PROGRAMAÇÃO WEB II

Curso Técnico Integrado em Informática

Lucas Sampaio Leite

INSTITUTO FEDERAL
Baiano

Persistência de dados com o Flask

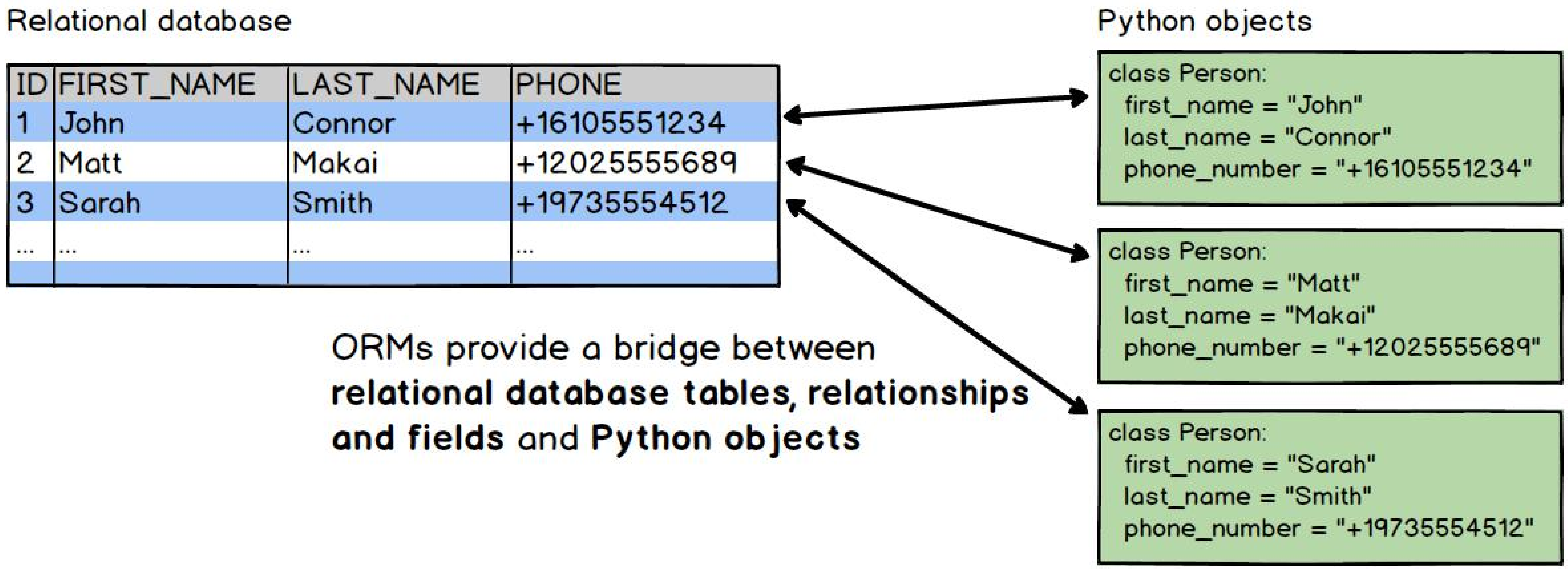
INSTITUTO FEDERAL Baiano

- 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"
```

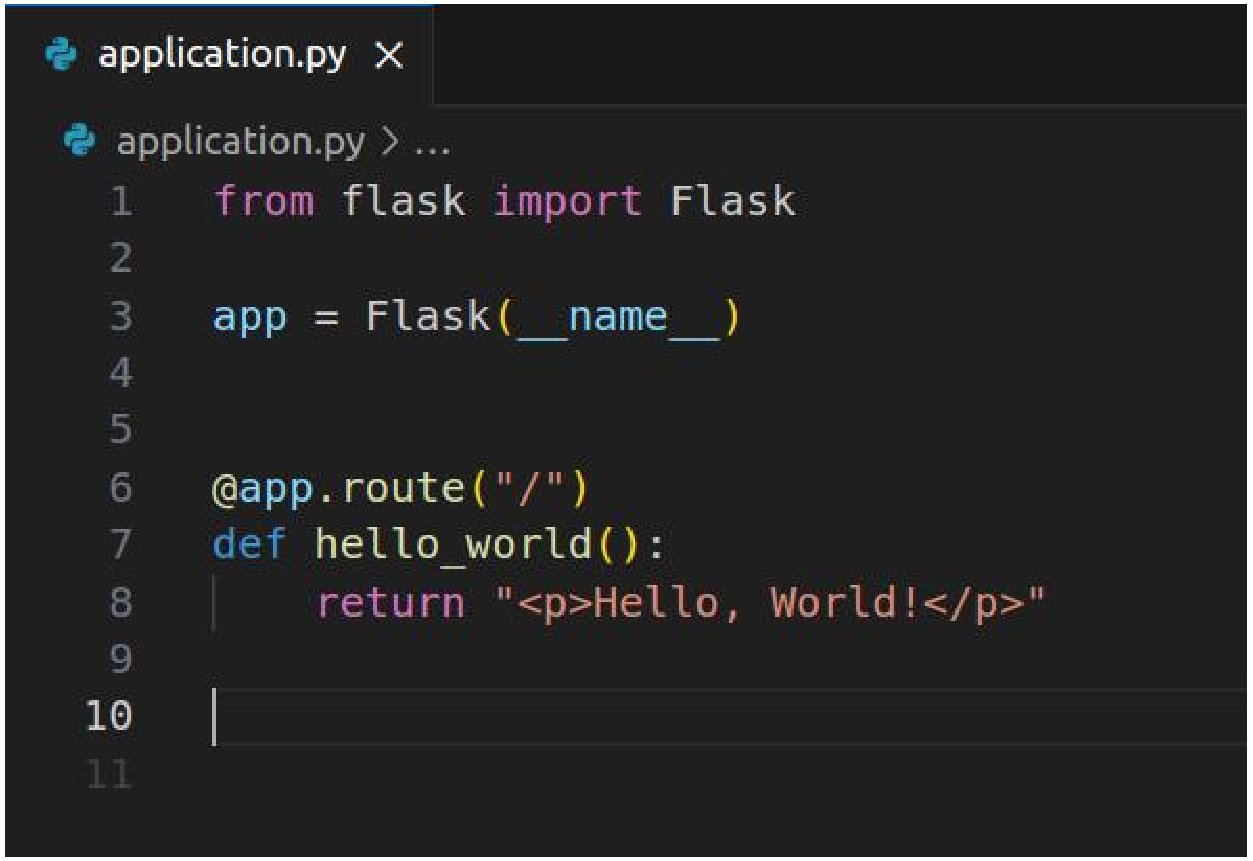
```
class Person:
  first_name = "Matt"
  last_name = "Makai"
  phone_number = "+12025555689"
```

```
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 ×

application.py > ...

```python
from flask import Flask

app = Flask(__name__)


@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"

```

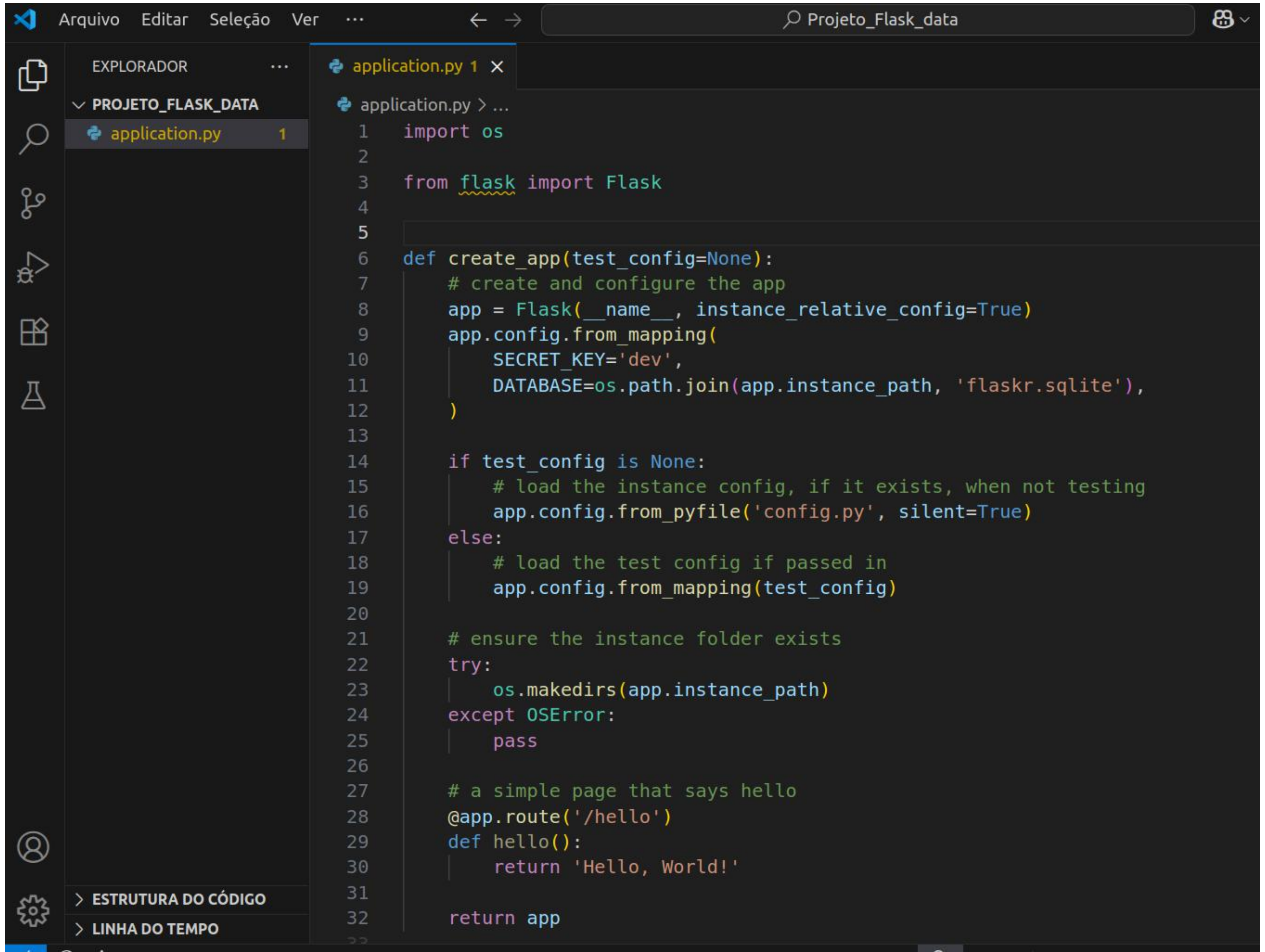
Criando uma nova aplicação Flask

INSTITUTO FEDERAL Baiano

- 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/

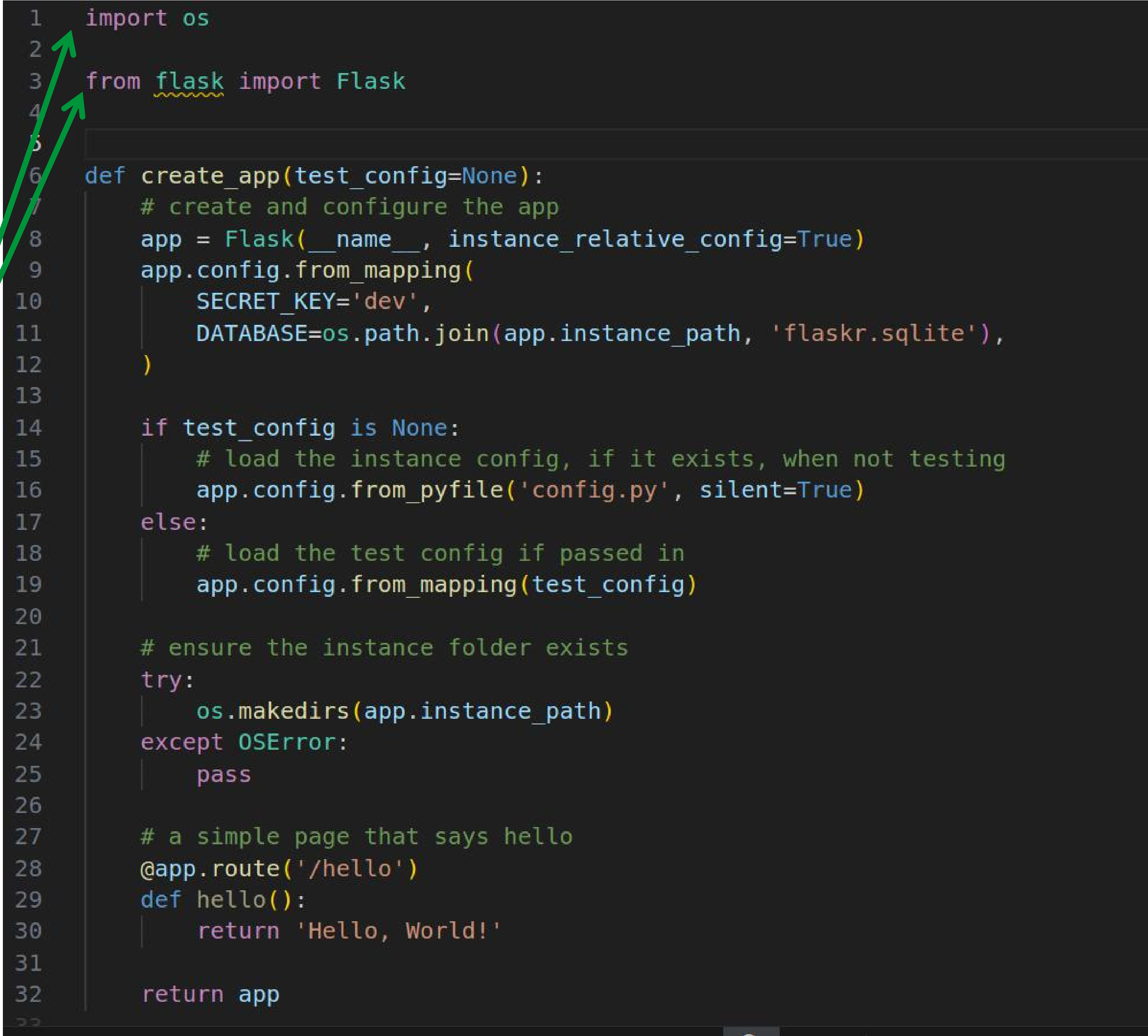


```python
import os

from flask import 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',
        DATABASE=os.path.join(app.instance_path, 'flaskr.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

    # a simple page that says hello
    @app.route('/hello')
    def hello():
        return 'Hello, World!'

    return app
```

Criando um novo app

os será usado para manipular caminhos de arquivos e pastas;

Flask é a classe principal do framework

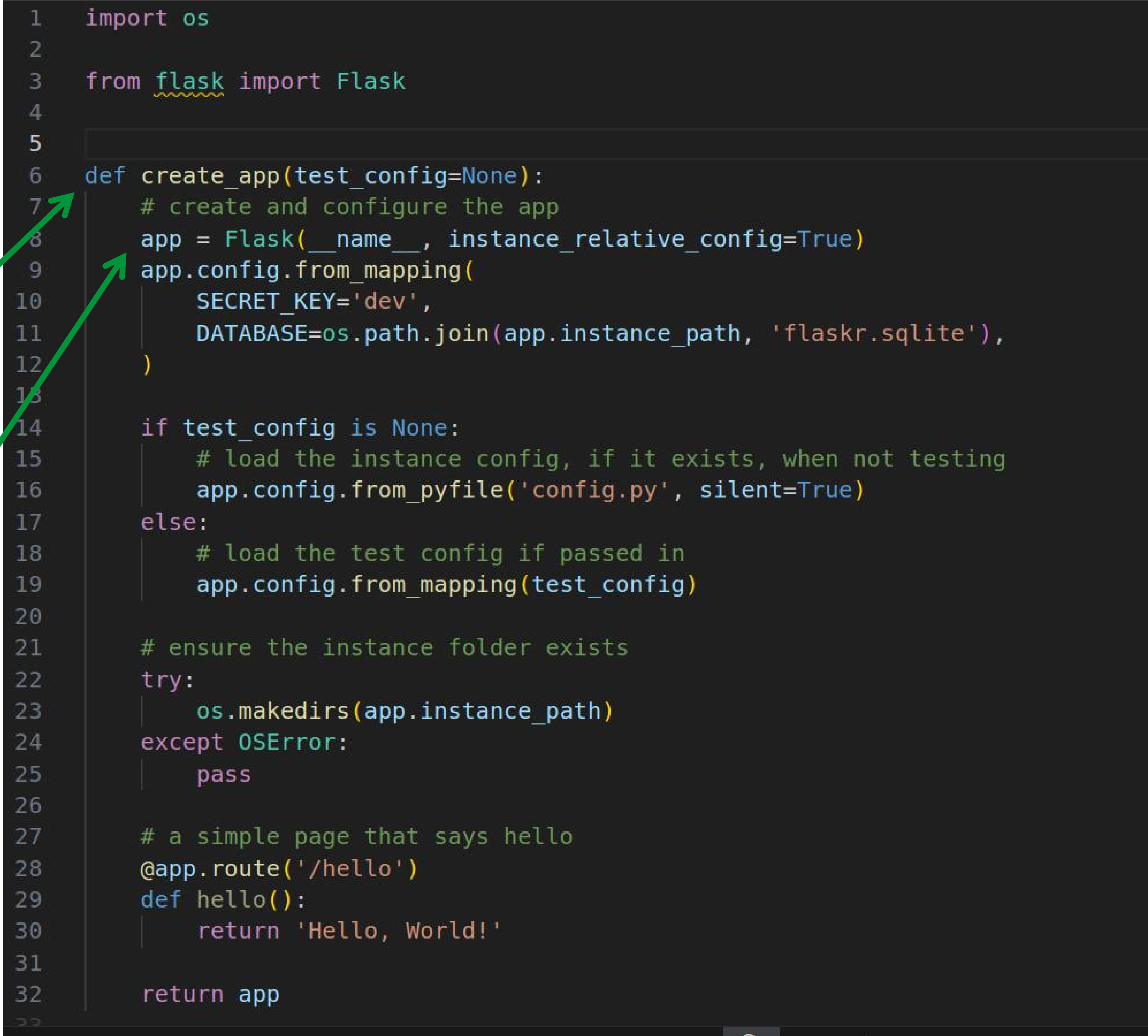


```python
import os

from flask import 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',
        DATABASE=os.path.join(app.instance_path, 'flaskr.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

    # a simple page that says hello
    @app.route('/hello')
    def hello():
        return 'Hello, World!'

    return app
```

Criando um novo app

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

Cria uma instância da aplicação Flask.



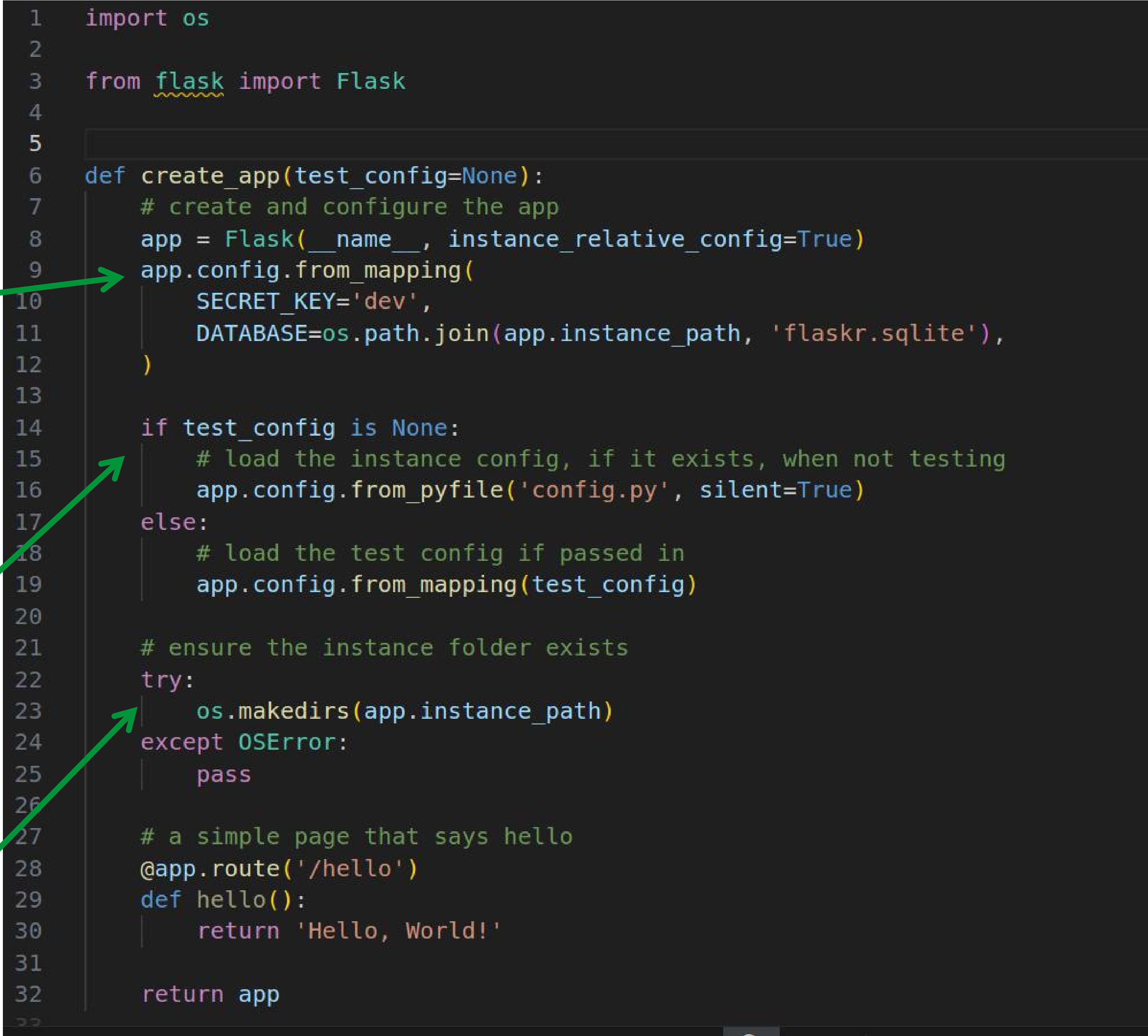
```python
import os

from flask import 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',
        DATABASE=os.path.join(app.instance_path, 'flaskr.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

    # a simple page that says hello
    @app.route('/hello')
    def hello():
        return 'Hello, World!'

    return app
```

Criando um novo app

Define a configuração padrão da aplicação

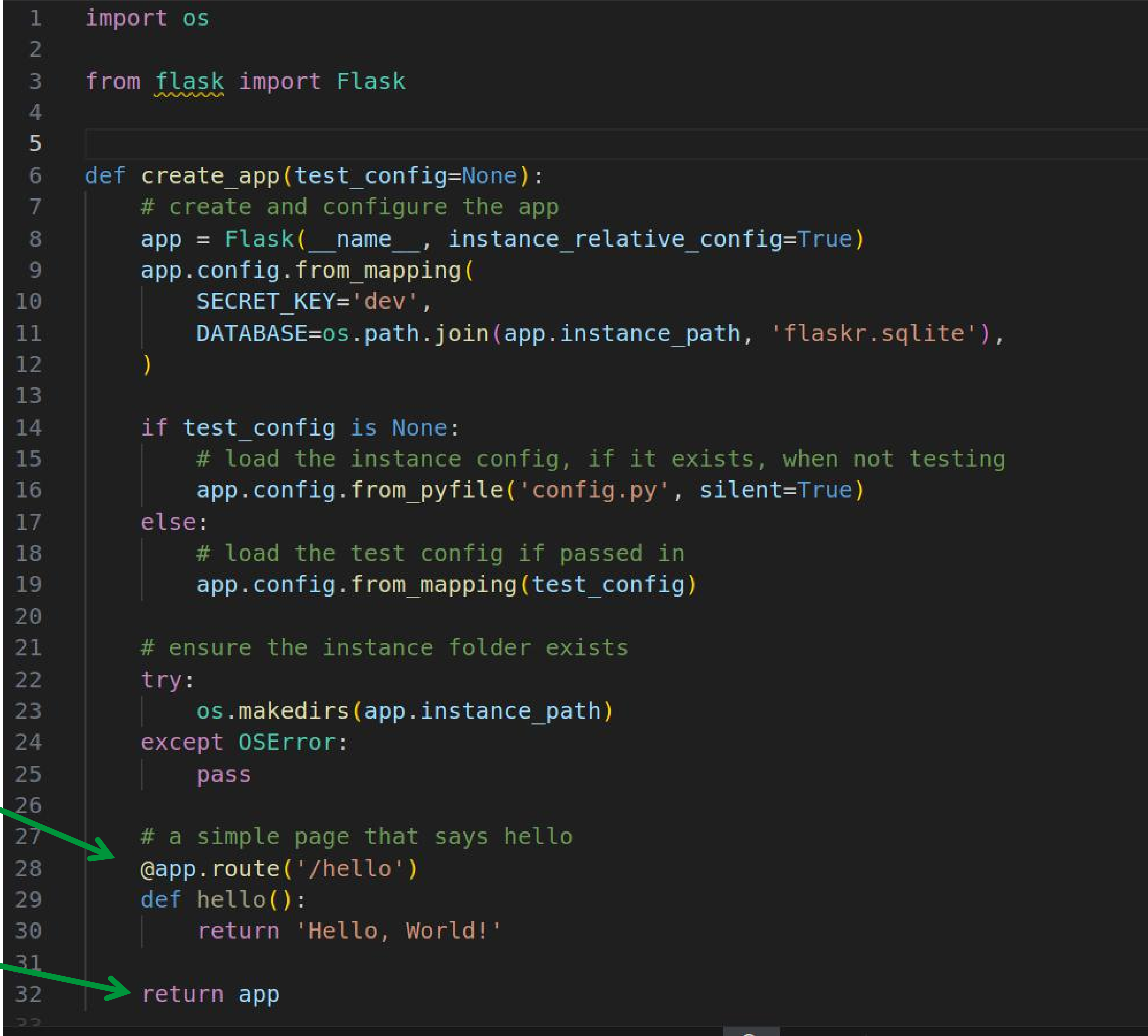
Se não estivermos em modo de teste, tenta carregar configurações adicionais a partir do arquivo instance/config.py

Garante que a pasta instance/ existe



```python
import os

from flask import 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',
        DATABASE=os.path.join(app.instance_path, 'flaskr.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

    # a simple page that says hello
    @app.route('/hello')
    def hello():
        return 'Hello, World!'

    return app
```

Criando um novo app



```python
import os

from flask import 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',
        DATABASE=os.path.join(app.instance_path, 'flaskr.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

    # a simple page that says hello
    @app.route('/hello')
    def hello():
        return 'Hello, World!'

    return app
```

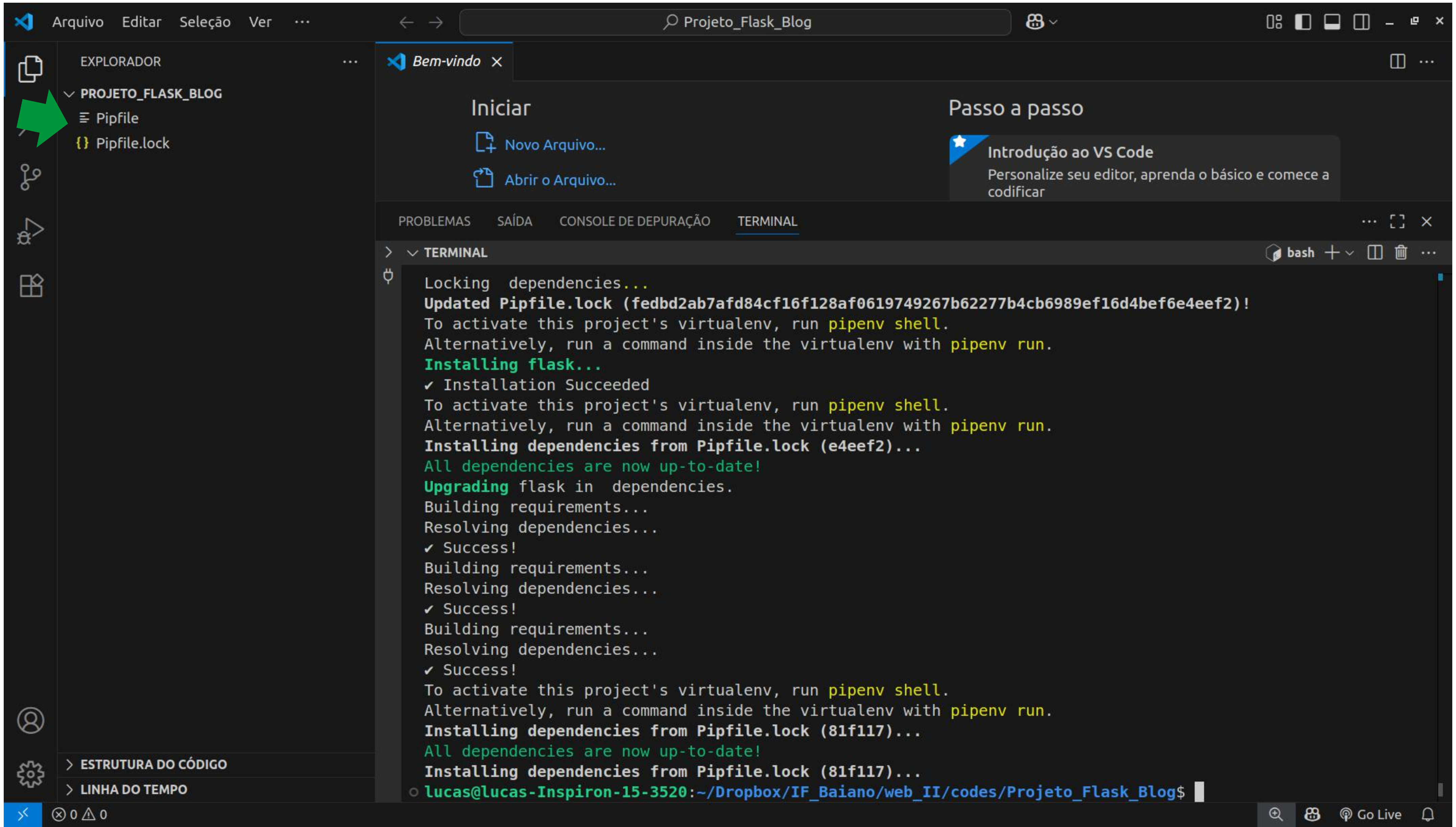
Cria uma rota simples

Retorna o app configurado

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

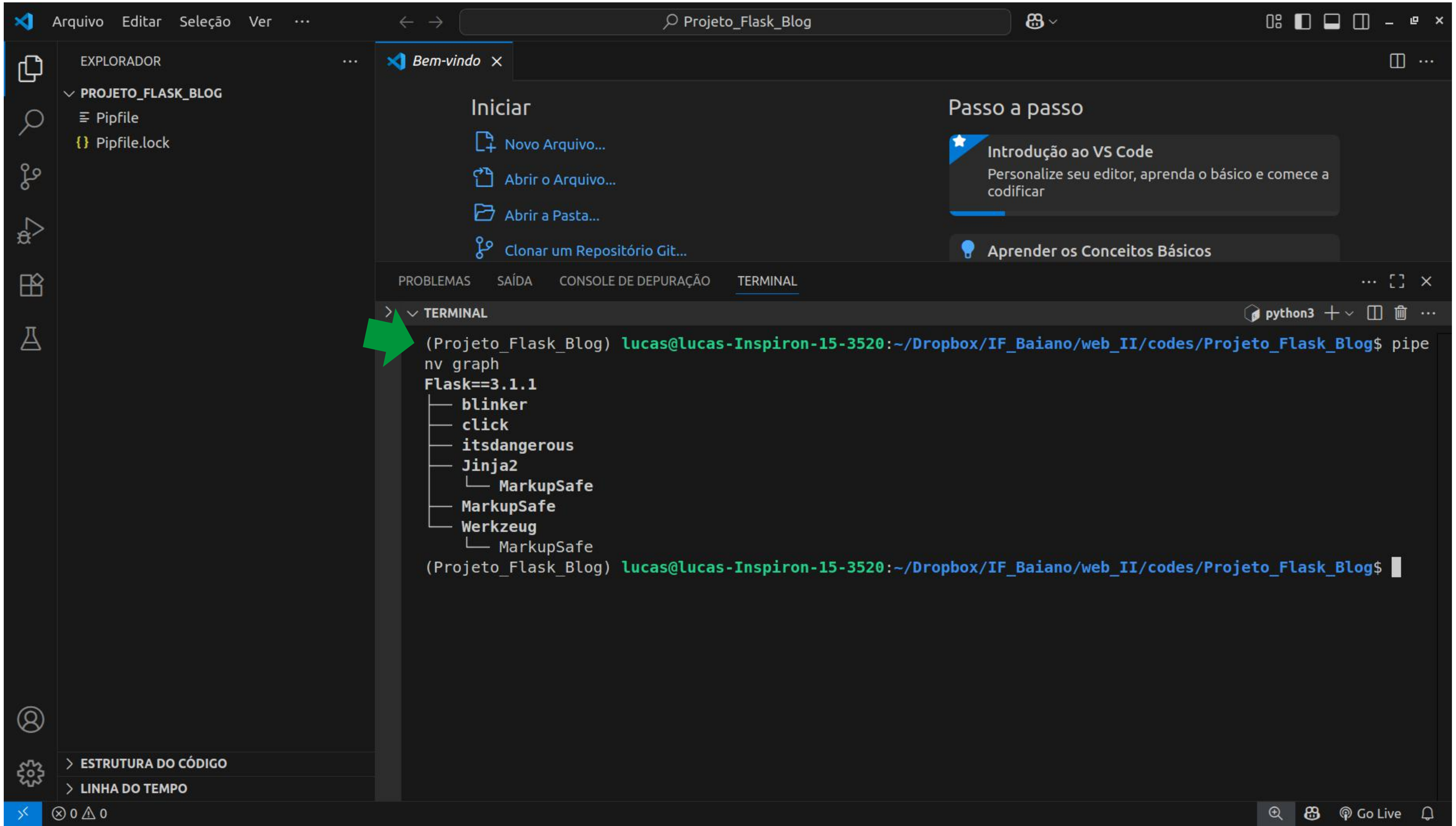
Criando uma nova aplicação Flask



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

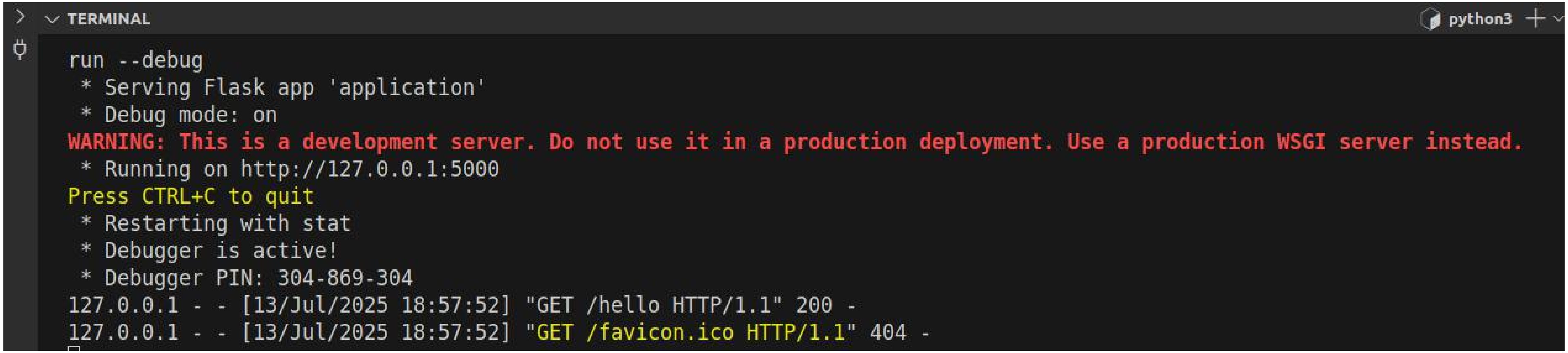
INSTITUTO FEDERAL
Baiano

Criando uma nova aplicação Flask

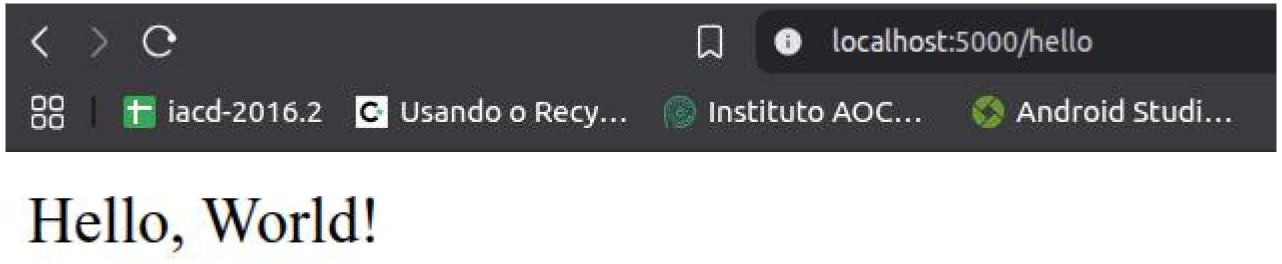


```
(Projeto_Flask_Blog) lucas@lucas-Inspiron-15-3520:~/Dropbox/IF_Baiano/web_II/codes/Projeto_Flask_Blog$ pipe
nv graph
Flask==3.1.1
├── blinker
├── click
├── itsdangerous
├── Jinja2
│   └── MarkupSafe
├── MarkupSafe
└── Werkzeug
    └── MarkupSafe
(Projeto_Flask_Blog) lucas@lucas-Inspiron-15-3520:~/Dropbox/IF_Baiano/web_II/codes/Projeto_Flask_Blog$
```

Testando a rota criada

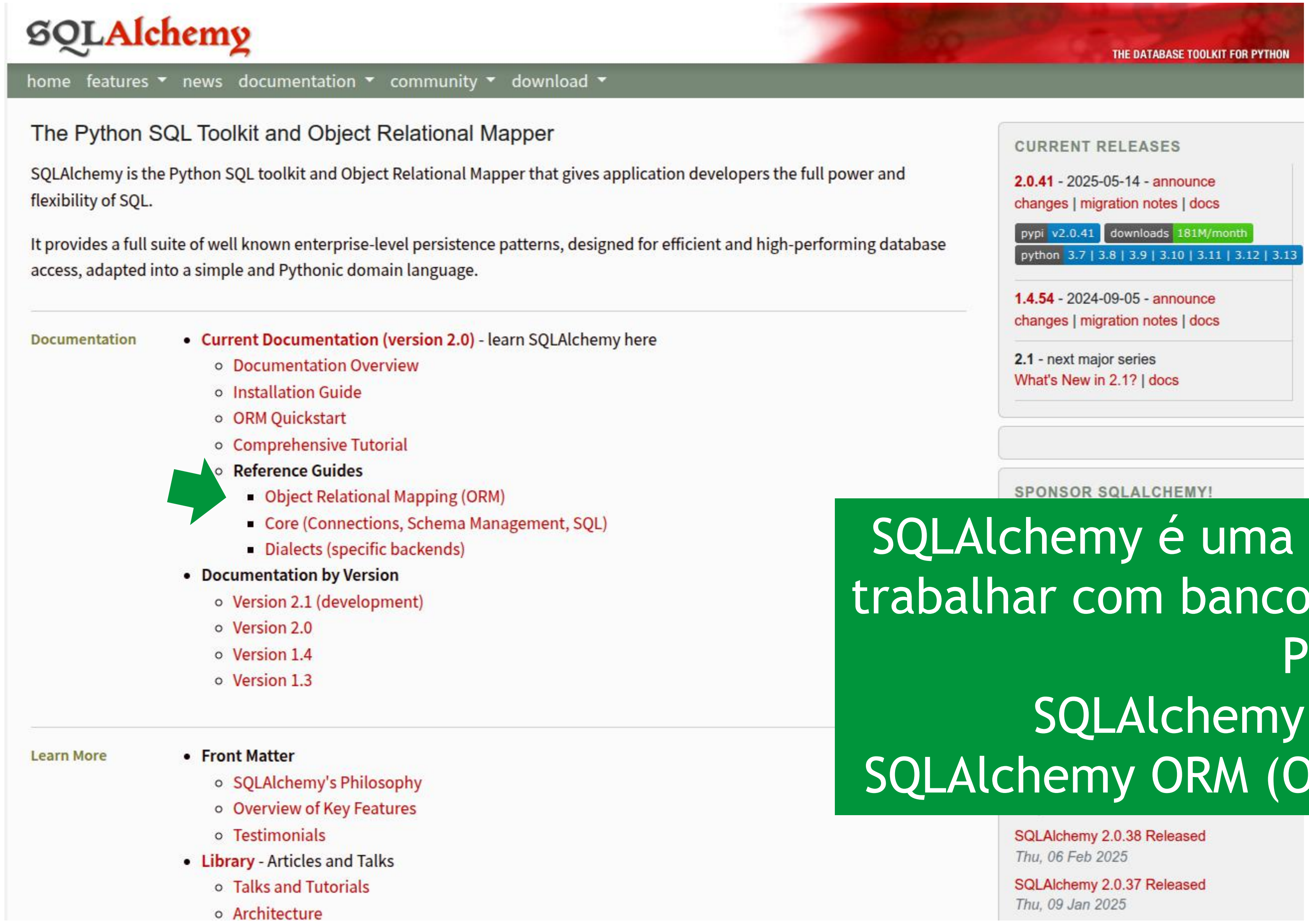


```
run --debug
 * Serving Flask app 'application'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger 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


SQLAlchemy
THE DATABASE TOOLKIT FOR PYTHON
home features news documentation community download

The Python SQL Toolkit and Object Relational Mapper

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

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.

Documentation

- 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
  - Version 1.3

Learn More

- Front Matter
  - SQLAlchemy's Philosophy
  - Overview of Key Features
  - Testimonials
- Library - Articles and Talks
  - Talks and Tutorials
  - Architecture

CURRENT RELEASES

2.0.41 - 2025-05-14 - announce
changes | migration notes | docs

pypi v2.0.41 downloads 181M/month
python 3.7 | 3.8 | 3.9 | 3.10 | 3.11 | 3.12 | 3.13

1.4.54 - 2024-09-05 - announce
changes | migration notes | docs

2.1 - next major series
What's New in 2.1? | docs

SPONSOR SQLALCHEMY!

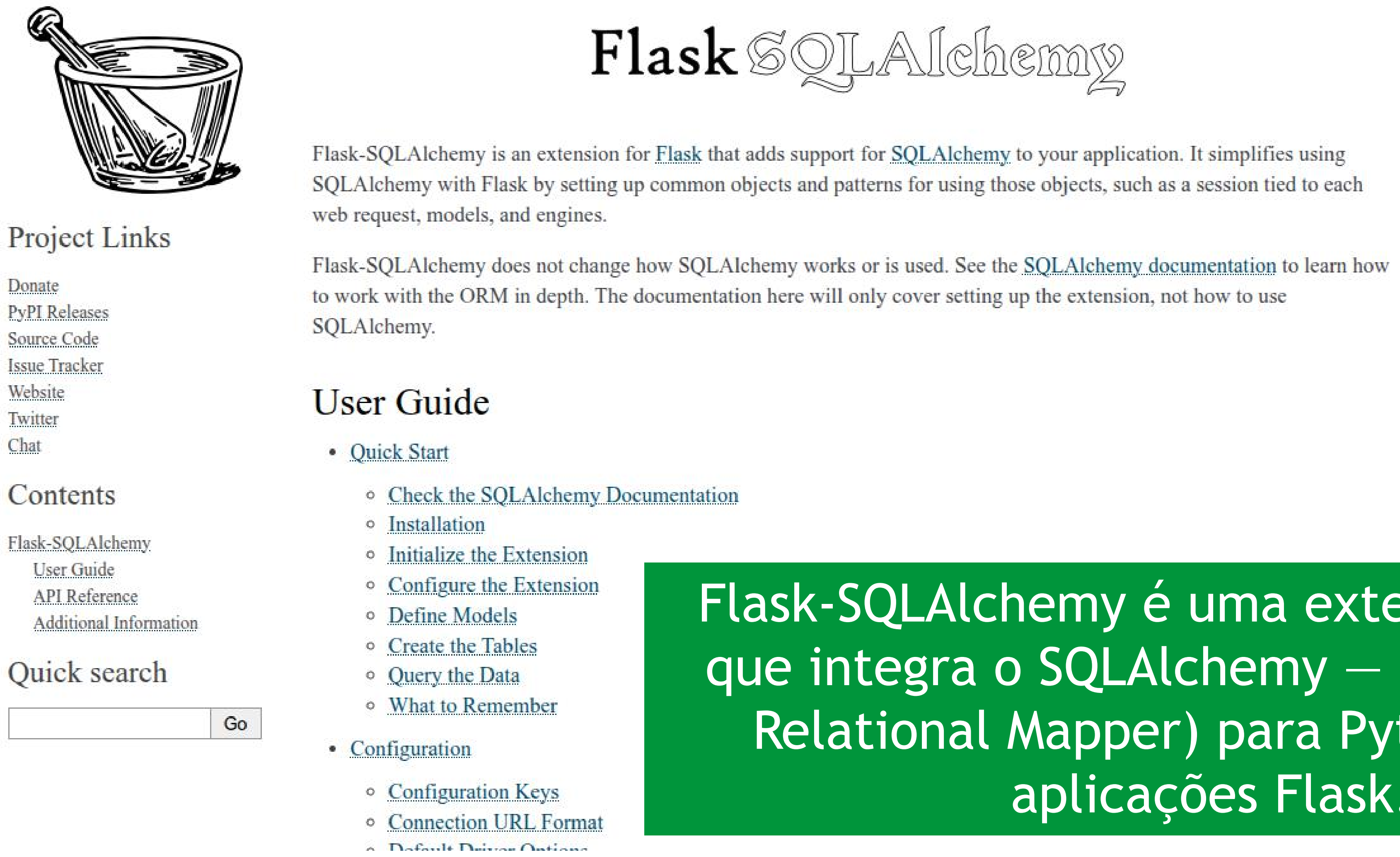
SQLAlchemy 2.0.38 Released
Thu, 06 Feb 2025

SQLAlchemy 2.0.37 Released
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



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

Project Links

Donate
PyPI Releases
Source Code
Issue Tracker
Website
Twitter
Chat

Contents

Flask-SQLAlchemy
User Guide
API Reference
Additional Information

Quick search

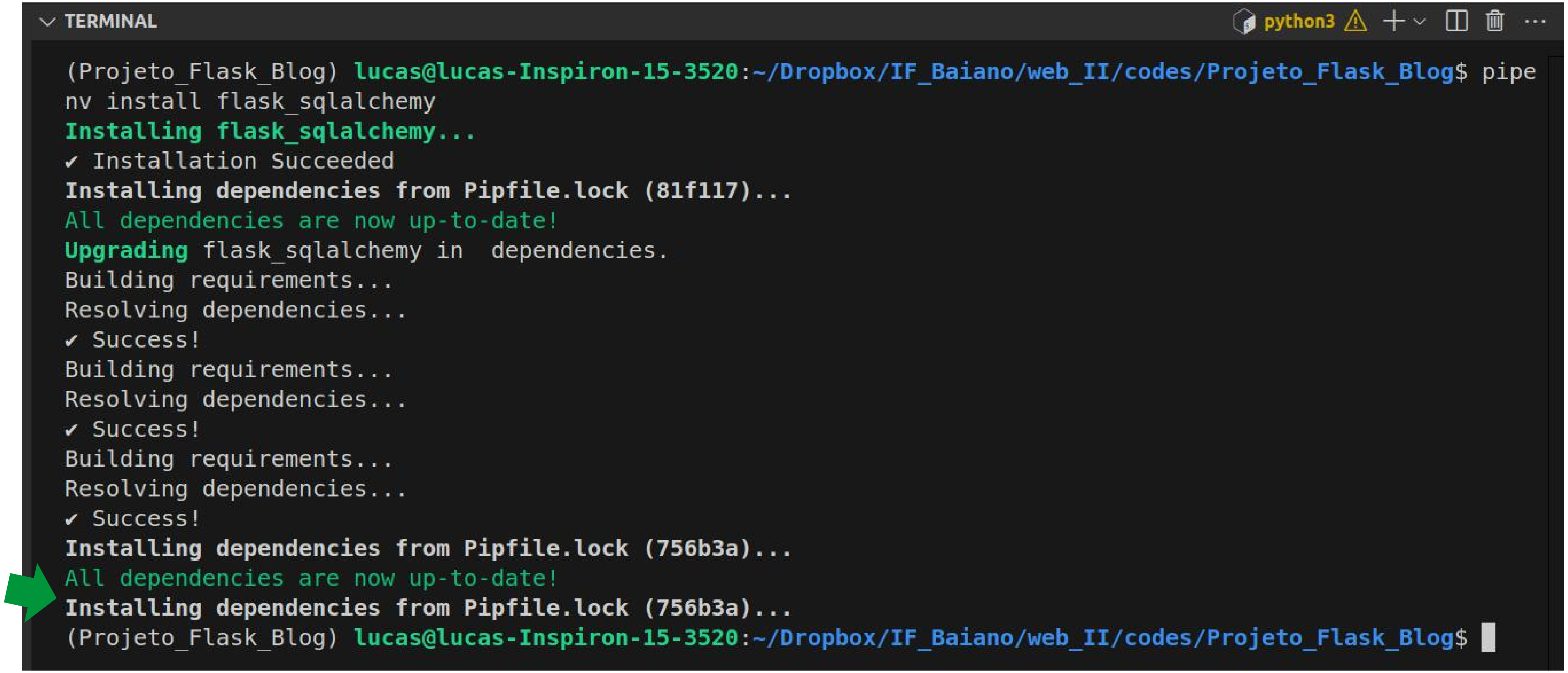
Go

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/

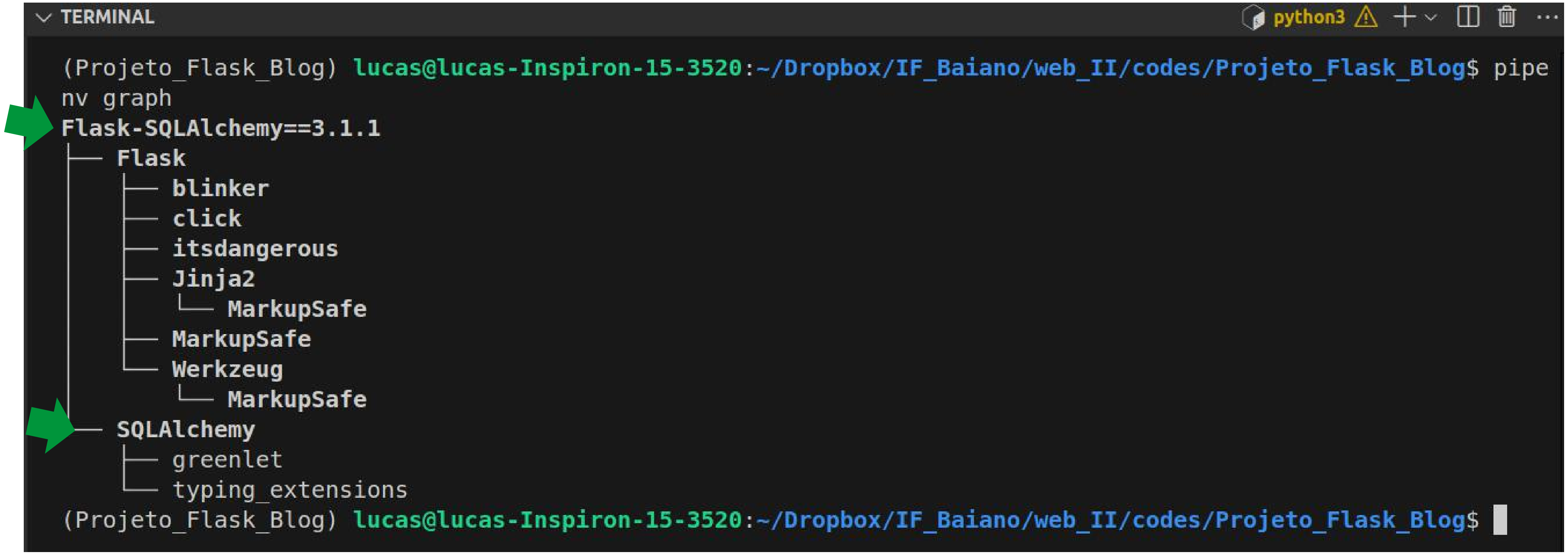
Instalando o Flask SQLAlchemy

- pipenv install flask_sqlalchemy



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



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

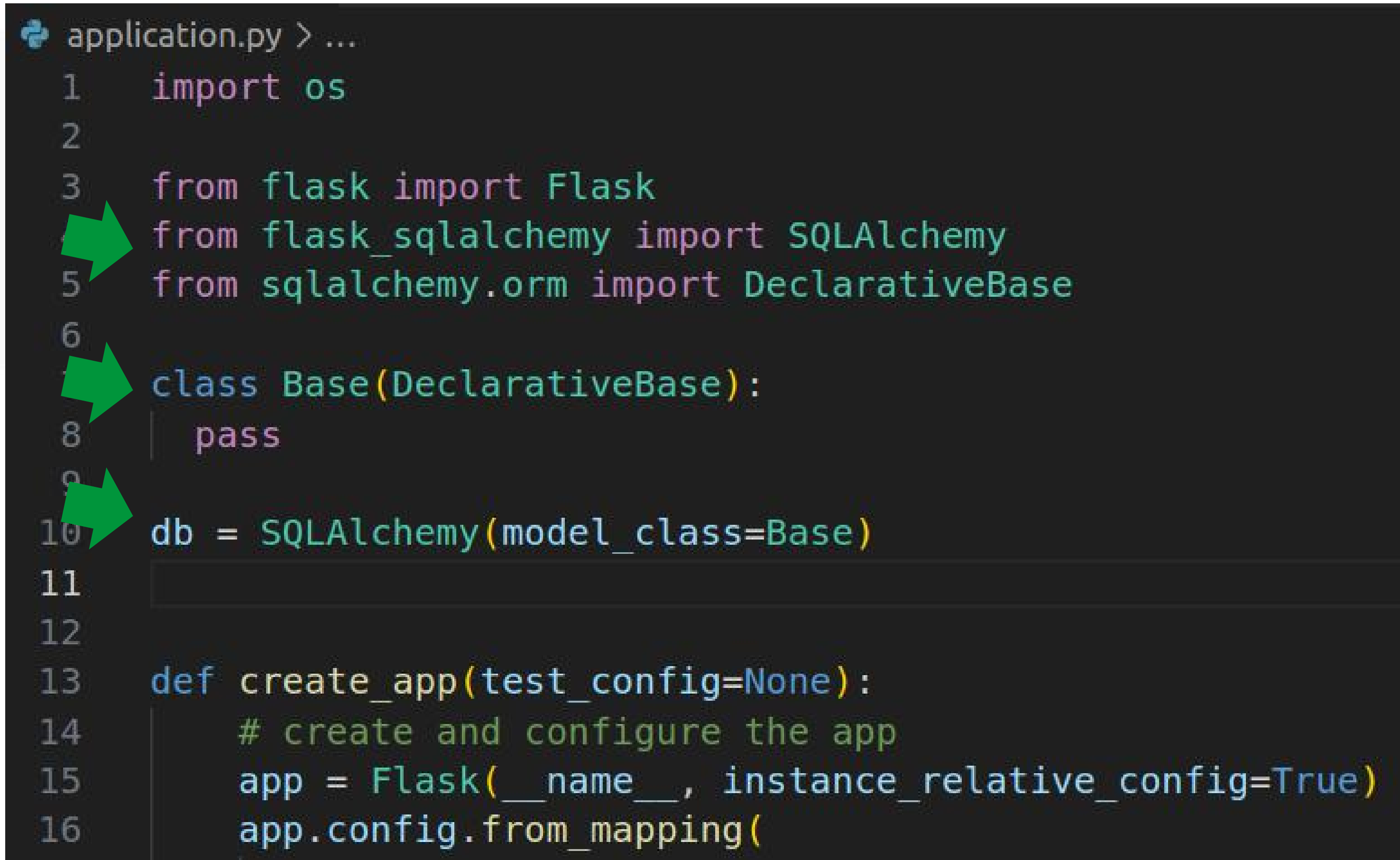
Inicializando a extensão SQLAlchemy

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)
```

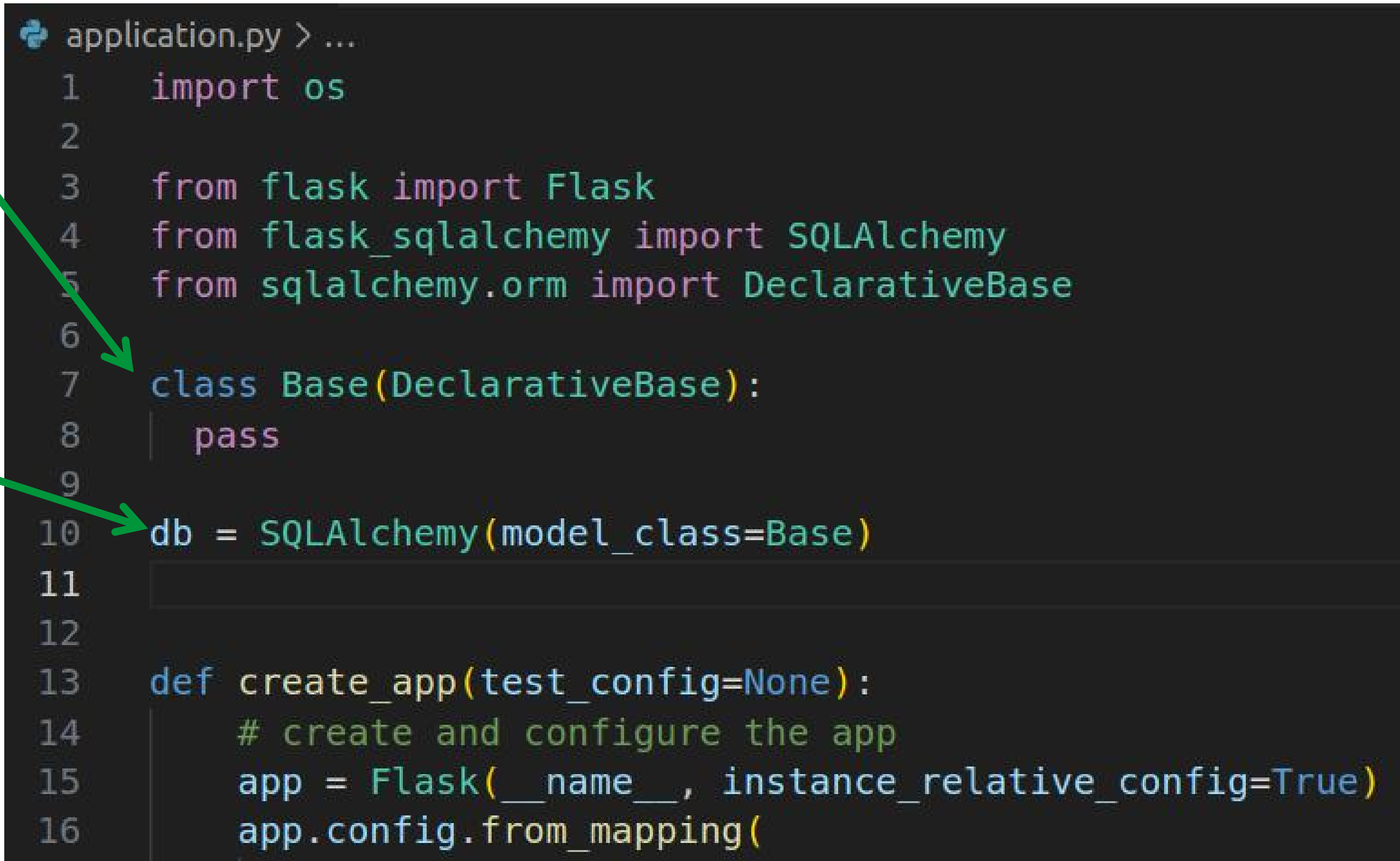

application.py > ...

```
import os

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

class Base(DeclarativeBase):
    pass

db = SQLAlchemy(model_class=Base)


def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    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

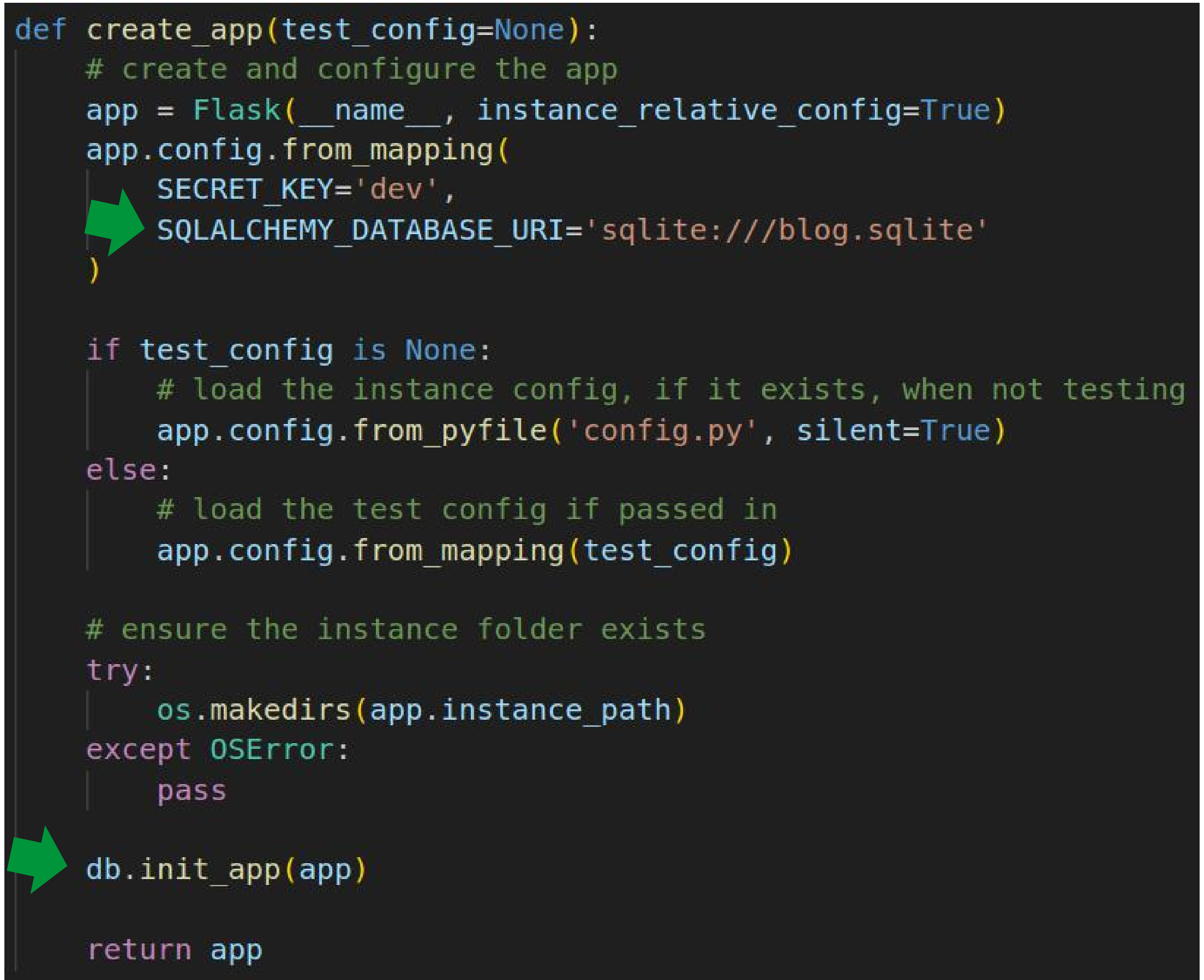
Cria a instância da extensão SQLAlchemy, informando que os modelos devem herdar de Base



application.py > ...

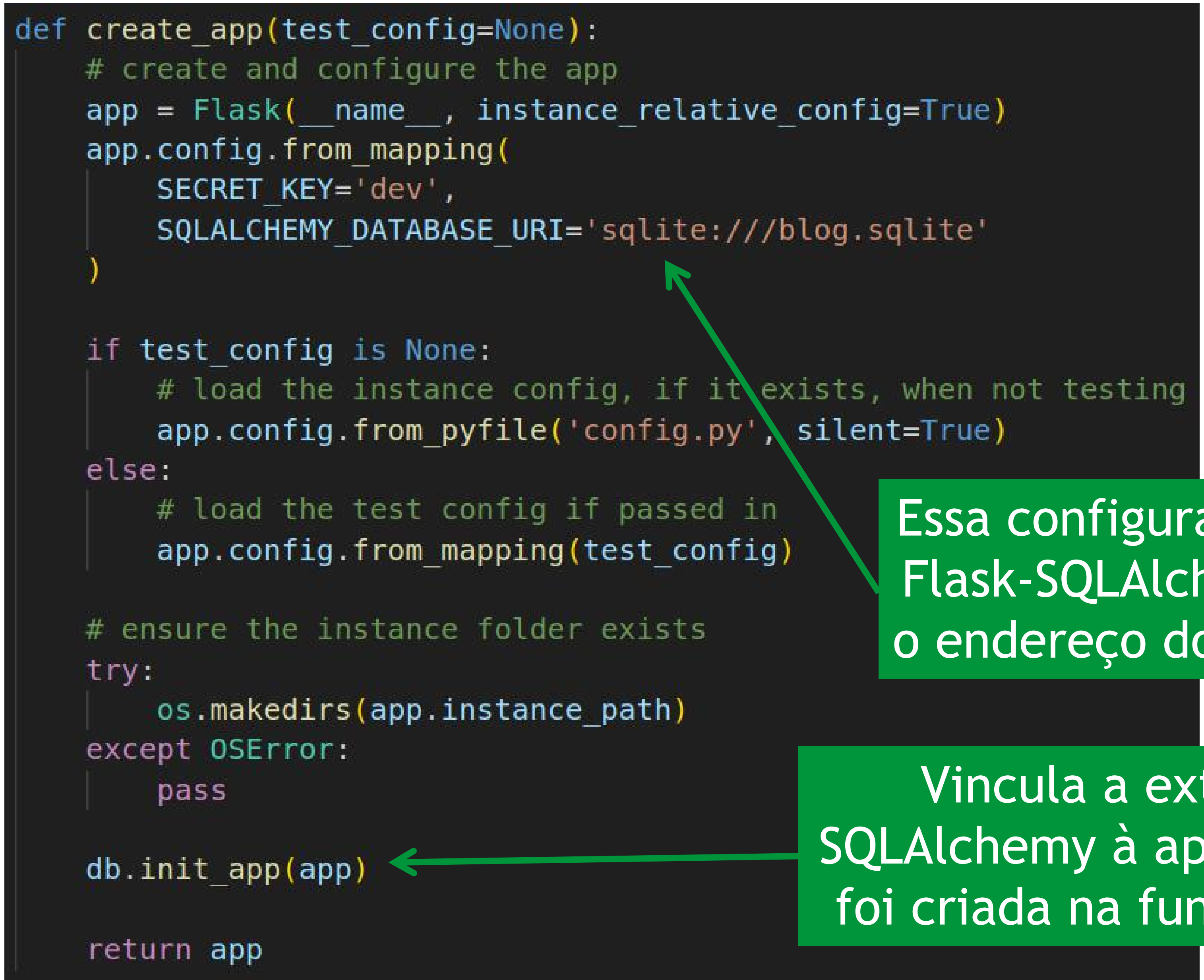
```python
import os

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

class Base(DeclarativeBase):
    pass

db = SQLAlchemy(model_class=Base)


def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
```

Inicializando a extensão do SQLAlchemy



```python
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

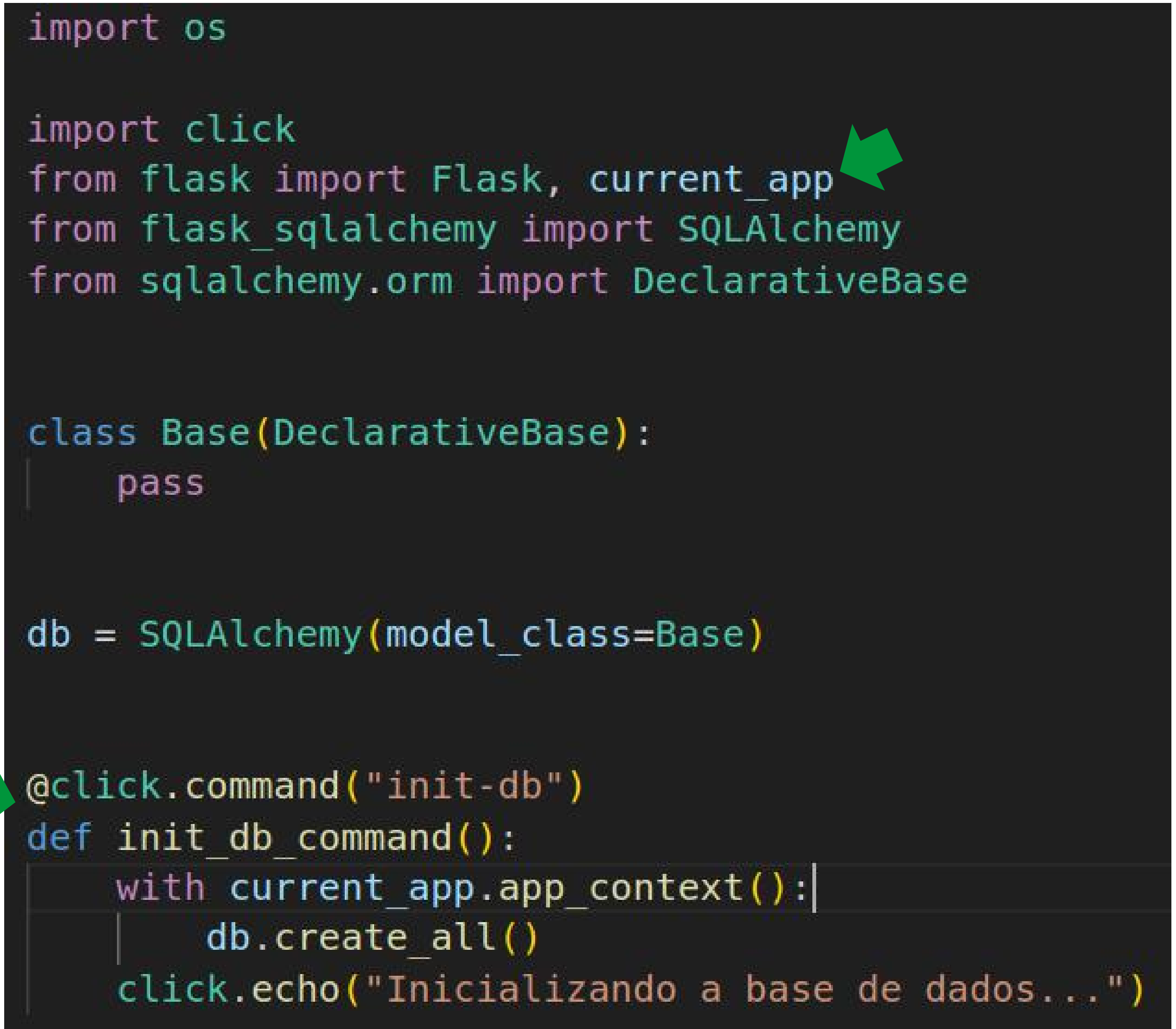


```python
def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
        SECRET_KEY='dev',
        SQLALCHEMY_DATABASE_URI='sqlite:///blog.sqlite'
    )

    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.

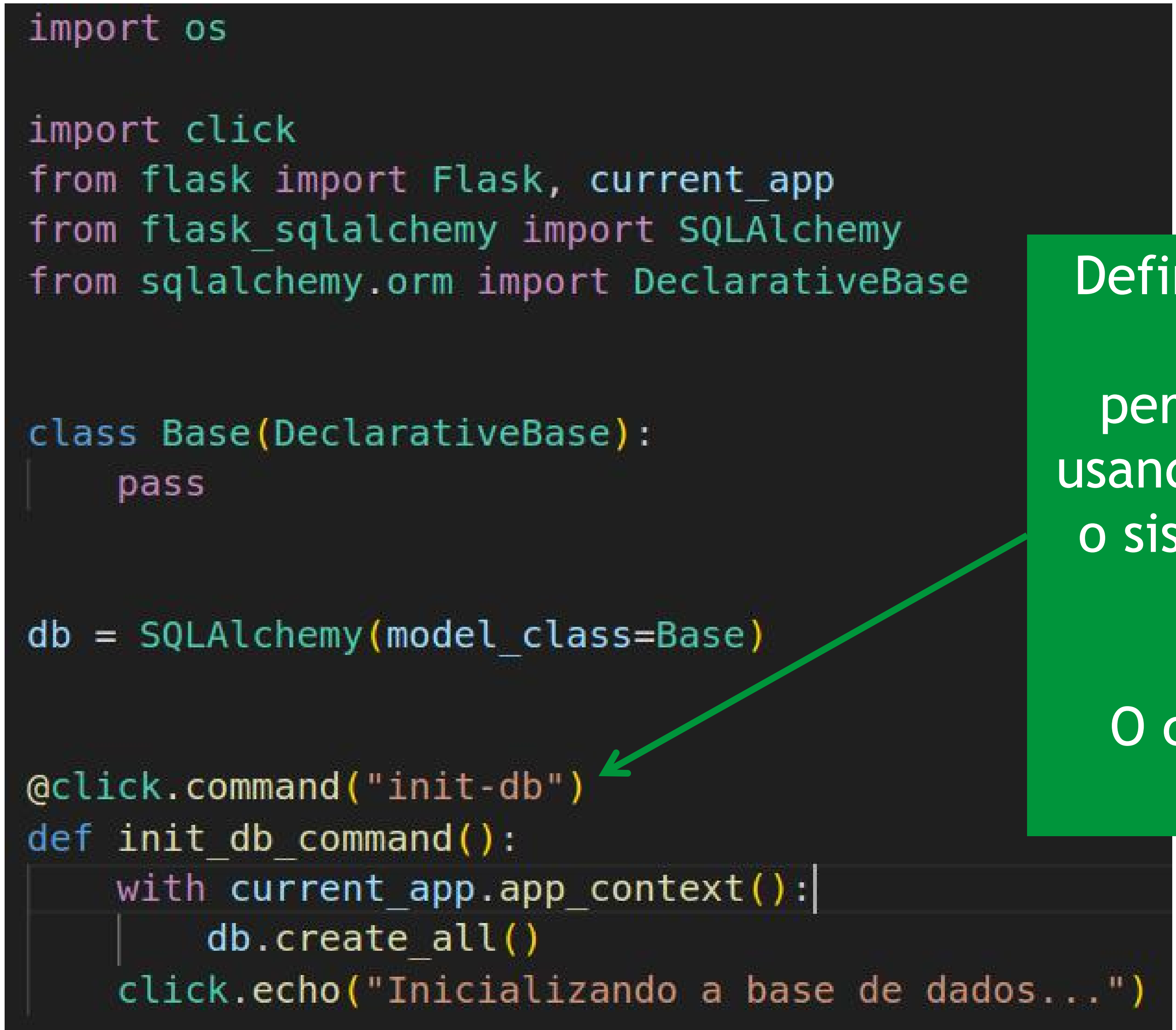
Vincula a extensão Flask-SQLAlchemy à aplicação Flask que foi criada na função create_app.

Inicializando o banco de dados via comandos CLI no Flask



```python
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

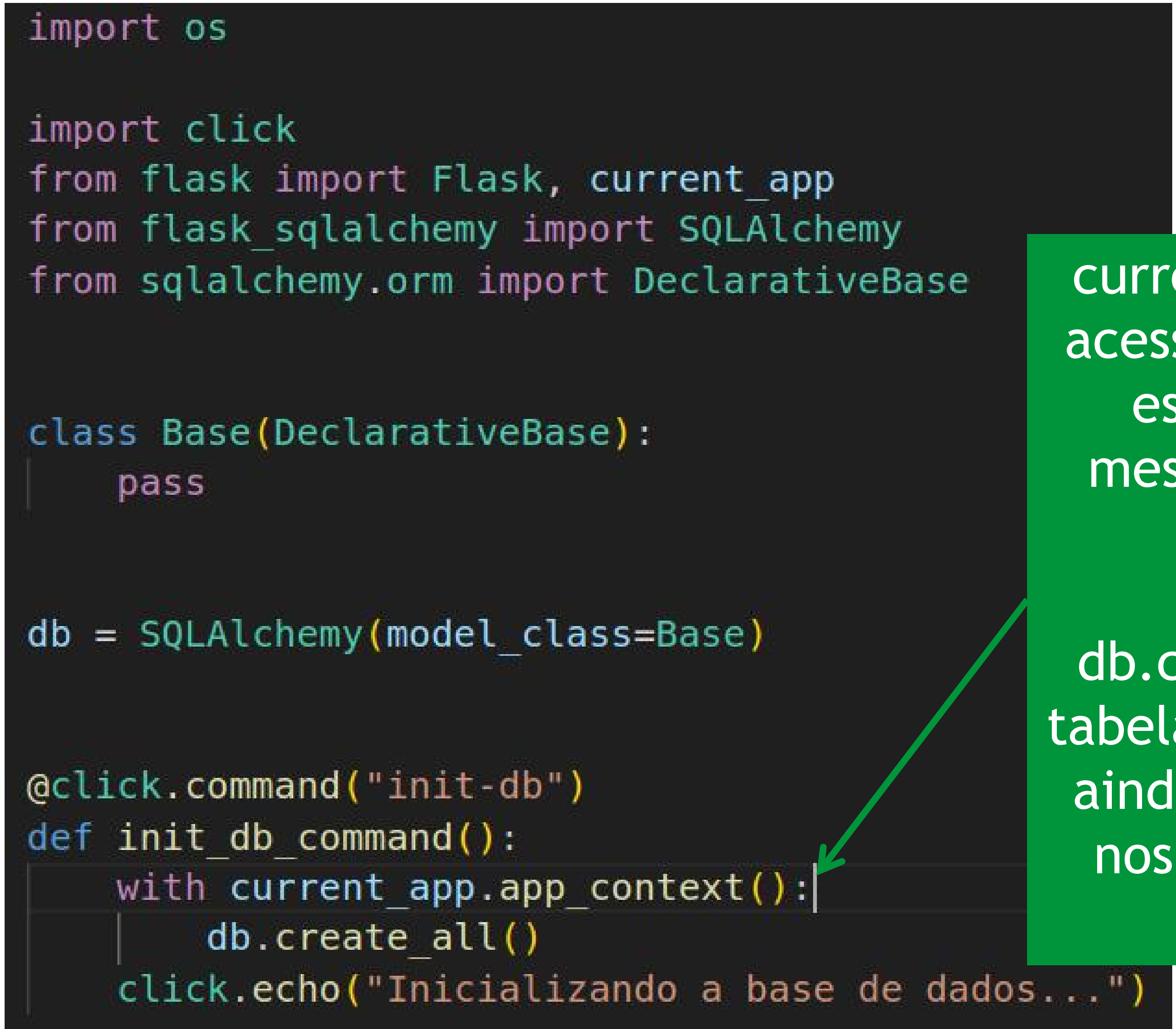


```python
import os

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

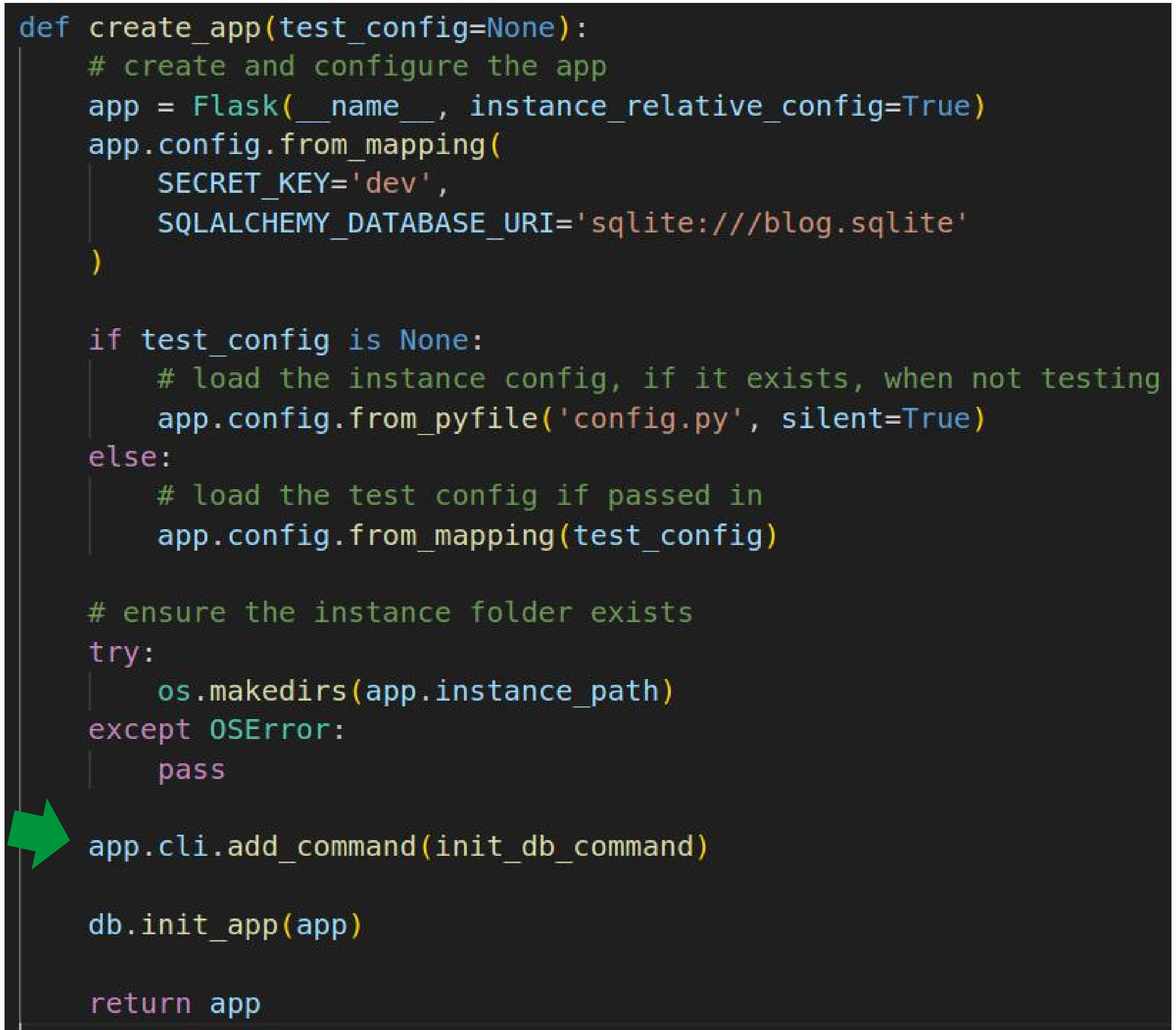


```python
import os

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.

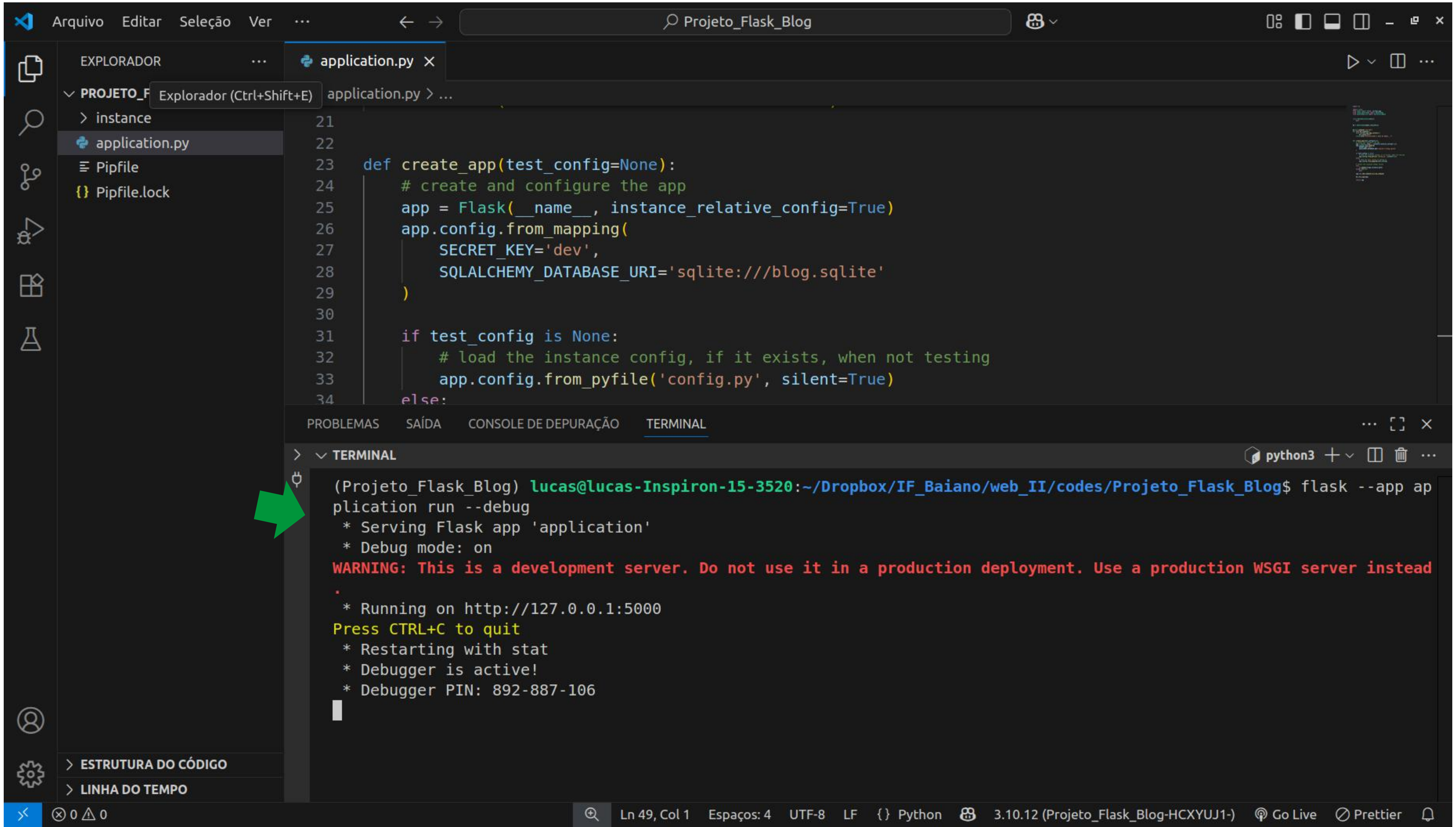
db.create_all() cria todas as tabelas do banco de dados que ainda não existem, baseadas nos seus modelos definidos com SQLAlchemy

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

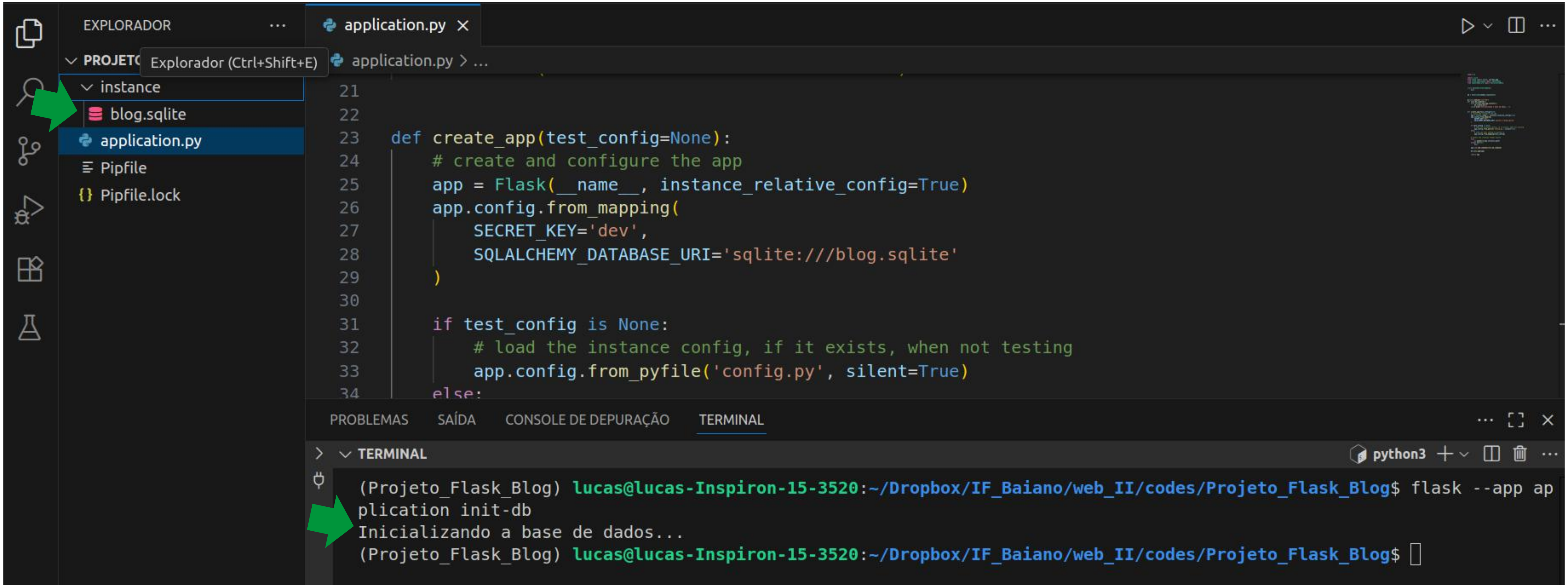


```python
def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
        SECRET_KEY='dev',
        SQLALCHEMY_DATABASE_URI='sqlite:///blog.sqlite'
    )

    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 ap
plication 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
```

Criando o banco de dados

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



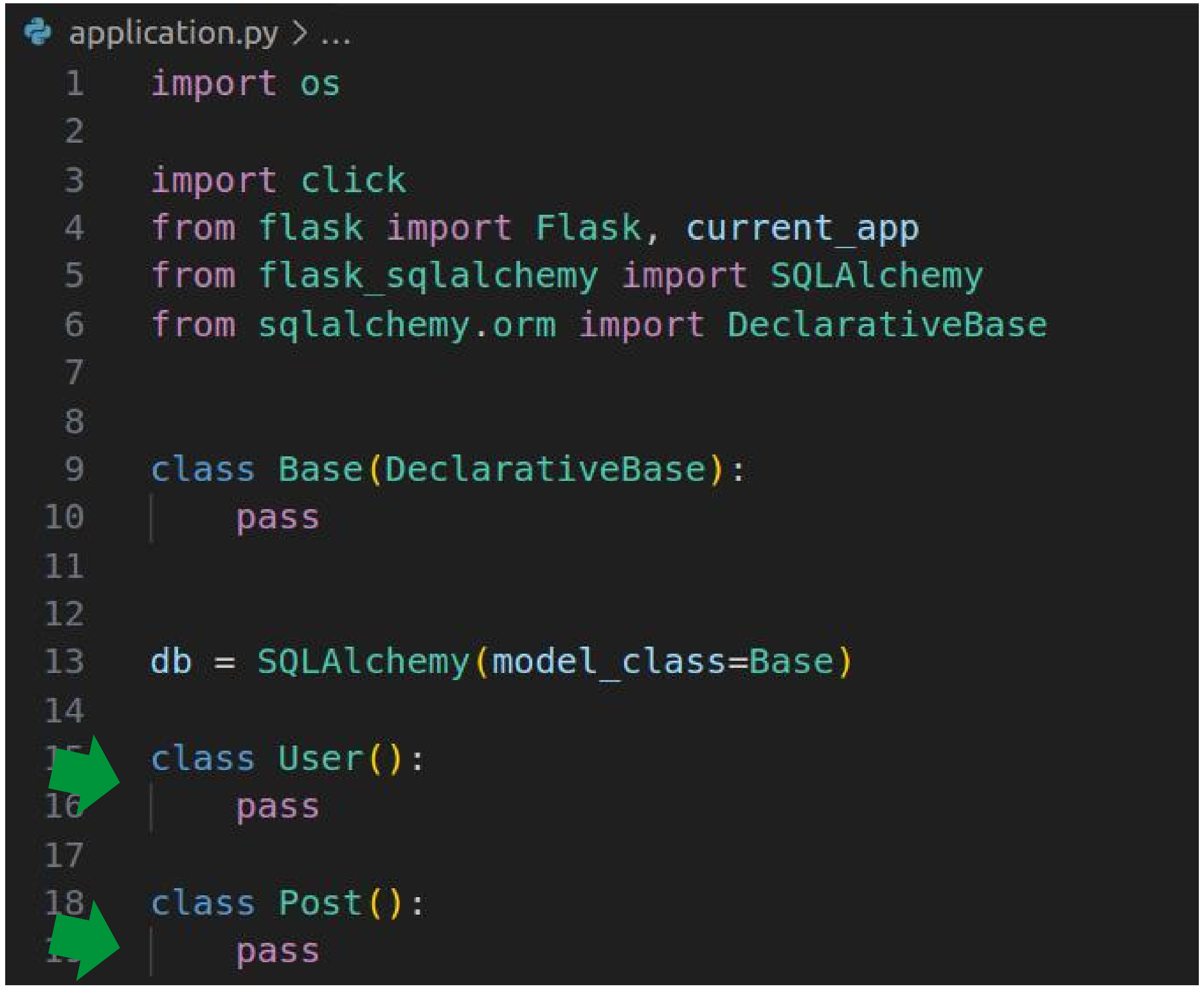
```python
def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
        SECRET_KEY='dev',
        SQLALCHEMY_DATABASE_URI='sqlite:///blog.sqlite'
    )

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

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

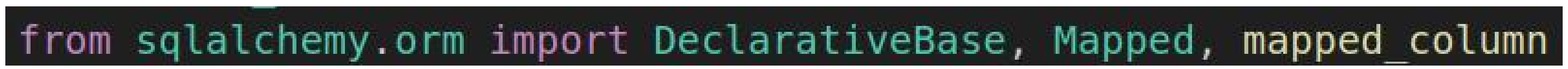


application.py > ...

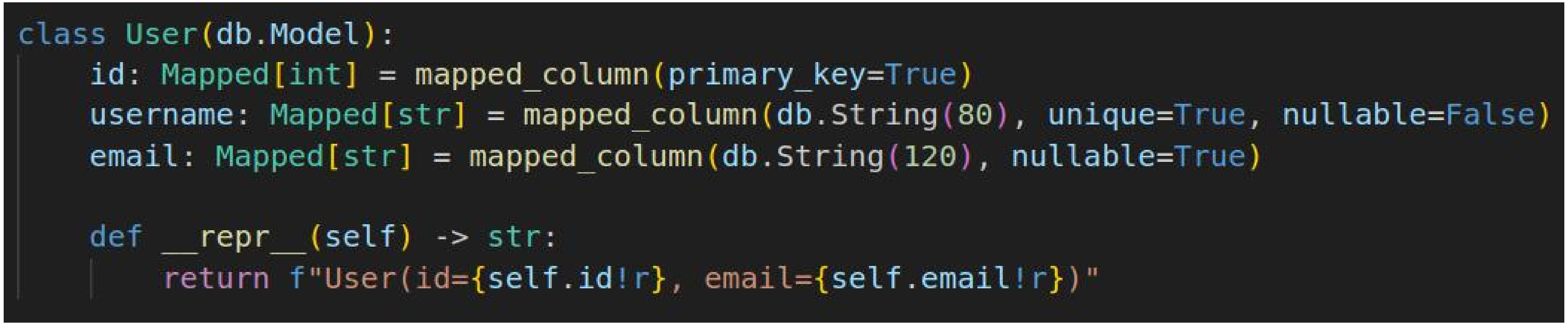
```python
import os

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)

class User():
    pass

class Post():
    pass
```

Definindo o modelo User

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



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

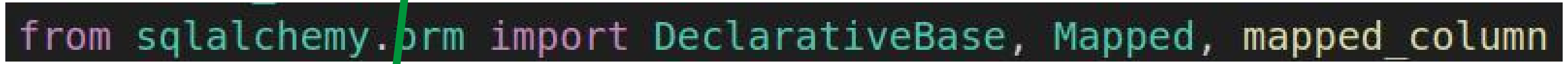


```python
class User(db.Model):
    id: Mapped[int] = mapped_column(primary_key=True)
    username: Mapped[str] = mapped_column(db.String(80), unique=True, nullable=False)
    email: Mapped[str] = mapped_column(db.String(120), nullable=True)

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

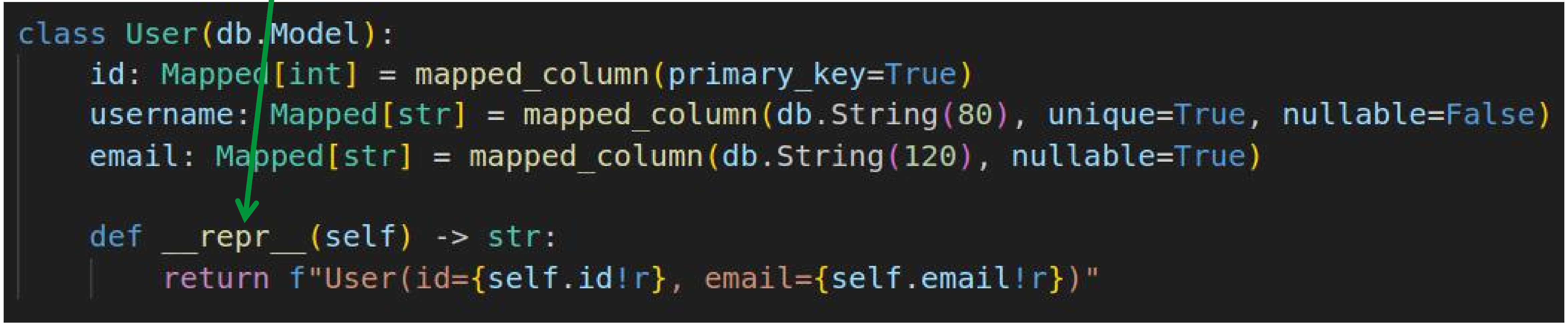
Definindo o modelo User

__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)
);
```



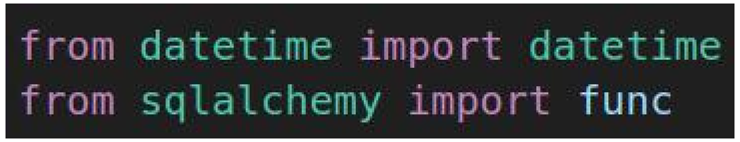
```python
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
```



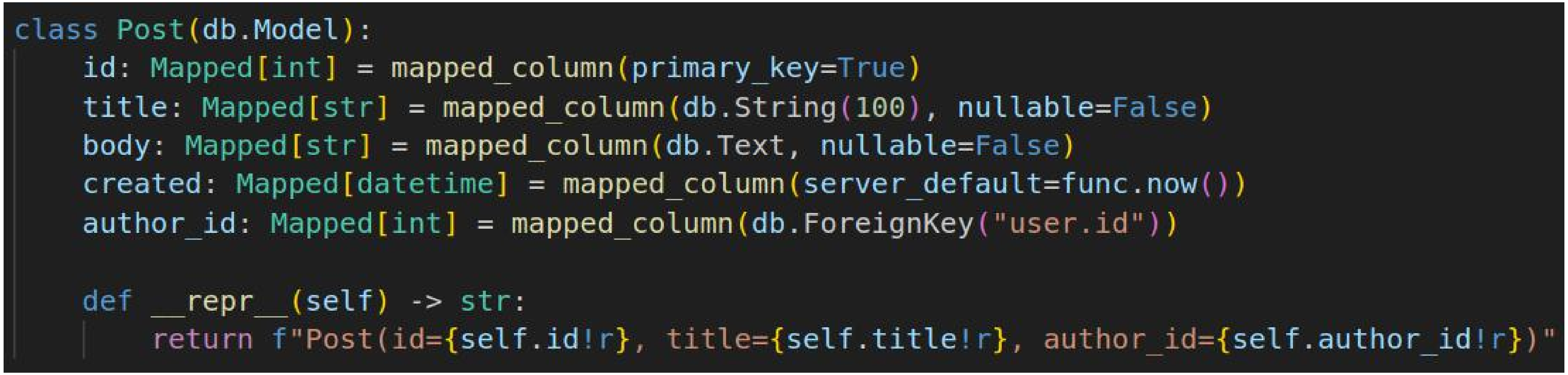
```python
class User(db.Model):
    id: Mapped[int] = mapped_column(primary_key=True)
    username: Mapped[str] = mapped_column(db.String(80), unique=True, nullable=False)
    email: Mapped[str] = mapped_column(db.String(120), nullable=True)

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

Definindo o modelo Post

```
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)
);
```



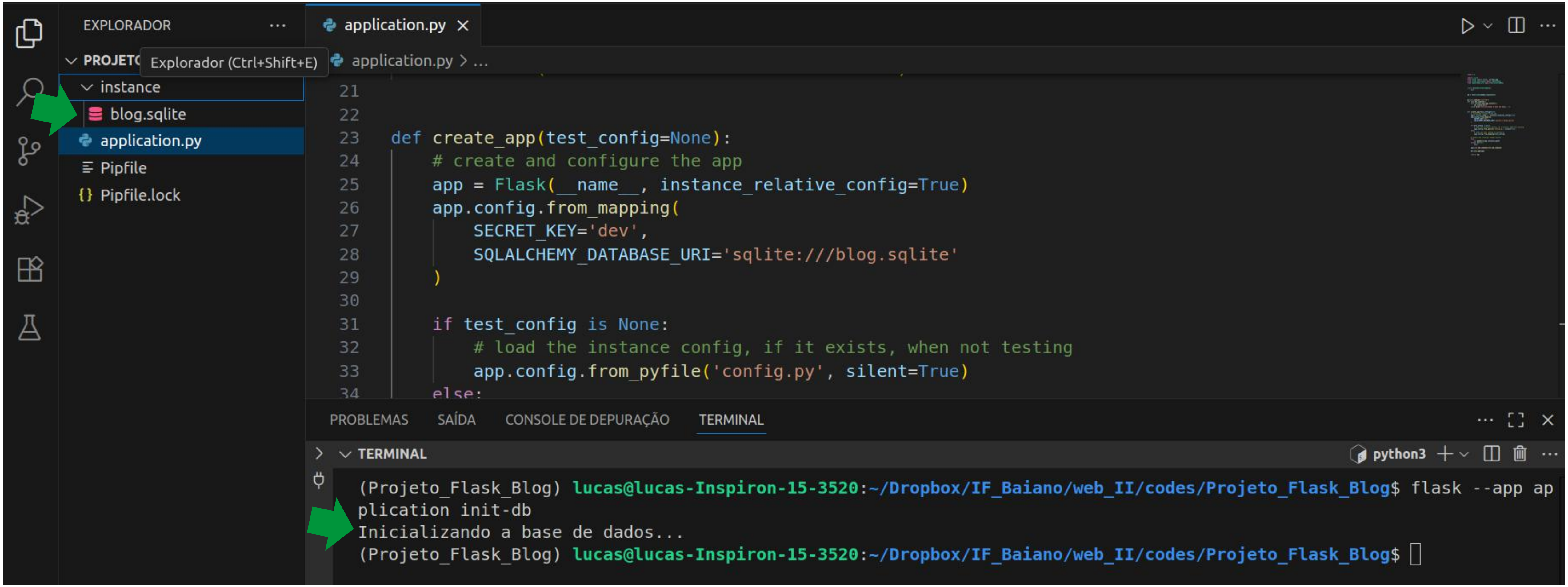
```python
from datetime import datetime
from sqlalchemy import func
```



```python
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

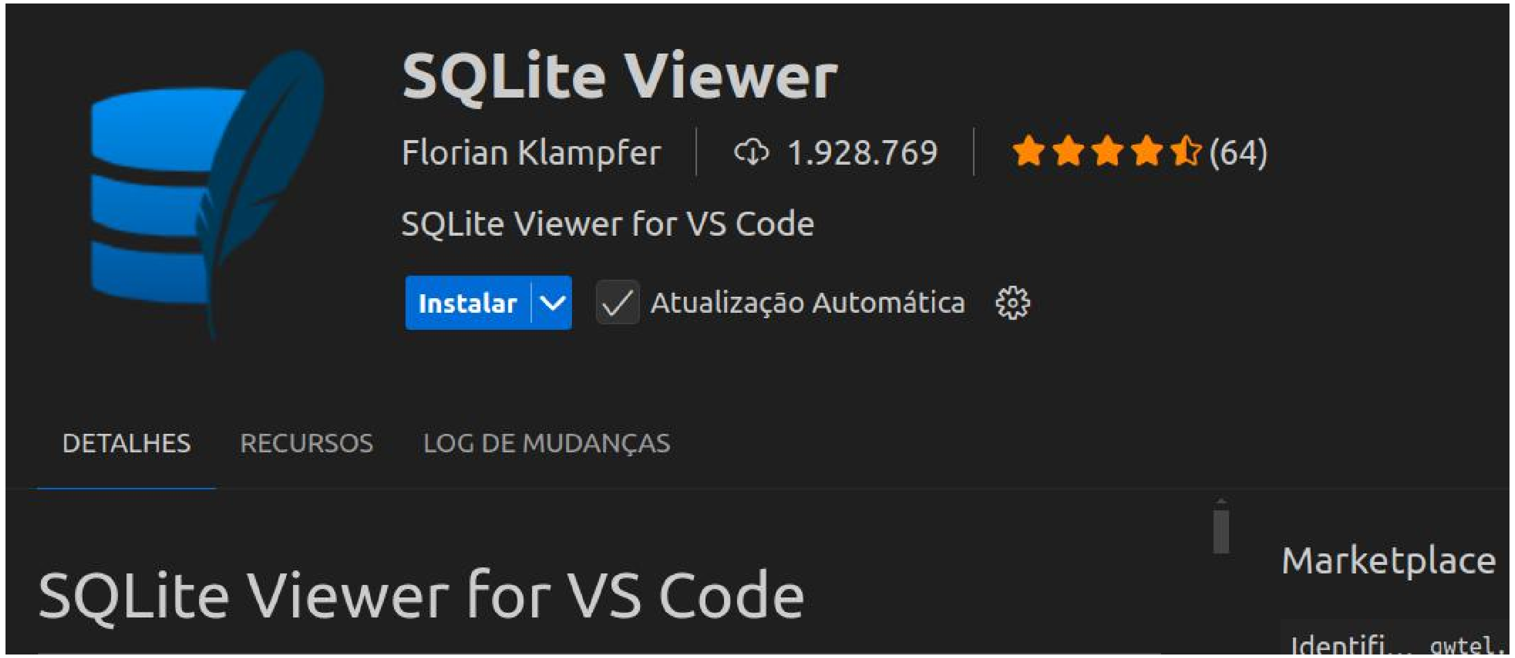
- Comando CLI: flask --app application init-db



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

Acessando as tabelas do banco



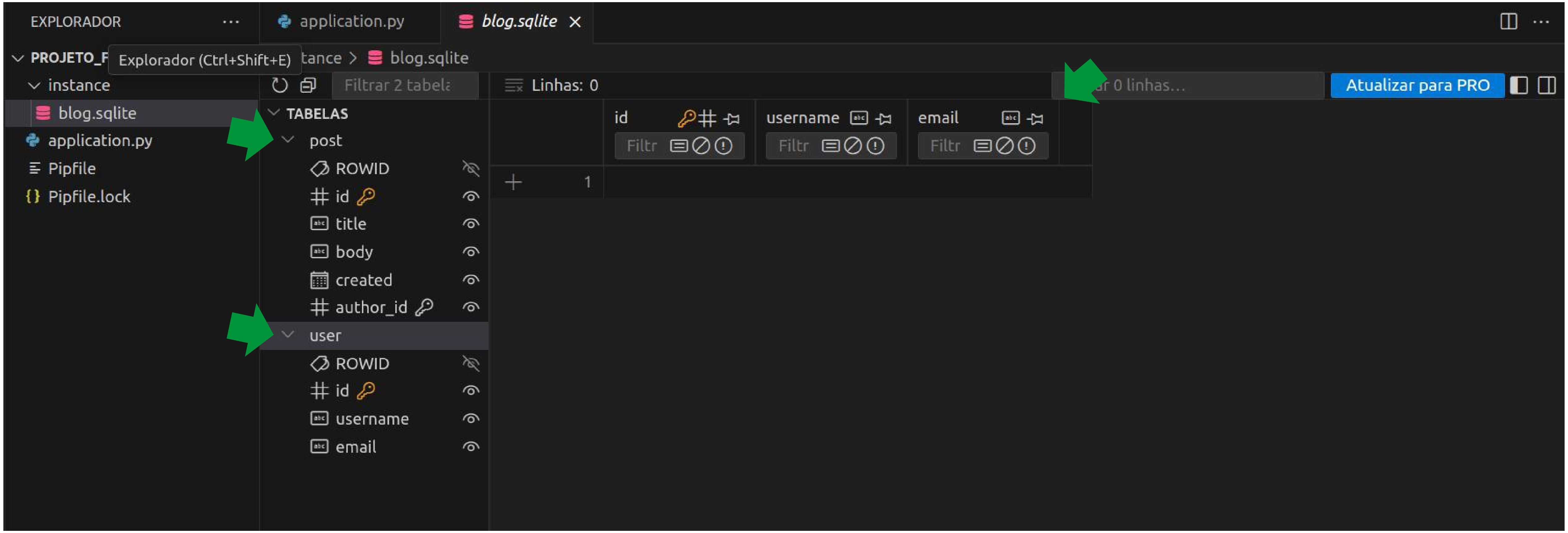
SQLite Viewer

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

SQLite Viewer for VS Code

SQLite Viewer for VS Code

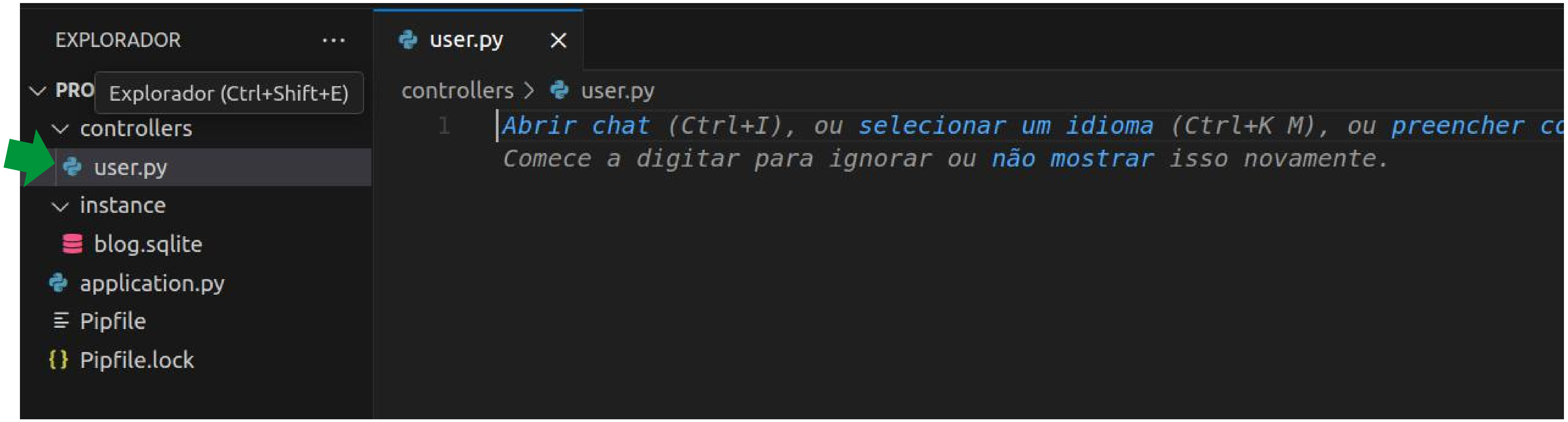
Acessando as tabelas do banco



Definindo os as rotas e controladores usando Blueprints

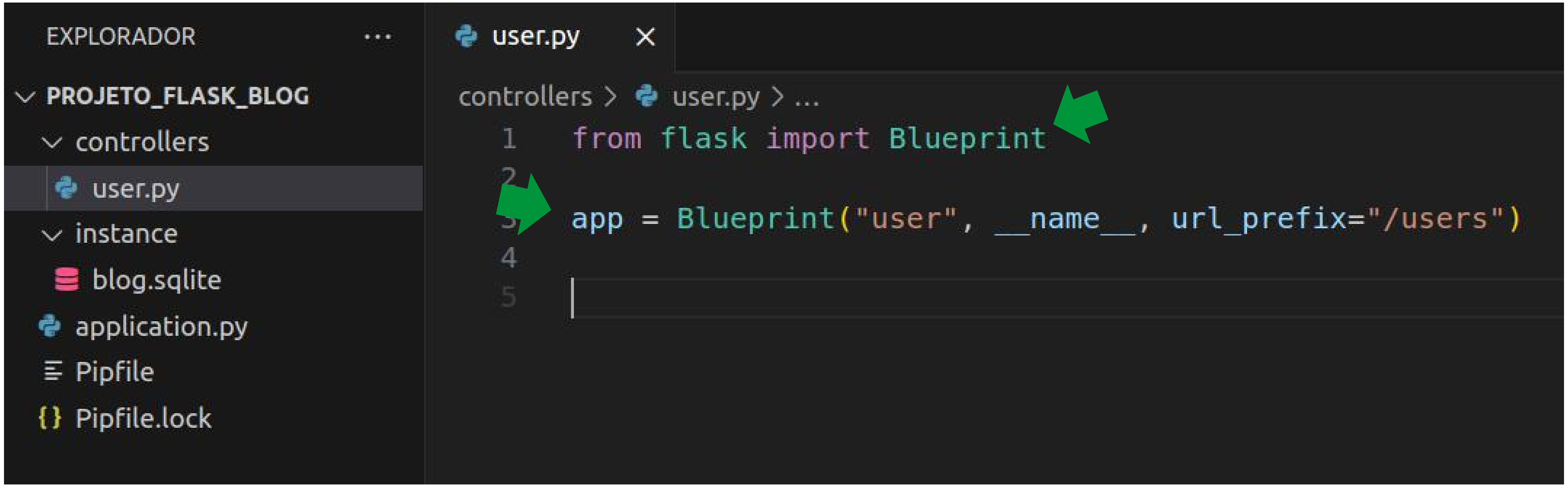
- No Flask, Blueprints são uma forma de organizar e modularizar a aplicação. Eles permitem dividir a aplicação em componentes reutilizáveis e independentes, cada um com suas próprias rotas e controladores.
- Os Blueprints no Flask permitem que você organize rotas separadas por funcionalidade, e isso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

Criando o controlador de usuários



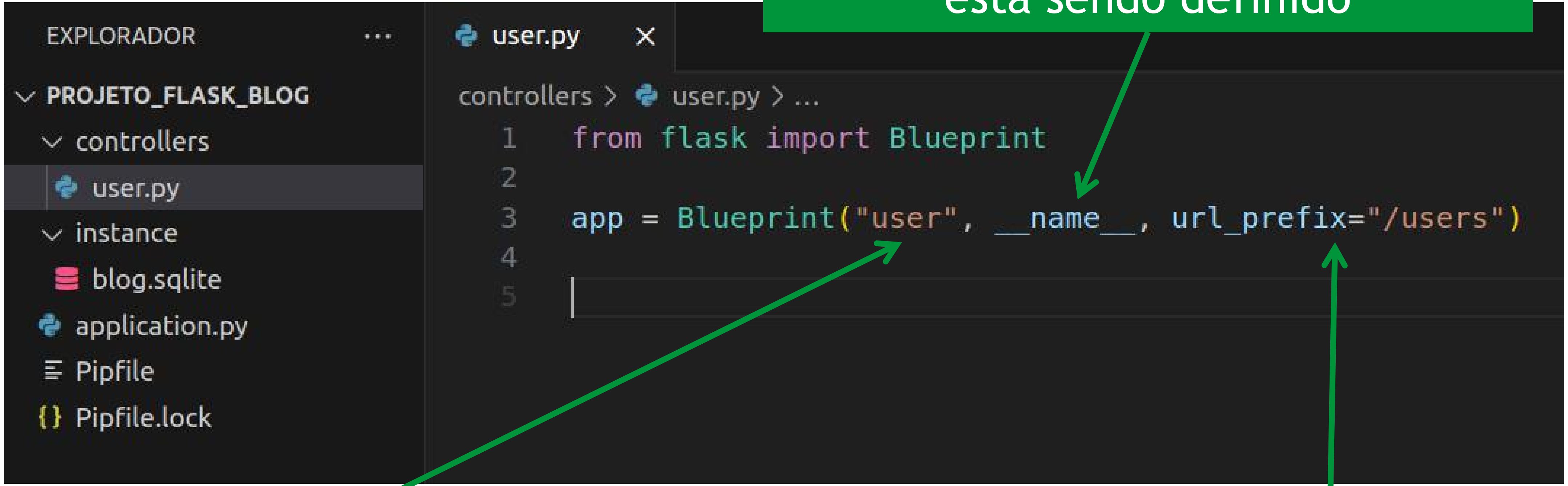
```
Abrir chat (Ctrl+I), ou selecionar um idioma (Ctrl+K M), ou preencher co
Comece a digitar para ignorar ou não mostrar isso novamente.
```

Criando o controlador de usuários



```python
from flask import Blueprint

app = Blueprint("user", __name__, url_prefix="/users")

```

Criando o controlador de usuários



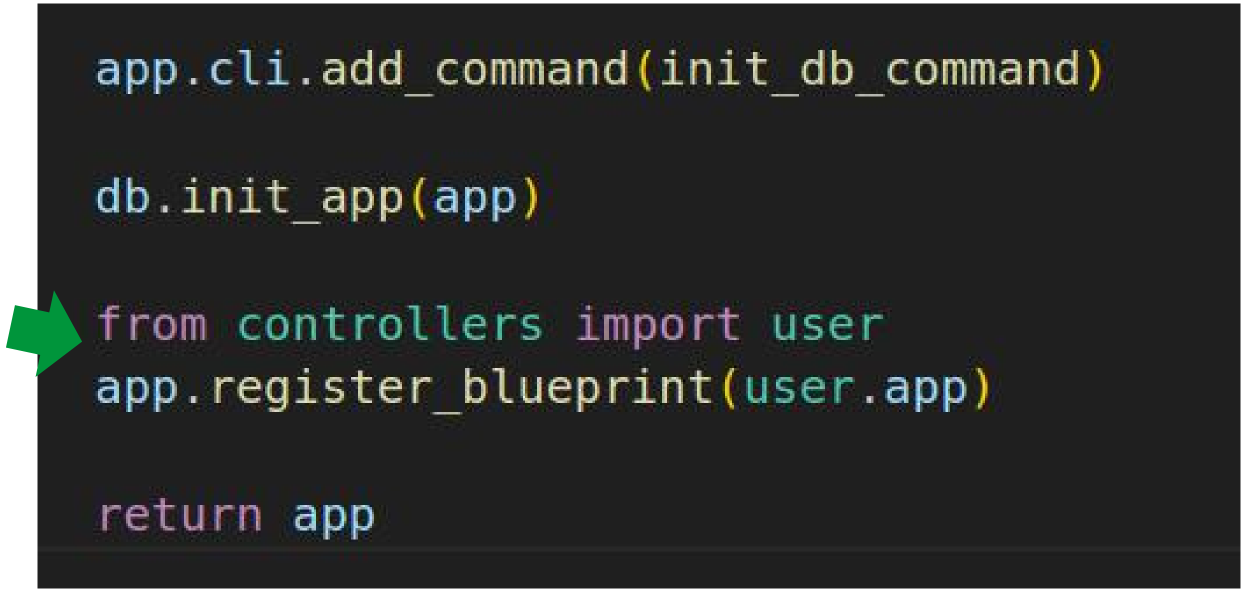
```python
from flask import Blueprint

app = Blueprint("user", __name__, url_prefix="/users")

```

Nome do módulo onde o blueprint está sendo definido

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

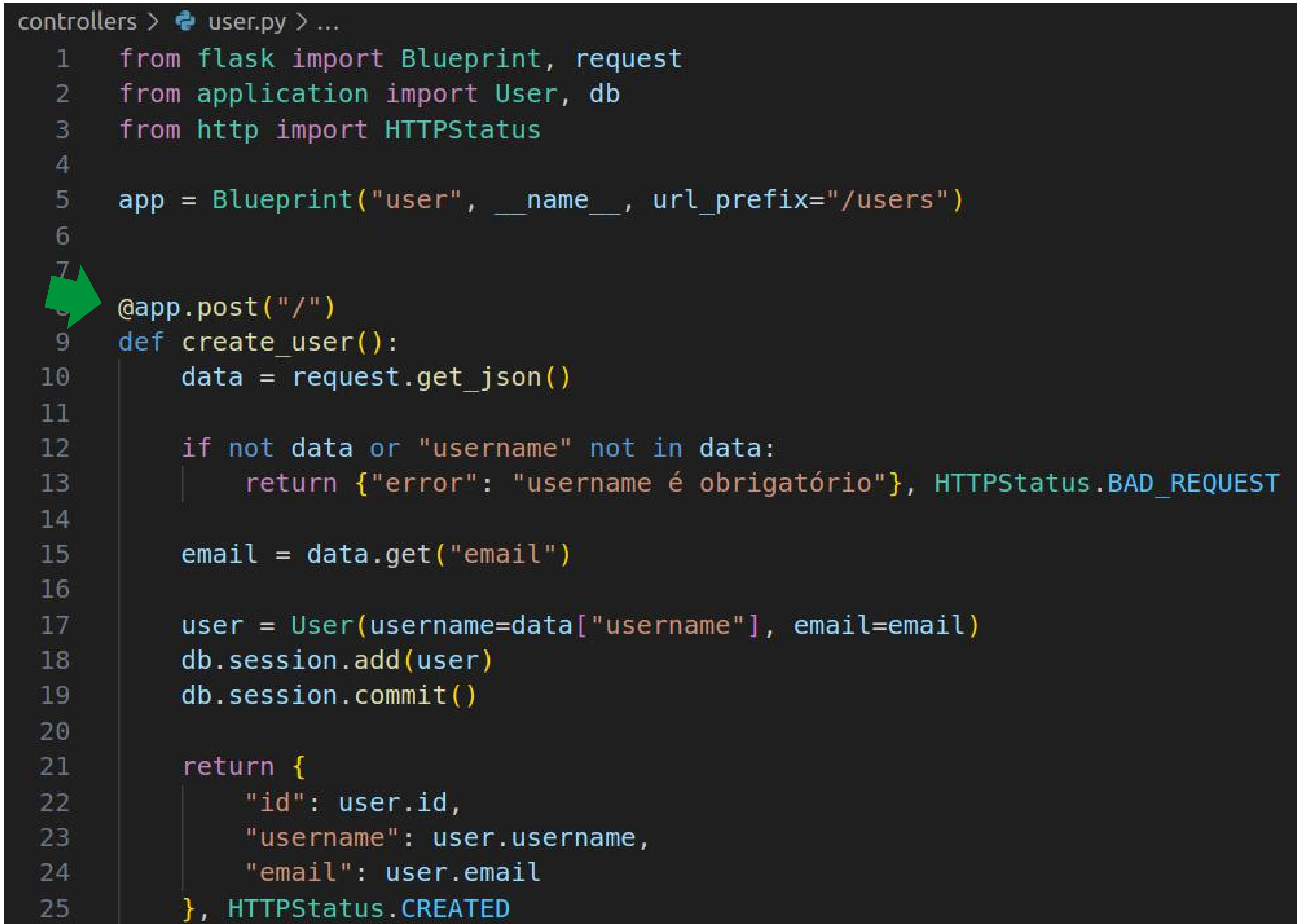
Define o prefixo de rota para todas as rotas registradas nesse blueprint

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
```

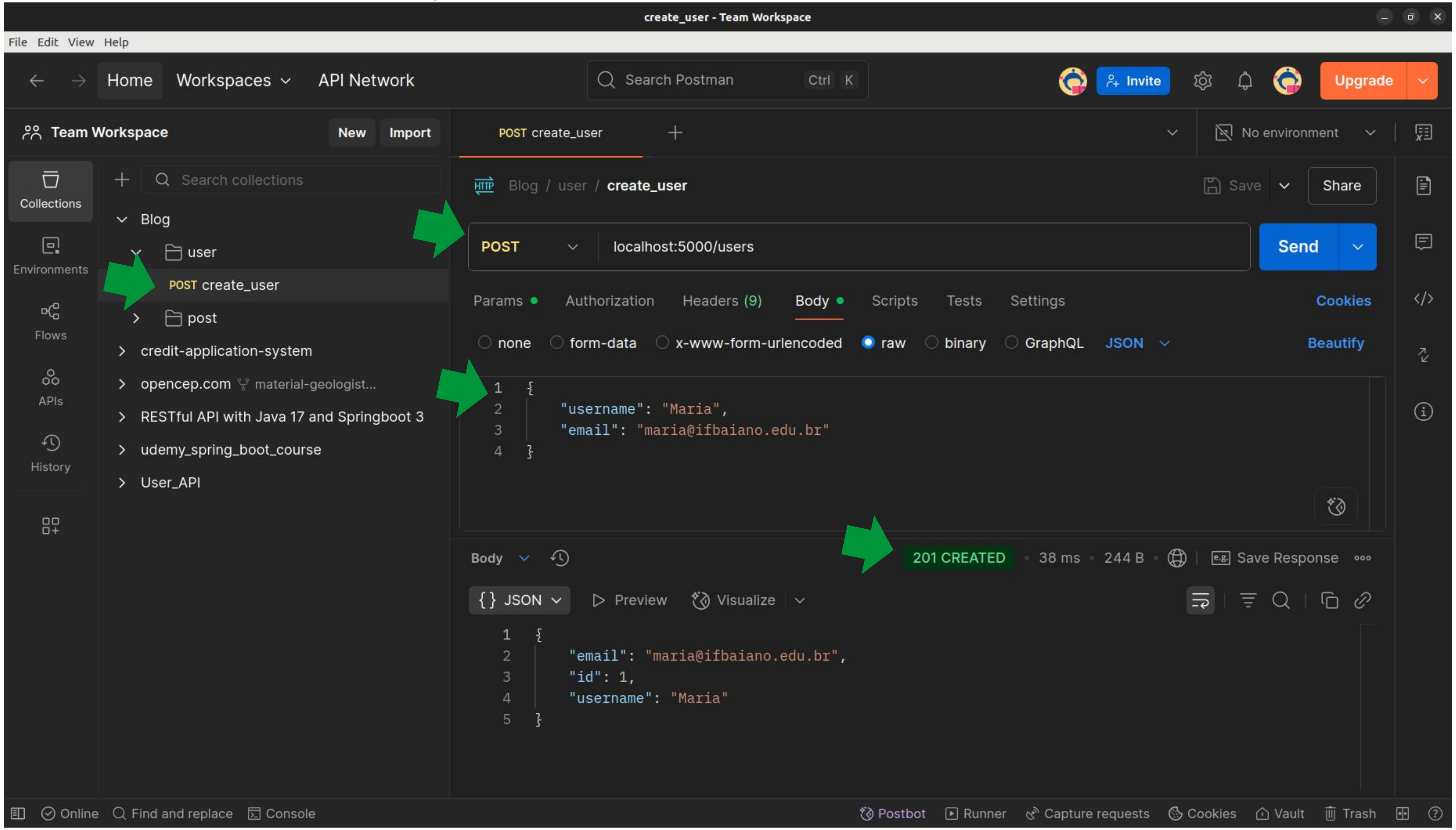
Definindo a operação de CREATE



controllers > user.py > ...

```python
from flask import Blueprint, request
from application import User, db
from http import HTTPStatus

app = Blueprint("user", __name__, url_prefix="/users")


@app.post("/")
def create_user():
    data = request.get_json()

    if not data or "username" not in data:
        return {"error": "username é obrigatório"}, HTTPStatus.BAD_REQUEST

    email = data.get("email")

    user = User(username=data["username"], email=email)
    db.session.add(user)
    db.session.commit()

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

Testando a operação de CREATE no Postman



POST localhost:5000/users

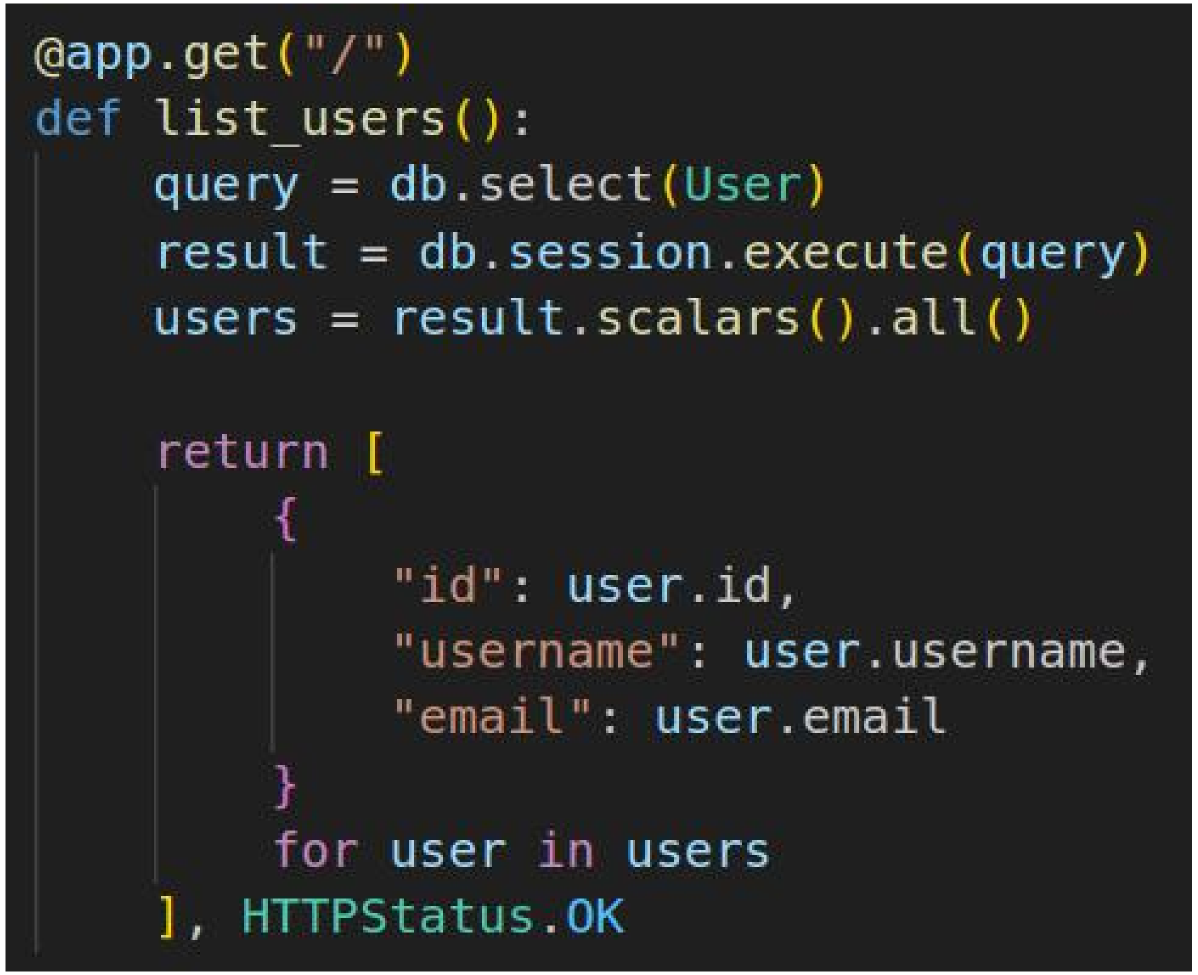
Body: [✓ raw] JSON

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

201 CREATED · 38 ms · 244 B

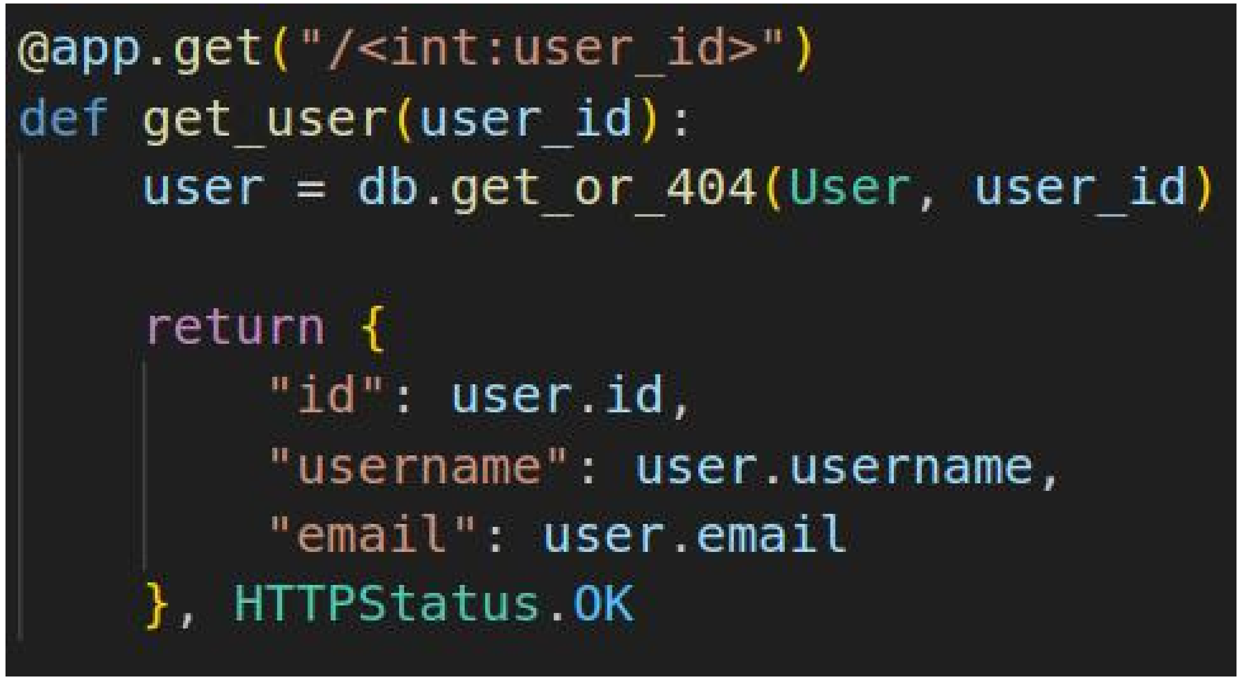
```
{
    "email": "maria@ifbaiano.edu.br",
    "id": 1,
    "username": "Maria"
}
```

Definindo as operações de READ



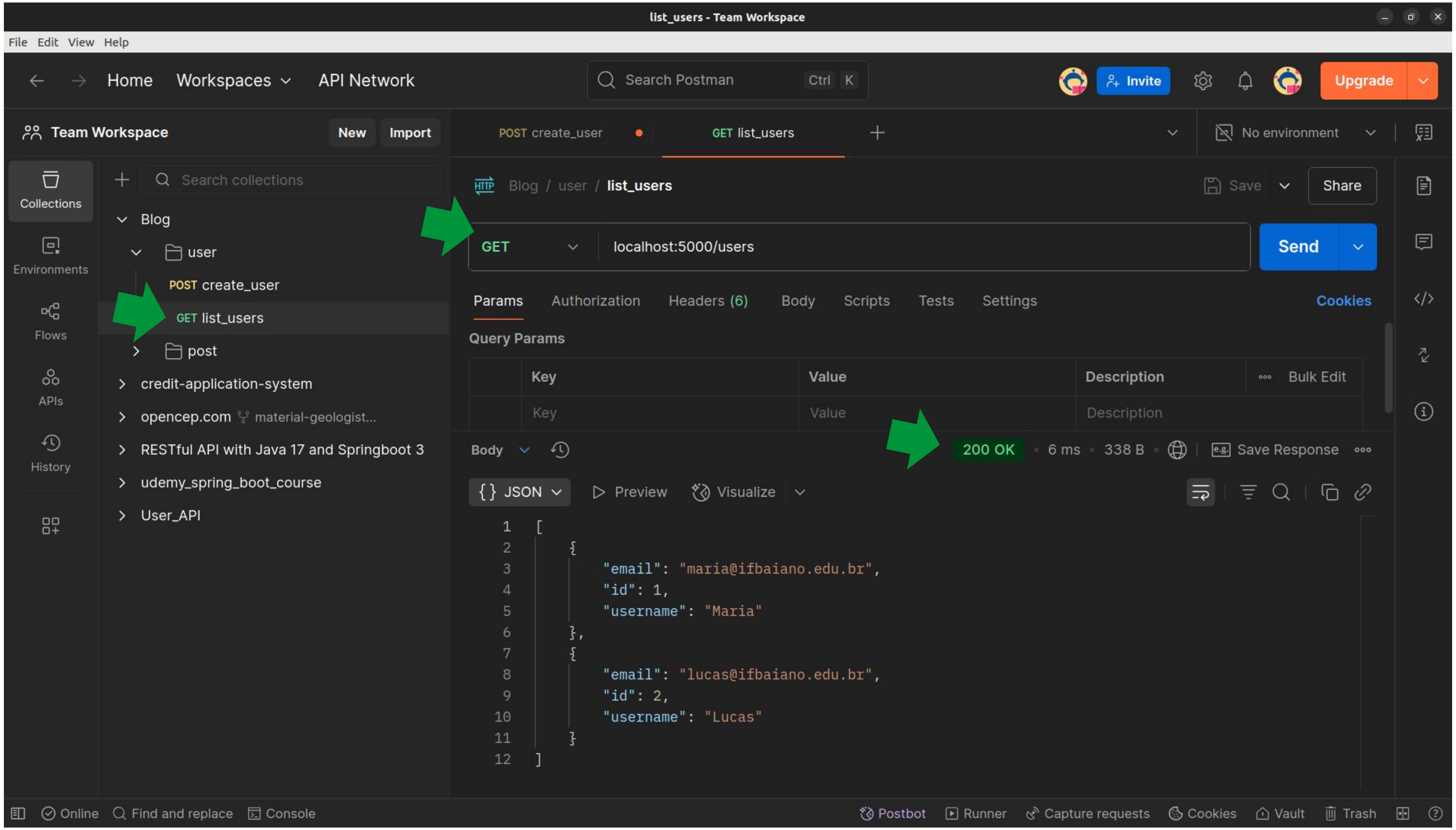
```
@app.get("/")
def list_users():
    query = db.select(User)
    result = db.session.execute(query)
    users = result.scalars().all()

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

Definindo as operações de READ



```python
@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

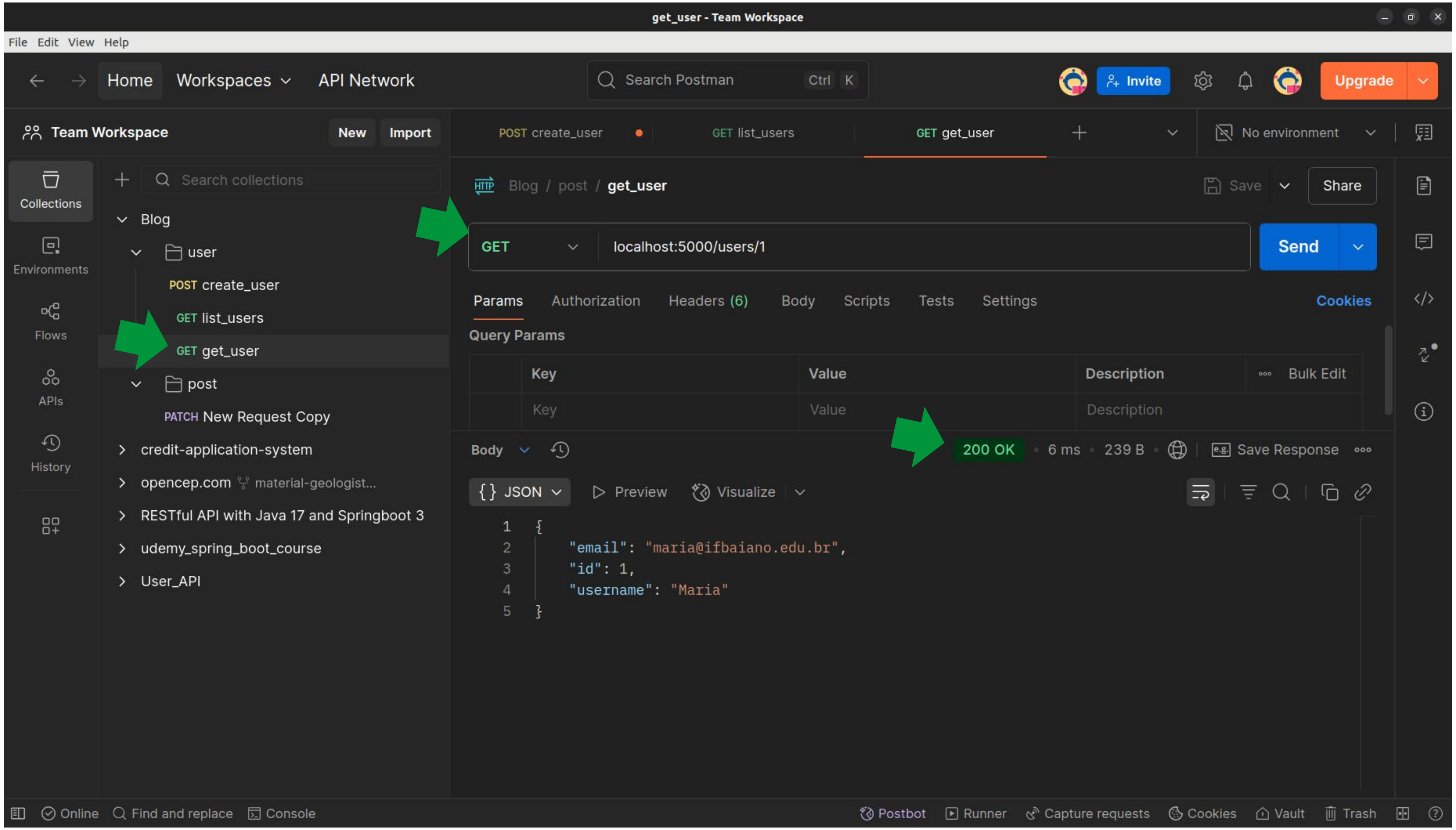

list_users - Team Workspace
File Edit View Help
Home Workspaces API Network
Search Postman
Invite Upgrade
Team Workspace New Import
POST create_user
GET list_users
No environment
Collections
Environments
Flows
APIs
History
Search collections
Blog
user
POST create_user
GET list_users
post
credit-application-system
opencep.com material-geologist...
RESTful API with Java 17 and Springboot 3
udemy_spring_boot_course
User_API
Blog / user / list_users
Save Share
Send
Params Authorization Headers (6) Body Scripts Tests Settings Cookies
Query Params
Key Value Description Bulk Edit
Body
200 OK 6 ms 338 B
Save Response
JSON Preview Visualize
Online Find and replace Console
Postbot Runner Capture requests Cookies Vault Trash

GET localhost:5000/users

```
[
    {
        "email": "maria@ifbaiano.edu.br",
        "id": 1,
        "username": "Maria"
    },
    {
        "email": "lucas@ifbaiano.edu.br",
        "id": 2,
        "username": "Lucas"
    }
]
```

INSTITUTO FEDERAL Baiano

Testando as operações de READ no Postman

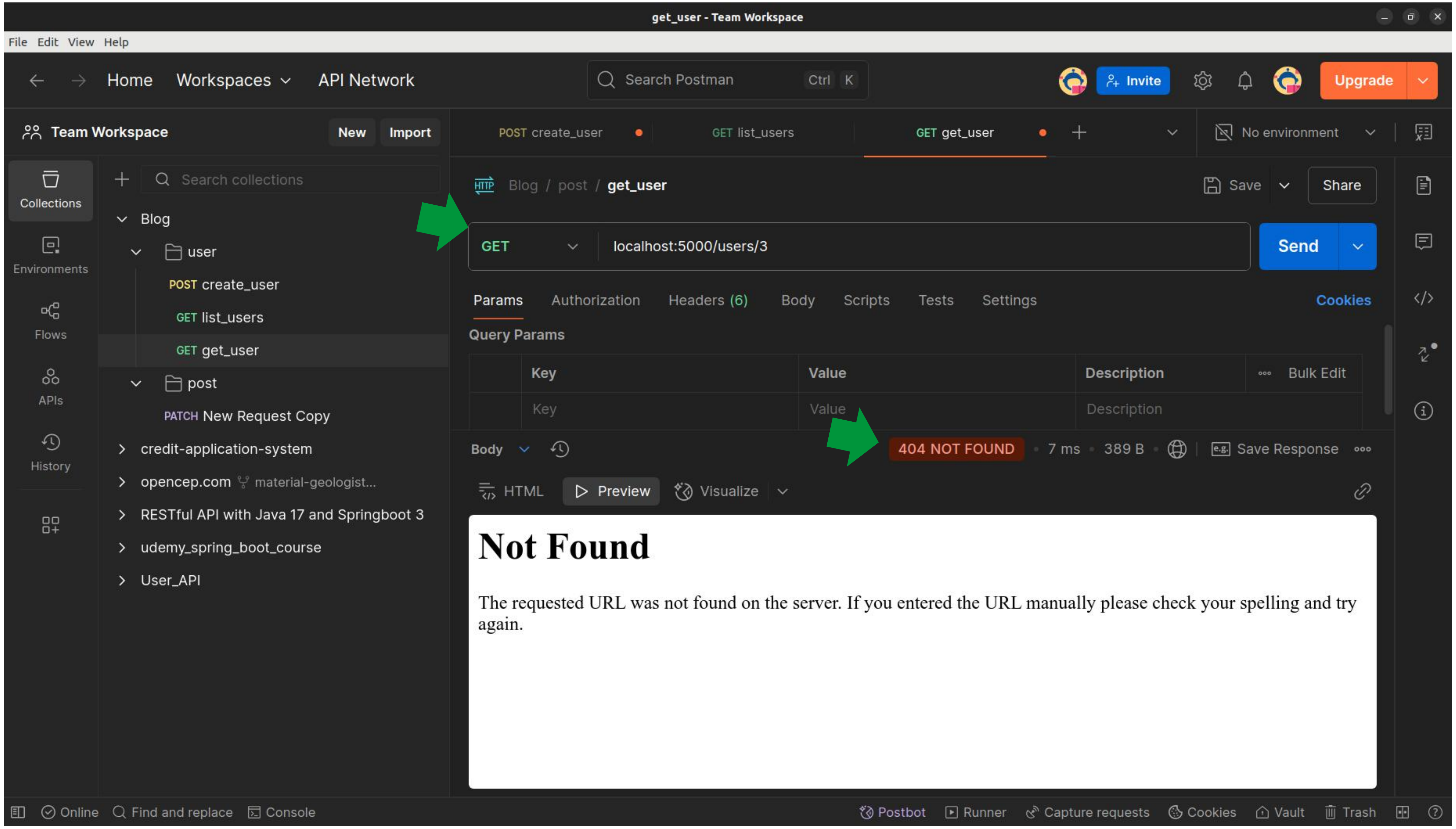


```
GET localhost:5000/users/1
```

```
200 OK • 6 ms • 239 B
```

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

Testando as operações de READ no Postman



GET localhost:5000/users/3

Query Params

Key	Value	Description
Key	Value	Description

404 NOT FOUND · 7 ms · 389 B

Not Found

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

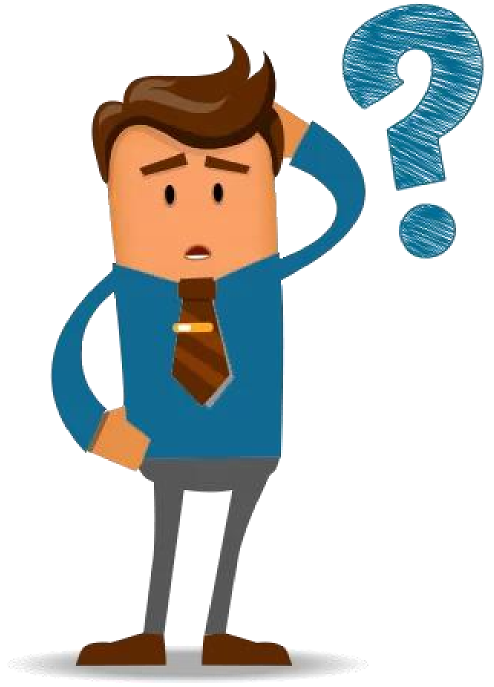
Exercícios

INSTITUTO FEDERAL
Baiano

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

Dúvidas

INSTITUTO FEDERAL
Baiano



PROGRAMAÇÃO WEB II

Curso Técnico Integrado em Informática
Lucas Sampaio Leite

INSTITUTO FEDERAL
Baiano