# 💻 Módulo de Branding Pro+ — Exemplos de Código

## 📋 Índice

# 🗄️ Schema Prisma

**prisma/schema.prisma (adição)**

```
// Tabela de empresas/organizações
model Company {
  id                    String            @id @default(cuid())
  name                  String
  slug                  String            @unique
  subscriptionPlan      String            @default("starter") // starter, pro, busi-
ness

  // Relações
  users                 User[]
  branding              CompanyBranding?

  createdAt             DateTime          @default(now())
  updatedAt             DateTime          @updatedAt

  @@map("companies")
}

// Tabela de configurações de branding
model CompanyBranding {
  id                    String   @id @default(cuid())
  companyId             String   @unique

  // Logotipo
  logoUrl               String?  // URL do logo (S3 ou local)
  logoSize              Int      @default(150) // Tamanho em pixels
  faviconUrl            String?  // Favicon personalizado

  // Cores Corporativas
  primaryColor          String   @default("#3B82F6") // Cor principal
  secondaryColor        String   @default("#8B5CF6") // Cor secundária
  accentColor           String   @default("#10B981") // Cor de destaque
  backgroundColor       String   @default("#FFFFFF") // Cor de fundo
  textColor             String   @default("#1F2937") // Cor do texto

  // Tela de Login
  loginBackgroundUrl    String?  // Imagem de fundo do login
  loginBackgroundType   String   @default("color") // "color", "image", "gradient"
  loginWelcomeMessage   String?  @db.Text // Mensagem de boas-vindas

  // Tema
  theme                 String   @default("light") // "light" ou "dark"
  borderRadius          String   @default("medium") // "none", "small", "medium", "lar
ge"
  fontFamily            String   @default("system") // "system", "inter", "roboto"

  // Subdomínio
  customSubdomain       String?  @unique // Ex: "minhaempresa"
  customDomain          String?  @unique // Ex: "gestao.minhaempresa.cv"

  // Configurações de Email
  emailHeaderColor      String?  // Cor do cabeçalho do email
  emailFooterText       String?  @db.Text // Texto do rodapé

  // Metadados
  isActive              Boolean  @default(true)
  planLevel             String   @default("basic") // "basic" ou "complete"

  createdAt             DateTime @default(now())
  updatedAt             DateTime @updatedAt

  // Relação
```

```
  company               Company  @relation(fields: [companyId], references: [id], onDe
lete: Cascade)

  @@map("company_branding")
}

// Extensão da tabela User (adicionar se não existir)
model User {
  // ... campos existentes

  companyId             String?
  company               Company? @relation(fields: [companyId], references: [id])

  // ... resto dos campos
}
```

## Migration

```
npx prisma migrate dev --name add_branding_tables
```

---

## 🛣️ API Routes

### app/api/branding/route.ts — GET/POST Configurações

```
import { NextRequest, NextResponse } from 'next/server';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth';
import { prisma } from '@/lib/db';

// GET - Obter configurações de branding
export async function GET(req: NextRequest) {
  try {
    const session = await getServerSession(authOptions);
    if (!session?.user) {
      return NextResponse.json({ error: 'Não autorizado' }, { status: 401 });
    }

    const user = await prisma.user.findUnique({
      where: { id: session.user.id },
      include: {
        company: {
          include: {
            branding: true
          }
        }
      }
    });

    if (!user?.company) {
      return NextResponse.json({ error: 'Empresa não encontrada' }, { status: 404 });
    }

    // Retornar branding ou configurações padrão
    const branding = user.company.branding || getDefaultBranding();

    return NextResponse.json(branding);
  } catch (error) {
    console.error('Erro ao obter branding:', error);
    return NextResponse.json({ error: 'Erro interno do servidor' }, { status: 500 });
  }
}

// POST - Criar/Atualizar configurações de branding
export async function POST(req: NextRequest) {
  try {
    const session = await getServerSession(authOptions);
    if (!session?.user) {
      return NextResponse.json({ error: 'Não autorizado' }, { status: 401 });
    }

    // Verificar se é Admin
    const user = await prisma.user.findUnique({
      where: { id: session.user.id },
      include: { company: true }
    });

    if (user?.role !== 'ADMIN') {
      return NextResponse.json({ error: 'Apenas administradores podem editar o brand-
ing' }, { status: 403 });
    }

    const body = await req.json();
    const {
      logoUrl,
      primaryColor,
      secondaryColor,
```

```
      accentColor,
      theme,
      loginWelcomeMessage,
      loginBackgroundUrl
    } = body;

    // Validar cores
    if (primaryColor && !isValidColor(primaryColor)) {
      return NextResponse.json({ error: 'Cor primária inválida' }, { status: 400 });
    }

    // Validar contraste
    if (primaryColor && !hasGoodContrast(primaryColor, '#FFFFFF')) {
      return NextResponse.json(
        { warning: 'Contraste baixo detectado. Recomendamos escolher uma cor mais es-
cura.' },
        { status: 200 }
      );
    }

    const companyId = user.companyId!;

    // Upsert (criar ou atualizar)
    const branding = await prisma.companyBranding.upsert({
      where: { companyId },
      create: {
        companyId,
        logoUrl,
        primaryColor,
        secondaryColor,
        accentColor,
        theme,
        loginWelcomeMessage,
        loginBackgroundUrl
      },
      update: {
        logoUrl,
        primaryColor,
        secondaryColor,
        accentColor,
        theme,
        loginWelcomeMessage,
        loginBackgroundUrl,
        updatedAt: new Date()
      }
    });

    return NextResponse.json({ success: true, branding });
  } catch (error) {
    console.error('Erro ao salvar branding:', error);
    return NextResponse.json({ error: 'Erro ao salvar configurações' }, { status:
500 });
  }
}

// Funções auxiliares
function getDefaultBranding() {
  return {
    primaryColor: '#3B82F6',
    secondaryColor: '#8B5CF6',
    accentColor: '#10B981',
    theme: 'light',
    logoUrl: '/branding/default-logo.png'
```

```
  };
}

function isValidColor(color: string): boolean {
  const hexRegex = /^#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})$/;
  return hexRegex.test(color);
}

function hasGoodContrast(color1: string, color2: string): boolean {
  // Implementação simplificada - use biblioteca como 'tinycolor2' para produção
  return true; // Placeholder
}
```

### app/api/branding/logo/route.ts — Upload de Logo

```
import { NextRequest, NextResponse } from 'next/server';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth';
import { uploadFile } from '@/lib/s3';
import { prisma } from '@/lib/db';

export async function POST(req: NextRequest) {
  try {
    const session = await getServerSession(authOptions);
    if (!session?.user) {
      return NextResponse.json({ error: 'Não autorizado' }, { status: 401 });
    }

    const user = await prisma.user.findUnique({
      where: { id: session.user.id }
    });

    if (user?.role !== 'ADMIN') {
      return NextResponse.json({ error: 'Apenas administradores' }, { status: 403 });
    }

    const formData = await req.formData();
    const file = formData.get('logo') as File;

    if (!file) {
      return NextResponse.json({ error: 'Nenhum ficheiro enviado' }, { status: 400 });
    }

    // Validações
    const ALLOWED_TYPES = ['image/png', 'image/jpeg', 'image/svg+xml'];
    const MAX_SIZE = 2 * 1024 * 1024; // 2MB

    if (!ALLOWED_TYPES.includes(file.type)) {
      return NextResponse.json({ error: 'Tipo de ficheiro não permitido' }, { status:
400 });
    }

    if (file.size > MAX_SIZE) {
      return NextResponse.json({ error: 'Ficheiro muito grande (máx 2MB)' }, {
status: 400 });
    }

    // Converter para Buffer
    const buffer = Buffer.from(await file.arrayBuffer());

    // Upload para S3
    const fileName = `branding/${user.companyId}/logo-${Date.now()}.$
{file.name.split('.').pop()}`;
    const logoUrl = await uploadFile(buffer, fileName);

    // Atualizar BD
    await prisma.companyBranding.upsert({
      where: { companyId: user.companyId! },
      create: {
        companyId: user.companyId!,
        logoUrl
      },
      update: {
        logoUrl,
        updatedAt: new Date()
      }
    });
```

```
    return NextResponse.json({ success: true, logoUrl });
  } catch (error) {
    console.error('Erro ao fazer upload do logo:', error);
    return NextResponse.json({ error: 'Erro ao fazer upload' }, { status: 500 });
  }
}
```

---

## ⚛️ Componentes React

### components/branding/color-picker.tsx

```
'use client';

import { useState } from 'react';
import { HexColorPicker } from 'react-colorful';
import { Input } from '@/components/ui/input';
import { Label } from '@/components/ui/label';
import { Popover, PopoverContent, PopoverTrigger } from '@/components/ui/popover';

interface ColorPickerProps {
  label: string;
  value: string;
  onChange: (color: string) => void;
}

export function ColorPicker({ label, value, onChange }: ColorPickerProps) {
  const [isOpen, setIsOpen] = useState(false);

  return (
    <div className="space-y-2">
      <Label>{label}</Label>
      <div className="flex gap-2">
        <Popover open={isOpen} onOpenChange={setIsOpen}>
          <PopoverTrigger asChild>
            <button
              className="w-12 h-12 rounded border-2 border-gray-300 hover:border-
gray-400 transition"
              style={{ backgroundColor: value }}
              aria-label={`Escolher ${label}`}
            />
          </PopoverTrigger>
          <PopoverContent className="w-auto p-3">
            <HexColorPicker color={value} onChange={onChange} />
          </PopoverContent>
        </Popover>
        <Input
          type="text"
          value={value}
          onChange={(e) => onChange(e.target.value)}
          placeholder="#3B82F6"
          className="flex-1"
        />
      </div>
    </div>
  );
}
```

**components/branding/logo-uploader.tsx**

```tsx
'use client';

import { useState } from 'react';
import { useDropzone } from 'react-dropzone';
import { Upload, X } from 'lucide-react';
import { Button } from '@/components/ui/button';
import Image from 'next/image';
import { toast } from 'sonner';

interface LogoUploaderProps {
  currentLogo?: string;
  onUpload: (url: string) => void;
}

export function LogoUploader({ currentLogo, onUpload }: LogoUploaderProps) {
  const [isUploading, setIsUploading] = useState(false);
  const [preview, setPreview] = useState(currentLogo);

  const { getRootProps, getInputProps, isDragActive } = useDropzone({
    accept: {
      'image/png': ['.png'],
      'image/jpeg': ['.jpg', '.jpeg'],
      'image/svg+xml': ['.svg']
    },
    maxSize: 2 * 1024 * 1024, // 2MB
    multiple: false,
    onDrop: async (acceptedFiles) => {
      if (acceptedFiles.length === 0) return;

      const file = acceptedFiles[0];
      setIsUploading(true);

      try {
        // Preview local
        const objectUrl = URL.createObjectURL(file);
        setPreview(objectUrl);

        // Upload para servidor
        const formData = new FormData();
        formData.append('logo', file);

        const response = await fetch('/api/branding/logo', {
          method: 'POST',
          body: formData
        });

        if (!response.ok) throw new Error('Erro ao fazer upload');

        const data = await response.json();
        onUpload(data.logoUrl);
        toast.success('Logo enviado com sucesso!');
      } catch (error) {
        console.error('Erro no upload:', error);
        toast.error('Erro ao enviar logo');
        setPreview(currentLogo);
      } finally {
        setIsUploading(false);
      }
    }
  });

  return (
```

```
    <div className="space-y-4">
      <div
        {...getRootProps()}
        className={`
          border-2 border-dashed rounded-lg p-8
          flex flex-col items-center justify-center
          cursor-pointer transition
          ${isDragActive ? 'border-primary bg-primary/5' : 'border-gray-300 hov-
er:border-primary'}
          ${isUploading ? 'opacity-50 cursor-not-allowed' : ''}
        `}
      >
        <input {...getInputProps()} disabled={isUploading} />

        {preview ? (
          <div className="relative w-32 h-32">
            <Image
              src={preview}
              alt="Logo preview"
              fill
              className="object-contain"
            />
          </div>
        ) : (
          <Upload className="w-12 h-12 text-gray-400 mb-2" />
        )}

        <p className="text-sm text-gray-600 text-center mt-2">
          {isDragActive
            ? 'Solte o ficheiro aqui'
            : 'Arraste o logo ou clique para selecionar'}
        </p>
        <p className="text-xs text-gray-400 mt-1">
          PNG, JPG ou SVG (máx 2MB)
        </p>
      </div>

      {preview && (
        <Button
          variant="outline"
          size="sm"
          onClick={() => {
            setPreview(undefined);
            onUpload('');
          }}
        >
          <X className="w-4 h-4 mr-2" />
          Remover logo
        </Button>
      )}
    </div>
  );
}
```

**app/settings/branding/page.tsx — Painel de Configuração**

```tsx
'use client';

import { useState, useEffect } from 'react';
import { Button } from '@/components/ui/button';
import { Card, CardHeader, CardTitle, CardContent } from '@/components/ui/card';
import { ColorPicker } from '@/components/branding/color-picker';
import { LogoUploader } from '@/components/branding/logo-uploader';
import { toast } from 'sonner';
import { Loader2 } from 'lucide-react';

export default function BrandingPage() {
  const [loading, setLoading] = useState(true);
  const [saving, setSaving] = useState(false);
  const [branding, setBranding] = useState({
    logoUrl: '',
    primaryColor: '#3B82F6',
    secondaryColor: '#8B5CF6',
    accentColor: '#10B981',
    theme: 'light'
  });

  useEffect(() => {
    fetchBranding();
  }, []);

  async function fetchBranding() {
    try {
      const response = await fetch('/api/branding');
      if (response.ok) {
        const data = await response.json();
        setBranding(data);
      }
    } catch (error) {
      console.error('Erro ao carregar branding:', error);
      toast.error('Erro ao carregar configurações');
    } finally {
      setLoading(false);
    }
  }

  async function handleSave() {
    setSaving(true);
    try {
      const response = await fetch('/api/branding', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(branding)
      });

      if (!response.ok) throw new Error('Erro ao salvar');

      toast.success('Configurações salvas com sucesso!');
    } catch (error) {
      console.error('Erro ao salvar:', error);
      toast.error('Erro ao salvar configurações');
    } finally {
      setSaving(false);
    }
  }

  if (loading) {
    return (
```

```
      <div className="flex items-center justify-center h-96">
        <Loader2 className="w-8 h-8 animate-spin" />
      </div>
    );
  }

  return (
    <div className="container max-w-4xl mx-auto p-6 space-y-6">
      <div>
        <h1 className="text-3xl font-bold">Personalização de Branding</h1>
        <p className="text-gray-600 mt-2">
          Configure a identidade visual da sua empresa no OrganiZen
        </p>
      </div>

      <Card>
        <CardHeader>
          <CardTitle>Logotipo</CardTitle>
        </CardHeader>
        <CardContent>
          <LogoUploader
            currentLogo={branding.logoUrl}
            onUpload={(url) => setBranding({ ...branding, logoUrl: url })}
          />
        </CardContent>
      </Card>

      <Card>
        <CardHeader>
          <CardTitle>Cores Corporativas</CardTitle>
        </CardHeader>
        <CardContent className="space-y-4">
          <ColorPicker
            label="Cor Primária"
            value={branding.primaryColor}
            onChange={(color) => setBranding({ ...branding, primaryColor: color })}
          />
          <ColorPicker
            label="Cor Secundária"
            value={branding.secondaryColor}
            onChange={(color) => setBranding({ ...branding, secondaryColor: color })}
          />
          <ColorPicker
            label="Cor de Destaque"
            value={branding.accentColor}
            onChange={(color) => setBranding({ ...branding, accentColor: color })}
          />
        </CardContent>
      </Card>

      <Card>
        <CardHeader>
          <CardTitle>Preview</CardTitle>
        </CardHeader>
        <CardContent>
          <div className="border rounded-lg p-6 space-y-4">
            <Button style={{ backgroundColor: branding.primaryColor }}>
              Botão Primário
            </Button>
            <Button
              variant="outline"
              style={{ borderColor: branding.secondaryColor, color: brand-
ing.secondaryColor }}
```

```
            >
              Botão Secundário
            </Button>
            <div
              className="h-12 rounded"
              style={{ backgroundColor: branding.accentColor }}
            />
          </div>
        </CardContent>
      </Card>

      <div className="flex justify-end gap-4">
        <Button variant="outline" onClick={fetchBranding}>
          Cancelar
        </Button>
        <Button onClick={handleSave} disabled={saving}>
          {saving && <Loader2 className="w-4 h-4 mr-2 animate-spin" />}
          Guardar Alterações
        </Button>
      </div>
    </div>
  );
}
```

---

## 🎨 Aplicação Dinâmica de CSS

### lib/branding/apply-theme.ts

```
export function applyBranding(branding: CompanyBranding) {
  if (typeof document === 'undefined') return; // SSR safety

  const root = document.documentElement;

  // Aplicar cores como CSS variables
  root.style.setProperty('--primary-color', branding.primaryColor);
  root.style.setProperty('--secondary-color', branding.secondaryColor || brand-
ing.primaryColor);
  root.style.setProperty('--accent-color', branding.accentColor || brand-
ing.primaryColor);

  // Aplicar tema
  if (branding.theme === 'dark') {
    root.classList.add('dark');
  } else {
    root.classList.remove('dark');
  }
}
```

### app/layout.tsx — Aplicação Global

```
'use client';

import { useEffect } from 'react';
import { useSession } from 'next-auth/react';
import { applyBranding } from '@/lib/branding/apply-theme';

export default function RootLayout({ children }: { children: React.ReactNode }) {
  const { data: session } = useSession();

  useEffect(() => {
    async function loadBranding() {
      if (!session?.user) return;

      try {
        const response = await fetch('/api/branding');
        if (response.ok) {
          const branding = await response.json();
          applyBranding(branding);
        }
      } catch (error) {
        console.error('Erro ao carregar branding:', error);
      }
    }

    loadBranding();
  }, [session]);

  return (
    <html lang="pt">
      <body>{children}</body>
    </html>
  );
}
```

## 📄 Geração de PDF com Branding

### lib/pdf/branded-pdf.ts

```
import PDFDocument from 'pdfkit';
import { CompanyBranding } from '@prisma/client';

export function createBrandedPDF(branding: CompanyBranding, data: any) {
  const doc = new PDFDocument({
    size: 'A4',
    margins: { top: 50, bottom: 50, left: 50, right: 50 }
  });

  // Cabeçalho com logo
  if (branding.logoUrl) {
    doc.image(branding.logoUrl, 50, 30, { width: 100 });
  }

  // Linha decorativa com cor corporativa
  doc
    .rect(50, 100, doc.page.width - 100, 3)
    .fill(branding.primaryColor);

  // Título
  doc
    .fontSize(20)
    .fillColor(branding.textColor || '#000000')
    .text('Relatório de Produtividade', 50, 120);

  // Conteúdo
  doc
    .fontSize(12)
    .fillColor('#000000')
    .text(data.content, 50, 160);

  // Rodapé com cor corporativa
  doc
    .rect(50, doc.page.height - 80, doc.page.width - 100, 2)
    .fill(branding.secondaryColor || branding.primaryColor);

  doc
    .fontSize(10)
    .fillColor('#666666')
    .text(
      `Gerado por ${branding.company?.name} | ${new Date().toLocaleDateString()}`,
      50,
      doc.page.height - 60,
      { align: 'center' }
    );

  return doc;
}
```

## 📧 Templates de Email

### lib/email/branded-template.ts

```
import { CompanyBranding } from '@prisma/client';

export function createBrandedEmailHTML(
  branding: CompanyBranding,
  subject: string,
  content: string
): string {
  const primaryColor = branding.primaryColor || '#3B82F6';
  const logoUrl = branding.logoUrl || '';

  return `
    <!DOCTYPE html>
    <html>
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>${subject}</title>
    </head>
    <body style="margin: 0; padding: 0; font-family: Arial, sans-serif; background-
color: #f4f4f4;">
      <table width="100%" cellpadding="0" cellspacing="0" style="background-color:
#f4f4f4;">
        <tr>
          <td align="center" style="padding: 20px 0;">
            <table width="600" cellpadding="0" cellspacing="0" style="background-col-
or: #ffffff; border-radius: 8px; overflow: hidden; box-shadow: 0 2px 4px
rgba(0,0,0,0.1);">

              <!-- Cabeçalho com branding -->
              <tr>
                <td style="background-color: ${primaryColor}; padding: 30px; text-
align: center;">
                  ${logoUrl ? `<img src="${logoUrl}" alt="Logo" style="max-width:
150px; height: auto;">` : ''}
                </td>
              </tr>

              <!-- Conteúdo -->
              <tr>
                <td style="padding: 40px 30px;">
                  <h2 style="color: #333333; margin-top: 0;">${subject}</h2>
                  <div style="color: #666666; line-height: 1.6;">
                    ${content}
                  </div>
                </td>
              </tr>

              <!-- Rodapé -->
              <tr>
                <td style="background-color: #f8f8f8; padding: 20px 30px; text-align:
center; border-top: 1px solid #e0e0e0;">
                  <p style="margin: 0; color: #999999; font-size: 12px;">
                    ${branding.emailFooterText || 'Esta mensagem foi enviada automat-
icamente. Por favor, não responda.'}
                  </p>
                  <p style="margin: 10px 0 0 0; color: #999999; font-size: 12px;">
                    © ${new Date().getFullYear()} ${branding.company?.name || 'Organ-
iZen'}. Todos os direitos reservados.
                  </p>
                </td>
              </tr>
```

```
            </table>
          </td>
        </tr>
      </table>
    </body>
    </html>
  `;
}
```

## Uso do Template

```
import { createBrandedEmailHTML } from '@/lib/email/branded-template';
import { sendEmail } from '@/lib/email/send';

async function sendNotification(userId: string) {
  const user = await prisma.user.findUnique({
    where: { id: userId },
    include: {
      company: {
        include: { branding: true }
      }
    }
  });

  const branding = user?.company?.branding;
  if (!branding) return;

  const emailHTML = createBrandedEmailHTML(
    branding,
    'Nova Tarefa Atribuída',
    `<p>Olá ${user.name},</p>
     <p>Foi-lhe atribuída uma nova tarefa: <strong>Revisar Relatório Mensal</strong></
p>
     <p>Por favor, aceda ao sistema para mais detalhes.</p>`
  );

  await sendEmail({
    to: user.email,
    subject: 'Nova Tarefa Atribuída',
    html: emailHTML
  });
}
```

---

## ✅ Validação de Cores

### lib/branding/validate-colors.ts

```
import tinycolor from 'tinycolor2';

export function isValidColor(color: string): boolean {
  return tinycolor(color).isValid();
}

export function getContrastRatio(color1: string, color2: string): number {
  const tc1 = tinycolor(color1);
  const tc2 = tinycolor(color2);
  return tinycolor.readability(tc1, tc2);
}

export function hasGoodContrast(
  color1: string,
  color2: string,
  level: 'AA' | 'AAA' = 'AA'
): boolean {
  const ratio = getContrastRatio(color1, color2);
  return level === 'AA' ? ratio >= 4.5 : ratio >= 7;
}

export function suggestTextColor(backgroundColor: string): string {
  const color = tinycolor(backgroundColor);
  return color.isLight() ? '#000000' : '#FFFFFF';
}

export function validateBrandingColors(branding: {
  primaryColor: string;
  secondaryColor?: string;
  backgroundColor?: string;
}): { valid: boolean; warnings: string[] } {
  const warnings: string[] = [];

  // Validar formato
  if (!isValidColor(branding.primaryColor)) {
    return { valid: false, warnings: ['Cor primária inválida'] };
  }

  // Validar contraste
  const bgColor = branding.backgroundColor || '#FFFFFF';
  if (!hasGoodContrast(branding.primaryColor, bgColor)) {
    warnings.push('Contraste baixo entre cor primária e fundo. Pode dificultar a
leitura.');
  }

  if (branding.secondaryColor) {
    if (!isValidColor(branding.secondaryColor)) {
      return { valid: false, warnings: ['Cor secundária inválida'] };
    }

    if (!hasGoodContrast(branding.secondaryColor, bgColor)) {
      warnings.push('Contraste baixo entre cor secundária e fundo.');
    }
  }

  return { valid: true, warnings };
}
```

Documento criado em: 20 de Outubro de 2025
Status: Referência de código - Exemplos prontos para implementação