from accounts.api.serializers import UserSerializer

twitter-clone-resume-app

clone twitter for resume-app project

记个笔记吧，记忆力不太好，怕忘了，回顾的时候一篇茫然

::

1-3缺省

1-3主要在考虑建一个什么样的app项目，思来想去还是去先打基础吧，之前买的科还没学完，先去搞明白前人的智慧到底在说什么，因此，1-3就是在创建项目

- 下载pycharm pro ,python
- 创建django项目,起名`resume_app`(之前的想法whatever)
- 配置数据库`postgresql`,这次来点不一样的，拜拜mysql

就这么多啦，下面开始学习前人的智慧

4-user-authentication-api

配置了`djangorestframework` 简历路由替代django默认路由，localhost:8000打开就是DRF的页面

```
from rest_framework.routers import DefaultRouter
from accounts.views import UserViewSet
from django.urls import include,path

router = DefaultRouter()
router.register("/api/users", UserViewSet)
url_pattern=[
#...
path("",include(router.urls)),
path("user-api", include("rest_framework.urls", namespace='rest_framework'))
]
```

创建视图和序列化器 `UserViewSet`和`UserSerializer`

```
from rest_framework import viewsets
from accounts.api.serializers import UserSerializer
from django.contrib.auth.models import User

class UserViewSet(viewsets.ModelViewSet):
```

```
        serializer_class = UserSerializer
        queryset = User.objects.all()
```

```
from rest_framework.serializers import ModelSerializer
from django.contrib.auth.models import User
class UserSerializers(ModelSerializer):
    class Meta:
        model = User
        fields = ("username", "email", "password")
```

要注意的是，以上的两个东西是restframework的核心 继承关系是：

```
view (Django)
  |
  └── View (Django)
      └── APIView (DRF)
          └── GenericAPIView
              ├── Mixins
              |    ├── CreateModelMixin
              |    ├── RetrieveModelMixin
              |    ├── UpdateModelMixin
              |    ├── DestroyModelMixin
              |    └── ListModelMixin
              |
              └── ViewSets
                   ├── ViewSet
                   ├── GenericViewSet
                   └── ModelViewSet
```

继承关系说明：

1. 所有的视图都继承自APIView,二APIView继承了django的view.View类
2. APIView的作用是 蜜汁源码，暂不关心
3. GenericAPIView继承自APIView增加了 serializer_class, queryset的支持

```
from rest_framework.views import APIView

class GenericAPIView(APIView):
    serializer_class = None
    queryset = None
    def get_serializer_class(self):
        pass
    def get_queryset(self):
        pass
```

4. Mixins是一个独立的类，路数变化方面，结合了GenericAPIView变成了专门用于list,retrieve,create方法的子类
    - CreateModelMixin + GenericAPIView = CreateAPIView
    - ListModelMixin + GenericAPIView = ListAPIView
    - UpdateModelMixin + GenericAPIView = UpdateAPIView
    - RetrieveModelMixin + GenericAPIView = RetrieveAPIView
5. ViewSets 是一个包名，下辖几个视图集
    - ViewSet: APIView + ViewSetMixin, 但是ViewSetMixin是个蜜汁类，暂时不关心
    - GenericViewSet: GenericAPIView + ViewSetMixin
    - ModelViewSet: GenericViewSet +
        - mixins.ListModelMixin
        - mixins.CreateModelMixin
        - mixins.RetrieveModelMixin
        - mixins.UpdateModelMixin
        - mixins.DestroyModelMixin
    - ReadOnlyModelViewSet: 蜜汁类，暂不关心

小结一下：GenericViewSet最常用，继承了GenericAPIView,因此支持serializer_class和queryset，然后mixins里的五种方法加持下， 变成了一种自定义的强大的视图集，后面的tweetViewSet里可以看到具体用法。

5-account-api-unit-tests

6-tweet-model

7-tweet-api-and-tests

8-friendships-model-api-and-tests

1. `FriendShip` 模型是让两个用户产生关联关系的模型
    - 在模型中follower是发起者，following是接受者，
    - 在url对应的视图中
        - `friendship/{pk}/follower`请求follower视图,意味着请求pk的followers,也就是这个用户的追随者，传参时`following=pk`, 无需登录就可查看有哪些追随者；
        - `friendship/{pk}/following`请求following视图，意味着查看pk用户正在关注哪些对象，`follower=pk`；
        - `friendship/{pk}/follow` 请求follow视图，必须要登录的状态下，请求时`pk!=request.user.id`，因为自己无法关注自己, `follower=request.user`, `following=pk`,这样就可以建立两者的关系。

名称	view	model	url
follower	关系发起的人	发起人为用户	查看pk的follower
following	关系指向的人	指向人为用户	查看pk的following

2. 要理解业务模型的意义，才能指定测试的策略，否则一脸懵逼
3. 没有太多技巧，这一分支主要是理解业务字段

9-newsfeed-model-api-tests

1. 外键的字段会生成对应`外键_id`阶段，这样可以传入外键的实例或者外键的主键id值
2. 测试阶段，创建一个用户的客户端不同于创建用户的实例，两者完全不相关

10-comments-model-admin

1. 不明白注释里说的 这个版本中，我们先实现一个比较简单的评论 评论只评论在某个Tweet上，不能评论别人的评论， azhen可以评论aqiang的推文，难道你要实现的是非好友也能评论吗？

11-comment-create-api

1. Q:user,user_id,tweet,tweet_id傻傻分不清楚，什么时候传实例，什么时候传id? A: 前后端约定了传什么就传什么
2. 要细心一点，serializer.create方法评论内容都没保存，创建了个寂寞

12-comment-update-and-destroy-api

1. `update`被路由默认绑定,而不是在视图中显式指定detail=True,类似的方法还有`retrieve`,`delete`
2. 学到一招：serializer的作用还可以告诉接口，我接受哪些model字段进行验证，即使用户传来了一些不需要的字段，`validate()`方法可以忽略不验证也不会报错。

13-comments-list-api

1. `django-filter`应该是**`django_filter`**,我槽，这么个小坑，根本无法察觉
2. 所有的方法都会被`permission_class`过滤，所以才需要设置`get_permissions()`来定义哪些情况可以豁免

15-likes-model-and-admin

1. 创建Likes时 content_object 是不需要传递的，直接拿到django-admin里展示，会展示某个模型的实例的__str__
2. Comment的@property的属性Likes.objects.filter(`object_id,content_type`)找到的是该条评论的**所有**likes

16-likes-create-api

1. 理解更深刻了，like_set是实例方法，只有Tweet的实例才有count()熟悉和数据
2. 一个用户对一个tweet发送多次like,只算一次，因为，unique_index 规定了 user,content_type,object_id
3. 用户传递字符串 tweet,被手动映射为Tweet模型，是在serializer里完成的，没什么神奇的
4. get_or_create方法更稳健，如果直接create,就会傻傻的创建另一个重复的记录，直接引发数据库unique index报错。
5. 测试的时候client会带用户信息，这一点要记得下次写tests时有思路
6. permission_class如果不指定，匿名用户会引发代码报错，所以必须要检查
7. require_params装饰器方法的巧妙之处，现在有进一步了解，他是静态的定义所需要的参数，然后检查request.data里是否有需要的参数

17-likes-cancel-api

1. GenericViewSet中的create方法自动映射为http的post请求，@action装饰的方法自动映射函数名为url

2. 用户没点赞，但是如果请求取消点赞也不会报错
3. 点赞需要登录之后才能操作

18-inject-like-info-to-other-api

1. 这个分支把tweet-comment-like串联起来，做到了一次查询，所有tweet相关的comment和like都查询出来，包括comment的like也能查询出来

2. 所以TweetSerializer增加了`comment_count`,`like_count`和`has_liked`, `CommentSerializer`增加了`like_count`和`has_liked`

Serializer	comment_count	like_count	has_liked
CommentSerializer	✗	✔	✔
TweetSerializer	✔	✔	✔

3. 测试的url分为两个维度，tweet维度和comment维度，分别看到的内容不一样

维度	list	detail	anonymous
tweet	has_liked		get
tweet	likes_count		get
tweet	comment_count		get
tweet	user		get
tweet		comments	get
tweet		likes	get
comment	has_liked		get
comment	likes_count		get
newsfeed	内嵌 tweet		get