Projeto 3 - Encurtador de links

Desafio Back-end API

O desafio de hoje, será construir uma REST API com Python e Django para um encurtador de link.

Sua API deve:

- Cadastrar novos links
- Atualizar links
- Redirecionamento automático
- Estatísticas
- QRCode

Os links podem possuir um tempo de expiração e quantidade máxima de cliques únicos.

As estatísticas devem exibir o total de cliques únicos e totais de um link.

O usuário pode desativar o link a qualquer momento.

▼ Setup

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 django-ninja
```

Vamos criar o nosso projeto Django:

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

Rode o servidor para testar:

```
python manage.py runserver
```

Crie um APP usuários:

```
    python3 manage.py shortener
```

▼ Configurações iniciais

Em core/urls.py Crie uma URL para API:

```
from django.contrib import admin
from django.urls import path
from .api import api

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', api.urls)
]
```

Instancie o NinjaAPI() em api.py

```
from ninja import NinjaAPI
from shortener.api import shortener_router

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

Em shortener/api.py:

```

shortener_router = Router()
```

Crie as tabelas necessárias:

```
from django.db import models
from secrets import token_urlsafe
from django.utils import timezone

class Links(models.Model):
    redirect_link = models.URLField()
    token = models.CharField(max_length=12, unique=True, null=True, blank=True)
    create_at = models.DateTimeField(auto_now_add=True)
    expiration_time = models.DurationField(null=True, blank=True)
    max_uniques_cliques = models.PositiveIntegerField(null=True, blank=True)
    active = models.BooleanField(default=True)
```

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

    def save(self, *args, **kwargs):
        if not self.token:
            while True:
                self.token = token_urlsafe(6)
                if not Links.objects.filter(token=self.token).exists():
                    break

        super().save(*args, **kwargs)

    def expired(self):
        return True if timezone.now() > (self.create_at + self.expiration_time) else False

class Clicks(models.Model):
    link = models.ForeignKey(Links, on_delete=models.CASCADE)
    ip = models.GenericIPAddressField()
    create_at = models.DateTimeField(auto_now_add=True)
```

Execute as migrações!

em schemas.py crie o LinkSchema:

```
from ninja import ModelSchema, Schema
from .models import Links
from datetime import timedelta

class LinkSchema(ModelSchema):
    expiration_time: int

    class Meta:
        model = Links
        fields = ['redirect_link', 'token', 'expiration_time', 'max_uniques_cliques']

    def to_model_data(self):
        return {
            "redirect_link": self.redirect_link,
            "token": self.token,
            "expiration_time": timedelta(minutes=self.expiration_time),
            "max_uniques_cliques": self.max_uniques_cliques,
        }

    @classmethod
    def from_model(cls, instance):
        return cls(
            redirect_link=instance.redirect_link,
            token=instance.token,
            expiration_time=int(instance.expiration_time.total_seconds() // 60),
            max_uniques_cliques=instance.max_uniques_cliques,
        )
```

API

Crie o endpoint para criar um novo link:

```
@shortener_router.post('/', response={200: LinkSchema, 409: dict})
def create_shortener(request, link_schema: LinkSchema):
    '''redirect_link = link_schema.dict()['redirect_link']
    token = link_schema.dict()['token']
    expiration_time = link_schema.dict()['expiration_time']
    max_uniques_cliques = link_schema.dict()['max_uniques_cliques']

    link = Links(
        redirect_link=redirect_link,
        token=token,
        expiration_time=expiration_time,
        max_uniques_cliques=max_uniques_cliques
    )
    link.save()'''

    token = link_schema.dict()['token']
    if token and Links.objects.filter(token=token):
        return 409, {'error': 'Token já existe, use outro'}

    link = Links(**link_schema.to_model_data())
    link.save()

    return 200, LinkSchema.from_model(link)
```

Crie o Schema para atualizar um link:

```
class UpdateLinkSchema(Schema):
    redirect_link: str = None
    token: str = None
    max_uniques_cliques: int = None
    active: bool = None
```

Adicione o endpoint:

```
@shortener_router.put("/{link_id}/", response={200: UpdateLinkSchema, 409: dict})
def update_link(request, link_id: int, link_schema: UpdateLinkSchema):
    link = get_object_or_404(Links, id=link_id)

    token = link_schema.dict()['token']
    if token and Links.objects.filter(token=token):
        return 409, {'error': 'Token já existe, use outro'}

    for field, value in link_schema.dict().items():
        if value is not None:
            setattr(link, field, value)

    link.save()
    return 200, link
```

Para as estatísticas crie um novo endpoint:

```
@shortener_router.get("statistics/{link_id}/", response={200: dict})
def statistics(request, link_id: int):
```

```
    link = get_object_or_404(Links, id=link_id)
    uniques_clicks = Clicks.objects.filter(link=link).values('ip').distinct().count()
    total_clicks = Clicks.objects.filter(link=link).values('ip').count()

    return 200, {'uniques_clicks': uniques_clicks, 'total_clicks': total_clicks}
```

▼ QRCode

Para gerar o QRCode instale a lib:

```
pip install qrcode
```

Desenvolva a lógica para geração:

```
import qrcode
from io import BytesIO
import base64

def get_api_url(request, token):
    scheme = request.scheme
    host = request.get_host()
    return f"{scheme}://{host}/api/{token}"

@shortener_router.get("qrcode/{link_id}/", response={200: dict})
def get_qrcode(request, link_id: int):
    link = get_object_or_404(Links, id=link_id)
    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_L,
        box_size=10,
        border=4,
    )
    print(get_api_url(request, link.token))
    qr.add_data(get_api_url(request, link.token))
    qr.make(fit=True)

    content = BytesIO()
    img = qr.make_image(fill_color="black", back_color="white")
    img.save(content)
    data = base64.b64encode(content.getvalue()).decode('UTF-8')
    return 200, {'content_image': data}
```