🎯 Guia de Implementação do Sistema de Pagamento Stripe

📋 Índice

---

1. Setup Inicial

1.1 Instalar Dependências

```
cd /home/ubuntu/timming_loveu
npm install stripe @stripe/stripe-js
```

1.2 Criar Conta Stripe

1. Acesse https://dashboard.stripe.com/register
2. Crie uma conta
3. Ative o modo de teste
4. Obtenha as chaves:
   - **Publishable Key** (começa com `pk_test_`)
   - **Secret Key** (começa com `sk_test_`)

1.3 Configurar Variáveis de Ambiente

Adicione ao `.env`:

```
# Stripe Configuration
STRIPE_SECRET_KEY=sk_test_seu_secret_key_aqui
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY=pk_test_seu_publishable_key_aqui
STRIPE_WEBHOOK_SECRET=whsec_seu_webhook_secret_aqui
```

---

2. Configuração do Stripe

2.1 Criar Produtos no Stripe Dashboard

1. Acesse https://dashboard.stripe.com/test/products
2. Clique em “Add product”

Produto 1: Plano Mensal
- Nome: `Timming LoveU - Plano Mensal`
- Descrição: `Acesso completo à plataforma por 1 mês`
- Preço: `R$ 9,90 BRL`
- Tipo: `Recurring`
- Intervalo: `Monthly`
- ID do Price: Anote o `price_xxx` gerado

Produto 2: Plano Anual
- Nome: `Timming LoveU - Plano Anual`
- Descrição: `Acesso completo à plataforma por 12 meses com desconto`
- Preço: `R$ 99,00 BRL`
- Tipo: `Recurring`
- Intervalo: `Yearly`
- ID do Price: Anote o `price_yyy` gerado

2.2 Configurar Webhooks

1. Acesse https://dashboard.stripe.com/test/webhooks
2. Clique em “Add endpoint”
3. URL do endpoint:
   - Local (com Stripe CLI): `http://localhost:3000/api/payment/webhook`
   - Produção: `https://seu-dominio.com/api/payment/webhook`
4. Selecione eventos:
   - `checkout.session.completed`
   - `customer.subscription.created`
   - `customer.subscription.updated`
   - `customer.subscription.deleted`
   - `invoice.payment_succeeded`
   - `invoice.payment_failed`
5. Copie o **Webhook signing secret** (começa com `whsec_`)

---

3. Mudanças no Database

3.1 Atualizar Schema Prisma

Edite `prisma/schema.prisma` e adicione:

```
// Modelo de Assinatura do Usuário
model UserSubscription {
  id                   String    @id @default(cuid())
  userId               String    @unique @map("user_id")
  planoId              String    @map("plano_id")
  stripeCustomerId     String?   @unique @map("stripe_customer_id")
  stripeSubscriptionId String?   @unique @map("stripe_subscription_id")
  stripePriceId        String?   @map("stripe_price_id")
  status               String    // active, canceled, past_due, trialing, incomplete,
incomplete_expired
  currentPeriodStart   DateTime  @map("current_period_start")
  currentPeriodEnd     DateTime  @map("current_period_end")
  cancelAtPeriodEnd    Boolean   @default(false) @map("cancel_at_period_end")
  canceledAt           DateTime? @map("canceled_at")
  trialStart           DateTime? @map("trial_start")
  trialEnd             DateTime? @map("trial_end")
  createdAt            DateTime  @default(now()) @map("created_at")
  updatedAt            DateTime  @updatedAt @map("updated_at")

  user  User            @relation(fields: [userId], references: [id], onDelete: Cas-
cade)
  plano PlanoAssinatura @relation(fields: [planoId], references: [id])

  @@index([userId])
  @@index([stripeCustomerId])
  @@index([stripeSubscriptionId])
  @@map("user_subscriptions")
}
```

Atualize o modelo `User`:

```
model User {
  // ... campos existentes ...

  subscription UserSubscription?

  // resto do modelo...
}
```

Atualize o modelo `PlanoAssinatura`:

```
model PlanoAssinatura {
  // ... campos existentes ...

  subscriptions UserSubscription[]

  // resto do modelo...
}
```

3.2 Criar e Executar Migration

```
# Gerar migration
npx prisma migrate dev --name add_user_subscription

# Gerar Prisma Client atualizado
npx prisma generate
```

3.3 Seed de Planos

Crie ou atualize `scripts/seed.ts`:

```
import { PrismaClient } from '@prisma/client'

const prisma = new PrismaClient()

async function main() {
  console.log('Seeding database...')

  // Criar planos de assinatura
  const planoMensal = await prisma.planoAssinatura.upsert({
    where: { id: 'plano-mensal' },
    update: {},
    create: {
      id: 'plano-mensal',
      nome: 'Plano Mensal',
      preco: 9.90,
      descricao: 'Acesso completo à plataforma por 1 mês',
      duracaoMeses: 1,
      maxPaginas: 1,
      stripePriceId: 'price_XXX', // Substituir com ID real do Stripe
      stripeProductId: 'prod_XXX', // Substituir com ID real do Stripe
      ativo: true
    }
  })

  const planoAnual = await prisma.planoAssinatura.upsert({
    where: { id: 'plano-anual' },
    update: {},
    create: {
      id: 'plano-anual',
      nome: 'Plano Anual',
      preco: 99.00,
      descricao: 'Acesso completo à plataforma por 12 meses com desconto',
      duracaoMeses: 12,
      maxPaginas: 1,
      stripePriceId: 'price_YYY', // Substituir com ID real do Stripe
      stripeProductId: 'prod_YYY', // Substituir com ID real do Stripe
      ativo: true
    }
  })

  console.log('✅ Planos criados:', { planoMensal, planoAnual })
}

main()
  .then(() => prisma.$disconnect())
  .catch((e) => {
    console.error(e)
    prisma.$disconnect()
    process.exit(1)
  })
```

Executar seed:

```
npm run prisma:seed
```

---

4. Implementação Backend

4.1 Criar Helpers do Stripe

Crie `lib/stripe.ts`:

```
import Stripe from 'stripe'

if (!process.env.STRIPE_SECRET_KEY) {
  throw new Error('STRIPE_SECRET_KEY não configurado')
}

export const stripe = new Stripe(process.env.STRIPE_SECRET_KEY, {
  apiVersion: '2024-10-28.acacia',
  typescript: true
})

export const STRIPE_CONFIG = {
  currency: 'brl',
  successUrl: `${process.env.NEXTAUTH_URL}/dashboard?payment=success`,
  cancelUrl: `${process.env.NEXTAUTH_URL}/pricing?payment=canceled`
}
```

4.2 Criar Helper de Validação de Plano

Crie `lib/subscription-helpers.ts`:

```
import { prisma } from './db'

export async function checkUserSubscription(userId: string) {
  const subscription = await prisma.userSubscription.findUnique({
    where: { userId },
    include: { plano: true }
  })

  if (!subscription) {
    return {
      isActive: false,
      subscription: null
    }
  }

  const isActive =
    subscription.status === 'active' &&
    subscription.currentPeriodEnd > new Date()

  return {
    isActive,
    subscription,
    daysUntilExpiration: isActive
      ? Math.ceil((subscription.currentPeriodEnd.getTime() - Date.now()) / (1000 * 60
* 60 * 24))
      : 0
  }
}

export async function requireActiveSubscription(userId: string) {
  const { isActive } = await checkUserSubscription(userId)

  if (!isActive) {
    throw new Error('Plano inativo ou expirado')
  }

  return true
}
```

4.3 Criar API Routes

4.3.1 GET /api/payment/plans

Crie `app/api/payment/plans/route.ts`:

```
import { NextResponse } from 'next/server'
import { prisma } from '@/lib/db'

export async function GET() {
  try {
    const planos = await prisma.planoAssinatura.findMany({
      where: { ativo: true },
      orderBy: { preco: 'asc' }
    })

    return NextResponse.json({ planos })
  } catch (error) {
    console.error('Erro ao buscar planos:', error)
    return NextResponse.json(
      { error: 'Erro ao buscar planos' },
      { status: 500 }
    )
  }
}
```

4.3.2 POST /api/payment/create-checkout

Crie `app/api/payment/create-checkout/route.ts`:

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

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

    const { priceId } = await req.json()

    if (!priceId) {
      return NextResponse.json({ error: 'priceId é obrigatório' }, { status: 400 })
    }

    // Buscar plano no banco
    const plano = await prisma.planoAssinatura.findFirst({
      where: { stripePriceId: priceId }
    })

    if (!plano) {
      return NextResponse.json({ error: 'Plano não encontrado' }, { status: 404 })
    }

    // Buscar ou criar Customer no Stripe
    const user = await prisma.user.findUnique({
      where: { id: session.user.id },
      include: { subscription: true }
    })

    let customerId = user?.subscription?.stripeCustomerId

    if (!customerId) {
      const customer = await stripe.customers.create({
        email: user!.email,
        name: user!.name || undefined,
        metadata: {
          userId: user!.id
        }
      })
      customerId = customer.id
    }

    // Criar Checkout Session
    const checkoutSession = await stripe.checkout.sessions.create({
      customer: customerId,
      mode: 'subscription',
      payment_method_types: ['card'],
      line_items: [
        {
          price: priceId,
          quantity: 1
        }
      ],
      success_url: `${STRIPE_CONFIG.successUrl}&session_id={CHECKOUT_SESSION_ID}`,
      cancel_url: STRIPE_CONFIG.cancelUrl,
      metadata: {
```

```
        userId: user!.id,
        planoId: plano.id
      },
      subscription_data: {
        trial_period_days: 7, // 7 dias de teste grátis
        metadata: {
          userId: user!.id,
          planoId: plano.id
        }
      }
    })

    return NextResponse.json({ url: checkoutSession.url })
  } catch (error: any) {
    console.error('Erro ao criar checkout:', error)
    return NextResponse.json(
      { error: error.message || 'Erro ao criar checkout' },
      { status: 500 }
    )
  }
}
```

4.3.3 POST /api/payment/webhook

Crie `app/api/payment/webhook/route.ts`:

```
import { NextRequest, NextResponse } from 'next/server'
import { headers } from 'next/headers'
import Stripe from 'stripe'
import { stripe } from '@/lib/stripe'
import { prisma } from '@/lib/db'

const webhookSecret = process.env.STRIPE_WEBHOOK_SECRET!

export async function POST(req: NextRequest) {
  try {
    const body = await req.text()
    const headersList = headers()
    const signature = headersList.get('stripe-signature')!

    let event: Stripe.Event

    // Verificar assinatura do webhook
    try {
      event = stripe.webhooks.constructEvent(body, signature, webhookSecret)
    } catch (err: any) {
      console.error('❌ Webhook signature verification failed:', err.message)
      return NextResponse.json(
        { error: 'Webhook signature verification failed' },
        { status: 400 }
      )
    }

    // Processar evento
    console.log('✅ Webhook recebido:', event.type)

    switch (event.type) {
      case 'checkout.session.completed':
        await handleCheckoutCompleted(event.data.object as Stripe.Checkout.Session)
        break

      case 'customer.subscription.created':
      case 'customer.subscription.updated':
        await handleSubscriptionUpdated(event.data.object as Stripe.Subscription)
        break

      case 'customer.subscription.deleted':
        await handleSubscriptionDeleted(event.data.object as Stripe.Subscription)
        break

      case 'invoice.payment_succeeded':
        await handlePaymentSucceeded(event.data.object as Stripe.Invoice)
        break

      case 'invoice.payment_failed':
        await handlePaymentFailed(event.data.object as Stripe.Invoice)
        break

      default:
        console.log(`Evento não tratado: ${event.type}`)
    }

    return NextResponse.json({ received: true })
  } catch (error: any) {
    console.error('Erro no webhook:', error)
    return NextResponse.json(
      { error: error.message || 'Erro ao processar webhook' },
      { status: 500 }
```

```
    )
  }
}

async function handleCheckoutCompleted(session: Stripe.Checkout.Session) {
  const userId = session.metadata?.userId
  const planoId = session.metadata?.planoId

  if (!userId || !planoId) {
    console.error('Metadata incompleto no checkout session')
    return
  }

  console.log('💰 Checkout completado para usuário:', userId)

  // Buscar subscription do Stripe
  const subscription = await stripe.subscriptions.retrieve(session.subscription as str
ing)

  // Criar ou atualizar UserSubscription
  await prisma.userSubscription.upsert({
    where: { userId },
    create: {
      userId,
      planoId,
      stripeCustomerId: session.customer as string,
      stripeSubscriptionId: subscription.id,
      stripePriceId: subscription.items.data[0].price.id,
      status: subscription.status,
      currentPeriodStart: new Date(subscription.current_period_start * 1000),
      currentPeriodEnd: new Date(subscription.current_period_end * 1000),
      trialStart: subscription.trial_start ? new Date(subscription.trial_start *
1000) : null,
      trialEnd: subscription.trial_end ? new Date(subscription.trial_end * 1000) : nul
l
    },
    update: {
      planoId,
      stripeSubscriptionId: subscription.id,
      stripePriceId: subscription.items.data[0].price.id,
      status: subscription.status,
      currentPeriodStart: new Date(subscription.current_period_start * 1000),
      currentPeriodEnd: new Date(subscription.current_period_end * 1000),
      trialStart: subscription.trial_start ? new Date(subscription.trial_start *
1000) : null,
      trialEnd: subscription.trial_end ? new Date(subscription.trial_end * 1000) : nul
l
    }
  })

  // Atualizar User
  await prisma.user.update({
    where: { id: userId },
    data: {
      planoAtivo: true,
      dataExpiracaoPlano: new Date(subscription.current_period_end * 1000)
    }
  })

  console.log('✅ Assinatura criada/atualizada para usuário:', userId)
}

async function handleSubscriptionUpdated(subscription: Stripe.Subscription) {
```

```
  console.log('🔄 Subscription atualizada:', subscription.id)

  const userSub = await prisma.userSubscription.findUnique({
    where: { stripeSubscriptionId: subscription.id }
  })

  if (!userSub) {
    console.error('UserSubscription não encontrado para subscription:', subscrip-
tion.id)
    return
  }

  // Atualizar UserSubscription
  await prisma.userSubscription.update({
    where: { id: userSub.id },
    data: {
      status: subscription.status,
      currentPeriodStart: new Date(subscription.current_period_start * 1000),
      currentPeriodEnd: new Date(subscription.current_period_end * 1000),
      cancelAtPeriodEnd: subscription.cancel_at_period_end,
      canceledAt: subscription.canceled_at ? new Date(subscription.canceled_at *
1000) : null
    }
  })

  // Atualizar User
  const isActive = subscription.status === 'active'
  await prisma.user.update({
    where: { id: userSub.userId },
    data: {
      planoAtivo: isActive,
      dataExpiracaoPlano: new Date(subscription.current_period_end * 1000)
    }
  })

  console.log('✅ Subscription atualizada')
}

async function handleSubscriptionDeleted(subscription: Stripe.Subscription) {
  console.log('❌ Subscription cancelada:', subscription.id)

  const userSub = await prisma.userSubscription.findUnique({
    where: { stripeSubscriptionId: subscription.id }
  })

  if (!userSub) {
    console.error('UserSubscription não encontrado para subscription:', subscrip-
tion.id)
    return
  }

  // Atualizar UserSubscription
  await prisma.userSubscription.update({
    where: { id: userSub.id },
    data: {
      status: 'canceled',
      canceledAt: new Date()
    }
  })

  // Atualizar User
  await prisma.user.update({
    where: { id: userSub.userId },
```

```
    data: {
      planoAtivo: false
    }
  })

  console.log('✅ Subscription cancelada no banco')
}

async function handlePaymentSucceeded(invoice: Stripe.Invoice) {
  console.log('💳 Pagamento bem sucedido:', invoice.id)

  // Você pode adicionar lógica adicional aqui, como enviar email de confirmação
}

async function handlePaymentFailed(invoice: Stripe.Invoice) {
  console.log('⚠️ Pagamento falhou:', invoice.id)

  // Você pode adicionar lógica adicional aqui, como enviar email de alerta
}
```

4.3.4 POST /api/payment/portal

Crie `app/api/payment/portal/route.ts`:

```
import { NextRequest, NextResponse } from 'next/server'
import { getServerSession } from 'next-auth'
import { authOptions } from '@/lib/auth/auth-options'
import { stripe } from '@/lib/stripe'
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 })
    }

    // Buscar subscription do usuário
    const userSub = await prisma.userSubscription.findUnique({
      where: { userId: session.user.id }
    })

    if (!userSub?.stripeCustomerId) {
      return NextResponse.json(
        { error: 'Nenhuma assinatura encontrada' },
        { status: 404 }
      )
    }

    // Criar portal session
    const portalSession = await stripe.billingPortal.sessions.create({
      customer: userSub.stripeCustomerId,
      return_url: `${process.env.NEXTAUTH_URL}/dashboard`
    })

    return NextResponse.json({ url: portalSession.url })
  } catch (error: any) {
    console.error('Erro ao criar portal session:', error)
    return NextResponse.json(
      { error: error.message || 'Erro ao acessar portal' },
      { status: 500 }
    )
  }
}
```

4.3.5 GET /api/payment/subscription-status

Crie `app/api/payment/subscription-status/route.ts`:

```
import { NextResponse } from 'next/server'
import { getServerSession } from 'next-auth'
import { authOptions } from '@/lib/auth/auth-options'
import { checkUserSubscription } from '@/lib/subscription-helpers'

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

    const subscriptionData = await checkUserSubscription(session.user.id)

    return NextResponse.json(subscriptionData)
  } catch (error: any) {
    console.error('Erro ao buscar status da assinatura:', error)
    return NextResponse.json(
      { error: error.message || 'Erro ao buscar status' },
      { status: 500 }
    )
  }
}
```

---

5. Implementação Frontend

5.1 Criar Página de Planos

Crie `app/pricing/page.tsx`:

```tsx
'use client'

import { useState, useEffect } from 'react'
import { useSession } from 'next-auth/react'
import { useRouter } from 'next/navigation'
import { Button } from '@/components/ui/button'
import { Card, CardContent, CardDescription, CardFooter, CardHeader, CardTitle } from
'@/components/ui/card'
import { Badge } from '@/components/ui/badge'
import { Check, Loader2, ArrowRight } from 'lucide-react'
import { toast } from 'sonner'
import Link from 'next/link'

interface Plano {
  id: string
  nome: string
  preco: number
  descricao: string | null
  duracaoMeses: number
  stripePriceId: string | null
}

export default function PricingPage() {
  const { data: session, status } = useSession()
  const router = useRouter()
  const [planos, setPlanos] = useState<Plano[]>([])
  const [loading, setLoading] = useState(true)
  const [subscribingTo, setSubscribingTo] = useState<string | null>(null)

  useEffect(() => {
    loadPlanos()
  }, [])

  async function loadPlanos() {
    try {
      const res = await fetch('/api/payment/plans')
      const data = await res.json()
      setPlanos(data.planos || [])
    } catch (error) {
      console.error('Erro ao carregar planos:', error)
      toast.error('Erro ao carregar planos')
    } finally {
      setLoading(false)
    }
  }

  async function handleSubscribe(plano: Plano) {
    if (status !== 'authenticated') {
      router.push('/login?callbackUrl=/pricing')
      return
    }

    if (!plano.stripePriceId) {
      toast.error('Plano indisponível no momento')
      return
    }

    setSubscribingTo(plano.id)

    try {
      const res = await fetch('/api/payment/create-checkout', {
        method: 'POST',
```

```
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ priceId: plano.stripePriceId })
      })

      const data = await res.json()

      if (!res.ok) {
        throw new Error(data.error || 'Erro ao criar checkout')
      }

      // Redirecionar para Stripe Checkout
      window.location.href = data.url
    } catch (error: any) {
      console.error('Erro ao iniciar assinatura:', error)
      toast.error(error.message || 'Erro ao processar pagamento')
      setSubscribingTo(null)
    }
  }

  if (loading) {
    return (
      <div className="flex items-center justify-center min-h-screen">
        <Loader2 className="w-8 h-8 animate-spin text-pink-500" />
      </div>
    )
  }

  return (
    <div className="min-h-screen bg-gradient-to-b from-pink-50 to-white py-16 px-4">
      <div className="max-w-6xl mx-auto">
        {/* Header */}
        <div className="text-center mb-12">
          <h1 className="text-4xl md:text-5xl font-bold text-gray-900 mb-4">
            Escolha Seu Plano
          </h1>
          <p className="text-xl text-gray-600 max-w-2xl mx-auto">
            Comece com 7 dias grátis. Cancele quando quiser, sem compromisso.
          </p>
        </div>

        {/* Planos */}
        <div className="grid md:grid-cols-2 gap-8 max-w-4xl mx-auto">
          {planos.map((plano) => {
            const isAnual = plano.duracaoMeses === 12
            const precoMensal = isAnual ? plano.preco / 12 : plano.preco
            const economia = isAnual ? ((9.90 * 12) - plano.preco) : 0

            return (
              <Card
                key={plano.id}
                className={`relative ${isAnual ? 'border-pink-500 border-2 shadow-
lg' : ''}`}
              >
                {isAnual && (
                  <Badge className="absolute -top-3 left-1/2 -translate-x-1/2 bg-
pink-500">
                    Mais Popular
                  </Badge>
                )}

                <CardHeader>
                  <CardTitle className="text-2xl">{plano.nome}</CardTitle>
                  <CardDescription>{plano.descricao}</CardDescription>
```

```
            </CardHeader>

            <CardContent>
              <div className="mb-6">
                <div className="flex items-baseline gap-2">
                  <span className="text-4xl font-bold">
                    R$ {plano.preco.toFixed(2)}
                  </span>
                  <span className="text-gray-600">
                    / {plano.duracaoMeses === 1 ? 'mês' : 'ano'}
                  </span>
                </div>
                {isAnual && (
                  <div className="mt-2">
                    <p className="text-sm text-gray-600">
                      R$ {precoMensal.toFixed(2)}/mês
                    </p>
                    <p className="text-sm text-green-600 font-semibold">
                      Economize R$ {economia.toFixed(2)}
                    </p>
                  </div>
                )}
              </div>

              <ul className="space-y-3">
                <li className="flex items-start gap-2">
                  <Check className="w-5 h-5 text-green-500 shrink-0 mt-0.5" />
                  <span>1 página personalizada de casal</span>
                </li>
                <li className="flex items-start gap-2">
                  <Check className="w-5 h-5 text-green-500 shrink-0 mt-0.5" />
                  <span>Galeria ilimitada de fotos e vídeos</span>
                </li>
                <li className="flex items-start gap-2">
                  <Check className="w-5 h-5 text-green-500 shrink-0 mt-0.5" />
                  <span>Música personalizada</span>
                </li>
                <li className="flex items-start gap-2">
                  <Check className="w-5 h-5 text-green-500 shrink-0 mt-0.5" />
                  <span>Cronômetro do relacionamento</span>
                </li>
                <li className="flex items-start gap-2">
                  <Check className="w-5 h-5 text-green-500 shrink-0 mt-0.5" />
                  <span>Compartilhamento ilimitado</span>
                </li>
                {isAnual && (
                  <>
                    <li className="flex items-start gap-2">
                      <Check className="w-5 h-5 text-green-500 shrink-0 mt-0.5" />
                      <span className="font-semibold">2 meses grátis</span>
                    </li>
                    <li className="flex items-start gap-2">
                      <Check className="w-5 h-5 text-green-500 shrink-0 mt-0.5" />
                      <span className="font-semibold">Suporte prioritário</span>
                    </li>
                  </>
                )}
              </ul>
            </CardContent>

            <CardFooter>
              <Button
                size="lg"
```

```
                    className={`w-full ${isAnual ? 'bg-pink-500 hover:bg-pink-600' : '
'}`}
                    onClick={() => handleSubscribe(plano)}
                    disabled={subscribingTo !== null}
                  >
                    {subscribingTo === plano.id ? (
                      <>
                        <Loader2 className="w-4 h-4 mr-2 animate-spin" />
                        Processando...
                      </>
                    ) : (
                      <>
                        Começar Agora
                        <ArrowRight className="w-4 h-4 ml-2" />
                      </>
                    )}
                  </Button>
                </CardFooter>
              </Card>
            )
          })}
        </div>

        {/* Features Adicionais */}
        <div className="mt-16 text-center">
          <h3 className="text-2xl font-bold mb-4">Todos os planos incluem:</h3>
          <div className="grid md:grid-cols-3 gap-6 max-w-3xl mx-auto">
            <div>
              <div className="text-3xl mb-2">🎁</div>
              <h4 className="font-semibold mb-1">7 dias grátis</h4>
              <p className="text-sm text-gray-600">Teste sem compromisso</p>
            </div>
            <div>
              <div className="text-3xl mb-2">🔒</div>
              <h4 className="font-semibold mb-1">Pagamento seguro</h4>
              <p className="text-sm text-gray-600">Criptografado com Stripe</p>
            </div>
            <div>
              <div className="text-3xl mb-2">🚫</div>
              <h4 className="font-semibold mb-1">Cancele quando quiser</h4>
              <p className="text-sm text-gray-600">Sem multas ou taxas</p>
            </div>
          </div>
        </div>

        {/* Footer */}
        <div className="mt-12 text-center">
          <Link href="/" className="text-pink-500 hover:underline">
            ← Voltar para home
          </Link>
        </div>
      </div>
    </div>
  )
}
```

5.2 Adicionar Badge de Status no Dashboard

Atualize `app/dashboard/components/dashboard-client.tsx` para incluir status da assinatura:

```
// Adicionar ao início do componente
const [subscriptionStatus, setSubscriptionStatus] = useState<any>(null)

useEffect(() => {
  loadSubscriptionStatus()
}, [])

async function loadSubscriptionStatus() {
  try {
    const res = await fetch('/api/payment/subscription-status')
    const data = await res.json()
    setSubscriptionStatus(data)
  } catch (error) {
    console.error('Erro ao carregar status da assinatura:', error)
  }
}

// Adicionar no JSX (no header do dashboard):
{subscriptionStatus?.isActive ? (
  <Badge variant="success" className="bg-green-500">
    Plano Ativo
  </Badge>
) : (
  <Badge variant="destructive">
    Plano Inativo
  </Badge>
)}
```

5.3 Criar Botão de Gerenciar Assinatura

Adicione no dashboard:

```
async function handleManageSubscription() {
  try {
    const res = await fetch('/api/payment/portal', { method: 'POST' })
    const data = await res.json()

    if (data.url) {
      window.location.href = data.url
    }
  } catch (error) {
    toast.error('Erro ao acessar portal de gerenciamento')
  }
}

// No JSX:
<Button onClick={handleManageSubscription}>
  Gerenciar Assinatura
</Button>
```

---

6. Testing

6.1 Testing Local com Stripe CLI

Instale o Stripe CLI:

```
# macOS
brew install stripe/stripe-cli/stripe

# Linux
wget https://github.com/stripe/stripe-cli/releases/download/v1.17.0/
stripe_1.17.0_linux_x86_64.tar.gz
tar -xvf stripe_1.17.0_linux_x86_64.tar.gz
sudo mv stripe /usr/local/bin/
```

Login e forward webhooks:

```
# Login
stripe login

# Forward webhooks para localhost
stripe listen --forward-to localhost:3000/api/payment/webhook
```

Isso vai gerar um webhook secret temporário. Use-o no `.env`:

```
STRIPE_WEBHOOK_SECRET=whsec_temporario_do_cli
```

6.2 Testar Fluxo de Pagamento

Cartões de teste do Stripe:

- **Sucesso:** `4242 4242 4242 4242`
- **Falha:** `4000 0000 0000 0002`
- **3D Secure:** `4000 0025 0000 3155`

Dados de teste:
- CVV: Qualquer 3 dígitos
- Data: Qualquer data futura
- CEP: Qualquer valor

6.3 Testar Webhooks Manualmente

```
# Simular checkout.session.completed
stripe trigger checkout.session.completed

# Simular subscription.deleted
stripe trigger customer.subscription.deleted
```

6.4 Testes Automatizados

Crie `__tests__/api/payment.test.ts`:

```
import { POST as createCheckout } from '@/app/api/payment/create-checkout/route'
import { getServerSession } from 'next-auth'

jest.mock('next-auth')
jest.mock('@/lib/stripe')

describe('Payment API', () => {
  beforeEach(() => {
    jest.clearAllMocks()
  })

  describe('POST /api/payment/create-checkout', () => {
    it('should create checkout session for authenticated user', async () => {
      (getServerSession as jest.Mock).mockResolvedValue({
        user: { id: 'user-123', email: 'test@example.com' }
      })

      const req = new Request('http://localhost/api/payment/create-checkout', {
        method: 'POST',
        body: JSON.stringify({ priceId: 'price_test123' })
      })

      const res = await createCheckout(req as any)
      const data = await res.json()

      expect(res.status).toBe(200)
      expect(data).toHaveProperty('url')
    })

    it('should return 401 for unauthenticated user', async () => {
      (getServerSession as jest.Mock).mockResolvedValue(null)

      const req = new Request('http://localhost/api/payment/create-checkout', {
        method: 'POST',
        body: JSON.stringify({ priceId: 'price_test123' })
      })

      const res = await createCheckout(req as any)
      expect(res.status).toBe(401)
    })
  })
})
```

---

7. Deploy

7.1 Configurar Variáveis de Ambiente em Produção

Vercel:

1. Acesse projeto no dashboard da Vercel
2. Settings → Environment Variables
3. Adicione:

- `STRIPE_SECRET_KEY` (produção: `sk_live_...`)
- `NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY` (`pk_live_...`)
- `STRIPE_WEBHOOK_SECRET` (produção: `whsec_...`)

Docker:
Adicione ao `.env.production`:

```
STRIPE_SECRET_KEY=sk_live_...
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY=pk_live_...
STRIPE_WEBHOOK_SECRET=whsec_...
```

7.2 Configurar Webhook em Produção

1. Acesse https://dashboard.stripe.com/webhooks
2. Adicione endpoint: `https://seu-dominio.com/api/payment/webhook`
3. Selecione os mesmos eventos do teste
4. Copie o webhook secret
5. Atualize variável de ambiente

7.3 Ativar Modo Live no Stripe

1. Complete o onboarding da conta Stripe
2. Ative o modo Live
3. Crie os produtos/preços em modo Live
4. Atualize IDs no banco de dados (seed de produção)

7.4 Deploy

```
# Vercel
vercel --prod

# Docker
docker-compose -f docker-compose.prod.yml up -d

# Manual
npm run build
npm run start
```

7.5 Verificar Saúde da Aplicação

```
curl https://seu-dominio.com/api/health
```

7.6 Monitorar Webhooks

Acesse https://dashboard.stripe.com/webhooks e monitore:
- Eventos recebidos
- Falhas
- Latência

🎉 Pronto!

Seu sistema de pagamento está implementado e funcionando!

Próximos Passos:

1. ✅ Testar fluxo completo em produção

2. ✅ Configurar alertas de pagamento falho
3. ✅ Implementar emails transacionais (confirmação, expiração, etc.)
4. ✅ Adicionar analytics (conversão, MRR, churn)
5. ✅ Criar dashboard admin para gerenciar assinaturas

Recursos Úteis:

- Stripe Docs (https://stripe.com/docs)
- Stripe Testing (https://stripe.com/docs/testing)
- Webhooks Best Practices (https://stripe.com/docs/webhooks/best-practices)
- Next.js API Routes (https://nextjs.org/docs/api-routes/introduction)

Autor: DeepAgent - Abacus.AI
Data: 23/10/2024
Versão: 1.0