

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA “PAULA SOUZA”

FACULDADE DE TECNOLOGIA DE BEBEDOURO

CURSO SUPERIOR DE TECNOLOGIA EM BIG DATA NO AGRONEGÓCIO

**GESTÃO EFICIENTE DE INSUMOS: OTIMIZANDO A
PRODUTIVIDADE NO SETOR AGRÍCOLA**

AUTOR: RICARDO PIRES DE CARVALHO JUNIOR

ORIENTADOR: ME. ALEXANDRE DE SOUZA FERNANDES

BEBEDOURO

2025

RICARDO PIRES DE CARVALHO JUNIOR

**GESTÃO EFICIENTE DE INSUMOS: OTIMIZANDO A
PRODUTIVIDADE NO SETOR AGRÍCOLA**

Monografia apresentada à Faculdade de
Tecnologia de Bebedouro, como parte dos
Requisitos para a obtenção do título de Tecnólogo
em Big Data no Agronegócio.

Orientador: Me. Alexandre de Souza Fernandes

BEBEDOURO

2025

JÚNIOR, RICARDO PIRES DE CARVALHO. **Gestão Eficiente de Insumos:** Otimizando a Produtividade no Setor Agrícola. Trabalho de Graduação (Monografia). Centro Estadual de Educação Tecnológica “Paula Souza”. Faculdade de Tecnologia de Bebedouro.

RESUMO

Este Trabalho de Conclusão de Curso apresenta o desenvolvimento de um sistema de gestão de insumos agrícolas, voltado para pequenos e médios produtores, com o objetivo de otimizar o controle de estoque e o registro de aplicações. A aplicação permite o cadastro detalhado de insumos e o registro de suas aplicações em talhões específicos, atualizando automaticamente o estoque disponível. Além disso, acrescenta uma funcionalidade de "Estoque Reservado", que contabiliza as aplicações agendadas para datas futuras, oferecendo uma visão antecipada das necessidades de reposição. A integração de técnicas de Big Data neste sistema para análise e controle de insumos, proporciona uma análise mais precisa, auxiliando na tomada de decisões estratégicas e na redução de desperdícios. O sistema visa promover uma gestão mais eficiente e sustentável dos recursos agrícolas, alinhando-se às práticas modernas de agricultura de precisão.

Palavras-chave: Big Data, Agronegócio, Gestão de Insumos, Controle de Estoque, Agricultura de Precisão.

JÚNIOR, RICARDO PIRES DE CARVALHO. **Gestão Eficiente de Insumos:** Otimizando a Produtividade no Setor Agrícola. Trabalho de Graduação (Monografia). Centro Estadual de Educação Tecnológica “Paula Souza”. Faculdade de Tecnologia de Bebedouro.

ABSTRACT

This paper presents the development of an agricultural input management system aimed at small and medium-sized producers, with the goal of optimizing inventory control and application records. The application enables detailed registration of inputs and their application to specific plots, automatically updating the available stock. Additionally, it introduces a "Reserved Stock" feature that accounts for scheduled future applications, providing an early view of replenishment needs. The integration of Big Data techniques enhances the analysis and control of input consumption, assisting in strategic decision-making and waste reduction. The system aims to promote more efficient and sustainable management of agricultural resources, aligning with modern precision agriculture practices.

Keywords: Big Data, Agrobusiness, Input Management, Inventory Control, Precision Agriculture.

DEDICATÓRIA

A todos que buscam inovar e fazer do mundo um lugar melhor.

AGRADESCIMENTO

Primeiramente a Deus, por ter me dado o dom a vida e me sustentado até aqui.

Aos meus pais, Ricardo Pires de Carvalho e Jandira Francisco de Carvalho, e meus parentes mais próximos, meus avós, Mariano Borges de Carvalho e Benedita Pires de Carvalho, meu tio, Antonino Borges de Carvalho, e minhas tias, Lourdes Pires de Carvalho e Aparecida Borges de Carvalho, que me acompanharam dès de o nascimento até minha mocidade, com muito amor, carinho e dedicação, me trazendo ensinamentos de fé e de vida, e que fizeram de mim a pessoa que sou hoje.

Ao meu orientador. Alexandre de Souza Fernandes, que acolheu minha proposta de escrever sobre o tema escolhido, e que, apesar das mudanças que fui fazendo ao longo do caminho, não deixou de me apoiar e me ajudar para que pudesse finalizar este trabalho de conclusão de curso.

Aos meus professores, que seriam muitos nomes para recordar aqui, além do risco de esquecer-me de algum, mas deixo meus sinceros agradecimentos, por tudo que me ensinaram, e não apenas no campo acadêmico, e que me tornaram apto a elaborar este trabalho monográfico.

Aos meus companheiros de turma, tanto os da turma atual quando da minha primeira, pelas conversas, risadas, trocas de experiência e ajuda sempre recíproca.

A todos, de forma geral, que sendo em maior ou menor parte, fizeram-se presentes em minha vida e me ajudaram a me tornar hoje quem eu sou: muito obrigado!

“Não existe qualquer tipo de dúvida que os desafios postos à agricultura somente serão superados com a adoção de tecnologias modernas.”

(Fernanda Mendes Lamas)

LISTA DE FIGURAS

FIGURA 1: INTERFACE GRÁFICA DA FUNÇÃO MAIN().	18
FIGURA 2: INTERFACE GRÁFICA DA FUNÇÃO ABRIR_CADASTRO_INSUMOS().	21
FIGURA 3: INTERFACE GRÁFICA DA FUNÇÃO ABRIR_EXCLUSAO_INSUMOS().	23
FIGURA 4: INTERFACE GRÁFICA DA FUNÇÃO ABRIR_VISUALIZACAO_ESTOQUE().	27
FIGURA 5: INTERFACE GRÁFICA DA FUNÇÃO ABRIR_CADASTRO_APLICACOES().	30
FIGURA 6: INTERFACE GRÁFICA DA FUNÇÃO ABRIR_EXCLUCAO_APLICACAO().	33
FIGURA 7: INTERFACE GRÁFICA DA FUNÇÃO VISUALIZAR_ESTOQUE_RESERVADO().	35
FIGURA 8: INTERFACE GRÁFICA DA FUNÇÃO ABRIR_RELATORIOS_GRAFICOS().	39
FIGURA 9: MODELO DO GRÁFICO GERADO	39
FIGURA 10: INTERFACE GRÁFICA DA FUNÇÃO ABRIR_RELATORIO_INSUMOS_POR_TALHAO().	42
FIGURA 11: MODELO DO GRÁFICO DE SETORES GERADO	42

SUMÁRIO

1 INTRODUÇÃO	10
2 DESENVOLVIMENTO	11
CAPÍTULO I: A IMPORTÂNCIA DO GERENCIAMENTO EFICIENTE DE ESTOQUES NO AGRONEGÓCIO.....	11
CAPÍTULO II: DESENVOLVIMENTO DO PROGRAMA PARA AUXILIAR O PRODUTOR NA GESTÃO DO ESTOQUE DE INSUMOS.....	13
2.1 ESCOLHA DA LINGUAGEM DE PROGRAMAÇÃO: PYTHON	13
2.2 ESCOLHA DO BANCO DE DADOS: SQLITE.....	13
2.3 ESTRUTURA DO CÓDIGO E SUAS FUNCIONALIDADES.....	14
<i>2.3.1 Bibliotecas Utilizadas no Desenvolvimento do Sistema</i>	<i>14</i>
<i>2.3.2 Funções criadas e suas funcionalidades.....</i>	<i>16</i>
2.3.2.1 Função main().....	16
2.3.2.2 Função abrir_cadastro_insumos().....	18
2.3.2.2 Função abrir_exclusao_insumos().....	21
2.3.2.3 Função abrir_visualizacao_estoque().....	23
2.3.2.4 Função abrir_cadastro_aplicacoes().....	27
2.3.2.5 Função abrir_exclusao_aplicacao().....	30
2.3.2.6 visualizar_estoque_reservado().....	33
2.3.2.7 Função abrir_relatorios_graficos().....	36
2.3.2.8 Função abrir_relatorio_insumos_por_talhao().....	40
3 CONSIDERAÇÕES FINAIS.....	43
REFERÊNCIAS	45

1 INTRODUÇÃO

O agronegócio brasileiro destaca-se como um dos principais pilares da economia nacional, sendo responsável por significativa parcela do Produto Interno Bruto (PIB) e das exportações do país. Apesar dos avanços tecnológicos nas grandes propriedades, muitos pequenos e médios produtores ainda enfrentam desafios na gestão eficiente de seus recursos, especialmente no que tange ao controle de insumos agrícolas. A ausência de sistemas integrados e ferramentas adequadas pode resultar em desperdícios, aumento de custos operacionais e impactos negativos na produtividade (BATEMAN; SNELL, 1998).

Nesse contexto, a aplicação de tecnologias emergentes, como o Big Data, tem se mostrado promissora para transformar a gestão agrícola. O Big Data permite a coleta, armazenamento e análise de grandes volumes de dados provenientes de diversas fontes, como sensores, imagens de satélite e registros operacionais, proporcionando insights valiosos para a tomada de decisões estratégicas no campo (CASAROTTO et al., 2022). A adoção dessas tecnologias possibilita uma gestão mais precisa dos insumos, otimizando seu uso e contribuindo para a sustentabilidade das operações agrícolas (EJNISMAN et al., 2019).

Este trabalho propõe o desenvolvimento de um sistema de gestão de insumos agrícolas voltado para pequenos e médios produtores, integrando princípios de Big Data para aprimorar o controle de estoque e o registro de aplicações. A solução visa oferecer uma ferramenta acessível e eficiente, permitindo o cadastro de insumos, o planejamento e registro de aplicações, além de fornecer análises que auxiliem na tomada de decisões, alinhando-se às práticas modernas de agricultura de precisão (BORGES et al., 2022).

2 DESENVOLVIMENTO

CAPÍTULO I: A Importância do Gerenciamento Eficiente de Estoques no Agronegócio

O agronegócio brasileiro desempenha um papel fundamental na economia nacional, sendo responsável por uma significativa parcela do Produto Interno Bruto (PIB) e das exportações do país. Nesse contexto, a gestão eficiente de estoques de insumos agrícolas torna-se essencial para garantir a continuidade das operações, a redução de custos e o aumento da produtividade nas propriedades rurais.

A gestão de estoques é uma atividade estratégica que envolve o planejamento, a organização e o controle dos materiais necessários para as atividades produtivas. Segundo Ballou (2006), o estoque compreende "as acumulações de matérias-primas, suprimentos, componentes, materiais em processo e produtos acabados que surgem em numerosos pontos do canal de produção e logística das empresas", sendo que seu custo de manutenção pode representar de 20% a 40% do valor do estoque por ano.

No agronegócio, a gestão de estoques de insumos, como sementes, fertilizantes e defensivos agrícolas, é particularmente desafiadora devido à sazonalidade da produção, às condições climáticas e às variações de mercado. Uma gestão inadequada pode resultar em perdas significativas, seja por falta de insumos no momento necessário, seja por excesso de produtos que podem vencer ou deteriorar-se. Conforme destaca Foganholti (2023), "a falta deste controle nos estoques pode ocasionar prejuízos, uma vez que este tem parte fundamental nas finanças da empresa".

Além disso, uma gestão eficiente de estoques permite melhor planejamento de compras, negociação com fornecedores e aproveitamento de condições favoráveis de mercado. De acordo com a ECONT Sistemas (2023),

A gestão de estoque rural também permite que você saiba quando é o momento certo de repor os itens, levando em consideração, inclusive, o período que o mercado oferece melhores descontos e condições de pagamentos.

A adoção de tecnologias de informação e sistemas de gestão integrada contribui para a eficiência na gestão de estoques. Conforme aponta a Brid Soluções (2022),

Quando a gestão de estoque é bem-feita por meio de uma tecnologia de Business Intelligence (BI), os riscos operacionais no estoque são mitigados, mesmo com os diversos processos logísticos e de armazenagem que produtos como fertilizantes, defensivos, sementes e suprimentos necessitam.

Em suma, a gestão eficiente de estoques no agronegócio é fundamental para garantir a disponibilidade de insumos no momento adequado, otimizar os recursos financeiros e aumentar a competitividade das propriedades rurais. A implementação de práticas e ferramentas de gestão adequadas pode resultar em melhorias significativas na produtividade e na sustentabilidade das operações agrícolas.

CAPÍTULO II: DESENVOLVIMENTO DO PROGRAMA PARA AUXILIAR O PRODUTOR NA GESTÃO DO ESTOQUE DE INSUMOS

Este capítulo descreve o desenvolvimento do sistema de gestão de insumos agrícolas, detalhando sua estrutura, funcionalidades e as tecnologias empregadas. O sistema foi concebido para atender às necessidades de pequenos e médios produtores rurais, proporcionando uma ferramenta acessível e eficiente para o controle de estoque e registro de aplicações de insumos.

2.1 Escolha da Linguagem de Programação: Python

A linguagem de programação Python foi selecionada para o desenvolvimento do sistema devido à sua simplicidade, legibilidade e ampla comunidade de desenvolvedores. Python é uma linguagem de alto nível, interpretada e de tipagem dinâmica, que favorece a rápida prototipagem e desenvolvimento de aplicações (PYTHON SOFTWARE FOUNDATION, 2025).

Além disso, Python possui uma vasta gama de bibliotecas e frameworks que facilitam o desenvolvimento de interfaces gráficas, manipulação de dados e integração com bancos de dados. Sua sintaxe clara e objetiva contribui para a manutenção e escalabilidade do código, aspectos essenciais em projetos de software voltados para o agronegócio, onde a adaptabilidade às necessidades do produtor é fundamental (DATACAMP, 2025).

A escolha por Python também se justifica pela sua crescente adoção no setor agrícola, especialmente em aplicações de análise de dados e automação de processos, alinhando-se às tendências de digitalização e uso de tecnologias emergentes no campo (SCICROP, 2019).

2.2 Escolha do Banco de Dados: SQLite

Para o armazenamento e gerenciamento dos dados do sistema, optou-se pelo uso do SQLite, um sistema de gerenciamento de banco de dados relacional leve, embutido e de código aberto. O SQLite é caracterizado por sua simplicidade de implementação, não requerendo a instalação de um servidor separado, o que o torna ideal para aplicações de pequeno e médio porte.

Entre as principais vantagens do SQLite destacam-se: ZERO CONFIGURAÇÃO¹: dispensa processos complexos de instalação e administração, facilitando sua integração ao sistema desenvolvido; PORTABILIDADE²: o banco de dados é armazenado em um único arquivo, permitindo fácil transferência e backup dos dados; CONFIABILIDADE³: oferece suporte a transações ACID⁴, garantindo a integridade e consistência dos dados mesmo em casos de falhas (FERNANDES, 2023).

A escolha do SQLite alinha-se à proposta de desenvolver uma solução prática e eficiente para produtores rurais, que muitas vezes operam em ambientes com recursos computacionais limitados e necessitam de sistemas que funcionem de forma autônoma e confiável.

2.3 Estrutura do código e suas funcionalidades

2.3.1 Bibliotecas Utilizadas no Desenvolvimento do Sistema

No contexto da programação, uma BIBLIOTECA⁵ é um conjunto de funções, classes e métodos pré-definidos que facilitam o desenvolvimento de aplicações, promovendo a reutilização de código e a padronização de funcionalidades. Essas bibliotecas são desenvolvidas para atender a necessidades específicas, como manipulação de interfaces gráficas, acesso a bancos de dados, geração de gráficos, entre outras.

A seguir, são descritas as bibliotecas utilizadas no desenvolvimento do sistema de gestão de insumos agrícolas:

```
import tkinter as tk
from tkinter import ttk, messagebox
from tkcalendar import DateEntry
import sqlite3
import matplotlib.pyplot as plt
import mplcursors
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import datetime
from datetime import datetime
import plotly.graph_objects as go
```

¹ Grifo nosso

² Grifo nosso

³ Grifo nosso

⁴ ACID é uma sigla para as quatro principais características que definem uma transação: Atomicidade, Consistência, Isolamento e Durabilidade

⁵ Grifo nosso

```
from plotly.offline import plot
import webbrowser
import tempfile
```

tkinter: Biblioteca padrão do Python para criação de interfaces gráficas (GUI). Permite o desenvolvimento de janelas, botões, menus e outros componentes gráficos, facilitando a interação do usuário com o sistema.

ttk: Subconjunto do tkinter que oferece *widgets* (dispositivos) com estilos aprimorados, proporcionando uma aparência mais moderna e consistente às interfaces gráficas.

messagebox: Módulo do tkinter utilizado para exibir caixas de diálogo, como mensagens de aviso, erro ou confirmação, melhorando a comunicação com o usuário.

tkcalendar.DateEntry: Widget que permite a seleção de datas por meio de um calendário interativo, facilitando a entrada de informações temporais no sistema.

sqlite3: Módulo integrado ao Python que fornece uma interface para o banco de dados SQLite, permitindo a execução de comandos SQL e a manipulação de dados de forma eficiente e leve.

matplotlib.pyplot: Biblioteca para geração de gráficos em duas dimensões, como linhas, barras e dispersão, possibilitando a visualização de dados de forma clara e informativa.

mplcursors: Extensão do matplotlib que adiciona interatividade aos gráficos, permitindo que o usuário obtenha informações detalhadas ao passar o cursor sobre os elementos do gráfico.

FigureCanvasTkAgg: Classe que integra os gráficos do matplotlib às interfaces do tkinter, permitindo a incorporação de gráficos diretamente nas janelas da aplicação.

datetime: Módulo padrão do Python para manipulação de datas e horas, essencial para o registro e controle temporal das operações no sistema.

plotly.graph_objects: Biblioteca para criação de gráficos interativos e dinâmicos, oferecendo uma ampla variedade de visualizações avançadas.

plotly.offline.plot: Função que permite a geração de gráficos interativos do Plotly em arquivos HTML, possibilitando a visualização dos gráficos em navegadores web sem necessidade de conexão com servidores externos.

webbrowser: Módulo que fornece uma interface para exibição de documentos em navegadores web, utilizado para abrir automaticamente os gráficos gerados em HTML.

tempfile: Módulo que permite a criação de arquivos temporários, úteis para armazenar dados intermediários ou resultados gerados durante a execução do sistema.

A integração dessas bibliotecas no desenvolvimento do sistema proporcionou uma solução robusta, interativa e eficiente para o gerenciamento de insumos agrícolas, atendendo às necessidades específicas dos produtores rurais.

2.3.2 Funções criadas e suas funcionalidades

2.3.2.1 Função main()

```
# Função principal para criar a janela principal e o menu
def main():
    janela_principal = tk.Tk()
    janela_principal.title("Sistema de Gestão Agrícola")
    janela_principal.geometry("500x400")

    barra_menus = tk.Menu(janela_principal)

    menu_insumos = tk.Menu(barra_menus, tearoff=0)
    menu_insumos.add_command(label="Cadastrar Insumos",
command=abrir_cadastro_insumos)
    menu_insumos.add_command(label="Visualizar Estoque",
command=abrir_visualizacao_estoque)
    menu_insumos.add_command(label="Visualizar Estoque Reservado",
command=visualizar_estoque_reservado)
    menu_insumos.add_command(label="Excluir Quantidade de Insumo",
command=abrir_exclusao_insumos)
    barra_menus.add_cascade(label="Insumos", menu=menu_insumos)

    # Menu "Aplicações"
    menu_aplicacoes = tk.Menu(barra_menus, tearoff=0)
    menu_aplicacoes.add_command(label="Cadastrar Aplicações",
command=abrir_cadastro_aplicacoes)
    # Adicione o botão na aba "Aplicações"
    menu_aplicacoes.add_command(label="Excluir Aplicação",
command=abrir_exclusao_aplicacao)
    barra_menus.add_cascade(label="Aplicações", menu=menu_aplicacoes)

    # Menu "Relatórios"
    menu_relatorios = tk.Menu(barra_menus, tearoff=0)
    menu_relatorios.add_command(label="Gerar Relatório de Barras",
command=abrir_relatorios_graficos)
```

```

menu_relatorios.add_command(label="Relatório de Insumos por Talhão e
Data", command=abrir_relatorio_insumos_por_talhao)
barra_menus.add_cascade(label="Relatórios", menu=menu_relatorios)

# Adicionando a barra de menus à janela principal
janela_principal.config(menu=barra_menus)

# Iniciando o loop principal
janela_principal.mainloop()

# Executando a função principal
if __name__ == "__main__":
    main()

```

A função `main()` inicia o aplicativo criando uma janela principal por meio de `tk.Tk()`, que recebe título e dimensões padronizadas (500×400). Em seguida, é construída uma barra de menus (`tk.Menu`) para facilitar a navegação do usuário entre as opções do sistema.

O primeiro menu, rotulado como “Insumos”, agrupa comandos essenciais para a gestão de insumos: Cadastrar Insumos, que abre a interface de registro; Visualizar Estoque, que exibe tanto os insumos ativos quanto os reservas; Visualizar Estoque Reservado, que acessa especificamente a reserva; Excluir Quantidade de Insumo, que permite reduzir o estoque manualmente.

O segundo menu, intitulado “Aplicações”, contém comandos voltados às aplicações agrícolas: permite registrar novas aplicações e excluir registros já inseridos.

O terceiro menu, “Relatórios”, oferece acesso direto às funções de geração de relatórios: uma visualização em forma de gráfico de barras consolidadas por talhão e outra mais detalhada de insumos aplicados por data e localização.

Após a definição das opções e inclusão dessas categorias na barra, o menu é fixado à janela principal por `config(menu=...)`. Por fim, `mainloop()` mantém a interface ativa, escutando eventos do usuário e permitindo que ele interaja com os comandos disponíveis. A condição `if __name__ == "__main__":` `main()` garante que essa interface só seja inicializada quando o arquivo for executado diretamente—evitando que a interface abra se o módulo for importado em outro script.

Em resumo, a função `main()` organiza a janela principal, fornece acesso centralizado às funcionalidades de cadastro, visualização, exclusão e geração de relatórios, e garante que

todas essas operações sejam acionadas por meio de um menu intuitivo, proporcionando uma experiência integrada de uso dentro do contexto do sistema de gestão agrícola.

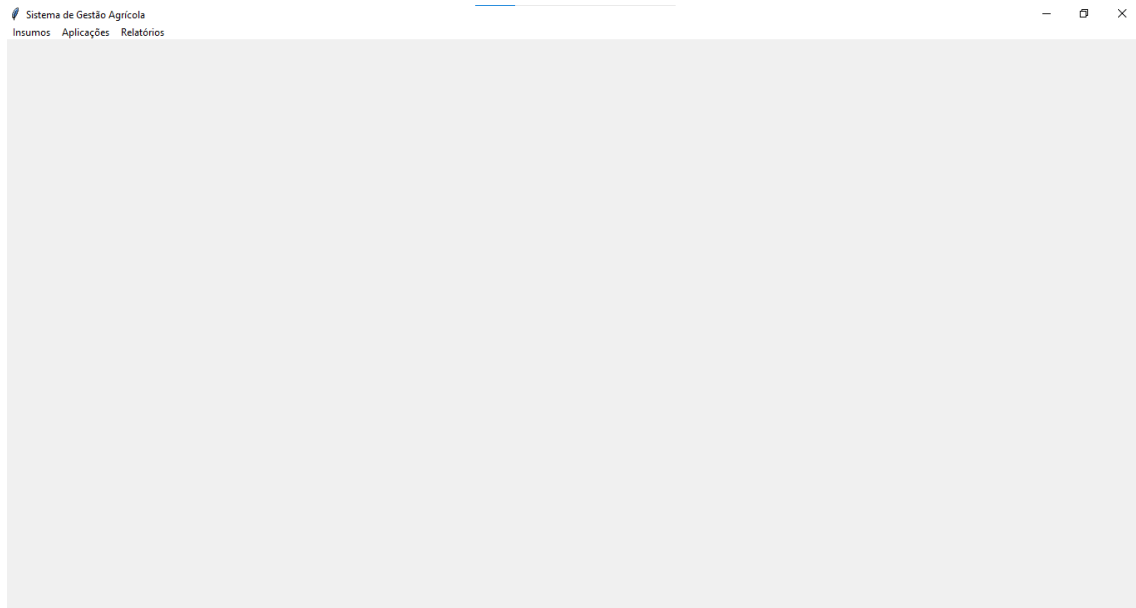


Figura 1: Interface Gráfica da Função main().

2.3.2.2 Função abrir_cadastro_insumos().

```
def abrir_cadastro_insumos():
    def salvar_insumo():
        grupo = entry_grupo.get().strip().title()
        nome = entry_nome.get().strip().title()
        quantidade = entry_quantidade.get().strip()
        unidade = entry_unidade.get().strip().capitalize()

        if not grupo or not nome or not quantidade or not unidade:
            messagebox.showerror("Erro", "Todos os campos devem ser
preenchidos.")
            return

        try:
            quantidade = float(quantidade)
        except ValueError:
            messagebox.showerror("Erro", "Quantidade deve ser um número.")
            return

        try:
            conn = sqlite3.connect('gestao_agricola.db')
            cursor = conn.cursor()
            cursor.execute('''
                CREATE TABLE IF NOT EXISTS insumos
                (
                    id
                    INTEGER
                    PRIMARY
                    KEY
                    AUTOINCREMENT,
```

```

        grupo
        TEXT
        NOT
        NULL,
        nome
        TEXT
        NOT
        NULL,
        quantidade
        REAL
        NOT
        NULL,
        unidade
        TEXT
        NOT
        NULL,
        UNIQUE
    (
        grupo,
        nome
    )
'''
cursor.execute('''
INSERT INTO insumos (grupo, nome, quantidade,
unidade)
VALUES (?, ?, ?, ?) ON CONFLICT (grupo, nome) DO
UPDATE SET
    quantidade = quantidade +
excluded.quantidade,
    unidade = excluded.unidade
''', (grupo, nome, quantidade, unidade))

conn.commit()
conn.close()

messagebox.showinfo("Sucesso", "Insumo cadastrado ou atualizado
com sucesso.")
entry_grupo.delete(0, tk.END)
entry_nome.delete(0, tk.END)
entry_quantidade.delete(0, tk.END)
entry_unidade.delete(0, tk.END)
except Exception as e:
    messagebox.showerror("Erro", f"Erro ao salvar insumo: {e}")

janela_insumos = tk.Toplevel()
janela_insumos.title("Cadastro de Insumos")
janela_insumos.geometry("400x300")

tk.Label(janela_insumos, text="Grupo:").pack(pady=5)
entry_grupo = tk.Entry(janela_insumos, width=50)
entry_grupo.pack(pady=5)

tk.Label(janela_insumos, text="Nome do Insumo:").pack(pady=5)
entry_nome = tk.Entry(janela_insumos, width=50)
entry_nome.pack(pady=5)

tk.Label(janela_insumos, text="Quantidade:").pack(pady=5)
entry_quantidade = tk.Entry(janela_insumos, width=50)
entry_quantidade.pack(pady=5)

tk.Label(janela_insumos, text="Unidade (kg, L, etc.):").pack(pady=5)

```

```
entry_unidade = tk.Entry(janela_insumos, width=50)
entry_unidade.pack(pady=5)

tk.Button(janela_insumos, text="Salvar",
command=salvar_insumo).pack(pady=20)
```

A função `abrir_cadastro_insumos()` cria uma janela dedicada ao cadastro de insumos agrícolas. Ao ser chamada, ela abre um componente `Toplevel()` do Tkinter, configurado para ter um tamanho de 400×300 pixels, com campos para entrada de grupo, nome, quantidade e unidade do insumo, cada um acompanhado por um rótulo claro que orienta o usuário.

Internamente, essa janela contém a função `salvar_insumo()`, que é acionada quando se clica no botão “Salvar”. Inicialmente, essa função coleta os valores informados nos campos de entrada, limpando espaços ao redor do texto (`.strip()`), padronizando as palavras: o grupo e o nome ficam em formato title case, enquanto a unidade é formatada apenas com a primeira letra em maiúscula (`.capitalize()`). Em seguida, é realizado um pré-processamento de validação: caso qualquer campo esteja vazio, um alerta aparece solicitando o preenchimento completo. Se os campos estiverem todos preenchidos, tenta-se converter a quantidade para número real (float). Em caso de falha nessa conversão, é exibida outra mensagem de erro avisando que a quantidade precisa ser um número.

Após essa validação, a função estabelece conexão com o banco de dados SQLite, no arquivo `gestao_agricola.db`. Primeiro, executa um comando SQL para criar a tabela `insumos` caso ela ainda não exista. Essa tabela tem cinco colunas: `id` (chave primária com auto incremento); `grupo` e `nome` (ambos do tipo texto e obrigatórios); `quantidade` (tipo real, obrigatório) e `unidade` (texto obrigatório). Além disso, foi definido um índice único sobre os campos `grupo` e `nome`, que impede duplicatas com o mesmo par de valores.

Na sequência, a função executa um comando `INSERT ... ON CONFLICT(grupo, nome) DO UPDATE`, que atua de forma inteligente: se um insumo com aquele par grupo-nome já existir, ele não cria um novo registro, mas sim atualiza a quantidade existente, somando àquela que já estava registrada, e atualiza a unidade com a nova informada. Se não existir, ele cria um novo registro. Isso permite que o usuário cadastre insumos repetidos sem gerar duplicidade, apenas acumulando a quantidade de maneira segura.

Após a execução bem-sucedida da transação, o banco é atualizado com `conn.commit()`, a conexão é fechada, e uma mensagem de sucesso é apresentada ao usuário. Por fim, os campos da interface são limpos (`.delete()`) para facilitar o próximo cadastro.

Se houver qualquer exceção durante o processo — seja ela relacionada ao banco ou à execução do script — o código a captura e exibe uma mensagem de erro mais detalhada para o usuário, garantindo transparência e melhor experiência.

Em resumo, `abrir_cadastro_insumos()` combina uma interface gráfica amigável com um backend robusto: valida dados, padroniza entradas, gerencia conflitos via SQL inteligente, mantém a consistência dos dados e oferece feedback ao usuário, tudo de maneira integrada. Essa abordagem garante eficiência operacional no controle de insumos agrícolas no contexto de sistemas de gestão rural.

Figura 2: Interface Gráfica da Função `abrir_cadastro_insumos()`.

2.3.2.2 Função `abrir_exclusao_insumos()`.

```
def abrir_exclusao_insumos():
    def excluir_quantidade():
        insumo = combo_insumo.get().strip()
        quantidade = entry_quantidade.get().strip()

        if not insumo or not quantidade:
            messagebox.showerror("Erro", "Todos os campos devem ser
preenchidos.")
            return

        try:
            quantidade = float(quantidade)
        except ValueError:
            messagebox.showerror("Erro", "Quantidade deve ser um número.")
            return

        try:
```

```

        conn = sqlite3.connect('gestao_agricola.db')
        cursor = conn.cursor()

        cursor.execute('SELECT quantidade FROM insumos WHERE nome = ?',
(insumo,))
        resultado = cursor.fetchone()

        if resultado is None:
            messagebox.showerror("Erro", f"Insumo '{insumo}' não
encontrado.")
            return

        estoque_atual = resultado[0]

        if quantidade > estoque_atual:
            messagebox.showerror("Erro", f"Quantidade a excluir excede
o estoque atual ({estoque_atual}).")
            return

        novo_estoque = estoque_atual - quantidade

        # Atualiza sempre, mesmo que zere
        cursor.execute('UPDATE insumos SET quantidade = ? WHERE nome =
?', (novo_estoque, insumo))

        conn.commit()
        conn.close()

        messagebox.showinfo("Sucesso", "Quantidade atualizada com
sucesso.")
        entry_quantidade.delete(0, tk.END)
    except Exception as e:
        messagebox.showerror("Erro", f"Erro ao excluir quantidade:
{e}")

janela_exclusao = tk.Toplevel()
janela_exclusao.title("Excluir Quantidade de Insumo")
janela_exclusao.geometry("400x200")

try:
    conn = sqlite3.connect('gestao_agricola.db')
    cursor = conn.cursor()
    cursor.execute('SELECT nome FROM insumos')
    insumos = [row[0] for row in cursor.fetchall()]
    conn.close()
except Exception as e:
    messagebox.showerror("Erro", f"Erro ao carregar insumos: {e}")
    return

tk.Label(janela_exclusao, text="Nome do Insumo:").pack(pady=5)
combo_insumo = ttk.Combobox(janela_exclusao, values=insumos, width=47)
combo_insumo.pack(pady=5)

tk.Label(janela_exclusao, text="Quantidade a Excluir:").pack(pady=5)
entry_quantidade = tk.Entry(janela_exclusao, width=50)
entry_quantidade.pack(pady=5)

tk.Button(janela_exclusao, text="Excluir",
command=excluir_quantidade).pack(pady=20)

```

A função `abrir_exclusao_insumos()` é responsável por criar uma janela secundária na interface gráfica do sistema, permitindo ao usuário excluir uma quantidade específica de um insumo previamente cadastrado no banco de dados. Ao ser executada, ela abre uma nova janela com campos para seleção do nome do insumo (através de uma caixa de combinação que é automaticamente preenchida com os nomes disponíveis no banco) e para digitação da quantidade a ser removida.

Ao pressionar o botão “Excluir”, o programa executa a função `excluir_quantidade()`, que realiza validações importantes: verifica se todos os campos estão preenchidos, se a quantidade digitada é um número válido e se o insumo selecionado existe no banco. Em seguida, consulta o estoque atual do insumo e compara com a quantidade informada para remoção; se o valor a ser excluído for maior que o disponível, exibe uma mensagem de erro e interrompe a operação. Caso contrário, atualiza o valor no banco de dados, subtraindo a quantidade informada, e exibe uma mensagem de sucesso.

Ao final, limpa o campo de entrada para facilitar novas exclusões. A estrutura garante que apenas valores válidos sejam removidos e protege o banco contra inconsistências de estoque, sendo uma função essencial para a gestão precisa dos insumos agrícolas.

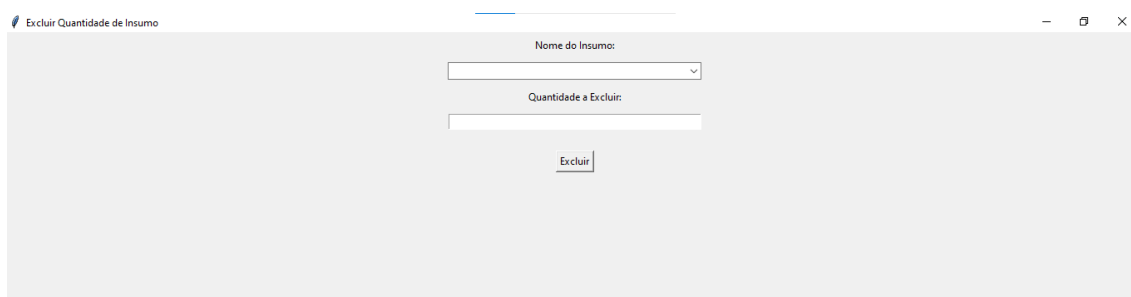


Figura 3: Interface Gráfica da Função `abrir_exclusao_insumos()`.

2.3.2.3 Função `abrir_visualizacao_estoque()`.

```
def abrir_visualizacao_estoque():
    def carregar_estoque(filtro_query="", params=()):
        for item in tree.get_children():
            tree.delete(item)
        try:
            conn = sqlite3.connect('gestao_agricola.db')
            cursor = conn.cursor()
            query = 'SELECT id, nome, quantidade, unidade, grupo FROM
insumos'
            if filtro_query:
                query += " " + filtro_query
            cursor.execute(query, params)
            registros = cursor.fetchall()
            conn.close()
```

```

        for registro in registros:
            tree.insert('', tk.END, values=registro)
        except Exception as e:
            messagebox.showerror("Erro", f"Erro ao carregar estoque: {e}")

    def aplicar_filtros():
        grupos_selecionados = [grupos_listbox.get(i) for i in
                                grupos_listbox.curselection()]
        unidades_selecionadas = [unidades_listbox.get(i) for i in
                                  unidades_listbox.curselection()]

        condicoes = []
        params = []

        if grupos_selecionados:
            condicoes.append("grupo IN (%s)" % ",".join("?" *
len(grupos_selecionados)))
            params.extend(grupos_selecionados)

        if unidades_selecionadas:
            condicoes.append("unidade IN (%s)" % ",".join("?" *
len(unidades_selecionadas)))
            params.extend(unidades_selecionadas)

        where_clause = "WHERE " + " AND ".join(condicoes) if condicoes else ""

        carregar_estoque(where_clause, params)

    def ordenar(coluna, ordem):
        campos = {
            'ID': 'id',
            'Nome do Insumo': 'nome',
            'Grupo': 'grupo',
            'Quantidade': 'quantidade'
        }

        campo = campos.get(coluna)
        if not campo:
            return

        grupos_selecionados = [grupos_listbox.get(i) for i in
                                grupos_listbox.curselection()]
        unidades_selecionadas = [unidades_listbox.get(i) for i in
                                  unidades_listbox.curselection()]

        condicoes = []
        params = []

        if grupos_selecionados:
            condicoes.append("grupo IN (%s)" % ",".join("?" *
len(grupos_selecionados)))
            params.extend(grupos_selecionados)

        if unidades_selecionadas:
            condicoes.append("unidade IN (%s)" % ",".join("?" *
len(unidades_selecionadas)))
            params.extend(unidades_selecionadas)

        where_clause = "WHERE " + " AND ".join(condicoes) if condicoes else ""

        carregar_estoque(f"{where_clause} ORDER BY {campo} {ordem}",

```

```

params)

def menu_ordenacao(event):
    region = tree.identify_region(event.x, event.y)
    if region == "heading":
        coluna = tree.identify_column(event.x)
        indice = int(coluna[1:]) - 1
        nome_coluna = colunas[indice]

        menu = tk.Menu(janela_estoque, tearoff=0)
        if nome_coluna in ['Nome do Insumo', 'Grupo']:
            menu.add_command(label="Ordenar A → Z", command=lambda:
ordenar(nome_coluna, "ASC"))
            menu.add_command(label="Ordenar Z → A", command=lambda:
ordenar(nome_coluna, "DESC"))
        elif nome_coluna in ['ID', 'Quantidade']:
            menu.add_command(label="Maior → Menor", command=lambda:
ordenar(nome_coluna, "DESC"))
            menu.add_command(label="Menor → Maior", command=lambda:
ordenar(nome_coluna, "ASC"))
        try:
            menu.post(event.x_root, event.y_root)
        finally:
            menu.grab_release()

# Criar janela
janela_estoque = tk.Toplevel()
janela_estoque.title("Visualização de Estoque")
janela_estoque.geometry("800x500")

# Frame de filtros
frame_filtros = tk.Frame(janela_estoque)
frame_filtros.pack(pady=10)

# Carregar grupos e unidades únicas
try:
    conn = sqlite3.connect('gestao_agricola.db')
    cursor = conn.cursor()
    cursor.execute('SELECT DISTINCT grupo FROM insumos')
    grupos = sorted([row[0] for row in cursor.fetchall()])
    cursor.execute('SELECT DISTINCT unidade FROM insumos')
    unidades = sorted([row[0] for row in cursor.fetchall()])
    conn.close()
except Exception as e:
    messagebox.showerror("Erro", f"Erro ao carregar dados de filtro:
{e}")

    return

# Filtro de Grupos
tk.Label(frame_filtros, text="Grupo(s):").grid(row=0, column=0, padx=5,
pady=5, sticky="w")
grupos_listbox = tk.Listbox(frame_filtros, selectmode=tk.MULTIPLE,
height=5, width=30, exportselection=False)
for grupo in grupos:
    grupos_listbox.insert(tk.END, grupo)
grupos_listbox.grid(row=1, column=0, padx=5, pady=5)

# Filtro de Unidades
tk.Label(frame_filtros, text="Unidade(s):").grid(row=0, column=1,
padx=5, pady=5, sticky="w")
unidades_listbox = tk.Listbox(frame_filtros, selectmode=tk.MULTIPLE,

```

```

height=5, width=30, exportselection=False)
    for unidade in unidades:
        unidades_listbox.insert(tk.END, unidade)
    unidades_listbox.grid(row=1, column=1, padx=5, pady=5)

    # Botão de aplicar filtros
    btn_aplicar = tk.Button(frame_filtros, text="Aplicar Filtros",
command=aplicar_filtros, width=20)
    btn_aplicar.grid(row=2, column=0, columnspan=2, pady=10)

    # Treeview
    colunas = ('ID', 'Nome do Insumo', 'Quantidade', 'Unidade', 'Grupo')
    tree = ttk.Treeview(janela_estoque, columns=colunas, show='headings')
    for coluna in colunas:
        tree.heading(coluna, text=coluna)
        tree.column(coluna, width=130)
    tree.pack(fill=tk.BOTH, expand=True)

    tree.bind("<Button-3>", menu_ordenacao)

    tk.Button(janela_estoque, text="Atualizar Estoque",
command=carregar_estoque).pack(pady=10)

carregar_estoque()

```

A função `abrir_visualizacao_estoque()` é responsável por criar uma interface visual onde o usuário pode consultar os insumos disponíveis no estoque, com possibilidade de aplicar filtros e ordenar os resultados. Ao ser chamada, ela abre uma nova janela que contém uma tabela (Treeview) exibindo os dados de cada insumo — como ID, nome, quantidade, unidade e grupo — diretamente do banco de dados SQLite.

Essa exibição inicial é feita pela função `carregar_estoque()`, que executa uma consulta SQL com parâmetros opcionais de filtragem. Para refinar os resultados mostrados, o usuário pode selecionar múltiplos grupos e unidades por meio de duas listas e clicar em “Aplicar Filtros”; isso executa a função `aplicar_filtros(coluna, ordem)`, que monta uma cláusula WHERE com os critérios selecionados e recarrega os dados da tabela. A interface também oferece um recurso de ordenação: ao clicar com o botão direito do mouse no cabeçalho da tabela, o evento `menu_ordenacao(evento)` é disparado, exibindo um menu contextual que permite ordenar as colunas por ordem crescente ou decrescente, com base em seu tipo (alfabético ou numérico).

Essa ordenação é feita pela função `ordenar`, que reconstrói a consulta SQL adicionando cláusulas ORDER BY. Antes de construir os filtros, a função tenta obter do banco todos os grupos e unidades distintas para preencher dinamicamente as listas. Por fim, há um botão de “Atualizar Estoque” que recarrega a tabela com todos os dados, limpando os

filtros. Assim, essa função torna possível uma análise detalhada e dinâmica do estoque, essencial para a tomada de decisões no gerenciamento dos insumos agrícolas.

Visualização de Estoque

Grupo(s):

Unidade(s):

Aplicar Filtros

ID	Nome do Insumo	Quantidade	Unidade	Grupo
16	Calcário	900.0	Kg	Corretivos De Solo
17	Gesso Agrícola	0.0	Kg	Corretivos De Solo
18	Silicato De Cálcio E Magnésio	1000.0	Kg	Corretivos De Solo
20	Nitrogênio	950.0	Kg	Fertilizantes Minerais - Macronutrientes Primários
21	Fósforo	1000.0	Kg	Fertilizantes Minerais - Macronutrientes Primários
22	Potássio	1100.0	Kg	Fertilizantes Minerais - Macronutrientes Primários
23	Cálcio	1000.0	Kg	Fertilizantes Minerais - Macronutrientes Secundário
24	Magnésio	900.0	Kg	Fertilizantes Minerais - Macronutrientes Secundário
25	Enxofre	850.0	Kg	Fertilizantes Minerais - Macronutrientes Secundário
26	Boro	1000.0	Kg	Fertilizantes Minerais - Micronutrientes
27	Cobre	1000.0	Kg	Fertilizantes Minerais - Micronutrientes
28	Manganês	1000.0	Kg	Fertilizantes Minerais - Micronutrientes
29	Molibdênio	1000.0	Kg	Fertilizantes Minerais - Micronutrientes
30	Zinco	1350.0	Kg	Fertilizantes Minerais - Micronutrientes
31	Torta De Filtro	1000.0	Kg	Orgânicos E Biológicos - Tortas E Resíduos Da Agri
32	Vinhaça	1000.0	Kg	Orgânicos E Biológicos - Tortas E Resíduos Da Agri
33	Cinzas	1000.0	Kg	Orgânicos E Biológicos - Tortas E Resíduos Da Agri

Atualizar Estoque

Figura 4: Interface Gráfica da Função `abrir_visualizacao_estoque()`.

2.3.2.4 Função `abrir_cadastro_aplicacoes()`.

```
def abrir_cadastro_aplicacoes():
    def salvar_aplicacao():
        insumo = combo_insumo.get().strip()
        talhao = entry_talhao.get().strip()
        quantidade = entry_quantidade.get().strip()
        data_aplicacao = entry_data.get().strip()

        if not insumo or not talhao or not quantidade or not data_aplicacao:
            messagebox.showerror("Erro", "Todos os campos devem ser preenchidos.")
            return

        try:
            quantidade = float(quantidade)
        except ValueError:
            messagebox.showerror("Erro", "Quantidade deve ser um número.")
            return

        try:
            data_obj = datetime.strptime(data_aplicacao, "%d/%m/%Y")
            data_str = data_obj.strftime("%d/%m/%Y")
        except ValueError:
            messagebox.showerror("Erro", "Data deve estar no formato DD/MM/AAAA.")
            return

        try:
            conn = sqlite3.connect('gestao_agricola.db')
            cursor = conn.cursor()

            # Verificar o estoque atual do insumo
```

```

        cursor.execute('SELECT quantidade FROM insumos WHERE nome = ?',
(insumo,))
        resultado = cursor.fetchone()

        if resultado is None:
            messagebox.showerror("Erro", f"Insumo '{insumo}' não
encontrado no estoque.")
            return

        estoque_atual = resultado[0]

        if quantidade > estoque_atual:
            messagebox.showerror("Erro", f"Estoque insuficiente.
Quantidade disponível: {estoque_atual}")
            return

        hoje = datetime.now().date()
        if data_obj.date() <= hoje:
            # Aplicação imediata
            cursor.execute('''
                                INSERT INTO aplicacoes (insumo, talhao,
quantidade, data_aplicacao)
                                VALUES (?, ?, ?, ?)
                                ''', (insumo, talhao, quantidade, data_str))

            novo_estoque = estoque_atual - quantidade
            cursor.execute('UPDATE insumos SET quantidade = ? WHERE
nome = ?', (novo_estoque, insumo))
        else:
            # Aplicação futura - reservar estoque
            cursor.execute('''
                                INSERT INTO estoque_reservado (insumo,
quantidade, data_aplicacao, talhao)
                                VALUES (?, ?, ?, ?)
                                ''', (insumo, quantidade, data_str, talhao))

            novo_estoque = estoque_atual - quantidade
            cursor.execute('UPDATE insumos SET quantidade = ? WHERE
nome = ?', (novo_estoque, insumo))

        conn.commit()
        messagebox.showinfo("Sucesso", "Aplicação registrada com
sucesso.")
        entry_talhao.delete(0, tk.END)
        entry_quantidade.delete(0, tk.END)
        entry_data.delete(0, tk.END)
    except Exception as e:
        messagebox.showerror("Erro", f"Ocorreu um erro ao salvar a
aplicação: {e}")
    finally:
        conn.close()

janela_aplicacoes = tk.Toplevel()
janela_aplicacoes.title("Cadastro de Aplicações")
janela_aplicacoes.geometry("600x450")

tk.Label(janela_aplicacoes, text="Nome do Insumo:").pack(pady=5)
combo_insumo = ttk.Combobox(janela_aplicacoes, width=47)
combo_insumo.pack(pady=5)

# Carregar os insumos disponíveis

```

```

try:
    conn = sqlite3.connect('gestao_agricola.db')
    cursor = conn.cursor()
    cursor.execute('SELECT nome FROM insumos')
    insumos = [row[0] for row in cursor.fetchall()]
    conn.close()
    combo_insumo['values'] = insumos
except Exception as e:
    messagebox.showerror("Erro", f"Erro ao carregar insumos: {e}")

tk.Label(janela_aplicacoes, text="Talhão:").pack(pady=5)
entry_talhao = tk.Entry(janela_aplicacoes, width=50)
entry_talhao.pack(pady=5)

tk.Label(janela_aplicacoes, text="Quantidade Aplicada:").pack(pady=5)
entry_quantidade = tk.Entry(janela_aplicacoes, width=50)
entry_quantidade.pack(pady=5)

tk.Label(janela_aplicacoes, text="Data da Aplicação
(DD/MM/AAAA):").pack(pady=5)
entry_data = tk.Entry(janela_aplicacoes, width=50)
entry_data.pack(pady=5)

tk.Button(janela_aplicacoes, text="Salvar",
command=salvar_aplicacao).pack(pady=20)

```

A função `abrir_cadastro_aplicacoes()` é responsável por abrir uma janela para o registro de aplicações de insumos nos talhões, permitindo ao usuário informar qual insumo foi aplicado, em qual talhão, a quantidade utilizada e a data da aplicação.

Ao clicar em "Salvar", a função interna `salvar_aplicacao()` é executada, realizando primeiramente a validação dos campos para garantir que nenhum esteja vazio e que a quantidade seja numérica. A data inserida também é validada e convertida para o formato padrão do banco de dados. Em seguida, o sistema verifica no banco SQLite se o insumo informado existe e se há quantidade suficiente em estoque. Caso a data da aplicação seja igual ou anterior à data atual, a aplicação é registrada imediatamente na tabela `aplicacoes`, e a quantidade do insumo é subtraída do estoque. Se a data for futura, o sistema entende que é uma aplicação agendada e registra a informação na tabela `estoque_reservado`, também descontando o insumo do estoque atual para reservar a quantidade.

A interface é composta por campos de entrada e uma Combobox que é automaticamente preenchida com os nomes dos insumos disponíveis no banco. Caso haja sucesso na operação, o sistema limpa os campos e exibe uma mensagem confirmando o cadastro; em caso de erro, exibe mensagens informativas. Com isso, a função integra o planejamento e controle de uso dos insumos no campo, oferecendo um fluxo organizado e seguro para registrar tanto aplicações imediatas quanto reservas futuras.

Figura 5: Interface Gráfica da Função `abrir_cadastro_aplicacoes()`.

2.3.2.5 Função `abrir_exclusao_aplicacao()`.

```
def abrir_exclusao_aplicacao():
    def carregar_aplicacoes_filtradas():
        listbox_aplicacoes.delete(0, tk.END)
        talhao = combo_talhao.get().strip()
        insumo = combo_insumo.get().strip()
        data_de = entry_data_de.get().strip()
        data_ate = entry_data_ate.get().strip()

        query = 'SELECT id, insumo, talhao, data_aplicacao, quantidade FROM
        aplicacoes WHERE 1=1'
        params = []

        if talhao:
            query += ' AND talhao = ?'
            params.append(talhao)
        if insumo:
            query += ' AND insumo = ?'
            params.append(insumo)
        if data_de:
            query += ' AND date(data_aplicacao) >= date(?)'
            params.append(data_de)
        if data_ate:
            query += ' AND date(data_aplicacao) <= date(?)'
            params.append(data_ate)

        query += ' ORDER BY data_aplicacao DESC'

        try:
            conn = sqlite3.connect('gestao_agricola.db')
            cursor = conn.cursor()
            cursor.execute(query, params)
            dados = cursor.fetchall()
            conn.close()

            for row in dados:
                texto = f"{row[0]} | {row[1]} | {row[2]} | {row[3]} |
                {row[4]}"
                listbox_aplicacoes.insert(tk.END, texto)
```

```

        except Exception as e:
            messagebox.showerror("Erro", f"Erro ao carregar aplicações:
{e}")

    def excluir_aplicacoes():
        selecionadas = listbox_aplicacoes.curselection()
        if not selecionadas:
            messagebox.showwarning("Aviso", "Selecione ao menos uma
aplicação para excluir.")
            return

        confirmacao = messagebox.askyesno("Confirmação", "Tem certeza que
deseja excluir as aplicações selecionadas?")
        if not confirmacao:
            return

    try:
        conn = sqlite3.connect('gestao_agricola.db')
        cursor = conn.cursor()

        for i in selecionadas:
            item = listbox_aplicacoes.get(i)
            partes = item.split('|')
            id_aplicacao = int(partes[0].strip())
            insumo = partes[1].strip()
            quantidade = float(partes[4].strip())

            cursor.execute('UPDATE insumos SET quantidade = quantidade
+ ? WHERE nome = ?', (quantidade, insumo))
            cursor.execute('DELETE FROM aplicacoes WHERE id = ?',
(id_aplicacao,))

            conn.commit()
            conn.close()
            messagebox.showinfo("Sucesso", "Aplicações excluídas e estoque
atualizado.")
            carregar_aplicacoes_filtradas()

    except Exception as e:
        messagebox.showerror("Erro", f"Erro ao excluir aplicações:
{e}")

# Janela principal
janela_exclusao = tk.Toplevel()
janela_exclusao.title("Excluir Aplicações")
janela_exclusao.geometry("800x500")

# Filtros
frame_filtros = tk.Frame(janela_exclusao)
frame_filtros.pack(pady=10)

tk.Label(frame_filtros, text="Talhão:").grid(row=0, column=0, padx=5)
combo_talhao = ttk.Combobox(frame_filtros, width=15)
combo_talhao.grid(row=0, column=1, padx=5)

tk.Label(frame_filtros, text="Insumo:").grid(row=0, column=2, padx=5)
combo_insumo = ttk.Combobox(frame_filtros, width=20)
combo_insumo.grid(row=0, column=3, padx=5)

tk.Label(frame_filtros, text="Data de (dd/mm/aaaa):").grid(row=1,
column=0, padx=5)

```

```

entry_data_de = tk.Entry(frame_filtros, width=15)
entry_data_de.grid(row=1, column=1, padx=5)

tk.Label(frame_filtros, text="Data até (dd/mm/aaaa):").grid(row=1,
column=2, padx=5)
entry_data_ate = tk.Entry(frame_filtros, width=15)
entry_data_ate.grid(row=1, column=3, padx=5)

tk.Button(frame_filtros, text="Filtrar Aplicações",
command=carregar_aplicacoes_filtradas).grid(row=0, column=4, rowspan=2,
padx=10)

# Lista de aplicações
tk.Label(janela_exclusao, text="Selecione as aplicações para
excluir:").pack()
listbox_aplicacoes = tk.Listbox(janela_exclusao,
selectmode=tk.MULTIPLE, width=110, height=15)
listbox_aplicacoes.pack(pady=10)

tk.Button(janela_exclusao, text="Excluir Seleccionadas",
command=excluir_aplicacoes).pack(pady=10)

# Preencher valores de talhões e insumos
try:
    conn = sqlite3.connect('gestao_agricola.db')
    cursor = conn.cursor()

    cursor.execute('SELECT DISTINCT talhao FROM aplicacoes')
    talhoes = [row[0] for row in cursor.fetchall()]
    combo_talhao['values'] = talhoes

    cursor.execute('SELECT DISTINCT insumo FROM aplicacoes')
    insumos = [row[0] for row in cursor.fetchall()]
    combo_insumo['values'] = insumos

    conn.close()
except Exception as e:
    messagebox.showerror("Erro", f"Erro ao carregar filtros: {e}")

# Carregar inicialmente todas
carregar_aplicacoes_filtradas()

```

A função `abrir_exclusao_aplicacao()` abre uma janela gráfica dedicada à exclusão de registros de aplicações de insumos já cadastradas, oferecendo filtros e controle direto sobre o banco de dados. A interface permite ao usuário filtrar as aplicações por talhão, insumo e intervalo de datas. A função `carregar_aplicacoes_filtradas()` é responsável por buscar os registros na tabela `aplicacoes`, de acordo com os filtros aplicados, e exibi-los em uma `Listbox`, onde cada linha contém o ID, nome do insumo, talhão, data da aplicação e quantidade utilizada.

Após o carregamento dos dados, o usuário pode selecionar um ou mais registros e acionar o botão de exclusão. Nesse momento, a função `excluir_aplicacoes()` é executada. Para cada item selecionado, o sistema identifica o ID da aplicação e o insumo correspondente, e

então atualiza o banco: primeiro, somando de volta ao estoque da tabela insumos a quantidade que havia sido utilizada na aplicação; em seguida, removendo o registro da tabela aplicacoes. Ao final do processo, é exibida uma mensagem de confirmação e a lista de aplicações é atualizada automaticamente.

A interface também carrega automaticamente os valores únicos de talhões e insumos já registrados no banco para facilitar a filtragem. Todo o processo é envolvido em tratamento de exceções, garantindo que falhas na conexão ou operações com o banco sejam informadas ao usuário com mensagens claras. Essa função, portanto, oferece um meio seguro, controlado e eficiente de desfazer aplicações, mantendo a integridade do estoque e do histórico de uso de insumos.

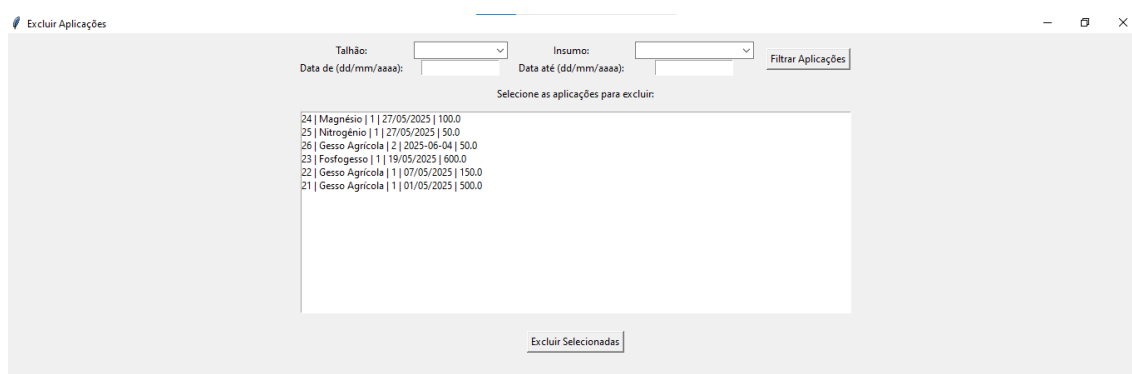


Figura 6: Interface Gráfica da Função `abrir_exclusao_aplicacao()`.

2.3.2.6 `visualizar_estoque_reservado()`.

```
def visualizar_estoque_reservado():
    def carregar_dados(ordem_coluna=None, reverso=False):
        for item in tree.get_children():
            tree.delete(item)

        try:
            conn = sqlite3.connect("gestao_agricola.db")
            cursor = conn.cursor()

            query = "SELECT id, insumo, quantidade, data_aplicacao, talhao
FROM estoque_reservado"
            if ordem_coluna:
                query += f" ORDER BY {ordem_coluna} {'DESC' if reverso else
'ASC'}"

            cursor.execute(query)
            dados = cursor.fetchall()
            conn.close()

            for linha in dados:
                tree.insert("", "end", values=linha)
```

```

except Exception as e:
    messagebox.showerror("Erro", f"Erro ao carregar dados: {e}")

def ordenar_por(coluna):
    reverso = ordenacoes.get(coluna, False)
    ordenacoes[coluna] = not reverso
    carregar_dados(ordem_coluna=coluna, reverso=reverso)

def excluir_selecionados():
    selecionados = tree.selection()
    if not selecionados:
        messagebox.showwarning("Nenhuma seleção", "Selecione pelo menos
um item para excluir.")
        return

    confirmar = messagebox.askyesno("Confirmação", "Deseja excluir os
itens selecionados?")
    if not confirmar:
        return

    try:
        conn = sqlite3.connect("gestao_agricola.db")
        cursor = conn.cursor()

        for item in selecionados:
            id_aplicacao = tree.item(item)["values"][0]
            cursor.execute("DELETE FROM estoque_reservado WHERE id =
?", (id_aplicacao,))
            tree.delete(item)

        conn.commit()
        conn.close()
        messagebox.showinfo("Sucesso", "Itens excluídos com sucesso.")

    except Exception as e:
        messagebox.showerror("Erro", f"Erro ao excluir: {e}")

# Janela principal
janela = tk.Toplevel()
janela.title("Estoque Reservado")
janela.geometry("700x450")

# Tabela
colunas = ("id", "insumo", "quantidade", "data_aplicacao", "talhao")
nomes_colunas = {
    "id": "ID",
    "insumo": "Insumo",
    "quantidade": "Quantidade",
    "data_aplicacao": "Data Aplicação",
    "talhao": "Talhão"
}

tree = ttk.Treeview(janela, columns=colunas, show="headings",
selectmode="extended")

for col in colunas:
    tree.heading(col, text=nomes_colunas[col], command=lambda c=col:
ordenar_por(c))
    tree.column(col, anchor="center", width=120)

```

```

tree.pack(fill="both", expand=True, padx=10, pady=10)

# Botão de exclusão
btn_excluir = tk.Button(janela, text="Excluir Seleccionados",
command=excluir_seleccionados)
btn_excluir.pack(pady=5)

# Inicialização
ordenacoes = {}
carregar_dados()

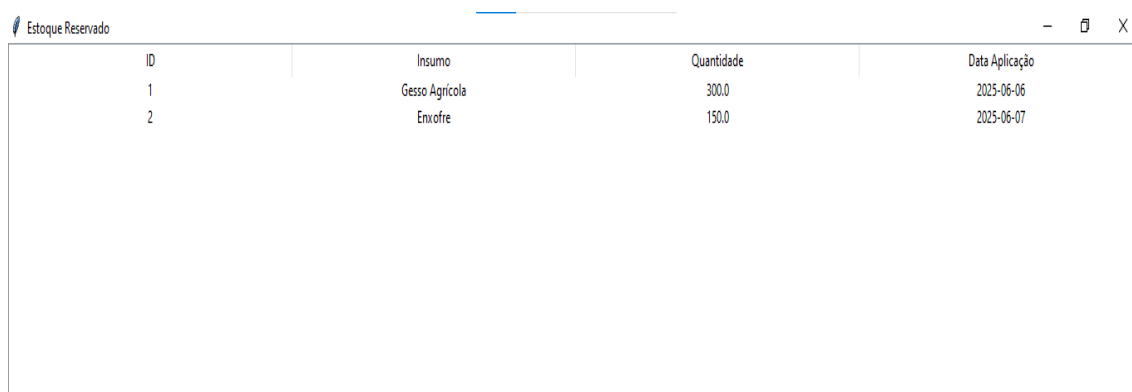
```

A função `visualizar_estoque_reservado()` exibe uma janela com uma tabela contendo os insumos que foram reservados para aplicações futuras, permitindo ao usuário consultar rapidamente o que já está comprometido do estoque atual.

Ao ser chamada, a função cria uma nova janela principal (`tk.Tk()`), define seu título e tamanho, e em seguida configura um widget Treeview do `tkk` com quatro colunas: ID, Insumo, Quantidade e Data Aplicação. Essas colunas representam, respectivamente, o identificador único do registro no banco, o nome do insumo reservado, a quantidade reservada e a data em que a aplicação está prevista.

O banco de dados `gestao_agricola.db` é acessado diretamente para buscar os dados da tabela `estoque_reservado`. Os resultados obtidos são inseridos linha por linha no Treeview, apresentando de forma clara e organizada os dados de reservas pendentes. Caso ocorra qualquer erro na conexão ou na consulta, uma mensagem de erro será exibida ao usuário usando `tk.messagebox.showerror()`.

Essa visualização é essencial para o gerenciamento eficiente dos insumos, pois permite que o agricultor saiba o que já está reservado e evite o risco de comprometer o estoque com aplicações duplicadas ou não planejadas.



ID	Insumo	Quantidade	Data Aplicação
1	Gesso Agrícola	300.0	2025-06-06
2	Enxofre	150.0	2025-06-07

Figura 7: Interface Gráfica da Função `visualizar_estoque_reservado()`.

2.3.2.7 Função abrir_relatorios_graficos().

```
def abrir_relatorios_graficos():
    def gerar_grafico():
        insumos_selecionados = [listbox_insumos.get(i) for i in
listbox_insumos.curselection()]
        talhao = combo_talhao.get().strip()
        data_inicio = entry_data_inicio.get().strip()
        data_fim = entry_data_fim.get().strip()

        if not insumos_selecionados:
            messagebox.showerror("Erro", "Selecione pelo menos um insumo.")
            return
        if not talhao:
            messagebox.showerror("Erro", "Selecione um talhão.")
            return
        if not data_inicio or not data_fim:
            messagebox.showerror("Erro", "Informe o período completo.")
            return

        try:
            data_inicio_dt = datetime.strptime(data_inicio, "%d/%m/%Y")
            data_fim_dt = datetime.strptime(data_fim, "%d/%m/%Y")
        except ValueError:
            messagebox.showerror("Erro", "Datas no formato inválido. Use
dd/mm/aaaa.")
            return

        try:
            conn = sqlite3.connect('gestao_agricola.db')
            cursor = conn.cursor()

            dados_por_insumo = {}
            datas_unicas = set()

            for insumo in insumos_selecionados:
                cursor.execute("""
SELECT data_aplicacao, SUM(quantidade)
FROM aplicacoes
WHERE insumo = ?
      AND talhao = ?
      AND data_aplicacao >= ?
      AND data_aplicacao <= ?
GROUP BY data_aplicacao
ORDER BY data_aplicacao
""", (
                    insumo,
                    talhao,
                    data_inicio_dt.strftime("%d/%m/%Y"),
                    data_fim_dt.strftime("%d/%m/%Y")
                ))
                dados = cursor.fetchall()
                dados_por_insumo[insumo] = {row[0]: row[1] for row in
dados}

                datas_unicas.update(row[0] for row in dados)

            conn.close()

            if not datas_unicas:
                messagebox.showinfo("Sem dados", "Nenhuma aplicação
encontrada no período selecionado.")
```

```

        return

        datas_ordenadas = sorted(datas_unicas, key=lambda d:
datetime.strptime(d, "%d/%m/%Y"))
        datas_formatadas = [datetime.strptime(data,
"%d/%m/%Y").strftime("%d/%m") for data in datas_ordenadas]

        # Criar gráfico interativo com Plotly
        fig = go.Figure()

        for insumo in insumos_selecionados:
            quantidades = [dados_por_insumo[insumo].get(data, 0) for
data in datas_ordenadas]
            fig.add_trace(go.Bar(
                x=datas_formatadas,
                y=quantidades,
                name=insumo,
                text=[f"{q:.2f}" for q in quantidades],
                textposition="auto"
            ))

        fig.update_layout(
            title=f"Aplicações por Insumo - Talhão: {talhao}",
            xaxis_title="Data",
            yaxis_title="Quantidade Aplicada",
            barmode='group',
            template='plotly_white',
            legend_title_text='Insumos'
        )

        with tempfile.NamedTemporaryFile('w', delete=False,
suffix='.html') as f:
            fig.write_html(f.name)
            webbrowser.open(f.name)

    except Exception as e:
        messagebox.showerror("Erro", f"Erro ao gerar gráfico:\n{e}")

# Janela principal
janela_relatorios = tk.Toplevel()
janela_relatorios.title("Relatórios e Gráficos")
janela_relatorios.geometry("800x600")

# Talhão
tk.Label(janela_relatorios, text="Selecione o Talhão:").pack()
combo_talhao = ttk.Combobox(janela_relatorios, width=52)
combo_talhao.pack(pady=5)

# Datas
frame_datas = tk.Frame(janela_relatorios)
frame_datas.pack(pady=5)

tk.Label(frame_datas, text="Data Início (dd/mm/aaaa):").grid(row=0,
column=0)
entry_data_inicio = tk.Entry(frame_datas)
entry_data_inicio.grid(row=0, column=1, padx=10)

tk.Label(frame_datas, text="Data Fim (dd/mm/aaaa):").grid(row=0,
column=2)
entry_data_fim = tk.Entry(frame_datas)
entry_data_fim.grid(row=0, column=3)

```

```

# Lista de insumos (criar antes de popular!)
tk.Label(janela_relatorios, text="Selecione os Insumos:").pack()
listbox_insumos = tk.Listbox(janela_relatorios, selectmode=tk.MULTIPLE,
width=50, height=10)
listbox_insumos.pack(pady=5)

# Botão
tk.Button(janela_relatorios, text="Gerar Gráfico",
command=gerar_grafico).pack(pady=10)

# Área do gráfico
frame_grafico = tk.Frame(janela_relatorios)
frame_grafico.pack(fill=tk.BOTH, expand=True)

# Carregar dados do banco
try:
    conn = sqlite3.connect('gestao_agricola.db')
    cursor = conn.cursor()

    # Insumos
    cursor.execute('SELECT DISTINCT insumo FROM aplicacoes')
    insumos = [row[0] for row in cursor.fetchall()]
    for insumo in insumos:
        listbox_insumos.insert(tk.END, insumo)

    # Talhões
    cursor.execute('SELECT DISTINCT talhao FROM aplicacoes')
    talhoes = [row[0] for row in cursor.fetchall()]
    combo_talhao['values'] = talhoes

    conn.close()
except Exception as e:
    messagebox.showerror("Erro", f"Erro ao carregar dados: {e}")

```

A função `abrir_relatorios_graficos` cria uma interface gráfica utilizando Tkinter para gerar gráficos interativos com a biblioteca Plotly, permitindo ao usuário analisar a aplicação de insumos em diferentes talhões ao longo do tempo.

Ao abrir a janela, o usuário encontra campos para selecionar o talhão desejado, definir um intervalo de datas e escolher os insumos que deseja incluir no gráfico. Esses elementos são implementados com widgets como `ttk.Combobox` para seleção de talhões e `tk.Listbox` para a escolha múltipla de insumos. Os campos de entrada de datas utilizam o formato "dd/mm/aaaa" para facilitar a compreensão.

O botão "Gerar Gráfico" aciona a função `gerar_grafico`, que realiza a validação dos dados inseridos. Ele verifica se pelo menos um insumo foi selecionado, se um talhão foi escolhido e se as datas de início e fim foram fornecidas corretamente. As datas são convertidas para objetos `datetime` para garantir a consistência no formato.

Após a validação, a função conecta-se ao banco de dados SQLite 'gestao_agricola.db' e executa consultas SQL para obter a soma das quantidades aplicadas de cada insumo selecionado no talhão especificado, dentro do intervalo de datas definido. Os resultados são organizados em um dicionário, associando cada insumo às suas respectivas datas de aplicação e quantidades.

Com os dados organizados, a função utiliza a biblioteca Plotly para criar um gráfico de barras agrupadas. Cada insumo é representado por uma série de barras, mostrando as quantidades aplicadas em cada data. As datas são formatadas para exibição no eixo X, e as quantidades no eixo Y. O gráfico é personalizado com títulos e legendas para facilitar a interpretação.

Por fim, o gráfico é salvo como um arquivo HTML temporário e aberto automaticamente no navegador padrão do usuário, permitindo uma visualização interativa dos dados. Essa abordagem aproveita os recursos do Plotly para fornecer uma análise visual dinâmica e informativa das aplicações de insumos.

Seleção de Insumos:

- Gesso Agrícola
- Fosfogesso
- Magnésio
- Nitrogénio

Gerar Gráfico

Figura 8: Interface Gráfica da Função `abrir_relatorios_graficos()`.

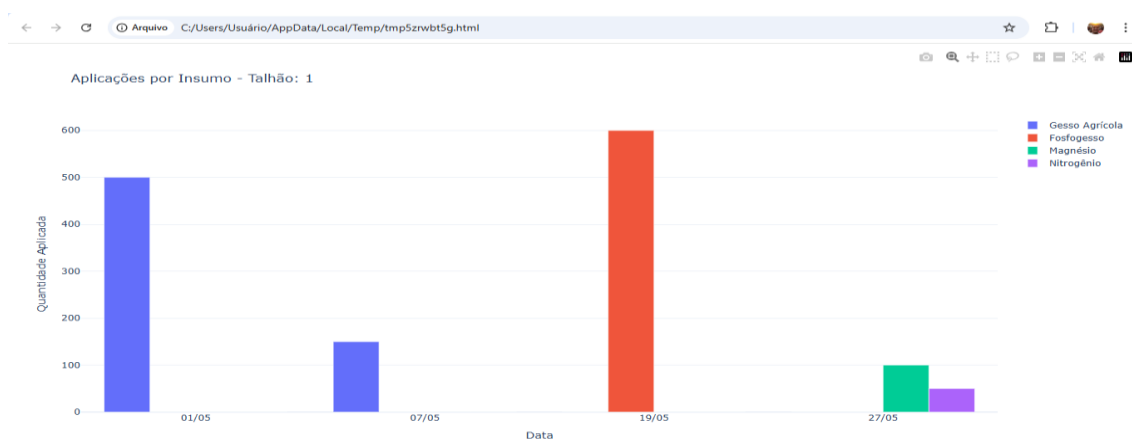


Figura 9: Modelo do Gráfico de Barras Gerado

2.3.2.8 Função abrir_relatorio_insumos_por_talhao().

```
def abrir_relatorio_insumos_por_talhao():
    def gerar_grafico():
        data = entrada_data.get().strip()
        talhao = entrada_talhao.get().strip()

        if not data or not talhao:
            messagebox.showwarning("Campos obrigatórios", "Por favor,
preencha a data e o talhão.")
            return

        try:
            # Conectar ao banco de dados
            conn = sqlite3.connect('gestao_agricola.db')
            cursor = conn.cursor()

            # Consultar os dados de aplicações para a data e talhão
            selecionados
            cursor.execute("""
                SELECT insumo, SUM(quantidade) as total
                FROM aplicacoes
                WHERE data_aplicacao = ? AND talhao = ?
                GROUP BY insumo
            """, (data, talhao))
            resultados = cursor.fetchall()
            conn.close()

            if not resultados:
                messagebox.showinfo("Sem dados", "Nenhum insumo encontrado
para os filtros selecionados.")
                return

            # Preparar dados para o gráfico
            insumos = [row[0] for row in resultados]
            quantidades = [row[1] for row in resultados]
            total_aplicado = sum(quantidades)
            percentagens = [(q / total_aplicado) * 100 for q in
quantidades]

            hover_text = [f"{insumo}<br>{quantidade:.2f}
unidades<br>{porcentagem:.1f}%"
                           for insumo, quantidade, percentagem in
zip(insumos, quantidades, percentagens)]

            # Criar gráfico de pizza interativo com Plotly
            fig = go.Figure(data=[go.Pie(
                labels=insumos,
                values=quantidades,
                hoverinfo='text',
                textinfo='percent',
                textfont_size=14,
                hovertext=hover_text
            )])
            fig.update_layout(
                title=f"Insumos aplicados em {talhao} em {data}",
                showlegend=True
            )

            # Salvar o gráfico como um arquivo HTML temporário e abrir no
navegador
            with tempfile.NamedTemporaryFile('w', delete=False,
```

```

suffix='.html') as f:
    fig.write_html(f.name)
    webbrowser.open(f.name)

    except Exception as e:
        messagebox.showerror("Erro", f"Ocorreu um erro ao gerar o
gráfico:\n{e}")

# Criar janela do relatório
janela_relatorio = tk.Toplevel()
janela_relatorio.title("Relatório de Insumos por Talhão e Data")
janela_relatorio.geometry("400x200")

# Frame para os filtros
frame_filtros = tk.Frame(janela_relatorio)
frame_filtros.pack(pady=10)

# Campo para data
tk.Label(frame_filtros, text="Data (AAAA-MM-DD):").grid(row=0,
column=0, padx=5, pady=5, sticky="e")
entrada_data = tk.Entry(frame_filtros)
entrada_data.grid(row=0, column=1, padx=5, pady=5)

# Campo para talhão
tk.Label(frame_filtros, text="Talhão:").grid(row=1, column=0, padx=5,
pady=5, sticky="e")
entrada_talhao = tk.Entry(frame_filtros)
entrada_talhao.grid(row=1, column=1, padx=5, pady=5)

# Botão para gerar gráfico
btn_gerar = tk.Button(janela_relatorio, text="Gerar Gráfico",
command=gerar_grafico)
btn_gerar.pack(pady=10)

```

A função `abrir_relatorio_insumos_por_talhao` cria uma interface gráfica com Tkinter para permitir ao usuário gerar um gráfico de pizza que mostra a distribuição de insumos aplicados em um talhão específico em uma data determinada.

A interface consiste em uma janela com dois campos de entrada: um para a data no formato "AAAA-MM-DD" e outro para o nome do talhão. Após o preenchimento desses campos, o botão "Gerar Gráfico" aciona a função `gerar_grafico`.

Dentro dessa função, os dados inseridos pelo usuário são primeiramente validados. Se a data ou o talhão estiverem em branco, uma mensagem de aviso é exibida. Caso os campos estejam preenchidos corretamente, o sistema se conecta ao banco de dados SQLite `gestao_agricola.db` e executa uma consulta SQL para obter o total de insumos aplicados no talhão e data informados, agrupando os resultados por tipo de insumo.

Se não houver dados correspondentes à consulta, o sistema informa que não foram encontrados insumos para os filtros selecionados. Caso contrário, os dados retornados são

processados para calcular as porcentagens de cada insumo com base na quantidade total aplicada. Esses dados são usados para montar o gráfico de pizza com a biblioteca Plotly, onde cada fatia representa a participação percentual de um insumo. O gráfico inclui informações interativas com nome do insumo, quantidade aplicada e sua porcentagem no total.

Por fim, o gráfico é salvo como um arquivo HTML temporário e aberto automaticamente no navegador padrão do usuário, permitindo uma visualização interativa clara e intuitiva da distribuição dos insumos utilizados. Essa funcionalidade é útil para relatórios rápidos de aplicação por talhão em datas específicas.

A interface gráfica da função `abrir_relatorio_insumos_por_talhao()` é uma janela com o título "Relatório de Insumos por Talhão e Data". No topo, há uma barra de busca com o ícone de lupa. Abaixo, há dois campos de entrada: "Data (AAAA-MM-DD):" e "Talhão:". Abaixo dos campos, há um botão "Gerar Gráfico".

Figura 10: Interface Gráfica da Função `abrir_relatorio_insumos_por_talhao()`.

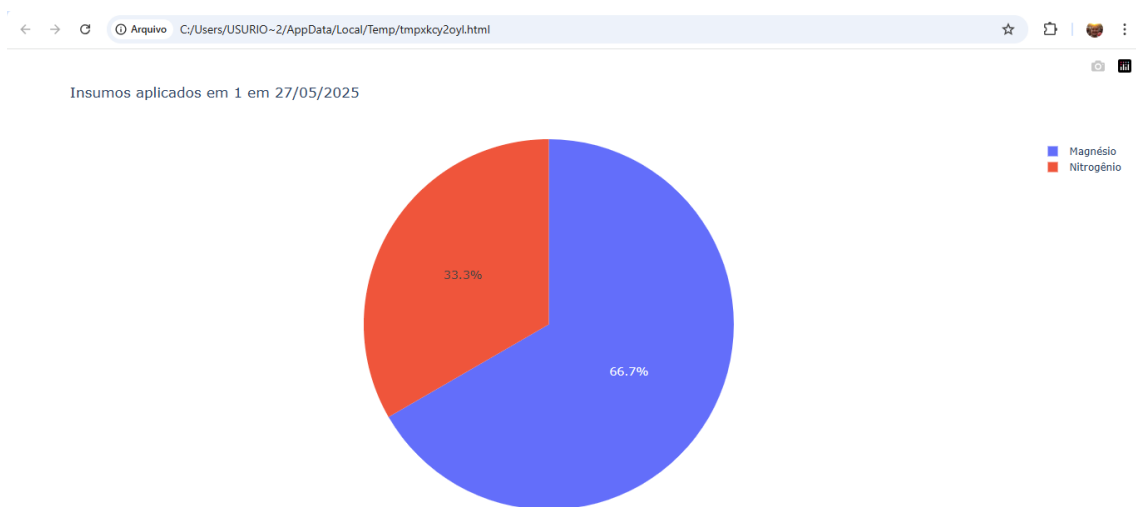


Figura 11: Modelo do Gráfico de setores gerado

3 CONSIDERAÇÕES FINAIS

As considerações finais deste Trabalho de Conclusão de Curso refletem a jornada empreendida no desenvolvimento de uma aplicação desktop voltada à gestão agrícola, utilizando a linguagem Python em conjunto com as bibliotecas Tkinter e SQLite. O objetivo principal foi criar uma ferramenta que facilitasse o controle de insumos, aplicações e estoques, proporcionando aos usuários uma interface intuitiva e funcionalidades eficientes para a administração das atividades agrícolas.

Ao longo do projeto, foram implementadas diversas funcionalidades essenciais, como o registro e exclusão de aplicações, visualização de estoques reservados e geração de relatórios gráficos. Cada uma dessas funcionalidades foi cuidadosamente planejada e desenvolvida para atender às necessidades específicas do gerenciamento agrícola, permitindo uma visão abrangente e detalhada das operações realizadas.

A implementação da função de exclusão de aplicações, por exemplo, não apenas remove registros do banco de dados, mas também atualiza automaticamente o estoque de insumos, garantindo a integridade e a consistência das informações. A visualização dos estoques reservados oferece uma perspectiva clara sobre os recursos disponíveis, auxiliando na tomada de decisões estratégicas. Além disso, a geração de relatórios gráficos proporciona uma análise visual das aplicações por insumo e talhão, facilitando a identificação de padrões e tendências ao longo do tempo.

Durante o desenvolvimento do sistema, foram enfrentados desafios relacionados à manipulação de dados, tratamento de exceções e criação de interfaces gráficas amigáveis. A superação desses obstáculos contribuiu significativamente para o aprimoramento das habilidades técnicas e para a consolidação do conhecimento adquirido ao longo do curso.

Este trabalho destaca-se por sua relevância prática, oferecendo uma solução concreta para a gestão eficiente de recursos agrícolas. A aplicação desenvolvida pode ser adaptada e expandida para atender a diferentes contextos e necessidades, demonstrando sua versatilidade e potencial de aplicação no setor agrícola.

Reconhecendo as limitações do projeto, como a ausência de funcionalidades avançadas de análise de dados e integração com outras plataformas, sugere-se, para trabalhos futuros, a implementação de recursos adicionais, como a previsão de consumo de insumos, alertas automáticos para reposição de estoque e integração com sistemas de georreferenciamento. Essas melhorias poderiam ampliar ainda mais a utilidade e a eficácia da aplicação, proporcionando uma ferramenta ainda mais robusta para o gerenciamento agrícola.

Em suma, este Trabalho de Conclusão de Curso representa uma contribuição significativa para a área de sistemas de informação aplicados à agricultura, evidenciando a importância da tecnologia na otimização e no controle das atividades do setor. A experiência adquirida durante o desenvolvimento deste projeto servirá como base sólida para futuras iniciativas profissionais e acadêmicas, reforçando o compromisso com a inovação e a excelência na área de tecnologia aplicada à agricultura e sustentabilidade.

REFERÊNCIAS

BALLOU, R. H. *Gerenciamento da cadeia de suprimentos: logística empresarial*. 5. ed. Porto Alegre: Bookman, 2006.

BATEMAN, T. S.; SNELL, S. A. *Administração: construindo vantagem competitiva*. São Paulo: Atlas, 1998.

BORGES, L. C.; NASCIMENTO, A. dos R.; MORGADO, C. M. A. Agricultura de precisão: ferramenta de gestão na rentabilidade e produtividade de grãos. *Scientific Electronic Archives*, v. 15, n. 3, 2022. Disponível em: <https://scientificelectronicarchives.org/index.php/SEA/article/view/1520> . Acesso em: 7 jun. 2025.

BRID SOLUÇÕES. Gestão de estoque no agro. 2022. Disponível em: <https://bridsolucoes.com.br/2022/08/30/gestao-de-estoque-no-agro/> . Acesso em: 7 jun. 2025.

CASAROTTO, E. L.; BINOTTO, E.; MARTÍNEZ, M. P.; CUNHA, G. Evidências de utilização de Big Data no agronegócio. *Revista Terceira Margem Amazônia*, v. 8, n. 19, 2022. Disponível em: <https://www.revistaterceiramargem.com/index.php/terceiramargem/article/view/501> . Acesso em: 7 jun. 2025.

DATA CAMP. Para que o Python é usado? 7 usos reais do Python. 2025. Disponível em: <https://www.datacamp.com/pt/blog/what-is-python-used-for> . Acesso em: 7 jun. 2025.

ECONT SISTEMAS. Gestão de estoque rural: benefícios e as melhores práticas. 2023. Disponível em: <https://econt.com.br/agro/gestao/gestao-de-estoque-rural-beneficios-e-as-melhores-praticas/> . Acesso em: 7 jun. 2025.

EJNISMAN, M. W.; BATTILANA, C. C. H.; ANDRADE, T. B. de. O aumento do uso de tecnologia no agronegócio: uma análise sob a ótica da proteção de dados. *Revista TECCOGS*, v. 20, p. 113-124, 2019. Disponível em: <https://revistas.pucsp.br/index.php/teccogs/article/view/48562> . Acesso em: 7 jun. 2025.

FERNANDES, F. SQLite: o banco de dados subestimado. *Medium*, 2023. Disponível em: <https://medium.com/@fabiojmf/sqlite-o-banco-de-dados-subestimado-cc96386c2f5c> . Acesso em: 7 jun. 2025.

FOGANHOLLI, M. S. Gestão de estoque: a importância da organização e planejamento. Universidade Federal do Paraná, 2023. Disponível em: <https://acervodigital.ufpr.br/handle/1884/84143> . Acesso em: 7 jun. 2025.

PYTHON SOFTWARE FOUNDATION. Python. 2025. Disponível em: <https://www.python.org/> . Acesso em: 7 jun. 2025.

SCICROP. A adoção de analytics no agro e por que ensinamos Python. 2019. Disponível em: <https://scicrop.com/2019/07/09/a-adocao-de-analytics-no-agro-e-por-que-ensinamos-python/> . Acesso em: 7 jun. 2025.