

Universidade de Fortaleza

Análise e Desenvolvimento de Sistemas

Semestre 5

Programação Funcional

Equipe:

Samuel Vidal Roberto – 2213826 - samuelVRcoder

Ian de Moura Façanha – 2213843 - IanMouraF

Icaro Oliveira- 2317508 - IcaroIN

Tulio Victor Pereira Santos - 2317811 - tuliopss

GitHub da atividade:

<https://github.com/tuliopss/employee-management-python>

Papéis:

- Documentação dos requisitos : Samuel Vidal e Icaro Oliveira.
- Código : Ian de Moura , Túlio Victor e Samuel Vidal.
- Testes : Icaro Oliveira.

## 1 Requisitos

Requisitos funcionais	Requisitos não funcionais
1 - O usuário poderá inserir novos funcionários com nome, email, função e salário	1 - O sistema tem que ser desenvolvido utilizando a linguagem Python.
2 - O usuário poderá excluir um funcionário a partir do seu ID	2 - O sistema deve rodar via terminal
3 - O usuário poderá pesquisar um funcionário específico a partir do seu id.	3 - O sistema deve estabelecer uma conexão persistente com o banco de dados MySQL
4 - O usuário poderá ver todos os funcionários registrados	4 - O sistema deve ser manutenível
5 - O usuário poderá editar dados de um funcionário, com exceção do seu e-mail	5 - O sistema deve ser desenvolvido seguindo o máximo possível dos paradigmas de programação funcional.
6 - O usuário pode pesquisar os funcionários pela a sua função	6 - A interface do usuário deve ser intuitiva e de fácil utilização, permitindo que os usuários realizem suas tarefas de forma rápida e sem dificuldades.
7 - O usuário poderá interagir com o sistema através de inputs no terminal.	

### **Mapeamento da codificação dos requisitos**

RF1 está na função `insertEmployee(name, email, role, salary)`, implementada no arquivo `repositoryEmployee.py`

RF2 está na função `deleteEmployee(id)`, implementada no arquivo `repositoryEmployee.py`

RF3 está na função `getEmployeeById(id)` implementada no arquivo `repositoryEmployee.py`

RF4 está na função `getEmployees`, implementada no arquivo `repositoryEmployee.py`

RF5 está na função `editEmployee(id)`, implementada no arquivo `repositoryEmployee.py`

RF6: está na função `searchEmployeesByRole(role)`, implementada no arquivo `repositoryEmployee.py`

RF7: a interação com o sistema ocorre com a apresentação de um menu com as opções de funções que o usuário pode realizar, presente na função `printMenu()`. A interação com os inputs está presente nas funções: `runOption()`, `inputInsertEmployee`, `inputGetEmployeeById`, `inputDeleteEmployee`, `inputEditEmployee`, `inputGetEmployeeByRole`. e é executado a partir da função `executeMenu()` Tudo isso no arquivo `menu.py`

## Identificação dos conceitos de programação funcional

Função lambda e List Comprehension:

```
def searchEmployeesByRole(role):
    try:
        connection, cursor = openConn()
        query = f"SELECT * FROM employees"
        cursor.execute(query)

        listEmployees = cursor.fetchall()

        listEmployeesByRole = lambda role : [emp for emp in listEmployees if emp[3].startswith(role) ]

        if(len(listEmployeesByRole(role)) == 0):
            return print(f'Employees with role "{role}" not found')

        print(listEmployeesByRole(role))

        return listEmployeesByRole(role)
    except Exception as e:
```

Função de continuação

```
def inputInsertEmployee(callback):
    name = input("Enter employee name: ")
    email = input("Enter employee email: ")
    role = input("Enter employee role: ")
    salary = float(input("Enter employee salary: "))
    callback(name, email, role, salary)

def inputGetEmployeeById(callback):
    id = input('Enter the employee ID: ')
    if not id:
        return runOption()

    callback(id, printEmp)
```

```
options = {
    1: lambda: inputInsertEmployee(insertEmployee),
    2: lambda: inputDeleteEmployee(deleteEmployee),
    3: lambda: inputGetEmployeeById(getEmployeeById),
    4: getEmployees,
    5: lambda: inputGetEmployeeForRole(searchEmployeesByRole),
    6: lambda: inputEditEmployee(editEmployee),
    7: exitProgram,
}
```

Função de alta ordem (employeeRepository) e closures(insert, edit, gets, search, delete employees )

```
def employeeRepository():
    connection, cursor = openConn()

    if connection is None or cursor is None:
        print("Failed to open connection. Check your database configuration.")
        return None

> def insertEmployee(name, email, role, salary): ...
> def editEmployee(id, updatedName, updatedRole, updatedSalary): ...
> def getEmployees(): ...
> def getEmployeeById(id, printCb): ...
> def deleteEmployee(id): ...
> def searchEmployeesByRole(role): ...
```

## Monad

```
def insertEmployee(name, email, role, salary):
    try:
        connection, cursor = get_connection_and_cursor()

        dicEmployee = {"name": name, "email": email, "role": role, "salary": salary}
        employeesColumns = ', '.join(dicEmployee.keys())

        query = f"INSERT INTO employees ({employeesColumns}) VALUES ('{dicEmployee['name']}', '{dicEmployee['email']}', '{dicEmployee['role']}', '{dicEmployee['salary']}')"
        cursor.execute(query)
        connection.commit()

        closeConn(connection, cursor)
        print("Employee registered")
        return ResultMonad("Employee registered")
    except Exception as e:
        return ResultMonad(str(e), success=False)
```

```
resultMonad.py > ...
1  class ResultMonad:
2      def __init__(self, value, success=True):
3          self.value = value
4          self.success = success
5
6      def bind(self, func):
7          if self.success:
8              try:
9                  result = func(self.value)
10                 return ResultMonad(result)
11             except Exception as e:
12                 return ResultMonad(str(e), success=False)
13         else:
14             return self
15
```

## 2 Casos de Teste

### RF1: Inserir funcionário (insertEmployee)

Caso de Teste 1.1: Inserir funcionário com dados válidos

- Descrição: Verificar se é possível inserir um funcionário com dados válidos.
- Entradas: Nome, E-mail, Cargo, Salário
- Resultado Esperado: Funcionário inserido com sucesso no sistema.

Caso de Teste 1.2: Inserir funcionário com dados inválidos

- Descrição: Verificar se o sistema rejeita a inserção de um funcionário com dados inválidos.
- Entradas: Nome vazio, E-mail inválido, Cargo vazio, Salário negativo
- Resultado Esperado: Sistema exibe mensagem de erro e não realiza a inserção do funcionário.

### RF2: Excluir funcionário (deleteEmployee)

Caso de Teste 2.1: Excluir funcionário existente

- Descrição: Verificar se é possível excluir um funcionário existente no sistema.
- Entrada: ID do funcionário a ser excluído
- Resultado Esperado: Funcionário é removido com sucesso do sistema.

Caso de Teste 2.2: Excluir funcionário inexistente

- Descrição: Verificar se o sistema trata adequadamente a exclusão de um funcionário inexistente.
- Entrada: ID de um funcionário não cadastrado
- Resultado Esperado: Sistema exibe mensagem de erro informando que o funcionário não foi encontrado.

### RF3: Buscar funcionário por ID (getEmployeeById)

Caso de Teste 3.1: Buscar funcionário por ID válido

- Descrição: Verificar se é possível buscar um funcionário pelo ID válido.
- Entrada: ID de um funcionário cadastrado

- Resultado Esperado: Sistema exibe os dados do funcionário encontrado.

#### Caso de Teste 3.2: Buscar funcionário por ID inválido

- Descrição: Verificar se o sistema trata adequadamente a busca por um ID inválido.
- Entrada: ID de um funcionário não cadastrado
- Resultado Esperado: Sistema exibe mensagem informando que o funcionário não foi encontrado.

### **RF4: Listar todos os funcionários (getEmployees)**

#### Caso de Teste 4.1: Listar funcionários cadastrados

- Descrição: Verificar se o sistema lista corretamente todos os funcionários cadastrados.
- Resultado Esperado: Sistema exibe a lista de funcionários com seus dados.

### **RF5: Editar dados de um funcionário (editEmployee)**

#### Caso de Teste 5.1: Editar dados de funcionário existente

- Descrição: Verificar se é possível editar os dados de um funcionário existente.
- Entradas: ID do funcionário a ser editado, Novos dados (Nome, E-mail, Cargo, Salário)
- Resultado Esperado: Dados do funcionário são atualizados com sucesso.

#### Caso de Teste 5.2: Editar dados de funcionário inexistente

- Descrição: Verificar se o sistema trata adequadamente a edição de um funcionário inexistente.
- Entradas: ID de um funcionário não cadastrado, Novos dados (Nome, E-mail, Cargo, Salário)
- Resultado Esperado: Sistema exibe mensagem de erro informando que o funcionário não foi encontrado.

### **RF6: Buscar funcionários por cargo**

#### **(searchEmployeesByRole)**



#### Caso de Teste 6.1: Buscar funcionários por cargo existente

- Descrição: Verificar se é possível buscar funcionários por um cargo existente.
- Entrada: Cargo válido
- Resultado Esperado: Sistema exibe a lista de funcionários com o cargo especificado.

#### Caso de Teste 6.2: Buscar funcionários por cargo inexistente

- Descrição: Verificar se o sistema trata adequadamente a busca por um cargo inexistente.
- Entrada: Cargo não cadastrado no sistema
- Resultado Esperado: Sistema exibe mensagem informando que não há funcionários com o cargo especificado.

### **RF7: Menu interativo para interação com o usuário**

#### Caso de Teste 7.1: Exibir menu corretamente

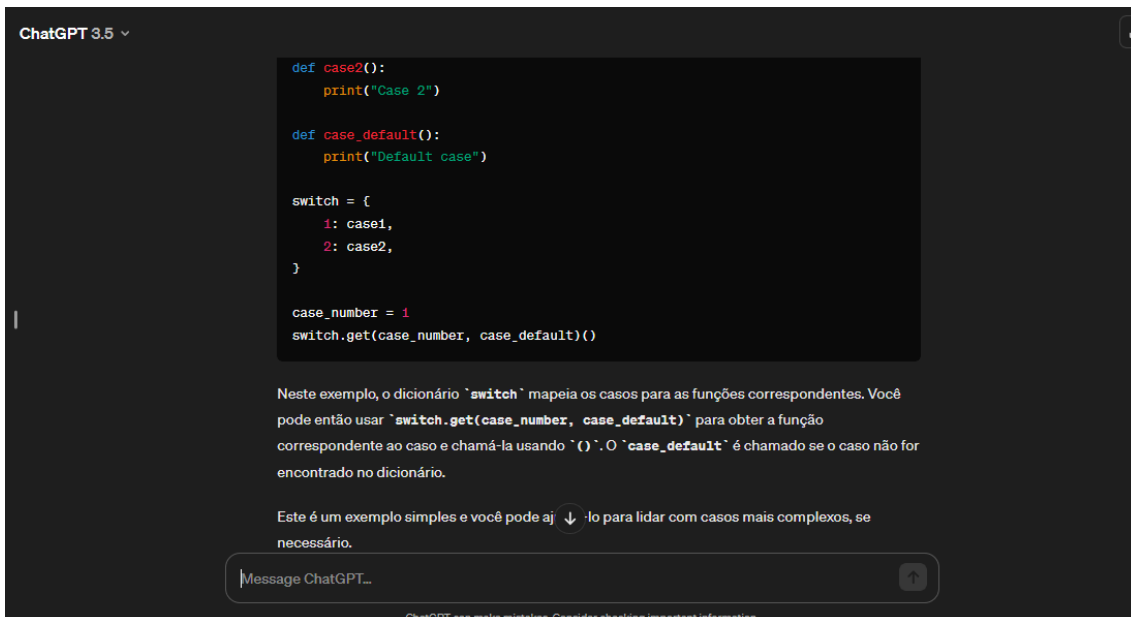
- Descrição: Verificar se o sistema exibe o menu interativo corretamente.
- Resultado Esperado: Menu com opções disponíveis para o usuário.

#### Caso de Teste 7.2: Selecionar opção do menu

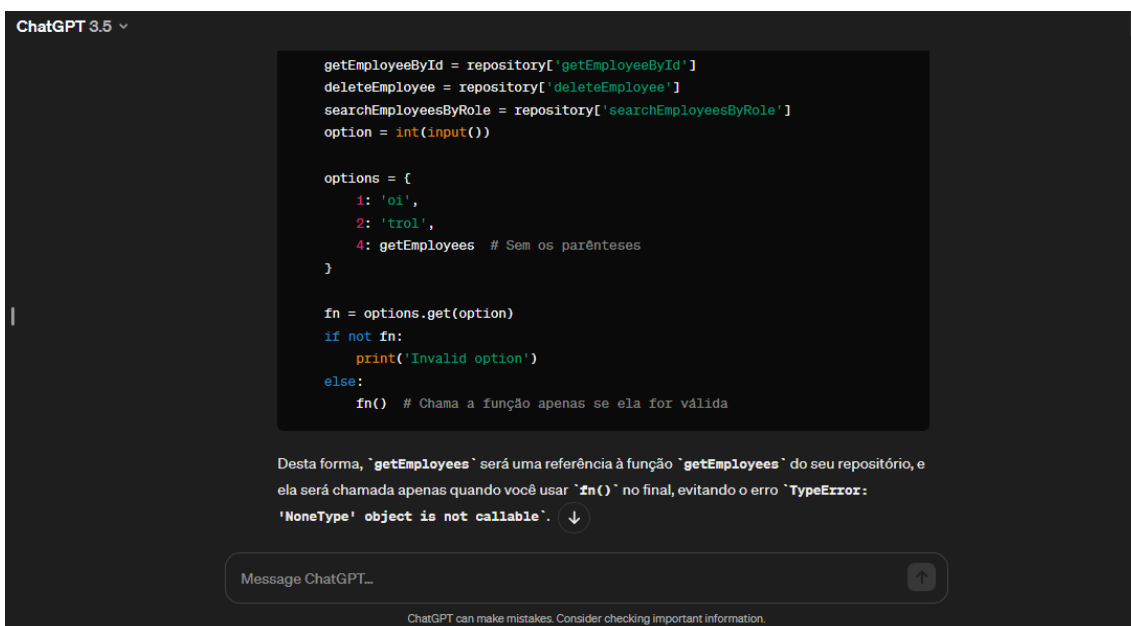
- Descrição: Verificar se o sistema executa a função correspondente à opção selecionada pelo usuário.
- Entrada: Opção selecionada pelo usuário
- Resultado Esperado: Sistema executa a função correspondente à opção escolhida.

### **Usos ChatGPT**

Utilizei o chatbot para perguntar como se usa o switch-case em Python, sob a programação funcional (não tenho profundidade nessa linguagem) e ele respondeu que não existia essa estrutura e mostrou uma simulação:



Nesse caso, perguntei o porquê do código quebrar depois de acessar uma função que não existe e executar a que eu não escolhi no input:



Aqui, não sabia que a função lambda tinha esse comportamento

ChatGPT 3.5

You

como faço para fazer com que a função que precisa de um parametro, seja declarada sem que execute igual na `getEmployees`

ChatGPT

Para declarar uma função que precise de um parâmetro sem que ela seja executada imediatamente, você pode usar uma função lambda. No caso da `'getEmployees'`, você pode fazer o seguinte:

python

Copy code

```
options = {  
    1: 'oi',  
    2: 'trol',  
    3: lambda: menuGetEmployeeById(getEmployeeById),  
    4: getEmployees  
}
```

Desta forma, a função `'menuGetEmployeeById'` só será chamada quando a opção 3 for selecionada, pois está sendo encapsulada em uma função lambda que não recebe

Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.