UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE ESTATÍSTICA E MATEMÁTICA APLICADA

CURSO DE GRADUAÇÃO EM CIÊNCIA DE DADOS

DAVI DA SILVA ARAUJO

EMANUEL VICTOR DA SILVA COSTA

FRANCISCO LEMUEL FERREIRA LIMA

GABRIEL MOREIRA CAVALCANTE

PAULO GUSTAVO DUARTE DA COSTA

MYTWITTER

FORTALEZA

2025

RESUMO

MyTwitter é uma aplicação isnpirada na rede social Twitter, hoje chamada de X, na qual é possível criar usuários, menssagens - tweets - seguir outros usuários e visualizar os prórprios tweets e quem se segue - timeline.

Palavras-chave: CienciaDeDados. Python. MyTwitter. TecnicasdeProgramacao.

SUMÁRIO

1 MAPEAMENTO DOS RESPONSÁVEIS PELA IMPLEMENTAÇÃO DE CADA FUNCIONALIDADE

Esta é a divisão das atriuições de cada participante do grupo.

1.1 Davi da Silva Araújo

- Perfil.get_timeline()
- Perfil.set_usuario()
- Perfil.get_usuario()
- Perfil.set_ativo()
- Perfil.is_ativo()
- PessoaFisica.get_cpf()
- PessoaJuridica.get_cnpj()
- RepositorioUsuarios.cadastrar()
- RepositorioUsuarios.buscar()

1.2 Emanuel Victor da Silva Costa

- RepositorioUsuarios.atualizar()
- MyTwitter.criar_perfil()
- MyTwitter.cancelar_perfil()
- MyTwitter.tweetar()
- MyTwitter.timeline()
- MyTwiter.seguir()
- MyTwitter.seguidores()
- MyTwitter.seguidos()

1.3 Francisco Lemuel Ferreira Lima

- Relatório
- Vídeo

1.4 Gabriel Moreira Cavalcante

- Tweet.get_id()
- Tweeet.get_usuario()
- Tweet.get_mensagem()
- Tweet.get_data_postagem()
- Perfil.add_seguidor()
- Perfil.add_seguidos()
- Perfil.add_tweet()
- Perfil.get_tweets()
- Perfil.get_tweet()

1.5 Paulo Gustavo Duarte da Costa

- Terminal

2 ARQUIVOS PRINCIPAIS

2.1 Main

Main é o arquivo que contém as classes Tweet, Perfis, Repositório de usuários e MyTwitter. O arquivo inicia chamando a biblioteca datetime, necessária para criação de tweets, e as exceções.

Código-fonte 1 – Main chamada

```
1 from datetime import datetime
2 from exception import PEException, PIException, PDException
    , MFPException, SIException, UJCException, UNCException
```

2.1.1 Classe Tweet

Código-fonte 2 – Tweet

```
1  class Tweet:
2      def __init__(self, usuario, mensagem, gerador_id):
3          self.__id = next(gerador_id)
4          self.__usuario = usuario
5          self.__mensagem = mensagem
6          self.__data_postagem = datetime.now().replace(
               second=0, microsecond=0)
7
8      def get_id(self):
9          return self.__id
10
11     def get_usuario(self):
12         return self.__usuario
13
14     def get_mensagem(self):
15         return self.__mensagem
```

```
16
17      def get_data_postagem(self):
18          return self.__data_postagem
```

A classe Tweet é responsável por criar os tweets, recebe três parâmetros e possui os seguintes atributos:

- **__id**: id único para cada mensagem criado por um gerador, linha 3.
- **__usuario**: Nome do usuário, linha 4.
- **__mensagem**: O tweet de até 140 caracteres, linha 5.
- **__data_postagem**: Recebe a data e hora de criação da mensagem, linha 6.

Esta classe possui os seguintes métodos:

- **get_id**: Retorna o id da mensagem, linha 9.
- **get_usuario**: Retorna o nome de usuário, linha 10.
- **get_mensagem**: Retorna a mensagem, linha 15.
- **get_data_postagem**: Retorna a data da postagem, linha 18.

2.1.2 Classes de Perfis

A principal classe é a Perfil da qual as outras classes de perfis herdam seus atributos e métodos.

2.1.2.1 Classe Perfil

Código-fonte 3 – Perfil

```
1  class Perfil():
2      def __init__(self, usuario):
3          self.__usuario = usuario
4          self.__seguidos = []
5          self.__seguidores = []
6          self.__tweets = []
7          self.__ativo = True
8
9      def add_seguidor(self, perfil):
```

```
10          if perfil not in self.__seguidores:
11              self.__seguidores.append(perfil)
12
13      def add_seguidos(self, perfil):
14          if perfil not in self.__seguidos:
15              self.__seguidos.append(perfil)
16
17      def add_tweet(self, tweet):
18          self.__tweets.append(tweet)
19          self.__tweets.sort(key=lambda t: t.
              get_data_postagem())
20
21      def get_tweets(self):
22          return sorted(self.__tweets, key=lambda t: t.
              get_data_postagem())
23
24      def get_tweet(self, tweet_id):
25          for tweet in self.__tweets:
26              if tweet.get_id() == tweet_id:
27                  return tweet
28          return None
29
30      def get_timeline(self):
31          timeline = self.__tweets[:]
32          for perfil in self.__seguidos:
33              timeline.extend(perfil.__tweets)
34          timeline.sort(key=lambda tweet: tweet.
              get_data_postagem())
35          return timeline
36
37      def set_usuario(self, usuario):
38          self.__usuario = usuario
```

```
39
40      def get_usuario(self):
41          return self.__usuario
42
43      def set_ativo(self, ativo):
44          self.__ativo = ativo
45
46      def is_ativo(self):
47          return self.__ativo
48
49      def get_seguidores(self):
50          return self.__seguidores
51
52      def get_seguidos(self):
53          return self.__seguidos
54
55    def seguir(self, perfil):
56          self.add_seguidos(perfil)
57          perfil.add_seguidor(self)
```

A classe Perfil possui cinco atributos:

- **__usuario**: Nome do dono do perfil, linha 3.
- **__seguidos**: Lista inicialmente vazia onde são armazenados todos os perfis os quais seguem o usuário, linha 4.
- **__seguidores**: Lista inicialmente vazia onde são armazenados os perfis que o usuário segue, linha 5.
- **__twees**: Lista inicialmente vazia a qual são armazenados os tweets do usuário, linha 6.
- **__ativo**: Inicialmente True, indica se o perfil está ativo, linhas 7.

Os métodos são:

- **add_seguidor**: Recebe um perfil, testa se faz parte da lista de seguidores, caso contrário, o adiciona à lista, linhas 9-11.
- **add_seguidos**: Recebe um perfil, testa se faz parte da lista de seguidos, caso contrário, o

adiciona à lista, linhas 13-15.

- **add_tweets**: Retorna a lista de tweets do usuário em ordem de postagem, linha 22.
- **get_tweet**: Recebe o id de um tweet - linha 24 -, busca na lista de tweets - linhas 25-26 -, caso positivo, retorna a mensaggem, linha 27. Caso contrário retona None, linha 28.
- **get_timeline**: Toma todos os tweets do usuário, linha 31, busca os tweets dos perfis seguidos, linha 32-33, arranja o resultado por data de publicação, linha 34 e retorna a lista, linha 35.
- **set_usuario**: Recebe o nome de usuário e o atribui a __usuario, linha 37-38.
- **get_usuario**: Retorna o nome de usuário, linha 41.
- **set_ativo**: recebe um valor, True ou False, e o atribui a __ativo, linha 43-44.
- **is_ativo**: Retorna o valor do atributo __ativo, linha 47.
- **get_seguidores**: Retorna a lista de seguidores, linha 50.
- **get_seguidos**: Retorna a lista de seguidos, linha 53.
- **seguir**: Segue outro usuário. Recebe um perfil, linha 55, adiciona perfil a lista de seguidos, linha 56 e adiciona o usuário a lista de seguidores do perfil seguido, linha 57.

2.1.2.2 Classe Pessoa Física

Esta classe herda da classe Perfil, linhas 1-3, adiciona um novo atributo, **__cpf** linha 4, e método adicional **get_cpf** que retorna o valor do atributo **__cpf**.

Código-fonte 4 – PessoaFisica

```
1 class PessoaFisica(Perfil):
2     def __init__(self, usuario, cpf):
3         super().__init__(usuario)
4         self.__cpf = cpf
5 
6     def get_cpf(self):
7         return self.__cpf
```

2.1.2.3 Classe Pessoa Jurídica

Esta classe herda da classe Perfil, linhas 1-3, adiciona um novo atributo, **__cnpj** linha 4, e método adicional **get_cnpj** que retorna o valor do atributo **__cnpj**.

Código-fonte 5 – PessoaJuridica

```
1 class PessoaJuridica(Perfil):
2     def __init__(self, usuario, cnpj):
3         super().__init__(usuario)
4         self.__cnpj = cnpj
5 
6     def get_cnpj(self):
7         return self.__cnpj
```

2.1.3 Classe Repositório de Usuários

A classe repositório de usuário gerencia todos os perfis salvos. Há um único atributo: **__usuarios** que é a lista, iniciamente vazia, com todos os usuários cadastrados.

Código-fonte 6 – RepositorioUsuario

```
1  class RepositorioUsuarios():
2      def __init__(self):
3          self.__usuarios = []
4 
5      def cadastrar(self, perfil):
6          for usuario in self.__usuarios:
7              if usuario.get_usuario() == perfil.get_usuario
                   ():
8                  raise UJCException('Usuario ja existe')
9          self.__usuarios.append(perfil)
10 
11     def buscar(self, nome_usuario):
12         for usuario in self.__usuarios:
```

```
13              if usuario.get_usuario() == nome_usuario:
14                  return usuario
15          return None
16
17      def atualizar(self, perfil):
18          for i, usuario in enumerate(self.__usuarios):
19              if usuario.get_usuario() == perfil.get_usuario
                    ():
20                  self.__usuarios[i] = perfil
21                  return
22          raise UNCException("Usuario nao cadastrado")
```

Os métodos dessa classe são os seguintes:

- **cadastrar**: recebe um perfil, linha 5, busca esse perfil em **__usuarios**, linha 6-7, caso positivo levanta uma exceção Usuário Já Cadastrado, linha 8, caso falso adiciona o perfil a lista de usuários, linha 9.
- **atualizar**: reecebe um perfil, linha 17, busca o perfil dentre os usuários cadastrados, linhas 18-19, caso positivo muda o perfil cadastrado pelo novo, linha 20, caso contrário sobe uma exceção usuário não cadastrado, linha 22.

2.1.4 Classe MyTwitter

A classe MyTwitter gerencia as funções da plataforma.

Código-fonte 7 – Classe MyTwitter

```
1  class MyTwitter:
2      def __init__(self):
3          self.__repositorio = RepositorioUsuarios()
4          self.__gerador_id = self.__gerador_id_func()
5
6      def __gerador_id_func(self):
7          id = 1
8          while True:
```

```
9               yield id
10              id += 1
11
12      def verificar_cpf_cnpj_existente(self, cpf_cnpj):
13          for usuario in self.__repositorio.
                _RepositorioUsuarios__usuarios:
14              if isinstance(usuario, PessoaFisica) and
                    usuario.get_cpf() == cpf_cnpj:
15                  return True
16              elif isinstance(usuario, PessoaJuridica) and
                    usuario.get_cnpj() == cpf_cnpj:
17                  return True
18          return False
19
20      def criar_perfil(self, perfil):
21          if self.__repositorio.buscar(perfil.get_usuario())
                is not None:
22              raise PEException("Perfil ja existe")
23          self.__repositorio.cadastrar(perfil)
24
25      def tweetar(self, usuario, mensagem):
26          perfil = self.__repositorio.buscar(usuario)
27          if perfil is None:
28              raise PIException("Perfil inexistente")
29          if not perfil.is_ativo():
30              raise PDException("Perfil desativado")
31          if not (1 <= len(mensagem) <= 140):
32              raise MFPException("Mensagem deve ter entre 1 e
                    140 caracteres")
33
34          tweet = Tweet(usuario, mensagem, self.__gerador_id)
35          perfil.add_tweet(tweet)
```

```
36
37      def timeline(self, usuario):
38          perfil = self.__repositorio.buscar(usuario)
39          if perfil is None:
40              raise PIException("Perfil inexistente")
41          if not perfil.is_ativo():
42              raise PDException("Perfil desativado")
43          return perfil.get_timeline()
44
45      def tweets(self, usuario):
46          perfil = self.__repositorio.buscar(usuario)
47          if perfil is None:
48              raise PIException("Perfil inexistente")
49          if not perfil.is_ativo():
50              raise PDException("Perfil desativado")
51          return perfil.get_tweets()
52
53      def seguir(self, seguidor, seguido):
54          perfil_seguidor = self.__repositorio.buscar(
              seguidor)
55          perfil_seguido = self.__repositorio.buscar(seguido)
56          if perfil_seguidor is None or perfil_seguido is
              None:
57              raise PIException("Perfil inexistente")
58          if not perfil_seguidor.is_ativo() or not
              perfil_seguido.is_ativo():
59              raise PDException("Um dos perfis esta
                  desativado")
60          if seguidor == seguido:
61              raise SIException("Um usuario nao pode seguir a
                   si mesmo")
62          perfil_seguidor.add_seguidos(perfil_seguido)
```

```
63          perfil_seguido.add_seguidor(perfil_seguidor)
64
65      def numero_seguidores(self, usuario):
66          perfil = self.__repositorio.buscar(usuario)
67          if perfil is None:
68              raise PIException("Perfil inexistente")
69          if not perfil.is_ativo():
70              raise PDException("Perfil desativado")
71          return len(perfil.get_seguidores())
72
73      def seguidores(self, usuario):
74          perfil = self.__repositorio.buscar(usuario)
75          if perfil is None:
76              raise PIException("Perfil inexistente")
77          if not perfil.is_ativo():
78              raise PDException("Perfil desativado")
79          return perfil.get_seguidores()
80
81      def seguidos(self, usuario):
82          perfil = self.__repositorio.buscar(usuario)
83          if perfil is None:
84              raise PIException("Perfil inexistente")
85          if not perfil.is_ativo():
86              raise PDException("Perfil desativado")
87          return perfil.get_seguidos()
88
89      def cancelar_perfil(self, usuario):
90          perfil = self.__repositorio.buscar(usuario)
91          if perfil is None:
92              raise PIException("Perfil inexistente")
93          if not perfil.is_ativo():
94              raise PDException("Perfil ja esta desativado")
```

```
95          perfil.set_ativo(False)
```

Atributos

- **__repositorio**: Instancia um objeto da classe RepositorioUsuarios, linha 3.
- **__gerador_id**: Armazena o gerador de ids para as mensagens __id_generator, linha 4.

Métodos

- **__gerador_id_func**: Gera ids únicos para as mensagens. Inicia de 1, linha 7, toda vez que é requisitado novo id retorna o valor armazenado em id_atual e o incrementa em 1, linhas 9-10.
- **verificar_cpf_cnpj_existentes**: Verifica se já existe um usuário cadastrado com determinado cpf ou cnpj. Recebe um perfil, linha 12, faz uma busca dentre os usuários existentes, linha 13, caso pessoa física checa o cpf, linha 14, caso jurídica checa o cnpj, inha 16. Caso verdadeiro retorna True, linhas 15 e 17, caso falso retorna False, linha 18.
- **criar_perfil**: Cadastra novo usuário. Recebe um perfil, linha 20, busca o perfil nos já cadastrados, linha 21, caso positivo levanta a exceção de Perfil existente, linha 22, caso negativo adiciona o perfil ao repositório, linha 23.
- **tweetar**: Cria novo tweet. Faz uma usca pelo usuário, linha 26, caso negativa levanta a exceção Perfil inexistente, linha 27-28. Caso o perfil esteja desativado levanta a exceção de Perfil desativado, linha 29-30. Caso a mensagem tenha mais de 140 caracteres, levanta a exceção Mensagem fora do padrão, linhas 31-32. Por fim, instancia uma nova mensagem, linha 34, e adiciona a perfil do usuário, linha 35.
- **timeline**: Cria uma linha do tempo das mensagens do usuário. Busca o perfil, levanando as exceções de inexistente ou desativado quando for o caso, linhas 38-42. Retorna o timeline do perfil, linha 43.
- **tweets**: Busca os tweets do perfil. Busca o perfil levantando exceções quando necessário, linha s 46-30. Retorna as mensagens do perfil, linha 51.
- **seguir**: Faz um perfil seguir outro. Recebe um perfil seguidor, e outro seguido, linha 53. Busca se os dois perfis existem e estão ativos, linhas 54-59. Testa se o perfi seguidor e perfil seguido são os mesmos, caso positivo sobe a exceção Seguidor inválido, linhas 60-61. Caso contrário, adiciona o perfil seguido à lista de seguidos do seguidor, linha 62, e o seguidor à lista de seguidores do seguido, linha 63.
- **numero_seguidores**: Conta o número de seguidores de um perfil. Recebe o perfil e busca

se existe e está ativo, liinhas 65-70. Retorna o tamanho da lista de seguidores, linha 71.

- **seguidores**: Retorna a lista de seguidores. Recebe um perfil e busca se o usuário está cadastrado e ativo, linhas 74-78. Retorna os seguidores do pperfil, linha 79.
- **seguidos**: Retorna a lista de seguidos. Recebe um perfil e busca se o usuário está cadastrado e ativo, linhas 82-86. Retorna os seguidores do pperfil, linha 87.
- **cancelar_perfil**: Desativa um perfil, garantindo que ele exista e esteja ativo. Recebe um perfil, linha 89, realiza uma busca se o perfi existe e está ativo, linhas 90-94. Muda o status de ativo para False, linha 95.

2.2 Exception

Código-fonte 8 – Exception

```
1  class PEException(Exception):
2      pass
3
4  class PIException(Exception):
5      pass
6
7  class PDException(Exception):
8      pass
9
10 class MFPException(Exception):
11     pass
12
13 class SIException(Exception):
14     pass
15
16 class UJCException(Exception):
17     pass
18
19 class UNCException(Exception):
20     pass
```

- **PEException**: Perfil Existente
- **PIException**: Perfil Inexistente
- **PDException**: Perfil Desativado
- **MFPException**: Mensagem Fora do Padrão
- **SIException**: Seguidor Inválido
- **UJCException**: Usuário Já Cadastrado
- **UNCException**: Usuário Não Cadastrado

2.3 Terminal

O Terminal é a interface que conecta o usuário ao sistema MyTwitter. O arquivo importa a biblioteca datetime para marcar as datas de cada mensagem, linha 2, o arquivo main que contém as classes do MyTwitter, linha 3, e o arquivo com as classes de exceções, linha 1.

Código-fonte 9 – Terminal

```
1  from exception import *
2  from datetime import datetime
3  from main import *
4
5  class TerminalTwitter:
6      def __init__(self, my_twitter):
7          self.my_twitter = my_twitter
8
9      def menu(self):
10         while True:
11             print("\nMyTwitter\n")
12             print("0. Meu Twitter")
13             print("1. Criar perfil")
14             print("2. Sair")
15             opcao = input("Escolha uma opcao: ")
16
17             if opcao == "0":
18                 self.meu_twitter()
```

```
19              elif opcao == "1":
20                  self.criar_perfil()
21              elif opcao == "2":
22                  print("Saindo...")
23                  break
24              else:
25                  print("Opcao invalida! Tente novamente.")
26
27      def meu_twitter(self):
28          usuario = input("Digite seu usuario: ")
29
30          perfil = self.my_twitter._MyTwitter__repositorio.
               buscar(usuario)
31          if perfil is None:
32              print("Erro: Usuario nao encontrado! Crie um
                   perfil primeiro.")
33              return
34
35          try:
36              while True:
37                  print(f"\nMeu Twitter: {usuario}")
38                  print("1. Tweetar")
39                  print("2. Ver timeline")
40                  print("3. Ver seguidores")
41                  print("4. Ver seguidos")
42                  print("5. Seguir usuario")
43                  print("6. Voltar")
44
45                  opcao = input("Escolha uma opcao: ")
46
47                  if opcao == "1":
48                      self.tweetar(usuario)
```

```
49                elif opcao == "2":
50                    self.ver_timeline(usuario)
51                elif opcao == "3":
52                    self.ver_seguidores(usuario)
53                elif opcao == "4":
54                    self.ver_seguidos(usuario)
55                elif opcao == "5":
56                    self.seguir_usuario(usuario)
57                elif opcao == "6":
58                    break
59                else:
60                    print("Opcao invalida! Tente novamente.
                          ")
61
62            except PIException:
63                print("Erro: Perfil inexistente!")
64            except PDException:
65                print("Erro: Perfil desativado!")
66
67        def criar_perfil(self):
68            usuario = input("Digite o nome do usuario: ")
69            tipo = input("Tipo de perfil (1 - Pessoa Fisica, 2
                  - Pessoa Juridica): ")
70            try:
71                if tipo == "1":
72                    cpf = input("Digite o CPF: ")
73                    if self.my_twitter.
                          verificar_cpf_cnpj_existente(cpf):
74                        print("Erro: Ja cadastrado!")
75                        return
76                    perfil = PessoaFisica(usuario, cpf)
77                elif tipo == "2":
```

```
78                  cnpj = input("Digite o CNPJ: ")
79                  if self.my_twitter.
                       verificar_cpf_cnpj_existente(cnpj):
80                      print("Erro: Ja cadastrado!")
81                      return
82                  perfil = PessoaJuridica(usuario, cnpj)
83              else:
84                  print("Opcao invalida!")
85                  return
86
87              self.my_twitter.criar_perfil(perfil)
88              print("Perfil criado com sucesso!")
89          except PEException:
90              print("Erro: Perfil ja existe!")
91
92      def tweetar(self, usuario):
93          mensagem = input("Digite sua mensagem (max. 140
                caracteres): ")
94          try:
95              self.my_twitter.tweetar(usuario, mensagem)
96              print("Tweet enviado com sucesso!")
97          except MFPException:
98              print("Erro: A mensagem deve ter entre 1 e 140
                    caracteres!")
99          except PIException:
100             print("Erro: Perfil inexistente!")
101         except PDException:
102             print("Erro: Perfil desativado!")
103
104     def ver_seguidores(self, usuario):
105         try:
106             seguidores = self.my_twitter.seguidores(usuario
```

```
            )
            if not seguidores:
                print("Voce ainda nao tem seguidores.")
            else:
                print("\nSeus seguidores:")
                for seguidor in seguidores:
                    print(f"- {seguidor.get_usuario()}")
        except PIException:
            print("Erro: Perfil inexistente!")
        except PDException:
            print("Erro: Perfil desativado!")

    def ver_seguidos(self, usuario):
        try:
            seguidos = self.my_twitter.seguidos(usuario)
            if not seguidos:
                print("Voce ainda nao segue ninguem.")
            else:
                print("\nVoce segue:")
                for seguido in seguidos:
                    print(f"- {seguido.get_usuario()}")
        except PIException:
            print("Erro: Perfil inexistente!")
        except PDException:
            print("Erro: Perfil desativado!")

    def seguir_usuario(self, usuario):
        seguido = input("Digite o nome do usuario que
           deseja seguir: ")
        try:
            self.my_twitter.seguir(usuario, seguido)
            print(f"Agora voce esta seguindo {seguido}!")
```

```
137             except PIException:
138                 print("Erro: Um dos perfis nao existe!")
139             except PDException:
140                 print("Erro: Um dos perfis esta desativado!")
141             except SIException:
142                 print("Erro: Voce nao pode seguir a si mesmo!")
143 
144         def ver_timeline(self, usuario):
145             try:
146                 timeline = self.my_twitter.timeline(usuario)
147                 if not timeline:
148                     print("Sua timeline esta vazia!")
149                 else:
150                     print("\nTimeline:")
151                     for tweet in timeline:
152                         print(f"{tweet.get_usuario()} ({tweet.
                                get_data_postagem()}): {tweet.
                                get_mensagem()}")
153             except PIException:
154                 print("Erro: Perfil inexistente!")
155             except PDException:
156                 print("Erro: Perfil desativado!")
157 
158 if __name__ == "__main__":
159     from main import MyTwitter
160     my_twitter = MyTwitter()
161     terminal = TerminalTwitter(my_twitter)
162     terminal.menu()
```

A classe Terminal Twitter instancia um objeto da classe MyTwitter e o coloca como aributo em my_twitter, linhas 67. Os métodos são os seguintes:

Pertencentes ao menu

- **menu**: O primeiro menu que o usuário tem acesso, apresenta 3 opções, linhas 12-14, recebe a resposta do usuário, linha 15. Caso seja 0, dá acesso ao meu_twitter, linha 18. Caso 1, acesso à criação de nova conta, linha 20. Caso 2, sai do sistema, linha 22-23. Caso o usuário digite outro valor, aparece uma mensagem de opção inválida e retorna novamene ao menu, linha 25.
- **meu_twitter**: Este é o menu que o usuário tem para realizar as movimentações da sua conta. Primeiro é necessário fornercer um nome de usuário existente, linhas 30-35. Em seguida há o menu de opções do que o usuário deve escolher uma opção válida, linhas 35-65. Cada ação realizada pelo usuário retorna novamente a este menu até ser escolhida a opção 6, linha 58.
- **criar_perfil**: Primeira ação para novos usuários. Recebe o nome de usuário e um tipo de perfil, linhas 68-69. Recebe um número de cpf ou cnpj, linhas 72 e 78, e checa se não há um cadastro já existente 73 e 79. Caso negativo cria um perfil, linhas 76 e 82, e adiciona ao MyTwitter, linha 87-88. Caso posisivo manda um aviso de conta já existente, linhas 89-90.

Pertencentes a meu twitter

- **tweetar**: Cria tweets. Recebe uma mensagem, linha 93. Se for menor que 140 caracteres, é adicionada, linha 95. Caso contrário, levanta uma mensagem de erro, linhas 97-98.
- **ver_seguidores**: Acessa a lista de seguidores do perfil, linhas 106 e 110-112. Caso a lista esteja vazia, linha 108.
- **ver_seguidos**: Acessa a lista dos perfis seguidos, linhas 120 e 124-126. Caso vazia, linha 122.
- **seguir_usuario**: Permite seguir outro perfil. Recebe o nome de um usuário, linha 133. Caso o usuário exista e esteja ativo, realiza a operação, linhas 135-136. Caso contrário, linhas 137-142.
- **ver_timeline**: Acessa o histórico de tweets do usuário, linha 146, e seus seguidos, linhas 150-152. Caso vazio, linha 148.

3 MAIN TEST

3.1 Ambiente

Importou a biblioteca unitest para relaizar os testes. O arquivo main e a biblioteca datetime. Também foram criados perfis e tweets.

Código-fonte 10 – Ambiente

```
1  import unittest
2  from main import *
3  from datetime import datetime
4
5  def gerador_id():
6      yield 1
7
8  perfil1 = Perfil("usuario1")
9  perfil2 = Perfil("usuario2")
10 perfil3 = Perfil("usuario3")
11
12 tweet1 = Tweet("usuario1", "mensagem1", gerador_id())
13 tweet2 = Tweet("usuario1", "mensagem2", gerador_id())
```

3.2 Classe TesteTeet

A classe TestTweet cria s testes para cada método da classe Tweet.

Código-fonte 11 – TesteTweet

```
1  class TesteTweet(unittest.TestCase):
2      def test_get_id(self):
3          self.assertEqual(tweet1.get_id(), 1)
4
5      def test_get_usuario(self):
6          self.assertEqual(tweet1.get_usuario(), "usuario1")
```

```
7
8       def test_get_mensagem(self):
9           self.assertEqual(tweet1.get_mensagem(), "mensagem1"
                )
10
11      def teste_get_data_postagem(self):
12          self.assertEqual(tweet1.get_data_postagem(),
                datetime.today().replace(second=0, microsecond
                =0))
```

3.3 Classe TestPerfil

A classe TestPerfil cria testes para cada método da classe Perfil.

Código-fonte 12 – TestePerfil

```
1  class TestPerfil(unittest.TestCase):
2      def test_add_seguidor(self):
3          perfil1.add_seguidor(perfil2)
4          self.assertIn(perfil2, perfil1.get_seguidores())
5
6      def test_add_seguidos(self):
7          perfil1.add_seguidos(perfil2)
8          self.assertIn(perfil2, perfil1.get_seguidos())
9
10     def teset_add_tweet(self):
11         perfil1.add_tweet(tweet1)
12         self.assertIn(tweet1, perfil1.get_tweet())
13         self.assertEqual(tweet1, perfil1.get_tweets())
14
15     def test_get_tweets(self):
16         perfil1.add_tweet(tweet1)
17         perfil2.add_tweet(tweet2)
```

```
18          tweets = perfil1.get_tweets()
19          self.assertEqual(tweets, sorted(tweets, key=lambda
                tweet: tweet.get_data_postagem()))
20
21      def test_get_seguidos(self):
22          perfil1.add_seguidos(perfil2)
23          perfil1.add_seguidos(perfil3)
24          self.assertEqual(perfil1.get_seguidos(), [perfil2,
                perfil3])
25
26      def test_get_seguidores(self):
27          perfil1.add_seguidor(perfil2)
28          perfil1.add_seguidor(perfil3)
29          self.assertEqual(perfil1.get_seguidores(), [perfil2
                , perfil3])
30
31      def test_get_tweet(self):
32          perfil1.add_tweet(tweet1)
33          perfil1.add_tweet(tweet2)
34          self.assertEqual(perfil1.get_tweet(tweet1.get_id())
                , tweet1)
35          self.assertEqual(perfil1.get_tweet(tweet2.get_id())
                , tweet2)
36          self.assertIsNone(perfil1.get_tweet(999))
37
38      def test_get_timeline(self):
39          pass
40
41      def test_set_usuario(self):
42          pass
43
44      def test_get_usuario(self):
```

```
45          pass
46
47      def test_set_ativo(self):
48          pass
49
50      def test_is_ativo(self):
51          pass
```

4 TESTES DO VÍDEO

O vídeo com os testes estão em (AUTORES, 2025). A seguir a lista com os testes realizados.

1. Digitar opção incorreta no menu principal
2. Tentar acessar MyTwitter de uma conta não cadastrada
3. Criar perfil pessoa física: Nome - ana; CPF - 1
4. Criar perfil pessoa física: Nome - ana; CPF - 1
5. Criar perfil pessoa física: Nome - ana; CPF - 2
6. Criar perfil pessoa física: Nome - maria; CPF - 1
7. Criar perfil pessoa jurídica: Nome - zero; CNPJ - 1
8. Criar perfil pessoa jurídica: Nome - zero; CNPJ - 2
9. Entrar perfil Ana
10. Digitar opção inexisente
11. Ver seguidores
12. Ver seguidos
13. Tweettar - mais de 140 caracteres
14. Tweettar - menos de 140 caracteres
15. Seguir usuario - inexistente
16. Voltar e Entrar perfil Zero
17. Ver timeline
18. Ver seguidos
19. Tweettar - menos de 140 caracteres
20. Ver timeline
21. Seguir perfil ana
22. Ver timeline
23. Ver seguidos
24. Voltar e entrar perfil ana
25. Ver seguidores
26. Voltar e sair

REFERÊNCIAS

AUTORES. **MyTwitter – teste - 2**. 2025. Disponível em: <https://youtu.be/lrVb-6TfzkM>. Acesso em: 25 fev. 2025.