Account

**admin.py**

from django.contrib import admin

from django.contrib.auth.admin import UserAdmin as DjangoUserAdmin

from django.utils.translation import gettext\_lazy as \_

from django.contrib.auth.models import Group

from django.contrib.auth.admin import GroupAdmin as DjangoGroupAdmin

from .models import CustomUser

@admin.register(CustomUser)

class CustomUserAdmin(DjangoUserAdmin):

"""Add additional fields to user admin page."""

fieldsets = (

(None, {"fields": ("username", "password")}),

(\_("Personal info"), {"fields": ("name", "email", "device", "phone\_number", "gender", "exp\_id", "exp\_name", "exp\_state")}),

(

\_("Permissions"),

{

"fields": (

"is\_active",

"is\_staff",

"is\_superuser",

"groups",

"user\_permissions",

),

},

),

(\_("Important dates"), {"fields": ("last\_login", "date\_joined")}),

)

list\_display = ("username", "email")

actions = ['delete\_selected'] # Ensure bulk delete is enabled

class GroupProxy(Group):

"""Proxy model for Group. Dedicated for django-admin."""

class Meta:

"""Declare model being proxy."""

proxy = True

verbose\_name\_plural = verbose\_name = '用户组'

@admin.register(GroupProxy)

class MyGroupAdmin(DjangoGroupAdmin):

"""Grouping useradmin with groupadmin"""

actions = ['delete\_selected'] # Ensure bulk delete is enabled

admin.site.unregister(Group) # Avoid multiple GroupAdmin

**models.py**

from django.db import models

from django.contrib.auth.models import AbstractUser

from django.utils import timezone

class CustomUser(AbstractUser):

class Meta:

verbose\_name = '用户'

verbose\_name\_plural = '用户管理表'

email = models.EmailField(verbose\_name="邮箱地址")

device = models.CharField(max\_length=150, default="", verbose\_name="设备及其操作系统", blank=True)

phone\_number = models.CharField(max\_length=11, verbose\_name="手机号码")

gender = models.CharField(

max\_length=10,

choices=(

("male", "男"),

("female", "女"),

),

verbose\_name="性别",

default="未知"

)

token = models.CharField(max\_length=6, verbose\_name="修改密码令牌")

name = models.CharField(max\_length=20, verbose\_name="真名", default="")

token\_expires = models.DateTimeField(verbose\_name="令牌过期时间", default=timezone.now)

exp\_state = models.CharField(

max\_length=64,

verbose\_name="实验状态",

default="inactive",

)

exp\_name = models.CharField(

max\_length=64,

verbose\_name="实验名称",

default="",

blank=True,

)

exp\_id = models.IntegerField(

verbose\_name="实验ID",

default=-1,

)

**urls.py**

from django.urls import path

from .apis import UserRegisterApi,UserLoginApi

from .apis import modifyPasswordApi,modifyEmailApi,modifyPhoneApi,modifyGenderApi,modifyNameApi,getUserInfoApi

from .apis import Is\_PasswordApi,ResetPasswordApi,checkphoneApi

from rest\_framework\_simplejwt.views import (

TokenObtainPairView,

TokenRefreshView,

TokenVerifyView

)

urlpatterns = [

path("register/", UserRegisterApi.as\_view(), name="register"),

path("login/", UserLoginApi.as\_view(), name="Login"),

path("token\_obtain/", TokenObtainPairView.as\_view(), name="obtain"),

path("token\_refresh/", TokenRefreshView.as\_view(), name="refresh"),

path("token\_verify/", TokenVerifyView.as\_view(), name="verify"),

path("isPassword/",Is\_PasswordApi.as\_view(),name="isPassword"),

path("resetPassword/",ResetPasswordApi.as\_view(),name="resetPassword"),

path("modify/password/",modifyPasswordApi.as\_view(),name="modifyPassword"),

path("modify/email/",modifyEmailApi.as\_view(),name="modifyEmail"),

path("modify/phone\_number/",modifyPhoneApi.as\_view(),name="modifyPhone"),

path("modify/gender/",modifyGenderApi.as\_view(),name="modifyGender"),

path("modify/name/",modifyNameApi.as\_view(),name="modifyName"),

path("getUserInfo/",getUserInfoApi.as\_view(),name="getUserInfo"),

path("checkphone/",checkphoneApi.as\_view(),name="checkphone"),

]

**Analysis**

**admin.py**

from django.contrib import admin

from .models import gps\_cluster,bt\_cluster

class gps\_clusterAdmin(admin.ModelAdmin):

fields = ('username', 'cluster\_name', 'longitude', 'latitude')

list\_display = ('username', 'cluster\_name', 'longitude', 'latitude','label', 'timestamp')

search\_fields = ['username']

list\_per\_page = 20

actions = ['delete\_selected'] # Ensure bulk delete is enabled

list\_editable = ("cluster\_name",)

admin.site.register(gps\_cluster, gps\_clusterAdmin)

class bt\_clusterAdmin(admin.ModelAdmin):

fields = ('username', 'label', 'bt\_device')

list\_display = ('username', 'label','bt\_device')

search\_fields = ['username']

list\_per\_page = 20

list\_editable = ("label",)

actions = ['delete\_selected'] # Ensure bulk delete is enabled

admin.site.register(bt\_cluster, bt\_clusterAdmin)

**models.py**

from django.db import models

class gps\_cluster(models.Model):

class Meta:

verbose\_name = '定位聚类'

verbose\_name\_plural = '定位聚类表'

username = models.CharField(max\_length=10, verbose\_name="用户名")

cluster\_name = models.CharField(max\_length=10, verbose\_name="聚类名称",blank=True)

label = models.IntegerField(blank=True,null=True, verbose\_name="聚类标签")

timestamp = models.DateTimeField(auto\_now=True, verbose\_name="时间戳")

longitude = models.FloatField(default=0.0, verbose\_name="经度")

latitude = models.FloatField(default=0.0, verbose\_name="纬度")

class bt\_cluster(models.Model):

class Meta:

verbose\_name = '蓝牙聚类'

verbose\_name\_plural = '蓝牙聚类表'

username = models.CharField(max\_length=10, verbose\_name="用户名")

label = models.CharField(max\_length=10, verbose\_name="聚类名称",blank=True)

timestamp = models.DateTimeField(auto\_now=True, verbose\_name="时间戳")

bt\_device = models.CharField(max\_length=1500000, verbose\_name="蓝牙连接设备")

**serializers.py**

from .models import gps\_cluster,bt\_cluster

from sensor.models import LocationInf

from rest\_framework import serializers

class LabelSerializer(serializers.ModelSerializer):

class Meta:

model = LocationInf

fields = [

'longitude', 'latitude','label',

]

longitude = serializers.FloatField(required=True)

latitude = serializers.FloatField(required=True)

label = serializers.CharField(max\_length=20, required=True)

class UpdateBTLabelSerializer(serializers.Serializer):

class Meta:

model = bt\_cluster

fields = [

'bt\_device',

'label',

]

bt\_device = serializers.CharField(max\_length=17, required=True)

label = serializers.CharField(max\_length=255, required=True)

class get\_GpsclusterSerializers(serializers.ModelSerializer):

class Meta:

model = gps\_cluster

fields = [

'username',

'cluster\_name',

'latitude',

'longitude',

'timestamp',

]

class getBTlabelSerializer(serializers.ModelSerializer):

class Meta:

model = bt\_cluster

fields = [

'bt\_device',

'label',

]

**urls.py**

from django.urls import path

from .apis import updateLabelApi,get\_gps\_cluster,updateBTlabelApi,getBTLabelApi

from .apis import getGpsName

from rest\_framework\_simplejwt.views import (

TokenObtainPairView,

TokenRefreshView,

TokenVerifyView

)

urlpatterns = [

path("token\_obtain/", TokenObtainPairView.as\_view(), name="obtain"),

path("token\_refresh/", TokenRefreshView.as\_view(), name="refresh"),

path("token\_verify/", TokenVerifyView.as\_view(), name="verify"),

path("updateLabel/",updateLabelApi.as\_view(),name="updateLabel"),

path("get\_gpscluster/",get\_gps\_cluster.as\_view(),name="get\_gps\_cluster"),

path("updateBTlabel/",updateBTlabelApi.as\_view(),name="updateBTlabel"),

path("getBTLabel/",getBTLabelApi.as\_view(),name="getBTLabel"),

path("getGpsName/",getGpsName.as\_view(),name="getGpsName"),

]

**Backend**

**asgi.py**

"""

ASGI config for backend project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see

https://docs.djangoproject.com/en/4.2/howto/deployment/asgi/

"""

import os

from django.core.asgi import get\_asgi\_application

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'backend.settings')

application = get\_asgi\_application()

**urls.py**

"""

URL configuration for backend project.

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

https://docs.djangoproject.com/en/4.2/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

from django.urls import path

urlpatterns = [

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

path('api/', include('account.urls'), name='account'),

path('survey/', include('survey.urls'), name='survery'),

path('exp/', include('experiment.urls'), name='experiment'),

path('sensor/', include('sensor.urls'), name='sensor'),

path('analysis/', include('analysis.urls'), name='analysis'),

path('pose/', include('pose.urls'), name='pose'),

]

**settings.py**

"""

Django settings for backend project.

Generated by 'django-admin startproject' using Django 4.2.2.

For more information on this file, see

https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/4.2/ref/settings/

"""

from pathlib import Path

import os

import dotenv

# Build paths inside the project like this: BASE\_DIR / 'subdir'.

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

dotenv.load\_dotenv(dotenv\_path=BASE\_DIR.parent / ".env", verbose=True)

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = os.environ.get(

'DJANGO\_SECRET\_KEY',

'django-insecure-k4^s#n\*^xq7\*fp((\*7\*(kt@1-s4h46jq3wib-v(pil!a2l2)#1'

)

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = os.environ.get("DEBUG", "True")

ALLOWED\_HOSTS = [

"gps.primedigitaltech.com",

"127.0.0.1",

]

# Application definition

APPS\_PREFIX = ""

# 包含所有应用程序

APPS = [

"account",

"survey",

"experiment",

"sensor",

"analysis",

"pose",

]

LOCAL\_APPS = [

f"{APPS\_PREFIX}{app}" for app in APPS

]

THIRD\_PARTY\_APPS = [

'corsheaders',

'rest\_framework\_simplejwt',

'rest\_framework',

]

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

\*LOCAL\_APPS,

\*THIRD\_PARTY\_APPS,

]

MIDDLEWARE = [

'corsheaders.middleware.CorsMiddleware',

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

# 'django.middleware.csrf.CsrfViewMiddleware',

'corsheaders.middleware.CorsMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

# 'base.LogRequestMiddleware.LogRequestMiddleware'

]

ROOT\_URLCONF = 'backend.urls'

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [os.path.join(BASE\_DIR, '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',

],

},

},

]

WSGI\_APPLICATION = 'backend.wsgi.application'

# Database

# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

if DEBUG:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'db.sqlite3',

}

}

else:

DB\_NAME = os.environ.get("DB\_NAME", None)

DB\_USER = os.environ.get("DB\_USER", None)

DB\_PSWD = os.environ.get("DB\_PSWD", None)

DB\_HOST = os.environ.get("DB\_HOST", "127.0.0.1")

DB\_PORT = os.environ.get("DB\_PORT", "5432")

if not DB\_NAME or not DB\_USER or not DB\_PSWD:

raise Exception("DB\_NAME, DB\_USER, DB\_PSWD not found in .env file")

DATABASES = {

'default': {

'ENGINE': "django.db.backends.postgresql",

'NAME': DB\_NAME,

'USER': DB\_USER,

'PASSWORD': DB\_PSWD,

'HOST': DB\_HOST,

'PORT': DB\_PORT,

}

}

# Password validation

# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

# Internationalization

# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'Asia/Shanghai'

USE\_I18N = True

USE\_TZ = False

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC\_URL = '/static/'

# Default primary key field type

# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

from base.cors import \*

from base.rest\_framework import \*

from base.simplejwt import \*

from base.email\_inf import \*

AUTH\_USER\_MODEL = 'account.CustomUser'

STATIC\_ROOT = '/var/www/gps\_inf/static'

SECURE\_CROSS\_ORIGIN\_OPENER\_POLICY = 'None'

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

EMAIL\_HOST\_PASSWORD = os.environ.get("EMAIL\_HOST\_PASSWORD", None) #发送邮件的邮箱密码(这里使用的是授权码)

# django内部日志设置

BASE\_LOG\_DIR = os.path.join(BASE\_DIR, "logs")

LOGGING = {

'version': 1,

'disable\_existing\_loggers': False,

'formatters': {

'standard': {

'format': '[%(asctime)s][%(threadName)s:%(thread)d][task\_id:%(name)s][%(filename)s:%(lineno)d]'

'[%(levelname)s][%(message)s]'

},

'simple': {

'format': '[%(levelname)s][%(asctime)s][%(filename)s:%(lineno)d]%(message)s'

},

'collect': {

'format': '%(message)s'

}

},

'filters': {

'require\_debug\_true': {

'()': 'django.utils.log.RequireDebugTrue',

},

},

'handlers': {

'console': {

'level': 'DEBUG',

'filters': ['require\_debug\_true'],

'class': 'logging.StreamHandler',

'formatter': 'simple'

},

'default': {

'level': 'INFO',

'class': 'logging.handlers.RotatingFileHandler',

'filename': os.path.join(BASE\_LOG\_DIR, "django.info.log"),

'maxBytes': 1024 \* 1024 \* 50,

'backupCount': 3,

'formatter': 'standard',

'encoding': 'utf-8',

},

'error': {

'level': 'ERROR',

'class': 'logging.handlers.RotatingFileHandler',

'filename': os.path.join(BASE\_LOG\_DIR, "django.error.log"),

'maxBytes': 1024 \* 1024 \* 50,

'backupCount': 5,

'formatter': 'standard',

'encoding': 'utf-8',

},

'collect': {

'level': 'INFO',

'class': 'logging.handlers.RotatingFileHandler',

'filename': os.path.join(BASE\_LOG\_DIR, "django.collect.log"),

'maxBytes': 1024 \* 1024 \* 50,

'backupCount': 5,

'formatter': 'collect',

'encoding': "utf-8"

}

},

'loggers': {

'': {

'handlers': ['default', 'console', 'error'],

'level': 'DEBUG',

'propagate': True,

},

'collect': {

'handlers': ['console', 'collect'],

'level': 'INFO',

}

},

}

**wsgi.py**

"""

WSGI config for backend project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see

https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/

"""

import os

from django.core.wsgi import get\_wsgi\_application

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'backend.settings')

application = get\_wsgi\_application()

**base**

**LogRequestMiddleware.py**

import logging

import os

from pathlib import Path

class LogRequestMiddleware:

def \_\_init\_\_(self, get\_response):

self.get\_response = get\_response

# 设置日志文件路径，放在项目根目录下

project\_root = Path(\_\_file\_\_).resolve().parent.parent

log\_file\_path = os.path.join(project\_root, 'request\_log.log')

# 配置日志记录器

self.logger = logging.getLogger('RequestLogger')

handler = logging.FileHandler(log\_file\_path)

formatter = logging.Formatter('%(asctime)s - %(message)s')

handler.setFormatter(formatter)

self.logger.addHandler(handler)

self.logger.setLevel(logging.INFO)

def \_\_call\_\_(self, request):

# 记录请求体

body = request.body.decode('utf-8')

self.logger.info(f'Request Body: {body}')

# 记录请求头

headers = {k: v for k, v in request.META.items() if k.startswith('HTTP\_')}

self.logger.info(f'Request Headers: {headers}')

response = self.get\_response(request)

return response

**cors.py**

'''配置跨域资源共享（CORS）的相关设置。CORS 是一种机制，它允许服务器在响应头中加入一些特定的头信息，从而允许浏览器在不同的域名下请求资源。'''

CORS\_ALLOW\_CREDENTIALS = True

CORS\_ALLOW\_ALL\_ORIGINS = True

**email\_inf.py**

#邮件设置

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

EMAIL\_USE\_TLS = True #是否使用TLS安全传输协议(用于在两个通信应用程序之间提供保密性和数据完整性。)

EMAIL\_HOST ='smtp.163.com' #发送邮件的邮箱 的 SMTP服务器，这里用了163邮箱

EMAIL\_PORT = 25 #发件箱的SMTP服务器端口

EMAIL\_HOST\_USER = '17620642718@163.com' #发送邮件的邮箱地址

EMAIL\_FROM = '17620642718<17620642718@163.com>' #收件人看到的发件人

**rest\_framework.py**

REST\_FRAMEWORK = {

'DEFAULT\_PERMISSION\_CLASSES': (

'rest\_framework.permissions.IsAuthenticated', # 认证用户才给访问

),

'DEFAULT\_AUTHENTICATION\_CLASSES': (

'rest\_framework\_simplejwt.authentication.JWTAuthentication', # 基于JSON Web Token的认证

'rest\_framework.authentication.BasicAuthentication', # 基于用户名和密码的认证

# 'rest\_framework.authentication.SessionAuthentication', # 基于session的认证

),

}

**simplejwt.py**

from datetime import timedelta

# from account.serializers import MyTokenObtainPairSerializer

# 自定义参数

SIMPLE\_JWT = {

'ACCESS\_TOKEN\_LIFETIME': timedelta(days=30),

'REFRESH\_TOKEN\_LIFETIME': timedelta(days=30),

# "TOKEN\_OBTAIN\_SERIALIZER": MyTokenObtainPairSerializer,

}

**Experiment**

**admin.py**

from django.contrib import admin

from .models import experiment,exp\_history

# Register your models here.

class experimentAdmin(admin.ModelAdmin):

fields = ( 'start\_time','end\_time','exp\_name','description',"acc\_frequency","bt\_frequency","gps\_frequency","gyro\_frequency","participants\_name",)

list\_display = ('exp\_id','exp\_name','description','start\_time','end\_time',"acc\_frequency","bt\_frequency","gps\_frequency","gyro\_frequency","participants\_name",)

list\_per\_page = 10

search\_fields = ['exp\_name']

list\_filter = ('exp\_name',)

list\_editable = ("start\_time","end\_time",'exp\_name','description',"acc\_frequency","bt\_frequency","gps\_frequency","gyro\_frequency","participants\_name")

admin.site.register(experiment,experimentAdmin)

class exp\_historyAdmin(admin.ModelAdmin):

fields = ('exp\_name','username')

list\_display = ('exp\_id','username','exp\_name','join\_time','exit\_time')

list\_per\_page = 10

search\_fields = ['exp\_name']

list\_filter = ('exp\_name',)

list\_editable = ("username","exp\_name",)

admin.site.register(exp\_history,exp\_historyAdmin)

**models.py**

from django.db import models

class experiment(models.Model):

class Meta:

verbose\_name = '实验'

verbose\_name\_plural = '实验表'

start\_time = models.DateTimeField(verbose\_name="实验开始时间")

end\_time = models.DateTimeField(verbose\_name="实验结束时间")

exp\_id = models.AutoField(primary\_key=True, verbose\_name="实验ID")

exp\_name = models.CharField(max\_length=64, verbose\_name="实验名称", default="")

description = models.CharField(max\_length=255, verbose\_name="实验描述", default="")

gps\_frequency = models.IntegerField(verbose\_name="GPS调用间隔(分钟)", default=-1)

bt\_frequency = models.IntegerField(verbose\_name="蓝牙调用频率(分钟)", default=-1)

acc\_frequency = models.IntegerField(verbose\_name="加速度调用频率(秒)", default=-1)

gyro\_frequency = models.IntegerField(verbose\_name="陀螺仪调用频率(秒)", default=-1)

participants\_name = models.CharField(max\_length=64, verbose\_name="实验参与者", default="", blank=True)

class exp\_history(models.Model):

class Meta:

verbose\_name = '实验历史'

verbose\_name\_plural = '实验历史表'

username = models.CharField(max\_length=10, verbose\_name="用户名")

exp\_id = models.IntegerField(verbose\_name="实验ID")

exp\_name = models.CharField(max\_length=64, verbose\_name="实验名称", default="", blank=True)

description = models.CharField(max\_length=255, verbose\_name="实验描述", default="")

join\_time = models.DateTimeField(verbose\_name="参加时间", auto\_now\_add=True)

exit\_time = models.DateTimeField(verbose\_name="退出时间", null=True, blank=True)

**serializers.py**

from .models import experiment,exp\_history

from rest\_framework import serializers

class seeExperimentSerializer(serializers.ModelSerializer):

class Meta:

model=experiment

fields = ['exp\_name','description','start\_time','end\_time','gps\_frequency','acc\_frequency','bt\_frequency','gyro\_frequency']

class exp\_historySerializer(serializers.ModelSerializer):

class Meta:

model=exp\_history

fields = ['exp\_id','exp\_name','username','description','join\_time','exit\_time']

**urls.py**

from django.urls import path

from .apis import seeExperimentApi,chooseExperimentApi,exitExperimentApi,myExperimentApi,seeExperimentHistoryApi

urlpatterns = [

path('seeExp/', seeExperimentApi.as\_view(),name="seeExperiment"),

path('chooseExp/', chooseExperimentApi.as\_view(),name='chooseExperiment'),

path('exitExp/', exitExperimentApi.as\_view(),name='exitExperiment'),

path('myExp/', myExperimentApi.as\_view(),name='myExperiment'),

path('seeExpHistory/', seeExperimentHistoryApi.as\_view(),name='seeExperimentHistory'),

]

**views.py**

from django.shortcuts import render

# Create your views here.

**Pose**

**admin.py**

from django.contrib import admin

from .models import PoseSource

class PoseSourceAdmin(admin.ModelAdmin):

list\_display = ('video', 'upload\_date', 'username')

list\_filter = ('upload\_date', 'username')

ordering = ('-upload\_date',)

admin.site.register(PoseSource, PoseSourceAdmin)

**apis.py**

from rest\_framework import status

from rest\_framework.response import Response

from rest\_framework.views import APIView

from .models import PoseSource

from .serializers import PoseSourceSerializer

from rest\_framework.permissions import IsAuthenticated

class PoseSourceUploadView(APIView):

permission\_classes = [IsAuthenticated]

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

serializer = PoseSourceSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

class PoseSourceListView(APIView):

permission\_classes = [IsAuthenticated]

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

queryset = PoseSource.objects.filter(username=request.user) # 过滤当前用户的视频

serializer = PoseSourceSerializer(queryset, many=True)

return Response(serializer.data)

**models.py**

from django.db import models

# Create your models here.

class PoseSource(models.Model):

class Meta:

verbose\_name = '视频'

verbose\_name\_plural = '姿势视频表'

username = models.CharField(max\_length=100,verbose\_name="用户名")

video = models.FileField(upload\_to='videos/',verbose\_name="视频路径")

upload\_date = models.DateTimeField(auto\_now\_add=True,verbose\_name="上传时间")

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

return f"PoseSource: {self.video.name} uploaded on {self.upload\_date}"

**serializers.py**

from rest\_framework import serializers

from .models import PoseSource

class PoseSourceSerializer(serializers.ModelSerializer):

class Meta:

model = PoseSource

fields = ['id', 'video', 'upload\_date']

read\_only\_fields = ['upload\_date']

**urls.py**

from django.urls import path

from .apis import PoseSourceUploadView,PoseSourceListView

urlpatterns = [

path('upload/', PoseSourceUploadView.as\_view(), name='pose\_source\_upload'),

path('list/', PoseSourceListView.as\_view(), name='pose\_source\_list'),

]

**Sensor**

**admin.py**

from django.contrib import admin

from django.utils.translation import gettext\_lazy as \_

from .models import LocationInf, AccelerometerInf, BlueToothInf,GyroInf,BatteryInf

class LocationAdmin(admin.ModelAdmin):

fields = ('username', 'device', 'longitude', 'latitude', 'accuracy')

list\_display = ('username', 'device', 'longitude', 'latitude', 'accuracy', 'timestamp', 'label')

search\_fields = ['username']

list\_per\_page = 20

actions = ['delete\_selected'] # Ensure bulk delete is enabled

admin.site.register(LocationInf, LocationAdmin)

class BlueToothAdmin(admin.ModelAdmin):

fields = ('username', 'device', 'connection\_device')

list\_display = ('username', 'device', 'connection\_device', 'timestamp')

search\_fields = ['username']

list\_per\_page = 6

actions = ['delete\_selected'] # Ensure bulk delete is enabled

admin.site.register(BlueToothInf, BlueToothAdmin)

class AccelerometerAdmin(admin.ModelAdmin):

fields = ('username', 'device', 'acc\_x', 'acc\_y', 'acc\_z')

list\_display = ('username', 'device', 'acc\_x', 'acc\_y', 'acc\_z', 'timestamp')

search\_fields = ['username']

list\_per\_page = 20

actions = ['delete\_selected'] # Ensure bulk delete is enabled

admin.site.register(AccelerometerInf, AccelerometerAdmin)

class GyroAdmin(admin.ModelAdmin):

fields = ('username', 'device', 'x', 'y', 'z')

list\_display = ('username', 'device', 'x', 'y', 'z', 'timestamp')

search\_fields = ['username']

list\_per\_page = 20

actions = ['delete\_selected'] # Ensure bulk delete is enabled

admin.site.register(GyroInf, GyroAdmin)

class BatteryAdmin(admin.ModelAdmin):

fields = ('username', 'device', 'battery\_level', 'battery\_status')

list\_display = ('username', 'device', 'battery\_level', 'battery\_status', 'timestamp')

search\_fields = ['username']

list\_per\_page = 20

actions = ['delete\_selected'] # Ensure bulk delete is enabled

admin.site.register(BatteryInf, BatteryAdmin)

**models.py**

from django.db import models

class LocationInf(models.Model):

class Meta:

verbose\_name = '定位记录'

verbose\_name\_plural = '定位表'

username = models.CharField(max\_length=10, verbose\_name="用户名")

longitude = models.FloatField(default=0.0, verbose\_name="经度")

latitude = models.FloatField(default=0.0, verbose\_name="纬度")

device = models.CharField(max\_length=150, default="", verbose\_name="设备",blank=True,null=True)

timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name="时间戳")

accuracy = models.FloatField(default=0.0, verbose\_name="精确范围")

label = models.CharField(max\_length=20, default="", verbose\_name="用户标注")

class BlueToothInf(models.Model):

class Meta:

verbose\_name = '蓝牙记录'

verbose\_name\_plural = '蓝牙表'

username = models.CharField(max\_length=10, verbose\_name="用户名")

connection\_device = models.CharField(max\_length=1500000, verbose\_name="连接设备")

device = models.CharField(max\_length=150, default="", verbose\_name="设备",blank=True,null=True)

timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name="时间戳")

class AccelerometerInf(models.Model):

class Meta:

verbose\_name = '加速度计'

verbose\_name\_plural = '加速度表'

username = models.CharField(max\_length=10, verbose\_name="用户名")

acc\_x = models.FloatField(default=0.0, verbose\_name="加速度X")

acc\_y = models.FloatField(default=0.0, verbose\_name="加速度Y")

acc\_z = models.FloatField(default=0.0, verbose\_name="加速度Z")

device = models.CharField(max\_length=150, default="", verbose\_name="设备",null=True,blank=True)

timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name="时间戳")

class GyroInf(models.Model):

class Meta:

verbose\_name = '陀螺仪计'

verbose\_name\_plural = '陀螺仪表'

username = models.CharField(max\_length=10, verbose\_name="用户名")

x = models.FloatField(default=0.0, verbose\_name="X轴")

y = models.FloatField(default=0.0, verbose\_name="Y轴")

z = models.FloatField(default=0.0, verbose\_name="Z轴")

device = models.CharField(max\_length=150, default="", verbose\_name="设备",null=True,blank=True)

timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name="时间戳")

class BatteryInf(models.Model):

class Meta:

verbose\_name = '电池信息'

verbose\_name\_plural = '电池信息表'

username = models.CharField(max\_length=10, verbose\_name="用户名")

battery\_level = models.FloatField(default=0.0, verbose\_name="电池电量")

battery\_status = models.CharField(max\_length=150, default="", verbose\_name="电池状态")

device = models.CharField(max\_length=150, default="", verbose\_name="设备",null=True,blank=True)

timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name="时间戳")

**urls.py**

from django.urls import path

from .apis import UpdateLocationApi,UpdateBTApi,UpdateACCApi,GetACCData,GetBTData,GetGPSData,GetGyroData,UpdateGyroApi

from .apis import GetBatteryData,updateBatteryApi

urlpatterns = [

path("updateAcc/", UpdateACCApi.as\_view(), name="updateAcc"),

path("updateBT/", UpdateBTApi.as\_view(), name="updateBoothTooth"),

path("updateLocation/", UpdateLocationApi.as\_view(), name="updateLocation"),

path("getACCdata/",GetACCData.as\_view(),name="getACCdata"),

path("getGPSdata/",GetGPSData.as\_view(),name="getGPSdata"),

path("getBTdata/",GetBTData.as\_view(),name="getBTdata"),

path("getGyrodata/",GetGyroData.as\_view(),name="getGyrodata"),

path("updateGyro/",UpdateGyroApi.as\_view(),name="updateGyro"),

path("getBatterydata/",GetBatteryData.as\_view(),name="getBatteryData"),

path("updateBattery/",updateBatteryApi.as\_view(),name="updateBattery"),

]

**Survey**

**admin.py**

from django.contrib import admin

from .models import Question, Survey, Answer

# 注册Question模型的管理员界面

class QuestionAdmin(admin.ModelAdmin):

fields = ('question\_text','question\_group', 'question\_type', 'choices')

list\_display = ('question\_id','question\_group', 'question\_text', 'question\_type', 'choices')

list\_per\_page = 10

search\_fields = ['question\_id']

list\_filter = ('question\_id',)

list\_editable = ('question\_text', 'question\_type', 'question\_group','choices')

admin.site.register(Question, QuestionAdmin)

# 注册Survey模型的管理员界面

class SurveyAdmin(admin.ModelAdmin):

fields = ('title', 'description', 'questions')

list\_display = ('survey\_id', 'title', 'description', 'created\_at', 'questions','get\_question\_text')

list\_per\_page = 10

search\_fields = ['title']

list\_filter = ('title',)

list\_editable = ('title', 'description', 'questions')

def get\_question\_text(self, obj):

res=""

for i in obj.questions.split(';'):

if i=='':

continue

res+=f"{i}."+Question.objects.get(question\_id=i).question\_text

return res

get\_question\_text.short\_description = '具体问题'

admin.site.register(Survey, SurveyAdmin)

# 注册Answer模型的管理员界面

class AnswerAdmin(admin.ModelAdmin):

list\_display = ('response\_id', 'survey\_id', 'question\_id', 'get\_question\_text', 'answer\_text', 'username', 'response\_date')

list\_per\_page = 10

search\_fields = ['username']

list\_filter = ('survey\_id',)

list\_display\_links = ('survey\_id',)

def get\_question\_text(self, obj):

return Question.objects.get(question\_id=obj.question\_id).question\_text

get\_question\_text.short\_description = '具体问题'

admin.site.register(Answer, AnswerAdmin)

**models.py**

from django.db import models

class Survey(models.Model):

class Meta:

verbose\_name = '调查问卷'

verbose\_name\_plural = '调查问卷表'

survey\_id = models.AutoField(primary\_key=True,verbose\_name="问卷ID")

title = models.CharField(max\_length=200,verbose\_name="问卷标题")

description = models.TextField(verbose\_name="问卷描述",blank=True)

created\_at = models.DateTimeField(auto\_now\_add=True,verbose\_name="创建时间")

questions= models.CharField(max\_length=200,verbose\_name="问卷包含的问题ID")

class Question(models.Model):

class Meta:

verbose\_name = '问题'

verbose\_name\_plural = '问题表'

question\_id = models.AutoField(primary\_key=True,verbose\_name="问题ID")

question\_text = models.CharField(max\_length=200,verbose\_name="问题文本")

question\_type = models.CharField(max\_length=20, choices=[('text', 'Text'), ('choice', 'Choice'), ('rating', 'Rating')],verbose\_name="问题类型")

choices = models.TextField(null=True, blank=True,verbose\_name="选择项")

question\_group = models.CharField(max\_length=200,verbose\_name="问题分组")

class Answer(models.Model):

class Meta:

verbose\_name = '问卷回答'

verbose\_name\_plural = '问卷回答表'

response\_id = models.AutoField(primary\_key=True,verbose\_name="响应ID")

survey\_id = models.IntegerField(verbose\_name="问卷ID")

question\_id = models.IntegerField(verbose\_name="问题ID")

username = models.CharField(max\_length=20,verbose\_name="回答者用户名")

answer\_text = models.TextField(verbose\_name="回答文本")

response\_date = models.DateTimeField(auto\_now\_add=True,verbose\_name="响应日期")

**serializers.py**

from rest\_framework import serializers

from .models import Survey, Question, Answer

class SurveySerializer(serializers.ModelSerializer):

class Meta:

model = Survey

fields = ['survey\_id', 'title', 'description', 'created\_at', 'questions']

title = serializers.CharField(max\_length=200)

description = serializers.CharField()

questions = serializers.CharField()

class QuestionSerializer(serializers.ModelSerializer):

class Meta:

model = Question

fields = ['question\_id', 'question\_text', 'question\_type', 'question\_group','choices']

question\_text = serializers.CharField(max\_length=200)

question\_type = serializers.CharField(max\_length=20)

question\_group = serializers.CharField(max\_length=200)

class AnswerSerializer(serializers.ModelSerializer):

class Meta:

model = Answer

fields = ['response\_id', 'survey\_id', 'question\_id', 'username', 'answer\_text', 'response\_date']

**urls.py**

from django.urls import path

from .apis import ShowQuestionApi,SendResApi,createSurveyApi,submitQuestionApi,getquestionsApi

urlpatterns = [

path("createSurvey/",createSurveyApi.as\_view(),name="createSurvey"),

path("sendRes/",SendResApi.as\_view(),name="sendRes"),

path("showQuestion/<int:survey\_id>",ShowQuestionApi.as\_view(),name="showQuestion"),

path("submitQuestion/",submitQuestionApi.as\_view(),name="submitQuestion"),

path("getquestions/",getquestionsApi.as\_view(),name="getquestions"),

]

**views.py**

from django.shortcuts import render

# Create your views here.