



**FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR
CENTRO DE CIENCIAS TECNOLOGIAS (CCT)
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

PROGRAMAÇÃO FUNCIONAL : FORMULÁRIO DE CADASTRO

**INÁCIO DE OLIVEIRA MENEZES – 23182025
MIGUEL ÂNGELO RUSSO NETO - 2315291
MAURICIO DE SOUZA VIANA - 2314676
ESTHEFANE RAELY DE CASTRO SOUZA - 2318048
CARLA DANIELE DUARTE DE SOUSA - 2424142**

Fortaleza-CE

2025

1. Introdução

Desenvolvemos um aplicativo simples para cadastro de usuários utilizando a linguagem Python, com as bibliotecas Flet e Hashlib. O projeto foi elaborado com o objetivo de aplicar os princípios da programação funcional. Além disso, a maior parte da equipe utilizou a IDE VS Code durante o desenvolvimento.

2. Objetivos

Cadastro de usuários.

3. Atribuições do projeto

INTEGRANTE	TAREFA
Inácio Oliveira, Miguel Ângelo	Implementação do código e requisitos funcionais.
Mauricio Viana	Implementação do código e requisitos não funcionais.
Carla Daniele	Implementação do código e testes
Esthefane Raely	Documentação dos requisitos

4. Documentação; Requisito funcional ;

Descrição	Código
• O sistema deve permitir o cadastro de usuários com e-mail e senha	Função <i>adicionar_usuario</i> dentro da closure <i>gerenciar_credenciais</i>
• O sistema deve armazenar as senhas de forma segura.	Função <i>hash_senha</i>
• O sistema deve permitir listar todos os usuários cadastrados.	Função <i>listar_usuarios</i> dentro da closure <i>gerenciar_credenciais</i>
• O sistema deve impedir o cadastro de um mesmo e-mail mais de uma vez.	Função <i>verificar_usuario</i>
• A interface deve permitir a interação do usuário para cadastrar e visualizar os usuários	Função <i>main</i> usando componentes do Flet.

Requisitos não funcionais:

Descrição	Código
•O sistema deve garantir a segurança das credenciais.	Função <i>hash_senha</i> utilizando <i>hashlib.sha256</i>
•A interface deve ser responsiva e intuitiva.	Implementado com a biblioteca Flet na função <i>main</i>
• O sistema deve ser modular e seguir princípios da programação funcional.	Funções puras e estruturas funcionais, como closures e funções de alta ordem

5. Código fonte do sistema:

5.1 Backend Python

```
EXPLORER  ...  cadastro_usuarios_flet.py  grupo1_ads_pf.jfif  teste_sistem.py
PROGRAMAÇÃO FUNCIONAL
  cadastro_flet.py
  cadastro_usuarios_flet.py
  grupo1_ads_pf.jfif
  import tkinter as tk.py
  teste_sistem.py

cadastro_usuarios_flet.py  gerenciador_credenciais
1  import flet as ft
2  import hashlib
3  import json
4
5  #Função para o gerenciador de credenciais de usuario implementado por Miguel Ângelo
6  def gerenciador_credenciais():  #Função de Ordem Alta
7      #Criado um banco fake para o armazenamento das credenciais
8      credenciais_local = {}
9      def adicionar_usuarios(nome, email, senha):
10         if email in credenciais_local:
11             return "Erro E-mail já cadastrado"
12         credenciais_local[email] = {"nome": nome, "senha": hashlib.sha256(senha.encode()).hexdigest()}
13         return "Usuário cadastrado com sucesso!"
14     #Verificação de usuario criado por Estephane Raely usando a função Lambda
15     verifica_usuario = lambda email: email in credenciais_local
16     #Listagem de usuarios criado por Mauricio Viana
17     def listar_usuarios():
18         return [{"dados": "nome"}, email] for email, dados in credenciais_local.items()
19
20     return adicionar_usuarios, verifica_usuario, listar_usuarios
21
22 adicionar_usuarios, verificar_usuario, listar_usuarios = gerenciador_credenciais()
23
```

5.2 Frontend Framework Flet

```
EXPLORER  ...  cadastro_usuarios_flet.py  grupo1_ads_pf.jfif  teste_sistem.py
PROGRAMAÇÃO FUNCIONAL
  cadastro_flet.py
  cadastro_usuarios_flet.py
  grupo1_ads_pf.jfif
  import tkinter as tk.py
  teste_sistem.py

cadastro_usuarios_flet.py  gerenciador_credenciais
24  #A interface do flet python foi criada e implementada por Inacio Oliveira e Carla Daniele
25  def main(page: ft.Page):
26      #título da página
27      page.title = "Cadastro de Usuarios > Grupo"
28      page.theme_mode = ft.ThemeMode.LIGHT
29
30      #labels da tela inicial
31      nome = ft.TextField(label="Nome")
32      email = ft.TextField(label="Email")
33      senha = ft.TextField(label="Senha", password=True)
34      mensagem = ft.Text()
35
36      #imagem da janela do projeto
37      imagem = ft.Image(src="grupo1_ads_pf.jfif", width=150, height=150)
38
39      #funções da tela do front flet enviadas para back implementada por Inacio Oliveira
40      def cadastrar():
41          if verificar_usuario(email.value):
42              mensagem.value = "Erro email já cadastrado!"
43          else:
44              resultado = adicionar_usuarios(nome.value, email.value, senha.value)
45              mensagem.value = resultado
46              nome.value = ""
47              email.value = ""
48              senha.value = ""
49          page.update()
50      btn_cadastrar = ft.ElevatedButton("Cadastrar", on_click=cadastrar)
51
52      def listar_usuarios_handler():
53          usuarios = [ft.Text(f"Nome: {nome} / Email: {email}") for nome, email in listar_usuarios()]
54          page.add(*usuarios)
55
56      btn_listar = ft.ElevatedButton("Listar Usuários", on_click=listar_usuarios_handler)
57
58      page.add(imagem, nome, email, senha, btn_cadastrar, btn_listar, mensagem)
59
60      ft.app(target=main)
61
62
63
64
```

Todo código fonte do projeto esta disponível através do link do Github:

https://github.com/menezes1986/Programacao-funcional/blob/main/cadastro_usuarios_flet.py

6. Conceitos de Programação Funcional:

•Função *lambda*:

```
#Verificação de usuario criado por Estephane Raely usando a função lambda
verifica_usuario = lambda email: email in credenciais_local
```

Função verifica se existe um Email já cadastrado para cada usuários dentro da lista de credenciais locais.

•List Comprehension:

```
def listar_usuarios_handler(e):
    usuarios = [ft.Text(f"Nome: {nome} | Email: {email}") for nome, email in listar_usuarios()]
    page.add(*usuarios)
```

Ao chamar a função **listar_usuarios_handler(e)**, é retornado em tela a lista de usuários já cadastrados.

Função de alta ordem e clousure:

```
#Função para o gerenciador de credenciais de usuario implementado por Miguel Angêlo
def gerenciar_credenciais(): #Função de Ordem Alta
    #criado um banco fake para o armazenamento das credenciais
    credenciais_local = {}
    def adicionar_usuarios(nome, email,senha):
        if email in credenciais_local:
            return "Erro E-mail já cadastrado"
        credenciais_local[email] = {"nome": nome, "senha": hashlib.sha256(senha.encode()).hexdigest()}
        return "Usuário cadastrado com sucesso!"
    #Verificação de usuario criado por Estephane Raely usando a função lambda
    verifica_usuario = lambda email: email in credenciais_local
    #listagem de usuarios criado por Mauricio Viana
    def listar_usuarios():
        return [(dados["nome"], email) for email,dados in credenciais_local.items()]

    return adicionar_usuarios,verifica_usuario,listar_usuarios

adicionar_usuarios, verificar_usuario, listar_usuarios = gerenciar_credenciais()
```

A closure foi implementada na função **gerenciar_credenciais**, que encapsula **credenciais_local** e fornece funções internas para manipulação dos dados. Já a função de alta ordem é utilizada na função **verificar_usuario_existente**, que recebe uma função como parâmetro e retorna uma nova função.

7. Teste de Validação:

- **Primeiro Teste** : Testar o cadastro de um novo usuário com e-mail e senha válidos.

Passos:

1. Inserir o e-mail válido *usuario@exemplo.com* no campo de e-mail.
2. Inserir a senha válida *Senha@123* no campo de senha.
3. Clicar no botão "**Cadastrar**".

Resultado esperado: O sistema deve retornar a mensagem "**Usuário cadastrado com sucesso!**" e o e-mail *usuario@exemplo.com* deve ser adicionado ao banco de dados fictício com a senha criptografada.

Resultado obtido: *Usuário cadastrado com sucesso!*

E o banco de dados fictício *credenciais_local* agora conterá a chave *usuario@exemplo.com* com o valor sendo a senha criptografada.

- Segundo Teste: Cadastro de E-mail Duplicado

Passos:

1. Inserir o e-mail *usuario@exemplo.com* no campo de e-mail.
2. Inserir a senha válida *Senha@123* no campo de senha.
3. Clicar no botão "**Cadastrar**".

Resultado esperado:

O sistema deve retornar a mensagem "**Usuário já cadastrado!**" e não deve adicionar o usuário novamente

Resultado obtido: Mensagem exibida será:

Usuário já cadastrado!

E o banco de dados fictício *credenciais_local* não será alterado.

8. Observações:

Para garantir a segurança e a criptografia das senhas dos usuários, utilizamos a biblioteca *hashlib* com o algoritmo **SHA-256**. Durante a implementação, contamos com o auxílio do ChatGPT para revisar o código, o que nos permitiu evitar a programação extensa e a repetição de comandos, melhorando a eficiência e a qualidade do desenvolvimento.