

**PLANTAE**  
**Biodiversidade Vegetal para as Gerações Futuras**

Integrantes:

Matheus Matias  
Pedro Penteado  
Pedro Otavio

Orientador:  
Renato Cividini Matthiesen

Trabalho referente ao Projeto Interdisciplinar do 3º semestre do curso de Desenvolvimento de Software Multiplataforma desenvolvido na Faculdade de Tecnologia (FATEC) Araras nos componentes curriculares de Gestão Ágil de Projetos, Desenvolvimento WEB III e Banco de dados Não Relacional.

## Sumário

<b>1. Introdução</b>	<b>2</b>
<b>1.1. Visão geral do projeto</b>	<b>2</b>
<b>1.2. Objetivo</b>	<b>2</b>
<b>1.3. Público alvo</b>	<b>2</b>
<b>1.4. Requisitos</b>	<b>2</b>
<b>1.4.1. Funcionalidades Principais</b>	<b>2</b>
• Consulta Personalizada	2
• Inteligência de Dados	3
• Interface Intuitiva	3
<b>1.4.2. Requisitos Funcionais</b>	<b>3</b>
• Cadastro de Produtores	3
• Consulta de Informações	3
• Exibição de Resultados	3
• Filtragem e Classificação	3
<b>1.4.3. Requisitos Não Funcionais</b>	<b>3</b>
• Desempenho	3
• Compatibilidade	3
• Facilidade de Uso	4
• Escalabilidade	4
• Manutenção	4
• Documentação	4
<b>2. Diagramas</b>	<b>4</b>
2.1. Diagrama de atividades	4
2.2. Diagrama de caso de uso	4
<b>3. Design de Interface</b>	<b>5</b>
<b>4. Arquitetura</b>	<b>8</b>
<b>4.1. Tecnologias Utilizadas</b>	<b>8</b>
• Desenvolvimento Backend	8
• Banco de Dados	8
• Desenvolvimento Frontend	17
<b>5. Gestão do Projeto (SCRUM)</b>	<b>17</b>
5.1. Como foi gerenciado o projeto?	17
5.2. Stakeholders	17
5.3. Artefatos do SCRUM	18
<b>6. Conclusão</b>	<b>21</b>

## **1. Introdução**

### **1.1. Visão geral do projeto**

O projeto abrange o desenvolvimento de uma plataforma web que permitirá aos produtores agrícolas realizar consultas sobre culturas ideais para suas regiões. O escopo inclui a integração de APIs de dados climáticos e de solo, a implementação de algoritmos de filtragem e classificação, e a apresentação clara dos resultados em tabelas informativas.

### **1.2. Objetivo**

O propósito principal da plataforma é capacitar produtores agrícolas a fornecer informações precisas e personalizadas sobre as melhores plantas a serem cultivadas em suas regiões. A plataforma visa otimizar o processo de tomada de decisão dos agricultores, considerando fatores como clima, solo e condições adversas. Ao facilitar o acesso a dados confiáveis, o projeto busca contribuir para a disseminação de conhecimento sobre a diversidade de plantas e promover práticas agrícolas mais sustentáveis.

### **1.3. Público alvo**

O público-alvo principal são os produtores agrícolas de diversas escalas, desde pequenos agricultores familiares até grandes empreendimentos agrícolas. A plataforma visa atender usuários com diferentes níveis de conhecimento técnico, garantindo uma experiência de usuário intuitiva e acessível. Além dos produtores, a plataforma também pode interessar a pesquisadores, extensionistas agrícolas e demais profissionais ligados ao setor agrícola que buscam informações especializadas sobre culturas e práticas agrícolas sustentáveis.

### **1.4. Requisitos**

#### **1.4.1. Funcionalidades Principais**

- **Consulta Personalizada**

A plataforma oferece consultas personalizadas, considerando características únicas de cada região e preferências individuais dos agricultores.

- **Inteligência de Dados**  
Utiliza algoritmos avançados para filtrar e classificar as plantas mais adequadas, considerando sazonalidade, resistência a pragas e requisitos específicos de cultivo.
- **Interface Intuitiva**  
Apresenta uma interface de usuário intuitiva, garantindo acessibilidade a agricultores com diferentes níveis de habilidade tecnológica.

#### 1.4.2. Requisitos Funcionais

- **Cadastro de Produtores**  
Permite que produtores agrícolas se cadastrem na plataforma, coletando informações como nome, endereço e tipo de cultura cultivada.
- **Consulta de Informações**
  - Integrar APIs de dados climáticos e de solo para oferecer informações precisas.
  - Permite que produtores busquem informações sobre plantas adequadas para sua região com base em critérios como clima e solo.
- **Exibição de Resultados**  
Apresenta resultados de consultas em tabelas e descrições claras, incluindo detalhes sobre requisitos de cultivo e características das plantas.
- **Filtragem e Classificação**
  - Permite a filtragem de resultados com base em parâmetros como sazonalidade, resistência a pragas, etc.
  - Classifica as plantas de acordo com a adequação para a região.

#### 1.4.3. Requisitos Não Funcionais

- **Desempenho**  
Garante um desempenho eficiente, mesmo durante períodos de alta demanda, para oferecer uma experiência de usuário fluida.
- **Compatibilidade**  
Certifica-se de que a plataforma seja compatível com uma variedade de navegadores web para garantir uma ampla acessibilidade.

- Facilidade de Uso

Desenvolve uma interface intuitiva e de fácil navegação para atender usuários com diferentes níveis de habilidade tecnológica.

- Escalabilidade

Projetada para ser escalável, permitindo que a plataforma cresça para lidar com um aumento no número de usuários e dados.

- Manutenção

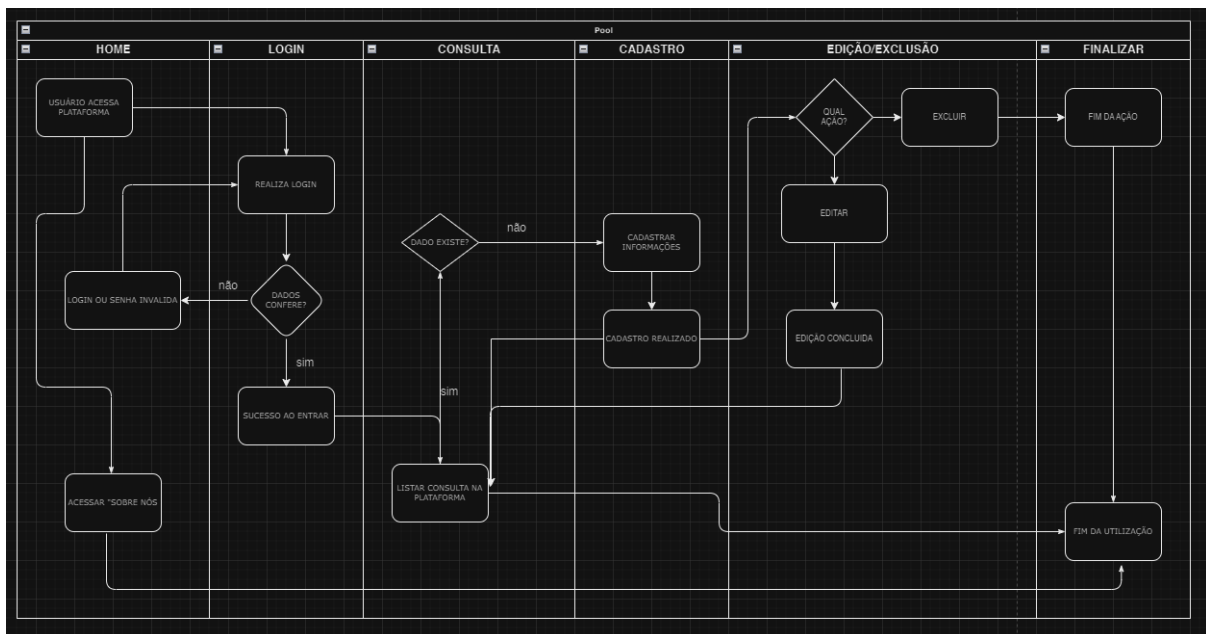
Facilita a manutenção da plataforma, garantindo atualizações regulares e eficientes para manter a relevância e eficácia ao longo do tempo.

- Documentação

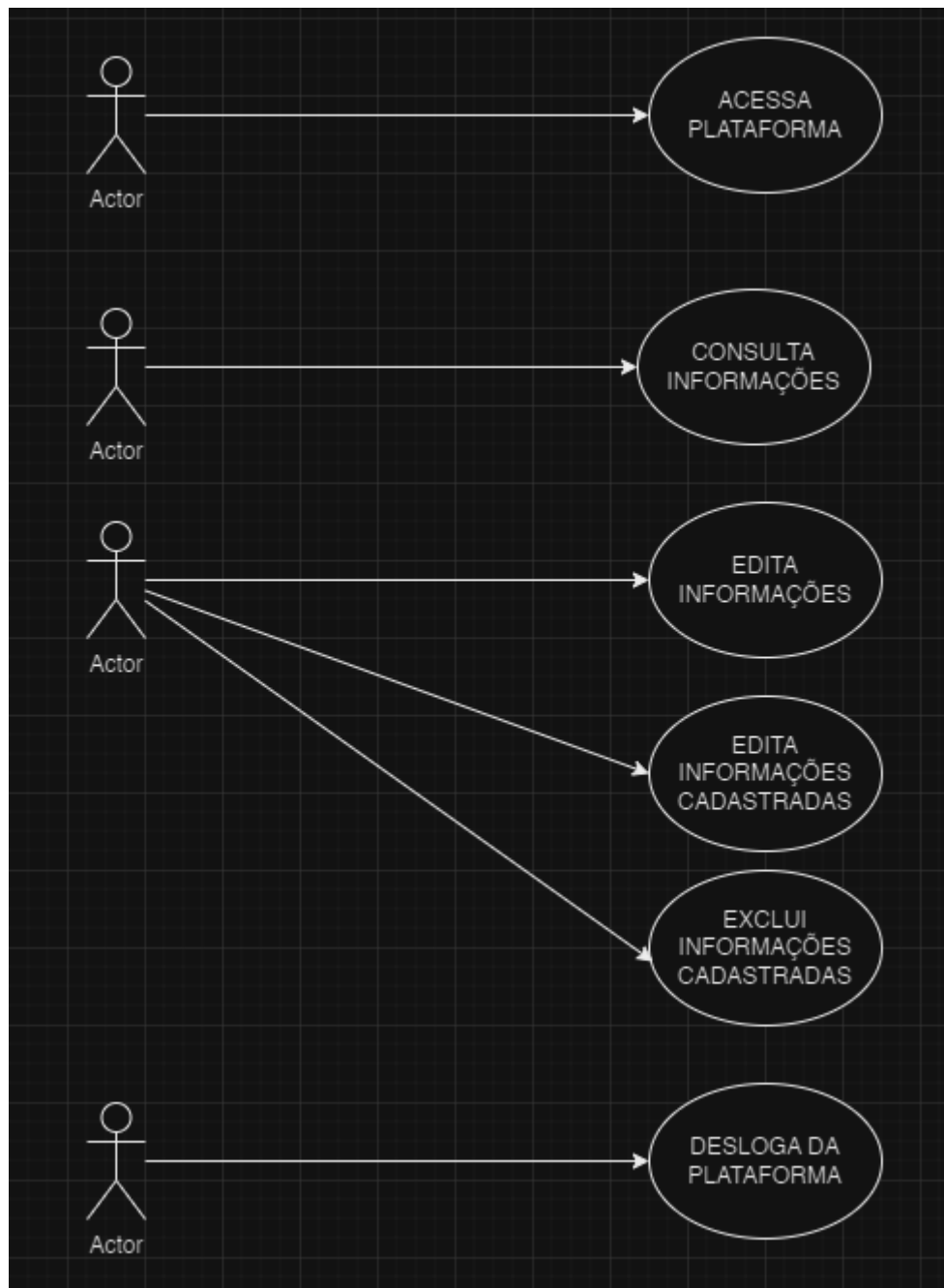
Cria documentação abrangente para desenvolvedores, administradores e usuários finais para facilitar a compreensão e a manutenção contínua do sistema.

## 2. Diagramas

### 2.1. Diagrama de atividades



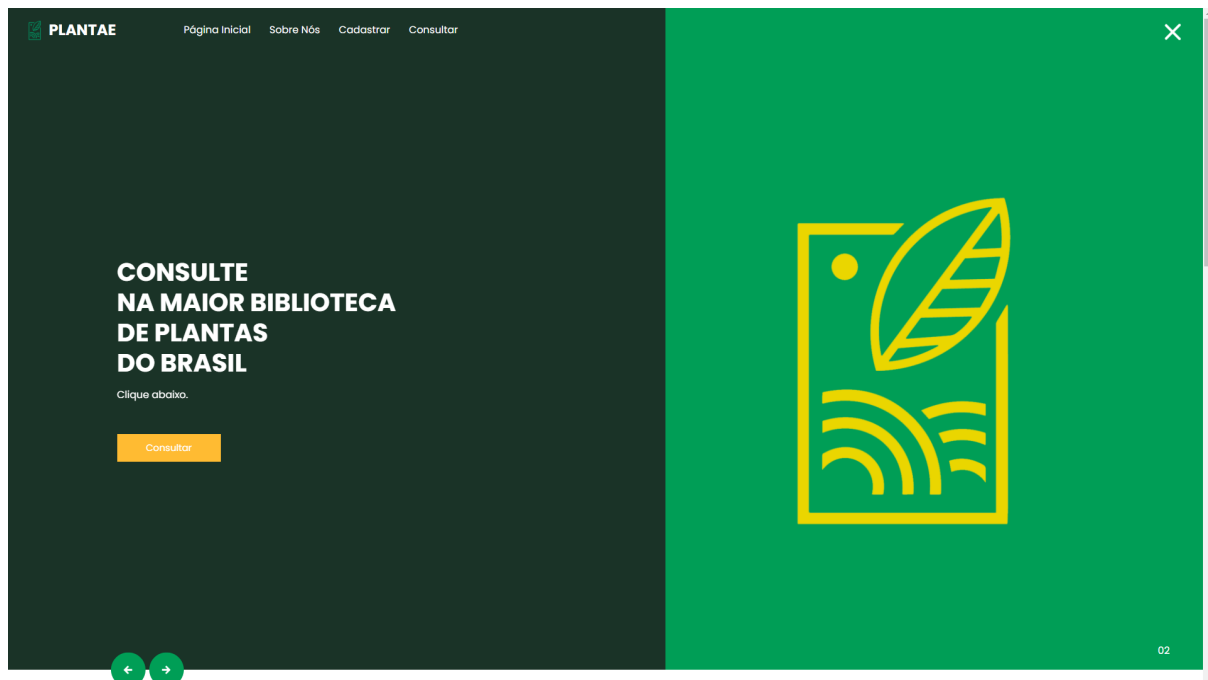
### 2.2. Diagrama de caso de uso



### 3. Design de Interface

O design da interface foi desenvolvido para ser o mais simples e intuitivo possível.

Na tela inicial da plataforma vemos no cabeçalho uma barra de navegação onde está contido todos os caminhos para as funcionalidades do sistema, tornando assim a experiência do usuário mais objetiva.



Contamos também com depoimentos de usuários que se cadastraram em nossa plataforma e utilizam frequentemente.

#### MAS, POR QUE UTILIZAR O PLANTAE?

-  **Totalmente Gratuito**  
**Utilizamos APIs livres**
-  **Melhor aproveitamento**  
**Sua plantação pode crescer de forma exponencial com conhecimento**
-  **98%**  
**Aprovado pelos maiores produtores rurais do Brasil**



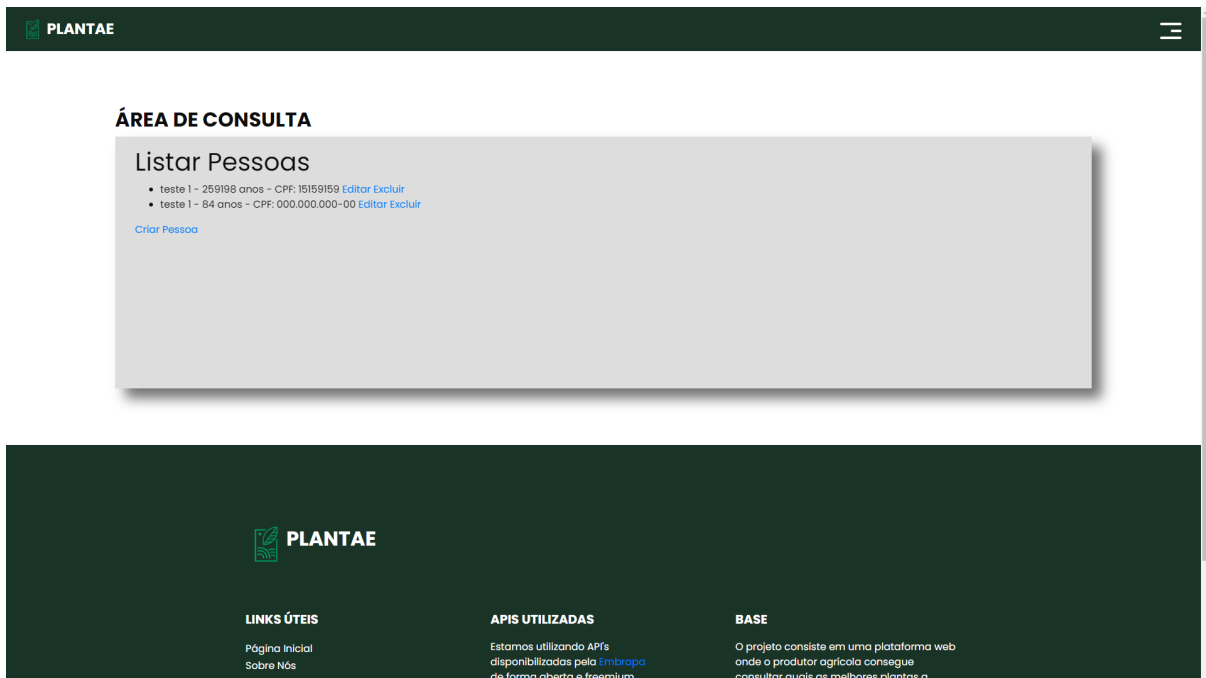
#### DEPOIMENTOS



Caso o usuário queira saber mais sobre a equipe, temos uma página de sobre nós onde ele poderá encontrar nossas informações e contatos.

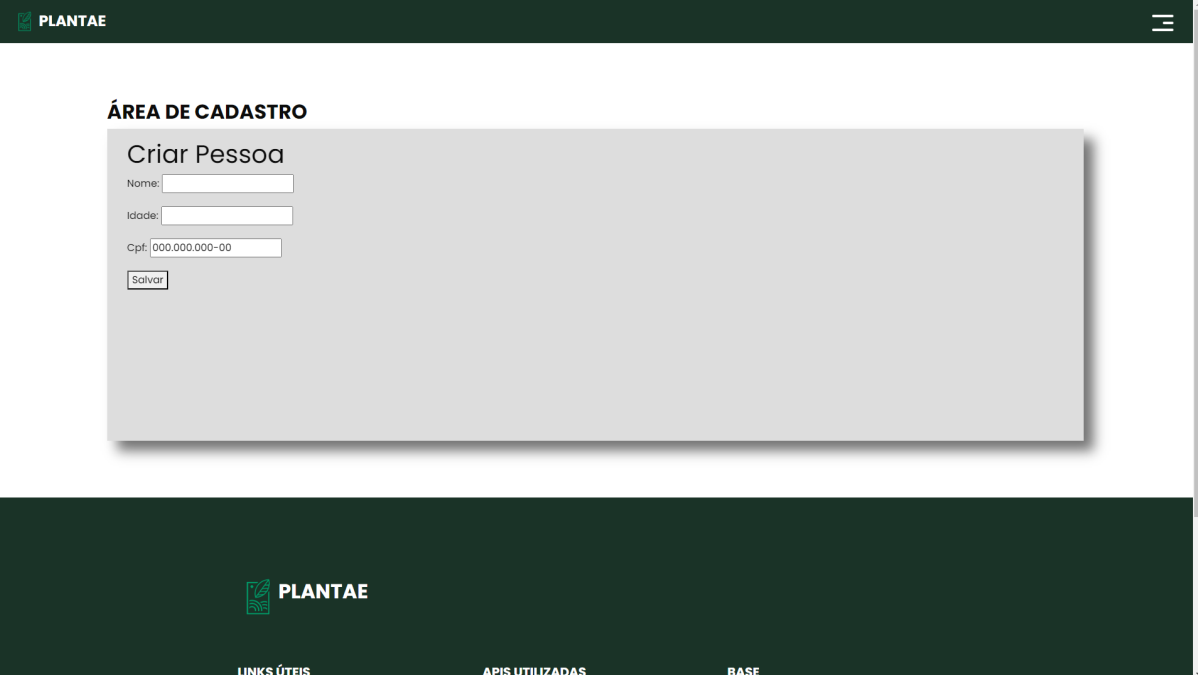


A principal tela é a de consulta de informações, onde estão sendo retornados os dados consultados pelos usuários.





Caso você seja um usuário autorizado poderá inserir mais informações sobre determinada semente ou até mesmo criá-la do zero para abastecer nosso banco de dados.



The screenshot displays the PLANTAE web application interface. At the top, a dark green header contains the PLANTAE logo on the left and a hamburger menu icon on the right. Below the header, the main content area is titled 'ÁREA DE CADASTRO' (Registration Area). Within this area, there is a light gray box titled 'Criar Pessoa' (Create Person). This box contains three input fields: 'Nome:' (Name), 'Idade:' (Age), and 'Cpf:' (CPF) with a placeholder '000.000.000-00'. Below these fields is a 'Salvar' (Save) button. At the bottom of the page, a dark green footer contains the PLANTAE logo on the left and three links: 'LINKS ÚTEIS' (Useful Links), 'APIS UTILIZADAS' (APIs Used), and 'BASE' (Database).

## 4. Arquitetura

### 4.1. Tecnologias Utilizadas

- Desenvolvimento Backend

Desenvolvido em Python utilizando o framework Django, proporcionando uma estrutura robusta e flexível para a manipulação de dados e a lógica de negócios da plataforma.

- Banco de Dados

O projeto oferece uma biblioteca de dados robustos sobre a biodiversidade de plantas de todo o território nacional e para conseguir realizar o armazenamento destes dados utilizamos o MongoDB em documentos com esquema chave-valor. O intuito da utilização prática do MongoDB é a sua velocidade de retorno das consultas realizadas pelo usuário na plataforma.

No esquema do projeto não será necessário o cadastro de usuários na plataforma, portanto toda a estrutura é voltada ao CRUD das

informações de plantas e sementes, com o usuário podendo editar, excluir e até mesmo criar novas informações sobre plantas inexistentes no banco.

No princípio, o projeto utilizaria informações concebidas pelas API's freemium da Embrapa, porém ao realizar testes de conexões e consultas não obtivemos êxito, portanto criamos nossa própria API utilizando a aplicação Flask que alimenta o banco de dados local onde a partir dele realizamos as consultas que serão enviadas aos Forms do Django. Abaixo contém uma explicação detalhada de cada processo:

### **API com Flask**

Conexão com o MongoDB:

O código se conecta a um banco de dados MongoDB local chamado 'API' e a uma coleção chamada 'Plantas'.

População do Banco de Dados:

A função *“inserir\_arquivos\_json”* verifica se a coleção está vazia e, se estiver, preenche-a com dados de um arquivo JSON chamado *“API.Plantas.json”*.

### **Rotas da API:**

*“/:”* Rota principal que fornece uma mensagem de boas-vindas e descreve as diferentes rotas disponíveis na API.

*“/plantas:”* Retorna todas as plantas da coleção.

*“/solo:”* Retorna os tipos únicos de solo presentes nas plantas.

*“/nome:”* Retorna os nomes científicos e populares de todas as plantas.

*“/dificuldades:”* Retorna os diferentes tipos de dificuldades de cultivo presentes nas plantas.

*“/dificuldades/<Dificuldade>:”* Retorna informações sobre plantas que têm uma dificuldade específica de cultivo.

Manipulação de Dados:

*“busca\_plantas:”* Retorna todas as plantas da coleção, excluindo o campo *“\_id”*.

Exemplo de código:

```
@app.route('/plantas')
def busca_plantas():
    try:
        dados = colecao.find({}, {'_id': 0})
        dados_lista = list(dados)
        return jsonify(dados_lista), 200, {'Content-Type': 'application/json; charset=utf-8'} # 200 OK
    except Exception as e:
        return jsonify({"error": str(e)}), 500, {'Content-Type': 'application/json; charset=utf-8'} # 500 Internal Server Error
```

“**busca\_solo:**” Utiliza uma agregação para encontrar os tipos únicos de solo nas plantas.

“**busca\_nome:**” Retorna os nomes científicos e populares de todas as plantas.

“**busca\_dificuldade:**” Retorna informações sobre plantas que têm uma dificuldade específica de cultivo.

“**mostrar\_dificuldade:**” Utiliza uma agregação para encontrar os diferentes tipos de dificuldades de cultivo presentes nas plantas.

Execução da Aplicação:

A aplicação Flask é inicializada e executada quando o script é chamado diretamente (`__name__ == '__main__'`).

Por fim, este código implementa uma API básica para interagir com dados sobre plantas armazenados em um banco de dados MongoDB, oferecendo várias consultas e informações sobre as plantas e seus atributos.

## Conexão com o banco de dados:

### Função de Conexão:

- A função “*get\_mongo\_connection*” tenta estabelecer uma conexão com um servidor MongoDB local na porta 27017.
- Se a conexão for bem-sucedida, retorna o cliente MongoDB; caso contrário, imprime uma mensagem de erro e retorna None.

### Teste da Conexão:

- A função “*get\_mongo\_connection*” é chamada para obter o cliente MongoDB.
- Se a conexão for bem-sucedida, o script continua a executar operações no banco de dados; caso contrário, imprime uma mensagem indicando que a conexão falhou.

## Operações no MongoDB (Exemplo):

- Se a conexão for bem-sucedida, o script continua a executar operações no MongoDB.
- Define o nome do banco de dados como 'PROJETO' e o nome da coleção como 'semente'.
- Imprime mensagens indicando a conexão bem-sucedida ao banco de dados e à coleção.

### Manuseio de Erros:

- O código utiliza uma estrutura de try-except para capturar exceções que possam ocorrer durante a tentativa de conexão com o MongoDB.
- Se ocorrer um erro, imprime uma mensagem indicando o problema.

```
from pymongo import MongoClient

def get_mongo_connection():
    try:
        client = MongoClient('mongodb://localhost:27017/')
        print("Conexão com o MongoDB estabelecida com sucesso.")
        return client
    except Exception as e:
        print(f"Erro ao conectar ao MongoDB: {e}")
        return None

# Teste a conexão
mongo_client = get_mongo_connection()

# Verifique se a conexão foi bem-sucedida antes de usá-la
if mongo_client:
    # Faça o que precisa com o `mongo_client`
    db_name = 'PROJETO'
    collection_name = 'semente'

    print(f"Conectado ao banco de dados: {db_name}")
    db = mongo_client[db_name]

    print(f"Conectado à coleção: {collection_name}")
    colecao = db[collection_name]

    # ... (operações no MongoDB)
else:
    print("A conexão com o MongoDB falhou. Verifique as configurações.")
```

Este código é um script simples que estabelece uma conexão com um servidor MongoDB local e, se bem-sucedido, realiza operações no banco de dados. Ele inclui mensagens informativas para indicar o status da conexão e fornece uma abordagem básica para lidar com possíveis erros de conexão.

### **Alimentação do Banco de Dados com a API:**

#### **Configurações e Importações:**

- Importa o MongoClient da biblioteca pymongo.
- Importa as variáveis `colecacao` e `db` de um módulo chamado `mongodb_utils`.
- Importa a biblioteca `requests` para realizar solicitações HTTP.

#### **Configurações do MongoDB e da API:**

- Define a URI do MongoDB como `"mongodb://localhost:27017/"`.
- Define a URL da API como `"http://127.0.0.1:5000/plantas"`.

#### **Conexão ao MongoDB:**

- Cria uma instância do MongoClient usando a URI configurada.
- Não está claro por que há referências a `db` e `colecacao` após a criação do cliente, pois eles parecem não estar sendo utilizados corretamente.

#### **Função para Obter Dados da API:**

- A função `obter_dados_da_api` faz uma solicitação GET para a URL da API e retorna os dados JSON obtidos.
- Em caso de erro na solicitação, imprime uma mensagem indicando o problema e retorna `None`.

#### **Função para Inserir Dados no MongoDB:**

- A função `inserir_dados_no_mongodb` tenta inserir os dados fornecidos como argumento na coleção MongoDB configurada.
- Em caso de sucesso, imprime uma mensagem indicando o sucesso; em caso de erro, imprime uma mensagem indicando o problema.

#### **Função Principal (main):**

- Chama a função `obter_dados_da_api` para obter dados da API.
- Se os dados são obtidos com sucesso, chama a função `inserir_dados_no_mongodb` para inserir esses dados no MongoDB.
- Fecha a conexão com o MongoDB.

### Execução do Script:

- A função `main` é chamada se o script for executado diretamente (ou seja, se `__name__ == "__main__"`).

Este script obtém dados de uma API, utilizando a biblioteca `requests`, e os insere em um banco de dados MongoDB usando a biblioteca `pymongo`. O código inclui tratamento básico de erros durante a solicitação da API e a inserção de dados no MongoDB.

### CRUD:

O projeto possui um CRUD simples, onde o usuário pode criar novos cadastros de plantas, editar e excluir este cadastro e consultar toda a base de plantas contida no banco de dados. A estrutura para esse crud possui as seguintes características:

#### Create (Criar):

- Função `criar_planta`: Cria um novo documento (planta) na coleção MongoDB 'semente' com os dados fornecidos como parâmetros. Retorna o ID do documento recém-criado.

```
def criar_planta(nome_cientifico, nome_popular, melhor_solo, clima, regioao, dificuldade_cultivar, ml_dia):
    client = get_mongo_connection()
    db = client['PROJETO']
    colecao = db['semente']

    plantas = {"nome_cientifico": nome_cientifico, "nome_popular": nome_popular, "melhor_solo": melhor_solo, "clima": clima, "regiao": regioao, "dificuldade_cultivar": dificuldade_cultivar, "ml_dia": ml_dia}
    result = colecao.insert_one(plantas)

    client.close()
    return result.inserted_id
```

#### Read (Ler):

- Função `obter_planta`: Obtém todas as plantas da coleção 'semente' e retorna uma lista delas, excluindo o campo '`_id`'.
- Função `obter_planta_por_nome_cientifico`: Obtém uma planta específica da coleção com base no nome científico fornecido como parâmetro. Retorna os dados da planta, excluindo o campo '`_id`'.

```
def obter_planta():
    client = get_mongo_connection()
    db = client['PROJETO']
    colecao = db['semente']
    plantas = list(colecao.find({}, {'_id': 0}))

    client.close()
    return plantas

def obter_planta_por_nome_cientifico(busca):
    client = get_mongo_connection()
    db = client['PROJETO']
    colecao = db['semente']

    plantas_data = colecao.find_one({"nome_cientifico": busca})

    client.close()

    if plantas_data:
        plantas_data.pop('_id', None)
        print(f"DEBUG: Documento encontrado antes da atualização: {plantas_data}")
        return plantas_data
    else:
        return None
```

### Update (Atualizar):

- Função *atualizar\_planta*: Atualiza os dados de uma planta na coleção com base no nome científico fornecido. Os novos dados são passados como parâmetros para a função. Retorna o número de documentos modificados.

```
def atualizar_planta(nome_cientifico, novonome1, novonome2, novosolo, novoclima, novaregiao, novadiff, novoml):
    client = get_mongo_connection()
    db = client['PROJETO']
    colecao = db['semente']

    filtro = {"nome_cientifico": nome_cientifico}
    atualizacao = {"$set": {"nome_cientifico": novonome1, "nome_popular": novonome2, "melhor_solo": novosolo, "clima": novoclima, "regiao": novaregiao, "dificuldade_cultivar": novadiff, "ml_dia": novoml}}

    result = colecao.update_one(filtro, atualizacao)

    client.close()
    return result.modified_count
```

### Delete (Excluir):

- Função *excluir\_planta*: Exclui uma planta da coleção com base no nome científico fornecido. Retorna o número de documentos excluídos.

```
def excluir_planta(nome_planta):
    client = get_mongo_connection()
    db = client['PROJETO']
    colecao = db['semente']

    filtro = {"nome_cientifico": nome_planta}
    result = colecao.delete_one(filtro)

    client.close()
    return result.deleted_count
```

### Abordagem de agregações:

Neste projeto, escolhemos realizar as agregações do banco de dados diretamente na API. Essas agregações são usadas para obter informações específicas do banco de dados MongoDB e formatar os resultados para serem retornados pela API nas rotas correspondentes.

### Agregação em /solo - 'busca\_solo':

```
@app.route('/solo')
def busca_solo():
    pipeline = [
        {
            '$group': {
                '_id': None,
                'solos_unicos': {'$addToSet': '$melhor_solo'}
            }
        },
        {
            '$project': {
                '_id': 0,
                'solos_unicos': 1
            }
        }
    ]

    dados = colecao.aggregate(pipeline)

    solos_unicos = next(dados)['solos_unicos']

    return jsonify({'solos_unicos': solos_unicos})
```



## Objetivo:

Esta agregação é usada para encontrar os diferentes tipos únicos de solo presentes nas plantas.

## Passos:

- *'\$group'*: Agrupa os documentos da coleção, considerando todos como um único grupo (*\_id*: None).
- *'solos\_unicos'*: Utiliza o operador *\$addToSet* para adicionar os valores distintos do campo *melhor\_solo* a um conjunto.
- *'\$project'*: Projeta os resultados para excluir o campo *\_id* e manter apenas os solos únicos.

## Resultado:

A resposta é um conjunto de solos únicos, que é retornado pela rota */solo*.

## Agregação em */dificuldades* - *'mostrar\_dificuldade'*:

```
@app.route('/dificuldades/<dificuldade>')
def busca_dificuldade(dificuldade):
    pipeline = [
        {
            '$match': {
                'dificuldade_cultivar': dificuldade
            }
        },
        {
            '$project': {
                'dificuldade_cultivar': 1,
                '_id': 0,
                'nome_cientifico': 1,
                'nome_popular': 1,
                'clima': 1,
                'regiao': 1,
                'ml_dia': 1,
                'melhor_solo': 1,
            }
        }
    ]

    resultado = list(colecao.aggregate(pipeline))

    resposta_json = json.dumps({'Todas as plantas com a dificuldade': dificuldade, 'resultados': resultado}, ensure_ascii=False)

    return resposta_json

@app.route('/dificuldades')
def mostrar_dificuldade():
    pipeline = [
        {
            '$group': {
                '_id': None,
                'dificuldades_unicas': {'$addToSet': '$dificuldade_cultivar'}
            }
        },
        {
            '$project': {
                '_id': 0,
                'dificuldades_unicas': 1
            }
        }
    ]

    dados = colecao.aggregate(pipeline)

    dificuldades_unicas = next(dados)['dificuldades_unicas']

    resposta_json = json.dumps({'Todos os Tipos de dificuldades': dificuldades_unicas}, ensure_ascii=False)

    return resposta_json

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

**Objetivo:**

Essa agregação é usada para encontrar os diferentes tipos únicos de dificuldades de cultivo presentes nas plantas.

**Passos:**

- *'\$group'*: Agrupa os documentos da coleção, considerando todos como um único grupo (*\_id*: None).
- *'dificuldades\_unicas'*: Utiliza o operador *\$addToSet* para adicionar os valores distintos do campo *dificuldade\_cultivar* a um conjunto.
- *'\$project'*: Projeta os resultados para excluir o campo *\_id* e manter apenas as dificuldades únicas.

**Resultado:**

A resposta é um conjunto de dificuldades únicas, que é retornado pela rota */dificuldades*.

- Desenvolvimento Frontend

Utiliza HTML, CSS e JavaScript para criar uma interface limpa e resumida, garantindo uma experiência de usuário intuitiva e eficaz.

## 5. Gestão do Projeto (SCRUM)

### 5.1. Como foi gerenciado o projeto?

Para o gerenciamento do desenvolvimento da plataforma, utilizamos Scrum que é um framework ágil utilizado para desenvolver e entregar produtos complexos. Ele foi originalmente concebido para gerenciar projetos de software, mas tem sido aplicado com sucesso em uma variedade de contextos. Scrum proporciona uma abordagem iterativa e incremental para o desenvolvimento, enfatizando a colaboração, a transparência e a adaptação contínua.

### 5.2. Stakeholders

O Time Scrum (Scrum Team): O Time Scrum é composto por três papéis principais:

- **Product Owner (PO):** Responsável por definir as funcionalidades do produto, priorizá-las e garantir que a equipe esteja trabalhando nos itens mais valiosos para o negócio, este papel foi designado ao Pedro Otavio.
- **Scrum Master:** Facilitador do processo Scrum. O Scrum Master ajuda a equipe a entender e adotar os princípios e práticas do Scrum, além de remover obstáculos que possam prejudicar o progresso, este papel foi designado ao Matheus Matias.
- **Development Team:** A equipe responsável por desenvolver o produto. É auto-organizada e multifuncional, o que significa que inclui todas as habilidades necessárias para entregar um incremento de produto completo, este papel foi empenhado por todos da equipe.

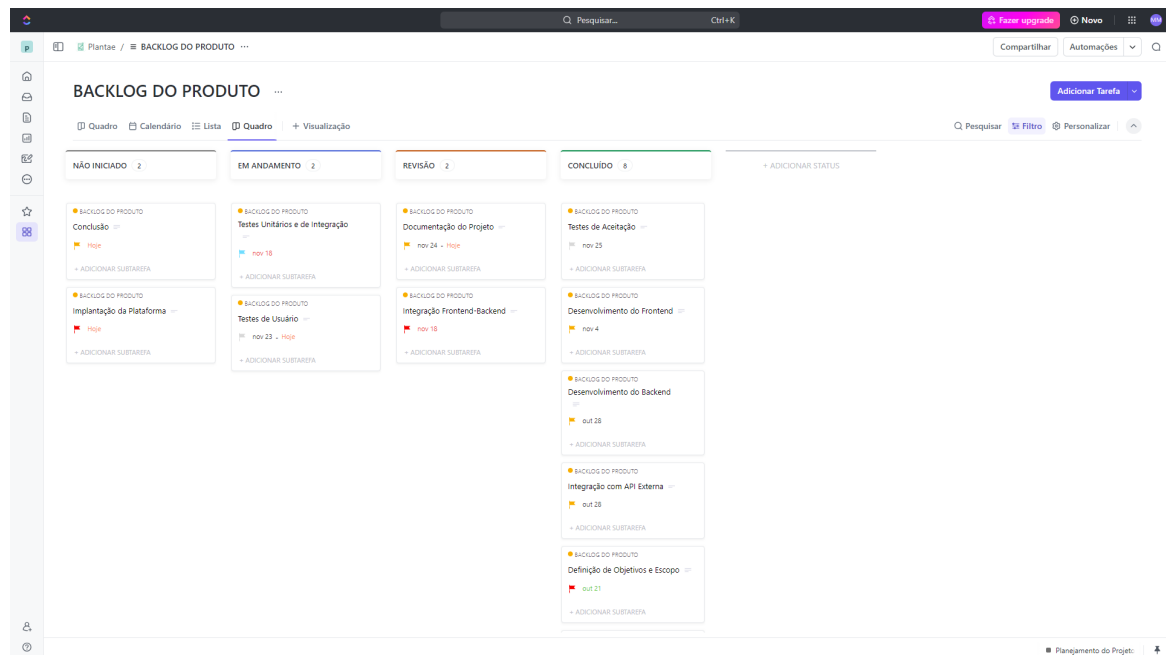
Para melhor entendimento, segue abaixo a matriz RACI criada para distribuir as funções:

Atividade	Matheus Matias	Pedro Otavio	Pedro Beck	Matheus Rodrigues
Criar Banco de Dados NoSQL	C	A	R	I
Criar CRUD backend	C	R	A	I
Criar Frontend	R	I	C	A
Design Layout	A	I	C	R
TDD	C	R	A	I
Integração API	C	A	R	I
Integração Frontend e Backend	R	C	A	I
Gerenciamento SCRUM	R	A	C	I

### 5.3. Artefatos do SCRUM

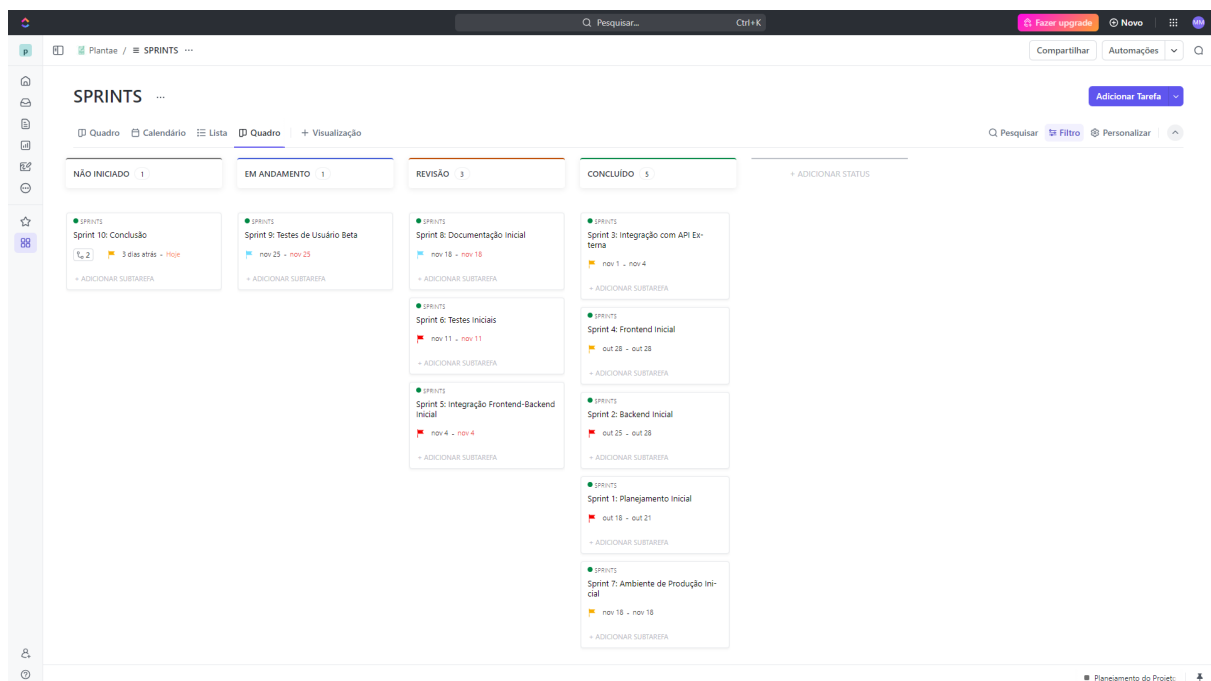
**Product Backlog:** Uma lista priorizada de todas as funcionalidades, melhorias e correções que precisam ser feitas no produto. O Product Owner é responsável por gerenciar o Product Backlog.

Abaixo um print demonstrando como foi desenvolvido nosso backlog, o programa utilizado em questão e em todos os próximos prints de gerenciamento é o ClickUp:



**Sprint Backlog:** Uma lista de tarefas específicas que a equipe seleciona para concluir durante uma Sprint (um período fixo de tempo, geralmente de 2 a 4 semanas).

Abaixo um print demonstrando como foi desenvolvido nosso sistema de Sprints:



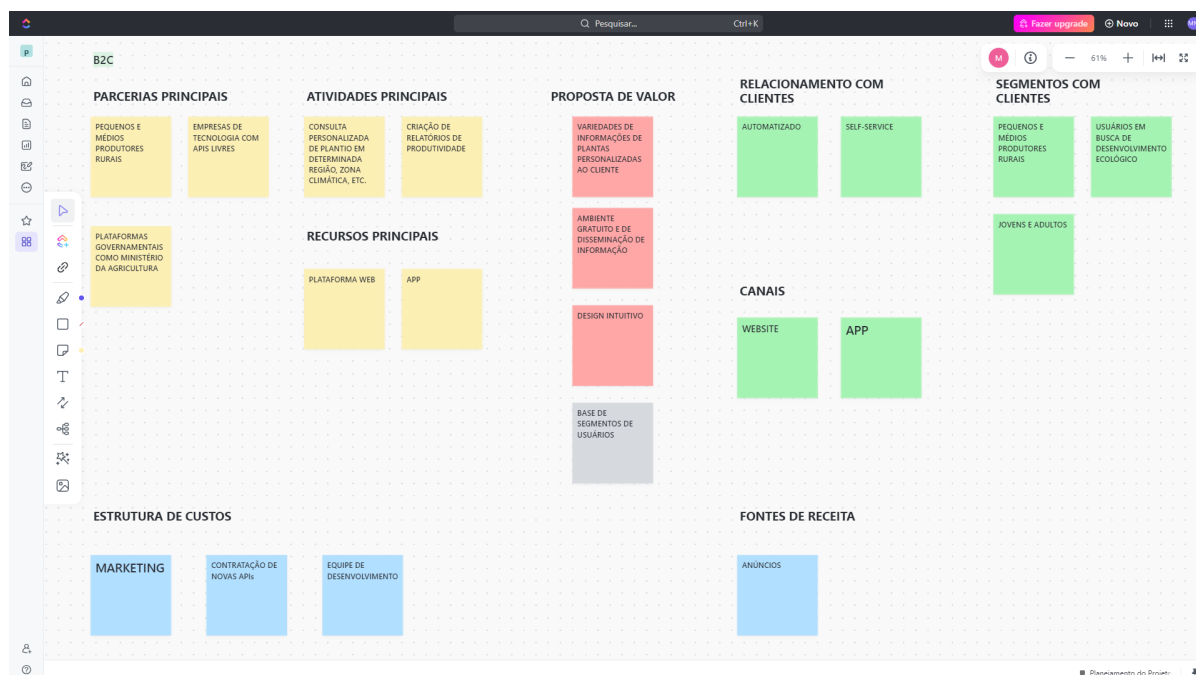
Vale ressaltar que nossas Sprints se iniciaram dia 07/10, onde em um período de 10 sprints de duração de 1 semana cada realizamos pequenas entregas de código ou documentações acompanhadas de nossas weekly's (reuniões de alinhamento semanal).

**OKR:** sigla para "Objective Key Results", uma metodologia para orientar os esforços das empresas em direção a objetivos cruciais mensuráveis. O modelo de gestão por OKRs busca estabelecer direções claras para alcançar os números que indicam o crescimento de um negócio.

Abaixo um print demonstrando como foi desenvolvido nosso sistema de OKR:

Nome	OKR Type	Responsável	Prioridade	Data inicial	Data de ve...	Progress...	OKR Point
Desenvolver uma Plataforma Web Funcional	OBJETIVO	Rr	Normal	set 1	dez 9	0%	Em andamen...
Desenvolver uma Plataforma Web Funcional	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Lançar a versão beta da plataforma em 02/12/2023	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Desenvolver uma Plataforma Web Funcional	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Garantir a integração eficiente com as API's da Embrapa	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Proporcionar Informações Detalhadas sobre Plantas	OBJETIVO	Rr	Normal	set 1	dez 9	0%	Em andamen...
Proporcionar Informações Detalhadas sobre Plantas	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Implementar consultas precisas baseadas em clima, solo e condições advers...	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Proporcionar Informações Detalhadas sobre Plantas	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Apresentar os resultados de consultas de forma clara e acessível	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Promover a Disseminação de Informações Agrícolas	OBJETIVO	Rr	Normal	set 1	dez 9	0%	Em andamen...
Promover a Disseminação de Informações Agrícolas	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Aumentar o número de consultas realizadas mensalmente	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Promover a Disseminação de Informações Agrícolas	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Estabelecer parcerias com organizações agrícolas para ampliar o alcance	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Garantir Acessibilidade e Uso Generalizado	OBJETIVO	Rr	Normal	set 1	dez 9	0%	Em andamen...
Garantir Acessibilidade e Uso Generalizado	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Atingir uma taxa de satisfação do usuário de 80% ou mais	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Garantir Acessibilidade e Uso Generalizado	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Certificar-se de que a plataforma seja otimizada para dispositivos móveis	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Manter a Qualidade e Atualização Constante	OBJETIVO	Rr	Normal	set 1	dez 9	0%	Em andamen...
Manter a Qualidade e Atualização Constante	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Realizar atualizações mensais na base de dados de plantas	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Manter a Qualidade e Atualização Constante	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...
Monitorar e resolver problemas de desempenho em tempo hábil	KEY RESULT	Rr	Normal	set 1	dez 9	0%	Em andamen...

Antes de todo o planejamento direto no Kanban, realizamos a criação de um bussines model canvas para podermos entender de maneira visual o cerne do projeto como um todo:



## 6. Conclusão

O projeto "PLANTAE: Biodiversidade Vegetal para as Gerações Futuras" representou um esforço colaborativo e interdisciplinar do time composto por Matheus Matias, Matheus Rodrigues, Pedro Penteado e Pedro Otávio, sob a orientação do professor Renato Cividini Matthiesen, como parte do Projeto Interdisciplinar do 3º semestre do curso de Desenvolvimento de Software Multiplataforma na FATEC Araras.

A visão geral do projeto abrangeu o desenvolvimento de uma plataforma web inovadora, cujo objetivo primário é capacitar produtores agrícolas a tomar decisões informadas sobre as melhores culturas para suas regiões. O foco na biodiversidade vegetal, aliado a algoritmos avançados e integração de dados climáticos e de solo, visa promover práticas agrícolas sustentáveis.

Os requisitos funcionais e não funcionais foram cuidadosamente delineados para assegurar uma experiência de usuário intuitiva, desempenho eficiente, compatibilidade e escalabilidade. A arquitetura do sistema, utilizando tecnologias como Django para o desenvolvimento backend e MongoDB para o banco de dados, demonstra uma abordagem robusta e eficaz.

A metodologia ágil SCRUM foi adotada para a gestão do projeto, permitindo uma abordagem iterativa e adaptativa. O time Scrum, composto pelo Product Owner

Pedro Otavio, Scrum Master Matheus Matias e o Development Team, demonstrou engajamento e colaboração ao longo das sprints, resultando em entregas incrementais e alinhamento contínuo com os objetivos do projeto.

Os artefatos do SCRUM, como o Product Backlog, Sprint Backlog e a aplicação de OKRs, foram essenciais para o planejamento, acompanhamento e alinhamento dos esforços da equipe. A transparência e a comunicação eficaz foram fundamentais para o sucesso do projeto.

O design de interface intuitivo e a arquitetura tecnológica escolhida refletem o compromisso da equipe com a usabilidade e eficácia da plataforma. A documentação abrangente, desde o business model canvas até o desenvolvimento do sistema, proporciona uma base sólida para futuras manutenções e expansões.

Em conclusão, o projeto PLANTAE representa não apenas um marco acadêmico, mas também uma contribuição potencial para a promoção de práticas agrícolas sustentáveis, capacitando os produtores a tomar decisões mais informadas para as gerações futuras. A colaboração, inovação e comprometimento da equipe foram fundamentais para alcançar os objetivos propostos.