Sistema de Relatórios Automatizados SRAG

Arquitetura Técnica Detalhada

Organização: ABC HealthCare Inc.

Versão: 1.0.0

Data: Setembro 2025

Autor: Equipe de Desenvolvimento ABC HealthCare Inc.

---

1. Visão Geral da Arquitetura

1.1 Introdução

O Sistema de Relatórios Automatizados SRAG é uma solução baseada em Inteligência Artificial Generativa projetada para auxiliar profissionais da área da saúde no monitoramento e análise de surtos de Síndrome Respiratória Aguda Grave (SRAG).

1.2 Objetivos Arquiteturais

- **Modularidade**: Componentes independentes e reutilizáveis
- **Escalabilidade**: Capacidade de processar grandes volumes de dados
- **Governança**: Transparência e auditoria completas
- **Segurança**: Proteção rigorosa de dados sensíveis de saúde
- **Manutenibilidade**: Código limpo e bem documentado

1.3 Princípios de Design

1. **Separação de Responsabilidades**: Cada componente tem função específica
2. **Arquitetura Baseada em Agentes**: Orquestração inteligente de tarefas
3. **Design orientado a Tools**: Ferramentas especializadas e intercambiáveis
4. **Guardrails First**: Segurança e validação em todas as camadas

---

2. Arquitetura de Componentes

2.1 Camada de Apresentação

2.1.1 Aplicação Principal (main.py)

Responsabilidade: Ponto de entrada do sistema, gerencia interface CLI

Componentes:

SRAGApplication
├── Inicialização do sistema
├── Validação de parâmetros
├── Coordenação de execução
└── Tratamento de erros globais

Interfaces:

- CLI com argumentos: `--date`, `--status-only`, `--no-charts`, `--no-news`
- Retorno estruturado em JSON
- Logs detalhados de execução

2.2 Camada de Orquestração

2.2.1 Agente Orquestrador (SRAGOrchestrator)

Responsabilidade: Coordena todas as operações e ferramentas

Fluxo de Execução:

1. Inicialização → Validação de Request
2. Carregamento de Dados → DatabaseTool
3. Cálculo de Métricas → MetricsCalculatorTool
4. Geração de Gráficos → ChartGeneratorTool (opcional)
5. Análise de Notícias → NewsSearchTool (opcional)
6. Compilação Final → ReportGeneratorTool
7. Validação de Saída → Guardrails

Características:

- Estado de execução rastreável
- Decisões auditadas
- Recuperação de erros
- Métricas de performance

2.3 Camada de Ferramentas (Tools)

2.3.1 DatabaseTool

Função: Acesso e processamento de dados SRAG

Operações:

- Carregamento otimizado por chunks
- Parsing e validação de CSV
- Transformações de dados
- Cache inteligente
- Anonimização automática

Pipeline de Processamento:

Arquivo CSV → Chunks → Validação → Limpeza → Transformação → Campos Derivados → Cache

2.3.2 MetricsCalculatorTool

Função: Cálculo das métricas epidemiológicas

Métricas Implementadas:

1. **Taxa de Aumento de Casos**
    - Fórmula: `((Casos Atuais - Casos Anteriores) / Casos Anteriores) × 100`
    - Período padrão: 30 dias
    - Comparação temporal
2. **Taxa de Mortalidade**
    - Fórmula: `(Óbitos / Total de Casos) × 100`
    - Período padrão: 90 dias
    - Classificação: Baixa (<5%), Moderada (5-15%), Alta (>15%)
3. **Taxa de Ocupação de UTI**
    - Fórmula: `(Casos em UTI / Total Hospitalizados) × 100`
    - Período padrão: 30 dias
    - Indicador de gravidade
4. **Taxa de Vacinação**
    - Fórmula: `(Casos Vacinados / Total de Casos) × 100`
    - Período padrão: 90 dias
    - Breakdown por tipo de dose

2.3.3 NewsSearchTool

Função: Busca e análise de notícias contextuais

Fontes:

- RSS Feeds confiáveis (G1, Folha, Estadão, Fiocruz)
- News API (opcional)
- Filtros de confiabilidade

Processamento:

Busca → Filtro de Relevância → Deduplicação → Análise de Contexto → Score → Integração

2.3.4 ChartGeneratorTool

Função: Geração de visualizações

Gráficos Gerados:

1. **Gráfico Diário** (30 dias)
    - Linha temporal
    - Média móvel 7 dias
    - Marcação de picos
2. **Gráfico Mensal** (12 meses)
    - Gráfico de barras
    - Comparação temporal
    - Tendências

Formatos: PNG, SVG, HTML (Plotly interativo)

2.3.5 ReportGeneratorTool

Função: Compilação do relatório final

Template HTML:

- Responsivo (Bootstrap 5)
- Seções organizadas
- Gráficos integrados
- Metadados completos

2.4 Camada de Governança

2.4.1 Sistema de Guardrails (SRAGGuardrails)

Validações Aplicadas:

1. **Validação de Entrada**
    - Formato de datas
    - Ranges aceitáveis
    - Parâmetros obrigatórios
2. **Validação de Dados**
    - Integridade temporal
    - Consistência de valores
    - Detecção de outliers
3. **Proteção de Dados Sensíveis**

Dados Brutos → Identificação PII → Anonimização → Agregação → Validação → Dados Protegidos

4. **Validação de Saída**
    - Verificação de completude
    - Ranges de métricas
    - Assinatura digital

Dados Sensíveis Removidos:

- CPF, RG, CNS
- Nomes de pacientes
- Telefones e emails
- Endereços específicos
- Datas de nascimento exatas

2.4.2 Sistema de Auditoria

Logs Estruturados:

```
{
  "timestamp": "ISO-8601",
  "component": "nome_componente",
  "event_type": "tipo_evento",
  "user_action": "ação",
  "data_accessed": "recursos",
  "decision": "decisão_tomada",
  "reasoning": "justificativa"
}
```

Rastreabilidade:

- ID único por execução
- Trail completo de decisões

- Timestamps em todas operações
- Correlação de eventos

3. Fluxo de Dados

3.1 Fluxo Principal

Usuário
↓
Interface CLI
↓
Validação Inicial (Guardrails)
↓
Orquestrador
↓
DatabaseTool | NewsSearch | MetricsTool | ChartTool
↓
ReportGeneratorTool
↓
Validação Final (Guardrails)
↓
Relatório HTML + Gráficos

3.2 Fluxo de Segurança

Dados Brutos (DataSUS CSV)
↓
Carregamento Seguro
↓
Identificação de Dados Sensíveis
↓
Anonimização Automática
↓
Validação de Integridade
↓
Processamento
↓
Agregação (quando necessário)
↓
Validação Final
↓
Dados Seguros para Relatório

4. Segurança e Privacidade

4.1 Conformidade LGPD

Princípios Aplicados:

1. **Finalidade**: Dados usados apenas para relatórios SRAG
2. **Adequação**: Tratamento alinhado com objetivo de saúde pública
3. **Necessidade**: Mínimo de dados necessários
4. **Transparência**: Logs completos de processamento
5. **Segurança**: Múltiplas camadas de proteção

4.2 Medidas de Proteção

Camada	Medida	Implementação
Entrada	Validação	Schema validation, type checking
Processamento	Anonimização	Remoção automática de PII
Armazenamento	Agregação	Dados agrupados geograficamente
Saída	Filtros	Verificação de vazamento de dados
Auditoria	Logs	Registro completo de acessos

4.3 Criptografia

- **Em trânsito**: HTTPS para APIs externas
- **Em repouso**: Arquivos de configuração sensíveis
- **Logs**: Sanitização automática de dados sensíveis

---

5. Performance e Escalabilidade

5.1 Otimizações Implementadas

1. **Processamento em Chunks**
    - Tamanho configurável (padrão: 10.000 registros)
    - Memória controlada
    - Processamento paralelo quando possível
2. **Cache Inteligente**
    - Dados processados em memória

    - TTL configurável (padrão: 1 hora)
    - Invalidação automática
3. **Otimização de Memória**
    - Downcasting de tipos numéricos
    - Categorização de strings repetidas
    - Garbage collection explícito

5.2 Métricas de Performance

Benchmarks (dataset 10.000 registros):

- Carregamento de dados: < 5s
- Cálculo de métricas: < 2s por métrica
- Geração de gráficos: < 3s por gráfico
- Relatório completo: < 30s

5.3 Escalabilidade

Capacidades:

- Suporta datasets de até 1M registros
- Processamento paralelo de chunks
- Distribuição possível via containers
- API REST futura para integração

---

6. Tecnologias Utilizadas

6.1 Stack Principal

Componente	Tecnologia	Versão	Propósito
Linguagem	Python	3.9+	Desenvolvimento principal
IA/LLM	LangChain	0.1+	Orquestração de agentes
LLM	OpenAI GPT	4+	Análise contextual
Dados	Pandas	2.0+	Manipulação de dados
Gráficos	Matplotlib/Plotly	-	Visualizações
Logs	Structlog	23+	Logs estruturados

Validação	Pydantic	2.0+	Validação de schemas

6.2 Dependências Críticas

Core:

- pandas, numpy (processamento)
- structlog (logging)
- langchain, openai (IA)
- pydantic, python-dotenv (configuração)

Opcionais:

- plotly (gráficos interativos)
- redis (cache distribuído)
- pytest (testes)

---

7. Padrões de Código

7.1 Clean Code

Princípios Aplicados:

1. **SOLID**
    - Single Responsibility
    - Open/Closed
    - Liskov Substitution
    - Interface Segregation
    - Dependency Inversion
2. **DRY** (Don't Repeat Yourself)
    - Classes base reutilizáveis
    - Utilitários compartilhados
    - Configurações centralizadas
3. **KISS** (Keep It Simple)
    - Funções pequenas e focadas
    - Nomes descritivos
    - Lógica clara

7.2 Convenções

Nomenclatura:

Classes: PascalCase

```
class MetricsCalculatorTool:
 pass
```

Funções/métodos: snake_case

```
def calculate_mortality_rate():
 pass
```

Constantes: UPPER_SNAKE_CASE

```
MAX_RETRY_ATTEMPTS = 3
```

Privados: prefixo _

```
def _internal_method():
 pass
```

Documentação:

```
def function_name(param: type) -> return_type:
    """
    Descrição curta da função.

    Args:
        param: Descrição do parâmetro

    Returns:
        Descrição do retorno

    Raises:
        ExceptionType: Quando ocorre erro
    """
```

8. Deployment

8.1 Requisitos de Sistema

Mínimo:

- Python 3.9+
- 2GB RAM

- 1GB espaço em disco
- Conexão internet (para APIs)

Recomendado:

- Python 3.11+
- 4GB RAM
- 5GB espaço em disco
- Conexão estável

8.2 Instalação

```
# 1. Clonar repositório
git clone <repo>

# 2. Criar ambiente virtual
python -m venv venv
source venv/bin/activate

# 3. Instalar dependências
pip install -r requirements.txt

# 4. Configurar .env
cp .env.example .env
# Editar .env com chaves de API

# 5. Baixar dados SRAG
# Colocar em data/raw/srag_data.csv

# 6. Executar
python -m src.main --status-only
```

9. Monitoramento e Manutenção

9.1 Logs

Localizações:

- Aplicação: `logs/srag_system.log`
- Auditoria: `logs/audit_trail.json`
- Erros: `logs/errors.log`

9.2 Métricas de Saúde

Health Checks:

- Status de cada ferramenta
- Conectividade com fontes de dados
- Uso de memória
- Tempo de resposta

9.3 Manutenção

Atividades Regulares:

- Atualização de dados SRAG (mensal)
- Revisão de logs de erro (semanal)
- Atualização de dependências (trimestral)
- Backup de configurações (contínuo)

11. Glossário

- **SRAG**: Síndrome Respiratória Aguda Grave
- **DataSUS**: Departamento de Informática do Sistema Único de Saúde
- **LLM**: Large Language Model
- **PII**: Personally Identifiable Information
- **LGPD**: Lei Geral de Proteção de Dados
- **ETL**: Extract, Transform, Load
- **API**: Application Programming Interface
- **REST**: Representational State Transfer
- **CSV**: Comma-Separated Values
- **JSON**: JavaScript Object Notation
- **TTL**: Time To Live
- **CLI**: Command Line Interface

---

12. Referências

1. OpenDataSUS - https://opendatasus.saude.gov.br/
2. LangChain Documentation - https://docs.langchain.com/
3. LGPD - Lei nº 13.709/2018
4. Clean Code - Robert C. Martin
5. Python Best Practices - PEP 8

Documento gerado em: Setembro 2025
Versão do Sistema: 1.0.0
Status: Production Ready (PoC)
