

Relatório de RPA

Nome: Lucas Curtolo Belem – RA 2400206

Automação de Consulta e Relatório de Raças de Cães

1. Introdução

Neste projeto, desenvolvi uma automação em Python que consulta dados de raças de cachorros por meio da API pública The Dog API. O sistema coleta, processa, armazena e envia por e-mail um relatório com as informações obtidas, cumprindo todos os requisitos da atividade de RPA.

2. Escolha da API

API escolhida foi “The Dog API”, pois acho interessante a diversidade de raças caninas existentes no mundo e achei interessante para conhecer algumas raças que nunca tinha ouvido falar e ver como são esses cachorros por meio das fotos que a API me disponibilizou.

3. Criação do Banco de Dados

Primeiro eu criei as duas tabelas que foi solicitado no escopo da atividade no meu banco de dados SQLite usando SQLAlchemy. A tabela Cachorros ficou responsável por armazenar os dados brutos obtidos da API, como nome da raça, grupo, altura, peso, expectativa de vida, temperamentos e URL da imagem e a tabela DadosProcessados foi criada para armazenar informações adicionais que eu processei a partir dos dados da API, A quantidade de temperamentos listados para cada raça armazenando e A expectativa de vida mínima e máxima da raça.

```
from sqlalchemy import Column, Integer, String, ForeignKey, create_engine
from sqlalchemy.orm import declarative_base, sessionmaker, relationship
```

```

Base = declarative_base()

class Cachorro(Base):
    __tablename__ = "cachorros"

    id = Column(Integer, primary_key=True)
    name = Column(String)
    breed_for = Column(String)
    breed_group = Column(String)
    life_span = Column(String)
    temperament = Column(String)
    country_code = Column(String)
    height_metric = Column(String)
    weight_metric = Column(String)
    image_url = Column(String)

    dados_processados = relationship(
        "DadosProcessados", back_populates="cachorro", uselist=False)

class DadosProcessados(Base):
    __tablename__ = "dados_processados"

    id = Column(Integer, primary_key=True)
    cachorro_id = Column(Integer, ForeignKey("cachorros.id"))
    temperament_count = Column(Integer)
    min_life_span = Column(Integer)
    max_life_span = Column(Integer)

    cachorro = relationship("Cachorro", back_populates="dados_processados")

engine = create_engine("sqlite:///projeto_rpa.db")
Base.metadata.create_all(engine)

Session = sessionmaker(bind=engine)

```

4. Requisição à API e Visualização

Após a criação do banco de dados, utilizei a biblioteca requests para fazer a requisição à API e acessar os dados das raças de cães. Em seguida, utilizei as bibliotecas PIL (Python Imaging Library) e io para permitir a visualização das imagens das raças diretamente no programa. Essas imagens são acessadas por meio da URL fornecida pela API e exibidas automaticamente para o usuário durante a execução.

```

prova_final.py > ...
1  import requests
2  from PIL import Image
3  from io import BytesIO

```

```

50 def mostrar_raca_por_id(raca_id):
51     url = "https://api.thedogapi.com/v1/breeds"
52     headers = {
53         "x-api-key": "live_lQAZAFRRtvwdlcc64vjw4je7BsrtfSquofW40NFCCPEgoNPNDofTZoq00sQestVd"
54     }
55
56     response = requests.get(url, headers=headers)
57     if response.status_code != 200:
58         print("Erro na API:", response.status_code)
59         return
60
61     dados = response.json()
62
63     raca = next((r for r in dados if r['id'] == raca_id), None)
64
65     if raca is None:
66         print(f"Raça com id {raca_id} não encontrada.")
67         return
68
69     print(f"Nome: {raca.get('name', 'N/A')}")
70     print(f"Bred for: {raca.get('bred_for', 'N/A')}")
71     print(f"Breed group: {raca.get('breed_group', 'N/A')}")
72     print(f"Life span: {raca.get('life_span', 'N/A')}")
73     print(f"Temperament: {raca.get('temperament', 'N/A')}")
74     print(f"country_code: {raca.get('country_code', 'N/A')}")
75     print(f"Height (cm): {raca.get('height', {}).get('metric', 'N/A')}")
76     print(f"Weight (kg): {raca.get('weight', {}).get('metric', 'N/A')}")
77
78     if 'image' in raca and 'url' in raca['image']:
79         imagem_url = raca['image']['url']
80         print("Mostrando imagem...")
81         imagem_response = requests.get(imagem_url)
82         if imagem_response.status_code == 200:
83             imagem = Image.open(BytesIO(imagem_response.content))
84             imagem.show()
85         else:
86             print("Erro ao baixar a imagem.")
87     else:
88         print("Imagem não disponível para esta raça.")
89
90     session = Session()
91     if session.query(Cachorro).filter_by(id=raca_id).first() is None:
92         novo_cachorro = Cachorro(
93             id=raca.get('id'),
94             name=raca.get('name'),
95             bred_for=raca.get('bred_for'),
96             breed_group=raca.get('breed_group'),
97             life_span=raca.get('life_span'),
98             breed_group=raca.get('breed_group'),
99             life_span=raca.get('life_span'),
100             temperament=raca.get('temperament'),
101             country_code=raca.get('country_code'),
102             height_metric=raca.get('height', {}).get('metric'),
103             weight_metric=raca.get('weight', {}).get('metric'),
104             image_url=raca.get('image', {}).get('url')
105         )
106     session.add(novo_cachorro)
107     session.commit()
108     print("Raça salva no banco de dados.")
109 else:
110     print("Raça já existe no banco de dados.")
111 processar_dados(raca_id, session)
112 session.close()

```

5. Processamento dos Dados

Na função `processar_dados()`, eu usei a biblioteca `re`, que serve para trabalhar com expressões regulares — ou seja, para encontrar partes específicas de um texto. No meu caso, eu queria pegar apenas os números do campo `life_span`, que mostra a expectativa de vida do cachorro (por exemplo, "10 - 12 years"). Usando o comando `re.findall(r'(\d+)', life_span)`, eu consigo pegar todos os números dessa frase, mesmo que venham juntos com palavras. Depois disso, salvo o mínimo e o máximo desses valores para dizer qual é a idade mínima e máxima que aquela raça costuma viver. Também conto quantos temperamentos a raça tem, separando por vírgulas, e salvo tudo isso no banco de dados. Assim, o sistema consegue guardar mais informações úteis sobre cada raça de cachorro.

```
4 import re
```

```
def processar_dados(raca_id, session):
    cachorro = session.query(Cachorro).filter_by(id=raca_id).first()
    if not cachorro:
        print("Raça não encontrada no banco para processar.")
        return

    # Contar temperamentos
    temperament = cachorro.temperament or ""
    temperament_list = [t.strip() for t in temperament.split(",") if t.strip()]
    temperament_count = len(temperament_list)

    # Extrair min e max life span com regex
    life_span = cachorro.life_span or ""
    anos = re.findall(r'(\d+)', life_span)
    min_life_span = int(anos[0]) if anos else None
    max_life_span = int(anos[1]) if len(anos) > 1 else min_life_span

    dados_existentes = session.query(
        DadosProcessados).filter_by(cachorro_id=raca_id).first()

    if dados_existentes is None:
        dados_processados = DadosProcessados(
            cachorro_id=raca_id,
            temperament_count=temperament_count,
            min_life_span=min_life_span,
            max_life_span=max_life_span
        )
        session.add(dados_processados)
    else:
        dados_existentes.temperament_count = temperament_count
        dados_existentes.min_life_span = min_life_span
        dados_existentes.max_life_span = max_life_span

    session.commit()
    print(f"Dados processados para a raça {cachorro.name} foram salvos.")
```

6. Geração do Relatório

Criei a função `gerar_relatorio()` para fazer um levantamento em forma de texto organizado com os dados de todas as raças de cachorros que foram buscadas e processadas no banco de dados. Essa função acessa as duas tabelas do banco: a que contém as informações da API (nome, temperamentos, expectativa de vida, etc.) e a tabela com os dados adicionais que eu processei (como o número de temperamentos, idade mínima e máxima de vida). O relatório final é montado em formato de texto, mostrando essas informações de forma clara para cada raça. Esse texto depois é enviado por e-mail como o corpo da mensagem.

```
def gerar_relatorio():
    session = Session()
    relatorio = "Relatório de Raças de Cães Processadas\n\n"
    dados = session.query(Cachorro).join(DadosProcessados).all()
    if not dados:
        session.close()
        return "Nenhum dado processado encontrado."
    for cachorro in dados:
        dp = cachorro.dados_processados
        relatorio += (
            f"Raça: {cachorro.name}\n"
            f"Temperamentos: {cachorro.temperament}\n"
            f"Número de temperamentos: {dp.temperament_count}\n"
            f"Vida útil: {cachorro.life_span} (Min: {dp.min_life_span}, Max: {dp.max_life_span})\n"
            "-----\n"
        )
    session.close()
    return relatorio
```

7. Criptografia da Senha

Para manter a segurança dos dados, especialmente da senha do meu e-mail (que é usada para o envio automático do relatório), utilizei a biblioteca cryptography. Com ela, criei uma chave secreta e usei essa chave para criptografar a senha do meu e-mail, evitando que ela fique exposta de forma visível no código.

```
from cryptography.fernet import Fernet
```

```
chave = b"ek0dnW1rX-_5L3_zCYQucBEaIlxcNigEh0E1sKWbZfk="
senha_criptografada = b"gAAAAABoQgtb7SV5Vnuypo70fpiphyVuL7RZNUIXYD_x311G8BPWTUdNy63FEbKCOLa5KsQ96bvDvShSnK9UugdCLJIn-GiQHkUNTB_KOfNQiCO_KEkRsVg="

def obter_senha():
    fernet = Fernet(chave)
    return fernet.decrypt(senha_criptografada).decode()
```

8. Envio de E-mail Automatizado

Após gerar o relatório com os dados das raças de cães, eu criei uma função chamada enviar_email() que é responsável por enviar automaticamente esse relatório por e-mail. Essa função usa a biblioteca smtplib, que permite se conectar a servidores de e-mail (neste caso, o Gmail). Dessa forma, todo o processo de envio de e-mail é automático e seguro, e o relatório com os dados das raças de cachorro é enviado sem precisar fazer isso manualmente.

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
```

```
def enviar_email(corpo):
    remetente = "lucascurtolobelem@gmail.com"
    destinatario = "lucasc.belem@gmail.com"
    senha = obter_senha()

    assunto = "Relatório Automático - Raças de Cães"

    msg = MIMEText(corpo, "plain")
    msg["From"] = remetente
    msg["To"] = destinatario
    msg["Subject"] = assunto
    msg.attach(MIMEText(corpo, "plain"))

    servidor_smtp = "smtp.gmail.com"
    porta_smtp = 587

    with smtplib.SMTP(servidor_smtp, porta_smtp) as server:
        server.starttls()
        server.login(remetente, senha)
        server.send_message(msg)
    print("E-mail enviado com sucesso!")
```

9. Execução do Projeto

Na parte final do meu projeto, utilizei a estrutura `if __name__ == "__main__":` para garantir que todo o código fosse executado apenas quando o arquivo fosse rodado diretamente. Dentro dela, comecei fazendo uma busca das raças de cães com IDs de 1 a 10, utilizando a função `mostrar_raca_por_id()`, que realiza a requisição na API, exibe os dados da raça, salva no banco de dados e também processa informações adicionais, como o número de temperamentos e a expectativa de vida. Em seguida, utilizei a função `gerar_relatorio()` para montar um relatório com as raças coletadas, organizando essas informações de forma clara em um texto. Por fim, chamei a função `enviar_email(corpo_email)` para enviar esse relatório automaticamente por e-mail, com o conteúdo detalhado diretamente no corpo da mensagem. Com isso, o sistema realiza de forma automática todas as etapas do projeto: coleta, armazenamento, processamento, geração de relatório e envio por e-mail.

```

if __name__ == "__main__":
    # Buscar e salvar/processar raças com IDs de 1 a 10
    for raca_id in range(1, 11):
        mostrar_raca_por_id(raca_id)

    # Gerar o relatório
    corpo_email = gerar_relatorio()

    # Enviar o e-mail com o relatório
    enviar_email(corpo_email)

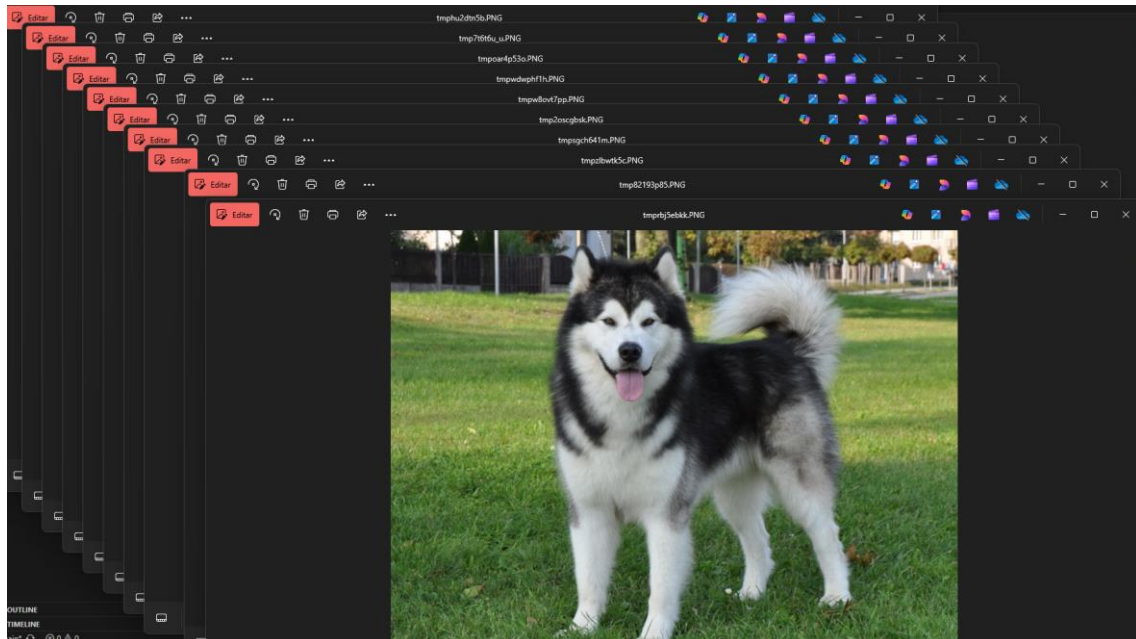
```

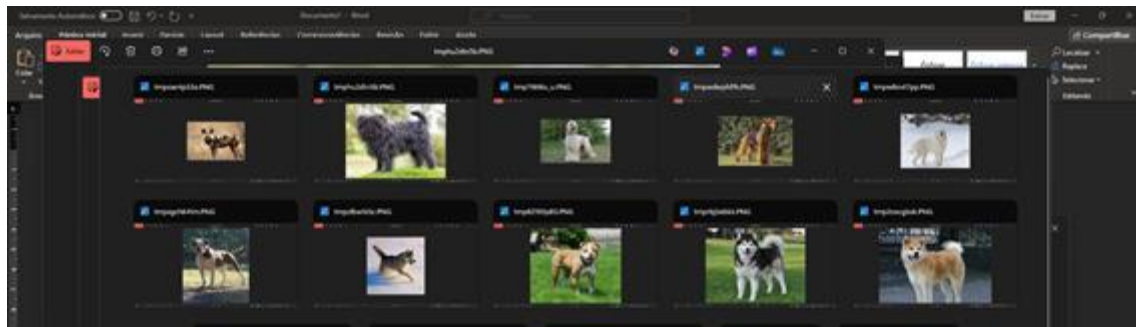
10. Prints do Projeto

Para começar minha automação, digitei o nome do meu arquivo e dei um “enter”

```
PS C:\Users\Lucas\Documents\RPA\Prova_Final_RPA> python prova_final.py
```

Aqui estão as 10 raças de cachorros que aparecem quando fazemos a requisição inicial à API.

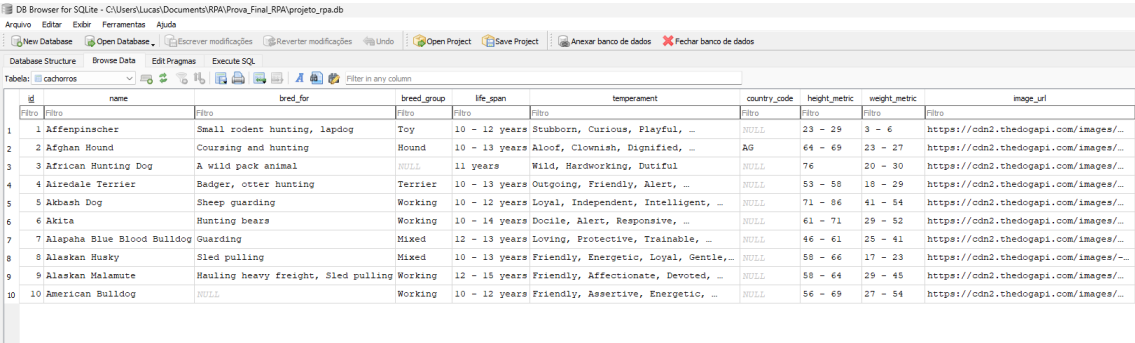




Aqui podemos ver que, para cada ID consultado, o sistema retorna as informações da raça do cachorro, exibe sua imagem e, em seguida, armazena os dados nas duas tabelas do banco de dados.

```
PS C:\Users\Lucas\Documents\RPA\Prova_Final_RPA> python prova_final.py
Nome: Affenpinscher
Bred for: Small rodent hunting, lapdog
Breed group: Toy
Life span: 10 - 12 years
Temperament: Stubborn, Curious, Playful, Adventurous, Active, Fun-loving
country_code: N/A
Height (cm): 23 - 29
Weight (kg): 3 - 6
Mostrando imagem...
Raça salva no banco de dados.
Dados processados para a raça Affenpinscher foram salvos.
Nome: Afghan Hound
Bred for: Coursing and hunting
Breed group: Hound
Life span: 10 - 13 years
Temperament: Aloof, Clownish, Dignified, Independent, Happy
country_code: AG
Height (cm): 64 - 69
Weight (kg): 23 - 27
Mostrando imagem...
Raça salva no banco de dados.
Dados processados para a raça Afghan Hound foram salvos.
Nome: African Hunting Dog
Bred for: A wild pack animal
Breed group: N/A
Life span: 11 years
Temperament: Wild, Hardworking, Dutiful
country_code: N/A
Height (cm): 76
Weight (kg): 20 - 30
Mostrando imagem...
Raça salva no banco de dados.
Dados processados para a raça African Hunting Dog foram salvos.
Nome: Airedale Terrier
Bred for: Badger, otter hunting
Breed group: Terrier
Life span: 10 - 13 years
Temperament: Outgoing, Friendly, Alert, Confident, Intelligent, Courageous
country_code: N/A
Height (cm): 53 - 58
Weight (kg): 18 - 29
Mostrando imagem...
Raça salva no banco de dados.
Dados processados para a raça Airedale Terrier foram salvos.
Nome: Akbash Dog
Bred for: Sheep guarding
Breed group: Working
Life span: 10 - 12 years
Temperament: Loyal, Independent, Intelligent, Brave
country_code: N/A
Height (cm): 71 - 86
Weight (kg): 41 - 54
Mostrando imagem...
Raça salva no banco de dados.
```

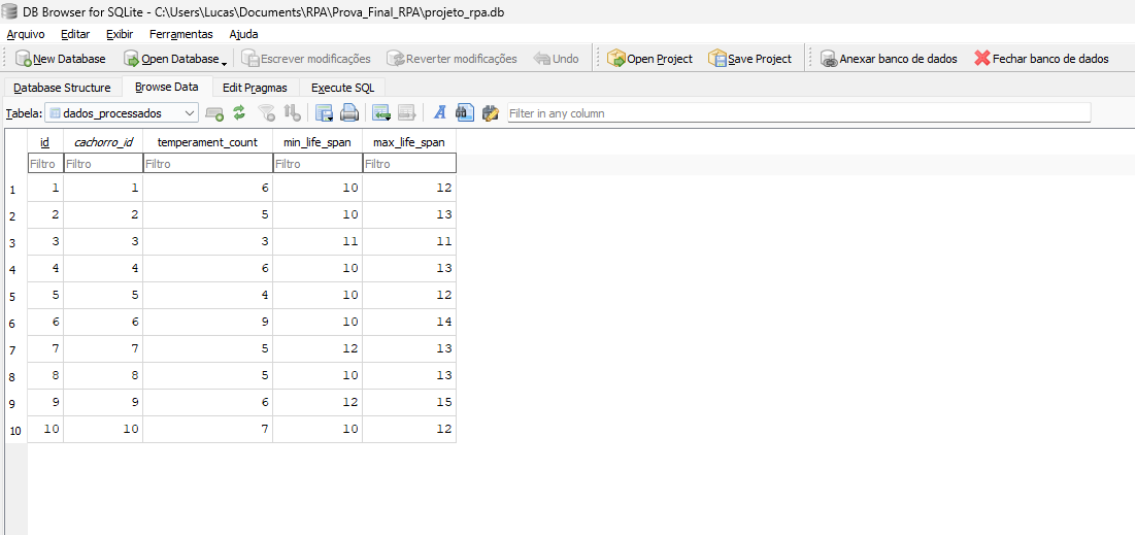

Esta é a tabela dos cachorros, mostrando todas as informações que foram salvas no banco de dados após a requisição.



The screenshot shows the DB Browser for SQLite interface with the 'cachorros' table selected. The table contains 10 rows of data, each representing a dog breed with various attributes like name, breed_for, breed_group, life_span, temperament, country_code, height_metric, weight_metric, and image_url.

id	name	breed_for	breed_group	life_span	temperament	country_code	height_metric	weight_metric	image_url
1	Affenpinscher	Small rodent hunting, lapdog	Toy	10 - 12 years	Stubborn, Curious, Playful, ...	NULL	23 - 29	3 - 6	https://cdn2.thedogapi.com/images/...
2	Afghan Hound	Coursing and hunting	Hound	10 - 13 years	Alloof, Clownish, Dignified, ...	AG	64 - 69	23 - 27	https://cdn2.thedogapi.com/images/...
3	African Hunting Dog	A wild pack animal	NULL	11 years	Wild, Hardworking, Dutiful	NULL	76	20 - 30	https://cdn2.thedogapi.com/images/...
4	Alredale Terrier	Badger, otter hunting	Terrier	10 - 13 years	Outgoing, Friendly, Alert, ...	NULL	53 - 58	18 - 29	https://cdn2.thedogapi.com/images/...
5	Akbash Dog	Sheep guarding	Working	10 - 12 years	Loyal, Independent, Intelligent, ...	NULL	71 - 86	41 - 54	https://cdn2.thedogapi.com/images/...
6	Akita	Hunting bears	Working	10 - 14 years	Docile, Alert, Responsive, ...	NULL	61 - 71	29 - 52	https://cdn2.thedogapi.com/images/...
7	Alapaha Blue Blood Bulldog	Guarding	Mixed	12 - 13 years	Loving, Protective, Trainable, ...	NULL	46 - 61	25 - 41	https://cdn2.thedogapi.com/images/...
8	Alaskan Husky	Sled pulling	Mixed	10 - 13 years	Friendly, Energetic, Loyal, Gentle, ...	NULL	58 - 66	17 - 23	https://cdn2.thedogapi.com/images/...
9	Alaskan Malamute	Hauling heavy freight, sled pulling	Working	12 - 15 years	Friendly, Affectionate, Devoted, ...	NULL	58 - 64	29 - 45	https://cdn2.thedogapi.com/images/...
10	American Bulldog	NULL	Working	10 - 12 years	Friendly, Assertive, Energetic, ...	NULL	56 - 69	27 - 54	https://cdn2.thedogapi.com/images/...

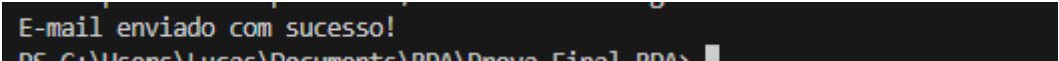
Nesta imagem, podemos visualizar a tabela de dados processados, onde são armazenadas informações adicionais, como a quantidade de temperamentos e a expectativa de vida mínima e máxima de cada raça.



The screenshot shows the DB Browser for SQLite interface with the 'dados_processados' table selected. The table contains 10 rows of data, each representing a processed dog breed with attributes like id, cachorro_id, temperament_count, min_life_span, and max_life_span.

id	cachorro_id	temperament_count	min_life_span	max_life_span
1	1	6	10	12
2	2	5	10	13
3	3	3	11	11
4	4	6	10	13
5	5	4	10	12
6	6	9	10	14
7	7	5	12	13
8	8	5	10	13
9	9	6	12	15
10	10	7	10	12

Por fim, o terminal confirma que o e-mail com o relatório das raças de cachorros foi enviado com sucesso ao destinatário e ao abrir o e-mail, podemos ver como ele chegou ao seu destino final.



E-mail enviado com sucesso!
D5: C:\Users\Lucas\Documents\RPA\Prova_Final_RPA\



lucascurtolobelem@gmail.com

para mim ▾

Traduza para o português

Relatório de Raças de Cães Processadas

Raça: Affenpinscher
Temperamentos: Stubborn, Curious, Playful, Adventurous, Active, Fun-loving
Número de temperamentos: 6
Vida útil: 10 - 12 years (Min: 10, Max: 12)

Raça: Afghan Hound
Temperamentos: Aloof, Clownish, Dignified, Independent, Happy
Número de temperamentos: 5
Vida útil: 10 - 13 years (Min: 10, Max: 13)

Raça: African Hunting Dog
Temperamentos: Wild, Hardworking, Dutiful
Número de temperamentos: 3
Vida útil: 11 years (Min: 11, Max: 11)

Raça: Airedale Terrier
Temperamentos: Outgoing, Friendly, Alert, Confident, Intelligent, Courageous
Número de temperamentos: 6
Vida útil: 10 - 13 years (Min: 10, Max: 13)

Raça: Akbash Dog
Temperamentos: Loyal, Independent, Intelligent, Brave
Número de temperamentos: 4
Vida útil: 10 - 12 years (Min: 10, Max: 12)

Raça: Akita
Temperamentos: Docile, Alert, Responsive, Dignified, Composed, Friendly, Receptive, Faithful, Courageous
Número de temperamentos: 9
Vida útil: 10 - 14 years (Min: 10, Max: 14)

Raça: Alapaha Blue Blood Bulldog
Temperamentos: Loving, Protective, Trainable, Dutiful, Responsible
Número de temperamentos: 5

Temperamentos: Calm, Friendly, Alert, Confident, Intelligent, Courageous
Número de temperamentos: 6
Vida útil: 10 - 13 years (Min: 10, Max: 13)

Raça: Akbash Dog
Temperamentos: Loyal, Independent, Intelligent, Brave
Número de temperamentos: 4
Vida útil: 10 - 12 years (Min: 10, Max: 12)

Raça: Akita
Temperamentos: Docile, Alert, Responsive, Dignified, Composed, Friendly, Receptive, Faithful, Courageous
Número de temperamentos: 9
Vida útil: 10 - 14 years (Min: 10, Max: 14)

Raça: Alapaha Blue Blood Bulldog
Temperamentos: Loving, Protective, Trainable, Dutiful, Responsible
Número de temperamentos: 5
Vida útil: 12 - 13 years (Min: 12, Max: 13)

Raça: Alaskan Husky
Temperamentos: Friendly, Energetic, Loyal, Gentle, Confident
Número de temperamentos: 5
Vida útil: 10 - 13 years (Min: 10, Max: 13)

Raça: Alaskan Malamute
Temperamentos: Friendly, Affectionate, Devoted, Loyal, Dignified, Playful
Número de temperamentos: 6
Vida útil: 12 - 15 years (Min: 12, Max: 15)

Raça: American Bulldog
Temperamentos: Friendly, Assertive, Energetic, Loyal, Gentle, Confident, Dominant
Número de temperamentos: 7
Vida útil: 10 - 12 years (Min: 10, Max: 12)

Responder Encaminhar

11. Conclusão

Durante o desenvolvimento deste projeto, enfrentei alguns desafios, principalmente na hora de entender como organizar o banco de dados usando o SQLAlchemy e fazer a integração com a API dos cachorros. Também tive certa dificuldade para lidar com a parte de segurança no envio de e-mails, como a criptografia da senha, que foi algo novo pra mim. Apesar disso, foi um aprendizado muito válido. Consegui entender melhor como funciona o consumo de APIs, como salvar e processar informações em um banco de dados, e até como automatizar o envio de um relatório por e-mail. Foi interessante ver como várias bibliotecas do Python se conectam para construir algo funcional e útil. No fim, o projeto me ajudou a colocar em prática várias coisas que vi em aula e me deixou mais confiante para criar automações com Python.