Task 1:

In serializers.py

from rest\_framework import serializers

from .models import UserProfile

class UserProfileSerializer(serializers.ModelSerializer):

class Meta:

model = UserProfile

fields = ['id', 'name', 'email', 'profile\_picture']

In views.py

from rest\_framework import generics, permissions

from .models import UserProfile

from .serializers import UserProfileSerializer

from rest\_framework.pagination import PageNumberPagination

from django\_filters.rest\_framework import DjangoFilterBackend

from rest\_framework.filters import SearchFilter, OrderingFilter

class UserProfilePagination(PageNumberPagination):

page\_size = 10

page\_size\_query\_param = 'page\_size'

max\_page\_size = 100

class UserProfileListCreate(generics.ListCreateAPIView):

queryset = UserProfile.objects.all()

serializer\_class = UserProfileSerializer

permission\_classes = [permissions.IsAuthenticated]

pagination\_class = UserProfilePagination

filter\_backends = [DjangoFilterBackend, SearchFilter, OrderingFilter]

filterset\_fields = ['name']

search\_fields = ['name']

ordering\_fields = ['name']

class UserProfileRetrieveUpdateDelete(generics.RetrieveUpdateDeleteAPIView):

queryset = UserProfile.objects.all()

serializer\_class = UserProfileSerializer

permission\_classes = [permissions.IsAuthenticated]

In urls.py

from django.urls import path

from .views import UserProfileListCreate, UserProfileRetrieveUpdateDestroy

urlpatterns = [

path('profiles/', UserProfileListCreate.as\_view(), name='profile-list-create'),

path('profiles/<int:pk>/', UserProfileRetrieveUpdateDestroy.as\_view(), name='profile-retrieve-update-destroy'),

]

In models.py

from django.db import models

class UserProfile(models.Model):

name = models.CharField(max\_length=100)

email = models.EmailField(unique=True)

profile\_picture = models.ImageField(upload\_to='profile\_pictures/', blank=True, null=True)

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

return self.name

Task 2 :

In views.py

from django.http import StreamingHttpResponse

import time

def generate\_sentence():

sentence = "This is a sample sentence."

while True:

yield sentence + '\n'

time.sleep(1)

def stream\_sentence(request):

response = StreamingHttpResponse(generate\_sentence(), content\_type='text/plain')

return response

In urls.py

from django.urls import path

from .views import stream\_sentence

urlpatterns = [

path('stream\_sentence/', stream\_sentence, name='stream\_sentence'),

]

In html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Streamed Sentence</title>

</head>

<body>

<h1>Streamed Sentence</h1>

<hr>

<pre id="sentence"></pre>

<script>

const sentenceElement = document.getElementById('sentence');

const streamUrl = '/stream\_sentence/';

fetch(streamUrl)

.then(response => {

const reader = response.body.getReader();

return new ReadableStream({

start(controller) {

const decoder = new TextDecoder();

function push() {

reader.read().then(({ done, value }) => {

if (done) {

controller.close();

return;

}

controller.enqueue(decoder.decode(value, { stream: true }));

push();

});

}

push();

}

});

})

.then(stream => new Response(stream))

.then(response => response.text())

.then(text => {

sentenceElement.innerText = text;

})

.catch(err => {

console.error('Error streaming sentence:', err);

});

</script>

</body>

</html>

Task 4:

In models.py

from django.db import models

class Category(models.Model):

name = models.CharField(max\_length=100)

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

return self.name

class Subcategory(models.Model):

category = models.ForeignKey(Category, on\_delete=models.CASCADE, related\_name='subcategories')

name = models.CharField(max\_length=100)

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

return self.name

In serializers.py

from rest\_framework import serializers

from .models import Category, Subcategory

class SubcategorySerializer(serializers.ModelSerializer):

class Meta:

model = Subcategory

fields = ['id', 'name']

class CategorySerializer(serializers.ModelSerializer):

subcategories = SubcategorySerializer(many=True, read\_only=True)

class Meta:

model = Category

fields = ['id', 'name', 'subcategories']

In views.py

from rest\_framework import generics

from .models import Category

from .serializers import CategorySerializer

class CategoryList(generics.ListAPIView):

queryset = Category.objects.all()

serializer\_class = CategorySerializer

In urls.py

from django.urls import path

from .views import CategoryList

urlpatterns = [

path('categories/', CategoryList.as\_view(), name='category-list'),

]

Task 5

from django.db import models

class PostModel(models.Model):

title = models.CharField(max\_length=100)

author = models.CharField(max\_length=100)

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

class CommentModel(models.Model):

post = models.ForeignKey(PostModel, related\_name='comments', on\_delete=models.CASCADE)

comment = models.TextField()

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

In serializers.py

from rest\_framework import serializers

from .models import PostModel, CommentModel

class CommentSerializer(serializers.ModelSerializer):

class Meta:

model = CommentModel

fields = ['id', 'comment', 'publication\_date']

class PostSerializer(serializers.ModelSerializer):

comments = CommentSerializer(many=True, read\_only=True)

total\_comments = serializers.SerializerMethodField()

class Meta:

model = PostModel

fields = ['id', 'title', 'author', 'created\_at', 'comments', 'total\_comments']

def get\_total\_comments(self, obj):

return obj.comments.count()

In views.py

from rest\_framework import generics

from rest\_framework import filters

from .models import PostModel

from .serializers import PostSerializer

class PostList(generics.ListCreateAPIView):

queryset = PostModel.objects.all()

serializer\_class = PostSerializer

class PostDetail(generics.RetrieveUpdateDestroyAPIView):

queryset = PostModel.objects.all()

serializer\_class = PostSerializer

class PostFilterNoneTitle(generics.ListAPIView):

queryset = PostModel.objects.filter(title=None)

serializer\_class = PostSerializer

class PostRecentComments(generics.ListAPIView):

queryset = PostModel.objects.all().order\_by('-comments\_\_publication\_date')

serializer\_class = PostSerializer

class PostOrderByCreatedAt(generics.ListAPIView):

queryset = PostModel.objects.all().order\_by('-created\_at')

serializer\_class = PostSerializer

In urls.py

from django.urls import path

from .views import PostList, PostDetail, PostFilterNoneTitle, PostRecentComments, PostOrderByCreatedAt

urlpatterns = [

path('posts/', PostList.as\_view(), name='post-list'),

path('posts/<int:pk>/', PostDetail.as\_view(), name='post-detail'),

path('posts/filter/none-title/', PostFilterNoneTitle.as\_view(), name='post-filter-none-title'),

path('posts/recent-comments/', PostRecentComments.as\_view(), name='post-recent-comments'),

path('posts/order-by-created-at/', PostOrderByCreatedAt.as\_view(), name='post-order-by-created-at'),

]

In views.py

from rest\_framework import status

from rest\_framework.response import Response

class PostDetail(generics.RetrieveUpdateDestroyAPIView):

queryset = PostModel.objects.all()

serializer\_class = PostSerializer

def delete(self, request, \*args, \*\*kwargs):

instance = self.get\_object()

if instance.comments.count() == 0:

self.perform\_destroy(instance)

return Response(status=status.HTTP\_204\_NO\_CONTENT)

else:

return Response({"error": "Cannot delete post with comments."}, status=status.HTTP\_400\_BAD\_REQUEST)