**Django introduction**

v-02

install python and local environment setup

v-03 (virtual environment)

*Declaring environment*

python –m venv myenv(windows)

source myenv/bin/activate(linux)

*Activate it :*

myenv\Scripts\activate

then just install it :   
pip install Django

v-04 (creating project)

first create a folder:

mkdir tutorial

cd tutorial

python -m venv myenv

myenv\Scripts\activate

now install:  
pip install Django

create project:

django-admin startproject first .

python manage.py runserver

v-05 (pycharm trick and installation)

first install pycharm and it will automatically add the virtual environment into it.

After that run the manage.py

It will show inside of the pycharm and then just run by F10

Then configure and set parameter : runserver then save and run

It will run automatically : python manage.py runserver

v-06 (files overview and app)

creating an app command:

django-admin startapp demo

v-07 (migrations)

py manage.py migrate

then we gonna make model in our application:

models.py

set the app to the settings.py file

run: py manage.py makemigrations

now again.: py manage.py migrate (now it ill add inside of our database)

so we have created and learned how to use .gitignore file and just ignore vitual environment to push in git repo:   
open the file and write the name of virtual env:

**myenv/**

That’s it as simple as that

v-08 (user and admin)

so in this video I learned :   
how to create a super user:

py manage.py createsuperuser

and to show the models inside of database:

just go inside of admin.py file:

from .models import Book  
  
# Register your models here.  
admin.site.register(Book)

then just indicate the location and model inside of admin   
after save and reloading all the models will appear here.

v-09 (field option)

Here discussed about how work field options like null, blank, choices , unique,

# STATUSES = (  
# (0, 'Unknown'),  
# (1, 'processed'),  
# (2, 'paid')  
# )  
  
# title = models.CharField(null=True, blank=False, unique=True,  
# default='', choices=STATUSES)

Then just command:

py manage.py makemigrations

finally:

py manage.py migrate

v-10 (field type)

CharField()

TextField()

Blank= True (means store blank)

Null = True means store null

title = models.CharField(max\_length=36, blank=False, unique=True)  
description = models.TextField(max\_length=256, blank=True)  
  
# price = models.FloatField(default=0, max\_digits=0, decimal\_places=2)  
price = models.DecimalField(default=0, decimal\_places=2)  
# price = models.BigIntegerField(default=0)  
  
# published = models.DateField(auto\_now=True, auto\_now\_add=True)  
# published = models.TimeField(auto\_now=True, auto\_now\_add=True)  
# published = models.DateTimeField(auto\_now=True, auto\_now\_add=True)  
published = models.DateField()  
is\_published = models.BooleanField(default=False)  
  
cover = models.FileField(upload\_to='covers')  
# cover = models.ImageField(upload\_to='covers', height\_field='')

saw all the types of field such as Decimal, Date, etc

then:   
py manage.py makemigrations

py manage.py migrate

v-11 (urls)

so basically the I am changing the urls from project first/urls.py

from django.contrib import admin  
from django.urls import path, include  
  
urlpatterns = [  
 path('demo/', include('demo.urls')),  
 path('admin/', admin.site.urls),  
]

then changing demo/urls.py

from django.urls import path  
from . import views  
  
urlpatterns = {  
 path('', views.first),  
}

finally changing the view file:

from django.shortcuts import render  
from django.http import HttpResponse  
# Create your views here.  
def first(request):  
 return HttpResponse('First message!')

v-12 (class based view)

so class based view we use for so many functionality builtin here and easy to work.

To work with class view you have to go views.py:

Then import

from django.views import View

after that write the class based view:

class Another(View):  
 def get(self, request):  
 return HttpResponse('another method in class!')

then edit the demo/urls.py:

from .views import Another

urlpatterns = {  
 path('first', views.first),  
 path('another', Another.as\_view()),  
  
}

v-13 (Model objects method)

we gonna show all the models data from database in frontpage.

So first go to the views.py

from .models import Book

then just work with that: objects.all

from django.shortcuts import render  
from django.http import HttpResponse  
from django.views import View  
from .models import Book  
  
  
  
# Create your views here.  
  
class Another(View):  
  
 books = Book.objects.all()  
 output = ''  
 for book in books:  
 output += f"we have {book.title} books with ID {book.id}!!!<br>"  
  
 # output = f"we have {len(books)} books in DB !!!"  
 def get(self, request):  
 # return HttpResponse('another method in class!')  
 return HttpResponse(self.output)

we can also use filter:

class Another(View):  
  
 # books = Book.objects.all()  
 books = Book.objects.filter(is\_published=True)  
 output = ''  
 for book in books:  
 output += f"we have {book.title} books with ID {book.id}!!!<br>"

get:

class Another(View):  
  
 # books = Book.objects.all()  
 # books = Book.objects.filter(is\_published=True)  
 book = Book.objects.get(id=1) #always brings 1 record  
  
 output = f"we have {book.title} books with ID {book.id}!!!<br>"  
  
 # output = ''  
 # for book in books:  
 # output += f"we have {book.title} books with ID {book.id}!!!<br>"  
  
 # output = f"we have {len(books)} books in DB !!!"  
 def get(self, request):  
 # return HttpResponse('another method in class!')  
 return HttpResponse(self.output)

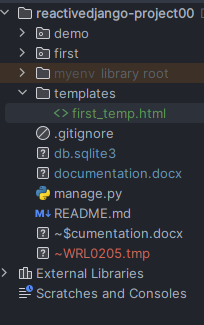
v-14 (Templates)

for creating template first open the settings.py

select templates directory and:

TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': ['templates'],  
 'APP\_DIRS': True,  
 'OPTIONS': {  
 'context\_processors': [  
 'django.template.context\_processors.debug',  
 'django.template.context\_processors.request',  
 'django.contrib.auth.context\_processors.auth',  
 'django.contrib.messages.context\_processors.messages',  
 ],  
 },  
 },  
]

Then create folder outside



Then finally : change the views.py

edit the demo/urls.py

from django.urls import path  
from . import views  
from .views import Another  
  
  
urlpatterns = [  
 path('first', views.first)  
 # path('another', Another.as\_view()),  
  
]

finally:

views.py

from django.shortcuts import render  
from django.http import HttpResponse  
from django.views import View  
from .models import Book  
from django.shortcuts import render  
  
# class Another(View):  
#  
# # books = Book.objects.all()  
# # books = Book.objects.filter(is\_published=True)  
# book = Book.objects.get(id=1) #always brings 1 record  
#  
# output = f"we have {book.title} books with ID {book.id}!!!<br>"  
#  
# # output = ''  
# # for book in books:  
# # output += f"we have {book.title} books with ID {book.id}!!!<br>"  
#  
# # output = f"we have {len(books)} books in DB !!!"  
# def get(self, request):  
# # return HttpResponse('another method in class!')  
# return HttpResponse(self.output)  
  
def first(request):  
 return render(request, 'first\_temp.html')

v-15 (Dynamic Templates)

change the view.py option:

def first(request):  
 books = Book.objects.all()  
  
  
 return render(request, 'first\_temp.html', {'books': books})

so many templating system like jinja template

finally template:

<h1> This is my first template </h1>  
<!-- <h2> {{ data }} </h2>-->  
  
 {{ books }}  
  
 {% for book in books %}  
 {% if book.is\_published %}  
 <h2> {{ book.title }} </h2>  
 {% endif %}  
 {% endfor %}

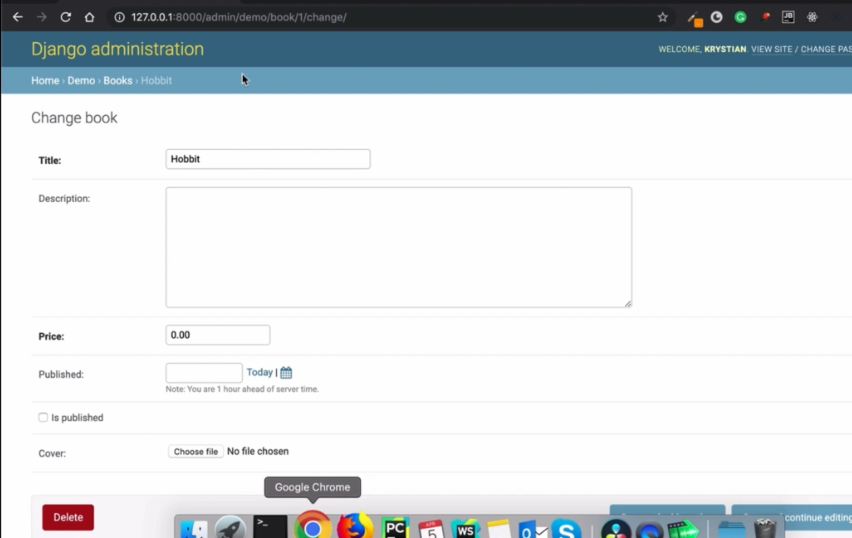
v-16 (admin customize)

for handing that to human readable:

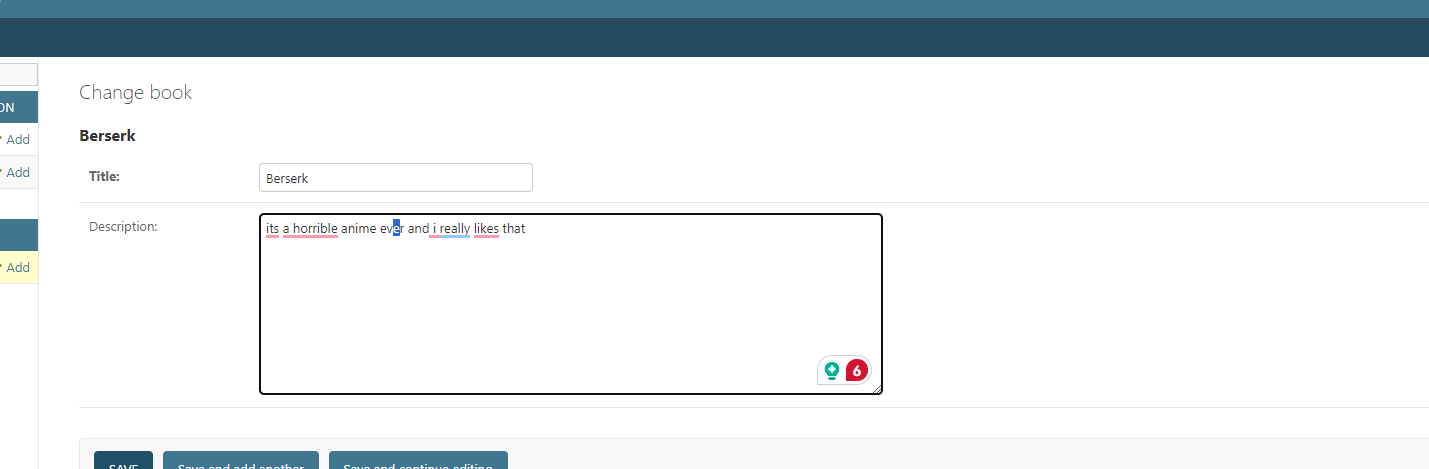


We added last 2 line and fix it to title:

from django.db import models  
  
# Create your models here.  
class Book(models.Model):  
  
 # STATUSES = (  
 # (0, 'Unknown'),  
 # (1, 'processed'),  
 # (2, 'paid')  
 # )  
  
 # title = models.CharField(null=True, blank=False, unique=True,  
 # default='', choices=STATUSES)  
  
 title = models.CharField(max\_length=36, blank=False, unique=True)  
 description = models.TextField(max\_length=256, blank=True)  
  
 # price = models.FloatField(default=0, max\_digits=0, decimal\_places=2)  
 price = models.DecimalField(default=0, max\_digits=4, decimal\_places=2)  
 # price = models.BigIntegerField(default=0)  
  
 # published = models.DateField(auto\_now=True, auto\_now\_add=True)  
 # published = models.TimeField(auto\_now=True, auto\_now\_add=True)  
 # published = models.DateTimeField(auto\_now=True, auto\_now\_add=True)  
 published = models.DateField(blank=True, null=True, default=None)  
 is\_published = models.BooleanField(default=False)  
  
 # cover = models.FileField(upload\_to='covers')  
 cover = models.ImageField(upload\_to='covers', blank=True)  
  
 def \_\_str\_\_(self):  
 return self.title

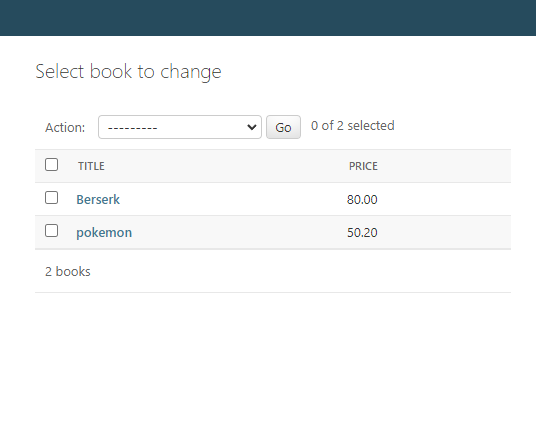


from django.contrib import admin  
from .models import Book  
  
# Register your models here.  
# generic site of admin book  
# admin.site.register(Book)  
  
@admin.register(Book)  
class BookAdmin(admin.ModelAdmin):  
 fields = ['title']



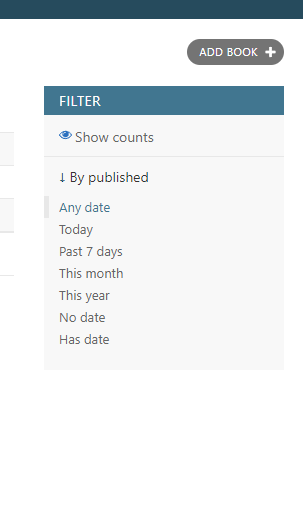
We can hide if we want each fields.

list\_display = ['title', 'price']



Filter:

list\_filter = ['published']



v-17 (Rest)

pip install djangorestframework

then just add in settings.py > app section:

INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'rest\_framework',  
 'demo',  
  
]

If some of the framework comes in then its prebuiltin. No need to use py manage.py makemigrations because its prebuilt in rest.

py manage.py migrate

v-18 (Serializers)

we create a serializers.py file

and write this script:

from rest\_framework import serializers  
from .models import Book  
  
class BookSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Book  
 fields = ['title']

then go to the views.py file to write viewsets:

# from django.shortcuts import render  
# from django.http import HttpResponse  
# from django.views import View  
# from .models import Book  
# from django.shortcuts import render  
  
from rest\_framework import viewsets  
from .serializers import BookSerializer  
from .models import Book  
  
# class Another(View):  
#  
# # books = Book.objects.all()  
# # books = Book.objects.filter(is\_published=True)  
# book = Book.objects.get(id=1) #always brings 1 record  
#  
# output = f"we have {book.title} books with ID {book.id}!!!<br>"  
#  
# # output = ''  
# # for book in books:  
# # output += f"we have {book.title} books with ID {book.id}!!!<br>"  
#  
# # output = f"we have {len(books)} books in DB !!!"  
# def get(self, request):  
# # return HttpResponse('another method in class!')  
# return HttpResponse(self.output)  
  
# def first(request):  
# books = Book.objects.all()  
#  
#  
# return render(request, 'first\_temp.html', {'books': books})  
  
  
#so we are creating a builtin view for our books that will use all power of django and create option like HTTP methods for us  
class BookViewSet(viewsets.ModelViewSet):  
 serializer\_class = BookSerializer  
 queryset = Book.objects.all()

Now create url router inside of demo/urls.py

from django.urls import path, include  
# from . import views  
# from .views import Another  
from rest\_framework import routers  
from .views import BookViewSet  
  
router = routers.DefaultRouter()  
router.register('books', BookViewSet)  
  
  
urlpatterns = [  
 path('', include(router.urls))  
 # path('first', views.first)  
 # path('another', Another.as\_view()),  
  
]

v-19 (Postman Http methods)

learned to use postman methods using api:

get post put delete

v-20 (Token)

settings.py   
add another application inside of installed apps:  
'rest\_framework.authtoken',

then apply migrate:

py manage.py migrate

then we can create token from admin panel and generate our unique token

now how can we get that token.???

Go to the project first or demo

First//urls.py

from django.contrib import admin  
from django.urls import path, include  
from rest\_framework.authtoken.views import obtain\_auth\_token  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('demo/', include('demo.urls')),  
 path('auth/', obtain\_auth\_token)  
  
]

after that just run the server and open postman

<http://127.0.0.1:8000/auth/>

after hitting the server not allowed

so select post option and send option again

select body > form data:

after giving username and pass the token will appear.

v-21 (Permission)

if we put below code in settings.py file:

REST\_FRAMEWORK = {  
 'DEFAULT\_PERMISSION\_CLASSES': (  
 'rest\_framework.permissions.IsAuthenticated',  
 )  
}

And run postman to get the api. We never can allow to read the data because its now secure

To get in it we just have to hit the url:

Generate the token from <http://127.0.0.1:8000/auth/>

By providing username and password using post method

Get : <http://127.0.0.1:8000/demo/books/>

The we set in “HEADERS” option:

Authorization Token fcac03dfc1a0c905119691974cf176364c4c5683

Now just view.py and add two line:

# from django.shortcuts import render  
# from django.http import HttpResponse  
# from django.views import View  
# from .models import Book  
# from django.shortcuts import render  
  
from rest\_framework import viewsets  
from .serializers import BookSerializer  
from .models import Book  
  
from rest\_framework.authentication import TokenAuthentication  
  
# class Another(View):  
#  
# # books = Book.objects.all()  
# # books = Book.objects.filter(is\_published=True)  
# book = Book.objects.get(id=1) #always brings 1 record  
#  
# output = f"we have {book.title} books with ID {book.id}!!!<br>"  
#  
# # output = ''  
# # for book in books:  
# # output += f"we have {book.title} books with ID {book.id}!!!<br>"  
#  
# # output = f"we have {len(books)} books in DB !!!"  
# def get(self, request):  
# # return HttpResponse('another method in class!')  
# return HttpResponse(self.output)  
  
# def first(request):  
# books = Book.objects.all()  
#  
#  
# return render(request, 'first\_temp.html', {'books': books})  
  
  
#so we are creating a builtin view for our books that will use all power of django and create option like HTTP methods for us  
class BookViewSet(viewsets.ModelViewSet):  
 serializer\_class = BookSerializer  
 queryset = Book.objects.all()  
 authentication\_classes = (TokenAuthentication,)

So now if we hit then we can see. This is for whom have authentication

We can also use :

REST\_FRAMEWORK = {  
 'DEFAULT\_PERMISSION\_CLASSES': (  
 'rest\_framework.permissions.AllowAny',  
 )  
}

So that we can do it both ways.

We also can ALLOW THE AUTHENTICATE USING views.py file :  
add this upper:

from rest\_framework.permissions import IsAuthenticated

add the last line:

class BookViewSet(viewsets.ModelViewSet):  
 serializer\_class = BookSerializer  
 queryset = Book.objects.all()  
 authentication\_classes = (TokenAuthentication,)  
 permission\_classes = (IsAuthenticated,)

now we cant login without authentication

v-22 (relationship in databases One to one)

we have 3 relationship basically

* One to one
* One to many
* Many to many

DESIGN model first carefully and take time and then start implementing models because of up to down.

from django.db import models  
  
# Create your models here.  
  
  
#something extra information related to that particular book  
class BookNumber(models.Model):  
 isbn\_10 = models.CharField(max\_length=10, blank=True)  
 isbn\_13 = models.CharField(max\_length=13, blank=True)  
  
  
class Book(models.Model):  
  
 # STATUSES = (  
 # (0, 'Unknown'),  
 # (1, 'processed'),  
 # (2, 'paid')  
 # )  
  
 # title = models.CharField(null=True, blank=False, unique=True,  
 # default='', choices=STATUSES)  
  
 title = models.CharField(max\_length=36, blank=False, unique=True)  
 description = models.TextField(max\_length=256, blank=True)  
  
 # price = models.FloatField(default=0, max\_digits=0, decimal\_places=2)  
 price = models.DecimalField(default=0, max\_digits=4, decimal\_places=2)  
 # price = models.BigIntegerField(default=0)  
  
 # published = models.DateField(auto\_now=True, auto\_now\_add=True)  
 # published = models.TimeField(auto\_now=True, auto\_now\_add=True)  
 # published = models.DateTimeField(auto\_now=True, auto\_now\_add=True)  
 published = models.DateField(blank=True, null=True, default=None)  
 is\_published = models.BooleanField(default=False)  
  
 # cover = models.FileField(upload\_to='covers')  
 cover = models.ImageField(upload\_to='covers', blank=True)  
  
 # To integrate our new field  
  
 number = models.OneToOneField(BookNumber, null=True,  
 blank=True, on\_delete=models.CASCADE)  
  
  
  
 def \_\_str\_\_(self):  
 return self.title

py manage.py makemigrations

py manage.py migrate

now go admin panel and play

to show it on postman:

class BookSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Book  
 fields = ['id', 'title','description', 'price', 'published',  
 'is\_published', 'number']

to show the actual isbn number in postman:

from rest\_framework import serializers  
from .models import Book, BookNumber  
  
  
class BookNumberSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = BookNumber  
 fields = ['id', 'isbn\_10','isbn\_13']  
  
class BookSerializer(serializers.ModelSerializer):  
  
 number = BookNumberSerializer(many=False)  
 class Meta:  
 model = Book  
 fields = ['id', 'title','description', 'price', 'published',  
 'is\_published', 'number']

finally we can see id, isbn10 and 13 number also.

v-23 (relationship in databases One to Many)

class Character(models.Model):  
 name = models.CharField(max\_length=30)  
 book = models.ForeignKey(Book, on\_delete=models.CASCADE)

py manage.py makemigrations

py manage.py migrate

register inside of admin.py

from django.contrib import admin  
from .models import Book, BookNumber, Character  
  
# Register your models here.  
# generic site of admin book  
# admin.site.register(Book)  
  
@admin.register(Book)  
class BookAdmin(admin.ModelAdmin):  
 # fields = ['title', 'description']  
 list\_display = ['title', 'price']  
 list\_filter = ['published']  
 search\_fields = ['title']  
  
admin.site.register(BookNumber)  
admin.site.register(Character)

we can now create new character and can assign them on a specific book.

So we have added a new serializer:

from rest\_framework import serializers  
from .models import Book, BookNumber, Character  
  
  
class BookNumberSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = BookNumber  
 fields = ['id', 'isbn\_10','isbn\_13']  
  
  
class CharacterSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Character  
 fields = ['id', 'name']  
  
class BookSerializer(serializers.ModelSerializer):  
  
 number = BookNumberSerializer(many=False)  
 characters = CharacterSerializer(many=True)  
 class Meta:  
 model = Book  
 fields = ['id', 'title','description', 'price', 'published',  
 'is\_published', 'number', 'characters']

then add related\_name inside of models.py:

from django.db import models  
  
# Create your models here.  
  
  
#something extra information related to that particular book  
class BookNumber(models.Model):  
 isbn\_10 = models.CharField(max\_length=10, blank=True)  
 isbn\_13 = models.CharField(max\_length=13, blank=True)  
  
  
class Book(models.Model):  
  
 # STATUSES = (  
 # (0, 'Unknown'),  
 # (1, 'processed'),  
 # (2, 'paid')  
 # )  
  
 # title = models.CharField(null=True, blank=False, unique=True,  
 # default='', choices=STATUSES)  
  
 title = models.CharField(max\_length=36, blank=False, unique=True)  
 description = models.TextField(max\_length=256, blank=True)  
  
 # price = models.FloatField(default=0, max\_digits=0, decimal\_places=2)  
 price = models.DecimalField(default=0, max\_digits=4, decimal\_places=2)  
 # price = models.BigIntegerField(default=0)  
  
 # published = models.DateField(auto\_now=True, auto\_now\_add=True)  
 # published = models.TimeField(auto\_now=True, auto\_now\_add=True)  
 # published = models.DateTimeField(auto\_now=True, auto\_now\_add=True)  
 published = models.DateField(blank=True, null=True, default=None)  
 is\_published = models.BooleanField(default=False)  
  
 # cover = models.FileField(upload\_to='covers')  
 cover = models.ImageField(upload\_to='covers', blank=True)  
  
 # To integrate our new field  
  
 number = models.OneToOneField(BookNumber, null=True,  
 blank=True, on\_delete=models.CASCADE)  
  
  
  
 def \_\_str\_\_(self):  
 return self.title  
  
class Character(models.Model):  
 name = models.CharField(max\_length=30)  
 book = models.ForeignKey(Book, on\_delete=models.CASCADE,  
 related\_name='characters')

remember the order.

Finally we got an array because we have specify the serializer

\*\*\*\* Remember model book before the character model. For referencing one to many relationship which is foreign key relationship.

characters = CharacterSerializer(many=True)

 **many=False** is used when you expect a **single** related object.

 It is common for relationships like **One-to-One** or **ForeignKey**, where only one instance is related to the main object.

Just 1 to 1 and have only one object

Just 1 to many and having array object for many

v-24 (relationship in databases Many to Many)

models.py

class Author(models.Model):  
 name = models.CharField(max\_length=30)  
 surname = models.CharField(max\_length=30)  
 books = models.ManyToManyField(Book)

py manage.py makemigrations

py manage.py migrate

now we can set author for all book inside of admin panel.

Lets set it into the serializers.py

from rest\_framework import serializers  
from .models import Book, BookNumber, Character, Author  
  
  
class BookNumberSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = BookNumber  
 fields = ['id', 'isbn\_10','isbn\_13']  
  
  
class CharacterSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Character  
 fields = ['id', 'name']

class AuthorSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Author  
 fields = ['id', 'name', 'surname']  
  
class BookSerializer(serializers.ModelSerializer):  
  
 number = BookNumberSerializer(many=False)  
 characters = CharacterSerializer(many=True)  
 authors = AuthorSerializer(many=True)  
 class Meta:  
 model = Book  
 fields = ['id', 'title','description', 'price', 'published',  
 'is\_published', 'number', 'characters', 'authors']

then if we hit in the postman:

<http://127.0.0.1:8000/demo/books/>

we will get eerrror to solve this add related\_name = “authors” inside of author model

models.py

class Author(models.Model):  
 name = models.CharField(max\_length=30)  
 surname = models.CharField(max\_length=30)  
 books = models.ManyToManyField(Book, related\_name='authors')

Important note:

When start an applicate its important to know :

What should store it? what models? how would you use it? and what will be the relationship between them?

Its depending on you how you would like to add the fields and how you want to structure.

# You don’t have to one serialized per model

So if we want to create another minibook model and we specify 2 fields there:

serializers.py

class BookminiSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Book  
 fields = ['id', 'title']

add this for getting all the:

information from Book model:

from rest\_framework.response import Response

class BookViewSet(viewsets.ModelViewSet):  
 # serializer\_class = BookSerializer  
 serializer\_class = BookminiSerializer  
 queryset = Book.objects.all()  
 authentication\_classes = (TokenAuthentication,)  
 permission\_classes = (IsAuthenticated,)  
  
 def retrieve(self, request, \*args, \*\*kwargs):  
 instance = self.get\_object()  
 serializer = BookSerializer(instance)  
 return Response(serializer.data)

just press ctrl and click on ModelViewSet you will get :

def retrieve(self, request, \*args, \*\*kwargs):  
 instance = self.get\_object()  
 serializer = BookSerializer(instance)  
 return Response(serializer.data)

this piece of script and just replate serializer to BookSerializer

then run postman and hit the books url.

<http://127.0.0.1:8000/demo/books/1/>

you will get all the single information the database.

We can have more than one serializers for one class

We may also decide how many field we want

For default and certain one

Thank you now lets move on the first project