Resumo - Por José Alfredo F. Costa - Novembro de 2024

**Sistema de gestão de consultas médicas utilizando Django e Python (início – informações básicas)**

Vamos organizar o projeto desde o início, seguindo uma estrutura que cubra desde o ambiente inicial até as funcionalidades principais, incluindo autenticação JWT para segurança e operações CRUD (Criar, Ler, Atualizar, Deletar) para pacientes, médicos e consultas. Esse sistema permitirá o cadastro e a gestão de pacientes, médicos e agendamento de consultas, com data e horário.

**Passo 1: Configurar o Ambiente de Desenvolvimento**

Primeiro, vamos criar o ambiente virtual e instalar as dependências necessárias para o Django e o Django REST Framework (DRF) para a criação de APIs RESTful.

# Criar o ambiente virtual

python3 -m venv venv

source venv/bin/activate # Para Linux/Mac

venv\Scripts\activate # Para Windows

# Instalar Django e Django REST Framework

pip install django djangorestframework djangorestframework-simplejwt

Agora, vamos criar o projeto Django e o aplicativo para o sistema de consultas médicas.

# Criar o projeto Django

django-admin startproject sistema\_consultas

cd sistema\_consultas

# Criar o aplicativo para o sistema de consultas

python manage.py startapp consultas

**Passo 2: Configurar o Django e o Django REST Framework (DRF)**

No arquivo settings.py, configure as aplicações do Django e o DRF, incluindo as configurações para autenticação JWT.

python

# settings.py

INSTALLED\_APPS = [

# Apps padrões do Django

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

# Apps do projeto

'consultas',

# Django REST Framework e JWT

'rest\_framework',

'rest\_framework\_simplejwt',

]

# Configurações do Django REST Framework

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': (

'rest\_framework\_simplejwt.authentication.JWTAuthentication',

),

}

Isso configura o DRF para usar JWT para autenticação, o que será útil para proteger as rotas do sistema.

**Passo 3: Configurar URLs para Autenticação JWT e APIs**

No arquivo urls.py do projeto principal, vamos adicionar as rotas para a obtenção de tokens JWT.

python

# sistema\_consultas/urls.py

from django.contrib import admin

from django.urls import path, include

from rest\_framework\_simplejwt.views import TokenObtainPairView, TokenRefreshView

urlpatterns = [

path('admin/', admin.site.urls),

path('api/token/', TokenObtainPairView.as\_view(), name='token\_obtain\_pair'),

path('api/token/refresh/', TokenRefreshView.as\_view(), name='token\_refresh'),

path('consultas/', include('consultas.urls')), # Incluir as URLs do app "consultas"

]

Com essas rotas configuradas, o sistema estará pronto para autenticar usuários utilizando JWT.

**Passo 4: Criar Modelos para Pacientes, Médicos e Consultas**

Agora vamos definir os modelos para as três principais entidades do sistema: Paciente, Medico e Consulta.

python

# consultas/models.py

from django.db import models

class Paciente(models.Model):

nome = models.CharField(max\_length=255)

data\_nascimento = models.DateField()

email = models.EmailField(unique=True)

def \_\_str\_\_(self):

return self.nome

class Medico(models.Model):

nome = models.CharField(max\_length=255)

crm = models.CharField(max\_length=20, unique=True)

especialidade = models.CharField(max\_length=100)

def \_\_str\_\_(self):

return f"{self.nome} - {self.especialidade}"

class Consulta(models.Model):

paciente = models.ForeignKey(Paciente, on\_delete=models.CASCADE)

medico = models.ForeignKey(Medico, on\_delete=models.CASCADE)

data = models.DateField()

horario = models.TimeField()

def \_\_str\_\_(self):

return f"Consulta de {self.paciente.nome} com {self.medico.nome} em {self.data} às {self.horario}"

Após definir os modelos, execute as migrações para criar as tabelas no banco de dados.

python manage.py makemigrations

python manage.py migrate

**Passo 5: Criar Serializers para os Modelos**

Vamos criar serializers para converter os dados dos modelos para JSON, permitindo que eles sejam utilizados nas APIs.

python

# consultas/serializers.py

from rest\_framework import serializers

from .models import Paciente, Medico, Consulta

class PacienteSerializer(serializers.ModelSerializer):

class Meta:

model = Paciente

fields = ['id', 'nome', 'data\_nascimento', 'email']

class MedicoSerializer(serializers.ModelSerializer):

class Meta:

model = Medico

fields = ['id', 'nome', 'crm', 'especialidade']

class ConsultaSerializer(serializers.ModelSerializer):

class Meta:

model = Consulta

fields = ['id', 'paciente', 'medico', 'data', 'horario']

**Passo 6: Criar Views para as Funcionalidades de API**

Em views.py, vamos criar as views que gerenciarão as operações CRUD para pacientes, médicos e consultas.

python

# consultas/views.py

from rest\_framework import viewsets

from rest\_framework.permissions import IsAuthenticated

from .models import Paciente, Medico, Consulta

from .serializers import PacienteSerializer, MedicoSerializer, ConsultaSerializer

class PacienteViewSet(viewsets.ModelViewSet):

queryset = Paciente.objects.all()

serializer\_class = PacienteSerializer

permission\_classes = [IsAuthenticated]

class MedicoViewSet(viewsets.ModelViewSet):

queryset = Medico.objects.all()

serializer\_class = MedicoSerializer

permission\_classes = [IsAuthenticated]

class ConsultaViewSet(viewsets.ModelViewSet):

queryset = Consulta.objects.all()

serializer\_class = ConsultaSerializer

permission\_classes = [IsAuthenticated]

Essas views permitem criar, ler, atualizar e deletar registros para pacientes, médicos e consultas.

**Passo 7: Configurar URLs do App consultas**

No arquivo urls.py dentro do app consultas, vamos registrar as rotas para os viewsets criados.

python

# consultas/urls.py

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from . import views

router = DefaultRouter()

router.register(r'pacientes', views.PacienteViewSet)

router.register(r'medicos', views.MedicoViewSet)

router.register(r'consultas', views.ConsultaViewSet)

urlpatterns = [

path('', include(router.urls)),

]

**Passo 8: Testar a Autenticação com JWT e as Rotas de API**

Agora, o sistema de gestão de consultas médicas está configurado. Para testar, siga o fluxo de autenticação e utilização da API:

1. **Obter o Token JWT**: Faça uma solicitação POST para /api/token/ com as credenciais de login.

json

{

"username": "admin",

"password": "senha"

}

Isso retorna um token JWT que pode ser usado para autenticação.

1. **Utilizar o Token JWT nas Requisições**: Envie o token JWT no cabeçalho de autorização para acessar as rotas protegidas (pacientes, médicos, consultas).

http

Authorization: Bearer <token\_jwt>

Com isso, você terá acesso às rotas protegidas, permitindo realizar operações de CRUD para pacientes, médicos e consultas.

**Vantagens do Uso do Django para o Sistema de Gestão de Consultas Médicas**

1. **Painel Administrativo**: O Django oferece um painel administrativo pronto para uso, facilitando a gestão de pacientes, médicos e consultas sem precisar criar interfaces adicionais para essas operações.
2. **Segurança e Autenticação JWT**: Com o Django REST Framework e simplejwt, podemos configurar um sistema robusto de autenticação para proteger as rotas da API.
3. **Facilidade de Configuração e Modulação**: O Django organiza o projeto em "apps", facilitando a manutenção e a escalabilidade. No caso do sistema de consultas médicas, isso é útil para separar as responsabilidades de cada parte do sistema (pacientes, médicos e consultas).
4. **Controle de Acesso**: Com o DRF, é possível definir permissões específicas para cada rota, permitindo que apenas usuários autenticados possam acessar ou modificar os dados.
5. **Comunidade e Documentação**: O Django possui uma ampla documentação e uma comunidade ativa, facilitando o desenvolvimento e a implementação de boas práticas no projeto.

Esse sistema, utilizando Django e o DRF com autenticação JWT, oferece uma solução robusta e escalável para a gestão de consultas médicas, garantindo segurança e facilidade de manutenção.