Planejamento da Integração com Mercado Pago

Visão Geral

Este documento detalha o planejamento da integração do e-commerce com o Mercado Pago, uma das principais plataformas de pagamento da América Latina. A integração permitirá processar pagamentos de forma segura e eficiente, oferecendo diversas opções de pagamento aos clientes, como cartão de crédito, boleto bancário, Pix e outros métodos disponíveis no Brasil.

Requisitos da Integração

1. Funcionalidades Principais

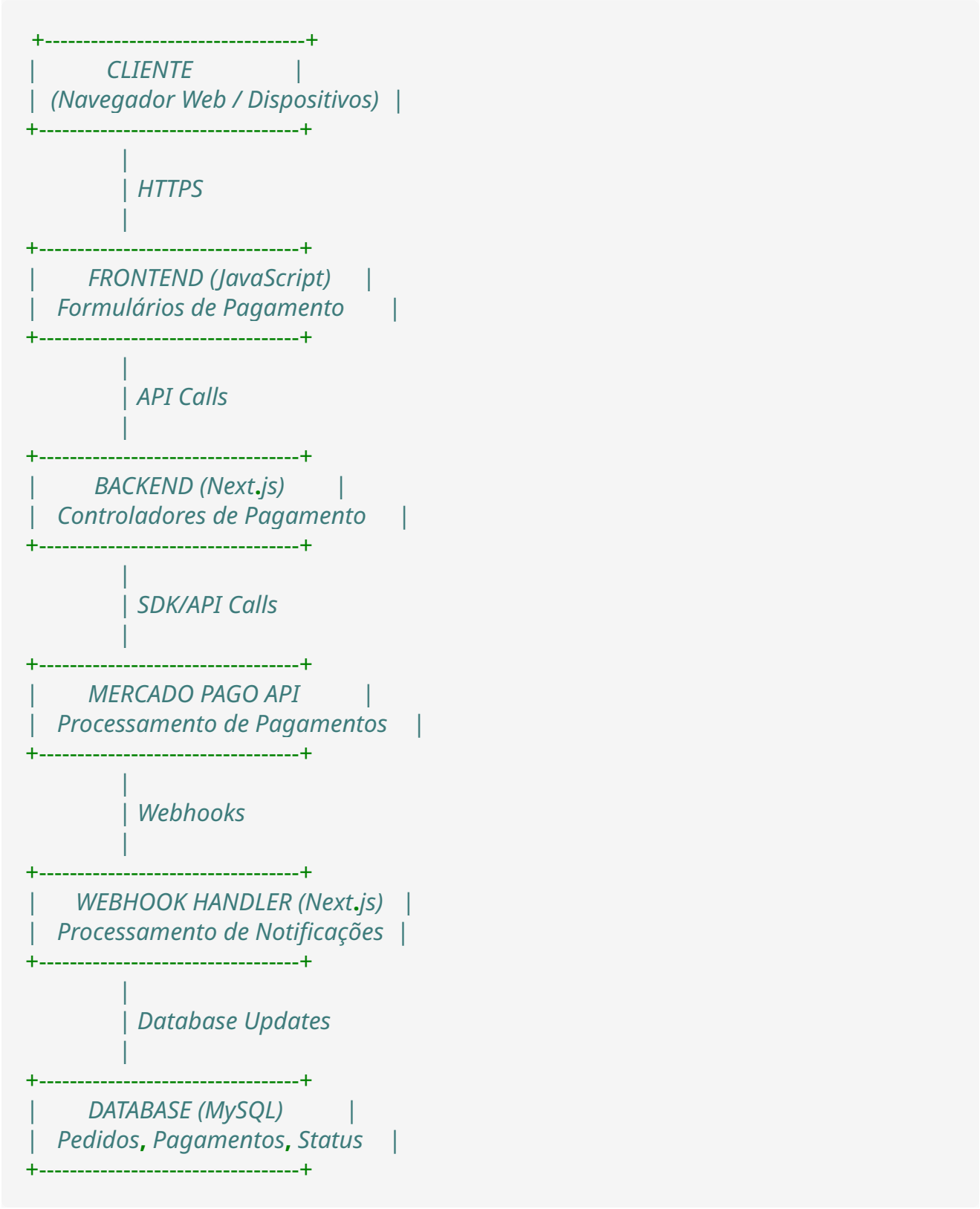
- Processamento de pagamentos via Checkout Pro (redirecionamento)
- Processamento de pagamentos via Checkout Transparente (integrado)
- Recebimento e processamento de notificações de pagamento (webhooks)
- Consulta de status de pagamentos
- Estornos e cancelamentos
- Geração de boletos e códigos Pix
- Parcelamento de compras

2. Requisitos Técnicos

- Conta Mercado Pago para vendedores (conta comercial)
- Credenciais de API (chaves de acesso)
- Certificado SSL para o site (HTTPS obrigatório)
- Configuração de URLs de notificação (webhooks)
- Conformidade com requisitos de segurança PCI DSS

Arquitetura da Integração

Diagrama de Fluxo



Componentes da Integração

1. **Frontend**: Formulários de pagamento e interface do usuário

2. **Backend**: Controladores para criar preferências de pagamento e processar transações
3. **SDK Mercado Pago**: Biblioteca oficial para comunicação com a API
4. **Webhook Handler**: Endpoint para receber notificações de mudanças de status
5. **Banco de Dados**: Armazenamento de informações de pedidos e pagamentos

Configuração do Mercado Pago

1. Criação e Configuração de Conta

1. **Criar Conta Mercado Pago**:
2. Acessar https://www.mercadopago.com.br
3. Registrar uma conta comercial (vendedor)
4. Completar o processo de verificação de identidade
5. **Obter Credenciais**:
6. Acessar o Dashboard do Mercado Pago
7. Navegar para "Credenciais" > "Credenciais de Produção"
8. Obter `ACCESS_TOKEN` e `PUBLIC_KEY`
9. Para ambiente de testes, obter também as credenciais de teste
10. **Configurar Webhooks**:
11. No Dashboard do Mercado Pago, navegar para "Webhooks"
12. Adicionar URL de notificação: `https://seu-dominio.com/api/webhooks/mercadopago`
13. Selecionar eventos para receber notificações (payment, merchant_order, etc.)

2. Variáveis de Ambiente

Configurar as seguintes variáveis de ambiente no projeto Next.js:

```
# Mercado Pago - Produção
MERCADOPAGO_PUBLIC_KEY=APP_USR-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
MERCADOPAGO_ACCESS_TOKEN=APP_USR-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxx

# Mercado Pago - Teste
MERCADOPAGO_TEST_PUBLIC_KEY=TEST-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
MERCADOPAGO_TEST_ACCESS_TOKEN=TEST-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxxxxx
```

```
# URLs
NEXT_PUBLIC_API_URL=https://seu-dominio.com
NEXT_PUBLIC_FRONTEND_URL=https://seu-dominio.com

# Configurações da Loja
STORE_NAME=Nome da Loja
```

Implementação do SDK do Mercado Pago

1. Instalação do SDK

```
npm install mercadopago
```

2. Configuração Básica

```
// /lib/mercadopago/index.js
import { MercadoPagoConfig, Payment, Preference } from 'mercadopago';

// Configuração do cliente Mercado Pago
const client = new MercadoPagoConfig({
  accessToken: process.env.MERCADOPAGO_ACCESS_TOKEN
});

export default client;
```

Implementação do Checkout Pro (Redirecionamento)

O Checkout Pro é a solução mais simples, onde o cliente é redirecionado para a página de pagamento do Mercado Pago.

1. Criação de Preferência de Pagamento

```
// /lib/mercadopago/preference.js
import client from './index';
import { Preference } from 'mercadopago';

export async function createPreference(data) {
  try {
    const preference = new Preference(client);

    const preferenceData = {
      items: data.items.map(item => ({
        id: item.id,
```

```
        title: item.title,
        description: item.description || '',
        picture_url: item.picture_url || '',
        category_id: item.category_id || 'others',
        quantity: item.quantity,
        currency_id: 'BRL',
        unit_price: parseFloat(item.unit_price)
      })),
      payer: {
        name: data.payer.name,
        surname: data.payer.surname,
        email: data.payer.email,
        phone: {
          area_code: data.payer.phone?.area_code || '',
          number: data.payer.phone?.number || ''
        },
        address: {
          zip_code: data.payer.address?.zip_code || '',
          street_name: data.payer.address?.street_name || '',
          street_number: data.payer.address?.street_number || ''
        }
      },
      back_urls: {
        success: `${process.env.NEXT_PUBLIC_FRONTEND_URL}/checkout/success`,
        failure: `${process.env.NEXT_PUBLIC_FRONTEND_URL}/checkout/failure`,
        pending: `${process.env.NEXT_PUBLIC_FRONTEND_URL}/checkout/pending`
      },
      auto_return: 'approved',
      external_reference: data.external_reference,
      notification_url: `${process.env.NEXT_PUBLIC_API_URL}/api/webhooks/
mercadopago`,
      statement_descriptor: process.env.STORE_NAME || 'E-commerce',
      shipments: data.shipments ? {
        cost: parseFloat(data.shipments.cost),
        mode: 'not_specified'
      } : undefined,
      payment_methods: {
        excluded_payment_methods: data.excluded_payment_methods || [],
        excluded_payment_types: data.excluded_payment_types || [],
        installments: data.installments || 12
      }
    };

    const response = await preference.create({ body: preferenceData });
    return response;
  } catch (error) {
    console.error('Error creating Mercado Pago preference:', error);
    throw error;
  }
}
```

2. Endpoint para Criar Preferência

```
// /app/api/payment/mercadopago/preferences/route.js
import { NextResponse } from 'next/server';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth/auth';
import { createPreference } from '@/lib/mercadopago/preference';
import prisma from '@/lib/prisma';

export async function POST(request) {
  try {
    const session = await getServerSession(authOptions);

    // Verificar se usuário está autenticado
    if (!session) {
      return NextResponse.json(
        { error: 'Autenticação necessária' },
        { status: 401 }
      );
    }

    const data = await request.json();

    // Validar dados
    if (!data.order_id) {
      return NextResponse.json(
        { error: 'ID do pedido é obrigatório' },
        { status: 400 }
      );
    }

    // Buscar pedido
    const order = await prisma.order.findFirst({
      where: {
        id: data.order_id,
        userId: parseInt(session.user.id)
      },
      include: {
        items: true,
        shippingAddress: true,
        user: {
          select: {
            name: true,
            email: true,
            phone: true
          }
        }
      }
    });

    if (!order) {
```

```
    return NextResponse.json(
      { error: 'Pedido não encontrado' },
      { status: 404 }
    );
  }

  // Preparar dados para preferência
  const preferenceData = {
    items: order.items.map(item => ({
      id: item.sku,
      title: item.productName,
      quantity: item.quantity,
      unit_price: parseFloat(item.unitPrice)
    })),
    payer: {
      name: order.user.name.split(' ')[0],
      surname: order.user.name.split(' ').slice(1).join(' '),
      email: order.user.email,
      phone: {
        area_code: '',
        number: order.user.phone || order.shippingAddress?.phone || ''
      },
      address: {
        zip_code: order.shippingAddress?.zipCode || '',
        street_name: order.shippingAddress?.street || '',
        street_number: order.shippingAddress?.number || ''
      }
    },
    external_reference: order.code,
    shipments: {
      cost: parseFloat(order.shipping)
    }
  };

  // Criar preferência
  const preference = await createPreference(preferenceData);

  // Atualizar pedido com dados da preferência
  await prisma.payment.create({
    data: {
      orderId: order.id,
      method: 'MERCADOPAGO',
      status: 'PENDING',
      amount: parseFloat(order.total),
      installments: 1,
      gateway: 'mercadopago',
      paymentData: preference
    }
  });

  return NextResponse.json({
    preference_id: preference.id,
```

```
      init_point: preference.init_point,
      sandbox_init_point: preference.sandbox_init_point
    });
  } catch (error) {
    console.error('Error creating payment preference:', error);
    return NextResponse.json(
      { error: 'Erro ao criar preferência de pagamento' },
      { status: 500 }
    );
  }
}
```

3. Implementação no Frontend

```
// /js/components/checkout/MercadoPagoCheckout.js
import { useEffect } from 'react';

const MercadoPagoCheckout = ({ preferenceId, onSuccess, onFailure,
onPending }) => {
  useEffect(() => {
    // Carregar script do Mercado Pago
    const script = document.createElement('script');
    script.src = 'https://sdk.mercadopago.com/js/v2';
    script.async = true;

    script.onload = () => {
      // Inicializar SDK
      const mp = new
window.MercadoPago(process.env.NEXT_PUBLIC_MERCADOPAGO_PUBLIC_KEY, {
        locale: 'pt-BR'
      });

      // Inicializar checkout
      mp.checkout({
        preference: {
          id: preferenceId
        },
        render: {
          container: '.checkout-button',
          label: 'Pagar'
        },
        callbacks: {
          onError: (error) => {
            console.error('Mercado Pago error:', error);
            if (onFailure) onFailure(error);
          }
        }
      });
    };
```

```
    document.body.appendChild(script);

    return () => {
      // Limpar script ao desmontar componente
      if (document.body.contains(script)) {
        document.body.removeChild(script);
      }
    };
  }, [preferenceId, onSuccess, onFailure, onPending]);

  return (
    <div className="mercadopago-checkout">
      <div className="checkout-button"></div>
    </div>
  );
};

export default MercadoPagoCheckout;
```

Implementação do Checkout Transparente

O Checkout Transparente permite que o cliente faça o pagamento sem sair do site, proporcionando uma experiência mais integrada.

1. Configuração do CardForm

```
// /js/components/checkout/MercadoPagoCardForm.js
import { useEffect, useState } from 'react';

const MercadoPagoCardForm = ({ onSubmit, onError }) => {
  const [cardForm, setCardForm] = useState(null);
  const [loading, setLoading] = useState(true);
  const [formData, setFormData] = useState({
    cardholderName: '',
    identificationType: '',
    identificationNumber: '',
    installments: 1,
    issuer: ''
  });

  useEffect(() => {
    // Carregar script do Mercado Pago
    const script = document.createElement('script');
    script.src = 'https://sdk.mercadopago.com/js/v2';
    script.async = true;

    script.onload = () => {
      // Inicializar SDK
```

```
      const mp = new
window.MercadoPago(process.env.NEXT_PUBLIC_MERCADOPAGO_PUBLIC_KEY, {
        locale: 'pt-BR'
      });

      // Inicializar formulário de cartão
      const cardForm = mp.cardForm({
        amount: '100.00',
        autoMount: true,
        form: {
          id: 'form-checkout',
          cardholderName: {
            id: 'form-checkout__cardholderName',
            placeholder: 'Nome no cartão',
          },
          cardholderEmail: {
            id: 'form-checkout__cardholderEmail',
            placeholder: 'E-mail',
          },
          cardNumber: {
            id: 'form-checkout__cardNumber',
            placeholder: 'Número do cartão',
          },
          cardExpirationMonth: {
            id: 'form-checkout__cardExpirationMonth',
            placeholder: 'MM',
          },
          cardExpirationYear: {
            id: 'form-checkout__cardExpirationYear',
            placeholder: 'YY',
          },
          securityCode: {
            id: 'form-checkout__securityCode',
            placeholder: 'CVV',
          },
          installments: {
            id: 'form-checkout__installments',
            placeholder: 'Parcelas',
          },
          identificationType: {
            id: 'form-checkout__identificationType',
            placeholder: 'Tipo de documento',
          },
          identificationNumber: {
            id: 'form-checkout__identificationNumber',
            placeholder: 'Número do documento',
          },
          issuer: {
            id: 'form-checkout__issuer',
            placeholder: 'Banco emissor',
          },
        },
```

```
      callbacks: {
        onFormMounted: error => {
          if (error) {
            console.error('Form Mounted error:', error);
            if (onError) onError(error);
          } else {
            setLoading(false);
          }
        },
        onFormUnmounted: error => {
          if (error) {
            console.error('Form Unmounted error:', error);
          }
        },
        onIdentificationTypesReceived: (error, identificationTypes) => {
          if (error) {
            console.error('Identification Types error:', error);
          }
        },
        onPaymentMethodsReceived: (error, paymentMethods) => {
          if (error) {
            console.error('Payment Methods error:', error);
          }
        },
        onIssuersReceived: (error, issuers) => {
          if (error) {
            console.error('Issuers error:', error);
          }
        },
        onInstallmentsReceived: (error, installments) => {
          if (error) {
            console.error('Installments error:', error);
          }
        },
        onCardTokenReceived: (error, token) => {
          if (error) {
            console.error('Token error:', error);
            if (onError) onError(error);
          } else {
            if (onSubmit) {
              onSubmit({
                token,
                ...formData
              });
            }
          }
        },
        onSubmit: event => {
          event.preventDefault();

          const {
            cardholderName,
```

```
            identificationType,
            identificationNumber,
            installments,
            issuer
          } = cardForm.getCardFormData();

          setFormData({
            cardholderName,
            identificationType,
            identificationNumber,
            installments,
            issuer
          });

          cardForm.createCardToken();
        }
      }
    });

    setCardForm(cardForm);
  };

  document.body.appendChild(script);

  return () => {
    // Limpar script ao desmontar componente
    if (document.body.contains(script)) {
      document.body.removeChild(script);
    }
  };
 }, [onSubmit, onError]);

 return (
  <div className="mercadopago-card-form">
   {loading && <div className="loading">Carregando formulário de
pagamento...</div>}

   <form id="form-checkout" className={loading ? 'hidden' : ''}>
    <div className="form-group">
     <label htmlFor="form-checkout__cardholderName">Nome no cartão</label>
     <div id="form-checkout__cardholderName"></div>
    </div>

    <div className="form-group">
     <label htmlFor="form-checkout__cardholderEmail">E-mail</label>
     <div id="form-checkout__cardholderEmail"></div>
    </div>

    <div className="form-group">
     <label htmlFor="form-checkout__cardNumber">Número do cartão</label>
     <div id="form-checkout__cardNumber"></div>
    </div>
```

```
      <div className="form-row">
        <div className="form-group col-6">
          <label htmlFor="form-checkout__cardExpirationMonth">Mês</label>
          <div id="form-checkout__cardExpirationMonth"></div>
        </div>

        <div className="form-group col-6">
          <label htmlFor="form-checkout__cardExpirationYear">Ano</label>
          <div id="form-checkout__cardExpirationYear"></div>
        </div>
      </div>

      <div className="form-group">
        <label htmlFor="form-checkout__securityCode">Código de segurança</label>
        <div id="form-checkout__securityCode"></div>
      </div>

      <div className="form-group">
        <label htmlFor="form-checkout__installments">Parcelas</label>
        <div id="form-checkout__installments"></div>
      </div>

      <div className="form-group">
        <label htmlFor="form-checkout__identificationType">Tipo de documento</
label>
        <div id="form-checkout__identificationType"></div>
      </div>

      <div className="form-group">
        <label htmlFor="form-checkout__identificationNumber">Número do
documento</label>
        <div id="form-checkout__identificationNumber"></div>
      </div>

      <div className="form-group">
        <label htmlFor="form-checkout__issuer">Banco emissor</label>
        <div id="form-checkout__issuer"></div>
      </div>

      <button type="submit" className="btn btn--primary btn--block">Pagar</
button>
    </form>
  </div>
 );
};

export default MercadoPagoCardForm;
```

2. Endpoint para Processar Pagamento com Cartão

```
// /app/api/payment/mercadopago/process/route.js
import { NextResponse } from 'next/server';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth/auth';
import client from '@/lib/mercadopago';
import { Payment } from 'mercadopago';
import prisma from '@/lib/prisma';

export async function POST(request) {
  try {
    const session = await getServerSession(authOptions);

    // Verificar se usuário está autenticado
    if (!session) {
      return NextResponse.json(
        { error: 'Autenticação necessária' },
        { status: 401 }
      );
    }

    const data = await request.json();

    // Validar dados
    if (!data.token || !data.order_id) {
      return NextResponse.json(
        { error: 'Dados incompletos' },
        { status: 400 }
      );
    }

    // Buscar pedido
    const order = await prisma.order.findFirst({
      where: {
        id: data.order_id,
        userId: parseInt(session.user.id)
      },
      include: {
        user: {
          select: {
            email: true
          }
        }
      }
    });

    if (!order) {
      return NextResponse.json(
        { error: 'Pedido não encontrado' },
        { status: 404 }
```

```
    );
  }

  // Criar pagamento
  const payment = new Payment(client);

  const paymentData = {
    transaction_amount: parseFloat(order.total),
    token: data.token,
    description: `Pedido #${order.code}`,
    installments: parseInt(data.installments) || 1,
    payment_method_id: data.payment_method_id,
    issuer_id: data.issuer_id,
    payer: {
      email: data.payer_email || order.user.email,
      identification: {
        type: data.identification_type,
        number: data.identification_number
      }
    },
    external_reference: order.code,
    notification_url: `${process.env.NEXT_PUBLIC_API_URL}/api/webhooks/
mercadopago`,
    statement_descriptor: process.env.STORE_NAME || 'E-commerce'
  };

  const paymentResponse = await payment.create({ body: paymentData });

  // Mapear status do Mercado Pago para status interno
  let paymentStatus;
  let orderStatus;

  switch (paymentResponse.status) {
    case 'approved':
      paymentStatus = 'APPROVED';
      orderStatus = 'PAYMENT_APPROVED';
      break;
    case 'pending':
    case 'in_process':
      paymentStatus = 'PENDING';
      orderStatus = 'AWAITING_PAYMENT';
      break;
    case 'rejected':
      paymentStatus = 'REJECTED';
      orderStatus = 'AWAITING_PAYMENT';
      break;
    default:
      paymentStatus = 'PENDING';
      orderStatus = 'AWAITING_PAYMENT';
  }

  // Registrar pagamento
```

```
    await prisma.payment.create({
      data: {
        orderId: order.id,
        transactionId: paymentResponse.id.toString(),
        method: 'CREDIT_CARD',
        status: paymentStatus,
        amount: parseFloat(order.total),
        installments: parseInt(data.installments) || 1,
        gateway: 'mercadopago',
        paymentData: paymentResponse
      }
    });

    // Atualizar status do pedido
    const orderUpdate = {
      status: orderStatus
    };

    if (orderStatus === 'PAYMENT_APPROVED') {
      orderUpdate.paidAt = new Date();
    }

    await prisma.order.update({
      where: { id: order.id },
      data: orderUpdate
    });

    return NextResponse.json({
      success: true,
      status: paymentResponse.status,
      payment_id: paymentResponse.id,
      order_status: orderStatus
    });
  } catch (error) {
    console.error('Error processing payment:', error);
    return NextResponse.json(
      { error: 'Erro ao processar pagamento', details: error.message },
      { status: 500 }
    );
  }
}
```

Implementação de Pagamento com Pix

1. Endpoint para Gerar Pagamento Pix

```
// /app/api/payment/mercadopago/pix/route.js
import { NextResponse } from 'next/server';
```

```
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth/auth';
import client from '@/lib/mercadopago';
import { Payment } from 'mercadopago';
import prisma from '@/lib/prisma';

export async function POST(request) {
  try {
    const session = await getServerSession(authOptions);

    // Verificar se usuário está autenticado
    if (!session) {
      return NextResponse.json(
        { error: 'Autenticação necessária' },
        { status: 401 }
      );
    }

    const data = await request.json();

    // Validar dados
    if (!data.order_id) {
      return NextResponse.json(
        { error: 'ID do pedido é obrigatório' },
        { status: 400 }
      );
    }

    // Buscar pedido
    const order = await prisma.order.findFirst({
      where: {
        id: data.order_id,
        userId: parseInt(session.user.id)
      },
      include: {
        user: {
          select: {
            name: true,
            email: true,
            cpf: true
          }
        }
      }
    });

    if (!order) {
      return NextResponse.json(
        { error: 'Pedido não encontrado' },
        { status: 404 }
      );
    }
```

```
    // Criar pagamento Pix
    const payment = new Payment(client);

    const paymentData = {
      transaction_amount: parseFloat(order.total),
      description: `Pedido #${order.code}`,
      payment_method_id: 'pix',
      payer: {
        email: order.user.email,
        first_name: order.user.name.split(' ')[0],
        last_name: order.user.name.split(' ').slice(1).join(' '),
        identification: {
          type: 'CPF',
          number: order.user.cpf?.replace(/\D/g, '') || data.cpf?.replace(/\D/g, '')
        }
      },
      external_reference: order.code,
      notification_url: `${process.env.NEXT_PUBLIC_API_URL}/api/webhooks/
mercadopago`
    };

    const paymentResponse = await payment.create({ body: paymentData });

    // Registrar pagamento
    await prisma.payment.create({
      data: {
        orderId: order.id,
        transactionId: paymentResponse.id.toString(),
        method: 'PIX',
        status: 'PENDING',
        amount: parseFloat(order.total),
        installments: 1,
        gateway: 'mercadopago',
        paymentData: paymentResponse
      }
    });

    // Extrair dados do Pix
    const pixData = paymentResponse.point_of_interaction.transaction_data;

    return NextResponse.json({
      success: true,
      payment_id: paymentResponse.id,
      qr_code: pixData.qr_code,
      qr_code_base64: pixData.qr_code_base64,
      ticket_url: pixData.ticket_url
    });
  } catch (error) {
    console.error('Error generating Pix payment:', error);
    return NextResponse.json(
      { error: 'Erro ao gerar pagamento Pix', details: error.message },
      { status: 500 }
```

```
    );
  }
}
```

2. Componente de Pagamento Pix

```
// /js/components/checkout/MercadoPagoPix.js
import { useState } from 'react';

const MercadoPagoPix = ({ pixData, orderId, onSuccess }) => {
  const [copied, setCopied] = useState(false);

  const copyPixCode = () => {
    navigator.clipboard.writeText(pixData.qr_code).then(() => {
      setCopied(true);
      setTimeout(() => setCopied(false), 3000);
    });
  };

  const checkPaymentStatus = async () => {
    try {
      const response = await fetch(`/api/payment/status?order_id=${orderId}`);
      const data = await response.json();

      if (data.status === 'PAYMENT_APPROVED') {
        if (onSuccess) onSuccess();
      } else {
        alert('Pagamento ainda não confirmado. Por favor, tente novamente em
alguns instantes.');
      }
    } catch (error) {
      console.error('Error checking payment status:', error);
      alert('Erro ao verificar status do pagamento.');
    }
  };

  return (
    <div className="pix-payment">
      <div className="pix-payment__header">
        <h3>Pagamento via Pix</h3>
        <p>Escaneie o QR Code abaixo ou copie o código para pagar:</p>
      </div>

      <div className="pix-payment__qrcode">
        <img
          src={`${pixData.qr_code_base64}`}
          alt="QR Code Pix"
        />
      </div>
```

```
      <div className="pix-payment__code">
        <button
          className="btn btn--secondary"
          onClick={copyPixCode}
        >
          {copied ? 'Código copiado!' : 'Copiar código Pix'}
        </button>
      </div>

      <div className="pix-payment__instructions">
        <ol>
          <li>Abra o aplicativo do seu banco</li>
          <li>Escolha a opção de pagamento via Pix</li>
          <li>Escaneie o QR Code ou cole o código copiado</li>
          <li>Confirme as informações e finalize o pagamento</li>
        </ol>
      </div>

      <div className="pix-payment__actions">
        <button
          className="btn btn--primary"
          onClick={checkPaymentStatus}
        >
          Já realizei o pagamento
        </button>
      </div>
    </div>
  );
};

export default MercadoPagoPix;
```

Implementação de Pagamento com Boleto

1. Endpoint para Gerar Boleto

```
// /app/api/payment/mercadopago/boleto/route.js
import { NextResponse } from 'next/server';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth/auth';
import client from '@/lib/mercadopago';
import { Payment } from 'mercadopago';
import prisma from '@/lib/prisma';

export async function POST(request) {
  try {
    const session = await getServerSession(authOptions);
```

```
    // Verificar se usuário está autenticado
    if (!session) {
      return NextResponse.json(
        { error: 'Autenticação necessária' },
        { status: 401 }
      );
    }

    const data = await request.json();

    // Validar dados
    if (!data.order_id) {
      return NextResponse.json(
        { error: 'ID do pedido é obrigatório' },
        { status: 400 }
      );
    }

    // Buscar pedido
    const order = await prisma.order.findFirst({
      where: {
        id: data.order_id,
        userId: parseInt(session.user.id)
      },
      include: {
        shippingAddress: true,
        user: {
          select: {
            name: true,
            email: true,
            cpf: true
          }
        }
      }
    });

    if (!order) {
      return NextResponse.json(
        { error: 'Pedido não encontrado' },
        { status: 404 }
      );
    }

    // Criar pagamento com boleto
    const payment = new Payment(client);

    const paymentData = {
      transaction_amount: parseFloat(order.total),
      description: `Pedido #${order.code}`,
      payment_method_id: 'bolbradesco',
      payer: {
        email: order.user.email,
```

```
        first_name: order.user.name.split(' ')[0],
        last_name: order.user.name.split(' ').slice(1).join(' '),
        identification: {
          type: 'CPF',
          number: order.user.cpf?.replace(/\D/g, '') || data.cpf?.replace(/\D/g, '')
        },
        address: {
          zip_code: order.shippingAddress?.zipCode?.replace(/\D/g, '') || '',
          street_name: order.shippingAddress?.street || '',
          street_number: order.shippingAddress?.number || '',
          neighborhood: order.shippingAddress?.neighborhood || '',
          city: order.shippingAddress?.city || '',
          federal_unit: order.shippingAddress?.state || ''
        }
      },
      external_reference: order.code,
      notification_url: `${process.env.NEXT_PUBLIC_API_URL}/api/webhooks/
mercadopago`
    };

    const paymentResponse = await payment.create({ body: paymentData });

    // Registrar pagamento
    await prisma.payment.create({
      data: {
        orderId: order.id,
        transactionId: paymentResponse.id.toString(),
        method: 'BOLETO',
        status: 'PENDING',
        amount: parseFloat(order.total),
        installments: 1,
        gateway: 'mercadopago',
        paymentData: paymentResponse
      }
    });

    return NextResponse.json({
      success: true,
      payment_id: paymentResponse.id,
      boleto_url: paymentResponse.transaction_details.external_resource_url,
      barcode: paymentResponse.barcode?.content
    });
  } catch (error) {
    console.error('Error generating boleto payment:', error);
    return NextResponse.json(
      { error: 'Erro ao gerar boleto', details: error.message },
      { status: 500 }
    );
  }
}
```

2. Componente de Pagamento com Boleto

```
// /js/components/checkout/MercadoPagoBoleto.js
const MercadoPagoBoleto = ({ boletoData }) => {
 return (
  <div className="boleto-payment">
   <div className="boleto-payment__header">
    <h3>Pagamento via Boleto</h3>
    <p>Seu boleto foi gerado com sucesso!</p>
   </div>

   <div className="boleto-payment__info">
    <p>O boleto pode levar até 3 dias úteis para ser compensado após o
pagamento.</p>
    <p>Seu pedido será processado após a confirmação do pagamento.</p>
   </div>

   {boletoData.barcode && (
    <div className="boleto-payment__barcode">
     <p>Código de barras:</p>
     <code>{boletoData.barcode}</code>
    </div>
   )}

   <div className="boleto-payment__actions">
    <a
     href={boletoData.boleto_url}
     target="_blank"
     rel="noopener noreferrer"
     className="btn btn--primary"
    >
     Visualizar Boleto
    </a>

    <a
     href={boletoData.boleto_url}
     download="boleto.pdf"
     className="btn btn--secondary"
    >
     Baixar Boleto
    </a>
   </div>
  </div>
 );
};

export default MercadoPagoBoleto;
```

Processamento de Webhooks

1. Manipulador de Webhooks

```
// /lib/mercadopago/webhook.js
import prisma from '@/lib/prisma';
import client from './index';
import { Payment } from 'mercadopago';

export async function handleWebhook(data) {
  try {
    // Verificar tipo de notificação
    if (data.type !== 'payment') {
      return { success: true, message: 'Notificação ignorada: não é um pagamento' };
    }

    // Buscar detalhes do pagamento
    const payment = new Payment(client);
    const paymentInfo = await payment.get({ id: data.data.id });

    // Buscar pedido pelo código de referência externa
    const order = await prisma.order.findFirst({
      where: { code: paymentInfo.external_reference },
      include: { payments: true }
    });

    if (!order) {
      throw new Error(`Pedido não encontrado: ${paymentInfo.external_reference}`);
    }

    // Mapear status do Mercado Pago para status interno
    let paymentStatus;
    let orderStatus;

    switch (paymentInfo.status) {
      case 'approved':
        paymentStatus = 'APPROVED';
        orderStatus = 'PAYMENT_APPROVED';
        break;
      case 'pending':
      case 'in_process':
        paymentStatus = 'PENDING';
        orderStatus = 'AWAITING_PAYMENT';
        break;
      case 'rejected':
        paymentStatus = 'REJECTED';
        orderStatus = 'AWAITING_PAYMENT';
        break;
      case 'refunded':
      case 'cancelled':
```

```
      paymentStatus = 'REFUNDED';
      orderStatus = 'CANCELED';
      break;
    default:
      paymentStatus = 'PENDING';
      orderStatus = 'AWAITING_PAYMENT';
  }

  // Buscar pagamento existente
  const existingPayment = await prisma.payment.findFirst({
    where: { transactionId: paymentInfo.id.toString() }
  });

  if (existingPayment) {
    // Atualizar pagamento existente
    await prisma.payment.update({
      where: { id: existingPayment.id },
      data: {
        status: paymentStatus,
        paymentData: paymentInfo
      }
    });
  } else {
    // Criar novo registro de pagamento
    await prisma.payment.create({
      data: {
        orderId: order.id,
        transactionId: paymentInfo.id.toString(),
        method: mapPaymentMethod(paymentInfo.payment_method_id),
        status: paymentStatus,
        amount: parseFloat(paymentInfo.transaction_amount),
        installments: paymentInfo.installments || 1,
        gateway: 'mercadopago',
        paymentData: paymentInfo
      }
    });
  }

  // Atualizar pedido
  const orderUpdate = {
    status: orderStatus
  };

  if (orderStatus === 'PAYMENT_APPROVED' && !order.paidAt) {
    orderUpdate.paidAt = new Date();
  } else if (orderStatus === 'CANCELED' && !order.canceledAt) {
    orderUpdate.canceledAt = new Date();
  }

  await prisma.order.update({
    where: { id: order.id },
    data: orderUpdate
```

```
    });

    // Registrar log
    await prisma.log.create({
      data: {
        type: 'PAYMENT',
        action: 'WEBHOOK_RECEIVED',
        description: `Webhook do Mercado Pago processado para o pedido ${order.code}`,
        data: {
          payment_id: paymentInfo.id,
          order_id: order.id,
          order_code: order.code,
          status: paymentInfo.status,
          payment_method: paymentInfo.payment_method_id
        }
      }
    });

    return {
      success: true,
      message: 'Webhook processado com sucesso',
      order: order.code,
      status: orderStatus
    };
  } catch (error) {
    console.error('Error processing Mercado Pago webhook:', error);

    // Registrar erro
    await prisma.log.create({
      data: {
        type: 'ERROR',
        action: 'WEBHOOK_ERROR',
        description: `Erro ao processar webhook do Mercado Pago: ${error.message}`,
        data: { error: error.message, data }
      }
    });

    throw error;
  }
}

// Função auxiliar para mapear método de pagamento
function mapPaymentMethod(mpMethod) {
  const methodMap = {
    'credit_card': 'CREDIT_CARD',
    'debit_card': 'CREDIT_CARD',
    'bolbradesco': 'BOLETO',
    'pix': 'PIX',
    'account_money': 'MERCADOPAGO'
  };
```

```
  return methodMap[mpMethod] || 'MERCADOPAGO';
}
```

2. Endpoint de Webhook

```
// /app/api/webhooks/mercadopago/route.js
import { NextResponse } from 'next/server';
import { handleWebhook } from '@/lib/mercadopago/webhook';

export async function POST(request) {
  try {
    const data = await request.json();

    // Registrar recebimento do webhook
    console.log('Mercado Pago webhook received:', JSON.stringify(data));

    // Processar webhook
    const result = await handleWebhook(data);

    return NextResponse.json(result);
  } catch (error) {
    console.error('Error in Mercado Pago webhook:', error);
    return NextResponse.json(
      { error: 'Erro ao processar webhook', details: error.message },
      { status: 500 }
    );
  }
}
```

Consulta de Status de Pagamento

```
// /app/api/payment/status/route.js
import { NextResponse } from 'next/server';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth/auth';
import client from '@/lib/mercadopago';
import { Payment } from 'mercadopago';
import prisma from '@/lib/prisma';

export async function GET(request) {
  try {
    const session = await getServerSession(authOptions);

    // Verificar se usuário está autenticado
    if (!session) {
      return NextResponse.json(
        { error: 'Autenticação necessária' },
```

```
      { status: 401 }
    );
  }

  const { searchParams } = new URL(request.url);
  const orderId = searchParams.get('order_id');
  const orderCode = searchParams.get('order_code');

  if (!orderId && !orderCode) {
    return NextResponse.json(
      { error: 'ID ou código do pedido é obrigatório' },
      { status: 400 }
    );
  }

  // Buscar pedido
  const where = {};

  if (orderId) {
    where.id = parseInt(orderId);
  } else {
    where.code = orderCode;
  }

  where.userId = parseInt(session.user.id);

  const order = await prisma.order.findFirst({
    where,
    include: {
      payments: {
        orderBy: { createdAt: 'desc' },
        take: 1
      }
    }
  });

  if (!order) {
    return NextResponse.json(
      { error: 'Pedido não encontrado' },
      { status: 404 }
    );
  }

  // Se não houver pagamentos, retornar status do pedido
  if (!order.payments.length) {
    return NextResponse.json({
      order_id: order.id,
      order_code: order.code,
      status: order.status,
      payment_status: null
    });
  }
```

```
const payment = order.payments[0];

// Se o pagamento já estiver aprovado, não precisa consultar a API
if (payment.status === 'APPROVED') {
  return NextResponse.json({
    order_id: order.id,
    order_code: order.code,
    status: order.status,
    payment_id: payment.transactionId,
    payment_status: payment.status,
    payment_method: payment.method
  });
}

// Consultar status atualizado na API do Mercado Pago
if (payment.transactionId) {
  try {
    const mpPayment = new Payment(client);
    const paymentInfo = await mpPayment.get({ id: payment.transactionId });

    // Mapear status do Mercado Pago para status interno
    let paymentStatus;
    let orderStatus;

    switch (paymentInfo.status) {
      case 'approved':
        paymentStatus = 'APPROVED';
        orderStatus = 'PAYMENT_APPROVED';
        break;
      case 'pending':
      case 'in_process':
        paymentStatus = 'PENDING';
        orderStatus = 'AWAITING_PAYMENT';
        break;
      case 'rejected':
        paymentStatus = 'REJECTED';
        orderStatus = 'AWAITING_PAYMENT';
        break;
      case 'refunded':
      case 'cancelled':
        paymentStatus = 'REFUNDED';
        orderStatus = 'CANCELED';
        break;
      default:
        paymentStatus = 'PENDING';
        orderStatus = 'AWAITING_PAYMENT';
    }

    // Atualizar status se necessário
    if (payment.status !== paymentStatus) {
      await prisma.payment.update({
```

```
          where: { id: payment.id },
          data: {
            status: paymentStatus,
            paymentData: paymentInfo
          }
        });

        const orderUpdate = {
          status: orderStatus
        };

        if (orderStatus === 'PAYMENT_APPROVED' && !order.paidAt) {
          orderUpdate.paidAt = new Date();
        } else if (orderStatus === 'CANCELED' && !order.canceledAt) {
          orderUpdate.canceledAt = new Date();
        }

        await prisma.order.update({
          where: { id: order.id },
          data: orderUpdate
        });
      }

      return NextResponse.json({
        order_id: order.id,
        order_code: order.code,
        status: orderStatus,
        payment_id: payment.transactionId,
        payment_status: paymentStatus,
        payment_method: payment.method
      });
    } catch (error) {
      console.error('Error fetching payment from Mercado Pago:', error);
      // Em caso de erro, retornar o status atual
      return NextResponse.json({
        order_id: order.id,
        order_code: order.code,
        status: order.status,
        payment_id: payment.transactionId,
        payment_status: payment.status,
        payment_method: payment.method,
        error: 'Erro ao consultar status atualizado'
      });
    }
  }

  // Retornar status atual
  return NextResponse.json({
    order_id: order.id,
    order_code: order.code,
    status: order.status,
    payment_id: payment.transactionId,
```

```
      payment_status: payment.status,
      payment_method: payment.method
    });
  } catch (error) {
    console.error('Error checking payment status:', error);
    return NextResponse.json(
      { error: 'Erro ao verificar status do pagamento', details: error.message },
      { status: 500 }
    );
  }
}
```

Estornos e Cancelamentos

```
// /app/api/payment/mercadopago/refund/route.js
import { NextResponse } from 'next/server';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth/auth';
import client from '@/lib/mercadopago';
import { Payment } from 'mercadopago';
import prisma from '@/lib/prisma';

export async function POST(request) {
  try {
    const session = await getServerSession(authOptions);

    // Verificar se usuário é admin
    if (!session || session.user.type !== 'ADMIN') {
      return NextResponse.json(
        { error: 'Não autorizado' },
        { status: 401 }
      );
    }

    const data = await request.json();

    // Validar dados
    if (!data.payment_id) {
      return NextResponse.json(
        { error: 'ID do pagamento é obrigatório' },
        { status: 400 }
      );
    }

    // Buscar pagamento
    const payment = await prisma.payment.findFirst({
      where: { transactionId: data.payment_id.toString() },
      include: { order: true }
    });
```

```
    if (!payment) {
      return NextResponse.json(
        { error: 'Pagamento não encontrado' },
        { status: 404 }
      );
    }

    // Verificar se pagamento pode ser estornado
    if (payment.status !== 'APPROVED') {
      return NextResponse.json(
        { error: 'Apenas pagamentos aprovados podem ser estornados' },
        { status: 400 }
      );
    }

    // Realizar estorno no Mercado Pago
    const mpPayment = new Payment(client);
    const refundResponse = await mpPayment.refund({ id: data.payment_id });

    // Atualizar status do pagamento
    await prisma.payment.update({
      where: { id: payment.id },
      data: {
        status: 'REFUNDED',
        paymentData: {
          ...payment.paymentData,
          refund: refundResponse
        }
      }
    });

    // Atualizar status do pedido
    await prisma.order.update({
      where: { id: payment.orderId },
      data: {
        status: 'CANCELED',
        canceledAt: new Date()
      }
    });

    // Registrar log
    await prisma.log.create({
      data: {
        userId: parseInt(session.user.id),
        type: 'PAYMENT',
        action: 'REFUND',
        description: `Estorno realizado para o pagamento ${data.payment_id} do
pedido ${payment.order.code}`,
        data: {
          payment_id: data.payment_id,
          order_id: payment.orderId,
```

```
          order_code: payment.order.code,
          amount: payment.amount
        }
      }
    });

    return NextResponse.json({
      success: true,
      refund_id: refundResponse.id,
      status: 'REFUNDED'
    });
  } catch (error) {
    console.error('Error processing refund:', error);
    return NextResponse.json(
      { error: 'Erro ao processar estorno', details: error.message },
      { status: 500 }
    );
  }
}
```

Considerações de Segurança

1. Proteção de Credenciais

- Armazenar todas as chaves de API como variáveis de ambiente
- Nunca expor a chave de acesso (`ACCESS_TOKEN`) no frontend
- Utilizar apenas a chave pública (`PUBLIC_KEY`) no frontend

2. Validação de Webhooks

- Verificar a autenticidade das notificações recebidas
- Implementar validação de IP para garantir que as notificações venham do Mercado Pago
- Confirmar os dados recebidos consultando a API do Mercado Pago

3. Proteção contra Fraudes

- Implementar validação de dados do comprador
- Utilizar ferramentas de análise de risco do Mercado Pago
- Monitorar transações suspeitas

4. Conformidade com PCI DSS

- Nunca armazenar dados completos de cartão de crédito
- Utilizar tokenização para processamento de pagamentos

- Seguir as diretrizes de segurança do Mercado Pago

Testes e Homologação

1. Ambiente de Testes

- Utilizar credenciais de teste para desenvolvimento
- Criar usuários de teste para simular compradores e vendedores
- Utilizar cartões de teste fornecidos pelo Mercado Pago

2. Casos de Teste

- Pagamento aprovado com cartão de crédito
- Pagamento rejeitado por fundos insuficientes
- Pagamento com Pix
- Pagamento com boleto
- Recebimento e processamento de webhooks
- Estorno de pagamento
- Cancelamento de pedido

3. Homologação

- Testar fluxo completo de pagamento em ambiente de teste
- Verificar integração com o banco de dados
- Validar atualização de status de pedidos
- Confirmar recebimento e processamento correto de notificações

Documentação para Usuários

1. Métodos de Pagamento Disponíveis

- **Cartão de Crédito**: Pagamento imediato com possibilidade de parcelamento
- **Pix**: Pagamento instantâneo com QR Code
- **Boleto Bancário**: Pagamento em até 3 dias úteis após a emissão

2. Instruções para Pagamento

- **Cartão de Crédito**: Preencher dados do cartão, escolher número de parcelas e confirmar
- **Pix**: Escanear QR Code ou copiar código para pagar no aplicativo do banco

- **Boleto**: Imprimir boleto ou copiar código de barras para pagar em banco, lotérica ou aplicativo

3. Prazos de Processamento

- **Cartão de Crédito**: Aprovação imediata (sujeito a análise antifraude)
- **Pix**: Confirmação em até 1 minuto após o pagamento
- **Boleto**: Confirmação em até 3 dias úteis após o pagamento

Conclusão

A integração com o Mercado Pago foi planejada para oferecer uma experiência de pagamento completa e segura, com múltiplas opções de pagamento adaptadas ao mercado brasileiro. A implementação segue as melhores práticas recomendadas pelo Mercado Pago, garantindo segurança, conformidade e uma experiência de usuário fluida.

O sistema foi projetado para lidar com todo o ciclo de vida do pagamento, desde a criação até o processamento de notificações e possíveis estornos, com foco na segurança e na experiência do usuário.

Esta integração permitirá que o e-commerce processe pagamentos de forma eficiente e segura, oferecendo diversas opções aos clientes e garantindo a segurança das transações.