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**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | INSTALLED\_APPS = [      'django.contrib.admin',      'django.contrib.auth',      'django.contrib.contenttypes',      'django.contrib.sessions',      'django.contrib.messages',      'django.contrib.staticfiles',  ] |

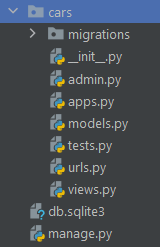
И, конечно, мы можем создавать свои приложения, которые реализуют определенный функционал. Для создания приложения в проекте применяется команда

#### python manage.py startapp название\_приложения

Так, создадим в проекте, созданном в прошлой теме или новом, первое приложение, которое будет называться **hello**. Для этого выполним в командной строке/терминале следующую команду:

##### python manage.py startapp cars

В результате в проекте Django будет создано приложение hello. И после выполнения этой команды мы увидим в проекте новую папку, которое будет хранить все файлы созданного приложения:



Рассмотрим вкратце его структуру:

папка **migrations**: предназначена для хранения миграций - скриптов, которые позволяют синхронизировать структуру базы данных с определением моделей

**\_\_init\_\_.py**: указывает интерпретатору python, что текущий каталог будет рассматриваться в качестве пакета

**admin.py**: предназначен для административных функций, в частности, здесь призводится регистрация моделей, которые используются в интерфейсе администратора

**apps.py**: определяет конфигурацию приложения

**models.py**: хранит определение моделей, которые описывают используемые в приложении данные

**tests.py**: хранит тесты приложения

**views.py**: определяет функции, которые получают запросы пользователей, обрабатывают их и возвращают ответ

Но пока приложение никак не задействуется. Его надо зарегистрировать в проекте Django. Для этого откроем файл **settings.py** и добавим в конец массива INSTALLED\_APPS наше приложение:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | INSTALLED\_APPS = [      'django.contrib.admin',      'django.contrib.auth',      'django.contrib.contenttypes',      'django.contrib.sessions',      'django.contrib.messages',      'django.contrib.staticfiles',      'hello',  ] |

В проекте может быть несколько приложений, и каждое из них надо добавлять таким вот образом.

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 **hello** к файлу **views.py**, который по умолчанию должен выглядеть следующим образом:

|  |  |
| --- | --- |
| 1  2 | from django.shortcuts import render  # Create your views here. |

Изменим код следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4 | from django.http import HttpResponse    def index(request):      return HttpResponse("Hello WORLD") |

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 WORLD") мы отправляем пользователю строку "Hello WORLD"

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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | 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 WORLD". Поэтому изменим файл **urls.py** следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | from django.urls import path  from hello import views    urlpatterns = [      path('', views.index, name='home'),  ] |

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 METANIT.COM":

