

PROGRAMAÇÃO WEB II

Curso Técnico Integrado em Informática
Lucas Sampaio Leite



Persistência de dados com o Flask

- A persistência de dados em Flask pode ser implementada de diferentes maneiras, sendo a mais comum o uso de bancos de dados relacionais, como SQLite, PostgreSQL ou MySQL, e não relacionais, como MongoDB.
- Em geral, esse processo é facilitado por ORMs (Object-Relational Mappers), como o SQLAlchemy.
- Nesta aula, vamos usar o ORM SQLAlchemy com o banco de dados SQLite.

ORMs (Object-Relational Mappers)

- Um ORM (Object-Relational Mapping ou Mapeamento Objeto-Relacional) é uma ferramenta que permite interagir com um banco de dados relacional usando objetos da linguagem de programação, em vez de escrever comandos SQL diretamente.
- Um ORM traduz classes e objetos em tabelas e registros do banco de dados, e vice-versa.

ORMs (Object-Relational Mappers)

Relational database

ID	FIRST_NAME	LAST_NAME	PHONE
1	John	Connor	+16105551234
2	Matt	Makai	+12025555689
3	Sarah	Smith	+19735554512
...

Python objects

```
class Person:  
    first_name = "John"  
    last_name = "Connor"  
    phone_number = "+16105551234"
```

```
class Person:  
    first_name = "Matt"  
    last_name = "Makai"  
    phone_number = "+12025555689"
```

```
class Person:  
    first_name = "Sarah"  
    last_name = "Smith"  
    phone_number = "+19735554512"
```

ORMs provide a bridge between
**relational database tables, relationships
and fields** and **Python objects**

Criando uma nova aplicação Flask

- Até agora, criamos nossa aplicação Flask instanciando o objeto Flask diretamente no topo do código, como neste exemplo:

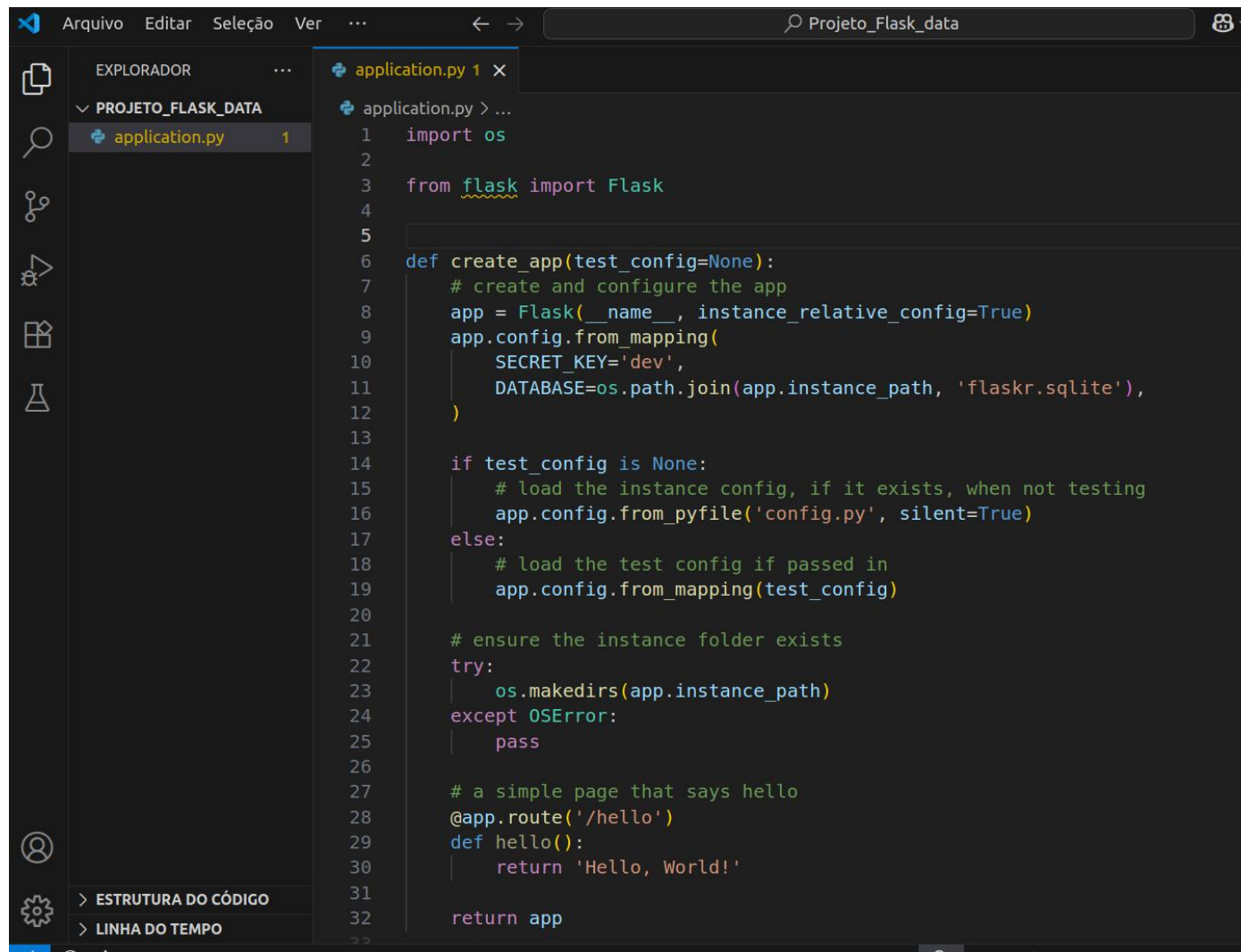
```
application.py X
application.py > ...
1  from flask import Flask
2
3  app = Flask(__name__)
4
5
6  @app.route("/")
7  def hello_world():
8      return "<p>Hello, World!</p>"
9
10
11
```

Criando uma nova aplicação Flask

- Esse modelo é direto e funciona bem para projetos simples. No entanto, à medida que o projeto cresce, essa abordagem começa a apresentar limitações:
 - Dificulta testes automatizados;
 - Complica o uso de múltiplas configurações;
 - Prejudica a modularização do código (como registro de extensões e blueprints).
- Para evitar esses problemas, adotamos uma abordagem mais flexível: a função fábrica (application factory). Nela, criamos e configuramos a aplicação dentro de uma função e retornamos a instância já preparada.

Criando um novo app

- Application factory: <https://flask.palletsprojects.com/en/stable/tutorial/factory/>



```
Arquivo  Editar  Seleção  Ver  ...  Projeto_Flask_data  88 v
EXPLORADOR
PROJETO_FLASK_DATA
  application.py  1
application.py 1 x
application.py > ...
1  import os
2
3  from flask import Flask
4
5
6  def create_app(test_config=None):
7      # create and configure the app
8      app = Flask(__name__, instance_relative_config=True)
9      app.config.from_mapping(
10         SECRET_KEY='dev',
11         DATABASE=os.path.join(app.instance_path, 'flaskr.sqlite'),
12     )
13
14     if test_config is None:
15         # load the instance config, if it exists, when not testing
16         app.config.from_pyfile('config.py', silent=True)
17     else:
18         # load the test config if passed in
19         app.config.from_mapping(test_config)
20
21     # ensure the instance folder exists
22     try:
23         os.makedirs(app.instance_path)
24     except OSError:
25         pass
26
27     # a simple page that says hello
28     @app.route('/hello')
29     def hello():
30         return 'Hello, World!'
31
32     return app
```


Criando um novo app

os será usado para
manipular caminhos de
arquivos e pastas;

Flask é a classe principal
do framework

```
1 import os
2
3 from flask import Flask
4
5
6 def create_app(test_config=None):
7     # create and configure the app
8     app = Flask(__name__, instance_relative_config=True)
9     app.config.from_mapping(
10         SECRET_KEY='dev',
11         DATABASE=os.path.join(app.instance_path, 'flaskr.sqlite'),
12     )
13
14     if test_config is None:
15         # load the instance config, if it exists, when not testing
16         app.config.from_pyfile('config.py', silent=True)
17     else:
18         # load the test config if passed in
19         app.config.from_mapping(test_config)
20
21     # ensure the instance folder exists
22     try:
23         os.makedirs(app.instance_path)
24     except OSError:
25         pass
26
27     # a simple page that says hello
28     @app.route('/hello')
29     def hello():
30         return 'Hello, World!'
31
32     return app
```


Criando um novo app

Define a função fábrica que irá criar e configurar a aplicação Flask

Cria uma instância da aplicação Flask.

```
1  import os
2
3  from flask import Flask
4
5
6  def create_app(test_config=None):
7      # create and configure the app
8      app = Flask(__name__, instance_relative_config=True)
9      app.config.from_mapping(
10         SECRET_KEY='dev',
11         DATABASE=os.path.join(app.instance_path, 'flaskr.sqlite'),
12     )
13
14     if test_config is None:
15         # load the instance config, if it exists, when not testing
16         app.config.from_pyfile('config.py', silent=True)
17     else:
18         # load the test config if passed in
19         app.config.from_mapping(test_config)
20
21     # ensure the instance folder exists
22     try:
23         os.makedirs(app.instance_path)
24     except OSError:
25         pass
26
27     # a simple page that says hello
28     @app.route('/hello')
29     def hello():
30         return 'Hello, World!'
31
32     return app
```

Criando um novo app

Define a configuração padrão da aplicação

Se não estivermos em modo de teste, tenta carregar configurações adicionais a partir do arquivo instance/config.py

Garante que a pasta instance/ existe

```
1 import os
2
3 from flask import Flask
4
5
6 def create_app(test_config=None):
7     # create and configure the app
8     app = Flask(__name__, instance_relative_config=True)
9     app.config.from_mapping(
10         SECRET_KEY='dev',
11         DATABASE=os.path.join(app.instance_path, 'flaskr.sqlite'),
12     )
13
14     if test_config is None:
15         # load the instance config, if it exists, when not testing
16         app.config.from_pyfile('config.py', silent=True)
17     else:
18         # load the test config if passed in
19         app.config.from_mapping(test_config)
20
21     # ensure the instance folder exists
22     try:
23         os.makedirs(app.instance_path)
24     except OSError:
25         pass
26
27     # a simple page that says hello
28     @app.route('/hello')
29     def hello():
30         return 'Hello, World!'
31
32     return app
```

Criando um novo app

Cria uma rota simples

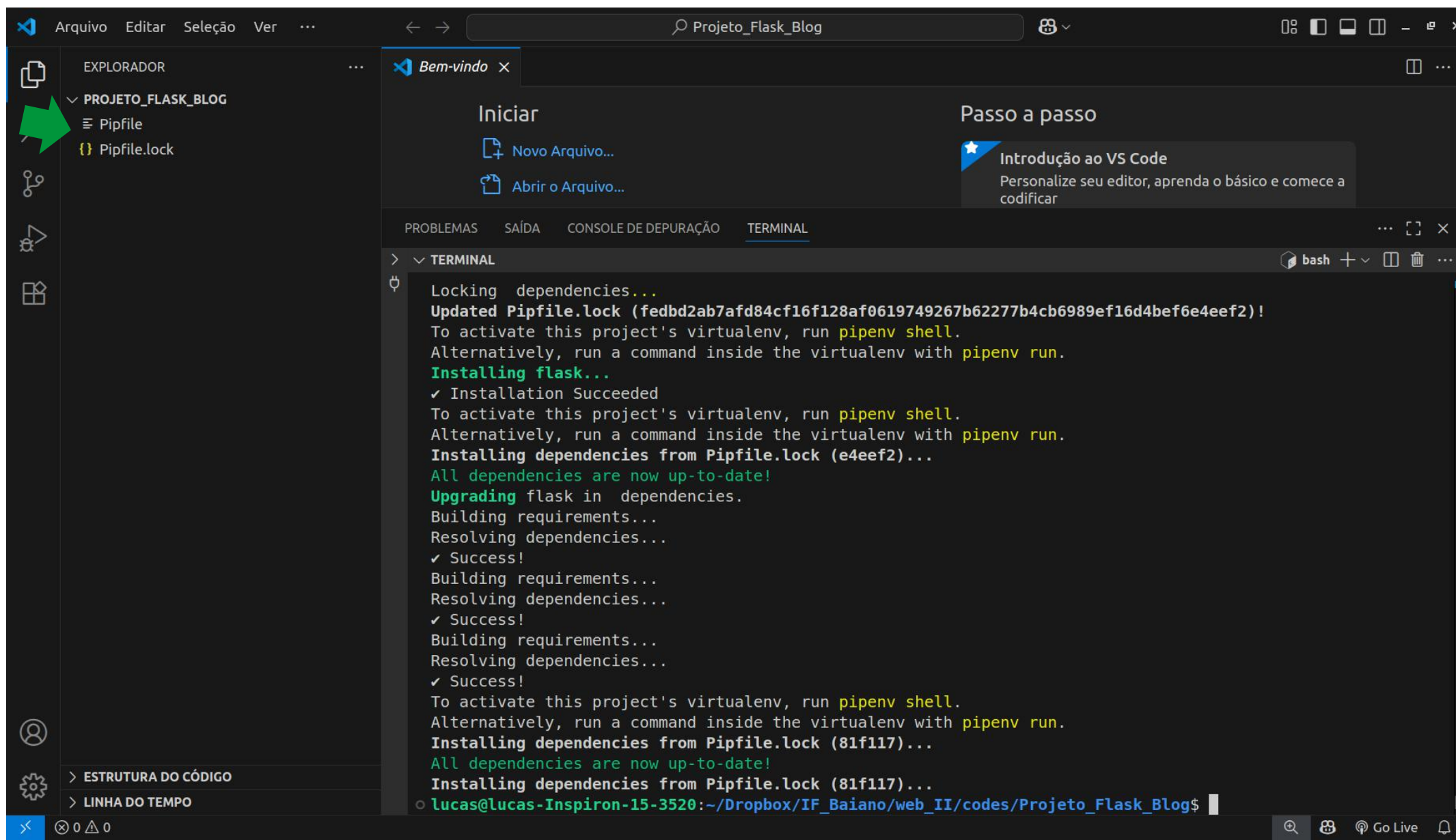
Retorna o app configurado

```
1  import os
2
3  from flask import Flask
4
5
6  def create_app(test_config=None):
7      # create and configure the app
8      app = Flask(__name__, instance_relative_config=True)
9      app.config.from_mapping(
10         SECRET_KEY='dev',
11         DATABASE=os.path.join(app.instance_path, 'flaskr.sqlite'),
12     )
13
14     if test_config is None:
15         # load the instance config, if it exists, when not testing
16         app.config.from_pyfile('config.py', silent=True)
17     else:
18         # load the test config if passed in
19         app.config.from_mapping(test_config)
20
21     # ensure the instance folder exists
22     try:
23         os.makedirs(app.instance_path)
24     except OSError:
25         pass
26
27     # a simple page that says hello
28     @app.route('/hello')
29     def hello():
30         return 'Hello, World!'
31
32     return app
```

Criando uma nova aplicação Flask

- Crie um novo ambiente virtual utilizando o pipenv e instale a dependência do Flask.
- Selecione o interpretador do ambiente virtual no VS Code.
- Inicie a aplicação com o comando: `flask --app applicationname run --debug`
- Acesse a URL `http://localhost/hello` no navegador ou via Postman para testar a rota definida na função `create_app`

Criando uma nova aplicação Flask



Arquivo Editar Seleção Ver ...

Projeto_Flask_Blog

EXPLORADOR

PROJETO_FLASK_BLOG

Pipfile

Pipfile.lock

Bem-vindo

Iniciar

Novo Arquivo...

Abrir o Arquivo...

Passo a passo

Introdução ao VS Code

Personalize seu editor, aprenda o básico e comece a codificar

PROBLEMAS SAÍDA CONSOLE DE DEPURAÇÃO TERMINAL

TERMINAL

bash

```
Locking dependencies...
Updated Pipfile.lock (fedbd2ab7afd84cf16f128af0619749267b62277b4cb6989ef16d4bef6e4eef2)!
To activate this project's virtualenv, run pipenv shell.
Alternatively, run a command inside the virtualenv with pipenv run.
Installing flask...
✓ Installation Succeeded
To activate this project's virtualenv, run pipenv shell.
Alternatively, run a command inside the virtualenv with pipenv run.
Installing dependencies from Pipfile.lock (e4eef2)...
All dependencies are now up-to-date!
Upgrading flask in dependencies.
Building requirements...
Resolving dependencies...
✓ Success!
Building requirements...
Resolving dependencies...
✓ Success!
Building requirements...
Resolving dependencies...
✓ Success!
To activate this project's virtualenv, run pipenv shell.
Alternatively, run a command inside the virtualenv with pipenv run.
Installing dependencies from Pipfile.lock (81f117)...
All dependencies are now up-to-date!
Installing dependencies from Pipfile.lock (81f117)...
o lucas@lucas-Inspiron-15-3520:~/Dropbox/IF_Baiano/web_II/codes/Projeto_Flask_Blog$
```

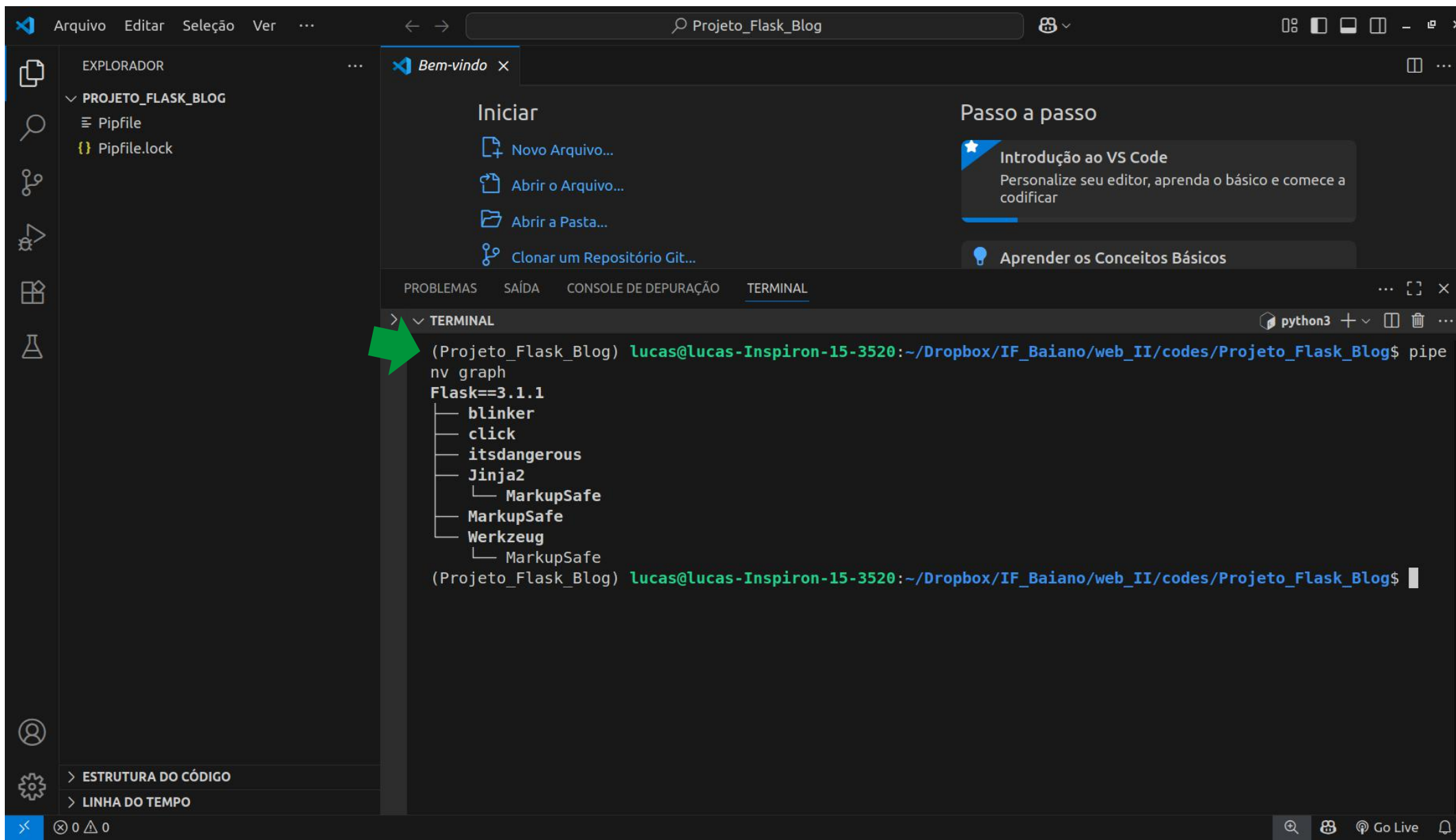
ESTRUTURA DO CÓDIGO

LINHA DO TEMPO

0 0 0

Go Live

Criando uma nova aplicação Flask



Arquivo Editar Seleção Ver ...

Projeto_Flask_Blog

EXPLORADOR

PROJETO_FLASK_BLOG

Pipfile

Pipfile.lock

Bem-vindo

Iniciar

Novo Arquivo...

Abrir o Arquivo...

Abrir a Pasta...

Clonar um Repositório Git...

Passo a passo

Introdução ao VS Code

Personalize seu editor, aprenda o básico e comece a codificar

Aprender os Conceitos Básicos

PROBLEMAS SAÍDA CONSOLE DE DEPURAÇÃO TERMINAL

python3

(Projeto_Flask_Blog) lucas@lucas-Inspiron-15-3520:~/Dropbox/IF_Baiano/web_II/codes/Projeto_Flask_Blog\$ pipenv graph

nv graph

Flask==3.1.1

- blinker
- click
- itsdangerous
- Jinja2
 - MarkupSafe
- MarkupSafe
- Werkzeug
 - MarkupSafe

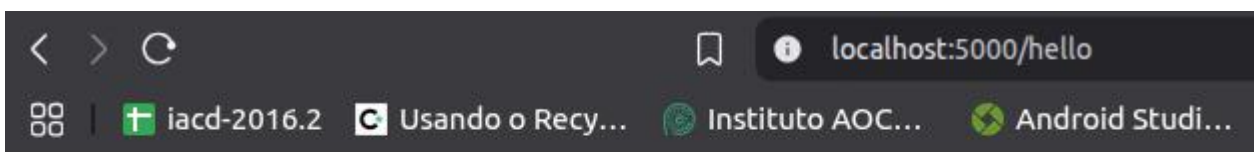
(Projeto_Flask_Blog) lucas@lucas-Inspiron-15-3520:~/Dropbox/IF_Baiano/web_II/codes/Projeto_Flask_Blog\$

ESTRUTURA DO CÓDIGO

LINHA DO TEMPO

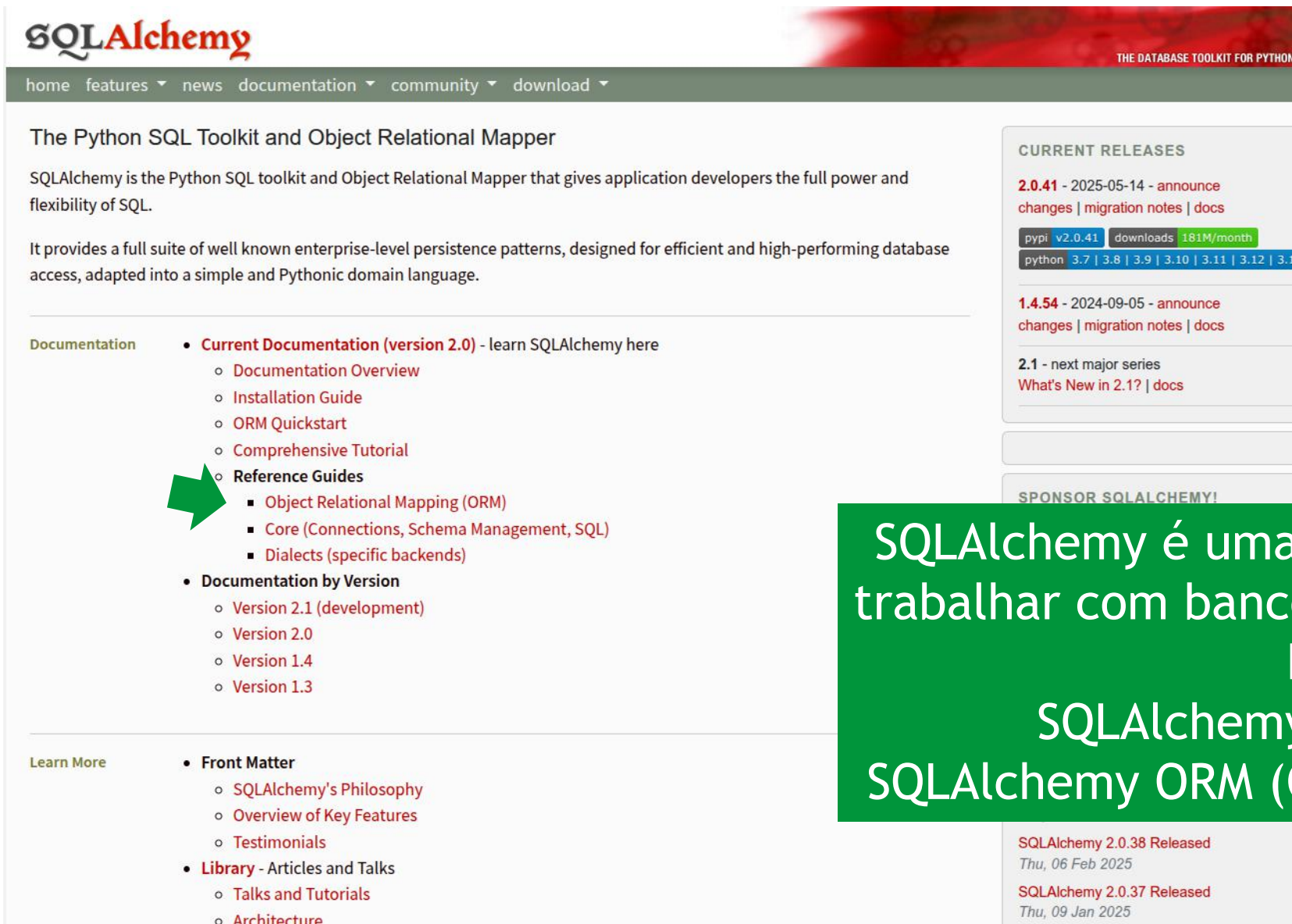
Testando a rota criada

```
> ▾ TERMINAL python3 +  
run --debug  
* Serving Flask app 'application'  
* 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: 304-869-304  
127.0.0.1 - - [13/Jul/2025 18:57:52] "GET /hello HTTP/1.1" 200 -  
127.0.0.1 - - [13/Jul/2025 18:57:52] "GET /favicon.ico HTTP/1.1" 404 -
```



Hello, World!

SQLAlchemy



The screenshot shows the SQLAlchemy website homepage. At the top, there's a navigation bar with links: home, features, news, documentation, community, and download. Below the navigation bar, the main heading reads "The Python SQL Toolkit and Object Relational Mapper". A descriptive paragraph follows, stating that SQLAlchemy is the Python SQL toolkit and ORM that gives application developers the full power and flexibility of SQL. It also mentions that it provides a full suite of well-known enterprise-level persistence patterns designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

On the right side, there's a "CURRENT RELEASES" section. It lists two releases: 2.0.41 (dated 2025-05-14) and 1.4.54 (dated 2024-09-05). Each release includes links for "announce", "changes", "migration notes", and "docs". Below the releases, there's a section for "2.1 - next major series" with links for "What's New in 2.1?" and "docs".

On the left side, there's a "Documentation" section. It lists several links: "Current Documentation (version 2.0) - learn SQLAlchemy here", "Documentation Overview", "Installation Guide", "ORM Quickstart", "Comprehensive Tutorial", "Reference Guides", "Object Relational Mapping (ORM)", "Core (Connections, Schema Management, SQL)", "Dialects (specific backends)", "Documentation by Version", "Version 2.1 (development)", "Version 2.0", "Version 1.4", and "Version 1.3". A green arrow points to the "Reference Guides" link.

At the bottom, there's a "Learn More" section. It lists "Front Matter" (including "SQLAlchemy's Philosophy", "Overview of Key Features", and "Testimonials") and "Library - Articles and Talks" (including "Talks and Tutorials" and "Architecture").

At the bottom right, there's a "SPONSOR SQLALCHEMY!" section. It lists "SQLAlchemy 2.0.38 Released" (dated Thu, 06 Feb 2025) and "SQLAlchemy 2.0.37 Released" (dated Thu, 09 Jan 2025).

SQLAlchemy é uma biblioteca poderosa para trabalhar com bancos de dados relacionais em Python.

SQLAlchemy Core (baixo nível)
SQLAlchemy ORM (Object-Relational Mapper)

Documentação:
<https://www.sqlalchemy.org/>

Flask SQLAlchemy



Project Links

[Donate](#)
[PyPI Releases](#)
[Source Code](#)
[Issue Tracker](#)
[Website](#)
[Twitter](#)
[Chat](#)

Contents

[Flask-SQLAlchemy](#)
[User Guide](#)
[API Reference](#)
[Additional Information](#)

Quick search

Flask SQLAlchemy

Flask-SQLAlchemy is an extension for [Flask](#) that adds support for [SQLAlchemy](#) to your application. It simplifies using SQLAlchemy with Flask by setting up common objects and patterns for using those objects, such as a session tied to each web request, models, and engines.

Flask-SQLAlchemy does not change how SQLAlchemy works or is used. See the [SQLAlchemy documentation](#) to learn how to work with the ORM in depth. The documentation here will only cover setting up the extension, not how to use SQLAlchemy.

User Guide

- [Quick Start](#)
 - [Check the SQLAlchemy Documentation](#)
 - [Installation](#)
 - [Initialize the Extension](#)
 - [Configure the Extension](#)
 - [Define Models](#)
 - [Create the Tables](#)
 - [Query the Data](#)
 - [What to Remember](#)
- [Configuration](#)
 - [Configuration Keys](#)
 - [Connection URL Format](#)
 - [Default Driver Options](#)

Flask-SQLAlchemy é uma extensão do Flask que integra o SQLAlchemy — ORM (Object-Relational Mapper) para Python — com aplicações Flask.

Documentação: <https://flask-sqlalchemy.readthedocs.io/en/stable/>

Instalando o Flask SQLAlchemy

- `pipenv install flask_sqlalchemy`

✓ TERMINAL

python3 ⚠ + ▢ 🗑 ...

```
(Projeto_Flask_Blog) lucas@lucas-Inspiron-15-3520:~/Dropbox/IF_Baiano/web_II/codes/Projeto_Flask_Blog$ pipenv install flask_sqlalchemy
Installing flask_sqlalchemy...
✓ Installation Succeeded
Installing dependencies from Pipfile.lock (81f117)...
All dependencies are now up-to-date!
Upgrading flask_sqlalchemy in dependencies.
Building requirements...
Resolving dependencies...
✓ Success!
Building requirements...
Resolving dependencies...
✓ Success!
Building requirements...
Resolving dependencies...
✓ Success!
Installing dependencies from Pipfile.lock (756b3a)...
All dependencies are now up-to-date!
Installing dependencies from Pipfile.lock (756b3a)...
(Projeto_Flask_Blog) lucas@lucas-Inspiron-15-3520:~/Dropbox/IF_Baiano/web_II/codes/Projeto_Flask_Blog$
```

Instalando o Flask SQLAlchemy

```
▼ TERMINAL python3 ⚠ + ▾ 🗑 ...  
(Projeto_Flask_Blog) lucas@lucas-Inspiron-15-3520:~/Dropbox/IF_Baiano/web_II/codes/Projeto_Flask_Blog$ pip  
nv graph  
Flask-SQLAlchemy==3.1.1  
├── Flask  
│   ├── blinker  
│   ├── click  
│   ├── itsdangerous  
│   ├── Jinja2  
│   │   └── MarkupSafe  
│   ├── MarkupSafe  
│   ├── Werkzeug  
│   │   └── MarkupSafe  
└── SQLAlchemy  
    ├── greenlet  
    └── typing_extensions  
(Projeto_Flask_Blog) lucas@lucas-Inspiron-15-3520:~/Dropbox/IF_Baiano/web_II/codes/Projeto_Flask_Blog$
```


Inicializando a extensão SQLAlchemy

Initialize the Extension

First create the `db` object using the `SQLAlchemy` constructor.

Pass a subclass of either `DeclarativeBase` or `DeclarativeBaseNoMeta` to the constructor.

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
    pass

db = SQLAlchemy(model_class=Base)
```

```
application.py > ...
1  import os
2
3  from flask import Flask
4  from flask_sqlalchemy import SQLAlchemy
5  from sqlalchemy.orm import DeclarativeBase
6
7  class Base(DeclarativeBase):
8      pass
9
10 db = SQLAlchemy(model_class=Base)
11
12
13 def create_app(test_config=None):
14     # create and configure the app
15     app = Flask(__name__, instance_relative_config=True)
16     app.config.from_mapping(
```

Inicializando a extensão SQLAlchemy

Define uma nova classe Base que herda de DeclarativeBase. Ela será a superclasse para todos os modelos ORM

Cria a instância da extensão SQLAlchemy, informando que os modelos devem herdar de Base

```
application.py > ...
1  import os
2
3  from flask import Flask
4  from flask_sqlalchemy import SQLAlchemy
5  from sqlalchemy.orm import DeclarativeBase
6
7  class Base(DeclarativeBase):
8      pass
9
10 db = SQLAlchemy(model_class=Base)
11
12
13 def create_app(test_config=None):
14     # create and configure the app
15     app = Flask(__name__, instance_relative_config=True)
16     app.config.from_mapping(
```

Inicializando a extensão do SQLAlchemy

```
def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
        SECRET_KEY='dev',
        SQLALCHEMY_DATABASE_URI='sqlite:///blog.sqlite'
    )

    if test_config is None:
        # load the instance config, if it exists, when not testing
        app.config.from_pyfile('config.py', silent=True)
    else:
        # load the test config if passed in
        app.config.from_mapping(test_config)

    # ensure the instance folder exists
    try:
        os.makedirs(app.instance_path)
    except OSError:
        pass

    db.init_app(app)

    return app
```


Inicializando a extensão do SQLAlchemy

```
def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
        SECRET_KEY='dev',
        SQLALCHEMY_DATABASE_URI='sqlite:///blog.sqlite'
    )

    if test_config is None:
        # load the instance config, if it exists, when not testing
        app.config.from_pyfile('config.py', silent=True)
    else:
        # load the test config if passed in
        app.config.from_mapping(test_config)

    # ensure the instance folder exists
    try:
        os.makedirs(app.instance_path)
    except OSError:
        pass

    db.init_app(app)

    return app
```

Essa configuração é usada pelo Flask-SQLAlchemy para definir o endereço do banco de dados.

Vincula a extensão Flask-SQLAlchemy à aplicação Flask que foi criada na função create_app.

Inicializando o banco de dados via comandos CLI no Flask

```
import os

import click
from flask import Flask, current_app
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
    pass

db = SQLAlchemy(model_class=Base)

@click.command("init-db")
def init_db_command():
    with current_app.app_context():
        db.create_all()
    click.echo("Inicializando a base de dados...")
```

Inicializando o banco de dados via comandos CLI no Flask

```
import os

import click
from flask import Flask, current_app
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
    pass

db = SQLAlchemy(model_class=Base)

@click.command("init-db")
def init_db_command():
    with current_app.app_context():
        db.create_all()
    click.echo("Inicializando a base de dados...")
```

Define um comando de linha de comando (CLI) personalizado para o Flask usando o pacote Click (Click é o sistema de CLI que o Flask usa internamente).

O comando será chamado "init-db".

Inicializando o banco de dados via comandos CLI no Flask

```
import os

import click
from flask import Flask, current_app
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
    pass

db = SQLAlchemy(model_class=Base)

@click.command("init-db")
def init_db_command():
    with current_app.app_context():
        db.create_all()
    click.echo("Inicializando a base de dados...")
```

`current_app` é uma forma de acessar a aplicação Flask que está ativa no momento, mesmo fora do contexto da função `create_app`.

`db.create_all()` cria todas as tabelas do banco de dados que ainda não existem, baseadas nos seus modelos definidos com SQLAlchemy

Inicializando o banco de dados via comandos CLI no Flask

```
def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
        SECRET_KEY='dev',
        SQLALCHEMY_DATABASE_URI='sqlite:///blog.sqlite'
    )

    if test_config is None:
        # load the instance config, if it exists, when not testing
        app.config.from_pyfile('config.py', silent=True)
    else:
        # load the test config if passed in
        app.config.from_mapping(test_config)

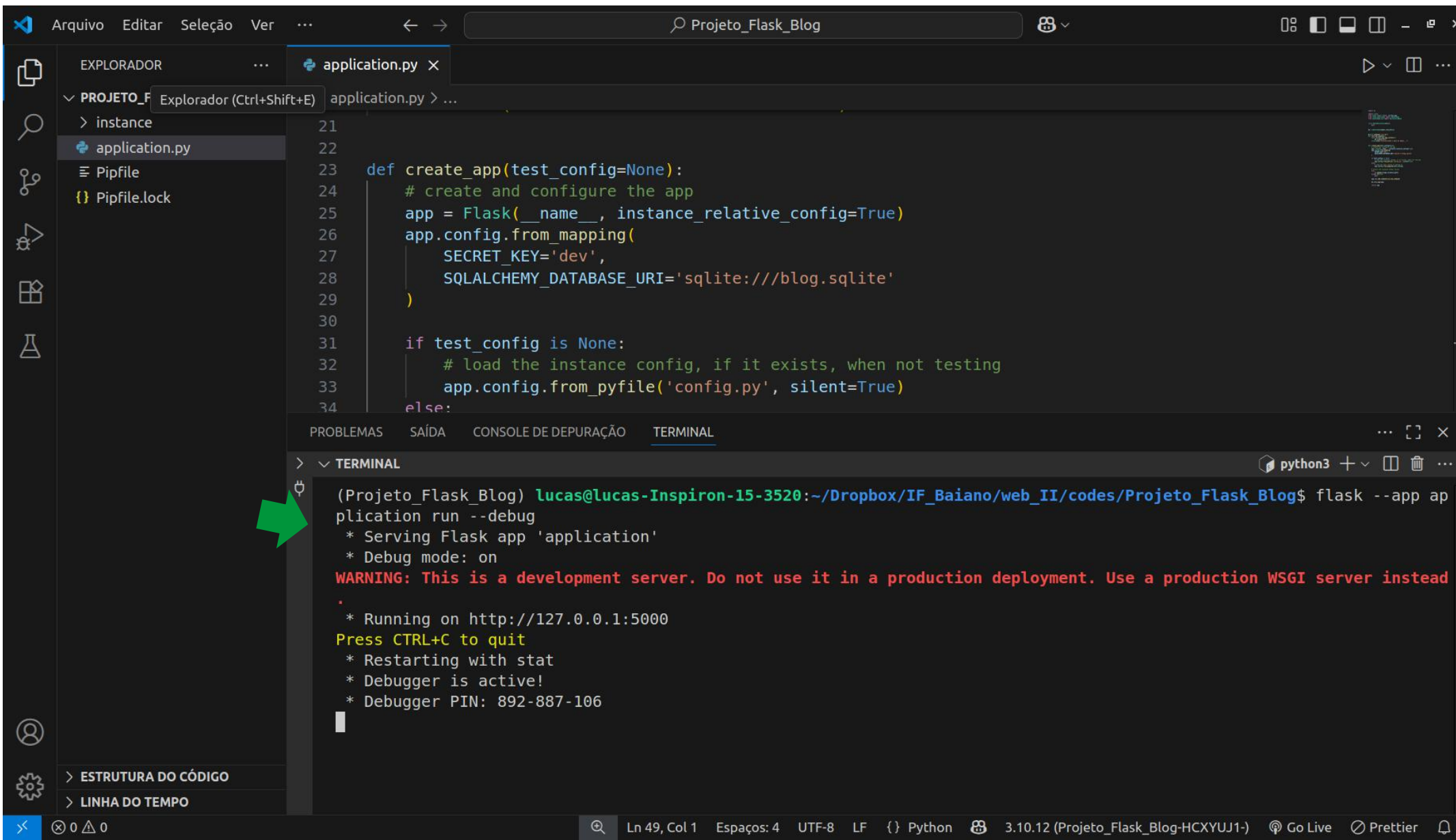
    # ensure the instance folder exists
    try:
        os.makedirs(app.instance_path)
    except OSError:
        pass

    app.cli.add_command(init_db_command)

    db.init_app(app)

    return app
```

Iniciando a aplicação



The image shows a Visual Studio Code editor window with a dark theme. The Explorer sidebar on the left shows a project named 'PROJETO_F' with files 'instance', 'application.py', 'Pipfile', and 'Pipfile.lock'. The main editor area displays the code for 'application.py'. The code defines a 'create_app' function that initializes a Flask application with a development configuration, including a secret key and a SQLite database URI. It also shows logic to load a configuration file if it exists. The bottom panel shows the 'TERMINAL' tab with the command 'flask --app application run --debug' executed. The terminal output shows the application running on http://127.0.0.1:5000 in debug mode. A green arrow points to the terminal output.

```
def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
        SECRET_KEY='dev',
        SQLALCHEMY_DATABASE_URI='sqlite:///blog.sqlite'
    )

    if test_config is None:
        # load the instance config, if it exists, when not testing
        app.config.from_pyfile('config.py', silent=True)
    else:
```

(Projeto_Flask_Blog) lucas@lucas-Inspiron-15-3520:~/Dropbox/IF_Baiano/web_II/codes/Projeto_Flask_Blog\$ flask --app application run --debug

* Serving Flask app 'application'

* 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

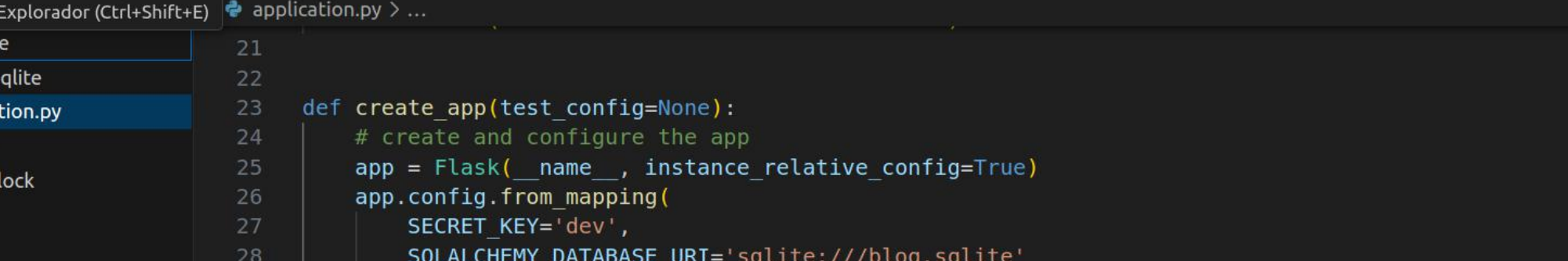
Press CTRL+C to quit

* Restarting with stat

* Debugger is active!

* Debugger PIN: 892-887-106

- Comando CLI: `flask --app application init-db`



The screenshot shows the VS Code interface. The Explorer sidebar on the left displays the project structure: `instance` (expanded), `blog.sqlite`, `application.py` (selected), `Pipfile`, and `Pipfile.lock`. A green arrow points to the `blog.sqlite` file. The main editor displays the `application.py` file with the following code:

```
21
22
23 def create_app(test_config=None):
24     # create and configure the app
25     app = Flask(__name__, instance_relative_config=True)
26     app.config.from_mapping(
27         SECRET_KEY='dev',
28         SQLALCHEMY_DATABASE_URI='sqlite:///blog.sqlite'
29     )
30
31     if test_config is None:
32         # load the instance config, if it exists, when not testing
33         app.config.from_pyfile('config.py', silent=True)
34     else:
```

The bottom panel shows the `TERMINAL` tab with the following output:

```
(Projeto_Flask_Blog) lucas@lucas-Inspiron-15-3520:~/Dropbox/IF_Baiano/web_II/codes/Projeto_Flask_Blog$ flask --app ap
plication init-db
Iniciando a base de dados...
(Projeto_Flask_Blog) lucas@lucas-Inspiron-15-3520:~/Dropbox/IF_Baiano/web_II/codes/Projeto_Flask_Blog$
```

A green arrow points to the `flask --app application init-db` command in the terminal.

Modelos de dados

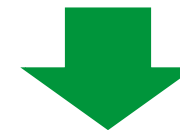
- Modelos de dados são representações estruturadas das informações que um sistema ou aplicação vai armazenar e manipular.
- Em programação com ORM (como Flask-SQLAlchemy):
 - Um modelo de dados é normalmente definido como uma classe.
 - Essa classe representa uma tabela no banco de dados.
 - Os atributos da classe representam colunas da tabela.
 - Cada instância (objeto) dessa classe representa uma linha (registro) no banco.

Criando os modelos de dados

```
application.py > ...
1  import os
2
3  import click
4  from flask import Flask, current_app
5  from flask_sqlalchemy import SQLAlchemy
6  from sqlalchemy.orm import DeclarativeBase
7
8
9  class Base(DeclarativeBase):
10 |     pass
11
12
13  db = SQLAlchemy(model_class=Base)
14
15  class User():
16 |     pass
17
18  class Post():
19 |     pass
```

Definindo o modelo User

```
CREATE TABLE user (  
    id INTEGER PRIMARY KEY,  
    username VARCHAR(80) NOT NULL UNIQUE,  
    email VARCHAR(120)  
);
```



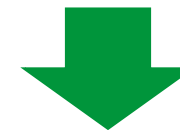
```
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
```

```
class User(db.Model):  
    id: Mapped[int] = mapped_column(primary_key=True)  
    username: Mapped[str] = mapped_column(db.String(80), unique=True, nullable=False)  
    email: Mapped[str] = mapped_column(db.String(120), nullable=True)  
  
    def __repr__(self) -> str:  
        return f"User(id={self.id!r}, email={self.email!r})"
```

Definindo o modelo User

`__repr__` é um método especial do Python que define como o objeto será representado quando for impresso ou mostrado no console/debug.

```
CREATE TABLE user (  
    id INTEGER PRIMARY KEY,  
    username VARCHAR(80) NOT NULL UNIQUE,  
    email VARCHAR(120)  
);
```




```
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
```

```
class User(db.Model):  
    id: Mapped[int] = mapped_column(primary_key=True)  
    username: Mapped[str] = mapped_column(db.String(80), unique=True, nullable=False)  
    email: Mapped[str] = mapped_column(db.String(120), nullable=True)  
  
    def __repr__(self) -> str:  
        return f"User(id={self.id!r}, email={self.email!r})"
```

Definindo o modelo Post

```
from datetime import datetime
from sqlalchemy import func
```

```
CREATE TABLE post (
    id INTEGER PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    body TEXT NOT NULL,
    created TIMESTAMP DEFAULT
CURRENT_TIMESTAMP,
    author_id INTEGER,
    FOREIGN KEY (author_id) REFERENCES user(id)
);
```

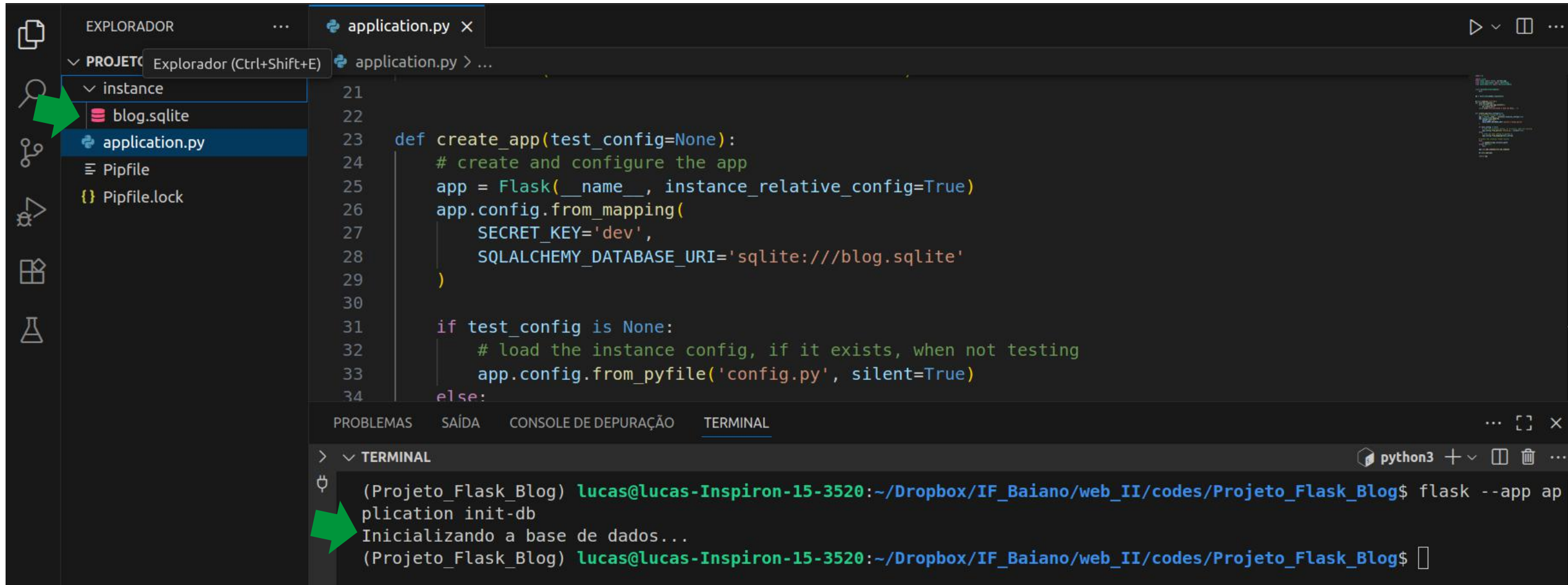


```
class Post(db.Model):
    id: Mapped[int] = mapped_column(primary_key=True)
    title: Mapped[str] = mapped_column(db.String(100), nullable=False)
    body: Mapped[str] = mapped_column(db.Text, nullable=False)
    created: Mapped[datetime] = mapped_column(server_default=func.now())
    author_id: Mapped[int] = mapped_column(db.ForeignKey("user.id"))

    def __repr__(self) -> str:
        return f"Post(id={self.id!r}, title={self.title!r}, author_id={self.author_id!r})"
```


Criando o banco de dados

- Comando CLI: `flask --app application init-db`




The screenshot shows the Visual Studio Code editor with a project named 'Projeto_Flask_Blog'. The Explorer sidebar on the left shows the file structure: 'instance' folder, 'blog.sqlite' file, and 'application.py' file. The 'application.py' file is open in the editor, showing the following code:

```
21
22
23 def create_app(test_config=None):
24     # create and configure the app
25     app = Flask(__name__, instance_relative_config=True)
26     app.config.from_mapping(
27         SECRET_KEY='dev',
28         SQLALCHEMY_DATABASE_URI='sqlite:///blog.sqlite'
29     )
30
31     if test_config is None:
32         # load the instance config, if it exists, when not testing
33         app.config.from_pyfile('config.py', silent=True)
34     else:
```

The terminal at the bottom shows the command `flask --app application init-db` being executed, resulting in the output: `Inicializando a base de dados...`.

Acessando as tabelas do banco



SQLite Viewer

Florian Klampfer | 1.928.769 | ★★★★★ (64)

SQLite Viewer for VS Code

[Instalar](#) ☒ Atualização Automática ⚙️

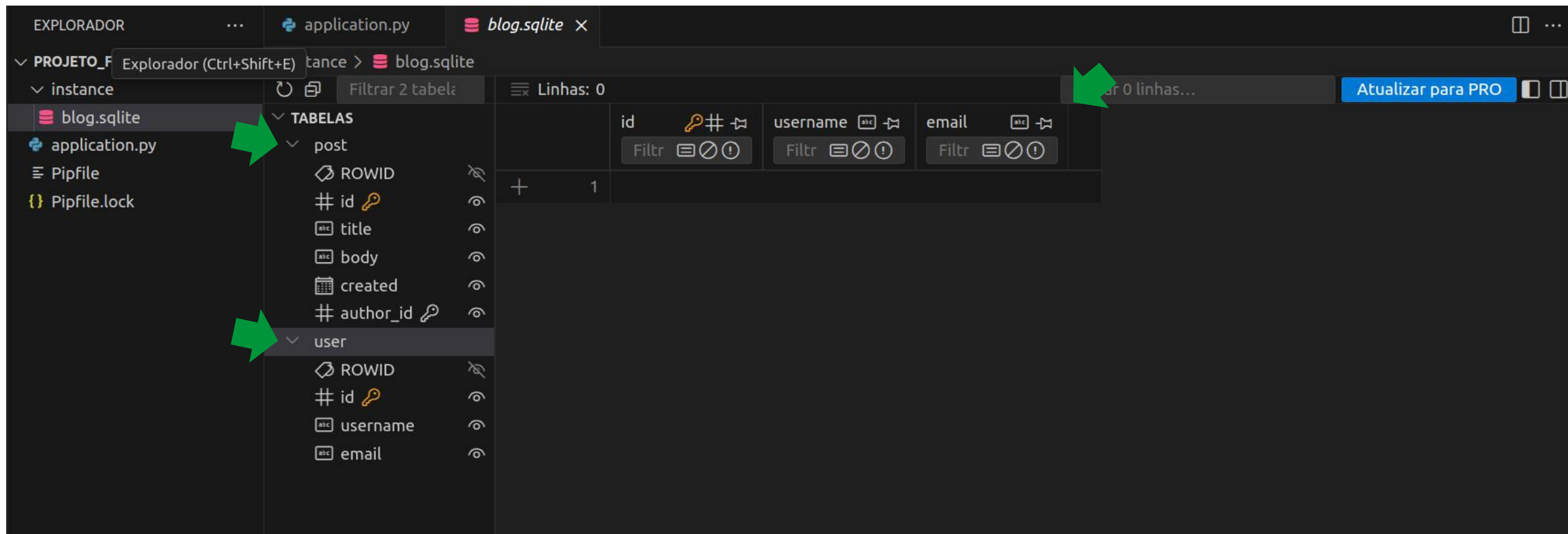
DETALHES RECURSOS LOG DE MUDANÇAS

SQLite Viewer for VS Code

Marketplace

Identifi... gwtel...

Acessando as tabelas do banco



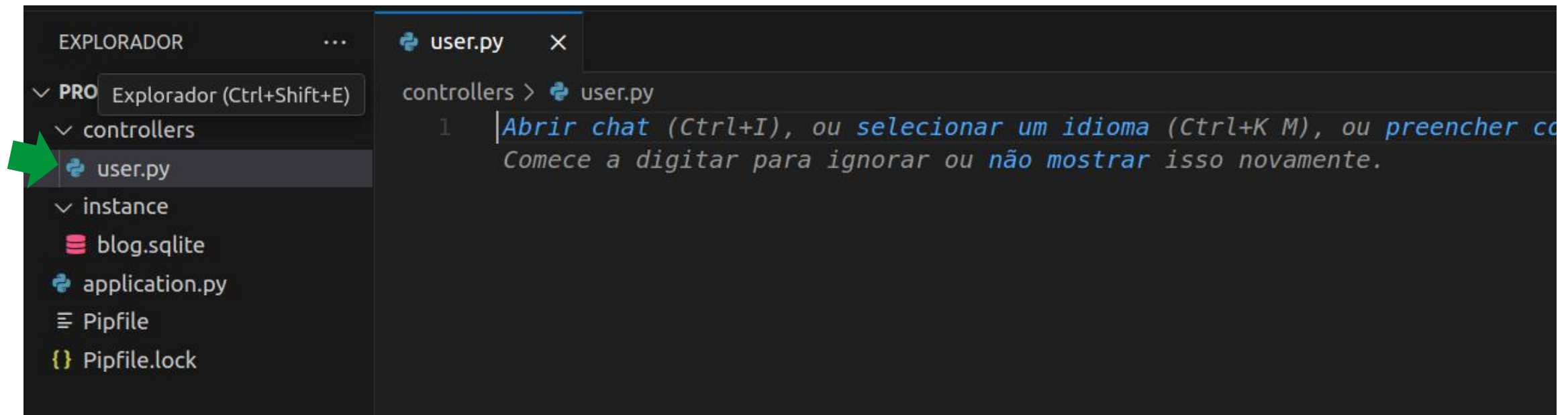
The screenshot shows a database management tool interface. The top bar includes tabs for 'EXPLORADOR', 'application.py', and 'blog.sqlite'. The left sidebar shows a tree view of the project structure, including 'PROJETO_F', 'instance', and 'blog.sqlite'. The main area displays the 'blog.sqlite' database with two tables: 'post' and 'user'. The 'post' table has columns: ROWID, id (primary key), title, body, created, and author_id (foreign key). The 'user' table has columns: ROWID, id (primary key), username, and email. The interface also shows a search bar, a filter button, and a 'Linhas: 0' indicator. A green arrow points to the 'blog.sqlite' tab in the sidebar, and another green arrow points to the 'user' table in the table list.

id	username	email
1		

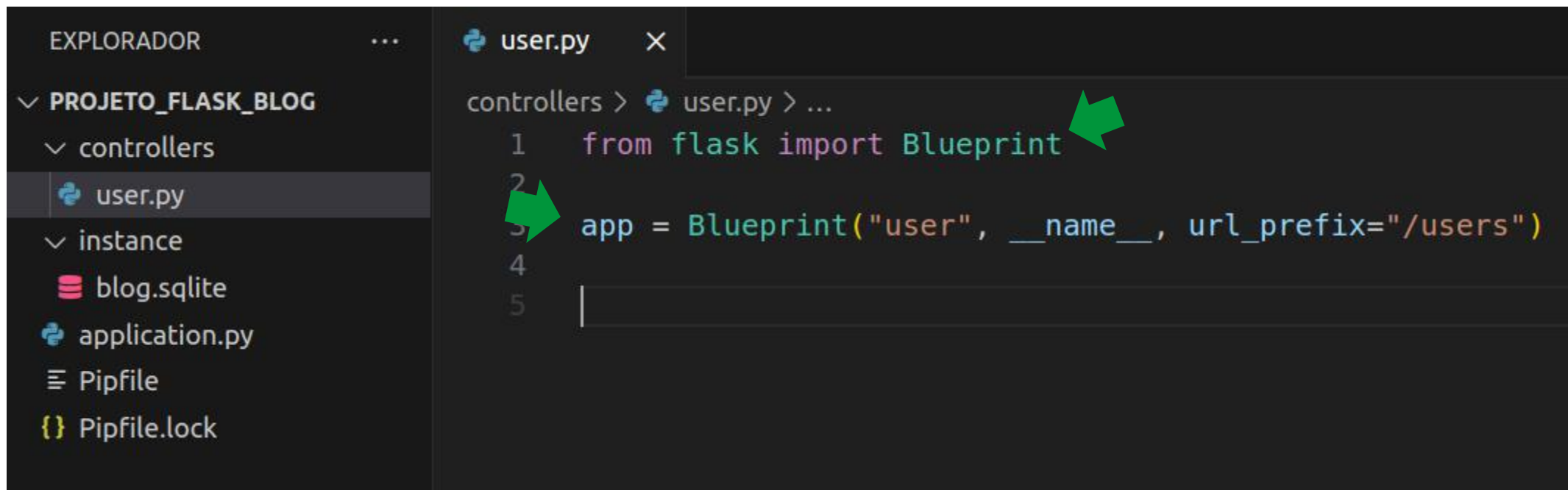
Definindo as rotas e controladores usando Blueprints

- No Flask, Blueprints são uma forma de organizar e modularizar a aplicação. Eles permitem dividir a aplicação em componentes reutilizáveis e independentes, cada um com suas próprias rotas e controladores.
- Os Blueprints no Flask permitem que você organize rotas separadas por funcionalidade, e isso frequentemente coincide com a separação por modelos.

Criando o controlador de usuários



Criando o controlador de usuários

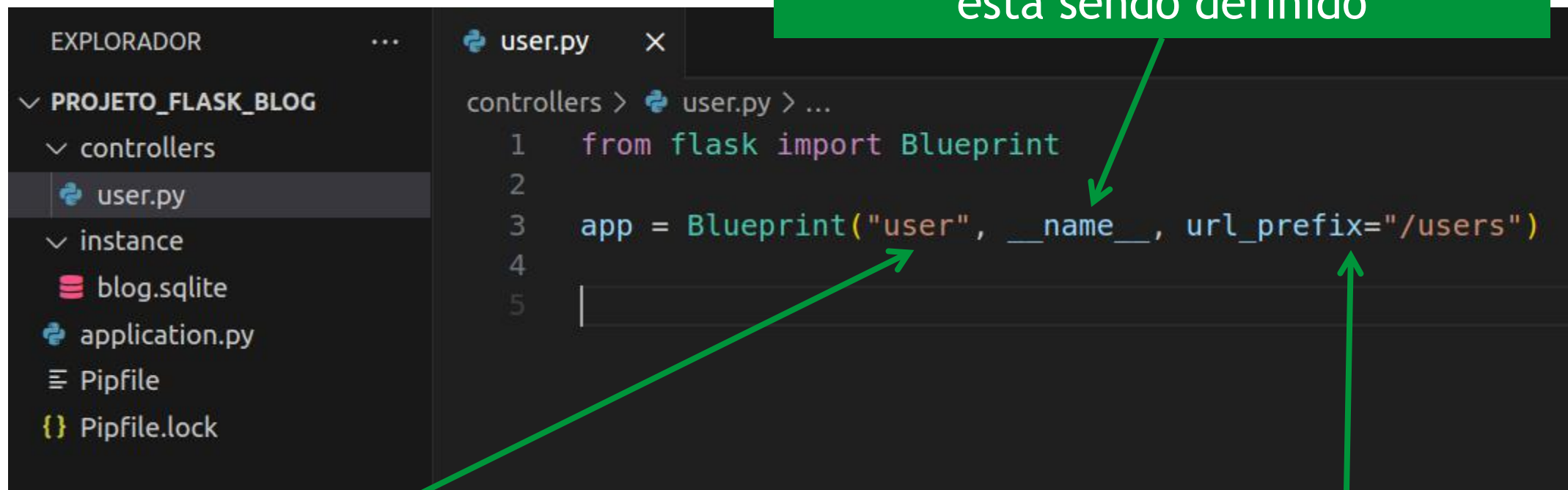


The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure for 'PROJETO_FLASK_BLOG', including a 'controllers' folder and a 'user.py' file. The main editor window shows the 'user.py' file with the following code:

```
controllers > user.py > ...  
1  from flask import Blueprint  
2  
3  app = Blueprint("user", __name__, url_prefix="/users")  
4  
5  |
```

Two green arrows highlight the code: one points to the 'Blueprint' import on line 1, and the other points to the 'app = Blueprint' assignment on line 3.

Criando o controlador de usuários



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar shows a project named 'PROJETO_FLASK_BLOG' with a 'controllers' folder containing 'user.py'. The main editor window shows the code in 'user.py' with the following content:

```
1 from flask import Blueprint
2
3 app = Blueprint("user", __name__, url_prefix="/users")
4
5
```

Three green arrows point from text boxes to specific parts of the code: one to the 'user' string in the Blueprint constructor, one to the '__name__' argument, and one to the '/users' string in the url_prefix argument.

Nome do módulo onde o blueprint está sendo definido

Nome interno do blueprint (pode ser usado para referenciar ou registrar)

Define o prefixo de rota para todas as rotas registradas nesse blueprint

Criando o controlador de usuários

```
app.cli.add_command(init_db_command)

db.init_app(app)

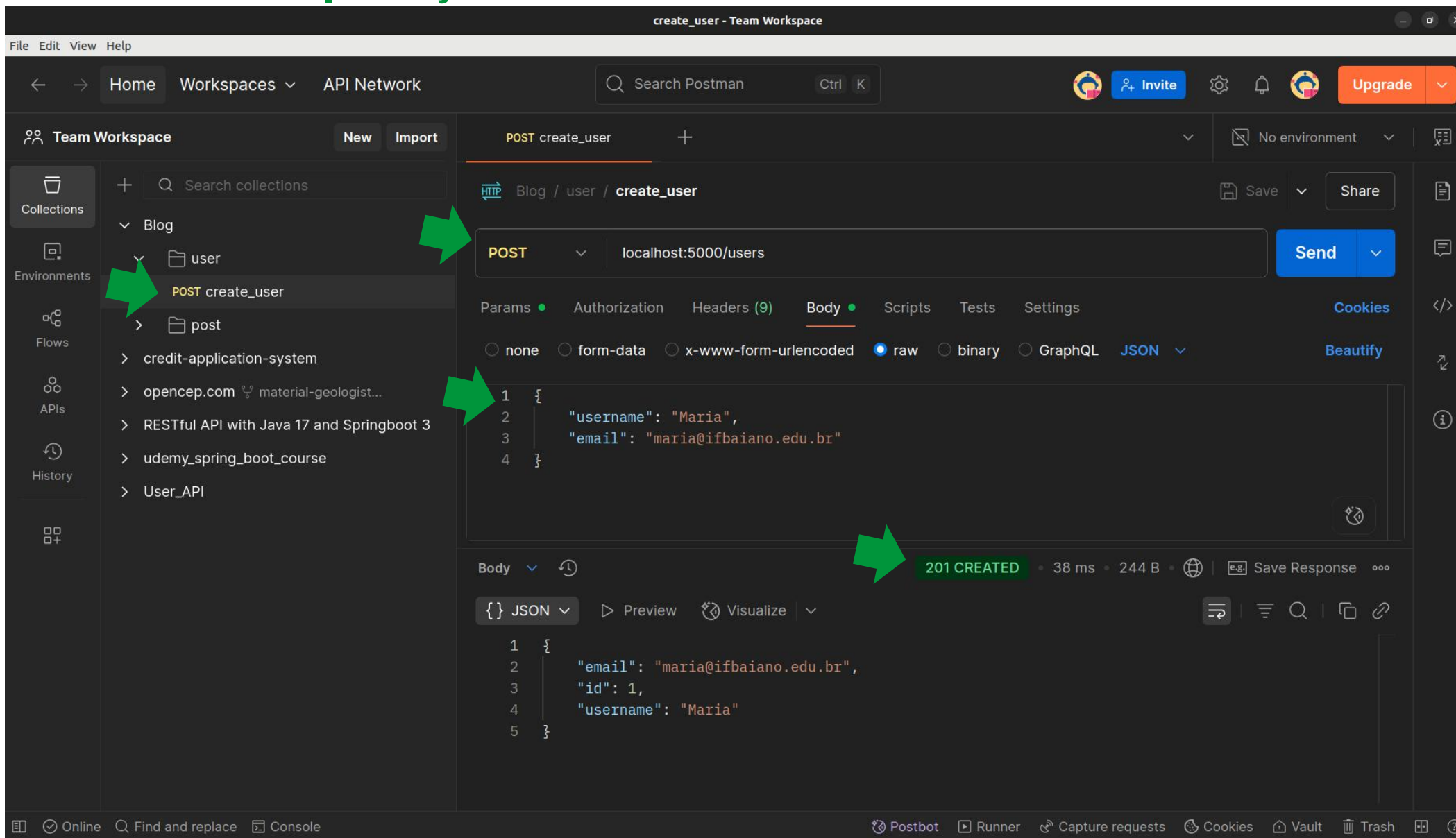
from controllers import user
app.register_blueprint(user.app)

return app
```

Definindo a operação de CREATE

```
controllers > user.py > ...
1  from flask import Blueprint, request
2  from application import User, db
3  from http import HTTPStatus
4
5  app = Blueprint("user", __name__, url_prefix="/users")
6
7  @app.post("/")
8  def create_user():
9      data = request.get_json()
10
11      if not data or "username" not in data:
12          return {"error": "username é obrigatório"}, HTTPStatus.BAD_REQUEST
13
14      email = data.get("email")
15
16      user = User(username=data["username"], email=email)
17      db.session.add(user)
18      db.session.commit()
19
20      return {
21          "id": user.id,
22          "username": user.username,
23          "email": user.email
24      }, HTTPStatus.CREATED
```

Testando a operação de CREATE no Postman



The screenshot displays the Postman interface for a workspace named "create_user - Team Workspace". The left sidebar shows a collection structure: "Blog" > "user" > "POST create_user". The main panel shows the details of the "POST create_user" request. The URL is "localhost:5000/users". The request body is a JSON object:

```
{  "username": "Maria",  "email": "maria@ifbaiano.edu.br"}
```

. The response status is "201 CREATED" with a response time of 38 ms and a body size of 244 B. The response body is a JSON object:

```
{  "email": "maria@ifbaiano.edu.br",  "id": 1,  "username": "Maria"}
```

. Three green arrows highlight the collection path, the request body, and the response status/body.

Team Workspace

POST create_user

Blog / user / create_user

POST localhost:5000/users

Params Authorization Headers (9) Body Scripts Tests Settings Cookies Beautify

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

```
1 {
2   "username": "Maria",
3   "email": "maria@ifbaiano.edu.br"
4 }
```

Body

JSON Preview Visualize

```
1 {
2   "email": "maria@ifbaiano.edu.br",
3   "id": 1,
4   "username": "Maria"
5 }
```

201 CREATED • 38 ms • 244 B • Save Response

Definindo as operações de READ



```
@app.get("/")
def list_users():
    query = db.select(User)
    result = db.session.execute(query)
    users = result.scalars().all()

    return [
        {
            "id": user.id,
            "username": user.username,
            "email": user.email
        }
        for user in users
    ], HTTPStatus.OK
```

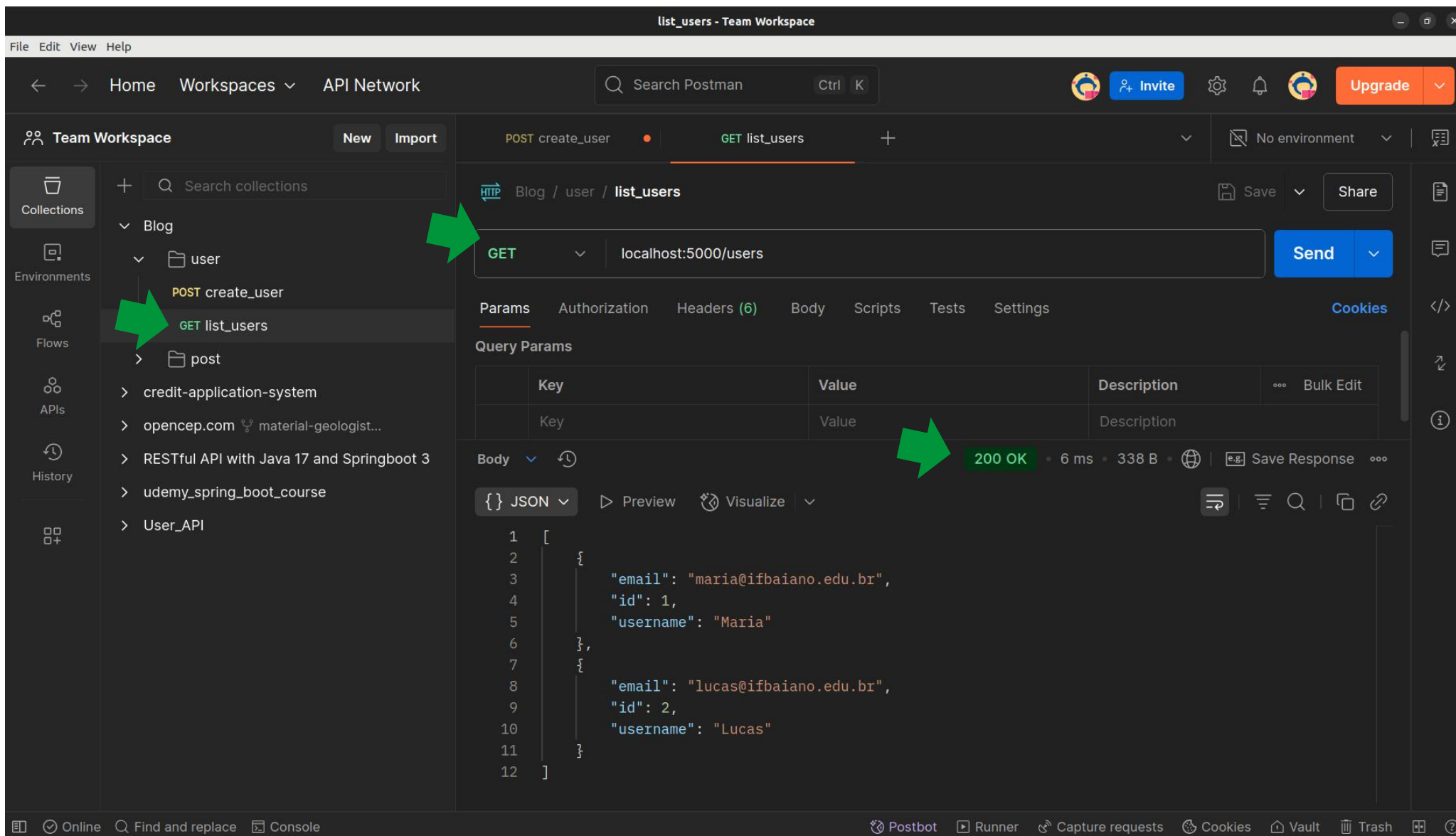

Definindo as operações de READ



```
@app.get("/<int:user_id>")
def get_user(user_id):
    user = db.get_or_404(User, user_id)

    return {
        "id": user.id,
        "username": user.username,
        "email": user.email
    }, HTTPStatus.OK
```

Testando as operações de READ no Postman



The screenshot displays the Postman Team Workspace interface. On the left sidebar, the 'Collections' panel shows a tree structure with 'Blog' > 'user' > 'GET list_users' selected. A green arrow points from this collection item to the main workspace. The main workspace shows a GET request to 'localhost:5000/users'. A green arrow points from the 'GET' method dropdown to the URL. Below the URL bar, the 'Params' tab is active, showing a table with headers 'Key', 'Value', and 'Description'. A green arrow points from the 'Value' header to the '200 OK' status in the response section. The response section shows a status of '200 OK', a time of '6 ms', and a size of '338 B'. The response body is displayed in JSON format, showing an array of two user objects.

list_users - Team Workspace

File Edit View Help

Home Workspaces API Network

Search Postman Ctrl K

Team Workspace New Import

POST create_user GET list_users

Blog / user / list_users

GET localhost:5000/users

Send

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

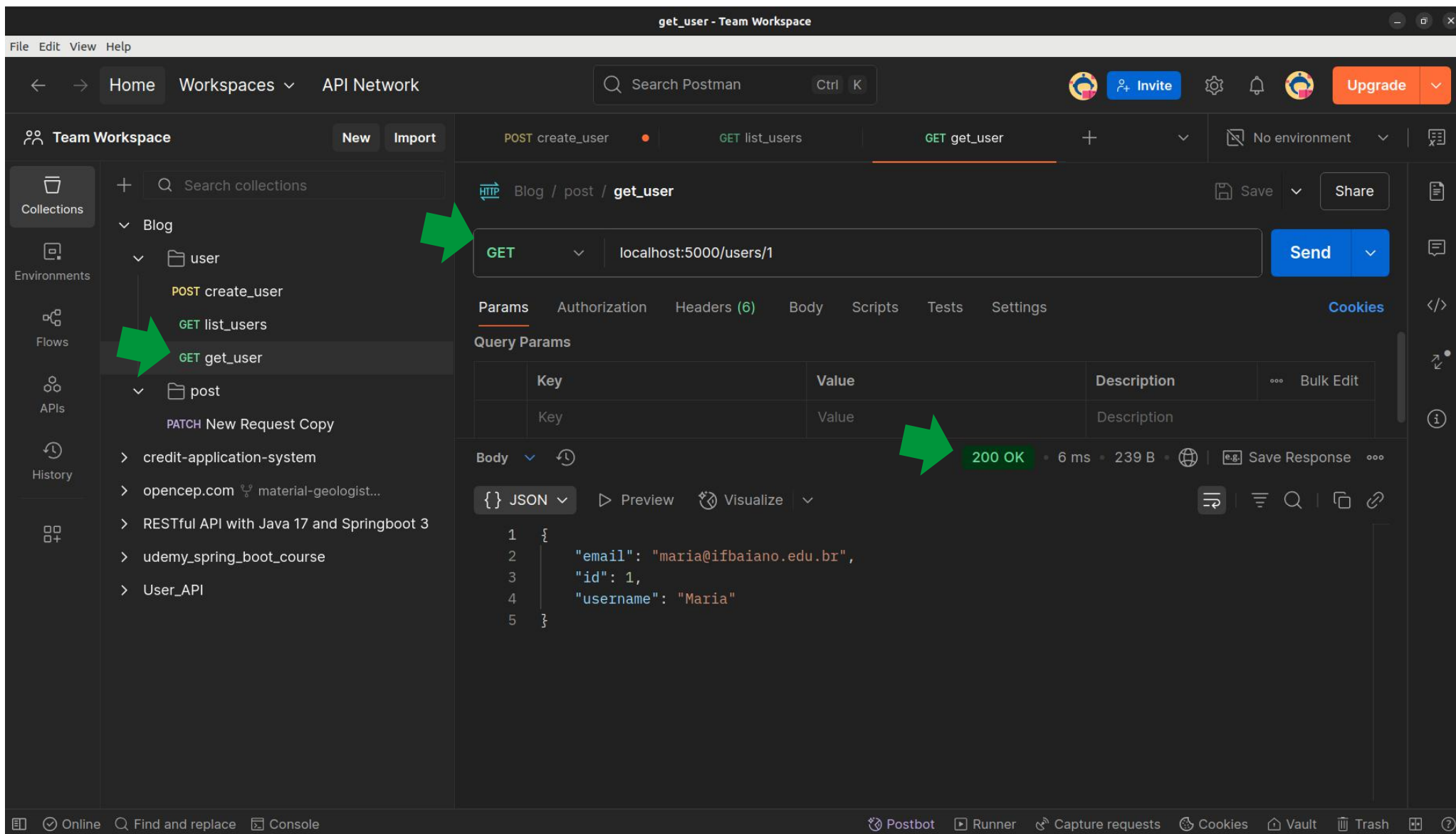
Key	Value	Description
Key	Value	Description

Body JSON Preview Visualize

200 OK • 6 ms • 338 B • Save Response

```
1 [
2   {
3     "email": "maria@ifbaiano.edu.br",
4     "id": 1,
5     "username": "Maria"
6   },
7   {
8     "email": "lucas@ifbaiano.edu.br",
9     "id": 2,
10    "username": "Lucas"
11  }
12 ]
```

Testando as operações de READ no Postman

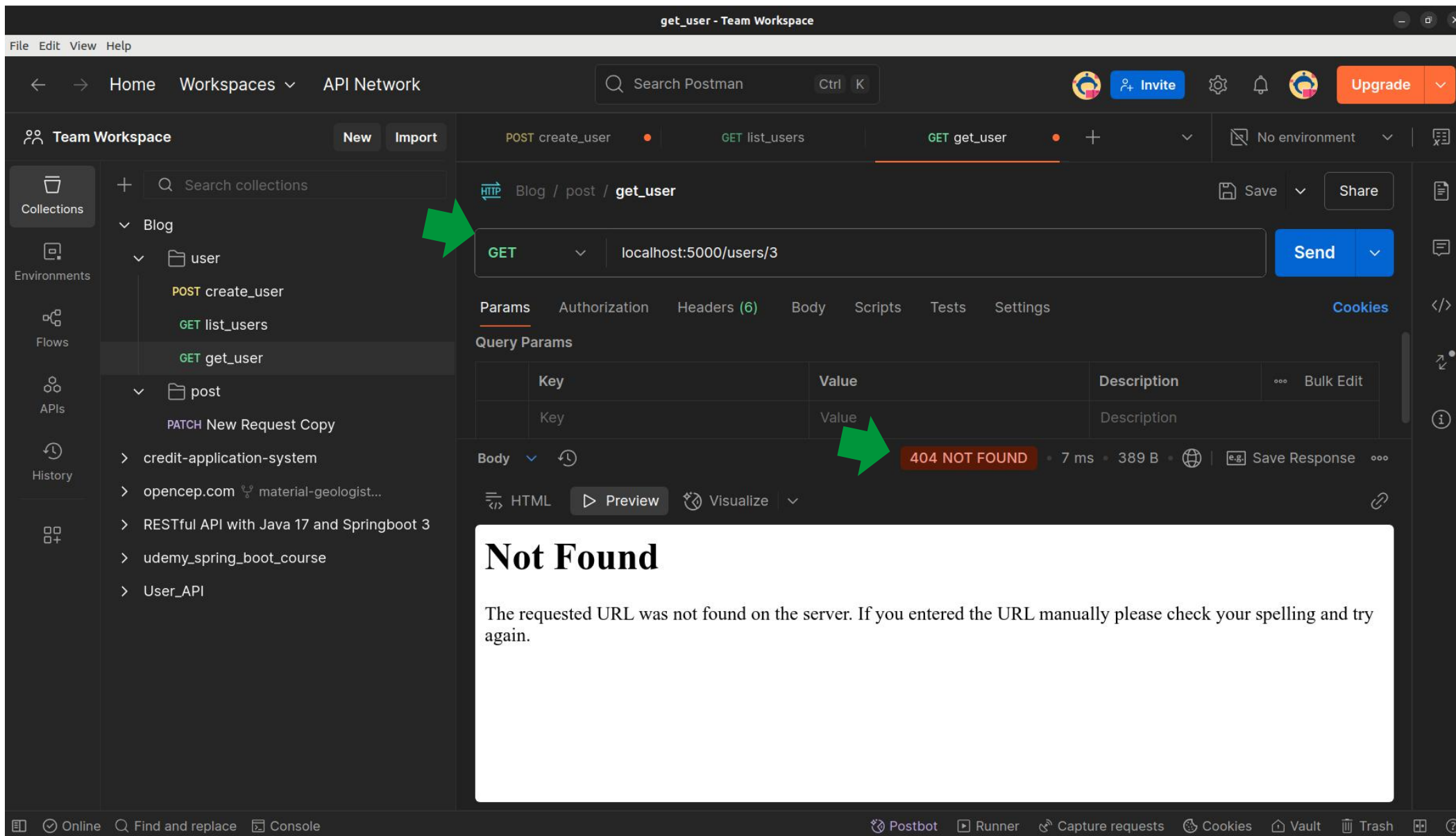


The screenshot displays the Postman Team Workspace interface. The left sidebar shows the 'Collections' panel with a tree structure: 'Blog' > 'user' > 'GET get_user'. A green arrow points to this collection. The main panel shows the selected request: 'GET localhost:5000/users/1'. A green arrow points to the 'GET' method dropdown. Below the URL bar, the 'Params' tab is active, showing a table for 'Query Params' with columns 'Key', 'Value', and 'Description'. A green arrow points to the 'Body' tab, which shows a JSON response. The response status is '200 OK' with a green arrow pointing to it. The JSON body is:

```
{
  "email": "maria@ifbaiano.edu.br",
  "id": 1,
  "username": "Maria"
}
```

The bottom status bar shows '200 OK', '6 ms', '239 B', and a 'Save Response' button.

Testando as operações de READ no Postman



The screenshot shows the Postman Team Workspace interface. On the left sidebar, the 'Collections' panel is expanded, showing a 'Blog' collection with a 'user' folder containing 'POST create_user', 'GET list_users', and 'GET get_user'. The 'GET get_user' request is selected. The main panel shows the request details: Method 'GET', URL 'localhost:5000/users/3'. Below the URL bar, the 'Params' tab is active, showing a table with headers 'Key', 'Value', and 'Description'. The 'Body' tab is also visible, showing a '404 NOT FOUND' response. A green arrow points from the 'GET get_user' request in the sidebar to the 'GET' method in the main panel. Another green arrow points from the '404 NOT FOUND' response in the 'Body' tab to the '404 NOT FOUND' text in the response body.

get_user - Team Workspace

File Edit View Help

Home Workspaces API Network

Search Postman Ctrl K

Team Workspace New Import

Search collections

Blog

user

POST create_user

GET list_users

GET get_user

post

PATCH New Request Copy

credit-application-system

opencep.com material-geologist...

RESTful API with Java 17 and Springboot 3

udemy_spring_boot_course

User_API

HTTP Blog / post / get_user

Save Share

Send

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body

404 NOT FOUND • 7 ms • 389 B • Save Response

HTML Preview Visualize

Not Found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

Online Find and replace Console Postbot Runner Capture requests Cookies Vault Trash

Exercícios

- Implemente as operações de PUT, PATCH e DELETE e teste as operações no Postman.

Dúvidas



PROGRAMAÇÃO WEB II

Curso Técnico Integrado em Informática
Lucas Sampaio Leite

