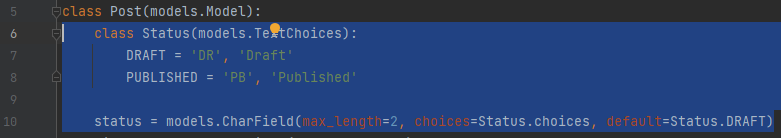
**Конспект первого проекта по книги**

**Глава 1(Добавление поля статуса)**

Для выбора значение мы определили перечисляемый класс путем подклассирования класса.



Для получения имеющихся вариантов: **Post.Status.choices**

Вывод: [('DF', 'Draft'), ('PB', 'Published')]

Для получения удобочитаемые имена: **Post.Status.labels**

Вывод: ['Draft', 'Published']

Для получения фактических значений: **Post.Status.values**

Вывод: ['DF', 'PB']

Для получения конкретного значения элемента: **Post.Status.DRAFT.value**

Вывод: 'DR'

Для получения конкретного значения элемента: **Post.Status.DRAFT.name**

Вывод: 'DRAFT'

**Глава 1(Создания сайта администрирования для моделей)**

Настройка админки:

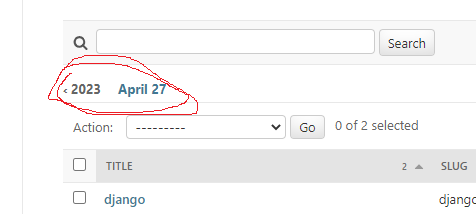
**list\_filter** – активирует боковую панель фильтрации

**prepopulated\_fields** - автоматическое заполнение слага

**raw\_id\_fields** - это список полей, которые вы хотели бы превратить в Input виджет для a ForeignKey или ManyToManyField

**date\_hierarchy** - быстрая фильтрация по датам

Фото пример:



**Глава 1** (**Когда вычисляются наборы запросов QuerySet )**

Наборы запросов QuerySet вычисляются только в следующих ниже случаях:

**•** при первом их прокручивании в цикле;

**•** при их нарезке, например Post.objects.all()[:3];

**•** при их консервации в поток байтов или кешировании;

**•** при вызове на них функций repr() или len();

**•** при вызове на них функции list() в явной форме;

**•** при их проверке в операциях bool(), or, and или if.

**Глава 1 (Добавление шаблонов URL-адресов представлений)**

**namespace** – может пригодиться в случае, если в проекте используется несколько экземпляров одного и того же приложения, которые манипулируют разными данными.

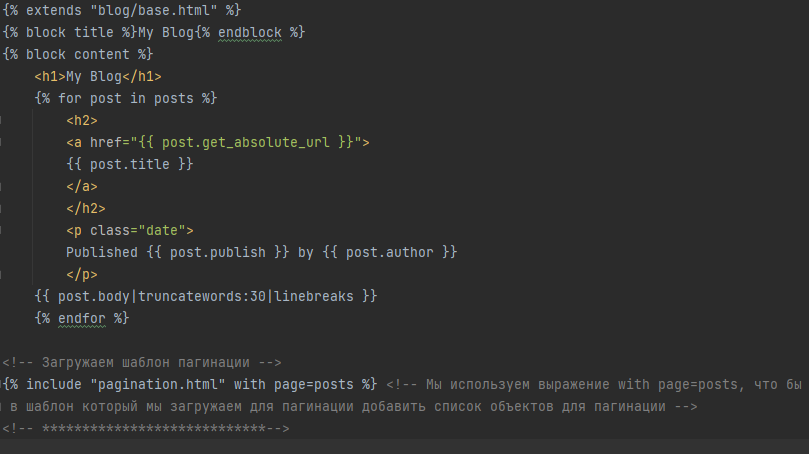
**Глава 2(Использование канонических URL-адресов для моделей)**

Канонический URL-адрес – это предпочтительный URL адрес ресурса,

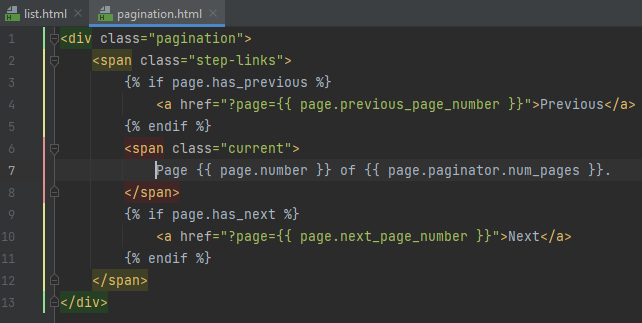
Простыми словами лучше использую метод **get\_absolute\_url** в шаблонах

**Глава 2(Создание шаблона постраничной разбивки)**

Мы используем выражение with page=posts, что бы передать в шаблон (который импортируем для пагинации ) список объектов которые нужно разбить на страницах



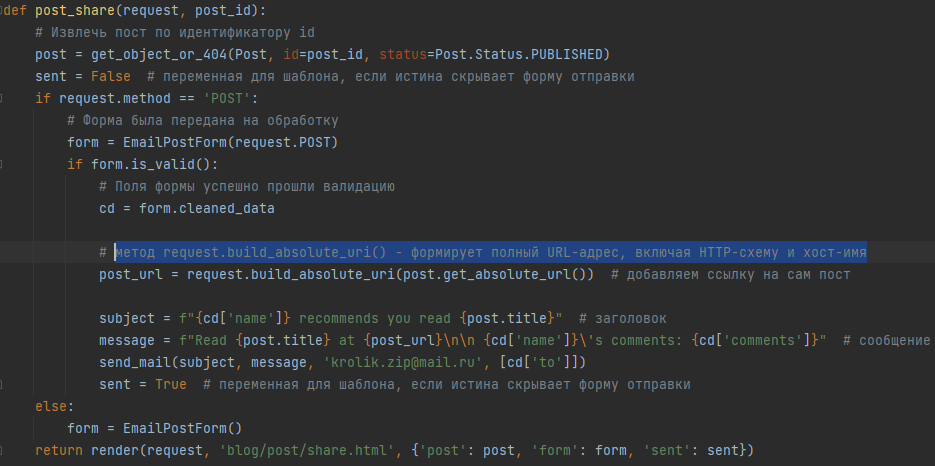
Пример шаблона который я импортирую для пагинации pagination.html



**Глава 2(Отправка электронных писем в представлениях)**

метод **request.build\_absolute\_uri()** - формирует полный URL-адрес, включая HTTP-схему и хост-имя.

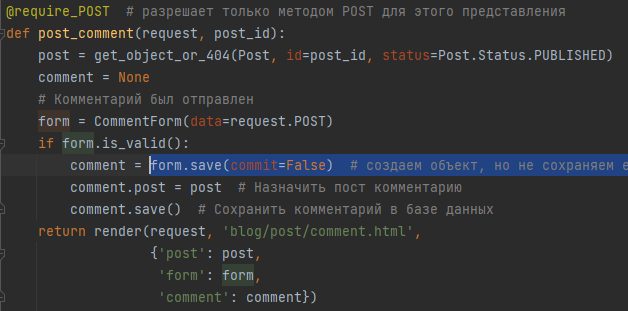
Пример кода:



**Глава 2(Оперирование формами ModelForm в представлениях)**

Декоратор **require\_POST**  - разрешает только методом POST **form.save(commit=False)** - создаем объект в оперативной памяти, но не сохраняем его в базе данных, чтобы модифицироваться его данные (добавлять/изменять данные)

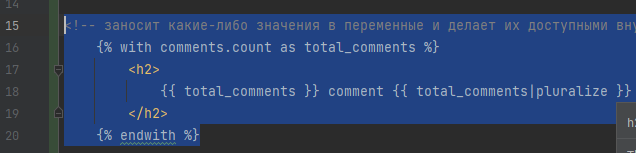
Фото пример:



**Глава 2(Добавление комментариев в шаблон детальной информации о посте)**

**with** . . . **endwith**— заносит какие-либо значения в переменные и делает их доступными внутри своего содержимого. Может использоваться для временного сохранения в переменных результатов каких-либо вычислений (например, полученных при обращении к методу класса) — чтобы потом не выполнять эти вычисления повторно.

Пример:



**Глава 2(Создание шаблонов комментарной формы)**

<form action="{% url 'blog:post\_comment' post.id %}" method="post">

В теге form в **action** – мы указываем url на который, будет отправлена форма.

**Глава 2(Добавление функциональности тегирования)**

**django-taggit** – это приспособленное для реиспользования приложение,

которое в первую очередь предлагает модель **Tag** и менеджер для удобного

добавления тегов в любую модель. Исходный код приложения доступен для

просмотра на странице https://github.com/jazzband/django-taggit.