SAPIEN - Sistema de Agente para Ensino e Pesquisa

Documentação Técnica Completa - 19/09/2025

ÍNDICE

1. VISÃO GERAL DO PROJETO

O SAPIEN é um sistema multi-agente inteligente desenvolvido para automatizar pesquisas científicas e educacionais. O sistema utiliza uma arquitetura baseada em agentes especializados que trabalham em conjunto para coletar, processar, validar e armazenar informações de fontes acadêmicas e web.

Principais Características:

• Sistema multi-agente com supervisão inteligente

• Busca automática em arXiv e web científica

• Processamento de linguagem natural (NLP)

• Validação semântica com embeddings

• Armazenamento vetorial com ChromaDB

• Interface web interativa

• Agendamento de pesquisas periódicas

2. ARQUITETURA E ESTRUTURA

Estrutura de Diretórios:

Diretório	Descrição
app/	Aplicação principal Flask
app/core/	Núcleo do sistema multi-agente
app/core/tools/	Ferramentas especializadas dos agentes
app/static/	Arquivos estáticos (CSS, JS, imagens)
app/templates/	Templates HTML
chroma_db/	Banco de dados vetorial ChromaDB
venv/	Ambiente virtual Python
requirements.txt	Dependências do projeto
run.py	Ponto de entrada da aplicação

3. ANÁLISE DETALHADA DOS ARQUIVOS

3.1 run.py - Ponto de Entrada

Arquivo principal que inicializa a aplicação Flask. Cria a instância da aplicação e executa o servidor em modo debug.

Código:

```
from app import create_app
app = create_app()
if __name__ == '__main__':
    app.run(debug=True)
```

3.2 app/__init__.py - Factory da Aplicação

Implementa o padrão Factory para criação da aplicação Flask. Registra os blueprints e configura a estrutura modular da aplicação.

Código:

```
from flask import Flask
def create_app():
    app = Flask(__name__)
    from .routes import routes_bp
    app.register_blueprint(routes_bp)
    return app
```

3.3 app/routes.py - Rotas da Aplicação

Define as rotas da aplicação web. Inclui a rota principal (/) que renderiza a interface e a rota /chat que processa as mensagens do usuário através do sistema multi-agente.

Funcionalidades:

• Rota GET /: Renderiza a interface principal

• Rota POST /chat: Processa mensagens e retorna respostas dos agentes

3.4 app/core/config.py - Configuração Central

Arquivo de configuração central que gerencia todas as dependências e configurações do sistema. Inclui inicialização do LLM, embeddings, banco vetorial e scheduler.

Componentes Principais:

• Inicialização do modelo Claude 3.5 Sonnet com retry automático

• Configuração de embeddings HuggingFace (all-MiniLM-L6-v2)

• Inicialização do ChromaDB para armazenamento vetorial

• Scheduler APScheduler para tarefas periódicas

• Caches globais para controle de duplicatas

3.5 app/core/agents.py - Sistema Multi-Agente

Núcleo do sistema multi-agente. Define todos os agentes especializados e o supervisor que coordena suas atividades. Cada agente tem responsabilidades específicas no pipeline de processamento.

Agentes Especializados:

• tavily_agent: Busca web com foco em fontes científicas

• arxiv_agent: Pesquisa específica no arXiv

• sched_agent: Agendamento de pesquisas periódicas

• nlp_agent: Processamento de linguagem natural

• validation_agent: Validação semântica com embeddings

• chromadb_agent: Armazenamento vetorial

3.6 app/core/services.py - Serviços de Processamento

Implementa o grafo principal de processamento usando LangGraph. Gerencia o fluxo de mensagens entre o usuário e o sistema multi-agente, incluindo controle de threads e

processamento de respostas.

Funcionalidades:

• Criação de threads únicos para cada conversa

• Processamento streaming das respostas dos agentes

• Filtragem e seleção das melhores respostas

3.7 app/core/shared_state.py - Estado Compartilhado

Gerencia o estado compartilhado entre os agentes do pipeline. Permite que os dados processados por um agente sejam acessados pelos agentes subsequentes no fluxo.

Funcionalidades:

• Armazenamento temporário de dados processados

• Controle de hashes para evitar duplicatas

• Gerenciamento de resultados do scheduler

4. FLUXO DE PROCESSAMENTO

O sistema implementa um pipeline padronizado que toda informação coletada deve seguir:

Etapa	Agente Responsável	Descrição
1. Coleta	tavily_agent / arxiv_agent	Busca informações em fontes web ou arXiv
2. NLP	nlp_agent	Processamento linguístico e normalização
3. Validação	validation_agent	Validação semântica com embeddings
4. Armazenamento	chromadb_agent	Indexação vetorial no ChromaDB

5. AGENTES E FERRAMENTAS

5.1 Ferramentas de Busca

web_search_with_flow.py:

Implementa busca web usando Tavily Search com foco em fontes científicas. Automaticamente processa resultados através do pipeline completo (NLP → Validação → ChromaDB).

simple_arxiv_search.py:

Busca artigos no arXiv usando a API oficial. Inclui wrapper resiliente que aceita diferentes formatos de entrada e processa automaticamente através do pipeline.

5.2 Ferramentas de Processamento

nlp_process.py:

Processa conteúdo através de normalização linguística, extração de metadados e preparação para indexação. Gera hashes únicos para controle de duplicatas.

validate_content.py:

Valida conteúdo usando similaridade semântica com embeddings. Calcula similaridade de cosseno entre texto e tópico de interesse para determinar relevância.

store_in_chromadb.py:

Armazena conteúdo validado no ChromaDB com embeddings vetoriais. Divide texto em chunks otimizados para busca semântica.

5.3 Ferramentas de Agendamento

sheduler_tools.py:

Implementa sistema de agendamento de pesquisas periódicas. Permite agendar pesquisas automáticas com duração e intervalo configuráveis.

Funcionalidades:

• schedule_research: Agenda pesquisas periódicas

• cancel_research: Cancela pesquisas ativas

• check_scheduler_results: Verifica resultados agendados

6. INTERFACE WEB

Interface web moderna desenvolvida com Flask, Bootstrap 5 e JavaScript. Fornece uma experiência de chat interativa para interação com o sistema multi-agente.

Componentes:

• index.html: Template principal com interface de chat

• style.css: Estilos customizados para interface moderna

• chat.js: JavaScript para interação em tempo real

7. DEPENDÊNCIAS E CONFIGURAÇÃO

Principais Dependências:

Biblioteca	Versão	Propósito
flask	latest	Framework web
langchain	latest	Framework para LLMs
langgraph	latest	Grafos de agentes
chromadb	latest	Banco vetorial
sentence-transformers	latest	Embeddings
langchain-anthropic	latest	Integração Claude
langchain-tavily	latest	Busca web
apscheduler	3.11.0	Agendamento
arxiv	latest	API arXiv

Configuração de Ambiente:

O sistema requer as seguintes variáveis de ambiente:

• TAVILY_API_KEY: Chave da API Tavily para busca web

• ANTHROPIC_API_KEY: Chave da API Anthropic para Claude

CONCLUSÃO

O SAPIEN representa uma implementação sofisticada de um sistema multi-agente para pesquisa científica automatizada. A arquitetura modular permite fácil extensão e manutenção, enquanto o pipeline padronizado garante consistência na qualidade dos dados processados. O sistema combina tecnologias de ponta em IA, processamento de linguagem natural e busca semântica para criar uma ferramenta poderosa para pesquisadores e educadores.