

Machine Learning Engineer

Python para ML e IA

Desenvolvimento de API com Flask II

Leonardo Pena

/ Seja muito bem vindo



DEFINIÇÕES

Iniciar configuração de Flask com banco de dados e JWT



CONTEXTO

Foco em JWT para autenticação de usuários



DINÂMICA

Parte 2



ESTRUTURAÇÃO

Documentaremos as rotas no Swagger e criaremos rotas `/register` e `/login`.

Objetivo dessa primeira parte



Passo 1

Explicar
JWTManager
e fluxo de
tokens
(Bearer
Token)



Passo 2

Implementar
rotas de
cadastro e
login de
usuário



Passo
3

Mostrar
docstrings
em YAML
para
swagger nas
rotas de auth



Passo 4

Validar
credenciais
no banco de
dados (User)



Passo
5

Comentar
boas
práticas de
segurança
(hash de
senhas)



Autenticação com JWT



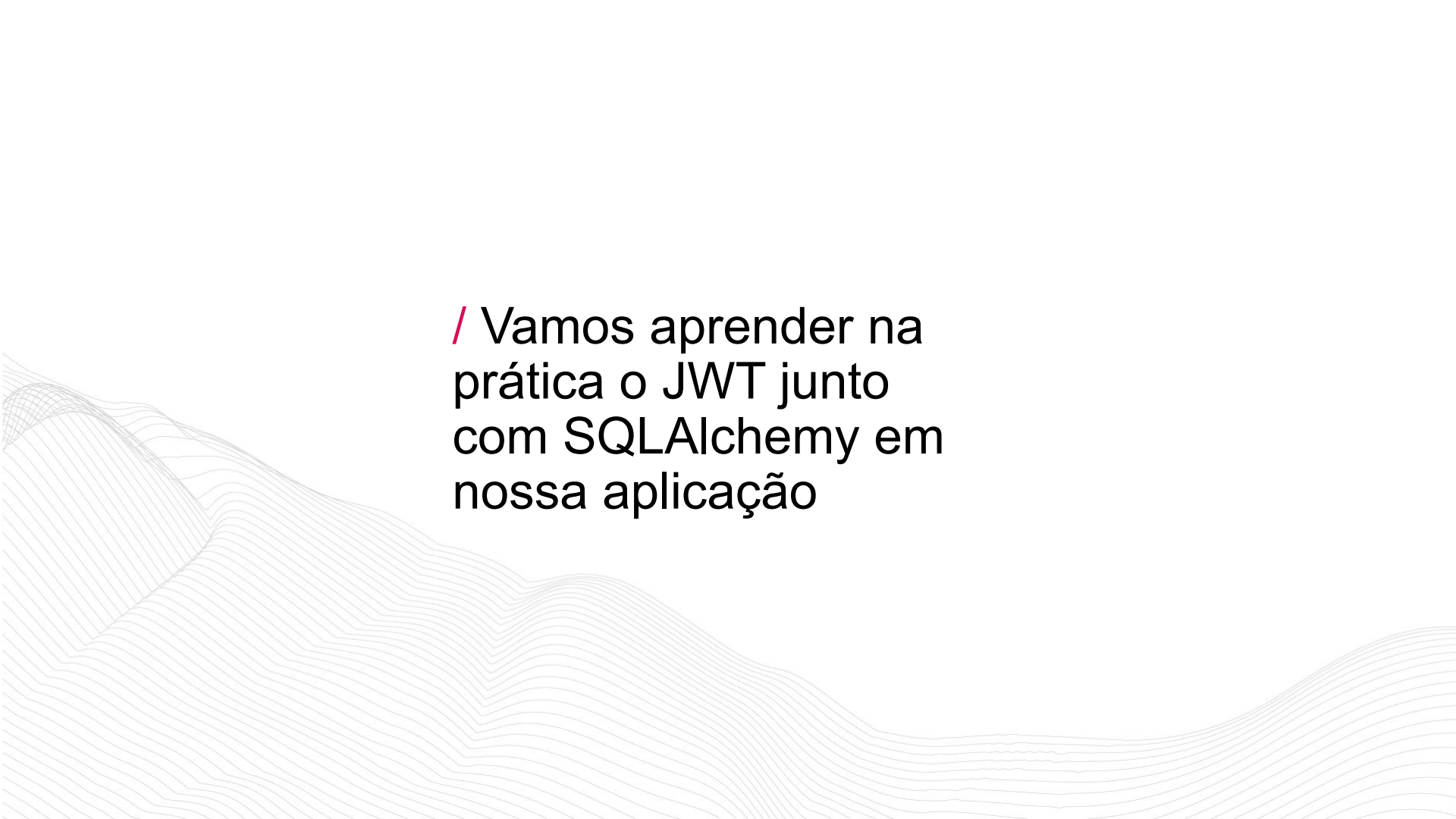
/ Autenticação

- Como vimos em aulas anteriores, temos diversas opções de autenticação, como:
 - Auth Básica
 - JWT

A decorative background consisting of numerous thin, wavy, light gray lines that create a sense of movement and depth, primarily concentrated in the lower half of the slide.

/ Autenticação JWT (JSON Web Token)

- Padrão para autenticação e troca segura de informações em formato compactado.
- Se divide em três partes (Header, Payload e Signature) que garantem a estrutura e a validade dos dados.
- Após o login, o servidor gera um token que o cliente envia em cada requisição para validar seu acesso.
- O servidor não precisa manter sessões, pois a verificação do token ocorre no próprio conteúdo assinado.
- É amplamente utilizado em APIs REST, sistemas de login, Single Sign-On (SSO) e microserviços.



/ Vamos aprender na
prática o JWT junto
com SQLAlchemy em
nossa aplicação

/ Autenticação JWT

Prática

- Vamos começar instalando a biblioteca necessária

```
% pip install Flask-JWT-Extended
```

- Importar o necessário

```
3 from flask_jwt_extended import (  
4     JWTManager, create_access_token,  
5     jwt_required, get_jwt_identity  
6 )  
7
```

- E instanciar o JWTManager

```
11 db = SQLAlchemy(app)  
12 jwt = JWTManager(app)  
13
```


/ Fluxo de Login com JWT

- Usuário chama */login* passando username e password
- Se credenciais válidas, geramos token com *create_access_token(identity=user.id)*
- Resposta: *{"access_token": "<jwt_token>"}*
- Em chamadas subsequentes, usuário envia *Authorization: Bearer <token>*
- Rotas protegidas exigem *@jwt_required()* no decorator

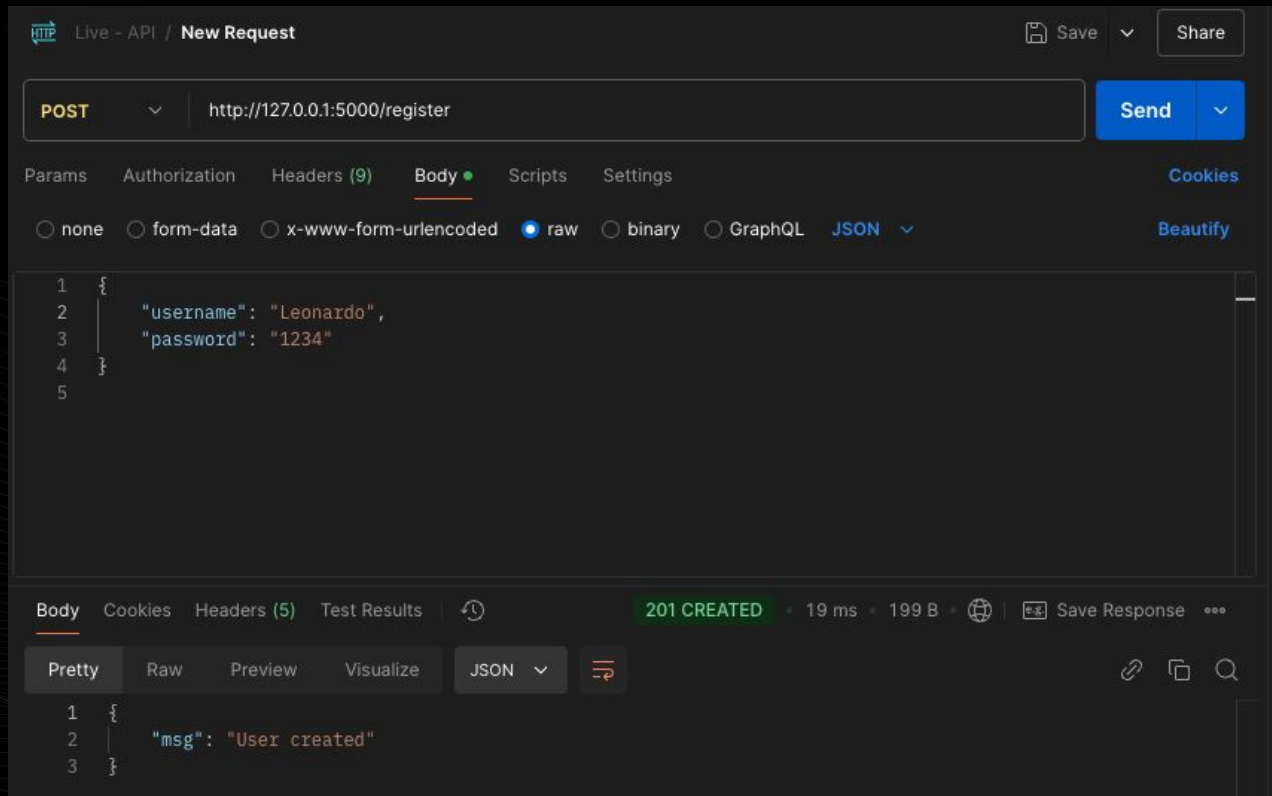
/

Vamos criar nossa primeira rota pra ver funcionando

```
@app.route('/register', methods=['POST'])
def register_user():
    """
    Registra um novo usuário.
    ---
    parameters:
      - in: body
        name: body
        required: true
        schema:
          type: object
          properties:
            username:
              type: string
            password:
              type: string
    responses:
      201:
        description: Usuário criado com sucesso
      400:
        description: Usuário já existe
    """
    data = request.get_json()
    if User.query.filter_by(username=data['username']).first():
        return jsonify({"error": "User already exists"}), 400
    new_user = User(username=data['username'], password=data['password'])
    db.session.add(new_user)
    db.session.commit()
    return jsonify({"msg": "User created"}), 201
```

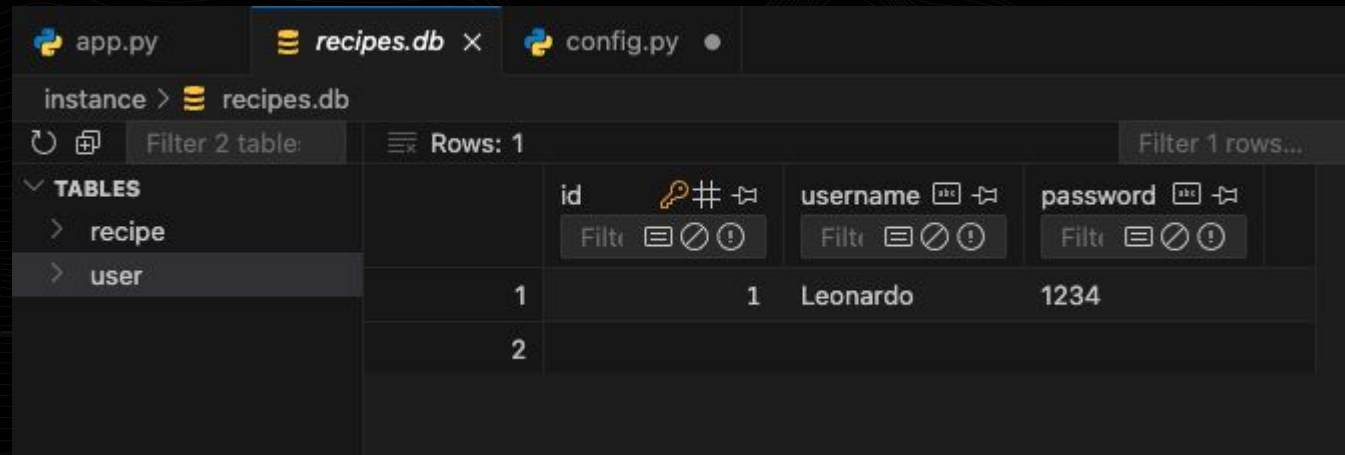
/

Agora é só registrar o usuário no postman



/

Volte ao vscode e veja o usuário criado!



The screenshot shows the VS Code interface with a database connection to 'recipes.db'. The 'TABLES' section on the left lists 'recipe' and 'user'. The 'user' table is selected, and its data is displayed in a table view. The table has three columns: 'id', 'username', and 'password'. The first row shows 'id' 1, 'username' 'Leonardo', and 'password' '1234'. The second row shows 'id' 2 and empty fields for 'username' and 'password'.

	id	username	password
1	1	Leonardo	1234
2	2		

/

Vamos criar nossa rota de login agora!

```
56 @app.route('/login', methods=['POST'])
57 def login():
58     """
59     Faz login do usuário e retorna um JWT.
60     ---
61     parameters:
62         - in: body
63           name: body
64           required: true
65           schema:
66               type: object
67               properties:
68                   username:
69                       type: string
70                   password:
71                       type: string
72     responses:
73         200:
74             description: Login bem sucedido, retorna JWT
75         401:
76             description: Credenciais inválidas
77     """
78     data = request.get_json()
79     user = User.query.filter_by(username=data['username']).first()
80     if user and user.password == data['password']:
81         # Converter o ID para string
82         token = create_access_token(identity=str(user.id))
83         return jsonify({"access_token": token}), 200
84     return jsonify({"error": "Invalid credentials"}), 401
```




/

Vamos criar uma rota simples, para demonstrar o jwt

```
86 @app.route('/protected', methods=['GET'])
87 @jwt_required()
88 def protected():
89     current_user_id = get_jwt_identity() # Retorna o 'identity' usado na criação do token
90     return jsonify({"msg": f"Usuário com ID {current_user_id} acessou a rota protegida."}), 200
91
92
```

Primeiro testando sem passar o token



Live - API / New Request

GET http://127.0.0.1:5000/protected Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (5) Test Results 401 UNAUTHORIZED 8 ms 220 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "msg": "Missing Authorization Header"  
3 }
```

/

E passando o token gerado



Live - API / **protected** Save Share

GET ⌵ `http://127.0.0.1:5000/protected` Send ⌵

Params **Auth** • Headers (8) Body Scripts Settings Cookies

Auth Type

Bearer Token ⌵

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

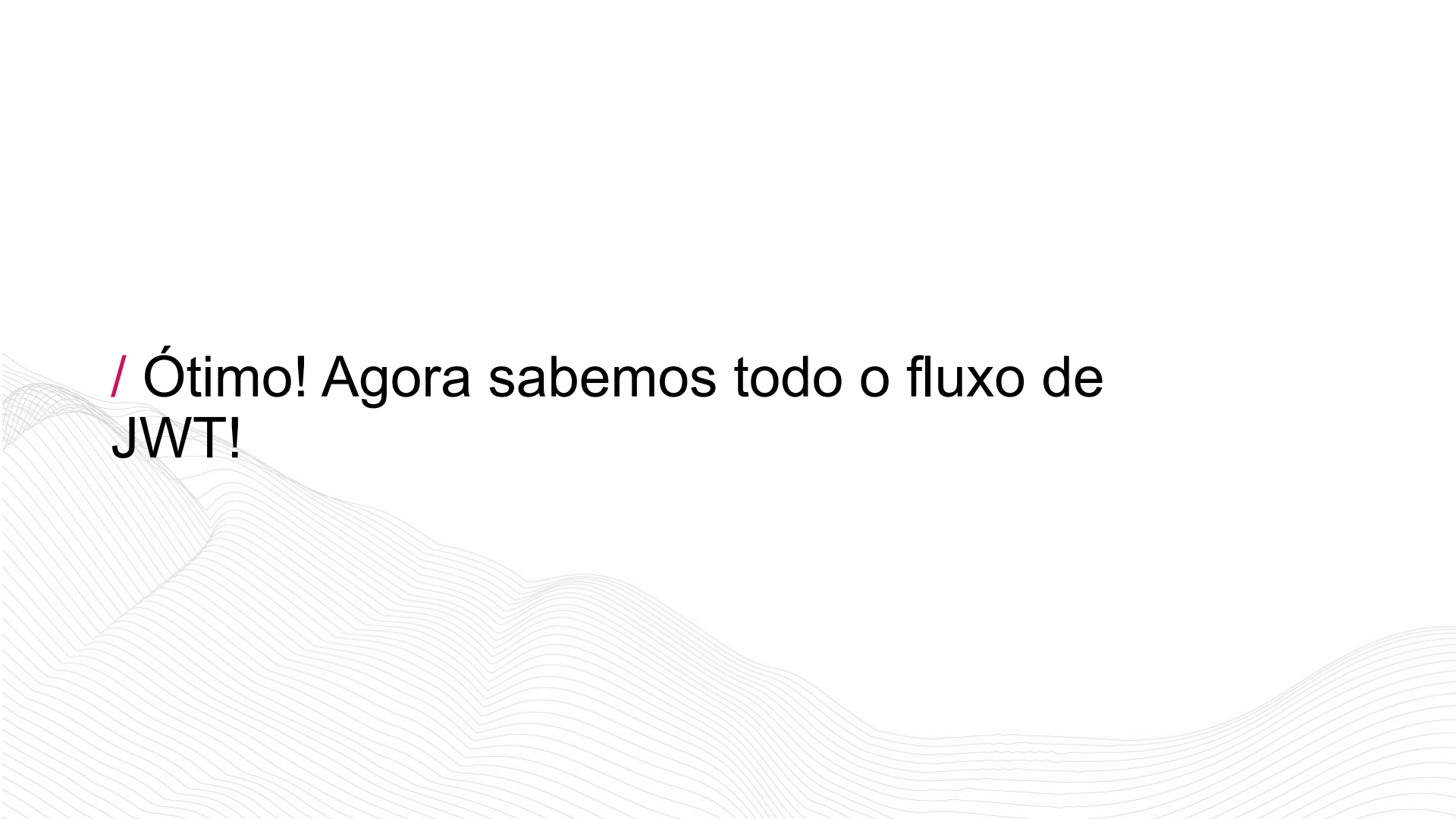
Token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1

Body ⌵ 🔄 200 OK • 4 ms • 229 B • 🌐 💾 Save Response ⋮

Pretty Raw Preview Visualize **JSON** ⌵ 🔍

```
1 {
2   "msg": "Usuário com ID 1 acessou a rota protegida."
3 }
```

The background of the slide features a series of thin, light gray, wavy lines that create a sense of depth and movement, 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.

**/ Ótimo! Agora sabemos todo o fluxo de
JWT!**



/ Revisão

- */register* -> cria usuário (sem token)
- */login* -> autentica e retorna token
- **No próximo vídeo: */recipes* e */scrape* exigirão token**
- ***/apidocs* para documentação**
- **Aplicação ganha forma de API robusta**

POSTECH

FIAP + alura