day08 认证相关

认证结合之后的效果：

- 发布页：未登录，点发布跳转到登录页面。
- 首页：不用登录
- 详细页面：不用登录、如果登录就需要在访问记录中添加一条数据。
    - 评论，跳转到登录页面
    - 点赞，跳转到登录页面
- 个人中心，无需登录
- 登录页面，无需登录

今日概要

- 代码整合
- 认证
- 自定义tabbar

今日详细

1. 代码整合

登录成功之后保存 nickname和avatar

2.通用认证组件

request.user

- None，未登录
- 不为空，用户对象

2.1 处理访问记录

【demos - 4 - 认证组件.zip】

【demo - 4 - 认证组件.zip】

2.2 登录之后才具有评论和发布

- 小程序
    - 按钮点击的时候先做判断?
    - 自定义tabbar
    - 携带token
        - url

        - header
            - Authorization: token asdfauposdifuaskjdfpoaiusdfj
- api
    - 两个认证类
    - 应用场景：
        - 都不需要登录视图
        - 都需要认证的视图
        - 有的需要&有的不需要（自定义get_authenticators）

```
class CommentView(APIView):

    def get_authenticators(self):
        if self.request.method == 'POST':
            return [UserAuthentication(), ]
        return [GeneralAuthentication(), ]

    def get(self,request,*args,**kwargs):
        root_id = request.query_params.get('root')
        # 1. 获取这个根评论的所有子孙评论
        node_queryset =
models.CommentRecord.objects.filter(root_id=root_id).order_by('id')
        # 2. 序列化
        ser =
CommentModelSerializer(instance=node_queryset,many=True)

        return Response(ser.data,status=status.HTTP_200_OK)

    def post(self,request,*args,**kwargs):
        # 1. 进行数据校验: news/depth/reply/content/root
        ser = CreateCommentModelSerializer(data=request.data)
        if ser.is_valid():
            # 保存到数据库
            ser.save(user_id=1)
            # 对新增到的数据值进行序列化(数据格式需要调整)
            news_id = ser.data.get('news')

models.News.objects.filter(id=news_id).update(comment_count=F('comm
ent_count')+1)

            return
Response(ser.data,status=status.HTTP_201_CREATED)
        return
Response(ser.errors,status=status.HTTP_400_BAD_REQUEST)
```

3.随意的接口

结论：在serializer中可以调用request

```
class TestSER(serializers.ModelSerializer):
    xx = serializers.SerializerMethodField()
```

```
    class Meta:
        model = models.Topic
        fields = "__all__"

    def get_xx(self,obj):
        # 获取request 、 request.user
        self.context['request'].user
        return 123


class TestView(ListAPIView):
    queryset = models.Topic.objects
    serializer_class = TestSER
```

4.完善详细页面

- 点赞

总结

1. 小程序

1.1 申请账号APPID

1.2 搭建开发者工具

1.3 目录结构

小程序文件的目录结构

- config
  - api.js
  - settings.js
- components
- pages
  - index
    - index.js
    - index.wxml
    - index.wxss
    - index.json
- utils
- static
- app.js
- app.json
- app.wxss

1.4 组件

- view/text/image

- textarea/navigator/input/button
- block/cover-view/cover-image/swiper

1.5 微信API

- request/navigateto/back/switchtab/chooseimage/chooseloacation
- getUserInfo + button + opensetting
- showToast/showloading/hideloadding
- setstoragesync/setstoreagesync
- getApp/ getcurrenpages

1.6 事件

- 自定义事件
  - bindtab
  - bindinput
- onload/onshow/onread/onunload/onhide/onpulldownrefresh/onreachbuttom/onshareapp message
- onlunch

1.7 双向绑定 mvvm

- setData全部
- setData局部

```
setData({
    ['xx.xx.xx.x']:123
})
```

```
var v1 = 1;
var v2 = 2;
var v3 =v1+v2;
setData({
    ['xx.xx['+ v3 +']x']:123
})
```

1.8 指令

- for
- if

1.9 自定义tabbar

2.API（DRF）

2.1 视图

- APIView
- ListAPIView...
- 视图中自定义方法
  - 选择serializer

```
class TestView(ListAPIView,CreateAPIView):
    queryset = models.Topic.objects
    serializer_class = TestSER

    def get_serializer_class(self):
        if self.request.method == "GET":
            return TestSER
        return TestSER
```

- 选择authenticator

```
class TestView(ListAPIView,CreateAPIView):
    queryset = models.Topic.objects
    serializer_class = TestSER

    def get_authenticators(self):
        if self.request.method == "GET":
            return [UserAuthentication(), ]
        return [GeneralAuthentication(), ]
```

- 自定义视图函数

```
class TestView(ListAPIView,CreateAPIView):
    queryset = models.Topic.objects
    serializer_class = TestSER

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

        response = super().get(request,*args,**kwargs)

        return response
```

- 自定义filter做筛选（maxid/minid）

2.2 序列化器

他有两个作用：序列化、校验。

2.2.1 校验

```
class TestSER(serializers.ModelSerializer):
    class Meta:
        model = models.Topic
        fields = "__all__"
```

```
is_valid
```

- 自定义校验规则

```
import re
from rest_framework.exceptions import ValidationError


def phone_validator(value):
    if not re.match(r"^(1[3|4|5|6|7|8|9])\d{9}$",value):
        raise ValidationError('手机格式错误')

class TestSER(serializers.ModelSerializer):
    title = serializers.CharField(label='手机号',validators=
[phone_validator,])
    class Meta:
        model = models.Topic
        fields = "__all__"
```

- 自定义方法校验

```
class TestSER(serializers.ModelSerializer):
    title = serializers.CharField(label='手机号')
    class Meta:
        model = models.Topic
        fields = "__all__"

    def validate_title(self, value):
        # 具体的校验规则
        return value
```

- 嵌套校验

```
class CreateNewsTopicModelSerializer(serializers.Serializer):
    key = serializers.CharField()
    cos_path = serializers.CharField()


class CreateNewsModelSerializer(serializers.ModelSerializer):
    imageList = CreateNewsTopicModelSerializer(many=True)

    class Meta:
        model = models.News
        exclude = ['user', 'viewer_count', 'comment_count',"favor_count"]

    def create(self, validated_data):
        # 把imageList切走
        image_list = validated_data.pop('imageList')

        # 创建New表中的数据
        news_object = models.News.objects.create(**validated_data)

        data_list = models.NewsDetail.objects.bulk_create(
            [models.NewsDetail(**info, news=news_object) for info in
image_list]
        )
        news_object.imageList = data_list

        if news_object.topic:
            news_object.topic.count += 1
```

```
            news_object.save()

        return news_object
```

- 注意事项：如果定义的字段指向做序列化不做校验。

```
class TestSER(serializers.ModelSerializer):
    title = serializers.CharField(label='手机号',read_only=True)
    class Meta:
        model = models.Topic
        fields = "__all__"
```

2.2.2 序列化

```
class TestSER(serializers.ModelSerializer):
    class Meta:
        model = models.Topic
        fields = "__all__"
```

- 自定义字段+source

```
class TestSER(serializers.ModelSerializer):
    v1 = serializers.CharField(label='手机号',source='get_gender_display')
    class Meta:
        model = models.Topic
        fields = "__all__"
```

- 自定义时间字段

```
class TestSER(serializers.ModelSerializer):
    v1 = serializers.DateTimeField(format="%Y")
    class Meta:
        model = models.Topic
        fields = "__all__"
```

- 复杂操作字段

```
class NewsDetailModelSerializer(serializers.ModelSerializer):
    images = serializers.SerializerMethodField()
    create_date = serializers.DateTimeField(format="%Y-%m-%d %H:%M")

    user = serializers.SerializerMethodField()
    topic = serializers.SerializerMethodField()

    viewer = serializers.SerializerMethodField()
    comment = serializers.SerializerMethodField()

    is_favor = serializers.SerializerMethodField()

    class Meta:
        model = models.News
```

```
        exclude = ['cover',]

    def get_images(self,obj):
        detail_queryset = models.NewsDetail.objects.filter(news=obj)
        # return [row.cos_path for row in detail_queryset]
        # return [{'id':row.id,'path':row.cos_path} for row in
detail_queryset]
        return [model_to_dict(row,['id','cos_path']) for row in
detail_queryset]

    def get_user(self, obj):
        return model_to_dict(obj.user, fields=['id', 'nickname', 'avatar'])

    def get_topic(self, obj):
        if not obj.topic:
            return
        return model_to_dict(obj.topic, fields=['id', 'title'])

    def get_viewer(self,obj):
        # 根据新闻的对象 obj(news)
        # viewer_queryset =
models.ViewerRecord.objects.filter(news_id=obj.id).order_by('-id')[0:10]
        queryset = models.ViewerRecord.objects.filter(news_id=obj.id)
        viewer_object_list = queryset.order_by('-id')[0:10]
        context = {
            'count':queryset.count(),
            'result': [model_to_dict(row.user,['nickname','avatar']) for row
in viewer_object_list]
        }
        return context

    def get_comment(self,obj):
        """
        获取所有的1级评论，再给每个1级评论获取一个耳机评论。
        :param obj:
        :return:
        """

        # 1.获取所有的 一级 评论
        first_queryset =
models.CommentRecord.objects.filter(news=obj,depth=1).order_by('id')
[0:10].values(
            'id',
            'content',
            'depth',
            'user__nickname',
            'user__avatar',
            'create_date'
        )
        first_id_list = [ item['id'] for item in first_queryset]
        # 2.获取所有的二级评论
        # second_queryset =
models.CommentRecord.objects.filter(news=obj,depth=2)
        # 2. 获取所有1级评论下的二级评论
        # second_queryset = models.CommentRecord.objects.filter(news=obj,
depth=2,reply_id__in=first_id_list)
        # 2. 获取所有1级评论下的二级评论(每个二级评论只取最新的一条)
        from django.db.models import Max
```

```
        result = models.CommentRecord.objects.filter(news=obj, depth=2,
reply_id__in=first_id_list).values('reply_id').annotate(max_id=Max('id'))
        second_id_list = [item['max_id'] for item in result] # 5, 8

        second_queryset =
models.CommentRecord.objects.filter(id__in=second_id_list).values(
            'id',
            'content',
            'depth',
            'user__nickname',
            'user__avatar',
            'create_date',
            'reply_id',
            'reply__user__nickname'
        )

        import collections
        first_dict = collections.OrderedDict()
        for item in first_queryset:
            item['create_date'] = item['create_date'].strftime('%Y-%m-%d')
            first_dict[item['id']] = item

        for node in second_queryset:
            first_dict[node['reply_id']]['child'] = [node,]

        return first_dict.values()

    def get_is_favor(self,obj):
        # 1. 用户未登录
        user_object = self.context['request'].user
        if not user_object:
            return False

        # 2. 用户已登录
        exists =
models.NewsFavorRecord.objects.filter(user=user_object,news=obj).exists()
        return exists
```

注意：序列化的方法中获取request

```
class TestSER(serializers.ModelSerializer):
    title = serializers.SerializerMethodField()
    class Meta:
        model = models.Topic
        fields = "__all__"

    def get_title(self,obj):
        request = self.context['request']
```

2.3 认证

- 编写

```
```

```
from rest_framework.authentication import BaseAuthentication
from api import models
from rest_framework import exceptions


class GeneralAuthentication(BaseAuthentication):
    """
    通用认证，如果认证功能则返回数据，认证失败自己不处理，交给下一个认证组件处理。
    """
    def authenticate(self, request):
        token = request.META.get('HTTP_AUTHORIZATION', None)
        # 1.如果用户没有提供token,返回None（我不处理，交给下一个认证类处理，则默认是None）
        if not token:
            return None
        # 2.token错误，,返回None（我不处理，交给下一个认证类处理，则默认是None）
        user_object = models.UserInfo.objects.filter(token=token).first()
        if not user_object:
            return None
        # 3.认证成功
        return (user_object,token) # request.user/request.auth


class UserAuthentication(BaseAuthentication):
    """
    用户认证，用户必须先登录。
    """
    def authenticate(self, request):
        token = request.META.get('HTTP_AUTHORIZATION', None)
        # 1.如果用户没有提供token,返回None（我不处理，交给下一个认证类处理，则默认是None）
        if not token:
            raise exceptions.AuthenticationFailed()
        # 2.token错误，,返回None（我不处理，交给下一个认证类处理，则默认是None）
        user_object = models.UserInfo.objects.filter(token=token).first()
        if not user_object:
            raise exceptions.AuthenticationFailed()
        # 3.认证成功
        return (user_object,token) # request.user/request.auth
```

- 应用认证类
  - 全不需要登录
  - 全需要登录
  - 部分需要登录，重写方法

```
class CommentView(APIView):

    def get_authenticators(self):
        if self.request.method == 'POST':
            return [UserAuthentication(), ]
        return [GeneralAuthentication(), ]

    def get(self,request,*args,**kwargs):
        root_id = request.query_params.get('root')
        # 1. 获取这个根评论的所有子孙评论
```

```
        node_queryset = models.CommentRecord.objects.filter(root_id=root_id).order_by('id')
        # 2. 序列化
        ser = CommentModelSerializer(instance=node_queryset,many=True)

        return Response(ser.data,status=status.HTTP_200_OK)

    def post(self,request,*args,**kwargs):
        # 1. 进行数据校验: news/depth/reply/content/root
        ser = CreateCommentModelSerializer(data=request.data)
        if ser.is_valid():
            # 保存到数据库
            ser.save(user_id=1)
            # 对新增到的数据值进行序列化(数据格式需要调整)
            news_id = ser.data.get('news')

 models.News.objects.filter(id=news_id).update(comment_count=F('comment_count')+1)

            return Response(ser.data,status=status.HTTP_201_CREATED)
        return Response(ser.errors,status=status.HTTP_400_BAD_REQUEST)
```

2.4 状态码

```
from rest_framework import status

例如：
    Response(serializer.data, status=status.HTTP_201_CREATED, headers=headers)
```

3.其他

- js文件导入
- 三元运算

  ```
  条件 ? 真 : 假
  ```

- js闭包
- 自执行函数
- this 和 箭头函数（es6）
- django orm
  - F
  - 分组
  - 排序
  - bluk_create

```
result1 = models.UserInfo.objects.create(**{})
result2 = models.UserInfo.objects.create(**{})

print(result1.id,result1.name,result2.id,result2.name,)

data = models.UserInfo.objects.blukcreate([
    models.UserInfo(...),
    models.UserInfo(...)
])
data[0].name/title/email/ ->  id
```

  - get_or_create
  - 自关联（related_name）
- 基于Token做的认证、jwt
- 腾讯对象存储 COS
- 腾讯短信API
- django中编写离线脚本

```
import os
import sys
import django

base_dir = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
sys.path.append(base_dir)

# 将配置文件的路径写到 DJANGO_SETTINGS_MODULE 环境变量中
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "demos.settings")
django.setup()

from api import models
models.Topic.objects.create(title="春运")
models.Topic.objects.create(title="火车票")
```

注意：可以结合定时任务。