

PROJETO DE DESENVOLVIMENTO RÁPIDO DE APLICAÇÕES EM PYTHON

Aluno: Erick de Faria Pereira

SISTEMA DE CADASTRO DO SUPERMERCADO BATATINHA

Requisitos:

Propósito	Sistema com propósito do cadastro de produtos do supermercado batatinha.
Escopo	Sistema feito com uma tela home, uma tela de cadastro na qual ira ser feito o cadastro do profissional responsavel, tela de login, e a tela do programa onde é realizado o cadastramento de produtos, atualização ou exclusão do produto.
Interfaces do Sistema	Sistema compativel com Linux e com Windows.
Interfaces de Usuário	Textos, caixas para escrita, e botões.
Interfaces de Software	No software é possivel fazer cadastro, atualizar, excluir, limpar as caixas de texto, calcular o juros. Tudo feito com conexão ao banco de dados PostgreeSQL
Requisitos Funcionais	<p>É capaz de:</p> <ul style="list-style-type: none">• Realizar o cadastramento do produto.• Realizar o cadastramento do funcionario.• Atualizar o cadastro do produto;• Excluir os dados;• Limpar os Campos;• Mostrar Registros dos produtos cadastrados;

Caso de uso:

O funcionário cadastrado no sistema é responsável por cadastrar, atualizar, excluir, limpar e mostrar o registro. Ao chegar no cadastramento dos produtos, é pedido Código do Produto, Nome, Preço, o juro (que pode ser os 10% ou qualquer outro juro que o funcionário tenha recebido ordem para adicionar), e o Preço com o juro calculado.

Criação do código:

CRUD:

A classe AppBD é responsável por gerenciar a interação do programa com o banco de dados PostgreSQL. Ela possui métodos para abrir a conexão, inserir dados, atualizar dados e excluir dados na tabela "produto". Abaixo, eu detalho cada método:

Método 'abrirConexão'

```
7     def abrirConexao(self):
8         try:
9             self.connection = psycopg2.connect(user="postgres", password="trabalho",
10             host="localhost", port="5432",
11             database= "Python")
12         except (Exception, psycopg2.Error) as error:
13             if(self.connection):
14                 print("Falha ao se conectar ao Banco de Dados", error)
15
```

Responsável por abrir a conexão com o banco de dados PostgreSQL, caso a conexão falhe a mensagem "Falha ao se conectar ao Banco de Dados" é exibida.

Método 'Inserir Dados'

```

16 def inserirDados(self, codigo, nome, preco):
17     try:
18         self.abrirConexao()
19         cursor = self.connection.cursor()
20         postgres_insert_query = """ INSERT INTO public."produto"
21         ("codigo", "nome", "preco") VALUES (%s,%s,%s)"""
22         record_to_insert = (codigo, nome, preco)
23         cursor.execute(postgres_insert_query, record_to_insert)
24         self.connection.commit()
25         count = cursor.rowcount
26         print(count, "Registro inserido com sucesso na tabela PRODUTO")
27     except (Exception, psycopg2.Error) as error:
28         if(self.connection):
29             print("Falha ao inserir registro na tabela PRODUTO", error)
30     finally:
31         if(self.connection):
32             cursor.close()
33             self.connection.close()
34             print("A conexão com o PostgreSQL foi fechada")

```

Inserir um novo registro na tabela "produto". Ele abre a conexão, cria um cursor, executa a inserção e, em seguida, confirma a transação e fecha a conexão. O código, nome e preço do produto são passados como parâmetros.

Metodo 'atualizarDados'

```

def atualizarDados(self, codigo, nome, preco, juros):
    try:
        self.abrirConexao()
        cursor = self.connection.cursor()
        sql_update_query = """Update public."produto" set "nome" = %s, "preco" = %s, "juros" = %s where "codigo" = %s"""
        cursor.execute(sql_update_query, (nome, preco, juros, codigo))
        self.connection.commit()
        count = cursor.rowcount
        print(count, "Registro atualizado com sucesso!")
        print("Registro Depois da Atualização ")
        sql_select_query = """select * from public."produto" where "codigo" = %s"""
        cursor.execute(sql_select_query, (codigo,))
        record = cursor.fetchone()
        print(record)
    except (Exception, psycopg2.Error) as error:
        print("Erro na Atualização", error)
    finally:
        if (self.connection):
            cursor.close()
            self.connection.close()
            print("A conexão com o PostgreSQL foi fechada")

```

Atualiza os dados de um produto na tabela "produto". Ele segue uma abordagem semelhante ao método inserirDados, mas, antes de realizar a atualização, exibe o registro antes e depois da atualização.

Método 'excluirDados'

```

58     def excluirDados(self, codigo):
59         try:
60             self.abrirConexao()
61             cursor = self.connection.cursor()
62             sql_delete_query = """Delete from public."produto"
63             where "codigo" = %s"""
64             cursor.execute(sql_delete_query, (codigo, ))
65             self.connection.commit()
66             count = cursor.rowcount
67             print(count, "Registro excluído com sucesso!")
68         except (Exception, psycopg2.Error) as error:
69             print("Erro na Exclusão", error)
70         finally:
71             if (self.connection):
72                 cursor.close()
73                 self.connection.close()
74                 print("A conexão com o PostgreSQL foi fechada")
75

```

Exclui um registro da tabela "produto" com base no código fornecido como parâmetro. Ele abre a conexão, cria um cursor, executa a exclusão, confirma a transação e fecha a conexão.

Main:

```

1  import tkinter as tk
2  from home import TelaHome
3
4  if __name__ == "__main__":
5      janela = tk.Tk()
6      TelaInicial = TelaHome(janela)
7      janela.title('Tela Inicial')
8      janela.geometry("800x600")
9      janela.mainloop()
10

```

O arquivo Main é utilizado para iniciar o programa, o programa deve ser executado a partir dele, o qual leva para a tela home.

Home:

Metodo `__init__`:

```

def __init__(self, win):
    self.win = win

    self.lblBemVindo = tk.Label(win, text='Seja bem-vindo', font=('Arial', 24))
    self.lblBemVindo.place(relx=0.5, rely=0.2, anchor=tk.CENTER)

    self.lblBemVindo2 = tk.Label(win, text='ao sistema de cadastramento do SuperMercado', font=('Arial', 18))
    self.lblBemVindo2.place(relx=0.5, rely=0.3, anchor=tk.CENTER)

    self.lblBemVindo3 = tk.Label(win, text='BATATINHA', font=('Arial', 24))
    self.lblBemVindo3.place(relx=0.5, rely=0.4, anchor=tk.CENTER)

    self.btnIniciar = tk.Button(win, text="Iniciar", command=self.abrirCadastro)
    self.btnIniciar.place(relx=0.5, rely=0.6, anchor=tk.CENTER)

```

Construtor da classe TelaHome. É chamado automaticamente quando um objeto dessa classe é criado. No construtor, são inicializadas variáveis de instância e criados rótulos e um botão para a interface gráfica.

Método abrirCadastro:

```

21     def abrirCadastro(self):
22         self.janela_cadastro = tk.Toplevel(self.win)
23         CadastroFuncionario(self.janela_cadastro, self)
24         self.janela_cadastro.protocol("WM_DELETE_WINDOW", self.fecharCadastro)
25         self.janela_cadastro.title('Tela de Cadastro')
26         self.janela_cadastro.geometry("800x600")
27         self.win.iconify()

```

Esse método é chamado quando o botão "Iniciar" é clicado. Ele cria uma nova janela para o cadastro, instancia a classe CadastroFuncionario, configura o comportamento ao fechar a janela de cadastro, define o título e as dimensões da janela de cadastro e minimiza a janela principal enquanto a janela de cadastro está aberta.

Cadastro

Método Inserir dados

```

def inserirDados(self, nome, cpf, senha):
    try:
        self.abrirConexao()
        cursor = self.connection.cursor()
        postgres_insert_query = """ INSERT INTO public."funcionario"
        ("nome", "cpf", "senha") VALUES (%s,%s,%s)"""
        record_to_insert = (nome, cpf, senha)
        cursor.execute(postgres_insert_query, record_to_insert)
        self.connection.commit()
        count = cursor.rowcount
        print(count, "Registro inserido com sucesso na tabela FUNCIONARIO")
    except (Exception, psycopg2.Error) as error:
        if(self.connection):
            print("Falha ao inserir registro na tabela FUNCIONARIO", error)
    finally:
        if(self.connection):
            cursor.close()
            self.connection.close()
            print("A conexão com o PostgreSQL foi fechada")

```

Este método insere um novo registro na tabela "funcionario", Inicia abrindo a conexão e cria um cursor para executar comandos SQL, Utiliza uma consulta SQL parametrizada para evitar injeção de SQL. Executa a consulta com os dados fornecidos, confirma a transação e imprime a quantidade de registros afetados. Em caso de erro, imprime a mensagem de falha e, no bloco finally, garante que a conexão seja fechada, evitando vazamentos de recursos.

Classe CadastroFuncionario

```

class CadastroFuncionario:
    def __init__(self, win, tela_login):
        self.win = win
        self.tela_login = tela_login
        self.objBD = AppBD()

        win.geometry("800x600")
        win.title('Cadastro de funcionário')

        self.lblcadastro = tk.Label(win, text='Cadastro de funcionário', font=('Arial', 28))
        self.lblcadastro.pack(pady=(30, 10))

        self.lblNome = tk.Label(win, text='Nome:')
        self.lblCPF = tk.Label(win, text='CPF:')
        self.lblSenha = tk.Label(win, text='Senha:')

        self.txtNome = tk.Entry(win)
        self.txtCPF = tk.Entry(win)
        self.txtSenha = tk.Entry(win, show="*")

        self.btnCadastrar = tk.Button(win, text='Cadastrar', command=self.fCadastrarFuncionario)

        self.lblNome.pack()
        self.txtNome.pack()
        self.lblCPF.pack()
        self.txtCPF.pack()
        self.lblSenha.pack()
        self.txtSenha.pack()
        self.btnCadastrar.pack(pady=10)

        self.lblJaTemConta = tk.Label(win, text="Já tem uma conta? Logue-se")
        self.lblJaTemConta.pack()

        self.btnLogin = tk.Button(win, text="Login", command=self.abrirTelaLogin)
        self.btnLogin.pack()

        self.msgLabel = tk.Label(win, text='', fg='red')
        self.msgLabel.pack(pady=(10, 20))

```

Configura a interface gráfica da janela de cadastro, cria uma instância de AppBD para interagir com o banco de dados.

Método fCadastrarFuncionario:

```

def fCadastrarFuncionario(self):
    try:
        nome = self.txtNome.get()
        cpf = self.txtCPF.get()
        senha = self.txtSenha.get()

        if not nome or any(char.isdigit() or not char.isalpha() and char not in (" ", "'") for char in nome):
            raise ValueError("Nome inválido. Por favor, insira um nome válido.")

        if not cpf or not cpf.isdigit() or len(cpf) != 11:
            raise ValueError("CPF inválido. Por favor, insira um CPF válido com 11 dígitos numéricos.")

        if not senha or len(senha) < 3:
            raise ValueError("Senha inválida. A senha deve ter no mínimo 3 caracteres.")

        self.objBD.inserirDados(nome, cpf, senha)
        print('Funcionário Cadastrado com Sucesso!')
        self.abrirTelaLogin()

    except ValueError as ve:
        self.msgLabel.config(text=str(ve))
    except Exception as e:
        print('Não foi possível fazer o cadastro. Erro:', e)

```

Este método é chamado ao clicar no botão "Cadastrar", Obtém os dados inseridos nos campos de texto, Realiza validações nos dados (nome, CPF, senha).
Chama o método inserirDados da classe AppBD para inserir os dados no banco, Em caso de erro, exibe uma mensagem de erro na interface gráfica.

Método abrirTelaLogin:

```

100     def abrirTelaLogin(self):
101         from login import TelaLogin
102         janela_login = tk.Tk()
103         TelaLogin(janela_login)
104         self.win.withdraw()
105         janela_login.title('Tela de Login')
106         janela_login.geometry("800x600")
107
108         janela_login.eval('tk::PlaceWindow . center')
109
110         janela_login.mainloop()

```

Este método abre a tela de login quando chamado, Cria uma instância da classe TelaLogin, Esconde a janela atual de cadastro (self.win) e exibe a janela de login. Configura o título e as dimensões da janela de login.

Login

Classe TelaLogin


```

class TelaLogin:
    def __init__(self, win):
        self.win = win
        self.win.title("Tela de Login")

        self.lbllogin = tk.Label(win, text='Login', font=('Arial', 26))
        self.lbllogin.place(relx=0.5, rely=0.15, anchor=tk.CENTER)

        win.eval('tk::PlaceWindow . center')

        self.lblNome = tk.Label(win, text='Nome:')
        self.lblSenha = tk.Label(win, text='Senha:')

        self.txtNome = tk.Entry(win)
        self.txtSenha = tk.Entry(win, show="*")

        self.btnLogin = tk.Button(win, text='Login', command=self.realizarLogin)
        self.btnCadastrar = tk.Button(win, text='Cadastre-se', command=self.abrirCadastro)

        self.lblNome.place(relx=0.5, rely=0.3, anchor=tk.CENTER)
        self.txtNome.place(relx=0.5, rely=0.35, anchor=tk.CENTER)
        self.lblSenha.place(relx=0.5, rely=0.40, anchor=tk.CENTER)
        self.txtSenha.place(relx=0.5, rely=0.45, anchor=tk.CENTER)
        self.btnLogin.place(relx=0.5, rely=0.55, anchor=tk.CENTER)
        self.btnCadastrar.place(relx=0.5, rely=0.65, anchor=tk.CENTER)

```

Configura a interface gráfica da tela de login, incluindo rótulos, campos de entrada e botões.

Método realizarLogin

```

def realizarLogin(self):
    nome = self.txtNome.get()
    senha = self.txtSenha.get()

    if not nome or not senha:
        mensagem_erro = tk.Label(self.win, text='Campos vazios. Por favor, insira os devidos dados nos campos.', fg='red')
        mensagem_erro.place(relx=0.5, rely=0.9, anchor=tk.CENTER)
    else:
        try:
            connection = psycopg2.connect(user="postgres",
                                           password="trabalho",
                                           host="localhost",
                                           port="5432",
                                           database="Python")

            cursor = connection.cursor()
            cursor.execute(f"SELECT * FROM public.\"funcionario\" WHERE nome = '{nome}' AND senha = '{senha}'")
            record = cursor.fetchone()

            if record is not None:
                self.abrirPrograma()
            else:
                print("Credenciais inválidas. Tente novamente.")

        except (Exception, psycopg2.Error) as error:
            print("Erro ao conectar ao banco de dados:", error)

        finally:
            if connection:
                cursor.close()
                connection.close()

```

Este método é chamado ao clicar no botão "Login", Obtém os dados inseridos nos campos de texto, Realiza uma verificação se algum campo está vazio e exibe uma

mensagem de erro se necessário, Tenta conectar ao banco de dados e executar uma consulta SQL para verificar as credenciais do usuário. Se as credenciais são válidas, chama o método abrirPrograma.

Método abrirPrograma:

```
def abrirPrograma(self):
    self.win.withdraw()
    janelaPrograma = tk.Tk()
    PrincipalBD(janelaPrograma)
    janelaPrograma.title('Tela do Programa')
    janelaPrograma.geometry("1000x900")
    janelaPrograma.deiconify()
```

Este método é chamado após a autenticação bem-sucedida. Esconde a janela de login (self.win) e cria uma nova janela para o programa principal. Configura o título e as dimensões da nova janela.

Método AbrirCadastro

```
def abrirCadastro(self):
    self.janela_cadastro = tk.Tk()
    CadastroFuncionario(self.janela_cadastro, self)
    self.janela_cadastro.protocol("WM_DELETE_WINDOW", self.fecharCadastro)
    self.janela_cadastro.title('Tela de Cadastro')
    self.janela_cadastro.geometry("800x600")
    self.win.iconify()
```

Este método é chamado ao clicar no botão "Cadastre-se". Cria uma nova instância da janela de cadastro (CadastroFuncionario). Define a ação a ser executada quando a janela de cadastro é fechada. Configura o título e as dimensões da janela de cadastro.

Programa

Método `__init__`

(Apenas parte do código)

```
class PrincipalBD:
    def __init__(self, win):
        self.objBD = AppBD()

        win.title('Bem Vindo à Tela de Cadastro')
        win.geometry("1000x900")
        win.eval('tk::PlaceWindow . center')
        win.maxsize(width=1200, height=800)
        win.minsize(width=300, height=200)
        win.columnconfigure(0, weight=1)
        win.rowconfigure(0, weight=1)
```

Configura a interface gráfica da tela principal de cadastro.

Metodo fLerCampos

```
def fLerCampos(self):
    try:
        codigo = self.txtCodigo.get()
        if not codigo.isdigit():
            self.msgLabel.config(text='Código deve conter apenas números', fg='red')
            return None
        codigo = int(codigo)
        nome = self.txtNome.get()
        if not nome.isalnum():
            self.msgLabel.config(text='Nome deve conter apenas letras e números', fg='red')
            return None
        preco = self.txtPreco.get()
        try:
            preco = float(preco)
        except ValueError:
            self.msgLabel.config(text='Preço deve conter apenas números', fg='red')
            return None
        calculo_juros = self.txtCalculoJuros.get()
        try:
            calculo_juros = float(calculo_juros)
        except ValueError:
            self.msgLabel.config(text='Cálculo de Juros deve conter apenas números', fg='red')
            return None
        print('Leitura dos Dados com Sucesso!')
        self.msgLabel.config(text='', fg='black')
    except ValueError:
        self.msgLabel.config(text='Campos Código e Preço devem conter apenas números', fg='red')
        return None
    return codigo, nome, preco, calculo_juros
```

Este método lê os valores dos campos da interface gráfica e realiza validações. Retorna os valores lidos ou None se ocorrer um erro durante a leitura.

Metodo fCadastrarProduto

```
def fCadastrarProduto(self):
    try:
        codigo, nome, preco, juros = self.fLerCampos()
        self.objBD.inserirDados(codigo, nome, preco, juros)
        self.fLimparTela()
        print('Produto Cadastrado com Sucesso!')
    except Exception as e:
        print('Não foi possível fazer o cadastro. Erro:', e)
```

Este método chama o método correspondente na instância AppBD para cadastrar um produto com base nos campos preenchidos na interface gráfica. Limpa a tela em caso de sucesso.

Metodo fLimparTela

```
def fLimparTela(self):
    try:
        self.txtCodigo.delete(0, tk.END)
        self.txtNome.delete(0, tk.END)
        self.txtPreco.delete(0, tk.END)
        self.txtCalculoJuros.delete(0, tk.END)
        self.txtPrecoComJuros.delete(0, tk.END)
        print('Campos Limpos!')
    except:
        print('Não foi possível limpar os campos.')
```

Limpa os campos da interface ao o usuario apertar o botão.

Metodo fAtualizarProduto

```

def fAtualizarProduto(self):
    try:
        codigo, nome, preco, juros = self.fLerCampos()

        preco_com_juros = preco + (preco * (juros / 100))

        self.txtPrecoComJuros.delete(0, tk.END)
        self.txtPrecoComJuros.insert(0, str(preco_com_juros))

        self.objBD.atualizarDados(codigo, nome, preco, juros)
        self.fLimparTela()
        print('Produto Atualizado com Sucesso!')
    except Exception as e:
        print('Não foi possível fazer a atualização. Erro:', e)

```

Chama o método correspondente na instância AppBD para atualizar um produto com base nos campos preenchidos na interface gráfica. Calcula o preço com juros e imprime mensagens indicando o resultado da operação. Limpa a tela em caso de sucesso.

Metodo fExcluirProduto

```

def fExcluirProduto(self):
    try:
        codigo, nome, preco, _ = self.fLerCampos()
        self.objBD.excluirDados(codigo)
        self.fLimparTela()
        print('Produto Excluído com Sucesso!')
    except Exception as e:
        print('Não foi possível fazer a exclusão do produto. Erro:', e)

```

Exclui um produto com base no código fornecido na interface gráfica. Limpa a tela em caso de sucesso.

Metodo fCalcularJuros

```

def fCalcularJuros(self):
    try:
        result = self.fLerCampos()
        if result is not None:
            _, _, preco, calculo_juros = result

            preco_com_juros = preco + (preco * (calculo_juros / 100))

            self.txtPrecoComJuros.delete(0, tk.END)
            self.txtPrecoComJuros.insert(0, str(preco_com_juros))
    except Exception as e:
        print('Não foi possível calcular o juro. Erro:', e)

```

Calcula o preço com juros com base nos campos preenchidos na interface gráfica e exibe o resultado no campo correspondente.

Metodo fListarProdutos

```

def fListarProdutos(self):
    try:
        self.objBD.abrirConexao()
        cursor = self.objBD.connection.cursor()

        sql_select_query = """SELECT * FROM public."produto" """
        cursor.execute(sql_select_query)
        records = cursor.fetchall()

        self.txtProdutos.config(state=tk.NORMAL)
        self.txtProdutos.delete('1.0', tk.END)

        for record in records:
            codigo, nome, preco, juros, preco_com_juros = record
            texto = f'Código: {codigo}, Nome: {nome}, Preço: {preco}, Juros: {juros},  
Preço com Juros: {preco_com_juros}\n'
            self.txtProdutos.insert(tk.END, texto)

        self.txtProdutos.config(state=tk.DISABLED)

    except Exception as e:
        print('Erro ao listar produtos:', e)
    finally:
        if self.objBD.connection:
            cursor.close()
            self.objBD.connection.close()
            print("A conexão com o PostgreSQL foi fechada")

```

lista todos os produtos na interface gráfica, obtendo os dados do banco de dados e exibindo-os no campo de texto.

Banco de Dados

Tabela produto

```

1 CREATE TABLE produto (
2     codigo SERIAL PRIMARY KEY,
3     nome VARCHAR(255) NOT NULL,
4     preco NUMERIC(10, 2) NOT NULL,
5     juros NUMERIC(5, 2) NOT NULL,
6     preco_com_juros NUMERIC(10, 2) NOT NULL
7 );
8

```

1 `select * from produto`

Data Output Messages Notifications

	codigo [PK] integer	nome text	preco numeric	juros numeric	preco_com_juros numeric
1	1234	alcatra	10.0	10.0	11.0
2	4531	bacalhau	40.0	10.0	44.0
3	5342	alface	8.0	10.0	8.8
4	5428	maizena	3.0	10.0	3.3
5	9383	arroz	15.0	10.0	16.5
6	1111	feijão	8.0	10.0	8.8
7	2222	hamburguer	40.0	10.0	44.0
8	3333	salsicha	9.0	10.0	9.9
9	4444	papicra	10.0	10.0	11.0
10	5555	agrião	16.0	10.0	17.6
11	6666	ketchup	15.0	10.0	16.5
12	7777	frango	12.0	10.0	13.2
13	8888	picanha	60.0	10.0	66.0
14	9999	cupim	80.0	10.0	88.0
15	987	salgadinho	10.0	10.0	11.0
16	0	biscoito	21.0	10.0	23.1
17	897	paobrioche	40.0	10.0	44.0
18	6789	maça	7.0	10.0	7.7
19	2393	pera	2.0	10.0	2.2
20	3456	morango	10.0	10.0	11.0

Tabela Funcionario

```

CREATE TABLE public."funcionario"
(
    id serial PRIMARY KEY,
    nome character varying(255) NOT NULL,
    cpf character varying(14) NOT NULL,
    senha character varying(255) NOT NULL
);

```

```
1 SELECT * FROM funcionario;
```

Data Output Messages Notifications

	id [PK] integer	nome character varying (255)	cpf character varying (14)	senha character varying (255)
1	7	erick	1111111111	1234
2	8	simone	2222222222	123456