

pip install djangorestframework

django-admin startproject DjangoCDAC # create project

Add rest\_framework in **settings.py** INSTALLED\_APPS list

INSTALLED\_APPS = [

...

# Django REST framework

'rest\_framework',

]

To add application to our project do

cd DjangoCDAC

python manage.py startapp hydcdac # this will create a app named hydcdac

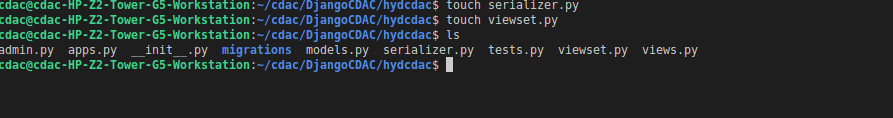
Ls

admin.py apps.py \_\_init\_\_.py migrations models.py tests.py views.py

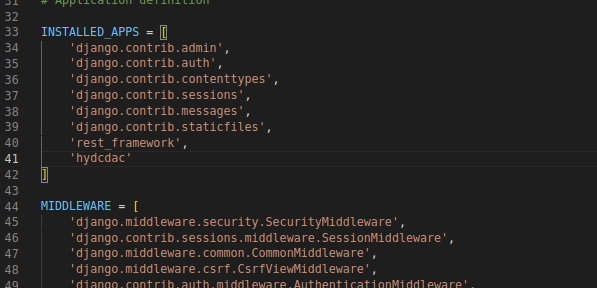
These all files are created.

We need additional file serializer.py and viewset.py

Do touch serializer.py and touch viewset.py



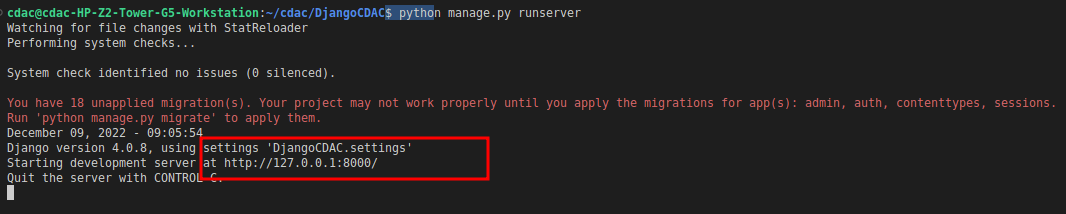
As we created our application named hydcdac we need to add this application name in settings.py present in our project DjangoCDAC



Now lets check of our project is running or not

Cd ..

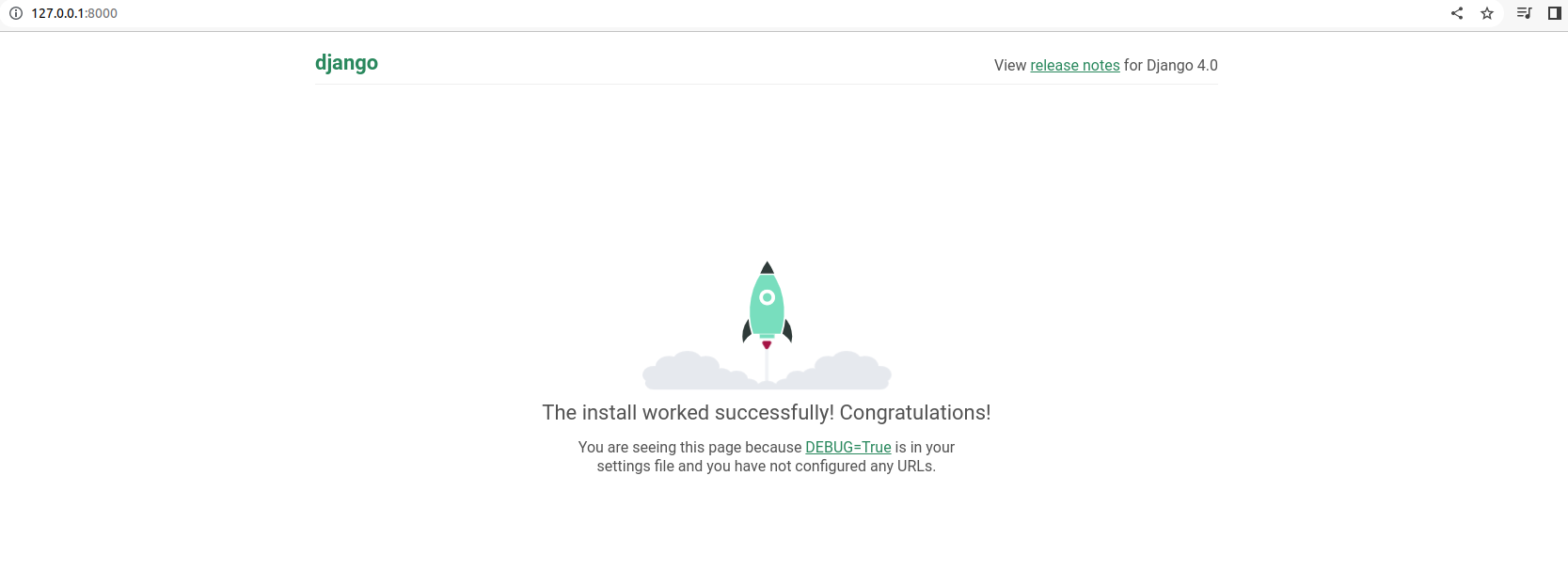
Python manage.py runserver



Our application is running on localhost as shown in the figure

Open browser type

<http://127.0.0.1:8008/>



This means our project is setup

Now we need to add application Config file from our project settings.py

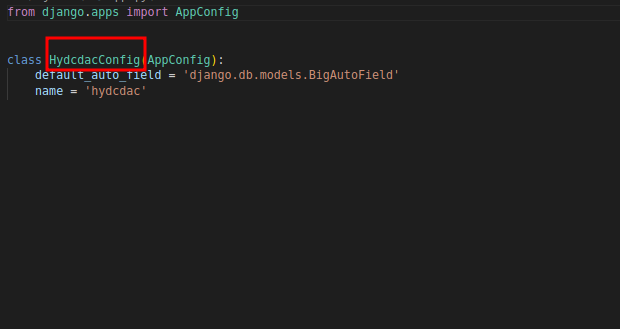
Cd hydcdac

Ls

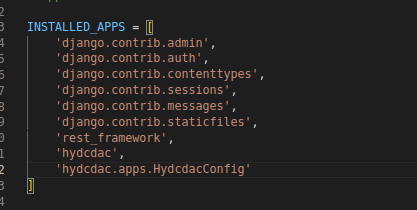


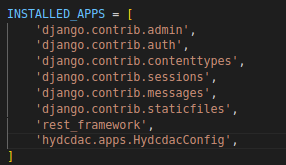
Open app.py file

By default some configurations are present



We need to add HydcdacConfig class to our settings.py installed\_apps list





## **Configure CORS and middleware**

We need to allow requests to our Django application from other origins.

In this example, we’re gonna configure CORS to accept requests from 127.0.0.1:8080.

First, install the *django-cors-headers* library:

pip install django-cors-headers

In *settings.py*, add configuration for CORS:

INSTALLED\_APPS = [

...

# CORS

'corsheaders',

]

You also need to add a middleware class to listen in on responses:

MIDDLEWARE = [

...

# CORS

'corsheaders.middleware.CorsMiddleware',

'django.middleware.common.CommonMiddleware',

]

## **Setup Mongodb database configurations**

pip install pymongo

open *settings.py* and change declaration of DATABASES

DATABASES = {

'default': {

'ENGINE': ''

}

}

from pymongo import MongoClient

def get\_database():

CONNECTION\_STRING = "mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.6.0"

# Create a connection using MongoClient. You can import MongoClient or use pymongo.MongoClient

client = MongoClient(CONNECTION\_STRING)

# Create the database for our example (we will use the same database throughout the tutorial

return client['CDAC\_db']

db=get\_database()

## **Define the Django Model for hydCDAC application**

Add the following in DjangoCDAC/hydcdac/models.py file

from django.db import models

from pygments.lexers import get\_all\_lexers

from pygments.styles import get\_all\_styles

# Create your models here.

class HydCdac(models.Model):

joining\_date = models.DateTimeField(auto\_now\_add=True)

employee\_name = models.CharField(max\_length=100, blank=True, default='')

age = models.TextField(max\_length=3,default=0)

place = models.CharField(max\_length=100, blank=True, default='')

class Meta:

ordering = [‘joining\_date’]

## **Migrate Data Model to MongoDB database**

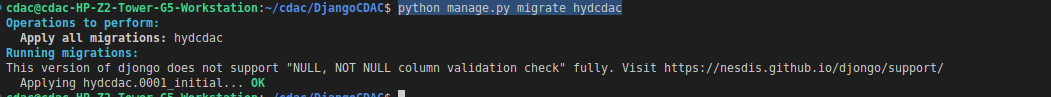
python manage.py makemigrations hydcdac



As shown in the pic createmodel HydCdac create a model

Still no database is not created to create database we need to add this command

python manage.py migrate hydcdac

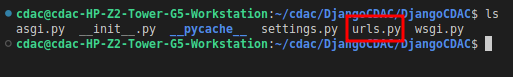


Now we need to API’s in our Django framework

Now lets add urls from our Project to application

Open urls.py present in

/DjangoCDAC/urls.py



Add the path and import include as shown below in our project urls not application urls

"""DjangoCDAC URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:

https://docs.djangoproject.com/en/4.0/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('hydcdac/',include(('hydcdac.urls',"hydcdac")))

]

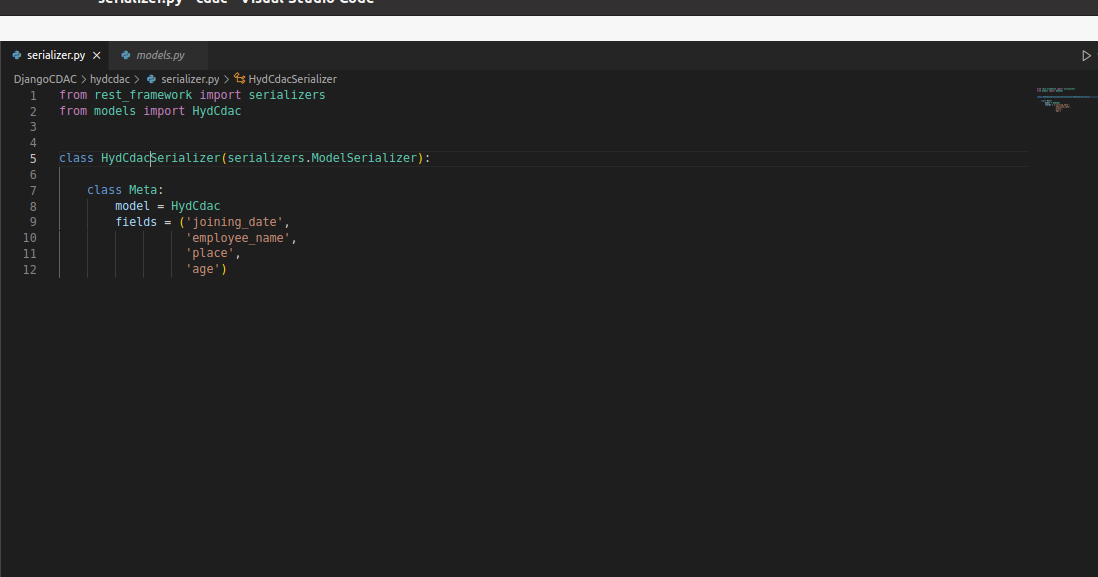
## **Create Serializer class for Data Model**

Serializer class will manage serialization and deserialization from JSON.

rest\_framework.serializers.ModelSerializer superclass which automatically populates a set of fields and default validators. We need to specify the model class here.

Serializer class should be made in the location

DjangoCDAC/hydcdac/serializer.py

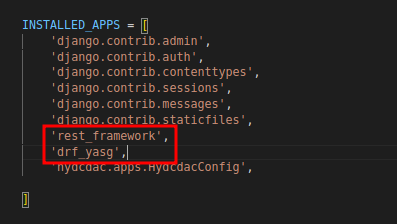


In the inner class Meta, we declare 2 attributes:

* model: the model for Serializer
* fields: a tuple of field names to be included in the serialization

## **Swagger Setup**

Add ‘rest\_framework’,’drf\_yasg’ in installed apps in settings.py file



Next,

Open Djangocdac/urls.py file

from rest\_framework import permissions

from drf\_yasg.views import get\_schema\_view

from drf\_yasg import openapi

from django.urls import path, include

schema\_view = get\_schema\_view(

openapi.Info(

title="CDAC API",

default\_version='v1',

description="Chase your Dreams at CDAC",

terms\_of\_service="https://www.google.com/policies/terms/",

contact=openapi.Contact(email="contact@cdac.local"),

license=openapi.License(name="BSD License"),

),

public=True,

permission\_classes=[permissions.AllowAny],

)

urlpatterns = [

path('swagger/schema/', schema\_view.with\_ui('swagger', cache\_timeout=0), name="swagger-schema"),

path(r'^swagger(?P<format>\.json|\.yaml)$', schema\_view.without\_ui(cache\_timeout=0), name='schema-json'),

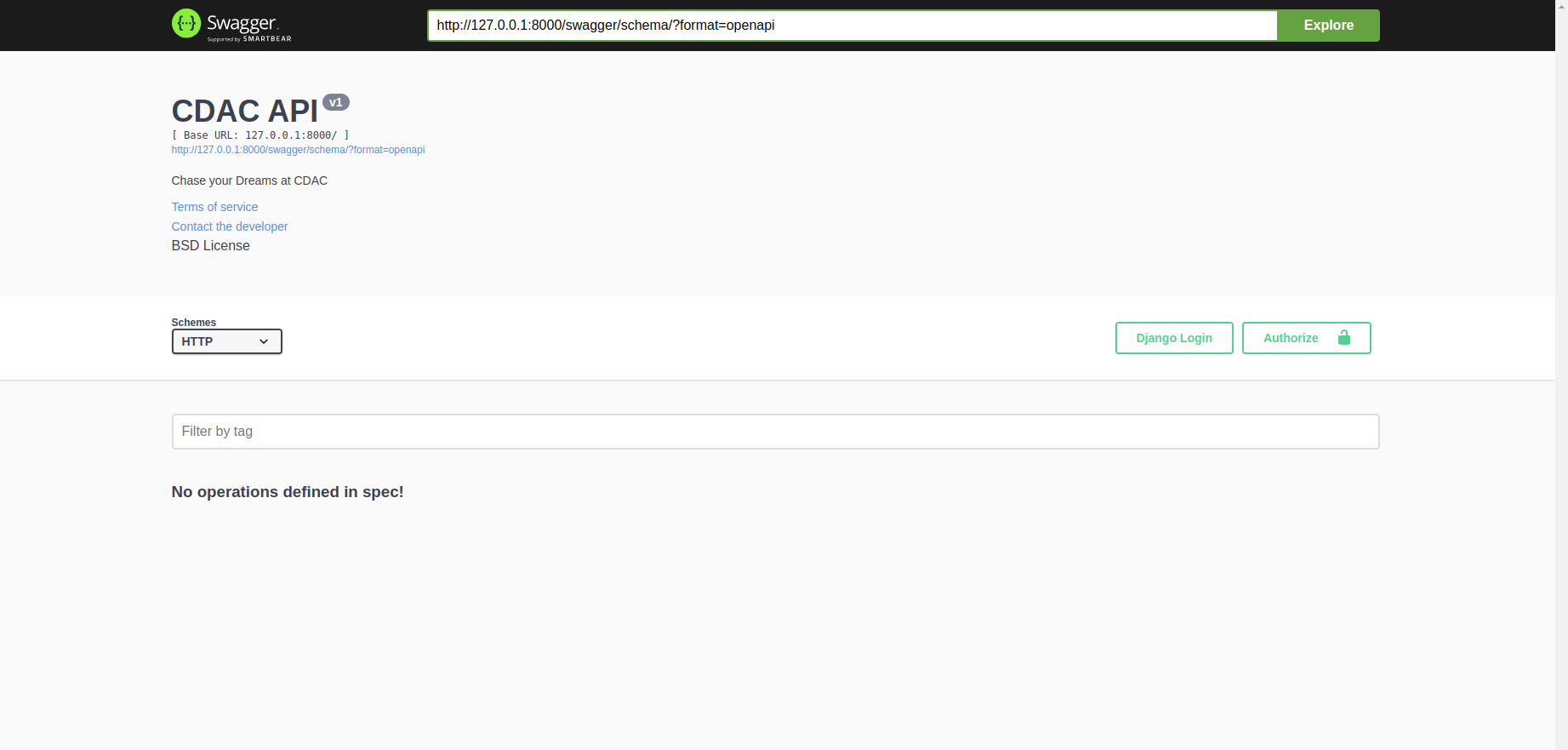
path(r'^swagger/$', schema\_view.with\_ui('swagger', cache\_timeout=0), name='schema-swagger-ui'),

path('swagger/schema/', schema\_view.with\_ui('swagger', cache\_timeout=0), name="swagger-schema"),

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

path('hydcdac/',include(('hydcdac.urls',"hydcdac")))

]

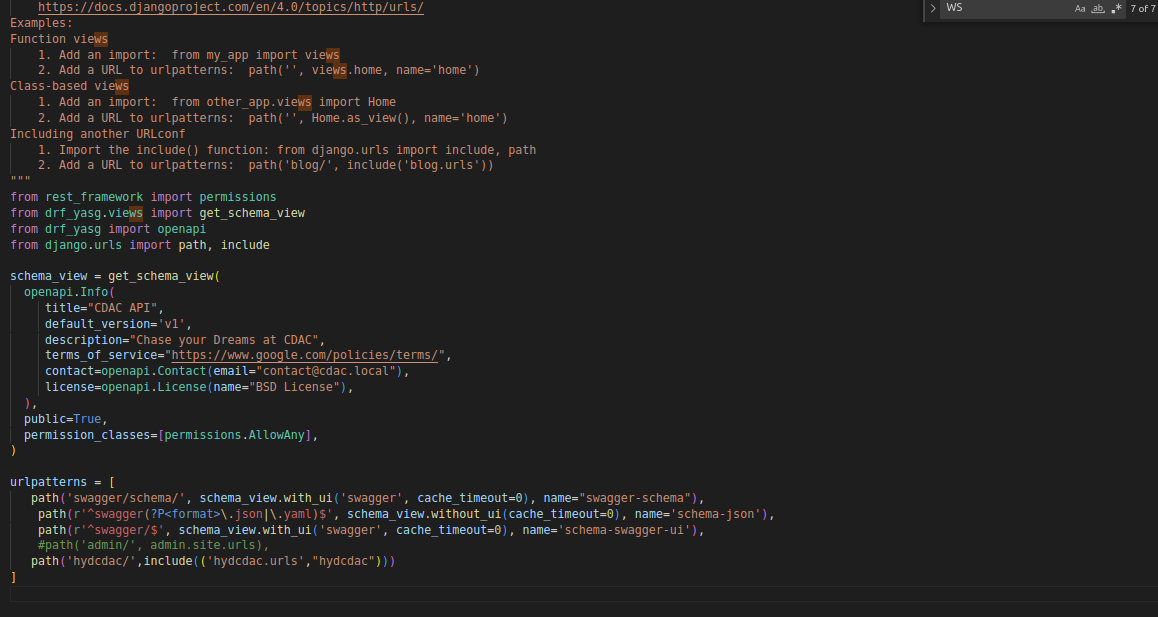


Now lets create a simple GET request to fetch data from a script (without any database connection)

First we need the root of the project to include application hydcdac urls so for that,

Add url pattern in Djangocdac/urls.py

path('hydcdac/',include(('hydcdac.urls',"hydcdac")))



Now open hydcdac/urls.py(note this file does not exit we need to create)

from django.urls import path

from rest\_framework.urlpatterns import format\_suffix\_patterns

from .views import GetDatabase

urlpatterns = [

path('db\_data\_fetch/<str:packagename>',GetDatabase.as\_view(),name="database\_fetch"),

]

urlpatterns = format\_suffix\_patterns(urlpatterns)

hydcdc/views.py

from django.shortcuts import render

# Create your views here.

from rest\_framework import status

from django.http import Http404

from rest\_framework.views import APIView

from rest\_framework.response import Response

import json

import rest\_framework

from .dbrelateddata import package\_name\_extract

class GetDatabase(APIView):

def get(self, \*args, \*\*kwargs):

get\_resp={"status":200}

get\_pack=self.kwargs.get("packagename")

get\_resp["data" ]= package\_name\_extract(get\_pack)

print(get\_resp["data"])

return Response(get\_resp, status=status.HTTP\_200\_OK)

Add dbreleateddata.py in .hydcdac folder

import subprocess

from subprocess import call

import os

import sqlite3

import json

import shutil

def package\_name\_extract(packagename):

cmd=["adb","shell"]

cmd\_root=["adb","root"]

sp=subprocess.Popen(cmd\_root,shell=False, stdout=subprocess.PIPE, stderr=subprocess.PIPE, universal\_newlines=True)

out,err=sp.communicate()

if len(err)==0:

cmd\_db="adb pull data/data/"+packagename+"/databases"

call=subprocess.Popen(cmd\_db,shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)

exit\_code=call.wait()

out,err=call.communicate()

print(out)

if exit\_code==0:

get\_resp=read\_db\_files()

return get\_resp

else:

return("No Database file found in the application")

def read\_db\_files():

extension=['.db','.sqlite']

get\_dir=os.getcwd()+"/databases/"

for file\_name in os.listdir(get\_dir):

f=os.path.join(get\_dir,file\_name)

for get\_ext in extension:

if f.endswith(get\_ext):

try:

#importdb(f)

list\_tablenames=[]

try:

conn = sqlite3.connect(f)

cursor = conn.cursor()

sql\_query = """SELECT name FROM sqlite\_master WHERE type='table';"""

cursor.execute(sql\_query)

list\_tablenames=cursor.fetchall()

test = extract\_table\_info(list\_tablenames,cursor)

except:

print("exception found")

except:

print("Database folder is empty")

try:

shutil.rmtree(get\_dir, ignore\_errors=True)

except OSError as error:

print("cannot remove")

return test

'''def importdb(db):

list\_tablenames=[]

try:

conn = sqlite3.connect(db)

cursor = conn.cursor()

sql\_query = """SELECT name FROM sqlite\_master WHERE type='table';"""

cursor.execute(sql\_query)

list\_tablenames=cursor.fetchall()

extract\_table\_info(list\_tablenames,cursor)

except:

print("exeption found")

'''

def extract\_table\_info(list\_tablenames,cursor):

store={}

list\_db=[]

try:

for tab in list\_tablenames:

store[tab[0]]={}

print("\n\n<<]------"+ tab[0] +"------>>\n\n")

for row in cursor.execute('SELECT \* FROM '+tab[0]+';'):

list\_db.append(row)

store[tab[0]]=list\_db

finally:

cursor.close()

print(store)

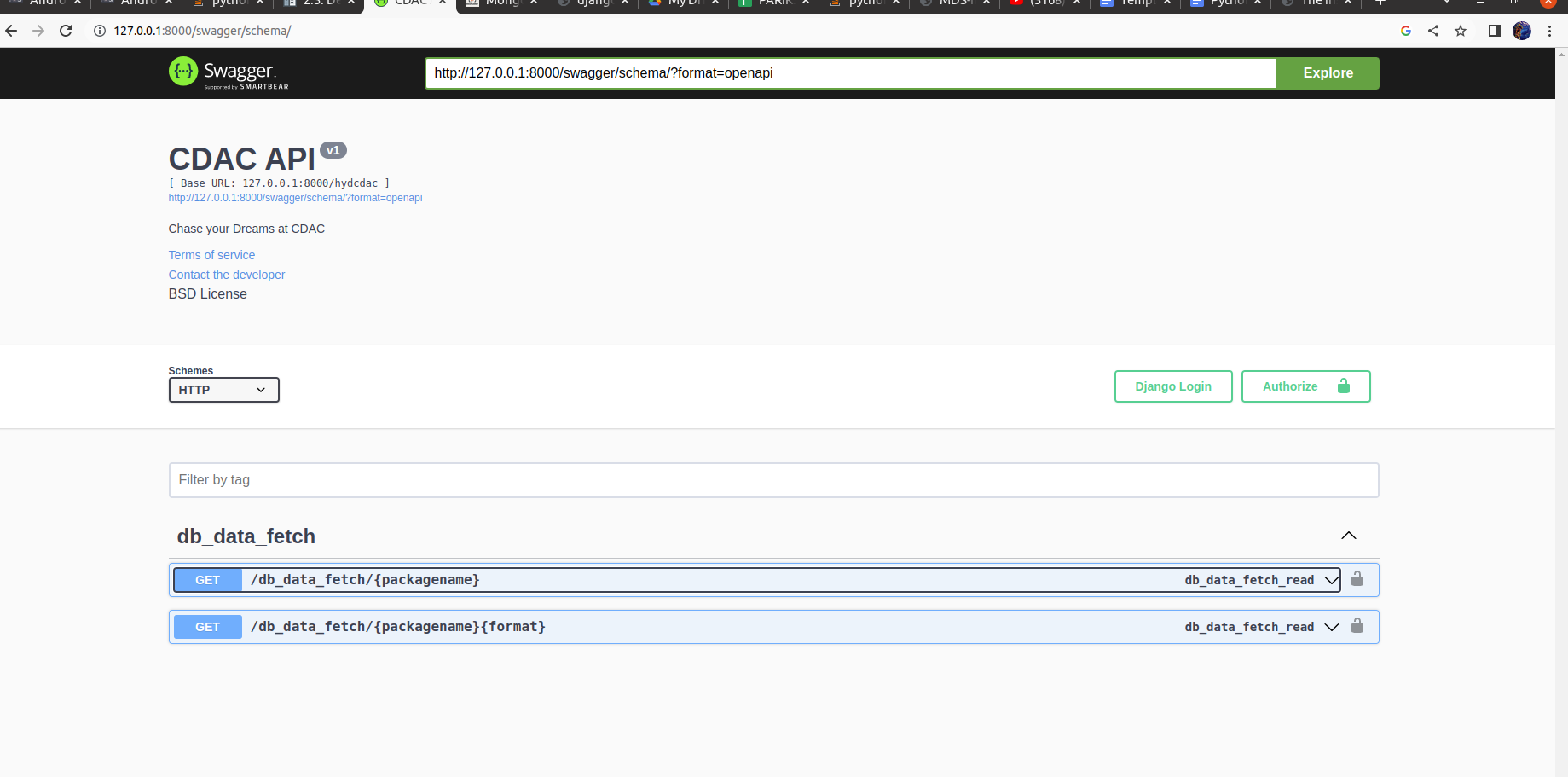
return(store)

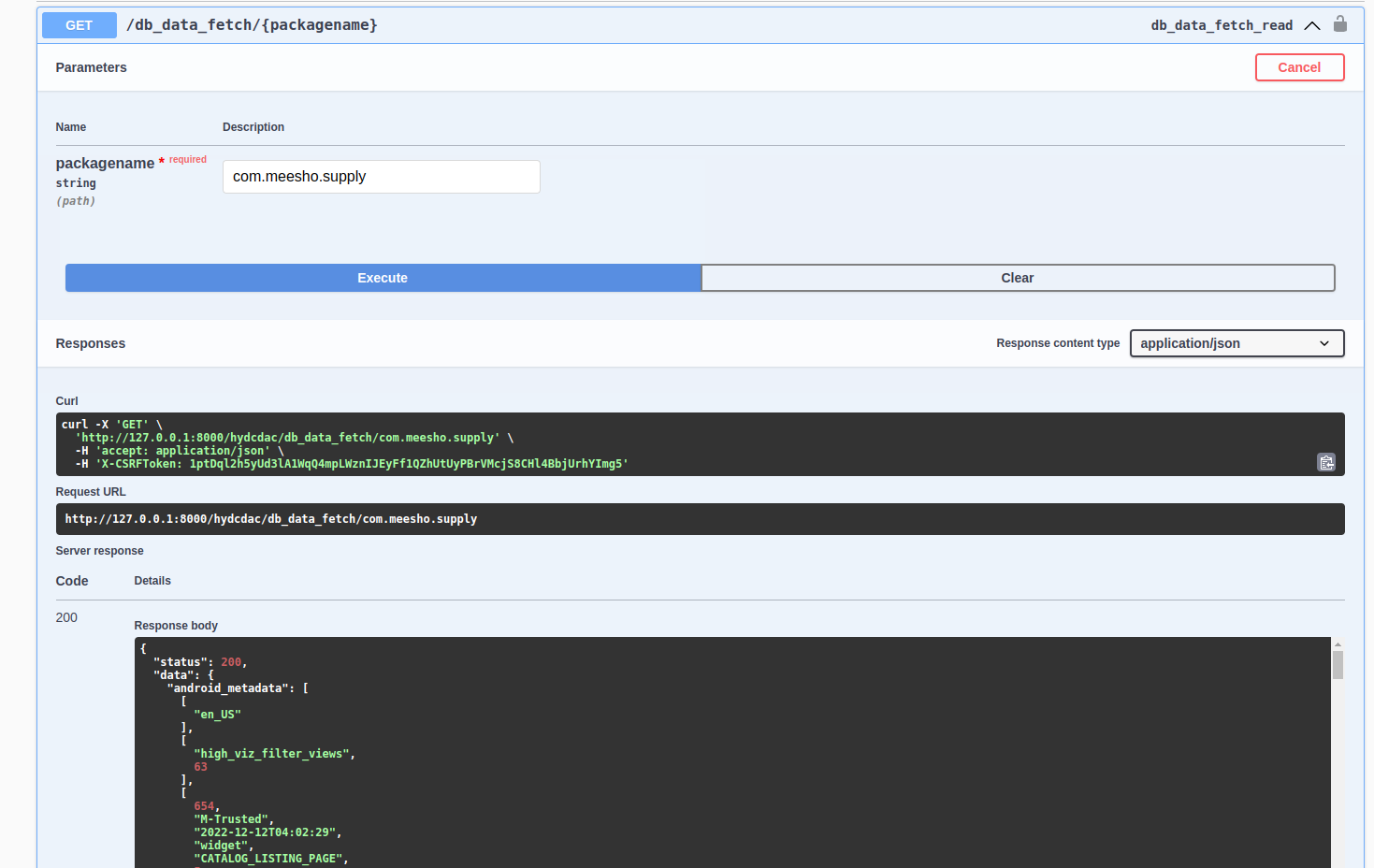
Now run server

Python manage.py makemigrations

Python manage.py migrate

Python manage.py runserver





**http://127.0.0.1:8000/hydcdac/db\_data\_fetch/com.meesho.supply**

This url can be used to fetch data

Add this to hydcdac/viewset.py

from rest\_framework.viewsets import ModelViewSet,ViewSet

from hydcdac.models import HydCdac

from hydcdac.serializer import HydCdacSerializer

class HydcdacViewset(ModelViewSet):

queryset = HydCdac.objects.all()

serializer\_class= HydCdacSerializer

***ModelViewSet is used to define the parameters in the request form at swagger UI***

Now lets include this to our urls.py file

from django.urls import path

from rest\_framework.urlpatterns import format\_suffix\_patterns

from .views import GetDatabase

from rest\_framework.routers import SimpleRouter

from hydcdac import viewset

router = SimpleRouter()

router.register(r'cdachyd', viewset.HydcdacViewset, basename='')

urlpatterns = [

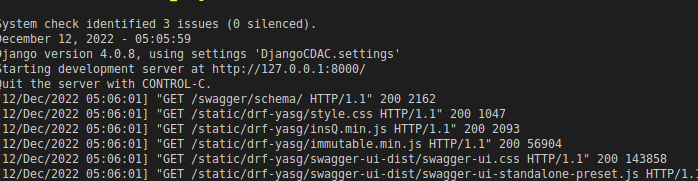
path('db\_data\_fetch/<str:packagename>',GetDatabase.as\_view(),name="database\_fetch"),

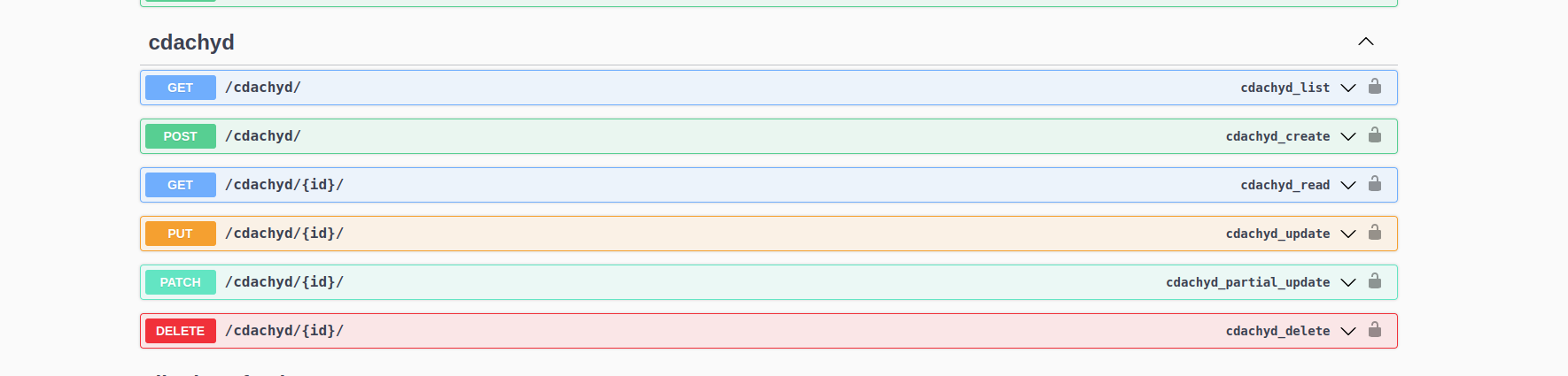
]+router.get\_urls()

urlpatterns = format\_suffix\_patterns(urlpatterns)

As shown above router will create the default apis get,post,put,patch which are created automatically with the help of router

Now run the project python manage.py runserver





Following 6 apis are created with the help of router .Router by default creates 6 API’s.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Creating API’s not by automation routers in our personalized way*

To define our required type of method we need to declare in urls.py of our application

emp\_add= viewset.Hydcdacup.as\_view({

'put': 'update',

'post':'create',

'get': 'list'

})

Add the path in urlpatterns

urlpatterns = [

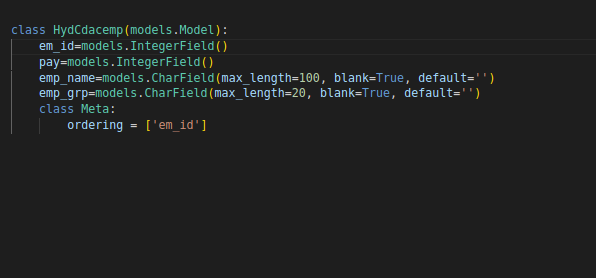
path('emp\_add',emp\_add),

#path('/snippet\_list',snippet\_list)

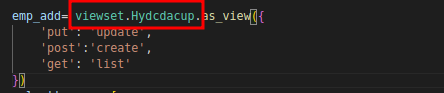
]

Since we want to make changes in our database, by default collection is created by the name of our app name and fields of that collection are defined in hydcdac/models.py





After defining model go back to hydcdac/urls.py and now we need to define viewset.Hydcdacup viewset



We need to define viewset.Hydcdacup, navigate to hydcdac/viewset.py



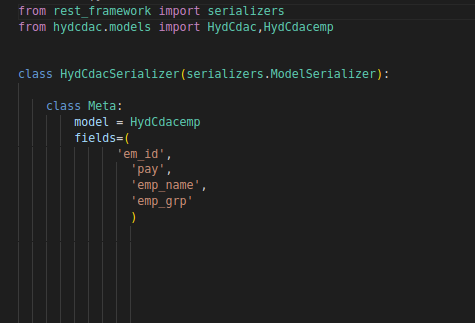
As shown in the fig we have created a class Hydcdacup with 3 methods update,create, list

Now we have defined the functionality and also created required urls.

But we have not defined the parameters required for the api to work.

To define the parameter fields go to serializer.py

Add the following fields



**References:**

[**https://www.django-rest-framework.org/**](https://www.django-rest-framework.org/)

[**https://testdriven.io/blog/drf-views-part-1/**](https://testdriven.io/blog/drf-views-part-1/)

[**https://www.bezkoder.com/django-mongodb-crud-rest-framework/**](https://www.bezkoder.com/django-mongodb-crud-rest-framework/)