FIAP – FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA

EDUARDO ABREU DE SOUZA

GABRIEL DOS ANJOS

GABRIELLY LORENTZ

HEITOR FERNANDES

JOÃO PEDRO DE SOUZA

MATEUS MATTOS

SOL IA

SISTEMA DE LIMPEZA INTELIGENTE E AUTÔNOMO

SÃO PAULO

2025

Tecnologias utilizadas no Algoritmo:

Linguagem	Python
API	FASTAPI
BANCO DE DADOS	MONGODB
LLM	OPENAI,MODELO=gpt-4o-mini

Análises de complexidade

API Análise da imagem via OpenAI:

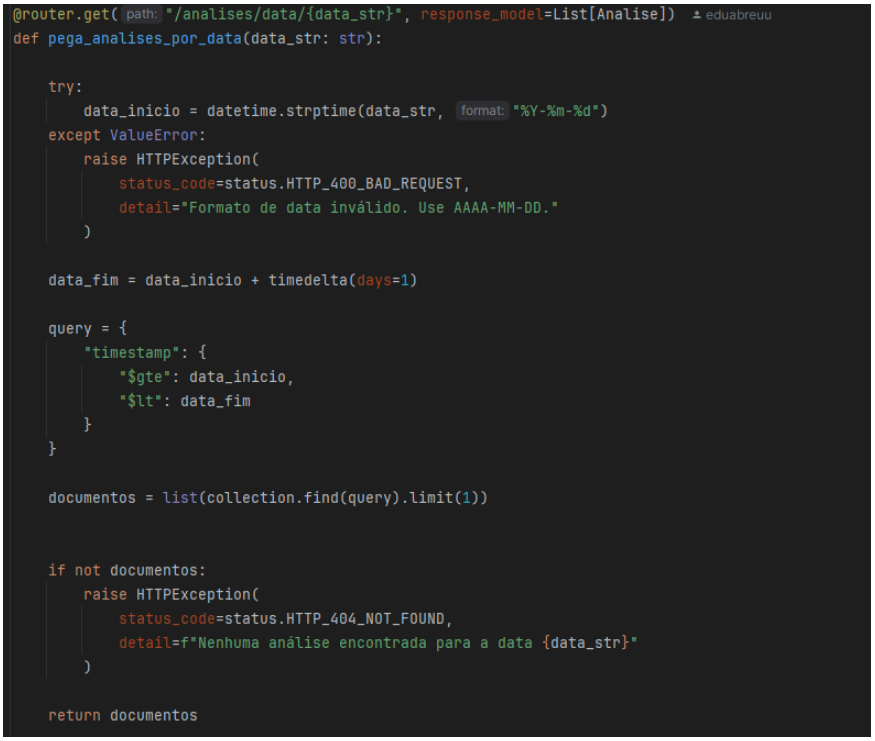
```python
@router.post("/analisar")  eduabreuu
async def analisar(input_data: AnaliseRequest):
    resultado = await analisar_painel(input_data.url)

    registro = {
        "entrada": input_data.model_dump(),
        "saida": resultado,
        "timestamp": datetime.now(timezone.utc)
    }
    collection.insert_one(registro)

    return ResponseOutput(
        status="success",
        url=input_data.url,
        need_cleaning=resultado.get("needCleaning", False),
        solar_level= resultado.get("nivelSolar", 0),
        response=resultado.get("analise", ""),
        date_time=datetime.now(timezone.utc)
    )
```

O método POST recebe uma URL de imagem e a envia para a função `analisar_painel`. Esta função faz uma requisição à API da OpenAI, passando a URL da imagem juntamente com as instruções para a análise. A API retorna a análise da imagem no formato JSON. Após receber essa resposta, a função armazena a análise na variável resultado, que possui o tipo DICT. A análise é então transformada em um objeto JSON, que é adicionado no banco de dados MongoDB.

`Endpoint Post:(“/analisar”) → O(1)`

`Função analisar_painel→ O(1)`

API Retorno de uma análise por DATA:



```python
@router.get( path: "/analises/data/{data_str}", response_model=List[Analise])  eduabreuu
def pega_analises_por_data(data_str: str):

    try:
        data_inicio = datetime.strptime(data_str,  format: "%Y-%m-%d")
    except ValueError:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Formato de data inválido. Use AAAA-MM-DD."
        )

    data_fim = data_inicio + timedelta(days=1)

    query = {
        "timestamp": {
            "$gte": data_inicio,
            "$lt": data_fim
        }
    }

    documentos = list(collection.find(query).limit(1))


    if not documentos:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail=f"Nenhuma análise encontrada para a data {data_str}"
        )

    return documentos
```

O método GET recebe uma data em formato (YYYY-MM-DD) no body da requisição e cria uma query para buscar no banco uma análise de imagem com a mesma data passada na requisição.

`Endpoint Get("/analises/data/{data_str}") → O(log n)`

API Retorno das análises:

O método GET retorna todas as análises do banco em formato JSON.

`Endpoint Get("/analises") → O(n)`

```python
@router.get( path: "/analises", response_model=List[Analise])  eduabreuu
async def pegaAnalises():
    documentos = list(collection.find({})) # Converte o cursor para uma lista
    return documentos
```

```python
def recebeAnaliseData():  3 usages
    dataUsuario = data()
    url = "http://127.0.0.1:8000/challenge/analises/data/" + dataUsuario

    headers = {
        "Authorization": f"Bearer {bearer}"
    }
    response = requests.get(url,headers=headers)
    response.raise_for_status()

    analises = response.json()
    analises = analises[0]

    objeto_saida = analises["saida"]
    analiseResp = objeto_saida["analise"]

    i = 1

    generate_speech(analiseResp, i)
```

```python
def generate_speech(text,i):  4 usages
  speech_file_path = f"analise{i}.mp3"

  with client.audio.speech.with_streaming_response.create(
      model="gpt-4o-mini-tts",
      voice="coral",
      input=text,
      instructions="Pronunciation: Clear, precise: Ensures clarity, especially with key details.",
  ) as response:
      response.stream_to_file(speech_file_path)
      path = "C:/PythonProject/challenge_goodwe/Assistant/" + speech_file_path
      playsound(path)
```

API retorno da análise e assistente de voz:

`Requisição da análise por data → O(log n)`

`generate_speech → O(n)`

A função recebeAnaliseData() recebe uma data, concatena com a URL e retorna uma análise. Em seguida, o código acessa o campo analise da resposta e armazena seu valor em uma variável, que é passada para a função generate_speech. Essa função utiliza a API da OpenAI para transformar o texto em áudio e, por fim, o arquivo gerado é reproduzido.

Resumo da análise

Algoritmo	Complexidade
`Endpoint Post:(“/analisar”)`	`O(1)`
`Função analisar_painel`	`O(1)`
`Endpoint Get(“/analises/data/{data_str}”)`	`O(log n)`
`Endpoint Get(“/analises”`	`O(n)`
`Requisição da análise por data`	`O(log n)`
`generate_speech`	`O(n)`

Conclusão

O código do nosso grupo aparenta ser bem escalável, porém alguns algoritmos merecem atenção. Um exemplo é o método GET("/analises"), cuja complexidade é O(n). Caso o sistema acumule centenas ou milhares de objetos, isso pode gerar problemas de desempenho ao carregar e transformar os registros em uma lista contendo imagens e seus respectivos dados associados. Uma solução possível seria criar uma coluna que indique se o dado está ativo ou não e aplicar uma política de exclusão lógica, removendo ou arquivando análises com mais de 6 meses.

Outro ponto que merece atenção está nos algoritmos que utilizam a API da OpenAI, como a função generate_speech. Apesar de sua complexidade O(n), a dependência de chamadas externas torna o sistema exposto a latências maiores e a custos elevados em cenários de alto volume de requisições ou textos longos.