

Machine Learning Engineer

Python para ML e IA

Desenvolvimento de API com Flask I

Leonardo Pena

/ Seja muito bem vindo



OBJETIVO

Começar o desenvolvimento de uma API Flask



CONTEXTO

Abordaremos teoria + prática de forma incremental



EXPECTATIVA

Ao final da aula, teremos um projeto Flask completo



ESTRUTURAÇÃO

Introdução, conceitos e casos de uso

Objetivo dessa primeira parte



Passo 1

Instalar e configurar o Flask no ambiente local



Passo 2

Criar a estrutura inicial do projeto



Passo 3

Entender como definir rotas básicas (GET/POST/PUT/DELETE)



Passo 4

Construir um CRUD em memória para treinar conceitos



Passo 5

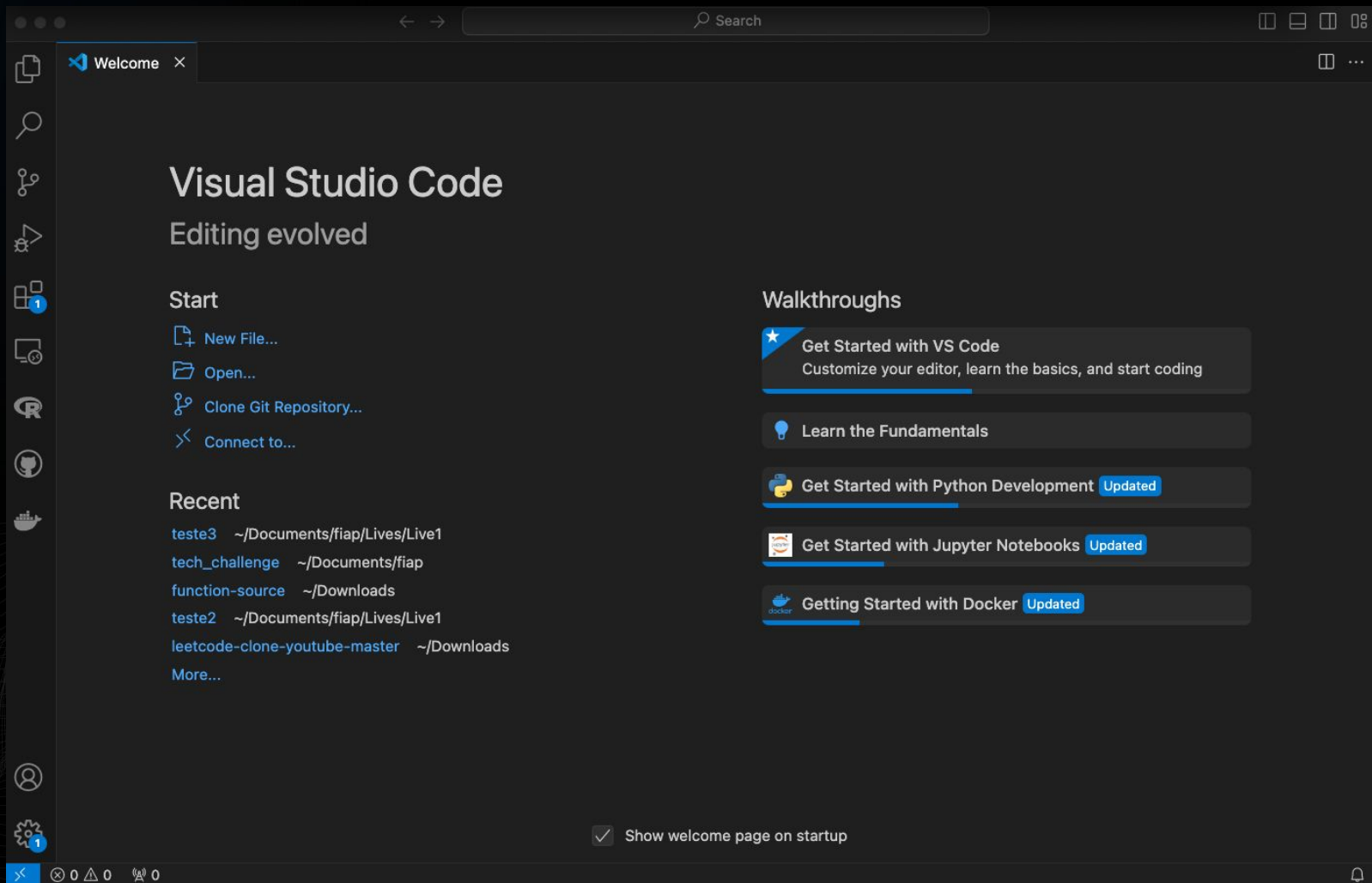
Preparar terreno para adicionar segurança e outras extensões



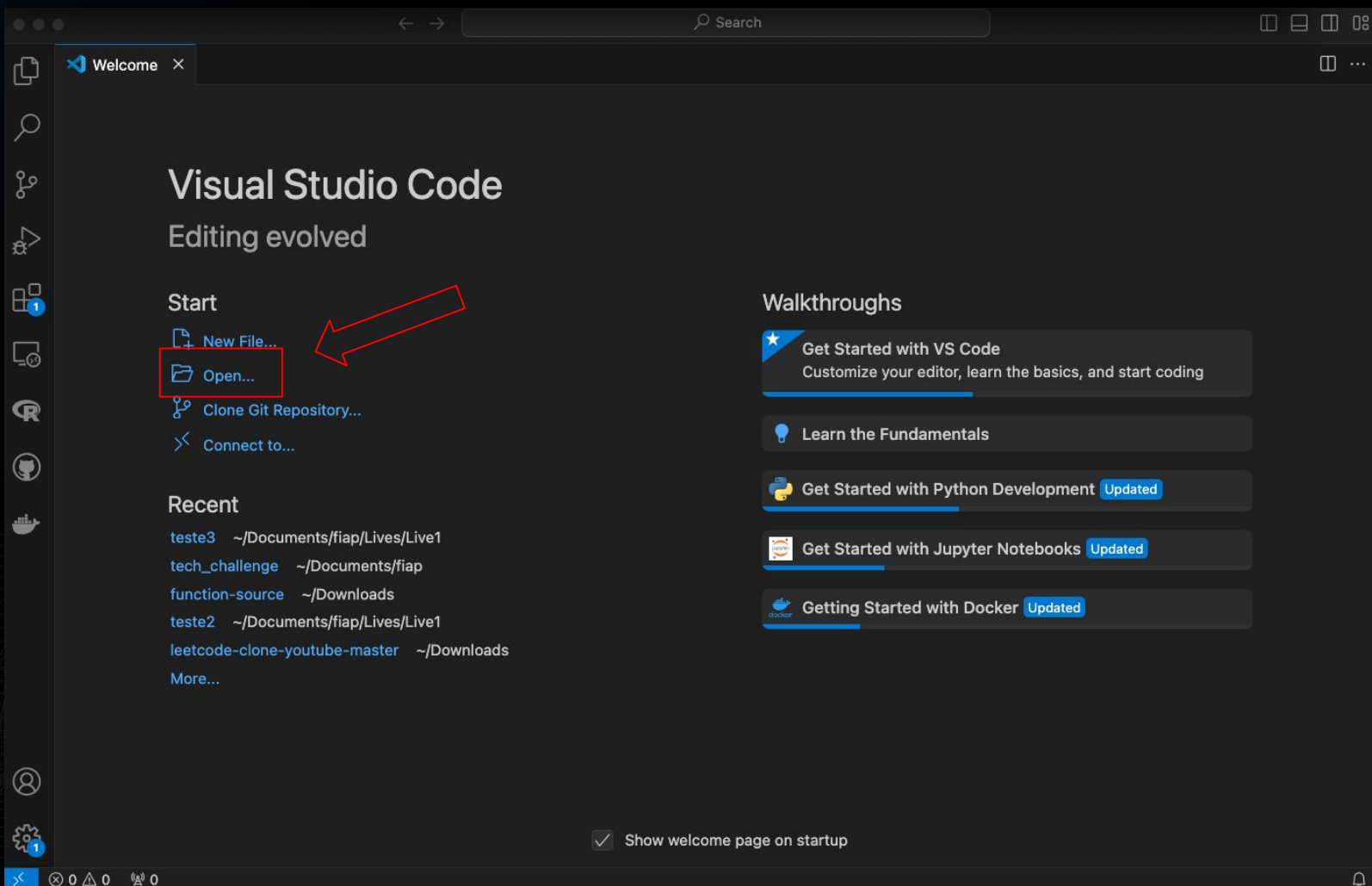
Configurando o ambiente



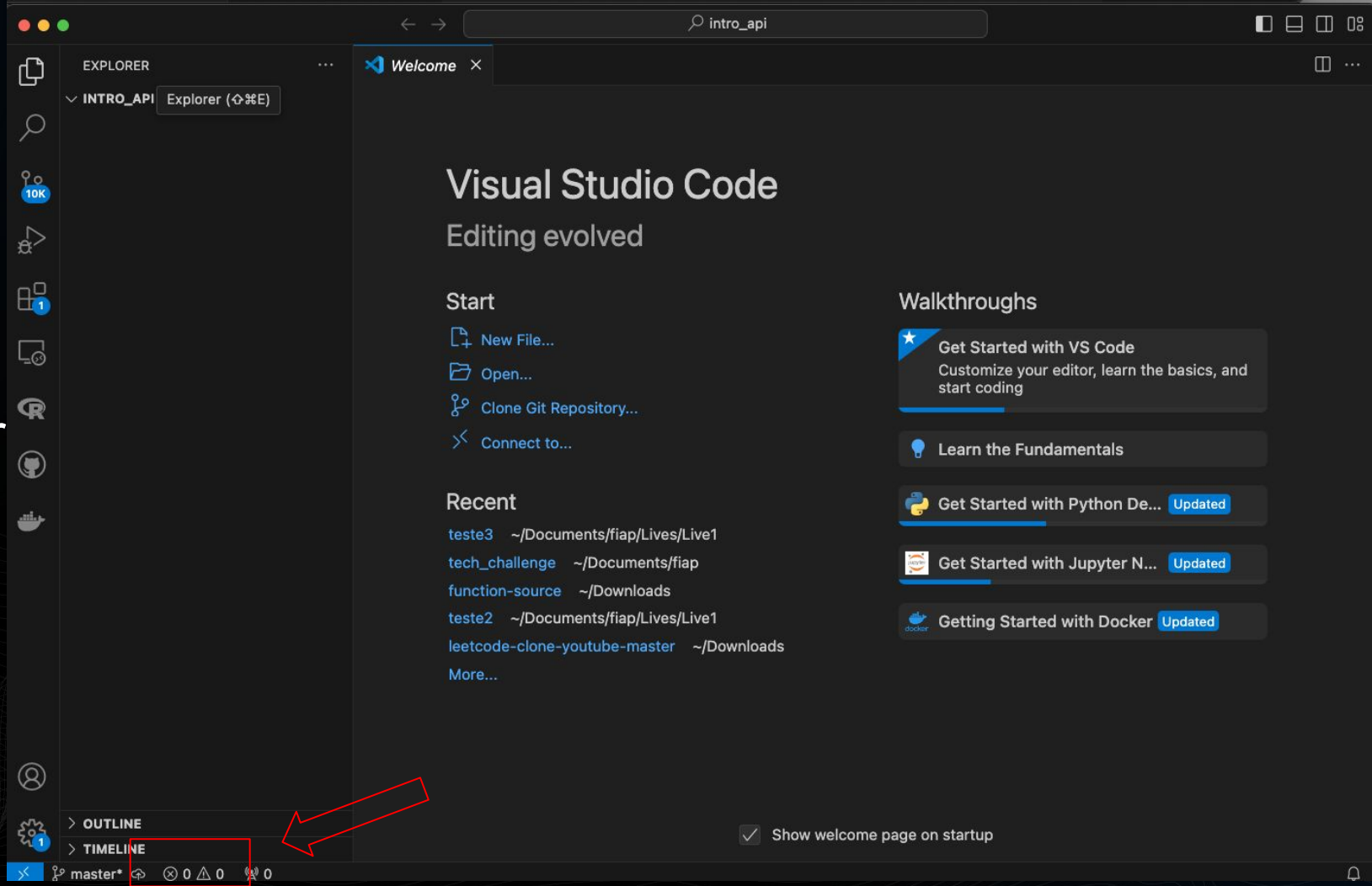
Vamos
ver
como é
o
VSCode



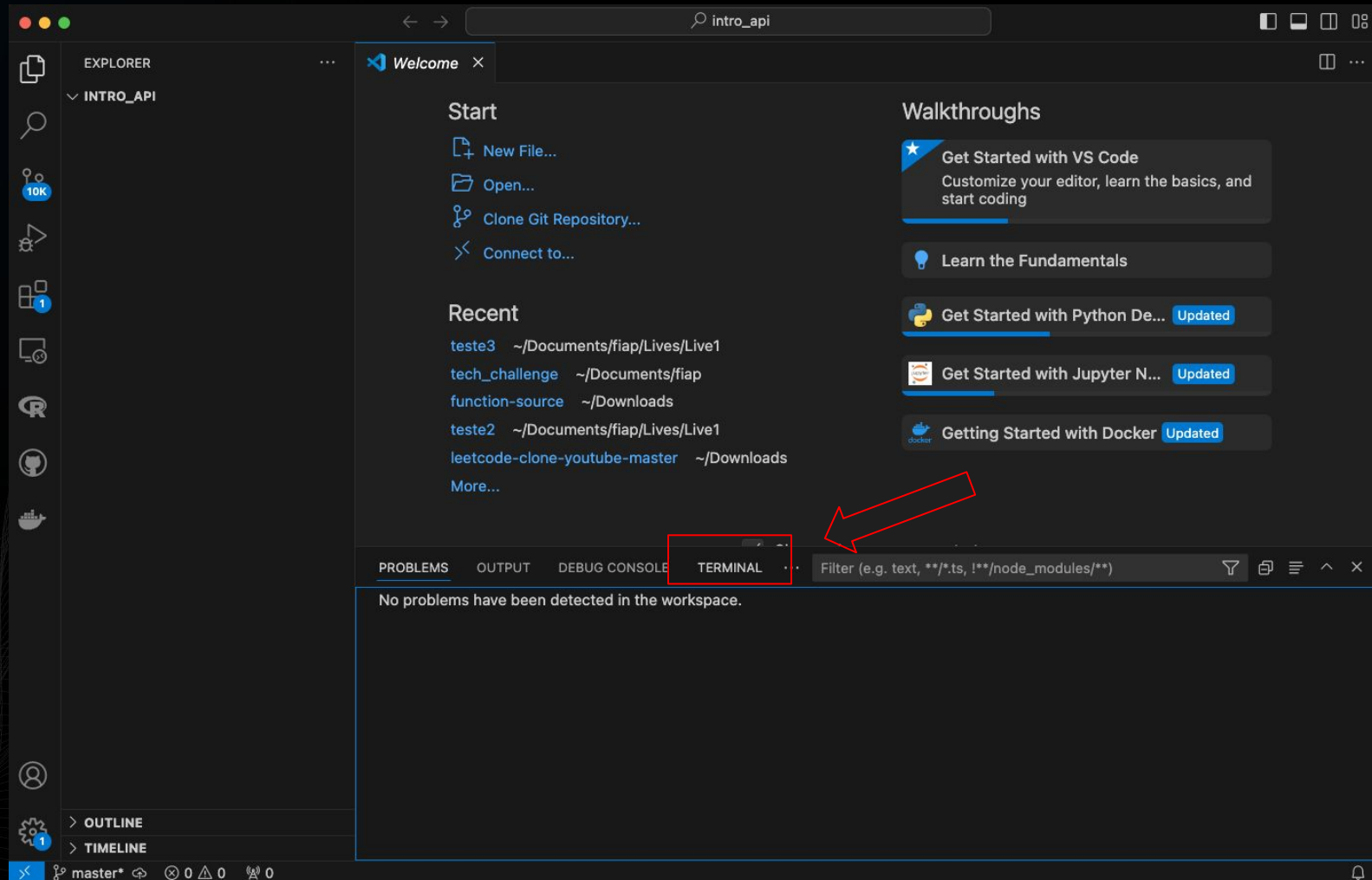
/
Abra
uma
pasta
pro seu
projeto



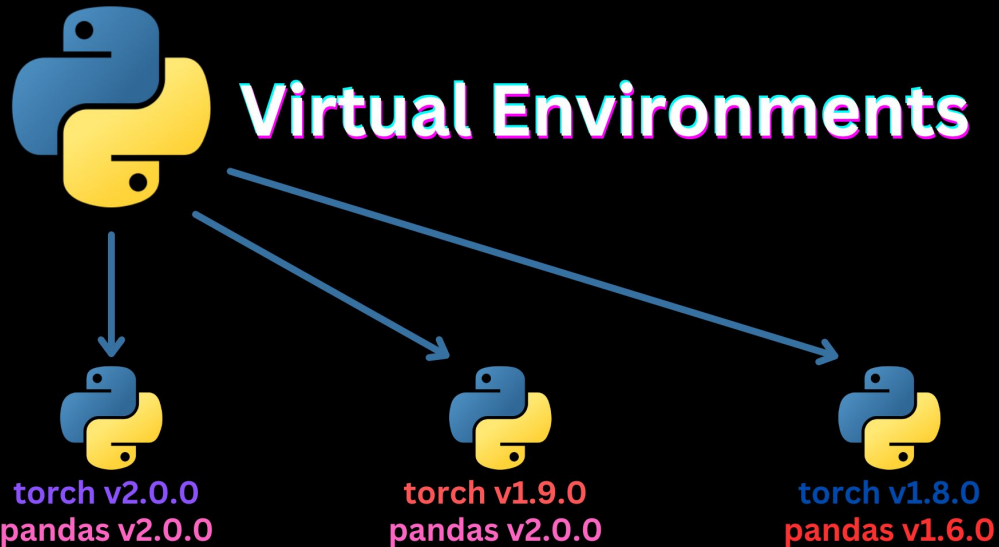
Vamos
abrir o
terminal
para
trabalhar



/
Vamos
abrir o
terminal
para
trabalhar

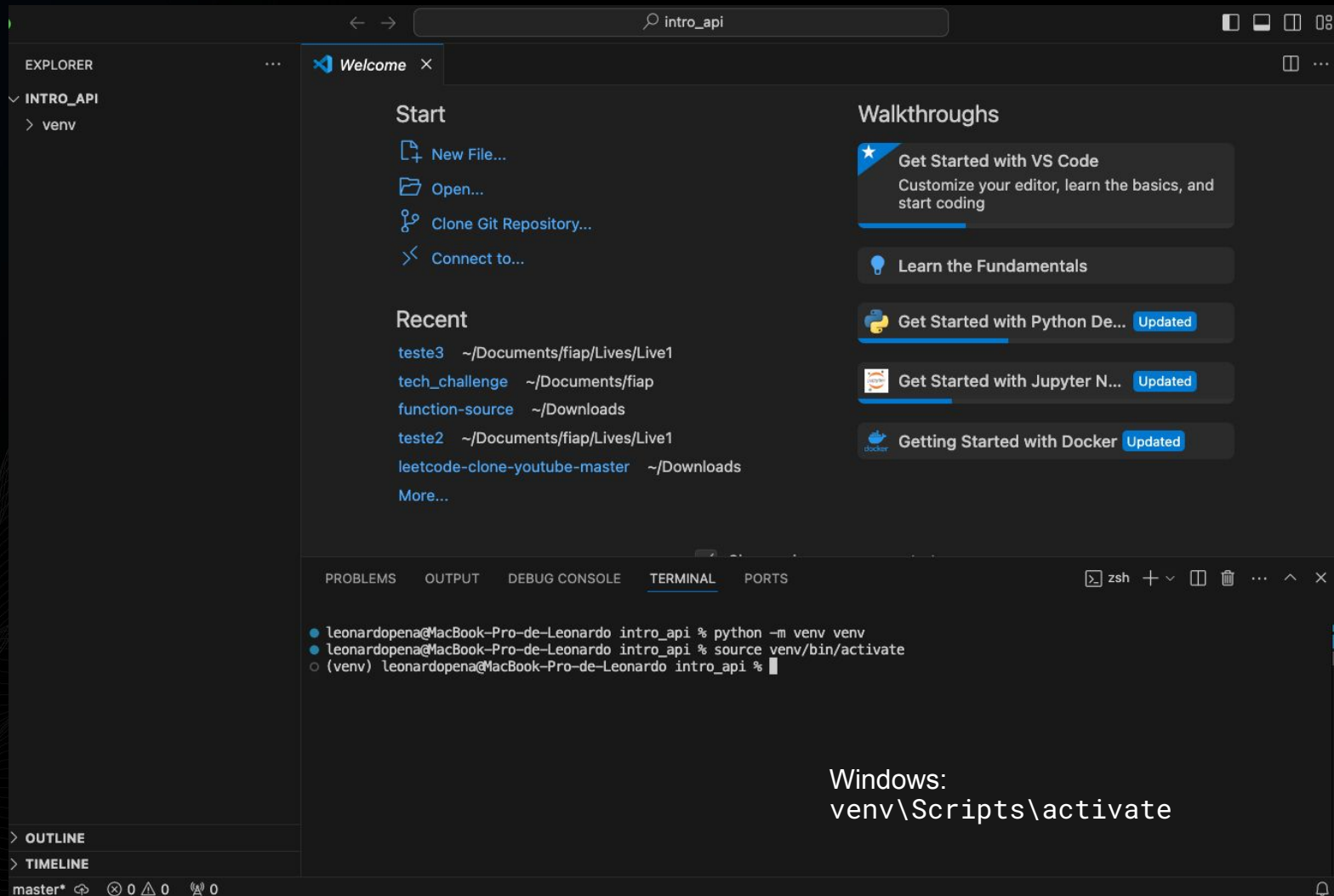


/ Ambiente Virtual



Um ambiente virtual em Python é um ambiente isolado que permite instalar pacotes e dependências específicos para um projeto, sem interferir nas configurações globais do sistema ou de outros projetos.

Criando um ambiente virtual



/

Agora precisamos instalar as bibliotecas necessárias

```
• leonardopena@MacBook-Pro-de-Leonardo aula2-new % python -m venv venv
• leonardopena@MacBook-Pro-de-Leonardo aula2-new % source venv/bin/activate
• (venv) leonardopena@MacBook-Pro-de-Leonardo aula2-new % pip install flask
Collecting flask
  Using cached flask-3.1.0-py3-none-any.whl.metadata (2.7 kB)
Collecting Werkzeug>=3.1 (from flask)
  Using cached werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Downloading jinja2-3.1.5-py3-none-any.whl.metadata (2.6 kB)
```

/

Você pode checar quais bibliotecas estão instaladas no ambiente virtual com pip freeze

```
(venv) leonardopena@MacBook-Pro-de-Leonardo aula2-new % pip freeze
blinker==1.9.0
click==8.1.8
Flask==3.1.0
itsdangerous==2.2.0
Jinja2==3.1.5
MarkupSafe==3.0.2
Werkzeug==3.1.3
```



Introdução ao Flask

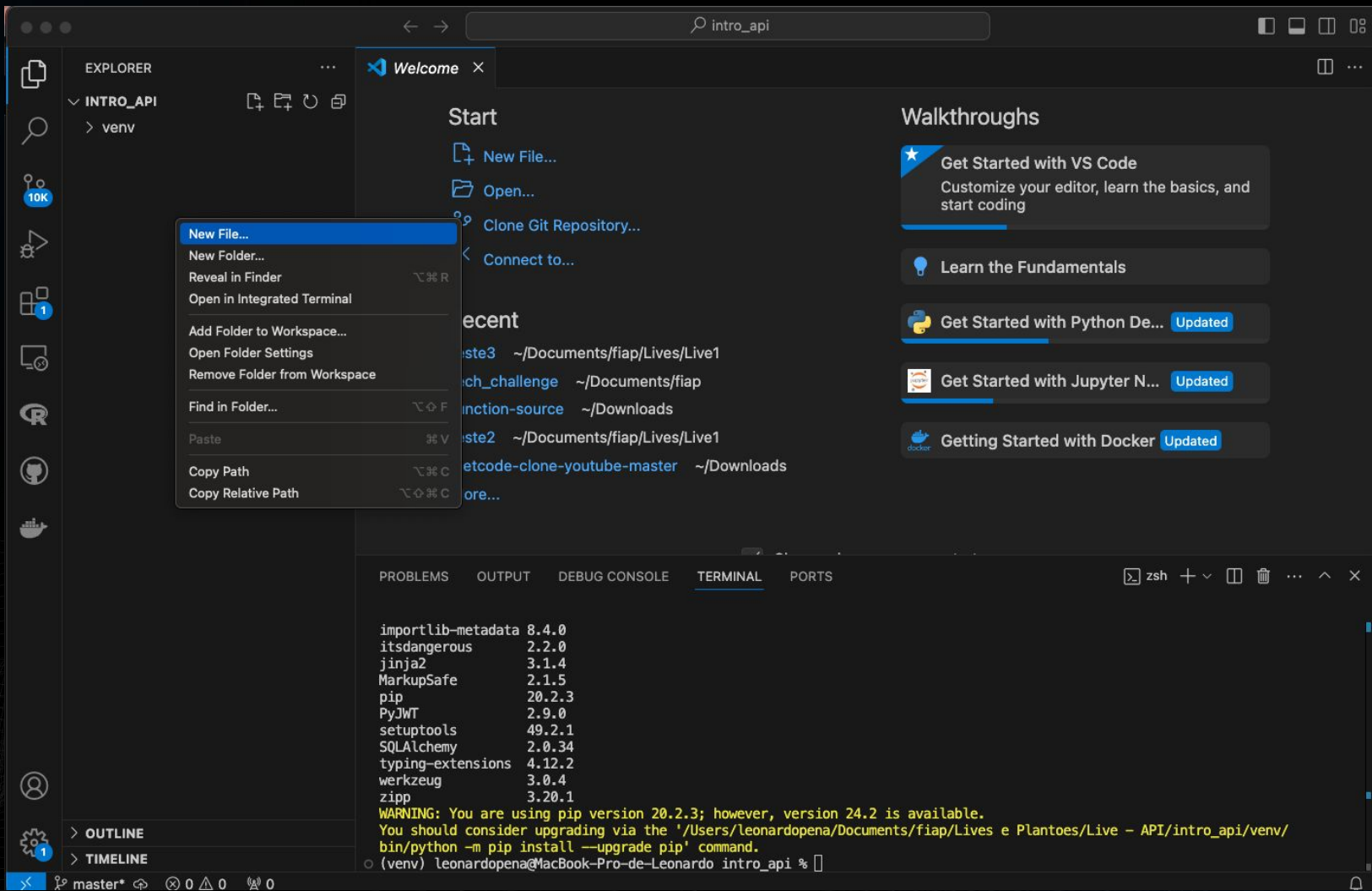




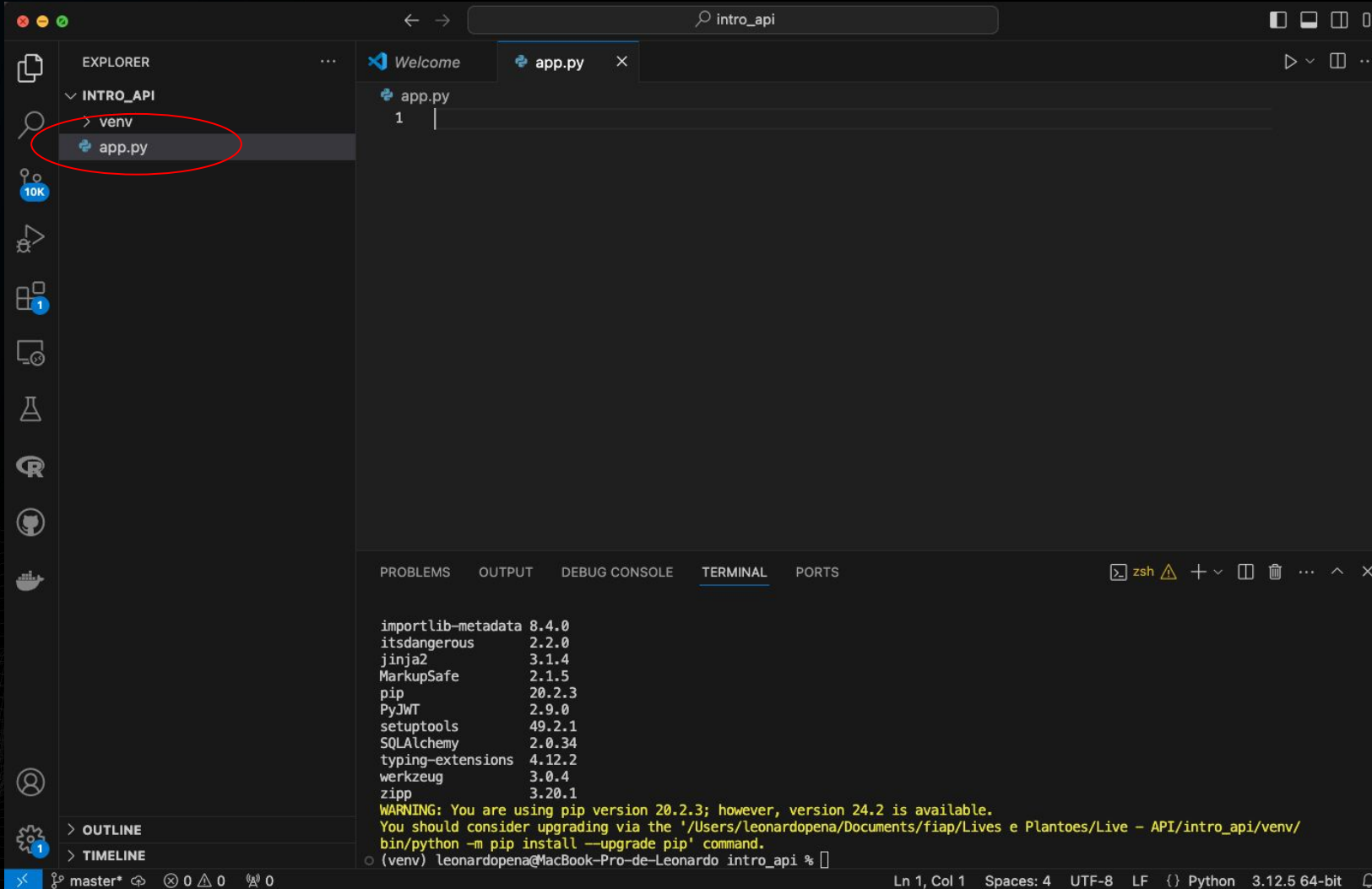
/ Flask

Flask é um microframework web em Python, minimalista e flexível, ideal para criar APIs e aplicações web **de forma rápida e simples**, com suporte a extensões para funcionalidades adicionais.

Primeiro passo é criar o código principal: app.py



Primeiro passo é criar o código principal: app.py



1. Import do Flask e criação da instância
2. Definição da rota / (raiz)
3. Retorno simples de string "Hello, Flask!"
4. `if __name__ == '__main__':` para rodar localmente
5. `debug=True` facilita testes e mostra erros no navegador


```
app.py
app.py > ...
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def home():
7      return "Hello, Flask!"
8
9  if __name__ == '__main__':
10     app.run(debug=True)
11
```

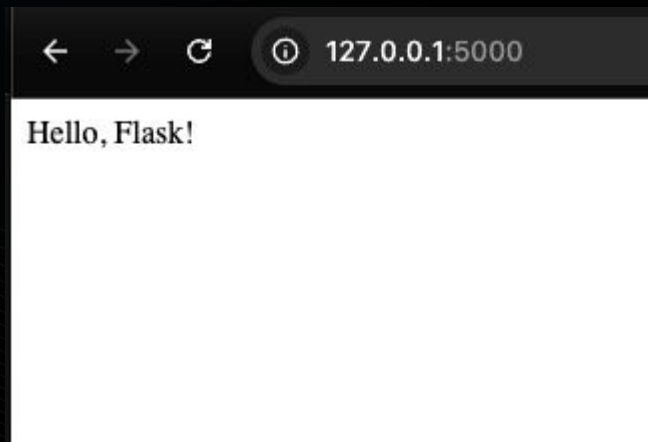
The background of the slide is a dark navy blue. It features a series of thin, light blue wavy lines that create a sense of depth and movement, resembling a stylized landscape or a digital wave pattern. These lines are more pronounced in the lower half of the image and fade into the dark background towards the top.

/ Vamos testar!

1. Salve a aplicação (ctrl+s)
2. No terminal, rode: python app.py

```
> ▾ TERMINAL
o (venv) leonardopena@MacBook-Pro-de-Leonardo aula2-new % python app.py
  * Serving Flask app 'app'
  * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
  * Running on http://127.0.0.1:5000
Press CTRL+C to quit
  * Restarting with stat
  * Debugger is active!
  * Debugger PIN: 877-383-895
□
```

- 
1. Salve a aplicação (ctrl+s)
 2. No terminal, rode: `python app.py`
 3. Acesse a url que aparece na saída (copie e cole no navegador)



Obs: mantenha o vscode aberto, para mostrar os logs e acompanhar as requisições

Parabéns!! Você acaba de
criar sua primeira aplicação
em Flask



CRUD Básico com Flask



/ CRUD



Create



Read



Update



Delete

C R U D

Em Flask, CRUD se refere à criação de rotas para realizar operações de Create (POST), Read (GET), Update (PUT/PATCH) e Delete (DELETE) em recursos de uma aplicação web, permitindo gerenciar dados através de endpoints específicos.



Create



Read



Update



Delete


C R U D

/ CRUD

- Vamos melhorar nosso script.
- Usaremos uma lista Python para armazenar itens
- Não teremos banco de dados nesta fase (didático)
- Útil para aprender rotas e métodos HTTP
- Posteriormente, podemos migrar para um DB real

1. Primeiro pause o script no terminal, com ctrl+c

```
• (venv) leonardopena@MacBook-Pro-de-Leonardo aula2-new % python app.py
  * Serving Flask app 'app'
  * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment
  * Running on http://127.0.0.1:5000
Press CTRL+C to quit
  * Restarting with stat
  * Debugger is active!
  * Debugger PIN: 877-383-895
127.0.0.1 - - [03/Jan/2025 08:43:51] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [03/Jan/2025 08:43:51] "GET /favicon.ico HTTP/1.1" 404 -
^C
❖ (venv) leonardopena@MacBook-Pro-de-Leonardo aula2-new %
```

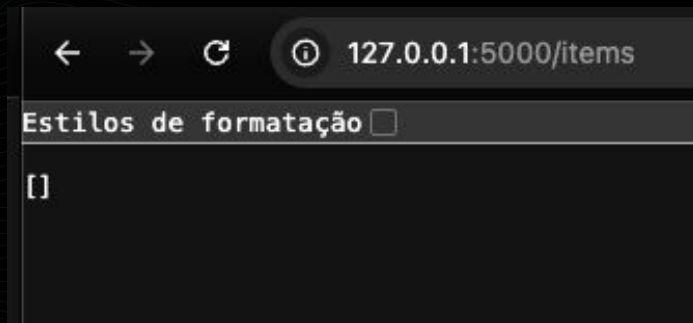
A decorative graphic at the bottom of the slide consisting of numerous thin, light gray wavy lines that create a sense of movement and depth, resembling a stylized landscape or water ripples.


**/ Criando o GET:
Ler Item**

1. Lista items armazenando dicionários in-memory
2. Rota /items com método GET para ler todos os itens
3. Import jsonify de flask (não esquecer no topo)
4. Retorna a lista como JSON (Flask converte automaticamente)
5. Primeiro passo do CRUD: R – Read

```
app.py
app.py > ...
1  from flask import Flask, jsonify
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def home():
7      return "Hello, Flask!"
8
9  items = []
10
11  @app.route('/items', methods=['GET'])
12  def get_items():
13      return jsonify(items)
14
15
16  if __name__ == '__main__':
17      app.run(debug=True)
18
```

- /
1. Agora ao acessar temos



A decorative graphic at the bottom of the slide consisting of numerous thin, light gray wavy lines that create a sense of movement and depth, resembling a stylized landscape or water ripples.

**/ Criando o POST:
Criar Item**


1. Usa método POST para criar novo item
2. `request.get_json()` captura o corpo da requisição
3. Adiciona o item à lista `items`
4. Retorna o item criado e status 201 (Created)
5. Import `request` de `flask` se ainda não tiver feito

app.py

app.py > ...


```
1 from flask import Flask, jsonify, request
```

```
9 items = []
10
11 @app.route('/items', methods=['GET'])
12 def get_items():
13     return jsonify(items)
14
15 @app.route('/items', methods=['POST'])
16 def create_item():
17     data = request.get_json()
18     items.append(data)
19     return jsonify(data), 201
20
21
```

- 
1. Explicação da rota POST:
 - a. Enviar JSON no corpo, ex.:

```
{ "nome": "Item1", "preco": 10.0 }
```
 - b. Armazena em itens de forma in-memory
 - c. Retorna o objeto recém-criado
 - d. Código de status 201 indica sucesso na criação

Obs: Vamos fazer o PUT e o DELETE e depois testamos todos!

A decorative graphic at the bottom of the slide consisting of numerous thin, light gray wavy lines that create a sense of movement and depth, resembling a stylized landscape or water ripples.

**/ Criando o PUT:
Atualizar Item**

1. Identificação do item pelo índice
<int:item_id>
2. data obtida do JSON para atualizar
o item
3. Checa se item_id está em intervalo
válido
4. Caso ok, atualiza e retorna o item
5. Se não existir, retorna erro 404


```
@app.route('/items', methods=['GET'])
def get_items():
    return jsonify(items)

@app.route('/items', methods=['POST'])
def create_item():
    data = request.get_json()
    items.append(data)
    return jsonify(data), 201

@app.route('/items/<int:item_id>', methods=['PUT'])
def update_item(item_id):
    data = request.get_json()
    if 0 <= item_id < len(items):
        items[item_id].update(data)
        return jsonify(items[item_id])
    return jsonify({"error": "Item not found"}), 404
```

A decorative graphic at the bottom of the slide consisting of numerous thin, light gray wavy lines that create a sense of movement and depth, resembling a stylized landscape or water ripples.

**/ Criando o DELETE:
Deletar Item**

- 
1. `pop(item_id)` remove e retorna o item na posição
 2. Retorna o objeto deletado como confirmação
 3. Verifica se índice está no intervalo para evitar erro
 4. Caso não exista, retorna 404
 5. Fecha o ciclo CRUD (Create, Read, Update, Delete)

```
@app.route('/items/<int:item_id>', methods=['DELETE'])
def delete_item(item_id):
    if 0 <= item_id < len(items):
        removed = items.pop(item_id)
        return jsonify(removed)
    return jsonify({"error": "Item not found"}), 404
```

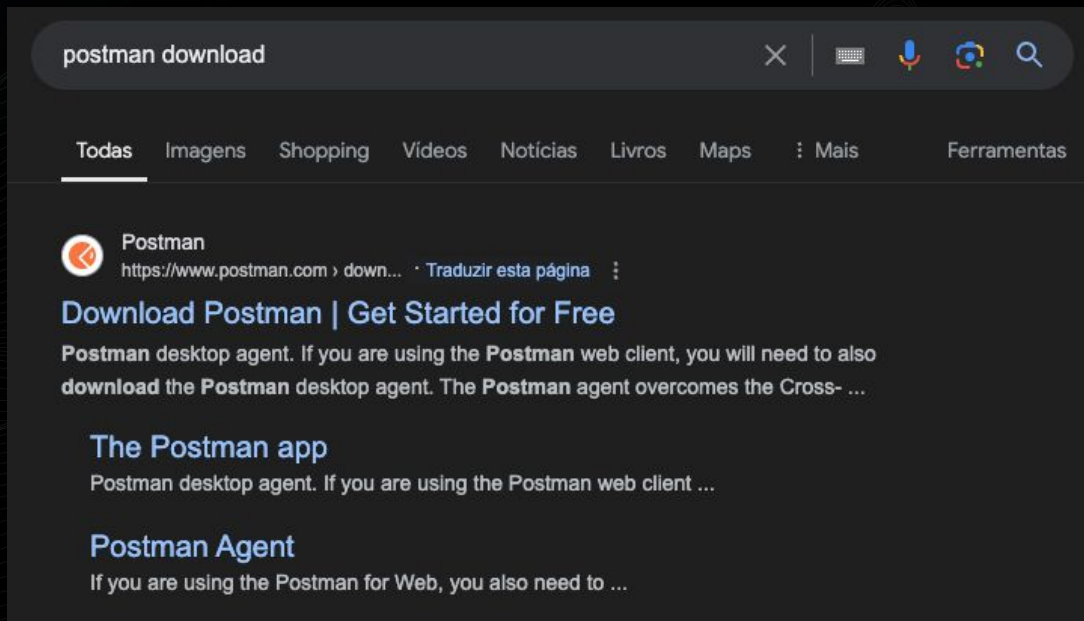
/ Revisando

- **GET /items** -> Lista todos os itens
- **POST /items** -> Cria um novo item
- **PUT /items/<item_id>** -> Atualiza item específico
- **DELETE /items/<item_id>** -> Remove item específico
- Já temos o esqueleto de uma pequena API Flask 🚀

The background of the slide features a series of thin, light gray wavy lines that create a sense of movement and depth, resembling a stylized landscape or a topographical map. These lines are concentrated in the lower half of the image, while the upper half is a plain white space.

/ Vamos testar!

Para analisar as rotas, vamos conhecer o Postman





POSTMAN

/ Postman

O Postman é uma ferramenta de desenvolvimento utilizada para testar, documentar e automatizar APIs, permitindo enviar requisições HTTP e visualizar respostas, facilitando o trabalho de desenvolvedores na criação, depuração e documentação de serviços web.



Download Postman

Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman.

The Postman app

Download the app to get started with the Postman API Platform.

Mac Intel Chip

Mac Apple Chip

By downloading and using Postman, I agree to the [Privacy Policy](#) and [Terms](#).

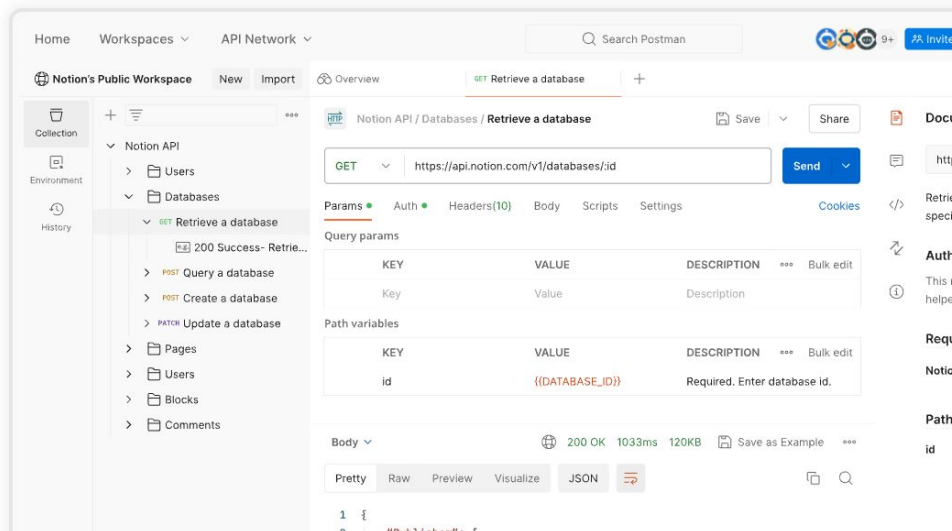
[Release Notes](#) →

Not your OS? Download for Windows ([x64](#)) or Linux ([x64](#), [arm64](#))

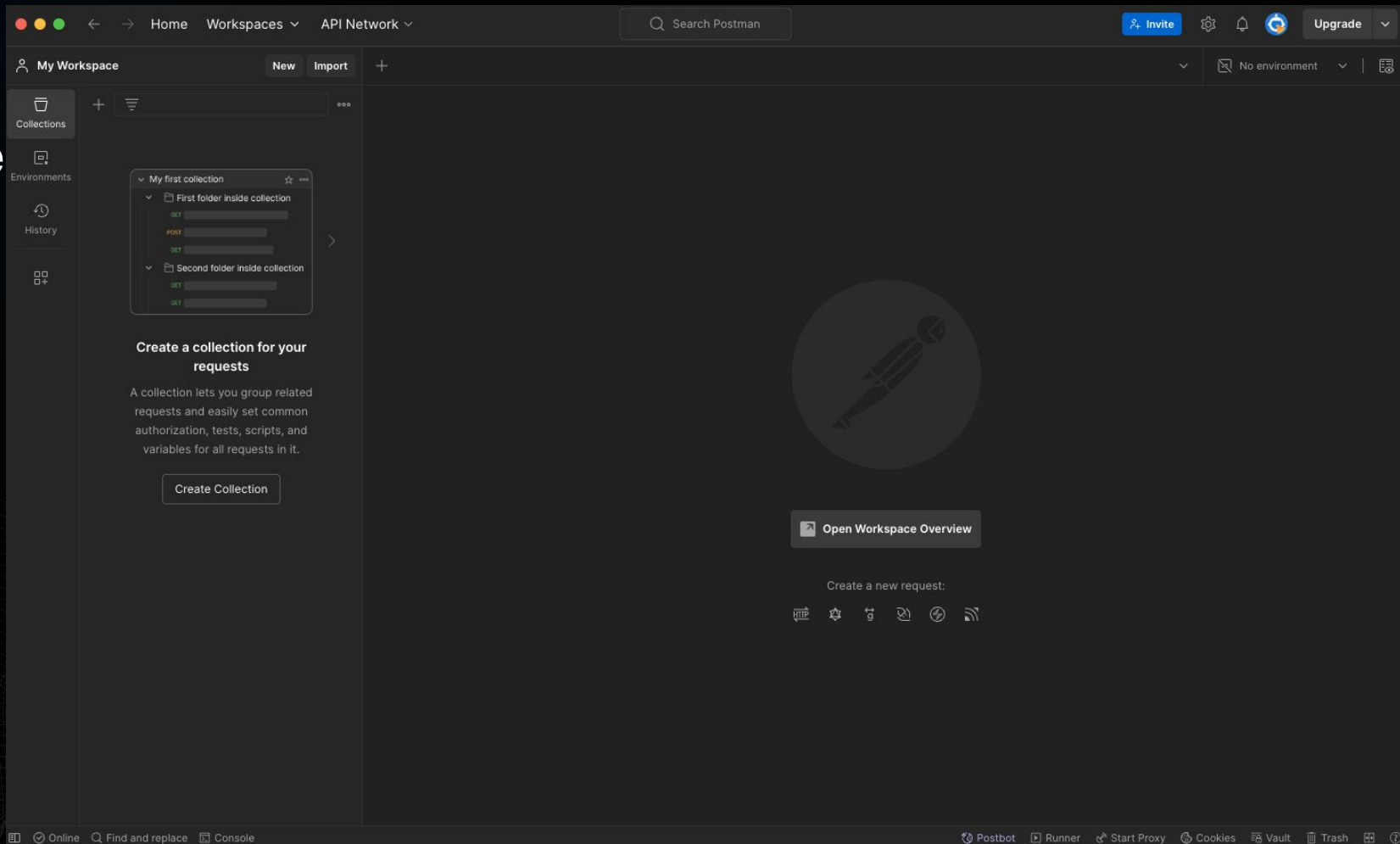
Postman on the web

Access the Postman API Platform through your web browser. Create a free account, and you're in.

Try the Web Version

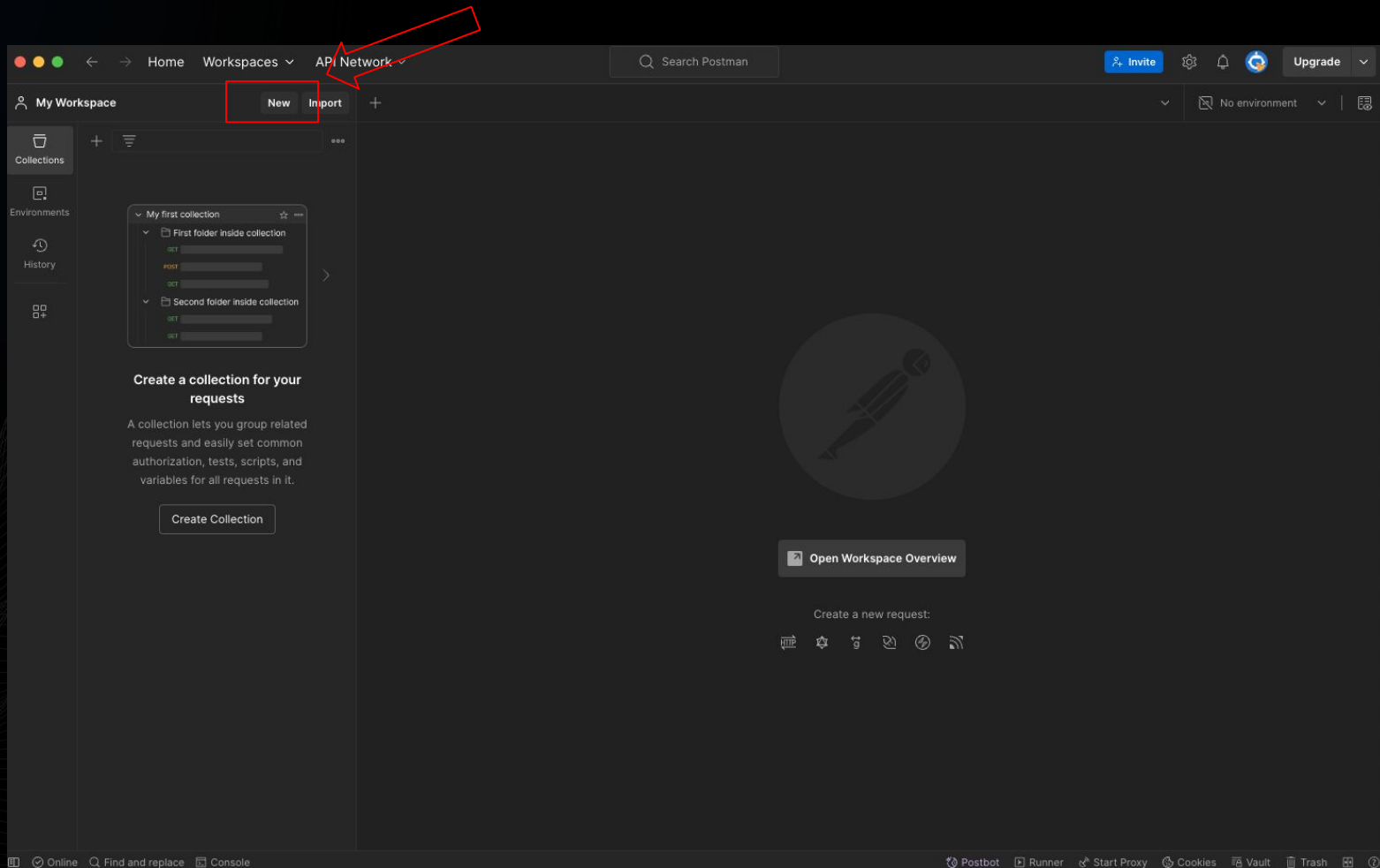


Interface

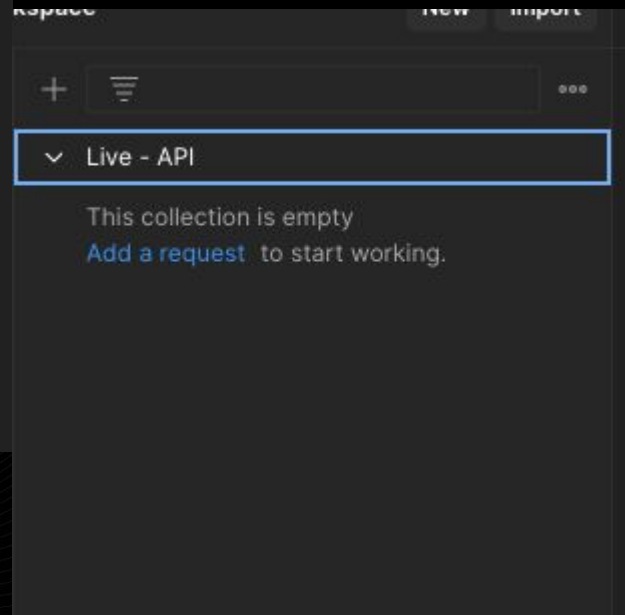
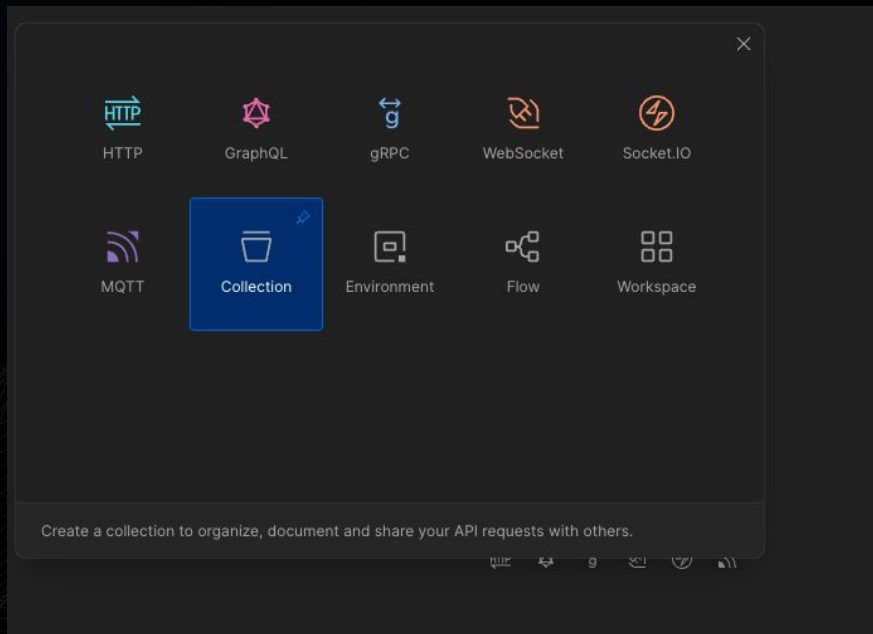


/

Começa mos criando uma coleção



/
Começa
mos
criando
uma
coleção



/ Assim
criamos
uma
requisição

GET

The screenshot displays the Postman interface for a workspace named "Live - API". A GET request named "Get Items" is configured with the URL "http://127.0.0.1:5000/items". The request has been sent, resulting in a "200 OK" status with a response time of 102 ms and a body size of 166 B. The response body is shown in JSON format as an empty array: `[]`.

Workspace: Live - API

Request: GET `http://127.0.0.1:5000/items`

Params: Authorization, Headers (7), Body, Scripts, Tests, Settings

Query Params:

Key	Value	Description
Key	Value	Description

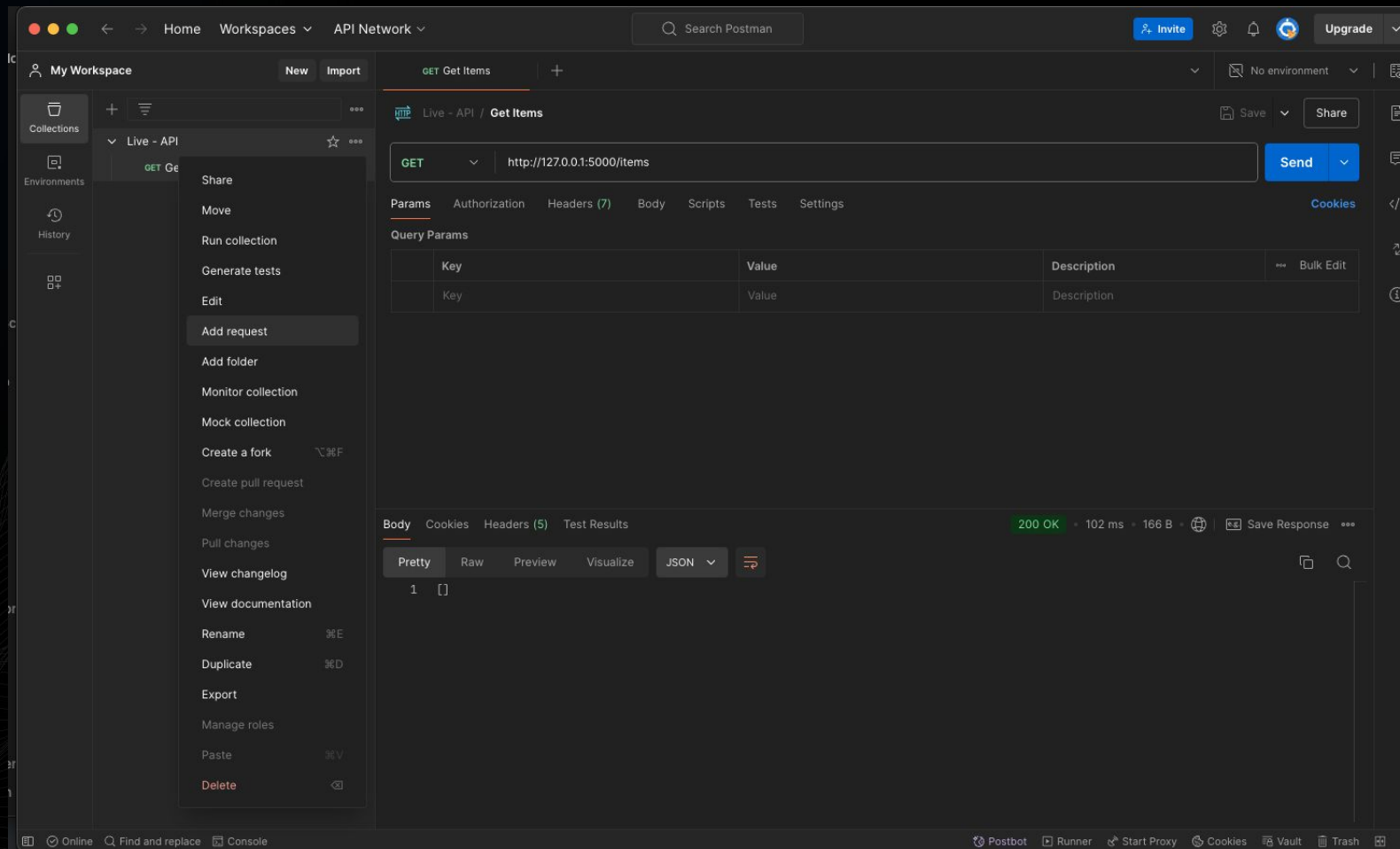
Response: 200 OK • 102 ms • 166 B

Body: Pretty, Raw, Preview, Visualize, JSON

```
1 []
```

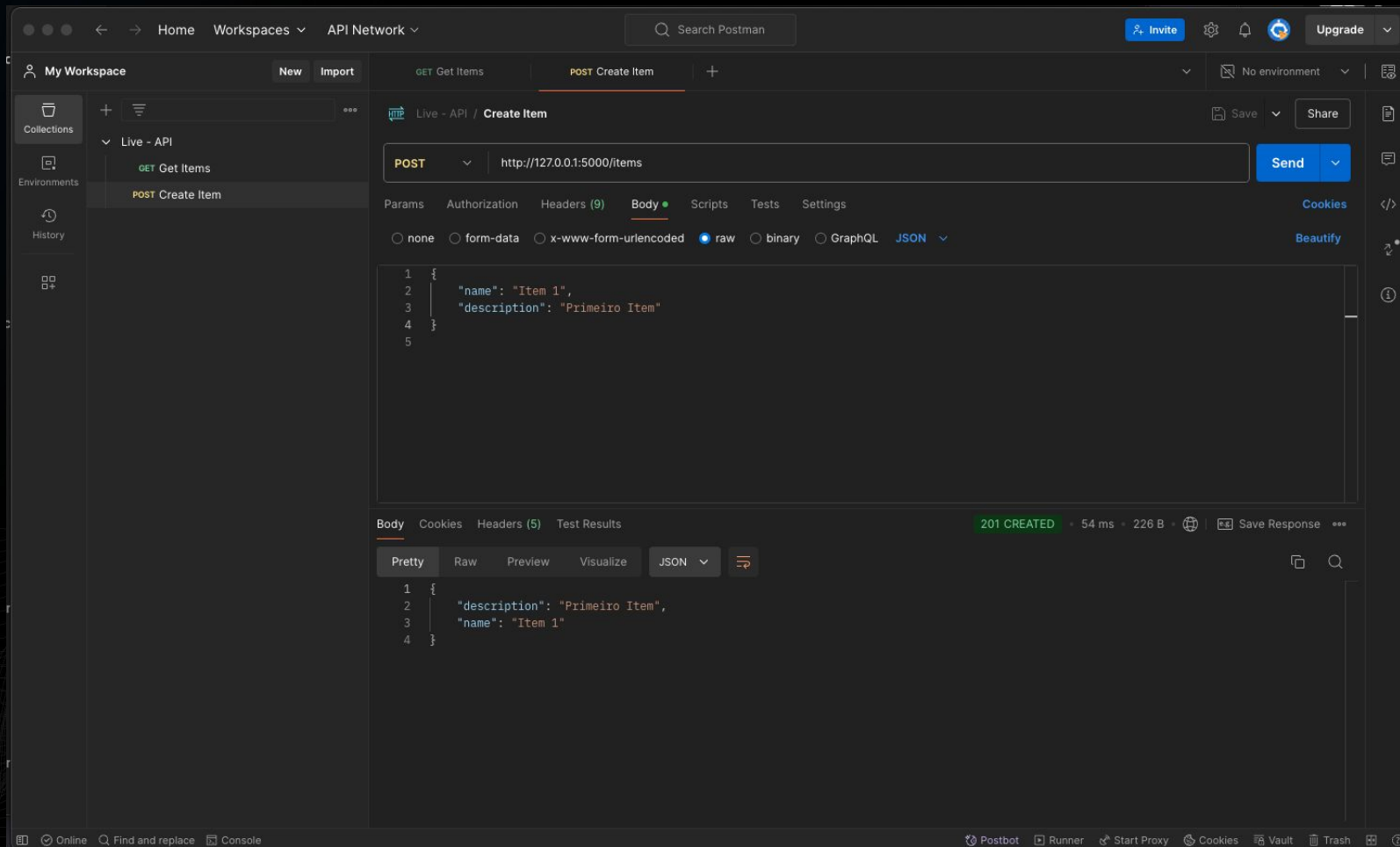
Buscamos os
itens atuais

Criando uma nova requisição POST



/

Podemos adicionar itens com POST



/

Usando GET novamente, na rota de itens, vemos os itens atualizados!!

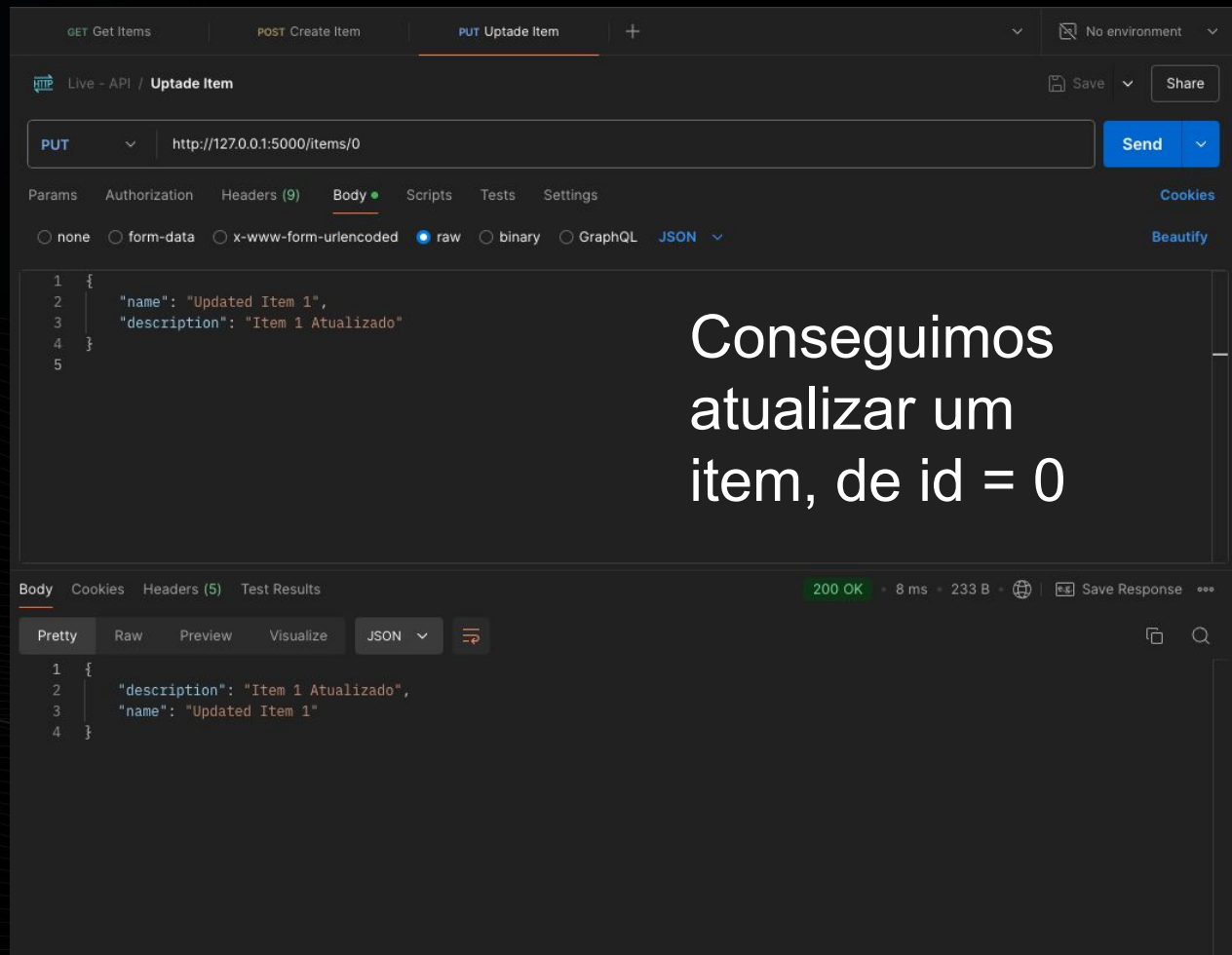
The screenshot shows the Postman interface with the following details:

- Request:** GET `http://127.0.0.1:5000/items`
- Response:** 200 OK, 17 ms, 233 B
- Body (JSON):**

```
1 [
2   {
3     "description": "Primeiro Item",
4     "name": "Item 1"
5   }
6 ]
```

Key	Value	Description
Key	Value	Description

Criando uma nova requisição PUT



The screenshot displays a REST client interface with a dark theme. At the top, there are tabs for different HTTP methods: GET Get Items, POST Create Item, and PUT Uptade Item (note the typo). The PUT Uptade Item tab is selected. Below the tabs, the URL bar shows 'http://127.0.0.1:5000/items/0'. To the right of the URL bar is a 'Send' button. Below the URL bar, there are tabs for Params, Authorization, Headers (9), Body, Scripts, Tests, and Settings. The 'Body' tab is selected, and the request body is shown in a text area with the following JSON:

```
1 {  
2   "name": "Updated Item 1",  
3   "description": "Item 1 Atualizado"  
4 }  
5
```

 To the right of the text area, there are radio buttons for different content types: none, form-data, x-www-form-urlencoded, raw (selected), binary, and GraphQL. There is also a 'JSON' button. Below the text area, there are tabs for Body, Cookies, Headers (5), and Test Results. The 'Body' tab is selected, and the response body is shown in a text area with the following JSON:

```
1 {  
2   "description": "Item 1 Atualizado",  
3   "name": "Updated Item 1"  
4 }
```

 At the top right of the interface, there is a 'No environment' dropdown. At the bottom right, there is a status bar showing '200 OK', '8 ms', '233 B', and a 'Save Response' button.

PUT Uptade Item

PUT http://127.0.0.1:5000/items/0

Send

Params Authorization Headers (9) Body Scripts Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {  
2   "name": "Updated Item 1",  
3   "description": "Item 1 Atualizado"  
4 }  
5
```

Body Cookies Headers (5) Test Results

200 OK 8 ms 233 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "description": "Item 1 Atualizado",  
3   "name": "Updated Item 1"  
4 }
```

Conseguimos
atualizar um
item, de id = 0

/

Voltando ao GET

Item atualizado!

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://127.0.0.1:5000/items
- Status:** 200 OK
- Response Time:** 274 ms
- Response Size:** 245 B
- Response Body (JSON):**

```
[
  {
    "description": "Item 1 Atualizado",
    "name": "Updated Item 1"
  }
]
```

The interface also shows tabs for Params, Authorization, Headers (7), Body, Scripts, Tests, and Settings. The Body tab is currently selected, displaying the JSON response in a Pretty format.

/

E finalmente com DELETE podemos deletar um item de id definido

The screenshot shows a REST client interface with a dark theme. At the top, there are tabs for different HTTP methods: GET Get Items, POST Create Item, PUT Update Item, and DEL Delete Item (which is currently selected). Below the tabs, the URL bar shows 'http://127.0.0.1:5000/items/0' with a 'DELETE' method dropdown. The interface has several tabs for request configuration: Params, Authorization, Headers (7), Body, Scripts, Tests, and Settings. Below these is a 'Query Params' table with columns for Key, Value, and Description. The 'Body' tab is selected at the bottom, showing a JSON response in 'Pretty' format. The response status is '200 OK' with a response time of 68 ms and a size of 233 B. The JSON body contains an object with 'description' and 'name' fields.

GET Get Items POST Create Item PUT Update Item **DEL Delete Item** +

Live - API / Delete Item

DELETE http://127.0.0.1:5000/items/0

Params Authorization Headers (7) Body Scripts Tests Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body Cookies Headers (5) Test Results **200 OK** • 68 ms • 233 B •

Pretty Raw Preview Visualize JSON

```
1 {
2   "description": "Item 1 Atualizado",
3   "name": "Updated Item 1"
4 }
```

POSTECH

FIAP + alura