UNIVERSIDADE PRESBITERIANA MACKENZIE
ENGENHARIA DA COMPUTAÇÃO

Projeto: Sistema de Gerenciamento de Chamados do ERP

Imaginem que sua empresa possui um ERP (*Enterprise Resource Planning*, ou Sistema Integrado de Gestão Empresarial), uma ferramenta vital para o funcionamento diário e estratégico de todos os departamentos. Como parte integrante e essencial da equipe de TI, vocês foram incumbidos de um projeto crítico: criar um sistema robusto e eficiente para gerenciar os chamados de manutenção, dúvidas, relatórios de erros, bugs, e outras solicitações referentes ao ERP.

Este sistema de gerenciamento de chamados será a espinha dorsal para assegurar que qualquer questão técnica ou dúvida relacionada ao ERP seja tratada de maneira ágil e eficaz, garantindo a continuidade dos processos empresariais sem maiores interrupções. Será responsável não só por catalogar e organizar os chamados de forma eficiente, mas também por prover aos atendentes e técnicos as ferramentas necessárias para uma gestão eficaz desses chamados, assegurando que todos os problemas sejam resolvidos dentro do Acordo de Nível de Serviço (SLA - *Service Level Agreement*) estabelecido.

Para atingir esse objetivo, o projeto será dividido em várias etapas, cada uma focando em aspectos diferentes do sistema, como a **interface de usuário**, **gerenciamento de usuários** e **atendentes**, **categorização de problemas e SLAs**, e a lógica por trás do **gerenciamento de status dos chamados**. Este projeto não só testará suas habilidades técnicas em programação, utilizando **Python** e a **biblioteca Tkinter** para desenvolver a interface gráfica, como também desafiará sua capacidade de pensar criticamente na resolução de problemas e no design de sistemas que sejam ao mesmo tempo robustos e flexíveis.

Adicionalmente, todos os dados manipulados pelo sistema deverão ser persistentes, armazenados em um banco de dados, garantindo a integridade e disponibilidade das informações em qualquer circunstância. Esta é uma excelente oportunidade para aplicar na prática os conhecimentos adquiridos em sala de aula e desenvolver habilidades que serão essenciais na sua carreira profissional, criando um sistema que terá um impacto real na operacionalidade e eficiência da empresa.

UNIVERSIDADE PRESBITERIANA MACKENZIE
ENGENHARIA DA COMPUTAÇÃO

DETALHAMENTO TÉCNICO DO SISTEMA

Estruturas e Atributos Necessários:

Usuários (Técnicos/Atendentes)
Atributos: ID, Nome, Email, Senha, Cargo
Métodos: Cadastrar, Visualizar, Alterar, Excluir

Clientes (Funcionários que utilizam o ERP)
Atributos: ID, Nome, Email, Empresa, Telefone
Métodos: Cadastrar, Visualizar, Alterar, Excluir

Categoria de Problemas
Atributos: ID, Descrição, SLA (em horas)
Métodos: Cadastrar, Visualizar, Alterar, Excluir

Chamados
Atributos: ID, Título, Descrição, Categoria, ID do Cliente, ID do Atendente, Status (Aberto, Em atendimento, Fechado), Data de Abertura, Data Máxima para Atendimento, Data de Fechamento
Métodos: Abrir, Atribuir Atendente, Alterar Status, Fechar, Visualizar, Alterar, Excluir

Interface Gráfica:
Será desenvolvida utilizando a biblioteca **Tkinter do Python**.
A interface deve incluir telas para o gerenciamento (cadastro, visualização, alteração e exclusão) de usuários, clientes, categorias de problemas e chamados.
Deve haver funcionalidade para listar chamados baseados em seu status, mostrando as datas relevantes (abertura, máxima para atendimento, fechamento).

Módulo de Login
Tela de Login: Uma interface gráfica construída com Tkinter que solicita ao usuário seu email e senha.
Autenticação: A lógica de backend que verifica as credenciais fornecidas contra as informações armazenadas no banco de dados SQLite.
Feedback de Autenticação: Mensagens claras ao usuário sobre o sucesso ou falha na tentativa de login.
Redirecionamento Após o Login: Após um login bem-sucedido, o usuário é direcionado à interface principal do sistema. Se for um atendente, deve ser direcionado para a interface de gerenciamento de chamados; se for um usuário regular, deve ser direcionado para uma interface onde possa visualizar ou abrir novos chamados.

Persistência de Dados:
Banco de Dados SQLite: Para a persistência dos dados do sistema, todos os dados manipulados (usuários, clientes, categorias de problemas, chamados, etc.) serão armazenados utilizando o SQLite. Obs.: Clientes e Usuário podem ser gravados na mesma tabela, crie uma coluna **tipo** para diferenciá-los, isso facilitará o login.

Estrutura do Banco de Dados: desenhar e implementar um esquema de banco de dados que suporte eficientemente todas as operações de CRUD (Criar, Ler, Atualizar, Deletar) para as entidades do sistema. Isso inclui tabelas para usuários, atendentes, categorias de problemas, chamados, entre outras necessárias, garantindo relações apropriadas entre elas.

Implementação de Acesso ao Banco de Dados: Deverão ser criadas classes ou módulos específicos para a interação com o banco de dados SQLite. Isso inclui a execução de comandos SQL para inserção, consulta, atualização e exclusão de dados, bem como para a realização de consultas específicas relacionadas ao gerenciamento dos chamados.

Migrações e Versionamento de Esquema: Recomenda-se (não obrigatório) o uso de uma ferramenta de migração de banco de dados, como o Alembic, para gerenciar as versões do esquema do banco de dados. Isso facilitará a atualização do esquema do banco de dados conforme o sistema evolui, sem perder dados. Veja um pequeno tutorial no final do arquivo.

UNIVERSIDADE PRESBITERIANA MACKENZIE
ENGENHARIA DA COMPUTAÇÃO

Requisitos de Entrega e Orientações Finais

Código Fonte Comentado: Todos os códigos fonte do projeto deverão ser bem documentados com comentários claros e concisos que expliquem a funcionalidade das seções importantes, facilitando assim a compreensão e futuras manutenções no sistema.

Utilização de Type Hints: Para garantir a clareza e a consistência do código, o uso de type hints será obrigatório em todas as funções e métodos, melhorando assim a legibilidade e a manutenção do código.

Documentação Detalhada: Incluir um arquivo **README** detalhado no repositório do GitHub, contendo instruções sobre como configurar, instalar e executar o sistema. Além disso, deve-se incluir uma descrição sobre a arquitetura do sistema, explicando como as diferentes partes do sistema interagem entre si e com o banco de dados.

Versionamento e GitHub: O projeto deve ser desenvolvido utilizando práticas de versionamento de código com o Git. O repositório do projeto deverá ser hospedado no GitHub, facilitando o acesso ao código, a colaboração entre os membros da equipe e a visualização do progresso do desenvolvimento. Espera-se que o histórico de commits reflita a evolução do projeto, com ***commits*** claros e descritivos das alterações realizadas.

Demonstração do Sistema: Uma demonstração funcional do sistema deverá ser preparada, evidenciando as funcionalidades implementadas, a interação com o banco de dados e a eficácia da interface gráfica desenvolvida com **Tkinter**. Esta demonstração poderá ser em forma de vídeo ou apresentação ao vivo, acompanhada de exemplos de uso reais dentro do contexto da gestão de chamados do ERP.

Importância do Projeto:

Este projeto não somente serve como uma aplicação prática dos conceitos aprendidos em sala de aula, mas também proporciona uma experiência valiosa no desenvolvimento de um sistema que atende a necessidades reais dentro de um contexto empresarial. Ao trabalhar neste projeto, vocês estarão não apenas aprimorando suas habilidades técnicas, mas também desenvolvendo competências essenciais como trabalho em equipe, resolução de problemas e comunicação eficaz, preparando-se para os desafios do mercado de trabalho na área de TI.

UNIVERSIDADE PRESBITERIANA MACKENZIE
ENGENHARIA DA COMPUTAÇÃO

Código Exemplo para Tela de Login (Tkinter):

```python
import tkinter as tk
from tkinter import messagebox

def attempt_login():
    username = entry_username.get()
    password = entry_password.get()
    # adicione aqui a lógica de verificação das credenciais no banco de dados
    # No exemplo o usuário "admin" com senha "1234" é válido
    if username == "admin" and password == "1234":
        messagebox.showinfo("Login", "Login bem-sucedido!")
        # Redirecionamento para a janela principal após o login
    else:
        messagebox.showerror("Login", "Nome de usuário ou senha inválidos.")

app = tk.Tk()
app.title("Login - Sistema de Gerenciamento de Chamados do ERP")

tk.Label(app, text="Nome de Usuário:").pack()
entry_username = tk.Entry(app)
entry_username.pack()

tk.Label(app, text="Senha:").pack()
entry_password = tk.Entry(app, show="*")
entry_password.pack()

tk.Button(app, text="Login", command=attempt_login).pack()

app.mainloop()
```

Acesse um bom e rápido tutorial do Tkinter.
https://www.tutorialspoint.com/python/python_gui_programming.htm

Instalação e utilização do Alembic

Para instalar o **Alembic**, você pode utilizar o gerenciador de pacotes do Python, o pip. O pip facilita a instalação de bibliotecas e ferramentas na sua ambiente Python. Aqui estão os passos para instalar o **Alembic**:

Abra o terminal:

O primeiro passo é abrir o terminal no seu sistema operacional.

Certifique-se de que o pip está atualizado:

```
pip install --upgrade pip setuptools wheel
```

Instalar o Alembic:

```
pip install alembic
```

```
alembic --version
```

Este comando deve retornar a versão do **Alembic** que foi instalada, confirmando que a instalação foi bem-sucedida.

Agora que o **Alembic** está instalado, você pode começar a usá-lo para gerenciar migrações de banco de dados em seus projetos Python. Para iniciar um projeto com o Alembic, você pode criar um diretório de migração com o comando *alembic init <nome_do_diretorio>*, onde *<nome_do_diretorio>* é o nome que você deseja dar ao diretório de migrações dentro do seu projeto.

Utilização do Alembic

1. Inicializando o Alembic no seu Projeto

Primeiro, navegue até a pasta do seu projeto no terminal. Em seguida, inicialize o Alembic:

```
alembic init alembic
```

Este comando cria uma nova pasta chamada **alembic** no seu projeto, contendo a configuração inicial e um diretório versions para armazenar as migrações. Dentro da pasta **alembic**, você encontrará um arquivo de configuração chamado **alembic.ini**, além de uma pasta env.py que define o ambiente de migração.

2. Configurando a Conexão com o Banco de Dados

Abra o arquivo alembic.ini e encontre a linha que começa com sqlalchemy.url =. Altere o valor dessa linha para a string de conexão do seu banco de dados.

```
sqlalchemy.url = sqlite:///./meubanco.db
```

3.

3. Criando uma Migração

Com o Alembic configurado, você pode agora criar sua primeira migração.

```
alembic revision -m "Criar tabela de usuários"
```

Isso cria um novo arquivo de migração no diretório **alembic/versions** com um nome de arquivo contendo o hash único da revisão e a mensagem de migração. Abra este arquivo e você verá duas funções: **upgrade()** e **downgrade()**. Aqui é onde você define como aplicar e como reverter a migração, respectivamente.

```
def upgrade():
    op.create_table(
        'usuarios',
        sa.Column('id', sa.Integer, primary_key=True),
        sa.Column('nome', sa.String(50), nullable=False),
        sa.Column('email', sa.String(50), unique=True, nullable=False)
    )

def downgrade():
    op.drop_table('usuarios')
```

4. Aplicando a Migração

Para aplicar a migração ao seu banco de dados, execute:

```
alembic upgrade head
```

Esse comando aplica todas as migrações pendentes (até a "cabeça", que é a última migração na linha de tempo de migração).

5. Revertendo uma Migração

Se você precisar reverter a última migração aplicada, pode usar:

```
alembic downgrade -1
```

Isso move uma revisão para trás, revertendo a última migração aplicada.

Este é um exemplo básico de como começar a usar o **Alembic** para gerenciar migrações de banco de dados. O **Alembic** é uma ferramenta poderosa e flexível que pode ser adaptada para muitos fluxos de trabalho de desenvolvimento diferentes.