EKKO

Dashboard que promove a agricultura de precisão

Documentação Técnica do Backend

Tecnologias, Arquitetura e Implementação

Instituição:	Escola Técnica Estadual Ferreira Master Cel (ETE FMC)
Curso:	Desenvolvimento de Sistemas
Equipe:	34DS08
Evento:	45ª Projete - Fraternidade e Ecologia Integral
Área:	Tecnologia da Informação e Agricultura de Precisão

Documento gerado em 07 de Outubro de 2025

"A agricultura não é apenas sobre cultivar alimentos,
é sobre cultivar o futuro."

Dedicado aos agricultores brasileiros que,
com sua dedicação e conhecimento,
alimentam nossa nação.

Índice

1. Visão Geral do Projeto

Objetivo

O projeto Ekko, desenvolvido pela equipe 3408, do curso ”Desenvolvimento de Sistemas“, alinhado à proposta da Projete de fraternidade e ecologia integral, consiste em um website no qual o agricultor pode acessar dados e feedbacks relacionados ao solo de sua região de plantio, junto à análises produzidas por IA, suporte de um ChatBot treinado com fontes renomadas do setor agrícola brasileiro, estatísticas, gráficos, mapas de calor e mais. Além disso, o grupo também desenvolveu uma simulação gamificada que permite ao usuário entender de que maneira o dispositivo móvel Ekko, ou 'carrinho', encarregado de coletar os valores do parâmetros do solo, como pH, umidade, temperatura, NPK e outros, funcionaria na prática, por meio da coleta e transmissão de dados em determinada região de plantio.

Componentes Principais

Componente	Tecnologia	Função
Frontend	HTML5, CSS3, JavaScript	Desenvolvimento do website
Backend	Python + FastAPI	API REST e processamento de dados
Banco de Dados	MongoDB Atlas	Armazenamento de dados na nuvem
Chatbot	Llama 3.2 + Ollama	Assistente agrícola inteligente
Análise de Solo	Python + IA	Avaliação de 12 parâmetros do solo
Busca RAG	FAISS + Transformers	Recuperação de conhecimento agrícola

2. Framework Web & API

FastAPI 0.104.1

FastAPI é um framework web moderno e de alta performance para construção de APIs com Python 3.7+, baseado em type hints padrão do Python.

Características Principais:

- **Performance:** Uma das mais rápidas disponíveis, comparável ao NodeJS e Go
- **Documentação Automática:** Gera automaticamente documentação interativa (Swagger UI)
- **Validação:** Validação automática de dados baseada em Python type hints
- **Async/Await:** Suporte nativo para programação assíncrona

Exemplo de Implementação:

```
@app.post("/unity/soil/save/{unity_id}")
def save_soil(unity_id: str, soil: SoilData):
    # Validação automática via Pydantic
    if not unity_profiles.find_one({"_id": unity_id}):
        raise HTTPException(status_code=404)

    # Processamento dos dados
    soil_doc = {
        "unity_id": unity_id,
        "timestamp": datetime.utcnow(),
        "soil_parameters": {...}
    }
    return unity_soil_data.insert_one(soil_doc)
```

Pydantic 2.5.0

Biblioteca para validação de dados usando Python type annotations. Garante que os dados recebidos estejam no formato correto.

Modelo de Dados do Solo:

```
class SoilData(BaseModel):
    session_id: str
    ph: float
    umidade: float
    temperatura: float
    salinidade: float
    condutividade: float
    nitrogenio: float
    fosforo: float
    potassio: float
    # Validação automática de tipos
```

3. Banco de Dados

MongoDB Atlas

Banco de dados NoSQL na nuvem que oferece escalabilidade automática, backup e replicação. Ideal para dados semi-estruturados como perfis de usuários e dados de sensores.

Estrutura das Collections:

Collection	Propósito	Documentos Típicos
Python_userData	Perfis dos agricultores	~100-1000 usuários
Unity_soilData	Dados de solo da simulação	~10000+ registros

Exemplo de Documento - Perfil do Usuário:

```
{
  "_id": "unity_bf87c29494e0",
  "dados_pessoais": {
    "nome": "João Silva",
    "email": "joao@fazenda.com",
    "telefone": "(35) 99999-9999"
  },
  "propriedade": {
    "nome": "Fazenda Esperança",
    "area_hectares": 120.5,
    "cultivos_principais": ["Café", "Milho"],
    "regiao": "Sul de Minas Gerais"
  },
  "unity_stats": {
    "total_sessions": 15,
    "best_score": 850
  }
}
```

SQLite - Memória Local

Usado para armazenar a memória de longo prazo do chatbot. Permite que o sistema lembre de informações específicas sobre cada usuário entre sessões.

4. Inteligência Artificial

Ollama + Llama 3.2

Ollama é uma ferramenta que permite executar modelos de linguagem grandes localmente. O **Llama 3.2** é um modelo de IA desenvolvido pela Meta, otimizado para conversação e tarefas de texto.

Vantagens da Execução Local:

- **Privacidade:** Dados não saem do servidor local
- **Custo:** Sem taxas por token/requisição
- **Controle:** Parâmetros totalmente configuráveis
- **Latência:** Respostas mais rápidas (sem rede externa)

Configuração do Modelo:

```
MODEL_OPTIONS = {
    "num_predict": 500,      # Máximo de tokens por resposta
    "temperature": 0.7,       # Criatividade (0.0-1.0)
    "top_p": 0.9               # Consistência (0.0-1.0)
}

# Configuração de timeout e retry
STREAM_TIMEOUT = 120          # 2 minutos para streaming
MAX_RETRIES = 3               # Tentativas em caso de falha
```

Sentence Transformers

Biblioteca especializada em criar embeddings (representações vetoriais) de texto. Usado para converter documentos e consultas em vetores numéricos para busca semântica.

Modelo Utilizado: all-MiniLM-L6-v2

- **Dimensões:** 384 (cada texto vira um vetor de 384 números)
- **Idiomas:** Suporte a português e outros idiomas
- **Performance:** Balanceio entre qualidade e velocidade
- **Tamanho:** ~90MB (modelo compacto)

FAISS (Facebook AI Similarity Search)

Biblioteca otimizada para busca de similaridade em grandes volumes de vetores. Permite encontrar rapidamente os documentos mais relevantes para uma consulta.

Implementação da Busca RAG:

```
def local_database_search(query: str) -> str:
    # Converte a consulta em vetor
    query_embedding = embedding_model.encode([query])

    # Busca os 3 documentos mais similares
    _, indices = index.search(
        np.array(query_embedding, dtype=np.float32), 3
    )

    # Retorna os trechos encontrados
    context = "\n---\n".join([documents[i] for i in indices[0]])
    return f"Resultados: {context}"
```

5. Sistema de Análise de Solo

Parâmetros Analisados

Parâmetro	Faixa Ideal	Unidade	Importância
pH do Solo	6.0 - 7.0	-	Disponibilidade de nutrientes
Umidade	40 - 70	%	Absorção de água pelas plantas
Temperatura	20 - 30	°C	Atividade microbiana e raízes
Salinidade	≤ 600	ppm	Estresse osmótico das plantas
Condutividade	≤ 1.5	dS/m	Concentração de sais
Nitrogênio (N)	20 - 100	mg/kg	Crescimento vegetativo
Fósforo (P)	15 - 50	mg/kg	Desenvolvimento radicular
Potássio (K)	100 - 250	mg/kg	Resistência e qualidade
Drenagem	60 - 90	%	Aeração das raízes
Aeração	10 - 30	%	Respiração radicular
Compactação	≤ 2.0	g/cm³	Penetração das raízes
Ativ. Microbiana	≥ 50	mg CO2/kg	Ciclagem de nutrientes

Sistema de Classificação

Status	Descrição	Ação Recomendada
Ideal	Parâmetro dentro da faixa ótima	Manter práticas atuais
Atenção	Próximo do ideal, requer monitoramento	Ajustes preventivos
Crítico	Fora da faixa, ação imediata necessária	Correção urgente

Algoritmo de Análise:

```
def analyze_parameter(param_name, value):
    ranges = SOIL_RANGES[param_name]
    status = determine_status(param_name, value, ranges)

    return {
        "valor": value,
        "faixa_ideal": get_ideal_range_text(ranges),
        "status": status,
        "impacto": MESSAGES[param_name][status],
        "sugestao": get_suggestion(param_name, value, status)
    }
```

6. APIs Externas

INMET - Instituto Nacional de Meteorologia

API oficial do governo brasileiro para dados meteorológicos. Fornece previsões precisas e atualizadas para todo o território nacional.

Implementação:

```
def get_inmet_forecast(lat: float, lon: float) -> str:
    # Encontra estação mais próxima
    closest_station = find_closest_station(lat, lon)
    station_code = closest_station['CD_ESTACAO']

    # Consulta API do INMET
    url = f"https://apitempo.inmet.gov.br/previsao/{station_code}"
    response = requests.get(url, timeout=10)
    data = response.json()

    # Processa previsão de 5 dias
    return format_weather_forecast(data)
```

DuckDuckGo Search

Motor de busca que preserva privacidade, usado para buscar informações atualizadas em fontes confiáveis de agricultura brasileira.

Fontes Confiáveis Configuradas:

- site:embrapa.br - Empresa Brasileira de Pesquisa Agropecuária
- site:epamig.br - Empresa de Pesquisa Agropecuária de Minas Gerais
- site:noticiasagricolas.com.br - Portal de notícias do agronegócio
- site:canalrural.com.br - Canal Rural
- site:globorural.globo.com - Globo Rural
- site:cepea.esalq.usp.br - Centro de Estudos Avançados em Economia Aplicada

7. Endpoints da API

Método	Endpoint	Descrição	Uso
GET	/unity/status	Status da API e MongoDB	Health check
GET	/unity/login/{unity_id}	Login por Unity ID	Autenticação
POST	/unity/profile/create	Criar novo perfil	Registro
POST	/unity/soil/save/{unity_id}	Salvar dados de solo	Simulação
GET	/unity/dashboard/{unity_id}	Dashboard completo	Interface
GET	/unity/monitoring/{unity_id}	Monitoramento tempo real	Gráficos
GET	/unity/analise-ia/{unity_id}	Análise IA completa	Diagnóstico
POST	/api/chat/{unity_id}	Chat com streaming	Chatbot
POST	/api/generate_title	Gerar título conversa	UI Chat
POST	/api/soil-tips/{unity_id}	Dicas de melhoria	Sugestões

Exemplo de Resposta - Análise IA:


```
{
  "status": "success",
  "unity_id": "unity_bf87c29494e0",
  "diagnostico": {
    "saude_geral": 78.5,
    "alertas_criticos": [
      "pH crítico: 5.2",
      "Nitrogênio baixo: 15 mg/kg"
    ],
    "parametros": {
      "ph": {
        "valor": 5.2,
        "status": "critico",
        "sugestao": "Aplicar calcário urgentemente"
      }
    },
    "previsao_colheita": "Produtividade estimada: 520 toneladas"
  }
}
```

8. Performance e Otimização

Estratégias de Performance

Técnica	Implementação	Benefício
Async/Await	FastAPI nativo	I/O não bloqueante
Streaming	Server-Sent Events	Respostas em tempo real
Caching	Embeddings pré-computados	Busca RAG mais rápida
Connection Pooling	PyMongo com pool	Reutilização de conexões
Retry Logic	Backoff exponencial	Resiliência a falhas
Indexação FAISS	Vetores em memória	Busca O(log n)
Validação Pydantic	Type hints Python	Validação eficiente

Configuração de Retry com Backoff:

```
retry_strategy = Retry(
    total=3,                        # 3 tentativas
    backoff_factor=1,                # Delay progressivo
    status_forcelist=[429, 500, 502, 503, 504]  # Códigos para retry
)

# Timeouts configuráveis
STREAM_TIMEOUT = 120        # Streaming do chatbot
CONNECTION_TIMEOUT = 10      # Conexões HTTP
NON_STREAM_TIMEOUT = 60     # Requisições normais
```

9. Conclusão

Pontos Fortes da Arquitetura

- **Modularidade:** Código bem organizado em módulos especializados
- **Escalabilidade:** MongoDB Atlas permite crescimento automático
- **Performance:** FastAPI + async/await para alta concorrência
- **IA Local:** Ollama elimina custos e garante privacidade
- **Robustez:** Tratamento de erros e retry automático
- **Documentação:** Swagger UI automático para todos os endpoints

Impacto e Inovação

O projeto EKKO representa uma abordagem inovadora para a agricultura de precisão,combinando um dashboard que viabiliza o monitoramento em tempo real do solo e análises e recomendações ao agricultorembasadas em instituições renomadas de pesquisa agrícola. Demonstra maturidade no uso de ferramentas modernas comoFastAPI, MongoDB Atlas e modelos de IA locais, resultando em uma solução robusta, escalável eeconomicamente viável para os pequenos agricultores do país.

A integração do website com uma simulação 3D gamificada cria uma experiência de aprendizado, permitindo que o objetivo e funcionamento do projeto sejam facilmente compreendidos não só por agricultores e técnicos, mas também por qualquer um que esteja interessado, contribuindo para uma agricultura mais sustentável e produtiva no Brasil e para a democratização do conhecimento técnico.

Equipe de Desenvolvimento

Turma: 34DS08

Instituição: ETE FMC

Evento: 45ª Projete

Tema: Fraternidade e Ecologia Integral