universidade de aveiro
deti departamento de electrónica, telecomunicações e informática

Tecnologias e Programação Web 2019/2020

Django Framework

Django Framework

RESTful Web Services

Django REST framework

Web Services

- Web services are services offered by electronic devices to send data to another electronic devices, using web technologies.
- Normally, data is transmitted in JSON or XML format.
- One of the main purposes of web services is to provide interoperability and data integration between heterogeneous information systems.

REST *Web Services*

- REST – *Representational State Transfer*
  - It's an architectural model for hypermedia applications, mainly used for web services implementation, which are considered to be light, simple, sustainable and scalable.
  - A service based on this technology is named as RESTful Service.
  - REST services don't depend on any particular protocol, however most of them use HTTP for transporting.
  - Another kind of web services are the SOAP Web Services. These are based on the SOAP protocol, which is very formal, strict and heavy. That's why it's not often used.

Django REST framework (DRF)

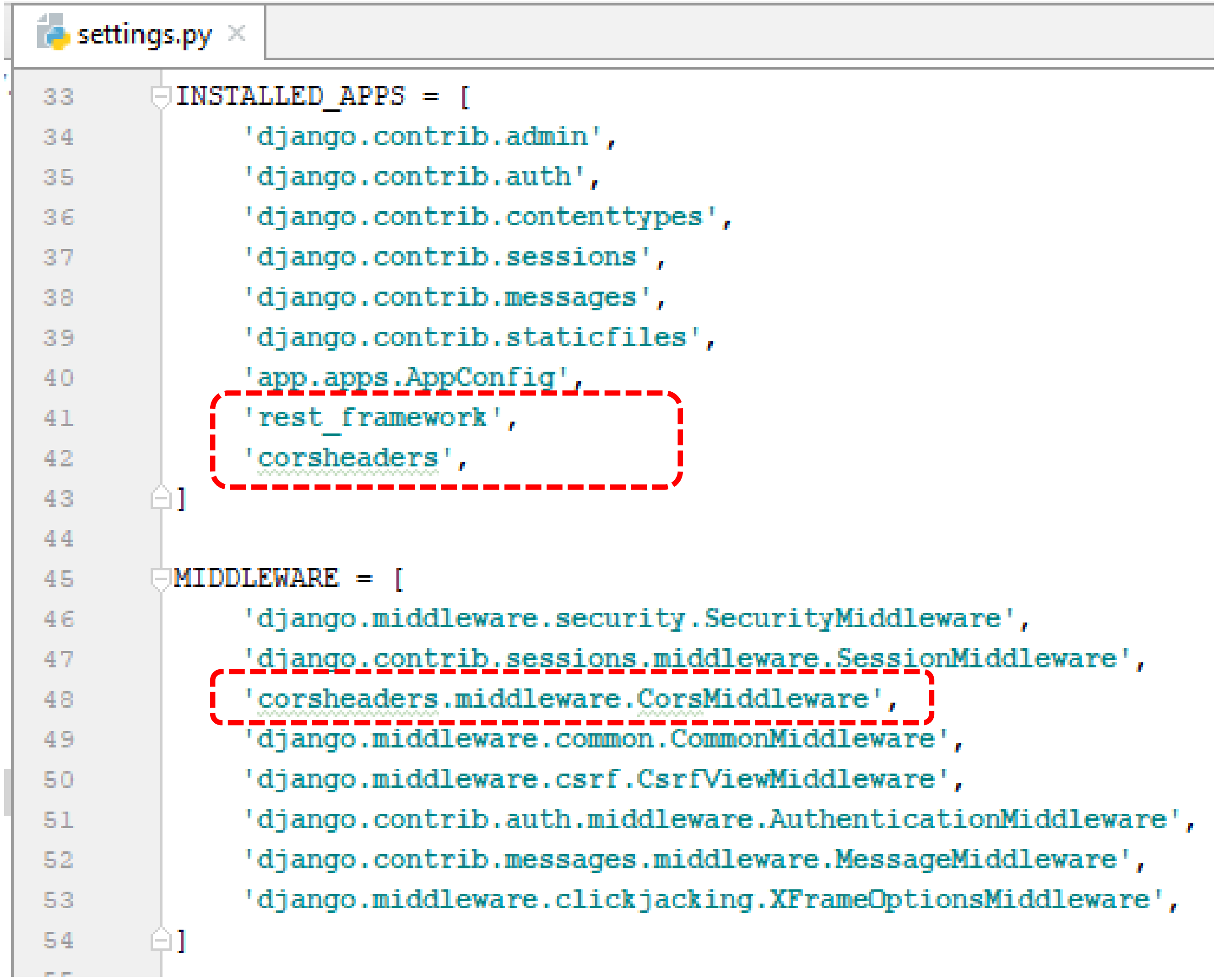
- DRF is a python library to create REST Web Services integrated with Django framework.
- It provides an important set of functions for ease programming this kind of services, as:
  - The possibility to publish the provided API;
  - Authentication policies, using OAuth1a and OAuth2 protocols;
  - Data serialization from DBs, through Django ORM or other means;
  - It can use general views if advanced facilities aren’t needed;
  - Currently, it’s used by big organizations (Mozilla, Red Hat, etc.), what proves its credibility.

DRF - Installing

- Installing
  - pip install djangorestframework
  - pip install markdown
  - pip install django-filter
  - pip install django-cors-headers

Configuring (i)

- Add the following text lines “settings.py” file:

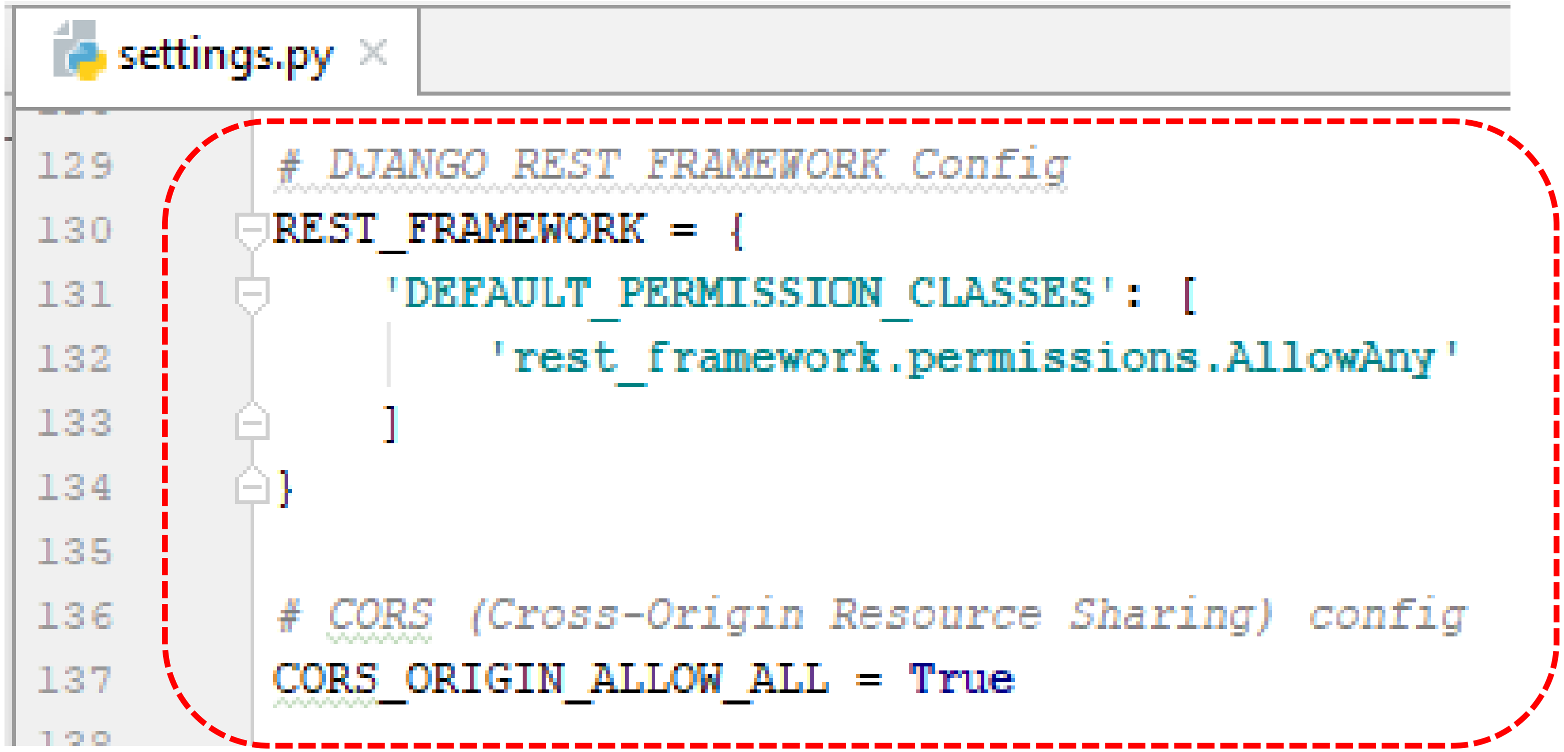


settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app.apps.AppConfig',
    'rest_framework',
    'corsheaders',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

Configuring (ii)

- Add the following configuration to “settings.py” file:

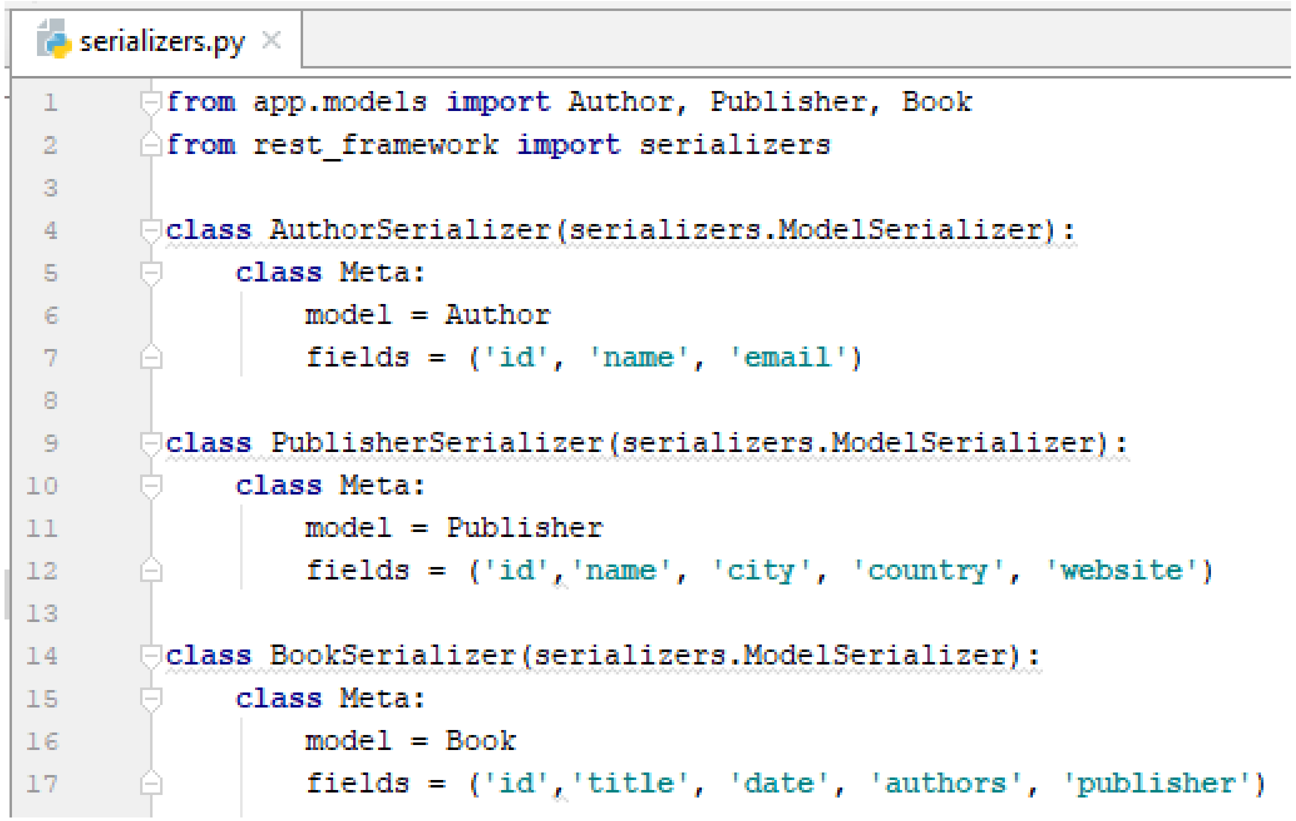


settings.py

```
# DJANGO REST FRAMEWORK Config
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny'
    ]
}

# CORS (Cross-Origin Resource Sharing) config
CORS_ORIGIN_ALLOW_ALL = True
```

Serializers

- Creating serializers to put data from BD in a sending format.
  - Create a file named “serializers.py” in folder “app”.

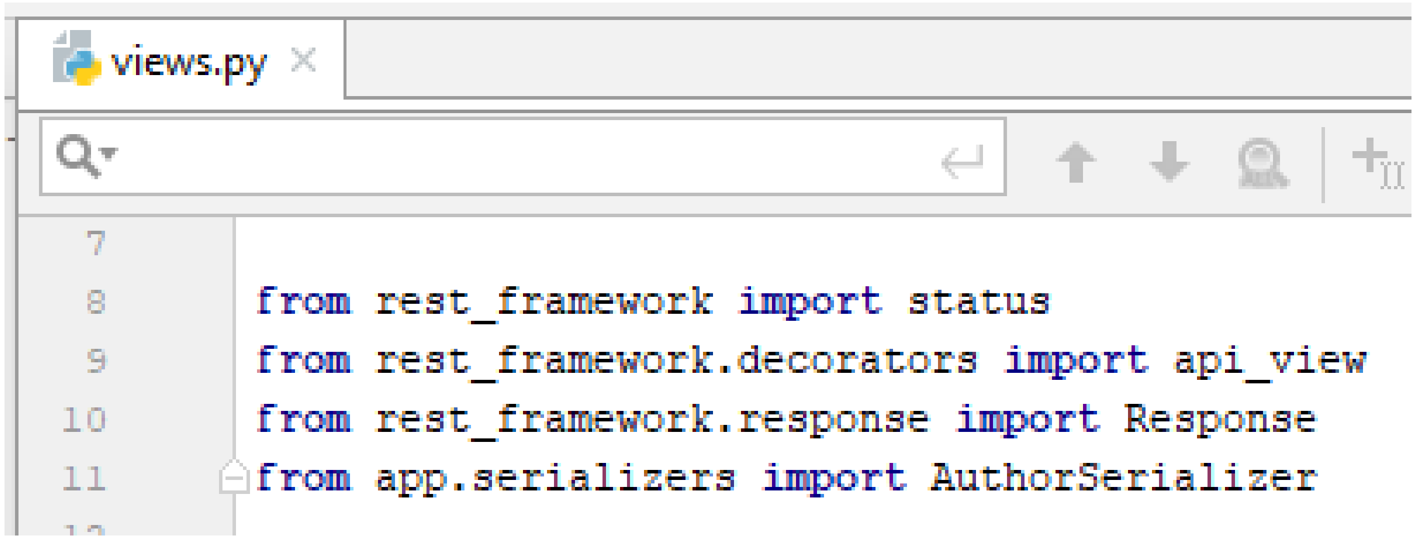


serializers.py

```
from app.models import Author, Publisher, Book
from rest_framework import serializers

class AuthorSerializer(serializers.ModelSerializer):
    class Meta:
        model = Author
        fields = ('id', 'name', 'email')

class PublisherSerializer(serializers.ModelSerializer):
    class Meta:
        model = Publisher
        fields = ('id','name', 'city', 'country', 'website')

class BookSerializer(serializers.ModelSerializer):
    class Meta:
        model = Book
        fields = ('id','title', 'date', 'authors', 'publisher')
```

Views (i)

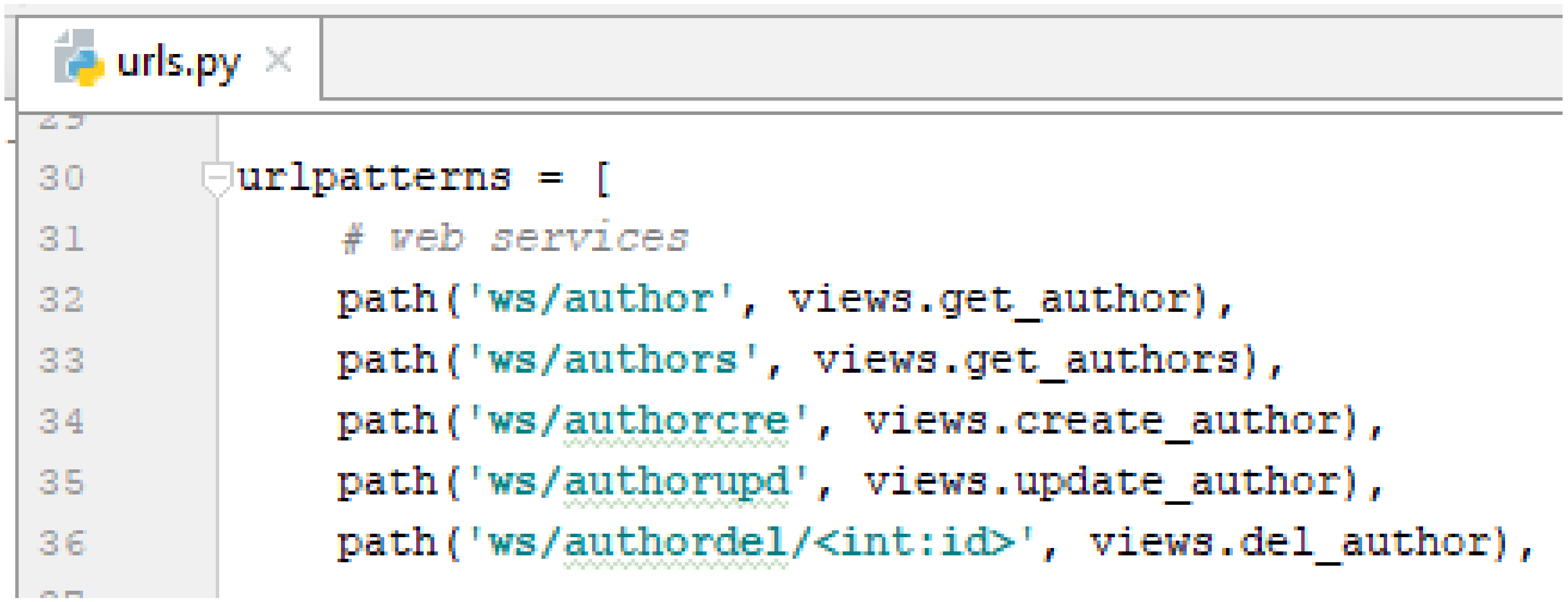
- Creating views to send data
  - Imports:



```python
from rest_framework import status
from rest_framework.decorators import api_view
from rest_framework.response import Response
from app.serializers import AuthorSerializer
```

Views (ii)

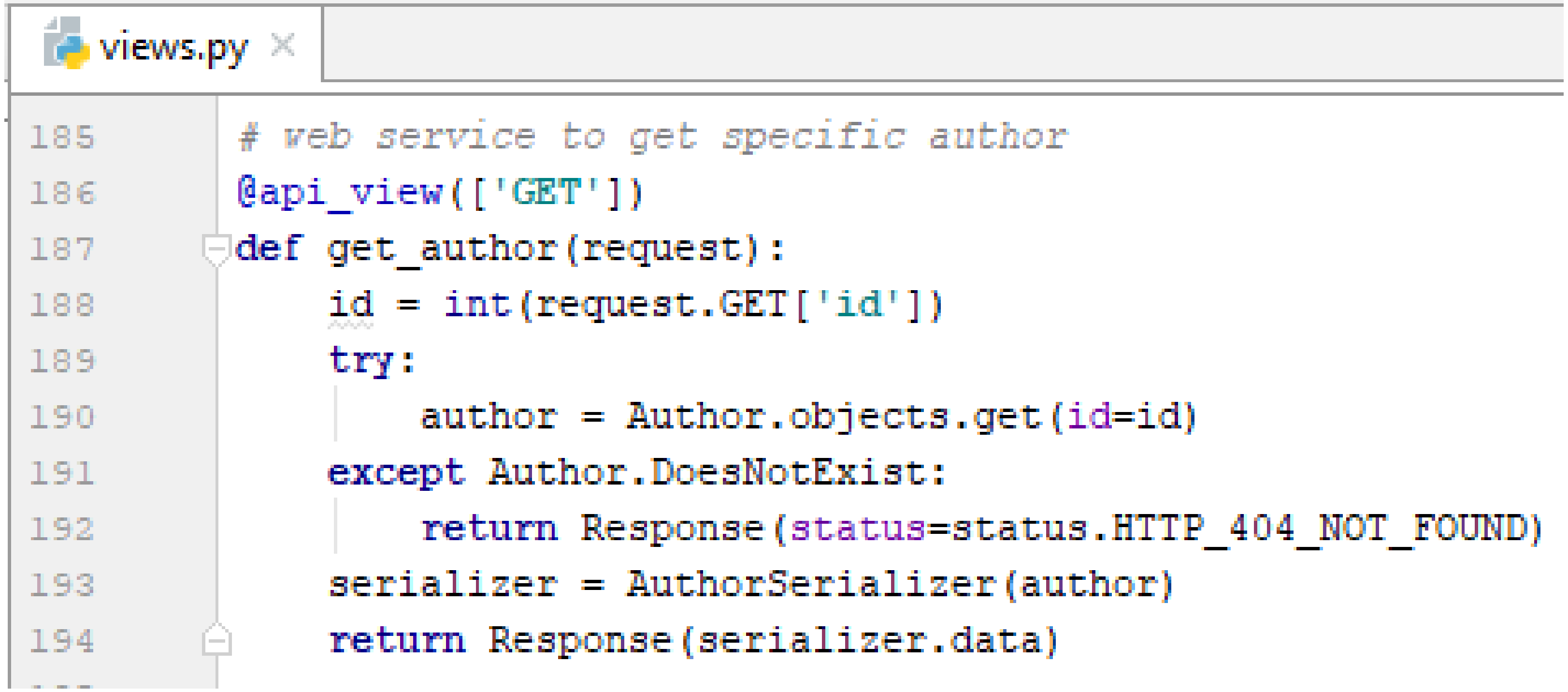
- Configuring urls routes.


urls.py

```
urlpatterns = [
    # web services
    path('ws/author', views.get_author),
    path('ws/authors', views.get_authors),
    path('ws/authorcre', views.create_author),
    path('ws/authorupd', views.update_author),
    path('ws/authordel/<int:id>', views.del_author),
```

Views (iii)

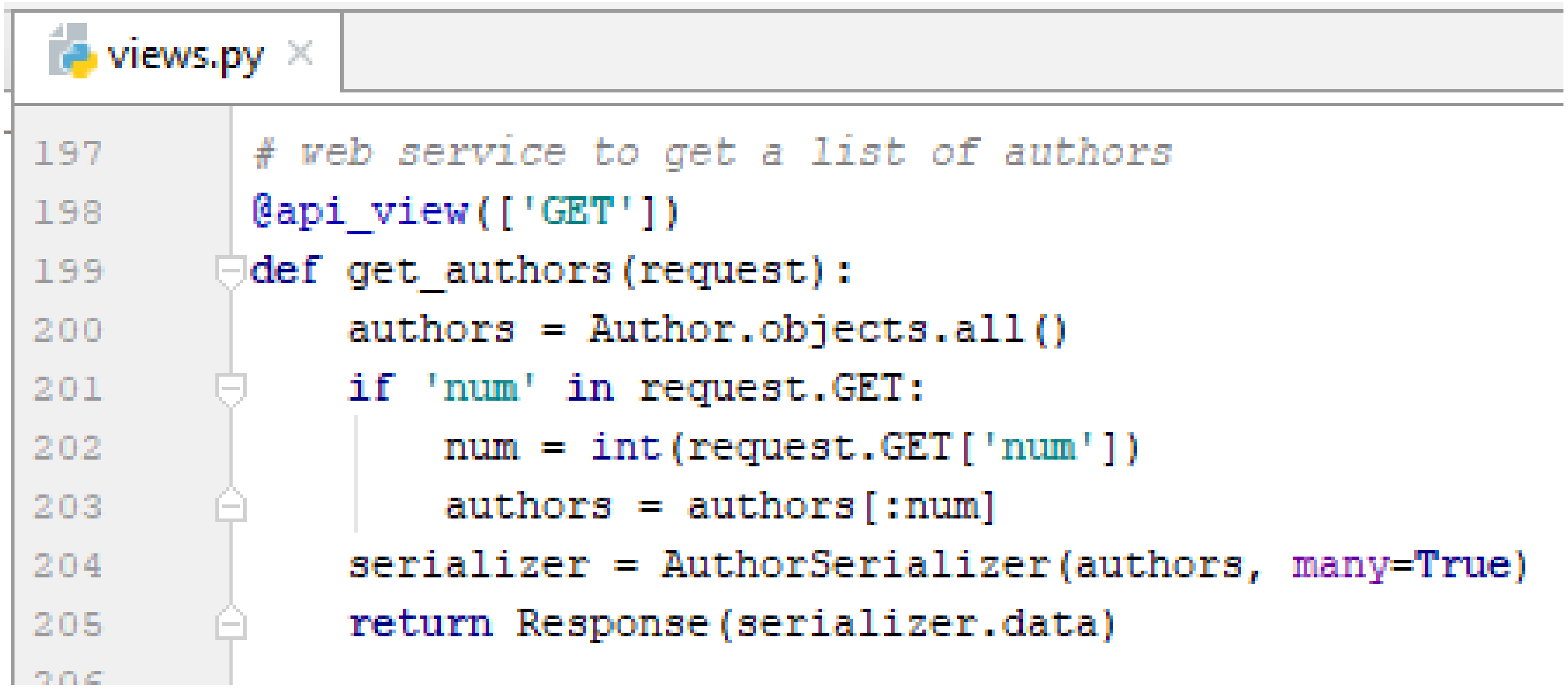
- View to get one author.


views.py

```python
# web service to get specific author
@api_view(['GET'])
def get_author(request):
    id = int(request.GET['id'])
    try:
        author = Author.objects.get(id=id)
    except Author.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)
    serializer = AuthorSerializer(author)
    return Response(serializer.data)
```

Views (iv)

- View to get a list of authors.

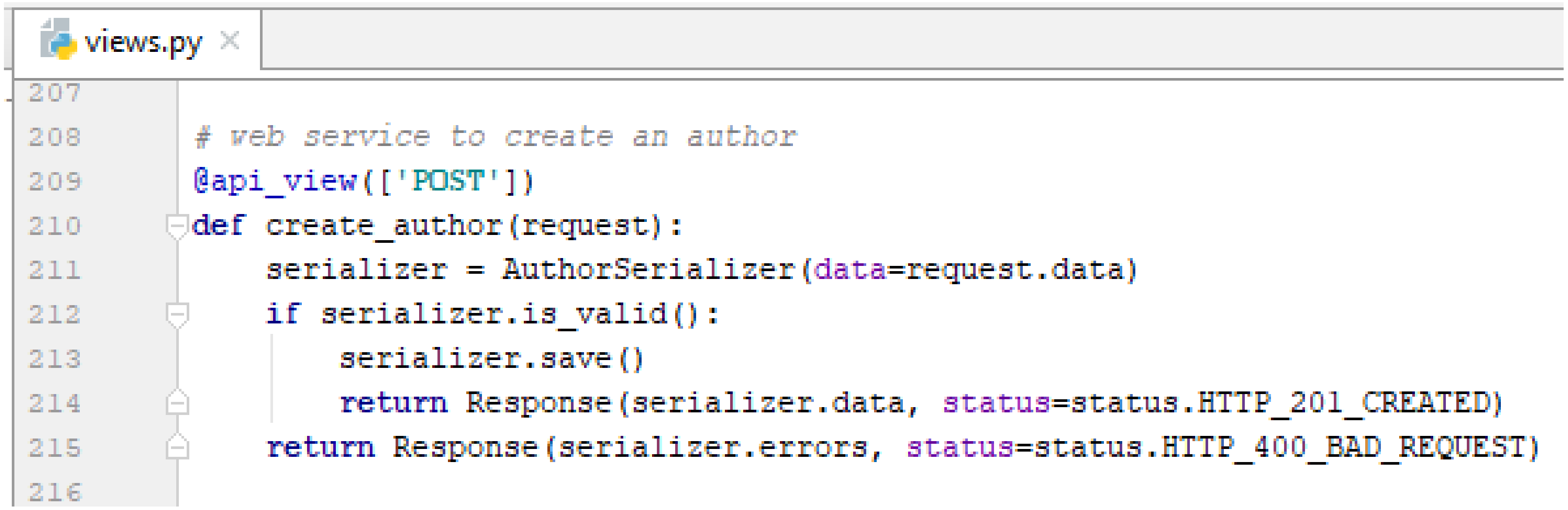


views.py

```python
# web service to get a list of authors
@api_view(['GET'])
def get_authors(request):
    authors = Author.objects.all()
    if 'num' in request.GET:
        num = int(request.GET['num'])
        authors = authors[:num]
    serializer = AuthorSerializer(authors, many=True)
    return Response(serializer.data)
```

Views (v)

- View to create an author.

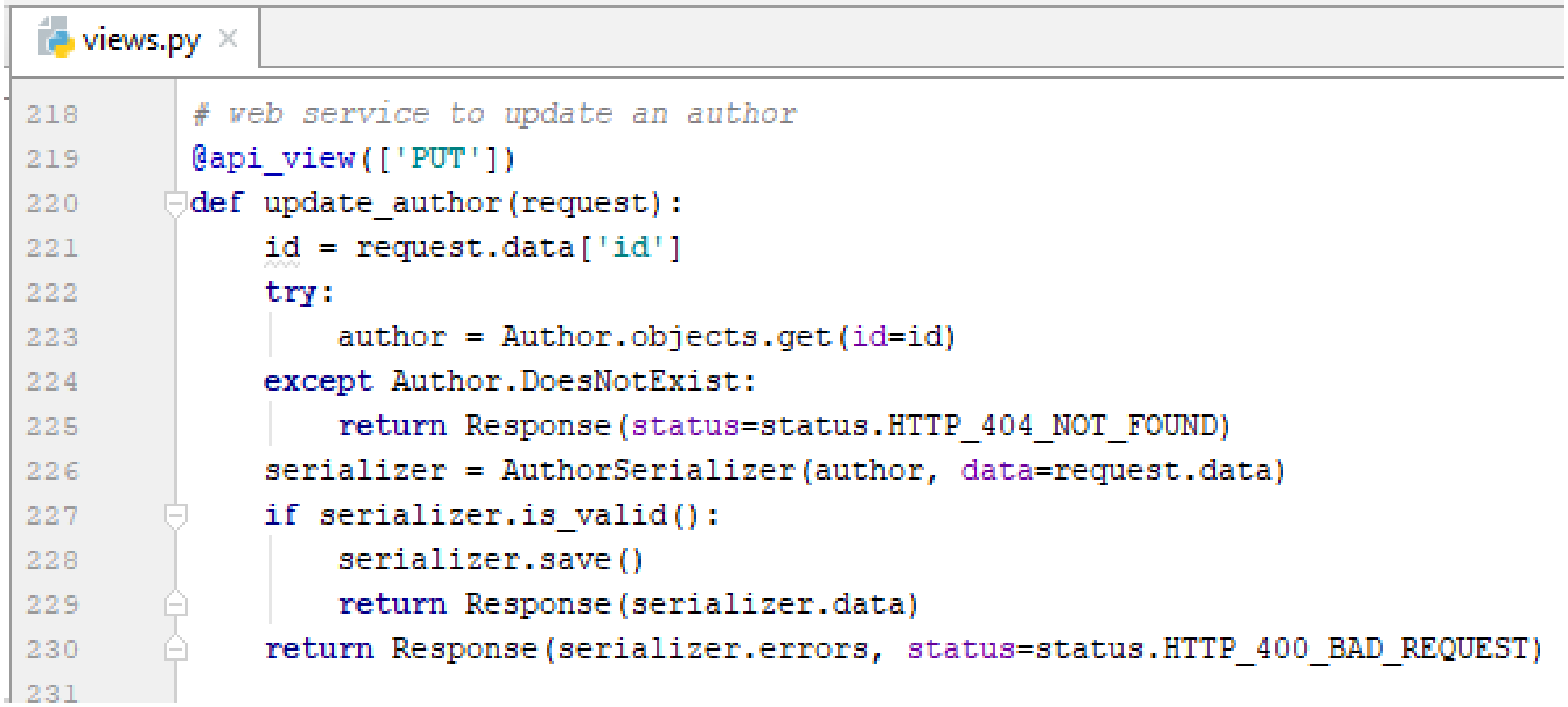


views.py

```python

# web service to create an author
@api_view(['POST'])
def create_author(request):
    serializer = AuthorSerializer(data=request.data)
    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)

```

Views (vi)

- View to update an author.

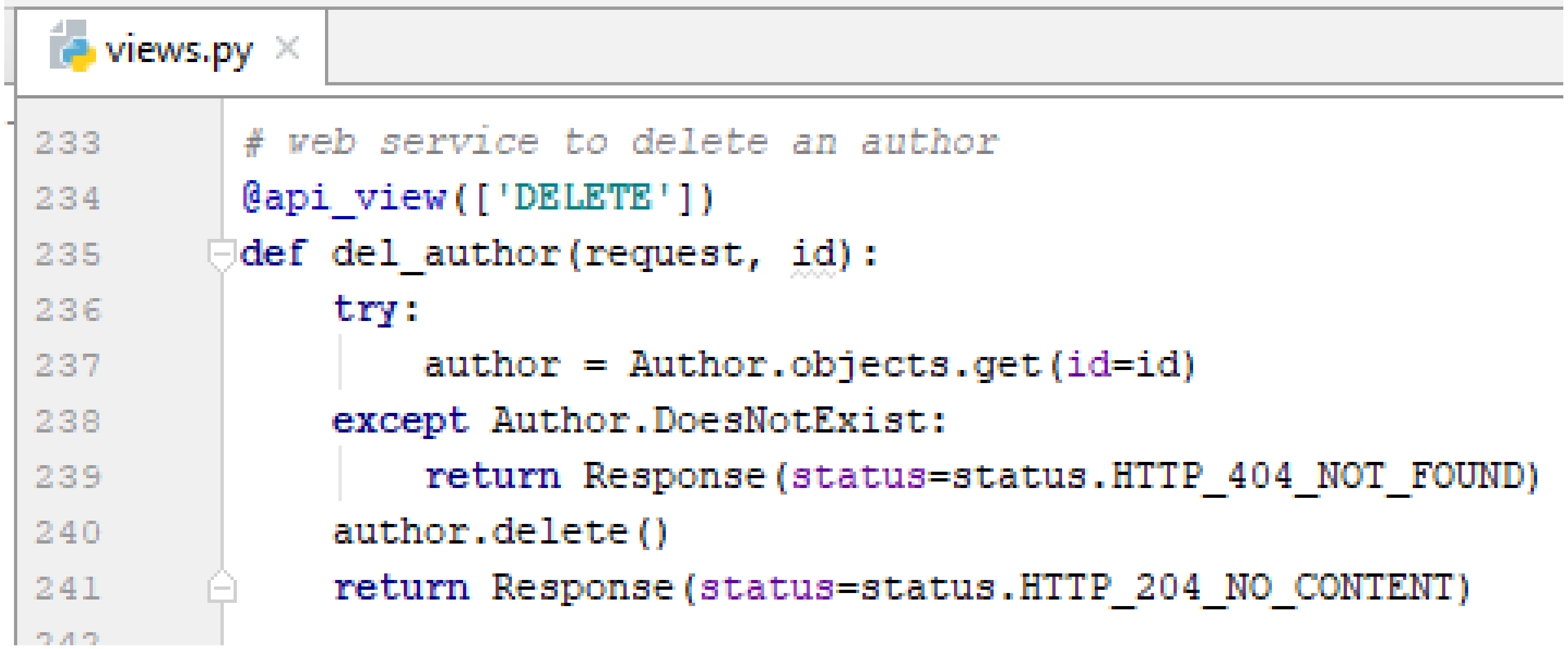


views.py

```
# web service to update an author
@api_view(['PUT'])
def update_author(request):
    id = request.data['id']
    try:
        author = Author.objects.get(id=id)
    except Author.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)
    serializer = AuthorSerializer(author, data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Views (vii)

- View to delete an author.



views.py ×

```
# web service to delete an author
@api_view(['DELETE'])
def del_author(request, id):
    try:
        author = Author.objects.get(id=id)
    except Author.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)
    author.delete()
    return Response(status=status.HTTP_204_NO_CONTENT)
```