Лекция №14

Django

- Общая информация
- Принципы и методики
- Архитектурная концепция
- Как работает интернет-сайт
- Проект веб-сайта
- Запуск прототипа
- Приложения
- Представления
- Представления и шаблоны
- Миграции
- Модели
- Модели и базы данных
- Superuser
- Практика

Общая информация

Django (джанго) - бесплатный и свободный фреймворк для веб-приложений, написанный на Python. Фактически, это набор компонентов, которые помогают разрабатывать веб-сайты быстро и просто.

Каждый раз при разработке веб-сайтов требуются похожие компоненты: способ аутентификации пользователей (вход, выход, регистрация), панель управления сайтом, формы, инструменты для загрузки файлов и т. д.

Как раз для решения таких однотипных проблем при веб-разработке и были созданы специальные веб-фреймворки (Django и другие), которые предлагают готовые шаблоны для использования.

Веб-фреймворк Django используется в таких крупных и известных интернет-проектах, как Instagram, Mozilla, Pinterest, YouTube, Google и др.

Также Django используется в качестве веб-компонента в различных проектах, таких как Graphite — система построения графиков и наблюдения, FreeNAS — свободная реализация системы хранения и обмена файлами и др.

Принципы и методики Django

Django поощряет свободное связывание и строгое разделение частей приложения. Если следовать этой философии, то легко вносить изменения в одну конкретную часть приложения без ущерба для остальных частей.

Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий этого фреймворка от некоторых других (например, Ruby on Rails). Один из основных принципов фреймворка — DRY (Don't repeat yourself).

Также, в отличие от других фреймворков, обработчики URL в Django конфигурируются явно при помощи регулярных выражений.

Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.

Архитектурная концепция

Три аспекта — логика доступа к данным, бизнес-логика и логика отображения — совместно составляют концепцию, которую называют шаблоном Модель-Представление-Контроллер (Model-View-Controller, MVC) архитектуры программного обеспечения.

В этой концепции термин «Модель» относится к логике доступа к данным; термин «Представление» - к той части системы, которая определяет, что показать и как; а термин «Контроллер» - к управляющей части системы, которая определяет какое представление надо использовать, в зависимости от пользовательского ввода, по необходимости получая доступ к модели.

Архитектурная концепция

Django следует модели MVC достаточно близко, т.е., может быть назван MVC совместимой средой разработки. Вот примерно как M, V и C используются в Django:

M - доступ к данным, обрабатывается слоем работы с базой данных.

V - определяет какие данные получать и как их отображать, обрабатывается представлениями и шаблонами.

C - выбирает представление в зависимости от пользовательского ввода, обрабатывается самой средой разработки, следуя созданной разработчиком схеме URL, и вызывает соответствующую функцию Python для указанного URL.

Контроллер классической модели MVC примерно соответствует уровню, который в Django называется Представление (View), а презентационная логика Представления реализуется в Django уровнем Шаблонов (Template). Из-за этого архитектуру Django часто называют «Модель-Шаблон-Представление» (MTV).

Как работает интернет-сайт

Как работает интернет-сайт

Веб-сайт фактически представляет собой группу файлов, сохраненных на жестком диске. Эти файлы содержат компьютерный код, называемый HTML. HTML-файл, как любой другой тип файла, необходимо хранить на жестком диске. Веб-сервера обслуживают обращения к этим файлам по протоколу HTTP. Простейшая аналогия - почта. Если нужно получить ответ от кого-то (например, от сайта google.com), то нужно послать письмо с запросом. Тогда в ответ придет письмо с ответом (контентом сайта). Вместо адресов с названием улицы, города, почтового индекса и названия страны используются IP-адреса. Компьютер сначала отправляет DNS (Domain Name System) запрос, чтоб перевести имя google.com в IP-адрес. Это работает примерно как в случае старых телефонных книг, где можно по имени человека, с которым нужно связаться, найти его номер телефона и адрес. В роли таких книг выступают DNS-сервера, которые возвращают компьютеру IP-адрес, соответствующий имени в DNS-запросе. Компьютер отправляет на это IP-адрес HTTP-запрос. Запрос приходит на веб-сервер и передается на обработку установленной на сервере системе.

Проект веб-сайта

Задача back-end разработчика состоит в создании системы на веб-сервере, которая будет обрабатывать поступающие запросы. Фреймворк Django позволяет упростить эту задачу.
Django устанавливается при помощи pip:

```
// устанавливаем Django
$ pip3 install django
// проверяем версию
$ python3 -c "import django; print(django.get_version())"
```

Django включает несколько утилит, позволяющих создать каркас будущего сайта:

```
// создаем проект в текущей директории
$ django-admin startproject coolsite .
```

Проект веб-сайта

Команда startproject создает систему директорий:

```
cool_site/
    manage.py
    coolsite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

- manage.py - скрипт, позволяющий взаимодействовать с проектом Django. каталог coolsite - это пакет Python нашего проекта; его название – это название пакета Python, которое будет использоваться для импорта чего-либо из проекта (например, coolsite.urls).
- coolsite/settings.py - настройки/конфигурация проекта.
- coolsite/urls.py - конфигурация URL-ов для нашего проекта; это контент всех Django-сайтов.
- coolsite/wsgi.py - точка входа в проект для WSGI-совместимых веб-серверов.

Настройка часового пояса

Для начала установим корректный часовой пояс на сайте. В файле settings.py находим строку, содержащую TIME_ZONE, и модифицируем ее в соответствии со нашим часовым поясом. Нам также необходимо добавить в настройки информацию о расположении статических файлов. В конце файла и после переменной STATIC_URL добавляем новую переменную - STATIC_ROOT.

```
TIME_ZONE = 'Europe/Moscow'
# ...
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

Дополнительная настройка для Windows (Django-2.1.4)

В прототипе сайта, предоставляемой Django для Windows существуют некоторые недоработки. Для их устранения в файле settings.py надо заменить MIDDLEWARE_CLASSES на MIDDLEWARE:

```
MIDDLEWARE_CLASSES = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

MIDDLEWARE = [
    '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',
    'django.middleware.security.SecurityMiddleware'
]
```

Запуск прототипа

Попробуем запустить получившийся прототип веб-сервера

```
$ python3 manage.py runserver
```

Можно попробовать зайти на сайт, набрав в браузере http://127.0.0.1:8000.
Важное замечание: НИКОГДА НЕ используйте этот сервер на реальном сайте. Он создан исключительно для разработки.
Кстати, посмотрите на настройку базы данных, используемой по умолчанию (понятно, что сайт без БД практически бесполезен), в файле settings.py (используется sqlite, работающая через файл db.sqlite3):

```
# Database
# https://docs.djangoproject.com/en/2.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Приложения в проекте Django

Теперь, после создания окружения для нашего проекта, можно заняться его наполнением. Django позволяет создать структуру, в которой будут выполняться созданные разработчиком web-приложения, которые предоставляют посетителям сайта определенный функционал – например, web-блог, просто хранилище каких-то записей или приложение для голосования. Проект – это совокупность приложений и конфигурации сайта. Проект может содержать несколько приложений. Приложение может использоваться несколькими проектами.

Приложения в проекте Django

Для создания приложения, находясь в том же каталоге, что и файл manage.py, выполняем команду:

```
// создаем каталог приложения coolapp
$ python3 manage.py startapp coolapp

coolapp/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

Создание представления

Для создания представления откроем файл coolapp/views.py и добавим в него следующий код:

```
# файл coolapp/views.py
from django.http import HttpResponse


def index(request):
    return HttpResponse("Любой текст")
```

Создадим файл coolapp/urls.py и добавим в него следующий код, чтоб все обращения к coolapp/urls приводили к вызову views.index:

```
from django.urls import path
from . import views


urlpatterns = [
    path('', views.index, name='index'),
]
```

Создание представления

Следующий шаг – добавление ссылки на coolapp.urls в главной конфигурации URL-ов. В coolsite/urls.py добавляем include(coolapp.urls) в список urlpatterns: можно использовать объект url (устаревший вариант, импортируется из django.conf.urls.include) или path (импортируется из django.urls). Теперь при обращении к coolapp в строке адресации url будут использоваться паттерны из coolapp.urls. Получается следующий код:

```
from django.conf.urls import include, url
from django.urls import path
from django.contrib import admin


urlpatterns = [
    path('admin/', admin.site.urls),
    path('coolapp/', include('coolapp.urls')),
]
```

Проверяем, что получилось, набрав в браузере http://127.0.0.1:8000/coolapp/
Вопрос: как изменить coolsite/urls.py так, чтобы наш текст открывался по url http://127.0.0.1:8000/, а не по http://127.0.0.1:8000/coolapp/?

Представление и шаблоны

В получившейся реализации представление по модели MVC (то, что возвращается функцией views.index) практически неразрывно с контроллером (кодом, управляющим этим представлением). Чтоб отделить представление от кода, нужно воспользоваться системой шаблонов Django.

Создаем каталог templates в каталоге приложения coolapp. Django будет искать шаблоны в этом каталоге.

В только что созданном каталоге templates, создаем каталог coolapp, и в нем создаем файл index.html (coolapp/templates/coolapp/index.html). Загрузчик шаблонов app_directories работает так, что к созданному шаблону можно обращаться посредством coolapp/index.html.

```
// пример содержимого index.html
<h1>Hello everybody!</h1>
```

Представление и шаблоны

Поправим код views.index:

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

И добавляем приложение в settings.py, чтоб для него можно было использовать шаблоны:

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

Миграции

Как уже было отмечено ранее, важнейшим компонентов сайта является база данных. По умолчанию, фреймворк предоставляет движок sqlite, что указывается в файле settings.py. Таблица INSTALLED_APPS в этом же файле содержит все используемые приложения. Некоторые приложения используют минимум одну таблицу в базе данных, поэтому нам необходимо создать соответствующие таблицы перед тем, как их использовать. Для этого выполним следующую команду:

```
$ python3 manage.py migrate
```

Команда migrate проверяет настройку INSTALLED_APPS и создает все необходимые таблицы в базе данных, указанной в settings.py, применяя миграции, которые находятся в приложении.

Миграции

- Миграции похожи на систему контроля версий (как Git, но только для базы данных - БД).
- Они позволяют команде разработчиков легко изменять схему БД приложения и делиться этими изменениями.
- Django использует миграции для переноса изменений в моделях (добавление поля, удаление модели и т.д.) на структуру базы данных.
- Грубо говоря, команда migrate позволяет накатить изменения таблиц на текущую БД, а команда makemigrations позволяет сохранить текущее состояние БД.
- Файлы с миграциями находятся в каталоге “migrations” приложения. Они являются частью приложения и должны распространятся вместе с остальным кодом приложения.

Модели

Ранее были рассмотрены уровни контроллер/представление (или представление/шаблон). Теперь рассмотрим создание моделей. Модель - это основной источник данных. Она содержит набор полей и задает поведение хранимых данных. Django следует принципу Don't repeat yourself (DRY). Чтоб следовать этому принципу, следует определять модели в одном месте, где все поля и таблицы выражаются средствами Python. Для приложения coolapp модели можно указать в файле coolapp/models.py

```
class Film(models.Model):
    name = models.CharField(max_length=200)
    desc = models.TextField()
    pub_date = models.DateTimeField('date published', auto_now_add=True)
```

Возможные типы полей: BigIntegerField, BooleanField, EmailField, FileField, ImageField, URLField и многие другие.

Модели и базы данных

Созданные поля нужно активировать:

- создать структуру базы данных (CREATE TABLE) для приложения,
- создать Python API для доступа к данным объектов.

```
$ python3 manage.py makemigrations coolapp
```

Выполняя makemigrations, мы говорим Django, что внесли некоторые изменения в наши модели (в нашем случае мы создали несколько новых) и хотели бы сохранить их в миграции. Содержимое папки migrations, при этом, изменится.

Модели и базы данных

Команда sqlmigrate принимает название миграции в качестве параметра и возвращает SQL запрос:

```
$ python3 manage.py sqlmigrate coolapp 0001

BEGIN;
--
-- Create model Film
--
CREATE TABLE "coolapp_film"
("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
"name" varchar(200) NOT NULL, "desc" text NOT NULL, "pub_date" datetime NOT
NULL);
COMMIT;
```

Снова вызываем migrate:

```
$ python3 manage.py migrate
```

С помощью SQL консоли или браузера можно посмотреть, что в результате запишется в базу данных.

```
$ sqlite3
$ sqlite> attach 'db.sqlite3' as db1;
```

Модели и базы данных

Чтобы работать с созданными моделями через Python, достаточно запустить консоль Python в той же директории, где лежит manage.py и выполнить код:

```
import os
os.environ['DJANGO_SETTINGS_MODULE'] = 'coolsite.settings'
import django
django.setup()
```

Теперь можно обращаться к базе данных напрямую, используя возможности Django. Можно добавлять, изменять, удалять данные, но разумеется не структуры базы (т.е. таблицы, на которые отображаются модели). Чтоб записи модели выводились в читаемом виде, желательно определить у модели метод __repr__, возвращающий строку.
Отображение классов моделей написанных на Python в таблицы базы данных обеспечивается технологией Django ORM.

Модели и базы данных

```
# Дальше можно работать с моделями
>>> from coolapp.models import Film
>>> Film.objects.all()
<QuerySet []>
>>> f = Film(name='Alien', desc='Old fantastic')
>>> f
<Film: Film object (None)>
>>> f.id
>>> f.name
'Alien'
>>> f.desc
'Old fantastic'
>>> f.pub_date
>>> f.save()
>>> f.id
1
>>> f.pub_date
datetime.datetime(2018, 12, 4, 1, 51, 37, 386401, tzinfo=<UTC>)
>>> f.name = 'Alien 2'
>>> f.save()
>>> Film.objects.all()
<QuerySet [<Film: Film object (1)>]>
```

Модели и базы данных

```
>>> f.delete()
(1, {'coolapp.Film': 1})
>>> Film.objects.all()
<QuerySet []>
>>> f = Film(name='Alien', desc='Old fantastic')
>>> f
<Film: Film object (None)>
>>> f.save()
>>> Film.objects.first()
<Film: Film object (2)>
>>> Film.objects.get(id=1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/linuxuser/.local/lib/python3.6/site-
packages/django/db/models/manager.py", line 82, in manager_method
    return getattr(self.get_queryset(), name)(*args, **kwargs)
  File "/home/linuxuser/.local/lib/python3.6/site-
packages/django/db/models/query.py", line 399, in get
    self.model._meta.object_name
coolapp.models.DoesNotExist: Film matching query does not exist.
>>> f.id
2
>>> Film.objects.get(id=2)
<Film: Film object (2)>
```

Superuser

Для управления сайтом необходимо завести аккаунт суперпользователя (администратора):

```
$ python3 manage.py createsuperuser
```

Добавляем приложение coolapp в интерфейс администратора:

```
# файл coolapp/admin.py
from django.contrib import admin
from .models import Film

admin.site.register(Film)
```

Снова запускаем наш веб-сервер и через браузер заходим в административную панель сайта: http://127.0.0.1:8000/admin/

Практика

1. У нас получился сайт с одним view и шаблоном, посвященный описанию фильмов. Какие еще могут быть у него странички? Добавьте необходимое количество views и соответствующие html шаблоны для них. В шаблонах напишите простенький html код с текстом, соответствующим данному view (например, шаблон главной странички должен содержать описание сайта и т. д.). Достаточно условного описания и простого оформления.
2. * Самостоятельно изучите язык шаблонов django (например, тут https://docs.djangoproject.com/en/1.7/topics/templates/) и попробуйте в функции render в вашем view прокинуть объекты из питоновского кода внутрь вашего html шаблона и отобразить их, например, так:

```
# это во views.py
def index(request):
    return render(request, 'coolapp/index.html', {'sitename': 'ABOUT FILMS'})

# это в coolapp/index.html
<h1>HELLO, You on {{ sitename }}</h1>
```

Практика

3. Заполните через админку табличку фильмов и попробуйте отобразить ее содержимое в шаблонах. Например, в зависимости от URL, если человек запрашивает http://127.0.0.1:8000/films/1, то ему должна быть показана информация по фильму с id==1. Также надо предусмотреть шаблон со списком всех фильмов.

```
# можно использовать следующий паттерн URL
url(r'^(?P<question_id>[0-9]+)/$', views.detail, name='detail')
```