Relatório de Aprimoramento da Interface Gráfica (GUI)

Este relatório detalha as melhorias e reestruturações realizadas na interface gráfica do sistema ERP, atendendo às solicitações de uma tela de login, uma tela principal com barra de menu e menus cascata para diversas funcionalidades.

1. Implementação da Tela de Login (`gui/login_app.py`)

O que foi feito:

Foi criado um novo arquivo `login_app.py` para gerenciar o processo de autenticação do usuário antes de acessar a aplicação principal. Esta tela agora é o ponto de entrada do sistema.

Como foi feito:

- **Criação da Classe `LoginApp`:** Uma classe `LoginApp` foi definida para encapsular a lógica e os elementos da interface da tela de login.
- **Campos de Entrada:** Foram adicionados campos (`Entry`) para `Usuário` e `Senha`, com o campo de senha configurado para ocultar os caracteres (`show="*"`).
- **Botão de Login:** Um botão `Login` foi implementado para acionar a função de autenticação.
- **Integração com `UserManager`:** A lógica de autenticação utiliza o `UserManager` (já existente na camada de lógica de negócios) para verificar as credenciais do usuário. Isso garante que o hashing de senhas seja utilizado para segurança.
- **Criação de Usuário Admin Padrão:** Para facilitar o primeiro uso, foi adicionada uma lógica para criar um usuário `admin` padrão (`admin/admin123`) se nenhum usuário for encontrado no banco de dados. Isso é útil para demonstração e

configuração inicial, mas deve ser removido ou modificado em um ambiente de produção.

- **Lançamento da Aplicação Principal:** Após um login bem-sucedido, a janela de login é destruída (`self.master.destroy()`) e a aplicação principal (`main_app.py`) é lançada usando `subprocess.Popen` . O uso de `subprocess.Popen` com `sys.executable` e o argumento `-m` é crucial para garantir que o `main_app.py` seja executado como um módulo, permitindo que suas importações relativas funcionem corretamente dentro da estrutura de pacotes do projeto.

Exemplo de Código (`gui/login_app.py`):

```
 import tkinter as tk
from tkinter import messagebox
from ..business_logic.user_manager import UserManager
from ..database.database_manager import DatabaseManager
import subprocess
import sys

class LoginApp:
    def __init__(self, master):
        self.master = master
        master.title("Login - SoftX ERP")
        master.geometry("300x200")
        master.resizable(False, False)

        self.user_manager = UserManager()
        self.db_manager = DatabaseManager()
        self.db_manager.create_tables() # Ensure tables are created on startup

        self.username_label = tk.Label(master, text="Usuário:")
        self.username_label.pack(pady=5)
        self.username_entry = tk.Entry(master)
        self.username_entry.pack(pady=5)

        self.password_label = tk.Label(master, text="Senha:")
        self.password_label.pack(pady=5)
        self.password_entry = tk.Entry(master, show="*")
        self.password_entry.pack(pady=5)

        self.login_button = tk.Button(master, text="Login", command=self.login)
        self.login_button.pack(pady=10)

    def login(self):
        username = self.username_entry.get()
        password = self.password_entry.get()

        # For demonstration, let\'s create a default admin user if none exists
        users = self.user_manager.get_all_users()
        if not users:
            from ..models.models import User
            default_admin = User(nome_usuario="admin",
cpf_usuario="00000000000", email_usuario="admin@softx.com",
                                 telefone_usuario="999999999",
data_nascimento="2000-01-01", rua="Rua Admin",
                                 cep="00000000", bairro="Centro",
cidade="Cidade Admin",
                                 senha="admin123", tipo="admin",
permissao="avancado")
            self.user_manager.add_user(default_admin)
            messagebox.showinfo("Info", "Usuário admin padrão criado:
admin/admin123")

        # Authenticate user
        authenticated_user = None
        for user in self.user_manager.get_all_users():
            if user.nome_usuario == username and
self.user_manager.check_password(password, user.senha):
                authenticated_user = user
                break

        if authenticated_user:
            messagebox.showinfo("Sucesso", f"Bem-vindo,
```

```
{authenticated_user.nome_usuario}!")
            self.master.destroy() # Close login window
            subprocess.Popen([sys.executable, "-m", "gui.main_app"],
cwd="erp_refatorado")
        else:
            messagebox.showerror("Erro", "Usuário ou senha inválidos.")

if __name__ == "__main__":
    root = tk.Tk()
    app = LoginApp(root)
    root.mainloop()
```

2. Reestruturação da Tela Principal (`gui/main_app.py`)

A tela principal foi significativamente modificada para incorporar a barra de menu e os menus cascata, substituindo a abordagem anterior baseada em `ttk.Notebook`.

O que foi feito:

- **Título da Janela:** O título da janela principal foi definido como "SoftX ERP".
- **Barra de Menu (`MenuBar`):** Uma barra de menu foi adicionada na parte superior da janela.
- **Menus Cascata:** Foram criados menus cascata para:
  - **Cadastro:** Contendo submenus para `Usuário`, `Cliente`, `Fornecedor` e `Produto`.
  - **Consulta:** Contendo submenus para `Usuário`, `Cliente`, `Fornecedor` e `Produto`.
  - **Financeiro:** Com itens de menu de exemplo (`Contas a Pagar`, `Contas a Receber`) que exibem uma mensagem de "Funcionalidade em desenvolvimento".
  - **Vendas:** Com itens de menu para `Nova Venda` e `Histórico de Vendas` (este último também com mensagem de desenvolvimento).
- **Substituição de `ttk.Notebook` por `Frame`s:** O componente `ttk.Notebook` (que criava abas) foi removido. Agora, cada seção (Cadastro de Cliente, Cadastro de Usuário, etc.) é um `Frame` Tkinter separado. A visibilidade desses `Frame`s é controlada pela função `show_frame`.
- **Função `show_frame`:** Esta função é responsável por ocultar o `Frame` atualmente visível e exibir o `Frame` correspondente à opção de menu selecionada. Ela

também repopula as listas (`Treeview`) e comboboxes (`Combobox`) de cada `Frame` ao ser exibido, garantindo que os dados estejam sempre atualizados.

Como foi feito:

- **`menubar = Menu(self.root)`:** Cria a barra de menu principal.
- **`menubar.add_cascade(label="Cadastro", menu=self.cadastro_menu)`:** Adiciona um menu de nível superior (`Cadastro`) e associa a ele um submenu (`self.cadastro_menu`).
- **`self.cadastro_menu.add_command(label="Usuário", command=lambda: self.show_frame("user_cadastro"))`:** Adiciona um item ao submenu `Cadastro`. Quando clicado, ele chama `show_frame` passando o nome do `Frame` a ser exibido (`user_cadastro`).
- **Gerenciamento de `Frame`s:** Um dicionário `self.frames` armazena instâncias de `Frame` para cada seção. Por exemplo, `self.frames["client_cadastro"] = Frame(self.root)`. Para as telas de consulta, os mesmos `Frame`s de cadastro são reutilizados, pois já contêm a funcionalidade de busca e listagem.
- **Lógica de `show_frame`:** `python def show_frame(self, frame_name): if self.current_frame: self.current_frame.pack_forget() # Oculta o frame atual frame = self.frames[frame_name] frame.pack(pady=10, expand=True, fill="both") # Exibe o novo frame self.current_frame = frame # Lógica para repopular dados ao exibir o frame # ...`

Exemplo de Código (`gui/main_app.py` - trechos relevantes):

```python
# ... imports ...

class Application:
    def __init__(self):
        # ... inicializações ...
        self.current_frame = None
        self.frames = {}
        self.setup_gui()
        self.root.mainloop()

    def setup_gui(self):
        self.root.title("SoftX ERP")
        self.root.geometry("800x600")

        menubar = Menu(self.root)
        self.root.config(menu=menubar)

        # File Menu
        file_menu = Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Arquivo", menu=file_menu)
        file_menu.add_command(label="Sair", command=self.root.quit)

        # Cadastro Menu
        self.cadastro_menu = Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Cadastro", menu=self.cadastro_menu)
        self.cadastro_menu.add_command(label="Usuário", command=lambda:
self.show_frame("user_cadastro"))
        self.cadastro_menu.add_command(label="Cliente", command=lambda:
self.show_frame("client_cadastro"))
        self.cadastro_menu.add_command(label="Fornecedor", command=lambda:
self.show_frame("supplier_cadastro"))
        self.cadastro_menu.add_command(label="Produto", command=lambda:
self.show_frame("product_cadastro"))

        # Consulta Menu
        self.consulta_menu = Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Consulta", menu=self.consulta_menu)
        self.consulta_menu.add_command(label="Usuário", command=lambda:
self.show_frame("user_cadastro")) # Reusing frame
        self.consulta_menu.add_command(label="Cliente", command=lambda:
self.show_frame("client_cadastro")) # Reusing frame
        self.consulta_menu.add_command(label="Fornecedor", command=lambda:
self.show_frame("supplier_cadastro")) # Reusing frame
        self.consulta_menu.add_command(label="Produto", command=lambda:
self.show_frame("product_cadastro")) # Reusing frame

        # Financeiro Menu
        self.financeiro_menu = Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Financeiro", menu=self.financeiro_menu)
        self.financeiro_menu.add_command(label="Contas a Pagar",
command=lambda: GUIComponents.show_info("Financeiro", "Funcionalidade em
desenvolvimento."))
        self.financeiro_menu.add_command(label="Contas a Receber",
command=lambda: GUIComponents.show_info("Financeiro", "Funcionalidade em
desenvolvimento."))

        # Vendas Menu
        self.vendas_menu = Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Vendas", menu=self.vendas_menu)
        self.vendas_menu.add_command(label="Nova Venda", command=lambda:
self.show_frame("sale"))
```

```
        self.vendas_menu.add_command(label="Histórico de Vendas",
command=lambda: GUIComponents.show_info("Vendas", "Funcionalidade em
desenvolvimento."))

        # Initialize frames for each section
        self.frames["client_cadastro"] = Frame(self.root)
        self.frames["user_cadastro"] = Frame(self.root)
        self.frames["supplier_cadastro"] = Frame(self.root)
        self.frames["product_cadastro"] = Frame(self.root)
        self.frames["sale"] = Frame(self.root)

        # Create content for each frame
        self.create_client_tab(self.frames["client_cadastro"])
        self.create_user_tab(self.frames["user_cadastro"])
        self.create_supplier_tab(self.frames["supplier_cadastro"])
        self.create_product_tab(self.frames["product_cadastro"])
        self.create_sale_tab(self.frames["sale"])

        # Show initial frame (e.g., client frame)
        self.show_frame("client_cadastro")

    def show_frame(self, frame_name):
        if self.current_frame:
            self.current_frame.pack_forget()
        frame = self.frames[frame_name]
        frame.pack(pady=10, expand=True, fill="both")
        self.current_frame = frame

        # Repopulate lists when showing a frame to ensure data is fresh
        if frame_name == "client_cadastro":
            self.populate_client_list()
        elif frame_name == "user_cadastro":
            self.populate_user_list()
        elif frame_name == "supplier_cadastro":
            self.populate_supplier_list()
            self.populate_supplier_combobox() # For product tab
        elif frame_name == "product_cadastro":
            self.populate_product_list()
            self.populate_supplier_combobox()
        elif frame_name == "sale":
            self.populate_client_combobox()
            self.populate_product_combobox()

    # ... funções create_client_tab, create_user_tab, etc. (permanecem as
mesmas) ...

if __name__ == "__main__":
    # This part will be handled by login_app.py
    pass
```

3. Integração das Funcionalidades Existentes

As funcionalidades de CRUD (Criar, Ler, Atualizar, Deletar) para Clientes, Usuários, Fornecedores e Produtos, bem como a tela de Vendas, foram integradas à nova estrutura de menus.

O que foi feito:

- **Reutilização de Funções:** As funções `create_client_tab`, `create_user_tab`, `create_supplier_tab`, `create_product_tab` e `create_sale_tab` (que constroem a interface de cada seção) foram mantidas e agora são chamadas durante a inicialização da aplicação para criar os `Frame`s correspondentes.
- **Vinculação de Comandos:** Os comandos dos itens de menu (`command=lambda: self.show_frame("...")`) foram vinculados à função `show_frame`, que por sua vez exibe o `Frame` correto.
- **Atualização de Dados:** A função `show_frame` foi aprimorada para chamar as funções de `populate_list` e `populate_combobox` relevantes sempre que um `Frame` é exibido. Isso garante que os dados exibidos (listas de clientes, produtos, etc.) estejam sempre atualizados, refletindo quaisquer alterações feitas em outras seções do sistema.

Considerações Finais:

Esta reestruturação oferece uma interface mais organizada e intuitiva, alinhada com as expectativas de um sistema ERP. A separação da tela de login e a navegação baseada em menus cascata melhoram a experiência do usuário e a modularidade do código da GUI. As funcionalidades de Financeiro e Histórico de Vendas foram adicionadas como placeholders, indicando onde futuras implementações podem ser integradas.