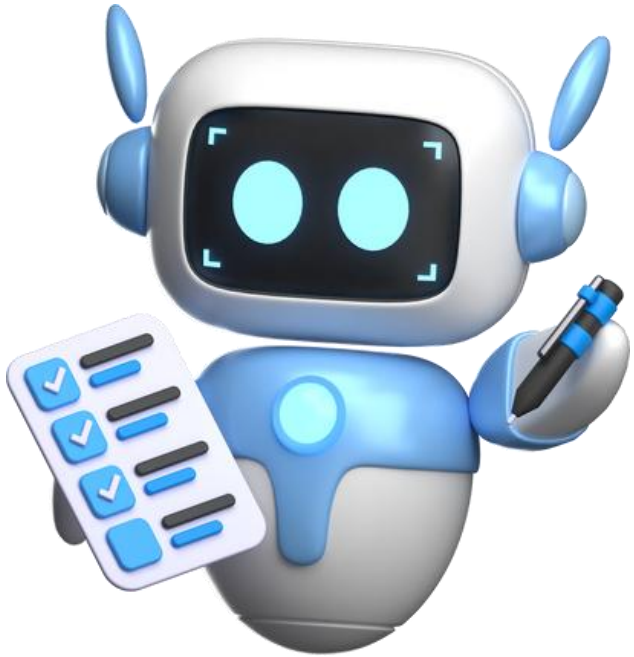
UNIVASSOURAS
CAMPUS UNIVERSITÁRIO DE MARICÁ

Curso Bacharelado em
Engenharia de Software

Inteligência Artificial e Machine Learning

Prof. Tiago Ruiz

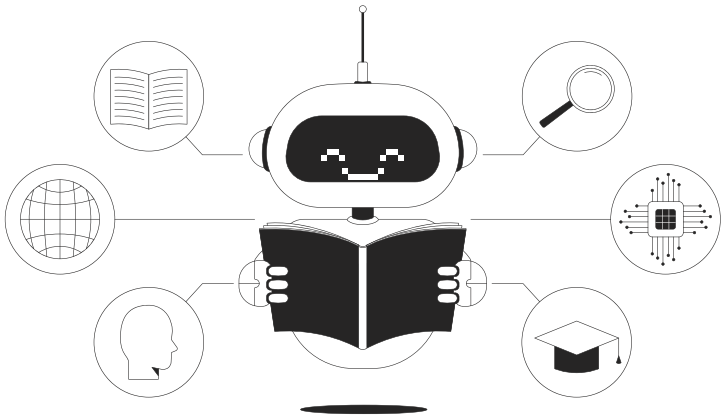
Aula 06/11/2025



Introdução

Integração de APIs de LLMs

Nos últimos anos, os Modelos de Linguagem de Grande Escala (LLMs) transformaram a forma como interagimos com sistemas inteligentes. Ferramentas como o ChatGPT, Gemini, Claude dentre outros... Vêm sendo amplamente utilizadas em aplicações reais de chatbots corporativos a assistentes de código, sistemas de recomendação e análise semântica de texto.



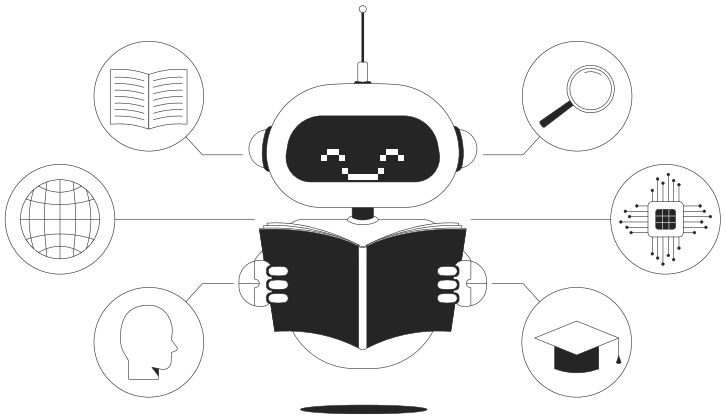
Nesta aula, exploraremos **como integrar e consumir APIs de diferentes LLMs usando Node.js e Python**, entendendo suas semelhanças, diferenças e boas práticas de autenticação, requisição e tratamento de respostas.

Introdução

Integração de APIs de LLMs

O foco será prático:

- Aprender a enviar prompts e interpretar respostas;
- Compreender o fluxo de comunicação entre cliente e modelo;
- **Criar uma base para conectar múltiplos provedores** de IA em uma mesma aplicação;
- E comparar o uso de **SDKs** e requisições **REST** para cada serviço.



Ao final, você será capaz de **integrar diversos LLMs** em seus próprios projetos, combinando poder computacional, flexibilidade e inteligência em soluções reais.

Introdução

Integração de APIs de LLMs

O foco será prático:

- Aprender a enviar prompts e interpretar respostas;
- Compreender o fluxo de comunicação entre cliente e modelo;
- **Criar uma base para conectar múltiplos provedores** de IA em uma mesma aplicação;
- E comparar o uso de **SDKs** e requisições **REST** para cada serviço.

Ao final, você será capaz de **integrar diversos LLMs** em seus próprios projetos, combinando poder computacional, flexibilidade e inteligência em soluções reais.

Estruturas de APIs

Estrutura Geral das APIs de LLMs

A grande maioria das LLM´s seguem um padrão de estrutura para serem implementados parecidos, por exemplo o **OpenAi°** **e a Anthropic Claude** seguem padrões bem parecidos.

O funcionamento básico dos LLMs baseia-se em princípios técnicos e arquiteturas de **pesquisa abertas**, como a arquitetura **Transformer**, que foi introduzida pela Google em um **artigo de pesquisa em 2017** e se tornou a base para a maioria dos modelos modernos.

Permitiu o processamento paralelo das sequências de texto e implementou o mecanismo de atenção, melhorando a compreensão do contexto entre as palavras.

Estruturas de APIs

Estrutura Geral das APIs de LLMs

Métodos:

- POST - Para o envio dos Prompts
- GET - Para listagem dos modelos

Endpoints Padrões:

- **/chat/completions -** Aqui é onde a LLM responde completado requisições ou uma série de requisições.
- **/embeddings -** Para realizar algum tipo de vetorização
- **/models -** Para saber qual tipo de modelos a API tem disponível em determinado momento.
- **Rate Limit -** São as limitações que os modelos possuem de uso, calculados em tokens.

Estruturas de APIs

Estrutura Geral das APIs de LLMs

Chaves de Autenticação

São as formas que as LLM's possuem de autorizar o uso das APIs, elas normalmente ficam linkadas a uma conta criada no site do provedor da API junto a um cadastro de pagamento para serem contabilizados e cobrados os tokens utilizados.

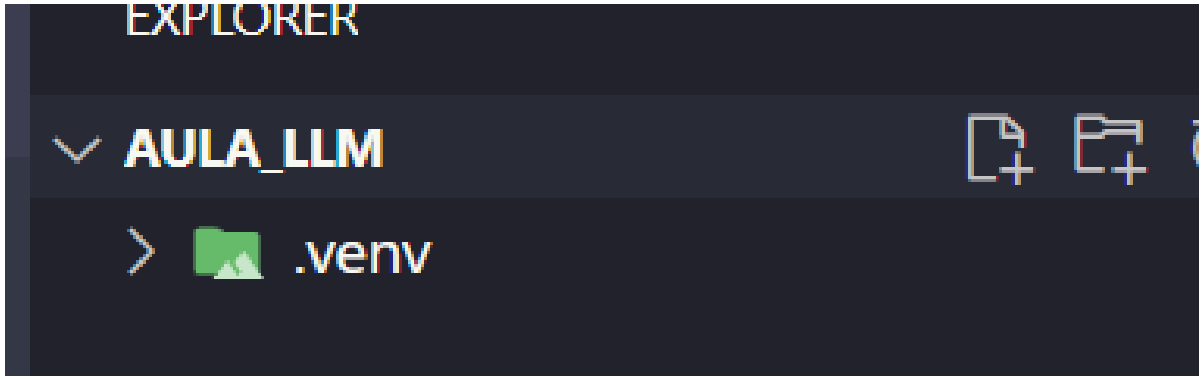
Autenticação (Authentication)
Significa **provar quem você é.**
No contexto de APIs de IA, é o processo pelo qual o sistema confirma que o cliente (você, sua aplicação) tem identidade válida para acessar a API.

Autorização (Authorization)
Significa o que você pode fazer depois de autenticado.
Ou seja, mesmo após a API confirmar quem você é, ela precisa verificar quais permissões você tem.

Ambiente

Configurando o ambiente em Python

Para dar inicio a parte prática, vamos iniciar o nosso ambiente virtual dentro de um diretório:

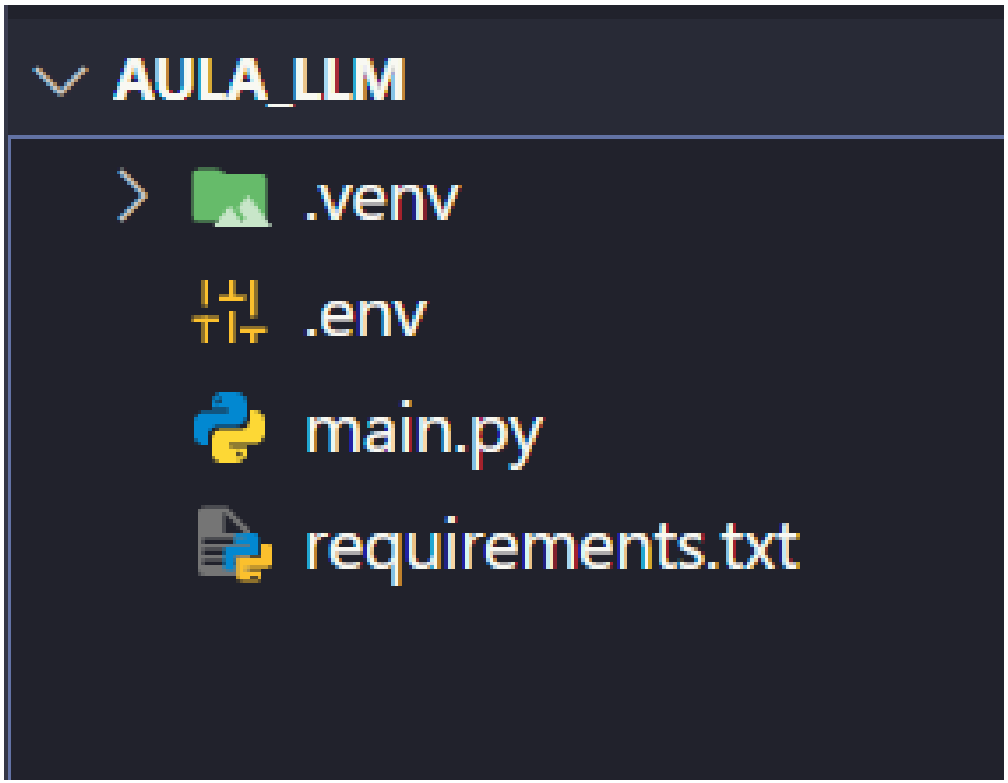


Ambiente

Configurando o ambiente em Python

Crie também os arquivos

- **main.py -** Arquivo principal do projeto
- **.env -** Aonde será criado as chaves das API's
- **requirements.txt -** Lista de dependências para o projeto



Ambiente

Configurando o ambiente em Python

Dentro de requirements.txt vamos solicitar a instalação das dependências:

- **python-dotenv** - Para ler o arquivo .env criado
- **openais, anthropic, google-generativeai, xai-sdk** - são as API's que vamos usar para a aula.
- **groq, ollama** - sao LLMs também porem a ollama roda localmente e o groq usa um conceito diferente, ele da acesso a várias LLM's OpenSource existentes, como Llama 3, Mixtral e outros modelos.

Ambiente

Configurando o ambiente em Python

Agora rode o comando:

pip install -r requirements.txt

Aguarde a instalação de todas as dependências.

```
(.venv) PS C:\Users\Tiago\Desktop\Aula_LLM> pip install -r requirements.txt
Downloading typing_inspection-0.4.2-py3-none-any.whl (14 kB)
Downloading zipp-3.23.0-py3-none-any.whl (10 kB)
Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Downloading google_api_python_client-2.187.0-py3-none-any.whl (14.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.6/14.6 MB 38.8 MB/s  0:00:00
Downloading google_auth_httplib2-0.2.1-py3-none-any.whl (9.5 kB)
Downloading httplib2-0.31.0-py3-none-any.whl (91 kB)
Using cached pyparsing-3.2.5-py3-none-any.whl (113 kB)
```

Ambiente

Configurando o ambiente em Python

Dentro de main.py crie a seguinte estrutura:

```python
import os
from dotenv import load_dotenv

load_dotenv()

import openai
import anthropic
import google.generativeai as gemini
import groq
import xai_sdk
import ollama
```

Ambiente

Configurando o ambiente em Python

Dentro de main.py crie a seguinte estrutura:

Adicione também um print ao final para rodar o código e verificar se está tudo certo.

```python
openai_Cliente = openai.OpenAI(api_key=os.getenv("OPENAI_API_KEY"))
anthropic_client = anthropic.Anthropic(api_key=os.getenv("ANTROPIC_API_KEY"))
xai_cliente = xai_sdk.Client(api_key=os.getenv("XAI_API_KEY"))
gemini.configure(api_key=os.getenv("GOOGLE_API_KEY"))
groq_cliente = groq.Groq(api_keys=os.getenv("GROQ_API_KEY"))
```

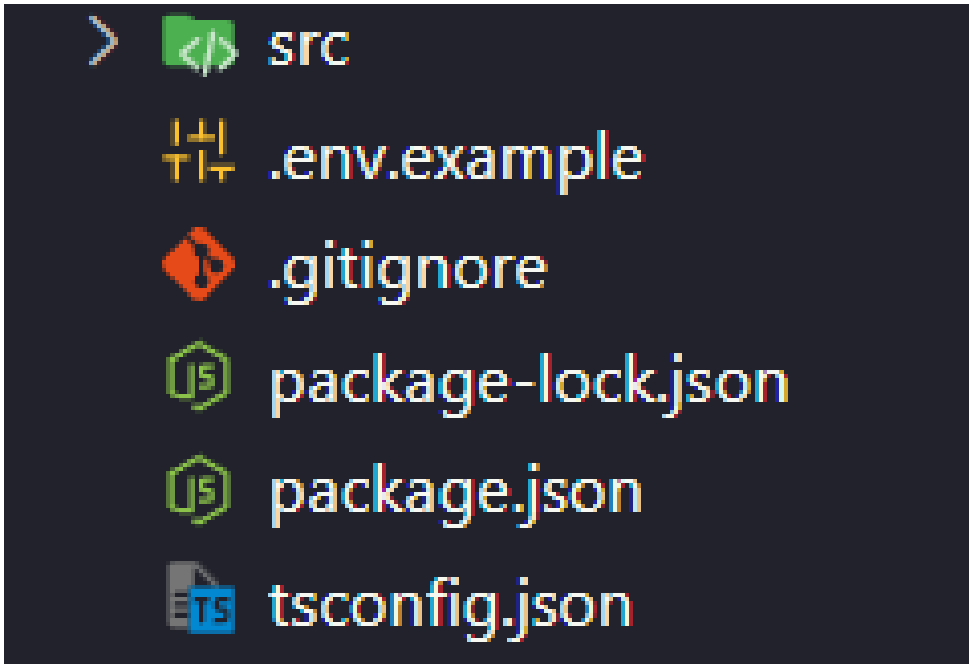
```python
print('Todos as configuracoes foram feitas com sucesso!')
```

Ambiente

Configurando o ambiente em Node

No AVA na aula de hoje tem um arquivo com a base em NODE pré configurado com as configurações necessárias para a implementação das Keys das LLMs que vamos usar.

- **Faça Download do arquivo e extraia em uma Diretório.**
- **Em seguida abra este diretório no VsCode.**



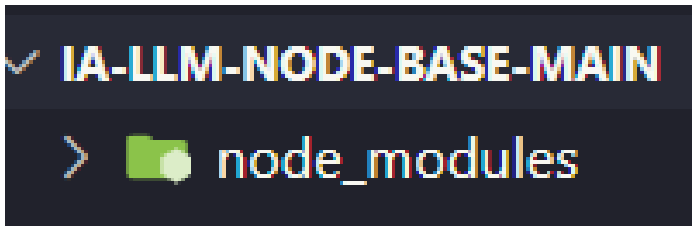
Ambiente

Configurando o ambiente em Node

Rode o comando NPM INSTALL no terminal para instalar as dependências necessárias neste projeto:

```
PS C:\Users\Tiago\Desktop\ia-llm-node-base-main> npm install
```

Será criado as dependências no diretório:

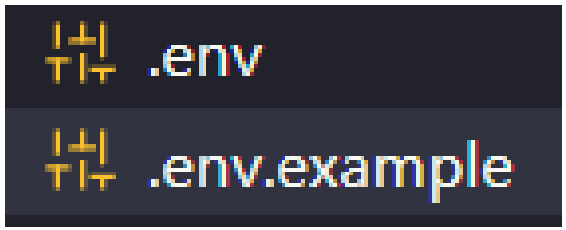


Ambiente

Configurando o ambiente em Node

Rode no terminal o comando “CP .ENV.EXAMPLE .ENV” para criar um arquivo .env com as configurações corretas para alocar as chaves de acesso as LLMs:

```
PS C:\Users\Tiago\Desktop\ia-llm-node-base-main> cp .env.example .env
```



Ambiente

Configurando o ambiente em Node

Este projeto já está com as bibliotecas necessárias para integração com diversos modelos diferentes que temos no mercado tantos comerciais quantos locais também.

server.ts por exemplo contem algumas configurações de rotas e gerenciamento de erros:

```
// Routes
app.use('/api/llm', llmRoutes);
```

E quando iniciado, ele nos retorna no console a porta que esta sendo executada e todos os **endpoints** disponíveis para cada LLM:

```
app.listen(PORT, () => {
    console.log(`🚀 Servidor rodando na porta ${PORT}`);
    console.log(`📍 Endpoints disponíveis:`);
    console.log(`   GET /api/llm/openai`);
    console.log(`   GET /api/llm/anthropic`);
    console.log(`   GET /api/llm/xai`);
    console.log(`   GET /api/llm/google`);
    console.log(`   GET /api/llm/groq`);
    console.log(`   GET /api/llm/ollama`);
});
```

Ambiente

Configurando o ambiente em Node

Dentro do diretório **router no arquivo llm.ts** na rota criada para o serviço da **OpenAI** por exemplo:

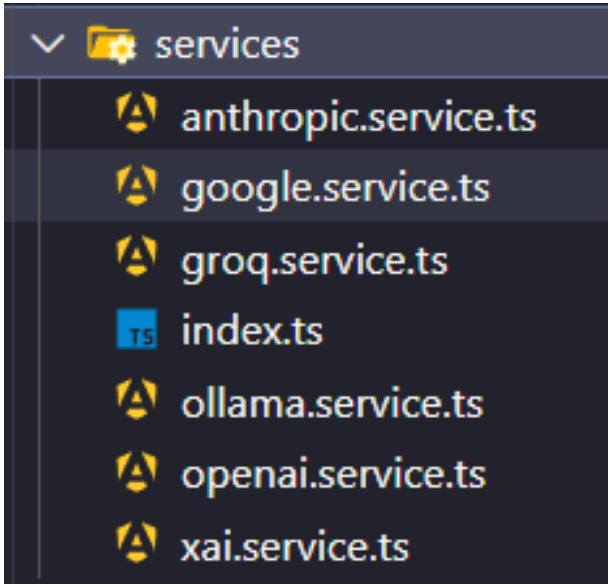
```
// Route para OpenAI
router.get('/openai', async (req, res) => {
    const client = createOpenAIClient();
    res.json({
        provider: 'OpenAI',
        message: 'Cliente OpenAI criado com sucesso',
        client: !!client
    });
});
```

Todas as APIs que foram apresentadas tem 2 formas de serem implementadas, a primeira é requisitando de forma direta na API, e a segunda é usando uma biblioteca para fazer essas requisições, oque facilita um pouco a implementação seguindo um padrão pré-definido.

Ambiente

Configurando o ambiente em Node

Dentro do diretório **services,** temos um service diferente para cada provedor, dentro de cada arquivo contem as configurações das chaves de acesso para cada provedor que a gente precisa, puxando da variável de ambiente de cada provedor.




```ts
import OpenAI from 'openai';

export function createOpenAIClient() {
    if (!process.env.OPENAI_API_KEY) {
        throw new Error('OPENAI_API_KEY não encontrada nas variáveis de ambiente');
    }

    return new OpenAI({
        apiKey: process.env.OPENAI_API_KEY,
    });
}
```

Ambiente

Configurando o ambiente em Node

Ao abrir cada um dos serviços vocês podem notar que as suas configurações são bem parecidas, mudando apenas algumas coisas especificas para alguns provedores, por exemplo da **google** não passamos a chave diretamente como um objeto, mas passamos dentro da própria função:

```
1  import { GoogleGenerativeAI } from '@google/generative-ai';
2
3  export function createGoogleClient() {
4      if (!process.env.GOOGLE_API_KEY) {
5          throw new Error('GOOGLE_API_KEY não encontrada nas variáveis de ambiente');
6      }
7
8      return new GoogleGenerativeAI(process.env.GOOGLE_API_KEY);
9  }
10
```

Ambiente

Configurando o ambiente em Node

O service do **xai** por exemplo, ela ainda não possui oficialmente um SDKey, portanto estamos utilizando o da **Versel** por ser gratuito:

```
1  import { createXai } from '@ai-sdk/xai';
2
3  export function createXAIClient() {
4      if (!process.env.XAI_API_KEY) {
5          throw new Error('XAI_API_KEY não encontrada nas variáveis de ambiente');
6      }
7
8      return createXai({
9          apiKey: process.env.XAI_API_KEY
10     })
11 }
```

PS: A Versel tem SDKey para varios serviços (OpenAI, Anthropic... e varios outros)

Ambiente

Configurando o ambiente em Node

E temos também o **Groq** que na verdade não se trata de um modelo, mas sim de um serviço que permite vocês utilizarem vários modelos dentro de uma biblioteca:

```
import Groq from 'groq-sdk';

export function createGroqClient() {
    if (!process.env.GROQ_API_KEY) {
        throw new Error('GROQ_API_KEY não encontrada nas variáveis de ambiente');
    }

    return new Groq({
        apiKey: process.env.GROQ_API_KEY,
    });
}
```

Observe que a sua configuração é bem parecida com a de qualquer um dos modelos.

Ambiente

Configurando o ambiente em Node

E por fim o **Ollama** que vai nos permitir rodar modelos localmente, e por ser local não necessita de uma chave de acesso, mas sim de um servidor que vai rodar os modelos:

```
1  import { Ollama } from 'ollama';
2
3  export function createOllamaClient() {
4      const baseUrl = process.env.OLLAMA_BASE_URL || 'http://localhost:11434';
5
6      return new Ollama({
7          host: baseUrl,
8      });
9  }
```

Ambiente

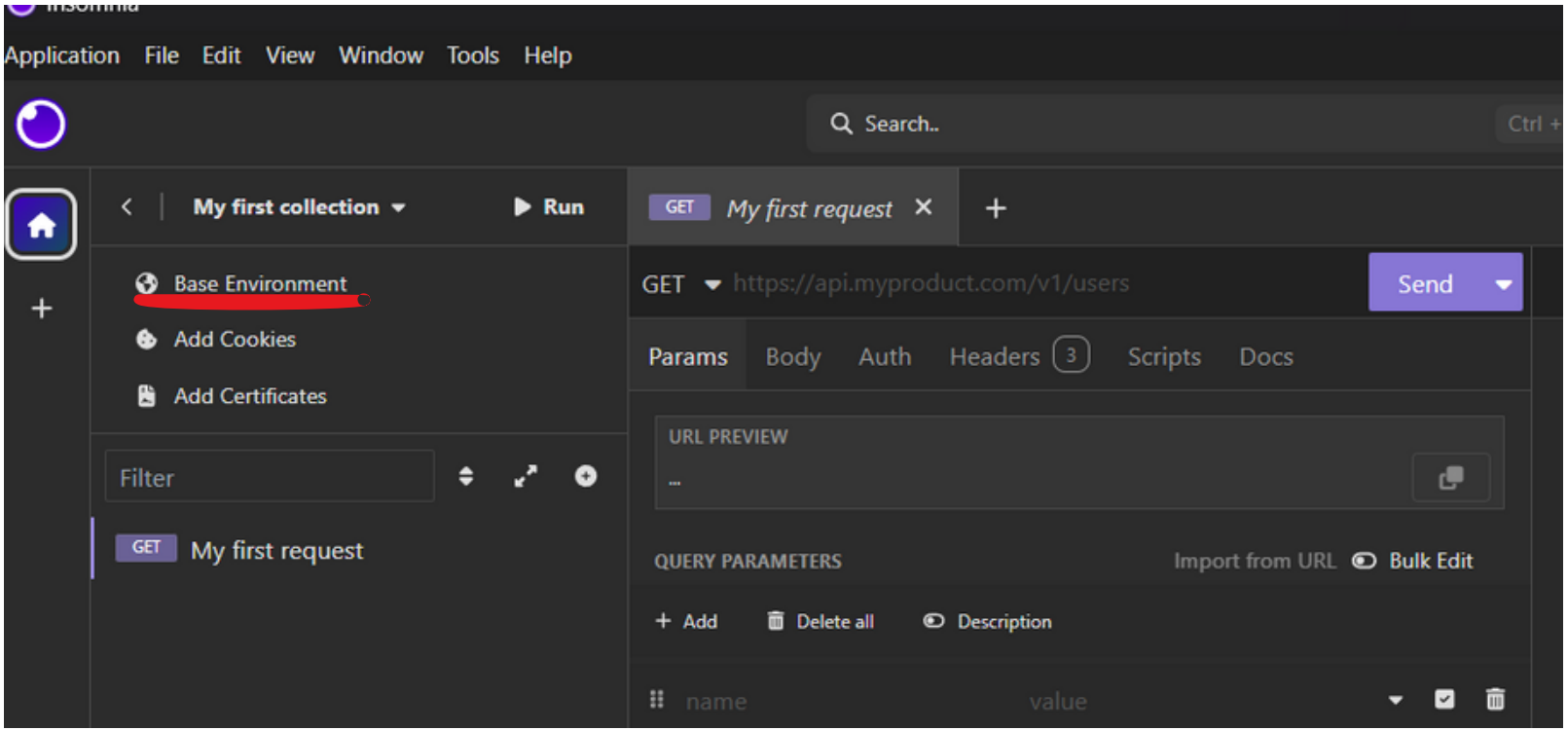
Configurando o ambiente em Node

Para testar vamos então rodar o servidor com o comando

npm run dev

```
S C:\Users\Tiago\Desktop\ia-llm-node-base-main> npm run dev
```

Em seguida vamos abrir o insomnia para verificar se estes EndPoints estão funcionando:

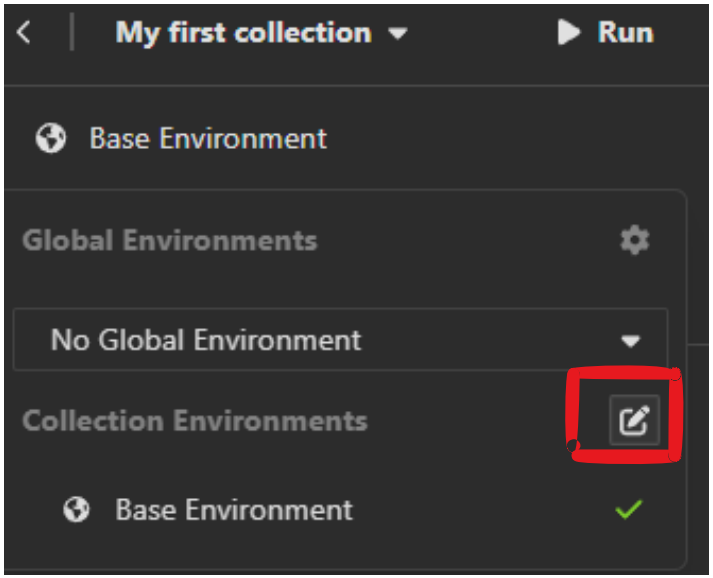


Em Base Environment vamos configurar a base para testar um GET por exemplo

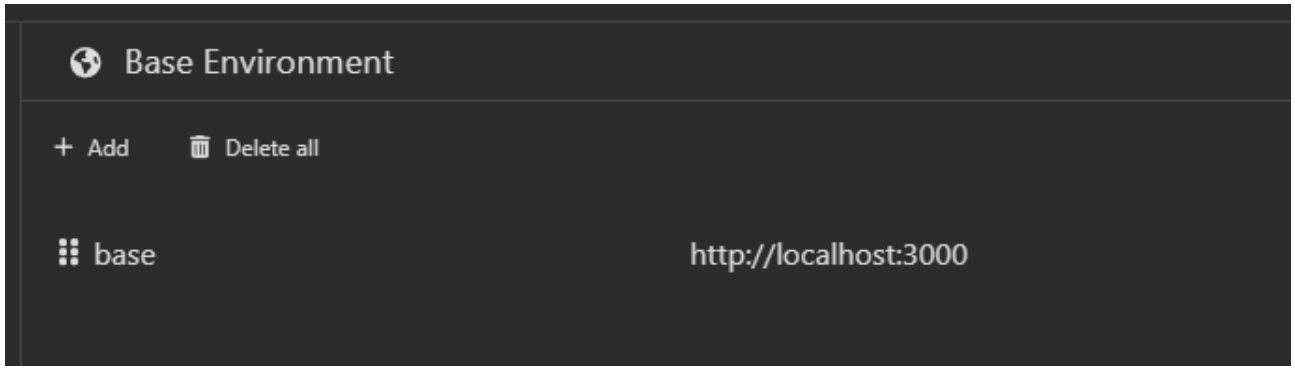
Ambiente

Configurando o ambiente em Node

Clique em editar:



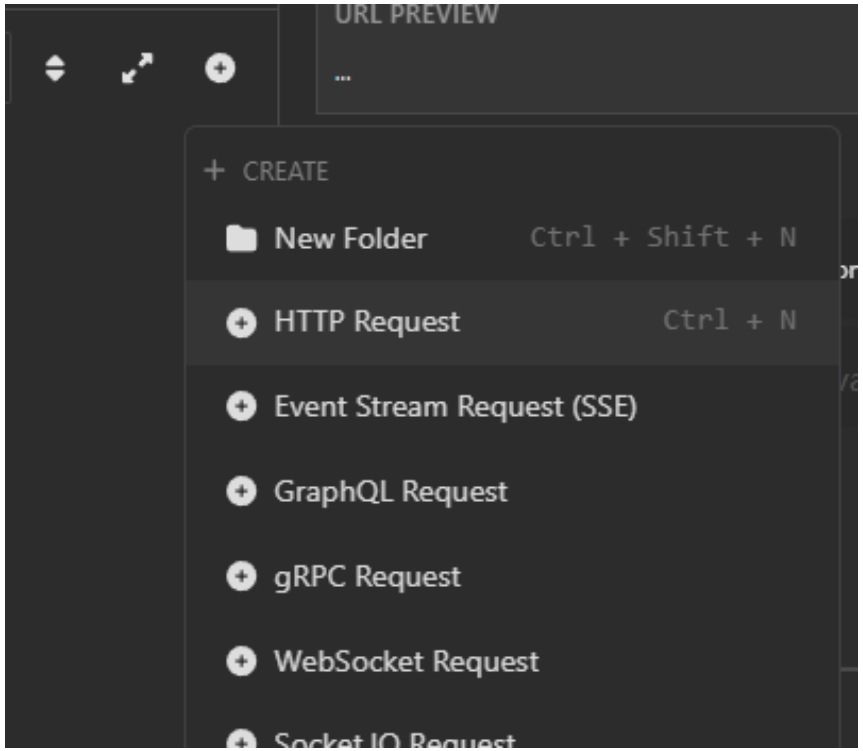
Em seguida configure a base para o http://localhost:3000:



Ambiente

Configurando o ambiente em Node

Na sequencia configure uma requisição **HTTP Request:**

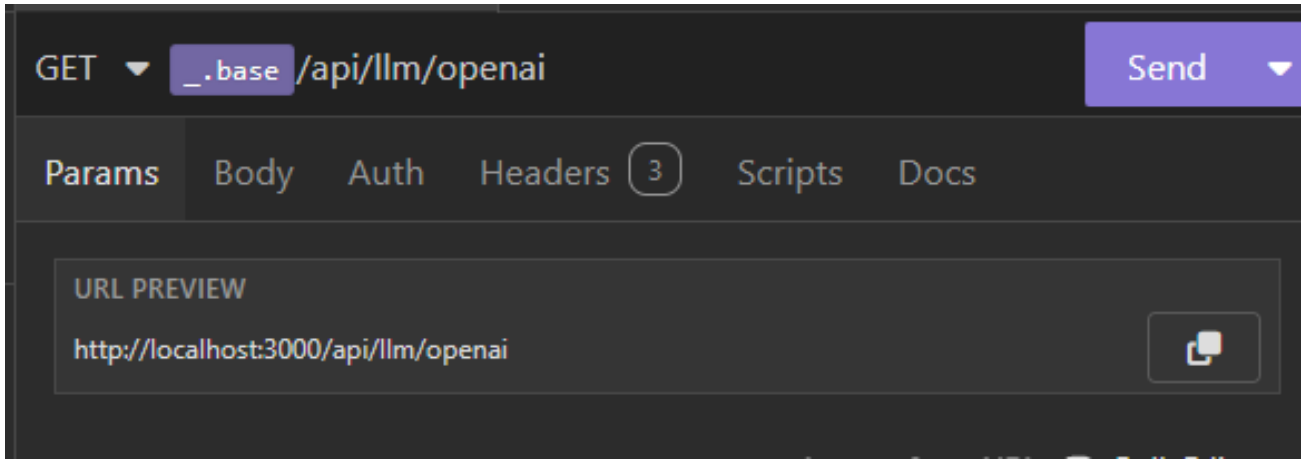


Renomeie este GET para a rota que criamos:

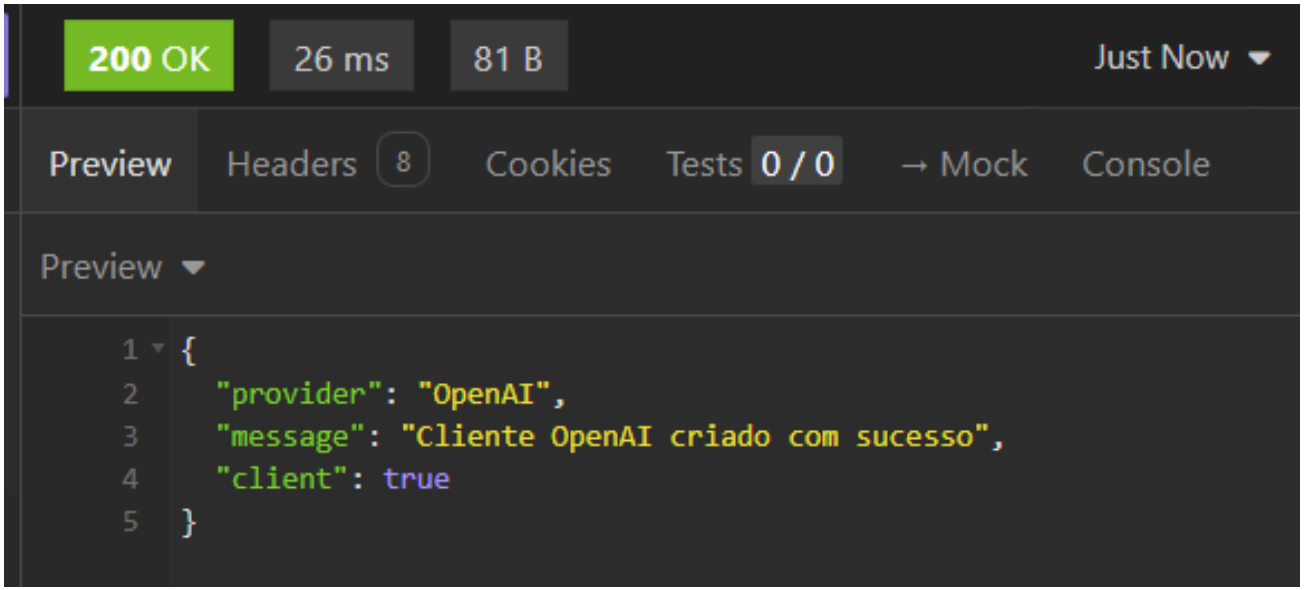
Ambiente

Configurando o ambiente em Node

Agora execute a requisição GET no caminho:



Se tudo estiver certo, vai rodar a requisição sem problemas:



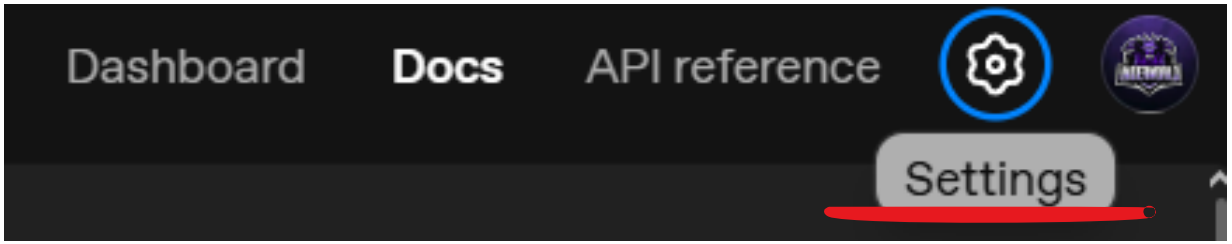
Python

Python: Primeiros passos com a API da OpenAI

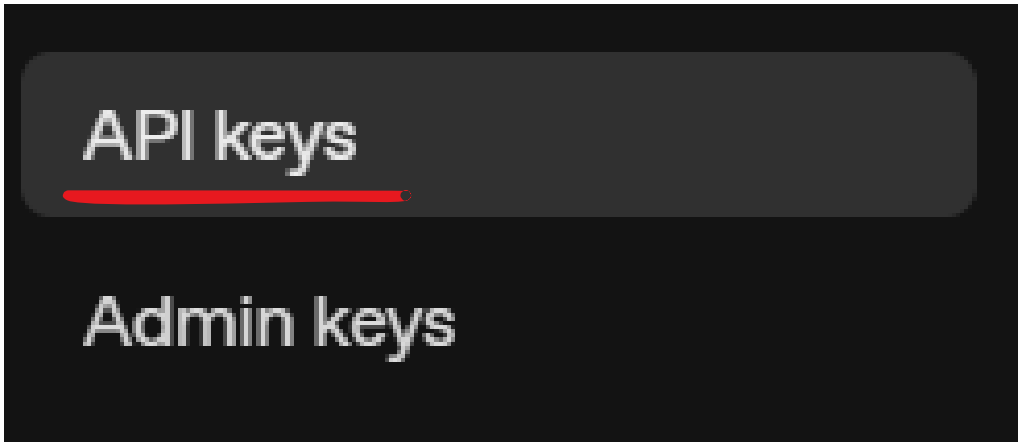
Para buscar uma chave Key da **OpenAI** acesse:

https://platform.openai.com/

Acesse a aba de configurações:



Acessando a aba API Keys na side bar você terá acesso ao ambiente que você cria as chaves:



Python

Python: Primeiros passos com a API da OpenAI

Como estas chaves dependem de um determinado valor para ser usada, estou disponibilizando para aula de hoje uma chave OpenAI para o projeto com validade até o final da aula:

Baixe no AVA o arquivo keys.txt

Python

Python: Primeiros passos com a API da OpenAI

No nosso projeto vamos comentar as outras rotas que criamos e deixar apenas a da OpenAI funcionando:

```python
import os
from dotenv import load_dotenv

load_dotenv()

import openai
import anthropic
import google.generativeai as gemini
import groq
import xai_sdk
import ollama

openai_Cliente = openai.OpenAI(api_key=os.getenv("OPENAI_API_KEY"))
# anthropic_client = anthropic.Anthropic(api_key=os.getenv("ANTROPIC_API_KEY"))
# # xai_cliente = xai_sdk.Client(api_key=os.getenv("XAI_API_KEY"))
# gemini.configure(api_key=os.getenv("GOOGLE_API_KEY"))
# groq_cliente = groq.Groq(api_key=os.getenv("GROQ_API_KEY"))

print('Todos as configuracoes foram feitas com sucesso!')
```

Python

Python: Primeiros passos com a API da OpenAI

Em seguida cole a chave KEY da OpenAI no arquivo de .env (Variáveis de ambiente)

```
1  OPENAI_API_KEY = 'sk-proj-36nNJ78MEUI2r1ErP0Zh-O5Hocy0aaycNmsrPx3qB6RhNs
2  ANTROPIC_API_KEY = ''
3  XAI_API_KEY = ''
4  GOOGLE_API_KEY = ''
5  GROQ_API_KEY = ''
```

Python

Python: Primeiros passos com a API da OpenAI

Na sequencia vamos criar as configurações das requisições:

- **model** = é responsável por apontar o modelo que queremos utilizar no projeto
- **input** = (pode ser uma frase ou um array)
  - **Sem Array** = Uma mensagem direta do usuário
  - **Com Array** = usamos orientações para o comportamento das respostas que ela lhe dará ao usuário

- **reasoning =** É aonde determinamos por exemplo o esforço do modelo, podendo ser **low, medium ou high.**

```python
response = openai_Cliente.responses.create(
    model="gpt-5-nano",
    reasoning= {
        "effort" : "low"
    },
    input="Qual o Sentido da Vida?"
)

print(response.output_text)
```

Python

Python: Primeiros passos com a API da OpenAI

Vamos modificar então o **input** para orientar o modo de resposta do nosso modelo, primeiramente vamos enviar uma config para o sistema, em seguida uma pergunta direta:

```
input=[
    {
        "role": "system",
        "content": [
            {
                "type": "input_text",
                "text": "Seja Direto e Consiso, use uma unica frase nas respostas."
            }
        ]
    },
```

Python

Python: Primeiros passos com a API da OpenAI

Em seguida passamos nossa mensagem nesta configuração:

```
        {
            "role": "user",
            "content": [
                {
                    "type": "input_text",
                    "text": "Qual o Sentido da Vida?"
                 }
            ]
        }
    ]
)
```

Python

Python: Primeiros passos com a API da OpenAI

Caso você queira ver como funciona as respostas, basta criar o print diretamente no **response** que criamos, temos então todas as informações destas requisições feitas:

```
1  print(response)
```

```
(.venv) PS C:\Users\Tiago\Desktop\Python_Aula_LLM> py main.py
Response(id='resp_028ac146e085fe0600690d1a944cb4819d87bf861a1b9dc295', created_at=1762466452.0, error=None, incomplete_details=None, instructions=None, metadata
={}, model='gpt-5-nano-2025-08-07', object='response', output=[ResponseReasoningItem(id='rs_028ac146e085fe0600690d1a94c64c819dafc2b40eb1e6814d', summary=[], typ
e='reasoning', content=None, encrypted_content=None, status=None), ResponseOutputMessage(id='msg_028ac146e085fe0600690d1a95aa34819d96bcbd290146118c', content=[R
esponseOutputText(annotations=[], text='O sentido da vida é construído por cada pessoa por meio de relações significativas, propósito próprio, aprendizado e con
tribuições que gerem bem-estar e amor.', type='output_text', logprobs=[])], role='assistant', status='completed', type='message')], parallel_tool_calls=True, te
mperature=1.0, tool_choice='auto', tools=[], top_p=1.0, background=False, conversation=None, max_output_tokens=None, max_tool_calls=None, previous_response_id=N
one, prompt=None, prompt_cache_key=None, reasoning=Reasoning(effort='low', generate_summary=None, summary=None), safety_identifier=None, service_tier='default',
 status='completed', text=ResponseTextConfig(format=ResponseFormatText(type='text'), verbosity='medium'), top_logprobs=0, truncation='disabled', usage=ResponseU
sage(input_tokens=32, input_tokens_details=InputTokensDetails(cached_tokens=0), output_tokens=102, output_tokens_details=OutputTokensDetails(reasoning_tokens=64
), total_tokens=134), user=None, billing={'payer': 'openai'}, prompt_cache_retention=None, store=True)
```

UNIVASSOURAS
CAMPUS UNIVERSITÁRIO DE MARICÁ

Muito Obrigado!

Fim da aula de Hoje

Contatos:
@TiagoBombista
(21)96696-2564