Python/Django Уровень 2

Урок 1

Отправка электронной почты. Контекстные процессоры

Переходим на новый Django. Отправляем сообщение с кодом активации пользователя. Работаем с контекстными процессорами на примере корзины.

Django new: создаем виртуальное окружение и адаптируем проект

Фреймворк Django регулярно обновляется и представляет новые релизы версий. Конечно, переход хостингов на новую версию выполняется постепенно, но он неизбежен. Поэтому в курсе «Django. Уровень 2» будем работать именно с новой версией. Хорошая новость заключается в том, что для запуска кода, созданного на 1 уровне, необходимы минимальные правки.

VIRTUALENV – создаем виртуальное окружение

Часто возникает необходимость работать с разными наборами пакетов и их версий в Python. Для этого существует удобный инструмент, позволяющий создавать различные виртуальные окружения и переключаться между ними: virtualenv.

Установка и работа в Windows:

```
pip install virtualenv
```

Для создания окружения с именем «django_new» в корневой папке проекта (где находится файл «manage.py») выполняем команду:

```
virtualenv django_new
```

После этого появится папка с именем окружения, содержащая его настройки.

Для активации окружения выполняем команду (важно использовать именно символ «\»):

```
django_new\scripts\activate
```

Мы только что выполнили файл «activate.bat», расположенный в папке «scripts». А сейчас, если все прошло успешно – увидим в круглых скобках имя окружения перед путем к текущей папке. Теперь необходимо установить пакеты, которые используются приложением и, конечно, сам Django:

```
pip install django
pip install pillow
```

Если попробуем запустить development-сервер Django: увидим сообщения об ошибках – это правильно, так как наш проект был написан для другой, старой версии Django.

Установка и работа в Ubuntu 17 (в других версиях могут быть отличия):

```
sudo apt update
```

```
sudo apt install python3-pip
pip3 install virtualenv
sudo apt install virtualenv
```

Далее окружение создается такой же командой, как и в Windows. Активация окружения:

```
source django_new/bin/activate
```

Альтернативный вариант для Python 3.6 и новее:

```
sudo apt install python3-venv
python3 -m venv django_new
source django_new/bin/activate
```

Установку пакетов в *nix системах проводим при помощи «pip3».

Выполняем проверку версии Django в любой системе командой:

```
django-admin --version
```

Если увидели новую версию Django – можно двигаться дальше.

Выйти из виртуального окружения можно, выполнив команду:

```
deactivate
```

Следует иметь ввиду, что активацию окружения необходимо выполнять каждый раз при открытии окна с командной строкой. Запуск скриптов через файлы «.bat», которые создали на курсе «Django 1» будет происходить как до создания виртуального окружения.

Существует обертка для VIRTUALENV – пакет virtualenvwrapper (для Windows: virtualenvwrapper-win). Он делает работу с виртуальным окружением еще более удобным. Обязательно поработайте с ней (требуется время на настройку и освоение).

Замечание: следует учитывать, что объем папки (в нашем случае «django_new»), которая появляется в проекте при работе с виртуальным окружением через VIRTUALENV, может быть значительным – в ней находятся все установленные в окружение пакеты. Это с одной стороны плюс – можно запускать проект в этом окружении на другой системе независимо от ее настроек. С другой стороны минус – значительно увеличивается размер проекта (важно в нашем курсе при отправке практического задания).

Дальше в проекте будем предполагать, что Django установлен в основной системе, а для предыдущих проектов на старой версии Django создано виртуальное окружение, например, «Django_1».

Напомним команды удаления и установки Django.

```
pip uninstall django
```

```
pip install django==1.11  # указана ваша старая версия Django
```

Адаптация старого проекта Django для запуска в новом Django

В моделях Django при использовании полей «models.ForeignKey()» теперь обязательно необходимо задавать значение аргумента «on_delete» (обычно «on_delete=models.CASCADE»). В нашем проекте это требование уже выполнено - ошибок не должно быть. Но стоит проверить, прежде чем двигаться дальше.

В Django существенно изменился диспетчер URL. Вместо функции «url()» теперь будем использовать «re_path()» (аналог «url()») или «path()» (работа с путями без регулярных выражений)). Также, при использовании пространств имен, в файле «urls.py» каждого приложения необходимо наличие строки с его именем:

```
app_name = 'mainapp'
```

Исправим файл главного диспетчера URL:

geekshop/urls.py

```
from django.conf.urls import include
 ...
from django.urls import re_path

urlpatterns = [
    re_path(r'^$', mainapp.main, name='main'),
    re_path(r'^products/', include('mainapp.urls', namespace='products')),
    ...
]
...
```

В диспетчерах URL приложений также выполняем правки.

mainapp/urls.py

```
import mainapp.views as mainapp
from django.urls import re_path

app_name="mainapp"

urlpatterns = [
    re_path(r'^$', mainapp.products, name='index'),
    re_path(r'^category/(?P<pk>\d+)/$', mainapp.products, name='category'),
    re_path(r'^product/(?P<pk>\d+)/$', mainapp.product, name='product'),
    re_path(r'^category/(?P<pk>\d+)/page/(?P<page>\d+)/$', mainapp.products,
name='page'),
]
```

После исправлений проект должен запуститься в виртуальном окружении «Django_new»:

```
(django_new)C:\PyProjects\geekshop>python manage.py runserver
```

Если этого не произошло, возможно была допущена ошибка при исправлениях, либо вы использовали модули Django, которые были существенно изменены.

Отправка электронной почты в Django. Активация пользователя по e-mail

Настройка проекта Django для отправки почты

Для отправки почты в Django необходимо в файл настроек добавить настройки сервера исходящей почты (SMTP):

geekshop/settings.py

```
...
DOMAIN_NAME = 'http://localhost:8000'

EMAIL_HOST = 'localhost'
EMAIL_PORT = '25'
EMAIL_HOST_USER = 'django@geekshop.local'
EMAIL_HOST_PASSWORD = 'geekshop'
EMAIL_USE_SSL = False

#вариант python -m smtpd -n -c DebuggingServer localhost:25
# EMAIL_HOST_USER, EMAIL_HOST_PASSWORD = None, None

#вариант логирования сообщений почты в виде файлов вместо отправки
# EMAIL_BACKEND = 'django.core.mail.backends.filebased.EmailBackend'
# EMAIL_FILE_PATH = 'tmp/email-messages/'
```

Поясним на примере реальных настроек SMTP mail.ru:

- EMAIL_HOST – URL адрес почтового сервера (smtp.mail.ru).
- EMAIL_PORT – порт сервера (465).
- EMAIL_HOST_USER – имя пользователя, от которого будет отправлена почта (someuser@mail.ru).
- EMAIL_HOST_PASSWORD – пароль пользователя, от которого будет отправлена почта.
- EMAIL_USE_SSL – флаг использования шифрования (True).

Если сервер поддерживает шифрование TLS – используем настройку:

```
EMAIL_USE_TLS = True
```

Константу «DOMAIN_NAME» будем использовать при формировании почтового сообщения пользователю. В будущем её необходимо заменить на реальное доменное имя ресурса.

Работа с почтой на этапе разработки Django-проекта

Для тестирования работы с электронной почтой можно пойти разными путями:

- использовать реальный сервер и учетную запись – хороший вариант, но не подходит для нашего курса (ваши учетные данные будут видны участникам проекта);
- установить локальный почтовый сервер (например, в Windows hmailserver) и учетные записи на нем – отличный вариант (именно для него приведены настройки «settings.py»), но требует времени на настройку;
- использовать встроенный в Python «Debugging» сервер – позволяет отслеживать работу с почтой в консоли, запуск:

```
python -m smtpd -n -c DebuggingServer localhost:25
```

- подключить Django-backend 'django.core.mail.backends.filebased.EmailBackend', создающий файлы с сообщениями, вместо реальной отправки.

В исходниках и при отправке практического задания мы будем пользоваться третьим или четвертым методом. Но настоятельно рекомендуем протестировать работу с реальным почтовым сервером.

Модель пользователя: создаем и сохраняем код подтверждения

Добавим в код модели строки:

authapp/models.py

```
from django.db import models
from django.contrib.auth.models import AbstractUser
from django.utils.timezone import now
from datetime import timedelta

class ShopUser(AbstractUser):
    avatar = models.ImageField(upload_to='users_avatars', blank=True)
    age = models.PositiveIntegerField(verbose_name = 'возраст')

    activation_key = models.CharField(max_length=128, blank=True)
    activation_key_expires = models.DateTimeField(      default=(now() +
timedelta(hours=48)))


    def is_activation_key_expired(self):
        if now() <= self.activation_key_expires:
            return False
        else:
            return True
```

Сам ключ подтверждения «activation_key» будем создавать с помощью хеш-функции при регистрации пользователя. Срок действия ключа храним в атрибуте «activation_key_expires» - задали 48 часов от момента создания учетной записи. Метод «is_activation_key_expired()» выполняет проверку актуальности ключа.

После внесения изменений, выполняем миграции.

Регистрация пользователя: работаем с методом «.save()» формы

При регистрации пользователя теперь необходимо выполнить два дополнительных действия:

- создать и сохранить в модели код подтверждения;
- сделать пользователя неактивным.

Для реализации этого функционала переопределим метод «.save()» формы регистрации.

authapp/forms.py

```
...
import random, hashlib
...

class ShopUserRegisterForm(UserCreationForm):
    ...

    def save(self):
        user = super(ShopUserRegisterForm, self).save()

        user.is_active = False
        salt = hashlib.sha1(str(random.random()).encode('utf8')).hexdigest()[:6]
        user.activation_key = hashlib.sha1((user.email +
salt).encode('utf8')).hexdigest()
        user.save()

        return user
```

Оригинальный метод «.save()» создает объект пользователя и возвращает нам его:

```
user = super(ShopUserRegisterForm, self).save()
```

При вычислении ключа активации можно использовать модификатор (соль):

```
salt = hashlib.sha1(str(random.random()).encode('utf8')).hexdigest()[:6]
```

Не забываем сохранить (user.save()) и вернуть объект пользователя (return user).

Замечание: советуем более подробно познакомиться с библиотекой хеш-функции hashlib.

Отправка почтового сообщения пользователю

Можно было для работы с почтовыми сообщениями создать новое приложение. Но мы пока ограничимся добавлением функции «send_verify_mail()» в файл «views.py» приложения «authapp»:

authapp/views.py

```
...
from django.core.mail import send_mail
from django.conf import settings
from authapp.models import ShopUser
...

def send_verify_mail(user):
    verify_link = reverse('auth:verify', args=[user.email, user.activation_key])

    title = f'Подтверждение учетной записи {user.username}'

    message = f'Для подтверждения учетной записи {user.username} на портале \
{settings.DOMAIN_NAME} перейдите по ссылке:
\n{settings.DOMAIN_NAME}{verify_link}'

    return send_mail(title, message, settings.EMAIL_HOST_USER, [user.email],
fail_silently=False)
```

Для формирования ссылки подтверждения «verify_link» создадим в диспетчере адресов приложения «authapp» запись:

```
re_path(r'^verify/(?P<email>.+)/(?P<activation_key>\w+)/$', authapp.verify,
name='verify'),
```

Функция отправки сообщения «send_mail()» находится в модуле «django.core.mail». Ей передаём заголовок сообщения, текст сообщения, адрес отправителя, список адресов получателей и параметр «fail_silently» (при значении False, в случае неудачной отправки, генерируется ошибка smtplib.SMTPException). В точке вызова функции «send_mail()» получим число успешно отправленных сообщений.

Добавим вызов функции «send_mail()» в контроллер регистрации пользователя:

authapp/views.py

```
def register(request):
    title = 'регистрация'

    if request.method == 'POST':
        register_form = ShopUserRegisterForm(request.POST, request.FILES)
        if register_form.is_valid():
            user = register_form.save()
            if send_verify_mail(user):
                print('сообщение подтверждения отправлено')
                return HttpResponseRedirect(reverse('auth:login'))
            else:
                print('ошибка отправки сообщения')
                return HttpResponseRedirect(reverse('auth:login'))
        else:
            register_form = ShopUserRegisterForm()
            content = {'title': title, 'register_form': register_form}
            return render(request, 'authapp/register.html', content)
```

Теперь можно запускать проект и проверять отправку сообщения.

Для этого запустим из консоли встроенный в Python SMTP-сервер:

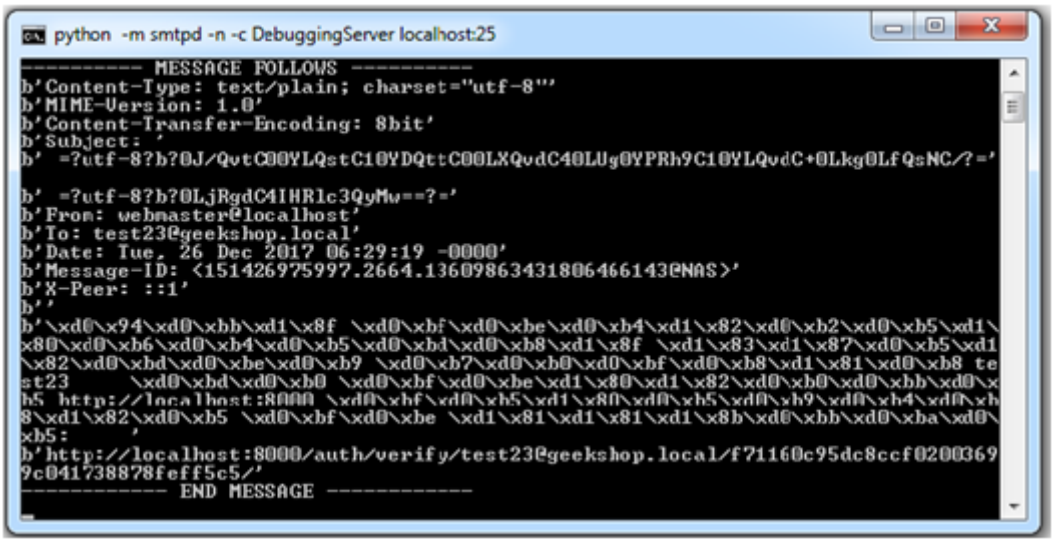
```
python -m smtpd -n -c DebuggingServer localhost:25
```

Зайдем на страницу регистрации нового пользователя и создадим его. При этом увидим ошибку:

Причина – Python SMTP-сервером не поддерживается аутентификация. Если в файле настроек раскомментировать строку

```
EMAIL_HOST_USER, EMAIL_HOST_PASSWORD = None, None
```

увидим отправленное пользователю сообщение почты в консоли:



Можно сделать вывод, что отправка сообщений работает. В принципе, уже сейчас можно для проверки скопировать ссылку активации прямо из консоли, но это не очень удобно. Попробуем вариант с «EMAIL_BACKEND».

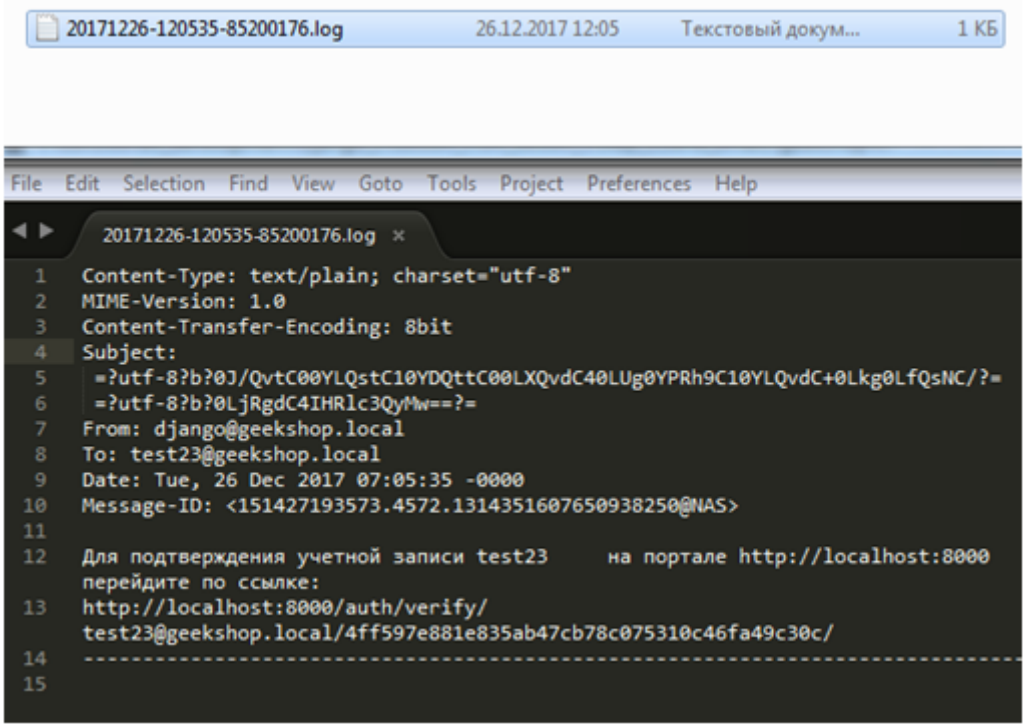
geekshop/settings.py

```
...
DOMAIN_NAME = 'http://localhost:8000'
EMAIL_HOST = 'localhost'
EMAIL_PORT = '25'
EMAIL_HOST_USER = 'django@geekshop.local'
EMAIL_HOST_PASSWORD = 'geekshop'
EMAIL_USE_SSL = False

#вариант python -m smtpd -n -c DebuggingServer localhost:25
# EMAIL_HOST_USER, EMAIL_HOST_PASSWORD = None, None

#вариант логирования сообщений почты в виде файлов вместо отправки
EMAIL_BACKEND = 'django.core.mail.backends.filebased.EmailBackend'
EMAIL_FILE_PATH = 'tmp/email-messages/'
```

Снова регистрируем пользователя и видим результат в папке «tmp/email-messages/»:



Именно такое сообщение получит пользователь по почте. Обязательно проверьте работу с реальным сервером (начальный вариант настроек «settings.py»).

Итак, мы организовали отправку сообщения электронной почты средствами Django. Аналогичным образом можно организовать рассылку административных сообщений или другой информации.

Активация пользователя

Теперь необходимо реализовать механизм активации пользователя при переходе по ссылке из сообщения. URL адрес уже прописан в диспетчере. Остался контроллер:

authapp/views.py

```
def verify(request, email, activation_key):
    try:
        user = ShopUser.objects.get(email=email)
        if user.activation_key == activation_key and not
user.is_activation_key_expired():
            user.is_active = True
            user.save()
            auth.login(request, user)
            return render(request, 'authapp/verification.html')
        else:
            print(f'error activation user: {user}')
            return render(request, 'authapp/verification.html')
    except Exception as e:
        print(f'error activation user : {e.args}')
        return HttpResponseRedirect(reverse('main'))
```

Проверяем совпадение ключа активации и его срок действия. Если все хорошо – активируем и осуществляем логин пользователя на сайт. Показываем приветственное окно:

authapp/templates/authapp/verification.html

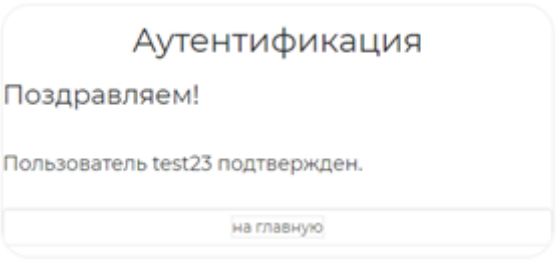
```
{% extends 'authapp/base.html' %}
{% load staticfiles %}

{% block content %}

  {% if user %}
    <h3>Поздравляем!</h3>
    <br>
    <h4>Пользователь {{user.username}} подтвержден.</h4>
  {% else %}
    <h3>Верификация не пройдена.</h3>
  {% endif %}
  <br>
  <button class="btn btn-round form-control">
    <a href="{% url 'main' %}" class="">на главную</a>
  </button>

{% endblock %}
```

Проверяем:



Можно считать задачу подтверждения подлинности пользователя по e-mail решенной.

Контекстные процессоры в Django

Если проанализировать контроллеры нашего приложения «mainapp», то обнаружим, что почти в каждом из них в контексте передается корзина:

```
'basket': get_basket(request.user),
```

В Django существует механизм, позволяющий передавать в шаблоны данные без участия контроллеров: контекстные процессоры. Для создания своего контекстного процессора добавим строку в файл настроек:

```
'mainapp.context_processors.basket'
```

geekshop/settings.py

```
...
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'mainapp.context_processors.basket',
            ],
        },
    },
]
...
```

Как видно из файла настроек, мы уже пользовались контекстными процессорами в проекте: «request» и «auth».

Файл контекстного процессора можно расположить в любом приложении. Мы поместим его в «mainapp»:

mainapp/context_processors.py

```
from basketapp.models import Basket

def basket(request):
    print(f'context processor basket works')
    basket = []

    if request.user.is_authenticated:
        basket = Basket.objects.filter(user=request.user)

    return {
        'basket': basket,
    }
```

Теперь можно убрать ключ «basket» из контекста всех контроллеров в приложении «mainapp» и всё будет работать как раньше, но с той разницей, что теперь доставкой корзины в шаблон занимаются не контроллеры, а наш контекстный процессор.

Практическое задание

1. Установить VENV и создать два виртуальных окружения: для старой версии вашего Django проекта и для нового Django. Проверить их работу. Установить новый Django в основную систему.
2. Адаптировать Django проект для запуска.
3. Организовать выдачу сообщения об успешной отправке письма с кодом подтверждения в окне регистрации пользователя.
4. Реализовать активацию пользователя при переходе по ссылке из письма.
5. Создать контекстный процессор для корзины и скорректировать код контроллеров основного приложения.

При отправке практического задания *не нужно* отправлять папку с виртуальным окружением (если она есть в проекте – просто удалите). Можно проверить по объему: если больше 50 МБ – значит папка с виртуальным окружением есть.

Дополнительные материалы

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

1. Django 3.2
2. virtualenv
3. virtualenvwrapper
4. virtualenvwrapper-win
5. Настройки SMTP mail.ru
6. hmailserver
7. «Соль» в криптографии
8. hashlib

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Официальная документация