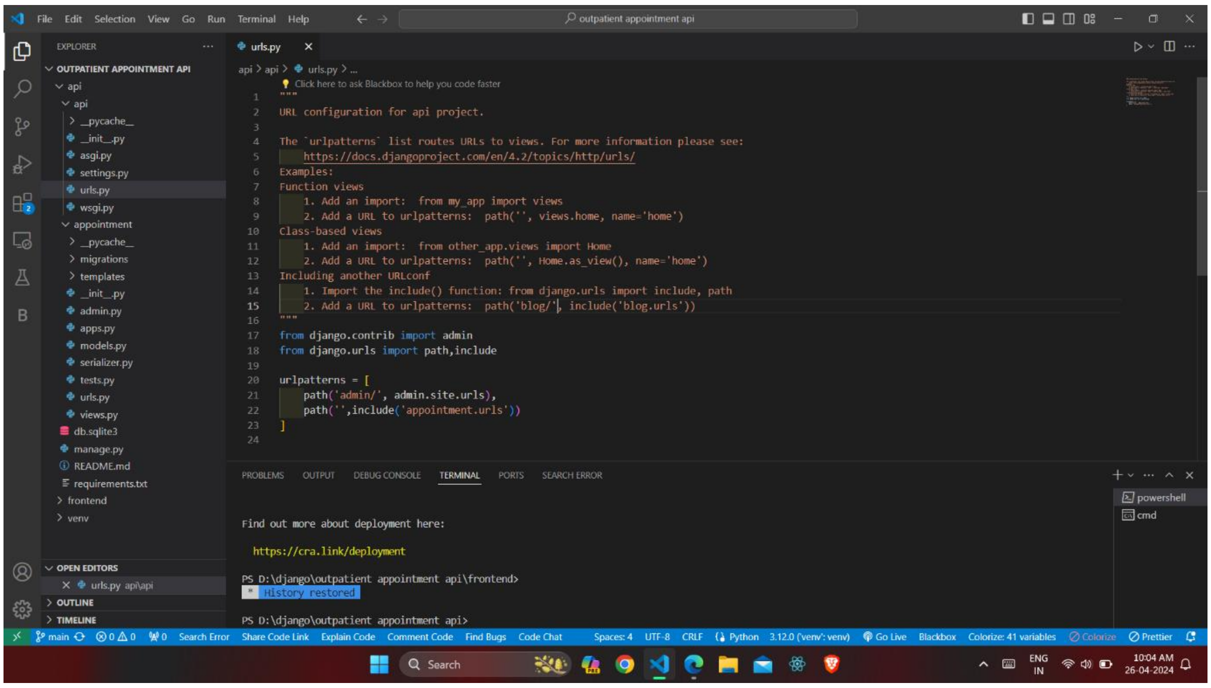
OUTPATIENT APPOINTMENT API

Project Title and Purpose:

- The project is titled "**OUTPATIENT APPOINTMENT API**."
- It appears to be related to managing outpatient appointments.

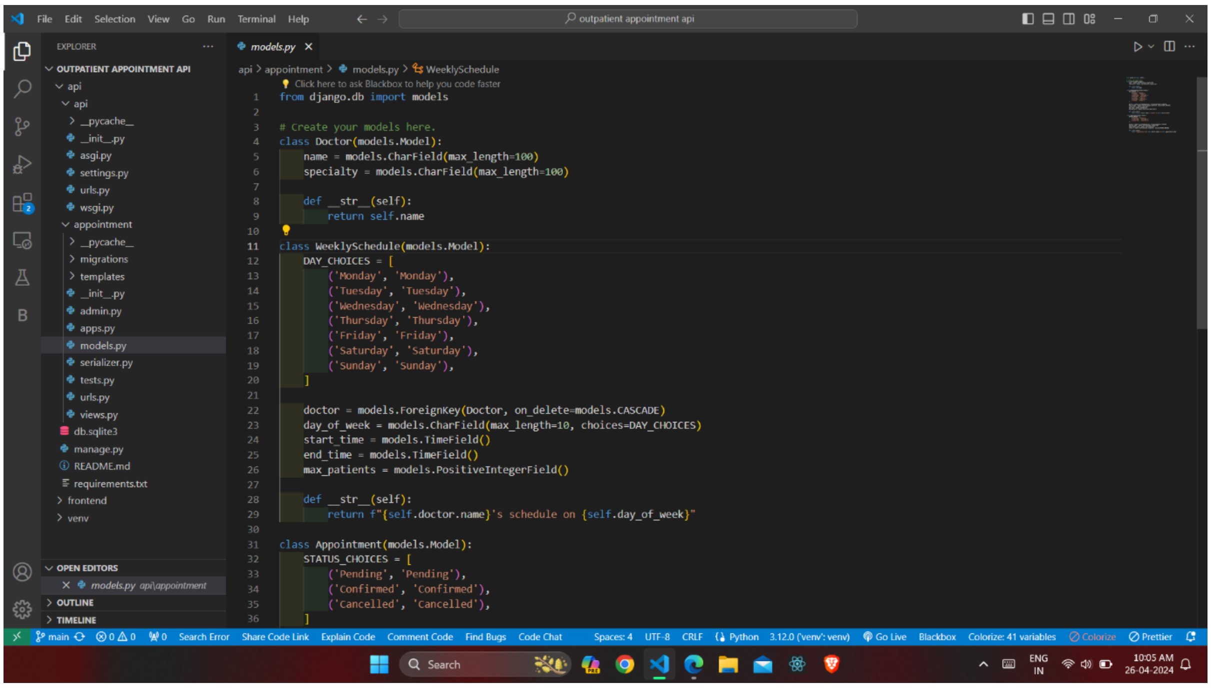


```python
"""
URL configuration for api project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path,include

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

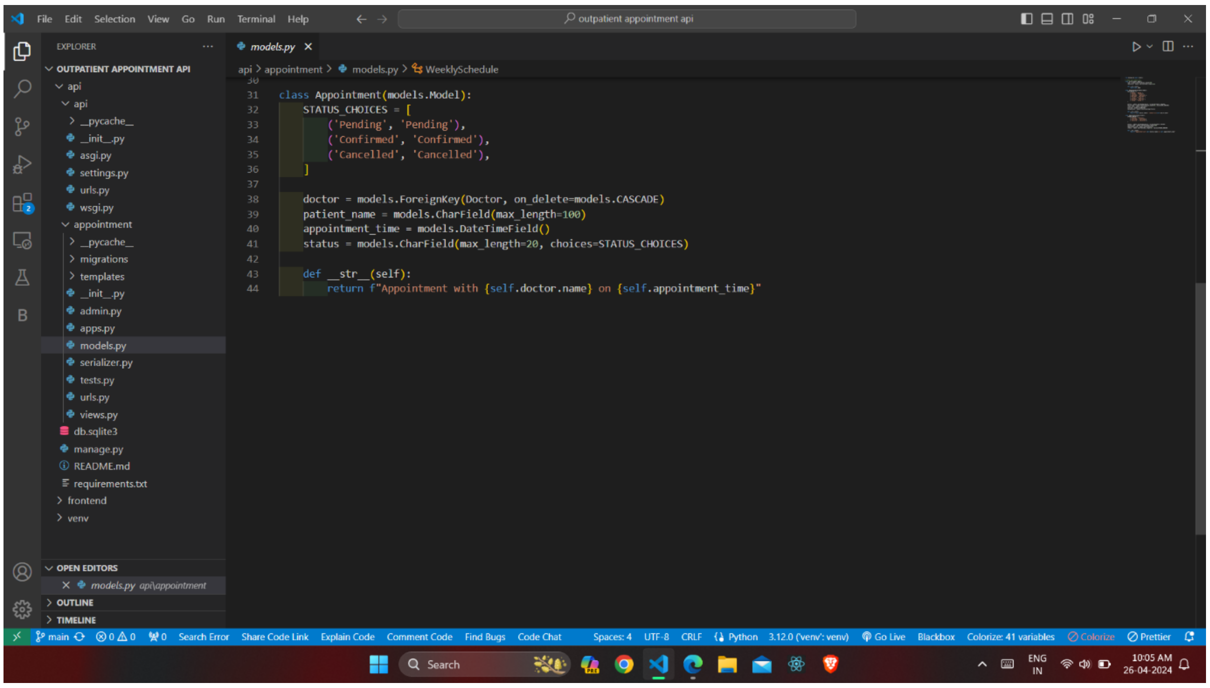
```
Find out more about deployment here:

  https://cra.link/deployment

PS D:\django\outpatient appointment api\frontend>
 *  History restored

PS D:\django\outpatient appointment api>
```

1. **URL Configuration (urls.py)**:
   - The file `urls.py` is used for configuring URL patterns in a Django web application.
   - The URL patterns map specific URLs to corresponding views (functions or classes).
   - The `urlpatterns` list contains routes for different parts of the application.
2. **Examples of URL Patterns**:
   - Example 1:
     - URL: `/admin/`
     - Corresponding view: Django admin interface
   - Example 2:
     - URL: `/`
     - Corresponding view: Includes the app named "appointment" (defined in `appointment.urls`)

OUTPATIENT APPOINTMENT API



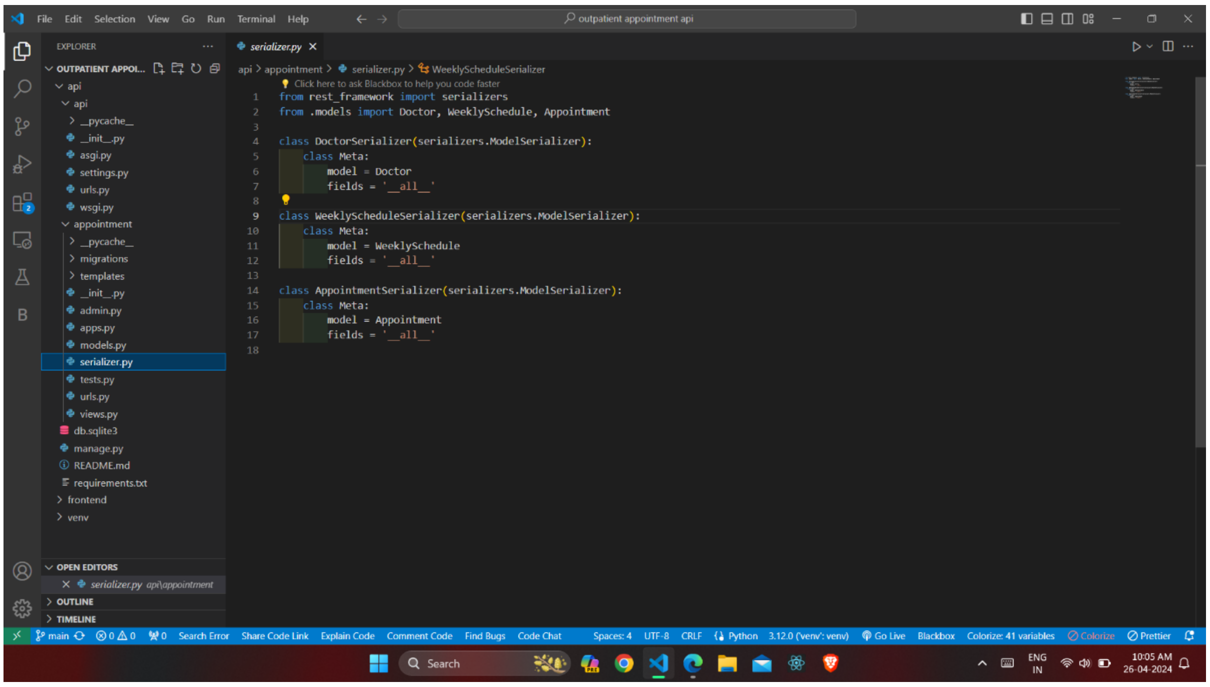
```python
from django.db import models

# Create your models here.
class Doctor(models.Model):
    name = models.CharField(max_length=100)
    specialty = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class WeeklySchedule(models.Model):
    DAY_CHOICES = [
        ('Monday', 'Monday'),
        ('Tuesday', 'Tuesday'),
        ('Wednesday', 'Wednesday'),
        ('Thursday', 'Thursday'),
        ('Friday', 'Friday'),
        ('Saturday', 'Saturday'),
        ('Sunday', 'Sunday'),
    ]

    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE)
    day_of_week = models.CharField(max_length=10, choices=DAY_CHOICES)
    start_time = models.TimeField()
    end_time = models.TimeField()
    max_patients = models.PositiveIntegerField()

    def __str__(self):
        return f"{self.doctor.name}'s schedule on {self.day_of_week}"

class Appointment(models.Model):
    STATUS_CHOICES = [
        ('Pending', 'Pending'),
        ('Confirmed', 'Confirmed'),
        ('Cancelled', 'Cancelled'),
    ]
```



```python
class Appointment(models.Model):
    STATUS_CHOICES = [
        ('Pending', 'Pending'),
        ('Confirmed', 'Confirmed'),
        ('Cancelled', 'Cancelled'),
    ]

    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE)
    patient_name = models.CharField(max_length=100)
    appointment_time = models.DateTimeField()
    status = models.CharField(max_length=20, choices=STATUS_CHOICES)

    def __str__(self):
        return f"Appointment with {self.doctor.name} on {self.appointment_time}"
```

1. **Models.py**:
    - This is the Object Relation Mapping (ORM) file.
    - This file contains the structure of the database.
    - Classes like Doctor, WeeklySchedule and, Appointment are the tables that explaining the structure of the tables.

OUTPATIENT APPOINTMENT API



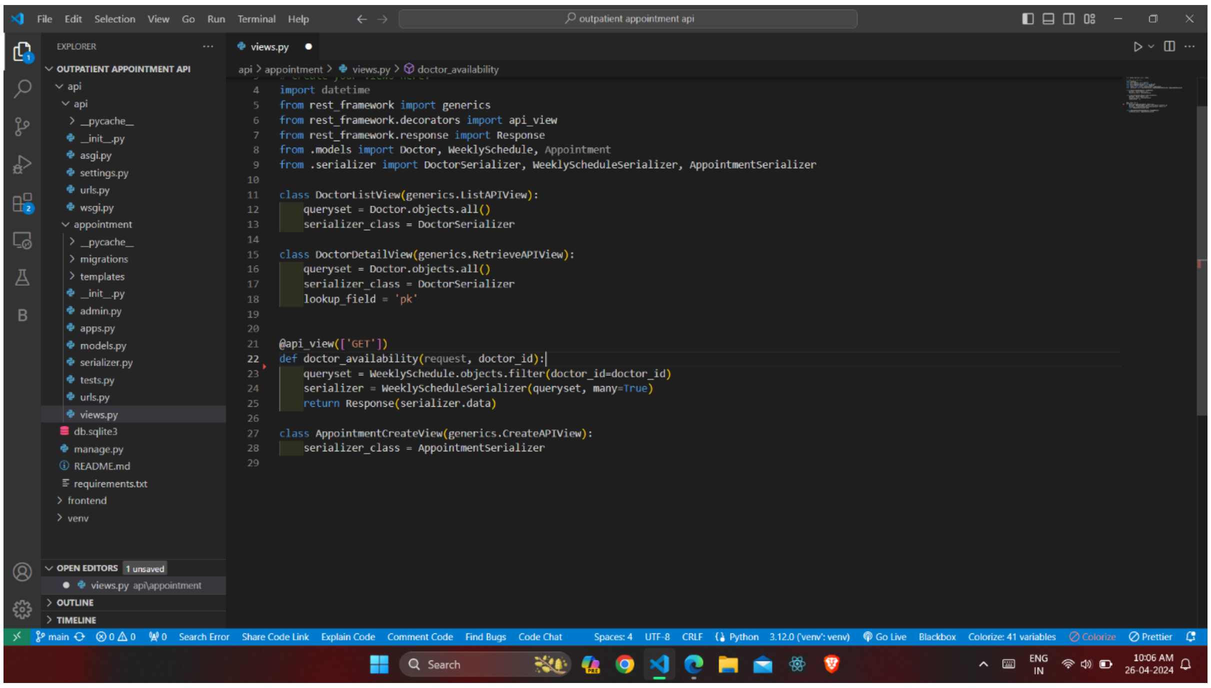
```python
from rest_framework import serializers
from .models import Doctor, WeeklySchedule, Appointment

class DoctorSerializer(serializers.ModelSerializer):
    class Meta:
        model = Doctor
        fields = '__all__'

class WeeklyScheduleSerializer(serializers.ModelSerializer):
    class Meta:
        model = WeeklySchedule
        fields = '__all__'

class AppointmentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Appointment
        fields = '__all__'
```

Serializer.py

1. **DoctorSerializer**:
   - This serializer class is for the `Doctor` model. It includes all fields of the model in the serialization process.
2. **WeeklyScheduleSerializer**:
   - This serializer class handles the `WeeklySchedule` model, serializing all its fields.
3. **AppointmentSerializer**:
   - This serializer class is for the `Appointment` model, including all fields in the serialization.

Each `Meta` class within the serializers specifies the model it serializes and the fields to include. The `'__all__'` value indicates that all fields in the respective model should be included in the serialized data. These serializers are typically used to convert complex data types to Python data types that can then be easily rendered into JSON for API responses.

OUTPATIENT APPOINTMENT API



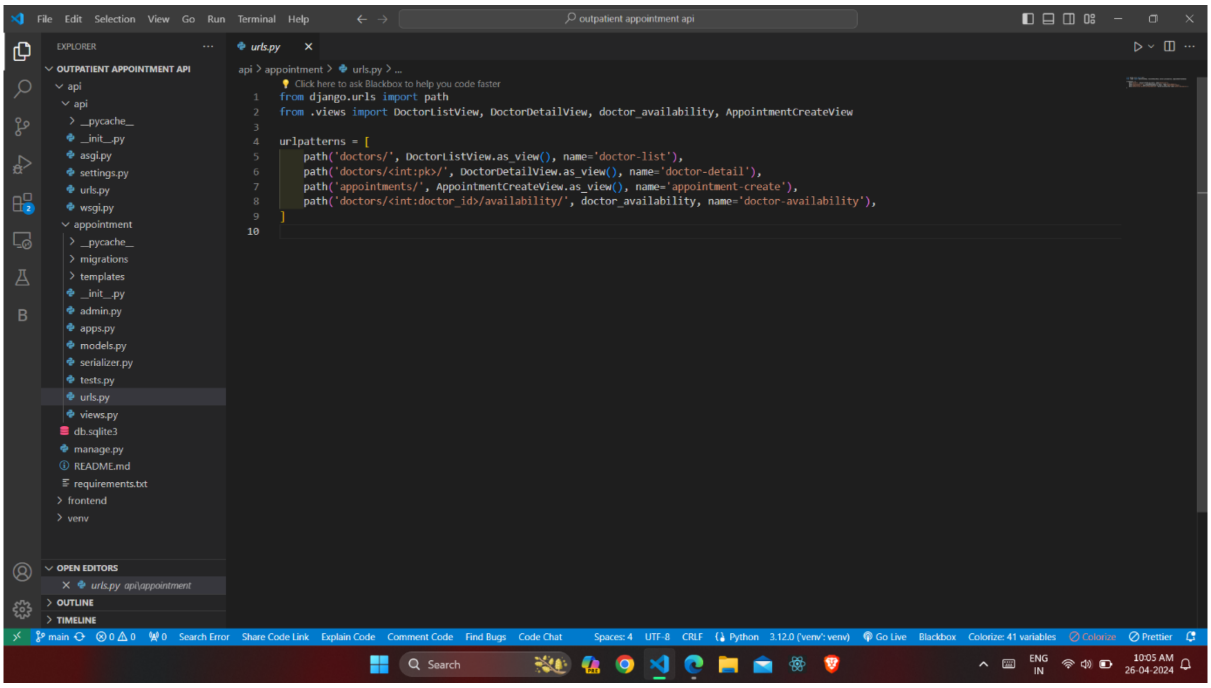
```python
import datetime
from rest_framework import generics
from rest_framework.decorators import api_view
from rest_framework.response import Response
from .models import Doctor, WeeklySchedule, Appointment
from .serializer import DoctorSerializer, WeeklyScheduleSerializer, AppointmentSerializer

class DoctorListView(generics.ListAPIView):
    queryset = Doctor.objects.all()
    serializer_class = DoctorSerializer

class DoctorDetailView(generics.RetrieveAPIView):
    queryset = Doctor.objects.all()
    serializer_class = DoctorSerializer
    lookup_field = 'pk'


@api_view(['GET'])
def doctor_availability(request, doctor_id):
    queryset = WeeklySchedule.objects.filter(doctor_id=doctor_id)
    serializer = WeeklyScheduleSerializer(queryset, many=True)
    return Response(serializer.data)

class AppointmentCreateView(generics.CreateAPIView):
    serializer_class = AppointmentSerializer
```

Views.py

1. **DoctorAvailability**: This class inherits from `generics.ListAPIView` and is set up to list all doctor objects using `DoctorSerializer`.
2. **DoctorDetail**: Inherits from `generics.RetrieveAPIView` and retrieves a specific doctor by their primary key (`pk`), again using `DoctorSerializer`.
3. **doctor_availability Function**: Defined with the `@api_view` decorator for HTTP GET requests, it filters the `WeeklySchedule` by `doctor_id`, serializes the data, and returns it in the response.
4. **AppointmentCreateView**: A class for creating new appointments, inheriting from `generics.CreateAPIView` and using `AppointmentSerializer`.

This script is part of the backend API, allowing front-end applications to interact with the database to retrieve doctors' availability and manage appointments. The use of class-based views and function-based view demonstrates Django's flexibility in handling different types of HTTP requests.

OUTPATIENT APPOINTMENT API



```python
from django.urls import path
from .views import DoctorListView, DoctorDetailView, doctor_availability, AppointmentCreateView

urlpatterns = [
    path('doctors/', DoctorListView.as_view(), name='doctor-list'),
    path('doctors/<int:pk>/', DoctorDetailView.as_view(), name='doctor-detail'),
    path('appointments/', AppointmentCreateView.as_view(), name='appointment-create'),
    path('doctors/<int:doctor_id>/availability/', doctor_availability, name='doctor-availability'),
]
```

Urls.py:

1. **URL Patterns**:
   - A list named `urlpatterns` contains the URL patterns for the app.
   - **List View**: The URL ending in ‘doctors/’ is mapped to `DoctorListView`, which likely displays a list of doctors.
   - **Detail View**: A URL like ‘doctors/5/’ (where ‘5’ is a doctor’s ID) is directed to `DoctorDetailView`, showing details for a specific doctor.
   - **Appointment Creation**: Adding ‘appointment/’ to the URL directs to `AppointmentCreateView` for scheduling an appointment with that doctor.
   - **Availability**: The URL ending in ‘availability/’ maps to `doctor_availability`, likely showing when the doctor is available for appointments.

This setup helps users navigate to different parts of the application to view doctors, their details, schedule appointments, and check availability.