*Django Commands*

*django-admin —version #to check version 4.2.4*

*django-admin startproject NAME #to start project*

*python manage.py startapp NAME #to start app*

*python manage.py runserver #to runserver*

*python manage.py shell # to run python shell*

*python manage.py createsuperuser #to give password and username*

*pip freeze > requirements.txt*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Settings file*

-Add app in installed apps(‘Detail.apps.DetailConfig’ or Detail)

-import os

-Templates: os.path.join(BASE\_DIR, ’<Dir where temp stored(main)>’, templates) in DIR

Templates stored in main app or each app then don’t write in settings [BASE\_DIR / 'templates', ],

*Postgres*

-Databases: default:{

ENGINE: django.db.backends.postgresql

NAME: ’<name of database>’

USER: ’postgres’,

PASSWORD: ‘1234’,

HOST: ‘localhost’, or 127.0.0.1 : 8000

PORT: ‘5432’}

*To add static n media files*

-Static Files: STATICFILES\_DIR= os.path.join(BASE\_DIR,’<Dir where temp stored(main)>/ static’)

: STATIC\_ROOT= os.path.join(BASE\_DIR,’static’)

: STATIC\_URL=‘/static/’

-Media Files: MEDIA\_ROOT=os.path.join(BASE\_DIR,’media’)

: MEDIA\_URL=‘/media/‘

After adding static files:

*python manage.py collectstatic*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*URLS.py*

**In main urls**

-from django.urls import path, include # to include paths of other app urls

urlpatterns=[

path(‘<path>/’, include( ‘<appname>.urls’ ) ) ]

**In app:**

import from views

urlpatterns=[

path(‘<path>/’, views.home, name=‘<ref name used in templates>’ )]

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

For static files:

**In main urls:**

-from django.conf import settings

-from django.conf.urls.static import static

-urlpatterns=[ ]+static(settings.MEDIA\_URL,document\_root =settings.MEDIA\_ROOT)

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

For APIVIEW:

urlpatterns = [path('zone/', ZoneAPIVIEW.as\_view()),]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Models & Migration:

class Customer(models.Model):

MALE=1

FEMALE=2

FIELDS=[(MALE,'Male'),(FEMALE,'Female')]

name=models.CharField('Customer', max\_length=50)

age=models.IntegerField('Age')

active=models.BooleanField(default=True)

note=models.TextField('Note',null=True)

height=models.FloatField(blank=True)

created\_on=models.DateTimeField(auto\_now\_add=True)

updated=models.DateTimeField(auto\_now=True)

Date=models.DateTimeField()

DOB=models.DateField()

gender=models.CharField(choices=FIELDS, max\_length=10)

home=models.ForeignKey(Home, on\_delete=models.CASCADE)

class Meta:

verbose\_name\_plural='Customer'

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

return self.name

**auto\_now:** updates the value of field to current time, date every time the Model.save() is called.

**auto\_now\_add:** updates the value with the time and date of creation of record.

There is *no difference*. But the version with author\_id will be more efficient than author.id.

To migrate - do models

*python manage.py makemigrations #to make*

*python manage.py migrate #to migrate*

*python manage.py showmigrations #to get all migrations*

By default sqlite is database

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

**migrate**: which is responsible for applying and unapplying migrations.

**makemigrations**: which is responsible for creating new migrations based on the changes you have made to your models.

**Situation**: Migrations of a model name Profile is done and you have made 2 profiles but u want to add more fields to it and then migrate again you get a error how to resolve that error?

1)You add null=True inside the field: passed=models.BooleanField(null=True)

2)You encounter this error:

It is impossible to add a non-nullable field 'active' to profile without specifying a default. This is because the database needs something to populate existing rows.

Please select a fix: 1) Provide a one-off default now (will be set on all existing rows with a null value for this column)

2) Quit and manually define a default value in models.py.

option 1: Enter the same datatype field (this value will be reflected everywhere) then you can change it in admin of django manually

option 2: add default=“value” to make it work.

blank=True for form validation, allowing empty values when validating input from forms.

null=True for database schema, allowing NULL values in the database for the specified field.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*For Serializers:*

class ZoneSerializer(serializers.ModelSerializer):

class Meta:

model=Zone

fields='\_\_all\_\_'

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

Nested:

class PatientCheckSerializer(serializers.ModelSerializer):

patient=PatientSerializer(read\_only=True)

class Meta:

model=DailyCheckup

fields='\_\_all\_\_'

depth=1

class ZoneIDSerializer(serializers.ModelSerializer):

patient = PatientSerializer1()

class Meta:

model = DailyCheckup

exclude = ['doctor']

depth = 2

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

Custom Serializer:

class DailyCheckupSerializer1(serializers.ModelSerializer):

doctor=serializers.SerializerMethodField()

class Meta:

model=DailyCheckup fields=[‘height’,'weight','checkup\_date','doctor']

def get\_doctor(self,obj):

return obj.doctor.name

class GetSerializer(serializers.ModelSerializer):

age=serializers.SerializerMethodField()

daily\_checkups=serializers.SerializerMethodField()

class Meta:

model=Patient

fields=['name','hospital','gender','age']

def get\_age(self, obj):

yr=str(obj.dob)[:4]

current\_yr=datetime.datetime.now().year

year=int(yr)

return (current\_yr-year)

def get\_daily\_checkups(self,obj):

obj1=DailyCheckup.objects.filter(patient=obj)

if obj1:

serializer=DailyCheckupSerializer1(obj1, many=True)

return serializer.data

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Views :

Function based: def home(self, request, \*args, \*\*kargs):

return “HIII”

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

@api\_view decorator:

@api\_view(['POST'])

def login(request):

serializer=LoginSerializer(data=request.data)

if serializer.is\_valid():

data=serializer.data

return Response(data)

return Response(serializer.errors)

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

Class based:

Viewsets:

from rest\_framework.routers import DefaultRouter

router=DefaultRouter()

router.register(r’accounts’,AccountViewSet,basename=‘accounts') #whatever u register

urlpatterns=[path(‘account/‘,include(router.urls))] #so url is account/accounts

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

APIView:

class ZoneAPIVIEW(APIView):

def get(self, request):

obj = Zone.objects.all()

serializer = ZoneSerializer(obj, many=True)

return Response(serializer.data, status.HTTP\_200\_OK)

def post(self, request):

obj = request.data

serializer = ZoneSerializer(data=obj)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status.HTTP\_200\_OK)

else:

return Response(serializer.errors)

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

Logic

patient\_unique\_id = data.get('patient', {}).get('patient\_unique\_id')

obj = DailyCheckup.objects.get(patient\_\_patient\_unique\_id=patient\_unique\_id)

serializer = PatientCheckSerializer(obj, data=data, partial=True)

if serializer.is\_valid():

serializer.save(patient=obj.patient, doctor=obj.doctor)

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

data=request.data

patient\_data=data.get(‘patient',{})

obj=patient\_data[‘hospital’]

patient\_obj=Patient.objects.create(patient\_unique\_id=patient\_data['patient\_unique\_id'], hospital=Hospital.objects.get(id=obj),

name= patient\_data[‘name'], patient\_data['created\_on'],)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*For admin:*

class PatientAdmin(admin.ModelAdmin):

search\_fields=('name',)

list\_filter=('gender','active','hospital')

list\_display=["patient\_unique\_id","dob","city","active"]

admin.site.register(Patient, PatientAdmin)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Fixtures:*

—a convenient way to manage and load sample or initial data into your Django application’s database.

—critical role in testing Django applications.

—It can be used to migrate data between different database systems or versions

—used as a lightweight backup mechanism for your Django application’s data.

—to load initial data, developers can quickly and easily set up their development environment

—not tied to any specific database system.

—JSON: JavaScript Object Notation, XML: Extensible Markup Language, YAML(initial data)

[{ "model":"product.Product",

"pk":6,

"fields":{"title":"Product 5",

"content":"New brand",

"price":90 }}]

*python manage.py dumpdata appname.modelname —output=filename.json*

*python manage.py dumpdata product.Product --indent 2 --output abc.json*

*python manage.py loaddata abc.json*

It gets generated in the main block with the manage.py file all model data gets printed in that file

* --format: Specifies the serialization format (e.g., --format=xml).
* --indent: Controls the indentation level for the output (e.g., --indent 4).
* --exclude: Excludes specific models or fields from the output (e.g., —exclude = app\_name.ModelName or --exclude=app\_name.ModelName.field\_name).
* --natural-primary and --natural-foreign: Use natural primary / foreign keys, if available, instead of primary key references.

Test file if fixture is loaded properly

from django.test import TestCase

from product.models import Product

class Pro(TestCase):

fixtures=[‘abc’]

def text\_fixture\_data\_loaded(self):

count=Product.objects.count()

self.assertEqual(count,2)

*python manage.py test*

Found 0 test(s).

System check identified no issues (0 silenced).

-----------------------------------------------

Ran 0 tests in 0.000s

OK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Signals:*

from django.db.models.signals import pre\_save, post\_save, pre\_delete, post\_delete

**In Apps file:**

def ready(self):

import product.signals

**In Signals file:**

#gives both previous and updated instance

@receiver(pre\_save, sender=Leave)

def print\_data(sender, instance, \*\*kwargs):

print(sender.objects.get(id=instance.id)) #previous instance #pending

print(instance) #approved

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Transactions:*

A **commit** operation signifies that the transaction is successful and all changes made within the transaction are permanently saved to the database.

**Rollback** operation discards any changes made within the transaction and reverts the database to its state before the transaction begins.

*ACID*

**Atomicity** -a database transaction is treated as a single, indivisible unit of work. Either the entire **transaction** is completed successfully/ entirely rolled back to its previous state if any part fails.

**Consistency**-data should be consistent

**Isolated**- all transactions should be isolated. No transaction should affect the other

**Durable**-Once transaction is done it will remain in the system even if the system crashes

from django.db import transaction

with transaction.atomic():

new=Treasure(name="Dragon Chest", value=100)

new.save()

savept\_id=transaction.savepoint() #raise Exception with no savepoint nothing gets saved

NoTres=get\_object\_or\_404(Treasure, id=99) #False hence exception

NoTres.name="Gem"

NoTres.save()

return HttpResponse("Quest failed")

except Exception as e:

transaction.savepoint\_rollback( savept\_id) #that creation is done n saved at savepoint

return HttpResponse(f"Quest failed {e}") #while other things are rollbacked

**raise Exception** even if it is present(in the middle) due to transaction the entire process will rollback till the entire transaction is complete. if savepoints then it is different

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

For loggers:

LOGGING = {

'version': 1,

'disable\_existing\_loggers': False,

'handlers': {

'console': {

'class': 'logging.StreamHandler',

},},

'root': {

'handlers': ['console'],

'level': 'INFO',

},}

In views : import logging

logger = logging.getLogger(\_\_name\_\_)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Django Manger:*

To overwrite or add more methods

class MyCustomManager(models.Manager):

def search(self, query):

return MyModel.objects.filter(title=query)

class MyModel(models.Model):

man = MyCustomManager()

res=MyModel. man.search()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*QuerySets:*

*ManyToMany, reverse Foreign:*

City.objects.prefetch\_related('state')

<QuerySet [<City: Margao>, <City: Gangtok>, <City: Unakoti>]>

OnetoOne, Foreignkey

City.objects.select\_related('state')

<QuerySet [<City: Margao>, <City: Gangtok>, <City: Unakoti>]>

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

*Annotate*

>>> from django.db.models import Count

>>> City.objects.annotate(Num=Count('state'))

<QuerySet [<City: Margao>, <City: Gangtok>]>

>>> obj=City.objects.annotate(Num=Count('state'))

>>> obj[1].Num

>>> 1

>>> A=City.objects.values('name').annotate(Count('state'))

>>> A

<QuerySet [{'name': 'Agra', 'state\_\_count': 2}, {'name': 'Agra1a', 'state\_\_count': 1}, {'name': 'Agra', 'state\_\_count': 1}]>

C=DailyCheckup.objects.annotate(Tot=Count(F(‘patient'))+ Count(F(‘doctor')))

>>> C[0].Tot

2

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

*Case Keyword*- used for conditional situations

obj= MyModel.objects.annotate(New=Case( When(field\_name='value1', then=Value(1)),

When(field\_name='value2', then=Value(2)), default=Value(0), output\_field=IntegerField(),))

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

from django.db.models.functions import Cast #forces the object to change its data type

A=Model.objects.annotate(field=Cast(‘age’, output\_field = FloatField())).get()

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

Coalesce:

SELECT COALESCE(column1, column2, 'Default') AS result FROM my\_table;

In this example, if column1 is not null, it will be returned. Otherwise, if column2 is not null, it will be returned. If both are null, the string 'Default' will be returned.

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

from django.db.models import Q

obj= MyModel.objects.filter(field\_\_in=[1, 2, 3])

obj= MyModel.objects.filter(field\_\_range=(10, 20))

# Exclude empty strings

obj= MyModel.objects.exclude(field\_name=‘')

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Celery & RabbitMQ:

**In celery.py(main project):**

from \_\_future\_\_ import absolute\_import, unicode\_literals

import os

from celery import Celery

os.environ.setdefault(‘DJANGO\_SETTINGS\_MODULE', ’<name\_of\_project>.settings')

app=Celery('name\_of\_project')

app.config\_from\_object('django.conf:settings',namespace='CELERY')

app.autodiscover\_tasks()

**In settings.py:**

Installed\_apps=[ ‘celery']

CELERY\_BROKER\_URL='pyamqp://guest:guest@localhost:5672//'

*celery -A <name\_of\_project> worker —loglevel=INFO*

A **Celery worker** is a process that executes the tasks defined in your Celery application. INFO is one of the logging levels. It specifies that the worker should log messages at the INFO level and above. DEBUG, WARNING, ERROR, and CRITICAL.

**In tasks.py(in apps where you need add tasks):**

from celery import shared\_task

@shared\_task # to define asynchronous tasks.

def add(x, y):

return x + y

**In views file call the task:**

from api.tasks import add

result=add.delay(4,5)

RabbitMQ: (In another Terminal)

*brew install rabbitmq*

*brew services start rabbitmq #to start service*

*rabbitmqcl status*

*rabbitmq-server #To start ….7 plugins…*

*rabbitmqctl stop #to stop all services*

*brew services restart rabbitmq #Restarts the service*

result=add.apply\_async(args=(4,5), priority=10) #Add task is received first due to priority

res=sub.apply\_async(args=(4,5), priority=5)

Task product.tasks.add[c014e0df-acca-450b-9255-799732e5d85e] received

[2023-11-20 05:05:53,280: INFO/MainProcess] Task product.tasks.sub[id] received

Task product.tasks.sub [id] succeeded in 0.00044754200007446343s: -1

Task product.tasks.add[id] succeeded in 0.0007949159999043331s: 9

result=add.apply\_async(args=(4,5), priority=5)

res=sub.apply\_async(args=(4,5), priority=10) #sub task has a higher priority (10), so it should be processed before the add task (5).

from datetime import timedelta

chan=add.apply\_async(args=(4,7),countdown=20) #applies delay

**Tasks** are basically used when the processing of the task takes time or u need to check if the task works properly. These mostly work in the background (celery terminal)

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

Getting email:

def signup(request):

if request.method=='POST':

if User.objects.filter(username=request.POST['username']).exists()

return render(request, "sign.html", {'error': 'Username taken’})

user=User.objects.create\_user(username=request.POST['username'],

password=request.POST['password'],

email=request.POST['email'])

send\_welcome.apply\_async(args=(user.username, user.email),countdown=10)

login(request, user)

return redirect("home")

return render(request, “sign.html")

**tasks.py file:**

@shared\_task

def send\_welcome(username, email):

subject='Welcome to My site'

message=f"Hello {username}, Thank you for your time"

email\_from=settings.EMAIL\_HOST\_USER

recipient\_list=[email,]

send\_mail(subject, message, email\_from, recipient\_list)

return f"Mail is sent to {username} on {email}!”

**settings.py:**

EMAIL\_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL\_HOST = 'smtp.gmail.com'

EMAIL\_PORT = 587

EMAIL\_USE\_TLS = True

EMAIL\_HOST\_USER = 'email'

EMAIL\_HOST\_PASSWORD = ‘app password'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Apps.py page

**ready(self)**:This method is called as soon as the app is imported.

def ready(self): import product.signals

**default\_auto\_field(self)**:It allows you to specify the default auto field to use for models in the app.  
 default\_auto\_field = 'django.db.models.BigAutoField'  
  
**\*\*create(kwargs):** This method is called when the app is created.

def create(self, \*\*kwargs): pass

**create\_template\_tags**(self, tag\_module, template\_dirs): Used to create template tags for the app.

def create\_template\_tags(self, tag\_module, template\_dirs): pass

**create\_template\_postprocesso**r(self, post\_processor, template\_dirs): Used to create a template post processor for the app.

def create\_template\_postprocessor(self, post\_processor, template\_dirs):

**get\_model**(self, model\_name, require\_ready=True): It returns the model with the given name/None if the model is not found.

def get\_model(self, model\_name, require\_ready=True):

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Swagger:

For DRF versions before 3.5.3:

*pip install django-rest-swagger*

**settings file:** INSTALLED\_APPS = [ 'rest\_framework\_swagger']

**urls.py:**

from django.conf.urls import url

from rest\_framework\_swagger.views import get\_swagger\_view

schema\_view = get\_swagger\_view(title='Project')

urlpatterns = [ url(r'^$', schema\_view)]

Django REST Framework 3.5.3:

**In Urls(main):**

from rest\_framework import permissions

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

from drf\_yasg import openapi

schema\_view=get\_schema\_view(

openapi.Info(

title="Swagger Project",

default\_version='v1',

description='Medical',

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

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

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

),

public=True,

permission\_classes=(permissions.AllowAny,),)

urlpatterns = [

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

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

]

**In Settings:**

Installed apps=['drf\_yasg']

REST\_FRAMEWORK={

'DEFAULT\_SCHEMA\_CLASS' :’rest\_framework.schemas.coreapi.AutoSchema',}

for post, put in ApiView:

@swagger\_auto\_schema(

request\_body=DailyCheckupSerializer,

responses={status.HTTP\_201\_CREATED:DailyCheckupSerializer}

)

for post, put in @api\_view:

from drf\_yasg.utils import swagger\_auto\_schema

@swagger\_auto\_schema(

methods=['GET'],

responses={200: ProductS()},

operation\_description="Get a random product.")

@swagger\_auto\_schema(

methods=['POST'],

request\_body=ProductS,

responses={200: ProductS()},

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Register, authenticate:*

from django.contrib.auth import authenticate, login, logout

from django.contrib import messages

def register(request):

if request.method=="POST":

if request.POST['username']:

if request.POST['password1']==request.POST['password2']:

try:

user=User.objects.get(username=request.POST['username'])

return render(request,'accounts/register.html',messages.error(request,'User exists'))

except User.DoesNotExist:

user= User.objects.create\_user(name= request.POST[‘name1’] , email=request.POST[‘email'], username=request.POST[‘username'], password=request.POST['password1'])

login(request, user)

messages.success(request,"Registered successfully")

return redirect('login')

else:

return render(request, ’register.html’,messages.error(request,"Both passwords need to be same"))

else:

return render(request,'accounts/register.html',messages.error(request,'Enter all fields'))

else:

return render(request,'accounts/register.html')

def Loginuser(request):

if request.method=="POST":

user=authenticate(username=request.POST['username'], password=request.POST['password'])

if user is not None:

login(request, user)

messages.success(request,"You have logged in")

return redirect('home')

else:

return render(request,'accounts/login.html',messages.error(request,'Error in logging in'))

else:

return render(request,'accounts/login.html')

def Logout\_user(request):

logout(request)

messages.success(request,"You have been logged out")

return redirect('login')

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Pagination:*

from django.core.paginator import Paginator, EmptyPage

def paginate\_query(obj, request):

page = request.GET.get("page", 1)

page\_no = 10

try:

paginator = Paginator(obj, page\_no)

print(paginator.page(page))

except EmptyPage:

return Response({"Message": "Page Not found"}, status.HTTP\_404\_NOT\_FOUND)

except Exception as exc:

return Response({"Message": "Invalid page no”}, status.HTTP\_400\_BAD\_REQUEST)

serializer = StudentSerializer(paginator.page(page), many=True)

return Response(serializer.data, status.HTTP\_200\_OK)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Extra:*

*Models:*

AbstractBaseModel: basemodel (u write things like created\_on, updated\_on fields that can be inherited/common by other models ). Then u inherit it to other models

TimeStampedModel: Similar to above only created\_on, updated\_on fields are written no extra fields.

UUIDModel:to use uuid as the field.id = models.UUIDField(primary\_key=True,

default= uuid.uuid4, editable=False)

**Mixins** are reusable pieces of code that can be mixed into class-based views to add or extend functionality. Mixins allow you to modularize and reuse common patterns in your views.

**\_\_call\_\_** method in a class, instances of that class become callable objects, just like functions.

**data=request.data** when you specifically want to access the parsed data from the request body.

**request = self.context.get('request')** when you need the entire request object, including metadata and headers, providing a more comprehensive view of the request context.

request=self.context.get('request')

print(f"request.method: {request.method}") # request.method: GET

print(f"request.data: {request.data}") #request.data: {}

print(f"request.query\_params: {request.query\_params}") # <QueryDict: {}>

print(f"request.path: {request.path}")#request.path:/product/new/

print(f"request.headers: {request.headers}") #request.headers: {'Content-Length': '','Content-Type': 'text/plain', 'User-Agent': 'PostmanRuntime/7.35.0','Accept': '\*/\*', 'Postman-Token': 'id', 'Host': '127.0.0.1:8000', 'Accept-Encoding': 'gzip, deflate, br', 'Connection': 'keep-alive', 'Cookie': 'csrftoken=token'}

Obj in query sets are lazy they don’t retrieve data until we use them

**serializer.data** when you want to access the serialized representation of the data after it has been processed by the serializer

**hasattr** is a built-in Python function. It is used to check if an object has a given attribute or not. The syntax for hasattr is as follows:

hasattr(object, attribute)

Use serializer.data when you want to access the serialized representation of the data after it has been processed by the serializer

hasattr is a built-in Python function. It is used to check if an object has a given attribute or not. The syntax for hasattr is as follows:

hasattr(object, attribute)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*GIT*

mkdir folder #to make a new folder

rm -rf folder #removes all contents of the folder+filer

rm file.txt #removes file

touch file.txt #creates file

--help

esc + :wq vi (To move out of vi/vim editor, ESC : q ! enter(for no changes)

(HEAD -> main) --Head is the pointer to the main where the commit is done

------\*---------\*-------------\*-----------------\*----------------\*-------------\*------------------\*--------------

Local repo:

git init #initialisation of git on your local repo

git status #gives u the status of the git process

git config --global user.email ".com" #to configure details

git config --global user.name "cc"

branch master / main--top level file

git bash--git terminal

git add file / git add .--(add everything) /git add -A #added to staging area

git rm file + -f(force remove) #remove file from staging area, also removes from main directory

git rm --cached file #removes file only from staging area

git commit -m "new" #committing changes

git log / git log --oneline #all commits are displayed (q to quit)

git checkout (id of the commit found by git log)

#this lets u go back to the previous commit of ur choice(The data also gets reverted in the file.The HEAD pointer will be attached to that commit wen u do git log.

git checkout main #reverts back to the original file

git revert (id of the commit) #goes back one commit and also commits the changes

git reset (id of the commit found by git log)

#soft-goes back in time in the committery (head pointer back, similar to checkout), mixed(mixture with soft n moving back in time in the working directory), hard(moving back in time in the working directory and staying there))

git reset --hard (id of commit)

git checkout -b (branch name) # creating new branch n switching to it

git checkout main #to switch back to main branch

git branch newB2 #create branch

git branch -a # to check all branches. (\*) one is the branch which u r working on

out:\* main

newB

newB2

git checkout (branch name) #to go to that branch

git branch -d (branch name) #to delete branch

git merge newB #merge to main branch

git pull #pulling the remote repo to ur local repo

git pull origin master

git stash pop #pops the top stash off of the stack and merges it with your working environment

git clone [repository\_url]: Clone a repository into a new directory.

git add [file]: Add changes in the working directory to the staging area.

git commit -m "[commit\_message]": Record changes from the staging area to the repository.

git status: Show the status of changes as untracked, modified, or staged.

git diff: Show changes between commits, commit and working tree, etc.

git log: Display the commit history.

git branch: List, create, or delete branches.

git checkout [branch\_name]: Switch branches or restore working tree files.

git merge [branch\_name]: Combine changes from different branches.

git pull: Fetch from and integrate with another repository or a local branch.

git push: Update remote refs along with the associated objects.

git remote -v: Show remote repositories and their URLs.

git fetch [remote\_name]: Download objects and refs from a remote repository.

git remote add [remote\_name] [repository\_url]: Add a remote repository.

git remote remove [remote\_name]: Remove a remote repository.

git rm [file]: Remove files from the working tree and from the index.

git mv [old\_file] [new\_file]: Move or rename a file, a directory, or a symlink.

git reset [commit]: Reset current HEAD to the specified state.

git stash: Stash changes in a dirty working directory away.

git tag [tag\_name]: Create a tag.

git fetch --all: Fetch all remotes.

------\*---------\*-------------\*-----------------\*----------------\*-------------\*------------------\*--------------

fork from diff repo(Dhan)

clone to ur local

make changes

add, commit, push

send pull request to Dhan

she will accept/reject

------\*---------\*-------------\*-----------------\*----------------\*-------------\*------------------\*--------------

git branch newB #creating new branch

git checkout newB #to switch to newB branch

git branch -a #to get all branches on remote n local

Make changes

git add, commit

git push origin main

Make a new branch in remote repo(Github)

git pull #to get all branches

git branch -a

git checkout -b B2 origin/B2 # switches to B2 branch (pulled from remote GitHub)

git branch -a #make changes n then push

git clone -b B2 --single-branch http://git.. #to clone B2 branch in local from remote(Github)

\*---------\*-------------\*-----------------\*----------------\*-------------\*------------------\*--------------

make a python file

clone it another's 2 laptop--> git clone http://...

opened document, made changes on same line

git stash --> whatever changes u want to add

git pull --> the code

git stash pop --> ur changes get, and do necessary changes

git add .

git commit

git push

------\*---------\*-------------\*-----------------\*----------------\*-------------\*------------------\*--------------

WHILE COMMITTING YOUR CODE

git stash

git pull

git stash pop

git add .

git commit -m "message"

git push

TO UPDATE YOUR WORK BRANCH WITH BASE BRANCH

git stash - (if there are any changes in your branch)

git checkout base\_branch

git pull

git checkout work\_branch

git pull

git merge base\_branch

git commit -m "merged base branch into workbranch"

git push

git stash pop ( to get back your changes)

git pull origin base\_branch

To check code changes

git diff .

To restore changes

git restore filepath

If u did a git add and then want to undo it:

git restore --staged filepath

------\*---------\*-------------\*-----------------\*----------------\*-------------\*------------------\*--------------

sudo docker-compose -f kafka-setup.yml up -d #Running docker

docker ps # Docker status check

sudo -u postgres psql # Opening PSQL in terminal

CREATE DATABASE <DB name>; #Creating New DB

CREATE USER trellis WITH PASSWORD 'Trellis123'; # Creating User

GRANT ALL PRIVILEGES ON DATABASE database\_name TO trellis;

#Granting Access

DROP DATABASE <DB name>; #Dropping DB

nano ~/.bashrc #Updating OS variables

source ~/.bashrc && source /home/trellissoft/projects/fdr\_venv/bin/activate

#Updating OS variables in virtual env

------\*---------\*-------------\*-----------------\*----------------\*-------------\*------------------\*--------------

###### Running Flask

flask run -p 5000 --host=0.0.0.0 #Running flask server on virtual IP

flask db migrate # Migrating database

OR

flask db migrate -m "message"

flask db upgrade # Upgrading flask database

flask loaddata --file=currencies # Loading Fixtures

python grpc\_server.py <IP>:<PORT> # Running GRPC Server

faust -A faust\_notification worker -l info # Running Faust Server in Notification service

$ chmod og+X /home /home/(username)