



UNIVERSIDADE DE FORTALEZA
CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: PROGRAMAÇÃO FUNCIONAL

Relatório do Trabalho Final – Documentação de requisitos

INTEGRANTES:

Isabele Silva Souza Monteiro - 2222868

FORTALEZA-CE
Setembro/2024

1. Definição de Papéis

Como o trabalho será desenvolvido individualmente, todos os papéis descritos serão desempenhados por uma única pessoa:

- Documentação dos Requisitos: Responsável por criar e organizar o documento de requisitos funcionais e não funcionais. Neste caso, a tarefa foi realizada por mim.
- Implementação do Código: Responsável por escrever o código, seguindo as boas práticas de programação funcional e atendendo aos requisitos. Também foi realizada por mim.
- Testes: Responsável por implementar e executar os testes para garantir que todas as funcionalidades do sistema estejam funcionando corretamente. Esta tarefa foi realizada por mim novamente.

2. Documento de Requisitos

- **Requisitos Funcionais**

2.1 - Adicionar Contato: O sistema deve permitir que o usuário adicione novos contatos com nome, telefone e e-mail.

- Implementado na função: *adicionar_contato*.
- Esta função recebe três parâmetros (nome, telefone, e-mail) e adiciona um novo contato à lista de contatos. Ela utiliza uma lista interna para armazenar essas informações.

2.2 - Remover Contato: O sistema deve permitir a remoção de um contato existente pelo nome.

- Implementado na função: *remover_contato*.
- A função filtra a lista de contatos e remove aquele cujo nome corresponde ao parâmetro fornecido.

2.3 - Buscar Contato: O sistema deve permitir a busca de contatos pelo nome.

- Implementado na função: *buscar_contato*.
- Esta função percorre a lista de contatos e retorna aquele cujo nome corresponde ao parâmetro fornecido.

2.4 - Listar Contatos: O sistema deve listar todos os contatos armazenados.

- Implementado na função: *listar_contatos*.
- A função retorna todos os contatos presentes na lista. Caso a lista esteja vazia, retorna uma mensagem indicando que não há contatos.

- **Requisitos Não Funcionais**

2.5 - Desempenho: O sistema é capaz de lidar com uma grande quantidade de contatos de forma eficiente.

- O uso de programação funcional, como funções de alta ordem, list comprehensions e closures, garante que o código seja executado de forma rápida e eficiente.

2.6 - Facilidade de Manutenção: O código é modular e fácil de manter.

- Todas as funcionalidades estão separadas em funções distintas, facilitando futuras manutenções ou adições de novas funcionalidades.

3. Mapeamento dos Requisitos no Código

- **Funções Mapeadas:**

- Adicionar Contato: A função *adicionar_contato* implementa o requisito de adicionar novos contatos.

- Remover Contato: A função *remover_contato* implementa o requisito de remover contatos existentes.

- Buscar Contato: A função *buscar_contato* implementa o requisito de buscar contatos pelo nome.

- Listar Contatos: A função *listar_contatos* implementa o requisito de listar todos os contatos.

4. Utilização dos Conceitos de Programação Funcional

4.1 - Função Lambda:

- Utilizada na função: *formatar_contato*.
- A função lambda *formatar_contato* é responsável por formatar a exibição de cada contato no formato "nome - telefone - e-mail". Ela torna o código mais conciso e legível.

4.2 - List Comprehension:

- Utilizada na função: *listar_contatos*.
- A função *listar_contatos* utiliza uma list comprehension para percorrer a lista de contatos e retornar todos os contatos armazenados. Isso melhora o desempenho e a legibilidade do código.

4.3 - Closure:

- Utilizada na função: *agenda*.
- A função *agenda* contém uma closure que encapsula a lista de contatos, permitindo que as funções de adicionar, remover, buscar e listar contatos sejam acessadas sem expor diretamente a lista de contatos.

4.4 - Função de Alta Ordem:

- Utilizada na função: *processar_contatos*.
- A função *processar_contatos* recebe uma função como argumento (*formatar_contato*) e a aplica a todos os contatos na lista. Isso explica o uso da função de alta ordem, onde funções são passadas como argumentos para outras funções.

5. Testes Automatizados

- Descrição dos Testes: O código foi testado manualmente com exemplos de contatos para garantir que todas as funcionalidades, como adicionar, remover, buscar e listar contatos, estejam funcionando corretamente.
- Casos de Teste:
 - Adicionar Contato: Foi testada a adição de três contatos, e a função retornou as mensagens corretas.
 - Remover Contato: Um contato foi removido com sucesso, e a tentativa de removê-lo novamente retornou a mensagem de erro apropriada.
 - Buscar Contato: A busca por contatos específicos retornou os resultados corretos. Buscas por contatos inexistentes retornaram a mensagem de erro.
 - Listar Contatos: Após a remoção de um contato, a lista foi atualizada corretamente e exibiu apenas os contatos restantes.

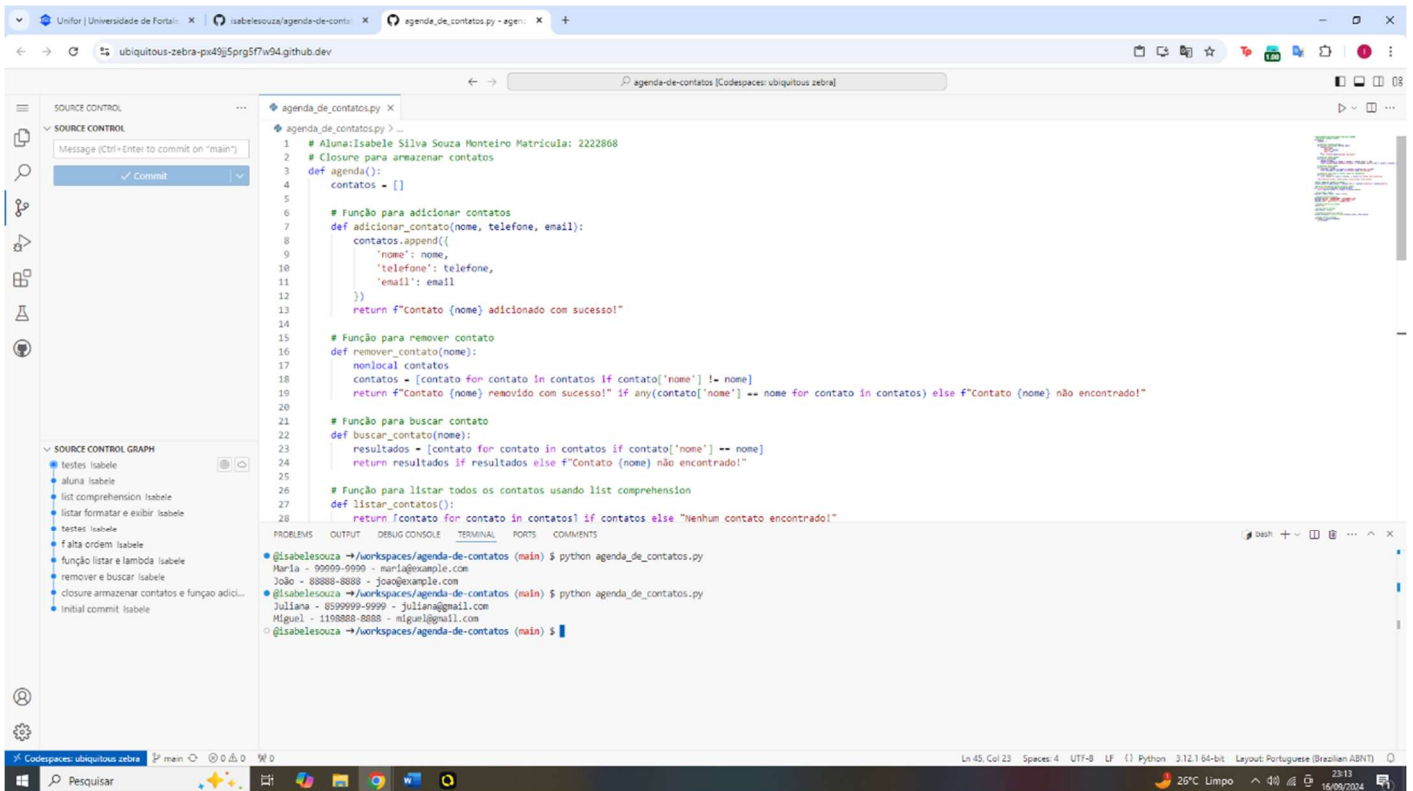
6. Conclusão

O código foi implementado e testado seguindo todos os requisitos funcionais e não funcionais descritos. Todas as construções de programação funcional foram implementadas, garantindo o uso adequado de lambdas, list comprehensions, closures e funções de alta ordem.

Github com o código fonte e o documento:

<https://github.com/isabelesouza/agenda-de-contatos>

7- Anexos



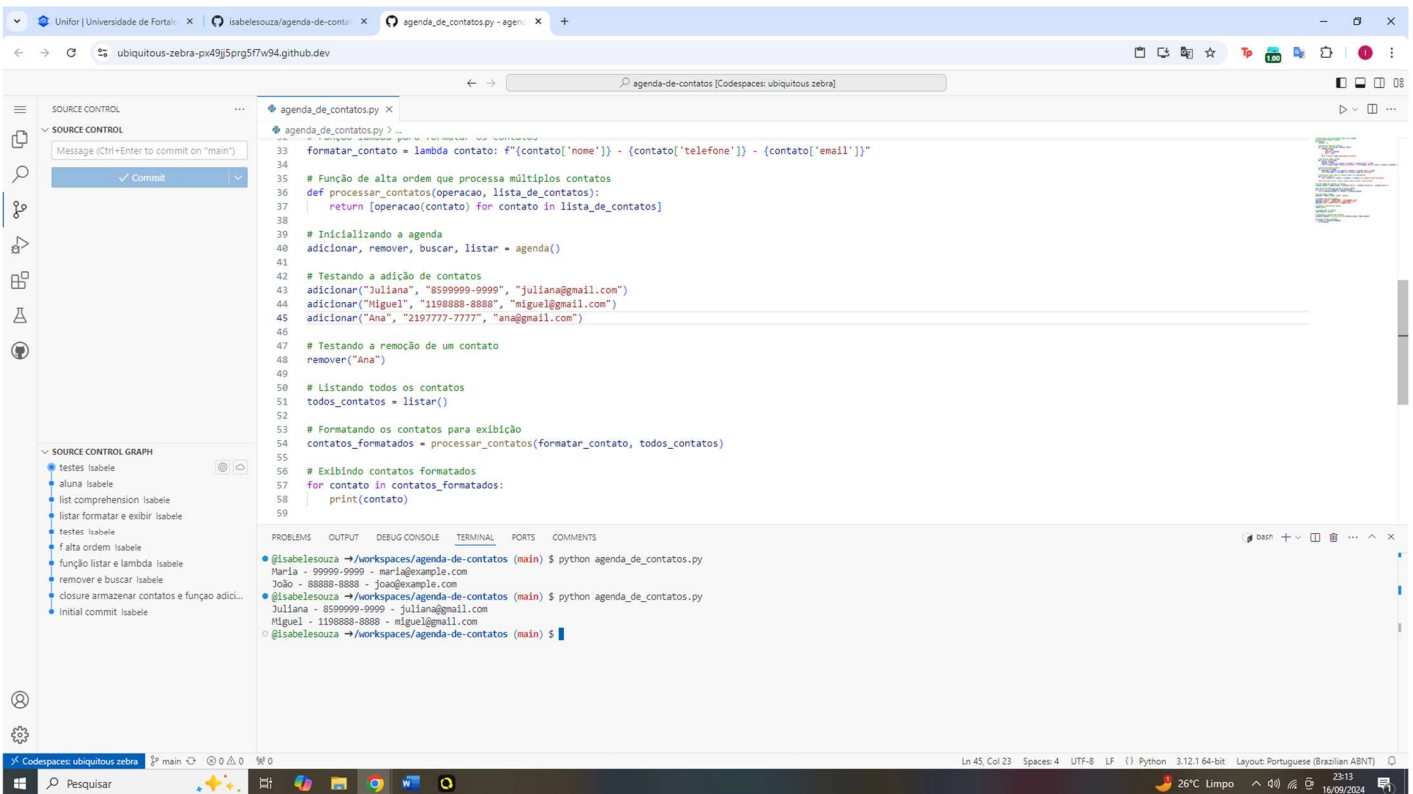
The screenshot shows a GitHub Codespace environment with a browser window at the top displaying the repository URL. Below the browser, the 'SOURCE CONTROL' panel on the left shows a commit message and a 'Commit' button. The main editor displays the file 'agenda_de_contatos.py' with the following Python code:

```
1 # Aluna:Isabele Silva Souza Monteiro Matricula: 2222868
2 # Closure para armazenar contatos
3 def agenda():
4     contatos = []
5
6     # Função para adicionar contatos
7     def adicionar_contato(nome, telefone, email):
8         contatos.append({
9             'nome': nome,
10            'telefone': telefone,
11            'email': email
12        })
13     return f"Contato {nome} adicionado com sucesso!"
14
15 # Função para remover contato
16 def remover_contato(nome):
17     nonlocal contatos
18     contatos = [contato for contato in contatos if contato['nome'] != nome]
19     return f"Contato {nome} removido com sucesso!" if any(contato['nome'] == nome for contato in contatos) else f"Contato {nome} não encontrado!"
20
21 # Função para buscar contato
22 def buscar_contato(nome):
23     resultados = [contato for contato in contatos if contato['nome'] == nome]
24     return resultados if resultados else f"Contato {nome} não encontrado!"
25
26 # Função para listar todos os contatos usando list comprehension
27 def listar_contatos():
28     return [contato for contato in contatos] if contatos else "Nenhum contato encontrado!"
```

The bottom panel shows the 'TERMINAL' output with the following commands and results:

```
@isabelesouza -> /workspaces/agenda-de-contatos (main) $ python agenda_de_contatos.py
Maria - 99999-9999 - maria@example.com
João - 88888-8888 - joao@example.com
@isabelesouza -> /workspaces/agenda-de-contatos (main) $ python agenda_de_contatos.py
Juliana - 8599999-9999 - juliana@gmail.com
Miguel - 1198888-8888 - miguel@gmail.com
@isabelesouza -> /workspaces/agenda-de-contatos (main) $
```

Ambiente de desenvolvimento usado;foi utilizado o codespace do github.



The screenshot shows a GitHub Codespace environment with a browser window at the top displaying the repository URL. Below the browser, the 'SOURCE CONTROL' panel on the left shows a commit message and a 'Commit' button. The main editor displays the file 'agenda_de_contatos.py' with the following Python code:

```
33 formatar_contato = lambda contato: f"{contato['nome']} - {contato['telefone']} - {contato['email']}"
34
35 # Função de alta ordem que processa múltiplos contatos
36 def processar_contatos(operacao, lista_de_contatos):
37     return [operacao(contato) for contato in lista_de_contatos]
38
39 # Inicializando a agenda
40 adicionar, remover, buscar, listar = agenda()
41
42 # Testando a adição de contatos
43 adicionar("Juliana", "8599999-9999", "juliana@gmail.com")
44 adicionar("Miguel", "1198888-8888", "miguel@gmail.com")
45 adicionar("Ana", "2197777-7777", "ana@gmail.com")
46
47 # Testando a remoção de um contato
48 remover("Ana")
49
50 # Listando todos os contatos
51 todos_contatos = listar()
52
53 # Formatando os contatos para exibição
54 contatos_formatados = processar_contatos(formatar_contato, todos_contatos)
55
56 # Exibindo contatos formatados
57 for contato in contatos_formatados:
58     print(contato)
59
```

The bottom panel shows the 'TERMINAL' output with the following commands and results:

```
@isabelesouza -> /workspaces/agenda-de-contatos (main) $ python agenda_de_contatos.py
Maria - 99999-9999 - maria@example.com
João - 88888-8888 - joao@example.com
@isabelesouza -> /workspaces/agenda-de-contatos (main) $ python agenda_de_contatos.py
Juliana - 8599999-9999 - juliana@gmail.com
Miguel - 1198888-8888 - miguel@gmail.com
@isabelesouza -> /workspaces/agenda-de-contatos (main) $
```

Ambiente de desenvolvimento usado;foi utilizado o codespace do github. Prova de que são retornados valores corretos ao chamar as funções do código.