Readme

- 1 FACIN_IA - Documentação do Projeto
  - 1.1 📋 Visão Geral
    - 1.1.1 Aplicação Principal
  - 1.2 🎯 Objetivos
  - 1.3 📁 Estrutura do Projeto
  - 1.4 🚀 Quick Start
    - 1.4.1 1. Pré-requisitos
    - 1.4.2 2. Configuração do Ambiente
    - 1.4.3 3. Criar Banco de Dados
    - 1.4.4 4. Executar Aplicação
  - 1.5 🤖 Exemplos de Consultas
  - 1.6 🔧 Sistema Multi-Agentes
    - 1.6.1 Arquitetura LangGraph
    - 1.6.2 Estado Compartilhado (AgentState)
    - 1.6.3 Ferramentas
  - 1.7 📊 Monitoramento com AgenticOps
    - 1.7.1 Nível de Maturidade 1
    - 1.7.2 Variáveis de Ambiente
  - 1.8 🔄 CI/CD com GitHub Actions
    - 1.8.1 Validação Automática (OBRIGATÓRIA)
    - 1.8.2 Executar Localmente
  - 1.9 📝 Documentação por Módulo
    - 1.9.1 API Reference
    - 1.9.2 Arquitetura
    - 1.9.3 Guia de Contribuição
  - 1.10 🐛 Rastreamento de Erros
  - 1.11 📦 Dependências Principais
  - 1.12 🤝 Contribuindo
  - 1.13 📄 Licença
  - 1.14 👤 Autor
  - 1.15 📞 Suporte

1 FACIN_IA - Documentação do Projeto

1.1 📋 Visão Geral

FACIN_IA é um Sistema Inteligente de Gerenciamento Multi-Agentes baseado em: - **LangChain** + **LangGraph**: Orquestração de multi-agentes de IA - **Streamlit**: Interface web interativa - **Groq** + **OpenAI**: Modelos de linguagem - **SQLite/Access DB**: Persistência de dados - **AgenticOps**: Monitoramento e observabilidade

1.1.1 Aplicação Principal

Sistema de automação de folha de pagamento com: - Consultas inteligentes ao banco de dados - Processamento de dados de servidores - Análise de remuneração e cargo - Gerenciamento de contexto e memória

---

1.2 🎯 Objetivos

1. ✅ Automação da folha de pagamento
2. ✅ Sistema multi-agentes inteligente
3. ✅ Interface web responsiva
4. ✅ Gerenciamento de memória e contexto
5. ✅ Monitoramento com AgenticOps
6. ✅ CI/CD com validação obrigatória

1.3 📁 Estrutura do Projeto

```
FACIN_IA/
├── app.py                      # Aplicação principal Streamlit
├── cria_db.py                  # Script de criação do banco de dados
├── requirements.txt            # Dependências Python
├── config/
│   ├── agenticops_config.yaml # Configuração AgenticOps
│   └── vscode_copilot_settings.json
├── .github/
│   └── workflows/
│       └── validate.yml        # Pipeline CI/CD
├── .vscode/
│   ├── settings.json          # Configurações do VS Code
│   └── extensions.json        # Extensões recomendadas
├── docs/                       # Documentação do projeto
│   ├── API.md
│   ├── ARCHITECTURE.md
│   ├── QUICKSTART.md
│   └── generated/              # Documentação gerada automaticamente
├── errors/                     # Rastreamento de erros e soluções
│   └── ERRORS_LOG.md
├── tests/                      # Testes unitários
│   ├── test_app.py
│   └── test_database.py
├── logs/                       # Logs da aplicação
└── data/                       # Dados e arquivos CSV
```

1.4 🚀 Quick Start

1.4.1 1. Pré-requisitos

- Python 3.10+
- pip ou conda
- Git
- Chaves API: OpenAI, Groq, AgenticOps (opcional)

1.4.2 2. Configuração do Ambiente

```bash
# Criar ambiente virtual
python -m venv .venv

# Ativar ambiente
# Windows
.venv\Scripts\activate
# macOS/Linux
```

```
source .venv/bin/activate

# Instalar dependências
pip install -r requirements.txt

# Configurar variáveis de ambiente
cp .env.example .env
# Editar .env com suas chaves API
```

1.4.3 3. Criar Banco de Dados

```
python cria_db.py
```

1.4.4 4. Executar Aplicação

```
streamlit run app.py
```

A aplicação estará disponível em: `http://localhost:8501`

1.5 🤖 Exemplos de Consultas

```
"Quais servidores estão ativos?"
"Quantos servidores estão ativos?"
"Qual é a remuneração do Servidor 2?"
"Quantos servidores ocupam o cargo de Assistente?"
"Quantos servidores são da Secretaria da Saúde?"
"Na Fazenda, quantos servidores ocupam o cargo de Assistente?"
"Quantos servidores tiveram aumento?"
"Algum servidores foram demitidos? Quais?"
```

1.6 🔧 Sistema Multi-Agentes

1.6.1 Arquitetura LangGraph

O projeto utiliza **LangGraph** para orquestração de agentes alternados:

Nós do Grafo:

1. **router (route_junction_node)**: Hub central de roteamento
2. **groq_agent**: Agente Groq com Llama 3.1 (temp=0.2)
3. **openai_agent**: Agente OpenAI com GPT-3.5 (temp=0.2)
4. **tools**: Executor de ferramentas (@tool decorator)

Lógica de Roteamento (router_logic):

```
# Decisões:
1. Se AIMessage.tool_calls → "tools"
2. Se AIMessage sem tools → "__end__"
3. Se "@groq" na mensagem → "groq_agent"
4. Se "@openai" na mensagem → "openai_agent"
5. Alternância automática:
   - Mensagens AI pares → Groq
   - Mensagens AI ímpares → OpenAI
```

1.6.2 Estado Compartilhado (AgentState)

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

Tipos de Mensagem: - `HumanMessage`: Entrada do usuário - `AIMessage`: Resposta dos agentes (com tool_calls opcional) - `ToolMessage`: Resultado de ferramentas executadas

1.6.3 Ferramentas

query_folha_database(sql_query: str) - Executa **apenas SELECT** no SQLite - Validação de segurança automática - Formata resultados em tabela (max 15 linhas) - Acessa: `tb_servidores`, `tb_folha_pagamento`

1.7 📊 Monitoramento com AgenticOps

1.7.1 Nível de Maturidade 1

Configuração básica inclui: - ✅ Logging de eventos - ✅ Rastreamento de execução - ✅ Captura de erros - ✅ Métricas de performance - ✅ Integração LangChain/LangGraph

1.7.2 Variáveis de Ambiente

```
export AGENTICOPS_API_KEY=your_api_key
export OPENAI_API_KEY=your_openai_key
export GROQ_API_KEY=your_groq_key
```

1.8 🔄 CI/CD com GitHub Actions

1.8.1 Validação Automática (OBRIGATÓRIA)

A pipeline executa: 1. **Formatação**: Black (estilo código) 2. **Imports**: isort (organização) 3. **Linting**: Flake8 (qualidade) 4. **Type Checking**: mypy (tipos) 5. **Testes**: pytest (cobertura) 6. **Validação de Spec**: JSON schemas (OBRIGATÓRIO)

1.8.2 Executar Localmente

```
# Formatar código
black . --line-length=100

# Verificar formatação
black --check .

# Executar testes
pytest -v --cov=.
```

1.9 📝 Documentação por Módulo

1.9.1 API Reference

Referência completa de funções e classes.

1.9.2 Arquitetura

Diagrama e explicação da arquitetura do sistema.

1.9.3 Guia de Contribuição

Como contribuir para o projeto.

1.10 🐛 Rastreamento de Erros

Veja errors/ERRORS_LOG.md para: - Erros encontrados - Soluções implementadas - Status de resolução - Versão corrigida

---

1.11 📦 Dependências Principais

```
langchain==0.3.24
langchain-community==0.3.23
langchain-groq==0.3.2
langchain-openai==0.3.14
langgraph==0.3.34
streamlit==1.28.0
openai==1.76.0
groq==0.23.1
```

Veja requirements.txt para lista completa.

1.12 🤝 Contribuindo

1. Faça um Fork do projeto
2. Crie uma branch para sua feature (`git checkout -b feature/AmazingFeature`)
3. Commit seus mudanças (`git commit -m 'Add some AmazingFeature'`)
4. Push para a branch (`git push origin feature/AmazingFeature`)
5. Abra um Pull Request

1.13 📄 Licença

Este projeto está sob a licença MIT.

1.14 👤 Autor

Gustavo Passos - GitHub: [@gutpassos](https://github.com/gutpassos) - Email: gut.passos@gmail.com - Projeto: https://github.com/gutpassos/FACIN_IA

1.15 📞 Suporte

Para dúvidas ou problemas: 1. Abra uma issue no GitHub 2. Verifique a seção de erros conhecidos 3. Consulte a documentação técnica

---

Versão: 1.0.0
Nível de Maturidade: 1
Data de Atualização: 27/02/2026