

Machine Learning Engineer

Python para ML e IA

Desenvolvimento de API com Flask II

Leonardo Pena

/ Seja muito bem vindo



DEFINIÇÕES

Rotas de CRUD para
Recipe e scraping
autenticado



CONTEXTO

Uso de JWT nas rotas
com `@jwt_required()`



DINÂMICA

Parte 3

Objetivo dessa primeira parte



Passo 1

Criar rotas
`/recipes`
(POST e GET)
com filtros e
caching



Passo 2

Rotas
`/recipes/<int:recipe_id>` (PUT e
DELETE)



Passo 3

Exigir token
JWT em
todas as
rotas de
CRUD



Passo 4

Mostrar
scraping
(`/scrape/title`,
`/scrape/content`)
autenticado



Passo 5

Concluir
aplicação e
discutir
próximos
passos



Rotas de receita





/ API de Receitas Gourmet

- Vamos começar criando rotas de receita com os seguintes propósitos:
 - */recipes (POST)* -> cria receita, token obrigatório
 - */recipes (GET)* -> lista receitas (com cache) + filtros
 - */recipes/<id> (PUT)* -> atualiza receita existente
 - */recipes/<id> (DELETE)* -> remove receita específica
 - Todos usam *@jwt_required()* para validação do token

/

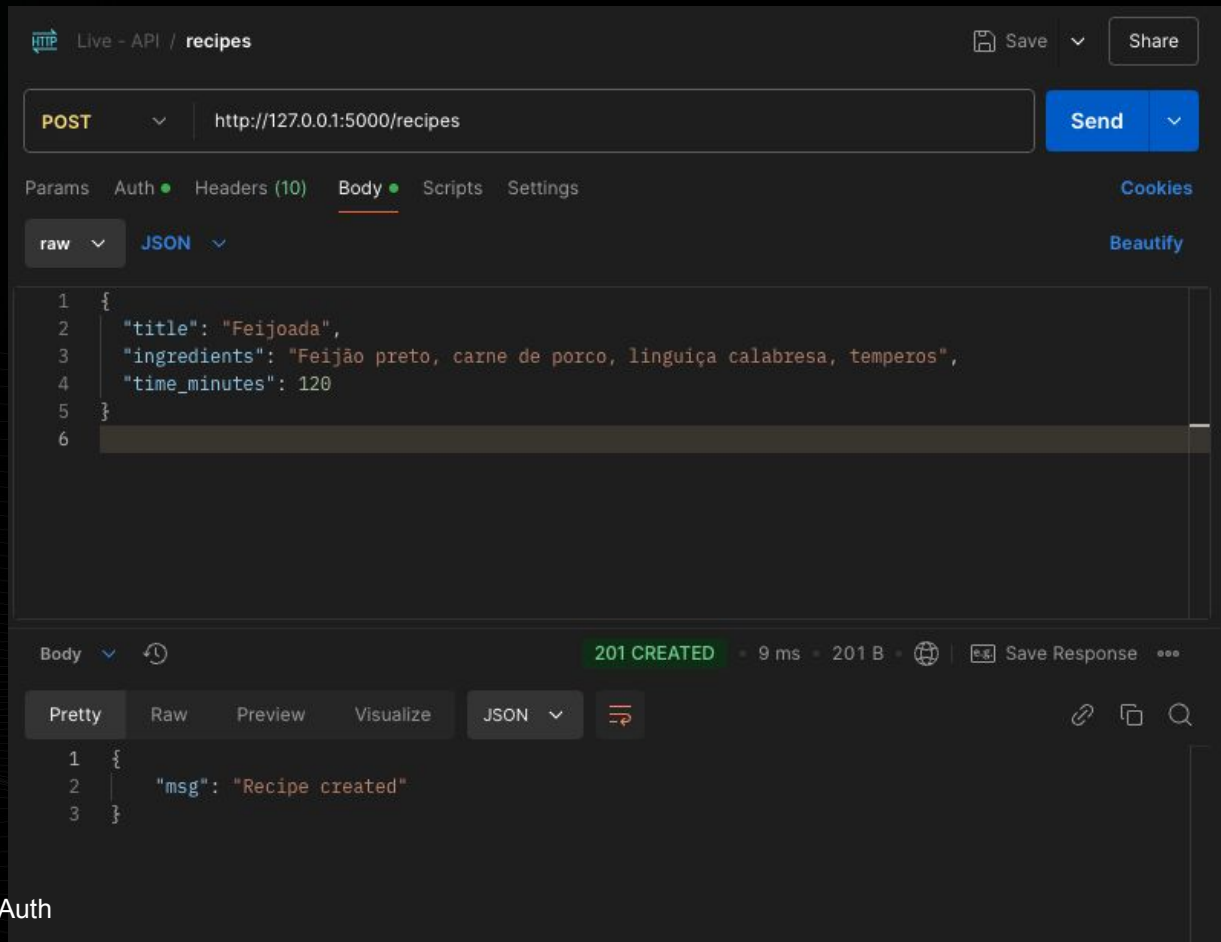
Vamos começar criando nossa rota (POST) de receitas

```
95 @app.route('/recipes', methods=['POST'])
96 @jwt_required()
97 def create_recipe():
98     """
99     Cria uma nova receita.
100     ---
101     security:
102     - BearerAuth: []
103     parameters:
104     - in: body
105       name: body
106       schema:
107         type: object
108         required: true
109         properties:
110           title:
111             type: string
112           ingredients:
113             type: string
114           time_minutes:
115             type: integer
116     responses:
117       201:
118         description: Receita criada com sucesso
119       401:
120         description: Token não fornecido ou inválido
121     """
122     data = request.get_json()
123     new_recipe = Recipe(
124         title=data['title'],
125         ingredients=data['ingredients'],
126         time_minutes=data['time_minutes']
127     )
128     db.session.add(new_recipe)
129     db.session.commit()
130     return jsonify({"msg": "Recipe created"}), 201
```

Agora é só registrar a receita no postman



Obs: não esqueça de autenticar em Auth



/

Nosso banco já vai ter atualizado!!

ir ~/Documents/FIAP/MLET/material_plataforma/aula3-new/app.py

Filter 2 tables Rows: 1 Filter 1 rows...

TABLES	id	title	ingredients	time_mi...
> recipe	1	Feijoada	Feijão preto, carne de porco, linguiça calabresa, tem...	120
> user	2			

/

Mas e se quisermos ver a receita?

Vamos criar um GET

```
@app.route('/recipes', methods=['GET'])
def get_recipes():
    """
    Lista receitas com filtros opcionais.
    """
    parameters:
    - in: query
      name: ingredient
      type: string
      required: false
      description: Filtra por ingrediente
    - in: query
      name: max_time
      type: integer
      required: false
      description: Tempo máximo de preparo (minutos)

    responses:
    200:
      description: Lista de receitas filtradas
      schema:
        type: array
        items:
          type: object
          properties:
            id:
              type: integer
            title:
              type: string
            time_minutes:
              type: integer

    ingredient = request.args.get('ingredient')
    max_time = request.args.get('max_time', type=int)

    query = Recipe.query
    if ingredient:
        query = query.filter(Recipe.ingredients.ilike(f'%{ingredient}%'))
    if max_time is not None:
        query = query.filter(Recipe.time_minutes <= max_time)

    recipes = query.all()
    return jsonify([
        {
            "id": r.id,
            "title": r.title,
            "ingredients": r.ingredients,
            "time_minutes": r.time_minutes
        }
        for r in recipes
    ])

```

/

E no postman veríamos



Live - API / New Request

GET http://127.0.0.1:5000/recipes Send

Params Auth Headers (7) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	⋮ Bulk Edit
	Key	Value	Description	

Body 200 OK • 12 ms • 332 B Save Response

Pretty Raw Preview Visualize JSON

```
1  [
2    {
3      "id": 1,
4      "ingredients": "Feijão preto, carne de porco, linguiça calabresa, temperos",
5      "time_minutes": 120,
6      "title": "Feijoada"
7    }
8  ]
```

/

Mas e se quisermos alterar a receita?

Vamos criar um PUT

```
@app.route('/recipes/<int:recipe_id>', methods=['PUT'])
@jwt_required()
def update_recipe(recipe_id):
    """
    Atualiza uma receita existente.
    """
    security:
    - BearerAuth: []
    parameters:
    - in: path
      name: recipe_id
      required: true
      type: integer
    - in: body
      name: body
      schema:
        type: object
        properties:
          title:
            type: string
          ingredients:
            type: string
          time_minutes:
            type: integer
    responses:
      200:
        description: Receita atualizada
      404:
        description: Receita não encontrada
      401:
        description: Token não fornecido ou inválido
    """
    data = request.get_json()
    recipe = Recipe.query.get_or_404(recipe_id)
    if 'title' in data:
        recipe.title = data['title']
    if 'ingredients' in data:
        recipe.ingredients = data['ingredients']
    if 'time_minutes' in data:
        recipe.time_minutes = data['time_minutes']

    db.session.commit()
    return jsonify({"msg": "Recipe updated"}), 200
```

/

E no postman veríamos



Live - API / recipes

PUT http://127.0.0.1:5000/recipes/1 Send

Params Auth Headers (10) Body Scripts Settings Cookies Beautify

raw JSON

```
1 {
2   "title": "Feijoada Atualizada",
3   "ingredients": "Feijão preto, carne de porco, linguiça calabresa, etc.",
4   "time_minutes": 180
5 }
6
```




















Body 200 OK • 40 ms • 196 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "msg": "Recipe updated"
3 }
```

/

Nosso banco já vai ter atualizado!!

Rows: 1					Filter 1 rows...
	id  # 	title  	ingredients  	time_mi... # 	
	Filter...   	Filter...   	Filter...   	Filter...   	
1	1	Feijoada Atualizada	Feijão preto, carne de porco, linguiça calabresa, etc.	180	
2					

/
Tente você agora criar uma
de DELETE!



Documentação



/ Documentação com Swagger



- Swagger é uma documentação interativa para APIs.
- Facilita a visualização e teste de endpoints.
- Integração simples com Flask usando bibliotecas como flasgger.
- Permite definir parâmetros e respostas diretamente no código.
- Garante padronização e validação dos dados da API.

/ API de Receitas Gourmet

- Via pip install, instale o *flasgger*

```
% pip install flasgger
```

- Importe no código

```
from flasgger import Swagger
```

- Configure o Swagger após a criação do app

```
9 app = Flask(__name__)
10 app.config.from_object('config')
11
12 db = SQLAlchemy(app)
13 jwt = JWTManager(app)
14 swagger = Swagger(app)
15 |
```

/ Pronto! Agora se rodar a aplicação e ir na rota de */apidocs*, você já verá a documentação

Catálogo de Receitas Gourmet 0.0.1

[/apispec_1.json](#)

powered by [Flasgger](#)

[Terms of service](#)

default



POST **/login** Faz login do usuário e retorna um JWT.

post_login

GET **/recipes** Lista receitas com filtros opcionais.

get_recipes

POST **/recipes** Cria uma nova receita.

post_recipes

PUT **/recipes/{recipe_id}** Atualiza uma receita existente.

put_recipes_recipe_id_

POST **/register** Registra um novo usuário.

post_register

[Powered by [Flasgger](#) 0.9.7.1]



/ API de Receitas Gourmet

- Agora você pode testar as rotas via swagger!
- Repare que há implementação de filtros também:
 - Filtros opcionais para busca de receitas.
 - Filtra por ingrediente específico no campo 'ingredients'.
 - Permite limitar o tempo máximo de preparo com 'max_time'.
 - Consulta dinâmica combinando múltiplos filtros.

POSTECH

FIAP + alura