

Relatório Solução Etapa 2

Programa IT Academy – Processo Seletivo – Edição #15

Nome: Lucas Mariano Leipnitz de Fraga

Instruções para Execução do Programa

Versão do python: 3.9

Todas as bibliotecas utilizadas são nativas do python

Para rodar o programa é preciso executar o arquivo `manager.py` a partir da pasta `program`, para que ele consiga encontrar o caminho correto para os arquivos `csv` e `db` na pasta `data`. No caso de `UnicodeDecodeError` é preciso salvar o arquivo `csv` no formato `utf-8` com `BOM`.

Introdução

Na etapa dois do processo seletivo, tivemos que desenvolver um programa responsável por ler uma base de dados, disponibilizada em um arquivo `csv`, e executar diversas tarefas em cima dos dados da base. Como muitas das tarefas podiam ser executadas facilmente por SQL query, eu decidi utilizar o pacote `SQLite3`, um pacote de python que permite fazer um banco de dados SQL local, para fazer um bando de dados com os dados do arquivo `csv`.

Após isso, basta chamar o banco de dados para consultar de forma rápida e segura os dados, tratar eles e mostrar para o usuário. Abaixo, cada consulta do banco de dados feita para sua respectiva tarefa:

1. [Consultar medicamento pelo nome]:
`"SELECT * FROM Medicamento WHERE SUBSTANCIA LIKE '%{name}%' AND COMERCIALIZACAO_2020 == 'Sim'"`
2. [Buscar pelo código de barras]:
`SELECT * FROM Medicamento WHERE EAN_1 == '{code}' ORDER BY PMC_0`
3. [Comparativa PIS/COFINS]:
`"SELECT count(PIS_COFINS) FROM Medicamento where PIS_COFINS == '{param}'"`
Onde `param` pode ser "Positiva", "Negativa" ou "Neutra"

A partir deste raciocínio, criei uma arquitetura em camadas do projeto, para que o usuário possa solicitar uma tarefa e, sem com que ele tem acesso ao banco, busque as

informações no banco, faça operações lógicas em cima das informações e imprima o resultado para o usuário.

Arquitetura do Projeto

A arquitetura do projeto é uma arquitetura em camadas, onde existem seis módulos, um que se comunica com o usuário, dois que se comunicam com a base de dados, dois que fazem a intermediação entre o usuário e a base de dados e um que serve para armazenar os dados em memória. Escolhi fazer a aplicação com esta arquitetura pois acho ela simples e fácil de implementar, mas também bem modular por ter seis módulos, e bem robusta, já que poucos módulos se comunicam entre si, diminuindo a possibilidade de passarem bugs uns para os outros e segura, já que o usuário só tem comunicação com o módulo de interface. Abaixo, um diagrama UML da arquitetura:

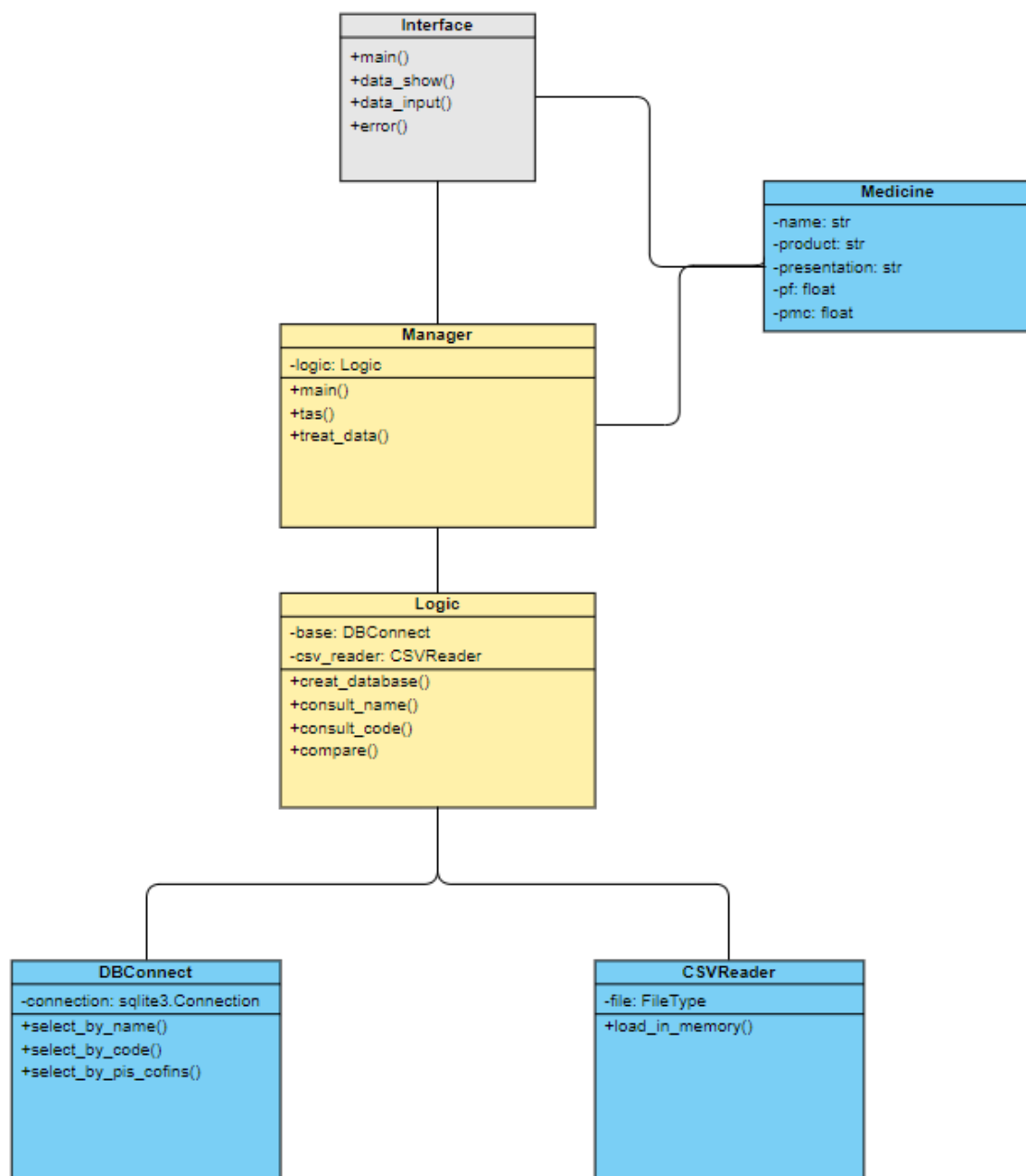


Imagem 1: Diagrama de classes UML da arquitetura.

Módulos

Interface

A classe Interface faz parte do pacote interface e tem o objetivo de se comunicar com o usuário final, imprimindo as opções de tarefas e os resultados para o usuário:

```
##### CONSULTA MEDICAMENTOS #####  
Selecione uma das opções  
1 - Consultar medicamento pelo nome  
2 - Buscar pelo código de barras  
3 - Comparativo PIS/COFINS  
4 - Terminar o programa  
Insira aqui:
```

Imagem 2: Comunicação entre a interface e o usuário via terminal.

Manager

A classe Manager é responsável por fazer a administração entre a interface com o usuário (frontend) e a lógica do programa (backend), armazenar e tratar os dados de entrada e de pesquisa e controlar em que estado o programa está. Ele é o arquivo principal do programa e será o primeiro módulo a ser criado e em seguida, será chamado a sua função main(). Ela começa verificando se o meds.db está na pasta data e, se não estiver, ela chama a função de Logic que será responsável por criar e preencher o banco de dados a partir do arquivo csv. É nela, também, onde está o laço principal do programa, que será executado até o usuário escolher a opção 4- Terminar o programa.

```
...  
main():  
    entrada: nenhuma  
    saída: nenhuma  
    objetivo: primeira função chamada pelo programa. Responsável por receber as informações do usuário, tratá-las e escolher qual tarefa será feita com seus respectivos dados baseado nas  
    Cria base de dados caso ela não exista. Tem o loop principal do programa que ficará ativo até a opção 5 ser selecionada.  
    ...  
def main(self):  
    path_file = os.getcwd() + "\\data\\" + "meds.db"  
    if(not os.path.exists(path_file)):  
        self.logic.create_database()  
    while(not self.exit_program):  
        self.clear_medicines()  
        if(self.option == -1):  
            self.option = self.treat_option(self.interface.main())  
        else:  
            self.data = self.interface.data_input(self.option)  
            if(self.option == 1):  
                self.data = self.treat_data(self.data)  
                self.task_1()  
            elif(self.option == 2):  
                self.data = self.treat_data(self.data)  
                self.task_2()  
            elif(self.option == 3):  
                self.data = self.treat_data(self.data)  
                self.task_3()  
            elif(self.option == 4):  
                self.task_4()
```

Imagem 3: Função main() de Manager, o laço principal do programa.

No laço principal está a chamada da função `main()` de `Interface`, que vai imprimir as mensagens mostradas na imagem 1 e esperar que o usuário digite algo. Além disso, `Manager` tem as funções `treat_option()` e `treat_data` que tratam os dados de entrada do usuário, para que ele não digite opções inválidas, letras no lugar de números e caracteres que possam causar erro na consulta ao banco de dados.

Caso os dados inseridos pelo usuário forem válidos, ela chamará uma das três funções `task()` que vão, então, chamar as funções de `Logic` com os dados passados pelo usuário para fazer o processamento lógico da requisição. Após isso, `Logic` vai retornar uma lista de tuplas para representar medicamentos, que será então transformada em um objeto `Medicine` por `Manager`, ou três números representando as porcentagens da tarefa 3, e serão enviados para a `Interface` apresentar o resultado para o usuário.

Logic

A classe `Logic` é responsável pela lógica das tarefas do programa. É ela quem se comunica com o banco de dados (`DBConnect`) e o arquivo csv (`CSVReader`) e não `Manager`, adicionando uma camada a mais de segurança entre o usuário final e os dados reais no banco de dados, evitando possíveis invasões, além de distribuir melhor o acoplamento do programa.

A classe `Logic` tem uma série de funções responsáveis por se comunicar com o banco de dados e pedir requisições para ele e se comunicar com o arquivo CSV para povoar o banco de dados com cada linha do arquivo.

```
create_database()
entrada: nenhuma
saida: nenhuma
objetivo: responsável por criar o arquivo SQLite do banco de dados e a tabela Medicamentos, chamar csv_reader para ler o arquivo csv e salvar as linhas em memória e adicionar cada linha ao banco dados.
...
def create_database(self):
    self.base.open_connection()
    self.base.init_db()
    self.csv_reader.load_in_memory()
    for row in self.csv_reader.data:
        self.base.add_row(row)
    self.base.commit()
    self.base.close_connection()
```

Imagem 4: Função `create_database()` de `Logic`, responsável por preencher o banco de dados. Podemos ver que ela chama a função `load_in_memory()` de `csv_reader` para isso, e depois percorre os dados adicionando no banco de dados com a função `add_row()` de `base`.

Além disso, é o módulo `Logic` quem vai solicitar as consultas do banco, verificar se a consulta não encontrou resultados e informar `Manager`. Abaixo, algumas funções de `Logic` e seus objetivos:

```

...
consult_name():
    entrada: string que contém o nome a ser consultado
    saída: lista de tuplas que representa a consulta
    objetivo: chama base para fazer a consulta de medicamentos que completam o nome inserido.
...

def consult_name(self, name):
    self.base.open_connection()
    consult = self.base.select_by_name(name)
    self.base.close_connection()
    meds = []
    for medicine in consult:
        new_med = []
        new_med.append(medicine[0])
        new_med.append(medicine[8])
        new_med.append(medicine[9])
        new_med.append(medicine[13])
        new_med.append(medicine[23])
        meds.append(tuple(new_med))
    return meds

```

Imagem 5: Função consult_name() de Lógic responsável por fazer a consulta por nome

```

...

consult_code():
    entrada: string que contém o código a ser consultado
    saída: lista de tuplas que representa a consulta
    objetivo: chama base para fazer a consulta de medicamentos com o código inserido.
...

def consult_code(self, code):
    self.base.open_connection()
    consult = self.base.select_by_code(code)
    self.base.close_connection()
    print(consult)
    if(len(consult) == 0):
        return None
    else:
        meds = []
        new_med = []
        new_med.append(consult[0][0])
        new_med.append(consult[0][8])
        new_med.append(consult[0][9])
        new_med.append(float(consult[0][13].replace(',','.')))
        new_med.append(float(consult[0][23].replace(',','.')))
        meds.append(tuple(new_med))
        new_med = []
        new_med.append(consult[-1][0])
        new_med.append(consult[-1][8])
        new_med.append(consult[-1][9])
        new_med.append(float(consult[-1][13].replace(',','.')))
        new_med.append(float(consult[-1][23].replace(',','.')))
        meds.append(tuple(new_med))
    return meds

...

compare():
    entrada:
    saída: três inteiros que prepresentam as porcentagens
    objetivo: faz a comparação de avaliações PIS/COFINS
...

def compare(self):
    self.base.open_connection()
    neutral_counter = self.base.count_by_pis_cofins("Neutra")[0][0]
    positive_counter = self.base.count_by_pis_cofins("Positiva")[0][0]
    negative_counter = self.base.count_by_pis_cofins("Negativa")[0][0]
    self.base.close_connection()

    total = negative_counter + neutral_counter + positive_counter
    negative_percent = negative_counter*100/total
    positive_percent = positive_counter*100/total
    neutral_percent = neutral_counter*100/total
    return negative_percent, neutral_percent, positive_percent

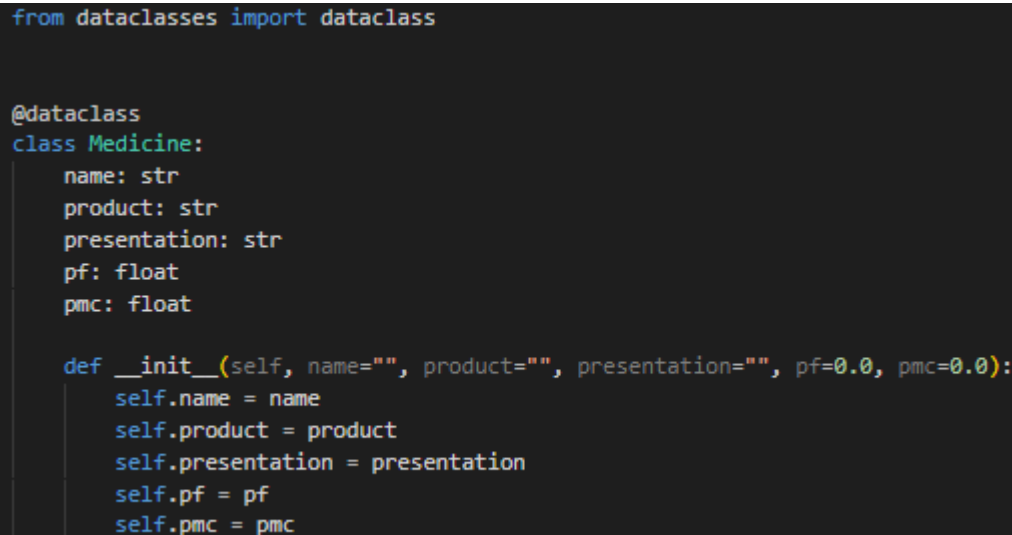
```

Imagem 6: Funções consult_code() e compare()

DBConnect, CSVReader e Medicine

Por último, temos os módulos que fazem parte do pacote data. Eles tem o objetivo de se comunicar com os dados persistentes do programa (DBConnect com o banco de dados e CSVReader com o arquivo csv) e armazenar os dados em memória de forma clara e de fácil compreensão (Medicine). Falaremos de cada um.

Medicine é uma dataclass, uma classe específica para armazenar dados e não fazer operações com eles. Implementei essa classe para armazenar os dados de forma clara para debugar o programa mais facilmente. Abaixo, uma imagem da classe com seus atributos:



```
from dataclasses import dataclass

@dataclass
class Medicine:
    name: str
    product: str
    presentation: str
    pf: float
    pmc: float

    def __init__(self, name="", product="", presentation="", pf=0.0, pmc=0.0):
        self.name = name
        self.product = product
        self.presentation = presentation
        self.pf = pf
        self.pmc = pmc
```

Imagem 7: Classe Medicine, ela armazena todos os 5 dados que serão necessários.

DBConnect é responsável por fazer a conexão com o banco de dados e solicitar que ele faça uma busca. O método `init_db()` é responsável por criar a tabela no banco, quando for solicitado por Logic e Manager que isso seja feito.

As funções `open_connection()`, `close_connection()` mudarão o estado do atributo `con`, que representa a conexão do banco, para aberto ou fechado. A função `commit()` tem o objetivo de fazer o commit das alterações no banco.

Por último, os métodos de seleção são chamados por Logic, dependente de qual tarefa foi solicitado por Manager, e receberão uma string nome ou ano ou um parâmetro de seleção.

Imagem 8: Funções de seleção de DBConnec

O módulo CSVReader é responsável por ler o arquivo csv e salvar as informações dele em memória. Ele é chamado por Logic apenas que Manager detecta que o banco de dados não existe ainda no sistema, para que todo o conteúdo do arquivo csv seja copiado para o banco de dados.

Imagem 9: A classe CSVReader e seus atributos e métodos

Testes

Para os testes dos módulos CSVReader e DBConnect, foram feitos testes de unidades utilizando a biblioteca unittest nativa do python. Além disso, foram feitos testes de integração do sistema inteira ou parte dele rodando o programa e vendo as saídas e erros que ele retornava.


```

'''
Classes responsáveis pelos testes de unidade de CSVReader
'''
class TestReaderMethods(unittest.TestCase):

    csv_reader = rd.CSVReader('TA_PRECO_MEDICAMENTO.csv')

    def test_absolute_name(self):
        self.assertEqual(self.csv_reader.absolute_name, answers.TEST_1_READER_ANSWER)

    def test_load_in_memory_first_element(self):
        index = 0
        attribute = 'CNPJ'
        self.csv_reader.load_in_memory()
        self.assertEqual(self.csv_reader.data[index][attribute], answers.TEST_2_READER_ANSWER)

    def test_load_in_memory_last_element(self):
        index = -1
        attribute = 'CNPJ'
        self.csv_reader.load_in_memory()
        self.assertEqual(self.csv_reader.data[index][attribute], answers.TEST_3_READER_ANSWER)

    def test_load_in_memory_intermediary_element(self):
        index = 10
        attribute = 'CNPJ'
        self.csv_reader.load_in_memory()
        self.assertEqual(self.csv_reader.data[index][attribute], answers.TEST_4_READER_ANSWER)

    def test_attribute_1(self):
        index = 10
        attribute = "SUBSTÂNCIA"
        self.csv_reader.load_in_memory()
        self.assertEqual(self.csv_reader.data[index][attribute], answers.TEST_5_READER_ANSWER)

    def test_attribute_2(self):
        index = 10
        attribute = "PRODUTO"
        self.csv_reader.load_in_memory()
        self.assertEqual(self.csv_reader.data[index][attribute], answers.TEST_6_READER_ANSWER)

```

Imagem 10: Testes de unidade da classe CSVReader

Projeto Final

Por fim, temos o programa inteiro dentro da pasta program, que é o pacote principal. Dentro dele estão os arquivos manager.py que implementa a classe Manager, logic.py que implementa a classe Logic, além do arquivo que implementa os testes de unidade (teste.py) e das respostas para cada teste (tests_answer.py). Dentro do pacote program, temos mais dois pacotes, data que abriga os arquivos que implementam as classes CSVReader (arquivo csvreader.py), DBConnect (database.py) e Medicine (medicine.py).

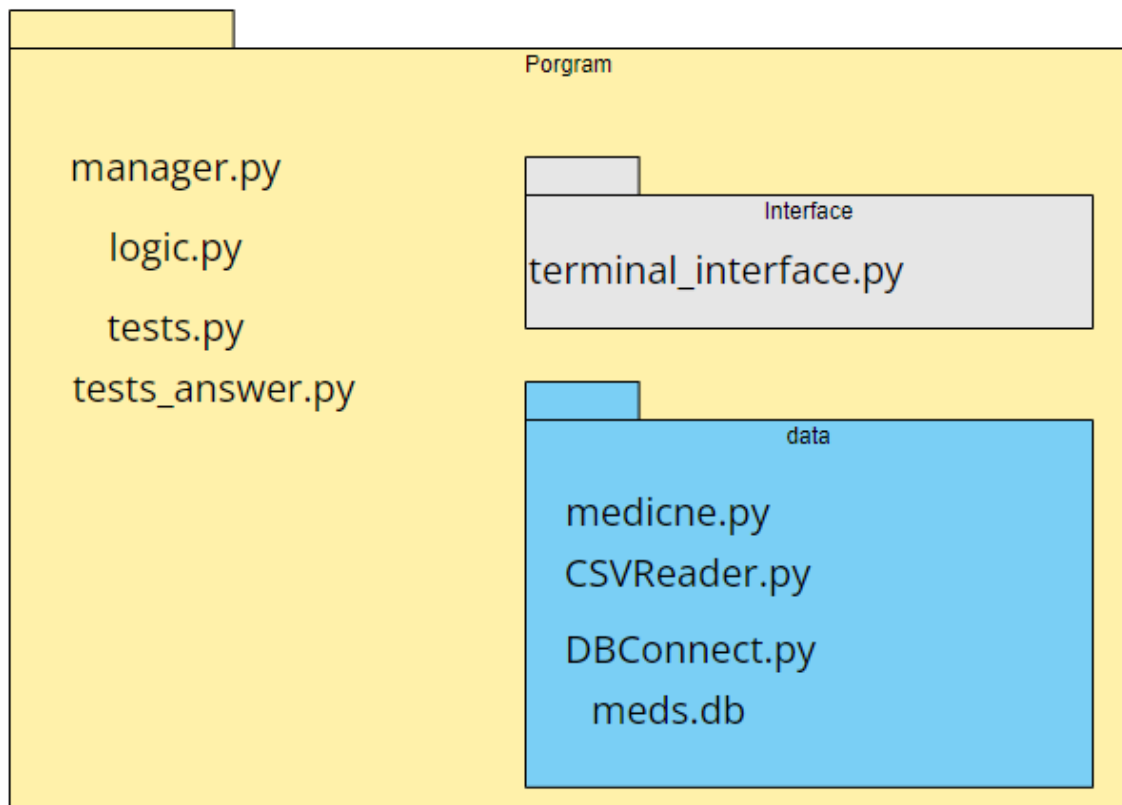


Imagem 11: Diagrama de pacotes do projeto

Resultados

Para a primeira tarefa, podemos ver que o programa encontrou os medicamentos Montelucaste e Montelucaste de sódio ao ser inserido o nome “monte”.

```

##### CONSULTA MEDICAMENTOS #####
Selecione uma das opções
1 - Consultar medicamento pelo nome
2 - Buscar pelo código de barras
3 - Comparativo PIS/COFINS
4 - Terminar o programa
Insira aqui: 1
Insira o nome: monte

##### RESULTADO DA PESQUISA #####
Nome do medicamento: MONTELUCASTE Produto: SINGULAIR Apresentação: 5 MG COM MAST CT 3 BL AL PLAS INC X 10 PF sem impost: 44,67
Nome do medicamento: MONTELUCASTE Produto: SINGULAIR Apresentação: 10 MG COM REV CT 3 BL AL PLAS INC X 10 PF sem impost: 44,67
Nome do medicamento: MONTELUCASTE Produto: SINGULAIR Apresentação: 4 MG COM MAST CT BL AL PLAS INC X 10 PF sem impost: 30,03
Nome do medicamento: MONTELUCASTE Produto: SINGULAIR Apresentação: 4 MG COM MAST CT 3 BL AL PLAS INC X 10 PF sem impost: 44,67
Nome do medicamento: MONTELUCASTE Produto: SINGULAIR Apresentação: 5 MG COM MAST CT BL AL PLAS INC X 10 PF sem impost: 30,03
Nome do medicamento: MONTELUCASTE Produto: SINGULAIR Apresentação: 10 MG COM REV CT BL AL PLAS INC X 10 PF sem impost: 30,03
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: PIEMONTE Apresentação: 4 MG COM MAST CT BL AL/AL X 10 PF sem impost: 15,70
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: PIEMONTE Apresentação: 4 MG COM MAST CT BL AL/AL X 30 PF sem impost: 47,07
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: PIEMONTE Apresentação: 5 MG COM MAST CT BL AL/AL X 30 PF sem impost: 54,13
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: PIEMONTE Apresentação: 4 MG COM MAST CT BL AL/AL X 60 PF sem impost: 58,67
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: PIEMONTE Apresentação: 5 MG COM MAST CT BL AL/AL X 60 PF sem impost: 58,67
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: PIEMONTE Apresentação: 10 MG COM REV CT BL AL/AL X 30 PF sem impost: 41,51
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: PIEMONTE Apresentação: 10 MG COM REV CT BL AL/AL X 60 PF sem impost: 58,67
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 4 MG COM MAST CT BL AL/AL X 30 PF sem impost: 58,64
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 5 MG COM MAST CT BL AL/AL X 30 PF sem impost: 58,64
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 4 MG COM MAST CT BL AL/AL X 60 PF sem impost: 70,10
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 5 MG COM MAST CT BL AL/AL X 60 PF sem impost: 79,99
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 10 MG COM REV CT BL AL/AL X 30 PF sem impost: 58,50
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 10 MG COM REV CT BL AL/AL X 60 PF sem impost: 87,50
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: PIEMONTE Apresentação: 4 MG GRAN SOL OR CT 30 ENV PLAS PES/AL/PLAS PEBD PF sem impost: 49,04
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: VIATINE Apresentação: 10 MG COM REV CT BL AL AL X 10 PF sem impost: 19,10
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: VIATINE Apresentação: 10 MG COM REV CT BL AL AL X 30 PF sem impost: 57,53
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: VIATINE Apresentação: 5 MG COM MAST CT BL AL AL X 10 PF sem impost: 22,28
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: VIATINE Apresentação: 5 MG COM MAST CT BL AL AL X 30 PF sem impost: 66,78
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: VIATINE Apresentação: 4 MG COM MAST CT BL AL AL X 10 PF sem impost: 21,51
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: VIATINE Apresentação: 4 MG COM MAST CT BL AL AL X 30 PF sem impost: 64,46
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 10 MG COM REV CT BL AL/AL X 30 PF sem impost: 58,58
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELAIR Apresentação: 10 MG COM REV CT BL AL/AL X 30 PF sem impost: 140,60
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELAIR Apresentação: 4 MG GRAN CT 30 SACH X 350 MG PF sem impost: 140,94
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELAIR Apresentação: 4 MG COM MAST CT BL AL AL X 10 PF sem impost: 16,56
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELAIR Apresentação: 4 MG COM MAST CT BL AL AL X 30 PF sem impost: 49,60
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELAIR Apresentação: 5 MG COM MAST CT BL AL AL X 10 PF sem impost: 20,71
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELAIR Apresentação: 5 MG COM MAST CT BL AL AL X 30 PF sem impost: 62,10
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELAIR Apresentação: 4 MG COM MAST CT BL AL AL X 60 PF sem impost: 124,78
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELAIR Apresentação: 5 MG COM MAST CT BL AL AL X 60 PF sem impost: 120,70
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELAIR Apresentação: 4 MG GRAN CT 60 SACH X 350 MG PF sem impost: 196,74
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELAIR Apresentação: 10 MG COM REV CT BL AL/AL X 60 PF sem impost: 180,30
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 10 MG COM REV CT BL AL/AL X 10 PF sem impost: 25,02
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 10 MG COM REV CT BL AL/AL X 30 PF sem impost: 58,65
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 4 MG COM MAST CT BL AL AL X 30 PF sem impost: 75,11
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 5 MG COM MAST CT BL AL AL X 30 PF sem impost: 58,61
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 4 MG COM MAST CT BL AL/AL X 30 PF sem impost: 27,84
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 5 MG COM MAST CT BL AL/AL X 30 PF sem impost: 23,43
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 10 MG COM REV CT BL AL/AL X 30 PF sem impost: 27,84
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: LEVOLUKAST Apresentação: 10MG + 5MG COM REV CT FR PLAS OPC X 7 PF sem impost: 31,75
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: LEVOLUKAST Apresentação: 10MG + 5MG COM REV CT FR PLAS OPC X 14 PF sem impost: 63,51
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: UNIAIR Apresentação: 4 MG COM MAST CT BL AL/AL X 30 PF sem impost: 58,40
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: UNIAIR Apresentação: 5 MG COM MAST CT BL AL/AL X 30 PF sem impost: 37,61
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: UNIAIR Apresentação: 10 MG COM REV CT BL AL/AL X 30 PF sem impost: 90,17
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 4 MG COM MAST CT BL AL/AL X 30 PF sem impost: 91,63
Nome do medicamento: MONTELUCASTE DE SÓDIO Produto: MONTELUCASTE DE SÓDIO Apresentação: 5 MG COM MAST CT BL AL/AL X 30 PF sem impost: 58,64

```

Imagem 12: Testes da primeira tarefa

Para o segundo teste, testamos o código 7891317421618, o único código que encontrei no csv que pertence a dois remédios. Podemos ver que ele encontrou os dois remédios e os imprimiu, com a diferença de valores que é zero, já que os dois tem o mesmo valor.

```

##### CONSULTA MEDICAMENTOS #####
Selecione uma das opções
1 - Consultar medicamento pelo nome
2 - Buscar pelo código de barras
3 - Comparativo PIS/COFINS
4 - Terminar o programa
Insira aqui: 2
Insira o código de barras: 7891317421618

##### RESULTADO DA PESQUISA #####
Nome do medicamento com maior PMC: CLORIDRATO DE PAROXETINAPMC: 194.59
Nome do medicamento com menor PMC: CLORIDRATO DE PAROXETINAPMC: 194.59
Diferença entre os dois PMCs: 0.0

```

Imagem 13: Testes da segunda tarefa

Para a terceira tarefa, os resultados foram 30,2% de avaliações negativas, 69,3% de avaliações positivas e 0,3% de neutras. Podemos ver que nem um asterisco foi impresso para neutras pois ela é menor que 1%.

```
##### CONSULTA MEDICAMENTOS #####
Selecione uma das opções
1 - Consultar medicamento pelo nome
2 - Buscar pelo código de barras
3 - Comparativo PIS/COFINS
4 - Terminar o programa
Insira aqui: 3
-
##### RESULTADO DA PESQUISA #####
CLASSIFICACAO  PERCENTUAL  TGRAFICO
Negativa       30.26605853287723 *****
Neutra         0.33447358418852147
Positiva       69.39946788293425 *****

##### CONSULTA MEDICAMENTOS #####
Selecione uma das opções
1 - Consultar medicamento pelo nome
2 - Buscar pelo código de barras
3 - Comparativo PIS/COFINS
4 - Terminar o programa
Insira aqui: -
```

Imagem 14: Resultados encontrados pelo programa

Conclusões e Autoavaliação

Portanto, o programa parece estar bem robusto e funcional. Aparentemente ele consegue resolver todas as 3 tarefas e não foi encontrado nenhum bug na execução dos testes.

Fiquei bem satisfeito com meu resultado, considerando que sou novato com SQLite e testes de unidade para python, consegui ganhar bastante conhecimento nestas duas áreas. Consegui aplicar vários conceitos que aprendi e cadeiras relacionadas a engenharia de software como modularidade e arquitetura em camadas, que é uma área que tenho bastante interesse.

Por fim, gostaria só de ter implementado uma interface gráfica para fazer a comunicação com o usuário, mas infelizmente não deu tempo.