Arquitetura de Gateways de Pagamento - Clivus

Visão Geral

O Clivus implementa uma **arquitetura genérica de gateways de pagamento** que permite integração com múltiplos provedores (Efi, Stripe, Mercado Pago, etc) através de uma interface comum.

Características Principais

✅ **Configuração Global**: Um único gateway ativo para todo o sistema (não por tenant)
✅ **Múltiplos Provedores**: Suporte para diferentes gateways através de adapters
✅ **Provisionamento Automático**: Criação de usuários e liberação de acesso após pagamento
✅ **Email Automático**: Envio de credenciais de acesso por email
✅ **Idempotência**: Webhooks são processados apenas uma vez

Estrutura do Projeto

```
lib/payment/
├── types.ts              # Interfaces e tipos genéricos
├── orchestrator.ts       # Orquestrador de gateways
└── adapters/
    └── efi.ts            # Adapter EFI (Gerencianet)

app/api/payments/
├── create-checkout/   # Endpoint genérico de checkout
└── webhook/           # Webhook genérico
```

Configuração

Variáveis de Ambiente (Prioridade 1)

```
# Definir o provider ativo
PAYMENT_PROVIDER=efi

# Credenciais EFI
EFI_ENVIRONMENT=sandbox  # ou "production"
EFI_CLIENT_ID=seu_client_id
EFI_CLIENT_SECRET=seu_client_secret

# Credenciais Stripe (exemplo futuro)
STRIPE_ENVIRONMENT=sandbox
STRIPE_SECRET_KEY=sk_test_...
STRIPE_WEBHOOK_SECRET=whsec_...
```

Banco de Dados (Prioridade 2)

Se não houver variáveis de ambiente, o sistema busca configurações no banco:

- **Tabela**: `PaymentGatewayConfig`
- **Condição**: `isActive = true`
- **Painel**: SuperAdmin > Gateways

Fluxo de Pagamento

1️⃣ Cliente clica em “Assinar” na Landing Page

```
Landing Page (#precos)
  ↓
/assinar?plano=basico
```

2️⃣ Usuário preenche dados e clica em “Ir para Pagamento”

```
POST /api/payments/create-checkout
{
  "planSlug": "basico",
  "name": "João Silva",
  "email": "joao@email.com"
}
```

O que acontece:
- Busca o plano no banco
- Obtém gateway ativo via `getActiveGateway()`
- Cria sessão de checkout no gateway (via adapter)
- Cria `PaymentIntent` com status `pending`
- Cria ou atualiza `Lead`
- Retorna `checkoutUrl`

3️⃣ Usuário é redirecionado para o gateway

```
Redireciona para: checkoutUrl
(ex: /checkout-simulado ou URL do gateway real)
```

4️⃣ Usuário confirma pagamento no gateway

5️⃣ Gateway envia webhook para o sistema

```
POST /api/payments/webhook
{
  // Payload específico do gateway
  "evento": "charge.paid",
  "chargeId": "EFI_SANDBOX_...",
  "status": "paid",
  ...
}
```

O que acontece:
- Obtém gateway ativo
- Adapter normaliza o webhook para formato padrão
- Busca `PaymentIntent` correspondente
- Verifica idempotência (`provisionedAt`)
- **Se pagamento aprovado:**
- Atualiza `PaymentIntent` para `paid`
- Chama `liberarAcesso()` que:
- Cria usuário (se novo)
- Ativa módulos do plano
- Cria/atualiza `Subscription`
- Registra `PaymentTransaction`
- Converte `Lead`
- Marca `provisionedAt`
- Envia email com `enviarEmailBoasVindas()`

6️⃣ Usuário recebe email com credenciais

```
Para: joao@email.com
Assunto: Bem-vindo ao Clivus!

Seu acesso foi liberado:
Email: joao@email.com
Senha temporária: ABC123xyz

Link: https://clivus.marcosleandru.com.br/auth/login
```

Interface PaymentGateway

Todos os adapters devem implementar:

```
interface PaymentGateway {
  provider: PaymentProvider;

  createCheckoutSession(params: CreateCheckoutParams): Promise<CreateCheckoutResult>;

  handleWebhook(payload: any, headers: any): Promise<WebhookEvent>;

  validateConfig(): Promise<{ valid: boolean; missingFields?: string[] }>;
}
```

Métodos

`createCheckoutSession`

- **Entrada**: Dados do plano e cliente
- **Saída**: URL de checkout + ID externo
- **Responsabilidade**: Criar cobrança no gateway

`handleWebhook`

- **Entrada**: Payload bruto do gateway
- **Saída**: Evento normalizado (`WebhookEvent`)

- **Responsabilidade**: Validar e normalizar webhook

`validateConfig`

- **Saída**: Status de validação e campos faltantes
- **Responsabilidade**: Verificar se credenciais estão completas

---

Adicionar Novo Gateway

Exemplo: Stripe

1. **Criar adapter**

```
// lib/payment/adapters/stripe.ts
import { PaymentGateway, CreateCheckoutParams, ... } from '../types';

export class StripeAdapter implements PaymentGateway {
  provider = 'stripe' as const;
  private config: { secretKey: string; webhookSecret: string };

  constructor(config: any) {
    this.config = config;
  }

  async validateConfig() {
    const missing = [];
    if (!this.config.secretKey) missing.push('secretKey');
    if (!this.config.webhookSecret) missing.push('webhookSecret');
    return { valid: missing.length === 0, missingFields: missing };
  }

  async createCheckoutSession(params: CreateCheckoutParams) {
    // Lógica específica do Stripe
    const session = await stripe.checkout.sessions.create({...});
    return {
      checkoutUrl: session.url,
      externalId: session.id,
      provider: 'stripe'
    };
  }

  async handleWebhook(payload: any, headers: any) {
    // Validar assinatura
    const signature = headers['stripe-signature'];
    const event = stripe.webhooks.constructEvent(payload, signature, this.con-
fig.webhookSecret);

    // Normalizar para formato padrão
    return {
      eventType: event.type === 'checkout.session.completed' ? 'PAYMENT_APPROVED' : 'P
AYMENT_PENDING',
      externalPaymentId: event.data.object.id,
      email: event.data.object.customer_email,
      ...
    };
  }
}
```

1. **Registrar no orquestrador**

```
// lib/payment/orchestrator.ts
import { StripeAdapter } from './adapters/stripe';

export async function getActiveGateway(): Promise<PaymentGateway> {
  const config = await getGlobalGatewayConfig();

  switch (config.provider) {
    case 'efi':
      return new EfiAdapter(config);

    case 'stripe':  // ← ADICIONAR
      return new StripeAdapter(config);

    default:
      throw new Error(`Gateway não suportado: ${config.provider}`);
  }
}
```

1. **Configurar variáveis**

```
PAYMENT_PROVIDER=stripe
STRIPE_SECRET_KEY=sk_test_...
STRIPE_WEBHOOK_SECRET=whsec_...
```

Models do Banco

PaymentIntent (Novo)

```
model PaymentIntent {
  id                  String   @id @default(uuid())
  tenantId            String?  // Null para checkout público
  status              String   // "pending", "paid", "failed", "canceled"
  planId              String
  planSlug            String
  planName            String
  email               String
  name                String
  amountCents         Int
  gatewayProvider     String   // "efi", "stripe", etc
  gatewayExternalId   String   @unique
  metadata            String?  @db.Text
  provisionedAt       DateTime? // Data de provisionamento
  createdAt           DateTime @default(now())
  updatedAt           DateTime @updatedAt
}
```

Campos importantes:

- `gatewayExternalId`: ID da transação no gateway (usado no webhook)
- `provisionedAt`: Garante idempotência (se preenchido, já foi processado)
- `tenantId`: Null para vendas da landing page

EfiTransaction (Legado)

Continua existindo para compatibilidade, mas novos fluxos usam `PaymentIntent`.

Testes

Simular Pagamento (Sandbox)

1. Acesse: `https://clivus.marcosleandru.com.br/#precos`
2. Clique em “Assinar Agora” em qualquer plano
3. Preencha nome e email
4. Clique em “Ir para Pagamento”
5. Na tela de simulação, clique em “Confirmar Pagamento”
6. Verifique:
   - Usuário criado (verifica no banco `User`)
   - Módulos ativados (`UserModule`)
   - Email enviado (logs no console)
   - Acesso disponível em `/auth/login`

Logs Importantes

```
# Criar checkout
[CREATE CHECKOUT] Início da requisição
[ORCHESTRATOR] Gateway ativo: efi
[EFI ADAPTER] Criando sessão de checkout
[CREATE CHECKOUT] PaymentIntent criado: xxx

# Webhook
[WEBHOOK GENÉRICO] Recebida notificação
[EFI ADAPTER] Processando webhook
[WEBHOOK GENÉRICO] PAGAMENTO APROVADO - Provisionando acesso...
[LIBERAR ACESSO] Novo usuário criado: xxx
[LIBERAR ACESSO] Módulos ativados para plano: Básico
[WEBHOOK GENÉRICO] Email de boas-vindas enviado
```

Troubleshooting

Erro: PAYMENT_CONFIG_INCOMPLETE

Causa: Nenhum gateway configurado ou credenciais faltando

Solução:
1. Verificar variáveis de ambiente (`.env`)
2. Ou configurar no SuperAdmin > Gateways > Efi

Webhook não recebe eventos

Causa: URL do webhook não registrada no gateway

Solução:
1. SuperAdmin > Webhooks > Reinstalar Webhook
2. Ou registrar manualmente: `POST https://api-efi.com.br/webhooks`

Usuário não criado após pagamento

Causa: Erro na função `liberarAcesso()`

Solução:
1. Verificar logs do webhook
2. Verificar se `PaymentIntent.provisionedAt` está null
3. Reprocessar manualmente se necessário

Próximos Passos

- [ ] Implementar adapter Stripe
- [ ] Implementar adapter Mercado Pago
- [ ] Email real (substituir console.log por Resend/SendGrid)
- [ ] Validação de assinatura de webhook em produção
- [ ] Painel SuperAdmin para visualizar `PaymentIntent`
- [ ] Retry automático de webhooks falhados
- [ ] Migrar fluxo antigo `/api/payments/efi/create-charge` para usar nova arquitetura

Contato

Para dúvidas ou sugestões sobre a arquitetura de pagamentos, consulte a documentação técnica em `/docs` ou abra uma issue.