ДОДАТОК А

КОД ПРОГРАМИ

users.models.py

```
from django.contrib.auth.models import AbstractUser, Group as DjangoGroup
from django.db import models


class CustomUser(AbstractUser):
 name = models.CharField(blank=True, max_length=255)
 lang = models.CharField(blank=True, max_length=255)
 display_name = models.CharField(blank=True, max_length=255)
 email = models.EmailField(unique=True)
 image = models.ImageField(upload_to='images/upload', blank=True)
 mason_group = models.ForeignKey(
 DjangoGroup,
 null=True,
 blank=True,
 on_delete=models.SET_NULL,

 help_text='User group used for access separation between chat `Flow`s',
 )

 def __str__(self):
 return self.email
```

user.serializers.py

```
from rest_framework import serializers
from django.contrib.auth.models import Group, Permission
```

```
from django.contrib.auth.hashers import make_password
from . import models


class PermissionSerializer(serializers.ModelSerializer):



 class Meta:
 model = Permission
 lookup_field = 'id'
 fields = ('id', 'name', 'codename')


class PermissionPrivateSerializer(serializers.ModelSerializer):
 class Meta:
 model = Permission
 fields = ('id', 'name')


class GroupSerializer(serializers.ModelSerializer):
 class Meta:
 model = Group
 lookup_field = 'id'
 fields = ('url', 'id', 'name', 'permissions')


class GroupListSerializer(serializers.HyperlinkedModelSerializer):
 permissions = PermissionPrivateSerializer(many=True)

 class Meta:
 model = Group
 fields = ('url', 'id', 'name', 'permissions')
```

```
class GroupPrivateSerializer(serializers.ModelSerializer):
 class Meta:
 model = Group
 fields = ('name', 'id', )


class UserDetailsSerializer(serializers.ModelSerializer):



 class Meta:
 model = models.CustomUser
 fields = ('pk', 'first_name', 'last_name', 'email', 'display_name', 'username', 'image',
'lang', 'groups',  'password', )


class UserListSerializer(serializers.HyperlinkedModelSerializer):
 groups = GroupPrivateSerializer(many=True)

 class Meta:
 model = models.CustomUser
 fields = ('pk', 'first_name', 'last_name', 'email', 'display_name', 'username', 'image',
'lang', 'groups',  'password', )


class UserSerializer(serializers.ModelSerializer):

 def validate_password(self, value: str) -> str:
 return make_password(value)

 class Meta:
 model = models.CustomUser
 fields = ('pk', 'first_name', 'last_name', 'email', 'display_name', 'username', 'image',
'lang', 'groups',  'password', )
```

users.views.py

```
from rest_framework import viewsets
from django.contrib.auth.models import Group, Permission
from django.db.transaction import atomic
from rest_framework.permissions import IsAdminUser, IsAuthenticated
from .models import CustomUser
from rest_framework.response import Response



from .serializers import GroupSerializer, UserSerializer, PermissionSerializer,
UserListSerializer,  GroupListSerializer

# granting global permissions to our CRUDs
# permission_rule = [IsAdminUser]



class UserViewSet(viewsets.ModelViewSet):
 """
 API endpoint that allows users to be viewed or edited.
 """
 queryset = CustomUser.objects.all()
 serializer_class = UserSerializer
 permission_classes = (IsAuthenticated, )

 def list(self, request):
 queryset = CustomUser.objects.all()
 serializer = UserListSerializer(queryset, many=True)
 return Response(serializer.data)
 # permission_classes = permission_rule

 def _reset_chat_authorization(self):
 # User Groups can be changed
 # (needs reset authorization timestamp)
```

```
user = self.get_object()
user.chat_logs.authorized_now().update(authorized_at=None)

@atomic
def update(self, *args, **kwargs):
self._reset_chat_authorization()
return super().update(*args, **kwargs)

@atomic
def partial_update(self, *args, **kwargs):



self._reset_chat_authorization()
return super().partial_update(*args, **kwargs)

def destroy(self, request, pk=None, permission_classes=(IsAdminUser,)):
pass



class GroupViewSet(viewsets.ModelViewSet):
"""
API endpoint that allows groups to be viewed or edited.
"""
queryset = Group.objects.all()
serializer_class = GroupSerializer
permission_classes = (IsAuthenticated, )

def list(self, request):
queryset = Group.objects.all()
serializer_context = {
'request': request,
}
serializer = GroupListSerializer(queryset,
context=serializer_context, many=True)  return
```

```
Response(serializer.data)
 # permission_classes = permission_rule


class PermissionViewSet(viewsets.ModelViewSet):
 """
 API endpoint that allows permissions to be viewed or edited.
 """
 queryset = Permission.objects.all()
 serializer_class = PermissionSerializer
 # permission_classes = permission_rule
```

vision.models.py

```
from django.db import models


# Create your models here.

class Dataset(models.Model):
 id = models.AutoField(primary_key=True, help_text="Dataset ID")
 name = models.CharField(blank=False, unique=True, max_length=255,
help_text="Internal name for  Dataset.")
 created_at = models.DateTimeField(auto_now_add=True)
 updated_at = models.DateTimeField(auto_now=True)


class WebcamObject(models.Model):
 id = models.AutoField(primary_key=True, help_text="Webcam ID")
 name = models.CharField(blank=False, unique=True, max_length=255,
help_text="Internal name for  Webcam.")
 description = models.CharField(blank=False, max_length=255, help_text="Short
description")  folder = models.CharField(blank=False, default=3306,
```

```
max_length=255, help_text="Folder where  incoming data will be stored")
 image = models.ImageField(blank=True, upload_to='images/upload')
 dataset = models.ForeignKey(Dataset, on_delete=models.PROTECT,
help_text="Dataset ID",  blank=True)
 created_at = models.DateTimeField(auto_now_add=True)
 updated_at = models.DateTimeField(auto_now=True)
```

vision.serializers.py

```
from rest_framework import serializers
from . import models

class DatasetSerializer(serializers.ModelSerializer):



 class Meta:
 model = models.Dataset
 lookup_field = 'id'
 fields = (
 'id',
 'name',
 'created_at',
 'updated_at',
 )


class WebcamObjectSerializer(serializers.ModelSerializer):
 class Meta:
 model = models.WebcamObject
 lookup_field = 'id'
 fields = (
 'id',
 'name',
 'description',
```

```
'folder',
'image',
'dataset',
'created_at',
'updated_at',
)
```

vision.views.py

```
from django.shortcuts import render
from rest_framework import viewsets, status
from rest_framework.decorators import action
from rest_framework.permissions import IsAdminUser, IsAuthenticated
from . import models, serializers
from .utils import getobj



from rest_framework.response import Response

class DatasetViewSet(viewsets.ModelViewSet):
 """
 API endpoint that allows External Databases settings to be viewed or edited.
 """
 queryset = models.Dataset.objects.all()
 serializer_class = serializers.DatasetSerializer
 permission_classes = (IsAuthenticated, )



class WebcamObjectViewSet(viewsets.ModelViewSet):
 """
 API endpoint that allows External Databases settings to be viewed or edited.
 """
 queryset = models.WebcamObject.objects.all()
 serializer_class = serializers.WebcamObjectSerializer
```

```
permission_classes = (IsAuthenticated, )

@action(detail=True, methods=['get'])
def test_state(self, request, pk=None):
webcam = self.get_object()

return Response({'detail': "Test finished", "img": getobj(webcam)})
```

vision.utils.py

```
import tensorflow as tf
import sys
import os
import urllib


# Disable tensorflow compilation warnings
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import tensorflow as tf

def getobj(image_path):
 # Read the image_data
 image_data = tf.gfile.FastGFile(image_path, 'rb').read()
 print(image_path)

 # Loads label file, strips off carriage return
 label_lines = [line.rstrip() for line
 in tf.gfile.GFile(r"./models/tf_files/retrained_labels.txt")]

 # Unpersists graph from file
 with tf.gfile.FastGFile(r"./models/tf_files/retrained_graph.pb", 'rb') as f:
 graph_def = tf.GraphDef()
```

```
graph_def.ParseFromString(f.read())
_ = tf.import_graph_def(graph_def, name='')

with tf.Session() as sess:
# Feed the image_data as input to the graph and get first prediction
softmax_tensor = sess.graph.get_tensor_by_name('final_result:0')

predictions = sess.run(softmax_tensor, \
{'DecodeJpeg/contents:0': image_data})

# Sort to show labels of first prediction in order of confidence
top_k = predictions[0].argsort()[-len(predictions[0]):][::-1]


for node_id in top_k:
count = 1
human_string = label_lines[node_id]
score = predictions[0][node_id]



print(count)
count += 1
print('%s (score = %.5f)' % (human_string, score))
score = (round((score * 100), 2))
return human_string,score
```