Python/Django Уровень 2

Урок 5

Развертывание Django-проекта на сервере

Готовим проект к развертыванию. Переходим на базу данных «PostgreSQL». Имитируем работу с VPS - устанавливаем и настраиваем сервер Ubuntu Server 17. Реализуем связку «nginx»+«gunicorn».

Работа с заказом: исправление ошибок в программе

- Убираем неактивные продукты с формы
- Асинхронно загружаем цену продукта
- Обнуляем количество товара в массиве при удалении формы из набора

Подготовка проекта к развертыванию

- Создание файла «requirements.txt»
- Переход на другую подсистему хранения
- Выключаем режим отладки

Варианты развертывания

Установка сервера

Настройка сервера

- Настраиваем сеть и доступ по SSH
- Обеспечиваем обмен файлами по FTP
- Настраиваем окружение для проекта
- Сервер БД «PostgreSQL»
- Миграции и импорт данных

Развертывание Django-проекта при помощи «gunicorn»

- *Управление «gunicorn» при помощи «supervisor»

Практическое задание

Дополнительные материалы

Используемая литература

Работа с заказом: исправление ошибок в программе

Перед развертыванием проекта на сервере исправим некоторые существенные недочеты - «баги».

Убираем неактивные продукты с формы

Django автоматически заполняет выпадающие списки набора форм всеми экземплярами класса Product. Мы можем вмешаться в этот процесс через атрибут поля формы «queryset»:

geekshop/ordersapp/forms.py

```
...
class OrderItemForm(forms.ModelForm):
    ...

    def __init__(self, *args, **kwargs):
        ...
        self.fields['product'].queryset = Product.get_items()
```

Необходимо в классе «Product» приложения «mainapp» прописать статический метод «.get_items()»:

geekshop/mainapp/models.py

```
...
class Product(models.Model):
    ...

    @staticmethod
    def get_items():
        return Product.objects.filter(is_active=True).\
                               order_by('category', 'name')
```

Асинхронно загружаем цену продукта

Допишем обработчик выбора продукта на форме из прошлого урока:

geekshop/static/js/orders_scripts.js

```
...
$('.order_form select').change(function () {
   var target = event.target;
   orderitem_num = parseInt(target.name.replace('orderitems-', '').\
                                        replace('-product', ''));
   var orderitem_product_pk = target.options[target.selectedIndex].value;

   if (orderitem_product_pk) {
       $.ajax({
           url: "/order/product/" + orderitem_product_pk + "/price/",
           success: function (data) {
               if (data.price) {
                   price_arr[orderitem_num] = parseFloat(data.price);
                   if (isNaN(quantity_arr[orderitem_num])) {
                       quantity_arr[orderitem_num] = 0;
                   }
                   var price_html = '<span>' + \
                                    data.price.toString().replace('.', ',') +\
                                    '</span> руб';
                   var current_tr = $('.order_form table').\
                                   find('tr:eq(' + (orderitem_num + 1) + ')');

                   current_tr.find('td:eq(2)').html(price_html);

                   if (isNaN(current_tr.find('input[type="number"]').val())) {
                       current_tr.find('input[type="number"]').val(0);
                   }
                   orderSummaryRecalc();
               }
           },
       });
   }
});
...
```

Django, при формировании выпадающего списка с продуктами, автоматически заполняет атрибут «val» каждого элемента значением «pk» из базы данных. Сохраняем это значение в переменную «orderitem_product_pk» и делаем запрос цены контроллеру через «.ajax()».

В диспетчер URL приложения «ordersapp» необходимо добавить строку:

```
re_path(r'^product/(?P<pk>\d+)/price/$', ordersapp.get_product_price)
```

И написать сам контроллер:

geekshop/ordersapp/views.py

```
...
from django.http import JsonResponse
from mainapp.models import Product
...
def get_product_price(request, pk):
    if request.is_ajax():
        product = Product.objects.filter(pk=int(pk)).first()
        if product:
            return JsonResponse({'price': product.price})
        else:
            return JsonResponse({'price': 0})
```

В массив «price_arr» заносим полученное в ответе значение цены и выводим на странице:

```
current_tr.find('td:eq(2)').html(price_html)
```

Сначала получаем и сохраняем в переменную «current_tr» актуальную строку таблицы при помощи jQuery метода «.find()». Затем обновляем содержимое ячейки с индексом «2» данными о цене продукта. Тут же исправляем еще один баг: когда при добавлении новых форм к набору поле количества создается пустым - задаем нулевое значение:

```
current_tr.find('input[type="number"]').val(0)
```

В функцию «orderSummaryRecalc()» обернули код расчета стоимости заказа:

geekshop/static/js/orders_scripts.js

```
...
if (!order_total_quantity) {
    orderSummaryRecalc();
}

function orderSummaryRecalc() {
    order_total_quantity = 0;
    order_total_cost = 0;

    for (var i=0; i < TOTAL_FORMS; i++) {
        order_total_quantity += quantity_arr[i];
        order_total_cost += quantity_arr[i] * price_arr[i];
    }
    $('.order_total_quantity').html(order_total_quantity.toString());
    $('.order_total_cost').html(Number(order_total_cost.toFixed(2)).\
                                                        toString());
}
...
```

Обнуляем количество товара в массиве при удалении формы из набора

Если мы удалим товар из заказа, а потом добавим на его место другой, то обнаружим, что общее число продуктов в заказе будет считаться неправильно. Необходимо обнулить значение в массиве:

geekshop/static/js/orders_scripts.js

```
...
function deleteOrderItem(row) {
    ...
    delta_quantity = -quantity_arr[orderitem_num];
    quantity_arr[orderitem_num] = 0;
    if (!isNaN(price_arr[orderitem_num]) && !isNaN(delta_quantity)) {
        orderSummaryUpdate(price_arr[orderitem_num], delta_quantity);
    }
}
...
```

Теперь при *любых* манипуляциях на странице создания и редактирования заказа его статистика корректно обновляется.

Подготовка проекта к развертыванию

Создание файла «requirements.txt»

Для переноса Django-проекта нам необходим файл зависимостей «requirements.txt», содержащий сведения об используемых пакетах и их версиях. Для генерации этого файла можно воспользоваться инструментом «pip-tools»:

```
pip install pip-tools
```

После установки в папке с проектом создаем файл со списком установленных в проекте пакетов «requirements.in»:

geekshop/requirements.in

```
# requirements.in
psycopg2
pillow
django
django-debug-toolbar
social_auth_app_django
```

Создаем файл «requirements.txt», выполнив в папке с проектом код:

```
pip-compile
```

Если вы использовали виртуальное окружение, можно получить файл requirements.txt более простым способом:

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

При работе в виртуальном окружении в «*nix» системах можно писать «pip» и «Python» вместо «pip3» и «Python3». Можете проверить это:

```
      python --version
```

покажет версию Pithon 3.x. Если бы выполнили эту команду вне виртуального окружения - увидели бы 2.x.

Переход на другую подсистему хранения

Подсистема хранения (database engine, storage engine) — компонент системы управления базами данных (СУБД), управляющий механизмами хранения, или библиотека, подключаемая к программам и дающая им функции СУБД. На простом языке - «движок» базы данных.

При развертывании проекта мы, по очевидным причинам, больше не будем использовать базу данных «sqlite3». Вместо нее будет «PostgreSQL» или «MySQL». С точки зрения Django, такой переход не вызовет проблем: просто поменяем бекенд в файле настроек и выполним миграции. Для переноса данных необходимо до внесения изменений выполнить их экспорт при помощи команды «dumpdata»:

```
python manage.py dumpdata -e contenttypes -o db.json
```

Ключ «-e» позволяет опустить данные приложения или модели при экспорте. Ключ «-o» позволяет задать имя файла для экспорта. По умолчанию экспорт происходит в формате JSON, но можно задать и другие при помощи опции «--format».

Для перехода на базу данных «PostgreSQL» внесем изменения в файл настроек:

geekshop/geekshop/settings.py

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

```
    'default': {
        'NAME': 'geekshop',
        'ENGINE': 'django.db.backends.postgresql',
        'USER': 'django',
        'PASSWORD': 'geekbrains',
        'HOST': 'localhost'
    }
}...
```

Выключаем режим отладки

При выключении режима отладки необходимо заполнить список разрешенных хостов «ALLOWED_HOSTS»:

geekshop/geekshop/settings.py

```
...
DEBUG = False

ALLOWED_HOSTS = ['*']
...
```

Здесь мы разрешили доступ со всех хостов. Можно через запятую перечислить конкретные адреса.

Все. Наш проект готов к развертыванию.

Варианты развертывания

Для размещения сайта можно воспользоваться услугами хостинга или установить и настроить собственный сервер (необходим «белый», т.е. публичный, IP адрес).

Выбор хостинга - сложная задача. Для понимания механизма хостинга Python-проектов можно бесплатно зарегистрироваться на pythonanywhere.com.

Для развертывания реального проекта можете обратить внимание на:

- digitalocean.com;
- jino.ru;
- komtet.ru;
- locum.ru;
- hetzner.com;
- sweb.ru;

- hostfabrica.ru;
- fastvps.ru;
- reg.ru (см. инструкцию)..

Ситуация на рынке хостинга очень быстро меняется, поэтому лучше всего брать тестовый период и делать собственные выводы.

С точки зрения развертывания Django-проекта лучшим вариантом будет аренда VPS. Вы гарантированно установите нужную версию Django и получите больше возможностей для настроек.

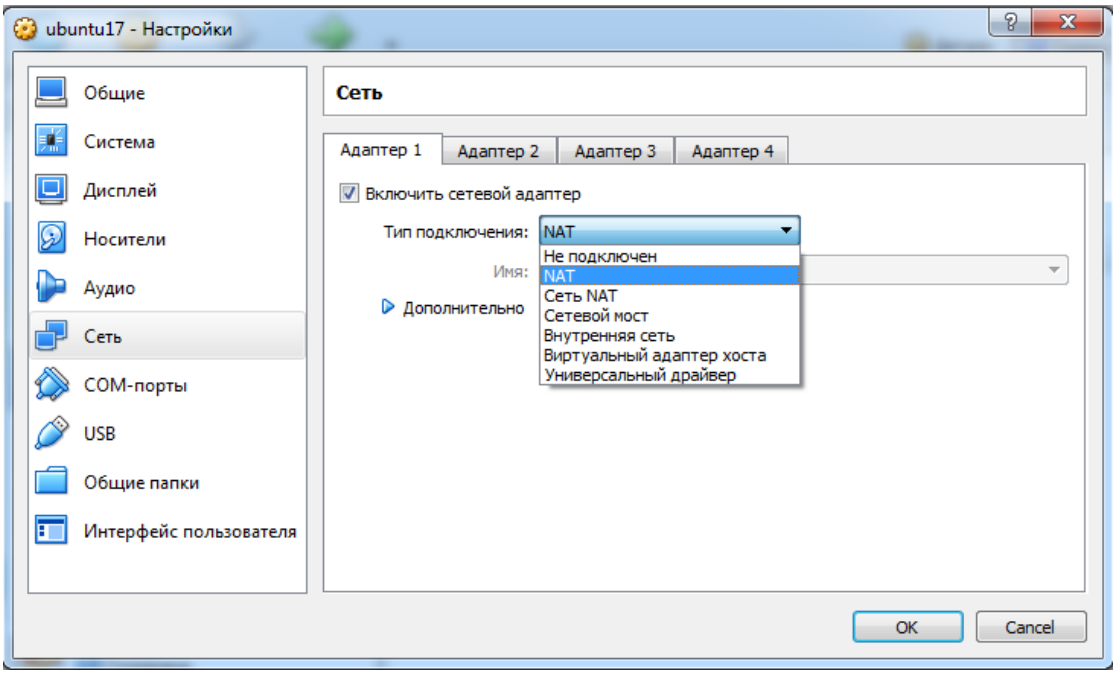
Именно такой вид развертывания мы рассмотрим в курсе.

Установка сервера

Первым шагом рекомендуем скачать и установить среду виртуализации, например www.virtualbox.org.

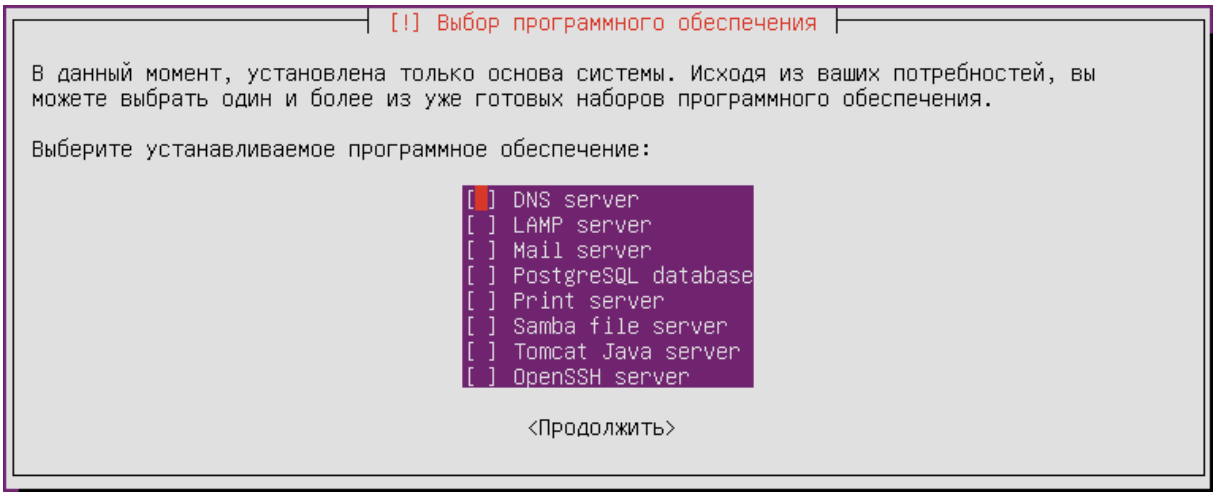
Для комфортной работы необходимо 8Гб ОЗУ и процессор с поддержкой аппаратной виртуализации VT-x (часто бывает выключена в BIOS - необходимо проверять).

Скачиваем дистрибутив «**Ubuntu 17.10.1 Server (64-bit)**» с официального сайта Ubuntu. И создаем новую виртуальную машину. В ее сетевых настройках выбираем вместо «NAT» вариант «Сетевой мост»:



При этом имитируется подключение сервера к вашей локальной сети.

Во вкладке «Носители» подключаем скачанный образ с дистрибутивом ОС и запускаем машину - начнется процесс установки. Все параметры оставляем по умолчанию. В конце установки будет меню со списком предустановленных программ:



Оставим без изменений - позже установим все необходимое вручную.

Итак, *исходное состояние*: чистый Ubuntu 17.10.1 Server (64-bit). Рекомендуем сделать снимок состояния виртуальной машины, чтобы в случае ошибки можно было «откатиться» к началу.

Настройка сервера

Настройка версии Ubuntu 17 Server несколько отличается от предыдущих редакций, например, Ubuntu 16 Server, поэтому, если вы используете другую версию, обратитесь к документации.

Настраиваем сеть и доступ по SSH

Первым шагом выполним настройку локальной сети:

```
django@ubuntu17django:~$ cd /etc/netplan
django@ubuntu17django:/etc/netplan$ ls
01-netcfg.yaml
django@ubuntu17django:/etc/netplan$ sudo nano 01-netcfg.yaml
```

01-netcfg.yaml

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s3:
      dhcp4: no
      dhcp6: no
      addresses: [192.168.0.98/24]
      gateway4: 192.168.0.1
      nameservers:
        addresses: [8.8.8.8, 192.168.0.1]
```

Для сохранения текста во встроенном редакторе nano используем сочетание «**Ctrl+O**», для выхода «**Ctrl+X**».

Важно: в качестве отступа использовать ровно 2 пробела.

Применяем конфигурацию сети и проверяем результат:

```
django@ubuntu17django:/etc/netplan$ sudo netplan apply
django@ubuntu17django:/etc/netplan$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1
000
    link/ether 08:00:27:e9:5b:b2 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.98/24 brd 192.168.0.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fee9:5bb2/64 scope link
       valid_lft forever preferred_lft forever
django@ubuntu17django:/etc/netplan$ _
```

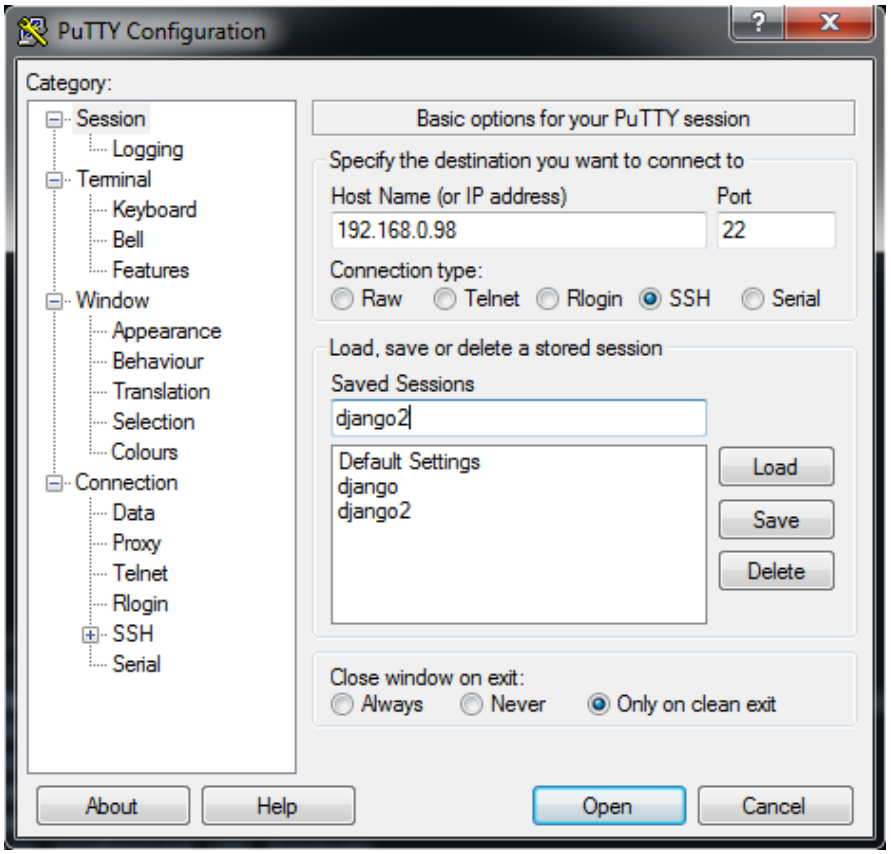
Проверьте связь из основной машины при помощи утилиты **ping**.

```
C:\Users\Jo>ping 192.168.0.98

Обмен пакетами с 192.168.0.98 по с 32 байтами данных:
Ответ от 192.168.0.98: число байт=32 время<1мс TTL=64
Ответ от 192.168.0.98: число байт=32 время<1мс TTL=64
Ответ от 192.168.0.98: число байт=32 время<1мс TTL=64
Ответ от 192.168.0.98: число байт=32 время<1мс TTL=64

Статистика Ping для 192.168.0.98:
    Пакетов: отправлено = 4, получено = 4, потеряно = 0
    (0% потерь)
Приблизительное время приема-передачи в мс:
    Минимальное = 0мсек, Максимальное = 0 мсек, Среднее = 0 мсек
```

В реальных условиях вы будете работать через удаленный терминал по протоколу SSH. Для этого установим SSH сервер:

```
sudo apt install openssh-server -y
```

Устанавливаем SSH-клиент в основную машину (например, **PuTTY**) и проверяем:



После ввода имени пользователя и пароля окажемся в терминале и дальше будем работать в нём.

Обеспечиваем обмен файлами по FTP

Устанавливаем FTP сервер:

```
sudo apt install vsftpd
```

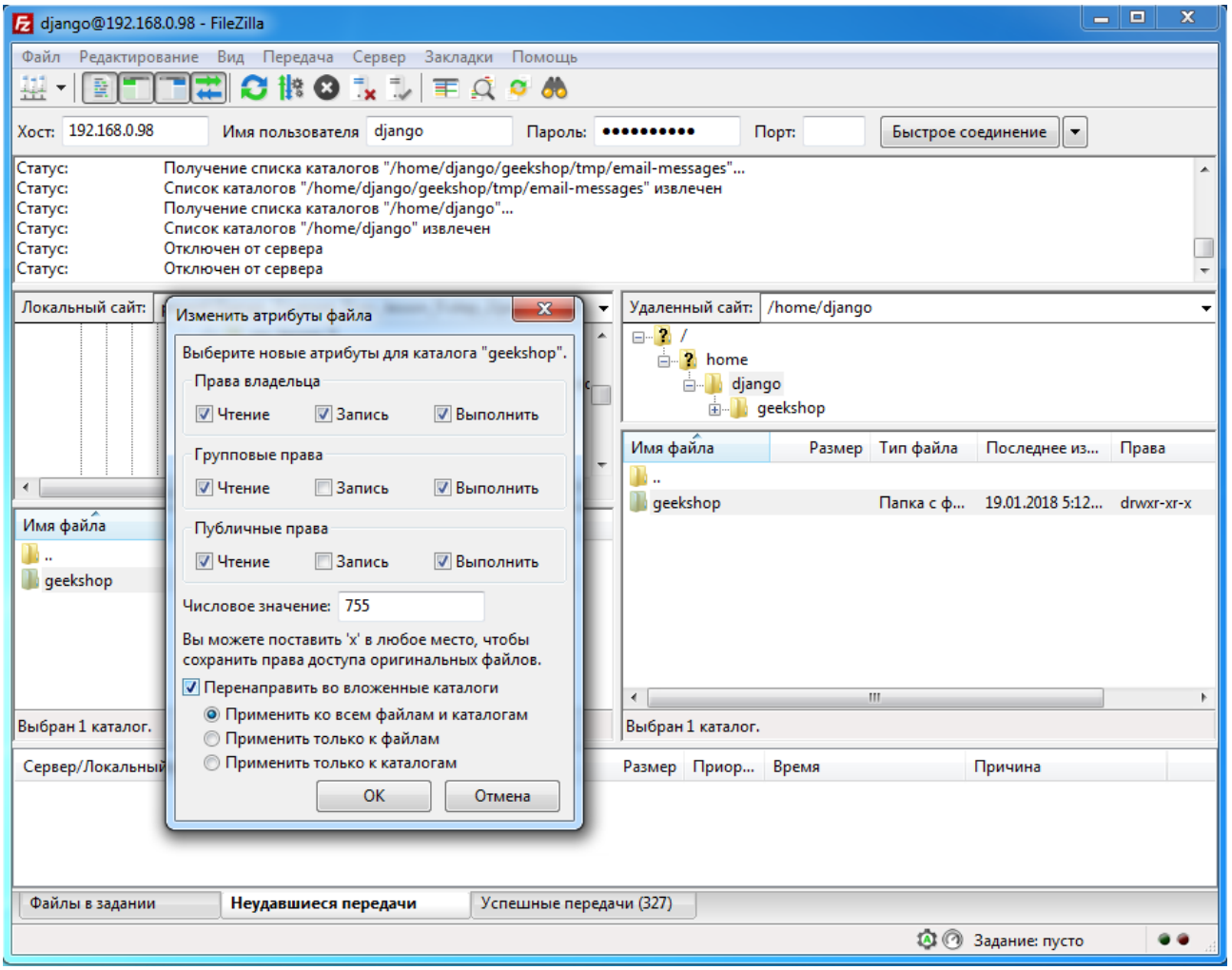
Разрешаем запись данных. Для этого необходимо раскомментировать строку «write_enable=YES» в файле конфигурации:

```
sudo nano /etc/vsftpd.conf
```

Перезапускаем службу FTP-сервера:

```
sudo systemctl restart vsftpd
```

Устанавливаем FTP-клиент (например «FileZilla») и копируем папку с проектом на сервер:



Не забываем настроить значение 755 атрибута доступа к папке и всем ее файлам.

Настраиваем окружение для проекта

В папке проекта на сервере (/home/django/geekshop/) создаем и активируем виртуальное окружение средствами Python 3:

```
python3 -m venv django2
source django2/bin/activate
```

Запустили «**venv**» (появился в Python 3.3) как Python-модуль.

Обновляем окружение из файла «requirements.txt»:

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

Если возникнут ошибки - устанавливаем пакеты вручную.

Сервер БД «PostgreSQL»

Устанавливаем:

```
sudo apt-get install postgresql postgresql-contrib
```

Запускаем интерпретатор команд сервера:

```
sudo -u postgres psql
```

Приглашение в терминале должно измениться на «postgres=#».

Выполняем команды:

```
CREATE DATABASE geekshop;
CREATE USER django with NOSUPERUSER PASSWORD 'geekbrains';
GRANT ALL PRIVILEGES ON DATABASE geekshop TO django;

ALTER ROLE django SET CLIENT_ENCODING TO 'UTF8';
ALTER ROLE django SET default_transaction_isolation TO 'READ COMMITTED';
ALTER ROLE django SET TIME ZONE 'Asia/Yekaterinburg';
```

Итак, мы создали новую базу и пользователя, настроили привилегии доступа пользователя к базе. Также задали параметры подключения: кодировку, уровень изоляции транзакций, временная зона.

Для выхода пишем «\q».

Миграции и импорт данных

Миграции выполняем как обычно:

```
python manage.py migrate
```

Если увидели ошибку - необходимо проверить настройки подключения к БД в проекте и статус сервера:

```
sudo systemctl status postgresql
```

После миграций импортируем данные:

```
python manage.py loaddata db.json
```

Пробуем запустить dev-сервер Django:

```
python manage.py runserver
```

Если сервер не запустился - ищем ошибку.

Развертывание Django-проекта при помощи «gunicorn»

Для работы любого сайта необходим web-сервер. В настоящее время самые популярные - «Apache» и «NGINX». Именно web-сервер отвечает на запросы пользователя. Нам необходимо обеспечить взаимодействие web-сервера и python-приложения. Наиболее популярные решения:

- mod_wsgi - кроссплатформенный модуль, предназначен для сервера «Apache»;
- uwsgi - кроссплатформенный модуль, может работать как самостоятельный сервер и в составе «Apache» и «NGINX»;
- gunicorn - высокопроизводительный модуль для *nix-систем.

Ситуация с эффективностью различных подходов со временем изменяется, поэтому необходимо проводить реальные тесты для конкретного проекта и принимать решение. В качестве примера можно посмотреть тестирование 2016 года.

В настоящее время большой популярностью пользуется модуль «gunicorn» - именно его и будем использовать.

С точки зрения операционной системы web-сервер и python-сервер - это два процесса. Для организации межпроцессного взаимодействия в *nix-системах существует несколько механизмов. Мы будем использовать «unix socket».

Начнем с установки модуля «gunicorn» в виртуальное окружение:

```
pip install gunicorn
```

Проверяем, что он работает - выполняем команду в папке проекта:

```
gunicorn geekshop.wsgi
```

```
(django2) django@ubuntu17django:~/geekshop$ gunicorn geekshop.wsgi
[2018-01-20 20:10:39 +0500] [1112] [INFO] Starting gunicorn 19.7.1
[2018-01-20 20:10:39 +0500] [1112] [INFO] Listening at: http://127.0.0.1:8000 (1112)
[2018-01-20 20:10:39 +0500] [1112] [INFO] Using worker: sync
[2018-01-20 20:10:39 +0500] [1115] [INFO] Booting worker with pid: 1115
[2018-01-20 20:10:44 +0500] [1112] [INFO] Handling signal: winch
```

Настроим параметры службы «**gunicorn**» для нашего проекта

```
sudo nano /etc/systemd/system/gunicorn.service
```

/etc/systemd/system/gunicorn.service

```
[Unit]
Description=gunicorn daemon
After=network.target

[Service]
User=django
Group=www-data
WorkingDirectory=/home/django/geekshop
ExecStart=/home/django/geekshop/django2/bin/gunicorn --access-logfile -
--workers 3 --bind unix:/home/django/geekshop/geekshop.sock geekshop.wsgi

[Install]
WantedBy=multi-user.target
```

Так как мы работаем в виртуальном окружении «django2», для запуска прописали путь:

```
/home/django/geekshop/django2/bin/gunicorn
```

Задаем параметры для запуска:

- **--access-logfile** - доступ к файлу логов для записи;
- **--workers** - количество работающих процессов;
- **--bind** - сокет для связи.

Разрешаем и запускаем службу «gunicorn», проверяем ее статус:

```
sudo systemctl enable gunicorn
sudo systemctl start gunicorn
sudo systemctl status gunicorn
```

Если все хорошо - устанавливаем web-сервер «nginx»:

```
sudo apt install nginx
```

Настраиваем параметры сайта «geekshop»:

```
sudo nano /etc/nginx/sites-available/geekshop
```

/etc/nginx/sites-available/geekshop

```
server {
    listen 80;
    server_name 192.168.0.98;

    location = /favicon.ico { access_log off; log_not_found off; }
    location /static/ {
        root /home/django/geekshop;
    }

    location /media/ {
        root /home/django/geekshop;
    }

    location / {
        include proxy_params;
        proxy_pass http://unix:/home/django/geekshop/geekshop.sock;
    }
}
```

Вместо доменного имени сайта написали IP-адрес. Настроили раздачу статических и медиафайлов:

```
location /static/ {}
location /media/ {}
```

Используем «nginx» как прокси для Python-сервера gunicorn:

```
proxy_pass http://unix:/home/django/geekshop/geekshop.sock;
```

Связь - через сокет «geekshop.sock».

Внимание: файл сокета не нужно создавать - это сделает процесс «gunicorn». Если возникнут ошибки при доступе к этому файлу - проверьте разрешения доступа для папки «geekshop» (должны быть 755).

Создаем ссылку в папке разрешенных сайтов «/etc/nginx/sites-enabled»:

```
sudo ln -s /etc/nginx/sites-available/geekshop /etc/nginx/sites-enabled
```

Проверяем настройки «nginx»:

```
sudo nginx -t
```

Перезапускаем службу «nginx» и добавляем разрешения в сетевой экран:

```
sudo systemctl restart nginx
sudo ufw allow 'Nginx Full'
```

Всё. Можно проверить работу нашего сайта, набрав IP адрес сервера в браузере:

```
192.168.0.98
```

В случае ошибок проверяем логи сервера **nginx**:

```
sudo tail -F /var/log/nginx/error.log
```

*Управление «gunicorn» при помощи «supervisor»

В принципе, после установки статуса enabled для службы «gunicorn», она должна запускаться автоматически после перезагрузки сервера. Этот момент нужно обязательно проверить.

Также для управления запуском «gunicorn» можно воспользоваться полезным инструментом «supervisor»:

```
sudo apt install supervisor
```

По сути, это альтернатива созданной ранее службе «/etc/systemd/system/gunicorn.service».

Практическое задание

1. Создать файл зависимостей «requirements.txt» для проекта.
2. Экспортировать данные из базы.
3. Установить и настроить сервер Ubuntu Server 17.
4. Развернуть проект на сервере.

Так как образ виртуальной машины достаточно большого размера, вместо него необходимо в архиве с ДЗ выслать скриншоты с выполненными шагами. Если на каком-то шаге начались проблемы - необходимо написать о них в файле «readme.txt». Если удастся развернуть проект на реальном хостинге - высылайте ссылку.

Дополнительные материалы

Все то, о чем сказано в методичке, но подробнее:

1. pip-tools
2. dumpdata
3. Virtualbox
4. Аппаратная виртуализация
5. Процессоры Intel
6. Модуль venv
7. Apache + python = mod_wsgi
8. Uwsgi
9. Gunicorn
10. Документация по gunicorn
11. Тест серверов для python приложений (2016)
12. Параметры gunicorn
13. Система управления процессами Supervisor

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Официальная документация