

Virtual Environments and Django

Creating a Python Virtualenv in Powershell

There are three methods that can be used to create a virtual environment in powershell:

- venv
- virtualenv
- pipenv
 - This one only applies if you're using python

When inside one of these virtual environments we can install packages that will remain local to the environment that we are in rather than have them be available globally within our machine. This can be useful for avoiding things like naming conflicts between packages and it avoids unnecessary clutter when working in a project because the project will only have access to the packages installed within its virtual environment.

Creating a virtual environment using venv and virtualenv

To create a virtual environments using these methods you first navigate to the directory that you want to have a virtual environment. Once inside the directory, if you use venv run the command **python -m venv enviroName** and, if using virtualenv, use the command **virtualenv envName**. Once the virtual environment has been created we can write **.\Scripts\activate** to activate the environment. While active, to deactivate the environment we can write **deactivate** or **exit**.

Virtual environments using pipenv

When using pipenv to create our virtual environment we can run the **pipenv shell** command which will both create and activate the virtual environment for us. Once active, to leave a pipenv environment we use **exit**.

For further detail about creating python virtual environments visit this page:

<https://www.youtube.com/playlist?list=PLEsfXFp6DpzQ33Q5xhD5naEV9K-Z6upZw>

Creating a Django Project

After setting up and activating the virtual environment the next step is to install Django; to do this, use the command ***pip install Django***. When running this command, if you want to install a specific version of Django the command would look like this ***pip install Django==version number***.

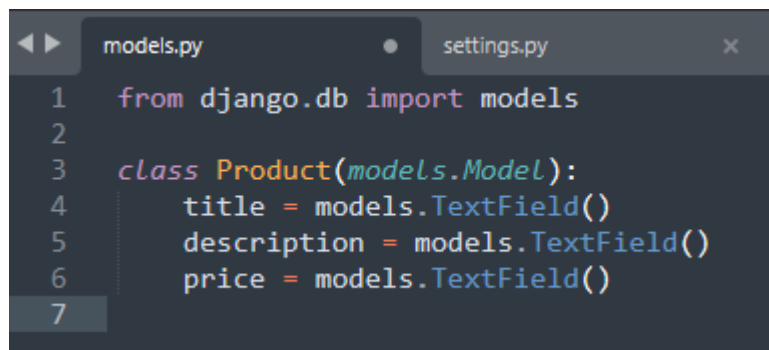
Creating Project directory

Once the venv has been activated and Django has been installed, use ***Django-admin startproject fileName*** .. Move into the newly created project directory and you should see a manage.py file; this file allows a server to run when the command ***python manage.py runserver*** is used. When the server starts a web address will be displayed which be entered into a web browser to access the web page, if the landing page is shown then the server is running correctly. Also, when running the server a port number as well as an IP address can be specified for the server by using ***python manage.py runserver (IP) (port number)***. When the project is created go ahead and open it in whatever text editor you like.

Creating an App

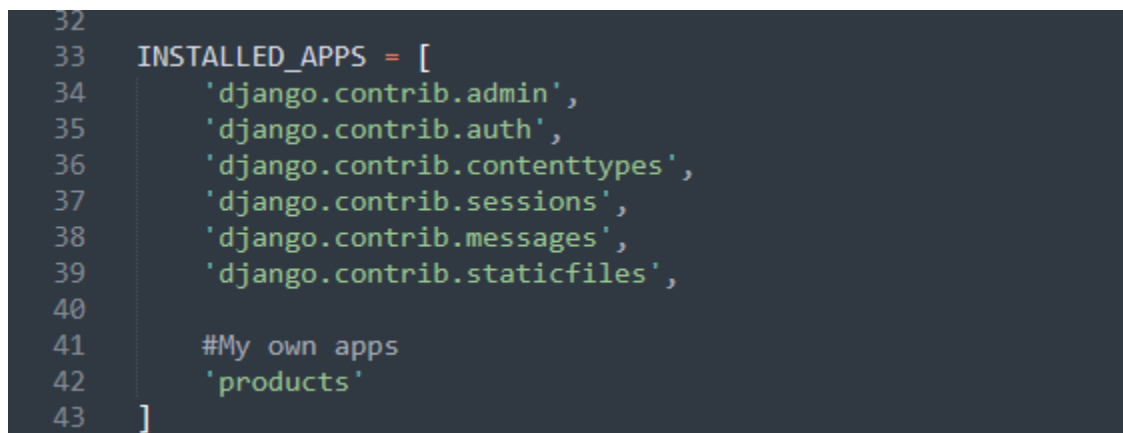
An app in django isn't like an app that would be on a phone or a computer, instead a Django app works more like a module than a fully fleshed out application. Basically, they operate specific areas within a project rather than being the entire project itself. As an example, if the goal were to build a some website where we have a user profile, a cart, and a product page each of these three things would be their own app in Django but each are just one piece of the entire project. For better detail this <https://youtu.be/XviOaYU-uQo?t=70> is a good reference since it goes into further detail about how to actually think about an app and what their focus should be.

Now, to actually create our own app for our project we use **python** *manage.py startapp appName* when inside the directory of our project (the directory that contains *manage.py*). Next, go into the newly created app and open the *models.py*. Inside *models.py* we create a Product class and give it a few items.



```
1 from django.db import models
2
3 class Product(models.Model):
4     title = models.TextField()
5     description = models.TextField()
6     price = models.TextField()
7
```

After this step, we open the *settings.py* file of our project inside a text editor and scroll down to the `INSTALLED_APPS` section. Here, we will add the app we just created to the list.



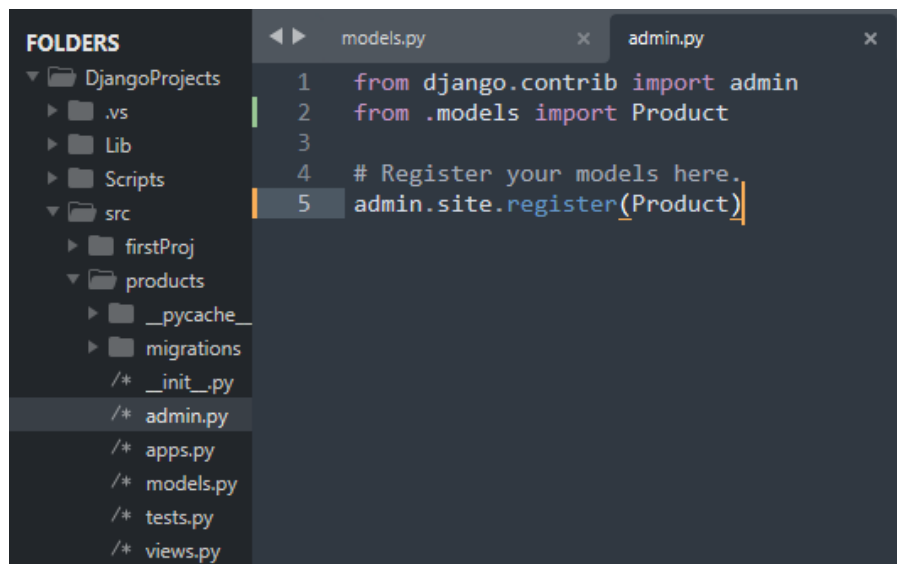
```
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40
41     #My own apps
42     'products'
43 ]
```

Now we need to update our database to include the changes that have been to the *setting.py* file. To update it we need to use powershell, or whatever terminal is available for the OS being used, and, inside the virtual environment, run the command **python** *manage.py makemigrations* and then **python** *manage.py migrate*.

While the *makemigrations* and *migrate* commands may seem similar they do serve different purposes. From the official Django documentation **makemigrations** is used for packaging model changes into individual migration files and **migrate** applies those changes to the database. Here is a link to the Django doc to refer back to if you want more info on this.

<https://docs.djangoproject.com/en/3.2/topics/migrations/>

Since we've created a product app and migrated the changes onto the database we can see the results of what's been so far if after a few more steps; the first of which is to add the products model into the *admin.py* file and register it.



```
FOLDERS
  DjangoProjects
    .vs
    Lib
    Scripts
    src
      firstProj
      products
        __pycache__
        migrations
        /* __init__.py
        /* admin.py
        /* apps.py
        /* models.py
        /* tests.py
        /* views.py

models.py
admin.py
1 from django.contrib import admin
2 from .models import Product
3
4 # Register your models here.
5 admin.site.register(Product)
```

In the above image we opened the *admin.py* and added lines 2 & 5. The *.models* import is what's known as a relative import which just means that something is being imported from a file that is in the same directory as *admin.py*. We can see on the left of the image that both files (*models.py* and *admin.py*) are in the same directory so the relative import works. There is more to relative imports which can be read here (about [relative imports](#)) but it isn't relevant in this case. W

Setting up an Admin Account

Even though we've added Products to our admin file we still can't see what's been done just because we need to create administrator account first. So, to do this, we access our virtual environment in powershell, moving to the director that contains *manage.py* and running **python** *manage.py* *createsuperuser*. In my case I created the admin with Username: 'aron', Email: 'left empty', and Password: 'aronpass'.

After creating the superuser we need to run the server and then go to this address <http://127.0.0.1:8000/admin/> to reach the login page. Here, enter the information that was used to create a superuser and we'll be taken to an admin

home page.

Django administration

WELCOME, **ARON** / [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups

[+ Add](#) [Change](#)

Users

[+ Add](#) [Change](#)

PRODUCTS

Products

[+ Add](#) [Change](#)

Recent actions

My actions

None available

This is where can see the first results of the work we've done so far which, in this case, is adding products onto our admin home page. Clicking the "Add" button for a product directs us to a page where enter information into the fields that we created earlier; so we give a product a title, description, price, and summary.

Errors Encountered and Solutions

- 1) When trying to activate a virtual environment using `.\Scripts\activate` there was an error stating that it wouldn't work because the files were not digitally signed. To fix it, we just need to run powershell as an admin and then run command `Set-ExecutionPolicy Unrestricted`.
- 2) Trying to add a new product as a superuser gives this as an error.

```
OperationalError at /admin/products/product/add/  
no such table: main.auth_user__old  
  
Request Method: POST  
Request URL: http://127.0.0.1:8000/admin/products/product/add/  
Django Version: 2.0.7  
Exception Type: OperationalError  
Exception Value: no such table: main.auth_user__old  
Exception Location: E:\Programming\Projects\DjangoProjects\lib\site-packages\django\db\backends\sqlite3\base.py in execute, line 303  
Python Executable: E:\Programming\Projects\DjangoProjects\Scripts\python.exe  
Python Version: 3.9.5  
Python Path: ['E:\\Programming\\Projects\\DjangoProjects\\src',  
             'c:\\users\\aron\\appdata\\local\\programs\\python\\python39\\python39.zip',  
             'c:\\users\\aron\\appdata\\local\\programs\\python\\python39\\DLLs',  
             'c:\\users\\aron\\appdata\\local\\programs\\python\\python39\\lib',  
             'c:\\users\\aron\\appdata\\local\\programs\\python\\python39',  
             'E:\\Programming\\Projects\\DjangoProjects',  
             'E:\\Programming\\Projects\\DjangoProjects\\lib\\site-packages']  
Server time: Mon, 7 Jun 2021 13:54:24 +0000
```

Not sure what's causing this error just yet but will update with a fix when one is found.

Update: The error seems to come from sqlite db not being updated properly seemingly due to differences because sqlite was operating on a more recent version while I was using an older version of Django. Ran `pip install Django --upgrade` to update to the most recent version of Django and then did `python manage.py migrate` to update the database. Afterward I was able to successfully add a product without error.