**UNIVERSIDADE ESTÁCIO DE SÁ**

**TRABALHO DE RAD EM PYTHON**

Aluno: Matheus Silva Fonseca Rodrigues

Matrícula: 202203433199

**Projeto de Cadastro de Produtos em Python usando Metodologia RAD**

**Introdução**

Este documento detalha o desenvolvimento de um sistema de cadastro de produtos em Python utilizando a Metodologia de Desenvolvimento Rápido de Aplicações (RAD). O projeto visa criar uma interface gráfica para facilitar operações básicas, como cadastro, atualização, exclusão e seleção de produtos em um banco de dados PostgreSQL.

**Metodologia RAD (Desenvolvimento Rápido de Aplicações)**

A Metodologia RAD é uma abordagem ágil que se destaca por iterações curtas, colaboração intensiva, prototipagem rápida e adaptação frequente aos requisitos do cliente. Esses princípios foram incorporados ao projeto para proporcionar um desenvolvimento mais ágil e interativo.

**Principais Características do RAD no Projeto**

**1. Iterações Curtas**: O desenvolvimento foi dividido em iterações curtas, cada uma concentrando-se em funcionalidades específicas. Isso permitiu entregas mais rápidas e frequentes.

**2. Colaboração Intensiva:** A colaboração entre desenvolvedores, designers e usuários finais foi incentivada. A comunicação constante garantiu que o produto atendesse às expectativas e necessidades do usuário.

**3. Prototipagem Rápida:** Protótipos foram criados rapidamente para validar a interface do usuário e obter feedback contínuo. Isso ajudou a refinar as funcionalidades de acordo com as preferências do usuário.

**4. Mudanças Frequentes:** A capacidade de adaptação a mudanças nos requisitos foi fundamental. O projeto foi flexível o suficiente para incorporar ajustes conforme necessário.

**Componentes Principais**

**`connection.py`**

- **AppBD (Aplicativo de Banco de Dados):**

**- `\_\_init\_\_(self):` Método construtor que inicia a conexão com o banco de dados.**

*import* psycopg2

class AppBD:

    def \_\_init\_\_(self):

        print("Método construtor")

**- `abrirConexao(self):` Abre a conexão com o PostgreSQL.** Gráfico

Descrição gerada automaticamente

- `**inserirDados(self, codigo, nome, preco\_acrescido):` Insere um novo produto no banco de dados.**

 def *inserirDados*(self, codigo, nome, preco\_acrescido):

*try*:

            self.abrirConexao()

            cursor = self.connection.cursor()

            postgres\_insert\_query = """ INSERT INTO public."produto"

                ("codigo", "nome", "preco") VALUES (%s,%s,%s)"""

            record\_to\_insert = (codigo, nome, preco\_acrescido)

            cursor.execute(postgres\_insert\_query, record\_to\_insert)

            self.connection.commit()

            count = cursor.rowcount

            print(count, "Registro inserido com sucesso na tabela PRODUTO")

*except* (Exception, psycopg2.Error) *as* error:

*if* self.connection:

                print("Falha ao inserir registro na tabela PRODUTO", error)

*finally*:

*if* self.connection:

                cursor.close()

                self.connection.close()

                print("A conexão com o PostgreSQL foi fechada.")

- **`excluirDados(self, codigo):` Exclui um produto com base no código**.

 def *excluirDados*(self, codigo):

*try*:

            self.abrirConexao()

            cursor = self.connection.cursor()

            sql\_delete\_query = """DELETE FROM public."produto" WHERE "codigo" = %s"""

            cursor.execute(sql\_delete\_query, (codigo,))

            self.connection.commit()

            count = cursor.rowcount

            print(count, "Registro excluído com sucesso na tabela PRODUTO")

*except* (Exception, psycopg2.Error) *as* error:

            print("Falha ao excluir registro na tabela PRODUTO", error)

*finally*:

*if* self.connection:

                cursor.close()

                self.connection.close()

                print("A conexão com o PostgreSQL foi fechada.")

**- `atualizarDados(self, codigo, nome, preco):` Atualiza informações de um produto.**

def *atualizarDados*(self, codigo, nome, preco):

*try*:

            self.abrirConexao()

            cursor = self.connection.cursor()

            preco\_acrescido = float(preco) \* 1.10

            sql\_update\_query = sql\_update\_query = """UPDATE public."produto" SET "nome" = %s,

                    "preco" = %s WHERE "codigo" = %s"""

            cursor.execute(sql\_update\_query, (nome, preco\_acrescido, codigo))

            self.connection.commit()

            count = cursor.rowcount

            print (count, "Registro atualizado com sucesso na tabela PRODUTO")

*except*(Exception, psycopg2.Error) *as* error:

            print("Falha ao atualizar registro na tabela PRODUTO", error)

*finally*:

*if*(self.connection):

                cursor.close()

                self.connection.close()

                print("A conexão com o PostgreSQL foi fechada.")

**- `consultarProduto(self, codigo):` Seleciona um produto com base no código**.

def *consultarProduto*(self, codigo):

*try*:

            self.abrirConexao()

            cursor = self.connection.cursor()

            sql\_select\_query = """SELECT "nome", "preco" FROM public."produto" WHERE "codigo" = %s"""

            cursor.execute(sql\_select\_query, (codigo,))

            result = cursor.fetchone()

*if* result:

                nome, preco = result

*return* {"nome": nome, "preco": preco}

*else*:

*return* None

*except* (Exception, psycopg2.Error) *as* error:

            print("Falha ao consultar produto na tabela PRODUTO", error)

*finally*:

*if* self.connection:

                cursor.close()

                self.connection.close()

                print("A conexão com o PostgreSQL foi fechada.")

**`tela.py**`

**- PrincipalBD (Controlador):**

**- `\_\_init\_\_(self, win):` Método construtor que inicializa a interface gráfica.**

*import* tkinter *as* tk

*import* connection *as* cn

*import* psycopg2

*from* ttkbootstrap *import* Style

class PrincipalBD:

    def \_\_init\_\_(self, win):

        self.objBD = cn.AppBD()

        style = Style("morph")

        style.theme\_use("morph")

        style.configure("TLabel", font="Arial 12", padding=10)

        style.configure("TButton", font="Arial 12", padding=10)

        self.lbCodigo = tk.Label(win, text = 'Código do Produto:')

        self.lblNome = tk.Label(win, text = 'Nome do Produto')

        self.lblPreco = tk.Label(win, text = 'Preço')

        self.txtCodigo = tk.Entry(bd = 3)

        self.txtNome = tk.Entry()

        self.txtPreco = tk.Entry()

        self.btnCadastrar = tk.Button(win, text='Cadastrar', command=lambda: self.fCadastrarProduto(win))

        self.btnAtualizar = tk.Button(win, text='Atualizar', command=self.abrirTelaAtualizacao)

        self.btnExcluir = tk.Button(win, text = 'Excluir', command = self.fExcluirProduto)

        self.btnLimpar = tk.Button(win, text = 'Limpar', command = self.fLimparTela)

        self.btnSelecionar = tk.Button(win, text = 'Selecionar', command = self.fabrirTelaSelect)

        self.lbCodigo.place(x=100, y=50)

        self.txtCodigo.place(x=250, y=50)

        self.lblNome.place(x=100, y=100)

        self.txtNome.place(x=250, y=100)

        self.lblPreco.place(x=100, y=150)

        self.txtPreco.place(x=250, y=150)

        self.btnCadastrar.place(x=100, y=200)

        self.btnAtualizar.place(x=200, y=200)

        self.btnExcluir.place(x=300, y=200)

        self.btnLimpar.place(x=400, y=200)

        self.btnSelecionar.place(x=500, y=200)

- **fabrirTelaSelect(self): Abre uma nova tela para fazer a query de SELECT nos produtos.**

def *fabrirTelaSelect*(self):

        janela\_select = tk.Toplevel()

        janela\_select.title('Consulta de Produtos')

        lblCodigoSelect = tk.Label(janela\_select, text='Código do Produto:')

        txtCodigoSelect = tk.Entry(janela\_select, bd=3)

        btnSelect = tk.Button(janela\_select, text='Pesquisar', command=lambda: self.fSelecionarProduto(janela\_select, txtCodigoSelect.get()))

        lblCodigoSelect.grid(row=0, column=0)

        txtCodigoSelect.grid(row=0, column=1)

        btnSelect.grid(row=1, column=0)

-**abrirTelaAtualizaçao(self): Abre uma nova tela para atualizar os produtos no banco de dados.**

def *abrirTelaAtualizacao*(self):

        janela\_atualizacao = tk.Toplevel()

        janela\_atualizacao.title('Atualização de Produtos')

        lblCodigoAtualizacao = tk.Label(janela\_atualizacao, text='Código do Produto:')

        txtCodigoAtualizacao = tk.Entry(janela\_atualizacao, bd=3)

        lblNomeAtualizacao = tk.Label(janela\_atualizacao, text='Novo Nome do Produto:')

        txtNomeAtualizacao = tk.Entry(janela\_atualizacao)

        lblPrecoAtualizacao = tk.Label(janela\_atualizacao, text='Novo Preço:')

        txtPrecoAtualizacao = tk.Entry(janela\_atualizacao)

        btnAtualizar = tk.Button(janela\_atualizacao, text='Atualizar', command=lambda: self.atualizarDados(janela\_atualizacao, txtCodigoAtualizacao.get(), txtNomeAtualizacao.get(), txtPrecoAtualizacao.get()))

        lblCodigoAtualizacao.grid(row=0, column=0)

        txtCodigoAtualizacao.grid(row=0, column=1)

        lblNomeAtualizacao.grid(row=1, column=0)

        txtNomeAtualizacao.grid(row=1, column=1)

        lblPrecoAtualizacao.grid(row=2, column=0)

        txtPrecoAtualizacao.grid(row=2, column=1)

        btnAtualizar.grid(row=3, column=1)

**- `fCadastrarProduto(self, win):` Realiza o cadastro de um novo produto.**

def *fCadastrarProduto*(self, win):

*try*:

            codigo, nome, preco = self.fLerCampos()

            preco\_acrescido = preco \* 1.10

            self.lblPrecoAcrescido = tk.Label(win, text=f"Preço acrescido: {preco\_acrescido}")

            self.lblPrecoAcrescido.place(x=100, y=250)

            self.objBD.inserirDados(codigo, nome, preco\_acrescido)

            self.fLimparTela()

            print('Produto Cadastrado com Sucesso!')

*except* Exception *as* e:

            print('Não foi possível fazer o cadastro:', e)

**- `fExcluirProduto(self):` Exclui um produto**.

def *fExcluirProduto*(self):

*try*:

            codigo, \_, \_ = self.fLerCampos()

            self.abrirTelaAtualizacao(codigo)

            print('Produto Excluído com Sucesso!')

*except* Exception *as* e:

            print(f'Não foi possível fazer a exclusão: {e}')

**- `fAtualizarProduto(self):` Atualiza informações de um produto.**

def *fAtualizarProduto*(self):

*try*:

            codigo, nome, preco = self.fLerCampos()

            preco\_acrescido = preco \* 1.10

            self.objBD.atualizarDados(codigo, nome, preco)

            self.fLimparTela()

            print('Produto Atualizado com Sucesso!')

*except*:

            print('Não foi possível fazer a atualização.')

**- `fSelecionarProduto(self, janela\_select, codigo):` Exibe as informações de um produto selecionado.**

def *fSelecionarProduto*(self, janela\_select, codigo):

*try*:

            codigo = int(codigo)

            resultado = self.objBD.consultarProduto(codigo)

*if* resultado:

                lblResultado = tk.Label(janela\_select, text=f"Nome: {resultado['nome']}, Preço: {resultado['preco']}")

                lblResultado.grid(row=2, column=0, columnspan=2)

*else*:

                lblResultado = tk.Label(janela\_select, text="Produto não encontrado.")

                lblResultado.grid(row=2, column=0, columnspan=2)

*except* ValueError:

            lblResultado = tk.Label(janela\_select, text="Código inválido.")

            lblResultado.grid(row=2, column=0, columnspan=2)

**- `fLerCampos(self):` Lê os campos da interface gráfica.**

def *fLerCampos*(self):

        codigo = int(self.txtCodigo.get())

        nome = self.txtNome.get()

        preco = float(self.txtPreco.get())

*return* codigo, nome, preco

**- `atualizarPrecoAcrescido(self, event, win):` Atualiza a exibição do preço acrescido.**

 def *atualizarPrecoAcrescido*(self, event, win):

*try*:

            # *Obtendo o valor do preço informado pelo usuário*

            preco = float(self.txtPreco.get())

            # *Calculando o novo preço acrescido*

            preco\_acrescido = preco \* 1.10

            # *Atualizando o valor do preço acrescido na label*

            self.lblPrecoAcrescido.config(text=f"Preço acrescido: {preco\_acrescido}")

*except* Exception *as* e:

            print('Erro ao atualizar preço acrescido:', e)

**- `fLimparTela(self):` Limpa os campos da interface gráfica.**

 def *fLimparTela*(self):

*try*:

            self.txtCodigo.delete(0, tk.END)

            self.txtNome.delete(0, tk.END)

            self.txtPreco.delete(0, tk.END)

            print('Campos Limpos!')

*except*:

            print('Não foi possível limpar os campos.')

**- `fabrirTelaSelect(self):` Abre uma nova tela para seleção de produtos.**

 def *fLimparTela*(self):

*try*:

            self.txtCodigo.delete(0, tk.END)

            self.txtNome.delete(0, tk.END)

            self.txtPreco.delete(0, tk.END)

            print('Campos Limpos!')

*except*:

            print('Não foi possível limpar os campos.')

**Interface gráfica**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**Tela de atualização**

Interface gráfica do usuário

Descrição gerada automaticamente com confiança baixa

**Tela de consultar produtos(SELECT)**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**Executando o Projeto**

**1. Instale as dependências:**

**pip install psycopg2**

**pip install ttkbootstrap (para ver o tema da interface)**

**2. Execute o script `tela.py` para iniciar a aplicação.**

**Conclusão**

O projeto de cadastro de produtos em Python usando a Metodologia RAD é uma demonstração prática de como é possível desenvolver rapidamente uma aplicação funcional e eficiente, implementando um CRUD (Create, Read, Update, Delete) com um banco de dados PostgreSQL. A estrutura simplificada e ágil proporciona uma experiência de desenvolvimento rápida, adequada para projetos de escopo mais limitado. Este projeto serve como uma sólida base para aplicações mais complexas e extensíveis, demonstrando os benefícios de uma abordagem ágil e colaborativa.