

ARTEFATOS DO PROJETO DE SOFTWARE

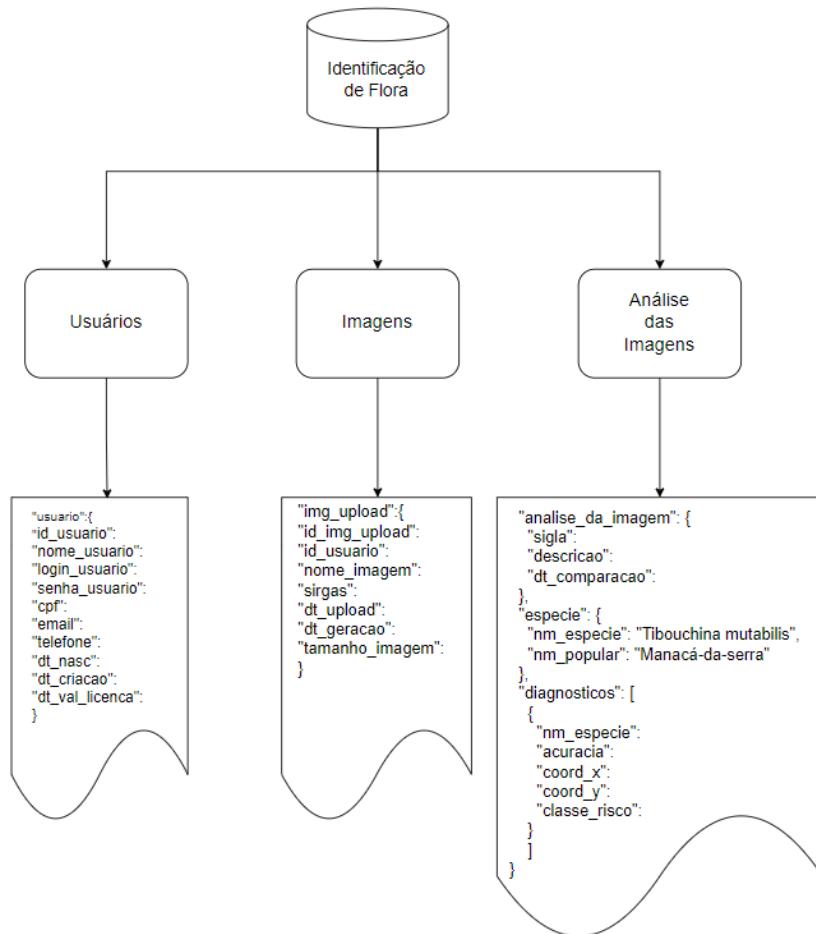
SUMÁRIO

BANCO DE DADOS	3
INTERFACE GRÁFICA	3
ALGORITMO DE ORDENAÇÃO	7
MODELO DE APRENDIZAGEM FCNN	7
APLICAÇÃO MOBILE	9
PROTÓTIPO DA APLICAÇÃO	9
API PARA COMUNICAÇÃO COM RECURSOS EM NUVEM	12

BANCO DE DADOS

Para a implementação do sistema, foram desenvolvidos o Modelo Lógico(Figura 1) do banco de dados não-relacional do aplicativo, que inclui uma base de dados das imagens a serem analisadas pelo sistema, assim como uma base de dados dos clientes cadastrados e o diagnóstico realizado pelo sistema, apontando quais espécies foram identificadas em uma determinada imagem.

Figura 1 – Modelo lógico- Banco NoSQL.

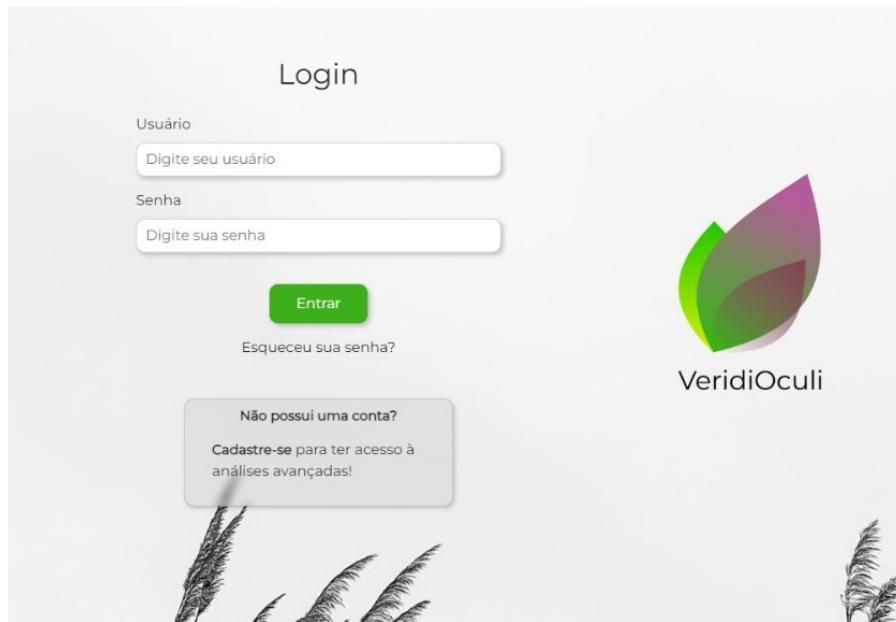


Fonte: Fonte: Os Autores(2023)

INTERFACE GRÁFICA

Utilizando a interface gráfica em Java, foi desenvolvido o sistema de Acesso do Usuário, bem como a funcionalidade do CRUD, onde é possível cadastrar,listar, alterar e excluir usuários do sistema. Em busca de tornar o sistema intuitivo e de fácil acessibilidade ao usuário, foram feitas modificações nos protótipos. Com base nas teorias de processamento humano da informação.

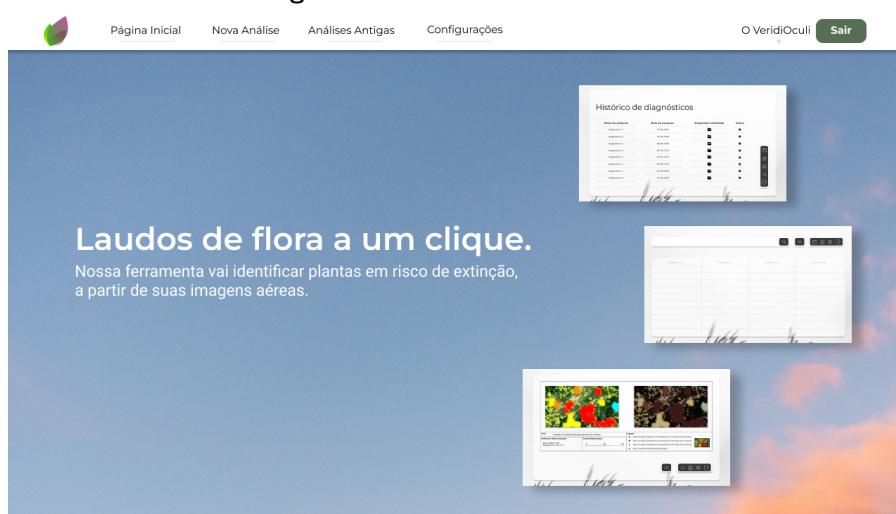
Figura 2 – Tela de Login



Fonte: Fonte: Os Autores(2023)

A tela de login para o sistema web foi desenvolvido de acordo com os Padrões de Interface, como campo se senha após o campo de login, botão de confirmar em destaque e link para recuperação de senha, colocado em menor destaque.

Figura 3 – Tela de boas-vindas.



Fonte: Fonte: Os Autores(2023)

Após o login do usuário, a tela de boas vindas mostra ao usuário o preview das telas que irá visualizar no sistema.



Figura 4 – Tela de Configurações pessoais. Fonte: Os Autores

Na área de configurações de usuário, este será capaz de visualizar as proprias informações.

Informações básicas		Informações de contato		Endereços	
Nome	Gabriel Soares da Silva	Email	gabriel_soares_da_silva@outlook.com	Pessoal	Rua Das Folhas Verdes, 885, Caiati-SP
Data de nascimento	01/01/1975	Telefone	(13) 99669-5885	Trabalho	Rua Dos Arbustos, 154, Registro-SP
Gênero	Masculino		(13) 99775-5878		

Figura 5 – Informações do usuário. Fonte: Os Autores

Será também capaz de adicionar e atualizar as informações de usuário.

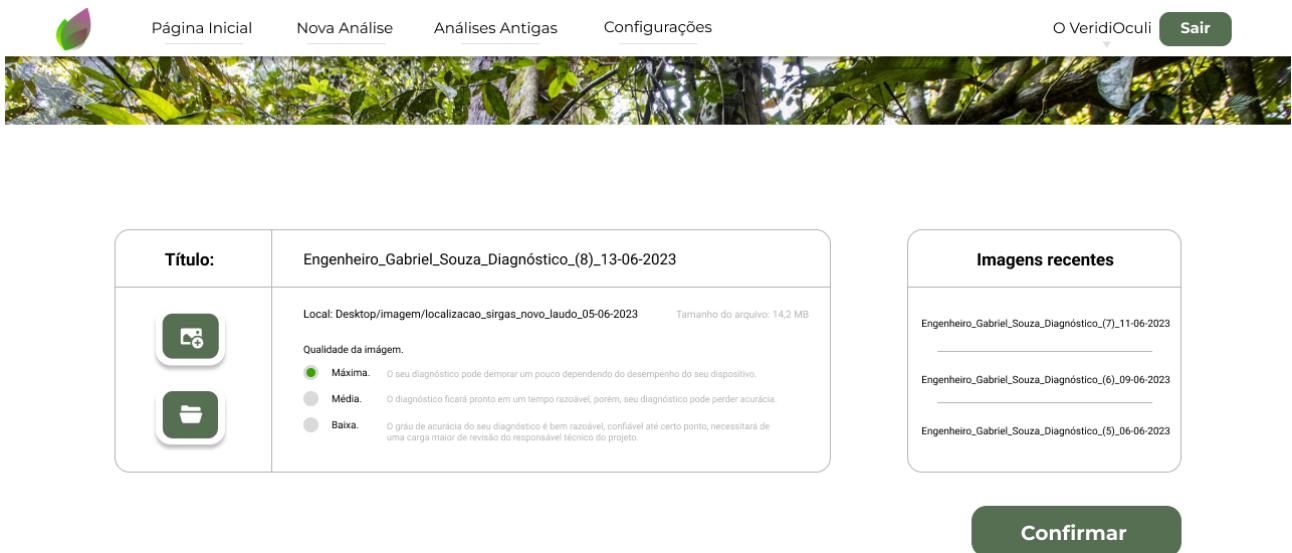


Figura 6 – Tela do sistema de carregamento de imagens. Fonte: Os Autores

O protótipo de tela de carregamento de uma nova imagem para análise leva em conta a usabilidade e clareza nas informações do sistema.

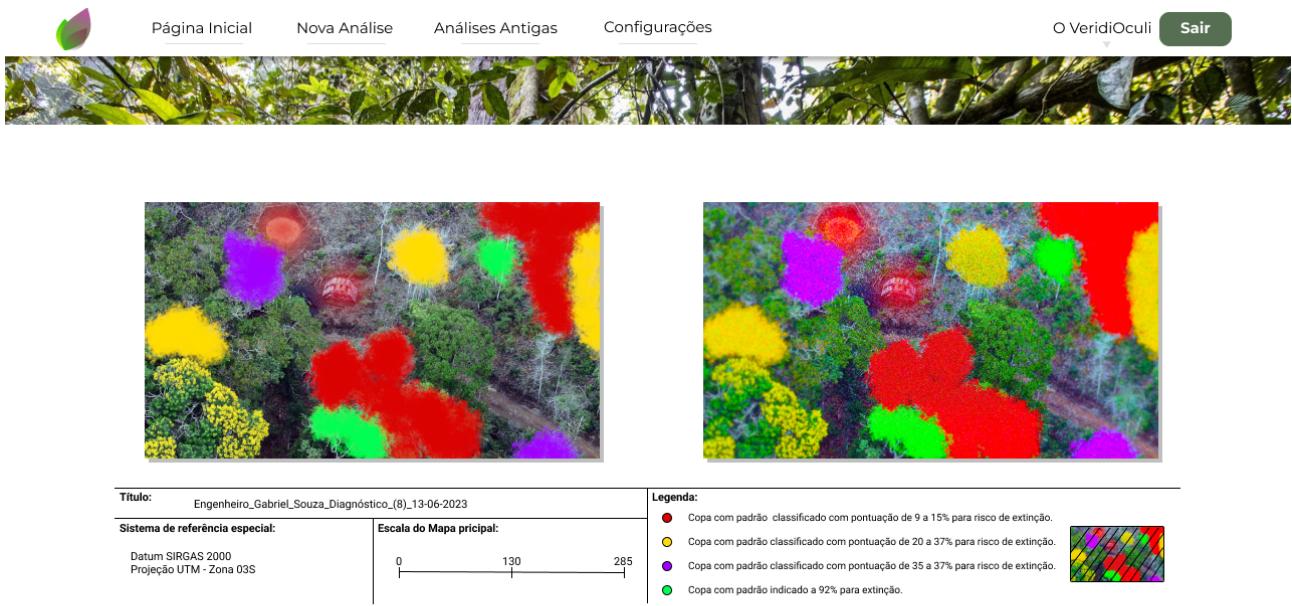


Figura 7 – Tela do sistema de análise. Fonte: Os Autores

Após a análise feita pelo modelo treinado, será possível buscar as espécies de árvores encontradas na foto.

ALGORITMO DE ORDENAÇÃO

A análise de complexidade, realizada no presente trabalho para avaliar a eficiência do algoritmo de ordenação QuickSort, nos mostrou que o melhor caso e o caso médio, temos que o número de iterações necessárias para chegar ao caso base, onde as partições têm tamanho 1, é log de n na base 2. Assim, a complexidade em termos de "k" é $O(\log n)$.

Substituindo essa complexidade na recorrência:

$$T(n) = 2T(k) + O(n) = 2O(\log n) + O(n) = O(\log n) + O(n) \quad (1)$$

$$T(n) = O(n \log n) \quad (2)$$

Portanto, no melhor e caso médio, a complexidade do algoritmo QuickSort é $O(n \log n)$ (Equação 6)

A análise de recorrência para o pior caso, onde o número de iterações necessárias para chegar ao caso base é "n".

Substituindo " $T(n-1)$ " por " $T(k)$ " e considerando o número de iterações "n":

$$T(n) = T(n - 1) + C + O(n) = T(k) + Cn + O(n) \quad (3)$$

$$T(n) = O(k) + Cn + O(n) = O(n) + O(n) + O(n) = O(n^2) \quad (4)$$

$$T(n) = O(n^2) \quad (5)$$

Portanto, no pior caso, a complexidade do algoritmo QuickSort é $O(n^2)$ (Equação 11).

MODELO DE APRENDIZAGEM DE MÁQUINA

Para a implementação do sistema, foram desenvolvidos o Modelo Lógico(Figura 2) e o Modelo Físico(Anexo 1)do banco de dados não-relacional do aplicativo, que inclui uma base de dados das imagens a serem analisadas pelo sistema, assim como uma base de dados dos clientes cadastrados e o diagnóstico realizado pelo sistema, apontando quais espécies foram identificadas em uma determinada imagem.

MODELO DE APRENDIZAGEM FCNN

Para desenvolver o treinamento do modelo de aprendizagem profunda, foi utilizado o código em python, para classificar as imagens catalogadas no conjunto.

Listing 1 – Treinamento do modelo de aprendizagem profunda

```
1  
2 import tensorflow as tf  
3 from tensorflow.keras import layers, models  
4 import matplotlib.pyplot as plt  
5  
6 import numpy as np
```

```

7  from sklearn.model_selection import KFold
8  import os
9
10 data_dir = 'img'
11
12 batch_size = 32
13 img_height = 170
14 img_width = 170
15 num_classes = len(os.listdir(data_dir))
16 num_folds = 10
17 epochs = 100
18
19 def create_model():
20     model = models.Sequential()
21     model.add(layers.Flatten(input_shape=(img_height, img_width, 3)))
22     model.add(layers.Dense(512, activation='relu'))
23     model.add(layers.Dense(256, activation='relu'))
24     model.add(layers.Dense(num_classes, activation='softmax'))
25     model.compile(optimizer='adam',
26                     loss='sparse_categorical_crossentropy',
27                     metrics=['accuracy'])
28     return model
29
30 def load_data(file_paths, labels, img_height, img_width):
31     images = []
32     for file_path in file_paths:
33         img = tf.keras.preprocessing.image.load_img(file_path,
34             target_size=(img_height, img_width))
35         img_array = tf.keras.preprocessing.image.img_to_array(img)
36         images.append(img_array)
37     return np.array(images), np.array(labels)
38
39 # Preparar os caminhos e labels
40 file_paths = []
41 labels = []
42 class_names = os.listdir(data_dir)
43 class_indices = {class_name: i for i, class_name in enumerate(
44     class_names)}
45
46 for class_name in class_names:
47     class_dir = os.path.join(data_dir, class_name)
48     class_files = os.listdir(class_dir)
49     file_paths.extend([os.path.join(class_dir, f) for f in class_files
50         ])
51     labels.extend([class_indices[class_name]] * len(class_files))
52
53 # Inicializar KFold
54 kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)

```

```

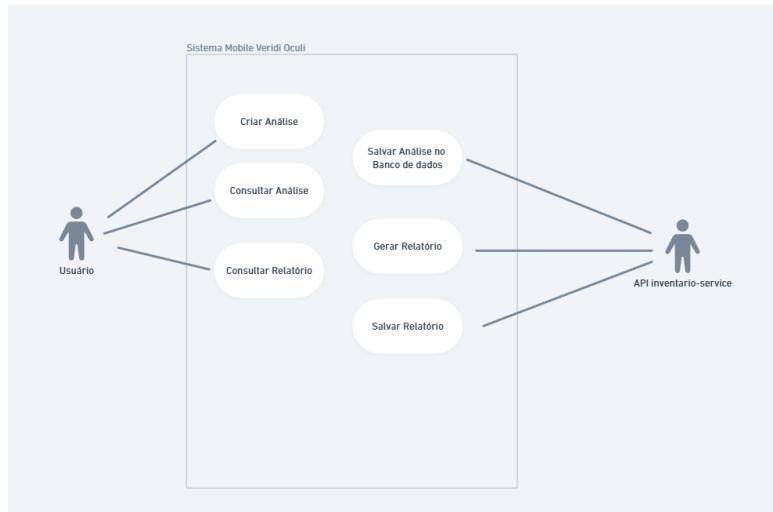
53
54 fold_accuracies = []
55 fold_no = 1
56 for train_index, val_index in kf.split(file_paths):
57     print(f'Training fold {fold_no}...')
58
59     train_files = [file_paths[i] for i in train_index]
60     val_files = [file_paths[i] for i in val_index]
61     train_labels = [labels[i] for i in train_index]
62     val_labels = [labels[i] for i in val_index]
63
64     x_train, y_train = load_data(train_files, train_labels, img_height,
65         , img_width)
65     x_val, y_val = load_data(val_files, val_labels, img_height,
66         img_width)
67
68     # Normalizar os dados
69     x_train = x_train / 255.0
70     x_val = x_val / 255.0
71
72     model = create_model()
73
74     # Treinar o modelo
75     history = model.fit(x_train, y_train, epochs=epochs, batch_size=
76         batch_size, validation_data=(x_val, y_val))
77
78     # Avaliar o modelo
79     val_loss, val_acc = model.evaluate(x_val, y_val)
80     print(f'Fold {fold_no} validation accuracy: {val_acc}')
81
82     fold_accuracies.append(val_acc)
83     fold_no += 1
84
85 print(f'Validation accuracies for each fold: {fold_accuracies}')
86 print(f'Mean validation accuracy: {np.mean(fold_accuracies)}')
87 print(f'Número de classes: {num_classes}')
88 plt.plot(history.history['accuracy'], label='accuracy')
89 plt.plot(history.history['val_accuracy'], label='val_accuracy')
90 plt.xlabel('Epoch')
91 plt.ylabel('Accuracy')
92 plt.legend(loc='lower right')
93 plt.show()

```

APLICAÇÃO MOBILE - PROTÓTIPO DA APLICAÇÃO

Desenvolvimento de aplicação mobile para geração e visualização de relatórios com base nas análises criadas com o modelo de identificação.

Figura 8 – Casos de Uso - Diagrama Mobile



Fonte: Fonte: Os Autores(2024)

O diagrama da figura 8, demonstra de forma simplificada os casos de uso da aplicação mobile.

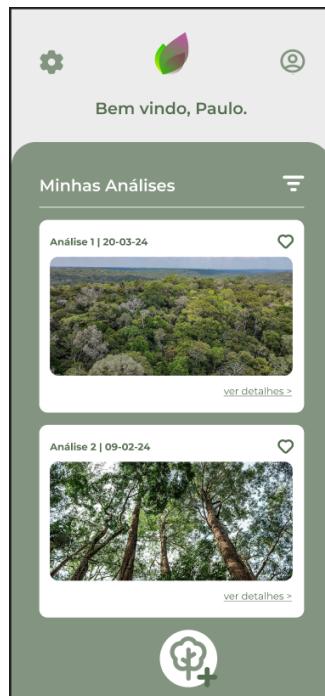
Figura 9 – Tela de entrada - Mobile



Fonte: Fonte: Os Autores(2024)

A tela de entrada (figura 9), utiliza os mesmos padrões de tipografia e cores aplicados na versão web.

Figura 10 – Tela de Análises - Mobile



Fonte: Fonte: Os Autores(2024)

Feito o login, o usuário tem à sua disposição as análises realizadas anteriormente.

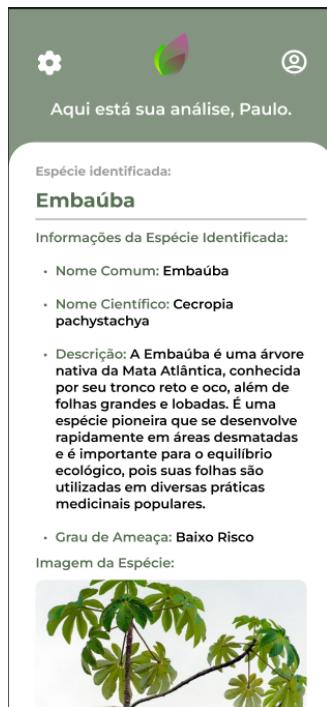
Figura 11 – Tela de Nova Análise - Mobile



Fonte: Fonte: Os Autores(2024)

O usuário pode também solicitar uma nova análise. Devendo para tal, inserir uma imagem e solicitar a identificação das espécies de árvores.

Figura 12 – Tela de Relatório - Mobile



Fonte: Fonte: Os Autores(2024)

Ao selecionar uma análise já processada, será disponibilizado o relatório de identificação das espécies encontradas na imagem disponibilizada pelo usuário. Nele constará informações sobre a espécie encontrada, e as marcações onde foram identificadas as espécies e outras informações como acurácia e posição na foto.

API PARA COMUNICAÇÃO COM RECURSOS EM NUVEM

Para fazer a comunicação com os recursos em nuvem como banco de dados, Storage Services e API generativa de textos, foi desenvolvido uma API em Python, hospedada em um EC2 (Elastic Computing Cloud) na AWS.

Listing 2 – API para geração de relatórios e interação com S3

```
1 import boto3
2 import json
3 import openai
4 from flask import Blueprint, jsonify, request
5 from dotenv import load_dotenv
6 import os
7 from app.services.genAnalise import genAnalise
8 from app.services.s3_service import fetch_json_data
9
10
11 load_dotenv()
12 bp = Blueprint('analysis', __name__)
13
14 # key = "analise01.json"
```

```

15     s3 = boto3.client(
16         's3',
17         aws_access_key_id=os.getenv("AWS_ACCESS_KEY_ID"),
18         aws_secret_access_key=os.getenv("AWS_SECRET_ACCESS_KEY")
19     )
20
21     openai.api_key = os.getenv("OPENAI_API_KEY")
22     BUCKET_ANALISES = os.getenv("BUCKET_ANALISES")
23     BUCKET_RELATORIO = os.getenv("BUCKET_RELATORIO")
24     BUCKET_IMAGENS = os.getenv("BUCKET_IMAGENS")
25
26     def save_report_to_bucket(bucket_name, key, content):
27         """Salvar o relatório gerado no bucket S3"""
28         s3.put_object(Bucket=bucket_name, Key=key, Body=json.dumps(
29             content))
30
31     @bp.route('/generate-report/<analysis_key>', methods=['POST'])
32     def generate_report(analysis_key):
33         """Gera o relatório com base em uma análise."""
34         try:
35             # 1. Buscar a análise no bucket de análises
36             analysis_data = fetch_json_data(BUCKET_ANALISES,
37                 analysis_key)
38
39             # 2. Obter a URL da imagem associada
40             image_id = analysis_data.get('image_details', {}).get(
41                 'image_id', '')
42             image_url = f"https://{BUCKET_IMAGENS}.s3.amazonaws.com/{
43                 image_id}"
44
45             # 3. Gerar o relatório com a OpenAI API
46             report = genAnalise(analysis_data, image_url)
47             print(report)
48             if report:
49                 # 4. Salvar o relatório no bucket de relatórios
50                 report_key = f"report_{analysis_key.split('.')[0]}.json"
51                 save_report_to_bucket(BUCKET_RELATORIO, report_key,
52                     json.loads(report))
53                 return jsonify({"status": "success", "report_key": report_key}), 200
54             else:
55                 return jsonify({"status": "error", "message": "Erro ao
56                     gerar relatório"}), 500
57         except Exception as e:
58             return jsonify({"status": "error", "message": str(e)}),
59             500

```

Esta API(Listing 2) cria métodos para que a aplicação mobile possa interagir com os recursos

em nuvem, buscando e salvando documentos relacionados ao usuário.

Listing 3 – API para comunicação com AI generativa OpenAI

```
1 from openai import OpenAI
2 import json
3 import os
4 from dotenv import load_dotenv
5
6 load_dotenv()
7 client = OpenAI(api_key=os.environ.get("OPENAI_API_KEY"))
8
9
10 def genAnalise(analysis_data, image_url):
11     print("Gera o relatório de inventário florestal usando a"
12          "API do OpenAI.")
13     print(analysis_data)
14     try:
15         prompt = f"""
16             Crie um relatório de inventário florestal com base nos
17             seguintes dados:
18
19             - Dados da análise (resumo): {json.dumps(analysis_data,
20                 ensure_ascii=False, indent=2)[:1000]}...
21             - Imagem associada: {image_url}
22
23             Siga o seguinte modelo:
24             - ID da Análise
25             - Data da Análise
26             - Localização (latitude e longitude)
27             - Resumo das identificações (alta, média e baixa
28                 confiança)
29             - Detalhes dos pontos de identificação (coordenadas e
30                 confiança)
31             """
32
33             response = client.chat.completions.create(
34                 model="gpt-3.5-turbo",
35                 messages=[
36                     {"role": "system", "content": "Você é um assistente especializado em criação de relatórios."},
37                     {"role": "user", "content": prompt}]
38             )
39
40             if response and response.choices:
41                 return response.choices[0].message.content
42             else:
43                 raise ValueError("Resposta inválida ou vazia da OpenAI
44                             API.")
```

```
39     except Exception as e:  
40         print(f"Erro ao gerar relatório: {e}")  
41     return None
```

Esta segunda API (Listing 3) é utilizada para comunicação com a API generativa da OpenAi, para tornar os dados conseguidos através da análise, legíveis e de fácil compreensão pelo usuário.