

Материалы занятия

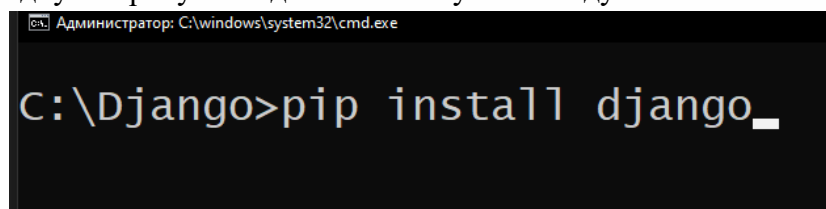
Курс: Web-разработка

Дисциплина: Создание web-приложений с использованием
фреймворка Django

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

1. УСТАНОВКА ФРЕЙМВОРКА

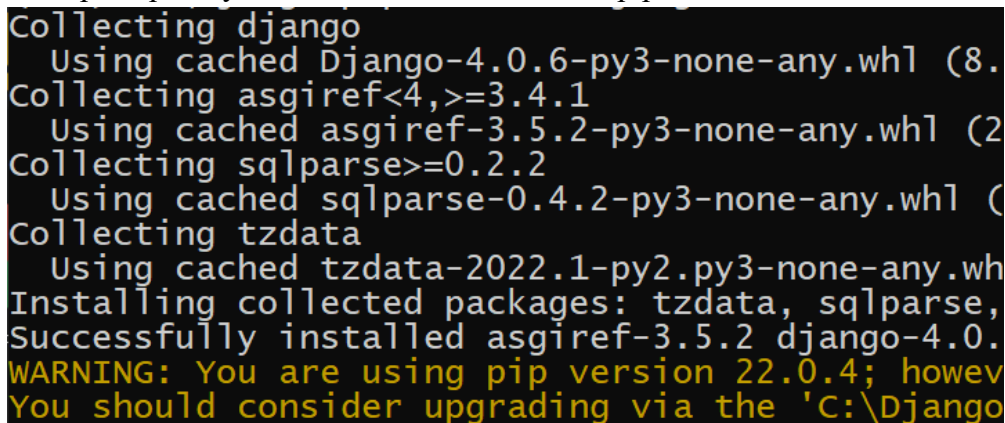
Установить Django проще всего посредством утилиты `pip`, поставляемой в составе Python и выполняющей установку дополнительных библиотек из интернет-репозитория `pypi`. Запустим командную строку и введем в ней такую команду:



```
Администратор: C:\windows\system32\cmd.exe  
C:\Django>pip install django_
```

Помимо Django, будут установлены библиотеки `pytz` (обрабатывает временные отметки — комбинации даты и времени), `sqlparse` (служит для разбора SQL-кода) и `asgiref` (реализует интерфейс ASGI, посредством которого эксплуатационный веб-сервер взаимодействует с сайтом, написанным на Django, и который мы рассмотрим позже), необходимые фреймворку для работы. Не удаляйте эти библиотеки!

Спустя некоторое время установка закончится, о чем `pip` нам обязательно сообщит:



```
Collecting django  
Using cached Django-4.0.6-py3-none-any.whl (8.  
Collecting asgiref<4,>=3.4.1  
Using cached asgiref-3.5.2-py3-none-any.whl (2  
Collecting sqlparse>=0.2.2  
Using cached sqlparse-0.4.2-py3-none-any.whl (C  
Collecting tzdata  
Using cached tzdata-2022.1-py2.py3-none-any.wh  
Installing collected packages: tzdata, sqlparse,  
Successfully installed asgiref-3.5.2 django-4.0.  
WARNING: You are using pip version 22.0.4; howev  
You should consider upgrading via the 'C:\Django
```

Теперь мы можем начинать разработку нашего первого веб-сайта.

2. ПРОЕКТ DJANGO

Первое, что нам нужно сделать, — создать новый проект. Проектом называется совокупность всего программного кода, составляющего разрабатываемый сайт. Физически он представляет собой папку, в которой находятся папки и файлы с исходным кодом (назовем ее папкой проекта).

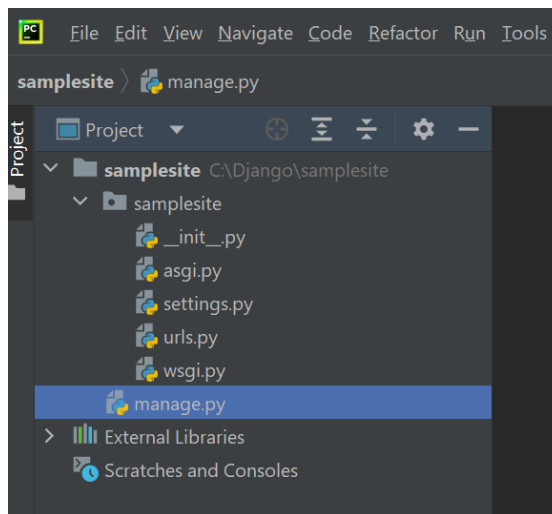
Создадим новый, пока еще пустой проект Django, которому дадим имя `samplesite`. Для этого в запущенной ранее командной строке перейдем в папку, в которой должна находиться папка проекта, и отдадим команду:

```
Администратор: C:\windows\system32\cmd.exe

C:\Django>django-admin startproject samplesite
```

Утилита `django-admin` служит для выполнения разнообразных административных задач. В частности, команда `startproject` указывает ей создать новый проект с именем, записанным после этой команды.

"Внешняя" папка `samplesite` — это, как нетрудно догадаться, и есть папка проекта. Как видим, ее имя совпадает с именем проекта, записанным в вызове утилиты `django-admin`. А содержимое этой папки таково:



- `Manage.py` — программный файл с кодом одноименной служебной утилиты, выполняющей различные действия над проектом;
- "внутренняя" папка `samplesite` — пакет языка Python, содержащий модули, которые относятся к проекту целиком и задают его конфигурацию (в частности, ключевые настройки). Название этого пакета совпадает с названием проекта, и менять его не стоит — в противном случае придется вносить в код обширные правки.

Пакет конфигурации включает в себя такие модули:

- `init.py` — пустой файл, сообщающий Python, что папка, в которой он находится, является полноценным пакетом;
- `settings.py` — модуль с настройками самого проекта. Включает описание конфигурации базы данных проекта, пути ключевых папок, важные параметры, связанные с безопасностью, и пр.;
- `urls.py` — модуль с маршрутами уровня проекта (о них мы поговорим позже);
- `wsgi.py` — модуль, связывающий проект с веб-сервером посредством интерфейса WSGI;
- `asgi.py` (начиная с Django 3.0) — модуль, связывающий проект с веб-сервером через интерфейс ASGI.

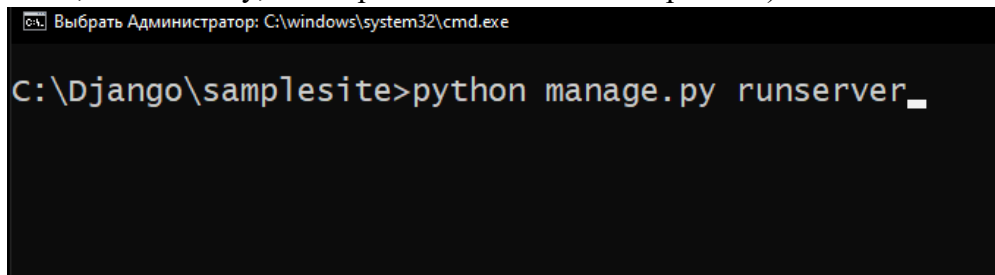
Модули `wsgi.py` и `asgi.py` используются при публикации готового сайта в Интернете.

Еще раз отметим, что пакет конфигурации хранит настройки, относящиеся к самому проекту и влияющие на все приложения, которые входят в состав этого проекта (о приложениях мы поведем разговор очень скоро).

Проект Django мы можем поместить в любое место файловой системы компьютера. Мы также можем переименовать папку проекта. В результате всего этого проект не потеряет своей работоспособности.

3. ОТЛАДОЧНЫЙ ВЕБ-СЕРВЕР DJANGO

В состав Django входит отладочный веб-сервер, написанный на самом языке Python. Чтобы запустить его, следует в командной строке перейти непосредственно в папку проекта (именно в нее, а не в папку, в которой находится папка проекта!) И отдать команду:



```
Выбрать Администратор: C:\windows\system32\cmd.exe  
C:\Django\samplesite>python manage.py runserver_
```

Здесь мы пользуемся уже утилитой `manage.py`, сгенерированной программой `django-admin` при создании проекта. Команда `runserver`, которую мы записали после имени этой утилиты, как раз и запускает отладочный веб-сервер.

Последний выдаст сообщение о том, что сайт успешно запущен (конечно, если его код не содержит ошибок) и доступен по интернет-адресу `http://127.0.0.1:8000/` (или `http://localhost:8000/`). Как видим, отладочный сервер по умолчанию работает через TCP-порт 8000 (впрочем, при необходимости можно использовать другой порт).

Запустим веб-обозреватель и наберем в нем один из интернет-адресов нашего сайта. Мы увидим информационную страничку, предоставленную самим Django и сообщающую, что сайт, хоть еще и "пуст", но в целом работает.

Для остановки отладочного веб-сервера достаточно нажать комбинацию клавиш `<Ctrl>+<Break>`.

4. ПРИЛОЖЕНИЯ

Приложение в терминологии Django — это отдельный фрагмент функциональности разрабатываемого сайта, более или менее независимый от других таких же фрагментов и входящий в состав проекта. Приложение может реализовывать работу всего сайта, его раздела или же какой-либо внутренней подсистемы сайта, используемой другими приложениями.

Любое приложение представляется обычным пакетом Python (пакет приложения), в котором содержатся модули с программным кодом. Этот пакет находится в папке проекта — там же, где располагается пакет конфигурации. Имя пакета приложения станет именем самого приложения.

Нам нужно сделать так, чтобы наш сайт выводил перечень объявлений, оставленных посетителями. Для этого мы создадим новое приложение, которое незатейливо назовем `bboard`.

Новое приложение создается следующим образом. Сначала остановим отладочный веб-сервер. В командной строке проверим, находимся ли мы в папке проекта, и наберем команду:

```
Администратор: C:\windows\system32\cmd.exe

c:\Django\samplesite>python manage.py startapp bboard_
```

Команда `startapp` утилиты `manage.py` запускает создание нового "пустого" приложения, имя которого указано после этой команды.

Посмотрим, что создала утилита `manage.py`. Прежде всего это папка `bboard`, формирующая одноименный пакет приложения и расположенная в папке проекта. В ней находятся следующие папки и файлы:

- `Migrations` — папка вложенного пакета, в котором будут храниться сгенерированные Django миграции (о них разговор пойдет позже). Пока что в папке находится лишь пустой файл `init.py`, помечающий ее как полноценный пакет Python;
- `Init.py` — пустой файл, сигнализирующий исполняющей среде Python, что эта папка — пакет;
- `Admin.py` — модуль административных настроек и классов-редакторов;
- `Apps.py` — модуль с настройками приложения;
- `Models.py` — модуль с моделями;
- `Tests.py` — модуль с тестирующими процедурами;
- `Views.py` — модуль с контроллерами.

Зарегистрируем только что созданное приложение в проекте. Найдем в пакете конфигурации файл `settings.py` (о котором уже упоминалось ранее), откроем его в текстовом редакторе и отыщем следующий фрагмент кода:

```
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40 ]
41
```

Список, хранящийся в переменной `INSTALLED_APPS`, перечисляет все приложения, зарегистрированные в проекте и участвующие в его работе. Изначально в этом списке присутствуют только стандартные приложения, входящие в состав Django и реализующие различные встроенные подсистемы фреймворка. Так, приложение `django.contrib.auth`

реализует подсистему разграничения доступа, а приложение `django.contrib.sessions` — подсистему, обслуживающую серверные сессии.

В этой "теплой" компании явно не хватает нашего приложения `bboard`. Добавим его, включив в список новый элемент:

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'bboard.apps.BboardConfig'
]
```

Мы указали строку с путем к классу `bboardconfig`, описывающему конфигурацию приложения и объявленному в модуле `apps.py` пакета приложения `bboard`.

Сохраним и закроем файл `settings.py`. Но запускать отладочный веб-сервер пока не станем. Вместо этого сразу же напишем первый в нашей практике Django-программирования контроллер.

5. КОНТРОЛЛЕРЫ

Контроллер Django — это код, запускаемый при обращении по интернет-адресу определенного формата и в ответ, выводящий на экран определенную веб-страницу.

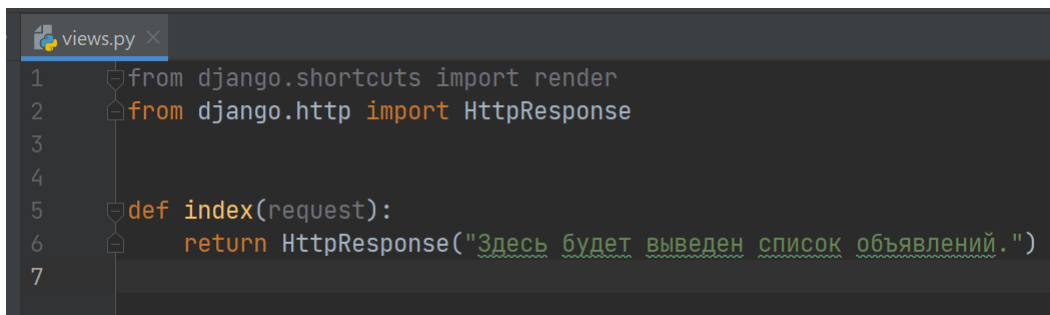
В документации по Django используется термин "view" (вид, или представление).

Контроллер Django может представлять собой как функцию (контроллер-функция), так и класс (контроллер-класс). Первые более универсальны, но зачастую трудоемки в программировании, вторые позволяют выполнить типовые задачи, наподобие вывода списка каких-либо позиций, минимумом кода.

Для хранения кода контроллеров изначально предназначается модуль `views.py`, создаваемый в каждом пакете приложения. Однако ничто не мешает нам поместить контроллеры в другие модули.

Напишем контроллер, который будет выводить... Нет, не список объявлений — этого списка у нас пока нет (у нас и базы данных-то нет), а пока только текст, сообщающий, что будущие посетители сайта со временем увидят на этой страничке список объявлений. Это будет контроллер-функция.

Откроем модуль `views.py` пакета приложения `bboard`, удалим имеющийся там небольшой код и заменим его кодом:



```

1  from django.shortcuts import render
2  from django.http import HttpResponseRedirect
3
4
5  def index(request):
6      return HttpResponseRedirect("Здесь будет выведен список объявлений.")
7

```

Наш контроллер — это, собственно, функция `index()`. Единственное, что она делает, — отправляет клиенту текстовое сообщение: Здесь будет выведен список объявлений. Но это только пока.

Любой контроллер-функция в качестве единственного обязательного параметра принимает экземпляр класса `HttpRequest`, хранящий различные сведения о полученном запросе: запрашиваемый интернет-адрес, данные, полученные от посетителя, служебную информацию от самого веб-обозревателя и пр. По традиции этот параметр называется `request`. В нашем случае мы его никак не используем.

В теле функции мы создаем экземпляр класса `HttpResponse` (он объявлен в модуле `django.http`), который будет представлять ответ, отправляемый клиенту. Содержимое этого ответа — собственно текстовое сообщение — указываем единственным параметром конструктора этого класса. Готовый экземпляр класса возвращаем в качестве результата.

Что ж, теперь мы с гордостью можем считать себя программистами — поскольку уже самостоятельно написали какой-никакой программный код. Осталось запустить отладочный веб-сервер, набрать в любимом веб-обозревателе адрес вида `http://localhost:8000/bboard/` и посмотреть, что получится.

6. МАРШРУТЫ И МАРШРУТИЗАТОР

Мы ни где явно не связали интернет-адрес с контроллером. И как это сделать?

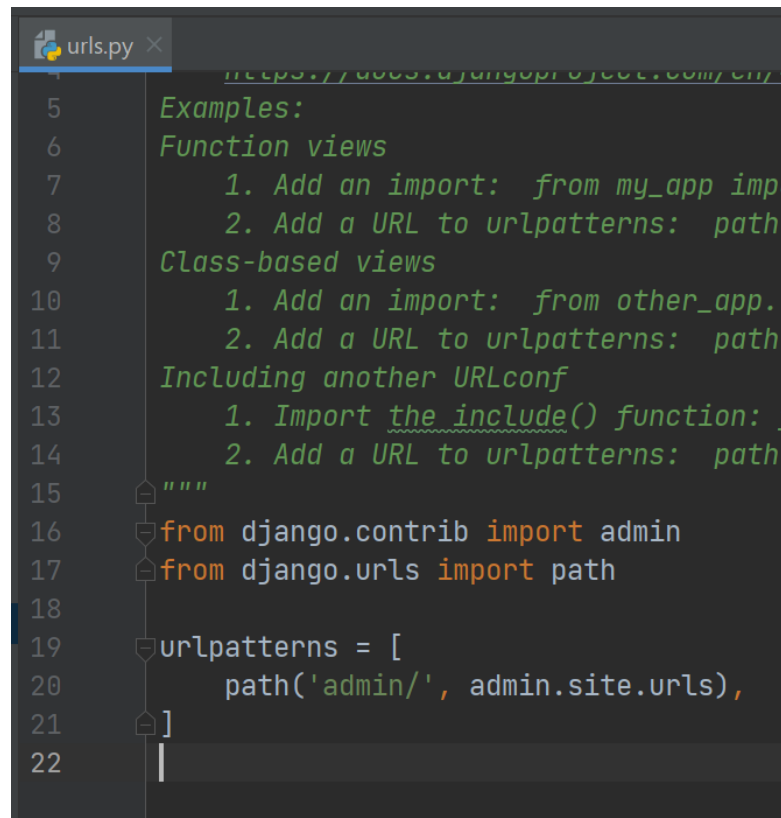
Сделать это очень просто. Нужно всего лишь:

- Объявить связь пути определенного формата (шаблонного пути) с определенным контроллером — иначе говоря, маршрут.
- Путь — это часть интернет-адреса, находящаяся между адресом хоста и набором GET-параметров (например, интернет-адрес `http://localhost:8000/bboard/ 34/edit/` содержит путь `bboard/34/edit/`).
- Шаблонный путь должен завершаться символом слеша. Напротив, начальный слеш в нем не ставится;
- Оформить все объявленные нами маршруты в виде списка маршрутов;
- Оформить маршруты в строго определенном формате, чтобы подсистема маршрутизатора смогла использовать готовый список в работе.

При поступлении любого запроса от клиента Django выделяет из запрашиваемого интернет-адреса путь, который передает маршрутизатору. Последний последовательно сравнивает его с шаблонными путями из списка маршрутов. Как только будет найдено совпадение, маршрутизатор передает управление контроллеру, связанному с совпавшим шаблонным путем.

Чтобы при запросе по интернет-адресу `http://localhost:8000/bboard/` запускался только что написанный нами контроллер `index()`, нам нужно связать таковой с шаблонным путем `bboard/`.

Знакомясь с проектом, мы заметили хранящийся в пакете конфигурации модуль `urls.py`, в котором записываются маршруты уровня проекта. Откроем этот модуль в текстовом редакторе и посмотрим, что он содержит



```
1  # Example: http://docs.djangoproject.com/en/1.10.1/topics/http/urls/
2
3  Examples:
4
5  Function views
6      1. Add an import: from my_app import views
7      2. Add a URL to urlpatterns: path('', views.home, name='home')
8
9  Class-based views
10     1. Add an import: from other_app.views import HomeView
11     2. Add a URL to urlpatterns: path('', HomeView.as_view(), name='example')
12
13 Including another URLconf
14     1. Import the include() function: from django.urls import include
15     2. Add a URL to urlpatterns: path('', include('other_app.urls'), name='example')
16
17 """
18 from django.contrib import admin
19 from django.urls import path
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23 ]
```

Список маршрутов, оформленный в виде обычного списка Python, присваивается переменной `urlpatterns`. Каждый элемент списка маршрутов (т. Е. Каждый маршрут) должен представляться в виде результата, возвращаемого функцией `path()` из модуля `django.urls`. Последняя в качестве параметров принимает строку с шаблонным путем и ссылку на контроллер-функцию.

В качестве второго параметра функции `path()` также можно передать список маршрутов уровня приложения.

Сейчас добавим в список новый маршрут, связывающий шаблонный путь `bboard/` и контроллер-функцию `index()`.


```
urls.py
7      1. Add an import: from my_app imp
8      2. Add a URL to urlpatterns: path
9      Class-based views
10     1. Add an import: from other_app.
11     2. Add a URL to urlpatterns: path
12     Including another URLconf
13     1. Import the include() function:
14     2. Add a URL to urlpatterns: path
15     """
16     from django.contrib import admin
17     from django.urls import path
18     from bboard.views import index
19
20     urlpatterns = [
21         path('bboard/', index),
22         path('admin/', admin.site.urls),
23     ]
24
```

Сохраним исправленный файл, запустим отладочный веб-сервер и наберем в веб-обозревателе интернет-адрес `http://localhost:8000/bboard/`. Мы увидим текстовое сообщение, сгенерированное нашим контроллером.



Здесь будет выведен список объявлений.

Что ж, наши первые контроллер и маршрут работают, и по этому поводу можно порадоваться. Но лишь до поры до времени. Как только мы начнем создавать сложные сайты, состоящие из нескольких приложений, количество маршрутов в списке вырастет до таких размеров, что мы просто запутаемся в них. Поэтому создатели Django настоятельно рекомендуют применять для формирования списков маршрутов другой подход, о котором мы сейчас поговорим.

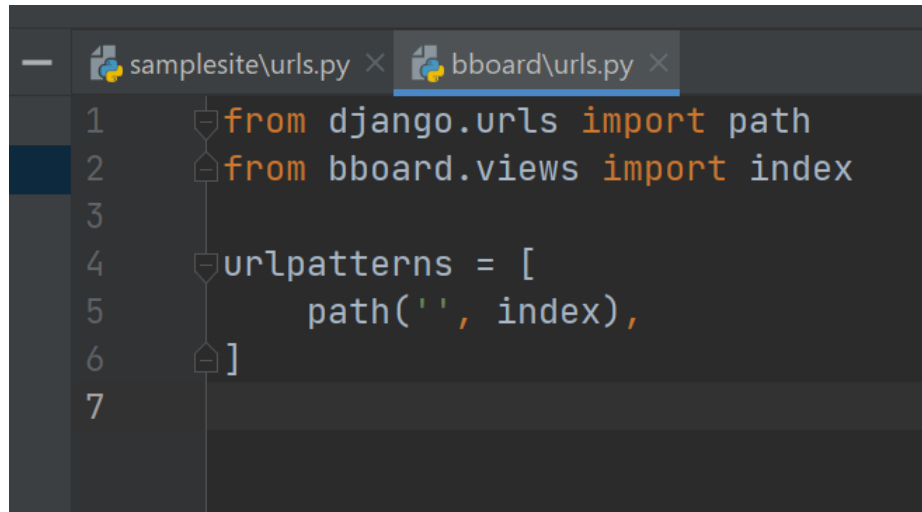
Маршрутизатор Django при просмотре списка маршрутов не требует, чтобы путь, полученный из клиентского запроса (реальный), и шаблонный путь, записанный в очередном маршруте, совпадали полностью. Достаточно лишь того факта, что шаблонный путь совпадает с началом реального.

Как было сказано ранее, функция `path()` позволяет указать во втором параметре вместо ссылки на контроллер-функцию другой, вложенный, список маршрутов. В таком случае маршрутизатор, найдя совпадение, удалит из реального пути его начальную часть (префикс), совпавшую с шаблонным путем, и приступит к просмотру маршрутов из вложенного списка, используя для сравнения реальный путь уже без префикса.

Исходя из всего этого, мы можем создать иерархию списков маршрутов. В списке из пакета конфигурации (списке маршрутов уровня проекта) запишем маршруты, которые указывают на вложенные списки маршрутов, принадлежащие отдельным приложениям

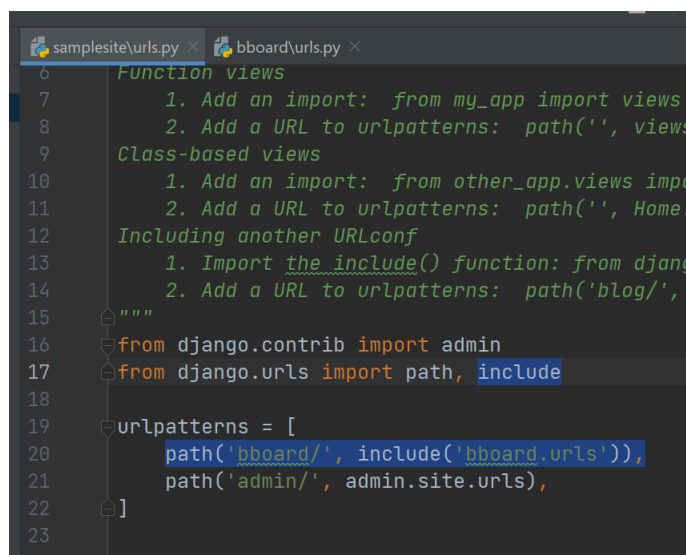
(списки маршрутов уровня приложения). А в последних непосредственно запишем нужные контроллеры.

Начнем со списка маршрутов уровня приложения bboard. Создадим в пакете этого приложения (т. е. в папке bboard) файл urls.py и занесем в него код:



```
1 from django.urls import path
2 from bboard.views import index
3
4 urlpatterns = [
5     path('', index),
6 ]
7
```

Наконец, исправим код модуля urls.py из пакета конфигурации:



```
6 Function views
7     1. Add an import: from my_app import views
8     2. Add a URL to urlpatterns: path('', views)
9 Class-based views
10    1. Add an import: from other_app.views import
11    2. Add a URL to urlpatterns: path('', Home)
12 Including another URLconf
13    1. Import the include() function: from django
14    2. Add a URL to urlpatterns: path('blog/',
15
16 from django.contrib import admin
17 from django.urls import path, include
18
19 urlpatterns = [
20     path('bboard/', include('bboard.urls')),
21     path('admin/', admin.site.urls),
22 ]
23
```

Вложенный список маршрутов, указываемый во втором параметре функции path(), должен представлять собой результат, возвращенный функцией include() из модуля django.urls. В качестве единственного параметра эта функция принимает строку с путем к модулю, где записан вложенный список маршрутов.

Как только наш сайт получит запрос с интернет-адресом `http://localhost:8000/ bboard/`, маршрутизатор обнаружит, что присутствующий в нем путь совпадает с шаблонным путем `bboard/`. Он удалит из реального пути префикс, соответствующий шаблонному пути, и получит новый путь — пустую строку. Этот путь совпадет с единственным маршрутом из вложенного списка в результате чего запустится записанный

В этом маршруте контроллер-функция `index()`, и на экране появится уже знакомое нам текстовое сообщение.

Поскольку зашла речь о вложенных списках маршрутов, давайте посмотрим на выражение, создающее второй маршрут из списка уровня проекта:

`Path(«admin/ », admin.site.urls),`

Этот маршрут связывает шаблонный путь `admin/` со списком маршрутов из свойства `urls` экземпляра класса `adminsite`, который хранится в переменной `site` и представляет текущий административный веб-сайт Django. Следовательно, набрав интернет-адрес `http://localhost:8000/admin/`, мы попадем на этот административный сайт (разговор о нем будет позже).