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 ;

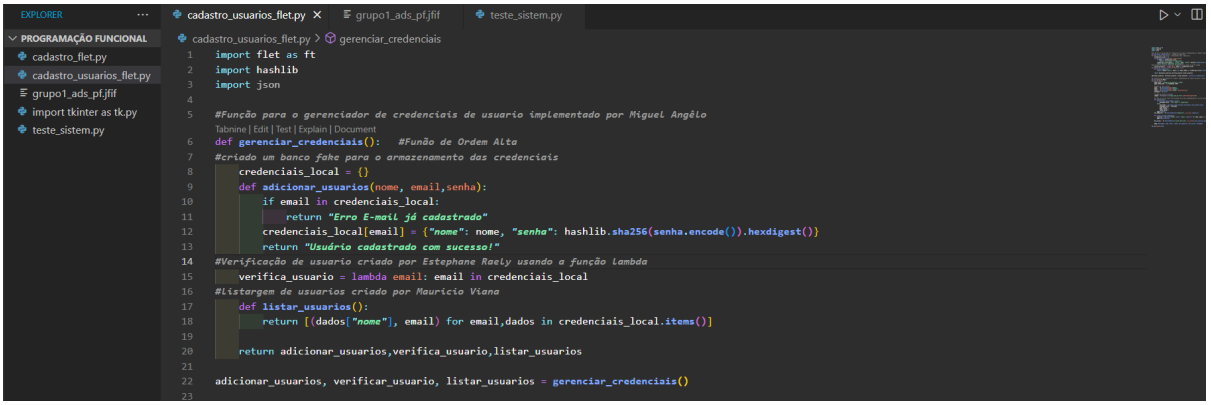
Descrição	Código
• O sistema deve permitir o cadastro de usuários com e-mail e senha	Função ***adicionar_usuario*** dentro da closure ***gerenciar_credenciais***
• O sistema deve armazenar as senhas de forma segura.	Função ***hash_senha***
• O sistema deve permitir listar todos os usuários cadastrados.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

Requisitos não funcionais:

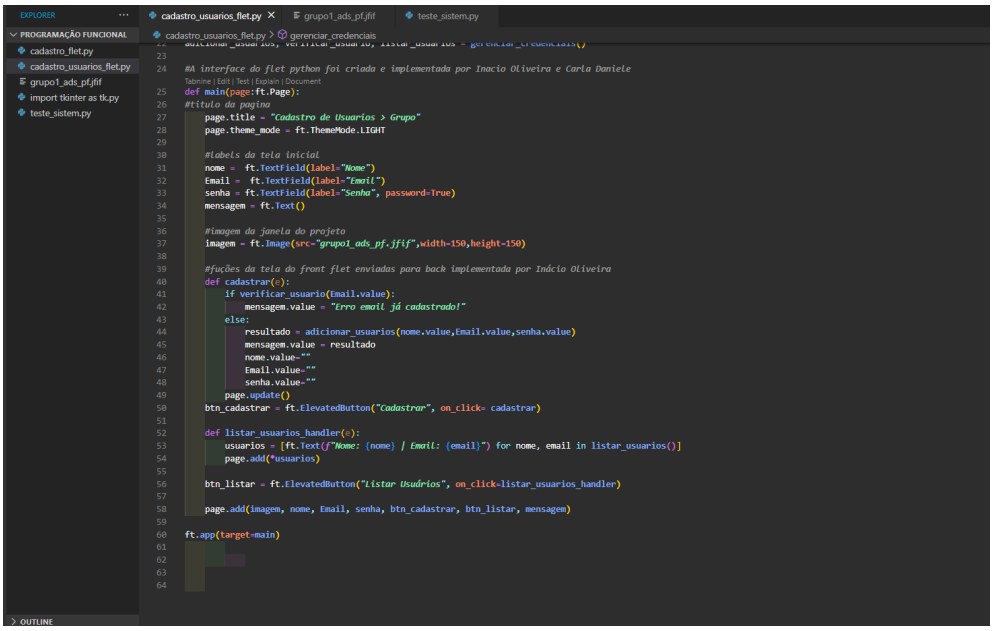
Descrição	Código
•O sistema deve garantir a segurança das credenciais.	Função ***hash_senha*** utilizando ***hashlib.sha256***
•A interface deve ser responsiva e intuitiva.	Implementado com a biblioteca Flet na função ***main***
• O sistema deve ser modular e seguir princípios da programação funcional.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

5. Código fonte do sistema:

5.1 Backend Pyhton



5.2 Frontend Framework Flet



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_usuários_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():   #Funão 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
#listargem 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.