REST API | Projeto 3

Curso completo de Python com certificado do MEC e reuniões com professores.

Conheça nosso curso completo de Python e Django que te da acesso à:

- Mais de 630 aulas
- Agendamento de reuniões com professores
- Reconhecido pelo MEC
- Análises de códigos
- Eventos entre alunos
- Exercícios automáticos
- E muito mais

Para quem participou da 4D4P terá um desconto especial, confira no link abaixo:

https://youtu.be/d_p7jAPDD8E

Acesse diretamente pelo Notion:

https://grizzly-amaranthus-f6a.notion.site/REST-API-Projeto-3-1946cf8ea89f80eea521d2c9f890a0e4?pvs=4

▼ Passos iniciais

Primeiro devemos criar o ambiente virtual:

```
# Criar
   # Linux
      python3 -m venv venv
   # Windows
      python -m venv venv
```

Após a criação do venv vamos ativa-lo:

```
#Ativar
   # Linux
      source venv/bin/activate
   # Windows
      venv\Scripts\Activate

# Caso algum comando retorne um erro de permissão execute o código e tente novamente:

Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
```

Agora vamos fazer a instalação do Django e as demais bibliotecas:

```
pip install django
pip install pillow
pip install django-ninja
```

Vamos criar o nosso projeto Django:

```
django-admin startproject core .
```

Rode o servidor para testar:

```
python manage.py runserver
```

Crie o app usuario:

```
python manage.py startapp treino
```

Ative o auto-save

INSTALE O APP!

Crie uma URL para API:

```
from .api import api

path('api/', api.urls)
```

Crie um router para api em core/api.py:

```
from ninja import NinjaAPI
from treino.api import treino_router
```

```
api = NinjaAPI()
api.add_router('', treino_router)
```

▼ Alunos

Primeiro passo, crie a model para armazenar os alunos:

```
faixa_choices = (
        ('B', 'Branca'),
        ('A', 'Azul'),
        ('R', 'Roxa'),
        ('M', 'Marrom'),
        ('P', 'Preta')
    )

class Alunos(models.Model):

    nome = models.CharField(max_length=255)
    email = models.EmailField(unique=True)
    faixa = models.CharField(max_length=1, choices=faixa_choices, default='B')
    data_nascimento = models.DateField(null=True, blank=True)

    def __str__(self):
        return self.nome
```

Crie o SCHEMA da model:

```
from ninja import ModelSchema
from .models import Alunos

class AlunosSchema(ModelSchema):
    class Meta:
        model = Alunos
        fields = ['nome', 'email', 'faixa', 'data_nascimento']
```

Desenvolva um endpoint para criar o aluno:

```
from ninja import Router
from ninja.errors import HttpError
from .schemas import AlunosSchema
from .models import Alunos

treino_router = Router()

@treino_router.post('/', response={200: AlunosSchema})
def criar_aluno(request, aluno_schema: AlunosSchema):
    nome = aluno_schema.dict()['nome']
    email = aluno_schema.dict()['email']
    faixa = aluno_schema.dict()['faixa']
    data_nascimento = aluno_schema.dict()['data_nascimento']

    if Alunos.objects.filter(email=email).exists():
        raise HttpError(400, "E-mail já cadastrado.")

    aluno = Alunos(nome=nome, email=email, faixa=faixa, data_nascimento=data_nascimento)
    aluno.save()

    return aluno
```

E o endpoint para listar todos os alunos:

```
@treino_router.get('/alunos/', response=List[AlunosSchema])
def listar_alunos(request):
    alunos = Alunos.objects.all()
    return alunos
```

▼ Aulas

Aqui, vamos criar uma tabela para armazenar as aulas realizadas por um aluno:

```
class AulasConcluidas(models.Model):
    aluno = models.ForeignKey(Alunos, on_delete=models.CASCADE)
    data = models.DateField(auto_now_add=True)
    faixa_atual = models.CharField(max_length=1, choices=faixa_choices)

    def __str__(self):
        return self.aluno.nome
```

Crie a função responsável pelo nível de progressão das faixas:

```
import math

order_belt = {'Branca': 0, 'Azul': 1, 'Roxa': 2, 'Marrom': 3, 'Preta': 4 }

def calculate_lessons_to_upgrade(n):
    d = 1.47
    k = 30 / math.log(d)

    aulas = k * math.log(n + d)

    return round(aulas)
```

Crie o Schema:

```
class ProgressoAlunoSchema(Schema):
    email: str
    nome: str
    faixa: str
    total_aulas: int
    aulas_necessarias_para_proxima_faixa: int
```

Construa a VIEW:

```
@treino_router.get('/progresso_aluno/', response={200: ProgressoAlunoSchema})
def progresso_aluno(request, email_aluno: str):
    aluno = Alunos.objects.get(email=email_aluno)

    total_aulas_concluidas = AulasConcluidas.objects.filter(aluno=aluno).count()

    faixa_atual = aluno.get_faixa_display()

    n = order_belt.get(faixa_atual, 0)

    total_aulas_proxima_faixa = calcula_aulas_necessarios_proximo_nivel(n)

    total_aulas_concluidas_faixa = AulasConcluidas.objects.filter(aluno=aluno, faixa_atual=aluno.faixa).count()

    aulas_faltantes = max(total_aulas_proxima_faixa - total_aulas_concluidas_faixa, 0)
```

E para finalizar o projeto, desenvolva a funcionalidade para atualizar os dados de um aluno:

```
@treino_router.put("/alunos/{aluno_id}", response=AlunosSchema)
def update_aluno(request, aluno_id: int, aluno_data: AlunosSchema):
    aluno = get_object_or_404(Alunos, id=aluno_id)

    idade = date.today() - aluno.data_nascimento

    if int(idade.days/365) < 18 and aluno_data.dict()['faixa'] in ('A', 'R', 'M', 'P'):
        raise HttpError(400, "O aluno é menor de idade e não pode ser graduado para essa faixa.")

    #exclude_unset=True
    for attr, value in aluno_data.dict().items():
        if value:
            setattr(aluno, attr, value)

    aluno.save()
    return aluno
```