



Урок 4

Фреймворк Django

Основные понятия ORM, структура и особенности проектирования.

[Django](#)

[Шаблон проектирования Django](#)

[Создание проекта Django](#)

[Django-приложение и его разработка](#)

[Файл settings.py и его настройка](#)

[Сервер разработки Django](#)

[Модели в Django](#)

[Админка в Django](#)

[URL-адреса проекта и приложения](#)

[Контроллеры в Django](#)

[Шаблоны в Django](#)

[Контекст в шаблонах Django](#)

[Статика в шаблонах Django](#)

[URL-ссылки в шаблонах](#)

[Наследование шаблонов](#)

[Логика работы Django-приложения](#)

[Развертывание проекта](#)

[Понятие mod_wsgi](#)

[Практическое задание](#)

[Дополнительные материалы](#)

[Используемая литература](#)

Django

- **Что такое Django, и каковы его преимущества?**

Это свободно распространяемый фреймворк, который позволяет создавать веб-приложения различных уровней сложности (сайты-визитки, интернет-магазины, интернет-сервисы). Благодаря Django можно обеспечить взаимосвязь веб-приложения с различными СУБД и реализовать для него полноценную панель администратора.

Django появился в 2005 году и зарекомендовал себя эффективным решением для веб-разработки. Он отличается прозрачностью в написании кода, понятен в освоении для разработчиков и предлагает заказчикам и их клиентам все необходимые возможности для создания веб-приложений.

Достоинства Django

- **Быстрота изучения.** Создание веб-приложений на связке Django+Python требует значительно меньше времени, чем на других фреймворках.
- **Высокий уровень функциональности.** Django считается одним из лучших фреймворков для разработки как простых интернет-ресурсов, так и высоконагруженных проектов с дизайном любого уровня сложности. Значительная доля веб-сервисов с высоким трафиком реализована с применением данного фреймворка.
- **Широкий диапазон встроенных функций.** Аутентификация пользователя, генерация карты сайта, управление контентом и RSS и другое.
- **Безопасность.** Проекты, созданные с использованием Django, обладают высоким уровнем защиты от угроз: SQL-инъекций, кросс-сайтового скриптинга, кросс-сайт подлогов.

Шаблон проектирования Django

- **Какой шаблон проектирования лежит в основе Django?**

В основе Django лежит шаблон проектирования **MVT**, который представляет собой взаимосвязь трех компонентов: модели, представления и шаблона (Model-View-Template).

Понятие модели в Django соответствует программному коду, обеспечивающему взаимодействие с базами данных. За реализацию бизнес-логики отвечает компонент «представление». Он обеспечивает доступ к моделям и извлечение соответствующих данных из баз, а также их передачу в определенный шаблон. Представление в MTV выступает связующим звеном между моделями и шаблонами. А последние определяют, как должны быть отображены извлеченные данные на веб-странице.

Создание проекта Django

- **Как создать новый проект Django?**

Разработка проектов на Django при решении реальных задач ведется применительно к ОС, на которой работает «боевой» сервер. Поэтому создавать проект следует под ту ОС, которая будет использоваться веб-приложением в продакшене. Большинство веб-серверов функционирует на Unix-ОС. Разработка Django-проекта под одну из них — например, под Linux,— связано с меньшими техническими сложностями, чем при выполнении аналогичной задачи под Windows. Поэтому дальнейшие практические задачи по Django-фреймворку будем решать в ОС Linux.

Надо принять во внимание, что на одной ОС может вестись разработка любого количества проектов, каждый из которых может работать под определенным набором библиотек. Например, один веб-сервис реализован на Django 1.10 и Python 3, а второй — на Django 1.5 и Python 2. Так что проекты могут иметь разные зависимости (в том числе конфликтующие), и при этом функционировать на одной ОС. Проблему несовместимости библиотек решают за счет использования виртуального окружения.

Работа начинается с подготовки виртуального окружения для Django-проекта. Для этого необходимо открыть терминал и выполнить набор команд.

Создание виртуального окружения:

```
virtualenv virtualenvs/test_ve
```

Появляется директория **virtualenvs** с папкой виртуального окружения **test_ve**.

Активация виртуального окружения:

```
source ./virtualenvs/test_ve/bin/activate
```

Далее устанавливаются зависимости. Они указываются, как правило, в простом текстовом файле **requirements.txt**, который содержит список зависимостей (перечень устанавливаемых для работы проекта библиотек). Пример файла **requirements.txt**:

```
Django==1.11.5  
django-widget-tweaks==1.4.1
```

Установка зависимостей:

```
pip install -r requirements.txt
```

Далее переходим непосредственно к созданию Django-проекта:

```
django-admin.py startproject test_prj
```

По выполнении этой команды создается внешний каталог Django-проекта **test_prj**, содержащий внутренний каталог **test_prj** и скрипт управления Django-проектом **manage.py**.

Внешний каталог представляет собой контейнер для проекта, причем его название может быть любым. Оно существует для разработчика, Django его не использует. Внутренний каталог является Python-пакетом проекта. Это название будет использоваться в дальнейшем для импорта из проекта, например:

```
import test_prj.settings
```

Если внутренний каталог переименовать уже после создания проекта, то изменится и директория импорта. Возникнет конфликт имен и придется вносить существенные правки. Поэтому желательно с именем проекта определиться изначально.

Django-приложение и его разработка

- *Что такое приложение Django и как его создать?*

Необходимо понимать разницу между проектом и приложением в Django. Приложение представляет собой программу, решающую определенную задачу, а проект — это совокупность приложений. Например, интернет-магазин — это проект, а страницы, содержащие информацию о компании, номенклатуру товаров, гостевую книгу, контактные данные, новости — отдельные приложения.

Для создания Django-приложения необходимо перейти в директорию проекта (при этом находясь под виртуальным окружением):

```
cd test_prj
```

И выполнить команду создания нового приложения:

```
python manage.py startapp test_app
```

В директории проекта будет создана папка **test_app**, содержащая служебные файлы нового приложения.

Файл settings.py и его настройка

- *Для чего нужен файл settings.py и как его настроить?*

Этот файл находится во внутреннем каталоге проекта. Он позволяет сконфигурировать проект: например, указать настройки его базы данных, организовать раздачу статики, выполнить привязку созданных приложений.

Настройка файла **settings.py** выполняется постепенно. Для работы с файлом подходит любой текстовый редактор с подсветкой синтаксиса — например, Notepad++.

На начальном этапе правки файла **settings.py** минимальны, в основном это добавление новых приложений в список **INSTALLED_APPS**:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'test_app',  
    'widget_tweaks',  
]
```

Последнее приложение не создается в рамках работы над проектом. Оно установилось автоматически из файла **requirements.txt**. Теперь его надо подключить. Данное приложение применяется при рендеринге элементов форм в шаблонах и упрощает реализацию верстки проекта.

Второй обязательный шаг — организация раздачи статических файлов (из директорий **CSS**, **JS**, **IMG**, **FONTS**) посредством сервера Django. По умолчанию директорией размещения статики является папка **static**, которая создается вручную разработчиком и наполняется необходимыми служебными файлами. Папка **static** должна располагаться во внешнем каталоге проекта.

Для организации раздачи статики необходимо добавить в конец файла **settings.py** следующие строки:

```
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static'),
]
```

Параметр **BASE_DIR** указывает на путь к корню Django-проекта (внешний каталог).

По умолчанию Django работает с СУБД **SQLite3** (файл БД создается автоматически в корневом каталоге). Но на практике часто встречаются проекты, работающие в привязке к **PostgreSQL**. Если проект связан с данной СУБД, необходимо предварительно создать файл БД в корне проекта. Для этого можно воспользоваться приложением **PgAdmin** с графическим интерфейсом пользователя для работы с СУБД **PostgreSQL**. После этого следует изменить параметры **DATABASES** файла **settings.py**:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'db_name',
        'USER': 'user_name',
        'PASSWORD': '12345',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

С помощью файла **settings.py** в Django-проекте также можно настроить механизм загрузки шаблонов. Для этого используется настройка **TEMPLATES**. Опция **APP_DIRS** принимает значение **TRUE** или **FALSE**. Со значением **TRUE** она указывает, что поиск шаблонов будет вестись по установленным приложениям:

```
'APP_DIRS': True
```

Опция **DIRS** содержит указание на список каталогов с шаблонами. Именно по порядку элементов этого списка будет выполняться поиск шаблонов. То есть сначала на наличие нужных шаблонов будут проверяться именно директории из этого списка. Например, имеем в корне проекта директорию **templates** с шаблонами общего назначения и хотим, чтобы поиск шаблонов в первую очередь осуществлялся именно в ней, а уже потом — по директориям **templates** директорий, соответствующих приложениям. Для этого изменим содержимое файла **settings.py** следующим образом:

```
'DIRS': [
    os.path.join(BASE_DIR, 'templates')
],
```

Сервер разработки Django

- *Как запустить сервер разработки Django?*

Это встроенный легкий веб-сервер, позволяющий проверить правильность установки фреймворка Django и отследить результаты разработки, не тратя время на настройку боевого веб-сервера (например, Apache). Веб-сервер Django автоматически отслеживает изменения программного кода и перезагружает проект. Мы можем видеть изменения, если все хорошо, или возникающие ошибки. При этом перезагружать веб-сервер не надо.

Для запуска сервера Django необходимо, находясь под виртуальным окружением проекта, перейти в его внешний каталог и выполнить команду:

```
python manage.py runserver
```

Далее в любом веб-браузере открыть новую страницу и ввести следующий url-адрес:

```
127.0.0.1:8000 или localhost:8000
```

Если появляется надпись “**It worked**”, значит Django работает нормально и можно переходить к следующим этапам разработки проекта.

Модели в Django

- *Что такое модель в Django и как ее создать?*

Модели в Django являются объектами, которые отображают информацию о данных, хранящихся в БД. Модель может использоваться для хранения информации о товаре (его карточка), о категориях товаров, о брендах, поставщиках. Каждой модели в Django-проекте соответствует определенная таблица БД.

Программный код моделей каждого приложения хранится в файле **models.py** соответствующей директории. Этот файл появляется при создании нового приложения и изначально содержит только служебные инструкции, в частности:

```
from django.db import models
```

С помощью данной инструкции мы импортируем модуль, содержащий модели базового класса (`models.Model`). От этого стандартного базового класса и наследуются все модели, создаваемые разработчиком. Указывая родителем класса модели базовый класс **models.Model**, мы указываем Django, что создаваемый объект (класс модели) является моделью и его нужно сохранить в базу данных.

Создание новой модели проекта на Django инициируется с помощью следующей конструкции:

```
class имя_модели(models.Model)
```

Рассмотрим простейший пример создания модели «Карточка товара» для Django-проекта:

```
...
class Good_Item(models.Model):

    created_at = models.DateTimeField(
        verbose_name=u'Дата добавления',
        auto_created=True,
        auto_now_add=True
    )

    title = models.CharField(
        verbose_name=u'Название',
        max_length=255
    )

    price = models.PositiveIntegerField(
        verbose_name=u'Цена',
        default=0
    )

    vendor = models.CharField(
        verbose_name=u'Поставщик',
        max_length=255
    )

    def __unicode__(self):
        return self.title

    class Meta:
        verbose_name = u'Карточка товара'
        verbose_name_plural = u'Карточки товаров'
```

Чтобы модель записалась в базу данных, необходимо создать миграции, находясь под виртуальным окружением в корне проекта. Для этого используется такая конструкция:

```
python manage.py makemigrations имя_приложения
```

Для нашего примера эта инструкция будет выглядеть так:

```
python manage.py makemigrations catalog
```

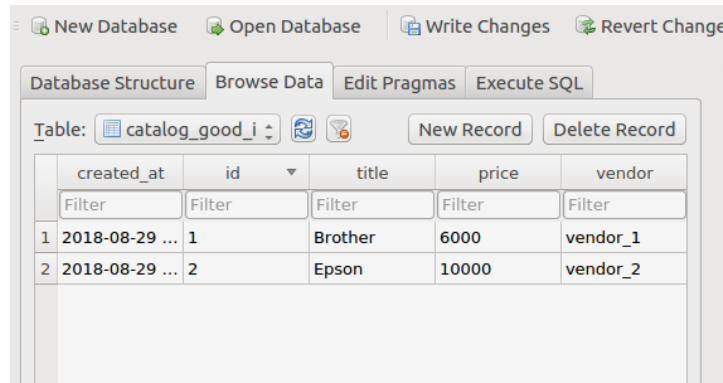
С помощью этой команды мы даем знать Django, что внесли изменения в модель, а именно — создали ее. При этом Django добавит соответствующий файл с миграцией. Теперь, чтобы изменения вступили в силу (были отправлены в БД), следует выполнить такую команду:

```
python manage.py migrate имя_приложения
```


Для нашего приложения так:

```
python manage.py migrate catalog
```

Если теперь открыть соответствующий проекту файл БД, можно увидеть созданную в соответствии с моделью таблицу. Для просмотра ее содержимого можно воспользоваться одной из свободно распространяемых программ с графическим интерфейсом — например, **SQLiteBrowser**:



Админка в Django

- **Что такое админка проекта на Django, и как ее создать?**

Панель администратора Django реализует интерфейс для работы с моделями. Админка позволяет оперировать записями таблицы БД, создаваемыми на основе соответствующей модели.

Чтобы начать работать с моделью через админку, необходимо зарегистрировать ее:

```
admin.site.register(имя_модели)
```

Для нашего примера:

```
admin.site.register(Good_Item)
```

Если запустим Django-сервер и попытаемся войти в админку, увидим страницу авторизации, где будет предложено указать логин и пароль. Но как быть, если мы еще ни одного пользователя не создали? Надо создать суперпользователя с полными правами доступа к проекту. Но ведь данные о нем также должны куда-то заноситься? А если мы сейчас откроем файл БД, никаких подходящих таблиц не увидим.

В Django для хранения информации о пользователях и их правах предусмотрены отдельные служебные модели. С ними работают по тому же принципу, что и с обычными моделями — только вручную ничего создавать не надо.

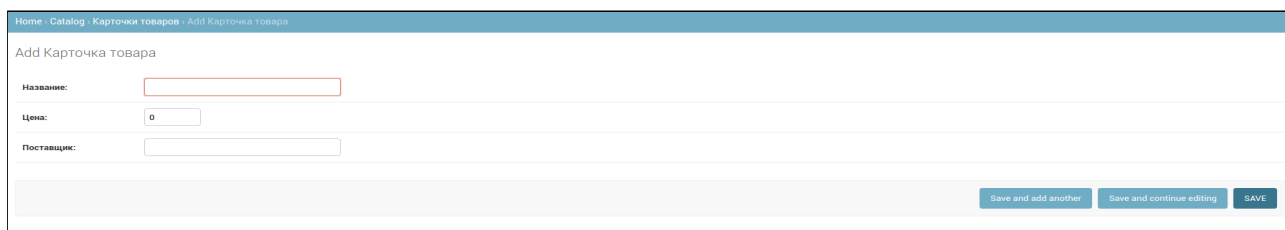
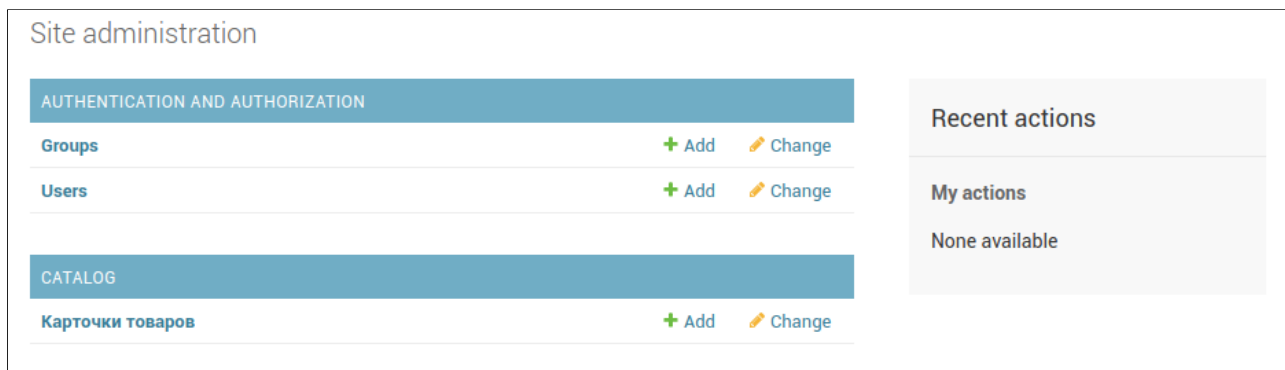
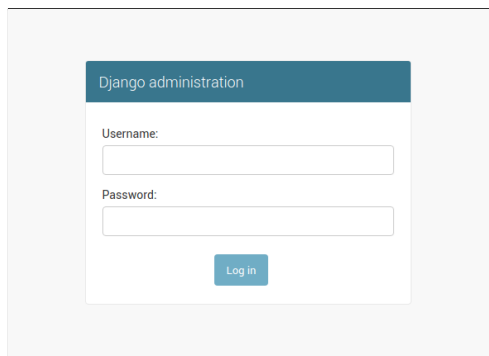
Мы лишь создаем миграции и отправляем изменения в БД:

```
python manage.py makemigrations
python manage.py migrate
```

Теперь остается создать суперпользователя, указав логин, пароль (минимум 8 символов) и электронную почту:

```
python manage.py createsuperuser
```

И войти в админку проекта:



URL-адреса проекта и приложения

- **Как задаются url-адреса для всего проекта в целом и для отдельного приложения?**

Любая веб-страница имеет свой url-адрес. В Django реализован свой механизм определения url-адресов. Во внутреннем каталоге проекта (в нашем примере — это **test_prj/test_prj**) находится файл **urls.py**. Он содержит шаблоны url-адресов и определяет систему маршрутизации проекта.

В исходном виде (после создания проекта) файл **urls.py** содержит шаблон только одного url-адреса:

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
]
```

Это значит, что если мы перейдем по адресу “**url-адрес проекта/admin**”, то окажемся в панели администратора проекта.

Если перейдем по самому url-адресу проекта сейчас, увидим лишь стандартную страницу приветствия Django с надписью “**It worked**”. Но мы хотим, чтобы url-адрес проекта возвращал что-то конкретное — например, страницу с каталогом товаров.

Представим, что приложение-каталог у нас уже реализовано, и для него сделан список url-адресов, файл **urls.py**. Остается импортировать его из директории приложения-каталога — за это отвечает инструкция **include**. Не забываем импортировать и ее:

```
from django.conf.urls import url, include
```

При импорте файла с адресами указываем имя приложения и через точку — импортируемого файла. И тогда главный файл **urls.py** из внутреннего каталога проекта примет такой вид:

```
from django.conf.urls import url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'', include('catalog.urls')),
]
```

Теперь при переходе по url-адресу проекта запрос будет перенаправляться к файлу **catalog.urls**, где состоится поиск дальнейших инструкций.

Далее необходимо подготовить шаблон url-адреса, при переходе на который посетитель сайта увидит веб-страницу с товарами. В этом шаблоне следует указать контроллер, который будет отвечать за формирование страницы. Необходимо перейти в директорию приложения **catalog**, создать в ней файл **urls.py** и записать в нем следующие инструкции:

```
from django.conf.urls import url
from catalog.views import ItemsListView

urlpatterns = [
    url(r'^$', ItemsListView.as_view()),
]
```

Регулярное выражение **^\$** в шаблоне означает пустую строку. Адрес нашего ресурса (**http://127.0.0.1:8000**) не является частью url-адреса для Django-обработчика **url**. Значит нет необходимости готовить специальное регулярное выражение — достаточно конструкции **^\$**.

Но если сейчас перейти по адресу **http://127.0.0.1:8000**, увидим ошибку, поскольку указанный контроллер **ItemsListView** еще не создан и запрос некому обрабатывать.

Ошибка может появиться уже в терминале:

```
ImportError: cannot import name ItemsListView
```

Файл **urls.py** приложения-каталога может содержать привязки не только к главной странице сайта, но и к карточке товара, системе добавления отзывов и так далее.

Если созданное приложение не предусматривает дальнейших переходов пользователя (например, это стандартная страница «О компании»), то для соответствующего приложения (назовем его **about**) нет необходимости создавать отдельный файл с шаблонами url-адресов. Достаточно добавить шаблон url-адреса в файл **urls.py** корневого каталога проекта и привязать к этому шаблону предварительно импортированный контроллер **AboutView**. Тогда файл **urls.py** корня проекта будет выглядеть следующим образом:

```
from django.conf.urls import url, include
from django.contrib import admin
from about.views import AboutView

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'', include('catalog.urls')),
    url(r'^about/', AboutView.as_view() , name = "about"),
]
```

Если контроллер, обрабатывающий шаблон url-адреса с информацией о компании, корректно работает, остается перейти по соответствующему url-адресу:

```
127.0.0.1:8000/about/
```

Контроллеры в Django

- **Что такое контроллеры в Django?**

В Django понятие контроллера равнозначно понятию URL-маршрутизатора, обеспечивающего реализацию логики «запрос-ответ». При переходе пользователя по url-адресу веб-страницы запускается поиск соответствующего шаблона и выполняется привязанный к нему контроллер.

Контроллер является обработчиком запроса. Он принимает параметры запроса, обращается к модели и извлекает из нее данные, после чего формирует ответ — вызывает определенный html-шаблон и передает в него извлеченные из БД данные. Они называются контекстом шаблона.

Рассмотрим код простейшего контроллера, реализованного на базе класса **ListView** и извлекающего из модели **Good_Item** все записи (карточки товара):

```
from django.views.generic.list import ListView
from catalog.models import Good_Item

class ItemsListView(ListView):
    model = Good_Item
    template_name = "items_index.html"

    def get_queryset(self):
        items = Good_Item.objects.all()
        return items
```

Аналогичный контроллер можно сделать в виде функции. Но на многих реальных проектах требуется умение работать именно со встроенными классами, поскольку в них уже реализована определенная функциональность, и это позволяет уйти от избыточности кода.

Шаблоны в Django

- **Что такое шаблоны в Django и как с ними работать?**

Поскольку Django является веб-фреймворком, в нем на основе шаблонов реализован механизм генерации html-кода. Каждый шаблон содержит статический html-код, который не меняется в процессе работы пользователя с приложением. Еще в шаблоне есть динамические данные — они описываются специальными переменными. А значения этих переменных (контекст шаблона) передаются из контроллера, который вызывает шаблон и передает в него данные, полученные из модели.

Механизм динамической подстановки данных реализуется с помощью шаблонизатора. Значения переменных, созданных пользователем в контроллере и переданных в шаблон, подставляются с помощью следующей конструкции:

```
{{ имя переменной из контроллера(запись базы данных).поле базы данных }}
```

В языке шаблонов также реализованы служебные операторы — теги шаблонов. Они записываются следующим образом:

```
{% имя тега %}
```

С помощью тегов в Django реализуются циклы, условные операторы, вывод текущей даты, подключение статики.

Создадим для примера с каталогом товаров цикл перебора карточек с выводом названий товаров и цен. Здесь не работаем с контекстом шаблона, а используем функцию **get_queryset**, которая возвращает список объектов заданной модели. После вызова шаблона в этом случае доступна переменная **object_list** со списком записей модели. Выполним обработку этого списка в цикле и выведем данные в табличном виде:

```

<table>
  <tr>
    <th>Название</th><th>Цена</th>
  </tr>
  {% for object in object_list %}
    <tr>
      <td>{{ object.title }}</td><td>{{ object.price }}</td>
    </tr>
  {% endfor %}
</table>

```

Здесь мы обращаемся к полям **title** (название) и **price** (цена) каждого объекта (записи базы данных).

Результат:

Название	Цена
Brother	6000
Epson	10000

Контекст в шаблонах Django

- *Как передать контекст в шаблон?*

В зависимости от того, реализуются контроллеры в виде функций или классов, применяются два способа передачи контекста (динамически обновляемых данных шаблонов) в шаблон.

Для функции-контроллера:

```

from django.shortcuts import render

def index(request):
    user = {"name" : "Ivan Ivanov", "age" : 30}
    data = {"user": user}
    return render(request, "index.html", context=data)

```

В данном случае используем функцию **render**, в которую передаем три параметра:

- поступивший запрос — пользователь перешел по url-адресу, и сработал один из шаблонов адресов;
- шаблон, который должен быть вызван данной функцией — т.е. html-страницу, которую должен увидеть пользователь;
- переменная **context**, как раз и содержащая указание на передаваемый контекст. В нашем случае это словарь.

Теперь чтобы получить доступ к имени пользователя, достаточно добавить в шаблон конструкцию:

```
{{ user.name }}
```

Для класса контроллера:

```
from django.views.generic.base import TemplateView
class Index(TemplateView):
    template_name = 'index.html'
    def get_context_data(self, **kwargs):
        context = super(Index, self).get_context_data(**kwargs)
        user = {"name" : "Ivan Ivanov", "age" : 30}
        context.update({
            'user': user
        })
        return context
```

Статика в шаблонах Django

- *Как правильно настроить статику в шаблонах?*

К статике в проектах на Django относятся файлы, содержащие JavaScript-код, CSS-инструкции, а также графические файлы. По умолчанию они сохраняются в директории **static**. Указание на это содержится в файле **settings.py** внутренней папки проекта. При этом саму директорию **static** приходится создавать принудительно.

Чтобы иметь возможность подключить к шаблону CSS-стилизацию или реализовать интерактивность с помощью JavaScript-сценария, необходимо в начале шаблона прописать следующий тег:

```
{% load staticfiles %}
```

Теперь чтобы, например, стилизовать веб-страницу, необходимо указать ссылку на соответствующий статический файл с CSS-инструкциями:

```
<link rel="stylesheet" href="{% static 'css/style.css' %}">
```

Выполним стилизацию html-страницы из нашего примера с таблицей, содержащей информацию о товарах. Для этого создадим в директории **catalog** папку **static** и сохраним в ней файл **style.css** с инструкциями стилизации таблицы:

```

table
{
    border-collapse:collapse;
}

td
{
    border: 1px solid black;
}

td
{
    padding: 10px;
}

th
{
    background-color:#333;
    color:#fff;
    border-radius: 10px 10px 0 0;
}

```

Тогда шаблон страницы со списком товаров в html-представлении будет выглядеть следующим образом:

```

{% load staticfiles %}
<link rel="stylesheet" href="{% static 'css/style.css' %}">
<table>
    <tr>
        <th>Название</th><th>Цена</th>
    </tr>
    {% for object in object_list %}
        <tr>
            <td>{{ object.title }}</td><td>{{ object.price }}</td>
        </tr>
    {% endfor %}
</table>

```

Результат его выполнения:

Название	Цена
Brother	6000
Epson	10000

URL-ссылки в шаблонах

- *Как добавить url-ссылку в шаблон?*

Возможность добавления url-ссылок в шаблоны реализуется в Django с помощью соответствующего тега и имени шаблона. Например, чтобы разместить ссылку на раздел «О компании» на главной странице сайта, зададим имя шаблону url-адреса, соответствующее адресу страницы «О компании»:

```
urlpatterns = [
    url(r'^about/', About.as_view(), name='about')
]
```

Теперь в шаблоне главной страницы сайта необходимо добавить следующий тег:

```
<a href="{% url 'about' %}">О компании</a>
```

То есть конструкция тега url-ссылки имеет такой вид:

```
{% url '<имя_url-шаблона>' %}
```

Такой подход избавляет разработчика от необходимости редактировать ссылки в шаблонах при изменении схемы URL-адресов сайта. Достаточно внести изменения в файл **urls.py** проекта или аналогичные файлы приложений.

Наследование шаблонов

- *Как реализовать наследование шаблонов?*

Если проанализировать html-код веб-страниц, можно обнаружить повторяющиеся фрагменты: одинаковую шапку, меню, подвал. От избыточности кода надо уходить, поскольку при внесении изменений их придется дублировать в несколько файлов, что снижает надежность и скорость разработки.

В современных фреймворках, в том числе в Django, реализован механизм наследования шаблонов. Суть подхода в том, что создается базовый html-шаблон— например, **base.html** — с общей частью и блоками, содержимое которых будет динамически изменяться в шаблонах-наследниках.

Для выделения таких блоков с динамически изменяющимся содержимым применяется тег шаблона следующей конструкции:

```
{% block <имя блока> %}:
{% endblock %}
```

Если теперь указать этот тег в шаблоне-наследнике, то его содержимое встанет на место контента блока в базовом шаблоне. Если не прописать содержимое блока в шаблоне-наследнике, то его контент будет аналогичен базовому. Такой подход минимизирует код и упрощает его изменение.

Необходимо принимать во внимание, что если в шаблоне-наследнике прописать блоки, отсутствующие в базовом шаблоне, то они не будут выводиться в браузере. А если написать код вне блоков в шаблонах-наследниках, то результат его выполнения тоже отображаться не будет.

Чтобы установить дочернему шаблону базовый, необходимо в начале кода каждого наследника прописать следующую конструкцию:

```
{% extends '<путь_к_базовому_шаблону>' %}
```

Например:

```
{% extends 'base.html' %}
```

Логика работы Django-приложения

- **Какие действия выполняются в Django-проекте после перехода пользователя в приложение по ссылке в браузере?**
1. После перехода по ссылке на веб-страницу в файле **settings** выполняется проверка значения переменной **ROOT_URLCONF**. Как правило, значением переменной является ссылка на файл **urls.py** корня проекта.
 2. В файле **urls.py** проверяется каждый шаблон url-адреса сверху вниз. При обнаружении первого соответствия проверяется содержимое этого шаблона. Если в нем подключается набор шаблонов url-адресов определенного приложения, то далее будут просматриваться шаблоны файла **urls.py** уже на уровне приложения.
 3. Если к найденному шаблону привязан контроллер, он вызывается и выполняется. В контроллере происходит обращение к модели, связанной с определенной таблицей БД, и извлечение данных.
 4. Извлеченные данные могут обрабатываться по алгоритму и передаваться в контекст шаблона, привязанного к данному контроллеру.
 5. После завершения работы контроллера загружается соответствующий шаблон со статикой и динамическими данными, полученными из контекста. Таким образом формируется целевая веб-страница, которую видит пользователь в браузере.

Развертывание проекта

- **Как развернуть приложение на Django?**
1. **Развертывание Django с Apache.** В соответствии с документацией фреймворка, рекомендуемый вариант развертывания Django-проекта — это использование веб-сервера Apache. При этом необходим дополнительный инструмент — модуль **mod_python** или **mod_wsgi**. Django предусматривает возможность работы с http-сервером Apache версии 2.0 (и более поздних) и модулем **mod_python 3.0** (и последующих версий). Модуль **mod_python** интегрирует в веб-сервер Apache интерпретатор языка Python. Проект был заморожен с 2010 по 2013 год, но сейчас возобновлен. Тем не менее официальная документация рекомендует как альтернативу использовать модуль **mod_wsgi**.

2. **Развертывание Django с uWSGI.** Это легковесный и дружелюбный к разработчикам веб-сервер, реализованный для запуска Python-приложений посредством протокола **WSGI** (Web Server Gateway Interface). Он представляет собой универсальный стандарт взаимодействия между веб-сервером и веб-приложением. **uWSGI** может работать и как самостоятельный веб-сервер, и в связке с Apache и Nginx.
3. **Развертывание Django с Nginx.** Достойная альтернатива Apache: мощный легковесный веб-сервер, ориентированный как на самостоятельную работу, так и на интеграцию с другими решениями — например, с **uWSGI**. Совместим с Apache — чтобы снижать нагрузку на сервер и увеличивать скорость обработки запросов посетителей.

Понятие mod_wsgi

- *Что такое mod_wsgi?*

Это дополнение (модуль) к веб-серверу Apache, предоставляющий WSGI-совместимый интерфейс для работы с веб-приложениями, созданными на базе языка программирования Python версий 2 и 3.

Во встроенном режиме интегрирует Python в Apache и выполняет загрузку кода Python в память при запуске сервера. Код сохраняется в памяти, пока существует процесс Apache, что значительно повышает производительность по сравнению с другими механизмами.

В режиме демона **mod_wsgi** иницирует независимый процесс-демон, обрабатывающий запросы. Процесс может функционировать от имени различных пользователей веб-сервера — это повышает безопасность. Его можно перезапустить, минуя останов и перезапуск самого Apache-сервера. Django работает с любой версией Apache, поддерживающей **mod_wsgi**.

Практическое задание

Чтобы закрепить знания по фреймворку Django и подготовиться к собеседованию, предлагаем реализовать несложное приложение.

Практическим заданием будет каталог товаров, состоящий из двух страниц:

- главной со списком товаров и кнопкой их добавления;
- и страницы добавления товара.

Приложение должно функционировать в синхронном режиме: пользователь нажимает кнопку «Добавить товар», переходит на предназначенную для этого страницу, указывает в форме необходимые данные и нажимает «Сохранить». После этого он перенаправляется на главную страницу, где в табличной форме отображается список всех товаров. Проект должен быть привязан к базе данных **SQLite3** (БД по умолчанию) и иметь минимальную сложность стилевого оформления.

При работе над проектом:

1. Создать виртуальное окружение проекта, под которым установить необходимый инструментарий (файл **requirements.txt**).
2. Под виртуальным окружением создать Django-проект и одно приложение, настроить файл **settings.py**, выполнить базовые миграции. Запустить Django-сервер для проверки работоспособности проекта.

3. В каталоге приложения создать модель, которая должна хранить информацию о поступивших товарах: название, дату поступления, цену, единицу измерения, имя поставщика. Выполнить миграции.
4. Проверить правильность созданной модели, зарегистрировав ее в админке приложения.
5. На основе модели добавить класс формы указания данных о товаре. Использовать наследование от **forms.ModelForm**.
6. Настроить файл **urls.py** внутреннего каталога проекта. Он должен содержать два шаблона url-адресов: привязку к url-адресу админки проекта (будет в файле по умолчанию после создания проекта) и привязку к набору шаблонов url-адресов созданного приложения (оператор **include**).
7. Создать и настроить файл привязок **urls.py** для приложения. В этом файле создать две привязки: к url-адресу главной страницы проекта и к странице добавления товара. Для каждой из привязок указать функцию-контроллер и название. Функции-контроллеры должны отвечать за загрузку списка товаров на главной странице и добавление товара на второй странице.
8. В файле **views.py** каталога проекта реализовать два контроллера в формате функций. Первый должен извлекать все записи из модели с каталогом товаром и передавать переменную со списком товаров в контекст шаблона (html-страница со списком товаров). Во втором контроллере должен создаваться объект формы для ввода данных о товаре и выполняться рендеринг шаблона страницы добавления товара. В контекст шаблона необходимо передавать объект формы.
9. В корне проекта создать директорию **templates** с двумя стандартными шаблонами: базовым (**base.html**) и шаблоном формы (**form.html**). Базовый шаблон будет соответствовать каркасу главной страницы. В нем необходимо реализовать один динамически обновляемый блок — например, **{% block content %}{% endblock %}**. Он будет содержать таблицу со списком товаров, которая динамически подгружается из шаблона-наследника (html-страница со списком товаров). В файле **base.html** необходимо подключить статику и указать ссылку на CSS-файл со стилизацией проекта. Можно воспользоваться файлом **bootstrap.min.css** (его нужно скачать и поместить в каталог **.static/css**).
10. В шаблоне формы **form.html**, используя теги шаблонов, реализовать разметку формы. При этом использовать переменную контекста шаблона, содержащую объект формы, — например, **form**. К надписям полей обращаться по **field.label**, к самим полям — **field**.
11. В каталоге приложения создать директорию **templates** с двумя шаблонами: шаблоном html-страницы со списком товаров (**goods_list.html**) и html-страницы (формы) их добавления (**good_create.html**). В первом шаблоне необходимо указать шаблон-родитель **base.html**, кнопку добавления товара и разметить html-таблицу. Каждая из ее строк (кроме той, что с заголовками) должна формироваться при переборе содержимого переменной со списком товаров — мы ее предварительно передали в контекст данного шаблона из соответствующего контроллера. Для каждого из значений переменной (фактически — это запись базы данных), полученного в цикле, необходимо обратиться к нужному полю и вывести его в соответствующей ячейке. К кнопке привязать ссылку на страницу добавления товара. Для этого использовать имя нужного шаблона url-адреса файла **urls.py** приложения.
12. В шаблоне **good_create.html** создать html-тег **form**. В него поместить тег **include**, добавляющий html-разметку формы (**form.html**) и кнопку добавления товара. Для тега **form** необходимо определить два атрибута: **method** со значением **post** и **enctype** со значением **multipart/form-data**.

Следующий урок будет посвящен особенностям связи фреймворка Django с инструментарием AJAX, JavaScript и jQuery. Рассмотрим популярные вопросы с собеседований, затрагивающие эту тему. Продолжим работать Django-проектом: реализуем асинхронность загрузки формы — без перезагрузки главной страницы.

Дополнительные материалы

1. [Создаем свое первое веб-приложение при помощи Django.](#)
2. [Django на production. uWSGI + nginx. Подробное руководство.](#)
3. [Введение в NGINX: как его установить и настроить.](#)
4. [Разворачиваем приложения Django на production-сервере.](#)
5. [Развертывание Django-проекта под nginx.](#)
6. [Django. Запуск проекта в связке uWSGI и Nginx.](#)
7. [Apache vs Nginx: практический взгляд.](#)

Используемая литература

1. [The Django book.](#)
2. [Руководство Django.](#)
3. [Django Documentation.](#)
4. [Вопросы к собеседованию по Python и Django.](#)
5. [Setting up Django and your web server with uWSGI and nginx.](#)
6. [mod_python.](#)
7. [mod_wsgi.](#)
8. [uWSGI.](#)
9. [nginx.](#)