, action, details) {
    await prisma.adminLog.create({
      data: {
        adminId,
        action,
        details: JSON.stringify(details),
        timestamp: new Date(),
        ipAddress: this.getClientIP(),
        userAgent: this.getUserAgent()
      }
    });
  }

  static async getAuditTrail(filters = {}) {
    return await prisma.adminLog.findMany({
      where: {
        ...filters,
        timestamp: {
          gte: filters.startDate,
          lte: filters.endDate
        }
      },
      include: {
        admin: {
          select: { name: true, email: true }
        }
      },
      orderBy: { timestamp: 'desc' }
    });
  }

  static async exportAuditTrail(filters, format = 'csv') {
    const logs = await this.getAuditTrail(filters);

    switch (format) {
      case 'csv':
        return this.exportToCSV(logs);
      case 'json':
        return this.exportToJSON(logs);
      case 'pdf':
        return this.exportToPDF(logs);
      default:
        throw new Error('Unsupported export format');
    }
  }
}
```

Dashboard de Logs

```jsx
// components/admin/LogsDashboard.jsx
export function LogsDashboard() {
  const [logs, setLogs] = useState([]);
  const [filters, setFilters] = useState({
    level: 'all',
    service: 'all',
    timeRange: '24h'
  });

  return (
    <div className="logs-dashboard">
      <LogsFilters filters={filters} onChange={setFilters} />

      <LogsChart data={logs} />

      <LogsTable
        logs={logs}
        onExport={handleExport}
        onFilter={handleFilter}
      />

      <LogsRealtime />
    </div>
  );
}
```

Gestão Financeira

1. Dashboard Financeiro

Métricas de Receita

```jsx
// components/admin/FinancialDashboard.jsx
export function FinancialDashboard() {
  const [metrics, setMetrics] = useState({
    mrr: 0, // Monthly Recurring Revenue
    arr: 0, // Annual Recurring Revenue
    churn: 0,
    ltv: 0, // Lifetime Value
    cac: 0, // Customer Acquisition Cost
  });

  return (
    <div className="financial-dashboard">
      <RevenueChart data={revenueData} />

      <MetricsGrid>
        <MetricCard title="MRR" value={formatCurrency(metrics.mrr)} />
        <MetricCard title="ARR" value={formatCurrency(metrics.arr)} />
        <MetricCard title="Churn Rate" value={`${metrics.churn}%`} />
        <MetricCard title="LTV" value={formatCurrency(metrics.ltv)} />
        <MetricCard title="CAC" value={formatCurrency(metrics.cac)} />
      </MetricsGrid>

      <SubscriptionBreakdown />
      <RevenueProjection />
    </div>
  );
}
```

2. Gestão de Assinaturas

Operações de Assinatura

```
// lib/admin/subscription-management.js
export class SubscriptionManagement {
  static async bulkUpdatePricing(planId, newPrice) {
    const subscriptions = await prisma.subscription.findMany({
      where: { planId, status: 'active' }
    });

    for (const subscription of subscriptions) {
      await this.updateSubscriptionPrice(subscription.id, newPrice);
      await this.notifyPriceChange(subscription.userId, newPrice);
    }
  }

  static async applyPromoCode(userId, promoCode) {
    const promo = await prisma.promoCode.findUnique({
      where: { code: promoCode }
    });

    if (!promo || promo.expiresAt < new Date()) {
      throw new Error('Código promocional inválido ou expirado');
    }

    await prisma.subscription.update({
      where: { userId },
      data: {
        discountPercent: promo.discountPercent,
        discountExpiresAt: promo.expiresAt
      }
    });
  }

  static async generateRevenueReport(startDate, endDate) {
    const revenue = await prisma.payment.aggregate({
      where: {
        createdAt: { gte: startDate, lte: endDate },
        status: 'completed'
      },
      _sum: { amount: true },
      _count: true
    });

    const subscriptions = await prisma.subscription.groupBy({
      by: ['planId'],
      where: {
        createdAt: { gte: startDate, lte: endDate }
      },
      _count: true,
      _sum: { amount: true }
    });

    return {
      totalRevenue: revenue._sum.amount,
      totalTransactions: revenue._count,
      subscriptionBreakdown: subscriptions
    };
  }
}
```

Manutenção do Sistema

1. Tarefas de Manutenção

Limpeza Automática

```
// scripts/maintenance/cleanup.js
export async function runCleanupTasks() {
  console.log('  Iniciando tarefas de limpeza...');

  // Limpar logs antigos (> 90 dias)
  await prisma.log.deleteMany({
    where: {
      createdAt: {
        lt: new Date(Date.now() - 90 * 24 * 60 * 60 * 1000)
      }
    }
  });

  // Limpar sessões expiradas
  await prisma.session.deleteMany({
    where: {
      expiresAt: { lt: new Date() }
    }
  });

  // Limpar arquivos temporários
  await cleanupTempFiles();

  // Otimizar banco de dados
  await optimizeDatabase();

  console.log('  Limpeza concluída!');
}

async function cleanupTempFiles() {
  const tempDir = '/tmp/dmca-guard';
  const files = await fs.readdir(tempDir);

  for (const file of files) {
    const filePath = path.join(tempDir, file);
    const stats = await fs.stat(filePath);

    // Remover arquivos > 24h
    if (Date.now() - stats.mtime.getTime() > 24 * 60 * 60 * 1000) {
      await fs.unlink(filePath);
    }
  }
}
```

Backup Automático

```
#!/bin/bash
# scripts/maintenance/backup.sh

set -e

BACKUP_DIR="/backups/$(date +%Y%m%d)"
mkdir -p $BACKUP_DIR

echo "  Iniciando backup..."

# Backup do banco de dados
pg_dump $DATABASE_URL | gzip > "$BACKUP_DIR/database.sql.gz"

# Backup de arquivos de upload
tar -czf "$BACKUP_DIR/uploads.tar.gz" uploads/

# Backup de configurações
cp -r config/ "$BACKUP_DIR/config/"

# Upload para S3
aws s3 sync $BACKUP_DIR s3://dmca-guard-backups/$(date +%Y%m%d)/

# Limpar backups antigos (> 30 dias)
find /backups -type d -mtime +30 -exec rm -rf {} \;

echo "  Backup concluído!"
```

2. Monitoramento de Performance

Métricas de Sistema

```
// lib/admin/performance-monitoring.js
export class PerformanceMonitoring {
  static async collectMetrics() {
    return {
      system: await this.getSystemMetrics(),
      database: await this.getDatabaseMetrics(),
      api: await this.getAPIMetrics(),
      application: await this.getApplicationMetrics()
    };
  }

  static async getSystemMetrics() {
    return {
      cpu: await this.getCPUUsage(),
      memory: await this.getMemoryUsage(),
      disk: await this.getDiskUsage(),
      network: await this.getNetworkStats()
    };
  }

  static async getDatabaseMetrics() {
    const result = await prisma.$queryRaw`
      SELECT
        schemaname,
        tablename,
        attname,
        n_distinct,
        correlation
      FROM pg_stats
      WHERE schemaname = 'public'
    `;

    return {
      connectionCount: await this.getConnectionCount(),
      queryPerformance: await this.getSlowQueries(),
      tableStats: result
    };
  }

  static async generatePerformanceReport() {
    const metrics = await this.collectMetrics();

    const report = {
      timestamp: new Date(),
      summary: this.generateSummary(metrics),
      recommendations: this.generateRecommendations(metrics),
      metrics
    };

    await this.saveReport(report);
    return report;
  }
}
```

Segurança e Compliance

1. Auditoria de Segurança

Checklist de Segurança

```
// lib/admin/security-audit.js
export class SecurityAudit {
  static async runSecurityCheck() {
    const checks = [
      this.checkPasswordPolicies(),
      this.checkSSLCertificates(),
      this.checkAPIKeyRotation(),
      this.checkAccessControls(),
      this.checkDataEncryption(),
      this.checkBackupSecurity(),
      this.checkLogIntegrity()
    ];

    const results = await Promise.all(checks);

    return {
      overall: this.calculateOverallScore(results),
      checks: results,
      recommendations: this.generateSecurityRecommendations(results)
    };
  }

  static async checkPasswordPolicies() {
    const weakPasswords = await prisma.user.count({
      where: {
        password: {
          // Verificar senhas fracas
          in: await this.getCommonPasswords()
        }
      }
    });

    return {
      name: 'Password Policies',
      status: weakPasswords === 0 ? 'pass' : 'fail',
      details: `${weakPasswords} usuários com senhas fracas`,
      severity: weakPasswords > 0 ? 'high' : 'low'
    };
  }

  static async checkAPIKeyRotation() {
    const lastRotation = await prisma.apiKey.findFirst({
      orderBy: { rotatedAt: 'desc' }
    });

    const daysSinceRotation = lastRotation
      ? Math.floor((Date.now() - lastRotation.rotatedAt.getTime()) / (1000 * 60 * 60 *
24))
      : 999;

    return {
      name: 'API Key Rotation',
      status: daysSinceRotation < 90 ? 'pass' : 'fail',
      details: `Última rotação há ${daysSinceRotation} dias`,
      severity: daysSinceRotation > 180 ? 'high' : 'medium'
    };
  }
}
```

2. Compliance LGPD

Ferramentas de Compliance

```
// lib/admin/lgpd-compliance.js
export class LGPDCompliance {
  static async handleDataRequest(userId, requestType) {
    switch (requestType) {
      case 'access':
        return await this.exportUserData(userId);
      case 'correction':
        return await this.enableDataCorrection(userId);
      case 'deletion':
        return await this.scheduleDataDeletion(userId);
      case 'portability':
        return await this.exportPortableData(userId);
      default:
        throw new Error('Tipo de solicitação inválido');
    }
  }

  static async exportUserData(userId) {
    const userData = await prisma.user.findUnique({
      where: { id: userId },
      include: {
        brandProfiles: true,
        monitoringSessions: true,
        detectedContent: true,
        dmcaRequests: true,
        payments: true,
        supportTickets: true
      }
    });

    // Anonimizar dados sensíveis
    const anonymizedData = this.anonymizeSensitiveData(userData);

    // Gerar relatório
    const report = await this.generateDataReport(anonymizedData);

    // Log da solicitação
    await this.logDataRequest(userId, 'access');

    return report;
  }

  static async scheduleDataDeletion(userId) {
    // Verificar se há obrigações legais que impedem a exclusão
    const legalHolds = await this.checkLegalHolds(userId);

    if (legalHolds.length > 0) {
      throw new Error('Não é possível excluir dados devido a obrigações legais');
    }

    // Agendar exclusão para 30 dias
    await prisma.dataDeletionRequest.create({
      data: {
        userId,
        scheduledFor: new Date(Date.now() + 30 * 24 * 60 * 60 * 1000),
        status: 'scheduled'
      }
    });
```

```
    // Notificar usuário
    await this.notifyDataDeletion(userId);
  }
}
```

Gestão de Suporte

1. Sistema de Tickets

Dashboard de Suporte

```
// components/admin/SupportDashboard.jsx
export function SupportDashboard() {
  const [tickets, setTickets] = useState([]);
  const [metrics, setMetrics] = useState({
    open: 0,
    pending: 0,
    resolved: 0,
    avgResponseTime: 0
  });

  return (
    <div className="support-dashboard">
      <SupportMetrics metrics={metrics} />

      <TicketQueue
        tickets={tickets}
        onAssign={handleAssignTicket}
        onResolve={handleResolveTicket}
      />

      <SupportAnalytics />
    </div>
  );
}
```

Automação de Suporte

```
// lib/admin/support-automation.js
export class SupportAutomation {
  static async autoAssignTicket(ticket) {
    // Classificar ticket por categoria
    const category = await this.classifyTicket(ticket.content);

    // Encontrar agente disponível com expertise
    const agent = await this.findBestAgent(category);

    if (agent) {
      await this.assignTicket(ticket.id, agent.id);
      await this.notifyAgent(agent, ticket);
    } else {
      await this.escalateTicket(ticket.id);
    }
  }

  static async generateAutoResponse(ticket) {
    const intent = await this.detectIntent(ticket.content);

    const responses = {
      'password_reset': 'Para redefinir sua senha, acesse...',
      'billing_question': 'Para questões de cobrança, verifique...',
      'technical_issue': 'Para problemas técnicos, tente...'
    };

    return responses[intent] || 'Obrigado pelo contato. Nossa equipe responderá em
breve.';
  }
}
```

Otimização e Escalabilidade

1. Otimização de Performance

Cache Strategy

```
// lib/admin/cache-management.js
export class CacheManagement {
  static async optimizeCache() {
    // Limpar cache expirado
    await this.clearExpiredCache();

    // Pré-carregar dados frequentes
    await this.preloadFrequentData();

    // Otimizar estratégia de cache
    await this.optimizeCacheStrategy();
  }

  static async clearExpiredCache() {
    const redis = new Redis(process.env.REDIS_URL);

    const keys = await redis.keys('cache:*');
    const expiredKeys = [];

    for (const key of keys) {
      const ttl = await redis.ttl(key);
      if (ttl === -1 || ttl === 0) {
        expiredKeys.push(key);
      }
    }

    if (expiredKeys.length > 0) {
      await redis.del(...expiredKeys);
    }
  }

  static async getCacheStats() {
    const redis = new Redis(process.env.REDIS_URL);
    const info = await redis.info('memory');

    return {
      usedMemory: this.parseMemoryInfo(info, 'used_memory'),
      maxMemory: this.parseMemoryInfo(info, 'maxmemory'),
      hitRate: await this.calculateHitRate(),
      keyCount: await redis.dbsize()
    };
  }
}
```

2. Scaling Strategies

Auto-scaling Configuration

```yaml
# kubernetes/autoscaling.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: dmca-guard-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: dmca-guard-app
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

Recursos Adicionais

Scripts Úteis

- `scripts/admin/user-cleanup.sh` - Limpeza de usuários inativos
- `scripts/admin/performance-check.sh` - Verificação de performance
- `scripts/admin/security-scan.sh` - Scan de segurança
- `scripts/admin/backup-restore.sh` - Backup e restore

Documentação Técnica

- API Documentation (api-docs.md)
- Database Schema (database-schema.md)
- Security Guidelines (security-guidelines.md)
- Deployment Guide (deployment-guide.md)

Contatos de Emergência

- **DevOps**: devops@dmcaguard.com
- **Security**: security@dmcaguard.com
- **Legal**: legal@dmcaguard.com

Lembre-se: Como administrador, você tem acesso a dados sensíveis. Sempre siga as melhores práticas de segurança e compliance.