# Глава 1. Введение в Django

## Что такое Django

Django - это фреймворк для создания веб-приложений с помощью языка программирования Python.

Django был создан в 2005 году, когда веб-разработчики из газеты Lawrence Journal-World стали использовать Python в качестве языка для создания веб-сайтов. А в 2008 году вышел публичный первый релиз фреймворка. На сегодняшний день он продолжает развиваться. Так, текущей версией фреймворка на момент написания этой статьи является версия 4.0, которая вышла в декабре 2021 года. Ну и также постоянно выходят подверсии.

Django довольно популярен. Он используется на многих сайтах, в том числе таких, как Pinterest, PBS, Instagram, BitBucket, Washington Times, Mozilla и многих других.

Фреймворк является бесплатным. Он развивается как open source, его исходный код открыт, его можно найти репозитории на [githube](https://github.com/django/django).

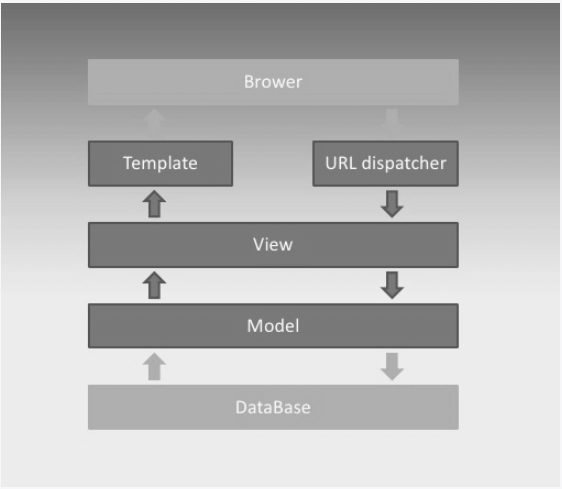
На Django можно создавать широкий диапазон веб-приложений: от небольших персональных сайтов до высоконагруженных сложных веб-сервисов.

Django по умолчанию предлагает готовую функциональность для ряда распространенных задач, например, систему аутентификации, генерацию карт сайта и т.д., благодаря чему нам можно не изобретать велосипед и достаточно взять уже готовые компоненты.

В Django большое внимание уделяется безопасности, благодаря чему фреймворк помогает разработчикам избежать многих распространенных проблем в системе безопасности, например, sql-инъекций.

Фреймворк Django реализует архитектурный паттерн Model-View-Template или сокращенно MVT, который по факту является модификацией распростаненного в веб-программировании паттерна MVC (Model=-View-Controller).

Схематично мы можем представить архитектуру MVT в Django следующим образом:



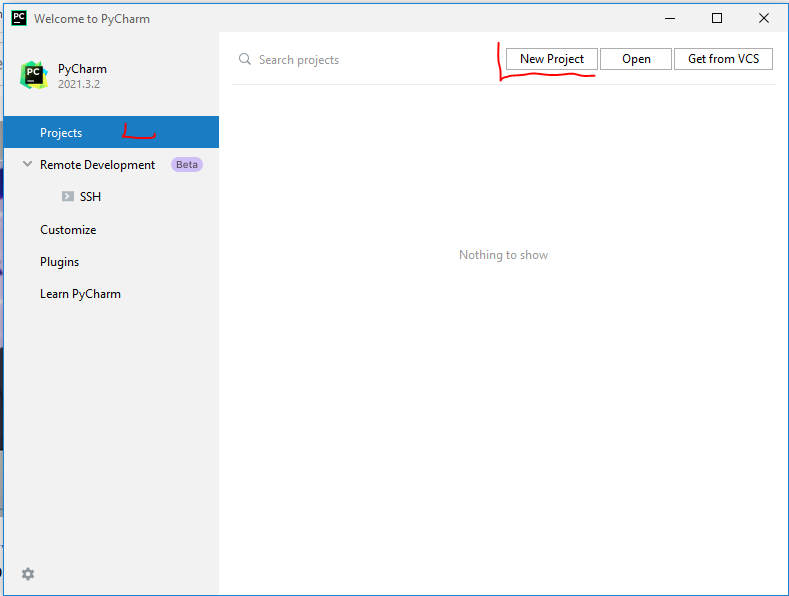
Основные элементы паттерна:

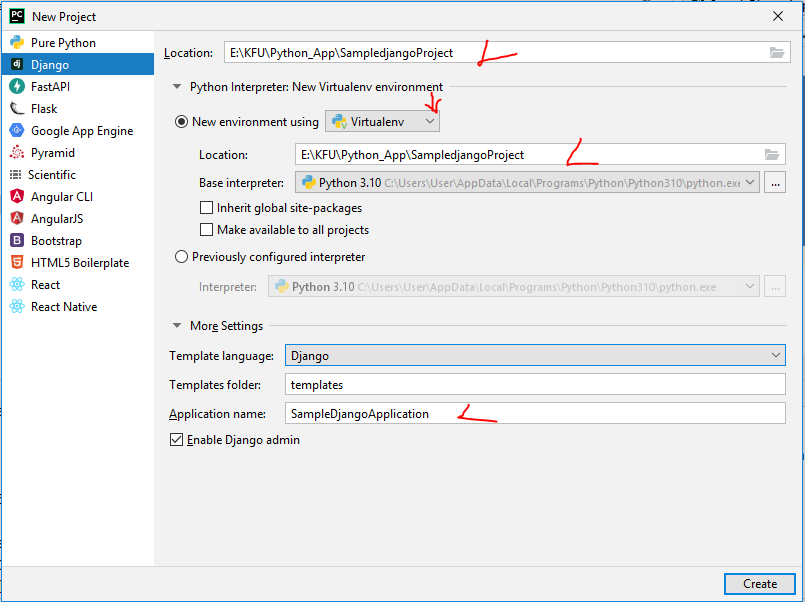
* URL dispatcher: при получение запроса на основании запрошенного адреса URL определяет, какой ресурс должен обрабатывать данный запрос.
* View: получает запрос, обрабатывает его и отправляет в ответ пользователю некоторый ответ. Если для обработки запроса необходимо обращение к модели и базе данных, то View взаимодействует с ними. Для создания ответа может применять Template или шаблоны. В архитектуре MVC этому компоненту соответствуют контроллеры (но не представления).
* Model: описывает данные, используемые в приложении. Отдельные классы, как правило, соответствуют таблицам в базе данных.
* Template: представляет логику представления в виде сгенерированной разметки html. В MVC этому компоненту соответствует View, то есть представления.

Когда к приложению приходит запрос, то URL dispatcher определяет, с каким ресурсом сопоставляется данный запрос и передает этот запрос выбранному ресурсу. Ресурс фактически представляет функцию или View, который получает запрос и определенным образом обрабатывает его. В процессе обработки View может обращаться к моделям и базе данных, получать из нее данные, или, наоборот, сохранять в нее данные. Результат обработки запроса отправляется обратно, и этот результат пользователь видит в своем браузере. Как правило, результат обработки запроса представляет сгенерированный html-код, для генерации которого применяются шаблоны (Template).

## Создание первого проекта

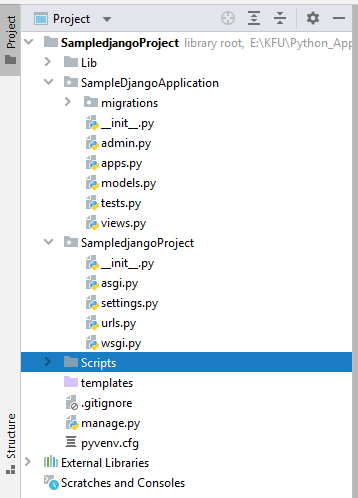
1. В первую очередь скачать и установить Pycharm с <https://www.jetbrains.com/ru-ru/pycharm/download/#section=windows>
2. Откройте Pycharm и нажмите «Новый проект».



В результате откроется следующее окно:

После выбора и ввода необходимых параметров нажмите кнопку «Создать».

В итоге наш проект открывается следующим образом:

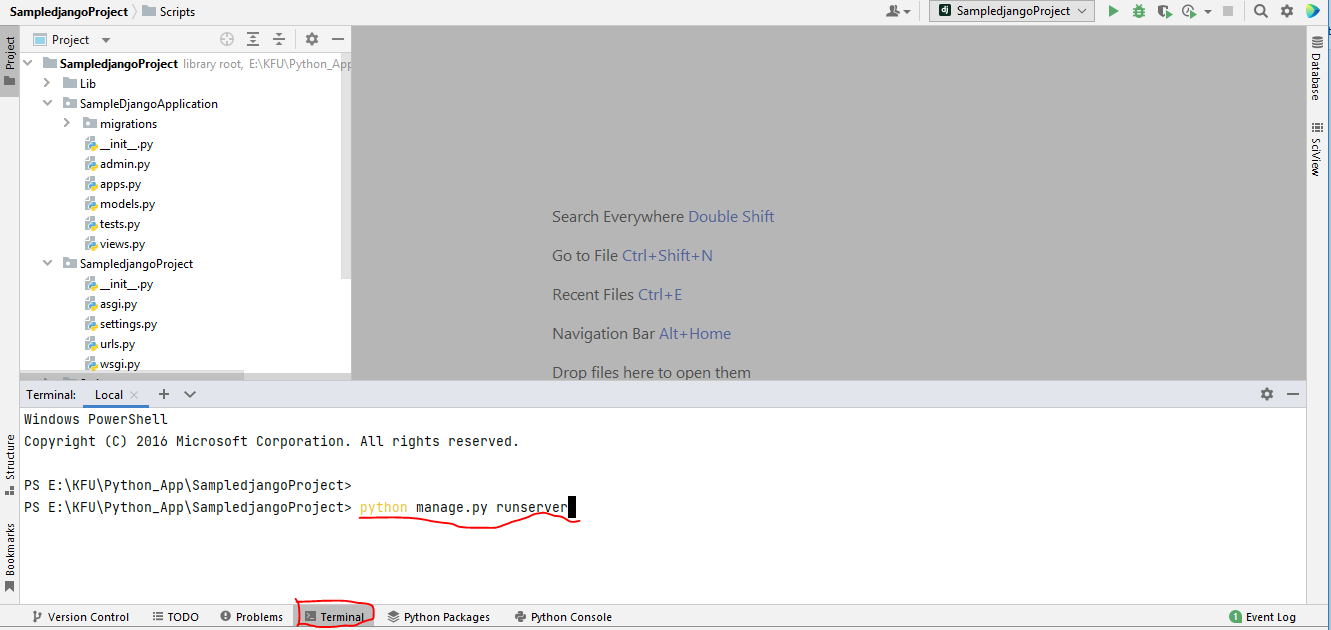


Проект будет состоять из следующих элементов:

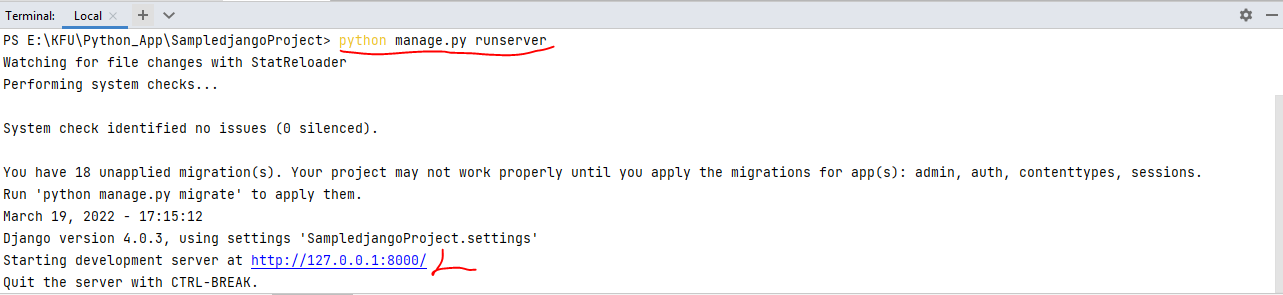
* manage.py: выполняет различные команды проекта, например, запускает приложение
* \_\_init\_\_.py: данный файл указывает, что папка, в которой он находится, будет рассматриваться как модуль. Это стандартный файл для Python.
* settings.py: содержит настройки конфигурации проекта
* urls.py: содержит шаблоны URL-адресов, по сути определяет систему маршрутизации проекта
* wsgi.py: содержит свойства конфигурации WSGI (Web Server Gateway Inerface). Он используется при развертывании проекта.
* asgi.py: название файла представляет сокращение от Asynchronous Server Gateway Interface и расширяет возможности WSGI, добавляя поддержку для взавимодействия между асинхронными веб-серверами и приложениями.

Запустим проект на выполнение. Для этого с помощью команды cd перейдем в консоли к папке проекта. И затем для запуска проекта выполним следующую команду:

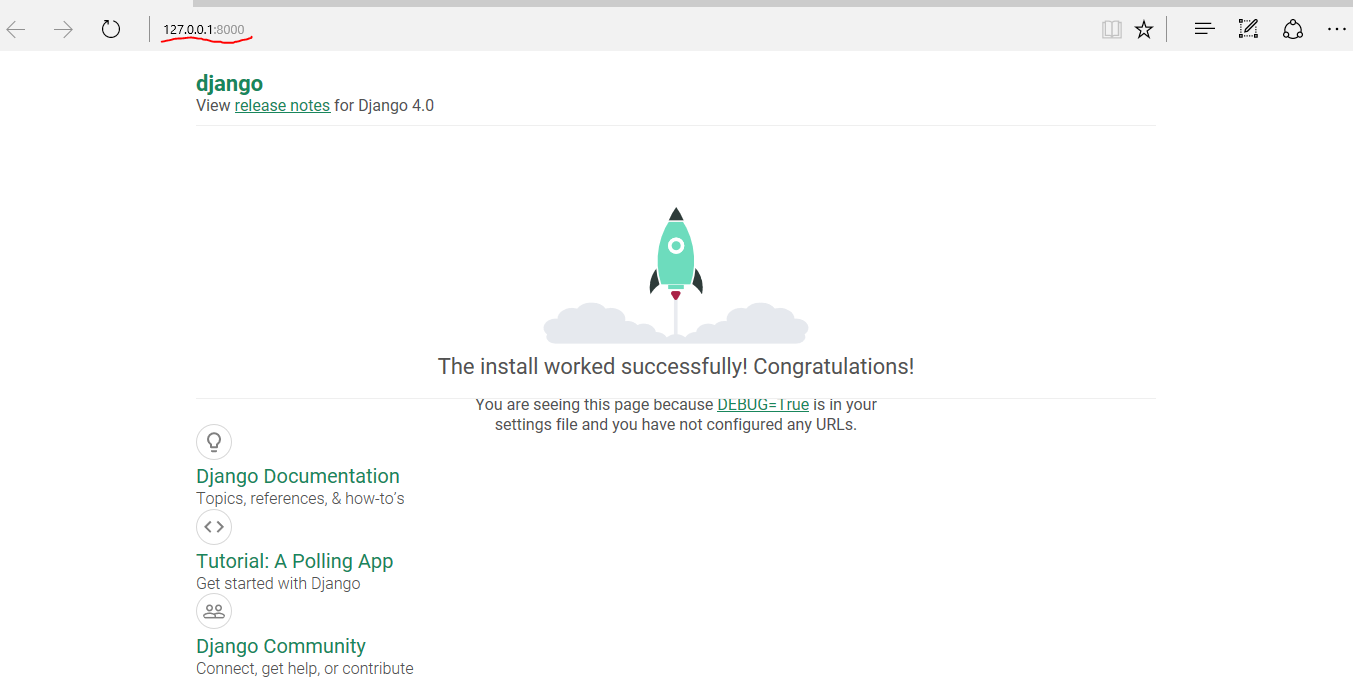
|  |
| --- |
| python manage.py runserver |



После нажатия клавиши Enter появится следующая форма:

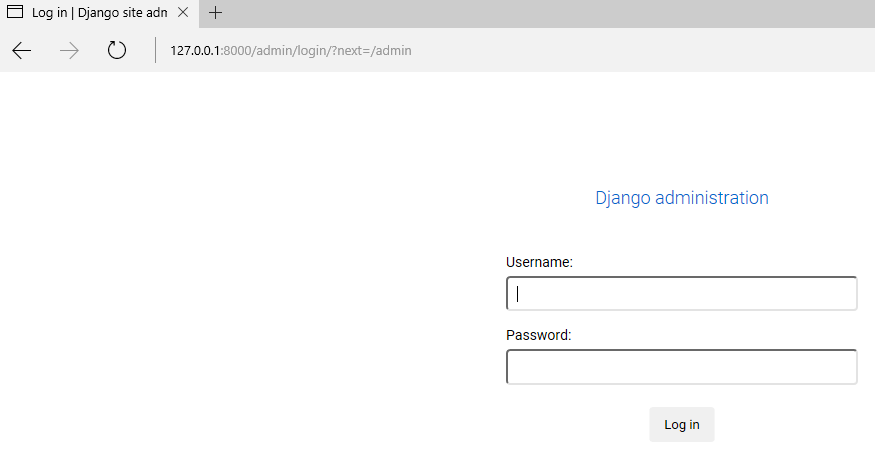


После запуска проекта в консоли мы увидим адрес, по которому запущен проект. Как правило, это адрес http://127.0.0.1:8000/. Откроем любой веб-браузер и введем данный адрес в адресную строку браузера. И нам откроется содержимое по умолчанию.



## Настройка входа администратора

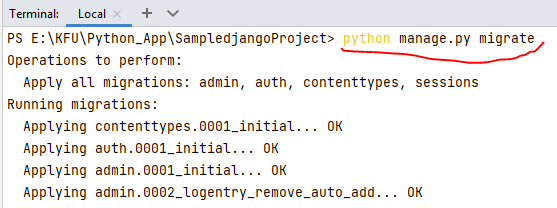
По умолчанию Django поставляется с готовой панелью администратора, которую вы можете использовать. Он находится по адресу http://127.0.0.1:8000/admin и выглядит примерно так:



Прежде чем мы сможем использовать его, нам нужно добавить пользователя-администратора. Для этого вам нужно остановить сервер приложений Django, используя Ctrl + c в командной строке. Создайте базу данных для вашего проекта Django с помощью следующей команды:

|  |
| --- |
| python manage.py migrate |

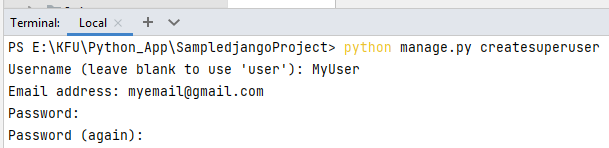
Ваша командная строка должна выглядеть примерно так:



Теперь вы можете запустить команду createsuperuser, чтобы создать нового пользователя для вашего приложения Django. Вот команда для него:

|  |
| --- |
| python manage.py createsuperuser |

В командной строке вы можете ввести свое имя пользователя, адрес электронной почты, а затем пароль. Так как это на вашем локальном компьютере, вы можете использовать слабый пароль, который выдаст слишком распространенное сообщение о пароле. Вы можете обойти это, набрав y при появлении запроса. Как только это будет завершено, вы успешно создадите суперпользователя. Сообщение.



Теперь вы можете снова запустить сервер, используя python manage.py runserver. Обновите браузер, и вы сможете войти в систему, используя свои учетные данные. Вот как выглядит экран зарегистрированного администратора:

**Создание первого приложения**

Веб-приложение или проект Django состоит из отдельных приложений. Вместе они образуют полноценное веб-приложение. Каждое приложение представляет какую-то определенную функциональность или группу функциональностей. Один проект может включать множество приложений. Это позволяет выделить группу задач в отдельный модуль и разрабатывать их относительно независимо от других. Кроме того, мы можем переносить приложение из одного проекта в другой независимо от другой функциональности проекта.

При создании проекта он уже содержит несколько приложений по умолчанию.

* django.contrib.admin
* django.contrib.auth
* django.contrib.contenttypes
* django.contrib.sessions
* django.contrib.messages
* django.contrib.staticfiles

Откройте файл settings.py, который имеет следующий формат:

|  |
| --- |
| *""" Настройки Django для проекта SampledjangoProject. Сгенерировано «django-admin startproject» с использованием Django 4.0.3. Дополнительные сведения об этом файле см. https://docs.djangoproject.com/en/4.0/topics/settings/ Полный список настроек и их значений см. https://docs.djangoproject.com/en/4.0/ref/settings/ """* **from** pathlib **import** Path  *# Создайте пути внутри проекта следующим образом: BASE\_DIR / 'subdir'.* BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent  *# Настройки быстрого старта разработки - непригодны для продакшена # См. https://docs.djangoproject.com/en/4.0/howto/deployment/checklist/  # ПРЕДУПРЕЖДЕНИЕ О БЕЗОПАСНОСТИ: держите секретный ключ, используемый в производстве, в секрете!* SECRET\_KEY = **'django-insecure-!d@kl0#d!k+\*+73+3d8i!t@s\*ju31&28xw^@\*i\*6t(!%ut19\*h'** *# ПРЕДУПРЕЖДЕНИЕ О БЕЗОПАСНОСТИ: не запускайте программу с включенной отладкой!* DEBUG = **True** ALLOWED\_HOSTS = []  *# Определение приложения* INSTALLED\_APPS = [  **'django.contrib.admin'**,  **'django.contrib.auth'**,  **'django.contrib.contenttypes'**,  **'django.contrib.sessions'**,  **'django.contrib.messages'**,  **'django.contrib.staticfiles'**,  **'SampledjangoApplication.apps.SampledjangoapplicationConfig'**, ]  MIDDLEWARE = [  **'django.middleware.security.SecurityMiddleware'**,  **'django.contrib.sessions.middleware.SessionMiddleware'**,  **'django.middleware.common.CommonMiddleware'**,  **'django.middleware.csrf.CsrfViewMiddleware'**,  **'django.contrib.auth.middleware.AuthenticationMiddleware'**,  **'django.contrib.messages.middleware.MessageMiddleware'**,  **'django.middleware.clickjacking.XFrameOptionsMiddleware'**, ]  ROOT\_URLCONF = **'SampledjangoProject.urls'** TEMPLATES = [  {  **'BACKEND'**: **'django.template.backends.django.DjangoTemplates'**,  **'DIRS'**: [BASE\_DIR / **'templates'**],  **'APP\_DIRS'**: **True**,  **'OPTIONS'**: {  **'context\_processors'**: [  **'django.template.context\_processors.debug'**,  **'django.template.context\_processors.request'**,  **'django.contrib.auth.context\_processors.auth'**,  **'django.contrib.messages.context\_processors.messages'**,  ],  },  }, ]  WSGI\_APPLICATION = **'SampledjangoProject.wsgi.application'** *# Database # https://docs.djangoproject.com/en/4.0/ref/settings/#databases* DATABASES = {  **'default'**: {  **'ENGINE'**: **'django.db.backends.sqlite3'**,  **'NAME'**: BASE\_DIR / **'db.sqlite3'**,  } }  *# Проверка пароля # https://docs.djangoproject.com/en/4.0/ref/settings/#auth-password-validators* AUTH\_PASSWORD\_VALIDATORS = [  {  **'NAME'**: **'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator'**,  },  {  **'NAME'**: **'django.contrib.auth.password\_validation.MinimumLengthValidator'**,  },  {  **'NAME'**: **'django.contrib.auth.password\_validation.CommonPasswordValidator'**,  },  {  **'NAME'**: **'django.contrib.auth.password\_validation.NumericPasswordValidator'**,  }, ]  *# Интернационализация # https://docs.djangoproject.com/en/4.0/topics/i18n/* LANGUAGE\_CODE = **'en-us'** TIME\_ZONE = **'UTC'** USE\_I18N = **True** USE\_TZ = **True** *# Статические файлы (CSS, JavaScript, Images) # https://docs.djangoproject.com/en/4.0/howto/static-files/* STATIC\_URL = **'static/'** *# Тип поля первичного ключа по умолчанию # https://docs.djangoproject.com/en/4.0/ref/settings/#default-auto-field* DEFAULT\_AUTO\_FIELD = **'django.db.models.BigAutoField'** |

Список всех приложений можно найти в проекте в файле settings.py в переменной INSTALLED\_APPS:

|  |
| --- |
| INSTALLED\_APPS = [  **'django.contrib.admin'**,  **'django.contrib.auth'**,  **'django.contrib.contenttypes'**,  **'django.contrib.sessions'**,  **'django.contrib.messages'**,  **'django.contrib.staticfiles'**,  **'SampledjangoApplication.apps.SampledjangoapplicationConfig'**, ] |

Теперь определим какие-нибудь простейшие действия, которые будет выполнять данное приложение, например, отправлять в ответ пользователю строку "Hello World".

Для этого перейдем в проекте приложения SampledjangoApplication к файлу views.py, который по умолчанию должен выглядеть следующим образом:

|  |
| --- |
| **from** django.shortcuts **import** render *# Create your views here* |

Изменим код следующим образом:

|  |
| --- |
| **from** django.http **import** HttpResponse **def** index(request):  **return** HttpResponse(**"Hello KFU"**) |

В данном случае мы импортируем класс HttpResponse из стандартного пакета django.http. Затем определяется функция index(), которая в качестве параметра получает объект запроса request. Класс HttpResponse предназначен для создания ответа, который отправляется пользователю. И с помощью выражения return HttpResponse("Hello KFU") мы отправляем пользователю строку " Hello KFU"

Теперь также в основном проекте Django откроем файл urls.py, который позволяет сопоставить маршруты с представлениями, которые будут обрабатывать запрос по этим маршрутам. По умолчанию этот файл выглядит следующим образом:

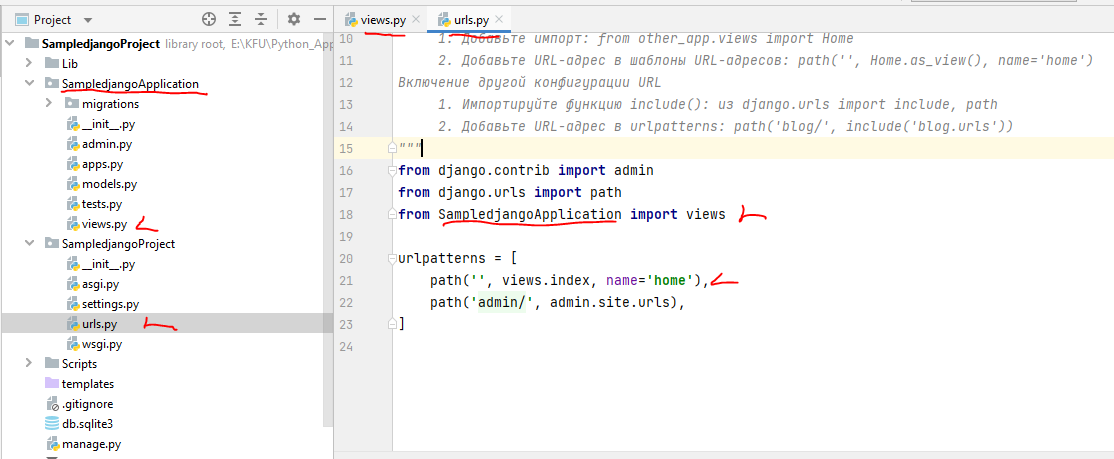
|  |
| --- |
| *"""Конфигурация URL-адреса SampledjangoProject  Список `urlpatterns` направляет URL-адреса в представления. Для получения дополнительной информации см.:  https://docs.djangoproject.com/en/4.0/topics/http/urls/ Примеры: Представления функций  1. Добавьте импорт: из представлений импорта my\_app  2. Добавьте URL-адрес в urlpatterns: path('', views.home, name='home') Представления на основе классов  1. Добавьте импорт: from other\_app.views import Home  2. Добавьте URL-адрес в шаблоны URL-адресов: path('', Home.as\_view(), name='home') Включение другой конфигурации URL  1. Импортируйте функцию include(): из django.urls import include, path  2. Добавьте URL-адрес в urlpatterns: path('blog/', include('blog.urls')) """* **from** django.contrib **import** admin **from** django.urls **import** path  urlpatterns = [  path(**'admin/'**, admin.site.urls), ] |

Первой строкой из модуля django.contrib импортируется класс AdminSite, который предоставляет возможности работы с интерфейсом администратора. Второй строкой из модуля django.urls импортируется функция path. Эта функция задает сопоставление определенного маршрута с функцией обработки. Так, в данном случае маршрут "admin/" будет обрабатываться методом admin.site.urls.

Но выше мы определили функцию index в файле views.py, который возвращает пользователю строку "Hello KFU". Поэтому изменим файл urls.py следующим образом:

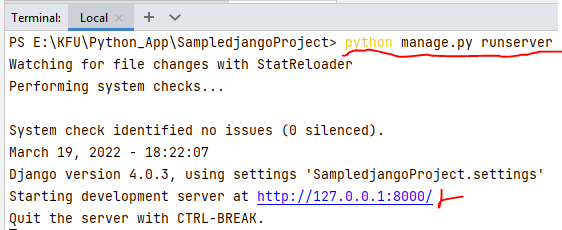
|  |
| --- |
| **from** django.contrib **import** admin **from** django.urls **import** path **from** SampledjangoApplication **import** views  urlpatterns = [  path(, views.index, name=**'home'**),  path(**'admin/'**, admin.site.urls), ] |

Чтобы использовать функцию views.index вначале импортируем модуль views. Затем определяем сопоставление маршрута ' ' и функции views.index и также дополнительно имя для маршрута: name='home'. По сути маршрут ' ' будет сопоставляться с запросом к корню приложения.



Теперь снова запустим приложение командой

|  |
| --- |
| python manage.py runserver |



И вновь перейдем в браузере по адресу http://127.0.0.1:8000/, и браузер нам отобразит строку "Hello KFU":



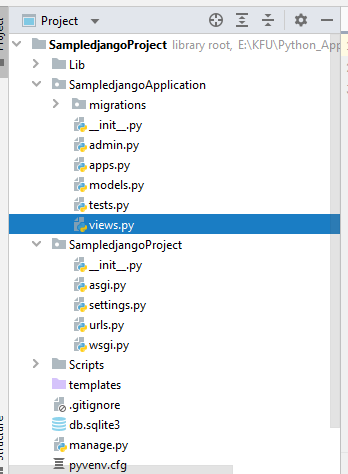
**Глава 2. Представления и маршрутизация**

**Обработка запроса**

Центральным моментом любого веб-приложения является обработка запроса, который отправляет пользователь. В Django за обработку запроса отвечают представления или views. По сути представления представляют функции обработки, которые принимают данные запроса в виде объекта request и генерируют некоторый результат, который затем отправляется пользователю.

По умолчанию представления размещаются в приложении в файле views.py.

Например, возьмем стандартный проект, который добавлено приложение (например, проект созданный в прошлой главе).



При создании нового проекта файл views.py имеет следующее содержимое:

|  |
| --- |
| **from** django.shortcuts **import** render *# Create your views here.* |

Данный код пока никак не обрабатывает запросы, он только импортирует функцию render(), которая может использоваться для обработки.

Генерировать результат можно различными способами. Один из них представляет использование класса HttpResponse из пакета django.http, который позволяет отправить текстовое содержимое.

Так, изменим файл views.py следующим образом:

|  |
| --- |
| **from** django.http **import** HttpResponse  **def** index(request):  **return** HttpResponse(**"<h2>Главная</h2>"**)  **def** about(request):  **return** HttpResponse(**"<h2>О сайте</h2>"**)  **def** contact(request):  **return** HttpResponse(**"<h2>Контакты</h2>"**) |

В данном случае определены три функции, которые будут обрабатывать запросы. Каждая функция принимает в качестве параметра объект request. Для генерации ответа в конструктор объекта HttpResponse передается некоторая строка. Это может быть в том числе и код html в виде строки.

Чтобы эти функции сопоставлялись с запросами, надо определить для них маршруты в проекте в файле urls.py. В частности, изменим этот файл следующим образом:

|  |
| --- |
| **from** django.contrib **import** admin **from** django.urls **import** path **from** SampledjangoApplication **import** views urlpatterns = [  path(**''**, views.index),  path(**'about'**, views.about),  path(**'contact'**, views.contact), ] |

Переменная urlpatterns определяет набор сопоставлений функций обработки с определенными строками запроса. Например, запрос к корню веб-сайта будет обрабатываться функцией index, запрос по адресу "about" будет обрабатываться функцией about, а запрос "contact" - функцией contact.

Запусим проект и обратимся по некоторым из этих адресов.

|  |  |
| --- | --- |
|  |  |

## Определение маршрутов и функции path и re\_path

В прошлой теме рассматривалось сопоставление адресов URL и функций, которые обрабатывают запросы по этим адресам. Например, у нас есть следующие функции в файле views.py:

|  |
| --- |
| **from** django.http **import** HttpResponse  **def** index(request):  **return** HttpResponse(**"<h2>Главная</h2>"**)  **def** about(request):  **return** HttpResponse(**"<h2>О сайте</h2>"**)  **def** contact(request):  **return** HttpResponse(**"<h2>Контакты</h2>"**) |

И в файле urls.py они сопоставляются с адресами URL с помощью функции path():

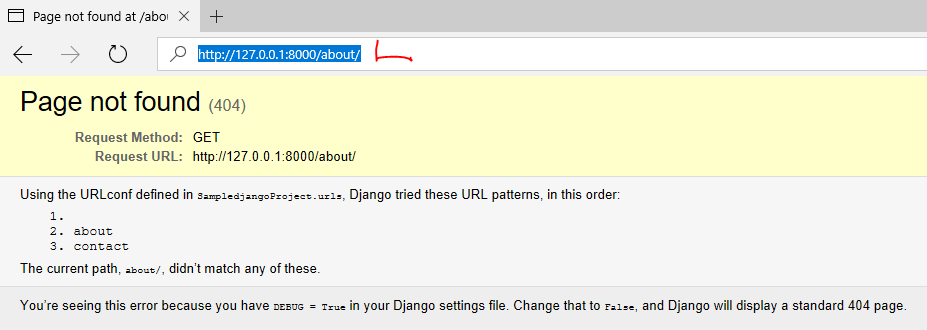
|  |
| --- |
| **from** django.urls **import** path **from** SampledjangoApplication **import** views  urlpatterns = [  path(**''**, views.index),  path(**'about'**, views.about),  path(**'contact'**, views.contact), ] |

Функция path() располагается в пакете django.urls и принимает два параметра: запрошенный адрес URL и функция, которая обрабатывает запрос по этому адресу. Дополнительно через третий параметр можно указать имя маршрута:

|  |
| --- |
| path(**''**, views.index, name=**'home'**) |

В данном случае маршрут будет называться "home".

Однако функция path ограничена по своему действию. Запрошенный путь должен в точности соответствовать указанному в маршруте адресу URL. Так, в примере выше, что функция views.about могла обрабатывать запрос, адрес должен быть в точности "about". Например, стоит нам указать слеш в конце: "about/" и django уже не сможет сопоставить путь с запросом.



**re\_path**

В качестве альтернативы для определения маршрутов мы можем использовать функцию re\_path(), которая также располагается в пакете django.urls. Ее преимущество состоит в том, что она позволяет задать адреса URL с помощью регулярных выражений.

Например, изменим определение файла urls.py следующим образом:

|  |
| --- |
| **from** django.urls **import** path **from** django.urls **import** re\_path **from** SampledjangoApplication **import** views  urlpatterns = [  path(**''**, views.index),  re\_path(**r'^about'**, views.about),  re\_path(**r'^contact'**, views.contact), ] |

Адрес в первом маршруте по-прежнему образуется с помощью функции path и указывает на корень веб-приложения.

Остальные два маршрута образуются с помощью функции re\_path(). Причем, поскольку определяется регуляное выражение, то перед строкой с шаблоном адреса URL ставится буква r. В самом шаблоне адреса можно использовать различные элементы синтаксиса регулярных выражений. В частности, выражение ^about указывает, что адрес должен начинаться с "about". Однако он необязательно в точности должен соответствовать строке "about", как это было в случае с функцией path.

Например, мы можем обратиться по любому адресу, главное чтобы он начинался с "about", и тогда подобный запрос будет обрабатываться функцией views.about.

**Очередность маршрутов**

Когда запрос приходит к приложению, то система проверяет соответствие запроса маршрутам по мере их определения: вначале сравнивается первый маршрут, если он не подходит, то сравнивается второй и так далее. Поэтому более общие маршруты должны определяться в последнюю очередь, а более конкретные маршруты должны идти в начале. Например:

|  |
| --- |
| **from** django.urls **import** path **from** django.urls **import** re\_path **from** SampledjangoApplication **import** views  urlpatterns = [  re\_path(**r'^about/contact/'**, views.contact),  re\_path(**r'^about'**, views.about),  path(**''**, views.index), ] |

В данном случае адрес "^about/contact" представляет более конкретный маршрут по сравнению c "^about". Поэтому он определяется в первую очередь.

Основные элементы синтаксиса регуляных выражений

Некоторые базовые элементы регуляных выражений, которые можно использовать для определения адресов URL:

* ^(начало адреса)
* $(конец адреса)
* +(1 и более символов)
* ?(0 или 1 символ)
* {n}(n символов)
* {n, m}(от n до m символов)
* .(любой символ)
* \d+(одна или несколько цифр)
* \D+(одна или несколько НЕ цифр)
* \w+(один или несколько буквенных символов)

Рассмотрим несколько возможных сопоставлений шаблонов адресов и запросов:

|  |  |
| --- | --- |
| Адрес | Запрос |
| r'^$' | http://127.0.0.1/ (корень сайта) |
| r'^about' | http://127.0.0.1/about/ или http://127.0.0.1/about/ |
| r'^about/contact' | http://127.0.0.1/about/contact |
| r'^products/\d+/' | http://127.0.0.1/products/23/ или http://127.0.0.1/products/6459/abc  Но не соответствует запросу http://127.0.0.1/products/abc/ |
| r'^products/\D+/' | http://127.0.0.1/products/abc/ или http://127.0.0.1/products/abc/123  Не соответствует запросу http://127.0.0.1/products/123/ или http://127.0.0.1/products/123/abc |
| r'^products/phones|tablets/' | http://127.0.0.1/products/phones/1 или http://127.0.0.1/products/tablets/  Не соответствует запросу http://127.0.0.1/products/clothes/ |
| r'^products/\w+' | http://127.0.0.1/products/abc/ или http://127.0.0.1/products/123/  Не соответствует запросу http://127.0.0.1/products/abc-123 |
| r'^products/[-\w]+/' | http://127.0.0.1/products/abc-123 |
| r'^products/[A-Z]{2}/' | http://127.0.0.1/products/RU  Не соответствует запросам http://127.0.0.1/products/Ru или http://127.0.0.1/products/RUS |

## Параметры представлений

Функции-представления могут принимать параметры, через которые могут передаваться различные данные. Подобный параметры передаются в адресе URL. Например, в запросе http://localhost/index/3/5/ последние два сегмента 3/5/ могут представлять параметры URL, которые могут быть связанны с параметрами функции-представления через систему маршрутизации.

Определение параметров через функцию re\_path

Определим в приложении в файле views.py следующие функции:

|  |
| --- |
| **from** django.http **import** HttpResponse  **def** products(request, productid):  output = **"<h2>Product № {0}</h2>"**.format(productid)  **return** HttpResponse(output)  **def** users(request, id, name):  output = **"<h2>User</h2><h3>id: {0} name: {1}</h3>"**.format(id, name)  **return** HttpResponse(output) |

Функция **products** кроме параметра request принимает также параметр productid - условный номер товара. Отправляемый пользователю ответ содержит значение этого параметра.

Функция users принимает два дополнительных параметра: id и name - условные идентификатор и имя пользователя. И подобным образом эти данные отправляются обратно пользователю.

Теперь изменим файл urls.py, чтобы он мог сопоставить данные функции с запросами:

|  |
| --- |
| **from** django.urls **import** re\_path **from** SampledjangoApplication **import** views  urlpatterns = [  re\_path(**r'^products/(?P<productid>\d+)/'**, views.products),  re\_path(**r'^users/(?P<id>\d+)/(?P<name>\D+)/'**, views.users), ] |

Для представления параметра в шаблоне адреса используется выражение ?P<>. Общее определение параметру соответствует формату (?P<имя\_параметра>регулярное\_выражение). Между угловыми скобками помещается название параметра. После закрывающей угловой скобки идет регулярное выражение, которому дожно соответствовать значение параметра.

Например, ?P<productid>\d+ указывает, что параметр называется productid, и он должен соответствовать регулярному выражению \d+, то есть представлять последовательность цифр.

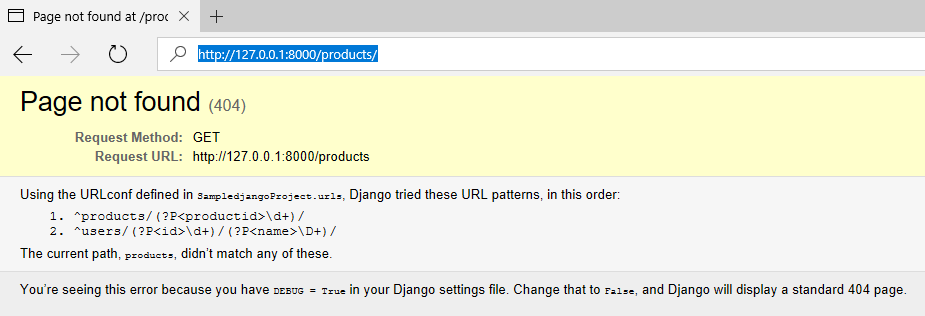
Во втором шаблоне адреса определяются два параметра: id и name ((?P<id>\d+)/(?P<name>\D+)). При этом параметр id должен представлять число, а параметр name должен состоять только из буквенных символов.

Ну и также отметим, что количество и название параметров в шаблонах адресов URL соответствуют количеству и названиям параметров соответствующих функций, которые обрабатывают запросы по данным адресам.

Теперь мы можем чем адресную строку передать данные в приложений.

|  |  |
| --- | --- |
|  |  |

Однако, что если мы не передадим значение для параметра или передадим ему значение, которое не соответствует регулярному выражению? В этом случае система не сможет найти ресурс для обработки запроса:



В этом случае мы можем определить в файле views.py для параметра функции products значение параметра по умолчанию:

|  |
| --- |
| **def** products(request, productid = 21):  output = **"<h2>Product № {0}</h2>"**.format(productid)  **return** HttpResponse(output) |

То есть если в функцию не передается значение для параметра productid, то он получает значение 21.

В этом случае надо дополнительно определить еще один маршрут в файле urls.py:

|  |
| --- |
| **from** django.urls **import** re\_path **from** SampledjangoApplication **import** views  urlpatterns = [  re\_path(**r'^products/$'**, views.products),  re\_path(**r'^products/(?P<productid>\d+)/'**, views.products),   re\_path(**r'^users/(?P<id>\d+)/(?P<name>\D+)/'**, views.users), ] |

**Определение параметров через функцию path**

Возьмем те же функции в файле views.py:

|  |
| --- |
| **from** django.http **import** HttpResponse  **def** products(request, productid):  output = **"<h2>Product № {0}</h2>"**.format(productid)  **return** HttpResponse(output)  **def** users(request, id, name):  output = **"<h2>User</h2><h3>id: {0} name: {1}</h3>"**.format(id, name)  **return** HttpResponse(output) |

Определение параметров с помощью функции path() будет выглядеть следующим образом:

|  |
| --- |
| **from** django.urls **import** path **from** SampledjangoApplication **import** views  urlpatterns = [  path(**'products/<int:productid>/'**, views.products),  path(**'users/<int:id>/<name>/'**, views.users), ] |

Параметры заключаются в угловые скобки в формате <спецификатор:название\_параметра>. Например, в первом маршруте параметр productid имеет спецификатор int.

По умолчанию Django предоставляет следующие спецификаторы:

* str: соответствует любой строке за исключенем символа "/". Если спецификатор не указан, то используется по умолчанию
* int: соответствует любому положительному числу
* slug: соответствует последовательности буквенных символов ASCII, цифр, дефиса и символа подчеркивания, например, building-your-1st-django-site
* uuid: сооветствует идентификатору UUID, например, 075194d3-6885-417e-a8a8-6c931e272f00
* path: соответствует любой строке, которая также может включать символ "/" в отличие от спецификатора str

Значения для параметров по умолчанию

Определеним для функций в views.py значения для параметров по умолчанию:

|  |
| --- |
| **from** django.http **import** HttpResponse  **def** products(request, productid=3):  output = **"<h2>Product № {0}</h2>"**.format(productid)  **return** HttpResponse(output)  **def** users(request, id=1, name=**"Bob"**):  output = **"<h2>User</h2><h3>id: {0} name: {1}</h3>"**.format(id, name)  **return** HttpResponse(output) |

В этом случае для каждой функции надо определить по два маршрута:

|  |
| --- |
| **from** django.urls **import** path **from** SampledjangoApplication **import** views  urlpatterns = [  path(**'products/'**, views.products),  path(**'products/<int:productid>/'**, views.products),   path(**'users/'**, views.users),  path(**'users/<int:id>/<str:name>/'**, views.users), ] |

## Параметры строки запроса

От параметров, которые передаются через адрес URL, следует отличать параметры, которые передаются через строку запроса. Например, в запросе http://localhost/index/3/Tom/ два последних сегмента 3/Tom/ представляют параметры URL. А в запросе http://localhost/index?id=3&name=Tom те же самые значения 3 и Tom представляют параметры строки запроса.

Параметры строки запроса указывается после вопросительного знака ?. Каждый параметр предсталяет пару ключ-значение, напимер, в id=3 : id - название или ключ параметра, а 3 - его значение. Параметры в строке запроса отделяются друг от друга знаком амперсанда.

Для получения параметров из строки запроса применяется метод request.GET.get().

Например, определим в файле views.py следующие функции:

|  |
| --- |
| **from** django.http **import** HttpResponse  **def** products(request, productid):  category = request.GET.get(**"cat"**, **""**)  output = **"<h2>Product № {0} Category: {1}</h2>"**.format(productid, category)  **return** HttpResponse(output)  **def** users(request):  id = request.GET.get(**"id"**, 1)  name = request.GET.get(**"name"**, **"Tom"**)  output = **"<h2>User</h2><h3>id: {0} name: {1}</h3>"**.format(id, name)  **return** HttpResponse(output) |

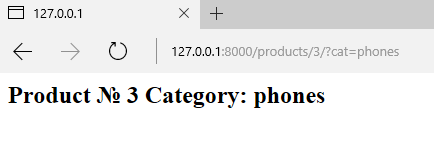
Функция products принимает обычный параметра productid, который будет передаваться через URL. И также из строки запроса извлекается значение параметра cat: request.GET.get("cat", "") - первый аргумент функции - это название параметра строки запроса, значение которого надо извлечь, а второй аргумент - значение по умолчанию, если вдруг в строке запроса не оказалось подобного параметра.

В функции users из строки запроса извлекаются значения параметров id и name.

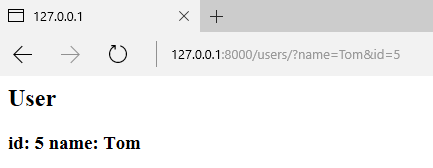
В файле urls.py определим следующие маршруты:

|  |
| --- |
| **from** django.urls **import** path **from** SampledjangoApplication **import** views  urlpatterns = [  path(**'products/<int:productid>/'**, views.products),  path(**'users/'**, views.users), ] |

При обращении к приложению по адресу http://127.0.0.1:8000/products/3/?cat=phones, число 3 будет представлять параметр URL будет переходить в параметр productid, а cat="phones" будет представлять параметр cat строки запроса.



А при обращении по адресу http://127.0.0.1:8000/users/?name=Tom&id=5 значения Tom и 5 будет представлять соответственно параметры name и id:



## Переадресация и отправка статусных кодов

**Переадресация**

При перемещении документа с одного адреса на другой мы можем воспользоваться механизмом переадресации, чтобы указать пользователям и поисковику, что документу теперь доступен по новому адресу.

Переадресация бывает временная и постоянная. При временной переадресации мы указываем, что документ временно перемещен на новый адрес. В этом случае в ответ отправляется статусный код 302. При постоянной переадресации мы уведомляем, что документ теперь постоянно будет достуен по новому адресу

Для создания временной переадресации применяется класс HttpResponseRedirect, а для постоянной - класс HttpResponsePermanentRedirect, которые расположены в пакете django.http.

Так, определим в файле views.py следующий код:

|  |
| --- |
| **from** django.http **import** HttpResponse, \  HttpResponseRedirect, HttpResponsePermanentRedirect  **def** index(request):  **return** HttpResponse(**"Index"**)  **def** about(request):  **return** HttpResponse(**"About"**)  **def** contact(request):  **return** HttpResponseRedirect(**"/about"**)  **def** details(request):  **return** HttpResponsePermanentRedirect(**"/"**) |

При обращении к функции contact она будет перенаправлять по пути "about", который будет обрабатываться функцией about. А функция details будет использовать постоянную переадресацию и перенаправлять на корень веб-приложения.

И также в файле urls.py для тестирования определим следующие маршруты:

|  |
| --- |
| **from** django.urls **import** path **from** SampledjangoApplication **import** views  urlpatterns = [  path(**''**, views.index),  path(**'about/'**, views.about),  path(**'contact/'**, views.contact),  path(**'details/'**, views.details), ] |

**Отправка статусных кодов**

Также в пакете django.http есть ряд классов, которые позволяют отправлять определенный статусный код:

|  |  |
| --- | --- |
| Статусный код | Класс |
| 304 (Not Modified) | HttpResponseNotModified |
| 400 (Bad Request) | HttpResponseBadRequest |
| 403 (Forbidden) | HttpResponseForbidden |
| 404 (Not Found) | HttpResponseNotFound |
| 405 (Method Not Allowed) | HttpResponseNotAllowed |
| 410 (Gone) | HttpResponseGone |
| 500 (Internal Server Error) | HttpResponseServerError |

Применение классов:

|  |
| --- |
| **from** django.http **import** \*  **def** m304(request):  **return** HttpResponseNotModified()  **def** m400(request):  **return** HttpResponseBadRequest(**"<h2>Bad Request</h2>"**)  **def** m403(request):  **return** HttpResponseForbidden(**"<h2>Forbidden</h2>"**)  **def** m404(request):  **return** HttpResponseNotFound(**"<h2>Not Found</h2>"**)  **def** m405(request):  **return** HttpResponseNotAllowed(**"<h2>Method is not allowed</h2>"**)  **def** m410(request):  **return** HttpResponseGone(**"<h2>Content is no longer here</h2>"**)  **def** m500(request):  **return** HttpResponseServerError(**"<h2>Something is wrong</h2>"**) |