Django Framework

development backend web apps

For beginners

Урок 2. Структура и настройка проекта

Создание Django-проекта

Для создания нового проекта Django воспользуемся командой *startproject* утилиты *django-admin*. Данная утилита предназначена для решения разного рода административных задач, среди которых: управление миграциями, создание приложений в самом проекте, запуск сервера, создание кэша базы данных и многое другое.

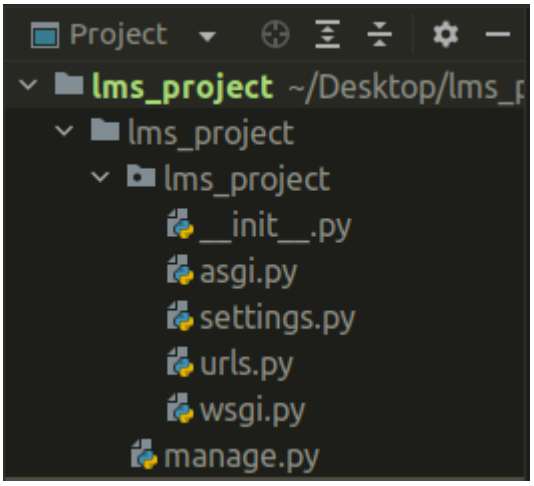
Формат ввода этой команды следующий:

django-admin startproject <имя проекта> <путь к проекту>

В том случае, если не будет указан путь к проекту, то Django создаст проект в текущей директории, откуда запускалась эта команда.

```
(venv) codeby@django:~/Desktop/lms_project$ django-admin startproject lms_project
(venv) codeby@django:~/Desktop/lms_project$
```

В результате исполнения данной команды ничего не будет выведено в консоль, но в папке, которую мы создали в прошлом уроке, мы увидим новую папку с тем же названием, состоящую из ряда файлов.



Подробнее о том, что это за файлы узнаем далее.

Список команд django-admin

Как говорилось ранее, утилита django-admin решает многие задачи. Рассмотрим её самые распространённые команды:

- *version* — позволяет узнать текущую версию Django.

```
(venv) codeby@django:~/Desktop/lms_project$ django-admin version
4.0
```

- *check <имя приложения_1> <имя приложения_2> <...>*

Позволяет проверить весь Django-проект на наличие самых распространённых ошибок (особенно полезно это делать при подготовке к развёртыванию нашего сайта в сети)

- *dbshell*

Эта команда предоставляет интерфейс командной строки для взаимодействия с базой данных. С её помощью мы можем выполнять SQL-запросы.

- *flush*

Очищает содержимое БД и перезапускает все обработчики после синхронизации. При этом таблица, указанная в последней миграции, остаётся нетронутой.

- *createcachetable*

Создаёт кеш-таблицу для совместного использования с кешем базы данных

- *sqlflush*

Распечатает в консоль все SQL-инструкции команды flush.

- *makemigrations <app_label> <...>*

Создаёт миграцию, генерируя при этом python-файл с произошедшими изменениями в модели. В атрибутах можно указать 1 или более названий приложений через пробел.

- migrate <app_name> <migration_name>

Изменяет состояние БД в соответствии с указанным списком моделей и миграций

- *app_name* — осуществляется миграция для нужного приложения до последней выполненной миграции.
- *migration_name* - приводит БД к состоянию последней миграции, при этом первые миграции в приложении не будут применяться.

- showmigrations

Позволяет показать все миграции, совершённые в проекте.

- squashmigrations <app> <start_migration_name> migration_name

Позволяет объединить, если это возможно, все миграции приложения *app* в меньшее число миграций до *migration_name*. Полученные миграции могут спокойно лежать рядом с необъединёнными.

- runserver

Запускает отладочный сервер на локальном ПК. Django настроен таким образом, что изначально проект запускается на следующем адресе: 127.0.0.1:8000 (или же на localhost:8000). Вам можно явно указать IP-адрес и номер порта. Но следует быть внимательным, так как если будет указан порт, занятый другим процессом, приложение не запустится.

Отладочный сервер использует WSGI (Web Server Gateaway Interface), прописанный в параметре WSGI_APPLICATION в файле настроек settings.py.

Этот сервер не рекомендуется к использованию для выпуска в production-режиме. Он не соответствует критериям безопасности.

- startapp <app_name>

Создаёт приложение с именем app_name, внутри созданного Django-проекта.

Django при старте проекта также создаёт файла manage.py, который по функциям отличается от утилиты django-admin только тем, что дополнительно создаёт переменную окружения DJANGO_SETTINGS_MODULE с указанием пути до модуля с настройками проекта. Это помогает быстро запустить проект.

```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'lms_project.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

Создание и структура приложений (apps)

Пришло время и нам создать наше приложение. Наша LMS будет содержать 2 приложения: приложение для управления логикой авторизации и приложение для управления онлайн-курсами.

Создадим же их!

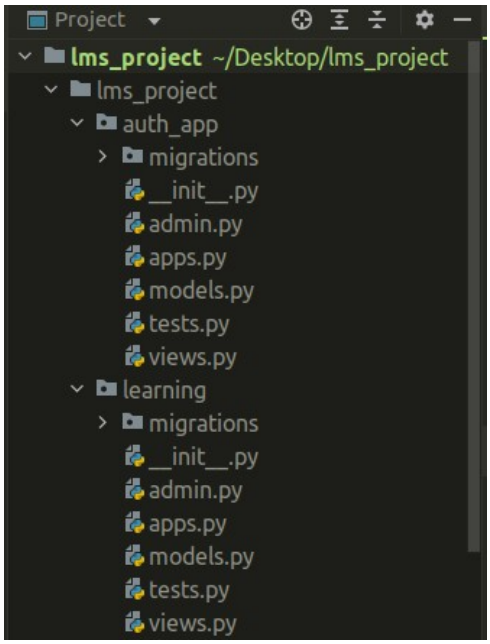
Для этого перейдём в папку нашего созданного проекта и введём команду: django-admin startapp learning и django-admin startapp auth_app. Данные команды необходимо вводить по отдельности.

```
(venv) codeby@django:~/Desktop/lms_project$ cd lms_project
(venv) codeby@django:~/Desktop/lms_project/lms_project$ django-admin startapp learning
```

```
(venv) codeby@django:~/Desktop/lms_project/lms_project$ django-admin startapp auth_app
(venv) codeby@django:~/Desktop/lms_project/lms_project$
```

Эта команда так же не выводит информации об успешности создания наших приложений, поэтому в этом мы можем сами убедиться, заглянув в окно IDE, показывающее дерево нашего проекта. И мы увидим созданные приложения, которые имеют свою иерархию и файлы.

Мы можем увидеть, что оба приложения имеют одни и те же файлы: admin.py, apps.py, models.py, tests.py и views.py. Для чего же они нужны?



- папка migrations хранит все созданные миграции модели

- admin.py — позволяет регистрировать созданные модели в админке Django

- apps.py — содержит объявление класса настроек приложения

```python
from django.apps import AppConfig


class AuthConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'auth'

```

В классе AuthAppConfig, унаследованном от базового класса Django AppConfig из модуля apps, установленные следующие атрибуты:

- default_auto_field — этот атрибут необходим для неявного добавления primary key (PK) поля в модель, если он не был указан ни в одном поле модели.
- name — путь к приложению относительно папки проекта. Само приложение мы можем зарегистрировать в файле настроек settings.py в параметре INSTALLED_APPS.

- models.py — хранит структуру всех моделей приложения и определяет логику обработки при совершении CRUD-операций.

- tests.py — содержит все тесты, применённые к нашему приложению для проверки корректности выполнения всех возложенных на него задач.

- views.py — отвечает за обработку логики представлений приложения. Соответственно, именно в этом файле мы определяем то, что увидит клиент.

Во всех этих файлах Django уже облегчил нам работу, сгенерировав импорты нужных модулей для реализации собственной логики каждого файла.

Файл admin.py

```
1  from django.contrib import admin
2
3  # Register your models here.
```

Файл models.py

```
from django.test import TestCase

# Create your tests here.
```

Файл tests.py

```python
from django.shortcuts import render

# Create your views here.
```

Файл views.py

```python
from django.db import models

# Create your models here.
```

Каталог настроек Django

Кроме файла settings.py в папке настроек проекта имеются и другие файлы: asgi.py, wsgi.py и urls.py.

- asgi.py (ASGI - Asynchronous Server Gateway Interface) нужен для взаимодействия асинхронных серверов с приложением Django. Является преемником WSGI.

```python
"""
ASGI config for lms_project project.

It exposes the ASGI callable as a module-level variable named ``applica

For more information on this file, see
https://docs.djangoproject.com/en/4.0/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'lms_project.settings')

application = get_asgi_application()
```

- wsgi.py нужен, наоборот, для взаимодействия с сайтом на Django в синхронном виде. Эти 2 файла различаются лишь способом получения объекта приложения.

```python
"""
WSGI config for lms_project project.

It exposes the WSGI callable as a module-level variable named ``applicat

For more information on this file, see
https://docs.djangoproject.com/en/4.0/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'lms_project.settings')

application = get_wsgi_application()
```

- urls.py — здесь прописываются все пути, по которым можно будет переходить по нашему сайту. В этом файле можно указывать все маршруты нашего сайта, но лучше делегировать эту обязанность конкретным приложениям, отвечающим за реализацию логики конкретного пут. Подробнее об этом мы поговорим в уроке, посвящённом маршрутизации.

Изначально, Django в качестве маршрута подключает маршруты только для взаимодействия с административным сайтом.

```python
"""lms_project URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
import ...

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

Файл настроек Django

Рассмотрение этого файла стоит вынести в отдельный пункт, так как этот файл очень важен для Django-проекта. Django, как говорилось в прошлом уроке, имеет монолитную архитектуру и все его настройки расположены в одном файле. Следовательно, нужно уметь так сконфигурировать наш проект, чтобы он был безопасным Ведь, если мы что-то настроим неправильно, то наш сайт может «упасть» и привести к нежелательным последствиям.

По умолчанию, все настройки проекта уже имеют свои начальные значения. Они представлены обычными объектами Python, такими как: словари (dict), списки (list) или же строки (str).

Давайте разберёмся, что означает каждая из настроек?

```
"""
Django settings for lms_project project.

Generated by 'django-admin startproject' using Django 4.0.

For more information on this file, see
https://docs.djangoproject.com/en/4.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.0/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
```

BASE_DIR указывает на текущий путь к проекту. Объект Path модуля pathlib позволяет разработчику решить проблему представления путей для различных операционных систем.

```
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-!2uszlkm76979^1)i1$$x_*ry&nih9*lamxt&brfv9b=7g!be!'
```

SECRET_KEY представляет строку со случайным набором символов. Он может использоваться подсистемой разграничения доступа для шифрования важных данных. При создании проекта он был автоматически сгенерирован утилитой django-admin. Этот секретный ключ не стоит предоставлять НИКОМУ в личный доступ, как и файл настроек проекта в целом (например, если Вы решите внести свой вклад в open-source проект, написанный на Django, не выкладывайте его в VCS — Version Control System, так как с помощью этого ключа определённый человек может совершить атаку на ваш сайт).

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

Булева (bool) переменная DEBUG устанавливает режим работы сайта.

Выделяют два режима работы: тестовый (при разработке) и production (когда мы собираемся развернуть свой проект на удалённом сервере). В том случае, когда сайт находится на стадии разработки, то лучше ставить True. Это Вам поможет быстро узнавать про ошибки при работе сайта, так как Django при генерации исключения генерирует полный traceback в виде обычной web-страницы и показывает её в самом браузере.

Напротив, в production-режиме, когда переменная DEBUG должна стать False, стандартному пользователю не нужно знать о багах, возникающих на сайте. Вместо этого придётся выводить страницу, наподобие «Error 404. Страница не найдена» и не более этого. Помимо прочего, в этом режиме действуют более строгие настройки безопасности.

```
28  ALLOWED_HOSTS = []
```

Параметр ALLOWED_HOSTS представляет список имён доменов, запросы от которых сайт может обрабатывать.

```
31  # Application definition
32
33  INSTALLED_APPS = [
34      'django.contrib.admin',
35      'django.contrib.auth',
36      'django.contrib.contenttypes',
37      'django.contrib.sessions',
38      'django.contrib.messages',
39      'django.contrib.staticfiles',
40  ]
```

Параметр INSTALLED_APPS представляет собой список всех зарегистрированных приложений во всём проекте. При создании проекта этот список принимает следующие значения:

- django.contrib.admin — админка Django

- django.contrib.auth — система разграничения доступа

- django.contrib.contenttypes — необходим для хранения всех моделей, реализованных во всём проекте (также используется всеми приложениями, перечисленными выше).

- django.contrib.sessions — нужен для работы с сессиями на стороне сервера

- django.contrib.messages — используется для показа и обработки всплывающих сообщений (например, всё в той же админке сайта)

- django.contrib.staticfiles — нужен для корректной работы со статическими файлами (допустим, нам нужно подгружать файлы со стилями (CSS), скрипты для придания динамики нашему сайту (JS) или же дополнительные картинки)

Давайте расширим список наших приложений. Вы не забыли, что мы создали два приложения: auth_app и learning. Их и нужно подключить следом за стандартными приложениями, указав конфигурационные классы каждого из приложений.

```
32  # Application definition
33
34  INSTALLED_APPS = [
35      'django.contrib.admin',
36      'django.contrib.auth',
37      'django.contrib.contenttypes',
38      'django.contrib.sessions',
39      'django.contrib.messages',
40      'django.contrib.staticfiles',
41      # Custom apps
42      'learning.apps.LearningConfig',
43      'auth_app.apps.AuthAppConfig',
44  ]
```

По стилю написания кода Django приветствует после завершения указания каждого приложения ставить знак «,», так как мы, когда будем добавлять новое приложение с новой строки, можем забыть её указать и, приложение не

запустится. Такая же практика используется при указании маршрутов приложений, о которых мы поговорим в отдельном уроке.

Вот мы и получили полный список наших приложений. В процессе разработки мы можем добавлять в этот список аналогичным образом и сторонние библиотеки, поддерживающие Django.

Параметр MIDDLEWARE (от en. посредник) отвечает за обработку каждого запроса перед его передачей view и пост-обработку, созданную тем же view, перед его отправкой клиенту в браузер.

```
45  MIDDLEWARE = [
46      'django.middleware.security.SecurityMiddleware',
47      'django.contrib.sessions.middleware.SessionMiddleware',
48      'django.middleware.common.CommonMiddleware',
49      'django.middleware.csrf.CsrfViewMiddleware',
50      'django.contrib.auth.middleware.AuthenticationMiddleware',
51      'django.contrib.messages.middleware.MessageMiddleware',
52      'django.middleware.clickjacking.XFrameOptionsMiddleware',
53  ]
```

Данный список также был сгенерирован при старте проекта.

Он содержит следующих посредников:

- django.middleware.security.SecurityMiddleware — обеспечивает защиту сайта от сетевых атак

- django.contrib.sessions.middleware.SessionMiddleware — обрабатывает серверные сессии на низком уровне

- django.middleware.common.CommonMiddleware — предварительная обработка запроса

- django.middleware.csrf.CsrfViewMiddleware — нужен для защиты сайта от подделки межсайтовых запросов (например, при отправке формы POST-методом)

- django.contrib.auth.middleware.AuthenticationMiddleware — служит для добавления в текущий запрос текущего пользователя

- django.contrib.messages.middleware.MessageMiddleware — используется для обработки всплывающих окон на низком уровне

- django.middleware.clickjacking.XframeOptionsMiddleware — служит для защиты сайта от атак типа clickjacking

Следует отметить, что каждый запрос проходит через все посредники сверху вниз по списку. В том случае, если хоть один middleware не вернул ни запрос, ни ответ, то запрос к следующим middleware не пойдёт. Он вернёт ответ только от тех, от которых запрос прошёл успешно.

Следующий параметр — ROOT_URLCONF представляет собой путь к модулю, в котором указаны маршруты всех приложений проекта.

```
55    ROOT_URLCONF = 'lms_project.urls'
```

Параметр TEMPLATES

```
57  TEMPLATES = [
58      {
59          'BACKEND': 'django.template.backends.django.DjangoTemplates',
60          'DIRS': [],
61          'APP_DIRS': True,
62          'OPTIONS': {
63              'context_processors': [
64                  'django.template.context_processors.debug',
65                  'django.template.context_processors.request',
66                  'django.contrib.auth.context_processors.auth',
67                  'django.contrib.messages.context_processors.messages',
68              ],
69          },
70      },
71  ]
```

Он является обычным списком словарей, каждый ключ элемента которого служит для этого:

- BACKEND — указывает на используемый шаблонизатор

- DIRS — список путей к папкам, где хранятся шаблоны

- APP_DIRS — переменная типа bool, равная True, позволяет разрешить шаблонизатору искать шаблоны в папках с именем templates, в противном случае ей придётся искать шаблоны в списке папок указанных в DIRS

- OPTIONS — значением является вложенный словарь, с помощью которого задаются дополнительные настройки шаблонизатора, такие как: вставка в шаблоны переменных, вставка в контекст шаблона переменной request, хранящую объект текущего запроса; добавление в контекст шаблона переменных user и perms и многое другое.

```
73    WSGI_APPLICATION = 'lms_project.wsgi.application'
```

Данная настройка, как и говорилось ранее, нужна для указания пути до объекта приложения.

```
76    # Database
77    # https://docs.djangoproject.com/en/4.0/ref/settings/#databases
78
79    DATABASES = {
80        'default': {
81            'ENGINE': 'django.db.backends.sqlite3',
82            'NAME': BASE_DIR / 'db.sqlite3',
83        }
84    }
```

Словарь DATABASES хранит настройки для корректного соединения с базой данных. По умолчанию, Django предлагает использовать SQLite для разработки, в нашем же проекте мы будем использовать СУБД MySQL, поэтому сейчас нам придётся немного отредактировать данный словарь.

Для этого сначала потребуется установить клиент данной СУБД и библиотеку Python **mysqlclient**, нужную для связи с Python-кодом.

В Ubuntu для установки в терминале необходимо ввести следующую команду:

```
codeby@django:~$ sudo apt-get install mysql-server mysql-client
```

В процессе установки терминал попросит Вас ввести пароль для будущего доступа к БД. После установки зайдём под root-пользователем и введём ранее придуманный пароль.

```
codeby@django:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.29 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Для того чтобы подключить MySQL в Django нужно указать БД, которую мы будем использовать в проекте. Поэтому создадим её:

```
mysql> create database lms_project;
Query OK, 1 row affected (0.12 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| lms_project        |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql>
```

Также перед тем, как установить библиотеку mysqlclient, нужно скачать ещё один пакет командой:

```
codeby@django:~$ sudo apt install libmysqlclient-dev
```

Этот пакет содержит заголовочные файлы для разработки баз данных MySQL.

Теперь наконец установим библиотеку командой: pip install mysqlclient

Далее, нам стоит внести все настройки БД в параметр DATABASES файла настроек Django

```
77  # Database
78  # https://docs.djangoproject.com/en/4.0/ref/settings/#databases
79
80  DATABASES = {
81      'default': {
82          'ENGINE': 'django.db.backends.mysql',
83          'HOST': 'localhost',
84          'USER': os.environ.get('USER_DB'),
85          'PASSWORD': os.environ.get('PASSWORD_DB'),
86          'NAME': 'lms_project',
87      }
88  }
```

Рекомендую не указывать явно пароли, а записывать их в переменную окружения, и уже при помощи модуля os извлекать значение.

Переменную окружения вы можете задать при помощи команды в терминале:

Для Ubuntu: export <VAR_NAME>=<VAR_VALUE>

Для Windows: set <VAR_NAME>=<VAR_VALUE>

Следующая настройка — AUTH_PASSWORD_VALIDATORS

Служит для валидации паролей при регистрации / авторизации пользователей на сайте. Например, Django не разрешает создавать слишком маленькие пароли (< 4 символов), запрещает использовать только цифры и спец. символы и т. д.

```
# Password validation
# https://docs.djangoproject.com/en/4.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

Всё это делается для того, чтобы обезопасить самого пользователя от взлома его аккаунта.

Следующие настройки необходимы для интернационализации и локализации нашего сайта.

```
# Internationalization+
# https://docs.djangoproject.com/en/4.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True
```

LANGUAGE_CODE указывает язык нашего сайта. На нём будут выводиться сообщения и страницы админки.

TIME_ZONE устанавливает временную зону в виде строки. В данном случае, укаазана зона по Гринвичу.

USE_I18N — булева переменная указывает на разрешение переводить сайт на язык, указанный в LANGUAGE_CODE.

USE_TZ — позволяет хранить значение даты и времени с отметкой временной зоны, если True, в противном случае False.

```
118    # Static files (CSS, JavaScript, Images)
119    # https://docs.djangoproject.com/en/4.0/howto/static-files/
120
121    STATIC_URL = 'static/'
```

STATIC_URL указывает на путь для хранения статических файлов.

И наконец последний параметр настроек Django-проекта — DEFAULT_AUTO_FIELD

Данный параметр абсолютно идентичен по смыслу атрибуту класса настроек приложения, указанного ранее.

Как видим, настроек не так мало, поэтому на разработчика сайта ложится особая ответственность за корректную конфигурацию и в дальнейшем работу сайта.

Запуск проекта

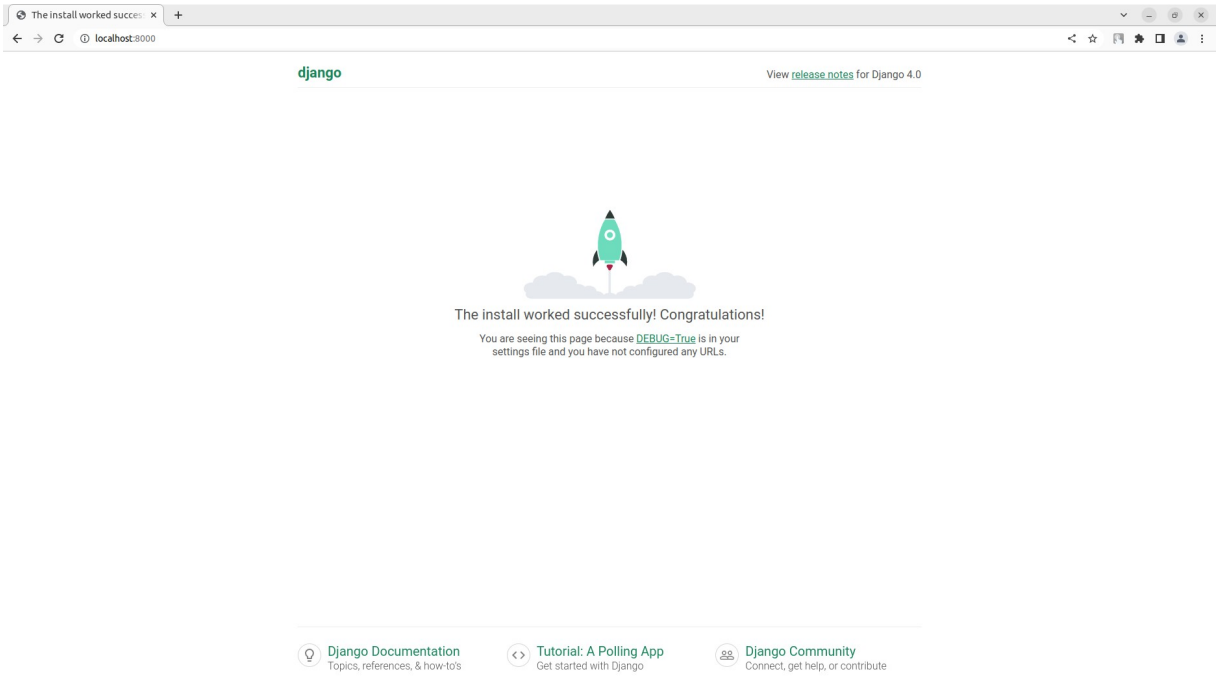
Когда всё правильно сконфигурировано, мы можем совершить первый запуск нашего учебного проекта.

```
^C(venv) codeby@django:~/Desktop/lms_project/lms_project$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the mig
, sessions.
Run 'python manage.py migrate' to apply them.
July 10, 2022 - 15:19:03
Django version 4.0, using settings 'lms_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Перейдём по ссылке на сайт.



Если появилась именно эта страница, то запуск прошёл успешно и все настройки правильно подобраны.

Мы видим, Django нас предупреждает, что нужно сделать миграции в базу данных, создав таблицы для управления сессиями и т. д. Этим мы и займёмся на следующем уроке.

Домашнее задание

✓ Создать структуру вашего проекта: сам проект и приложения

✓ Установить сервер и клиент СУБД MySQL, пакет libmysqlclient-dev и python-библиотеку mysqlclient

✓ Внести информацию об используемой СУБД в файл настроек проекта Django

✓ Запустить проект