**Few Important commands:**

* Inside the parent directory of your Desktop directory. ‘Dev’ directory is created. Go to

cd Dev/trydjango2

source bin/activate //this activates the virtual environment

cd src

* After creating app don’t forget to mention it in the INSTALLED APPS section of trydjango2/src/trydjango2/settings.py
* After updating anything in the models.py inside Product\_app Run this:

1. python manage.py makemigrations
2. python manage.py migrate

* To host this page on the server go to the pwd as above inside the activated virtual environment just run(everything should be same as this terminal itself, just another window is required):

python manage.py runserver

This command outputs an http server link, copy paste it in the browser. After adding the superuser put /admin at the end of the link of the website.

* Now to create a superuser, run: python manage.py createsuperuser and enter superuser name and a strong password too.
* To enter data in the apps from the python shell run: python manage.py shell

Inside this shell, just import your class of the app like this:

from Product\_app.models import Product

Here *Product\_app* is the appname and *models* is referring to the *Product\_app/models.py* where the class *Product* is declared.

* Now, to enter the tuples of Product\_app do: it is like sql.

1. Product.objects.create(title='hitest',Description='no desc', Price=56.89, Stars=3.5, Buyers=12335, Company\_Assured=True)
2. To print all the objects or tuples created till now: Product.objects.all()

Now, to see the changes just go to the server you ran in the other terminal.

* When you change the fields in the class of a given app, you need to delete the *sqlite database* located in this *src* folder. You also need to delete the pycache folder present in *Product\_app* folder. You also need to empty the *migrations* folder except the *\_\_init\_\_.py* no .py file should be present in there. Now you are freshly ready to experiment with new changes you’ve made. This is part of the practice, learn it and use it.

**Few Important Points:**

1. Blank = False means saying you can’t keep this field blank or “This field is required inside the website i.e. how is this field rendered”.
2. Blank = True means the field may or may not be filled it is upto the data entry person.
3. Null = True means the new field that you have created right now doesn’t affect the previously stored tuples in the database. That means the previously stored tuples which have no idea of this tuple can be kept simply empty.
4. Null = False means the above thing cannot be done and you have to declare a default thing in your new field.

**Views:**

Think of views as if it handles your webpages, it is an integral part of each and every app. So views contains containers and each one of them is just a python Function or a python class.

So, at first we go to the views folder of any app lets say *‘pages’* and create our ‘home\_view’ function’ which in turn returns a ‘HttpResponse’ in response to the futuristic ‘HttpRequest’ made by your client. To make your ‘home\_view’ function fully functional, consider reading abourt *\*args* and *\*\*kwargs* in Python: <https://www.geeksforgeeks.org/args-kwargs-python/>

So the link to your specified function or class based view is specified within ‘urls.py’ contained within trydjango2/src/trydjango2. ‘urls.py’ is unique to a given ‘trydjango2’ directory.

The import methods of class-based views and function based views are a little different.

Examples:

*Function* views

1. Add an import: from my\_app import views

2. Add a URL to urlpatterns: path('', views.home, name='home')

*Class-based* views

1. Add an import: from other\_app.views import Home

2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home')

*Including another URLconf*

1. Import the include() function: from django.urls import include, path

2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

For more information views, visit <https://docs.djangoproject.com/en/3.0/topics/http/urls/>.

**Lets talk about Forms in Python:**

* action = ‘some/url/path/’ sends the form to whatever url you put in it.

<form action = ‘http://www.google.com/search’ method = ‘GET’></form> and change the 'name' attribute of input to 'q'.

* Putting something like above sends the user’s data to the Google search engine to be searched and as you have guessed it well, this input data need not be saved at the backend therefore the method shall be “GET”. GET request changes the url itself. Observe this.

**CSRF Token:**

* Inside forms, if you use GET request, then no CSRF error is thrown because no data is getting save in the database at the backend.
* Therefore, usage of POST method is suggested, and then adding CSRF token just after that does the job.
* CSRF token is basically a security measure thankfully provided by Django, which should be made sure before posting anything into the Database.

How to Deploy your Django website?

* Goto your src folder and in all of its \_\_init\_\_.py files write a comment, #comment. Otherwise it will cause problem while uploading on github.
* Take the ‘src’ folder and copy all of its stuff into a github repository(say ‘my\_new\_blog). Now your repository looks like this:

my\_new\_blog,

|`---Blog(my\_app),

|`---migrations

|`---static

|`---templates

|`---\_\_init\_\_.py

|`---admin.py

|`---views.py

|`---forms.py

|`---models.py

|`---tests.py

`---urls.py

|`---templates

|`---trydjango2(defaultly generated folder),

|`---\_\_init.py

|`---settings.py

|`---urls.py

`---wsgi.py

|`---manage.py

`---db.sqlite3

Copy the link of this repositery.(say <link>)

* goto <https://www.pythonanywhere.com/user/animeshk/> and open the bash console
* check your pwd
* Inside the github repository, make a new folder called ‘media’, inside media paste whatever it was within your static\_cdn-→media\_root. This is because we are not having static\_cdn on github, therefore in order to server those media files we have found a different way, and this process requires us to change the settings.py file a little bit, which we shall explain in further step-by-step way.
* now do ‘git clone <link>’ hit Enter.
* This will clone the whole my\_new\_blog folder on the server
* run’ mkvirtualenv –python=/usr/bin/python3.7 myenv Now this creates a virtual environment named ‘myenv’ within bash terminal and automatically activates it too.
* Now ‘cd’ into the folder you just cloned.
* Assuming that you have already created requirements.txt file, just go to the bash terminal and run pip install -r requirements.txt and it’ll install all the required packages you specified in requirements.txt
* Now open another window of this pythonanywhere website and go to the *Files* section and make sure git-cloned files are there.
* Now go to the *Web* section of the website
* Add a new web app
* Next
* Manual Configuration
* python3.7
* Next
* A dummy website is deployed, you can check on http://<username>.pythonanywhere.com
* Inside the *Web* section, go to the *“Virtualenv:”* section. Write your virtual environment name *myenv* in the form.
* Inside the Code section, go to the *WSGI Configuration file* section and click on the link to the file.
* Delete all the comments written below *Flask* heading. You can retain only the comments within the Django heading, before the Flask heading.
* Uncomment all the comments within the Django heading, only which are related to code part, not the English explanation.
* Update path = '/home/<your username on pythonanywhere>/<name of git-cloned-folder or repository you cloned>'
* Update os.environ['DJANGO\_SETTINGS\_MODULE'] = '<your main app component name which contains settings.py>.settings'
* Now you’ll see somethings like this:
* import os
* import sys
* # assuming your django settings file is at '/home/netcartecommerce/mysite/mysite/settings.py'
* # and your manage.py is is at '/home/netcartecommerce/mysite/manage.py'
* path = '/home/netcartecommerce/My\_Ecommerce\_website'
* if path not in sys.path:
* sys.path.append(path)
* os.environ['DJANGO\_SETTINGS\_MODULE'] = 'ecommerce.settings'
* # then:
* from django.core.wsgi import get\_wsgi\_application
* application = get\_wsgi\_application()
* Save it and Close this file, we are done with it,
* Go into the *Files* section and open up your settings.py and Update ALLOWED HOST = [‘<your usename on pythonanywhere>.pythonanywhere.com’] with no slashes.
* Remember here you do not have your static\_cdn folder which used to server media and static files within your local PC, so we have to serve the media and static files in a different way. This was the day for which we had prepared our ‘media’ folder parallel to where db.sqlite3, manage.py or static\_my\_project reside.
* Comment the STATIC\_ROOT = os.path.join(os.path.dirname(BASE\_DIR), 'static\_cdn', 'static\_root') #will be used in PC line
* Add the STATIC\_ROOT = '/home/<your username on python anywehre>/<name of git-cloned-folder or repository you cloned>/static' #for deployment on pythonanywhere line
* Comment the MEDIA\_ROOT = os.path.join(os.path.dirname(BASE\_DIR), 'static\_cdn', 'media\_root') #will be used in PC # so uploaded files go to MEDIA\_ROOT, Remember these are not

# hard-coded files, these will be uploaded by may be 'enduser' line

* Add the MEDIA\_ROOT = '/home/<your username on python anywehre>/<name of git-cloned-folder or repository you cloned>/media' #for deployment on pythonanywhere line
* Goto the bash terminal once again, make sure you are within the folder you git-cloned
* Run python manage.py collectstatic, this creates a new folder called *static* and copies all of your static files into it.
* Now, make sure the static directory is created with all the static files into it.
* Inside the *Web* section, scroll down and go to the *Static files:* section.
* Do this

| URL | Directory | Delete |
| --- | --- | --- |
| [/static/](https://www.pythonanywhere.com/user/animeshkumar/webapps/) | [/home/<your username on python anywehre>/<name of git-cloned-folder or repository you cloned>/static](https://www.pythonanywhere.com/user/animeshkumar/webapps/) |  |
| [/media/](https://www.pythonanywhere.com/user/animeshkumar/webapps/) | [/home/<your username on python anywehre>/<name of git-cloned-folder or repository you cloned>/media](https://www.pythonanywhere.com/user/animeshkumar/webapps/) |  |
| [*Enter URL*](https://www.pythonanywhere.com/user/animeshkumar/webapps/) | [*Enter path*](https://www.pythonanywhere.com/user/animeshkumar/webapps/) |  |

For example:

| URL | Directory | Delete |
| --- | --- | --- |
| [/static/](https://www.pythonanywhere.com/user/animeshkumar/webapps/) | [/home/animeshkumar/My\_Ecommerce\_website/static](https://www.pythonanywhere.com/user/animeshkumar/webapps/) |  |
| [/media/](https://www.pythonanywhere.com/user/animeshkumar/webapps/) | [/home/animeshkumar/My\_Ecommerce\_website/media](https://www.pythonanywhere.com/user/animeshkumar/webapps/) |  |
| [*Enter URL*](https://www.pythonanywhere.com/user/animeshkumar/webapps/) | [*Enter path*](https://www.pythonanywhere.com/user/animeshkumar/webapps/) |  |

**Its is still incomplete.**

**Below is the way we interact with the python shell to exercise the privileges of RDBMS, foreign key and stuff:**

*In this example every ‘Tag’ object is having many ‘Product’ objects. Both ‘Tag’ and ‘Product’ are models in ‘models.py’ folder of respective app folders.*

(ecommerce) ┌─[AnimeshK@kali]─[~/Dev/ecommerce/src]

└──╼ $python manage.py shell

Python 3.7.3rc1 (default, Mar 13 2019, 11:01:15)

[GCC 8.3.0] on linux

Type "help", "copyright", "credits" or "license" for more information.

(InteractiveConsole)

>>> from tags.models import Tag

>>> Tag.objects.all()

<QuerySet [<Tag: tshirt>, <Tag: bag>, <Tag: code>]>

>>> tagobject = Tag.objects.get(id=2)

>>> tagobject

<Tag: bag>

>>> tagobject.title

'bag'

>>> tagobject.slug

'bag'

>>> tagobject.products #now this is like model manager “.objects()” means something like .get(), .all() can be done.

<django.db.models.fields.related\_descriptors.create\_forward\_many\_to\_many\_manager.<locals>.ManyRelatedManager object at 0x7f5ba6ec1e80>

>>> tagobject.products.all()

<ProductQuerySet [<Product: School Bag>, <Product: Airbag>]>

>>> tagobject.products.last()

<Product: Airbag>

>>> exit()

(ecommerce) ┌─[AnimeshK@kali]─[~/Dev/ecommerce/src]

└──╼ $python manage.py shell

Python 3.7.3rc1 (default, Mar 13 2019, 11:01:15)

[GCC 8.3.0] on linux

Type "help", "copyright", "credits" or "license" for more information.

(InteractiveConsole)

>>> from products.models import Product

>>> productobj = Product.objects.first()

>>> productobj

<Product: T-shirt>

>>> productobj.title

'T-shirt'

>>> productobj.description

'Great T-shirt . My fav T-shirt'

>>> productobj.tag\_set #now this is like model manager “.objects()” means something like .get(), .all() can be done.

<django.db.models.fields.related\_descriptors.create\_forward\_many\_to\_many\_manager.<locals>.ManyRelatedManager object at 0x7f7ad527c630>

>>> productobj.tag\_set.all()

<QuerySet [<Tag: tshirt>]>

>>> productobj.tag\_set.first()

<Tag: tshirt>

>>> tagobj = productobj.tag\_set.first() #Now this is awesome

>>> tagobj

<Tag: tshirt>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Courtesy:*** *freecodecamp channel’s tutorial on YouTUBE and Django official documentation website. This PDF is fail proof on LINUX systems. On windows these commands may or may not work.*