Процесс проверки проекта на диф. зачете

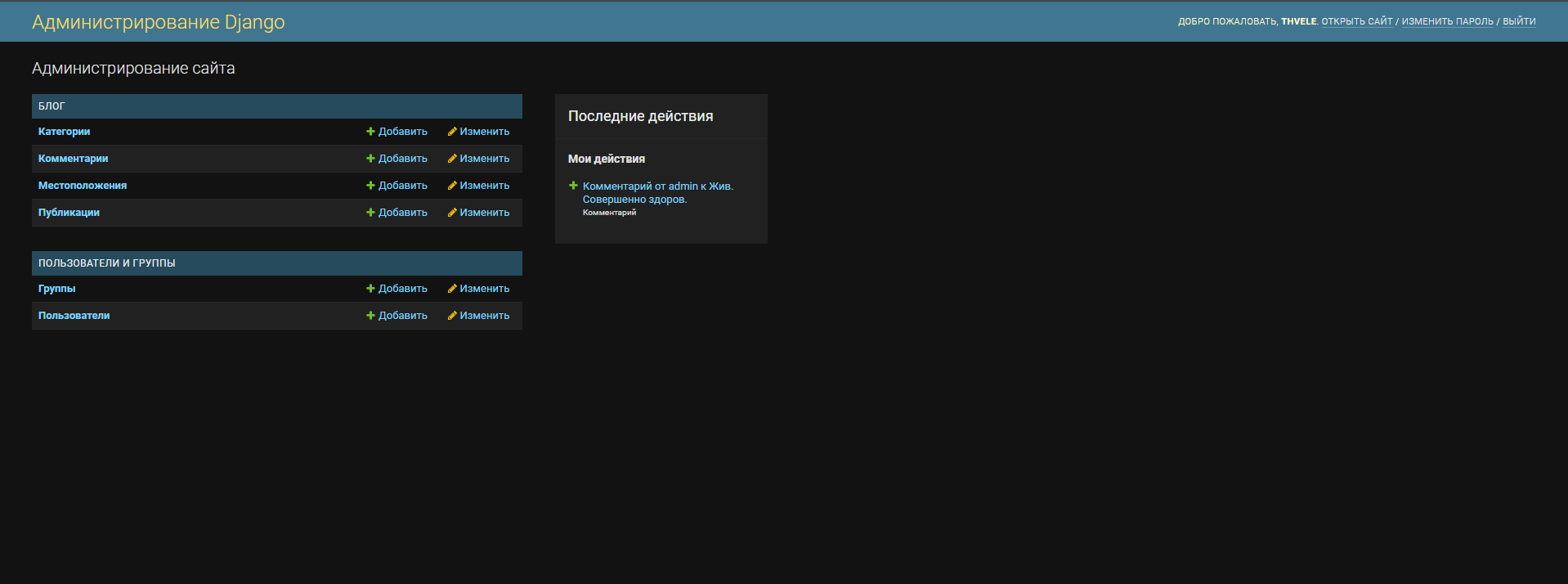
До дня проведения зачета должны быть успешно пройдены все автотесты разделов: Создание проекта, Базы данных, Тестирование на Python.

Итоговый проект, из темы с называнием «Проект. Приложение «Блогикум». Часть 3. Доработка», должен быть передан преподавателю через «Яндекс Практикум».

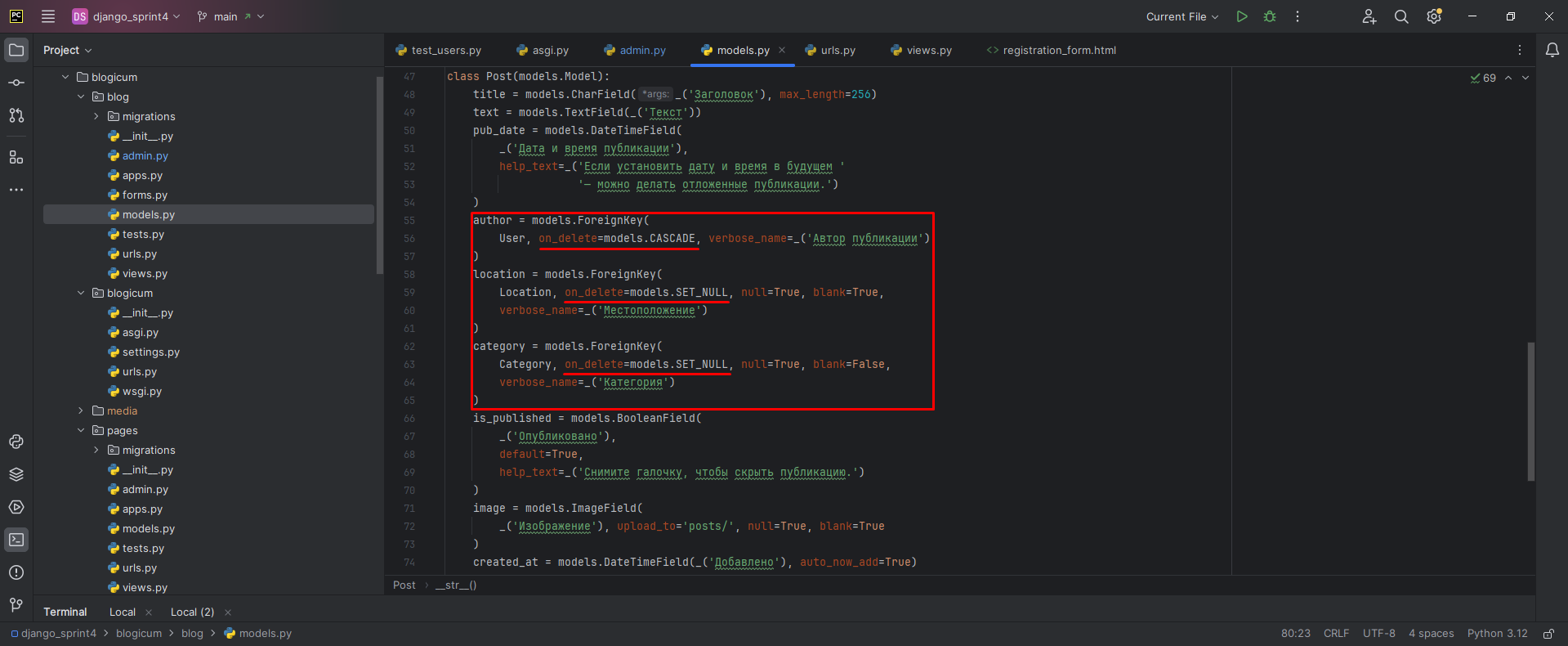
На зачёте показать преподавателю и пояснить назначение следующих позиций в вашем проекте:

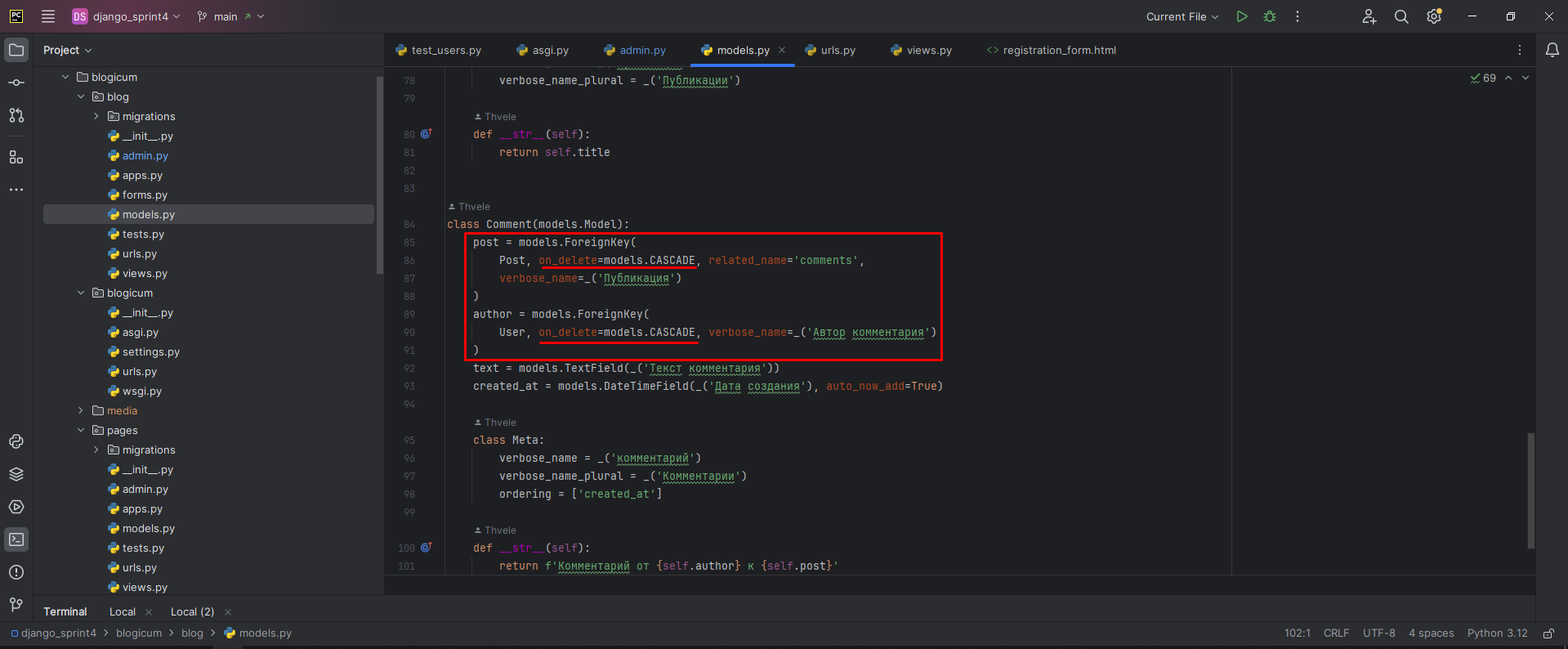
**Раздел 1**

1. Модели **Category**, **Location**, **Post**, **Comment** должны быть представлены в админке – **admin.py**

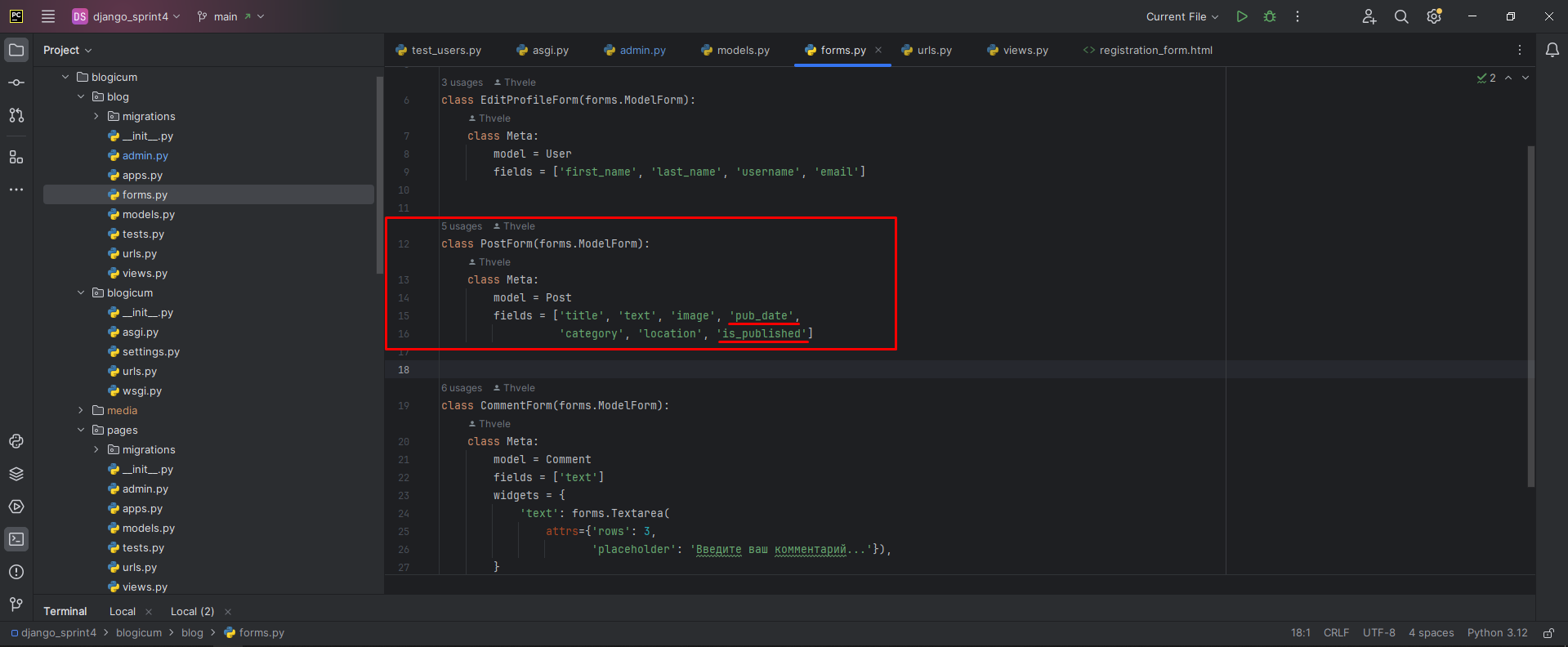


1. В моделях у всех полей для связи между таблицами с типом **ForeignKey** должны быть заданы параметры **on\_delete** . Для большинства — со значением **CASCADE** , для локации и категории — **SET\_NUL**L .

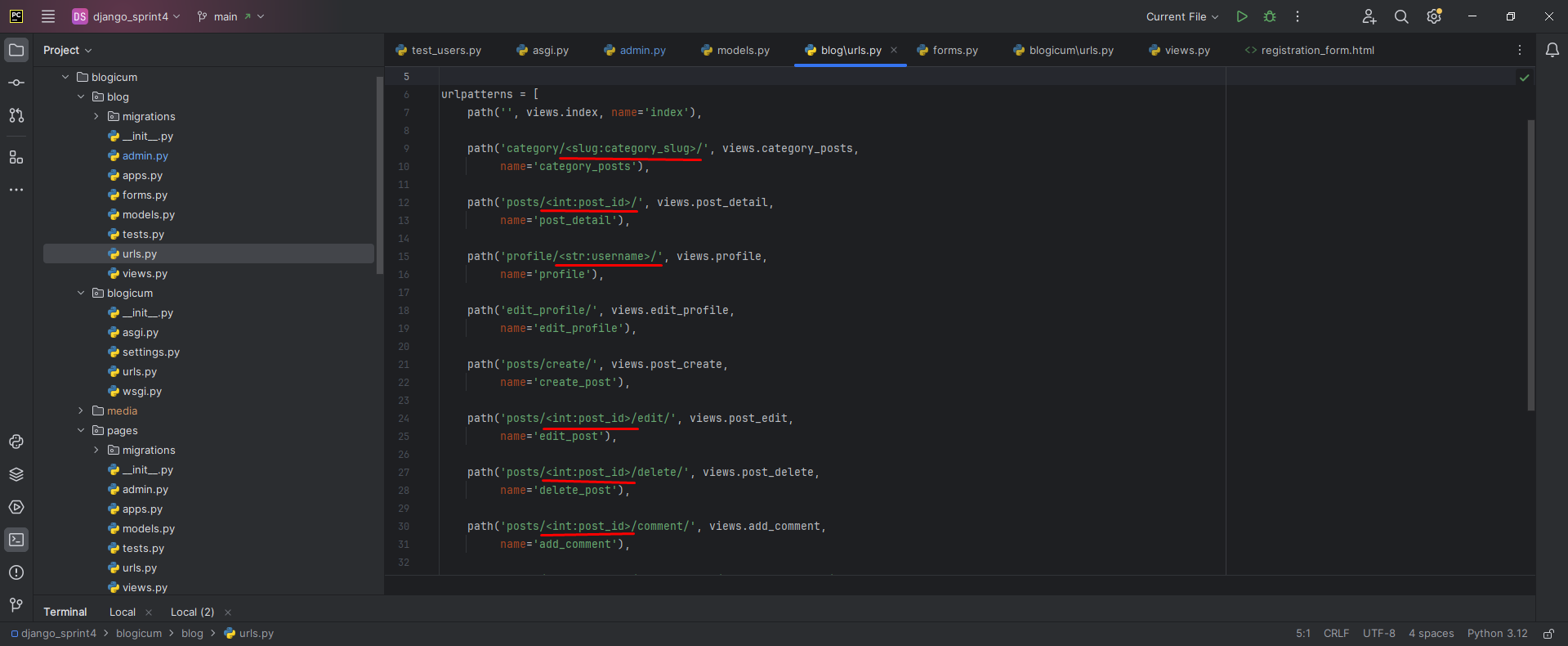




1. При настройке формы в **forms.py** для поста нужно предоставить на редактирование поля **is\_published**, **pub\_date** , чтобы автор смог изменять их: публиковать, откладывать, снимать с публикации конкретный пост.

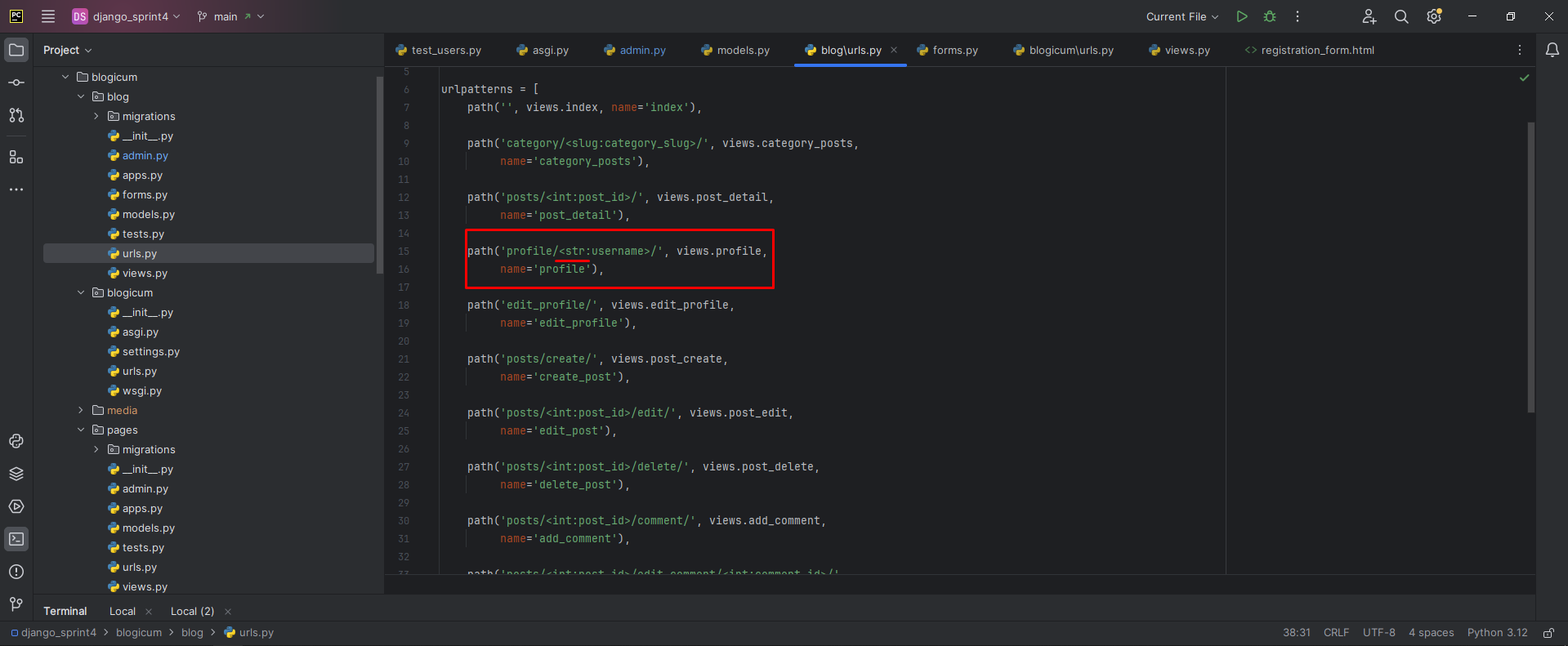


1. Извлекаемые из URL ключи — в **urls.py** — должны быть содержательными: **pk** или **id** не годятся. Нужно добавлять имя модели или таблицы.



1. В маршрутах, которые настраиваются в **urls.py** , для ника пользователя тип параметра в URL **не** может быть slug

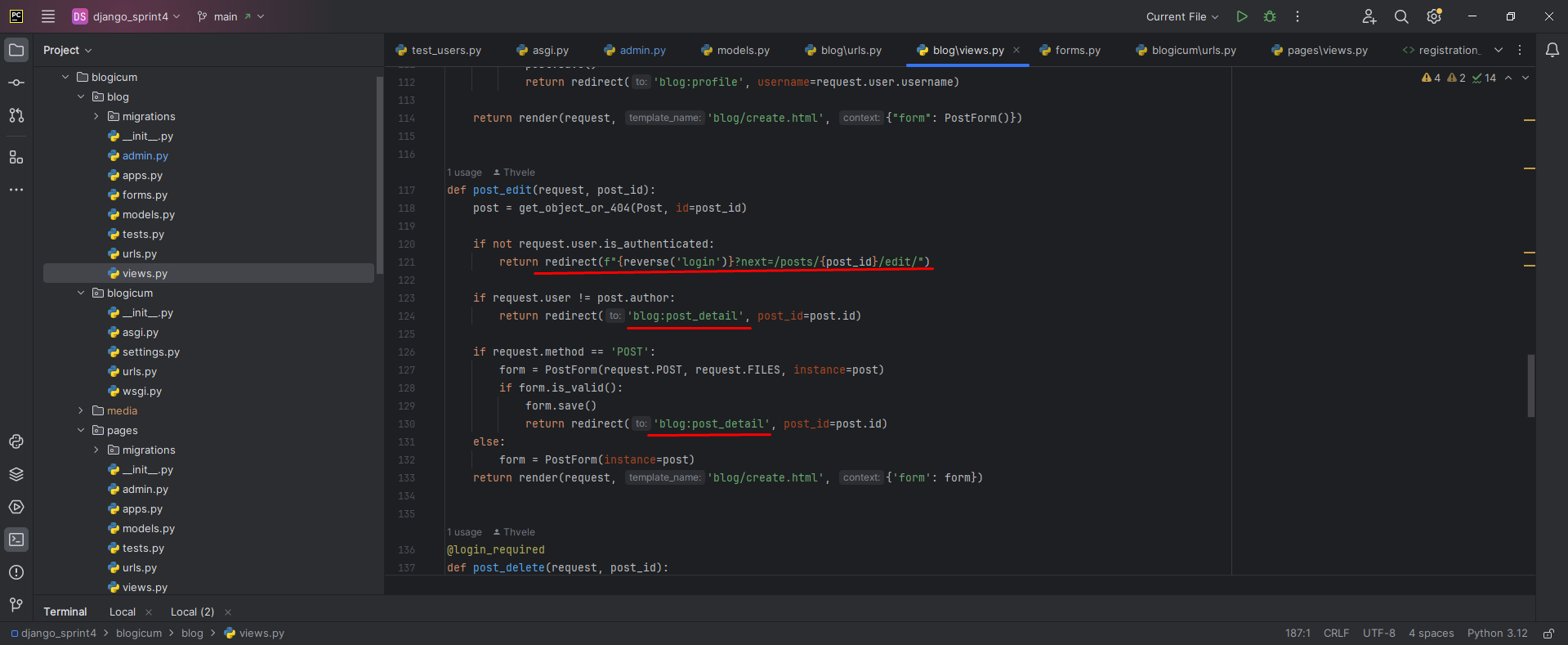
**path ('profile/<slug:username>/', ...)** , так как у слага мало вариантов.



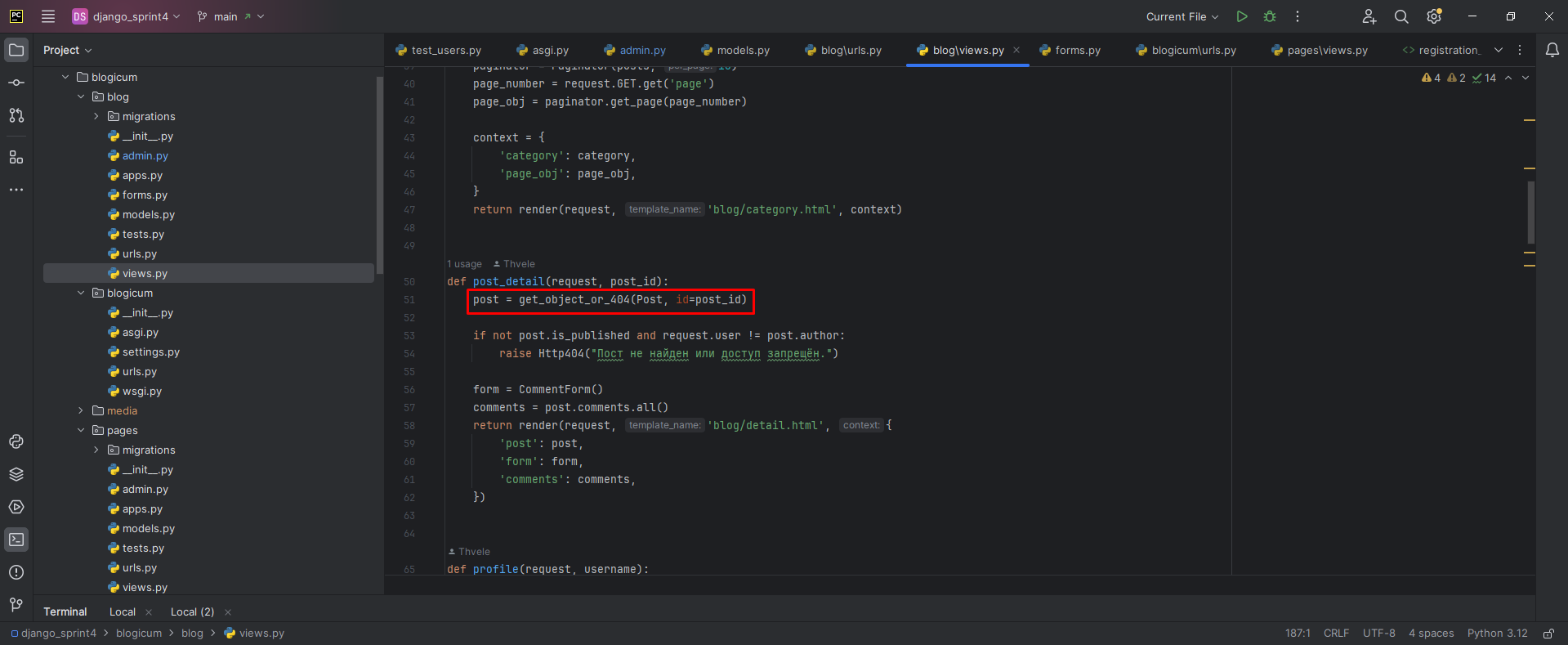
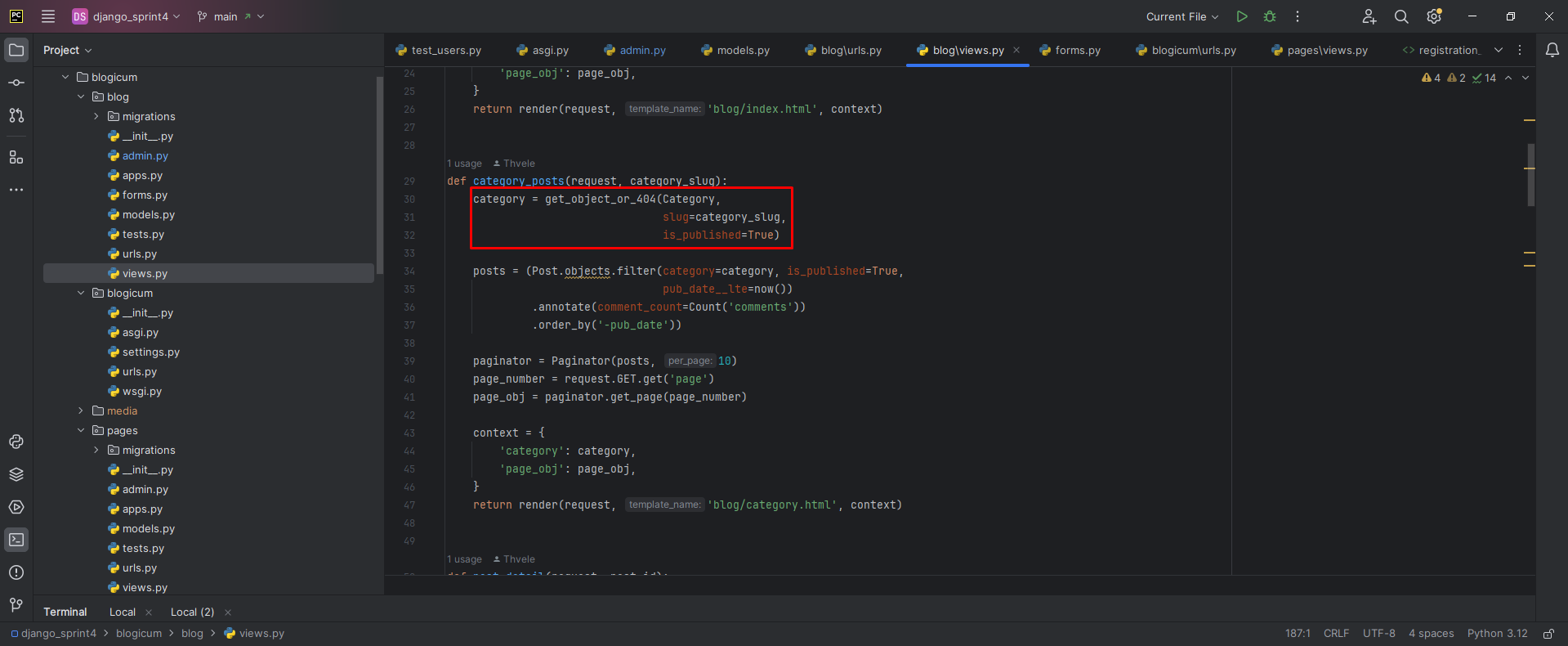
1. Явные URL – например, **'/'** или **'posts/create/'** или **'profile/oleg/'** — не могут применяться нигде, кроме вызовов **path()** в **urls.py** . Во всех остальных частях проекта URL должны вычисляться через имя маршрута, заданного в **path(url, контроллер, name='имя-маршрута')** .

Допустимые способы:

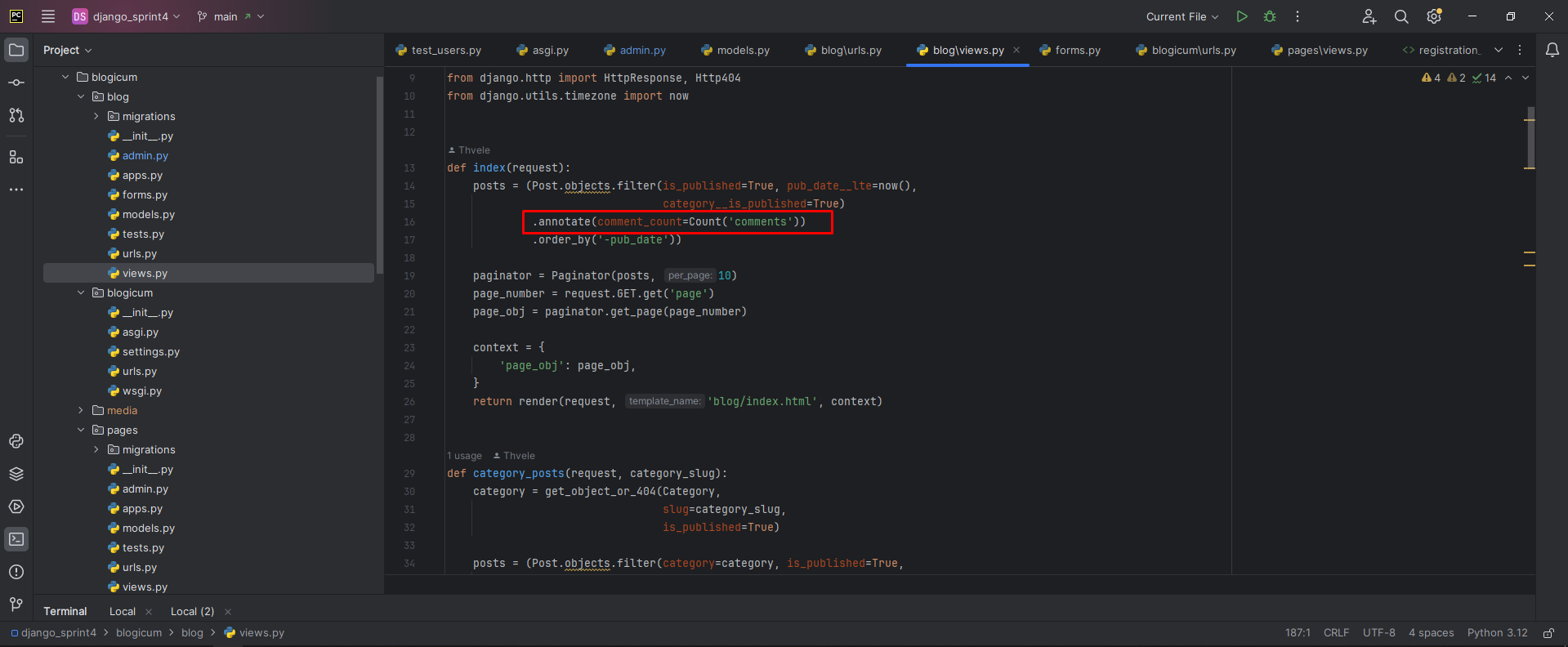
* В контроллерах в **views.py** через **reverse('имя-маршрута', параметр)** (можно и без параметра) или **redirect('имя-маршрута', параметр )** .
* В шаблонах через **{% url 'имя-маршрута' параметр %}** .

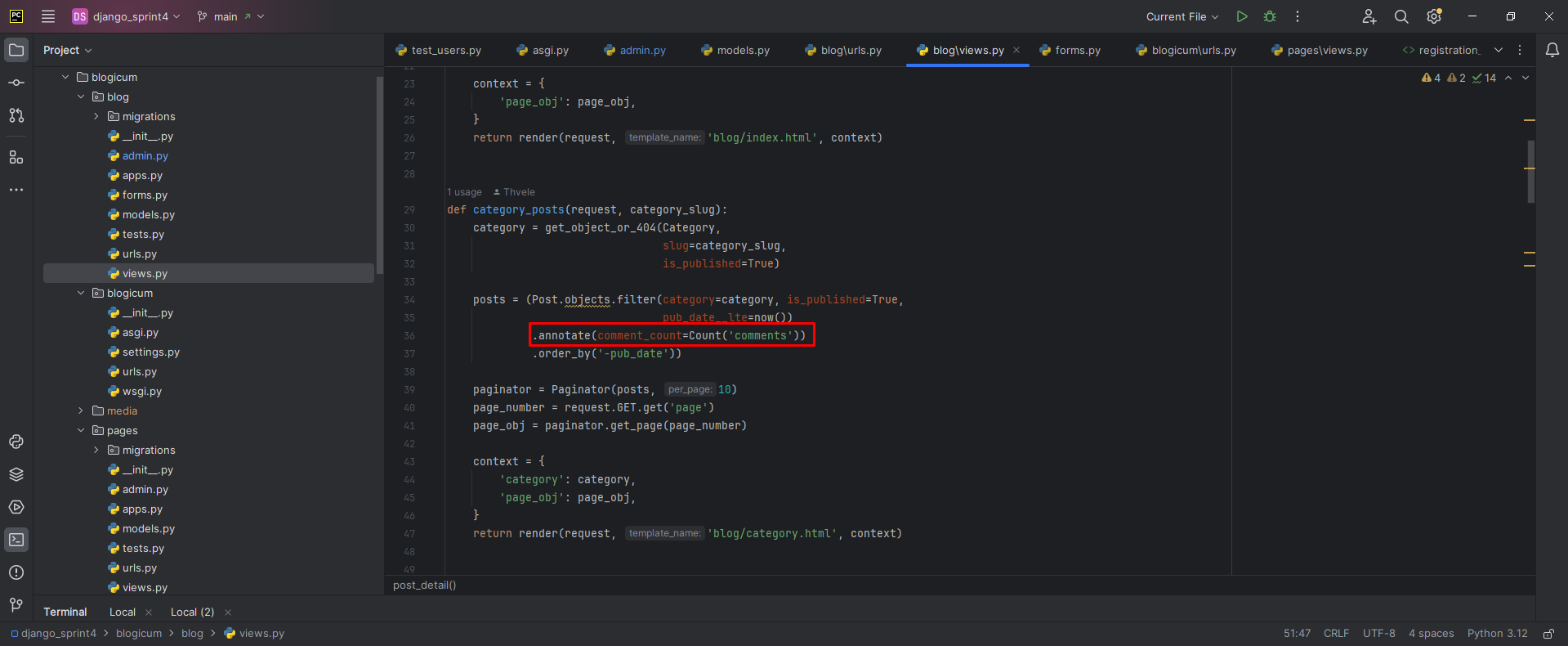


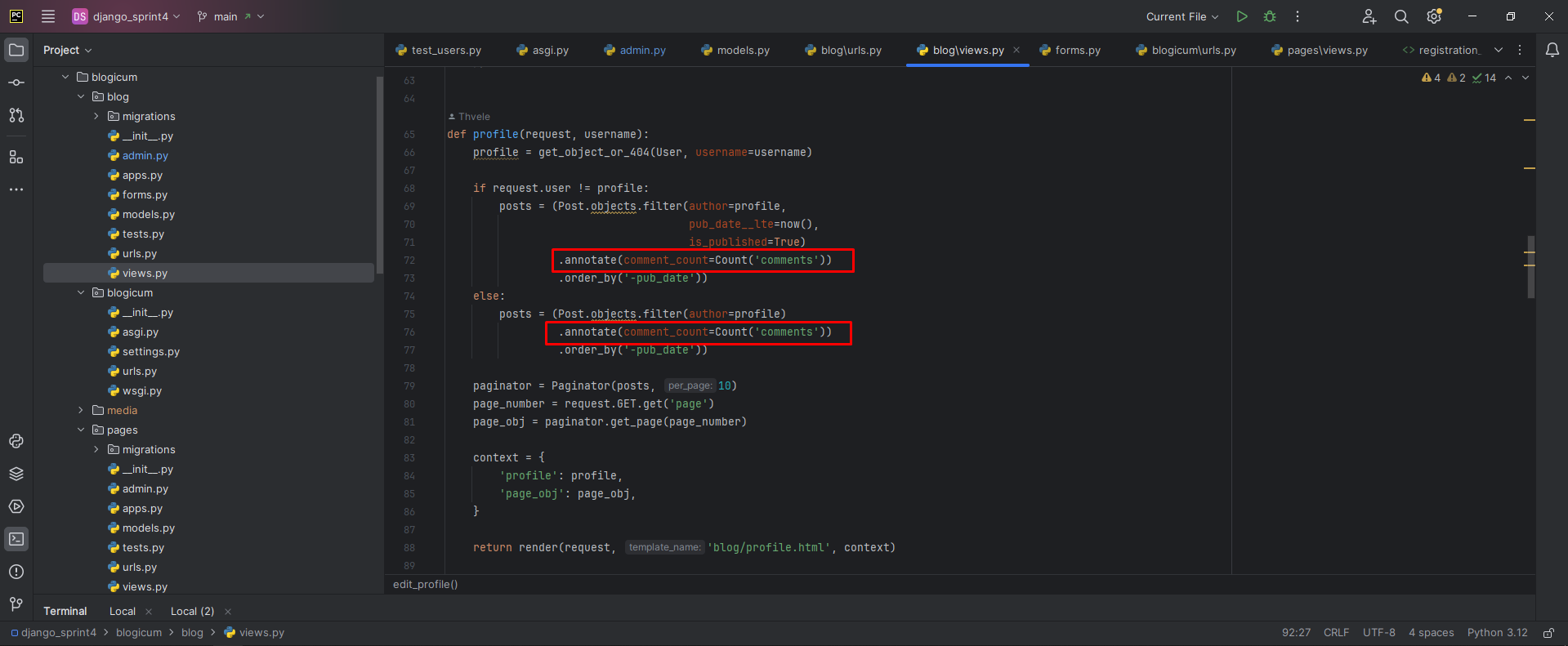
1. Извлечение поста, категории, комментария из базы выполняется только через вызов **get\_object\_or\_404()** .

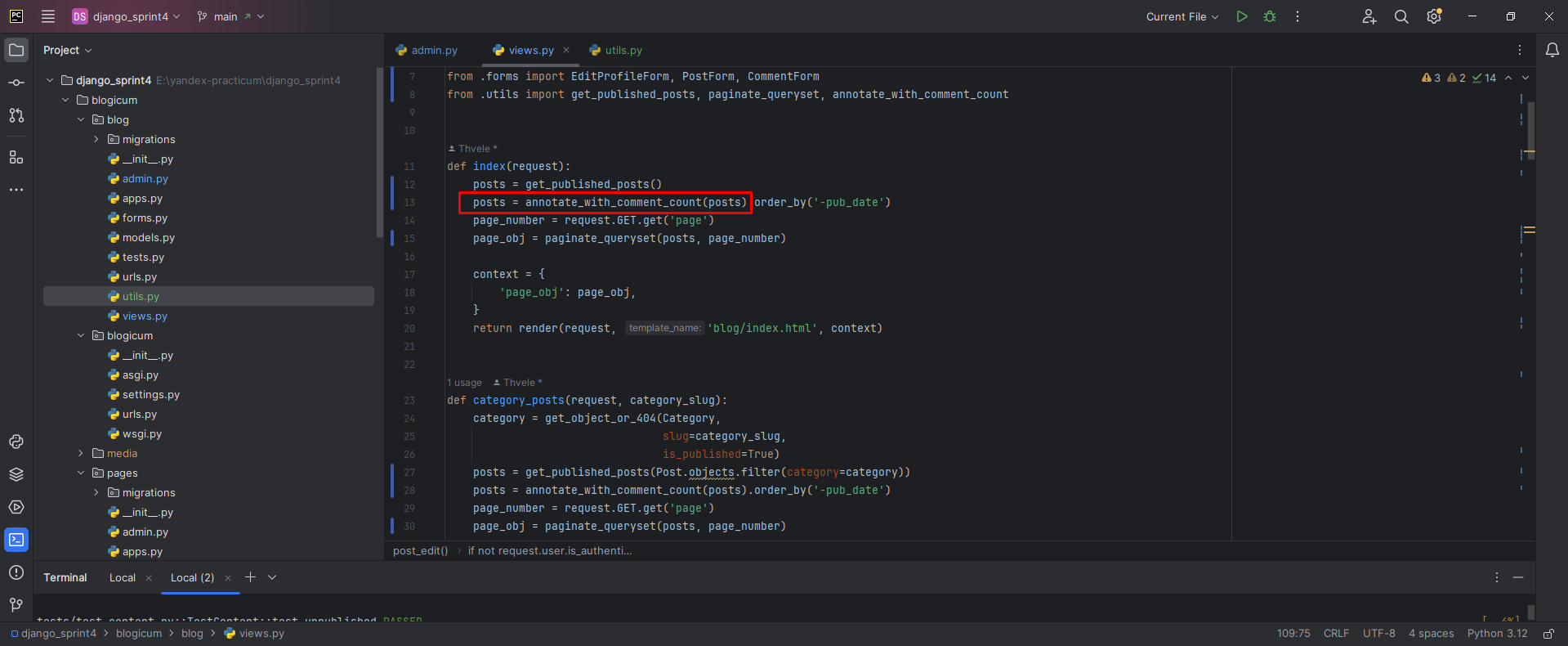
1. Все посты в наборах, попадающих на страницы «Главная», «Посты категории», «Посты автора», должны быть дополнены количеством комментариев.



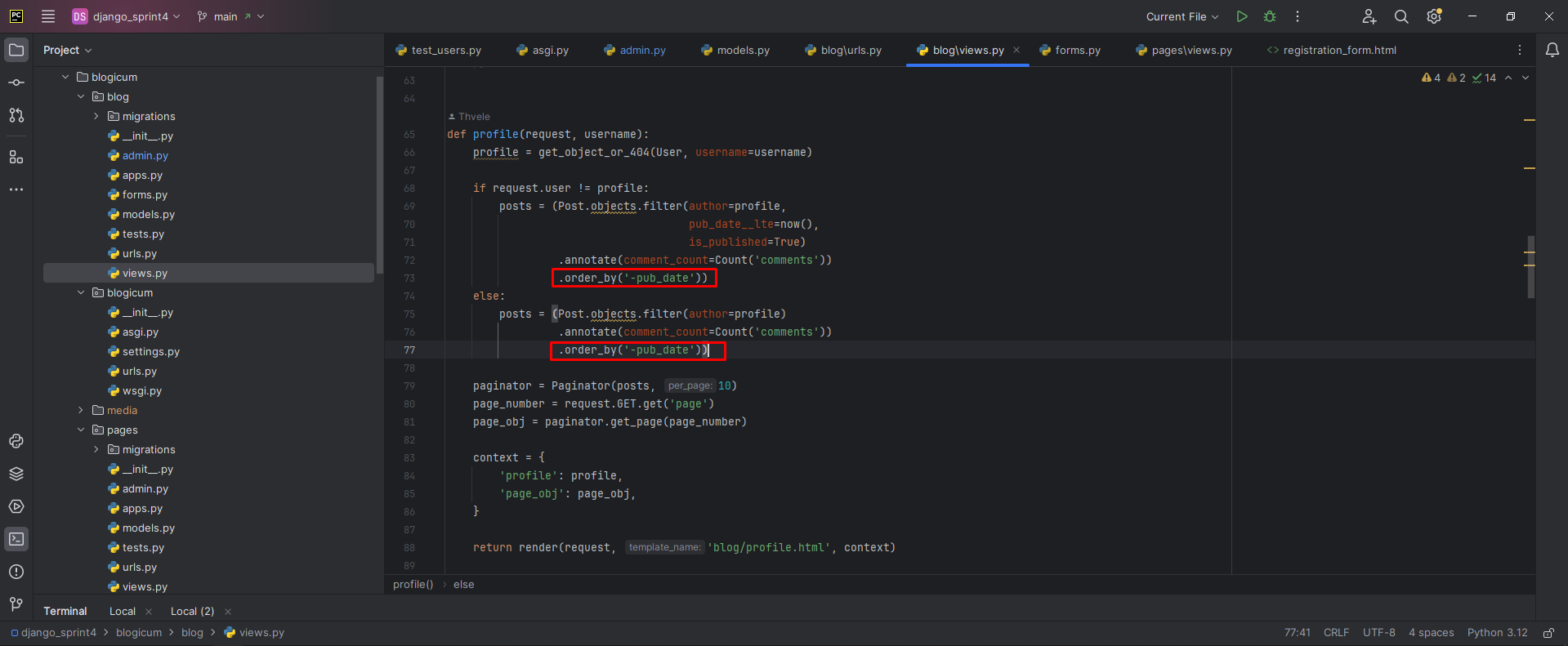




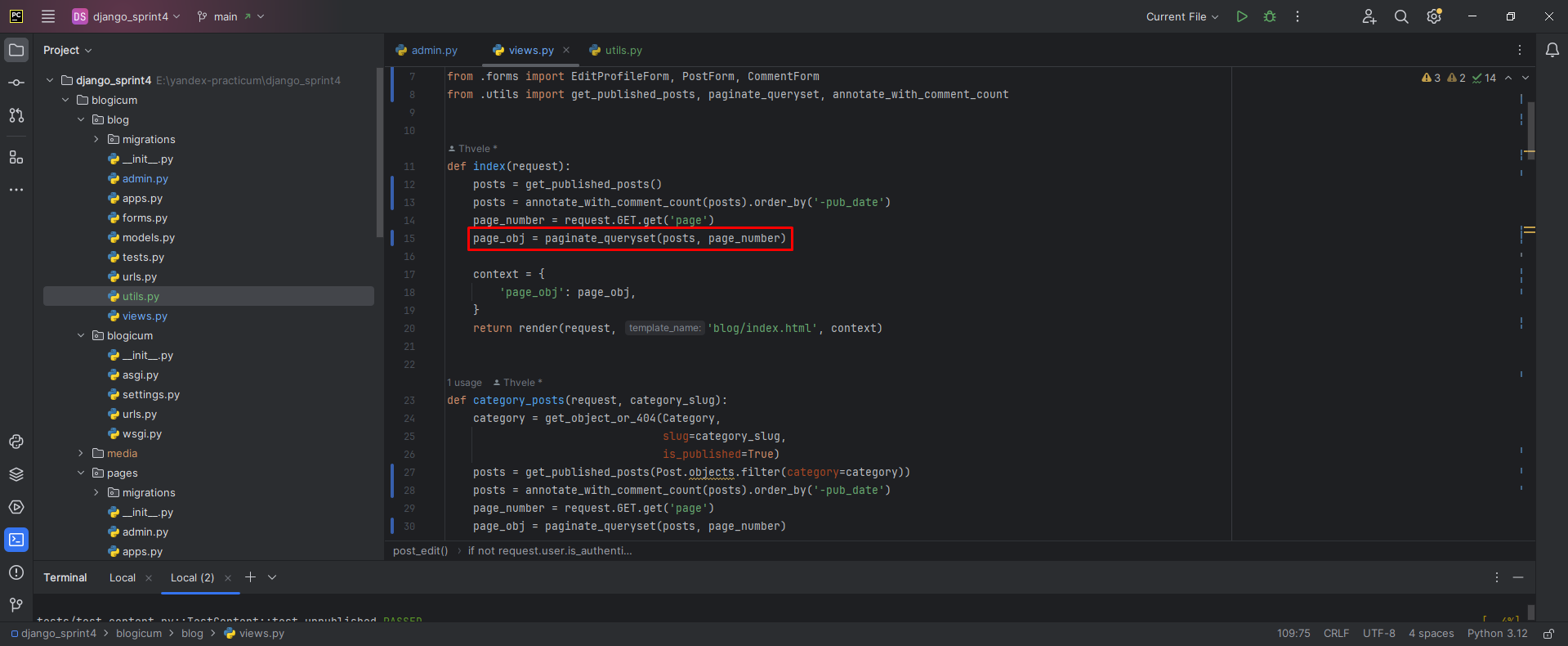
1. Вычисление количества комментариев к постам должно находится в единственном месте — в новой функции. Само вычисление выполняется методом кверисета **annotate()**.



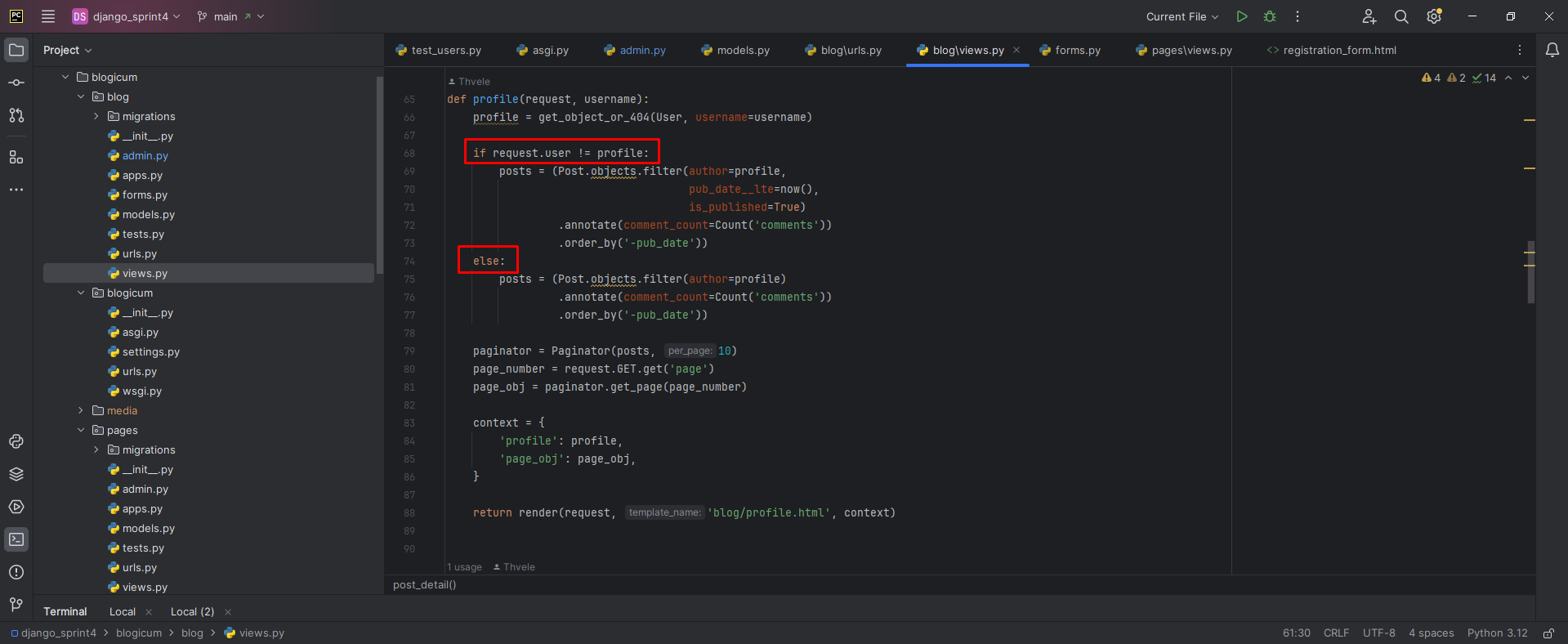
1. После вызова **annotate()** обязательно нужен вызов сортирующего метода: сортировка из модели уже будет неприменима, так как добавилось поле. Без сортировки пагинация работать не сможет.

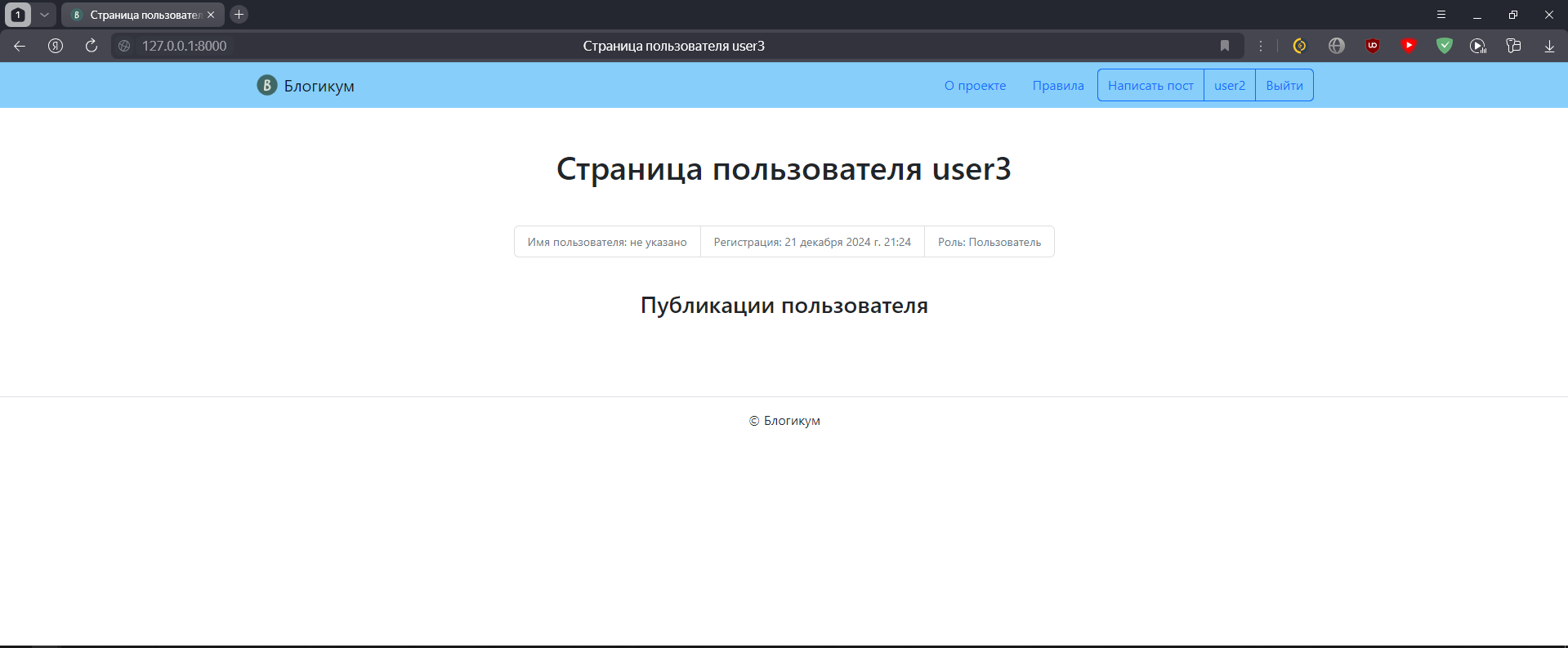


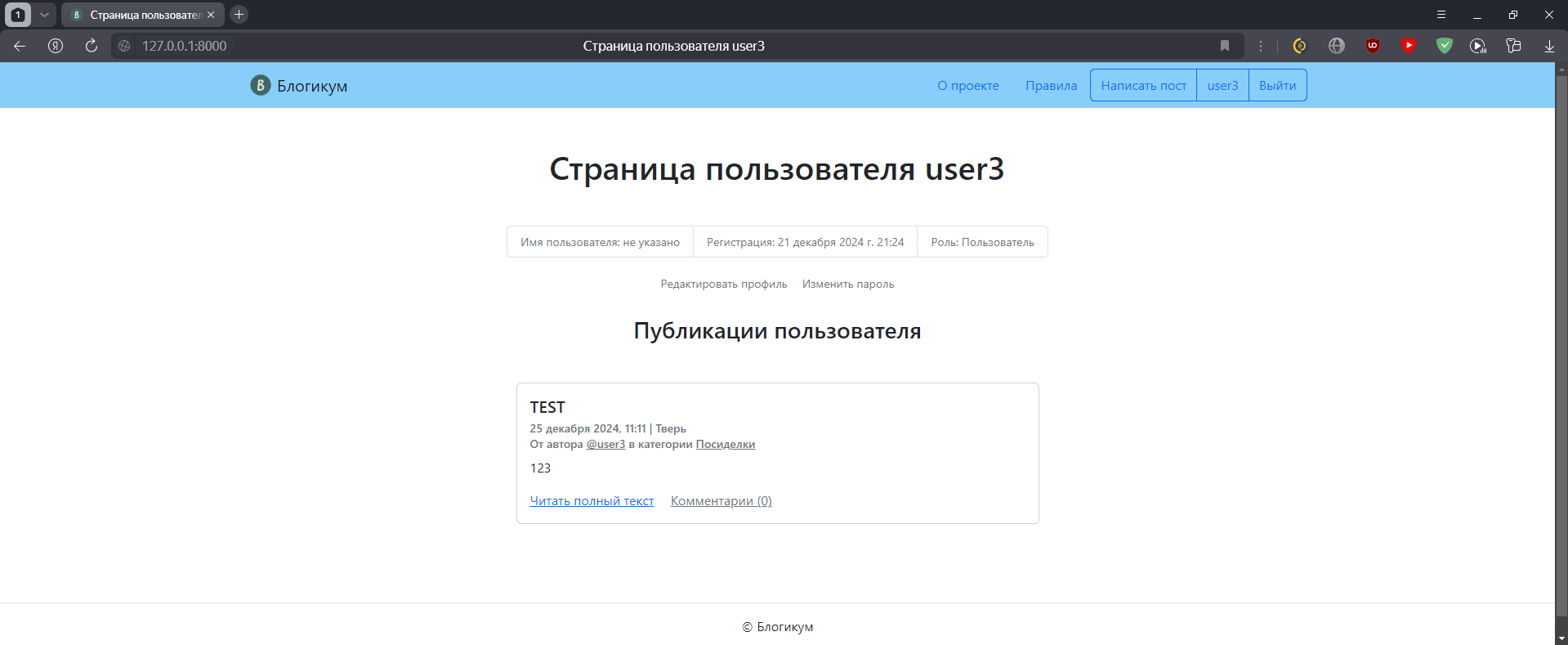
1. Вычисление одной страницы пагинатора нужно разместить в новой функции.



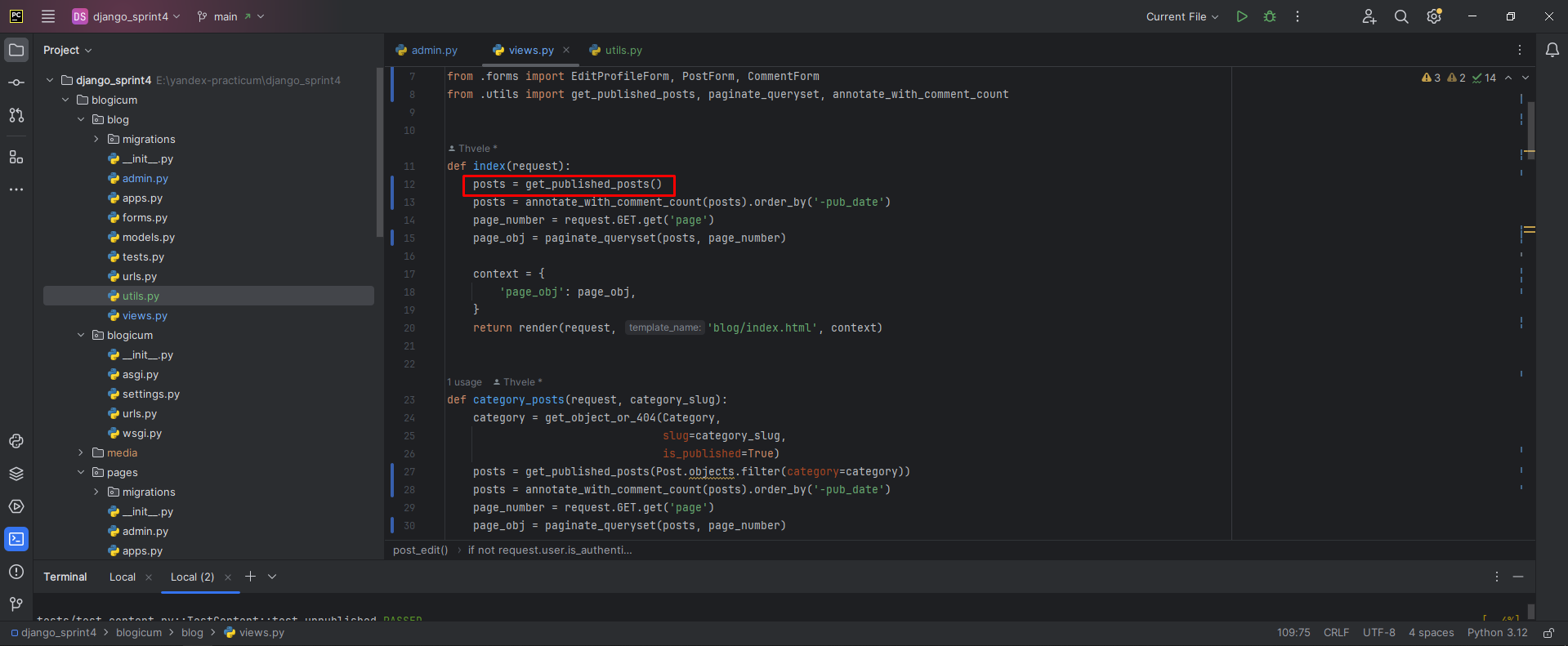
1. Набор постов на странице автора должен зависеть от посетителя. Только посетитель-автор может видеть свои неопубликованные посты.





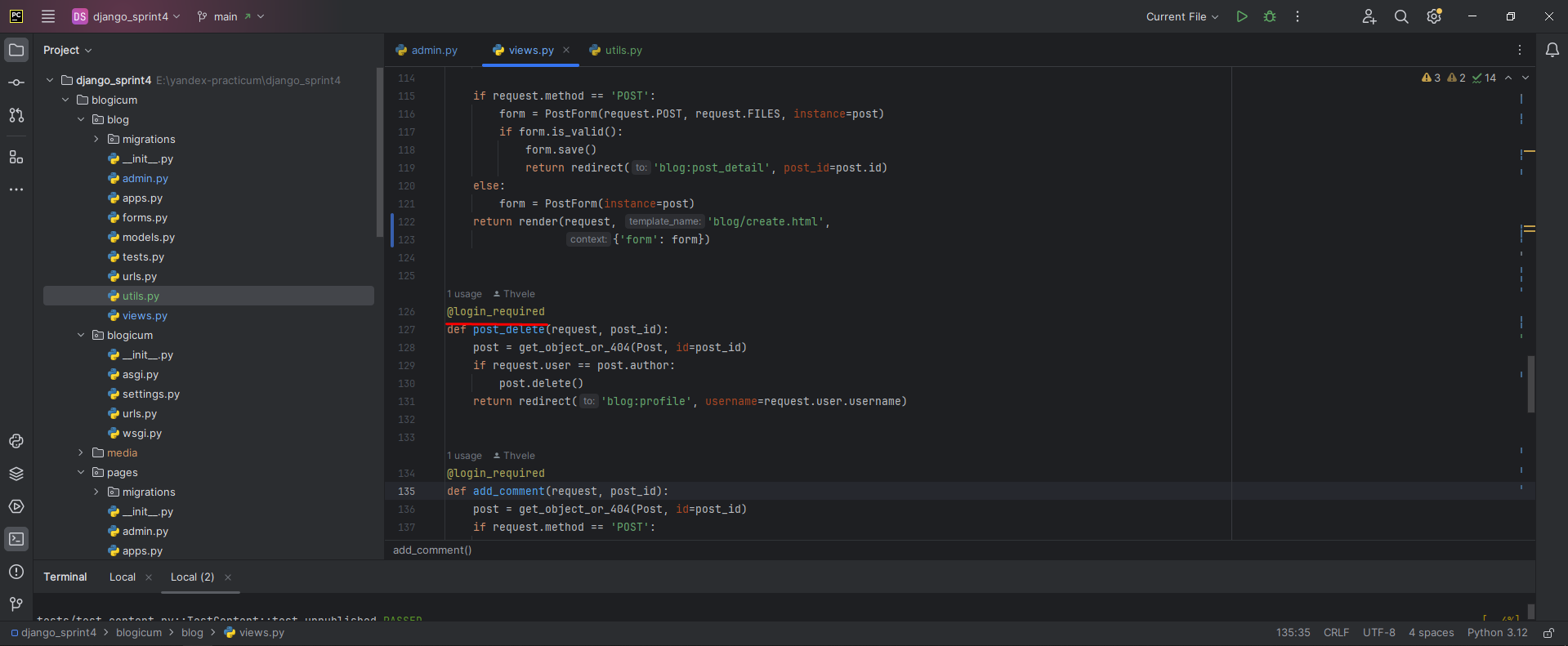


1. Фильтрация записей из таблицы постов по опубликованности должна размещаться в новой функции.

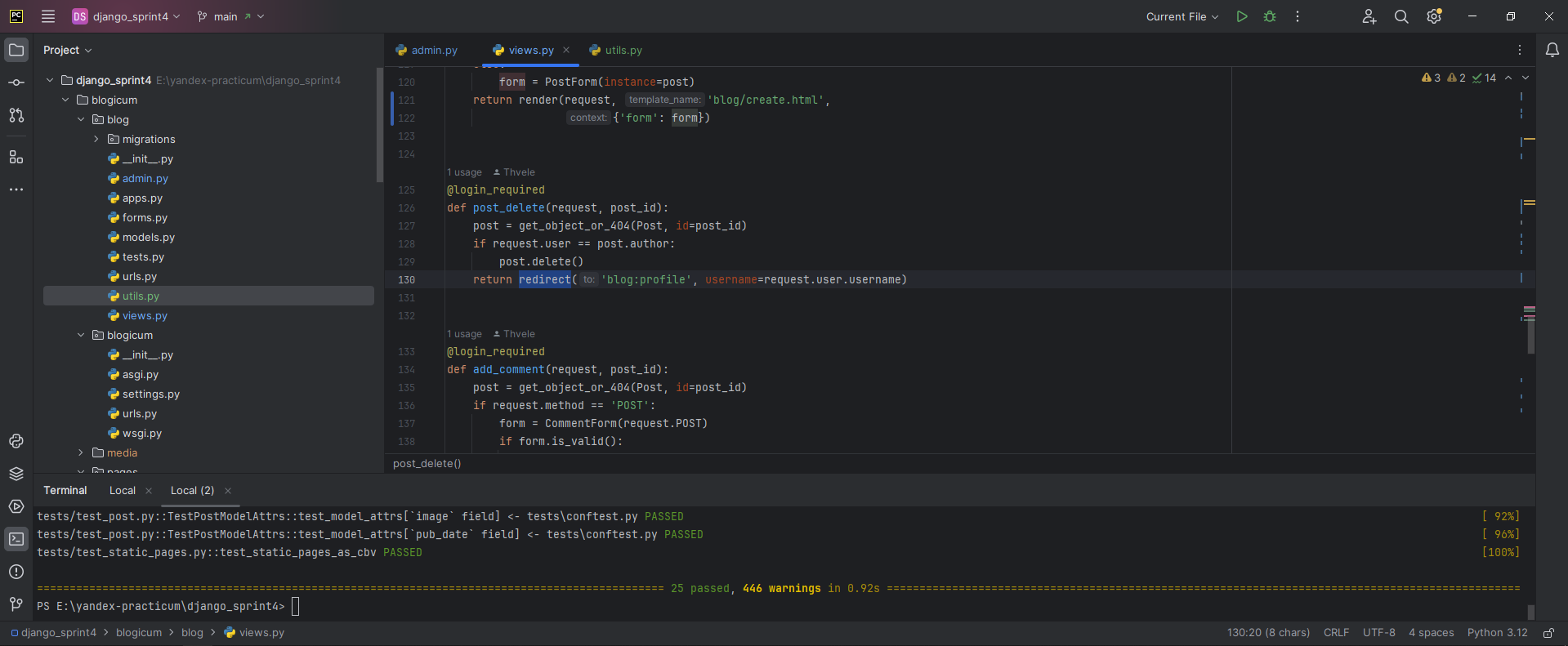


1. Контроллер-функции для создания, редактирования, удаления работают не с анонимом. Нужно применить Django-декоратор **@login\_required** .

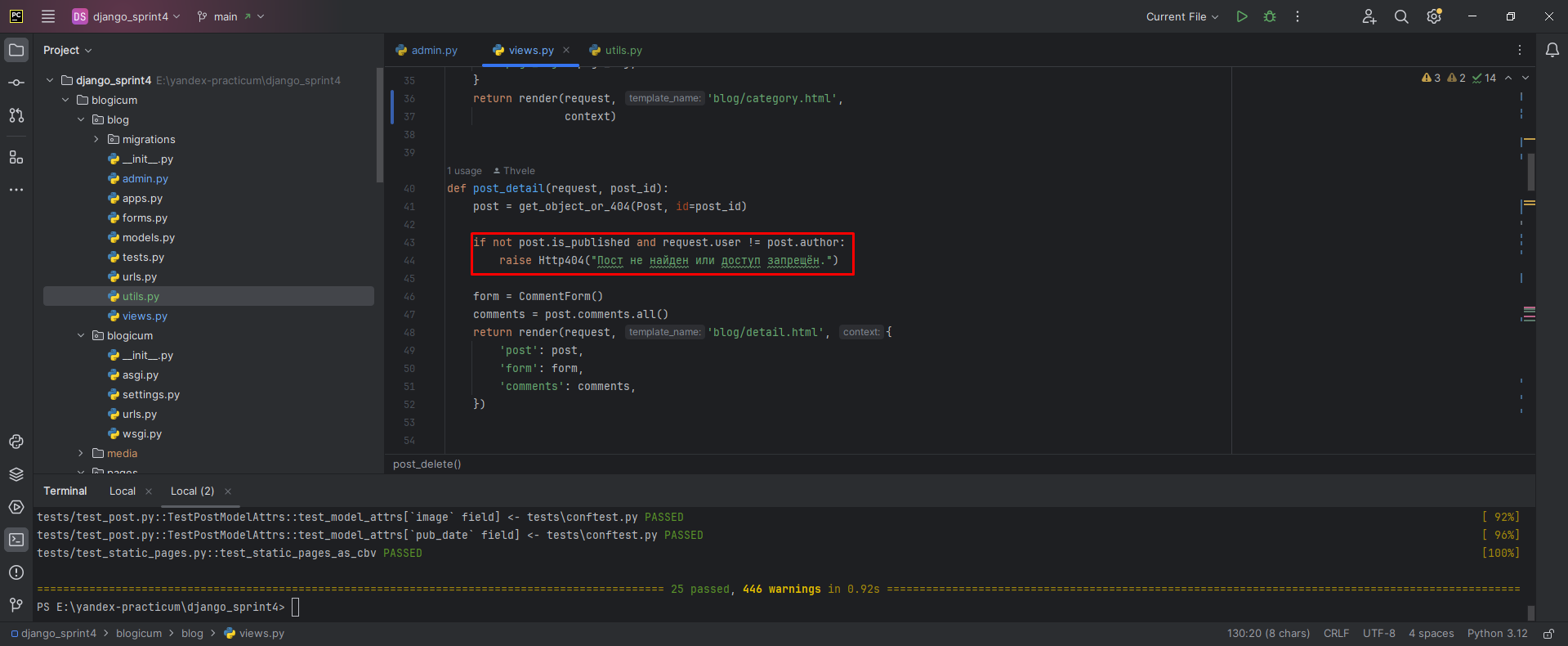




1. Для применения **redirect()** не нужно вычислять URL через **reverse()** . **redirect()** сам умеет работать с маршрутами.

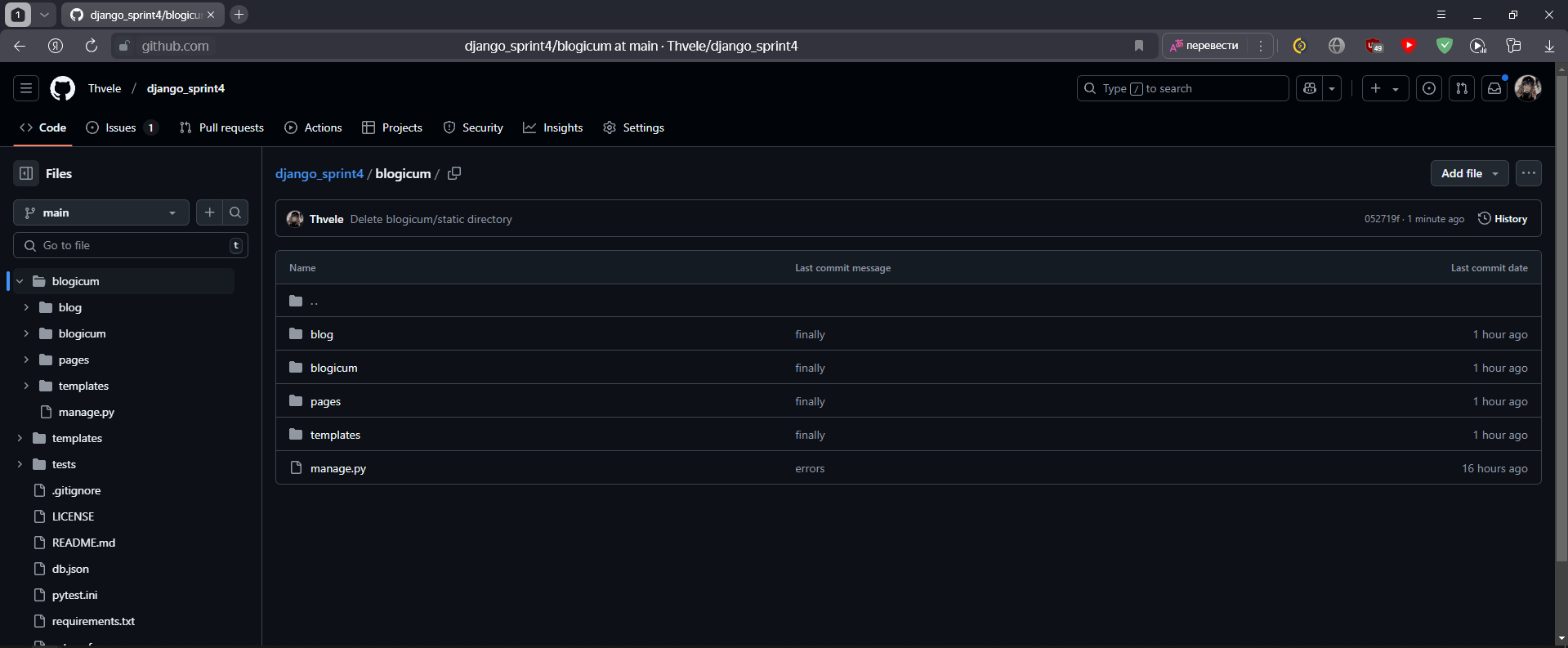


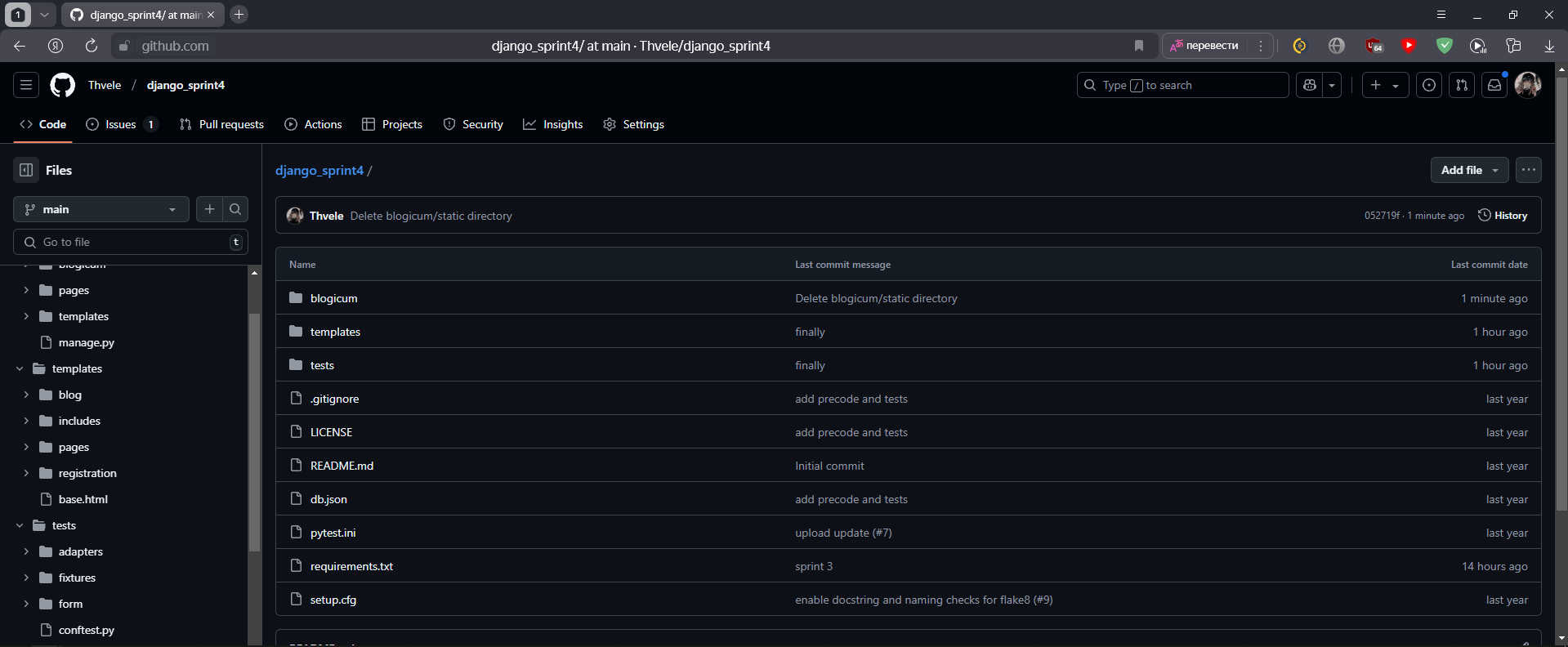
1. В контроллере **post\_detail ()** нужно анализировать автора поста, чтобы отказать в показе неопубликованного не авторам.



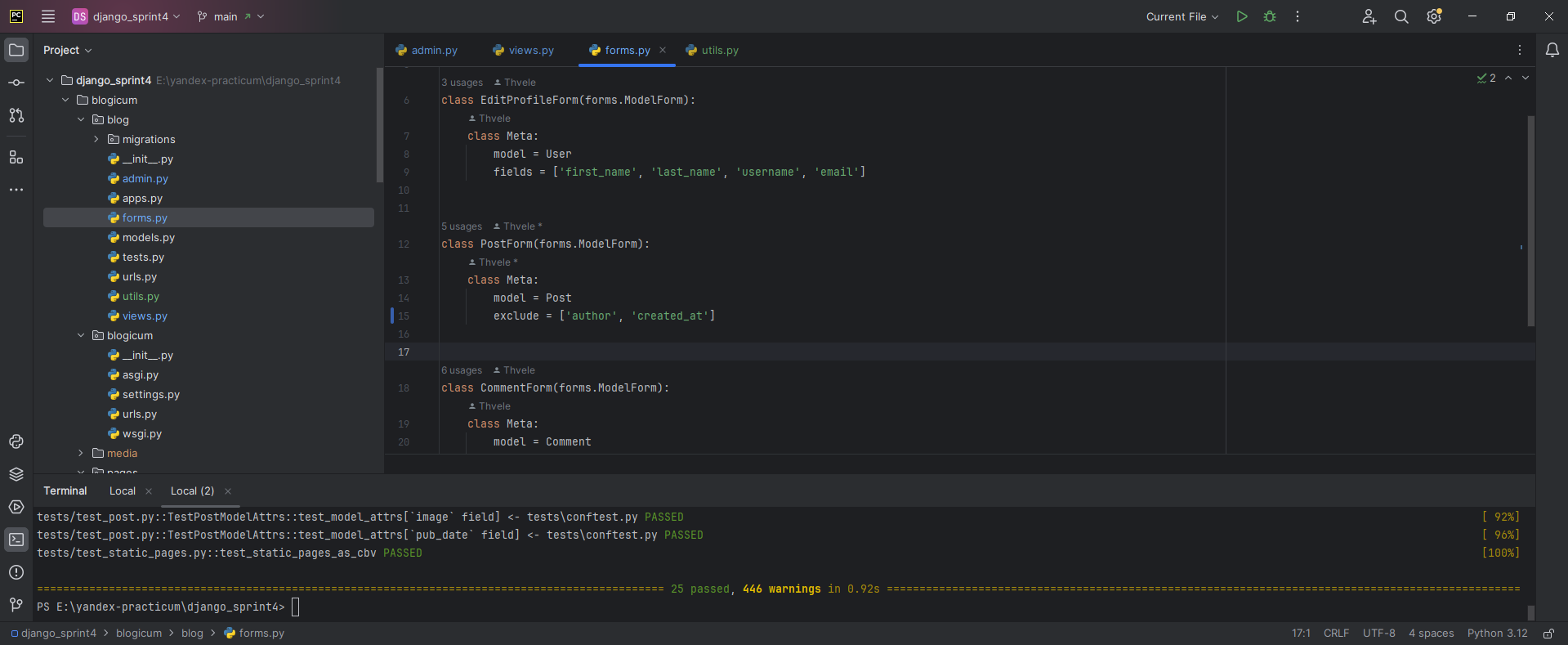
**Раздел 2**

1. В гит-репозитории не должны храниться папки **static** , **static-dev** , **html**



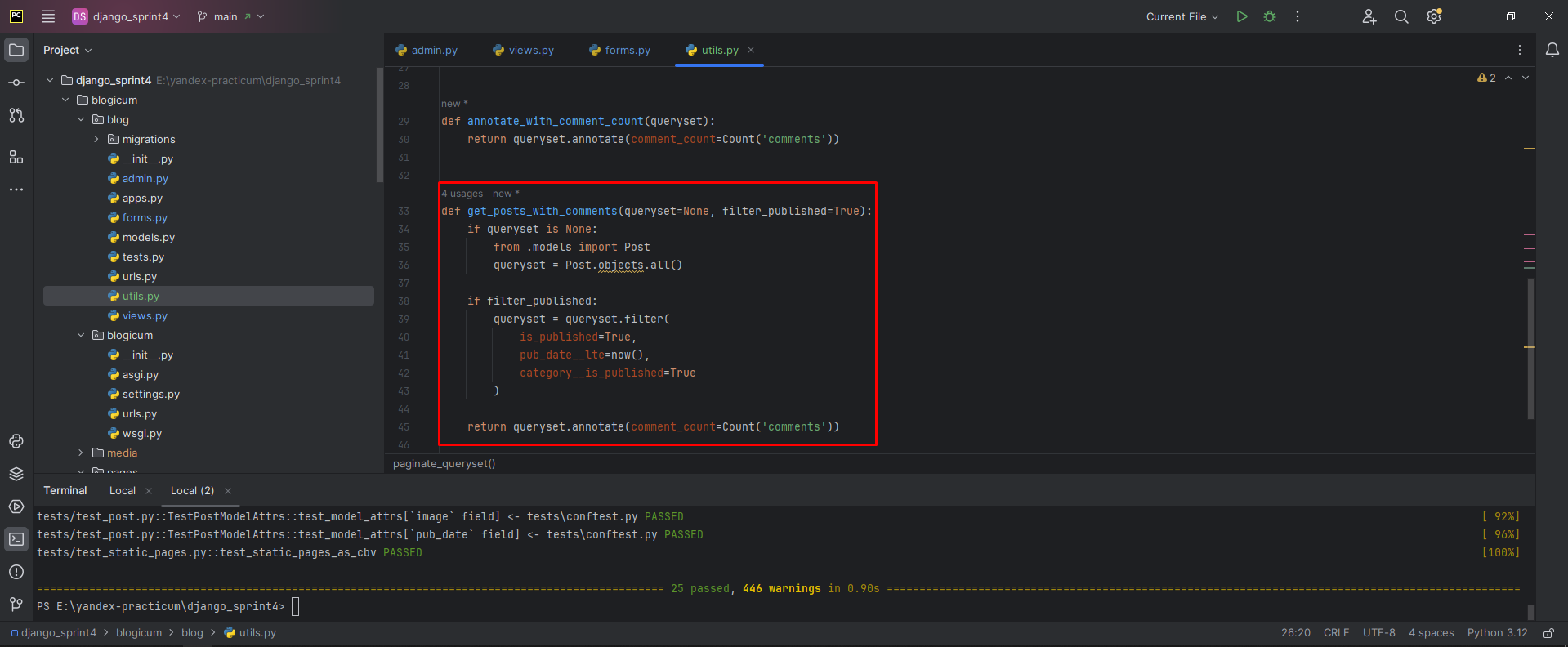


1. Класс формы для поста в **forms.py** лучше настраивать не через **fields** , а через **exclude** , так как нужны изменения всех полей кроме автора. Проверить, что поле **created\_at** не попадёт в форму: по модели оно нередактируемое.

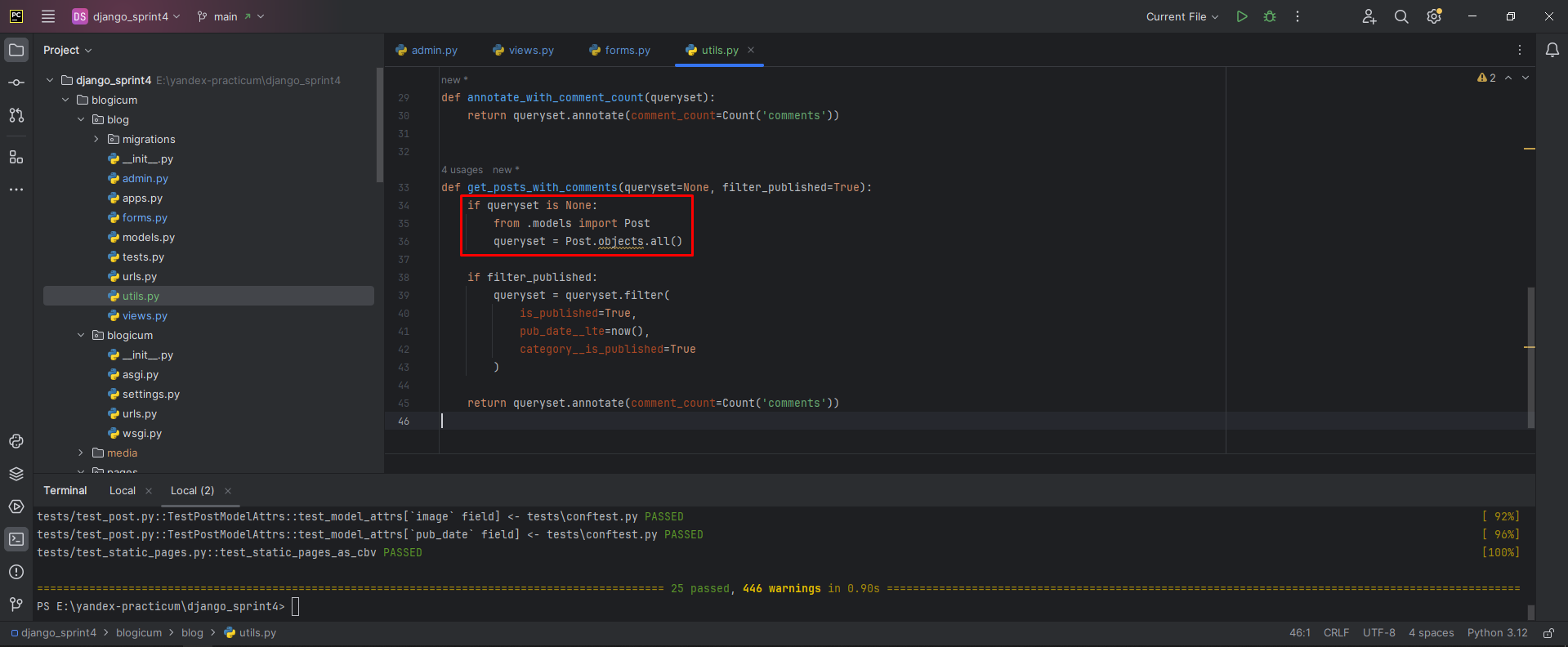


1. Функции «фильтрация по опубликованным» и «дополнение числа комментариев» может быть объединена в одну. Тогда для этой функции потребуется параметр со значениями «Да» или «Нет», чтобы можно было пропускать лишнюю фильтрацию для постов на странице автора для самого автора.



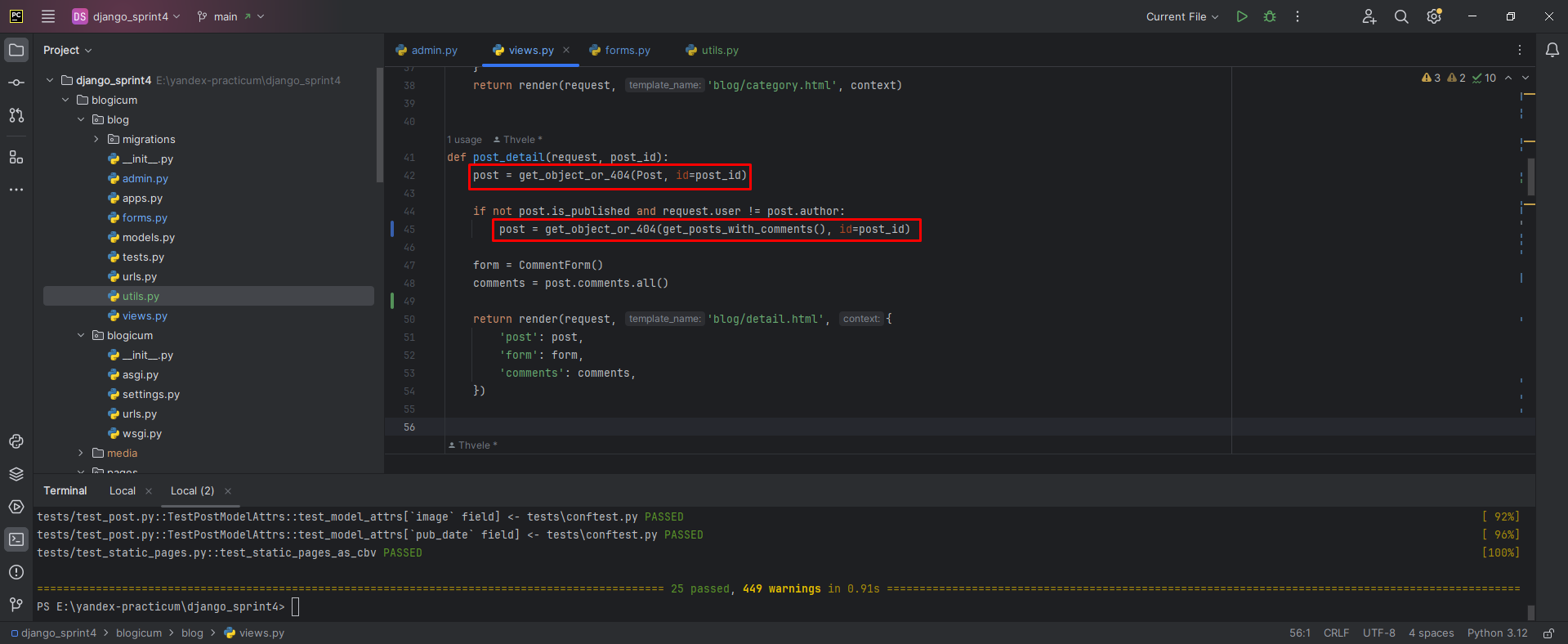


1. Функция, фильтрующая по опубликованности, может принимать параметром набор постов для фильтрации. Этому параметру можно дать значение по умолчанию «все посты таблицы», что будет ещё лучше.

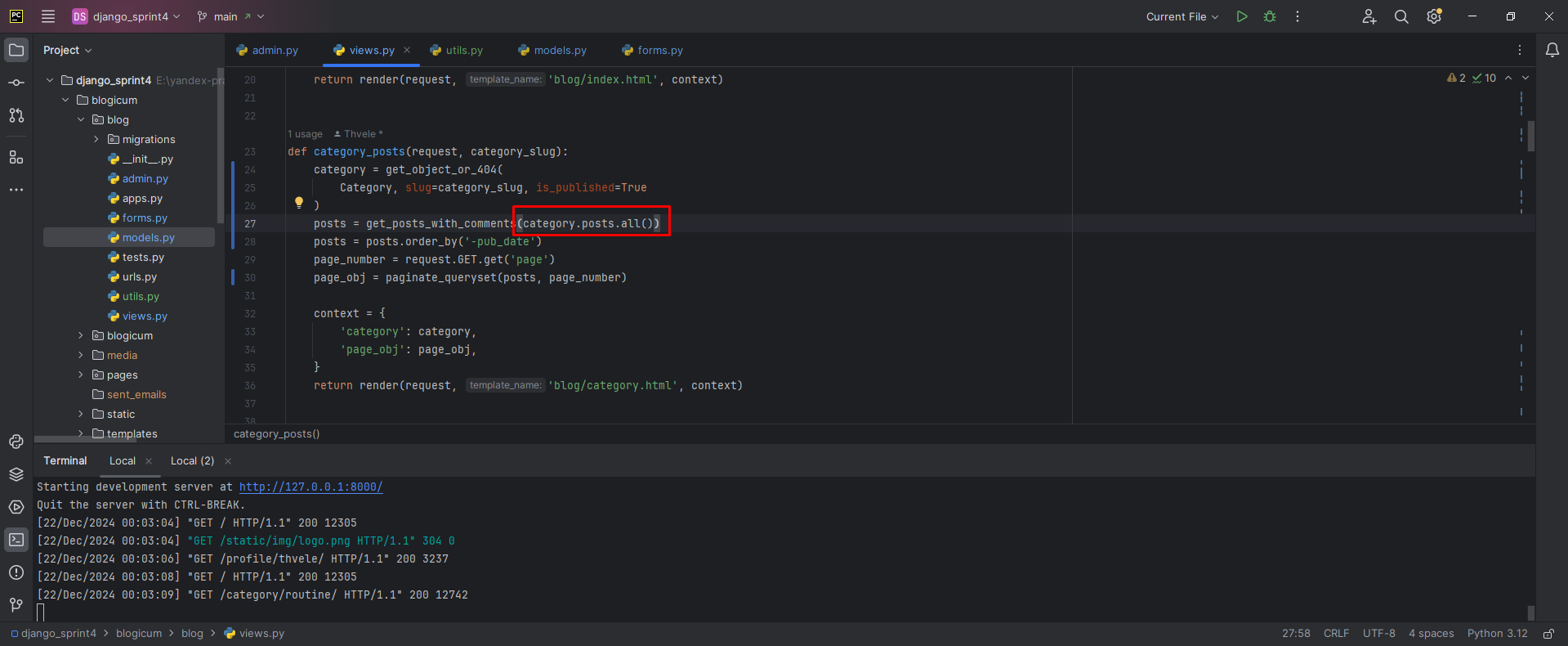


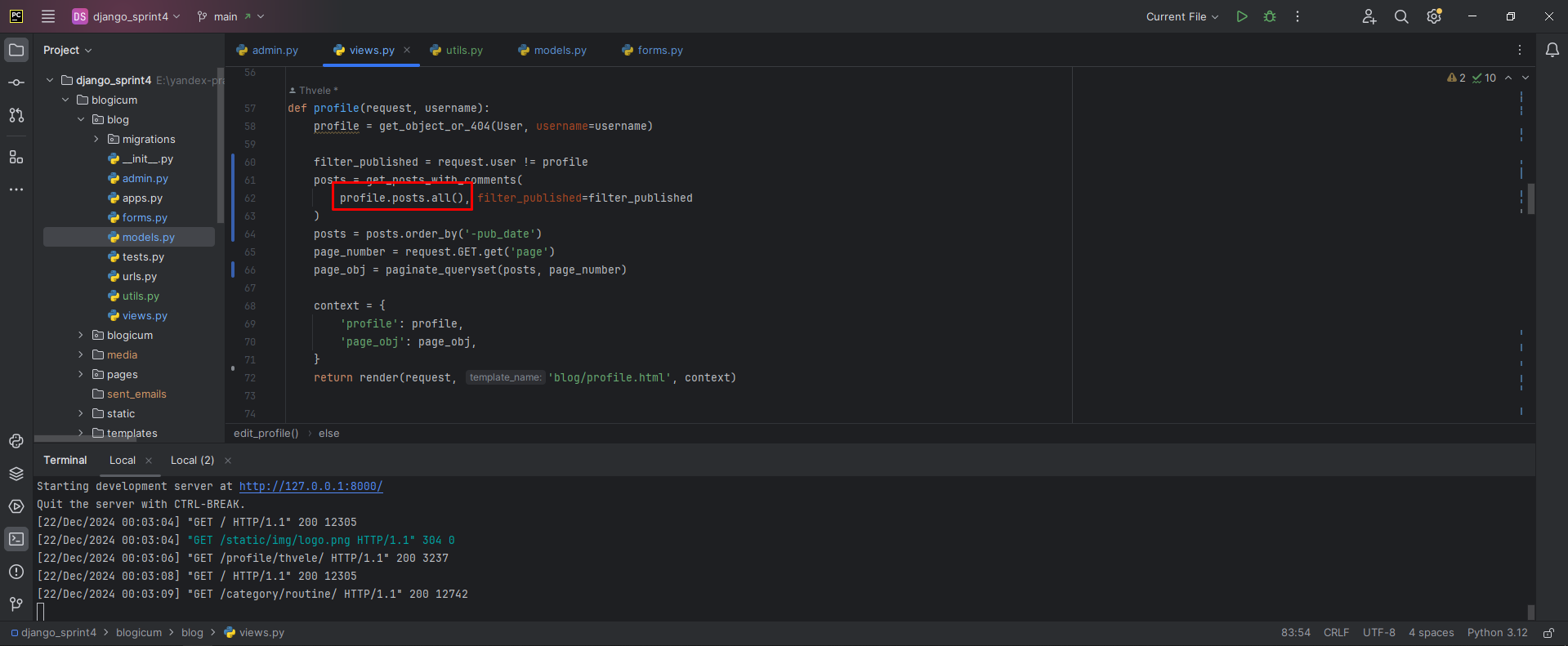
1. Чтобы в контроллере **post\_detail()** отказать в показе неопубликованного поста не автору, лучше всего делать два вызова **get\_object\_or\_404()** :

* Первый для извлечения поста по ключу из полной таблицы.
* Второй, после проверки авторства — из набора опубликованных постов.



1. Извлечения постов для уже излвеченной категории лучше выполнять, применяя «поле связи». То есть вместо **посты = Post.objects.filter(category=категория)** писать **посты = категория.поле-связи** . То же самое с постами автора: **посты = автор.поле-связи** . И с комментариями к постам **комментарии = пост.поле-связи** . Важно: *внутри шаблонов второй способ практически единственный, так как Python-вставки не допускают прямых вызовов методов с параметрами. А вот обращения к свойствам выполняются легко.*

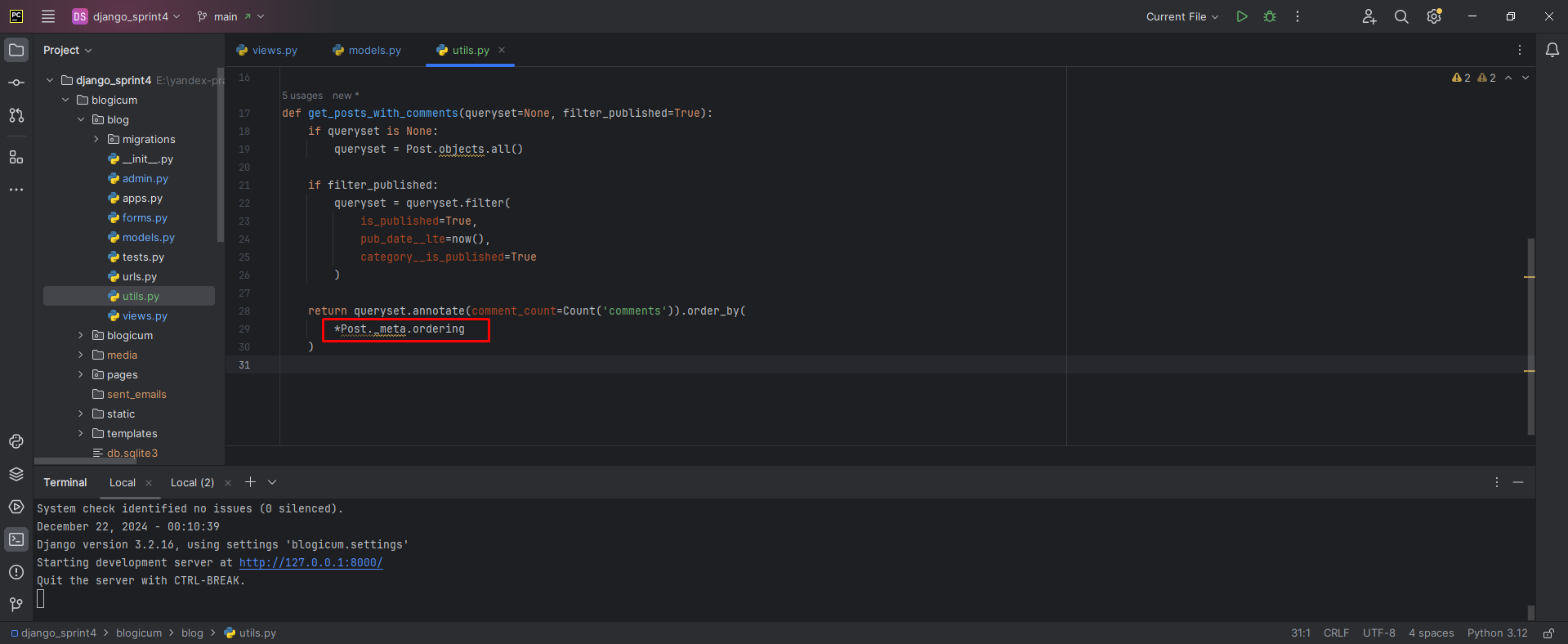




1. При указании сортировки после вызова **annotate()** лучше не угадывать значение, а брать точно такое же, какое уже есть в модели. Пригодится магическое поле **.\_meta** и распаковка. Код будет иметь следующий вид:

**предыдущие-действия.order\_by(\*Post.\_meta.ordering )**





1. В контроллерах, которые создают или редактируют пост, нужна разная обработка для GET и POST-запросов. Есть способ избежать прямых проверок вида

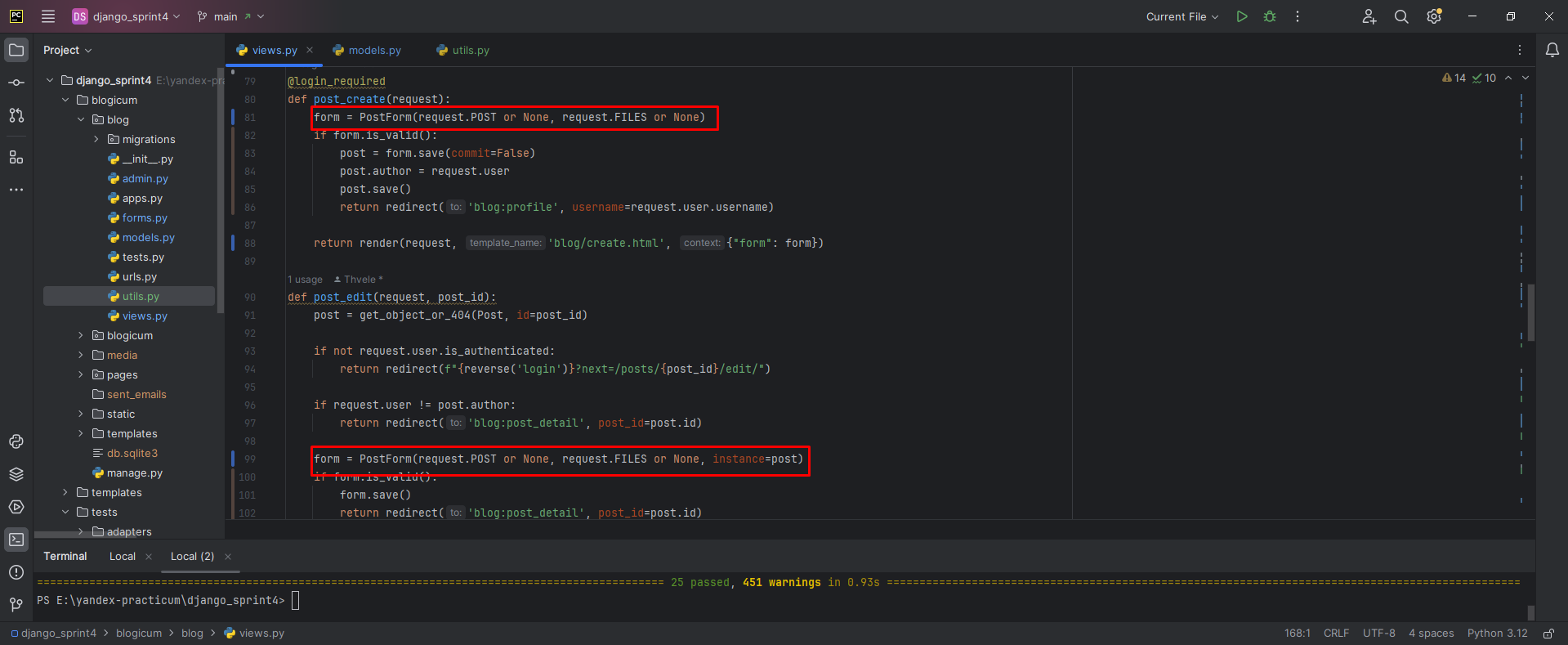
**if request.method == 'POST':** .

Для этого форма создается так: **form = PostForm(request.POST or None, ...)** .Такая форма годится для обоих запросов. Её валидация **form.is\_valid()** всегда неуспешна при GET-запросе. Значит, проверку метода выполнит проверка ответа от **.is\_valid()** . Удачное совмещение: и при провале валидации, и при GET-запросе как раз и нужно вызывать рендер:

**if not form.is\_valid():**

**…**

**return render(...)**



1. У модели «Пользователь» есть поля — например, пароль — которые нельзя показывать в админке. Поэтому админ-класс нельзя наследовать от **django.contrib.admin.ModelAdmin** . Для этого есть класс **django.contrib.auth.admin.UserAdmin** , который будет прятать секреты.

