Orientações para utilização da API USERS

Pelo exercício foi solicitada a construção de uma API que gera um token quando solicitado e a criação de dois endpoints um **USER** e o outro **ADMIN** usando a biblioteca de geração de token **JWT** sendo que os dois endpoints são acessados após a autenticação sendo o endpoint **USER** visualizado por usuários normais após autenticação e o endpoint **ADMIN** sendo visualizado apenas por usuários administradores. Procurei mostrar boas práticas de programação separando as funções por classes e aplicações. Para isso eu utilizei o framework **Django e o DRF Django Rest Framework,** além é claro da biblioteca **JWT e o banco sqlite**. Criei o projeto **apiUser** e as aplicações **USER** e **PRODUCT.** Criei uma api também com o **CRUD** para produtos também funcional tendo o **POST, GET ,PUT e DELETE** em que coloquei acessível a partir do endpoint **ADMIN.** Além disso não foi pedido mas achei por bem criar um endpoint para criação de usuário usando o token com possibilidade de envio da ativação da conta por e-mail. O código está comentado pois por questões de segurança eu tirei as minha informações pessoas para autenticação para o serviço de email. Pois usei a minha conta do gmail com o token gerado para acesso à minha conta de e-mail. Mas se colocadas as informações necessárias a plicação também enviará o e-mail com o link para ativação. Essa é uma prática bastante usada e mais segura para acesso principalmente em ecommerces . Segue abaixo partes do código e telas que utilizei. Além disso no final tem o link do projeto totalmente funcional no GIT HUB.

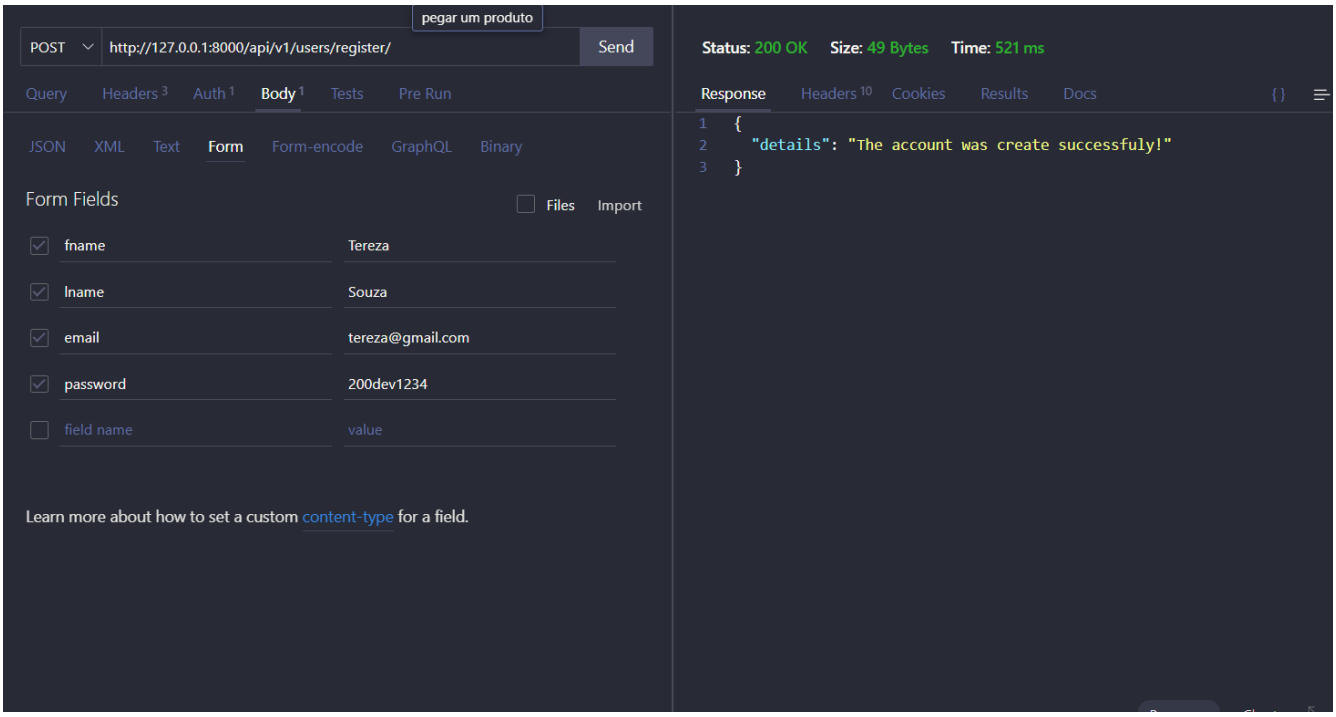
OBS.: Para facilitar criei o usuário **200dev** e senha **200dev1234 (tudo minúsculo)**

Arquivo pasta user/view.py

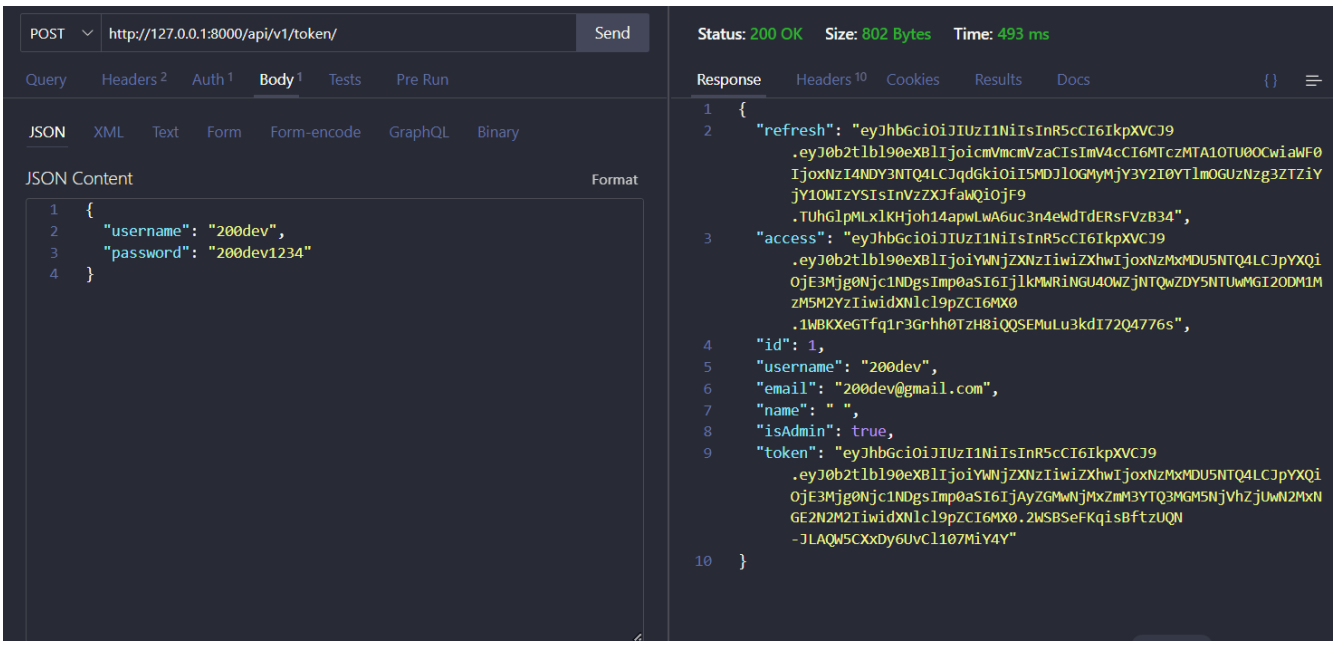
#Endpont que cria o o usuário

```
@api_view(['POST'])
def registerUser(request):
    data=request.data
    try:
        user=
User.objects.create(first_name=data['fname'],last_name=data['lname'],username=data['email'],email=data['email'],password=make_password(data['passwor
d']),is_active=True)

        # Para aumentar a segurança é possível ativar a conta após o recebimento do e-mail com o link para ativação
        # deixei comentado, mas está totalmente funcional, também por questões de segurança eu não deixei as
        # minhas informações de chave de email e conta estando o código pronto caso seja colocada a chave e configurações corretas
        # No settings deixei comentado com uma chave fake caso seja colocada a chave correta só é descomentar o código abaixo e  não esquecer
        # de apagar a linha que está descomentada e ativando a conta

    # user=
User.objects.create(first_name=data['fname'],last_name=data['lname'],username=data['email'],email=data['email'],password=make_password(data['passwor
d']),is_active=False)
        # #generate token for sending mail
        # email_subject="Activate Your Account"
        # message=render_to_string(
        #     "activate.html",
        #     {
        #     'user':user,
        #     'domain':'127.0.0.1:8000',
        #     'uid':urlsafe_base64_encode(force_bytes(user.pk)),
        #     'token':generate_token.make_token(user)
        #     }

        # )
        # print(message)
        # email_message=EmailMessage(email_subject,message,settings.EMAIL_HOST_USER,[data['email']])
        # EmailThread(email_message).start()

        # message={'details':'Activate your account please check the link in email for account activation'}
        message={'details': "The account was create successfuly!"}
        return Response(message)

    except Exception as e:
        message={'details':'User with this email already exists or something went wrong'}

        return Response(message)
```

Tela que usei para criar o usuário com o respectivo endpoint
OBS.: Para facilitar criei o usuário **200dev** e senha **200dev1234 (tudo minúsculo) ele é usuário administrador também no Django quando acessar http://127.0.0.1:8000/admin**



Usei o Thunder Client no Vscode para acessar os endpointes segue abaixo a tela que utilizei



Parte do código para geração do token e respectiva validação

Arquivo pasta user/view.py

```
class MyTokenObtainPairSerializer(TokenObtainPairSerializer):
    def validate(self, attrs):
        data = super().validate(attrs)
        serializer=UserSerializerWithToken(self.user).data
        for k,v in serializer.items():
            data[k]=v
        return data


class MyTokenObtainPairView(TokenObtainPairView):
    serializer_class=MyTokenObtainPairSerializer
```

Parte em que os endpoints só são acessíveis após autenticação

```
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def getUsers(request):
    users=User.objects.all()
    serializer=UserSerializer(users,many=True)
    return Response(serializer.data)

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def getUser(request,pk):
    user=User.objects.get(id=pk)
    serializer=UserSerializer(user,many=False)
    return Response(serializer.data)
```

Arquivo pasta product/view.py

#Endpoints acessados a partir do endpoint /admin que só usuários administradores podem acessar

```
class ProductView(APIView):

    @api_view(['POST'])
    #@permission_classes([IsAdminUser ])
    def createProduct(request):
        serializer = ProductsSerializer(data=request.data, many=False)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    @api_view(['GET'])
    @permission_classes([IsAdminUser])
    def getProducts(request):
        products=Products.objects.all()
        serializer=ProductsSerializer(products,many=True)
        return Response(serializer.data)


    @api_view(['GET'])
    @permission_classes([IsAdminUser])
    def getProduct(request,pk):
        product=Products.objects.get(_id=pk)
        serializer=ProductsSerializer(product,many=False)
        return Response(serializer.data)

    @api_view(['PUT'])
    @permission_classes([IsAdminUser])
    def updateProduct(request, pk):
        product = Products.objects.get(_id=pk)
        serializer = ProductsSerializer(product, data=request.data,many=False)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=400)


    @api_view(['DELETE'])
    @permission_classes([IsAdminUser ])
    def deleteProduct(request, pk):
        try:
            product = Products.objects.get(_id=pk)
            product.delete()
            return Response(status=status.HTTP_204_NO_CONTENT)
        except Products.DoesNotExist:
            return Response(status=status.HTTP_404_NOT_FOUND)
```

Arquvio com os endpoints tanto user quanto admin

ARQUIVO apiUser/urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('user.urls')),
    path('api/v1/admin/', include('product.urls')),
]

urlpatterns+=static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Arquivo /user/urls.py

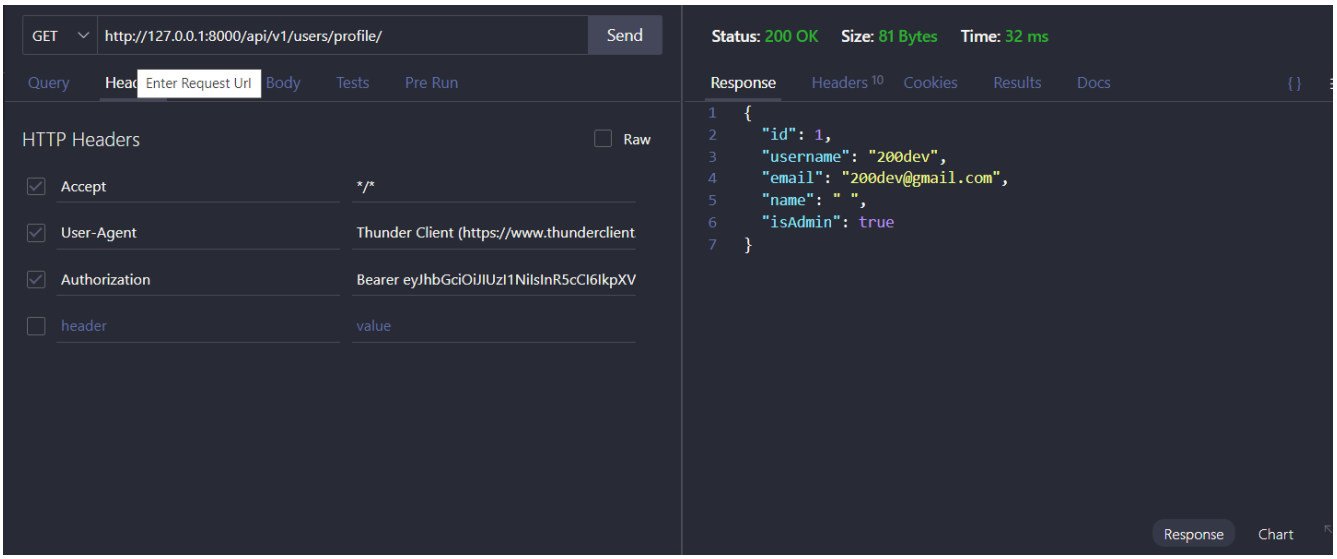
arquivo com os endpoints do user

```
from . import views
from django.urls import path
from rest_framework_simplejwt.views import ( # type: ignore
    TokenObtainPairView
)

urlpatterns = [
    path('', views.getRoutes,name="getRoutes"),
    path('token/', views.MyTokenObtainPairView.as_view(), name='token_obtain_pair'), #rota que cria o token
    path('users/profile/',views.getUserProfiles,name="getUserProfiles"),  #pega o token do usuário criado
    path('users/register/',views.registerUser,name="register"),  #cria uma conta de usuário
    path('users/',views.getUsers,name="getUsers"), # lista os usuarios
    path('user/<str:pk>',views.getUser,name="getUser"), # lista um usuário específico
     path('activate/<uidb64>/<token>',views.ActivateAccountView.as_view(),name='activate'),# gera un token, cria um link de ativação e envia por email
]
```

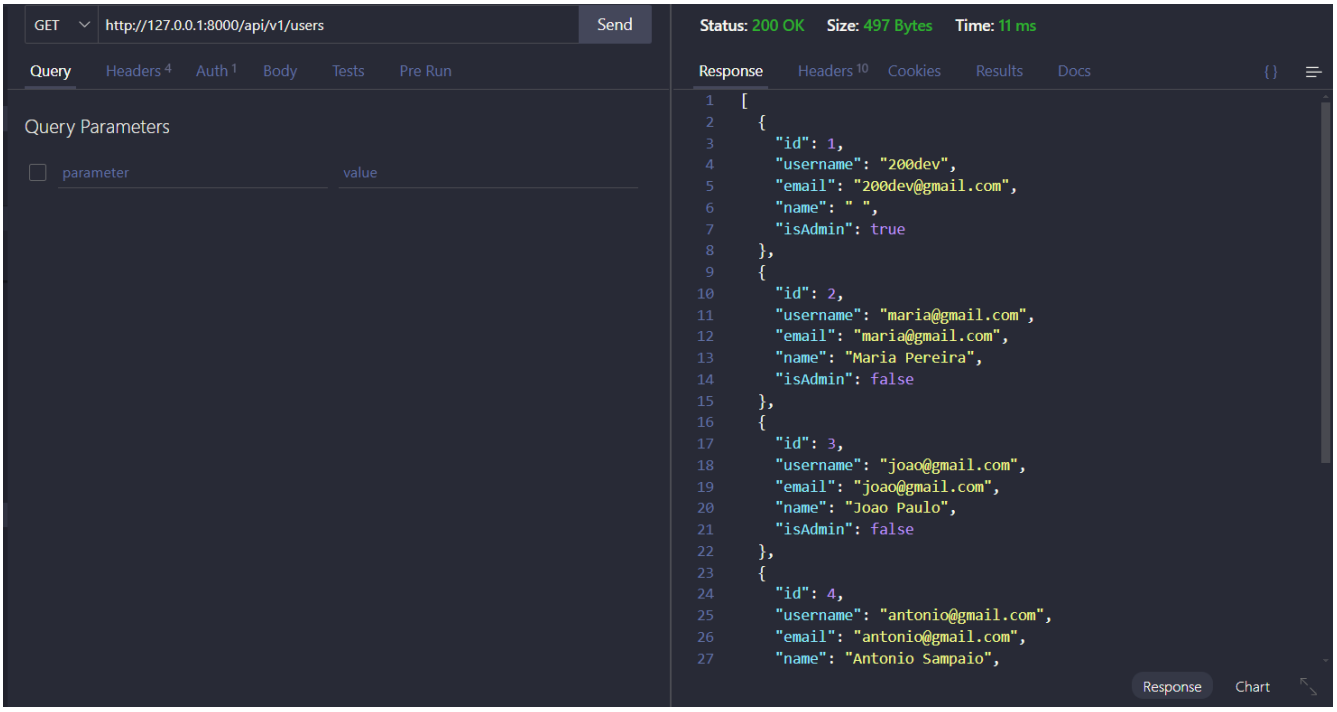
Arquivo com as urls do endpoind admin
Arquivo product\urls.py

```
from . import views
from django.urls import path

urlpatterns = [
    path('', views.getRoutes,name="getRoutes"),
    path('products/',views.ProductView.getProducts,name="getProducts"),  #cria o produto
    path('products/',views.ProductView.createProduct,name="createProduct"),  #pega o produto
    path('product/<str:pk>',views.ProductView.getProduct,name="getProduct"),
    path('product/update/<str:pk>',views.ProductView.updateProduct,name="updateProduct"),  #cria uma conta de usuário
    path('product/delete/<str:pk>',views.ProductView.deleteProduct,name="deleteProduct"), # lista os usuarios

]
```

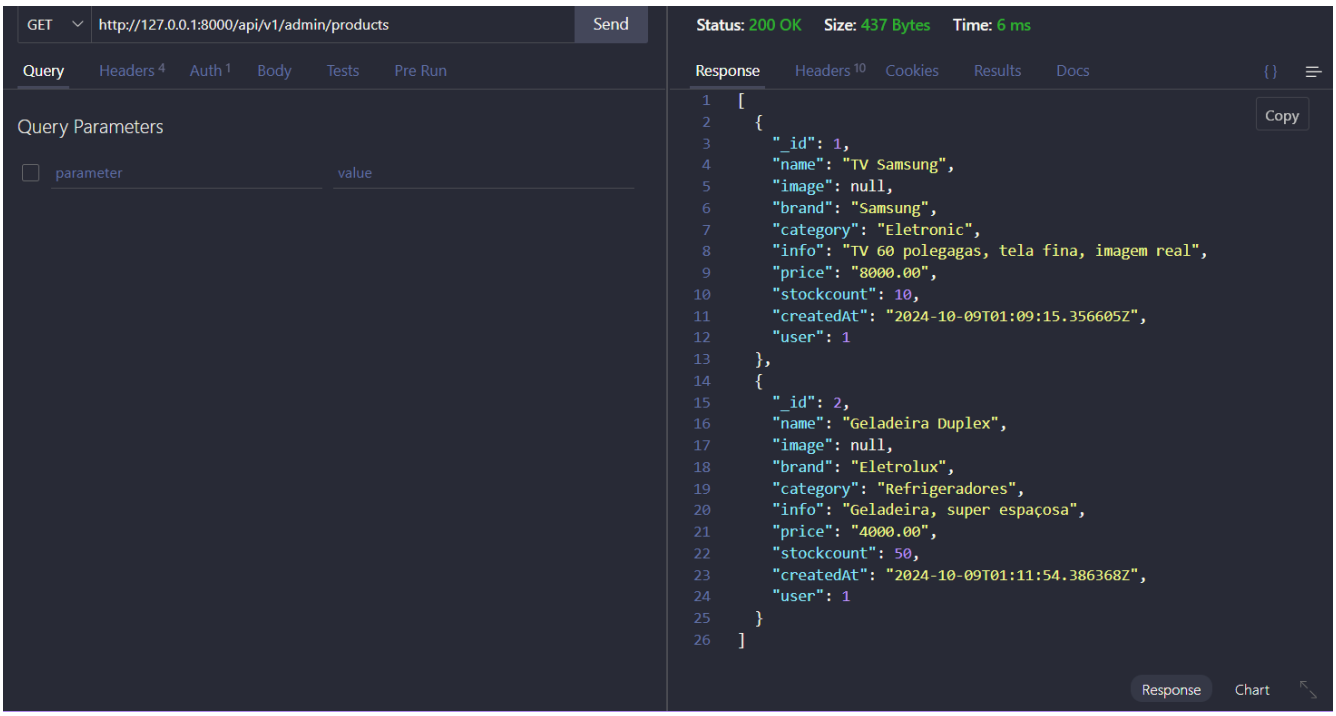
Tela que carrega o usuário passando o token



Tela que traz trodos os usuários



Tela com o carregando os endpoints do Endpoint ADMIN



Para configurar o ambiente instalar todos os pacotes do arquivo requirements.txt:

```
asgiref==3.8.1
Django==5.1.2
django-cors-headers==4.4.0
djangorestframework==3.15.2
djangorestframework-simplejwt==5.3.1
pillow==10.4.0
PyJWT==2.9.0
six==1.16.0
sqlparse==0.5.1
tzdata==2024.2
```

Preparação do ambiente:

python3 -m venv venv

pip install django
pip install djangoestframework

django-admin startproject apiUser
entra na pasta apiUser
cd apiUser
python .\manage.py startapp user

python .\manage.py startapp product