# Integração dos Módulos: Produtos e Clientes

**Sistema de Gestão de Lojas - Arquitetura Modular**

**Autor**: Manus AI

**Data**: 31 de Agosto de 2024

**Versão**: 1.0

## Resumo Executivo

Este documento detalha as estratégias, implementações e melhores práticas para integrar o Módulo 1.2 (Cadastro de Produtos) com o Módulo de Clientes (anteriormente desenvolvido como Módulo 1.3). A integração visa criar uma experiência unificada e coesa no Sistema de Gestão de Lojas, estabelecendo as bases para funcionalidades avançadas como vendas, relatórios combinados e recomendações personalizadas.

A abordagem proposta mantém a independência dos módulos enquanto estabelece pontos de comunicação eficientes através de APIs RESTful, banco de dados compartilhado e interfaces de usuário integradas. Esta estratégia permite que cada módulo continue evoluindo independentemente, ao mesmo tempo que oferece funcionalidades integradas quando necessário.

## Análise dos Pontos de Integração

### Contexto Arquitetural

O Sistema de Gestão de Lojas foi projetado seguindo uma arquitetura modular que permite desenvolvimento, teste e implantação independentes de cada componente. Esta abordagem oferece vantagens significativas em termos de manutenibilidade, escalabilidade e flexibilidade de desenvolvimento. No entanto, para oferecer uma experiência completa aos usuários, é essencial estabelecer pontos de integração bem definidos entre os módulos.

O Módulo de Produtos gerencia todo o catálogo de itens disponíveis para venda, incluindo informações como preços, categorias, marcas, status de disponibilidade e características técnicas. Por outro lado, o Módulo de Clientes mantém informações detalhadas sobre os consumidores, incluindo dados pessoais, histórico de interações, preferências e segmentação. A integração entre esses módulos é fundamental para operações comerciais efetivas.

## Pontos de Interação Identificados

A análise detalhada dos dois módulos revela cinco pontos principais de integração que são essenciais para o funcionamento completo do sistema de gestão de lojas.

O primeiro e mais crítico ponto de integração é o processo de vendas e pedidos. Quando um cliente realiza uma compra, é necessário associar produtos específicos a esse cliente, criando um registro que conecta ambos os módulos. Esta operação requer validação da disponibilidade do produto, verificação dos dados do cliente, cálculo de preços e geração de documentos fiscais. A integração neste ponto deve ser robusta e confiável, pois representa o core business da aplicação.

O segundo ponto de integração envolve o histórico de compras e relacionamento cliente-produto. Cada transação cria um vínculo histórico entre cliente e produto que pode ser utilizado para análises futuras, recomendações personalizadas e estratégias de marketing direcionado. Este histórico deve ser facilmente acessível e consultável por ambos os módulos, permitindo que o módulo de clientes exiba produtos comprados anteriormente e que o módulo de produtos identifique quais clientes compraram determinados itens.

O terceiro ponto refere-se a promoções e descontos personalizados. A capacidade de aplicar preços diferenciados ou promoções específicas baseadas no perfil do cliente ou histórico de compras representa uma funcionalidade avançada que requer integração profunda entre os módulos. Isso inclui descontos por volume, preços especiais para clientes VIP, promoções baseadas em categorias de produtos preferidas pelo cliente, entre outras estratégias comerciais.

O quarto ponto de integração envolve relatórios e análises combinadas. Gestores de loja frequentemente necessitam de relatórios que cruzem informações de produtos e clientes, como produtos mais vendidos por segmento de cliente, análise de lucratividade por cliente, identificação de produtos com baixa rotatividade em determinados grupos de clientes, e análises de sazonalidade baseadas no comportamento de compra.

O quinto e último ponto identificado é o sistema de recomendações e sugestões. Utilizando dados históricos de compras e preferências dos clientes, o sistema pode sugerir produtos relevantes, alertar sobre novos itens em categorias de interesse, ou recomendar produtos complementares. Esta funcionalidade requer acesso em tempo real a dados de ambos os módulos e algoritmos de análise que processem essas informações de forma eficiente.

## Requisitos de Integração

Para implementar efetivamente esses pontos de integração, é necessário estabelecer requisitos técnicos e funcionais claros. Do ponto de vista técnico, a integração deve manter a independência dos módulos, permitir comunicação eficiente entre eles, garantir

consistência de dados, e oferecer performance adequada mesmo com grandes volumes de informações.

A independência dos módulos deve ser preservada através de interfaces bem definidas e acoplamento baixo. Cada módulo deve continuar funcionando mesmo se o outro estiver temporariamente indisponível, degradando graciosamente as funcionalidades que dependem da integração. Isso é essencial para manter a robustez do sistema e facilitar a manutenção.

A comunicação entre módulos deve ser eficiente e confiável, utilizando protocolos padronizados como HTTP/REST para chamadas síncronas e sistemas de mensageria para comunicação assíncrona quando apropriado. As APIs devem ser versionadas para permitir evolução independente dos módulos sem quebrar a compatibilidade.

A consistência de dados é crucial, especialmente em operações que envolvem ambos os módulos simultaneamente. Mecanismos de transação distribuída ou padrões como Saga podem ser necessários para garantir que operações complexas sejam executadas de forma atômica e consistente.

A performance deve ser otimizada através de estratégias como cache distribuído, indexação adequada de dados compartilhados, e minimização de chamadas entre módulos através de agregação de dados quando possível.

# Estratégias de Integração a Nível de API e Banco de Dados

## Arquitetura de Comunicação entre Módulos

A integração efetiva entre os módulos de Produtos e Clientes requer uma arquitetura de comunicação bem estruturada que permita troca de informações de forma eficiente e confiável. A abordagem recomendada combina comunicação síncrona através de APIs RESTful para operações em tempo real com comunicação assíncrona através de eventos para atualizações que não requerem resposta imediata.

Para comunicação síncrona, cada módulo expõe endpoints específicos que podem ser consumidos pelo outro módulo. O Módulo de Produtos disponibiliza endpoints para consulta de informações de produtos, verificação de disponibilidade, e atualização de status. O Módulo de Clientes oferece endpoints para validação de dados de clientes, consulta de histórico, e verificação de elegibilidade para promoções. Essa comunicação direta é ideal para operações que requerem resposta imediata, como validação durante o processo de venda.

A comunicação assíncrona é implementada através de um sistema de eventos que permite que os módulos notifiquem uns aos outros sobre mudanças importantes sem bloquear operações. Por exemplo, quando um produto é descontinuado, o Módulo de Produtos pode

emitir um evento que será processado pelo Módulo de Clientes para atualizar recomendações ou alertar clientes que compraram esse produto anteriormente.

## Estratégia de Banco de Dados Compartilhado

A estratégia de banco de dados para integração dos módulos segue o padrão de "Database per Service" com elementos compartilhados estratégicos. Cada módulo mantém seu próprio schema de banco de dados, garantindo independência e permitindo otimizações específicas para cada domínio. No entanto, algumas tabelas e views são compartilhadas para facilitar operações que envolvem ambos os módulos.

O Módulo de Produtos utiliza o schema `produtos` com tabelas para produtos, categorias, marcas e relacionamentos. O Módulo de Clientes utiliza o schema `clientes` com tabelas para clientes, endereços, contatos e segmentação. Para integração, é criado um schema adicional `integracao` que contém tabelas e views que facilitam operações combinadas.

A tabela principal de integração é `vendas_itens` que registra cada item vendido associando cliente_id e produto_id. Esta tabela atua como uma ponte entre os dois módulos, permitindo consultas eficientes sobre histórico de compras, análises de comportamento, e geração de relatórios combinados. A estrutura desta tabela inclui campos como data da venda, quantidade, preço unitário, desconto aplicado, e metadados adicionais que podem ser úteis para análises futuras.

Views materializadas são criadas para otimizar consultas frequentes que envolvem dados de ambos os módulos. Por exemplo, uma view `cliente_produto_historico` agrega informações de compras por cliente e produto, facilitando consultas sobre preferências e padrões de compra. Outra view `produto_cliente_stats` fornece estatísticas agregadas sobre vendas de produtos por segmento de cliente.

## Implementação de APIs de Integração

A implementação das APIs de integração segue princípios RESTful com extensões específicas para suportar operações complexas que envolvem múltiplos módulos. Cada módulo expõe endpoints dedicados para integração, separados dos endpoints de uso geral para permitir otimizações específicas e controle de acesso diferenciado.

O Módulo de Produtos implementa endpoints de integração como `/api/v1/integration/produtos/bulk-info` que permite consulta eficiente de informações de múltiplos produtos simultaneamente, otimizando operações de venda que envolvem vários itens. O endpoint `/api/v1/integration/produtos/availability` verifica disponibilidade em tempo real, considerando estoque e status do produto. O endpoint `/api/v1/integration/produtos/pricing` calcula preços considerando promoções ativas e regras de desconto.

O Módulo de Clientes oferece endpoints como `/api/v1/integration/clientes/validate` para validação rápida de dados de cliente durante o processo de venda, `/api/v1/integration/clientes/eligibility` para verificar elegibilidade para promoções específicas, e `/api/v1/integration/clientes/history` para consultar histórico de compras de forma otimizada.

Um aspecto crucial da implementação é o tratamento de falhas e degradação graciosa. Quando um módulo está indisponível, o outro deve continuar funcionando com funcionalidades reduzidas. Por exemplo, se o Módulo de Clientes estiver indisponível durante uma venda, o sistema pode permitir vendas para "cliente avulso" sem perder a funcionalidade principal.

## Gerenciamento de Transações Distribuídas

Operações que envolvem ambos os módulos, como registrar uma venda, requerem coordenação cuidadosa para manter consistência de dados. A abordagem recomendada utiliza o padrão Saga para gerenciar transações distribuídas, evitando a complexidade de transações ACID distribuídas enquanto mantém consistência eventual.

Uma venda típica envolve várias etapas: validação do cliente, verificação de disponibilidade de produtos, cálculo de preços, aplicação de descontos, atualização de estoque, e registro da transação. Cada etapa é implementada como uma operação compensável que pode ser revertida se etapas subsequentes falharem.

O processo inicia com a validação do cliente no Módulo de Clientes. Se bem-sucedida, prossegue para verificação de produtos no Módulo de Produtos. Cada etapa registra seu estado em uma tabela de controle de saga, permitindo recuperação em caso de falha. Se alguma etapa falha, operações compensatórias são executadas na ordem reversa para desfazer mudanças já realizadas.

## Cache Distribuído e Sincronização

Para otimizar performance e reduzir latência em operações de integração, é implementado um sistema de cache distribuído usando Redis. Este cache armazena informações frequentemente acessadas de ambos os módulos, como dados básicos de produtos, informações de clientes ativos, e resultados de consultas complexas.

A estratégia de cache utiliza padrões como Cache-Aside para dados relativamente estáticos e Write-Through para dados que mudam frequentemente. Informações de produtos como nome, preço base, e categoria são cacheadas com TTL longo, enquanto dados de disponibilidade e preços promocionais têm TTL mais curto.

A sincronização do cache é gerenciada através de eventos. Quando um produto é atualizado no Módulo de Produtos, um evento é emitido para invalidar entradas relacionadas no cache. O mesmo ocorre para atualizações de clientes. Esta abordagem

garante que o cache permaneça consistente sem impactar significativamente a performance.

### Monitoramento e Observabilidade

A integração entre módulos introduz complexidade adicional que requer monitoramento especializado. Métricas específicas são coletadas para operações de integração, incluindo latência de chamadas entre módulos, taxa de sucesso de operações distribuídas, e utilização do cache compartilhado.

Logs estruturados são implementados com correlation IDs que permitem rastrear operações que atravessam múltiplos módulos. Quando uma venda é processada, o mesmo correlation ID é usado em todos os logs relacionados, facilitando debugging e análise de performance.

Alertas são configurados para detectar problemas de integração, como alta latência em chamadas entre módulos, falhas frequentes em operações distribuídas, ou inconsistências de dados detectadas através de verificações periódicas.

### Versionamento e Evolução de APIs

As APIs de integração seguem estratégias rigorosas de versionamento para permitir evolução independente dos módulos. Cada endpoint de integração inclui versão no path (ex: `/api/v1/integration/`) e suporta múltiplas versões simultaneamente durante períodos de transição.

Mudanças backward-compatible são implementadas na mesma versão, enquanto mudanças breaking requerem nova versão. Documentação detalhada é mantida para cada versão, incluindo exemplos de uso e guias de migração.

Um processo de deprecação gradual é seguido para versões antigas, com avisos antecipados e períodos de suporte estendido para facilitar migração. Métricas de uso são coletadas para cada versão de API, ajudando a determinar quando versões antigas podem ser descontinuadas com segurança.

## Estratégias de Integração a Nível de Frontend

### Arquitetura de Interface Unificada

A integração dos módulos de Produtos e Clientes no frontend requer uma abordagem arquitetural que mantenha a modularidade do código enquanto oferece uma experiência de usuário coesa e intuitiva. A estratégia recomendada utiliza uma arquitetura de micro-frontends com um shell application que orquestra os diferentes módulos, permitindo desenvolvimento independente enquanto garante consistência visual e funcional.

O shell application atua como o ponto de entrada principal do sistema, gerenciando navegação global, autenticação, estado compartilhado, e comunicação entre módulos. Este componente é responsável por carregar dinamicamente os módulos conforme necessário, implementar lazy loading para otimizar performance, e manter um design system consistente em toda a aplicação.

Cada módulo mantém sua própria estrutura de componentes React, mas segue padrões estabelecidos pelo design system compartilhado. O Módulo de Produtos continua gerenciando suas interfaces de cadastro, listagem e edição de produtos, enquanto o Módulo de Clientes mantém suas funcionalidades específicas. A integração ocorre através de componentes compartilhados e interfaces bem definidas que permitem comunicação entre módulos.

## Design System e Componentes Compartilhados

A implementação de um design system robusto é fundamental para garantir consistência visual e funcional entre os módulos integrados. Este sistema inclui componentes base reutilizáveis, tokens de design padronizados, e diretrizes de interface que são seguidas por ambos os módulos.

Os componentes base incluem elementos como botões, campos de formulário, modais, tabelas, e indicadores de status que são utilizados consistentemente em ambos os módulos. Estes componentes são desenvolvidos com variações que atendem às necessidades específicas de cada contexto, mas mantêm aparência e comportamento consistentes.

Tokens de design definem cores, tipografia, espaçamentos, e outros elementos visuais que são aplicados uniformemente. Por exemplo, a cor utilizada para indicar status "ativo" de um produto é a mesma utilizada para indicar um cliente "ativo", criando uma linguagem visual coerente em todo o sistema.

A biblioteca de componentes compartilhados inclui elementos específicos para integração, como seletores de cliente que podem ser utilizados em interfaces de produtos, seletores de produto para interfaces de clientes, e componentes de busca unificada que permitem pesquisar em ambos os módulos simultaneamente.

## Estado Global e Gerenciamento de Dados

O gerenciamento de estado em uma aplicação integrada requer estratégias sofisticadas para manter dados sincronizados entre módulos sem criar acoplamento excessivo. A abordagem recomendada utiliza uma combinação de estado local para dados específicos de cada módulo e estado global para informações compartilhadas.

O estado global é gerenciado através de um store centralizado utilizando Redux Toolkit ou Zustand, que mantém informações como usuário autenticado, configurações da aplicação,

dados de sessão, e cache de informações frequentemente acessadas de ambos os módulos. Este estado é acessível por todos os componentes, mas modificações seguem padrões rigorosos para evitar inconsistências.

Cada módulo mantém seu próprio slice do estado global para dados específicos, mas pode acessar slices de outros módulos quando necessário. Por exemplo, o módulo de produtos pode acessar informações básicas de clientes para exibir histórico de compras, enquanto o módulo de clientes pode acessar dados de produtos para mostrar recomendações.

A sincronização de dados entre frontend e backend é gerenciada através de React Query ou SWR, que implementam cache inteligente, invalidação automática, e sincronização em tempo real. Estas bibliotecas facilitam o compartilhamento de dados entre módulos enquanto mantêm performance otimizada.

## Navegação e Roteamento Integrado

A navegação em uma aplicação modular integrada deve ser intuitiva e permitir transições fluidas entre diferentes contextos. O sistema de roteamento utiliza React Router com configuração hierárquica que reflete a estrutura modular da aplicação.

A estrutura de rotas principais inclui `/produtos` para o módulo de produtos, `/clientes` para o módulo de clientes, e `/vendas` para funcionalidades integradas futuras. Cada módulo define suas próprias sub-rotas, mas o shell application gerencia a navegação de nível superior e transições entre módulos.

Breadcrumbs dinâmicos são implementados para ajudar usuários a entender sua localização atual e navegar facilmente entre seções relacionadas. Por exemplo, ao visualizar o histórico de compras de um cliente, breadcrumbs mostram o caminho desde a listagem de clientes até o cliente específico e seu histórico.

Deep linking é suportado para permitir acesso direto a contextos específicos, como visualizar um produto específico a partir de uma notificação ou email. URLs são estruturadas de forma semântica e incluem informações suficientes para reconstruir o estado da aplicação.

## Interfaces de Busca e Filtros Unificados

Uma das funcionalidades mais valiosas da integração é a capacidade de buscar e filtrar informações que atravessam múltiplos módulos. A implementação de busca unificada permite que usuários encontrem rapidamente produtos, clientes, ou relacionamentos entre eles através de uma interface intuitiva.

A interface de busca global é posicionada proeminentemente na navegação principal e oferece sugestões em tempo real conforme o usuário digita. Os resultados são

categorizados por tipo (produtos, clientes, vendas) e incluem informações contextuais relevantes para cada categoria.

Filtros avançados permitem refinamento de resultados baseado em critérios específicos de cada módulo ou critérios combinados. Por exemplo, usuários podem filtrar produtos por categoria e simultaneamente por clientes que os compraram em um período específico.

A implementação utiliza debouncing para otimizar performance, evitando chamadas excessivas à API durante a digitação. Resultados são cacheados inteligentemente para melhorar responsividade em buscas repetidas.

## Dashboards e Visualizações Integradas

Dashboards que combinam informações de produtos e clientes oferecem insights valiosos para gestão do negócio. A implementação utiliza bibliotecas como Recharts ou Chart.js para criar visualizações interativas que apresentam dados de forma clara e acionável.

O dashboard principal inclui métricas como produtos mais vendidos, clientes mais ativos, tendências de vendas por categoria, e análises de performance que cruzam dados de ambos os módulos. Estas visualizações são atualizadas em tempo real e permitem drill-down para análises mais detalhadas.

Widgets modulares permitem que usuários personalizem seus dashboards conforme suas necessidades específicas. Administradores podem ter widgets focados em performance geral, enquanto vendedores podem preferir widgets que mostram oportunidades de venda ou clientes que não compram há tempo.

A responsividade é crucial para dashboards, garantindo que visualizações sejam legíveis e funcionais em dispositivos móveis. Gráficos se adaptam automaticamente ao tamanho da tela, e interações são otimizadas para touch em dispositivos móveis.

## Formulários e Workflows Integrados

Formulários que envolvem dados de ambos os módulos requerem design cuidadoso para manter usabilidade enquanto coletam informações complexas. O exemplo mais comum é o formulário de venda, que precisa capturar informações de cliente e produtos simultaneamente.

A implementação utiliza formulários multi-step que guiam usuários através do processo de forma lógica. O primeiro step pode ser seleção ou cadastro de cliente, seguido por seleção de produtos, configuração de quantidades e preços, e finalização com resumo e confirmação.

Validação em tempo real é implementada para cada step, verificando disponibilidade de produtos, validando dados de cliente, e calculando totais automaticamente. Mensagens de erro são contextuais e oferecem sugestões para resolução quando possível.

Auto-complete e sugestões inteligentes melhoram a experiência do usuário. Por exemplo, ao selecionar um cliente, o sistema pode sugerir produtos baseados no histórico de compras. Ao selecionar produtos, pode sugerir quantidades baseadas em compras anteriores.

## Notificações e Feedback Integrado

Um sistema de notificações unificado mantém usuários informados sobre eventos importantes que podem afetar ambos os módulos. Notificações podem incluir alertas sobre produtos com estoque baixo, clientes inativos há muito tempo, ou oportunidades de venda baseadas em padrões de comportamento.

A implementação utiliza toast notifications para feedback imediato de ações, notificações persistentes para informações importantes que requerem ação, e um centro de notificações para histórico e gerenciamento de alertas.

Notificações são categorizadas por prioridade e tipo, permitindo que usuários configurem suas preferências de recebimento. Integração com sistemas externos como email ou SMS pode ser implementada para notificações críticas.

## Performance e Otimização

A integração de múltiplos módulos no frontend requer atenção especial à performance para manter responsividade da aplicação. Estratégias de otimização incluem lazy loading de módulos, code splitting automático, e cache inteligente de componentes.

Lazy loading é implementado tanto a nível de módulo quanto de componente, carregando código apenas quando necessário. Isso reduz significativamente o tempo de carregamento inicial da aplicação e melhora a experiência do usuário.

Virtualização é utilizada em listas longas de produtos ou clientes para manter performance mesmo com grandes volumes de dados. Componentes como react-window ou react-virtualized renderizam apenas itens visíveis, reduzindo uso de memória e melhorando responsividade.

Memoização estratégica de componentes e hooks evita re-renderizações desnecessárias, especialmente importante em interfaces que exibem dados de múltiplos módulos simultaneamente.

## Acessibilidade e Usabilidade

A integração deve manter altos padrões de acessibilidade em todas as interfaces, seguindo diretrizes WCAG 2.1 para garantir que a aplicação seja utilizável por pessoas com diferentes necessidades e capacidades.

Navegação por teclado é implementada consistentemente em todos os módulos, com indicadores visuais claros de foco e atalhos de teclado para ações frequentes. Screen readers são suportados através de markup semântico apropriado e labels descritivos.

Contraste de cores e tamanhos de fonte seguem padrões de acessibilidade, e a aplicação oferece opções de personalização como modo escuro e ajuste de tamanho de fonte para atender diferentes preferências e necessidades.

Testes de usabilidade são conduzidos regularmente para identificar pontos de fricção na experiência integrada e oportunidades de melhoria na interface.

# Exemplos de Código e Cenários de Uso

## Cenário 1: Processo de Venda Integrado

O processo de venda representa o cenário mais crítico de integração entre os módulos de Produtos e Clientes. Este exemplo demonstra como implementar um workflow completo que valida dados de cliente, verifica disponibilidade de produtos, calcula preços, e registra a transação de forma consistente entre ambos os módulos.

### Backend - API de Integração para Vendas

Python

```
# vendas_service.py - Serviço de integração para vendas
from typing import List, Dict, Optional
from dataclasses import dataclass
from datetime import datetime
import uuid
import requests
from sqlalchemy.orm import Session
from sqlalchemy import create_engine

@dataclass
class ItemVenda:
    produto_id: str
    quantidade: int
    preco_unitario: Optional[int] = None
    desconto_percentual: Optional[float] = None

@dataclass
class DadosVenda:
    cliente_id: str
    itens: List[ItemVenda]
    observacoes: Optional[str] = None
    vendedor_id: Optional[str] = None
```

```
class VendasIntegrationService:
    def __init__(self, db_session: Session, produtos_api_url: str,
clientes_api_url: str):
        self.db = db_session
        self.produtos_api = produtos_api_url
        self.clientes_api = clientes_api_url
        self.correlation_id = str(uuid.uuid4())

    async def processar_venda(self, dados_venda: DadosVenda) -> Dict:
        """
        Processa uma venda completa integrando validações de cliente e
produto.
        Implementa padrão Saga para garantir consistência distribuída.
        """
        saga_id = str(uuid.uuid4())

        try:
            # Etapa 1: Validar cliente
            cliente_valido = await
self._validar_cliente(dados_venda.cliente_id)
            if not cliente_valido:
                raise ValueError(f"Cliente {dados_venda.cliente_id} não
encontrado ou inativo")

            # Etapa 2: Validar e reservar produtos
            produtos_reservados = await
self._validar_e_reservar_produtos(dados_venda.itens)

            # Etapa 3: Calcular totais e aplicar descontos
            totais = await self._calcular_totais(dados_venda.cliente_id,
produtos_reservados)

            # Etapa 4: Registrar venda
            venda_id = await self._registrar_venda(dados_venda,
produtos_reservados, totais)

            # Etapa 5: Atualizar histórico do cliente
            await self._atualizar_historico_cliente(dados_venda.cliente_id,
venda_id)

            # Etapa 6: Confirmar reservas de produto
            await self._confirmar_reservas_produtos(produtos_reservados)

            return {
                "venda_id": venda_id,
                "total": totais["total_final"],
                "status": "concluida",
```

```
                "correlation_id": self.correlation_id
            }

        except Exception as e:
            # Executar compensação em caso de erro
            await self._compensar_transacao(saga_id, dados_venda)
            raise e

    async def _validar_cliente(self, cliente_id: str) -> bool:
        """Valida se cliente existe e está ativo no módulo de clientes."""
        try:
            response = requests.get(
                f"
{self.clientes_api}/integration/clientes/{cliente_id}/validate",
                headers={"X-Correlation-ID": self.correlation_id},
                timeout=5
            )
            return response.status_code == 200 and
response.json().get("ativo", False)
        except requests.RequestException:
            return False

    async def _validar_e_reservar_produtos(self, itens: List[ItemVenda]) ->
List[Dict]:
        """Valida disponibilidade e reserva produtos temporariamente."""
        produtos_validados = []

        for item in itens:
            # Verificar disponibilidade
            response = requests.get(
                f"
{self.produtos_api}/integration/produtos/{item.produto_id}/availability",
                params={"quantidade": item.quantidade},
                headers={"X-Correlation-ID": self.correlation_id}
            )

            if response.status_code != 200:
                raise ValueError(f"Produto {item.produto_id} não disponível")

            produto_info = response.json()

            # Reservar produto temporariamente
            reserva_response = requests.post(
                f"
{self.produtos_api}/integration/produtos/{item.produto_id}/reserve",
                json={
                    "quantidade": item.quantidade,
                    "correlation_id": self.correlation_id,
```

```
                    "ttl_minutes": 10
                },
                headers={"X-Correlation-ID": self.correlation_id}
            )

            if reserva_response.status_code != 201:
                raise ValueError(f"Falha ao reservar produto
{item.produto_id}")

            produtos_validados.append({
                "produto_id": item.produto_id,
                "quantidade": item.quantidade,
                "preco_unitario": produto_info["preco_venda"],
                "nome": produto_info["nome"],
                "reserva_id": reserva_response.json()["reserva_id"]
            })

        return produtos_validados

    async def _calcular_totais(self, cliente_id: str, produtos: List[Dict]) -
> Dict:
        """Calcula totais aplicando descontos e promoções."""
        subtotal = sum(p["preco_unitario"] * p["quantidade"] for p in
produtos)

        # Verificar elegibilidade para descontos
        desconto_response = requests.post(
            f"
{self.clientes_api}/integration/clientes/{cliente_id}/calculate-discount",
            json={
                "produtos": [{"id": p["produto_id"], "valor":
p["preco_unitario"] * p["quantidade"]}
                             for p in produtos],
                "subtotal": subtotal
            },
            headers={"X-Correlation-ID": self.correlation_id}
        )

        desconto_info = desconto_response.json() if
desconto_response.status_code == 200 else {}
        desconto_valor = desconto_info.get("desconto_valor", 0)
        desconto_percentual = desconto_info.get("desconto_percentual", 0)

        return {
            "subtotal": subtotal,
            "desconto_valor": desconto_valor,
            "desconto_percentual": desconto_percentual,
            "total_final": subtotal - desconto_valor
```

```
        }

    async def _registrar_venda(self, dados_venda: DadosVenda, produtos:
List[Dict], totais: Dict) -> str:
        """Registra a venda na tabela de integração."""
        venda_id = str(uuid.uuid4())

        # Inserir cabeçalho da venda
        venda_query = """
        INSERT INTO integracao.vendas
        (id, cliente_id, data_venda, subtotal, desconto_valor, total_final,
         observacoes, vendedor_id, correlation_id, status)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """

        self.db.execute(venda_query, (
            venda_id, dados_venda.cliente_id, datetime.utcnow(),
            totais["subtotal"], totais["desconto_valor"],
totais["total_final"],
            dados_venda.observacoes, dados_venda.vendedor_id,
            self.correlation_id, "processando"
        ))

        # Inserir itens da venda
        for produto in produtos:
            item_query = """
            INSERT INTO integracao.vendas_itens
            (venda_id, produto_id, quantidade, preco_unitario, subtotal)
            VALUES (%s, %s, %s, %s, %s)
            """

            self.db.execute(item_query, (
                venda_id, produto["produto_id"], produto["quantidade"],
                produto["preco_unitario"],
                produto["preco_unitario"] * produto["quantidade"]
            ))

        self.db.commit()
        return venda_id

    async def _atualizar_historico_cliente(self, cliente_id: str, venda_id:
str):
        """Notifica módulo de clientes sobre nova venda."""
        requests.post(
            f"{self.clientes_api}/integration/clientes/{cliente_id}/add-
purchase",
            json={"venda_id": venda_id, "correlation_id":
self.correlation_id},
```

```
            headers={"X-Correlation-ID": self.correlation_id}
        )

    async def _confirmar_reservas_produtos(self, produtos: List[Dict]):
        """Confirma reservas temporárias de produtos."""
        for produto in produtos:
            requests.post(
                f"
{self.produtos_api}/integration/produtos/{produto['produto_id']}/confirm-
reserve",
                json={"reserva_id": produto["reserva_id"]},
                headers={"X-Correlation-ID": self.correlation_id}
            )

    async def _compensar_transacao(self, saga_id: str, dados_venda:
DadosVenda):
        """Executa operações de compensação em caso de falha."""
        # Cancelar reservas de produtos
        for item in dados_venda.itens:
            requests.delete(
                f"
{self.produtos_api}/integration/produtos/{item.produto_id}/cancel-reserve",
                headers={"X-Correlation-ID": self.correlation_id}
            )

        # Remover registros de venda parciais
        self.db.execute(
            "DELETE FROM integracao.vendas WHERE correlation_id = %s",
            (self.correlation_id,)
        )
        self.db.commit()

# Endpoint FastAPI para processar vendas
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session

router = APIRouter(prefix="/api/v1/vendas", tags=["vendas"])

@router.post("/processar")
async def processar_venda(
    dados_venda: DadosVenda,
    db: Session = Depends(get_db)
):
    """Endpoint para processar venda integrando produtos e clientes."""
    try:
        service = VendasIntegrationService(
            db_session=db,
            produtos_api_url="http://backend-produtos:8000/api/v1",
```

```
            clientes_api_url="http://backend-clientes:8000/api/v1"
        )

        resultado = await service.processar_venda(dados_venda)
        return resultado

    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
    except Exception as e:
        raise HTTPException(status_code=500, detail="Erro interno do
servidor")
```

## Frontend - Componente de Venda Integrado

TSX

```
// VendaForm.tsx - Componente para processo de venda integrado
import React, { useState, useEffect } from 'react';
import { useForm, useFieldArray } from 'react-hook-form';
import { useMutation, useQuery } from '@tanstack/react-query';
import { Search, Plus, Trash2, Calculator } from 'lucide-react';

interface ItemVenda {
  produto_id: string;
  produto_nome?: string;
  quantidade: number;
  preco_unitario: number;
  subtotal: number;
}

interface FormVenda {
  cliente_id: string;
  cliente_nome?: string;
  itens: ItemVenda[];
  observacoes?: string;
}

export const VendaForm: React.FC = () => {
  const [clienteBusca, setClienteBusca] = useState('');
  const [produtoBusca, setProdutoBusca] = useState('');
  const [mostrarResumo, setMostrarResumo] = useState(false);

  const { register, control, watch, setValue, handleSubmit, formState: {
errors } } = useForm<FormVenda>({
    defaultValues: {
      itens: [{ produto_id: '', quantidade: 1, preco_unitario: 0, subtotal:
0 }]
```

```
    }
  });

  const { fields, append, remove } = useFieldArray({
    control,
    name: 'itens'
  });

  const watchedItens = watch('itens');
  const watchedClienteId = watch('cliente_id');

  // Query para buscar clientes
  const { data: clientesEncontrados } = useQuery({
    queryKey: ['clientes-busca', clienteBusca],
    queryFn: () => buscarClientes(clienteBusca),
    enabled: clienteBusca.length > 2
  });

  // Query para buscar produtos
  const { data: produtosEncontrados } = useQuery({
    queryKey: ['produtos-busca', produtoBusca],
    queryFn: () => buscarProdutos(produtoBusca),
    enabled: produtoBusca.length > 2
  });

  // Mutation para processar venda
  const processarVendaMutation = useMutation({
    mutationFn: processarVenda,
    onSuccess: (data) => {
      alert(`Venda processada com sucesso! ID: ${data.venda_id}`);
      // Reset form ou redirecionar
    },
    onError: (error) => {
      alert(`Erro ao processar venda: ${error.message}`);
    }
  });

  // Calcular totais automaticamente
  useEffect(() => {
    watchedItens.forEach((item, index) => {
      const subtotal = item.quantidade * item.preco_unitario;
      setValue(`itens.${index}.subtotal`, subtotal);
    });
  }, [watchedItens, setValue]);

  const totalGeral = watchedItens.reduce((total, item) => total +
item.subtotal, 0);
```

```
  const selecionarCliente = (cliente: any) => {
    setValue('cliente_id', cliente.id);
    setValue('cliente_nome', cliente.nome);
    setClienteBusca('');
  };

  const selecionarProduto = (produto: any, index: number) => {
    setValue(`itens.${index}.produto_id`, produto.id);
    setValue(`itens.${index}.produto_nome`, produto.nome);
    setValue(`itens.${index}.preco_unitario`, produto.preco_venda / 100);
    setProdutoBusca('');
  };

  const adicionarItem = () => {
    append({ produto_id: '', quantidade: 1, preco_unitario: 0, subtotal: 0
});
  };

  const onSubmit = (data: FormVenda) => {
    const dadosVenda = {
      cliente_id: data.cliente_id,
      itens: data.itens.map(item => ({
        produto_id: item.produto_id,
        quantidade: item.quantidade,
        preco_unitario: Math.round(item.preco_unitario * 100) // Converter
para centavos
      })),
      observacoes: data.observacoes
    };

    processarVendaMutation.mutate(dadosVenda);
  };

  return (
    <div className="max-w-4xl mx-auto p-6 bg-white rounded-lg shadow-lg">
      <h2 className="text-2xl font-bold mb-6 text-gray-900">Nova Venda</h2>

      <form onSubmit={handleSubmit(onSubmit)} className="space-y-6">
        {/* Seleção de Cliente */}
        <div className="bg-gray-50 p-4 rounded-lg">
          <label className="block text-sm font-medium text-gray-700 mb-2">
            Cliente
          </label>

          {!watchedClienteId ? (
            <div className="relative">
              <div className="flex">
                <Search className="absolute left-3 top-3 h-4 w-4 text-gray-
```

```
400" />
                <input
                  type="text"
                  placeholder="Buscar cliente por nome, email ou
documento..."
                  value={clienteBusca}
                  onChange={(e) => setClienteBusca(e.target.value)}
                  className="w-full pl-10 pr-4 py-2 border border-gray-300
rounded-md focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
                />
              </div>

              {clientesEncontrados && clientesEncontrados.length > 0 && (
                <div className="absolute z-10 w-full mt-1 bg-white border
border-gray-300 rounded-md shadow-lg max-h-60 overflow-y-auto">
                  {clientesEncontrados.map((cliente: any) => (
                    <div
                      key={cliente.id}
                      onClick={() => selecionarCliente(cliente)}
                      className="p-3 hover:bg-gray-100 cursor-pointer border-
b border-gray-200 last:border-b-0"
                    >
                      <div className="font-medium text-gray-900">
{cliente.nome}</div>
                      <div className="text-sm text-gray-600">{cliente.email}
</div>
                      <div className="text-sm text-gray-500">
{cliente.documento}</div>
                    </div>
                  ))}
                </div>
              )}
            </div>
          ) : (
            <div className="flex items-center justify-between p-3 bg-blue-50
border border-blue-200 rounded-md">
              <div>
                <div className="font-medium text-blue-900">
{watch('cliente_nome')}</div>
                <div className="text-sm text-blue-700">ID: {watchedClienteId}
</div>
              </div>
              <button
                type="button"
                onClick={() => {
                  setValue('cliente_id', '');
                  setValue('cliente_nome', '');
                }}
```

```
                  className="text-blue-600 hover:text-blue-800"
                >
                  Alterar
                </button>
              </div>
            )}
          </div>

          {/* Itens da Venda */}
          <div className="bg-gray-50 p-4 rounded-lg">
            <div className="flex items-center justify-between mb-4">
              <label className="block text-sm font-medium text-gray-700">
                Produtos
              </label>
              <button
                type="button"
                onClick={adicionarItem}
                className="flex items-center px-3 py-2 text-sm bg-blue-600
text-white rounded-md hover:bg-blue-700"
              >
                <Plus className="h-4 w-4 mr-1" />
                Adicionar Item
              </button>
            </div>

            <div className="space-y-4">
              {fields.map((field, index) => (
                <div key={field.id} className="bg-white p-4 rounded-md border
border-gray-200">
                  <div className="grid grid-cols-1 md:grid-cols-12 gap-4 items-
end">
                    {/* Seleção de Produto */}
                    <div className="md:col-span-5">
                      <label className="block text-xs font-medium text-gray-
700 mb-1">
                        Produto
                      </label>
                      {!watchedItens[index]?.produto_id ? (
                        <div className="relative">
                          <Search className="absolute left-3 top-3 h-4 w-4
text-gray-400" />
                          <input
                            type="text"
                            placeholder="Buscar produto..."
                            value={produtoBusca}
                            onChange={(e) => setProdutoBusca(e.target.value)}
                            className="w-full pl-10 pr-4 py-2 border border-
gray-300 rounded-md focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
```

```
                        />

                        {produtosEncontrados && produtosEncontrados.length >
0 && (
                          <div className="absolute z-10 w-full mt-1 bg-white
border border-gray-300 rounded-md shadow-lg max-h-40 overflow-y-auto">
                            {produtosEncontrados.map((produto: any) => (
                              <div
                                key={produto.id}
                                onClick={() => selecionarProduto(produto,
index)}
                                className="p-2 hover:bg-gray-100 cursor-
pointer border-b border-gray-200 last:border-b-0"
                              >
                                <div className="font-medium text-sm">
{produto.nome}</div>
                                <div className="text-xs text-gray-600">
                                  R$ {(produto.preco_venda / 100).toFixed(2)}
                                </div>
                              </div>
                            ))}
                          </div>
                        )}
                      </div>
                    ) : (
                      <div className="p-2 bg-green-50 border border-green-
200 rounded-md">
                        <div className="text-sm font-medium text-green-900">
                          {watchedItens[index]?.produto_nome}
                        </div>
                        <button
                          type="button"
                          onClick={() => {
                            setValue(`itens.${index}.produto_id`, '');
                            setValue(`itens.${index}.produto_nome`, '');
                          }}
                          className="text-xs text-green-600 hover:text-green-
800"
                        >
                          Alterar produto
                        </button>
                      </div>
                    )}
                  </div>

                  {/* Quantidade */}
                  <div className="md:col-span-2">
                    <label className="block text-xs font-medium text-gray-
```

```
700 mb-1">
                      Quantidade
                    </label>
                    <input
                      type="number"
                      min="1"
                      {...register(`itens.${index}.quantidade`, {
                        required: true,
                        min: 1,
                        valueAsNumber: true
                      })}
                      className="w-full px-3 py-2 border border-gray-300
rounded-md focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
                    />
                  </div>

                  {/* Preço Unitário */}
                  <div className="md:col-span-2">
                    <label className="block text-xs font-medium text-gray-
700 mb-1">
                      Preço Unit.
                    </label>
                    <input
                      type="number"
                      step="0.01"
                      min="0"
                      {...register(`itens.${index}.preco_unitario`, {
                        required: true,
                        min: 0,
                        valueAsNumber: true
                      })}
                      className="w-full px-3 py-2 border border-gray-300
rounded-md focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
                    />
                  </div>

                  {/* Subtotal */}
                  <div className="md:col-span-2">
                    <label className="block text-xs font-medium text-gray-
700 mb-1">
                      Subtotal
                    </label>
                    <div className="px-3 py-2 bg-gray-100 border border-gray-
300 rounded-md text-right font-medium">
                      R$ {watchedItens[index]?.subtotal?.toFixed(2) ||
'0.00'}
                    </div>
                  </div>
```

```
                {/* Remover Item */}
                <div className="md:col-span-1">
                  {fields.length > 1 && (
                    <button
                      type="button"
                      onClick={() => remove(index)}
                      className="w-full p-2 text-red-600 hover:text-red-
800 hover:bg-red-50 rounded-md"
                    >
                      <Trash2 className="h-4 w-4 mx-auto" />
                    </button>
                  )}
                </div>
              </div>
            </div>
          ))}
        </div>
      </div>

      {/* Observações */}
      <div>
        <label className="block text-sm font-medium text-gray-700 mb-2">
          Observações
        </label>
        <textarea
          {...register('observacoes')}
          rows={3}
          className="w-full px-3 py-2 border border-gray-300 rounded-md
focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
          placeholder="Observações adicionais sobre a venda..."
        />
      </div>

      {/* Resumo da Venda */}
      <div className="bg-blue-50 p-4 rounded-lg border border-blue-200">
        <div className="flex items-center justify-between mb-2">
          <span className="text-lg font-medium text-blue-900">Total da
Venda</span>
          <Calculator className="h-5 w-5 text-blue-600" />
        </div>
        <div className="text-2xl font-bold text-blue-900">
          R$ {totalGeral.toFixed(2)}
        </div>
        <div className="text-sm text-blue-700 mt-1">
          {watchedItens.length} item(ns) • Cliente: {watch('cliente_nome')
|| 'Não selecionado'}
        </div>
```

```
        </div>

        {/* Botões de Ação */}
        <div className="flex space-x-4">
          <button
            type="submit"
            disabled={!watchedClienteId || watchedItens.some(item =>
!item.produto_id) || processarVendaMutation.isPending}
            className="flex-1 bg-green-600 text-white py-3 px-4 rounded-md
hover:bg-green-700 disabled:bg-gray-400 disabled:cursor-not-allowed font-
medium"
          >
            {processarVendaMutation.isPending ? 'Processando...' :
'Finalizar Venda'}
          </button>

          <button
            type="button"
            onClick={() => setMostrarResumo(!mostrarResumo)}
            className="px-6 py-3 border border-gray-300 text-gray-700
rounded-md hover:bg-gray-50"
          >
            {mostrarResumo ? 'Ocultar' : 'Mostrar'} Resumo
          </button>
        </div>
      </form>

      {/* Modal de Resumo */}
      {mostrarResumo && (
        <div className="fixed inset-0 bg-black bg-opacity-50 flex items-
center justify-center z-50">
          <div className="bg-white p-6 rounded-lg max-w-md w-full mx-4">
            <h3 className="text-lg font-bold mb-4">Resumo da Venda</h3>

            <div className="space-y-2 mb-4">
              <div className="flex justify-between">
                <span>Cliente:</span>
                <span className="font-medium">{watch('cliente_nome') || 'Não
selecionado'}</span>
              </div>

              <div className="border-t pt-2">
                <div className="text-sm font-medium mb-2">Itens:</div>
                {watchedItens.map((item, index) => (
                  <div key={index} className="flex justify-between text-sm">
                    <span>{item.produto_nome || 'Produto não selecionado'}
</span>
                    <span>R$ {item.subtotal.toFixed(2)}</span>
```

```
                  </div>
                ))}
              </div>

              <div className="border-t pt-2 flex justify-between font-bold">
                <span>Total:</span>
                <span>R$ {totalGeral.toFixed(2)}</span>
              </div>
            </div>

            <button
              onClick={() => setMostrarResumo(false)}
              className="w-full bg-blue-600 text-white py-2 rounded-md
hover:bg-blue-700"
            >
              Fechar
            </button>
          </div>
        </div>
      )}
    </div>
  );
};

// Funções auxiliares para API
async function buscarClientes(termo: string) {
  const response = await fetch(`/api/v1/clientes/search/${termo}?limit=5`);
  return response.json();
}

async function buscarProdutos(termo: string) {
  const response = await fetch(`/api/v1/produtos/search/${termo}?limit=5`);
  return response.json();
}

async function processarVenda(dadosVenda: any) {
  const response = await fetch('/api/v1/vendas/processar', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(dadosVenda)
  });

  if (!response.ok) {
    throw new Error('Falha ao processar venda');
  }
```

```
  return response.json();
}
```

## Cenário 2: Dashboard Integrado com Análises Combinadas

Este cenário demonstra como criar um dashboard que combina informações de produtos e clientes para fornecer insights valiosos sobre o negócio.

## Backend - API de Analytics Integrada

Python

```
# analytics_service.py - Serviço de análises integradas
from typing import Dict, List, Optional
from datetime import datetime, timedelta
from sqlalchemy.orm import Session
from sqlalchemy import text

class AnalyticsIntegrationService:
    def __init__(self, db_session: Session):
        self.db = db_session

    def get_dashboard_data(self, periodo_dias: int = 30) -> Dict:
        """Retorna dados consolidados para dashboard."""
        data_inicio = datetime.utcnow() - timedelta(days=periodo_dias)

        return {
            "resumo_geral": self._get_resumo_geral(data_inicio),
            "produtos_mais_vendidos":
self._get_produtos_mais_vendidos(data_inicio),
            "clientes_mais_ativos":
self._get_clientes_mais_ativos(data_inicio),
            "vendas_por_categoria":
self._get_vendas_por_categoria(data_inicio),
            "tendencias_vendas": self._get_tendencias_vendas(data_inicio),
            "analise_cliente_produto":
self._get_analise_cliente_produto(data_inicio)
        }

    def _get_resumo_geral(self, data_inicio: datetime) -> Dict:
        """Resumo geral de vendas e métricas."""
        query = text("""
        SELECT
            COUNT(DISTINCT v.id) as total_vendas,
            COUNT(DISTINCT v.cliente_id) as clientes_unicos,
            COUNT(DISTINCT vi.produto_id) as produtos_vendidos,
            SUM(v.total_final) as receita_total,
```

```
            AVG(v.total_final) as ticket_medio
        FROM integracao.vendas v
        JOIN integracao.vendas_itens vi ON v.id = vi.venda_id
        WHERE v.data_venda >= :data_inicio
        AND v.status = 'concluida'
        """)

        result = self.db.execute(query, {"data_inicio":
data_inicio}).fetchone()

        return {
            "total_vendas": result.total_vendas or 0,
            "clientes_unicos": result.clientes_unicos or 0,
            "produtos_vendidos": result.produtos_vendidos or 0,
            "receita_total": float(result.receita_total or 0) / 100,  #
Converter de centavos
            "ticket_medio": float(result.ticket_medio or 0) / 100
        }

    def _get_produtos_mais_vendidos(self, data_inicio: datetime, limit: int
= 10) -> List[Dict]:
        """Top produtos por quantidade vendida."""
        query = text("""
        SELECT
            p.id,
            p.nome,
            p.preco_venda,
            c.nome as categoria_nome,
            m.nome as marca_nome,
            SUM(vi.quantidade) as quantidade_vendida,
            SUM(vi.subtotal) as receita_produto,
            COUNT(DISTINCT v.cliente_id) as clientes_unicos
        FROM integracao.vendas_itens vi
        JOIN integracao.vendas v ON vi.venda_id = v.id
        JOIN produtos.produtos p ON vi.produto_id = p.id
        LEFT JOIN produtos.categorias c ON p.categoria_id = c.id
        LEFT JOIN produtos.marcas m ON p.marca_id = m.id
        WHERE v.data_venda >= :data_inicio
        AND v.status = 'concluida'
        GROUP BY p.id, p.nome, p.preco_venda, c.nome, m.nome
        ORDER BY quantidade_vendida DESC
        LIMIT :limit
        """)

        results = self.db.execute(query, {
            "data_inicio": data_inicio,
            "limit": limit
        }).fetchall()
```

```
        return [
            {
                "produto_id": row.id,
                "nome": row.nome,
                "preco_venda": float(row.preco_venda) / 100,
                "categoria": row.categoria_nome,
                "marca": row.marca_nome,
                "quantidade_vendida": row.quantidade_vendida,
                "receita": float(row.receita_produto) / 100,
                "clientes_unicos": row.clientes_unicos
            }
            for row in results
        ]

    def _get_clientes_mais_ativos(self, data_inicio: datetime, limit: int =
10) -> List[Dict]:
        """Top clientes por valor de compras."""
        query = text("""
        SELECT
            c.id,
            c.nome,
            c.email,
            c.tipo_pessoa,
            COUNT(v.id) as total_compras,
            SUM(v.total_final) as valor_total,
            AVG(v.total_final) as ticket_medio,
            MAX(v.data_venda) as ultima_compra,
            COUNT(DISTINCT vi.produto_id) as produtos_diferentes
        FROM integracao.vendas v
        JOIN clientes.clientes c ON v.cliente_id = c.id
        JOIN integracao.vendas_itens vi ON v.id = vi.venda_id
        WHERE v.data_venda >= :data_inicio
        AND v.status = 'concluida'
        GROUP BY c.id, c.nome, c.email, c.tipo_pessoa
        ORDER BY valor_total DESC
        LIMIT :limit
        """)

        results = self.db.execute(query, {
            "data_inicio": data_inicio,
            "limit": limit
        }).fetchall()

        return [
            {
                "cliente_id": row.id,
                "nome": row.nome,
```

```
                "email": row.email,
                "tipo_pessoa": row.tipo_pessoa,
                "total_compras": row.total_compras,
                "valor_total": float(row.valor_total) / 100,
                "ticket_medio": float(row.ticket_medio) / 100,
                "ultima_compra": row.ultima_compra.isoformat(),
                "produtos_diferentes": row.produtos_diferentes
            }
            for row in results
        ]

    def _get_vendas_por_categoria(self, data_inicio: datetime) -> List[Dict]:
        """Vendas agrupadas por categoria de produto."""
        query = text("""
        SELECT
            COALESCE(c.nome, 'Sem Categoria') as categoria,
            COUNT(vi.id) as itens_vendidos,
            SUM(vi.quantidade) as quantidade_total,
            SUM(vi.subtotal) as receita_categoria,
            COUNT(DISTINCT v.cliente_id) as clientes_unicos
        FROM integracao.vendas_itens vi
        JOIN integracao.vendas v ON vi.venda_id = v.id
        JOIN produtos.produtos p ON vi.produto_id = p.id
        LEFT JOIN produtos.categorias c ON p.categoria_id = c.id
        WHERE v.data_venda >= :data_inicio
        AND v.status = 'concluida'
        GROUP BY c.nome
        ORDER BY receita_categoria DESC
        """)

        results = self.db.execute(query, {"data_inicio":
data_inicio}).fetchall()

        return [
            {
                "categoria": row.categoria,
                "itens_vendidos": row.itens_vendidos,
                "quantidade_total": row.quantidade_total,
                "receita": float(row.receita_categoria) / 100,
                "clientes_unicos": row.clientes_unicos
            }
            for row in results
        ]

    def _get_tendencias_vendas(self, data_inicio: datetime) -> List[Dict]:
        """Tendências de vendas por dia."""
        query = text("""
        SELECT
```

```
            DATE(v.data_venda) as data,
            COUNT(v.id) as vendas_dia,
            SUM(v.total_final) as receita_dia,
            COUNT(DISTINCT v.cliente_id) as clientes_unicos_dia,
            AVG(v.total_final) as ticket_medio_dia
        FROM integracao.vendas v
        WHERE v.data_venda >= :data_inicio
        AND v.status = 'concluida'
        GROUP BY DATE(v.data_venda)
        ORDER BY data
        """)

        results = self.db.execute(query, {"data_inicio":
data_inicio}).fetchall()

        return [
            {
                "data": row.data.isoformat(),
                "vendas": row.vendas_dia,
                "receita": float(row.receita_dia) / 100,
                "clientes_unicos": row.clientes_unicos_dia,
                "ticket_medio": float(row.ticket_medio_dia) / 100
            }
            for row in results
        ]

    def _get_analise_cliente_produto(self, data_inicio: datetime) -> Dict:
        """Análise de relacionamento cliente-produto."""
        # Produtos mais comprados por tipo de cliente
        query_tipo_cliente = text("""
        SELECT
            c.tipo_pessoa,
            p.nome as produto_nome,
            cat.nome as categoria,
            SUM(vi.quantidade) as quantidade,
            SUM(vi.subtotal) as receita
        FROM integracao.vendas v
        JOIN clientes.clientes c ON v.cliente_id = c.id
        JOIN integracao.vendas_itens vi ON v.id = vi.venda_id
        JOIN produtos.produtos p ON vi.produto_id = p.id
        LEFT JOIN produtos.categorias cat ON p.categoria_id = cat.id
        WHERE v.data_venda >= :data_inicio
        AND v.status = 'concluida'
        GROUP BY c.tipo_pessoa, p.nome, cat.nome
        ORDER BY c.tipo_pessoa, quantidade DESC
        """)

        results = self.db.execute(query_tipo_cliente, {"data_inicio":
```

```
data_inicio}).fetchall()

        analise_por_tipo = {}
        for row in results:
            if row.tipo_pessoa not in analise_por_tipo:
                analise_por_tipo[row.tipo_pessoa] = []

            analise_por_tipo[row.tipo_pessoa].append({
                "produto": row.produto_nome,
                "categoria": row.categoria,
                "quantidade": row.quantidade,
                "receita": float(row.receita) / 100
            })

        return {
            "por_tipo_cliente": analise_por_tipo
        }

# Endpoint FastAPI para analytics
@router.get("/dashboard")
async def get_dashboard_analytics(
    periodo_dias: int = 30,
    db: Session = Depends(get_db)
):
    """Endpoint para dados do dashboard integrado."""
    service = AnalyticsIntegrationService(db)
    return service.get_dashboard_data(periodo_dias)
```

## Frontend - Dashboard Integrado

TSX

```
// DashboardIntegrado.tsx - Dashboard com dados de produtos e clientes
import React from 'react';
import { useQuery } from '@tanstack/react-query';
import {
  BarChart, Bar, XAxis, YAxis, CartesianGrid, Tooltip, Legend,
ResponsiveContainer,
  LineChart, Line, PieChart, Pie, Cell, Area, AreaChart
} from 'recharts';
import {
  TrendingUp, Users, Package, DollarSign, ShoppingCart,
  Calendar, Target, Award
} from 'lucide-react';

const COLORS = ['#0088FE', '#00C49F', '#FFBB28', '#FF8042', '#8884D8'];
```

```
export const DashboardIntegrado: React.FC = () => {
  const { data: dashboardData, isLoading, error } = useQuery({
    queryKey: ['dashboard-analytics'],
    queryFn: () => fetch('/api/v1/analytics/dashboard').then(res =>
res.json()),
    refetchInterval: 5 * 60 * 1000 // Atualizar a cada 5 minutos
  });

  if (isLoading) {
    return (
      <div className="flex items-center justify-center h-64">
        <div className="animate-spin rounded-full h-32 w-32 border-b-2
border-blue-600"></div>
      </div>
    );
  }

  if (error) {
    return (
      <div className="bg-red-50 border border-red-200 rounded-md p-4">
        <p className="text-red-800">Erro ao carregar dados do dashboard</p>
      </div>
    );
  }

  const {
    resumo_geral,
    produtos_mais_vendidos,
    clientes_mais_ativos,
    vendas_por_categoria,
    tendencias_vendas,
    analise_cliente_produto
  } = dashboardData;

  return (
    <div className="space-y-6 p-6">
      {/* Header */}
      <div className="flex items-center justify-between">
        <h1 className="text-3xl font-bold text-gray-900">Dashboard
Integrado</h1>
        <div className="flex items-center space-x-2 text-sm text-gray-600">
          <Calendar className="h-4 w-4" />
          <span>Últimos 30 dias</span>
        </div>
      </div>

      {/* Cards de Resumo */}
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-5 gap-6">
```

```
        <div className="bg-white p-6 rounded-lg shadow-md border border-gray-
200">
          <div className="flex items-center">
            <div className="p-2 bg-blue-100 rounded-lg">
              <ShoppingCart className="h-6 w-6 text-blue-600" />
            </div>
            <div className="ml-4">
              <p className="text-sm font-medium text-gray-600">Total de
Vendas</p>
              <p className="text-2xl font-bold text-gray-900">
{resumo_geral.total_vendas}</p>
            </div>
          </div>
        </div>

        <div className="bg-white p-6 rounded-lg shadow-md border border-gray-
200">
          <div className="flex items-center">
            <div className="p-2 bg-green-100 rounded-lg">
              <DollarSign className="h-6 w-6 text-green-600" />
            </div>
            <div className="ml-4">
              <p className="text-sm font-medium text-gray-600">Receita
Total</p>
              <p className="text-2xl font-bold text-gray-900">
                R$ {resumo_geral.receita_total.toLocaleString('pt-BR', {
minimumFractionDigits: 2 })}
              </p>
            </div>
          </div>
        </div>

        <div className="bg-white p-6 rounded-lg shadow-md border border-gray-
200">
          <div className="flex items-center">
            <div className="p-2 bg-purple-100 rounded-lg">
              <Users className="h-6 w-6 text-purple-600" />
            </div>
            <div className="ml-4">
              <p className="text-sm font-medium text-gray-600">Clientes
Únicos</p>
              <p className="text-2xl font-bold text-gray-900">
{resumo_geral.clientes_unicos}</p>
            </div>
          </div>
        </div>

        <div className="bg-white p-6 rounded-lg shadow-md border border-gray-
```

```
200">
          <div className="flex items-center">
            <div className="p-2 bg-orange-100 rounded-lg">
              <Package className="h-6 w-6 text-orange-600" />
            </div>
            <div className="ml-4">
              <p className="text-sm font-medium text-gray-600">Produtos
Vendidos</p>
              <p className="text-2xl font-bold text-gray-900">
{resumo_geral.produtos_vendidos}</p>
            </div>
          </div>
        </div>

        <div className="bg-white p-6 rounded-lg shadow-md border border-gray-
200">
          <div className="flex items-center">
            <div className="p-2 bg-red-100 rounded-lg">
              <Target className="h-6 w-6 text-red-600" />
            </div>
            <div className="ml-4">
              <p className="text-sm font-medium text-gray-600">Ticket
Médio</p>
              <p className="text-2xl font-bold text-gray-900">
                R$ {resumo_geral.ticket_medio.toLocaleString('pt-BR', {
minimumFractionDigits: 2 })}
              </p>
            </div>
          </div>
        </div>
      </div>

      {/* Gráficos Principais */}
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        {/* Tendências de Vendas */}
        <div className="bg-white p-6 rounded-lg shadow-md border border-gray-
200">
          <h3 className="text-lg font-semibold text-gray-900 mb-
4">Tendências de Vendas</h3>
          <ResponsiveContainer width="100%" height={300}>
            <AreaChart data={tendencias_vendas}>
              <CartesianGrid strokeDasharray="3 3" />
              <XAxis
                dataKey="data"
                tickFormatter={(value) => new
Date(value).toLocaleDateString('pt-BR')}
              />
              <YAxis />
```

```
              <Tooltip
                labelFormatter={(value) => new
Date(value).toLocaleDateString('pt-BR')}
                formatter={(value, name) => [
                  name === 'receita' ? `R$ ${value.toFixed(2)}` : value,
                  name === 'receita' ? 'Receita' : 'Vendas'
                ]}
              />
              <Area
                type="monotone"
                dataKey="receita"
                stackId="1"
                stroke="#8884d8"
                fill="#8884d8"
                fillOpacity={0.6}
              />
              <Area
                type="monotone"
                dataKey="vendas"
                stackId="2"
                stroke="#82ca9d"
                fill="#82ca9d"
                fillOpacity={0.6}
              />
            </AreaChart>
          </ResponsiveContainer>
        </div>

        {/* Vendas por Categoria */}
        <div className="bg-white p-6 rounded-lg shadow-md border border-gray-
200">
          <h3 className="text-lg font-semibold text-gray-900 mb-4">Vendas
por Categoria</h3>
          <ResponsiveContainer width="100%" height={300}>
            <PieChart>
              <Pie
                data={vendas_por_categoria}
                cx="50%"
                cy="50%"
                labelLine={false}
                label={({ categoria, percent }) => `${categoria} ${(percent
* 100).toFixed(0)}%`}
                outerRadius={80}
                fill="#8884d8"
                dataKey="receita"
              >
                {vendas_por_categoria.map((entry, index) => (
                  <Cell key={`cell-${index}`} fill={COLORS[index %
```

```
COLORS.length]} />
                ))}
              </Pie>
              <Tooltip formatter={(value) => [`R$ ${value.toFixed(2)}`,
'Receita']} />
            </PieChart>
          </ResponsiveContainer>
        </div>
      </div>

      {/* Tabelas de Top Performers */}
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        {/* Top Produtos */}
        <div className="bg-white p-6 rounded-lg shadow-md border border-gray-
200">
          <div className="flex items-center justify-between mb-4">
            <h3 className="text-lg font-semibold text-gray-900">Top
Produtos</h3>
            <Award className="h-5 w-5 text-yellow-500" />
          </div>
          <div className="overflow-x-auto">
            <table className="min-w-full divide-y divide-gray-200">
              <thead className="bg-gray-50">
                <tr>
                  <th className="px-6 py-3 text-left text-xs font-medium
text-gray-500 uppercase tracking-wider">
                    Produto
                  </th>
                  <th className="px-6 py-3 text-left text-xs font-medium
text-gray-500 uppercase tracking-wider">
                    Qtd. Vendida
                  </th>
                  <th className="px-6 py-3 text-left text-xs font-medium
text-gray-500 uppercase tracking-wider">
                    Receita
                  </th>
                </tr>
              </thead>
              <tbody className="bg-white divide-y divide-gray-200">
                {produtos_mais_vendidos.slice(0, 5).map((produto, index) => (
                  <tr key={produto.produto_id} className={index % 2 === 0 ?
'bg-white' : 'bg-gray-50'}>
                    <td className="px-6 py-4 whitespace-nowrap">
                      <div className="text-sm font-medium text-gray-900">
{produto.nome}</div>
                      <div className="text-sm text-gray-500">
{produto.categoria}</div>
                    </td>
```

```
                    <td className="px-6 py-4 whitespace-nowrap text-sm text-
gray-900">
                      {produto.quantidade_vendida}
                    </td>
                    <td className="px-6 py-4 whitespace-nowrap text-sm text-
gray-900">
                      R$ {produto.receita.toFixed(2)}
                    </td>
                  </tr>
                ))}
              </tbody>
            </table>
          </div>
        </div>

        {/* Top Clientes */}
        <div className="bg-white p-6 rounded-lg shadow-md border border-gray-
200">
          <div className="flex items-center justify-between mb-4">
            <h3 className="text-lg font-semibold text-gray-900">Top
Clientes</h3>
            <Users className="h-5 w-5 text-blue-500" />
          </div>
          <div className="overflow-x-auto">
            <table className="min-w-full divide-y divide-gray-200">
              <thead className="bg-gray-50">
                <tr>
                  <th className="px-6 py-3 text-left text-xs font-medium
text-gray-500 uppercase tracking-wider">
                    Cliente
                  </th>
                  <th className="px-6 py-3 text-left text-xs font-medium
text-gray-500 uppercase tracking-wider">
                    Compras
                  </th>
                  <th className="px-6 py-3 text-left text-xs font-medium
text-gray-500 uppercase tracking-wider">
                    Valor Total
                  </th>
                </tr>
              </thead>
              <tbody className="bg-white divide-y divide-gray-200">
                {clientes_mais_ativos.slice(0, 5).map((cliente, index) => (
                  <tr key={cliente.cliente_id} className={index % 2 === 0 ?
'bg-white' : 'bg-gray-50'}>
                    <td className="px-6 py-4 whitespace-nowrap">
                      <div className="text-sm font-medium text-gray-900">
{cliente.nome}</div>
```

```
                      <div className="text-sm text-gray-500">{cliente.email}
</div>
                    </td>
                    <td className="px-6 py-4 whitespace-nowrap text-sm text-
gray-900">
                      {cliente.total_compras}
                    </td>
                    <td className="px-6 py-4 whitespace-nowrap text-sm text-
gray-900">
                      R$ {cliente.valor_total.toFixed(2)}
                    </td>
                  </tr>
                ))}
              </tbody>
            </table>
          </div>
        </div>
      </div>

      {/* Análise Cliente-Produto */}
      <div className="bg-white p-6 rounded-lg shadow-md border border-gray-
200">
        <h3 className="text-lg font-semibold text-gray-900 mb-4">Análise por
Tipo de Cliente</h3>
        <div className="grid grid-cols-1 md:grid-cols-2 gap-6">

{Object.entries(analise_cliente_produto.por_tipo_cliente).map(([tipo,
produtos]) => (
            <div key={tipo} className="border border-gray-200 rounded-lg p-
4">
              <h4 className="font-medium text-gray-900 mb-3 capitalize">
                {tipo === 'fisica' ? 'Pessoa Física' : 'Pessoa Jurídica'}
              </h4>
              <div className="space-y-2">
                {produtos.slice(0, 3).map((produto, index) => (
                  <div key={index} className="flex justify-between items-
center">
                    <div>
                      <div className="text-sm font-medium text-gray-900">
{produto.produto}</div>
                      <div className="text-xs text-gray-500">
{produto.categoria}</div>
                    </div>
                    <div className="text-right">
                      <div className="text-sm font-medium text-gray-900">
                        {produto.quantidade} un.
                      </div>
                      <div className="text-xs text-gray-500">
```

```
                      R$ {produto.receita.toFixed(2)}
                    </div>
                  </div>
                </div>
              ))}
            </div>
          </div>
        ))}
      </div>
    </div>
  );
};
```

## Cenário 3: Busca Unificada e Recomendações

Este cenário demonstra como implementar uma funcionalidade de busca que atravessa ambos os módulos e oferece recomendações inteligentes.

## Backend - Sistema de Busca e Recomendações

Python

```
# search_service.py - Serviço de busca unificada e recomendações
from typing import List, Dict, Optional
from sqlalchemy.orm import Session
from sqlalchemy import text, or_, and_
import re

class UnifiedSearchService:
    def __init__(self, db_session: Session):
        self.db = db_session

    def search_unified(self, termo: str, filtros: Optional[Dict] = None,
limit: int = 20) -> Dict:
        """Busca unificada em produtos e clientes."""
        termo_limpo = self._clean_search_term(termo)

        resultados = {
            "termo": termo,
            "produtos": self._search_produtos(termo_limpo, filtros, limit),
            "clientes": self._search_clientes(termo_limpo, filtros, limit),
            "vendas_relacionadas":
self._search_vendas_relacionadas(termo_limpo, limit)
        }

        # Adicionar sugestões baseadas nos resultados
```

```
        resultados["sugestoes"] = self._generate_suggestions(resultados)

        return resultados

    def _clean_search_term(self, termo: str) -> str:
        """Limpa e normaliza termo de busca."""
        # Remove caracteres especiais, mantém apenas letras, números e
espaços
        termo_limpo = re.sub(r'[^\w\s]', '', termo)
        return termo_limpo.strip().lower()

    def _search_produtos(self, termo: str, filtros: Optional[Dict], limit:
int) -> List[Dict]:
        """Busca produtos por termo."""
        query = text("""
        SELECT DISTINCT
            p.id,
            p.nome,
            p.descricao,
            p.sku,
            p.codigo_barras,
            p.preco_venda,
            p.status,
            c.nome as categoria_nome,
            m.nome as marca_nome,
            -- Calcular relevância baseada em vendas
            COALESCE(vendas_stats.total_vendido, 0) as popularidade
        FROM produtos.produtos p
        LEFT JOIN produtos.categorias c ON p.categoria_id = c.id
        LEFT JOIN produtos.marcas m ON p.marca_id = m.id
        LEFT JOIN (
            SELECT
                vi.produto_id,
                SUM(vi.quantidade) as total_vendido
            FROM integracao.vendas_itens vi
            JOIN integracao.vendas v ON vi.venda_id = v.id
            WHERE v.data_venda >= CURRENT_DATE - INTERVAL '90 days'
            GROUP BY vi.produto_id
        ) vendas_stats ON p.id = vendas_stats.produto_id
        WHERE (
            LOWER(p.nome) LIKE :termo OR
            LOWER(p.descricao) LIKE :termo OR
            LOWER(p.sku) LIKE :termo OR
            p.codigo_barras LIKE :termo_exato OR
            LOWER(c.nome) LIKE :termo OR
            LOWER(m.nome) LIKE :termo
        )
        AND (:status IS NULL OR p.status = :status)
```

```
        AND (:categoria_id IS NULL OR p.categoria_id = :categoria_id)
        ORDER BY
            -- Priorizar matches exatos no nome
            CASE WHEN LOWER(p.nome) = :termo_exato THEN 1 ELSE 2 END,
            -- Depois por popularidade
            vendas_stats.total_vendido DESC NULLS LAST,
            -- Por fim, ordem alfabética
            p.nome
        LIMIT :limit
        """)

        params = {
            "termo": f"%{termo}%",
            "termo_exato": termo,
            "status": filtros.get("status") if filtros else None,
            "categoria_id": filtros.get("categoria_id") if filtros else None,
            "limit": limit
        }

        results = self.db.execute(query, params).fetchall()

        return [
            {
                "id": row.id,
                "tipo": "produto",
                "nome": row.nome,
                "descricao": row.descricao,
                "sku": row.sku,
                "codigo_barras": row.codigo_barras,
                "preco_venda": float(row.preco_venda) / 100,
                "status": row.status,
                "categoria": row.categoria_nome,
                "marca": row.marca_nome,
                "popularidade": row.popularidade or 0
            }
            for row in results
        ]

    def _search_clientes(self, termo: str, filtros: Optional[Dict], limit:
int) -> List[Dict]:
        """Busca clientes por termo."""
        query = text("""
        SELECT DISTINCT
            c.id,
            c.nome,
            c.email,
            c.documento,
            c.tipo_pessoa,
```

```
            c.status,
            c.data_nascimento,
            -- Calcular atividade baseada em compras recentes
            COALESCE(compras_stats.total_compras, 0) as atividade_recente,
            COALESCE(compras_stats.valor_total, 0) as valor_total_compras
        FROM clientes.clientes c
        LEFT JOIN (
            SELECT
                v.cliente_id,
                COUNT(v.id) as total_compras,
                SUM(v.total_final) as valor_total
            FROM integracao.vendas v
            WHERE v.data_venda >= CURRENT_DATE - INTERVAL '90 days'
            GROUP BY v.cliente_id
        ) compras_stats ON c.id = compras_stats.cliente_id
        WHERE (
            LOWER(c.nome) LIKE :termo OR
            LOWER(c.email) LIKE :termo OR
            c.documento LIKE :termo_exato
        )
        AND (:status IS NULL OR c.status = :status)
        AND (:tipo_pessoa IS NULL OR c.tipo_pessoa = :tipo_pessoa)
        ORDER BY
            -- Priorizar matches exatos no nome
            CASE WHEN LOWER(c.nome) = :termo_exato THEN 1 ELSE 2 END,
            -- Depois por atividade recente
            compras_stats.total_compras DESC NULLS LAST,
            -- Por fim, ordem alfabética
            c.nome
        LIMIT :limit
        """)

        params = {
            "termo": f"%{termo}%",
            "termo_exato": termo,
            "status": filtros.get("status") if filtros else None,
            "tipo_pessoa": filtros.get("tipo_pessoa") if filtros else None,
            "limit": limit
        }

        results = self.db.execute(query, params).fetchall()

        return [
            {
                "id": row.id,
                "tipo": "cliente",
                "nome": row.nome,
                "email": row.email,
```

```
                "documento": row.documento,
                "tipo_pessoa": row.tipo_pessoa,
                "status": row.status,
                "data_nascimento": row.data_nascimento.isoformat() if
row.data_nascimento else None,
                "atividade_recente": row.atividade_recente or 0,
                "valor_total_compras": float(row.valor_total_compras or 0) /
100
            }
            for row in results
        ]

    def _search_vendas_relacionadas(self, termo: str, limit: int) ->
List[Dict]:
        """Busca vendas que podem estar relacionadas ao termo."""
        query = text("""
        SELECT DISTINCT
            v.id as venda_id,
            v.data_venda,
            v.total_final,
            c.nome as cliente_nome,
            STRING_AGG(p.nome, ', ') as produtos_nomes
        FROM integracao.vendas v
        JOIN clientes.clientes c ON v.cliente_id = c.id
        JOIN integracao.vendas_itens vi ON v.id = vi.venda_id
        JOIN produtos.produtos p ON vi.produto_id = p.id
        WHERE (
            LOWER(c.nome) LIKE :termo OR
            LOWER(p.nome) LIKE :termo
        )
        GROUP BY v.id, v.data_venda, v.total_final, c.nome
        ORDER BY v.data_venda DESC
        LIMIT :limit
        """)

        results = self.db.execute(query, {"termo": f"%{termo}%", "limit":
limit}).fetchall()

        return [
            {
                "venda_id": row.venda_id,
                "tipo": "venda",
                "data_venda": row.data_venda.isoformat(),
                "total": float(row.total_final) / 100,
                "cliente_nome": row.cliente_nome,
                "produtos": row.produtos_nomes
            }
            for row in results
```

```
        ]

    def _generate_suggestions(self, resultados: Dict) -> List[str]:
        """Gera sugestões baseadas nos resultados da busca."""
        sugestoes = []

        # Sugestões baseadas em produtos encontrados
        categorias = set()
        marcas = set()
        for produto in resultados["produtos"]:
            if produto["categoria"]:
                categorias.add(produto["categoria"])
            if produto["marca"]:
                marcas.add(produto["marca"])

        for categoria in list(categorias)[:3]:
            sugestoes.append(f"Produtos da categoria {categoria}")

        for marca in list(marcas)[:3]:
            sugestoes.append(f"Produtos da marca {marca}")

        # Sugestões baseadas em clientes encontrados
        if resultados["clientes"]:
            sugestoes.append("Histórico de compras destes clientes")
            sugestoes.append("Produtos recomendados para estes clientes")

        return sugestoes[:5]  # Limitar a 5 sugestões

class RecommendationService:
    def __init__(self, db_session: Session):
        self.db = db_session

    def get_product_recommendations(self, cliente_id: str, limit: int = 10) -
> List[Dict]:
        """Recomenda produtos para um cliente baseado no histórico."""
        # Produtos comprados pelo cliente
        produtos_comprados = self._get_customer_products(cliente_id)

        if not produtos_comprados:
            # Se cliente não tem histórico, recomendar produtos populares
            return self._get_popular_products(limit)

        # Encontrar clientes similares (que compraram produtos similares)
        clientes_similares = self._find_similar_customers(cliente_id,
produtos_comprados)

        # Recomendar produtos que clientes similares compraram
        recomendacoes = self._get_recommendations_from_similar_customers(
```

```
            cliente_id, clientes_similares, limit
        )

        return recomendacoes

    def _get_customer_products(self, cliente_id: str) -> List[str]:
        """Obtém lista de produtos comprados pelo cliente."""
        query = text("""
        SELECT DISTINCT vi.produto_id
        FROM integracao.vendas v
        JOIN integracao.vendas_itens vi ON v.id = vi.venda_id
        WHERE v.cliente_id = :cliente_id
        AND v.data_venda >= CURRENT_DATE - INTERVAL '365 days'
        """)

        results = self.db.execute(query, {"cliente_id":
cliente_id}).fetchall()
        return [row.produto_id for row in results]

    def _find_similar_customers(self, cliente_id: str, produtos_comprados:
List[str]) -> List[str]:
        """Encontra clientes que compraram produtos similares."""
        if not produtos_comprados:
            return []

        query = text("""
        SELECT
            v.cliente_id,
            COUNT(DISTINCT vi.produto_id) as produtos_em_comum
        FROM integracao.vendas v
        JOIN integracao.vendas_itens vi ON v.id = vi.venda_id
        WHERE vi.produto_id = ANY(:produtos_comprados)
        AND v.cliente_id != :cliente_id
        AND v.data_venda >= CURRENT_DATE - INTERVAL '365 days'
        GROUP BY v.cliente_id
        HAVING COUNT(DISTINCT vi.produto_id) >= 2
        ORDER BY produtos_em_comum DESC
        LIMIT 20
        """)

        results = self.db.execute(query, {
            "produtos_comprados": produtos_comprados,
            "cliente_id": cliente_id
        }).fetchall()

        return [row.cliente_id for row in results]

    def _get_recommendations_from_similar_customers(
```

```
        self, cliente_id: str, clientes_similares: List[str], limit: int
    ) -> List[Dict]:
        """Obtém recomendações baseadas em clientes similares."""
        if not clientes_similares:
            return self._get_popular_products(limit)

        query = text("""
        SELECT
            p.id,
            p.nome,
            p.preco_venda,
            c.nome as categoria_nome,
            m.nome as marca_nome,
            COUNT(DISTINCT v.cliente_id) as clientes_que_compraram,
            SUM(vi.quantidade) as total_vendido,
            AVG(vi.preco_unitario) as preco_medio
        FROM integracao.vendas v
        JOIN integracao.vendas_itens vi ON v.id = vi.venda_id
        JOIN produtos.produtos p ON vi.produto_id = p.id
        LEFT JOIN produtos.categorias c ON p.categoria_id = c.id
        LEFT JOIN produtos.marcas m ON p.marca_id = m.id
        WHERE v.cliente_id = ANY(:clientes_similares)
        AND p.id NOT IN (
            SELECT DISTINCT vi2.produto_id
            FROM integracao.vendas v2
            JOIN integracao.vendas_itens vi2 ON v2.id = vi2.venda_id
            WHERE v2.cliente_id = :cliente_id
        )
        AND p.status = 'ativo'
        AND v.data_venda >= CURRENT_DATE - INTERVAL '180 days'
        GROUP BY p.id, p.nome, p.preco_venda, c.nome, m.nome
        ORDER BY clientes_que_compraram DESC, total_vendido DESC
        LIMIT :limit
        """)

        results = self.db.execute(query, {
            "clientes_similares": clientes_similares,
            "cliente_id": cliente_id,
            "limit": limit
        }).fetchall()

        return [
            {
                "produto_id": row.id,
                "nome": row.nome,
                "preco_venda": float(row.preco_venda) / 100,
                "categoria": row.categoria_nome,
                "marca": row.marca_nome,
```

```
                "popularidade": row.clientes_que_compraram,
                "total_vendido": row.total_vendido,
                "motivo_recomendacao": f"Comprado por
{row.clientes_que_compraram} clientes similares"
            }
            for row in results
        ]

    def _get_popular_products(self, limit: int) -> List[Dict]:
        """Obtém produtos populares como fallback."""
        query = text("""
        SELECT
            p.id,
            p.nome,
            p.preco_venda,
            c.nome as categoria_nome,
            m.nome as marca_nome,
            SUM(vi.quantidade) as total_vendido
        FROM produtos.produtos p
        LEFT JOIN produtos.categorias c ON p.categoria_id = c.id
        LEFT JOIN produtos.marcas m ON p.marca_id = m.id
        LEFT JOIN integracao.vendas_itens vi ON p.id = vi.produto_id
        LEFT JOIN integracao.vendas v ON vi.venda_id = v.id
        WHERE p.status = 'ativo'
        AND (v.data_venda IS NULL OR v.data_venda >= CURRENT_DATE - INTERVAL
'90 days')
        GROUP BY p.id, p.nome, p.preco_venda, c.nome, m.nome
        ORDER BY total_vendido DESC NULLS LAST
        LIMIT :limit
        """)

        results = self.db.execute(query, {"limit": limit}).fetchall()

        return [
            {
                "produto_id": row.id,
                "nome": row.nome,
                "preco_venda": float(row.preco_venda) / 100,
                "categoria": row.categoria_nome,
                "marca": row.marca_nome,
                "total_vendido": row.total_vendido or 0,
                "motivo_recomendacao": "Produto popular"
            }
            for row in results
        ]

# Endpoints FastAPI
@router.get("/search")
```

```
async def unified_search(
    q: str,
    status: Optional[str] = None,
    categoria_id: Optional[str] = None,
    tipo_pessoa: Optional[str] = None,
    limit: int = 20,
    db: Session = Depends(get_db)
):
    """Endpoint para busca unificada."""
    filtros = {
        "status": status,
        "categoria_id": categoria_id,
        "tipo_pessoa": tipo_pessoa
    }

    service = UnifiedSearchService(db)
    return service.search_unified(q, filtros, limit)

@router.get("/recommendations/{cliente_id}")
async def get_recommendations(
    cliente_id: str,
    limit: int = 10,
    db: Session = Depends(get_db)
):
    """Endpoint para recomendações de produtos."""
    service = RecommendationService(db)
    return service.get_product_recommendations(cliente_id, limit)
```

## Frontend - Componente de Busca Unificada

TSX

```
// BuscaUnificada.tsx - Componente de busca integrada
import React, { useState, useEffect, useRef } from 'react';
import { useQuery } from '@tanstack/react-query';
import { Search, Filter, User, Package, ShoppingCart, Star } from 'lucide-
react';
import { useDebounce } from '../hooks/useDebounce';

interface ResultadoBusca {
  produtos: any[];
  clientes: any[];
  vendas_relacionadas: any[];
  sugestoes: string[];
}

export const BuscaUnificada: React.FC = () => {
```

```
  const [termo, setTermo] = useState('');
  const [filtros, setFiltros] = useState({
    status: '',
    categoria_id: '',
    tipo_pessoa: ''
  });
  const [mostrarFiltros, setMostrarFiltros] = useState(false);
  const [resultadoSelecionado, setResultadoSelecionado] = useState<any>
(null);

  const termoDebounced = useDebounce(termo, 300);
  const inputRef = useRef<HTMLInputElement>(null);

  // Query para busca unificada
  const { data: resultados, isLoading, error } = useQuery<ResultadoBusca>({
    queryKey: ['busca-unificada', termoDebounced, filtros],
    queryFn: () => buscarUnificado(termoDebounced, filtros),
    enabled: termoDebounced.length >= 2
  });

  // Focar no input ao montar o componente
  useEffect(() => {
    inputRef.current?.focus();
  }, []);

  const handleSugestaoClick = (sugestao: string) => {
    setTermo(sugestao);
  };

  const handleResultadoClick = (resultado: any) => {
    setResultadoSelecionado(resultado);
  };

  const getIconeResultado = (tipo: string) => {
    switch (tipo) {
      case 'produto':
        return <Package className="h-4 w-4 text-blue-600" />;
      case 'cliente':
        return <User className="h-4 w-4 text-green-600" />;
      case 'venda':
        return <ShoppingCart className="h-4 w-4 text-purple-600" />;
      default:
        return <Search className="h-4 w-4 text-gray-600" />;
    }
  };

  const getCorResultado = (tipo: string) => {
    switch (tipo) {
```

```
      case 'produto':
        return 'border-l-blue-500 bg-blue-50';
      case 'cliente':
        return 'border-l-green-500 bg-green-50';
      case 'venda':
        return 'border-l-purple-500 bg-purple-50';
      default:
        return 'border-l-gray-500 bg-gray-50';
    }
  };

  return (
    <div className="max-w-4xl mx-auto p-6">
      {/* Header de Busca */}
      <div className="bg-white rounded-lg shadow-md p-6 mb-6">
        <div className="flex items-center space-x-4">
          <div className="flex-1 relative">
            <Search className="absolute left-3 top-3 h-5 w-5 text-gray-400"
/>
            <input
              ref={inputRef}
              type="text"
              placeholder="Buscar produtos, clientes, vendas..."
              value={termo}
              onChange={(e) => setTermo(e.target.value)}
              className="w-full pl-10 pr-4 py-3 border border-gray-300
rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-blue-500 text-lg"
            />
          </div>

          <button
            onClick={() => setMostrarFiltros(!mostrarFiltros)}
            className={`p-3 rounded-lg border ${
              mostrarFiltros ? 'bg-blue-50 border-blue-300' : 'border-gray-
300'
            } hover:bg-gray-50`}
          >
            <Filter className="h-5 w-5" />
          </button>
        </div>

        {/* Filtros Expandidos */}
        {mostrarFiltros && (
          <div className="mt-4 p-4 bg-gray-50 rounded-lg">
            <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
              <div>
                <label className="block text-sm font-medium text-gray-700 mb-
1">
```

```
                Status
              </label>
              <select
                value={filtros.status}
                onChange={(e) => setFiltros({...filtros, status:
e.target.value})}
                className="w-full px-3 py-2 border border-gray-300 rounded-
md focus:ring-2 focus:ring-blue-500"
              >
                <option value="">Todos</option>
                <option value="ativo">Ativo</option>
                <option value="inativo">Inativo</option>
              </select>
            </div>

            <div>
              <label className="block text-sm font-medium text-gray-700 mb-
1">
                Tipo de Pessoa
              </label>
              <select
                value={filtros.tipo_pessoa}
                onChange={(e) => setFiltros({...filtros, tipo_pessoa:
e.target.value})}
                className="w-full px-3 py-2 border border-gray-300 rounded-
md focus:ring-2 focus:ring-blue-500"
              >
                <option value="">Todos</option>
                <option value="fisica">Pessoa Física</option>
                <option value="juridica">Pessoa Jurídica</option>
              </select>
            </div>

            <div className="flex items-end">
              <button
                onClick={() => setFiltros({ status: '', categoria_id: '',
tipo_pessoa: '' })}
                className="w-full px-4 py-2 text-gray-600 border border-
gray-300 rounded-md hover:bg-gray-50"
              >
                Limpar Filtros
              </button>
            </div>
          </div>
        </div>
      )}
    </div>
```

```
      {/* Sugestões */}
      {resultados && resultados.sugestoes.length > 0 && (
        <div className="bg-white rounded-lg shadow-md p-4 mb-6">
          <h3 className="text-sm font-medium text-gray-700 mb-2">Sugestões:
</h3>
          <div className="flex flex-wrap gap-2">
            {resultados.sugestoes.map((sugestao, index) => (
              <button
                key={index}
                onClick={() => handleSugestaoClick(sugestao)}
                className="px-3 py-1 text-sm bg-blue-100 text-blue-800
rounded-full hover:bg-blue-200"
              >
                {sugestao}
              </button>
            ))}
          </div>
        </div>
      )}

      {/* Loading */}
      {isLoading && (
        <div className="bg-white rounded-lg shadow-md p-8 text-center">
          <div className="animate-spin rounded-full h-8 w-8 border-b-2
border-blue-600 mx-auto"></div>
          <p className="mt-2 text-gray-600">Buscando...</p>
        </div>
      )}

      {/* Erro */}
      {error && (
        <div className="bg-red-50 border border-red-200 rounded-lg p-4">
          <p className="text-red-800">Erro ao realizar busca. Tente
novamente.</p>
        </div>
      )}

      {/* Resultados */}
      {resultados && !isLoading && (
        <div className="space-y-6">
          {/* Produtos */}
          {resultados.produtos.length > 0 && (
            <div className="bg-white rounded-lg shadow-md p-6">
              <h3 className="text-lg font-semibold text-gray-900 mb-4 flex
items-center">
                <Package className="h-5 w-5 mr-2 text-blue-600" />
                Produtos ({resultados.produtos.length})
              </h3>
```

```
              <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3
gap-4">
                {resultados.produtos.map((produto) => (
                  <div
                    key={produto.id}
                    onClick={() => handleResultadoClick(produto)}
                    className={`p-4 border-l-4 rounded-lg cursor-pointer
hover:shadow-md transition-shadow ${getCorResultado('produto')}`}
                  >
                    <div className="flex items-start justify-between">
                      <div className="flex-1">
                        <h4 className="font-medium text-gray-900">
{produto.nome}</h4>
                        <p className="text-sm text-gray-600 mt-1">
{produto.categoria}</p>
                        <p className="text-sm text-gray-500">SKU:
{produto.sku}</p>
                      </div>
                      {getIconeResultado('produto')}
                    </div>
                    <div className="mt-3 flex items-center justify-between">
                      <span className="text-lg font-bold text-green-600">
                        R$ {produto.preco_venda.toFixed(2)}
                      </span>
                      <span className={`px-2 py-1 rounded-full text-xs font-
medium ${
                        produto.status === 'ativo' ? 'bg-green-100 text-
green-800' : 'bg-red-100 text-red-800'
                      }`}>
                        {produto.status}
                      </span>
                    </div>
                    {produto.popularidade > 0 && (
                      <div className="mt-2 flex items-center text-xs text-
gray-500">
                        <Star className="h-3 w-3 mr-1" />
                        {produto.popularidade} vendas recentes
                      </div>
                    )}
                  </div>
                ))}
              </div>
            </div>
          )}

          {/* Clientes */}
          {resultados.clientes.length > 0 && (
            <div className="bg-white rounded-lg shadow-md p-6">
```

```
              <h3 className="text-lg font-semibold text-gray-900 mb-4 flex
items-center">
                <User className="h-5 w-5 mr-2 text-green-600" />
                Clientes ({resultados.clientes.length})
              </h3>
              <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
                {resultados.clientes.map((cliente) => (
                  <div
                    key={cliente.id}
                    onClick={() => handleResultadoClick(cliente)}
                    className={`p-4 border-l-4 rounded-lg cursor-pointer
hover:shadow-md transition-shadow ${getCorResultado('cliente')}`}
                  >
                    <div className="flex items-start justify-between">
                      <div className="flex-1">
                        <h4 className="font-medium text-gray-900">
{cliente.nome}</h4>
                        <p className="text-sm text-gray-600">{cliente.email}
</p>
                        <p className="text-sm text-gray-500">
{cliente.documento}</p>
                      </div>
                      {getIconeResultado('cliente')}
                    </div>
                    <div className="mt-3 flex items-center justify-between">
                      <span className="text-sm text-gray-600 capitalize">
                        {cliente.tipo_pessoa === 'fisica' ? 'Pessoa Física'
: 'Pessoa Jurídica'}
                      </span>
                      <span className={`px-2 py-1 rounded-full text-xs font-
medium ${
                        cliente.status === 'ativo' ? 'bg-green-100 text-
green-800' : 'bg-red-100 text-red-800'
                      }`}>
                        {cliente.status}
                      </span>
                    </div>
                    {cliente.atividade_recente > 0 && (
                      <div className="mt-2 text-xs text-gray-500">
                        {cliente.atividade_recente} compras recentes •
                        R$ {cliente.valor_total_compras.toFixed(2)} total
                      </div>
                    )}
                  </div>
                ))}
              </div>
            </div>
          )}
```

```
          {/* Vendas Relacionadas */}
          {resultados.vendas_relacionadas.length > 0 && (
            <div className="bg-white rounded-lg shadow-md p-6">
              <h3 className="text-lg font-semibold text-gray-900 mb-4 flex
items-center">
                <ShoppingCart className="h-5 w-5 mr-2 text-purple-600" />
                Vendas Relacionadas ({resultados.vendas_relacionadas.length})
              </h3>
              <div className="space-y-3">
                {resultados.vendas_relacionadas.map((venda) => (
                  <div
                    key={venda.venda_id}
                    onClick={() => handleResultadoClick(venda)}
                    className={`p-4 border-l-4 rounded-lg cursor-pointer
hover:shadow-md transition-shadow ${getCorResultado('venda')}`}
                  >
                    <div className="flex items-center justify-between">
                      <div className="flex-1">
                        <div className="flex items-center space-x-2">
                          {getIconeResultado('venda')}
                          <span className="font-medium text-gray-900">
                            Venda #{venda.venda_id.slice(-8)}
                          </span>
                          <span className="text-sm text-gray-500">
                            {new
Date(venda.data_venda).toLocaleDateString('pt-BR')}
                          </span>
                        </div>
                        <p className="text-sm text-gray-600 mt-1">
                          Cliente: {venda.cliente_nome}
                        </p>
                        <p className="text-sm text-gray-500 mt-1">
                          Produtos: {venda.produtos}
                        </p>
                      </div>
                      <div className="text-right">
                        <span className="text-lg font-bold text-green-600">
                          R$ {venda.total.toFixed(2)}
                        </span>
                      </div>
                    </div>
                  </div>
                ))}
              </div>
            </div>
          )}
```

```
          {/* Nenhum resultado */}
          {resultados.produtos.length === 0 &&
           resultados.clientes.length === 0 &&
           resultados.vendas_relacionadas.length === 0 && (
            <div className="bg-white rounded-lg shadow-md p-8 text-center">
              <Search className="h-12 w-12 text-gray-400 mx-auto mb-4" />
              <h3 className="text-lg font-medium text-gray-900 mb-2">
                Nenhum resultado encontrado
              </h3>
              <p className="text-gray-600">
                Tente usar termos diferentes ou remover alguns filtros.
              </p>
            </div>
          )}
        </div>
      )}

      {/* Modal de Detalhes */}
      {resultadoSelecionado && (
        <div className="fixed inset-0 bg-black bg-opacity-50 flex items-
center justify-center z-50">
          <div className="bg-white rounded-lg max-w-2xl w-full mx-4 max-h-
[80vh] overflow-y-auto">
            <div className="p-6">
              <div className="flex items-center justify-between mb-4">
                <h3 className="text-xl font-bold text-gray-900">
                  Detalhes do {resultadoSelecionado.tipo}
                </h3>
                <button
                  onClick={() => setResultadoSelecionado(null)}
                  className="text-gray-400 hover:text-gray-600"
                >
                  ×
                </button>
              </div>

              <div className="space-y-4">
                {resultadoSelecionado.tipo === 'produto' && (
                  <ProdutoDetalhes produto={resultadoSelecionado} />
                )}
                {resultadoSelecionado.tipo === 'cliente' && (
                  <ClienteDetalhes cliente={resultadoSelecionado} />
                )}
                {resultadoSelecionado.tipo === 'venda' && (
                  <VendaDetalhes venda={resultadoSelecionado} />
                )}
              </div>
            </div>
```

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

// Componentes de detalhes
const ProdutoDetalhes: React.FC<{ produto: any }> = ({ produto }) => (
  <div className="space-y-3">
    <div><strong>Nome:</strong> {produto.nome}</div>
    <div><strong>SKU:</strong> {produto.sku}</div>
    <div><strong>Preço:</strong> R$ {produto.preco_venda.toFixed(2)}</div>
    <div><strong>Categoria:</strong> {produto.categoria}</div>
    <div><strong>Marca:</strong> {produto.marca}</div>
    <div><strong>Status:</strong> {produto.status}</div>
    {produto.descricao && <div><strong>Descrição:</strong>
{produto.descricao}</div>}
  </div>
);

const ClienteDetalhes: React.FC<{ cliente: any }> = ({ cliente }) => (
  <div className="space-y-3">
    <div><strong>Nome:</strong> {cliente.nome}</div>
    <div><strong>Email:</strong> {cliente.email}</div>
    <div><strong>Documento:</strong> {cliente.documento}</div>
    <div><strong>Tipo:</strong> {cliente.tipo_pessoa === 'fisica' ? 'Pessoa
Física' : 'Pessoa Jurídica'}</div>
    <div><strong>Status:</strong> {cliente.status}</div>
    <div><strong>Atividade Recente:</strong> {cliente.atividade_recente}
compras</div>
    <div><strong>Valor Total:</strong> R$
{cliente.valor_total_compras.toFixed(2)}</div>
  </div>
);

const VendaDetalhes: React.FC<{ venda: any }> = ({ venda }) => (
  <div className="space-y-3">
    <div><strong>ID da Venda:</strong> {venda.venda_id}</div>
    <div><strong>Data:</strong> {new
Date(venda.data_venda).toLocaleDateString('pt-BR')}</div>
    <div><strong>Cliente:</strong> {venda.cliente_nome}</div>
    <div><strong>Produtos:</strong> {venda.produtos}</div>
    <div><strong>Total:</strong> R$ {venda.total.toFixed(2)}</div>
  </div>
);

// Função auxiliar para busca
```

```
async function buscarUnificado(termo: string, filtros: any):
Promise<ResultadoBusca> {
  const params = new URLSearchParams({
    q: termo,
    ...Object.fromEntries(Object.entries(filtros).filter(([_, v]) => v))
  });

  const response = await fetch(`/api/v1/search?${params}`);
  return response.json();
}

// Hook personalizado para debounce
function useDebounce<T>(value: T, delay: number): T {
  const [debouncedValue, setDebouncedValue] = useState<T>(value);

  useEffect(() => {
    const handler = setTimeout(() => {
      setDebouncedValue(value);
    }, delay);

    return () => {
      clearTimeout(handler);
    };
  }, [value, delay]);

  return debouncedValue;
}
```

Estes exemplos demonstram como implementar integração robusta entre os módulos de Produtos e Clientes, cobrindo cenários críticos como vendas, analytics e busca unificada. A abordagem mantém a independência dos módulos enquanto oferece funcionalidades integradas valiosas para os usuários.

# Implementação Prática e Roadmap

## Fases de Implementação Recomendadas

A implementação da integração entre os módulos de Produtos e Clientes deve seguir uma abordagem incremental que minimize riscos e permita validação contínua das funcionalidades. A estratégia recomendada divide a implementação em quatro fases distintas, cada uma construindo sobre os fundamentos estabelecidos na fase anterior.

A primeira fase, denominada "Integração Básica", foca na implementação dos componentes fundamentais de comunicação entre módulos. Esta fase inclui a criação do schema de integração no banco de dados, implementação das APIs básicas de

comunicação entre módulos, e desenvolvimento dos primeiros endpoints de integração. O objetivo principal é estabelecer a infraestrutura básica que permitirá comunicação confiável entre os módulos de Produtos e Clientes.

Durante esta fase inicial, é essencial implementar o sistema de correlation IDs para rastreamento de operações distribuídas, configurar o cache distribuído Redis para otimização de performance, e estabelecer os padrões de logging e monitoramento que serão utilizados em todas as operações de integração. A validação desta fase envolve testes de conectividade entre módulos, verificação da consistência de dados nas operações básicas, e confirmação de que os sistemas de monitoramento estão capturando métricas relevantes.

A segunda fase, "Funcionalidades de Venda", implementa o processo completo de vendas que representa o core business da integração. Esta fase inclui o desenvolvimento do serviço de vendas integrado utilizando o padrão Saga para transações distribuídas, implementação dos endpoints de validação de cliente e produto, criação do sistema de reservas temporárias de produtos, e desenvolvimento da interface de usuário para o processo de venda.

A complexidade desta fase requer atenção especial ao tratamento de erros e operações de compensação. É fundamental implementar testes abrangentes que cubram cenários de falha, como indisponibilidade temporária de um dos módulos, falhas de rede durante o processo de venda, e situações de concorrência onde múltiplos usuários tentam comprar o mesmo produto simultaneamente. A validação desta fase deve incluir testes de carga para verificar performance sob stress e testes de recuperação para garantir que o sistema se comporta adequadamente em situações de falha.

A terceira fase, "Analytics e Relatórios", implementa funcionalidades avançadas de análise que combinam dados de ambos os módulos. Esta fase inclui o desenvolvimento do serviço de analytics integrado, criação de views materializadas para consultas otimizadas, implementação do dashboard integrado com visualizações em tempo real, e desenvolvimento do sistema de relatórios combinados.

Esta fase requer consideração cuidadosa de performance, especialmente para consultas que envolvem grandes volumes de dados históricos. É recomendado implementar estratégias de cache agressivo para dados analíticos, utilizar indexação otimizada para consultas frequentes, e considerar a implementação de um data warehouse separado para análises históricas complexas. A validação desta fase deve incluir testes de performance para consultas analíticas e verificação da precisão dos cálculos em diferentes cenários de dados.

A quarta e última fase, "Funcionalidades Avançadas", implementa recursos sofisticados como busca unificada, sistema de recomendações, e automações baseadas em eventos. Esta fase inclui o desenvolvimento do serviço de busca unificada com suporte a filtros

complexos, implementação do sistema de recomendações baseado em machine learning, criação de automações para marketing e gestão de estoque, e desenvolvimento de integrações com sistemas externos.

Esta fase representa o maior nível de complexidade e requer expertise em algoritmos de recomendação, processamento de linguagem natural para busca, e sistemas de eventos em tempo real. A validação desta fase deve incluir testes de precisão para o sistema de recomendações, testes de performance para busca em grandes volumes de dados, e verificação da efetividade das automações implementadas.

## Considerações de Performance e Escalabilidade

A integração entre módulos introduz complexidade adicional que pode impactar significativamente a performance do sistema se não for adequadamente planejada. As principais considerações de performance envolvem latência de comunicação entre módulos, throughput de operações integradas, utilização de recursos de banco de dados, e eficiência do cache distribuído.

A latência de comunicação entre módulos pode ser minimizada através de várias estratégias. A implementação de connection pooling para chamadas HTTP reduz o overhead de estabelecimento de conexões. O uso de HTTP/2 ou gRPC pode melhorar significativamente a eficiência da comunicação, especialmente para operações que envolvem múltiplas chamadas sequenciais. A implementação de circuit breakers previne que falhas em um módulo afetem a disponibilidade de outros módulos.

Para otimizar throughput, é essencial implementar processamento assíncrono sempre que possível. Operações que não requerem resposta imediata, como atualizações de histórico de cliente ou recálculos de estatísticas, devem ser processadas através de filas de mensagens. A implementação de batching para operações similares pode reduzir significativamente o número de chamadas entre módulos.

A utilização eficiente do banco de dados requer atenção especial ao design de índices para consultas de integração. Views materializadas devem ser utilizadas para consultas complexas que envolvem joins entre tabelas de diferentes módulos. A implementação de particionamento para tabelas de grande volume, como vendas e histórico de interações, pode melhorar significativamente a performance de consultas analíticas.

O cache distribuído deve ser utilizado estrategicamente para dados frequentemente acessados. Informações básicas de produtos e clientes são candidatos ideais para cache com TTL longo. Resultados de consultas complexas podem ser cacheados temporariamente para melhorar responsividade. A implementação de cache warming para dados críticos garante que informações importantes estejam sempre disponíveis rapidamente.

Para escalabilidade horizontal, é importante que a arquitetura de integração suporte distribuição de carga entre múltiplas instâncias. O uso de load balancers para distribuir chamadas entre instâncias de cada módulo é essencial. A implementação de sharding para dados de grande volume pode ser necessária conforme o sistema cresce. A utilização de CDN para assets estáticos e dados de cache pode reduzir significativamente a carga nos servidores principais.

## Segurança e Controle de Acesso

A integração entre módulos requer implementação de controles de segurança robustos que protejam dados sensíveis e previnam acesso não autorizado. As principais considerações de segurança incluem autenticação entre módulos, autorização granular para operações de integração, criptografia de dados em trânsito e em repouso, e auditoria completa de operações sensíveis.

A autenticação entre módulos deve utilizar tokens JWT com assinatura digital para garantir integridade e autenticidade das chamadas. Cada módulo deve ter credenciais únicas e rotação automática de tokens deve ser implementada para reduzir riscos de comprometimento. A implementação de mutual TLS para comunicação entre módulos adiciona uma camada extra de segurança.

A autorização deve ser implementada em múltiplas camadas. Cada endpoint de integração deve verificar se o módulo solicitante tem permissão para executar a operação específica. Controles de acesso baseados em roles devem ser implementados para diferentes tipos de operações. A implementação de rate limiting previne abuso e ataques de negação de serviço.

A criptografia de dados em trânsito deve utilizar TLS 1.3 ou superior para todas as comunicações entre módulos. Dados sensíveis como informações pessoais de clientes e detalhes financeiros devem ser criptografados em repouso utilizando algoritmos aprovados como AES-256. A implementação de key management adequado garante que chaves de criptografia sejam protegidas e rotacionadas regularmente.

A auditoria deve capturar todas as operações de integração, incluindo tentativas de acesso negadas, modificações de dados sensíveis, e operações administrativas. Logs de auditoria devem ser imutáveis e armazenados em sistema separado para prevenir tampering. A implementação de alertas em tempo real para atividades suspeitas permite resposta rápida a potenciais incidentes de segurança.

## Monitoramento e Observabilidade

Um sistema de monitoramento abrangente é essencial para manter a saúde e performance da integração entre módulos. O monitoramento deve cobrir métricas de aplicação,

infraestrutura, negócio, e experiência do usuário, fornecendo visibilidade completa sobre o comportamento do sistema integrado.

As métricas de aplicação incluem latência de chamadas entre módulos, taxa de sucesso de operações de integração, throughput de transações distribuídas, e utilização de recursos como CPU e memória. Estas métricas devem ser coletadas em tempo real e agregadas em dashboards que permitam identificação rápida de problemas de performance.

As métricas de infraestrutura cobrem saúde dos servidores, utilização de rede, performance do banco de dados, e status do cache distribuído. Alertas devem ser configurados para condições que podem impactar a disponibilidade do sistema, como alta utilização de CPU, espaço em disco baixo, ou falhas de conectividade de rede.

As métricas de negócio incluem volume de vendas processadas, tempo médio para completar uma venda, taxa de conversão de leads, e satisfação do cliente. Estas métricas fornecem insights sobre o impacto da integração nos resultados do negócio e ajudam a identificar oportunidades de otimização.

A experiência do usuário deve ser monitorada através de métricas como tempo de carregamento de páginas, taxa de erro em operações do usuário, e tempo para completar workflows críticos como processo de venda. A implementação de Real User Monitoring (RUM) fornece insights sobre a experiência real dos usuários em diferentes condições de rede e dispositivos.

A implementação de distributed tracing permite rastrear operações que atravessam múltiplos módulos, facilitando debugging e análise de performance. Correlation IDs devem ser utilizados consistentemente para conectar logs relacionados à mesma operação. A implementação de structured logging facilita análise automatizada e criação de alertas baseados em padrões específicos.

## Testes e Validação

A complexidade da integração entre módulos requer estratégia abrangente de testes que cubra diferentes níveis e tipos de validação. A estratégia de testes deve incluir testes unitários para componentes individuais, testes de integração para comunicação entre módulos, testes de sistema para workflows completos, e testes de aceitação para validação de requisitos de negócio.

Os testes unitários devem cobrir todas as funções de integração, incluindo serialização/deserialização de dados, validação de entrada, e lógica de negócio específica. Mock objects devem ser utilizados para simular dependências externas, permitindo testes isolados e determinísticos. A cobertura de código deve ser mantida acima de 80% para componentes críticos de integração.

Os testes de integração devem validar comunicação real entre módulos, incluindo cenários de sucesso e falha. Estes testes devem utilizar ambientes que simulem condições de produção, incluindo latência de rede e limitações de recursos. A implementação de contract testing garante que mudanças em um módulo não quebrem a compatibilidade com outros módulos.

Os testes de sistema devem validar workflows completos que envolvem múltiplos módulos, como processo de venda do início ao fim. Estes testes devem incluir cenários de falha e recuperação, validando que o sistema se comporta adequadamente em situações adversas. A implementação de chaos engineering pode ajudar a identificar pontos fracos na arquitetura de integração.

Os testes de performance devem validar que o sistema atende aos requisitos de latência e throughput sob diferentes cargas. Testes de carga devem simular picos de tráfego esperados, enquanto testes de stress devem identificar os limites do sistema. A implementação de testes de performance automatizados permite detecção precoce de regressões de performance.

Os testes de segurança devem validar que controles de acesso funcionam adequadamente e que dados sensíveis estão protegidos. Testes de penetração devem ser realizados regularmente para identificar vulnerabilidades. A implementação de testes automatizados de segurança no pipeline de CI/CD permite detecção precoce de problemas de segurança.

# Conclusões e Recomendações

## Benefícios da Integração

A integração entre os módulos de Produtos e Clientes oferece benefícios significativos que justificam o investimento em desenvolvimento e a complexidade adicional introduzida. Os benefícios podem ser categorizados em operacionais, estratégicos, e técnicos, cada um contribuindo para o sucesso geral do sistema de gestão de lojas.

Os benefícios operacionais incluem melhoria significativa na eficiência de processos de negócio. O processo de venda integrado elimina a necessidade de entrada manual de dados em múltiplos sistemas, reduzindo erros e tempo de processamento. A disponibilidade de informações consolidadas permite tomada de decisão mais rápida e informada. A automação de processos como atualização de histórico de cliente e cálculo de recomendações reduz carga de trabalho manual e melhora consistência.

Os benefícios estratégicos incluem capacidade aprimorada de análise de negócio e identificação de oportunidades. A combinação de dados de produtos e clientes permite insights que não seriam possíveis com módulos isolados, como identificação de produtos com maior potencial de venda para segmentos específicos de clientes. O sistema de

recomendações pode aumentar significativamente o valor médio de vendas e satisfação do cliente. A capacidade de rastrear jornada completa do cliente fornece base para estratégias de marketing mais efetivas.

Os benefícios técnicos incluem arquitetura mais robusta e escalável. A implementação de padrões de integração estabelece fundação sólida para adição de novos módulos no futuro. O sistema de monitoramento e observabilidade implementado para integração beneficia todo o sistema. A padronização de APIs e protocolos de comunicação facilita manutenção e evolução do sistema.

## Riscos e Mitigações

A integração entre módulos também introduz riscos que devem ser cuidadosamente gerenciados. Os principais riscos incluem aumento de complexidade, dependências entre módulos, potencial impacto de falhas, e desafios de manutenção.

O aumento de complexidade pode tornar o sistema mais difícil de entender e manter. Este risco pode ser mitigado através de documentação abrangente, implementação de testes automatizados, e treinamento adequado da equipe de desenvolvimento. A utilização de padrões de design bem estabelecidos e ferramentas de monitoramento adequadas também ajuda a gerenciar complexidade.

As dependências entre módulos podem criar pontos únicos de falha e dificultar evolução independente. Este risco pode ser mitigado através de implementação de circuit breakers, degradação graciosa de funcionalidades, e versionamento cuidadoso de APIs. A manutenção de interfaces estáveis e implementação de testes de compatibilidade ajudam a prevenir quebras acidentais.

O potencial impacto de falhas pode ser maior em sistema integrado, onde problema em um módulo pode afetar outros. Este risco pode ser mitigado através de implementação de isolamento de falhas, monitoramento proativo, e planos de recuperação bem definidos. A implementação de redundância para componentes críticos e backup automatizado de dados reduz impacto de falhas.

Os desafios de manutenção podem aumentar devido à necessidade de coordenar mudanças entre múltiplos módulos. Este risco pode ser mitigado através de implementação de pipeline de CI/CD robusto, testes automatizados abrangentes, e processos de release coordenados. A utilização de feature flags permite deployment seguro de novas funcionalidades.

## Próximos Passos

Para implementar com sucesso a integração entre os módulos de Produtos e Clientes, é recomendado seguir um roadmap estruturado que priorize funcionalidades de maior valor e minimize riscos.

O primeiro passo deve ser a preparação da infraestrutura básica, incluindo configuração do ambiente de desenvolvimento integrado, implementação do schema de banco de dados de integração, e configuração das ferramentas de monitoramento e logging. Esta preparação estabelece a fundação necessária para todas as atividades subsequentes.

O segundo passo deve ser a implementação da integração básica, focando na comunicação entre módulos e operações fundamentais. Isto inclui desenvolvimento dos primeiros endpoints de integração, implementação do sistema de correlation IDs, e configuração do cache distribuído. A validação desta etapa através de testes automatizados é crucial antes de prosseguir.

O terceiro passo deve ser a implementação do processo de venda integrado, que representa a funcionalidade de maior valor para o negócio. Isto inclui desenvolvimento do serviço de vendas utilizando padrão Saga, implementação da interface de usuário para vendas, e criação de testes abrangentes para cenários de sucesso e falha.

O quarto passo deve ser a implementação de funcionalidades de analytics e relatórios, que fornecem insights valiosos para gestão do negócio. Isto inclui desenvolvimento do serviço de analytics, criação do dashboard integrado, e implementação de relatórios automatizados.

O quinto e último passo deve ser a implementação de funcionalidades avançadas como busca unificada e sistema de recomendações. Estas funcionalidades, embora valiosas, são menos críticas para operação básica do sistema e podem ser implementadas após validação das funcionalidades fundamentais.

Durante todo o processo, é essencial manter foco na qualidade através de testes automatizados, revisões de código, e monitoramento contínuo. A documentação deve ser atualizada continuamente para refletir mudanças na arquitetura e funcionalidades. O feedback dos usuários deve ser coletado e incorporado para garantir que a integração atende às necessidades reais do negócio.

A implementação bem-sucedida desta integração estabelecerá uma base sólida para futuras expansões do sistema, incluindo adição de novos módulos como gestão de estoque, vendas online, e integrações com sistemas externos. O investimento em arquitetura de integração robusta pagará dividendos significativos conforme o sistema evolui e cresce.

---

# Referências e Recursos Adicionais

## Documentação Técnica

Para implementação bem-sucedida da integração entre módulos, é essencial consultar documentação técnica atualizada das tecnologias utilizadas. A documentação oficial do FastAPI [1] fornece guias abrangentes sobre implementação de APIs RESTful, incluindo melhores práticas para versionamento, autenticação, e documentação automática. A documentação do SQLAlchemy [2] oferece insights detalhados sobre modelagem de dados, otimização de consultas, e implementação de relacionamentos complexos entre tabelas.

A documentação do React [3] e do React Query [4] fornece orientações sobre desenvolvimento de interfaces de usuário modernas e gerenciamento eficiente de estado em aplicações frontend. Para implementação de cache distribuído, a documentação do Redis [5] oferece guias sobre configuração, otimização de performance, e padrões de uso recomendados.

## Padrões de Arquitetura

A implementação de integração entre módulos se beneficia significativamente da aplicação de padrões de arquitetura bem estabelecidos. O padrão Saga [6] é fundamental para implementação de transações distribuídas que mantêm consistência entre múltiplos serviços. A documentação sobre microservices patterns [7] fornece orientações sobre decomposição de sistemas monolíticos, comunicação entre serviços, e gerenciamento de dados distribuídos.

O padrão API Gateway [8] é essencial para gerenciamento centralizado de chamadas entre módulos, fornecendo funcionalidades como roteamento, autenticação, e rate limiting. A implementação de Circuit Breaker [9] é crucial para manter resiliência em sistemas distribuídos, prevenindo cascata de falhas entre módulos dependentes.

## Ferramentas de Desenvolvimento

O sucesso da implementação depende da utilização de ferramentas adequadas para desenvolvimento, teste, e monitoramento. Para desenvolvimento de APIs, ferramentas como Postman [10] ou Insomnia facilitam teste e documentação de endpoints. Para testes automatizados, pytest [11] para backend e Jest [12] para frontend fornecem frameworks robustos com amplo ecossistema de plugins.

Para monitoramento e observabilidade, ferramentas como Prometheus [13] para coleta de métricas, Grafana [14] para visualização, e Jaeger [15] para distributed tracing fornecem visibilidade completa sobre comportamento do sistema integrado. Para gerenciamento de logs, o stack ELK (Elasticsearch, Logstash, Kibana) [16] oferece capacidades avançadas de agregação, busca, e análise.

## Recursos de Aprendizado

Para equipes que implementarão a integração, é recomendado investir em treinamento e recursos de aprendizado. Cursos online sobre arquitetura de microservices [17] fornecem fundamentos teóricos e práticos para design de sistemas distribuídos. Workshops sobre DevOps e CI/CD [18] são essenciais para implementação de pipelines de deployment automatizados.

Livros como "Building Microservices" de Sam Newman [19] e "Designing Data-Intensive Applications" de Martin Kleppmann [20] oferecem insights profundos sobre design de sistemas escaláveis e gerenciamento de dados distribuídos. Conferências e meetups locais sobre tecnologias específicas fornecem oportunidades de networking e aprendizado contínuo.

---

**Referências:**


[1] FastAPI Documentation - https://fastapi.tiangolo.com/

[2] SQLAlchemy Documentation - https://docs.sqlalchemy.org/

[3] React Documentation - https://react.dev/

[4] TanStack Query Documentation - https://tanstack.com/query/latest

[5] Redis Documentation - https://redis.io/documentation

[6] Saga Pattern - https://microservices.io/patterns/data/saga.html

[7] Microservices Patterns - https://microservices.io/patterns/

[8] API Gateway Pattern - https://microservices.io/patterns/apigateway.html

[9] Circuit Breaker Pattern - https://martinfowler.com/bliki/CircuitBreaker.html

[10] Postman - https://www.postman.com/

[11] Pytest Documentation - https://docs.pytest.org/

[12] Jest Documentation - https://jestjs.io/

[13] Prometheus - https://prometheus.io/

[14] Grafana - https://grafana.com/

[15] Jaeger Tracing - https://www.jaegertracing.io/

[16] Elastic Stack - https://www.elastic.co/elastic-stack/

[17] Microservices Architecture Course - https://www.coursera.org/learn/microservices

[18] DevOps Foundation - https://www.devopsinstitute.com/

[19] Building Microservices - https://www.oreilly.com/library/view/building-microservices/9781491950340/

[20] Designing Data-Intensive Applications - https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/