**Тема занятия № 2: Модуль 1. Основные понятия Django. Вывод данных**

1. Модели

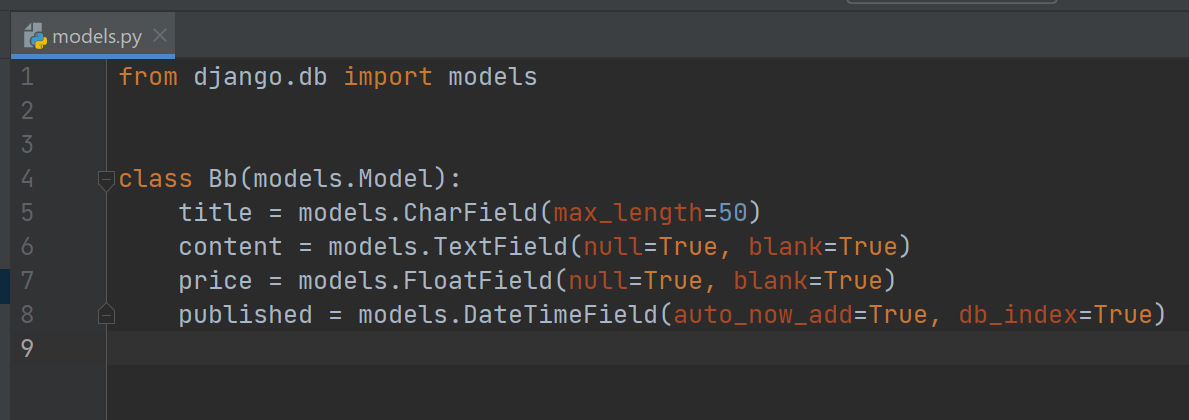
Настала пора сделать так, чтобы вместо намозолившего глаза текстового сообщения выводились реальные объявления, хранящиеся в таблице базы данных. Для этого нам понадобится прежде всего объявить модель.

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 — временнáя 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:

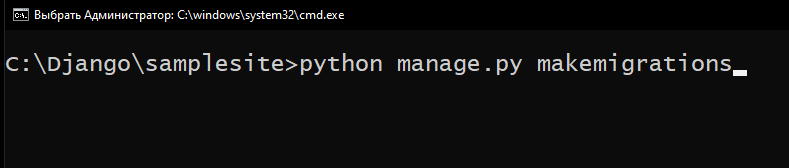
* Charfield — обычное строковое поле фиксированной длины. Допустимая длина значения указывается параметром max\_length конструктора;
* Textfield — текстовое поле неограниченной длины, или memo-поле. Присвоив параметрам null и blank конструктора значения True, мы укажем, что это поле можно не заполнять (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.

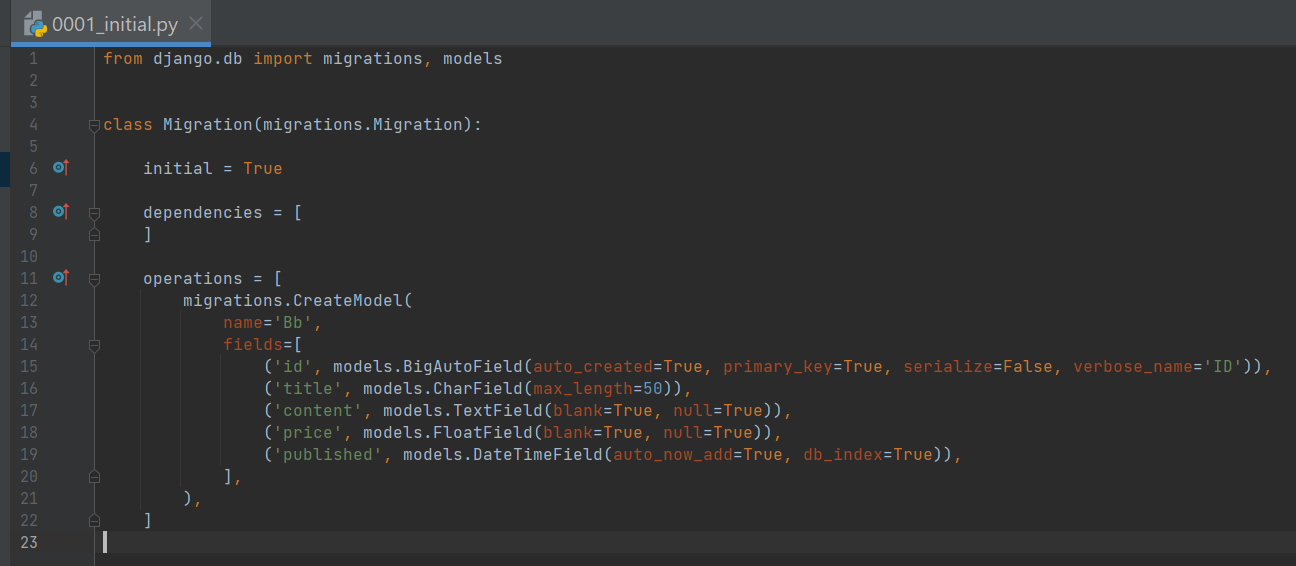
Сохраним исправленный файл. Сейчас мы сгенерируем на его основе миграцию, которая создаст в базе данных все необходимые структуры.

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:



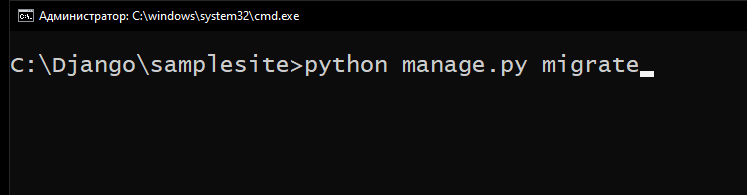
Модуль с кодом нашей первой миграции будет иметь имя **0001\_initial.py**. Откроем его в текстовом редакторе и посмотрим на хранящийся в нем код:



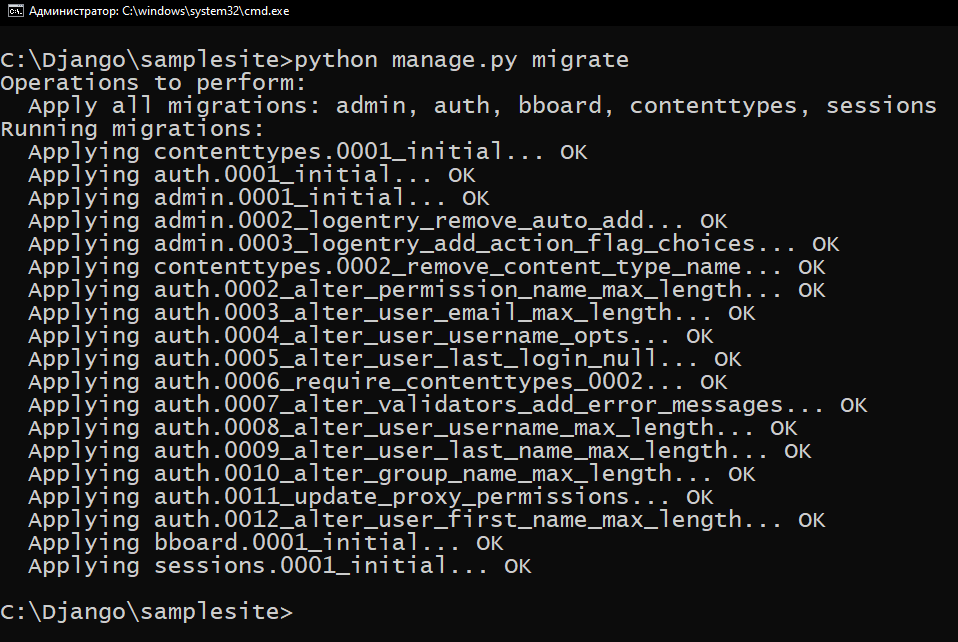
Код миграции вполне понятен и напоминает код написанной ранее модели. Создаваемая в базе данных таблица будет содержать поля id, title, content, price и published. Ключевое поле id для хранения уникальных номеров записей Django создаст самостоятельно.

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.



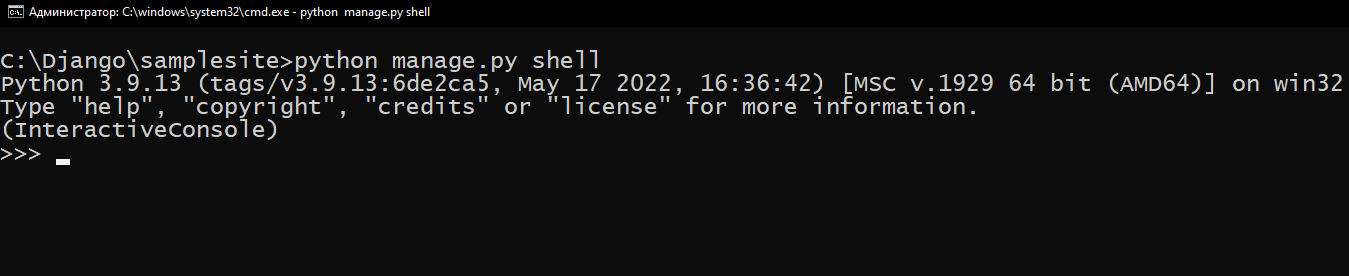
Дождемся, когда их выполнение завершится, и продолжим.

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.

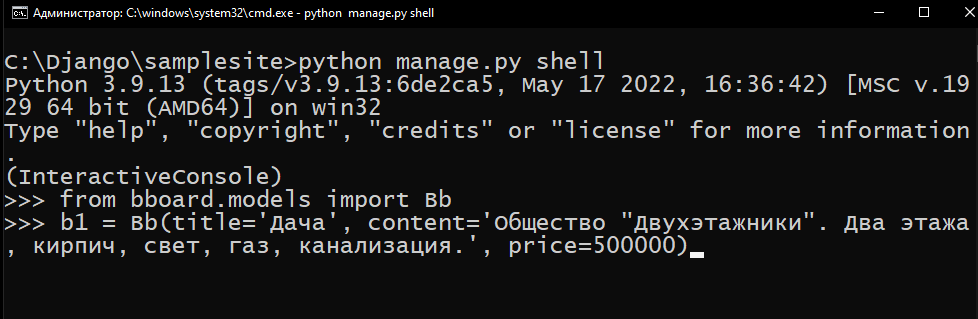
Фреймворк включает в свой состав собственную редакцию консоли Python Shell, называемую консолью Django. От аналогичной командной среды Python она отличается тем, что в ней в состав путей поиска модулей добавляется путь к папке проекта, в которой запущена эта консоль.

В командной строке наберем команду для запуска консоли Django:



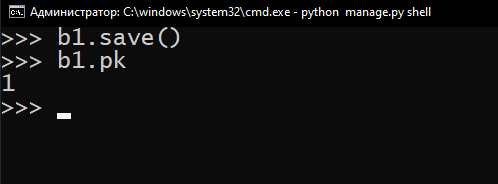
1. Работа с моделями

Создадим первое объявление — первую запись модели Bb:



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

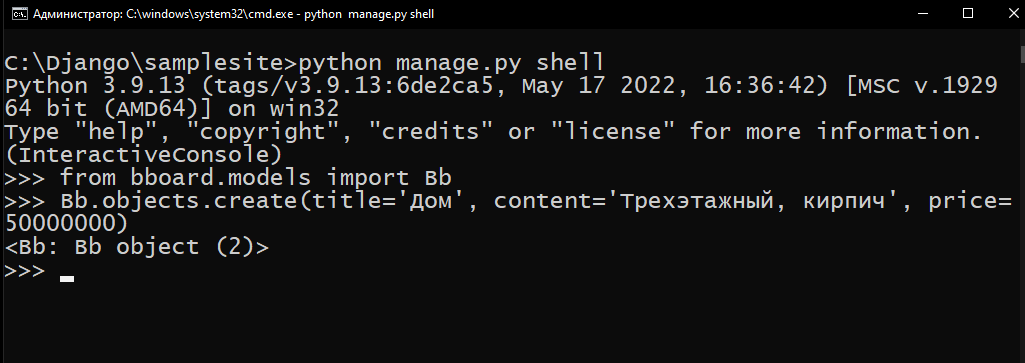
Созданная таким образом запись модели не сохраняется в базе данных, а существует только в оперативной памяти. Чтобы сохранить ее, достаточно вызвать у нее метод save() без параметров:



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

Отлично! Сохранилось.

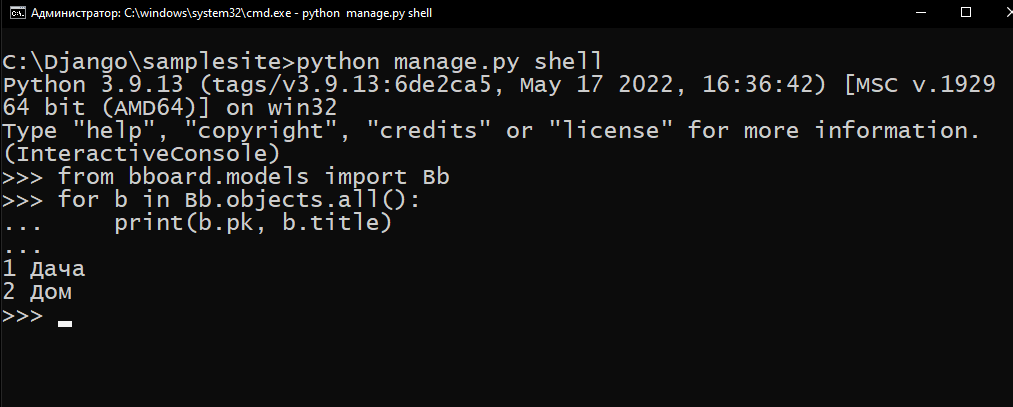
Добавим еще одно объявление:



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

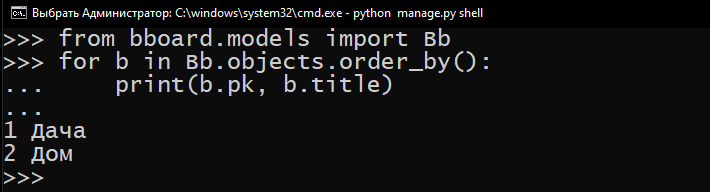
Метод create() диспетчера записей создает новую запись модели, принимая в качестве набора именованных параметров значения ее полей, сразу же сохраняет ее и возвращает в качестве результата.

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



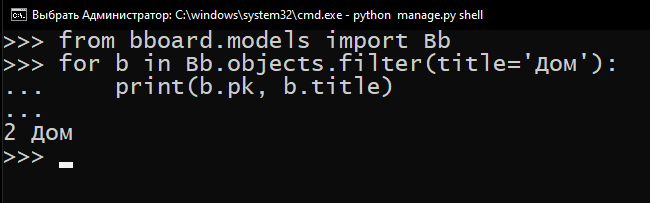
Метод all() диспетчера записей возвращает набор записей — последовательность из всех записей модели, которую можно перебрать в цикле. Сам набор записей представляется экземпляром класса queryset, а отдельные записи — экземплярами соответствующего класса модели.

Отсортируем записи модели по заголовку:



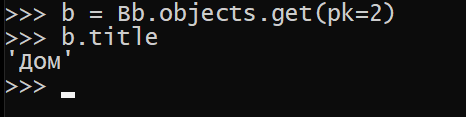
Метод order\_by() диспетчера записей сортирует записи по значению поля, имя которого указано в параметре, и сразу же возвращает набор записей, получившийся в результате сортировки.

Извлечем объявления о продаже домов:

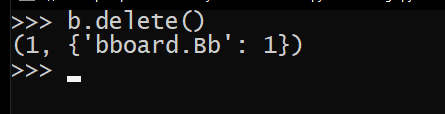


Метод filter() диспетчера записей фильтрует записи по заданным критериям. В частности, чтобы получить только записи, у которых определенное поле содержит заданное значение, следует указать в вызове этого метода именованный параметр, чье имя совпадает с именем поля, и присвоить ему значение, которое должно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

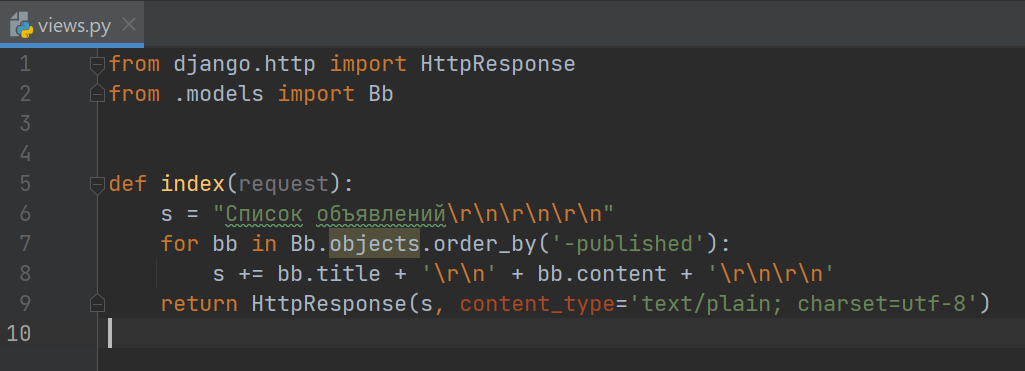


Метод delete() модели, как уже понятно, удаляет текущую запись и возвращает сведения о количестве удаленных записей, обычно малополезные.

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.

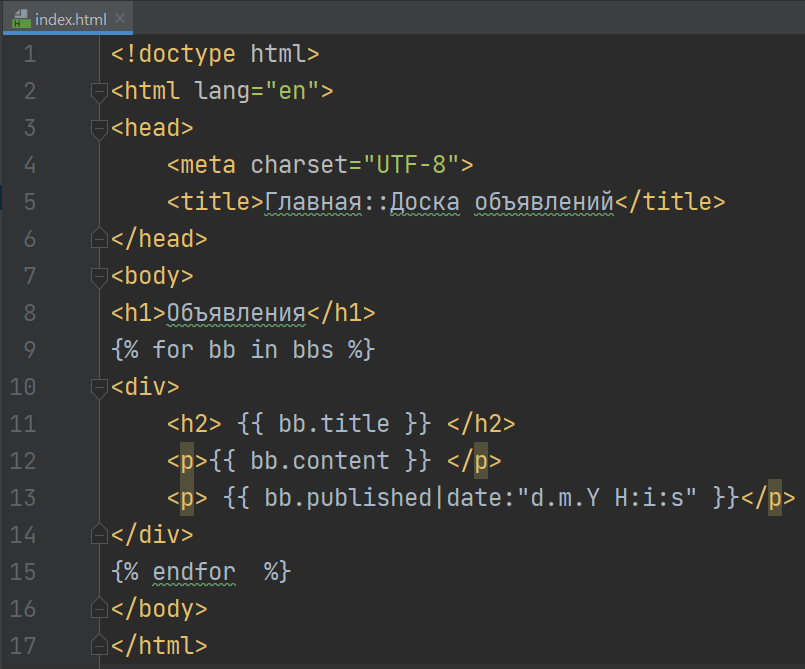
1. Шаблоны

**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.

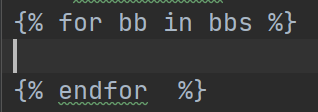
По умолчанию шаблонизатор ищет все шаблоны в папках templates, вложенных в папки пакетов приложений. Сами файлы шаблонов веб­ страниц должны иметь расширение html.

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.

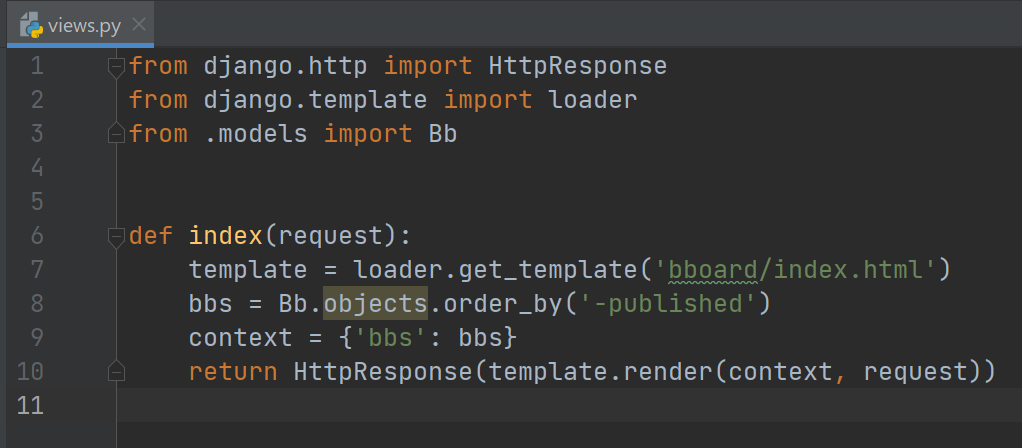
И, наконец, фильтр date:



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:

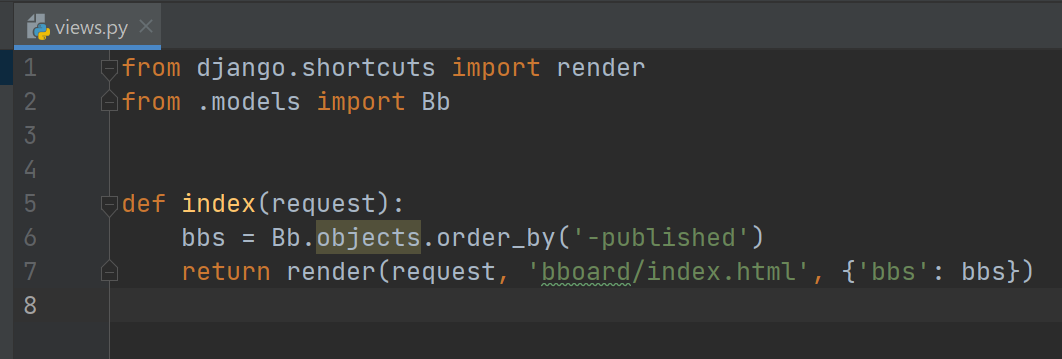


Функция **get\_template** вернет экземпляр класса тemplate, представляющий загруженный из заданного файла шаблон.

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.

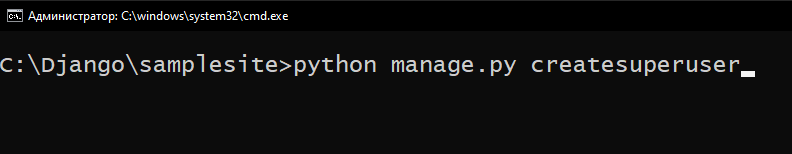
Административный веб-сайт предоставляет доступ ко всем моделям, объявленным во всех приложениях проекта. Мы можем просматривать, добавлять, править и удалять записи, фильтровать и сортировать их. Помимо этого, административный сайт не пускает к данным сайта посторонних, используя для этого встроенную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

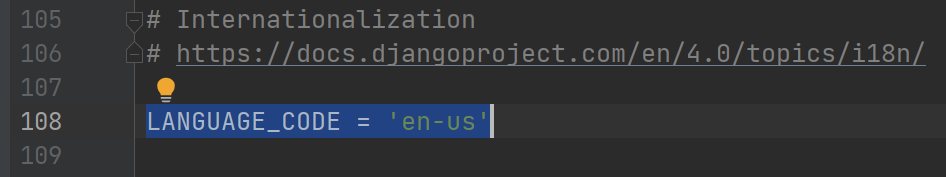
Стандартное приложение django . Contrib . Auth использует для хранения списков зарегистрированных пользователей, групп и их привилегий особые модели. Для них в базе данных должны быть созданы таблицы, и создание этих таблиц выполняют особые миграции. Следовательно, чтобы встроенные в Django средства 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:

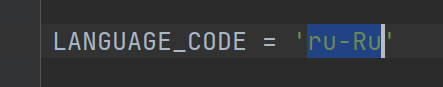


Утилита manage.py запросит у нас имя создаваемого суперпользователя, его адрес электронной почты и пароль, который потребуется ввести дважды. Как только суперпользователь будет успешно создан, появится уведомление.

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:

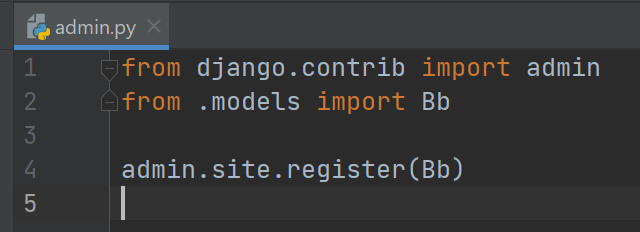


Сохраним исправленный модуль и закроем его - более он нам не понадобится.

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.

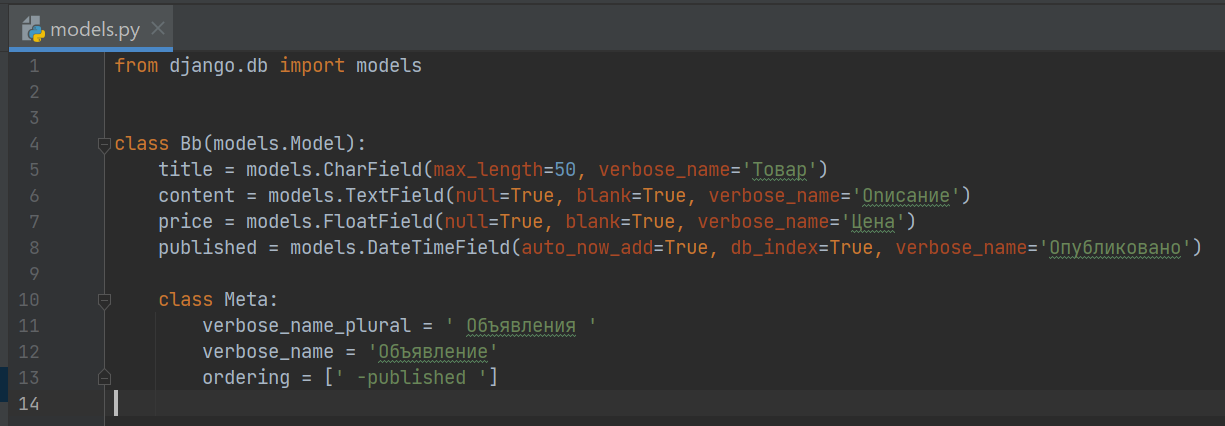
Мы можем попробовать ради эксперимента добавить несколько записей в модель bb, исправить кое-какие записи и удалить ненужные.

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.