# Introdução ao SQLAlchemy

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

# AGENDA DA SEMANA

- Introdução ao SQLAlchemy, configuração da API e Flask-Migrate
- Modelagem de dados e documentação com SwaggerUI
- Trabalhando com relacionamentos de dados SQL com SQLAlchemy
- Flask-Script

# AGENDA

- Introdução ao SQLAlchemy
- Instalação das dependências
- Configuração da API
- Modelagem de dados
- Flask-Migrate

# 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.

# Instalação das Dependências

Nessa etapa, com a virtualização ativada, iremos instalar as seguintes dependências:

- **Flask-SQLAlchemy**
- **psycopg2**
- **flask-marshmallow**
- **marshmallow-sqlalchemy**
- **Flask-Migrate**

Usando o comando: **poetry add Flask-SQLAlchemy psycopg2 flask-marshmallow marshmallow-sqlalchemy Flask-Migrate**

# Configuração da API

- Nesse momento, iremos refatorar as configurações básicas da nossa API e iremos adicionar novos módulos dentro do __init__.py da pasta **app**.

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_marshmallow import Marshmallow
from src.app.config import app_config
from src.app.routes import routes
from flask_migrate import Migrate
db = SQLAlchemy()
ma = Marshmallow()
```

# Configuração da API

- E por fim, abaixo iremos criar a função create_app para definir a instância do app.

```
def create_app():
  app = Flask(__name__)
  app.config.from_object(app_config[os.getenv('FLASK_ENV')])
  db.init_app(app)
  ma.init_app(ma)
  routes(app)
  Migrate(app=app, db=db, directory="./src/app/migrations")
  return app
```

- Feito isso, iremos para o arquivo app.py e iremos refarorar seu código.

# Configuração da API

```python
from src.app import create_app


app = create_app()

if __name__ == "__main__":
  app.run()
```

# Configuração da API

- Feito a configuração inicial da aplicação, iremos conectá-la ao banco de dados PostgreSQL, para tal, precisamos configurar as devidas informações nas suas variáveis de ambiente no arquivo .env

```
SQLALCHEMY_DATABASE_URI=postgresql://user:password@localhost:5432/database
SQLALCHEMY_TRACK_MODIFICATIONS = False
```

- Feito isso, crie esta database no seu banco de dados para a utilizarmos.

# Configuração da API

- Agora, iremos na pasta config no arquivo __init__.py e adicionaremos em cada Classe, 2 atributos, conforme abaixo:

```
class Development(object):
    DEBUG = True
    TESTING = False
    SQLALCHEMY_TRACK_MODIFICATIONS = os.getenv('SQLALCHEMY_TRACK_MODIFICATIONS')
    SQLALCHEMY_DATABASE_URI = os.getenv('SQLALCHEMY_DATABASE_URI')
class Production(object):
    DEBUG = False
    TESTING = False
    SQLALCHEMY_TRACK_MODIFICATIONS = os.getenv('SQLALCHEMY_TRACK_MODIFICATIONS')
    SQLALCHEMY_DATABASE_URI = os.getenv('SQLALCHEMY_DATABASE_URI')
```

# Modelagem de dados

- Nesta parte, iremos criar a primeira parte do modelo de dados, utilizando a mesma ideia das aulas passadas, falando sobre a base de tecnologias. Portanto iremos criar uma pasta chamada src/app/models e dentro, um arquivo chamado technology.py

# Modelagem de dados

- Agora, iremos na pasta config no arquivo __init__.py e adicionaremos em cada Classe, 2 atributos, conforme abaixo:

```
from src.app import db, ma

class Technology(db.Model):
  __tablename__ = 'technologies'
  id = db.Column(db.Integer, autoincrement=True, primary_key=True)
  name = db.Column(db.String(84), nullable=False)
  def __init__(self, name):
    self.name = name
class TechnologySchema(ma.Schema):
  class Meta:
    fields = ('id', 'name')
technology_share_schema = TechnologySchema()
```

# DEVinHouse

Parcerias para desenvolver a sua carreira

## OBRIGADO!

SENAI

<LAB365>