



Лекция 6.

Развертывание проекта



На этой лекции мы

1. Узнаем о профилировании Django
2. Разберемся в развертывании проекта на сервере

Краткая выжимка, о чём говорилось в предыдущей лекции

На прошлой лекции мы:

1. Узнали об административной панели Django
2. Разобрались в настройке админ панели
3. Изучили добавление пользовательских моделей в панель
4. Узнали о способах персонализации админ панели

План лекции

[На этой лекции мы](#)

[Краткая выжимка, о чём говорилось в предыдущей лекции](#)

[План лекции](#)

[Подробный текст лекции](#)

[Профилирование и оптимизация в Django](#)

[Вместо старта](#)

[Описание и возможности Django Debug Toolbar](#)

[Установка и настройка](#)

[Установка](#)

[Проверка настроек Django](#)

[Дополнительные настройки проекта](#)

[Проверка работоспособности Debug Toolbar](#)

[Оптимизация проекта](#)

[Подготовка данных](#)

[Создаём представления](#)

[Представление для суммирования в БД](#)
[Представление для суммирования в представлении](#)
[Представление для суммирования в модели из шаблона](#)
[Создаём свойство модели](#)
[Шаблон](#)
[Настраиваем маршруты](#)
[Сравнение результатов](#)
[Развертывание проекта на сервере](#)
[Регистрация на платформе](#)
[Подготовка проекта к развертыванию](#)
[Настройки безопасности](#)
[Настройки доступа](#)
[Настройка подключения к базе данных](#)
[Формируем список пакетов](#)
[Репозиторий GitHub](#)
[Список команд и действий для тех, кто не использовал GitHub](#)
[Настройка проекта на сервере](#)
[Создаём веб-приложение](#)
[Настройки веб-приложения](#)
[Сохраним “секреты” в окружении](#)
[База данных](#)
[Раздача статики сервером](#)
[Суперпользователь](#)
[Вывод](#)
[Домашнее задание](#)

Подробный текст лекции

Профилирование и оптимизация в Django

В комплекте Django нет встроенного инструмента для профилирования кода. Но фреймворк позволяет подключать пакеты, созданные другими разработчиками. И в качестве пакета для профилирования обычно используют Django Debug Toolbar. Проект был создан Робом Хадсоном в августе 2008 года и активно развивается командой Jazzband <https://jazzband.co>

Кроме того Django Debug Toolbar рекомендуют сами разработчики фреймворка Django как топ-3 подключаемых дополнений.

Вместо старта

Если вы создавали новое приложение для каждого занятия, выполните команды:

```
>cd myproject
>python manage.py startapp myapp6
```

Отлично! Новое приложение создано в проекте. Сразу подключим его в настройках `setting.py`

```
INSTALLED_APPS = [
    'django.contrib.admin',
    ...
    'myapp6',
]
```

Всё готово к началу изучения админки на практике.

Описание и возможности Django Debug Toolbar

Django Debug Toolbar — это инструмент для отладки Django-приложений, который предоставляет дополнительную информацию о запросах и ответах, используя различные панели, которые можно настроить. Он может быть полезен для оптимизации производительности, отслеживания ошибок и улучшения качества кода.

Основные возможности Django Debug Toolbar:

- Панель запросов: отображает время выполнения каждого запроса к базе данных, а также общее количество запросов.
- Панель шаблонов: показывает, какие шаблоны были использованы для генерации страницы и сколько времени заняло их выполнение.
- Панель SQL: отображает все SQL-запросы, выполненные приложением, включая параметры запросов.
- Панель кэша: показывает, какие запросы были сохранены в кэше и сколько времени они там находились.

- Панель профилирования: предоставляет информацию о времени выполнения каждой функции приложения.
- Панель HTTP-заголовков: отображает HTTP-заголовки запросов и ответов.
- Панель среды выполнения: показывает информацию о системе, на которой работает приложение.
- Панель логирования: позволяет просматривать сообщения журнала приложения.

Кроме того Django Debug Toolbar также позволяет создавать свои собственные панели, чтобы отслеживать дополнительную информацию, которая может быть полезна для вашего приложения.

Установка и настройка

Как и большинство проектов с открытым исходным кодом, пакет отладки находится в открытом репозитории GitHub <https://github.com/jazzband/django-debug-toolbar>

Установка

Для простой и привычной Python разработчику установки проще всего воспользоваться компонентом PIP для установки. Выполним команду:

```
pip install django-debug-toolbar
```



Внимание! Убедитесь, что перед выполнением установки вы активировали виртуальное окружение вашего Django проекта.

Для краткости далее в лекции будем сокращать название пакета Django Debug Toolbar до DjDT.

Проверка настроек Django

У вас уже должен быть создан проект Django. Т.е. вы выполнили команду

```
django-admin startproject <project_name>
```

раньше, чем установили пакет отладки. Если так, то в файле settings.py проекта уже заполнены четыре параметра, влияющие на DjDT:

```

...
INSTALLED_APPS = [
    ...
    'django.contrib.staticfiles',
    ...
]
...
TEMPLATES = [
    {
        'BACKEND':
        'django.template.backends.django.DjangoTemplates',
        ...
        'APP_DIRS': True,
        ...
    },
]
...
STATIC_URL = 'static/'
...

```

Если вы вносили правки в конфигурацию проекта, стоит проверить следующие пункты:

- django.contrib.staticfiles включён в список INSTALLED_APPS
- в качестве значения для ключа BACKEND в списке TEMPLATES установлен django.template.backends.django.DjangoTemplates
- в качестве значения для ключа APP_DIRS в списке TEMPLATES установлена истина - True
- задано значение константы STATIC_URL. Обычно это каталог static/

Если ваши настройки отличаются от перечисленных выше, внесите необходимые правки в код settings.py.

Дополнительные настройки проекта

Продолжаем работать с файлом settings.py.

```

...
INTERNAL_IPS = [
    '127.0.0.1',
]
...
INSTALLED_APPS = [
    ...

```

```

    'debug_toolbar',
    ...
]

MIDDLEWARE = [
    'debug_toolbar.middleware.DebugToolbarMiddleware',
    ...
]
...
```

Добавим элементы в три списка:

- в список `INTERNAL_IPS` добавим локальный адрес компьютера - `127.0.0.1`. Панель отображается только в случае совпадения адреса.
- в список `INSTALLED_APPS` добавим DjDT как приложение `debug_toolbar`
- в список `MIDDLEWARE` добавим промежуточный слой `debug_toolbar.middleware.DebugToolbarMiddleware`



Внимание! Константа `INTERNAL_IPS` по-умолчанию отсутствует в `settings.py`. Мы впервые создаём её в рамках курса.



Важно! Важен порядок подключения компонентов промежуточного слоя в `MIDDLEWARE`. Рекомендуется подключить `DebugToolbarMiddleware` как можно раньше.

Осталась финальная настройка в файле `urls.py` проекта. Нам нужно добавить маршрут для работы профилировщика.

```

...
from django.urls import path, include
...

urlpatterns = [
    ...
    path('__debug__/', include("debug_toolbar.urls")),
]
```

Сам путь `debug` не даст полезной информации. Но пока параметр `DEBUG` установлен в истину, справа будет появляться панель с отладочной информацией.

Проверка работоспособности Debug Toolbar

Для проверки работоспособности запустим сервер и перейдём по любому из прописанных в `urls` адресу.

```
python manage.py runserver
```

Если в правом верхнем углу страницы появился прямоугольник DjDT, мы верно сделали настройки. Нажав на него можно посмотреть подробную отладочную информацию, разбитую по категориям.



Оптимизация проекта

Рассмотрим как DjDT помогает оценивать код и оптимизировать его. Мы воспользуемся таблицей `Продукты`, которую создали на прошлой лекции. И выведем общее количество всех продуктов, сумму поля `quantity` для всех записей. Попробуем посчитать сумму разными способами:

- в базе данных
- в представлении
- в модели через вызов в шаблоне

Ранее в курсе мы копировали модели автора и статьи из одного приложения в другое. Сегодня подключимся к моделям `myapp5` из приложения `myapp6`.

Подготовка данных

Создадим команду, которая заполнит таблицу `Продукты` большим количеством фейковых данных. Для этого в каталоге приложения создадим пакет `management`, а в нём пакет `commands`. Внутри поместим команду `make_products`. Таким образом путь до файла, который мы заполняем будет `.../myapp6/management/commands/make_products.py`

```
from random import choice, randint, uniform

from django.core.management.base import BaseCommand
```



```

from myapp5.models import Category, Product

class Command(BaseCommand):
    help = "Generate fake products."

    def add_arguments(self, parser):
        parser.add_argument('count', type=int, help='User ID')

    def handle(self, *args, **kwargs):
        categories = Category.objects.all()
        products = []
        count = kwargs.get('count')
        for i in range(1, count + 1):
            products.append(Product(
                name=f'продукт номер {i}',
                category=choice(categories),
                description='длинное описание продукта, которое и
так никто не читает',
                price=uniform(0.01, 999_999.99),
                quantity=randint(1, 10_000),
                rating=uniform(0.01, 9.99),
            ))
        Product.objects.bulk_create(products)

```

Что же делает команда?

- Импортируются модули choice, randint, uniform из библиотеки random и две модели Category и Product из приложения myapp5.
- Класс Command наследуется от базового класса BaseCommand, который предоставляет функциональность для создания пользовательских команд в Django.
- В методе add_arguments() определяется аргумент командной строки 'count', который будет содержать количество продуктов, которые нужно сгенерировать.
- Метод handle() вызывается при выполнении команды. Он получает все категории из БД, создает список продуктов и заполняет его фейковыми данными (названия, категории, описания, цены, количества и рейтинги). Затем, используя метод bulk_create(), продукты сохраняются в БД.



Внимание! Вместо метода `save` для сохранения каждого продукта по отдельности, используем метод `bulk_create`. Он получает список продуктов и сохраняет его в базу данных одной операцией.

Выполним команду `python manage.py make_products 10000`

Создаём представления

Создадим три представления, решающие задачу суммирования количества разными способами.

Представление для суммирования в БД

В первом случае возложим задачу по подсчёту общего количества продуктов на базу данных:

```
from django.shortcuts import render
from django.db.models import Sum

from myapp5.models import Product

def total_in_db(request):
    total = Product.objects.aggregate(Sum('quantity'))
    context = {
        'title': 'Общее количество посчитано в базе данных',
        'total': total,
    }
    return render(request, 'myapp6/total_count.html', context)
```

Метод `aggregate(Sum('quantity'))` отправит в базу агрегирующий запрос с суммированием всех значений столбца “количество”. Результат пробрасывается в шаблон `total_count.html` как параметр `total`.

Представление для суммирования в представлении

Во втором случае возложим задачу по подсчёту общего количества продуктов на само представление:

```
from django.shortcuts import render
from django.db.models import Sum
```

```

from myapp5.models import Product

...

def total_in_view(request):
    products = Product.objects.all()
    total = sum(product.quantity for product in products)
    context = {
        'title': 'Общее количество посчитано в представлении',
        'total': total,
    }
    return render(request, 'myapp6/total_count.html', context)

```

Метод `all` возвращает все продукты из базы данных. Далее в цикле перебираем продукты и функция `sum` подсчитывает результат по `product.quantity`. Передача данных в шаблон проходит аналогично варианту 1.

Представление для суммирования в модели из шаблона

В третьем случае возложим задачу по подсчёту общего количества продуктов на модель `Product`, а представление пробросит её в шаблон

```

from django.shortcuts import render
from django.db.models import Sum

from myapp5.models import Product

...

def total_in_template(request):
    context = {
        'title': 'Общее количество посчитано в шаблоне',
        'products': Product,
    }
    return render(request, 'myapp6/total_count.html', context)

```

Представление ничего не вычисляет. Мы передаём модель `Product` в шаблон `total_count.html`. Внутри шаблона вызовем метод модели, подсчитывающий общее количество продуктов.

Создаём свойство модели

Для третьего представления необходимо внести дополнения в код модели.



Внимание! Мы используем модель из прошлой лекции. Если для каждого занятия вы создавали новое приложение, правки необходимо внести в `myapp5/models.py`

Создадим функцию, которая подсчитывает сумму и сделаем её свойством.

```
from django.db import models

class Category(models.Model):
    name = models.CharField(max_length=50, unique=True)

    def __str__(self):
        return self.name

class Product(models.Model):
    name = models.CharField(max_length=50)
    category = models.ForeignKey(Category,
on_delete=models.CASCADE)
    description = models.TextField(default='', blank=True)
    price = models.DecimalField(default=999999.99, max_digits=8,
decimal_places=2)
    quantity = models.PositiveSmallIntegerField(default=0)
    date_added = models.DateTimeField(auto_now_add=True)
    rating = models.DecimalField(default=5.0, max_digits=3,
decimal_places=2)

    def __str__(self):
        return self.name

    @property
    def total_quantity(self):
        return sum(product.quantity for product in
Product.objects.all())
```

Декоратор `@property` позволяет обращаться к методам класса как к свойствам. `total_quantity` собирает все записи о продуктах - метод `all`. Далее в цикле продукты перебираются и функция `sum` вычисляет результат по столбцу количество.

Шаблон

Создадим код шаблона `total_count.html` в каталоге `myapp6/templates/myapp6/`

```
{% extends 'base.html' %}

{% block title %}{{ title }}{% endblock %}

{% block content %}
    <h3>{{ title }}</h3>
    {% if total %}
        <p>{{ total }}</p>
    {% endif %}
    {% if products %}
        <p>{{ products.total_quantity }}</p>
    {% endif %}
{% endblock %}
```

Каталог расширяет базовый шаблон проекта `base.html`. Переменная `title` используется и в качестве заголовка страницы и в качестве заголовка третьего уровня в теле страницы.

Если передана переменная `total`, выводим её содержимое. Тут мы увидим вывод для представления один - из базы данных и для представления два - из самого представления.

Если передана переменная `products`, значит шаблон получил доступ к модели `Product` и может обратиться к свойству `total_quantity` модели.

Мы создали универсальный шаблон для трёх вариантов вывода суммы

Настраиваем маршруты

Чтобы код заработал, настроим маршрут в `urls.py` проекта

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

```
urlpatterns = [
    path('admin/', admin.site.urls),
    ...
    path('debug/', include("debug_toolbar.urls")),
    path('les6/', include('myapp6.urls')),
]
```

Теперь создадим файл `myapp6/urls.py`. Мы подключили его к маршруту с префиксом `les6/`.

```
from django.urls import path
from .views import total_in_db, total_in_view, total_in_template

urlpatterns = [
    path('db/', total_in_db, name='db'),
    path('view/', total_in_view, name='view'),
    path('template/', total_in_template, name='template'),
]
```

Каждое из трёх представлений импортировано и подключено со своим маршрутом. Можно запускать сервер и смотреть на работу DjDT.

Сравнение результатов

Мы можем сравнить время выполнения каждого запроса к базе данных, а также общее количество запросов, посмотреть какие шаблоны были использованы для генерации страницы и сколько времени заняло их выполнение и много другое. Чем сложнее будет ваш проект, тем больше полезной информации о слабых местах вы будете получать. А знание - силы. У вас появится возможность внести правки, улучшить проект.

Развертывание проекта на сервере

В финале курса перенесём написанный на лекциях код на сервер. Аналогичным образом вы сможете разворачивать в облаке ваши учебные и реальные проекты в будущем.

Регистрация на платформе

В качестве площадки мы выбрали <https://www.pythonanywhere.com/>. Она позволяет бесплатно иметь один веб проект, а значит любой из вас может пользоваться сервисом. Для регистрации нажмите кнопку **Start running Python online in less than a minute!** На открывшейся страницы выберите **Create a Beginner account** и вы окажетесь в форме регистрации.



Внимание! Бесплатный тариф создаст для вашего Django проекта адрес в интернете вида `username.pythonanywhere.com`. Введи в качестве username при регистрации то имя, которое хотите видеть в адресной строке браузера.

После регистрации вы увидите цепочку обучающих окон. Отлично! С платформой пока всё, внесём правки в код до его переноса.



Важно! Не забудьте подтвердить электронную почту, которую вы ввели при регистрации. Письмо придёт на почту в течение нескольких минут.

Подготовка проекта к развертыванию

Подготовим код, который мы писали в рамках лекций к развёртыванию на сервере.

Настройки безопасности

Начнём с файла `settings.py`.

В первую очередь выключаем режим отладки

```
DEBUG = False
```

Тут же добавим две константы. Так мы повышаем безопасность работы с сессиями и с csrf токенами

```
SESSION_COOKIE_SECURE = True  
CSRF_COOKIE_SECURE = True
```

И секретный ключ. Его стоит хранить не в файле настроек, а в переменных окружения. Поэтому заменяем строку с ключом от Django на следующие пару строк:

```
import os
...
SECRET_KEY = os.getenv('SECRET_KEY')
```

Сгенерируем и добавим секретный ключ в переменные окружения чуть позже.

Настройки доступа

Также добавим адрес сайта в список доступных хостов:

```
ALLOWED_HOSTS = [
    '127.0.0.1',
    ...
    'username.pythonanywhere.com',
]
```



Внимание! Здесь и далее вместо username подставляйте имя, которое вы использовали для регистрации на Python Anywhere.

И ещё одна константа для правильной настройки работы со статическими файлами на сервере:

```
...
STATIC_ROOT = BASE_DIR / 'static/'
...
```

Настройка подключения к базе данных

Откроем страницу Базы данных на сайте pythonanywhere. Для бесплатного использования нам доступна БД MySQL. Придумаем пароль доступа к базе данных. Его стоит запомнить, чтобы чуть позже подключить Django к БД. Ожидаем завершение инициализации базы данных. По умолчанию будет создана база username\$default. Кликаем по имени, чтобы открыть консоль MySQL. Вводим команду смены кодировки с латиницы по умолчанию, на UTF-8:

```
ALTER DATABASE username$default CHARACTER SET utf8 COLLATE
utf8_general_ci;
```

После выполнения команды выключаем консоль командой exit.

Вернёмся в файл `seetings.py` и настроим подключение к MySQL

```
...
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': '<your_username>$<your_database_name>',
        'USER': '<your_username>',
        'PASSWORD': os.getenv('MYSQL_PASSWORD'),
        'HOST': '<your_mysql_hostname>',
        'OPTIONS': {
            'init_command': "SET NAMES 'utf8mb4';SET
sql_mode='STRICT_TRANS_TABLES'",
            'charset': 'utf8mb4',
        },
    }
}
...
```

Данные об именах пользователя, базы данных и хосте доступны на странице доступа к базе данных. Пароль к БД вы придумали несколько минут назад. Мы также добавим его в переменные ОС чуть позже. Внимательно скопируйте данные из сайта в файл настроек.

Значения ключа `OPTIONS` переносим из методички в настройки без изменения. Они понадобятся для правильной обработки кириллицы и сохранности данных.

Формируем список пакетов

В терминале где мы обычно запускали сервер Django введём команду

```
pip freeze > requirements.txt
```

В результате в каталоге проекта появится файл `requirements.txt` примерно следующего содержания

```
asgiref==3.7.2
Django==4.2.1
django-debug-toolbar==4.1.0
Pillow==9.5.0
sqlparse==0.4.4
tzdata==2023.3
```



Внимание! У вас могут отличаться версии пакетов. Это нормально, так как методические материалы готовились в одно время, а вы работаете с кодом в другое. Используйте свои версии пакетов, а не из примера выше.

Запишем в requirements.txt ещё пару строк в конец файла

```
mysqlclient
python-dotenv
```

Эти пакеты понадобятся чтобы подружить Django с базой данных и получить доступ к секретным паролям из переменных окружения.

Так же вы можете удалить из списка django-debug-toolbar. При этом проверьте, что ваша версия для развёртывания не задействует DjDT. Пройдите этапы настройки в обратном порядке, отключите лишнее. Как минимум отключите маршрут в urls.py проекта:

```
urlpatterns = [
    ...
    # path('debug/', include("debug_toolbar.urls")),
    ...
]
```

Для переноса кода на сервер всё готово.

Репозиторий GitHub

Скорее всего вы уже создали репозиторий на github с вашим проектом. Добавьте в него финальные изменения проекта, чтобы клонировать в pythonanywhere командой

```
git clone https://github.com/myusername/myproject.git
```

Для её ввода необходимо перейти на вкладку Dashboard и открыть Bash консоль

Список команд и действий для тех, кто не использовал GitHub

На вашем ПК открываем консоль в директории проекта myproject и инициализируем Git

```
git init
```

Создаём в директории проекта файл `.gitignore` и добавляем в него “лишние” файлы, например базы данных, логи, кеш Python:

```
/media/  
/static/  
*.sqlite3  
*.log  
*.pyc
```

Добавляем остальные файл в Git:

```
git add *
```

Фиксируем изменения коммитом:

```
git commit -m "Initial commit"
```

Создаём репозиторий на сайте <https://github.com/>

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *

Repository name *

✔ myproject is available.

Great repository names are short and memorable. Need inspiration? How about [ideal-spoon](#) ?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: **None** ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **None** ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

You are creating a public repository in your personal account.

Create repository

Соединяем локальный и удалённый репозитории

```
git remote add origin https://github.com/username/myproject.git
```

Пересылаем код в удалённый репозиторий

```
git push -u origin master
```

Переходим на сайт [pythonanywhere](#), открываем консоль и клонируем репозиторий

```
git clone https://github.com/myusername/myproject.git
```

Дожидаемся завершения клонирования. В разделе Dashboard нажав на кнопку Browse files вы увидите клонированный каталог проекта.

Настройка проекта на сервере

После завершения клонирования остаёмся в консоли, запускаем команду на создание виртуального окружения:

```
mkvirtualenv --python=/usr/bin/python3.10 virtualenv
```

Активация виртуального окружения происходит автоматически после создания. Не закрывая консоль устанавливаем необходимые пакеты:

```
cd myproject  
pip install -r requirements.txt
```



Внимание! Процесс создания файлов достаточно медленный и может занимать несколько минут. Иногда 5, а иногда 20 и более. Наберитесь терпения, выпейте свой любимый напиток.

Создаём веб-приложение

Дождавшись установки пакетов нажимаем на вкладке Dashboard пункт Web apps и создаём новое кнопкой Add a new web app

1. Подтверждаем доменное имя для бесплатного профиля кнопкой Next
2. Выбираем пункт Manual configuration (including virtualenvs)
3. Выбираем последнюю из доступных версий Python
4. Подтверждаем выбор очередным нажатием Next
5. All done! Your web app is now set up. Details below.

После завершения работы сервиса мы окажемся на вкладке настроек веб приложения.

Настройки веб-приложения

В первую очередь находим раздел Virtualenv и указываем путь до созданного нами окружения:

```
/home/username/.virtualenvs/virtualenv
```

Теперь отредактируем wsgi файл, ссылка на который находится в разделе Code.



Внимание! Не путайте wsgi.py сервера с wsgi.py в каталоге вашего Django проекта.



Важно! Команда `manage.py runserver localhost:8000` не будет работать на PythonAnywhere, потому что наши консольные серверы недоступны из внешнего мира. Вместо этого надо отредактировать файл `wsgi` и раскомментировать раздел для Django. После этого ваш сайт Django будет работать по адресу `username.pythonanywhere.com`.

В файле находим раздел Django (примерно 74-90 строки) и удаляем всё лишнее до и после него. У вас должно получиться примерно следующее:

```
# ++++++ DJANGO ++++++
# To use your own django app use code like this:
import os
import sys

from dotenv import load_dotenv

project_folder = os.path.expanduser('~/.myproject') # adjust as
appropriate
load_dotenv(os.path.join(project_folder, '.env'))

## assuming your django settings file is at
'/.home/username/mysite/mysite/settings.py'
## and your manage.py is at '/home/username/mysite/manage.py'
path = '/home/username/myproject'
if path not in sys.path:
    sys.path.append(path)
```

```
os.environ['DJANGO_SETTINGS_MODULE'] = 'myproject.settings'

## then:
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

Мы подключили dotenv для доступа к “секретам” из проекта. Далее настроили путь до каталога с проектом и дали команду на запуск wsgi сервера.

Сохраним “секреты” в окружении

Для начала создадим секретный ключ. Для этого в консоли запускаем интерпретатор Python и воспользуемся функцией token_hex из модуля secrets

```
>>> python
Python 3.10.5 (main, Jul 22 2022, 17:09:35) [GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import secrets
>>> secrets.token_hex()
'53397a811810419188d117156a1a1f324bd65cb75fc925f852d8e66f058833cc'
>>> exit()
```

Копируем токен и выходим из режима интерпретатора командой exit()


Проверяем, что мы находимся в той же директории, что указана в wsgi файле в для переменной project_folder. Если нет, переходим в нужный каталог и выполняем команды добавления “секретов”:

```
(virtualenv) 12:01~/myproject (master)$ echo "export
SECRET_KEY=53397a811810419188d117156a1a1f324bd65cb75fc925f852d8e6
6f058833cc" >> .env
(virtualenv) 12:01 ~/myproject (master)$ echo "export
MYSQL_PASSWORD=dbpassword" >> .env
```

Если всё выполнено верно, обновляем приложение и переходим по ссылке для проверки его работы.



Внимание! Вам нужно будет нажать кнопку «Обновить» , чтобы перезагрузить веб-приложение всякий раз, когда вы хотите увидеть влияние изменений кода на свой сайт.

 **Важно!** Если приложение не запускается, проверьте раздел Log files на наличие ошибок. Например в первой лекции мы прописывали логирование в файл. Если вы указывали сохранение логов в каталоги, их надо создать самостоятельно. В версии из лекций это каталог log в корневой директории.

База данных

Следующий этап - создать таблицы в базе данных MySQL на основе миграций.

Для начала научим консоль работать с “секретами”. Введём команду

```
echo 'set -a; source ~/myproject/.env; set +a' >>  
~/virtualenvs/virtualenv/bin/postactivate
```

Таким образом при старте консоли мы будем получать доступ к переменным окружения. Выходим из консоли командой exit и открываем снова на закладке Web в разделе Virtualenv. Кликаем по ссылке Start a console in this virtualenv

Далее применяем миграции к базе данных:

```
python manage.py migrate
```

Если всё выполнено верно, увидим как Django применяет цепочку миграций.

Раздача статики сервером

Нам снова понадобится консоль. Если вы не закрывали её после создания миграций, продолжаем работать в ней. Соберём статические файлы проекта и приложений в одном месте. Для этого выполним команду:

```
python manage.py collectstatic
```

Выполнение команды может занять несколько минут. Зависит от размера проекта, количество папок и файлов со статикой в приложениях. После завершения сбора переходим на вкладку Web в раздел Static files.

В поле URL запишем /static/. Это содержимое нашей константы STATIC_URL в настройках проекта. В поле Directory введём абсолютный путь до каталога со статикой. Его нам сообщил Django как результат работы команды collectstatic. Скорее всего это путь /home/username/myproject/static

Перезагружаем сервер, статика должна начать автоматически раздаваться.

Суперпользователь

Наш проект настроен и успешно работает. Остаётся создать суперпользователя, чтобы получить доступ к административной панели.

Открываем консоль и выполняем команду

```
python manage.py createsuperuser
Имя пользователя (leave blank to use 'username'): superadmin
Адрес электронной почты: super@admin.ru
Password:
Password (again):
Superuser created successfully.
```



Внимание! Пароль не отображается при вводе, но учитывает нажатие клавиш

Финальный раз перезагружаем сервер. Переходим в админ панель, проверяем логин и пароль, радуемся успешному развёртыванию проекта в облаке.

Вывод

На этой лекции мы:

1. Узнали о профилировании Django
2. Разобрались в развёртывании проекта на сервер

Домашнее задание

1. Для закрепления материалов лекции попробуйте самостоятельно набрать и запустить демонстрируемые примеры.