Preparando modelos e criando módulo de autenticação

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA DA SEMANA

- Preparação dos modelos e criando middleware de autenticação usando JWT;
- Evoluindo o módulo para Role Based Access Control;
- Finalizando a aplicação com a documentação do Swagger;
- Revisão do módulo de Flask para o Projeto 2 do módulo 2.

AGENDA

- Finalização e criação dos novos modelos;
- JWT;
- Instalação das dependências;
- Criando o módulo de autenticação;
- Criando endpoints novos e realizando as devidas documentações.

Atualizando modelo de dados - STATE

```
#Dentro da classe State, iremos adicionar esse bloco
  country = DB.relationship("Country", foreign_keys=[country_id])


#Dentro da classe StateSchema, iremos adicionar esse bloco
  country = MA.Nested(country_share_schema)


#Dentro da classe Meta, iremos adicionar esse acrescentar, no fields, o nome:
“country”
```

Atualizando modelo de dados - CITY

```
#Dentro da classe City, iremos adicionar esse bloco
  state = DB.relationship("State", foreign_keys=[state_id])


#Dentro da classe CitySchema, iremos adicionar esse bloco
  state = MA.Nested(state_share_schema)


#Dentro da classe Meta, iremos adicionar esse acrescentar, no fields, o nome: “state”
```

Atualizando modelo de dados - DEVELOPER

```
#Dentro da classe Developer, iremos adicionar esse bloco
  user = DB.relationship("User", foreign_keys=[user_id])


#Dentro da classe DeveloperSchema, iremos adicionar esse bloco
  user = MA.Nested(user_list_share_schema)


#Dentro da classe Meta, iremos adicionar esse acrescentar, no fields, o nome: “user”
```

Criando modelo de dados - PERMISSION

```
from src.app import DB, MA

class Permission(DB.Model):
  __tablename__ = "permissions"
  id = DB.Column(DB.Integer, autoincrement = True, primary_key = True)
  description = DB.Column(DB.String(84), nullable = False)
  def __init__(self, description):
    self.description = description
  @classmethod
  def seed(cls, description):
    permission = Permission(
      description = description
    )
    permission.save()
```

Criando modelo de dados - PERMISSION

```
  def save(self):
    DB.session.add(self)
    DB.session.commit()
class PermissionSchema(MA.Schema):
  class Meta:
    fields = ('id', 'description')
permission_share_schema = PermissionSchema()
permissions_share_schema = PermissionSchema(many = True)
```

Criando modelo de dados - ROLE

```
from src.app import DB, MA
from src.app.models.permission import permissions_share_schema
role_permissions = DB.Table('role_permissions',
                    DB.Column('role_id', DB.Integer, DB.ForeignKey('roles.id')),
                    DB.Column('permission_id', DB.Integer,
DB.ForeignKey('permissions.id'))
                    )
class Role(DB.Model):
  __tablename__ = "roles"
  id = DB.Column(DB.Integer, autoincrement = True, primary_key = True)
  description = DB.Column(DB.String(84), nullable = False)
  permissions = DB.relationship('Permission', secondary=role_permissions,
backref='roles')
```

Criando modelo de dados - ROLE

```
  def __init__(self, description, permissions):
    self.description = description
    self.permissions = permissions
  @classmethod
  def seed(cls, description, permissions):
    role = Role(
      description = description,
      permissions = permissions
    )
    role.save()
  def save(self):
    DB.session.add(self)
    DB.session.commit()
```

Criando modelo de dados - ROLE

```
class RoleSchema(MA.Schema):
  permissions = MA.Nested(permissions_share_schema)
  class Meta:
    fields = ('id', 'description', 'permissions')
role_share_schema = RoleSchema()
roles_share_schema = RoleSchema(many = True)
```

Atualizando modelo de dados - USER

```
from src.app.models.city import City, city_share_schema #Adicionar nas importações
from src.app.models.role import roles_share_schema #Adicionar nas importações


users_roles = DB.Table('users_role',
                       DB.Column('user_id', DB.Integer, DB.ForeignKey('users.id')),
                       DB.Column('role_id', DB.Integer, DB.ForeignKey('roles.id'))
                       )


#Dentro da classe User, iremos adicionar esse bloco
  city = DB.relationship("City", foreign_keys=[city_id])
  roles = DB.relationship('Role', secondary=users_roles, backref='users')
```

Atualizando modelo de dados - USER

```
#Dentro do método __init__ e na seed, iremos adicionar no final o “roles” dentro dos argumentos
    self.roles = roles #__init__
    roles = roles #seed
#Dentro da classe User, iremos adicionar um novo método chamado:
  def check_password(self, password):
    return bcrypt.checkpw(password.encode("utf-8"), self.password.encode("utf-8"))
#Dentro da classe UserSchema
  city = MA.Nested(city_share_schema)
  roles = MA.Nested(roles_share_schema)
 #Dentro da classe Meta, iremos adicionar esse acrescentar, no fields, os nomes: “city” e “roles”
```

Atualizando modelo de dados - USER

```
#Dentro da classe User, iremos adicionar esse bloco
  @staticmethod
  def encrypt_password(password):
      return bcrypt.hashpw(password, bcrypt.gensalt()).decode('utf-8')


  def save(self):
     DB.session.add(self)
     DB.session.commit()
```

Atualizando modelo de dados - TECHNOLOGY

```
#Dentro da classe Technology, iremos adicionar esse bloco
  @classmethod
  def seed(cls, name):
     tech = Technology(
        name = name
     )
     tech.save()


  def save(self):
     DB.session.add(self)
     DB.session.commit()
```

Atualizando modelo de dados - DEVELOPER

Nesta etapa, iremos excluir o arquivo developer_technology.py, pois iremos adicionar as informações dessa tabela satélite (join table) dentro da tabela de developer.

```
developer_technologies = DB.Table('developer_technologies',
                    DB.Column('developer_id', DB.Integer,
DB.ForeignKey('developers.id')),
                    DB.Column('technology_id', DB.Integer,
DB.ForeignKey('technologies.id'))
                    )
```

Atualizando modelo de dados - DEVELOPER

```
  #Atualizar os valores do modelo de dados de Developer e da def __init__
  id = DB.Column(DB.Integer, autoincrement=True, primary_key=True)
  months_experience = DB.Column(DB.Integer, nullable = True)
  accepted_remote_work = DB.Column(DB.Boolean, nullable = False, default = True)
  user_id = DB.Column(DB.Integer, DB.ForeignKey(User.id), nullable = False)
  technologies = DB.relationship('Technology', secondary=developer_technologies,
backref='developers')
  def __init__(self, months_experience, accepted_remote_work, user_id, technologies):
    self.months_experience = months_experience
    self.accepted_remote_work = accepted_remote_work
    self.user_id = user_id
    self.technologies = technologies
```

Atualizando modelo de dados - DEVELOPER

```
#Dentro da classe Developer, iremos adicionar esse bloco
  @classmethod
  def seed(cls, months_experience, accepted_remote_work, user_id, technologies):
    developer = Developer(
      months_experience = months_experience,
      accepted_remote_work = accepted_remote_work,
      user_id = user_id,
      technologies = technologies
    )
    developer.save()
  def save(self):
    DB.session.add(self)
    DB.session.commit()
```

Instalação das dependências

- **poetry add PyJWT**

JWT

O JSON Web Token é um padrão da Internet para a criação de dados com assinatura opcional e/ou criptografia cujo payload contém o JSON que afirma algum número de declarações.

Criando o módulo de autenticação

Agora, iremos criar uma pasta chama src/app/middlewares para utilizarmos nosso método de autenticação, dentro da pasta, um arquivo auth.py

Criando o módulo de autenticação

Primeiramente, adicionaremos uma nova configuração na nossa config, chamado SECRET_KEY para utilizarmos no nosso JWT como garantia de validação.

Feito o passo anterior, iremos criar uma pasta chamado src/app/middlewares para utilizarmos nosso método de autenticação, dentro da pasta, um arquivo auth.py

Criando o módulo de autenticação

```
from functools import wraps
from jwt import decode
from flask import request, jsonify, current_app
from src.app.models.user import User


def jwt_required(function_current):
    @wraps(function_current)
    def wrapper(*args, **kwargs):
      token = None

      if 'authorization' in request.headers:
        token = request.headers['authorization']
```

Criando o módulo de autenticação

```
        if not token:
            return jsonify({ "error": "Você não tem permissão para acessar essa rota" }),
403
        if not "Bearer" in token:
            return jsonify({ "error": "Você não tem permissão para acessar essa rota" }),
401
        try:
            token_pure = token.replace("Bearer ", "")
            decoded = decode(token_pure, current_app.config['SECRET_KEY'], 'HS256')
            current_user = User.query.get(decoded['user_id'])
        except:
            return jsonify({"error": "O token é inválido"}), 403
        return function_current(current_user=current_user, *args, **kwargs)
    return wrapper
```

DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!

SENAI

<LAB365>