**Тема занятия № 2: Основные понятия Django. Вывод данных**

1. Модели

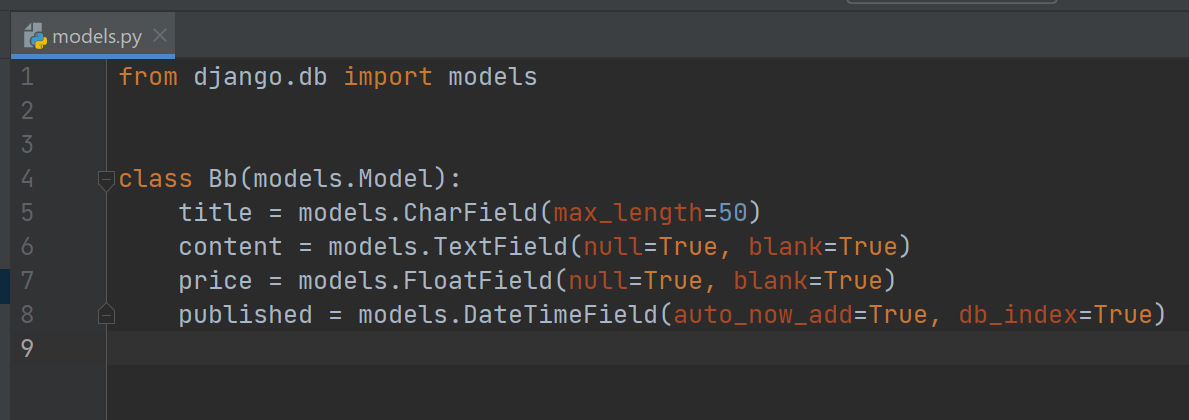
Настала пора сделать так, чтобы вместо намозолившего глаза текстового сообщения выводились реальные объявления, хранящиеся в таблице базы данных. Для этого нам понадобится прежде всего объявить модель.

Модели объявляются в модуле models.py пакета приложения. Изначально этот модуль пуст.

Объявим модель Bb, представляющую объявление, со следующими полями:

* Title — заголовок объявления с названием продаваемого товара (тип — строковый, длина — 50 символов). Поле, обязательное к заполнению;
* Content — сам текст объявления, описание товара (тип — memo);
* Price — цена (тип — вещественное число);
* Published — дата публикации (тип — временная отметка, значение по умолчанию — текущие дата и время, индексированное).

Завершим работу отладочного веб-сервера. Откроем модуль models.py пакета приложения bboard и запишем в него код:



Модель должна быть подклассом класса Model из модуля django.db.models. Отдельные поля модели объявляются в виде атрибутов класса, которым присваиваются экземпляры классов, представляющих поля разных типов и объявленных в том же модуле. Параметры полей указываются в конструкторах классов полей в виде значений именованных параметров.

Рассмотрим использованные нами классы полей и их параметры:

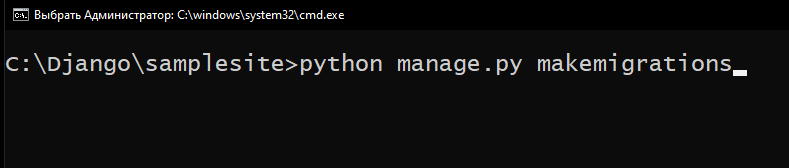
* Charfield — обычное строковое поле фиксированной длины. Допустимая длина значения указывается параметром max\_length конструктора;
* Textfield — текстовое поле неограниченной длины, или memo-поле. Присвоив параметрам null и blank конструктора значения True, мы укажем, что это поле можно не заполнять (по умолчанию любое поле обязательно к заполнению);
* Floatfield — поле для хранения вещественных чисел. Оно также необязательно для заполнения (см. Параметры его конструктора);
* Datetimefield — поле для хранения временнóй отметки. Присвоив параметру auto\_now\_add конструктора значение True, мы предпишем Django при создании новой записи заносить в это поле текущие дату и время. А параметр db\_index при присваивании ему значения True укажет создать для этого поля индекс (при выводе объявлений мы будем сортировать их по убыванию даты публикации, и индекс здесь очень пригодится).

Практически всегда таблицы баз данных имеют поле для хранения ключей — уникальных значений, однозначно идентифицирующих записи (ключевое поле). Как правило, это поле целочисленного типа и помечено как автоинкрементное — тогда сама СУБД будет заносить в него уникальные номера. В моделях Django такое поле явно объявлять не надо — фреймворк создаст его самостоятельно.

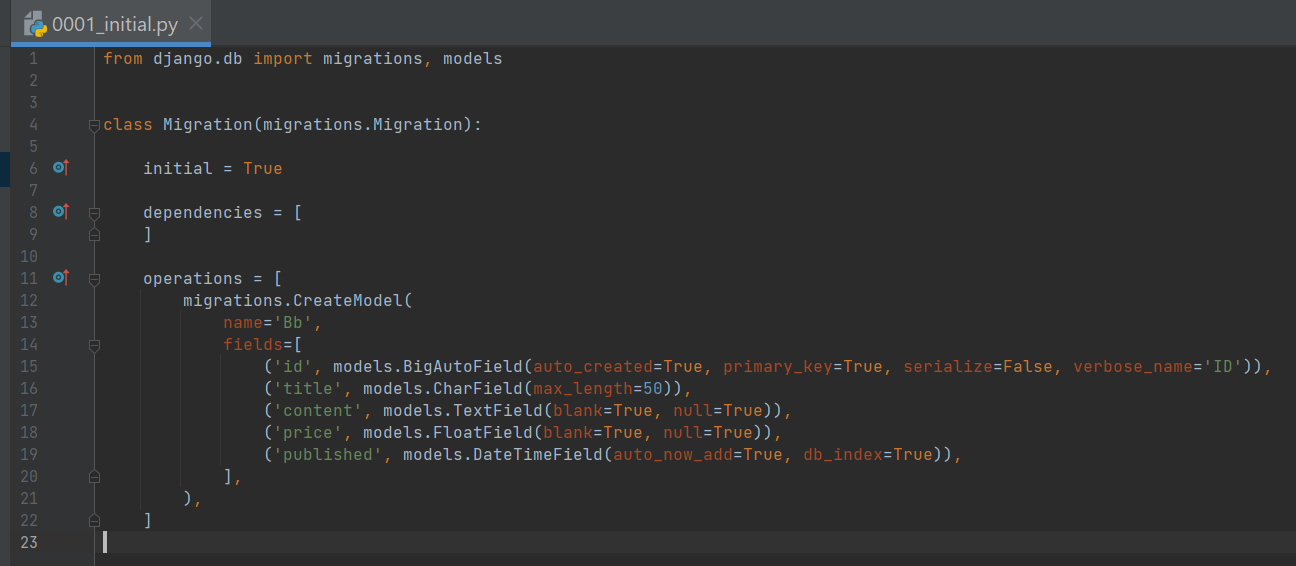
Сохраним исправленный файл. Сейчас мы сгенерируем на его основе миграцию, которая создаст в базе данных все необходимые структуры.

1. Миграции

Чтобы сгенерировать миграцию на основе модели Bb, переключимся в командную строку, проверим, остановлен ли отладочный веб-сервер и находимся ли мы в папке проекта, и дадим команду:



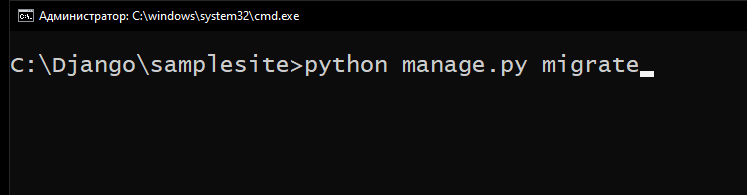
Модуль с кодом нашей первой миграции будет иметь имя **0001\_initial.py**. Откроем его в текстовом редакторе и посмотрим на хранящийся в нем код:



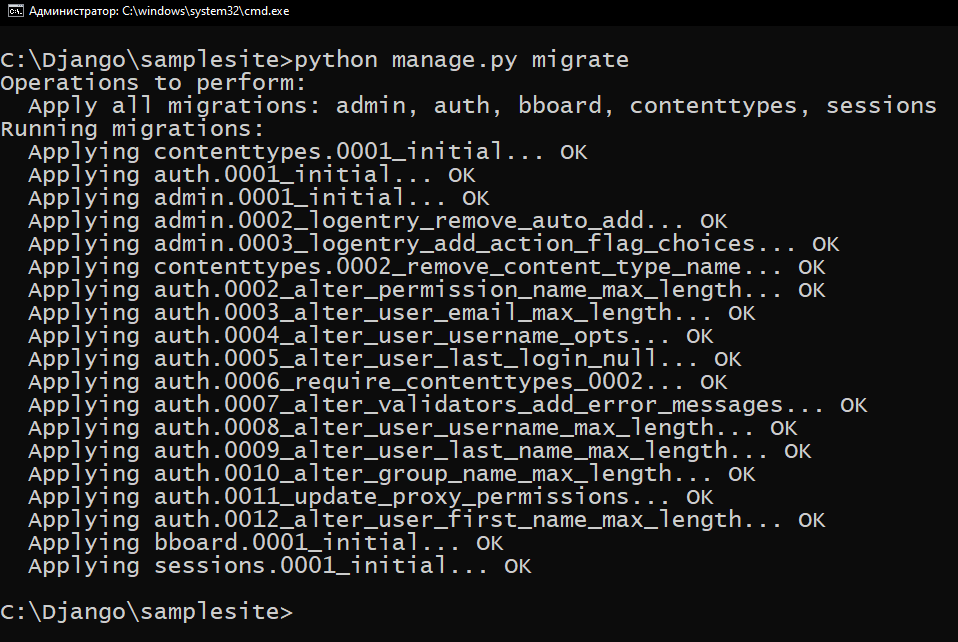
Код миграции вполне понятен и напоминает код написанной ранее модели. Создаваемая в базе данных таблица будет содержать поля id, title, content, price и published. Ключевое поле id для хранения уникальных номеров записей Django создаст самостоятельно.

Миграция при выполнении порождает команды на языке SQL, создающие в базе необходимые структуры.

Налюбовавшись на нашу первую миграцию, выполним ее. Для этого наберем в командной строке команду:



Судя по выводящимся в командной строке сообщениям, таких миграций много — десятка два.



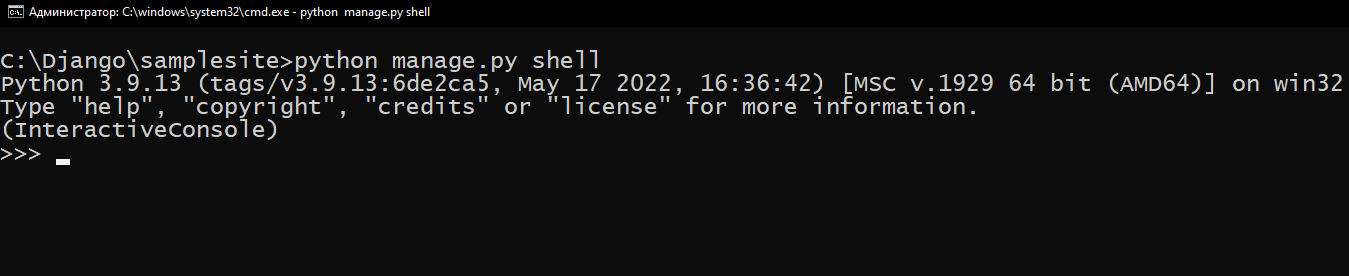
Дождемся, когда их выполнение завершится, и продолжим.

1. КОнсоль Django

Итак, у нас есть готовая модель для хранения объявлений. Но пока что нет ни одного объявления. Давайте создадим парочку для целей отладки.

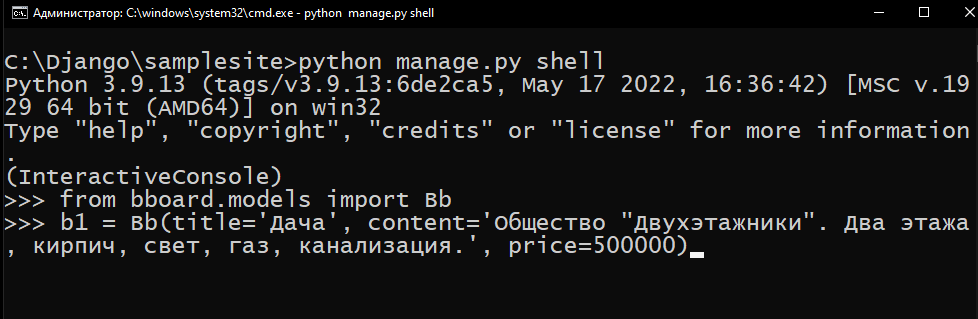
Фреймворк включает в свой состав собственную редакцию консоли Python Shell, называемую консолью Django. От аналогичной командной среды Python она отличается тем, что в ней в состав путей поиска модулей добавляется путь к папке проекта, в которой запущена эта консоль.

В командной строке наберем команду для запуска консоли Django:



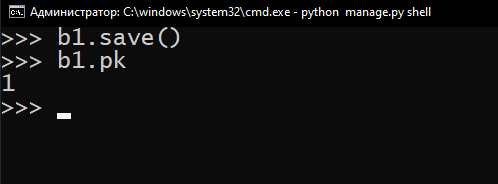
1. Работа с моделями

Создадим первое объявление — первую запись модели Bb:



Запись модели создается аналогично экземпляру любого другого класса — вызовом конструктора. Значения полей можно указать в именованных параметрах.

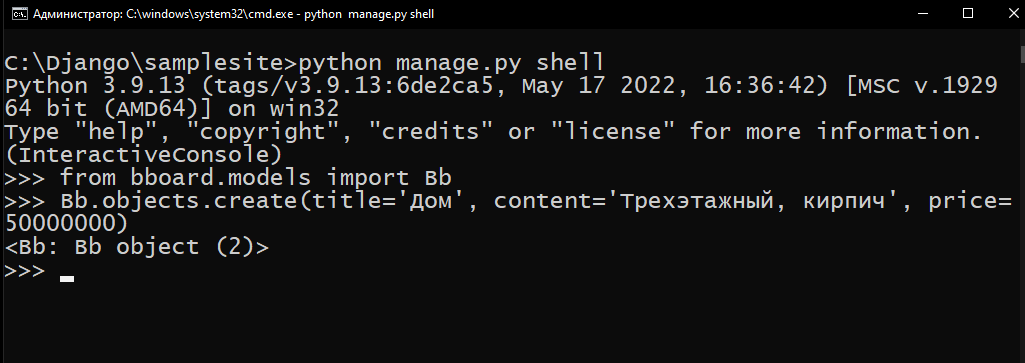
Созданная таким образом запись модели не сохраняется в базе данных, а существует только в оперативной памяти. Чтобы сохранить ее, достаточно вызвать у нее метод save() без параметров:



Проверим, сохранилось ли наше первое объявление, получив значение ключевого поля:

Отлично! Сохранилось.

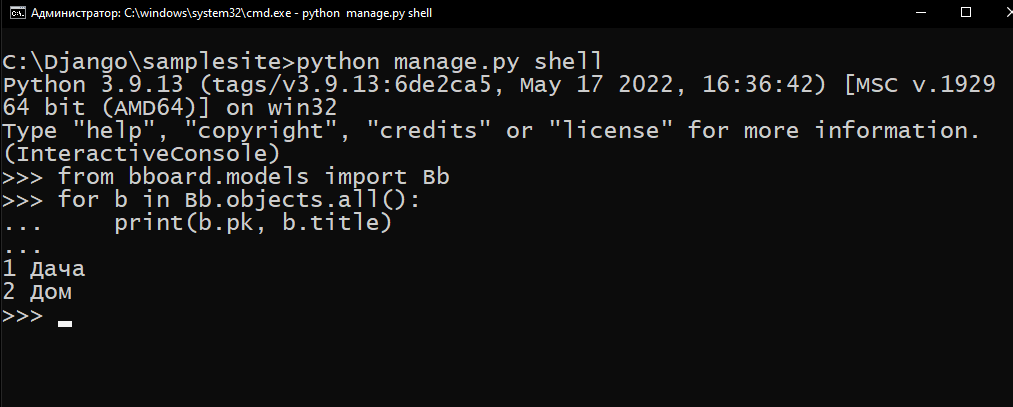
Добавим еще одно объявление:



Все классы моделей поддерживают атрибут класса objects. Он хранит диспетчер записей — объект, представляющий все имеющиеся в модели записи и являющийся экземпляром класса Manager.

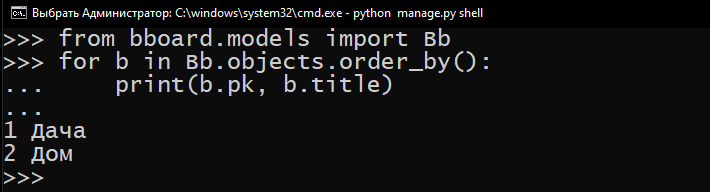
Метод create() диспетчера записей создает новую запись модели, принимая в качестве набора именованных параметров значения ее полей, сразу же сохраняет ее и возвращает в качестве результата.

Выведем ключи и заголовки всех объявлений, имеющихся в модели Bb:



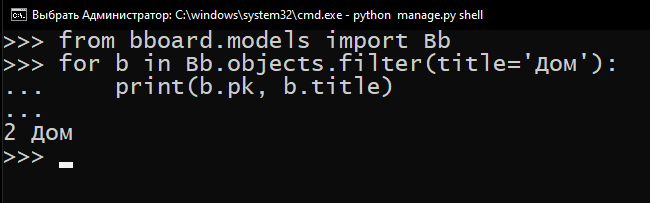
Метод all() диспетчера записей возвращает набор записей — последовательность из всех записей модели, которую можно перебрать в цикле. Сам набор записей представляется экземпляром класса queryset, а отдельные записи — экземплярами соответствующего класса модели.

Отсортируем записи модели по заголовку:



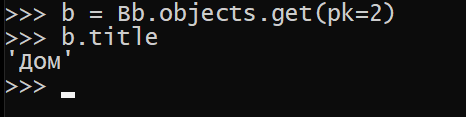
Метод order\_by() диспетчера записей сортирует записи по значению поля, имя которого указано в параметре, и сразу же возвращает набор записей, получившийся в результате сортировки.

Извлечем объявления о продаже домов:

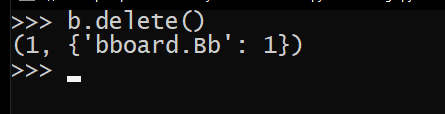


Метод filter() диспетчера записей фильтрует записи по заданным критериям. В частности, чтобы получить только записи, у которых определенное поле содержит заданное значение, следует указать в вызове этого метода именованный параметр, чье имя совпадает с именем поля, и присвоить ему значение, которое должно содержаться в указанном поле. Метод возвращает другой диспетчер записей, содержащий только отфильтрованные записи.

Объявление о продаже автомобиля имеет ключ 2. Отыщем его:



Метод get() диспетчера записей имеет то же назначение, что и метод filter(), и вызывается аналогичным образом. Однако он ищет не все подходящие записи, а лишь одну и возвращает ее в качестве результата.

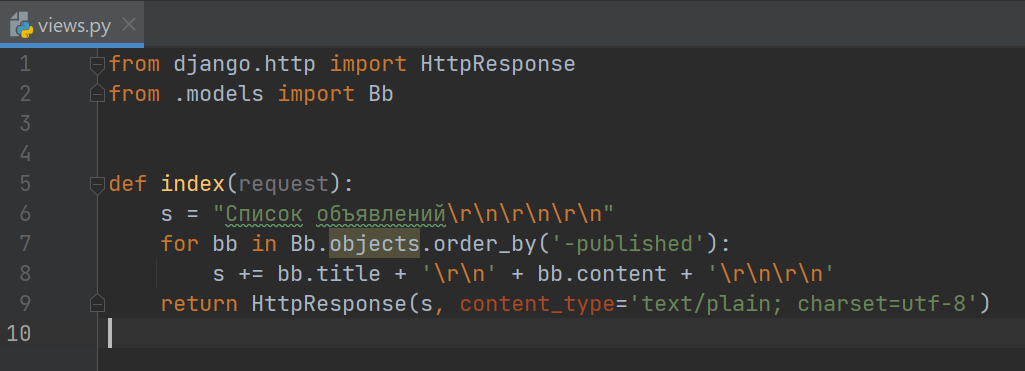


Метод delete() модели, как уже понятно, удаляет текущую запись и возвращает сведения о количестве удаленных записей, обычно малополезные.

Выйдем из консоли Django, набрав команду exit().

И сделаем так, чтобы контроллер index() выводил список объявлений, отсортированный по убыванию даты их публикации.

Откроем модуль views.py пакета приложения bboard и исправим хранящийся в нем код



Чтобы отсортировать объявления по убыванию даты их публикации, мы в вызове метода order\_by() диспетчера записей предварили имя поля published символом "минус". Список объявлений мы представили в виде обычного текста, разбитого на строки символами \r\n.

При создании экземпляра класса httpresponse, представляющего отсылаемый клиенту ответ, в именованном параметре content\_type конструктора указали тип отправляемых данных: обычный текст, набранный в кодировке UTF-8 (если мы этого не сделаем, веб-обозреватель посчитает текст HTML-кодом и выведет его одной строкой, скорее всего, в нечитаемом виде).

Сохраним исправленный файл и запустим отладочный веб-сервер.

Точно так же можно сгенерировать полноценную веб-страницу. Но есть более простой способ - применение шаблонов.

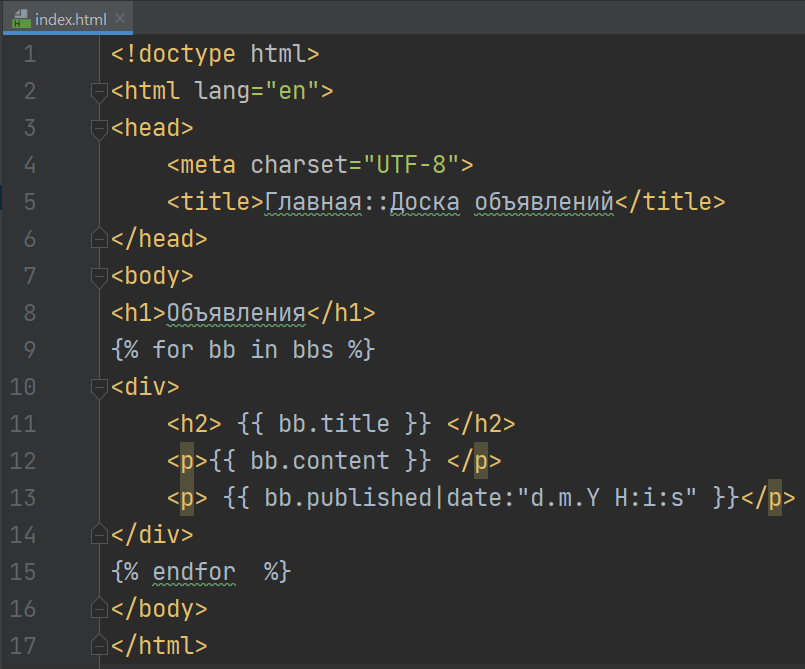
1. Шаблоны

**Шаблон** — это образец для генерирования веб-страницы, отправляемой клиенту в составе ответа. Генерированием страниц на основе шаблонов занимается подсистема Django, называемая шаблонизатором.

Шаблон Django — это файл с НТМL-кодом страницы, содержащий особые команды шаблонизатора: директивы, теги и фильтры. Директивы указывают поместить в заданное место НТМL-кода какое-либо значение, теги управляют генерированием содержимого, а фильтры выполняют какие-либо преобразования указанного значения перед выводом.

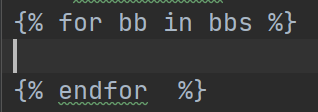
По умолчанию шаблонизатор ищет все шаблоны в папках templates, вложенных в папки пакетов приложений. Сами файлы шаблонов веб­ страниц должны иметь расширение html.

Остановим отладочный сервер. Создадим в папке пакета приложения bbоаrd папку templates, а в ней - вложенную папку bboard. Сохраним в этой папке наш первый шаблон index.html, код которого приведен:



В целом здесь нам все знакомо. За исключением команд шаблонизатора. Давайте познакомимся с ними.

Начнем вот с этого тега шаблонизатора:



Аналогично циклу for . . . In языка Python, он перебирает последовательность, хранящуюся в переменной bbs (которая входит в состав контекста шаблона, о котором мы поговорим чуть позже), присваивая очередной элемент переменной bb, которая доступна в теле цикла. У нас переменная bbs будет хранить перечень объявлений, и, таким образом, переменной bb будет присваиваться очередное объявление.

Теперь познакомимся с директивой шаблонизатора:



Она указывает извлечь значение из атрибута title объекта, хранящегося в переменной bb, и вставить это значение в то место кода, в котором находится она сама.

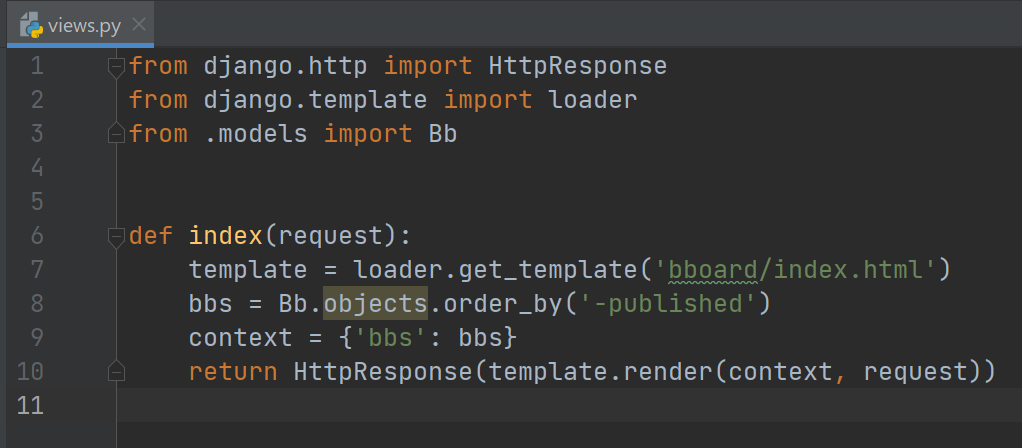
И, наконец, фильтр date:



Он преобразует значение из атрибута published объекта bb, т. Е. Временну'ю отметку публикации объявления, в формат, указанный в виде строки после двоеточия. Строка "d .m.Y H: i : s " задает формат <число> .<номер месяца>. <год из четырех цифр> <часы в 24-часовом формате>: <минуты>:<секунды>.

1. Контекст шаблона, рендеринг И СОКРАЩЕНИЯ

Откроем модуль views.py пакета приложения bbоаrd и внесем исправления в его код:

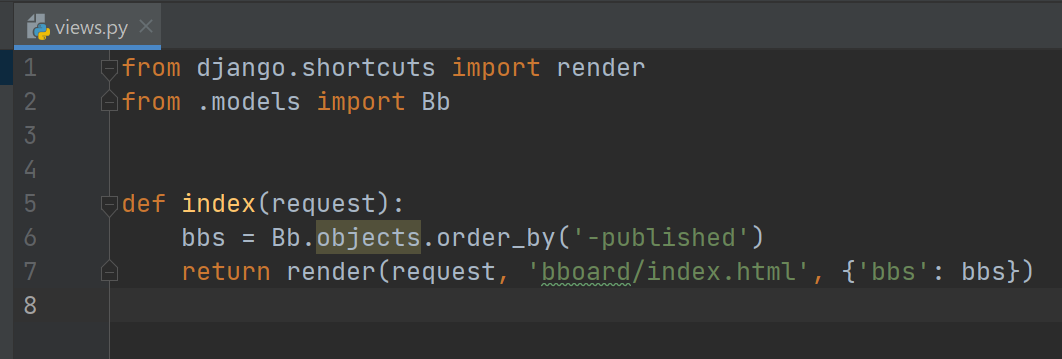


Функция **get\_template** вернет экземпляр класса тemplate, представляющий загруженный из заданного файла шаблон.

Далее формируем контекст шаблона - набор данных, которые будут выведены на генерируемой странице. Контекст шаблона должен представлять собой обычный словарь Python, элементы которого преобразуются в доступные внутри шаблона переменные, одноименные ключам этих элементов. Так, элемент bbs создаваемого нами контекста шаблона, содержащий перечень объявлений, будет преобразован в переменную bbs, доступную в шаблоне.

Наконец, выполняем рендеринг шаблона, т. Е. Генерирование на его основе веб­ страницы. Для этого вызываем метод render ( ) класса Template, передав ему подготовленный ранее контекст шаблона и экземпляр класса httprequest, представляющий клиентский запрос и полученный контроллером-функцией через параметр request. Результат - строку с НТМL-кодом готовой веб-страницы - передаем конструктору класса нttpresponse для формирования ответа.

В коде контроллера index ( ) (см. Код далее для рендеринга мы использовали низкоуровневые инструменты, несколько усложнив код. Но Django предоставляет средства более высокого уровня - функции-сокращения (shortcuts). Так, функция­ сокращение render ( ) из модуля django . Shortcuts выполняет и загрузку, и рендеринг шаблона. Попробуем ее в деле, исправив код модуля views.py:



Обновим веб-страницу со списком объявлений и убедимся, что этот код работает точно так же, как и написанный ранее, однако при этом имеет меньший объем.

1. Административный веб-сайт Django

Все-таки два объявления — это слишком мало. Давайте добавим еще несколько. Только сделаем это не в консоли Django, а на встроенном в этот фреймворк административном сайте.

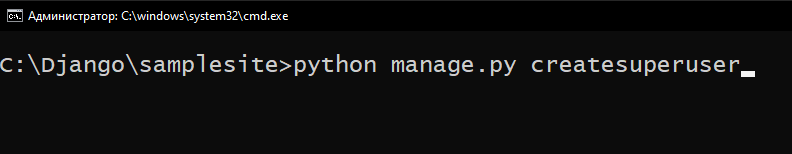
Административный веб-сайт предоставляет доступ ко всем моделям, объявленным во всех приложениях проекта. Мы можем просматривать, добавлять, править и удалять записи, фильтровать и сортировать их. Помимо этого, административный сайт не пускает к данным сайта посторонних, используя для этого встроенную во фреймворк подсистему разграничения доступа.

Эта подсистема реализована в стандартном приложении dj ango . Contrib . Auth.

Работу самого административного сайта обеспечивает стандартное приложение dj ango . Contrib . Admin. Оба этих приложения заносятся в список зарегистрированных в проекте изначально.

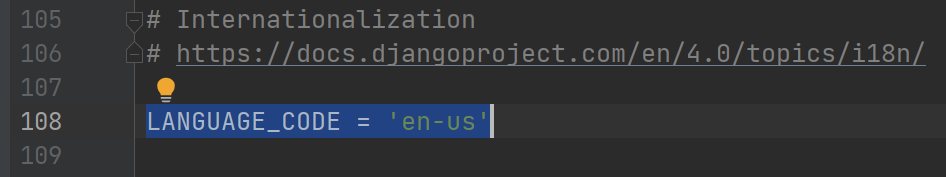
Стандартное приложение django . Contrib . Auth использует для хранения списков зарегистрированных пользователей, групп и их привилегий особые модели. Для них в базе данных должны быть созданы таблицы, и создание этих таблиц выполняют особые миграции. Следовательно, чтобы встроенные в Django средства разграничения доступа работали, нужно хотя бы один раз выполнить миграции

Еще нужно создать зарегистрированного пользователя сайта с максимальными правами - суперпользователя. Для этого остановим отладочный веб-сервер и отдадим в командной строке команду:

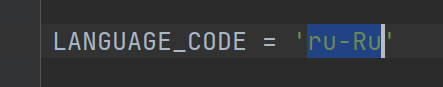


Утилита manage.py запросит у нас имя создаваемого суперпользователя, его адрес электронной почты и пароль, который потребуется ввести дважды. Как только суперпользователь будет успешно создан, появится уведомление.

После этого русифицируем проект Django. Оrкроем модуль settings.py пакета кон­ фигурации и найдем в нем вот такое выражение:



Переменная LANGUAGE\_ CODE задает код языка, используемого при выводе системных сообщений и страниц административного сайта. Изначально это американский английский язык код en-us. Исправим это выражение, занеся в него код русского языка:

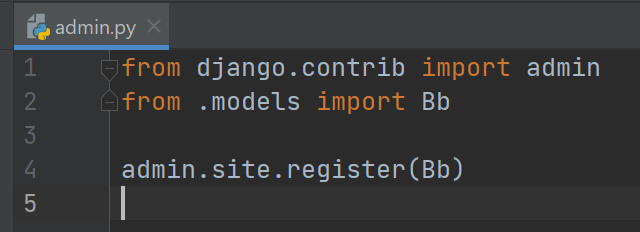


Сохраним исправленный модуль и закроем его - более он нам не понадобится.

Запустим отладочный веб-сервер и войдем на административный сайт, перейдя по интернет-адресу http://localhost:8000/admin/. Будет выведена страница входа с формой, в которой нужно набрать имя и пароль, введенные при создании суперпользователя, и нажать кнопку Войти.

Если мы ввели имя и пароль пользователя без ошибок, то увидим страницу со списком приложений, зарегистрированных в проекте и объявляющих какие-либо модели. Под названием каждого приложения перечисляются объявленные в нем модели.

Чтобы приложение появилось в списке административного сайта, его нужно явно зарегистрировать там. Сделать это очень просто. Откроем модуль административных настроек admin.py пакета приложения bbоаrd и заменим имеющийся в нем небольшой код фрагментом:



Мы вызвали метод register ( ) у экземпляра класса admins ite, представляющего сам административный сайт и хранящегося в переменной site модуля dj ango . Contrib. Admin. Этому методу мы передали в качестве параметра ссылку на класс нашей модели bb.

Как только мы сохраним модуль и обновим открытую в веб-обозревателе страницу списка приложений, сразу увидим, что наше приложение также присутствует в списке.

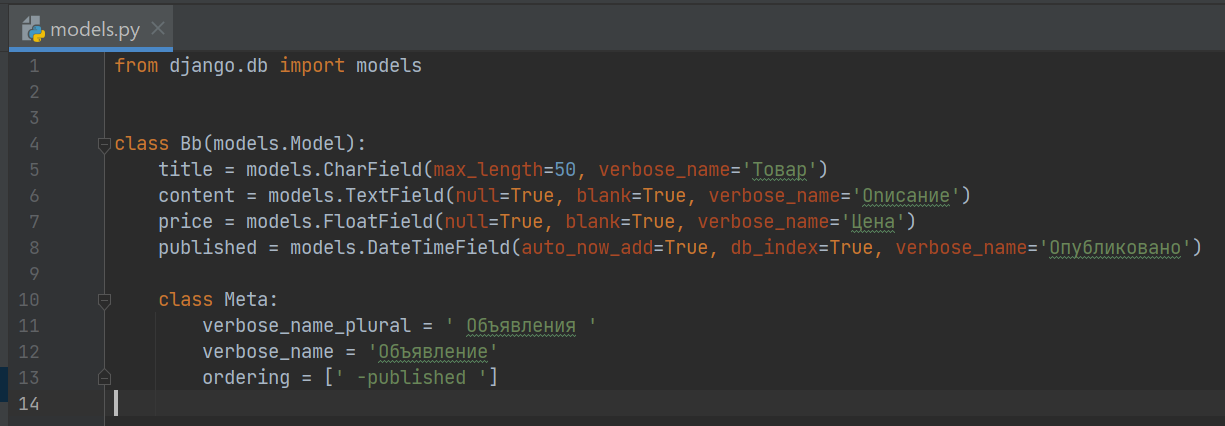
Мы можем попробовать ради эксперимента добавить несколько записей в модель bb, исправить кое-какие записи и удалить ненужные.

1. Параметры полей и моделей

Во-первых, наша модель представляется какой-то непонятной аббревиатурой Bbs, а не простым и ясным текстом Объявления. Во-вторых, на страницах добавления и правки записи в качестве надписей у элементов управления проставлены имена полей модели (title, content и price), что обескуражит пользователя. И в-третьих, объявления было бы неплохо отсортировать по убыванию даты публикации.

Одним словом, нам надо задать параметры как для полей модели, так и для самой модели.

Откроем модуль models. Py пакета приложения ььоаrd и исправим код класса модели вь, как показано:



В вызов каждого конструктора класса поля мы добавили именованный параметр verbose\_name. Он указывает "человеческое" название поля, которое будет выводиться на экран.

В классе модели мы объявили вложенный класс меtа, а в нем - атрибуты класса, которые зададут параметры уже самой модели:

Verbose\_name\_plural - название модели во множественном числе

Verbose\_name - название модели в единственном числе

Ordering - последовательность полей, по которым по умолчанию будет выполняться сортировка записей.

1. Редактор модели

Если же нас не устраивает отображение модели в списках, мы можем задать свои параметры представления модели, объявив для нее класс-редактор.

Редактор объявляется в модуле административных настроек admin.py пакета приложения. Откроем его и заменим имеющийся в нем код:

****

Класс редактора объявляется как производный от класса modeladmin из модуля dj ango . Contrib . Admin. Он содержит набор атрибутов класса, которые и задают параметры представления модели. Мы использовали следующие атрибуты класса:

List\_display - последовательность имен полей, которые должны выводиться в списке записей.

List\_display\_links - последовательность имен полей, которые должны быть преобразованы в гиперссылки, ведущие на страницу правки записи.

Search\_fields - последовательность имен полей, по которым должна выполняться фильтрация.

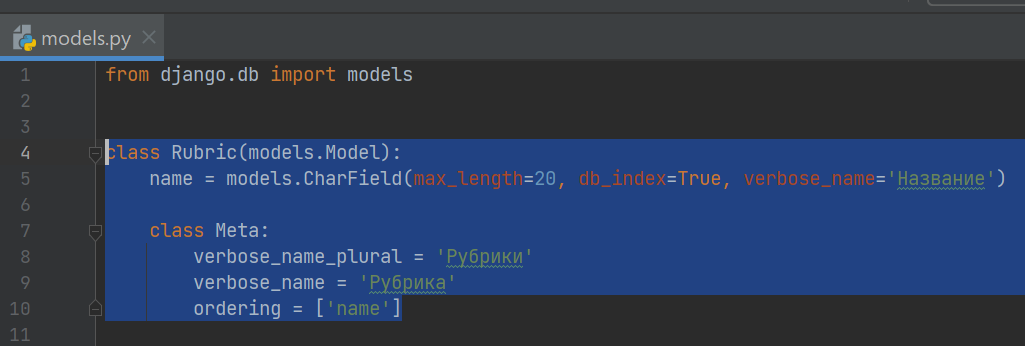
Обновим открытый в веб-обозревателе административный сайт и перейдем на страницу со списком записей модели.

Помимо всего прочего, мы можем выполнять фильтрацию записей по значениям полей, перечисленных в последовательности, которая была присвоена атрибуту search fields класса редактора. Для этого достаточно занести в расположенное над списком поле ввода искомое слово и нажать находящуюся правее кнопку Найти.

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.

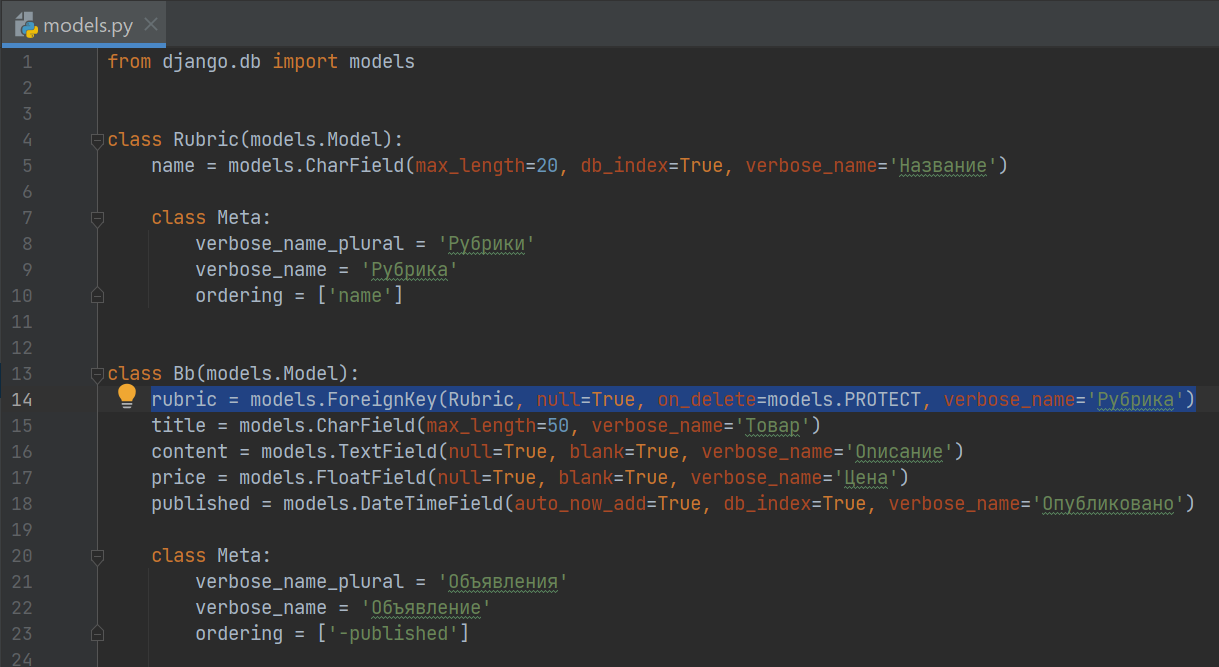
Теперь нужно добавить в модель Bb поле внешнего ключа, устанавливающее связь между текущей записью этой модели и записью модели Rubric, т. Е. Между объявлением и рубрикой, к которой оно относится. Таким образом будет создана связь "один-со-многими", при которой одна запись модели Rubric (рубрика) будет 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.

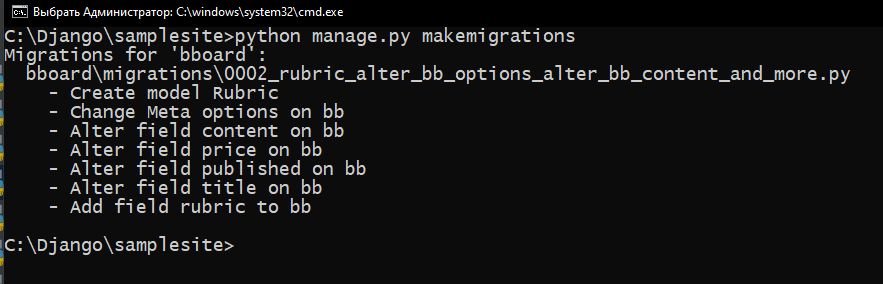
Все поля моделей по умолчанию обязательны к заполнению. Следовательно, добавить новое, обязательное к заполнению поле в модель, которая уже содержит записи, нельзя - сама СУБД откажется делать это и выведет сообщение об ошибке.

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



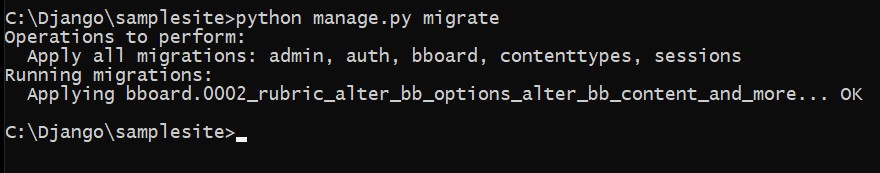
Именованный параметр on\_delete управляет каскадными удалениями записей вторичной модели после удаления записи первичной модели, с которой они были связаны. Значение PROTECT этого параметра запрещает каскадные удаления (чтобы какой-нибудь несообразительный администратор не стер разом уйму объявлений,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

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

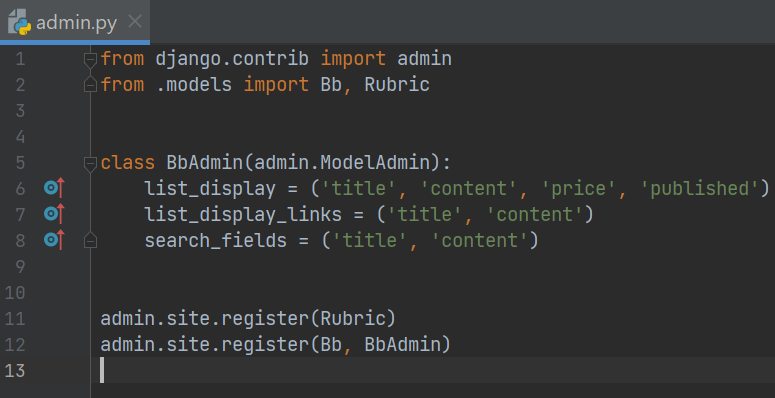


В результате в папке migrations будет создан модуль миграции с именем вида 0002\_auto\_ <отметка текущих даты и времени>.ру.

Выполним созданную миграцию:



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

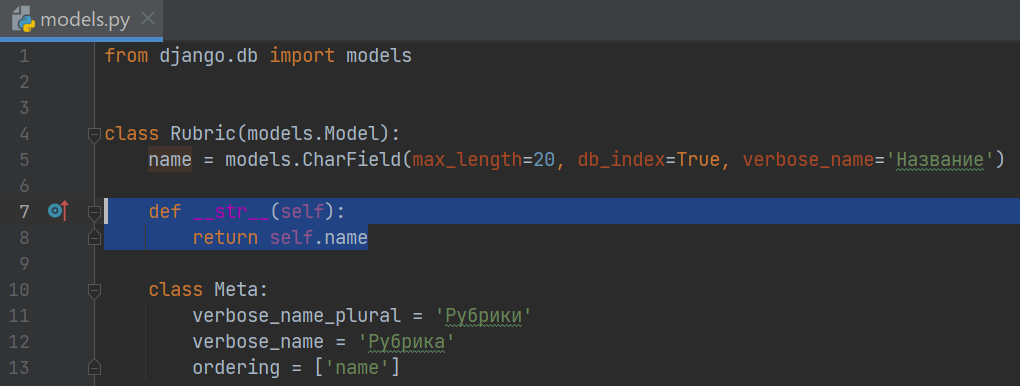


Запустим отладочный веб-сервер, войдем на административный сайт и добавим в модель Rubric рубрики "Недвижимость" и "Транспорт".

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.

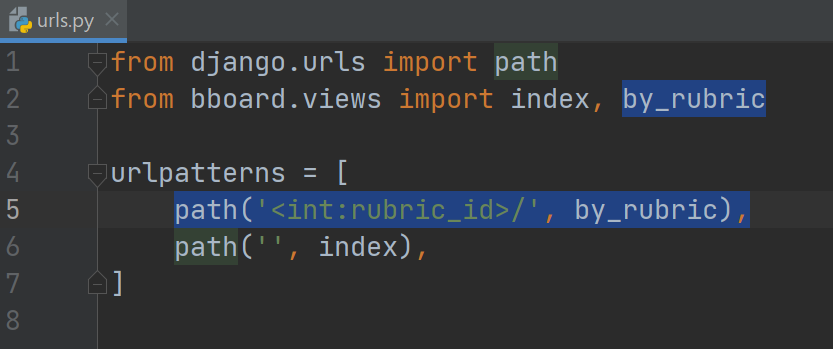
1. URL-параметры и параметризованные запросы

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.

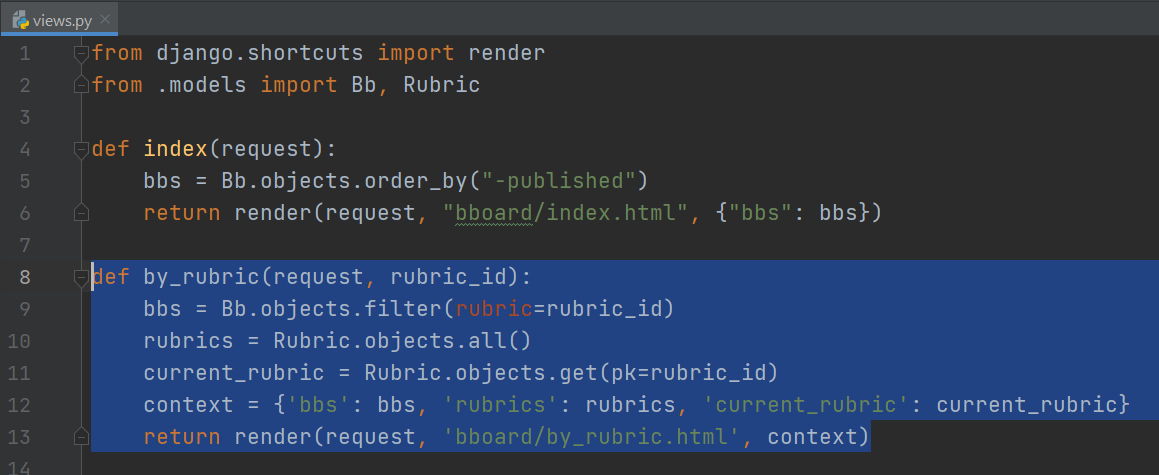
****

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.

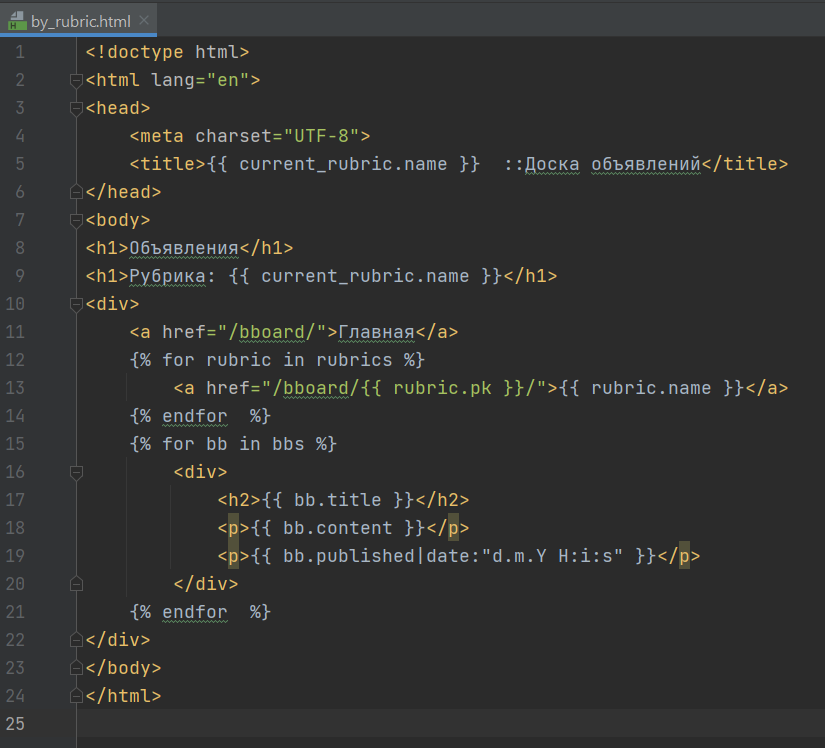
Маршруты, содержащие URL-параметры, носят название параметризованных.

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



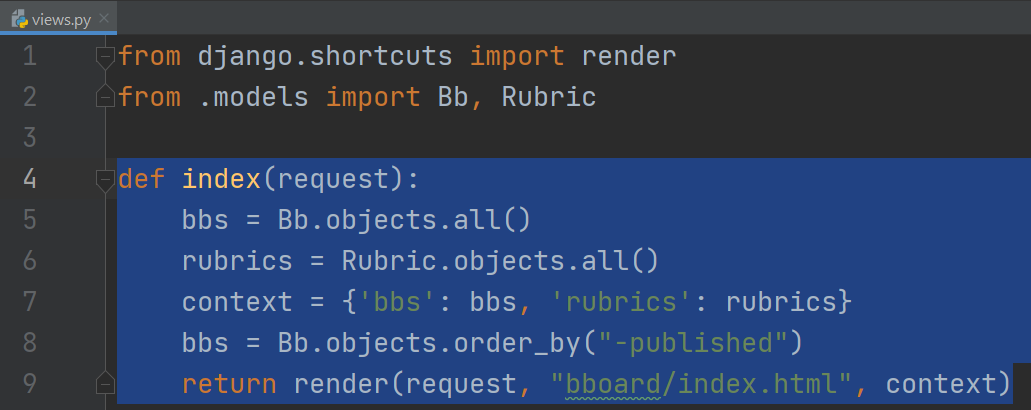
В объявление функции мы добавили параметр rubric\_id - именно ему будет присвоено значение URL-параметра, выбранное из интернет-адреса. В состав контекста шаблона поместили список объявлений, отфильтрованных по полю внешнего ключа rubric, список всех рубрик и текущую рубрику (она нужна нам, чтобы вы­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

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

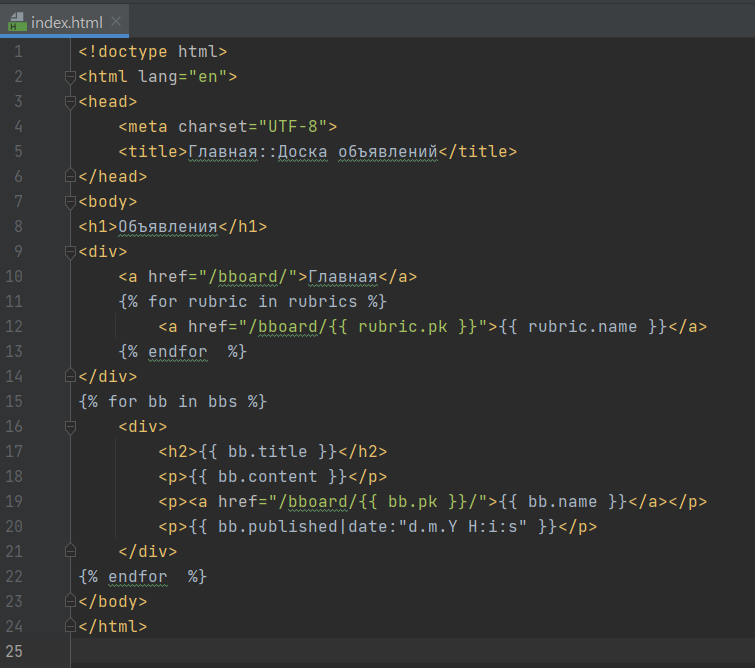


Обратим внимание, как формируются интернет-адреса в гиперссылках, находящихся в панели навигации и представляющих отдельные рубрики.

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

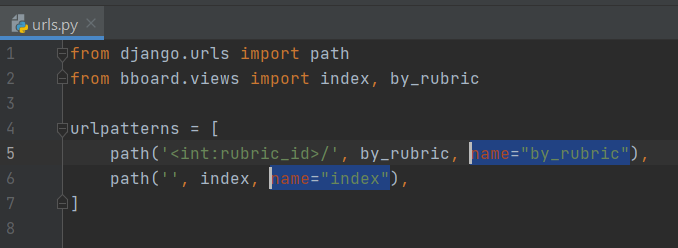
Посмотрим на НТМL-код гиперссылок



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.

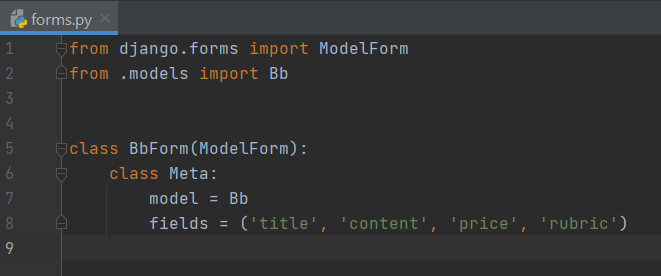
Осталось создать еще одну страницу - для добавления новых объявлений.

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:

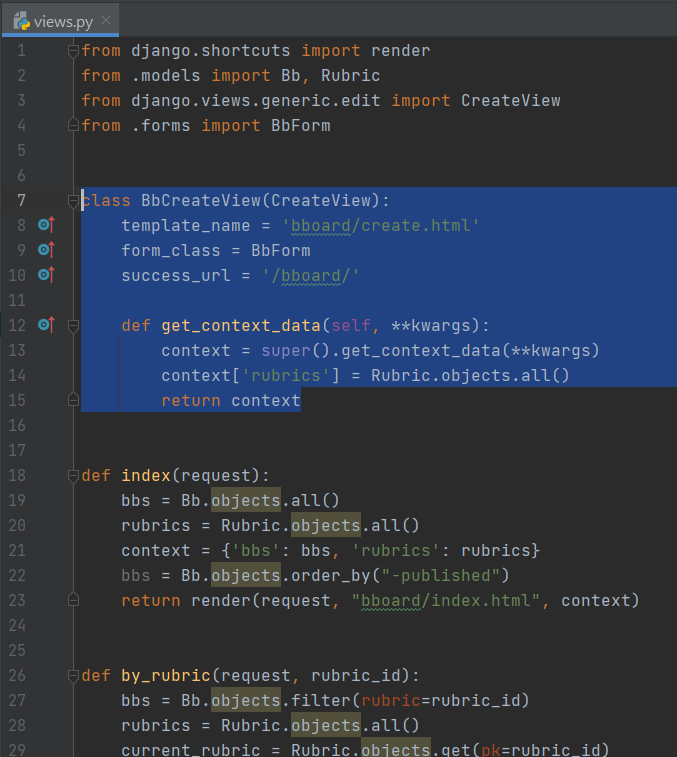


Класс формы, связанной с моделью, является производным от класса modelfoпn из модуля dj ango . Forms. Во вложенном классе Meta указываются параметры формы: класс модели, с которой она связана (атрибут класса model , и последовательность из имен полей модели, которые должны присутствовать в форме (атрибут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

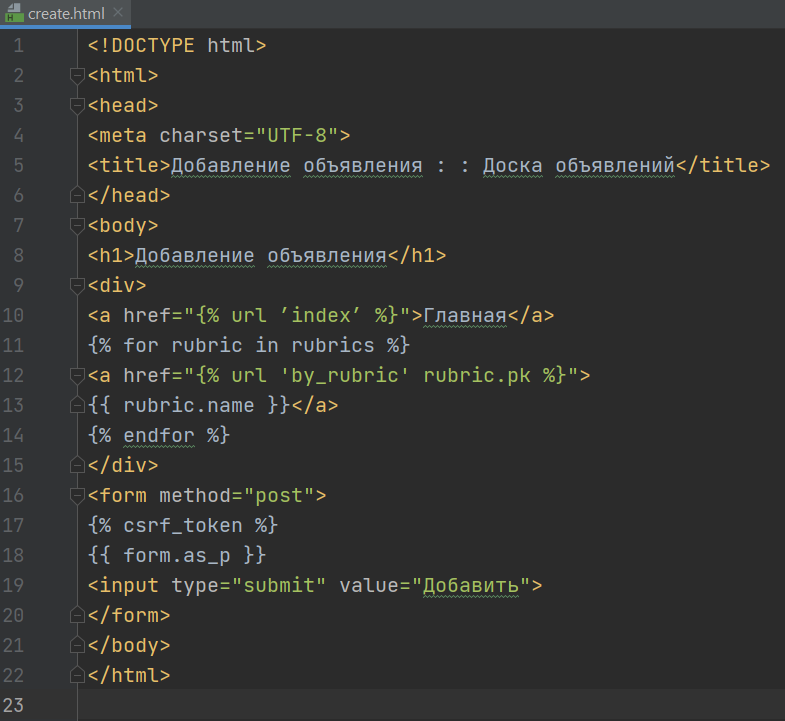
(] template\_name - путь к файлу шаблона, создающего страницу с формой;

(] 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;

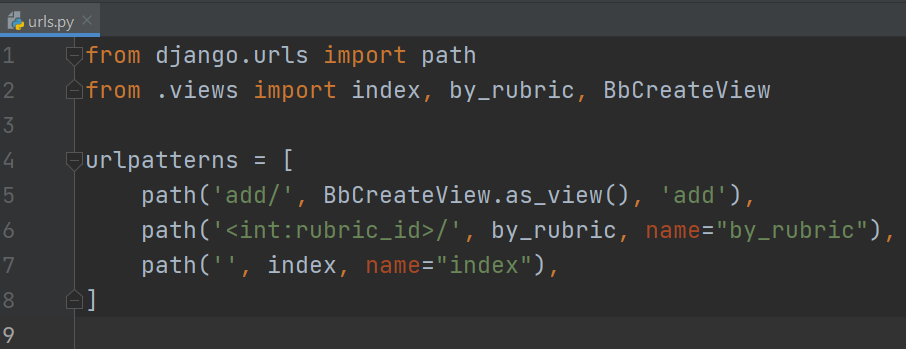
Метод as\_Р () генерирует только код, создающий элементы управления. Тег <form>, необходимый для создания самой формы, и тег <input>, формирующий кнопку отправки данных, придется писать самостоятельно.

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:



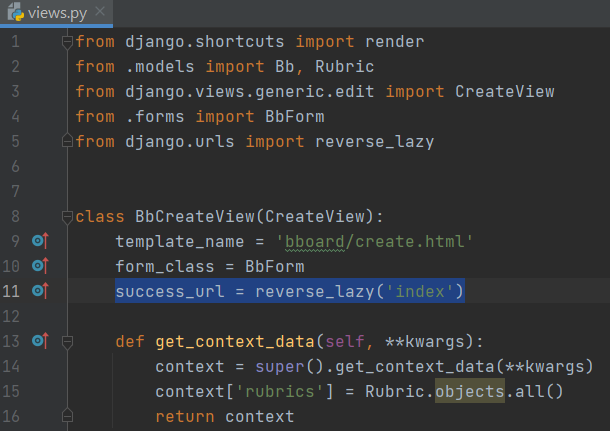
Не забудем создать в панели навигации всех страниц гиперссылку на страницу добавления объявления:



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.

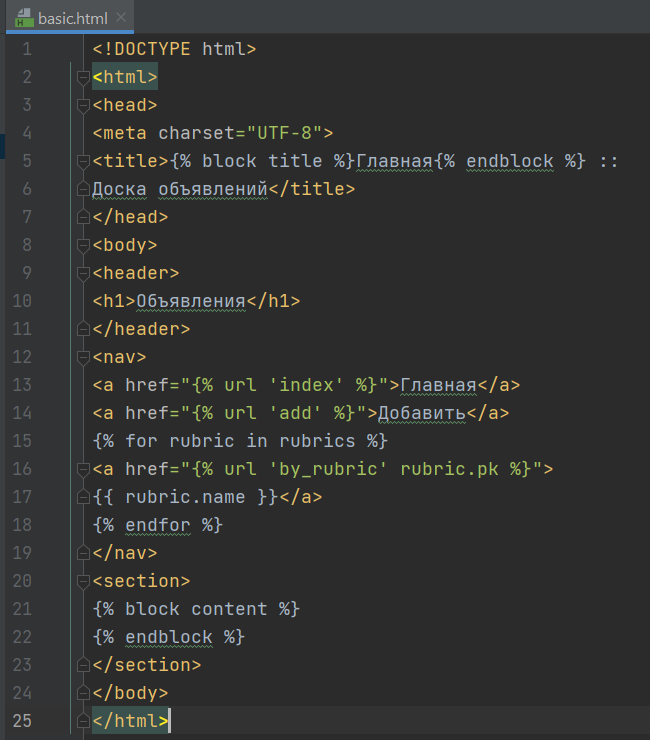
Решить эту проблему можно, применив наследование шаблонов, аналогичное наследованию классов. Шаблон, являющийся базовым, содержит повторяющийся код, в нужных местах которого объявлены блоки, помечающие места, куда будет вставлено содержимое из производных шаблонов. Каждый блок имеет уникальное в пределах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

Создадим базовый шаблон со следующими блоками:

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

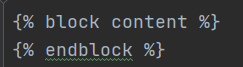
- content - будет использоваться для размещения уникального содержимого страниц.

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:

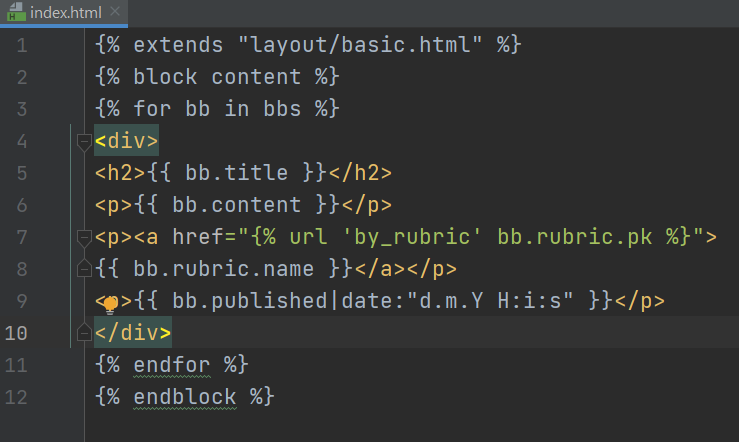


Или иметь какое-либо изначальное содержимое:



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

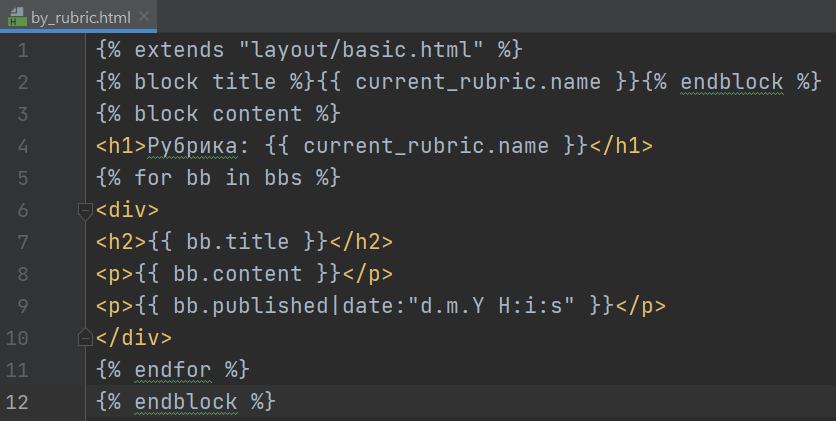
Сделаем шаблон bboard\index.html производным от шаблона layout\basic.html. Новый код

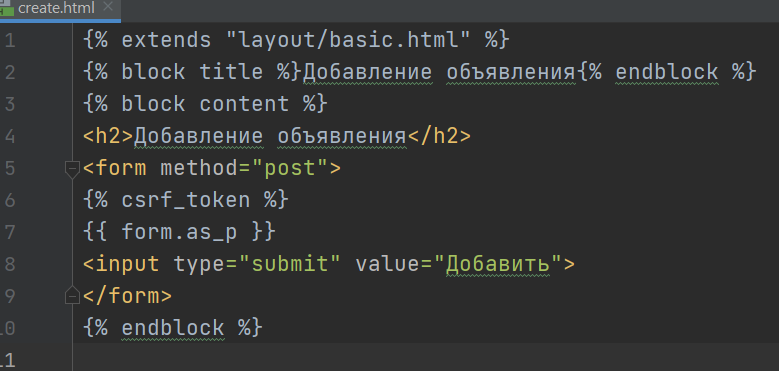


В самом начале кода любого производного шаблона ставится тег шаблонизатора extends, в котором указывается путь к базовому шаблону. Далее следуют объявления блоков, обозначаемые теми же тегами Blосk и endblock, в которых записывается их содержимое.

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

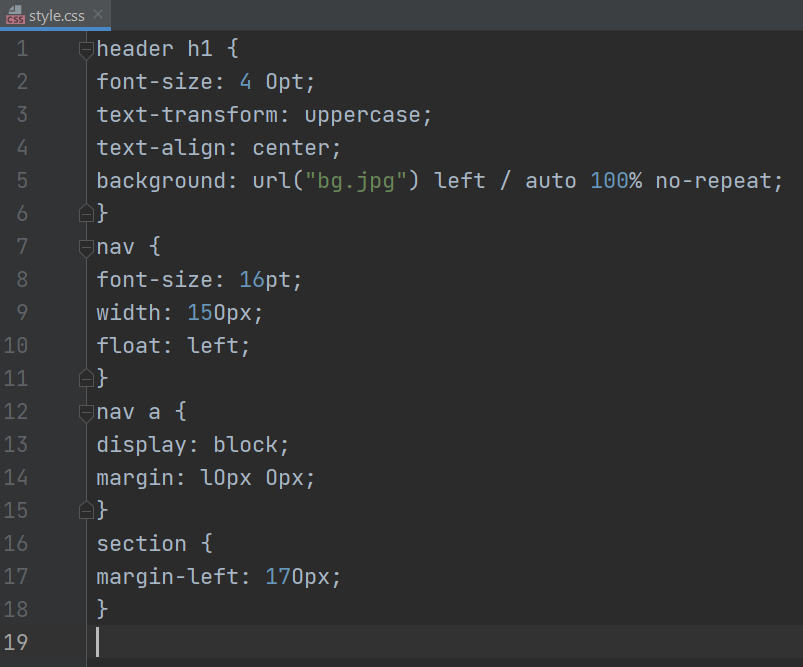
Аналогично исправим шаблоны bboard\Ьy\_rubric.html





1. Статические файлы

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

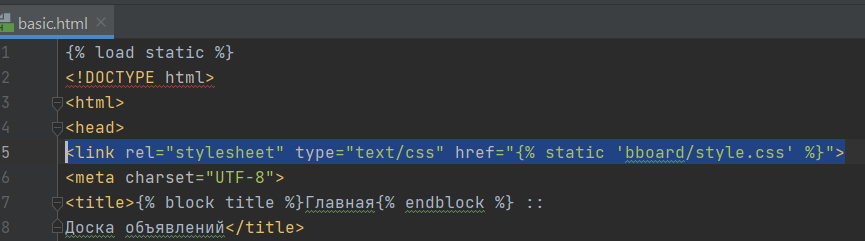


Но где нам сохранить эту таблицу стилей и файл с фоновым изображением bg.jpg?

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.