PLANEJAMENTO REVISADO

Sistema de Workflow Kanban

VibeCForms v4.0

Regras de Negócio com IA e Aprendizado por Padrões

Versão:	2.0 - Planejamento Revisado
Data:	25/10/2025
Status:	Em Revisão

PLANEJAMENTO REVISADO: Sistema de Workflow Kanban

VibeCForms v4.0 - Regras de Negócio com IA e Aprendizado por Padrões

Data: 25/10/2025Versão: 2.0 - Planejamento RevisadoStatus: Em Revisão

ÍNDICE

1. CORREÇÕES E ALINHAMENTOS

1.1. Correções Aplicadas

Item	Versão Anterior (PDF)	Versão Corrigida
Conceito de Kanban	Kanban = Regra de Negócio	■ Kanban = 1 Processo
Bloqueio de Transições	Controlled Mode bloqueia	■ NUNCA bloqueia, apenas avisa
Agentes de IA	Seguem fluxo padrão	■ Seguem determinação, cada agente = 1 estado
Sistema de Usuários	Auditoria com usuários	■ Auditoria sem usuários (apenas log)
Estimativas	Horas detalhadas	■ Removido
State Machine	Progressivamente rígida	■ Sempre flexível, só orienta

1.2. Princípios Fundamentais Revisados

Filosofia Central: "Avisar, Não Bloquear"

■■■

■ NUNCA BLOQUEAR ■

■ ■■ Avisar quando transição é anormal ■

■ ■■ Pedir justificativa para auditoria ■

■ ■■ Registrar tudo em log ■

■ ■■ Permitir IA reaprender novos padrões ■

■■■

Definições-Chave:

1. Kanban = 1 Processo Específico
• Exemplo: "Pedidos", "Suporte", "RH-Contratação"
• Cada Kanban define suas colunas (estados válidos)
• Estados podem transitar para QUALQUER outro estado
2. Colunas = Estados do Processo
• Definidas pelo usuário
• Podem ter pré-requisitos (checks)
• Pré-requisitos NÃO bloqueiam, apenas alertam
3. Pré-requisitos = Checks Opcionais
• Validam se algo foi feito
• Sistema avisa se não cumpridos
• Usuário pode ignorar e justificar
4. Agentes de IA = Executores Determinísticos
• Cada agente responsável por 1 estado (coluna)
• Preenche requisitos daquele estado
• Checa e passa para próximo agente determinado
• NÃO seguem "fluxo padrão", seguem script/determinação

2. VISÃO GERAL DO SISTEMA

2.1. Ciclo de Vida de um Processo

1. CRIAÇÃO DO KANBAN (Processo)
■■ Usuário define nome: "Pedidos"
■■ Define colunas: ["Orçamento", "Aprovação", "Entrega", "Concluído"]
■■ Define pré-requisitos opcionais por coluna
■■ Sistema cria estrutura JSON

2. OPERAÇÃO DO KANBAN
■■ Usuário move cards entre colunas livremente
■■ Sistema checa pré-requisitos
■■ Se não cumpridos: AVISA (não bloqueia)
■■ Pede justificativa se transição anormal
■■ Registra tudo em log de auditoria

3. ANÁLISE POR IA
■■ IA monitora movimentações
■■ Detecta padrões comuns (80% segue X→Y→Z)
■■ Identifica transições raras/anormais
■■ Sugere otimizações (estados desnecessários, etc)
■■ NÃO cria State Machine rígida

4. AGENTES DE IA
■■ Cada agente opera em 1 estado específico
■■ Preenche requisitos daquele estado

■■ Chama próximo agente (determinado, não "padrão")

■■ Registra ações em log

2.2. Filosofia "Avisar, Não Bloquear"

Exemplo Prático:

Usuário move card de "Orçamento" → "Entrega" (pulando "Aprovação")

Sistema NÃO faz:

- Bloquear movimento
- Exigir aprovação de gestor
- Reverter ação

Sistema FAZ:

1. Move o card normalmente
2. Exibe alerta: " Transição incomum: 95% dos pedidos passam por 'Aprovação'"
3. Solicita justificativa: "Por que pulou 'Aprovação'?"
4. Registra em log:

{

"timestamp": "2025-10-25T14:30:00",

"process*id": "proc*123",

"from_state": "Orçamento",

"to_state": "Entrega",

"expected_flow": false,

"pattern_match": 5%, // 95% não fazem isso

"justification": "Cliente VIP, aprovação verbal"

}

5. IA aprende: se isso virar padrão, ajusta sugestões

3. ARQUITETURA TÉCNICA

3.1. Visão Geral dos Componentes

■■■
■■

■ CAMADA DE APRESENTAÇÃO ■

■ ■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■ ■

■ ■ Kanban UI ■ ■ Form Builder ■ ■ Dashboard ■ ■

■ ■ (Drag&Drop)■ ■ (JSON/Visual)■ ■ (Analytics) ■ ■

■ ■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■ ■

■■■
■■

↓

■■■
■■

■ CAMADA DE APLICAÇÃO ■

■ ■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■ ■

■ ■ Workflow Manager ■ ■ Transition ■ ■ Pattern ■ ■

■ ■ - CRUD Kanbans ■ ■ Handler ■ ■ Analyzer ■ ■

■ ■ - CRUD Processos ■ ■ - Move cards ■ ■ (IA) ■ ■

■ ■ ■ ■ - Validate ■ ■ ■ ■

■ ■ ■ ■ - Alert ■ ■ ■ ■

■ ■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■ ■

■■■
■■

↓

■■■
■■

■ CAMADA DE PERSISTÊNCIA INTELIGENTE ■

■ ■■■ ■

■ ■ WorkflowRepository (Extended BaseRepository) ■ ■

■ ■ ■ ■

■ ■ Métodos Específicos: ■ ■

■ ■ - change*state(process*id, new_state) ■ ■

■ ■ - get*all*by*state(kanban*name, state) ■ ■

■ ■ - get*transition*history(process_id) ■ ■

■ ■ - log*transition(process*id, from, to, justification)■ ■

■ ■ - get*pattern*data(kanban_name) ■ ■

■ ■ ■ ■

■ ■ Backend-Agnostic (usa índices quando SQL) ■ ■

■ ■■■ ■

■■■
■■

↓

■■■
■■

■ CAMADA DE DADOS ■

■ ■■■■■■■■■ ■■■■■■■■■■ ■■■■■■■■ ■■■■■■■■■■■■ ■

■ ■ TXT ■ ■ SQLite ■ ■ JSON ■ ■ MongoDB ■ ... ■

■ ■■■■■■■■■ ■■■■■■■■■■ ■■■■■■■■ ■■■■■■■■■■■■ ■

■■■
■■

3.2. Estrutura de Diretórios

VibeCForms/

■■■ src/

■ ■■■ workflow/

■ ■ ■■■ **init**.py

■ ■ ■■■ engine/
■ ■ ■ ■■■ workflow_manager.py # CRUD Kanbans e Processos
■ ■ ■ ■■■ transition_handler.py # Movimentação de cards
■ ■ ■ ■■■ prerequisite_checker.py # Valida (não bloqueia)
■ ■ ■ ■■■ audit_logger.py # Log de auditoria
■ ■ ■■■ ai/
■ ■ ■ ■■■ pattern_analyzer.py # Detecta padrões
■ ■ ■ ■■■ anomaly_detector.py # Identifica transições raras
■ ■ ■ ■■■ suggestion_engine.py # Sugere otimizações
■ ■ ■■■ agents/
■ ■ ■ ■■■ base_agent.py # Agente base
■ ■ ■ ■■■ state_agent.py # Agente de estado
■ ■ ■■■ repository/
■ ■ ■■■ workflow_repository.py # Persistência inteligente
■ ■■■ workflows/ # Definições JSON de Kanbans
■ ■ ■■■ pedidos.json
■ ■ ■■■ suporte.json
■ ■ ■■■ ...
■ ■■■ templates/
■ ■ ■■■ kanban/
■ ■ ■■■ board.html # Interface Kanban
■ ■ ■■■ card.html # Card de processo
■ ■■■ static/
■ ■■■ js/
■ ■■■ kanban.js # Drag & Drop, alertas
■■■ data/ # Dados persistidos
■ ■■■ workflows*pedidos*processes.txt # Processos (backend config)
■ ■■■ workflows*pedidos*transitions.txt# Transições (log)
■ ■■■ workflows*pedidos*patterns.json # Padrões detectados (IA)
■■■ tests/
■■■ workflow/
■■■ test*workflow*manager.py
■■■ test*transition*handler.py
■■■ test*pattern*analyzer.py

4. MODELO DE DADOS

4.1. Kanban Definition (Definição de Processo)

Arquivo: src/workflows/{kanban_name}.json

{

"kanban_name": "pedidos",
"title": "Fluxo de Pedidos",
"description": "Gerenciamento de pedidos de clientes",
"icon": "fa-shopping-cart",
"created_at": "2025-10-25T10:00:00",
"states": [
{
"id": "orcamento",
"name": "Orçamento",
"order": 1,
"color": "#FFC107",
"icon": "fa-calculator",
"prerequisites": [
{
"id": "cliente_informado",
"type": "field_check",
"field": "cliente",
"condition": "not_empty",
"label": "Cliente informado",
"blocking": false,
"alert_message": " Cliente não informado. Deseja continuar?"
},
{
"id": "produtos_selecionados",
"type": "field_check",
"field": "produtos",
"condition": "not_empty",
"label": "Produtos selecionados",
"blocking": false,
"alert_message": " Nenhum produto selecionado. Deseja continuar?"
}
]
},
{
"id": "aprovacao",
"name": "Aprovação",
"order": 2,
"color": "#2196F3",
"icon": "fa-check-circle",

"prerequisites": [
{
"id": "valor_calculado",
"type": "field_check",
"field": "valor_total",
"condition": "greater_than",
"value": 0,
"label": "Valor total calculado",
"blocking": false,
"alert_message": " Valor total não calculado (R$ 0,00)."
}
]
},
{
"id": "entrega",
"name": "Entrega",
"order": 3,
"color": "#FF9800",
"icon": "fa-truck",
"prerequisites": [
{
"id": "pagamento_confirmado",
"type": "system_check",
"script": "scripts/check_payment.py",
"label": "Pagamento confirmado",
"blocking": false,
"alert_message": " Pagamento não confirmado. Prosseguir pode gerar problemas."
}
]
},
{
"id": "concluido",
"name": "Concluído",
"order": 4,
"color": "#4CAF50",
"icon": "fa-check",
"prerequisites": []
}
],

"agents": {
"enabled": true,
"state_assignments": {
"orcamento": "orcamento_agent",
"aprovacao": "aprovacao_agent",
"entrega": "entrega_agent"
},
"flow_sequence": [
"orcamento",
"aprovacao",
"entrega",
"concluido"
]
},

"ai_settings": {
"pattern_detection": true,
"min*samples*for_pattern": 10,
"anomaly_threshold": 0.1,
"learning_enabled": true
}
}

Características:

- Estados não limitam transições (UI permite mover para qualquer coluna)
- blocking: false em TODOS os pré-requisitos
- Agentes seguem flow_sequence (determinado, não "padrão detectado")

4.2. Process Instance (Instância de Processo)

Armazenado via: WorkflowRepository

{
"id": "proc_001",
"kanban_name": "pedidos",
"form_name": "pedidos",
"record_id": "123",

"current_state": "aprovacao",
"previous_state": "orcamento",

"created_at": "2025-10-25T10:00:00",
"updated_at": "2025-10-25T14:30:00",

"form*data*snapshot": {

"cliente": "Acme Corp",
"produtos": ["Produto A", "Produto B"],
"valor_total": 1500.00
},
"metadata": {
"priority": "high",
"tags": ["urgente", "cliente-vip"]
},
"last_transition": {
"from_state": "orcamento",
"to_state": "aprovacao",
"timestamp": "2025-10-25T14:30:00",
"was_anomaly": false,
"justification": null
}
}

4.3. Transition Log (Log de Transições - Auditoria)

Armazenado via: WorkflowRepository.log_transition()

{
"id": "trans_001",
"process*id": "proc*001",
"kanban_name": "pedidos",
"from_state": "orcamento",
"to_state": "entrega",
"timestamp": "2025-10-25T14:30:00",
"prerequisites_status": {
"cliente_informado": true,
"produtos_selecionados": true,
"valor_calculado": true,
"pagamento_confirmado": false
},
"was_anomaly": true,
"anomaly_reason": "Pulou estado 'aprovacao' (95% passam por ele)",
"pattern*match*score": 0.05,
"justification": "Cliente VIP solicitou urgência, aprovação verbal por telefone",
"triggered_by": "manual",

"agent_id": null

}

Campos de Auditoria (SEM sistema de usuários):

- timestamp: Quando aconteceu
- from*state* / *to*state: Estados anterior e atual
- justification: Por que fez (se fornecida)
- triggered*by: "manual" ou "agent:{agent*id}"
- NÃO inclui: user*id, user*name (sistema de usuários será futuro)

4.4. Pattern Data (Dados de Padrões - IA)

Armazenado: data/workflows*{kanban*name}_patterns.json

{

"kanban_name": "pedidos",

"last_analysis": "2025-10-25T15:00:00",

"total_processes": 150,

"total_transitions": 620,

"transition_frequencies": {

"orcamento → aprovacao": {

"count": 140,

"percentage": 0.93,

"avg*time*seconds": 3600,

"success_rate": 1.0

},

"orcamento → entrega": {

"count": 10,

"percentage": 0.07,

"avg*time*seconds": 1800,

"success_rate": 0.8,

"is_anomaly": true,

"anomaly_reason": "Raro, apenas 7% fazem"

},

"aprovacao → entrega": {

"count": 135,

"percentage": 0.90,

"avg*time*seconds": 7200

},

"entrega → concluido": {

"count": 150,

"percentage": 1.0,

"avg*time*seconds": 86400

}
},
"common_flows": [
{
"sequence": ["orcamento", "aprovacao", "entrega", "concluido"],
"count": 130,
"percentage": 0.87
},
{
"sequence": ["orcamento", "entrega", "concluido"],
"count": 10,
"percentage": 0.07
}
],
"bottlenecks": [
{
"state": "entrega",
"avg*time*seconds": 86400,
"reason": "Aguardando logística"
}
],
"suggestions": [
{
"type": "remove_state",
"state": "aprovacao",
"reason": "10% dos processos pulam esta etapa sem problemas",
"confidence": 0.3
}
]
}

5. CAMADA DE PERSISTÊNCIA INTELIGENTE

5.1. Extensão do BaseRepository

O WorkflowRepository estende o BaseRepository existente com operações específicas de workflow.

Arquivo: src/workflow/repository/workflow_repository.py

```
from src.persistence.repository import BaseRepository
from typing import List, Dict, Optional, Any
import json
```

```
from datetime import datetime

class WorkflowRepository(BaseRepository):
"""
Repositório inteligente para workflows.

Funcionalidades:
    • Herda CRUD básico do BaseRepository
    • Adiciona operações específicas de workflow
    • Usa otimizações do backend (índices em SQL)
    • NÃO valida regras de negócio (apenas persiste)
"""

def init(self, form_name: str):
"""
form_name será:
    • workflows{kanbanname}_processes (para processos)
    • workflows{kanbanname}_transitions (para logs)
    • workflows{kanbanname}_patterns (para IA)
"""
super().init(form_name)

# ========== OPERAÇÕES ESPECÍFICAS DE WORKFLOW ==========

def change_state(
self,
process_id: str,
new_state: str,
justification: Optional[str] = None
) -> Dict[str, Any]:
"""
Muda estado de um processo.

Backend-aware:
    • SQL: UPDATE processes SET current_state = ? WHERE id = ?
    • TXT: Reescreve linha com novo estado
    • JSON: Atualiza objeto no array

NÃO valida se transição é válida (isso é feito em TransitionHandler).
"""
# Busca processo atual
process = self.getbyid(process_id)
if not process:
raise ValueError(f"Process {process_id} not found")

oldstate = process.get('currentstate')
```

```
# Atualiza estado
process['previous_state'] = old_state
process['current_state'] = new_state
process['updated_at'] = datetime.utcnow().isoformat()
process['last_transition'] = {
'from_state': old_state,
'to_state': new_state,
'timestamp': datetime.utcnow().isoformat(),
'justification': justification
}

# Persiste (usa método update do BaseRepository)
self.update(process_id, process)

return process

def get_all_by_state(
self,
kanban_name: str,
state: str
) -> List[Dict[str, Any]]:
"""
Busca todos os processos em um estado específico.

Backend-aware:
    • SQL: SELECT * FROM processes WHERE current_state = ?
(usa índice em current_state)
    • TXT/JSON: Filtra em memória (lê todos, filtra)
"""
# Para backends SQL, o adapter pode otimizar com índice
all_processes = self.get_all()

return [
p for p in all_processes
if p.get('current_state') == state
]

def get_transition_history(
self,
process_id: str
) -> List[Dict[str, Any]]:
"""
Busca histórico de transições de um processo.

Lê do log de transições (workflows/{kanban}/transitions).
```

```
"""
# Cria repository para transitions
transitions_repo = WorkflowRepository(
f"workflows{self.formname.split('')[1]}transitions"
)

alltransitions = transitionsrepo.get_all()

return [
t for t in all_transitions
if t.get('processid') == processid
]

def log_transition(
self,
process_id: str,
from_state: str,
to_state: str,
was_anomaly: bool,
anomaly_reason: Optional[str],
patternmatchscore: float,
justification: Optional[str],
prerequisites_status: Dict[str, bool],
triggeredby: str, # "manual" ou "agent:{agentid}"
agent_id: Optional[str] = None
) -> str:
"""
Registra transição no log de auditoria.

Persiste em workflows{kanban}transitions.
"""
kanbanname = self.formname.split('_')[1]
transitions_repo = WorkflowRepository(
f"workflows{kanbanname}_transitions"
)

transition = {
"id": f"trans_{datetime.utcnow().timestamp()}",
"processid": processid,
"kanbanname": kanbanname,
"fromstate": fromstate,
"tostate": tostate,
"timestamp": datetime.utcnow().isoformat(),
```

"prerequisites*status": prerequisites*status,

"was*anomaly": was*anomaly,

"anomaly*reason": anomaly*reason,

"pattern*match*score": pattern*match*score,

"justification": justification,

"triggered*by": triggered*by,

"agent*id": agent*id

}

return transitions_repo.create(transition)

def get*pattern*data(self, kanban_name: str) -> Dict[str, Any]:

""""

Busca dados de padrões para análise de IA.

Lê todos os transitions e calcula estatísticas.

""""

transitions_repo = WorkflowRepository(

f"workflows*{kanban*name}_transitions"

)

all*transitions = transitions*repo.get_all()

Análise será feita por PatternAnalyzer

return {

"total*transitions": len(all*transitions),

"transitions": all_transitions

}

Características-Chave:

Herda BaseRepository: Mantém compatibilidade com sistema existente Operações Específicas: change_state(),

get*all*by_state(), etc Backend-Aware: SQL usa índices, TXT/JSON filtram em memória NÃO valida regras: Apenas

persiste dados, validação é em TransitionHandler Log separado: Transitions vão para tabela/arquivo separado

6. FLUXOS DE OPERAÇÃO

6.1. Fluxo 1: Criação de Kanban

ENTRADA: Usuário quer criar processo "Pedidos"

OPÇÕES DE CRIAÇÃO:

■■■

■ 1. JSON Direto ■

■ ■■ Edita src/workflows/pedidos.json manualmente ■

■ ■

■ 2. Interface Gráfica (FUTURA - Fase 3) ■

■ ■■ Arrasta colunas, define requisitos visualmente ■

■ ■

■ 3. Assistente IA (FUTURA - Fase 4) ■

■ ■■ Descreve processo, IA gera JSON ■

■■

RESULTADO:

Arquivo src/workflows/pedidos.json criado

Kanban aparece na lista de processos disponíveis

Pronto para receber processos (cards)

6.2. Fluxo 2: Movimentação de Processo (Usuário)

ENTRADA: Usuário arrasta card de "Orçamento" → "Entrega"

■■■

■ 1. Frontend (kanban.js) ■

■ ■■ Detecta drag & drop ■

■ ■■ Envia: POST /workflow/transition ■

■ { ■

■ "process*id": "proc*001", ■

■ "to_state": "entrega" ■

■ } ■

■■■

↓

■■■

■ 2. TransitionHandler.handle() ■

■ ■■ Busca processo atual (current_state="orcamento") ■

■ ■■ Busca definição do Kanban (pedidos.json) ■

■ ■■ Checa pré-requisitos do estado "entrega" ■

■ ■ ■■ pagamento_confirmado: FALSE ■

■ ■■ Consulta PatternAnalyzer ■

■ ■ ■■ Transição "orcamento→entrega": 5% (ANOMALIA) ■

■ ■■ DECISÃO: Permitir? SIM (SEMPRE) ■

■ ■■ Gera alerta para usuário ■

■ ■■ Solicita justificativa ■

■■■

↓

■■■

■ 3. Frontend Exibe Modal ■

```
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ Atenção ■ ■
■ ■ ■ ■
■ ■ Problemas detectados: ■ ■
■ ■ • Pagamento não confirmado ■ ■
■ ■ • Transição incomum (apenas 5% fazem isso) ■ ■
■ ■ ■ ■
■ ■ Justificativa (opcional para auditoria): ■ ■
■ ■ [______________________________] ■ ■
■ ■ ■ ■
■ ■ [Cancelar] [Prosseguir Mesmo Assim] ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
↓
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ 4. Usuário Justifica e Confirma ■
■ ■■ "Cliente VIP, urgência, aprovação verbal" ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
↓
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ 5. TransitionHandler.execute() ■
■ ■■ WorkflowRepository.change_state( ■
■ ■ processid="proc001", ■
■ ■ new_state="entrega", ■
■ ■ justification="Cliente VIP..." ■
■ ■ ) ■
■ ■■ WorkflowRepository.log_transition( ■
■ ■ from="orcamento", ■
■ ■ to="entrega", ■
■ ■ was_anomaly=True, ■
■ ■ justification="Cliente VIP...", ■
■ ■ triggered_by="manual" ■
■ ■ ) ■
■ ■■ PatternAnalyzer.record_transition() ■
■ ■■ Incrementa contador "orcamento→entrega" ■
■ ■■ Se virar padrão (>10%), ajusta sugestões ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
↓
RESULTADO:
Card movido para "Entrega"
```

Log de auditoria registrado

IA aprende novo dado

Se repetir, IA para de alertar

6.3. Fluxo 3: Movimentação de Processo (Agente IA)

ENTRADA: Agente "orcamento*agent" processando proc*001

■■■

■ 1. OrcamentoAgent.execute(process*id="proc*001") ■

■ ■■ Busca dados do processo ■

■ ■■ Preenche requisitos do estado "orcamento": ■

■ ■ ■■ Calcula valor_total (soma produtos) ■

■ ■ ■■ Valida cliente (consulta API) ■

■ ■ ■■ Marca pré-requisitos como ■

■ ■■ Checa definição do Kanban (pedidos.json) ■

■ ■ ■■ flow_sequence: ["orcamento", "aprovacao", ...]■

■ ■■ Próximo estado: "aprovacao" (determinado) ■

■ ■■ Chama TransitionHandler.handle() ■

■■■

↓

■■■

■ 2. TransitionHandler.handle() ■

■ ■■ Checa pré-requisitos: TODOS ■

■ ■■ Consulta PatternAnalyzer ■

■ ■ ■■ "orcamento→aprovacao": 95% (PADRÃO) ■

■ ■■ Transição normal, sem alertas ■

■ ■■ TransitionHandler.execute() ■

■ ■■ change*state(new*state="aprovacao") ■

■ ■■ log*transition(triggered*by="agent:orcamento")■

■ ■■ Chama próximo agente: AprovacaoAgent ■

■■■

↓

■■■

■ 3. AprovacaoAgent.execute(process*id="proc*001") ■

■ ■■ Preenche requisitos do estado "aprovacao": ■

■ ■ ■■ Verifica crédito do cliente ■

■ ■ ■■ Valida margem de lucro ■

■ ■ ■■ Marca pré-requisitos como ■

■ ■■ Próximo estado: "entrega" ■

■ ■■ Repete processo... ■

■■■

↓

RESULTADO:

Processo movido automaticamente

Requisitos preenchidos em cada etapa

Agentes encadeados conforme flow_sequence

Log registra triggered*by="agent:{agent*name}"

Diferença Crítica:

Aspecto	PDF Original	Versão Corrigida
Fluxo do Agente	Segue "padrão detectado pela IA"	■ Segue flow_sequence (determinado no JSON)
Responsabilidade	Agente decide próximo estado	■ Agente executa estado atual, JSON define próximo
Aprendizado	IA redefine fluxo	■ IA apenas sugere otimizações, fluxo é configurado

7. SISTEMA DE AUDITORIA

7.1. Estrutura do Log

SEM sistema de usuários (será implementado no futuro):

{

"id": "trans_001",

"process*id": "proc*001",

"kanban_name": "pedidos",

// ESTADOS

"from_state": "orcamento",

"to_state": "entrega",

// TEMPORAL (QUEM/QUANDO sem user_id)

"timestamp": "2025-10-25T14:30:00",

"triggered*by": "manual", // ou "agent:orcamento*agent"

"agent*id": null, // se triggered*by=agent, qual agente

// PRÉ-REQUISITOS

"prerequisites_status": {

"cliente_informado": true,

"produtos_selecionados": true,

"pagamento_confirmado": false

},

// ANÁLISE

"was_anomaly": true,

"anomaly_reason": "Pulou estado 'aprovacao' (95% passam por ele)",

"pattern*match*score": 0.05,

// JUSTIFICATIVA

"justification": "Cliente VIP solicitou urgência, aprovação verbal"

}

7.2. Queries de Auditoria

Exemplos de consultas:

1. Histórico de um processo

transitions = workflow*repo.get*transition*history("proc*001")

2. Todas as anomalias de hoje

all*transitions = transitions*repo.get_all()

anomalies_today = [

t for t in all_transitions

if t['was_anomaly']

and t['timestamp'].startswith('2025-10-25')

]

3. Transições feitas por agentes

agent_transitions = [

t for t in all_transitions

if t['triggered_by'].startswith('agent:')

]

4. Processos que pularam um estado específico

skipped_aprovacao = [

t for t in all_transitions

if t['from_state'] == 'orcamento'

and t['to_state'] != 'aprovacao'

]

7.3. Interface de Auditoria (Futura - Fase 5)

■■

■ Auditoria - Processo #proc_001 ■

■■

■ Timeline: ■

■ ■

■ ■■■ ■

■ ■ 2025-10-25 10:00 ■ manual ■ - → Orçamento■

■ ■ ■ ■ Processo criado ■ ■

■ ■■■ ■

■ ■

■ ■■■ ■

■ ■ 2025-10-25 14:30 ■ manual ■ Orçamento → Entrega ■ ■

■ ■ ■ ■ ANOMALIA ■ ■

■ ■ Justificativa: "Cliente VIP, urgência..." ■ ■

■ ■■ ■

■ ■

■ ■■ ■

■ ■ 2025-10-26 09:15 ■ agent ■ Entrega → Concluído ■ ■

■ ■ ■entrega ■ Automático ■ ■

■ ■■ ■

■ ■

■ [Exportar CSV] [Exportar PDF] ■

■■

8. AGENTES DE IA

8.1. Conceito de Agente de Estado

Cada agente é responsável por 1 coluna (estado) do Kanban:

Kanban "Pedidos":

■■

■ Orçamento ■ Aprovação ■ Entrega ■ Concluído ■

■ ↓ ■ ↓ ■ ↓ ■ ↓ ■

■ orcamento ■*aprovacao* ■entrega_ ■ - ■

■ agent ■ agent ■ agent ■ ■

■■

Responsabilidades do Agente:

1. Processar estado atual
   - Preencher campos necessários
   - Executar checagens (pré-requisitos)
   - Chamar APIs/serviços externos
2. Validar pré-requisitos
   - Marcar como cumpridos/não cumpridos
   - NÃO bloqueia se não cumpridos (apenas registra)
3. Transitar para próximo estado
   - Consulta flow_sequence do JSON
   - Chama TransitionHandler.handle()
   - Passa controle para próximo agente

8.2. Implementação Base

Arquivo: src/workflow/agents/base_agent.py

```
from abc import ABC, abstractmethod
from typing import Dict, Any, Optional
```

```
from src.workflow.engine.transition_handler import TransitionHandler
from src.workflow.repository.workflow_repository import WorkflowRepository

class BaseAgent(ABC):
"""
Agente base para processar estados de workflow.

Cada agente concreto implementa lógica de 1 estado.
"""

def __init__(
self,
kanban_name: str,
state_id: str,
next_agent: Optional['BaseAgent'] = None
):
self.kanban_name = kanban_name
self.state_id = state_id
self.next_agent = next_agent

self.workflow_repo = WorkflowRepository(
f"workflows_{kanban_name}_processes"
)
self.transition_handler = TransitionHandler(kanban_name)

def execute(self, process_id: str) -> bool:
"""
Executa processamento do estado.

Fluxo:
    1. Processar estado (implementado por subclasse)
    2. Validar pré-requisitos
    3. Transitar para próximo estado
    4. Chamar próximo agente

Returns:
True se processou com sucesso
"""
print(f"[{self.state_id}_agent] Processando {process_id}...")

# 1. Processar estado
success = self.process_state(process_id)
if not success:
print(f"[{self.state_id}_agent] Falha ao processar")
return False
```

```
# 2. Validar pré-requisitos
prerequisites = self.validate_prerequisites(process_id)

# 3. Buscar próximo estado (definido no JSON)
next_state = self.get_next_state()
if not next_state:
print(f"[{self.state_id}] Estado final, encerrando")
return True

# 4. Transitar
self.transition_handler.handle(
process_id=process_id,
to_state=next_state,
triggered_by=f"agent:{self.state_id}",
prerequisites_met=prerequisites
)

# 5. Chamar próximo agente
if self.next_agent:
self.next_agent.execute(process_id)

return True

@abstractmethod
def process_state(self, process_id: str) -> bool:
"""
Implementado por cada agente concreto.

Exemplo (OrcamentoAgent):
    • Calcular valor_total
    • Validar cliente em API externa
    • Atualizar snapshot do processo
"""
pass

@abstractmethod
def validate_prerequisites(self, process_id: str) -> Dict[str, bool]:
"""
Valida pré-requisitos do estado.

Returns:
{"cliente_informado": True, "produtos_selecionados": True, ...}
"""
pass

def get_next_state(self) -> Optional[str]:
```

"""

Busca próximo estado no flow_sequence do JSON.

"""

Lê definição do Kanban

with open(f"src/workflows/{self.kanban_name}.json") as f:

kanban_def = json.load(f)

flow*sequence* = *kanban*def.get('agents', {}).get('flow_sequence', [])

Encontra índice do estado atual

try:

current*index* = *flow*sequence.index(self.state_id)

if current*index* < *len(flow*sequence) - 1:

return flow*sequence[current*index + 1]

except ValueError:

pass

return None

8.3. Exemplo de Agente Concreto

Arquivo: src/workflow/agents/orcamento_agent.py

from src.workflow.agents.base_agent import BaseAgent

from typing import Dict, Any

class OrcamentoAgent(BaseAgent):

"""

Agente responsável pelo estado "Orçamento".

Tarefas:

- Calcular valor total do pedido
- Validar cliente (consulta API)
- Verificar disponibilidade de produtos

"""

def **init**(self, kanban_name: str = "pedidos"):

super().**init**(

kanban*name*=*kanban*name,

state_id="orcamento"

)

def process*state(self, process*id: str) -> bool:

"""

Processa orçamento do pedido.

"""

Busca processo

```
process = self.workflow_repo.get_by_id(process_id)
if not process:
return False

form_data = process.get('form_data_snapshot', {})

# 1. Calcular valor total
produtos = form_data.get('produtos', [])
valor_total = self.calcular_valor_total(produtos)

# 2. Atualizar snapshot
form_data['valor_total'] = valor_total
process['form_data_snapshot'] = form_data

# 3. Salvar
self.workflow_repo.update(process_id, process)

print(f"[orcamento_agent] Valor total calculado: R$ {valor_total}")
return True

def validate_prerequisites(self, process_id: str) -> Dict[str, bool]:
"""
Valida pré-requisitos do orçamento.
"""
process = self.workflow_repo.get_by_id(process_id)
form_data = process.get('form_data_snapshot', {})

return {
"cliente_informado": bool(form_data.get('cliente')),
"produtos_selecionados": len(form_data.get('produtos', [])) > 0,
"valor_calculado": form_data.get('valor_total', 0) > 0
}

def calcular_valor_total(self, produtos: list) -> float:
"""
Lógica de negócio: calcular valor total.

Em produção, consultaria tabela de preços, etc.
"""
# Exemplo simplificado
total = 0.0
for produto in produtos:
preco = self.get_preco_produto(produto)
total += preco

return total
```

```
def get_preco_produto(self, produto_nome: str) -> float:
"""
Consulta preço de produto (mock).
"""
# Mock - em produção consultaria banco de dados
precos = {
"Produto A": 500.0,
"Produto B": 1000.0,
"Produto C": 250.0
}
return precos.get(produto_nome, 0.0)
```

8.4. Orquestração de Agentes

Como agentes são encadeados:

```
# Arquivo: src/workflow/agents/orchestrator.py

def setup_pedidos_agents():
"""
Configura cadeia de agentes para Kanban "Pedidos".
"""
from src.workflow.agents.orcamento_agent import OrcamentoAgent
from src.workflow.agents.aprovacao_agent import AprovacaoAgent
from src.workflow.agents.entrega_agent import EntregaAgent

# Cria agentes
entrega_agent = EntregaAgent() # Último da cadeia
aprovacao_agent = AprovacaoAgent()
orcamento_agent = OrcamentoAgent()

# Encadeia
orcamento_agent.next_agent = aprovacao_agent
aprovacao_agent.next_agent = entrega_agent
entrega_agent.next_agent = None # Final

return orcamento_agent # Retorna primeiro da cadeia

# Uso:
first_agent = setup_pedidos_agents()
first_agent.execute("proc_001") # Processa todo o fluxo
```

9. INTERFACE DO USUÁRIO

9.1. Kanban Board

Template: src/templates/kanban/board.html

```
<!-- Visão simplificada -->
<div class="kanban-board" data-kanban="pedidos">

<div class="kanban-header">
<h2>■ Pedidos</h2>
<div class="kanban-stats">
<span>15 ativos</span>
<span>8 concluídos hoje</span>
</div>
</div>

<!-- Colunas geradas dinamicamente do JSON -->
<div class="kanban-columns">

<div class="kanban-column"
data-state="orcamento"
style="border-top: 3px solid #FFC107">
<div class="column-header">
<i class="fa fa-calculator"></i>
<h3>Orçamento</h3>
<span class="count">5</span>
</div>

<div class="column-cards" id="col-orcamento">
<!-- Cards (draggable) -->
<div class="kanban-card" data-process-id="proc_001" draggable="true">
<div class="card-header">
<strong>#001</strong>
<span class="priority-high"> Alta</span>
</div>
<div class="card-body">
<p>Cliente: Acme Corp</p>
<p>Valor: R$ 1.500,00</p>
</div>
<div class="card-footer">
<span>Há 2 horas</span>
</div>
</div>

<!-- Mais cards... -->
</div>
</div>
```

```
<div class="kanban-column" data-state="aprovacao">
<!-- ... -->
</div>

<div class="kanban-column" data-state="entrega">
<!-- ... -->
</div>

<div class="kanban-column" data-state="concluido">
<!-- ... -->
</div>

</div>

</div>
```

9.2. Modal de Alerta/Justificativa

```
<!-- Modal exibido quando transição tem problemas -->
<div id="transition-alert-modal" class="modal" style="display:none">
<div class="modal-content">

<div class="modal-header">
<h3> Atenção: Transição Incomum</h3>
</div>

<div class="modal-body">

<!-- Pré-requisitos não cumpridos -->
<div class="section">
<h4>Pré-requisitos não cumpridos:</h4>
<ul>
<li> Pagamento não confirmado</li>
</ul>
</div>

<!-- Análise de padrão -->
<div class="section">
<h4>Análise de padrão:</h4>
<p>
<i class="fa fa-chart-line"></i>
Esta transição é <strong>rara</strong>: apenas <strong>5%</strong> dos processos
vão de "Orçamento" direto para "Entrega".
</p>
<p>
<strong>95%</strong> passam por "Aprovação" antes.
```

```
</p>
</div>

<!-- Justificativa -->
<div class="section">
<h4>Justificativa (para auditoria):</h4>
<textarea id="justification-input"
placeholder="Ex: Cliente VIP solicitou urgência..."
rows="3"></textarea>
<small>Opcional, mas recomendado para auditoria</small>
</div>

</div>

<div class="modal-footer">
<button class="btn btn-secondary" onclick="cancelTransition()">
Cancelar
</button>
<button class="btn btn-warning" onclick="confirmTransition()">
Prosseguir Mesmo Assim
</button>
</div>

</div>
</div>
```

9.3. JavaScript (Drag & Drop + Alertas)

Arquivo: src/static/js/kanban.js

```
// Simplificado - usa SortableJS

document.addEventListener('DOMContentLoaded', function() {

// Inicializa drag & drop em todas as colunas
const columns = document.querySelectorAll('.column-cards');

columns.forEach(column => {
new Sortable(column, {
group: 'kanban',
animation: 150,

onEnd: function(evt) {
// Card foi dropado
const processId = evt.item.dataset.processId;
const newState = evt.to.parentElement.dataset.state;

// Envia ao backend
```

```
handleTransition(processId, newState);
}
});
});

});

async function handleTransition(processId, toState) {

// Envia requisição
const response = await fetch('/workflow/transition', {
method: 'POST',
headers: {'Content-Type': 'application/json'},
body: JSON.stringify({
process_id: processId,
to_state: toState
})
});

const result = await response.json();

if (result.needs_confirmation) {
// Tem problemas, exibe modal
showTransitionAlert(result.alerts, processId, toState);
} else {
// Sucesso direto
showToast(' Processo movido com sucesso');
}
}

function showTransitionAlert(alerts, processId, toState) {

// Popula modal com alertas
const modal = document.getElementById('transition-alert-modal');

// ... preenche dados ...

modal.style.display = 'block';

// Salva contexto para confirmação
window.pendingTransition = {
processId,
toState
};
}

async function confirmTransition() {
```

```
const justification = document.getElementById('justification-input').value;

const response = await fetch('/workflow/transition/confirm', {
method: 'POST',
headers: {'Content-Type': 'application/json'},
body: JSON.stringify({
process_id: window.pendingTransition.processId,
to_state: window.pendingTransition.toState,
justification: justification || null
})
});

if (response.ok) {
document.getElementById('transition-alert-modal').style.display = 'none';
showToast(' Processo movido');
}
}

function cancelTransition() {
// Desfaz drag & drop
location.reload();
}
```

10. PLANO DE IMPLEMENTAÇÃO POR FASES

Visão Geral das Fases

FASE 1: Fundação (Backend Core)
■■ Etapa 1.1: Persistência Inteligente
■■ Etapa 1.2: Workflow Manager
■■ Etapa 1.3: Transition Handler
■■ Etapa 1.4: Audit Logger

FASE 2: Interface Kanban
■■ Etapa 2.1: UI Básica
■■ Etapa 2.2: Drag & Drop
■■ Etapa 2.3: Modal de Alertas

FASE 3: Análise de Padrões (IA)
■■ Etapa 3.1: Pattern Analyzer
■■ Etapa 3.2: Anomaly Detector
■■ Etapa 3.3: Integração com UI

FASE 4: Agentes de IA
■■ Etapa 4.1: Base Agent

■■ Etapa 4.2: Agentes Concretos

■■ Etapa 4.3: Orquestração

FASE 5: Melhorias e Análises

■■ Etapa 5.1: Dashboard de Analytics

■■ Etapa 5.2: Editor Visual de Kanbans

■■ Etapa 5.3: Exportações e Relatórios

FASE 1: Fundação (Backend Core)

Objetivo: Criar infraestrutura backend para workflows sem bloqueios.

Etapa 1.1: Persistência Inteligente

Entregáveis:

1. WorkflowRepository
• Estende BaseRepository
• Implementa:
• change_state()
• get*all*by_state()
• get*transition*history()
• log_transition()
• get*pattern*data()
2. Estrutura de Dados
• workflows*{kanban}*processes (processos)
• workflows*{kanban}*transitions (log)
• workflows*{kanban}*patterns (dados IA)
3. Testes
• test*workflow*repository.py
• Testa CRUD + operações específicas
• Valida multi-backend (TXT, SQLite, JSON)

Critérios de Aceitação:

• Criar processo e persistir
• Mudar estado sem validações
• Buscar processos por estado
• Log de transição registrado
• Funciona em TXT, SQLite e JSON

Etapa 1.2: Workflow Manager

Entregáveis:

1. WorkflowManager Class

class WorkflowManager:

def create_kanban(definition: dict) -> str

def get*kanban(kanban*name: str) -> dict

def list_kanbans() -> list

def delete*kanban(kanban*name: str) -> bool

def create*process(kanban*name, form_data) -> str

def get*process(process*id) -> dict

def update*process(process*id, data) -> bool

def delete*process(process*id) -> bool

2. Carregamento de Definições

• Lê src/workflows/*.json

• Valida estrutura (JSON Schema)

• Cache em memória

3. API Endpoints (Flask)

Kanbans

POST /workflow/kanbans

GET /workflow/kanbans

GET /workflow/kanbans/<name>

DELETE /workflow/kanbans/<name>

Processos

POST /workflow/processes

GET /workflow/processes/<id>

GET /workflow/processes?kanban=<name>&state=<state>

PUT /workflow/processes/<id>

DELETE /workflow/processes/<id>

Critérios de Aceitação:

• CRUD completo de Kanbans via API

• CRUD completo de Processos via API

• Validação de JSON (estrutura correta)

• Filtragem de processos por estado

Etapa 1.3: Transition Handler

Entregáveis:

1. TransitionHandler Class

class TransitionHandler:

def handle(

process_id: str,

to_state: str,

triggered_by: str,

prerequisites_met: dict = None

) -> dict:

"""

Retorna:
{
"success": True,
"needs_confirmation": False,
"alerts": []
}

OU

{
"success": False,
"needs_confirmation": True,
"alerts": [
{
"type": "prerequisite_missing",
"message": "Pagamento não confirmado"
},
{
"type": "anomaly",
"message": "Transição rara (5%)"
}
]
}
"""

def execute(
process_id: str,
to_state: str,
justification: str = None,
triggered_by: str = "manual"
) -> bool:
"""
Executa transição SEM validar.
SEMPRE permite.
"""

2. PrerequisiteChecker Class

class PrerequisiteChecker:
def check(process, state_definition) -> dict:
"""
Retorna status de cada pré-requisito.
NÃO bloqueia, apenas informa.
"""

3. API Endpoints

POST /workflow/transition

{

"process*id": "proc*001",

"to_state": "entrega"

}

Resposta: {needs_confirmation: true/false, alerts: [...]}

POST /workflow/transition/confirm

{

"process*id": "proc*001",

"to_state": "entrega",

"justification": "Cliente VIP..."

}

Executa diretamente

Critérios de Aceitação:

- handle() detecta problemas mas NÃO bloqueia
- execute() SEMPRE executa transição
- Pré-requisitos checados mas não forçados
- Retorna alertas estruturados

Etapa 1.4: Audit Logger

Entregáveis:

1. AuditLogger Class

class AuditLogger:

def log_transition(

process_id,

from_state,

to_state,

timestamp,

triggered_by,

prerequisites_status,

was_anomaly,

anomaly_reason,

justification

) -> str

def get*log(process*id) -> list

def get*all*logs(kanban_name, filters) -> list

2. API Endpoints

GET /workflow/audit/process/<id>

GET /workflow/audit/kanban/<name>?from*date=...&to*date=...

GET /workflow/audit/anomalies?kanban=<name>

Critérios de Aceitação:

- Toda transição registrada em log
- Log inclui timestamp, estados, justificativa
- Busca de logs por processo
- Filtragem de anomalias

Validação da Fase 1:

Após concluir todas as etapas:

1. Teste Integrado:

```
# 1. Criar Kanban
kanban_id = workflow_manager.create_kanban(pedidos_definition)

# 2. Criar Processo
process_id = workflow_manager.create_process(
"pedidos",
{"cliente": "Acme", "produtos": ["A", "B"]}
)

# 3. Mover para estado anormal
result = transition_handler.handle(
process_id,
"entrega", # Pula "aprovacao"
"manual"
)

assert result["needs_confirmation"] == True
assert len(result["alerts"]) > 0

# 4. Confirmar mesmo assim
success = transition_handler.execute(
process_id,
"entrega",
justification="Teste"
)

assert success == True

# 5. Verificar log
logs = audit_logger.get_log(process_id)
assert len(logs) == 1
assert logs[0]["justification"] == "Teste"
```

2. Teste Multi-Backend:

- Executar acima com TXT
- Executar acima com SQLite
- Executar acima com JSON

FASE 2: Interface Kanban

Objetivo: Interface visual para mover processos entre colunas.

Etapa 2.1: UI Básica

Entregáveis:

1. Template board.html
- Header com nome do Kanban
- Colunas geradas dinamicamente
- Cards estáticos (sem drag)
2. Template card.html
- Dados do processo
- Badge de prioridade
- Timestamp
3. Rota Flask

@app.route('/workflow/<kanban_name>')

def kanban*board(kanban*name):

Carrega definição

Busca processos

Renderiza

Critérios de Aceitação:

- Exibe Kanban com colunas
- Mostra cards em cada coluna
- CSS responsivo

Etapa 2.2: Drag & Drop

Entregáveis:

1. SortableJS Integration
- Drag & drop entre colunas
- Animações
2. JavaScript handler
- Detecta drop
- Envia POST /workflow/transition

Critérios de Aceitação:

- Arrastar card entre colunas
- Backend atualiza estado
- UI reflete mudança

Etapa 2.3: Modal de Alertas

Entregáveis:

1. Modal HTML
• Lista de alertas
• Campo de justificativa
• Botões Cancelar/Confirmar
2. JavaScript modal logic
• Exibe quando needs_confirmation: true
• Envia justificativa para backend

Critérios de Aceitação:

• Modal exibido quando transição anormal
• Usuário pode justificar
• Justificativa salva em log

Validação da Fase 2:

1. Teste Manual:
• Abrir Kanban "Pedidos"
• Arrastar card de "Orçamento" → "Entrega"
• Ver modal de alerta
• Justificar e confirmar
• Verificar card na coluna correta
• Abrir auditoria, ver log com justificativa

FASE 3: Análise de Padrões (IA)

Objetivo: IA detecta padrões e anomalias.

Etapa 3.1: Pattern Analyzer

Entregáveis:

1. PatternAnalyzer Class

```
class PatternAnalyzer:
def analyze(kanban_name: str) -> dict:
"""
Retorna:
{
"transition_frequencies": {...},
"common_flows": [...],
"bottlenecks": [...],
"suggestions": [...]
}
"""
```

def get*transition*score(from*state, to*state) -> float:
"""
Retorna 0.0-1.0 (quão comum é essa transição)
"""

2. Algoritmo de Análise
• Lê todos os transitions
• Conta frequências
• Detecta sequências comuns
• Identifica gargalos (avg_time)

Critérios de Aceitação:

• Analisa 100+ transições corretamente
• Identifica padrão majoritário (>80%)
• Detecta anomalias (<10%)

Etapa 3.2: Anomaly Detector

Entregáveis:

1. AnomalyDetector Class

class AnomalyDetector:
def is_anomaly(
kanban_name,
from_state,
to_state,
threshold=0.1
) -> tuple[bool, str]:
"""
Retorna (is_anomaly, reason)
"""

2. Integração com TransitionHandler
• handle() consulta AnomalyDetector
• Adiciona alert se anomalia

Critérios de Aceitação:

• Transição rara marcada como anomalia
• Mensagem explicativa gerada
• Threshold configurável

Etapa 3.3: Integração com UI

Entregáveis:

1. Badge visual em colunas
• "95%" em setas entre colunas comuns
• "5%" em transições raras

2. API Endpoint

GET /workflow/analytics/<kanban_name>

Retorna transition*frequencies, common*flows, etc

3. JavaScript visualization

• Exibe porcentagens

• Destaca anomalias em vermelho

Critérios de Aceitação:

• UI mostra % de transições comuns

• Anomalias destacadas visualmente

Validação da Fase 3:

1. Teste com Dados Reais:

• Processar 50 pedidos seguindo padrão

• Processar 5 pedidos com anomalia

• Executar análise

• Verificar:

• 90% detectado como padrão

• 10% detectado como anomalia

• Mover novo processo

• Ver alerta se anomalia

FASE 4: Agentes de IA

Objetivo: Agentes automatizam preenchimento de requisitos e transições.

Etapa 4.1: Base Agent

Entregáveis:

1. BaseAgent Class (visto em seção 8.2)
2. Testes

• test*base*agent.py

• Mock de agente concreto

Critérios de Aceitação:

• BaseAgent executa fluxo completo

• Chama process*state() e validate*prerequisites()

• Transita para próximo estado

• Encadeia com next_agent

Etapa 4.2: Agentes Concretos

Entregáveis:

1. OrcamentoAgent (visto em seção 8.3)

• Calcula valor_total

• Valida cliente

2. AprovacaoAgent
• Verifica crédito
• Valida margem
3. EntregaAgent
• Verifica endereço
• Agenda entrega

Critérios de Aceitação:

• Cada agente preenche seus requisitos
• Valida pré-requisitos corretamente
• Log registra triggered_by="agent:..."

Etapa 4.3: Orquestração

Entregáveis:

1. Orchestrator Module
• Setup de cadeias de agentes
2. API Endpoint

POST /workflow/agents/start

{

"process*id": "proc*001",

"kanban_name": "pedidos"

}

Dispara primeiro agente da cadeia

3. Agendamento (Opcional)
• APScheduler
• Processar processos pendentes a cada X minutos

Critérios de Aceitação:

• Agentes encadeados corretamente
• Processo atravessa todos os estados
• Log completo de ações dos agentes

Validação da Fase 4:

1. Teste End-to-End:
• Criar processo manual
• Disparar agentes
• Verificar:
• Passa por todos os estados
• Requisitos preenchidos
• Log registra cada agente
• Estado final = "concluido"

FASE 5: Melhorias e Análises

Objetivo: Dashboard, editor visual, relatórios.

Etapa 5.1: Dashboard de Analytics

Entregáveis:

1. Template dashboard.html
• Visão geral de todos os Kanbans
• Métricas:
• Total de processos
• Taxa de conclusão
• Tempo médio por estado
• Gargalos
• Taxa de anomalias
2. Gráficos (Chart.js)
• Funil de conversão
• Timeline de estados
• Heatmap de transições

Critérios de Aceitação:

• Dashboard exibe métricas em tempo real
• Gráficos interativos
• Filtragem por período

Etapa 5.2: Editor Visual de Kanbans

Entregáveis:

1. Interface Drag & Drop
• Arrastar para criar colunas
• Clicar para editar pré-requisitos
2. Geração de JSON
• Exporta definição criada visualmente

Critérios de Aceitação:

• Criar Kanban sem editar JSON
• JSON gerado é válido
• Kanban funciona imediatamente

Etapa 5.3: Exportações e Relatórios

Entregáveis:

1. Export CSV
• Log de auditoria
• Processos por estado
2. Export PDF
• Relatório de processo individual
• Relatório de Kanban (analytics)

Critérios de Aceitação:

- Exportação CSV funcional
- PDF formatado corretamente

11. RISCOS E MITIGAÇÕES

-------|

manual |

Risco	Probabilidade	Impacto
----------	----------	---------
Performance com muitos processos	Média	Alto
IA detecta padrões errados	Média	Médio
Usuários não justificam anomalias	Alta	Baixo
Agentes entram em loop infinito	Baixa	Alto
Drag & drop não funciona mobile	Média	Médio
JSON mal formatado quebra sistema	Baixa	Alto

12. EXEMPLOS PRÁTICOS

12.1. Caso Completo: Pedidos

1. Criação do Kanban

```
// src/workflows/pedidos.json
{
"kanban_name": "pedidos",
"title": "Fluxo de Pedidos",
"states": [
{"id": "orcamento", "name": "Orçamento", ...},
{"id": "aprovacao", "name": "Aprovação", ...},
{"id": "entrega", "name": "Entrega", ...},
{"id": "concluido", "name": "Concluído", ...}
],
"agents": {
"flow_sequence": ["orcamento", "aprovacao", "entrega", "concluido"]
}
}
```

2. Uso Manual

Usuário cria pedido → Card em "Orçamento"

Usuário arrasta para "Entrega" (pula "Aprovação")

■■ Modal: " Transição rara (5%), justifique"

■■ Usuário: "Cliente VIP, urgência"

■■ Sistema: Move + registra log

3. Uso com Agentes

```
# Dispara agentes
orchestrator.start("proc_001", "pedidos")

# Fluxo:
# 1. OrcamentoAgent: calcula valor, valida cliente
# 2. AprovacaoAgent: verifica crédito, aprova
# 3. EntregaAgent: agenda entrega
# 4. Estado final: "concluido"
```

4. Análise

```
# Após 100 pedidos
analyzer.analyze("pedidos")

# Resultado:
# - 90% seguem: orcamento → aprovacao → entrega
# - 10% pulam aprovacao (anomalia)
# - Gargalo: entrega (avg 24h)
# - Sugestão: Adicionar "entrega_express" para VIPs
```

REVISÃO E PRÓXIMOS PASSOS

Este planejamento:

Corrige todas as divergências apontadas Kanban = 1 Processo (não regra de negócio) NUNCA bloqueia, apenas

avisa e audita Adapter inteligente de persistência Agentes determinísticos (não seguem "padrão") Auditoria

sem usuários (timestamp, estados, justificativa) SEM estimativas de horas Implementação em fases/etapas claras

Pronto para revisão!

Por favor, revise o planejamento e aponte:

- Algum conceito ainda divergente?
- Alguma seção confusa?
- Algum ajuste necessário?

Após sua aprovação, gero o PDF completo.

> Quando todos os pré requisitos de um dado estado já estão cumpridos o SISTEMA (não o usuário ou o agente de I.A.)

muda automáticamente para o próximo estado que tenham requisitos a cumprir (seguindo o fluxo padrão). O Usuário ou o

Agente podem retornar manualmente para algum estado anterior, onde será emitido um alerta de fluxo incomum e

solicitado a justificativa. O Agente de I.A. sempre justifica. O Usuário não é obrigado a justificar por enquanto

(posteriormente poderá ser dependendo do seu papel na empresa - Futuro).

Inclua se o sistema não ainda contemplar isso.

● Perfeito! Essa é uma funcionalidade crucial que muda significativamente o comportamento do sistema. Vou integrar

isso ao planejamento.

Deixe-me atualizar as seções relevantes:

ADENDO CRÍTICO: Transições Automáticas do Sistema

NOVA REGRA: Auto-Progressão por Cumprimento de Requisitos

Comportamento Automático

■■■
■■

■ TRANSIÇÃO AUTOMÁTICA DO SISTEMA ■

■■■
■■

■ ■

■ QUANDO: Todos os pré-requisitos de um estado são cumpridos ■

■ ■

■ O QUE FAZ: ■

■ 1. Sistema detecta: requisitos 100% OK ■

■ 2. Sistema busca próximo estado (flow_sequence) ■

■ 3. Sistema move automaticamente ■

■ 4. Repete até encontrar estado com requisitos pendentes ■

■ ■

■ NÃO GERA: Alertas (é movimento esperado) ■

■ NÃO PEDE: Justificativa ■

■ REGISTRA: Log com triggered_by="system" ■

■ ■

■■■
■■

Exemplo Prático

Estado Atual: "Orçamento"

Pré-requisitos:

cliente_informado: true

produtos_selecionados: true

valor_calculado: true

→ SISTEMA detecta: todos OK

→ SISTEMA move para: "Aprovação" (próximo no flow_sequence)

Estado Atual: "Aprovação"

Pré-requisitos:

credito_verificado: true (já estava OK)

margem_validada: true (já estava OK)

→ SISTEMA detecta: todos OK novamente

→ SISTEMA move para: "Entrega"

Estado Atual: "Entrega"

Pré-requisitos:

pagamento_confirmado: false (pendente)

endereco_validado: false (pendente)

→ SISTEMA para aqui (aguarda preenchimento)

TIPOS DE TRANSIÇÃO REVISADOS

1. Transição AUTOMÁTICA (Sistema)

Trigger: Pré-requisitos cumpridosDireção: Sempre para frente (flow_sequence)Alerta: NãoJustificativa: NãoLog:

triggered_by: "system"

Sistema detecta e move

auto*transition*engine.check*and*advance(process_id)

2. Transição MANUAL (Usuário)

Trigger: Drag & drop / botãoDireção: Qualquer (para frente ou para trás)Alerta: Se anormal ou para trásJustificativa: Opcional (por enquanto)Log: triggered_by: "manual"

Usuário arrasta card

transition_handler.handle(

process_id,

to_state="entrega",

triggered_by="manual"

)

3. Transição POR AGENTE (IA)

Trigger: Agente executandoDireção: Conforme flow_sequenceAlerta: Se anormal ou para trásJustificativa: OBRIGATÓRIA (sempre)Log: triggered*by: "agent:{agent*id}"

Agente move

```
agent.execute(process_id)
# Internamente:
transition_handler.handle(
process_id,
to_state=next_state,
triggered_by=f"agent:{agent_id}",
justification="Requisitos preenchidos automaticamente"
)
```

REGRAS DE ALERTA REVISADAS

Matriz de Alertas

Tipo	Direção	Pré-requisitos	Padrão	Alerta?	Justificativa
Sistema	Frente	■ Todos OK	Sim	■ Não	Não
Sistema	Frente	■ Todos OK	Não	■ Não	Não
Manual	Frente	■ Todos OK	Sim	■ Não	Não
Manual	Frente	■ Faltam	Sim	■■ Sim (leve)	Opcional
Manual	Frente	■ Faltam	Não	■■ Sim (forte)	Opcional
Manual	Trás	Qualquer	Qualquer	■■ Sim (sempre)	Opcional
Agente	Frente	■ Todos OK	Sim	■ Não	Sim (automática)
Agente	Frente	■ Faltam	Sim	■■ Sim	■ Obrigatória
Agente	Trás	Qualquer	Qualquer	■■ Sim (sempre)	■ Obrigatória

Legenda:

- Direção Frente: Segue flow_sequence (ex: Orçamento → Aprovação)
- Direção Trás: Contra flow_sequence (ex: Aprovação → Orçamento)
- Padrão: Conforme análise de IA (>80% fazem)

COMPONENTE NOVO: AutoTransitionEngine

Arquivo: src/workflow/engine/auto_transition_engine.py

```
from src.workflow.repository.workflow_repository import WorkflowRepository
from src.workflow.engine.prerequisite_checker import PrerequisiteChecker
from src.workflow.engine.transition_handler import TransitionHandler
import json

class AutoTransitionEngine:
"""
Responsável por mover processos automaticamente quando
todos os pré-requisitos de um estado são cumpridos.

Filosofia:
```

- Move para FRENTE (seguindo flow_sequence)
- NÃO gera alertas (é comportamento esperado)
- PARA quando encontra estado com requisitos pendentes
- Registra log com triggered_by="system"

"""

def **init**(self, kanban_name: str):

self.kanban*name* = *kanban*name

self.workflow_repo = WorkflowRepository(

f"workflows*{kanban*name}_processes"

)

self.prerequisite_checker = PrerequisiteChecker()

self.transition*handler* = *TransitionHandler(kanban*name)

Carrega definição do Kanban

with open(f"src/workflows/{kanban_name}.json") as f:

self.kanban_def = json.load(f)

def check*and*advance(self, process_id: str) -> bool:

"""

Verifica se processo pode avançar automaticamente.

Lógica:

1. Busca processo
2. Checa pré-requisitos do estado atual
3. Se TODOS OK:
   - Move para próximo estado (flow_sequence)
   - Repete recursivamente
4. Se algum pendente: PARA

Returns:

True se avançou (1+ estados)

"""

advanced = False

max_iterations = 10 # Previne loop infinito

for *in range(max*iterations):

Busca processo

process = self.workflow*repo.get*by*id(process*id)

if not process:

break

current*state* = *process['current*state']

Busca definição do estado atual

state*def* = *self.*get*state*definition(current_state)

```
if not state_def:
break

# Checa pré-requisitos
prerequisites_status = self._prerequisite_checker.check(
process,
state_def
)

# Todos cumpridos?
all_met = all(prerequisites_status.values())

if not all_met:
# Tem requisitos pendentes, PARA aqui
print(f"[AutoTransition] {process_id} parou em '{current_state}' (requisitos pendentes)")
break

# Busca próximo estado
next_state = self._get_next_state(current_state)
if not next_state:
# Último estado, nada a fazer
print(f"[AutoTransition] {process_id} já está no estado final")
break

# MOVE AUTOMATICAMENTE
print(f"[AutoTransition] {process_id}: '{current_state}' → '{next_state}' (auto)")

self.transition_handler.execute(
process_id=process_id,
to_state=next_state,
justification=None,
triggered_by="system",
skip_alerts=True # Não gera alertas
)

advanced = True

# Continua loop (checa próximo estado)

return advanced

def _get_state_definition(self, state_id: str) -> dict:
"""Busca definição de um estado no JSON."""
for state in self.kanban_def['states']:
if state['id'] == state_id:
return state
```

return None

def *get*next*state(self, current*state: str) -> str:

"""Busca próximo estado no flow_sequence."""

flow*sequence* = *self.kanban*def.get('agents', {}).get('flow_sequence', [])

try:

current*index* = *flow*sequence.index(current_state)

if current*index* < *len(flow*sequence) - 1:

return flow*sequence[current*index + 1]

except ValueError:

pass

return None

def monitor*all*processes(self) -> int:

"""

Monitora TODOS os processos do Kanban e avança os que podem.

Útil para:

- Execução periódica (cron job)
- Após agentes preencherem requisitos

Returns:

Quantidade de processos avançados

"""

all*processes* = *self.workflow*repo.get_all()

count = 0

for process in all_processes:

Ignora processos concluídos

if process['current_state'] == 'concluido':

continue

if self.check*and*advance(process['id']):

count += 1

print(f"[AutoTransition] {count} processos avançados automaticamente")

return count

MODIFICAÇÕES EM COMPONENTES EXISTENTES

1. TransitionHandler (Atualizado)

class TransitionHandler:

def execute(

self,

```
process_id: str,
to_state: str,
justification: str = None,
triggered_by: str = "manual",
skip_alerts: bool = False # ← NOVO
) -> bool:
"""
Executa transição.

skip_alerts=True: Usado por AutoTransitionEngine
"""

# ... (código existente)

# Log da transição
self.workflow_repo.log_transition(
process_id=process_id,
from_state=old_state,
to_state=to_state,
triggered_by=triggered_by, # "system", "manual", "agent:..."
justification=justification,
# ...
)

# Se foi mudança manual/agente E preenche requisitos:
# Dispara auto-transition
if triggered_by != "system" and not skip_alerts:
from src.workflow.engine.auto_transition_engine import AutoTransitionEngine
auto_engine = AutoTransitionEngine(self.kanban_name)
auto_engine.check_and_advance(process_id)

return True
```

2. BaseAgent (Atualizado)

```
class BaseAgent(ABC):

def execute(self, process_id: str) -> bool:
"""
Executa processamento do estado.
"""
# 1. Processar estado
success = self.process_state(process_id)

# 2. Validar pré-requisitos
prerequisites = self.validate_prerequisites(process_id)
```

```
# 3. Justificativa AUTOMÁTICA (agente sempre justifica)
justification = self.generate_justification(prerequisites)

# 4. Buscar próximo estado
next_state = self.get_next_state()

# 5. Transitar
self.transition_handler.handle(
process_id=process_id,
to_state=next_state,
triggered_by=f"agent:{self.state_id}",
justification=justification # ← OBRIGATÓRIA
)

return True

def generate_justification(self, prerequisites: dict) -> str:
"""
Gera justificativa automática do agente.

Exemplo:
"Agente orcamento_agent: Preenchidos requisitos cliente_id informado,
produtos_selecionados, valor_total calculado."
"""
met = [k for k, v in prerequisites.items() if v]
return f"Agente {self.state_id} agent: Preenchidos requisitos {', '.join(met)}."
```

3. Lógica de Alertas (Atualizada)

```
class TransitionHandler:

def handle(
self,
process_id: str,
to_state: str,
triggered_by: str,
prerequisites_met: dict = None
) -> dict:
"""
Avalia se transição precisa de confirmação/alerta.
"""

process = self.workflow_repo.get_by_id(process_id)
current_state = process['current_state']

alerts = []
```

```
needs_confirmation = False

# 1. MOVIMENTO PARA TRÁS? (sempre alerta)
if self.isbackward(currentstate, tostate):
alerts.append({
"type": "backward_transition",
"severity": "warning",
"message": f" Retrocesso: '{tostate}' vem ANTES de '{currentstate}' no fluxo padrão"
})
needs_confirmation = True

# Agente DEVE justificar
if triggered_by.startswith("agent:"):
alerts.append({
"type": "justification_required",
"severity": "error",
"message": " Agente deve fornecer justificativa"
})

# 2. PRÉ-REQUISITOS PENDENTES?
if prerequisitesmet and not all(prerequisitesmet.values()):
pending = [k for k, v in prerequisites_met.items() if not v]
alerts.append({
"type": "prerequisites_missing",
"severity": "info",
"message": f" Pré-requisitos pendentes: {', '.join(pending)}"
})
needs_confirmation = True

# 3. TRANSIÇÃO ANORMAL? (IA)
isanomaly, anomalyreason = self.anomalydetector.isanomaly(
self.kanban_name,
current_state,
to_state
)

if is_anomaly:
alerts.append({
"type": "anomaly",
"severity": "warning",
"message": f" {anomaly_reason}"
})
needs_confirmation = True
```

```
# 4. TRIGGERED BY SYSTEM? Nunca alerta
if triggered_by == "system":
alerts = []
needs_confirmation = False

return {
"success": True,
"needsconfirmation": needsconfirmation,
"alerts": alerts,
"justificationrequired": triggeredby.startswith("agent:")
}

def isbackward(self, fromstate: str, tostate: str) -> bool:
"""
Checa se movimento é para trás no flow_sequence.
"""
flowsequence = self.kanbandef.get('agents', {}).get('flow_sequence', [])

try:
fromindex = flowsequence.index(from_state)
toindex = flowsequence.index(to_state)
return toindex < fromindex # Indo para trás
except ValueError:
return False
```

FLUXOS ATUALIZADOS

Fluxo 1: Preenchimento Manual → Auto-Progressão

1. Processo em "Orçamento"
Requisitos: cliente, produtos, valor

2. Usuário preenche formulário:
   - Cliente: "Acme Corp"
   - Produtos: ["A", "B"]
   - (Valor é calculado automaticamente)

3. Sistema detecta: TODOS os requisitos OK
cliente_informado: true
produtos_selecionados: true
valor_calculado: true (auto)

4. AutoTransitionEngine.checkandadvance("proc_001")

5. Sistema move automaticamente:
"Orçamento" → "Aprovação"

6. Sistema checa "Aprovação":

credito_verificado: false (pendente)

7. Sistema PARA em "Aprovação"

RESULTADO:

Card agora em "Aprovação"

Log: triggered_by="system"

SEM alertas, SEM justificativa

Fluxo 2: Usuário Retrocede (Movimento para Trás)

1. Processo em "Entrega"
2. Usuário arrasta card para "Orçamento" (TRÁS)
3. TransitionHandler.handle() detecta:
   • is_backward("entrega", "orcamento") = TRUE
   • Gera alerta: " Retrocesso no fluxo"
4. Modal exibido:

■■

■ Movimento Incomum ■

■■

■ Você está movendo PARA TRÁS no fluxo: ■

■ "Entrega" → "Orçamento" ■

■ ■

■ Justificativa (opcional): ■

■ [____________________________________] ■

■ ■

■ [Cancelar] [Prosseguir] ■

■■

5. Usuário pode:
   • Justificar: "Erro no cálculo, refazer orçamento"
   • OU deixar em branco (permitido)
6. Sistema move e registra:

{

"triggered_by": "manual",

"was_backward": true,

"justification": "Erro no cálculo..." (ou null)

}

Fluxo 3: Agente Retrocede (DEVE Justificar)

1. AprovacaoAgent processando proc_001
2. Agente detecta: Crédito insuficiente

3. Agente decide retroceder para "Orçamento"

4. Agente chama:

```
transition_handler.handle(
processid="proc001",
to_state="orcamento",
triggered_by="agent:aprovacao",
justification="Crédito insuficiente. Cliente deve reduzir pedido."
)
```

5. TransitionHandler valida:
- is_backward = TRUE
- triggered_by = "agent:..."
- justification_required = TRUE
- justification presente? SIM

6. Sistema permite e registra

```
SE justification = None:
Erro: "AgentJustificationRequired"
```

ADIÇÃO À FASE 1

Etapa 1.5: Auto-Transition Engine (NOVA)

Entregáveis:

1. AutoTransitionEngine Class
- check*and*advance(process_id): Avança 1 processo
- monitor*all*processes(): Avança todos processos pendentes
- *get*next*state(): Consulta flow*sequence
- *get*state_definition(): Lê JSON

2. Integração com TransitionHandler
- Após transição manual/agente, dispara auto-check
- Sistema move automaticamente se requisitos OK

3. Testes
- test*auto*transition_engine.py
- Simula preenchimento de requisitos
- Valida que sistema avança automaticamente
- Testa que para quando encontra requisito pendente

4. API Endpoint (Opcional)

```
POST /workflow/auto-advance/<process_id>
# Força checagem e avanço manual (debug)

POST /workflow/monitor-all/<kanban_name>
# Processa todos os processos (cron job)
```

Critérios de Aceitação:

- Processo avança automaticamente quando requisitos OK
- Sistema para em estado com requisito pendente
- Log registra triggered_by="system"
- Sem alertas em movimentos automáticos
- Não entra em loop infinito (max_iterations)

REGRAS DE JUSTIFICATIVA (RESUMO)

Quem	Movimento	Justificativa
Sistema	Qualquer	■ Não (N/A)
Usuário	Para Frente	■ Opcional
Usuário	Para Trás	■ Opcional (por enquanto)
Agente	Para Frente	■ Obrigatória
Agente	Para Trás	■ Obrigatória

Nota: No futuro, justificativa pode ser obrigatória para usuários dependendo do papel/permissões.

PLANEJAMENTO ATUALIZADO

O planejamento agora contempla:

Transições automáticas quando requisitos cumpridos 3 tipos de transição: Sistema, Manual, Agente Alertas

inteligentes: Movimento para trás sempre alerta Justificativas: Agente obrigatória, Usuário opcional

Auto-progressão em cascata: Move até encontrar requisito pendente Log detalhado: triggered_by diferencia origem