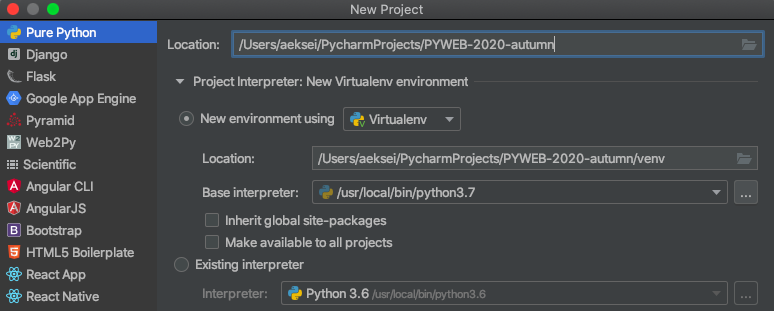
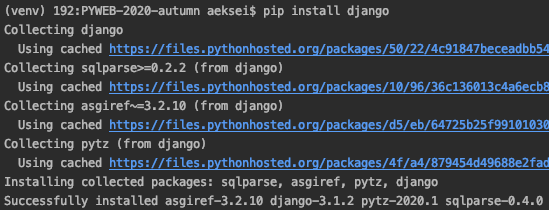
# Создание Django проекта и приложений

1. Создаем чистый проект (Pure Python) и создаем для него виртуальное окружение, чтобы можно было выделить зависимости, которые будет необходимы для конкретного django проекта.



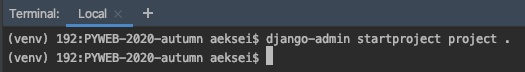
1. Далее в терминале IDE PyCharm нужно установить пакет Django с помощью команды

| pip install django |
| --- |

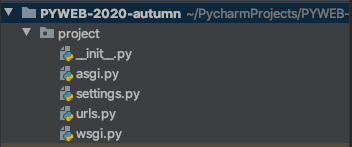


1. С помощью команды django-admin startproject <название django проекта> . необходимо создать новый проект. Точка в конце указывает, что проект сделает создать в текущей директории

| django-admin startproject project . |
| --- |



После этого у вас создастся django проект

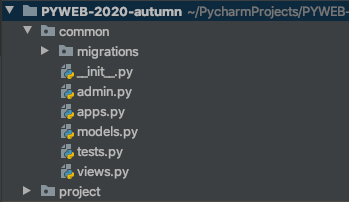


1. Создадим django приложение под названием **common**  с помощью команды python manage.py startapp <название приложения>

| python3.7 manage.py startapp common |
| --- |



И получим вот такой результат



1. В файле views.py, отвечающего за слой представлений, приложения common необходимо создать классовое представление, которое будет возвращать пользователю текущее время. Для этого необходимо создать класс CurrentDateView, в котором будет функция, которая обрабатывает GET запросы.

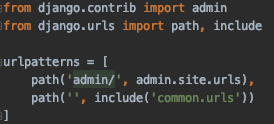
| from datetime import datetime  from django.views import View from django.http import HttpResponse   class CurrentDateView(View):  def get(self, request):  html = f"{datetime.now()}"  return HttpResponse(html) |
| --- |

1. Теперь необходимо связать это представление с конкретным URL. Для в приложении common создадим файл urls.py (автоматические не создается при создании приложения) и поместим туда следующий код:

| from django.urls import path  from .views import CurrentDateView  urlpatterns = [  path('datetime/', CurrentDateView.as\_view()), ] |
| --- |

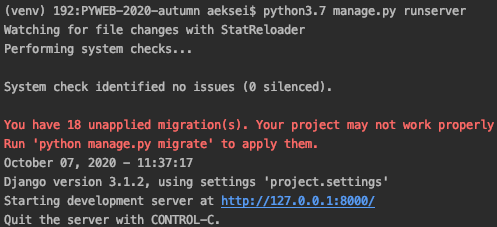
Где с помощью функции path создается маршрут, который связывает url **datetime/** и представление CurrentDateView.as\_view(), которые импортировано из модуля views.py приложения common.

1. Но Django проект не знает о существовании файла urls.py из приложения common, поэтому необходимо включить все маршруты приложения common в файл urls.py **проекта.** Сделать это можно с помощью команды include.

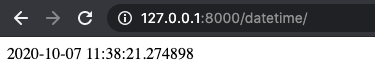


1. Теперь можем запуска сервер и проверять результаты:

| python3.7 manage.py runserver |
| --- |



Если всё успешно, то вы получить текущее время на момент запроса.



**Самостоятельно**:

1. В приложении **test\_app** создайте новое представление (View) используя классовые представления, которое будет возвращать случайное число в браузере.

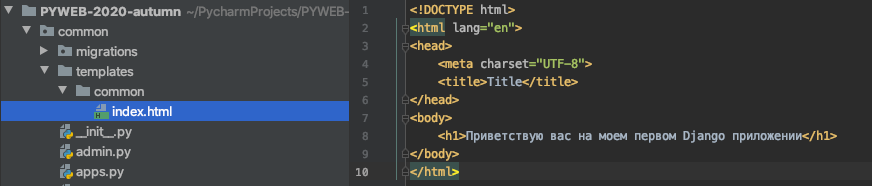
| from random import random  random\_number = random() |
| --- |

1. Создайте новое приложение **common**, в нем представление, которое будет возвращать в браузер строку """<h1>Hello, World</h1>""" по url: **hello/**

# Работа с шаблонами и статическим файлами

1. В приложении common создайте папку “templates”, в котором будут храниться шаблоны для данного приложения. В папке “templates” создайте папку с названием соответствующего приложения, в данном случае это common. И в конце следует создать html шаблон (html файл) под названием index.html, который будет хранить следующий код:

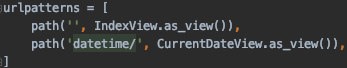
| <!DOCTYPE html> <**html** lang="en"> <**head**>  <**meta** charset="UTF-8">  <**title**>Title</**title**> </**head**> <**body**>  <**h1**>Приветствую вас на моем первом Django приложении</**h1**> </**body**> </**html**> |
| --- |



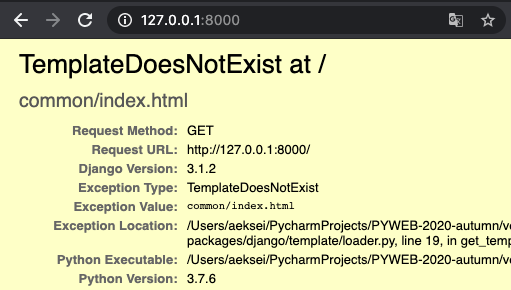
1. После необходимо сделать представление, которое будет использовать данный шаблон:

| from django.shortcuts import render  class IndexView(View):  def get(self, request):  return render(request, 'common/index.html') |
| --- |

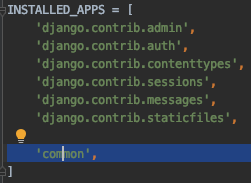
1. Не забываем подключить представление в файл urls.py приложения common:



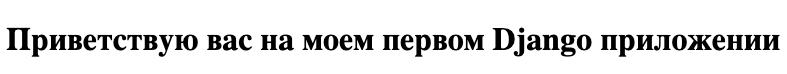
1. Пробуем проверить результат



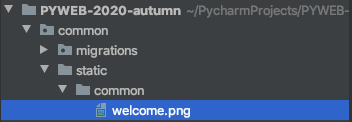
И получаем вот такую ошибку. Связана она с тем, что Django проект пока что не умеет работать с шаблонами приложений. Для этого нужно подключить приложение к проекту. В файле settings.py проекта находим переменную INSTALLED\_APPS и дописывает в конец списка наше приложение:



1. Обновляем страницу и видим нужную страницу



1. Теперь добавим статические файлы на главную страницу. Статические файлы приложения должны храниться в папке static/<название приложения> (по аналогии с templates). Скачиваем из интернета нужную картинку и помещаем в указанную директорию:

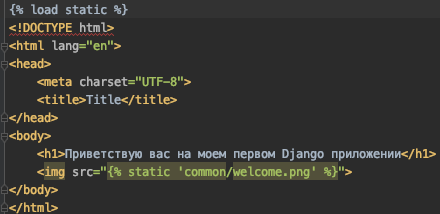


1. Далее в шаблон index.html нужно добавить html тег img.

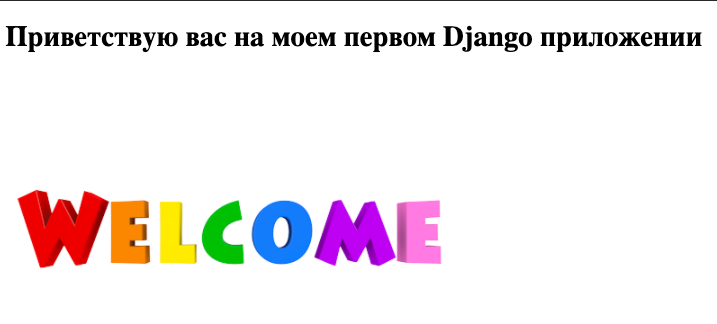
| <img src="путь\_до\_изображения"> |
| --- |

Наша задача, научить работать шаблоны со статическими файлами с помощью Django. Для этого есть специальные теги. Для начала нужно добавить в начало html страницы тег **{% load static %}.** После чего все пути до статических файлов должны выглядеть следующим образом:

| {% static '<**название\_приложения**>/<**необходимый** файл>' %} |
| --- |

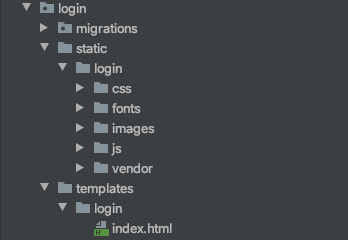


1. Обновляем страницу и убеждаемся, что картинка подгружается

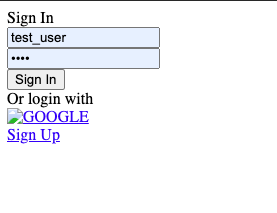


**Самостоятельно:**

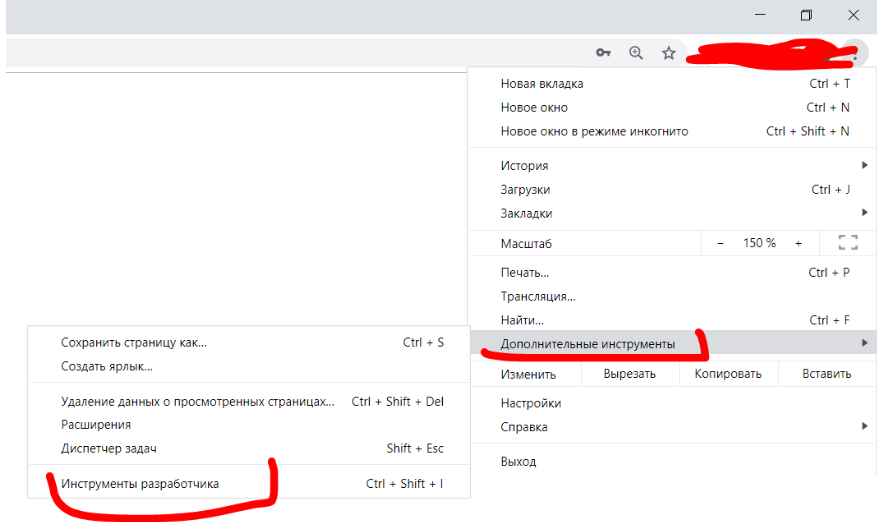
1. Создайте приложение login
2. Создайте папки templates и static.
3. Зайдите на сайт: <https://www.internet-technologies.ru/articles/60-besplatnyh-html5-i-css3-form-avtorizacii-dlya-sayta.html>
4. Выберите шаблон входа на сайт и скачайте архив. Далее будет рассматриваться шаблон login\_v9. Из этого архива страницу index.html поместите в templates, всё остальное в static.



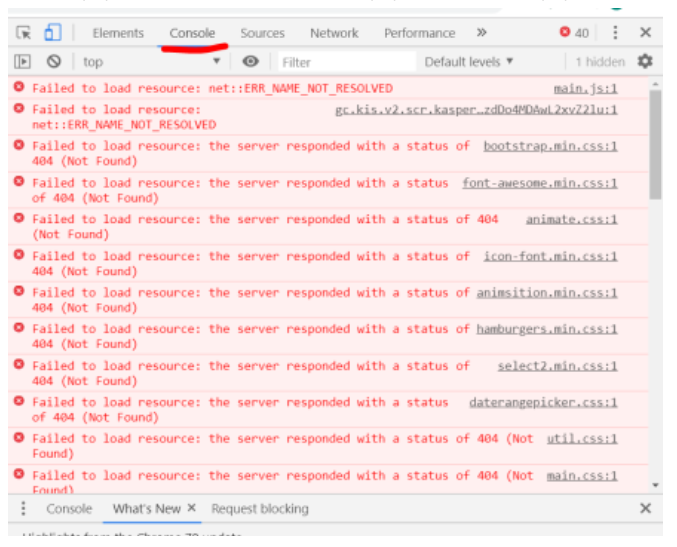
1. Создайте класс представление, который будет возвращать шаблон index.html приложения login
2. Создайте файл urls.py и добавьте туда маршрут login/, который будет использовать ранее созданное представление.
3. Подключите приложение login в настройках проекта.
4. Проверьте работоспособность страницы **login/.** Вы должны увидеть примерно такую картину



Связана она с тем, что браузер не смог подгрузить статические файлы. Убедиться в этом можно в консоли разработчика:



Здесь представлены все ошибки, которые возникли при загрузке страницы, а именно перечень файлов, которые не смогли загрузиться



1. Следующая задача отредактировать все ссылки в шаблоне index.html в пригодный формат для работы с Django. И не забудьте подключить тег **{% load static %}**

Исходная ссылка:

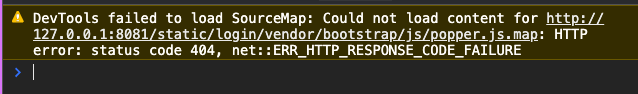


Конечный результат:

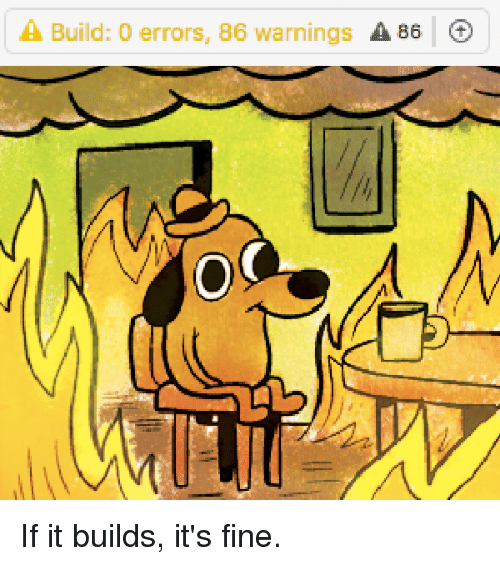


В качестве ключевых слов для поиска можно использовать src, href. Далее проверять в консоли разработчика всё ли удалось подгрузить.

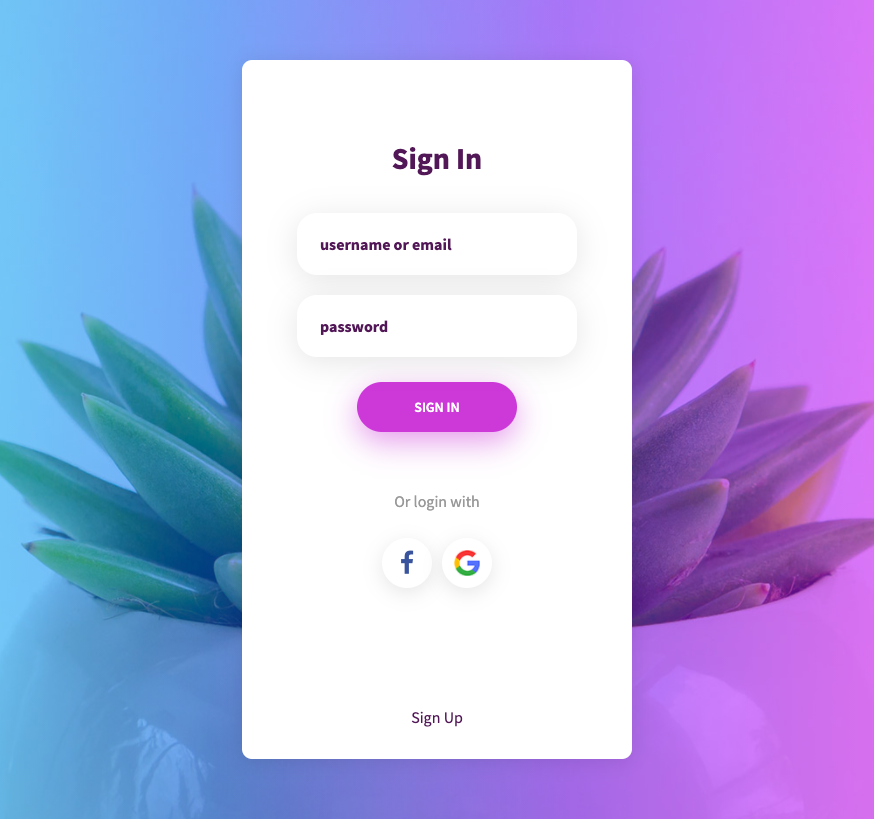
1. Конечный результат, чтобы консоль не содержала ошибок:



P.S. Warning не ошибка)))



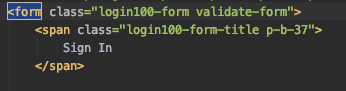
1. И в итоге вы увидите прекрасный результат своей работы:



На этом работу с GET запросам закончим, и перейдем к post запросам.

# Обработчик POST запросов

1. В шаблоне index.html найдите тег с началом (Форма начинается с <form)



и концом формы (форма завершается </form>



1. В тег **form** необходимо добавить два атрибута: action и method. Т.к. URL POST-запроса совпадает с URL GET-запроса, то для **action** необходимо поставить точку. А вот в качестве метода указать post



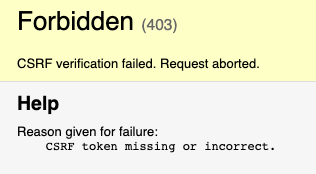
1. Теперь те данные, которые вводятся в поля логин и пароль будут переданы на сервер через POST запрос в его теле. Наша задача написать для этого обработчик.
2. Параметры POST запроса хранятся в request, который приходит от клиента в виде словаря. И получить их можно с помощью атрибута

| request.POST |
| --- |

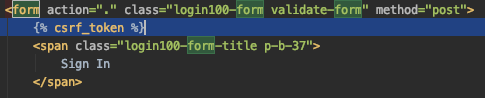
Полученные атрибуты будем отправлять обратно в виде JSON:



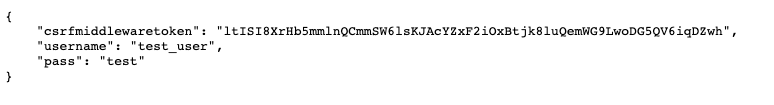
1. Пытаемся выполнить попытку залогиниться и получаем следующую ошибку:



1. Связанная она со встроенной в Django защитой от межсайтовых атак. Для того чтобы работать с форма могла отправлять POST запрос, необходимо подключить к ней тег **{% csrf\_token %}.**  Такой токен нужно подключать к каждой форме.



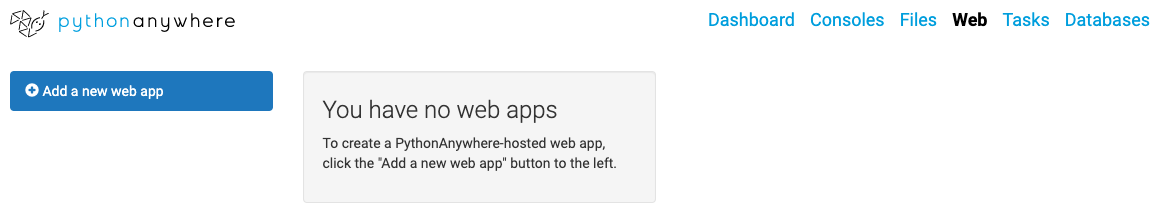
1. После добавления указанного токена, вновь выполняем попытку “залогинить“ и получаем следующий результат



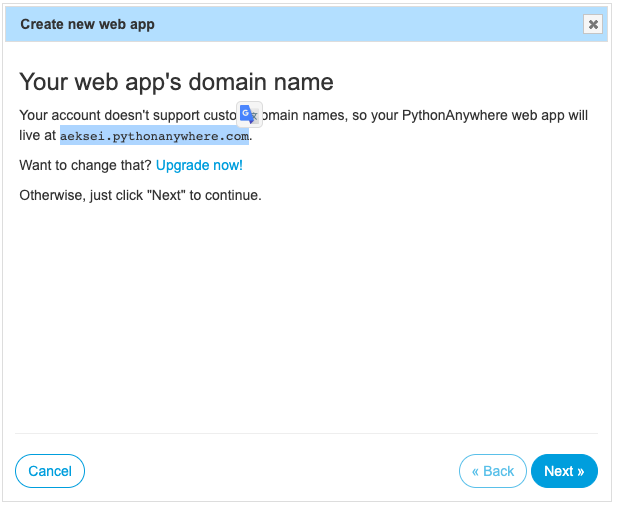
Те самые логин и пароль, которые вы указывали.

# Развертывание сайта

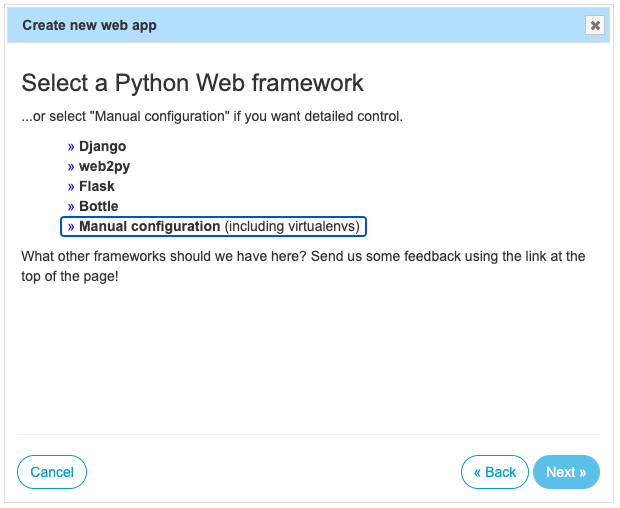
1. Зарегистрируйтесь на сайте <https://www.pythonanywhere.com/>
2. Создать Web приложение нажав “Add a new app”.



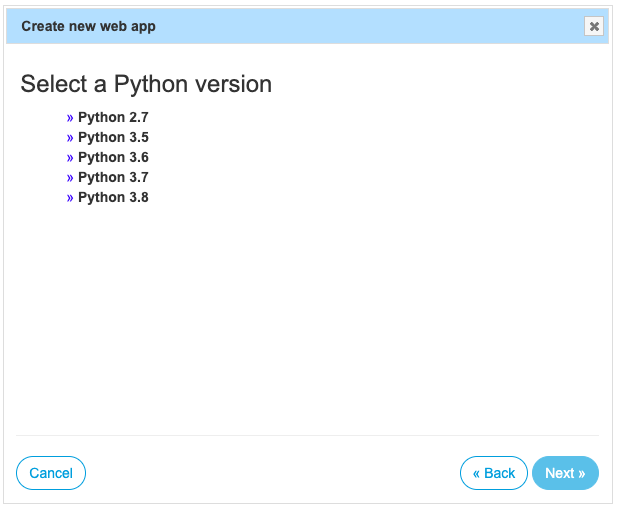
1. Ваш бесплатный домен, по которому вы можете получать доступ к вашему приложению. Сохраните название этого домена, он понадобится чуть позже. Нажимаем Next



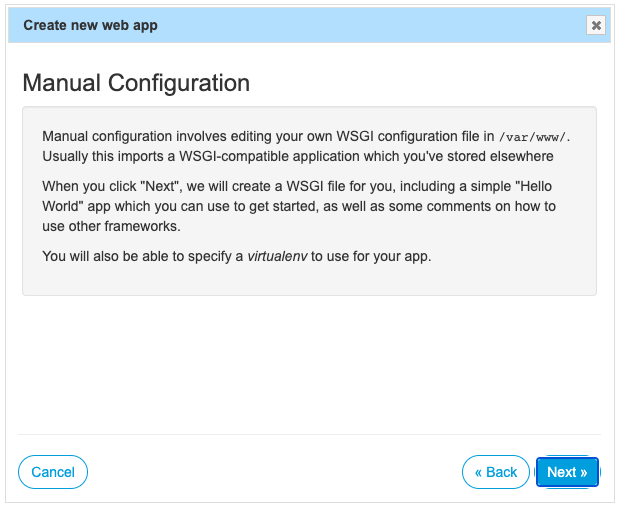
1. Выбираем “Manual configuration” (мы же не ищем легких путей)))).

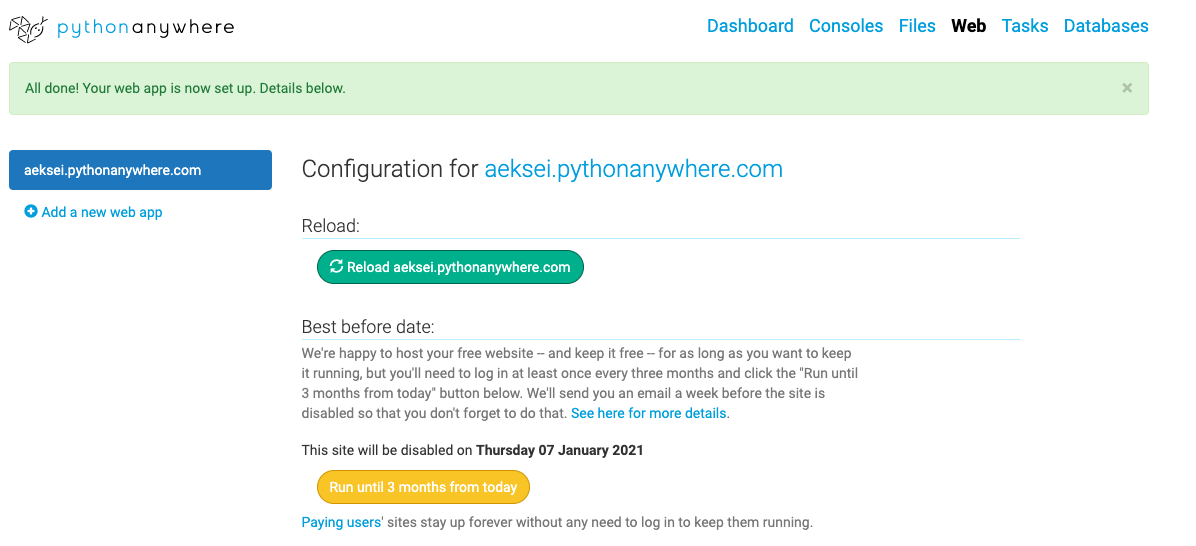


1. Чтобы избежать непредвиденных ситуаций при развертывании выберите ту версию интерпретатора, которая была у вас на локальной машине

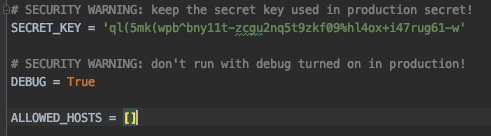


1. Нажимаем Next и ждем пока создастся приложение

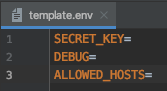




1. Теперь необходимо оформить git repo по вашему Django проекту, чтобы можно было облегчить развертывание результатов вашей локальной разработке на сервере.
2. Но прежде чем создавать локальный репозиторий необходимо выполнить несколько действий, которые позволят упростить и ваше развертывание.
3. Во-первых важно перед первым коммитом скрыть всю чувствительную информацию такую как, например, SECRET\_KEY, который хранится в файле settings.py. Хранить такую информацию будем в переменных окружения операционной системы.



1. Для начала создадим шаблон файла .env, где будем хранить переменные окружения. Этот файл не будет содержать самих значений, а только названия переменных. Этот файл можно коммитить

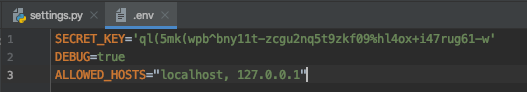


1. Далее делаем копию от файла template.env в файл .env, где будем хранить нужную информацию, которая не должна попадать под коммиты, такой файл необходимо настраивать на каждой машине, где будет развернут ваш Django проект. В нашем случае это локальный сервер и сервер хостинга.

| cp template.env .env |
| --- |

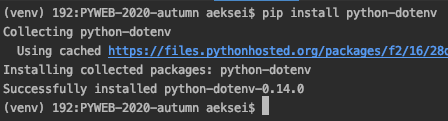


1. Переносим информацию из settings.py в файл .env

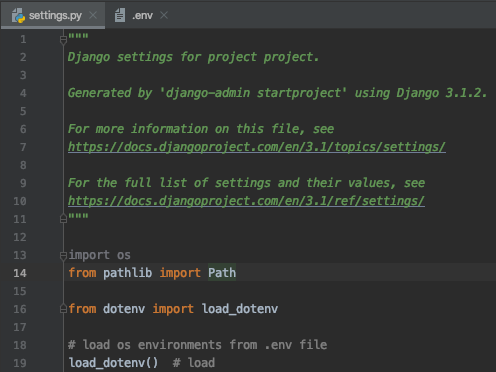


1. Устанавливаем пакет для работы удобной работы с переменными среды

| pip install python-dotenv |
| --- |

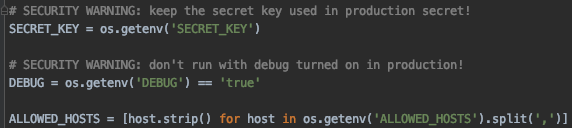


1. В файле settings.py выполняем загрузку переменных среды из файла .env с помощью установленного пакета:



1. И заменяем открытые значению в файле settings.py

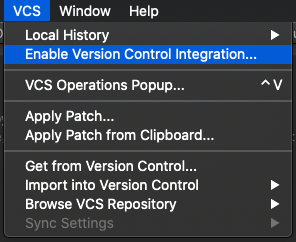
| SECRET\_KEY = os.getenv('SECRET\_KEY')  # SECURITY WARNING: don't run with debug turned on in production! DEBUG = os.getenv('DEBUG') == 'true'  ALLOWED\_HOSTS = [host.strip() for host in os.getenv('ALLOWED\_HOSTS').split(',')] |
| --- |



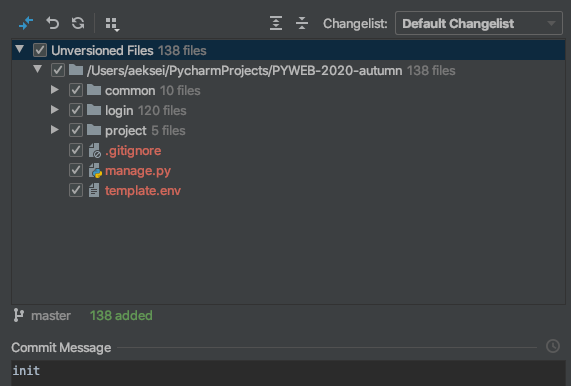
1. Во-вторых необходимо добавить файл .gitignore. Например его можно взять здесь <https://github.com/github/gitignore/blob/main/Python.gitignore>

Его нужно дополнить папкой .idea/, если работаете в PyCharm

1. Инициализируем репозиторий



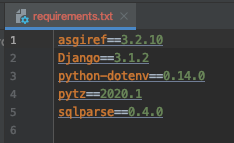
1. Перепроверяем, чтобы не попало ничего лишнего и комитим



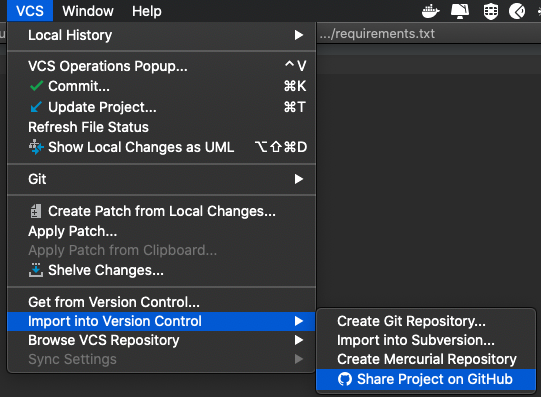
1. Третье и заключительного перед развертыванием на удаленном сервере это создать список зависимостей и сохранить его в файл requirements.txt

| pip freeze > requirements.txt |
| --- |

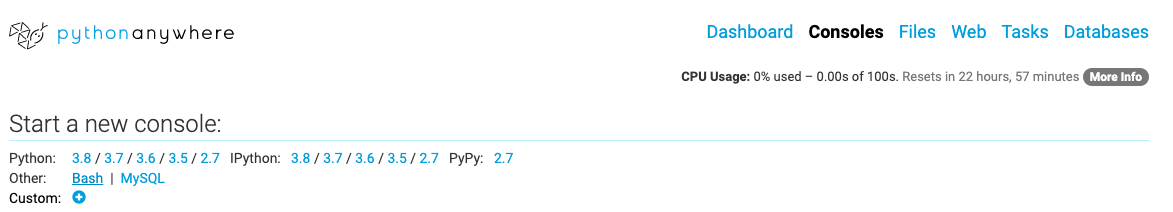




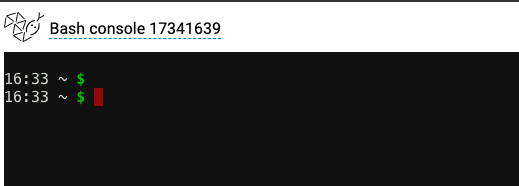
1. Коммитим его и отправляем всё на github c помощью команды “Share Project on Github”



1. В pythonanywhere выбираем Consoles и Bash

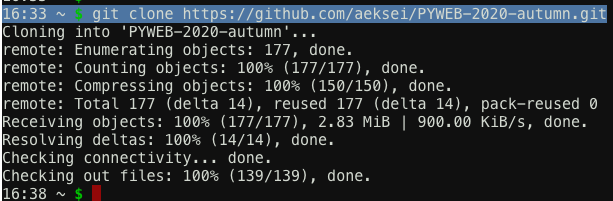


1. Перед вами откроется консоль, через которую вы сможете настроить удаленный сервер и загрузить туда ваш Django



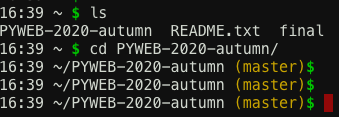
1. Далее необходимо склонировать ваш репозиторий, который был отправлен на github в пункте 20 с помощью команды:

| git clone <url репозитория> |
| --- |

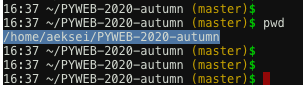


1. Далее перейти в папку (список все папок можно получить командой ls) проекта с помощью команды

| cd <название репозитория> |
| --- |



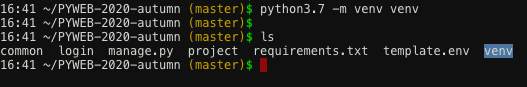
1. С помощью команды pwd выведите и сохраните путь до текущей директории, нам это понадобится для дальнейшей настройки веб приложения.



1. Далее нужно настроить виртуальное окружение, чтобы установить все зависимости, которые были установлены на вашей локальной машине. Создать виртуальное окружение можно с помощью команды

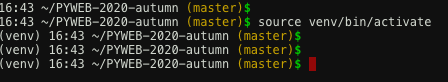
| python3.7 -m venv <название виртуального окружения> |
| --- |

После выполнения команды создается одноименная папка, которая содержит в себе виртуальное окружение



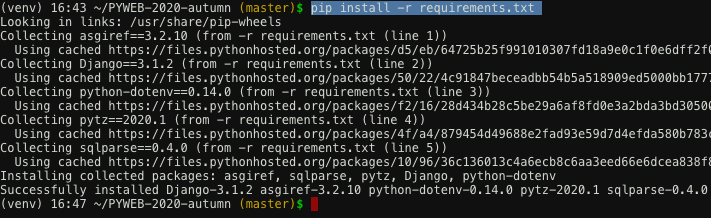
1. Теперь следует активировать виртуальное окружение с помощью следующей команды

| source <название виртуального окружения>/bin/activate |
| --- |



1. Установить все зависимости из файла requirements.txt

| pip install -r requirements.txt |
| --- |



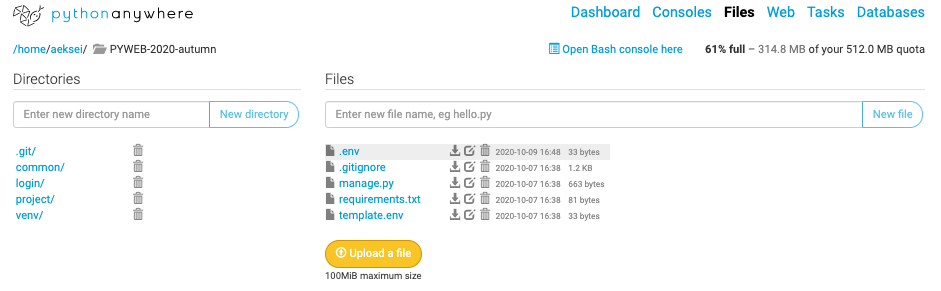
1. Создать файл .env и заполнить его необходимыми данными:

| cp template.env .env |
| --- |

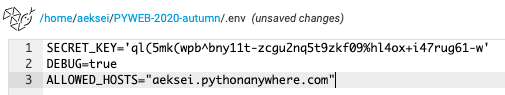


Любым удобным способом заполнить файл .env на удаленной машине (например, через графический интерфейс)

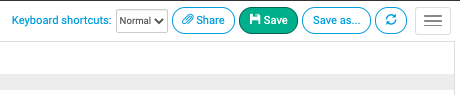
Files -> <папка с вашим проектом> -> .env



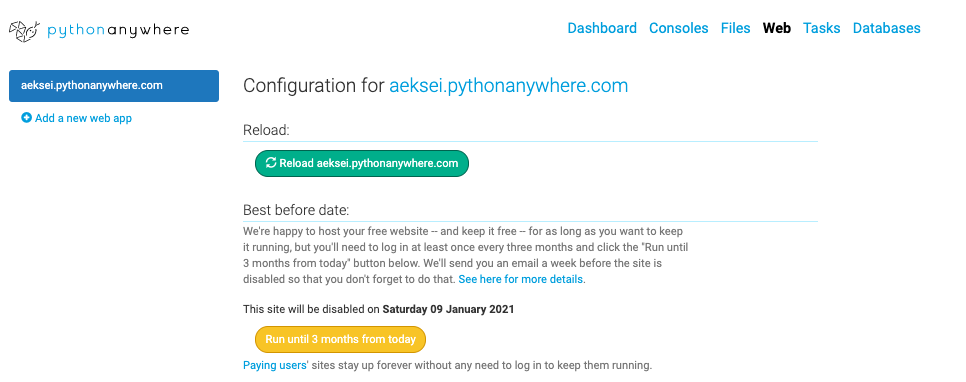
**SECRET\_KEY** как на локальной машине, **ALLOWED\_HOSTS** согласно вашему домену, полученному на 3 шаге



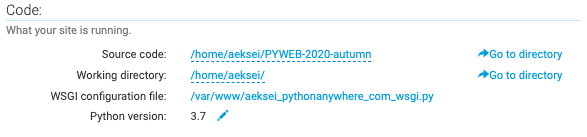
Зелененькая кнопочка, чтобы сохранить результат



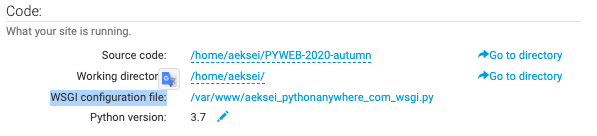
1. Подготовка среды закончена теперь, нужно настроить приложение. Переходим во вкладку Web



1. Находим блок “Code”, настраиваем папку с исходным кодом вашего проекта (Его ма получили на 25 шаге)



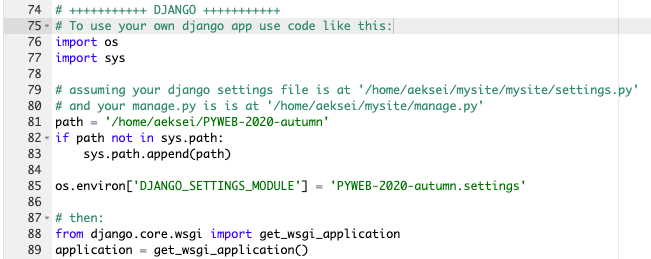
1. Далее нужно настроить “WSGI configuration file”:



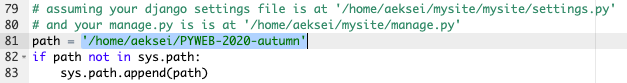
1. В файле /var/www/aeksei\_pythonanywhere\_com\_wsgi.py комментируем 19-47 строки



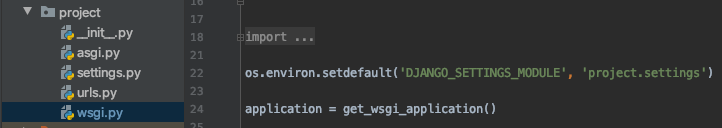
1. Раскомментируем 76-89 строки



1. Далее нужно настроить путь до файлов с настройками вашего приложения (файл settings.py) и файла, который управляет вашим приложением (manage.py). Переменная path должна быть равна пути до вашего проекта (см 25 пункт)



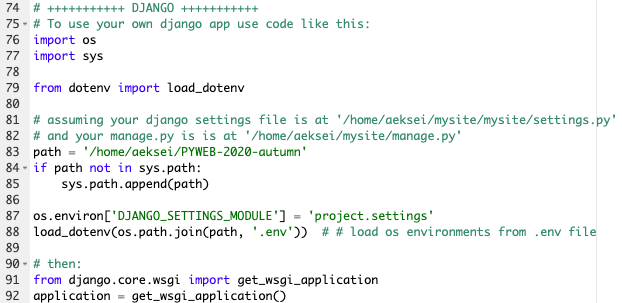
1. Переменная окружения на ваш модуль с настройками проекта должна быть изменена в соответствии с названием вашего проекта. Подсмотреть её значение можно в файле wsgi.py вашего Django проекта





1. Также нужно подгрузить переменные среды, находящиеся в файле .env (79 и 88 строка)

| from dotenv import load\_dotenv  load\_dotenv(os.path.join(path, '.env')) # # load os environments from .env file |
| --- |

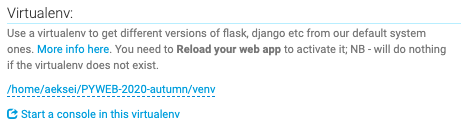


1. Сохраняем файл. Кнопка Save

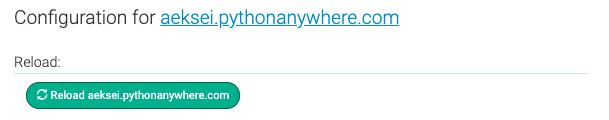


1. Настраиваем путь до вашего виртуального окружения во вкладке Web

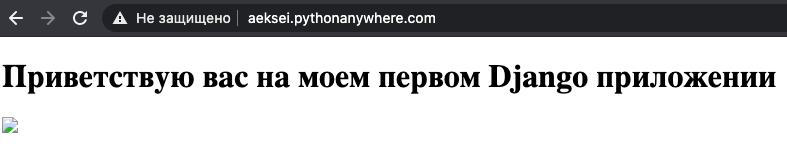
<путь до проекта>/<названием виртуального окружения> (пункт 25 и 26 соответственно)



1. Перезапускаем приложение (зеленая кнопка) и переходим по ссылке выше (доменное имя)

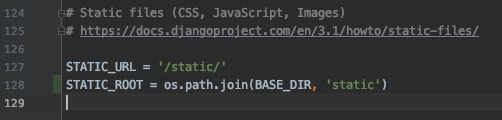


1. В итоге вы должны увидеть рабочее приложение, но опять без статических файлов (((



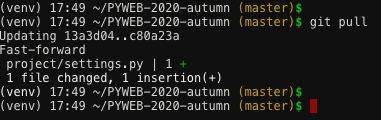
1. Нужно настроить путь до статических файлов для web сервера pythonanywhere. Для этого в настройках проекта нужно добавить переменную **STATIC\_ROOT**.

| STATIC\_ROOT = os.path.join(BASE\_DIR, 'static') |
| --- |



1. Коммитим и пушим изменения на сервер github
2. Открываем консоль pythonanywhere, идем в папку проекта, и обновляем исходный код с помощью команды

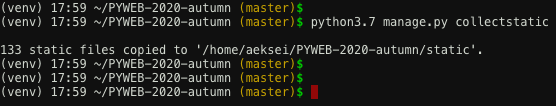
| git pull |
| --- |



1. Запускаем команду

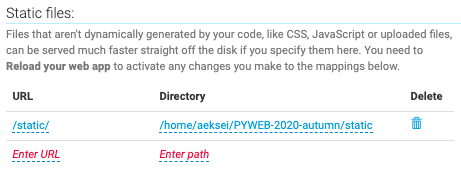
| python3.7 manage.py collectstatic |
| --- |

Чтобы собрать все статические файлы из приложений в одном месте



1. Во вкладке Web настраиваем раздел “Static files”:

* **URL** = **STATIC\_URL** из settings.py (см. пункт 42)
* **Directory** = **<путь до проекта>/<название папки со статическими файлами>** (см. пункт 25 и 42)



1. Перезапускаем приложение и наслаждаемся результатом проделанной работы!!!

