Занятие #41. Введение в Django

Подготовка окружения для проекта

Внешние пакеты, необходимые для работы над тем или иным проектом, например, пакет Django, называются **зависимостями проекта** (англ. **project dependencies**).

Для создания первого проекта на Django у вас должны быть установлены: Python 3, pip, virtualenv.

Для начала создадим проектную папку и перейдём в неё:

```
$ mkdir -p ~/projects
$ cd ~/projects
$ mkdir hello_django
$ cd hello_django
```

Здесь `~/projects` - это имя папки, где вы будете хранить проекты, а `hello_django` - название проекта. Можно заменить их на любые другие названия. Будем называть эту папку *корнем проекта*, т.к. весь код и другие файлы проекта будут находиться внутри неё.

Далее в корне проекта создадим виртуальное окружение Python 3:

```
$ virtualenv -p python3 venv
```

или:

```
$ python3 -m virtualenv -p python3 venv
```

Если ни одна из приведённых команд не работает, убедитесь, что у вас корректно установлена библиотека virtualenv.

Эта команда запускает питон, установленный в вашей системе с модулем virutalenv, и передаёт ему необходимые параметры для создания локального виртуального окружения - версию питона (python3) и название папки для виртуального окружения (venv).

Чтобы начать пользоваться виртуальным окружением, его нужно **активировать**. *Не забывайте это делать*. *Пакеты, установленные в виртуальное окружение, доступны только внутри него, и, забыв активировать виртуальное окружение, вы не сможете выполнять команды Django!* Для активации окружения используется следующая команда:

```
$ source venv/bin/activate
```

или:

```
$ . venv/bin/activate
```

Здесь `venv/bin/activate` - это путь к специальному скрипту внутри окружения. Скрипт настраивает вашу командную строку таким образом, чтобы система

использовала в первую очередь Python и pip из виртуального окружения, и только потом - из системы. Если вы всё сделали правильно, то приветствие вашей командной строки изменится, и в его начале вы увидите имя виртуального окружения `(venv)`:

```
(venv) ...$
```

А команда `which python` ("где питон?") будет показывать вам путь внутри виртуального окружения:

```
(venv) ...$ which python
/какой-то/путь/в/системе/.../venv/bin/python
```

Внутри виртуального окружения не обязательно писать `python3` или как-то иначе указывать версию питона, т.к. в нём доступна всего одна версия питона - та, которая была указана при создании.

Чтобы **отключить** (**деактивировать**) виртуальное окружение, используйте команду `deactivate`:

```
(venv) ...$ deactivate
...$
```

При успешной деактивации индикатор `(venv)` исчезнет.

Снова активируем виртуальное окружение и установим туда пакет Django версии 2.2:

```
(venv) ...$ pip install Django==2.2
```

Проверяем установленные пакеты:

```
(venv) ...$ pip freeze
Django==2.2
pytz==2020.1
sqlparse==0.3.1
```

requirements.txt

Рано или поздно в большинстве веб-проектов возникает необходимость восстановить окружение проекта на другом компьютере. Например, при поставке на сервер или передаче проекта другому разработчику. Само окружение копировать не принято, т.к. оно может содержать много пакетов и много весить. Кроме того, оно содержит ссылки на системные файлы и может сломаться при переносе. Т.к. все пакеты при необходимости скачиваются с репозитория PyPI (Python Package Index), то достаточно передать файл с названиями и версиями установленных пакетов.

Этот файл принято называть **requirements.txt**, он содержит вывод команды pip freeze и может быть создан следующей командой:

```
(venv) ...$ pip freeze > requirements.txt
```

Стрелка ">" обозначает перенаправление вывода команды в файл. Обратите внимание, что команда выполняется, когда окружение активно.

Для установки зависимостей из файла используется команда pip install с флагом -r:

```
(venv) ...$ pip install -r requirements.txt
```

Создавайте этот файл в каждом проекте, использующем виртуальное окружение Python и передавайте вместе с кодом.

Обновляйте файл requirements.txt каждый раз, когда вы добавляете в проект какую-нибудь библиотеку.

Django

Что вообще представляет из себя Django? **Django** - это **web-фреймворк** (**web-framework**) - дословно - "каркас" из готового кода на основе которого вы можете быстро создать собственный сайт.

Фреймворк отличается от библиотеки тем, что библиотека содержит какие-то базовые блоки кода - функции или классы, решающие какую-то одну задачу. Фреймворк - объединяет в себе код множества библиотек и представляет собой "пустой" проект, который можно запустить, но он не содержит никакой *бизнес*-логики, связанной конкретной с вашим сайтом. Грубо говоря, можно сравнить библиотеки с отдельными микросхемами и деталями, из которых можно собрать, например, робота (или что-нибудь другое). Фреймворк - это уже готовый робот, но с пустыми мозгами. Он знает, как можно что-то сделать, но ещё не знает, что ему нужно делать, и поэтому ничего не делает - эту часть вам нужно дописать самостоятельно.

Django включает в себя код для работы с базами данных, для обработки HTTP-запросов, для работы с формами, для авторизации пользователей, имеет встроенное приложение для администратора, предусматривает защиту проекта от возможных попыток взлома, предоставляет инструменты для тестирования, миграции данных, логирования, отправки почты и многое другое. Кроме того, для Django написано огромное количество плагинов, ещё больше расширяющих его функциональность.

Как создать проект

Сейчас у вас есть только корень проекта с виртуальным окружением внутри. Чтобы создать непосредственно проект на Django (скелет проекта), нужно выполнить следующую команду (*продолжаем работать в виртуальном окружении*):

```
(venv) ...$ django-admin.py startproject hello
```

`django-admin.py` - это специальный скрипт, поставляемый с Django, который используется для управления проектами. Он содержит набор команд для создания компонентов проекта, настройки, и тестирования.

`startproject` - это команда django-admin.py для создания проекта.

`hello` - название папки, куда будет помещён проект, и *главного приложения* проекта.

Полный список команд вы можете увидеть, запустив django-admin.py без параметров:

```
(venv) ...$ django-admin.py
```

Позже мы познакомимся с этими командами подробнее. Сейчас командой `startproject` мы создали папку `hello`, в которой находится исходный код проекта:

```
(venv) ...$ ls
hello venv
```

Скелет проекта имеет следующую структуру:

```
hello_django
|-- hello
|   |-- hello
|   |   |-- __init__.py
|   |   |-- settings.py
|   |   |-- urls.py
|   |   |-- wsgi.py
|   |-- manage.py
|-- venv
|-- requirements.txt
```

Здесь:

- `hello` - папка с кодом проека. Если папка hello_django содержит весь проект целиком, то hello - это только исходный код. Такая структура позволяет отделить вспомогательные файлы и папки от кода проекта.
- `hello/hello` - главное *приложение* проекта. По умолчанию оно называется так же, как сам проект.
- `settings.py` - главный файл настроек проекта. Он всегда находится в главном приложении и содержит настройки для подключения к базе данных, рассылки email, список приложений в проекте и т.д.
- `manage.py` - скрипт управления проектом. Он аналогичен django-admin.py, но использует настройки из settings.py для этого конкретного проекта, и может содержать дополнительные команды из приложений в проекте.

Чтобы не путать папку с кодом (hello) и главное приложение (тоже hello), можно переименовать папку с кодом в `hello_src` (суффикс `src` обозначает `source`), или сразу в `source` (англ. "источник", "исходник"):

```
(venv) .../hello_django$ mv hello source
```

Работа проекта на Django не зависит от названия папки с кодом, поэтому такое переименование можно сделать безопасно для проекта. . После переименования структура проекта выглядит так:

```
hello_django
|-- source
|   |-- hello
|   |   |-- __init__.py
|   |   |-- settings.py
|   |   |-- urls.py
|   |   |-- wsgi.py
|   |-- manage.py
|-- venv
|-- requirements.txt
```

В дальнейшем мы будем придерживаться этого названия для папки с исходниками проекта в каждом проекте, чтобы избежать разногласий, о какой папке мы говорим.

Приложения

Проект на Django состоит из нескольких самостоятельных **приложений** (англ. **application**), каждое из которых является пакетом Python. Приложения нужны, чтобы отделить код, связанный с разными разделами или интерфейсами сайта друг от друга и позволить легко переносить какие-то однотипные приложения из сайта в сайт.

Например, в проекте могут быть следующие приложения:

- интерфейс администратора ("админка")
- приложение для аутентификации и авторизации пользователей
- веб-приложение
- API

Кроме того, в проекте присутствует главное приложение, связывающее между собой все остальные и содержащее файлы конфигурации (настройки), главный скрипт, который запускается при старте сервера и т.д. Оно обычно не содержит бизнес-логики, и по умолчанию называется так же, как сам проект.

Каждое приложение, кроме главного, может содержать код, описывающий классы для связи с базой данных (модели), код для обработки HTTP-запросов (представления), шаблоны HTML-страниц, статические файлы (CSS, JS, картинки), код для обработки форм, тесты и т.д.

Django содержит встроенные приложения, такие, как админка или приложение для аутентификации пользователей - они подключаются в проект по умолчанию. Список приложений в проекте находится в `settings.py` в константе INSTALLED_APPS. Библиотеки, работающие с Django, обычно предоставляют для проекта такие же приложения, которые подключаются в проект путём добавления записи о них в INSTALLED_APPS.

Django уже содержит несколько встроенных приложений: приложение для авторизации пользователей, админку и пользовательские сессии, поэтому перед запуском проекта нужно настроить базу данных. В Django это делается командой `migrate` (англ. "мигрировать"). При первом запуске команда `migrate` создаёт в базе данных необходимые для приложений таблицы и связи между ними:

```
(venv) ...$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
```

```
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
```

Если вы сейчас проверите содержимое папки `source`, то увидите файл базы данных `db.sqlite3`:

```
(venv) ...$ ls
db.sqlite3  hello  manage.py
```

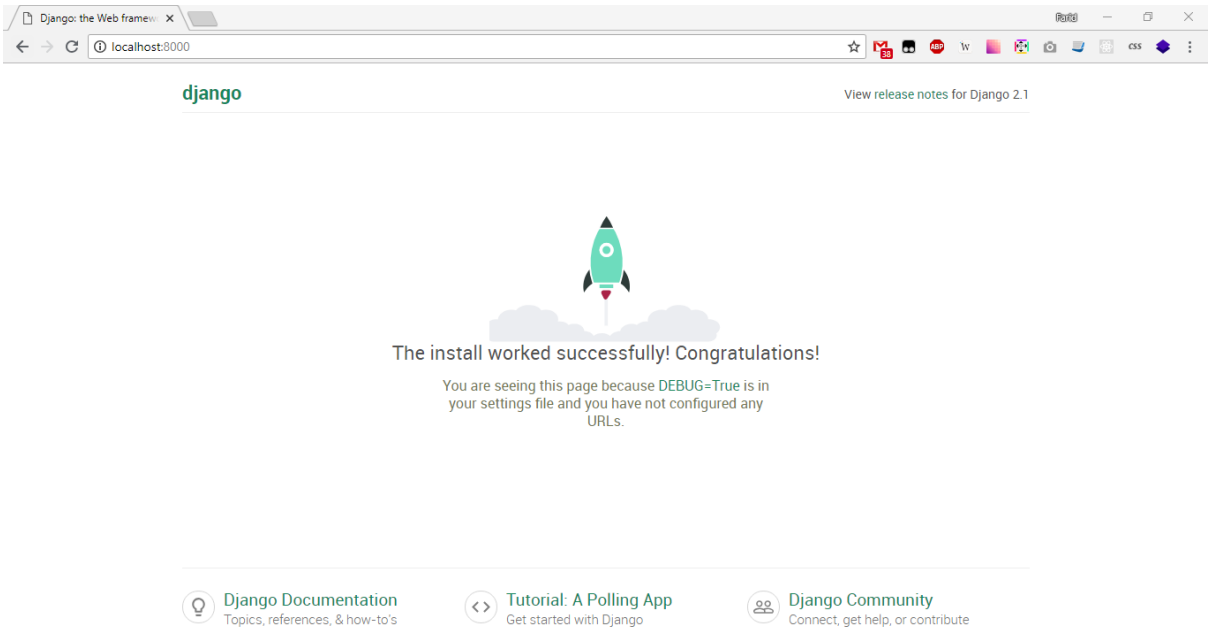
sqlite3 - это движок баз данных, база данных которого состоит из одного файла, работа с которым осуществляется локально, без запуска сервера и доступа по логину и паролю. Такая база данных удобна для разработки и тестирования, а также в мобильных приложениях.

Теперь можно запустить локальный сервер командой `runserver`:

```
(venv) ...$ python manage.py runserver 0:8000
```

Здесь `0:8000` обозначает адрес, запросы с которого слушает сервер. `0:8000` является сокращением от адреса `0.0.0.0:8000`. Адрес `0.0.0.0` обозначает, что сервер может принимать запросы, в которых указано любое имя хоста, а не только локальные адреса. В браузере вы можете обращаться к серверу по адресам `localhost` или `127.0.0.1`. За счёт настройки 0.0.0.0 вы также можете обратиться к серверу по адресу в локальной сети или сети Интернет.

Если вы откроете в браузере адрес `localhost:8000`, то увидите стартовую страницу Django:



Как создать приложение

Сейчас наш проект состоит из одного приложения - главного, `hello`. Создадим ещё одно приложение для логики сайта под названием `webapp` (сокращение от web application) командой `startapp`:

```
(venv) ...$ python manage.py startapp webapp
(venv) ...$ ls
db.sqlite3  hello  manage.py  webapp
```

Пока проект ничего не знает о новом приложении, поэтому добавьте его в конец списка установленных приложений - INSTALLED_APPS в настройках проекта (`hello/settings.py`):

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

Первая страница

Пока что наше приложение ничего не делает. Для примера добавим в него главную страницу, которая будет выводить приветственный текст.

Сначала нужно создать в приложении `webapp` папку `templates`, в которой будет храниться *шаблон* страницы, представляющий из себя файл .html с вёрсткой. Кроме разметки HTML шаблоны Django могут содержать специальные теги для генерации разметки и контента, с которыми мы познакомимся позже.

В папке `templates` создайте файл `index.html` со следующим кодом:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Hello</title>
    <style type="text/css">
        body {
            font-family: Arial, sans-serif;
            font-size: 16px;
            color: #444444;
        }

        .container {
            margin: 0 auto;
            width: 100%;
```

```
            max-width: 900px;
            text-align: center;
        }

    </style>
</head>
<body>
<div class="container">
    <p>This is my first Django project!</p>
    <p>Hello World!</p>
</div>
</body>
</html>
```

Первое представление

Шаблон ещё не является в данном случае страницей и не предназначен для прямого открытия в браузере. Чтобы показать страницу пользователю, нужно вернуть её в качестве *ответа* на *запрос* пользователя.

Запрос (англ. **request**) - это обращение *клиента*, например браузера, к *серверу*. Запрос обычно выполняется по определённому адресу или URL (вы видите его в адресной строке браузера) и может содержать дополнительные параметры или данные, включая файлы. Также запросы обычно содержат разную техническую информацию, передаваемую в *заголовках* (англ. headers)

Ответ (англ. **response**) - это ответ *сервера клиенту*. Сервер может возвращать в ответе текст, данные в различных форматах, html-страницы, картинки, файлы и т.д.

Структура запросов и ответов и правила обмена обычно описаны неким стандартом - **протоколом**. В данном случае - это протокол **HTTP** - HyperText Transfer Protocol.

За обработку HTTP-запросов в Django отвечают *представления* - специальные функции или классы, в которых содержится логика по генерации (*рендерингу*) HTML-страниц путём заполнения шаблонов данными.

Откройте файл `webapp/views.py` и напишите в нём следующий код:

```
from django.shortcuts import render

def index_view(request):
    return render(request, 'index.html')
```

Здесь:

- `index_view` - представление, функция, которая принимает объект запроса - `request`. Также представления могут принимать параметры запросов, но в данном случае их нет.
- `request` - объект Python, представляющий запрос.
- `render` - одна из функций Django, которая заполняет указанный шаблон данными и возвращает объект-ответ с полученной страницей.

Роутинг (маршрутизация)

Если сейчас запустить сервер Django и зайти на 127.0.0.1:8000, то вы по-прежнему будете видеть стартовую страницу Django, а не только что созданную новую страницу. Это происходит потому что Django пока не знает, какое представление должно обрабатывать запросы по этому адресу.

Роутинг (или маршрутизация, от англ **route** - "маршрут") - это связывание путей на сайте (англ. "**path**"), которые указаны в адресах запросов, и представлений, которые обрабатывают эти запросы.

Чтобы связать наше новое представление `index_view` и адрес запроса, напишите следующий код в файле `hello/urls.py`:

```
from django.contrib import admin
from django.urls import path
from webapp import views as webapp_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', webapp_views.index_view)
]
```

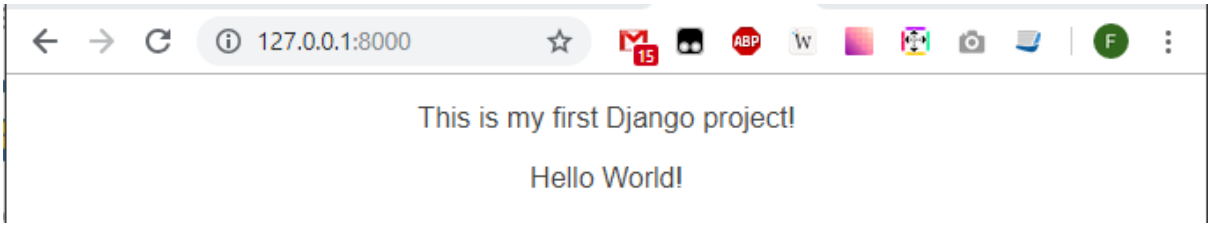
Здесь:

- `admin` - приложение для администрирования сайта, встроенное в Django
- `django.urls` - модуль django, где содержатся функции для работы с маршрутизацией.
- `path` - функция, которая регистрирует в *диспетчере* адресов (URL диспетчер) Django определённые пути на сайте и связанные с ними представления
- `''` - путь, где будет зарегистрировано новое представление `index_view`.
- `webapp_views.index_view` - представление, которое требуется подключить.

Обратите внимание, что при внесении изменений в код проекта, после их сохранения тестовый сервер перезапускается автоматически. Если он завис (например, в коде была ошибка и очередной перезапуск не удался), то перезапустите его вручную:

- Нажмите в терминале, где запущен сервер, Ctrl+C.
- Снова выполните команду `python manage.py runserver 0:8000`.

После перезапуска вы увидите вашу новую страницу:



Структура проекта

Все проекты на Django имеют определённую структуру. В первую очередь, это необходимо для того, чтобы программисты и сам Django могли находить компоненты проекта на заранее известных местах. Сейчас ваш проект имеет следующую структуру:

`hello_django`	корень проекта
`|-- source`	исходный код проекта
`|  |-- db.sqlite3`	база данных проекта в формате sqlite
`|  |-- hello`	главное приложение проекта
`|  |  |-- __init__.py`	инициализация пакета `hello`
`|  |  |-- settings.py`	настройки всего проекта
`|  |  |-- urls.py`	главная конфигурация адресов проекта, "роутер"
`|  |  |-- wsgi.py`	конфигурация для запуска приложения с wsgi-сервером
`|  |-- manage.py`	скрипт для управления проектом
`|  |-- webapp`	приложение `webapp`
`|  |  |-- admin.py`	настройки приложения `webapp` для панели администратора
`|  |  |-- apps.py`	общие настройки приложения `webapp` в проекте
`|  |  |-- __init__.py`	инициализация пакета `webapp`
`|  |  |-- migrations`	миграции приложения `webapp`
`|  |  |  |-- __init__.py`	инициализация пакета `migrations`
`|  |  |-- models.py`	модели приложения `webapp`
`|  |  |-- templates`	шаблоны приложения `webapp`
`|  |  |  |-- index.html`	шаблон главной страницы
`|  |  |-- tests.py`	тесты приложения `webapp`
`|  |  |-- views.py`	представления приложения `webapp`
`|-- venv`	виртуальное окружение
`|-- requirements.txt`	файл со списком зависимостей проекта

Обратите внимание на то, что виртуальное окружение не обязательно хранить вместе с проектом. Можно выделить специальную папку в вашей системе отдельно от проектов, например `environments`, где будут храниться виртуальные окружения для разных проектов. В этом случае можно напрямую создавать проект в `projects` (или другой папке, где вы храните проекты) с помощью `django-admin.py`, и не заводить отдельную внешнюю папку для объединения проекта и его окружения (в примере - `hello_django`). `requirements.txt` и другие технические файлы при их наличии будут храниться прямо в папке с исходниками (у нас - `source`) или какой-то папке внутри неё.

Дополнительно

1. https://www.djangoproject.com/ - официальный сайт Django
2. https://docs.djangoproject.com/en/2.2/ - документация по Django на английском
3. https://django.fun/docs/django/ru/2.2/ - документация по Django на русском
4. https://djangosites.org/ - список веб-сайтов на Django (разных)
5. https://www.wappalyzer.com/technologies/django - ещё один список веб-сайтов, использующих Django