**Тема занятие № 60-62: Практическая работа: Доска объявлений**

1. Дизайн. Вспомогательные веб-страницы

Наша электронная доска объявлений позволит зарегистрированным пользователям публиковать объявления о продаже чего-либо. Объявления будут разноситься по рубрикам, причем структура рубрик будет иметь два уровня иерархии: на первом уровне расположатся рубрики общего плана (’’недвижимость”, ’’транспорт” и пр.), а на втором— более конкретные (’’жилье”, ’’гаражи”, ’’дачи”, ’’легковой”, ’’грузовой”, ’’специальный”).

Для вывода списка объявлений мы применим пагинацию, т. к. объявлений может оказаться очень много, и страница, содержащая все объявления, будет слишком большой. Также мы предусмотрим возможность поиска объявлений по введенному посетителем слову.

Под любым объявлением (на странице сведений об объявлении) может быть оставлено произвольное количество комментариев. Оставлять комментарии будет позволено любому пользователю, в том числе и гостю.

В составе объявления пользователь может поместить основную графическую иллюстрацию, которая будет выводиться и в списке объявлений, и в составе сведений об объявлении, а также произвольное количество дополнительных иллюстраций, которые можно будет увидеть лишь на странице сведений об объявлении. И основная, и дополнительные иллюстрации не являются обязательными к размещению.

Процедура регистрации нового пользователя на сайте будет разбита на два этапа.

На первом этапе посетитель вводит свои данные на странице регистрации, после чего на указанный им адрес электронной почты приходит письмо с гиперссылкой, ведущей на страницу активации. На втором этапе посетитель переходит по гиперссылке, полученной в письме, попадает на страницу активации и становится полноправным пользователем.

Сайт доски объявлений включит в себя следующие страницы:

□ главная— показывающая десять последних опубликованных объявлений без разбиения их на рубрики;

□ страница списка объявлений— показывающая (с использованием пагинации) объявления из определенной рубрики. Также она будет содержать форму для поиска объявления по введенному слову;

□ страница сведений о выбранном объявлении— выведет еще все оставленные для него комментарии и форму для добавления нового комментария;

□ страницы регистрации и активации нового пользователя;

□ страницы входа и выхода;

□ страница профиля зарегистрированного пользователя — выведет список объявлений, оставленных текущим пользователем;

□ страницы добавления, правки, удаления объявлений;

□ страницы изменения пароля, правки и удаления пользовательского профиля;

□ страницы сведений о сайте, о правах его разработчика, пользовательского соглашения и пр.

Таков, в самых общих чертах, план разрабатываемого нами сайта. Всевозможные мелочи мы уточним по ходу дела.

**Подготовка проекта и приложения main**

Сейчас мы создадим проект нашего сайта, незатейливо назвав его bboard, и приложение main, которое реализует всю функциональность сайта, за исключением Веб-службы (последняя будет удостоена особого рассмотрения — в будущем мы создадим под нее отдельное приложение api).

**Создание и настройка проекта**

Запустим командную строку, перейдем в папку, в которой будет находиться папка нового проекта, и подадим команду на создание проекта bboard;

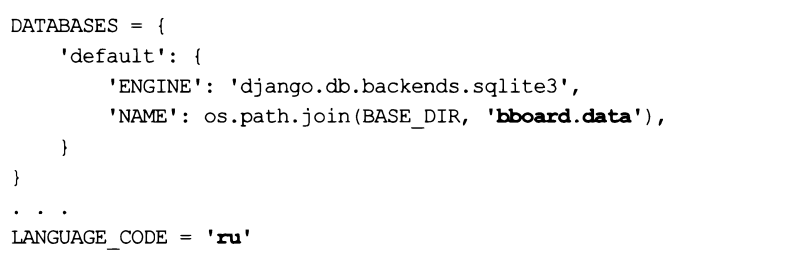
Django-admin startproject bboard

Когда проект будет создан, откроем модуль настроек проекта settings.py из пакета конфигурации и внесем в него следующие правки:

□ изменим имя файла, в котором будет храниться база данных сайта,— на Bboard.data;

□ изменим код языка для вывода системных сообщений и страниц административного сайта — на "ru".

Исправленные фрагменты кода модуля settings.py должны выглядеть так:



Пока этого достаточно. Остальные необходимые правки в настройки проекта мы внесем позднее, по ходу работы.

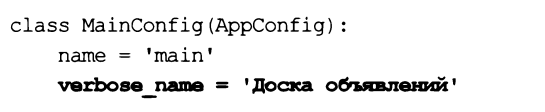
**Создание и настройка приложения main**

В командной строке перейдем в папку проекта и подадим команду на создание приложения main:

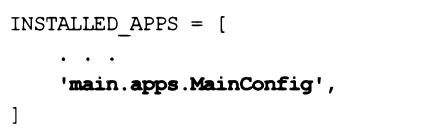


В пакете приложения найдем модуль настроек приложения арр.ру и откроем его.

Добавим В объявление конфигурационного класса mainconfig атрибут verbose\_name с названием приложения:



Вернемся к модулю настроек проекта settings.py и добавим только что созданное приложение в список зарегистрированных в проекте:



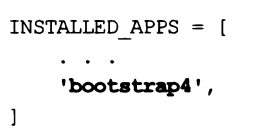
**Базовый шаблон**

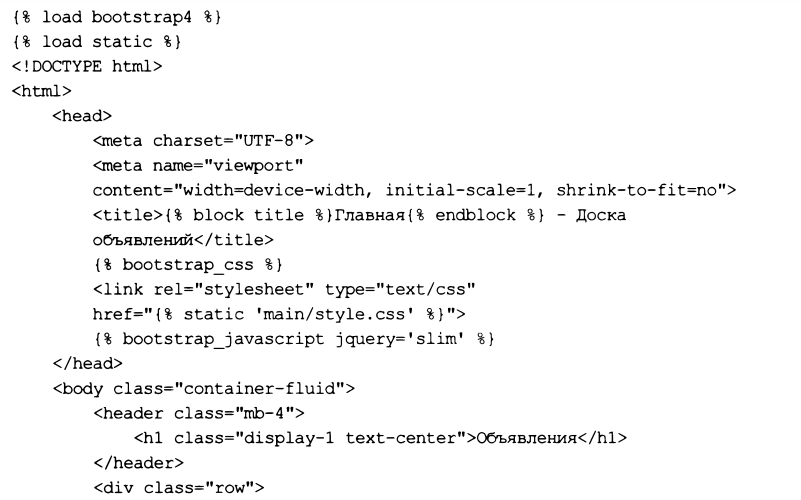
Создадим в пакете приложения main папки templates\layout и static\main. В первой мы сохраним базовый шаблон для страниц сайта, во второй — нашу таблицу стилей.

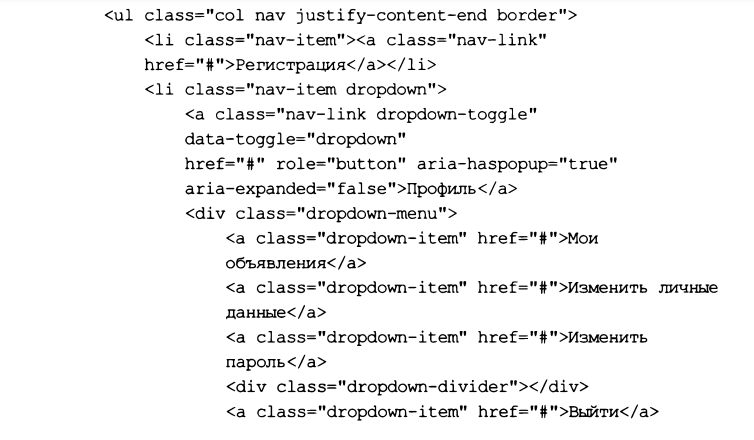
Для оформления страниц используем популярный CSS-фреймворк Bootstrap 4. Сразу же установим дополнительную библиотеку django-bootstrap4 (если не сделали этого ранее), задав в командной строке команду:

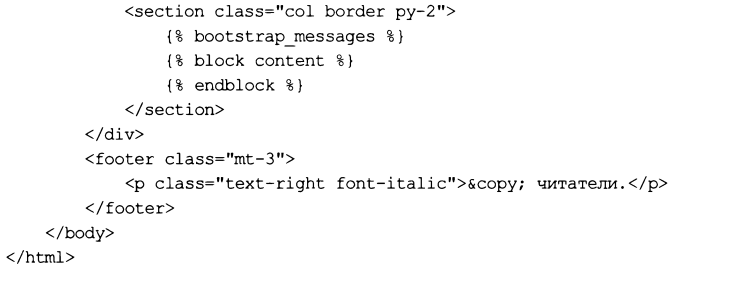


Добавим в список зарегистрированных в проекте приложение bootstrap4 — программное ядро библиотеки django-bootstrap4:



Приведен код базового шаблона. Сохраним его в файле templates\Layout\basic.html.





Код базового шаблона очень велик, сложен и включает много тегов и стилевых классов, формирующих разметку и оформление в стиле Bootstrap. Рассмотрим наиболее значимые фрагменты этого кода и выясним, что они делают:



Этот метатег, помещенный в секцию заголовка страницы (в парный тег <head>), необходим для того, чтобы Bootstrap правильно обработал страницу;



В теге <titie> мы создаем первый блок, назвав его title. С его помощью будет выводиться название страницы;



Привязываем к странице таблицы стилей Bootstrap;

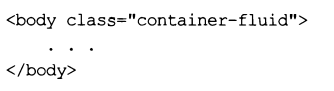


Привязываем таблицу стилей static\main\sty1e.css, которую создадим чуть позже.

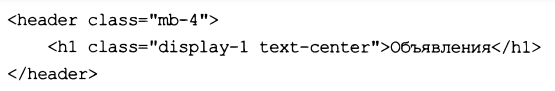
В ней мы запишем некоторые специфические для нашего сайта стили;



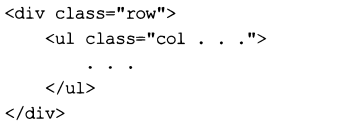
Привязываем файлы веб-сценариев с программным кодом Bootstrap вместе с сокращенной редакцией библиотеки jquery, без которой не заработает созданное нами раскрывающееся меню (AJAX и анимацию мы использовать не собираемся, так что полная редакция jquery нам не нужна);



К телу страницы (тегу <body>) привязываем стилевой класс container-fluid, как того требует Bootstrap;

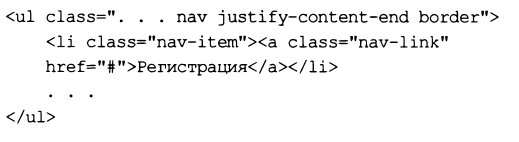


Стилевой класс mb-4, привязанный к элементу страницы, установит у него достаточно большой внешний отступ снизу. А стилевые классы display-1 И textcenter, привязанные к заголовку, предпишут веб-обозревателю вывести текст увеличенным шрифтом и выровнять его посередине:



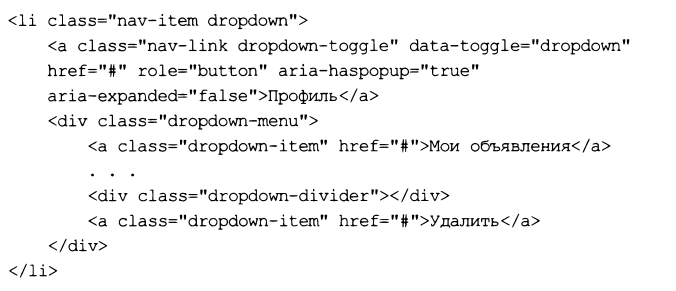
Bootstrap позволяет выполнять табличную верстку без участия собственно таблиц. Стилевой класс row, привязанный к элементу страницы, вынуждает его вести себя как строка таблицы, а стилевой класс col — как ячейка в этой строке.

В нашем случае мы создаем строку (сформированную блоком — тегом <div>) с единственной ячейкой — маркированным списком. Мы сделали так для того, чтобы убрать у создаваемого элемента страницы просветы слева и справа, которые выглядят очень некрасиво;



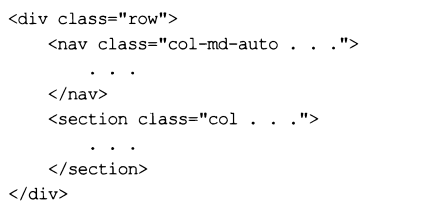
Упомянутый ранее маркированный список мы используем для создания горизонтальной полосы навигации. Для этого привяжем к нему стилевой класс nav. стилевой класс justify-content-end укажет выровнять пункты полосы навигации по правому краю, а стилевой класс border создаст рамку вокруг нее.

Пункты полосы навигации создаются так же, как и пункты списков, — тегами <и> — с привязанным стилевым классом nav-item. Внутри этих тегов должны находиться гиперссылки с привязанными стилевыми классами nav-link:



А данный код создает в полосе навигации пункт с раскрывающимся меню. Для этого внутрь тега <ii> помещается, помимо гиперссылки, еще и блок, который создаст само меню. Обратим внимание на стилевые классы, привязываемые к различным элементам, и дополнительные атрибуты, которые обязательно должны присутствовать в тегах.

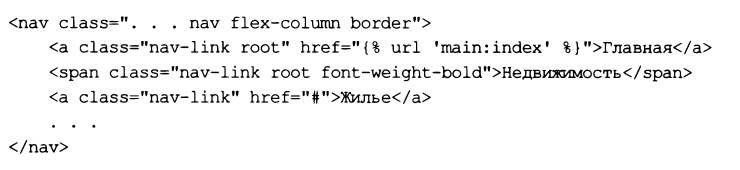
Пункты раскрывающегося меню формируются с помощью обычных гиперссылок СО стилевым классом dropdown-item. А ’’пустой” блок со стилевым классом Dropdown-divider создаст разделитель между пунктами;



Здесь мы снова применили табличную верстку, но на этот раз из двух ’’ячеек”:

Семантической панели навигации (тега <nav>) и семантической секции страницы (тега <section>).

Знакомый нам стилевой класс col при привязке к элементам –“ячейкам” создаст “Ячейки” одинаковой ширины. Если нужно сделать так, чтобы ширина какой-либо ячейки соответствовала ширине ее содержимого, то нужно привязать к создающему эту ячейку элементу страницы стилевой класс coi-md-auto. Мы привязали его к панели навигации;



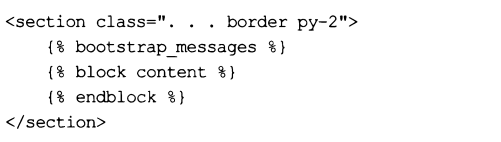
Мы привязали к семантической панели навигации стилевые классы nav и fiexcoiumn (чтобы превратить ее в вертикальную панель навигации в стиле Bootstrap), а также создающий рамку стилевой класс border.

Разные пункты панели навигации мы формируем тремя разными способами:

• пункты, ведущие на служебные страницы, — гиперссылками с двумя привязанными стилевыми классами: nav-iink, который нам уже знаком, и root, который мы запишем в таблице стилей static\main\style.css (он задаст увеличенный размер шрифта);

• пункты, обозначающие рубрики верхнего уровня (надрубрики),— тегами <span> со стилевыми классами nav-iink, root и font-weight-bold. Последний стилевой класс задаст полужирное начертание шрифта — так мы визуально выделим пункты этого типа. Обратим внимание, что такие пункты не являются гиперссылками;

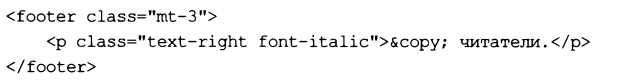
• пункты, ведущие на рубрики нижнего уровня (подрубрики), — гиперссылками со стилевым классом nav-link;



К семантической секции мы также привязали стилевые классы border и ру-2.

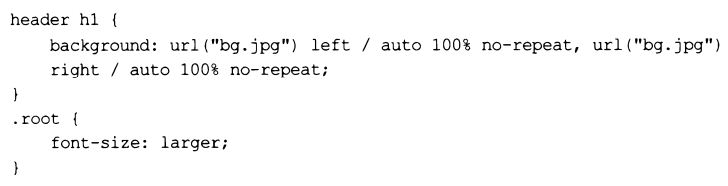
Первый нам уже знаком, а второй установит для тега небольшие внутренние отступы сверху и внизу, чтобы содержимое тега не примыкало к рамке вплотную.

В семантическую секцию мы поместили код, выводящий всплывающие сообщения, и блок content, в котором будет выводиться основное содержимое страниц;



’’Поддон” сайта мы создаем специализированным тегом <footer>. Стилевой класс mt-з укажет для него внешний отступ сверху средних размеров, чтобы отделить его от вышерасположенных элементов. Чтобы выровнять текст абзаца по правому краю и вывести его курсивом, мы привязали к тегу <р> стилевые классы Text-right И font-italic.

Приведен код таблицы стилей static\main\style.css. Создадим ее.

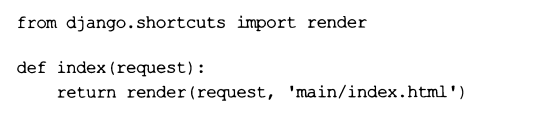


Первый стиль создаст у заголовка сайта фон в виде двух изображений доски объявлений, выведенных слева и справа. Подходящее изображение найдем в Интернете и также сохраним в папке static\main пакета приложения под именем bg.jpg.

**Главная веб-страница**

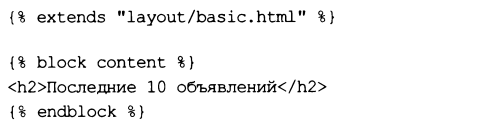
Главную страницу сделаем совсем минималистичной— только чтобы удостовериться, работает ли сайт.

Приведен код контроллера index (), выводящего главную страницу, который запишем в модуль views.py пакета приложения main. Этот контроллер мы реализовали в виде функции — так проще.



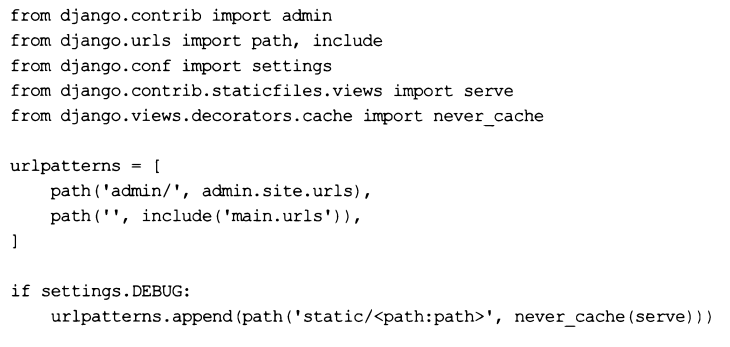
В пакете приложения main создадим папку templates\main, в которой будем сохранять шаблоны страниц.

Создадим шаблон главной страницы templates\main\index.html, записав в него код.



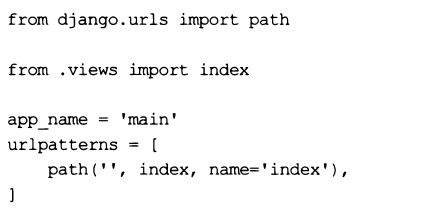
Теперь нужно написать маршруты. Заодно мы сделаем так, чтобы статические файлы сайта не кэшировались веб-обозревателем — ведь работа над таблицей стилей еще не закончена.

Начнем со списка маршрутов уровня проекта, хранящегося в модуле uris.py пакета конфигурации. Откроем и исправим его код.



Приложение main мы установили в качестве корневого.

Приступим к созданию списка маршрутов уровня приложения. Создадим в пакете приложения модуль uris.py и запишем в него код.



Сохраним все вновь созданные и исправленные файлы и запустим отладочный вебсервер Django с отключенной обработкой статических файлов:



Откроем веб-обозреватель, выполним переход по интернет-адресу bttp://localhost:

8000/ и попадем на главную страницу нашего сайта

**Вспомогательные веб-страницы**

Сразу же создадим первую из вспомогательных веб-страниц — со сведениями о сайте и правах его разработчиков. Сделать это можно двумя способами.

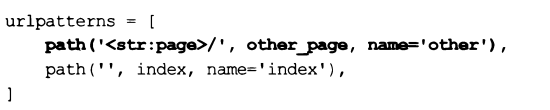
Первый способ, самый очевидный, заключается в том, что для каждой страницы пишется отдельный контроллер и отдельный маршрут. Этот способ весьма трудоемок, поскольку придется писать несколько контроллеров с практически одинаковым кодом, и подходит лишь для тех случаев, когда страницы должны выводить какие-либо данные, извлекаемые из базы или формируемые программно.

Второй способ — вывод всех страниц с применением одного контроллера и, соответственно, одного маршрута. Какой-либо идентификатор страницы, предназначенной к выводу на экран, передается контроллеру с URL-параметром. Трудоемкость работы в таком случае существенно снижается, поскольку нужно написать всего один контроллер.

Реализуем вывод вспомогательных страниц вторым способом. В качестве идентификатора страницы используем имя формирующего ее шаблона без пути и без расширения — так проще.

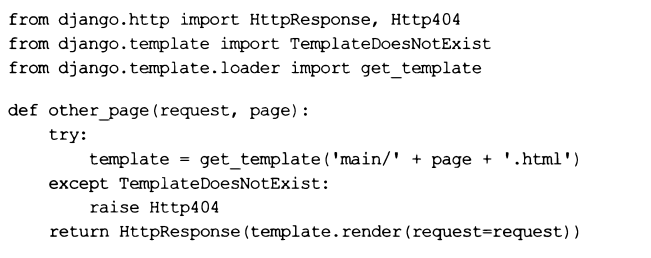
В список маршрутов уровня приложения, что хранится в модуле urls.py пакета приложения, добавим такой код:





Имя шаблона выводимой страницы передаем через URL-параметр раде. А контроллер, выводящий вспомогательные страницы, назовем other\_page и реализуем в виде функции. Вообще, контроллеры-функции — идеальный инструмент для написания чего-либо нестандартного, специфического.

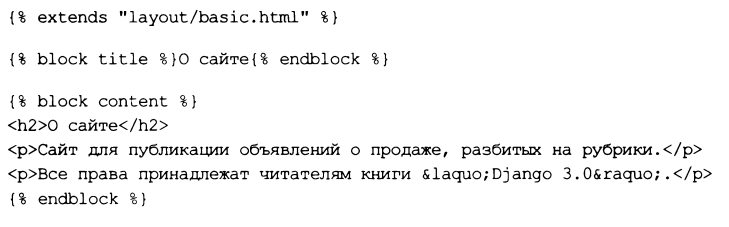
В модуль views.py пакета приложения, где хранится код контроллеров, добавим код контроллера-функции other page (), приведен.



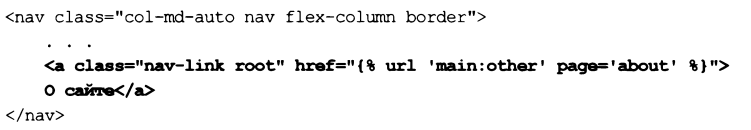
Имя выводимой страницы получаем из параметра раде, добавляем к нему путь и расширение, получив тем самым полный путь к нужному шаблону, и пытаемся загрузить его вызовом функции get tempiate (). Если загрузка прошла успешно, то формируем на основе этого шаблона страницу.

Если же шаблон загрузить не удалось, то функция get tempiate () возбудит исключение templatedoesnotexist. Мы перехватываем это исключение и возбуждаем другое исключение — Http404, которое приведет к отправке страницы с сообщением об ошибке 404 (запрошенная страница не существует).

Странице со сведениями о сайте и правах его разработчиков дадим имя about. Код формирующего ее шаблона templates\main\about.html приведен



Откроем базовый шаблон templates\layout\basic.html и добавим в левую панель навигации такой код:



Сохраним все исправленные и вновь созданные файлы, обновим открытую в вебобозревателе главную страницу, перейдем по гиперссылке оО сайте — и увидим только что созданную страницу.

1. Работа с пользователями и ра зграничение доступа

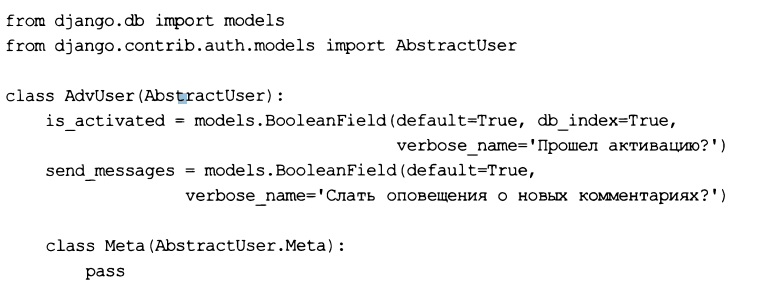
Теперь займемся инструментами для работы с пользователями и разграничения доступа. Мы создадим страницы для входа и выхода, регистрации, активации, страницы пользовательского профиля, для смены данных о пользователе, смены его пароля и удаления профиля.

**Модель пользователя**

Стандартная модель пользователя user, предлагаемая стандартным же приложением django.contrib.auth, не подходит, поскольку нам нужно хранить дополнительные данные о пользователе. Поэтому создадим свою собственную модель, сделав ее производной от стандартной абстрактной модели Abstractuser, объявленной в модуле dj ango.contrib.auth.

Наша модель будет носить название advuser.

Код, объявляющий эту модель, приведен. Запишем его в модуль Models.ру пакета приложения.



Сразу же укажем ее как модель пользователя, используемую подсистемой разграничения доступа Django. Для этого откроем модуль settings.py пакета конфигурации и добавим в него строку:



Остановим отладочный веб-сервер Django и зададим в командной строке команду сначала на создание миграций:



А потом — на их выполнение:

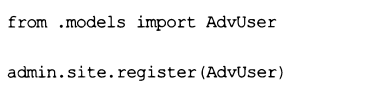


Как только миграции будут выполнены, создадим суперпользователя, подав команду:



Введем выбранные нами имя, адрес электронной почты и пароль создаваемого пользователя.

Напоследок откроем модуль admin.ру пакета приложения, в котором объявляются классы-редакторы и регистрируются модели в административном сайте. Зарегистрируем нашу модель пользователя, добавив в этот модуль код:



Запустим отладочный веб-сервер, опять же отключив обработку статических файлов. Откроем административный веб-сайт, набрав интернет-адрес http://Localhost:8000/admin/, и попробуем выполнить вход от имени только что созданного суперпользователя.

Основные веб-страницы:

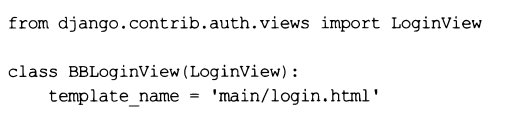
Входа, профиля и выхода

Далее ради простоты мы будем, по возможности, следовать установленным Django соглашениям.

**Веб-страница входа**

Для реализации входа мы создадим подкласс контроллера-класса Loginview, в котором запишем все необходимые для работы контроллера параметры.

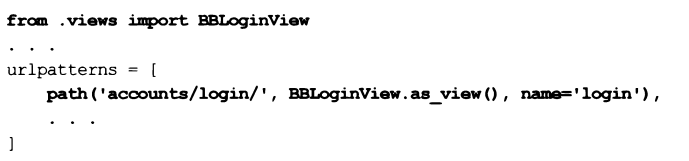
Код контроллера-класса, выполняющего вход и носящего имя bbloginview, приведен. Добавим его в модуль views.py пакета приложения.



В классе мы указали лишь путь к файлу шаблона, занеся его в атрибут templatename. Остальные параметры сохранят значения по умолчанию, т. к. мы собираемся следовать принятым в Django соглашениям.

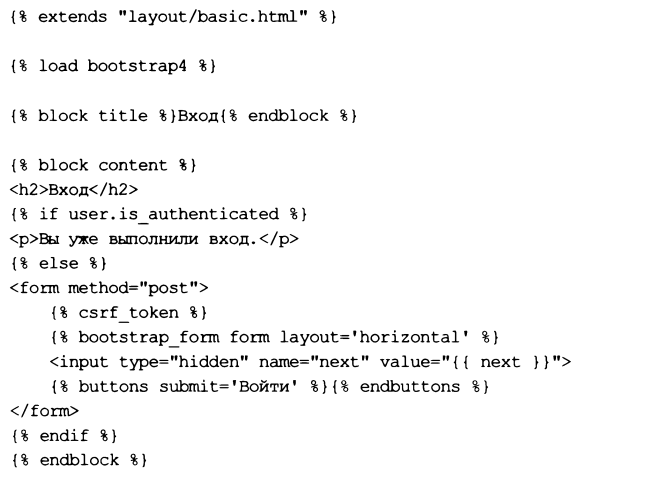
Шаблон страницы входа login.html мы поместили в папку templates\main — туда же, где находятся все остальные шаблоны. Поскольку наш сайт включает относительно немного страниц, будем хранить их в одной папке, чтобы упростить сопровождение сайта.

Запишем новый маршрут, указывающий на контроллер bbloginview, в списке уровня приложения. Откроем модуль urls.py пакета приложения и добавим в него код:



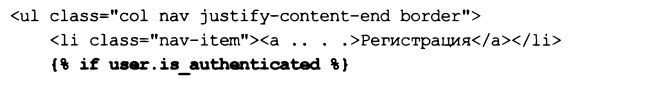
В маршруте мы указали шаблонный путь accounts/login/. По умолчанию именно по нему Django выполняет перенаправление при попытке гостя получить доступ к закрытой от него странице.

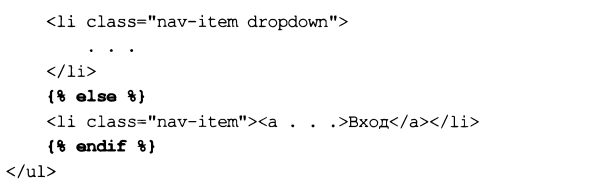
Напишем шаблон страницы входа templates\main\login.html. Его код приведен.



Поля ввода имени и пароля невелики, так что мы вывели форму в ’’горизонтальной” разметке Bootstrap, когда надпись и относящийся к ней элемент управления располагаются по горизонтали.

Напоследок внесем правки в базовый шаблон templates\layout\basic.html. Найдем в нем фрагмент кода, создающего пункт Вход и пункт с раскрывающимся меню Профиль горизонтальной полосы навигации, и исправим его следующим образом:





В результате пункт Вход будет выводиться только гостям, а пункт Профиль — только пользователям, выполнившим вход.

Запишем в тег <а>, создающий гиперссылку Вход, интернет-адрес страницы входа:

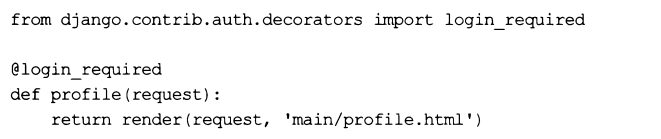


Сохраним все новые и исправленные файлы, обновим открытую в веб-обозревателе главную страницу и щелкнем на гиперссылке Вход. Если мы все сделали без ошибок, то сразу же попадем на страницу входа.

Но пока не будем выполнять вход, иначе возникнет ошибка. Сначала сделаем страницы профиля и выхода.

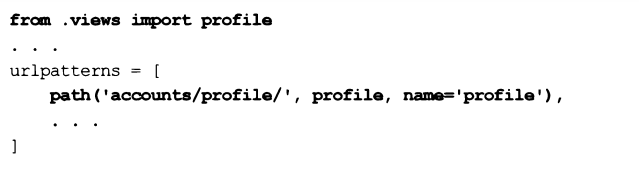
**Веб-страница пользовательского профиля**

Контроллер, выводящий страницу пользовательского профиля, реализуем в виде функции и назовем profile(). Его код чрезвычайно прост— см. Ниже. Не забываем, что все контроллеры объявляются в модуле views.py пакета приложения.



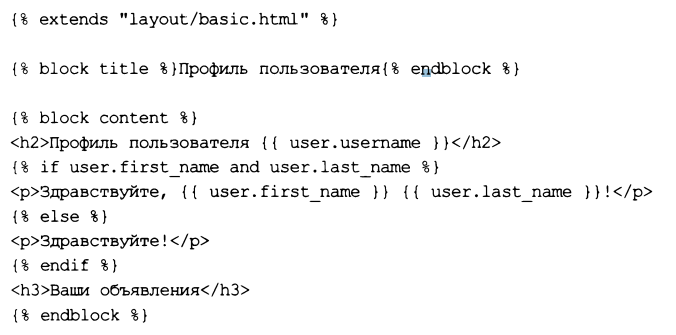
Поскольку страница пользовательского профиля должна быть доступна только зарегистрированным пользователям, выполнившим вход на сайт, мы пометили контроллер-функцию profile () декоратором login\_required ().

Добавим в список маршрутов уровня приложения (не забываем, что он хранится в модуле uris.py пакета приложения) маршрут на контроллер profile ():



Мы указали в этом маршруте шаблонный путь aceaunts/profile/— по нему Django по умолчанию выполняет перенаправление после успешного входа.

Код шаблона templates\main\profile.html, формирующего страницу профиля, приведен:



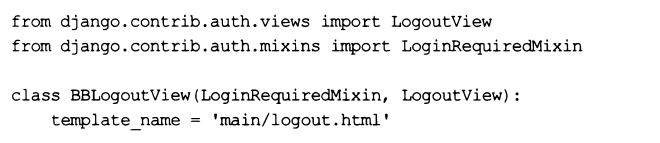
Если пользователь при регистрации написал свои имя и фамилию, то на странице будет выведено персонализированное приветствие. Если же пользователь не ввел эти данные, появится простое приветствие.

В дальнейшем на странице пользовательского профиля будет выводиться список объявлений, оставленных текущим пользователем. Но объявлений у нас пока что нет (более того, сама функциональность по их написанию еще не создавалась), так что более на этой странице ничего не выводится.

В шаблоне templates/layout/basic.html найдем тег <а>, выводящий гиперссылку Мои объявления, и вставим в него интернет-адрес страницы профиля:

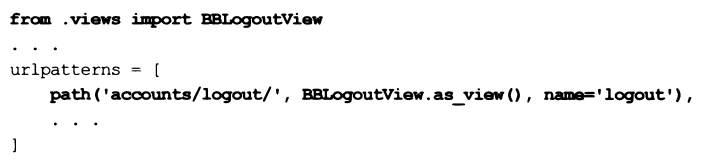


Контроллер выхода реализуем в виде класса bblogoutview, производного от класса Logoutview. Его код приведен:

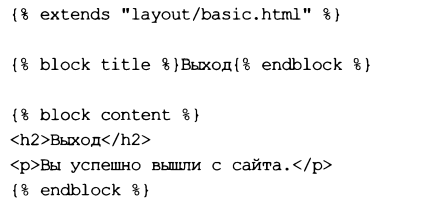


Страница выхода должна быть доступна только зарегистрированным пользователям, выполнившим вход. Поэтому мы добавили в число суперклассов контроллера-класса bblogoutview примесь loginrequiredmixin.

В списке маршрутов уровня приложения запишем маршрут, ведущий на этот контроллер:



Напишем шаблон страницы выхода templates\main\logout.html, очень простой код которого приведен:



В шаблоне templates/layout/basic.html найдем тег <а>, выводящий гиперссылку Выход, и поместим в него интернет-адрес страницы выхода:



Сохраним все файлы, подождем, пока отладочный веб-сервер не перезапустится, и обновим страницу входа, открытую в веб-обозревателе. Занесем в форму имя и пароль созданного ранее суперпользователя и выполним вход. Посмотрим на страницу профиля и выйдем с сайта.

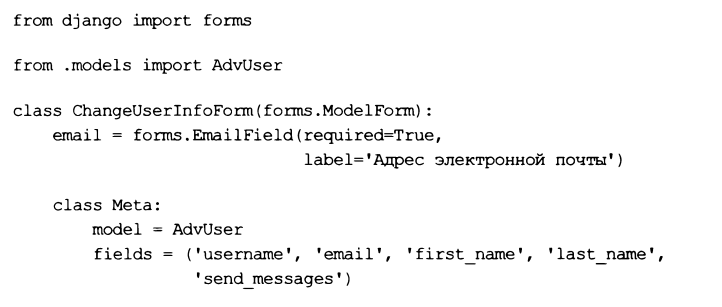
**Веб-страница правки основных сведений**

На этой странице пользователь сможет исправить свои имя (логин), адрес электронной почты, реальные имя, фамилию и признак, хочет ли он получать по электронной почте оповещения о появлении новых комментариев к его объявлениям.

Адрес электронной почты будет обязательным к заполнению.

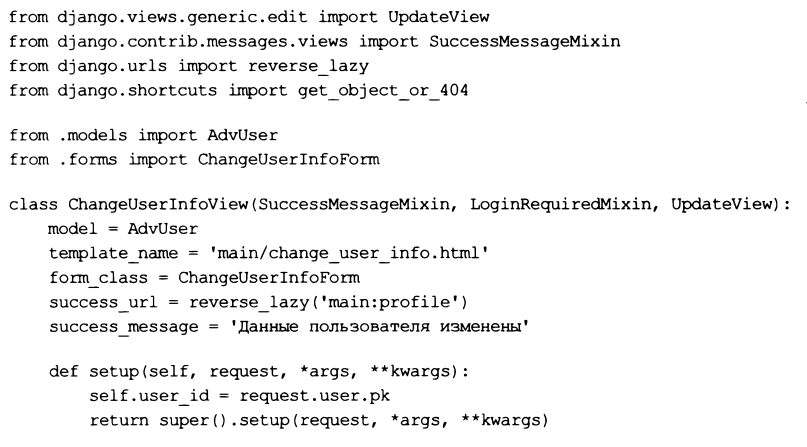
Сначала объявим форму changeuserlnfoform, связанную с моделью advuser и предназначенную для ввода основных данных. Код формы приведен.

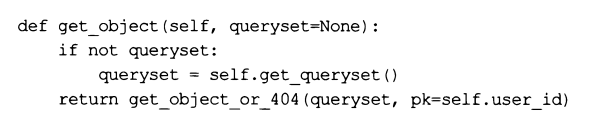
Его мы запишем во вновь созданный модуль forms.py пакета приложения.



Так как мы хотим сделать поле email модели advuser обязательным для заполнения, то выполним полное объявление поля email формы. А поскольку параметры остальных полей формы: username, first\_name, last\_name И send\_messages — у нас не меняются, в их отношении мы применим быстрое объявление.

Контроллер страницы основных данных должен выполнять правку записи модели, так что мы можем написать его на базе высокоуровневого класса updateview. Готовый код контроллера-класса changeuserinfoview приведен.





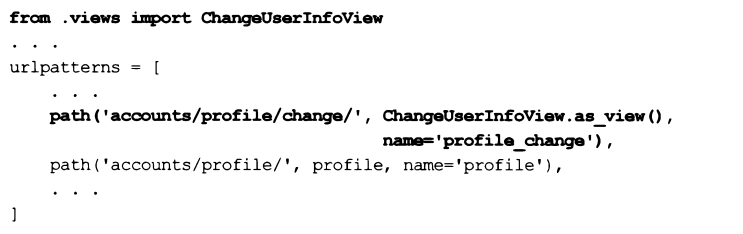
В процессе работы этот контроллер должен извлечь из модели advuser запись, представляющую текущего пользователя, для чего ему нужно предварительно получить ключ текущего пользователя. Получить его можно из объекта текущего пользователя, хранящегося в атрибуте user объекта запроса.

Вероятно, наилучшее место для получения ключа текущего пользователя — метод Setupo, наследуемый всеми контроллерами-классами от их общего суперкласса View. Этот метод выполняется в самом начале исполнения контроллера-класса и получает объект запроса в качестве одного из параметров. В переопределенном методе setup () мы извлечем ключ пользователя и сохраним его в атрибуте user id.

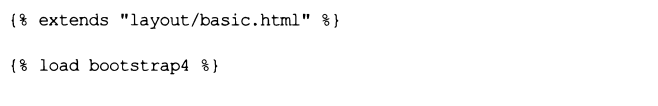
Извлечение исправляемой записи выполняем в методе get objectо, который контроллер-класс унаследовал от примеси singieobjectmixin. В переопределенном методе сначала учитываем тот момент, что набор записей, из которого следует извлечь искомую запись, может быть передан методу с параметром queryset, а может быть и не передан — в этом случае набор записей следует получить вызовом метода get queryset (). После этого непосредственно ищем запись, представляющую текущего пользователя.

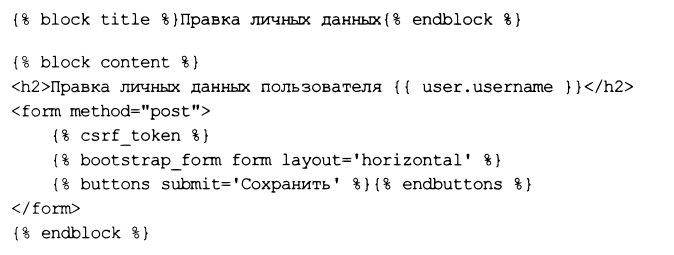
В качестве одного из суперклассов этого контроллера-класса мы указали примесь Loginrequiredmixin, запрещающую доступ к контроллеру гостям, и примесь Successmessagemixin, которая применяется для вывода всплывающих сообщений об успешном выполнении операции. Зря мы, что ли, вставили в шаблон templates\Layout\basic.html код, выводящий всплывающие сообщения.

В список маршрутов уровня приложения добавим соответствующий маршрут:



Код шаблона templates\main\change\_user\_info.html, создающего страницу для правки основных данных, приведен. Ничего особо сложного в нем нет.





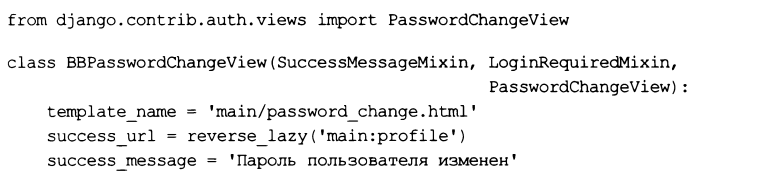
В шаблоне templates\layout\basic.html отыщем тег <а>, выводящий гиперссылку Изменить личные данные, и вставим в него интернет-адрес страницы правки основных данных:



Сохраним файлы, перейдем на страницу пользовательского профиля и проверим только что созданную страницу правки основных данных в работе.

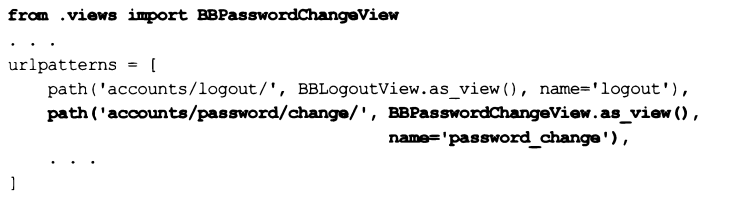
**Веб-страница правки пароля**

Контроллер-класс bbpasswordchangeview, выводящий эту страницу, сделаем производным от класса passwordchangeview, который реализует смену пароля. Код нашего контроллера-класса приведен.

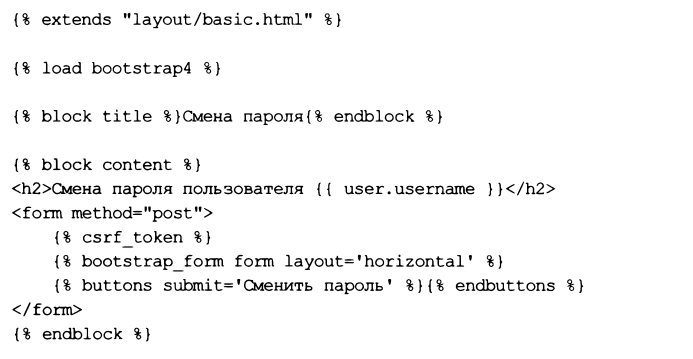


После успешной смены пароля выполняем перенаправление на страницу профиля пользователя с выводом соответствующего всплывающего сообщения.

Добавим в список маршрутов уровня приложения маршрут, который укажет на новый контроллер:



Приведен код шаблона страницы для смены пароля templates\Main\password\_change.html.



Осталось в коде шаблона templates\layout\basic.html найти код, создающий гиперссылку Изменить пароль, и поместить в нее правильный интернет-адрес:



Теперь можно сохранить все файлы и проверить, работает ли страница смены пароля.

**Веб-страницы регистрации нового пользователя**

Мы напишем форму для ввода сведений о новом пользователе, контроллеры и шаблоны для страниц непосредственно регистрации и уведомления об успешной регистрации.

Для отправки письма о необходимости активации мы объявим свой сигнал. Называться он будет user\_registered и получит в качестве единственного параметра Instance объект вновь созданного пользователя.

Сигнал мы объявим в модуле apps.py пакета приложения. Этот модуль выполняется непосредственно при инициализации приложения и, таким образом, является идеальным местом для записи кода, объявляющего сигналы.

**Форма для занесения сведений о новом пользователе**

Код класса формы registeruserform, приведенный, запишем в модуль forms.py пакета приложения.





Здесь мы также комбинируем быстрое и полное объявление полей. Полное объявление используем для создания полей электронной почты (поскольку хотим сделать его обязательным для заполнения) и обоих полей для занесения пароля. Согласно общепринятой практике, отведем для занесения пароля два поля, в которые нужно ввести один и тот же пароль.

В качестве дополнительного поясняющего текста у первого поля пароля указываем объединенный текст с требованиями к вводимому паролю, предоставленный всеми доступными в системе валидаторами, — там новый пользователь сразу поймет, какие требования предъявляются к паролю.

В методе ciean\_passwordi () выполняем валидацию пароля, введенного в первое поле, с применением доступных в системе валидаторов пароля. Проверять таким же образом пароль из второго поля нет нужды — если пароль из первого поля некорректен, не имеет значения, является ли корректным пароль из второго поля.

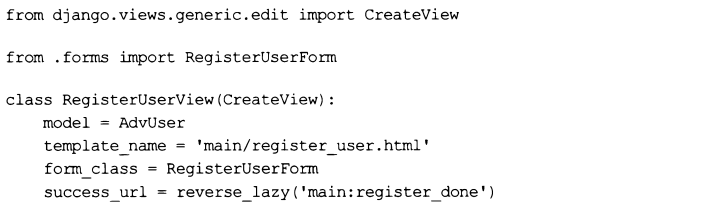
В переопределенном методе clean () проверяем, совпадают ли оба введенных пароля. Эта проверка будет проведена после валидации пароля из первого поля.

В переопределенном методе save () при сохранении нового пользователя заносим значения False в поля is active (признак, является ли пользователь активным) и is activated (признак, выполнил ли пользователь процедуру активации), тем самым сообщая фреймворку, что этот пользователь еще не может выполнять вход на сайт. Далее сохраняем в записи закодированный пароль и отправляем сигнал User registered, чтобы отослать пользователю письмо с требованием активации.

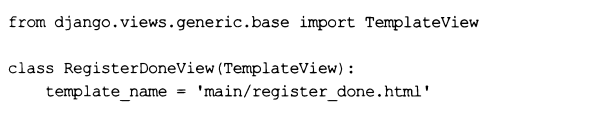
**Средства для регистрации пользователя**

Эти средства будут включать две страницы: одна выведет веб-форму для ввода данных о регистрирующемся пользователе, вторая сообщит об успешной регистрации и отправке письма с требованием активации.

Контроллер-класс, регистрирующий пользователя, мы назовем Registeruserview и сделаем производным от класса createview. Его код приведен.



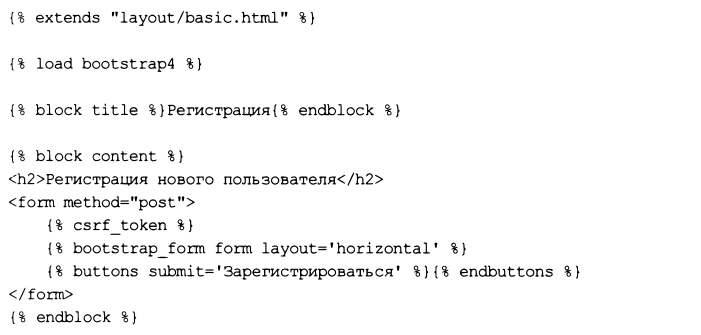
Контроллер, который выведет сообщение об успешной регистрации, будет называться registerdoneview и, в силу его исключительной простоты, станет производным от класса Templateview. Его код можно увидеть.

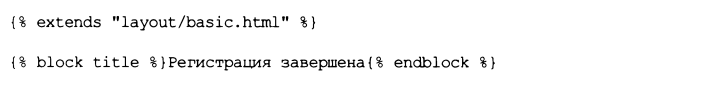


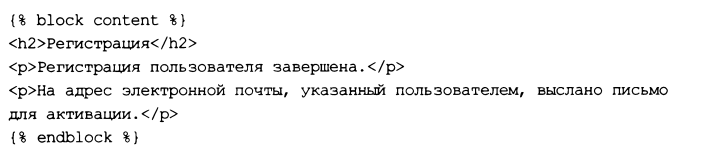
В список маршрутов уровня приложения добавим два маршрута, ведущие на только что написанные нами контроллеры:



Код шаблонов templates\main\register\_user.html и templates\main\register\_done.html, которые формируют страницы регистрации и уведомления о ее успешном завершении, приведен:







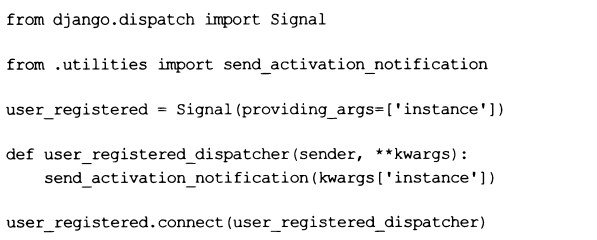
В шаблоне templates\layout\basic.html найдем фрагмент, создающий гиперссылку регистрация, и вставим в нее интернет-адрес страницы регистрации:



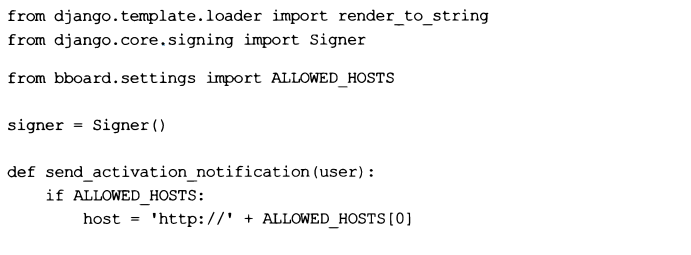
**Средства для отправки писем с требованиями активации**

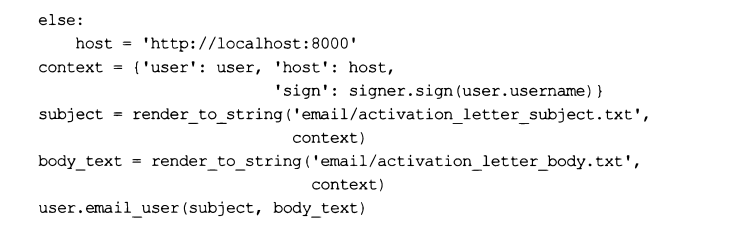
Непосредственную рассылку электронных писем будет выполнять функция Send\_activation\_notification (), которую МЫ объявим чуть позже, ВО ВНОВЬ созданном модуле utilities.py. Эта функция еще пригодится нам, когда мы будем писать редактор ДЛЯ модели advuser.

Откроем модуль apps.py пакета приложения и запишем в него код, который объявит сигнал user registered и привяжет к нему обработчик:



Создадим в пакете приложения модуль utilities.py. Занесем в него объявление функции send\_activation\_notification ()





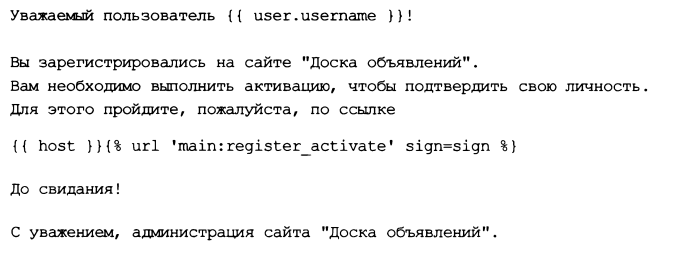
Чтобы сформировать интернет-адрес, ведущий на страницу подтверждения активации, понадобится, во-первых, домен, на котором находится наш сайт, а во-вторых, некоторое значение, уникально идентифицирующее только что зарегистрированного пользователя и при этом устойчивое к попыткам его подделать.

Домен мы можем извлечь из списка разрешенных доменов, который записан в параметре allowed\_hosts настроек проекта. Выберем самый первый домен, присутствующий в списке. Если же список доменов пуст, мы задействуем интернет-адрес, используемый отладочным веб-сервером Django.

В качестве уникального и стойкого к подделке идентификатора пользователя применяем его имя, защищенное цифровой подписью. Создание цифровой подписи выполняем посредством класса signer.

Текст темы и тела письма формируем с применением шаблонов templates\email\Activation\_letter\_subject.txt и templates\email\activation\_letter\_body.txt соответственно. Их код приведен:





**Веб-страницы активации пользователя**

Чтобы реализовать активацию нового пользователя, мы напишем один контроллер и целых три шаблона. Они создадут страницы с сообщением об успешной активации, о том, что активация была выполнена ранее, и о том, что цифровая подпись у идентификатора пользователя, полученного в составе интернет-адреса, скомпрометирована.

Контроллер мы реализуем в виде функции user\_activate (). Ее код приведен:



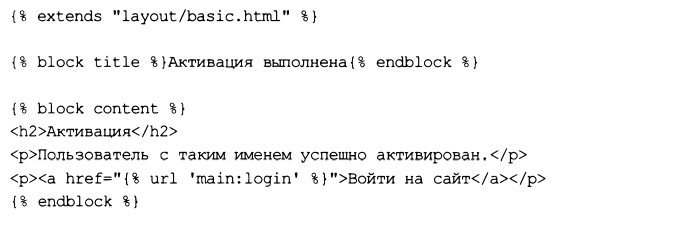
Подписанный идентификатор пользователя, передаваемый в составе интернет-адреса, получаем с параметром sign. Далее извлекаем из него имя пользователя, ищем пользователя с этим именем, делаем его активным, присвоив значения True полям is active и is activated модели, и выводим страницу с сообщением об успешной активации. Если цифровая подпись оказалась скомпрометированной, выводим страницу с сообщением о неуспехе активации, а если пользователь был активирован ранее (поле is activated уже хранит значение True) — страницу с сообщением, что активация уже произошла.

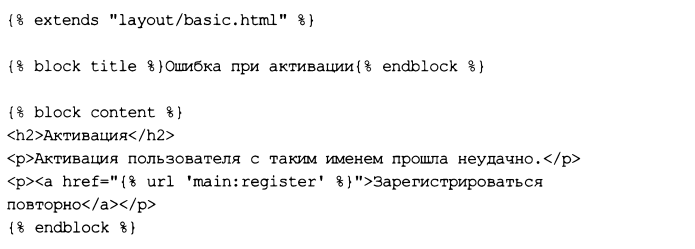
Для обработки подписанного значения используем экземпляр класса signer, созданный в модуле utilities.py и хранящийся в переменной signer. Так мы сэкономим оперативную память.

В списке маршрутов уровня приложения запишем маршрут, ведущий на контроллер user\_activate():

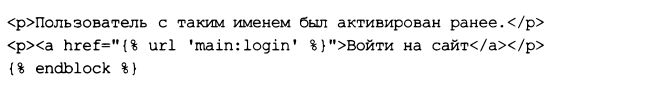


Код шаблонов templates\main\activation\_done.html, templates\main\bad\_signature.html и templates\Main\user\_is\_activated.html страниц с сообщениями соответственно об успешной, неуспешной и уже выполненной активации приведен:









Чтобы протестировать отправку электронных писем, воспользуемся отладочным SMTP-сервером. Добавим в модуль settings.py пакета конфигурации выражение, задающее TCP-порт № 1025, который используется этим сервером по умолчанию:



Запустим еще один экземпляр командной строки и отдадим команду, запускающую отладочный SMTP-сервер с использованием TCP-порта № 1025:



Сохраним файлы с исходным кодом, перейдем на страницу регистрации, введем сведения о новом пользователе и дождемся письма с требованием активации.

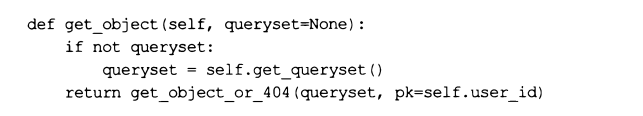
Перейдем по находящемуся в этом письме интернет-адресу, удостоверимся, что активация прошла успешно, и попытаемся выполнить вход от имени только что созданного пользователя.

Добавим таким же образом еще двух или трех пользователей— они пригодятся нам для реализации удаления пользователей.

**Веб-страница удаления пользователя**

Контроллер-класс Deleteuserview, удаляющий текущего пользователя, сделаем производным от класса Deleteview. Его код приведен:





Здесь мы использовали те же программные приемы, что и в контроллере Changeuserinfoview. В переопределенном методе setup о сохранили ключ текущего пользователя, а в переопределенном методе get object о отыскали по этому ключу пользователя, подлежащего удалению.

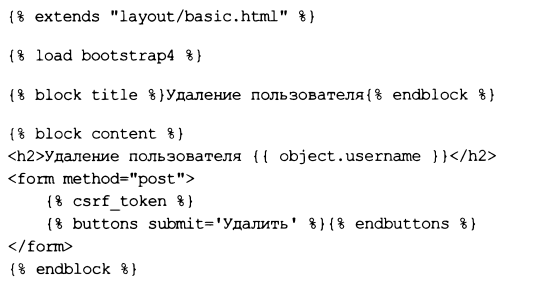
Перед удалением текущего пользователя необходимо выполнить выход, что мы и сделали в переопределенном методе post о. В том же методе post о мы создали всплывающее сообщение об успешном удалении пользователя.

В списке маршрутов уровня приложения запишем код, который добавит новый маршрут:



Напишем шаблон templates\main\delete\_user.html страницы для удаления пользователя.

Его код приведен:



Осталось записать в шаблоне templates\layout\basic.html интернет-адрес, ведущий на страницу удаления пользователя:

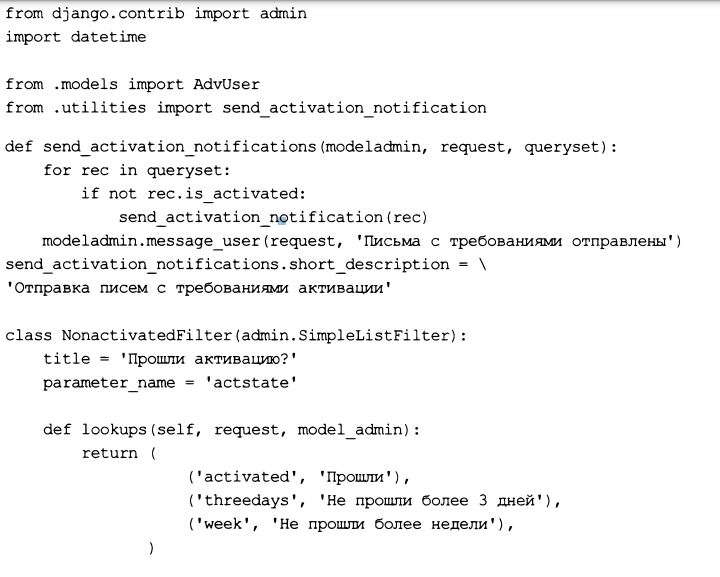


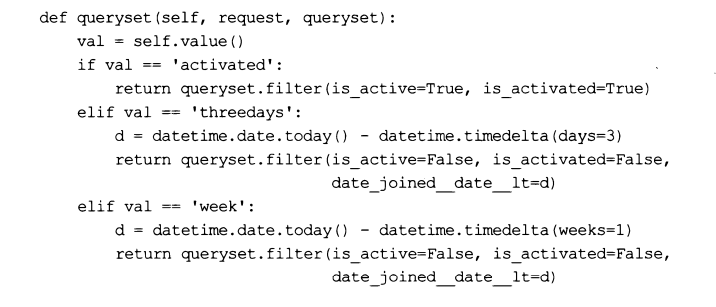
Для проверки попробуем войти на сайт от имени одного из ранее созданных пользователей (только не суперпользователя— иначе придется создавать его заново) и выполнить его удаление.

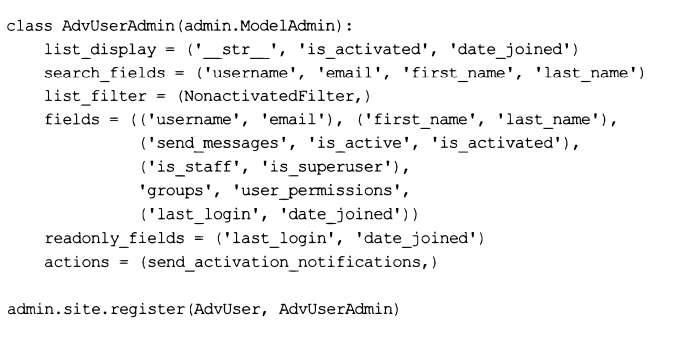
**Инструменты для администрирования пользователей**

Напоследок напишем редактор, посредством которого администрация сайта будет работать с зарегистрированными пользователями. В него добавим возможность фильтрации пользователей по именам, адресам электронной почты, настоящим именам и фамилиям. Также реализуем вывод пользователей, уже выполнивших активацию, не выполнивших ее в течение трех дней и недели, и действие по отправке выбранным пользователям писем с требованиями пройти активацию.

Полный код класса редактора advuseradmin, вспомогательного класса и функции приведен. Этот код следует занести в модуль admin.ру пакета приложения, заменив им имеющийся там код.







В списке записей указываем выводить строковое представление записи (имя пользователя — как реализовано в модели Abstractuser, от которой наследует наша модель), поле признака, выполнил ли пользователь активацию, временную отметку его регистрации. Также разрешаем выполнять фильтрацию по полям имени, адреса электронной почты, настоящих имени и фамилии.

Для выполнения фильтрации пользователей, выполнивших активацию, не выполнивших ее в течение трех дней и недели, используем класс nonactivatedfilter.

Обратим внимание на код, непосредственно фильтрующий пользователей по значению даты их регистрации.

Мы явно указываем список полей, которые должны выводиться в формах для правки пользователей, чтобы выстроить их в удобном для работы порядке. Поля даты регистрации пользователя и последнего его входа на сайт делаем доступными только для чтения.

Наконец, регистрируем действие, которое разошлет пользователям письма с предписаниями выполнить активацию. Это действие реализовано функцией send\_Activation\_notifications о. В ней мы перебираем всех выбранных пользователей И ДЛЯ каждого, КТО Не ВЫПОЛНИЛ активацию, вызываем функцию send\_activation\_Notification о, объявленную ранее в модуле utilities.py и непосредственно производящую отправку писем.

На этом все. Сохраним весь исправленный код и проверим написанный нами редактор в действии.

1. Рубрики

Мы реализуем двухуровневую структуру рубрик: более общие рубрики верхнего уровня (надрубрики) и вложенные в них рубрики нижнего уровня (подрубрики). Список рубрик будет выводиться в вертикальной панели навигации на каждой странице сайта.

**Модели рубрик**

Напишем базовую модель, в которой будут храниться и надрубрики, и подрубрики, и две производные от нее прокси-модели: для надрубрик и подрубрик.

Базовой модели, в которой будут храниться и надрубрики, и подрубрики, мы дадим имя Rubric.

Поле order будет хранить целое число, обозначающее порядок следования рубрик друг за другом: при выводе рубрики сначала будут сортироваться по возрастанию начения порядка, а уже потом — по их названиям.

Поле super rubric будет хранить надрубрику, к которой относится текущая подрубрика. Оно будет иметь следующие важные особенности:

□ связь, создаваемая этим полем, должна устанавливаться с моделью надрубрик, которую мы объявим чуть позже. Условимся называть эту модель superrubric;

□ это поле будет заполняться только в том случае, если запись хранит подрубрику.

Если запись хранит надрубрику, то поле заполнять не нужно (собственно, отсутствие значения в этом поле является признаком надрубрики — ведь надрубрика в принципе не может ссылаться на надрубрику). Поэтому данное поле сделано необязательным для заполнения;

□ нужно обязательно запретить каскадное удаление записей, чтобы пользователь по ошибке не удалил надрубрику вместе со всеми подрубриками.

Код класса модели Rubric приведен. Не забываем, что код всех моделей заносится в модуль models.ру пакета приложения.



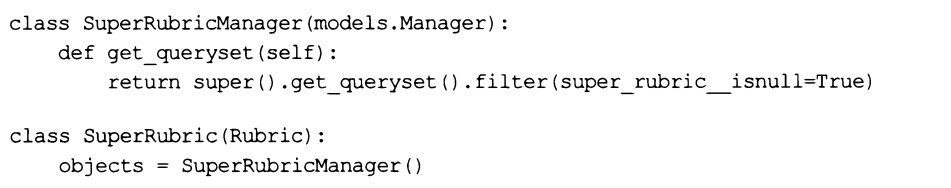
Мы не задаем никаких параметров самой модели— поскольку пользователи не будут работать с ней непосредственно, здесь это совершенно излишне.

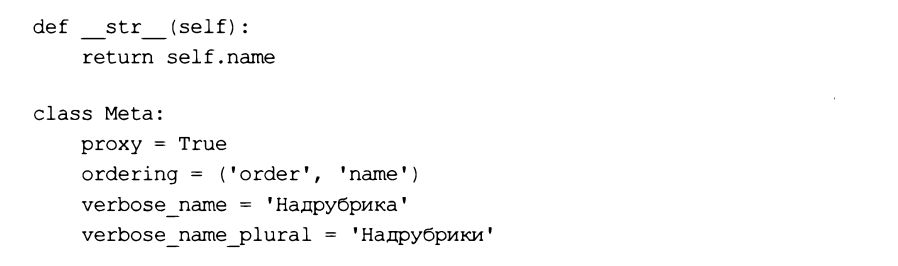
**Модель надрубрик**

Для работы с надрубриками объявим прокси-модель superrubric, производную от Rubric (прокси-модель позволяет менять лишь функциональность модели, но не набор объявленных в ней полей, однако нам и надо изменить лишь функциональность модели). Она будет обрабатывать только надрубрики.

Чтобы изменить состав обрабатываемых моделью записей, нужно задать для нее свой диспетчер записей, который и укажет необходимые условия фильтрации.

Код обоих классов: И модели superrubric, И диспетчера записей superrubricmanager — приведен:





Условия фильтрации записей указываем в переопределенном методе get\_Queryset о Класса Диспетчера записей superrubricmanager. Он станет выбирать только записи с пустым полем super rubric, т. е. надрубрики.

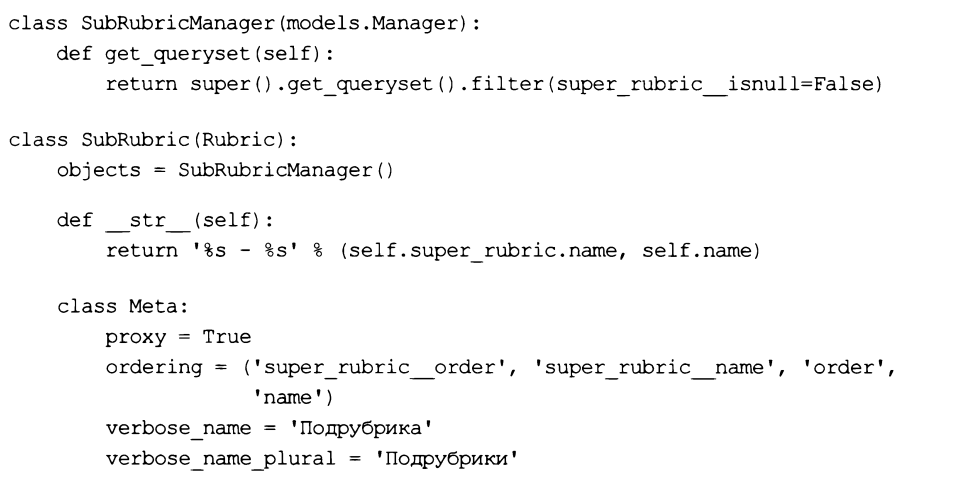
В самом классе модели superrubric задаем диспетчер записей superrubricmanager в качестве основного. И не забываем объявить метод\_\_str\_\_ о, который станет генерировать строковое представление надрубрики — ее название.

Как и условились ранее, указываем сортировку записей сначала по возрастанию значения порядка, а потом — по названию.

**Модель подрубрик**

Модель подрубрик subrubric мы создадим таким же образом, как и модель надрубрик. Только теперь диспетчер записей, который мы создадим для нее, будет выбирать лишь подрубрики.

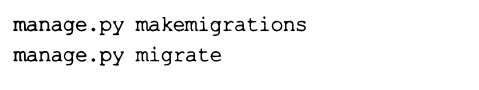
Код классов модели subrubric и диспетчера записей subrubricmanager приведен:



Диспетчер записей subrubricmanager будет отбирать лишь записи с непустым полем Super rubric (т. Е. Подрубрики). Строковое представление, создаваемое моделью, будет выполнено В формате сназвание надрубрики> - <название подрубрики>.

А сортировку записей укажем по порядку надрубрики, названию надрубрики, порядку подрубрики и названию подрубрики.

Объявив все необходимые классы, остановим отладочный веб-сервер (если он все еще работает), создадим и выполним миграции:

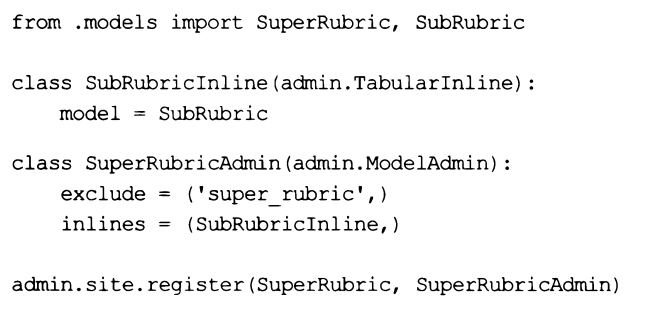


**Инструменты для администрирования рубрик**

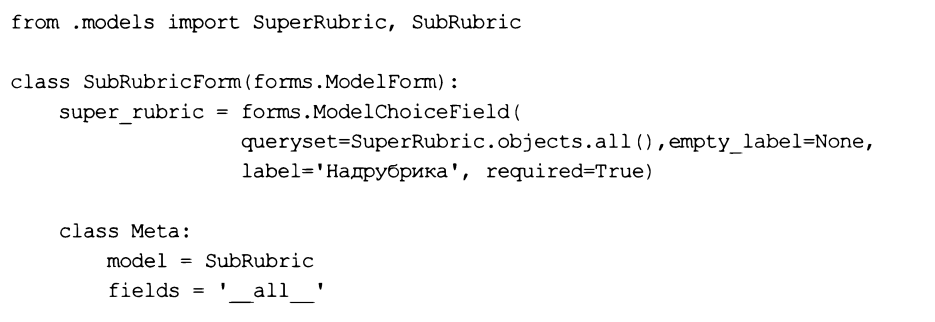
Вся работа с надрубриками и подрубриками будет проводиться средствами административного сайта.

Для надрубрик мы создадим встроенный редактор, чтобы пользователь, добавив новую надрубрику, смог сразу же заполнить ее подрубриками. Из формы для ввода и правки надрубрик мы исключим поле надрубрики (super rubric), поскольку оно там совершенно не нужно и, более того, собьет пользователя с толку.

Приведен код классов редактора superrubricadmin и встроенного редактора subrubricinline. Не забываем, что код редакторов, равно как и код, регистрирующий модели и редакторы в подсистеме административного сайта, должен записываться в модуль admin.ру пакета приложения.

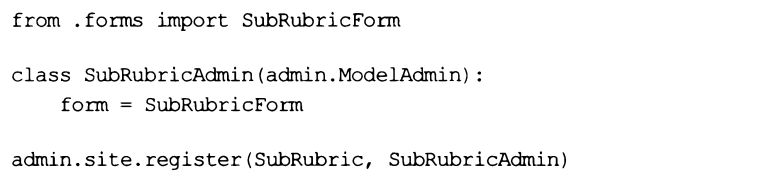


У подрубрик сделаем поле надрубрики (super rubric) обязательным для заполнения. Для этого мы объявим форму subrubricform, записав ее код, приведен, в модуле forms.py пакета приложения.



Мы убрали у раскрывающегося списка, с помощью которого пользователь будет выбирать подрубрику, ’’пустой” пункт, присвоив параметру empty iabei конструктора класса поля modelchoicefield значение None. Так мы дополнительно дадим понять, что в это поле обязательно должно быть занесено значение.

Приведен КОД, объявляющий класс редактора subrubricadmin.



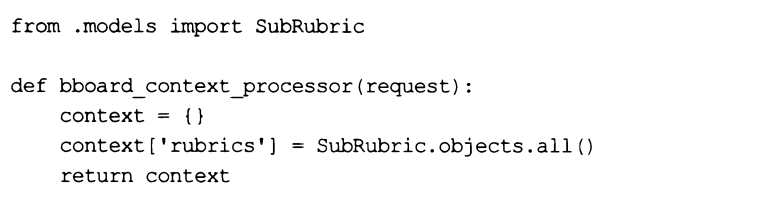
Сохраним весь исправленный код, запустим отладочный веб-сервер (не забыв при этом отключить обработку статических файлов), войдем на административный сайт и добавим несколько надрубрик и подрубрик.

**Вывод списка рубрик в вертикальной панели навигации**

Сначала необходимо поместить в состав контекста каждого шаблона переменную, в которой хранится список подрубрик (именно на его основе мы будем формировать пункты панели навигации). Можно создавать такую переменную в каждом контроллере, но это очень трудоемко. Поэтому объявим и зарегистрируем в проекте обработчик контекста, в котором и будет формироваться список подрубрик.

Условимся, что список подрубрик будет помещаться в переменную rubrics контекста шаблона.

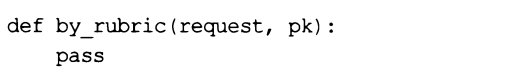
Создадим в пакете приложения модуль middlewares.py и запишем в него код обработчика контекста bboard context processor (), приведен.



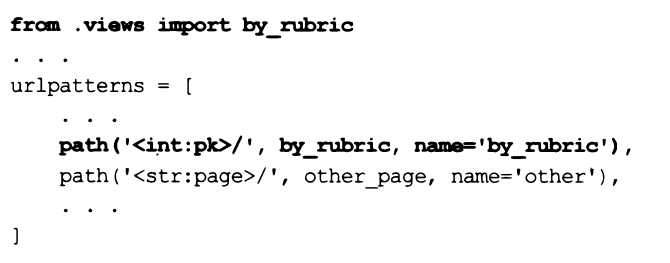
Откроем модуль settings.py пакета конфигурации и зарегистрируем только что написанный обработчик контекста. Для этого добавим имя этого обработчика в список Context\_processors из параметра options:



Выполним еще пару подготовительных действий. Во-первых, в модуле views.py пакета приложения объявим "пустой" контроллер-функцию by rubric ():



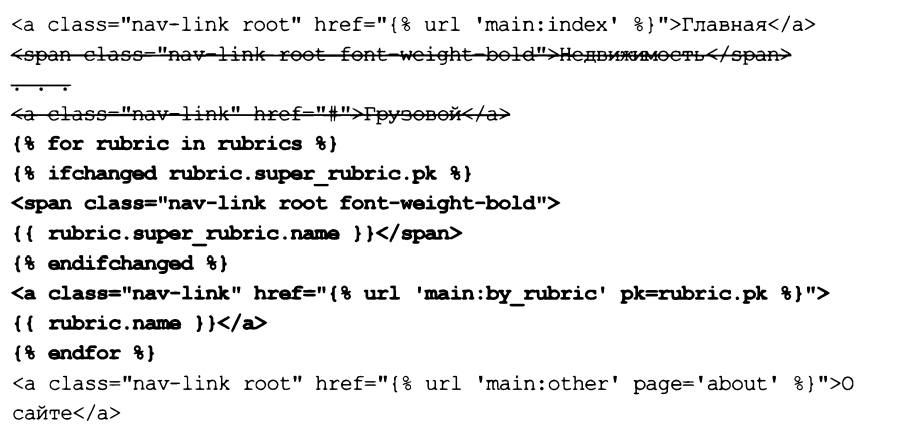
Во-вторых, добавим в список маршрутов уровня приложения маршрут, ведущий на этот контроллер:



Этот маршрут мы поместим перед маршрутом, ведущим на контроллер otherjpage (), который выводит вспомогательные страницы. Если же мы поместим его после упомянутого ранее маршрута, то при просмотре списка маршрутов Django примет присутствующий в интернет-адресе ключ рубрики за имя шаблона страницы и запустит контроллер other page (), что приведет к ошибке 404.

Зачем мы объявляли эти контроллер и маршрут? Чтобы прямо сейчас сформировать в панели навигации гиперссылки с правильными интернет-адресами. Позже мы заменим контроллер-“заглушку” другим, выполняющим полезную работу — вывод объявлений из выбранной посетителем рубрики.

Откроем шаблон layout\basic.html (давайте ради краткости не указывать папку Templates в путях к шаблонам — мы уже давно знаем, что шаблоны хранятся в папке Templates) и исправим код вертикальной панели навигации следующим образом:



Мы перебираем список подрубрик, хранящийся в переменной rubrics контекста шаблона (эту переменную создал наш обработчик контекста bboard\_context\_Processor ()), и для каждой подрубрики выводим:

□ если ключ связанной надрубрики изменился (т. е. если начали выводиться подрубрики из другой надрубрики) — пункт с именем надрубрики;

□ пункт-гиперссылку с именем подрубрики.

Сохраним код, на всякий случай перезапустим отладочный веб-сервер, обновим открытую в веб-обозревателе страницу и полюбуемся на список рубрик. Только не будем пока щелкать на них.

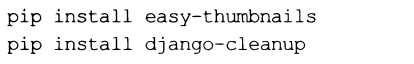
1. Объявления

Теперь можно приступать к работе над объявлениями. Мы создадим страницу для росмотра объявлений, относящихся к выбранной рубрике, с поддержкой пагинации и поиска, страницу сведений о выбранном объявлении, страницы для добавления, правки и удаления объявлений. И не забудем вывести на странице профиля объявления, оставленные текущим пользователем.

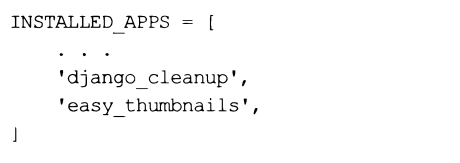
**Подготовка к обработке выгруженных файлов**

В составе каждого объявления будет присутствовать графическое изображение с основной иллюстрацией к продаваемому товару. Помимо этого, пользователь может создать в объявлении произвольное количество дополнительных иллюстраций.

Чтобы Django смог обработать выгруженные посетителями файлы, необходимо установить три дополнительных библиотеки: Easy Thumbnails (создает миниатюры), Django-cleanup (удаляет выгруженные файлы после удаления хранящих их записей моделей) и Pillow (обеспечивает поддержку графики, будет автоматически установлена при установке Easy Thumbnails). Установим их, набрав команды:

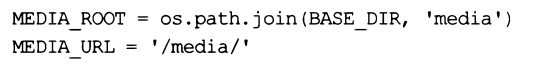


Добавим программные ядра двух последних библиотек: приложения easy Thumbnails и django cleanup— в список зарегистрированных в проекте. Для этого откроем модель settings.py пакета конфигурации и добавим в список параметра Installed apps псевдонимы этих приложений:

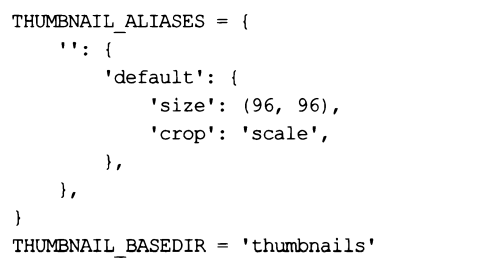


Для хранения самих выгруженных файлов отведем папку media, которую создадим в папке проекта. Для хранения миниатюр создадим в ней папку thumbnails.

В модуле settings.py укажем путь к папке media и префикс для интернет-адресов выгруженных файлов:

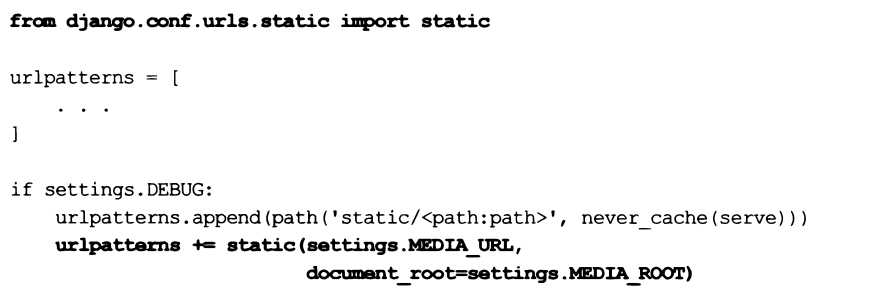


И сразу же добавим туда настройки приложения easy\_thumbnaiis:



Мы задали для миниатюр один-единственный пресет, указывающий выполнять простое масштабирование до размеров 96x96 пикселов. Также мы задали имя вложенной папки, в которой будут храниться миниатюры, — thumbnails.

И наконец, добавим в список маршрутов уровня проекта маршрут для обработки выгруженных файлов:



**Модели объявлений и дополнительных иллюстраций**

Мы создадим две модели: одну — для объявлений и вторую — для дополнительных иллюстраций.

**Модель самих объявлений**

Модель, хранящая объявления, будет называться вь.

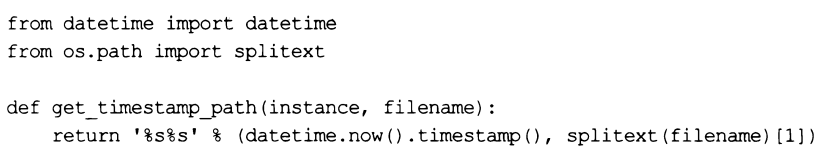
В поле rubric, устанавливающем связь с моделью подрубрик subrubric, мы указали запрет каскадного удаления, чтобы предотвратить случайное удаление подрубрики вместе со всеми относящимися к ней объявлениями.

Графические файлы, сохраняемые в поле image модели, будут иметь в качестве имен текущие временные отметки. Так мы приведем имена к единому типу и заодно устраним ситуацию, когда имя выгруженного файла настолько длинное, что оно не помещается в поле модели.

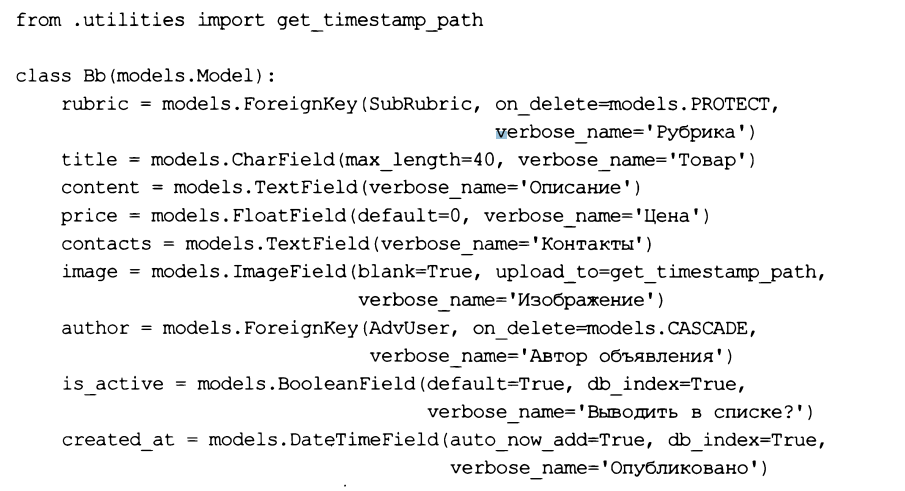
При разработке модели объявления нужно учесть еще один момент. Чуть позже мы напишем модель дополнительных иллюстраций, которую свяжем с моделью объявлений связью "один-со-многими". Если при объявлении этой связи мы разрешим каскадное удаление, то при удалении объявления будут уничтожены все относящиеся к нему дополнительные иллюстрации. Но это действие выполнит не Django, а СУБД, отчего приложение django\_cieanup не получит сигнала об удалении записей и не сможет в ответ удалить хранящиеся в них графические файлы. В результате эти файлы останутся на диске бесполезным мусором.

Мы обязательно решим эту проблему позже. А сейчас условимся об имени модели дополнительных иллюстраций — Additional Image.

Ранее мы создали в пакете приложения модуль utilities.py, в который записали объявление функции, выполняющей отправку писем. Этот модуль— отличное место для сохранения кода, не относящегося напрямую ни к моделям, ни к редакторам, ни к контроллерам. Поместим в него объявление функции get\_timestamp\_Path о, генерирующей имена сохраняемых в модели выгруженных файлов.



Приведен код, объявляющий сам класс модели вь.



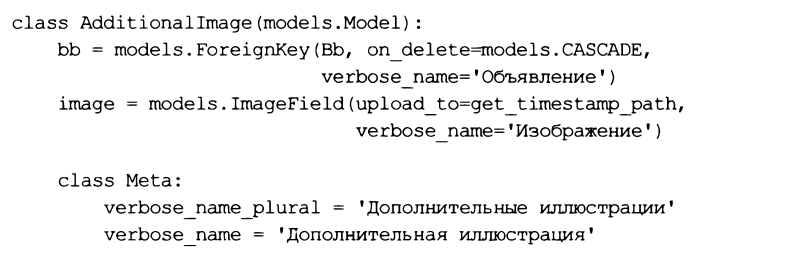


В переопределенном методе delete () перед удалением текущей записи мы перебираем и вызовом метода delete () удаляем все связанные дополнительные иллюстрации. При вызове метода delete () возникает сигнал post delete, обрабатываемый приложением django cieanup, которое в ответ удалит все файлы, хранящиеся в удаленной записи.

Модель дополнительных иллюстраций мы назовем, как условились ранее, Additionalimage.

Графические файлы, сохраняемые в поле image, также получат в качестве имен текущие временные отметки. Для формирования имен файлов применим объявленную ранее функцию get\_timestamp\_path () из модуля utilities.ру.

Готовый код модели Additional image приведен.

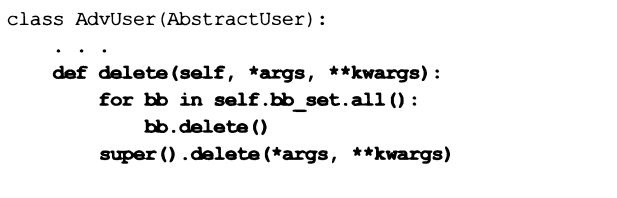


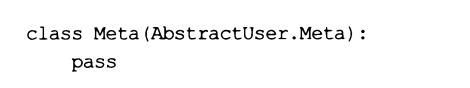
Сохраним код моделей, создадим и выполним миграции. И сделаем еще одно очень важное дело.

**Реализация удаления объявлений в модели пользователя**

Ранее мы сделали так, чтобы при удалении объявления явно удалялись все связанные с ним дополнительные иллюстрации. Это нужно для того, чтобы приложение django cleanup удалило хранящие ИХ файлы.

Теперь сделаем так, чтобы при удалении пользователя удалялись оставленные им объявления. Для этого добавим в код модели advuser следующий фрагмент:





**Инструменты для администрирования объявлений**

Чтобы с объявлениями можно было работать посредством административного сайта, объявим редактор объявлений bbadmin и встроенный редактор дополнительных иллюстраций Additionaiimageiniine. Их код приведен:



На страницах добавления и правки объявлений выведем раскрывающиеся списки подрубрики и пользователя в одну строку — ради компактности.

Сохраним код, запустим отладочный веб-сервер, войдем на административный сайт и добавим несколько объявлений, обязательно с дополнительными иллюстрациями.

Попробуем исправить одно объявление и удалить другое, проверив, действительно ли при этом будут удалены все файлы, хранящиеся в самом объявлении, и связанные с ним иллюстрации.

**Вывод объявлений**

Мы создадим две страницы:

□ страницу списка объявлений, относящихся к выбранной посетителем рубрике, с поддержкой пагинации и поиска объявлений по введенному слову;

□ страницу сведений о выбранном объявлении, на которой будут выводиться также и дополнительные иллюстрации.

Кроме того, реализуем вывод десяти наиболее "свежих” объявлений на главной странице.

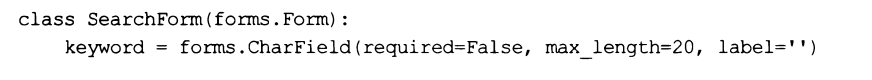
**Вывод списка объявлений**

Вывод списка объявлений с поддержкой поиска — достаточно сложная задача. Нам понадобится обычная, не связанная с моделью, форма для ввода искомого слова, контроллер и шаблон. А еще придется решить весьма серьезную проблему корректного возврата, о которой мы поговорим чуть позже.

**Форма поиска и контроллер списка объявлений**

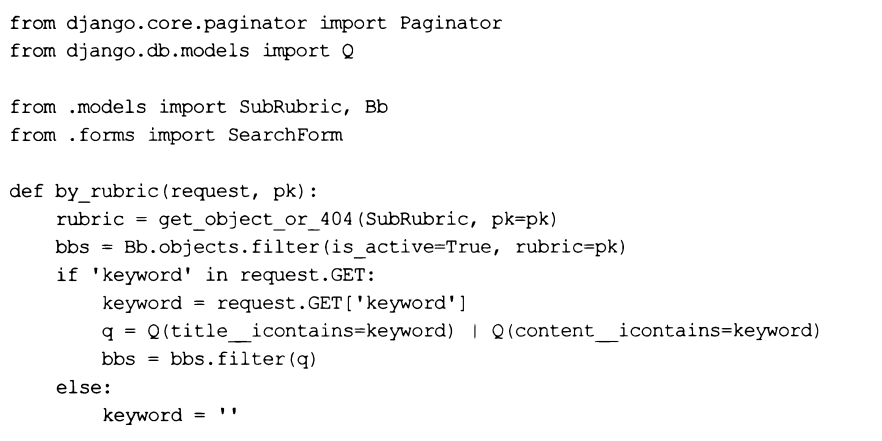
Сразу условимся, что искомое слово, введенное посетителем, будем пересылать контроллеру методом GET в GET-параметре keyword. Поле в форме для ввода искомого слова назовем так же.

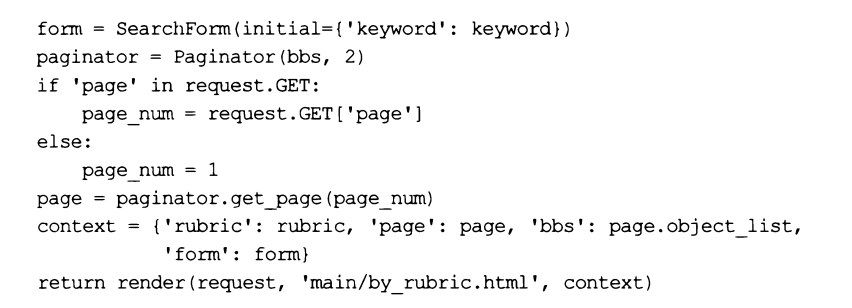
Код формы поиска searchform очень прост — убедимся в этом сами.



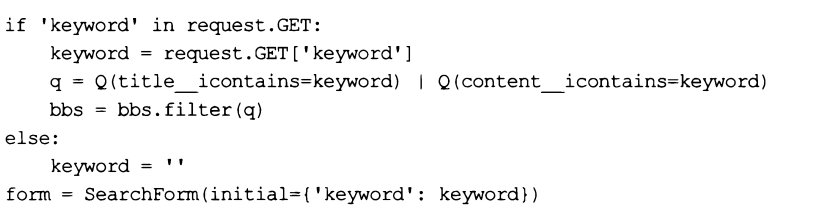
Поскольку посетитель может ввести в поле keyword искомое слово, а может и не ввести (чтобы отменить выполненный ранее поиск и вновь вывести все объявления из списка), мы пометили это поле как необязательное для заполнения. А еще убрали у этого поля надпись, присвоив параметру label пустую строку— все равно такого рода поля выводятся без надписей.

Ранее, чтобы проверить написанный тогда код, мы создали ничего не делающий контроллер-функцию by rubrico. Настала пора "наполнить" его полезным кодом.





Извлекаем выбранную посетителем рубрику — нам понадобится вывести на странице ее название. Затем выбираем объявления, относящиеся к этой рубрике и помеченные для вывода (те, у которых поле is active хранит значение True). После этого выполняем фильтрацию уже отобранных объявлений по введенному посетителем искомому слову, взятому из GET-параметра keyword.

Вот фрагмент кода, ’’отвечающий” за фильтрацию объявлений по введенному посетителем слову:

Ради простоты получаем искомое слово непосредственно из GET-параметра Keyword. Затем формируем на основе полученного слова условие фильтрации, применив объект Q, и выполняем фильтрацию объявлений.

Далее создаем экземпляр формы searchform, чтобы вывести ее на экран. Конструктору ее класса в параметре initial передаем полученное ранее искомое слово, чтобы оно присутствовало в выведенной на экран форме.

Не забываем создать пагинатор, указав у него количество записей в части равным 2.

Это позволит нам проверить, работает ли пагинация, имея в базе данных всего три-четыре объявления. Наконец, выводим страницу со списком объявлений, применив шаблон main\by\_rubric.html. С написанием которого придется подождать.

**Реализация корректного возврата**

Предположим, что мы написали весь код, который будет выводить на экран и списки объявлений, разбитые на рубрики, и сведения о выбранном объявлении. И вот посетитель заходит на сайт, выбирает какую-либо рубрику, пролистывает несколько частей, сформированных пагинатором, находит нужное ему объявление и щелкает на гиперссылке, чтобы просмотреть это объявление полностью. Открывается страница со сведениями об объявлении, посетитель смотрит их, после чего щелкает на гиперссылке возврата на список объявлений... И попадает на самую первую часть этого списка.

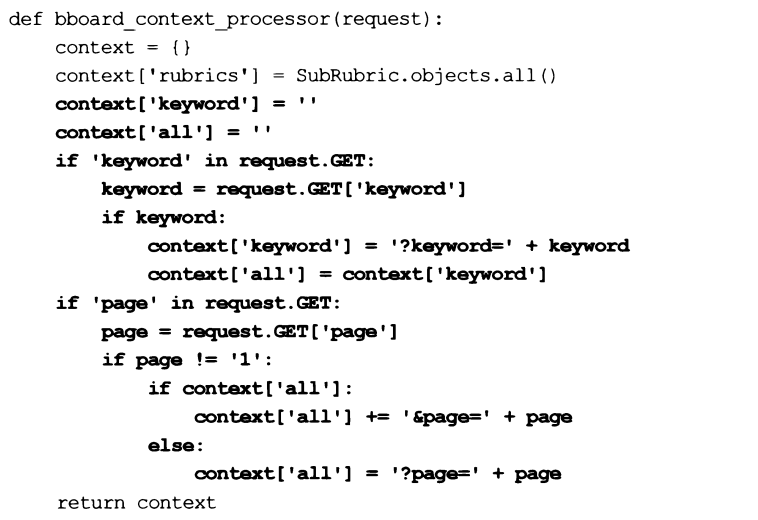
То же самое произойдет, если посетитель выполнит поиск, а уже потом отправится смотреть сведения о каком-либо объявлении. Когда он щелкнет на гиперссылке возврата, то вернется в изначальный список объявлений, в котором не был выполнен поиск.

Как избежать этой проблемы, в общем, понятно. Номер выводимой части и искомое слово у нас передаются посредством GET-параметров раде и keyword соответственно. Тогда, чтобы вернуться на нужную часть списка уже отфильтрованных по заданному слову объявлений, следует передать эти параметры странице сведений об объявлении.

Конечно, готовый набор GET-параметров можно получить из элемента с ключом Query string словаря, который хранится в атрибуте мета объекта запроса. Но нет смысла передавать параметр раде, если его значение равно 1, и параметр keyword с ’’пустой” строкой — это их значения по умолчанию.

Также можно формировать набор GET-параметров в контроллере. Но, по принятым в Django соглашениям, весь код, ’’ответственный” за формирование страниц, следует помещать в шаблон, посредник или — наш случай! — обработчик контекста.

Откроем модуль middlewares.py пакета приложения, найдем код обработчика контекста bboardcontextprocessor о, написанный ранее, и вставим в него следующий фрагмент:



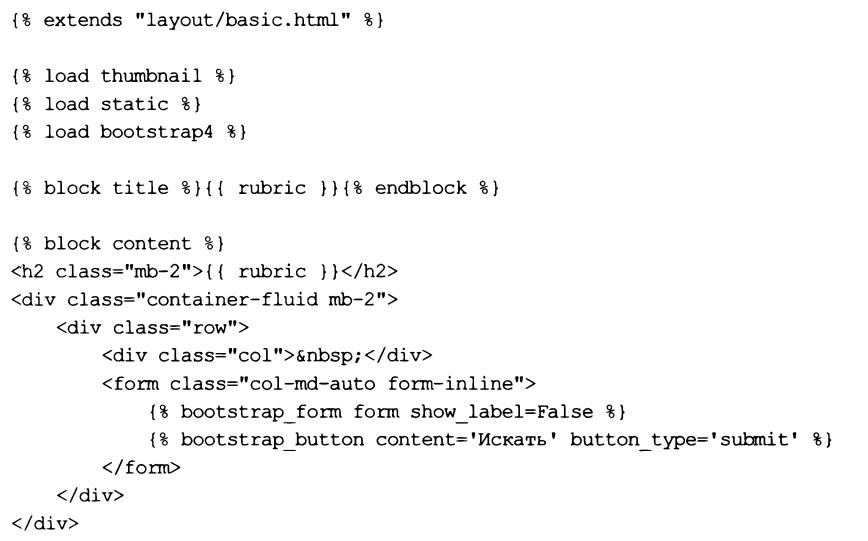
Добавленный код создаст в контексте шаблона две переменные:

Keyword — с GET-параметром keyword, который понадобится для генерирования интернет-адресов в гиперссылках пагинатора;

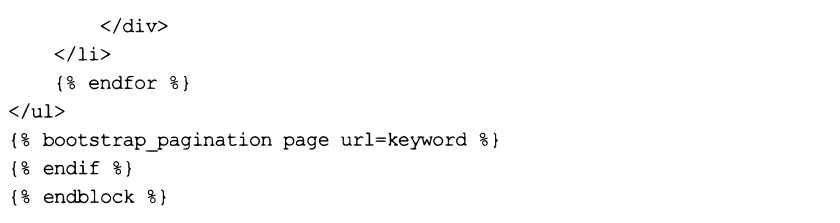
Ail— с GET-параметрами keyword и раде, которые мы добавим к интернет-адресам гиперссылок, указывающих на страницы сведений об объявлениях.

**Шаблон веб-страницы списка объявлений**

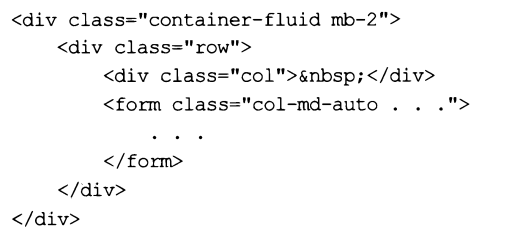
Код шаблона main\by\_rubric.html, который сформирует страницу списка объявлений, приведен:







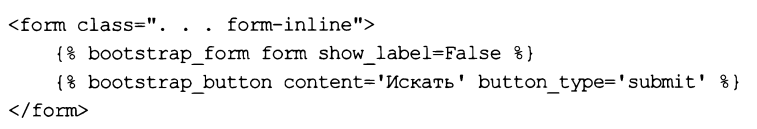
Чтобы вывести форму поиска, прижав ее к правой части страницы, используем конструкцию следующего вида:



Знакомый нам по коду ранее стилевой класс container-fluid заставляет элемент, к которому он привязан, вести себя как обычная таблица HTML (стилевой класс mb-з устанавливает средней величины внешний отступ снизу, отделяющий форму от собственно списка объявлений). Элемент с привязанным стилевым классом row, вложенный в этот элемент, ведет себя как строка таблицы. Вложенный в такую ’’Строку” элемент со стилевым классом col ведет себя как ячейка таблицы, растягивающаяся на всю доступную ширину, а элемент со стилевым классом coi-mdauto — как ячейка с шириной, равной ширине ее содержимого.

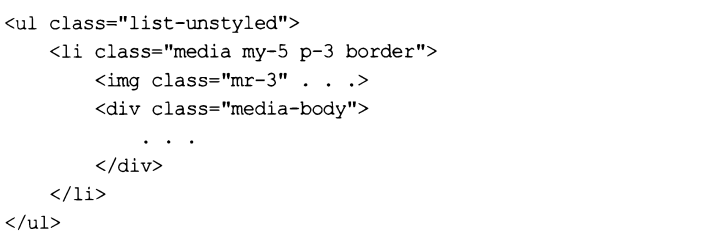
Первым элементом-“ячейкой” у нас является пустой блок, а вторым — форма поиска. В результате на экране мы увидим лишь форму, сдвинутую к правому краю страницы.

Посмотрим на код самой формы:



Стилевой класс form-inline укажет веб-обозревателю вывести все элементы управления формы в одну строку. В самой форме мы не помещаем электронный жетон защиты (который генерируется тегом шаблонизатора csrf token), поскольку он там совершенно не нужен, а для вывода кнопки применяем тег bootstrap button (использованный ранее тег buttons добавляет лишний блок, который в этом случае также не нужен).

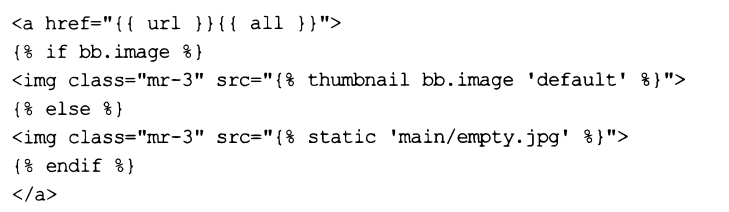
Для вывода очередной части списка объявлений применяем особые перечни Bootstrap. Взглянем на код, который их формирует:



Сам перечень создается маркированным списком HTML (тегом <ui>) со стилевым классом list-unstyled. Отдельная позиция перечня формируется пунктом списка (тегом <ii>) со стилевым классом media (стилевой класс ту-5 задает большие внешние отступы сверху и снизу, стилевой класс р-з — внутренние отступы среднего размера со всех сторон, а стилевой класс border — рамку вокруг элемента). В пункте помещается тег <img> со стилевым классом mr-з и произвольное количество других элементов.

Гиперссылку на страницу сведений об объявлении создаем на базе основной иллюстрации и названия товара. Чтобы не генерировать интернет-адрес для этих гиперссылок дважды, сохраним его в переменной url:





Если основная иллюстрация в объявлении указана, то будет выведена ее миниатюра. Если же автору объявления нечем его иллюстрировать, будет выведено изображение из статического файла main\empty.jpg.

Теперь посмотрим на самую первую строку приведенного ранее фрагмента. Там, в создающем гиперссылку теге <а>, мы сформировали интернет-адрес, объединив содержимое только что созданной в шаблоне переменной url и переменной ail контекста шаблона, в которой хранятся GET-параметры keyword и раде. В результате интернет-адрес гиперссылки включит искомое слово и номер части.

Еще не знакомый нам стилевой класс font-italic задает для элемента курсивное начертание шрифта.

Напоследок посмотрим на тег шаблонизатора, создающий пагинатор:

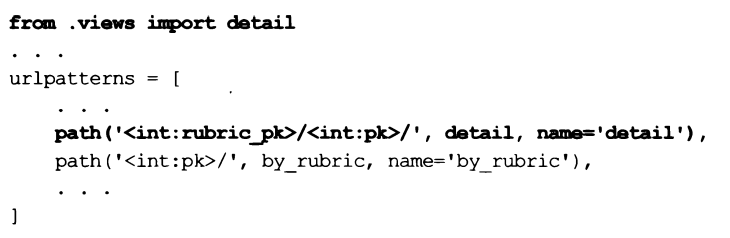


Базовый интернет-адрес берется из переменной keyword контекста шаблона, в которой хранится одноименный GET-параметр с искомым словом. В результате при переходе на другую часть пагинатора контроллер получит это слово и выведет отфильтрованный по нему список объявлений.

Осталось найти в Интернете какое-либо подходящее графическое изображение и сохранить его под именем empty.jpg в папке static\main папки проекта. Если же найденное изображение хранится в формате, отличном от JPEG, необходимо соответственно изменить расширение имени файла в коде шаблона.

**Веб-страница сведений о выбранном объявлении**

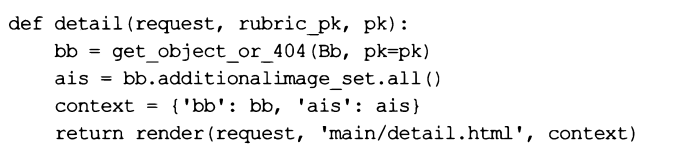
Эту страницу будет выводить контроллер-функция detail (), который мы напишем позже. А сейчас запишем ведущий на него маршрут, поместив его непосредственно перед маршрутом, ведущим на контроллер by rubric ():



Записанные в обоих маршрутах шаблонные пути показывают своего рода иерархию рубрик и отдельных объявлений:

□ страницы со списками объявлений, принадлежащих определенной рубрике, имеют шаблонные пути формата <рубрика>Г, о страницы с отдельными объявлениями, принадлежащими определенной рубрике, имеют шаблонные пути формата <рубрика>1<объявление>1.

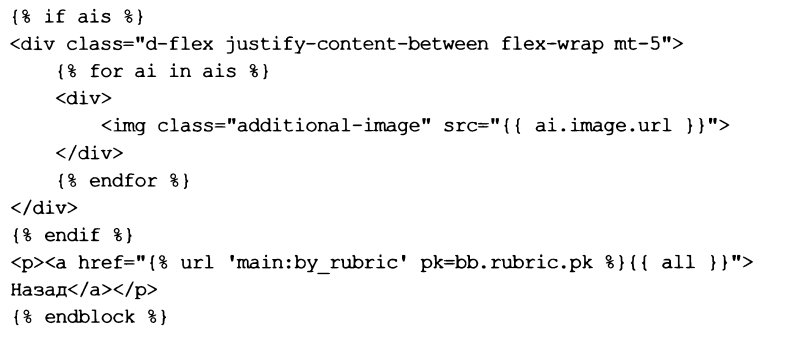
Код контроллера detail о приведен. Реализуем его в виде функции, поскольку далее будем формировать в нем еще и список комментариев к объявлению, а в контроллере-функции это сделать проще.



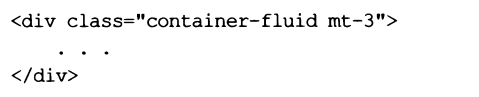
Помимо самого объявления, которое мы помещаем в переменную ьь контекста шаблона, также готовим перечень связанных с ним дополнительных иллюстраций, записав его в переменную ais.

Код шаблона main\detail.html, выводящего страницу сведений об объявлении, приведен.



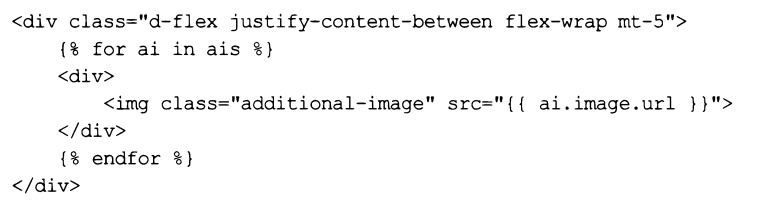


Код, выводящий основные сведения об объявлении (название, описание и цену товара, контакты, временную отметку добавления объявления, основную иллюстрацию, если таковая указана), располагается между вот этими тегами:



Аналогичный код мы рассматривали уже дважды (причем второй раз — непосредственно в текущей главе), так что он уже должен быть нам знаком.

А вот код, выводящий дополнительные иллюстрации, заслуживает более пристального рассмотрения. Вот он:



К блоку, в котором будут выводиться дополнительные иллюстрации, привязаны следующие стилевые классы:

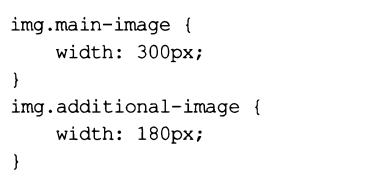
□ d-flex —устанавливает для элемента так называемую гибкую разметку, при которой дочерние элементы выстраиваются внутри родителя по горизонтали;

□ justify-content-between— указывает, что дочерние элементы (собственно дополнительные иллюстрации) должны располагаться внутри родителя на равномерном расстоянии друг от друга;

□ flex-wrap — если дочерним элементам не хватит места, чтобы выстроиться по горизонтали, то не помещающиеся элементы будут перенесены на следующую строку;

□ mt-5 — большой внешний отступ сверху, чтобы отделить дополнительные иллюстрации от основной информации.

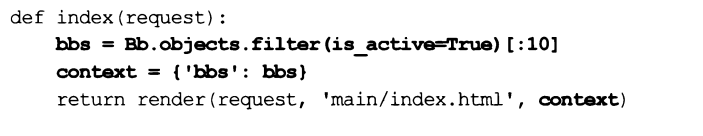
Осталось установить ширину основной и дополнительных иллюстраций— соответственно 300 и 180 пикселов. Откроем таблицу стилей main\style.css и добавим в нее следующий фрагмент кода:



Сохраним весь код, перезапустим отладочный веб-сервер и зайдем на страницу со списком объявлений, относящихся к какой-либо из категорий. После чего попробуем перейти на страницу со сведениями об объявлении.

**Вывод последних 10 объявлений на главной веб-странице**

Найдем код контроллера-функции index о, который выводит главную страницу, и добавим в него фрагмент, выбирающий из базы последние 10 объявлений:



Осталось дописать в шаблоне main\index.html код, выводящий объявления.

За основу можете взять аналогичный код из шаблона Main\by\_rubric.html, приведенный ранее.

**Работа с объявлениями**

Осталось подготовить инструменты, посредством которых зарегистрированные пользователи будут просматривать перечень своих объявлений, добавлять, править и удалять объявления.

**Вывод объявлений, оставленных текущим пользователем**

Сначала добавим в контроллер-функцию profile () следующий код:

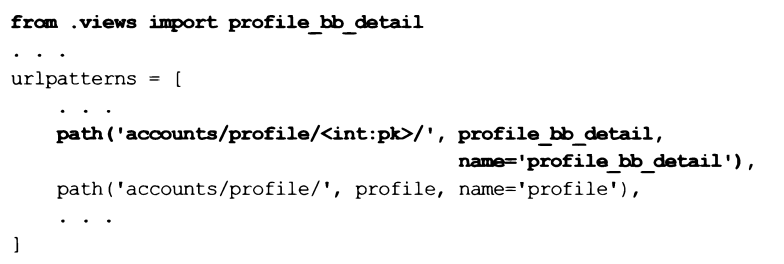


Он фильтрует объявления по значению поля author (ключ автора объявления, которым является зарегистрированный пользователь), сравнивая это значение с ключом текущего пользователя.

Для вывода списка объявлений мы вставим в шаблон main\profile.html, формирующий страницу пользовательского профиля, код, который мы практически без изменений можем взять из шаблона main\by\_rubric.html.

Если пользователь зайдет на страницу своего профиля и щелкнет на объявлении, то он попадет на общедоступную страницу сведений об объявлении, написанную ранее. Возможно, это окажется не очень удобно, так что давайте создадим другую, административную страницу сведений об объявлении, доступную лишь зарегистрированным пользователям.

Сначала объявим маршрут, который укажет на эту страницу, поместив его непосредственно перед маршрутом, ведущим на страницу профиля:

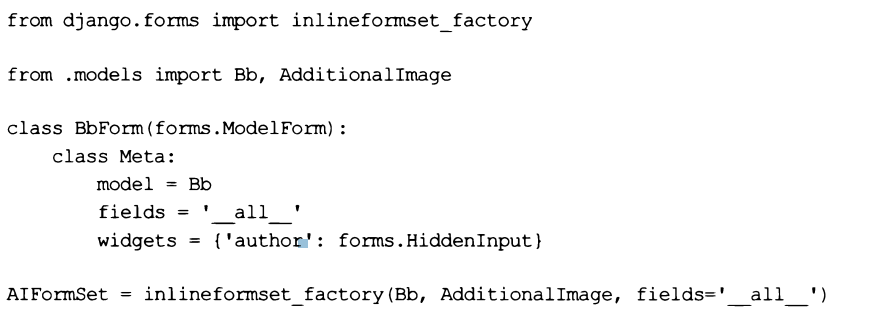


Контроллер-функцию profiie\_bb\_detaii(), который выведет страницу сведений об объявлении, напишем самостоятельно, взяв за основу контроллер detail. Не забудем сделать его доступным только для зарегистрированных пользователей, выполнивших вход. Также самостоятельно напишем шаблон, который формирует эту страницу и за основу которого можно взять шаблон

Main\detail.html.

**Добавление, правка и удаление объявлений**

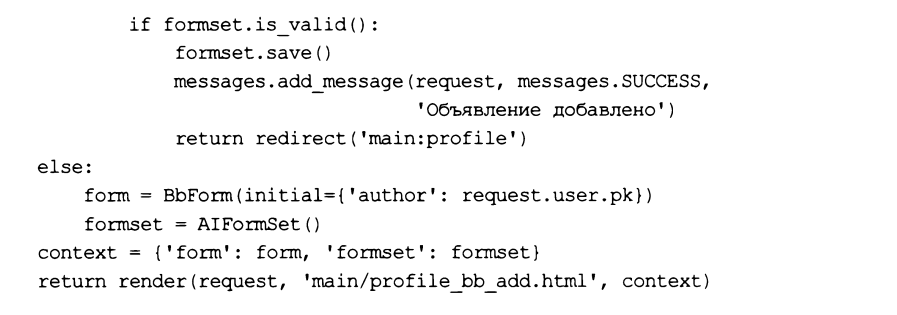
Объявим форму bbform, связанную с моделью вь, для ввода самого объявления и встроенный набор форм aiformset, связанный с моделью Additionaiimage, в которые будут заноситься дополнительные иллюстрации. Объявляющий их код приведен.



В форме будем выводить все поля модели вь. Для поля автора объявления author зададим в качестве элемента управления Hiddenlnput, т. е. скрытое поле — все равно значение туда будет заноситься программно.

Контроллер, добавляющий объявление, реализуем в виде функции (в виде класса его реализовать будет сложнее) и назовем profiie bb addo. Его код приведен.





Здесь нужно отметить три важных момента. Во-первых, при создании формы перед выводом страницы сохранения мы заносим в поле author формы ключ текущего пользователя, который станет автором объявления.

Во-вторых, во время сохранения введенного объявления, при создании объектов формы и набора форм, мы должны передать конструкторам их классов вторым позиционным параметром словарь со всеми полученными файлами (он хранится в атрибуте files объекта запроса). Если мы не сделаем этого, то отправленные пользователем иллюстрации окажутся потерянными.

В-третьих, при сохранении мы сначала выполняем валидацию и сохранение формы

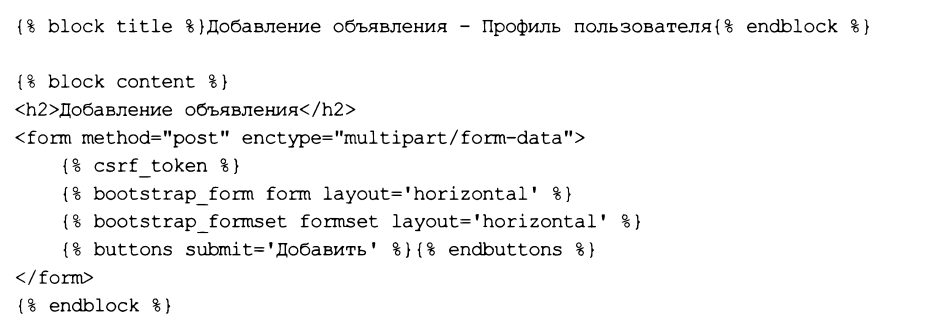
Самого объявления. Метод save о в качестве результата возвращает сохраненную запись, и эту запись мы должны передать через параметр instance конструктору класса набора форм. Это нужно для того, чтобы все дополнительные иллюстрации после сохранения оказались связанными с объявлением.

Напишем маршрут, который укажет на страницу добавления, поместив его перед маршрутом, указывающим на страницу профиля:



Займемся шаблоном main\profile\_bb\_add.html, который создаст страницу добавления объявления. Его код приведен.





Обязательно укажем у формы метод кодирования данных muitipart/form-data.

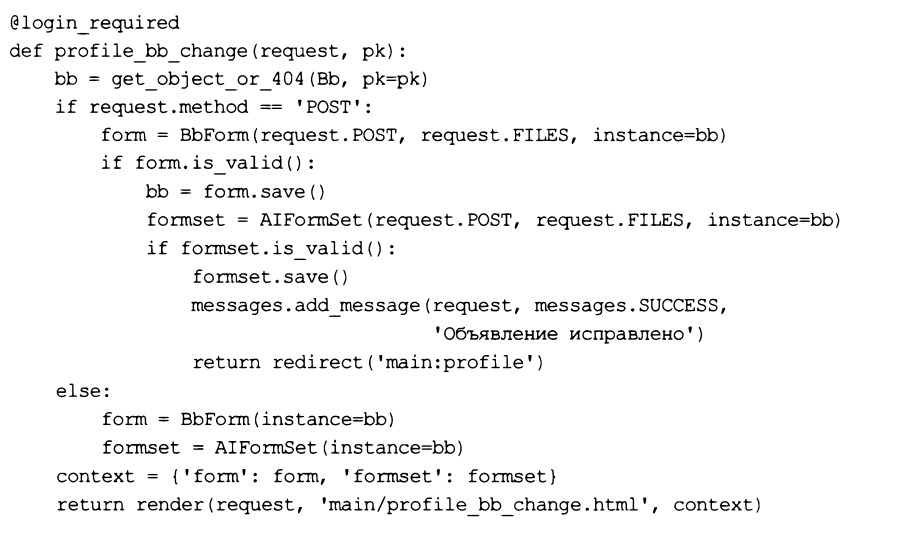
Если этого не сделать, то занесенные в форму файлы не будут отправлены. А набор форм выведем с помощью тега шаблонизатора bootstrap formset.

Наконец, в шаблон страницы профиля main\profile.html добавим гиперссылку на страницу добавления объявления:



После этого можно сохранить код и попробовать функциональность по добавлению новых объявлений в деле.

Код контроллеров profile\_bb\_change() И profile bb delete (), которые соответственно правят и удаляют объявление, приведен.





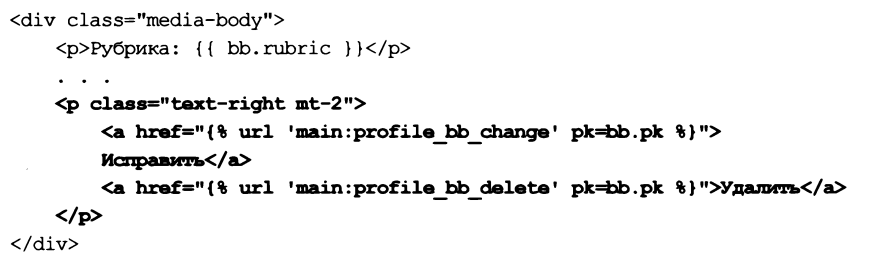
Для простоты на странице удаления объявления не будем выводить дополнительные иллюстрации — они там не особо нужны.

Объявим необходимые маршруты:



Шаблоны main\profile\_bb\_change.html и main\profile\_bb\_delete.html вы можете написать самостоятельно, использовав в качестве основы ранее написанные шаблоны.

В шаблон main\profile.html нужно добавить код, создающий гиперссылки для правки и удаления каждого из занесенных пользователем в базу объявлений:



Напоследок следует проверить в действии реализованную функциональность по правке и удалению объявлений.

1. Комментарии

К каждому из опубликованных на нашем сайте объявлений посетители смогут оставить комментарии.

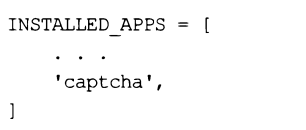
**Подготовка к выводу CAPTCHA**

Сделаем так, чтобы зарегистрированные пользователи могли оставлять комментарии беспрепятственно, а гости должны были дополнительно ввести CAPTCHA. Так мы хоть как-то обезопасим сайт от атаки служб рассылки спама.

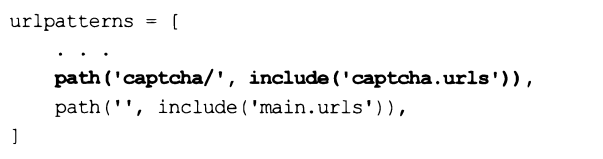
Установим библиотеку Django Simple Captcha:



Добавим в список зарегистрированных в проекте приложение captcha— программное ядро этой библиотеки:



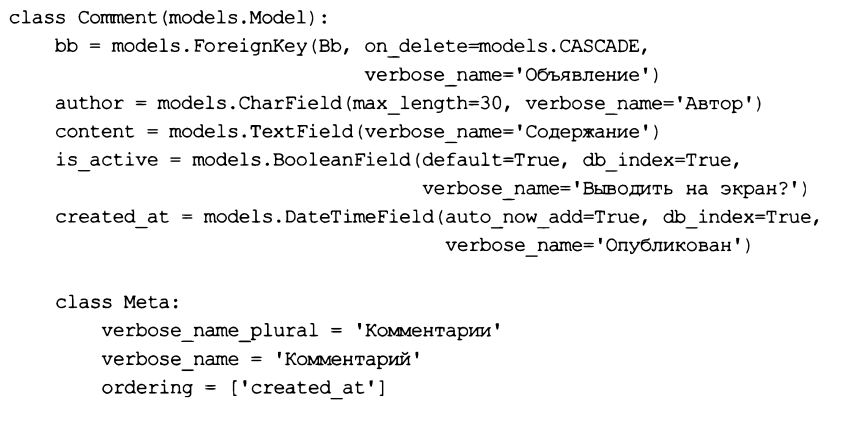
И объявим в списке маршрутов уровня проекта (модуль urls.py пакета конфигурации) маршрут, указывающий на это приложение:



Миграции пока выполнять не будем — еще нужно объявить модель для хранения комментариев.

Модель, хранящую комментарии, мы назовем comment.

Код, объявляющий модель comment, приведен.



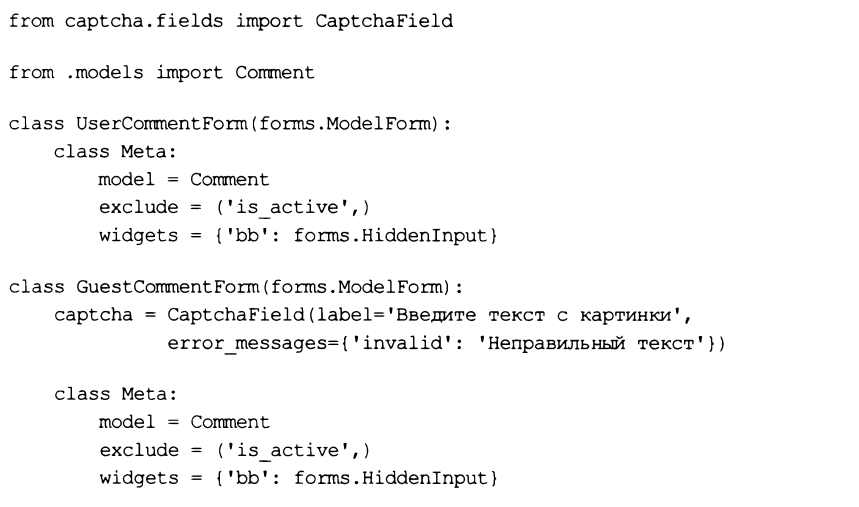
Для комментариев указываем сортировку по увеличению временной отметки их добавления. В результате более старые комментарии будут располагаться в начале списка, а более новые — в его конце.

Создадим миграции, после чего выполним их (при этом также будут выполнены миграции из библиотеки Django Simple Captcha).

**Вывод и добавление комментариев**

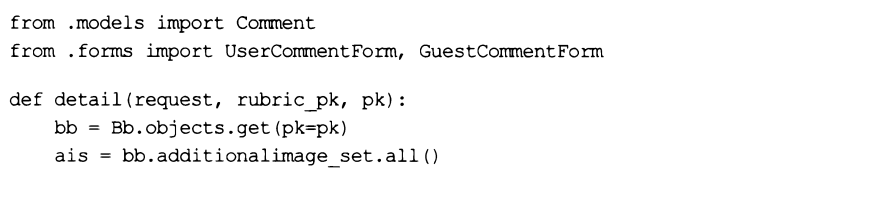
Список комментариев и форма для добавления комментария будут выводиться на общедоступной странице сведений об объявлении. Впоследствии мы сделаем так, чтобы комментарии выводились и на административной странице того же рода.

Объявим связанные с моделью Comment формы usercommentform И guestcommentform, в которые будут заносить комментарии, соответственно, зарегистрированные пользователи, выполнившие вход, и гости. Код этих форм приведен.



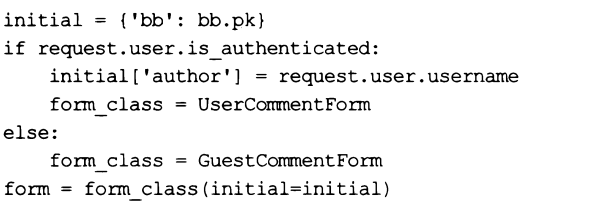
Поле is active (признак, будет ли комментарий выводиться на странице) уберем из форм, поскольку оно требуется лишь администрации сайта. У поля ьь, хранящего ключ объявления, с которым связан комментарий, укажем в качестве элемента управления скрытое поле.

Теперь необходимо существенно обновить код контроллера-функции detail о, написанного ранее. Реализовав в нем вывод комментариев и добавление нового комментария. Код обновленного контроллера приведен.





В поле ьь создаваемой формы ввода комментария заносим ключ выводящегося на странице объявления — с ним будет связан добавляемый комментарий. Если текущий пользователь выполнил вход на сайт, заносим его имя в поле author этой формы комментария. Наконец, если текущий пользователь выполнил вход на сайт, то создаем форму на основе класса usercommentform, а если не выполнил — на основе класса Guestcommentuser. Все эти действия выполняет фрагмент кода:



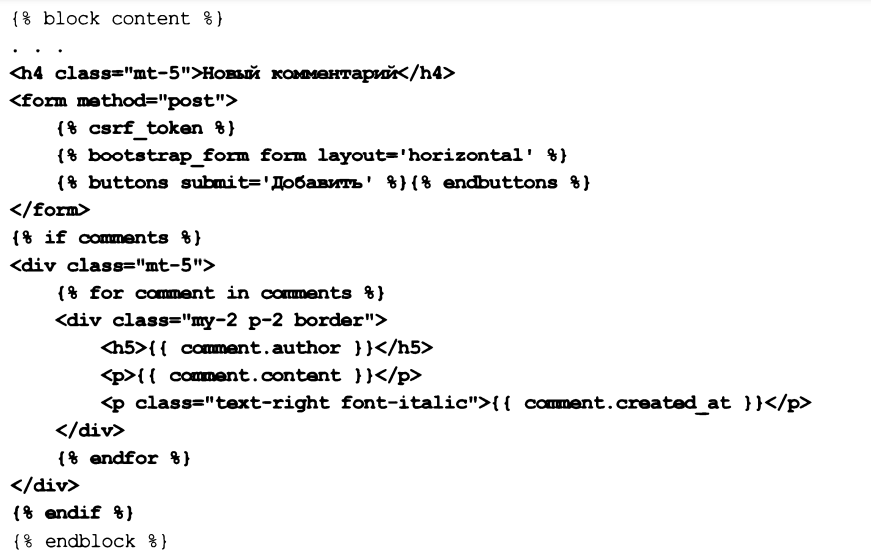
Объект формы сохраним в переменной с именем form. Форма из этой переменной впоследствии будет выведена на странице сведений об объявлении.

Далее, если полученный запрос был отправлен HTTP-методом POST, т. е. посетитель ввел комментарий и отправил его на сохранение, создаем еще один объект формы, передав конструктору полученные данные. Второй объект формы сохраняется в переменной c form. После этого выполняем валидацию второй формы.

Если валидация прошла успешно, сохраняем введенный комментарий и выводим соответствующее всплывающее сообщение. Когда страница будет выведена, она будет содержать только что добавленный комментарий и пустую форму для ввода комментария из переменной form.

Если же валидация прошла неуспешно, переносим форму из переменной c form в переменную form. Эта форма, хранящая некорректные данные и сообщения об ошибках ввода, впоследствии будет выведена на экран, и посетитель сразу увидит, что он ввел не так. Также выводим всплывающее сообщение о неуспешном добавлении комментария.

В шаблоне main\detail.html отыщем тег шаблонизатора block content . . . Endblock и вставим перед закрывающим тегом код, выводящий комментарии:



Стилевой класс my-2 задает небольшие внешние отступы сверху и снизу, а стилевой класс р-2 — небольшие внутренние отступы со всех сторон. Мы используем их, чтобы создать просветы между отдельными комментариями.

Закончив программирование, попробуем открыть страницу со сведениями о каком-либо объявлении и добавить один или два комментария. После этого выполним вход на сайт и снова попытаемся добавить комментарий. Отметим, что во втором случае форма для добавления комментария не включает поле ввода CAPTCHA.

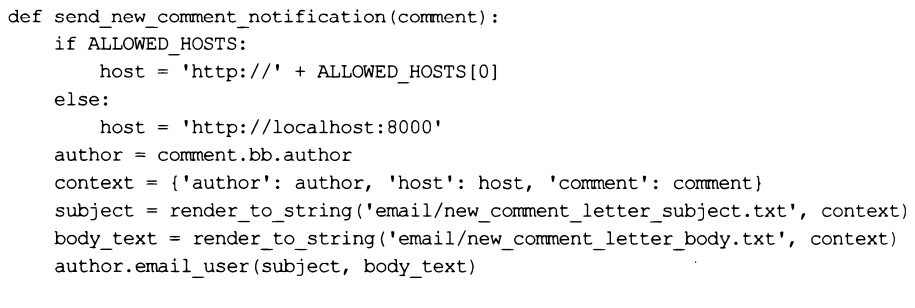
Осталось добавить список комментариев на административную страницу сведений об объявлении. Сделайте это самостоятельно. Форму для ввода комментария и соответствующую функциональность в контроллере можно не делать— вряд ли автор объявления будет комментировать его сам.

**Отправка уведомлений о новых комментариях**

Отправлять уведомления о появлении новых комментариев будем в обработчике сигнала post save, возникающего после сохранения записи в модели comment.

Откроем модуль utilities.py пакета приложения и добавим в него объявление функции send\_new\_comment\_notification (), которая и выполнит отправку уведомления.

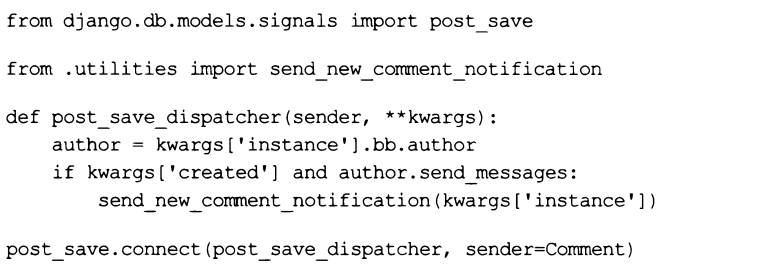
Ее КОД ПОХОЖ на КОД аналогичной функции send\_activation\_notification() (см. Код ранее) и приведен.



Шаблоны email\new\_comment\_letter\_subject.txt и email\new\_comment\_letter\_body.txt, которые сформируют тему и тело электронного письма, вы, уважаемые читатели, создайте самостоятельно. Их можно написать на основе аналогичных шаблонов Email\activation\_letter\_subject.txt и email\activation\_letter\_body.txt.

В письме нужно указать интернет-адрес административной страницы сведений об объявлении, к которому был оставлен новый комментарий.

Осталось лишь йривязать к сигналу post save обработчик, вызывающий функцию Send\_new\_comment\_notification() после Добавления комментария. Код, ВЫПОЛНЯЮЩИЙ привязку, поместим в модуль models.ру пакета приложения. Вот этот код:



Перед тем как отправлять оповещение, следует проверить, не запретил ли пользователь их отправку, т. Е. Не хранит ли поле send\_messages модели пользователя Advuser значение False.

Попробуем еще раз добавить комментарий к какому-либо объявлению и удостоверимся, что уведомление о новом комментарии было отправлено.

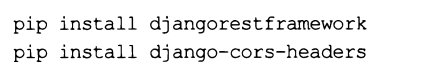
Осталось лишь объявить редактор для модели comment и зарегистрировать его на административном сайте Django. Автор полагает, что читатели сделают это самостоятельно.

1. Веб-служба REST

Мы напишем веб-службу, работающую по принципам REST. Она будет выдавать список из 10 последних объявлений, сведения о выбранном объявлении, список комментариев к заданному объявлению и позволит добавить новый комментарий. Для простоты разрешим комментировать объявления только зарегистрированным пользователям.

**Подготовка к разработке веб-службы**

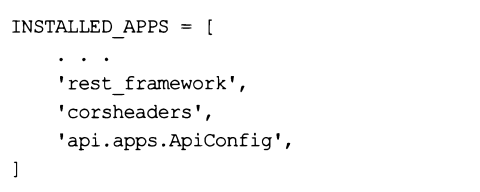
Установим библиотеки Django REST framework и django-cors-headers, для чего наберем команды:



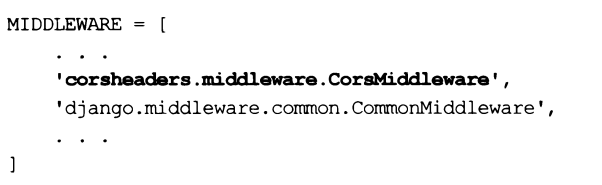
Сразу же создадим новое приложение api, в котором и реализуем функциональность веб-службы:



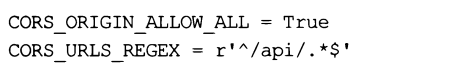
Добавим приложения rest\_framework И corsheaders — Программные Ядра ТОЛЬКО что установленных библиотек, а также только что созданное приложение api в список зарегистрированных в проекте:



Добавим в список зарегистрированных в проекте необходимый для работы посредник:

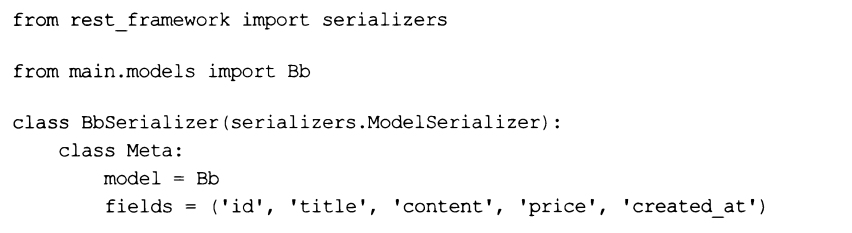


He забудем указать там же, в модуле settings.py пакета конфигурации, настройки, разрешающие доступ к веб-службе с любого домена:



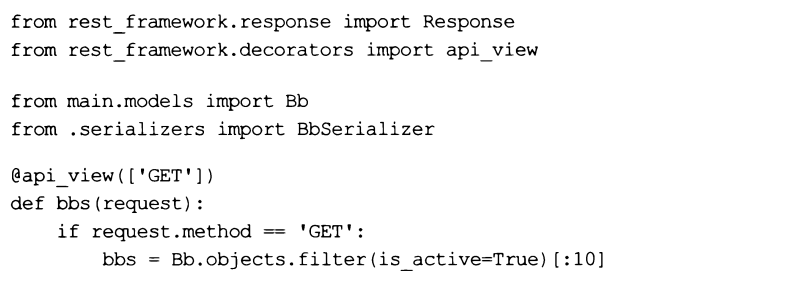
**Список объявлений**

Создадим в пакете приложения api модуль serializers.py. В нем сохраним код сериализатора bbserializer, формирующего список объявлений.



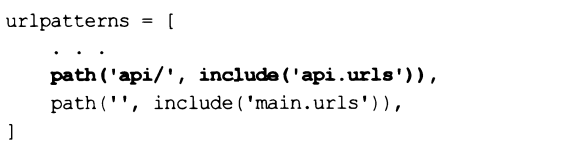
В составе сведений о каждом объявлении ради компактности будем отправлять лишь ключ, название, описание, цену товара и временную отметку создания объявления. Интернет-адрес основной иллюстрации и контакты отправим в составе сведений о выбранном объявлении.

Контроллер, который будет выдавать список объявлений, реализуем в виде функции и назовем bbs (). Его код приведен.

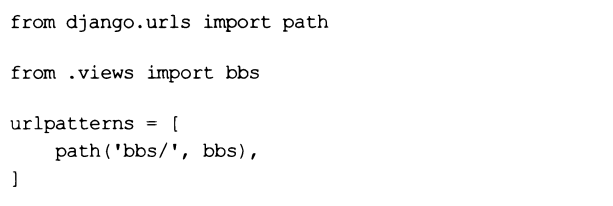




Откроем список маршрутов уровня проекта (модуль urls.py пакета конфигурации) и добавим маршрут, указывающий на приложение api:



В пакете приложения api создадим модель urls.py, в который запишем список маршрутов уровня этого приложения.



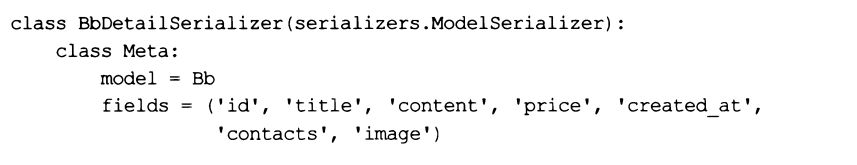
Пока что он содержит лишь маршрут, указывающий на только что написанный нами контроллер bbs ().

Сохраним код, запустим отладочный веб-сервер и попробуем получить список объявлений, перейдя по интернет-адресу http://localhost:8000/api/bbs/. Если мы все сделали правильно, то увидим веб-представление, показывающее последние 10 объявлений, которые были оставлены посетителями сайта.

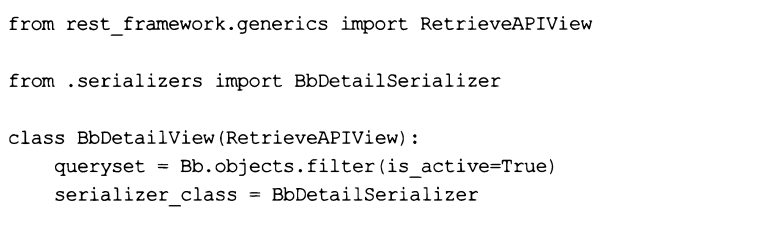
**Сведения о выбранном объявлении**

В состав сведений о выбранном объявлении, помимо ключа записи, названия, описания, цены товара и даты создания объявления, следует включить контакты и интернет-адрес основной иллюстрации.

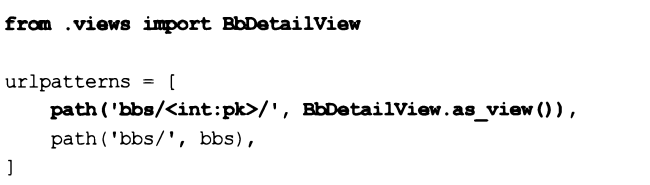
Учтем это при написании сериализатора bbdetaiiserializer, выдающего сведения об объявлении (код далее). Занесем его код в модуль serializers.py пакета приложения.



Контроллер, выдающий сведения о выбранном объявлении, назовем bbdetailview и реализуем в виде класса, производного от класса retrieveapiview. Его код, весьма компактный, приведен:



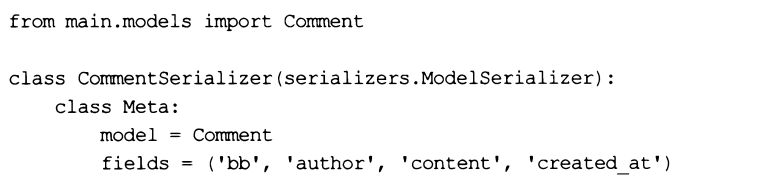
Добавим в список маршрутов уровня приложения маршрут, который укажет на наш новый контроллер:



Сохраним код и попытаемся получить сведения об объявлении с ключом 1, для чего выполним переход по интернет-адресу http://localhost:8000/api/bbs/l/. Далее запросим сведения о паре других объявлений.

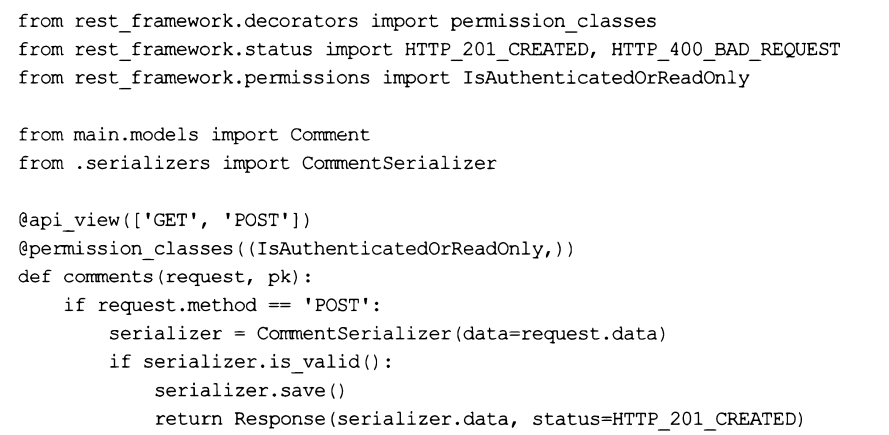
**Вывод и добавление комментариев**

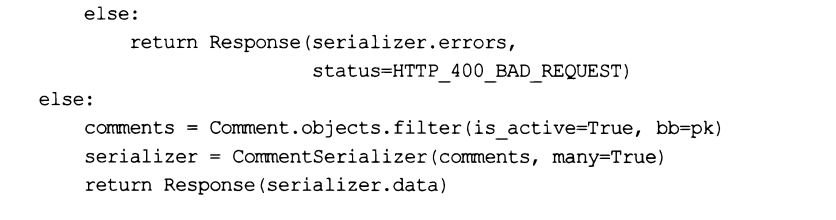
Код сериализатора Commentserializer, который будет отправлять список комментариев и добавлять новый комментарий, приведен:



Он отправит клиенту ключ объявления, с которым связан комментарий, имя автора, содержимое и временную отметку создания комментария.

Код контроллера-функции comments (), выдающего список комментариев и добавляющего новый комментарий, приведен:





Ранее мы условились, что разрешим добавлять комментарии только зарегистрированным пользователям, а просматривать их, напротив, позволим всем. Поэтому мы пометили контроллер декоратором permission ciasses о, в котором указали класс разграничения доступа isauthenticatedorreadonly.

Маршрут, который укажет на новый контроллер и который мы поместим в список уровня приложения, будет выглядеть так:



Проверим, удастся ли получить список комментариев, оставленных под объявлением с ключом 1. Какой интернет-адрес при этом нужно набирать в веб-обозревателе, сообразим самостоятельно.