

Documentação - Frontend do Sistema de Relatórios Genéricos

Índice

1. Visão Geral
 2. Arquitetura
 3. Stack Tecnológico
 4. Estrutura de Diretórios
 5. Gerenciamento de Estado
 6. Componentes Principais
 7. Views/Telas
 8. Fluxo de Dados
 9. Comunicação com API
 10. Modelo de Domínio
 11. Desenvolvimento
 12. Build e Deploy
 13. Testes
-

Visão Geral

O frontend do **Sistema de Relatórios Genéricos** é uma aplicação React moderna que permite aos usuários:

- **Criar workflows visuais** através de formulários intuitivos
- **Gerenciar nós** (etapas de processamento) com configurações detalhadas
- **Conectar nós** para definir fluxo de execução
- **Executar workflows** em tempo real com feedback via SSE
- **Interagir com IA** durante a execução (aprovações, correções)
- **Visualizar resultados** em formatos Markdown e JSON
- **Tema claro/escuro** com persistência

Características Principais

- **SPA (Single Page Application)** com React Router
 - **State Management** centralizado com Context API + useReducer
 - **Streaming em tempo real** via Server-Sent Events (SSE)
 - **Validação de workflows** no frontend antes da execução
 - **Arquitetura limpa** com separação de domínio, infra e apresentação
 - **TypeScript** para type safety completo
-

Arquitetura

Padrão Arquitetural: Clean Architecture + MVVM

PRESENTATION

| | | |
|---|----------------------------------|-----------------------------|
| Views (Dashboard, Node Mgr, Execution) | Components (Forms, Common) | Layouts (Header, Nav) |
|---|----------------------------------|-----------------------------|

STATE MANAGEMENT

```
WorkflowContext (useReducer)
- nodes[]
- connections[]
- documentos_anexados[]
- formato_resultado_final
- chat (messages, inputValue, isChatOpen)
```

DOMAIN LAYER

| | | |
|-----------------------------|----------------------------|------------------------------|
| NodeEntitie (Entity) | Grafo (Entity) | Workflow (Entity) |
| Aresta (Value Object) | ResultadoFinal (Entity) | Entrada, etc (Value Objs) |

INFRASTRUCTURE

Gateway Layer

WorkflowHttpGatewayV2

- gerarRelatorio()
- continuarInteracao()
- processStream() [SSE]

FetchAdapter

- get(), post(), put(), delete()

Backend API (FastAPI)

Fluxo de Navegação

Dashboard

Navigation

Gerenciar Nós
Gerenciar Conexões
Configurar Saídas
Executar Workflow

MainContent (renderiza view baseado em state)

NodeManager

NodeManagerCreate/Edit
FormCreateConnection

ConnectionManager

ConnectionsList

OutputConfiguration

useOutputConfiguration (hook)

WorkflowExecution

ExecuteProgress
InteractionBot (chat IA)
WorkflowOutput

WorkflowError

Stack Tecnológico

Core

| Tecnologia | Versão | Propósito |
|---------------------|--------|-------------------------|
| React | 19.1.1 | UI Framework |
| TypeScript | 5.9.3 | Type Safety |
| Vite | 7.1.7 | Build Tool & Dev Server |
| React Router | 7.9.4 | Roteamento SPA |

UI/Styling

| Tecnologia | Versão | Propósito |
|---------------------|--------|------------------------|
| Tailwind CSS | 3.4.17 | Utility-first CSS |
| Radix UI | - | Componentes acessíveis |
| Remixicon | 4.6.0 | Ícones |
| clsx | 2.1.1 | Manipulação de classes |

HTTP/API

| Tecnologia | Versão | Propósito |
|--------------|--------|--------------------|
| Axios | 1.12.2 | Cliente HTTP |
| SSE | Nativo | Server-Sent Events |

Testes

| Tecnologia | Versão | Propósito |
|-------------------------------|--------|-----------------------|
| Vitest | 3.2.4 | Test Runner |
| @testing-library/react | 16.3.0 | Testes de componentes |
| Cypress | 15.4.0 | Testes E2E |

Build/Dev

| Tecnologia | Versão | Propósito |
|---------------|--------|-----------|
| ESLint | 9.36.0 | Linter |

| Tecnologia | Versão | Propósito |
|----------------|--------|---------------|
| PostCSS | 8.5.5 | CSS Processor |
| serve | 14.2.5 | Static Server |

Estrutura de Diretórios

```

frontend/
  public/                # Assets estáticos
  src/
    assets/              # Imagens, fontes, etc.
    components/          # Componentes reutilizáveis
    common/              # Componentes comuns
      EmptyState.tsx
      ExecuteProgress.tsx
      InteractionBot.tsx  # Chat de interação com IA
      MarkdownRenderer.tsx
      ThemeToggle.tsx
      WorkflowError.tsx
      WorkflowOutput.tsx
      theme-provider.tsx
    forms/               # Formulários de criação/edição
      ConnectionsList.tsx
      FormCreateConnection.tsx
      ListNode.tsx
      NodeManagerCreate.tsx
      NodeManagerEdit.tsx

  context/               # Gerenciamento de estado global
    WorkflowContext.tsx  # Context API principal

  domain/                # Camada de domínio (Clean Arch)
    entities/            # Entidades de negócio
      Aresta.ts
      Grafo.ts
      NodeEntitie.ts
      ResultadoFinal.ts
      Workflow.ts

  gateway/               # Camada de infraestrutura (API)
    WorkflowGateway.ts    # Interface
    WorkflowHttpGateway.ts # Implementação HTTP (v1)
    WorkflowHttpGatewayV2.ts # Implementação HTTP (v2 - SSE)

```

```

hooks/                # Custom Hooks
  useOutputConfiguration.tsx

infra/                # Infraestrutura (HTTP, Storage, etc.)
  FetchAdapter.ts
  HttpClient.ts

layout/              # Componentes de layout
  Header.tsx
  MainContent.tsx
  Navigation.tsx

libs/                # Bibliotecas auxiliares

mock/                # Dados mock para desenvolvimento

tests/               # Testes unitários

types/               # Type definitions
  node.ts

ui/                  # Componentes UI primitivos (shadcn)
  progress.tsx

views/               # Views/Páginas principais
  ConnectionManager.tsx
  NodeManager.tsx
  OutputConfiguration.tsx
  WorkflowExecution.tsx

App.tsx              # Componente raiz
Dashboard.tsx        # Dashboard principal
main.tsx              # Entry point
index.css             # Estilos globais
setup.ts              # Setup de testes

cypress/             # Testes E2E
  e2e/
  fixtures/
  support/

package.json
vite.config.ts
tsconfig.json
tailwind.config.js
eslint.config.js

```

Estatísticas

- **Total de linhas:** ~6.328 linhas de código
 - **Linguagem:** TypeScript 100%
 - **Componentes:** ~30 componentes React
-

Gerenciamento de Estado

WorkflowContext

Arquivo: `src/context/WorkflowContext.tsx`

O estado global da aplicação é gerenciado usando **React Context API** + **useReducer** para garantir: - Previsibilidade (padrão Redux-like) - Performance (evita prop drilling) - Debugging facilitado

Estado Global (WorkflowState)

```
interface WorkflowState {
  nodes: NodeState[]; // Nós do workflow
  connections: Connection[]; // Conexões entre nós
  documentos_anexados: DocumentoAnexado[]; // Documentos anexados
  formato_resultado_final?: FormatoResultadoFinal; // Config de output
  chat: ChatState; // Estado do chat de interação
}
```

Estado do Chat

```
interface ChatState {
  messages: ChatMessage[]; // Histórico de mensagens
  inputValue: string; // Input atual do usuário
  isChatOpen: boolean; // Chat está aberto?
}
```

Actions Disponíveis

| Action | Payload | Descrição |
|-----------------------------|---------------------------------|-----------------------------|
| ADD_NODE | NodeState | Adiciona novo nó |
| UPDATE_NODE | NodeState | Atualiza nó existente |
| DELETE_NODE | { nodeId, chavesDocumentos } | Remove nó e docs associados |
| ADD_DOCUMENTO_ANEXO | DocumentoAnexado | Adiciona documento |
| REMOVE_DOCUMENTOS_POST_CHAT | string[] | Remove documentos |
| ADD_CONNECTION | Connection | Adiciona conexão |
| UPDATE_CONNECTION | Connection | Atualiza conexão |

| Action | Payload | Descrição |
|------------------------|---|---------------------------|
| DELETE_CONNECTION | { connectionId } | Remove conexão |
| UPDATE_RESULTADO_FINAL | { combinacoes, saidas_individuais } | Configura output |
| RESET_WORKFLOW | - | Limpa todo o workflow |
| ADD_CHAT_MESSAGE | ChatMessage | Adiciona mensagem ao chat |
| SET_CHAT_INPUT_VALUE | string | Atualiza input do chat |
| SET_CHAT_OPEN | boolean | Abre/fecha chat |
| CLEAR_CHAT_MESSAGES | - | Limpa histórico do chat |
| SET_CHAT_MESSAGES | ChatMessage[] | Define mensagens do chat |

Context API - Funções Expostas

```
interface WorkflowContextType {
  state: WorkflowState;
  dispatch: React.Dispatch<WorkflowAction>;

  // Node actions
  addNode: (node: NodeState) => void;
  updateNode: (node: NodeState) => void;
  deleteNode: (nodeId: string, chavesDocumentos?: string[]) => void;

  // Document actions
  addDocumentoAnexo: (documento: DocumentoAnexado) => void;
  removeDocumentosPorChave: (chaves: string[]) => void;

  // Connection actions
  addConnection: (connection: Connection) => void;
  updateConnection: (connection: Connection) => void;
  deleteConnection: (connectionId: string) => void;

  // Output configuration
  updateResultadoFinal: (combinacoes, saidas_individuais) => void;

  // Chat actions
  addChatMessage: (message: ChatMessage) => void;
  setChatInputValue: (value: string) => void;
  setChatOpen: (isOpen: boolean) => void;
  clearChatMessages: () => void;
  setChatMessages: (messages: ChatMessage[]) => void;

  // Workflow actions
  resetWorkflow: () => void;
  getWorkflowJSON: () => string; // Serializa para JSON
}
```

```

    validateWorkflow: () => { isValid: boolean; errors: string[] };
  }

```

Uso do Context

```

// Em qualquer componente:
import { useWorkflow } from '@context/WorkflowContext';

function MyComponent() {
  const { state, addNode, getWorkflowJSON } = useWorkflow();

  // Acessar estado
  const nodes = state.nodes;

  // Adicionar nó
  addNode({
    id: crypto.randomUUID(),
    nome: "Análise",
    prompt: "Analise o documento",
    entrada_grafo: true,
    saida: { nome: "resultado", formato: "markdown" },
    entradas: [],
    ferramentas: []
  });

  // Gerar JSON do workflow
  const json = getWorkflowJSON();
}

```

Componentes Principais

1. Dashboard

Arquivo: src/Dashboard.tsx

Componente raiz da aplicação que gerencia: - **Estado da view** atual (nodes, connections, output, execution) - **Bloqueio de navegação** durante execução de workflow - **Layout principal** (Header + Navigation + MainContent)

```

function Dashboard() {
  const [currentView, setCurrentView] = useState<ViewType>('nodes');
  const [isNavigationLocked, setIsNavigationLocked] = useState(false);

  return (
    <div>
      <Header />

```

```

    <Navigation
      currentView={currentView}
      onViewChange={setCurrentView}
      isNavigationLocked={isNavigationLocked}
    />
    <MainContent
      currentView={currentView}
      onNavigationLock={setIsNavigationLocked}
    />
  </div>
);
}

```

2. Navigation

Arquivo: src/layout/Navigation.tsx

Barra de navegação lateral com 4 seções:

| Seção | ViewType | Ícone | Descrição |
|---------------------------|---------------|-------|---------------------------------|
| Gerenciar Nós | 'nodes' | | Criar/editar nós |
| Gerenciar Conexões | 'connections' | | Conectar nós |
| Configurar Saídas | 'output' | | Definir formato do resultado |
| Executar Workflow | 'execution' | | Executar e visualizar resultado |

Features: - Bloqueio durante execução (isNavigationLocked) - Indicador visual de seção ativa - Responsivo

3. MainContent

Arquivo: src/layout/MainContent.tsx

Renderiza a view correta baseado em currentView:

```

function MainContent({ currentView, onNavigationLock }) {
  const renderView = () => {
    switch (currentView) {
      case 'nodes':
        return <NodeManager />;
      case 'connections':
        return <ConnectionManager />;
      case 'output':

```

```

        return <OutputConfiguration />;
      case 'execution':
        return <WorkflowExecution onNavigationLock={onNavigationLock} />;
    }
  };

  return <main>{renderView()}</main>;
}

```

4. NodeManager

Arquivo: src/views/NodeManager.tsx

Gerencia criação e edição de nós: - **Lista de nós** com opções de editar/deletar
 - **Formulário de criação** (NodeManagerCreate) - **Formulário de edição**
 (NodeManagerEdit)

5. NodeManagerCreate / NodeManagerEdit

Arquivos: - src/components/forms/NodeManagerCreate.tsx - src/components/forms/NodeManagerEdit.

Formulários complexos com validação para configurar:

Campos do Formulário Básicos: - **Nome do nó** (obrigatório, único) -
Prompt (instrução para o agente IA) - **Entrada do grafo** (checkbox)

Configurações LLM: - **Modelo LLM** (opcional, ex: “gemini-2.5-pro”) -
Temperatura (0.0 - 2.0, opcional) - **Ferramentas** (multi-select: stf, tcu, tcepe,
 busca-internet)

Entradas (array): - **Variável no prompt** (ex: {documento}) - **Origem:** docu-
 mento_anexado | resultado_no_anterior - **Chave do documento** (se origem =
 documento_anexado) - **Nome do nó origem** (se origem = resultado_no_an-
 terior) - **Executar em paralelo** (checkbox)

Saída: - **Nome da saída** - **Formato:** markdown | json

Interação com Usuário (opcional): - **Permitir usuário finalizar** - **IA pode**
concluir - **Requer aprovação explícita** - **Máximo de interações** (1-100) -
Modo de saída: ultima_mensagem | historico_completo | ambos

6. ConnectionManager

Arquivo: src/views/ConnectionManager.tsx

Interface para criar conexões (arestas) entre nós: - Visualiza conexões existentes
 - Permite criar novas (origem → destino) - Suporta “END” como destino especial
 - Validação de grafos acíclicos

7. OutputConfiguration

Arquivo: `src/views/OutputConfiguration.tsx`

Configura formatação do resultado final:

Opções: 1. **Combinações:** Mescla outputs de múltiplos nós - Nome da saída combinada - Nós fonte (multi-select) - Manter originais (opcional)

2. **Saídas Individuais:** Retorna outputs específicos separadamente

Hook: `useOutputConfiguration.tsx` gerencia a lógica

8. WorkflowExecution

Arquivo: `src/views/WorkflowExecution.tsx`

View mais complexa que gerencia execução:

Estados de Execução

```
type ExecutionState =  
  | 'idle' // Pronto para executar  
  | 'executing' // Executando  
  | 'awaiting_interaction' // Aguardando usuário  
  | 'completed' // Concluído  
  | 'error'; // Erro
```

Funcionalidades

1. **Validação pré-execução** (via `validateWorkflow()`)
2. **Execução via SSE** (streaming em tempo real)
3. **Progresso visual** (`ExecuteProgress`)
4. **Interação com chat** (`InteractionBot`)
5. **Visualização de resultados** (`WorkflowOutput`)
6. **Tratamento de erros** (`WorkflowError`)

Eventos SSE Processados

| Evento | Descrição | Ação |
|--|-----------------------|--|
| <code>started</code> | Nó iniciou | Atualiza status, inicia timer |
| <code>finished</code> | Nó concluiu | Atualiza status, calcula duração |
| <code>awaiting_interaction</code> | Pausou para interação | Abre chat, salva <code>session_id</code> |
| <code>interaction_limit_reached</code> | Limite atingido | Notifica usuário, continua |
| <code>completed</code> | Workflow concluído | Exibe resultados |

| Evento | Descrição | Ação |
|--------|--------------|------------------------|
| error | Erro ocorreu | Exibe mensagem de erro |

9. InteractionBot

Arquivo: src/components/common/InteractionBot.tsx

Componente de chat para interação humano-IA:

Features: - Interface de chat moderna - Histórico de mensagens (usuário + bot) - Envio de mensagens durante execução - Aprovação/rejeição de outputs - Animações de digitação

Integração:

```
<InteractionBot
  sessionId={sessionId}
  onSendMessage={handleContinueInteraction}
  isOpen={isChatOpen}
  onClose={() => setChatOpen(false)}
  agentMessage={interactionData?.agent_message}
/>
```

10. ExecuteProgress

Arquivo: src/components/common/ExecuteProgress.tsx

Barra de progresso visual com: - **Porcentagem** calculada dinamicamente - **Status de cada nó** (pending, running, completed, error) - **Tempo de execução** por nó - **Animações** de transição

11. WorkflowOutput

Arquivo: src/components/common/WorkflowOutput.tsx

Renderiza resultados do workflow:

Suporta: - **Markdown** (via MarkdownRenderer) - **JSON** (syntax highlighting) - **Metadados:** formato, nós fonte, tamanho, etc. - **Download** de resultados

12. MarkdownRenderer

Arquivo: src/components/common/MarkdownRenderer.tsx

Renderizador customizado de Markdown com: - **Syntax highlighting** para código - **Tabelas** formatadas - **Links** estilizados - **Download** como arquivo .md - **Copy to clipboard**

Views/Telas

Workflow de Uso

1. Usuário acessa Dashboard
↓
 2. Navega para "Gerenciar Nós"
↓
 3. Cria nós com NodeManagerCreate
↓
 4. Navega para "Gerenciar Conexões"
↓
 5. Conecta nós (define fluxo)
↓
 6. Navega para "Configurar Saídas" (opcional)
↓
 7. Define combinações de output
↓
 8. Navega para "Executar Workflow"
↓
 9. Clica em "Executar"
↓
 10. Acompanha progresso em tempo real
↓
 11. (Opcional) Interage via chat se workflow pausar
↓
 12. Visualiza resultados ao final
-

Fluxo de Dados

1. Criação de Workflow

NodeManagerCreate (UI)

Preenche formulário

Clica "Adicionar Nó"

Validação local (NodeEntitie.validate())

addNode(nodeData)

WorkflowContext.dispatch({ type: 'ADD_NODE', payload })

workflowReducer atualiza state.nodes[]

UI atualiza automaticamente (React re-render)

2. Execução de Workflow

WorkflowExecution (UI)

Usuário clica "Executar"

validateWorkflow() Valida localmente

Se inválido: exibe erros
Se válido: prossegue

getWorkflowJSON() Serializa state para JSON

WorkflowHttpGatewayV2.gerarRelatorio(json, callbacks)

POST /executar_workflow (SSE)

processStream() Lê eventos SSE

| | |
|------------------------|---------------------|
| 'started' | onNodeStarted() |
| 'finished' | onNodeFinished() |
| 'awaiting_interaction' | Abre chat |
| 'completed' | onCompleted(result) |
| 'error' | onError(error) |

UI atualiza em tempo real via setState

3. Interação durante Execução

InteractionBot (Chat)

Usuário digita mensagem

Clica "Enviar"

WorkflowHttpGatewayV2.continuarInteracao(sessionId, message)

POST /continuar_interacao (SSE)

processStream() Lê eventos SSE

'awaiting_interaction' Nova mensagem do bot

| | |
|-------------|---------------------|
| 'completed' | Workflow finalizado |
| 'error' | Erro |

Chat atualiza com resposta do agente

Comunicação com API

WorkflowHttpGatewayV2

Arquivo: src/gateway/WorkflowHttpGatewayV2.ts

Implementação da comunicação com backend via SSE.

Método: gerarRelatorio()

```
async gerarRelatorio(
  requestData: any,
  callbacks: GerarDocCallbacks
): Promise<any>
```

Parâmetros: - requestData: JSON do workflow - callbacks: Objeto com handlers para eventos

Callbacks:

```
interface GerarDocCallbacks {
  onNodeStarted?: (nodeName: string) => void;
  onNodeFinished?: (nodeName: string, output: any) => void;
  onCompleted?: (result: any) => void;
  onError?: (error: string) => void;
  onAwaitingInteraction?: (data: InteractionData) => void;
  onInteractionLimitReached?: (data: any) => void;
}
```

Fluxo: 1. Faz POST para /executar_workflow 2. Recebe stream SSE (ReadableStream) 3. Processa eventos linha por linha 4. Chama callbacks apropriados 5. Retorna quando stream finaliza

Método: continuarInteracao()

```
async continuarInteracao(
  sessionId: string,
  userMessage: string,
  callbacks: GerarDocCallbacks
): Promise<any>
```

Parâmetros: - sessionId: ID da sessão pausada - userMessage: Mensagem do usuário - callbacks: Mesmos callbacks de gerarRelatorio()

Payload:

```
{
  "session_id": "abc-123",
  "user_response": "Adicione mais detalhes...",
  "approve": false
}
```

FetchAdapter

Arquivo: src/infra/FetchAdapter.ts

Wrapper sobre fetch API nativa que implementa HttpClient interface.

Métodos: - get<T>(url, config) - post<T>(url, data, config) - put<T>(url, data, config) - delete<T>(url, config)

Features: - Suporte a **streaming** (responseType: 'stream') - Headers customizáveis - Timeout configurável - Error handling

Modelo de Domínio

Entidades Principais

1. **NodeEntitie** **Arquivo:** src/domain/entities/NodeEntitie.ts

Representa um nó (etapa) do workflow.

```
class NodeEntitie {
  constructor(
    public readonly nome: string,
    public readonly prompt: string,
    public readonly entrada_grafo: boolean = false,
    public readonly saida: NodeOutput,
    public readonly interacao_com_usuario?: InteracaoComUsuario,
    public readonly entradas: Entrada[] = [],
    public readonly modelo_llm?: string,
    public readonly temperatura?: number,
    public readonly ferramentas: string[] = []
  ) {}

  validate(existingNodes: NodeEntitie[] = []): void {
    // Valida:
    // - Nome obrigatório e único
    // - Prompt obrigatório
    // - Saída obrigatória
    // - Apenas uma entrada paralela
  }
}
```

```

    }
}

```

Interfaces:

```

interface Entrada {
    variavel_prompt: string;
    origem: "documento_anexado" | "resultado_no_anterior";
    chave_documento_origem?: string;
    nome_no_origem?: string;
    executar_em_paralelo?: boolean;
}

interface InteracaoComUsuario {
    permitir_usuario_finalizar: boolean;
    ia_pode_concluir: boolean;
    requer_aprovacao_explicita: boolean;
    maximo_de_interacoes: number;
    modo_de_saida: "ultima_mensagem" | "historico_completo" | "ambos";
}

interface NodeOutput {
    nome: string;
    formato: "markdown" | "json";
}

```

2. Aresta Arquivo: src/domain/entities/Aresta.ts

Representa uma conexão entre nós.

```

class Aresta {
    constructor(
        public readonly origem: string, // Nome do nó origem
        public readonly destino: string // Nome do nó destino ou "END"
    ) {}
}

```

3. Grafo Arquivo: src/domain/entities/Grafo.ts

Representa o grafo completo do workflow.

```

class Grafo {
    constructor(
        public readonly nos: NodeEntitie[],
        public readonly arestas: Aresta[]
    ) {}

    validate(): void {

```

```

        // Valida:
        // - Pelo menos um nó de entrada (entrada_grafo: true)
        // - Nomes únicos
        // - Arestas referenciam nós existentes
        // - Grafo é acíclico (DAG)
    }
}

```

4. Workflow Arquivo: src/domain/entities/Workflow.ts

Entidade raiz que representa o workflow completo.

```

class Workflow {
    constructor(
        public readonly documentos_anexados: DocumentoAnexado[],
        public readonly grafo: Grafo,
        public readonly formato_resultado_final?: FormatoResultadoFinal
    ) {}

    validate(): void {
        // Valida grafo e documentos anexados
    }

    toJsonString(): string {
        // Serializa para JSON compatível com API
    }
}

```

Interfaces:

```

interface DocumentoAnexado {
    chave: string;
    descricao: string;
    uuid_unico?: string;
    uuids_lista?: string[];
}

```

5. FormatoResultadoFinal Arquivo: src/domain/entities/ResultadoFinal.ts

Configura como os resultados devem ser retornados.

```

class FormatoResultadoFinal {
    constructor(
        public readonly combinacoes: Combinacao[],
        public readonly saidas_individuais: string[]
    ) {}
}

```

```
interface Combinacao {
  nome_da_saida: string;
  combinar_resultados: string[]; // Nomes dos outputs a combinar
  manter_originais: boolean;
}
```

Desenvolvimento

Pré-requisitos

- Node.js: 18+
- npm ou yarn

Instalação

```
cd frontend
npm install
```

Variáveis de Ambiente

Criar arquivo .env.local:

```
# API Backend
VITE_API_URL=http://localhost:5016
VITE_API_AUTH_TOKEN=seu_token_aqui
```

Ambientes suportados: - .env.local - Desenvolvimento local -
 .env.development - Modo dev (cross-env) - .env.production - Produção

Scripts Disponíveis

```
# Desenvolvimento (porta 5960)
npm run dev          # NODE_ENV=development
npm run local        # NODE_ENV=local
```

```
# Build
npm run build        # NODE_ENV=production
```

```
# Preview da build
npm run preview
```

```
# Servir build estática
npm run start          # Porta 5960
```

```
# Lint
npm run lint
```

```

# Testes
npm test                # Todos os testes
npm run test:node       # Testes de NodeEntitie
npm run test:grafo      # Testes de Grafo
npm run test:aresta     # Testes de Aresta
npm run test:workflow   # Testes de Workflow

```

Estrutura de URL

Base Path: /relatorios/

Produção:

<https://aurora.tcepe.tc.br/relatorios/>

Desenvolvimento:

<http://localhost:5960/relatorios/>

Configurado em: - vite.config.ts → base: '/relatorios/' - App.tsx →
 <Router basename="/relatorios">

Hot Module Replacement (HMR)

Vite suporta HMR out-of-the-box: - Alterações em .tsx/.ts recarregam automaticamente - Alterações em .css aplicam sem reload - Estado React é preservado quando possível

Path Aliases

Configurados em vite.config.ts:

```

{
  '@': './src',
  '@components': './src/components',
  '@lib': './src/lib',
  '@hooks': './src/hooks'
}

```

Uso:

```

import { useWorkflow } from '@context/WorkflowContext';
import Button from '@components/common/Button';

```

Build e Deploy

Build para Produção

```
npm run build
```

Output: `dist/` directory

Otimizações: - Code splitting automático - Tree shaking - Minificação (CSS + JS) - Assets otimizados

Estrutura da Build

```
dist/
  assets/
    index-[hash].js      # Bundle principal
    vendor-[hash].js     # Dependências
    index-[hash].css     # Estilos
  index.html             # Entry point
  ...
```

Deploy

Via Docker (Recomendado) O frontend é servido via **nginx** no container principal:

```
# Em docker-compose.yml
frontend:
  build:
    context: ./frontend
  ports:
    - "5960:5960"
  environment:
    - VITE_API_URL=http://api:5016
```

Deploy Manual

```
# Build
npm run build

# Servir com servidor estático
npm run start

# Ou usar nginx, Apache, etc.
```

Configuração Nginx

```
server {
  listen 5960;
  root /usr/share/nginx/html;
  index index.html;

  location /relatorios {
    try_files $uri $uri/ /relatorios/index.html;
```

```

    }

    # Proxy para API
    location /relatorios/api {
        proxy_pass http://api:5016;
    }
}

```

Testes

Testes Unitários (Vitest)

Framework: Vitest + Testing Library

Executar:

```

npm test                # Todos
npm run test:node        # Apenas NodeEntitie
npm run test:grafo       # Apenas Grafo

```

Exemplo de teste:

```

import { describe, it, expect } from 'vitest';
import NodeEntitie from '@domain/entities/NodeEntitie';

describe('NodeEntitie', () => {
  it('deve validar nome obrigatório', () => {
    expect(() => {
      const node = new NodeEntitie('', 'prompt', false, { nome: 'out', formato: 'markdown' });
      node.validate();
    }).toThrow('Nome do nó é obrigatório');
  });

  it('deve validar nomes únicos', () => {
    const node1 = new NodeEntitie('analise', 'prompt1', false, { nome: 'out1', formato: 'markdown' });
    const node2 = new NodeEntitie('analise', 'prompt2', false, { nome: 'out2', formato: 'markdown' });

    expect(() => {
      node2.validate([node1]);
    }).toThrow('Já existe um nó com o nome "analise"');
  });
});

```

Cobertura:

```
npm test -- --coverage
```

Testes E2E (Cypress)

Framework: Cypress 15.4.0

Executar:

```
npx cypress open    # UI interativa
npx cypress run     # Headless
```

Estrutura:

```
cypress/
  e2e/           # Specs de testes
  fixtures/      # Dados mock
  support/       # Comandos customizados
```

Exemplo:

```
describe('Workflow Creation', () => {
  it('deve criar um nó com sucesso', () => {
    cy.visit('/relatorios');
    cy.get('[data-testid="add-node-btn"]').click();
    cy.get('input[name="nome"]').type('Análise Inicial');
    cy.get('textarea[name="prompt"]').type('Analise o documento');
    cy.get('button[type="submit"]').click();
    cy.contains('Nó adicionado com sucesso').should('be.visible');
  });
});
```

Troubleshooting

Erro: “useWorkflow must be used within a WorkflowProvider”

Causa: Componente usando useWorkflow() fora do WorkflowProvider.

Solução: Garantir que App.tsx envolve a aplicação:

```
<WorkflowProvider>
  <Router>
    { /* suas rotas */ }
  </Router>
</WorkflowProvider>
```

SSE não está funcionando

Causas possíveis: 1. CORS bloqueando conexão 2. URL da API incorreta 3. Token de autenticação inválido

Debug:

```
// Verificar console do navegador
console.log('API URL:', import.meta.env.VITE_API_URL);
console.log('Token:', import.meta.env.VITE_API_AUTH_TOKEN);
```

Build falha com erro de tipo

Causa: TypeScript encontrou erros de tipo.

Solução:

```
# Ver erros
npm run build

# Verificar tipos sem build
npx tsc --noEmit
```

HMR não está funcionando

Causa: Porta 5960 já em uso.

Solução:

```
# Matar processo na porta
lsof -ti:5960 | xargs kill -9

# Ou mudar porta em vite.config.ts
server: { port: 5961 }
```

Roadmap

- ☐ Visualização gráfica do workflow (drag-and-drop)
- ☐ Persistência de workflows no backend
- ☐ Histórico de execuções
- ☐ Templates de workflows pré-configurados
- ☐ Modo offline (Service Worker)
- ☐ Exportação de workflows (JSON/YAML)
- ☐ Versionamento de workflows
- ☐ Colaboração multi-usuário
- ☐ Dashboard de analytics

Contribuindo

Padrões de Código

- **TypeScript:** Sempre tipar explicitamente
- **Componentes:** Functional components com hooks

- **Naming:** PascalCase para componentes, camelCase para funções
- **Imports:** Usar path aliases (@/...)
- **Styling:** Tailwind classes (evitar CSS inline)

Git Workflow

Seguir o GitFlow documentado em: <https://git.tce.pe/gdsi/tce-ia/-/wikis/gitflow>

Última Atualização: 23/11/2025 **Versão:** 1.0.0 **Mantido por:** Equipe GDSI/TCE-PE