Фреймворк Django

Что такое Django?

Django - это высокоуровневый фреймворк для веб-приложений на языке Python. Он был создан в 2005 году и с тех пор активно развивается и обновляется сообществом разработчиков по всему миру.

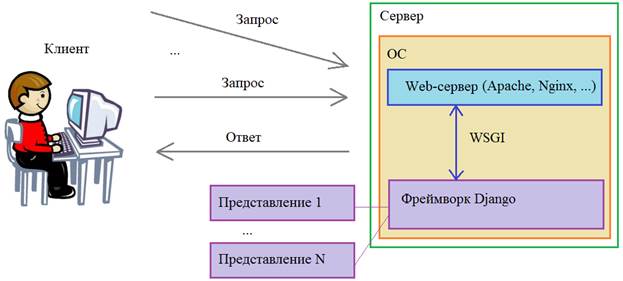
Но первый вопрос, почему именно Django? Наверное, этих фреймворков очень много и это действительно так. Однако именно Django завоевал огромную популярность благодаря простоте разработки даже очень сложных сайтов, например, таких как youtube, google search, dropbox и instagram. Все они сделаны с использованием фреймворка Django. И давайте теперь разберемся, для чего он вообще нужен. Если очень кратко, то для обеспечения работы сайтов на стороне сервера. То есть, это инструмент для бэкенд разработчиков. А если немного подробнее, то процесс взаимодействия пользователя с сайтом можно представить следующим образом:

Рисунок \_. Взаимодействие пользователя с сайтом.

Сначала от клиента поступает запрос, например, он вводит в браузере адрес https://youtube.com. Информационные пакеты начинают двигаться в сторону сервера с сайтом Youtube. Вопрос о том, как они это делают, мы оставим в стороне, это называется маршрутизацией в сети Интернет. Главное, что они доходят до сервера (то есть, компьютера), где расположен Youtube. На этом компьютере установлено специальное программное обеспечение – Web-сервер. Очень часто используют уже готовые разработки с открытым кодом: Apache и Nginx. Реже какие-либо свои программные продукты. Так вот, этот Web-сервер постоянно «слушает» каналы связи и в момент поступления запроса от пользователя должен перенаправить его на обработку соответствующему сайту (так как на одном сервере может находиться множество сайтов). Сайты способны по-разному обрабатывать входящие запросы, например, используя PHP и CGI скрипты, или же могут использоваться фреймворки, значительно упрощающие обработку запросов от клиентов сайта. В частности, Django – это и есть такой фреймворк, значительно упрощающий написание скриптов на языке Python. А взаимодействие с сервером происходит по интерфейсу WSGI, который передает обработку запроса в Django. При этом отрабатывает определенное представление, отвечающее за текущий запрос, и результатом обработки, как правило, является HTML-страница, которая сначала передается Web-серверу также по WSGI-интерфейсу, а он уже передает страницу конечному пользователю. Страница отображается в браузере клиента, и он видит заветный сайт Youtube. Вот так в очень упрощенном виде происходит взаимодействие между пользователем, сервером и фреймворком. WSGI (Web Server Gateway Interface) — стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера, и самим веб-сервером.

Далее, мы установим интегрированную среду программирования PyCharm, разработанную специально для написания программ на Python. Она довольно удобна для разработки программ с использованием Django, поэтому все дальнейшие действия мы будем производить в PyCharm.

\_ Установка и настройка Django

Так как разработка велась на операционной системе MacOS, все команды, используемые в терминале, будут записанные в этом проекте для данное ОС.

Python помогают решить проблему управления зависимостями проектов. Если у вас есть несколько проектов на Python, то вероятность того, что вам придется работать с разными версиями библиотек или самого Python, очень высока. Но использование виртуальных сред позволяет создавать независимые группы библиотек для каждого проекта, что предотвращает конфликты между версиями и не дает одному проекту повлиять на другой. В Python уже есть модуль venv для создания виртуальных сред, который можно использовать как в разработке, так и в производстве.

Создаём виртуальное окружение в Linux или MacOS:

mkdir project

cd project

python3 -m venv venv

Далее активируем виртуальное окружение, чтобы все дальнейшие действия выполнялись внутри него: venv/bin/activate

Для установки Django рекомендуется использовать менеджер пакетов pip. Для этого необходимо открыть терминал и выполнить команду: pip3 install Django.

Помимо самого фреймворка будет установлено несколько обязательных зависимостей. В прочем сам Django содержит в себе около 15 пакетов, позволяющих решать большинство задач разработки без установки дополнительных пакетов и модулей.

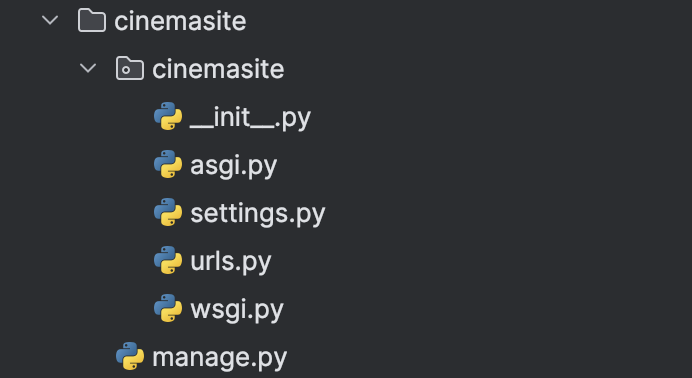
\_Создание проекта и запуск проекта

Первым шагом создания проекта мы определи папку, в которой будем вести разработку данного проекта. После выполним все действия, связанные с установкой и активации виртуального окружения и установкой самого Django.

Используя ядро Django, мы можем создавать множество разных сайтов под управлением Django в рамках текущего виртуального окружения. Чтобы посмотреть список команд ядра, достаточно в терминале записать: django-admin.

Для создания нового проекта в Django необходимо выполнить команду djangoadmin startproject <project\_name>. Здесь «project\_name», обычно, является доменным именем. Например, если мы собираемся располагать сайт на домене cinemasite, то пропишем: django-admin startproject cinemasite. Эта команда создаст структуру проекта, которая будет содержать все необходимые файлы и папки для работы с фреймворком.

Структура проекта Django имеет следующий вид:



Рассмотрим каждый из созданных файлов.

* manage.py - файл, который используется для управления проектом. С его помощью можно запустить сервер, создать миграции, создать суперпользователя и т.д.
* cinemasite/ - директория, которая содержит основные файлы проекта.
* \_\_init\_\_.py - файл, который сообщает Python, что директория myproject является пакетом.
* settings.py - файл, который содержит настройки проекта, такие как базы данных, шаблоны, статические файлы и т.д.
* urls.py - файл, который содержит маршруты приложения.
* asgi.py - файл, который используется для запуска проекта в ASGIсовместимых серверах.
* wsgi.py - файл, который используется для запуска проекта в WSGIсовместимых серверах.

Далее мы можем запустить наш сервер на нашем компьютере используя команду: python manage.py runserver. После запуска сервера можно открыть браузер и перейти по адресу <http://127.0.0.1:8000/>. Если все настроено правильно, то вы увидите страницу "The install worked successfully! Congratulations!" с ракетой.

Сервер Django используется лишь при разработке и тестировании проекта. Он способен раздавать статику в виде изображений, CSS файлов, JavaScript кода и т.п. Так же сервер отслеживает изменения, которые вы вносите в файлы проекта. После каждого такого изменения сервер автоматически перезагружается. Встроенный сервер нельзя использовать в продакшене. Он подходит только для разработки проекта.

При первом запуске сервера автоматически создается файл db.sqlite3 - файл БД SQLite3.

Дело в том, что по умолчанию Django использует именно такую СУБД. В дальнейшем, мы можем это изменить и указать любую другую СУБД, которую поддерживает данный фреймворк. Это может быть: PostgreSQL, MariaDB, MySQL, Oracle и SQLite.

Но вернемся к запуску сервера. Его можно запускать также и со следующими параметрами: python manage.py runserver 192.168.1.1:4000. Это значит что теперь мы можем перейти по ссылке <http://192.168.1.1:4000/> и увидеть наш проект.

\_MVT концепция. Создание первого приложения

Концепция Модель-Представление-Шаблон (MVT) является основой фреймворка Django. Она разделяет приложение на три основных компонента: модель (Model), представление (View) и шаблон (Template).

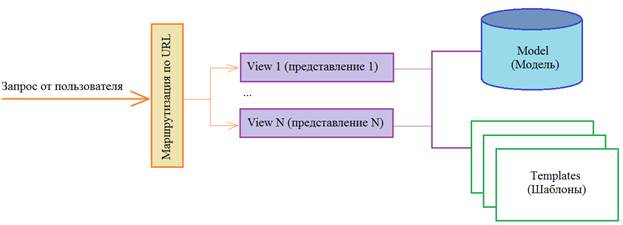


Рисунок \_. MVT модель.

Модель - это объект, который представляет данные в приложении и отвечает за их хранение и манипуляцию. Модели Django обычно соответствуют таблицам в базе данных. Представление - это компонент, который отвечает за обработку запросов и формирование ответов на основе данных из моделей. В Django представления обычно реализуются в виде функций или классов.

Шаблон - это компонент, который отвечает за отображение данных на странице. Шаблоны Django используют язык шаблонов, который позволяет вставлять данные из моделей и представлений в HTML-код.

Отдельно стоит упомянуть маршруты. Маршруты в Django играют роль посредника между пользователем и представлением. Они определяют, какой URL-адрес должен быть связан с каким представлением. Когда пользователь запрашивает URL-адрес, Django использует маршруты для определения соответствующего представления и передачи запроса ему. Представление обрабатывает запрос и формирует ответ, который затем возвращается пользователю.

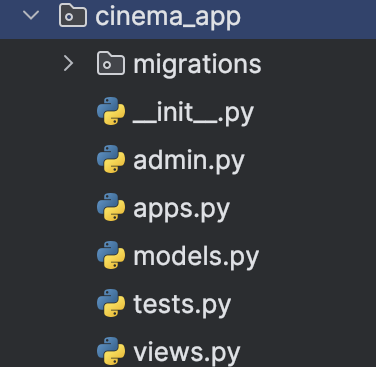
Добавление первого приложения.

Согласно философии Django мы должны создать новое приложение в рамках нашего сайта. Что это за приложение и зачем оно вообще нужно? Разработчики фреймворка решили, что каждая самостоятельная часть сайта должна представляться в виде своего отдельного приложения. Например, создавая информационный сайт, мы должны будем определить приложение для отображения страниц этого сайта по определенным запросам. Далее, к нам приходит руководитель проекта и сообщает, что еще нужно реализовать форум на сайте. И, так как это функционально независимая часть сайта, то мы создаем еще одно приложение для форума. Затем, руководитель почесал свою репу и вспомнил, что еще нужно сделать раздел с опросом пользователей по разным тематикам. И на сайте появляется еще одно приложение – для опроса. И так далее. Каждая логически и функционально независимая часть сайта предполагает его реализацию в виде отдельного приложения.

Приложения в Django следует реализовывать максимально независимыми, в идеале – полностью независимыми, чтобы в дальнейшем мы могли их просто скопировать в другой сайт и там оно сразу же начинало бы работать. Это не всегда удается, но к этому нужно стремиться.

Итак, давайте создадим в нашем сайте первое приложение, которое возьмет на себя базовый функционал, то есть, оно и будет являться ядром нашего сайта. Для этого мы откроем терминал и, находясь в каталоге cinemasite/, выполню команду: python manage.py startapp cinema\_app. Окончание app не является обязательным. Приложение может иметь любое имя, которое отражает его назначение. Но наличии суффикса app позволяет быстрее отличить пакет приложения от другого каталога внутри проекта.

Структура проекта после создания приложения имеет следующий вид:



После выполнения команды у нас в проекте появилась еще одна папка – cinema\_app, которая уже содержит несколько файлов, в том числе, и файл \_\_init\_\_.py. Следовательно, приложение в Django реализуется как пакет языка Python. Также здесь присутствует одна вложенная папка migrations для хранения миграций БД нашего приложения. Подробнее о ней мы поговорим позже. Остальные файлы имеют следующее назначение:

* admin.py – для настройки админ-панели сайта (админ-панель поставляется совместно с Django и каждый сайт может сразу ее использовать);
* apps.py – для настройки (конфигурирования) текущего приложения;
* models.py – для хранения ORM-моделей для представления данных из базы данных;
* tests.py – модуль с тестирующими процедурами;
* views.py – для хранения представлений (контроллеров) текущего приложения.

После создания приложения его необходимо зарегистрировать в проекте нашего сайта, чтобы фреймворк Django «знал» о его существовании и корректно с ним работал. Для этого нужно перейти в пакет конфигурации сайта (sitewomen), открыть файл settings.py и в списке INSTALLED\_APPS прописать новое приложение. В нем уже прописаны несколько стандартных приложений самого фреймворка и к ним мы просто добавим свое:

1. INSTALLED\_APPS = [
2. …
3. cinema\_app,
4. ]

Этого вполне достаточно и все будет работать, но в действительности Django обращаясь к этому пакету находит файл apps.py, откуда и берет настройки приложения из класса CinemaAppConfig. Чтобы в дальнейшем каждый раз не конкретизировать этот путь, я пропишу его сразу в списке приложений:

1. INSTALLED\_APPS = [
2. …
3. cinema\_app.apps.CinemaAppConfig,
4. ]

\_Маршрутизация и функции представления

Следующий пункт работы - создать представление. Так же их называют view или просто “вьюшками”.

Давайте для начала создадим обработчик главной страницы приложения cinema\_app с помощью функции представления. Как я уже отмечал, представления в Django можно реализовывать или в виде функций, или в виде классов. Но для лучшего понимания механики обработки запросов мы воспользуемся именно функцией. Эта функция будет срабатывать при обращении к главной странице приложения cinema\_app, называться index и формировать ответ в виде простой строки:

1. def index(request):
2. return HttpResponse("Страница приложения cinema\_app.")

Здесь указывается первый обязательный параметр request – это ссылка на экземпляр класса HttpRequest, который содержит информацию о запросе, о сессии, куках и так далее. То есть, через переменную request нам доступна вся возможная информация в рамках текущего запроса. На выходе эта функция возвращает экземпляр объекта HttpResponse, который будет автоматически формировать нужный заголовок ответа, а содержимое будет представлено простой строкой.

Теперь нам нужно связать эту функцию представления с соответствующим URL-адресом. Для настройки URL в Django необходимо определить маршруты (routes), которые будут связывать определенные URL с соответствующими представлениями (views) в приложении. Маршруты определяются в файле urls.py, который находится в корневой директории проекта и в директории приложения.

Открываем файл urls.py в корневой директории проекта Файл уже содержит комментарий с описанием трёх способов добавления маршрутов:

* функции представления
* классы представления
* подключение других файлов с настройками URL

Пропишем добавление файла конфигурации маршрутов из приложения  cinema\_app в проект:

1. from django.contrib import admin
2. from django.urls import path, include
3. urlpatterns = [
4. path('admin/', admin.site.urls), path('', include('cinema\_app.urls')),
5. ]

Создаем файл urls.py в директории приложения.

Убедитесь, что вы создаёте файл внутри каталога cinema\_app. Когда в вашем проекте будет несколько приложений, каждое будет иметь собственный urls.py с локальными маршрутами. А urls.py проекта будет включать (include()) их в глобальный список адресов. Внутри cinema\_app/urls.py пропишем следующий код:

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

Импортируем функцию path и файл с представлениями из текущего каталога (импорт через точку).

Функция path() принимает два аргумента: первый аргумент - это URL-адрес, а второй - это представление, которое будет обрабатывать запрос на этот URL. Также можно задать имя маршрута с помощью параметра name. Таким образом, настройка URL в Django позволяет определить маршруты для обработки запросов на определенные URL и связать их с соответствующими представлениями.