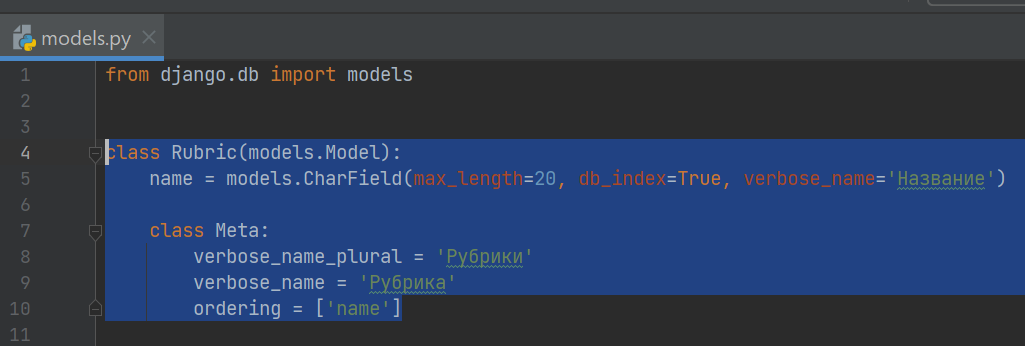
**Тема занятия № 3: Модуль 2. Связи. Ввод данных**

1. Связи между моделями

На всех приличных онлайновых досках объявлений все объявления разносятся по тематическим рубрикам: недвижимость, транспорт, бытовая техника и др. Давайте сделаем так и мы.

Сначала объявим класс модели Rubric, которая будет представлять рубрики объявлений, дописав в модуль models.py пакета приложения bboard код:



Модель содержит всего одно поле name, которое будет хранить название рубрики.

Для него мы сразу велели создать индекс, т. К. Будем выводить перечень рубрик отсортированным по их названиям.

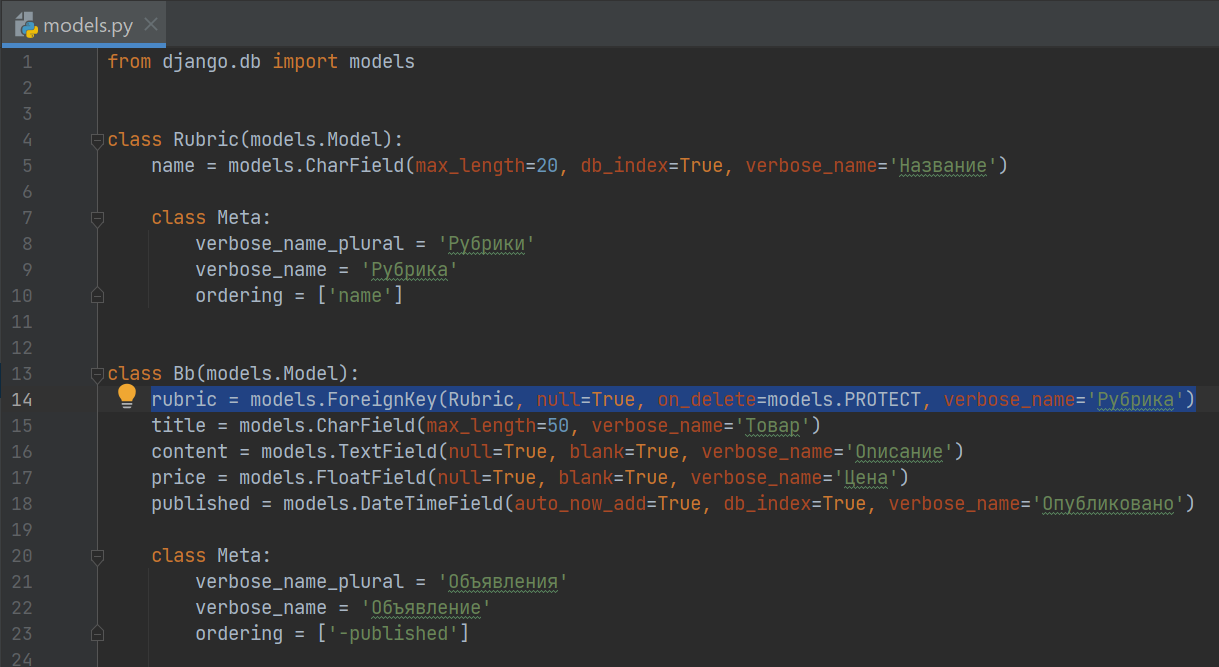
Теперь нужно добавить в модель Bb поле внешнего ключа, устанавливающее связь между текущей записью этой модели и записью модели Rubric, т. Е. Между объявлением и рубрикой, к которой оно относится. Таким образом будет создана связь "один-со-многими", при которой одна запись модели Rubric (рубрика) будет связана с произвольным количеством записей модели вь (объявлений). Модель Rubric станет первичной, а вь - вторичной.

Создадим во вторичной модели такое поле, назвав его Rubric:

Класс foreignkey представляет поле внешнего ключа, в котором фактически будет храниться ключ записи из первичной модели. Первым параметром конструктору этого класса передается строка с именем класса первичной модели, поскольку вторичная модель у нас объявлена раньше первичной.

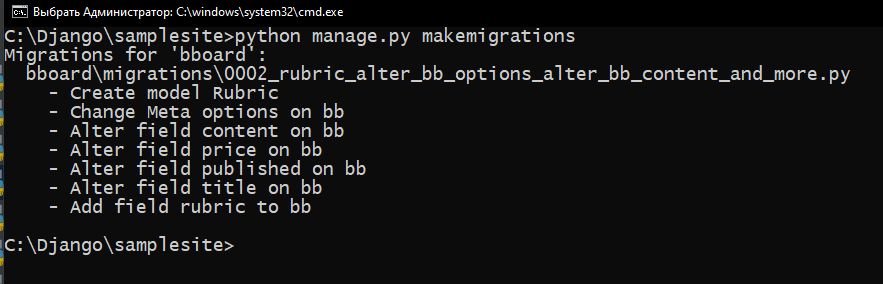
Все поля моделей по умолчанию обязательны к заполнению. Следовательно, добавить новое, обязательное к заполнению поле в модель, которая уже содержит записи, нельзя - сама СУБД откажется делать это и выведет сообщение об ошибке.

Нам придется явно пометить добавляемое поле Rubric как необязательное, присвоив параметру null значение True.



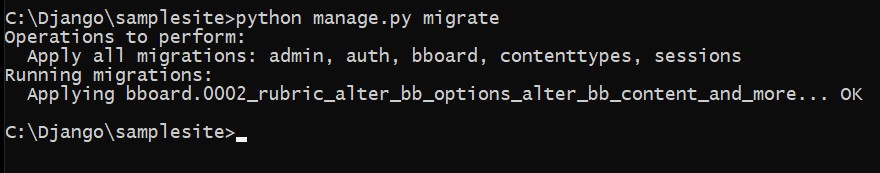
Именованный параметр on\_delete управляет каскадными удалениями записей вторичной модели после удаления записи первичной модели, с которой они были связаны. Значение PROTECT этого параметра запрещает каскадные удаления (чтобы какой-нибудь несообразительный администратор не стер разом уйму объявлений, удалив рубрику, к которой они относятся).

Сохраним исправленный модуль и сгенерируем миграции, которые внесут необходимые изменения в базу данных:

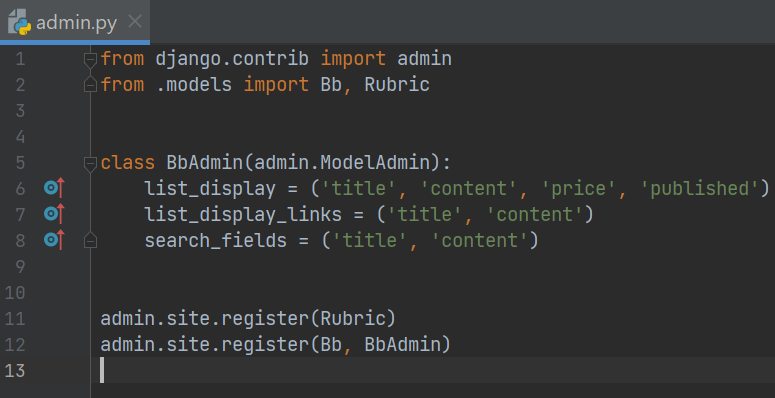


В результате в папке migrations будет создан модуль миграции с именем вида 0002\_auto\_ <отметка текущих даты и времени>.ру.

Выполним созданную миграцию:



Сразу же зарегистрируем новую модель на административном сайте, добавив в модуль admin.py пакета приложения два выражения:

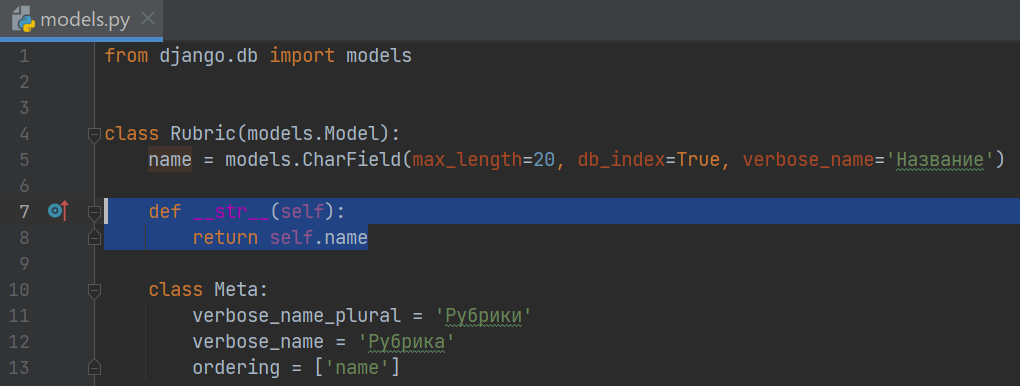


Запустим отладочный веб-сервер, войдем на административный сайт и добавим в модель Rubric рубрики "Недвижимость" и "Транспорт".

1. Строковое представление модели

Все хорошо, только в списке записей модели Rubric все рубрики представляются строками вида <имя класса модели> object (<значение ключа записи>) (нечто подобное поначалу выводилось у нас в списке записей модели вь.

Можно переопределить в классе модели метод **\_str\_(self)** , возвращающий строковое представление класса.



В качестве строкового представления мы выводим название рубрики.

Осталось сделать так, чтобы в списке записей модели Bb, помимо всего прочего, выводились рубрики объявлений. Для чего достаточно добавить в последовательность имен полей, присвоенную атрибуту list\_display класса bbadmin, поле rubric:



Обновим страницу списка объявления - и сразу увидим в нем новый столбец Рубрика. Отметим, что и здесь в качестве значения поля выводится строковое представление связанной записи модели.

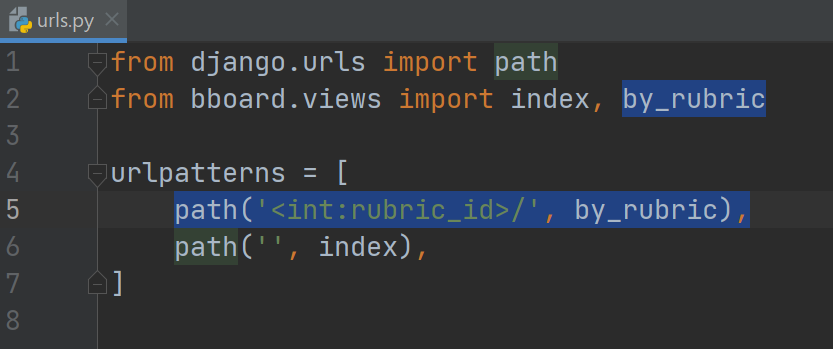
1. URL-параметры и параметризованные запросы

Логичным шагом выглядит разбиение объявлений по рубрикам не только при хранении, но и при выводе на экран. Давайте создадим на пока что единственной странице нашего сайта панель навигации, в которой выведем список рубрик, и при щелчке на какой-либо рубрике будем выводить лишь относящиеся к ней объявления. Остановим отладочный сервер и подумаем.

Чтобы контроллер, выводящий объявления, смог выбрать из модели лишь относящиеся к указанной рубрике, он должен получить ключ рубрики. Его можно передать через GЕТ-параметр: **/bboard/?Rubric=<ключ рубрики>.**

Однако Django позволяет поместить параметр непосредственно в составе интернет адреса: **ььоаrd/<ключ рубрики>/.** То есть через URL-пapaмeтp.

Для этого нужно указать маршрутизатору, какую часть интернет-адреса считать URL-параметром, каков тип значения этого параметра и какое имя должно быть у параметра контроллера, которому будет присвоено значение URL-параметра, извлеченного из адреса. Откроем модуль urls.py пакета приложения bbоаrd и внесем в него такие правки.

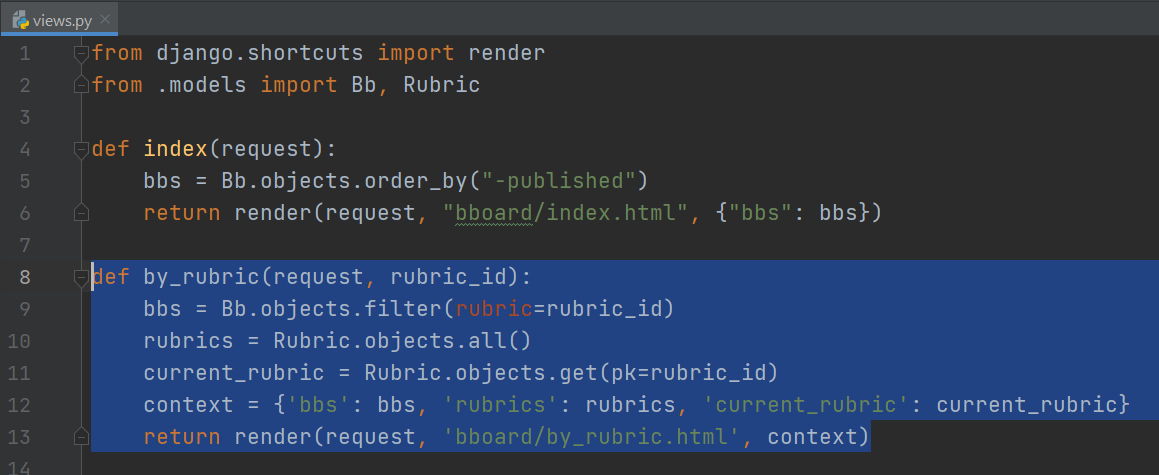
****

Мы добавили в начало набора маршрутов еще один, с шаблонным путем <int:rubric\_id>/. В нем угловые скобки помечают описание URL-параметра, языковая конструкция int задает целочисленный тип этого параметра, а rubric \_ id - имя параметра контроллера, которому будет присвоено значение этого urlпараметра. Созданному маршруту мы сопоставили конrроллер-функцию Bу \_ rubric (), который вскоре напишем.

Получив запрос по интернет-адресу http://localhost:8000/Ьboard/2/, маршрутизатор выделит путь bboard/2/, удалит из него префикс bboard и выяснит, что полученный путь совпадает с первым маршрутом из приведенного ранее списка. После чего запустит контроллер by\_rubric, передав ему в качестве параметра выделенный из интернет-адреса ключ рубрики 2.

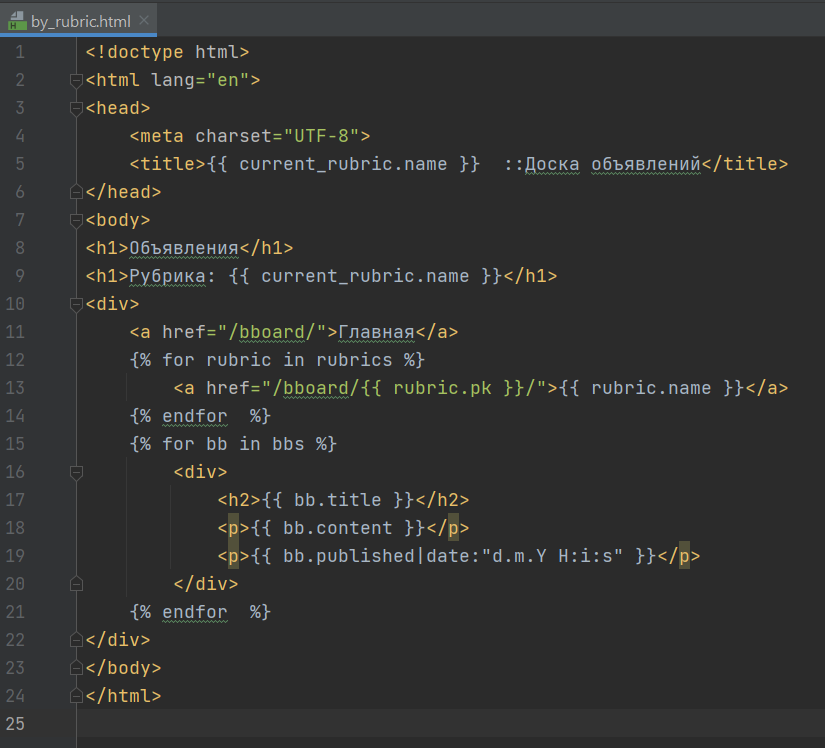
Маршруты, содержащие URL-параметры, носят название параметризованных.

Откроем модуль views. Py и добавим в него код контроллера-функции Bу\_ruьric



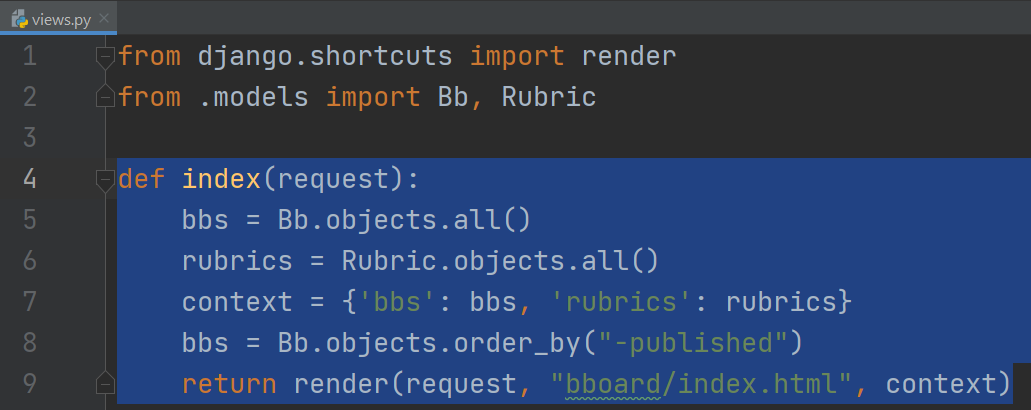
В объявление функции мы добавили параметр rubric\_id - именно ему будет присвоено значение URL-параметра, выбранное из интернет-адреса. В состав контекста шаблона поместили список объявлений, отфильтрованных по полю внешнего ключа rubric, список всех рубрик и текущую рубрику (она нужна нам, чтобы вы­ вести на странице ее название). Остальное нам уже знакомо.

Создадим в папке templates\bboard пакета приложения шаблон by\_rubric.html с кодом:

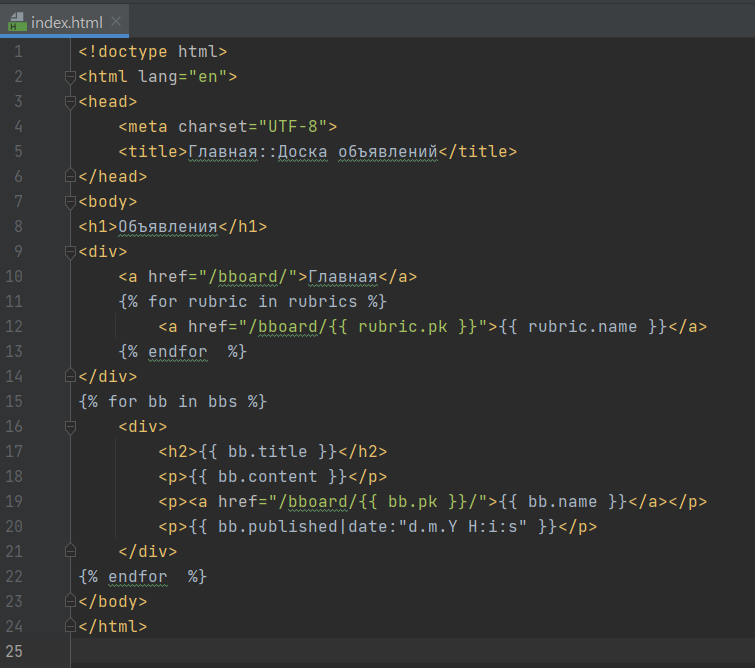


Обратим внимание, как формируются интернет-адреса в гиперссылках, находящихся в панели навигации и представляющих отдельные рубрики.

Исправим контроллер index ( ) и шаблон bboard\index.html таким образом, чтобы на главной странице нашего сайта выводилась та же самая панель навигации. Помимо этого, сделаем так, чтобы в составе каждого объявления выводилось название рубрики, к которой оно относится, выполненное в виде гиперссылки. Код исправленного контроллера index ( ):



Полный код исправленного шаблона bboard\index. Html:

****

Сохраним исправленные файлы, запустим отладочный веб-сервер и перейдем на главную страницу. Мы сразу увидим панель навигации, располагающуюся непосредственно под заголовком, и гиперссылки на рубрики, к которым относятся объявления.

1. Обратное разрешение интернет-адресов

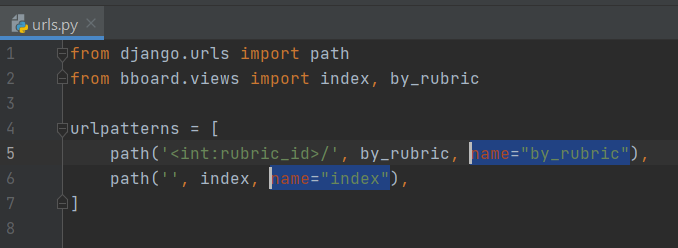
Посмотрим на НТМL-код гиперссылок



Интернет-адрес целевой страницы, подставляемый в атрибут href тега <а>, здесь формируется явно из префикса bboard и ключа рубрики, и, как мы убедились, все это работает. Но предположим, что мы решили поменять префикс, например, на bulletinboard. Неужели придется править код гиперссылок во всех шаблонах?

Избежать неприятностей такого рода позволяет инструмент Django, называемый обратным разрешением интернет-адресов. Это особый тег шаблонизатора, формирующий интернет-адрес на основе имени маршрута, в котором он записан, и набора URL-параметров, если это параметризованный маршрут.

Сначала нужно дать маршрутам имена, создав тем самым именованные маршруты. Откроем модуль urls.py пакета приложения и исправим код, создающий набор маршрутов, следующим образом



Имя маршрута указывается в именованном параметре name функции path ( ) .

Далее следует вставить в код гиперссылок теги шаблонизатора url которые и выполняют обратное разрешение интернет-адресов. Оrкроем шаблон bboard\index.html, найдем в нем фрагмент кода:



И заменим его на:



Имя маршрута указывается первым параметром тега url, а значения URL-пара­ метров - последующими.

Найдем код, создающий гиперссылку на главную страницу:



Заменим его на:



Маршрут index не является параметризованным, поэтому URL-параметры здесь не указываются.

Исправим код остальных гиперссылок в наших шаблонах. Обновим страницу, открытую в веб-обозревателе, и попробуем выполнить несколько переходов по гиперссылкам. Все должно работать.

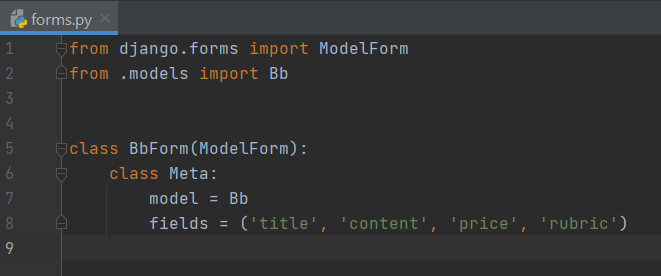
Осталось создать еще одну страницу - для добавления новых объявлений.

Для ввода данных Django предоставляет формы - объекты, "умеющие" выводить на страницу веб-формы с необходимыми элементами управления и проверять зане­ сенные в них данные на корректность. А форма, связанная с моделью, даже может самостоятельно сохранить данные в базе!

1. Формы, связанные с моделями

Создадим форму, связанную с моделью вь и служащую для ввода новых объявлений. Дадим ее классу имя Bbform.

Остановим отладочный веб-сервер. Создадим в пакете приложения Bbоаrd модуль forms.py, в который занесем код:

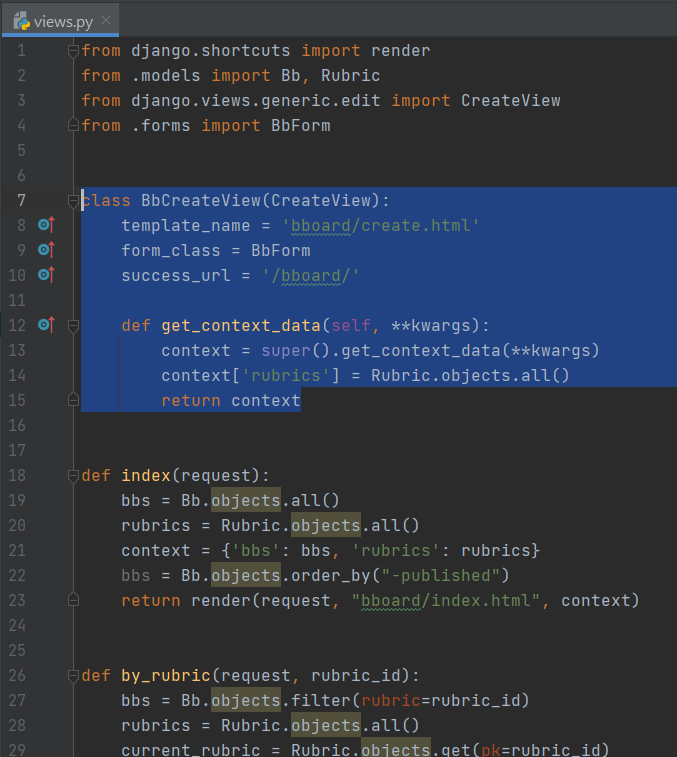


Класс формы, связанной с моделью, является производным от класса modelfoпn из модуля dj ango . Forms. Во вложенном классе Meta указываются параметры формы: класс модели, с которой она связана (атрибут класса model , и последовательность из имен полей модели, которые должны присутствовать в форме (атрибут класса fields).

1. Контроллеры-классы

Обрабатывать формы, связанные с моделью, можно в контроллерах-функциях. Но лучше использовать высокоуровневый контроллер-класс, который сам выполнит большую часть действий по выводу и обработке формы.

Наш первый контроллер-класс будет носить имя bbcreateview. Его мы объявим в модуле views.py пакета приложения. Код этого класса:



Контроллер-класс мы сделали производным от класса createview из модуля dj ango . Views . Generic . Edit. Базовый класс "знает", как создать форму, вывести на экран страницу с применением указанного шаблона, получить занесенные в форму данные, проверить их, сохранить в новой записи модели и перенаправить пользователя в случае успеха на заданный интернет-адрес.

Все необходимые сведения мы указали в атрибутах объявленного класса:

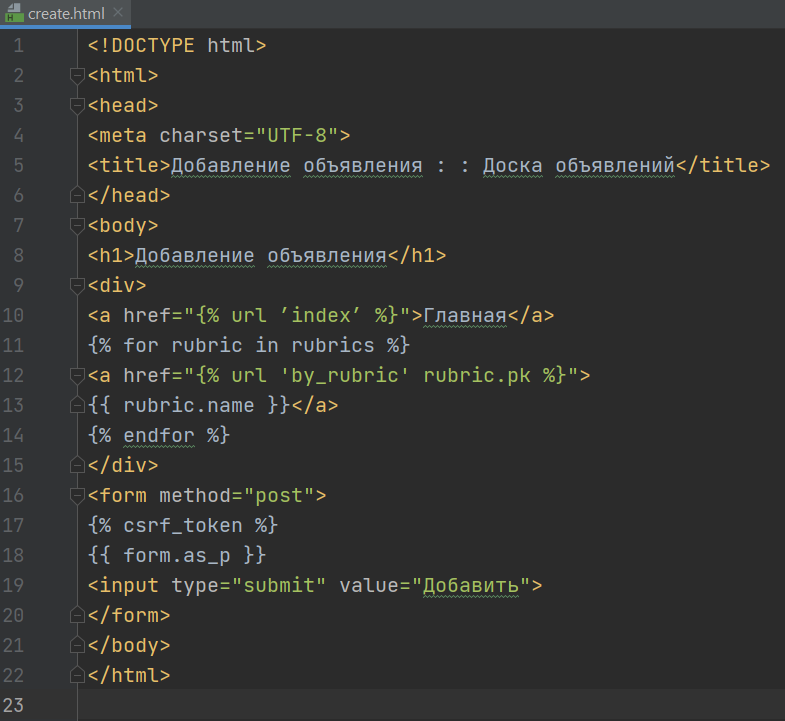
(] template\_name - путь к файлу шаблона, создающего страницу с формой;

(] form \_ class - ссылка на класс формы, связанной с моделью;

(] success\_url - интернет-адрес для перенаправления после успешного сохранения данных (в нашем случае это адрес главной страницы).

Поскольку на каждой странице сайта должен выводиться перечень рубрик, мы переопределили метод get \_ context \_ data ( ) , формирующий контекст шаблона. В теле метода получаем контекст шаблона от метода базового класса, добавляем в него список рубрик и, наконец, возвращаем его в качестве результата.

Займемся шаблоном bboard\create.html. Его код представлен:



Здесь нас интересует только код, создающий веб-форму. Обратим внимание на четыре важных момента:

Форма в контексте шаблона хранится в переменной form, создаваемой контроллером-классом;

Для вывода формы, элементы управления которой находятся на отдельных абзацах, применяется метод as\_Р () класса формы;

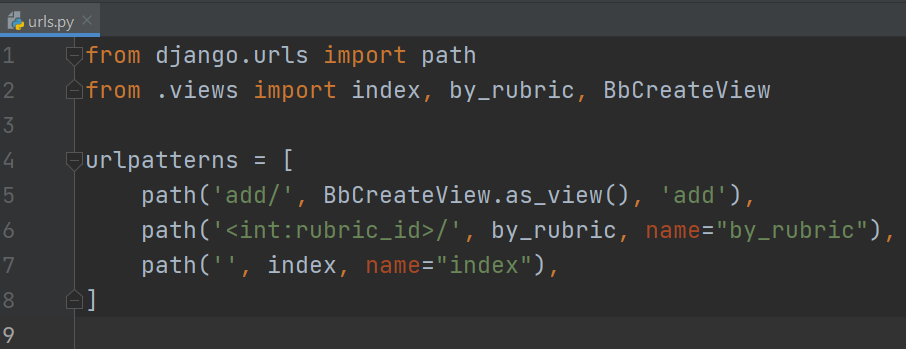
Метод as\_Р () генерирует только код, создающий элементы управления. Тег <form>, необходимый для создания самой формы, и тег <input>, формирующий кнопку отправки данных, придется писать самостоятельно.

В теге <form> мы указали метод отправки данных POST, но не записали интернет-адрес, по которому будут отправлены данные из формы. В этом случае данные будут отправлены по тому же интернет-адресу, с которого была загружена текущая страница, т. Е. В нашем случае тому же контроллеру-классу

Bbcreateview, который благополучно обработает и сохранит их;

В теге <form> мы поместили тег шаблонизатора csrf\_token. Он создает в форме скрытое поле, хранящее цифровой жетон, получив который контроллер "поймет", что данные были отправлены с текущего сайта и им можно доверять. Это часть 'ilодсистемы безопасности Django.

Добавим в модуль urls.py пакета приложения маршрут, указывающий на контроллер createview:



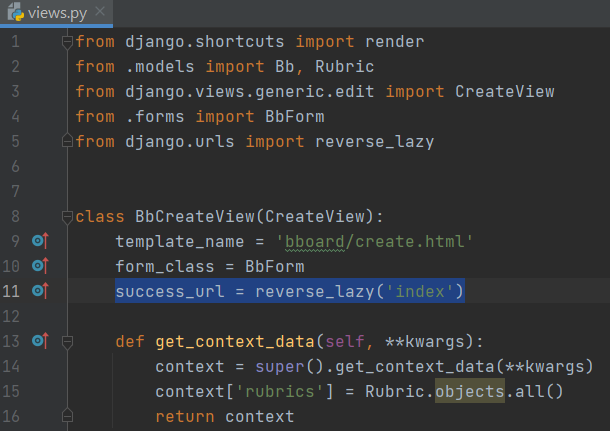
Не забудем создать в панели навигации всех страниц гиперссылку на страницу добавления объявления:



В объявлении класса bbcreateview мы опять указали интернет-адрес перенаправления (в атрибуте класса success uri) непосредственно, что считается дурным тоном

Программирования. Сгенерируем его путем обратного разрешения.

Откроем модель views,ру и внесем следующие правки в код класса bbcreateview:



1. Наследование шаблонов

Еще раз посмотрим на код. Что сразу бросается в глаза? Большой объем совершенно одинакового НТМL-кода: секция заголовка, панель навигации, всевозможные служебные теги. Эго увеличивает совокупный объем шаблонов и усложняет сопровождение - если мы решим изменить что-либо в этом коде, то будем вынуждены вносить правки в каждый из имеющихся у нас шаблонов.

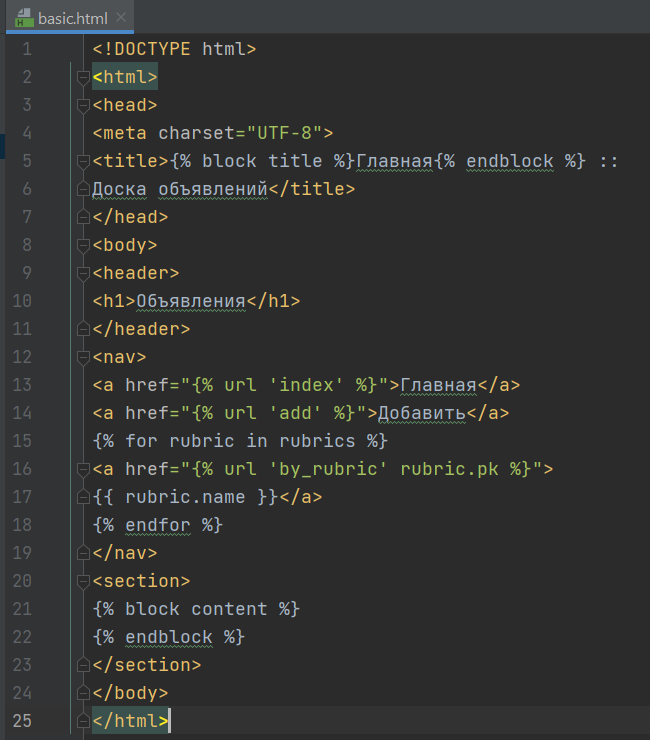
Решить эту проблему можно, применив наследование шаблонов, аналогичное наследованию классов. Шаблон, являющийся базовым, содержит повторяющийся код, в нужных местах которого объявлены блоки, помечающие места, куда будет вставлено содержимое из производных шаблонов. Каждый блок имеет уникальное в пределах шаблона имя.

Создадим базовый шаблон со следующими блоками:

- title - будет помещаться в теге <title> и служить для создания уникального названия для каждой страницы;

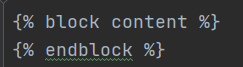
- content - будет использоваться для размещения уникального содержимого страниц.

Создадим в папке templates папку layout. Сохраним в ней наш базовый шаблон basic.html:



Начало объявляемого блока помечается тегом шаблонизатора Blосk, за которым должно следовать имя блока. Завершается блок тегом endblock.

Объявленный в базовом шаблоне блок может быть пустым:

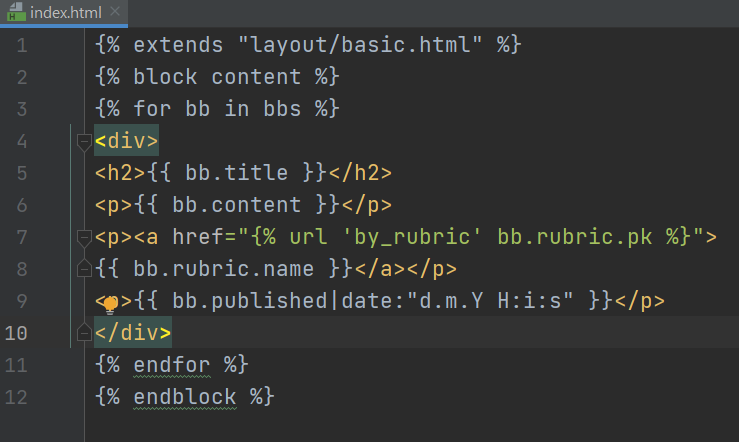


Или иметь какое-либо изначальное содержимое:



Оно будет выведено на страницу, если производный шаблон не задаст для блока свое содержимое.

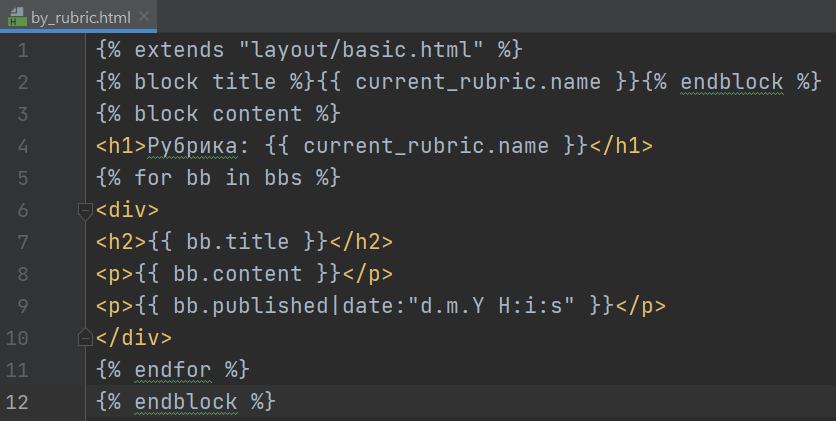
Сделаем шаблон bboard\index.html производным от шаблона layout\basic.html. Новый код

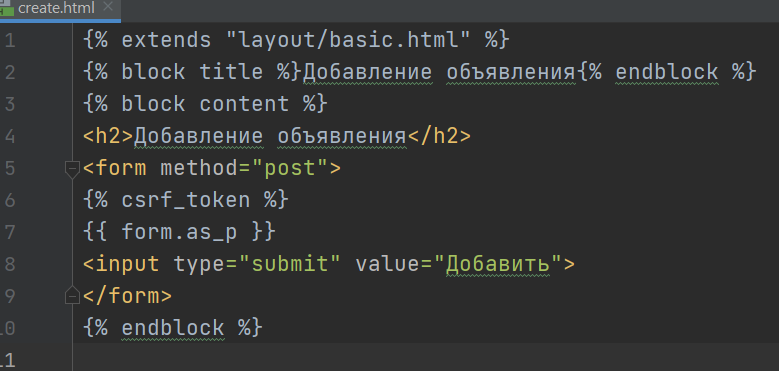


В самом начале кода любого производного шаблона ставится тег шаблонизатора extends, в котором указывается путь к базовому шаблону. Далее следуют объявления блоков, обозначаемые теми же тегами Blосk и endblock, в которых записывается их содержимое.

Если мы теперь сохраним исправленные файлы и обновим открытую в веб-обозревателе главную страницу, то увидим, что она выводится точно так же, как ранее.

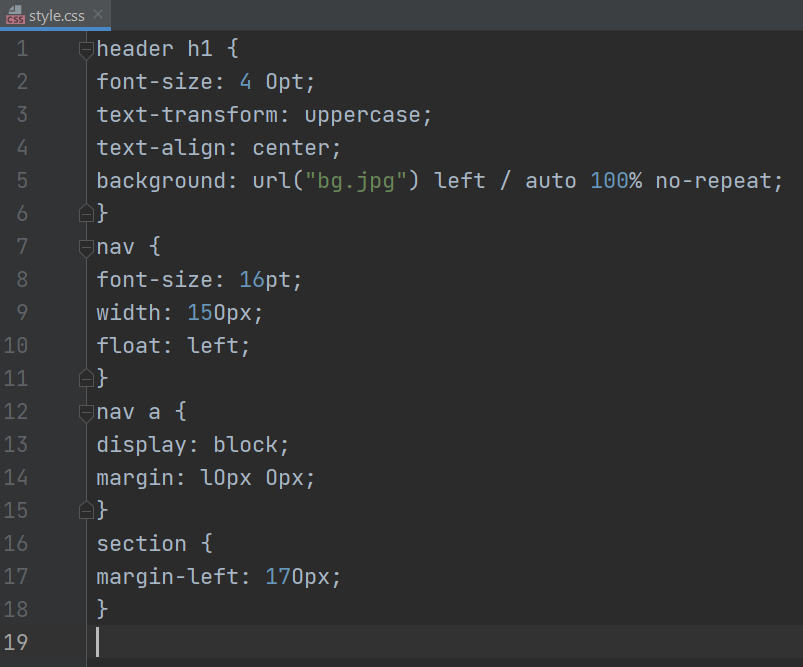
Аналогично исправим шаблоны bboard\Ьy\_rubric.html





1. Статические файлы

Наш первый сайт практически готов. Осталось навести небольшой лоск, применив таблицу стилей style.css:

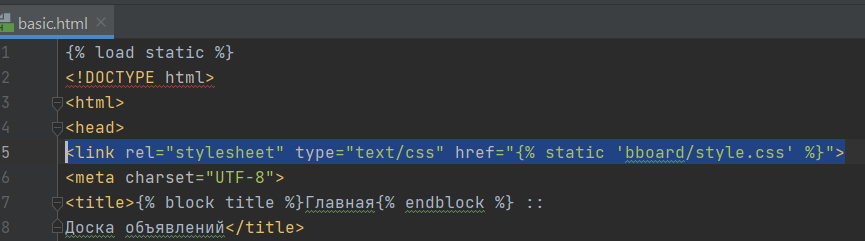


Но где нам сохранить эту таблицу стилей и файл с фоновым изображением bg.jpg?

И вообще, как привязать таблицу стилей к базовому шаблону?

Файлы, содержимое которых не обрабатывается программно, а пересылается клиенту как есть, в терминологии Django носят название статических. К таким файлам относятся, например, таблицы стилей и графические изображения, помещаемые на страницы.

Остановим отладочный веб-сервер. Создадим в папке пакета приложения bbоаrd папку static, а в ней - вложенную папку bboard. В последней сохраним файлы style.css и bg.jpg (можно использовать любое подходящее изображение, загруженное из Интернета).

Откроем базовый шаблон layout\Ьasic.html и вставим в него следующий код:

Тег шаблонизатора load загружает указанный в нем модуль расширения, содержащий дополнительные теги и фильтры. В нашем случае выполняется загрузка модуля static, который содержит теги для вставки ссылок на статические файлы.

Один из этих тегов - static - генерирует полный интернет-адрес статического файла на основе заданного в нем пути относительно папки static. Мы используем его, чтобы вставить в НТМL-тег <link> интернет-адрес таблицы стилей bboard\style.css.

Запустим отладочный сервер и откроем главную страницу сайта.

Занесем на сайт еще несколько объявлений (специально придумывать текст для них совершенно необязательно - для целей отладки можно указать что угодно). Попробуем при наборе очередного объявления не вводить какое-либо обязательное для занесения значение, скажем, название товара, и посмотрим, что случится. Попробуем добавить на страницы, например, поддон.