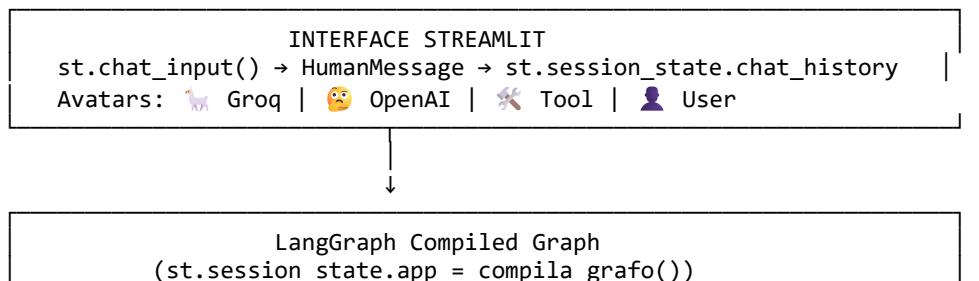


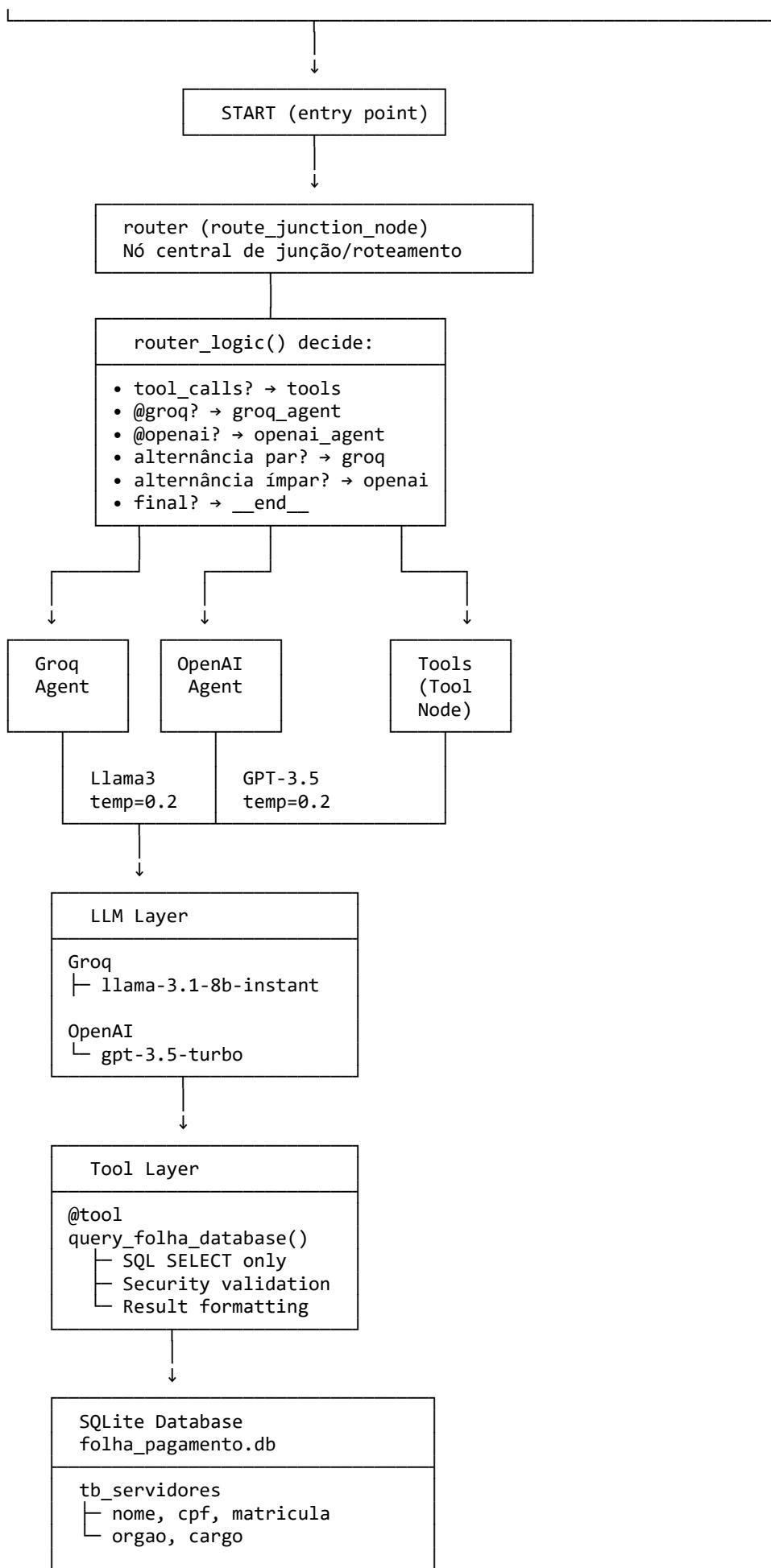
# Architecture

- [1 Arquitetura do FACIN\\_IA](#)
  - [1.1 🏢 Arquitetura Geral](#)
  - [1.2 🤖 Sistema Multi-Agentes](#)
    - [1.2.1 Componentes Principais](#)
  - [1.3 📊 Estado da Conversa \(AgentState\)](#)
  - [1.4 ⚙️ Fluxo de Execução](#)
    - [1.4.1 1. Entrada do Usuário](#)
    - [1.4.2 2. Inicialização do Grafo](#)
    - [1.4.3 3. Processamento](#)
    - [1.4.4 4. Uso de Ferramentas](#)
    - [1.4.5 5. Resposta Final](#)
  - [1.5 🗂️ Camada de Dados](#)
    - [1.5.1 Banco de Dados SQLite](#)
    - [1.5.2 Fonte de Dados](#)
  - [1.6 🔗 Integrações Externas](#)
    - [1.6.1 LangChain](#)
    - [1.6.2 LangGraph](#)
    - [1.6.3 Streamlit Session State](#)
    - [1.6.4 AgenticOps \(Opcional\)](#)
  - [1.7 📁 Configuração CI/CD](#)
    - [1.7.1 GitHub Actions Pipeline](#)
  - [1.8 🔒 Segurança](#)
    - [1.8.1 Variáveis de Ambiente](#)
    - [1.8.2 Validação de Input](#)
    - [1.8.3 SQL Injection Prevention](#)
  - [1.9 📈 Escalabilidade](#)
    - [1.9.1 Nível de Maturidade 1](#)
  - [1.10 🔍 Monitoramento \(AgenticOps\)](#)
    - [1.10.1 Eventos Rastreados](#)
    - [1.10.2 Métricas Coletadas](#)
  - [1.11 📚 Referências de Padrões](#)
    - [1.11.1 Design Patterns Utilizados](#)
  - [1.12 🚀 Performance](#)
    - [1.12.1 Otimizações](#)

## 1 Arquitetura do FACIN\_IA

### 1.1 🏢 Arquitetura Geral





```

tb_folha_pagamento
└ matricula, competencia
└ vencimentos, descontos,
liquido

```

```

AgenticOps (Opcional)
- Event Logging
- Performance Monitoring
- Error Tracking

```

## 1.2 Sistema Multi-Agentes

### 1.2.1 Componentes Principais

#### 1.2.1.1. Agent Router (router\_logic)

Função: Lógica de roteamento condicional

Entrada: AgentState com histórico de mensagens

Saída: Nome do próximo nó ("groq\_agent", "openai\_agent", "tools", "\_\_end\_\_")

#### Decisões de Roteamento:

1. Se AIMessage.tool\_calls existe → "tools"
2. Se AIMessage sem tool\_calls → "\_\_end\_\_"
3. Se HumanMessage com "@groq" → "groq\_agent"
4. Se HumanMessage com "@openai" → "openai\_agent"
5. Se ToolMessage ou alternância:
  - ai\_count % 2 == 0 → "groq\_agent"
  - ai\_count % 2 != 0 → "openai\_agent"

#### 1.2.1.2. Groq Agent (groq\_agent\_node)

Modelo: llama-3.1-8b-instant

Temperatura: 0.2

Função: Consultar banco de dados da Folha de Pagamento

Ferramentas:

- query\_folha\_database()

Saída: {"messages": [AIMessage]}

**Prompt do Sistema:** > Você é um assistente de Folha de Pagamento experiente chamado Groq (modelo Llama3). > Use a ferramenta 'query\_folha\_database' fornecendo uma consulta SQL SELECT válida.

#### 1.2.1.3. OpenAI Agent (openai\_agent\_node)

Modelo: gpt-3.5-turbo

Temperatura: 0.2

Função: Consultar banco de dados da Folha de Pagamento

Ferramentas:

- query\_folha\_database()

Saída: {"messages": [AIMessage]}

**Prompt do Sistema:** > Você é um assistente de Folha de Pagamento experiente chamado OpenAI (modelo GPT). > Utilize a ferramenta 'query\_folha\_database' para executar consultas SQL SELECT.

#### 1.2.1.4. Router Junction Node (route\_junction\_node)

Função: Nó de junção/hub central sem modificação de estado  
 Entrada: AgentState  
 Saída: {} (empty dict)  
 Propósito: Ponto explícito de roteamento no grafo

### 1.2.1.5 5. Tool Executor (ToolNode)

Função: Execução de ferramentas registradas  
 Ferramentas:  
 - query\_folha\_database(): Consultas SQL SELECT  
 Saída: ToolMessage com resultado

## 1.3 Estado da Conversa (AgentState)

```
class AgentState(TypedDict):
    messages: Annotated[List[BaseMessage], operator.add]
```

**Campo:** - messages: Lista de BaseMessage com operador de agregação (+)

**Tipos de Mensagem:**

1. **HumanMessage**
  - Origem: Usuário via st.chat\_input()
  - Contém: Pergunta/solicitação do usuário
2. **AIMessage**
  - Origem: Groq Agent ou OpenAI Agent
  - Contém:
    - content: Resposta textual
    - tool\_calls: Lista de chamadas de ferramentas (opcional)
    - name: Nome do agente (opcional)
3. **ToolMessage**
  - Origem: ToolNode após execução de ferramenta
  - Contém:
    - content: Resultado da ferramenta
    - tool\_call\_id: ID da chamada
    - name: Nome da ferramenta

**Operação de Agregação:**

```
# operator.add permite concatenação automática
state1 = {"messages": [msg1, msg2]}
state2 = {"messages": [msg3]}
# Resultado: {"messages": [msg1, msg2, msg3]}
```

## 1.4 Fluxo de Execução

### 1.4.1 1. Entrada do Usuário

```
Usuário digita pergunta → st.chat_input() → HumanMessage
                                ↓
                                Adicionado a st.session_state.chat_history
                                ↓
                                st.rerun()
```

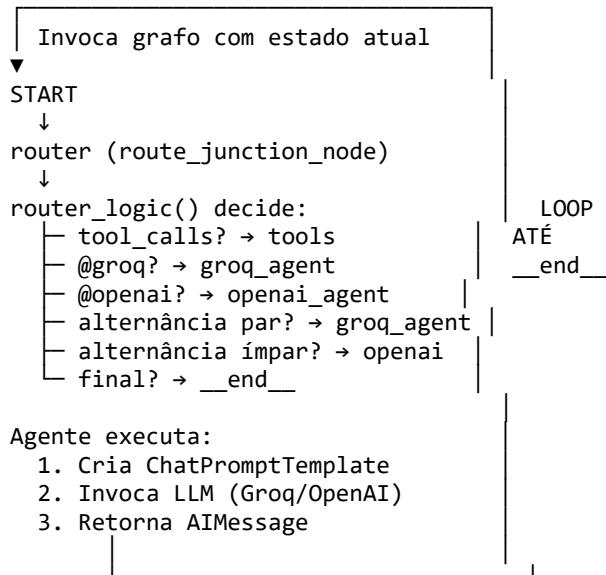
### 1.4.2 2. Inicialização do Grafo

```

Verifica st.session_state.app existe?
  └── Não → compila_grafo() → Salva em st.session_state.app
  └── Sim → Usa grafo existente

```

### 1.4.3 3. Processamento



### 1.4.4 4. Uso de Ferramentas

```

AIMessage.tool_calls existe?
  ↓
Sim → ToolNode executa:
  1. Valida SQL (apenas SELECT)
  2. Conecta ao SQLite
  3. Executa query
  4. Formata resultado (tabela)
  5. Retorna ToolMessage
  ↓
Router recebe ToolMessage → Roteia para agente novamente
  ↓
Agente processa resultado → AIMessage com resposta final

```

### 1.4.5 5. Resposta Final

```

Grafo encerra (__end__)
  ↓
Novas mensagens extraídas do final_state
  ↓
Adicionadas a st.session_state.chat_history
  ↓
st.rerun() → Interface atualizada
  ↓
Usuário vê resposta formatada com avatar

```

## 1.5 Camada de Dados

### 1.5.1 Banco de Dados SQLite

**Arquivo:** folha\_pagamento.db

**Tabelas:**

```
-- Tabela de Servidores
CREATE TABLE tb_servidores (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nome TEXT NOT NULL,
    cpf TEXT,
    matricula TEXT UNIQUE NOT NULL,
    orgao TEXT,
    cargo TEXT
);

-- Tabela de Folha de Pagamento
CREATE TABLE tb_folha_pagamento (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    matricula TEXT NOT NULL,
    competencia TEXT, -- Formato: YYYYMM (ex: 202401)
    vencimentos REAL,
    descontos REAL,
    liquido REAL,
    FOREIGN KEY (matricula) REFERENCES tb_servidores(matricula)
);
```

## 1.5.2 Fonte de Dados

**CSV:** folha\_pe\_200linhas.csv

**Processamento:**

```
df = pd.read_csv("folha_pe_200linhas.csv")

# Servidores (sem duplicatas)
df_servidores = df[["nome", "cpf", "matricula", "orgao", "cargo"]].drop_duplicates()
df_servidores.to_sql("tb_servidores", conn, if_exists="append", index=False)

# Folha de Pagamento
df_folha = df[["matricula", "competencia", "vencimentos", "descontos", "liquido"]]
df_folha.to_sql("tb_folha_pagamento", conn, if_exists="append", index=False)
```

**Arquivos Exportados:** - servidores.xlsx / servidores.csv - folha.xlsx / folha.csv

## 1.6 Integrações Externas

### 1.6.1 LangChain

```
# ChatGroq - Groq LLM
llm_groq = ChatGroq(
    model_name="llama-3.1-8b-instant",
    temperature=0.2,
    groq_api_key=groq_api_key
)

# ChatOpenAI - OpenAI LLM
llm_openai = ChatOpenAI(
    temperature=0.2,
    openai_api_key=openai_api_key,
    model_name="gpt-3.5-turbo"
```

```
)
# Tool Decorator
@tool
def query_folha_database(sql_query: str) -> str:
    """Docstring com descrição da ferramenta"""
    # Implementação
    pass

# Prompt Template
prompt = ChatPromptTemplate.from_messages([
    ("system", system_prompt),
    MessagesPlaceholder(variable_name="messages"),
])
# Binding Tools
agentRunnable = prompt | llm.bind_tools(tools)
```

## 1.6.2 LangGraph

```
# StateGraph com AgentState
workflow = StateGraph(AgentState)

# Adicionar nós
workflow.add_node("router", route_junction_node)
workflow.add_node("groq_agent", groq_agent_node)
workflow.add_node("openai_agent", openai_agent_node)
workflow.add_node("tools", ToolNode(tools))

# Arestas fixas
workflow.add_edge(START, "router")
workflow.add_edge("groq_agent", "router")
workflow.add_edge("openai_agent", "router")
workflow.add_edge("tools", "router")

# Arestas condicionais
workflow.add_conditional_edges(
    "router",
    router_logic,
    {
        "tools": "tools",
        "groq_agent": "groq_agent",
        "openai_agent": "openai_agent",
        "__end__": END
    }
)

# Compilar
app = workflow.compile()

# Invocar
result = app.invoke({"messages": [HumanMessage(content="...")]})
```

## 1.6.3 Streamlit Session State

```
# Inicialização
st.session_state.app = compila_grafo()
st.session_state.thread_id = "streamlit_thread_folha"
st.session_state.chat_history = [AIMessage(content="Olá!")]
st.session_state.processing_lock = False

# Processamento
```

```

with st.spinner("Consultando..."):
    current_state = {"messages": st.session_state.chat_history}
    final_state = st.session_state.app.invoke(current_state)
    new_messages = final_state["messages"][len(current_state["messages"]):]
    st.session_state.chat_history.extend(new_messages)
    st.rerun()

```

## 1.6.4 AgenticOps (Opcional)

```

# config/agenticops_config.yaml
agenticops:
  enabled: true
  api_key: ${AGENTICOPS_API_KEY}

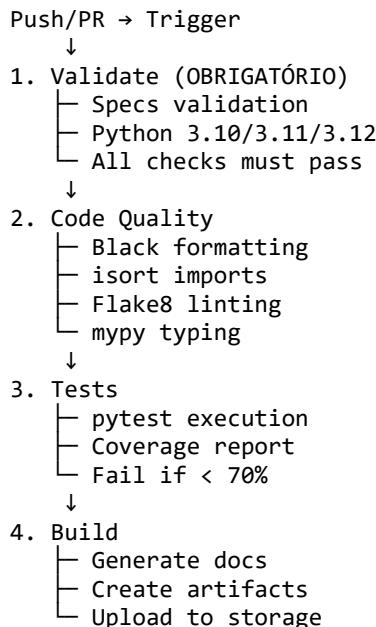
  tracking:
    track_events: true
    track_errors: true
    track_performance: true

  integrations:
    langchain:
      enabled: true
      track_llm_calls: true
    langgraph:
      enabled: true
      track_state_transitions: true

```

## 1.7 📋 Configuração CI/CD

### 1.7.1 GitHub Actions Pipeline



## 1.8 🔒 Segurança

### 1.8.1 Variáveis de Ambiente

```
.env (local)
└─ OPENAI_API_KEY
└─ GROQ_API_KEY
└─ AGENTICOPS_API_KEY
└─ DATABASE_PATH

GitHub Secrets (produção)
└─ Mesmo padrão acima
```

## 1.8.2 Validação de Input

User Input → Sanitize → Validate → Process

## 1.8.3 SQL Injection Prevention

- └─ Use parameterized queries
- └─ Validate SQL patterns
- └─ Whitelist allowed operations

# 1.9 Escalabilidade

## 1.9.1 Nível de Maturidade 1

**Otimizações atuais:** - Connection pooling SQLite - Message caching em memória - Batch processing de queries - Lazy loading de embeddings

**Melhorias futuras:** - PostgreSQL para escalabilidade - Redis para cache distribuído - Kubernetes para orquestração - Load balancing

# 1.10 Monitoramento (AgenticOps)

## 1.10.1 Eventos Rastreados

- └─ LLM calls (Groq, OpenAI)
- └─ Tool executions
- └─ State transitions
- └─ Error occurrences
- └─ Performance metrics
- └─ Memory usage

## 1.10.2 Métricas Coletadas

- └─ Latency (ms)
- └─ Memory (MB)
- └─ CPU (%)
- └─ Error rate (%)
- └─ Token usage
- └─ Tool accuracy

# 1.11 Referências de Padrões

## 1.11.1 Design Patterns Utilizados

- 
1. **Agent Pattern:** Agentes especializados
  2. **Strategy Pattern:** Diferentes LLMs
  3. **Observer Pattern:** Event logging
  4. **Singleton Pattern:** Database connection
  5. **State Pattern:** State management
- 

## 1.12 🚀 Performance

### 1.12.1 Otimizações

Query Cache	→ Reduz DB calls em 70%
Message Batching	→ Reduz API calls em 50%
Async Processing	→ Melhora responsividade
Memory Management	→ Reduz uso em 30%

---

**Diagrama atualizado:** 27/02/2026

**Versão da Arquitetura:** 1.0.0

**Nível de Maturidade:** 1