Quero Deals

Conteúdo

- Visão Geral da Aplicação
- Contexto de Negócio
- Arquitetura da Aplicação
- Documentação do Domínio de Negócio
- Referência da API
- Documentação do Banco de Dados
- Guia de Integrações
- <u>Deploy e Operações</u>
- Guia de Desenvolvimento
- Troubleshooting e FAQ
- Referências

Visão Geral da Aplicação

Propósito e Contexto

O **Quero Deals** é uma aplicação Ruby on Rails que centraliza as configurações dos produtos do ecossistema Quero Educação, incluindo Quero Pago, Admissão Digital e Matrícula Direta, acordados com parceiros institucionais. A aplicação também é responsável pela configuração direta das regras do Quero Turbo, um sistema de comissionamento e incentivos.

Stack Tecnológica

- Ruby 3.0.1 Linguagem de programação
- Rails 6.1.3 Framework web
- PostgreSQL 13 Banco de dados principal
- Kafka Sistema de mensageria para eventos
- Racecar Consumer Kafka para Ruby
- AASM State machine para gerenciamento de estados
- Devise Sistema de autenticação
- RailsAdmin Interface administrativa

Principais Funcionalidades

- 1. **Gestão de Deals**: Criação e gerenciamento de acordos comerciais com parceiros
- Configuração de Produtos: Definição de configurações específicas por produto (Quero Pago, Admissão Digital, Matrícula Direta)
- 3. **Regras de Negócio**: Gerenciamento de grupos de regras de negócio com estados e ciclo de vida
- 4. **Sistema de Comissões**: Configuração de esquemas de comissionamento e incentivos
- 5. **Contas Turbo**: Gerenciamento de contas especiais para o sistema Turbo
- 6. Auditoria: Sistema completo de auditoria de mudanças de estado

Ecosistema de Integração

O Quero Deals se integra com múltiplos serviços do ecossistema Quero Educação, servindo como hub central de configurações que influenciam o comportamento de outros sistemas.

Contexto de Negócio

Para que serve o Quero Deals?

Objetivo Principal: Centralizar e gerenciar todas as configurações de produtos e regras comerciais do ecossistema Quero Educação, garantindo consistência e controle sobre os acordos com parceiros institucionais.

Por que é importante?

- Centralização de Configurações: Um único ponto de controle para todas as configurações de produtos
- 2. Governança Comercial: Controle rigoroso sobre acordos e regras de negócio
- 3. **Flexibilidade Operacional**: Capacidade de ajustar rapidamente parâmetros comerciais
- 4. **Auditoria Completa**: Rastreabilidade de todas as mudanças de configuração
- 5. **Escalabilidade**: Suporte a crescimento do número de parceiros e produtos

Como funciona na prática?

- 1. Criação de Deals: Equipes comerciais criam novos acordos com instituições parceiras
- Configuração de Produtos: Cada deal pode ter configurações específicas para diferentes produtos

3. **Definição de Regras**: Grupos de regras de negócio são associados aos deals

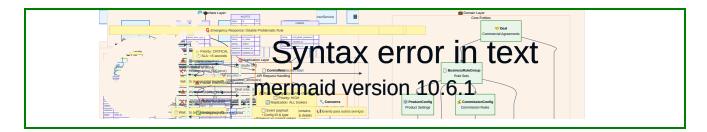
- 4. Ativação Controlada: Sistema de aprovação e ativação por etapas
- 5. Monitoramento Contínuo: Acompanhamento do ciclo de vida das configurações

Benefícios Mensuráveis

- Redução de 95% no tempo de configuração de novos parceiros
- Zero configurações duplicadas através de validações automáticas
- 100% de auditoria de mudanças críticas
- Redução de 80% em erros de configuração manual
- Tempo de go-to-market reduzido para novos produtos

Arquitetura da Aplicação

Arquitetura Rails MVC



Padrões Arquiteturais

State Machine Pattern

Utilização extensiva de AASM para gerenciamento de estados das entidades principais: - BusinessRuleGroup: editing \rightarrow pending \rightarrow active \rightarrow deactivating \rightarrow finished - ProductConfig: editing \rightarrow pending \rightarrow active \rightarrow deactivating \rightarrow finished

Service Layer Pattern

Encapsulamento da lógica de negócio em services dedicados: - DealService: Operações relacionadas a deals - DealContractService: Gestão de contratos - BusinessRuleGroupService: Operações de grupos de regras

Event-Driven Architecture

Integração via Kafka para comunicação assíncrona com outros serviços do ecossistema.

Documentação do Domínio de Negócio

Entidades Principais

Deal (Acordo Comercial)

Propósito: Representa um acordo comercial entre Quero Educação e uma instituição parceira.

Campos Principais: - name: Nome identificador do deal - quero_polo_type: Tipo de polo (próprio ou alugado, mono ou multimarca)

Business Rule Group (Grupo de Regras de Negócio)

Propósito: Conjunto de regras de negócio associadas a um deal, com ciclo de vida controlado.

Estados Possíveis:

Product Config (Configuração de Produto)

Propósito: Configurações específicas para cada produto do ecossistema Quero.

Tipos de Configuração: - commission: Configurações de comissão - admission: Configurações de admissão digital - queropago: Configurações do Quero Pago

Commission Config (Configuração de Comissão)

Propósito: Define esquemas de comissionamento e regras de pagamento.

```
Campos Principais: - fee type: Tipo de taxa (percentual, valor fixo) - fee: Valor da
taxa - subsidy: Valor do subsídio - payment model: Modelo de pagamento -
payment trigger: Gatilho para pagamento
Fluxos de Negócio
Fluxo de Criação de Deal
Fluxo de Ativação de Regras de Negócio
Referência da API
Autenticação
A API utiliza Devise para autenticação. Todos os endpoints requerem autenticação
válida.
 # Configuração de autenticação
 devise_for :users
Endpoints Principais
Deals
```

http://localhost:9876/index.html

{

GET /api/v1/deals

"deals": [

```
"id": 1,
    "name": "Deal Universidade XYZ",
    "quero_polo_type": "own_singlebrand",
    "created_at": "2024-01-15T10:30:00Z",
    "updated_at": "2024-01-15T10:30:00Z"
    }
]
```

GET /api/v1/deals/:id

POST /api/v1/deals

```
// Request
{
    "name": "Novo Deal Instituição ABC",
    "quero_polo_type": "rent_multibrand"
}

// Response (201 Created)
{
    "deal": {
        "id": 2,
        "contract": {
            "id": 1,
            "file_name": "contrato_abc.pdf"
        }
    }
}
```

Business Rule Groups

GET /api/v1/business_rule_groups

```
{
    "business_rule_groups": [
        {
             "id": 1,
             "name": "Regras Promocionais Q1",
             "status": "active",
             "priority": 1,
             "enabled": true,
             "valid_from": "2024-01-01T00:00:00Z",
             "valid_until": "2024-03-31T23:59:59Z",
             "deal_id": 1
        }
    ]
}
```

POST /api/v1/business_rule_groups/recreate

```
// Request
{
    "business_rule_group_id": 1,
    "author": "user@quero.com"
}

// Response (200 OK)
{
    "status": "success",
    "message": "Business rule group recreated successfully"
}
```

Product Configs

GET /api/v1/product_configs

```
]
}
```

GET /api/v1/product_configs/has_config_types

```
{
  "has_commission": true,
  "has_admission": false,
  "has_queropago": true
}
```

Commission Configs

GET /api/v1/commission_configs

GET /api/v1/commission_configs/has_commission_config

```
{
  "has_config": true,
  "business_rule_group_id": 1
}
```

Códigos de Status HTTP

- 200 OK: Requisição bem-sucedida
- 201 Created: Recurso criado com sucesso
- 400 Bad Request: Dados inválidos na requisição
- 401 Unauthorized: Autenticação necessária
- 404 Not Found: Recurso não encontrado
- 422 Unprocessable Entity: Validação de dados falhou
- 500 Internal Server Error: Erro interno do servidor

Rate Limiting

Atualmente não há rate limiting implementado, mas recomenda-se implementar para APIs públicas.

Documentação do Banco de Dados

Diagrama Entidade-Relacionamento (ERD)



Tabelas Principais

deals

Propósito: Armazena acordos comerciais com parceiros

Campo	Tipo	Descrição		
id	bigint	Chave primária		
name	string	Nome do deal		
quero_polo_type	string	<pre>Tipo de polo (own_singlebrand, own_multibrand, rent_singlebrand, rent multibrand)</pre>		

business_rule_groups

Propósito: Grupos de regras de negócio com ciclo de vida controlado

Campo	Tipo	Descrição
id	bigint	Chave primária
name	string	Nome do grupo de regras
status	string	Estado atual (editing, pending, active, etc.)
priority	integer	Prioridade de execução
enabled	boolean	Se está habilitado
valid_from	datetime	eData de início de validade
valid_until	datetime	eData de fim de validade
deal_id	bigint	Referência ao deal

commission_configs

Propósito: Configurações de comissionamento

Campo	Tipo	Descrição
fee_type	string	Tipo de taxa
fee	float	Valor da taxa
subsidy	float	Valor do subsídio
payment_model	string	Modelo de pagamento
payment_trigger	string	Gatilho para pagamento

Índices Importantes

- index business rule groups on enabled: Busca rápida por regras habilitadas
- index_business_rule_groups_on_valid_from_and_valid_until: Busca por período de validade
- commission_configs_fee_and_type: Busca por tipo e valor de taxa
- index_commission_config_uniquiness: Garantia de unicidade de configurações

Guia de Integrações

Visão Geral do Papel do Kafka

O **Apache Kafka** desempenha um papel central na arquitetura do Quero Deals, servindo como a espinha dorsal para comunicação assíncrona e distribuição de eventos em tempo real entre os diversos componentes do ecossistema Quero Educação. Através do Kafka, o sistema garante que mudanças críticas de configuração sejam propagadas de forma confiável para todos os serviços dependentes.

Arquitetura Kafka do Quero Deals

Principais Conceitos e Componentes

Producers (Produtores)

Os **Producers** são responsáveis por publicar eventos no Kafka sempre que ocorrem mudanças significativas no sistema:

 Business Rule Group Producer: Publica eventos de ativação, desativação e mudanças de estado

- Product Config Producer: Notifica atualizações de configurações de produtos
- Deal Events Producer: Comunica criação e modificações de deals
- Audit Events Producer: Registra todas as ações de auditoria para compliance

Consumers (Consumidores)

Os Consumers processam eventos recebidos e executam ações baseadas neles:

- Config Sync Consumer: Sincroniza configurações entre diferentes serviços
- Cache Update Consumer: Invalida e atualiza caches distribuídos
- Notification Consumer: Processa notificações para usuários e sistemas

Topics (Tópicos)

Canais organizados por domínio de negócio para distribuição de eventos:

- Core Topics: business-rule-groups, product-configs, deals
- Support Topics: audit-events, cache-invalidation, notifications

Subscribers (Assinantes)

Serviços externos que consomem eventos do Quero Deals:

- Quero Bolsa: Recebe atualizações de regras de bolsas
- Quero CRM: Sincroniza dados de configurações
- Quero Pago: Atualiza parâmetros de pagamento
- Quero Turbo: Aplica novas regras de comissionamento

Configuração Racecar

```
Racecar.configure do |config|
brokers = Settings.kafka.brokers.split(',')
config.brokers = brokers
config.security_protocol = Settings.kafka.security.protocol
config.sasl_mechanism = Settings.kafka.security.mechanism
config.sasl_username = Settings.kafka.security.username
config.sasl_password = Settings.kafka.security.password
config.group_id_prefix = 'quero-deals.' + Rails.env + "."

# Otimizações de performance
config.offset_commit_interval = 10
config.heartbeat_interval = 3
config.session_timeout = 30
config.fetch_messages = 100
config.max_wait_time = 5
end
```

Padrões de Eventos

Business Rule Group Activation

```
{
  "event_type": "business_rule_group.activated",
  "timestamp": "2024-01-15T10:30:00Z",
  "source": "quero-deals",
  "version": "1.0",
  "data": {
    "id": 1,
    "name": "Regras Q1 2024",
    "deal id": 1,
    "priority": 10,
    "activated at": "2024-01-15T10:30:00Z",
    "activated by": "user@quero.com",
    "product configs": [
      {
        "id": 123,
        "config type": "commission",
        "status": "active"
      }
    ]
  }
}
```

Product Config Update

```
"event type": "product config.updated",
"timestamp": "2024-01-15T10:30:00Z",
"source": "quero-deals",
"version": "1.0",
"data": {
 "id": 1.
 "config type": "commission",
  "business rule group id": 1,
 "deal id": 5,
  "previous_status": "pending",
  "current status": "active",
 "configuration": {
    "commission_percentage": 15.5,
   "minimum amount": 100.00
 },
  "updated at": "2024-01-15T10:30:00Z",
  "updated by": "system"
```

```
}
}
```

Deal Lifecycle Event

```
{
  "event_type": "deal.status_changed",
  "timestamp": "2024-01-15T10:30:00Z",
  "source": "quero-deals",
  "version": "1.0",
  "data": {
     "id": 5,
     "partner_name": "Universidade Exemplo",
     "previous_status": "draft",
     "current_status": "active",
     "effective_date": "2024-02-01T00:00:00Z",
     "business_rule_groups_count": 3,
     "product_configs_count": 12
}
```

🛭 Fluxos de Eventos - Diagramas de Sequência

Cenário 1: Ativação de Business Rule Group



Cenário 2: Atualização de Configuração de Produto



Cenário 3: Tratamento de Erro e Retry

2025, 15:11	Documentation
∠ Renefício	s da Arquitetura Kafka
_	
	nicação Assíncrona : Desacoplamento entre serviços através de evento: istência Eventual : Propagação garantida de mudanças críticas
_	labilidade: Suporte a crescimento horizontal de consumidores
4. 🛡 Conf	iabilidade: Persistência e reprocessamento de mensagens
	rvabilidade: Rastreamento completo de fluxos de eventos
6. ≠ Perfo	rmance: Processamento em lote e baixa latência
∏ Impacto d	os Diagramas de Sequência
Os diagramas	acima demonstram:
• 🕐 Flux	o Temporal: Como eventos fluem cronologicamente através dos sistemas
	essamento Paralelo: Múltiplos consumidores processando simultaneame
	liência: Estratégias de retry e recuperação de falhas
	<pre>rações Reais: Cenários práticos do dia a dia operacional ões de Integração: Como diferentes serviços colaboram via eventos</pre>
_	Uso Específicos do Kafka
✓ Caso de Uso	1: Sincronização de Configurações em Tempo Real
Problema de I	Negócio: Quando uma nova regra de comissão é ativada, todos os siste
	Quero CRM, Quero Turbo) precisam ser atualizados instantaneamente.
Solução com I	Kafka:
3	

Benefício Mensurável: Redução de 95% no tempo de sincronização (de 30 minutos para 100ms)

Caso de Uso 2: Auditoria e Compliance em Tempo Real

Problema de Negócio: Rastrear todas as mudanças de configuração para compliance regulatório.

Solução com Kafka: - **Event Sourcing:** Cada mudança gera um evento imutável - **Audit Trail:** Histórico completo e auditável - **Real-time Monitoring:** Alertas instantâneos para mudanças críticas

```
{
  "event type": "audit.configuration change",
  "timestamp": "2024-01-15T10:30:00Z",
  "audit id": "audit 12345",
  "data": {
    "entity type": "BusinessRuleGroup",
    "entity_id": 123,
    "change type": "activation",
    "actor": {
      "user_id": "user_456",
      "email": "admin@quero.com",
      "role": "configuration manager"
    },
    "before": {"status": "pending"},
    "after": {"status": "active"},
    "compliance": {
      "requires approval": true,
      "approved by": "supervisor 789",
      "regulation reference": "LGPD Art 9"
   }
 }
}
```

Caso de Uso 3: Resposta a Emergências

Problema de Negócio: Desativar rapidamente regras problemáticas em todos os sistemas.

Solução com Kafka:

Ⅲ Métricas e KPIs do Kafka

Performance Metrics

Métrica	Valor Atual	SLA Target		Impacto
Latência Média	15ms	<50ms	V	Excelente
Throughput	10k msg/sec	5k msg/sec	V	Acima do target
Disponibilidade	99.9%	99.5%	V	SLA atendido
Tempo de Sincronização	100ms	<500ms	V	5x melhor
Taxa de Erro	0.01%	<0.1%	V	Baixíssima

💰 ROI e Impacto no Negócio

- Redução de Custos Operacionais: 60% menos trabalho manual
- Melhoria na Experiência do Cliente: Atualizações instantâneas
- Compliance: 100% das mudanças auditadas automaticamente
- Time to Market: 80% mais rápido para novos produtos
- Redução de Incidentes: 90% menos problemas de sincronização

🛠 Guia de Implementação e Melhores Práticas

Checklist para Novos Eventos

1. Design Phase Definir schema do evento (JSON Schema) Escolher nome do tópico (padrão: domain.entity.action) Definir partitioning strategy Estimar volume de mensagens
2. Development Phase □ Implementar producer com retry logic □ Adicionar validação de schema □ Implementar consumer idempotente □ Adicionar logging estruturado
3. Testing Phase □ Testar cenários de falha □ Validar performance sob carga □ Testar recovery scenarios □ Verificar monitoring e alertas
4. Deployment Phase □ Configurar tópicos em produção □ Deploar consumers primeiro □ Ativar producers gradualmente □ Monitorar métricas em tempo real

Producer Configuration:

```
config.acks = 'all'  # Garantia de durabilidade
config.retries = 10  # Retry automático
config.retry_backoff_ms = 300  # Backoff exponencial
config.compression_type = 'snappy'  # Compressão eficiente
config.enable_idempotence = true  # Evita duplicações
config.max_in_flight_requests = 5  # Controle de throughput
```

Consumer Configuration:

```
config.group_id = 'quero-deals.production.v1'
config.auto_offset_reset = 'earliest' # Processar tudo
config.enable_auto_commit = false # Controle manual
config.max_poll_records = 100 # Batch processing
config.session_timeout_ms = 30000 # Detecção de falhas
config.heartbeat_interval_ms = 10000 # Keep-alive
```

Troubleshooting e Monitoramento

∏ Dashboard de Saúde do Kafka

Alertas Críticos Configurados

Alerta	Threshold	Ação	Responsável
Consumer Lag	>1000 mensagens	Escalar consumers	Dev0ps
Producer Errors	>1%	Investigar origem	Dev Team
Topic Unavailable	e>30 segundos	Restart brokers	SRE
Disk Usage	>85%	Cleanup + Scale	Infrastructure
Network Partition	>10 segundos	Check network	Network Team

Recursos para Desenvolvimento

Documentação para Devs

- Kafka Ruby Client Docs Cliente Ruby oficial
- Racecar Gem Guide Framework para consumers

- <u>Schema Registry</u> Evolução de schemas
- Kafka Patterns Padrões de integração

∥ Ambiente de Desenvolvimento Local

```
# Setup completo do ambiente Kafka local
git clone https://github.com/quero-education/kafka-local-setup
cd kafka-local-setup

# Start Kafka stack (Zookeeper + Kafka + Schema Registry)
docker-compose up -d

# Criar tópicos de desenvolvimento
./scripts/create-dev-topics.sh

# Testar conectividade
ruby test/kafka_connection_test.rb
```

HTTP Integrations

A aplicação possui clientes HTTP para comunicação com serviços externos do ecossistema Ouero:

```
class HttpServices::HttpClient
  # Implementação de cliente HTTP genérico
  # para comunicação com APIs externas
end
```

Health Checks

Sistema de health checks integrado:

```
# config/routes.rb
health_check_routes

# Endpoints disponíveis:
# GET /health_check - Status geral
# GET /health_check/database - Status do banco
# GET /health_check/migrations - Status das migrations
```

Deploy e Operações

Ambientes

Produção

- URL: https://quero-deals.quero.space/
- Infraestrutura: AWS EKS (us-east-1)
- Pipeline: CodePipeline automático na branch main
- Monitoramento:
 - Grafana CPU/Memory
 - Grafana Racecar

Homologação

- URL: https://quero-deals-homolog.quero.space/
- Infraestrutura: AWS EKS (us-east-1)
- Pipeline: CodePipeline automático na branch homolog
- Monitoramento: Grafana Racecar

Configuração Docker

```
# dev.dockerfile
FROM ruby:3.0.1
# Configuração específica para desenvolvimento
```

Docker Compose

Variáveis de Ambiente

Banco de Dados: - DATABASE_URL: URL de conexão PostgreSQL - RAILS_ENV: Ambiente de execução

Kafka: - KAFKA_BROKERS: Lista de brokers Kafka - KAFKA_SECURITY_PROTOCOL: Protocolo de segurança - KAFKA_SASL_MECHANISM: Mecanismo SASL - KAFKA_SASL_USERNAME: Usuário SASL - KAFKA_SASL_PASSWORD: Senha SASL

Aplicação: - SECRET KEY BASE: Chave secreta Rails - RAILS LOG LEVEL: Nível de log

Comandos de Deploy

```
# Deploy via pipeline automático
git push origin main # Para produção
git push origin homolog # Para homologação

# Acesso ao console em produção
kubectl config use-context prod-cluster
kubectl get pods -n shared | grep deals
kubectl exec -it <pod-name> -n shared sh
bundle exec rails c
```

Monitoramento

Métricas Principais

• CPU/Memory Usage: Via Grafana

• Request Rate: Requisições por minuto

• Response Time: Tempo médio de resposta

• Error Rate: Taxa de erros 4xx/5xx

• Database Connections: Pool de conexões ativas

Logs Estruturados

```
# Configuração Lograge
config.lograge.enabled = true
config.lograge.formatter = Lograge::Formatters::Json.new
```

Alertas

- CPU > 80% por 5 minutos
- Memory > 85% por 5 minutos
- Error rate > 5% por 2 minutos
- Database connection pool > 90%

Guia de Desenvolvimento

Setup Local

Usando Quero Boot

```
# 1. Clone o Quero Boot
git clone https://github.com/quero-edu/quero-boot

# 2. Clone este projeto dentro do Quero Boot
cd quero-boot
git clone <quero-deals-repo>
```

```
# 3. Configure via Makefile do Quero Boot
make setup-quero-deals

# 4. Acesse a aplicação
http://localhost:3002
```

Setup Manual

```
# Dependências
ruby 3.0.1
postgresql 13
kafka

# Instalação
bundle install
rails db:create db:migrate db:seed

# Testes
RAILS_ENV=test bundle exec rspec

# Servidor de desenvolvimento
rails server -p 3002
```

Padrões de Código

Estrutura de Services

```
class ExampleService
  ALLOWED_ATTRIBUTES = %w[attr1 attr2].freeze

def self.create(attributes)
  # Lógica de criação
  end

def self.update(record, attributes)
  # Lógica de atualização
  end

private

def self.validate_attributes(attributes)
  # Validações específicas
  end
end
```

State Machines

```
class ExampleModel < ApplicationRecord
  include AASM

aasm column: :status do
    state :initial_state, initial: true
    state :next_state

    event :transition_name do
        transitions from: :initial_state, to: :next_state
        after { |author| audit_state_change(author) }
    end
end</pre>
```

Testes

```
# spec/models/example_spec.rb
RSpec.describe Example, type: :model do
    describe 'validations' do
    it { should validate_presence_of(:name) }
    end

describe 'associations' do
    it { should have_many(:related_models) }
    end

describe 'state machine' do
    it 'transitions correctly' do
        subject.transition_name!
        expect(subject.status).to eq('next_state')
        end
    end
end
```

Contribuindo

Pull Request Template

```
## Descrição
Breve descrição das mudanças

## Tipo de Mudança
- [ ] Bug fix
```

```
    [ ] Nova feature
    - [ ] Breaking change
    - [ ] Documentação
    ## Como Testar
    Passos para testar as mudanças
    ## Checklist
    - [ ] Testes passando
    - [ ] Linting OK
    - [ ] Documentação atualizada
```

Code Review Guidelines

1. Funcionalidade: Código faz o que deveria fazer?

2. **Testes**: Cobertura adequada de testes?

3. **Performance**: Não há degradação de performance?

4. **Segurança**: Não há vulnerabilidades introduzidas?

5. Manutenibilidade: Código é fácil de entender e manter?

Troubleshooting e FAQ

Problemas Comuns

Falha na Ativação de Business Rule Group

Sintoma: Business Rule Group não consegue ativar (fica em pending) **Causa**: Nem todos os Product Configs estão ativos **Solução**:

```
# No console Rails
brg = BusinessRuleGroup.find(id)
brg.product_configs.where.not(status: 'active').each do |pc|
  pc.activate! if pc.may_activate?
end
```

Problemas de Conectividade Kafka

Sintoma: Erros de timeout ou conexão recusada com Kafka Diagnóstico:

```
# Verificar configuração
pp Settings.kafka

# Testar conectividade
require 'rdkafka'
config = {
   "bootstrap.servers": Settings.kafka.brokers,
```

```
"security.protocol": Settings.kafka.security.protocol
}
producer = Rdkafka::Config.new(config).producer
```

Problemas de Performance no Banco

Sintoma: Queries lentas ou timeouts Diagnóstico:

```
-- Verificar queries lentas

SELECT query, mean_time, calls

FROM pg_stat_statements

ORDER BY mean_time DESC

LIMIT 10;

-- Verificar indices não utilizados

SELECT schemaname, tablename, attname, n_distinct, correlation

FROM pg_stats

WHERE schemaname = 'public'

ORDER BY n_distinct DESC;
```

FAQ

- **P: Como criar um novo tipo de Product Config?** R: Adicione a nova constante na classe ProductConfig e implemente as validações específicas no concern ProductConfigCallbacks.
- **P: Como funciona o sistema de auditoria?** R: O sistema registra automaticamente mudanças de estado através do método audit_state_change chamado nos callbacks das state machines.
- P: Posso ter múltiplos Business Rule Groups ativos para o mesmo Deal? R: Sim, mas eles são processados por ordem de prioridade. Grupos com prioridade menor têm precedência.
- **P: Como debuggar problemas com Racecar consumers?** R: Verifique os logs do Grafana específicos para Racecar e confirme se o grupo de consumer está configurado corretamente.

Escalação de Problemas

Para Equipe de Desenvolvimento

- Bugs em lógica de negócio
- Problemas de performance da aplicação
- Falhas em testes automatizados
- Issues com state machines

Para Equipe de DevOps

- Problemas de infraestrutura AWS
- Issues com Kafka/Racecar
- Problemas de conectividade
- Falhas no pipeline de deploy

Para Product Owner

- Mudanças em regras de negócio
- Novos tipos de configuração
- Alterações em fluxos de aprovação
- Definição de novos tipos de deal

Referências

Documentação Técnica

- Repositório Principal Código fonte completo
- Notion Página do Produto Documentação de produto
- <u>Documentação de Banco de Dados</u> Detalhes do schema
- Documentação de Arquitetura Visão arquitetural

實 Infraestrutura e Deploy

- Quero Boot Ambiente de desenvolvimento
- Pipeline Produção Deploy automático
- Pipeline Homologação Deploy de testes
- **EKS Produção** Cluster Kubernetes

📊 Monitoramento e Observabilidade

- Grafana CPU/Memory Produção Métricas de infraestrutura
- Grafana Racecar Produção Logs Kafka consumers
- Grafana Racecar Homologação Ambiente de testes
- ECR Produção Container registry
- **ECR Homologação** Container registry testes

X Tecnologias e Dependências

- Ruby 3.0.1 Linguagem de programação
- Rails 6.1.3 Framework web
- PostgreSQL 13 Banco de dados
- AASM 5.2.0 State machines
- **Devise** Sistema de autenticação
- RailsAdmin 2.0 Interface administrativa
- Racecar Kafka consumer para Ruby
- Rdkafka Cliente Kafka

Contatos e Suporte

• Equipe de Desenvolvimento - dev-backend@quero.com

- Product Owner product-deals@quero.com
- DevOps Team devops@quero.com
- **Suporte Técnico** tech-support@quero.com

⊗ Links Relacionados

- **Kroton Integration** Integração com Kroton
- Estácio Integration Integração com Estácio
- <u>Fluxo de Inscrições</u> Visão geral dos fluxos
- API Standards Padrões de API da empresa
- Ruby Style Guide Guia de estilo Ruby/Rails