Continuação da Blueprint e Iniciando organização para utilizar SQLAlchemy

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA

- Continuação da aula passada
- Criar instância de um banco em JSON
- Criar 3 endpoints para trabalhar com esse banco
- Introdução ao SQLAlchemy
- Modelagem de dados

Continuação da aula passada

- Baseado no código da aula passada (27/07/2022), dentro da pasta src/app iremos criar uma pasta chamada **db**, que será responsável por persistir os dados salvos num banco de dados criado em JSON e dentro desta pasta, iremos criar um arquivo chamado __init__.py e nele iremos criar 2 funções no momento.

Criar instância de um banco em JSON

```
from flask import json

def save(data):
  json_object = json.dumps(data, indent=4)
  with open("src/app/db/technologies.json", "w") as outfile:
      outfile.write(json_object)
def read():
  try:
    with open('src/app/db/technologies.json', 'r') as openfile:
      json_object = json.load(openfile)
      return json_object
  except:
    return None
```

Criar 3 endpoints para trabalhar com esse banco

- Nesse momento, iremos refatorar o endpoint de POST do technologies, para que de fato, ele possa salvar dados na nossa aplicação.
- Para tal, iremos criar uma pasta nova chamada Utils, e dentro dela iremos adicionar um arquivo __init__.py para a aplicação e escrever os seguintes métodos

```
def exist_value(request_json, data_in_db):
  for json in data_in_db:
    if json['id'] == request_json['id'] or json['tech'] == request_json['tech']:
      return True
  return False
```

Criar 3 endpoints para trabalhar com esse banco

- O propósito desse método é verificar se todos os dados necessários já estão na requisição.

```
def exist_key(request_json, list_keys):
  keys_not_have_in_request = []
  for key in list_keys:
    if key in request_json:
      continue
    else:
      keys_not_have_in_request.append(key)
  if len(keys_not_have_in_request) == 0:
    return request_json
  return {"error":  f"Está faltando o item {keys_not_have_in_request}" }
```

Criar 3 endpoints para trabalhar com esse banco

- Nesse momento, iremos refatorar o endpoint de POST do technologies:

```
from src.app.utils import exist_key #Junto com as importações
from src.app.db import read, save  #Junto com as importações
  data = exist_key(request.get_json()) #Esse bloco até a ultima linha, adicionar na
função add_new_technology
  list_keys = ["id", "tech"]
  data = exist_key(request.get_json(), list_keys
  if 'error' in data:
    return jsonify(data), 400
  techs = read()
```

Criar 3 endpoints para trabalhar com esse banco

- Nesse momento, iremos refatorar o endpoint de POST do technologies:

```
  if techs == None or len(techs) == 0:
    save([data])
    return jsonify(data), 201
  if exist_value(data, techs):
    return jsonify({"error": "Algum dos items que foi enviado, já existe no banco de
dados"}), 400
  techs.append(data)
  save(techs)
  return jsonify(techs), 201
```

Criar 3 endpoints para trabalhar com esse banco

- Nesse momento, iremos criar o endpoint de DELETE do technologies:

```
@technology.route('/<int:id>', methods = ["DELETE"])
def delete_technology(id):
  techs = read()
  if techs == None or len(techs) == 0:
    return {"error": f"id {id} não foi encontrado"}, 404
  for index, data in enumerate(techs):
    if data['id'] == id:
      techs.pop(index)
      save(techs)
      return jsonify({"message": f"O id {id} foi deletado com sucesso"}), 200
  return jsonify({"error": f"id {id} não foi encontrado"}), 404
```

Introdução ao SQLAlchemy

É um ORM (Object-relational mapping) que basicamente permite mapear as tabelas do banco em classes e objetos de forma fácil e prática. Para exemplificar vamos continuar a usar nosso exemplo anterior da tabela de usuários, primeiro vamos deletar a tabela.

DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!

SENAI

<LAB365>