Flask

Equipe:

1. Lorene Marques
2. Felipe Henrique
3. Eduardo Freire
4. Rodrigo Gomes

Flask

Flask é um pequeno framework web escrito em Python e baseado na biblioteca WSGI Werkzeug e na biblioteca de Jinja2. Flask está disponível sob os termos da Licença BSD.

Flask tem a flexibilidade da linguagem de programação Python e provê um modelo simples para desenvolvimento web. Uma vez importando no Python, Flask pode ser usado para economizar tempo construindo aplicações web.

Criação e ativação do ambiente virtual

1. Instalar o Python, preferencialmente Python 3.
2. Criar uma pasta que armazenará o projeto.
3. Acessar a pasta em questão através do Terminal/Cmd.
4. Se (usando Windows):
    a. Digite “py -3 -m venv nome_do_ambiente_virtual”.
    b. Digite “nome_do_ambiente_virtual\Scripts\activate”.
5. Caso não:
    a. Digite “python3 -m venv nome_do_ambiente_virtual”
    b. Digite “nome_do_ambiente_virtual/bin/activate”.

Instalação do Flask

1. Certificar que o anbiente virtual está ativado.
2. Digite “pip install Flask”.

Preparação do Flask para uso

1. Se(usando_windows):
    a. Digitar o comando “set FLASK_APP=nome_do_projeto && set FLASK_ENV=development && flask run”
2. Se não:
    a. Digitar o comando “export FLASK_APP=nome_do_projeto && export FLASK_ENV=development && flask run”

EXEMPLO PRÁTICO
Mini Blog

Flask Restful API

- Representational State Transfer (REST) é um estilo de programa de desenvolvimento web que se refere à separação lógica dos recursos da API, habilitando fácil acesso, manipulação e dimensionamento.

- Através de requisições HTTP, utiliza os seguintes componentes: GET, POST, PUT e DELETE.

UserView.py

```python
#/src/views/UserView

from flask import request, json, Response, Blueprint, g
from ..models.UserModel import UserModel, UserSchema
from ..shared.Authentication import Auth

user_api = Blueprint('user_api', __name__)
user_schema = UserSchema()

@user_api.route('/', methods=['POST'])
def create():
  """
  Create User Function
  """
  req_data = request.get_json()
  data, error = user_schema.load(req_data)

  if error:
    return custom_response(error, 400)

  # check if user already exist in the db
  user_in_db = UserModel.get_user_by_email(data.get('email'))
  if user_in_db:
    message = {'error': 'User already exist, please supply another email address'}
    return custom_response(message, 400)

  user = UserModel(data)
  user.save()
  ser_data = user_schema.dump(user).data
  token = Auth.generate_token(ser_data.get('id'))
  return custom_response({'jwt_token': token}, 201)

@user_api.route('/', methods=['GET'])
@Auth.auth_required
def get_all():
  """
  Get all users
  """
  users = UserModel.get_all_users()
  ser_users = user_schema.dump(users, many=True).data
  return custom_response(ser_users, 200)

```

```python
@user_api.route('/<int:user_id>', methods=['GET'])
@Auth.auth_required
def get_a_user(user_id):
  """
  Get a single user
  """
  user = UserModel.get_one_user(user_id)
  if not user:
    return custom_response({'error': 'user not found'}, 404)

  ser_user = user_schema.dump(user).data
  return custom_response(ser_user, 200)

@user_api.route('/me', methods=['PUT'])
@Auth.auth_required
def update():
  """
  Update me
  """
  req_data = request.get_json()
  data, error = user_schema.load(req_data, partial=True)
  if error:
    return custom_response(error, 400)

  user = UserModel.get_one_user(g.user.get('id'))
  user.update(data)
  ser_user = user_schema.dump(user).data
  return custom_response(ser_user, 200)

@user_api.route('/me', methods=['DELETE'])
@Auth.auth_required
def delete():
  """
  Delete a user
  """
  user = UserModel.get_one_user(g.user.get('id'))
  user.delete()
  return custom_response({'message': 'deleted'}, 204)

@user_api.route('/me', methods=['GET'])
@Auth.auth_required
def get_me():
  """
  Get me
  """
  user = UserModel.get_one_user(g.user.get('id'))
  ser_user = user_schema.dump(user).data
  return custom_response(ser_user, 200)

```

```python
@user_api.route('/login', methods=['POST'])
def login():
    """
    User Login Function
    """
    req_data = request.get_json()

    data, error = user_schema.load(req_data, partial=True)
    if error:
        return custom_response(error, 400)
    if not data.get('email') or not data.get('password'):
        return custom_response({'error': 'you need email and password to sign in'}, 400)
    user = UserModel.get_user_by_email(data.get('email'))
    if not user:
        return custom_response({'error': 'invalid credentials'}, 400)
    if not user.check_hash(data.get('password')):
        return custom_response({'error': 'invalid credentials'}, 400)
    ser_data = user_schema.dump(user).data
    token = Auth.generate_token(ser_data.get('id'))
    return custom_response({'jwt_token': token}, 200)



def custom_response(res, status_code):
    """
    Custom Response Function
    """
    return Response(
        mimetype="application/json",
        response=json.dumps(res),
        status=status_code
    )
```

POSTMAN

•O Postman é uma ferramenta que tem como objetivo testar serviços RESTful (Web APIs) por meio do envio de requisições HTTP e da análise do seu retorno.

POST http://127.0.0.1:5000/api/v1/users/ Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Cookies Code

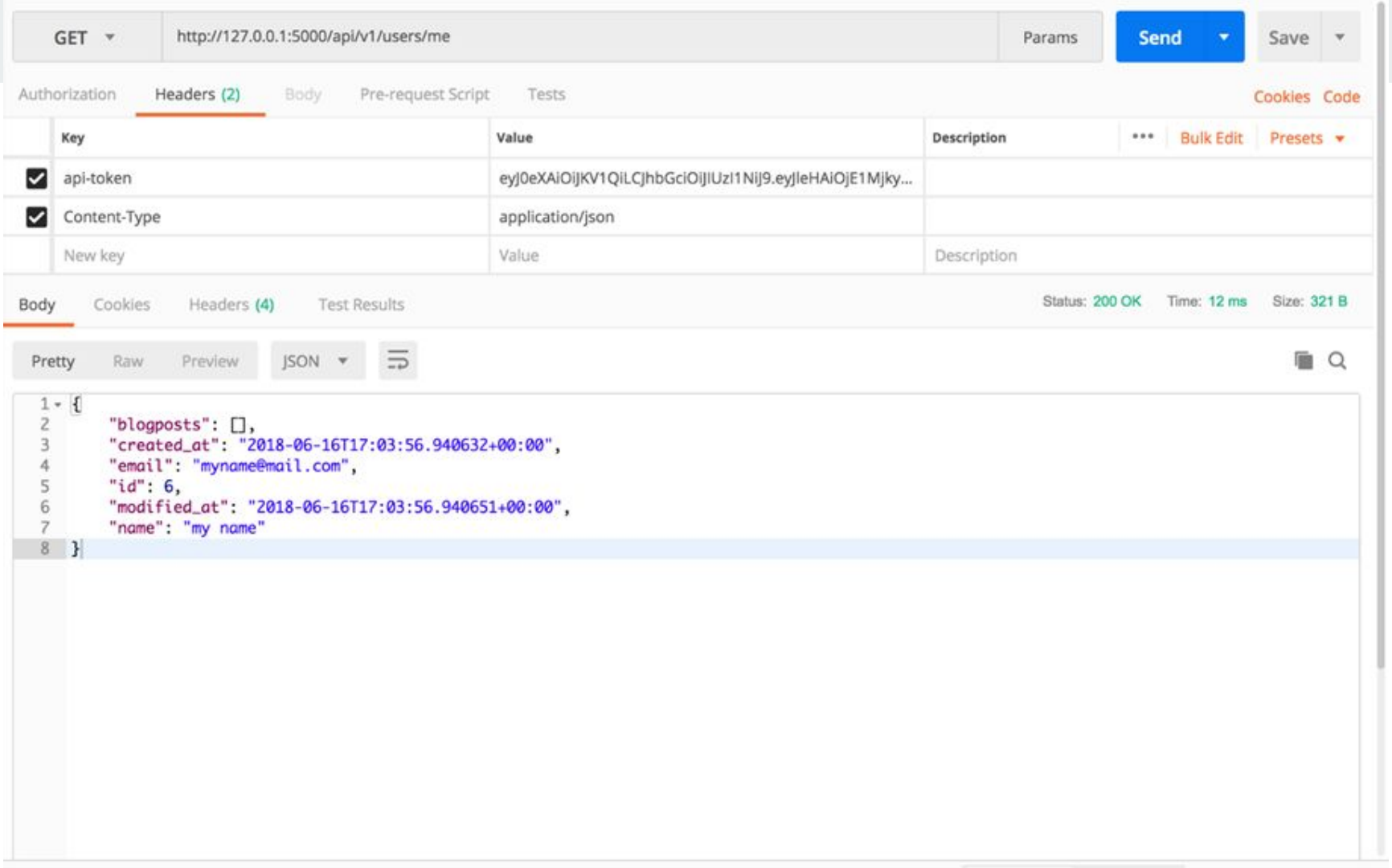
form-data x-www-form-urlencoded raw binary JSON (application/json)

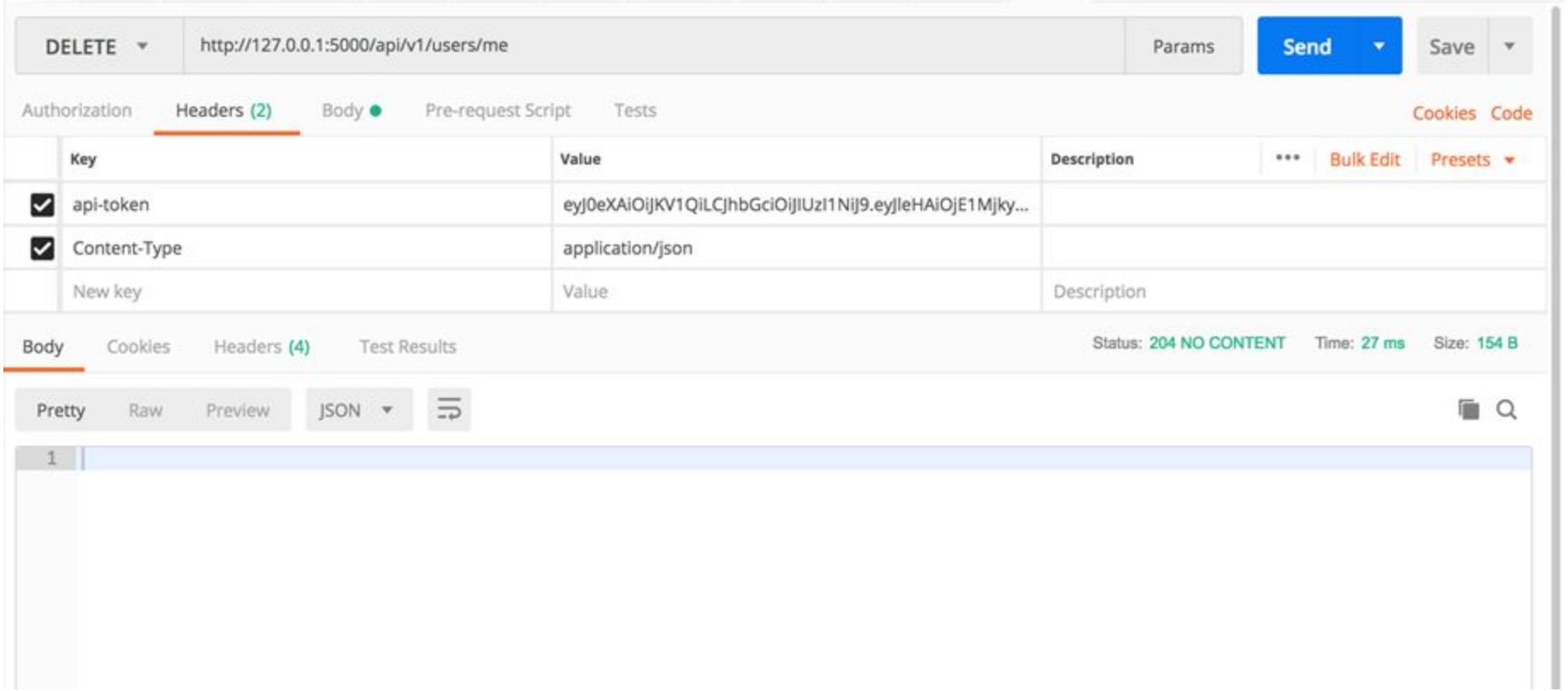
```
{
    "email": "myname@mail.com",
    "password": "Passw0rd!",
    "name": "my name"
}
```

Body Cookies Headers (4) Test Results Status: 201 CREATED Time: 222 ms Size: 308 B

Pretty Raw Preview JSON

```
{
    "jwt_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MjkyNTUwMzYsImlhdCI6MTUyOTE2ODYzNiwic3ViIjo2fQ.vkasp4IYulp-cy_FCT1qvPs6G
}
```


GET
http://127.0.0.1:5000/api/v1/users/me
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Cookies
Code
Key
Value
Description
Bulk Edit
Presets
api-token
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1Mjky...
Content-Type
application/json
New key
Value
Description
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 12 ms
Size: 321 B
Pretty
Raw
Preview
JSON
{
"blogposts": [],
"created_at": "2018-06-16T17:03:56.940632+00:00",
"email": "myname@mail.com",
"id": 6,
"modified_at": "2018-06-16T17:03:56.940651+00:00",
"name": "my name"
}


DELETE
http://127.0.0.1:5000/api/v1/users/me
Params
Send
Save
Authorization
Headers (2)
Body
Pre-request Script
Tests
Cookies
Code
Key
Value
Description
Bulk Edit
Presets
api-token
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1Mjky...
Content-Type
application/json
New key
Value
Description
Body
Cookies
Headers (4)
Test Results
Status: 204 NO CONTENT
Time: 27 ms
Size: 154 B
Pretty
Raw
Preview
JSON
1

Integração Flask + PostgreSQL

Segurança no flask

Pela documentação do flask, é apresentada várias opções viáveis de caminhos que podem ser seguidos pelo flask para tornar a comunicação cliente-servidor segura.

Menções:

- Cross-Site Scripting (XSS)
- JSON Security / JSON WebToken / CORS
- Security Headers

Fonte:

http://flask.pocoo.org/docs/1.0/security/

Flask em produção

Como a própria documentação fala:

"While lightweight and easy to use, Flask's built-in server is not suitable for production as it doesn't scale well and by default serves only one request at a time."

Uma solução possível é executar um container Docker em uma aplicação Flask usando o uWSGI.

Fonte:

https://medium.com/@cirolini/docker-flask-e-uwsgi-d10e58c56489

https://uwsgi-docs.readthedocs.io/en/latest/