Отправка электронной почты. Контекстные процессоры

Переходим на Django 2.0. Отправляем сообщение с кодом активации пользователя. Работаем с контекстными процессорами на примере корзины.

[Django 2.0: создаем виртуальное окружение и адаптируем проект](#_3znysh7)

[VIRTUALENV – создаем виртуальное окружение](#_2et92p0)

[Установка и работа в Windows:](#_o1jy6e1ihv6c)

[Установка и работа в Ubuntu 17 (в других версиях могут быть отличия):](#_qelcr0uwfq5c)

[Адаптация проекта Django 1.11 для запуска в Django 2.0](#_y2p3pwo37dev)

[Отправка электронной почты в Django. Активация пользователя по e-mail](#_tyjcwt)

[Настройка проекта Django для отправки почты](#_z34ecfn2t0vj)

[Работа с почтой на этапе разработки Django-проекта](#_f1y90v67f7we)

[Модель пользователя: создаем и сохраняем код подтверждения](#_ibcftutchvng)

[Регистрация пользователя: работаем с методом «.save()» формы](#_6truyvvubpbj)

[Отправка почтового сообщения пользователю](#_5sn8mvknate)

[Активация пользователя](#_8vhpqydhhpme)

[Контекстные процессоры в Django](#_kqg8w9z1264o)

[Практическое задание](#_z337ya)

[Дополнительные материалы](#_3j2qqm3)

[Используемая литература](#_1y810tw)

# 

# Django 2.0: создаем виртуальное окружение и адаптируем проект

2 декабря 2017 года в истории фреймворка Django произошло знаменательное событие - был представлен релиз версии 2.0. 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](https://virtualenv.pypa.io/en/stable/).

### Установка и работа в Windows:

|  |
| --- |
| *pip* install virtualenv |

Для создания окружения с именем «django\_2.0» в корневой папке проекта (где находится файл «manage.py») выполняем команду:

|  |
| --- |
| virtualenv django\_2.0 |

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

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

|  |
| --- |
| django\_2.0\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==2.0 pip **install** pillow |

Если попробуем запустить development-сервер Django: увидим сообщения об ошибках – это правильно, так как наш проект был написан для Django 1.11.

### 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\_2.0*/bin/*activate |

Альтернативный вариант для Python3.6 и новее:

|  |
| --- |
| sudo apt **install** python3-venv python3 -m venv django\_2.0  **source** django\_2.0*/bin/*activate |

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

Выполняем проверку версии Django в любой системе командой:

|  |
| --- |
| **django**-admin *--version* |

Если увидели «2.0» – 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](https://virtualenvwrapper.readthedocs.io/en/latest/) (для Windows: [virtualenvwrapper-win](https://pypi.python.org/pypi/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\_2.0»), 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 2.0 установлен в основной системе, а для предыдущих проектов на Django 1.11 и более ранних, создано виртуальное окружение, например, «Django\_1.11».

Напомним команды удаления и установки Django.

|  |
| --- |
| pip **uninstall** django pip **install** django==1.11 |

## Адаптация проекта Django 1.11 для запуска в Django 2.0

В моделях Django 2.0 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 2.0 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\_2.0»:

|  |
| --- |
| (django\_2.0)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](https://help.mail.ru/mail-help/mailer/popsmtp):

* 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](https://www.hmailserver.com/)) 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](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BB%D1%8C_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F))):

|  |
| --- |
| 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*](https://docs.python.org/3/library/hashlib.html)*.*

## 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](https://docs.python.org/3/library/smtplib.html#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 |

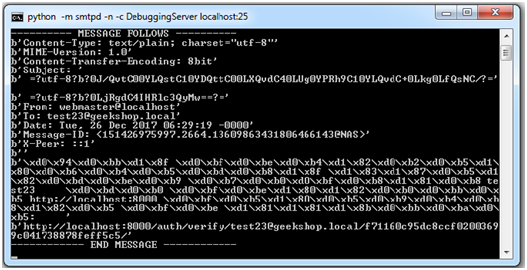
Зайдем на страницу регистрации нового пользователя и создадим его. При этом увидим ошибку:



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:

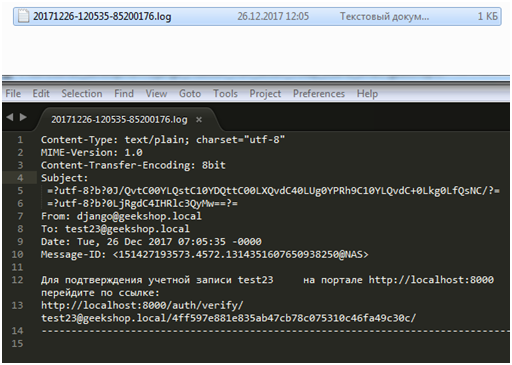


Можно сделать вывод, что отправка сообщений работает. В принципе, уже сейчас можно для проверки скопировать ссылку активации прямо из консоли, но это не очень удобно. Попробуем вариант с «EMAIL\_BACKEND».

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

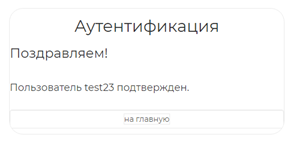
|  |
| --- |
| **def** **verify**(request, email, activation\_key):  **try**:  user = ShopUser.objects.get(email=email)  **if** user.activation\_key == activation\_key **and** **not** user.is\_activation\_key\_expired():  user.is\_active = **True**  user.save()  auth.login(request, user)  **return** 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.

# 

EMAIL\_HOST = 'smtp.yandex.ru'

EMAIL\_HOST\_PASSWORD = 'password'

EMAIL\_HOST\_USER = 'andreysperansky@yandex.ru'  
EMAIL\_PORT = '25'  
EMAIL\_USE\_SSL = *False*

SERVER\_EMAIL = EMAIL\_HOST\_USER

DEFOULT\_FROM\_EMAIL = EMAIL\_HOST\_USER

# чтобы приходили уведомления о 500х ошибках

ADMINS = (

('Andrey', andreysperansky@)

)

MANAGERS = ADMINS

# 

# Практическое задание

1. Установить VENV и создать два виртуальных окружения: для Django 1.11 и Django 2.0. Проверить их работу. Установить Django 2.0 в основную систему.
2. Адаптировать выполненный на Django 1.11 проект для запуска в Django 2.0.
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 2.0](https://docs.djangoproject.com/en/2.0/releases/2.0/)
2. [virtualenv](https://virtualenv.pypa.io/en/stable/)
3. [virtualenvwrapper](https://virtualenvwrapper.readthedocs.io/en/latest/)
4. [virtualenvwrapper-win](https://pypi.python.org/pypi/virtualenvwrapper-win)
5. [Настройки SMTP mail.ru](https://help.mail.ru/mail-help/mailer/popsmtp)
6. [hmailserver](https://www.hmailserver.com/)
7. [«Соль» в криптографии](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BB%D1%8C_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F))
8. [hashlib](https://docs.python.org/3/library/hashlib.html)

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. [Официальная документация](https://docs.djangoproject.com/en/2.0/)